Capítulo 5

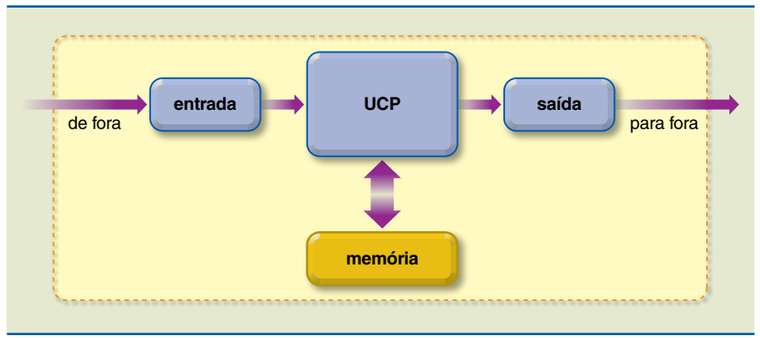
Sistemas microprocessados

Inicialmente, vejamos o conceito de computador, microprocessador e microcontrolador.

Computador

Computador é uma máquina que pode armazenar e processar informações. Hoje consideram-se computadores as máquinas digitais que operam com informações binárias. Basicamente, um computador é constituído de entrada, UCP (unidade central de processamento), memória e saída (figura 5.1).

Figura 5.1
Organização básica de um computador.



Registrador é um conjunto de posições (circuitos *flip-flops*) formando uma unidade que armazena uma informação binária. Os locais de memória contidos na UCP chamam-se registradores. Eles podem ser constituídos de diferentes números de bits, dependendo da quantidade de bits que armazenam.

UCP – Coordena todas as tarefas e executa os cálculos. É composta de três partes: unidade de controle, unidade lógico-aritmética (ULA) e um conjunto de registradores.

a) **Unidade de controle** – É de onde partem os sinais de controle de todo o sistema, estabelecendo a sincronização correta das tarefas que estão sendo realizadas.

b) **ULA** – É onde se realizam as operações lógicas e aritméticas determinadas pela unidade de controle. São operações aritméticas: subtrair, incrementar, setar bit etc. São operações lógicas: lógica "E", lógica "OU", comparação etc.

c) **Conjunto de registradores** – É constituído de registradores com várias finalidades, entre elas: contador de programa, armazenamento de dados processados pela UCP, armazenamento de endereços etc.

Entrada e saída – São compostas de todos os dispositivos que interligam as informações externas ao computador. É por meio desses dispositivos que podemos inserir informações no computador (entrada) ou receber informações dele (saída). São dispositivos de entrada: teclados, sensores, chaves etc. São dispositivos de saída: impressoras, motores, painéis etc. Os dispositivos de entrada e saída são ligados à UCP por interfaces apropriadas a cada dispositivo.

Memória – O conjunto de memórias é basicamente constituído de memórias RAM e ROM. As memórias RAM, por serem de leitura e escrita, armazenam dados que podem variar no decorrer do programa. As memórias ROM, apenas de leitura, armazenam dados fixos ou programas, ou seja, dados que não podem mudar durante toda a execução do programa.

5.1 Processadores

Os dois tipos de arquitetura mais comuns utilizados em microprocessadores e microcontroladores são Von-Neumann e Harvard.

A arquitetura Von-Neumann tem um único barramento por onde circulam os dados e instruções, tornando necessária maior quantidade de ciclos de máquina para executar uma instrução – uma simples soma de dois números, por exemplo, gasta três ciclos de máquina.

Essa arquitetura é utilizada na família de microcontroladores 8051. Nela, as instruções são estruturadas com base na tecnologia CISC, a qual envolve maior complexidade na execução da instrução. Comparada com a tecnologia RISC, usada na arquitetura Harvard, gasta mais ciclos de máquina.

Sigla em inglês de *complex instruction set computer*, computador com conjunto de instruções complexas.

Em geral, para uma mesma capacidade de processamento, o microprocessador com tecnologia CISC tem um *set* de instruções bem maior; assim, é possível executar programas menores com CISC. O mesmo programa em tecnologia RISC ficará bem maior, pois as instruções disponíveis em CISC não estão em RISC, sendo necessário criar as instruções faltantes com as RISC existentes.

Sigla em inglês de *reduced instruction set computer*, computador com conjunto de instruções reduzidas.

A arquitetura Harvard tem dois barramentos distintos, um para dados e outro para instruções, possibilitando maior rapidez no processamento – para fazer uma operação de soma de dois números, por exemplo, é necessário apenas um ciclo de máquina. Uma grande vantagem dessa arquitetura é o fato de que, enquanto uma instrução é processada, outra já pode estar executando seu ciclo de busca, carregando a próxima instrução. Tal processo de busca/execução é conhecido como *pipeline*. Essa é a arquitetura da família PIC, cujas instruções são estruturadas com base na tecnologia RISC, conforme já comentado.

Sigla em inglês de *programmable interface controller*, controlador de interface programável.

Microprocessador

O microprocessador MPU é um *chip* que substitui a UCP de um computador. As características técnicas do MPU basicamente definem as características do computador.

Sigla em inglês de *microprocessor unit*.

Antes dos microprocessadores, a UCP era construída com transistores e CI digitais discretos, o que tornava os computadores maiores e mais caros. Os microprocessadores de oito bits começaram a ser desenvolvidos em 1972. O pioneiro foi o 8008, que era capaz de endereçar 16 kB de memória e possuía 45 instruções e velocidade de 300 000 operações por segundo.

Com o passar dos anos, surgiram o 8080 (Intel), o 6800 (Motorola) e, em 1976, o "famoso" Z80 (Zilog), considerado na época um microprocessador com grande capacidade de processamento. O Z80 podia endereçar 64 kB de memória, possuía 176 instruções, além de executar todos os programas escritos para o 8080. Com essas vantagens, ganhou a preferência do mercado.

