UNIVERSIDADE FEDERAL DE MINAS GERAIS
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA NUCLEAR
Programa de Pós-Graduação em Ciência e Técnicas Nucleares

CONTROLE E AUTOMAÇÃO DAS MEDIDAS DE UM ELETRÔMETRO ACOPLADO A UM DETECTOR DE RADIAÇÃO, USANDO MICROCONTROLADOR PIC E LINGUAGEM ASSEMBLY.

Abner Pinto da Fonseca

Belo Horizonte

2013

Abner Pinto da Fonseca

CONTROLE E AUTOMAÇÃO DAS MEDIDAS DE UM ELETRÔMETRO ACOPLADO A UM DETECTOR DE RADIAÇÃO, USANDO MICROCONTROLADOR PIC E LINGUAGEM ASSEMBLY.

Tese apresentada ao Programa de Pós-Graduação em Ciências e Técnicas Nucleares da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito parcial à obtenção do Título de Doutor em Ciências e Técnicas Nucleares.

Área: Ciências das Radiações

Orientador: Prof. Arno Heeren de Oliveira
Departamento de Engenharia Nuclear

Belo Horizonte
2013

Fonseca, Abner Pinto da.
F676c Controle e automação das medidas de um eletrômetro acoplado a um detector de radiação, usando microcontrolador Pic e linguagem Assembler [manuscrito] / Abner Pinto da Fonseca. – 2013.
xxi, 293 f., enc.: il.

Orientador: Arno Heeren de Oliveira.

Tese (doutorado) - Universidade Federal de Minas Gerais, Escola de Engenharia.

Anexos: 120-293.
Bibliografia: f.116-119.

1. Engenharia nuclear – Teses. 2. Assembler (linguagem de programação de computador) – Teses. 3. Calibração – Teses. I. Oliveira, Arno Heeren de. II. Universidade Federal de Minas Gerais, Escola de Engenharia III. Título.

CDU: 621.039(043)

DEDICATÓRIA

Dedico esse trabalho à minha querida mãe Conceição (*in memoriam*) dado que essa tese é (foi) a real chance de que seu filho consiga (conseguisse) o título de doutor, o que a deixaria, em parte, sem constrangimento. Eu explico: o fato aconteceu em um churrasco que era de práxis comemorar o aniversário do Sr. Antônio Bié (*in memoriam*), um grande amigo da família, em sua fazenda no município de Bom Despacho, MG. Em um desses churrascos, minha querida mãe encontrou uma conhecida que há anos não se viam, a qual externou, em algum momento, sobre os filhos doutores que tinha. Minha mãe, constrangida, meio trêmula a voz, respondeu querendo não se mostrar menor, mas com clara postura duvidosa, em mesmo nível: "os meus também são doutores, todos". Minha mãe tinha ciência que os filhos eram graduados e não doutores. Mas, agora mamãe querida, onde quer que esteja, dedico a você esta possibilidade real ao Titulo de Doutor (agora, pós defesa, real mesmo), a você e ao meu pai, José Fonseca (*in memoriam*), que tanto lutaram para que todos os seus filhos (*) fossem as pessoas honradas que se tornaram. Obrigado, queridos. Eu sempre os amei, e os amo ainda que ausentes. Abner.

(*) Dedico esse trabalho a todos os meus queridos irmãos: Antônio, Maria Eunice, Arley, Acir, Aécio, Aures, Ana Maria e Érika. Que todos continuem propagando os bons e belos ensinamentos deixados por nossos pais. Num primeiro momento, propagando-os aos seus filhos (próxima geração) e num segundo, irradiando-os por toda a sociedade.

Dedico esse trabalho à minha querida esposa Shirley Martins da Costa Fonseca. Ela foi sempre a mulher que me amparou na solvência dos nossos muitos problemas, permitindo o retorno ao equilíbrio, à paz.

E por fim, mas não o último em importância, dedico esse trabalho ao meu querido filho Bruno. Este que de garoto ao homem que hoje é, só me dera motivos para a boa e ordeira (equilibrada) luta social em defesa da família.

AGRADECIMENTOS

Agradeço ao Prof. Arno por toda dedicação objetiva posta a orientar-me, mostrando o norte, que possibilitou a superação dos obstáculos rumo a materialização desse trabalho.

Ao meu filho Bruno que ajudou com suas criticas e realizando alguns desenhos e fotos.

A minha esposa Shirley por toda vida conjunta de enfrentamento das dificuldades que surgiam.

Muito obrigado a todos vocês.

EXTERNALIDADE

Se é guardada não ofende,
pode corroer, depende.
Se externar alivia,
pode machucar, também depende.
Se há leveza,
pode ensinar, pretende.
Se há dureza,
pode matar, certeza.

Abner P. Fonseca
15/02/2001 às 11:05h

RESUMO

Este trabalho teve como objetivo principal a automação das medidas de carga elétrica gerada no interior de um detector de radiação, ou seja, foi desenvolvido um dispositivo automatizado para aquisição de dados. Essas medidas são realizadas por um eletrômetro e permitirão a determinação da dose e/ou taxa de dose devida a uma fonte radioativa a diversas distâncias fonte-detector. Também permitirão a calibração de detectores com maior precisão, a simulação de uma fonte infinita e a verificação de homogeneidade das fontes.

O referido dispositivo automatizado é suportado por uma estrutura mecânica composta por duas mesas que se deslocam ortogonalmente e são acionadas, cada qual, por um motor de passo. O sistema possui duas direções de deslocamento, uma chamada de X (eixo cartesiano X), com liberdade de movimento de -20,0 cm a 20,0 cm e passos de 1,0 cm, enquanto a outra direção chamada de Y (eixo Y), com movimento de 2,0 a 92,0 cm e, também, passos de 1,0 cm. Para cada direção construiu-se uma mesa que suporta uma torre. A torre Y transporta um detector de radiação, câmara de ionização, cuja janela mantém-se em linha sobre o eixo Y e a torre X transporta a fonte radioativa. Sobre a torre X instalou-se outro motor de passo que ajusta a direção do feixe radiativo rumo ao centro geométrico do detector.

A cada incremento de 1,0 cm em X e em Y, totalizando 3.731 posições, o eletrômetro realiza a medida de carga elétrica gerada no interior de uma câmara de ionização, isso de forma automatizada. Estas medidas são lidas por um microcontrolador (PIC16F877A, programado em linguagem *assembly*) e armazenadas em uma memória serial (EEPROM), ambos residentes numa Placa de Circuito Impresso (PCI) que foi construída para esse projeto.

Finalizado o processo de medição os valores armazenados devem ser transmitidos a um computador pessoal, em modo texto, e transferidos a uma planilha eletrônica para serem trabalhados estatisticamente de forma a determinar a dose e taxa de dose absorvida.

Palavras-chaves: Microcontrolador, linguagem *assembly*, sistema de posicionamento, memória serial, leitura e escrita, tabela, calibração.

ABSTRACT

This work had as main objective the automation of electrical charge generated measures within a radiation detector, i.e., an automated device was developed for data acquisition. These measurements are performed by an electrometer and allow the determination of the dose and/or dose rate due to a radioactive source at various distances source-detector. It also allow the calibration of detectors with greater precision, simulating an infinite source and verification of sources homogeneity.

The above said automated device is supported by a mechanical structure comprising two tables moving orthogonally and each of them are driven by a stepping motor. The system has two directions of displacement, a call X (cartesian axis X) with freedom of movement -20.0 cm to 20.0 cm and 1.0 cm steps, while the other called the Y direction (axis Y) moves from 2.0 to 92.0 cm, and also 1.0 cm steps. For each direction it was built a table which supports a tower. The Y tower carries a radiation detector, ionization chamber, whose window remains inline on the Y axis and X tower carries the radioactive source. Upon the X tower it was settled another stepper motor that adjusts the direction of radiative beam towards the geometric center of the detector.

At each increment of 1.0 cm in X and Y, totaling 3,731 positions, the electrometer performs the measurement of electrical charge generated inside an ionization chamber in an automated form. These measures are read by a microcontroller (PIC16F877A, programmed in assembly language) and stored in a serial memory Electrically Erasable Programmable Read-Only Memory (EEPROM), both residing in a Printed Circuit Board (PCB) that was built for this project.

Finalized the process of measuring the stored values should be transmitted to a personal computer, in text mode, and transferred to a spreadsheet to be worked out statistically to determine the dose rate and absorbed dose.

Keywords: Microcontroller, assembly language, positioning system, serial memory, reading and writing, table, calibration.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ACK	*Acknowledge*
ASCII	*American Standard Code for Information Interchange*;
BCD	*Binary Coded Decimal*;
BIT	*BInary digiT*;
BYTE	Agrupamento de 8 bits;
bps	*Bits per second*;
CI	Circuito Integrado;
CMOS	*Complementary Metal-Oxide-Semiconductor*,
CP	Computador Pessoal;
CPU	*Central Processing Unit*;
CTS	*Clear To Send*;
DCD	*Data Carrier Detect*;
DCE	*Data Comunication Equipment;*
DEN	Departamento de Engenharia Nuclear da UFMG;
DIP	*Dual-in-line package*;
DTE	*Data Terminal Equipment;*
DSR	*Data Set Ready*;
DTR	*Data Terminal Ready*;
EIA	*Eletronic Industries Association;*
FIFO	*First In, First Out*;
EPROM	*Erasable Programmable Read-Only Memory*;
EEPROM	*Electrically Erasable Programmable Read-Only Memory*;
FIFO	*First In, First Out*;
GND	*Ground*;
LCD	*Liquid Crystal Display*;
LIFO	*Last In First Out;*
LSB	*Least Significant Bit*;
ICD 2	*In-Circuit Debugger 2*

I^2C ou IIC	*Inter-Integrated Circuit*;
I/O	*Input/Output*;
IR	*Instruction Reg*;
MCU	*Micro Controller Unit*;
MSSP	*Master Synchronous Serial Port*;
MODEM	***MO****dulator*/***DEM****odulator*;
MPU	*Micro Processor Unit*;
MSB	*Most Significant Bit*;
NIBBLE	Agrupamento de 4 bits;
OPCODE	*Operation Code*;
PC	*Program Counter*;
PCB	*Printed Circuit Board*;
PCH	*Program Counter High Byte*.
PCI	Placa de Circuito Impresso;
PCL	*Program Counter Low Byte*
PCLATH	*Program Counter High Byte Latch*.
PIC	*Programmable Intelligent Computer;*
PROM	*Programmable Read Only Memory*;
RAM	*Random Access Memory*;
RI	*Ring Indicator*;
RISC	*Reduced Instruction Set Computer*;
ROM	*Read Only Memory*;
RS-232	*Recommended Standard* (RS)-232;
RTS	*Request To Send*;
RX	*Receiver*;
SFR	*Special Function Registers*;
SPI	*Serial Peripheral Interface*;
SSPADD	*MSSP Address Register*
SSPBUF	*Serial Receive/Transmit Buffer Register*
SSPSR	*MSSP Shift Register*
TTL	*Transistor-Transitor Logic*;

TX	*Transmitter*;
TO	*Time Out*;
TOS	*Top Of Stack*;
ULA	Unidade Lógica e Aritmética;
USART	*Universal Synchronous Asynchronous Receiver Transmitter*;
USB	*Universal Serial Bus*;
WDT	*Watchdog Timer*;

SUMÁRIO

CAPÍTULO 1 – INTRODUÇÃO

Neste trabalho foi implementado um sistema automatizado para aquisição de dados. Esses dados são as medidas das cargas elétricas fornecidas pelo eletrômetro 616 (fabricado pela *Keithley Instruments*) e armazenadas em uma memória serial 24LC1025 que é uma *Electrically Erasable Programmable Read-Only Memory* (EEPROM). Estas cargas são geradas no volume sensível de uma câmara de ionização, devido à interação da radiação emitida por uma fonte radioativa e o gás no referido volume, ionizando-o. Os íons gerados são atraídos por eletrodos polarizados por uma fonte de tensão externa e uma malha resistor-capacitor de forma a produzir um pulso mensurável pelo eletrômetro. A partir dessas medidas pode-se determinar a dose absorvida a certa distância fonte-detector, a qual é definida como a energia absorvida por unidade de massa do material absorvedor (KNOLL, 1989).

Foi projetado e desenvolvido um sistema de posicionamento fonte-detector para trabalhar ortogonalmente, como os eixos cartesianos, e que, também, são interceptados em seus zeros, mas o deslocamento em Y começa a partir da posição 2,0. Sobre o eixo X uma mesa e uma torre acima dela transportam uma fonte radioativa de -20,0 a 20,0 cm. Sobre o eixo Y uma outra mesa e uma torre acima dela transportam uma câmara de ionização por 91 cm, da posição 2,0 à 92,0, cuja janela desta câmara fica fixamente direcionada sobre esse eixo. Esses deslocamentos, tanto em X quanto em Y, são incrementados a passos de 1,0 cm. Eles acontecem devido ao acoplamento de uma haste com rosca sem fim a uma porca fixa em cada mesa e são acionados por respectivos motores de passo. Um terceiro motor de passo, instalado sobre a torre X, tem a função de direcionar o feixe da fonte radioativa ao centro geométrico do detector (câmara de ionização).

O contorno dado a esse projeto condiz com os quesitos de relevância e ineditismo, ou seja, com esse desenvolvimento o Laboratório de Calibração do Departamento de Engenharia Nuclear da UFMG (DEN) passa a ter um sistema de medidas totalmente automatizado, permitindo a calibração de detectores com maior precisão. As rotinas desenvolvidas e o cálculo de dose possibilitam a simulação de uma fonte infinita, a verificação de homogeneidade das fontes e a dose transferida por uma fonte infinita. O programa de cálculo de dose, além da determinação da dose e/ou taxa de dose, determinam as flutuações estatísticas em função dos erros aleatórios e sistemáticos.

Além da infraestrutura mecânica, foi projetado e desenvolvido, também, toda infraestrutura eletrônica, hardware e software em linguagem *assembly*, necessária à plena execução do projeto proposto. O hardware foi composto por uma Placa de Circuito Impresso (PCI) principal onde se encontra: o microcontrolador PIC16F877A; o banco de memórias seriais 24LC1025, EEPROM; a interface *Recommended Standard*-232 (RS-232) (via um conector DB9 fêmea) para comunicação com um computador pessoal (CP)[1]; a interface com o eletrômetro (via um conector DB25 fêmea); um teclado com três teclas (I) POS_0_XY, (II) Automação, (III) TX ao computador pessoal; a interface com o *Liquid Crystal Display* (LCD) de 2 linhas e 16 colunas para externar os resultados parciais dos processamentos (comunicação máquina-homem); três saídas para controle dos *drivers* de três motores de passo. Para cada um desses *drivers* foi construído uma PCI específica. Também, dois cabos foram confeccionados, um cabo para interconexão da PCI principal ao CP e outro para interconexão PCI principal ao eletrômetro.

1.1 Objetivos

Esse trabalho teve como objetivo geral:

- Construir um dispositivo totalmente automatizado para aquisição de dados de um eletrômetro. Esse dispositivo foi composto por uma infraestrutura mecânica e sub-rotinas em linguagem *assembly*, que possibilitem o controle do deslocamento de uma torre sobre o eixo X (–20,0 a 20,0 cm) a passos de 1,0 cm, e outro, transversal a X, que desloca outra torre em Y (2,0 a 92,0 cm), também a passos de 1,0 cm – deslocamentos ortogonais. A torre X transporta a fonte radioativa e a torre Y uma câmera de ionização.

E teve como objetivos específicos:

- Desenvolver uma sub-rotina em linguagem *assembly* para gravar, ou escrever um dado na memória serial 24LC1025 (EEPROM), e outra para ler um dado gravado nessa memória.

[1] Escolheu-se a sigla CP para Computador Pessoal para não se confundir com PC de *Program Counter*, ou contador de programa, interno ao PIC16F877A, como será visto.

- Estudar o comportamento da interface digital externa do eletrômetro 6162, acoplado ao 616) produzidos pela *Keithley Instruments*.
- Elaborar e montar um circuito para interfaceamento do eletrômetro com o microcontrolador. Essa infraestrutura possibilita a realização das leituras das medidas realizadas por esse eletrômetro.
- Desenvolver uma sub-rotina em linguagem *assembly* que leia o dado *Binary Coded Decimal* (BCD) presente na porta digital do eletrômetro. Cada medida completa é composta por nove agrupamentos de 4 bits (*nibble*) em BCD.
- Desenvolver uma sub-rotina em linguagem *assembly* que agrupe dois dados BCD subseqüentes, lidos do eletrômetro, em um único agrupamento de 8 bits (byte), os quais devem ser armazenados em posições sequenciais na memória EEPROM serial.
- Desenvolver uma sub-rotina em linguagem *assembly* que divida um byte em dois *nibbles*, parte alta e parte baixa desse byte, e armazene-os temporariamente em registradores internos ao microcontrolador, esses previamente definidos.
- Desenvolver uma sub-rotina em linguagem *assembly* que converta um dado BCD em seu correspondente no código *American Standard Code for Information Interchange* (ASCII) para apresentação do dado no LCD. Essa conversão também é usada quando as medidas armazenadas na memória serial EEPROM forem transmitidas ao CP, via interface serial RS-232.
- Desenvolver uma sub-rotina em linguagem *assembly* para comunicação com o CP, via interface RS-232.

1.2 Compõem essa tese

Além desta introdução (Capítulo 1), a tese é composta pelo Capítulo 2 – Revisão Bibliográfica; Capítulo 3 – Materiais e Métodos; Capítulo 4 - Resultados e Conclusões; Referências e os Anexos. Esses anexos, ao fim do trabalho, dá a ideia completa de todo o desenvolvimento.

CAPÍTULO 2 – REVISÃO BIBLIOGRÁFICA

Neste trabalho foi realizada a automação das medidas de carga elétrica fornecidas por um eletrômetro. Essa automação foi suportada pelo microcontrolador PIC16F877A (fabricado pela Microchip Technology Inc.), o qual foi programado em linguagem *assembly*. Esse microcontrolador lê a medida na porta do eletrômetro, em BCD, combina duas leituras adjacentes em único byte e o armazena em uma posição endereçada da memória serial 24LC1025 (EEPROM). Após realizar todas as leituras do eletrômetro, a massa de dados gerada estará armazenada nessa EEPROM, sequencialmente. Esses dados são transmitidos ao CP via *Universal Synchronous Asynchronous Receiver Transmitter* (USART), pela interface RS-232, disponibilizando-os em uma planilha eletrônica.

Será estudado, nesse capítulo, o PIC16F877A e suas instruções *assembly*, a memória serial 24LC1025 que opera com o protocolo *Inter-Integrated Circuit* (IIC ou I^2C), a USART e o eletrômetro 616 e 6162 fabricados pela *Keithley Instruments*. Também fez parte desse desenvolvimento o controle da interface de comunicação máquina-homem pelo display LCD de 2 linhas e 16 colunas e, ainda, o controle de três motores de passo. Estes dois últimos pontos desenvolvidos, já se encontram publicados (FONSECA, 2008).

2.1 – O Microcontrolador PIC16F877A

Um microcontrolador é um Circuito Integrado (CI) que comporta toda infraestrutura básica de um microcomputador. Ou seja, integrou-se, em única pastilha semicondutora, todos os blocos funcionais de uma *Central Processing Unit* (CPU), um banco de memória não volátil e outro volátil, e, ainda, a maioria de seus pinos são utilizados como *Input/Output* (I/O). Por isso, pode-se admiti-lo como um microcomputador em único chip (ZANCO, 2005).

Nesse projeto, o microcontrolador PIC foi programado em linguagem *assembly*. A linguagem *assembly* mantém uma relação direta com o código executável, ou seja, cada instrução ocupa somente uma posição da memória de programa, o que não acontece nas demais linguagens

de programação. Por isso, dependendo da habilidade do programador, é possível desenvolver programas em *assembly* que utilizem menos espaço da memória de programa, e como, geralmente, os recursos dos microcontroladores são escassos (memória e pinos de I/O), o programador deve manter-se atento aos recursos disponíveis. Com o desenvolvimento desse projeto em linguagem *assembly* foi possível manter controlado a utilização dos referidos recursos.

2.1.1 – A sigla PIC

PIC é uma família de microcontroladores produzida pela Microchip Technology Inc.. Essa família teve início com o desenvolvimento do PIC1650 pela "*General Instrument's Microelectronic Division*", onde PIC era conhecido por "*Peripheral Interface Controller"*. Por isto costuma-se pensar que PIC seja "*Peripheral Interface Controller"*, mas, de início, os primeiros PICs referiam-se a *"Programmable Interface Controller",* o que foi substituído por *"Programmable Intelligent Computer"*. Portanto, nesse trabalho, a sigla PIC refere-se a *"Programmable Intelligent Computer"* (MICROELECTRONICS, 1977).

2.1.2 – Arquitetura

A Fig. 1 apresenta o *lay-out* externo do PIC16F877A. Esse microcontrolador é composto por 40 pinos, destes 33 são para I/O; 2 são para *Ground* (GND) (V_{SS}); 2 são para alimentação positiva (V_{DD}) ligados, nesse projeto, em $+5V_{CC}$; 1 é para *reset* (/MCLR) e os 2 últimos pinos são para entrada do circuito de *clock* (OSC1 e OSC2).

Os 33 pinos de I/O são divididos em cinco portas nomeadas por A, B, C, D e E. A porta A, que correlaciona com o registrador interno chamado de PORTA, tem pinos de I/O designados por RA0 a RA5 (6 pinos), onde RA0 é o *binary digit* (bit) menos significativo, ou *Least Significant Bit* (LSB), e o RA5 o bit mais significativo dessa porta, ou *Most Significant Bit* (MSB). A porta B, correlaciona-se com o registrador PORTB, tem pinos de I/O designados por

RB0 a RB7 (8 pinos), e PORTC (RC0 a RC7, 8 pinos), PORTD (RD0 a RD7, 8 pinos) e por fim, PORTE (RE0 a RE2, 3 pinos) (MICROCHIP, 2003a).

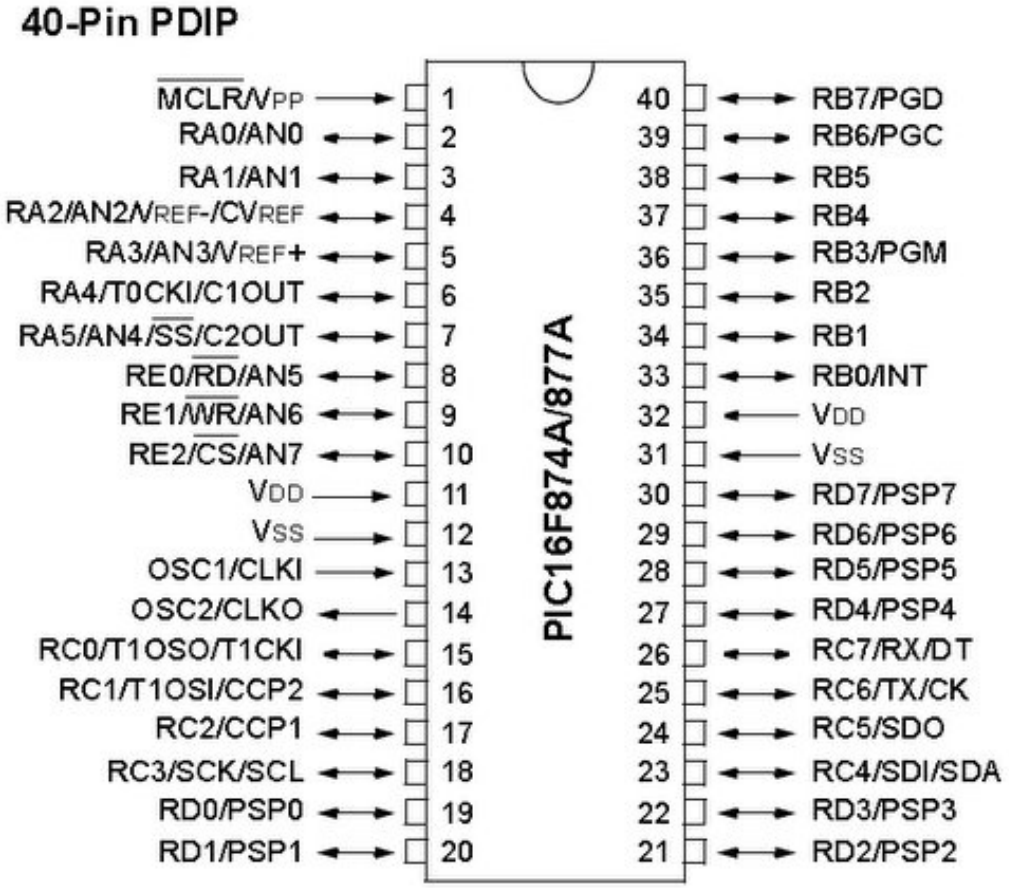


Figura 1 – *Lay-out* externo do microcontrolador PIC16F877A
Fonte: Microchip, 2003a.

A arquitetura interna do PIC16F877A pode ser vista na Fig. 2.

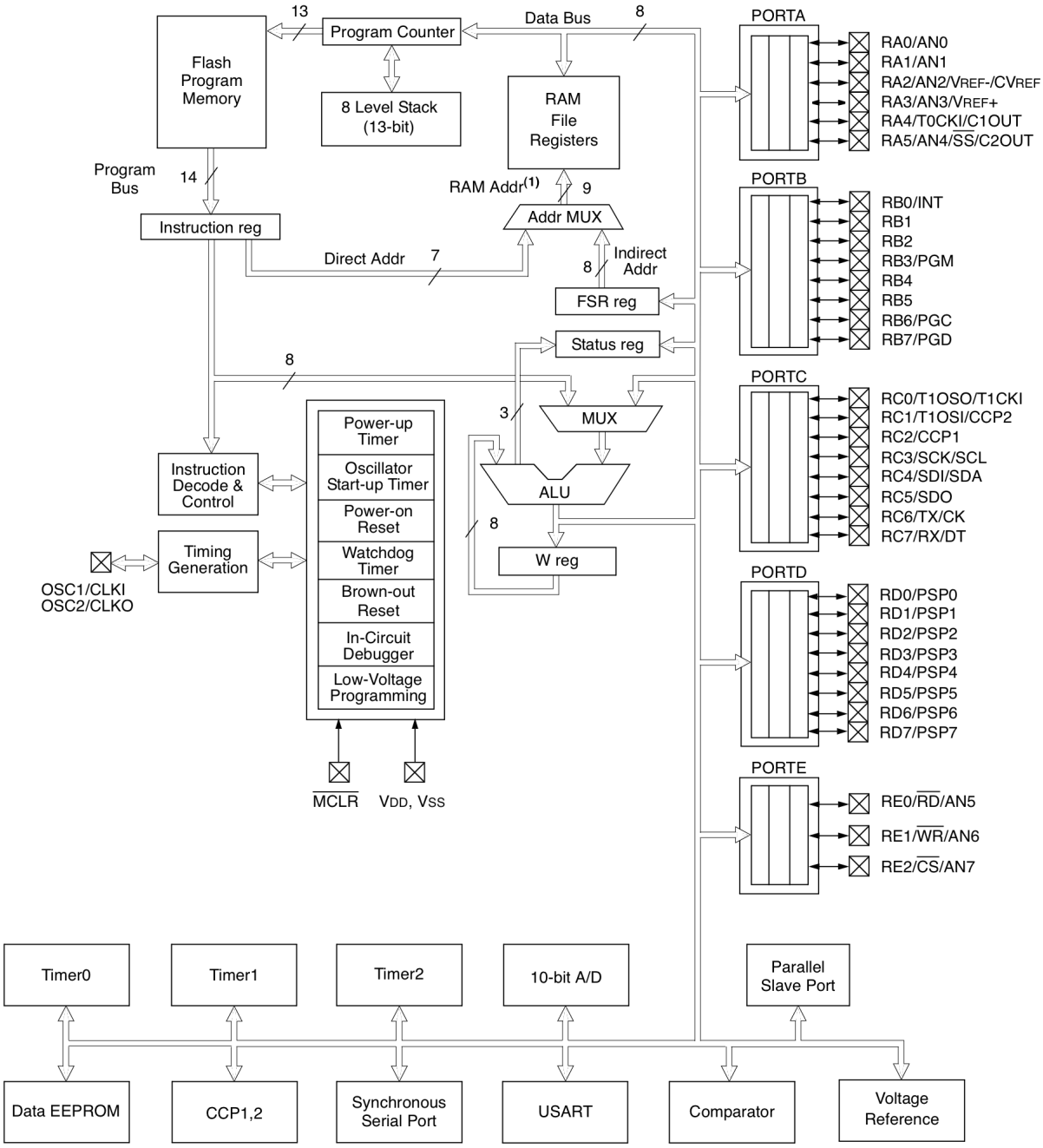


Figura 2 – *Lay-out* interno do microcontrolador PIC16F877A
Fonte: Microchip, 2003a.

A Fig. 2 mostra a arquitetura interna do PIC16F877A. Esse microcontrolador é composto por uma memória de programa (*Flash*), onde o programa executável é gravado; por duas memórias de dados, uma volátil, a *Random Access Memory* (RAM), onde reside os registradores especiais e os registradores definidos e utilizados pelo usuário, e a EEPROM, para armazenamento de dados não voláteis. Encontra-se, também, o *Program Counter* (PC), que aponta para a próxima instrução a ser executada; os oito níveis de pilha (*stack*), que funciona segundo o princípio *Last In First Out* (LIFO), ou seja, o último endereço de retorno que entrou na pilha é o primeiro a sair dela (TAUB, 1984). Contém, ainda, a Unidade Lógica e Aritmética (ULA); os registradores W (Work, ou acumulador noutras tecnologias); o *Instruction Reg*. (IR), ou registrador de instrução; o decodificador de instruções; a USART; os barramentos internos; as portas de I/O, e demais blocos funcionais.

2.1.3 – Período do oscilador e do ciclo de máquina

A base de tempo utilizada pelo microcontrolador advém, por vezes, de um oscilador a cristal, o qual oscila em sua frequência fundamental f_{OSC} (Gomes, 1985). Sabe-se que o período desse oscilador é dado pelo inverso de f_{OSC}, conforme a equação (1). O tempo gasto na execução de uma instrução, período do ciclo de máquina, ou período T_{CY}, é, de fato, o que importa para programar em *assemblly*. Esse período, T_{CY}, é calculado conforme a equação (2) (SOUZA, 2006; MICROCHIP, 2001).

$$T_{OSC} = (1/ f_{OSC}) \quad (1)$$

$$T_{CY} = 4\ T_{OSC} \quad (2)$$

Nesse projeto utilizou-se um cristal de 4MHz. Portanto, o período do oscilador (T_{OSC}) é de 0,25μs e período de um ciclo de máquina (T_{CY}) é de 1μs. A maioria das instruções do PIC requer de um ciclo de máquina (1 T_{CY}) para ser executada. Poucas instruções requerem de 2 T_{CY}.

2.1.4 – O contador de programa - PC

O PC, contador de programa, é um contador digital síncrono. Sua fonte de *clock*, base de tempo para contagem, pode ser fornecida por um oscilador a cristal externo ao PIC. A cada T_{CY}, nesse projeto a cada 1μs, o contador é incrementado (PC ← PC + 1). Como indicado na Fig. 2, o PC é o ponteiro para a próxima instrução a ser executada, ou seja, o seu conteúdo, valor atual da contagem, aponta para o endereço da memória de programa onde reside a instrução que deverá ser buscada (ciclo de busca, que é a carga do IR com os 14 bits da instrução), cujo conteúdo é formado por um *Operation Code* (OPCODE) (Microchip, 2011) e um ou mais operandos (Microchip, 1997). A designação desse código, OPCODE ou Código de Operação, acompanha a tecnologia dos microprocessadores desde seu início (MALVINO, 1985). Esse código também é conhecido por código objeto (OLIVEIRA, 1986; MALDONADO, 1990).

Pode-se perceber, também na Fig. 2, que o PC está ligado à pilha (*stack*) por uma via bidirecional. Isso possibilita que o conteúdo do PC seja gravado temporariamente em uma das oito posições da pilha, ou lido na pilha e escrito no PC. Essas atividades estão associadas às execuções das instruções CALL, RETURN, RETLW e, também, ao tratamento de uma interrupção e seu retorno ao programa principal, RETFIE.

Na execução da instrução CALL, chamada de uma sub-rotina, o conteúdo do PC é gravado na pilha, esse aponta para a instrução subseqüente ao endereço onde reside a instrução CALL (TAUB, 1984; CYPRIANO, 1984). As instruções RETURN e RETLW leem na pilha o endereço para retorno e o carrega no PC.

Para o tratamento de uma interrupção, solicitada por algum periférico, o conteúdo do PC, que aponta para a próxima instrução que seria executada se não ocorresse a interrupção, é escrito na pilha e, imediatamente, passa a conter o endereço 0x0004, que é o endereço onde se inicia a rotina de tratamento de uma interrupção. Ao terminar a execução da rotina de interrupção tem que retornar ao programa principal (RETFIE, onde o endereço de retorno é lido na pilha e carregado no PC) (VISCONTI, 1983; CYPRIANO, 1984; MALVINO, 1985).

Em todas as execuções que envolvem a pilha o conteúdo do PC é gravado em seu topo, ou *Top Of Stack* (TOS) (Microchip, 1997), e que o mecanismo de trabalho da pilha, como já se disse, é LIFO, o último endereço a entrar na pilha é o primeiro a sair dela.

2.1.5 – Memórias

Os tipos de memórias internas ao PIC16F877A são as memórias de dados (RAM e EEPROM) e a memória de programa (*Flash*). Serão apresentadas as memórias *Flash* e RAM, pois são as de interesse direto à aplicação nesse projeto.

2.1.5.1 – Memória de programa

A memória de programa do PIC16F877A é dividida em 4 páginas de 2k *Words.* Cada *word*, ou palavra nesse PIC, é composta por um conjunto de 14 bits. Na Fig. 2, na saída da memória de programa, pode-se observar que o *Program Bus* comporta 14 bits ou uma *word*. Portanto, a capacidade total de armazenamento da memória de programa é de 8k *Words*, ou 8.192 lugares endereçáveis que comportam 14 bits. Sendo o primeiro endereço 0000h, extrai-se que o último é 1FFFh.

A página 0 (zero) inicia-se no endereço 0005h (os endereços de 0000h ao 0004h são reservados, respectivamente para *reset* e para o início da rotina de tratamento de interrupção – esta somente quando for habilitada) e termina no endereço 07FFh. A página 1 (um) inicia-se no endereço 0800h e termina no 0FFFh. A página 2 (dois) inicia-se no endereço 1000h e termina no endereço 17FFh. E, por fim, a página 3 (três) inicia-se no endereço 1800h e termina no endereço 1FFFh.

Essa divisão da memória de programa gera a necessidade de maior atenção do programador quando o programa *assembly* ocupar mais de uma dessas páginas. Até 2k não gera nenhum problema. Deve-se ter essa atenção quando for executar uma instrução CALL (salto incondicional – chamada de uma sub-rotina com retorno à instrução subsequente a esse CALL) ou uma instrução GOTO (salto incondicional, sem retorno). Isso ocorre porque nessas instruções o fabricante alocou apenas os 11 bits menos significativos do endereço (k<10:0>) para onde deverá saltar o processamento, conforme observa-se na Tab.1. E, como se sabe, com 11 bits é possível endereçar apenas os referidos 2k, mantendo-se dentro de uma mesma página.

Tabela 1 – Instruções CALL e GOTO e seus OPCODEs

Instrução	Descrição	OPCODE (14 bits) (*)
CALL k	Chama a sub-rotina iniciada no endereço k	10 0kkk kkkk kkkk
GOTO k	Vá para o endereço k	10 1kkk kkkk kkkk

(*) Código identificador da instrução e operando
Fonte: Microchip, 2003a, com adaptações.

Para o endereçamento total da memória de programa – página 0 a 3 – o fabricante do PIC teve que agregar aos 11 bits, acima mencionados, outros dois bits. Estes bits servem como ponteiro de página, e foi utilizado os bits 3 e 4 do registrador PCLATH, conforme Tab. 2 e Fig. 3.

Tabela 2 – Páginas de 2k da Memória de Programa

PCLATH, ponteiro de página		Página	Endereço da Memória de Programa	
Bit 4	Bit 3		Inicial	Final
0	0	0	0x0005	0x07FF
0	1	1	0x0800	0x0FFF
1	0	2	0x1000	0x17FF
1	1	3	0x1800	0x1FFF

Fonte: Microchip, 2003a, com adaptações.

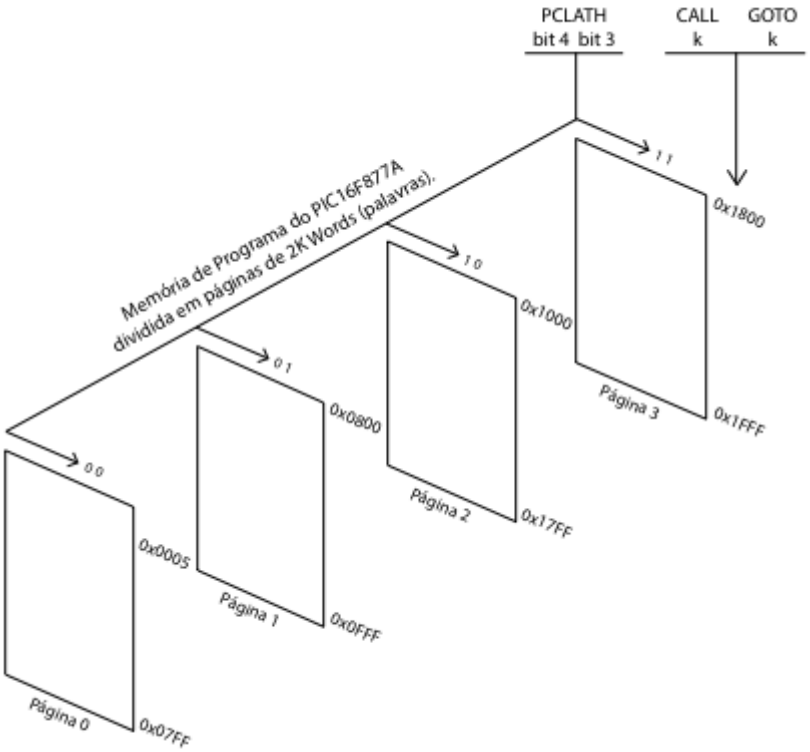


Figura 3 – Memória de programa do PIC16F877A dividida em 4 páginas.

Dois registradores do PIC podem atuar no contador de programa, o *Program Counter Low Byte* (PCL) e o *Program Counter High Byte Latch* (PCLATH). O contador PC é de 13 bits (PC<12:0>). Os seus oito bits menos significativos (PC<7:0>) podem ser carregados a partir do registrador PCL. O conteúdo do registrador PCL pode ser lido ou escrito, dependendo, somente, do que se executa. Os bits mais significativos do PC (PC<12:8>) não podem ser lidos, mas pode-se escrever nestas posições o conteúdo presente no registrador PCLATH (Microchip, 2003).

A Fig. 4 ilustra a carga do PC quando se executa a instrução CALL ou GOTO. O operando (k<10:0>) que compõe qualquer destas instruções, Tab.1, é carregado nos 11 bits menos significativos do PC. Os bits 11 e 12 do PC são carregados com o valor presente nos bits 3 e 4 do registrador PCLATH no momento da execução de qualquer destas instruções, como se observa na Fig.4.

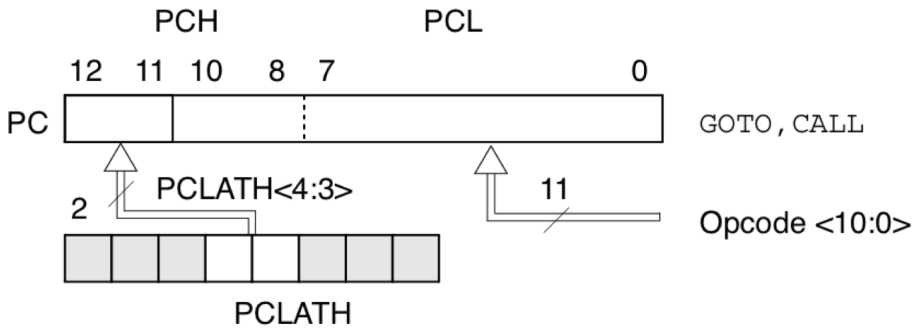


Figura 4 – Carga do PC no momento da execução das instruções CALL e GOTO
Fonte: Microchip, 2003a.

A Fig. 5 mostra que o valor atual do PC [*Program Counter High Byte* (PCH) + PCL] é alterado no exato momento em que se executa uma instrução envolvendo o registrador PCL. Ou seja, no exato momento que é carregado um valor no PCL, este resultante da execução de uma instrução que tenha o registrador PCL como destino final, é realizado o carrregamento do PC. Assim, pode-se pré-carregar um valor em PCLATH e não acontecerá nenhuma alteração com o valor presente no PC. Isto só ocorrerá caso haja uma operação envolvendo o registrador PCL com carga do resultado nesse registrador.

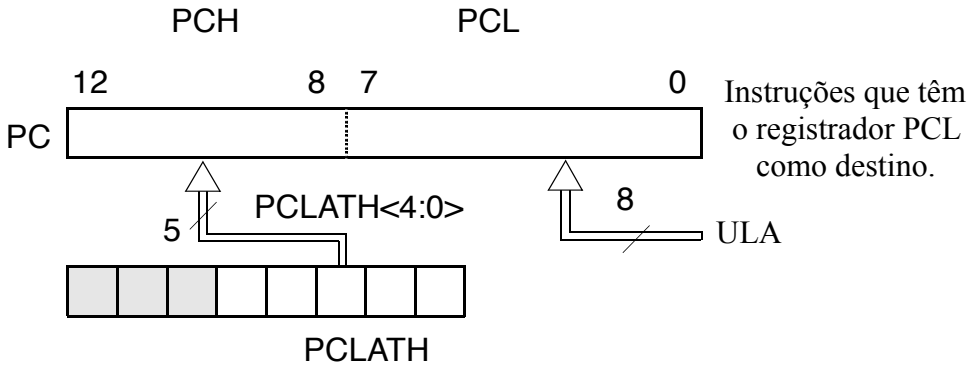


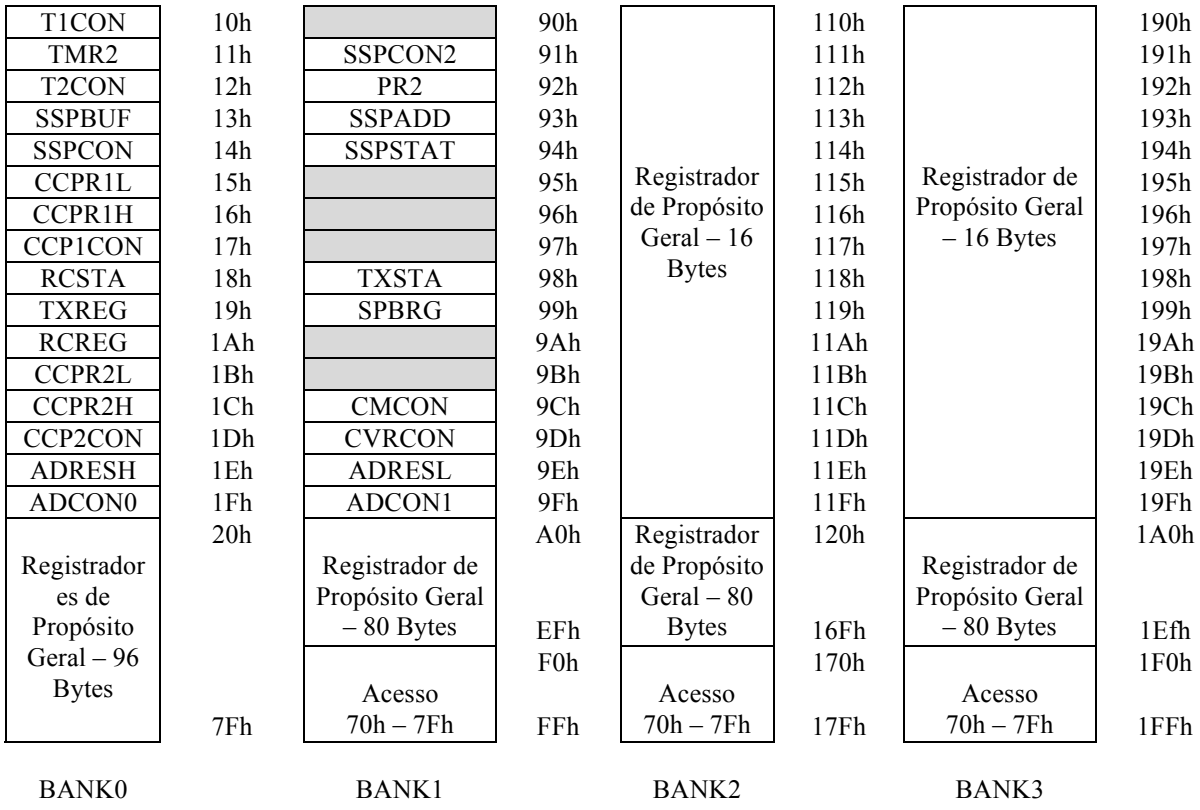
Figura 5 – Carga do PC na execução de instruções que envolvem o PCL
Fonte: Microchip, 2003a, com adaptações

Esses procedimentos – execução do CALL, do GOTO e das instruções que envolvem o registrador PCL – são úteis quando se trabalha com tabelas e o programa, como um todo, ocupar mais de 2k da memória de programa. Nesse projeto utilizou-se mais de 90 tabelas. A maior parte dessas tabelas guardam a quantidade de passos para o motor de passo de direcionamento do feixe radioativo ao centro geométrico do detector, dada uma posição X e Y. Esse grande número de tabelas ocupou mais de uma página (2k) da memória de programa, o que demandou por ajustes em PCLATH para evitar erros de leitura nessas tabelas.

2.1.5.2 – Memória de Dados

O PIC16F877A contém dois bancos de memória de dados. A RAM, volátil, será tratada nesse trabalho, e a EEPROM, não volátil, não será tratada aqui por não ter sido utilizada nesse projeto. A RAM é do tipo estática e comporta 512 bytes (endereços de 000h a 1FFh). Ela foi dividida em 4 bancos designados por BANK0 a BANK3. Nestes bancos criou-se um conjunto de Registradores Especiais e foi deixado um espaço livre para o programador definir e utilizar como variáveis do sistema implementado – espaço designado por Registradores de Propósito Geral, como mostrado na Fig. 6.

Reg.	End. Reg	Reg.	End. Reg	Reg.	End. Reg	Reg.	End. Reg
INDF(*)	00h	INDF(*)	80h	INDF(*)	100h	INDF(*)	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h		105h		185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h		107h		187h
PORTD	08h	TRISD	88h		108h		188h
PORTE	09h	TRISE	89h		109h		189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDATA	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh		18Eh
TMR1H	0Fh		8Fh	EEADRH	10Fh		18Fh
T1CON	10h		90h		110h		190h
TMR2	11h	SSPCON2	91h		111h		191h



Local não implementado na memória de dados, mantido em nível lógico baixo (0).
(*) Não foi fisicamente implementado estes registradores.

Figura 6 – Registradores especiais e de propósito geral – RAM
Fonte: Microchip, 2003a.

Observa-se na Fig. 6 que alguns registradores especiais, ou *Special Function Registers* (SFR), são espelhados, ou seja, eles estão presentes em todos os bancos (ex. PCL, STATUS, FSR, etc.). Neste caso, se o conteúdo de um deles for alterado os demais serão alterados para o mesmo conteúdo. Observa-se, também, que o registrador PORTB e seu registrador de configuração TRISB foram construídos em dois bancos distintos, espelhados entre si.

Exceto os registradores totalmente espelhados, os demais precisam ser apontados, ou melhor, o banco onde residem deve estar selecionado para que se possa ler ou escrever o seu conteúdo. Estes bancos (BANK0 a 3) são selecionados pelos bits 5 e 6 do registrador STATUS (endereçamento direto, Fig. 7 e item 2.1.6.2), chamados por RP1 e RP0, respectivamente. Quando seus valores forem 00 estarão apontando para o BANK0, e neste momento pode-se atuar, ler ou escrever, em qualquer dos registradores deste BANK0 – cujo endereçamento, dentro do

banco, vem nos 7 bits menos significativos do OPCODE da instrução, designados por f (ou *file*). Quando os valores de RP1 e RP0 forem 01 estarão apontando para o BANK1; 10, para o BANK2; e, 11, para o BANK3, (Souza, 2006; Microchip, 2003).

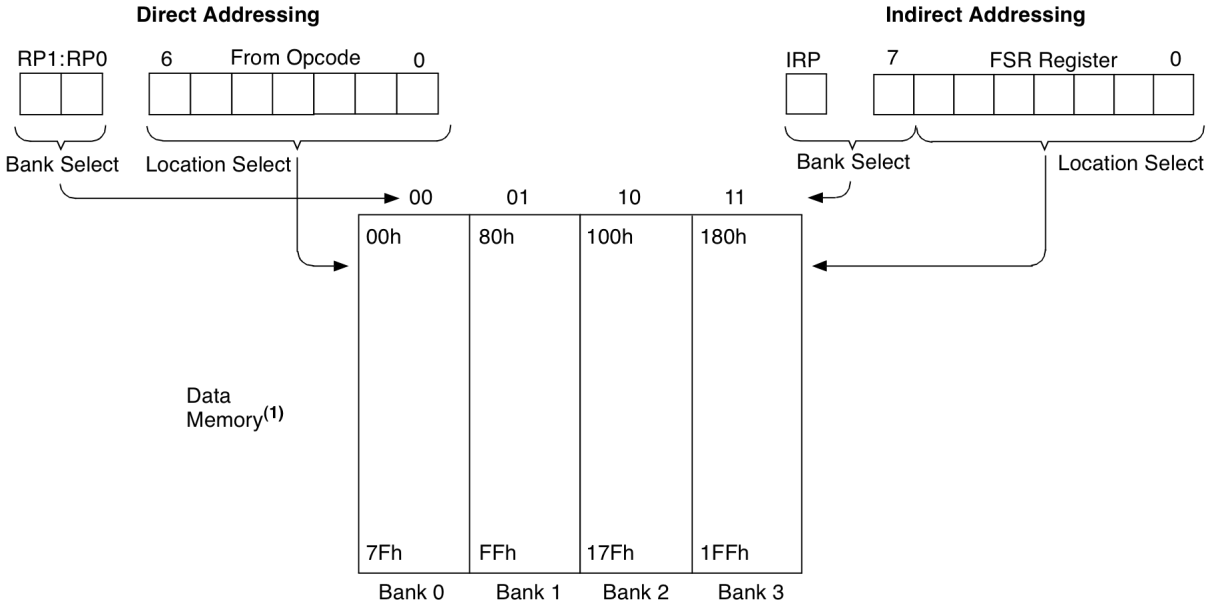


Figura 7 – Endereçamento direto e indireto do banco de memória de dados
Fonte: Microchip, 2003a.

Pode-se endereçar os referidos bancos de memória de dados, também, por endereçamento indireto. Como indicado na Fig. 7, participam da seleção dos bancos os bits IRP (bit 7 do registrador STATUS, item 2.1.6.2) e o bit 7 do registrador FSR. Nota-se, também, que os demais bits, presentes no registrador FSR, selecionam o registrador dentro de um determinado banco.

No próximo item será tratado de alguns registradores. Nesse primeiro momento, será tratado dos registradores especiais de uso amplo. Dos registradores especiais mais especializados, serão tratados quando se falar do assunto de sua especialização, ou seja, do protocolo I^2C e da USART.

2.1.6 – Registradores especiais – uso geral

Como foi apresentado nos itens anteriores, na memória RAM, memória de dados do PIC, o fabricante definiu endereços com funções dedicadas. Foi feito uma associação entre um endereço e um nome identificador de um determinado registrador – lugar bem definido onde pode-se guardar informações, ou dados de 8 bits, para configurar o perfil de trabalho de um determinado programa *assembly*. Um mesmo programa pode demandar perfis distintos em distintos processamentos. Isto não chega a ser um problema. Nesse projeto trabalhou-se com uma configuração ou perfil constante, exceto pelos ajustes, variações de perfil, promovidos ao nível de *hardware*.

Os registradores especiais têm um nome identificador para que possa ser lido ou escrito em algum momento do processamento, e, a grande maioria deles tem, também, um nome identificador para cada um de seus bits, que são numerados do menos significativo (b_0), mais à direita, ao mais significativo (b_7), mais à esquerda. Será apresentado o significado destes bits, componentes dos registradores utilizados nesse trabalho.

Por ora será tratado dos registradores TRISx e PORTx (onde x pode assumir as letras de A a E), STATUS, OPTION_REG, INTCON e do ADCON1, que são de uso geral. Em tópicos específicos serão tratados dos registradores especializados, referentes ao protocolo I^2C e à transmissão pela USART.

2.1.6.1 – Registradores TRISx e PORTx

Uma *Micro Controller Unit* (MCU) contém internamente todos os blocos funcionais de um microcomputador, e, também, as portas de I/O. A MCU PIC16F877A tem 5 portas de I/O, as quais comportam 33 pinos de entrada/saída. Destes pinos de I/O alguns foram configurados para aplicações específicas [ex. o pino RC3 pode ser configurado para SCK – *clock*, e o RC4 para SDA – *Transmitter* (TX) / *Receiver* (RX) de dados do protocolo I^2C – ambos usados para o controle da leitura e escrita da memória 24LC1025], e os demais pinos são para I/O convencionais.

Como são portas de I/O e os dispositivos ligados a elas podem ser de entrada (teclado, leitura do eletrômetro, etc.), ou podem ser de saída (LCD, controle dos motores de passo, etc.), então estas portas deverão ser configuradas segundo os dispositivos controlados e ligados a elas.

Para configurar uma das portas do PIC (PORTA a PORTE, Tab. 4) deve-se carregar com determinado byte o respectivo registrador TRIS, Tab. 3. Exemplificando: para configurar a PORTA, ou formatar um perfil de trabalho para ela, deve-se carregar o registrador TRISA, a PORTB o TRISB e assim por diante. Mas, como determinar o byte a ser carregado? Dado que a MCU é orientada a bit, cada bit tem seu significado – bem definido – dentro de um registrador. Os bits de uma porta, qualquer das porta de I/O, podem ser configurados como entrada (para isso, deve-se carregar com nível lógico alto – 1, o qual lembra I de *Input*, em inglês), ou podem ser configurados como saída (carrega o nível lógico baixo – 0, o qual lembra O de *Output*, em inglês), (PEREIRA, 2006).

Tabela 3 – Registrador TRISx

Registrador: **TRISx**				Endereços: **Ref. de cada TRISx**			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Rx7	Rx6	Rx5	Rx4	Rx3	Rx2	Rx1	Rx0

x = pode ser A a E.

Tabela 4 – Registrador PORTx

Registrador: **PORTx**				Endereços: **Ref. de cada PORTx.**			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Rx7	Rx6	Rx5	Rx4	Rx3	Rx2	Rx1	Rx0

x = pode ser A a E.

Dado o contexto anterior e evidenciando a PORTB (RB<7:0>, especificamente os seus terminais menos significativos (RB<3:0>), suponha que estes sejam ligados a quatro chaves de um teclado. E, ainda, que os seus outros pinos (RB<7:4>) sejam ligados a quatro LEDs. Então, para configurar a PORTB, de acordo com a situação apresentada, tem-se que executar: TRISB ←

0x0F (carrega TRISB com o valor binário ‘00001111’). Desse modo, os pinos menos significativos da PORTB (RB<3:0>) serão configuradas como entrada (1), e os mais significativos (RB<7:4) como saída (0), conforme segue o trecho de programa em *assembly*:

```
MOVLW       0x0F        ; W ← 0x0F
MOVWF       TRISB       ; TRISB ← W.
```

O trecho de programa acima segue uma estrutura de um programa organizado, ou seja, o código fonte deve ser elaborado obedecendo os seguintes campos: (VISCONTI, 1983)

Label Mnemônico da Instrução Operandos (;) Comentário

A palavra inglesa *label* significa etiqueta, rótulo, ou mesmo um nome para a chamada de uma rotina. Mnemônico é um conjunto de letras identificadoras da instrução. Os operandos podem ser um valor, um ou mais registradores, etc., que são utilizados na execução da instrução indicada pelo seu mnemônico. E, após o sinal gráfico de ponto e virgula (;), vem os comentários que, bem estruturados, podem ser de grande valia quando, no futuro, for promover uma melhoria no software.

Também deve ser observado que as portas PORTA e PORTE não têm todos os bits do byte; a PORTA<5:0>, portanto 6 bits, e a PORTE<2:0>, portanto 3 bits. As demais portas, PORTB, PORTC e PORD têm os 8 bits.

2.1.6.2 – Registrador STATUS

No registrador STATUS é salvo o estado da última operação realizada pela ULA (bits C, DC e Z). Este registrador é, também, composto pelo bit *Time Out* (/TO) que participa do processo de *WatchDog Timer* (WDT) – que não será tratado nesse projeto. E, ainda, pelos bits para endereçamento direto da memória de dados (RP1 e RP0) e para endereçamento indireto (IRP). A Tab. 5 fornece mais informações sobre o registrador STATUS.

Tabela 5 – Registrador STATUS

Registrador**: STATUS**				Endereços: **03h, 83h, 103h e 183h**			
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
R/W	R/W	R/W	R	R	R/W	R/W	R/W
IRP	RP1	RP0	/TO	/PD	Z	DC	C
Condição após *reset* por *Power-On Reset* (POR)							
0	0	0	1	1	X	X	X

Fonte: Microchip, 2003a, com adaptações. X = irrelevante.

IRP: Bit **s**eletor de banco de memória, usado para endereçamento indireto, Fig. 7.

0 = Banco 0 e 1 (00h – FFH).
1 = Banco 2 e 3 (100h – 1FFH).

RP1 e RP0: Bits para selecionar o banco ativo, usados para endereçamento direto, Fig. 7.

00 = Banco 0 (00h – 7Fh).
01 = Banco 1 (80h – FFh).
10 = Banco 2 (100h – 17Fh).
11 = Banco 3 (180h – 1FFh). **Obs**.: Cada banco comporta 128 bytes.

/TO: Bit que indica a ocorrência de *time-out* (ou NOT_TO)

0 = indica que ocorreu o estouro de *Watchdog timer* (WDT).

1 = Após um *power-up*, ou após a execução de uma instrução CLRWDT ou SLEEP.

/PD: Bit que indica a ocorrência de *power-down* (ou NOT_PD)

0 = Indica que a instrução SLEEP foi executada.

1 = Após um *power-up*, ou após a execução de uma instrução CLRWDT.

Z : Bit que indica uma operação cujo resultado foi igual a Zero

0 = Indica que o resultado da última operação (lógica ou aritmética) não foi zero.

1 = Indica que o resultado da última operação (lógica ou aritmética) resultou em zero.

DC: Bit de *Digit Carry* / $\overline{borrow}$, Transporte / Empréstimo: (PEREIRA/2006)

0 = A última operação da ULA não ocasionou um estouro de dígito.

1 = A última operação da ULA ocasionou um transporte (*carry*) entre o bit 3 e 4, isto é, o resultado ultrapassou os 4 bits menos significativos.

C: Bit de *Carry* / $\overline{borrow}$, Transporte / Empréstimo: (PEREIRA, 2006)

0 = A última operação da ULA não ocasionou um estouro (*carry*).

1 = A última operação da ULA ocasionou um estouro (*carry*) no bit mais significativo, isto é, o resultado ultrapassou os 8 bits disponíveis.

2.1.6.3 – Registrador OPTION_REG

O registrador OPTION_REG é utilizado para configurar alguns periféricos internos da MCU, como pode-se verificar na descrição de cada pino da Tab. 6.

Tabela 6 – Registrador OPTION_REG

Registrador**: OPTION_REG**				Endereços: **81h e 181h**			
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
/RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
Condição após *reset* por *Power-On Reset* (POR)							
1	1	1	1	1	1	1	1

Fonte: Microchip, 2003a, com adaptações.

/RBPU: Bit de habilitação dos *pull-ups* internos da PORTB:

0 = *Pull-ups* internos da PORTB habilitados em todos os seus pinos.

1 = *Pull-ups* internos da PORTB desabilitados.

INTEDG: Bit de seleção da transição (*edge*) ou borda que gerará uma interrupção externa pelo pino RB0/INT.

0 = A interrupção ocorrerá na borda de descida.

1 = A interrupção ocorrerá na borda de subida.

T0CS: Bit de seleção da fonte de *clock* que incrementará o temporizador TMR0.

0 = TMR0 será incrementado internamente pelo *clock* da máquina.

1 = TMR0 será incrementado externamente pela mudança no pino RA4/T0CKl.

T0SE: Bit de seleção da borda (*edge*) que incrementará o TMR0, quando T0CS =1.

0 = O incremento ocorrerá na borda de subida no pino RA4/T0CKl.

1 = O incremento ocorrerá na borda de descida no pino RA4/T0CKl..

PSA: Bit que configura a qual dispositivo será aplicação o *prescaler*.

0 = O *prescaler* será aplicado ao módulo TMR0.

1 = O *prescaler* será aplicado ao WDT.

PS2: PS0 Bits para a seleção da taxa do *prescaler*, conforme Tab. 7.

Tabela 7 – Bits PS2, PS1 e PS0

Bits PS (2, 1, 0)	TMR0	WDT
0 0 0	1:2	1:1
0 0 1	1:4	1:2
0 1 0	1:8	1:4
0 1 1	1:16	1:8
1 0 0	1:32	1:16
1 0 1	1:64	1:32
1 1 0	1:128	1:64
1 1 1	1:256	1:128

Fonte: Microchip, 2003a.

Não trabalhou, nesse projeto, com nenhum dos temporizadores (seja o TMR0 ou o WDT).

2.1.6.4 – Registrador INTCON

O registrador INTCON serve para configurar e identificar as interrupções. Não foi usado nenhuma interrupção nesse projeto, mas, ainda assim, este registrador deve ser configurado. O descritivo de cada bit do registrador INTCON (Tab. 8) facilita sua configuração.

Tabela 8 – Registrador INTCON

Registrador: **INTCON**				Endereços: **0Bh e 8Bh**			
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
GIE	EEIE	T01E	INTE	RBIE	T0IF	INTF	RBIF
Condição após *reset* por *Power-On Reset* (POR)							
0	**0**	**0**	**0**	**0**	**0**	**0**	**X**

Fonte: Microchip, 2003a, com adaptações. X = irrelevante.

GIE: Habilitação geral das interrupções (atua como uma chave geral):

0 = Nenhuma interrupção será tratada.

1 = As interrupções habilitadas individualmente serão tratadas.

EEIE: Habilitação da interrupção de final de escrita na EEPROM (chave individual)

0 = A interrupção não será tratada.

1 = A interrupção será tratada.

T0IE: Habilitação da interrupção de estouro de TMR0 (chave individual):

0 = A interrupção não será tratada.

1 = A interrupção será tratada

INTE: Habilitação da interrupção externa no pino RB0 (chave individual)

0 = A interrupção não será tratada.

1 = A interrupção será tratada

RBIE: Habilitação da interrupção por mudança de estado nos pinos RB4 a RB7 (chave individual)

0 = A interrupção não será tratada.

1 = A interrupção será tratada.

T0IF: Identificação da interrupção de estouro de TMR0:

0 = Esta interrupção não ocorreu.

1 = Esta interrupção ocorreu.

INTF: Identificação da interrupção externa no pino RB0:

0 = Esta interrupção não ocorreu.

1 = Esta interrupção ocorreu.

RBIF: Identificação da interrupção por mudança de estado nos pinos RB4 a RB7:

0 = Esta interrupção não ocorreu.

1 = Esta interrupção ocorreu.

2.1.6.5 – Registrador ADCON1

O registrador ADCON1 é usado para configurar as portas PORTA e PORTE como analógica (A) ou digital (D). Esta, assim configurada, é uma I/O convencional – Tab. 9, Tab. 10 e Tab. 11. Nesse projeto foi utilizado todas as portas como I/O convencional (D).

Tabela 9 – Registrador ADCON1

Registrador: **ADCON1**				Endereço: **9Fh**			
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
R/W	R/W	U	U	R/W	R/W	R/W	R/W
ADFM	ADCS2	-	-	PCFG3	PCFG2	PCFG1	PCFG0
Condição após *reset* por *Power-On Reset* (POR)							
0	0	0	0	0	0	0	0

Fonte: Microchip, 2003a, com adaptações. U = não implementado, em zero.

ADFM: Justificação para o resultado da conversão A/D de 10 bits:

0 = Justificado à esquerda. Os 6 bits menos significativos de ADRESL são lidos em 0.

1 = Justificado à direita. Os 6 bits menos significativos de ADRESH são lidos em 0.

ADCS2: Conversão A/D - bit seletor de *clock*, este bit trabalha conjuntamente com os bits ADCS1 e ADCS0, bits 7 e 6 respectivamente, do registrador ADCON0, segundo a Tab. 10.

Tabela 10 – Bit ADCS2 do registrador ADCON1

ADCON1 <ADCS2>	**ADCON0 <ADCS1:ADCS0>**	**Conversão de *clock***
0	00	F_{OSC} / 2
0	01	F_{OSC} / 8
0	10	F_{OSC} / 32
0	11	F_{RC} (*Clock* derivado do oscilador RC interno para A/D).
1	00	F_{OSC} / 4
1	01	F_{OSC} / 16
1	10	F_{OSC} / 34
1	11	F_{RC} (*Clock* derivado do oscilador RC interno para A/D).

Fonte: Microchip, 2003a, com adaptações.

PCFG<3:0> Esses bits são utilizados para configuração das portas A/D, mostrados na Tab. 11.

Tabela 11 – Configura PORTA e PORTE como A ou D

PCFG <3:0>	AN7 RE2	AN6 RE1	AN5 RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	V_{REF+}	V_{REF-}	C/R
0000	A	A	A	A	A	A	A	A	V_{DD}	V_{SS}	8/0
0001	A	A	A	A	V_{REF+}	A	A	A	AN3	V_{SS}	7/1
0010	D	D	D	A	A	A	A	A	V_{DD}	V_{SS}	5/0
0011	D	D	D	A	V_{REF+}	A	A	A	AN3	V_{SS}	4/1
0100	D	D	D	D	A	D	A	A	V_{DD}	V_{SS}	3/0
0101	D	D	D	D	V_{REF+}	D	A	A	AN3	V_{SS}	2/1
011x	D	D	D	D	D	D	D	D	–	–	0/0
1000	D	D	D	A	V_{REF+}	V_{REF-}	A	A	AN3	AN2	6/2
1001	D	D	D	A	A	A	A	A	V_{DD}	V_{SS}	6/0
1010	D	D	D	A	V_{REF+}	A	A	A	AN3	V_{SS}	5/1
1011	D	D	D	A	V_{REF+}	V_{REF-}	A	A	AN3	AN2	4/2
1100	D	D	D	A	V_{REF+}	V_{REF-}	A	A	AN3	AN2	3/2
1101	D	D	D	D	V_{REF+}	V_{REF-}	A	A	AN3	AN2	2/2
1110	D	D	D	D	D	D	D	A	V_{DD}	V_{SS}	1/0
1111	D	D	D	D	V_{REF+}	V_{REF-}	D	A	AN3	AN2	1/2

Fonte: Microchip, 2003a, com adaptações.

LEGENDA:

x = bit desconhecido. **A** = entrada Analógica. **D** = I/O é Digital

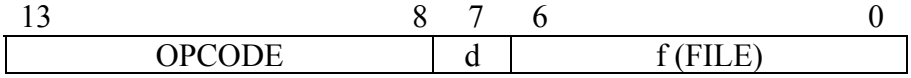
C/R = quantidade de canais analógicos / quantidade de entradas para referência de tensão A/D.

2.1.7 – As instruções *assembly* do PIC

Os recursos físicos dos microcontroladores geralmente são escassos. A linguagem *assembly* guarda uma relação direta com a linguagem de máquina ou com o código executável, ou seja, relação de uma para uma. Programar em *assembly* permite ao programador uma maior atenção ao consumo desses recursos escassos, de memórias e de pinos de I/O. Isto, aliado ao desenvolvimento de rotinas eficientes, pode ser caso de viabilizar ou não um projeto. Um exemplo simples de como otimizar um código fonte é, caso deseja gerar um atraso de dois períodos de máquina, não deve usar duas instruções NOP, que geram esses dois atrasos mas utilizam duas posições da memória de programa. O mesmo efeito pode ser conseguido utilizando

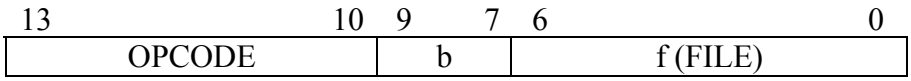
a instrução GOTO $+1. O $ assume o endereço onde reside a instrução GOTO que adicionado de um promove o salto para a instrução abaixo desse GOTO. Por isto, nesse projeto, dada a possibilidade e necessidade de otimização dos recursos físicos é que o microcontrolador PIC foi programado em linguagem *assembly*.

O PIC16F877A tem um conjunto (*set*) de 35 instruções *assembly*. Estas foram divididas em instruções (a) orientadas a byte; (b) orientadas a bit e (c) operações com valores constantes (chamados pela Microchip de literais) e de controle (Microchip, 1997). Cada OPCODE e seus operandos são formados por uma palavra (*word*) de 14 bits. Esta estrutura de formação geral das instruções está delineada nas Fig. 8, 9, 10 e 11.



d = 0, o resultado da operação tem como destino o registrador W;
d = 1, o resultado da operação tem como destino o registrador F;
f = endereço direto do registrador F, na memória de dados, de 7 bits (Fig. 7)

Figura 8 – Formato das instruções orientadas a byte
Fonte: Microchip, 1997.



b = 3 bits para ponteiro dos bits dentro do registrador F.
b3 b2 b1: 000 está apontando para o bit b0 do registrador F,
001 está apontando para o bit b1 do registrador F,
e assim por diante até, **b3 b2 b1**: ser 111 (aponta para b7).
f = endereço direto do registrador F, na memória de dados, de 7 bits (Fig. 7)

Figura 9 – Formato das instruções orientadas a bit
Fonte: Microchip, 1997.

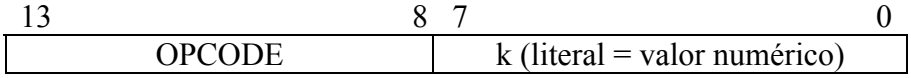


Figura 10 – Formato das instruções com literais e para controle
Fonte: Microchip, 1997.

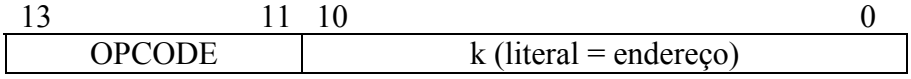


Figura 11 – Formato das instruções CALL e GOTO:
Fonte: Microchip, 1997.

Foi mostrado nas Fig. 8 a 11 a estrutura de formação da linguagem de máquina, estrutura dos bits do PIC. Como será mostrado a Microchip definiu 35 instruções para o PIC16F877A. Cada instrução utiliza uma das estruturas apresentadas nessas figuras. A linguagem *assembly* dá à instrução um conjunto de letras que ajuda a lembrar o que ela executa. Este conjunto de letras foi denominado por mnemônico. Então, cada instrução usa uma das estruturas de agrupamento dos bits e tem um mnemônico identificador da instrução (PEREIRA, 2006).

Para entender a lógica adotada para construção dos mnemônicos do PIC, deve-se reconhecer o significado de algumas letras utilizadas, como: W, diz respeito ao registrador *Work* (acumulador em outras tecnologias); F, de *File*, diz respeito, genericamente, a algum dos registradores especiais ou mesmo aos definidos pelo programador, sendo que quando for minúscula (f) refere-se ao conteúdo do registrador F; L, de *Literal*, refere-se a um valor numérico que fará parte da operação realizada pela instrução; d (minúsculo) é o lugar de destino onde o resultado da operação será armazenado – se d=0 implica que o resultado da operação será armazenado em W e se for d=1 será armazenado no registrador F indicado na própria instrução; B, de *Bit* – quando maiúsculo, o B compõe o mnemônico da instrução e quando minúsculo, o b (3 bits, apontam para um dos bits, de b0 a b7, de um registrador F) refere-se a um bit específico dentro do registrador especificado na própria instrução e, portanto, b é um dos argumentos da instrução; T, de *Test*, é usado no mnemônico de algumas instruções e indica que a instrução

realiza um teste em um bit específico num determinado registrador; S, de *Skip* (salto) ou *Set* (usado para forçar o valor de um bit para 1) - aqui o fabricante usou a mesma letra com dois sentidos diferentes, resta ao programador se acostumar com esses tipos de instruções; C, de *Clear* – refere-se ao *reset* de um bit (ou fazê-lo passar ao nível lógico baixo, ou zero); e, por fim, Z, de *Zero* – caso o resultado de uma operação seja zero, então o bit Z, do registrador STATUS, assume o nível lógico alto, ou um (lembrando que algumas instruções promovem desvios caso esta condição seja verdadeira) (SOUZA, 2005; Microchip, 2001).

Além dos significados da letras mostradas acima, precisa-se conhecer outras siglas que participam da formação dos mnemônicos, conforme apresentado na Tab. 12 (SOUZA, 2005).

Tabela 12 – Siglas que compõem o mnemônico das instruções *assembly*

Sigla	Descrição
ADD	Refere-se a uma soma aritmética;
AND	Refere-se a lógica E, bit a bit;
CLR	Zera o conteúdo de um registrador, ou reseta-o;
COM	Complementa um registrador, ou inverte-o bit a bit;
DEC	Decrementa um registrador, ou subtrai 1 do conteúdo desse registrador;
INC	Incrementa um registrador, ou soma 1 ao conteúdo desse registrador;
IOR	Refere-se a lógica OU, bit a bit;
MOV	Move, ou melhor, copia um valor para algum registrador;
RL	Roda 1 bit à esquerda (*Rotation Left*);
RR	Roda 1 bit à direita (*Rotation Right*);
SUB	Refere-se a uma subtração aritmética;
SWAP	Inversão do agrupamento de 4 bits (*nibble)*, menos significativos, com o *nibble* mais significativo e vice-versa, de um mesmo registrador;
XOR	Refere-se a lógica OU EXCLUSIVO; bit a bit;

O PIC16F877A foi construído suportado por uma arquitetura *Reduced Instruction Set Computer* (RISC) que reconhece apenas 35 instruções *assembly*, as quais são mostradas nas Tab. 13 a 16 (Microchip, 1997).

A Tab. 13 mostra as instruções que operam ao nível de bit. Estas instruções podem resetar, setar ou testar se um bit é zero ou um, e neste caso, sendo verdadeiro o teste, promove um salto, ou seja, salta a próxima instrução. Este tipo de salto, portanto, é um salto condicional.

Tabela 13 – Instruções do PIC orientadas a bit

Mnemônico	Operandos	Operação Simbólica	Ciclos de máquina	14 bits de código MSB LSB	Flags afetados
BCF	f,b	Reseta o bit b do reg. f	1	01 00bb bfff ffff	-
BSF	f,b	Seta o bit b do reg. f	1	01 01bb bfff ffff	-
BTFSC	f,b	Testa o bit b do reg. f e salta a próxima instrução se ele for zero.	1	01 10bb bfff ffff	-
BTFSS	f,b	Testa o bit b do reg. f e salta a próxima instrução se ele for um.	1	01 11bb bfff ffff	-

Fonte: Microchip, 1997, com adaptações.

A Tab.14 mostra as instruções do PIC para vários fins, a exemplo de resetar o registrador W, chamar uma sub-rotina, resetar o registrador WDT, promover um salto incondicional, retornar de uma interrupção, retornar de uma sub-rotina com um valor constante em W, promover o retorno normal de uma sub-rotina e, também, tem uma instrução que coloca o microcontrolador no modo de economia de energia.

Tabela 14 – Instruções do PIC para vários fins

Mnemônico	Operandos	Operação Simbólica	Ciclos maquina	14 bits de código MSB LSB	Flags afetados
CLRW		W ← reseta W	1	00 0001 0000 0011	Z
NOP		*No Operation*, perde 1μs qdo. cristal for de 4MHz.	1	00 0000 0xx0 0000	
CALL	K	Chama sub-rotina pelo rótulo K.	2	10 0kkk kkkk kkkk	
CLRWDT		WDT ← reseta contador WDT.	1	00 0000 0110 0100	/TO, /PD
GOTO	K	Vá para endereço K ou rótulo K.	2	10 1kkk kkkk kkkk	
RETFIE		Retorna após atendimento a uma interrupção.	2	00 0000 0000 1001	
RETLW	K	Retorna de uma sub-rotina fazendo W ← K	2	11 01kk kkkk kkkk	
RETURN		Retorna de uma sub-rotina	2	00 0000 0000 1000	
SLEEP		Põe o PIC na condição de economia de energia, *mode standby*.	2	00 0000 0110 0011	/TO, /PD

Fonte: Microchip, 1997, com adaptações. x = irrelevante.

A Tab. 15 mostra as instruções que operam com constantes numéricas. Estas instruções operam o conteúdo presente no registrador W com outro conteúdo numérico encontrado na própria instrução, este designado pelo fabricante pela letra k. Então, encontra-se neste conjunto de instruções as que executam soma, operações lógicas, subtração. A carga do resultado das operações, nestes casos, é direcionada ao registrador W.

Tabela 15 – Instruções do PIC que operam com constante numérica

Mnemônico	Operandos	Operação Simbólica	Ciclos de máquina	14 bits de código MSB LSB	Flags afetados
ADDLW	k	W ← W + k	1	11 111x kkkk kkkk	C, DC, Z
ANDLW	k	W ← W AND k	1	11 1001 kkkk kkkk	Z
IORLW	k	W ← W OR k	1	11 1000 kkkk kkkk	Z
MOVLW	k	W ← k	1	11 00xx kkkk kkkk	-
SUBLW	k	W ← k – W	1	11 110x kkkk kkkk	C, DC, Z
XORLW	k	W ← W XOR k	1	11 1010 kkkk kkkk	Z

Fonte: Microchip, 1997, com adaptações. x = irrelevante.

A Tab. 16 mostra as instruções que operam ao nível de byte. Estas instruções podem somar, subtrair, realizar operações lógicas, decrementar e incrementar testando ou não se resultou zero, e, caso resulte em zero, promover um salto; carregar um registrador com um byte, e, também, deslocar os bits de um byte, um bit por vez, à esquerda ou à direita.

Tabela 16 – Instruções do PIC orientadas a byte

Mnemônico	Operandos	Operação Simbólica	Ciclos de máquina	14 bits de código MSB LSB	Flags afetados
ADDWF	f,d	d ← W + f	1	00 0111 dfff ffff	C, DC, Z
ANDWF	f,d	d ← W AND f	1	00 0101 dfff ffff	Z
CLRF	f	F ← reseta f	1	00 0001 1fff ffff	Z
COMF	f,d	d ← complementa f	1	00 1001 dfff ffff	Z
DECF	f,d	d ← f – 1	1	00 0011 dfff ffff	Z
DECFSZ	f,d	d ← f – 1 e salta próxima instrução se esta subtração resultou em zero.	1 qdo não pula; 2 qdo pula.	00 1011 dfff ffff	-
INCF	f,d	d ← f + 1	1	00 1010 dfff ffff	Z
INCFSZ	f,d	d ← f + 1 e salta próxima instrução se esta soma resultou em zero.	1 qdo não pula; 2 qdo pula.	00 1111 dfff ffff	-
IORWF	f,d	d ← W OR f	1	00 0100 dfff ffff	Z
MOVF	f,d	d ← f	1	00 1000 dfff ffff	Z
MOVWF	f	f ← W	1	00 0000 1fff ffff	-

RLF	f,d	Rotaciona o reg. f um bit à esquerda, com o *carry*.	1	00	1101	dfff	ffff	C
RRF	f,d	Rotaciona o reg. f um bit à direita, com o *carry*.	1	00	1100	dfff	ffff	C
SUBWF	f,d	d ← f – W	1	00	0010	dfff	ffff	C, DC, Z
SWAPF	f,d	Troca de *nibbles* em f.	1	00	1110	dfff	ffff	-
XORWF	f,d	d ← W XOR f	1	00	0110	dfff	ffff	Z

Fonte: Microchip, 1997, com adaptações.

Foi apresentado, sucintamente, as instruções que o PIC16F877A reconhece e executa. O bom entendimento dessas instruções é o primeiro passo para poder programar este microcontrolador em linguagem *assembly*.

Na sequência foi tratado dos tópicos especializados, ou sejam: (a) dos recursos do PIC que suportam o protocolo I^2C; (b) dos recursos do PIC que suportam a USART.

2.1.8 – Suporte ao protocolo I^2C

A memória utilizada nesse projeto, externa ao PIC, para armazenar a massa de dados gerada pelas leituras do eletrômetro foi a memória serial 24LC1024 (EEPROM). A leitura e escrita nesta memória utiliza o protocolo I^2C, por isto é necessário o entendimento deste protocolo.

O PIC16F877A suporta a comunicação serial I^2C através do módulo *Master Synchronous Serial Port* (MSSP) interno ao microcontrolador. Associado a este módulo, trabalham os registradores SSPSTAT (registrador de *status*, ou estado); dois registradores de controle, SSPCON1 e SSPCON2; o registrador *Serial Receive/Transmit Buffer Register* (SSPBUF); o registrador *MSSP Shift Register* (SSPSR) que não é acessível ao programador; e o registrador *MSSP Address Register* (SSPADD). As diversas configurações possíveis para os bits destes registradores diferem suas aplicações, podendo o módulo MSSP operar no modo *Serial Peripheral Interface* (SPI), este não utilizado nesse projeto, ou no modo I^2C.

Os PICs são dispositivos ora orientados a bit, ora a byte. Como já se disse, cada registrador armazena 1 byte, está associado a um endereço em um dos bancos da memória RAM

e é conhecido por um nome, os quais (nome ou endereço) podem ser empregados na programação do dispositivo. Cada um dos bits desses registradores tem, também, um nome de chamada, ou de acesso, que pode assumir o valor 0 ou 1. Estes bits podem ser alterados, individualmente ou todo o byte, pelo próprio programa e, às vezes, pelo hardware, dependendo da situação em execução. A Tab. 17 mostra estes registradores do PIC16F877A e seus bits participantes do I^2C.

Tabela 17 – Registradores do PIC16F877A associados à implementação I^2C

Nome do Registrador	Bit 7 (b7)	Bit 6 (b6)	Bit 5 (b5)	Bit 4 (b4)	Bit 3 (b3)	Bit 2 (b2)	Bit 1 (b1)	Bit 0 (b0)
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
SSPSTAT	SMP	CKE	D/A	P	S	R/W	UA	BF
SSPCON1	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
SSPADD	Ajusta velocidade: SSPADD ← 0x09, para 100kbps, com cristal de 4MHz.							
SSPBUF	Buffer onde dado é carregado para TX e RX							
SSPSR	Registrador inacessível.							

Endereços associados aos nomes dos registradores:
SSPSTAT = 0x094 (*Bank* 1) // SSPCON = 0x014 (*Bank* 0) // SSPCON2 = 0x091 (*Bank* 1)
SSPADD = 0x093 (*Bank* 1) // SSPBUF = 0x013 (*Bank* 0) // SSPSR = inacessível

2.1.8.1 – Registradores associados ao protocolo I^2C

O módulo MSSP, interno ao PIC, pode ser configurado para trabalhar como I^2C ou como SPI, modo não utilizado nesse projeto. Os blocos funcionais de controle desse módulo, possibilitam a comunicação do PIC com diversos tipos de periféricos. Como exemplo o PIC pode comunicar-se com outro PIC, com memórias seriais externas, com *drivers* de LCD, com relógio-calendário, com sensores, etc. (Souza, 2006). A comunicação com a memória serial EEPROM 24LC1025, leitura e escrita, foi controlada via interface I^2C. O I^2C suporta trabalhar como (a) Modo *Master*, ou mestre, que será focado aqui, (b) Modo *Multi-Master*; e (c) Modo *Slave*, ou escravo. (Microchip, 2003).

Os registradores SSPSTAT, SSPCON1 e SSPCON2 são registradores para controle e estado do módulo MSSP. Os bits do registrador SSPSTAT, registrador do estado do MSSP, podem ter significados diferentes no modo SPI e no I^2C. O interesse desse projeto reside no I^2C, conforme apresentado nos próximos itens.

2.1.8.1.1 – O registrador SSPSTAT

A Tab. 18 mostra o registrador SSPSTAT do módulo MSSP. O nível lógico de cada um dos bits deste registrador fornece o estado que o módulo MSSP apresenta, quando configurado para I^2C.

Tabela 18 – Registrador SSPSTAT – modo I^2C

Registrador: **SSPSTAT**				Endereço: **94h**			
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
R/W	R/W	R	R	R	R	R	R
SMP	CKE	D/A’	P	S	R/W’	UA	BF
Condição após *reset* por *Power-On Reset* (POR)							
0	0	0	0	0	0	0	0

Fonte: Microchip, 2003a, com adaptações.

SMP Bit de controle da taxa de variação (*Slew Rate*) (I^2C):
0 = O controle do sinal de dados (taxa de variação) está habilitado para o modo alta velocidade (400kHz);
1 = O controle do sinal de dados (taxa de variação) está desabilitado para o modo velocidade padrão (100kHz e 1 MHz);

CKE Sistema de amostragem de dados (I^2C):
0 = Entradas conforme especificações I^2C;
1 = Entradas conforme especificações SMBUS;

D/A’ Bit para indicação de Dados ou Endereço (I^2C), para Modo *Slave*:
0 = Último byte recebido ou transmitido era um endereço;
1 = Último byte recebido ou transmitido era um dado;

P Indicação do bit *Stop* (I^2C):

0 = Não foi detectado o bit *Stop*;

1 = Foi detectado o bit *Stop*;

S Indicação do bit *Start* (I^2C):

0 = Não foi detectado o bit *Start*;

1 = Foi detectado o bit *Start*;

R/W' Bit que informa se é uma operação de Leitura/Escrita (I^2C):

Modo *Slave* 0 = Escrita;

1 = Leitura;

Modo *Master* 0 = Nenhuma transmissão em andamento;

1 = Transmissão em andamento. É zerado, por hardware, ao final da execução (SOUZA,2006).

UA Indicação de atualização de endereço (I^2C, Modo *Slave* a 10 bits):

0 = Não precisa atualizar o endereço;

1 = Indica que o programador deve atualizar o endereço no registrador SSPADD;

BF Este bit indica o estado de Buffer (Registrador SSPBUF) cheio (I^2C):

Modo de Transmissão 0 = Recepção ainda não terminou. Buffer SSPBUF vazio;

1 = Recepção complete. Buffer, SSPBUF**,** cheio;

Modo de Recepção 0 = Transmissão de dados terminou. Buffer SSPBUF vazio;

1 = Transmissão em andamento. Buffer SSPBUF ainda cheio.

Obs.: A Microchip observa que os bits P e S do registrador SSPSTAT serão zerados quando ocorrer:

(a) qualquer dos Resets do microcontrolador,

(b) quando for zerado o bit SSPEN do registrador SSPCON1.

2.1.8.1.2 – O registrador SSPCON1

A Tab. 19 mostra o registrador SSPCON1. Este registrador, juntamente com o SSPCON2, realizam o controle do módulo MSSP quando configurado para I^2C.

Tabela 19 – Registrador SSPCON1

Registrador: **SSPCON1**				Endereço: **14h**			
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
Condição após *reset* por *Power-On Reset* (POR)							
0	0	0	0	0	0	0	0

Fonte: Microchip, 2003a, com adaptações.

WCOL Colisão na escrita do registrador SSPBUF:

Master 0 = Sem colisão.

1 = Tentativa de escrita, sem permissão, no registrador SSPBUF. Deve ser zerado por comando de software.

Slave 0 = Sem colisão.

1 = Ocorreu uma escrita no registrador SSPBUF enquanto ainda estava transmitindo. Deve ser zerado por comando de software.

SSPOV Bit que indica estoura (*Overflow)* na recepção, o estado deste bit, na transmissão, é irrelevante (pode ser 0 ou 1) :

0 = Sem *Overflow*.

1 = Um byte foi recebido antes do registrador SSPBUF ser lido (deve ser zerado por software).

SSPEN Habilitação da porta *Synchronous Serial Port* (SSP) (I^2C):

0 = SSP desabilitada. Pinos RC3 e RC4 configurados como I/O normais;

1 = SSP habilitada. Pinos SCL e SDA configurados como *Serial Ports*;

CK**P** Controle do *Clock* (I^2C):

Modo *Slave* 0 = Mantém o *clock* em nível baixo (0);

1 = *Clock* habilitado;

Modo *Master*, não é usado;

SSPM <3:0> Seleção do modo de trabalho da porta SSP:

SSPM<3:0>

1 1 1 1	=	I^2C Modo *Slave*, 10 bits de endereço com habilitação do bit de interrupção para *Start* e *Stop*.
1 1 1 0	=	I^2C Modo *Slave*, 7 bits de endereço com habilitação do bit de interrupção para *Start* e *Stop*.
1 0 1 1	=	I^2C Firmware controlado pelo *master* (*slave* disponível).
1 0 0 0	=	I^2C Modo *Master*, *clock* = F_{OSC} / [4*(SSPADD +1)].
0 1 1 1	=	I^2C Modo *Slave*, 10 bits de endereço.
0 1 1 0	=	I^2C Modo *Slave*, 7 bits de endereço.

Obs.: Outras combinações destes bits foram implementados no Modo SPI.

2.1.8.1.3 – O registrador SSPCON2

A Tab. 20 mostra o registrador SSPCON2. Este registrador, juntamente com o SSPCON1, realizam o controle do módulo MSSP quando configurado para I^2C.

Tabela 20 – Registrador SSPCON2

Registrador: **SSPCON2**				Endereço: **91h**			
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
Condição após *reset* por *Power-On Reset* (POR)							
0	0	0	0	0	0	0	0

Fonte: Microchip, 2003a, com adaptações.

GCEN Habilita uma chamada global (I^2C *Slave*):

0 = Endereço de chamada global é desativado;

1 = Habilita uma interrupção quando um endereço de chamada global (0000h) é recebido no registrador SSPSR;

ACKSTAT Bit de STATUS de reconhecimento (*acknowledge*) (I^2C *Master TX*):

0 = Bit ACK foi recebido do *Slave*;

1 = Bit ACK não foi recebido do *Slave*;

ACKDT Valor enviado como resposta, no bit ACK, após recepção (I^2C *Master RX*):

0 = *Not* ACK;

1 = ACK;

ACKEN Habilita a sequência ACK (I^2C *Master RX*):

0 = Sequência ACK desabilitada;

1 = Inicia sequência ACK nos pinos SDA e SCL e transmite bit de dados em ACKDT. É zerado automaticamente pelo hardware;

RCEN Este bit habilita a recepção de dados (I^2C *Master*):

0 = Recepção desabilitada;

1 = Habilita o modo receptor para I^2C;

PEN Gera o procedimento do bit *Stop* na linha TX (I^2C *Master*):

0 = Não gera o procedimento que gera um bit *Stop*;

1 = Inicia o procedimento que gera um bit *Stop* nos pinos SDA e SCL. É zerado automaticamente pelo hardware;

RSEN Gera uma condição de *Re-Start* na linha (I^2C *Master*):

0 = Condição de *Re-Start* desabilitada;

1 = Inicia condição *Re-Start* nos pinos SDA e SCL. É zerado automaticamente pelo hardware;

SEN Gera uma condição de *Start* na linha:

Modo *Master*:

0 = Condição START desabilitada;

1 = Inicia condição START nos pinos SDA e SCL. É zerado automaticamente pelo hardware;

Modo *Slave*:

0 = *Clock* estendido habilitado somente para *Slave* RX;

1 = *Clock* estendido habilitado para *Slave* RX e *Slave* TX;

2.1.8.1.4 – O registrador SSPADD

A Tab. 21 mostra o registrador SSPADD do módulo MSSP. Configurando o módulo MSSP para I^2C, modo *slave*, neste registrador conterá o endereço do dispositivo escravo. Configurando-o para I^2C, modo *master*, nos seus sete bits menos significativos conterá o valor de referência para velocidade de transmissão, ou *baud rate*, que é o caso desse projeto (Microchip, 2003a).

Tabela 21 – Registrador SSPADD

Registrador: **SSPADD**				Endereço: **93h**			
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Endereço I^2C (parte alta e baixa).							
Condição após *reset* por *Power-On Reset* (POR)							
0	0	0	0	0	0	0	0

Fonte: Microchip, 2003a, com adaptações.

2.1.8.1.5 – O registrador SSPBUF

A Tab. 22 mostra o registrador SSPBUF do módulo MSSP. O SSPBUF é um registrador *buffer* construído na saída/entrada do módulo MSSP que é conectada ao barramento de dados interno ao PIC. Este *buffer* é utilizado tanto na transmissão quanto na recepção de dados.

Tabela 22 – Registrador SSPBUF

Registrador: **SSPBUF**				Endereço: **13h**			
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Buffer utilizado na TX e RX.							
Condição após *reset* por *Power-On Reset* (POR)							
0	0	0	0	0	0	0	0

Fonte: Microchip, 2003a, com adaptações.

2.1.8.2 – O protocolo I^2C

O protocolo I^2C é um padrão de especificações técnicas que possibilita a comunicação entre dispositivos físicos distintos.

Ao nível físico este protocolo só utiliza dois fios, um para TX e RX de dados, denominado por SDA, e outro para transmissão do sinal de *clock*, que fornece o sincronismo para transmissão/recepção, denominado por SCL. Portanto, o I^2C é um protocolo síncrono.

Ao nível lógico o protocolo I^2C se apresenta na forma de um quadro de transmissão, que, sumariamente, é estruturado como:

(a) verificar que o barramento I^2C esteja livre, *idle*;

(b) enviar *Start* bit (S);

(c) enviar o endereço do dispositivo receptor juntamente com o bit R/W' (se este for 1 = leitura, se 0 = escrita), pode ser utilizado um dos formatos:

- TX a 7 bits de endereço dos *Slaves*: A6 a A0 + R/W'; ou
- TX a 10 bits de endereço dos *Slaves*: 11110 + A9 + A8 + R/W', (após este envio) aguarda receber um bit ACK, e (continua a enviar) A7 a A0;

(d) aguardar, após a transmissão de (c), por um bit de reconhecimento do correto recebimento pelo *slave* (bit ACK);

(e) enviar (se TX) ou espera por receber (se RX) um dado de 8 bits (D7 a D0);

(f) aguardar por outro bit de reconhecimento (bit ACK);

(g) enviar o *Stop* bit (P), este fecha o quadro. Isto é melhor visualizado nas Fig. 12 e 13.

O módulo MSSP, configurado para I^2C, pode trabalhar no modo *Master* ou no modo *Slave*. O registrador SSPCON1<3:0> é que configura este perfil de trabalho. O mais comum é o microcontrolador ser configurado para *master* (opção feita nesse projeto) e os demais dispositivos ligados ao MCU serem *slaves*. É o *master* que fornece o sinal de *clock* e controla todo o processo de transferência de dados, seja de *Master*-TX para *Slave*-RX ou *Slave*-TX para *Master*-RX. Em termos do endereçamento dos dispositivos *Slaves*, estes podem ser a 7 bits (podendo endereçar até 128 dispositivos *slaves* distintos) ou a 10 bits (1024 dispositivos *slaves*) (SOUZA, 2006).

Para utilizar o módulo MSSP do PIC16F877A, primeiramente deve-se configurá-lo, ou definir um perfil de trabalho para o mesmo. Isto é feito seguindo os passos:

(a) os pinos SDA (RC4) e SCL (RC3) devem ser configurados como entrada, isto é o mesmo que carregar o registrador TRISC<4:3> com o valor binário 11.
(b) Deve-se habilitar os pinos SDA e SCL para I^2C, isto é realizado setando o bit SSPEN (SSPCON1<5>).
(c) Deve-se resetar o bit CKE (SSPSTAT<6>) para que no barramento só contenha níveis segundo as especificações I^2C.
(d) Os bits SSPM3:SSPM0 (SSPCON1<3:0>) configuram o modo de funcionamento da interface I^2C. Para que o módulo MSSP trabalhe no modo *Master*, estes bits devem assumir o valor binário 1000.

