MANUAL DO ALUNO

DISCIPLINA
SISTEMAS
DIGITAIS

Módulos 6, 7 e 8

República Democrática de Timor-Leste
Ministério da Educação

FICHA TÉCNICA

TÍTULO
MANUAL DO ALUNO - DISCIPLINA DE SISTEMAS DIGITAIS
Módulos 6 a 8

AUTOR
JORGE FLÁVIO

COLABORAÇÃO DAS EQUIPAS TÉCNICAS TIMORENSES DA DISCIPLINA
XXXXXXX

COLABORAÇÃO TÉCNICA NA REVISÃO
XXXXXXXXX

DESIGN E PAGINAÇÃO
UNDESIGN - JOAO PAULO VILHENA
EVOLUA.PT

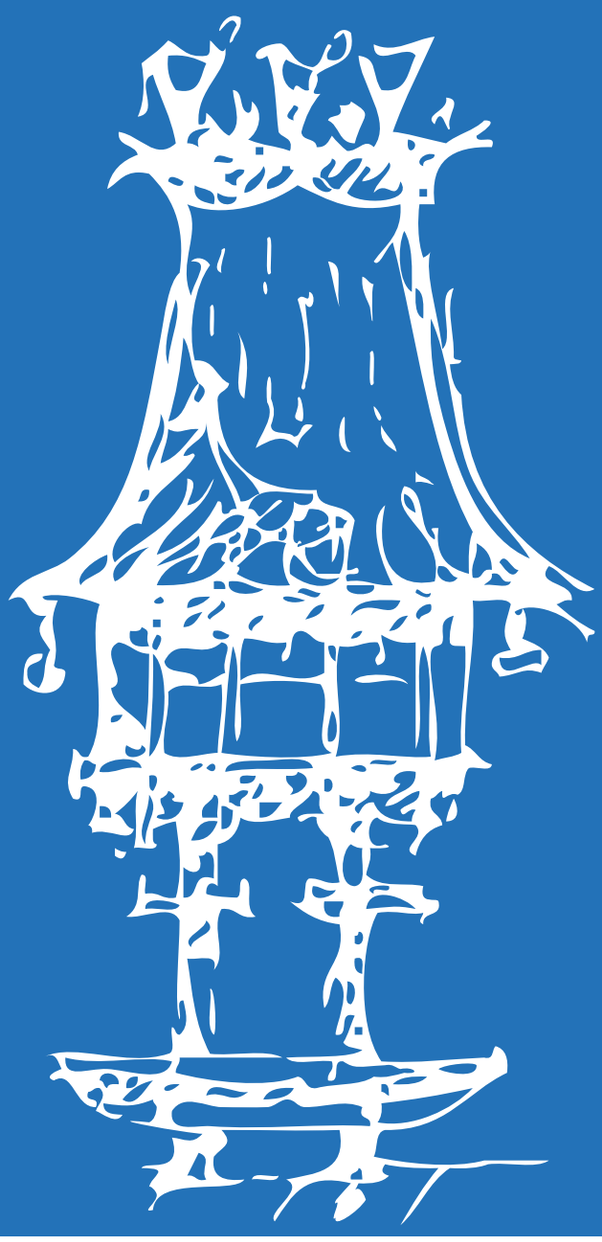
IMPRESSÃO E ACABAMENTO
XXXXXX

ISBN
XXX - XXX - X - XXXXX - X

TIRAGEM
XXXXXXX EXEMPLARES

COORDENAÇÃO GERAL DO PROJETO
MINISTÉRIO DA EDUCAÇÃO DE TIMOR-LESTE
2014

Índice



Introdução à Programação

Módulo 6

Apresentação

Este módulo tem carácter teórico-prático, por isso deverá decorrer em parte em ambiente laboratorial de forma a permitir aos alunos verificarem e comprovarem os conhecimentos teóricos adquiridos sobre uma linguagem de programação de baixo nível. Esta disciplina tem como intenção tornar o aluno apto a compreender a linguagem e as técnicas utilizadas, possibilitando assim um melhor aproveitamento na sequência dos estudos desta e das outras disciplinas técnicas e também na comunicação adequada com os profissionais da área.

Introdução

A abordagem deste módulo sobre introdução à programação leva-nos a uma melhor compreensão do funcionamento de vários tipos de aparelhos, que incorporam circuitos que utilizam estas características, existentes no mercado assim como a melhor escolha deste tipo de equipamentos para que se ajuste às crescentes evoluções disponíveis pelas diversas marcas.

Este módulo requer um conhecimento básico de matemática.

Objetivos de aprendizagem

- Explicar em que consiste um algoritmo.
- Compreender a estrutura e o uso dos principais tipos de variáveis numéricas, caracteres e de bit.
- Programar utilizando técnicas de ciclo.
- Compreender os processos de acesso a periféricos.

Âmbito de conteúdos

- Algoritmia e Fluxogramas.
- Instruções Lógicas e Aritméticas.
- Instruções de transferência de dados.
- Instruções de salto.
- Temporizações.
- Acesso a periféricos.

Conceitos básicos de programação

Introdução

Desde o início da existência do homem que este tem procurado criar maquinas que o auxiliem nos seus trabalhos, diminuindo esforços e economizando tempo. Dentre essas máquinas, o computador tem-se mostrado uma das mais versáteis, rápidas e seguras.
O computador é capaz de auxiliar em qualquer coisa que lhe seja solicitada, é consciente, trabalhador e possui muita energia, mas não tem iniciativa, nenhuma independência, não é criativo nem inteligente, por isso precisa receber instruções nos mínimos detalhes.
A finalidade de um computador é receber, manipular e armazenar dados. Se for apenas visto como uma caixa composta por circuitos eletrónicos, cabos e fontes de alimentação, certamente ele não tem utilidade alguma. O computador só consegue armazenar dados em discos, imprimir relatórios, gerar gráficos, realizar cálculos, entre outras funções, por meio de programas. Portanto, a sua finalidade principal é realizar a tarefa de processamento de dados, isto é, receber dados por um dispositivo de entrada (por exemplo, teclado, rato, scanner, entre outros), realizar operações com esses dados e gerar uma resposta que será expressa num dispositivo de saída (por exemplo, impressora, monitor, entre outros).

Portanto, um computador possui duas partes diferentes que trabalham juntas: o hardware, composto pelas partes físicas, e o software, composto pelos programas.
Quando queremos criar ou desenvolver um software para realizar determinado tipo de processamento de dados, devemos escrever um programa ou vários programas interligados. No entanto, para que o computador compreenda e execute esse programa, devemos escreve-lo usando uma linguagem que tanto o computador quanto o criador de software entendam. Essa linguagem e chamada de linguagem de programação.
As etapas para o desenvolvimento de um programa são:

- Analise - Nesta etapa estuda-se o enunciado do problema para definir os dados de entrada, o processamento e os dados de saída.
- Algoritmo - Ferramentas do tipo descrição narrativa, fluxograma ou português estruturado são utilizados para descrever o problema com as suas soluções.

- Codificação - O algoritmo é transformado em códigos da linguagem de programação escolhida para se trabalhar.

Portanto, um programa é a codificação de um algoritmo numa linguagem de programação.

Conceito de Algoritmo

A seguir são apresentados alguns conceitos de algoritmos.

“Algoritmo é uma sequência de passos que visa atingir um objetivo bem definido.” (FORBELLONE, 1999)

“ Algoritmo é a descrição de uma sequência de passos que deve ser seguida para a realização de uma tarefa.” (ASCENCIO, 1999)

“ Algoritmo é uma sequência finita de instruções ou operações cuja execução, em tempo finito, resolve um problema computacional, qualquer que seja a sua instância.” (SALVETTI, 1999)

“ Algoritmo são regras formais para a obtenção de um resultado ou da solução de um problema, englobando fórmulas de expressões aritméticas.’’ (MANZANO, 1997)

“ Ação é um acontecimento que, a partir de um estado inicial, apos um período de tempo finito, produz um estado final previsível e bem definido. Portanto, um algoritmo é a descrição de um conjunto de comandos que, obedecidos, resultam numa sucessão finita de ações.” (FARRER, 1999)

Analisando as definições anteriores, podemos perceber que executamos no dia-a-dia vários algoritmos, como se pode observar nos exemplos a seguir.

Algoritmo 1 - Somar três números

Passo 1 — Receber os três números.

Passo 2 — Somar os três números.

Passo 3 — Mostrar o resultado obtido.

Algoritmo 2 - Fazer um sanduiche

Passo 1 — Buscar o pão.

Passo 2 — Cortar o pão ao meio.

Passo 3 — Pegar na maionese.

Passo 4 — Passar a maionese no pão.

Passo 5 — Buscar e cortar alface e tomate.

Passo 6 — Colocar alface e tomate no pão.

Passo 7 — Buscar o hambúrguer.

Passo 8 — Fritar o hambúrguer.

Passo 9 — Colocar o hambúrguer no pão.

Algoritmo 3 - Trocar uma lâmpada

Passo 1 — Buscar uma lâmpada nova.

Passo 2 — Buscar uma escada.

Passo 3 — Posicionar a escada de baixo da lâmpada queimada.

Passo 4 — Subir a escada com a lâmpada nova na mão.

Passo 5 — Retirar a lâmpada queimada.

Passo 6 — Colocar a lâmpada nova.

Passo 7 — Descer da escada.

Passo 8 — Testar o interruptor.

Passo 9 — Guardar a escada.

Passo 10 — Colocar a lâmpada velha no lixo.

Algoritmo 4 - Ir para a escola

Passo 1 — Acordar cedo.

Passo 2 — Ir à casa de banho.

Passo 3 — Abrir o armário para escolher uma roupa.

Passo 4 — Se o tempo estiver quente, escolher uma t-shirt e umas calças; caso contrário, escolher um agasalho e umas calças.

Passo 5 — Vestir a roupa escolhida.

Passo 6 — Tomar pequeno-almoço.

Passo 7 — Apanhar Microlet.

Passo 8 — Sair próximo à escola.

Algoritmo 5 – Levantar dinheiro no Multibanco

Passo 1 — Ir até uma caixa multibanco.

Passo 2 — Colocar o cartão.

Passo 3 — Marcar a senha.

Passo 4 — Escolher a quantia desejada

Passo 5 — Se o saldo for maior ou igual à quantia desejada, levantar; caso contrário, mostrar mensagem de impossibilidade de levantamento.

Passo 6 — Retirar o cartão.

E se estivermos a pensar: " Mas eu realizo essas atividades de maneira diferente!". Este pensamento está correto, pois às vezes um problema pode ser resolvido de diversas maneiras, porem, obtendo a mesma resposta, ou seja, podem existir vários algoritmos para solucionar o mesmo problema.

Método para a Construção de algoritmos

Para a construção de qualquer tipo de algoritmo, é necessário seguir estes passos:

a. Compreender completamente o problema a ser resolvido, destacando os pontos mais importantes e os objetos que o compõem.
b. Definir os dados de entrada, ou seja, quais os dados que serão fornecidos e quais objetos fazem parte do cenário/problema.
c. Definir o processamento, ou seja, quais os cálculos a serem efetuados e quais as restrições para esses cálculos. O processamento é responsável pela transformação dos dados de entrada em dados de saída.
d. Alem disso, deve-se verificar quais os objetos que são responsáveis pelas atividades.
e. Definir os dados de saída, ou seja, quais os dados que se obtêm depois do processamento.
f. Construir o algoritmo utilizando um dos tipos descritos na próxima secção.
g. Testar o algoritmo realizando simulações.

Tipos de Algoritmos

Os três tipos mais utilizados de algoritmos são: descrição narrativa, fluxograma e pseudocódigo.

Descrição narrativa

A descrição narrativa consiste em analisar o enunciado do problema e escrever, utilizando uma linguagem natural (por exemplo, a língua portuguesa ou tétum), os passos a serem seguidos para a sua resolução.

Vantagem: não é necessário aprender nenhum conceito novo, pois uma língua natural, neste ponto, já é bem conhecida.

Desvantagem: a língua natural abre espaço para varias interpretações, o que posteriormente dificultara a transcrição desse algoritmo para o programa.

Fluxograma

O fluxograma consiste em analisar o enunciado do problema e escrever, utilizando símbolos gráficos predefinidos (Tabela 1.1), os passos a serem seguidos para a sua resolução.

Vantagem: o entendimento de elementos gráficos é mais simples que o entendimento de textos.

Desvantagem: é necessário aprender a simbologia dos fluxogramas e, alem disso, o algoritmo resultante não apresenta muitos detalhes, dificultando a sua transcrição para um programa.

Pseudocodigo

O pseudocómico consiste em analisar o enunciado do problema e escrever, por meio de regras predefinidas, os passos a serem seguidos para a sua resolução.

Vantagem: a passagem do algoritmo para qualquer linguagem de programação é quase imediata, bastando conhecer as palavras reservadas dessa linguagem que serão utilizadas.

Desvantagem: é necessário aprender as regras do pseudocodigo, que serão apresentadas nas próximas secções.

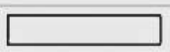
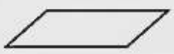
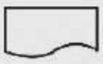
	Símbolo utilizado para indicar o início e o fim do algoritmo.
	Permite indicar o sentido do fluxo de dados. Serve exclusivamente para conectar os símbolos ou blocos existentes.
	Símbolo utilizado para indicar cálculos e atribuições de valores.
	Símbolo utilizado para representar a entrada de dados.
	Símbolo utilizado para representar a saída de dados.
	Símbolo utilizado para indicar que deve ser tomada uma decisão, apontando a possibilidade de desvios.

Tabela 1.1: Conjunto de símbolos utilizados no fluxograma.

Exemplos de Algoritmos

Os exemplos a seguir mostram alguns algoritmos desenvolvidos com os três tipos citados anteriormente,

- Faça um algoritmo para mostrar o resultado da multiplicação de dois números.

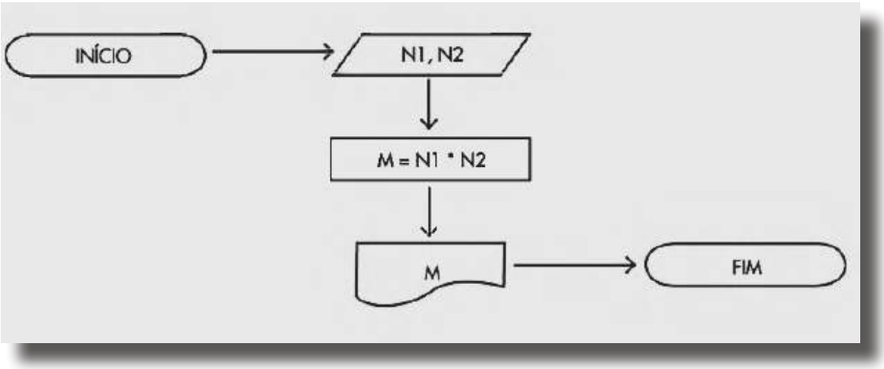
Algoritmo em descrição narrativa:

Passo 1 — Receber os dois números que serão multiplicados.

Passo 2 — Multiplicar os números.

Passo 3 — Mostrar o resultado obtido na multiplicação.

Algoritmo em fluxograma:



Algoritmo em pseudo código:

ALGORITMO

DECLARE N1, N2, M NUMÉRICO

ESCREVA “Escreva dois números”

LEIA N1, N2

M ← N1 * N2

ESCREVA “Multiplicação = “, M

FIM_ALGORITMO.

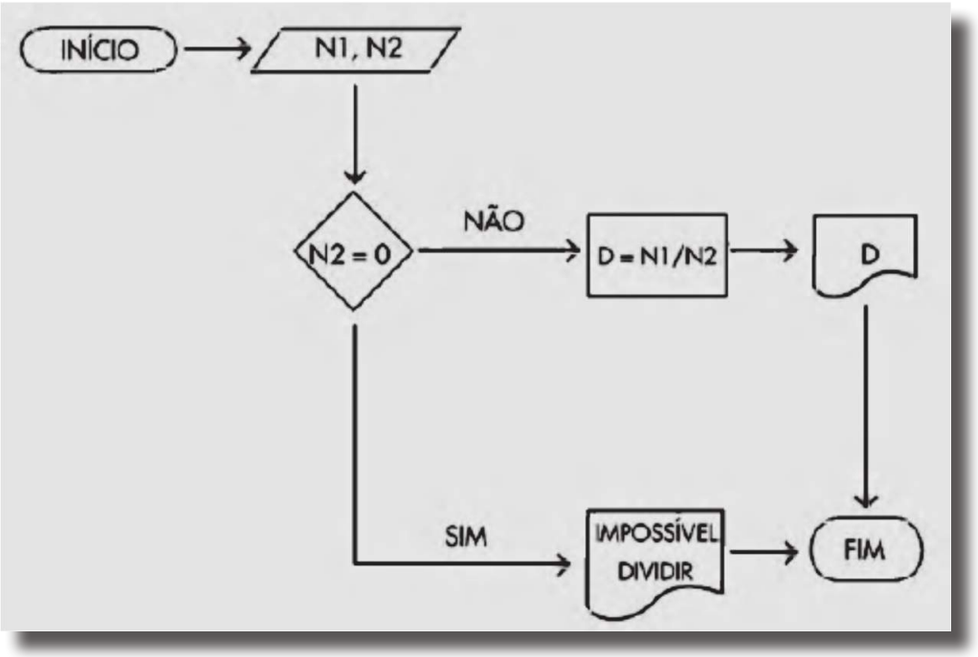
- Faça um algoritmo para mostrar o resultado da divisão de dois números.

Algoritmo em descrição narrativa:

Passo 1 — Receber os dois números que serão divididos.

P asso 2 — Se o segundo número for igual a zero, não poderá ser feita a divisão, pois não existe divisão por zero; caso contrário, dividir os números e mostrar o resultado da divisão.

Algoritmo em fluxograma:



Algoritmo em pseudo código:

ALGORITMO

DECLARE N1, N2, D NUMÉRICO

ESCREVA “Escreva dois números”

LEIA N1, N2

SE N2 = 0

ENTÃO ESCREVA “Impossível dividir”

SENÃO INICIO

D ← N1/N2

ESCREVA “Divisão = “, D

FIM

FIM ALGORITMO.

- Faça um algoritmo para calcular a média aritmética entre duas notas de um aluno e mostrar a sua situação, que pode ser aprovado ou reprovado.

Algoritmo em descrição narrativa:

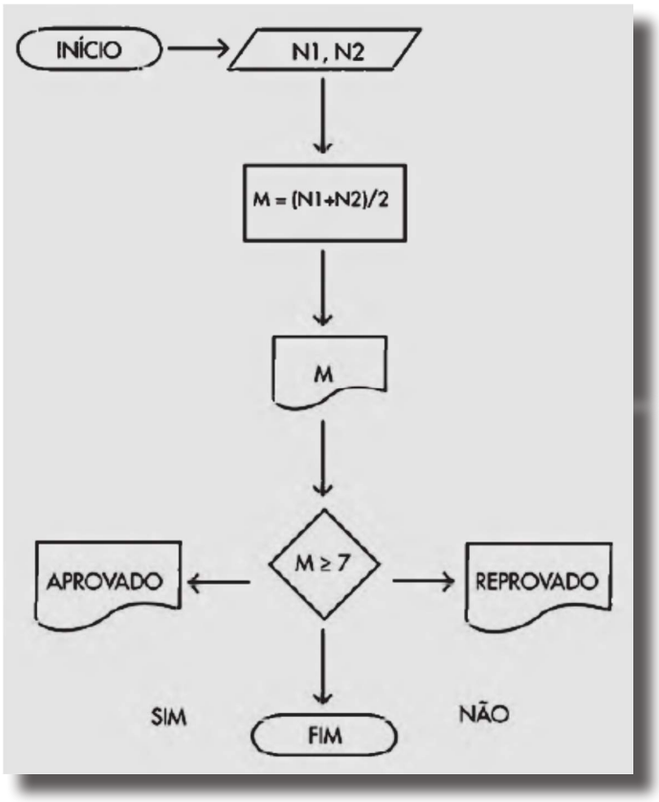
Passo 1 — Receber as duas notas.

Passo 2 — Calcular a média aritmética.

Passo 3 — Mostrar a média aritmética.

Passo 4 — Se a média aritmética for maior ou igual a 5, então a situação do aluno é aprovado; caso contrário, a situação é reprovado.

Algoritmo em fluxograma:



Algoritmo em pseudo código:

ALGORITMO

DECLARE N1, N2, M NUMÉRICO

ESCREVA “Escreva as duas notas”

LEIA N1, N2

M ← (N1 + N2)/2

ESCREVA “Media = “,M

SE M 5

ENTÃO ESCREVA “Aprovado”

SENÃO ESCREVA “Reprovado”

FIM_ALG0RITM0.

- Faça um algoritmo para calcular o novo salario de um funcionário. Sabe-se que os funcionários que recebem atualmente salario de até $ 500 terão aumento de 20%; os demais terão aumento de 10%.

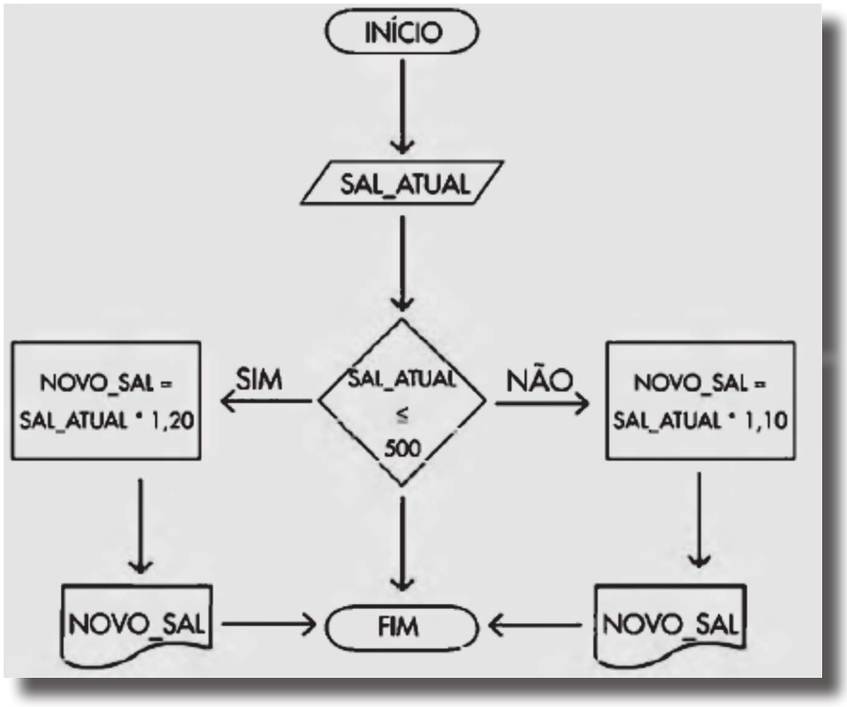
Algoritmo em descrição narrativa:

Passo 1 — Receber o salário atual do funcionário.

Passo 2 — Se o salário atual do funcionário for de até $ 500, calcular o novo salário com percentual de aumento de 20 %; caso contrário, calcular o novo salário com percentual de aumento de 10%.

Passo 3 — Mostrar o novo salário.

Algoritmo em fluxograma:



Algoritmo em pseudo código:

ALGORITMO

DECLARE SAL_ATUAL, NOVO_SAL NUMÉRICO

ESCREVA “Escreva o salário atual do funcionário”

LEIA SAL_ATUAL

SE SAL_ATUAL≤ 500

ENTÃO NOVO_SAL ← SAL_ATUAL * 1,20

SENÃO N0VO_SAL ← SAL_ATUAL * 1,10

ESCREVA “Novo salário = “,NOVO_SAL

FIM ALGORITMO.

Conceito de variável

Duas pessoas estão a conversar e precisam de realizar uma conta.

A primeira pessoa diz: " Vamos somar dois números" e continua: " O primeiro número é 5".

A segunda pessoa guarda o primeiro número na cabeça, ou seja, na memória.

A primeira pessoa diz: "O segundo número é 3".

A segunda pessoa também guarda o segundo número na cabeça, sem esquecer o primeiro número, ou seja, cada número foi armazenado em posições diferentes da memória humana, sem sobreposição.

A primeira pessoa pergunta: "Qual é o resultado da soma?"

A segunda pessoa recupera os valores armazenados na memória, realiza a conta e responde que o resultado é 8.

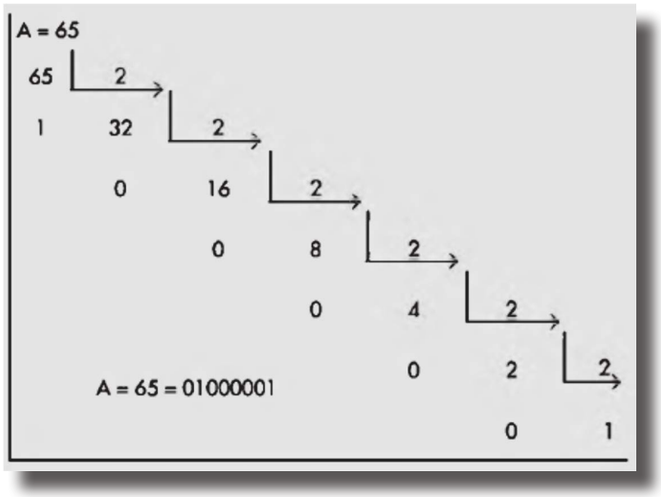
Um algoritmo é, posteriormente, um programa, recebem dados, que precisam ser armazenados no computador para serem utilizados no processamento. Esse armazenamento é feito na memória.

Todos os computadores trabalham com sistema numérico binário. Neste sistema, os dados são transformados em 0 e 1 ('zeros' e 'uns') para, então, serem armazenados na memória. Cada dígito binário (0 ou 1) ocupa porções de memória chamadas byte (8 bits), e cada byte é identificado e acedido por meio de um endereço.

Todos os caracteres existentes possuem um correspondente numérico na tabela ASCII, que é transformado em caractere binário pelo método da divisão para, então, ser armazenado na memória.

Desta maneira, uma variável representa uma posição de memória. Possui nome e tipo, e o seu conteúdo pode variar ao longo do tempo, durante a execução de um programa. Embora uma variável possa assumir diferentes valores, ela só pode armazenar um valor a cada instante.

Exemplo de transformação em binário:



Todos os computadores possuem uma tabela de alocação que contem o nome da variável, o tipo (para saber quantos bytes ocupara) e o seu endereço inicial de armazenamento. Desta maneira, quando queremos procurar algum dado na memória, basta sabermos o nome da variável que o computador, por meio da tabela de alocação, procura automaticamente.

Tipos de dados

Os tipos de dados mais utilizados são: numéricos, lógicos e literais ou caracteres.

Numéricos

Os dados numéricos dividem-se em dois grupos: inteiros e reais.

Os números inteiros podem ser positivos ou negativos e não possuem parte fracionária.

Exemplos de dados numéricos inteiros:

- -23
- 98
- 0
- -357
- 237
- -2

Os números reais podem ser positivos ou negativos e possuem parte fracionária.

Exemplos de dados numéricos reais:

- 23,45
- 346,89
- -34,88
- 0, 0
- -247,0

Observação: Os números reais seguem a notação da língua inglesa, ou seja, a parte decimal é separada da parte inteira por um. (ponto) e não por uma, (virgula).

Lógicos

São também chamados dados booleanos (por causa da álgebra de Boole) e podem assumir os valores: verdadeiro ou falso.

Caracteres

São dados formados por um único caractere ou por uma cadeia de caracteres. Estes caracteres podem ser as letras maiúsculas, as letras minúsculas, os números (não podem ser usados para cálculos) e os caracteres especiais (&, #, @, ?, +).

Exemplos de dados literais:

- ‘aluno’
- ‘1234’
- ‘@ internet’
- ‘0.34’
- ‘1 + 2’

Formação de identificadores

Os identificadores são os nomes das variáveis, dos programas, das constantes, das rotinas, das unidades etc.

As regras básicas para a formação dos identificadores são:

- Os caracteres que se podem utilizar são: os números, as letras maiúsculas, as letras minúsculas e o caractere sublinhado.
- O primeiro caractere deve ser sempre uma letra ou o caractere sublinhado.
- Não são permitidos espaços em branco e caracteres especiais (@, $, +, -, %, !).
- Não podemos usar as palavras reservadas nos identificadores, ou seja, palavras que pertençam a uma linguagem de programação.

Exemplos de identificadores

Exemplos de identificadores validos:

- A
- a
- nota
- NOTA
- X5
- A32
- NOTA1
- MATRICULA
- nota_1
- dia
- IDADE

Exemplos de identificadores inválidos:

- 5b - por começar com número;
- e 12 - por conter espaço em branco;
- x-y - por conter o caractere especial;
- prova 2n - por conter espaço em branco;
- nota (2) - por conter os caracteres especiais ();
- case - por ser palavra reservada;
- set - por ser palavra reservada.

Linguagem PASCAL

A linguagem PASCAL foi desenvolvida em 1968 por Niklaus Wirth, na Suíça, destinada principalmente à programação científica, mas a sua grande evolução permitiu que, nos dias de hoje, seja utilizada para qualquer fim.

Por se tratar de uma linguagem estruturada, isto é, uma linguagem que possui regras para a escrita dos seus programas, é muito utilizada por alunos que começam a aprender programação. A linguagem PASCAL possui um ambiente integrado de desenvolvimento chamado Pascal Zim com as seguintes características:

- Apresenta um editor que permite ao desenvolvedor do programa escrever, guardar e modificar o código dos seus programas.
- Possui um compilador que converte os códigos dos programas em instruções de máquina e permite-lhe compilar, ou seja, verificar a existência de erros de sintaxe nos programas sem retornar ao sistema operacional.
- Dispõe de um depurador que permite inspecionar um programa durante a sua execução, facilitando a localização de erros.
- Conta com um sistema de ajuda ativo que oferece diferentes níveis de informações.
- Possui ainda o ambiente de execução propriamente dito, que permite executar os programas sem sair do Pascal Zim (com ficheiros de extensão PAS) ou, se preferir, permite criar ficheiros a serem executados fora do ambiente do Pascal Zim (com ficheiros de extensão EXE).

Portanto, para fazer um programa utilizando a linguagem de programação PASCAL devemos:

- Analisar o enunciado do problema, algoritmo e codificação, utilizando o editor do ambiente de desenvolvimento.
- Pascal Zim.
- Compilar, utilizando o compilador do ambiente de desenvolvimento Pascal Zim.
- Executar.

Paradigmas de Programação

Um paradigma de programação está intimamente relacionado à forma de pensar do programador e como ele procura a solução para os problemas. É o paradigma que permite ou proíbe a utilização de algumas técnicas de programação. Ele é capaz, ainda, de mostrar como o programador analisou e abstraiu o problema a resolver.

Existem vários paradigmas de programação: estruturado, orientado a objetos, logico, funcional, dentre outros.

Vamos analisar com mais detalhes os paradigmas estruturados e orientados a objetos.

Pelo paradigma estruturado (também conhecido como imperativo), qualquer problema pode ser resolvido utilizando três estruturas: sequencial, condicional e iterativa (repetição). Alem disso, procura encontrar uma forma de resolver um problema complexo em partes mais pequenas sendo assim mais simples do que, trabalhadas conjuntamente, permitam soluciona-lo.

A ideia é que, utilizando corretamente tais estruturas, o recurso da modularização e a parametrização, seja possível criar programas com menor repetição possível de linhas de comandos.

Já o paradigma orientado a objetos compreende o problema como uma coleção de objetos interagindo por meio de trocas de mensagem. Os objetos são estruturas de dados incluindo logica. Desta maneira, um conjunto de objetos com informações comuns e com o mesmo comportamento dá origem a uma classe.

Exemplificando, um programador que utiliza o paradigma estruturado analisa o problema tentando relacionar as ações que deverão ser executadas e como poderão ser divididas em módulos. Um programador que utilize o paradigma orientado a objetos analisaria o mesmo problema tentando identificar os objetos que compõem essa realidade e como interagem.

Como o paradigma está ligado à forma de pensar do programador, o simples fato de se utilizar, por exemplo, uma linguagem com suporte nativo à orientação a objetos não implica que a solução apresentada seja orientada a objetos, ou, então, muitas soluções não estruturadas são feitas utilizando linguagens com suporte à estruturação.

Estrutura Sequencial

Estrutura Sequencial em Algoritmos

ALGORITMO

DECLARE

bloco de comandos

FIM_ALGORITMO.

Declaração de Variáveis em Algoritmos

As variáveis são declaradas após a palavra declare e os tipos mais utilizados são: numérico (para variáveis que receberão números), caractere (para variáveis que receberão caracteres) e lógico (para variáveis que receberão apenas dois valores: verdadeiro ou falso).

Exemplo:

DECLARE

X NUMERICO

Y, Z LITERAL

TESTE LÓGICO

Comando de Atribuição em Algoritmos

O comando de atribuição é utilizado para conceder valores ou operações a variáveis, sendo representado pelo símbolo.

Exemplo:

x ← ←4

x ← x + 2

y ← “aula”

teste ← falso

Comando de Entrada em Algoritmos

O comando de entrada é utilizado para receber dados escritos pelo utilizador, que serão armazenados em variáveis. Esse comando é representado pela palavra leia.

Exemplo:

LEIA X

Um valor escrito pelo utilizador será armazenado na variável x.

LEIA Y

Um ou vários caracteres escritos pelo utilizador serão armazenados na variável y.

Comando de Saída em Algoritmos

O comando de saída é utilizado para mostrar dados no ecrã ou na impressora. Este comando é representado pela palavra escreva, e os dados podem ser conteúdos de variáveis ou mensagens.