Microcontrolador

O microcontrolador (também designado por MCU) é um *chip* que contém, além da UCP, memórias RAM e ROM, oscilador interno de *clock*, I/O e outros recursos, o que o torna um verdadeiro computador em uma única pastilha. Atualmente, existe grande variedade de MCUs, como os das famílias 8051, PIC, COP, AVR etc.

O poder de processamento dos microprocessadores é maior que o dos microcontroladores. Por isso, os microprocessadores são usados em sistemas que necessitam de UCP mais sofisticada e com funções mais complexas (figuras 5.2 e 5.3).

Figura 5.2
Bloco microprocessador.

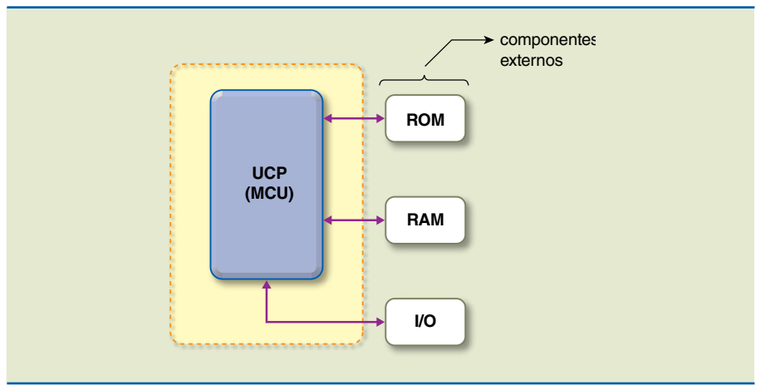
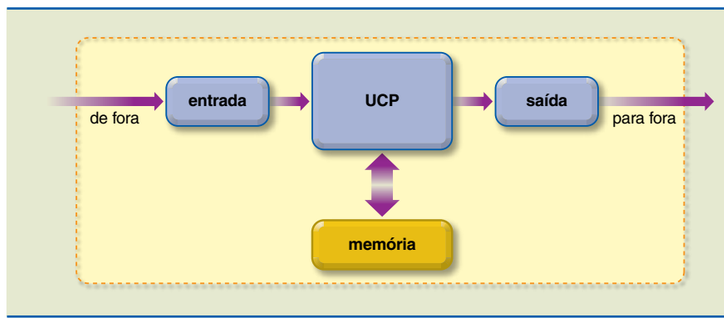


Figura 5.3
Bloco microcontrolador mínimo simplificado.



A figura 5.4 representa a estrutura interna de um microcontrolador constituído dos seguintes blocos internos:

- UCP
- Memória
- Entrada e saída
- Comunicação serial
- Temporizador
- *Watchdog*

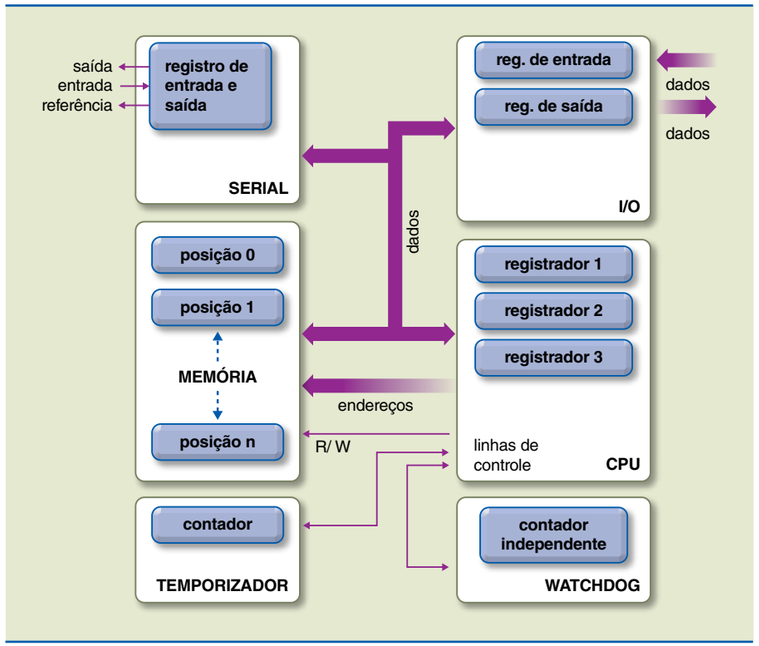


Figura 5.4
Microcontrolador: estrutura básica.

Comunicação serial – Nesse tipo de comunicação, são necessárias somente três linhas para enviar e/ou receber os dados: uma apenas para enviar dados, uma só para receber dados e uma para estabelecer o protocolo. Protocolo é a regra estabelecida para que receptor e emissor se entendam; assim, na terceira linha é colocado um sinal informando "quem" deve estar preparado para receber e "quem" deve estar preparado para enviar.

Temporizador – Permite usar intervalos de tempo para controlar eventos, informações de tempo gasto etc. A base dessa unidade é o contador, que é um registrador que aumenta em uma unidade seu valor em determinado tempo.

***Watchdog* (WDT)** – É um contador incrementado automaticamente com um *clock* independente. O *clock* do WDT provém de um temporizador RC só para ele; portanto, seu incremento independe do *clock* da máquina. Essa característica do WDT possibilita usá-lo para evitar o travamento do programa, da seguinte maneira: o tempo normal de "estouro" do WDT é de 18 ms; quando ocorre o estouro, é gerado um *reset* forçado e esse *reset* é usado para evitar o travamento. O programador, ao longo do programa, zera o WDT, evitando seu estouro e mantendo, assim, o desenvolvimento normal do programa. Nessa condição, caso ocorra

O estouro (*overflow*, em inglês) corresponde à situação em que o contador WDT atingiu o último estado de sua sequência de contagem. Ao fazer uso de um WDT, é necessário escrever o programa de maneira a redefinir o WDT com frequência suficiente (menor que 18 ms) para impedi-lo de atingir o último estado (estouro). Se isso acontecer é porque, por algum motivo, o programa travou.