Escolhido o modo *master* para operação, deve-se calcular o valor do conteúdo a ser carregado no registrador SSPADD, o que é feito utilizando a equação (3).

$$clock = F_{OSC} / [4*(SSPADD +1)] \quad (3)$$

Na equação (3), *clock* é igual *Baud Rate*, que é a velocidade de transmissão. Manipulando os termos da equação (3) de forma a isolar o termo SSPADD e trocando o termo *clock* por *Baud Rate*, chega-se à equação (4).

$$SSPADD = [F_{OSC} / (4* Baud\ Rate)] - 1 \quad (4)$$

Para um cristal de 4MHz e uma taxa de transferência (*Baud Rate*) de 100kbps o registrador SSPADD deve ser carregado com o valor 0x09, este encontrado com a solução da equação (4) (Microchip, 2000a; ZANCO, 2006).

A transmissão de um quadro é iniciada pelo envio do *start* bit. O módulo MSSP é quem gera o *start* bit. Ou seja, o MSSP mantém a linha de *clock* SCL em nível lógico alto e promove uma transição negativa, de 1 (um) para 0 (zero), do sinal na linha de dados, SDA. Isto é reconhecido pela arquitetura I^2C como o *start* bit.

A sinalização de um *stop* bit, ou bit que finaliza a transmissão de um quadro, ocorre quando o sinal de *clock*, na linha SCL, estiver em nível lógico alto e na linha de dados (SDA) ocorrer uma transição positiva, ou de 0 (zero) para 1 (um), do sinal na linha de dados, SDA. Isto é reconhecido pela arquitetura I^2C como *stop* bit.

Qualquer dispositivo conectado a um *master*, até mesmo outro dispositivo *master*, pode precisar de tempo para tratar, processar, os dados já recebidos. Neste caso, o dispositivo retardado, o mais lento, pode solicitar uma pausa na transferência de dados. A arquitetura I^2C reconhece como sinalização desta pausa quando o sinal de *clock* se mantém no nível lógico baixo (SOUZA, 2006).

As Fig. 12 e 13 mostram os quadros (linha de dados, SDA, e de *clock*, SCL) para leitura e escrita dos dispositivos *slaves* a 7 bits, e as Fig. 14 e 15 mostram os quadros para leitura e escrita dos dispositivos *slaves* a 10 bits de endereço.

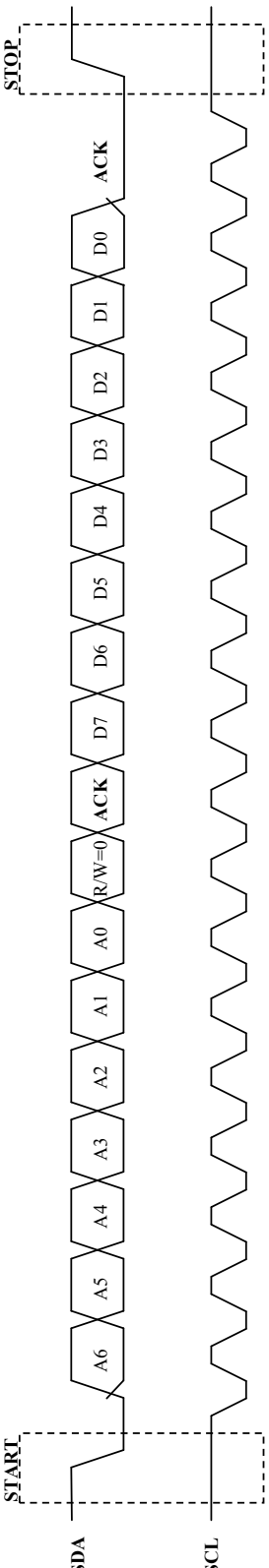


Figura 12 – Quadro de escrita no dispositivo *Slave* – a 7 bits de endereço.

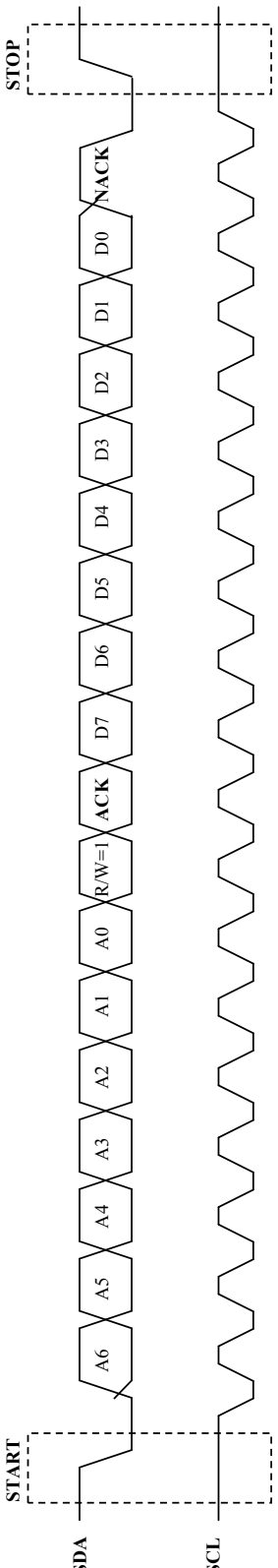


Figura 13 – Quadro de leitura no dispositivo *Slave* – a 7 bits de endereço.

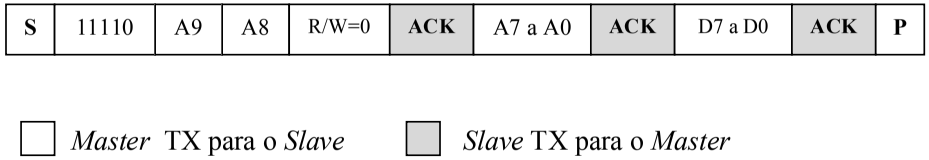


Figura 14 – Quadro de escrita no dispositivo *Slave* – a 10 bits de endereço.

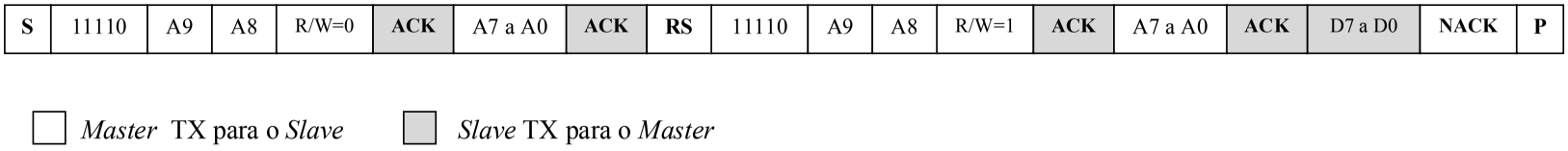


Figura 15 – Quadro de leitura no dispositivo *Slave* – a 10 bits de endereço.

O dispositivo *master* é a MCU, ela controla todo o processo de TX e RX de dados. Uma vez que o barramento esteja livre, o *master* inicia o processo de envio do *start* bit executando a sequência de instruções *assembly* (Microchip, 2000a):

```
BSF     SSPCON2,SEN     ; Seta o bit SEN do registrador SSPCON2
BTFSC   SSPCON2,SEN     ; Testa se o bit SEN foi zerado pelo hardware
GOTO    $-1             ; SEN ainda é um? mantém-se testando bit SEN.
```

As transmissões são feitas em blocos de um byte. Após cada byte transmitido a MCU espera receber do *slave* um bit ACK (*acknowledge*), ou de reconhecimento, validação, de ter recebido corretamente o byte enviado anteriormente. Após o envio do *start* bit a MCU envia o primeiro byte, que, nas Fig. 12 e 13, é composto pelos 7 bits de endereçamento do dispositivo *slave* (A6 a A0) acompanhado do bit R/W' (que quando for 0 implica ser um quadro de escrita, ou a MCU transmite o dado a ser gravado, ou escrito, no endereço A6 a A0 do *slave*; e quando for 1, o quadro é de leitura, e é o *slave* que transmitirá à MCU o dado (D_7 a D_0) previamente armazenado em seu endereço A6 a A0). Após o envio do bit menos significativo do primeiro byte (R/W'), a MCU espera pelo recebimento do bit ACK enviado pelo *slave*.

Recebido o primeiro ACK, se R/W' era 0, Fig. 12, a MCU inicia a transmissão do dado a ser escrito, gravado, no endereço enviado no primeiro byte. E, novamente, espera-se pelo recebimento de outro bit ACK, que, uma vez recebido corretamente, habilita a MCU (*master*) a fechar o quadro de transmissão, ou a controlar o processo de emissão do *stop* bit, executando a sequência de instruções *assembly* (Microchip, 2000a):

```
BSF     SSPCON2,PEN     ; Seta o bit PEN do registrador SSPCON2
BTFSC   SSPCON2,PEN     ; Testa se o bit PEN foi zerado pelo hardware
GOTO    $-1             ; PEN ainda é um? mantém-se testando bit PEN
```

As Fig. 14 e 15 mostram, em outro formato, os mesmos quadros, porém para endereçamento a 10 bits. A diferença é que para enviar 10 bits necessita-se de dois bytes. Então, em vez de só o primeiro byte ser para endereço, agora os dois primeiros bytes são para envio do endereço do dispositivo *slave*, preenchendo os bits faltantes para completar os 2 bytes pelo binário 11110.

A Fig. 15, leitura do dispositivo *slave*, é diferente da Fig. 14, escrita no dispositivo *slave*. Observa-se que antes da leitura, propriamente, faz-se a escrita do endereço, enviando R/W'=0.

Recebido o ACK do envio do endereço com R/W'=0 é enviado um bit de *Restart* (RS), que inicia, de fato, o quadro de leitura – agora com R/W'= 1. O *master* gera o *restart* seguindo a sequência *assembly* (Microchip, 2000a):

```
BSF     SSPCON2,RSEN    ; Seta o bit RSEN do registrador SSPCON2
BTFSC   SSPCON2,RSEN    ; Testa se o bit RSEN foi zerado pelo hardware
GOTO    $-1             ; RSEN ainda é um? mantém-se testando bit RSEN
```

Deve-se observar, também, que é o dispositivo *slave* quem envia os dados (D_7 a D_0) ao *master* (PIC), quando se realiza a leitura da memória 24LC1024 (*slave*).

Para cada byte, armazenado temporariamente no registrador chamado de DADO, que deve ser transmitido ao *slave*, a MCU deverá executar a sequência *assembly*: (Microchip, 2000a):

```
BCF     STATUS,RP0
BCF     STATUS,RP1      ; Aponta para o BANK0
MOVF    DADO,W          ; W ← DADO (dado a ser transmitido)
MOVWF   SSPBUF          ; SSPBUF ← W (carregue dado no SSPBUF)
BSF     STATUS,RP0
BCF     STATUS,RP1      ; Aponta para o BANK1
BTFSC   SSPSTAT,R_W     ; Testa se a transmissão continua em andamento
GOTO    $-1             ; Se R_W ainda é um, mantém-se testando bit R_W.
BCF     STATUS,RP0
BCF     STATUS,RP1      ; Volta a aponta para o BANK0.
```

Uma rotina funcional para verificar que o barramento está *Idle*, ocioso, é a sequência *assembly* (Microchip, 2000a):

```
BSF     STATUS,RP0
BCF     STATUS,RP1      ; Aponta para o BANK1
BTFSC   SSPSTAT,R_W     ; Se bit R_W =1, TX ou RX de dados em andamento
GOTO    $-1             ; Só sai deste loop quando R_W =0
MOVF    SSPCON2,W       ; W ← SSPCON2
ANDLW   0x1F            ; W ← W AND 0x1F
BTFSS   STATUS,Z        ; Os conteúdos dos bits SEN, RCEN, PEN, RCEN e
                        ; ACKEN estão em nível lógico baixo?
GOTO    $-3             ; Não, volte três instruções.
BCF     STATUS,RP0
BCF     STATUS,RP1      ; Aponta para o BANK0.
```

A sequência de instruções *assembly*, executadas pelo *master*, para gerar um ACK é (Microchip, 2000a):

```
BSF     STATUS,RP0
BCF     STATUS,RP1              ; Aponta para o BANK1
BTFSC   SSPCON2,ACKSTAT         ; ACKSTAT = 0 → ACK ok
GOTO    $-1
BCF     STATUS,RP0
BCF     STATUS,RP1              ; Aponta para o BANK0
```

A sequência de instruções *assembly*, executadas pelo *master*, para gerar um NACK é (Microchip, 2000a):

```
BSF     STATUS,RP0
BCF     STATUS,RP1      ; Aponta para o BANK1
BSF     SSPCON2,ACKDT   ; Nível alto (1) será TX para um NACK.
BSF     SSPCON2,ACKEN   ; Habilita iniciar uma sequência ACK com nível(1)
                        ; que no caso se entende como um NACK.
```

A sequência de instruções *assembly*, executadas pelo *master*, para verificar se terminou um evento de leitura (Microchip, 2000a):

```
BSF     STATUS,RP0
BCF     STATUS,RP1      ; Aponta para o BANK1
BSF     SSPCON2,RCEN    ; Habilita uma recepção no modo master.
BTFSC   SSPCON2,RCEN    ; Enquanto bit RCEN = 1 testa novamente
GOTO    $-1
```

A apresentação supra, sobre o protocolo I^2C na modalidade *master*, é suficiente para suportar o desenvolvimento da rotina *assembly* para ler/escrever um byte em posições endereçadas adjacentes da EEPROM 24LC1025.

2.1.8.3 – Diagrama em blocos do MSSP

A Fig. 16 ilustra o módulo MSSP para I^2C *master*, onde a Microchip apresenta o diagrama em blocos deste módulo.

O barramento bidirecional de dados, interno ao PIC, está conectado ao registrador SSPBUF, que é um *buffer* entre este barramento e o registrador de deslocamento SSPSR. O registrador SSPSR participa tanto da transmissão quanto da recepção de dados. Na transmissão recebe o dado na forma paralela do registrador SSPBUF e entrega-o ao pino SDA do PIC na forma serial. Na recepção de dados o SSPSR recebe o dado serial do pino SDA do PIC e entrega-o ao SSPBUF na forma paralela. Os demais blocos funcionais atuam como gerador de *baud rate*, do sinal de *clock*, da sequência de *start*, *stop* e ACK bits e na detecção de colisões, etc.

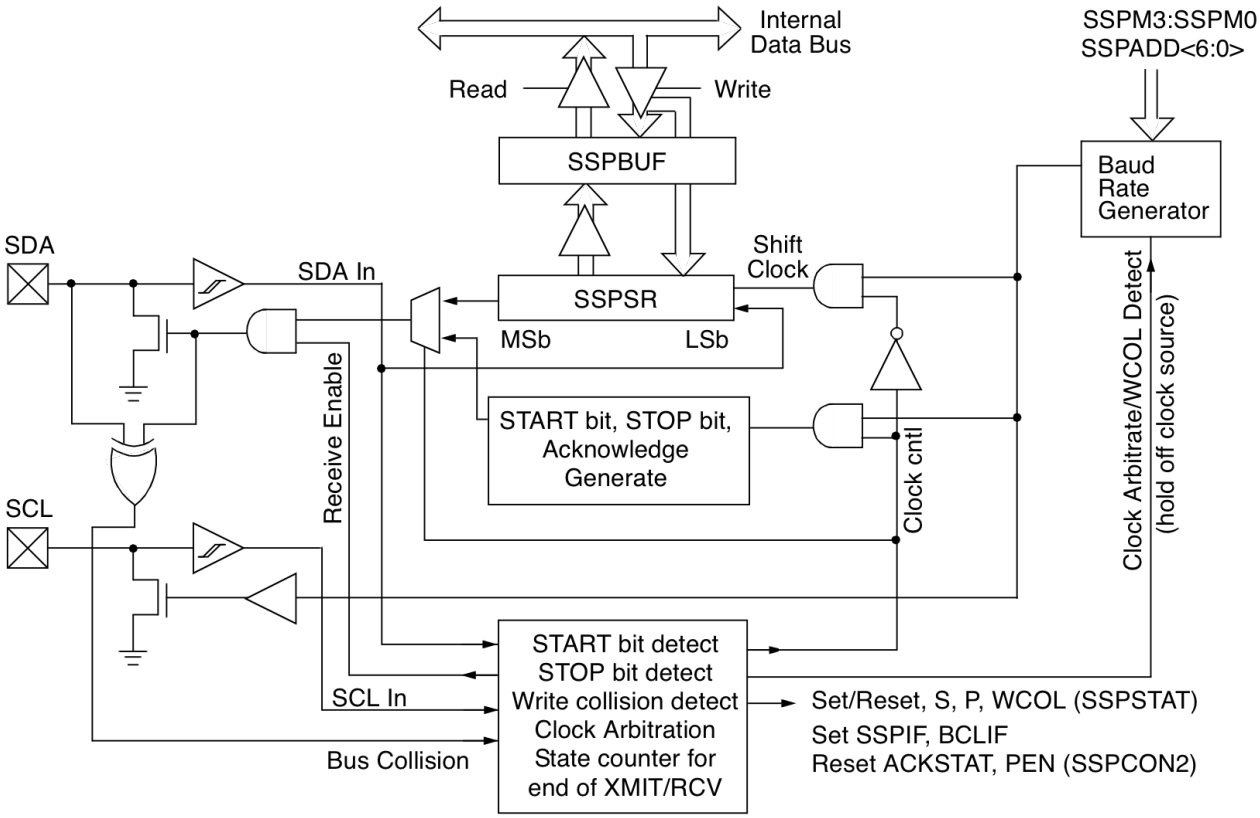


Figura 16 – Diagrama em blocos do modo I^2C *master*.
Fonte: Microchip, 2003a.

2.1.9 – Suporte à USART

O PIC16F877A tem, internamente, o módulo *Universal Syncronous Asyncronous Receiver Transmiter* (USART). Este módulo suporta tanto uma comunicação síncrona (que a transferência de dados acontece sob a regência de um sinal de *clock*, sincronismo entre as entidades comunicantes) quanto uma comunicação assíncrona (onde o sincronismo se dá pela duração de cada bit transmitido, ou seja, pela velocidade de transmissão adotada – em *bits per second* (bps) – com iguais valores em bps configurados tanto no transmissor quanto no receptor).

Foi utilizado nesse projeto a USART no modo assíncrono. Este modo utiliza o protocolo de linha *Start-Stop*, que é um protocolo de fácil utilização. O modo assíncrono, diferentemente do síncrono que utiliza protocolos de linha mais complexos, é um modo bastante utilizado e, por isso, não demandaria maiores desafios, estes deveriam residir nos pontos mais sensíveis à viabilização do projeto como a leitura e escrita na memória serial 24LC1025 (EEPROM), o controle dos motores de passo (várias tabelas para posicionamento da cabeça com a fonte radioativa rumo ao volume sensível da câmara de ionização) e a leitura do eletrômetro, etc.

2.1.9.1 – Possibilidades de uma transmissão

Além de uma transmissão poder ser síncrona (suportada por várias possibilidades de protocolos de linha) ou assíncrona (suportada pelo protocolo *Start-Stop*), como já se mencionou, entre dois ou mais dispositivos físicos que devem transmitir dados entre si, inicia-se pela forma de ligação física entre os envolvidos na comunicação. A forma de interligar estes dispositivos é designada por ligação ponto a ponto (onde um dispositivo A pode transmitir e receber dados de outro dispositivo B, que também pode transmitir e receber dados do dispositivo A) e por ligação multiponto, onde mais de dois dispositivos podem, segundo regras – protocolo – comunicarem entre si (SOARES, 1995).

Quanto ao uso do meio de transmissão, ele pode ser tomado por uma comunicação denominada por *simplex*, onde dois dispositivos interligados fisicamente, só um deles transmite dados ao outro; denominada por *half-duplex*, onde dois dispositivos A e B, ambos podem transmitir e receber ao outro, porém, não simultaneamente; e por fim, a comunicação pode ser

duplex ou *full-duplex*, onde dois dispositivos A e B, ambos podem transmitir e receber ao outro, simultaneamente (BARRADAS, 1981; SOARES, 1995).

Portanto, as possibilidade para realizar uma transmissão são: síncrona/assíncrona; ponto a ponto ou multiponto; *simplex*, *half-duplex* ou *duplex*.

2.1.9.2 – Modo assíncrono de transmissão de dados – *Start-Stop*.

O modo de transmissão de dados assíncrono, por não ter um sinal de *clock* para referência, para sincronismo, deve ser estruturado de forma que o dispositivo transmissor ou receptor possa identificar, sem erros, os dados enviados pelo outro dispositivo.

Primeiramente, deve-se configurar a mesma velocidade de transmissão (bps) nos dispositivos envolvidos na comunicação. Com isto, estes dispositivos conseguem detectar o início e fim de cada bit, dada a configuração inicial eles reconhecem o tempo necessário para transmitir e receber um determinado bit.

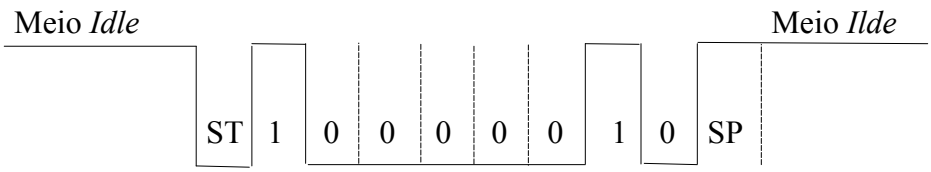
Além de determinar, por configuração, a duração de um bit, a transmissão assíncrona utiliza o protocolo *Start-Stop*. Ou seja, quando o meio de transmissão estiver ocioso (*idle*) ele estará em nível lógico alto. Uma vez ocioso, pode-se, então, iniciar a transmissão de um byte, o qual será "envelopado" pelo bit *Start*, no inicio, e pelo bit *Stop*, ao final. O bit *Start* é o envio de um bit em nível lógico baixo após o estado ocioso do meio de transmissão. Após o envio do *Start* bit deve-se enviar o byte, referente ao caractere a ser transmitido, ao dispositivo remoto. O bit menos significativo do caractere é o primeiro a ser transmitido. A transmissão pode acontecer sem envio do bit de paridade (bit *Start* + 8 bits de dados) ou com o envio do bit de paridade (bit *Start* + 8 bits de dados + bit de paridade). Neste caso a transmissão é a 9 bits. Após o envio dos bits de dados deve-se enviar o *Stop* bit, este, também configurado previamente em ambos dispositivos comunicantes, pode ter duração de 1; 1,5 ou 2 bits.

O bit de paridade, serve para verificação da ocorrência ou não de algum erro no byte transmitido. Os dispositivos podem ser configurados para trabalhar com a paridade impar ou com a paridade par. Com a paridade impar, se a quantidade de bits 1 do byte transmitido for par, neste caso, insere-se 1 ao bit de paridade. Assim, o somatório total dos bits 1, contando o 1 do bit de paridade, será impar. E se a quantidade de bits 1 no byte for impar, insere-se 0 ao bit de paridade,

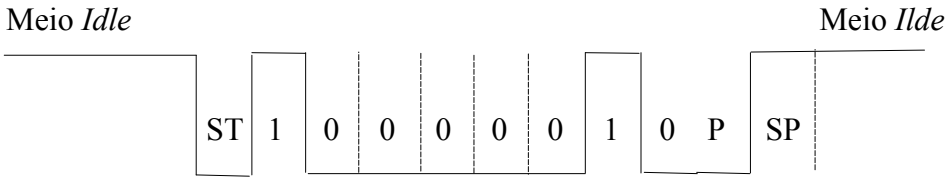
mantendo o somatório dos bits 1 em quantidade impar. Com a paridade par, o bit de paridade será 0 quando a quantidade de uns dentro do byte já for par, e será 1 quando esta quantidade for impar (SOARES, 1995).

Desta forma, no receptor, qualquer diferença entre a paridade configurada e a recebida é entendida como erro de transmissão. Mas, é um método pouco eficiente, pois dois bits simultaneamente podem mudar e dar a impressão de que o dado transmitido e recebido esteja correto. Outros métodos de detecção e correção de erros na transmissão, mais eficientes, poderiam ser empregados, mas aqui, foi trabalhado com *Start* + 8 bits + *Stop*, este com a duração de 1 bit. Não usou, nesse projeto, o bit de paridade.

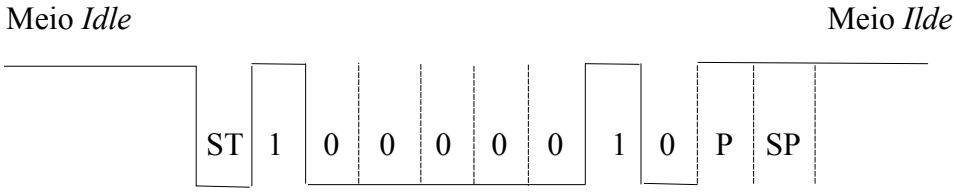
Chamando o *Start* bit por ST, o bit de paridade por P e o *Stop* bit por SP, a Fig. 17 mostra o formato para a transmissão assíncrona do caractere A do código ASCII, ou seja, seu valor 0x41. A Fig. 17 (a) mostra a transmissão sem paridade; a Fig. 17 (b) mostra a transmissão com paridade par, e a Fig. 17 (c) mostra a transmissão com paridade impar (SOARES, 1995).



(a) Transmissão caractere A, sem paridade.



(b) Transmissão caractere A, com paridade par.



(c) Transmissão caractere A, com paridade impar.

Figura 17 – Formas de onda da transmissão assíncrona

2.1.9.3 – Registradores do PIC relacionados à USART

A USART pode ser configurada para trabalhar no modo assíncrono *full-duplex*, o qual possibilita a comunicação da MCU com um computador pessoal, ou síncrono *half-duplex*, onde suporta a comunicação com periféricos como os CIs de relógio-calendário (ZANCO, 2005)

Os registradores associados à USART são o controle da transmissão (TXSTA); o controle da recepção (RCSTA); o registrador TXREG (*buffer* de transmissão); o registrador RCREG [é um duplo *buffer* de recepção, que trabalha sob o mecanismo *First In, First Out* (FIFO)]; e o registrador SPBRG, que é o gerador da velocidade de transmissão – *baud rate*. Estes registradores foram discriminados nas Tab. 23 a 27 com o significado de cada bit.

2.1.9.3.1 – O registrador TXSTA

O módulo USART é composto pelo registrador TXSTA, mostrado na Tab. 23. TXSTA é um registrador de *status* e controle da transmissão de dados.

Tabela 23 – Registrador TXSTA

Registrador: **TXSTA**				Endereço: **98h**			
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
R/W	R/W	R/W	R/W	U	R/W	R	R/W
CSRC	TX9	TXEN	SYNC	-	BRGH	TRMT	TX9D
Condição após *reset* por *Power-On Reset* (POR)							
0	0	0	0	0	0	1	0

Fonte: Microchip, 2003a, com adaptações.

CSRC Bit seletor da fonte de *clock*:
Modo Assíncrono: este bit é irrelevante;
Modo Síncrono:
0 = Modo Escravo (*Slave*);
1 = Modo Mestre (*Master*);

TX9 Habilitação a transmissão em 9 bits:
0 = Seleciona a transmissão a 8 bits;

1 = Seleciona a transmissão a 9 bits;

TXEN Habilitação da transmissão:

0 = Transmissão desabilitada;

1 = Transmissão habilitada. No modo síncrono os bits SREN/CREN do registrador RCSTA tem prioridade ao bit TXEN.

SYNC Seleção de um dos modos de transmissão, modo Assíncrono ou Síncrono:

0 = Modo Assíncrono;

1 = Modo Síncrono;

BRGR Seleção de alta taxa de transmissão (*Baud Rate*)

Modo Assíncrono:

0 = Baixa velocidade;

1 = Alta velocidade;

Obs.: No Modo Síncrono esse bit não tem função definida.

TRMT Status do registrador de deslocamento usado na transmissão (TSR):

0 = TSR cheio;

1 = TSR vazio;

TX9D Valor a ser transmitido como 9º bit de dados. Pode ser usado como bit de Paridade.

2.1.9.3.2 – O registrador RCSTA

O módulo USART é composto pelo registrador RCSTA, mostrado na Tab. 24. RCSTA é um registrador de *status* e controle da recepção de dados.

Tabela 24 – Registrador RCSTA

Registrador: **RCSTA**				Endereço: **18h**			
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
R/W	R/W	R/W	R/W	R/W	R	R	R
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
Condição após *reset* por *Power-On Reset* (POR)							
0	0	0	0	0	0	0	X

Fonte: Microchip, 2003a, com adaptações. X = Irrelevante.

SPEN Habilitação da porta serial - USART:

0 = USART desabilitada;

1 = USART habilitada – configurar os pinos RC7/RX/DT e RC6/TX/CK como pinos para a porta serial;

RX9 Habilitação da recepção a 9 bits:

0 = Recepção a 8 bits;

1 = Recepção a 9 bits;

SREN Habilita apenas uma recepção (só no modo síncrono na qualidade de mestre):

0 = Recepção unitária desabilitada;

1 = Recepção unitária habilitada. Após receber o dado, bit é resetado;

CREN Habilitação da recepção contínua:

0 = Recepção contínua desabilitada;

1 = Recepção contínua habilitada;

ADDEN Habilita detecção de endereço (só no modo Assíncrono a 9 bits – RX9 = 1):

0 = Desabilita detecção de endereço (todos os bytes são recebidos e o nono bit pode ser utilizado como bit de paridade);

1 = Habilita detecção de endereço, habilita interrupção e a carga do buffer de recepção quando *Receive (Serial) Shift Register* (RSR<8>) for um;

FERR Erro de frame (quadro) ou *Stop* bit (somente no modo Assíncrono):

0 = Não ocorreu de *frame* (quadro), *Stop* bit = 1;

1 = Ocorreu erro de *frame* (quadro). Ou o *Stop* bit = 0 (pode ser atualizado lendo o registrador **RCREG** e recebendo o próximo dado válido);

OERR Erro de transbordo (*overrun*) (bytes recebidos sem serem lidos);

0 = Não houve transbordo, ou não estourou o limite de recepção;

1 = Houve transbordo. OERR pode ser resetado quando reseta o bit **CREN**.

RX9D Valor recebido no nono bit de dados. Pode ser usado como bit de paridade, mas deve ser determinado pelo programa.

2.1.9.3.3 – O registrador TXREG

O registrador TXREG, mostrado na Tab. 25, é um *buffer* para armazenamento temporário do byte a ser transmitido. Este byte é recebido via barramento de dados (*data bus*) interno ao PIC e é repassado, todo o byte, ao registrador de deslocamento para a transmissão serial do caractere.

Tabela 25 – Registrador TXREG

Registrador: **TXREG**				Endereço: **19h**			
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Buffer de transmissão. Registrador para Transmissão de Dados.							
Condição após *reset* por *Power-On Reset* (POR)							
0	0	0	0	0	0	0	0

Fonte: Microchip, 2003a, com adaptações.

2.1.9.3.4 – O registrador RCREG

O registrador RCREG, mostrado na Tab. 26, é um duplo *buffer* para armazenamento temporário de até dois bytes recebidos. O byte recebido primeiro é entregue, também primeiramente (FIFO), ao barramento de dados (*data bus*) interno ao PIC.

Tabela 26 – Registrador RCREG

Registrador: **RCREG**				Endereço: **1Ah**			
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Buffer de recepção. Registrador para Recepção de Dados.							
Condição após *reset* por *Power-On Reset* (**POR**)							
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Fonte: Microchip, 2003a, com adaptações.

2.1.9.3.5 – O registrador SPBRG

A Tab. 27 mostra o registrador SPBRG. O registrador SPBRG deve se iniciado com um apropriado valor para *baud rate*, ou velocidade de transmissão.

Tabela 27 – Registrador SPBRG

Registrador: **SPBRG**				Endereço: **99h**			
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Ajusta a taxa de transmissão (*Baud Rate*).							
Condição após *reset* por *Power-On Reset* (POR)							
0	0	0	0	0	0	0	0

Fonte: Microchip, 2003a, com adaptações.

O Gerador de *Baud Rate* (BRG), ou da velocidade de transmissão a 8 bits (dimensionada em bps), pode atender tanto ao modo assíncrono quanto ao síncrono da USART. A Microchip fornece uma equação, Tab. 28, para o cálculo do valor a ser carregado no registrador SPBRG. Verifica-se, nesta equação, que o valor da velocidade de transmissão (*Baud Rate*) é função do bit BRGH (TXSTA<2>), do bit SYNC (TXSTA<4>), da freqüência do oscilador F_{OSC}, e do valor a ser carregado em SPBRG (X).

Tabela 28 – Cálculo do *Baud Rate*

SYNC	**BRGH** = 0 (baixa Velocidade)	**BRGH** = 1 (alta velocidade)
0	Modo Assíncrono *Baud Rate*=F_{OSC}/[64(X+1)]	Modo Assíncrono *Baud Rate*=F_{OSC}/[16(X+1)]
1	Modo Síncrono *Baud Rate*=F_{OSC}/[4(X+1)]	NA

X = valor entre 0 e 255 no registrador gerador de *baud rate* = SPBRG
Fonte: Microchip, 2003a, com adaptações.

Nesse projeto foi utilizada a transmissão com um *Baud Rate* (BR) de 4,8kbps e F_{OSC}=4MHz.. Para os bits SYNC = 0 e BRGH = 1, conforme Tab. 28 deve-se utilizar a equação (5).

$$BR=F_{OSC}/[16(SPBRG+1)] \quad (5)$$

Isolando o termo SPBRG, chega na equação (6).

$$SPBRG= [F_{OSC}/(16*BR)] -1)] \quad (6)$$

Calculando o valor de SPBRG chega-se ao valor decimal de 51, que é a parte inteira do cálculo de BR. Esse cálculo promove um erro, muito baixo, de apenas 0,16%, insuficiente para interferir negativamente na transmissão e recepção dos dados.

2.1.9.4 – USART – transmissão e recepção assíncrona

O pino 25 do PIC refere-se às possibilidade multiplexadas em RC6/TX/CK e o pino 26 em RC7/RX/DT. Os sinais RC6 e RC7 são usados como I/O digital, quando assim forem configurados. Serão designados por TX e RX quando forem configurados como USART no modo de transmissão assíncrono, e serão designados por CK e DT, ou *Clock* e Dados, quando forem configurados como USART no modo de transmissão síncrono. Como a USART foi trabalhada no modo assíncrono, os referidos pinos assumiram as designações TX e RX, respectivamente. Para configurar estes pinos como USART, no modo assíncrono, deve-se resetar o bit SYNC (TXSTA<4>) e setar o bit SPEN (RCSTA(<7>). Deve-se, também, configurar os pinos TX e RX como entradas, e isto é feito setando TRISC (7:6).

Além do exposto, para preparar o módulo USART para uma transmissão ou recepção de dados deve-se, antes, inicializar o registrador SPBRG com o valor decimal 51, conforme mostrado no item 2.1.9.3.5. Até o momento, um trecho de programa *assembly* que suporta a inicialização da USART no modo assíncrono, tanto para transmissão quanto para recepção, pode ser:

```
BSF     STATUS,RP0
BCF     STATUS,RP1      ; Aponta para o BANK1
BSF     TRISC,6
BSF     TRISC,7         ; Configura os pinos 25 e 26 do PIC p/ entrada.
BCF     TXSTA, 4        ; Zera o bit SYNC do registrador TXSTA.
MOVLW   .51             ; W ← .51
MOVWF   SPBRG           ; SPBRG ← W, configura baud rate para 4,8kbps.
BCF     STATUS,RP0
BCF     STATUS,RP1      ; Aponta para o BANK0
BSF     RCSTA, 7        ; Seta o bit SPEN do registrador TXSTA.
```

Uma vez inicializado o módulo USART, é interessante ter uma visão sobre os blocos que suportam a transmissão e recepção assíncrona. Uma vez que a USART pode ser configurada como *full-duplex*, os blocos dela, associados à transmissão e recepção, precisam ser distintos. As Fig. 18 e 19 ilustram estes módulos.

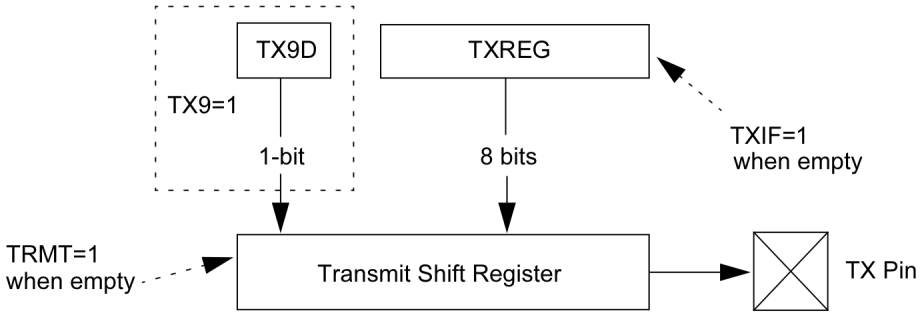


Figura 18 – Diagrama em blocos da transmissão assíncrona
Fonte: Microchip, 2000a.

A Fig. 18 ilustra, simplificadamente, os blocos envolvidos na transmissão assíncrona de dados. A transmissão pode acontecer a 8 ou a 9 bits. Para que a transmissão seja a 9 bits deve-se setar o bit TX9 do registrador TXSTA, que é o mesmo que configurar o dispositivo para transmitir a 9 bits. Observa-se que o nono bit refere-se ao bit de paridade. O valor a ser transmitido como nono bit é o valor presente na posição TX9D do registrador TXSTA, e cujo conteúdo deve ser carregado antes de carregar os 8 bits de dados no TXREG. Uma vez que todos os 9 bits foram carregados, estes são enviados para dentro do registrador de deslocamento (*Transmit Shift Register*) que começa a deslocá-los, a cada pulso de *clock*, bit a bit, para a saída

do PIC, pino 26, TX. Obviamente que, antes desse processo começar, é enviado um bit *Start*, e ao final um bit *Stop* (Microchip, 2000a).

Indicações importantes, presentes na Fig. 18, são as de registradores vazios. Quando o bit TXIF = 1 (do registrador PIR1) indica que o conteúdo do registrador TXREG foi transferido para o registrador de deslocamento, e, portanto, que TXREG está vazio, porém isto demora um ciclo de máquina, o que demanda uma rotina de *delay* semelhante a que se segue:

```
MOVLW     'A'         ; Carregue ASCII do caractere A em W.
MOVWF     TXREG       ; Transfere conteúdo de W para o registrador TXREG.
NOP                   ; TXIF não deve ser testado imediatamente após a carga
                      ; TXREG, atraso de 1 ciclo de máquina, no mínimo.
BTFSS     PIR1,TXIF   ; aguarde bit TXIF ser 1, antes de carregar novo byte.
GOTO      $-1         ; Espera TXREG ficar vazio.
```

Nesta condição pode-se carregar o registrador TXREG com novo byte a ser transmitido. Mas, isto não sinaliza que terminou a transmissão do byte carregado anteriormente. A indicação de que a referida transmissão terminou é fornecida pelo bit TRMT = 1 (do registrador TXSTA), que acontece no início da transmissão do *Stop* bit. Então, quando o bit TRMT estiver em nível lógico alto estará sinalizando que o registrador de deslocamento está vazio, ou que terminou uma transmissão (Microchip, 2000a).

Como a transmissão é totalmente separada da recepção, pode estar transmitindo um byte e recebendo outro, simultaneamente, sem problema algum, modo *full-duplex* ou *duplex*. Isto aumenta a taxa de *throughput* de dados, ou vazão (Microchip, 2000a).

A Fig. 19 ilustra o módulo USART, com seus blocos básicos, para a recepção de dados assíncronos. Igualmente à transmissão, a recepção pode ser a 8 ou 9 bits. Se for a 9 bits, isto deve ser configurado fazendo o bit RX9 = 1, no registrador RCSTA.

Após a recepção do *Start* bit, 8 ou 9 bits de dados entrantes pelo pino 26 do PIC (RX) são deslocados, bit a bit, dentro do registrador de deslocamento de recepção. Uma vez detectado o *Stop* bit, os dados recebidos são transferidos a uma estrutura de armazenamento temporário, em dois níveis, a qual trabalha sob o mecanismo *First In, First Out* (FIFO), ou o primeiro dado a entrar nesta pilha será, também, o primeiro dado a ser lido, a sair dela. Esta estrutura é composta por dois níveis de *buffer*, cuja saída dos dados se dá pelo registrador RCREG (Microchip, 2000a).

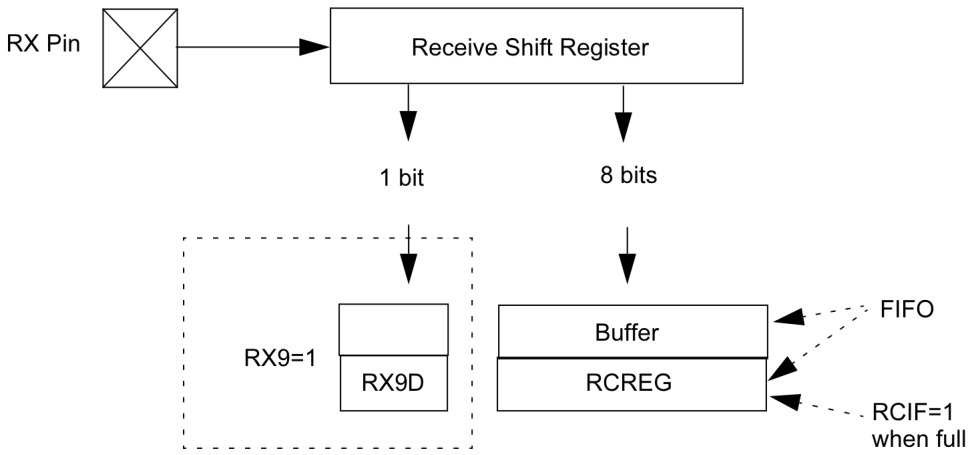


Figura 19 – Diagrama em blocos da recepção assíncrona
Fonte: Microchip, 2000a.

Deve-se garantir que o dado seja lido do registrador RCREG antes que um terceiro byte seja totalmente deslocado pelo registrador de deslocamento, pois caso não seja lido, desocupando recurso, este terceiro byte será descartado, ocorrendo, obviamente, um erro de recepção. Observa-se que se a comunicação for feita a 9 bits, o nono bits é repassado para o bit RX9D do registrador RCSTA, o qual, igualmente aos 8 bits de dados, o bit RX9D deve ser lido, também, antes que o terceiro byte e seu 9º (nono) bit sejam totalmente deslocados dentro do registrador de deslocamento.

Como só após o 3º byte recebido ser totalmente deslocado é que promoverá erro, perda de dados recebidos, isto é tempo (atraso) suficiente para que o software leia o byte presente no RCREG e evite a perda destes dados.

Na Fig. 19, quando o bit RCIF do registrador PIR1 estiver em nível lógico alto estará sinalizando que o dado se encontra disponível para ser lido no registrador RCREG, e também, causará uma interrupção, se esta estiver habilitada. RCIF só será zero quando todos os bytes tiverem sido lidos (Microchip, 2000a).

Será tratado, na sequência, de alguns pontos relevantes quanto a memória 24LC1025 e também quanto ao CI MAX232, este converte os sinais *Transistor-Transitor Logic* (TTL) da USART em sinais do padrão de transmissão da RS-232.

2.2 – A MEMÓRIA SERIAL 24LC1025

A memória serial 24LC1025 (EEPROM) é um circuito integrado com apenas oito terminais, cujo funcionamento é compatível com o protocolo de comunicação entre CIs, I^2C. É uma memória capaz de armazenar 128kBytes. Este CI é alimentado por uma fonte de tensão na faixa de 2,5 a 5,5V. Utilizou-se uma fonte de alimentação para CIs TTL, onde o terra (GND) foi ligado ao pino 4 (V_{SS}) e o +5Vcc ao pino 8 (V_{CC}). A freqüência máxima para o *clock* deste CI é de 400kHz. A Fig. 20 ilustra o *lay-out* externo desse dispositivo de memória e a Tab. 29 mostra os pontos mais relevantes no que tange aos oito pinos do CI 24LC1025.

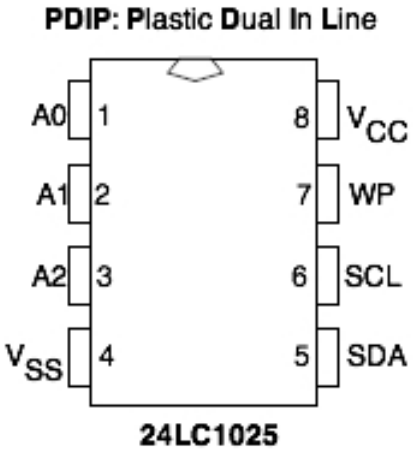


Figura 20 – *Lay-out* externo do CI 24LC1025
Fonte: Microchip, 2010.

Tabela 29 – Características dos pinos do CI 24LC1025

Designação	Nº pino	Função
A0	1	Juntamente com o sinal A1 seleciona o CI (*Chip Select*), possibilitando selecionar até 4 dispositivos 24LC1025.
A1	2	Idem ao pino A0.
A2	3	Pino de entrada, conforme informação do fabricante este pino deve ser ligado ao V_{CC}.
V_{SS}	4	Pino ligado ao terra (GND).
SDA	5	***S**erial **DA**ta*, pino bidirecional para TX e RX de dados.
SCL	6	***S**erial **CL**ok*.
WP	7	Entrada *Write-Protect*.
V_{CC}	8	Pino ligado ao +5Vcc da fonte de alimentação

Fonte: Microchip, 2010.

A linha SDA, Fig. 20 e Tab. 29, serve para TX e RX de dados entre o dispositivo *Master* e o *Slave*, não simultaneamente. Nesta linha de comunicação passa sinais de dados, de endereços e de controle. Deve-se ligar ao pino 5 (SDA) e ao pino 6 (SCL) um resistor *pull-up* tipicamente de 10kΩ, respectivamente, quando o *clock* for de 100kHz, e de 2kΩ quando for de 400kHz. O pino 6, entrada na memória, recebe o sinal de *clock* do *Master* (PIC). Este sinal serve para sincronizar a transferência de dados de/para o dispositivo 24LC1025. Em relação ao pino 7 (WP) deve-se ligá-lo ao terra para poder ler ou gravar um dado nesta memória. Ligando-o ao +5Vcc o dispositivo estará protegido contra gravação, mas a leitura acontece normalmente.

Deve-se observar que a transferência normal de dados se dá sempre quando o sinal de *clock* (SCL) estiver em nível baixo. Pois, quando há transição em SDA de 1 para 0 enquanto o sinal SCL estiver em nível alto isto equivale à seqüência de envio do *Start* bit. E quando há transição em SDA de 0 para 1 enquanto o sinal SCL estiver em nível alto, equivale à sequência de envio do *Stop* bit.

O dispositivo 24LC1025 pode ser lido ou escrito um dado de diferentes formas. Uma delas é a leitura/escrita byte a byte, que foi a escolhida para esse projeto. Outra forma é a gravação de um bloco de 128 bytes. A este bloco o fabricante chamou-o de página. Esta forma de gravação não atende ao propósito desse projeto de automação, motivo pelo qual não foi utilizada.

O I^2C é um protocolo *Start-Stop* síncrono, pois acontece sob a regência do sinal de *clock* (SCL). O *Start* bit indica o início do envio do quadro e o *Stop* bit o fim do quadro enviado, sendo que o sincronismo, como já foi dito, se dá pela linha SCL (*clock*). No próximo capítulo será explorado mais este tópico, dado que deverá ser aplicado como dispositivo armazenador de dados.

2.3 – O CI MAX232

Conforme o *datasheet* do CI MAX232, desenvolvido pela Texas Instruments, este CI é um conversor de sinais. Ele converte níveis lógicos TTL e, também, *Complementary Metal Oxide Semiconductor* (CMOS) em níveis *Eletronic Industries Association* (EIA)-232, ou *Recommended Standard* (RS)-232, e vice-versa, Tab. 30. A base para realizar esta conversão reside na ampliação da tensão por meio de capacitores (dado que a alimentação do CI é com +5Vcc) e, também, no uso de inversores de nível lógico alto em baixo e vice-versa, adequando os sinais TTL/CMOS ao RS-232 e vice-versa. A Fig. 21 mostra o *lay-out* externo desse CI. A esquerda está o seu *lay-out* com o nome de cada pino, e a direita o seu *Dual-in-line package* (DIP).

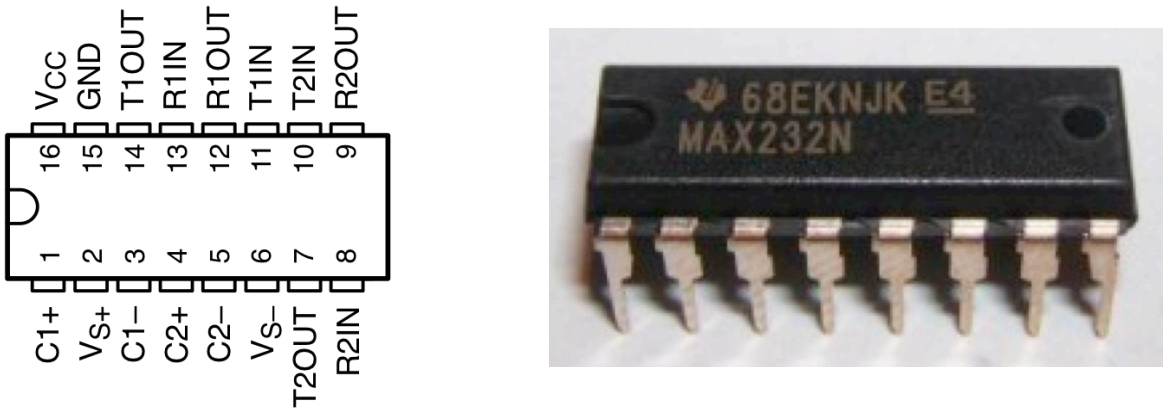


Figura 21 – Invólucro do CI MAX232
Fonte: Texas Instruments, 1989.

O PIC16F877A trabalha com níveis lógicos TTL, em lógica positiva, ou seja:

- o nível lógico baixo, é representado por ZERO (0) e seu potencial elétrico é próximo de zero volt;
- o nível lógico alto, é representado por UM (1) e seu potencial elétrico é próximo de +5Vcc.

O padrão de interfaceamento conhecido por RS-232 não sinaliza a linha de comunicação com os níveis de tensão TTL. Portanto, tais níveis de tensão devem, antes de serem direcionados à linha de comunicação, ser adaptados, ou convertidos, aos devidos níveis para transmissão. Esta é a função realizada pelo CI MAX232. A Tab. 30 mostra os níveis TTL, do PIC, e os níveis de linha RS-232.

Tabela 30 – Tensões TTL e RS-232

NÍVEL LÓGICO	TTL	RS-232
0	0 a +0,8V	+5V a +15V
1	+2 a +5V	-15V a -5V

Fonte: (TEXAS INSTRUMENTS, 1989; (ZANCO, 2006). Com adaptações.

Observa-se pela Tab. 30, que o padrão RS-232 representa o nível TTL baixo (0), por um potencial elétrico positivo e o nível TTL alto (1) por um potencial elétrico negativo. A recomendação padrão, RS-232, define uma faixa mais larga que a da Tab. 30, ou seja, a faixa de 3 a 25V (para 0, conhecido, também, como Espaço) e a faixa de -3 a -25V (para 1, conhecido, também, como Marca). Estes são os níveis limites da RS-232, (STRANGIO, 1993).

As Fig. 22 e 23 ilustram as formas de onda do caractere A, em ASCII e paridade par e impar, numa transmissão assíncrona via RS-232. Observa-se que o potencial elétrico de saída para marca e espaço deverá, para estar de acordo com o referido padrão de transmissão, estar entre os valores 3 e 25, sejam negativos ou positivos.

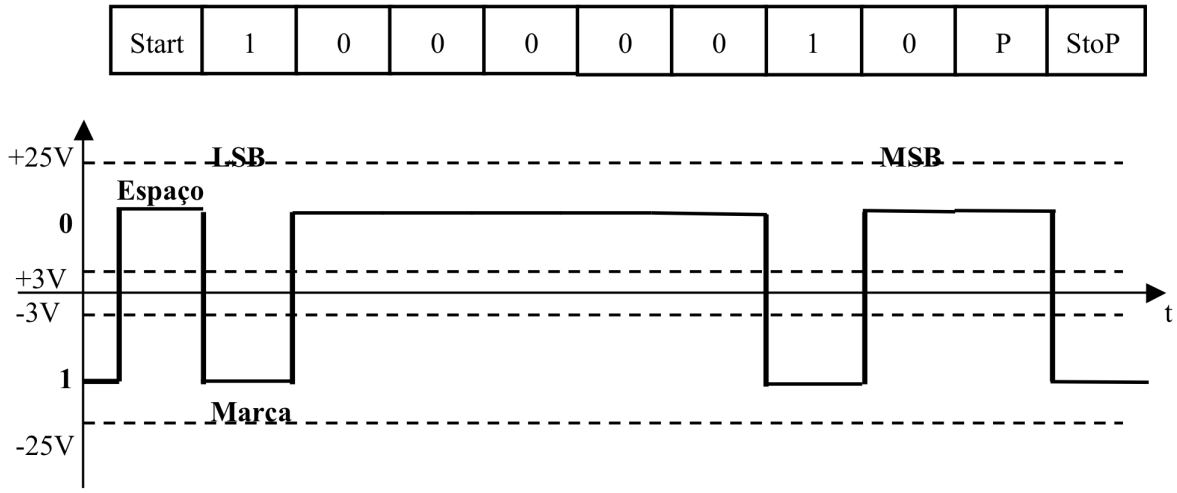


Figura 22 – Forma de onda da letra A, em ASCII, paridade par, na saída do MAX232
Fonte: Adaptações de STRANGIO, 1993.

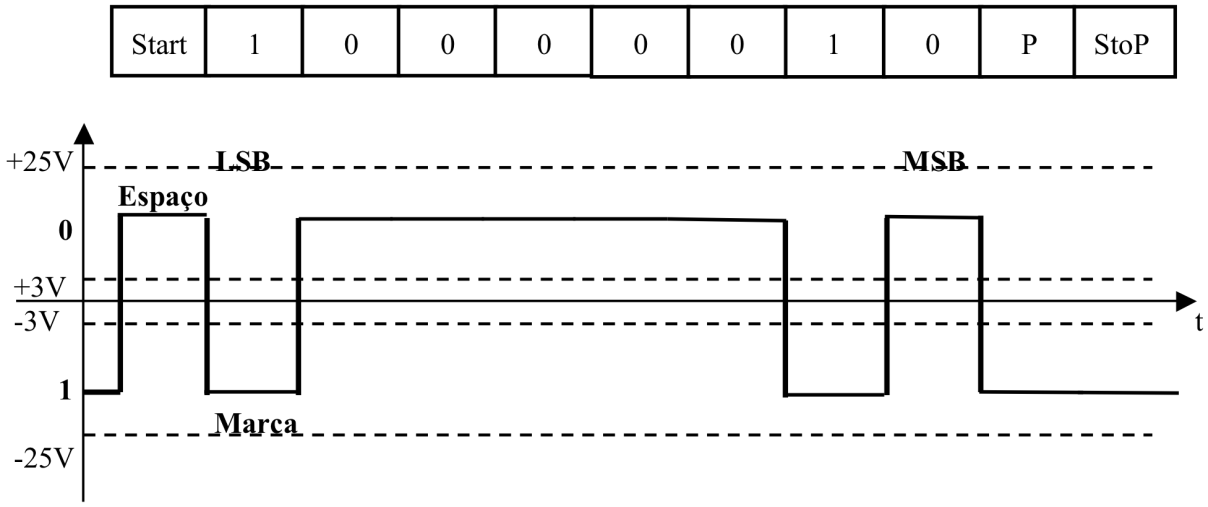
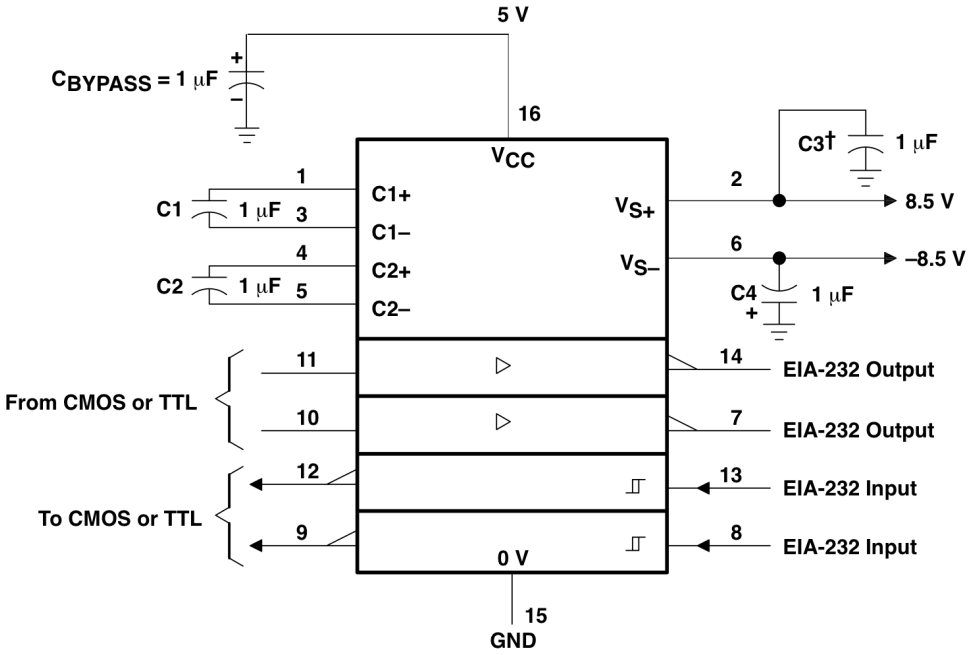


Figura 23 – Forma de onda da letra A, em ASCII, paridade impar, na saída do MAX232
Fonte: Adaptações de STRANGIO, 1993.

A Texas Instruments, no *datasheet* do CI MAX232, fornece uma ligação desse CI e isso facilita a sua utilização. A Fig. 24 mostra como ele deve ser ligado.



† C3 can be connected to V_{CC} or GND.

Figura 24 – Forma de ligação do MAX232
Fonte: Texas Instruments, 1989.

CAPÍTULO 3 – MATERIAIS E MÉTODOS

Nesse trabalho foi construído um sistema automatizado que controla o posicionamento entre uma fonte radioativa (residente em uma torre designada por X) e um detector, câmara de ionização (instalada em outra torre designada por Y). Uma vez que a fonte radioativa e o detector estejam posicionados, a interação da radiação com o volume sensível da câmara de ionização libera elétrons, os quais são atraídos para o devido pólo de uma fonte elétrica externa. Há correlação entre essas cargas liberadas e a dose absorvida pelo material a certa distância relativa entre X e Y (KNOLL, 1989).

Os terminais da câmara de ionização são ligados a um eletrômetro. O eletrômetro 616, utilizado nesse projeto, realiza a medida das cargas elétricas liberadas no volume sensível da câmara de ionização e mostra esta medida em seus *displays* de sete segmentos. O eletrômetro 616 é interconectado ao 6162, este disponibiliza o valor medido por sua interface digital. Esta interface fornece os dados da medida na forma BCD. Cada medida completa é composta por nove (9) *nibbles* BCD, que agrupados dois a dois em único byte ocupam 5 bytes da memória serial 24LC1025 (EEPROM), que foi a memória utilizada nesse projeto. Estes bytes somados a dois outros, que guardam o valor de X e de Y, compõem o que cada medida necessita, ou 7 (sete) posições endereçáveis da referida memória. Além desses valores serem armazenados na memória 24LC1025 eles também são mostrados, para acompanhamento, no *display* LCD.

O exposto condiz com a necessidade de controlar três motores de passo. Um deles para posicionar a torre X (que movimenta a fonte radioativa da posição -20,0 a +20,0 cm, a passos de 1,0 cm). Outro motor de passo para posicionar a torre Y (que movimenta a câmara de ionização da posição 2,0 a 92,0 cm, também a passos de 1,0 cm). Um terceiro, e último, motor de passo foi utilizado para direcionar o fluxo da fonte radioativa rumo ao centro geométrico do detector, câmara de ionização.

Duas interfaces de comunicação máquina-homem foram construídas. Uma delas, saída de dados pelo *display* LCD, seus circuitos e controle já foram publicados (FONSECA, 2008). A outra interface, apresentada nesse trabalho, seus circuitos e rotinas *assembly* foram construídos para transmitir os 3.731 resultados das medidas para um computador, via interface RS-232.

Dado que o LCD e os motores de passo, teorias e controles, já foram publicados (FONSECA, 2008) deve-se evidenciar, nesse trabalho, a leitura e escrita da memória serial

24LC1025 (EEPROM), a transmissão de dados via interface RS-232 ao computador (CP) e, ainda, ler a interface digital do eletrômetro. Estas são tratadas na sequência.

3.1 - Infraestrutura suporte ao desenvolvimento

Foi desenvolvido um projeto de automação baseado na MCU PIC16F877A. Este microcontrolador pode ser programado em linguagem *assembly*, de baixo nível, ou em linguagens de alto nível, a exemplo de programá-lo em linguagem C ou BASIC. Foi programado em linguagem *assembly*. Esta linguagem tem relação de um para um com o código executável e a MCU pôde ser programado otimizando seus recursos escassos. Isto foi fundamental, no que tange ao uso da memória de programa da MCU, para a viabilização desse desenvolvimento.

Um programa fonte em *assembly* (linguagem de montagem) deve ser convertido em linguagem de máquina (0s e 1s), programa executável, usando um programa *assembler* (montador), que é um tipo de programa tradutor. Após convertido para o código de máquina, este pode ser carregado na memória de programa do PIC (TANENBAUN, 2007).

A Microchip fornece em seu *site* (www.microchip.com) o programa para trabalhar com o PIC. Esse software é conhecido por MPLAB (para Windows, da Microsoft), o qual além de converter o programa fonte em programa executável, também simula a execução do programa e controla a gravação serial do programa executável diretamente ao PIC. Devido a esse agregado de funções num único software (MPLAB) é que faz dele um verdadeiro ambiente de gerenciamento de projetos.

Entre o computador, rodando o MPLAB, e o PIC é colocado o circuito conhecido por gravador de PIC. Este circuito é conectado ao computador por uma interface *Universal Serial Bus* (USB) e ao PIC por outra interface com os sinais necessários a gravação do dispositivo. A Fig. 25 ilustra o circuito do gravador do PIC *In-Circuit Debugger 2* (ICD 2) (MICROCHIP, 2002; MOSAICO, 2011). Este gravador é composto por duas PCIs e cabos de interconexões e é fornecido, no Brasil, pela Mosaico Indústria e Comércio Eletro Eletrônico Ltda. (www.labtools.com.br).

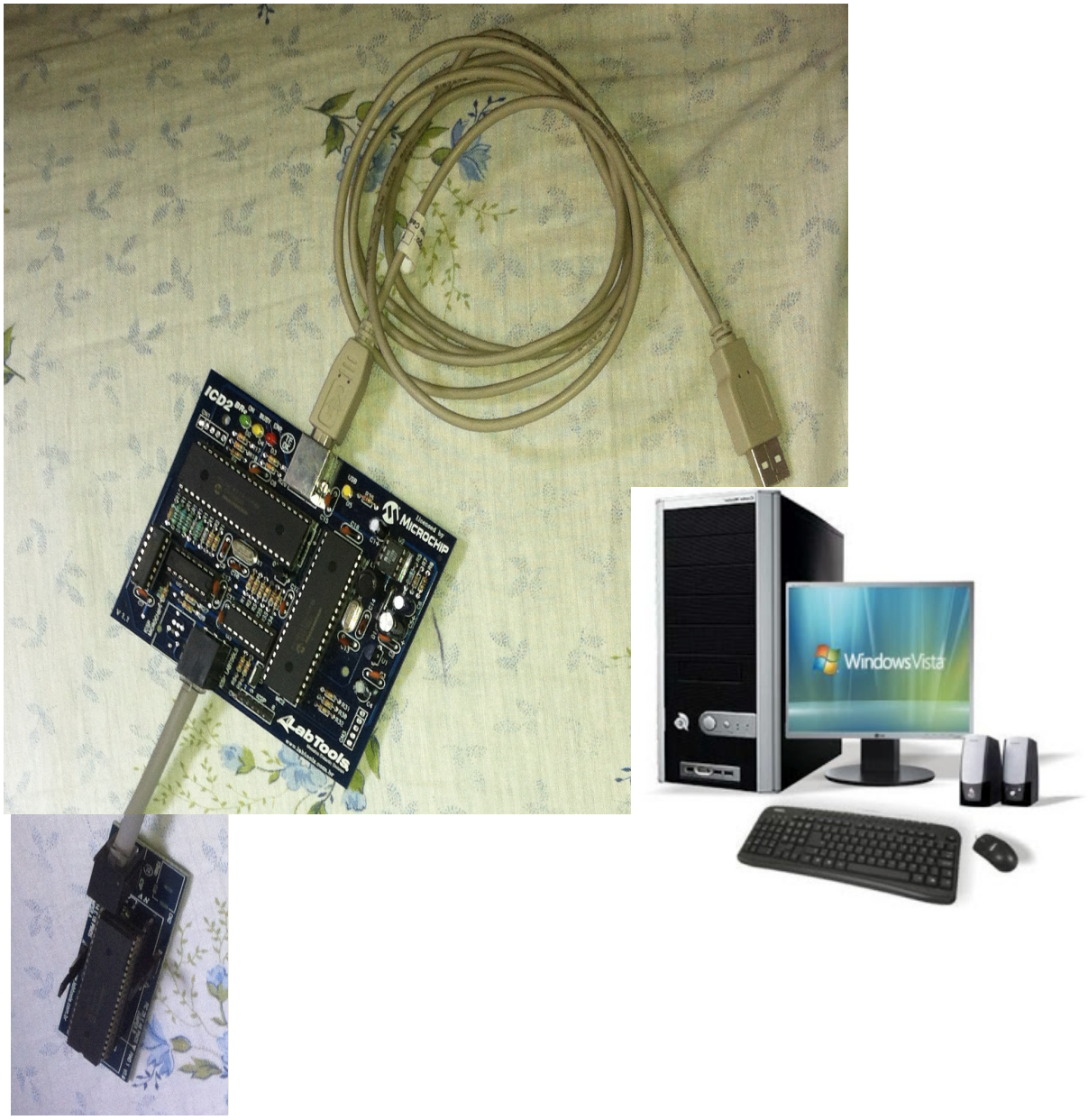


Figura 25 – Circuito auxiliar na gravação do PIC.

Além do MPLAB foi utilizado o software hyperterminal (htpe63.exe) (MICROSOFT, 2012). A informação para instalação e uso deste software é encontrada em (MICROSOFT, 2012a). Uma vez instalado e configurado o software hyperterminal no CP (ANEXO B), a interface RS-232 do CP pode ser conectada, via cabo específico, à entrada/saída RS-232 da PCI principal, a que contém o PIC. Esta conexão será vista quando se tratar das transferências de dados entre o PIC e o CP.

3.2 - Fluxograma global de funcionamento do projeto

Para desenvolver um sistema que controle e execute todas as etapas desse projeto, é preciso considerar o que foi apresentado no fluxograma da Fig. 26. Inicialmente deve-se configurar o microcontrolador PIC segundo os dispositivos a serem acionados ou lidos, a exemplo das portas de I/O devidamente configuradas, e entrar em um *loop,* numa rotina *assembly*, que execute as 3.731 medidas.

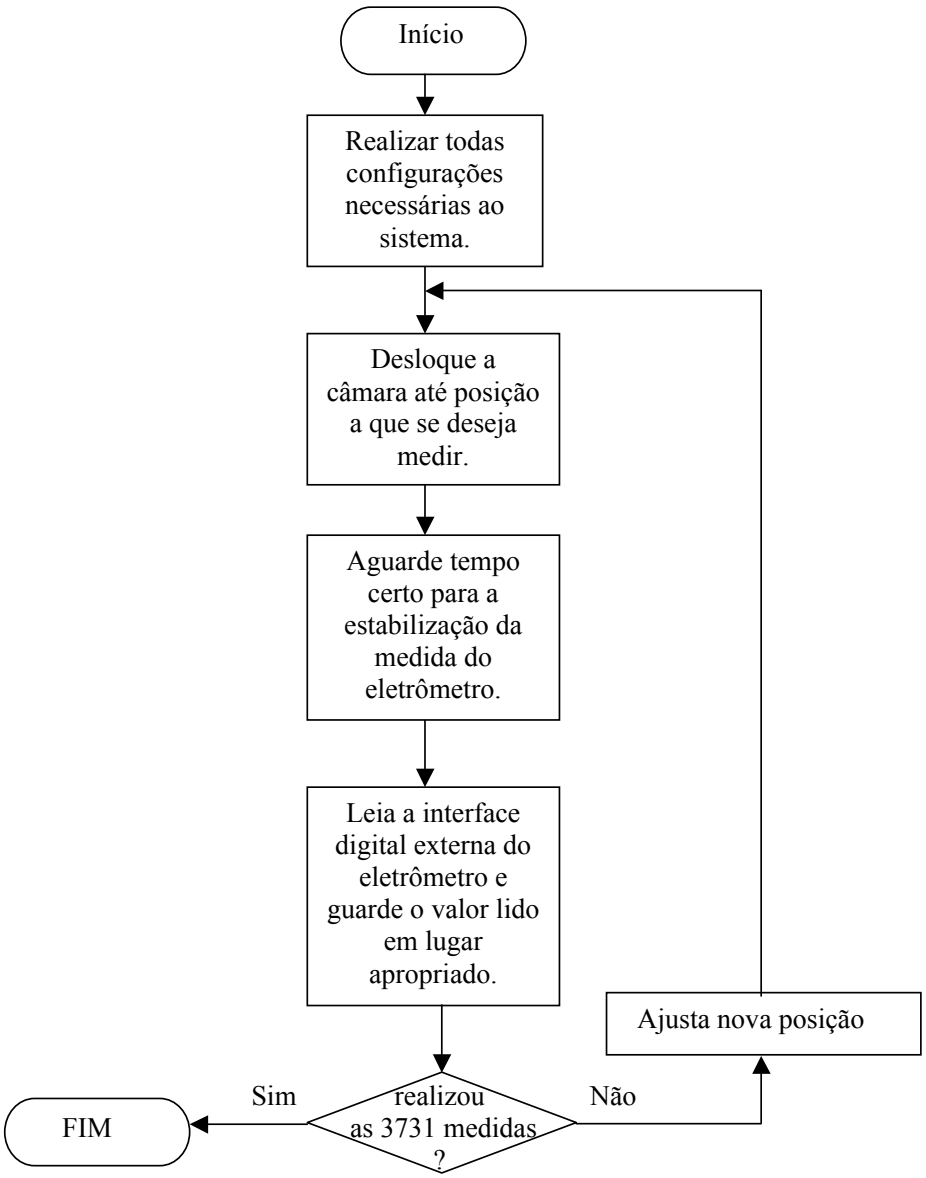


Figura 26– Fluxograma básico para automação das medidas do eletrômetro.

3.3 – Esquemático do projeto eletrônico

A Fig. 27 mostra, em diagrama de blocos, a estrutura eletrônica final de todo o projeto.

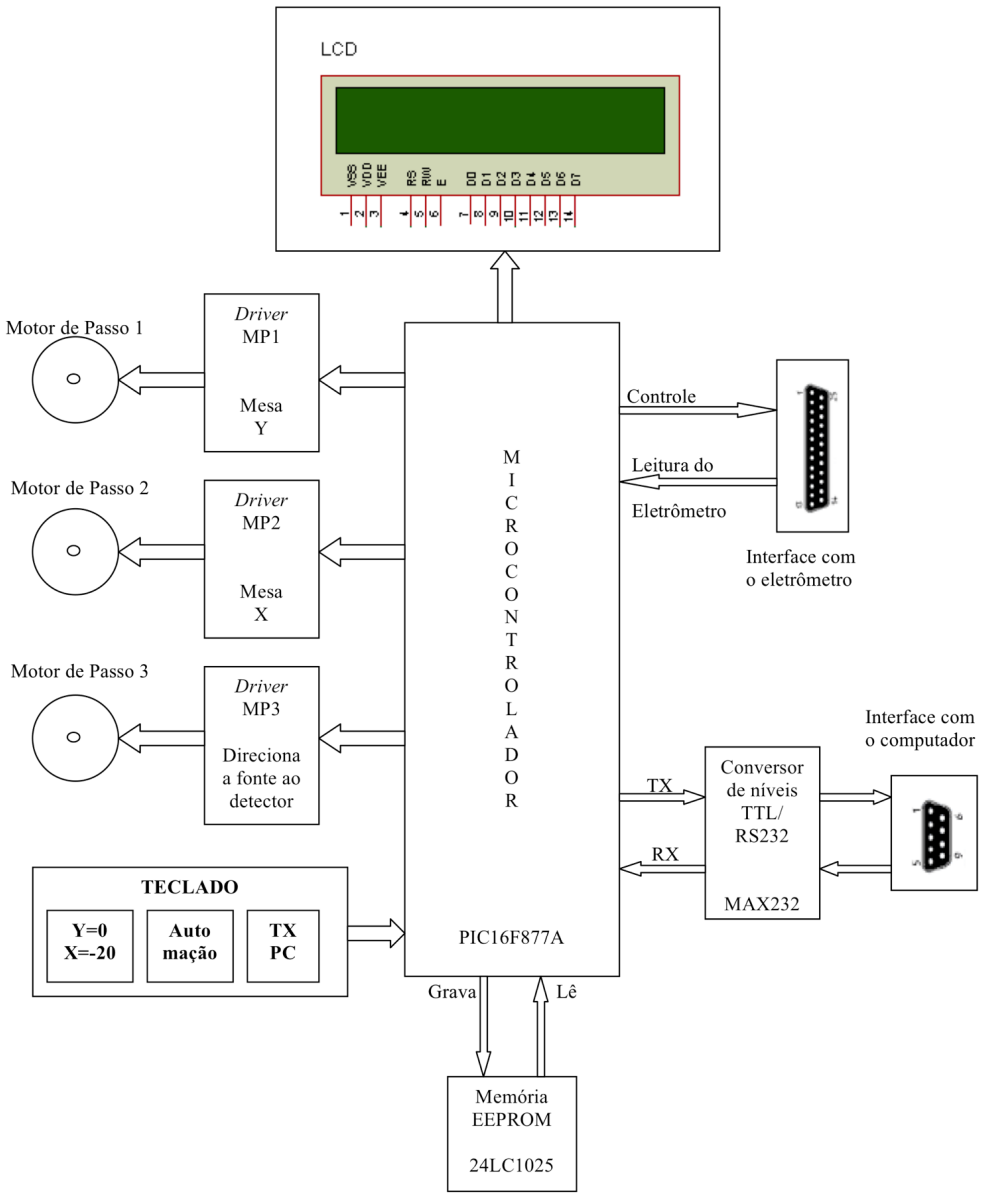


Figura 27– Diagrama em blocos do projeto

Operacionalmente, o sistema construído ficará num *loop* como na Fig. 26, isto por 3.731 vezes. Em cada *loop* o eletrômetro realizará a medida das cargas elétricas relativa a uma posição

fonte-detector e armazenará a medida na memória estática 24LC1025, mostrando-a, também, no LCD.

Na sequência foi listada as execuções dentro de cada um desses *loops*:

- posicionar a mesa Y;
- posicionar a mesa X;
- ajustar a cabeça, onde reside a fonte radioativa, diretamente ao volume sensível da câmara de ionização;
- aguardar suficiente tempo (atraso) para a estabilização da medida pelo eletrômetro;
- ler a interface digital do eletrômetro;
- agrupar duas leituras (BCD, *nibble*) em único byte;
- salvar os dados lidos na EEPROM 24LC1025 em endereços subseqüentes;
- realizar todas as medidas referentes ao mesmo posicionamento fonte-detector, armazenando-as na 24LC1015 seqüencialmente;
- reiniciar o processo.

No desenvolvimento desse projeto foi utilizado diversos materiais para a construção de sua estrutura mecânica e, em termos eletrônicos, desenvolveu-se uma PCI com recursos suficientes para suportar todo o projeto. Nesta PCI, PCI principal, encontra-se:

- um microcontrolador PIC16F877A, que é o principal componente do projeto,
- um circuito para comunicação com um LCD (que utiliza somente 4 pinos I/O do PIC, pois foi realizada uma transmissão série-paralelo através de um registrador de deslocamento, CI 74LS164),
- três saídas para os *drivers* dos motores de passo,
- um banco de memória serial implementado com o CI 24LC1025 (EEPROM),
- um teclado com três chaves. Uma das chaves aciona a rotina que controla o movimento das mesas X e Y até suas posições zero, iniciais; outra chave aciona o início da automação, controlando todo o processo de realização das 3.731 medidas do eletrômetro; e por fim, a última chave aciona o início da transmissão dos dados armazenados na memória 24LC1025, após realizadas todas as medidas (cada medida é armazenada em sete bytes), para um computador pessoal via interface RS-232.

- um circuito para adequar os níveis TTL aos da interface RS-232, cuja interface física é o conector DB9 fêmea;
- uma interface com o eletrômetro, entradas e saídas via conector DB25 fêmea.

Os tópicos sobre o LCD, sobre os motores de passo e, também, sobre a leitura do teclado (detecção de tecla acionada) já foram discutidos (FONSECA, 2008), tanto seus hardwares quanto seus softwares de controle. No ANEXO A foram reproduzidas estas rotinas *assembly* com a intenção de facilitar o entendimento sobre todo o projeto.

Além dos pontos relevados acima, resta trabalhar melhor: (a) a infraestrutura mecânica do projeto; (b) a leitura e escrita na memória serial 24LC1025; (c) o circuito de comunicação com o CP via interface serial RS-232; e (d) o estudo do eletrômetro e o controle da leitura de um dado presente em sua interface digital.

3.4 – O projeto mecânico

A infraestrutura mecânica desse projeto foi construída em madeira, foi composta por dois canais ortogonais onde duas mesas se movimentam independentemente, cada qual acionada por seu motor de passo. A mesa denominada por X transporta a fonte radioativa e a mesa denominada por Y transporta a câmara de ionização. Sobre a mesa X e mesa Y instalou-se uma torre. Sobre a torre X instalou-se um terceiro motor de passo com a função de derecionar o fluxo radioativo rumo ao centro geométrico da câmara de ionização. A Fig. 28 mostra os detalhes construtivos da torre X.

O posicionamento relativo entre X e Y acontece a diferenciais de posição dista da posição anterior em 1 cm, tanto em X quanto em Y. Na trajetória X a torre X tem liberdade de movimento de -20,0 a +20,0 cm, e na trajetória Y a torre Y pode movimentar-se de 2,0 a 92,0 cm, totalizando 3.731 posições relativas distintas. E, como já foi dito, em cada uma dessas posições relativas é realizado uma medida das cargas elétricas “geradas” no volume sensível da câmara de ionização e medidas pelo eletrômetro.

(a) (b)

Figura 28 – Detalhes construtivos da torre sobre a mesa X, (a) de lado e (b) de frente.

3.5 – Leitura e escrita da memória 24LC1025 – hardware e software

3.5.1 – Hardware – ligações da memória 24LC1025

Foram construídas três PCIs, uma contendo o microcontrolador e o controle do LCD, outra contendo os circuitos da fonte de alimentação, e uma última PCI contendo os circuitos para *driver* do motor de passo (FONSECA, 2008). Após esse primeiro momento, instalou-se na PCI do microcontrolador o banco de memórias seriais composto por até 4 memórias 24LC1025, conforme mostrado na Fig. 29. Não foi mostrado na Fig. 29 as ligações dos terminais A0 (pino 1) e A1 (pino 2), estes para endereçamento físico ou seleção dos CIs (*chip select*). A forma para ligar esses terminais foi apresentada na Tab. 31.

Os 4 CIs são alimentados ligando todos os pinos 4 (V_{SS}) ao terra (GND) e todos os pinos 8 (V_{CC}) ao +5V_{CC}. Os pinos 7 (WP) devem ser ligados ao terra (GND). Os pinos 3 (A2) devem ser ligados ao +5V_{CC}. Os circuitos SDA (pino 5) e SCL (pino 6) precisam ser ligados a um resistor *pull-up* de 10kΩ, conforme mostrado na Fig. 29.

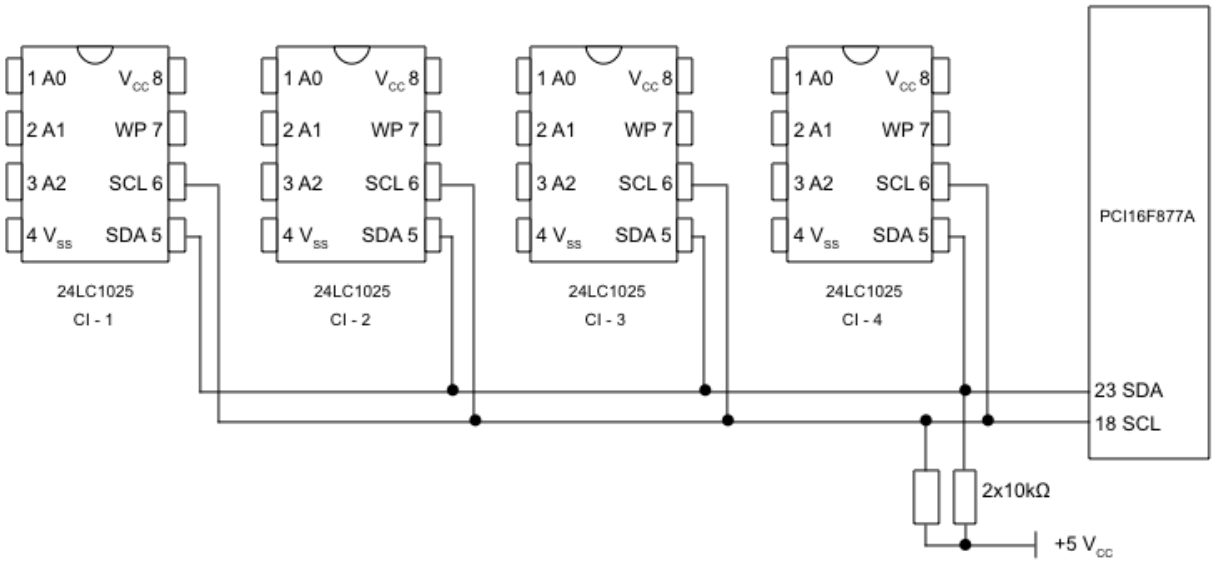


Figura 29 – Banco de memória de 512kBytes, I^2C, com CIs 24LC1025.

A Fig. 30 mostra o banco de memórias residente na placa principal – a que contém o microcontrolador PIC16F877A. Nesta PCI está grafado 24LC256 (32kBytes, onde um só desses CIs comportaria todo o projeto), mas foi usado o CI 24LC1025 (128kBytes), pois desenvolveria uma única vez um banco maior que atenderia esse e muitos outros futuros projetos (banco de 512kBytes em vez de 256kBytes). Com a memória 24LC256 necessitaria de 8 (oito) desses CIs e com a memória 24LC1025 somente 4 (quatro), tendo a vantagem do dobro de armazenamento.

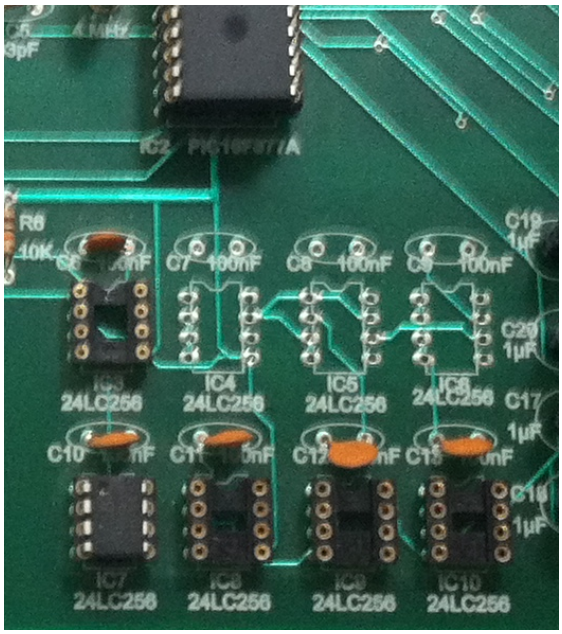


Figura 30 – Banco de memória 4 x 24LC1025.

Tabela 31 – Ligação dos pinos *chip select* dos 4 CIs 24LC1015

A1	A0	Ligação dos pinos A0 e A1 do:
GND	GND	CI1
GND	+5Vcc	CI2
+5Vcc	GND	CI3
+5Vcc	+5Vcc	CI4

Para desenvolver as rotinas de controle da escrita e leitura de um byte na memória serial foi necessário obter informações sobre o dispositivo de memória 24LC1025, sobre o protocolo de comunicação entre CIs (I^2C) aplicado à memória 24LC1025, e sobre os registradores do PIC16F877A que suportam o protocolo I^2C. De posse destas informações foi possível elaborar às sub-rotinas em *assembly* responsáveis pelo controle da leitura e escrita de um byte na memória 24LC1025. No que tange às informações sobre o protocolo I^2C e os registradores associados a eles, essas já foram apresentadas no capítulo "Revisão Bibliográfica". Cabe, agora, estudar o *datasheet* do componente 24LC1025, propriamente dito, para entender como aplicar o protocolo I^2C sobre esse dispositivo de memória.

3.5.2 – O protocolo I^2C aplicado à escrita e leitura de um byte na memória 24LC1025

O protocolo I^2C é implementado sobre um barramento composto por duas linhas, uma linha para enviar/receber dados (SDA) e outra para o mestre enviar o sinal de *clock* (SCL). Para iniciar uma transferência de dados entre o dispositivo mestre (PIC) e o escravo (memória 24LC1025), necessariamente, o barramento I^2C deverá estar desocupado (estado *Idle* da linha), situação que acontece somente quando o sinal SCL e o SDA se encontrarem em nível alto.

A primeira transmissão do mestre é o *Start* bit, o qual acontece quando o sinal em SCL estiver em nível lógico alto e na linha SDA ocorrer a variação do nível alto para o nível baixo. Imediatamente após ao envio do *Start* bit, uma rotina *assembly* prepara o byte de controle e o envia ao dispositivo escravo. Este byte de controle é mostrado na Fig. 31.

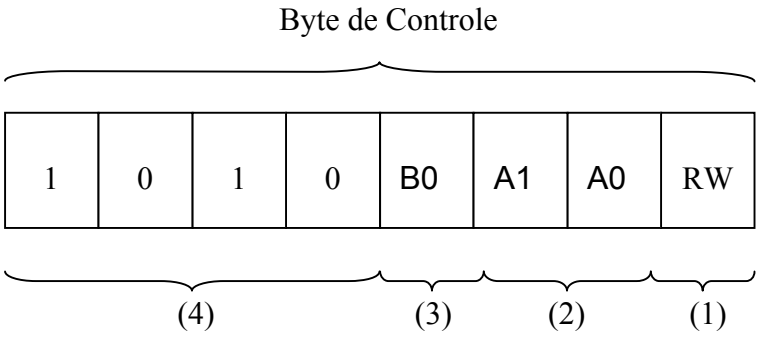


Figura 31 – Detalhes do primeiro byte do quadro de transmissão – byte de controle
Fonte: Microchip, 2010.

A Fig. 31 mostra o byte de controle. Ele é o primeiro byte transmitido pelo dispositivo *master* (PIC) ao *slave* (24LC1025), após o *start* bit ter sido enviado ao *slave*. O byte de controle é composto, do bit LSB ao MSB:

(1) Bit indicador de Leitura (*Read*, R=1) ou Escrita (*Write*, W=0) de/para uma posição endereçada;

(2) Bits para *Chip Select*,

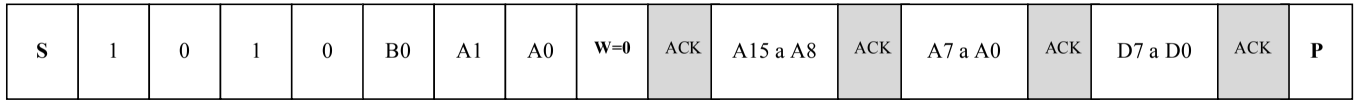
A1	A0		
0	0	→	Seleciona o primeiro CI do banco de memória;
0	1	→	Seleciona o segundo CI do banco de memória;
1	0	→	Seleciona o terceiro CI do banco de memória;
1	1	→	Seleciona o quarto e último CI do banco de memória.

(3) Bit para seleção de Bloco de 64kBytes num mesmo dispositivo de memória 24LC1025.

B0 = 0	→	Seleciona o primeiro Bloco de 64kBytes do CI 24LC1025
B1 = 1	→	Seleciona o último Bloco de 64kBytes do CI 24LC1025.

(4) Código de Controle (*nibble*: 1010)

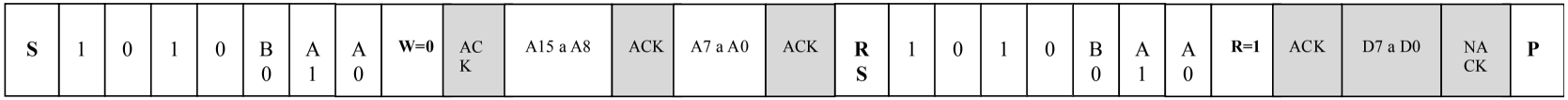
Para a leitura e escrita de um byte na memória serial deve-se seguir (ou executar) o quadro do protocolo I^2C. Este quadro mostra a sequência de TX e RX para escrever (ou gravar) um byte em um específico endereço da memória. Observa-se, na Fig. 32, que imediatamente após ao envio do byte de controle o *master* espera receber do *slave* uma confirmação de recebimento correto do referido byte. Esta comunicação refere-se ao envio do bit ACK (*Acknowledge*), o qual é enviado pelo *slave* ao *master*. Este procedimento do *slave* reconhecer o recebimento correto das transmissões do *master*, envio do bit ACK, acontece ao final de todo byte transmitido (exceto na leitura da memória, antes do *stop* bit, que TX NACK), conforme mostrado nas Fig. 32 e 33. Após o reconhecimento do byte de controle, o *master* envia ao *slave* o byte mais significativo do endereço onde se deseja escrever, ou gravar, um dado (byte). Após o seu reconhecimento (bit ACK como resposta) de bem recebido o byte, o *master* envia o byte menos significativo do referido endereço. Só assim, após o correspondente recebimento do sinal ACK, é que o *master* está liberado para enviar ao *slave* o byte de dados a ser gravado no endereço da memória previamente transmitido, que, também, é confirmado por um sinal ACK. Após este sinal o quadro é fechado e termina a transmissão com o envio pelo *master* do *Stop* bit



Bit gerado e transmitido pelo dispositivo ***Master*** – neste caso, pelo **PIC16F877A**.

Bit gerado e transmitido pelo dispositivo ***Slave*** – neste caso, pelo **24LC1025**.

Figura 32 – Gravar um byte num endereço da memória 24LC1025.
Fonte: Microchip, 2010.



Bit gerado e transmitido pelo dispositivo ***Master*** – neste caso, pelo **PIC16F877A**.

Bit gerado e transmitido pelo dispositivo ***Slave*** – neste caso, pelo **24LC1025**.

Figura 33 – Ler um byte num endereço da memória 24LC1025.
Fonte: Microchip, 2010.

Resumidamente a Fig. 32 mostra que o Start bit (S) é o primeiro bit a ser transmitido. Após o seu envio o *master* transmite o byte de controle (1010, bit B0, bits para *chip select* A1 e A0, e o bit para gravação W=0), e, então, o *master* espera pelo bit ACK enviado pelo *slave*. O *master* envia ao *slave* o byte mais significativo do endereço, recebe o sinal ACK, envia o byte menos significativo do endereço, recebe o sinal ACK, envia o byte a ser gravado, recebe o sinal ACK e fecha o quadro enviando o *Stop* bit (P).

Em relação a leitura de um byte previamente armazenado em um dos endereços da memória 24LC1025 deve-se seguir, como se fez na gravação de um byte, o quadro da Fig. 33. Observando essa figura pode-se perceber que esse quadro difere do anterior, principalmente, no que se refere ao envio do último bit W/R do byte de controle. A primeira transmissão, após o envio do *Start* bit (S), é feita com nível baixo neste bit, W=0, preparando a memória para receber o endereço a ser lido. Depois dos envios dos endereços, alto e baixo, transmite-se, um *Restart* bit (RS) (diferente do anterior) e, na sequência, o byte de controle com o bit W/R identificado como R=1, pois, desse modo, está preparando o dispositivo de memória para que ele transmita ao *master* o byte armazenado no endereço selecionado.

O conjunto de informações supra apresentadas é suficiente para a elaboração das rotinas *assembly* que deram suporte à leitura e escrita (ou gravação) de um byte na memória serial 24LC1025.

3.5.3 – Software suporte ao protocolo I^2C

No ANEXO A encontram-se as sub-rotinas que suportam tanto a escrita quanto a leitura de um byte na memória serial 24LC1025. Deve-se relembrar que o dispositivo 24LC1025 aceita a gravação/escrita de apenas um byte (byte a byte, que foi forma condizente a esse projeto) e, também, a gravação ou escrita em blocos de 128 bytes, chamados de página pela Microchip. Esta forma de gravação não foi a utilizada nesse projeto.

Nos itens subseqüentes serão mostradas as chamadas das sub-rotinas, conforme os quadros I^2C das Fig. 32 e Fig. 33. Isto é o mesmo que implementar o protocolo I^2C para escrita e leitura de um byte, respectivamente, na memória 24LC1025.

3.5.3.1 – Software I^2C – escrita de um byte na 24LC1025

A Fig. 32 mostra a estrutura do quadro do protocolo I^2C para gravar um byte na memória 24LC1025. Implementar este protocolo basta que o programador desenvolva as sub-rotinas que executam cada atividade, separadamente, a exemplo de se desenvolver a de envio do *start* bit ao *slave*. Uma vez que todas as sub-rotinas tenham sido desenvolvidas, deve-se fazer um programa de chamada seqüencial destas sub-rotinas respeitando o momento dentro do referido quadro do protocolo.

A sub-rotina nominada por GRAVA_MEDIDA_EEPROM no ANEXO A, item b36 e Fig. A56 chama outra sub-rotina nominada por ESCREVER_EEPROM_I2C no ANEXO A, item b37 e Fig. A58. Esta rotina segue a sequência de chamada de outras sub-rotinas, como:

1) Enviar *START* BIT (que é o mesmo que SCL=1, e SDA variando de 1 para 0). Rotina nominada por TX_START_BIT_I2C, Fig. A59.
2) Rodar a sub-rotina que espera o barramento I^2C desocupar, ficar *idle*, nominada por ESPERA_I2C_DESOCUPAR, Fig. A61.
3) Neste momento, o byte de controle, que é composto por 1010 B0 A1 A0 R/W=0 (se B0=0 está selecionando os primeiros 64kBytes da memória 24LC1025, cuja capacidade total é de 128kbytes), juntamente com o endereço corrente (A15 a A0, onde será gravado o byte atual) já foram acertados. Isso ocorre pois com a chamada da sub-rotina GRAVA_MEDIDA_EEPROM, esta chama outra sub-rotina nominada por ACERTA_CONTROLE_ENDERECO_I2C (Fig. A60), a qual, como o nome tenta ajudar a lembrar, atualiza tanto o byte de controle quanto o endereço (A15 a A0) onde será gravado o byte atual. Então, resta, resetar o bit R/W, pois está gravando um byte, e transmitir o byte de controle ao *slave*. Para transmitir o byte de controle ao *slave* (com W=0), basta carregá-lo no registrador SSPBUF do módulo MSSP interno ao PIC16F877A.
4) Rodar a sub-rotina que espera o barramento I^2C desocupar, ficar *idle*, nominada por ESPERA_I2C_DESOCUPAR, Fig. A61.
5) Chamar a sub-rotina nominada por TESTAR_ACK_I2C, Fig. A62.