Exemplo:

ESCREVA X

Mostra o valor armazenado na variável x.

ESCREVA “Conteúdo de Y = “, Y

Mostra a mensagem “Conteúdo de y = “ e em seguida o valor armazenado na variável y.

Estrutura Sequencial em Pascal

```
PROGRAM nome;
USES nomes_das_unidades;
VAR nome_das_variaveis : tipo;
BEGIN
    bloco de comandos;
END.
```

As unidades são bibliotecas utilizadas pela linguagem PASCAL para a correta execução do programa. A unidade CRT é obrigatória em todos os programas, pois faz a adequação do hardware com o programa.

Declaração de variáveis em PASCAL

As variáveis são declaradas após a palavra var e os tipos mais utilizados são: integer (para números inteiros), real (para números reais), char (para um caractere), string (para vários caracteres) e boolean (para verdadeiro ou falso).

Exemplo:

```
VAR X: INTEGER;
        Y, Z: REAL;
        NOME: STRING;
        SEXO: CHAR;
        TESTE: BOOLEAN;
```

Os identificadores são os nomes das variáveis, dos programas, das constantes, das rotinas e unidades, entre outras.

As regras básicas para a formação dos identificadores são:

- Podem ter qualquer tamanho. Entretanto, apenas os 63 primeiros caracteres são utilizados pelo compilador.
- Os caracteres que podem ser utilizados na formação dos identificadores são: os números, as letras maiúsculas, as letras minúsculas e o caractere sublinhado.
- O compilador não faz distinção entre letras maiúsculas e minúsculas, portanto, o identificador NUM é exatamente igual ao identificador num.
- O primeiro caractere deve ser sempre uma letra ou o caractere sublinhado.
- Não são permitidos espaços em branco e caracteres especiais (@, $, +, -, %, !) .
- Não é permitido usar palavras reservadas.

Palavras Reservadas são nomes utilizados pelo compilador para representar comandos, operadores e nomes de secções de programas. As palavras reservadas da linguagem PASCAL são:

and	goto	program
asm	if	record
array	implementation	repeat

begin	in	set
case	inherited	shl
const	inline	shr
constructor	interface	string
destructor	label	then
div	library	to
do	mod	type
downto	nil	unit
else	not	until
end	object	uses
exporte	of	var
file	or	while
for	packed	with
function	procedure	xor

Os tipos de dados mais utilizados na linguagem PASCAL estão descritos na tabela a seguir:

TIPO	FAIXA DE VALORES	TAMANHO (aproximado)
shortint	-128 a 127	8 bits
integer	-32.768 a 32.767	16 bits
longint	-2.147.483.648 a 2.147.483.647	32 butes
byte	0 a 255	8 bites
word	0 a 65.535	16 bites
real	2,9 x 10-39 a 1,7 x 1038 (11 a 12 dígitos com sinal)	6 bytes
single	1,5 x 10-45 a 3,4 x 1038 (7 a 8 dígitos com sinal)	4 bytes
double	5,0 x 10-324 a 1,7 x 10308 (15 a 16 dígitos com sinal)	8 bytes
extended	5,0 x 10-4932 a 1,7 x 104932 (19 a 20 dígitos com sinal)	10 bytes
comp	-9,2 x 1018 a 9,2 x 1018 (19 a 20 dígitos com sinal)	8 bytes
boolean	true ou false	8 bites
wordbool	true ou false	16 bites
longbool	true ou false	32 bits
bytebool	true ou false	8 bits
char	1 caractere qualquer	1 byte
string	cadeia de caracteres (no max 255)	tantos bytes quantos forem os caracteres

Comando de atribuição em PASCAL

O comando de atribuição é utilizado para dar valores ou operações a variáveis, sendo representado por: = (o sinal de dois-pontos e de igualdade).

Exemplo:

```
x : = 4;
x : = x + 2;
y : = 2.5;
nome : = 'AULA';
sexo : = 'F ';
teste : = false;
```

Em PASCAL, os caracteres literais são representados entre apóstrofos; os números reais utilizam o ponto como separador decimal; cada comando é finalizado com o sinal de ponto-e-vírgula.

Comando de entrada em PASCAL

O comando de entrada é utilizado para receber dados escritos pelo utilizador. Estes dados são armazenados em variáveis. Este comando é representado pela palavra readln. A sua sintaxe está representada a seguir:

Sintaxe:

```
READLN (nome_da_variavel);
READLN (nome_da_variavell,nome_da_variavel2);
```

Exemplo:

```
READLN (X) ;
```

Um valor escrito pelo utilizador será armazenado na variável x.

```
READLN (NOME);
```

Um ou vários caracteres escritos pelo utilizador serão armazenados na variável NOME.

Comando de saída em PASCAL

O comando de saída é utilizado para mostrar dados no ecrã ou na impressora. Este comando é representado pelas palavras write ou writeln e os dados podem ser conteúdos de variáveis ou mensagens.

Sintaxe:

WRITE (nome_da_variável);

WRITELN (nome_da_variável);

WRITE (‘ mensagem’);

WRITELN (‘mensagem’);

WRITE (‘ mensagem’, nome_da_variável);

WRITELN (‘mensagem’, nome_da_variável);

Exemplo:

WRITELN (X);

WRITE (X);

Mostra o valor armazenado na variável x.

WRITELN (‘Conteúdo de Y = ‘ , Y);

WRITE (‘Conteúdo de Y = \Y);

Mostra a mensagem “Conteúdo de y = “ e de seguida o valor armazenado na variável y.

A diferença entre estes comandos é que o comando writeln mostra o seu conteúdo e passa o cursor para a linha de baixo, enquanto o comando write mantem o cursor na mesma linha após mostrar a mensagem.

Comentários em PASCAL

Os comentários não são interpretados pelo compilador, servem apenas para esclarecer o programador. Constituem excelentes instrumentos de documentação e devem sempre estar entre {.....} ou entre (* ..... *).

Operadores e funções predefinidas em PASCAL

A linguagem PASCAL possui operadores e funções predefinidas destinados a cálculos matemáticos. Alguns são apresentados a seguir.

OPERADOR	EXEMPLO	COMENTÁRIO
: =	x : = y	O conteúdo da variável Y é atribuído à variável X. (A uma variável pode ser atribuído o conteúdo de outra, um valor constante ou, ainda, o resultado de uma função.)
+	x + y	Soma o conteúdo de X e de Y.
-	x – y	Subtrai o conteúdo de Y do conteúdo de X.
*	x * y	Multiplica o conteúdo de X pelo conteúdo de Y.
/	x / y	Obtém o quociente da divisão de X por Y.
div	x div y	Obtém o quociente inteiro da divisão de X por Y.
mod	x mod y	Obtém o resto da divisão de X por Y.

Observações:

a. Os operadores div e mod só podem ser aplicados com operandos do tipo inteiro.
b. O operador / conduz sempre a um resultado real.
c. Com os operadores * e /, se pelo menos um dos operandos for real, então o resultado será real.

OPERADOR	EXEMPLO	COMENTÁRIO
=	x = y	O conteúdo de X e igual ao conteúdo de Y.
<>	x <> y	O conteúdo de X e diferente do conteúdo de Y.
<=	x <= y	O conteúdo de X e menor ou igual ao conteúdo de Y.
>=	x >= y	O conteúdo de X e maior ou igual ao conteúdo de Y.
<	x < y	O conteúdo de X e menor que o conteúdo de Y.
>	x > y	O conteúdo de X e maior que o conteúdo de Y.

FUNÇÕES MATEMÁTICAS		
FUNÇÃO	EXEMPLO	COMENTÁRIO
abs	abs (x)	Obtém o valor absoluto de X.
exp	exp (x)	Obtém o logaritmo natural e elevado a potencia X.
log	log (x)	Obtém o logaritmo natural de X.
trunc	trunc (x)	Obtém a parte inteira do número real armazenado em X.
frac	frac (x)	Obtém a parte fracionaria do número real armazenado em X.
round	round (x)	Arredonda X.
sin	sin (x)	Calcula o seno de X (X deve estar representado em radianos).
cos	cos (x)	Calcula o cosseno de X (X deve estar representado em radianos).
pi	Pi	Retorna o valor de π.
sqrt	sqrt (x)	Calcula a raiz quadrada de x.
sqr	sqr (x)	Calcula X elevado ao quadrado.
inc	inc (x,y)	Incrementa a variável X com o valor da variável Y.
dec	dec (x,y)	Decrementa a variável X com o valor da variável Y.

Observação 1:

Por não existir o operador de potenciação, temos:

AB = EXP (B*LN(A))

Exemplo:

34 = exp (4*ln (3))

510 = exp (10*ln (5))

Observação 2:

As funções SIN e COS esperam receber argumentos no formato de radianos; para receber argumentos em graus, siga o próximo exemplo. Na linguagem PASCAL não existe uma função para tangente;

Sendo assim, utilizamos seno/cosseno.

Exemplo com variável para o valor de π:

```
VALORPI := 3.1415;
READLN (X); { X EM GRAUS }
Y := SIN ((VALORPI * X) / 180);
```

Exemplo utilizando a função pi:

```
READLN (X); { X EM GRAUS }
Y := SIN ((PI * X) / 180);
```

As prioridades entre os operadores são:

Iº) ( )

2º) funções

3º) *, /, DIV, MOD

4º) +, -

Quando se tem uma expressão em que os operadores têm a mesma prioridade, a expressão e resolvida da esquerda para a direita.

Exemplos:

2 +3 - 4 = 5 - 4 = 1

2 * 4/2 = 8/2 = 4

Exercícios Resolvidos

Exercício 1:

Faça um programa que receba quatro números inteiros, calcule e mostre a soma desses números.

Resolução:

Solução Algoritmo

ALGORITMO

```
DECLARE n1, n2, n3, n4, soma NUMÉRICO
LEIA n1, n2, n3, n4
soma ← n1 + n2 + n3 + n4
```

```
        ESCREVA soma
FIM_ALGORITMO.
```

Solução 1 PASCAL

```
PROGRAM EX1;
        USES CRT;
        VAR n1, n2, n3, n4, soma: INTEGER;
BEGIN
        {Recebe os quatro números}
        READLN (n1, n2, n3, n4);
        {Soma os números escritos}
        soma := n1 + n2 + n3 + n4;
        {Mostra o resultado da soma}
        WRITELN (soma);
        {Para o programa à espera de um enter}
READLN;


END.
```

Solução 2 PASCAL

```
PROGRAM EX1;
        USES CRT;
        VAR n1, n2, n3, n4, soma: INTEGER;
BEGIN
        {Limpa o ecrã}
        CLRSCR;
        {Mostra mensagem antes da leitura dos números}
        WRITELN ('Escreva quatro números');
        {Recebe os quatro neros}
        READLN (n1, n2, n3, n4);
        {Soma os números digitados}
        soma := n1 + n2 + n3 + n4;
        {Mostra mensagem e o resultado da soma}
        WRITELN ('Resultado da soma = ',soma);
```

{Para o programa à espera de um enter}

READLN;

END.

Exercício 2:

Faça um programa em PASCAL que receba três notas, calcule e mostre a média aritmética entre elas.

Resolução:

Solução 1 Algoritmo

ALGORITMO

DECLARE nota1, nota2, nota3, media NUMÉRICO

LEIA nota1, nota2, nota3

media ← (nota1 + nota2 + nota3)/3

ESCREVA media

FIM_ALGORITMO.

Solução 2 Algoritmo

ALGORITMO

DECLARE nota1, nota2, nota3, soma, media NUMÉRICO

LEIA nota1, nota2, nota3

soma ← notai + nota2 + nota3

media ← soma/3

ESCREVA média

FIM_ALGORITMO.

Solução 1 PASCAL

PROGRAM EX2;

USES CRT;

VAR nota1, nota2, nota3, media: REAL;

BEGIN

```
        {Limpa o ecrã}
        CLRSCR;
        {Recebe as três notas}
        READLN (nota1, nota2, nota3);
        {Calcula a média aritmética}
        media := (nota1 + nota2 + nota3)/3;
        {Mostra a média formatada para duas casas decimais}
        WRITELN (media:4:2);
        {Para o programa à espera de um enter}
        READLN;

END.
```

Solução 2 PASCAL

```
PROGRAM EX2;
        USES CRT;
        VAR nota1, nota2, nota3, soma, media: REAL;
BEGIN
        {Limpa o ecrã}
        CLRSCR;
        {Mostra mensagem antes da leitura das notas}
        WRITELN ('Digite as três notas');
        {Recebe as três notas}
        READLN (nota1, nota2, nota3);
        {Calcula a média}
        soma := (nota1 + nota2 + nota3);
        media := soma/3;
        {Mostra a média formatada com duas casas decimais}
        WRITELN ('Média = ',media:4:2);
        {Para o programa à espera de um enter}
        READLN;
```

END.

Exercício 3:

Faça um programa que receba três notas e seus respetivos pesos, calcule e mostre a média ponderada dessas notas.

Resolução:

Solução 1 Algoritmo

```
ALGORITMO
        DECLARE nota1, nota2, nota3, peso1, peso2, peso3, média NUMÉRICO
        LEIA nota1, nota2, nota3, peso1, peso2, peso3
        média ← (nota1 * peso1 + nota2 * peso2 + nota3 * peso3)/(peso1 + peso2 + peso3)
        ESCREVA média
FIM_ALGORITMO.
```

Solução 2 Algoritmo

```
ALGORITMO
        DECLARE nota1, nota2, nota3, peso1, peso2, peso3 NUMÉRICO
                soma1, soma2, soma3, total, media NUMERICO
        LEIA notai1 nota2, nota3, peso1, peso2, peso3
        soma1 ← nota1 * peso1
        soma2 ← nota2 * peso2
        soma3 ← nota3 * peso3
        total ← peso1 + peso2 + peso3
        média (soma1 + soma2 + soma3)/total
        ESCREVA média
FIM_ALGORITMO.
```

Solução 1 PASCAL

```
PROGRAM EX3;
        USES CRT;
        VAR nota1, nota2, nota3, peso1, peso2, peso3, media: REAL;
```

```
BEGIN
        {Limpa o ecrã}
        CLRSCR;
        {Recebe as três notas e os três pesos}
        READLN (nota1, nota2, nota3, peso1, peso2, peso3);
        {Calcula a média ponderada}
        media := (nota1 * peso1 + nota2 * peso2 + nota3 * peso3)/(peso1 + peso2 + peso3);
        {Mostra a média ponderada formatada com duas casas decimais}
        WRITELN (media:5:2);
        {Para o programa à espera de um enter}
        READLN;

END.
```

Solução 2 PASCAL

```
PROGRAM EX3;
        USES CRT;
        VAR     nota1, nota2, nota3, peso1, peso2, peso3: REAL;
                soma1, soma2, soma3, total, media: REAL;
BEGIN
        {Limpa o ecrã}
        CLRSCR;
        {Mostra mensagem antes da leitura das três notas}
        WRITELN (‘ Escreva as três notas’);
        {Recebe as três notas}
        READLN (nota1, nota2, nota3);
        {Mostra mensagem antes da leitura dos três pesos}
        WRITELN (‘ Escreva os três pesos’);
        {Recebe os três pesos}
        READLN (peso1, peso2, peso3);
        {Calcula a média ponderada}
        soma1:=nota1 * peso1;
```

```
        soma2:=nota2 * peso2;
        soma3:=nota3 * peso3;
        total:=peso1 + peso2 + peso3;
        media := (soma1 + soma2 + soma3)/total;
        {Mostra a média ponderada formatada com duas casas decimais}
        WRITELN ('Média Ponderada = ',media:5:2);
        {Para o programa à espera de um enter}
        READLN;
END.
```

Exercício 4:

Faça um programa que receba o salario de um funcionário, calcule e mostre o novo salario, sabendo-se que este sofreu um aumento de 25%.

Resolução:

Solução 1 Algoritmo

```
ALGORITMO
        DECLARE sal, novosal NUMÉRICO
        LEIA sal
        novosal ← sal + sal * 25/100
        ESCREVA novosal
FIM_ALGORITMO.
```

Solução 2 Algoritmo

```
ALGORITMO
        DECLARE sal, aumento, novosal NUMÉRICO
        LEIA sal
        aumento ← sal * 25/100
        novosal ← sal + aumento
        ESCREVA novosal
FIM_ALGORITMO.
```

Solução 1 PASCAL

```
PROGRAM EX4;
        USES CRT;
        VAR sal, novosal: REAL;
BEGIN
        {Limpa o ecrã}
        CLRSCR;
        {Mostra mensagem antes da leitura do sal rio}
        WRITELN (Escreva o salário do funcionário');
        {Recebe o salário}
        READLN (sal);
        {Calcula o novo sal rio}
        novosal := sal + sal * 25/100;
        {Mostra o novo sal rio calculado com duas casas decimais}
        WRITELN ('Novo salário = ',novosal:5:2);
        {Para o programa à espera de um enter}
        READLN;
END.
```

Solução 2 PASCAL

```
PROGRAM EX4;
        USES CRT;
        VAR sal, aumento, novosal: REAL;
BEGIN
        {Limpa o ecrã}
        CLRSCR;
        {Mostra mensagem antes da leitura do salário}
        WRITELN ('Escreva o salário');
        {Recebe o salário}
        READLN (sal);
        {Calcula o aumento}
        aumento := sal * 25/100;
```

```
        {Calcula o novo salário}
        novosal := sal + aumento;
        {Mostra o novo sal rio formatado com duas casas decimais}
        WRITELN ('Novo salário = ', novosal:5:2);
        {Para o programa à espera de um enter}
        READLN;

END.
```

Exercício 5:

Faça um programa que receba o salario de um funcionário e o percentual de aumento, calcule e mostre o valor do aumento e o novo salario.

Resolução:

Solução Algoritmo

```
ALGORITMO
        DECLARE sal, perc, aumento, novosal NUMÉRICO
        LEIA sal, perc
        aumento ← sal * perc/100
        ESCREVA aumento
        novosal ← sal + aumento
        ESCREVA novosal
FIM_ALGORITMO.
```

Solução PASCAL

```
PROGRAM EX5;
        USES CRT;
        VAR sal, perc, aumento, novosal: REAL;
BEGIN
        {Limpa o ecrã}
        CLRSCR;
        {Mostra mensagem antes da leitura do salário}
```

```
        WRITELN (‘Digite o sal rio do funcion rio›);
        {Recebe o salário}
        READLN (sal);
        {Mostra mensagem antes da leitura do percentual de aumento}
        WRITELN (‘Escreva o percentual de aumento’);
        {Recebe o percentual de aumento}
        READLN (perc);
        {Calcula o valor do aumento}
        aumento := sal * perc/100;
        {Mostra o valor do aumento formatado com duas casas decimais}
        WRITELN (‘Aumento = ‘,aumento:5:2);
        {Calcula o novo salário}
        novosal := sal + aumento;
        {Mostra o valor do novo salário formatado com duas casas decimais}
        WRITELN (‘Novo salário = ‘,novosal:5:2);
        {Para o programa à espera de um enter}
        READLN;
END.
```


Estrutura Condicional

Estrutura Condicional Em Algoritmos

Estrutura Condicional Simples

SE condição

ENTÃO comando

O comando só será executado se a condição for verdadeira. Uma condição é uma comparação que possui dois valores possíveis: verdadeiro ou falso.

SE condição

ENTÃO INICIO

comando1

comando2

comando3

FIM

Os comandos 1, 2 e 3 só serão executados se a condição for verdadeira. As palavras início e fim serão necessárias apenas quando dois ou mais comandos forem executados.

Estrutura Condicional Composta

SE condição

ENTÃO comando1

SENÃO comando2

Se a condição for verdadeira, será executado o comando1 caso contrario, será executado o comando2.

SE condição

ENTÃO INICIO

comando1

comando2

FIM

SENÃO INICIO

comando3

comando4

FIM

Se a condição for verdadeira, o comando1 e o comando2 serão executados; caso contrário, o comando3 e o comando4 serão executados.

Estrutura Condicional Em Pascal

Estrutura Condicional Simples

IF condição

THEN comando;

O comando só será executado se a condição for verdadeira. Uma condição é uma comparação que possui dois valores possíveis: verdadeiro ou falso.

IF condição

THEN BEGIN

comando1;

comando2;

comando3;

END;

Os comandos 1, 2 e 3 só serão executados se a condição for verdadeira.

Estrutura Condicional Composta

IF condição

THEN comando1

ELSE comando2;

Se a condição for verdadeira, será executado o comando1; caso contrario, será executado o comando2.

```
IF condição
THEN BEGIN
        comando1;
        comando2;
        END
ELSE BEGIN
        comando3;
        comando4;
        END;
```

Se a condição for verdadeira, o comando1 e o comando2 serão executados; se for falsa, o comando3 e o comando4 serão executados.

NOTA: Antes do comando ELSE não existe ponte e virgula.

Estrutura Case

Em alguns programas, existem situações mutuamente exclusivas, isto é, se uma situação for executada, as demais não serão. Quando for este o caso, um comando seletivo será o mais indicado, e esse comando, em PASCAL, tem a seguinte sintaxe:

```
CASE seletor OF
        lista de alvos1: comando1;
        lista de alvos2: comando2;
        alvo3: comando3;
        alvo4: BEGIN
                comando4;
                comando5;
                END;
        END;
```

Se o seletor atingir a lista de alvos 1, o comando1 será executado; se atingir a lista de alvos2, o comando2 será executado; se atingir o alvo3, o comando3 será executado; se atingir o alvo4, então o comando4 será executado. Se nenhum alvo for atingido, nada será executado.

```
CASE seletor OF
        lista de alvos1:        BEGIN
                                comando1;
                                comando2;
                                END;
        lista de alvos2: cornando3;
        ELSE comando4;
        END;
```

Se o seletor atingir a lista de alvos1, o comando1 e o comando2 serão executados; se atingir a lista de alvos2, o comando3 será executado. Se nenhum alvo for atingido, será executado o comando4.

Os alvos podem ser valores únicos, listas de valores (separados por virgulas) ou faixas de valores, como no exemplo que se segue.

Exemplo:

```
program teste;
uses crt;
var i: integer;
begin
clrscr;
writeln('Escreva um numero');
readln(i) ;
case i of
        1: writeln ('Numero 1');
        2,5,6:writeln ('Numero 2 ou numero 5 ou numero 6');
        7..10:writeln{'Numero entre 7 e 10');
        else writeln{'outro número');
end;
readln;
end.
```

A restrição da estrutura case é que o seletor só pode ser uma variável do tipo char, integer ou boolean.

Operadores Lógicos

Os principais operadores lógicos são: and , or e not. que significam e, ou, não e são usados para conjunção, disjunção e negação, respetivamente.

TABELA E	TABELA OU	TABELA NÃO
V e V = V	V ou V = V	NÃO V = F
V e F = F	V ou F = V	NÃO F = V
F e V = F	F ou V = V	
F e F = F	F ou F = F	

NOTA: Na linguagem PASCAL, quando existe mais de uma condição, elas devem estar entre parenteses.

Exemplos:

```
IF x = 3
THEN WRITELN (‘Numero igual a 3’);
```

No exemplo, existe apenas uma condição, logo, os parenteses são opcionais.

```
IF (X > 5) AND (X < 10)
THEN WRITELN (‘Número entre 5 e 10’);
```

No exemplo anterior, existe mais de uma condição, logo, os parenteses são obrigatórios, ou seja, cada condição deve estar entre parenteses.

```
IF ((X = 5) AND (Y = 2)) OR (Y = 3)
THEN WRITELN (‘X é igual a 5 e Y é igual a 2, ou Y é igual a 3’);
```

No exemplo acima, existe mais de uma condição e mais de um tipo de operador logico, logo, alem dos parenteses de cada condição, devem existir ainda parenteses que indiquem a prioridade de execução das condições.

Neste exemplo, as condições com o operador and, ou seja, ((x = 5) and (Y = 2)), serão testadas, e o resultado será testado com a condição or (y = 3).

Aqui, as condições com o operador or, ou seja, ((y = 2) or (y = 3)), serão testadas, e o resultado será testado com a condição and (x = 5).

Exercícios Resolvidos

Exercício 1

A nota final de um estudante é calculada a partir de três notas atribuídas, respetivamente, a um trabalho de laboratório, a uma avaliação semestral e a um exame final. A média das três notas mencionadas obedece aos pesos a seguir:

NOTA	PESO
Trabalho de laboratório	2
Avaliação semestral	3
Exame final	5

Faça um programa que receba as três notas, calcule e mostre a média ponderada e o conceito que segue a tabela:

MÉDIA PONDERADA			CONCEITO
8,0	⟷	10,0	A
7,0	⟷	8,0	B
6,0	⟷	7,0	C
5,0	⟷	6,0	D
0,0	⟷	5,0	E

Solução Algoritmo

ALGORITMO

DECLARE nota_trab, aval_sem/ exame, media NUMÉRICO

ESCREVA “Escreva a nota do trabalho de laboratório: “

LEIA nota_trab

ESCREVA “Escreva a nota da avaliação semestral: “

LEIA aval_sem

ESCREVA “Escreva a nota do exame final: “

LEIA exame

```
media <— (nota_trab * 2 + aval_sem * 3 + exame * 5) / 10
ESCREVA “Media ponderada: “ , media
SE media >= 8 E media <= 10
        ENTÃO ESCREVA “Obteve conceito A”
SE media >= 7 E media < 8
        ENTÃO ESCREVA “Obteve conceito B”
SE media >= 6 E media < 7
        ENTÃO ESCREVA “Obteve conceito C”
SE media >= 5 E media < 6
        ENTÃO ESCREVA “Obteve conceito D”
SE media >= 0 E media < 5
        ENTÃO ESCREVA “Obteve conceito E”
FIM ALGORITMO
```

Solução 1 PASCAL

```
PROGRAM EX1;
        USES CRT;
        VAR nota_trab, aval_sem, exame, media: REAL;
BEGIN
        {Limpa o ecrã}
        CLRSCR;
        {Mostra mensagem antes da leitura da nota do laboratório}
        WRITE(‘Escreva a nota do trabalho em laboratório: ‘);
        {Recebe a nota do laboratório}
        READLN(nota_trab);
        {Mostra mensagem antes da leitura da nota semestral}
        WRITE(‘Escreva a nota da avaliação semestral: ‘);
        {Recebe a nota semestral}
        READLN(aval_sem);
        {Mostra mensagem antes da leitura da nota do exame}
        WRITE(‘ Escreva a nota do exame final: ‘);
        {Recebe a nota do exame}
```

```
        READLN(exame);
        {Calcula a média ponderada}
        media := (nota_trab * 2 + aval_sem * 3 + exame * 5) / 10;
        {Mostra a média calculada com formatação}
        WRITELN('Média ponderada: ',media:5:2);
        {Mostra o conceito de acordo com a m,dia}
        IF (media >=8) AND (media <=10)
                THEN WRITELN('Obteve conceito A');
        IF (media >=7) AND (media < 8)
                THEN WRITELN('Obteve conceito B');
        IF (media >= 6) AND (media < 7)
                THEN WRITELN('Obteve conceito C');
        IF (media >= 5) AND (media < 6)
                THEN WRITELN('Obteve conceito D');
        IF (media >= 0) AND (media < 5)
                THEN WRITELN('Obteve conceito E');
        {Para o programa à espera de um ENTER}
        READLN;
END.
```

Solução 2 PASCAL

```
PROGRAM EX1;
        USES CRT;
        VAR nota_trab, aval_sem, exame, media: REAL;
BEGIN
        {Limpa o ecrã}
        CLRSCR;
        {Mostra mensagem antes da leitura da nota do laboratório}
        WRITE ('Escreva a nota do trabalho em laboratório: ');
        {Recebe a nota do laboratório}
        READLN (nota_trab);
        {Mostra mensagem antes da leitura da nota semestral}
```


```
        WRITE ('Escreva a nota da avaliação semestral: ');
        {Recebe a nota semestral}
        READLN (aval_sem);
        {Mostra mensagem antes da leitura da nota do exame}
        WRITE (' Escreva a nota do exame final: ');
        {Recebe a nota do exame}
        READLN (exame);
        {Calcula a média ponderada}
        media := (nota_trab * 2 + aval_sem * 3 + exame * 5) / 10;
        {Mostra a média calculada com formatação}
        WRITELN ('Média ponderada: ',media:5:2);
        {Mostra o conceito de acordo com a m,dia}
        IF media >=8
                THEN WRITELN('Obteve conceito A')
                ELSE IF media >=7
                        THEN WRITELN ('Obteve conceito B')
                        ELSE IF media >= 6
                                THEN WRITELN ('Obteve conceito C')
                                ELSE IF media >= 5
                                        THEN WRITELN ('Obteve conceito D')
                                        ELSE WRITELN ('Obteve conceito E');
        {Para o programa à espera de um ENTER}
        READLN;
END.
```

Exercício 2

Faça um programa que receba três notas de um aluno, calcule e mostre a média aritmética e a mensagem constante na tabela a seguir. Aos alunos que ficaram para exame, calcule e mostre a nota que deverão tirar para serem aprovados, considerando que a média exigida é 6,0.

MÉDIA ARITMÉTICA			MENSAGEM
0,0	●——○	3,0	Reprovado
3,0	●——○	7,0	Exame
7,0	●——●	10,0	Aprovado

Solução Algoritmo

```
ALGORITMO
DECLARE nota1, nota2, nota3, media, nota_exame NUMÉRICO
ESCREVA “Escreva a primeira nota: “
LEIA nota1
ESCREVA “Escreva a segunda nota: “
LEIA nota2
ESCREVA “Escreva a terceira nota: “
LEIA nota3
media ← (nota1 + nota2 + nota3) / 3
ESCREVA “Media aritmética: “, media
SE media >= 0 E media < 3
        ENTÃO ESCREVA “Reprovado”
SE media >= 3 E media < 7
        ENTÃO INICIO
                ESCREVA “Exame”
                nota_exame <— 12 - media;
                ESCREVA “Deve tirar nota”, nota_exame, “para ser aprovado”
        FIM
SE media >= 7 E media <= 10
        ENTÃO ESCREVA “Aprovado”
FIM ALGORITMO
```

Solução 1 PASCAL

```
PROGRAM EX2;
        USES CRT;
        VAR nota1, nota2, nota3, media, nota_exame: REAL;
```

```
BEGIN
        {Limpa o ecrã}
        CLRSCR;
        {Mostra mensagem antes da leitura da primeira nota}
        WRITE('Escreva a primeira nota: ');
        {Recebe a primeira nota}
        READLN(nota1);
        {Mostra mensagem antes da leitura da segunda nota}
        WRITE(' Escreva a segunda nota: ');
        {Recebe a segunda nota}
        READLN(nota2);
        {Mostra mensagem antes da leitura da terceira nota}
        WRITE(' Escreva a terceira nota: ');
        {Recebe a terceira nota}
        READLN(nota3);
        {Calcula a média aritmética}
        media := (nota1 + nota2 + nota3) / 3;
        {Mostra a média aritmética}
        WRITELN('Média aritmética: ',media:5:2);
        {Mostra a situação}
        IF (media >=0) AND (media <3)
                THEN WRITELN('Reprovado');
        IF (media >=3) AND (media < 7)
                THEN BEGIN
                        WRITELN('Exame ');
                        nota_exame := 12 - media;
                        WRITELN('Deve tirar nota ',nota_exame:4:2,' para ser
                        aprovado');
        END;
        IF (media >= 7) AND (media < 10)
                THEN WRITELN('Aprovado');
        {Para o programa à espera de um ENTER}
```

```
        READLN;
END.
```

Solução 2 PASCAL

```
PROGRAM EX2;
        USES CRT;
        VAR nota1, nota2, nota3, media, nota_exame: REAL;
BEGIN
        {Limpa o ecrã}
        CLRSCR;
        {Mostra mensagem antes da leitura da primeira nota}
        WRITE('Escreva primeira nota: ');
        {Recebe a primeira nota}
        READLN(nota1);
        {Mostra mensagem antes da leitura da segunda nota}
        WRITE(' Escreva a segunda nota: ');
        {Recebe a segunda nota}
        READLN(nota2);
        {Mostra mensagem antes da leitura da terceira nota}
        WRITE(' Escreva a terceira nota: ');
        {Recebe a terceira nota}
        READLN(nota3);
        {Calcula a média aritmética}
        media := (nota1 + nota2 + nota3) / 3;
        {Mostra a média aritmética}
        WRITELN('Média aritmética: ',media:5:2);
        {Mostra a situação}
        IF (media >=0) AND (media <3)
                THEN WRITELN('Reprovado')
                ELSE IF media < 7
                        THEN BEGIN
                        WRITELN('Exame ');
```

```
                    nota_exame := 12 - media;
                    WRITELN('Deve tirar nota ',nota_exame:4:2,' para ser
                    aprovado');
        END
        ELSE WRITELN('Aprovado');
{Para o programa à espera de um ENTER}
READLN;
END.
```

Exercício 3

Faça um programa que receba dois números e mostre o maior.