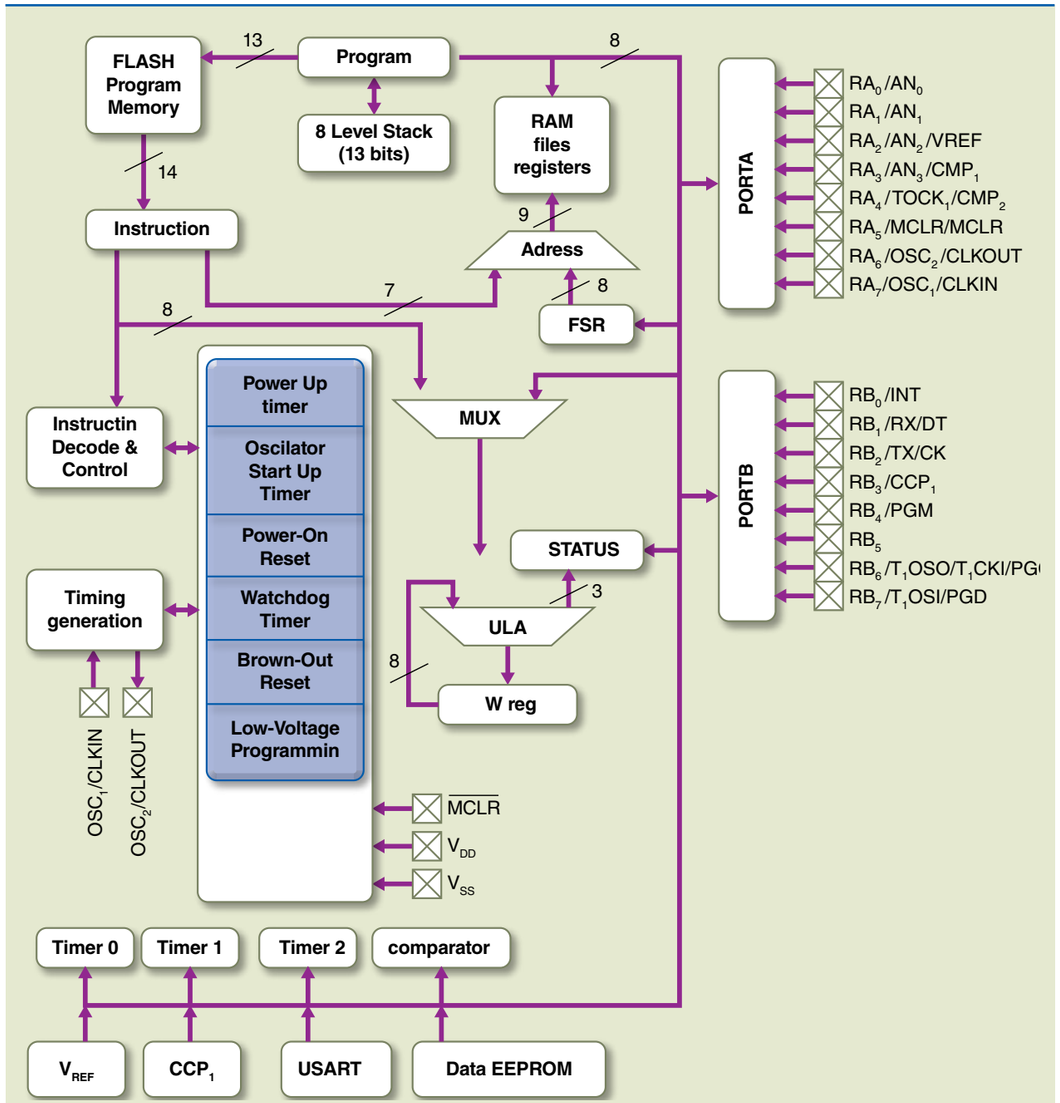
travamento, o WDT não é zerado e o WDT estoura, gerando um *reset* e saindo do travamento. O WDT pode ser desligado como opção de configuração.

Os demais blocos da figura 5.4 (UCP, memória, entrada e saída) foram explicados anteriormente. É importante ressaltar a ação das linhas de controle gerenciadas pela UCP existente entre ela e as unidades de memória, temporização e WDT.

5.1.1 Estrutura interna do PIC16F628A

A figura 5.5 apresenta a estrutura interna do microcontrolador PIC16F628A.

Figura 5.5
PIC16F628A: estrutura interna.



A estrutura interna apresentada na figura 5.5 encontra-se no *datasheet* da Microchip, que é sua fabricante. Observe que o registrador W (*work*) aparece em destaque e está diretamente ligado à ULA. A mais importante função desse registrador é sua utilização na troca de dados entre registradores, pois não é possível trocar diretamente dados entre registradores SFR e GPR, como veremos mais adiante.

A memória de dados é conectada a um barramento de oito bits, enquanto a memória de programa, devido à arquitetura Harvard, pode ser conectada a um barramento de 14 bits, possibilitando o processo de execução *pipeline*.