6) Enviar os 8 bits mais significativos do endereço (A15 a A8). Basta carregar este byte no registrador SSPBUF do módulo MSSP.
7) Rodar a sub-rotina que espera o barramento I^2C desocupar, ficar *idle*, nominada por ESPERA_I2C_DESOCUPAR, Fig. A61.
8) Chamar a sub-rotina TESTAR_ACK_I2C, Fig. A62.
9) Enviar os 8 bits menos significativos do endereço (A7 a A0). Conforme (6), acima.
10) Rodar a sub-rotina que espera o barramento I^2C desocupar, ficar *idle*, nominada por ESPERA_I2C_DESOCUPAR, Fig. A61.
11) Chamar a sub-rotina TESTAR_ACK_I2C, Fig. A62.
12) Enviar ao *slave* o dado a ser escrito, dado de 8 bits, ou gravado na 24LC1025, ou seja, deve-se carregar o referido byte no registrador SSPBUF do módulo MSSP.
13) Rodar a sub-rotina que espera o barramento I^2C desocupar, ficar *idle*, nominada por ESPERA_I2C_DESOCUPAR, Fig. A61.
14) Chamar a sub-rotina TESTAR_ACK_I2C, Fig. A62.
15) Enviar *STOP* BIT, término da transmissão do quadro (esse sistema entende como *stop* bit quando SCL=1 e SDA variando de 0 para 1). Rotina nominada por TX_STOP_BIT_I2C, Fig. A63.
16) Rodar a sub-rotina que espera o barramento I^2C desocupar, ficar *idle*, nominada por ESPERA_I2C_DESOCUPAR, Fig. A61.

3.5.3.2 – Software I^2C – leitura de um byte da 24LC1025

A Fig. 33 mostra a estrutura do quadro do protocolo I^2C para leitura de um byte previamente gravado na memória 24LC1025. Implementar este protocolo basta que o programador desenvolva as sub-rotinas que executam cada atividade, separadamente, a exemplo de se desenvolver a de envio do *start* bit ao *slave*. Tendo sido desenvolvidas todas as sub-rotinas, faz-se um programa de chamada seqüencial destas sub-rotinas respeitando o momento dentro do referido quadro do protocolo.

A sub-rotina nominada por LE_EEPROM_SERIAL no ANEXO A, item b36 e Fig. A57 chama outra sub-rotina nominada por LER_EEPROM_I2C no ANEXO A, item b37 e Fig. A64. Esta rotina segue a sequência de chamada de outras sub-rotinas, como:

1) Enviar *START* BIT (que é o mesmo que SCL=1, e SDA variando de 1 para 0). Rotina nominada por TX_START_BIT_I2C, Fig. A59.
2) Rodar a sub-rotina que espera o barramento I^2C desocupar, ficar *idle*, nominada por ESPERA_I2C_DESOCUPAR, Fig. A61.
3) Neste momento, o byte de controle, que é composto por 1010 B0 A1 A0 R/W=0 (se B0=0 está selecionando os primeiros 64kBytes da memória 24LC1025, cuja capacidade total é de 128kbytes), juntamente com o endereço corrente (A15 a A0, onde será lido o byte presente) já foram acertados. Isto ocorre pois com a chamada da sub-rotina LE_EEPROM_I2C, esta chama outra sub-rotina nominada por ACERTA_CONTROLE_ENDERECO_I2C (Fig. A60), a qual, como o nome tenta ajudar a lembrar, atualiza tanto o byte de controle quanto o endereço (A15 a A0) onde será lido o byte atual. Então, resta, resetar o bit R/W do byte de controle, pois está escrevendo (W=0) o endereço do *slave*, do lugar onde será lido o byte, e, então, transmiti-lo ao *slave*. Daí, inicialmente o byte de controle é transmitido ao *slave* com W=0. Esta transmissão ocorrerá após o byte de controle ser carregado no registrador SSPBUF do módulo MSSP interno ao PIC16F877A.
4) Chamar a sub-rotina nominada por TESTAR_ACK_I2C, Fig. A62.
5) Enviar os 8 bits mais significativos do endereço (A15 a A8). Basta carregar este byte no registrador SSPBUF do módulo MSSP.
6) Chamar a sub-rotina TESTAR_ACK_I2C, Fig. A62.
7) Enviar os 8 bits menos significativos do endereço (A7 a A0). Conforme (5), acima.
8) Chamar a sub-rotina TESTAR_ACK_I2C, Fig. A62.
9) Enviar *RESTART* BIT. Rotina nominada por TX_RESTART_BIT_I2C, Fig. A65.
10) Rodar a sub-rotina que espera o barramento I^2C desocupar, ficar *idle*, nominada por ESPERA_I2C_DESOCUPAR, Fig. A61.

11) Seta o bit R/W do byte de controle (R=1, agora, efetivamente para realizar a leitura do conteúdo da posição endereçada) e carrega-o no registrador SSPBUF do módulo MSSP, para ser transmitido ao *slave*.
12) Rodar a sub-rotina que espera o barramento I^2C desocupar, ficar *idle*, nominada por ESPERA_I2C_DESOCUPAR, Fig. A61.
13) Chamar a sub-rotina TESTAR_ACK_I2C, Fig. A62.
14) Seta o bit RCEN do registrador SSPCON2, indicando que o *master* está pronto para receber do *slave*.
15) Rodar a sub-rotina que espera o barramento I^2C desocupar, ficar *idle*, nominada por ESPERA_I2C_DESOCUPAR, Fig. A61.
16) Carrega o dado lido no registrador W e após no registrador criado especificamente para este dado, nominado por DADO_I2C.
17) Rodar a sub-rotina TX_NACK_I2C, Fig. A66.
18) Rodar a sub-rotina que espera o barramento I^2C desocupar, ficar *idle*, nominada por ESPERA_I2C_DESOCUPAR, Fig. A61.
19) Enviar *STOP* BIT, término da transmissão do quadro (esse sistema entende como *stop* bit quando SCL=1 e SDA variando de 0 para 1). Rotina nominada por TX_STOP_BIT_I2C, Fig. A63.
20) Rodar a sub-rotina que espera o barramento I^2C desocupar, ficar *idle*, nominada por ESPERA_I2C_DESOCUPAR, Fig. A61.

3.6 – Comunicação com o CP via RS-232

Foi desenvolvido um sistema eletro-eletrônico-mecânico capaz de realizar 3.731 distintos posicionamentos relativos entre uma fonte radioativa e uma câmara de ionização, aquela rumo ao centro geométrico desta. Em cada uma dessas posições relativas realiza-se uma medida das cargas elétricas ionizadas nesse processo. Realizada uma medida ela é armazenada na memória serial 24LC1025, seqüencialmente. Uma vez que as 3.731 medidas estejam armazenadas na memória serial, resta, para completar o processo, a disponibilização dessa massa de dados a um sistema autônomo, a exemplo de transmiti-la a uma planilha eletrônica. Assim sendo, os dados podem ser trabalhados estatisticamente, e deles pode-se extrair as informações que em si

carregam. Ou sejam, a dose e a taxa de dose. Portanto, foi preciso desenvolver e construir alguns circuitos e softwares necessários para transmitir a referida massa de dados ao CP. Foi utilizado, para isso, a interface serial RS-232 para interconectar, física e logicamente, o PIC16F877A ao CP. A interconexão lógica é realizada pelo protocolo de comunicação assíncrona *Start-Stop*, uma vez que no CP esteja instalado e rodando o software hyperterminal (htpe63.exe), da Microsoft.

3.6.1 – Hardware suporte à interface serial RS-232

Os pinos 25 (TX) e 26 (RX) do PIC16F877A são usados pela USART para Transmissão/Recepção dos dados. Como já se disse, para configurar os referidos pinos a trabalhar como USART, no modo assíncrono, deve-se resetar o bit SYNC (TXSTA<4>) e setar o bit SPEN (RCSTA(<7>). Deve-se, também, configurar os pinos TX e RX como entradas, e isto é feito setando TRISC(7:6). O PIC trabalha com sinais TTLs, e esses são incompatíveis com a recomendação padrão RS-232. Para adaptar esses sinais aos da interface RS-232 utilizou-se o CI MAX232, tratado no item 2.3. A Fig. 34 mostra o circuito de transmissão de dados ao CP, pelo PIC.

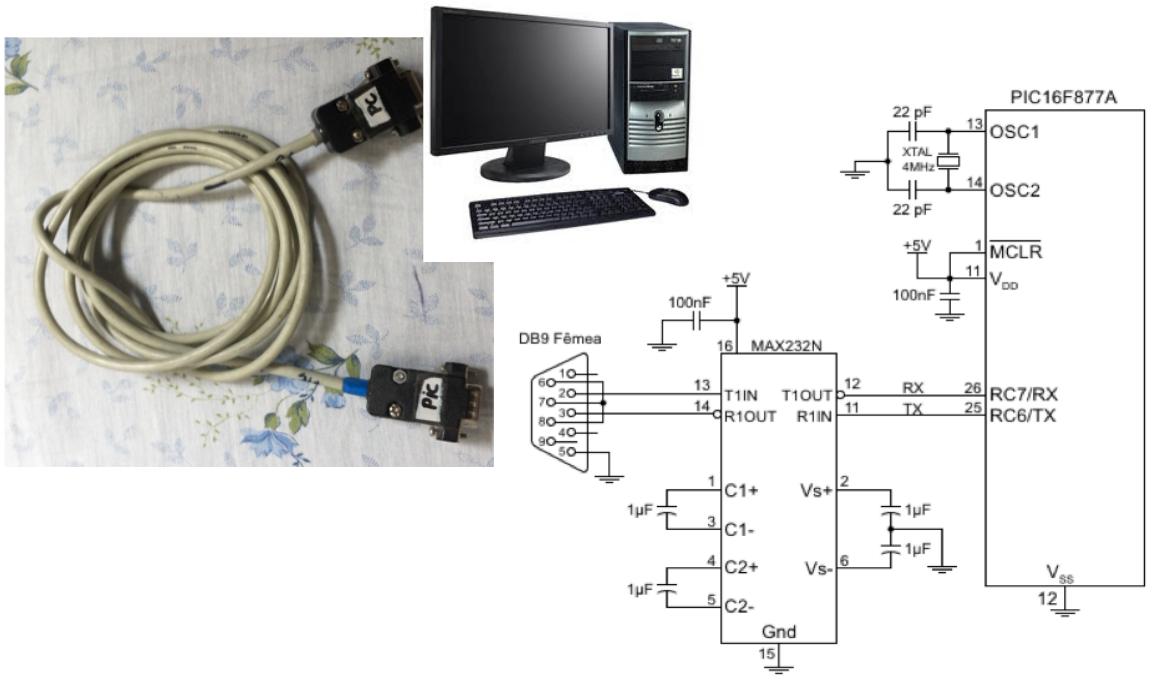


Figura 34 – Hardware suporte à comunicação RS-232, PIC – CP

As especificações mecânicas do conector DB9, com 9 pinos, tem suas designações apresentadas na Tab. 32.

Tabela 32 – Designação dos pinos do conector DB9

Pino nº.	Designação do Pino
1	*Data Carrier Detect* (DCD)
2	Terminal de Recepção de dados (RX)
3	Terminal de Transmissão de dados (TX)
4	*Data Terminal Ready* (DTR)
5	Terra, referência para TX e RX (GND)
6	*Data Set Ready* (DSR)
7	*Request To Send* (RTS)
8	*Clear To Send* (CTS)
9	*Ring Indicator* (RI)

Fonte: Adaptado (ZANCO, 2006)

Um terminal para comunicação de dados, microprocessado ou microcontrolado, é designado por *Data Terminal Equipment* (DTE) ou Equipamento Terminal de Dados. Por vezes um desses terminais é ligado a um *Data Comunication Equipment* (DCE) ou Equipamento de Comunicação de Dados, a exemplo de ligá-lo a um MODEM (sigla que advém da concatenação das iniciais das palavras inglesas *modulator*/*demodulator*), o qual entra no circuito de comunicação de dados para possibilitar vencer grandes distâncias entre o DTE-DCE local e o DCE-DTE remoto, estes podendo ser ligados por linhas físicas, por fibras óticas ou outro meio físico condizente com as saídas desses MODEMs.

Dos sinais presentes no conector DB9, o sinal DCD, é um sinal entrante no conector do DTE, e indica que o DCE (local) detectou o sinal da portadora na linha, condição inicial para que possa acontecer a comunicação entre os terminais DTEs, local e remoto. Os sinais RX e TX são recepção e transmissão de dados, respectivamente. O DTR é um sinal do DTE ao DCE (local) indicando que aquele está pronto para transmitir dados. O sinal RTS é uma solicitação do DTE ao DCE (local) para que aquele possa transmitir dados. O sinal CTS, sinal do DCE (local) ao DTE, indicando a este que pode enviar dados. O sinal DSR, o DCE (local) sinaliza ao DTE que está pronto para iniciar a comunicação. O sinal RI, sinalização do DCE (local) ao DTE indica que aquele está recebendo um *ring*, ou uma chamada da linha telefônica. A referência de GND deve

ser comum a ambos DTE e DCE, dado que a recomendação padrão RS-232 usa transmissão não balanceada (ZANCO, 2006).

Como o objetivo desse trabalho é a transferência da massa de dados armazenada sequencialmente na memória serial 24LC1025 ao CP, então é necessário, apenas, a conexão física da PCI do PIC diretamente ao CP. Esta ligação é realizada por um cabo DTE-DTE. Necessita-se, para esse fim, apenas das ligações ilustradas na Fig. 35.

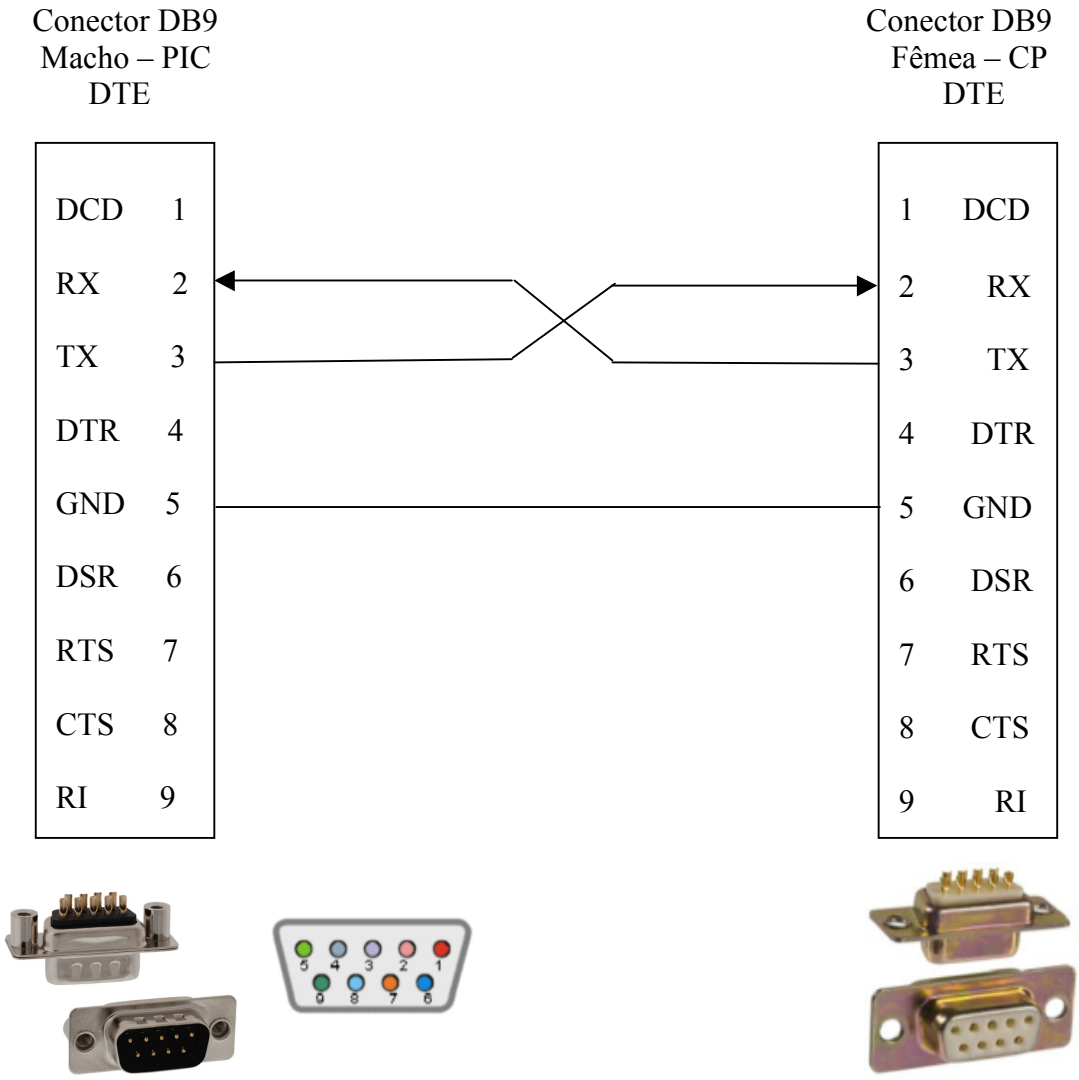
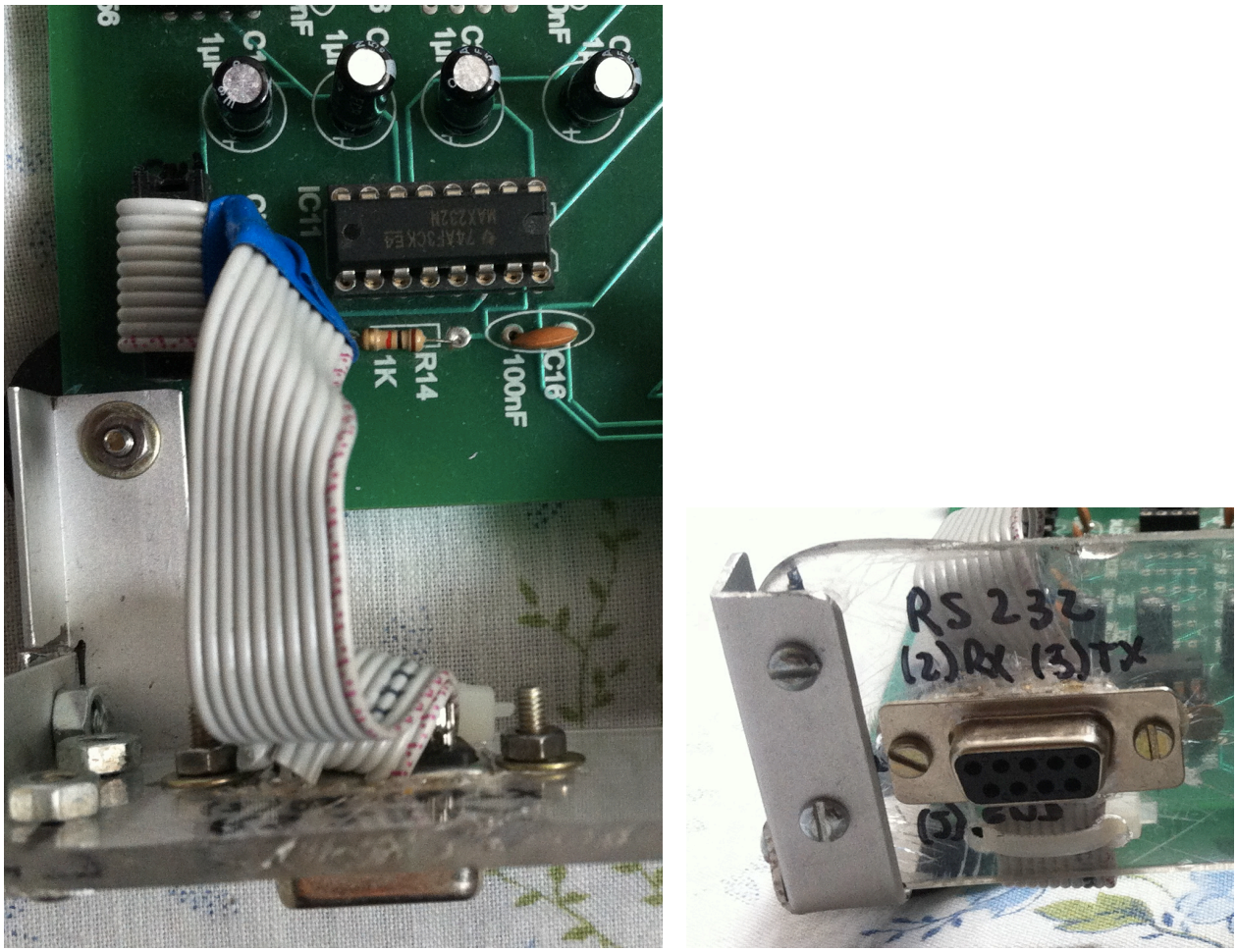


Figura 35 – Cabo DTE PIC – DTE CP**.**
Fonte: Adaptado (ZANCO, 2006)

As Fig. 36 (a) e (b) ilustram o hardware da RS-232, ou seja, elas ilustram a situação final dessa interface na PCI que contém o PIC16F877A.



(a) (b)

Figura 36 – Resultado final da RS-232 na PCI, (a) vista de cima e (b) de lado.

3.6.2 – Software suporte à comunicação via interface serial RS-232

Ao ligar a máquina é apresentado no LCD o *menu* inicial, indicando as teclas 1, 2 e 3 do teclado, no seguinte formato: <1> Pos_0_XY; <2> AUTOMAÇÃO e <3> TX RS232. A Fig. 37 mostra estas teclas na PCI. O grafado na PCI condiz com o momento inicial do projeto. Do início ao fim do projeto algumas coisas foram ajustadas, onde a tecla da esquerda é a <1>, a do meio é a <2> e a da direita é a <3>.

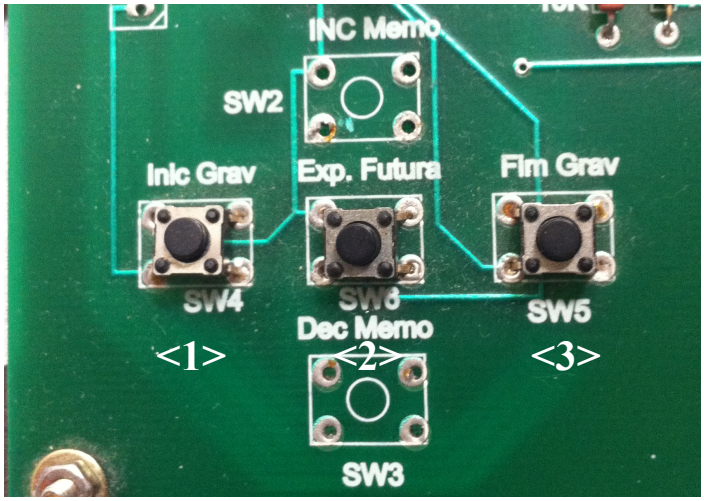


Figura 37 – Teclado de três chaves

Quando a massa de dados (medidas realizadas) estiver armazenada na memória serial 24LC1025, ela (massa) pode ser transferida ao CP. Para isso a PCI com o PIC deve ser conectada ao CP através de um cabo especificamente feito para isto (Fig. 35). Este cabo tem numa extremidade um conector DB9 macho e na outra um conector DB9 fêmea. O conector DB9 macho deve ser ligado ao conector DB9 fêmea da PCI (do PIC) e o conector DB9 fêmea do cabo ligado ao conector macho da interface RS-232 do CP. Assim, a conexão física para a transferência dos dados estará pronta. Deve-se, também, instalar, configurar e rodar o software hyperterminal (htpe63.exe) da Microsoft, no CP. A configuração e resultado dos dados recebidos com o processamento desse software são encontrados no ANEXO B. Após executado o procedimento acima basta pressionar a tecla <3> na PCI com o PIC que o software *assembly* se encarregará da transferência da massa de dados ao CP.

O *menu* que aparece no LCD ao ligar a máquina condiz com a Fig. A3, ANEXO A, e rotina *assembly* nominada por DENOVO. Essa rotina fica testando as três teclas, continuamente (num "*loop*"), até que uma delas seja pressionada. Sendo uma delas pressionada, o processamento é desviado para a rotina que executa a função que foi determinada para a tecla pressionada, ou seja:

- **Pressionando a tecla <1>**: o processamento é deslocado para a rotina nominada por LEVA_YX_0, Fig. A4 do ANEXO A, que é a primeira atividade da máquina, pois inicialmente deve-se providenciar o ajuste das mesas às suas posições iniciais.
- **Pressionando a tecla <2>:** o processamento é deslocado para a rotina nominada por PROCESSO_AUTOMAT, Fig. A5, Fig. A6 e Fig. A7 do ANEXO A. Esta rotina controla praticamente todo o processamento. É, de fato, a rotina cujo processamento é mais demorado. Ela controla o posicionamento relativo entre as torres Y e X; acerta o fluxo radioativo rumo ao centro geométrico da câmara de ionização; aguarda tempo de estabilização da medida feita pelo eletrômetro; faz a leitura da metida, que, de fato, esta é dividida em 9 (nove) leituras do tamanho de um *nibble* cada mais os valores das variáveis Y e X, as quais são agrupadas em sete bytes e armazenados em endereços adjacentes na memória serial 24LC1025. Este é um processo demorado, dado que ele é repetido, para ser completado, por 3.731 vezes.
- **Pressionando a tecla <3>:** o processamento é deslocado para a rotina nominada por TX_DADOS_RS232, Fig. A8 do ANEXO A. Uma vez que a massa de dados esteja armazenada na memória 24LC1025, essa rotina providencia o tratamento e envio dos dados ao CP. Ou seja, como os sete bytes armazenados sequencialmente na memória 24LC1025 referem-se a única medida, então, para cada byte lido deve-se analisá-lo, bit a bit – separadamente ou em conjunto, para realmente extrair a medida que neles é guardada. Cada resultado de uma análise individual deve ser convertido em seu representante no código ASCII, possibilitando, dessa forma, seu envio ao LCD ou ao CP – comunicações máquina-homem.

A seguir é mostrado como foi elaborado a emissão do relatório contendo a massa de dados – ANEXO B. Um relatório contendo uma massa de dados parece ser melhor apresentado na forma de uma tabela. Para isso, foi preciso imprimir um cabeçalho, com vários campos e, linha após linha, imprimir o conteúdo de cada campo em colunas, abaixo do equivalente campo do cabeçalho.

Como foi dito, ao ser pressionada a tecla <3> do teclado, o processamento desloca para a rotina TX_DADOS_RS232, Fig. A8, ANEXO A. A primeira atividade solicitada por esta rotina é chamar outra sub-rotina nominada por TX_CABECALHO_RS232, item b34 e Fig. A54, também do ANEXO A. O formato dado a esse cabeçalho foi:

b**ENDERECO**bbb**ENDERECO**bbbb**Y**bbb**X**bb_____**DADOS**_____bbbbb.

Deve-se observar que a letra b significa *blank*, ou espaço em branco. Já o primeiro ENDERECO refere-se ao endereço inicial da memória 24LC1025 onde reside o primeiro dos sete bytes da medida, e o último ENDERECO é o do sétimo byte da mesma medida. Y é a posição física da mesa, ou torre Y, compreendida de 2,0 a 92,0 cm. X é a posição física da mesa, ou torre X, compreendida de -20,0 a +20,0 cm. O campo DADOS é o valor já tratado, em ASCII, de toda a medida. Se a medida for de carga elétrica será salvo a letra C, de Coulomb; se for de resistência, será salvo a letra R, para lembrar Ohm; e assim, V para Volt e A para Ampère.

Como já apresentado, o protocolo assíncrono transmite um byte por vez, e este byte é "envelopado" pelo *Start* bit, no início, e fechado pelo *Stop* bit ao final de cada transmissão parcial. Então, a cada byte do cabeçalho que deve ser transmitido, necessariamente, a sub-rotina TX_CABECALHO_RS232 tem que chamar outra sub-rotina, nominada por TX_BYTE_RS232_PC, item b33 e Fig. A53 do ANEXO A.

Terminada a transmissão do cabeçalho, a sub-rotina TX_CABECALHO_RES232 retorna o processamento à sub-rotina TX_DADOS_RS232, que iniciará a leitura seqüencial da memória 24LC1025. Esta leitura é iniciada pelo endereço 0x0000, e para isso é necessário zerar os registradores ENDERECO_LOW e ENDERECO_HIGH. Além disso, deve-se zerar, também, o registrador A1A0_I2C e carregar o registrador CONTROLE_I2C com o byte 0xA0, que são condições iniciais necessárias ao processo de leitura.

Inicializado os registradores acima com aqueles valores, a sub-rotina TX_DADOS_RS232 chama outra sub-rotina nominada por LE_EEPROM_SERIAL, que já foi suscitada no item

3.5.3.2 e se encontra no ANEXO A, item b36 e Fig. A57. Essa sub-rotina, começa a ler a partir do endereço 0x0000 até o endereço 0x0006 – que são os lugares onde residem os dados da primeira medida feita pelo eletrômetro. Lido esses endereços, o conteúdo do endereço 0x0000 é armazenado no registrador denominado por Y, o do 0x0001 é armazenado no registrador X, o do 0x0002 é armazenado no registrador ST_1_2 (designado assim para lembrar que nesse byte contém os dados da leitura do eletrômetro referente ao *STROBE* 1 e *STROBE* 2); o conteúdo do endereço 0x0003 é armazenado no registrador ST_3_4; o do 0x0004 é armazenado no registrador ST_5_6 e o do 0x0005 é armazenado em ST_7_8; e por fim, completando a leitura de uma medida, o conteúdo do endereço 0x0006 é armazenado no registrador ST_9, este só contém os dados referentes aos *STROBE* 9 do eletrômetro. Então, a partir do endereço 0x0007 reside o Y da segunda medida realizada pelo eletrômetro, e assim por diante. Deve-se observar que todos esses dados estão armazenados em BCD, e precisam ser analisados e convertidos para ASCII, seja para envio ao LCD, seja para TX via RS-232 ao CP.

Assim sendo, inicialmente deve-se tratar dos bytes Y e X, convertendo-os para ASCII. Isto se realiza chamando sequencialmente duas sub-rotina, Y_ASCII, ANEXO A, item b9 e Fig. A29, e a sub-rotina SINAL_X_ASCII, ANEXO A, item b10 e Fig. A30.

Ao fim desse processamento os registradores Y_DEZENA e Y_UNIDADE estarão carregados com a dezena e unidade do valor de Y, em BCD. E, ainda, estarão carregados no registrador L1_C2 o conteúdo ASCII de Y_DEZENA e no registrador L1_C3 o conteúdo ASCII de Y_UNIDADE. Deve-se observar que os registradores assim grafados, a exemplo de L1_C2, busca ajudar a lembrar da posição física que será externado no LCD, ou seja, na L1 = Linha 1 e na C2 = Coluna 2 do LCD.

A sub-rotina SINAL_X_ASCII, após sua execução, resulta no valor de X que é carregado nos registrador X_DEZENA e X_UNIDADE, em BCD. O sinal de X, em ASCII, é carregado no registrador L1_C7, e o conteúdo de X_DEZENA, em ASCII, em L1_C8, e por fim, o conteúdo de X_UNIDADE, em ASCII, em L1_C9.

Nesse ponto do processamento, a sub-rotina TX_DADOS_RS232 chama outra sub-rotina, nominada por CONVERTE_ST_ASCII, ANEXO A, item b3, Fig. A12 até Fig. A19. Esta, pelo tamanho de seus fluxogramas já conota certa complexidade, mas sem nenhum problema real para entendimento.

A sub-rotina CONVERTE_ST_ASCII analisa bit a bit – individualmente ou em conjunto – cada registrador, a exemplo do ST_1_2, e extrai de cada bit ou conjunto de bits o seu significado. Ainda em referência ao registrador ST_1_2, este guarda um byte e é composto por duas parcelas (dentre nove) da leitura da medida referente aos *STROBEs* 1 e 2. Seus bits são mostrados na Tab. 33.

Tabela 33 – Composição do registrador ST_1_2

Dados quando STROBE 1 ativo:				**Dados quando STROBE 2 ativo:**			
Exp.$1x10^1$	Sinal Exp	F_2	F_1	Exp. $8x10^0$	Exp. $4x10^0$	Exp. $2x10^0$	Exp. $1x10^0$
b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

Fonte: Adaptado (KEITHLEY, 1980)

A Tab. 33 mostra o conteúdo do registrador ST_1_2. A sub-rotina CONVERTE_ST_ASCII ao processar o conteúdo do registrador ST_1_2, analisa e converte-o em seus representantes do código ASCII. Feita esta análise e conversão os resultados, em ASCII, são carregados nos registradores L2_C9 a L2_C15, conforme mostrado abaixo:

L2_C9 ← Carrega com o ASCII da letra E, de Expoente base 10.

L2_C10 ← interpretando o bit b_6 do registrador ST_1_2 extrai-se que o sinal do expoente é positivo (ASCII, 0x2B) ou negativo (ASCII, 0x2D).

L2_C11 ← interpreta o bit b_7 do registrador ST_1_2, que é o valor da dezena do expoente do número. Se este bit for zero o registrador L2_C11 deverá ser carregado com o ASCII do algarismo zero, ou seja, com o valor 0x30. Se o referido bit for um, o registrador L2_C11 será carregado com o valor ASCII do algarismo um, ou seja, com o valor 0x31.

L2_C12 ← Como os bits b_3 a b_0 tratam da unidade do expoente, deve-se enviá-los à sub-rotina nominada por HEXA_ASCII, no ANEXO A, item b6 e Fig. A23. Esta sub-rotina converte esse *nibble* em seu representante no código ASCII, que será carregado no registrador L2_C12.

L2_C13 ← carrega 0x20, espaço em ASCII.

L2_C14 ← Com a interpretação dos bits b_4 e b_5 do registrador ST_1_2, L2_C14 pode ser carregado com o ASCII da letra R, caso a medida seja de resistência; com o

ASCII da letra C, caso seja de carga elétrica; com o ASCII da letra A, caso seja de corrente elétrica; e por fim, com o ASCII da letra V, caso a medida seja de Volt, diferença de potencial elétrico.
L2_C15 ← carrega 0x20, espaço em ASCII.

Então, conforme foi mostrado com a decodificação do registrador ST_1_2, a sub-rotina CONVERTE_ST_ASCII realiza semelhante análise para todos os demais registradores ST_3_4 a ST_9 (carregando os demais registradores L2_C0 a L2_C8). Deve-se lembrar que todos esses STs, em conjunto, referem-se a única medida. Terminado o processamento dessa sub-rotina ela retorna o processamento à sub-rotina que a chamou, ou seja à sub-rotina TX_DADOS_RS232.

O controle tendo sido retornado à sub-rotina TX_DADOS_RS232, esta, agora, com todos os dados devidamente armazenados, providencia a chamada de outra sub-rotina, esta nominada por TX_LINHA_DADOS_RS232, do ANEXO A, item b35 e Fig. A55, a qual controla a transmissão de uma linha completa de dados ao CP.

O sistema, conforme implementado, repetirá, mesma sequência, até acabar todas as 3.731 medidas, Fig. 26. Terminada a transmissão dos dados ao CP, o sistema volta ao *menu* que é apresentado no LCD. O *menu* indica ao operador da máquina, que ele pode, se quiser, iniciar uma nova sequência de posicionamentos de X, Y e medidas das cargas elétricas, ou mesmo desligar todo sistema.

3.7 – O eletrômetro

Nesse projeto foi utilizado o eletrômetro digital modelo 616, fabricado pela *Keithley Instruments,* Inc.. Este eletrômetro é um equipamento que realiza múltiplas medidas com alta precisão, como a medição de resistência elétrica (Ohm), de tensão elétrica (Volt), de corrente elétrica (Ampère) e, também, de carga elétrica (Coulomb). Interessa-se, nesse projeto, pelo uso do eletrômetro 616 como medidor de carga elétrica.

Ao eletrômetro 616 deve-se conectar o equipamento 6162, também de fabricação da *Keithley*, pois é pelo 6162, por seu conector DB50, que são extraídos os dados das medidas realizadas pelo eletrômetro 616, conforme mostrado na Fig. 38.

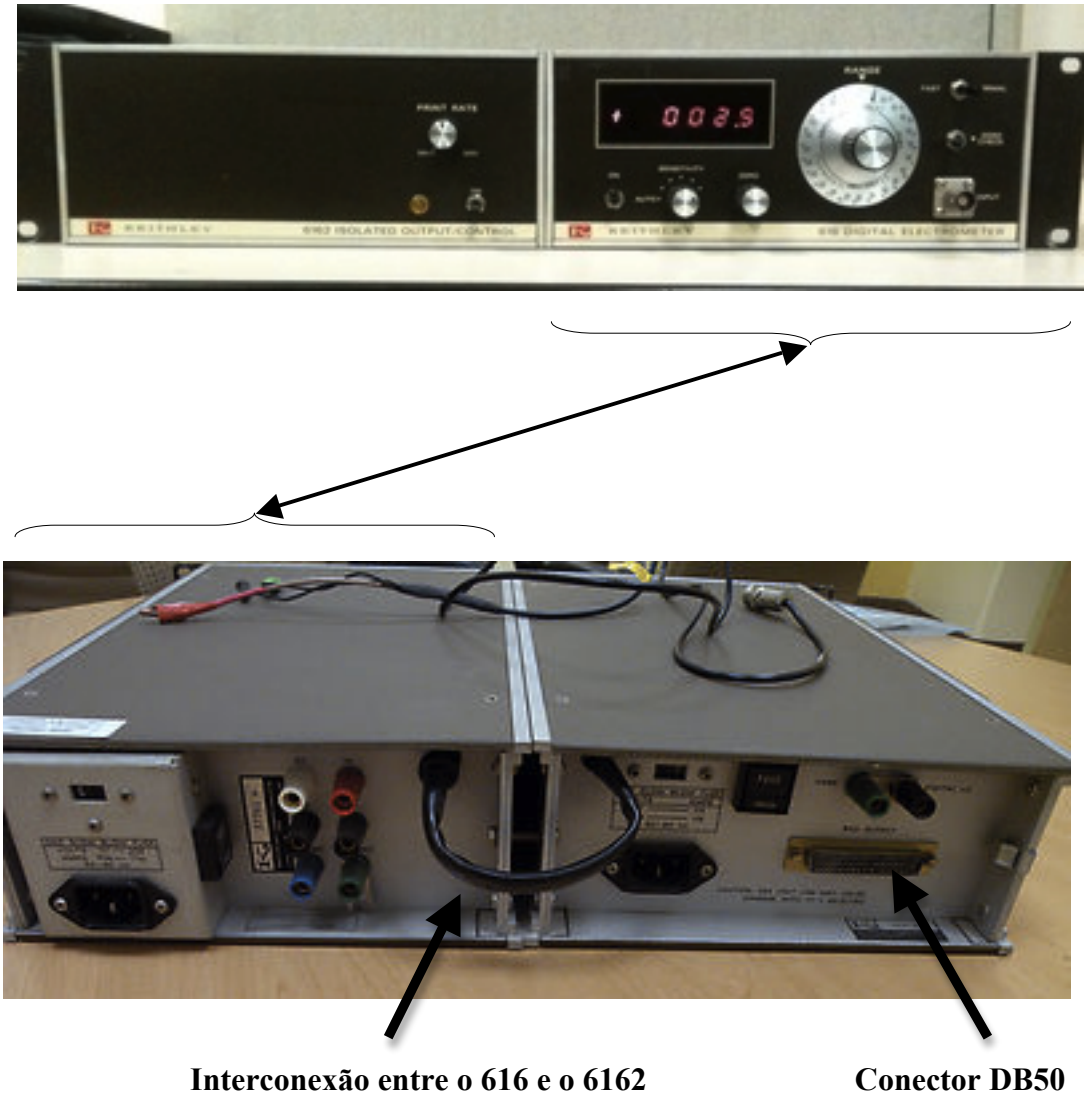


Figura 38 – Conexão do eletrômetro 616 ao 6162, frente e traseira.

Quando os equipamentos 616 e 6162 são conectados, como na Fig. 38, pode-se acessar aos dados presentes no display do 616 pelo conector DB50, este na traseira do 6162, Fig. 39. Esses dados são apresentados em conjunto de 4 bits (*nibble*) e sua apresentação depende de um específico sinal STROBE estar ativado, ou em nível lógico baixo, conforme mostra a Tab. 34. A Tab. 34 completa a ideia iniciada na Tab. 33. O controle dos STROBEs é realizado pelo

PIC16F877A, que ativa um dos STROBEs por vez, e, imediatamente após uma ativação, o PIC promove a leitura dos respectivos 4 bits, os quais compõem parte de uma medida.

Tabela 34 - Transferência de dados do eletrômetro

Linhas Strobe	**Níveis lógicos das linhas Strobe.**									**Saída**	**Saída**	**Saída**	**Saída**
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**D (+ sig.)**	**C**	**B**	**A (- sig.)**
1	0	1	1	1	1	1	1	1	1	Exp $1x10^1$	EXP POL.	F_2	F_1
2	1	0	1	1	1	1	1	1	1	Exp $8x10^0$	Exp $4x10^0$	Exp $2x10^0$	Exp $1x10^0$
3	1	1	0	1	1	1	1	1	1	$8x10^0$	$4x10^0$	$2x10^0$	$1x10^0$
4	1	1	1	0	1	1	1	1	1	$8x10^1$	$4x10^1$	$2x10^1$	$1x10^1$
5	1	1	1	1	0	1	1	1	1	$8x10^2$	$4x10^2$	$2x10^2$	$1x10^2$
6	1	1	1	1	1	0	1	1	1	-	-	UR	DR
7	1	1	1	1	1	1	0	1	1	-	-	/FLAG	FLAG
8	1	1	1	1	1	1	1	0	1	"0"	DP1	POL	$1x10^3$
9	1	1	1	1	1	1	1	1	0	DP5	DP4	DP3	DP2

Fonte: Manual de Instrução do eletrômetro 616 (KEITHLEY, 1980).

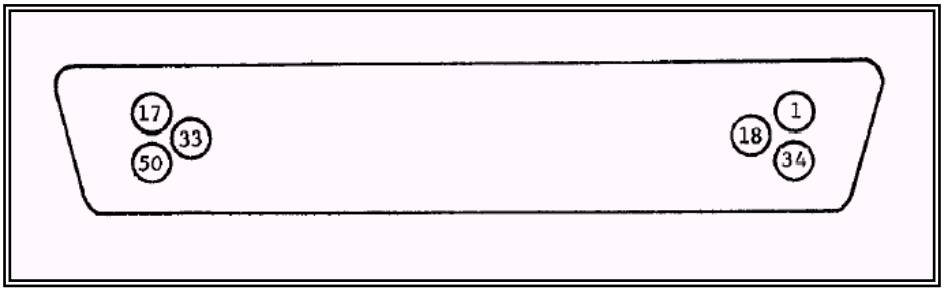


Figura 39– Conector DB50 fêmea, na traseira do 6162
Fonte: (KEITHLEY, 1980).

Um cabo foi confeccionado para conectar o eletrômetro à placa principal, a que contém o PIC16F877A, de acordo com o apresentado na Fig. 40.

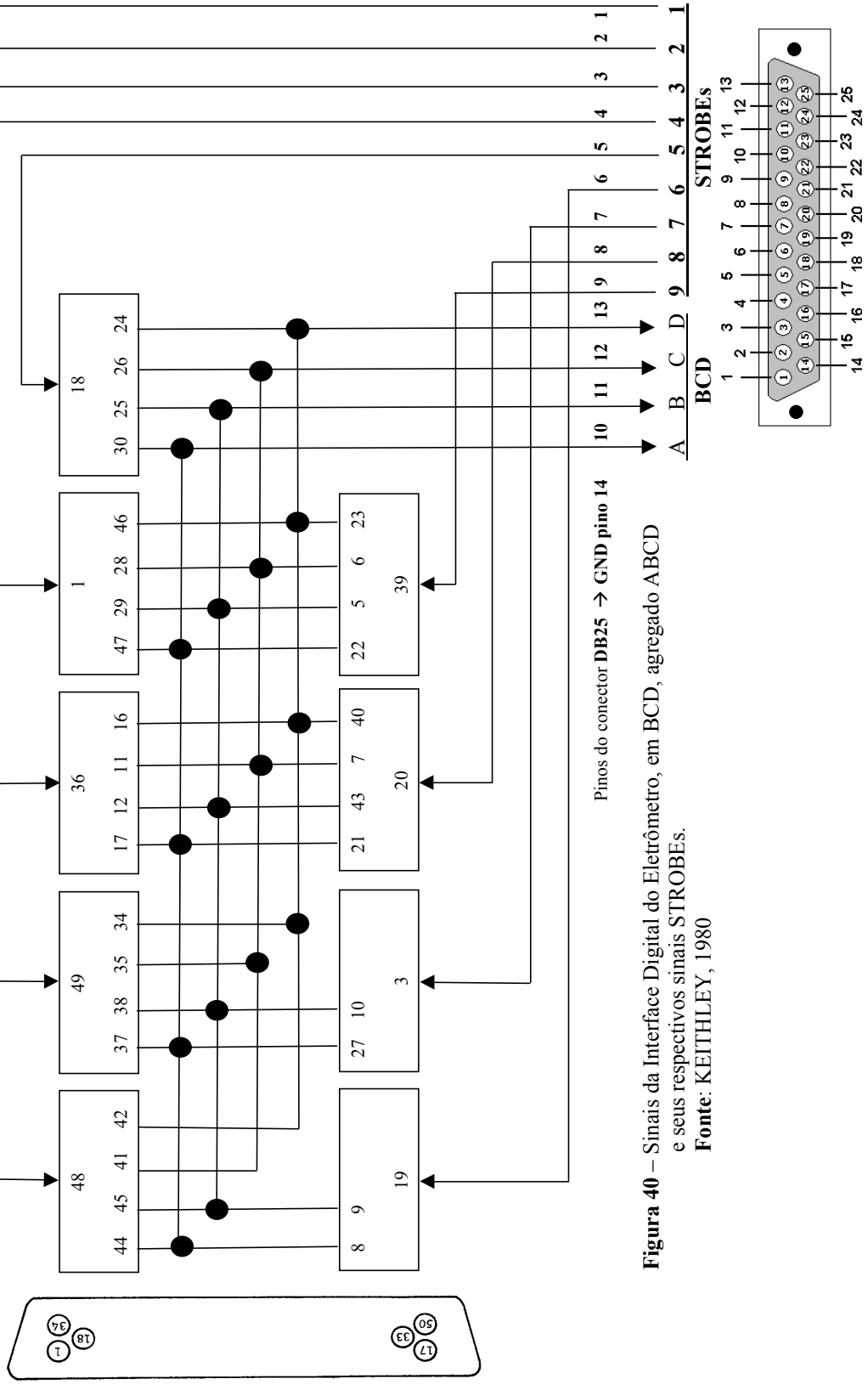


Figura 40 – Sinais da Interface Digital do Eletrômetro, em BCD, agregado ABCD e seus respectivos sinais STROBEs.
Fonte: KEITHLEY, 1980

A Fig. 41 ilustra como ficou o cabo de interconexão do eletrômetro à PCI principal. Numa extremidade do cabo foi instalado um conector DB50 macho que deve ser conectado ao DB50 fêmea do eletrômetro, e na outra extremidade foi instalado um conector DB25, também macho, para ser conectado ao DB25 fêmea instalado na PCI que contem o PIC16F877A.

Figura 41 – Cabo de interconexão do eletrômetro à PCI com o PIC16F877A.

Para que o software *assembly* realize uma leitura, sem erro, dos dados disponibilizados pelo conector DB50, deve-se, de início, interpretar os sinais DP1 a DP5 quando o STROBE 8 e 9, não simultaneamente, estiverem ativos – Tab. 34. Dada essa necessidade, para uma melhor visualização e entendimento do software desenvolvido, é que se desmembrou as possibilidades de

apresentação dos dados nos displays do equipamento 616, possibilidades estas mostradas na Fig. 42, que varia a posição de cada ponto decimal (DP).

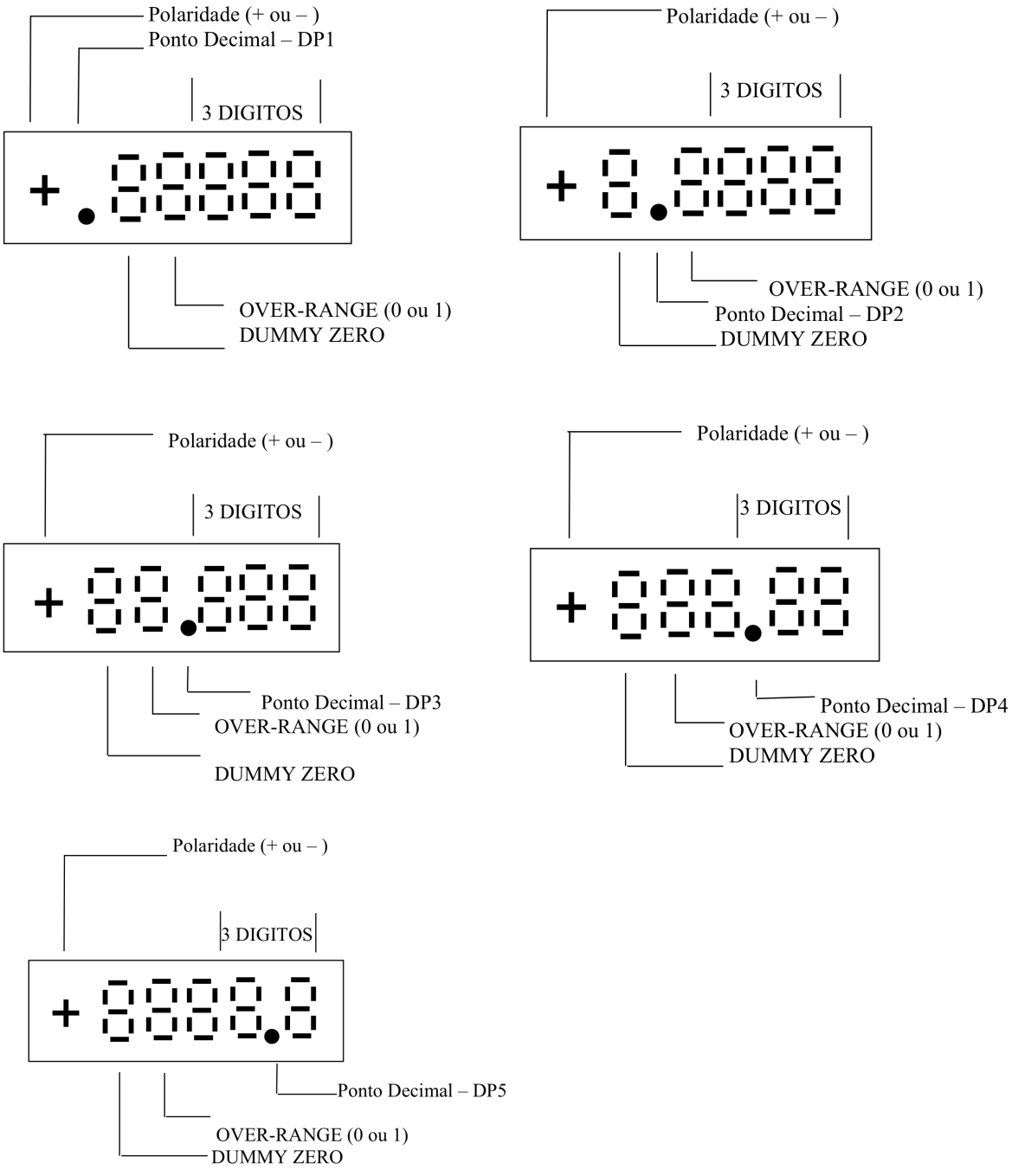


Figura 42 – Posições dos DP1 a DP5 no display do eletrômetro 616.

O manual do equipamento 616 orienta o seu usuário quanto as saídas digitais presentes no conector DB50 do equipamento 6162 (Tab. 34 e Fig. 40). Estas saídas digitais são saídas em coletor aberto e representam, no código BCD (8421), cada um dos três dígitos (STROBEs 3, 4 e 5, quando ativos) do display de sete segmentos do equipamento 616 (Fig. 42), a exemplo do zero (0), em decimal, ser representado como quatro zeros (0000), em BCD. Ainda encontram-se presentes nessas saídas digitais os sinais *over-range* (ou $1x10^3$), *up-range* (0) ou UR, polaridade (+ = 1), o Ponto Decimal ou DP (5 pinos do DB50), os bits de expoentes (5 pinos do DB50), a polaridade do expoente (+ = 1), o bit *downrange* (0) ou DR, identificador de *zero check* (1), e bits de função (código de 2 bits), todos estes sinais encontram-se na Tab. 34, (KEITHLEY, 1980).

Essas saídas foram agrupadas na confecção do cabo (Fig. 41) conforme as ligações apresentadas na Fig. 40. Isto pode ser feito somente porque as referidas saídas são em coletor aberto (Fig. 43) e ativadas, não simultaneamente, por sinais STROBEs (Tab. 34). Sendo as saídas de um CI (ou as do conector DB50) construídas em coletor aberto, elas necessitam, para funcionarem adequadamente, que sejam conectadas a um resistor ligado ao terminal positivo da fonte de alimentação (+5 Vcc), conhecido por resistor *pull-up* (MALVINO, 1985). Se esse resistor fosse ligado ao terminal GND da fonte de alimentação, ele seria chamado por *pull-down*.

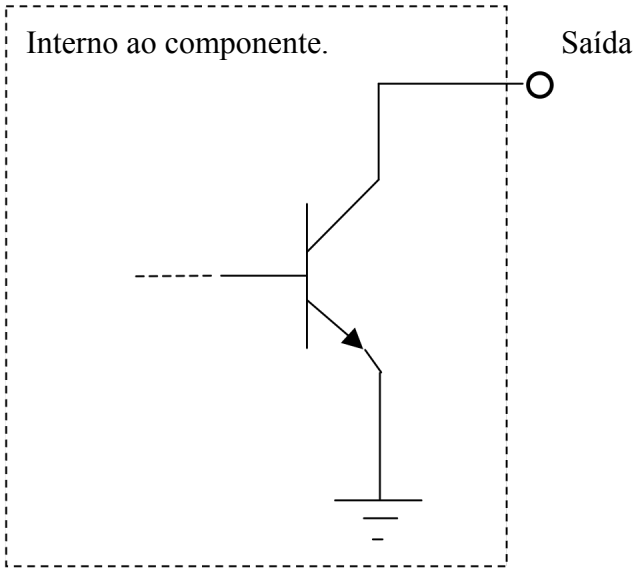


Figura 43 – Representação de uma saída em coletor aberto.

Para que haja a transferência de dados do eletrômetro ao PIC, foi necessário incluir resistores *pull-up* em cada uma das linhas de comunicação. Deve-se observar que estas linhas para transferência de dados ao PIC foram ligadas à PORTB do PIC e que esta porta já contém *pull-ups* internos, por configuração. À época de tomar esta decisão não se alertou para este fato e a PCI foi construída, também, com *pull-ups* externos, e esse projeto foi implementado utilizando os *pull-up*s externos.

A Fig. 44 mostra uma das saídas do eletrômetro ligada a uma das entradas da placa principal (a que contém o PIC16F877A) através de uma das linhas do cabo de conexão (Fig. 41).

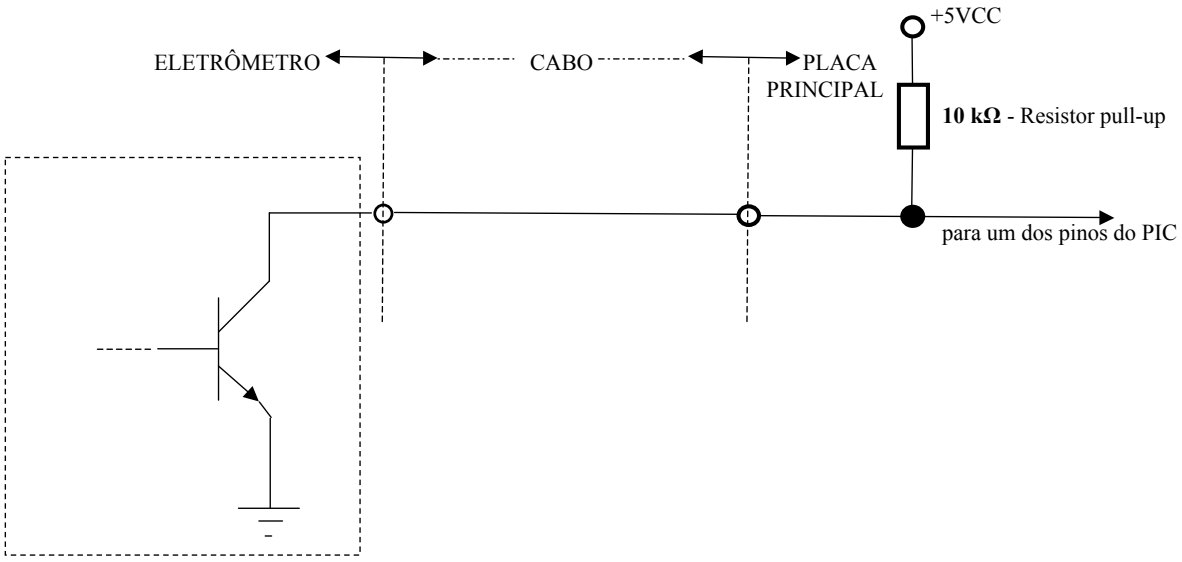


Figura 44 – Interconexão de uma saída em coletor aberto à placa principal.

A Tab. 35 mostra todas as interconecções (pinos) desde o microcontrolador PIC16F877A até o eletrômetro. Além disso esta tabela também ilustra, juntamente com a Fig. 45 e Fig. 46, em qual resistor *pull-up*, e respectivo *jumper* de ligação, cada conexão foi ligada.

Tabela 35 – Caminho físico do PIC ao eletrômetro

Sinal	Pino PIC	Pino Pull-up	Pino conector CN-4 Placa Principal	Pino do conector DB25 - Cabo	Pino do Conector DB50 - Cabo
Strobe 1	19	9	3	1	48
Strobe 2	20	8	5	2	49
Strobe 3	21	7	7	3	36
Strobe 4	22	6	9	4	1
Strobe 5	27	5	11	5	18
Strobe 6	28	4	13	6	19
Strobe 7	29	3	15	7	3
Strobe 8	30	2	17	8	20
Strobe 9	17	10	1	9	39
A (-) Sig	33	17	19	10	8 (**)
B	34	1	21	11	5 (**)
C	35	11	23	12	6 (**)
D (+)Sig	36	12	25	13	16 (**)
GND	12	NC(*)	14	14	4

(*) NC = Não Conectado.

(**) Foram conectados juntos os pinos do DB50: A = 8, 17, 21, 22, 27, 30, 37, 44, 47,
B = 5, 9, 10, 12, 25, 29, 38, 43, 45,
C = 6, 7, 11, 26, 28, 35, 41,
D = 16, 23, 24, 34, 40, 42, 46,

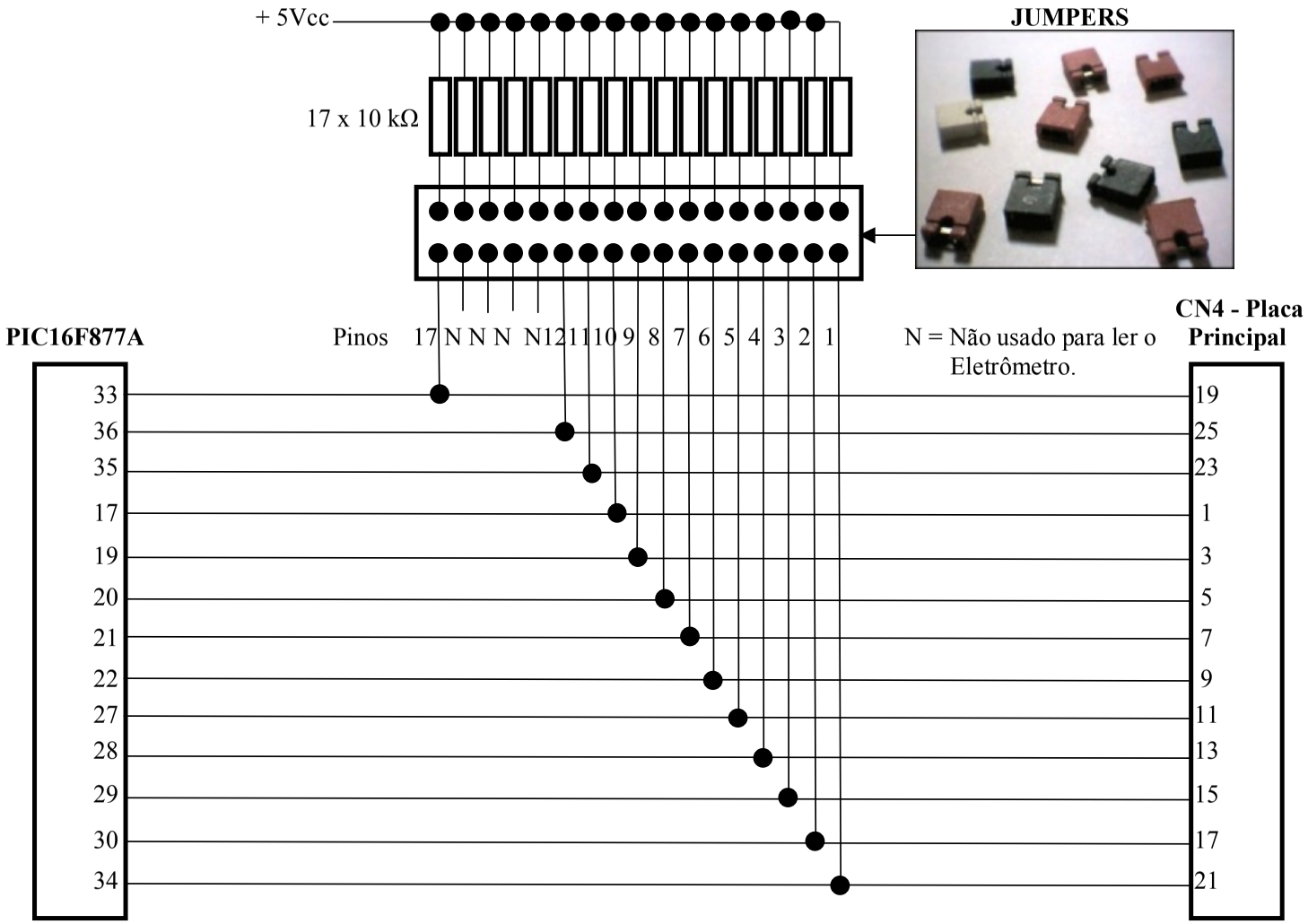


Figura 45 – Circuitos internos à Placa Principal para leitura do eletrômetro.

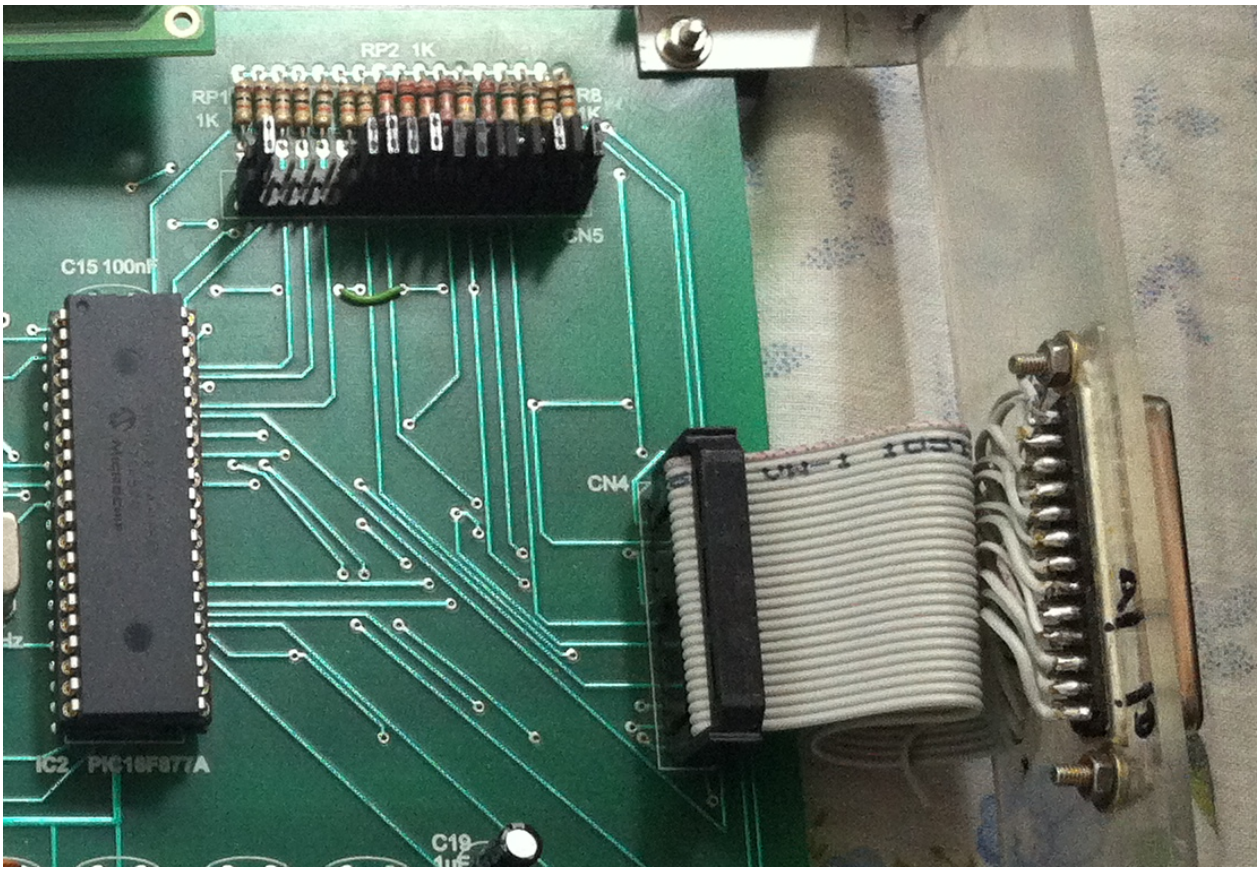


Figura 46 – Implementação da Fig. 45 na PCI.

Para otimizar o armazenamento dos dados na memória serial 24LC1025, optou-se por agrupar duas leituras subseqüentes em um único byte, e só então realizar o armazenamento em única posição endereçada dessa memória. Assim, os dados liberados pelo *strobe* 1 e 2, formam o primeiro byte a ser armazenado, o *strobe* 3 e 4, o segundo, o *strobe* 4 e 6, o terceiro, o *strobe* 7 e 8, o quarto, e por fim, o *strobe* 9, seus 4 bits são guardados no quinto byte. Portanto, 5 bytes são necessários para guardar uma única medida do eletrômetro. Além da medida propriamente dita foi guardado, também, o valor das variáveis X e Y, totalizando 7 bytes por medida. Como são 3.731 medidas, todo o projeto precisa de no mínimo 26.117 bytes (menos de 26kBytes) para todo o projeto, o que torna o CI de memória 24LC1025 (128kBytes) mais que suficiente para atender esse projeto. Nessa linha de raciocínio, a Tab. 36 mostra a composição dos registradores ST_1_2 até o ST_9 com todos os dados (bits) que compõem única leitura de uma medida do eletrômetros.

Tabela 36 – Composição do registrador ST_1_2 a ST_9

Registra-dor	Sinal associado à Leitura do Eletrômetro							
	b_7	**b_6**	**b_5**	**b_4**	**b_3**	**b_2**	**b_1**	**b_0**
ST_1_2	Exp. $1x10^1$	Sinal Exp.	F_2	F_1	Exp. $8x10^0$	Exp. $4x10^0$	Exp. $2x10^0$	Exp. $1x10^0$
ST_3_4	Unidade (8)	Unidade (4)	Unidade (2)	Unidade (1)	Dezena (8)	Dezena (4)	Dezena (2)	Dezena (1)
ST_5_6	Centena (8)	Centena (4)	Centena (2)	Centena (2)	X	X	UR	DR
ST_7_8	/Flag	Flag	X	X	Dummy Zero	DP1	Polari-dade	Overrange $1x10^1$
ST_9	X	X	X	X	DP5	DP4	DP3	DP2

Por fim, o detector de radiação, que é um dispositivo construído para converter a energia da radiação em um sinal elétrico que possa ser medido é mostrado na Fig. 47. Nesse trabalho utilizou-se um detector gasoso, especificamente, uma câmara de ionização.

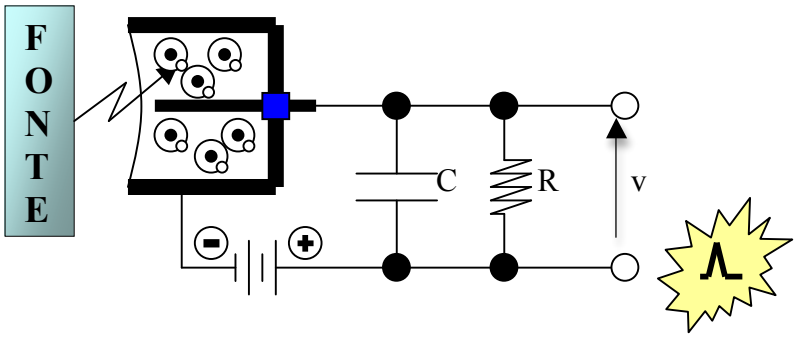


Figura 47 – Diagrama representativo de um detector a gás.

Quando a radiação interage com o gás contido no interior da câmara, volume sensível, ela provoca ionizações neste gás. Aplicando-se uma diferença de potencial entre os eletrodos do detector os íons formados migram para os respectivos pólos, gerando uma corrente elétrica. O

circuito RC (resistor-capacitor, externo à câmara) converte o sinal elétrico gerado em um pulso mensurável. A Fig. 47 ilustra a detecção do sinal gerado para ser encaminhado e medido pelo eletrômetro.

CAPÍTULO 4 – RESULTADOS E CONCLUSÕES

4.1 – Resultados

Todas as etapas desenvolvidas nesse projeto (mecânica, hardware e software em *assembly*), foram realizadas a contento. Quanto a parte mecânica, ela precisa de uma maior atenção, principalmente sobre os quesitos atrito e posicionamento do relé *reed*, que é o detector sinalizador da posição inicial para a fonte radioativa. Esta detecção deve acontecer sempre no mesmo local, sem variação de posição. Se assim for deve facilitar o acerto do número de passos rumo ao volume sensível da câmara de ionização, esta é uma condição necessária.

O relé *reed* é um contato construído dentro de uma ampola de vidro hermeticamente lacrada que quando um campo magnético (imã) aproxima desse contato ele, normalmente aberto, se fecha. O posicionamento desse relé, nesse projeto, deve ser de forma a não alterar, em cada posição da mesa X, a detecção do ponto zero para a cabeça onde reside a fonte radioativa. Este posicionamento, portanto, é uma condição crítica e o relé *reed* deve ser ajustado de forma a detectar o mesmo início, o mesmo ponto zero, para todas as posições da mesa X.

A infraestrutura mecânica de todo o projeto desenvolvido é mostrada nas Fig. 48 e Fig. 49. Além de outras coisas, ela é composta de duas mesas. Sobre cada mesa edificou uma torre, uma denominada por X – que transporta a fonte radioativa – e a outra por Y – que transporta o detector de radiação. Pode-se observar que tanto a mesa X (ou torre X) quanto a mesa ou torre Y são acionadas por motores de passo acoplados diretamente a respectiva haste rosqueada, onde em cada mesa móvel a haste é conectada a um sistema fixado em sua mesa.. A Fig. 48 também mostra a PCI com o PIC, hardware que suporta o controle de todo esse projeto, e mostra também a posição da fonte de alimentação.

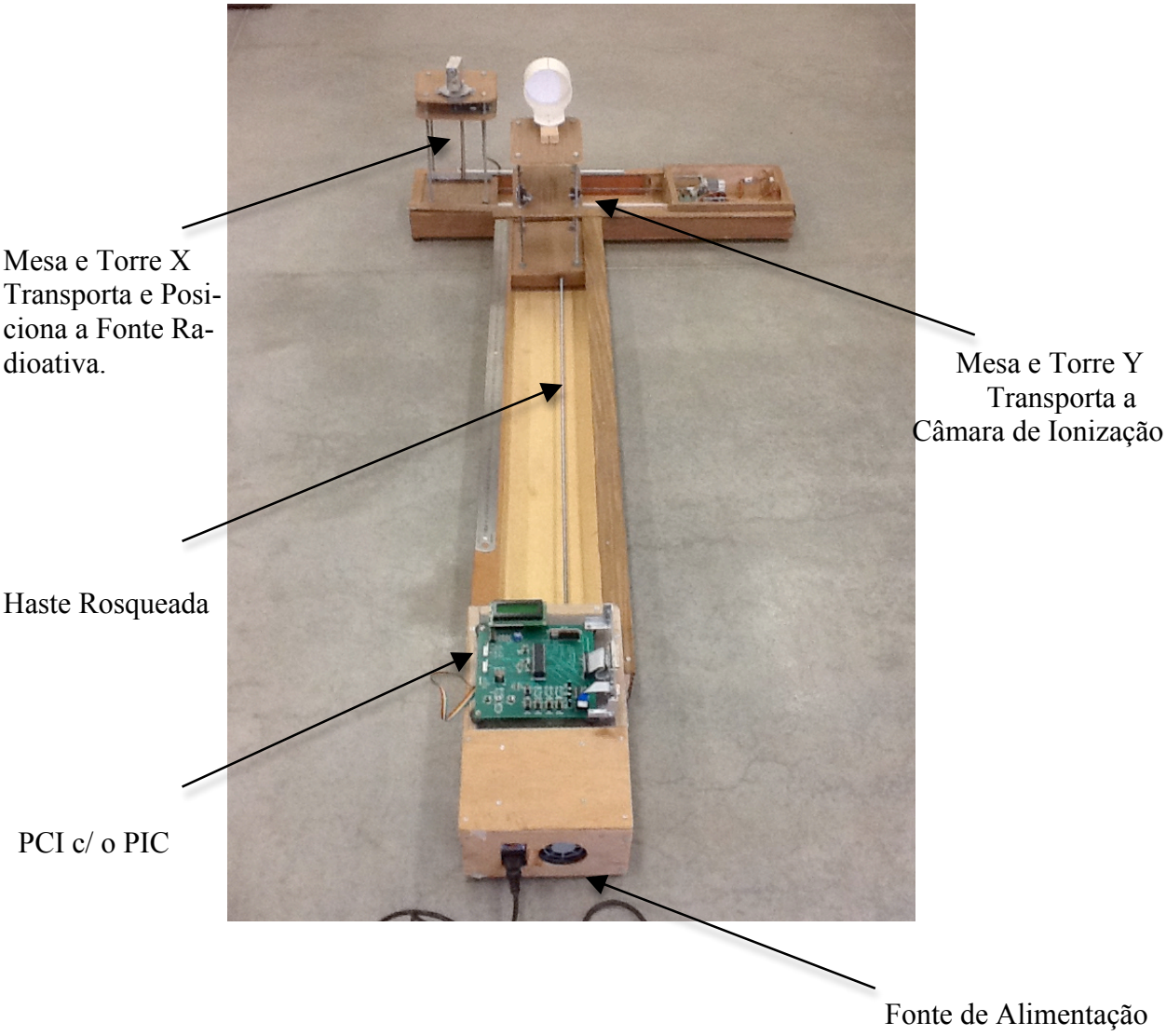


Figura 48 – Visão geral do projeto – vista traseira.

A Fig. 49 apresenta a vista frontal do projeto físico com melhor visualização que a Fig. 48. Nesta imagem pode verificar que são simuladas a posição física onde deverá fixar a câmara de ionização (detector de radiação) e também a posição da fonte radioativa. Além disso, a Fig. 49 mostra o motor de passo que aciona a mesa X e a escala, graduada em centímetros, que indica a

posição da mesa X quando em movimento, esta parte de -20,0 cm (posição inicial), passa pelo zero e evolui até a posição +20,0 cm. Esta é a trajetória completa da mesa X para uma determinada posição da mesa Y. Esta trajetória de X se repete por 91 vezes, que é a variação da mesa Y, ou seja, Y varia de 2,0 a 92,0 cm.

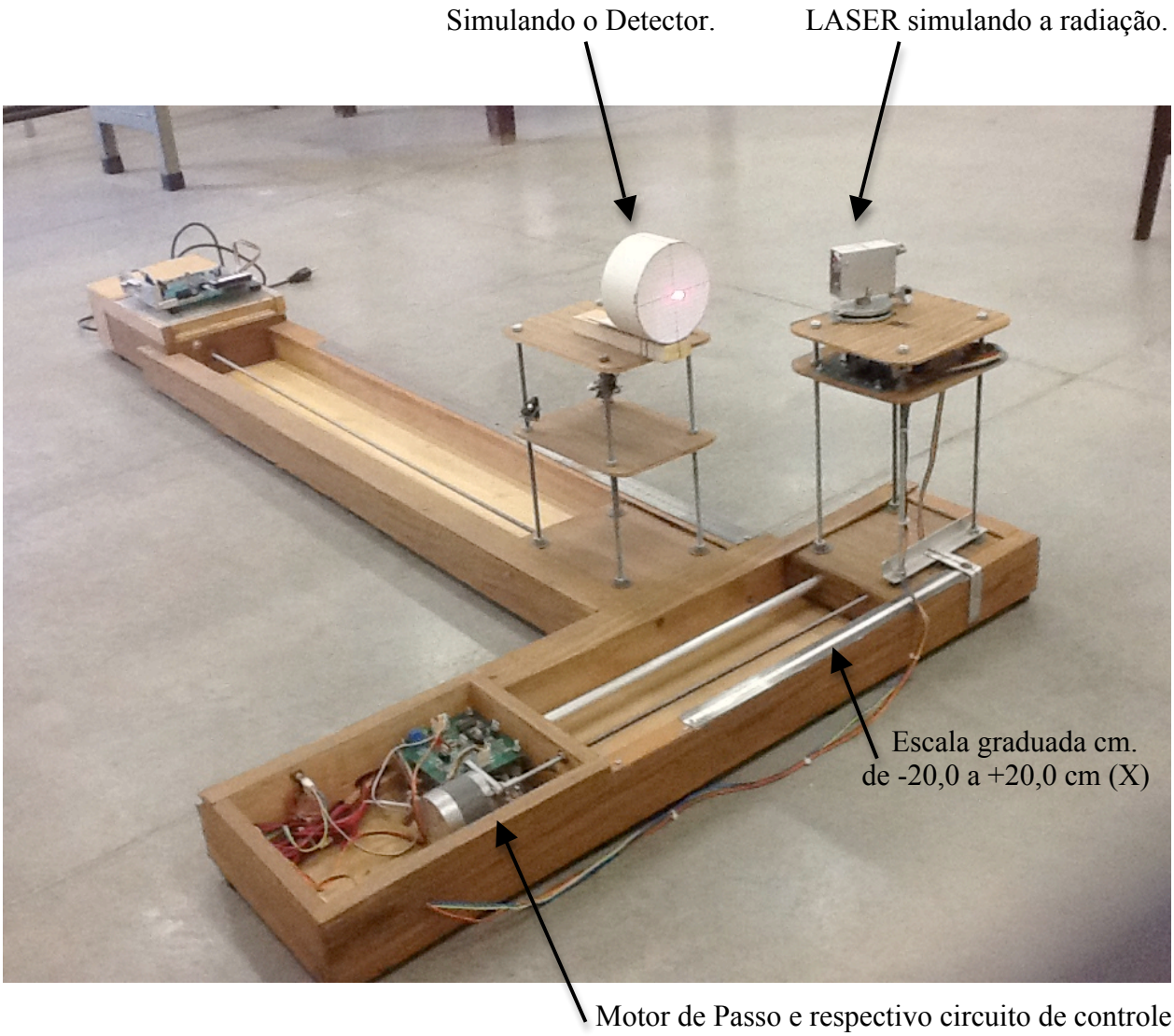


Figura 49 – Visão geral do projeto – vista frontal.

A eletrônica associada a esse projeto baseou-se no microcontrolador PIC16F877A, Fig. 50, o qual foi programado em linguagem *assembly*. As sub-rotinas desenvolvidas para esse projeto estão listadas na Tab. 37 e apresentadas no ANEXO A.

A primeira coluna da Tab. 37 é a do endereço inicial (em hexadecimal). Após a conversão do código fonte em código executável pelo software MPLAB da Microchip, este alocara cada uma das sub-rotinas a partir de determinado endereço inicial. Estes endereços referem-se à memória de programa do PIC. O nome da sub-rotina, ou *label* para sua chamada, encontra-se na segunda coluna da Ta. 37. E, por fim, a última coluna desta tabela apresenta um pequeno descritivo do que faz cada sub-rotina.

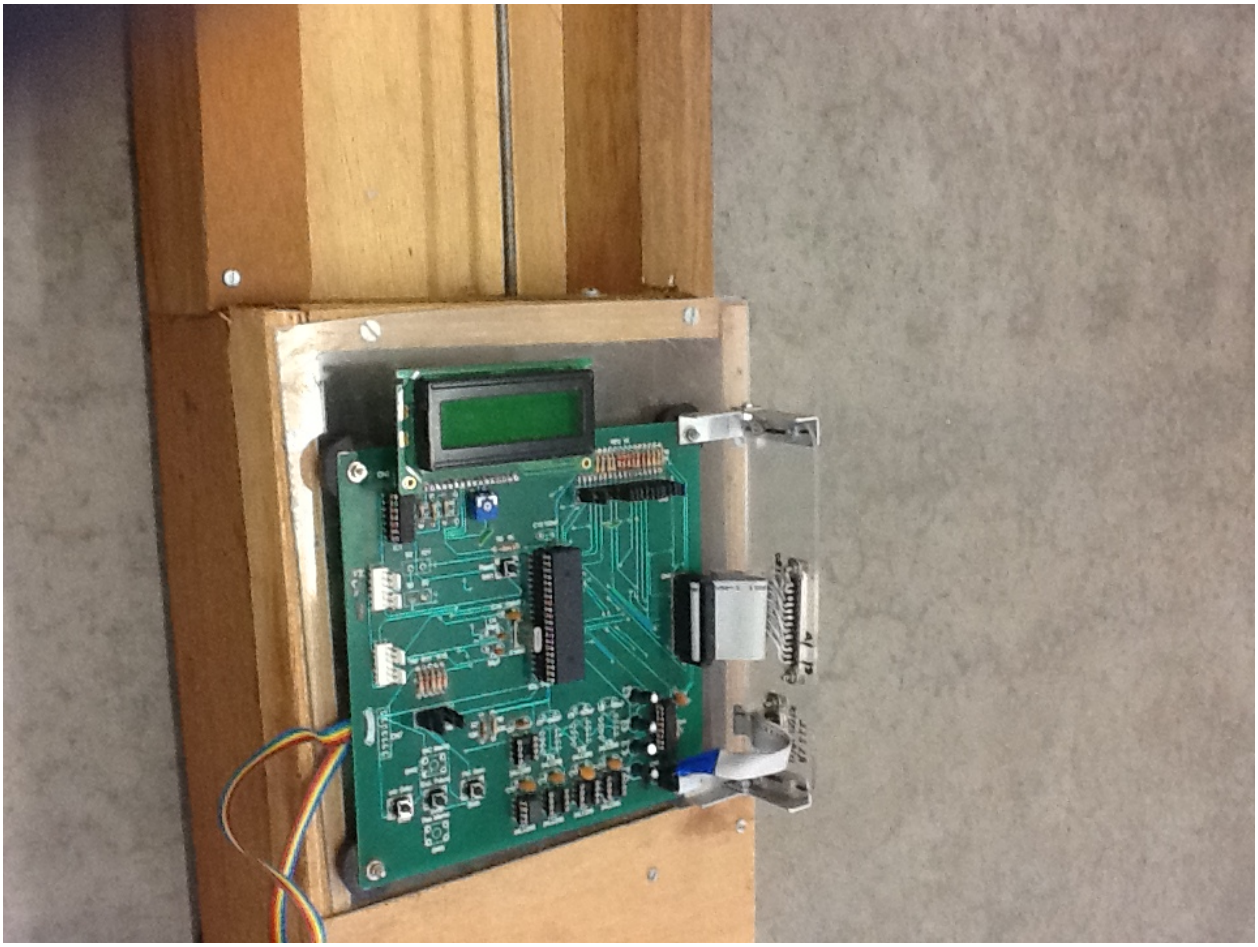


Figura 50 – Imagem completa da PCI com o PIC.

Tabela 37 – Conjunto das sub-rotinas desenvolvidas

End. (h)	Nome da sub-rotina	Execução aproximada
0003	HEXA_ASCI	Converte *nibble* inferior de W em seu valor ASCII.
0015	CONVERTE_X_BCD	Converte X de binário para BCD.
0040 ao 0F8E	Y_2 a Y_92	91 Tabelas para os passos de ajuste da fonte radioativa rumo ao centro geométrico do detector.
0F8F	BIN_BCD	Converte um número binário em dois *nibbles* BCD.
1000	ACHA_PASSO	Encontra quantos passos tem de ser dado para ajustar a cabeça. Trabalha conjuntamente com Y_2 a Y_92.
1396	Y_ASCII	Converte Y de binário para ASCII.
13A4	SINAL_X_ASCII	Converte X e seu sinal para ASCII.
13BC	Passo_rapido_0a90	Move mesa Y mais rápido à posição 0.
13FD	DELAY_MOTOR	Atraso para controle passos do motor de passo.
1412	UM_MM_XA	Avança ou retroage 1 mm em X.
141E	UM_MM_YA	Avança ou retroage 1 mm em Y.
142A	UM_CM_Y	Avança ou retroage 1 cm em Y.
1430	UM_CM_X	Avança ou retroage 1 cm em X.
1436	UMA_VOLTA_CABECA	Dá uma volta completa ajuste cabeça.
1442	DELAY_MS	Atrasa 1 ms.
144C	DELAY_3S	Atrasa 3 s.
144F	DELAY_5S	Atrasa 5 s.
1464	POS_DP1	Ajuste ponto decimal DP1.
146B	POS_DP2	Ajuste ponto decimal DP2.
1472	POS_DP3	Ajuste ponto decimal DP3.
1479	POS_DP4	Ajuste ponto decimal DP4.
147E	POS_DP5	Ajuste ponto decimal DP5.
1486	LE_PORTB	Lê porta B, dados do eletrômetro.
1498	ESCREVE	Escreve um byte no LCD.
14AF	LCD_LINHA_1	Envia dados para a linha 1 do LCD.
14D4	LCD_LINHA_2	Envia dados para a linha 2 do LCD.
14F9	Mens_1	Mensagem 1 para o LCD.
153C	Mens_2	Mensagem 2 para o LCD.
157F	Mens_3	Mensagem 3 para o LCD.
15C2	Mens_4	Mensagem 4 para o LCD.
15E3	Mens_5	Mensagem 5 para o LCD.
1626	Mens_6	Mensagem 6 para o LCD.
1649	Mens_7	Mensagem 7 para o LCD.

168C	Mens_8	Mensagem 8 para o LCD.
16CF	Mens_9	Mensagem 9 para o LCD.
1702	ESPERA_I2C_DESOCUPAR	Auxiliar de leitura/gravação EEPROM.
170B	TX_LE_ACK	Idem.
1713	TX_START_BIT_I2C	Idem.
1719	TX_RESTART_BIT_I2C	Idem.
171F	TX_STOP_BIT_I2C	Idem.
1725	TX_NACK_I2C	Idem.
172A	TESTAR_ACK_I2C	Idem.
1732	ACERTA_CONTOLE_ENDERECO_I2C	Idem.
1747	INICIA_CONTROLE_END	Idem.
174D	HIGH_ENDERECO_LOW_ASCII	Idem.
175F	ACERTA_A1_A0_B0_ASCII	Idem.
176C	LER_EEPROM_I2C	Idem.
178D	ESCREVER_EEPROM_I2C	Idem.
17FA	LEVA_YX_0	Controla mesas Y e X até posições 0.
17FE	PROCESSO_AUTOMAT	Controle todo o processo automático.
1863	LE_MEDIDA	Lê a medida no eletrômetro.
18BD	CONVERTE_ST_ASCII	Converte os dados de um *Strobe* para ASCII
198F	GRAVA_MEDIDA_EEPROM	Grava dado sequencialmente na EEPROM
19BC	LE_EEPROM_SERIAL	Lê dado gravado na EEPROM
1A88	TX_DADOS_RS232	Transmite dados pata um computado pessoal

As sub-rotinas apresentadas na Tab. 37 auxiliam o programa principal. O programa principal suporta o que já foi apresentado na Fig. 26 (fluxograma básico para automação das medidas do eletrômetro). No devido momento o programa principal chama uma das sub-rotinas, a qual resolve um específico problema, que no conjunto dá o contorno ao projeto.

Nesse projeto foi utilizado algumas rotinas para o controle de três motores de passo – um destes move a mesa X, outro a Y e outro move a cabeça com a fonte radioativa. Então, o programa principal, ao chamar estas sub-rotinas, consegue posicionar devidamente a cabeça com a fonte radioativa rumo ao volume sensível da câmara de ionização. Uma vez estando elas bem posicionadas, o programa espera tempo suficiente para a medida estabilizar no eletrômetro.

Cumprido este tempo o programa lê os dados da medida presentes na porta digital do eletrômetro (conector DB50) e a salva em endereços adjacentes na memória serial 24LC1025. Para cada medida são necessárias 7 (sete) posições físicas endereçadas dessa memória. Uma vez que a medida foi salva, o programa ajusta novamente as mesas X e Y para a realização de nova medida. Isto acontece por 3.731 vezes, momento em que todas as medidas estarão residentes na memória 24LC1025.

A Tab. 37 mostra todas as sub-rotinas que foram desenvolvidas. Com estas sub-rotinas foi possível, o controle das 91 tabelas (estas guardam, para determinado Y e X, o número de passos que devem ser dados para ajustar o feixe radioativo rumo ao volume sensível da câmara de ionização); as leituras do teclado de três chaves, uma para comandar as mesas até suas posições iniciais, a outra para a automação e a última para a transmissão de dados ao CP, pela RS-232; as leituras do eletrômetro; as conversões de diversos dados para adequá-los às diversas situações do projeto; o armazenamento e leitura da memória serial 24LC1025 (EEPROM). Para a transferência da massa de dados gerada ao computador foi utilizada a interface RS-232. Esta transferência acontece no modo texto tabulado, como ilustra o ANEXO B, o que facilitou a migração do referido texto a uma planilha eletrônica. Os dados, uma vez na planilha eletrônica, podem ser tratados estatisticamente e extraído deles a dose e/ou taxa de dose a diversas distâncias fonte-detector.

Foi construído, portanto, um sistema automatizado que posiciona fonte-detetor, que mede com um eletrômetro as cargas liberadas dessa interação, que armazena a medida em memória estática, e que transmite a massa de dados gerada ao CP. Uma vez que a massa de dados esteja disponibilizada em uma planilha eletrônica é possível calcular a dose absorvida e/ou taxa de dose fornecida por uma fonte radioativa.

4.2 – Conclusões

O objetivo principal desse projeto foi alcançado. Foi construído uma estrutura mecânica que trabalha ortogonalmente, a luz dos eixos cartesianos X e Y. Em cada um desses eixos foi construído uma mesa móvel e sobre cada mesa uma torre. Sobre a torre X foi instalado um motor de passo para acerto da cabeça com a fonte radioativa rumo ao centro geométrico do detector

(volume sensível), este detector deve ser instalado sobre a torre Y. O movimento de ambas as torres ocorre a passos de 1,0 cm em X e em Y, e são acionadas por motores de passo independentes.

Foi confirmado que é possível a automação das 3.731 medidas de cargas elétricas, estas geradas no interior de uma câmara de ionização e fornecidas por um eletrômetro. A leitura, o tratamento das medidas realizadas pelo eletrômetro e o seu devido armazenamento em memória serial apresentaram de acordo com o fim almejado.

O ponto que merece atenção refere-se à mecânica que suporta o projeto. Esta mecânica precisa ser elaborada com maior rigor, pois busca-se por um sistema de posicionamento com maior precisão em X e Y, e a mecânica conseguida não atingiu essa precisão. Isto impactou negativamente no sistema de posicionamento, primeiro porque ao girar a cabeça com a fonte radioativa o relé *reed* acionava-se em distintos instantes, dificultando a fixação da referência do zero deste posicionamento. Esta referência não pode mudar pois é a posição inicial para a contagem dos passos que o motor de passo deve executar para mover a cabeça com a fonte radioativa rumo ao centro geométrico do detector (volume sensível).

Os referidos passos são buscados em 91 tabelas especialmente construídas para suportarem este ponto do projeto. Todo o sistema microcontrolado foi testado a contento. A exemplo da correta leitura de todas as posições das 91 tabelas, sem nenhum erro. Isso ocorreu dada a atenção dispensada em relação à carga dos registradores PCLATH e PCL, melhor tratados **no** item 2.1.5.1 (Memória de Programa) e item b8 do ANEXO A (sub-rotina ACHA_PASSO) pois se assim não fosse feito ocorreria saltos indevidos que inviabilizariam todo o projeto, e sem dúvida, foi um ponto frágil do projeto.

Os procedimentos de controle dos motores de passo, das leituras e gravações de dados na memória serial 24LC1025 (EEPROM), das interfaces máquina-homem (LCD e as transmissões de dados via interface RS-232 para um CP), da leitura e do tratamento dos dados liberados pelo eletrômetro (das medidas de carga elétrica), e das leituras das chaves de funções a executar (do teclado), foram realizadas e testadas e é o que permite a automação do laboratório de calibração do Departamento de Engenharia Nuclear (DEN) da UFMG. Tudo isso funcionou como esperado.

Portanto, com pequenos ajustes pode-se chegar a um preciso sistema de posicionamento fonte-detector, todo automático, com registro, em memória estática, das medidas de carga realizadas por um eletrômetro. O conjunto das 3.731 medidas dá condição para se trabalhar

estatisticamente os dados e conhecer a dose e a taxa de dose a diversas posições relativas entre a fonte e o detector.

4.3 – Perspectivas futuras

É claro que o presente trabalho deixa margem às melhorias.

No que tange à mecânica ela deve ser edificada de forma a minimizar o atrito que dificulta o devido posicionamento das mesas X e Y e, também, as hastes rosqueadas podem ser construídas especificamente para esse projeto, e nesse caso deveriam ter passo de 1,0mm.

No que tange a eletrônica poderia substituir o relé *reed* por detectores fotoelétricos e, assim, verificar se a posição zero para a cabeça contendo a fonte radioativa mantém-se constante. Isto alcançado torna-se factível o acerto das 91 tabelas, onde residem a quantidade de passos para ajuste dessa cabeça a cada posição de Y e de X. Outro quesito que pode ser melhorado diz respeito ao passo dos motores de passo. Nesse trabalho foi utilizado passo completo, onde cada passo move $1,8^{o}$ mecânicos, e poderia melhorar ainda mais a precisão se trabalhasse com meio passo, ou $0,9^{o}$ mecânicos. Para isso deve ser desenvolvido novo *driver* que continue usando os mesmo pinos de I/O do PIC, ou sejam dois pinos, um para direção (giro horário ou anti-horário) e outro para *clock* do *driver*.

No que tange a programação *assembly*, ela foi desenvolvida para único perfil de trabalho, o qual comporta só um tipo de câmara de ionização. Porém, pode-se melhorar o sistema desenvolvido caso nele possa configurar qualquer perfil de trabalho, este, claro, associado a certo tipo de detector e tipo de eletrômetro. Para isto, como sugestão, pode-se incluir mais uma chave no teclado e sua função ser apresentada no *menu* ao ligar a máquina. Por exemplo, esta função poderia ser chamada de "Ajuste de Perfil". Esta chave ao ser pressionada faz o programa principal saltar para uma rotina de tratamento de perfil de trabalho (entrada de dados referentes ao detector e eletrômetro que serão usados). Para ajuste do perfil de trabalho cada característica desse detector e/ou eletrômetro deve ser apresentada no *display* LCD e à frente da característica, cursor piscando, deverá o operador entrar com um valor para a referida característica. Deste modo, quem for utilizar a máquina, entrará com os dados mais relevantes sobre o detector e/ou eletrômetro que será utilizado, a exemplo do tempo de espera (*delay*) necessário para que o

eletrômetro estabilize a medida. Só após este tempo é que a medida do eletrômetro poderá ser armazenada na memória serial 24LC1025 (EEPROM).