Solução Algoritmo

```
ALGORITMO
DECLARE num1, num2 NUMÉRICO
ESCREVA "Escreva o primeiro número: "
LEIA num1
ESCREVA "Escreva o segundo numero: "
LEIA num2
SE num1 > num2
        ENTÃO ESCREVA "O maior número é: ", num1
SE num2 > num1
        ENTÃO ESCREVA "O maior número é: ", num2
SE num1 = num2
        ENTÃO ESCREVA "Os números são iguais "
FIM ALGORITMO.
```

Solução 1 PASCAL

```
PROGRAM EX3;
        USES CRT;
        VAR  num1, num2: REAL;
BEGIN
```

```
        {Limpa o ecrã}
        CLRSCR;
        {Mostra mensagem de leitura do primeiro número}
        WRITE('Escreva o primeiro número: ');
        {Recebe o primeiro número}
        READLN(num1);
        {Mostra mensagem de leitura do segundo número}
        WRITE(' Escreva o segundo número: ');
        {Recebe o segundo número}
        READLN(num2);
        {Verifica qual o maior número}
        IF num1 > num2
                THEN WRITELN('O maior número : ',num1:5:2);
        IF num2 > num1
                THEN WRITELN('O maior número : ',num2:5:2);
        IF num1 = num2
                THEN WRITELN('Os números são iguais');
        {Para o programa à espera de um ENTER}
        READLN;
END.
```

Solução 2 PASCAL

```
PROGRAM EX3;
        USES CRT;
        VAR  num1, num2: REAL;
BEGIN
        {Limpa o ecrã}
        CLRSCR;
        {Mostra mensagem de leitura do primeiro número}
        WRITE('Escreva o primeiro número: ');
        {Recebe o primeiro número}
        READLN(num1);
```


```
        {Mostra mensagem de leitura do segundo número}
        WRITE('Escreva o segundo número: ');
        {Recebe o segundo número}
        READLN(num2);
        {Verifica qual o maior número}
        IF num1 > num2
                THEN WRITELN('O maior número : ',num1:5:2)
                ELSE IF num2 > num1
                        THEN WRITELN('O maior numero : ',num2:5:2)
                        ELSE WRITELN('Os números são iguais');
        {Para o programa à espera de um ENTER}
        READLN;
END
```

Estrutura de Repetição

Estrutura de repetição em algoritmos

Uma estrutura de repetição é utilizada quando uma "fração" do algoritmo ou até mesmo o algoritmo inteiro precisa de ser repetido. O numero de repetições pode ser fixo ou estar atrelado a uma condição. Assim, existem estruturas para tais situações, descritas a seguir.

Estrutura de repetição para número definido de repetições (estrutura PARA)

Essa estrutura de repetição é utilizada quando se sabe o número de vezes que uma "fração" do algoritmo deve ser repetida. O formato geral dessa estrutura é:

```
PARA I ← valor inicial ATÉ valor final FAÇA [PASSO n]
INICIO
Comando1
comando2
…
comando_m
FIM
```

O comando1, o comando2 e o comando_m serão executados utilizando-se a variável i como controlo, e o seu conteúdo vai variar do valor inicial até o valor final. A informação do passo está entre parenteses retos porque é opcional. O passo indica como será a variação da variável de controlo. Por exemplo, quando for indicado PASSO 2, a variável de controlo será aumentada em 2 unidades a cada iteração até atingir o valor final.

Quando a informação do passo for suprimida, isso significa que o incremento ou o decremento da variável de controlo será de 1 unidade.

Quando houver apenas um comando a ser repetido, os marcadores de bloco INICIO e FIM poderão ser suprimidos.

Exemplos:

PARA I ← 1 ATE 10 FAÇA

ESCREVA I

O comando ESCREVA I será executado dez vezes, ou seja, para I a variar de 1 a 10. Assim, os valores de I serão: 1, 2, 3, 4, 5, 6, 7, 8, 9 e 10.

PARA J ← 1 ATE 9 FAÇA PASSO 2

ESCREVA J

O comando ESCREVA J será executado cinco vezes, ou seja, para J a variar de 1 a 10, de 2 em 2. Assim, os valores de J serão: 1, 3, 5, 7 e 9.

PARA I ← 10 ATE 5 FAÇA

ESCREVA I

O comando ESCREVA I será executado seis vezes, ou seja, para I a variar de 10 a 5. Assim, os valores de I serão: 10, 9, 8, 7, 6 e 5.

PARA J ← 15 ATE 1 FAÇA PASSO -2

ESCREVA J

O comando ESCREVA J será executado oito vezes, ou seja, para J a variar de 15 a 1, de 2 em 2. Assim, os valores de J serão: 15, 13, 11, 9, 7, 5, 3 e 1.

Estrutura de repetição para número indefinido de repetições e teste no início (ESTRUTURA ENQUANTO)

Esta estrutura de repetição é utilizada quando não se sabe o número de vezes que uma fração do algoritmo deve ser repetida, embora também possa ser utilizada quando se conhece esse número.

Esta estrutura baseia-se na análise de uma condição. A repetição será feita enquanto a condição se mostrar verdadeira.

Existem situações em que o teste condicional da estrutura de repetição, que fica no início, resulta num valor falso logo na primeira comparação. Nesses casos, os comandos de dentro da estrutura de repetição não serão executados.

```
ENQUANTO condição FAÇA
comando1
```

Enquanto a condição for verdadeira, o comando1 será executado.

```
ENQUANTO condição FAÇA
INICIO
        comando1
        comando2
        comando3
FIM
```

Enquanto a condição for verdadeira, o comando1, o comando2 e o comando3 serão executados.

Exemplos:

```
x ← 1
Y ← 5
ENQUANTO X < Y FAÇA
INICIO
X ←  X + 2
Y ← Y + 1
FIM
```

Simulação:

X	Y	
1	5	Valores iniciais
3	6	Valores obtidos dentro da estrutura de repetição
5	7	
7	8	
9	9	

Na fração do algoritmo anterior, portanto, os comandos que estão dentro da estrutura de repetição serão repetidos quatro vezes.

```
x ← 1
y ← 1
ENQUANTO X <= 5 FAÇA
INICIO
y ← y * x
x ← x + 1
FIM
```

Simulação:

Y	X	
1	1	Valores iniciais
1	2	Valores obtidos dentro da estrutura de repetição
2	3	
6	4	
24	5	
120	6	

Na fração do algoritmo anterior, portanto, os comandos que se localizam na estrutura de repetição serão repetidos cinco vezes. Nesse exemplo, a estrutura ENQUANTO é utilizada para repetir a fração do algoritmo um número definido de vezes.

Estrutura de repetição para número indefinido de repetições e teste no final (ESTRUTURA REPITA)

Essa estrutura de repetição e utilizada quando não se sabe o numero de vezes que uma fração do algoritmo deve ser repetido, embora também possa ser utilizada quando se conhece esse número.

Essa estrutura baseia-se na análise de uma condição. A repetição será feita até a condição se tornar verdadeira.

A diferença entre a estrutura ENQUANTO e a estrutura REPITA é que nesta última os

comandos serão repetidos pelo menos uma vez, já que a condição de paragem se encontra no final.

REPITA

Comandos

ATE condição

Repita os comandos até a condição se tornar verdadeira.

Exemplos:

x ← 1

Y ← 5

REPITA

x ← x + 2

y ← y + 1

ATE x >= y

Simulação:

X	Y	
1	5	Valores iniciais
3	6	Valores obtidos dentro da estrutura de repetição
5	7	
7	8	
9	9	

No trecho do algoritmo anterior, portanto, os comandos de dentro da estrutura de repetição serão repetidos quatro vezes.

x ← 1

Y ← 1

REPITA

y ← y * x

x ← x + 1

ATE x = 6

Simulação:

Y	X	
1	1	Valores iniciais
1	2	
2	3	
6	4	Valores obtidos dentro da estrutura de repetição
24	5	
120	6	

Na fração do algoritmo anterior, portanto, os comandos que se localizam dentro da estrutura de repetição serão repetidos cinco vezes. Nesse exemplo, a estrutura REPITA é utilizada para repetir a fração do algoritmo um número definido de vezes.

Exercícios Resolvidos

Exercício 1

Um funcionário de uma empresa recebe aumento salarial anualmente. Sabe-se que:

a. Esse funcionário foi contratado em 2005, com salario inicial de $ 1.000,00.
b. Em 2006, ele recebeu aumento de 1,5% sobre seu salario inicial.
c. A partir de 2007 (inclusive), os aumentos salariais sempre corresponderam ao dobro do percentual do ano anterior.

Faça um programa que determine o salario atual desse funcionário.

Solução Algoritmo

```
ALGORITMO
    DECLARE     i, ano_atual, salario NUMERICO
                novo_salario/ percentual NUMERICO
    LEIA ano_atual
    salario ← 1000
    percentual ← 1,5/100
```

```
        novo_salario ← salario + percentual * salario
        PARA i ← 2007 ATÉ ano_atual FAÇA
        INICIO
                percentual ← 2 * percentual
                novo_salario ← novo_salario + percentual * novo_salario
        FIM
        ESCREVA novo_salario
FIM ALGORITMO.
```

Solução 1 PASCAL

```
PROGRAM EX1;
        USES CRT;
        VAR     i, ano_atual: INTEGER;
                salario, novo_salario, percentual: REAL;
BEGIN
        CLRSCR;
        WRITELN('Escreva o ano atual');
        READLN(ano_atual);
        salario := 1000;
        percentual := 1.5/100;
        novo_salario := salario + percentual * salario;
        FOR i := 2007 TO ano_atual DO
                BEGIN
                        percentual := 2 * percentual;
                        novo_salario := novo_salario + percentual * novo_salario;
                END;
        WRITELN('Novo sal rio = ',novo_salario:5:2);
        READLN;
END.
```


Solução 2 PASCAL

```
PROGRAM EX1;
        USES CRT;
        VAR     i, ano_atual: INTEGER;
                salario, novo_salario, percentual: REAL;
BEGIN
        CLRSCR;
        WRITELN('Escreva o ano atual');
        READLN(ano_atual);
        salario := 1000;
        percentual := 1.5/100;
        novo_salario := salario + percentual * salario;
        i := 2007;
        WHILE i <= ano_atual DO
                BEGIN
                        percentual := 2 * percentual;
                        novo_salario := novo_salario + percentual * novo_salario;
                        i := i + 1;
                END;
        WRITELN('Novo sal rio = ',novo_salario:5:2);
        READLN;
END.
```

Exercício 2

Faça um programa que leia um valor N inteiro e positivo, calcule e mostre o valor de E, conforme a fórmula a seguir:

$$E = 1 + 1/1! + 1/2! + 1/3! + \ldots + 1/N!$$

Solução Algoritmo

```
ALGORITMO
        DECLARE n, e, i, j, fat NUMERICO
        LEIA n
        e ← 1
        PARA i ← 1 ATÉ n FAÇA
        INICIO
                fat ← 1
                PARA j ← 1 ATÉ i FAÇA
                        INICIO
                                fat ← fat * j
                        FIM
                e ← e + 1/fat
        FIM
ESCREVA e
FIM_ALG0RITMO.
```

Solução 1 PASCAL

```
PROGRAM EX2;
        USES CRT;
        VAR n, i, j: INTEGER;
        e, fat: real;
BEGIN
        CLRSCR;
        WRITELN('Escreva o valor de N');
        READLN(n);
        e := 1;
        FOR i := 1 TO n DO
                BEGIN
                        fat := 1;
                        FOR j := 1 TO i DO
                                BEGIN
```


```
                                        fat := fat * j;
                                    END;
                                e := e + 1/fat;
                        END;
                WRITELN('Valor de E = ',e:5:2);
                READLN;
END.
```

Solução 2 PASCAL

```
PROGRAM EX2;
        USES CRT;
        VAR n, i, j: INTEGER;
        e, fat:REAL;
BEGIN
        CLRSCR;
        WRITELN('Escreva o valor de N');
        READLN(n);
        e := 1;
        i := 1;
        REPEAT
                j := 1;
                fat := 1;
                        REPEAT
                                fat := fat * j;
                                j := j + 1;
                        UNTIL j > i;
                                i := i + 1;
                                e := e + 1/fat;
        UNTIL i > n;
        WRITELN('Valor de E = ',e:5:2);
        READLN;
END.
```

Exercício 3

Faça um programa que leia um número N e que indique quantos valores inteiros e positivos devem ser lidos a seguir. Para cada número lido, mostre uma tabela contendo o valor lido e o fatorial desse valor.

Solução Algoritmo

```
ALGORITMO
        DECLARE n, num, i, j, fat NUMERICO
        LEIA n
        PARA i ← 1 ATÉ n FAÇA
                INICIO
                        LEIA num
                        fat ← 1
                        PARA j ← 1 ATÉ num FAÇA
                                INICIO
                                        Fat ← fat * j
                                FIM
                        ESCREVA fat
                FIM
FIM_ALG0RITM0.
```

Solução 1 PASCAL

```
ROGRAM EX3;
        USES CRT;
        VAR n, num, i, j:INTEGER;
        fat: REAL;
BEGIN
        CLRSCR;
        WRITELN('Escreva a quantidade de números que serão lidos');
        READLN(n);
        FOR i := 1 TO n DO
                BEGIN
```

```
                    WRITELN;
                    WRITELN('Digite o ', i,'º número');
                    READLN(num);
                    fat := 1;
                        FOR j := 1 TO num DO
                            BEGIN
                                fat := fat * j;
                            END;
                        WRITELN('Fatorial de ',num,' = ',fat:5:2);
                END;
        READLN;
END.
```

Solução 2 PASCAL

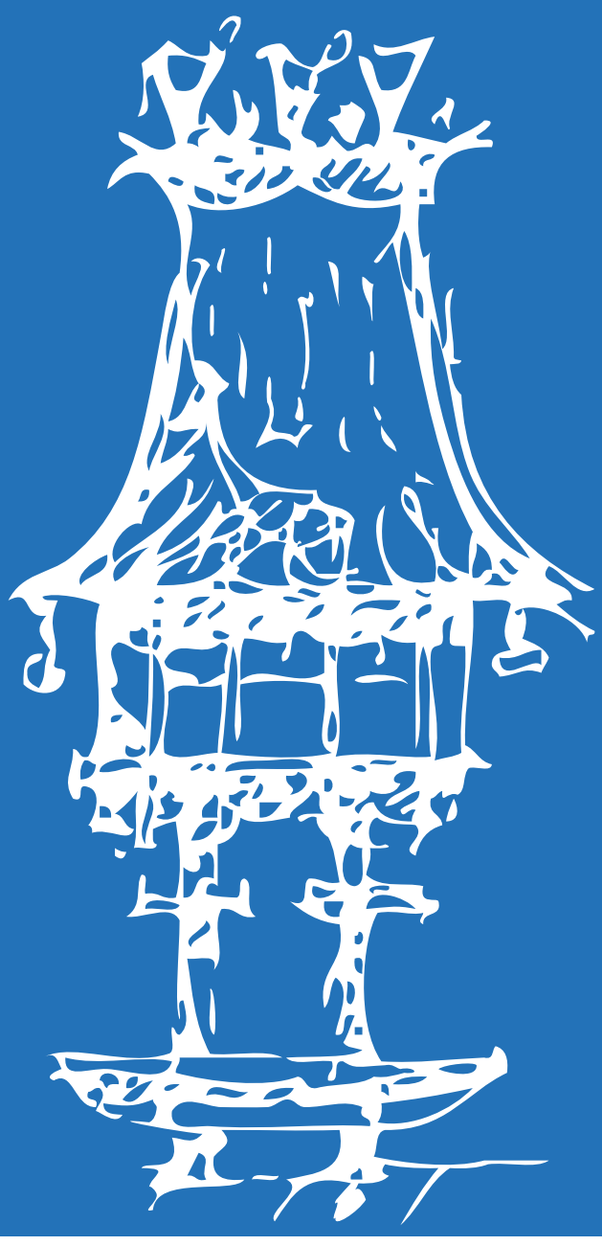
```
ROGRAM EX3;
        USES CRT;
        VAR n, num, i, j:INTEGER;
        fat: REAL;
BEGIN
        CLRSCR;
        WRITELN('Escreva a quantidade de números que serão lidos');
        READLN(n);
        i := 1;
        WHILE i <= n DO
                BEGIN
                    WRITELN;
                    WRITELN('Escreva o ', i,'º número');
                    READLN(num);
                    fat := 1;
                    j := 1;
                    WHILE j <= num DO
                        BEGIN
```

```
                    fat := fat * j;
                    j := j +1;
                END;
            WRITELN('Fatorial de ',num,' = ',fat:5:2);
        i := i + 1;
        END;
    READLN;
END.
```


Bibliografia

GONÇALVES, Victor, Sistemas Electrónicos com Microcontroladores. ETEP. (s.d.).

Notas



Microcontroladores

Módulo 7

Apresentação

Este módulo tem carácter teórico-prático, por isso deverá decorrer em parte em ambiente laboratorial de forma a permitir aos alunos verificarem e comprovarem os conhecimentos teóricos adquiridos sobre a estrutura e operação de um microcontrolador.

Esta disciplina tem como intenção tornar o aluno apto a compreender a linguagem e as técnicas utilizadas, possibilitando assim um melhor aproveitamento na sequência dos estudos desta e das outras disciplinas técnicas e também na comunicação adequada com os profissionais da área.

Introdução

A abordagem deste módulo sobre microcontroladores leva-nos a uma melhor compreensão do funcionamento de vários tipos de aparelhos, que incorporam circuitos que utilizam estas características, existentes no mercado assim como a melhor escolha deste tipo de equipamentos para que se ajuste às crescentes evoluções disponíveis pelas diversas marcas.

Este módulo requer um conhecimento básico de matemática e programação.

Objetivos de aprendizagem

- Descrever os blocos constituintes do microcontrolador e sua interligação.
- Identificar os registos de usos gerais e especiais.
- Caracterizar as memórias internas e externas.
- Compreender o modo de funcionamento das portas de entrada e saída de dados.
- Identificar os modos de endereço usados nas instruções do microcontrolador.
- Conhecer os diferentes grupos de instruções do microcontrolador.
- Construir programas que utilizem as instruções de transferência e processamento de dados, assim como as de teste e salto.
- Descrever os diferentes modos de funcionamento dos contadores/temporizadores.
- Compreender o funcionamento das interrupções no microcontrolador.

Âmbito de conteúdos

- Microcontrolador da família do 8051:
  - Estrutura interna.
  - Memória de programa e dados.
  - A unidade lógica e aritmética.
  - Registos de funções especiais.
  - Modos de endereçamento.
  - Tipos de instruções.
  - Controlo de interrupções.
  - Temporização e contagem.

Microcontrolador

Introdução

Atualmente um grande número de microcontroladores, integrados em diversos equipamentos, exerce um papel importante no dia-a-dia das pessoas. Despertar ao som de um CD Player programável, tomar café da manhã preparado por um micro-ondas digital e ir ao trabalho de carro, cuja injeção de combustível é microcontrolada, são apenas alguns exemplos.

O mercado de microcontroladores apresenta-se em franca expansão, ampliando seu alcance principalmente em aplicações residenciais, industriais e de telecomunicações. Segundo dados da National Semiconductor (1997), estima-se que, em 2010, em média uma pessoa interagirá com 250 dispositivos com microcontroladores diariamente.

Num passado recente, o alto custo dos dispositivos eletrônicos limitou o uso dos microcontroladores apenas aos produtos domésticos considerados de alta tecnologia (televisão, vídeo e som). Porém, com a constante queda nos preços dos circuitos integrados, os microcontroladores passaram a ser utilizados em produtos menos sofisticados do ponto de vista da tecnologia, como máquinas de lavar, micro-ondas, fogões e refrigeradores. Assim, a introdução do microcontrolador nestes produtos cria uma diferenciação e permite a inclusão de melhorias de segurança e de funcionalidade. Alguns mercados chegaram ao ponto de tornar obrigatório o uso de microcontroladores em determinados tipos de equipamentos, impondo um pré-requisito tecnológico.

Muitos produtos que temos disponíveis hoje em dia, simplesmente não existiriam, ou não teriam as mesmas funcionalidades sem um microcontrolador. É o caso, por exemplo, de vários instrumentos biomédicos, instrumentos de navegação por satélites, detetores de radar, equipamentos de áudio e vídeo, eletrodomésticos, entre outros.

Entretanto, o alcance dos microcontroladores vai muito além de oferecer algumas facilidades. Uma aplicação crucial, onde os microcontroladores são utilizados, é na redução de consumo de recursos naturais. Existem sistemas de aquecimento modernos que captam a luz solar e de acordo com a demanda dos utilizadores, controlam a temperatura de forma a minimizar perdas. Um outro exemplo, de maior impacto, é o uso de microcontroladores na redução do consumo de energia em motores elétricos,

que são responsáveis pelo consumo de, aproximadamente, 50% de toda a eletricidade produzida no planeta. Portanto, o alcance dessa tecnologia tem influência muito mais importante nas nossas vidas, do que se possa imaginar.

O universo de aplicações dos microcontroladores, como já mencionado, está em grande expansão, sendo que a maior parcela dessas aplicações é em sistemas embarcados. A expressão "sistema embarcado" (do inglês embedded system) refere-se ao fato do microcontrolador ser inserido nas aplicações (produtos) e usado de forma exclusiva por elas. Como a complexidade desses sistemas cresce vertiginosamente, o software tem sido fundamental para oferecer as respostas às necessidades desse mercado. Tanto é que o software para microcontroladores representa uma fatia considerável do mercado de software mundial. Segundo Edward Yourdon (consultor na área de computação, pioneiro nas metodologias de engenharia do software e programação estruturada) a proliferação dos sistemas embarcados, juntamente com o advento da Microsoft, são os responsáveis pela retoma do crescimento da indústria de software nos Estados Unidos da América.

Definição de Microcontrolador

Introdução

Um microcontrolador é um sistema computacional completo, no qual estão incluídos uma CPU (Central Processor Unit), memória de dados e programa, um sistema de clock, portas de I/O (Input/Output), além de outros possíveis periféricos, tais como, módulos de temporização e conversores A/D entre outros, integrados num mesmo encapsulamento. As partes integrantes de qualquer computador, e que também estão presentes, em menor escala, nos microcontroladores são:

- Unidade Central de Processamento (CPU);
- Sistema de clock para dar sequência às atividades da CPU;
- Memória para armazenamento de instruções e para manipulação de dados;
- Entradas para interiorizar na CPU informações do mundo externo;
- Saídas para exteriorizar informações processadas pela CPU para o mundo externo;
- Programa (firmware) para definir um objetivo ao sistema.

Unidade Central de Processamento (CPU)

A unidade central de processamento é composta por uma unidade lógica aritmética (ULA), por uma unidade de controlo e por unidades de memória especiais conhecidas por registradores. Para que a CPU possa realizar tarefas é necessário que se agregue outros componentes, como unidades de memória, unidades de entrada e unidades de saída. A figura a seguir apresenta um diagrama de blocos com uma possível interface entre a CPU e outros dispositivos.

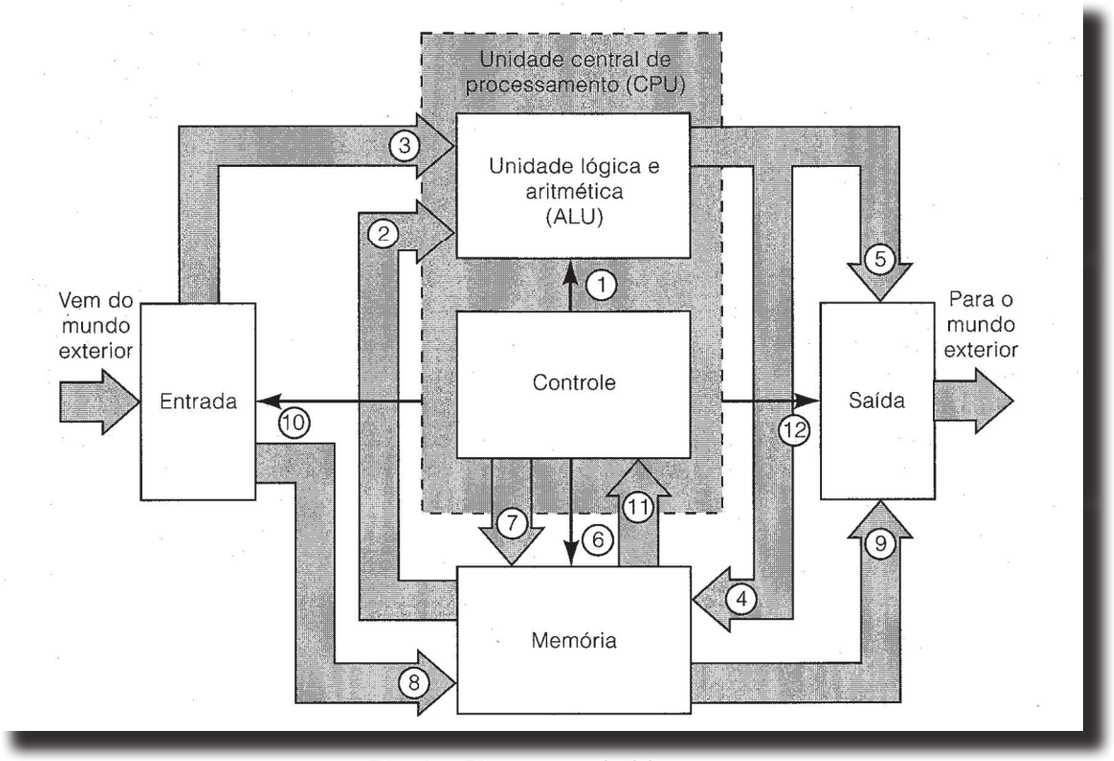


Fig. 1 – Diagrama de blocos.

A unidade de memória permite armazenar grupos de dígitos binários que podem representar instruções que o processador irá executar ou dados que serão manipulados pelo processador. A unidade de entrada consiste em todos os dispositivos utilizados para obter informações e dados externos ao processador. A unidade de saída consiste em dispositivos capazes de transferir dados e informações do processador para o exterior.

A ULA é a área de uma CPU na qual as operações lógicas e aritméticas são realizadas sobre os dados. O tipo de operação realizada é determinado pelos sinais da unidade de controlo. Os dados a serem operados pela ULA podem ser oriundos de uma memória ou de uma unidade de entrada. Os resultados das operações realizadas na ULA podem ser transferidos tanto para uma memória de dados como para uma unidade de saída.

A função da unidade de controlo é comandar as operações da ULA e de todas as outras unidades conectadas a CPU, fornecendo sinais de controlo e temporização. De certa maneira, a unidade de controlo é como um maestro que é responsável por manter cada um dos membros da orquestra em sincronismo. Essa unidade contém circuitos lógicos e de temporização que geram os sinais apropriados necessários para executar cada instrução de um programa.

A unidade de controlo procura uma instrução na memória enviando um endereço e um comando de leitura para a unidade de memória. A palavra da instrução armazenada na posição de memória é transferida para um registrador conhecido por registrador de instruções (RI) da unidade de controlo. Essa palavra de instrução, que está de alguma forma de código binário, é então decodificada pelos circuitos lógicos na unidade de controlo para determinar a instrução que está a ser invocada. A unidade de controlo usa essa informação para enviar os sinais apropriados para as unidades restantes a fim de executar a operação específica.

Essa sequência de procura de um código de instrução e de execução da operação indicada é repetida indefinidamente pela unidade de controlo. Essa sequência repetitiva de procura/execução continua até que a CPU seja desligada ou até que o RESET seja ativado. O RESET faz sempre a CPU procurar a sua primeira instrução no programa.
Uma CPU, também é conhecida por processador, repete indefinidamente as mesmas operações básicas de procura e execução. Naturalmente, os diversos ciclos de execução serão diferentes para cada tipo de instrução à medida que a unidade de controlo envia sinais diferentes para as outras unidades de execução de uma instrução em particular.
Um registrador é um tipo de memória de pequena capacidade porém muito rápida, contida na CPU, utilizado no armazenamento temporário de dados durante o processamento. Os registradores estão no topo da hierarquia de memória, sendo desta forma o meio mais rápido e de maior custo para armazenar um dado.

Cada registrador de um processador possui uma função especial. Um dos mais importantes é o contador de programa (program counter - PC), que armazena os endereços dos códigos das instruções à medida que são procurados na memória. Outros registradores são utilizados para realizar funções como: armazenamento de códigos de instrução (RI), manutenção dos dados operados pela ULA (acumulador), armazenamento de endereços de dados a serem lidos na memória (ponteiro de dados), além de outras funções de armazenamento e contagem. Todos os processadores possuem um registrador especial muito utilizado chamado acumulador ou registrador A. Ele armazena um operando para quaisquer instruções, lógica ou matemática. O resultado da operação é armazenado no acumulador após a instrução ser executada.

Para que exista comunicação entre as unidades que formam um processador devemos definir uma forma de conexão entre estas unidades. Num processador tradicional com arquitetura Von Neuman este meio é o barramento de dados. A largura do barramento de dados em bits é o que determina o número de bits para um dado processador. A figura a seguir apresenta a interface dos principais dispositivos que compõem um sistema microprocessado através de um barramento de dados.

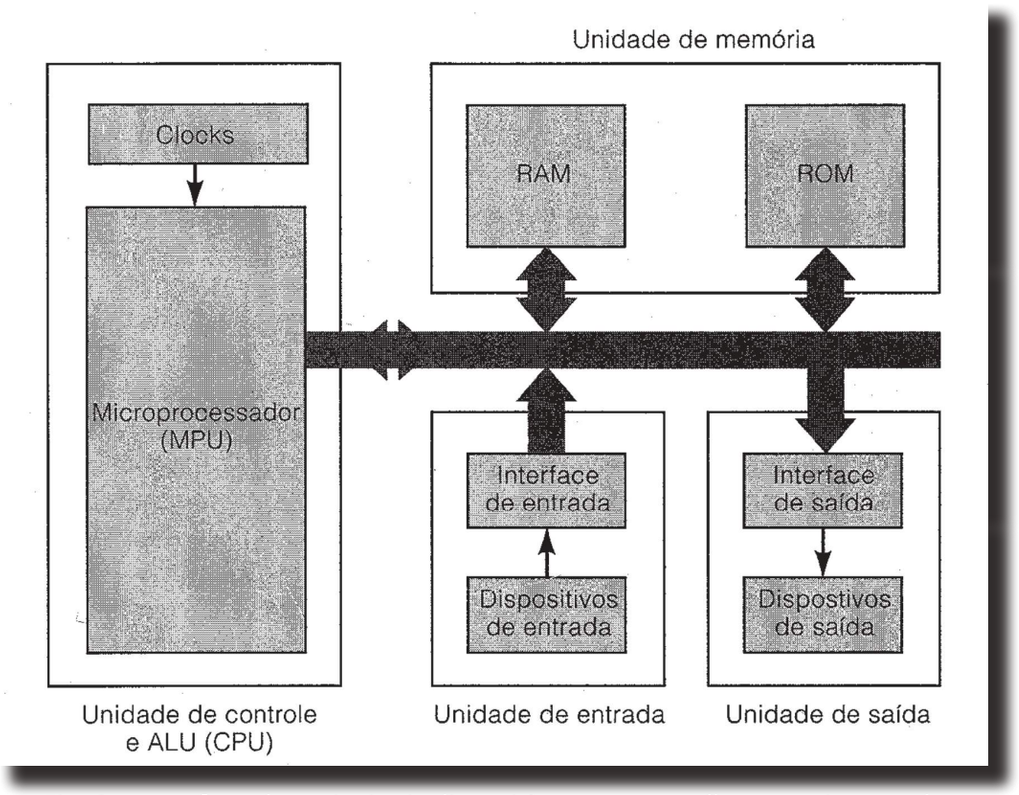


Fig. 2 – Interface dos principais dispositivos que compõem um sistema de microprocessador.

O CPU é o centro de todo sistema computacional e não é diferente quando se trata de microcontroladores. O trabalho da CPU é executar rigorosamente as instruções de um programa, na sequência programada, para uma aplicação específica. Um programa computacional (software) instrui a CPU a ler informações de entradas, ler e escrever informações na memória de dados, e escrever informações nas saídas. O diagrama de blocos simplificado da CPU está presente nos microcontroladores da família HC08, também denominado de CPU08, é apresentado na figura a seguir. Esta arquitetura de processador será utilizada como modelo neste documento.

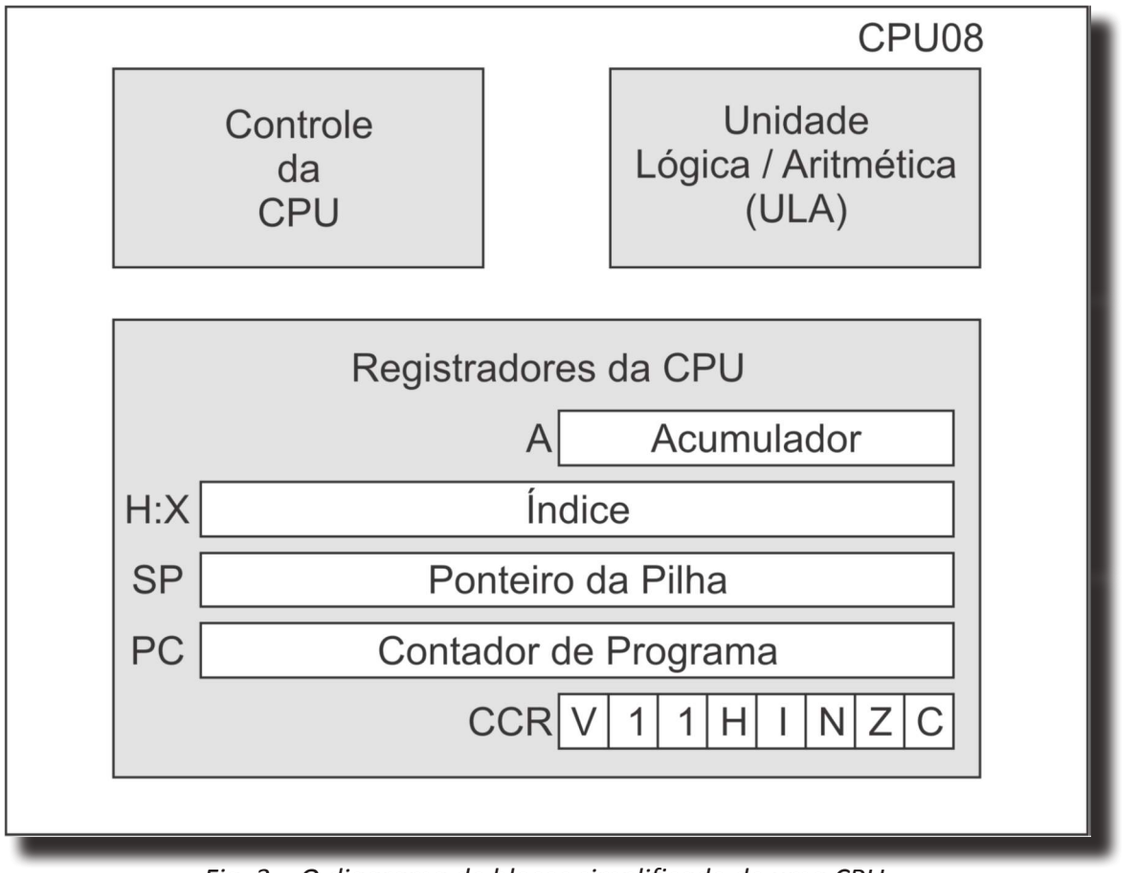


Fig. 3 – O diagrama de blocos simplificado de uma CPU.