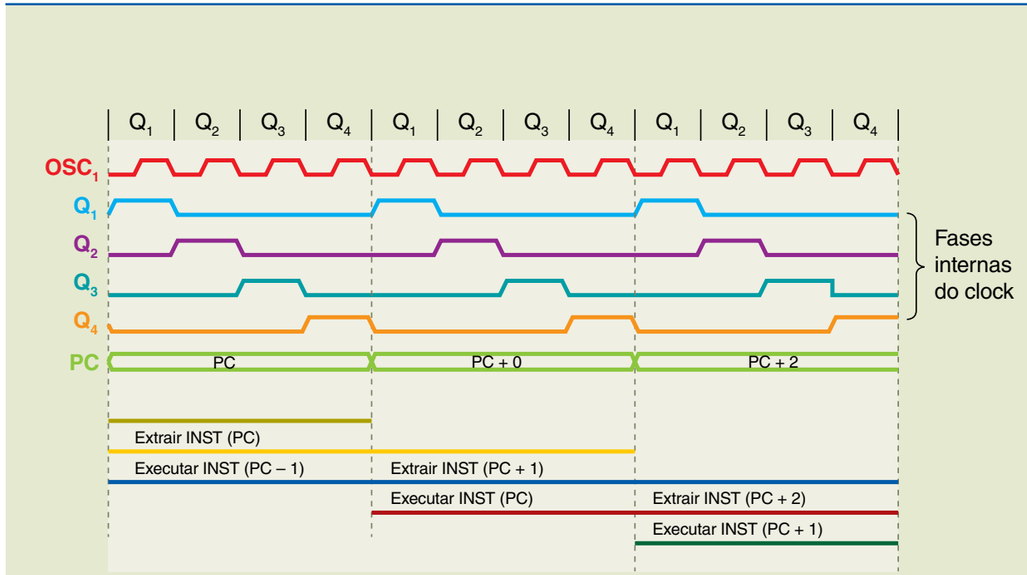
Na figura 5.5, notamos também blocos que representam circuitos (memórias, *timers*, comparadores, porta serial) que não são encontrados em microprocessadores, porém por ser um microcontrolador o modelo 16F628A inclui esses circuitos.

O PIC16F628A pode ter seu *clock* gerado por oscilador externo de no máximo 20 MHz ou por oscilador interno programado para 4 MHz ou 37 kHz. Em todos os casos, o *clock* gerado é dividido internamente por 4.

Vamos considerar um *clock* de 4 MHz. Após a divisão interna por 4, temos um *clock* interno de 1 MHz e, portanto, um período de 1 µs, que é o tempo denominado ciclo de máquina, no qual cada etapa da instrução é executada.

O diagrama da figura 5.6 consta do manual da Microchip.

Figura 5.6
Diagrama das fases internas do *clock* para o PIC16F628A.



Analisando a figura 5.6, podemos notar que durante um ciclo (composto de quatro fases: Q_1, Q_2, Q_3, Q_4), enquanto uma instrução é executada, a próxima é buscada para ser executada no ciclo seguinte; assim, cada instrução é executada em um único ciclo. Essa execução em *pipeline* é facilmente realizada pelo microcontrolador devido à arquitetura Harvard.

Em geral, as instruções são executadas em apenas um ciclo de máquina, exceto as que geram salto, como as chamadas de rotina e os saltos para outros endereços que não os da sequência normal do programa. Dessa maneira, para um *clock* externo de 4 MHz, temos uma instrução simples que leva 1 µs para ser executada e aquelas que produzem salto levam 2 µs.

5.2 Programação

Para que o processador execute tarefas, é necessário fazer a programação, especificando os passos que devem ser seguidos. Essa sequência é inicialmente descrita em um fluxograma e, depois, transformada em comandos de uma linguagem de programação.

5.2.1 Fluxograma

O fluxograma descreve a lógica do programa de acordo com a sequência em que as tarefas ocorrem; por isso, é uma ferramenta valiosa na execução do programa, qualquer que seja a linguagem utilizada. A figura 5.7 apresenta os blocos que usaremos para fazer o fluxograma de um evento.

Figura 5.7
Blocos do fluxograma.

	terminal	início ou fim do programa
	processo	o que deve ser executado (operações, cálculos etc.)
	sub-rotina	processo pré-definido
	decisão	testa determinada condição, dependendo do resultado segue um dos caminhos.
	conector	indica que o fluxograma continua em outro ponto.
	setas	indica o sentido do fluxo em que o evento está ocorrendo.

A figura 5.8 exemplifica a elaboração de um fluxograma para um evento simples: acender uma lâmpada sempre que um botão estiver pressionado.

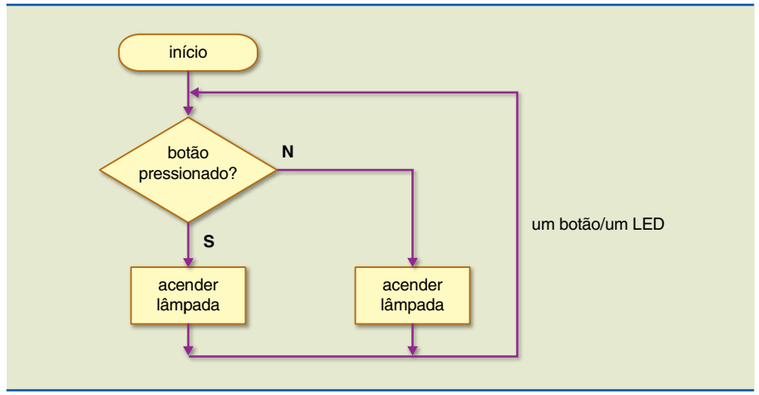


Figura 5.8
Fluxograma simples para acender uma lâmpada sempre que o botão estiver pressionado.