Implementadas estas melhorias terá construído um equipamento mais flexível em termos de utilização e com maior precisão, esta além da que foi conseguida com o trabalho original.

REFERÊNCIAS

BARRADAS, O.; BEVAN, Frederick W.. **Telecomunicações, Sistemas Telegráficos –** EMBRATEL, 1ª ed. Rio de Janeiro: Livros Técnicos e Científicos, 1981. 931p.

CYPRIANO**,** Luiz Benedito; CARDINALI, Paulo Roberto. **Microprocessador Z80, Hardware**, Vol. 1, 3ª ed. São Paulo: Érica, 1983, 170p.

CYPRIANO**,** Luiz Benedito. **Z80 - Software**, Vol. II, 3ª ed. São Paulo: Érica, 1984, 322p.

FONSECA**,** Abner P.. **Controle de posicionamento de um detector usando circuito eletrônico microcontrolado e programação *assembly***. 2008, 106f.. Dissertação (Mestrado) Universidade Federal de Minas Gerais – UFMG, Belo Horizonte, Disponível em: <http://www.dominiopublico.gov.br/pesquisa/DetalheObraForm.do?
select _action=&co_obra=141188> Acesso em 31 jul. 2011.

GOMES, Alcides Tadeu, **Telecomunicações, Transmissão Recepção –** AM-FM Sistemas Pulsados. 3ª ed. São Paulo: Érica, 1985, 457p.

KNOLL, G. F. **Radiation Detection and Mensurement**. 2ª ed. New York: John Wiley & Sons, 1989, 754p.

KEITHLEY Instruments, Inc.. **Instruction Manual Digital Electrometer Model 616**, 9ª ed. Ohio, USA: 1980, 79p. Disponível em: <http://www.qsl.net/k/k0ff/7Manuals/Keithly%20Electrometers/28039I(Model616).pdf> Acesso em 9 jul. 2011.

MALVINO, Albert Paul. **Microcomputadores e Microprocessadores**. Tradução: Anatólio Laschuk, Revisão técnica: Rodrigo Araês Caldas Farias. 1ª ed. São Paulo: McGraw-Hill do Brasil, 1985, 578p. Título original: An Introduction to Microcomputers.

MALDONADO, Paulo César. **Z80 Aplicações Práticas**, 1a ed. São Pulo: Érica, 1990, 188p.

MELO, Mairton. **Eletrônica Digital**, 1ª ed. São Paulo: MAKRON Books do Brasil, 1993, 414p.

MICROCHIP, Fabricante, ***Section 29*. Instruction Set,** 1997, 48p. Disponível em: <http//www.microchip.com> Acesso em 7 jan. 2013.

MICROCHIP, Fabricante, ***AN774 – Asynchronous Communications with the PICmicro USART*,** 2000a, Disponível em: <http//www.microchip.com> Acesso em 28 dez. 2010.

MICROCHIP, Fabricante, ***AN556 – Implementing a Table Read,*** 2000b, Disponível em: <http//www.microchip.com> Acesso em 21 out. 2011.

MICROCHIP, Fabricante, ***Datasheet* PIC16F84A,** 2001, 88p. Disponível em: <http//www.microchip.com> Acesso em 7 jan. 2013.

MICROCHIP, Fabricante, ***MPLAB ICD 2 In-Circuit Debugger*,** 2002. Disponível em: <http://www.es.co.th/schemetic/pdf/icd-2.pdf> Acesso em 27 dez. 2013.

MICROCHIP, Fabricante, ***Datasheet* PIC16F877A,** 2003a, 234p. Disponível em: <http//www.microchip.com> Acesso em 21 out. 2011.

MICROCHIP, Fabricante, ***AN735 – Using the PICmicro MSSP Module for Master* I2C™ Communications,** 2003b, Disponível em: <http//www.microchip.com> Acesso em 31 jul. 2011.

MICROCHIP, Fabricante, ***Datasheet* PIC16F627A/628A/648A,** 2009, 180p. Disponível em: <http//www.microchip.com> Acesso em 21 out. 2011.

MICROCHIP, Fabricante, ***Datasheet* 24AA1025/24LC1025/24FC1025 - 1024K I^2C™ CMOS Serial EEPROM,** 2010, Disponível em: <http//www.microchip.com> Acesso em 28 dez. 2010.

MICROCHIP, Fabricante, ***Datasheet* PIC12(L)F1501,** 2011, 278p. Disponível em: <http//www.microchip.com> Acesso em 7 jan. 2013.

MICROELECTRONICS, General Instrument Corporation; **PIC1650 – PIC Series Microcomputer,** 1977, 13p.. Disponível em: <http://www.rhoent.com/pic16xx.pdf> Acesso em: 02/08/2012

MICROSOFT, **Programa hyperterminal** (htpe63.exe), 2012. Disponível em: <http://www.5star-shareware.com/Windows/WebDev/TelnetApplications/hyperterminal-download.html>Acesso em 30/05/2012.

MICROSOFT, **Instalar HiperTerminal**, 2012a. Disponível em: <http://technet.microsoft.com/pt-br/library/cc737746(WS.10).aspx)> Acessado em 3/06/2012.

MOSAICO Indústria e Comércio Eletro Eletrônico Ltda., **Guia do Usuário ICD2BR, *In Circuit Debugger***, 2011. Disponível em: <http://www.mosaico.com.br/Midias/Documentacao/Manual%20ICD2-BR_rev_13.pdf> Acesso em: 27/12/2013.

NICOLOSI, Denys E. C., **Laboratório de Microcontroladores Família 8051 –** Treino de Instruções, Hardware e Software, 4a ed. São Paulo: Érica, 2005, 206p.

OLIVEIRA, Ney Acyr R.; RUBENS, André Gil, **Microprocessador Z-80, Hardware**, 1ª ed. Rio de Janeiro: A e N – Consultoria Projetos e Publicações, 1983, 199p.

OLIVEIRA, Ney Acyr R.; RUBENS, André Gil, **Programando Z-80, Linguagem Assembly**, 1ª ed. Rio de Janeiro: Ciência Moderna Computação Ltda, 1986, 262p.

PEREIRA, Fábio, **Microcontroladores PIC:** Programação em C, 4a ed, São Paulo: Érica , 2003, 358p.

PEREIRA, Fábio, **Microcontroladores PIC** – Técnicas Avançadas, 4a ed, São Paulo: Érica, 2006, 359p.

SOARES, Luiz Fernando Gomes; LEMOS, Guido; COLCHER, Sérgio, **Redes de Computadores** – das LANs MANs e WANs às Redes ATM, 1ª ed. Rio de Janeiro: Campos, 1995, 576p.

SOUZA**,** David José de; LAVINIA, Nicolas Cesar, **PIC16F877A Conectando o PIC –** Recursos Avançados**,** 3ª ed. São Paulo: Érica, 2006, 375p.

SOUZA**,** David José de, **Desbravando o PIC – Ampliado e Atualizado para o PIC16F628A,** 9ª ed. São Paulo: Érica, 2005, 375p.

STRANGIO, Christopher, ***The RS232 Standard – A Tutorial with Signal Names and Definitions***, 19p., 1993, disponível em: <http://www.camiresearch.com/Data_Com_Basics/RS232_standard. html > Acesso em: 29/01/2012.

TANENBAUN, Andrew S., **Organização Estruturada de Computadores**. Tradução: Arlete Simille Marques. Revisão técnica: Wagner Luiz Zucchi. 5ª ed., São Paulo: Pearson Prentice Hall, 2007, 449p.

TAUB, Herbert, **Circuitos Digitais e Microprocessadores**. Tradução: Ivan José de Albuquerque, Fernando Fontes Barbosa. Revisão técnica: Rodrigo Araês Caldas Farias. 1ª ed. São Paulo: McGraw-Hill do Brasil, 1984, 510p.

TEXAS INSTRUMENTS**,** Fabricante – **Datasheet MAX232, MAX232I – Dual EIA-232 DRIVERS/RECEIVERS**, 9p. 1989, Acesso em 29/01/2013, Disponível em: <http://www.ti.com/lit/ds/symlink/max232.pdf>

ZANCO**,** Wagner da Silva**, Microcontroladores PIC16F628A/648A –** Uma Abordagem Prática e Objetiva, 1ª ed, São Paulo: Érica, 2005, 364p.

ZANCO**,** Wagner da Silva**, Microcontroladores PIC – Técnicas de Software e Hardware para Projetos de Circuitos Eletrônicos** – com base no PIC16F877A**,** 1ª ed. São Paulo: Érica, 2006, 390p.

VISCONTI**,** Antônio Carlos José Franceschini, **Microprocessadores 8080 e 8085, Hardware**, Volume 1, 2ª ed. São Paulo: Érica, 1982, 138p.

VISCONTI**,** Antônio Carlos José Franceschini, **Microprocessadores 8080 e 8085, Software**, Volume 2, 2ª ed. São Paulo: Érica, 1983, 202p.

ANEXOS

Um programa principal – computacional – controla a chamada de várias sub-rotinas, o que torna necessário identificar suas etapas e desenvolvê-las para suportarem o funcionamento do projeto como um todo. O programa principal, no momento oportuno, chama determinada sub-rotina, a qual resolverá parte do problema maior, e, ao seu término, retorna o controle ao programa principal.

ANEXO A – Softwares, Programa em *Assembly*

a) Programa Principal

A Fig. A1 mostra as atividades iniciais para implementação do programa principal.

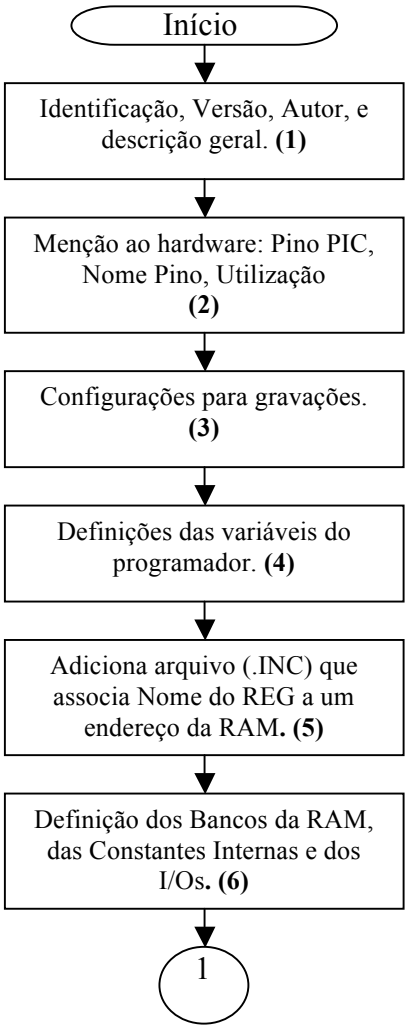


Figura A1 – Fluxograma do Programa Principal – Início.

Em qualquer programa de computador é comum iniciá-lo, além da sua autoria, por um pequeno descritivo que mencione a proposta básica do projeto. Deve-se lembrar que o MPLAB (Microchip), ao converter o programa fonte em executável, ignora tudo após ponto-e-vírgula (;).

```
;*****************************************************************
;*                  Programa Assembly para o PIC16F877A          (1)  *
;*                     Automação medidas eletrômetro                  *
;*                                                                    *
;*                        Autor: Abner P. Fonseca                     *
;*                                                                    *
;*****************************************************************
;*   VERSÃO : 1.0                                                     *
;*   DATA :     12/12/2012  às 12:12:12h                              *
;*****************************************************************

;*****************************************************************
;*                          DESCRIÇÃO GERAL                       *
;*****************************************************************
; Objetiva-se com este programa  em assembly, a automação das medidas das cargas elétricas detecta-  *
; das por uma câmara de ionização e medidas por um eletrômetro, em  cuja  interface  digital será lido  *
; este valor medido e armazenado em uma memória serial (24LC1025, de 128kBytes). Como unidade  *
; de saída de informação será usado um display de cristal líquido (LCD de  2 linhas  e  16 colunas),  e  *
; TX a um Computador Pessoal via RS232.                              *
;*****************************************************************
```

Para se ter uma idéia geral de como foi construído o hardware da placa principal, a que contém o microcontrolador PIC16F877A, é que, no próprio software *assembly*, foi eleborada uma lista que correlaciona o pino do PIC, o nome do pino e a utilização no circuito, conforme disposição infra mencionada:

```
;*****************************************************************
;*          DEFINIÇÃO DOS PINOS DO PIC16F877A NO HARDWARE UTILIZADO  (2) *
;*****************************************************************
; PINO PIC      Nome do Pino          Utilização no circuito
;   1           MCLR/Vpp              Reset da máquina
;
;   2           RA0/AN0               Driver MP (cabeça) Posicao_0
;   3           RA1/AN1               Driver MP (cabeça) Direção
;
;   4           RA2/AN2/Vref-         Chave Inic Grav (I/O)- acionada leva x e y a posicao_0
;   5           RA3/AN3/Vre+          Chave Fim Grav (I/O) - acionada TX dados CP via RS232.
;   6           RA4/T0CK1/ClOUT       Chave Exp.Futura(I/O)- acionada inicia automação
;
; OBS.: estas chaves, quando pressionadas, injetam GND (0)  se os estrapes do conector CN6 estiverem
;       ligados. Quando estas chaves não estiverem pressionadas injetarão nível lógico alto ( +5Vcc ).
;
;   7           RA5/AN4/ SS /C2OUT    Dado Serial, ligado ao pino 1 do CI1 - 74LS164
;
;   8           RE0/RD/AN5            Ligado ao Pino 6 conector CN3, Driver MP (0 a 90 cm) posição 0
;   9           RE1/WR/AN6            Ligado ao Pino 4 conector CN3, Driver MP (0 a 90 cm) CLOCK
```

```
;  10        RE2/CS/AN7          Ligado ao Pino 5 conector CN3, Driver MP (0 a 90 cm) Direção
;
;  11        Vdd                 Alimentação positiva do PIC16F877A ( +5Vcc ).
;  12        Vss                 Referência da alimentação do PIC16F877A ( GND ).
;  13        OSC1/CLKIN          Entrada de um dos terminais do cristal de 4MHz
;  14        OSC2/CLKOUT         Entrada do outro terminal do cristal de 4MHz
;
;  15        RC0/T1OS0/T1CK1     Ligado ao ENABLE do LCD
;  16        RC1/T1OS1/CCP2      Ligado ao RS do LCD
;
;  17        RC2/CCP1            Pino 1 do CN4 → Pino 1 DB25 - STROBE_9.
;
;  18        RC3/SCK/SCL         Ligado aos pinos 6 (SCL) do CIs3 a 6 (24LC1025)(output =0).
;
;  19        RD0/PSP0            Pino 3 do CN4 → Pino 2 DB25 - STROBE_1.
;  20        RD1/PSP1            Pino 5 do CN4 → Pino 3 DB25 - STROBE_2.
;  21        RD2/PSD2            Pino 7 do CN4 → Pino 4 DB25 - STROBE_3.
;  22        RD3/PSP3            Pino 9 do CN4 → Pino 5 DB25 - STROBE_4.
;
;  23        RC4/SDI/SDA         Ligado aos pinos 5 (SDA) do CIs3 a 10 (24LC1025)(Input =1).
;  24        RC5/SDO             Ligado ao pino 8 Que é o Clock do CI 74LS164.
;  25        RC6/TX/CK           Ligado ao pino 11 do CI11(MAX232) - Transmissão serial.
;  26        RC7/RX/DT           Ligado ao R14 (1k) que é ligado ao pino 12 do CI11 (RX).
;
;  27        RD4/PSP4            Pino 11 do CN4 → Pino 6 DB25 - STROBE_5.
;  28        RD5/PSP5            Pino 13 do CN4 → Pino 7 DB25 - STROBE_6.
;  29        RD6/PSP6            Pino 15 do CN4 → Pino 8 DB25 - STROBE_7.
;  30        RD7/PSP7            Pino 17 do CN4 → Pino 9 DB25 - STROBE_8.
;
;  31        Vss                 Referência da alimentação do PIC16F877A (GND ).
;  32        Vdd                 Alimentação positiva do PIC16F877A (+5Vcc ).
;
;  33        RB0/INT             Pino 19 do CN4 → Pino 10 DB25 - Bit menos sig.nibble medida.
;  34        RB1                 Pino 21 do CN4 → Pino 11 DB25 -              Nibble medida.
;  35        RB2                 Pino 23 do CN4 → Pino 12 DB25 -              Nibble medida.
;  36        RB3/PGM             Pino 25 do CN4 → Pino 13 DB25 - Bit mais sig. nibble medida.
;
;  37        RB4                 Driver MP (cabeça) Clock.
;
;  38        RB5                 Driver MP (0 a 41 cm) Posicao_0.
;  39        RB6/PGC             Driver MP (0 a 41 cm) clock.
;  40        RB7/PGD             Driver MP (0 a 42 cm) Direção.
;
; OBS.: Conector CN4 → Pino 10 = +5Vcc
;                    → Pino 12 = +12Vcc
;                    → Pinos 14, 16, 18, 20, 22, 24 e 26 ligados ao GND
;
; OBS.: Conector DB15 → Pinos de 18 até o 25 são ligados ao GND
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

As configurações utilizadas no momento da gravação do microcontrolador PIC16F877A (transferência do programa executável à memória *Flash* do PIC) foram as que se segue:

```
;*****************************************************************
;*                    CONFIGURAÇÕES PARA GRAVAÇÃO            (3)  *
;*****************************************************************
;
 __CONFIG _CP_OFF & _CPD_OFF & _DEBUG_OFF & _LVP_OFF & _WRT_OFF & _BODEN_OFF &
_PWRTE_ON & _WDT_OFF & _XT_OSC
;
;*****************************************************************
```

Qualquer programa de computador utiliza muitas variáveis, as quais, necessariamente, devem ser definidas ou declaradas. O fabricante do microcontrolador PIC16F877A (Microchip) designou uma faixa de endereçamento da memória RAM, interna ao PIC, para uso geral. Estes lugares físicos endereçáveis são locais onde o programador define as variáveis necessárias ao programa. Utilizando as diretrizes do MPLAB da Microchip CBLOCK / ENDC, e referindo-se ao endereço inicial 0x20, que é o primeiro lugar físico onde será ocupado pela variável ESPERA_CHAVES, conforme se pode perceber na sequência infra mencionada:

```
;*****************************************************************
;*                    DEFINIÇÃO DAS VARIÁVEIS                (4)  *
;*****************************************************************
;  ESTE BLOCO DE VARIÁVEIS ESTÁ LOCALIZADO LOGO NO INÍCIO DO BANCO 0

        CBLOCK      0X20            ; POSIÇÃO INICIAL DA RAM

                ESPERA_CHAVES       ; (0x20) Espera fim ruídos na chave quando apertada.
                CONTAR_ST           ; (0x21)

                TEMPO1              ; (0x22)
                TEMPO0              ; (0x23) CONTADORES P/ DELAY

                FLAG                ; (0x24) FLAG [PARA USO GERAL]

                AUX                 ; (0x25) REGISTRADOR AUXILIAR DE USO GERAL

                ENDERECO_HIGH       ; (0x26) REGISTRADORES DE ENDEREÇO PARA
                ENDERECO_LOW        ; (0x27) ACESSO À MEMÓRIA EEPROM SERIAL EXTERNA
                                    ;        MAPEADOS NO BANCO 0 DA RAM
                CONTROLE_I2C        ; (0X28) REGISTRADOR COM O ATUAL CONTROLE I2C
                A1A0_I2C            ; (0X29) POSICAO FÍSICA DA EEPROM SERIAL A1A0.
                DADO_I2C            ; (0x2A) REG GUARDA TEMPORARIAMENTE O DADO DA
                                    ;   LEITURA/ESCRITA DA/NA EEPROM SERIAL 24LC1025.
                VALOR               ; (0x2B) PARA SER TX SERIE-PARALELO
                Desloca             ; (0x2C) Registra quantos bits foram deslocados.
                WTEMP               ; (0x2D) GUARDAR W TEMPORARIAMENTE
                WTEMP1              ; (0x2E) Guarda temporariamente W

                                    ; (0x2F)
                L1_C0               ;F Byte ASCII a ser plotado na Linha1_Coluna0 do LCD
                L1_C1               ;0 Byte ASCII a ser plotado na Linha1_Coluna1 do LCD
                L1_C2               ;1 Byte ASCII a ser plotado na Linha1_Coluna2 do LCD
```

```
L1_C3               ;2 Byte ASCII a ser plotado na Linha1_Coluna3 do LCD
L1_C4               ;3 Byte ASCII a ser plotado na Linha1_Coluna4 do LCD
L1_C5               ;4 Byte ASCII a ser plotado na Linha1_Coluna5 do LCD
L1_C6               ;5 Byte ASCII a ser plotado na Linha1_Coluna6 do LCD
L1_C7               ;6 Byte ASCII a ser plotado na Linha1_Coluna7 do LCD
L1_C8               ;7 Byte ASCII a ser plotado na Linha1_Coluna8 do LCD
L1_C9               ;8 Byte ASCII a ser plotado na Linha1_Coluna9 do LCD
L1_C10              ;9 Byte ASCII a ser plotado na Linha1_Coluna10 do LCD
L1_C11              ;A Byte ASCII a ser plotado na Linha1_Coluna11 do LCD
L1_C12              ;B Byte ASCII a ser plotado na Linha1_Coluna12 do LCD
L1_C13              ;C Byte ASCII a ser plotado na Linha1_Coluna13 do LCD
L1_C14              ;D Byte ASCII a ser plotado na Linha1_Coluna14 do LCD
L1_C15              ;E Byte ASCII a ser plotado na Linha1_Coluna15 do LCD
                    ; (0x3E)
                    ; (0x3F)
L2_C0               ;F Byte ASCII a ser plotado na Linha2_Coluna0 do LCD
L2_C1               ;0 Byte ASCII a ser plotado na Linha2_Coluna1 do LCD
L2_C2               ;1 Byte ASCII a ser plotado na Linha2_Coluna2 do LCD
L2_C3               ;2 Byte ASCII a ser plotado na Linha2_Coluna3 do LCD
L2_C4               ;3 Byte ASCII a ser plotado na Linha2_Coluna4 do LCD
L2_C5               ;4 Byte ASCII a ser plotado na Linha2_Coluna5 do LCD
L2_C6               ;5 Byte ASCII a ser plotado na Linha2_Coluna6 do LCD
L2_C7               ;6 Byte ASCII a ser plotado na Linha2_Coluna7 do LCD
L2_C8               ;7 Byte ASCII a ser plotado na Linha2_Coluna8 do LCD
L2_C9               ;8 Byte ASCII a ser plotado na Linha2_Coluna9 do LCD
L2_C10              ;9 Byte ASCII a ser plotado na Linha2_Coluna10 do LCD
L2_C11              ;A Byte ASCII a ser plotado na Linha2_Coluna11 do LCD
L2_C12              ;B Byte ASCII a ser plotado na Linha2_Coluna12 do LCD
L2_C13              ;C Byte ASCII a ser plotado na Linha2_Coluna13 do LCD
L2_C14              ;D Byte ASCII a ser plotado na Linha2_Coluna14 do LCD
L2_C15              ;E Byte ASCII a ser plotado na Linha2_Coluna15 do LCD
                    ; (0x4E)

X                   ; (0x4F) deslocamento mesa X [0 a 41 cm, passos de 1 cm].
Y                   ; (0x50) deslocamento mesa Y [2 a 92 cm, passos de 1 cm].
BCD_UNIDADE         ; (0x51) Unidade do DADO em BCD
BCD_DEZENA          ; (0x52) Dezena do DADO em BCD
X_UNIDADE           ; (0x53) Unidade de X - BCD
X_DEZENA            ; (0x54) Dezena de X - BCD
SINAL_DE_X          ; (0x55) (-) ASCII 0x2D e (+) ASCII 0x2B
Y_UNIDADE           ; (0x56) Unidade de Y - BCD
Y_DEZENA            ; (0x57) Dezena de Y - BCD
N_PASSOS            ; (0x58) Número de Passos adquiridos das Tabelas
N_PASSOS_UNIDADE    ; (0x59) Unidade - Número de passos em BCD
N_PASSOS_DEZENA     ; (0x5A) Dezena - Número de passos em BCD
COMPLETA_X          ; (0x5B) após posicáo_0_0a41 dá mais passos até zero real
VOLTA               ; (0x5C) pulsos necessários para avancar a mesa.

;-------------------------------------------------------------------------------------------------------------
;    Registradores utilizados no processo de leitura do Eletrômetro.
;-------------------------------------------------------------------------------------------------------------
ST_1_2              ; (0x5D) armazena dados liberados pelo eletrômetro qdo
                    ;           Strobe 1 e 2 ativos.
ST_3_4              ; (0x5E) Idem.Para os Strobes 3 e 4 ativos.
ST_5_6              ; (0x5F) Idem.Para os Strobes 5 e 6 ativos.
ST_7_8              ; (0x60) Idem.Para os Strobes 7 e 8 ativos.
```

```
                ST_9                    ; (0x61) Idem.Para o Strobe 9
                ;--------------------------------------------------------------------------

                ENDERECO_I2C_1  ; (0x62) Endereco menos sig. (o mais a direita, em ASCII)
                ENDERECO_I2C_2  ; (0x63)
                ENDERECO_I2C_3  ; (0x64)
                ENDERECO_I2C_4  ; (0x65)
                ENDERECO_I2C_5  ; (0x66) Endereco mais sig. (o mais a esquerda, em ASCII)

                ENDERECO_I2C_11 ; (0x67) Endereco menos sig. (o mais a direita, em ASCII)
                ENDERECO_I2C_21 ; (0x68)
                ENDERECO_I2C_31 ; (0x69)
                ENDERECO_I2C_41 ; (0x6A)
                ENDERECO_I2C_51 ; (0x6B) Endereco mais sig. (o mais a esquerda, em ASCII)
        ENDC
;*****************************************************************
```

Além das disposições mencionadas, antes de iniciar o programa, é necessário definir diversas situações que facilitam a programação, bem como outras necessidades; conforme listado abaixo.

```
;*****************************************************************
; * O ARQUIVO .INC [ABAIXO] RESGATA OS REGISTRADORES [NOMES E ENDERECOS]      *
; * DADOS PELA MICROCHIP - EVITA-SE TER QUE DIGITÁ-LOS.                (5)   *
;*****************************************************************

        #INCLUDE <P16F877A.INC>         ; MICROCONTROLADOR utilizado.

;*****************************************************************
; *   DEFINIÇÃO DA CHAMADA DE UM DOS BANCOS DA RAM                    (6)   *
;*****************************************************************

#DEFINE         BANK1 BSF       STATUS,RP0      ; Seleciona o BANK1 da memória RAM.
#DEFINE         BANK0 BCF       STATUS,RP0      ; Seleciona o BANK0 da memória RAM.

;*****************************************************************
; *  DEFINICAO DAS CONSTANTES UTILIZADAS NO PROGRAMA                       *
;*****************************************************************

ESPERA_CHAVE        EQU     .200            ; Tempo espera fim ruídos nas chaves.

;*****************************************************************
; *                DEFINICAO DAS ENTRADAS                             *
;*****************************************************************
;  Associar nomes as entradas facilita a programação e alterações futuras.
;
;Definição das chaves [nome chave, porta e bit]
#DEFINE         INICIA_XY_0             PORTA,2         ; ESTADO das chaves (PUSH Botton)
#DEFINE         TXCP_RS232              PORTA,3                 ; 1 → LIBERADO
#DEFINE         AUTOMACAO               PORTA,4                 ; 0 → PRESSIONADO

;Definição dos strobes vindos do eletrômetro, [quando em zero está ativo].

#DEFINE         STROBE_1                PORTD,0         ; Emitir sinal strobe_1 pelo RD0
```

```
#DEFINE     STROBE_2          PORTD,1     ; Emitir sinal o strobe_2 pelo RD1
#DEFINE     STROBE_3          PORTD,2     ; Emitir sinal o strobe_3 pelo RD2
#DEFINE     STROBE_4          PORTD,3     ; Emitir sinal o strobe_4 pelo RD3
#DEFINE     STROBE_5          PORTD,4     ; Emitir sinal o strobe_5 pelo RD4
#DEFINE     STROBE_6          PORTD,5     ; Emitir sinal o strobe_6 pelo RD5
#DEFINE     STROBE_7          PORTD,6     ; Emitir sinal o strobe_7 pelo RD6
#DEFINE     STROBE_8          PORTD,7     ; Emitir sinal o strobe_8 pelo RD7
#DEFINE     STROBE_9          PORTC,2     ; Emitir sinal o strobe_9 pelo RC2

;--------------------Definição para entrada do dado lido(vindo) do eletrômetro-------------------------------------

#DEFINE     Nibble_Medida     PORTB       ; Entrada dado lido Eletrômetro em RB3, a RB0

#DEFINE     Posicao_0_0a90    PORTE,0     ; Quando em nível alto (1) está na posição 0.
#DEFINE     Posicao_0_0a41    PORTB,5
#DEFINE     Pos_0_cabeca      PORTA,0

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; *                  DEFINICAO DAS SAÍDAS                                                                              *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;  Associar nomes as saídas facilita a programação e alterações futuras.

;-------------------------------------------------Referente ao LCD-----------------------------------------------------
#DEFINE     ENABLE            PORTC,0     ; Sinal Enable p/ o DISPLAY
                                          ; Ativo na borda de descida.
#DEFINE     RS                PORTC,1     ; Indica p/ o DISPLAY um dado ou comando.
                                                ; 1 → Dado
                                                ; 0 → Comando

#DEFINE     Ck                PORTC,5     ; Clock do CI 74LS164, transfere na subida.
#DEFINE     Dserie            PORTA,5     ; Dados para ser enviado ao display (74LS164).
;-------------------------------------------------------------------------------------------------------------------------
;referente a memória 24LC1025
#DEFINE     SCL               PORTC,3     ; Sinal de clock da EEPROM (como saída no PIC)

;-----------------------------------------Referente aos motores de passo-------------------------------------------
#DEFINE     Clock_0a90        PORTE,1     ; A cada variação 0→1→0 dá um passo
#DEFINE     Direcao_0a90      PORTE,2     ; Dependendo do nível lógico o motor de passo
                                          ; girará no sentido horário ou anti-horário.
                                                ; 1 → Horário
                                                ; 0 → Anti-horário.

#DEFINE     Clock_0a41        PORTB,6     ; A cada variação 0→1→0 dá um passo
#DEFINE     Direcao_0a41      PORTB,7     ; Dependendo do nível lógico o motor de passo
                                          ; girará no sentido horário ou anti-horário.
                                                ; 1 → Horário
                                                ; 0 → Anti-horário.

#DEFINE     Clock_cabeca      PORTA,1     ; A cada variação 0→1→0 dá um passo
#DEFINE     Direcao_cabeca    PORTB,4     ; Dependendo do nível lógico o motor de passo
                                          ; girará no sentido horário ou anti-horário.
                                                ; 1 → Horário
                                                ; 0 → Anti-horário.
```

```
;*********************************************************************
;*      Definição das vias que ora são ENTRADA, ora são SAÍDA        *
;*********************************************************************
;referente a memória 24LC1025
#DEFINE         SDA             PORTC,4         ; Linha de dados da EEPROM é bidirecional, deve-se
                                                ; iniciá-la como entrada.
;*********************************************************************
```

O processamento é então iniciado com a execução da instrução GOTO INICIO, salto incondicional ao *label* INICIO. Todas as configurações necessárias ao funcionamento do projeto são realizadas neste momento, conforme mostra a Fig. A2.

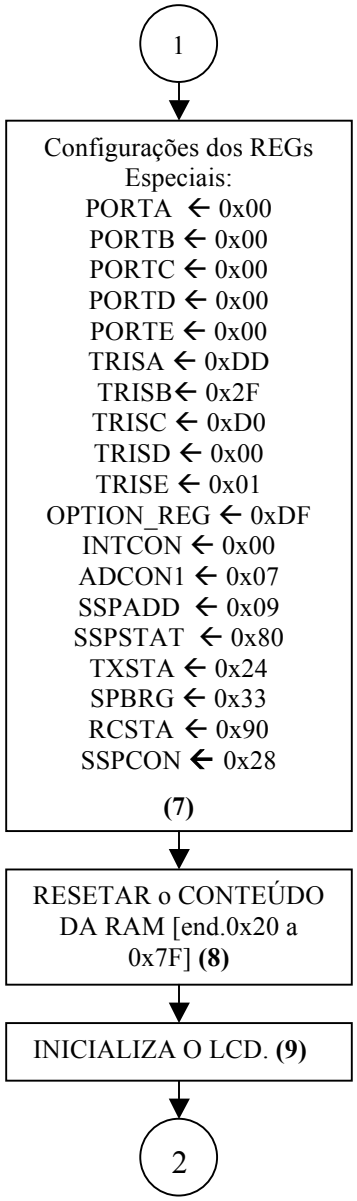


Figura A2 – Fluxograma do Programa Principal – Configurações REGs.

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; *               CONFIGURAÇÕES INICIAIS DO HARDWARE E DO SOFTWARE               (7)  *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;  NESTA ROTINA SÃO INICIALIZADAS AS PORTAS DE I/O DO MICROCONTROLADOR E AS
;  CONFIGURAÇÕES DOS REGISTRADORES ESPECIAIS (SFR)..

INICIAR
        CLRF        PORTA           ; Faz as saídas assumir nível lógico inicial baixo (0).
        CLRF        PORTB
        CLRF        PORTC
        CLRF        PORTD
        CLRF        PORTE

        BANK1                       ; Seleciona Banco 1 da RAM.

        MOVLW       B'11011101'     ; Confirmação da PORTA (I/O)
        MOVWF       TRISA           ; Bits → nd nd RA5 RA4 RA3 RA2 RA1 RA0
                                    ; bits 7 e 6 → irrelevantes
                                    ; RA5=0=Dserie → Output, dado serial para o 74LS164 do LCD
                                    ; RA4=1= chave AUTOMACAO, inicia processo de automação
                                    ; RA3=1= chave TXCP_RS232, acionada tx dados ao CP.
                                    ; RA2=1= chave INICIA_XY_0, faz mesa X e Y posição zero.
                                    ; RA1=0= controla a DIRECAO_CABECA.
                                    ; RA0=1= em nível alto indica que a cabeça esta na posicao_0.

        MOVLW       B'00101111'     ; Configuração da PORTB (I/O) -
        MOVWF       TRISB           ; Bits → RB7 RB6 RB5 RB4 RB3 RB2 RB1 RB0
                                    ; Todos os bits configurados para entrada de dados
                                    ; Nome associado à PORTB → Nibble_Medida
                                    ; serão utilizados os bits menos significativos de RB,
                                    ; ou sejam: RB3, RB2, RB1 e RB0, onde:
                                    ; RB0=1 → na leitura eletrômetro é o bit menos sig (A)
                                    ; RB1=1 → na leitura eletrômetro é o bit (B)
                                    ; RB2=1 → na leitura eletrômetro é o bit (C)
                                    ; RB3=1 → na leitura eletrômetro é o bit mais sig  (D)
                                    ; RB4=0 → é o clock do MP da cabeça.
                                    ;RB5=1 → em nível alto indica que a mesa X esta na posicao_0.
                                    ; RB6=0 → é o clock do MP da mesa X.
                                    ; RB7=0 → é o sinal DIRECAO do MP da mesa X

        MOVLW       B'11010000'     ; Configuração da PORTC (I/O)
        MOVWF       TRISC           ; Bits → RC7 RC6 RC5 RC4 RC3 RC2 RC1 RC0
                                    ; RC7=1=RX, Input, recepção dados do Max232
                                    ; RC6=0=TX, Output, transmissão dados p/ Max232
                                    ; RC5=0=Ck, Output do clock para o CI 74LS164.
                                    ; RC4=1=SDA, Input, mas é ent/saída dados p/ memo 24LC1025
                                    ; RC3=0=SCL, Output do clock para as memórias 24LC1025.
                                    ; RC2=0=STROBE_9, Output do strobe-9 para o eletrômetro.
                                    ; RC1=0=RS, Output para dados ou comando para o LCD
                                    ; RC0=0=ENABLE, Output para habilitar o LCD

        MOVLW       B'00000000'     ; Configuração da PORTD (I/O)
        MOVWF       TRISD           ; Bits → RD7 RD6 RD5 RD4 RD3 RD2 RD1 RD0
                                    ; RD7=0 PORTB,W
                                    ; W ← PORTB=STROBE_8, Saída strobe_8 para o eletrômetro.
```

```
                                ; RD6=0=STROBE_7, Output do strobe_7 para o eletrômetro.
                                ; RD5=0=STROBE_6, Output do strobe_6 para o eletrômetro.
                                ; RD4=0=STROBE_5, Output do strobe_5 para o eletrômetro.
                                ; RD3=0=STROBE_4, Output do strobe_4 para o eletrômetro.
                                ; RD2=0=STROBE_3, Output do strobe_3 para o eletrômetro.
                                ; RD1=0=STROBE_2, Output do strobe_2 para o eletrômetro.
                                ; RD0=0=STROBE_1, Output do strobe_1 para o eletrômetro.

MOVLW       B'00000001'         ; Configuracão da PORTE (I/O)
MOVWF       TRISE               ; Bits → nd nd nd nd nd RE2 RE1 RE0
                                ; bits de 7 a 5 para PSP, controle de bits.
                                ; PSPMODE →em zero: PORTD configurada para modo I/O
                                ; U-0 → lido como zero
                                ; RE2=0=Direcao_0a90, Out, sentido rotacão motor passo 0_90.
                                ; RE1=0=Clock_0a90, Saída, cadência para cada um dos passos
                                ; RE0=1=Posicao_0_0a90, Input,  nível alto encontra a posicão_0

MOVLW       B'11011111'
MOVWF       OPTION_REG          ; Definindo o perfil de Trabalho no OPTION_REG
                                ; /RBPU =1→ Pull ups internos da PORTB desabilitados p/saídas
                                ; INTEDG=1→ Interrupcão externa RB0 ocorrerá borda subida
                                ;T0SE  =0 → incrementa TMR0 pelo clock interno máquina
                                ; PSA   =1 → o prescaler será aplicado ao WDT
                                ; PS2 PS1 PS0   Qdo TMR0(PSA=0)   Qdo WDT (PSA=1)
                                ;  0   0   0        1:2              1:1
                                ;  0   0   1        1:4              1:2
                                ;  0   1   0        1:8              1:4
                                ;  0   1   1        1:16             1:8
                                ;  1   0   0        1:32             1:16
                                ;  1   0   1        1:64             1:32
                                ;  1   1   0        1:128            1:64
                                ;  1   1   1        1:256            1:128
                                ;
                                ; quando utilizado, WDT - 1:128, mas desativou-se esta função

MOVLW       B'00000000'         ; INTCON = Configuracão das Interrupcões
MOVWF       INTCON              ; Bits → GIE PEIE T0IE INTE RBIE T0IF INTF RBIF
                                ;              _/ __/_              ----------
                                ;                 |_____/ _          Identificam int
                                ;                 |_____/ _      TMR0  RB0  Mudança
                                ;                 |_____/ _           RB4 a RB7
                                ;
                                ;INTCON = 00h → Desabilitada todas as interrupções.

MOVLW       B'00000111'         ; ADCOM1 = Config. pinos em analógicos ou digitais
MOVWF       ADCON1              ; Bits → ADFM  nd  nd  nd  PCFG3  PCFG2  PCFG1  PCFG0
                                ; ADFH=0 → o resultado da conversão A/D de 10 bits será
                                ;          justificado à esquerda. Os 6 bits menoS sig
                                ;          de ADRESL são lidos como nível lógico baixo(0).
                                ;          os três próximos zeros (000) não têm significado.
                                ;
                        ; PCFG3: AN7 AN6 AN5 AN4 AN3 AN2 AN1 AN0 V    V    Canais  Ref.
                        ; PCFG0  RE2 RE1 RE0 RA5 RA3 RA2 RA1 RA0 ref+ ref- Analog. Ext.
                        ; ------------------------------------------------------------------------
                        ;  0111    D    D    D    D    D    D    D    D  Vdd  Vss    0      0
                        ; ------------------------------------------------------------------------
```

```
                        ; onde: A= Analógico e D=Digital  do respectivo pino do PIC.

MOVLW   B'00001001'     ; SSPADD = Endereco da unidade I2C (parte alta e baixa)
MOVWF   SSPADD          ; Velocidade: 100KHz @ 4MHz

MOVLW   B'10000000'     ; SSPSTAT =
MOVWF   SSPSTAT         ; bits → SMP CKE D/A P S R/W UA
                        ; SMP → para I2C ou SPI. P/I2C = 1 → desabilita o controle
                        ;           de sinal para alta velocidade (100kHz).
                        ; CKE → paraI2C ou SPI. P/I2C = 0 → Entradas conforme
                        ;                                   especificacões i2C.
                        ; D/A = 0 → Último byte recebido/transmitido foi um endereco.
                        ; P =Indicacão de Stop=0 → não ocorreu uma condicão de STOP.
                        ; S =Indicacão de Start=0 → não ocorreu a condicão de START.
                        ; R/W=Indicacão de Leitura/Escrita (I2C)=se Master =
                        ;         Nenhuma transmissão em andamento.
                        ; UA=Indicacão de atualizacão de endereco (I2C 10 bits)=0 →
                        ;       Não precisa atualizar o endereco.
MOVLW   B'00100100'     ; W ← 0x24
MOVWF   TXSTA           ; Configuração da USART, TX habilitado no modo assíncrono,
                        ; 8 bits de dados, e High speed baud rate.
MOVLW   .51             ; .51 Para velocidade TX de 4800bps .
MOVWF   SPBRG           ; Ajuste do Baud Rate para 4800 bps.

BANK0                   ; Seleciona o Bank0 da memória RAM.

MOVLW   B'10010000'     ; W ← 0x90
MOVWF   RCSTA           ; Configuração da USART, RX habilitado no modo assíncrono,
                        ; Recepção Contínua de 8 bits de dados, e desabilitado address
                        ; detect.

MOVLW   B'00101000'     ; SSPCON =
MOVWF   SSPCON          ; WCOL SSPOV SSPEN CKP SSPM3 SSPM2 SSPM1 SSPM0
                        ; WCOL=0 → Colisão na escrita do registrador SSPBUF
                        ;                 para MASTER -> quando 0(zero) = sem colisão.
                        ; SSPOV=0 → Erro de Overflow na recepcão, zero = sem erro.
                        ; SSPEN=1 → Habilitacao da porta serial SSP, para I2C,
                        ;  SSP habilitada. Pinos SCL e SDA controlados internamente.
                        ; CKP=0 → Controle de clock, sendo zero e I2C Master →
                        ;             sem uso.
                        ; SSPM3: → Selecão do modo de trabalho da porta SSP.
                        ; SSPM0
                        ;   0000  → SPI Master. Clock=Fosc/4.
                        ;   0001  → SPI Master. Clock=Fosc/16.
                        ;   0010  → SPI Master. Clock=Fosc/64.
                        ;   0011  → SPI Master. Clock=TMR2/2.
                        ;   0100  → SPI Slave. Pino /SS habilitado.
                        ;  0101  → SPI Slave. Pino /SS desabilitado (ou como I/O)
                        ;   0110  → I2C Slave. Enderecamento de 7 bits.
                        ;   0111  → I2C Slave. Enderecamento de 10 bits,
                        ; ---------------------------------------------------------------
                        ;   1000  → I2C Master. Clock = Fosc / (4x(SSPADD+1)).
                        ; ---------------------------------------------------------------
                        ; Outros → reservados
                        ; Habilita  I2C - MASTER MODE
                        ; Configura pinos como da I2C
```

```
        BSF             SCL             ; Inicia SCL em nível lógico ALTO.

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; *       RESETANDO o CONTEÚDO DE TODA A RAM [end. 0x20 a 0x7F]                                   (8)  *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

        MOVLW   0x20
        MOVWF   FSR             ; O registrador FSR deve conter o endereco no qual será gravado
                                ; o valor que estiver presente no registrador INDF, então, objeti-
                                ; vamente: incia-se  o  REG  FSR  com  20h, o qual é o primeiro
                                ; endereco da RAM onde definiu-se, neste programa,  para  ser o
                                ; o registrador de nome: ESPERA_CHAVES, sendo 7Fh o último
                                ; endereco destinado a implementação de variáveis do usuário.
ZERAR_RAM
        CLRF    INDF            ; zera o conteúdo de INDF, ou: INDF ← 00h,  que  é o conteúdo
                                ; (dado) que é carregado no registrador endereçado (apontando)
                                ; residente, neste momento, no registrador FSR, iniciado com o
                                ; valor 20h.
        INCF    FSR,F           ; Incrementa o apontador FSR para o próximo endereço (registra-
                                ; dor), ou FSR ←FSR+1, i.e.a 1ª vez ficará: FSR←20h+01h= 21h
        MOVF    FSR,W           ; W ← FSR
        XORLW   0X80            ; W ← W XOR 80h na 1a vez fica: W = 21h = 0010 0001
                                ;                                80h = 1000 0000
                                ;                          W ← XOR = 1010 0001
                                ; XOR afeta o flag Z que neste caso não resultou em zero, ou z=0
                                ; Qdo ao último endereco (7Fh) for somado 1 ficará: 80h, então:
                                ;                          W = 80h = 1000 0000
                                ;                           80h = 1000 0000
                                ;                          W ←XOR = 0000 0000   e Z = 1.
        BTFSS   STATUS,Z        ; Testa o bit Z do STATUS e salta a próxima instrucão caso o bit
                                ; Z seja 1 (ou o resultado da operacão anterior tenha dado zero.
                                ; Caso contrário, executa a próxima instrucão, ou seja, execute a
                                ; instrucão GOTO ZERAR_RAM, ou feche o loop ou reinicie o
                                ; processo de clear os registradores da RAM. Já limpou todos os
                                ; enderecos da RAM?
        GOTO    ZERAR_RAM       ; NÃO - retorne ao início para zerar a próxima posicão da RAM
                                ; SIM - dê sequencia aa execucão do programa.

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; *     A SEGUIR  INICIALIZA O DISPLAY DE CRISTAL LÍQUIDO (LCD)                                 (9) *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; INICIALIZACAO DO LCD PARA COMUNICAÇÃO EM 8 BITS, USANDO LCD DE 2 LINHAS E
; 16 COLUNAS, SEM CURSOR E DESLOCAMENTO DO CURSOR PARA DIREITA.

        MOVLW   0x08            ; W ← 0x08
        MOVWF   Desloca         ; quanto se deslocará à direita o byte a ser enviado ao LCD.

INICIAR_DISP

        BCF     RS              ; Seleciona o DISPLAY p/ comandos.
        MOVLW   .33
        CALL    DELAY_MS        ; DELAY de 33ms
        MOVLW   0x38            ; Escreve comando  0x38 para inicialização → Function Set.
        CALL    ESCREVE
        MOVLW   0x0C            ; Escreve comando  0x0C para inicialização →
```

```
        CALL      ESCREVE    ; → Display on/off control
        MOVLW     0x1        ; Escreve comando para Limpar todo o DISPLAY.
        CALL      ESCREVE
        MOVLW     .2
        CALL      DELAY_MS   ; DELAY DE 2ms
        MOVLW     0x6        ; Escreve comando para incremento,
        CALL      ESCREVE    ; com shift display desligado.
        BSF       RS         ; Seleciona o DISPLAY p/ dados
                             ; fim da inicialização
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

Neste momento o operador da máquina terá que tomar uma das três decisões possíveis: (a) pressionando a tecla <1> o operador garantirá que as mesas se encontram em suas posições zero, ou seja, é controlado o movimento da mesa Y e depois da mesa X até suas posições iniciais, Y em zero e X em (-20,0). Este movimento é realizado numa maior velocidade de forma a necessitar de um menor tempo de espera para a realização desta atividade. (b) Uma vez que o eletrômetro esteja ligado à placa principal, ao detector – câmara de ionização, e que a fonte radioativa esteja posicionada de forma a atingir a janela do detector (ou seu volume sensível) a cada posição relativa Y e X, então, deve-se pressionar a tecla <2> que iniciará o processo de automação. Essa rotina controla todas as 3.731 medidas, armazenando-as na memória estática 24LC1025. (c) Uma vez que todas as medidas tenham sido realizadas é preciso transmiti-las a um computador pessoal, através da interface RS-232 e cabo apropriado, de forma a tornar fácil o manuseio destes dados em aplicativos como, por exemplo, a planilha eletrônica da Microsoft conhecida por Excel.

A Fig. A3 sintetiza o procedimento apresentado acima, ou seja, o direcionamento do processamento ao pressionar uma das teclas <1>, <2> ou <3>.

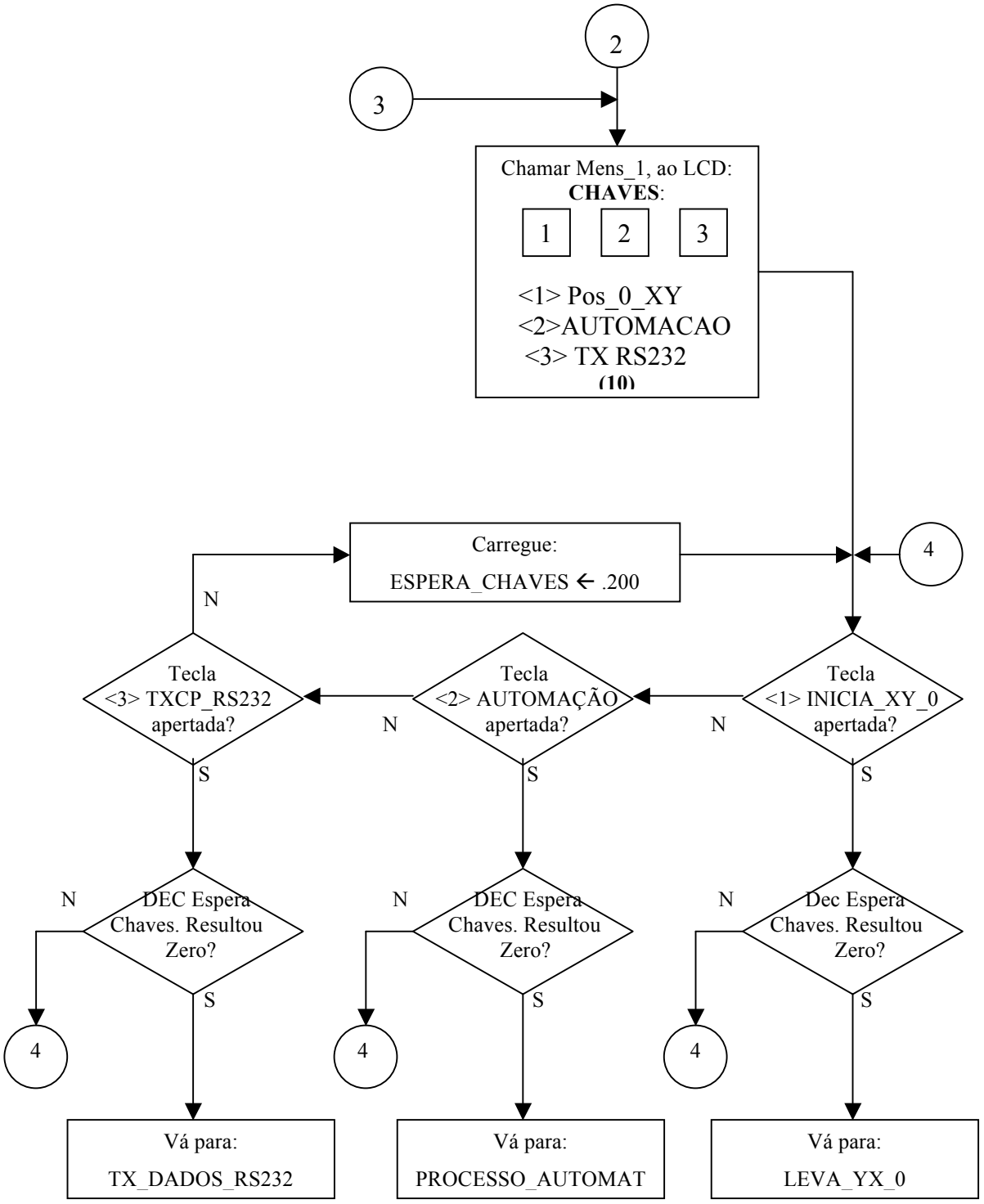


Figura A3– Fluxograma do Programa Principal – Decisões para processamento.

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; *                          Apresenta Mens_1 no LCD                          (10)   *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

DENOVO
        PAGESEL     Mens_1          ; deixar, pois qualquer salto para este ponto PCLATHé ajustado
        CALL        Mens_1          ; Escreve no LCD:
                                    ; <1>Pos_0_XY  <2>
                                    ; AUTOMACAO  <3>TX RS232

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; *                            Varredura das chaves                                  *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Verificação de chaves apertadas com devido encaminhamento para execução da função desejada.

VERIFIQUE

    ;*********************** Verificar se alguma chave está apertada ************************

VARRE_CHAVES
        BTFSS       INICIA_XY_0         ; A chave está pressionada?
        GOTO        ESPERA_CHAVE_1            ;     SIM (nível 0) - executa GOTO
                                              ;     NÃO (nível 1) - não executa GOTO

        BTFSS       AUTOMACAO           ; A chave está pressionada?
        GOTO        ESPERA _CHAVE_2           ;     SIM (nível 0) - executa GOTO
                                              ;     NÃO (nível 1) - não executa GOTO

        BTFSS       TXCP_RS232          ; A chave está pressionada?
        GOTO        ESPERA _CHAVE_3           ;     SIM (nível 0) - executa GOTO
                                              ;     NÃO (nível 1) - não executa GOTO

    ;----------------------------Carrega valor para DEBOUNCE (instabilidade)--------------------------------------

        MOVLW       ESPERA_CHAVE        ; Carrega o valor 200 em W
        MOVWF       ESPERA_CHAVES       ; ESPERA_CHAVES ← 200 [evitar ruído na chave]

        GOTO        VERIFIQUE           ; VOLTA PARA VARRER TECLADO

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; *                            Tratamento das chaves                                 *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

; ******************************** Tratamento da chave_1 *******************************

ESPERA _CHAVE_1                         ; Apertou a chave INCIA_XY_0
                                        ; Controla o DESLOCAMENTO DAS MESAS x E y
                                        ; ATÉ POSICAO_0.
        DECFSZ      ESPERA_CHAVES,F     ; Fim do ruído da chave?
        GOTO        VERIFIQUE           ; NÃO - VOLTA P/ VERIFIQUE [chave ainda apertada]
                                        ; SIM - BOTÃO PRESSIONADO
        GOTO        LEVA_YX_0
```

```
; ********************************* Tratamento da chave_2 *********************************

ESPERA _CHAVE_2                         ; Apertou a chave AUTOMACAO
                                        ; Controla todo processo de posicionamento
                                        ; relativo entre fonte e detector, promove
                                        ; a leitura do eletrômetro e o armazenamento
                                        ; do dado lido na devida posição da memória 24LC1025.
        DECFSZ      ESPERA_CHAVES,F     ; Fim do ruído da chave?
        GOTO        VERIFIQUE           ; NÃO - VOLTA P/ VARRE
                                        ; SIM - BOTÃO PRESSIONADO
        GOTO        PROCESSO_AUTOMAT

; ********************************* Tratamento da chave_3 *********************************

ESPERA _CHAVE_3                         ; Apertou a chave TXCP_RS232
                                        ; acerta os dados gravados na 24LC1025 e
                                        ; transmite-os, via RS232, ao CP.
        DECFSZ      ESPERA_CHAVES,F     ; Fim do ruído da chave?; GOTO   VERIFIQUE
        GOTO        VERIFIQUE           ; NÃO - VOLTA P/ VARRE
                                        ; SIM - BOTÃO PRESSIONADO

        PAGESEL     TX_DADOS_RS232      ; Tx dados lidos da 24LC1025 para o PC
        GOTO        TX_DADOS_RS232
;*********************************************************************************************
```

No início do processamento é apresentado o *menu* principal no *display* LCD (Mens_1, Fig. A50) e caso o operador pressionar a tecla <1> direcionará o processamento para a rotina de controle das mesas Y e X até suas posições iniciais (posição zero), condizente com a Fig. A4.

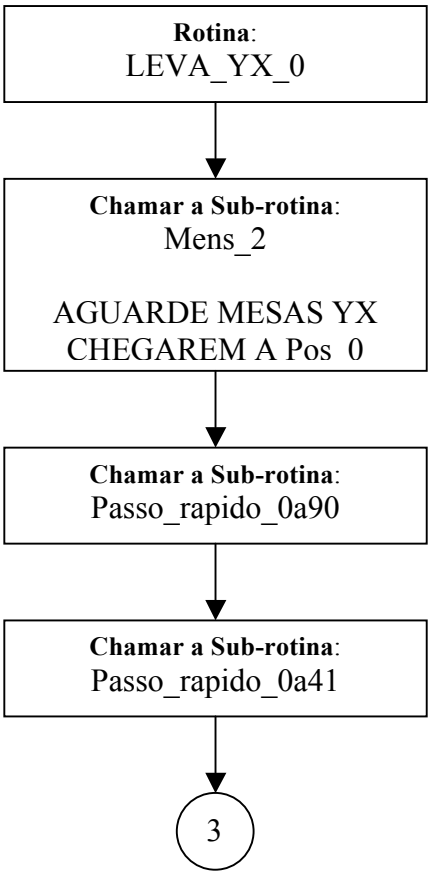


Figura A4– Fluxograma do Programa Principal.
Leva mesas Y e X às posições iniciais.

```
;*******************************************************************************
; Esta rotina controla o deslocamento rápido da mesa Y e da mesa X até a posição 0                *
;*******************************************************************************

LEVA_YX_0                               ; Leva mesas Y e X rapidamente à posição zero (inicio)
        CALL            Mens_2          ; AGUARDE MESAS YX
                                        ; CHEGAREM A Pos_0
        CALL            Passo_rapido_0a90
        CALL            Passo_rapido_0a41

        GOTO            DENOVO
;*******************************************************************************
```

Terminada a execução da Fig. A4, o processamento volta ao *menu* principal (Fig. A3) e se o operador pressionar a tecla <2> o direcionará à rotina de automação, ou de controle geral, ou seja, a que posiciona as mesas; direciona do fluxo radioativo ao volume sensível do detector; lê o eletrômetro e armazena suas medidas lidas na memória 24LC1025. Isto será executado por 3.731 vezes. Este procedimento é mostrado nas Fig. A5 a Fig. A7.

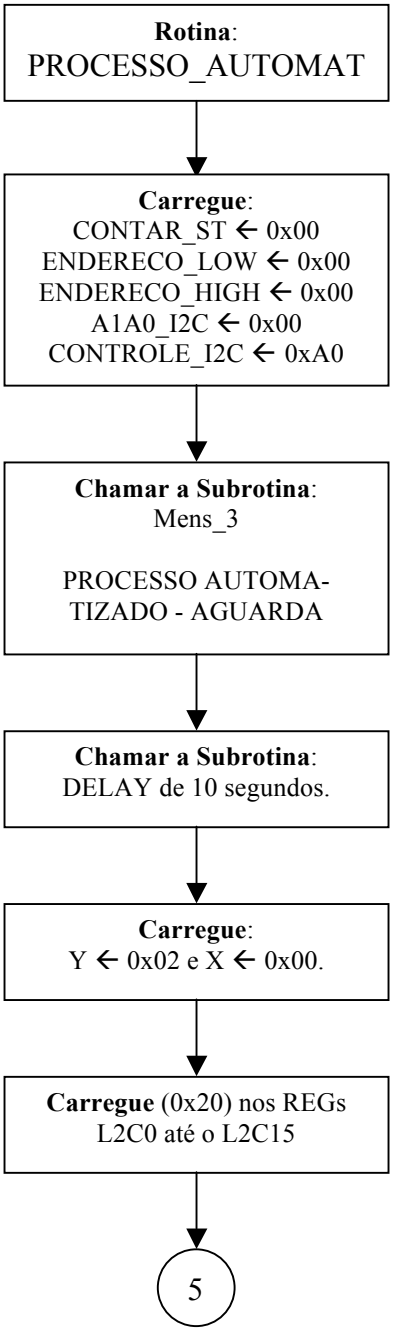


Figura A5– Fluxograma do Programa Principal – Processo Automático – início.

```
;*******************************************************************************
;*Esta rotina controla todo o processo de posicionamento,leitura eletrômetro e gravação do dado lido em  *
;*posições adjacentes da memória 24LC1025.                                      *
;*******************************************************************************

PROCESSO_AUTOMAT
        CLRF        CONTAR_ST           ; Inicia CONTAR_ST com 0x00

    ;----------------Inicialização REGs para controle da memória eeprom I2C, 24LC1025----------------------
        CLRF        ENDERECO_LOW        ; Inicia endereço Memória I2C menos signif c/ 0x00
        CLRF        ENDERECO_HIGH       ; Inicia endereço Memória I2C mais signif c/ 0x00    CLRF
        A1A0_I2C                ; Inicia A1A0_I2C com 0x00
        MOVLW       0xA0                ; W← 0xA0
        MOVWF       CONTROLE_I2C        ; Inicia CONTROLE_I2C com 0xA0
        ;-------------------------------------------------------------------------------------------------------------------

        CALL        Mens_3              ; PROCESSO AUTOMA-
                                        ; TIZADO - AGUARDA

        CALL        DELAY_5S            ; agurda 10 seg.
        CALL        DELAY_5S

        ;-----------------Inicialização dos registradores X e Y-------------------------------------------------
        MOVLW       0x02                ; W ← 0x02 - valor inicial para Y [Detector]
        MOVWF       Y                   ; Y inicia com 2 e é incrementado até 92
        MOVLW       0x00                ; W ← 0x00 - valor inicial para X {cabeca]
        MOVWF       X                   ; X iniciado com 0 e é incrementado até 41
        ;-------------------------------------------------------------------------------------------------------------------

        ;----------------------Inicialmente ESPAÇO nas posições do Display-----------------------------------
        MOVLW       ' '                 ; W ← blank, ou W ← 0x20
        MOVWF       L2_C0               ; L2_C0  ← W
        MOVWF       L2_C1               ; L2_C1  ← W
        MOVWF       L2_C2               ; L2_C2  ← W
        MOVWF       L2_C3               ; L2_C3  ← W
        MOVWF       L2_C4               ; L2_C4  ← W
        MOVWF       L2_C5               ; L2_C5  ← W
        MOVWF       L2_C6               ; L2_C6  ← W
        MOVWF       L2_C7               ; L2_C7  ← W
        MOVWF       L2_C8               ; L2_C8  ← W
        MOVWF       L2_C9               ; L2_C9  ← W
        MOVWF       L2_C10              ; L2_C10 ← W
        MOVWF       L2_C11              ; L2_C11 ← W
        MOVWF       L2_C12              ; L2_C12 ← W
        MOVWF       L2_C13              ; L2_C13 ← W
        MOVWF       L2_C14              ; L2_C14 ← W
        MOVWF       L2_C15              ; L2_C15 ← W
    ;-------------------------------------------------------------------------------------------------------------------
```

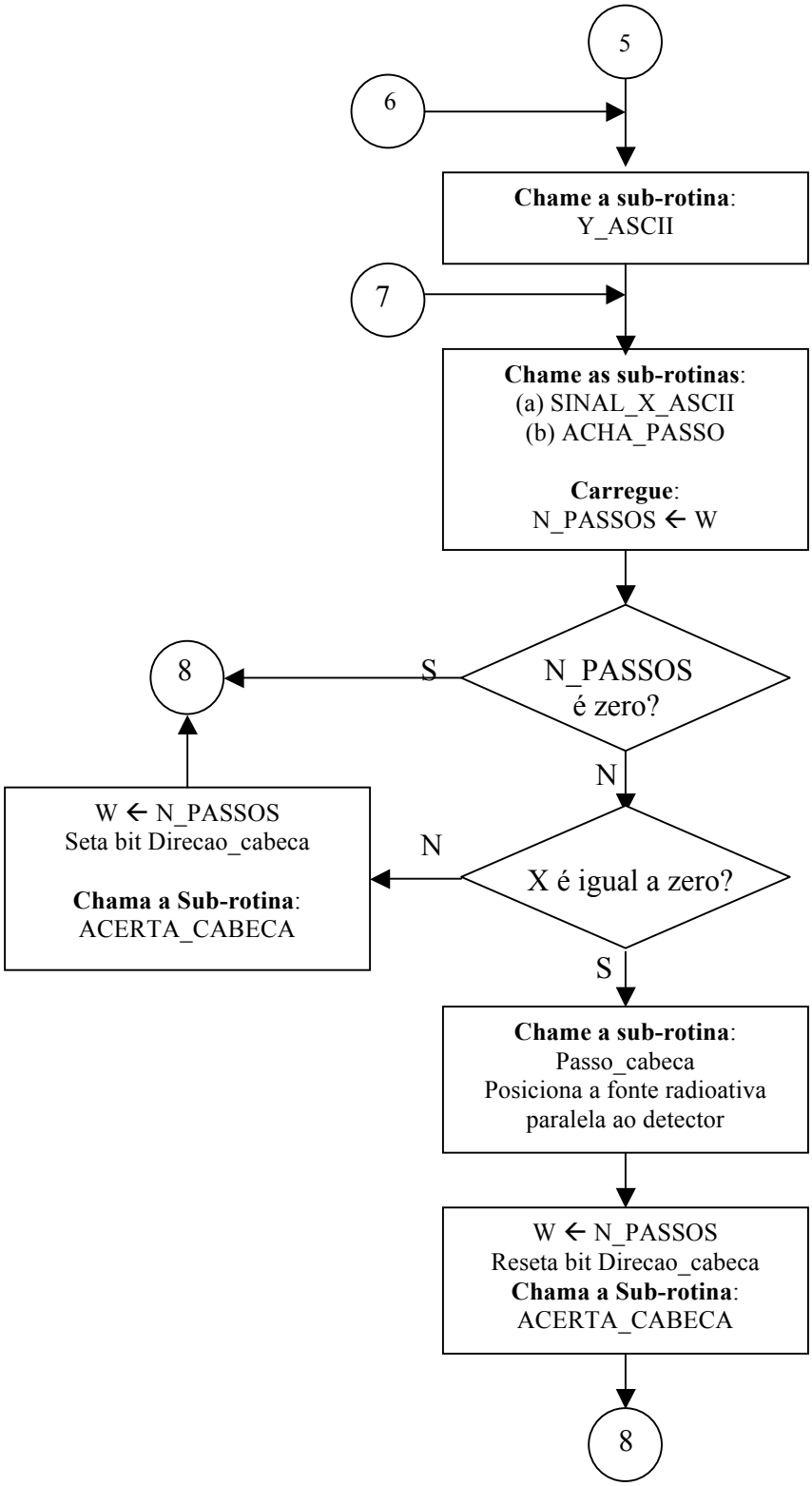


Figura A6– Fluxograma do Programa Principal
Processo Automático – intermediário.

```
;-------------------------------------------------------------------------------------------------------------------------
REPETE_YX
        CALL            Y_ASCII                 ; Carrega REGs Y_DEZENA e Y_UNIDADE em BCD
                                                ; Carrega L1_C2 c/ Y_DEZENA EM ASCII
                                                ; Carrega L1_C3 c/ Y_UNIDADE EM ASCII
                                                ; Retorna na Página 0 → Bit 4 e 3 PCLATH = 00
;-------------------------------------------------------------------------------------------------------------------------

;-------------------------------------------------------------------------------------------------------------------------
REPETE_X
        PAGESEL         SINAL_X_ASCII           ; Ajusta bit 4 e 3 PCLATH p/ página 2 = 10
        CALL            SINAL_X_ASCII           ; Carrega REGs X_DEZENA e X_UNIDADE em BCD
                                                ; Carrega L1_C7 c/ SINAL_DE_X EM ASCII
                                                ; Carrega L1_C8 c/ X_DEZENA EM ASCII
                                                ; Carrega L1_C9 c/ X_UNIDADE EM ASCII
                                                ; Retorna na Página 2 → Bit 3 e 4 PCLATH = 10

        PAGESEL         ACHA_PASSO              ; Para Y e X encontrar angulo da cabeca
        CALL            ACHA_PASSO              ; A quantidade de passos encontrar-se-á em W
                                                ; Retorna em diversas páginas, ajustar PCLATH 10

        PAGESEL         REPETE_X                ; Ajuste PCLATH = página 2 = B'10'

        MOVWF           N_PASSOS                ; N_PASSOS ← W [número de passos em hexadecimal].
        ;---------------------------------------------------------------------------------------------------------------

        ;------------------------Verificar se número de passos é ZERO-----------------------------------------------
        ANDLW           0xFF                     ; Verifica se N_PASSOS = 0x00
        BTFSC           STATUS,Z                 ; Testa bit Z do registrador STATUS salta próx. se 0
        GOTO            POS_ACERTO_CABECA        ; Se N_PASSOS igual a 0 → Z=1 executa GOTO
                                                 ; Se N_PASSOS diferente de 0 → Z=0 salta GOTO

       ;---É Executado qdo N_PASSOS for diferente de zero-----------------------------------------------------------
       ;-----------------------------Verificar se X=0, ou diferente de zero-----------------------------------------
        MOVF            X,W                     ; W ← X
        ANDLW           0xFF                    ; W ← W AND 0xFF, desta operação:
        BTFSS           STATUS,Z                ; se Z=1 → X = 0x00; salta próxima instrucao
        GOTO            X_DIFERE_0              ; se Z=0 → X diferente de 0x00, então faz o GOTO.

       ;-----------------------------Só será executado Qdo X=0------------------------------------------------------
        CALL            Passo_cabeca            ; Faz a fonte ir a posicao 0,direcao do detector avança no
                                                ; sentido horário.
        MOVF            N_PASSOS,W              ; W ← N_PASSOS
        BCF             Direcao_cabeca          ; prepara para girar cabeca no sentido anti-horário
        CALL            ACERTA_CABECA           ; acerta fonte rumo ao detector

        GOTO            POS_ACERTO_CABECA

        ;----------------------------Qdo X for diferente de zero-----------------------------------------------------
X_DIFERE_0
        MOVF            N_PASSOS,W              ; W ← N_PASSOS
        BSF             Direcao_cabeca          ; prepara para girar no sentido horário
        CALL            ACERTA_CABECA           ; acerta fonte rumo ao detector
        ;---------------------------------------------------------------------------------------------------------------
```

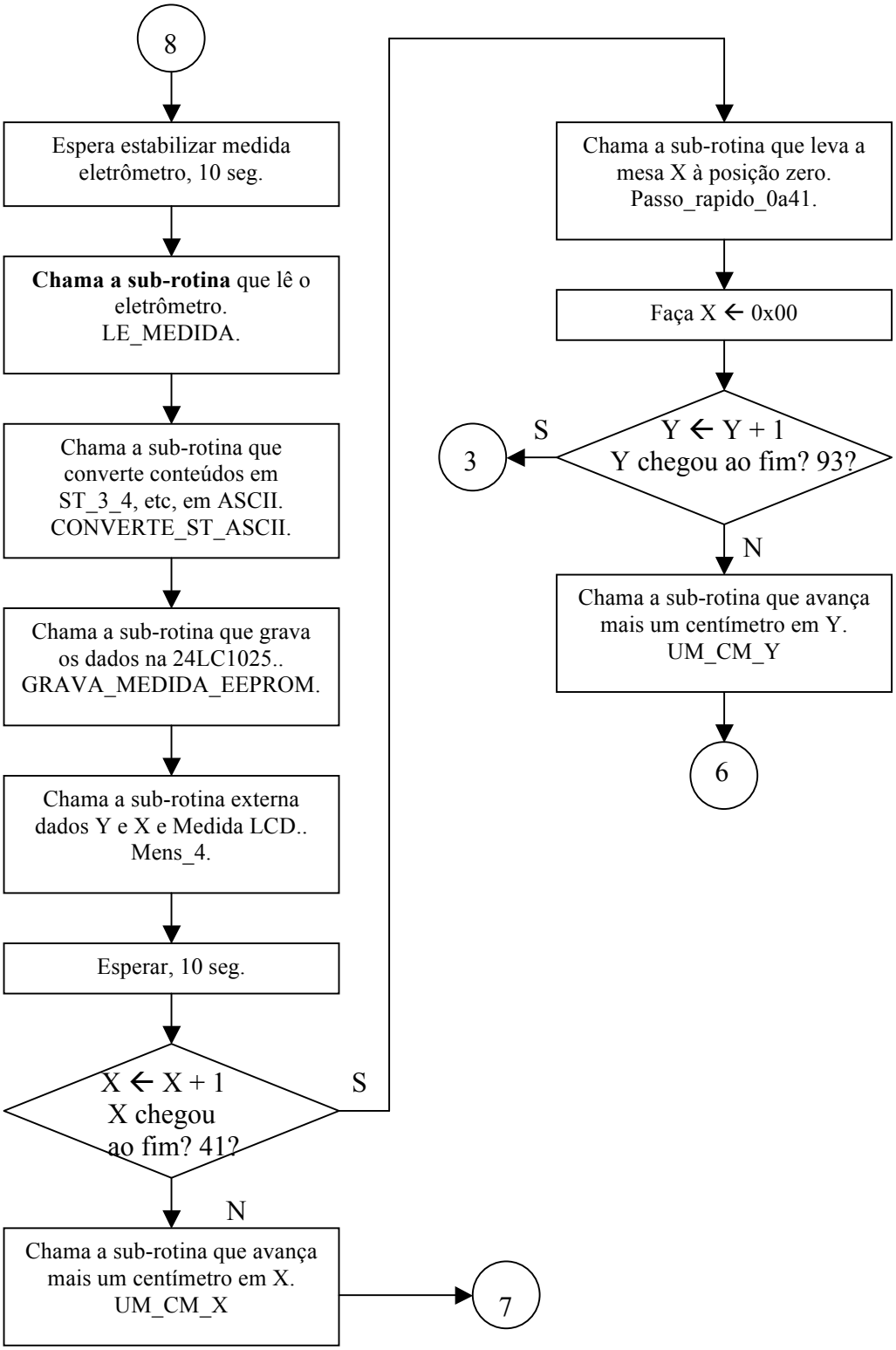


Figura A7– Fluxograma do Programa Principal – Processo Automático – final

```
        ;-------------------------------------------------------------------------------------------------------------------
        ; Após acertar a cabeça, esperará pela estabilização da medida para rodar a rotina de leitura do
        ; eletrômetro e escrita na EEPROM. [Então, após isto far-se-á X mover 1 cm.].
        ;-------------------------------------------------------------------------------------------------------------------
POS_ACERTO_CABECA
        ;---------------------Espera estabilizar medida eletrômetro-----------------------------------------------
        PAGESEL     DELAY_5S            ; Aguarda em torno de 10 segundos
        CALL        DELAY_5S
        CALL        DELAY_5S
        ;-------------------------------------------------------------------------------------------------------------------
        PAGESEL     LE_MEDIDA           ; Lê porta B [eletrômetro] guardando-as nos REGs de
        CALL        LE_MEDIDA           ; ST_1_2, ST_3_4, ST_5_6, ST_7_8 e ST_9  em BCD
        CALL        CONVERTE_ST_ASCII   ; ST_3_4, ST_5_6. ST_7_8 e ST_9 p/ ASCII colocando
                                        ; nos REGs para o LCD de L2_C0 a L2_c15
        ;--------Uma vez os dados da medida já armazenados, deve-se salvá-los na EEPROM I2C--------
        PAGESEL     GRAVA_MEDIDA_EEPROM     ; Ajusta PCLATH
        CALL        GRAVA_MEDIDA_EEPROM
        ;-------------------------------------------------------------------------------------------------------------
        PAGESEL     Mens_4              ; Y=90bX=+20bMedid  [exemplo]
        CALL        Mens_4              ; bb+0,1973E+19bCb
        PAGESEL     DELAY_5S            ; Delay para manter dado display por 10 seg.
        CALL        DELAY_5S
        CALL        DELAY_5S
        BSF         PCLATH,3
        BSF         PCLATH,4            ; Ajusta PCLATH
        INCF        X,F                 ; X ← X + 1
        MOVF        X,W                 ; W ← X
        BCF         STATUS,Z            ; GARANTE Z=0 ANTES SUBTRAÇÃO
        SUBLW       .41                 ; W ← 41 - W
        BTFSS       STATUS,Z            ; salta a próxima instrução se x=41
        GOTO        $+2
        GOTO        ATUALIZA_XY
        PAGESEL     UM_CM_X             ; Faz mesa (0a41) X mover 1 cm
        CALL        UM_CM_X
        GOTO        REPETE_X
ATUALIZA_XY
        PAGESEL     Passo_rapido_0a41   ; Ajusta PCLATH
        CALL        Passo_rapido_0a41
        BSF         PCLATH,3
        BSF         PCLATH,4            ; Ajusta PCLATH
        MOVLW       0x00                ; w← 0x00 - valor inicial para X {cabeça]
        MOVWF       X                   ; X inicia com 0 que será incrementado até 41
        INCF        Y,F                 ; Y ← Y + 1
        MOVF        Y,W                 ; W ← Y
        BCF         STATUS,Z            ; GARANTE Z=0 ANTES SUBTRAÇÃO
        SUBLW       .93                 ; W ← 93 - W
        BTFSS       STATUS,Z            ; salta a próxima instrução se y=93
        GOTO        $+2
        GOTO        $+5
        PAGESEL     UM_CM_Y             ; Faz mesa (2 a 92) Y mover a passo de 1 cm
        CALL        UM_CM_Y
        GOTO        REPETE_YX
        PAGESEL     DENOVO
        GOTO        DENOVO
  ;-------------------------------------------------------------------------------------------------------------------
```

Terminada as execuções das Fig. A5 a Fig. A7 o processamento volta ao *menu* principal (Fig. A3) e se o operador pressionar a tecla <3> o direcionará à rotina de transmissão de dados seriais (via interface RS232) para o CP, conforme mostra a Fig. A8. Primeiramente, o cabo desta interface deverá estar conectada entre a placa com o PIC e o conector DB9 do computador pessoal. E o software hyperterminal (usado nesse projeto) deverá estar ativo no CP. A transmissão do relatório (arquivo .TXT) não demanda muito tempo. Este relatório foi desenvolvido em formato de colunas, o que torna fácil sua migração a uma planilha eletrônica.

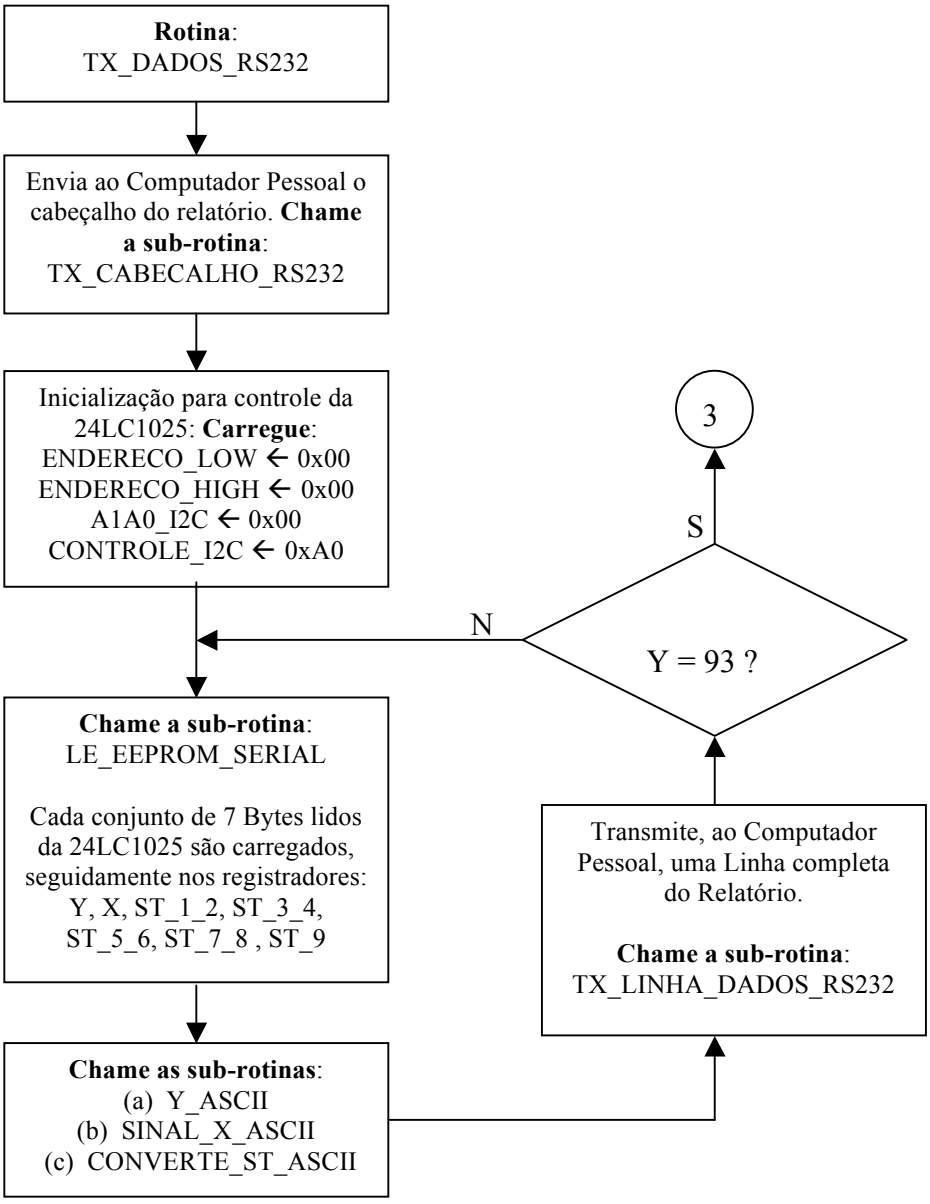


Figura A8– Fluxograma do Programa Principal.
TX dados ao Computador Pessoal.

```
;*******************************************************************************
;                    Esta rotina Transmite os dados ao Computador Pessoal via RS232.          *
;*******************************************************************************
TX_DADOS_RS232
        PAGESEL     TX_CABECALHO_RS232
        CALL        TX_CABECALHO_RS232
        ;----------------Inicializa memória 24LC1025---------------------------------
        CLRF        ENDERECO_LOW        ; Inicia ENDERECO MENOS SIGNIFICATIVO c/ 0x00
        CLRF        ENDERECO_HIGH       ; Inicia ENDERECO MAIS SIGNIFICATIVO c/ 0x00
        CLRF        A1A0_I2C            ; INICIA A1A0_I2C COM 0x00
        MOVLW       0xA0                ; W← 0xA0
        MOVWF       CONTROLE_I2C        ; INICIA CONTROLE_I2C COM 0xA0
        ;----------------fim da inicialização da memória 24LC1025---------------------

OUTRA_LINHA
        PAGESEL     LE_EEPROM_SERIAL    ; Lê EEPROM 24LC1025 guardando nos REGs Y e X
        CALL        LE_EEPROM_SERIAL    ; e de ST_1_2, ST_3_4, ST_5_6, ST_7_8 e ST_9 todos
                                        ; em BCD

        PAGESEL     Y_ASCII             ; W ← Y é carregado no início dessa sub-rotina
        CALL        Y_ASCII             ; Carrega REGs Y_DEZENA e Y_UNIDADE em BCD
                                        ; Carrega em BCD Y_DEZENA  e  Y_UNIDADE.
                                        ; Carrega L1_C2 c/ Y_DEZENA EM ASCII
                                        ; Carrega L1_C3 c/ Y_UNIDADE EM ASCII

        PAGESEL     SINAL_X_ASCII       ; W ← X é carregado no início dessa sub-rotina
        CALL        SINAL_X_ASCII       ; Ajusta bit 4 e 3 PCLATH p/ página 2 = 10
                                        ; Carrega REGs X_DEZENA e X_UNIDADE em BC
                                        ; Carrega L1_C7 C/ SINAL_DE_X EM ASCII
                                        ; Carrega L1_C8 C/ X_DEZENA EM ASCII
                                        ; Carrega L1_C9 C/ X_UNIDADE EM ASCII

        PAGESEL     CONVERTE_ST_ASCII   ; Converte a medida eletrômetro armazenadas nos REG s
                                        ; ST_1_2
        CALL        CONVERTE_ST_ASCII   ; ST_3_4, ST_5_6. ST_7_8 e ST_9 p/ ASCII colocando
                                        ; nos REGs para o LCD de L2_C0 a L2_C15

        PAGESEL     TX_LINHA_DADOS_RS232
        CALL        TX_LINHA_DADOS_RS232

        MOVF        Y,W                 ; W ← Y
        BCF         STATUS,Z            ; Garante  Z=0 antes da subtração
        SUBLW       .93                 ; W ← 93 - W
        BTFSS       STATUS,Z            ; salta a próxima instrução se y=93
        GOTO        OUTRA_LINHA

        PAGESEL     DENOVO
        GOTO        DENOVO
;*******************************************************************************
```

b) Sub-rotinas de suporte ao programa principal

b1) Sub-rotina para Leitura do Eletrômetro

As Fig. A9 e Fig. A10 mostram o fluxograma da da sub-rotina para leitura do eletrômetro.

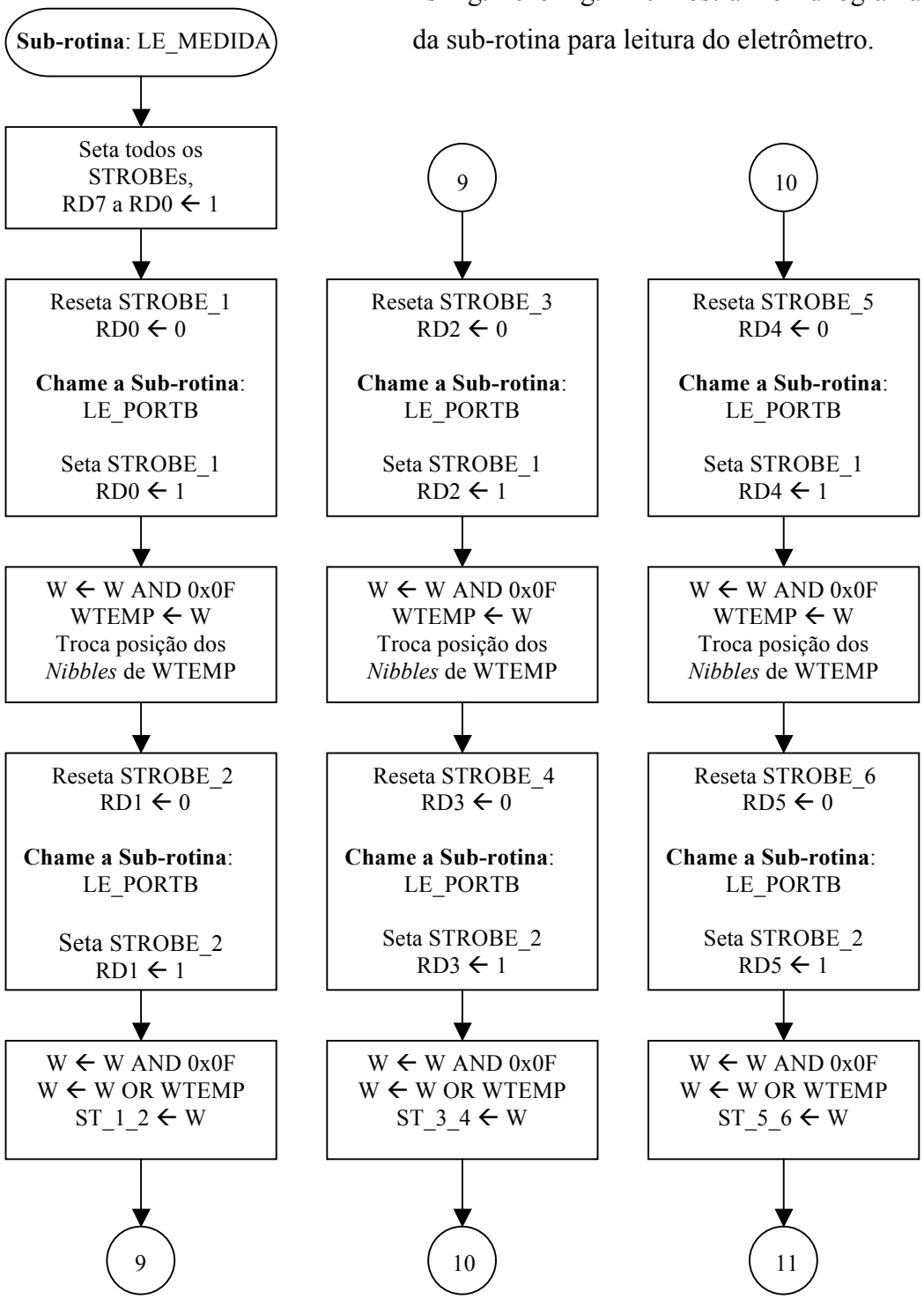


Figura A9– Fluxograma da Sub-rotina: Leitura do Eletrômetro, inicial.

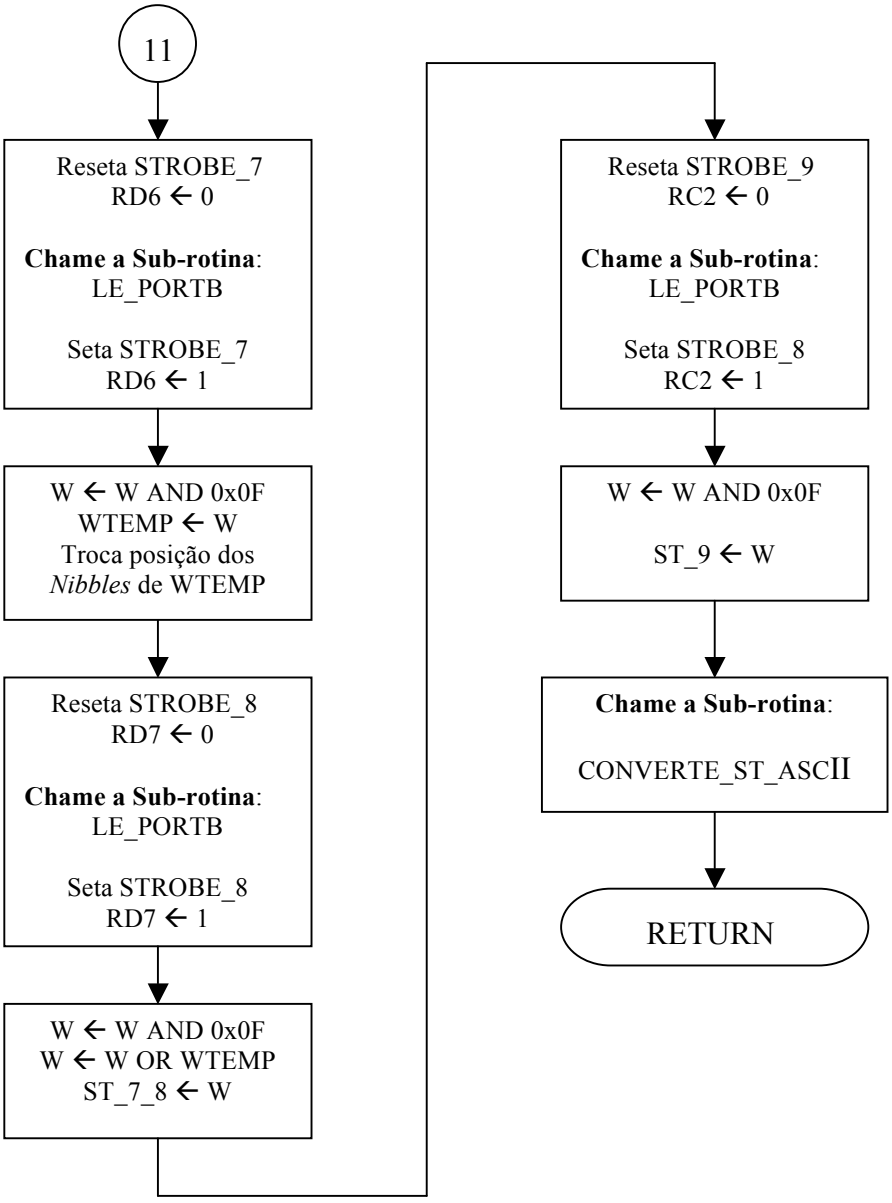


Figura A10– Fluxograma da Sub-rotina: Leitura do Eletrômetro, final.

```
;*******************************************************************************
; *                    Sub-rotina utilizada para fazer as leituras do eletrômetro                    *
;*******************************************************************************
LE_MEDIDA

        MOVLW       0xFF            ; W ← 0xFF
        MOVWF       PORTD           ; seta os bits da Porta D, STROBEs ← 1. PORTD ←W
        BSF         STROBE_9        ; seta o bit 2 da PORTC.

    ;-------------------------------------------------------------------------------------------------------------------
    ;                           Leitura de DADOS do ELETROMETRO
    ;-------------------------------------------------------------------------------------------------------------------

    ;----------------------Carregamento do registrador ST_1_2 ----------------------------------------------------

        BCF         STROBE_1        ; Gera sinal STROBE_1[RD0] ou reseta bit 0 da PORTD
        PAGESEL     LE_PORTB
        CALL        LE_PORTB        ; Resultado da leitura da PORTB retorna em W
        BSF         STROBE_1        ; Seta o bit 0 da PORTD [RD0].

        ANDLW       0x0F            ; W ← W AND k [mascaramento dos bits menos sig].
        MOVWF       WTEMP           ; WTEMP ← W
        SWAPF       WTEMP,F         ; troca nibbles de posição e resultado em WTEMP

        BCF         STROBE_2        ; Gera sinal STROBE_2 [RD1]ou reseta bit 1 da PORTD
        PAGESEL     LE_PORTB
        CALL        LE_PORTB        ; Resultado da leitura da PORTB retorna em W
        BSF         STROBE_2        ; Seta o bit 1 da PORTD [RD1].

        ANDLW       0x0F            ; W ← W AND k [mascaramento dos bits menos sig].
        IORWF       WTEMP,w         ; W ← W OR WTEMP [em W dados STROBE 1 e 2]
        MOVWF       ST_1_2          ; ST_1_2 ← W

    ;----------------------Carregamento do registrador ST_3_4 ----------------------------------------------------

        BCF         STROBE_3        ; Gera sinal STROBE_3 ou reseta bit 2 da PORTD.
        PAGESEL     LE_PORTB
        CALL        LE_PORTB        ; Resultado da leitura residirá em W
        BSF         STROBE_3        ; Seta o bit 2 da PORTD.

        ANDLW       0x0F            ; W ← W AND k [mascaramento dos bits menos sig].
        MOVWF       WTEMP           ; WTEMP ← W
        SWAPF       WTEMP,F         ; troca nibbles de posição e põe em WTEMP

        BCF         STROBE_4        ; Gera sinal STROBE_4 ou reseta bit 3 da PORTD.
        PAGESEL     LE_PORTB
        CALL        LE_PORTB        ; Resultado da leitura residirá em W
        BSF         STROBE_4        ; Seta o bit 3 da Porta D PORTD.

        ANDLW       0x0F            ; W ← W AND k [mascaramento dos bits menos sig].
        IORWF       WTEMP,w         ; W ← W OR WTEMP [em W dados STROBE 3 e 4]
        MOVWF       ST_3_4          ; ST_3_4 ← W

    ;----------------------Carregamento do registrador ST_5_6 ----------------------------------------------------
```

```
    BCF         STROBE_5            ; Gera sinal STROBE_5 ou reseta bit 4 da PORTD
    PAGESEL     LE_PORTB
    CALL        LE_PORTB            ; Resultado da leitura residirá em W
    BSF         STROBE_5            ; Seta o bit 4 da PORTD.

    ANDLW       0x0F                ; W ← W AND k [mascaramento dos bits menos sig].
    MOVWF       WTEMP               ; WTEMP ← W
    SWAPF       WTEMP,F             ; troca nibbles de posicao e põe em WTEMP

    BCF         STROBE_6            ; Gera sinal STROBE_6 ou reseta bit 5 da PORTD
    PAGESEL     LE_PORTB
    CALL        LE_PORTB            ; Resultado da leitura residirá em W
    BSF         STROBE_6            ; Seta o bit 5 da PORTD.

    ANDLW       0x0F                ; W ← W AND k [mascaramento dos bits menos sig].
    IORWF       WTEMP,w             ; W ← W OR WTEMP [em W dados STROBE 5 e 6]
    MOVWF       ST_5_6              ; ST_5_6 ← W

;----------------------Carregamento do registrador ST_7_8 ----------------------------------------------------

    BCF         STROBE_7            ; Gera sinal STROBE_7 ou reseta bit 6 da PORTD
    PAGESEL     LE_PORTB
    CALL        LE_PORTB            ; Resultado da leitura residirá em W
    BSF         STROBE_7            ; Seta o bit 6 da PORTD.

    ANDLW       0x0F                ; W ← W AND k [mascaramento dos bits menos sig].
    MOVWF       WTEMP               ; WTEMP ← W
    SWAPF       WTEMP,F             ; troca nibbles de posicao e põe em WTEMP

    BCF         STROBE_8            ; Gera sinal STROBE_8 ou reseta bit 7 da PORTD
    PAGESEL     LE_PORTB
    CALL        LE_PORTB            ; Resultado da leitura residirá em W
    BSF         STROBE_8            ; Seta o bit 7 da PORTD.

    ANDLW       0x0F                ; W ← W AND k [mascaramento dos bits menos sig].
    IORWF       WTEMP,w             ; W ← W OR WTEMP [em W dados STROBE 5 e 6]
    MOVWF       ST_7_8              ; ST_7_8 ← W

;----------------------Carregamento do registrador ST_9 ----------------------------------------------------

    BCF         STROBE_9            ; Gera sinal STROBE_9 ou reseta bit 2 da PORTC
    PAGESEL     LE_PORTB
    CALL        LE_PORTB            ; Resultado da leitura residirá em W
    BSF         STROBE_9            ; Seta o bit 2 da PORTC.

    ANDLW       0x0F                ; W ← W AND k [mascaramento dos bits menos sig].
    MOVWF       ST_9                ; ST_9 ← W
;-------------------------------------------------------------------------------------------------------------------------------
;                                   FIM das Leitura de DADOS do ELETROMETRO
;-------------------------------------------------------------------------------------------------------------------------------

    PAGESEL     CONVERTE_ST_ASCII   ; Converte a medida eletrômetro armazenadas nos REG s
                                    ; ST_1_2
    CALL        CONVERTE_ST_ASCII   ; ST_3_4, ST_5_6. ST_7_8 e ST_9 p/ ASCII colocando
                                    ; nos REGs para o LCD de L2_C0 a L2_C15
    RETURN
```

b2) Sub-rotina para Ler o conteúdo da PORTB

A Fig. A11 mostra o fluxograma necessário à implementação do programa *assembly* de leitura da porta PORTB.