As principais funções de cada um dos componentes do CPU08 serão apresentadas a seguir.

Unidade Lógica/Aritmética (ULA): A ULA é utilizada para realizar operações lógicas e aritméticas definidas no conjunto de instruções da CPU. Vários circuitos implementam as operações aritméticas binárias descodificadas pelas instruções e fornecem dados para a execução da operação na ULA. A maioria das operações aritméticas binárias é baseada em algoritmos de adição e subtração (adição com o valor negativo). A multiplicação é realizada através de uma série de adições e deslocamentos com a ULA sob controlo lógico da CPU.

Controle da CPU: O circuito de controlo da CPU implementa o sequenciamento de elementos lógicos necessários para a ULA realizar as operações requisitadas durante a execução do programa. O elemento central da seção de controlo da CPU é o descodificador de instruções. Cada opcode (código de instrução) é descodificado para determinar quantos operandos são necessários e qual a sequência de passos será necessária para completar a instrução em curso. Quando uma instrução é executada completamente, o próximo opcode é lido e descodificado.

Registradores da CPU: A CPU08 contém 5 registradores como apresentado na figura anterior. Os registradores da CPU são memórias especiais que não fazem parte do mapa de memória. O conjunto de registradores da CPU é frequentemente chamado de modelo de programação.

O acumulador, também chamado de registrador A, é frequentemente utilizado para armazenar um dos operandos ou o resultado de operações.

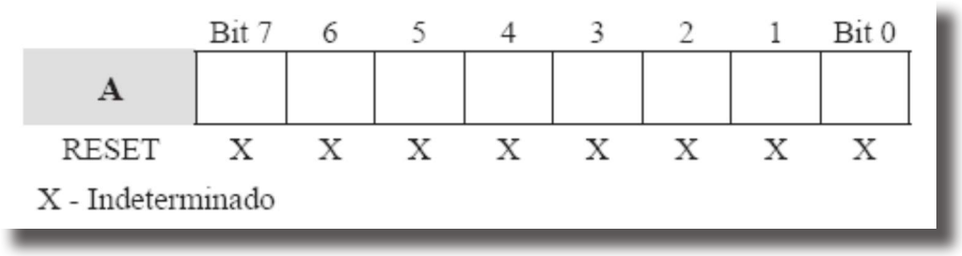


Fig. 4 – Representação do acumulador.

O registrador H:X é um registrador de 16 bits de índice que possibilita ao usuário endereçar indiretamente o espaço de memória de 64Kbytes. O byte mais significativo do registrador de índice é denominado H, e o byte menos significativo denominado X. A principal função é servir de apontador para uma área na memória onde a CPU irá carregar (ler) ou armazenar (escrever) informação. Quando não estiver a ser utilizado para apontar um endereço na memória, ele pode ser utilizado como registrador genérico.

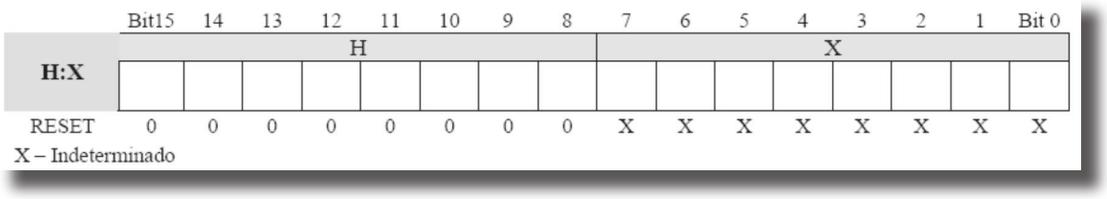


Fig. 5 – Representação do registo.

O registrador Program Counter (PC) é usado pelo CPU para controlar e conduzir ordenadamente a procura do endereço da próxima instrução a ser executada. Quando a CPU é energizado ou passa por um processo de reset, o PC é carregado com o conteúdo de um par de endereços específicos denominados vetor de reset (reset vector). O vetor de reset contém o endereço da primeira instrução a ser executada pela CPU. Assim que as instruções são executadas a CPU incrementa o PC, de tal forma que ele sempre aponte para a próxima informação que a CPU vai precisar. O número de bits do PC coincide

exatamente com o número de linhas do barramento de endereços, que por sua vez determina o espaço total disponível de memória que pode ser utilizado pela CPU.

O registrador Condition Code (CCR) é um registrador de 8 bits que armazena os bits de estado (flags) que refletem o resultado de algumas operações da CPU. As instruções de desvio usam estes bits de estado para tomar as suas decisões.

	Bit 7	6	5	4	3	2	1	Bit 0
CCR	V	1	1	H	I	N	Z	C
RESET	X	1	1	X	1	X	X	X

X – Indeterminado

Fig. 6 – Representação do registo.

A descrição dos bits do registrador de condição é apresentada a seguir:

V (Bit de Overflow) - O CPU leva o bit de overflow para nível lógico alto quando houver estouro no resultado de uma operação em complemento de 2. O bit V é utilizado pelas instruções de desvios condicionais BGT, BGE, BLE, e BLT.

H (Bit de Half-carry) - O CPU leva o bit de half-carry para nível lógico alto quando ocorrer estouro entre os bits 3 e 4 do acumulador durante as operações ADD e ADC. O bit H é importante nas operações aritméticas codificadas em binário (BCD). A instrução DAA utiliza o estado dos bits H e C para determinar o fator de correção apropriado.

I (Máscara de Interrupções) - Quando o bit I está no nível lógico alto, todas as interrupções são mascaradas (desabilitadas). As interrupções são habitadas quando o bit I é levado a nível lógico baixo. Quando ocorre uma interrupção, o bit que mascara as interrupções é automaticamente levado a nível lógico alto. Depois que os registradores da CPU são armazenados na pilha este bit volta ao nível lógico baixo. Se uma interrupção ocorrer enquanto o bit I estiver no nível lógico alto, o seu estado será guardado. As interrupções são atendidas, em ordem de prioridade, assim que o bit I for a nível lógico 0. A instrução retorno da interrupção (RTI) retorna os registradores da CPU da pilha e restaura o bit I no seu estado de nível lógico 0. Após qualquer reset, o bit I é colocado em nível lógico alto e só pode ser limpo por uma instrução de software (CLI).

N (Bit Negativo) - O CPU coloca o bit N em nível lógico alto quando uma operação aritmética, lógica ou de manipulação de dados produzir um resultado negativo. Corresponde ao 8º bit do registrador que contém o resultado.

Z (Bit Zero) - O CPU leva o bit Z para nível lógico alto quando uma operação aritmética, lógica ou de manipulação de dados produzir um resultado igual a 0.

C (Bit Carry/Borrow) - O CPU coloca o bit C no nível lógico alto quando uma operação de adição produzir um valor superior a 8 bits ou quando uma subtração necessitar de um empréstimo. Algumas operações lógicas e as instruções de manipulação de dados também podem modificar o estado do bit C.

O Stack Pointer (SP) é um registrador cuja função é apontar para a próxima localização disponível (endereço livre) de uma pilha (lista de endereços contíguos). A pilha pode ser vista como um monte de cartas empilhadas, onde cada carta armazena um byte de informação. A qualquer hora, a CPU pode colocar uma carta nova no topo da pilha ou retirar uma carta do topo da pilha. As cartas que estão no meio da pilha não podem ser retiradas até que todas que estejam acima dela sejam removidas primeiro. A CPU acompanha o efeito da pilha através do valor armazenado no SP. O SP aponta sempre para a localização de memória disponível para se colocar a próxima carta (byte).

	Bit15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
SP																
RESET	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Fig. 7 – Representação do registo.

Normalmente, a CPU usa a pilha para guardar os endereços de retorno e o contexto, isto é, os registradores do CPU, na ocorrência de uma exceção (interrupção ou reset).

Durante um reset, o Stack Pointer contém o endereço 0x00FF. A instrução RSP (Reset Stack Pointer) carrega o byte menos significativo com 0xFF e o byte mais significativo não é afetado.

Quando a CPU insere um novo dado na pilha, automaticamente o SP é decrementado para o próximo endereço livre. Quando a CPU retira um dado da pilha, o SP é incrementado para apontar para o dado mais recente e o valor do dado é lido nesta posição. Quando a CPU é energizada ou passa por um processo de reset, o SP aponta para um endereço específico na memória RAM (no caso dos microcontroladores HC08 e HCS08 = 0x00FF).

A CPU08 possui modos de endereçamento indexado com offsets de 8 ou 16 bits do SP para acesso de variáveis temporárias inseridas na pilha. A CPU utiliza o conteúdo do registrador SP para determinar o endereço efetivo do operando.

OBS: Embora o endereço inicial do SP seja 0x00FF, a localização da pilha é arbitrária e pode ser realocada pelo usuário em qualquer lugar na RAM. Movimentar o SP para fora da página de acesso direto (0x0000 a 0x00FF) permitirá que este espaço de memória seja utilizado para modos de endereçamento mais eficientes.

Sistema de Clock

Todo sistema computacional utiliza um clock para fornecer a CPU uma maneira de se mover de instrução em instrução, numa sequência pré-determinada.

Uma fonte de clock de alta frequência (normalmente derivada de um cristal ressonador conectado ao CPU) é utilizada para controlar o sequenciamento das instruções da CPU. Normalmente as CPUs dividem a frequência básica do cristal por 2 ou mais para chegar ao clock do barramento interno. Cada ciclo de leitura ou escrita da memória é executado num ciclo de clock do barramento interno, também denominado ciclo de barramento (bus cycle).

Memória

Podemos pensar na memória como sendo uma lista de endereços postais, onde o conteúdo de cada endereço é um valor fixo de 8 bits (para CPU de 8 bits). Se um sistema computacional tem n linhas (bits) de endereços, ele pode endereçar 2n posições de memória (por ex.: um sistema com 14 linhas pode ter acesso a 214 = 16384 endereços).

Entre os diversos tipos de memória encontram-se:

RAM (Random Access Memory) - Memória de acesso aleatório. Pode ser lida ou escrita pela execução de instruções da CPU e normalmente é utilizada para manipulação de dados pela CPU. O conteúdo é perdido na ausência de energia (memória volátil).

ROM (Read Only Memory) - Memória apenas de leitura. Pode ser lida, mas não é alterável. O conteúdo deve ser determinado antes que o circuito integrado seja fabricado. O conteúdo é mantido na ausência de energia (memória não volátil).

EPROM (Erasable and Programmable Read-Only Memory) – Memória ROM programável e apagável. O conteúdo dessa memória pode ser apagado com luz ultravioleta e posteriormente, reprogramado com novos valores. As operações de apagamento e

programação podem ser realizadas num número limitado de vezes depois que o circuito integrado ser fabricado. Da mesma forma que a ROM, o conteúdo é mantido na ausência de energia (memória não volátil).

OTP (One Time Programmable) - Memória programável uma única vez. Semelhante à EPROM quanto à programação, mas que não pode ser apagada.

EEPROM (Electrically Erasable and Programmable Read-Only Memory) - Memória ROM programável e apagável eletricamente. Pode ter seu conteúdo alterado através da utilização de sinais elétricos convenientes. Tipicamente, um endereço de uma EEPROM pode ser apagada e reprogramada até 100.000 vezes.

FLASH - Memória funcionalmente semelhante a EEPROM, porém com ciclos de escrita bem mais rápida.

I/O (Input/Output) - Registradores de controlo, estado e sinais de I/O são um tipo especial de memória porque a informação pode ser sentida (lida) e/ou alterada (escrita) por dispositivos diferentes do CPU.

Sinais de Entrada

Dispositivos de entrada que fornecem informação para a CPU processar, vindo do mundo externo. As maiorias das entradas que os microcontroladores processam são denominadas sinais de entrada digitais e utilizam níveis de tensão compatíveis com a fonte de alimentação do sistema. O sinal de 0V (GND ou VSS) indica o nível lógico 0 e o sinal de fonte positiva, que tipicamente é +5VDC (VDD) indica o nível lógico 1 (atualmente os microcontroladores começaram a reduzir a tensão de VDD para valores na faixa dos 3V).

Naturalmente que no mundo real existem sinais puramente analógicos (com uma infinidade de valores) ou sinais que utilizam outros níveis de tensão. Alguns dispositivos de entrada traduzem as tensões do sinal para níveis compatíveis com VDD e VSS. Outros dispositivos de entrada convertem os sinais analógicos em sinais digitais (valores binários formados por 0s e 1s) que a CPU pode entender e manipular. Alguns microcontroladores incluem circuitos conversores analógicos/digitais (ADC) encapsulados no mesmo componente.

Sinais de Saída

Dispositivos de saída são usados para informar ou agir com o mundo exterior através do processamento de informações realizados pela CPU. Circuitos eletrônicos (algumas vezes construídos no próprio microcontrolador) podem converter sinais digitais em níveis de tensão analógicos. Se necessário, outros circuitos podem alterar os níveis de tensão VDD e VSS nativos da CPU noutros níveis.

Códigos de operação (opcodes)

Os programas usam códigos para fornecer instruções para a CPU. Estes códigos são chamados de códigos de operação ou opcodes. Cada opcode instrui a CPU a executar uma sequência específica para realizar a sua operação. Microcontroladores de diferentes fabricantes usam diferentes conjuntos de opcodes porque são implementados internamente por hardware na lógica da CPU. O conjunto de instruções de uma CPU especifica todas as operações que podem ser realizadas. Opcodes são uma representação das instruções que são entendidas pela máquina, isto é, uma codificação em representação binária a ser utilizada pela CPU. Mnemónicos são outra representação para as instruções, só que agora, para serem entendidas pelo programador.

Mnemônicos das instruções e Assembler

Um opcode como 0x4C é entendido pela CPU, mas não é significativo para nós humanos. Para resolver esse problema, um sistema de instruções mnemónicas equivalentes foram criadas (Linguagem Assembly). O opcode 0x4C corresponde ao mnemónico INCA, lê-se “incrementa o acumulador”, que é muito mais inteligível. Para realizar a tradução de mnemônicos em códigos de máquina (opcodes e outras informações) utilizados pelo CPU é necessário um programa computacional chamado assembler (compilador para linguagem Assembly). Um programador utiliza um conjunto de instruções na forma de mnemônicos para desenvolver uma determinada aplicação e posteriormente, usa um assembler para traduzir estas instruções para opcodes que a CPU pode entender.

Após a descrição da unidade central de processamento de um microcontrolador podemos aprender a linguagem de programação Assembly. Recomenda-se a leitura da folha de dados (principalmente a seção que trata do conjunto de instruções) do microcontrolador, bem como da informação do microcontrolador HC08, família QT/QY. O próximo capítulo deste documento irá descrever diversos periféricos que podem compor um microcontrolador, como portas de entrada/saída, temporizadores, entre outros.

Periféricos

Os microcontroladores normalmente são classificados em famílias, dependendo da aplicação a que se destinam. A partir da aplicação que a família de microcontroladores se destina, um conjunto de periféricos específicos é escolhido e integrado a um determinado microprocessador. Estes microprocessadores normalmente operam com barramentos de 8, 16 ou 32 bits, e apresentam arquiteturas RISC (Reduced Instruction Set Computer) ou CISC (Complex Instruction Set Computer). Alguns exemplos de microcontroladores que utilizam microprocessadores com arquitetura RISC são o PIC (Microchip) e o MSP430 (Texas Instruments). Já o MC68HC08 e HCS08 (Freescale) e o 8051 (Intel) são exemplos de microcontroladores que utilizam arquitetura CISC.

Apesar da classificação dos microcontroladores em famílias, existem periféricos necessários a praticamente todas as aplicações, que são a memória de dados e a memória de programa. A memória de dados mais utilizada é a RAM (Random Access Memory), que é uma memória volátil, ou seja, não preserva o seu conteúdo sem uma fonte de alimentação.

Recentemente as memórias de programa sofreram uma grande mudança. A alguns anos atrás as memórias de programa mais utilizadas eram a ROM (Ready-Only Memory) e a EPROM (Erasable Programmable Read-Only Memory). O grande problema da utilização de tais memórias era que não era muito prático o desenvolvimento de um sistema embarcado. Com a popularização das memórias FLASH e ainda, devido à facilidade de utilização, cada vez mais os microcontroladores tendem a ser produzidos com esta memória, em substituição da ROM e da EPROM. Importante lembrar que a facilidade de utilização da memória FLASH se deve a esta memória ser uma variação da EEPROM (Electrically-Erasable Programmable Read-Only Memory) que permitem que múltiplos endereços sejam apagados ou escritos com sinais elétricos.

A seguir serão apresentadas as características e aplicações dos principais periféricos encontrados em microcontroladores, tais como: portas de entrada e saída, temporizadores, portas de comunicação série e conversores Analógico-digitais (A/D).

Temporizadores

Um microprocessador deve possuir um relógio (clock). O relógio pode ser implementado por um cristal oscilador que sincroniza todo o funcionamento do microprocessador, controlando o tempo de cada um dos eventos relacionados aos dispositivos integrados com ele.

Os temporizadores utilizam a base de tempo do relógio para poder implementar contagens de tempo bem específicas e configuráveis. Estes utilizam contadores, incrementados na mesma base de tempo do relógio. Desta forma é possível descrever tempo em número de ciclos de um relógio. Por exemplo, imagine um microprocessador utilizando um relógio de 20 MHz. O período relativo a esta frequência é 50 ns. Podemos representar então um tempo de 1ms através de períodos de 50ns, obtendo o valor de 20000. Ou seja, se incrementarmos um contador a cada ciclo de relógio, no caso 50ns, quando este contador atingir o valor de 20000, teremos atingido a contagem de 1 ms.

Para que este método seja aplicado, existe a necessidade da utilização de pelo menos 2 registradores. O registrador que será incrementado e o registrador que conterá o valor a ser atingido. No entanto, existe um problema relacionado a este método. Imagine que precisemos de um tempo na ordem de segundos, por exemplo, 1 segundos. Se estivermos utilizando um relógio de 4 MHz, seria necessário registradores de 24 bits para representar o valor de 4x106. Com o intuito de reduzir o tamanho destes contadores, os temporizadores apresentam a possibilidade de divisão do relógio, normalmente por valores múltiplos de 2. No caso do exemplo acima, poderíamos dividir o relógio por 128, obtendo uma frequência de 31,250 KHz. Então, para representarmos 1 segundo, seria necessário uma contagem de 31250 períodos de 32µs, ou seja, um valor que pode ser representado em 16 bits.

Devemos lembrar que a divisão terá influência somente no tempo de incrementação do contador, continuando o barramento interno do microprocessador a operar com o relógio original.

Para exemplificar a configuração dos registradores relativos a um temporizador num microcontrolador será utilizado o microcontrolador MC68HC908QY4. O processo de configuração é apresentado a seguir.

Os microcontroladores da linha HC08 normalmente possuem 1 ou 2 temporizadores. Os registradores relativos a estes temporizadores apresentam nomes semelhantes,

tendo apenas o número no temporizador para diferenciá-los. Os três registradores de configuração do temporizador neste microcontrolador são: TSC, TCNT e TMOD. Abaixo é apresentado o diagrama de blocos do temporizador destes microcontroladores. É importante ressaltar que alguns dos registradores apresentados na figura são relativos ao módulo PWM e captura de entrada.

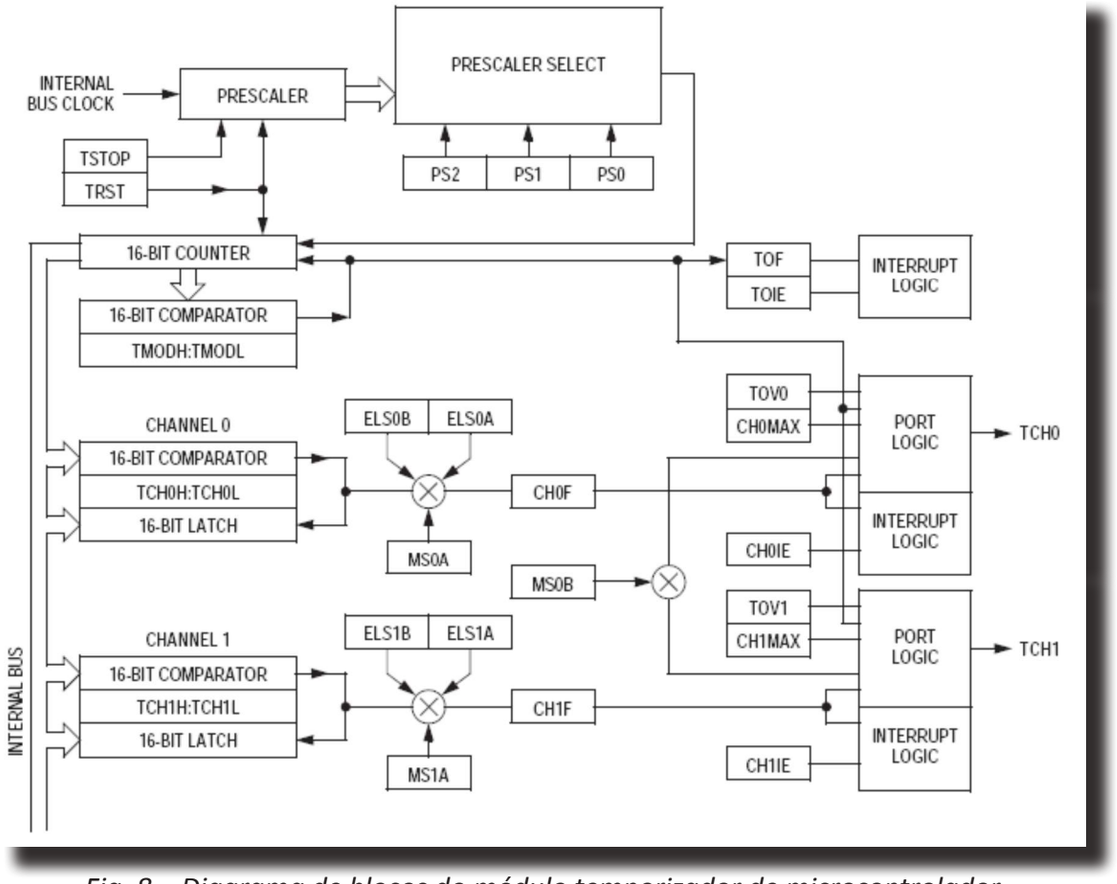


Fig. 8 – Diagrama de blocos do módulo temporizador de microcontrolador MC68HC908QY4.

TSC (Timer Status and Control Register):

Possibilita habilitar a interrupção do temporizador, verificar o estado da flag de interrupção, para-lo, reiniciar a contagem e dividir o relógio para obter a base de tempo.

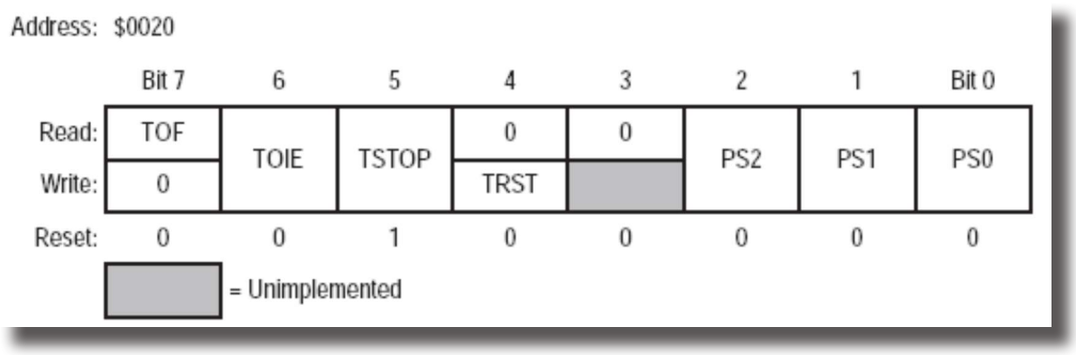


Fig. 9 – Registrador TSC

Abaixo serão descritos as funções de cada um dos bits deste registrador:

TOF (Timer Overflow Flag): Este bit de escrita/leitura torna-se no nível alto quando o registrador contador (TCNT) atinge o valor do registrador de módulo de contagem (TMOD), condição essa que indica o estouro da contagem de tempo. O procedimento correto para limpar esta indicação é ler o registrador TSC e escrever um "0" lógico para o bit TOF;

1 = O módulo temporizador atingiu o valor desejado

0 = O módulo temporizador não atingiu o valor desejado

TOIE (Timer Overflow Interrupt Enable Bit): Este bit de escrita/leitura habilita a interrupção do temporizador quando o bit TOF for levado ao nível alto.

1 = Interrupção do temporizador ativa

0 = Interrupção do temporizador desabilitado

TSTOP (Timer Stop Bit): Este bit de escrita/leitura é usado para o incremento do contador de tempo.

1 = Contador de tempo parado

0 = Contador de tempo ativo

TRST (Timer Reset Bit): Levar este bit de escrita para nível lógico 1 irá iniciar o contador de tempo com zero e colocar o divisor de base de tempo para o estado inicial, ou seja, divisão por 1.

1 = Pré-Escala de base de tempo e contador iniciado com o valor "0";

0 = Sem efeito

PS[2:0] (Prescaler Setect Bits) - Bits de pré-escala da base de tempo. Estes bits de escrita/leitura selecionam um dos sete possíveis valores de divisão da base de tempo do relógio para utilização como base de tempo do temporizador.

PS2	PS1	PS0	Base de tempo do temporizador
0	0	0	Clock de barramento interno / 1
0	0	1	Clock de barramento interno / 2
0	1	0	Clock de barramento interno / 4
0	1	1	Clock de barramento interno / 8
1	0	0	Clock de barramento interno / 16
1	0	1	Clock de barramento interno / 32
1	1	0	Clock de barramento interno / 64
1	1	1	Não disponível

Fig. 10 – Base de tempo do temporizador.

TCNT (Timer Count Registers):

Estes registradores são somente de leitura e contém o valor mais significativo (TCNTH) e menos significativo (TCNTL) do contador do temporizador. A leitura do registrador mais significativo deve ser realizada primeiro.

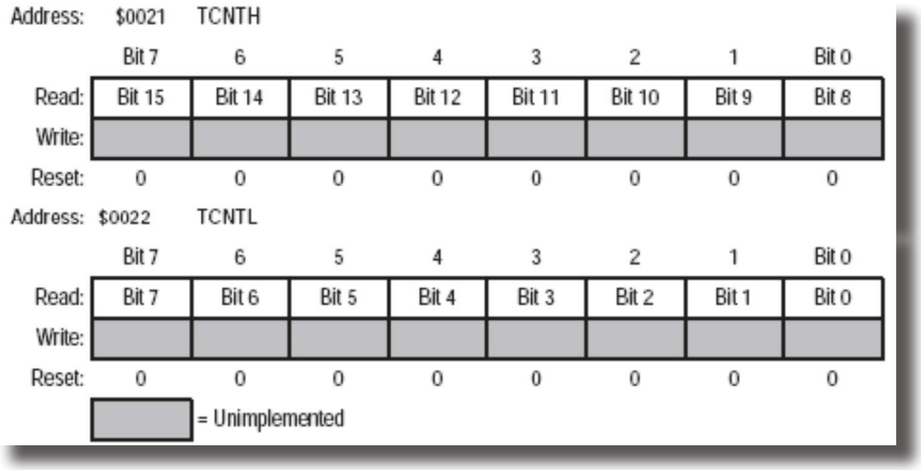


Fig. 11 – Registrador TCNTH e TCNTL.

TMOD (Timer Module Registers):

Estes registradores de escrita/leitura contêm o valor do módulo da contagem do temporizador. Quando os registradores de contagem (TCNT) atingem o valor dos

registradores de módulo (TMOD), o bit TOF torna-se nível lógico “1” e os registradores de contagem resumem a contagem para $0000 até o próximo passo de clock. Escrever no registrador TMODH inibe o bit TOF até que o registrador TMODL seja escrito.

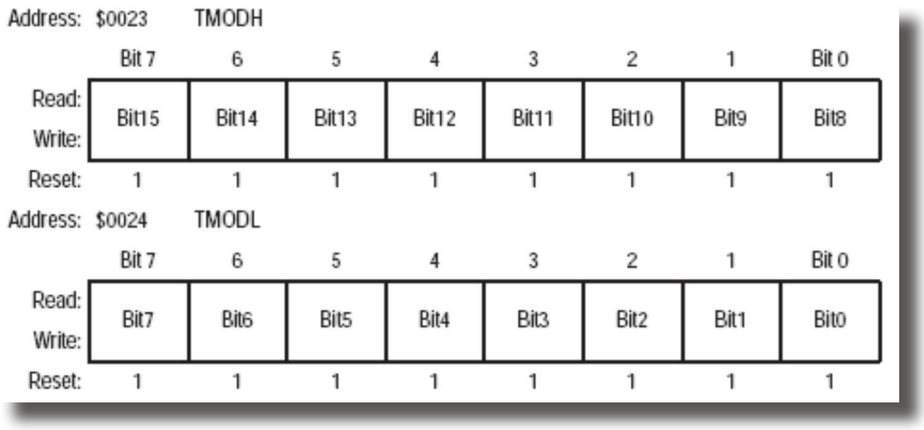


Fig. 12 – Registradores TMODH e TMODL.

Estando os registradores a serem configurados pode-se demonstrar um exemplo de utilização. Abaixo será implementada a configuração de um temporizador utilizando um relógio de 3,2 MHz para obter a base de tempo de 10ms.

Período relativo a base de tempo: $Período = \frac{1}{(3{,}2 \times 10^{6})} = 312{,}5 \times 10^{-9}$

Cálculo do valor de módulo de tempo: $Módulo = \frac{10 \times 10^{-3}}{(312{,}5 \times 10^{-9})} = 32000$

O valor 32000 deve ser escrito nos registradores TMODH e TMODL da seguinte maneira: O valor deve ser convertido para hexadecimal, sendo este igual a 7D00h. O valor obtido é um valor válido dentro dos 16 bits referentes aos registradores TMODH e TMODL, ou seja, a base de tempo do relógio foi dividida por 1. Desta forma, TMODH é igual a 7Dh e TMODL é igual a 00h. Abaixo o exemplo de configuração dos registradores para este caso é apresentado, tanto em assembly, quanto em linguagem “c”.

Assembly:

```
MOV       #$7D,TMODH
CLR       TMODL
; Configuração do registrador TSC:
; Divisão por 1, timer ativo com interrupção do temporizador habilitada.
; TOF  TOIE  TSTOP  TRST  Reservado  PS2 PS1 PS0
;  0     1       0       0        0       0    0    0
MOV       #$40,TSC
```

"c":

```
TMOD = 32000;
/*Configuração do registrador TSC:
Divisão por 1, timer ativo com interrupção do temporizador habilitada.
TOF  TOIE  TSTOP  TRST  Reservado  PS2 PS1 PS0
  0     1       0       0        0       0    0    0 */
TSC = 0x40;
```

Para demonstrar um caso onde é necessário a divisão da base de tempo, o temporizador será configurado utilizando um relógio de 3,2 MHz para obter a base de tempo de 100ms.

Período relativo a base de tempo: $Período = \frac{1}{(3{,}2 \times 10^{6})} = 312{,}5 \times 10^{-9}$

Cálculo do valor de módulo de tempo: $Módulo = \frac{100 \times 10^{-3}}{(312{,}5 \times 10^{-9})} = 320000$

Para representar 320000 é necessário mais do que os 16 bits disponíveis. Desta forma faz-se a divisão da base de tempo. Dividindo 320000 pelo maior valor possível em 16 bits, encontramos 4,88. Assumimos então o próximo valor válido de divisão, ou seja, oito, como fator de divisão da base de tempo.

Fator de divisão da base de tempo: $Fator = \frac{320000}{65535} = 4,88$

E, recalculando os valores de configuração:

Período relativo a base de tempo: $Período = \frac{1}{(\frac{3,2 \times 10^6}{8})} = 2,5 \times 10^{-6}$

Cálculo do valor de módulo de tempo: $Módulo = \frac{100 \times 10^{-3}}{(2,5 \times 10^{-6})} = 40000$

Sendo 40000 um valor válido em 16 bits, a configuração do temporizador em linguagem "c" é apresentada abaixo:

"c":

```
TMOD = 40000;
/*Configuração do registrador TSC:
Divisão por 8, timer ativo com interrupção do temporizador habilitada.
TOF  TOIE  TSTOP  TRST  Reservado  PS2 PS1 PS0
  0     1      0     0          0    0   1   1 */
TSC = 0x43;
```

Obs.: Devemos lembrar que o código contido nas interrupções deve ser o menor possível, com o intuito de evitar que o tempo de execução deste código seja superior ao tempo da próxima interrupção. Por exemplo, se configurarmos a interrupção do temporizador para ocorrer a cada 100µs, o tempo de execução do código contido nesta interrupção não deve ser superior a esta base de tempo.