Exemplo

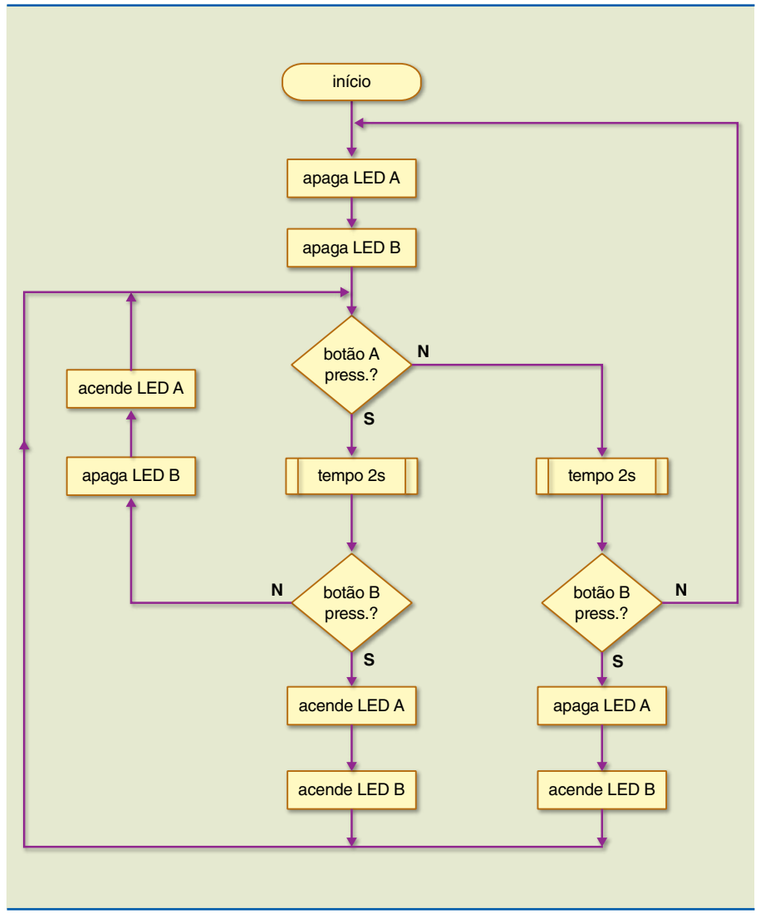
Faça o fluxograma do seguinte evento:

a) Inicialmente dois LEDs estão apagados.
b) Ao pressionar o botão A, mantendo o botão B solto, o LED A acende e o LED B permanece apagado.
c) Ao pressionar o botão B, mantendo o botão A solto, o LED B acende e o LED A permanece apagado.
d) Os dois botões pressionados acendem os dois LEDs.
e) Os dois botões não pressionados mantêm os dois LEDs apagados.
f) O segundo botão deve somente ser considerado pressionado se for pressionado no máximo 2 segundos após o primeiro.

Solução:

A figura 5.9 mostra o fluxograma solicitado.

Figura 5.9
Fluxograma dois botões/dois LEDs.



O bloco "tempo 2 s" é representado como sub-rotina, pois é um programa predefinido, ou seja, quando o programa principal necessita de um intervalo (*delay*) de 2 segundos, chama essa sub-rotina. A sub-rotina é executada e retorna à posição imediatamente seguinte no programa principal; assim, o programa da sub-rotina é escrito uma única vez.

5.2.2 Linguagens de programação

Linguagem de máquina ou **código objeto** é um conjunto de informações na forma binária dispostas em uma condição previamente definida de maneira que o microcontrolador possa processá-las.

Elaborar um programa dessa maneira é trabalhoso e cansativo, pois as instruções, os dados e todas as demais informações necessárias para a máquina devem estar em binário. A linguagem de máquina é própria de cada microcontrolador, sendo definida pelo fabricante; portanto, em geral, o código objeto de uma máquina não é compatível com o de outra.

A figura 5.10 mostra a aparência de um programa em linguagem de máquina.

endereços	dados
0000 0011 0011 0000	0000 1001
0000 0011 1101 0010	1001 0010
0000 0011 0110 0100	0100 1001
0000 0011 0001 1110	1000 0010

Figura 5.10
Aparência dos endereços e dados na linguagem de máquina.

Como podemos observar, um programa escrito desse modo pode se tornar confuso, de difícil compreensão, pois não indica o que o microcontrolador está executando, e a correção de erros é trabalhosa. Enfim, é uma linguagem somente para a máquina. Esse tipo de linguagem é chamado de linguagem de baixo nível.

Para facilitar o trabalho de programação e eliminar uma série de inconvenientes para o programador, foi desenvolvida a **linguagem *assembly***. Embora seja considerada linguagem de baixo nível, facilitou muito esse trabalho.

A linguagem *assembly* substitui códigos binários por mnemônicos, isto é, nomeia os códigos, permitindo fazer fácil associação com a função do código. O microcontrolador não entende diretamente um programa escrito em linguagem *assembly*; é necessário convertê-la em linguagem de máquina. O compilador *assembler* ou programa *assembler* faz essa transformação (figura 5.11).

Com o passar dos anos, surgiram linguagens classificadas como de alto nível. Elas apresentavam um conceito mais geral, podendo ser usadas para programar praticamente qualquer microcontrolador, independentemente de sua linguagem de baixo nível específica.

Para converter um programa fonte escrito em linguagem de médio ou alto nível em linguagem de máquina, utiliza-se um programa compilador (figura 5.12).

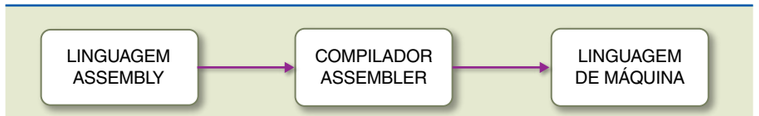


Figura 5.11
Compilador ***assembler***.