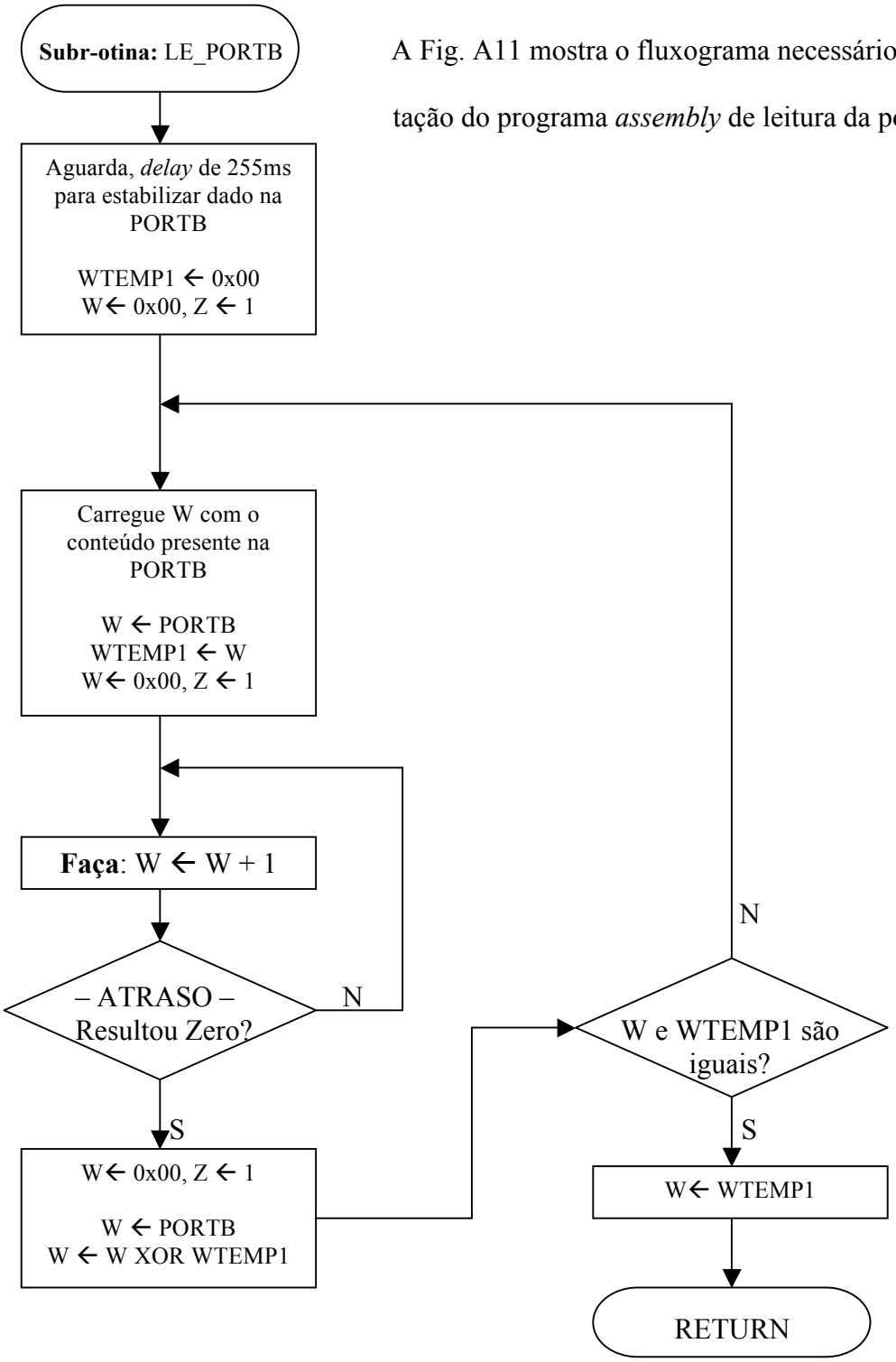


Figura A11– Fluxograma da Sub-rotina: Ler o conteúdo da PORTB.

```
;-------------------------------------------------------------------------------------------------------------------
;                                        Rotina para Leitura da PORTB
;                                             Resultado em W
;-------------------------------------------------------------------------------------------------------------------

        ;-----------------------Leitura Porta B [da medida presente no eletrômetro]-------------------------------
LE_PORTB
        MOVLW       .255                    ; Valor em W indica quantos ms de Delay.
        CALL        DELAY_MS                ; DELAY 255ms p/ estabilizar dado na porta.

        CLRF        WTEMP1                  ; WTEMP1 ← 0x00
        CLRW                                ; W ← 00h e Z ← 1

TENTA_OUTRA_VEZ
        MOVF        PORTB,W                 ; W ← PORTB
        MOVWF       WTEMP1                  ; WTEMP1 ← W
        CLRW                                ; W ← 00h e Z ← 1
    ;--------------------ATRASO-----------------------------------------------------------------------------------
ATRASO
        ADDLW       .1                      ; W ← W + k [1a vez = 0+1=1 difere de 0,
                                            ; ou  Z←0]
        BTFSC       STATUS,Z                ; Testa bit Z e salta se Z=0, quando resultado dif 0
        GOTO        $+2
        GOTO        ATRASO                  ; Manterá neste loop por 256 vezes até Z ←1
        ;-------------------------------------------------------------------------------------------------------------

        CLRW                                ; W ← 00h e Z ← 1

        MOVF        PORTB,W                 ; W ← PORTB
        XORWF       WTEMP1,W                ; W ← W XOR WTEMP1
        BTFSS       STATUS,Z                ; se W e WTEMP forem iguais salta [Z = 1]
        GOTO        TENTA_OUTRA_VEZ         ; se forem diferentes [Z =0] vá para:

        MOVF        WTEMP1,W                ; W ← WTEMP1
        RETURN                              ; resultado da leitura retorna em W
;-------------------------------------------------------------------------------------------------------------------
```

b3) Sub-rotina CONVERTE_ST_ASCII

Esta sub-rotina converte os conteúdos dos registradores ST em ASCII, conforme mostrado nas Fig. A12 a Fig. A20.

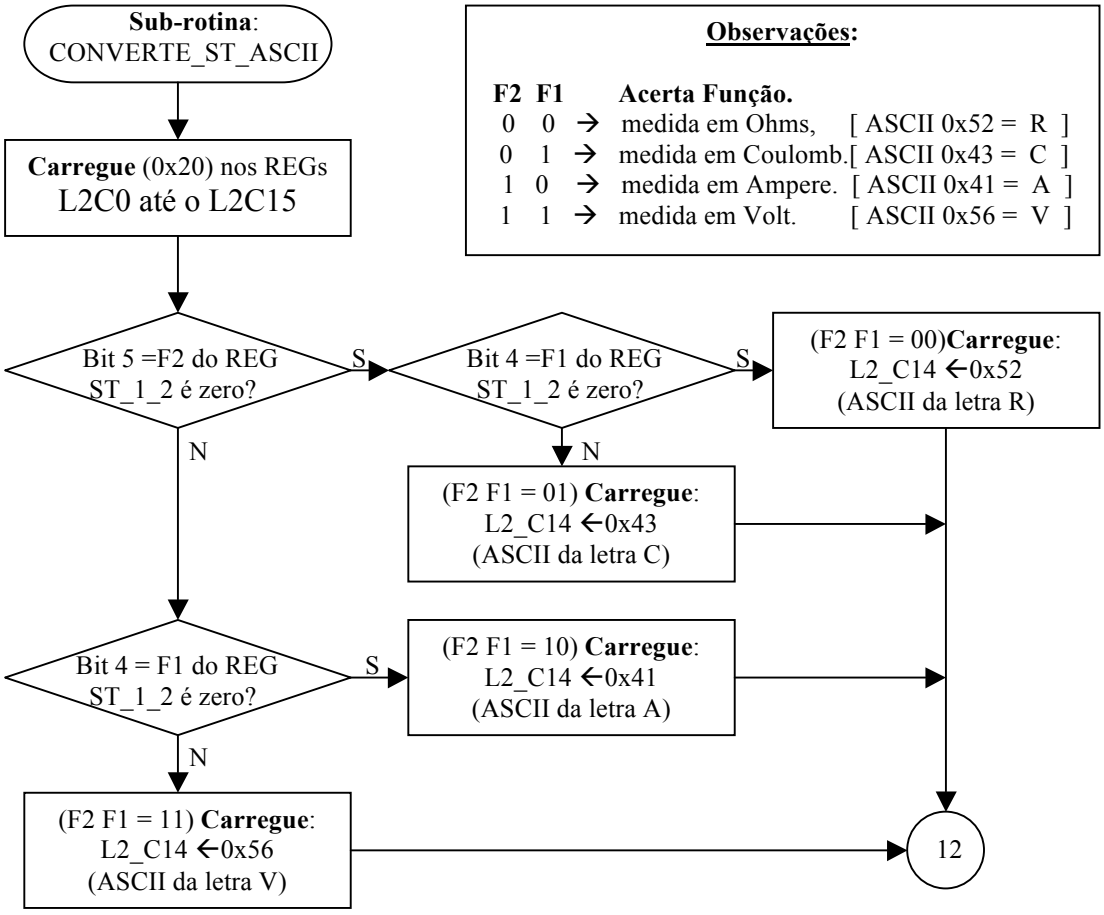


Figura A12– Fluxograma da Sub-rotina: Converte ST_ASCII.
Encontra unidade da medida.

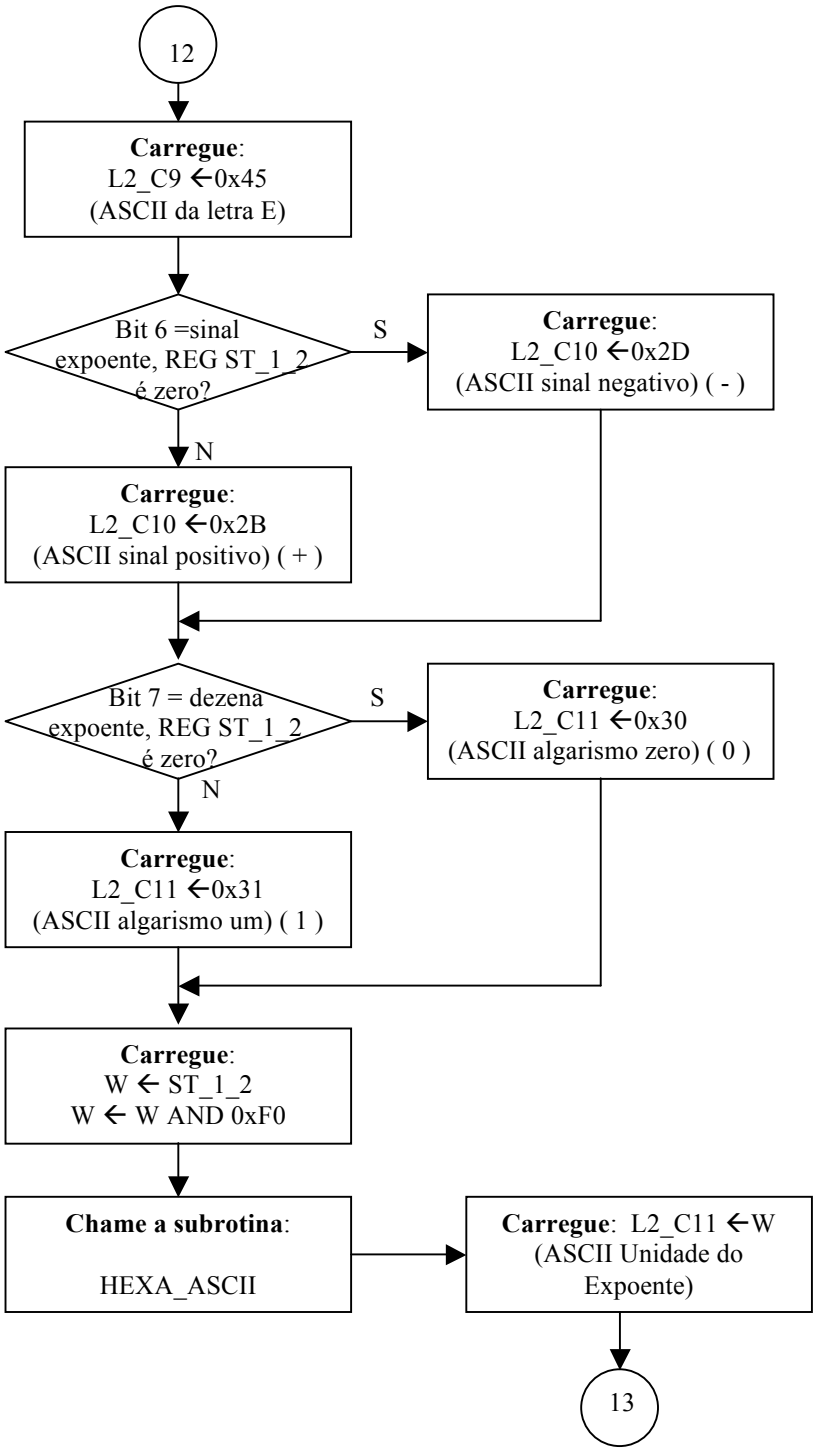


Figura A13– Fluxograma da Sub-rotina: Converte ST_ASCII.
Trata expoente da medida.

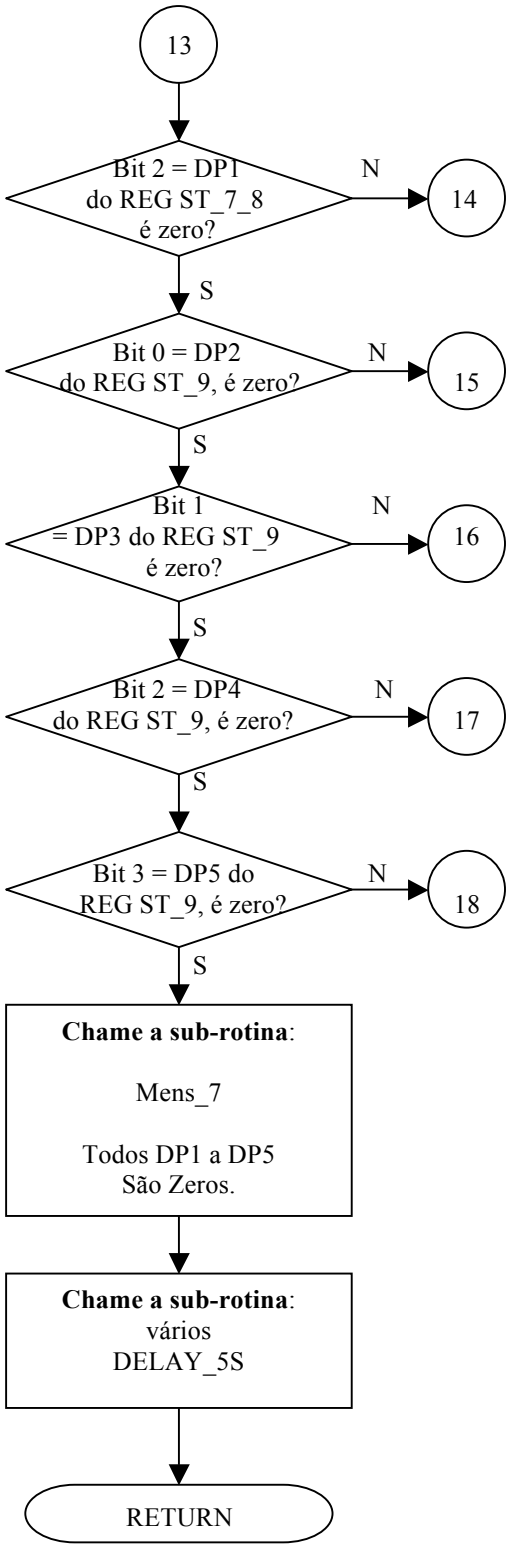


Figura A14– Fluxograma da Sub-rotina: Converte ST_ASCII. Determina a vírgula da medida.

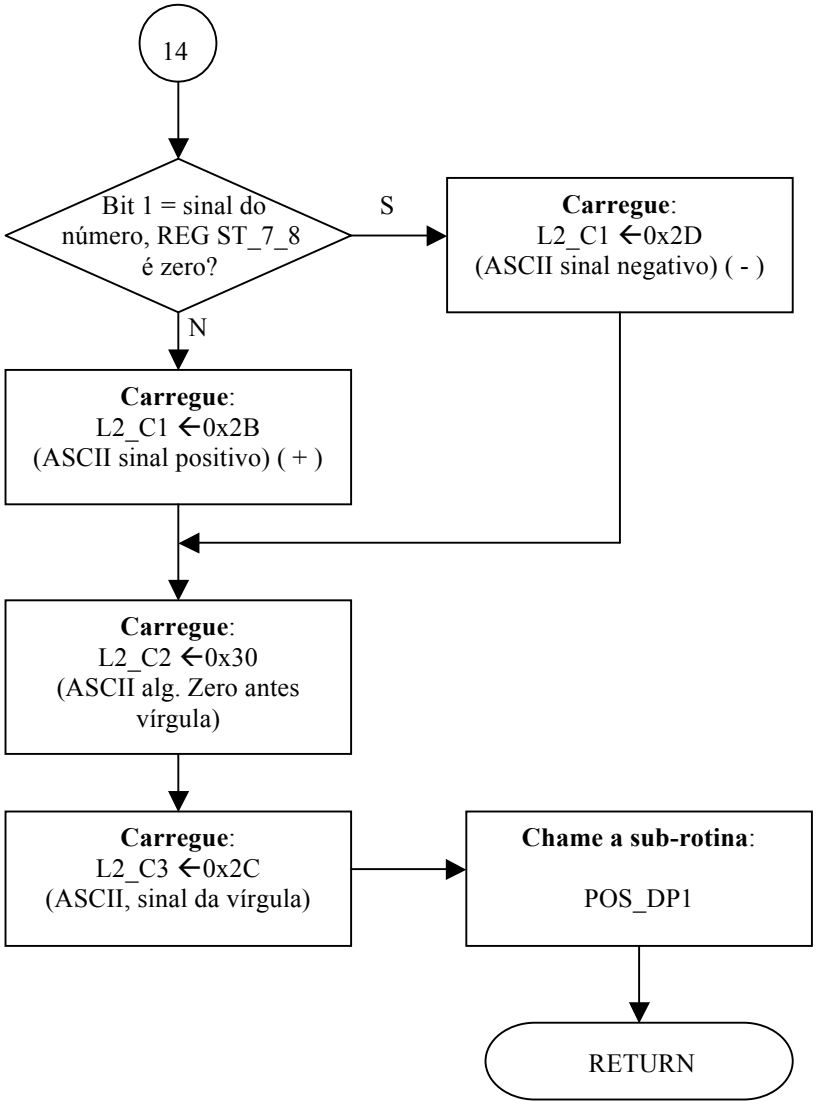


Figura A15– Fluxograma da Sub-rotina: Converte ST_ASCII.
Trata sinal e vírgula – DP1.

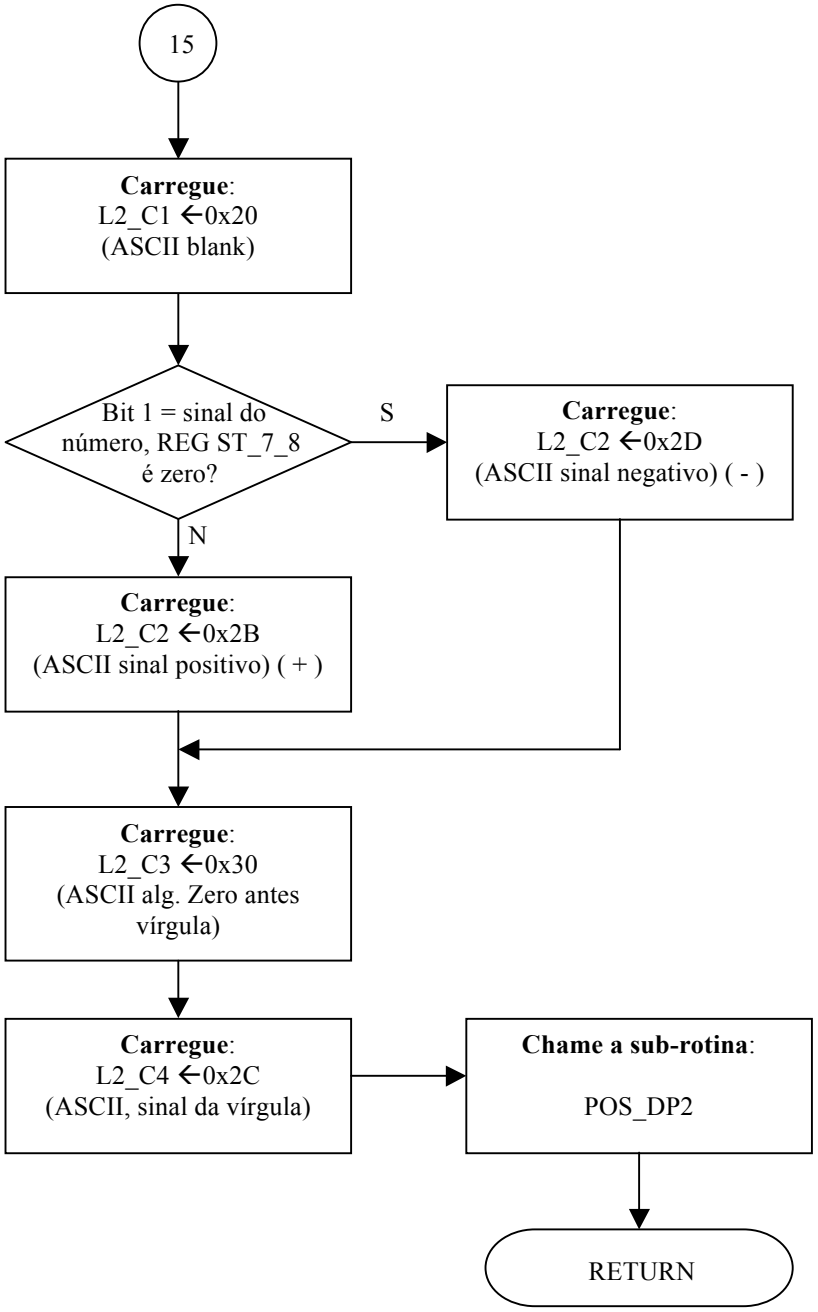


Figura A16– Fluxograma da Sub-rotina: Converte ST_ASCII.
Trata sinal e vírgula – DP2.

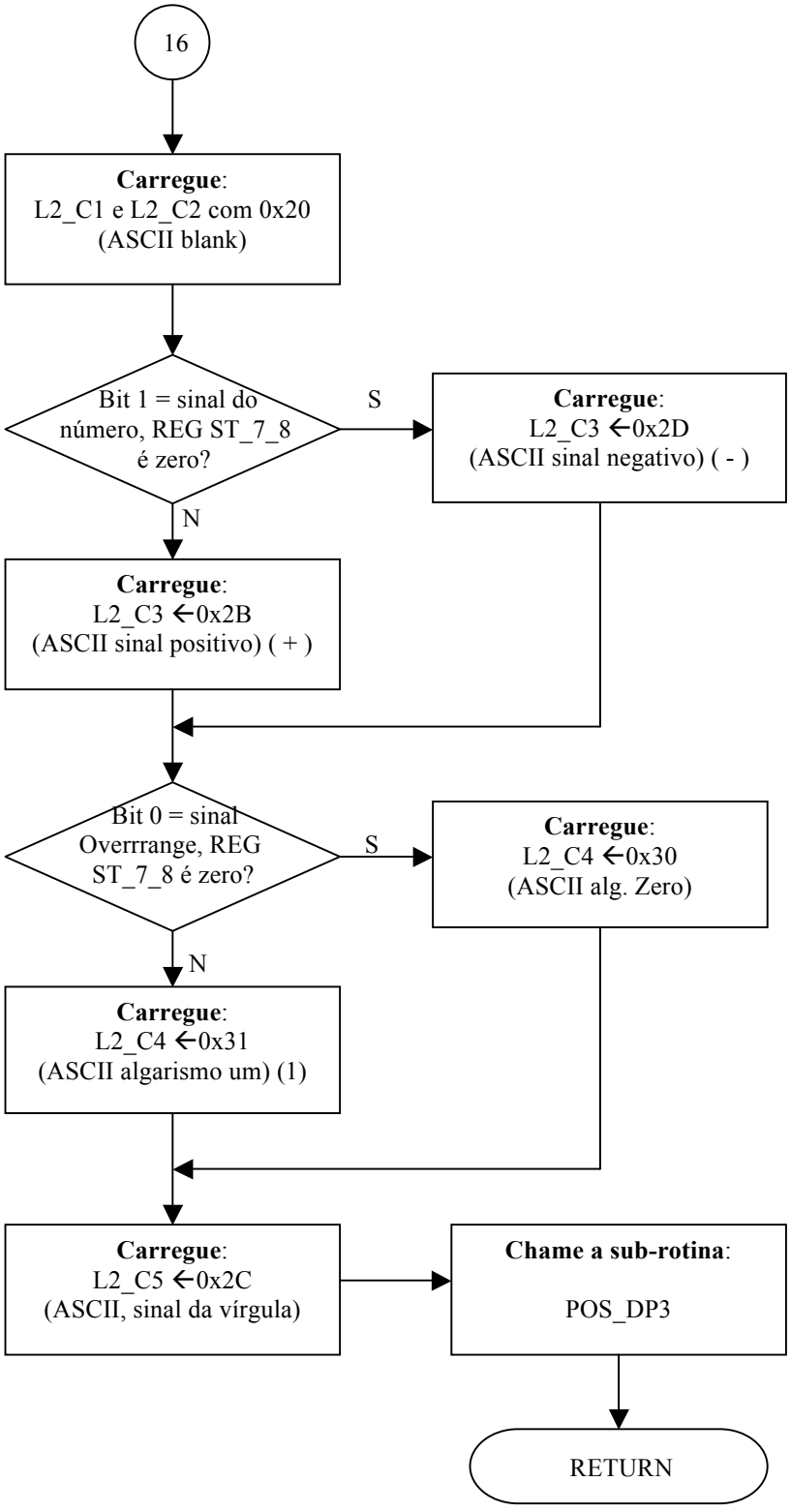


Figura A17– Fluxograma da Sub-rotina: Converte ST_ASCII.
Trata sinal e vírgula – DP3.

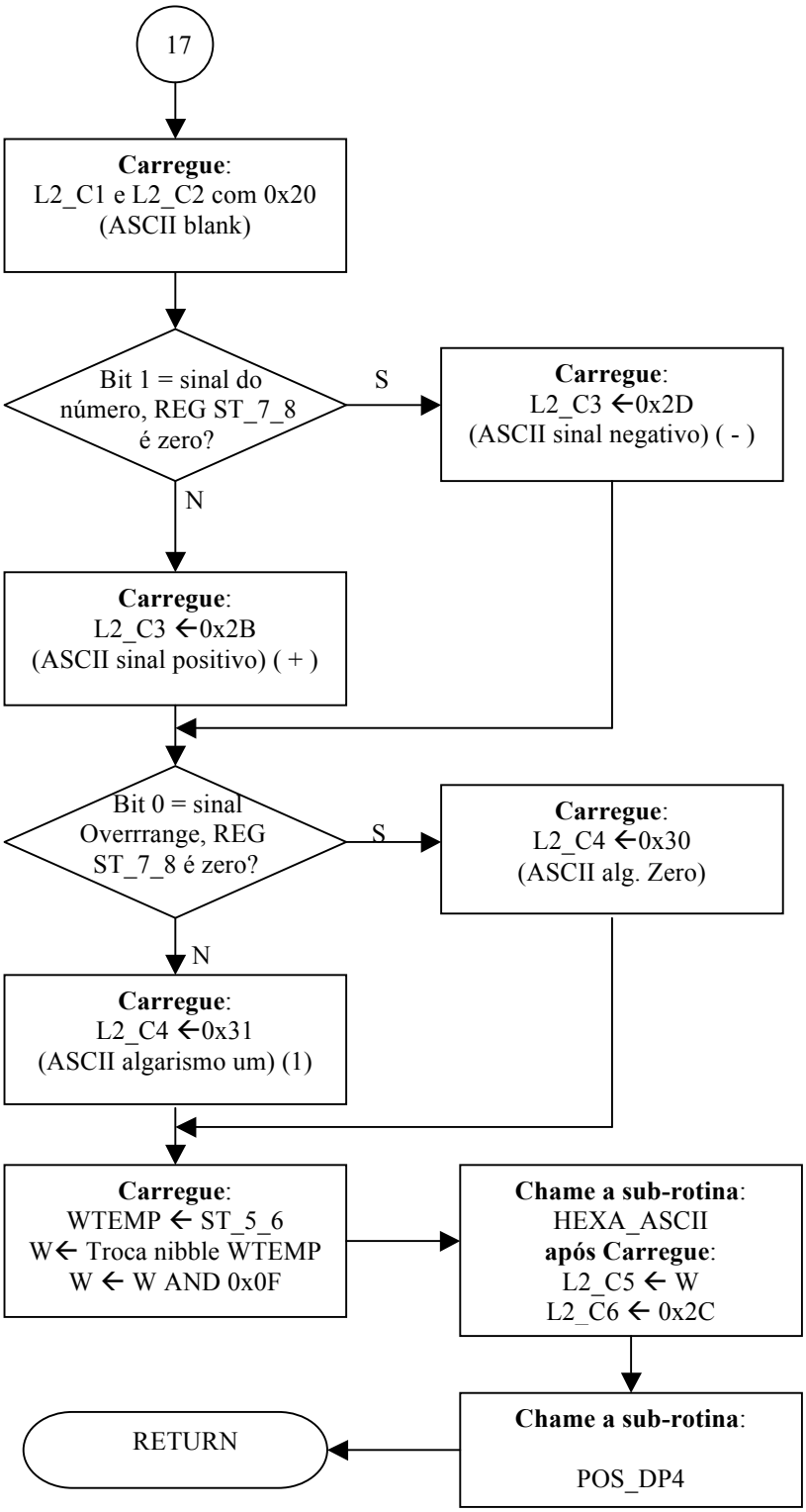


Figura A18– Fluxograma da Sub-rotina:
Converte ST_ASCII. Trata sinal e vírgula – DP4.

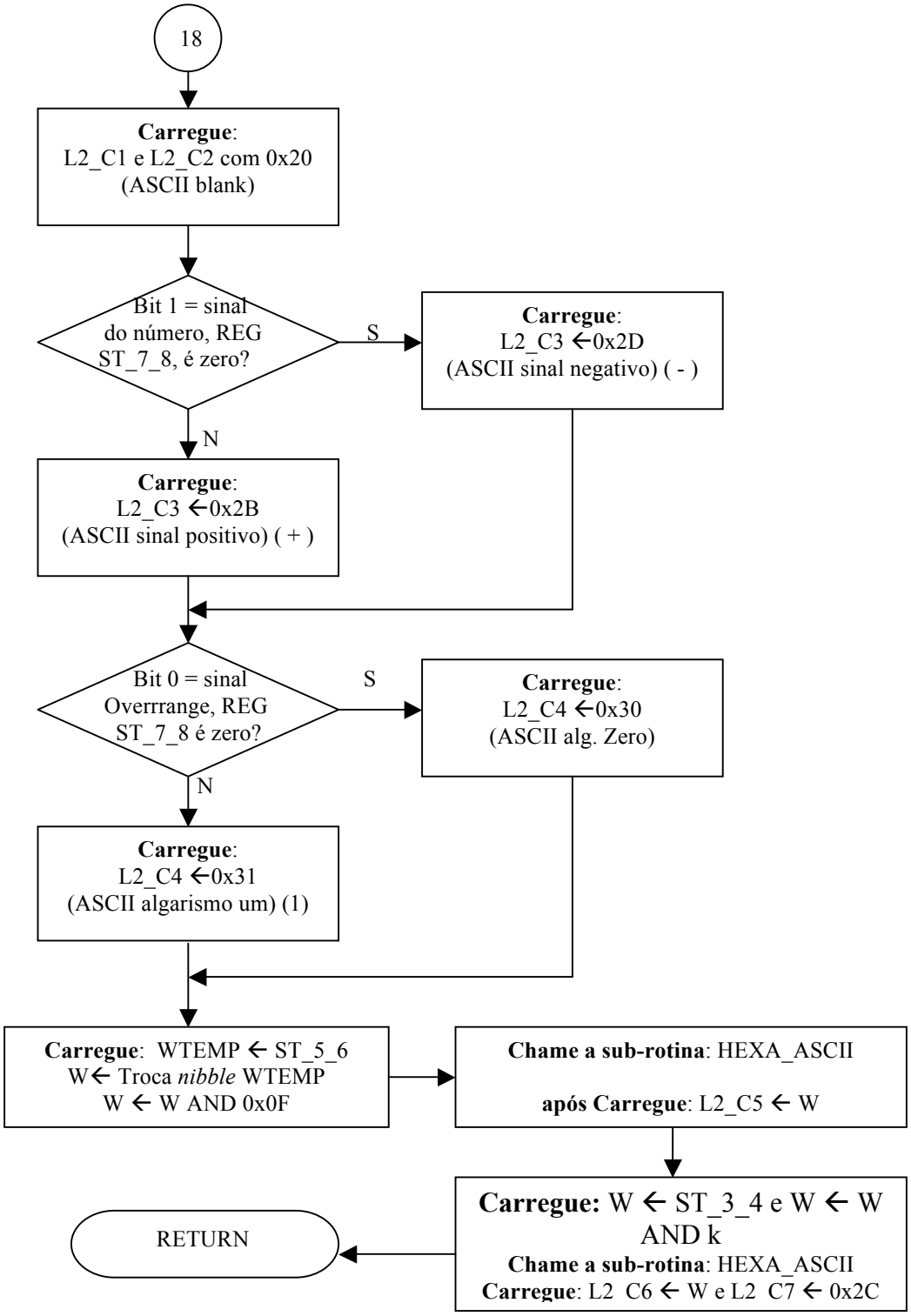


Figura A19– Fluxograma da Sub-rotina:
Converte ST_ASCII. Trata sinal e vírgula – DP5.

```
;*******************************************************************************
; Os REGs ST_1_2; ST_3_4, ST_5_6. ST_7_8 e ST_9 v6em com seu dado. Esta Sub-rotina converte-os para
; ASCII armazenando o resultado nos REGs para o LCD de L2_C0 a L2_C15
;*******************************************************************************
CONVERTE_ST_ASCII

        ;----------Análise dados nos REGs ST, convertendo-os p/ ASCII e L2C0 a C15----------

        BSF             PCLATH,3                ; Ajuste do PCLATH

        ;-------------------------ESPAÇO nas posições do Display-----------------------------------------------
        MOVLW           ' '                     ; W ← blank  ou W ← 0x20
        MOVWF           L2_C0                   ; L2_C0  ← W
        MOVWF           L2_C1                   ; L2_C1  ← W
        MOVWF           L2_C2                   ; L2_C2  ← W
        MOVWF           L2_C3                   ; L2_C3  ← W
        MOVWF           L2_C4                   ; L2_C4  ← W
        MOVWF           L2_C5                   ; L2_C5  ← W
        MOVWF           L2_C6                   ; L2_C6  ← W
        MOVWF           L2_C7                   ; L2_C7  ← W
        MOVWF           L2_C8                   ; L2_C8  ← W
        MOVWF           L2_C9                   ; L2_C9  ← W
        MOVWF           L2_C10                  ; L2_C10 ← W
        MOVWF           L2_C11                  ; L2_C11 ← W
        MOVWF           L2_C12                  ; L2_C12 ← W
        MOVWF           L2_C13                  ; L2_C13 ← W
        MOVWF           L2_C14                  ; L2_C14 ← W
        MOVWF           L2_C15                  ; L2_C15 ← W

        ;-------------------------------------------------------------------------------------------------------
        ;                       F2  F1      Acerta Função.
        ;                       0   0   →   medida em Ohms,     [ ASCII 0x52 =  R  ]
        ;                       0   1   →   medida em Coulomb.[ ASCII 0x43 =  C  ]
        ;                       1   0   →   medida em Ampere.  [ ASCII 0x41 =  A  ]
        ;                       1   1   →   medida em Volt.       [ ASCII 0x56 =  V  ]
        ;
        ; Obs.:   Bit 4 do reg  ST_1_2 é F1
        ;         Bit 5 do reg  ST_1_2 é F2
        ;-------------------------------------------------------------------------------------------------------

    ;-------------------------------------------------------------------------------------------------------
    ;                       Encontrar qual É A unidade da medida feita
    ;                   Se Ohm ( R ), Coulomb ( C ), Ampere ( A ) ou Volt ( V )
    ;-------------------------------------------------------------------------------------------------------
        BTFSC           ST_1_2,5                ; Salta [GOTO] se bit 5 for zero [bit 5 = F2]
        GOTO            F2_1                    ; vá para F2_1, pois F2=1[$+9]
        BTFSC           ST_1_2,4                ; Como F2=0, testa F1 [bit 4 = F1]
        GOTO            F1_1                    ; vá para F1_1, onde F2=0 e F1=1, Coulomb [$+4]
        MOVLW           0x52                    ; implica que F2=0 e F1=0, medida em Ohms.
        MOVWF           L2_C14                  ; 0x52 é o ASCII da letra R.
        GOTO            TRATA_EXPOENTE          ; $+11
F1_1
        MOVLW           0x43                    ; implica que F2=0 e F1=1, medida em Coulomb.
        MOVWF           L2_C14                  ; 0x43 é o ASCII da letra C
        GOTO            TRATA_EXPOENTE          ; $+8
```

```
F2_1
        BTFSC       ST_1_2,4            ; Como F2=1, testa F1 [bit 4 = F1]
        GOTO        F2_1_F1_1           ; vá para F1_1, onde F2=1 e F1=1, Volt.[$+4]
        MOVLW       0x41                ; implica que F2=1 e F1=0, medida em Ampere.
        MOVWF       L2_C14              ; 0x41 é o ASCII da letra A
        GOTO        TRATA_EXPOENTE
F2_1_F1_1
        MOVLW       0x56                ; implica que F2=1 e F1=1, medida em Volt.
        MOVWF       L2_C14              ; 0x56 é o ASCII da letra V
    ;-------------------------------------------------------------------------------------------------------------------

        ;-------------------------------------------------------------------------------------------------------------
        ;                   Tratamento do Expoente do número medido                                  (12)
        ;-------------------------------------------------------------------------------------------------------------
TRATA_EXPOENTE
        MOVLW       0x45                ; Carrega W com ASCII da Letra E [expoente base 10].
        MOVWF       L2_C9

    ;----------------------------------Tratamento sinal do expoente-----------------------------------------------
        BTFSC       ST_1_2,6            ; Salta se bit 6 for zero [bit 6 = sinal expoente]
        GOTO        $+4                 ; vá para a quarta instrução abaixo
        MOVLW       0x2D                ; Carrega W com ASCII do sinal negativo do expoente.
        MOVWF       L2_C10
        GOTO        $+3
        MOVLW       0x2B                ; Carrega W com ASCII do sinal positivo do expoente.
        MOVWF       L2_C10

    ;----------------------------------Tratamento valor do expoente-----------------------------------------------

        ;------------------------------------ Dezena do Expoente ---------------------------------------------------
        BTFSC       ST_1_2,7            ; Salta se bit 7 for zero [bit 7 = dezena expoente]
        GOTO        $+4                 ; vá para a quarta instrução abaixo
        MOVLW       0x30                ; Carrega W c/ ASCII Algarismo Zero Dezena expoente.
        MOVWF       L2_C11
        GOTO        $+3
        MOVLW       0x31                ; Carrega W c/ ASCII Algarismo Um Dezena expoente.
        MOVWF       L2_C11

        ;------------------------------------ Unidade do Expoente --------------------------------------------------
        MOVF        ST_1_2,W            ; W ← ST_1_2
        ANDLW       0x0F                ; mascarando conteúdo em W ← W AND k
        CLRF        PCLATH
        CALL        HEXA_ASCII          ; Converte Unidade expoente em ASCII (retorna em W).
        MOVWF       L2_C12              ;L2_C12 ← W

        ;-----------------------------Fim do tratamento do expoente do número medido---------------------------
        BSF         PCLATH,3            ; Ajuste do REG PCLATH
        BSF         PCLATH,4

        ;-------------------------------------------------------------------------------------------------------------
        ;                   Encontrar a Vírgula do número medido                                     (13)
        ;-------------------------------------------------------------------------------------------------------------

        ;----------------------------------Tratamento do Ponto Decimal DP1 a DP5--------------------------------
        BTFSC       ST_7_8,2            ; Testa o bit 2 (DP1) Salta próxima instrução se zero.
```

```
        GOTO    DP1             ; O ponto decimal está em DP1
        BTFSC   ST_9,0          ; Testa o bit 0 (DP2) Salta próxima instrução se zero.
        GOTO    DP2             ; O ponto decimal está em DP2
        BTFSC   ST_9,1          ; Testa o bit 1 (DP3) Salta próxima instrução se zero.
        GOTO    DP3             ; O ponto decimal está em DP3
        BTFSC   ST_9,2          ; Testa o bit 2 (DP4) Salta próxima instrução se zero.
        GOTO    DP4             ; O ponto decimal está em DP4
        BTFSC   ST_9,3          ; Testa o bit 3 (DP5) Salta próxima instrução se zero.
        GOTO    DP5             ; O ponto decimal está em DP5

        PAGESEL Mens_7          ; Todos DP1 a DP5
        CALL    Mens_7          ; São Zeros.
        PAGESEL DELAY_5S
        CALL    DELAY_5S
        PAGESEL DELAY_5S
        CALL    DELAY_5S
        PAGESEL DELAY_5S
        CALL    DELAY_5S
        PAGESEL DELAY_5S
        CALL    DELAY_5S
        PAGESEL DELAY_5S
        CALL    DELAY_5S
        BCF     PCLATH,3        ; AJUSTE DO PCLATH.
        BCF     PCLATH,4
        RETURN

        ;-------------------------------------------------------------------------------------------------------------------
        ;               Tratamento do sinal do número medido                                    (14)
        ;-------------------------------------------------------------------------------------------------------------------
DP1
        ;-----------------------Tratamento sinal do número-------------------------------------------------------------
        BTFSC   ST_7_8,1        ; Salta se bit 1 for zero [bit 1 = sinal do número]
        GOTO    SINAL_POSITIVO_1 ; vá para a quarta instrução abaixo
        MOVLW   0x2D            ; Carrega W com ASCII do sinal negativo do número.
        MOVWF   L2_C1           ; L2_C1 ← W
        GOTO    SEGUE_DP1
SINAL_POSITIVO_1
        MOVLW   0x2B            ; Carrega W com ASCII do sinal positivo do número.
        MOVWF   L2_C1           ; L2_C1 ← W
SEGUE_DP1
        MOVLW   0x30            ; Carrega W c/ ASCII Algarismo Zero antes da virgula.
        MOVWF   L2_C2           ; L2_C2 ← W
        MOVLW   0x2C            ; Carrega W com ASCII da virgula.
        MOVWF   L2_C3           ; L2_C3 ← W

        PAGESEL POS_DP1
        CALL    POS_DP1

        RETURN

        ;-------------------------------------------------------------------------------------------------------------------
        ;               Tratamento do sinal do número medido                                    (15)
        ;-------------------------------------------------------------------------------------------------------------------
DP2
        ;-----------------------Tratamento sinal do número-------------------------------------------------------------
        MOVLW   ' '
```

```
        MOVWF   L2_C1           ; carrega blank na coluna 2 linha 1
        BTFSC   ST_7_8,1        ; Salta se bit 1 for zero [bit 1 = sinal do número]
        GOTO    SINAL_POSITIVO_2 ; vá para a quarta instrução abaixo
        MOVLW   0x2D            ; Carrega W com ASCII do sinal negativo do número.
        MOVWF   L2_C2           ; L2_C2 ← W
        GOTO    ZERO_VIRGULA
SINAL_POSITIVO_2
        MOVLW   0x2B            ; Carrega W com ASCII do sinal positivo do número.
        MOVWF   L2_C2           ; L2_C2 ← W
ZERO_VIRGULA
        MOVLW   0x30            ; Carrega W c/ ASCII Algarismo Zero antes da virgula.
        MOVWF   L2_C3           ; L2_C3 ← W
        MOVLW   0x2C            ; Carrega W com ASCII da virgula.
        MOVWF   L2_C4           ; L2_C4 ← W

        PAGESEL POS_DP2
        CALL    POS_DP2

        RETURN

        ;-------------------------------------------------------------------------------------------------------------------
        ;                       Tratamento do sinal do número medido                                        (16)
        ;-------------------------------------------------------------------------------------------------------------------
DP3
        ;----------------------Tratamento sinal do número-------------------------------------------------------------
        MOVLW   ' '
        MOVWF   L2_C1           ; carrega blank na coluna 2 linha 1
        MOVLW   ' '
        MOVWF   L2_C2           ; carrega blank na coluna 2 linha 2
        BTFSC   ST_7_8,1        ; Salta se bit 1 for zero [bit 1 = sinal do número]
        GOTO    SINAL_POSITIVO_3 ; vá para a quarta instrução abaixo
        MOVLW   0x2D            ; Carrega W com ASCII do sinal negativo do número.
        MOVWF   L2_C3           ; L2_C3 ← W
        GOTO    OVERRANGE_1
SINAL_POSITIVO_3
        MOVLW   0x2B            ; Carrega W com ASCII do sinal positivo do número.
        MOVWF   L2_C3           ; L2_C3 ← W

    ;----------------------Tratamento do Overrange-----------------------------------------------------------------
OVERRANGE_1
        BTFSC   ST_7_8,0        ; Salta se bit 0 for zero [bit 0 = Sinal Overrange]
        GOTO    OVERRANGE_VIRGULA       ; vá para a quarta instrução abaixo
        MOVLW   0x30            ; Carrega W com ASCII do Algarismo zero.
        MOVWF   L2_C4           ; L2_C4 ← W
        GOTO    OVERRANGE_2
OVERRANGE_VIRGULA
        MOVLW   0x31            ; Carrega W c/ ASCII do Algarismo UM de overrange.
        MOVWF   L2_C4           ; L2_C4 ← W
OVERRANGE_2
        MOVLW   0x2C            ; Carrega W com ASCII da virgula.
        MOVWF   L2_C5           ; L2_C5 ← W

        PAGESEL POS_DP3
        CALL    POS_DP3

        RETURN
```

```
        ;-------------------------------------------------------------------------------------------------------------
        ;                       Tratamento do sinal do número medido                                    (17)
        ;-------------------------------------------------------------------------------------------------------------
DP4
        ;----------------------Tratamento sinal do número-------------------------------------------------------------
        MOVLW       ' '
        MOVWF       L2_C1               ; carrega blank na coluna 2 linha 1
        MOVLW       ' '
        MOVWF       L2_C2               ; carrega blank na coluna 2 linha 2
        BTFSC       ST_7_8,1            ; Salta se bit 1 for zero [bit 1 = sinal do número]
        GOTO        GRAFA_MAIS          ; vá para a quarta instrução abaixo
        MOVLW       0x2D                ; Carrega W com ASCII do sinal negativo do número.
        MOVWF       L2_C3               ; L2_C3 ← W
        GOTO        OVERRANGE_3
GRAFA_MAIS
        MOVLW       0x2B                ; Carrega W com ASCII do sinal positivo do número.
        MOVWF       L2_C3               ; L2_C3 ← W

    ;----------------------Tratamento do Overrange-----------------------------------------------------------------
OVERRANGE_3
        BTFSC       ST_7_8,0            ; Salta se bit 0 for zero [bit 0 = Sinal Overrange]
        GOTO        OVERRANGE_UM        ; vá para a quarta instrução abaixo
        MOVLW       0x30                ; Carrega W com ASCII do Algarismo zero.
        MOVWF       L2_C4               ; L2_C4 ← W
        GOTO        CENTENA_NUMERO_4
OVERRANGE_UM
        MOVLW       0x31                ; Carrega W c/ ASCII do Algarismo UM de overrange.
        MOVWF       L2_C4               ; L2_C4 ← W

        ;----------------------Centena do número-----------------------------------------------------------------
CENTENA_NUMERO_4
        MOVF        ST_5_6,W            ; W ← ST_5_6
        MOVWF       WTEMP               ; WTEMP ← W
        SWAPF       WTEMP,W             ; Inverte Nibbles de WTEMP e carregue em W
        ANDLW       0x0F                ; mascarando conteúdo em W ← W AND k
        CLRF        PCLATH
        CALL        HEXA_ASCII          ; Converte centena número em ASCII (conteúdo em W)
        MOVWF       L2_C5               ; L2_C5 ← W

        MOVLW       0x2C                ; Carrega W com ASCII da virgula.
        MOVWF       L2_C6               ; L2_C6 ← W

        PAGESEL     POS_DP4
        CALL        POS_DP4

        RETURN

        ;-------------------------------------------------------------------------------------------------------------
        ;                       Tratamento do sinal do número medido                                    (18)
        ;-------------------------------------------------------------------------------------------------------------
DP5
        ;----------------------Tratamento sinal do número-------------------------------------------------------------
        MOVLW       ' '
        MOVWF       L2_C1               ; carrega blank na coluna 2 linha 1
        MOVLW       ' '
```

```
        MOVWF   L2_C2           ; carrega blank na coluna 2 linha 2
        BTFSC   ST_7_8,1        ; Salta se bit 1 for zero [bit 1 = sinal do número]
        GOTO    SINAL_POSITIVO_5 ; vá para a quarta instrução abaixo
        MOVLW   0x2D            ; Carrega W com ASCII do sinal negativo do número.
        MOVWF   L2_C3           ; L2_C3 ← W
        GOTO    OVERRANGE_4
SINAL_POSITIVO_5
        MOVLW   0x2B            ; Carrega W com ASCII do sinal positivo do número.
        MOVWF   L2_C3           ; L2_C3 ← W

    ;----------------------Tratamento do Overrange-------------------------------------------------------------------
OVERRANGE_4
        BTFSC   ST_7_8,0        ; Salta se bit 0 for zero [bit 0 = Sinal Overrange]
        GOTO    OVERRANGE_4_1   ; vá para a quarta instrução abaixo
        MOVLW   0x30            ; Carrega W com ASCII do Algarismo zero.
        MOVWF   L2_C4           ; L2_C4 ← W
        GOTO    CENTENA_NUMERO_5
OVERRANGE_4_1
        MOVLW   0x31            ; Carrega W c/ ASCII do Algarismo UM de overrange.
        MOVWF   L2_C4           ; L2_C4 ← W
        ;----------------------Centena do número-------------------------------------------------------------------
CENTENA_NUMERO_5
        MOVF    ST_5_6,W        ; W ← ST_5_6
        MOVWF   WTEMP           ; WTEMP ← W
        SWAPF   WTEMP,W         ; Inverte Nibbles de WTEMP e carregue em W
        ANDLW   0x0F            ; mascarando conteúdo em W ← W AND k
        CLRF    PCLATH
        CALL    HEXA_ASCII      ; Converte centena número em ASCII (conteúdo em W)
        MOVWF   L2_C5           ; L2_C5 ← W

        ;----------------------Dezena do número--------------------------------------------------------------------
        MOVF    ST_3_4,W        ; W ← ST_3_4
        ANDLW   0x0F            ; mascarando conteúdo em W ← W AND k
        CALL    HEXA_ASCII      ; Converte dezena  número em ASCII (conteúdo em W)
        MOVWF   L2_C6           ; L2_C6 ← W

        MOVLW   0x2C            ; Carrega W com ASCII da virgula.
        MOVWF   L2_C7           ; L2_C7 ← W

        PAGESEL POS_DP5
        CALL    POS_DP5

        RETURN
```

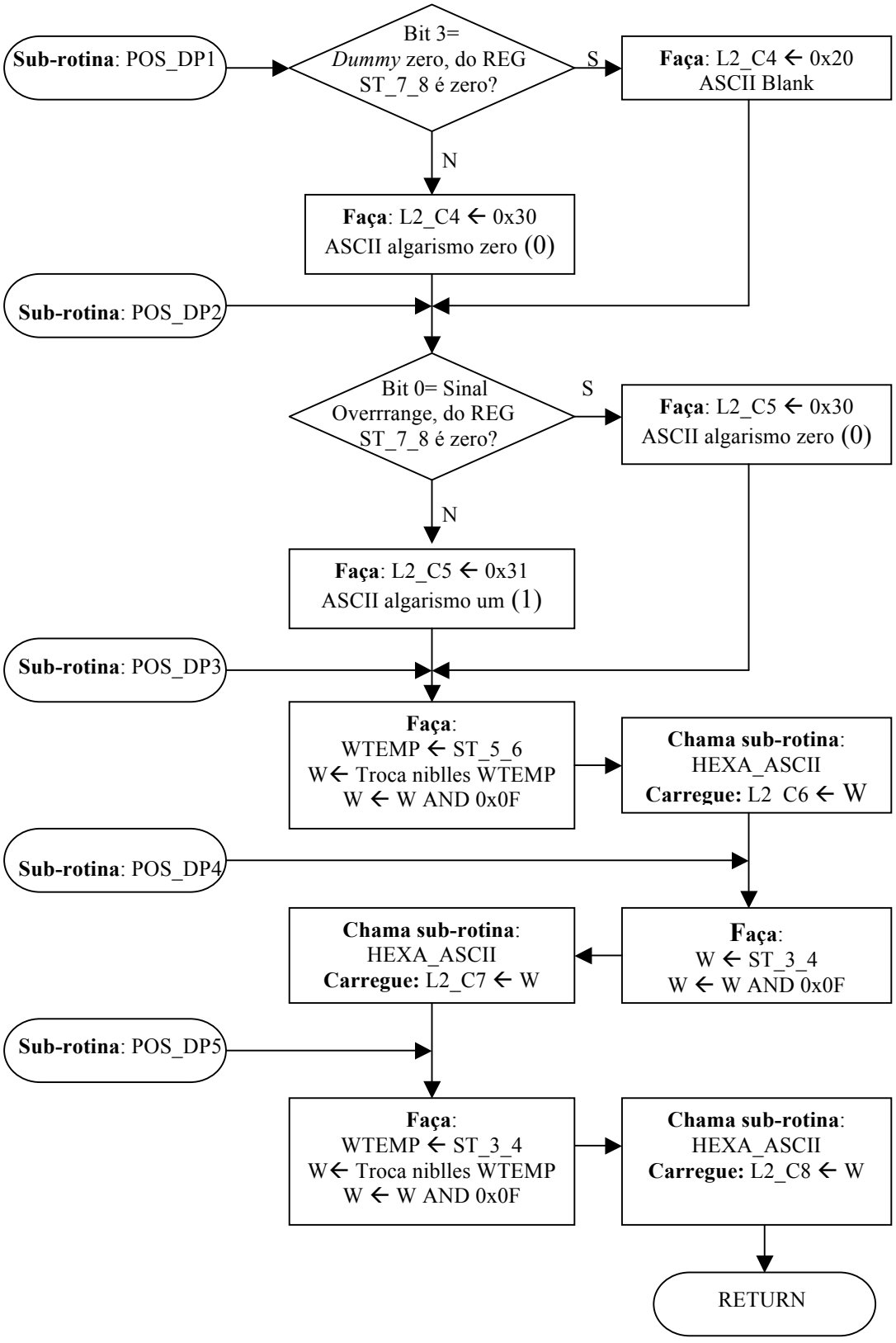


Figura A20– Fluxograma da Sub-rotina: Converte ST_ASCII – POS_DP1 a POS_DP5.

```
;-------------------------------------------------------------------------------------------------------------------------
;   Esta sub-rotina é parte do tratamento do Ponto Decimal DP1 a DP5 do Eletrômetro
;-------------------------------------------------------------------------------------------------------------------------
POS_DP1
    ;-----------------------------------Tratamento Dummy zero-------------------------------------------------------------
        BTFSC       ST_7_8,3            ; Salta se bit 3 for zero [bit 3 = Dummy zero]
        GOTO        DUMMY               ; vá para a quarta instrução abaixo
        MOVLW       ' '                 ; Carrega W com ASCII do sinal ESPACO.
        MOVWF       L2_C4
        GOTO        POS_DP2
DUMMY
        MOVLW       0x30                ; Carrega W c/ ASCII do Algarismo zero p/ DUMMY.
        MOVWF       L2_C4
POS_DP2
    ;-----------------------------------Tratamento do Overrange-----------------------------------------------------------
        BTFSC       ST_7_8,0            ; Salta se bit 0 for zero [bit 0 = Sinal Overrange]
        GOTO        OVERRANGE           ; vá para a quarta instrução abaixo
        MOVLW       0x30                ; Carrega W com ASCII do Algarismo zero.
        MOVWF       L2_C5
        GOTO        POS_DP3
OVERRANGE
        MOVLW       0x31                ; Carrega W c/ ASCII do Algarismo UM de overrange.
        MOVWF       L2_C5

POS_DP3
    ;-----------------------------------Tratamento dos 3 dígitos----------------------------------------------------------
        ;-----------------------------------Centena do número-------------------------------------------------------------
        MOVF        ST_5_6,W            ; W ← ST_5_6
        MOVWF       WTEMP               ; WTEMP ← W
        SWAPF       WTEMP,W             ; Inverte Nibbles de WTEMP e carregue em W
        ANDLW       0x0F                ; mascarando conteúdo em W ← W AND k
        CLRF        PCLATH              ; PCLATH ← 0x00
        CALL        HEXA_ASCII          ; Converte centena número em ASCII (conteúdo em W)
        MOVWF       L2_C6               ; L2_C6 ← W
                                        ; alterou da Página 2 para a página 0
POS_DP4
        ;-----------------------------------Dezena do número--------------------------------------------------------------
        MOVF        ST_3_4,W            ; W ← ST_3_4
        ANDLW       0x0F                ; mascarando conteúdo em W ← W AND k
        CLRF        PCLATH              ; PCLATH ← 0x00
        CALL        HEXA_ASCII          ; Converte dezena número em ASCII (conteúdo em W)
        MOVWF       L2_C7               ; L2_C7 ← W
POS_DP5
        ;-----------------------------------Unidade do número-------------------------------------------------------------
        MOVF        ST_3_4,W            ; W ← ST_3_4
        MOVWF       WTEMP               ; WTEMP ← W
        SWAPF       WTEMP,W             ; Inverte Nibbles de WTEMP e carregue em W
        ANDLW       0x0F                ; mascarando conteúdo em W ← W AND k
        CLRF        PCLATH              ; PCLATH ← 0x00
        CALL        HEXA_ASCII          ; Converte unidade número em ASCII (conteúdo em W)
        MOVWF       L2_C8               ; L2_C8 ← W

        RETURN                          ; Retorna ao fluxo normal do Programa.
;-------------------------------------------------------------------------------------------------------------------------
```

b4) Sub-rotina HIGH_ENDERECO_LOW_ASCII

Esta sub-rotina converte a parte alta e baixa do endereço para ASCII, conforme mostra a Fig. A21.

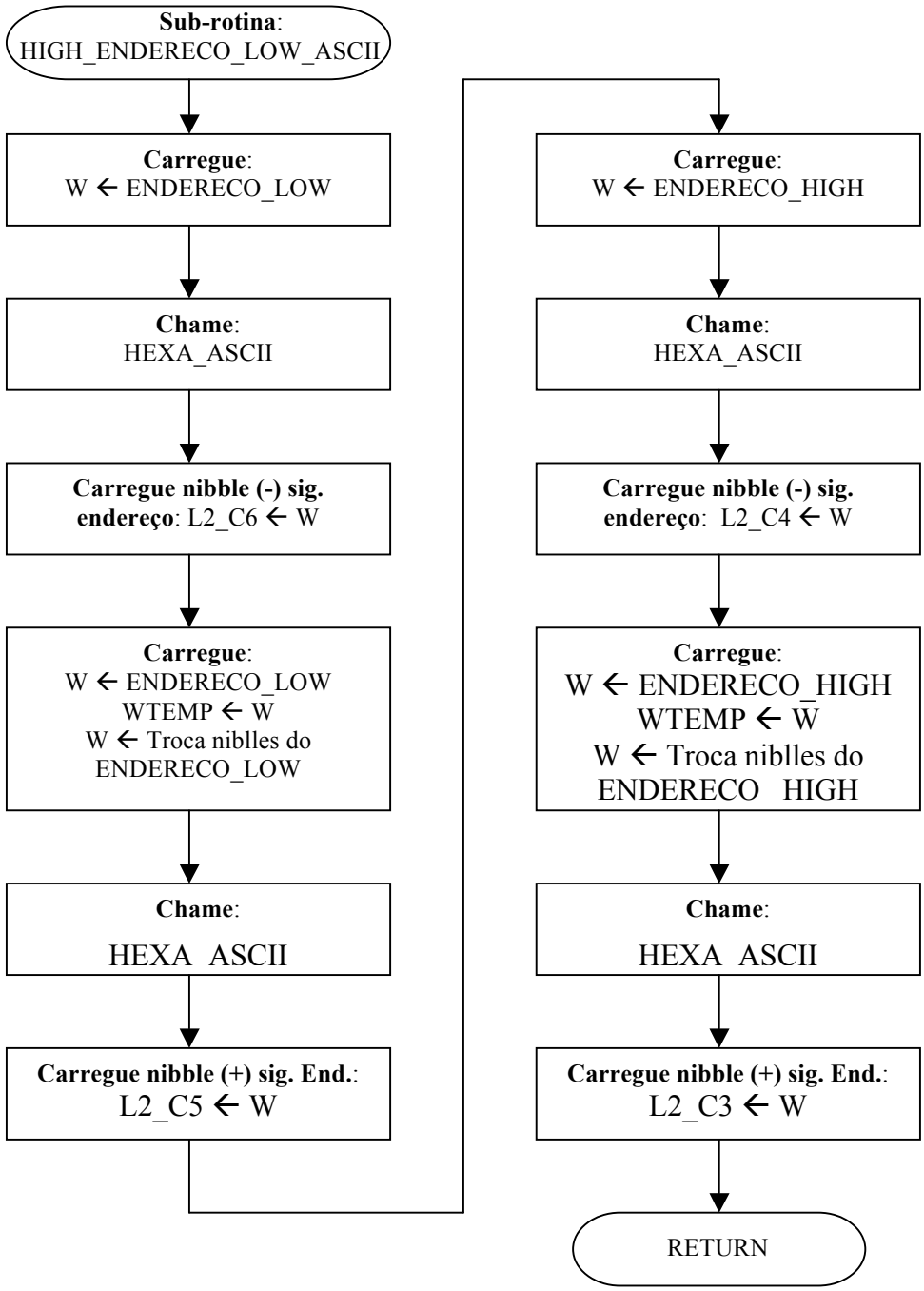


Figura A21– Fluxograma da Sub-rotina: HIGH_ENDERECO_LOW_ASCII.

```
;*******************************************************************************
;* Esta sub-rotina converte o ENDERECO_HIGH e o ENDERECO_LOW p/ ASCII → p/ envio ao LCD      *
;*******************************************************************************
HIGH_ENDERECO_LOW_ASCII
        MOVF        ENDERECO_LOW,W          ; W ← ENDERECO_LOW
        CLRF        PCLATH                  ; PCLATH ← 0x00
        CALL        HEXA_ASCII              ; Chama sub-rotina que Converte nibble menos
                                            ; sig. para ASCII.
        MOVWF       L2_C6                   ; L2_C6 ← W, carrega nibble menos sig do
                                            ; endereço em L2_C6

        MOVF        ENDERECO_LOW,W          ; W ← ENDERECO_LOW
        MOVWF       WTEMP                   ; WTEMP ← W
        SWAPF       WTEMP,W                 ; Troca nibbles menos e mais significativo de
        CALL        HEXA_ASCII              ; Chama sub-rotina que Converte nibble menos
                                            ; sig. para ASCII.
        MOVWF       L2_C5                   ; L2_C5 ← W, carrega nibble mais sig do
                                            ; endereço em L2_C5

        MOVF        ENDERECO_HIGH,W         ; W ← ENDERECO_HIGH
        CALL        HEXA_ASCII              ; Chama sub-rotina que Converte nibble menos
                                            ; sig. para ASCII.
        MOVWF       L2_C4                   ; L2_C4 ← W, carrega nibble menos sig do
                                            ; endereço em L2_C4

        MOVF        ENDERECO_HIGH,W         ; W ← ENDERECO_HIGH
        MOVWF       WTEMP                   ; WTEMP ← W
        SWAPF       WTEMP,W                 ; Troca nibbles menos e mais significativo de
        CALL        HEXA_ASCII              ; Chama sub-rotina que Converte nibble menos
                                            ; sig. para ASCII.
        MOVWF       L2_C3                   ; L2_C3 ← W, carrega nibble mais sig do
                                            ;  endereço em L2_C3
        RETURN                              ; Retorna ao programa que a chamou.
;*******************************************************************************
```

b5) Sub-rotina ACERTA_A1_A0_B0_ASCII

A sub-rotina mostrada na Fig. A22, converte o conteúdo do registrador CONTROLE_I2C para ASCII, onde seus bits menos significativos são B0 A1 A0 R/W.

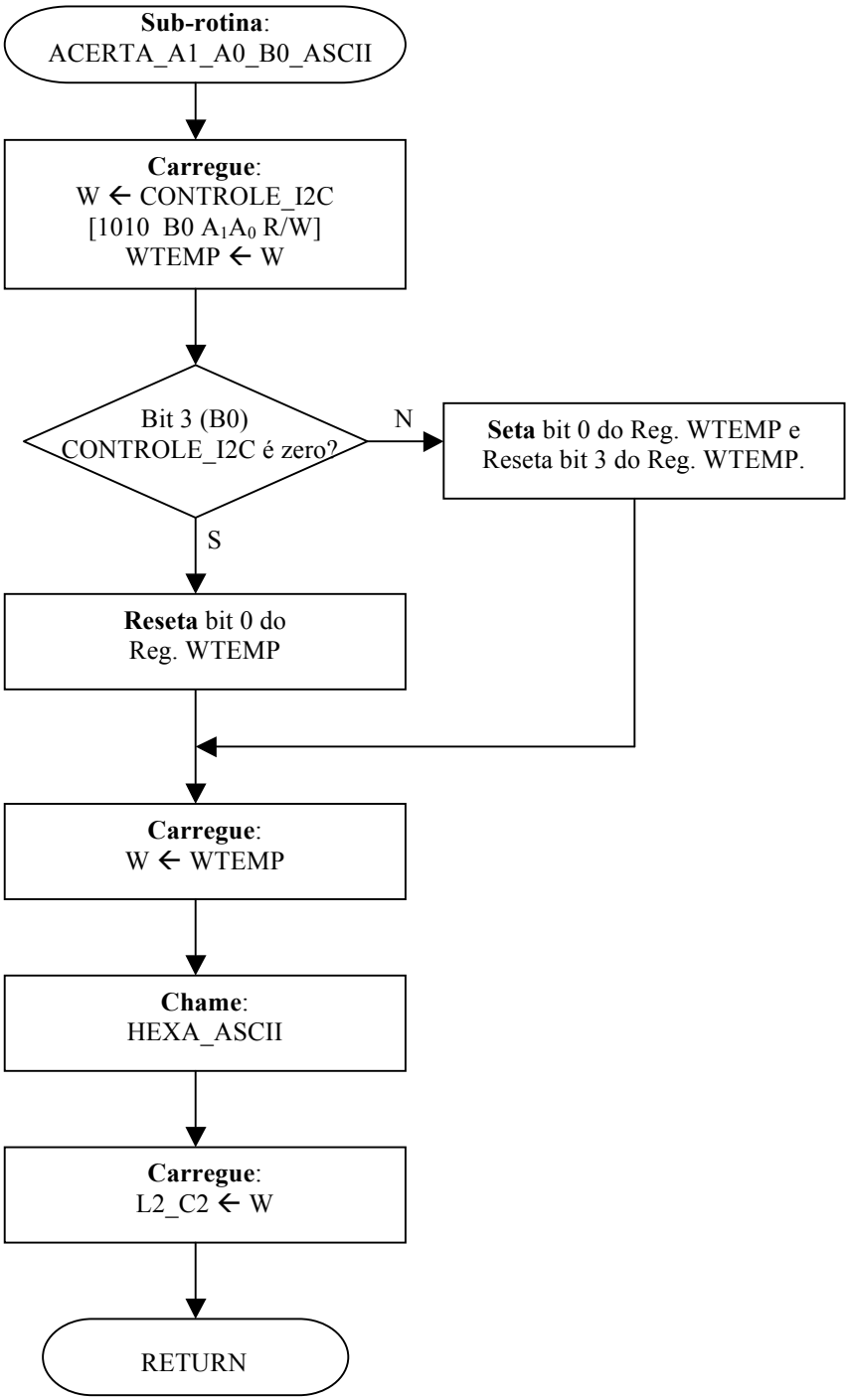


Figura A22– Fluxograma da Sub-rotina: ACERTA_A1_A0_B0_ASCII.

```
;*******************************************************************************
;* ESTA SUBROTINA ACERTA E CONVERTE A1_A0_B0 PARA ASCII → PARA ENVIO AO LCD     *
;*******************************************************************************
ACERTA_A1_A0_B0_ASCII                       ; CONTROLE_I2C ← 1010  B0 A1A0 R/W
                                            ; Qdo B0=0 aponta para o primeiro bloco de
                                            ; 64kbytes da 24LC1025, e qdo B0=1 aponta
                                            ; para o último bloco de 64kbytes deste
                                            ; dispositivo.

        MOVF        CONTROLE_I2C,W          ; W ← CONTROLE_I2C
        MOVWF       WTEMP                   ; WTEMP ← W
        BTFSC       CONTROLE_I2C,3          ; Verifica se bit 3 do reg CONTROLE_I2C é 1
        GOTO        AJUSTA_A1_A0_B0
        BCF         WTEMP,0                 ; Reseta bit 0 do byte WTEMP
                                            ; [que ajuste CONTROLE_I2C]
        GOTO        SEGUE
AJUSTA_A1_A0_B0
        BSF         WTEMP,0                 ; Seta bit 0 do byte WTEMP
                                            ; [que é ajuste CONTROLE_I2C]
        BCF         WTEMP,3
SEGUE
        MOVF        WTEMP,W                 ; W←WTEMP
        CLRF        PCLATH                  ; PCLATH ← 0x00
        CALL        HEXA_ASCII              ; Chama sub-rotina que Converte nibble menos
                                            ; sig. para ASCII.
        MOVWF       L2_C2                   ; L2_C2 ← W, carrega nibble mais sig. do
                                            ; endereço em L2_C2
        RETURN                              ; Retorna ao programa que a chamou.
;*******************************************************************************
```

b6) Sub-rotinas na forma de tabelas.

Na sequência são mostradas as sub-rotinas em formato de tabelas.

(1) **Sub-rotina HEXA_ASCII**

Esta sub-rotina, mostrada na Fig. A23, converte um número hexadecimal em seu correspondente ASCII.

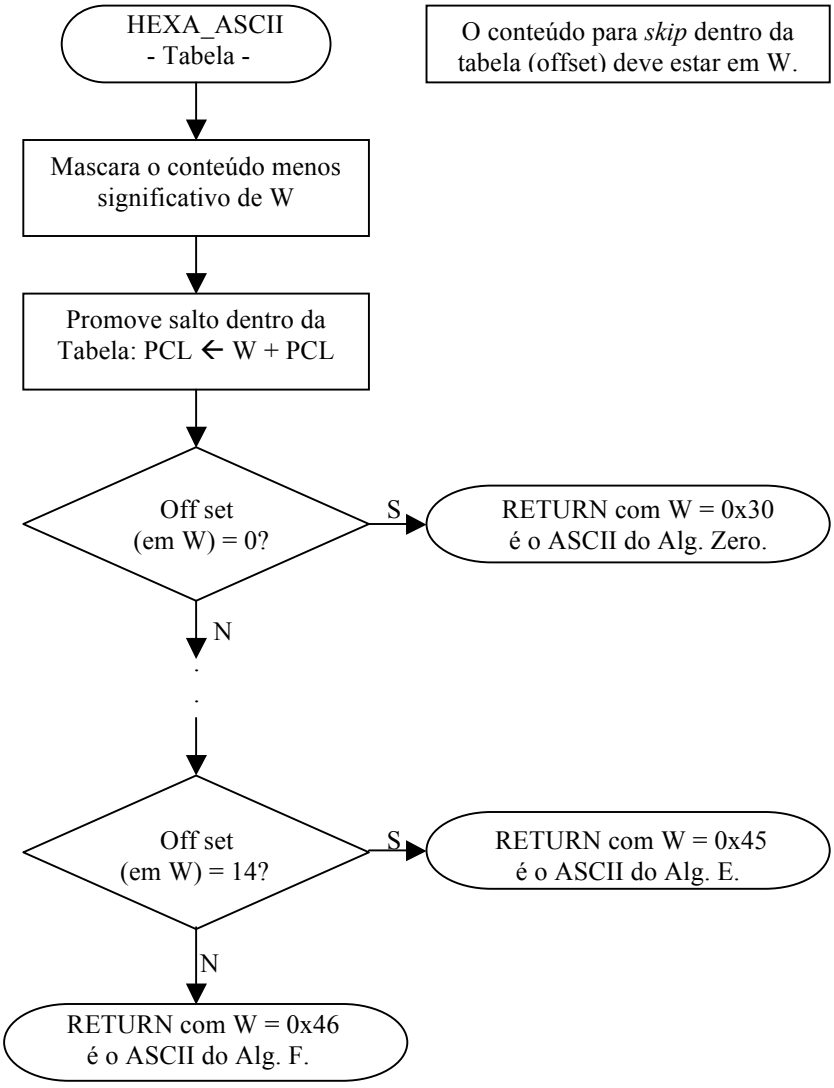


Figura A23– Fluxograma da Sub-rotina: HEXA_ASCII. Tabela.

```
;*******************************************************************************
;* Esta sub-rotina converte o nibble menos significativo ao seu equivalente em ASCII *
;*******************************************************************************
HEXA_ASCII
        ANDLW       B'00001111'     ; W ← W AND 0Fh. Mascarando o byte em W, optado pelo
                                    ; byte menos significativo.
        ADDWF       PCL,F           ; PCL ← W + PCL
                                    ; Adiciona ao PC o valor para saltar e encontrar o conteúdo
                                    ; tabelado da conversão desejada.
        RETLW       B'00110000'             ; 0h = ASCII 30h
        RETLW       B'00110001'             ; 1h = ASCII 31h
        RETLW       B'00110010'             ; 2h = ASCII 32h
        RETLW       B'00110011'             ; 3h = ASCII 33h
        RETLW       B'00110100'             ; 4h = ASCII 34h
        RETLW       B'00110101'             ; 5h = ASCII 35h
        RETLW       B'00110110'             ; 6h = ASCII 36h
        RETLW       B'00110111'             ; 7h = ASCII 37h
        RETLW       B'00111000'             ; 8h = ASCII 38h
        RETLW       B'00111001'             ; 9h = ASCII 39h
        RETLW       B'01000001'             ; Ah = ASCII 41h
        RETLW       B'01000010'             ; Bh = ASCII 42h
        RETLW       B'01000011'             ; Ch = ASCII 43h
        RETLW       B'01000100'             ; Dh = ASCII 44h
        RETLW       B'01000101'             ; Eh = ASCII 45h
        RETLW       B'01000110'             ; Fh  =ASCII 46h
;*******************************************************************************
```

(2) **Sub-rotina CONVERTE_X_BCD**

Esta sub-rotina converte o número binário contido no registrador X, cujo conteúdo assume um valor inteiro entre 0 a 40, em seu correspondente BCD de (+)20,0 a (-)20,0. Assim, toda vez que a informação contida em X for enviada por uma das interfaces máquina-homem (display LCD ou TX RS-232 ao computador pessoal), ela deve, também, ser convertida para seu correspondente no código ASCII.

```
;*******************************************************************************
; * Acerta X da Posição +20,0 a -20,0                                           *
;*******************************************************************************
CONVERTE_X_BCD                  ; Fornece o valor de X em BCD sem sinal. Antes de rodar esta
                                ; sub-rotina, deve-se carregar W com o valor de X.
        ANDLW       B'00111111' ; W ← W AND 3Fh. Mascarando o byte em W.
        ADDWF       PCL,F       ; PCL ← W + PCL
                                ; Adiciona ao PC o valor para saltar e encontrar o conteúdo
                                ; tabelado da conversão desejada.

            ;  128 64 32 16 8 4 2 1
        RETLW       B'00100000'         ; D'20' → X= 0cm
        RETLW       B'00011001'         ; D'19' → X= 1cm
        RETLW       B'00011000'         ; D'18' → X= 2cm
```

```
        RETLW       B'00010111'         ; D'17' → X= 3cm
        RETLW       B'00010110'         ; D'16' → X= 4cm
        RETLW       B'00010101'         ; D'15' → X= 5cm
        RETLW       B'00010100'         ; D'14' → X= 6cm
        RETLW       B'00010011'         ; D'13' → X= 7cm
        RETLW       B'00010010'         ; D'12' → X= 8cm
        RETLW       B'00010001'         ; D'11' → X= 9cm
        RETLW       B'00010000'         ; D'10' → X=10cm
        RETLW       B'00001001'         ; D'9'  → X=11cm
        RETLW       B'00001000'         ; D'8'  → X=12cm
        RETLW       B'00000111'         ; D'7'  → X=13cm
        RETLW       B'00000110'         ; D'6'  → X=14cm
        RETLW       B'00000101'         ; D'5'  → X=15cm
        RETLW       B'00000100'         ; D'4'  → X=16cm
        RETLW       B'00000011'         ; D'3'  → X=17cm
        RETLW       B'00000010'         ; D'2'  → X=18cm
        RETLW       B'00000001'         ; D'1'  → X=19cm
        RETLW       B'00000000'         ; D'0'  → X=20cm
        RETLW       B'00000001'         ; D'-1' → X=21cm
        RETLW       B'00000010'         ; D'-2' → X=22cm
        RETLW       B'00000011'         ; D'-3' → X=23cm
        RETLW       B'00000100'         ; D'-4' → X=24cm
        RETLW       B'00000101'         ; D'-5' → X=25cm
        RETLW       B'00000110'         ; D'-6' → X=26cm
        RETLW       B'00000111'         ; D'-7' → X=27cm
        RETLW       B'00001000'         ; D'-8' → X=28cm
        RETLW       B'00001001'         ; D'-9' → X=29cm
        RETLW       B'00010000'         ; D'-10'→ X=30cm
        RETLW       B'00010001'         ; D'-11'→ X=31cm
        RETLW       B'00010010'         ; D'-12'→ X=32cm
        RETLW       B'00010011'         ; D'-13'→ X=33cm
        RETLW       B'00010100'         ; D'-14'→ X=34cm
        RETLW       B'00010101'         ; D'-15'→ X=35cm
        RETLW       B'00010110'         ; D'-16'→ X=36cm
        RETLW       B'00010111'         ; D'-17'→ X=37cm
        RETLW       B'00011000'         ; D'-18'→ X=38cm
        RETLW       B'00011001'         ; D'-19'→ X=39cm
        RETLW       B'00100000'         ; D'-20'→ X=40cm
;*********************************************************************************
```

(3) **Sub-rotina Y_2**

Sem dúvida a parte principal do projeto é o posicionamento da fonte radioativa e do detector. Para este posicionamento foram necessárias 91 tabelas, nomeadas de Y_2 até Y_92. Para exemplificar foi mostrada a tabela Y_2, pois se todas as tabelas fossem mostradas elas ocupariam muito espaço, sem reputar proporcional benefício. As mesas X e Y movimentam a passos de 1 cm. A mesa Y inicia-se na posição física a 2,0 cm do eixo X, deslocando-se até a posição Y=92,0 cm. A mesa X varia da posição –20,0 passa pelo zero e avança até a posição

+20,0 cm. A Fig. A 24 mostra os limites destes deslocamentos. O cálculo do número de passos necessários para comandar os motores de passo (direcionar o feixe radioativo direto ao volume sensível do detector), pode ser determinado, por exemplo, tomando Y=2,0 cm e X = 20,0 cm, encontra-se igual a $\beta = 5{,}71^{o}$, o que implica em $\alpha = 84{,}29^{o}$ ($\alpha + \beta = 90^{o}$). Se cada passo do motor de passo perfaz um ângulo de $1{,}8^{o}$, então deve ser dado aproximadamente 46 passos quando as mesas estiverem nas posições (Y=2 e X=20). Percebe-se a ocorrência de certo erro a cada posicionamento, mas estes serão compensados pela dimensão geométrica da janela do detector, a qual, ainda assim, atingirá satisfatoriamente o centro geométrico do detector.

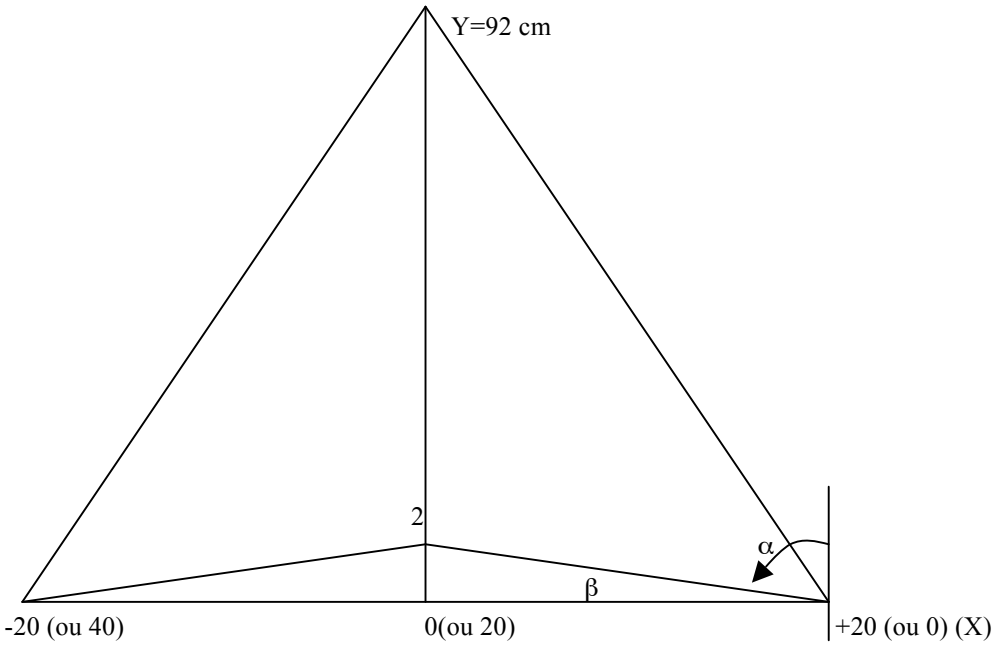


Figura A24 – Deslocamento das mesas X e Y.

Tabela 38: Quantidade de passos a serem dados – ajuste fonte-detector, Y=2cm

	X=20	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**X=0**
pas sos	46	0	0	0	0	0	0	0	1	0	1	0	1	1	2	2	2	4	7	10	0

```
;*******************************************************************************
;* Tabelas para ajuste da cabeça onde reside a Fonte Radioativa                *
;*******************************************************************************
Y_2                                     ; W já vem com o valor de X.
        ANDLW       B'00111111'         ; W ← W AND 0Fh. Mascarando o byte em W, optado
                                        ; pelo byte menos significativo.
        ADDWF       PCL,F               ; PCL ← W + PCL
                                        ; Adiciona ao PC o valor para saltar e encontrar o conteúdo
                                        ; tabelado da conversão desejada.

                ;  128 64 32 16 8 4 2 1
        RETLW       B'00101110'             ; D'20' → X= 0cm,46 passos= 00101110 anti-horário
        RETLW       B'00000000'             ; D'19' → X= 1cm, 0 passo   demais é no sentido
        RETLW       B'00000000'             ; D'18' → X= 2cm, 0 passo    horário.
        RETLW       B'00000000'             ; D'17' → X= 3cm, 0 passo
        RETLW       B'00000000'             ; D'16' → X= 4cm, 0 passo
        RETLW       B'00000000'             ; D'15' → X= 5cm, 0 passo
        RETLW       B'00000000'             ; D'14' → X= 6cm, 0 passo
        RETLW       B'00000000'             ; D'13' → X= 7cm, 0 passo
        RETLW       B'00000001'             ; D'12' → X= 8cm, 1 passo
        RETLW       B'00000000'             ; D'11' → X= 9cm, 0 passo
        RETLW       B'00000001'             ; D'10' → X=10cm, 1 passo
        RETLW       B'00000000'             ; D'9'  → X=11cm, 0 passo
        RETLW       B'00000001'             ; D'8'  → X=12cm, 1 passo
        RETLW       B'00000001'             ; D'7'  → X=13cm, 1 passo
        RETLW       B'00000010'             ; D'6'  → X=14cm, 2 passos
        RETLW       B'00000010'             ; D'5'  → X=15cm, 2 passos
        RETLW       B'00000010'             ; D'4'  → X=16cm, 2 passos
        RETLW       B'00000100'             ; D'3'  → X=17cm, 4 passos
        RETLW       B'00000111'             ; D'2'  → X=18cm, 7 passos
        RETLW       B'00001010'             ; D'1'  → X=19cm, 10 passos
        RETLW       B'00000000'             ; D'0'  → X=20cm, 0 passo
        RETLW       B'00000000'             ; D'-1' → X=21cm, 0 passo ---------
        RETLW       B'00001010'             ; D'-2' → X=22cm, 10 passos
        RETLW       B'00000111'             ; D'-3' → X=23cm, 7 passos
        RETLW       B'00000100'             ; D'-4' → X=24cm, 4 passos
        RETLW       B'00000010'             ; D'-5' → X=25cm, 2 passos
        RETLW       B'00000010'             ; D'-6' → X=26cm, 2 passos
        RETLW       B'00000010'             ; D'-7' → X=27cm, 2 passos
        RETLW       B'00000001'             ; D'-8' → X=28cm, 1 passo
        RETLW       B'00000001'             ; D'-9' → X=29cm, 1 passo
        RETLW       B'00000000'             ; D'-10'→ X=30cm, 0 passo
        RETLW       B'00000001'             ; D'-11'→ X=31cm, 1 passo
        RETLW       B'00000000'             ; D'-12'→ X=32cm, 0 passo
        RETLW       B'00000001'             ; D'-13'→ X=33cm, 1 passo
        RETLW       B'00000000'             ; D'-14'→ X=34cm, 0 passo
        RETLW       B'00000000'             ; D'-15'→ X=35cm, 0 passo
        RETLW       B'00000000'             ; D'-16'→ X=36cm, 0 passo
        RETLW       B'00000000'             ; D'-17'→ X=37cm, 0 passo
        RETLW       B'00000000'             ; D'-18'→ X=38cm, 0 passo
        RETLW       B'00000000'             ; D'-19'→ X=39cm, 0 passo
        RETLW       B'00000000'             ; D'-20'→ X=40cm, 0 passo
;*******************************************************************************
```

b7) Sub-rotina BIN_BCD: converte um número binário contido em W em BCD.

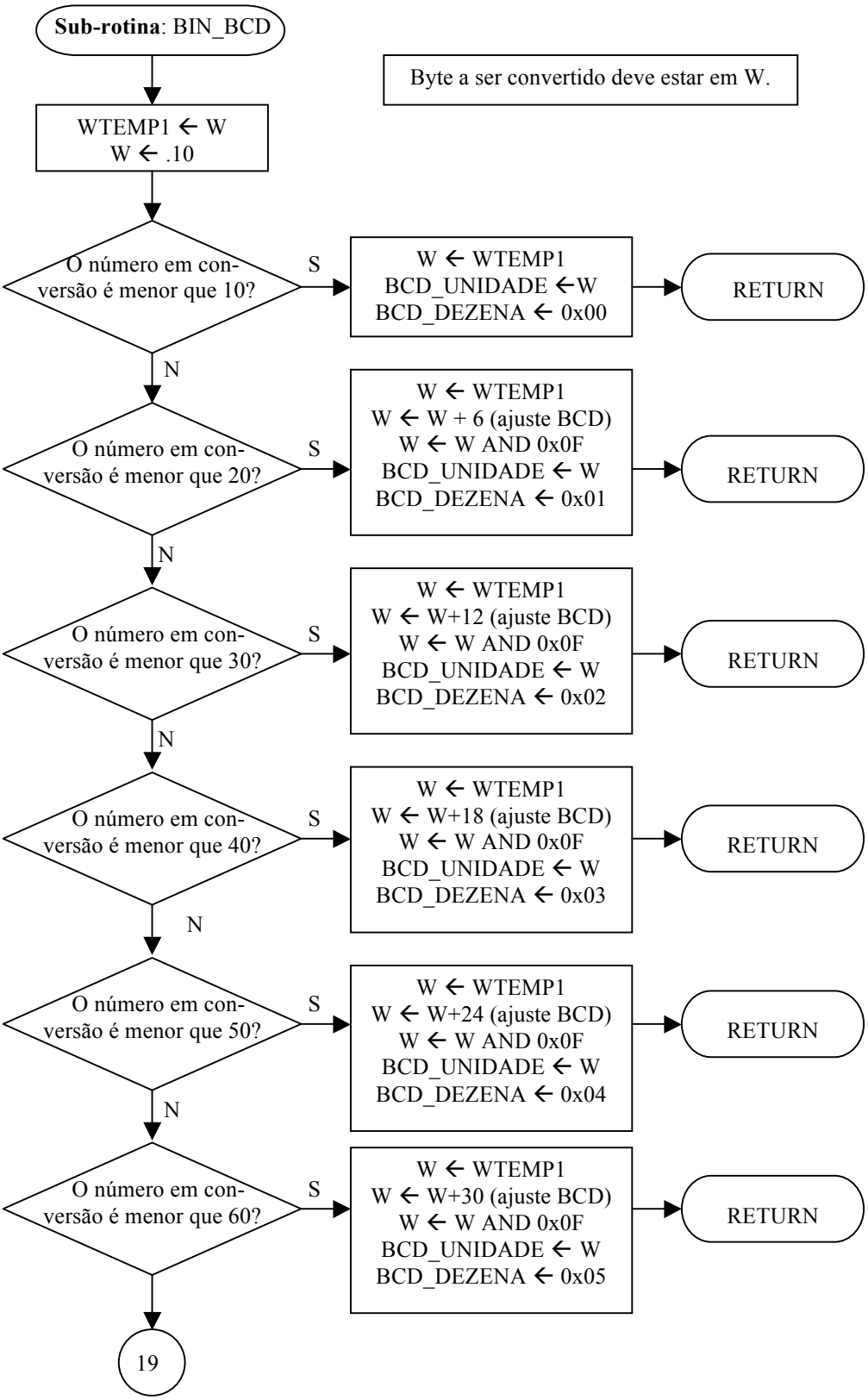


Figura A25– Fluxograma da Sub-rotina: BIN_BCD - Inicial.

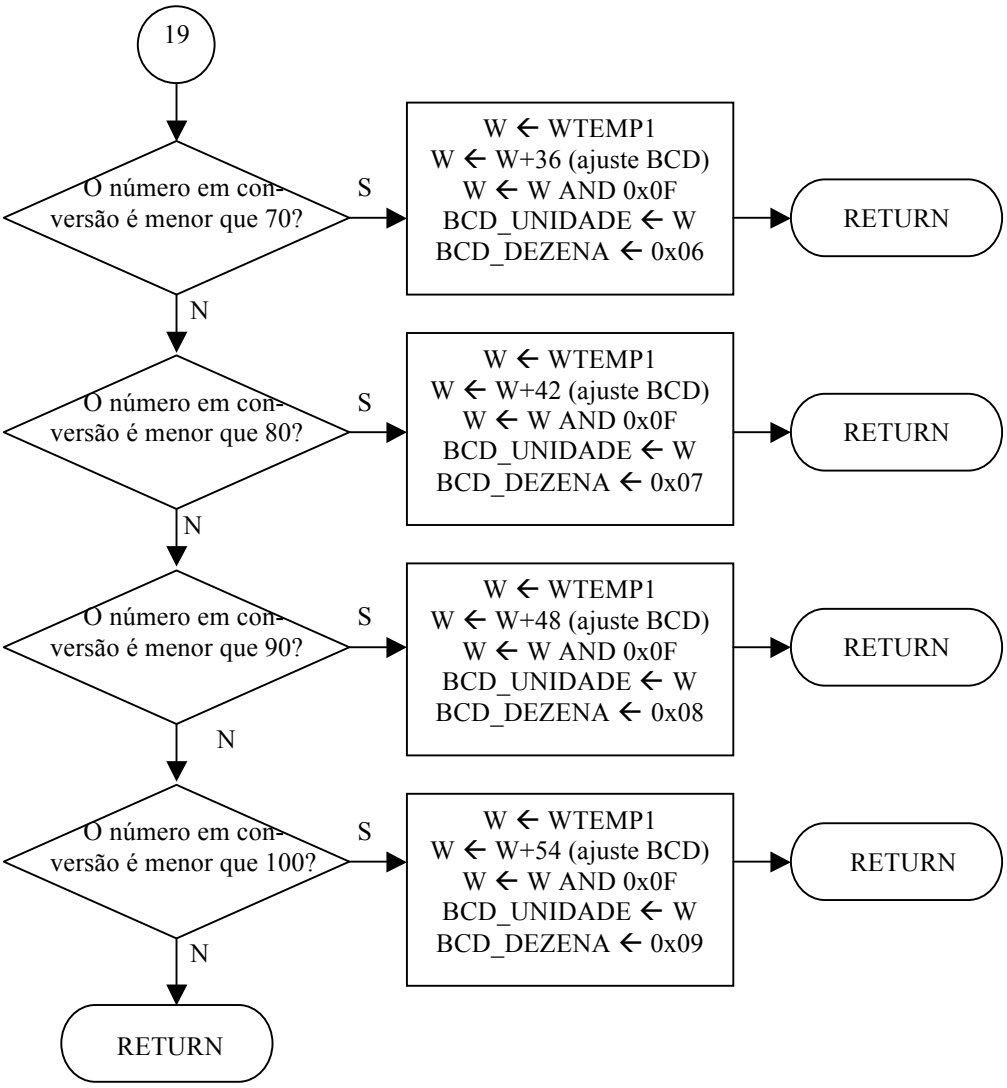


Figura A26– Fluxograma da Sub-rotina: BIN_BCD - Final.