Exemplo de utilização da interrupção do temporizador (esta interrupção é conhecida como interrupção de estouro de tempo):

Assembly:

```
; Interrupção do temporizador.
TOVER:   BCLR   7,TSC        ; Limpa a flag da interrupção do temporizador
         INC    I            ; Incrementa uma variável qualquer
         RTI                 ; Retorna da interrupção
```

"c":

```
// Interrupção do temporizador
interrupt void tover(void) {
   TSC_TOF = 0;              // Limpa a flag da interrupção do temporizador
    i++;                     // Incrementa uma variável qualquer
    }                        // Retorna da interrupção
```

PWM (Pulse Width Modulation)

O módulo de geração de Modulação por Largura de Pulso (PWM) é um recurso muito utilizado para o controle de motores e conversores CC-CC em geral. A partir dele é possível gerar um sinal analógico, apesar de sua saída ser um sinal digital que assume apenas os níveis lógicos alto (um) e baixo (zero). A saída gerada é uma onda quadrada, com frequência constante e largura de pulso variável. Estes conceitos estão diretamente relacionados com o período fixo e o ciclo ativo (duty cycle) respetivamente.

A frequência de uma onda pode ser definida como a quantidade de vezes que ela se repete no tempo. E o período é cada pedaço dessa onda que se irá repetir.

O duty cycle define o tempo de sinal ativo (nível lógico alto) num período fixo. Assim, quando temos um duty cycle de 100%, temos nível lógico alto por todo o período. Um duty cycle de 50% define a metade do período em nível lógico alto e a outra metade em nível lógico baixo. Se uma saída TTL for utilizada, a tensão média de saída num duty cycle de 50% será 2,5V. Estes conceitos são demonstrados na figura abaixo. Devemos lembrar que o PWM nem sempre possui estado inicial positivo, podendo iniciar o período com nível lógico baixo.

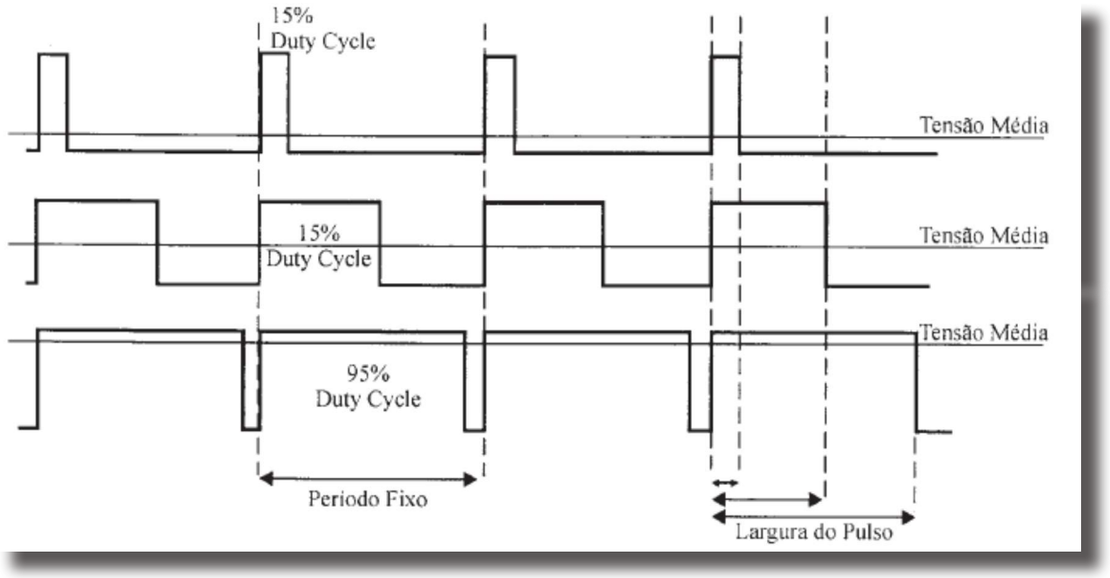


Fig. 13 – Sinais modulados por Largura de Pulso.

A base de tempo dos módulos PWM normalmente é implementada de duas formas. Uma destas formas é utilizando o próprio módulo temporizador como base de tempo no PWM, ou seja, se o temporizador está configurado para um período de 1ms, a frequência do PWM será de 1 KHz. A outra forma é utilizando um temporizador específico para o PWM, que deve ser configurado para a frequência desejada. Ainda, um temporizador pode ser utilizado como base de tempo de várias saídas PWM, ou seja, vários PWM com a mesma frequência, mas larguras de pulso diferentes.

A figura a seguir irá exemplificar o funcionamento de um PWM num microcontrolador onde o registrador PTPER possui o valor referente ao período do PWM e os registradores PWM1H e PWM2H representam dois canais de saída PWM.

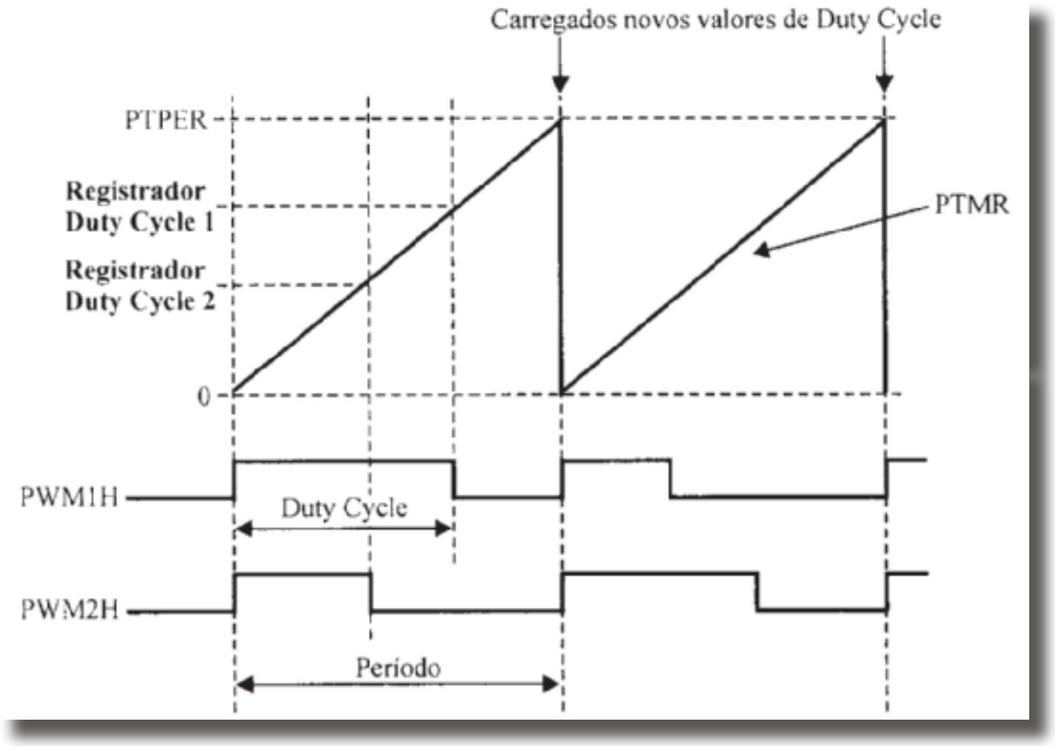


Fig. 14 – Sinais PWM com mesmo período e largura de pulso diferentes.

Para exemplificar a configuração de um módulo PWM será utilizado novamente o microcontrolador MC68HC908QY4. Este microcontrolador utiliza o temporizador como base de tempo para o período do PWM. Desta forma, para configurar a frequência do PWM deve-se utilizar a mesma metodologia adotada para o temporizador. Neste exemplo será configurado um PWM de 10 KHz com largura de pulso inicial do ciclo ativo de 40%, onde o relógio utilizado será de 3,2 MHz.

Período da base de tempo do relógio: $Período = \frac{1}{(3{,}2 \times 10^{6})} = 312{,}5 \times 10^{-9}$

Valor de módulo de tempo para 100µs: $Módulo = \frac{100 \times 10^{-6}}{(312{,}5 \times 10^{-9})} = 320$

Estando o módulo da base de tempo do PWM configurado, deve-se partir para a configuração da largura do pulso e do nível do ciclo inicial do PWM. Através do módulo PWM deste microcontrolador é possível obter duas saídas PWM para cada temporizador.

Como o microcontrolador utilizado possui somente um temporizador, é possível obter duas saídas PWM, conhecidas como canal 0 e canal 1. Os registradores de configuração do módulo PWM são: TSC0, TSC1, TCH0 e TCH1. Pode-se notar que os valores 0 e 1 são relativos ao canal a que se destina a configuração.

Este microcontrolador apresenta duas maneiras de se utilizar estes canais. As saídas PWM podem ser configuradas no modo com buffer ou sem buffer. No modo com buffer ambos os canais são aproveitados para gerar uma saída PWM. O modo com buffer opera da seguinte maneira:

Ambos os canais são configurados para operar com buffer. A configuração da largura de pulso inicial é realizada no canal zero. Quando se desejar alterar a largura de pulso da saída PWM, altera-se o valor da largura de pulso no canal um. Ou seja, todas as vezes que se desejar alterar o valor da largura de pulso é realizado um intercalamento entre os dois canais. A saída PWM ficará agregada ao pino de saída do canal zero.
Já no modo sem buffer, cada canal de tempo pode gerar uma saída PWM. O cuidado que deve ser tomado neste caso é que alterações de largura de pulso devem ser realizadas em locais bem específicos. A alteração da largura de pulso para um valor superior deve ser realizada na interrupção de estouro de tempo (temporizador). A alteração da largura de pulso para um valor inferior deve ser realizada na interrupção de comparação do devido canal. Na figura a seguir é apresentado os locais referentes as interrupções supracitadas.

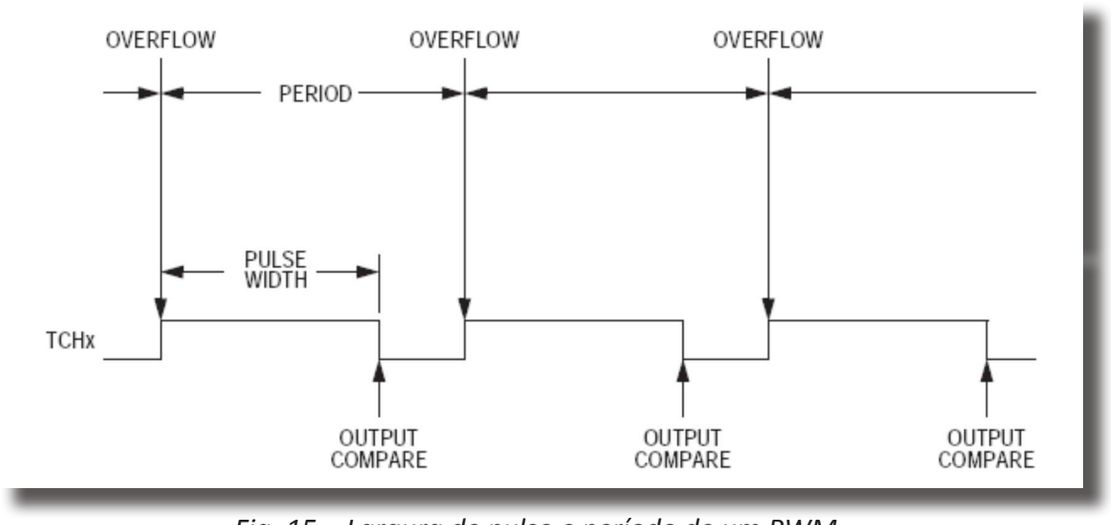


Fig. 15 – Largura de pulso e período de um PWM.

Abaixo temos a configuração de bits relativa aos registradores TSC0 e TSC1.

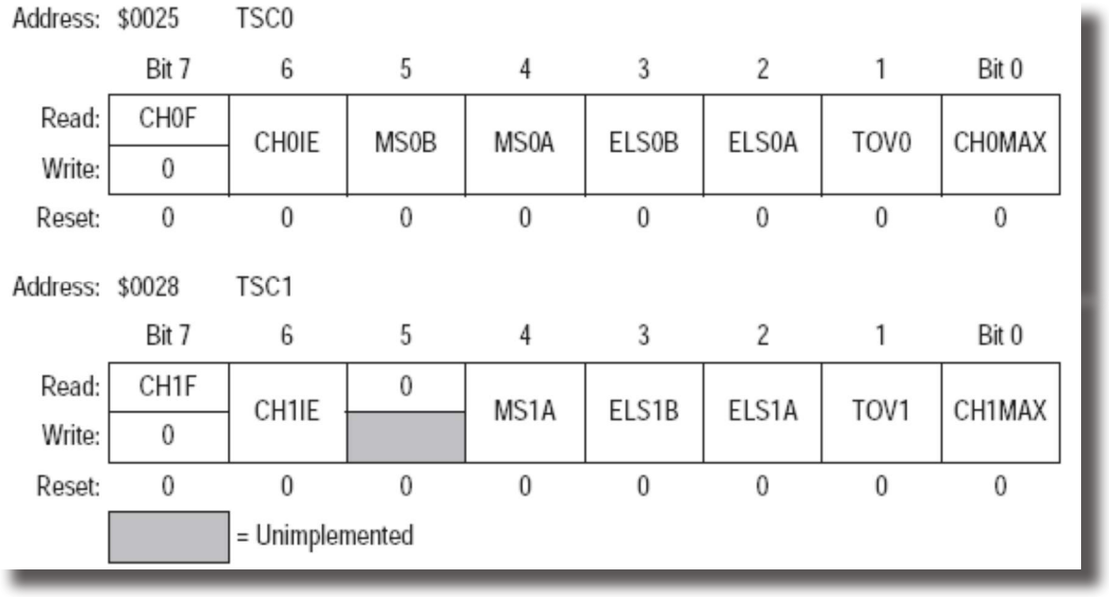


Fig. 16 – Timer Channel Status And Control Registers.

A seguir serão descritos as funções de cada um dos bits deste registrador:

CHxF (Channel x Flag Bit): Quando o canal x está configurado para comparação de saída (modo que permite a implementação de um PWM), esta flag torna-se 1 quando o valor do registrador contador de tempo (TCNT) atinge a valor contido no registrador do canal x (TCHx). Para limpar esta flag deve-se ler o registrador TSCx e escrever um zero lógico para este bit.

1 = Comparação de saída no canal x

0 = Sem comparação de saída no canal x

ChxIE (Channel x Interrupt Enable Bit): Este bit de escrita/leitura permite habilitar a interrupção de comparação de saída para o canal x.

1 = Interrupção do canal x habilitada

0 = Interrupção do canal x desabilitada

TOVx (Toggle On Overflow Bit): Quando o canal x está configurado para comparação de saída, este bit de escrita/leitura controla o comportamento da saída do canal x quando ocorre um estouro de tempo no temporizador.

1 = Valor lógico no pino relativo ao canal x se altera no estouro de tempo

0 = Valor lógico no pino relativo ao canal x não se altera no estouro de tempo

ChxMAX (Channel x Maximum Duty Cycle Enable Bit): Quando o bit TOVx está em nível lógico 1, ao colocar no nível 1 o bit ChxMAX irá forçar o Duty Cycle do canal para 100%. Como a figura a seguir demonstra, o efeito relativo a este bit só é notado 1 ciclo após deste bit ser alterado. O Duty Cycle permanece 100% até que este bit volte ao estado lógico zero.

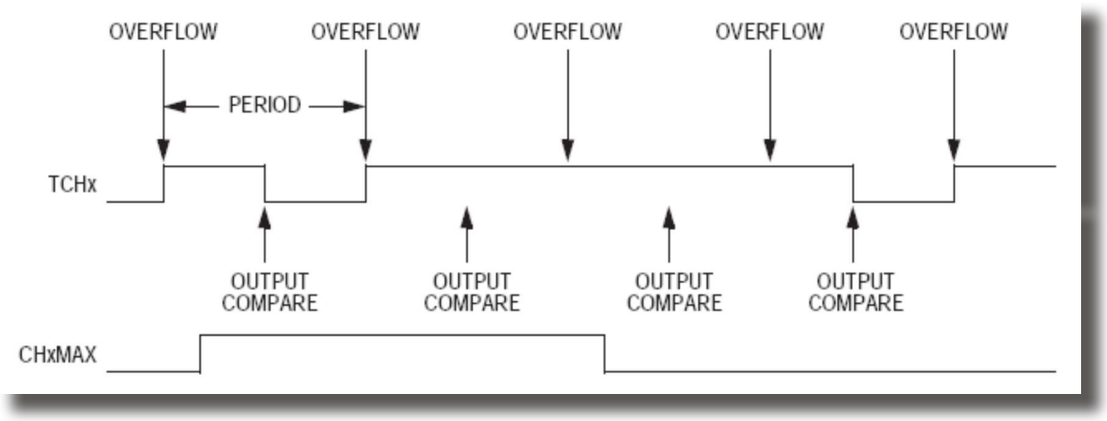


Fig. 17 – Latência do bit CHxMAX.

Os outros bits relativos a estes registradores são configurados a partir da tabela abaixo:

MSxB	MSxA	ELSxB	ELSxA	Modo	Configuração
X	0	0	0	Porta I/O	Nível de saída inicial alto
X	1	0	0		Nível de saída inicial baixa
0	0	0	1	Captura de Entrada	Captura somente na borda de subida
0	0	1	0		Captura somente na borda de descida
0	0	1	1		Captura em ambas as Bordas
0	1	0	1	Comparação de saída ou PWM	Inverte nível na comparação
0	1	1	0		Nível para baixo na comparação
0	1	1	1		Nível para cima na comparação
1	X	0	1	Comparação de saída ou PWM com Buffer	Inverte nível na comparação
1	X	1	0		Nível para baixo na comparação
1	X	1	1		Nível para cima na comparação

Fig. 18 – Seleção de modo, nível e borda.

O registrador onde é realizada a configuração da largura de pulso é o TCHx, sendo TCHxH o registrador de maior significado e TCHxL o de menor significado na palavra de 16 bits que irá representar a largura do pulso. Abaixo estes registradores são apresentados.

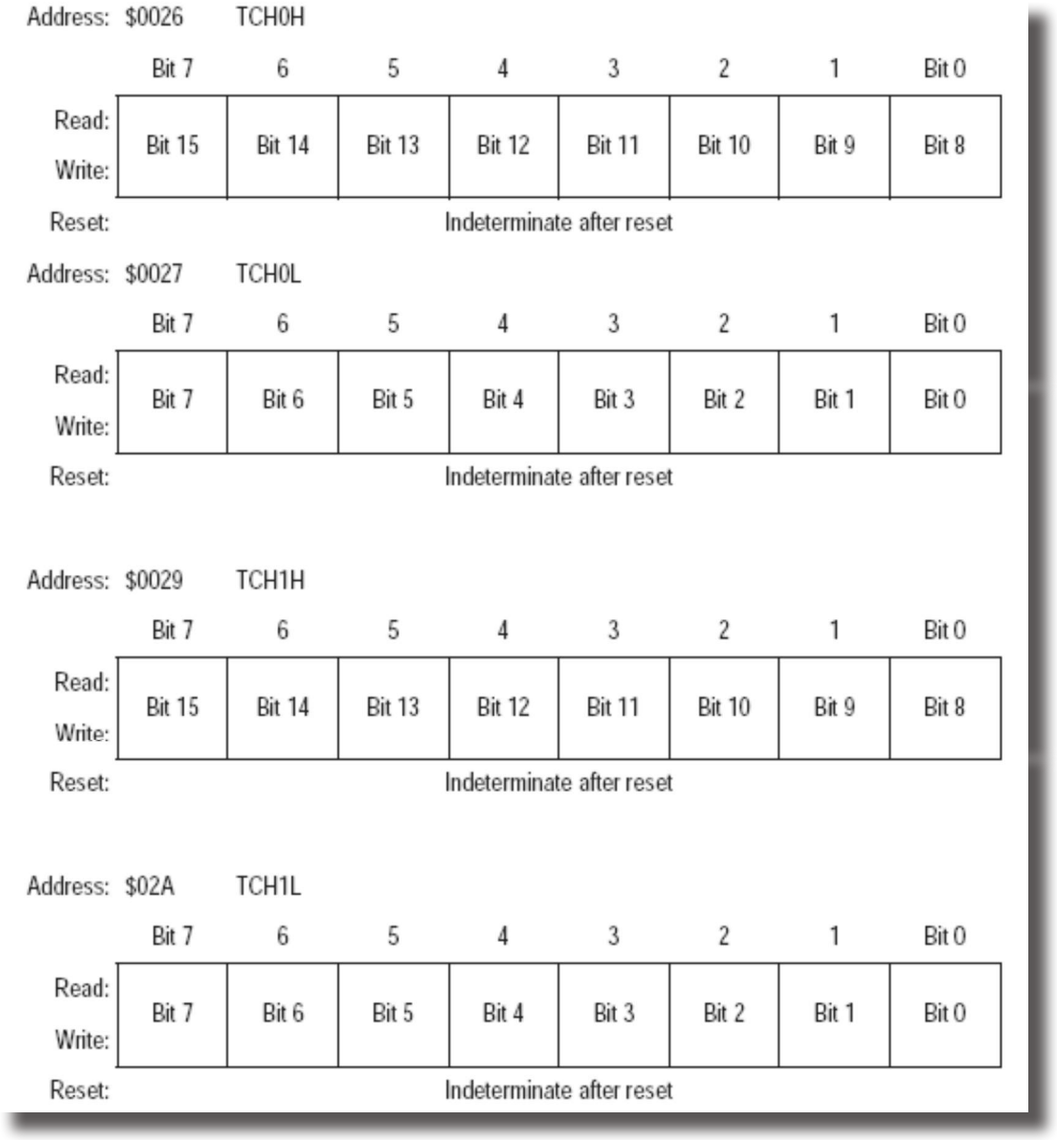


Fig. 19 – Registradores TCH0H, TCH0L, TCH1H e TCH1L.

Voltando ao exemplo onde se deseja configurar um PWM de 10KHz com largura de pulso inicial igual a 40% e ciclo inicial ativo, utilizando um relógio de 3,2MHz. O módulo referente ao período do PWM já foi definido, sendo igual a 320. Para configurarmos a largura inicial para 40% deve-se utilizar 40% do valor do módulo do período encontrado, ou seja, TCH0 = 320 x 40% = 128.

A seguir será apresentado a configuração do PWM do exemplo acima, utilizando modo sem buffer com saída no canal zero, tanto para Assembly quanto para “c”.

Assembly:

```
; Configuração do módulo de tempo
LHDX    #!320            ; Valor relativo ao período de 100us
STHX    TMD              ;  Move para o registrador de módulo do temporizador
LDHX    #!128             ; Valor de 40% de largura de pulso (40% de 320)
STHX    TCH0             ; Move para o registrador do canal 0
CLRH                     ; Limpa a parte alta do registrador de índice, utilizado anteriormente
; Saída PWM no canal 0, com alteração para nível lógico baixo na comparação
; CH0F  CH0IE  MS0B  MS0A  ELS0B  ELS0A  TOV0  CH0MAX
;   0      1      0      1      1      0      1      0
MOV     #$5A,TSC0
; Período do PWM igual a 100us
; TOF  TOIE  TSTOP  TRST  Reservado  PS2 PS1 PS0
;   0     1      0      0      0      0    0    0
MOV     #$40,TSC
```

“c”:

```
TMOD = 320;              /* Perído do PWM*/
TCH0 = 128;              /* 40% de PWM */

/*Saída PWM no canal 0, com alteração para nível lógico baixo na comparação
CH0F  CH0IE  MS0B  MS0A  ELS0B  ELS0A  TOV0  CH0MAX
  0      1      0      1      1      0      1      0      */
TSC0 = 0x5A;

/*Período do PWM igual a 100us
 TOF  TOIE  TSTOP  TRST  Reservado  PS2 PS1 PS0
   0     1      0      0      0      0    0    0    */
TSC = 0x40;              /* Não divide o clock */
```

Apesar de termos definido as inicializações, se o modo sem buffer está sendo utilizado, precisamos implementar também no código as interrupções de estouro de tempo e comparação de saída para ser possível alterar o valor da largura de pulso desta saída.

Assembly:

```
; Interrupção do temporizador.
TOVER:  BCLR    7,TSC                   ; Limpa a flag da interrupção do temporizador
        BRCLR   condição,SAIR_OVER      ; Verifica a necessidade de aumentar a largura
                                        ; Se a flag estiver com zero, não corrige
        BCLR    AUM,FLAG                ; Limpa a flag da necessidade de aumentar
        MOV     Valor_corrigido,TCH0    ; aumenta a largura de pulso
SAIR_OVER
         RTI                    ; Retorna da interrupção
```

```
; Interrupção da comparação

COMP:   LDA     TSC0
        BCLR    7,TSC0                  ; Limpa a flag da interrupção da comparação
        BRCLR   condição,SAIR_COMP      ; Verifica a necessidade de diminuir a largura
                                        ; Se a flag estiver com zero, não corrige
        BCLR    DIM,FLAG                ; Limpa a flag da necessidade de diminuir
        MOV     Valor_corrigido,TCH0    ; diminui a largura de pulso
SAIR_COMP
         RTI                            ; Retorna da interrupção
```

"c":

```
// Interrupção do temporizador
interrupt void tover(void) {
    TSC_TOF = 0;                    // Limpa a flag da interrupção do temporizador
     if (condição){                  /* Indicação para aumenta largura do PWM*/
        flag_aum = 0;               // Limpa flag de condição
        TCH0 = Valor_corrigido;     // Corrige a largura do pulso
     }
}                                   // Retorna da interrupção

// Interrupção da comparação
interrupt void comparacao(void){
  byte i;
  i = TSC0;                         // Lê registrador de estado do PWM
  TSC0_CH0F = 0;                    // Limpa a flag
  if (condição) {                    // Verifica se é necessário diminuir a largura
    flag_dim = 0;                   // Limpa flag de condição
    TCH0 = Valor_corrigido;         // Corrige a largura do pulso
  }
}
```

Em algumas aplicações é necessário a inserção de um tempo morto (dead time) devido ao tempo de comutação de certos tipos de componentes utilizados ou devido às topologias dos sistemas agregados a estas saídas PWM.

Normalmente estes tempos mortos são inseridos em pares complementares de saídas PWM, ou seja, quando uma das saídas comuta para nível lógico alto, a outra saída comuta para nível lógico baixo. Abaixo é apresentado um exemplo de utilização de tempos mortos.

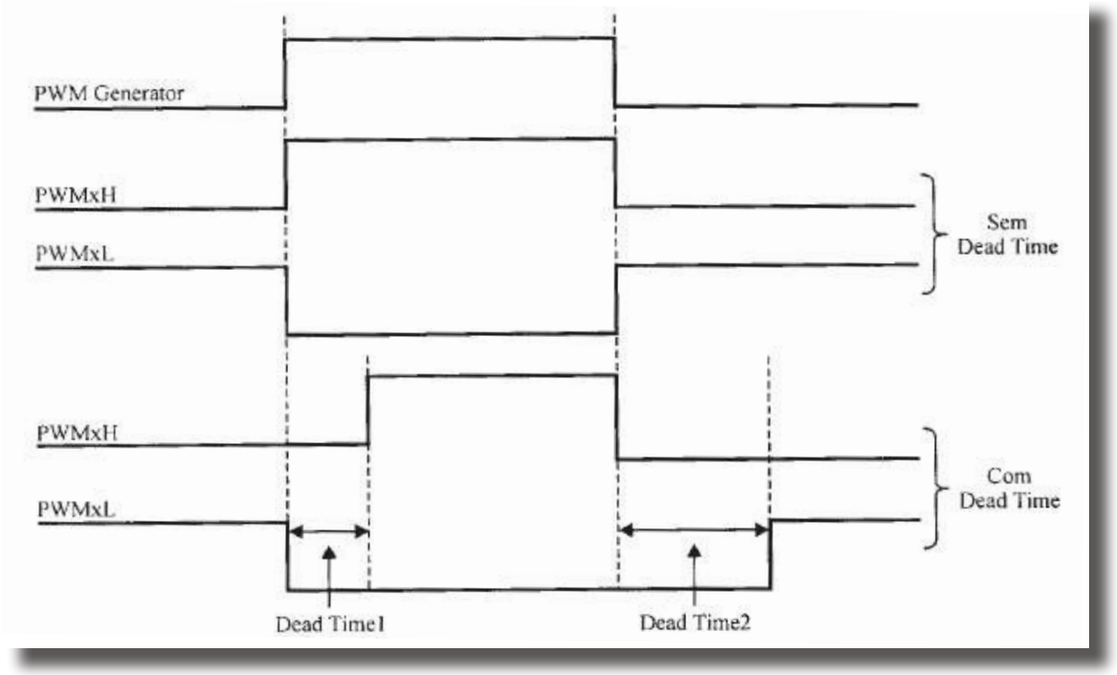


Fig. 20 – Exemplo de utilização de tempo morto.

Conversores Analógico-Digital e Digital-Analógico

De um modo geral, os sinais encontrados no mundo real são contínuos (ou analógicos, pois variam no tempo de forma contínua), como, por exemplo: a intensidade luminosa de um ambiente que se modifica com a distância; a aceleração de um carro de corrida, etc. Os sinais manipulados por computadores e sistemas embarcados são digitais, como por exemplo, uma faixa de áudio lida de um compact disk.

A conversão analógico-digital (A/D) é o processo que possibilita a representação de sinais analógicos no mundo digital. Desta forma é possível utilizar os dados extraídos do mundo real para cálculos ou operar os seus valores.

Em geral, o conversor A/D está presente internamente nos processadores e controladores de sinais digitais e alguns microcontroladores, mas também existem circuitos integrados dedicados a este fim.

Basicamente é um bloco que apresenta portas de entrada e saída. A entrada recebe sinais elétricos de forma contínua e possui uma faixa de tensão de entrada máxima e mínima. Nos microcontroladores que possuem um conversor A/D e operam na faixa de 5V, geralmente a faixa de tensão aceita sinais elétricos entre -5V e +5V.

Na saída o sinal é amostrado num dado intervalo de tempo fixo (determinado pela frequência de amostragem). Esta amostra disponibiliza um certo valor que representa o sinal original naquele momento (quantização). As características de quantização estão relacionadas à precisão do conversor.

Para ilustrar esta situação, imagine que você queira mostrar a temperatura de um forno num display de cristal líquido (LCD). Para isto seriam necessários alguns componentes eletrônicos. Os mais expressivos são: um transdutor (sensor de temperatura), um display de cristal líquido (LCD), um processador digital e um conversor analógico digital.

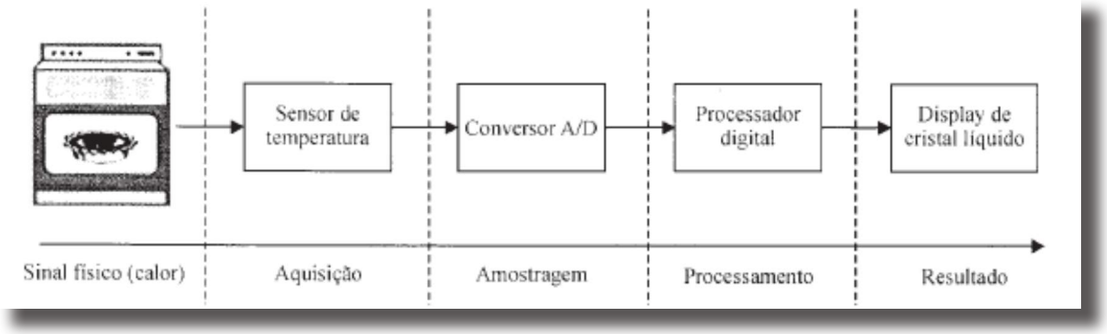


Fig. 21 – Diagrama de blocos de uma conversão A/D de um sinal de temperatura.

A temperatura é um sinal analógico. O sensor de temperatura converte a temperatura num sinal de impulsos elétricos analógicos. O conversor A/D recebe esse sinal e transforma-o em sinal digital, através de amostragem, entregando ao processador. Este, por sua vez, manipula esses dados e envia-os para o display, mostrando em graus a temperatura do forno. A figura abaixo mostra a representação do sinal analógico de temperatura e seu equivalente na forma digital.

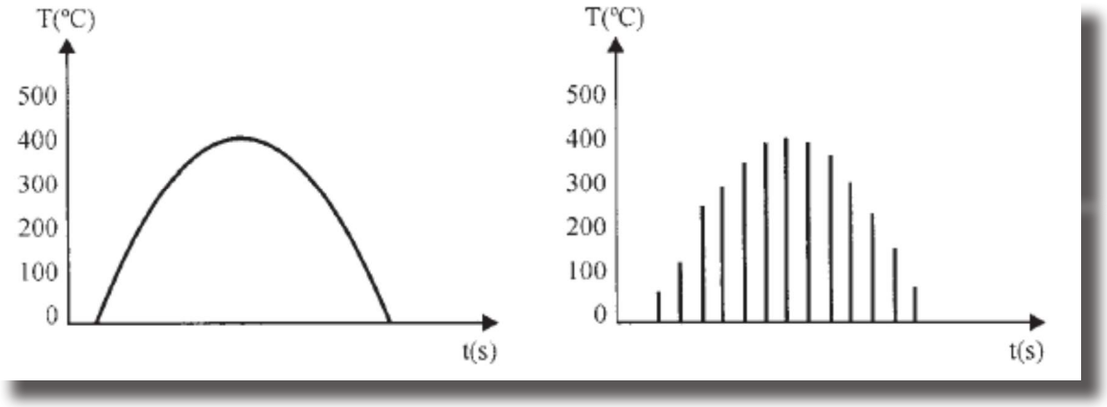


Fig. 22 – Representação de um sinal de temperatura analógico e digital.