Figura 5.12
Programa compilador.

Existem diversos tipos de compiladores, dependendo da linguagem: compilador C, compilador Pascal etc.

O programa compilador transforma primeiro todo o programa em linguagem de máquina e depois o executa.

Existe também o programa interpretador, no qual cada instrução é transformada em linguagem de máquina e em seguida executada antes de a próxima instrução ser transformada. Portanto, a execução de um programa compilado é mais rápida que a de um programa interpretado.

Podemos testar e corrigir um programa por meio de programas que simulam sua execução; esse procedimento é chamado de depuração.

5.2.3 Linguagem *assembly*

A linguagem que utilizamos para nos comunicar com o microcontrolador é a linguagem *assembly*. O *assembly* é composto de um conjunto de instruções específico para cada processador. Ocorre, porém, que o sistema digital somente entende uma linguagem composta por "0" e "1", sendo necessário, portanto, que o código *assembly* seja traduzido para o código binário.

A linguagem *assembly* corresponde a um conjunto de regras e instruções necessárias para escrever um programa que será utilizado em determinada CPU, microprocessador ou microcontrolador.

Assembler é um programa que, executado em um sistema digital microprocessado ou microcontrolado, traduz o código escrito em linguagem *assembly* para um código equivalente de "0" e "1", ou seja, em linguagem de máquina.

O programa fonte é uma sequência de instruções escritas conforme as regras do *assembly* do processador ou qualquer outra linguagem de programação de microcontroladores (por exemplo: linguagem C para PICmicros), que normalmente é gravado em disco para ser carregado na RAM. Vamos considerar, por exemplo, a instrução *return* que um microcontrolador utiliza para retornar de uma sub-rotina. Quando o programa *assembler* faz a tradução, obtemos uma série de "0" e "1" correspondente a 14 bits que o microcontrolador PIC pode reconhecer, nesse caso: 00 0000 0000 1000.

Quando o programa *assembler* realiza a compilação do código fonte, também é gerado um arquivo com extensão "HEX". Esse nome provém de uma representação hexadecimal do programa em linguagem de máquina. Uma vez concluído o processo de montagem e compilação, o arquivo em código de máquina gerado é inserido no microcontrolador por um programador.

Um programa em linguagem *assembly* pode ser escrito originalmente em qualquer editor de textos e depois copiado na tela do editor do ambiente de programação, ou, então, editado diretamente no próprio ambiente de programação, como o MPLAB (ver Apêndice C).

O conjunto de instruções do microcontrolador PIC

Vejamos alguns termos utilizados no conjunto de instruções do microcontrolador PIC:

Work – Registrador acumulador temporário, representado pela letra **W**.

File – Referência a um registrador (posição de memória) representado pela letra **F**.

Literal – Um número qualquer em binário, decimal ou hexadecimal, representado por **L** nas instruções e por **K** nos argumentos.

Destino – Onde o resultado de uma operação será armazenado. Só existem duas possibilidades para o destino: em **F** – no registrador passado como argumento – ou em **W** – no registrador *work*. Também podemos representar "**0**" para **W** e "**1**" para **F** no destino.

Exemplo de como fazer a construção do nome das instruções

Somar uma unidade ao valor armazenado em um registrador chamado REGX.

Solução:

Incrementar (**INC**) registrador (**F**) REGX = **INCF REGX**

Tabela 5.1
Conjunto de instruções retirado do original em inglês.

Mnemonic, Operands		Description	Cycles	14-Bit Instruction Word				Status Affected	Notes
				Msb			Lsb		
Byte-Oriented File Register Operations									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	lfff	ffff	Z	2
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1,2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff		1,2,3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	lfff	ffff		

Mnemonic, Operands		Description	Cycles	14-Bit Instruction Word				Status Affected	Notes
				Msb			Lsb		
NOP	-	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nobbles in f	1	00	1110	dfff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2
Bit-Oriented File Register Operations									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1,2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1(2)	01	10bb	bfff	ffff		3
BTFSS	f, b	Bit test f, Skip if Set	1(2)	01	11bb	bfff	ffff		3
Literal and Control Operations									
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call subroutine	2	10	0xxx	kkkk	kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0100	0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	-	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	-	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	-	Go into standby mode	1	00	0000	0110	0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

Note 1: When an I/O register is modified as a function of itself (e.g., MOVF, PORTB, 1) the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMRO register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0Module.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

O conjunto de instruções dos microcontroladores PIC da Microchip que utilizaremos em nosso curso consiste em um pequeno repertório de apenas 35 instruções de 12 bits, que podem ser agrupadas em cinco grupos ou categorias. Essas cinco categorias são definidas de acordo com a função e o tipo de operandos envolvidos:

- Instruções que operam com bytes e envolvem algum registrador da memória interna.
- Instruções que operam apenas sobre o registrador W e que permitem carregar uma constante implícita ou incluída literalmente na instrução (literal).
- Instruções que operam sobre bits individuais dos registradores da memória interna.
- Instruções de controle de fluxo do programa, ou seja, aquelas que permitem alterar a sequência de execução das instruções.
- Instruções especiais cujas funções ou tipos de operandos são muito específicos e não se encaixam nas descrições anteriores.

Instruções que operam com registradores

Essas instruções podem ser de operando de origem simples ou duplo. O primeiro operando de origem será o registrador selecionado na instrução e o segundo, se existir, o registrador W. O destino, onde ficará armazenado o resultado, será o registrador selecionado ou W.