As Fig. A25 e Fig. A26 mostram os fluxos de atividades que devem ser seguidas para implementar a conversão de um número binário em BCD. O produto desta implementação está mostrado na sequência.

```
;*******************************************************************************
;* Esta sub-rotina converte um número binário em um número BCD                 *
;*******************************************************************************
; O número a ser convertido deve estar residente em W e o resultado será armazenado em BCD_DEZENA
; e BCD_UNIDADE, respectivamente.

BIN_BCD
        MOVWF       WTEMP1              ; WTEMP1 ← W
        MOVLW       .10                 ; W←10
        SUBWF       WTEMP1,W            ; W ← WTEMP1-10
        BTFSS       STATUS,C            ; WTEMP1 é menor que 10?
                                        ; Qdo WTEMP1 < 10 implica que FLAG c=0
        GOTO        $+2                 ; Se sim (WTEMP1<10), salta 2 linhas à frente,
        GOTO        $+7                 ; Se não [WTEMP1>=10, ou c=1] salta 7 linhas a frente.
                                        ; qdo for menor que 10, já é o BCD, então:
                                        ; BCD_UNIDADE ← WTEMP1
        MOVF        WTEMP1,W            ; W ← WTEMP1
        MOVWF       BCD_UNIDADE         ; BCD_UNIDADE ← W
        MOVF        BCD_UNIDADE,W       ; W← BCD_UNIDADE

        MOVLW       0x00
        MOVWF       BCD_DEZENA          ; BCD_DEZENA ← 0x00
        RETURN

        MOVLW       .20                 ; W←20
        SUBWF       WTEMP1,W            ; W ← WTEMP1-20
        BTFSS       STATUS,C            ; WTEMP1 é menor que 20?
        GOTO        ENTRE_10e19         ; Se sim, vá para ENTRE_10e19
                                        ; Se não, dá sequência saltando este GOTO.

        MOVLW       .30                 ; W←30
        SUBWF       WTEMP1,W            ; W ← WTEMP1-30
        BTFSS       STATUS,C            ; WTEMP1 é menor que 30?
        GOTO        ENTRE_20e29         ; Se sim, vá para ENTRE_20e29
                                        ; Se não, dá sequência saltando este GOTO.

        MOVLW       .40                 ; W←20
        SUBWF       WTEMP1,W            ; W ← WTEMP1-40
        BTFSS       STATUS,C            ; WTEMP1 é menor que 40?
        GOTO        ENTRE_30e39         ; Se sim, vá para ENTRE_30e39
                                        ; Se não, dá sequência saltando este GOTO.

        MOVLW       .50                 ; W←50
        SUBWF       WTEMP1,W            ; W ← WTEMP1-50
        BTFSS       STATUS,C            ; WTEMP1 é menor que 40?
        GOTO        ENTRE_40e49         ; Se sim, vá para ENTRE_40e49
                                        ; Se não, dá sequência saltando este GOTO.

        MOVLW       .60                 ; W←60
        SUBWF       WTEMP1,W            ; W ← WTEMP1-60
        BTFSS       STATUS,C            ; WTEMP1 é menor que 60?
        GOTO        ENTRE_50e59         ; Se sim, vá para ENTRE_50e59
                                        ; Se não, dá sequência saltando este GOTO.
```

```
        MOVLW   .70             ; W←70
        SUBWF   WTEMP1,W        ; W ← WTEMP1-70
        BTFSS   STATUS,C        ; WTEMP1 é menor que 70?
        GOTO    ENTRE_60e69     ; Se sim, vá para ENTRE_60e69
                                ; Se não, dá sequência saltando este GOTO.

        MOVLW   .80             ; W←80
        SUBWF   WTEMP1,W        ; W ← WTEMP1-80
        BTFSS   STATUS,C        ; WTEMP1 é menor que 80?
        GOTO    ENTRE_70e79     ; Se sim, vá para ENTRE_70e79
                                ; Se não, dá sequência saltando este GOTO.

        MOVLW   .90             ; W←90
        SUBWF   WTEMP1,W        ; W ← WTEMP1-90
        BTFSS   STATUS,C        ; WTEMP1 é menor que 90?
        GOTO    ENTRE_80e89     ; Se sim, vá para ENTRE_80e89
                                ; Se não, dá sequência saltando este GOTO.

        MOVLW   .100            ; W←100
        SUBWF   WTEMP1,W        ; W ← WTEMP1-100
        BTFSS   STATUS,C        ; WTEMP1 é menor que 100?
        GOTO    ENTRE_90e99     ; Se sim, vá para ENTRE_90e99
                                ; Se não, dá sequência saltando este GOTO.
        RETURN
ENTRE_10e19
        MOVF    WTEMP1,W        ; W ← WTEMP1
        ADDLW   .6              ; W ← W+6
                        ; D10 = 0000 1010 + 0000 0110 = 0001 0000 → já é BCD
                        ; D11 = 0000 1011 + 0000 0110 = 0001 0001 → já é BCD
                        ; D12 = 0000 1100 + 0000 0110 = 0001 0010 → já é BCD
                        ; D13 = 0000 1101 + 0000 0110 = 0001 0011 → já é BCD
                        ; D14 = 0000 1110 + 0000 0110 = 0001 0100 → já é BCD
                        ; D15 = 0000 1111 + 0000 0110 = 0001 0101 → já é BCD
                        ; D16 = 0001 0000 + 0000 0110 = 0001 0110 → já é BCD
                        ; D17 = 0001 0001 + 0000 0110 = 0001 0111 → já é BCD
                        ; D18 = 0001 0010 + 0000 0110 = 0001 1000 → já é BCD
                        ; D19 = 0001 0011 + 0000 0110 = 0001 1001 → já é BCD

        ANDLW   0x0F            ; W ← W AND 0x0F, parte baixa byte convertido BCD
        MOVWF   BCD_UNIDADE     ; BCD_UNIDADE ← W
        MOVLW   0x01
        MOVWF   BCD_DEZENA      ; BCD_DEZENA ← 0x01
        RETURN

ENTRE_20e29
        MOVF    WTEMP1,W        ; W ← WTEMP1
        ADDLW   .12             ; W ← W+12
                        ; D20 = 0001 0100 + 0000 1100 = 0010 0000 → já é BCD
                        ; D21 = 0001 0101 + 0000 1100 = 0010 0001 → já é BCD
                        ; D22 = 0001 0110 + 0000 1100 = 0010 0010 → já é BCD
                        ; D23 = 0001 0111 + 0000 1100 = 0010 0011 → já é BCD
                        ; D24 = 0001 1000 + 0000 1100 = 0010 0100 → já é BCD
                        ; D25 = 0001 1001 + 0000 1100 = 0010 0101 → já é BCD
                        ; D26 = 0001 1010 + 0000 1100 = 0010 0110 → já é BCD
                        ; D27 = 0001 1011 + 0000 1100 = 0010 0111 → já é BCD
                        ; D28 = 0001 1100 + 0000 1100 = 0010 1000 → já é BCD
```

```
                                ; D29 = 0001 1101 + 0000 1100 = 0010 1001 → já é BCD
        ANDLW   0x0F            ; W ← W AND 0x0F, parte baixa byte convertido BCD
        MOVWF   BCD_UNIDADE     ; BCD_UNIDADE ← W
        MOVLW   0x02
        MOVWF   BCD_DEZENA      ; BCD_DEZENA ← 0x02
        RETURN

ENTRE_30e39
        MOVF    WTEMP1,W        ; W ← WTEMP1
        ADDLW   .18             ; W ← W+18
                                ; D30 = 0001 1110 + 0001 0010 = 0011 0000 → já é BCD
                                ; D31 = 0001 1111 + 0001 0010 = 0011 0001 → já é BCD
                                ; D32 = 0010 0000 + 0001 0010 = 0011 0010 → já é BCD
                                ; D33 = 0010 0001 + 0001 0010 = 0011 0011 → já é BCD
                                ; D34 = 0010 0010 + 0001 0010 = 0011 0100 → já é BCD
                                ; D35 = 0010 0011 + 0001 0010 = 0011 0101 → já é BCD
                                ; D36 = 0010 0100 + 0001 0010 = 0011 0110 → já é BCD
                                ; D37 = 0010 0101 + 0001 0010 = 0011 0111 → já é BCD
                                ; D38 = 0010 0110 + 0001 0010 = 0011 1000 → já é BCD
                                ; D39 = 0010 0111 + 0001 0010 = 0011 1001 → já é BCD
        ANDLW   0x0F            ; W ← W AND 0x0F, parte baixa byte convertido BCD
        MOVWF   BCD_UNIDADE     ; BCD_UNIDADE ← W
        MOVLW   0x03
        MOVWF   BCD_DEZENA      ; BCD_DEZENA ← 0x03
        RETURN

ENTRE_40e49
        MOVF    WTEMP1,W        ; W ← WTEMP1
        ADDLW   .24             ; W ← W+24
                                ; D40 = 0010 1000 + 0001 1000 = 0100 0000 → já é BCD
                                ; D41 = 0010 1001 + 0001 1000 = 0100 0001 → já é BCD
                                ; D42 = 0010 1010 + 0001 1000 = 0100 0010 → já é BCD
                                ; D43 = 0010 1011 + 0001 1000 = 0100 0011 → já é BCD
                                ; D44 = 0010 1100 + 0001 1000 = 0100 0100 → já é BCD
                                ; D45 = 0010 1101 + 0001 1000 = 0100 0101 → já é BCD
                                ; D46 = 0010 1110 + 0001 1000 = 0100 0110 → já é BCD
                                ; D47 = 0010 1111 + 0001 1000 = 0100 0111 → já é BCD
                                ; D48 = 0011 0000 + 0001 1000 = 0100 1000 → já é BCD
                                ; D49 = 0011 0001 + 0001 1000 = 0100 1001 → já é BCD
        ANDLW   0x0F            ; W ← W AND 0x0F, parte baixa byte convertido BCD
        MOVWF   BCD_UNIDADE     ; BCD_UNIDADE ← W
        MOVLW   0x04
        MOVWF   BCD_DEZENA      ; BCD_DEZENA ← 0x04
        RETURN

ENTRE_50e59
        MOVF    WTEMP1,W        ; W ← WTEMP1
        ADDLW   .30             ; W ← W+30

        ANDLW   0x0F            ; W ← W AND 0x0F, parte baixa byte convertido BCD
        MOVWF   BCD_UNIDADE     ; BCD_UNIDADE ← W
        MOVLW   0x05
        MOVWF   BCD_DEZENA      ; BCD_DEZENA ← 0x05
        RETURN

ENTRE_60e69
```

```
        MOVF    WTEMP1,W        ; W ← WTEMP1
        ADDLW   .36             ; W ← W+36

        ANDLW   0x0F            ; W ← W AND 0x0F, parte baixa byte convertido BCD
        MOVWF   BCD_UNIDADE     ; BCD_UNIDADE ← W
        MOVLW   0x06
        MOVWF   BCD_DEZENA      ; BCD_DEZENA ← 0x06
        RETURN

ENTRE_70e79
        MOVF    WTEMP1,W        ; W ← WTEMP1
        ADDLW   .42             ; W ← W+42

        ANDLW   0x0F            ; W ← W AND 0x0F, parte baixa byte convertido BCD
        MOVWF   BCD_UNIDADE     ; BCD_UNIDADE ← W
        MOVLW   0x07
        MOVWF   BCD_DEZENA      ; BCD_DEZENA ← 0x07
        RETURN

ENTRE_80e89
        MOVF    WTEMP1,W        ; W ← WTEMP1
        ADDLW   .48             ; W ← W+48

        ANDLW   0x0F            ; W ← W AND 0x0F, parte baixa byte convertido BCD
        MOVWF   BCD_UNIDADE     ; BCD_UNIDADE ← W
        MOVLW   0x08
        MOVWF   BCD_DEZENA      ; BCD_DEZENA ← 0x08
        RETURN

ENTRE_90e99
        MOVF    WTEMP1,W        ; W ← WTEMP1
        ADDLW   .54             ; W ← W+54

        ANDLW   0x0F            ; W ← W AND 0x0F, parte baixa byte convertido BCD
        MOVWF   BCD_UNIDADE     ; BCD_UNIDADE ← W
        MOVLW   0x09
        MOVWF   BCD_DEZENA      ; BCD_DEZENA ← 0x09
        RETURN
;*******************************************************************************
```

b8) Sub-rotina ACHA_PASSO

Esta sub-rotina busca nas tabelas de Y_2 a Y_92 a quantidade de passos que deverão ser dados para ajustar o feixe radioativo rumo ao volume sensível do detector, câmara de ionização. Foi mostrado como exemplo dessas tabelas a Y_2 no item (b6) tópico (3).

Idéia básica das tabelas construídas no projeto

Para utilizar uma tabela deve-se, antes de chamá-la, acertar os 5 bits do registrador PCLATH, Fig. 5. Na programação em *assembly* do microcontrolador PIC16F877A uma tabela é, de fato, uma sub-rotina cujo retorno ao programa principal ocorre pela execução da instrução RETLW, que retorna com certo valor no registrador W. Este valor retornado em W é a informação guardada na tabela em determinada posição. O ponteiro da posição da tabela onde contém o valor a ser buscado é o resultado da soma do valor presente no registrador PCL com o valor para deslocamento dentro da tabela – *offset*. Este resultado é carregado no próprio PCL. Dado que o registrador PCL comporta 8 bits (pode-se guardar nele valores entre 0x00 a 0xFF) e que o deslocamento, *offset*, somado ao valor presente em PCL pode provocar o estouro em seu valor, ou seja, quando o resultado dessa soma ultrapassar o valor 0xFF, então deve-se atentar para esta situação ao programar. Toda vez que ocorre o referido estouro deve-se incrementar (somar 1) ao registrador PCLATH, e caso isso não seja feito dará conflito no deslocamento dentro da tabela, o que inviabilizaria todo o projeto.

Para ilustrar o exposto, foi utilizado uma sub-rotina com seus endereços devidamente ajustados para o exemplo, como pode ser visto no trecho de programa a seguir.

```
;******************** Rotina para ajuste PCLATH quando PCL não estoura **********************

                MOVLW       HIGH_PC
                MOVWF       PCLATH              ; PCLATH ←W
                MOVF        Deslocamento,W      ; W← Deslocamento

                CALL        BCD_ASCII           ; Chama tabela conversão
BCD_ASCII.
                :
                :

                ;
```

```
End. Pós
Compilação
                BCD_ASCII
0x1A00          ANDLW     B'00001111'   ; Mascarando o byte em W
0x1A01          ADDWF     PCL,F         ; PCL ← W + PCL, adiciona o
                                        ; deslocamento  dentro da tabela
                                        ; ao PCL (valor do 1º endereço
                                        ; do 1º dado da tabela).

0x1A02          RETLW B'00110000'       ; 0   = ASCII 30h
0x1A03          RETLW B'00110001'       ; 1   = ASCII 31h
0x1A04          RETLW B'00110010'       ; 2   = ASCII 32h
0x1A05          RETLW B'00110011'       ; 3   = ASCII 33h
0x1a06          RETLW B'00110100'       ; 4   = ASCII 34h
0x1A07          RETLW B'00110101'       ; 5   = ASCII 35h
0x1A08          RETLW B'00110110'       ; 6   = ASCII 36h
0x1A09          RETLW B'00110111'       ; 7   = ASCII 37h
0x1A0A          RETLW B'00111000'       ; 8   = ASCII 38h
0x1A0B          RETLW B'00111001'       ; 9   = ASCII 39h
;*******************************************************************************************
```

Porém, quando a tabela convertida ao código executável ocupar, inicialmente, a parte menos significativa do PC, a exemplo de PCL=0xFA e terminar com PCL=0x05, isto implica que o registrador PCLATH, no meio da tabela deverá ser incrementado, ou seja o PCLATH que era 0x19, incrementado, resulta em 0x1A, para não ocorrer conflito no deslocamento dentro desta tabela, como exemplificado, abaixo, no trecho de programa adaptado.

```
;****************** Rotina para ajuste PCLATH quando PCL estoura ***************************

        MOVLW     HIGH_PC
        MOVWF     PCLATH              ; PCLATH ←W
;-------------------------------Teste estouro de PCL-------------------------------
        BCF       STATUS,C            ; reseta o bit carry
        MOVF      Deslocamento,W      ; W ← Deslocamento
        MOVWF     WTEMP               ; WTEMP ← W
        MOVLW     LOW_PC              ; W ← LOW_PC [PCL 1º dado
                                      ; da tabela].
        ADDWF     WTEMP,F             ; WTEMP ← W + WTEMP
        BTFSC     STATUS,C            ; Houve estouro de PCL?
        INCF      PCLATH,F            ; Sim, acerta valor de PCLATH
        ;-------------------------------------------------------------------------------
        MOVF      Deslocamento,W      ; W← Deslocamento
        CALL      BCD_ASCII           ; Chama tabela conversão
                                      ; BCD_ASCII.
        :
        :
        :
```

```
End. Pós
Compilado
        BCD_ASCII

0x19FA      ANDLW       B'00001111'     ; Mascarando o byte em W
0x19FB      ADDWF        PCL,F          ; PCL ← W + PCL, adiciona o
                                        ; deslocamento  dentro da tabela
                                        ; ao PCL (valor do 1º endereço
                                                ; do 1º dado da tabela).
0x19FC      RETLW B'00110000'           ; 0   = ASCII 30h
0x19FD      RETLW B'00110001'           ; 1   = ASCII 31h
0x19FE      RETLW B'00110010'           ; 2   = ASCII 32h
0x19FF      RETLW B'00110011'           ; 3   = ASCII 33h
0x1A00      RETLW B'00110100'           ; 4   = ASCII 34h
0x1A01      RETLW B'00110101'           ; 5   = ASCII 35h
0x1A02      RETLW B'00110110'           ; 6   = ASCII 36h
0x1A03      RETLW B'00110111'           ; 7   = ASCII 37h
0x1A04      RETLW B'00111000'           ; 8   = ASCII 38h
0x1A05      RETLW B'00111001'           ; 9   = ASCII 39h
;*********************************************************************************
```

Então, quando se trabalha com muitas tabelas, tal como nesse projeto, uma ou outra tabela ocupará uma faixa de endereço como a do exemplo supra, o que demanda do programador um teste prévio da soma do PCL com o deslocamento dentro da tabela, para verificar, inicialmente, se haverá, ou não, o estouro do PCL. Havendo estouro, antes de chamar a sub-rotina (tabela), deve-se incrementar o PCLATH, evitando, assim, deslocamentos errados.

É importante observar que, também, tem outras forma de evitar o ocorrido no exemplo anterior. Para isto, basta o programador determinar o local na memória de programa onde deseja que passe a residir a tabela (para isso deve-se fazer uso da diretiva do MPLAB de nome ORG, como já foi visto), evitando, assim, o estouro do PCL, porém este procedimento pode não ser um modo eficientemente de utilização da memória de programa, que, como já se sabe, é um recurso escasso do microcontrolador. O fluxograma das Fig. A27 e Fig. A28 reportam a idéia desenvolvida no projeto real.

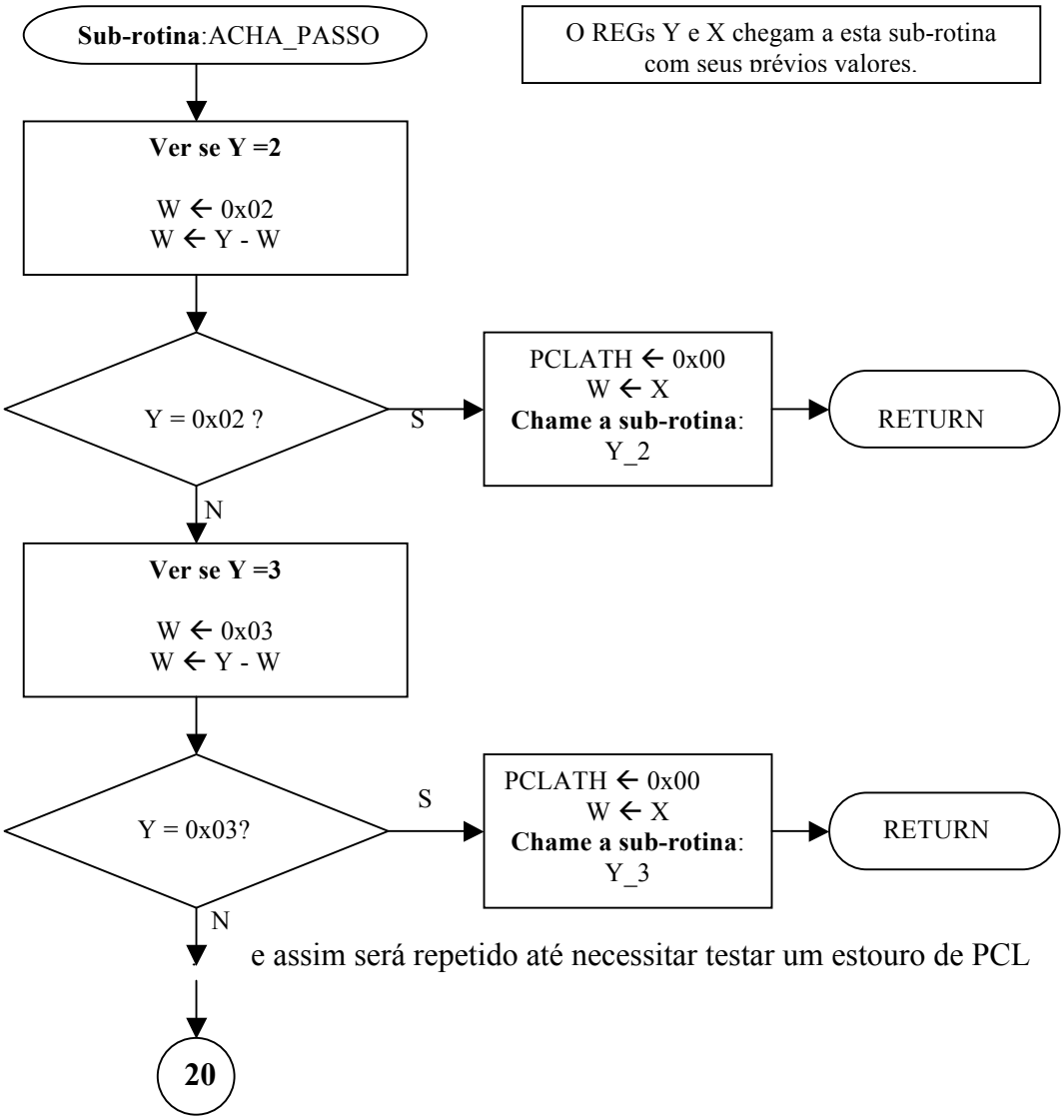


Figura A27– Fluxograma da Sub-rotina:
ACHA_PASSO – Inicial.

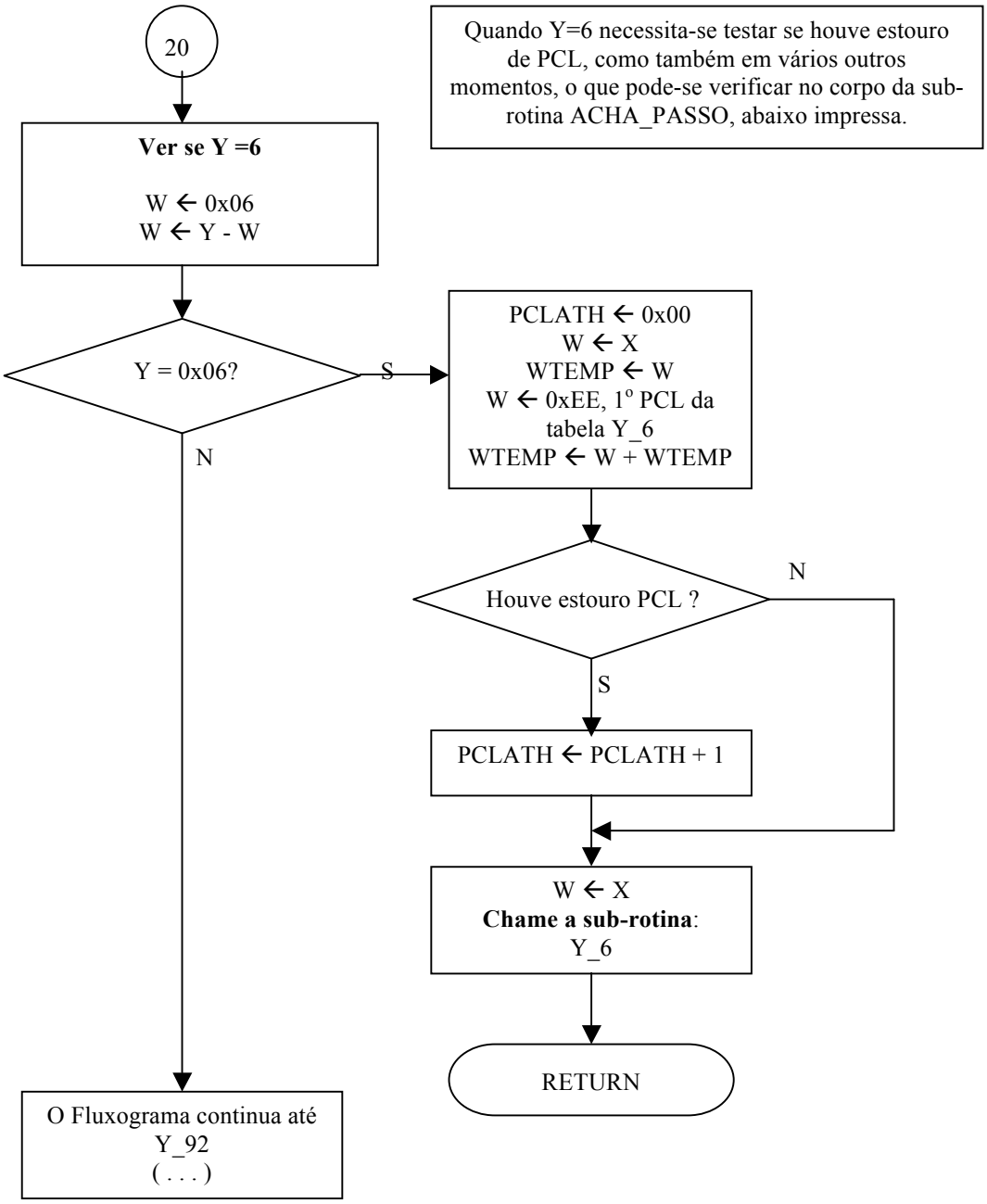


Figura A28 – Fluxograma da Sub-rotina: ACHA_PASSO – Final.

```
;*******************************************************************************
        ORG     0x1000                          ; Posiciona no 1º endereço da pág. 2
;*******************************************************************************
;* Esta sub-rotina acha o passo do motor para controle da cabeça fonte radioativa *
;*******************************************************************************
ACHA_PASSO                                      ; O Reg. Y já chega com certo valor entre 2 a 92
        BCF             STATUS,Z                ; Reseta o bit Z [BIT 2] do Reg. STATUS.
        MOVLW           0x02                    ; W ← 0x02
        SUBWF           Y,W                     ; W ← Y-W
        BTFSS           STATUS,Z                ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO            VE_SE_Y3
        MOVLW           0x00
        MOVWF           PCLATH                  ; PCLATH ← W, ajustado para a Página 0 = 00
        MOVF            X,W                     ; W ← X
        CALL            Y_2                     ; Chama tabela, dado X, encontra passo cabeça
        RETURN                                  ; Retorna ao programa principal.
VE_SE_Y3
        MOVLW           0x03                    ; W ← 0x03
        SUBWF           Y,W                     ; W ← Y-W
        BTFSS           STATUS,Z                ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO            VE_SE_Y4
        MOVLW           0x00
        MOVWF           PCLATH                  ; PCLATH ← W
        MOVF            X,W                     ; W ← X
        CALL            Y_3                     ; Chama tabela, dado X, encontra passo cabeça
        RETURN                                  ; Retorna ao programa principal.
VE_SE_Y4
        MOVLW           0x04                    ; W ← 0x04
        SUBWF           Y,W                     ; W ← Y-W
        BTFSS           STATUS,Z                ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO            VE_SE_Y5
        MOVLW           0x00
        MOVWF           PCLATH                  ; PCLATH ← W
        MOVF            X,W                     ; W ← X
        CALL            Y_4                     ; Chama tabela, dado X, encontra passo cabeça
        RETURN                                  ; Retorna ao programa principal.
VE_SE_Y5
        MOVLW           0x05                    ; W← 0x05
        SUBWF           Y,W                     ; W ← Y-W
        BTFSS           STATUS,Z                ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO            VE_SE_Y6
        MOVLW           0x00
        MOVWF           PCLATH                  ; PCLATH ← W
        MOVF            X,W                     ; W← X
        CALL            Y_5                     ; Chama tabela, dado X, encontra passo cabeça
        RETURN                                  ; Retorna ao programa principal.
VE_SE_Y6
        MOVLW           0x06                    ; W ← 0x06
        SUBWF           Y,W                     ; W ← Y-W
        BTFSS           STATUS,Z                ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO            VE_SE_Y7
        MOVLW           0x00
        MOVWF           PCLATH                  ; PCLATH ← W
    ;--------------------Teste estouro de PCL-------------------------------------------------------------------------
        BCF             STATUS,C                ; reseta o bit carry
```

```
        MOVF    X,W             ; W ← X
        MOVWF   WTEMP           ; WTEMP ← W
        MOVLW   0xEE            ; W ← 0xEE primeiro PCL da tabela Y_6
        ADDWF   WTEMP,F         ; WTEMP ← W + WTEMP
        BTFSC   STATUS,C        ; Houve estouro de PCL?
        INCF    PCLATH,F        ; Sim, acerta valor de PCLATH
    ;-------------------------------------------------------------------------------------------------------------

        MOVF    X,W             ; W ← X
        CALL    Y_6             ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y7
        MOVLW   0x07            ; W ← 0X07
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y8
        MOVLW   0x01
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W ← X
        CALL    Y_7             ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y8
        MOVLW   0x08            ; W ← 0x08
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y9
        MOVLW   0x01
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_8             ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y9
        MOVLW   0x09            ; W ← 0x09
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y10
        MOVLW   0x01
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W ← X
        CALL    Y_9             ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y10
        MOVLW   .10             ; W ← D'10'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y11
        MOVLW   0x01
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W ← X
        CALL    Y_10            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y11
        MOVLW   .11             ; W ← D'11'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y12
```

```
        MOVLW   0x01
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_11            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y12
        MOVLW   .12             ; W ← D'12'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y13

        MOVLW   0x01
        MOVWF   PCLATH          ; PCLATH ← W
    ;--------------------Teste estouro de PCL-----------------------------------------------------------------------
        BCF     STATUS,C        ; reseta o bit carry
        MOVF    X,W             ; W ← X
        MOVWF   WTEMP           ; WTEMP ← W
        MOVLW   0xF0            ; W ← 0xF0 primeiro PCL da tabela Y_12
        ADDWF   WTEMP,F         ; WTEMP ← W + WTEMP
        BTFSC   STATUS,C        ; Houve estouro de PCL?
        INCF    PCLATH,F        ; Sim, acerta valor de PCLATH
    ;-------------------------------------------------------------------------------------------------------------------

        MOVF    X,W             ; W← X
        CALL    Y_12            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y13
        MOVLW   .13             ; W ← D'13'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y14
        MOVLW   0x02
        MOVWF   PCLATH          ; PCLATH <- W
        MOVF    X,W             ; W← X
        CALL    Y_13            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y14
        MOVLW   .14             ; W ← D'14'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y15
        MOVLW   0x02
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_14            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y15
        MOVLW   .15             ; W ← D'15'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y16
        MOVLW   0x02
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_15            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
```

```
VE_SE_Y16
        MOVLW   .16             ; W ← D'16'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y17
        MOVLW   0x02
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W ← X
        CALL    Y_16            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y17
        MOVLW   .17             ; W ← D'17'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y18
        MOVLW   0x02
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_17            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y18
        MOVLW   .18             ; W ← D'18'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y19

        MOVLW   0x02
        MOVWF   PCLATH          ; PCLATH ← W
    ;-------------------Teste estouro de PCL-----------------------------------------------------------------------
        BCF     STATUS,C        ; reseta o bit carry
        MOVF    X,W             ; W ← X
        MOVWF   WTEMP           ; WTEMP ← W
        MOVLW   0xF2            ; W ← 0xF2 primeiro PCL da tabela Y_18
        ADDWF   WTEMP,F         ; WTEMP ← W + WTEMP
        BTFSC   STATUS,C        ; Houve estouro de PCL?
        INCF    PCLATH,F        ; Sim, acerta valor de PCLATH
    ;-------------------------------------------------------------------------------------------------------------

        MOVF    X,W             ; W← X
        CALL    Y_18            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y19
        MOVLW   .19             ; W ← D'19'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y20
        MOVLW   0x03
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_19            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y20
        MOVLW   .20             ; W ← D'20'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y21
```

```
        MOVLW   0x03
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_20            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y21
        MOVLW   .21             ; W ← D'21'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y22
        MOVLW   0x03
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W←X
        CALL    Y_21            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y22
        MOVLW   .22             ; W ← D'22'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y23
        MOVLW   0x03
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_22            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y23
        MOVLW   .23             ; W ← D'23'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y24
        MOVLW   0x03
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_23            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y24
        MOVLW   .24             ; W ← D'24'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y25

        MOVLW   0x03
        MOVWF   PCLATH          ; PCLATH ← W
    ;--------------------Teste estouro de PCL-----------------------------------------------------------------------------
        BCF     STATUS,C        ; reseta o bit carry
        MOVF    X,W             ; W ← X
        MOVWF   WTEMP           ; WTEMP ← W
        MOVLW   0xF4            ; W ← 0xF4 primeiro PCL da tabela Y_24
        ADDWF   WTEMP,F         ; WTEMP ← W + WTEMP
        BTFSC   STATUS,C        ; Houve estouro de PCL?
        INCF    PCLATH,F        ; Sim, acerta valor de PCLATH
    ;-------------------------------------------------------------------------------------------------------------------

        MOVF    X,W             ; W← X
        CALL    Y_24            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
```

```
VE_SE_Y25
        MOVLW   .25             ; W ← D'25'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y26
        MOVLW   0x04
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_25            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y26
        MOVLW   .26             ; W ← D'26'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y27
        MOVLW   0x04
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_26            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y27
        MOVLW   .27             ; W ← D'27'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y28
        MOVLW   0x04
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W ← X
        CALL    Y_27            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y28
        MOVLW   .28             ; W ← D'28'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y29
        MOVLW   0x04
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_28            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y29
        MOVLW   .29             ; W ← D'29'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y30
        MOVLW   0x04
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_29            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y30
        MOVLW   .30             ; W ← D'30'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y31
```

```
        MOVLW   0x04
        MOVWF   PCLATH          ; PCLATH ← W
    ;--------------------Teste estouro de PCL-----------------------------------------------------------------------
        BCF     STATUS,C        ; reseta o bit carry
        MOVF    X,W             ; W ← X
        MOVWF   WTEMP           ; WTEMP ← W
        MOVLW   0xF6            ; W ← 0xF6 primeiro PCL da tabela Y_30
        ADDWF   WTEMP,F         ; WTEMP ← W + WTEMP
        BTFSC   STATUS,C        ; Houve estouro de PCL?
        INCF    PCLATH,F        ; Sim, acerta valor de PCLATH
    ;-----------------------------------------------------------------------------------------------------------------

        MOVF    X,W             ; W← X
        CALL    Y_30            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y31
        MOVLW   .31             ; W ← D'31'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y32
        MOVLW   0x05
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_31            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y32
        MOVLW   .32             ; W ← D'32'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y33
        MOVLW   0x05
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_32            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y33
        MOVLW   .33             ; W ← D'33'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y34
        MOVLW   0x05
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_33            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y34
        MOVLW   .34             ; W ← D'34'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y35
        MOVLW   0x05
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_34            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y35
```

```
        MOVLW   .35             ; W ← D'35'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y36
        MOVLW   0x05
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_35            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y36
        MOVLW   .36             ; W ← D'36'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y37

        MOVLW   0x05
        MOVWF   PCLATH          ; PCLATH ← W
    ;-------------------Teste estouro de PCL-------------------------------------------------------------
        BCF     STATUS,C        ; reseta o bit carry
        MOVF    X,W             ; W ← X
        MOVWF   WTEMP           ; WTEMP ← W
        MOVLW   0xF8            ; W ← 0xF8 primeiro PCL da tabela Y_36
        ADDWF   WTEMP,F         ; WTEMP ← W + WTEMP
        BTFSC   STATUS,C        ; Houve estouro de PCL?
        INCF    PCLATH,F        ; Sim, acerta valor de PCLATH
    ;-----------------------------------------------------------------------------------------------------------

        MOVF    X,W             ; W← X
        CALL    Y_36            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y37
        MOVLW   .37             ; W ← D'37'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y38
        MOVLW   0x06
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_37            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y38
        MOVLW   .38             ; W ← D'38'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y39
        MOVLW   0x06
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_38            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y39
        MOVLW   .39             ; W ← D'39'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y40
        MOVLW   0x06
```

```
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W←X
        CALL    Y_39            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y40
        MOVLW   .40             ; W← D'40'
        SUBWF   Y,W             ; W← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y41
        MOVLW   0x06
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_40            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y41
        MOVLW   .41             ; W ← D'41'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y42
        MOVLW   0x06
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_41            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y42
        MOVLW   .42             ; W ← D'42'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y43

        MOVLW   0x06
        MOVWF   PCLATH          ; PCLATH ← W
    ;--------------------Teste estouro de PCL-----------------------------------------------------------------------
        BCF     STATUS,C        ; reseta o bit carry
        MOVF    X,W             ; W ← X
        MOVWF   WTEMP           ; WTEMP ← W
        MOVLW   0xFA            ; W ← 0xFA primeiro PCL da tabela Y_42
        ADDWF   WTEMP,F         ; WTEMP ← W + WTEMP
        BTFSC   STATUS,C        ; Houve estouro de PCL?
        INCF    PCLATH,F        ; Sim, acerta valor de PCLATH
    ;-------------------------------------------------------------------------------------------------------------------

        MOVF    X,W             ; W← X
        CALL    Y_42            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y43
        MOVLW   .43             ; W ← D'43'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y44
        MOVLW   0x07
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_43            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y44
```

```
        MOVLW   .44             ; W ← D'44'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y45
        MOVLW   0x07
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_44            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y45
        MOVLW   .45             ; W ← D'45'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y46
        MOVLW   0x07
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_45            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y46
        MOVLW   .46             ; W ← D'46'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST Se 1
        GOTO    VE_SE_Y47
        MOVLW   0x07
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_46            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y47
        MOVLW   .47             ; W ← D'47'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y48
        MOVLW   0x07
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_47            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y48
        MOVLW   .48             ; W ← D'48'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y49
        MOVLW   0x08            ; W← 0x08
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_48            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y49
        MOVLW   .49             ; W ← D'49'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y50
        MOVLW   0x08
        MOVWF   PCLATH          ; PCLATH ← W
```

```
        MOVF    X,W             ; W← X
        CALL    Y_49            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y50
        MOVLW   .50             ; W← D'50'
        SUBWF   Y,W             ; W← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y51
        MOVLW   0x08
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_50            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y51
        MOVLW   .51             ; W ← D'51'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y52
        MOVLW   0x08
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_51            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y52
        MOVLW   .52             ; W ← D'52'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y53
        MOVLW   0x08
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_52            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y53
        MOVLW   .53             ; W← D'53'
        SUBWF   Y,W             ; W← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y54
        MOVLW   0x08
        MOVWF   PCLATH          ; PCLATH ← W
    ;--------------------Teste estouro de PCL-------------------------------------------------------------------------
        BCF     STATUS,C        ; reseta o bit carry
        MOVF    X,W             ; W ← X
        MOVWF   WTEMP           ; WTEMP ← W
        MOVLW   0xD9            ; W ← 0xD9 primeiro PCL da tabela Y_53
        ADDWF   WTEMP,F         ; WTEMP ← W + WTEMP
        BTFSC   STATUS,C        ; Houve estouro de PCL?
        INCF    PCLATH,F        ; Sim, acerta valor de PCLATH
    ;-----------------------------------------------------------------------------------------------------------------

        MOVF    X,W             ; W← X
        CALL    Y_53            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y54
        MOVLW   .54             ; W ← D'54'
        SUBWF   Y,W             ; W ← Y-W
```

```
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y55
        MOVLW   0x09
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_54            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y55
        MOVLW   .55             ; W ← D'55'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y56
        MOVLW   0x09
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_55            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y56
        MOVLW   .56             ; W ← D'56'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y57
        MOVLW   0x09
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_56            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y57
        MOVLW   .57             ; W ← D'57'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y58
        MOVLW   0x09
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_57            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y58
        MOVLW   .58             ; W ← D'58'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y59
        MOVLW   0x09
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_58            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y59
        MOVLW   .59             ; W ← D'59'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y60
        MOVLW   0x09
        MOVWF   PCLATH          ; PCLATH ← W
    ;--------------------Teste estouro de PCL-------------------------------------------------------------------------------
        BCF     STATUS,C        ; reseta o bit carry
```

```
        MOVF    X,W             ; W ← X
        MOVWF   WTEMP           ; WTEMP ← W
        MOVLW   0xDB            ; W ← 0xDB primeiro PCL da tabela Y_59
        ADDWF   WTEMP,F         ; WTEMP ← W + WTEMP
        BTFSC   STATUS,C        ; Houve estouro de PCL?
        INCF    PCLATH,F        ; Sim, acerta valor de PCLATH
    ;-----------------------------------------------------------------------------------------------------------------
        MOVF    X,W             ; W← X
        CALL    Y_59            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y60
        MOVLW   .60             ; W ← D'60'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y61
        MOVLW   0x0A
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_60            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y61
        MOVLW   .61             ; W ← D'61'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y62
        MOVLW   0x0A
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_61            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y62
        MOVLW   .62             ; W ← D'62'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y63
        MOVLW   0x0A
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_62            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y63
        MOVLW   .63             ; W ← D'63'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y64
        MOVLW   0x0A
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_63            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y64
        MOVLW   .64             ; W ← D'64'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y65
        MOVLW   0x0A
```

```
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_64            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y65
        MOVLW   .65             ; W ← D'65'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y66
        MOVLW   0x0A
        MOVWF   PCLATH          ; PCLATH ← W

    ;--------------------Teste estouro de PCL-----------------------------------------------------------------------------
        BCF     STATUS,C        ; reseta o bit carry
        MOVF    X,W             ; W ← X
        MOVWF   WTEMP           ; WTEMP ← W
        MOVLW   0xDD            ; W ← 0xDD primeiro PCL da tabela Y_65
        ADDWF   WTEMP,F         ; WTEMP ← W + WTEMP
        BTFSC   STATUS,C        ; Houve estouro de PCL?
        INCF    PCLATH,F        ; Sim, acerta valor de PCLATH
    ;-------------------------------------------------------------------------------------------------------------------

        MOVF    X,W             ; W← X
        CALL    Y_65            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y66
        MOVLW   .66             ; W ← D'66'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y67
        MOVLW   0x0B
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_66            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y67
        MOVLW   .67             ; W ← D'67'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y68
        MOVLW   0x0B
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_67            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y68
        MOVLW   .68             ; W ← D'68'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y69
        MOVLW   0x0B
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_68            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y69
```

```
        MOVLW   .69             ; W ← D'69'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y70
        MOVLW   0x0B
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_69            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y70
        MOVLW   .70             ; W ← D'70'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y71
        MOVLW   0x0B
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_70            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y71
        MOVLW   .71             ; W ← D'71'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y72

        MOVLW   0x0B
        MOVWF   PCLATH          ; PCLATH ← W
    ;-------------------Teste estouro de PCL-------------------------------------------------------------------------
        BCF     STATUS,C        ; reseta o bit carry
        MOVF    X,W             ; W ← X
        MOVWF   WTEMP           ; WTEMP ← W
        MOVLW   0xDF            ; W ← 0xDF primeiro PCL da tabela Y_71
        ADDWF   WTEMP,F         ; WTEMP ← W + WTEMP
        BTFSC   STATUS,C        ; Houve estouro de PCL?
        INCF    PCLATH,F        ; Sim, acerta valor de PCLATH
    ;-------------------------------------------------------------------------------------------------------------------

        MOVF    X,W             ; W← X
        CALL    Y_71            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y72
        MOVLW   .72             ; W ← D'72'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y73
        MOVLW   0x0C
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_72            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y73
        MOVLW   .73             ; W ← D'73'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y74
        MOVLW   0x0C
```

```
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_73            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y74
        MOVLW   .74             ; W ← D'74'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y75
        MOVLW   0x0C
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_74            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y75
        MOVLW   .75             ; W ← D'75'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y76
        MOVLW   0x0C
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_75            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y76
        MOVLW   .76             ; W ← D'76'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y77
        MOVLW   0x0C
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_76            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y77
        MOVLW   .77             ; W ← D'77'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y78

        MOVLW   0x0C
        MOVWF   PCLATH          ; PCLATH ← W
    ;-------------------Teste estouro de PCL-------------------------------------------------------------------------
        BCF     STATUS,C        ; reseta o bit carry
        MOVF    X,W             ; W ← X
        MOVWF   WTEMP           ; WTEMP ← W
        MOVLW   0xE1            ; W ← 0xE1 primeiro PCL da tabela Y_77
        ADDWF   WTEMP,F         ; WTEMP ← W + WTEMP
        BTFSC   STATUS,C        ; Houve estouro de PCL?
        INCF    PCLATH,F        ; Sim, acerta valor de PCLATH
    ;-------------------------------------------------------------------------------------------------------------------
        MOVF    X,W             ; W← X
        CALL    Y_77            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y78
        MOVLW   .78             ; W ← D'78'
```

```
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y79
        MOVLW   0x0D
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_78            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y79
        MOVLW   .79             ; W ← D'79'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y80
        MOVLW   0x0D
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_79            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y80
        MOVLW   .80             ; W ← D'80'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y81
        MOVLW   0x0D
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_80            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y81
        MOVLW   .81             ; W ← D'81'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y82
        MOVLW   0x0D
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_81            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y82
        MOVLW   .82             ; W ← D'82'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y83
        MOVLW   0x0D
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_82            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y83
        MOVLW   .83             ; W ← D'83'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y84
        MOVLW   0x0D
        MOVWF   PCLATH          ; PCLATH ← W
    ;--------------------Teste estouro de PCL-------------------------------------------------------------------------------------
```

```
        BCF     STATUS,C        ; reseta o bit carry
        MOVF    X,W             ; W ← X
        MOVWF   WTEMP           ; WTEMP ← W
        MOVLW   0xE3            ; W ← 0xE3 primeiro PCL da tabela Y_83
        ADDWF   WTEMP,F         ; WTEMP ← W + WTEMP
        BTFSC   STATUS,C        ; Houve estouro de PCL?
        INCF    PCLATH,F        ; Sim, acerta valor de PCLATH
    ;-----------------------------------------------------------------------------------------------------------------------
        MOVF    X,W             ; W← X
        CALL    Y_83            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y84
        MOVLW   .84             ; W ← D'84'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y85
        MOVLW   0x0E
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_84            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y85
        MOVLW   .85             ; W ← D'85'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y86
        MOVLW   0x0E
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_85            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y86
        MOVLW   .86             ; W ← D'86'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y87
        MOVLW   0x0E
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_86            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y87
        MOVLW   .87             ; W ← D'87'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y88
        MOVLW   0x0E
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_87            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y88
        MOVLW   .88             ; W ← D'88'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y89
```

```
        MOVLW   0x0E
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_88            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y89
        MOVLW   .89             ; W ←D'89'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y90
        MOVLW   0x0E
        MOVWF   PCLATH          ; PCLATH ← W
    ;-------------------Teste estouro de PCL-----------------------------------------------------------------------------
        BCF     STATUS,C        ; reseta o bit carry
        MOVF    X,W             ; W ← X
        MOVWF   WTEMP           ; WTEMP ← W
        MOVLW   0xE5            ; W ← 0xE5 primeiro PCL da tabela Y_89
        ADDWF   WTEMP,F         ; WTEMP ← W + WTEMP
        BTFSC   STATUS,C        ; Houve estouro de PCL?
        INCF    PCLATH,F        ; Sim, acerta valor de PCLATH
    ;-------------------------------------------------------------------------------------------------------------------
        MOVF    X,W             ; W← X
        CALL    Y_89            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y90
        MOVLW   .90             ; W ← D'90'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y91
        MOVLW   0x0F
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_90            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y91
        MOVLW   .91             ; W ← D'91'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        GOTO    VE_SE_Y92
        MOVLW   0x0F
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_91            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
VE_SE_Y92
        MOVLW   .92             ; W ← D'92'
        SUBWF   Y,W             ; W ← Y-W
        BTFSS   STATUS,Z        ; TESTA BIT Z SALTA PRÓX. INST SE 1
        RETURN                  ; Retorna ao programa principal.
        MOVLW   0x0F
        MOVWF   PCLATH          ; PCLATH ← W
        MOVF    X,W             ; W← X
        CALL    Y_92            ; Chama tabela, dado X, encontra passo cabeça
        RETURN                  ; Retorna ao programa principal.
;-----------------------------------------------------------------------------------------------------------------------
```

b9) Sub-rotina Y_ASCII

Esta sub-rotina, mostrada na Fig. A29, converte o valor binário contido no registrador Y em ASCII, ou seja ASCII da dezena será armazenado no registrador Y_DEZENA e da unidade em Y_UNIDADE.

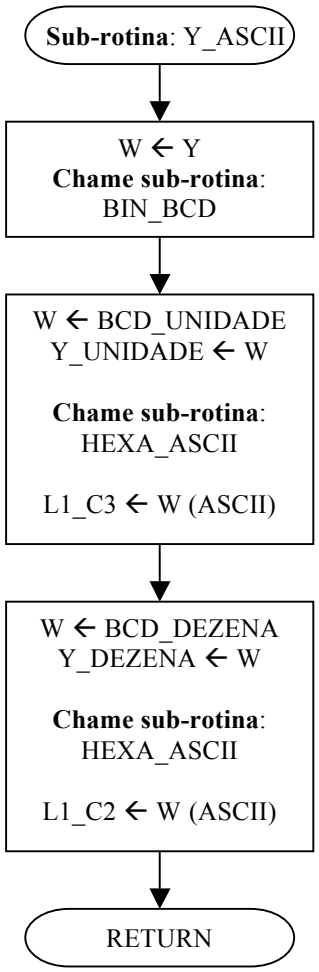


Figura A29 – Fluxograma da Sub-rotina: Y_ASCII

```
;*******************************************************************************
;* Converte Y (binário → ASCII [resultado nos registradores YDEZENA, Y_UNIDADE]                *
;*******************************************************************************
Y_ASCII
        MOVF        Y,W                 ; W ← Y
        PAGESEL     BIN_BCD             ; ajusta PCLATH
        CALL        BIN_BCD             ; Retorna BCD_UNIDADE e [nibble menos sig.]
                                        ; Dezena  em BCD_DEZENA .
        MOVF        BCD_UNIDADE,W       ; W ← BCD_UNIDADE
        MOVWF       Y_UNIDADE           ; Y_UNIDADE ← W [Y_UNIDADE, unidade Y, BCD]
        CLRF        PCLATH              ; PCLATH ← 0x00
        CALL        HEXA_ASCII          ; Converte BCD_UNIDADE (Y_UNIDADE) em ASCII
        MOVWF       L1_C3               ; L1_C3 ← W
                                        ; [L1_C3, unidade de Y em ASCII para envio ao LCD]
        MOVF        BCD_DEZENA,W        ; W ← BCD_DEZENA
        MOVWF       Y_DEZENA            ; Y_DEZENA ← W [Y_DEZENA, dezena de Y, BCD]
        CALL        HEXA_ASCII          ; Converte o BCD_DEZENA (Y_DEZENA) em ASCII
        MOVWF       L1_C2               ; L1_C2 ← W [L1_C2, dezena de Y já em ASCII]
        RETURN                          ; Situação de retorno: Conversão do valor em Y em:
                                        ; Y_UNIDADE → carregado com unidade de Y (BCD),
                                        ; Y_DEZENA → carregado com dezena de Y (BCD)
                                        ; L1_C3 ← Y_UNIDADE, agora em ASCII,
                                        ; L1_C2 ← Y_DEZENA, agora em ASCII.
;*******************************************************************************
```

b10) Sub-rotina SINAL_X_ASCII

Esta sub-rotina, mostrada na Fig. A30, converte o valor binário contido no registrador X em ASCII.

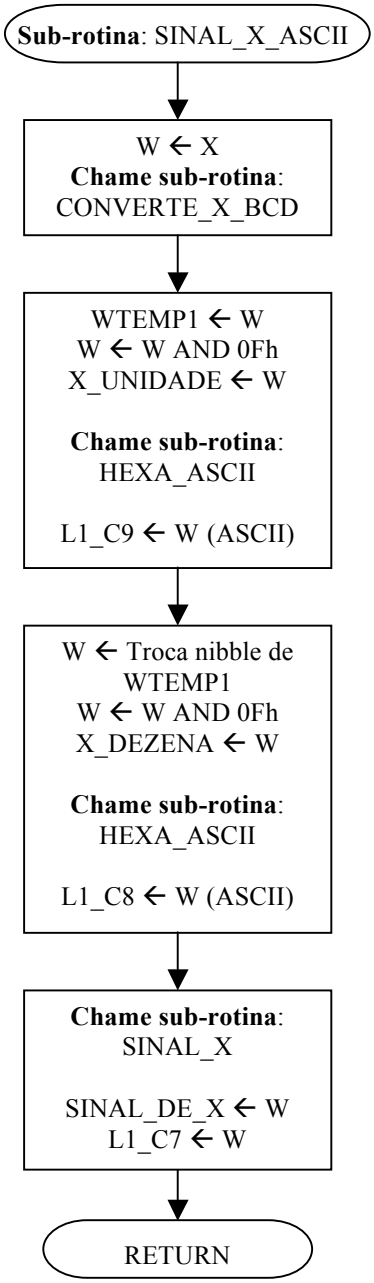


Figura A30 – Fluxograma da Sub-rotina: SINAL_X_ASCII

```
;*******************************************************************************
;* Converte X (binário → ASCII [resultado nos registradores Sinal de X, X_DEZENA X_UNIDADE.      *
;*******************************************************************************
SINAL_X_ASCII
        MOVF        X,W                 ; W ← X
                                        ; X de 0 a 41 é convertido para de +20 a -20 (sem sinal)
        CLRF        PCLATH              ; PCLATH ← 0x00
        CALL        CONVERTE_X_BCD      ; Fornece num byte X_DEZENA e X_UNIDADE em W
        MOVWF       WTEMP1              ; WTEMP1 ← W
        ANDLW       0x0F                ; W ← W AND 0Fh
        MOVWF       X_UNIDADE           ; X_UNIDADE ← W [X_UNIDADE,  em BCD]
        CALL        HEXA_ASCII          ; Converte o BCD em ASCII
        MOVWF       L1_C9               ; L1_C9 ← W [L1_C9, unidade X em ASCII no LCD]
        SWAPF       WTEMP1,W            ; troca nibble mais sig com menos sig, resultado em W
        ANDLW       0x0F                ; W ← W AND 0Fh
        MOVWF       X_DEZENA            ; X_DEZENA ← W [X_DEZENA, em BCD]
        CALL        HEXA_ASCII          ; Converte o BCD em ASCII
        MOVWF       L1_C8               ; L1_C8 ← W [L1_C8, dezena X já em ASCII no LCD]

        PAGESEL     SINAL_X
        CALL        SINAL_X
        MOVWF       SINAL_DE_X          ; SINAL_DE_X ← W   carrega ASCII do sinal
        MOVWF       L1_C7               ; L1_C7 ← W   carrega ASCII do sinal no LCD
        RETURN                          ; Situação de retorno: Conversão do valor em X em:
                                        ; X_UNIDADE → carregado com unidade de X (BCD),
                                        ; X_DEZENA → carregado com dezena de X (BCD)
                                        ; L1_C9 ← X_UNIDADE, agora em ASCII,
                                        ; L1_C8 ← X_DEZENA, agora em ASCII.
                                        ; L1_C7 ← SINAL_DE_X em ASCII
;*******************************************************************************
```

b11) Sub-rotina SINAL_X

Esta sub-rotina, mostrada na Fig. A31, grafa com o sinal negativo (-) e o positivo (*space*) o valor contido no registrador X, isto em ASCII.

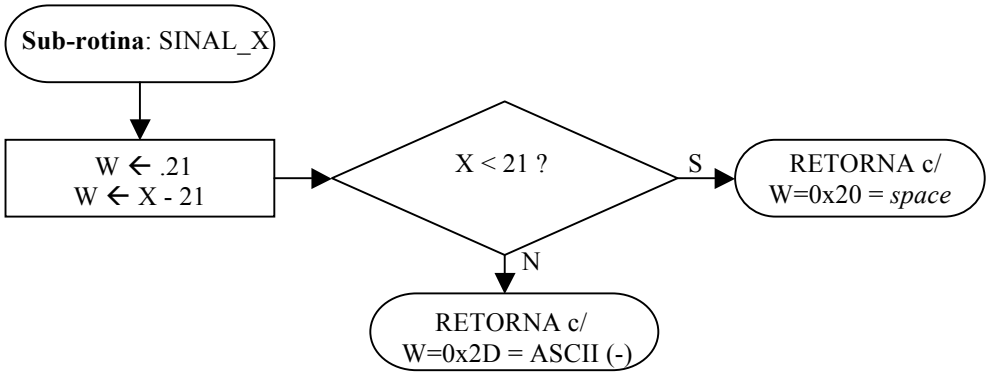


Figura A31 – Fluxograma da Sub-rotina: SINAL_X

```
;*******************************************************************************
;* Acerta 0 sinal de X já em ASCII: (-) 2D = 0010 1101  e (+) 2B = 0010 1011   *
;*******************************************************************************
SINAL_X
        MOVLW       .21                 ; Qdo X > 21 sinal é negativo
        SUBWF       X,W                 ; W ← X - 21
        BTFSS       STATUS,C            ; X < 21?
        RETLW       0x20                ; Sim, retorna em W valor ASCII sinal SPACE [=0x20]
        RETLW       0x2D                ; Não, retorna em W valor ASCII do sinal - [0x2D]
; *******************************************************************************
```

b12) Sub-rotina Passo_rapido_0a90

Esta sub-rotina, mostrada na Fig. A32, controla o movimento rápido da mesa Y até que atinja sua posição inicial.

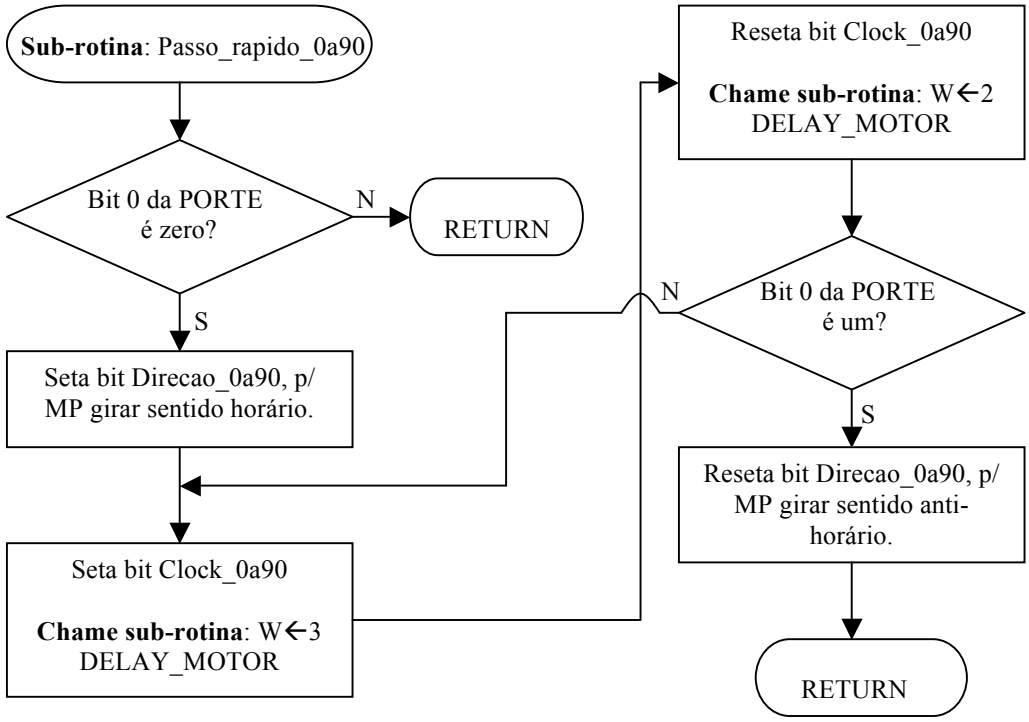


Figura A32 – Fluxograma da Sub-rotina: Passo_rapido_0a90

```
; ********************************************************************************************
; *    Rotina para controle do motor de passo girando no sentido horário, aproximando Y da sua posicão_0    *
; ********************************************************************************************

Passo_rapido_0a90
        BTFSC       Posicao_0_0a90      ; Testa bit 0 da PORTE e salta próxima instrução se zero.
        RETURN                          ; se carro está na posicão_0 → fim desta sub-rotina.
                                        ; se não, salta RETURN e dá sequência à sub-rotina.
        BSF         Direcao_0a90        ; seta bit 2 PORTE, bit p/ sentido rotação motor passo.
                                        ; Este bit=1, motor roda no sentido horário, mesa move
Pulso_0a90                              ; em direção à posição zero, início do processo.
        BSF         Clock_0a90          ; seta bit 1 PORTE, inicia pulso de clock de um passo.
        MOVLW       .3
        CALL        DELAY_MOTOR         ; DELAY Motor para clock setado
        BCF         Clock_0a90
        MOVLW       .2
        CALL        DELAY_MOTOR         ; DELAY Motor para clock resetado
        BTFSS       Posicao_0_0a90
        GOTO        Pulso_0a90
        BCF         Direcao_0a90
        RETURN
; ********************************************************************************************
```

b13) Sub-rotina Passo_rapido_0a41: Esta sub-rotina, mostradas nas Fig. A33 e Fig. A34, controla o movimento rápido da mesa X até que atinja sua posição inicial.

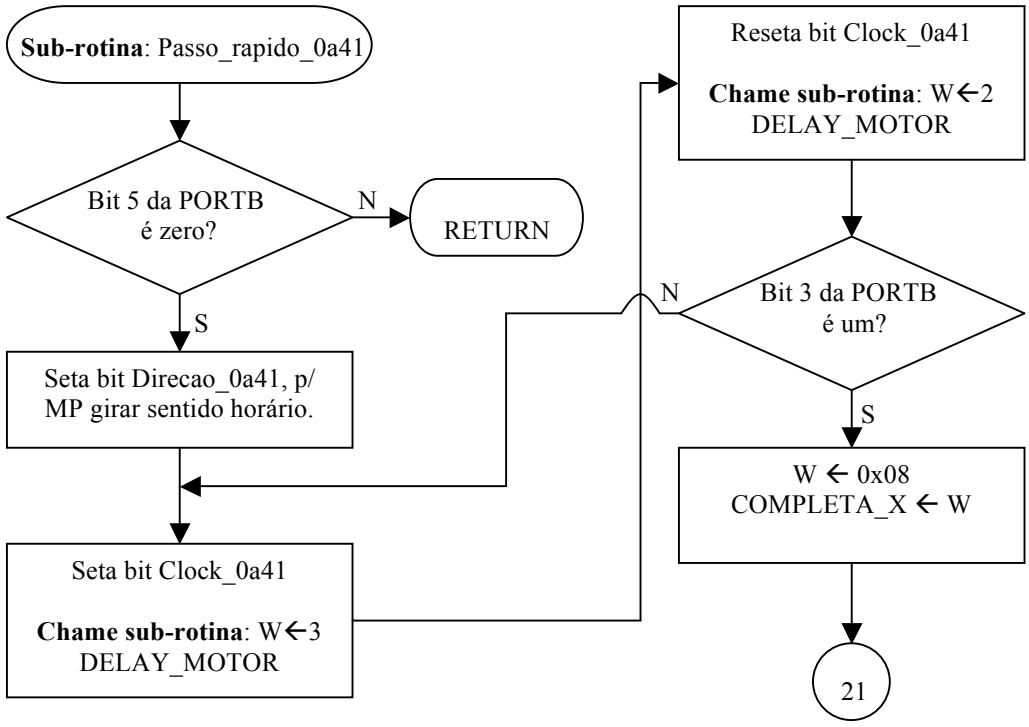


Figura A33 – Fluxograma da Sub-rotina: Passo_rapido_0a41 - Inicial

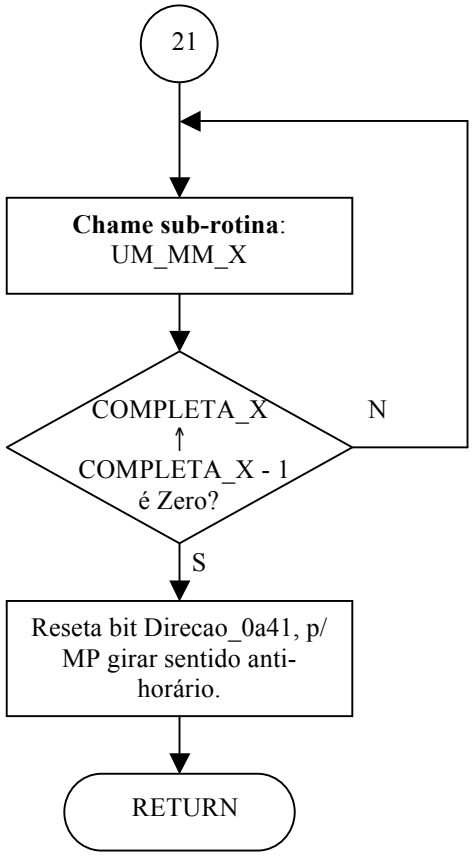


Figura A34 – Fluxograma da Sub-rotina: Passo_rapido_0a41 - Final

```
; ********************************************************************************
; *    Rotina para controle do motor de passo girando no sentido horário, aproximando X da sua posicão_0    *
; ********************************************************************************
Passo_rapido_0a41
        BTFSC       Posicao_0_0a41      ; Testa bit 5 da PORTB, salta próxima instrução se zero.
        RETURN                          ; se carro está na posicão_0 → fim desta sub-rotina.
        BSF         Direcao_0a41        ; seta bit 7 PORTB, bit para sentido rotação motor passo.
Pulso_0a41                              ; Este bit=1, motor roda no sentido horário, o que faz a
        BSF         Clock_0a41          ; seta bit 6 PORTB, inicia pulso de clock para um passo.
        MOVLW       .3
        CALL        DELAY_MOTOR         ; DELAY Motor para clock setado
        BCF         Clock_0a41
        MOVLW       .2
        CALL        DELAY_MOTOR         ; DELAY Motor para clock resetado
        BTFSS       Posicao_0_0a41
        GOTO        Pulso_0a41
        MOVLW       .8                  ; Deve-se,ainda, avançar 8 mm
        MOVWF       COMPLETA_X
AJUSTA_X
        CALL        UM_MM_X
        DECFSZ      COMPLETA_X,F
        GOTO        AJUSTA_X
        BCF         Direcao_0a41
        RETURN
; ********************************************************************************
```

b14) Sub-rotina Passo_cabeca

Esta sub-rotina, mostrada na Fig. A 35, controla o movimento da cabeça (onde reside a fonte radioativa) até sua posição inicial.

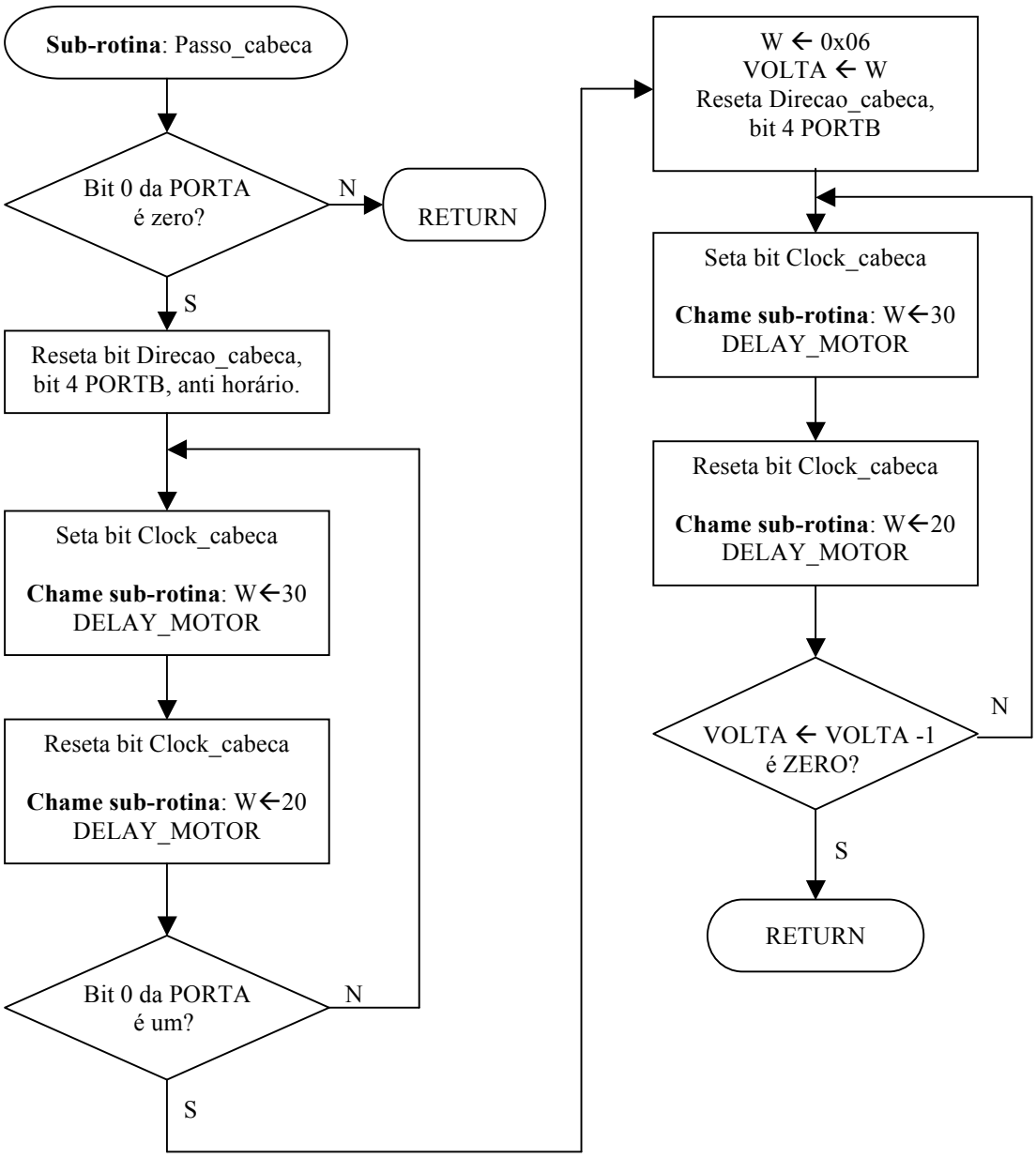


Figura A35 – Fluxograma da Sub-rotina: Passo_cabeca.

```
; ********************************************************************************
; *   Rotina para controle do motor de passo girando no sentido horário, aproximando CABECA posicão_0   *
; ********************************************************************************
Passo_cabeca
        BTFSC       Pos_0_cabeca        ; Testa bit 0, PORTA e salta próxima instrução se zero.
        RETURN                          ; se carro está na posicão_0 → fim desta sub-rotina.
                                        ; se não, salta RETURN e dá sequência à sub-rotina.
        BCF         Direcao_cabeca      ; reseta bit 4 PORTB, bit p/ sentido rotação motor passo.
                                        ; rotação no sentido anti-horário.
Pulso_cabeca                            ; Este bit=1, motor roda no sentido horário, o que faz a
                                        ; mover-se em direção à posição zero, início do processo.
        BSF         Clock_cabeca        ; seta bit 1 PORTA, inicia um pulso clock p/ um passo.
        MOVLW       .30
        CALL        DELAY_MOTOR         ; DELAY Motor para clock setado
        BCF         Clock_cabeca
        MOVLW       .20
        CALL        DELAY_MOTOR         ; DELAY Motor para clock resetado
        BTFSS       Pos_0_cabeca
        GOTO        Pulso_cabeca
        MOVLW       .6                  ; carrega número de passos necessários para uma volta.
        MOVWF       VOLTA
        BCF         Direcao_cabeca      ; seta bit 4 PORTB, bit para sentido rotação motor passo.
                                        ; rotação no sentido anti-horário.
PASSO_2
        BSF         Clock_cabeca        ; seta bit 1 PORTA, inicia um pulso clock p/ um passo.
        MOVLW       .30
        CALL        DELAY_MOTOR         ; DELAY Motor para clock setado
        BCF         Clock_cabeca
        MOVLW       .20
        CALL        DELAY_MOTOR         ; DELAY Motor para clock resetado
        DECFSZ      VOLTA,F             ; VOLTA←VOLTA-1, qdo zero salta próxima instrução.
        GOTO        PASSO_2

        RETURN
; ********************************************************************************
```

b15) Sub-rotina ACERTA_CABECA

Esta sub-rotina, Fig. A 36, acerta a direção da fonte radioativa rumo ao volume sensível do detector, câmara de ionização.

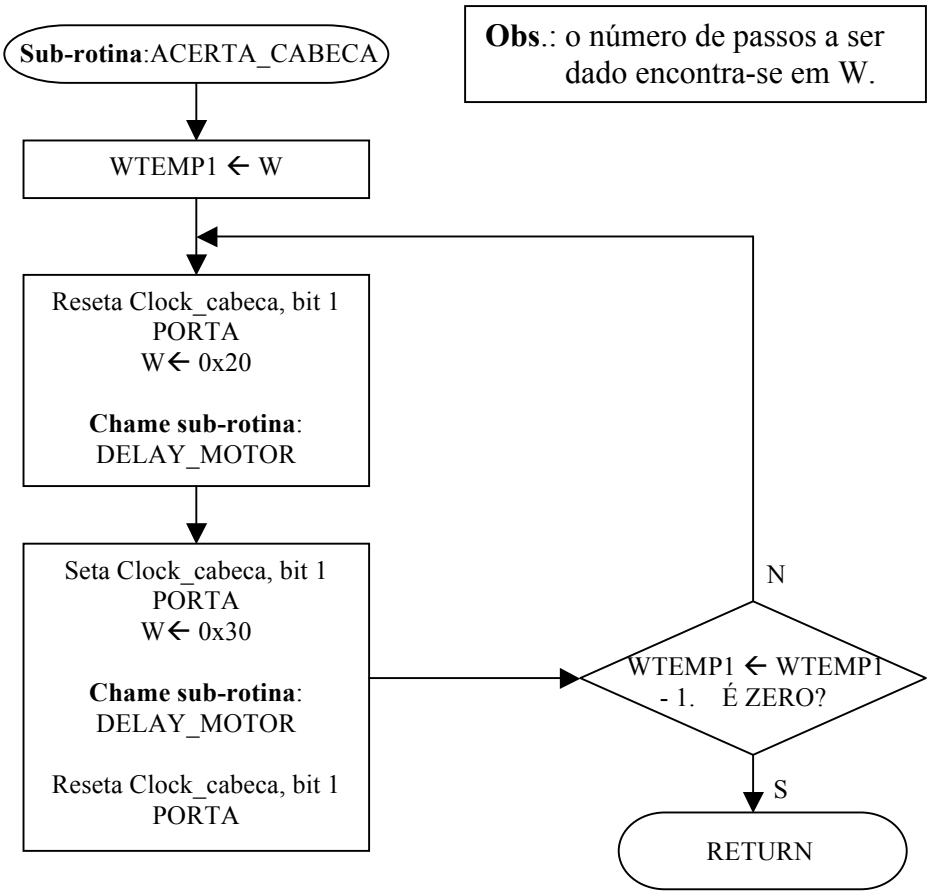


Figura A36 – Fluxograma da Sub-rotina: ACERTA_CABECA.

```
; ********************************************************************************
; *  Rotina para acertar a direção da fonte em relação ao detector, roda cabeça no sentido anti-horário  *
; ********************************************************************************
ACERTA_CABECA                           ; W já chega com a quantidade de passos a serem dados.
        MOVWF       WTEMP1              ; WTEMP1 ← W
NOVO_PASSO2
        BCF         Clock_cabeca        ; reseta bit 1 PORTA, inicia um pulso clock p/ um passo.
        MOVLW       .20
        CALL        DELAY_MOTOR         ; DELAY Motor para clock setado
        BSF         Clock_cabeca
        MOVLW       .30
        CALL        DELAY_MOTOR         ; DELAY Motor para clock resetado
        BCF         Clock_cabeca        ; seta bit 1 PORTA, inicia um pulso clock p/ um passo.
        DECFSZ      WTEMP1,F            ;WTEMP1←WTEMP1-1, se (0) salta próxima instrução.
        GOTO        NOVO_PASSO2
        RETURN
;********************************************************************************
```

b16) Sub-rotina DELAY_MOTOR

A sub-rotina mostrada na Fig. A37 é para atrasar de n * 486μs, entre um passo e outro do motor de passo, onde n é um valor em W antes de chamar esta sub-rotina.

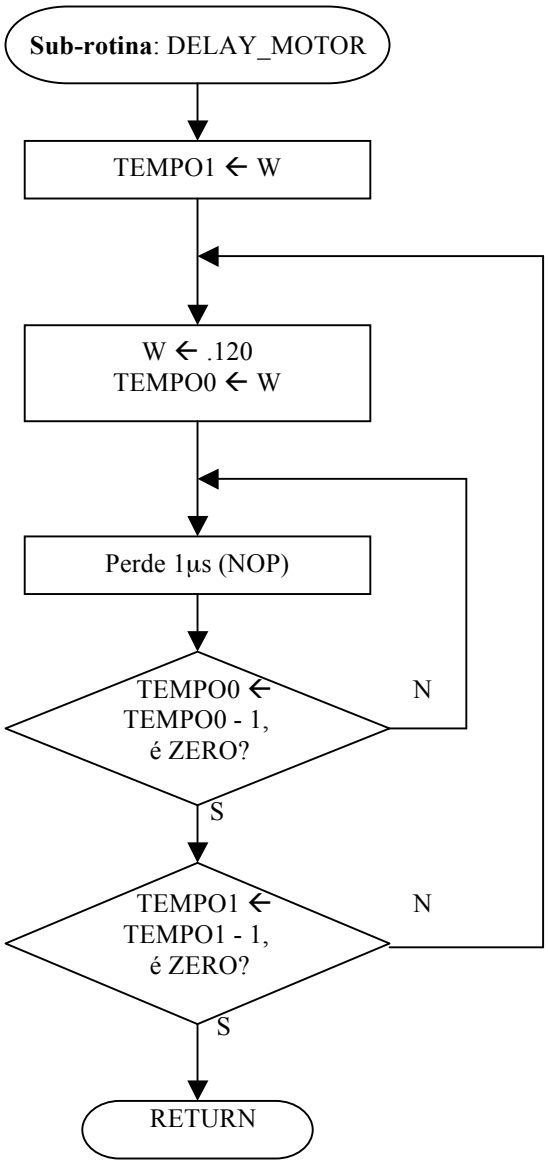


Figura A37 – Fluxograma da Sub-rotina: DELAY_MOTOR.

```
; ********************************************************************************
; *    ROTINA DE DELAY PARA ACIONAMENTO NECESSÁRIO DO MOTOR DE PASSO             *
; ********************************************************************************
DELAY_MOTOR                             ; W chega  com valor a ser carregado em TEMPO1
        MOVWF       TEMPO1              ; (1 T_CY) Carrega TEMPO1 (unidades de ms)
    ;-----------------------------------Loop2-------------------------------------------------------
        MOVLW       .120                ; (1 T_CY)
        MOVWF       TEMPO0              ; (1 T_CY) Carrega TEMPO0 (p/ contar 1ms)
        ;-----------------------------------Loop1-----------------------------------------------------
        NOP                             ; (1 T_CY)
        DECFSZ      TEMPO0,F            ; (1 ou 2 T_CY) Fim de TEMPO0 ?
        GOTO        $-2                 ; (2 T_CY)  Não - volta 2 instruções.
                                        ;             Sim – salta GOTO.
                                        ; 1+1+(1+1+2)*119+1+2 = 481 T_CY.
        ;-----------------------------------Fim Loop1-------------------------------------------------
        DECFSZ      TEMPO1,F            ; (1 ou 2 T_CY) Fim de TEMPO1 ?
        GOTO        $-6                 ; (2 T_CY)          Não - volta 6 instruções.
                                        ;                   Sim.
                                        ; [(481+1 +2)*TEMPO1]-1
    ;-----------------------------------------------Fim Loop2-------------------------------------------

        RETURN                          ; (2 T_CY) Retorna
                                        ; 1+ {[(481+1 +2)*TEMPO1]-1}+2 = 484*TEMPO1+2
                                        ; p/ TEMPO1 =1 → 486 T_CY
                                        ; p/ TEMPO1 =2 → 970 T_CY
                                        ; p/ TEMPO1 =3 → 1454 T_CY
; ********************************************************************************
```

b17) Sub-rotina DELAY_MS

A sub-rotina mostrada na Fig. A38 é para atrasar de n * ms (aproximadamente), entre um passo e outro do motor de passo, onde n é um valor em W antes de chamar esta sub-rotina.

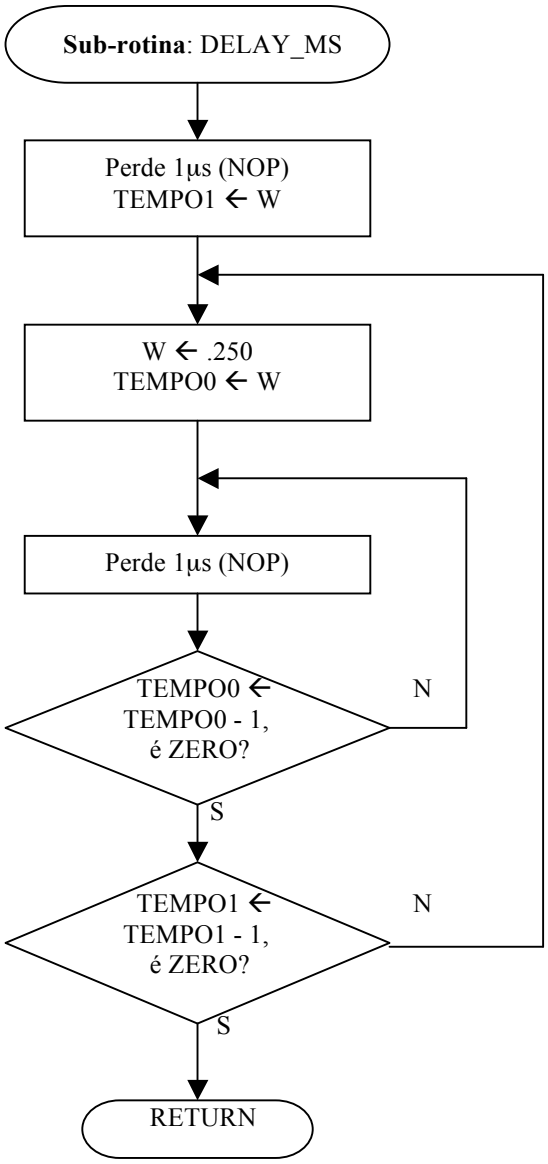


Figura A38 – Fluxograma da Sub-rotina: DELAY_MS.

```
; ********************************************************************************
; *                ROTINA DE DELAY (de 1ms até 255ms)                            *
; ********************************************************************************
DELAY_MS                                ; CALL perde 2 T_CY.
        NOP                             ; (1 T_CY)
        MOVWF   TEMPO1                  ; (1 T_CY) TEMPO1←W, ou carrega TEMPO1 (em ms)
   ;-----------------------------------Loop2--------------------------------------------
        MOVLW   .250                    ; (1 T_CY)
        MOVWF   TEMPO0                  ; (1 T_CY)Carrega TEMPO0 (p/ contar 1ms)
   ;-----------------------------------Loop1--------------------------------------------
        NOP                             ; (1 T_CY)
        DECFSZ  TEMPO0,F                ; (1 ou 2 T_CY) Fim do TEMPO0 ?
        GOTO    $-2                     ; (2 T_CY) Não – volta 2 instruções.
                                        ;          Sim – passou  1ms.
                                        ; 1+1+ [(1+1+2)*249+1+2]= 1001T_CY
   ;-------------------------------Fim loop1--------------------------------------------
        DECFSZ  TEMPO1,F                ; (1 ou 2 T_CY) Fim do  TEMPO1 ?
        GOTO    $-6                     ; (2 T_CY) Não - volta 6 instruções
                                        ;          Sim.
                                        ; (1001+1+2)*TEMPO1+2=Loop2
                                        ; [1004*TEMPO1] + 2 (T_CY)=Loop2
    ;-------------------------------Fim Loop2--------------------------------------------
        RETURN                          ; (2 T_CY) RETORNA
                                        ; 2 (call) + 1(nop)+1(movwf) + 2(return) +Loop2
                                        ; 6T_CY + Loop2
                                        ; se TEMPO1 = 1 → 6+1006 = 1012T_CY = 1,012ms
; ********************************************************************************
```

b18) Sub-rotina DELAY_3S

Esta sub-rotina, Fig. A39, um promove atraso de 3s (aproximadamente).

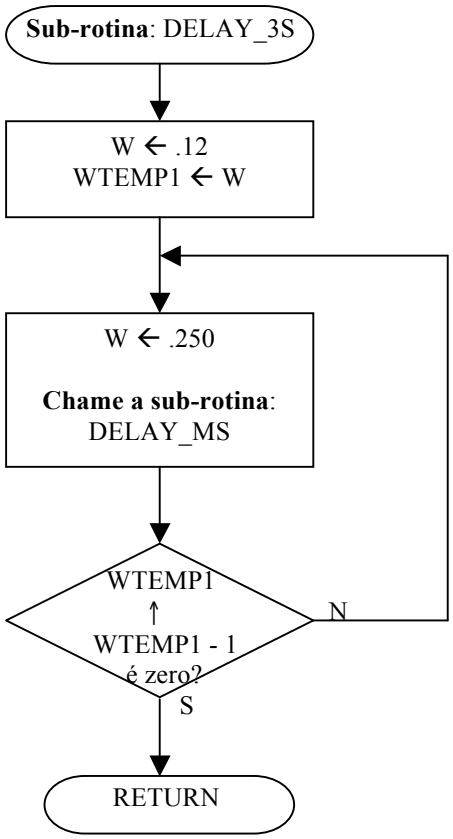


Figura A39 – Fluxograma da Sub-rotina: DELAY_3S.

```
; ********************************************************************************
; Rotina que atrasa aproximadamente 3 segundos.                                  *
; ********************************************************************************
DELAY_3S
        MOVLW       .12                 ; W ← .12
        MOVWF       WTEMP1              ; WTEMP1 ← W [ou carregue-o com 12]
Mais_250ms_a_3s
        MOVLW       .250
        CALL        DELAY_MS            ; DELAY DE 250ms
        DECFSZ      WTEMP1,F
        GOTO        Mais_250ms_a_3s
        RETURN
; ********************************************************************************
```

b19) Sub-rotina DELAY_5S

Esta sub-rotina, Fig. A40, promove um atraso de 5s (aproximadamente).

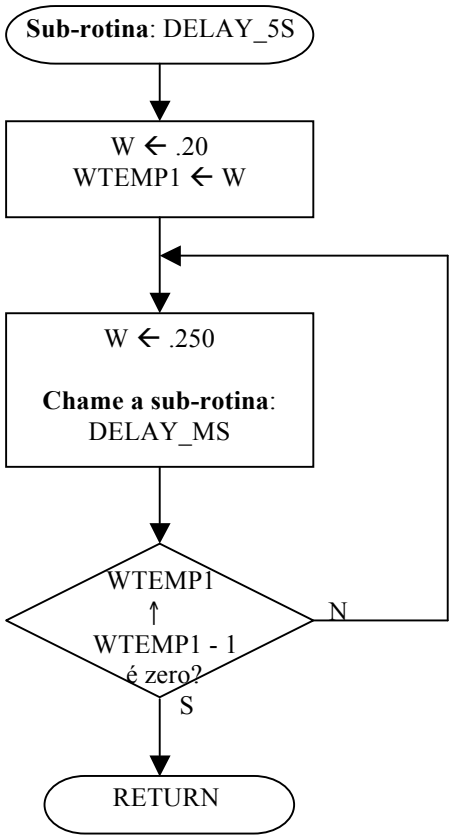


Figura A40 – Fluxograma da Sub-rotina: DELAY_5S.

```
; ********************************************************************************
; Rotina de aproximadamente 5 segundos.                                          *
; ********************************************************************************
DELAY_5S
        MOVLW       .20                 ; W ← .20
        MOVWF       WTEMP1              ; WTEMP1 ← W [ou carregue-o com 20]
Mais_250ms_a_5s
        MOVLW       .250
        CALL        DELAY_MS            ; DELAY DE 250ms
        DECFSZ      WTEMP1,F
        GOTO        Mais_250ms_a_5s
        RETURN
; ********************************************************************************
```

b20) Sub-rotina UM_MM_X

Esta sub-rotina, Fig. A41, controla o movimento da mesa por 1 mm (no sentido horário).

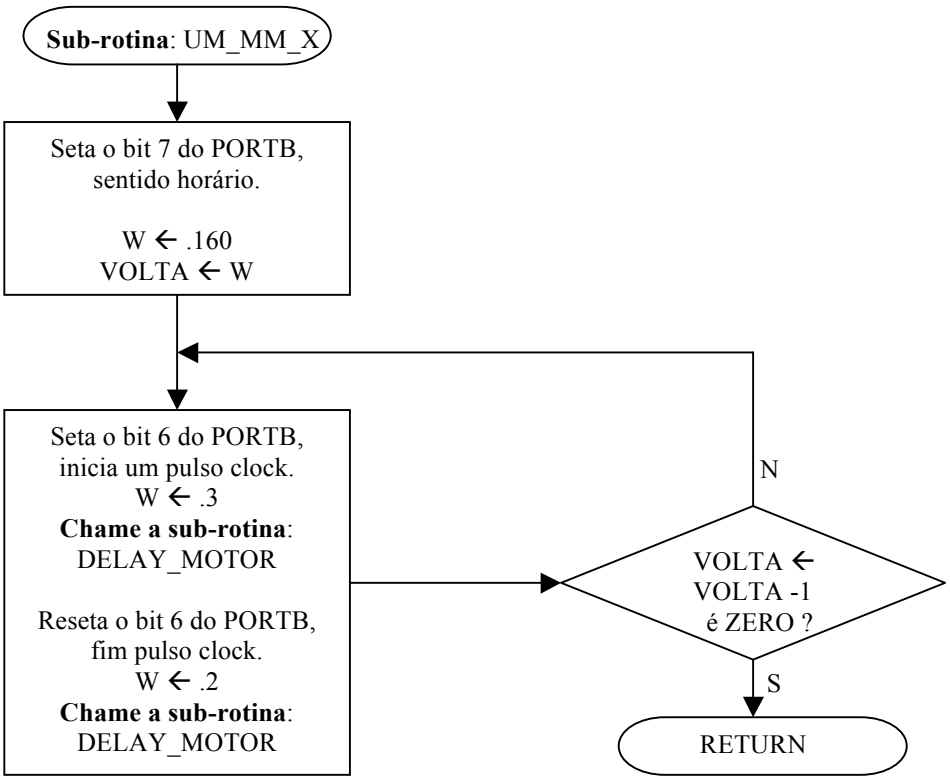


Figura A41 – Fluxograma da Sub-rotina: UM_MM_X.

```
; ********************************************************************************
; *    Esta rotina controla o motor de passo para mover a mesa 1 mm no sentido horário                *
; ********************************************************************************
UM_MM_X
        BSF             Direcao_0a41            ; Reseta Direção para girar sentido anti-horário o MP.
        MOVLW           .160                    ; carrega nº de passos necessários mover mesa 1 mm.
        MOVWF           VOLTA
NOVO_PASSO_X
        BSF             Clock_0a41              ; seta bit 6 PORTB, inicia um pulso clock p/ um passo.
        MOVLW           .3
        CALL            DELAY_MOTOR             ; DELAY Motor para clock setado
        BCF             Clock_0a41
        MOVLW           .2
        CALL            DELAY_MOTOR             ; DELAY Motor para clock resetado
        DECFSZ          VOLTA,F                 ; VOLTA←VOLTA-1, qdo zero salta próxima instrução.
        GOTO            NOVO_PASSO_X
        RETURN
; ********************************************************************************
```

b21) Sub-rotina UM_MM_XA

Esta sub-rotina, Fig. A42, controla o movimento da mesa por 1 mm (no sentido anti-horário).

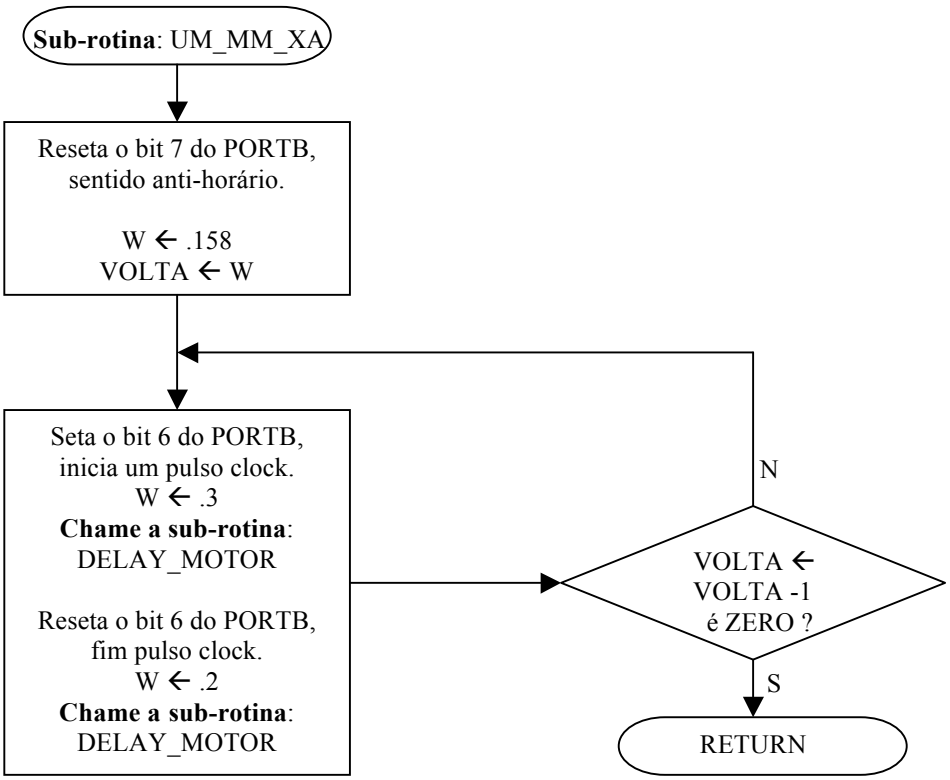


Figura A42 – Fluxograma da Sub-rotina: UM_MM_XA

```
; ********************************************************************************
; *    Rotina para o motor de passo mover a mesa 1 mm no sentido anti-horário    *
; ********************************************************************************
UM_MM_XA
        BCF         Direcao_0a41        ; Reseta Direção para girar sentido anti-horário o MP.
        MOVLW       .158                ; carrega nº de passos necessários mover mesa 1 mm.
        MOVWF       VOLTA
NOVO_PASSO_XA
        BSF         Clock_0a41          ; seta bit 6 PORTB, inicia um pulso clock p/ um passo.
        MOVLW       .3
        CALL        DELAY_MOTOR         ; DELAY Motor para clock setado
        BCF         Clock_0a41
        MOVLW       .2
        CALL        DELAY_MOTOR         ; DELAY Motor para clock resetado
        DECFSZ      VOLTA,F             ; VOLTA←VOLTA-1, qdo zero salta próxima instrução.
        GOTO        NOVO_PASSO_XA
        RETURN
; ********************************************************************************
```

b22) Sub-rotina UM_MM_YA

Esta sub-rotina, Fig. A43, controla o movimento da mesa por 1 mm (no sentido anti-horário).