A informação digital é diferente da sua forma original contínua em dois aspetos fundamentais:

- É amostrada porque é baseada em amostragens, ou seja, são realizadas leituras em um intervalo fixo de tempo no sinal contínuo;
- É quantificada porque é atribuído um valor proporcional a cada amostra.

Explorando um pouco mais o caso do forno, a figura abaixo detalha um pouco mais as três etapas mais importantes do processo: a aquisição, a amostragem e o processamento.
Neste diagrama de blocos, o sinal analógico é capturado pelo transdutor (sensor), em seguida passa por um filtro, denominado de anti-alias, a fim de diminuir os ruídos. A chave representa a frequência de amostragem do conversor A/D, entregando ao processador o sinal digitalizado.

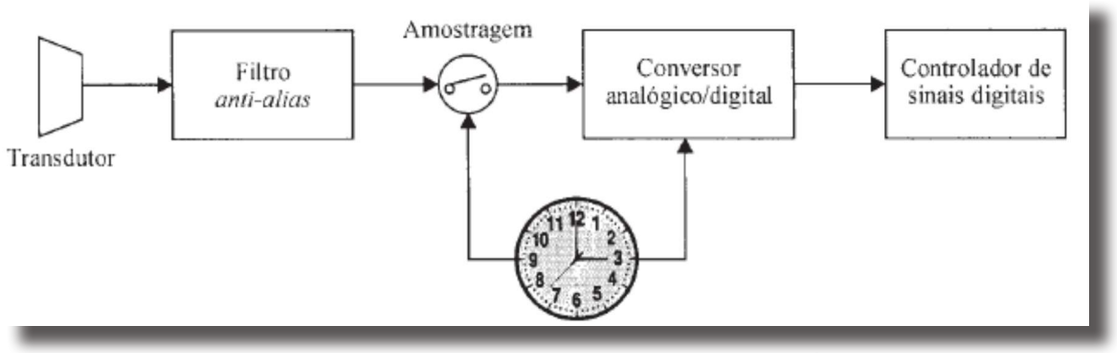


Fig. 23 – Diagrama de blocos da conversão A/D.

A frequência de amostragem é o número de amostras capturadas num segundo. Esta frequência é dada em Hertz (Hz) e é considerada adequada quando se pode reconstruir o sinal analógico razoável a partir de amostras obtidas na conversão.
A taxa de conversão ou frequência de amostragem é de suma importância para o processamento de sinais reais. Para obter uma taxa de amostragem adequada pode-se utilizar os teoremas de Nyquist ou Shannon. Estes teoremas indicam que um sinal contínuo x(t) pode ser amostrado adequadamente se tiver banda limitada, ou seja, o seu espectro de frequências não podem conter frequências acima de um valor máximo (Fmáx - frequência máxima). Ainda, outro ponto importante é que a taxa de amostragem (Fa - Frequência de amostragem) deve ser escolhida para ser no mínimo duas vezes maior do que a frequência máxima (Fmáx). Por exemplo, para representar um sinal de

áudio com frequências até 10 KHz, o conversor A/D deve amostrar esses sinais utilizando uma frequência de amostragem de no mínimo 20 KHz.

Para melhor entendimento, vamos ver como funciona um conversor A/D de 4 bits (Figura abaixo).

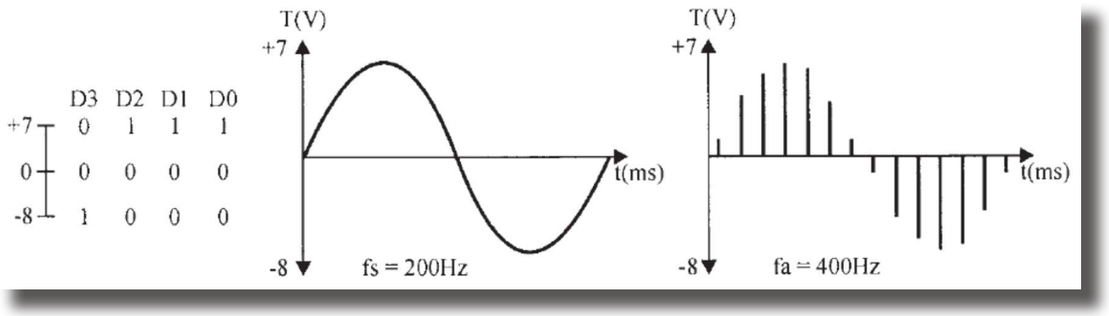


Fig. 23 – Conversão A/D de 4 bits.

Com 4 bits o máximo representável é o número 16. Isso quer dizer que temos uma faixa de 0 a 15 (não sinalizado) ou +7 a -8 (sinalizado). Nesse conversor fictício, teremos uma variação a cada 1 volt. A figura anterior mostra um sinal de áudio de 200 Hz variando de +7 a -8 volts, que pode ser capturado por um microfone. Conforme o teorema de Nyquist, seria necessário uma frequência de amostragem de 400 Hz.

Lembrando que, se o sinal de áudio possuísse amplitude maior que a faixa representável do conversor A/D (7V a -8V), então não seria possível converter tal sinal.

O conversor D/A possui todas as características do conversor A/D, as quais diferem apenas porque o conversor D/A trabalha com um sinal digital e o transforma num analógico. Por exemplo, numa aplicação de áudio, um microfone captura o áudio e o envia a um conversor A/D, que entrega o sinal amostrado e quantificado a um processador digital. Este último efetua diversas operações com o sinal de áudio. Só então o processador envia ao conversor D/A, para remontar o sinal analógico a partir do sinal digital, para ser reproduzido num alto-falante. Um exemplo de conversor D/A de 16 bits é o DAC1221, da Texas Instruments.

Novamente para exemplificar a configuração de um conversor A/D num microcontrolador será utilizado o microcontrolador MC68HC908QY4. Este microcontrolador possui um conversor A/D por aproximação sucessiva linear com quatro canais, ou seja, existem 4 entradas possíveis para sinais analógicos que são multiplexadas para um único conversor A/D. Isto implica em que só um canal será convertido num determinado momento.

A resolução deste A/D é de 8 bits, no entanto, a Freescale disponibilizou uma nova versão com A/D de 10 bits, o MC68HC908QY4A. Na figura a seguir é apresentado o diagrama de blocos do conversor A/D deste microcontrolador.

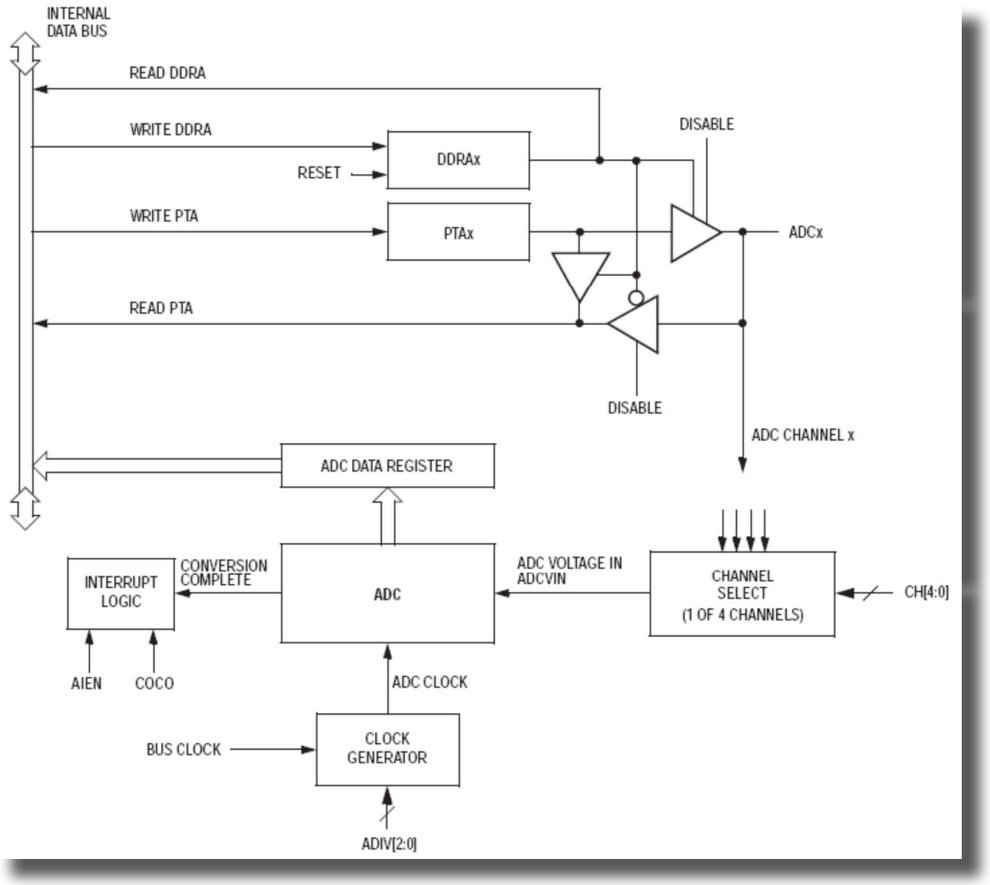


Fig. 24 – Diagrama de blocos do conversor A/D do microcontrolador MC68HC908QY4.

Exercícios propostos

Perguntas de exemplo sobre este módulo:

1. Quais as partes integrantes que constituem os Microcontroladores?

2. Que módulos compõem a CPU?

3. Qual a função da ULA?

4. Qual a função da Unidade de Controlo?

5. O que faz um Registrador?

6. Qual o trabalho que realiza uma CPU?

7. Qual a utilização do Acumulador?

8. Qual a função de um stack Pointer?

9. Para que se utiliza um clock?

10. Qual a função da memória RAM?

11. Qual a vantagem das memórias EEPROM?

12. O que é a linguagem Assembly?

13. Como trabalham os temporizadores?

14. Onde se pode utilizar o módulo PWM?

15. Porque se utiliza um conversor A/D?

16. O que se obtém à saída de um conversor A/D?

17. O que é a Amostragem num conversor A/D?

18. Quais as três etapas mais importantes no processo da conversão A/D?

19. O que é a Frequência de amostragem?

20. Qual a frequência de amostragem de um sinal de áudio de 10KHZ?

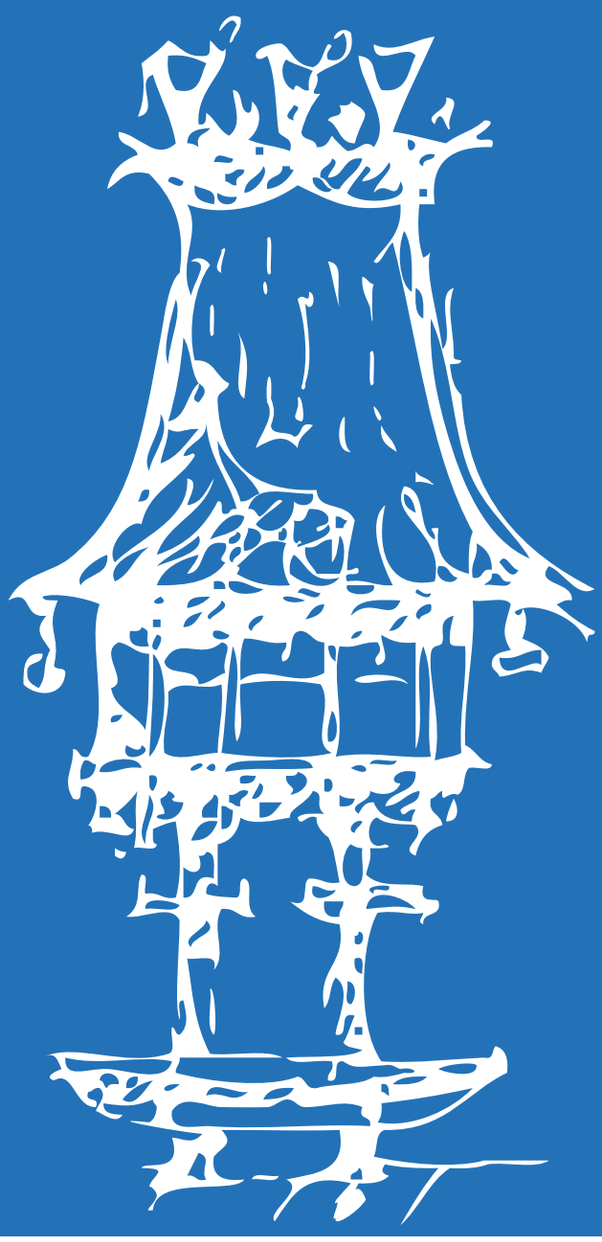
Bibliografia

GONÇALVES, Victor, Sistemas Electrónicos com Microcontroladores. ETEP. (s.d.).

JR, Vidal Pereira da Silva, Aplicações Práticas do Microcontrolador 8051. Editora Eriça, 12ª Edição 2004.

NICOLOSI, Denys E. C., Microcontrolador 8051. Editora Eriça, 5ª Edição 2004.

Notas



Aplicações com Microcontroladores

Módulo 8

Apresentação

Este módulo tem carácter teórico-prático, por isso deverá decorrer em parte em ambiente laboratorial de forma a permitir aos alunos verificarem e comprovarem os conhecimentos teóricos adquiridos sobre a estrutura e operação de um microcontrolador.

Esta disciplina tem como intenção tornar o aluno apto a compreender a linguagem e as técnicas utilizadas, possibilitando assim um melhor aproveitamento na sequência dos estudos desta e das outras disciplinas técnicas e também na comunicação adequada com os profissionais da área.

Introdução

A abordagem deste módulo sobre aplicações com microcontroladores leva-nos a uma melhor compreensão do funcionamento de vários tipos de aparelhos, que incorporam circuitos que utilizam estas características, existentes no mercado assim como a melhor escolha deste tipo de equipamentos para que se ajuste às crescentes evoluções disponíveis pelas diversas marcas.

Este módulo requer um conhecimento básico de matemática e programação.

Objetivos de aprendizagem

- Controlar um display através do programa do microcontrolador.
- Elaborar circuitos e programas adequados para controlar motores passo-a-passo.
- Implementar sistemas de aquisição de dados e controlo digital.
- Elaborar programas para controlo da velocidade de motores de corrente contínua por PWM.

Âmbito de conteúdos

- Visualização de dados.
- Controlo de motores passo-a-passo.
- Aquisição de dados.
- Controlo de motores DC.

Hardware dos Microcontroladores

Introdução

Características principais comuns aos microcontroladores da família intel 8051 mcs-51:

- Família de microcontroladores mais usada atualmente.
- CPU de 8 bits otimizada para aplicações de controlo.
- Clock típico de 12MHz (valor usado em aplicações gerais existindo também versões mais rápidas).
- Capacidade de 64 Kbytes de memória de programa (ROM) e 64 Kbytes de memória de dados (RAM).
- 4 Kbytes de memória de programa interna (ROM interna).
- RAM interna com 128 bytes (há versões com capacidades superiores).
- 4 Portas de I/O de 8 bits cada, com bits individualmente endereçáveis.
- Interrupções mascaráveis em dois níveis de prioridades (três internas e duas externas).
- 2 Temporizadores / contadores internos de 16 bits programáveis.
- Oscilador de clock interno.
- Canal de comunicação serial.
- Capacidade de execução de complexas operações aritméticas e lógicas (multiplicação, divisão, permuta e deslocamento de bits etc).
- Família com grande variedade de CPU's, com versões diferenciando-se na especialização, porém apresentando mesma arquitetura interna básica.
- Fornecido por diferentes fabricantes que personalizam o seu produto mantendo a compatibilidade com as versões originais.

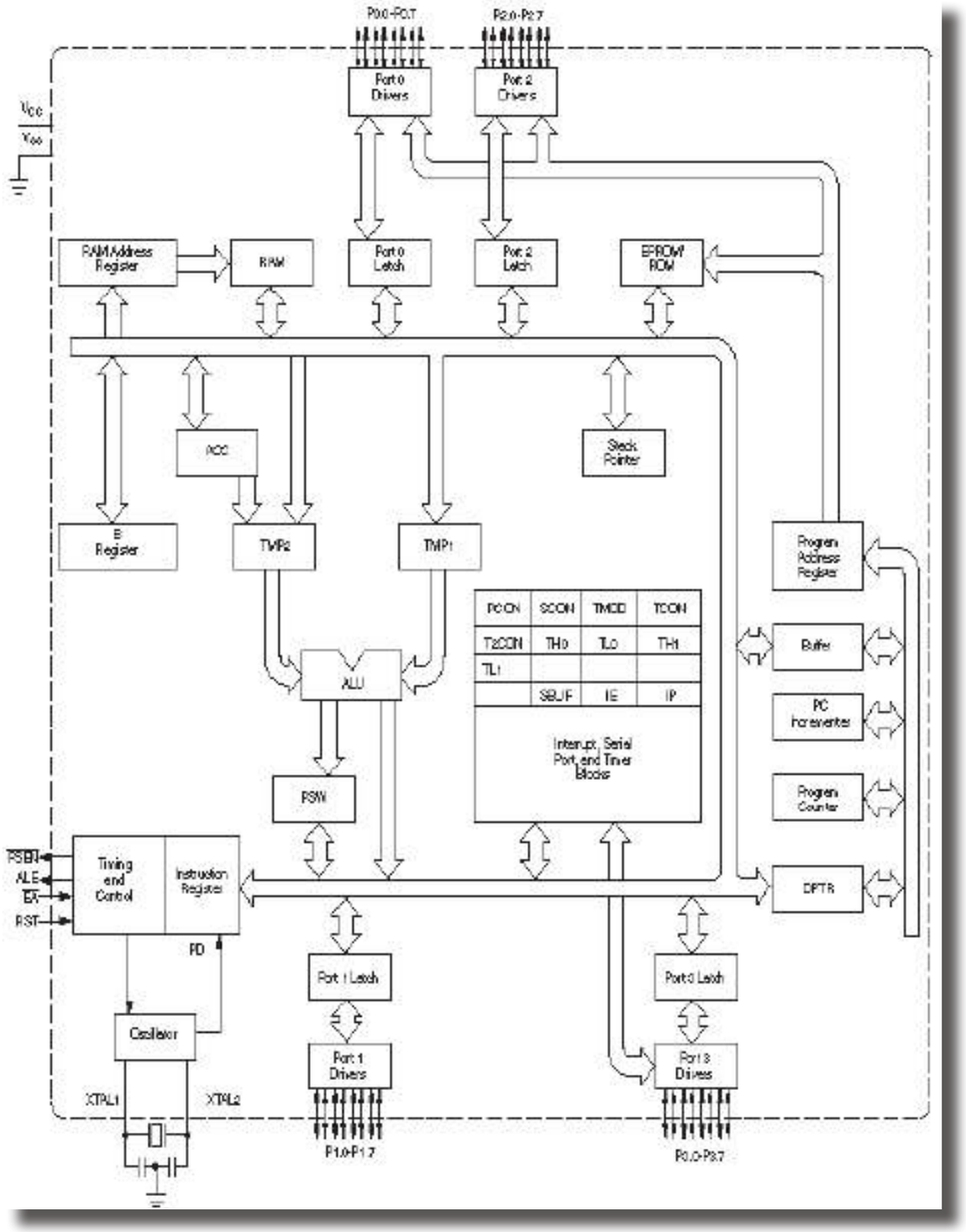
Quadro comparativo entre algumas versões da família 8051 e outras similares da Intel

Código do dispositivo	Versão sem ROM	Versão com EPROM	RAM Interna bytes	ROM Interna bytes	Portas E/S de 8 bits	Linhas de I/O	Timers /Conta-dores	UART	Canais de DMA	Modos de baixo consumo e Idle
8048AH	8035AHL	8748H	64	1 K		27	1	-	-	-
8049AH	8039AHL	8749H	128	2 K		27	1	-	-	-
8050AH	8040AHL	-	256	4 K		27	1	-	-	-
8051	8031	-	128	4 K	4	32	2	sim	-	-
8051AH	8031AH	8751AH	128	4 K	4	32	2	sim	-	-
80C51BH	80C31BH	87C51	128	4 K	4	32	2	sim	-	sim
8052AH	8032AH	8752BH	256	8 K	4	32	3	sim	-	sim
80C52	80C32	-	256	8K	4	32	3	sim	-	sim
83C51FA	80C51FA	87C51FA	256	8K	4	32	3	sim	-	sim
83C51FB	80C51FB	87C51FB	256	16K	4	32	3	sim	-	sim
83C152JA	80C152JA	-	256	8K	5	40	2	sim	2	sim
-	80C152JB	-	256	-	7	56	2	sim	2	sim
83C152JC	80C152JC	-	256	8K	5	40	2	sim	2	sim
-	80C152JD	-	256	-	7	56	2	sim	2	sim
83C452	80C452	87C452P	256	8K	5	40	2	Sim	-	sim

Observações:

- A letra “C” usada em alguns códigos indica que a tecnologia usada no projeto é a CHMOS, que apresenta um menor consumo de potência que o da tecnologia HMOS (dispositivos sem letra nos códigos).
- Dispositivos HMOS e CHMOS são intercambiáveis.
- O par de letras “BH” refere-se ao projeto interno do chip.
- Os modos “Idle” (preguiçoso, ocioso) e “de baixa potência” são programados por software e tem o objetivo de reduzir o consumo, neles a CPU é desligada, enquanto a RAM e outros periféricos internos continuam a operar.

Arquitetura interna da família 8051



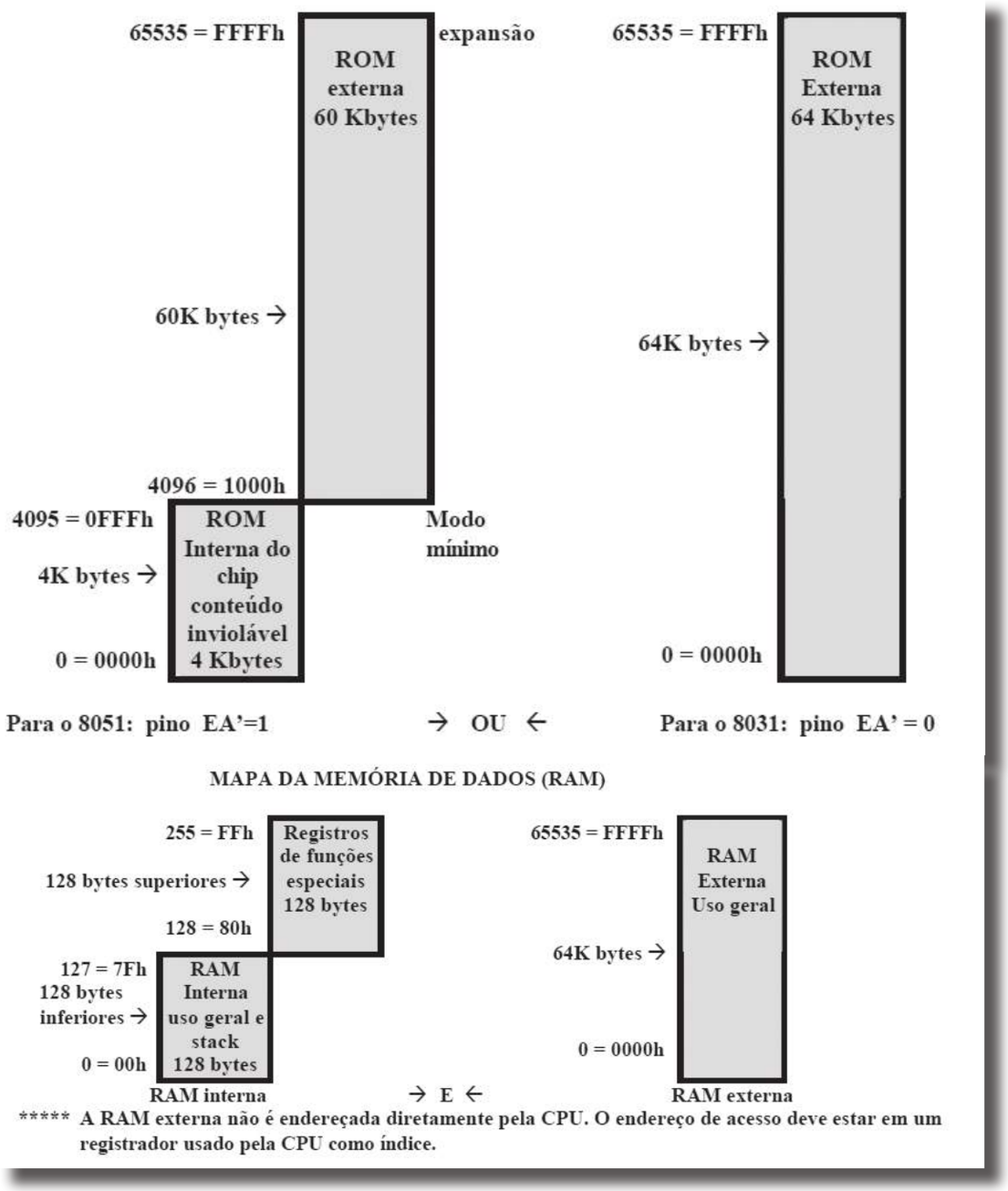
Descrição da pinagem do 8051

Número dos Pinos	Nome	Descrição resumida de sua função
1 a 8	P1.0 a P1.7 Port 1	Porta de I/O número 1
9	RST/ VPD	"Reset" do sistema (é necessário a aplicação de um nível alto TTL, durante 2 ou mais ciclos de máquina)
10 a 17	P3.0 a P3.7 Port 3	Porta de I/O número 3 Possibilita também funções especiais relacionadas ao Canal Serial, Interrupções e "Timer/Counter"
18	XTAL 2	Ligação do cristal oscilador
19	XTAL 1	Ligação do cristal oscilador
20	Vss	Terra
21 a 28	P2.0 a P2.7 Port 2 ou A8 a A15	Porta de I/O número 2 Saída do byte mais significativo do endereço, para memória externa.
29	PSEN'	Program Store Enable "Strobe" da memória de programa externa. Quando o sistema lê instruções ou operandos na memória externa, vai para nível zero e não é ativado (nível 1) durante busca na memória interna de programa.
30	ALE / PROG	Address Latch Enable Saída para habilitar o "latch" de endereços. Serve para separar, a parte menos significativa do endereço, dos dados, na aplicação de memória externa. Entrada do pulso de programação durante a gravação da EPROM.
31	EA' / Vpp	External Access Enable – Programming Supply Voltage Entrada de seleção da memória de programa. Quando em nível zero, a CPU trabalha apenas com a memória de programa externa.. Se em nível lógico 1, a CPU executa instruções da memória de programa interna, desde que o PC seja menor que 4096. Este pino recebe +21 volts durante a programação da ROM interna. Recebe +21V durante a programação da EPROM
32 a 39	P0.0 a P0.7 ou AD0 a AD7	Porta de I/O número 0 Fornece o byte menos significativo de endereço multiplexado com os dados.
40	Vcc	+ 5 volts

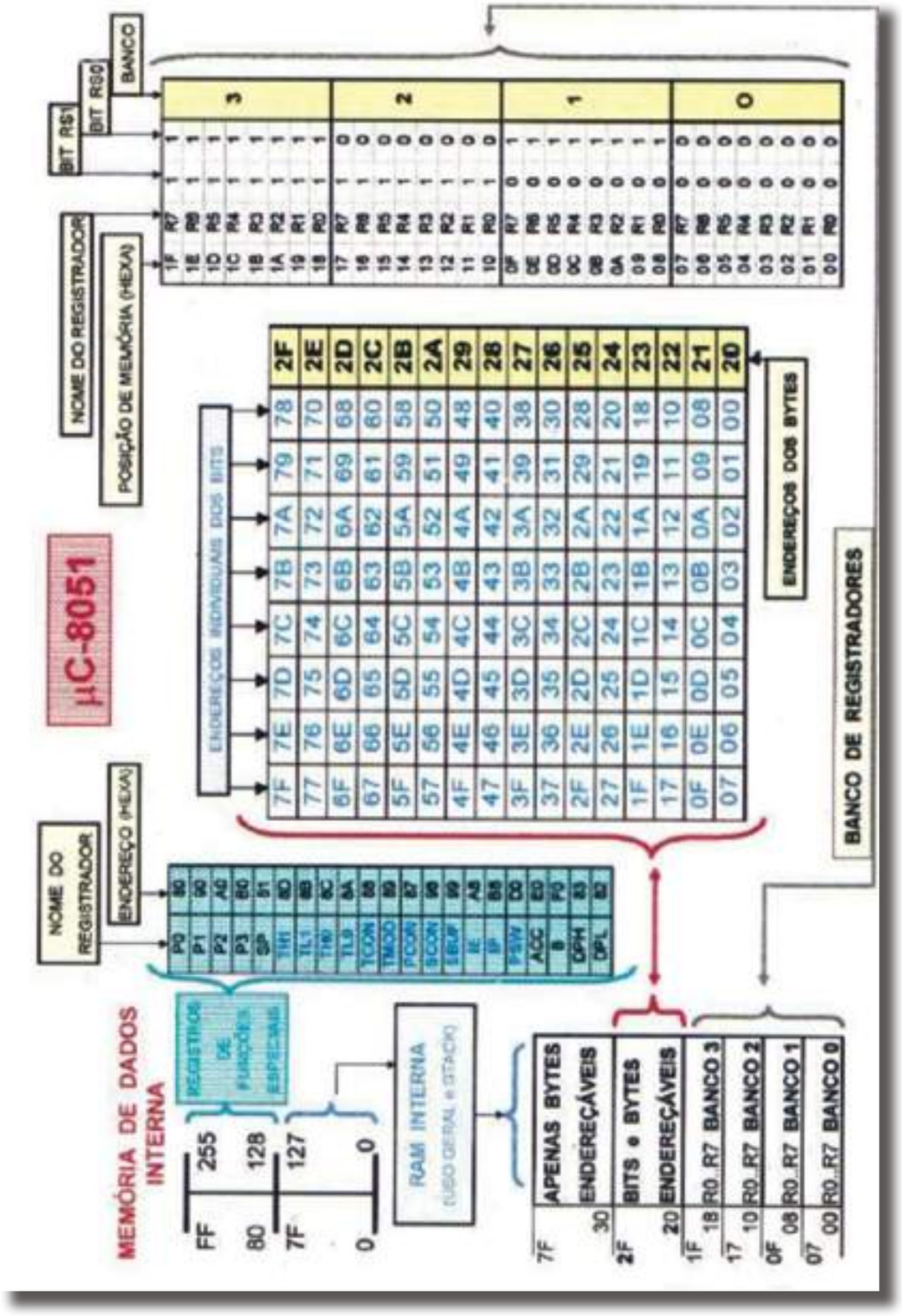
Funções especiais dos pinos da porta 3

Nomes do Pino	Descrição resumida de sua função
P3.0 = RXD/data	Receptor da porta serial assíncrona ou entrada e saída de dados síncronos (expansão de I/O pela porta serial).
P3.1 = TXD/ clock	Saída de transmissão da porta serial assíncrona, ou saída de clock para os registradores de deslocamento externos (expansão de I/O pela porta serial).
P3.2 = INT0'	Interrupção externa número 0, ou Bit de controle para o "timer/counter" 0.
P3.3 = INT1'	Interrupção externa número 1, ou Bit de controle para o "timer/counter" 1.
P3.4 = T0	Entrada externa para o "timer/counter" 0.
P3.5 = T1	Entrada externa para o "timer/counter" 1.
P3.6 = WR'	"Strobe" (sincronismo) de escrita na memória de dados externa (escrita na memória RAM).
P3.7 = RD'	Strobe (sincronismo) de leitura da memória de dados externa (leitura da memória RAM).