As instruções a seguir são operações lógicas de duplo operando:

- **ANDWF** **F,d:** operação lógica AND entre F e W, destino = W ou f
- **IORWF** **F,d:** operação lógica OR entre F e W, destino = W ou f
- **XORWF** **F,d:** operação lógica XOR entre W e F, destino = W ou f

Essas instruções correspondem a operações de simples operando:

- **MOVF** **F,d:** movimento de dados, destino = F ou W
- **COMF** **F,d:** complemento lógico, destino = F ou W
- **INCF** **F,d:** incremento aritmético, destino = F ou W
- **DECF** **F,d:** decremento aritmético, destino = F ou W

Nas sete instruções anteriores, o único bit afetado do registrador de *status* é o Z (bit de *zero*), que assumirá nível lógico "1" se o resultado da operação for "00000000" e nível lógico "0" se o resultado for qualquer outro valor.

As instruções de rotação de bits através do bit C (*carry*) do registrador de *status* são:

- **RLF** **F,d:** rotação de bits à esquerda, destino = F ou W
- **RRF** **F,d:** rotação de bits à direita, destino = F ou W

Nas operações Rotate Left File e Rotate Right File, os bits são deslocados de cada posição à seguinte, para a esquerda ou para a direita. O deslocamento é fechado, formando um anel com o bit C do registrador de *status*.

Nessas duas instruções, o único bit afetado do registrador de *status* é o bit C, que assumirá o valor que estava no bit "7" ou no bit "0", de acordo com o sentido de deslocamento.

A instrução seguinte realiza a troca de posição entre os quatro bits menos significativos e os quatro mais significativos (*nibble* baixo e *nibble* alto):

- **SWAPF F, d:** troca *nibbles*, destino = F ou W

A instrução SWAP File não afeta nenhum bit do registrador de *status*.

As duas operações que se seguem são a soma e a subtração aritméticas:

- **ADDWF F, d:** soma aritmética, destino = F ou D
- **SUBWF F, d:** subtração aritmética, destino = F ou D

As operações ADD WF e SUBtract W de F afetam três bits do registrador de *status*: C, DC e Z.

O bit Z assumirá nível lógico "1" se o resultado da operação for "00000000" e nível lógico "0" se o resultado for qualquer outro valor.

Os bits do registrador de *status* C e DC assumem o valor normal correspondente à soma de F com o complemento 2 de W. Dessa maneira, o significado para a operação de subtração resulta invertido, ou seja, C (*carry*) será "1" se não houver estouro na subtração (se o conteúdo de W for menor que o de F). O bit DC se comporta de modo similar, isto é, DC será "1" se não houver estouro na metade menos significativa, o que equivale a dizer que o *nibble* baixo do conteúdo de W é menor que o *nibble* baixo do conteúdo do registrador F.

As instruções a seguir são de simples operando, mas trata-se de casos especiais, pois o destino será sempre o próprio registro selecionado:

- **CLRF F:** zera todos os bits de F
- **MOVWF F:** copia conteúdo de W em F, destino = F

A instrução CLRF (CLeaR File) afeta somente o bit Z, que resulta sempre "0". A instrução MOVWF (MOVe W para F) não afeta nenhum bit do registrador de *status*.

Registrador de *status*

Esse registrador armazena o estado atual da unidade lógico-aritmética, do *reset* e do banco de memória utilizado.

Bit de escrita/ leitura	Bit de escrita/ leitura	Bit de escrita/ leitura	Bit de leitura	Bit de leitura	Bit de escrita/ leitura	Bit de escrita/ leitura	Bit de escrita/ leitura
IRP	RP1	RP0	TO	PD	Z	DC	C
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Bit 7 – IRP – Esse registro não é utilizado pelo PIC16F84 ou PIC16F628A, lido sempre como "0".

Bits 6, 5 – RP1, RP0 – Registros de seleção do banco de registros:

- 00 = banco 0 (00H – 7FH)
- 01 = banco 1 (80H – FFH)

Bit 4 – TO (*Time-out)*

- 1 = Após *power-up* (ligar), instrução CLRWDT (limpar cão de guarda) ou instrução SLEEP (dormir).
- 0 = Um *time-out* do WDT ocorreu.

Bit 3 – PD (*Power-down)*

- 1 = Após *power-up* (ligar) ou instrução CLRWDT (limpar cão de guarda).
- 0 = Execução da instrução SLEEP (dormir).

Bit *2* – Z (*Zero)*

- 1 = O resultado da operação aritmética ou lógica é igual a zero.
- 0 = O resultado da operação aritmética ou lógica não é zero.

Bit 1 – DC (Digito *Carry/borrow)*

- 1 = Ocorreu transbordamento do quarto bit menos significativo.
- 0 = Não ocorreu transbordamento do quarto bit menos significativo.

Bit 0 – C (*Carry)*

- 0 = Não ocorreu transbordamento do sétimo bit mais significativo.

Instruções que operam com registrador W e literais

Essas instruções se referem ao registrador W, isto é, um dos operandos de origem e o operando de destino serão sempre o registrador W. Nas instruções desse grupo que têm um segundo operando de origem, este será sempre uma constante incluída na instrução, chamada literal.

As instruções a seguir são as operações lógicas tradicionais, similares às que vimos anteriormente, porém realizadas entre a constante de programa e o registrador W:

- **IORLW** **K:** operação lógica OR entre W e L, destino = W
- **ANDLW** **K:** operação lógica AND entre W e L, destino = W
- **XORLW** **K:** operação lógica XOR entre W e L, destino = W

Nessas três instruções (Inclusive OR Literal W, AND Literal W e XOR Literal W), o único bit afetado do registrador de *status* é Z, que assumirá nível lógico "1" se o resultado da operação for "00000000" e nível lógico "0" se o resultado for qualquer outro valor.

A instrução que se segue é utilizada para armazenar uma constante no registrador W:

- **MOVLW** **K:** armazena constante em W, destino = W

A instrução MOVe Literal W não afeta nenhum bit do registrador de *status*.