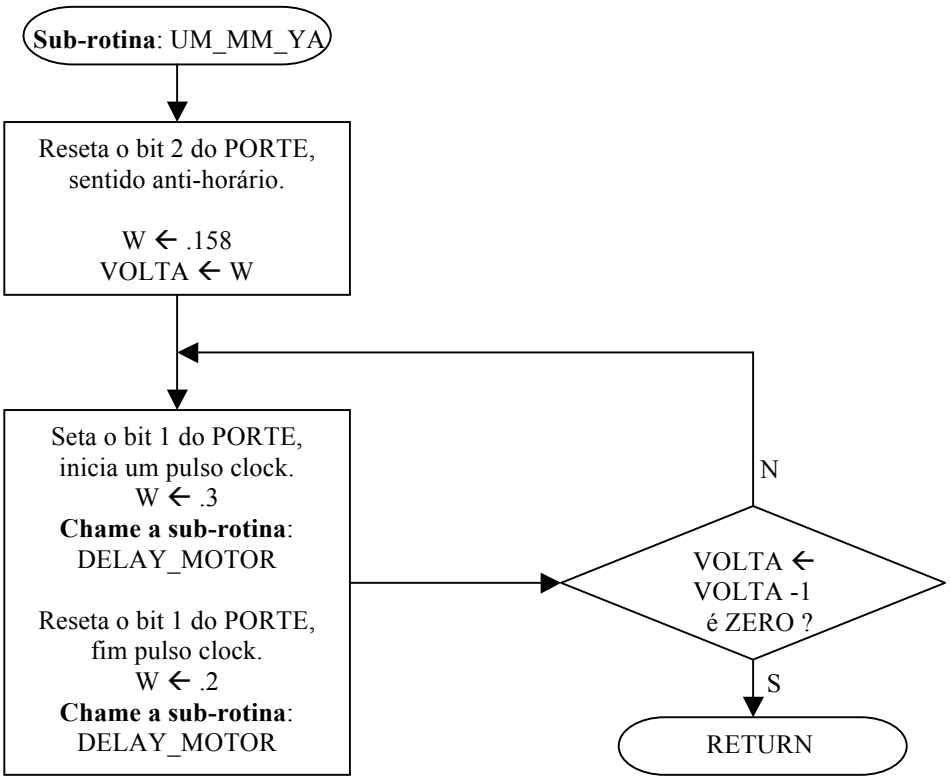


Figura A43 – Fluxograma da Sub-rotina: UM_MM_YA

```
; ********************************************************************************
; *    Rotina para o motor de passo mover a mesa 1 mm no sentido anti-horário    *
; ********************************************************************************
UM_MM_YA
        BCF         Direcao_0a90        ; Reseta Direção para girar sentido anti-horário o MP.
        MOVLW       .158                ; carrega nº de passos necessários mover mesa 1 mm.
        MOVWF       VOLTA
NOVO_PASSO_Y
        BSF         Clock_0a90          ; seta bit 6 PORTB, inicia um pulso clock p/ um passo.
        MOVLW       .3
        CALL        DELAY_MOTOR         ; DELAY Motor para clock setado
        BCF         Clock_0a90
        MOVLW       .2
        CALL        DELAY_MOTOR         ; DELAY Motor para clock resetado
        DECFSZ      VOLTA,F             ; VOLTA←VOLTA-1, qdo zero salta próxima instrução.
        GOTO        NOVO_PASSO_Y
        RETURN
; ********************************************************************************
```

b23) Sub-rotina UM_CM_Y

Esta sub-rotina, Fig. A44, controla o movimento da mesa Y por 1 cm (no sentido horário).

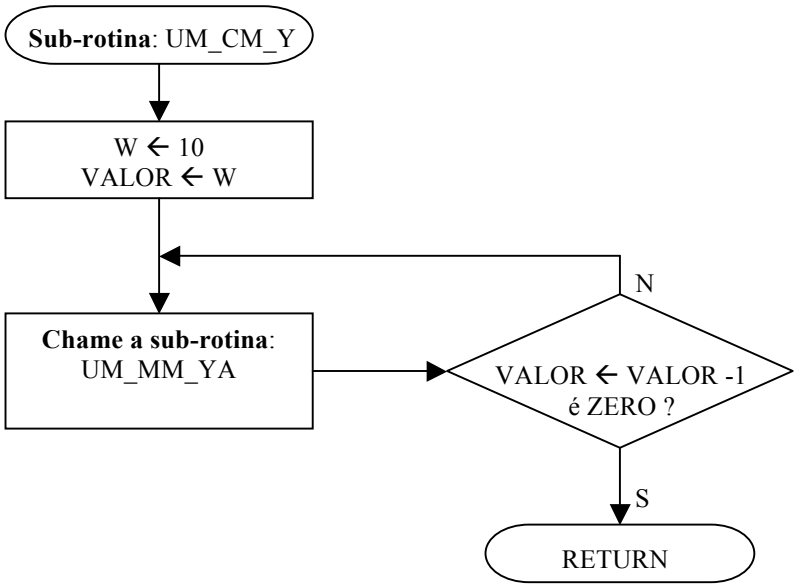


Figura A44 – Fluxograma da Sub-rotina: UM_CM_Y

```
; ********************************************************************************
; *    Rotina para o motor de passo mover a mesa (Y) 1 cm no sentido horário                    *
; ********************************************************************************
UM_CM_Y
        MOVLW       .10                 ; W ← 10
        MOVWF       VALOR               ; VALOR ← W
MAIS_1mm_Y
        CALL        UM_MM_YA            ; DELAY Motor para clock resetado
        DECFSZ      VALOR,F             ; VOLTA←VOLTA-1, qdo zero salta próxima instrução.
        GOTO        MAIS_1mm_Y
        RETURN
; ********************************************************************************
```

b24) Sub-rotina UM_CM_X

Esta sub-rotina, Fig. A45, controla o movimento da mesa X por 1 cm (no sentido horário).

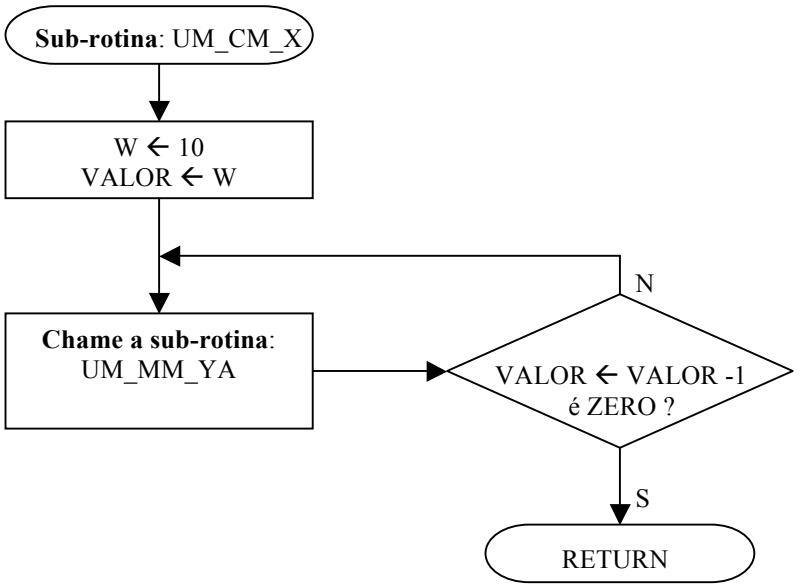


Figura A45 – Fluxograma da Sub-rotina: UM_CM_X

```
; ********************************************************************************
; *    Rotina para o motor de passo mover a mesa (X) 1 cm no sentido horário     *
; ********************************************************************************
UM_CM_X
        MOVLW       .10                 ; W ← 10
        MOVWF       VALOR               ; VALOR ← W
MAIS_1mm_X
        CALL        UM_MM_XA            ; DELAY Motor para clock resetado
        DECFSZ      VALOR,F             ; VOLTA←VOLTA-1, qdo zero salta próxima instrução.
        GOTO        MAIS_1mm_X
        RETURN
; ********************************************************************************
```

b25) Sub-rotina UMA_VOLTA_CABECA

Esta sub-rotina, Fig. A46, controla o movimento da cabeça (onde reside a fonte radioativa) para que ela dê uma volta completa (anti-horário).

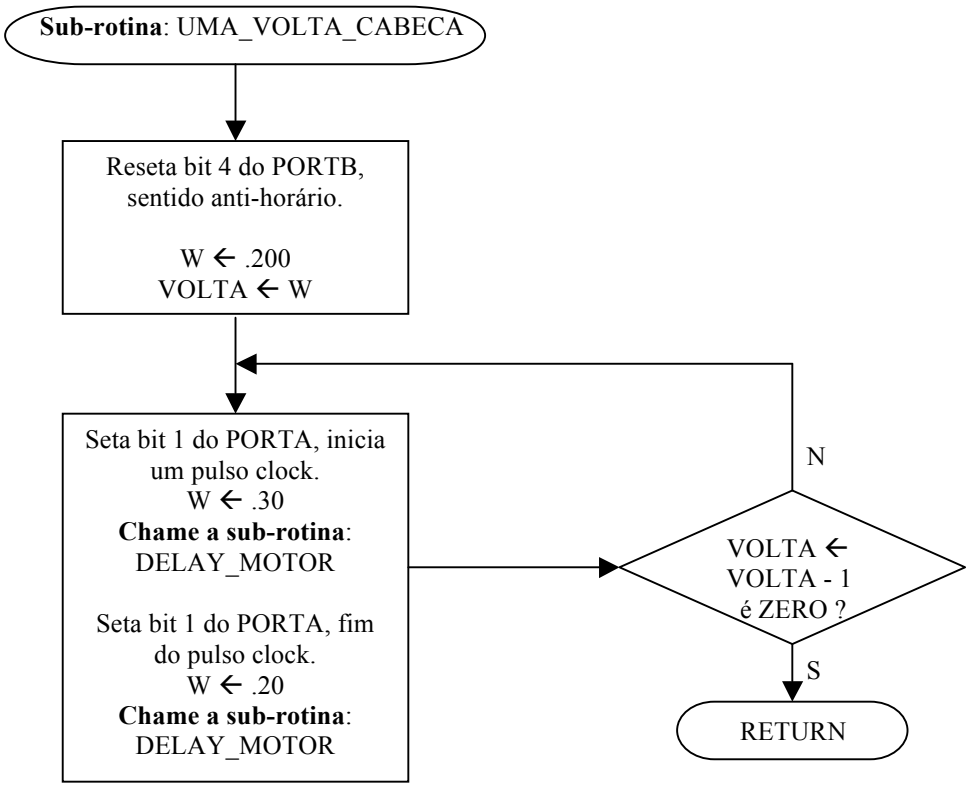


Figura A46 – Fluxograma da Sub-rotina: UMA_VOLTA_CABECA

```
; ********************************************************************************
; *    Rotina para o motor de passo CABECA girar uma volta completa no sentido anti-horário      *
; ********************************************************************************
UMA_VOLTA_CABECA
        BCF         Direcao_cabeca      ; Reseta Direção para girar sentido anti-horário o MP.
        MOVLW       .200                ; carrega número de passos necessários para uma volta.
        MOVWF       VOLTA
PASSO_1
        BSF         Clock_cabeca        ; seta bit 1 PORTE, inicia pulso de clock para um passo.
        MOVLW       .30
        CALL        DELAY_MOTOR         ; DELAY Motor para clock setado
        BCF         Clock_cabeca
        MOVLW       .20
        CALL        DELAY_MOTOR         ; DELAY Motor para clock resetado
        DECFSZ      VOLTA,F             ; VOLTA ← VOLTA-1, se zero salta próxima instrução.
        GOTO        PASSO_1
        RETURN
; ********************************************************************************
```

b26) Sub-rotina ESCREVE

Esta sub-rotina, Fig. A47, escreve um caractere no display LCD.

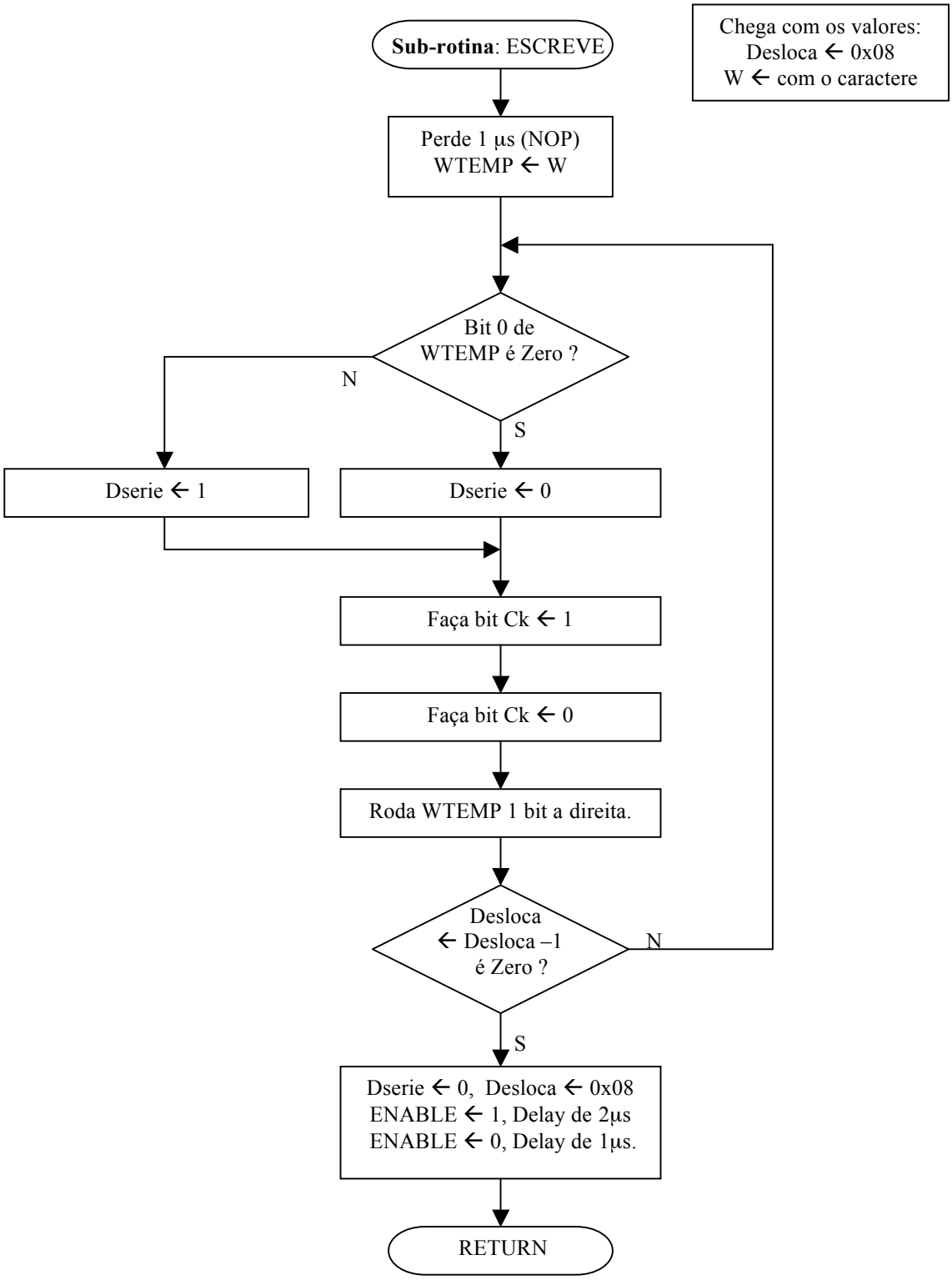


Figura A47 – Fluxograma da Sub-rotina: ESCREVE

```
; ********************************************************************************
; *          ROTINA DE ESCRITA DE UM CARACTER NO DISPLAY                         *
; ********************************************************************************
;  ESTA ROTINA ENVIA UM CARACTER PARA O MÓDULO DE LCD. O CARACTER A SER
;  ESCRITO DEVE SER COLOCADO EM WORK (W) ANTES DE CHAMAR ESTA ROTINA.

ESCREVE
        NOP
        MOVWF       WTEMP               ; WTEMP ← W
Write
        BTFSC       WTEMP,0             ; Testa o bit 0 do W e salta próx. .inst. se ZERO
        GOTO        UM
        BCf         Dserie
        BSF         Ck                  ; clock para shifitar um bit
        BCF         Ck
        GOTO        CONTINUA
UM
        BSf         Dserie
        BSF         Ck                  ; clock para shifitar um bit
        BCF         Ck
CONTINUA
        RRF         WTEMP,F             ; shifitar um bit à direita.
        NOP
        DECFSZ      Desloca,F           ; Desloca, inicialmente é 08H
        GOTO        Write
        BCF         Dserie
        MOVLW       0X08
        MOVWF       Desloca             ; quanto se deslocará à direita o byte.
        BSF         ENABLE              ; ENVIA UM PULSO DE ENABLE AO DISPLAY
        GOTO        $+1                 ; atrasa 2μs.
        BCF         ENABLE              ; o aceite da informação é na descida ENABLE
        NOP                             ; atrasa 1μs.
        RETURN                          ; RETORNA
; ********************************************************************************
```

b27) Sub-rotina LCD_LINHA_1

Esta sub-rotina, Fig. A48, escreve 16 caracteres na Linha 1 do display LCD.

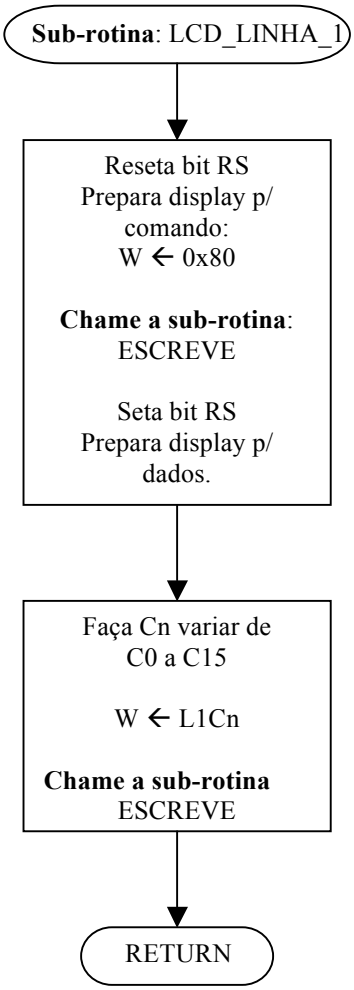


Figura A48 – Fluxograma da Sub-rotina: LCD_LINHA_1

```
; ********************************************************************************
; *            Rotina para se escrever na linha 1 do LCD                         *
; ********************************************************************************
;  Esta rotina escreve na linha 1 do LCD o seguinte: "END.(HEXA) DADO"

LCD_LINHA_1
        BCF         RS              ; SELECIONA O DISPLAY P/ COMANDOS
        MOVLW       0X80            ; COMANDO PARA POSICIONAR O CURSOR
        CALL        ESCREVE         ; LINHA 1 / COLUNA 0
        BSF         RS              ; SELECIONA O DISPLAY P/ DADOS
                                    ; Comandos para escrever o texto desejado
                                    ; na linha 1 do display LCD.
        MOVF        L1_C0,W         ; W ← L1_C0
        CALL        ESCREVE
        MOVF        L1_C1,W         ; W ← L1_C1
        CALL        ESCREVE
        MOVF        L1_C2,W         ; W ← L1_C2
        CALL        ESCREVE
        MOVF        L1_C3,W         ; W ← L1_C3
        CALL        ESCREVE
        MOVF        L1_C4,W         ; W ← L1_C4
        CALL        ESCREVE
        MOVF        L1_C5,W         ; W ← L1_C5
        CALL        ESCREVE
        MOVF        L1_C6,W         ; W ← L1_C6
        CALL        ESCREVE
        MOVF        L1_C7,W         ; W ← L1_C7
        CALL        ESCREVE
        MOVF        L1_C8,W         ; W ← L1_C8
        CALL        ESCREVE
        MOVF        L1_C9,W         ; W ← L1_C9
        CALL        ESCREVE
        MOVF        L1_C10,W        ; W ← L1_C10
        CALL        ESCREVE
        MOVF        L1_C11,W        ; W ← L1_C11
        CALL        ESCREVE
        MOVF        L1_C12,W        ; W ← L1_C12
        CALL        ESCREVE
        MOVF        L1_C13,W        ; W ← L1_C13
        CALL        ESCREVE
        MOVF        L1_C14,W        ; W ← L1_C14
        CALL        ESCREVE
        MOVF        L1_C15,W        ; W ← L1_C15
        CALL        ESCREVE
        RETURN                      ; RETORNA
```

b28) Sub-rotina LCD_LINHA_2

Esta sub-rotina, Fig. A49, escreve 16 caracteres na Linha 2 do display LCD.

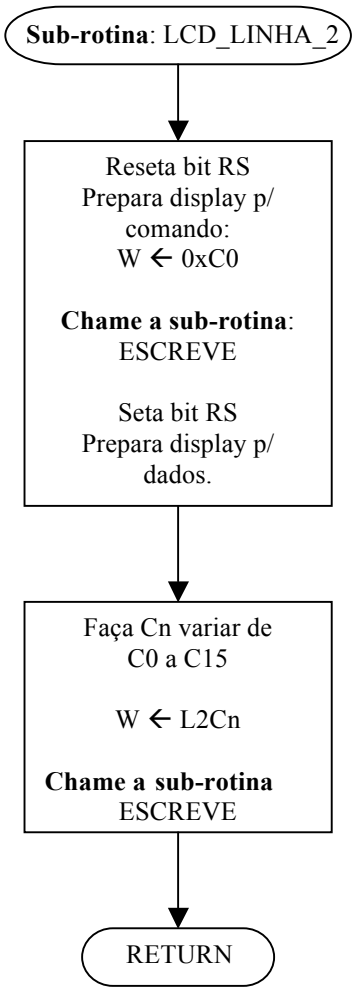


Figura A49 – Fluxograma da Sub-rotina: LCD_LINHA_2

```
; ********************************************************************************
; *            Rotina para se escrever na linha 2 do LCD                         *
; ********************************************************************************
;  Esta rotina escreve na linha 2 do LCD o seguinte: "END.(HEXA) DADO"

LCD_LINHA_2
        BCF             RS                      ; SELECIONA O DISPLAY P/ COMANDO
        MOVLW           0XC0                    ; COMANDO PARA POSICIONAR O CURSOR
        CALL            ESCREVE                 ; LINHA 2 / COLUNA 0\
        BSF             RS                      ; SELECIONA O DISPLAY P/ DADOS
                                                ; Comandos para escrever o texto desejado
                                                ; na linha 2 do display LCD.
        MOVF            L2_C0,W                 ; W ← L2_C0
        CALL            ESCREVE
        MOVF            L2_C1,W                 ; W ← L2_C1
        CALL            ESCREVE
        MOVF            L2_C2,W                 ; W ← L2_C2
        CALL            ESCREVE
        MOVF            L2_C3,W                 ; W ← L2_C3
        CALL            ESCREVE
        MOVF            L2_C4,W                 ; W ← L2_C4
        CALL            ESCREVE
        MOVF            L2_C5,W                 ; W ← L2_C5
        CALL            ESCREVE
        MOVF            L2_C6,W                 ; W ← L2_C6
        CALL            ESCREVE
        MOVF            L2_C7,W                 ; W ← L2_C7
        CALL            ESCREVE
        MOVF            L2_C8,W                 ; W ← L2_C8
        CALL            ESCREVE
        MOVF            L2_C9,W                 ; W ← L2_C9
        CALL            ESCREVE
        MOVF            L2_C10,W                ; W ← L2_C10
        CALL            ESCREVE
        MOVF            L2_C11,W                ; W ← L2_C11
        CALL            ESCREVE
        MOVF            L2_C12,W                ; W ← L2_C12
        CALL            ESCREVE
        MOVF            L2_C13,W                ; W ← L2_C13
        CALL            ESCREVE
        MOVF            L2_C14,W                ; W ← L2_C14
        CALL            ESCREVE
        MOVF            L2_C15,W                ; W ← L2_C15
        CALL            ESCREVE
        RETURN                                  ; RETORNA
; ********************************************************************************
```

b29) Sub-rotinas Mens_1; Mens_2 e Mens_3

Estas sub-rotinas, Fig. A50, escrevem suas mensagens no display LCD.

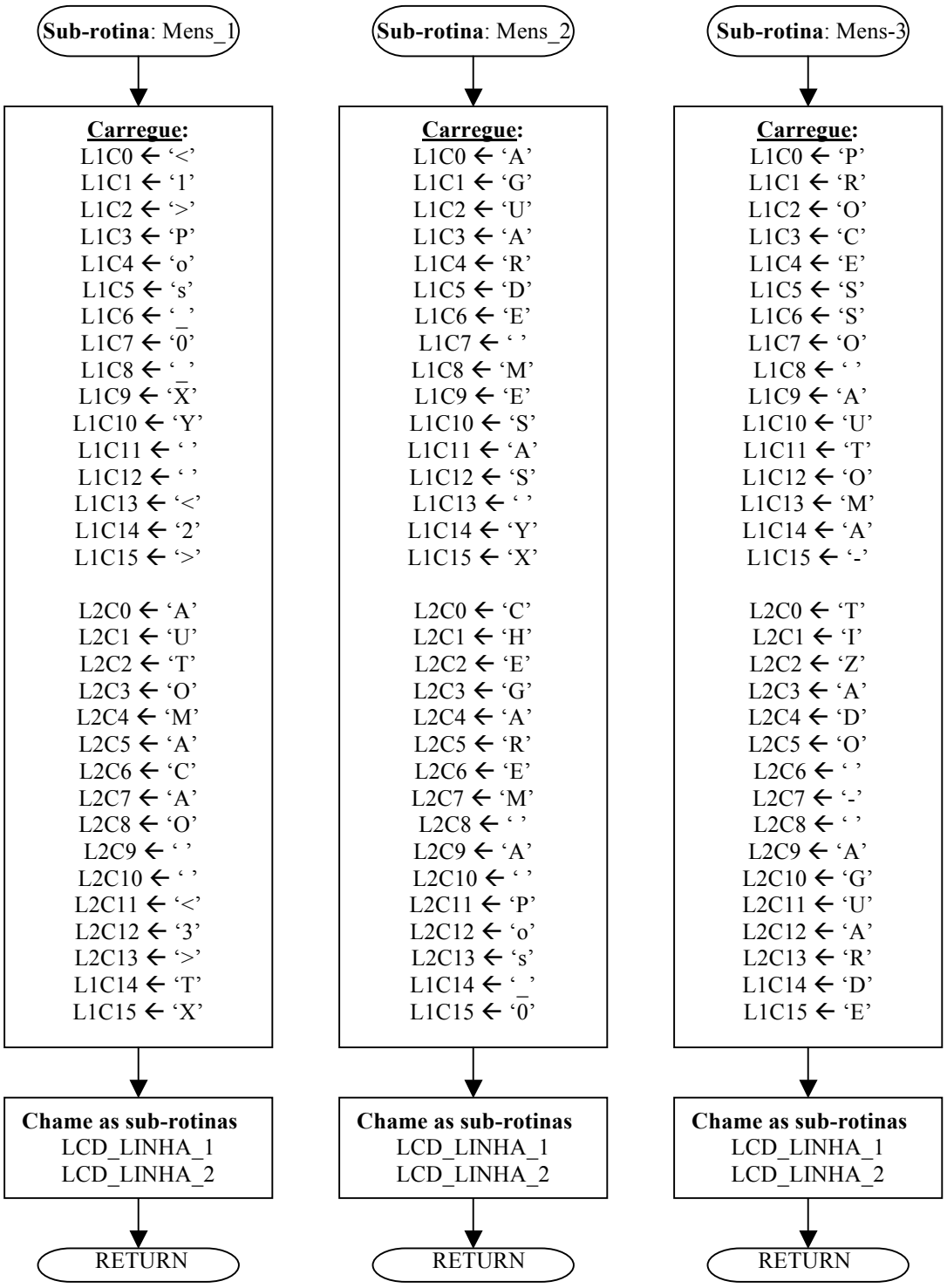


Figura A50 – Fluxograma das Sub-rotinas: Mens_1, Mens_2 e Mens_3

```
; ********************************************************************************************
; *          Esta rotina fornece os dados para a Mens_1 a ser enviada ao LCD                 *
; ********************************************************************************************
Mens_1
                                                ; <1> Pos_0_XY,
                                                ; acionada desloca rápido a mesa X e Y
                                                ; para posição zero
        MOVLW       '<'
        MOVWF       L1_C0
        MOVLW       '1'
        MOVWF       L1_C1
        MOVLW       '>'
        MOVWF       L1_C2
        MOVLW       'P'
        MOVWF       L1_C3
        MOVLW       'o'
        MOVWF       L1_C4
        MOVLW       's'
        MOVWF       L1_C5
        MOVLW       '-'
        MOVWF       L1_C6
        MOVLW       '0'
        MOVWF       L1_C7
        MOVLW       '_'
        MOVWF       L1_C8
        MOVLW       'X'
        MOVWF       L1_C9
        MOVLW       'Y'
        MOVWF       L1_C10
        MOVLW       ' '
        MOVWF       L1_C11
        MOVLW       ' '
        MOVWF       L1_C12
        MOVLW       '<'
        MOVWF       L1_C13
        MOVLW       '2'
        MOVWF       L1_C14
        MOVLW       '>'
        MOVWF       L1_C15
                                                ; <2> AUTOMACAO,
                                                ; acionada inicia o processo de automação.
                                                ; <3> TX, Transmissão ao CP via RS-232.
        MOVLW       'A'
        MOVWF       L2_C0
        MOVLW       'U'
        MOVWF       L2_C1
        MOVLW       'T'
        MOVWF       L2_C2
        MOVLW       'O'
        MOVWF       L2_C3
        MOVLW       'M'
        MOVWF       L2_C4
        MOVLW       'A'
        MOVWF       L2_C5
        MOVLW       'C'
```

```
        MOVWF   L2_C6
        MOVLW   'A'
        MOVWF   L2_C7
        MOVLW   'O'
        MOVWF   L2_C8
        MOVLW   ' '
        MOVWF   L2_C9
        MOVLW   ' '
        MOVWF   L2_C10
        MOVLW   '<'
        MOVWF   L2_C11
        MOVLW   '3'
        MOVWF   L2_C12
        MOVLW   '>'
        MOVWF   L2_C13
        MOVLW   'T'
        MOVWF   L2_C14
        MOVLW   'X'
        MOVWF   L2_C15

        CALL    LCD_LINHA_1
        CALL    LCD_LINHA_2

        RETURN
; ********************************************************************************************


; ********************************************************************************************
; *       Esta rotina fornece os dados para a Mens_2 a ser enviada ao LCD                    *
; ********************************************************************************************
Mens_2
                                                ; AGUARDE MESAS YX
        MOVLW   'A'
        MOVWF   L1_C0
        MOVLW   'G'
        MOVWF   L1_C1
        MOVLW   'U'
        MOVWF   L1_C2
        MOVLW   'A'
        MOVWF   L1_C3
        MOVLW   'R'
        MOVWF   L1_C4
        MOVLW   'D'
        MOVWF   L1_C5
        MOVLW   'E'
        MOVWF   L1_C6
        MOVLW   ' '
        MOVWF   L1_C7
        MOVLW   'M'
        MOVWF   L1_C8
        MOVLW   'E'
        MOVWF   L1_C9
        MOVLW   'S'
        MOVWF   L1_C10
        MOVLW   'A'
        MOVWF   L1_C11
```

```
        MOVLW       'S'
        MOVWF       L1_C12
        MOVLW       ' '
        MOVWF       L1_C13
        MOVLW       'Y'
        MOVWF       L1_C14
        MOVLW       'X'
        MOVWF       L1_C15
                                                    ; CHEGAREM A Pos_0
        MOVLW       'C'
        MOVWF       L2_C0
        MOVLW       'H'
        MOVWF       L2_C1
        MOVLW       'E'
        MOVWF       L2_C2
        MOVLW       'G'
        MOVWF       L2_C3
        MOVLW       'A'
        MOVWF       L2_C4
        MOVLW       'R'
        MOVWF       L2_C5
        MOVLW       'E'
        MOVWF       L2_C6
        MOVLW       'M'
        MOVWF       L2_C7
        MOVLW       ' '
        MOVWF       L2_C8
        MOVLW       'A'
        MOVWF       L2_C9
        MOVLW       ' '
        MOVWF       L2_C10
        MOVLW       'P'
        MOVWF       L2_C11
        MOVLW       'o'
        MOVWF       L2_C12
        MOVLW       's'
        MOVWF       L2_C13
        MOVLW       '_'
        MOVWF       L2_C14
        MOVLW       '0'
        MOVWF       L2_C15

        CALL        LCD_LINHA_1
        CALL        LCD_LINHA_2

        RETURN
; ********************************************************************************************


; ********************************************************************************************
; *           Esta rotina fornece os dados para a Mens_3 a ser enviada ao LCD                *
; ********************************************************************************************
Mens_3
                                                    ; PROCESSO AUTOMA-
        MOVLW       'P'
        MOVWF       L1_C0
```

```
MOVLW   'R'
MOVWF   L1_C1
MOVLW   'O'
MOVWF   L1_C2
MOVLW   'C'
MOVWF   L1_C3
MOVLW   'E'
MOVWF   L1_C4
MOVLW   'S'
MOVWF   L1_C5
MOVLW   'S'
MOVWF   L1_C6
MOVLW   'O'
MOVWF   L1_C7
MOVLW   ' '
MOVWF   L1_C8
MOVLW   'A'
MOVWF   L1_C9
MOVLW   'U'
MOVWF   L1_C10
MOVLW   'T'
MOVWF   L1_C11
MOVLW   'O'
MOVWF   L1_C12
MOVLW   'M'
MOVWF   L1_C13
MOVLW   'A'
MOVWF   L1_C14
MOVLW   '-'
MOVWF   L1_C15
                                        ; TIZADO - AGUARDE
MOVLW   'T'
MOVWF   L2_C0
MOVLW   'I'
MOVWF   L2_C1
MOVLW   'Z'
MOVWF   L2_C2
MOVLW   'A'
MOVWF   L2_C3
MOVLW   'D'
MOVWF   L2_C4
MOVLW   'O'
MOVWF   L2_C5
MOVLW   ' '
MOVWF   L2_C6
MOVLW   '-'
MOVWF   L2_C7
MOVLW   ' '
MOVWF   L2_C8
MOVLW   'A'
MOVWF   L2_C9
MOVLW   'G'
MOVWF   L2_C10
MOVLW   'U'
MOVWF   L2_C11
MOVLW   'A'
```

```
        MOVWF       L2_C12
        MOVLW       'R'
        MOVWF       L2_C13
        MOVLW       'D'
        MOVWF       L2_C14
        MOVLW       'E'
        MOVWF       L2_C15

        CALL        LCD_LINHA_1
        CALL        LCD_LINHA_2
        RETURN
; ********************************************************************************
```

b30) Sub-rotinas Mens_4; Mens_5 e Mens_6

Estas sub-rotinas, Fig. A51, escrevem suas mensagens no display LCD.

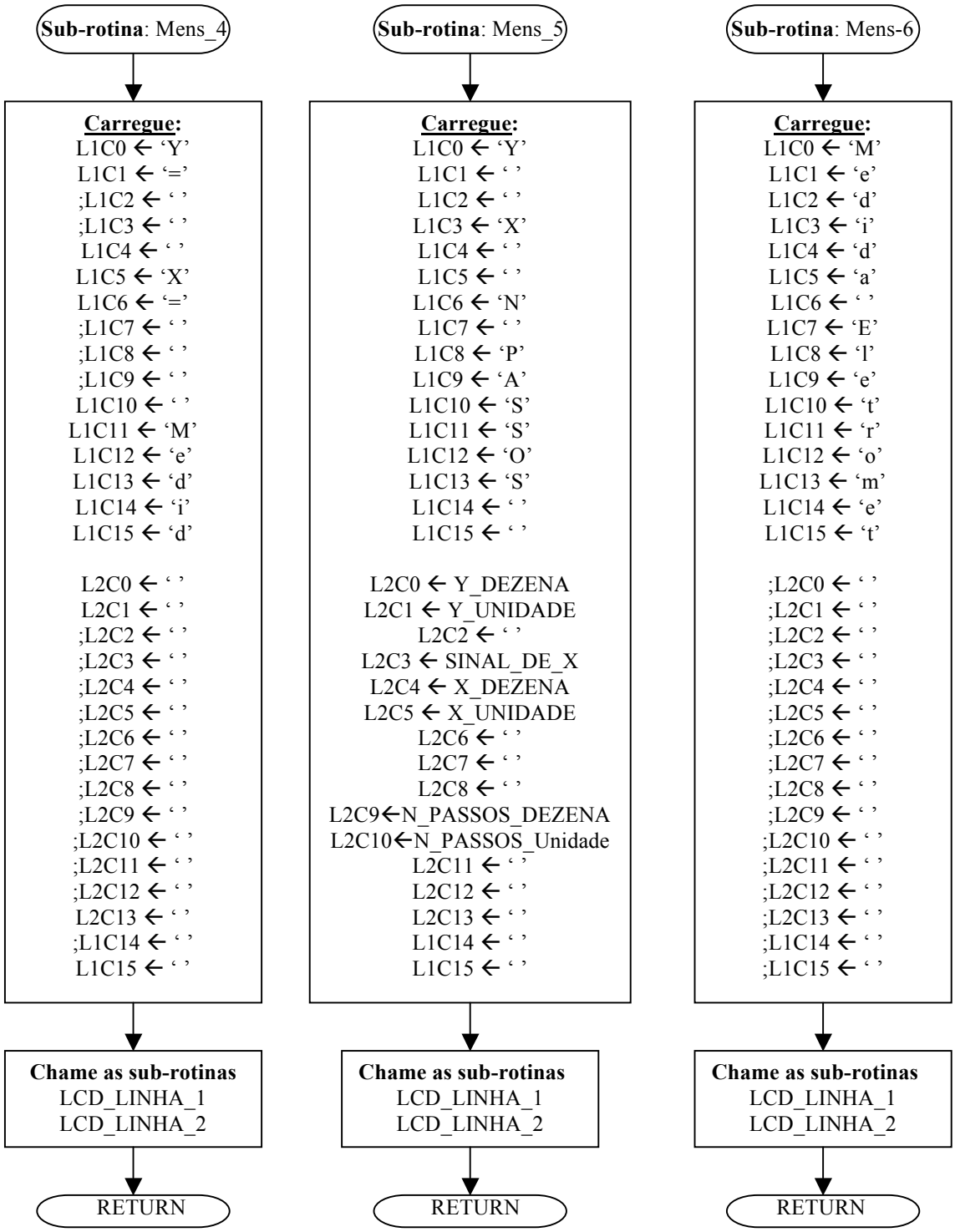


Figura A51 – Fluxograma das Sub-rotinas: Mens_4, Mens_5 e Mens_6

```
; ********************************************************************************************
; *            Esta rotina fornece os dados para a Mens_4 a ser enviada ao LCD                 *
; ********************************************************************************************
Mens_4
                                                    ; Y=90bX=+20bMedid
                                                    ; bb+0,1973E+19bCb
        MOVLW       'Y'
        MOVWF       L1_C0
        MOVLW       '='
        MOVWF       L1_C1
        ;MOVLW      ' '
        ;MOVWF      L1_C2
        ;MOVLW      ' '
        ;MOVWF      L1_C3
        MOVLW       ' '
        MOVWF       L1_C4
        MOVLW       'X'
        MOVWF       L1_C5
        MOVLW       '='
        MOVWF       L1_C6
        ;MOVLW      ' '
        ;MOVWF      L1_C7
        ;MOVLW      ' '
        ;MOVWF      L1_C8
        ;MOVLW      ' '
        ;MOVWF      L1_C9
        MOVLW       ' '
        MOVWF       L1_C10
        MOVLW       'M'
        MOVWF       L1_C11
        MOVLW       'e'
        MOVWF       L1_C12
        MOVLW       'd'
        MOVWF       L1_C13
        MOVLW       'i'
        MOVWF       L1_C14
        MOVLW       'd'
        MOVWF       L1_C15
                                                    ; Y=90bX=+20bMedid
                                                    ; bb+0,1973E+19bCb
        MOVLW       ' '
        MOVWF       L2_C0
        MOVLW       ' '
        MOVWF       L2_C1
        ;MOVLW      ' '
        ;MOVWF      L2_C2
        ;MOVLW      ' '
        ;MOVWF      L2_C3
        ;MOVLW      ' '
        ;MOVWF      L2_C4
        ;MOVLW      ' '
        ;MOVWF      L2_C5
        ;MOVLW      ' '
        ;MOVWF      L2_C6
        ;MOVLW      ' '
```

```
        ;MOVWF      L2_C7
        ;MOVLW      ' '
        ;MOVWF      L2_C8
        ;MOVLW      ' '
        ;MOVWF      L2_C9
        ;MOVLW      ' '
        ;MOVWF      L2_C10
        ;MOVLW      ' '
        ;MOVWF      L2_C11
        ;MOVLW      ' '
        ;MOVWF      L2_C12
        MOVLW       ' '
        MOVWF       L2_C13
        ;MOVLW      ' '
        ;MOVWF      L2_C14
        MOVLW       ' '
        MOVWF       L2_C15

        CALL        LCD_LINHA_1
        CALL        LCD_LINHA_2
        RETURN                                  ; RETORNA
; ********************************************************************************************


; ********************************************************************************************
; *           Esta rotina fornece os dados para a Mens_5 a ser enviada ao LCD                *
; ********************************************************************************************
Mens_5
                                                ; Y  X  N_Passos
                                                ; ** **   **
        MOVLW       'Y'
        MOVWF       L1_C0
        MOVLW       ' '
        MOVWF       L1_C1
        MOVLW       ' '
        MOVWF       L1_C2
        MOVLW       'X'
        MOVWF       L1_C3
        MOVLW       ' '
        MOVWF       L1_C4
        MOVLW       ' '
        MOVWF       L1_C5
        MOVLW       'N'
        MOVWF       L1_C6
        MOVLW       '_'
        MOVWF       L1_C7
        MOVLW       'P'
        MOVWF       L1_C8
        MOVLW       'A'
        MOVWF       L1_C9
        MOVLW       'S'
        MOVWF       L1_C10
        MOVLW       'S'
        MOVWF       L1_C11
        MOVLW       'O'
        MOVWF       L1_C12
```

```
        MOVLW       'S'
        MOVWF       L1_C13
        MOVLW       ' '
        MOVWF       L1_C14
        MOVLW       ' '
        MOVWF       L1_C15
                                            ; Y  X  N_Passos
                                            ; ** **    **

        MOVF        Y_DEZENA,W              ; W ← Y_DEZENA
        MOVWF       L2_C0                   ; L2_C0 ← W
        MOVF        Y_UNIDADE,W             ; W ← Y_UNIDADE
        MOVWF       L2_C1                   ; L2_C1 ← W
        MOVLW       ' '
        MOVWF       L2_C2
        MOVF        SINAL_DE_X,W            ; W ← SINAL_DE_X
        MOVWF       L2_C3                   ; L2_C3 ← W
        MOVF        X_DEZENA,W              ; W ← X_DEZENA
        MOVWF       L2_C4                   ; L2_C4 ← W
        MOVF        X_UNIDADE,W             ; W ← X_UNIDADE
        MOVWF       L2_C5                   ; L2_C5 ← W
        MOVLW       ' '
        MOVWF       L2_C6
        MOVLW       ' '
        MOVWF       L2_C7
        MOVLW       ' '
        MOVWF       L2_C8
        MOVF        N_PASSOS_DEZENA,W       ; W ← N_PASSOS_DEZENA
        MOVWF       L2_C9                   ; L2_C9 ← W
        MOVF        N_PASSOS_UNIDADE,W      ; W ← N_PASSOS_UNIDADE
        MOVWF       L2_C10                  ; L2_C10 ← W
        MOVLW       ' '
        MOVWF       L2_C11
        MOVLW       ' '
        MOVWF       L2_C12
        MOVLW       ' '
        MOVWF       L2_C13
        MOVLW       ' '
        MOVWF       L2_C14
        MOVLW       ' '
        MOVWF       L2_C15

        CALL        LCD_LINHA_1
        CALL        LCD_LINHA_2
        RETURN                              ; RETORNA
; ********************************************************************************


; ********************************************************************************
; *           Esta rotina fornece os dados para a Mens_6 a ser enviada ao LCD    *
; ********************************************************************************
Mens_6
                                            ; Medida Eletromet

        MOVLW       'M'
        MOVWF       L1_C0
```

```
MOVLW       'e'
MOVWF       L1_C1
MOVLW       'd'
MOVWF       L1_C2
MOVLW       'i'
MOVWF       L1_C3
MOVLW       'd'
MOVWF       L1_C4
MOVLW       'a'
MOVWF       L1_C5
MOVLW       ' '
MOVWF       L1_C6
MOVLW       'E'
MOVWF       L1_C7
MOVLW       'l'
MOVWF       L1_C8
MOVLW       'e'
MOVWF       L1_C9
MOVLW       't'
MOVWF       L1_C10
MOVLW       'r'
MOVWF       L1_C11
MOVLW       'o'
MOVWF       L1_C12
MOVLW       'm'
MOVWF       L1_C13
MOVLW       'e'
MOVWF       L1_C14
MOVLW       't'
MOVWF       L1_C15

;MOVLW      ' '
;MOVWF      L2_C0
;MOVLW      ' '
;MOVWF      L2_C1
;MOVLW      ' '
;MOVWF      L2_C2
;MOVLW      ' '
;MOVWF      L2_C3
;MOVLW      ' '
;MOVWF      L2_C4
;MOVLW      ' '
;MOVWF      L2_C5
;MOVLW      ' '
;MOVWF      L2_C6
;MOVLW      ' '
;MOVWF      L2_C7
;MOVLW      ' '
;MOVWF      L2_C8
;MOVLW      ' '
;MOVWF      L2_C9
;MOVLW      ' '
;MOVWF      L2_C10
;MOVLW      ' '
;MOVWF      L2_C11
;MOVLW      ' '
```

```
        ;MOVWF      L2_C12
        ;MOVLW      ' '
        ;MOVWF      L2_C13
        ;MOVLW      ' '
        ;MOVWF      L2_C14
        ;MOVLW      ' '
        ;MOVWF      L2_C15

        CALL        LCD_LINHA_1
        CALL        LCD_LINHA_2

        RETURN                                          ; RETORNA
; ********************************************************************************************
```

b31) Sub-rotinas Mens_7; Mens_8 e Mens_9

Estas sub-rotinas, Fig. A52, escrevem suas mensagens no display LCD.

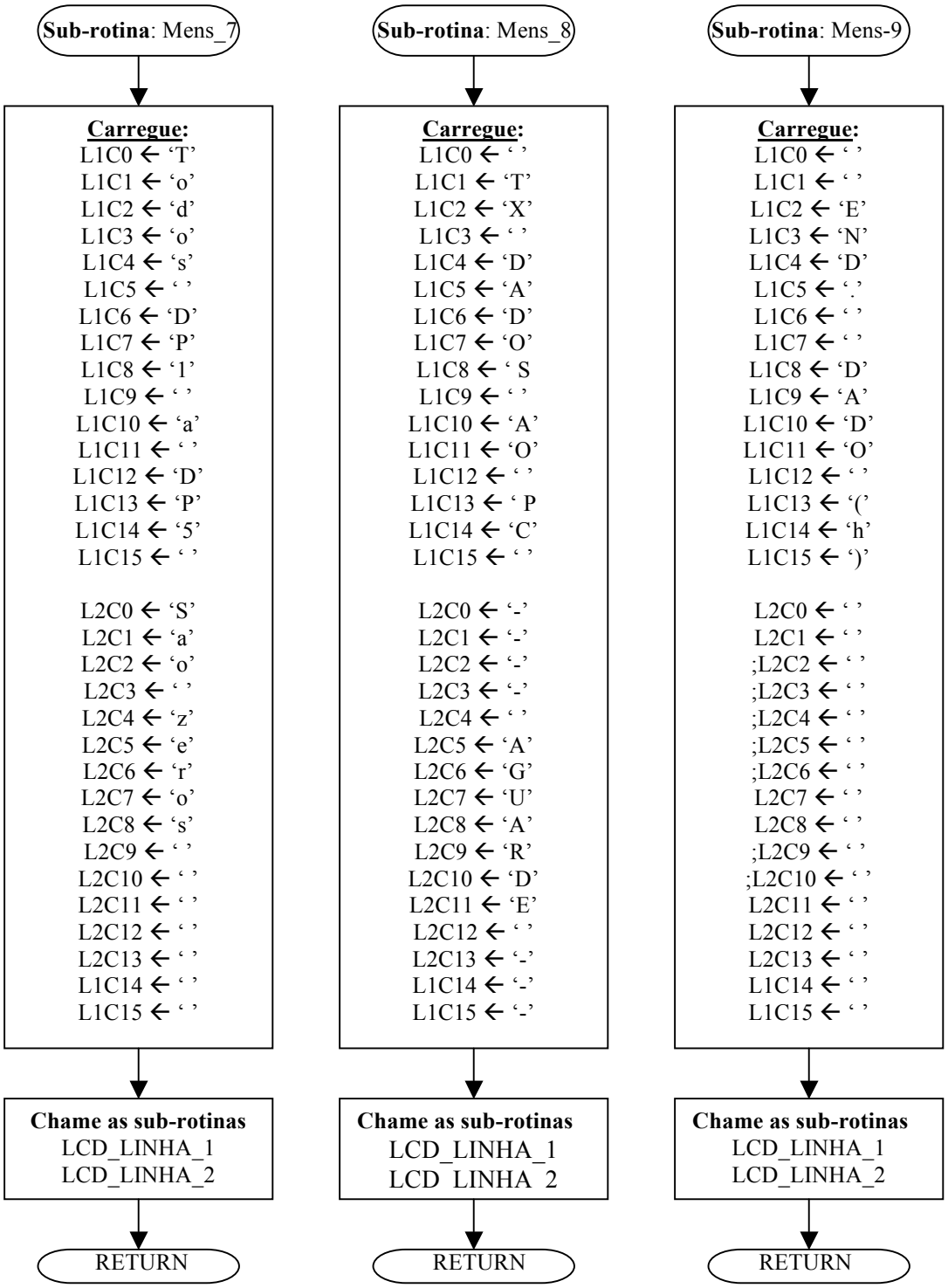


Figura A52 – Fluxograma das Sub-rotinas: Mens_7, Mens_8 e Mens_9

```
; ********************************************************************************
; *          Esta rotina fornece os dados para a Mens_7 a ser enviada ao LCD     *
; ********************************************************************************
Mens_7                                          ; Todos DP1 a DP5
                                                ; São Zeros.
        MOVLW       'T'
        MOVWF       L1_C0
        MOVLW       'o'
        MOVWF       L1_C1
        MOVLW       'd'
        MOVWF       L1_C2
        MOVLW       'o'
        MOVWF       L1_C3
        MOVLW       's'
        MOVWF       L1_C4
        MOVLW       ' '
        MOVWF       L1_C5
        MOVLW       'D'
        MOVWF       L1_C6
        MOVLW       'P'
        MOVWF       L1_C7
        MOVLW       '1'
        MOVWF       L1_C8
        MOVLW       ' '
        MOVWF       L1_C9
        MOVLW       'a'
        MOVWF       L1_C10
        MOVLW       ' '
        MOVWF       L1_C11
        MOVLW       'D'
        MOVWF       L1_C12
        MOVLW       'P'
        MOVWF       L1_C13
        MOVLW       '5'
        MOVWF       L1_C14
        MOVLW       ' '
        MOVWF       L1_C15
                                                ; Sao zeros.

        MOVLW       'S'
        MOVWF       L2_C0
        MOVLW       'a'
        MOVWF       L2_C1
        MOVLW       'o'
        MOVWF       L2_C2
        MOVLW       ' '
        MOVWF       L2_C3
        MOVLW       'z'
        MOVWF       L2_C4
        MOVLW       'e'
        MOVWF       L2_C5
        MOVLW       'r'
        MOVWF       L2_C6
        MOVLW       'o'
        MOVWF       L2_C7
```

```
        MOVLW   's'
        MOVWF   L2_C8
        MOVLW   '.'
        MOVWF   L2_C9
        MOVLW   ' '
        MOVWF   L2_C10
        MOVLW   ' '
        MOVWF   L2_C11
        MOVLW   ' '
        MOVWF   L2_C12
        MOVLW   ' '
        MOVWF   L2_C13
        MOVLW   ' '
        MOVWF   L2_C14
        MOVLW   ' '
        MOVWF   L2_C15

        CALL    LCD_LINHA_1
        CALL    LCD_LINHA_2

        RETURN                                  ; RETORNA

; ********************************************************************************************


; ********************************************************************************************
; *       Esta rotina fornece os dados para a Mens_8 a ser enviada ao LCD                    *
; ********************************************************************************************
;
Mens_8                                          ;  TX DADOS AO PC
        MOVLW   ' '
        MOVWF   L1_C0
        MOVLW   'T'
        MOVWF   L1_C1
        MOVLW   'X'
        MOVWF   L1_C2
        MOVLW   ' '
        MOVWF   L1_C3
        MOVLW   'D'
        MOVWF   L1_C4
        MOVLW   'A'
        MOVWF   L1_C5
        MOVLW   'D'
        MOVWF   L1_C6
        MOVLW   'O'
        MOVWF   L1_C7
        MOVLW   'S'
        MOVWF   L1_C8
        MOVLW   ' '
        MOVWF   L1_C9
        MOVLW   'A'
        MOVWF   L1_C10
        MOVLW   'O'
        MOVWF   L1_C11
        MOVLW   ' '
        MOVWF   L1_C12
        MOVLW   'P'
```

```
        MOVWF   L1_C13
        MOVLW   'C'
        MOVWF   L1_C14
        MOVLW   ' '
        MOVWF   L1_C15
                                                ; --- AGUARDE ----
        MOVLW   '-'
        MOVWF   L2_C0
        MOVLW   '-'
        MOVWF   L2_C1
        MOVLW   '-'
        MOVWF   L2_C2
        MOVLW   '-'
        MOVWF   L2_C3
        MOVLW   ' '
        MOVWF   L2_C4
        MOVLW   'A'
        MOVWF   L2_C5
        MOVLW   'G'
        MOVWF   L2_C6
        MOVLW   'U'
        MOVWF   L2_C7
        MOVLW   'A'
        MOVWF   L2_C8
        MOVLW   'R'
        MOVWF   L2_C9
        MOVLW   'D'
        MOVWF   L2_C10
        MOVLW   'E'
        MOVWF   L2_C11
        MOVLW   ' '
        MOVWF   L2_C12
        MOVLW   '-'
        MOVWF   L2_C13
        MOVLW   '-'
        MOVWF   L2_C14
        MOVLW   '-'
        MOVWF   L2_C15

        CALL    LCD_LINHA_1
        CALL    LCD_LINHA_2

        RETURN                                  ; RETORNA
; *************************************************************************************


; *************************************************************************************
; *       Esta rotina fornece os dados para a Mens_9 a ser enviada ao LCD             *
; *************************************************************************************
;
Mens_9                                          ; bbEND.bbDADOb(h)
                                                ; bbxxyybbbMNbbbbb
        MOVLW   ' '
        ;MOVWF  L1_C0
        MOVLW   ' '
        MOVWF   L1_C1
        MOVLW   ' '
```

```
MOVWF   L1_C2
MOVLW   'E'
MOVWF   L1_C3
MOVLW   'N'
MOVWF   L1_C4
MOVLW   'D'
MOVWF   L1_C5
MOVLW   '.'
MOVWF   L1_C6
MOVLW   ' '
MOVWF   L1_C7
MOVLW   'D'
MOVWF   L1_C8
MOVLW   'A'
MOVWF   L1_C9
MOVLW   'D'
MOVWF   L1_C10
MOVLW   'O'
MOVWF   L1_C11
MOVLW   ' '
MOVWF   L1_C12
MOVLW   '('
MOVWF   L1_C13
MOVLW   'h'
MOVWF   L1_C14
MOVLW   ')'
MOVWF   L1_C15
                                        ; bbEND.bbDADOb(h)
                                        ; bbxxyybbbMNbbbbb
;MOVLW  ' '
;MOVWF  L2_C0
MOVLW   ' '
MOVWF   L2_C1
;MOVLW  ' '
;MOVWF  L2_C2
;MOVLW  ' '
;MOVWF  L2_C3
;MOVLW  ' '
;MOVWF  L2_C4
;MOVLW  ' '
;MOVWF  L2_C5
;MOVLW  ' '
;MOVWF  L2_C6
MOVLW   ' '
MOVWF   L2_C7
MOVLW   ' '
MOVWF   L2_C8
;MOVLW  ' '
;MOVWF  L2_C9
;MOVLW  ' '
;MOVWF  L2_C10
MOVLW   ' '
MOVWF   L2_C11
MOVLW   ' '
MOVWF   L2_C12
MOVLW   ' '
```

```
        MOVWF       L2_C13
        MOVLW       ' '
        MOVWF       L2_C14
        MOVLW       ' '
        MOVWF       L2_C15

        CALL        LCD_LINHA_1
        CALL        LCD_LINHA_2

        RETURN                                          ; RETORNA
;*****************************************************************************************
```

b32) Sub-rotina TX_BYTE_RS232_PC

Esta sub-rotina, Fig. A53, transmite um byte ao computador pessoal via RS-232.

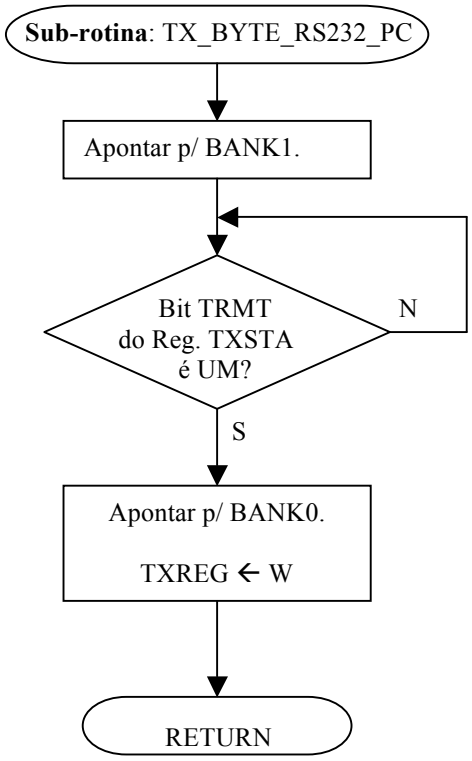


Figura A53 – Fluxograma da Sub-rotina: TX_BYTE_RS232_PC

```
;*****************************************************************************************
TX_BYTE_RS232_PC
        BANK1
        BTFSS       TXSTA,TRMT          ; Testa se Reg Shift resgister está vazio ou TRMT = 1.
        GOTO        $-1                 ; Aguarda buffer TX (TSR interno) esvaziar-se.
        BANK0
        MOVWF       TXREG               ; TXREG ← W, transmite RS232 conteúdo  W
        RETURN
;*****************************************************************************************
```

b33) Sub-rotina TX_CABECALHO_RS232

Esta sub-rotina, Fig. A54, transmite o cabeçalho do relatório ao computador pessoal via RS-232.

Formato do cabeçalho.

;bENDERECO_0bENDERECO_6bbYbbbXbb_____DADOS_____bbbbb

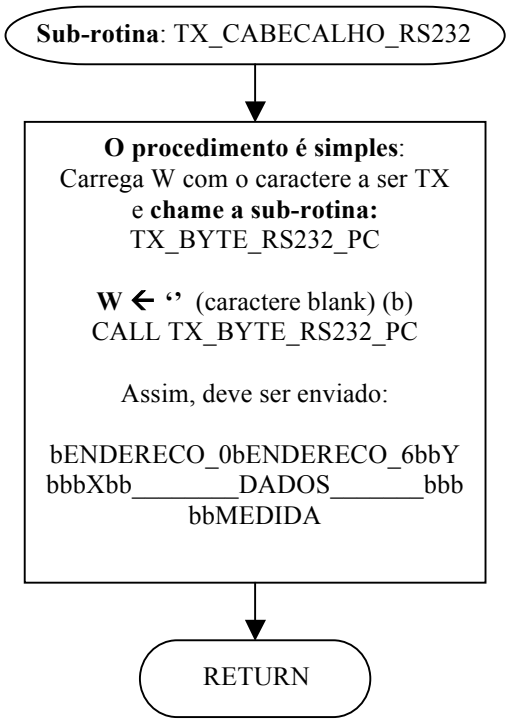


Figura A54 – Fluxograma da Sub-rotina: TX_CABECALHO_RS232

```
;*******************************************************************************
TX_CABECALHO_RS232
        ;bENDERECO_0bENDERECO_6bbYbbbXbb_____DADOS_____bbbbbMEDIDA

                                                ;bbb00000hbbbbb00006hbbb92b
                                                ;-20bb00b00b00b00b00b00b00b+0,1172E-19

        MOVLW       ' '                         ; W ← ' '
        PAGESEL     TX_BYTE_RS232_PC
        CALL        TX_BYTE_RS232_PC
        MOVLW       'E'                         ; W ← 'E'
        CALL        TX_BYTE_RS232_PC
        MOVLW       'N'                         ; W ← 'N'
        CALL        TX_BYTE_RS232_PC
```

```
MOVLW   'D'                     ; W ← 'D'
CALL    TX_BYTE_RS232_PC
MOVLW   'E'                     ; W ← 'E'
CALL    TX_BYTE_RS232_PC
MOVLW   'R'                     ; W ← 'R'
CALL    TX_BYTE_RS232_PC
MOVLW   'E'                     ; W ← 'E'
CALL    TX_BYTE_RS232_PC
MOVLW   'C'                     ; W ← 'C'
CALL    TX_BYTE_RS232_PC
MOVLW   'O'                     ; W ← 'O'
CALL    TX_BYTE_RS232_PC
MOVLW   '_'                     ; W ← '_'
CALL    TX_BYTE_RS232_PC
MOVLW   '0'                     ; W ← '0'
CALL    TX_BYTE_RS232_PC
MOVLW   ' '                     ; W ← ' '
CALL    TX_BYTE_RS232_PC
MOVLW   'E'                     ; W ← 'E'
CALL    TX_BYTE_RS232_PC
MOVLW   'N'                     ; W ← 'N'
CALL    TX_BYTE_RS232_PC
MOVLW   'D'                     ; W ← 'D'
CALL    TX_BYTE_RS232_PC
MOVLW   'E'                     ; W ← 'E'
CALL    TX_BYTE_RS232_PC
MOVLW   'R'                     ; W ← 'R'
CALL    TX_BYTE_RS232_PC
MOVLW   'E'                     ; W ← 'E'
CALL    TX_BYTE_RS232_PC
MOVLW   'C'                     ; W ← 'C'
CALL    TX_BYTE_RS232_PC
MOVLW   'O'                     ; W ← 'O'
CALL    TX_BYTE_RS232_PC
MOVLW   '_'                     ; W ← '_'
CALL    TX_BYTE_RS232_PC
MOVLW   '6'                     ; W ← '6'
CALL    TX_BYTE_RS232_PC
MOVLW   ' '                     ; W ← ' '
CALL    TX_BYTE_RS232_PC
MOVLW   ' '                     ; W ← ' '
CALL    TX_BYTE_RS232_PC
MOVLW   'Y'                     ; W ← 'Y'
CALL    TX_BYTE_RS232_PC
MOVLW   ' '                     ; W ← ' '
CALL    TX_BYTE_RS232_PC
MOVLW   ' '                     ; W ← ' '
CALL    TX_BYTE_RS232_PC
MOVLW   ' '                     ; W ← ' '
CALL    TX_BYTE_RS232_PC
MOVLW   'X'                     ; W ← 'X'
CALL    TX_BYTE_RS232_PC
MOVLW   ' '                     ; W ← ' '
CALL    TX_BYTE_RS232_PC
MOVLW   ' '                     ; W ← ' '
CALL    TX_BYTE_RS232_PC
```

```
MOVLW   '_'                     ; W ← '_'
CALL    TX_BYTE_RS232_PC
MOVLW   '_'                     ; W ← '_'
CALL    TX_BYTE_RS232_PC
MOVLW   '_'                     ; W ← '_'
CALL    TX_BYTE_RS232_PC
MOVLW   '_'                     ; W ← '_'
CALL    TX_BYTE_RS232_PC
MOVLW   '_'                     ; W ← '_'
CALL    TX_BYTE_RS232_PC
MOVLW   '_'                     ; W ← '_'
CALL    TX_BYTE_RS232_PC
MOVLW   '_'                     ; W ← '_'
CALL    TX_BYTE_RS232_PC
MOVLW   'D'                     ; W ← 'D'
CALL    TX_BYTE_RS232_PC
MOVLW   'A'                     ; W ← 'A'
CALL    TX_BYTE_RS232_PC
MOVLW   'D'                     ; W ← 'D'
CALL    TX_BYTE_RS232_PC
MOVLW   'O'                     ; W ← 'O'
CALL    TX_BYTE_RS232_PC
MOVLW   'S'                     ; W ← 'S'
CALL    TX_BYTE_RS232_PC
MOVLW   '_'                     ; W ← '_'
CALL    TX_BYTE_RS232_PC
MOVLW   '_'                     ; W ← '_'
CALL    TX_BYTE_RS232_PC
MOVLW   '_'                     ; W ← '_'
CALL    TX_BYTE_RS232_PC
MOVLW   '_'                     ; W ← '_'
CALL    TX_BYTE_RS232_PC
MOVLW   '_'                     ; W ← '_'
CALL    TX_BYTE_RS232_PC
MOVLW   '_'                     ; W ← '_'
CALL    TX_BYTE_RS232_PC
MOVLW   ' '                     ; W ← ' '
CALL    TX_BYTE_RS232_PC
MOVLW   ' '                     ; W ← ' '
CALL    TX_BYTE_RS232_PC
MOVLW   ' '                     ; W ← ' '
CALL    TX_BYTE_RS232_PC
MOVLW   ' '                     ; W ← ' '
CALL    TX_BYTE_RS232_PC
MOVLW   'M'                     ; W ← 'M'
CALL    TX_BYTE_RS232_PC
MOVLW   'E'                     ; W ← 'E'
CALL    TX_BYTE_RS232_PC
MOVLW   'D'                     ; W ← 'D'
CALL    TX_BYTE_RS232_PC
MOVLW   'I'                     ; W ← 'I'
```

```
        CALL    TX_BYTE_RS232_PC
        MOVLW   'D'                     ; W ← 'D'
        CALL    TX_BYTE_RS232_PC
        MOVLW   'A'                     ; W ← 'A'
        CALL    TX_BYTE_RS232_PC
        MOVLW   0x0D                    ; W ← 0x0D [carriage return: retorna o cursor]
        CALL    TX_BYTE_RS232_PC        ; de baixo na posição mais à esquerda.
        MOVLW   0x0A                    ; W ← 0x0A [line feed] deixa linha em branco]
        CALL    TX_BYTE_RS232_PC
        MOVLW   0x0A                    ; W ← 0x0A [line feed] deixa linha em branco
        CALL    TX_BYTE_RS232_PC
        RETURN
;*******************************************************************************************
```

b34) Sub-rotina TX_LINHA_DADOS_RS232

Esta sub-rotina, Fig. A55, transmite uma linha completa ao CP via RS-232.

Formato da linha a ser transmitida. Respeita o formato do cabeçalho.

;bENDERECO_0bENDERECO_6bbYbbbXbb_____DADOS_____bbbbb

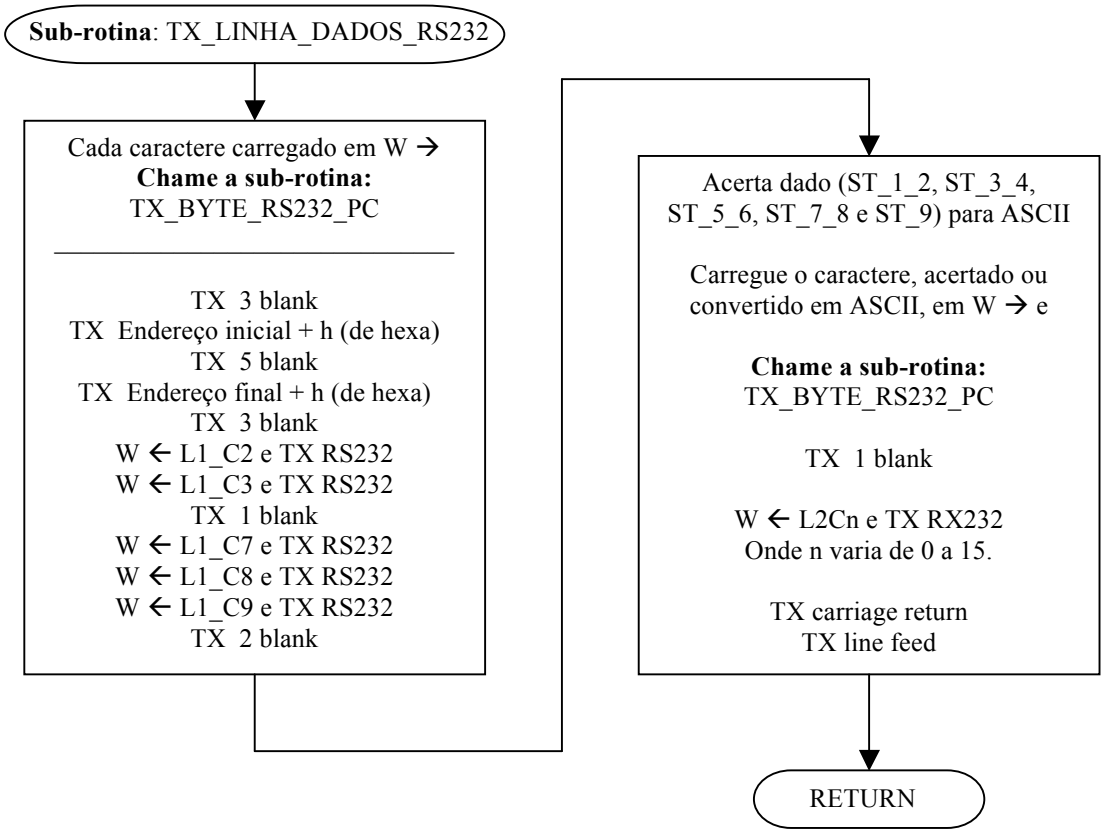


Figura A55 – Fluxograma da Sub-rotina: TX_LINHA_DADOS_RS232

```
;*********************************************************************************
TX_LINHA_DADOS_RS232
        ;bENDERECO_0bENDERECO_6bbYbbbXbb________DADOS_______bbbbbMEDIDA

                                                ;bbb00000hbbbbb00006hbbb92b-
                                                ;20bb00b00b00b00b00b00b00b+0,1172E-19

        MOVLW       ' '                         ; W ← ' '
        CALL        TX_BYTE_RS232_PC
        MOVLW       ' '                         ; W ← ' '
        CALL        TX_BYTE_RS232_PC
        MOVLW       ' '                         ; W ← ' '
        CALL        TX_BYTE_RS232_PC

        ;-----------TX endereço inicial da faixa de endereços dos dados dessa linha-----------------------------
        MOVF        ENDERECO_I2C_5,W            ; Endereço mais sig. (mais a esquerda, ASCII)
        CALL        TX_BYTE_RS232_PC
        MOVF        ENDERECO_I2C_4,W
        CALL        TX_BYTE_RS232_PC
        MOVF        ENDERECO_I2C_3,W
        CALL        TX_BYTE_RS232_PC
        MOVF        ENDERECO_I2C_2,W
        CALL        TX_BYTE_RS232_PC
        MOVF        ENDERECO_I2C_1,W            ; Endereço menos sig. (mais a direita, ASCII)
        CALL        TX_BYTE_RS232_PC
        MOVLW       'h'                         ; h = hexadecimal
        CALL        TX_BYTE_RS232_PC
        ;-------------------------------------------------------------------------------------------------------

        MOVLW       ' '                         ; W ← ' '
        CALL        TX_BYTE_RS232_PC
        MOVLW       ' '                         ; W ← ' '
        CALL        TX_BYTE_RS232_PC
        MOVLW       ' '                         ; W ← ' '
        CALL        TX_BYTE_RS232_PC
        MOVLW       ' '                         ; W ← ' '
        CALL        TX_BYTE_RS232_PC
        MOVLW       ' '                         ; W ← ' '
        CALL        TX_BYTE_RS232_PC

        ;-----------TX endereco final da faixa de enderecos dos dados dessa linha-----------------------------
        MOVF        ENDERECO_I2C_51,W           ; Endereco mais sig. (mais a esquerda, ASCII)
        CALL        TX_BYTE_RS232_PC
        MOVF        ENDERECO_I2C_41,W
        CALL        TX_BYTE_RS232_PC
        MOVF        ENDERECO_I2C_31,W
        CALL        TX_BYTE_RS232_PC
        MOVF        ENDERECO_I2C_21,W
        CALL        TX_BYTE_RS232_PC
        MOVF        ENDERECO_I2C_11,W           ; Endereco menos sig.(mais a direita, ASCII)
        CALL        TX_BYTE_RS232_PC
        MOVLW       'h'                         ; h = hexadecimal
        CALL        TX_BYTE_RS232_PC
        ;-------------------------------------------------------------------------------------------------------

        MOVLW       ' '                         ; W ← ' '
```

```
CALL        TX_BYTE_RS232_PC
MOVLW       ' '                     ; W ← ' '
CALL        TX_BYTE_RS232_PC
MOVLW       ' '                     ; W ← ' '
CALL        TX_BYTE_RS232_PC

MOVF        L1_C2,W                 ; W ← L1_C2 (Y_DEZENA em ASCII)
CALL        TX_BYTE_RS232_PC
MOVF        L1_C3,W                 ; W ← L1_C3 (Y_UNIDADE em ASCII)
CALL        TX_BYTE_RS232_PC

MOVLW       ' '                     ; W ← ' '
CALL        TX_BYTE_RS232_PC

MOVF        L1_C7,W                 ; W ← L1_C7 (Sinal do X em ASCII)
CALL        TX_BYTE_RS232_PC
MOVF        L1_C8,W                 ; W ← L1_C8 (X_DEZENA em ASCII)
CALL        TX_BYTE_RS232_PC
MOVF        L1_C9,W                 ; W ← L1_C9 (X_UNIDADE em ASCII)
CALL        TX_BYTE_RS232_PC

MOVLW       ' '                     ; W ← ' '
CALL        TX_BYTE_RS232_PC
MOVLW       ' '                     ; W ← ' '
CALL        TX_BYTE_RS232_PC

;--------------------Acerta dado ST_1_2 para ASCII e TX RS232-----------------------------------------
MOVF        ST_1_2,W                ; W ← ST_1_2
MOVWF       WTEMP                   ; WTEMP ← W
SWAPF       WTEMP,W                 ; Troca nibbles menos e mais significativo de
                                    ; VALOR_DADOS
CLRF        PCLATH                  ; PCLATH ← 0x00
CALL        HEXA_ASCII              ; Chama subrotina que Converte nibble menos
                                    ; sig. para ASCII.
PAGESEL     TX_BYTE_RS232_PC
CALL        TX_BYTE_RS232_PC

MOVF        ST_1_2,W                ; W ← ST_1_2
CLRF        PCLATH                  ; PCLATH ← 0x00
CALL        HEXA_ASCII              ; Chama subrotina que Converte nibble menos
                                    ; sig. para ASCII.
PAGESEL     TX_BYTE_RS232_PC
CALL        TX_BYTE_RS232_PC
;-------------------------------------------------------------------------------------------------------------------

MOVLW       ' '                     ; W ← ' '
CALL        TX_BYTE_RS232_PC

;--------------------Acerta dado ST_3_4 para ASCII e TX RS232-----------------------------------------
MOVF        ST_3_4,W                ; W ← ST_3_4
MOVWF       WTEMP                   ; WTEMP ← W
SWAPF       WTEMP,W                 ; Troca nibbles menos e mais significativo de
                                    ; VALOR_DADOS
CLRF        PCLATH                  ; PCLATH ← 0x00
CALL        HEXA_ASCII              ; Chama subrotina que Converte nibble menos
                                    ; sig. para ASCII.
```

```
PAGESEL     TX_BYTE_RS232_PC
CALL        TX_BYTE_RS232_PC

MOVF        ST_3_4,W                ; W ← ST_3_4
CLRF        PCLATH                  ; PCLATH ← 0x00
CALL        HEXA_ASCII              ; Chama subrotina que Converte nibble menos
                                    ; sig. para ASCII.
PAGESEL     TX_BYTE_RS232_PC
CALL        TX_BYTE_RS232_PC
;-------------------------------------------------------------------------------------------------------------------

MOVLW       ' '                     ; W ← ' '
CALL        TX_BYTE_RS232_PC

;--------------------Acerta dado ST_5_6 para ASCII e TX RS232-----------------------------------------
MOVF        ST_5_6,W                ; W ← ST_5_6
MOVWF       WTEMP                   ; WTEMP ← W
SWAPF       WTEMP,W                 ; Troca nibbles menos e mais significativo de
                                    ; VALOR_DADOS
CLRF        PCLATH                  ; PCLATH ← 0x00
CALL        HEXA_ASCII              ; Chama subrotina que Converte nibble menos
                                    ; sig. para ASCII.
PAGESEL     TX_BYTE_RS232_PC
CALL        TX_BYTE_RS232_PC

MOVF        ST_5_6,W                ; W ← ST_5_6
CLRF        PCLATH                  ; PCLATH ← 0x00
CALL        HEXA_ASCII              ; Chama subrotina que Converte nibble menos
                                    ; sig. para ASCII.
PAGESEL     TX_BYTE_RS232_PC
CALL        TX_BYTE_RS232_PC
;-------------------------------------------------------------------------------------------------------------------

MOVLW       ' '                     ; W ← ' '
CALL        TX_BYTE_RS232_PC

;--------------------Acerta dado ST_7_8 para ASCII e TX RS232-----------------------------------------
MOVF        ST_7_8,W                ; W ← ST_7_8
MOVWF       WTEMP                   ; WTEMP ← W
SWAPF       WTEMP,W                 ; Troca nibbles menos e mais significativo de
                                    ; VALOR_DADOS
CLRF        PCLATH                  ; PCLATH ← 0x00
CALL        HEXA_ASCII              ; Chama subrotina que Converte nibble menos
                                    ; sig. para ASCII.
PAGESEL     TX_BYTE_RS232_PC
CALL        TX_BYTE_RS232_PC

MOVF        ST_7_8,W                ; W ← ST_7_8
CLRF        PCLATH                  ; PCLATH ← 0x00
CALL        HEXA_ASCII              ; Chama subrotina que Converte nibble menos
                                    ; sig. para ASCII.
PAGESEL     TX_BYTE_RS232_PC
CALL        TX_BYTE_RS232_PC
;-------------------------------------------------------------------------------------------------------------------

MOVLW       ' '                     ; W ← ' '
```

```
CALL            TX_BYTE_RS232_PC

;--------------------Acerta dado ST_9 para ASCII e TX RS232-----------------------------------------------
MOVF            ST_9,W                  ; W ← ST_9
MOVWF           WTEMP                   ; WTEMP ← W
SWAPF           WTEMP,W                 ; Troca nibbles menos e mais significativo de
                                        ; VALOR_DADOS
CLRF            PCLATH                  ; PCLATH ← 0x00
CALL            HEXA_ASCII              ; Chama subrotina que Converte nibble menos
                                        ; sig. para ASCII.
PAGESEL         TX_BYTE_RS232_PC
CALL            TX_BYTE_RS232_PC

MOVF            ST_9,W                  ; W ← ST_9
CLRF            PCLATH                  ; PCLATH ← 0x00
CALL            HEXA_ASCII              ; Chama subrotina que Converte nibble menos
                                        ; sig. para ASCII.
PAGESEL         TX_BYTE_RS232_PC
CALL            TX_BYTE_RS232_PC
;-------------------------------------------------------------------------------------------------------------------

MOVLW           ' '                     ; W ← ' '
CALL            TX_BYTE_RS232_PC

MOVF            L2_C0,W                 ; W ← L2_C0
CALL            TX_BYTE_RS232_PC
MOVF            L2_C1,W                 ; W ← L2_C1
CALL            TX_BYTE_RS232_PC
MOVF            L2_C2,W                 ; W ← L2_C2
CALL            TX_BYTE_RS232_PC
MOVF            L2_C3,W                 ; W ← L2_C3
CALL            TX_BYTE_RS232_PC
MOVF            L2_C4,W                 ; W ← L2_C4
CALL            TX_BYTE_RS232_PC
MOVF            L2_C5,W                 ; W ← L2_C5
CALL            TX_BYTE_RS232_PC
MOVF            L2_C6,W                 ; W ← L2_C6
CALL            TX_BYTE_RS232_PC
MOVF            L2_C7,W                 ; W ← L2_C7
CALL            TX_BYTE_RS232_PC
MOVF            L2_C8,W                 ; W ← L2_C8
CALL            TX_BYTE_RS232_PC
MOVF            L2_C9,W                 ; W ← L2_C9
CALL            TX_BYTE_RS232_PC
MOVF            L2_C10,W                ; W ← L2_C10
CALL            TX_BYTE_RS232_PC
MOVF            L2_C11,W                ; W ← L2_C11
CALL            TX_BYTE_RS232_PC
MOVF            L2_C12,W                ; W ← L2_C12
CALL            TX_BYTE_RS232_PC
MOVF            L2_C13,W                ; W ← L2_C13
CALL            TX_BYTE_RS232_PC
MOVF            L2_C14,W                ; W ← L2_C14
CALL            TX_BYTE_RS232_PC
MOVF            L2_C15,W                ; W ← L2_C15
CALL            TX_BYTE_RS232_PC
```

```
        MOVLW       0x0D                ; W ← 0x0D [carriage return: retorna o cursor p/ linha]
        CALL        TX_BYTE_RS232_PC    ; de baixo na posição mais à esquerda.
        MOVLW       0x0A                ; W ← 0x0A [line feed] deixa uma linha em branco.
        CALL        TX_BYTE_RS232_PC
        RETURN
;*******************************************************************************
```

b35) Sub-rotina GRAVA_MEDIDA_EEPROM

Esta sub-rotina, Fig. A56, grava na memória 24LC1025 a medida do eletrômetro.

Iniciará a gravação no endereço 0000h, ou seja:
0000h ← Y / 0001h ← X / 0002h ← ST_1_2 / 0003h ← ST_3_4 / 0004h ← ST_5_6

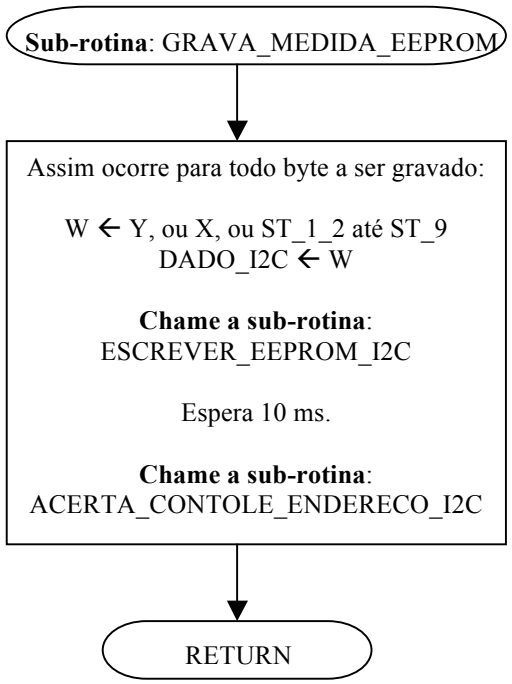


Figura A56 – Fluxograma da Sub-rotina: GRAVA_MEDIDA_EEPROM

```
;*******************************************************************************
;       Sub-rotina gravar dados da medida eletrômetro endereços sequênciais da EEPROM 24LC1025      *
;*******************************************************************************
; A partir do endereço 0x0000 será gravado os dados das medidas do eletrômetro, os quais já estarão
; residentes nos respectivos registradores, na seguinte ordem:
;
;   Endereço(h)     Registradores
;       0000            Y               \
;       0001            X                |
;       0002            ST_1_2           |
;       0003            ST_3_4            > Lugares onde residirão os dados da 1a medida do
;       0004            ST_5_6           | eletrômetro
;       0005            ST_7_8           |
;       0006            ST_9            /
;
;       0007            Y               \
;       0008            X                |
;       0009            ST_1_2           |
;       000A            ST_3_4            > Lugares onde residirão os dados da 2a medida do
;       000B            ST_5_6           | eletrômetro, e assim por diante.
;       000C            ST_7_8           |
;       000D            ST_9            /


GRAVA_MEDIDA_EEPROM
    ;--------------------Salva na 1a posição o valor de Y ----------------------------------------------------

        MOVF        Y,W                             ; W ← Y
        MOVWF       DADO_I2C                        ; DADO_I2C ← W [conteúdo de Y a
                                                    ; ser gravado]
        PAGESEL     ESCREVER_EEPROM_I2C             ; Ajusta PCLATH
        CALL        ESCREVER_EEPROM_I2C             ; Chama rotina gravação
        MOVLW       .10
        CALL        DELAY_MS                        ; Garante Tempo de ESCRITA (10ms).
        CALL        ACERTA_CONTOLE_ENDERECO_I2C     ; Chama subrotina acerta o controle
                                                    ; e o endereco memória 24LC1025
                                                    ;[para até 4 CIs ou 4x128kBytes].
        ;-------------------------------------------------------------------------------------------------------
    ;--------------------Salva na 2a posição o valor de X----------------------------------------------------

        MOVF        X,W                             ; W ← X
        MOVWF       DADO_I2C                        ; DADO_I2C ← W W [conteúdo de X
                                                    ; a ser gravado]
        CALL        ESCREVER_EEPROM_I2C             ; Chama rotina de gravação.
        MOVLW       .10
        CALL        DELAY_MS                        ; Garante tempo de escrita (10ms)
        CALL        ACERTA_CONTOLE_ENDERECO_I2C     ; Chama subrotina acerta o controle
                                                    ; e o endereco memória 24LC1025
                                                    ; [para até 4 CIs ou 4x128kBytes].
        ;-------------------------------------------------------------------------------------------------------
    ;--------------------Salva na 3a posição o valor de ST_1_2-----------------------------------------------
        MOVF        ST_1_2,W                        ; W ← ST_1_2
        MOVWF       DADO_I2C                        ; DADO_I2C ← W [Conteúdo de
                                                    ; ST_1_2 a ser gravado]
        CALL        ESCREVER_EEPROM_I2C             ; Chama rotina gravação
        MOVLW       .10
```

```
        CALL    DELAY_MS                        ; Garante tempo de escrita (10ms).
        CALL    ACERTA_CONTOLE_ENDERECO_I2C     ;Chama subrotina que acerta o controle
                                                ; e o endereco memória 24LC1025
                                                ; [para até 4 CIs ou 4x128kBytes].
        ;-------------------------------------------------------------------------------------------------------
    ;-------------------Salva na 4a posição o valor de ST_3_4-----------------------------------------------
        MOVF    ST_3_4,W                        ; W ← ST_3_4
        MOVWF   DADO_I2C                        ; DADO_I2C ← W[Comteúdo de
                                                ; ST_3_4 a ser gravado]
        CALL    ESCREVER_EEPROM_I2C             ; Chama rotina gravação
        MOVLW   .10
        CALL    DELAY_MS                        ; Garante tempo de escrita (10ms).
        CALL    ACERTA_CONTOLE_ENDERECO_I2C     ;Chama subrotina que acerta o controle
                                                ; e o endereco memória 24LC1025
                                                ; [para até 4 CIs ou 4x128kBytes].
        ;-------------------------------------------------------------------------------------------------------
    ;-------------------Salva na 5a posição o valor de ST_5_6-----------------------------------------------
        MOVF    ST_5_6,W                        ; W ← ST_5_6
        MOVWF   DADO_I2C                        ; DADO_I2C ← W [Conteúdo de
                                                ; ST_5_6 a ser gravado]
        CALL    ESCREVER_EEPROM_I2C             ; Chama rotina gravação
        MOVLW   .10
        CALL    DELAY_MS                        ; Garante tempo de escrita (10ms).
        CALL    ACERTA_CONTOLE_ENDERECO_I2C     ;Chama subrotina que acerta o controle
                                                ; e o endereço memória 24LC1025
                                                ; [para até 4 CIs ou 4x128kBytes].
        ;-------------------------------------------------------------------------------------------------------
    ;-------------------Salva na 6a posição o valor de ST_7_8-----------------------------------------------
        MOVF    ST_7_8,W                        ; W ← ST_7_8
        MOVWF   DADO_I2C                        ; DADO_I2C ← W [Conteúdo de
                                                ; ST_7_8 a ser gravado]
        CALL    ESCREVER_EEPROM_I2C             ; Chama rotina gravação O
        MOVLW   .10
        CALL    DELAY_MS                        ; Garante tempo de escrita (10ms).
        CALL    ACERTA_CONTOLE_ENDERECO_I2C     ;Chama subrotina que acerta o controle
                                                ;  e o endereco memória 24LC1025
                                                ; [para até 4 CIs ou 4x128kBytes].
        ;-------------------------------------------------------------------------------------------------------
    ;-------------------Salva na 7a posição o valor de ST_9-------------------------------------------------
        MOVF    ST_9,W                          ; W ← ST_9
        MOVWF   DADO_I2C                        ; DADO_I2C ← W [Conteúdo de
                                                ; ST_9 a ser gravado]
        CALL    ESCREVER_EEPROM_I2C             ; Chama rotina gravação
        MOVLW   .10
        CALL    DELAY_MS                        ; Garante tempo de escrita (10ms).
        CALL    ACERTA_CONTOLE_ENDERECO_I2C     ;Chama subrotina que acerta o controle
                                                ; e o endereco memória 24LC1025
                                                 ; [para até 4 CIs ou 4x128kBytes].
        ;-------------------------------------------------------------------------------------------------------
        RETURN
;*******************************************************************************
```

b36) Sub-rotina LE_EEPROM_SERIAL

Esta sub-rotina, Fig. A57, lê 7 endereços sequênciais da memória 24LC1025, os quais referem a uma única medida do eletrômetro.