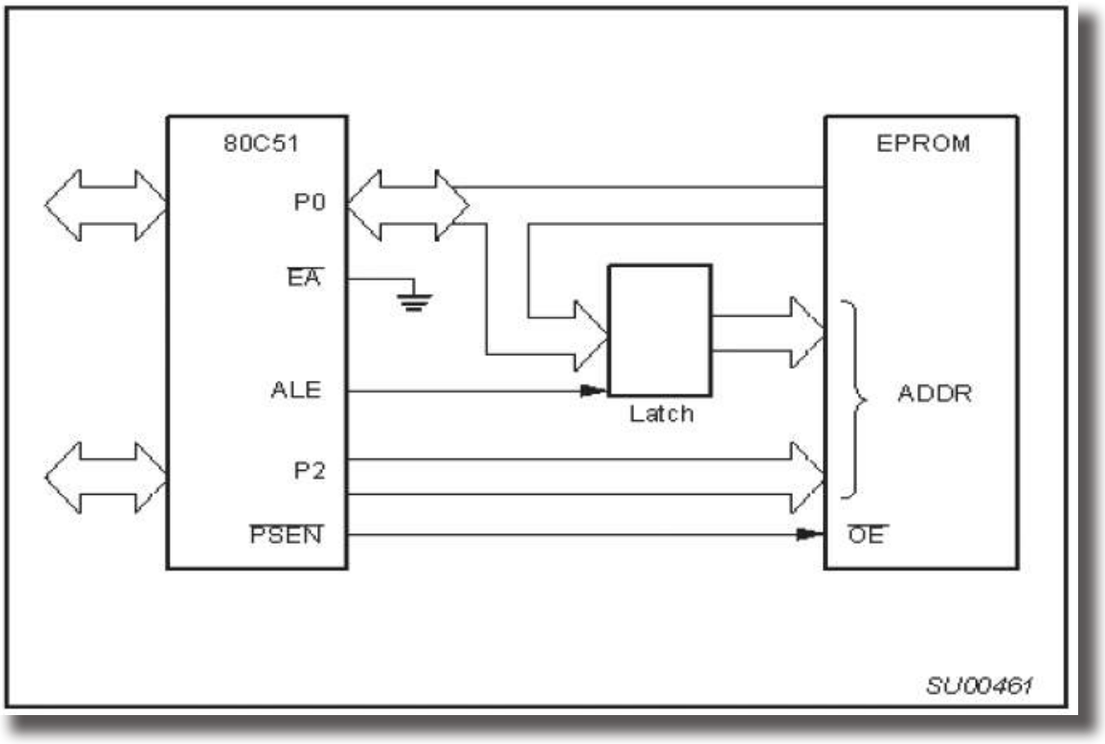
Organização da memória da família 8051 (e equivalentes)



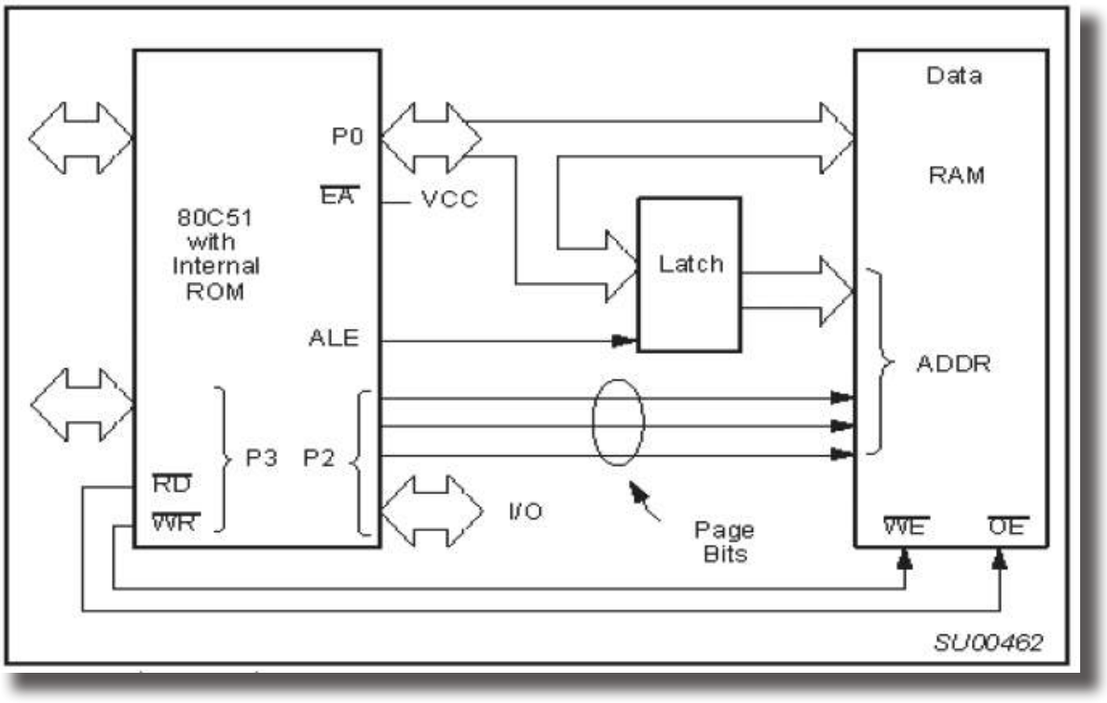
Estrutura da memória Ram interna



Ligação com a memória externa



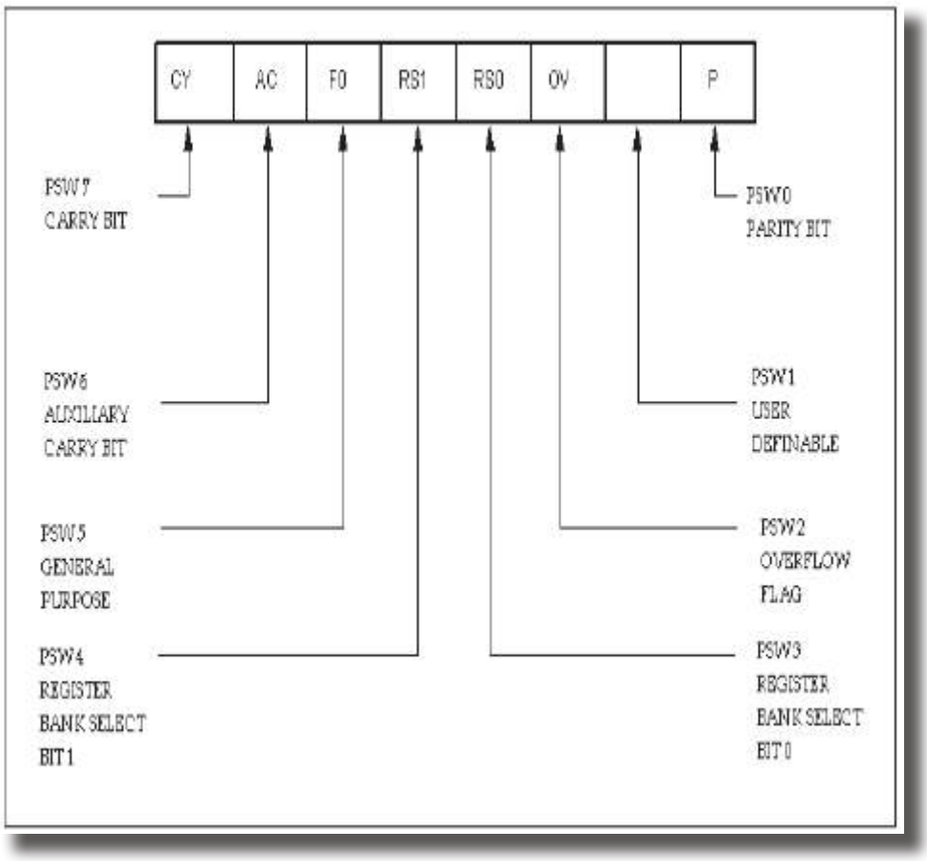
Execução a partir de memória externa.



Acesso à memória de dados externa.

Program Status Word (PSW) - endereço D0h

Este byte, localizado no espaço SFR (Special Status Word – RAM interna de 80h a FFh), possui alguns bits de status que refletem o estado da CPU. Mostrado na figura abaixo, contém as flags: Carry, Auxiliay Carry, Overflow, Paridade, dois bits (RS1 e RS0) de seleção de banco de registradores e dois bits de status definidos pelo utilizador.



Registrador PSW (Program Status Word)

Flag		Descrição	
C = PSW.7	Flag carry	C = 1 indica o transporte no bit 7 (vai 1 ou vem 1), C = 0 caso contrário	
AC = PSW.6	Flag auxiliar carry	C = 1 indica o transporte entre os bits 3 e 4 (vai 1 ou vem 1), C = 0 caso contr.	
F0 = PSW.5	Flag de uso geral	Pode-se setar ou resetar esta flag, indicando o status da condição escolhida	
RS1 = PSW.4 RS0 = PSW.3	Seleção do banco de registradores	RS1 RS0 = 00 → banco 0	RS1 RS0 = 10 → banco 2
		RS1 RS0 = 01 → banco 1	RS1 RS0 = 11 → banco 3
OV = PSW.2	Flag de overflow	OV = 1 indica uma condição de erro, o resultado não pode ser representado como um nº sinalizado (ex. soma de nºs negativos resultando em positivo)	
P = PSW.0	Flag de paridade	P = 0 p/ paridade par do acumulador e P = 1 p/ paridade impar do acumulador	

RESET – Principais características:

- Ativo quando o pino RST (9) permanecer em “nível 1” por 2 ou mais ciclos de máquina.
- RESET interno:
  - PC, Acumulador, B, Flags, DPTR, registros dos temporizadores / contadores são colocados a zero.
  - SP - 07h
  - SBUF (buffer serial) estará com conteúdo indeterminado e o registro de controlo da porta serial (SCON) são colocados a zero.
  - PCON terá apenas o seu bit mais significativo é colocado a zero.
  - IE e IP (registros de controle de interrupção) terão xxx00000.
  - As portas P0 a P3 terão o valor FFh. (durante o RESET, o nível nos pinos é indeterminado, indo a “nível 1” após a execução da rotina interna de RESET assim, o hardware externo, dependendo da aplicação, deve prever essa situação, evitando o acionamento indesejado de algum periférico.
  - A RAM interna não é afetada pelo RESET “forçado”, sendo que após o “power-up” seu valor é aleatório.

Interrupção

Existem três fontes de interrupção:

- A interrupção por software (instrução).
- A interrupção solicitada por um periférico externo.
- A interrupção solicitada por um periférico interno (Timer/Counter ou Porta Serial).

Vantagem da interrupção:

- Simplificação do hardware, pois o sistema não necessita ficar monitorando o funcionamento de alguns dispositivos externos.

Características das Interrupções:

- Mascaramento:
  - Possibilidade do sistema atender ou não uma solicitação de interrupção (mascaráveis).
  - Existem sistemas com interrupções não mascaráveis.

- Prioridades:
  - Para um sistema que tenha mais de uma interrupção, existe uma tabela de prioridades, que determina qual deve ser atendida primeiramente no caso de solicitações coincidentes.

- Interrupção de vetor:
  - Possuem o "Vetor de Interrupção" (endereço de início da interrupção) fixo, não sendo definido pelo utilizador (família 8051).

- Interrupção não devido a um vetor:
  - Os endereços de tratamento da interrupção são escolhidos pelo programador (outras famílias).
  - OBS: O significado das expressões "interrupção de vetor" e "interrupção não devido a um vetor" pode ser exatamente contrário ao descrito, dependendo da convenção do fabricante da família de microcontroladores usada.

- Reconhecimento da interrupção:
  - Pelo nível (1 ou 0).
  - Pela borda de subida ou de descida.
  - Pela soma de borda (subida ou descida) e um nível correspondente.

As interrupções da família 8051 (e equivalentes)

Essa família de microcontroladores apresenta 5 / 6 formas de interrupção:

- Pela interrupção externa INT0' (maior prioridade).
- Pelo timer/counter interno TIMER 0.
- Pela interrupção externa INT1'.
- Pelo timer/counter interno TIMER 1.
- Pelo canal de comunicação serial (menor prioridade).
- Pelo timer/counter interno TIMER 2 (somente para o 8032 / 52).

OBSERVAÇÕES:

- Todas as interrupções podem ser habilitadas ou não (exceto o RESET).
- Se o pedido da interrupção que ocorrer for de nível menor ou igual (mesmo número) ao da que já está sendo atendida, este ficará aguardando o término da mesma para ser executado.
- No caso de uma interrupção de prioridade menor estar em andamento, esta poderá ser interrompida por uma de prioridade maior, que ao seu término possibilita que a primeira seja concluída.

Endereços de desvio das interrupções

Mapa da memória de programa.

ENDEREÇO	NOME DA INTERRUPÇÃO	Intervalo
0000h	Reset	3 bytes
...		
0002h		
0003h	INT0	8 bytes
...		
000Ah		
000Bh	Timer/Counter 0	8 bytes
...		
0012h		
0013h	INT1	8 bytes
...		
001Ah		
001Bh	Timer/Counter 1	8 bytes
...		
0022h		
0023h	Canal Serial	8 bytes
...		
002Ah		
002Bh	Timer/Counter 2 Somente para 8032 / 52	8 bytes
...		
0032h		
0033h		

Registradores de controlo das interrupções

O 8051 possui dois registradores, na Região conhecida como SFR (Registradores de Funções Especiais) para controle das interrupções.

IE (Interrupt Enable) – Endereço A8h.

Tem por função indicar quais interrupções estão habilitadas.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
EA	X	X	ES	ET1	EX1	ET0	EX0

EA (Enable All):

- p/ = 0, desabilita todas as interrupções.
- p/ = 1, permite selecionar qual interrupção será habilitada, em função dos bits de controle individuais (abaixo).
-

ES (Enable Serial):

- p/ = 0, inibe a interrupção solicitada pelo canal serial, independentemente do valor de EA.
- p/ = 1, liberta a interrupção solicitada pelo canal serial, se EA = 1.

ET1 (Enable Timer 1):

- p/ = 0, inibe a interrupção solicitada pelo Timer/Counter 1, independentemente do valor de EA.
- p/ = 1, liberta a interrupção solicitada pelo Timer/Counter 1, se EA = 1.

ET0 (Enable Timer 0):

- p/ = 0, inibe a interrupção solicitada pelo Timer/Counter 0, independentemente do valor de EA.
- p/ = 1, liberta a interrupção solicitada pelo Timer/Counter 0, se EA = 1.

EX1 (Enable External 1):

- p/ = 0, inibe a interrupção solicitada pelo dispositivo externo ligado no pino INT1', independentemente do valor de EA.
- p/ = 1, liberta a interrupção solicitada pelo dispositivo externo ligado no pino INT1', se EA = 1.

EX0 (Enable External 0):

- p/ = 0, inibe a interrupção solicitada pelo dispositivo externo ligado no pino INT0', independentemente do valor de EA.
- p/ = 1, libera a interrupção solicitada pelo dispositivo externo ligado no pino INT0', se EA = 1.

IP (Interrupt Priority) – Endereço B8h

Tem por função fixar qual nível de prioridade das interrupções, sendo assim possível alterar a prioridade de atendimento de uma interrupção durante o processamento.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
X	X	X	PS	PT1	PX1	PT0	PX0

PS (Priority Serial):

- p/ = 0, indica prioridade baixa para a interrupção solicitada pelo canal serial.
- p/ = 1, indica prioridade alta para esta interrupção, se a mesma estiver habilitada.

PT1 (Priority Timer 1):

- p/ = 0, indica prioridade baixa para a interrupção solicitada pelo Temporizador/ Contador 1.
- p/ = 1, indica prioridade alta para esta interrupção, se a mesma estiver habilitada.

PT0 (Priority Timer 0):

- p/ = 0, indica prioridade baixa para a interrupção solicitada pelo Temporizador/ Contador 0.
- p/ = 1, indica prioridade alta para esta interrupção, se a mesma estiver habilitada.

PX1 (Priority External 1):

- p/ = 0, indica prioridade baixa para a interrupção solicitada pelo dispositivo externo ligado no pino INT1'.
- p/ = 1, indica prioridade alta para esta interrupção, se a mesma estiver habilitada.

PX0 (Priority External 0):

- p/ = 0, indica prioridade baixa para a interrupção solicitada pelo dispositivo externo ligado no pino INT0'.
- p/ = 1, indica prioridade alta para esta interrupção, se a mesma estiver habilitada.

Ajuste da forma de reconhecimento das interrupções externas

É possível ajustar as interrupções externas para serem detetadas pela transição de "1" para "0" ou pelo nível "0".

TCON (Timer Control) – Endereço 88h

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

IT1: Bit para a escolha da forma de reconhecimento do INT1

- p/ = 0, indica que será aceita a interrupção apenas pelo nível "0" presente no pino INT1'.
- p/ = 1, indica que será aceita a interrupção na transição de "1" para "0" no nível desse pino'.

IT0: Bit para a escolha da forma de reconhecimento do INT0

- p/ = 0, indica que será aceita a interrupção apenas pelo nível "0" presente no pino INT0'.

- p/ = 1, indica que será aceita a interrupção na transição de “1” para “0” no nível desse pino’.

IE1: Bit para o hardware de controlo da interrupção INT1

- É colocado a 1 pelo hardware interno quando for detetada uma transição de “1” para “0” no pino INT1’.
- Tem por função sinalizar internamente o pedido da interrupção. É efetuado o reset logo que a interrupção é atendida.

IE0: Bit para o hardware de controlo da interrupção INT0

- É colocado a 1 pelo hardware interno quando for detetada uma transição de “1” para “0” no pino INT0’.
- Tem por função sinalizar internamente o pedido da interrupção. É efetuado o reset logo que a interrupção é atendida.

Temporizadores e contadores

O 8051 possui 2 Temporizadores / Contadores, controláveis por programa, que podem operar de maneira totalmente independente dos demais sistemas do chip, podendo ser habilitados ou não por software ou hardware (pelos registros de controle ou pinos de interrupção).

TCON (Timer Control) – Endereço 88h

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Obs.:

Os bits IE1, IT1, IE0 e ITO são usados no controle das interrupções externas, sendo referentes ao ajuste da forma de reconhecimento destas.

TF1:

- Sempre que ocorrer um overflow no T/C 1, este bit será colocado a 1, gerando um pedido de interrupção do T/C 1. É efetuado o reset pelo hardware interno ao final da rotina de interrupção.

TR1:

- É colocado a 1 pelo software para ligar T/C 1. É efetuado o reset para desligar o T/C (parar contagem).

TF0:

- Sempre que ocorrer um overflow no T/C 0, este bit será colocado a 1, gerando um pedido de interrupção do T/C 0. É efetuado o reset pelo hardware interno ao final da rotina de interrupção.

TR0:

· É colocado a 1 pelo software para ligar T/C 0. É efetuado o reset para desligar o T/C (parar contagem).

TMOD (Timer Mode) – Endereço 89h

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Gate.1	C/T' .1	M 1 .1	M 0.1	Gate .0	C/T'.0	M 1.0	M 0.0

Gate.1 e Gate.0:

- Escolha da forma de como os T/C serão habilitados.

Exemplo:

- p/ Gate.0 = 0 - T/C 0, será habilitado (contando) fazendo-se TR0 = 1 (no registrador TCON).
- p/ Gate.0 = 1 - T/C 0, será habilitado (contando) somente p/ TR0 = 1 e o pino INT 0' = 0.

Exemplo de aplicação:

Determinação da largura de um impulso, colocando-se a entrada do sinal externo no pino de interrupção correspondente e fazendo com que T/C funcione com sinal interno. Desta maneira somente haverá contagem quando o sinal externo for = 1. O software calculará o tempo decorrido entre dois pulsos do sinal aplicado.

C/T'.1 e C/T'.0:

C/T'.0	Função escolhida para Timer / Counter 0
= 1	Contagem (para sinal externo)
= 0	Temporização (sinal será interno)

C/T'.1	Função escolhida para Timer / Counter 1
= 1	Contagem (para sinal externo)
= 0	Temporização (sinal será interno)

OBS:

Para qualquer um dos T/C, se o sinal for interno (timer), a frequência será a de clock dividida por 12 e o pino de entrada correspondente fica disponível.

Modos de operação dos t/c's.

Escolha de um dos possíveis modos de operação para cada T/C.

M1.1 e M0.1

M1.0 e M0.0

MODO 0

- Contador ou Temporizador de 8 bits, com divisor de frequência de até 32 (freqclock / 32).
- Para selecionar faz-se: M1.x = 0 e M0.x = 0
- Os registradores TL0 e TL1 servem como divisores de 5 bits (até 32).

TH0 ou TH1 | TL0 ou TL1

Contagem (bits 15...8) | Ignorar (bits 7...5) Prescaler (bits 4...0)

- Registrador TH0 (endereço 8Ch) ou registrador TH1 (endereço 8Dh) recebe o valor inicial da contagem (valor escrito pelo software, até FFh).
- Ao ocorre um overflow nesse registrador, o T/C em uso gera um pedido de interrupção interno, que poderá ser ou não aceito pela CPU.
- O valor presente nesse registrador pode ser lido pelo software a qualquer momento.
- O sinal de contagem (interno ou externo) será dividido pelo valor binário presente nos bits "0" a "4" do registrador TL0 (endereço 8Ah) ou TL1 (endereço 8Bh), no T/C usado.
- Os bits "5"a "7" devem ser ignorados em caso de leitura.

Exemplo:

É desejada uma contagem com as seguintes características:

De 9Bh até FFh e que a cada 20 pulsos aplicados na entrada externa do T/C 0 (pino T0), este incremente seu registro de uma unidade.

Assim temos: 9Bh – FFh = 64h = 100 contagens em decimal, sendo desejado então que a cada 100 contagens internas uma interrupção seja gerada e, possibilitar assim que um determinado controle do sistema seja feito pela CPU. Dessa forma temos uma interrupção a cada 2000 sinais externos.

Isso é conseguido programando-se T/C 0 como contador no modo 0, carregando-se TH0 com 9Bh e TL0 com 14h (20 decimal).

A rotina de interrupção, além do tratamento desta, deverá escrever o valor inicial em TH0 (máx. contagem = 255 x 32).

MODO 1

- Contador ou Temporizador de 16 bits.
- Para selecionar faz-se: M1.x = 0 e M0.x = 1.

- Temos um par de registradores TH0 e TL0 ou TH1 e TL1 escolhido para efetuar a contagem.
- É possível contagem de até FFFFh (65535 em decimal) com o valor inicial programável por software.
- Ao correr um overflow, o sistema recebe um pedido de interrupção interna que poderá ou não ser aceite.

TH0 ou TH1 | TL0 ou TL1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

MODO 2

- Contador ou Temporizador de 8 bits com recarga automática.
- Para selecionar faz-se: M1.x = 1 e M0.x = 0.
- Nos registradores TL0 e TL1 ocorre a contagem.
- Nos registradores TH0 e TH1 temos os valores que serão carregados automaticamente nos registradores TL0 e TL1, sempre que ocorrer um overflow (interrupção), prosseguindo o sistema sob o comando do sinal externo (contador) ou interno (temporizador).
- Com este modo, temos um sistema no qual não é necessário escrever novamente via software o valor a ser contado.
- Todos os registradores podem ser alterados a qualquer momento pelo software, resultando em uma grande flexibilidade no trabalho com temporizações e contagens.
- O pedido de interrupção interna poderá ou não ser aceito pela CPU.

TH0 ou TH1 | TL0 ou TL1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

MODO 3

- Contador de eventos de 8 bits e temporizador de 8 bits.
- Selecionado fazendo-se: M1.x = 1 e M0.x = 1
- Neste modo T/C 1 para sua operação e fica inerte, sem receber pulsos de contagem.
- Para T/C 0 temos dois sistemas de 8 bits, um em TH0 e outro em TL0.
- O T/C agora formado por TL0 será controlado pelos bits TR0 e TF0 do registrador TCON.
- O T/C agora formado em TH0 será controlado pelos bits TR1 e TF1 do registrador TCON.
- O T/C 0 será habilitado pelo bit de controlo do T/C1 (TR1) e ao ocorrer um overflow de TH0, é o bit TF1, que será colocado a 1 e não o TF0.
- Neste modo, o overflow em TH0 acionará o flag de requisição de interrupção referente ao T/C 1 e o overflow de TL0 acionará o flag de requisição de interrupção de T/C 0.
- Os pedidos de interrupções internas poderão ou não serem aceitos pela CPU.

TH0 ou TH1 | TL0 ou TL1

Apenas temporizador de 8 bits | Temporizador / contador de 8 bits

Ativa a interrupção do T/C 1 | Ativa interrupção do T/C 0

A Comunicação Serie

Modo síncrono de comunicação:

- Necessita de um sincronismo entre os sistemas de comunicação.
- Sincronismo é obtido através de um conjunto de bits, denominado “bits de sincronismo”, que ao serem recebidos pelo elemento recetor, ajustam o seu relógio interno para possibilitar o recebimento de outro conjunto de bits referentes aos “dados”.

- O transmissor envia um outro conjunto de bits chamado “bits de parada”.
- Esses bits de parada podem conter ou não, informações sobre a confirmação do recebimento dos dados.

Caracter de sincronismo	bits de dados	bits de dados	caracter de final de transmissão

6	5	4	3	2	1	0	6	5	4	3	2	1	0	6	5	4	3	2	1	0	6	5	4	3	2	1	0
0	0	1	0	1	1	0	x	x	x	x	x	x	x	x	x	x	X	x	x	x	0	0	0	0	1	0	0

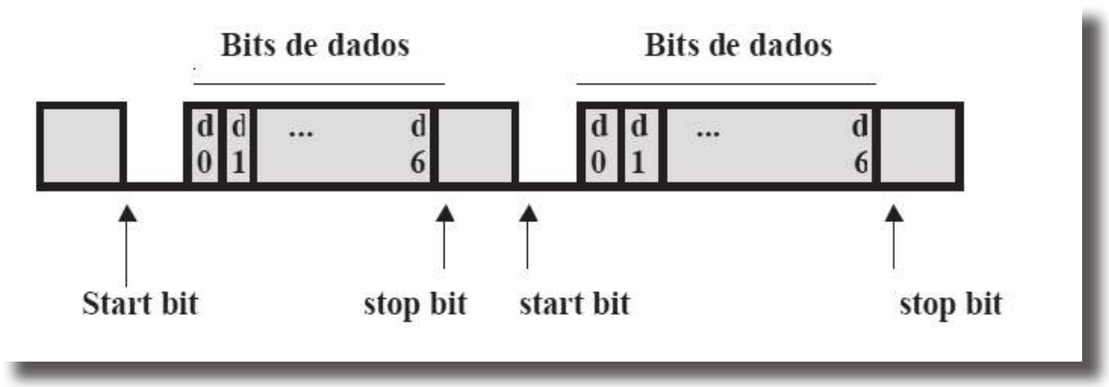
OBS:

16h = SYN = SYNcronism idle (Código ASCII)

04h = EOT = End Of Transmission (Código ASCII)

Modo assíncrono de comunicação

- Não existe a necessidade de caracteres de sincronismo
- Para cada conjunto de bits de dados transmitidos temos também transmitido um bit de início de transmissão (start bit) e um bit de final de transmissão (stop bit).
- O start bit é reconhecido pela transição de “1” para “0”, na linha.
- Após o start bit, o sistema efetua periodicamente uma varredura na linha, associando os níveis lógicos encontrados aos bits dos dados de entrada.
- Ao reconhecer o sétimo bit, o sistema fica esperando o stop bit.
- O stop bit é a transição de “0” para “1”, ou a permanência em “1” se a linha já se encontrava assim.
- Deve-se garantir que o transmissor e o recetor operem com a mesma taxa de comunicação.



OBS:

Os sinais de temporização e controlo, utilizados em cada modo são gerados por um hardware específico para comunicação serial, sendo transparentes para o utilizador.

Canais simplex, half-duplex e full-duplex

Canal SIMPLEX

- Este modo é constituído pela interligação de dispositivos no qual temos um elemento que apenas transmite e outro que apenas recebe.

Canal HALF-DUPLEX, ou SEMIDUPLEX

- Neste modo de comunicação, temos elementos que recebem e transmitem dados, embora as duas operações não possam ocorrer simultaneamente.

Canal FULL-DUPLEX ou simplesmente DUPLEX

- Consiste em um modo pelo qual os sistemas podem transmitir e receber dados simultaneamente.

A comunicação serie no 8051

- No 8051, a interface serie é do tipo Full-Duplex.
- O controlo da transmissão e receção de dados é feito por um registro denominado SBUF (serial Buffer – Endereço 99h).
- Uma escrita no SBUF implica em automática transmissão deste dado.

- Um dado ao "chegar" no pino correspondente será automaticamente recebido pelo sistema independentemente do usuário (p/ o canal serie habilitado e ajustado).
- Existem dois registradores SBUF, um destinado para a receção e outro para a transmissão de dados.
- O reconhecimento é feito pelo sistema através das instruções que acessão o mesmo, assim para uma instrução de escrita o registro de transmissão será alterado e para uma instrução de leitura o registro de receção será lido.
- A transmissão inicia-se assim que o dado é escrito no SBUF.
- A receção é controlada pelo registro SCON.

SCON (Serial Control) – Endereço 98h

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SM0	SM1	SM2	REN	TB8	RB8	T1	R1

SM0 e SM1 (Serial Mode):

- Estes dois bits controlam o funcionamento do canal série.

SM0	SM1	Modo de Funcionamento	Taxa de transmissão
0	0	0	Fclock/12
0	1	1	Variável
0	0	2	Fclock/32 ou Fclock/64
1	1	3	Variável

SM2

- No modo 0, não impõe qualquer efeito no funcionamento do canal série, devendo permanecer em 0.
- No modo 1, não produzirá pedido de interrupção se estiver no nível 1 e se o stop bit recebido for ilegal.
- Nos modos 2 e 3, habilita a comunicação entre várias cpu's 8051 p/=1.

REN (Reception Enable):

- Se estiver set (=1), habilita a receção mal um start bit seja detetado.
- Se estiver reset (=0), desabilita a receção, e o pino RXD pode ser usado como I/O.

TB8

- Nos modos 2 e 3, indica o estado do nono bit a ser transmitido.
- Pode ser efetuado o set ou reset por software.

RB8

- Não é usado no modo 0.
- No modo 1 indica o estado do stop bit recebido, desde que SM2 seja igual a 0.
- Nos modos 2 e 3, indica o estado do nono bit de dados que foi recebido.

TI

- Flag de requisição de interrupção de transmissão.
- É set pelo hardware após a transmissão do oitavo bit de dados quando no modo 0 e nos outros no início do stop bit.
- Permite que o programa de transmissão seja independente do programa principal, pois a cada final de transmissão a interrupção do canal serial pode gerenciar todo o processo, evitando assim os processos de varredura de alguns sistemas.
- Deve ser efetuado o reset pelo software da rotina de interrupção para permitir novas interrupções.

RI

- Flag de requisição de interrupção da receção.
- É efetuado o set pelo hardware após a receção do oitavo bit de dados quando no modo 0 e nos outros modos, ao moio do tempo de receção do stop bit.
- Deve ser efetuado o reset pelo software da rotina de interrupção para permitir novas interrupções.

Software

Modos de endereçamento

A família 8051 possui os seguintes modos de endereçamento:

Obs.:

Para diferenciar os modos de endereçamento e o tratamento dado a variáveis, constantes e endereços é utilizada a seguinte notação:

- @ Indica “Indireto”
- # Indica valor “Constante”
- H indica que o valor é expresso em “Hexadecimal”
- B indica que o valor é expresso em “Binário”

Modo imediato

Neste modo, temos um endereço de 8 bits logo após a instrução, no qual será efetuada a operação, dessa forma com ele podemos aceder apenas as primeiras 256 posições de memória (RAM interna e SFR).

Exemplos:

- MOV A, 85H A ← (85H)
- MOV 86H, A (86H) ← A

Modo registrador

Neste modo, o nome do registrador a ser acedido está incluído no mnemônico (economia de um byte na memória de programa).

O registrador afetado (R0, R1, R2, R3, R4, R5, R6 ou R7) será o do banco de registradores selecionado no momento (RB0, RB1, RB2 ou RB3).

O endereçamento do banco de registradores é feito pelos bits RS2 e RS1, no registrador PSW, no instante de execução da instrução.

Exemplos:

- MOV (98H), R5 (98H) ← R5
- MOV R2, (88H) R2 ← (88H)

Modo indireto

Neste modo, o endereço sobre o qual a instrução atuará está indicada de forma indireta pelos registradores R0 ou R1, de qualquer um dos bancos de registradores, para endereços de 8 bits, ou indicado pelo registrador DPTR (Data Pointer, formado por DPH endereço 83H e DPL endereço 82H), para endereços de 16 bits.

Por esse modo de endereçamento é possível aceder a memória RAM interna ou externa.

Exemplos:

- MOV @R1, 43H (R1) ← (43H)
- MOVX A, @DPTR A ← (DPTR/RAM ext.)

Modo específico a registro

Neste modo, o registrador em questão, já faz parte do mnemônico da instrução.

Exemplos:

- DA A Ajuste decimal de A
- CLR A A ← 00H

Modo constante imediata

Este modo permite trabalhar com uma constante de forma direta.

Exemplos:

- MOV A, #44H A ← 44H
- MOV R7, #63H R7 ← 63H

Modo indexado

Neste modo, o endereço efetivo é a soma do Acumulador e de um registrador de 16 bits, que poderá ser o PC ou o DPTR.

Esse modo de endereçamento tem como principal aplicação a leitura de tabelas presentes na memória ROM, transferindo-as para a memória RAM e/ou realizando processamento sobre esses dados.

Exemplos:

- MOVC A, @A+DPTR — A ← (A+DPTR)
- JMP @A+DPTR — PC ← A+DPTR

Modo desvio indexado

Esse modo é usado por instruções de desvio condicionais, que somam ao valor do PC já ajustado, o dado de 8 bits presente no final da instrução.