A instrução seguinte (CLeaR W) é um caso especial da instrução CLRF, com destino = W:

- **CLRW:** zera todos os bits de W

Como na instrução CLRF, o único bit do registrador de *status* afetado é Z, que assumirá nível lógico "1".

As instruções são operações aritméticas entre W e uma constante.

- **ADDLW** **K:** soma W e K, destino = W
- **SUBLW** **K:** subtrai W de K

As operações ADD LW e SUBtract W de K afetam três bits do registrador de *status*: C, DC e Z.

Instruções que operam com bits

Essas instruções operam somente sobre o bit especificado; todos os outros bits do registrador não são alterados. Elas não têm especificação de destino, já que este será sempre o próprio registrador selecionado.

- **BCF** **F, b:** zera o bit b de F
- **BSF** **F, b:** "seta" o bit b de F

As instruções Bit Clear File e Bit Set File não afetam nenhum bit do registrador de status.

Instruções de controle

- **GOTO** **K:** salta para a posição K do programa

Essa é uma instrução típica de salto incondicional para qualquer posição de memória de programa.

A constante literal K corresponde ao endereço de destino do salto, isto é, o novo endereço de memória de programa a partir do qual serão executadas as instruções após a execução de uma instrução GOTO. Essa instrução simplesmente armazena a constante K no registrador PC (contador de programa).

A instrução seguinte é chamada de sub-rotina:

- **CALL** **K:** salta para a sub-rotina na posição K

O comportamento dessa instrução é muito similar ao da instrução GOTO, salvo que, além de saltar, armazena no *stack* o endereço de retorno da sub-rotina (que será usado pela instrução RETURN e RETLW). Isso é realizado armazenando no *stack* uma cópia do PC incrementado, antes que ele seja carregado com o novo endereço K.

As instruções a seguir são de retorno de sub-rotina:

- **RETURN:** retorno de sub-rotina

Retorna para o endereço de programa imediatamente posterior ao de chamada da referida sub-rotina.

- **RETLW** **K:** retorno de sub-rotina com constante K armazenada em W

Essa instrução (RETurn with Literal in W) permite o retorno de sub-rotina com uma constante literal K armazenada no registrador W. A operação que essa instrução realiza consiste simplesmente em retirar do *stack* um valor e carregá-lo no PC. Esse valor é o PC antes de realizar o salto incrementado, proveniente da última instrução CALL executada; portanto, será o endereço da instrução seguinte ao CALL.

- **RETFIE:** retorno de interrupção

Retorna de uma interrupção, recuperando o último endereço colocado no *stack*.

A seguir encontram-se as únicas instruções de salto (*skip*) condicional, os únicos meios para implementar desvios condicionais em um programa. Elas são genéricas e muito poderosas, pois permitem ao programa tomar decisões em função

de qualquer bit de qualquer posição de memória interna de dados, incluindo os registradores de periféricos, as portas de entrada/saída e o próprio registrador de *status* do processador. Essas duas instruções substituem e superam todas as instruções de saltos condicionais dos processadores convencionais (salto por zero, por não zero, por *carry* etc.).

- **BTFSC F, b:** salta se bit = 0
- **BTFSS F, b:** salta se bit = 1

A instrução BTFSC (Bit Test File and Skip if Clear) salta a próxima instrução se o bit *b* do registrador F for "0", e a instrução BTFSS (Bit Test File and Skip if Set), se o bit *b* do registrador F for "1".

Essas instruções podem ser usadas para realizar ou não uma ação conforme o estado de um bit, ou, em combinação com a instrução GOTO, para realizar um desvio condicional.

As instruções a seguir são casos especiais de incremento e decremento vistos anteriormente, mas estas afetam o fluxo do programa:

- **DECFSZ F, d:** decrementa F e salta se 0, destino = F ou W
- **INCFSZ F, d:** incrementa F e salta se 0, destino = F ou W

Essas duas instruções (DECrement File and Skip if Zero e INCrement File and Skip if Zero) se comportam de maneira similar a DECF e INCF, salvo que não afetam nenhum bit do registrador de *status*. Uma vez realizado o incremento ou decremento, se o resultado for "00000000", o processador saltará a próxima instrução do programa.

Essas instruções são utilizadas geralmente em combinação com uma instrução de salto (GOTO), para o projeto de ciclos ou laços (*loops*) de instruções que devem ser repetidas determinado número de vezes.

Instruções especiais

Nesse grupo estão as instruções que controlam funções específicas do microcontrolador ou que atuam sobre registradores especiais não endereçados como memória interna normal.

A instrução a seguir é a típica NO OPERATION, existente em quase todos os processadores:

- **NOP:** não faz nada, consome tempo

É utilizada apenas para introduzir um atraso (*delay*) no programa, equivalente ao tempo de execução de uma instrução. Não afeta nenhum bit do registrador de *status*.

A instrução seguinte "resseta" o contador do watch dog timer. Esse registrador não está acessível como memória e essa é a única instrução que o modifica.

- **CLRWDT:** "resseta" o *watch dog timer*

Essa instrução também afeta os bits PD (*power-down*) e TO (*time-out*) do registrador de *status*.

A instrução seguinte é um controle especial do microcontrolador que o coloca no modo *power down*, no qual: o microcontrolador suspende a execução do programa; o oscilador fica em estado constante; os registradores e portos conservam seu estado; e o consumo se reduz ao mínimo. A única maneira de sair desse estado é por meio de um *reset* ou por *time-out* do *watch dog timer*.

- **SLEEP:** coloca o MC em modo *sleep*

Essa instrução zera o bit PD (*power-down*) e "seta" o bit TO (*time-out*) do registrador de status.