A leitura iniciará no endereço 0000h, ou seja:
0000h → Y / 0001h → X / 0002h → ST_1_2 / 0003h → ST_3_4 / 0004h → ST_5_6
0005h → ST_7_8 / 0006h → ST_9 esta é uma medida. E assim ocorre atá o final

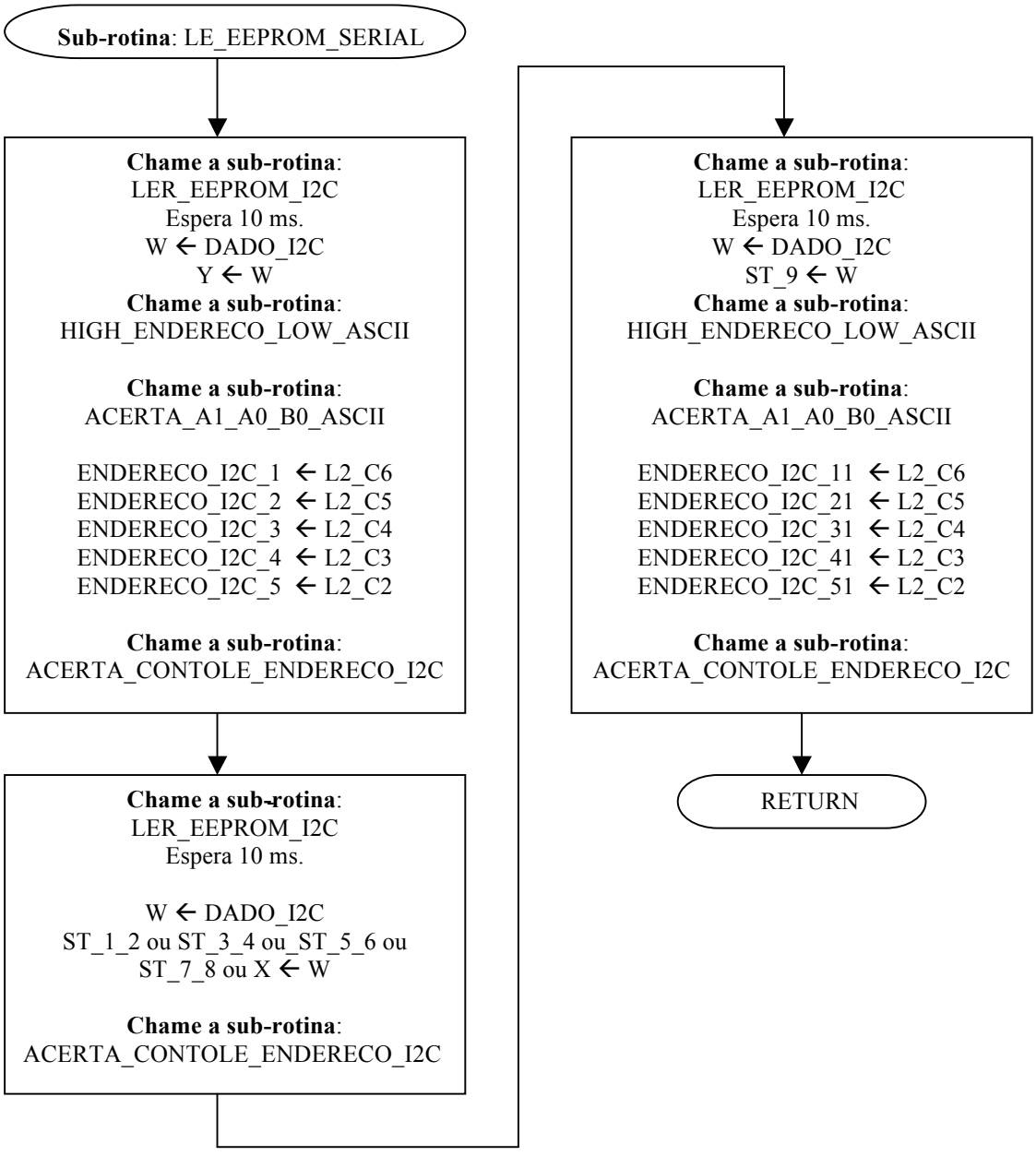


Figura A57 – Fluxograma da Sub-rotina: LE_EEPROM_SERIAL

```
;*******************************************************************************
;                Sub-rotina LÊ os dados de 7 endereços sequênciais da EEPROM 24LC1025          *
;*******************************************************************************
; A partir do endereço 0x0000 será lido os dados armazenados nas memórias 24LC1025, na seguinte ordem:
;
;   Endereço(h)      Registradores
;         0000             Y              \
;         0001             X               |
;         0002             ST_1_2          |
;         0003             ST_3_4           > Lugares onde residirão os dados da 1a medida do
;         0004             ST_5_6          | eletrômetro
;         0005             ST_7_8          |
;         0006             ST_9           /
;
;         0007             Y              \
;         0008             X               |
;         0009             ST_1_2          |
;         000A             ST_3_4           > Lugares onde residirão os dados da 2a medida do
;         000B             ST_5_6          | eletrômetro, e assim por diante.
;         000C             ST_7_8          |
;         000D             ST_9           /

LE_EEPROM_SERIAL
      ;--------------------Lê a 1a posição que tem o valor de Y ------------------------------

         PAGESEL     LER_EEPROM_I2C
         CALL        LER_EEPROM_I2C                  ; Chama rotina leitura Byte 24LC1025
         MOVLW       .10
         PAGESEL     DELAY_MS
         CALL        DELAY_MS                        ; Garante tempo de escrita (10ms).

         MOVF        DADO_I2C,W                      ; W ← DADO_I2C
         MOVWF       Y                               ; Y ← W

         PAGESEL     HIGH_ENDERECO_LOW_ASCII
         CALL        HIGH_ENDERECO_LOW_ASCII
         PAGESEL     ACERTA_A1_A0_B0_ASCII
         CALL        ACERTA_A1_A0_B0_ASCII

         MOVF        L2_C6,W                         ; W ← L2_C6
         MOVWF       ENDERECO_I2C_1                  ; ENDERECO_I2C_1 ← W
         MOVF        L2_C5,W                         ; W ← L2_C5
         MOVWF       ENDERECO_I2C_2                  ; ENDERECO_I2C_2 ← W
         MOVF        L2_C4,W                         ; W ← L2_C4
         MOVWF       ENDERECO_I2C_3                  ; ENDERECO_I2C_3 ← W
         MOVF        L2_C3,W                         ; W ← L2_C3
         MOVWF       ENDERECO_I2C_4                  ; ENDERECO_I2C_4 ← W
         MOVF        L2_C2,W                         ; W ← L2_C2
         MOVWF       ENDERECO_I2C_5                  ; ENDERECO_I2C_5 ← W

         PAGESEL     ACERTA_CONTOLE_ENDERECO_I2C
         CALL        ACERTA_CONTOLE_ENDERECO_I2C     ; Chama sub-rotina acerta o controle
                                                     ; e o endereco memória 24LC1025
                                                     ; [para até 4 CIs ou 4x128kBytes].
         ;-------------------------------------------------------------------------------------------------------------------
```

```
;--------------------Lê a 2a posição que tem o valor de X-----------------------------------------------------------

    PAGESEL     LER_EEPROM_I2C
    CALL        LER_EEPROM_I2C                  ; Chama rotina de rotina de leitura byte
                                                ; 24LC1025
    MOVLW       .10
    PAGESEL     DELAY_MS
    CALL        DELAY_MS                        ; Garante tempo de escrita (10ms).

    MOVF        DADO_I2C,W                      ; W ← DADO_I2C
    MOVWF       X                               ; X ←W

    PAGESEL     ACERTA_CONTOLE_ENDERECO_I2C
    CALL        ACERTA_CONTOLE_ENDERECO_I2C     ; Chama sub-rotina acerta o controle
                                                ; e o endereco memória 24LC1025
                                                ; [para até 4 CIs ou 4x128kBytes].
    ;-------------------------------------------------------------------------------------------------------------
;--------------------Lê a 3a posição que tem o valor de ST_1_2-----------------------------------------------------

    PAGESEL     LER_EEPROM_I2C
    CALL        LER_EEPROM_I2C                  ; Chama rotina de leitura 24LC1025.
    MOVLW       .10
    PAGESEL     DELAY_MS
    CALL        DELAY_MS                        ; Garante tempo de escrita (10ms).
    MOVF        DADO_I2C,W                      ; W ← DADO_I2C
    MOVWF       ST_1_2                          ; ST_1_2 ← W

    PAGESEL     ACERTA_CONTOLE_ENDERECO_I2C
    CALL        ACERTA_CONTOLE_ENDERECO_I2C     ; Chama sub-rotina acerta o controle
                                                ; e o endereco memória 24LC1025
                                                ; [para até 4 CIs ou 4x128kBytes].
    ;-------------------------------------------------------------------------------------------------------------
;--------------------Lê a 4a posição que tem o valor de ST_3_4-----------------------------------------------------

    PAGESEL     LER_EEPROM_I2C
    CALL        LER_EEPROM_I2C                  ; Chama rotina de leitura 24LC1025
    MOVLW       .10
    PAGESEL     DELAY_MS
    CALL        DELAY_MS                        ; Garante tempo de escrita (10ms).
    MOVF        DADO_I2C,W                      ; W ← DADO_I2C
    MOVWF       ST_3_4                          ; ST_3_4 ← W

    PAGESEL     ACERTA_CONTOLE_ENDERECO_I2C
    CALL        ACERTA_CONTOLE_ENDERECO_I2C     ; Chama sub-rotina acerta o controle
                                                ; e o endereco memória 24LC1025
                                                ; [para até 4 CIs ou 4x128kBytes].
    ;-------------------------------------------------------------------------------------------------------------
;--------------------Lê a 5a posição que tem o valor de ST_5_6-----------------------------------------------------

    PAGESEL     LER_EEPROM_I2C
    CALL        LER_EEPROM_I2C                  ; Chama rotina de leitura 24LC1025
    MOVLW       .10
    PAGESEL     DELAY_MS
    CALL        DELAY_MS                        ; Garante tempo de escrita (10ms).
    MOVF        DADO_I2C,W                      ; W ← DADO_I2C
    MOVWF       ST_5_6                          ; ST_5_6 ← W
```

```
    PAGESEL     ACERTA_CONTOLE_ENDERECO_I2C
    CALL        ACERTA_CONTOLE_ENDERECO_I2C  ; Chama sub-rotina acerta o controle
                                             ; e o endereco memória 24LC1025
                                             ; [para até 4 CIs ou 4x128kBytes].
    ;-------------------------------------------------------------------------------------------------------------
;-------------------- Lê a 6a posição que tem o valor de ST_7_8----------------------------------------------

    PAGESEL     LER_EEPROM_I2C
    CALL        LER_EEPROM_I2C               ; Chama rotina de leitura 24LC1025
    MOVLW       .10
    PAGESEL     DELAY_MS
    CALL        DELAY_MS                     ; Garante tempo de escrita (10ms).
    MOVF        DADO_I2C,W                   ; W ← DADO_I2C
    MOVWF       ST_7_8                       ; ST_7_8 ← W

    PAGESEL     ACERTA_CONTOLE_ENDERECO_I2C
    CALL        ACERTA_CONTOLE_ENDERECO_I2C  ; Chama sub-rotina acerta o controle
                                             ; e o endereco memória 24LC1025
                                             ; [para até 4 CIs ou 4x128kBytes].
    ;-------------------------------------------------------------------------------------------------------------
;--------------------Lê a 7a posição que tem o valor de ST_9-----------------------------------------------

    PAGESEL     LER_EEPROM_I2C
    CALL        LER_EEPROM_I2C               ; Chama rotina de leitura 24LC1025
    MOVLW       .10
    PAGESEL     DELAY_MS
    CALL        DELAY_MS                     ; Garante tempo de escrita (10ms).

    MOVF        DADO_I2C,W                   ; W ← DADO_I2C
    MOVWF       ST_9                         ; ST_9 ← W

    PAGESEL     HIGH_ENDERECO_LOW_ASCII
    CALL        HIGH_ENDERECO_LOW_ASCII
    PAGESEL     ACERTA_A1_A0_B0_ASCII
    CALL        ACERTA_A1_A0_B0_ASCII

    MOVF        L2_C6,W                      ; W ← L2_C6
    MOVWF       ENDERECO_I2C_11              ; ENDERECO_I2C_11 ← w
    MOVF        L2_C5,W                      ; W ← L2_C5
    MOVWF       ENDERECO_I2C_21              ; ENDERECO_I2C_21 ← w
    MOVF        L2_C4,W                      ; W ← L2_C4
    MOVWF       ENDERECO_I2C_31              ; ENDERECO_I2C_31 ← w
    MOVF        L2_C3,W                      ; W ← L2_C3
    MOVWF       ENDERECO_I2C_41              ; ENDERECO_I2C_41 ← w
    MOVF        L2_C2,W                      ; W ← L2_C2
    MOVWF       ENDERECO_I2C_51              ; ENDERECO_I2C_51 ← w

    PAGESEL     ACERTA_CONTOLE_ENDERECO_I2C
    CALL        ACERTA_CONTOLE_ENDERECO_I2C  ; Chama sub-rotina acerta o controle
                                             ; e o endereço memória 24LC1025
                                             ; [para até 4 CIs ou 4x128kBytes].
    ;-------------------------------------------------------------------------------------------------------------
    RETURN
```

b37) Sub-rotinas *assembly* que controlam a leitura e gravação do CI 24LC1025

Instalou-se na PCI do microcontrolador o banco de memórias seriais composto por até 4 memórias 24LC1025.

Para realizar a gravação de um byte em uma posição endereçada da memória 24LC1025 deve-se adotar a seguinte sequência:

1) Enviar *START* BIT (que é o mesmo que SCL=1, e SDA variando de 1 para 0),
2) Preparar e enviar o byte de controle, o qual é composto por 1010 B0 A1 A0 R/W=0 (quando B0=0, está selecionando os primeiros 64kBytes da memória 24LC1025, cuja capacidade total é de 128kbytes),
3) Chamar a sub-rotina TESTAR_ACK_I2C,
4) Enviar os 8 bits mais significativos do endereço,
5) Chamar a sub-rotina TESTAR_ACK_I2C,
6) Enviar os 8 bits menos significativos do endereço,
7) Chamar a sub-rotina TESTAR_ACK_I2C,
8) Carregar o registrador SSPBUF com o dado a ser gravado na memória 24LC1025,
9) Chamar a sub-rotina TESTAR_ACK_I2C,
10) Enviar *STOP* BIT→ Fim desta transmissão (que é o mesmo que SCL=1, e SDA variando de 0 para 1).

O fluxograma da Fig. A58 mostra as atividades necessárias à implementação da sub-rotina que controla a gravação de um byte em uma posição endereçada da memória 24LC1025.

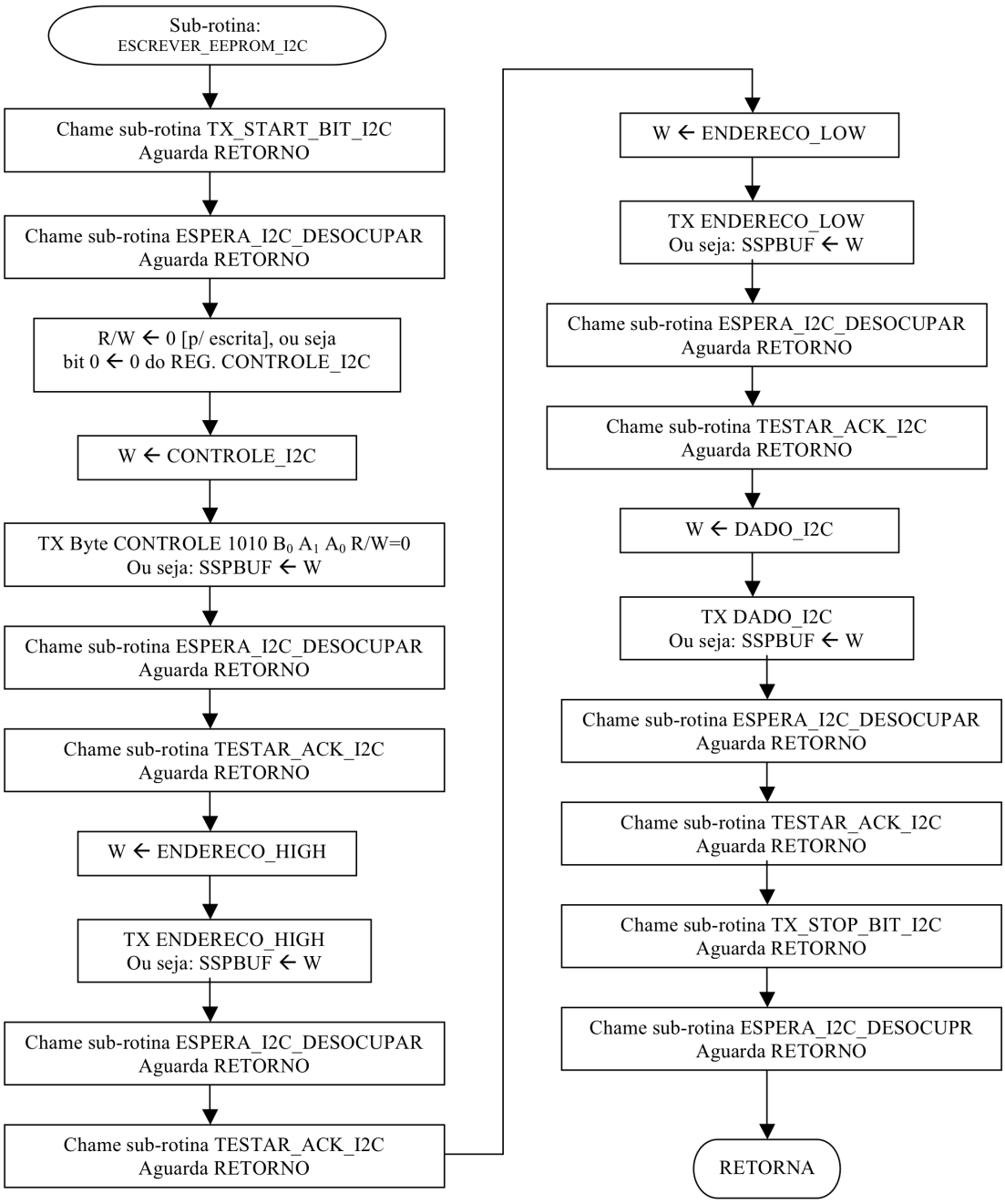


Figura A58 – Fluxograma da Sub-rotina: ESCREVER_EEPROM_I2C

```
;*********************************************************************
;*     SUB-ROTINA PARA ESCREVER UM BYTE NA EEPROM SERIAL 24LC1025                *
;*********************************************************************
ESCREVER_EEPROM_I2C
        CALL        TX_START_BIT_I2C        ; Gera o bit Start no barramento I2C.
        CALL        ESPERA_I2C_DESOCUPAR    ; Espera desocupar o barramento I2C.
        BCF         CONTROLE_I2C,0          ; Bit R/W=0 para escrever dado na memória.
        MOVF        CONTROLE_I2C,W          ; W ← CONTROLE_I2C valor atual do
                                            ; CONTROLE_I2C
        MOVWF       SSPBUF                  ; Transmitir o byte controle 1010 B0 A1 A0
                                            ; R/W=0
        CALL        ESPERA_I2C_DESOCUPAR    ; Espera desocupar o barramento I2C.
        CALL        TESTAR_ACK_I2C          ; Chama sub-rotina para testar se RX ACK ok.
        MOVF        ENDERECO_HIGH,W         ; W ← ENDERECO_HIGH
        MOVWF       SSPBUF                  ; TX ao Slave o ENDERECO_HIGH
        CALL        ESPERA_I2C_DESOCUPAR    ; Espera desocupar o barramento I2C.      CALL
        TESTAR_ACK_I2C                      ; Chama sub-rotina para testar se RX ACK ok.
        MOVF        ENDERECO_LOW,W          ; W ← ENDERECO_LOW.
        MOVWF       SSPBUF                  ; Transmitir ao Slave ENDERECO_LOW.
        CALL        ESPERA_I2C_DESOCUPAR    ; Espera desocupar o barramento I2C.
        CALL        TESTAR_ACK_I2C          ; Chama sub-rotina para testar se RX ACK ok.
        MOVF        DADO_I2C,W              ; Carregue o dado a ser gravado  em W
        MOVWF       SSPBUF                  ; TX ao Slave o conteúdo de DADO_I2C.
        CALL        ESPERA_I2C_DESOCUPAR    ; Espera desocupar o barramento I2C.      CALL
        TESTAR_ACK_I2C                      ; Chama sub-rotina para testar se RX ACK ok.
        CALL        TX_STOP_BIT_I2C         ; TX o bit Stop para fim desta comunicação
        CALL        ESPERA_I2C_DESOCUPAR    ; Espera desocupar o barramento I2C.

        RETURN                              ; Retorna ao Programa Principal.

;*********************************************************************
```

Observa-se que a sub-rotina de gravação de um byte na memória chama (instrução CALL) várias outras sub-rotinas, as quais, em conjunto, suportam a referida gravação.

Na sequência foi apresentado o fluxograma da Fig. A59 e a sub-rotina que envia o start bit.

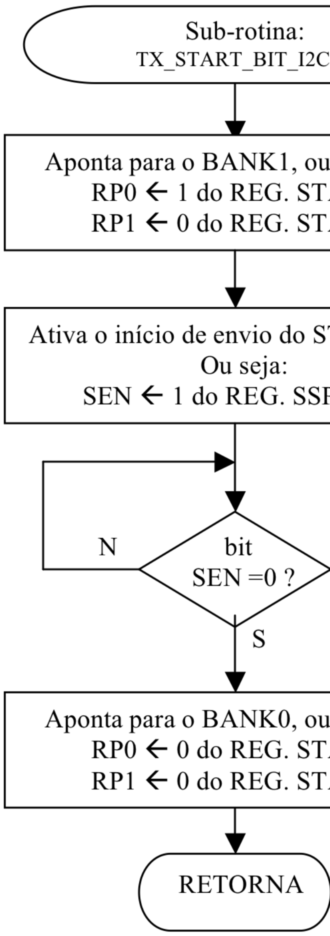


Figura A59 – Fluxograma da Sub-rotina: TX_START_BIT_I2C

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * ESTA SUBROTINA GERA UM START BIT E ESPERA O SEU TÉRMINO - NO I2C BUS.          *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

TX_START_BIT_I2C
        BANK1                           ; Aponta para o BANK1 da RAM
        BSF             SSPCON2,SEN     ; Ativa o início do envio do START BIT (Master → Slave)
        BTFSC           SSPCON2,SEN     ; Verifica se processo de envio do START BIT já terminou.
        GOTO            $-1             ; Não terminou. Testa novamente o BIT SEN do REG. SSPCON2
        BANK0                           ; Sim, terminou. Aponta para o BANK0 da RAM.
        RETURN                          ; Retorna ao Programa Principal.

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

O fluxograma da Fig. A60 contém as atividade a serem implementadas para acertar o campo CONTROLE I^2C, ENDERECO_LOW e ENDERECO_HIGH. Na sequência foi mostrado o resultado da sub-rotina que suporta acertar estes campos.

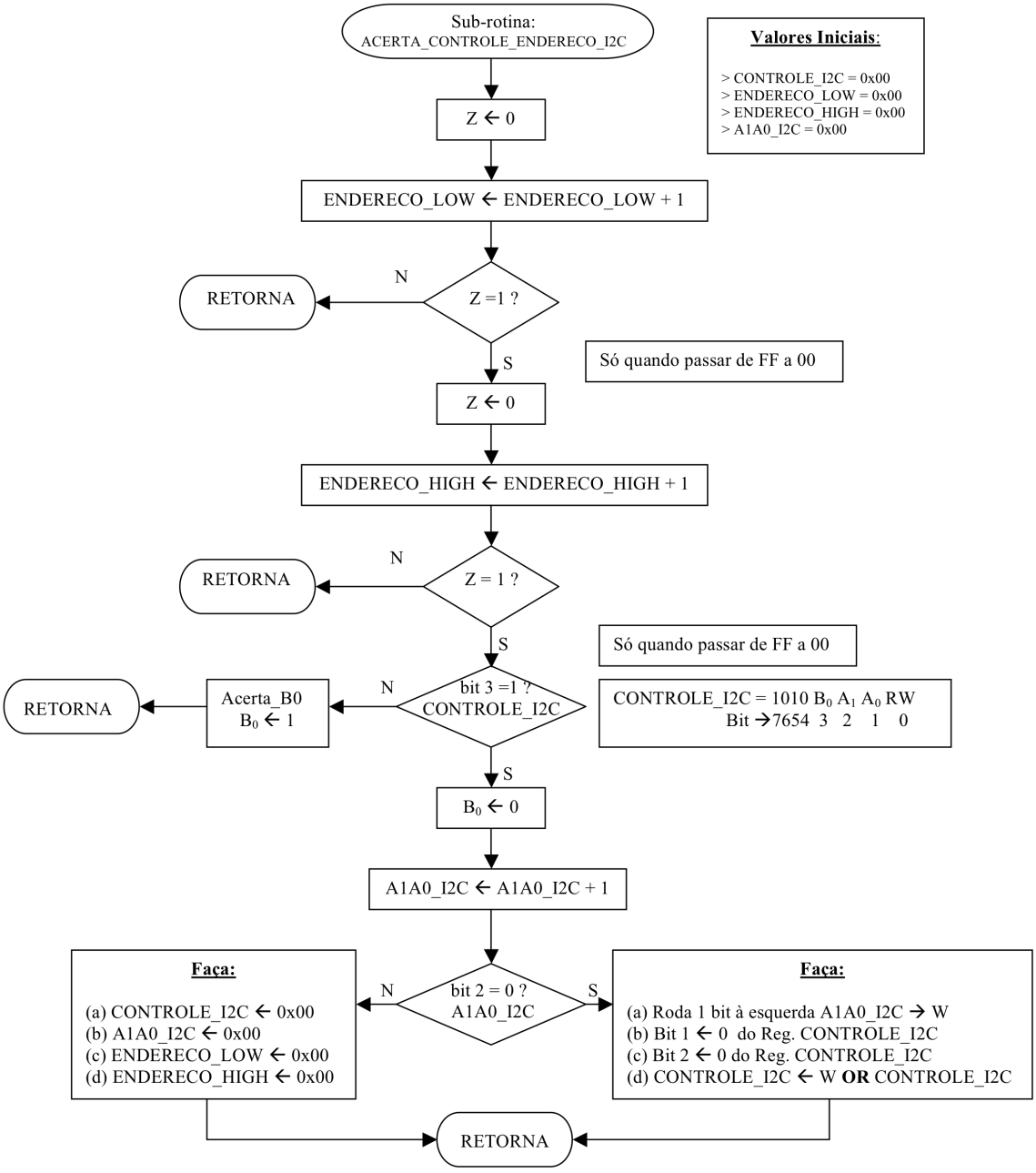


Figura A60 – Fluxograma da Sub-rotina: ACERTA_CONTROLE_ENDERECO_I2C

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * ESTA SUBROTINA ACERTA O CONTROLE [MENOS SEU BIT 0] E O ENDERECO I2C               *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

ACERTA_CONTOLE_ENDERECO_I2C             ; Registradores que já vem com valores prévios
                                        ; Valores iniciais: CONTROLE_I2C = 0xA0
                                        ;               ENDERECO_LOW  = 0x00
                                        ;               ENDERECO_HIGH = 0x00
        BCF         STATUS,Z            ; Para garantir que bit Z=0
        INCF        ENDERECO_LOW,F      ; Quando ENDERECO_LOW=0x00 → Z=1
        BTFSS       STATUS,Z            ; Se Z=1 Salta RETURN
        RETURN                          ; Se Z=0 Retorna, pois outros dados CONTROLE e
                                        ; ENDERECO já se encontram certos.
        BCF         STATUS,Z            ; Reseta bit Z do Registrador STATUS
        INCF        ENDERECO_HIGH,F     ; ENDERECO_HIGH ← ENDERECO_HIGH+1
        BTFSS       STATUS,Z            ; Se Z=1 Salta RETURN
        RETURN                          ; Se Z=0 Retorna, pois outros dados CONTROLE e
                                        ; ENDERECO já se encontram certos.
        BTFSS       CONTROLE_I2C,3      ; Testa o bit 3 do REG CONTROLE_I2C [o B0,
                                        ; Quando B0=0 aponta para o primeiro bloco de 64K
                                        ; bytes da 24LC1025, e quando B0=1 aponta para o
                                        ; último bloco de 64K do dispositivo 24LC1025].
        GOTO        ACERTA_B0           ; Se B0=0 ajusta B0 para 1
        BCF         CONTROLE_I2C,3      ; Quando END passar de 1 1111 1111 1111 1111 para
                                        ;                     0 0000 0000 0000 0000
        INCF        A1A0_I2C,F          ; Muda-se de dispositivo físico 24LC1025
                                        ; incrementando A1A0_I2C
        BTFSC       A1A0_I2C,2
        GOTO        INICIA_CONTROLE_END
        RLF         A1A0_I2C,W          ; Acerta a posição dos bits A1 E A0
        BCF         CONTROLE_I2C,1      ; Reseta o bit 1 do registrador CONTROLE_I2C
        BCF         CONTROLE_I2C,2      ; Reseta o bit 2 do registrador CONTROLE_I2C
        IORWF       CONTROLE_I2C,F      ; Concentra resultado CONTROLE_I2C e A1A0_I2C
                                        ;  em CONTROLE_I2C. Neste momento só falta acertar
                                        ; o bit 0 do CONTROLE_I2C [bit R/W=1 p/ leitura e
                                        ; R/W=0 para escrita].

        RETURN                          ; Retorna ao Programa Principal.

ACERTA_B0
        BSF         CONTROLE_I2C,3      ; Qdo END passar de     0 1111 1111 1111 1111
                                        ;                       1 0000 0000 0000 0000 →
                                        ;                       2º bloco de 64k bytes.

        RETURN                          ; Retorna ao Programa Principal.

INICIA_CONTROLE_END
        MOVLW       0xA0                ; W ← 0xA0
        MOVWF       CONTROLE_I2C        ; CONTROLE_I2C ← W
        CLRF        A1A0_I2C            ; A1A0_I2C ← 0x00
        CLRF        ENDERECO_LOW        ; ENDEECO_LOW ← 0x00
        CLRF        ENDERECO_HIGH       ; ENDERECO_HIGH ← 0x00
        RETURN                          ; Retorna ao Programa Principal.
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

Foi apresentado, na sequência, o fluxograma da Fig. A61 e a sub-rotina que espera o barramento I^2C ficar livre.

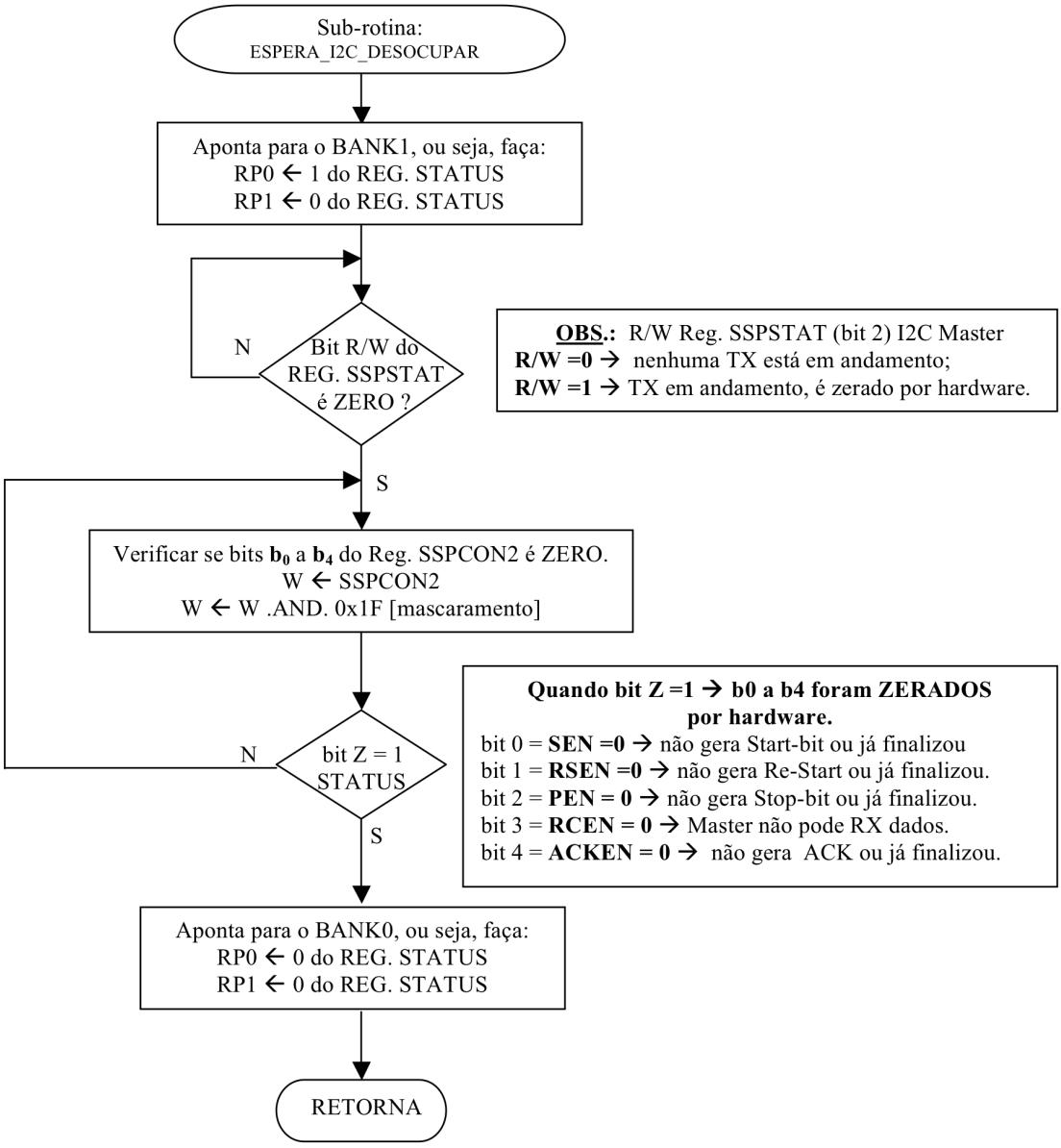


Figura A61 – Fluxograma da Sub-rotina: ESPERA_I2C_DESOCUPAR

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * SUBROTINA ESPERA TERMINAR TODA ATIVIDADE NO I2C BUS, ou I2C está livre[IDLE]
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

ESPERA_I2C_DESOCUPAR            ; Rotina baseada na AN735 "www.microchip.com"
                                ; com adaptações nas descrições.
        BANK1                   ; Aponta para o BANK1 da RAM.
        BTFSC   SSPSTAT,R_W     ; Alguma TX ou RX encontra-se em andamento? Ou R_W=1?
        GOTO    $-1             ; Sim, fazer teste do bit R_W até ele ser zero.
        MOVF    SSPCON2,W       ; W ←SSPCON2
        ANDLW   0x1F            ; Mascara SSPCON2. Para verificar quando b0 a b4 for ZERO.
                                ;----------------------Bits do Reg. SSPCON2 -------------------------
                                ; GCEN ACKSTAT ACKDT ACKEN RCEN PEN RSEN SEN
                                ;--------------------------------------------------------------------------
                                ; GCEN=0      →  Endereço global desabilitado,
                                ; ACKSTAT=0 → ACK recebido do Slave,
                                ; ACKDT=0     → continua recebendo mais bytes do Slave,
                                ; ACKEN=1     → inicia condição e indica condição ainda
                                ;                  em execução. … zerado por hardware.
                                ; RCEN=1      → Master pronto p/receber dados  do Slave.
                                ; PEN=1       → inicia condição e indica condição ainda
                                ;                  em execução. … zerado por hardware.
                                ; RCEN=1      → inicia condição e indica condição ainda
                                ;                  em execução. … zerado por hardware.
                                ; SEN=1       → inicia condição e indica condição ainda
                                ;                  em execução. … zerado por hardware.
                                ;-----------------------------------------------------------------------
        BTFSS   STATUS,Z        ; Quando Z=1 → resultou zero entre (SSPCON2 AND W) ou seja
                                ;             os bits em 1 acima, agora são zero que é o
                                ;             mesmo que o barramento I2C ter ficado livre
        GOTO    $-3             ; Se Z=0 → retorna 3 instruções [faz, MOVF SSPCON2,W]
        BANK0                   ; Se Z=1 → aponta para o BANK0 da RAM
        RETURN                  ; Retorna ao Programa Principal.
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

Foi apresentado, na sequência, o fluxograma da Fig. A62 e a sub-rotina que testa o recebimento do bit ACK, de reconhecimento.

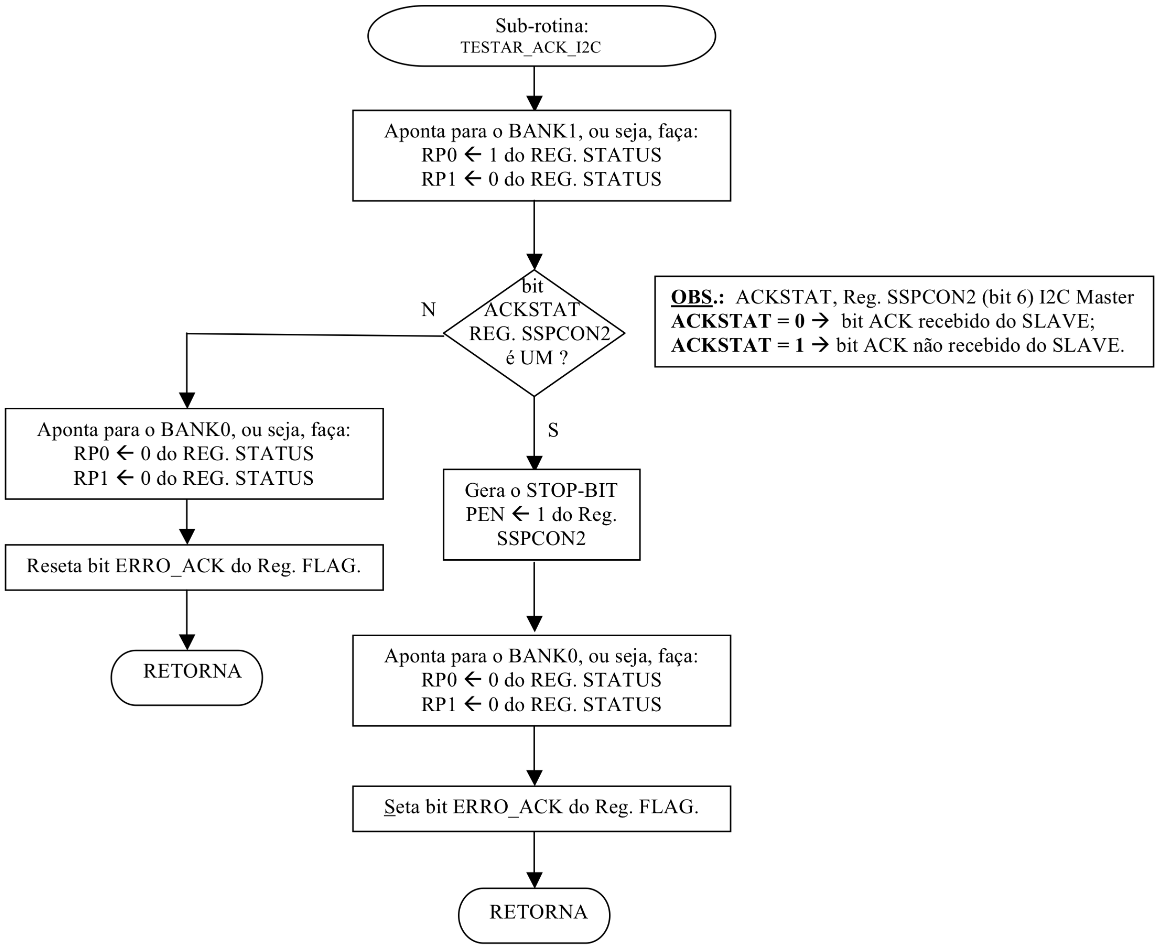


Figura A62 – Fluxograma da Sub-rotina: TESTAR_ACK_I2C

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * ESTA SUBROTINA TESTA SE O ACK FOI BEM RECEBIDO OU NAO PELO MASTER              *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

TESTAR_ACK_I2C                          ; Rotina baseada na AN735 "www.microchip.com"
                                        ; (ADAPTADA)
        BANK1                           ; Posiciona o ponteiro para o BANK1.
        BTFSS   SSPCON2,ACKSTAT         ; ACKSTAT=1 → recebeu NACK, [ERRO_ACK]
        GOTO    ACK_OK                  ; ACKSTAT=0 → recebeu ACK OK, GOTO ACK_OK.
        BSF     SSPCON2,PEN             ; Gera o STOP BIT
        BANK0                           ; Posiciona o ponteiro para o BANK0.
        BSF     ERRO_ACK                ; Seta bit ERRO_ACK do REG. FLAG p/indicar erro.
        RETURN                          ; Retorna ao Programa Principal.
ACK_OK
        BANK0                           ; Posiciona o ponteiro para o BANK0.
        BCF     ERRO_ACK                ; Reseta bit ERRO_ACK REG. FLAG indica ACK OK.
        RETURN                          ; Retorna ao Programa Principal.
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

Foi apresentado, na sequência, o fluxograma da Fig. A63 e a sub-rotina que gera e transmite o *stop bit.*

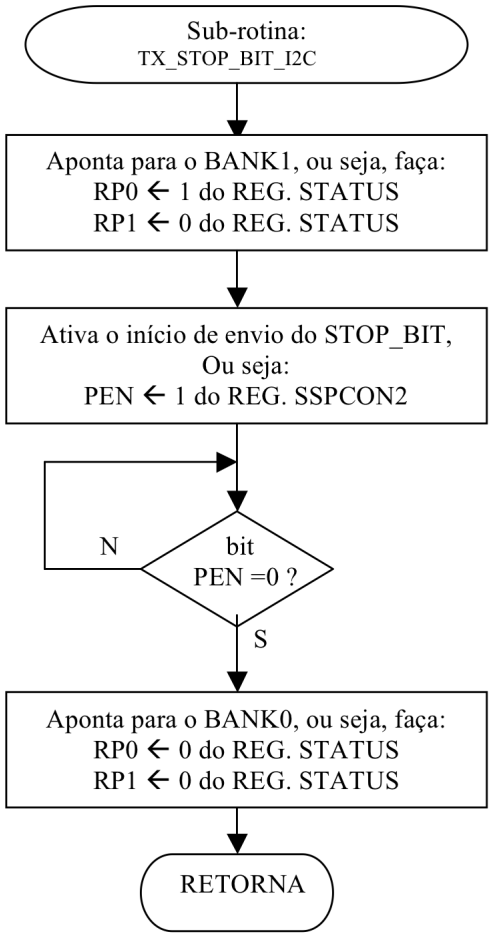


Figura A63 – Fluxograma da Sub-rotina: TX_STOP_BIT_I2C

```
;*****************************************************************
; * SUBROTINA GERA UM STOP BIT E ESPERA O SEU TÉRMINO - NO BARRAMENTO I2C.     *
;*****************************************************************
TX_STOP_BIT_I2C
        BANK1                       ; Aponta para o BANK1 da RAM.
        BSF             SSPCON2,PEN ; Ativa o início do envio do STOP BIT (Master → Slave)
        BTFSC           SSPCON2,PEN ; Verifica se o processo de envio do STOP BIT já terminou
        GOTO            $-1         ; Não terminou. Testa novamente o BIT PEN do REG. SSPCON2
        BANK0                       ; Sim, terminou. Aponta para o BANK0 da RAM.
        RETURN                      ; Retorna ao Programa Principal.
;*****************************************************************
```

Agora, para realizar a leitura de uma posição endereçada da memória 24LC1025 deve-se seguir a seguinte sequência:

1) Enviar o *START* BIT (que é o mesmo que SCL=1, e SDA variando de 1 para 0),
2) Preparar e enviar o byte de controle, composto por 1010 B0 A1 A0 R/W=0 (quando B0=0, está selecionando os primeiros 64kBytes da memória 24LC1025, cuja capacidade total é de 128kbytes),
3) Chamar a sub-rotina TESTAR_ACK_I2C,
4) Enviar os 8 bits mais significativos do endereço,
5) Chamar a sub-rotina TESTAR_ACK_I2C,
6) Enviar os 8 bits menos significativos do endereço,
7) Chamar a sub-rotina TESTAR_ACK_I2C,
8) Enviar o controle, agora com o bit R/W=1 (leitura),
9) Chamar a sub-rotina TESTAR_ACK_I2C,
10) Setar o bit RCEN, indicando que o Máster está pronto para receber do *Slave*,
11) Aguardar receber dado vindo da posição endereçada da memória;
12) Chamar a sub-rotina TX_NACK_I2C,
13) Enviar o *STOP* BIT→ Fim desta transmissão (que é o mesmo que SCL=1, e SDA variando de 0 para 1).

Foi apresentado, na sequência, o fluxograma da Fig. A64 e a sub-rotina que lê um byte armazenado na memória EEPROM..

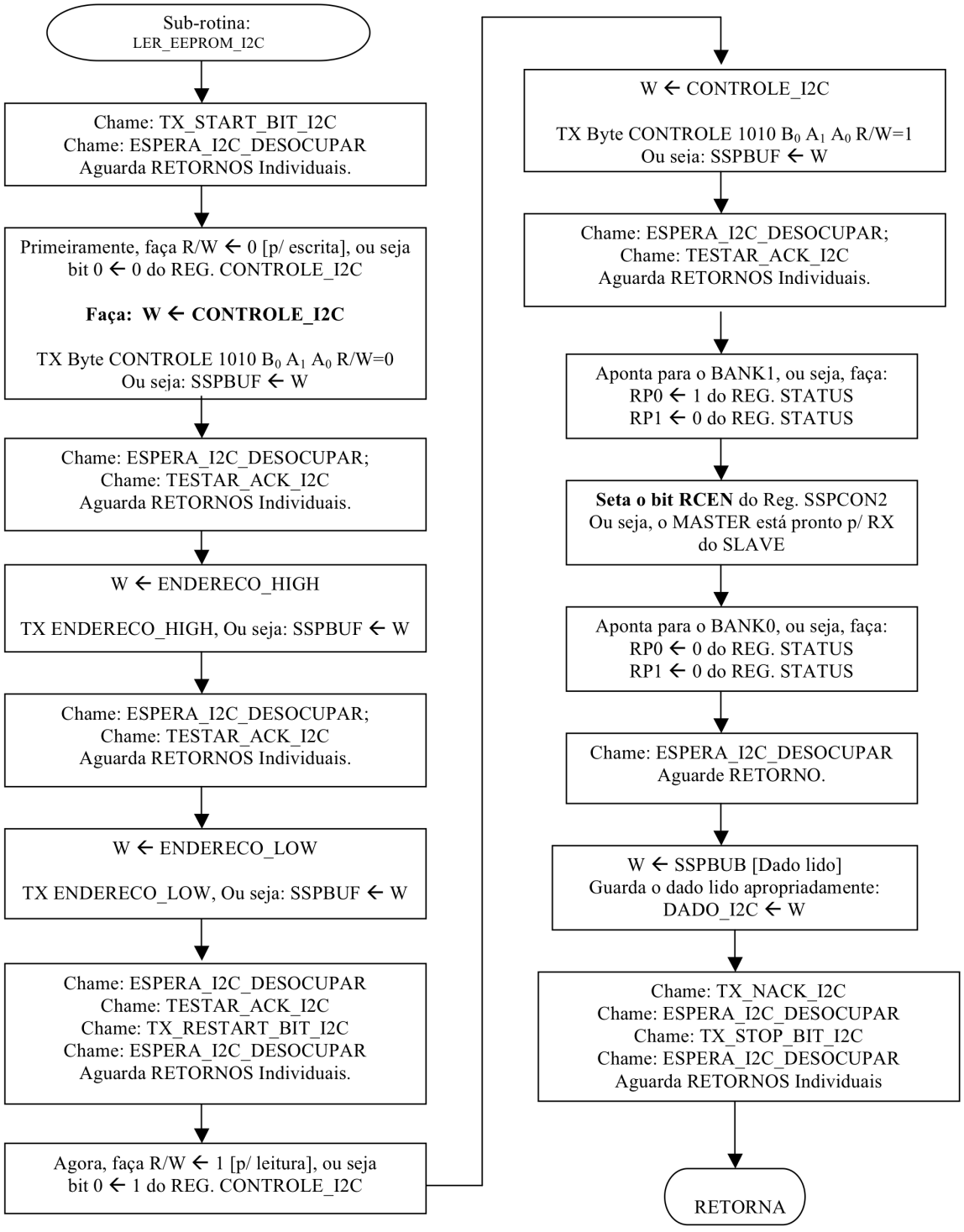


Figura A64 – Fluxograma da Sub-rotina: LER_EEPROM_I2C

```
;*********************************************************************
;*     SUBROTINA PARA LEITURA DE UM BYTE DA EEPROM SERIAL 24LC1025    *
;*********************************************************************

LER_EEPROM_I2C
        CALL        TX_START_BIT_I2C        ; Gera o bit Start no barramento I2C.
        CALL        ESPERA_I2C_DESOCUPAR    ; Espera desocupar o barramento I2C.
        BCF         CONTROLE_I2C,0          ; Inicialmente, reseta bit R/W.
        MOVF        CONTROLE_I2C,W          ; W ← CONTROLE_I2C [valor atual]
        MOVWF       SSPBUF                  ; TX o byte controle 1010 B0 A1 A0 R/W=0
        CALL        ESPERA_I2C_DESOCUPAR    ; Espera desocupar o barramento I2C.
        CALL        TESTAR_ACK_I2C          ; Chama sub-rotina para testar se RX ACK ok.
        MOVF        ENDERECO_HIGH,W         ; W ← ENDERECO_HIGH
        MOVWF       SSPBUF                  ; TX ao Slave o ENDERECO_HIGH
        CALL        ESPERA_I2C_DESOCUPAR    ; Espera desocupar o barramento I2C.
        CALL        TESTAR_ACK_I2C          ; Chama sub-rotina para testar se RX ACK ok.

        MOVF        ENDERECO_LOW,W          ; W ←ENDERECO_LOW
        MOVWF       SSPBUF                  ; TX ao Slave o ENDERECO_LOW
        CALL        ESPERA_I2C_DESOCUPAR    ; Espera desocupar o barramento I2C.
        CALL        TESTAR_ACK_I2C          ; Chama sub-rotina para testar se RX ACK ok.

        CALL        TX_RESTART_BIT_I2C      ; Gera e envia novo bit Start,
        CALL        ESPERA_I2C_DESOCUPAR    ; Espera desocupar o barramento I2C.
        BSF         CONTROLE_I2C,0          ; Bit R/W=1 para efetivar a leitura da memória
        MOVF        CONTROLE_I2C,W          ; W ← CONTROLE_I2C (valor atual)
        MOVWF       SSPBUF                  ; TX byte CONTROLE 1010 B0 A1 A0 R/W=1
        CALL        ESPERA_I2C_DESOCUPAR    ; Espera desocupar o barramento I2C.
        CALL        TESTAR_ACK_I2C          ; Chama sub-rotina para testar se RX ACK ok.

        BANK1                               ; Aponta para o BANK1 da RAM
        BSF         SSPCON2,RCEN            ; RCEN=1 → Mestre pronto para RX do Slave.
        BANK0                               ; Volta a apontar para o BANK0 da RAM
        CALL        ESPERA_I2C_DESOCUPAR    ; Espera desocupar o barramento I2C.
        MOVF        SSPBUF,W                ; W ← SSPBUF
        MOVWF       DADO_I2C                ; Salva o dado lido em DADO_I2C
        CALL        TX_NACK_I2C             ; Chama sub-rotina TX NACK I2C
        CALL        ESPERA_I2C_DESOCUPAR    ; Espera desocupar o barramento I2C.
        CALL        TX_STOP_BIT_I2C         ; TX o bit Stop para fim desta comunicação
        CALL        ESPERA_I2C_DESOCUPAR    ; Espera desocupar o barramento I2C.

        RETURN                              ; Retorna ao Programa Principal.
;*********************************************************************
```

Além das sub-rotinas usadas na gravação de um byte em um endereço da 24LC1025, precisa-se de algumas outras sub-rotinas pra a realização da leitura de um byte gravado nesta memória. Estas sub-rotinas, complementares, são a que envia o restart bit e a que envia o NACK bit.

Foi apresentado na sequência, o fluxograma da Fig. A65 e a sub-rotina que envia o bit restart.

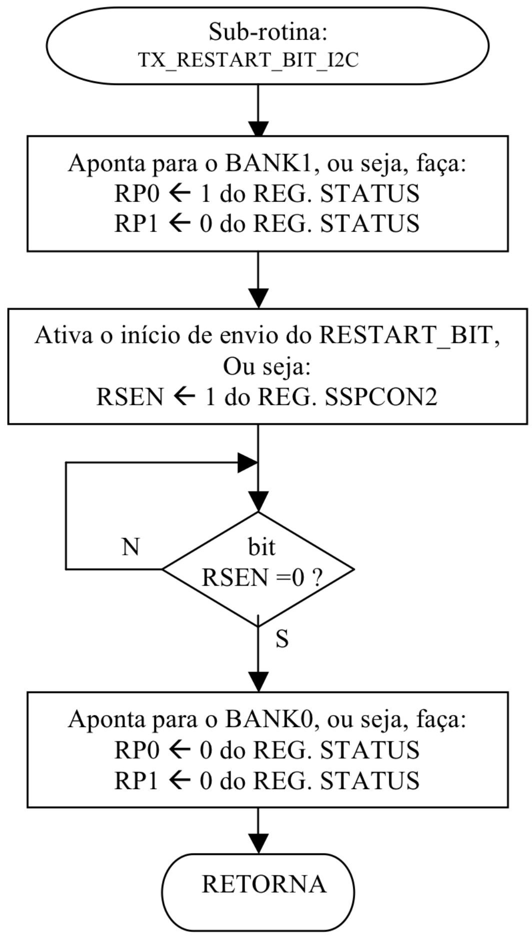


Figura A65 – Fluxograma da Sub-rotina: TX_RESTART_BIT_I2C

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * ESTA SUBROTINA GERA UM RESTART BIT E ESPERA SEU TÉRMINO - NO I2C BUS .          *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

TX_RESTART_BIT_I2C
        BANK1                           ; Aponta para o BANK1 da RAM.
        BSF         SSPCON2,RSEN; Ativa o início do envio do RESTART BIT (Master → Slave)
        BTFSC       SSPCON2,RSEN; Verifica se processo de envio do RESTART BIT já terminou.
        GOTO        $-1                 ; Não terminou. Testa novamente BIT RSEN do REG. SSPCON2
        BANK0                           ; Sim, terminou. Aponta para o BANK0 da RAM.
        RETURN                          ; Retorna.
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

Foi apresentado na sequência, o fluxograma da Fig. A66 e a sub-rotina que gera e envia um bit NACK

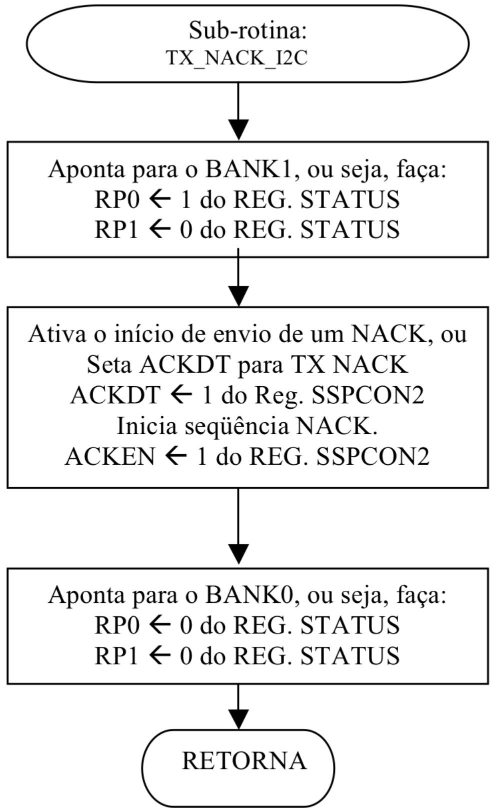


Figura A66 – Fluxograma da Sub-rotina: TX_NACK_I2C

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * ESTA SUBROTINA GERA UMA SEQUENCIA DE NACK NO BARRAMENTO I2C.                  *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

TX_NACK_I2C                                ; Rotina baseada na AN735 "www.microchip.com"
                                           ; (ADAPTADA)
        BANK1                              ; Aponta para o BANK1 da RAM.
        BSF         SSPCON2,ACKDT          ; Seta o bit ACKDT para TX NACK
        BSF         SSPCON2,ACKEN          ; Inicia a seqüência NACK
        BANK0                              ; Aponta para o BANK0 da RAM.
        RETURN                             ; Retorna ao Programa Principal
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

A Fig. A67 ilustra o instante da gravação do dado 0xCB no endereço 0x427CB e a leitura do dado 0x6D que estava armazenado no endereço 0x4006D.

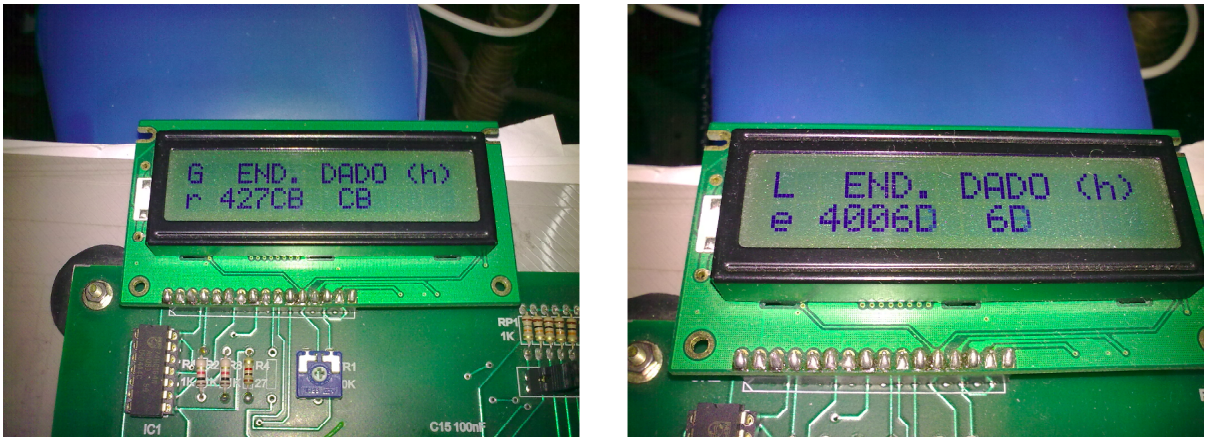


Figura A67 – Display LCD - gravação e leitura de dados na 24LC1025

ANEXO B – Roteiro para configurar o software Hyperterminal

Esta configuração possibilita o recebimento de dados enviados pelo PIC ao Computador Pessoal. Siga a sequência.

Iniciar → Programas → HiperTerminal Private Edition → HiperTerminal Private Edition [**clique**]

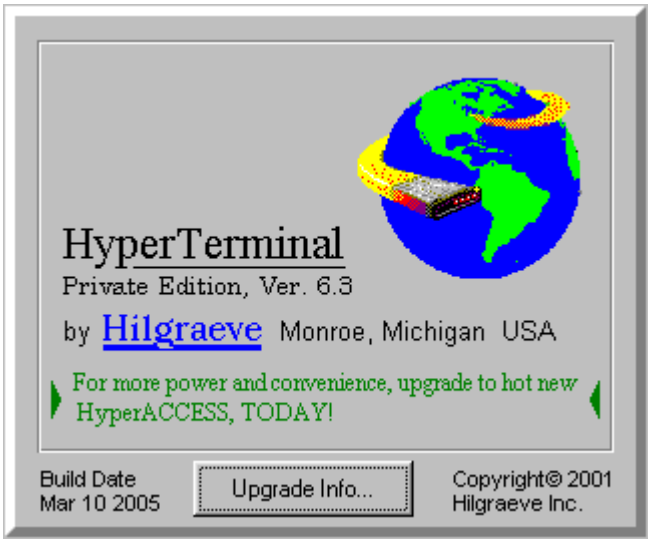


Figura A68 – Janela inicial do Hyperterminal

Aguarde e surgirá:

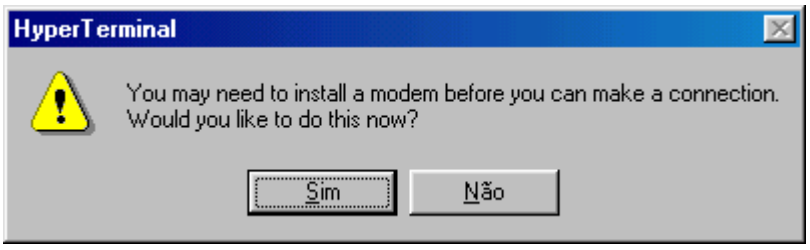


Figura A69 – Janela do Hyperterminal de inquirição instalação de um modem.

Responda [**Não**], pois não se trabalhará com MODEM. Então, surgirá:

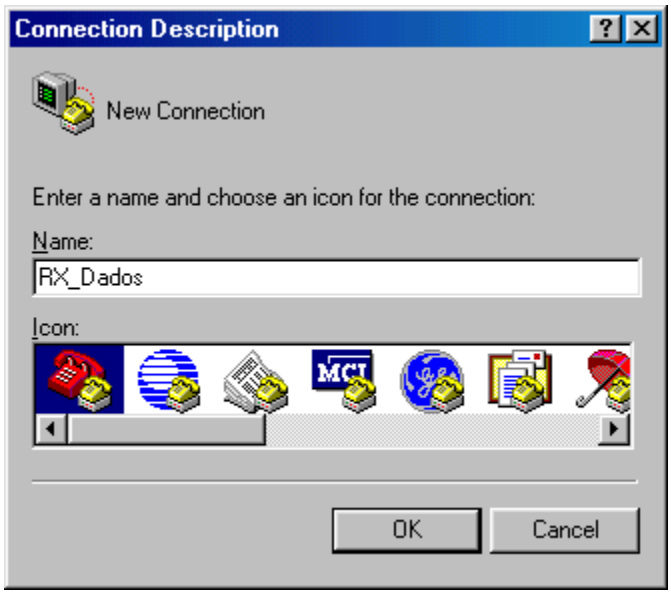


Figura A70 – Janela do Hyperterminal para nomear uma nova conexão.

Digite um Nome [**Name**] para uma nova conexão. Exemplo: RX_Dados [**OK**]

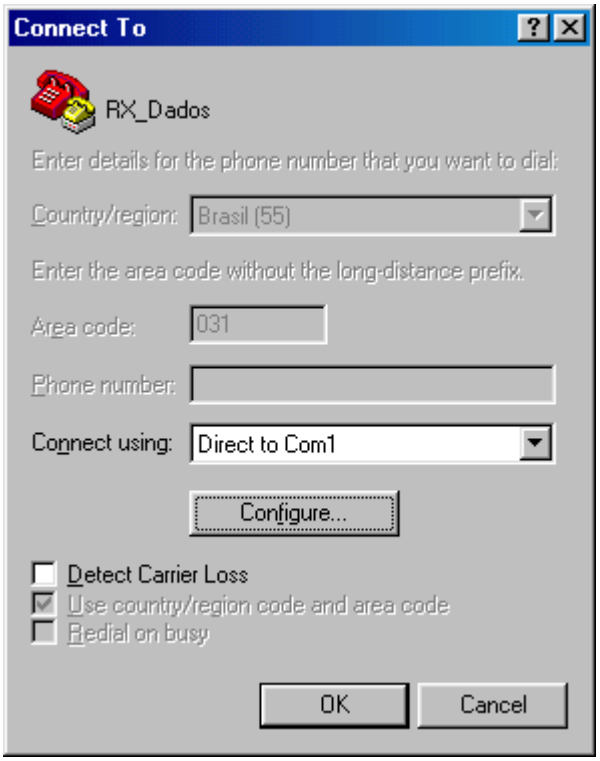


Figura A71 – Porta de comunicação utilizada.

Como a RS-232 está na COM1, aperte **Configure**:

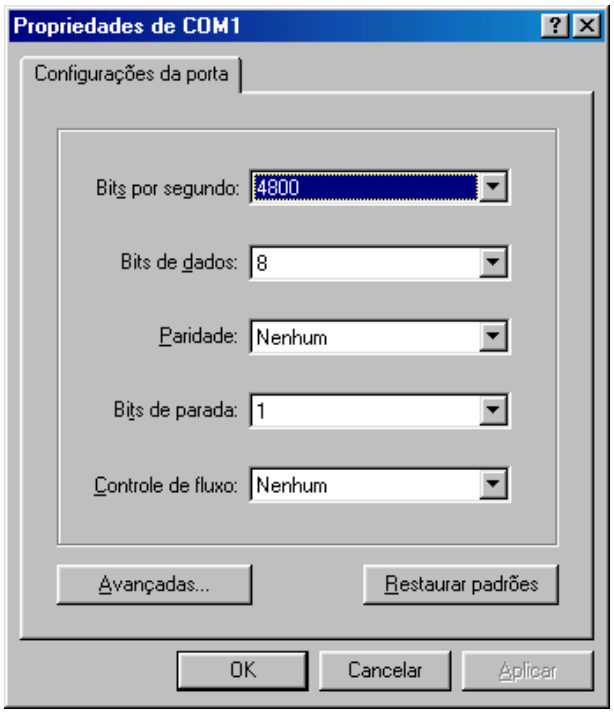


Figura A72 – Configurações da Porta COM1.

Acerte os parâmetros para a comunicação conforme mostrado acima e clique em 3x **OK**. Aparecerá a tela abaixo, o que indica já estar conectado.

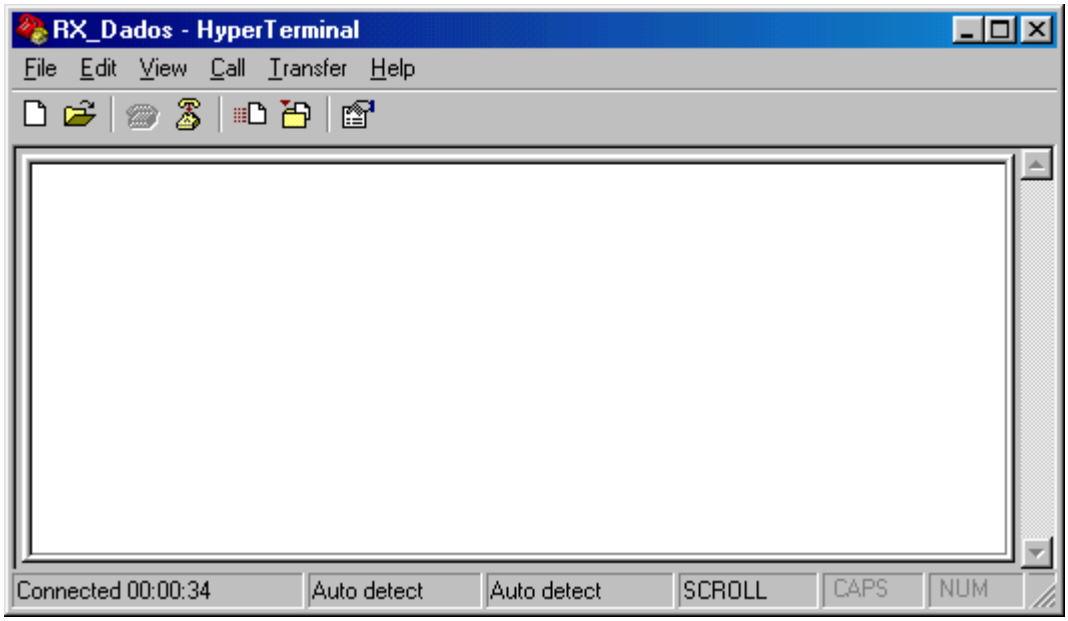


Figura A73 – Janela do Hyperterminal, já conectado.

Clique em: File → *Properties*

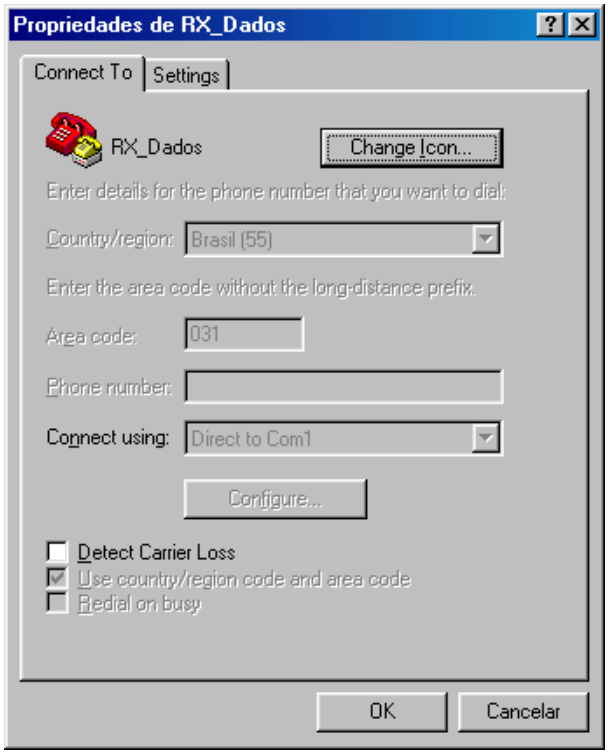


Figura A74 – Propriedades da conexão criada, RX_Dados.

Selecione **Settings**:

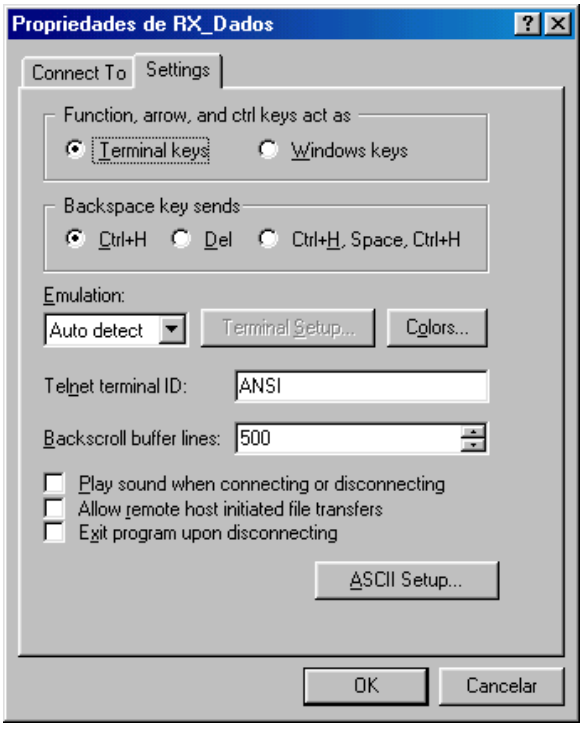


Figura A75 – Propriedades, *Settings*.

Clique em **ASCII Setup...**

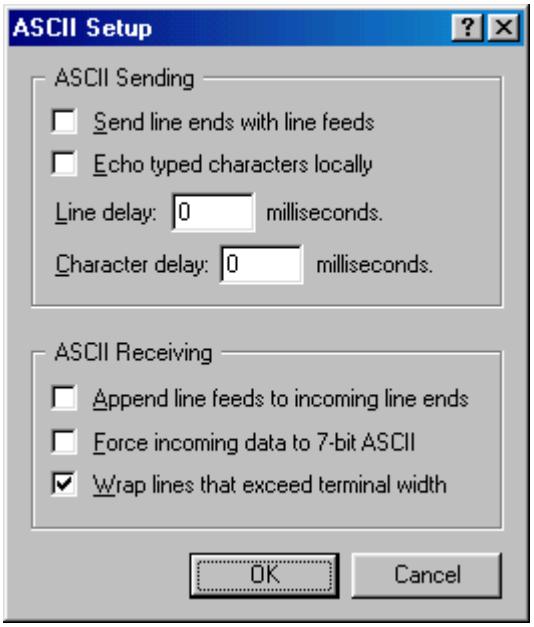


Figura A76 – ASCII *setup*.

Configurá-los conforme mostrado acima. 2x OK.

Preparar para capturar o texto, o qual, por exemplo, pode-se dar o nome de RX_Dados:

Seleciona: Transfer → Capture Text ...

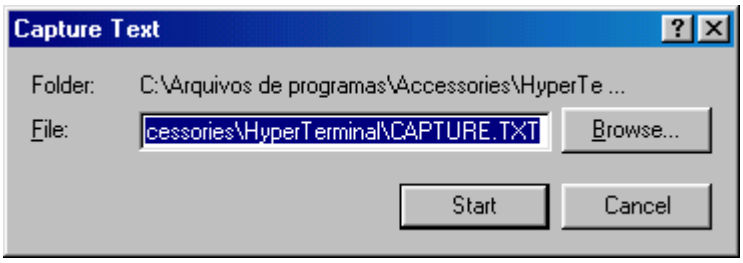


Figura A77 – Capturar Texto.

Troque o nome CAPTURE.TXT para RX_Dados.TXT

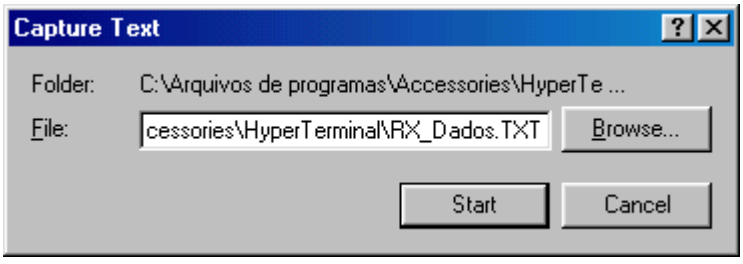


Figura A78 – Acerta o nome para RX_Dados.

O texto será capturado para:

C:\Arquivos de Programas|Accessories\HyperTerminal\RX_Dados.TXT

Clique em **Start** para iniciar a captura do texto.

Estes foram os primeiros dados transmitidos para a janela de recepção do HyperTerminal:

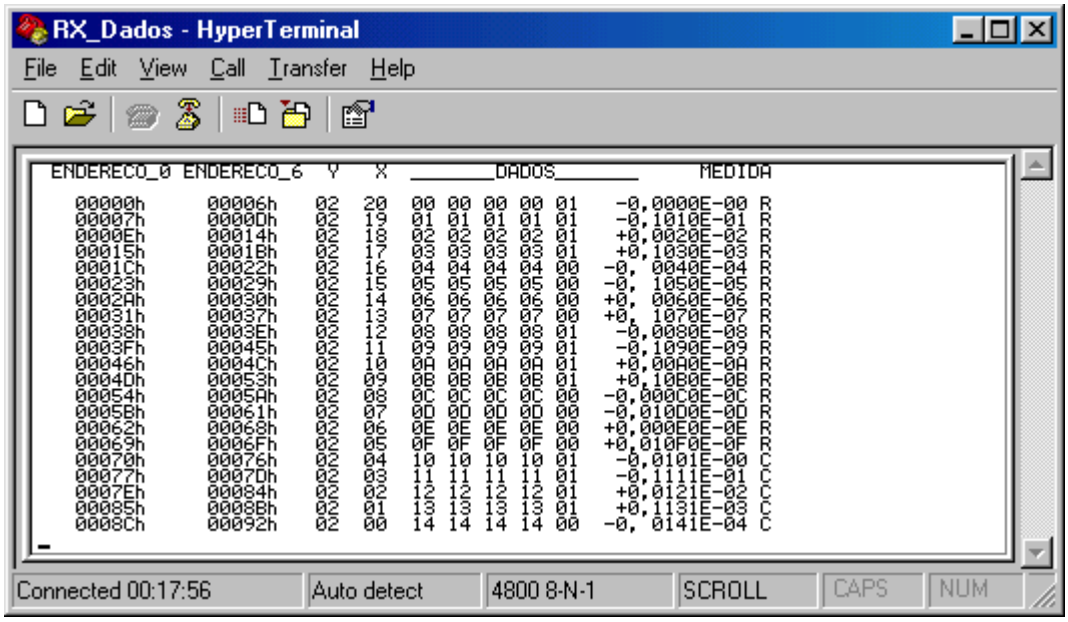


Figura A79 – Dados Recebidos pelo Heperterminal.

Para **encerrar a captura**, proceda da seguinte maneira:

Selecione: Transfer → Capture Text ... → Stop

Meu computador → Sitema (C:) → Arquivos de Programas → Accessories → HyperTerminal →

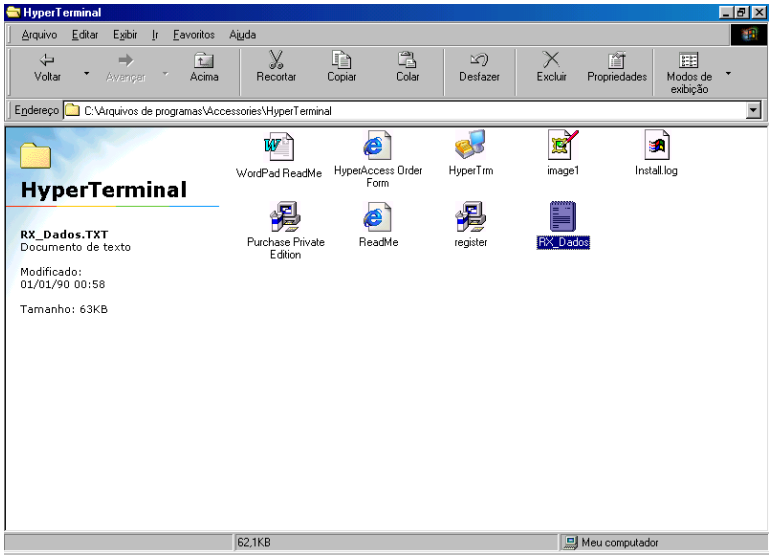


Figura A80 – Mostra do arquivo texto RX_Dados.

Tem-se, então, acesso ao arquivo capturado, que uma vez aberto, apresenta:

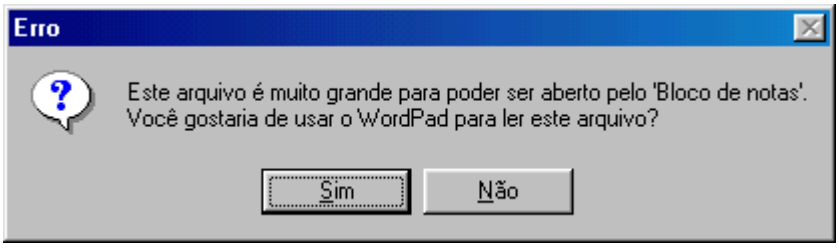


Figura A81 – Selecionar a abertura no WordPad

Clique em **Sim**.

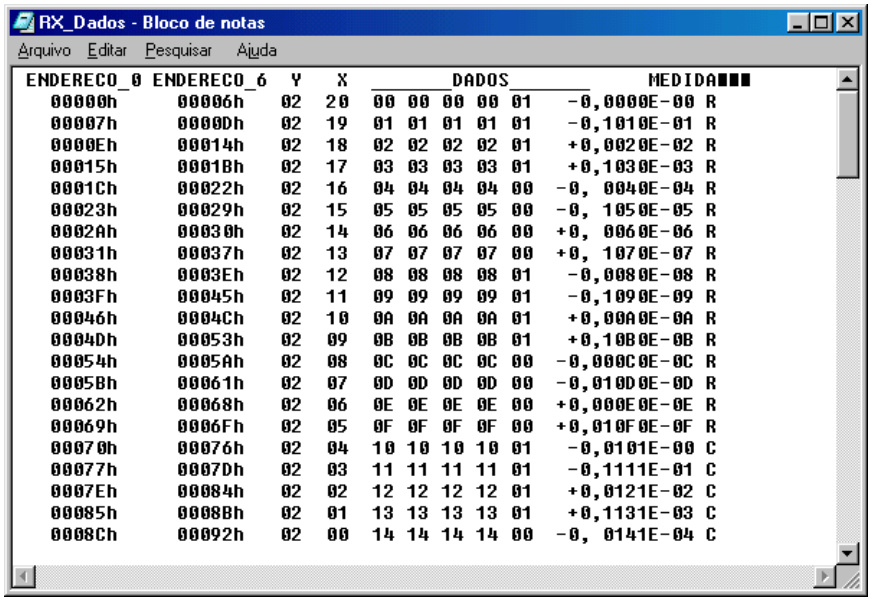


Figura A82 – Apresentação dos dados recebidos abertos na janela.

ANEXO C – Enviar arquivo texto à planilha eletrônica.

Seleciona todo o arquivo texto para enviá-lo à planilha eletrônica, exemplificado com o Excel da Microsoft.

Editar → Selecionar Tudo

Editar → Copiar

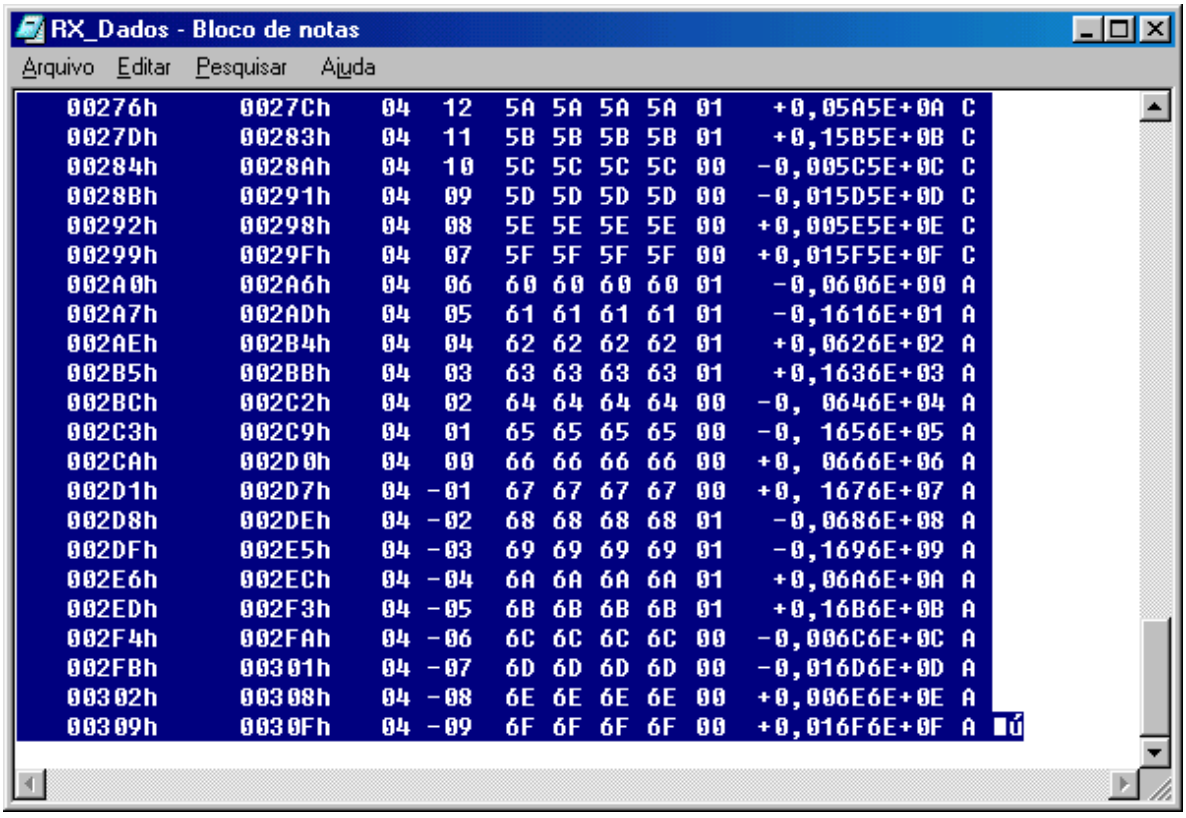


Figura A83 – Dados no arquivo texto selecionados.

Abra o Excel

Iniciar → Programas → Microsoft Excel

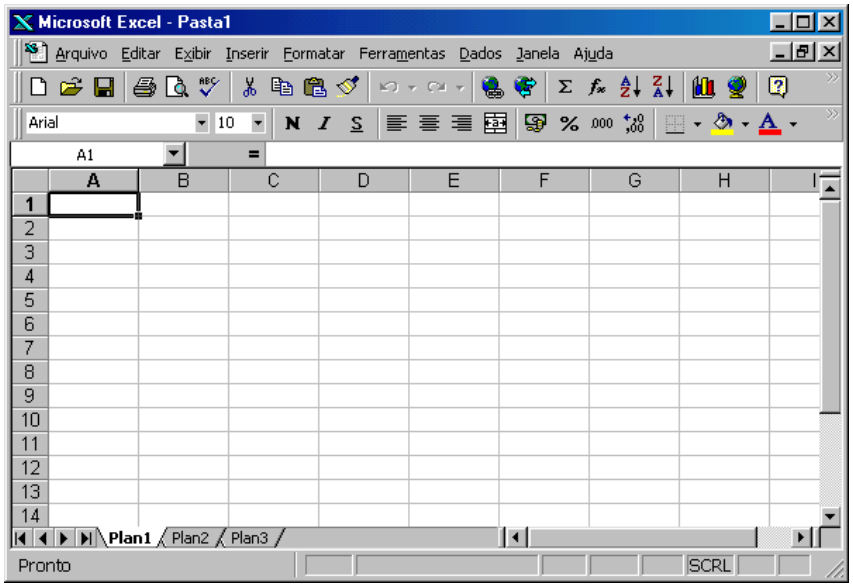


Figura A84 – Planilha Excel aberta.

No Excel, Clique em Editar → Colar, e teremos.

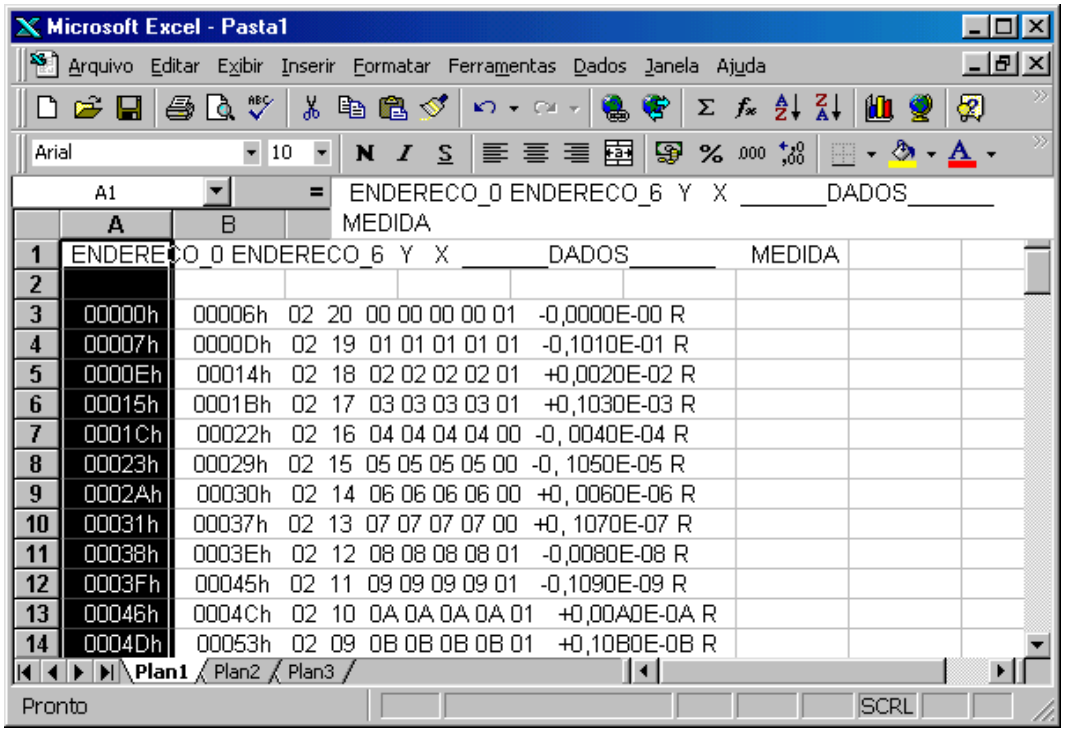


Figura A85 – Transferência dos dados para a planilha Excel.

Aparecerá uma informação totalmente desorganizada que poderá ser organizada, como:

Clique em Dados → Texto para colunas ... e teremos

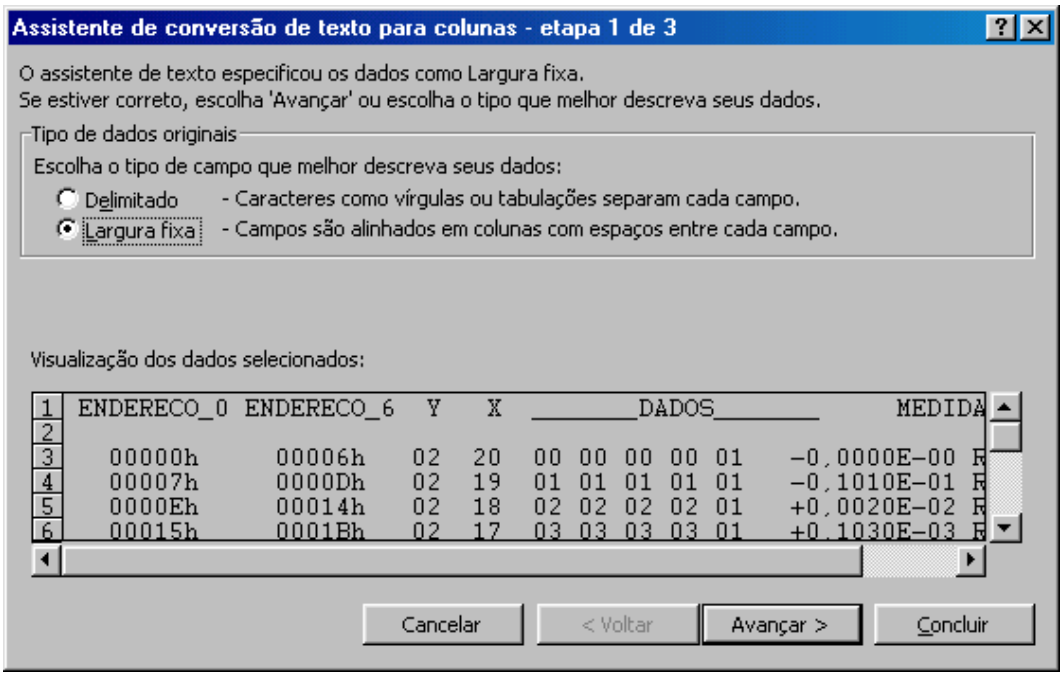


Figura A86 – Assistente de conversão de texto para colunas.

Clique em avançar e os dados já estarão em suas respectivas colunas, como abaixo:

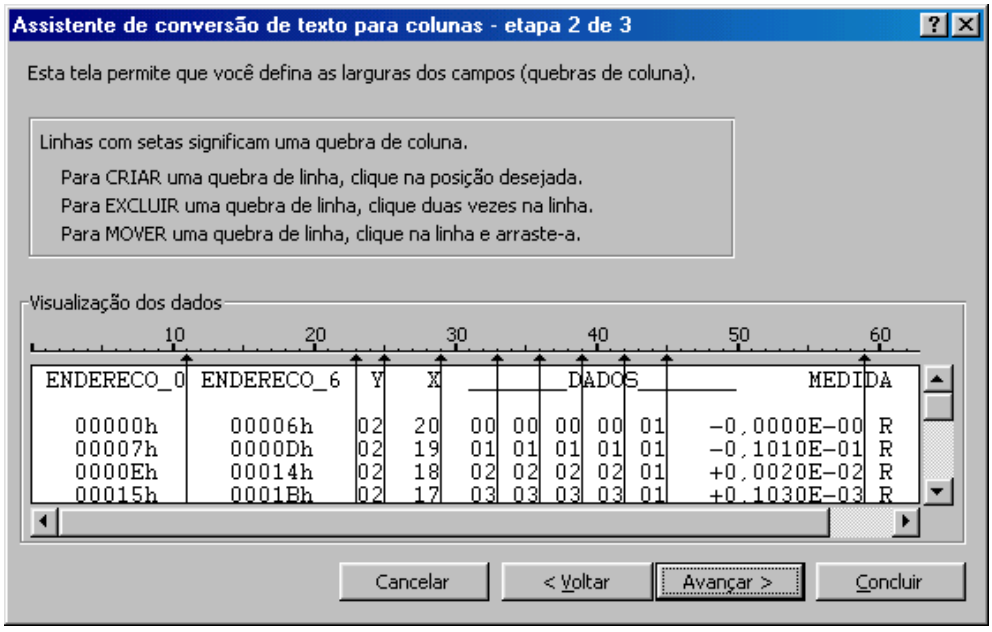
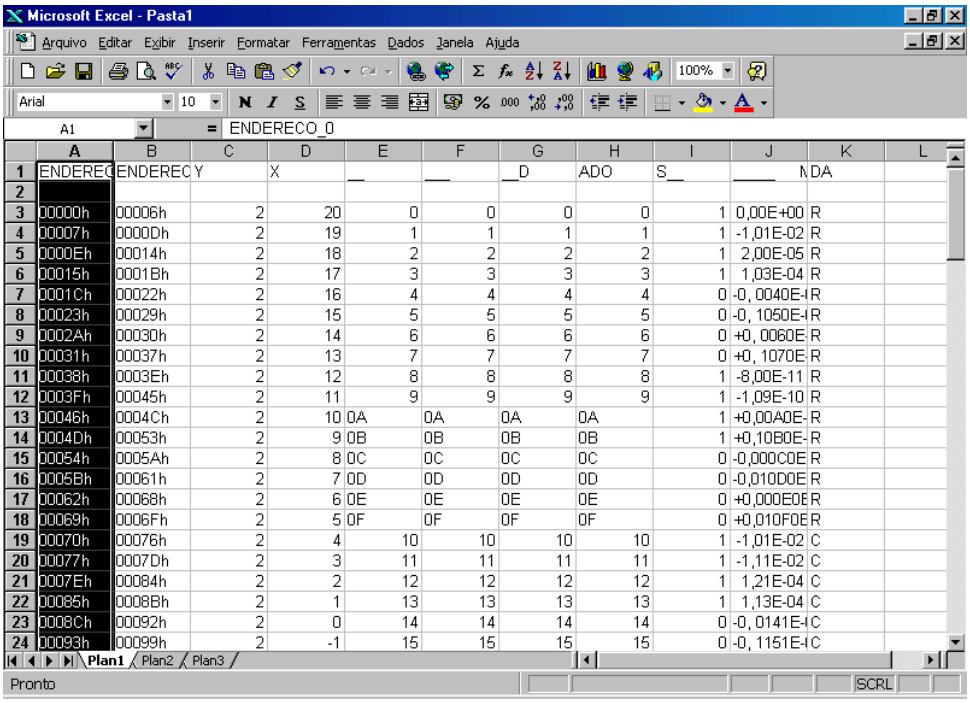


Figura A87 – Definição das larguras dos campos.

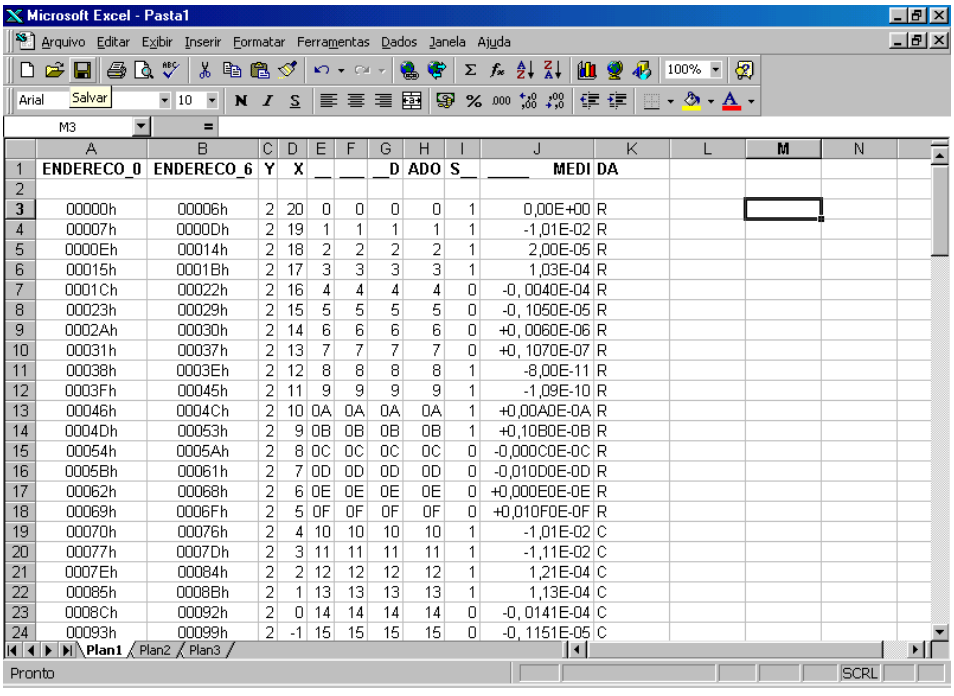
Alguns pequenos ajustes podem ser necessários. Clique novamente em avançar, e após em concluir. E terá:



	A	B	C	D	E	F	G	H	I	J	K	L
1	ENDEREC	ENDEREC	Y	X	__	___	__D	ADO	S__	_____	MDA	
2												
3	00000h	00006h	2	20	0	0	0	0	1	0,00E+00	R	
4	00007h	0000Dh	2	19	1	1	1	1	1	-1,01E-02	R	
5	0000Eh	00014h	2	18	2	2	2	2	1	2,00E-05	R	
6	00015h	0001Bh	2	17	3	3	3	3	1	1,03E-04	R	
7	0001Ch	00022h	2	16	4	4	4	4	0	-0, 0040E-I	R	
8	00023h	00029h	2	15	5	5	5	5	0	-0, 1050E-I	R	
9	0002Ah	00030h	2	14	6	6	6	6	0	+0, 0060E	R	
10	00031h	00037h	2	13	7	7	7	7	0	+0, 1070E	R	
11	00038h	0003Eh	2	12	8	8	8	8	1	-8,00E-11	R	
12	0003Fh	00045h	2	11	9	9	9	9	1	-1,09E-10	R	
13	00046h	0004Ch	2	10	0A	0A	0A	0A	1	+0,00A0E-	R	
14	0004Dh	00053h	2	9	0B	0B	0B	0B	1	+0,10B0E-	R	
15	00054h	0005Ah	2	8	0C	0C	0C	0C	0	-0,000C0E	R	
16	0005Bh	00061h	2	7	0D	0D	0D	0D	0	-0,010D0E	R	
17	00062h	00068h	2	6	0E	0E	0E	0E	0	+0,000E0E	R	
18	00069h	0006Fh	2	5	0F	0F	0F	0F	0	+0,010F0E	R	
19	00070h	00076h	2	4	10	10	10	10	1	-1,01E-02	C	
20	00077h	0007Dh	2	3	11	11	11	11	1	-1,11E-02	C	
21	0007Eh	00084h	2	2	12	12	12	12	1	1,21E-04	C	
22	00085h	0008Bh	2	1	13	13	13	13	1	1,13E-04	C	
23	0008Ch	00092h	2	0	14	14	14	14	0	-0, 0141E-I	C	
24	00093h	00099h	2	-1	15	15	15	15	0	-0, 1151E-I	C	

Figura A88 – Dados quase organizados no Excel.

Após pequenos ajustes ter-se-á:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	ENDERECO_0	ENDERECO_6	Y	X	__	___	__D	ADO	S__	_____ MEDI	DA			
2														
3	00000h	00006h	2	20	0	0	0	0	1	0,00E+00	R			
4	00007h	0000Dh	2	19	1	1	1	1	1	-1,01E-02	R			
5	0000Eh	00014h	2	18	2	2	2	2	1	2,00E-05	R			
6	00015h	0001Bh	2	17	3	3	3	3	1	1,03E-04	R			
7	0001Ch	00022h	2	16	4	4	4	4	0	-0, 0040E-04	R			
8	00023h	00029h	2	15	5	5	5	5	0	-0, 1050E-05	R			
9	0002Ah	00030h	2	14	6	6	6	6	0	+0, 0060E-06	R			
10	00031h	00037h	2	13	7	7	7	7	0	+0, 1070E-07	R			
11	00038h	0003Eh	2	12	8	8	8	8	1	-8,00E-11	R			
12	0003Fh	00045h	2	11	9	9	9	9	1	-1,09E-10	R			
13	00046h	0004Ch	2	10	0A	0A	0A	0A	1	+0,00A0E-0A	R			
14	0004Dh	00053h	2	9	0B	0B	0B	0B	1	+0,10B0E-0B	R			
15	00054h	0005Ah	2	8	0C	0C	0C	0C	0	-0,000C0E-0C	R			
16	0005Bh	00061h	2	7	0D	0D	0D	0D	0	-0,010D0E-0D	R			
17	00062h	00068h	2	6	0E	0E	0E	0E	0	+0,000E0E-0E	R			
18	00069h	0006Fh	2	5	0F	0F	0F	0F	0	+0,010F0E-0F	R			
19	00070h	00076h	2	4	10	10	10	10	1	-1,01E-02	C			
20	00077h	0007Dh	2	3	11	11	11	11	1	-1,11E-02	C			
21	0007Eh	00084h	2	2	12	12	12	12	1	1,21E-04	C			
22	00085h	0008Bh	2	1	13	13	13	13	1	1,13E-04	C			
23	0008Ch	00092h	2	0	14	14	14	14	0	-0, 0141E-04	C			
24	00093h	00099h	2	-1	15	15	15	15	0	-0, 1151E-05	C			

Figura A89 – Dados organizados no Excel – final.