Exemplo:

- JZ 15H — Desvio relativo se A=00

Exemplos da escrita de algumas instruções

Mnemônico	Descrição
ADD A, @R1	A ← A + (R1)
ADDC A, #58	A ← A + 58 + carry
SUBB A, R0	A ← A – R0 – carry
INC DPTR	DPTR ← DPTR + 1
MUL AB	B A ← A x B
DIV AB	A ← A/B B ← resto
ANL 58, #66	(58) ← (58) [and] #66
XRL A, @R1	A ← A [exclusive or] (R1)
RL A	d7←d6←d5←d4←d3←d2←d1←d0←d7
RRC A	cy→d7→d6→d5→d4→d3→d2→d1→d0→cy
SWAP A	em A: d7 d6 d5 d4 ←→ d3 d2 d1 d0
MOV A, 1Fh	A ← (1Fh)
MOV A, #3Ch	A ← #3Ch ;hexadecimal
MOV A, #10101110b	A ← #10101110b ;binário
MOV A, #78	A ← 78 ;decimal
MOV @R0, #0F2h	(R0) ← F2h { atenção para o caractere "0" }
MOV DPTR, #2000h	DPTR ← 2000h
MOVC A, @A+DPTR	A ← (A + DPTR) ;única forma de ler uma constante na memória de programa
MOVX A, @R0	A ← (R0) ;leitura da RAM externa
MOVX @DPTR, A	(DPTR) ← A ;escrita na RAM externa
XCH A, 28	A ←→ (28)
XCHD A, @R0	Para: A = 21h e (R0) = 4Dh, após a instrução temos: A = 2Dh e (R0) = 41h
POP 70	(70) ← (SP) SP ← SP + 1
PUSH 70	SP ← SP – 1 (SP) ← (70)
CLR EX1	bit EX1 ←0
CLR ABh	Bit ABh ← 0
CPL A	A ← complemento de A
ORL C, 1Ah	cy (or) (bit 1Ah)'
JNB P1.1, 2Ch	se P1.1 = 0 ;PC ← PC + 2Ch o programa desvia se P1.1 =1 ;programa prossegue

Tabelas de instruções da família 8051

Para diferenciar os modos de endereçamento, o tratamento dado a variáveis, constantes e endereços, registradores e seus conteúdos, além de valores numéricos genéricos de diferentes formatos e para diversas aplicações, é utilizada por alguns fabricantes a seguinte notação:

Rn - Indica registrador R0 a R7 genericamente, dependendo de “n”.

Ri - Indica o registrador R0 ou R1 genericamente, dependendo de “i”.

@ - Significa “endereçado pelo valor de ... “

#dado8 - Indica valor constante de 8 bits.

#dado16 - Indica valor constante d 16 bits.

Direto - Indica um endereço de memória de 8 bits (primeiras 256 posições internas ou externas).

rel - Indica que o endereço é relativo.

end2k - Indica endereço dentro de uma faixa de 2Kbytes.

end16 - Indica um endereço de 16 bits.

bit - Indica um bit individualmente endereçado.

Ciclos de máquina

Corresponde a um período de tempo equivalente a 12 períodos do clock, portanto para um microcontrolador com um cristal de 12M Hz, um ciclo de máquina tem a duração de 1 x 10-6 s.

Todas as instruções sempre são executadas em um número inteiro de ciclos de máquina, que na família 8051 são de 1, 2 ou 4 ciclos.

Instrução para transferência de dados

Mnemônico	Descrição	Nº de bytes	Nº de pulsos de clock
MOV A, Rn	Move o registrador para o acumulador	1	12
MOV A, direto	Move a memória para o acumulador	2	12
MOV A, @Ri	Move RAM endereçada por Ri para o acumulador	1	12
MOV A, #dado8	Move o dado para o acumulador	2	12
MOV Rn, A	Move o acumulador para o registrador	1	12
MOV Rn, direto	Move a memória para o registrador	2	12
MOV Rn, #dado8	Move o dado para o registrador	2	12
MOV direto, A	Move o acumulador para a memória	2	12
MOV direto, Rn	Move o registrador para a memória	2	24
MOV direto1, direto2	Move o conteúdo da memória direta2 para a memória direta1	3	24
MOV direto, @Ri	Move RAM endereçada por Ri para a memória	2	24
MOV direto, #dado8	Move o dado para a memória	3	24
MOV @Ri, A	Move o acumulador para a RAM endereçada por Ri	1	12
MOV @Ri, direto	Move a memória para a RAM endereçada por Ri	2	24
MOV @Ri, #dado8	Move o dado para a RAM indireta	2	12
MOV DPTR, #dado16	Move dado de 16 bits para o DPTR	3	24
MOVC A, @A + DPTR	Soma: A + DPTR obtendo um endereço de 16 bits na memória de programa e carrega acumulador com esta memória	1	24
MOVC A, @A + PC	Idem a anterior mas soma A + PC	1	24
MOVX A, @Ri	Move RAM externa (endereço de 8 bits) para o acumulador	1	24
MOVX A, @DPTR	Move RAM externa (endereço de 16 bits) para o acumulador	1	24
MOVX @Ri, A	Move acumulador para a RAM externa (endereço de 16 bits)	1	24
MOVX @DPTR, A	Move acumulador para a RAM externa (endereço de 16 bits)	1	24
PUSH direto	Incrementa o SP e então coloca a memória no stack	2	24
POP direto	Retira o dado do stack e o coloca na memória, depois decrementa o SP	2	24
XCH A, Rn	Troca os conteúdos do acumulador e do registrador	1	12
XCH A, direto	Troca a memória com o conteúdo do acumulador	2	12
XCH A, @Ri	Troca RAM indireta com o conteúdo do acumulador	1	12
XCHD A, @Ri	Troca o nibble menos significativo do acumulador e da RAM indireta entre si	1	12

Instruções aritméticas

Mnemônico	Descrição	Nº de bytes	Nº de pulsos de clock
ADD A, Rn	Soma o conteúdo de Rn ao acumulador. Resultado em A.	1	12
ADD A, direto	Soma o conteúdo da posição de memória ao acumulador. Resultado em A.	2	12
ADD A, @Ri	Soma o conteúdo da RAM endereçada por Ri ao acumulador. Resultado em A.	1	12
ADD A, #dado8	Soma o dado ao acumulador. Resultado em A.	2	12
ADDC A, Rn	Soma o conteúdo de Rn e o carry ao acumulador. Resultado em A.	1	12
ADDC A, direto	Soma o conteúdo da posição de memória e o carry ao acumulador. Resultado em A.	2	12
ADDC A, @Ri	Soma o conteúdo da RAM endereçada por Ri e o carry ao acumulador. Resultado em A.	1	12
ADDC A, #dado8	Soma o dado e o carry ao acumulador. Resultado em A.	2	12
SUBB A, Rn	Subtrai o conteúdo de Rn e o borrow do acumulador. Resultado em A.	1	12
SUBB A, direto	Subtrai o conteúdo da posição de memória e o borrow do acumulador. Resultado em A.	2	12
SUBB A, @Ri	Subtrai o conteúdo da RAM endereçada por Ri e o borrow do acumulador. Resultado em A.	1	12
SUBB A, #dado8	Subtrai o dado e o borrow do acumulador. Resultado em A.	2	12
INC A	Soma 1 ao acumulador.	1	12
INC Rn	Soma 1 ao conteúdo de Rn	1	12
INC direto	Soma 1 a posição de memória	2	12
INC @Ri	Soma 1 a RAM endereçada por Ri	1	12
DEC A	Subtrai 1 do acumulador.	1	12
DEC Rn	Subtrai 1 do conteúdo de Rn	1	12
DEC direto	Subtrai 1 da posição de memória	2	12
DEC @Ri	Subtrai 1 da RAM endereçada por Ri	1	12
INC DPTR	Soma 1 ao registro DPTR	1	24
MUL AB	Multiplica A e B, resultado em BA	1	48
DIV AB	Divide A por B, resultado em A, reto em B	1	48
DA A	Ajuste decimal do acumulador.	1	12

Instruções lógicas

Mnemónico	Descrição	Nº de bytes	Nº de pulsos de clock
ANL A, Rn	"E" entre o registrador e o acumulador. Resultado em A.	1	12
ANL A, direto	"E " entre a memória e o acumulador. Resultado em A.	2	12
ANL A, @Ri	"E" entre RAM indireta e o acumulador. Resultado em A.	1	12
ANL A, #dado8	"E" entre o dado e o acumulador. Resultado em A.	2	12
ANL direto, A	"E" entre acumulador e memória. Resultado na memória.	2	12
ANL direto, #dado8	"E " entre dado e memória. Resultado na memória	3	24
ORL A, Rn	"OU" entre o registrador e o acumulador. Resultado em A.	1	12
ORL A, direto	"OU " entre a memória e o acumulador. Resultado em A.	2	12
ORL A, @Ri	"OU" entre RAM indireta e o acumulador. Resultado em A.	1	12
ORL A, #dado8	"OU" entre o dado e o acumulador. Resultado em A.	2	12
ORL direto, A	"OU" entre acumulador e memória. Resultado na memória	2	12
ORL direto, #dado8	"OU " entre dado e memória. Resultado na memória	3	24
XRL A, Rn	"XOR" entre o registrador e o acumulador. Resultado em A.	1	12
XRL A, direto	"XOR " entre a memória e o acumulador. Resultado em A.	2	12
XRL A, @Ri	"XOR" entre RAM indireta e o acumulador. Resultado em A	1	12
XRL A, #dado8	"XOR"entre o dado e o acumulador.Resultado em A	2	12
XRL direto, A	"XOR" entre acumulador e memória. Resultado na memória	2	12
XRL direto, #dado8	"XOR " entre dado e memória.Resultado na memória	3	24
CRL A	Faz A = 0	1	12
CPL A	Inverte o estado de todos os bits do acumulador	1	12
RL A	Desloca o acumulador um bit a esquerda d7 ← d6 ← d5 ← d4 ← d3 ← d2 ← d1 ← d0 ← d7	1	12
RLC A	Desloca o acumulador um bit a esquerda pela carry CY ← d7 ← d6 ← d5 ← d4 ← d3 ← d2 ← d1 ← d0 ← CY	1	12
RR A	Desloca o acumulador um bit a direita d7 → d6 → d5 → d4 → d3 → d2 → d1 → d0 → d7	1	12
RRC A	Desloca o acumulador um bit a direita pela carry d7 → d6 → d5 → d4 → d3 → d2 → d1 → d0 → CY → d7	1	12
SWAP A	Troca os nibbles + e – significativos do acumulador	1	12

Instruções de desvio

Mnemônico	Descrição	Nº de bytes	Nº de pulsos de clock
ACALL end2K	Chama sub-rotina numa contida faixa de 2Kbytes da posição atual (dentro do espaço de –1024 a +1024 posições de memória).	2	24
LCALL end16	Chama sub-rotina em qualquer posição da memória de programa (até 64K bytes).	3	24
RET	Retorna da sub-rotina.	1	24
RETI	Retorna da interrupção.	1	24
AJMP end2K	Desvia para endereço dentro de uma faixa de 2Kbytes da posição atual (dentro do espaço de –1024 a +1024 posições de memória).	2	24
LJMP end16	Desvia para qualquer posição da memória de programa (até 64K bytes).	3	24
SJMP rel	Desvio curto relativo em uma faixa de 255 bytes (dentro do espaço –128 a +127 posições de memória).	2	24
JMP @A + DPTR	Desvia para o endereço obtido da soma do acumulador com o DPTR.	1	24
JZ rel	Desvia se o acumulador "for zero".	2	24
JNZ rel	Desvia se o acumulador "não for zero".	2	24
CJNE A, direto, rel	Compara e desvia se o acumulador for diferente da memória endereçada.	3	24
CJNE A, #dado8, rel	Compara e desvia se o acumulador for diferente do dado.	3	24
CJNE Rn, #dado8, rel	Compara e desvia se o registrador for diferente do dado..	3	24
CJNE @Ri, #dado8, rel	Compara e desvia se a RAM indireta for diferente do dado.	3	24
DJNZ Rn, rel	Decrementa o registrador e desvia se for "diferente de zero".	2	24
DJNZ direto, rel	Decrementa a memória e desvia se for "diferente de zero".	3	24
NOP	Nenhuma operação.	1	12

Instrução para variáveis booleanas

Mnemônico	Descrição	Nº de bytes	Nº de pulsos de clock
CLR C	Zera o carry flag	1	12
CLR bit	Zera o bit endereçado	2	12
SETB C	Seta o carry flag	1	12
SETB bit	Seta o bit endereçado	2	12
CPL C	Inverte o estado do carry flag	1	12
CPL bit	Inverte o estado do bit endereçado	2	12
ANL C, bit	"E" entre o carry flag e o bit endereçado. Resultado no carry flag.	2	24
ANL C, /bit	"E" entre o carry flag e o complemento do bit endereçado. Resultado no carry flag.	2	24
ORL C, bit	"OU" entre o carry flag e o bit endereçado. Resultado no carry flag.	2	24
ORL C, /bit	"OU" entre o carry flag e o complemento do bit endereçado. Resultado no carry flag.	2	24
MOV C, bit	Move o bit endereçado para o carry flag.	2	12
MOV bit, C	Move o carry flag para o bit endereçado.	2	24
JC rel	Desvia se o carry flag estiver setado.	2	24
JNC rel	Desvia se o carry flag estiver zerado.	2	24
JB bit, rel	Desvia se o bit endereçado estiver setado.	3	24
JNB bit, rel	Desvia se o bit endereçado estiver zerado.	3	24
JBC bit, rel	Desvia se o bit endereçado estiver setado e depois zera o bit.	3	24

Programação e simulação

"Compilação" e "Linkagem"

Um programa escrito em linguagem assembly (Programa Fonte), não pode ser diretamente processado pelo microcontrolador do sistema, devendo primeiramente ser traduzido para a sua linguagem de máquina, com o uso de tabelas ou através de um programa destinado para tal tarefa chamado de Compilador, que fornece então como saída o Programa Objeto.

Define-se Compilador como um programa aplicativo que transforma um arquivo constituído por códigos ASCII (Programa Fonte: obrigatoriamente com extensão ".ASM"), gerado normalmente por um editor de textos, em um arquivo binário que contém os bytes correspondentes às instruções (códigos de máquina) do microcontrolador.

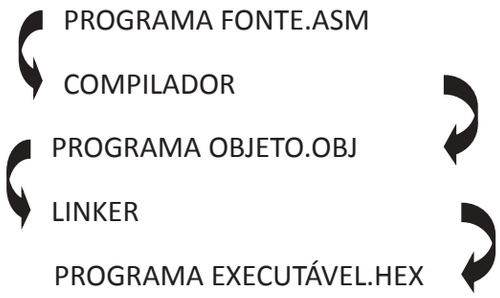
Como resultado da compilação são criados dois arquivos:

- Arquivo de mesmo nome porém com a extensão ".OBJ" (Programa Objeto).
- Arquivo de mesmo nome, porém com a extensão ".PRN", que corresponde a um arquivo texto que mostra o resultado da compilação, contendo para cada linha de programa, o código de máquina correspondente à instrução, sendo muito útil na depuração de erros de compilação.

A "linkagem" tem a finalidade de reunir, em um único arquivo, todos as rotinas escrita em um ou vários arquivos diferentes.

Após a "linkagem" são gerados dois arquivos:

- Arquivo de mesmo nome, sem extensão, usado pelo programa simulador.
- Arquivo de mesmo nome, porém com a extensão ".HEX", usado por gravadores de memórias e microcontroladores e também pelo programa simulador.



O compilador, o "Linker" e o Simulador usados no laboratório são produzidos pela AVOCET.

Diretivas (ou pseudo-instruções)

Além das instruções pertencentes ao microcontrolador em questão, a linguagem assembly possui ainda algumas instruções especiais, pseudo-instruções ou diretivas, que são usadas apenas para a estruturação do programa.

Estas instruções especiais, que não são traduzidas para o código de máquina por não pertencerem ao conjunto de instruções do microcontrolador escolhido, possuem apenas funções especiais no programa como: definir símbolos, estabelecer o endereço inicial do programa, reservar área de memória etc, não sendo portanto, processadas.

Pseudo-Instruções Mais Usadas

ORG (ORIGIN)

Formato:

Pseudo-instrução	operando
ORG	endereço

Função:

- Usado para o endereço inicial de memória, no qual o programa ou um trecho de programa será armazenado.

Exemplo:

ORG 0100H

Assim o programa objeto será carregado na memória a partir do endereço 0100H

DB (DEFINE BYTE)

Formato:

Pseudo-instrução operando

DB byte

Função:

- O byte do operando é carregado diretamente na posição de memória escolhida pelo ORG.

Exemplo:

ORG 0050H

DB 3FH

DB 1AH

Assim os bytes 3FH e 1AH foram armazenados nas posições de memória 0050H e 0051H respetivamente.

END

Formato:

Pseudo-instrução

END

Função:

- Indica o final do programa.

Exercícios / exemplos de programação

OBJETIVOS:

- Praticar o uso do programa simulador da família de microcontroladores estudada.
- Ampliar o conjunto de instruções e pseudo-instruções (diretivas) conhecido pelos alunos.
- Explorar recursos do microcontrolador como: interrupções, contadores etc.
- Obter respostas com estruturas próximas as empregadas em sistemas reais microcontrolados, equivalendo ao FIRMWARE destes (SOFTWARE armazenado exclusivamente em memória não volátil, e com a função de controlar o HARDWARE).
- 5. Em todos os enunciados (dos exercícios seguintes) fica subentendido a seguinte frase: “Escrever, compilar e simular um programa em linguagem assembly do microcontrolador 8751 para...”.

EXERCÍCIO:

Escrever e simular um programa para o microcontrolador 8751 que execute as seguintes tarefas:

a. Inicialmente carregue os registradores R0, R1, R2, R3, R4, R5, R6 e R7 com os valores: 00H, 10H, 20H, 30H, 40H, 50H, 60H e 70H respetivamente.
b. Envie os bytes 00H para P0, FFH para P1, 0FH para P2 e AAH para P3.
c. Incremente o conteúdo dos registradores R0, R1, R2, R3, R4, R5, R6 e R7.
d. Envie os bytes FFH para P0, 00H para P1, F0H para P2 e 55H para P3.
e. Volte ao passo C (entrando em loop).

OBS:

- Atenção com o uso das diretivas “ORG” e “END” e dos label “L1”.
- Para facilitar o entendimento e a depuração, procure sempre manter uma boa estética na digitação do programa.
- Testar a solução apresentada procurando ambientar-se com a operação do simulador:

EXERCÍCIO:

Escrever e simular um programa para o microcontrolador 8751 com o objetivo de controlar os bytes fornecidos para as portas P0 e P1 (saídas do sistema) através do status do bit P3.7 (entrada do sistema), da seguinte forma:

Enquanto a entrada P3.7 for = 0	Enquanto a entrada P3.7 for = 1
Enviar para a porta P0 alternadamente os bytes 00h e FFh, separados de um pequeno "delay". Deixa a porta P1 inalterada.	Enviar para a porta P1 alternadamente os bytes 55h e AAh, separados de um pequeno "delay". Deixa a porta P0 inalterada.

OBS:

- Atenção com o uso das diretivas "ORG" e "END" e dos labels "L1", "L2" e "DELAY".
- Verifique o uso da instrução de "teste de bit" e de chamada e retorno de sub-rotina.
- Para facilitar o entendimento e a depuração, procure sempre manter uma boa estética na digitação do programa.
- Testar a solução apresentada procurando ambientar-se com a operação do simulador:

EXERCÍCIO:

Escrever um programa que execute os seguintes passos:

- Inicie o sistema carregando: R0 ← 00h, R1←01h, R2←02h, R3 ← 03h, R4 ← 04h, R5 ←05h, R6 ← 06h, R7 ← 07h
- Reset a carry flag.
- Faça uma leitura de uma porta de entrada (P1) e através da análise deste byte possibilite a escolha de uma das funções descritas na tabela a seguir.
- Programa entra em loop e volta ao passo " C ".

Bit	= 0	= 1	
P1.7	----------	Envia p/ P2 o conteúdo de R0, R1,...R7 Separados por um "pequeno" delay	
P1.6	----------	Incrementar o conteúdo de todos Rn	
P1.5	----------	Decrementar o conteúdo de todos Rn	
P1.4	----------	Rotacionar todos Rn de 4 bits ou Trocar os seus respectivos nibbles	
P1.3	----------	Rotacionar todos Rn de 3 bits	
P1.2	----------	Rotacionar todos Rn de 2 bits	
P1.1	----------	Rotacionar todos Rn de 1 bit	
P1.0	Escolha do sentido de Rotação: p/ esquerda	Escolha do sentido de rotação: p/ direita	OBS: p/ os bits 4 ou 3 ou 2 ou 1 = "1"

OBS:

- Admitir que o sistema nunca fornecerá 2 bits iguais a "1" simultaneamente nos pinos: P1.7, P1.6, P1.5, P1.4 P1.3, P1.2 e P1.1.
- Antes de iniciar a simulação fazer P1 = 00h (condição inicial).
- Simular "passo a passo" o programa observando sua execução.

SUGESTÕES:

- Usar as instruções: RL A; RR A; SWAP A; CLR C; JNB bit, rel; LJMP end

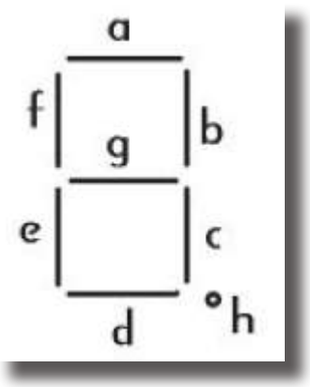
EXEMPLO:

Escrever um programa que realize as seguintes operações:

- Inicialmente leia o byte presente na porta P0.
- Depois atualize o display ligado na porta P1 com o valor em hexadecimal correspondente. ao número binário formado pelos bits presentes nos pinos: P0.3 P0.2 P0.1 P0.0.
- O programa deve entrar em loop.

Características do display e da conexão:

- Display de LED's, cátodo comum e de 7 segmentos + ponto decimal.



Ligação dos segmentos (através de um resistor de 220 ohms):

Pinos da porta 1:	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
Segmentos:	h	g	f	e	d	c	b	a

- Tabela de códigos de acendimento:

Caractere	h P1.7	g P1.6	f P1.5	e P1.4	d P1.3	c P1.2	b P1.1	a P1.0	Código
0	0	0	1	1	1	1	1	1	3FH
1	0	0	0	0	0	1	1	0	06H
2	0	1	0	1	1	0	1	1	5BH
3	0	1	0	0	1	1	1	1	4FH
4	0	1	1	0	0	1	1	0	66H
5	0	1	1	0	1	1	0	1	6DH
6	0	1	1	1	1	1	0	1	7DH
7	0	0	0	0	0	1	1	1	07H
8	0	1	1	1	1	1	1	1	7FH
9	0	1	1	0	1	1	1	1	6FH
A	0	1	1	1	0	1	1	1	77H
B	0	1	1	1	1	1	0	0	7CH
C	0	0	1	1	1	0	0	1	39H
D	0	1	0	1	1	1	1	0	5EH
E	0	1	1	1	1	0	0	1	79H
F	0	1	1	1	0	0	0	1	71H

EXERCÍCIO:

Dado o seguinte sistema microcontrolado baseado na CPU 8751:

- Motor de passo ligado nos bits: P1.3 , P1.2 , P1.1 e P1.0 conforme a tabela:

Tabela de acionamento do motor				
A→ P1.3	B→ P1.2	A'→ P1.1	B'→ P1.0	
0	0	0	1	Passo 1
0	0	1	0	Passo 2
0	1	0	0	Passo 3
1	0	0	0	Passo 4

Escrever um programa que execute o controlo do sistema da seguinte forma:

- Movimente o motor continuamente, com o sentido de rotação selecionado pelo bit de entrada P3.7:

  P3.7 = 0 → Sentido horário

  P3.7 = 1 → Sentido anti-horário

Projetos

Recomendações sobre projetos e montagens de sistemas microcontrolados

ALIMENTAÇÃO:

- Deve ser externa à placa.
- Conectar um condensador de poliéster (aproximadamente 10nF) ligado entre os pinos “Vcc” e “GND” , ao lado do microcontrolador.
- Conectar um condensador eletrolítico (mínimo 10uF) na entrada da alimentação na placa.

MONTAGEM:

- Microcontrolador deve ficar num soquete de pinos torneados.
- Usar conectores para os bits das portas envolvidos na aplicação (macho tipo “prego”) ou fios soldados diretamente na placa.
- Usar bornes simples para a alimentação ou fios soldados diretamente na placa.
- Usar pés de borracha ou outro isolador, para a sustentação da placa.
- Usar uma placa padrão preferencialmente com as trilhas desenhadas como um “proto board” (evitar as placas com ilhas isoladas que são ideais para montagens em wire-wrap).

OSCILADOR:

- A família 8051 trabalha tipicamente com cristais de 8, 10, 12MHz, conforme o tipo da CPU e de no mínimo 3,5MHz (um valor fácil de ser encontrado é de 11,0592MHz, usado pelo sinal de croma de televisão).
- O encapsulamento metálico do cristal deve ser soldado ao GND da placa.
- As ligações do circuito oscilador devem ser as mais curtas possíveis.

RESET:

- Para o reconhecimento da CPU este sinal deve permanecer em “1” por 2 ou mais ciclos de máquina (clock na frequência do cristal usado).
- Após o Reset:
  - O “PC”, o “Acumulador”, o registrador “B”, os “Flags”, o “DPTR” e os registros dos “TIMERS’ são zerados.
  - O “SBUF” (buffer serial) estará com o conteúdo indeterminado e o “SCON” (serial control) será colocado a 0.
  - Os registradores de controle de interrupção “IE” (Interrupt Enable) e “IP” (Interrupt Priority) terão o valor binário XXX00000.
  - No “SP” (stack pointer) é carregado o valor 07H.
  - As portas P0, P1, P2 e P3 terão o valor FFH.
- Durante o Reset:
  - O nível lógico dos pinos das portas é indeterminado, indo para “1” após esse período.
  - Deve-se considerar essas características para evitar o acionamento não desejado de qualquer periférico externo.
- A RAM interna não é afetada pelo “Reset forçado” ou partida quente (push-botton de Reset) e após o “Power-On” ou partida fria, o seu conteúdo é aleatório.

PORTAS:

- P0: Porta bidirecional de 8 bits.
  - Cada pino desta porta pode suprir / drenar 2 cargas TTL.
  - Quando configurada como entrada, terá o nível lógico de seus pinos flutuando na ausência de sinal.
- P1: Porta bidirecional de 8 bits com pull-ups internos.
  - Cada pino desta porta pode suprir / drenar uma carga TTL ou várias CMOS sem pull-ups externos.
  - Os pinos têm sempre um estado definido (“1”ou “0”), possibilitando que o nível lógico do pino possa ser medido, mesmo quando usado como entrada.

- P2: Porta bidirecional de 8 bits com pull-ups internos.
    - Cada pino desta porta pode suprir / drenar uma carga TTL ou várias CMOS sem pull-ups externos.
    - Os pinos têm sempre um estado definido (“1”ou “0”), possibilitando que o nível lógico do pino possa ser medido, mesmo quando usado como entrada.
- P3: Porta bidirecional de 8 bits com pull-ups internos, servindo também para funções especiais.
    - Os pinos têm sempre um estado definido (“1”ou “0”), possibilitando que o nível lógico do pino possa ser medido, mesmo quando usado como entrada.
    - Com o uso alguma das funções especiais, a P3 deve ser tratada apenas como porta de I/O de apenas bit endereçável.

HARDWARE ADICIONAL:

- Usar LED’s indicadores de status do sistema, ativos em “0”, conforme esquema.
- Não existe a necessidade do uso de foto-acopladores na aplicação.

SOFTWARE:

- Para que o programa ocupe apenas a memória ROM interna deve-se manter o PC < 4096 ou PC < 0FFFH, assim otimize esse espaço para alojar o seu programa e cuidado com aplicações que necessitem de grandes tabelas.
- Inicie o desenvolvimento do programa com um algoritmo ou fluxograma correspondente, tão detalhado quanto for a sua necessidade de compreender todos os passos intermediários necessários na aplicação.
- Durante o desenvolvimento, criar versões intermediárias do software para testar o hardware e rotinas importantes do projeto, jamais deixe para executar os testes na última versão do programa.

ESCOLHA DO CPU:

- Recomenda-se o uso de uma das seguintes CPU’s ou equivalentes:
    - 8751 da intel (modelo com EPROM)
    - AT89S51, AT89S52, AT89S53 ou AT89S8252 da ATMEL (modelos com memória FLASH)

PROGRAMAÇÃO:

- Para modelos com EPROM:
  - O grupo deve usar o “apagador” de forma segura não expondo-se a radiação ultra-violeta (o tempo de apagamento é de aproximadamente 20 min e tende a aumentar com o número de operações).
  - O grupo deve procurar o Professor para receber orientação sobre correto uso do gravador.
  - O arquivo “.HEX” pode ser usado para gravação do programa no microcontrolador de duas formas:
    - Usando o software de comunicação PC ←→ Gravador (gravação automática).
    - Digitando-se diretamente o arquivo em hexadecimal no gravador (gravação manual), sendo que para isso deve-se imprimir o arquivo “.HEX” (formato Hexa Intel) e separar os bytes do programa (que serão digitados) daqueles usados apenas para a comunicação (que devem ser rejeitados) conforme exemplo e descrições abaixo:
    - Neste formato temos a seguinte composição:

(Quantidade em Hexa) (Endereço do primeiro byte) 00 (Bytes do programa) CS

Exemplo: 10 00 20 00 23 80 F5 03 80 F2 78 0E 75 8D 07 75 8B 53 D2 8E 81

Neste arquivo temos:

- Os dois pontos (:);
- 10, indicando que são 10H (16 em decimal) os bytes na linha;
- 0020, indicando que o endereço da EPROM, do primeiro byte de programa na linha é 0020H;
- 00 é a separação dos campos (sempre será 00H);
- os 16 bytes da linha impressa (23 80 F5 03 80 F2 78 0E 75 8D 07 75 8B 53 D2 8E) os únicos que devem ser digitados;
- CS é um byte de Check Sum, usado para a verificação de integridade do arquivo após a transmissão.

- Assim é possível a gravação do arquivo sem a comunicação PC ←→ gravador, utilizando-se apenas os bytes referentes ao programa, observando-se o endereço inicial.
- O arquivo ".LST" também pode ser usado para a gravação manual do programa, dispensando a retirada de bytes como no ".HEX" más provavelmente necessitando de mais folhas de papel para a impressão.

Para modelos com memória FLASH:

- Usando um simples circuito gravador (com esquema fornecido a seguir), que pode opcionalmente ser montado pelo grupo e um software específico de comunicação PC ←→ Gravador.

PRODUTO FINAL

- Cpu montada em placa padrão, conforme esquema do sistema mínimo mostrado a seguir.
- Hardware adicional, necessário para a aplicação escolhida com forma de montagem livre.
- Documentação técnica incluindo:
  - Esquema eletrônico do hardware adicional.
  - Fluxograma ou algoritmo geral e listagem do software de controle comentada (arquivo .LST).

AVALIAÇÃO

- Trabalho em grupo de no máximo 3 ou 4 alunos (dependendo do Professor).
- Avaliação individual considerando-se o:
  - O funcionamento do projeto.
  - A documentação entregue.
  - A efetiva participação de cada componente do grupo
  - Argumentação feita pelo Professor individualmente aos alunos na data de apresentação.

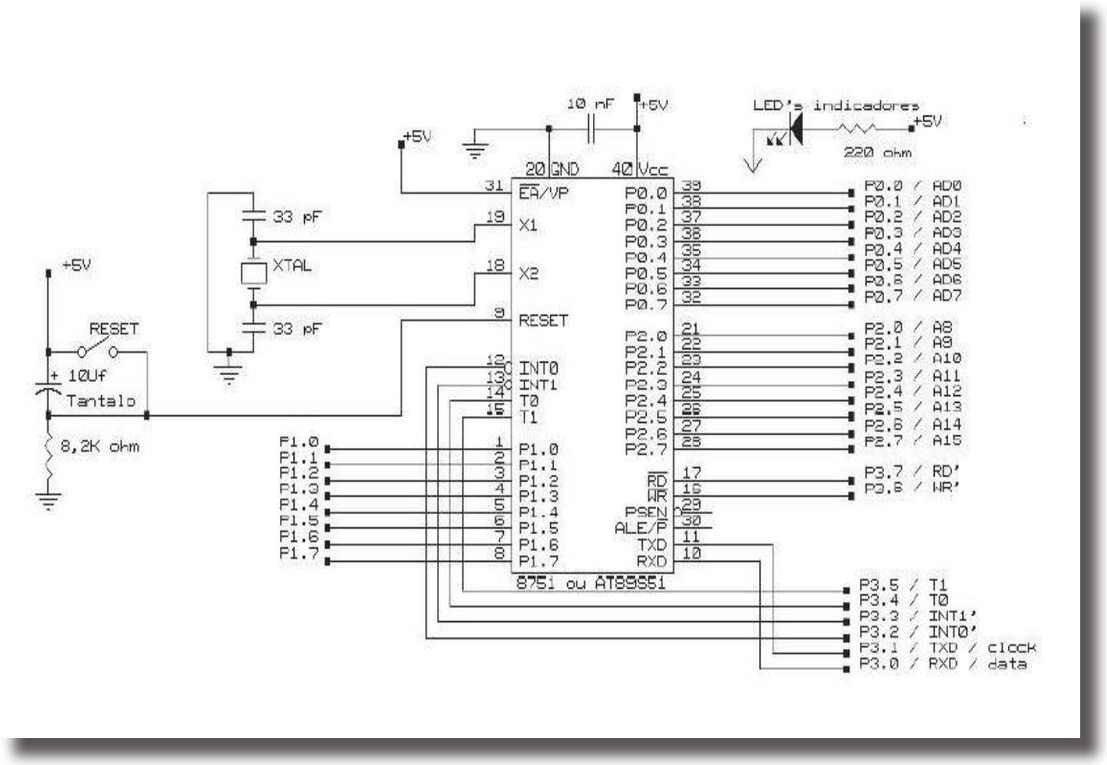
CALENDÁRIO:

Identificação do grupo e escolha do tema do projeto: _____

Datas para o desenvolvimento do projeto: _____

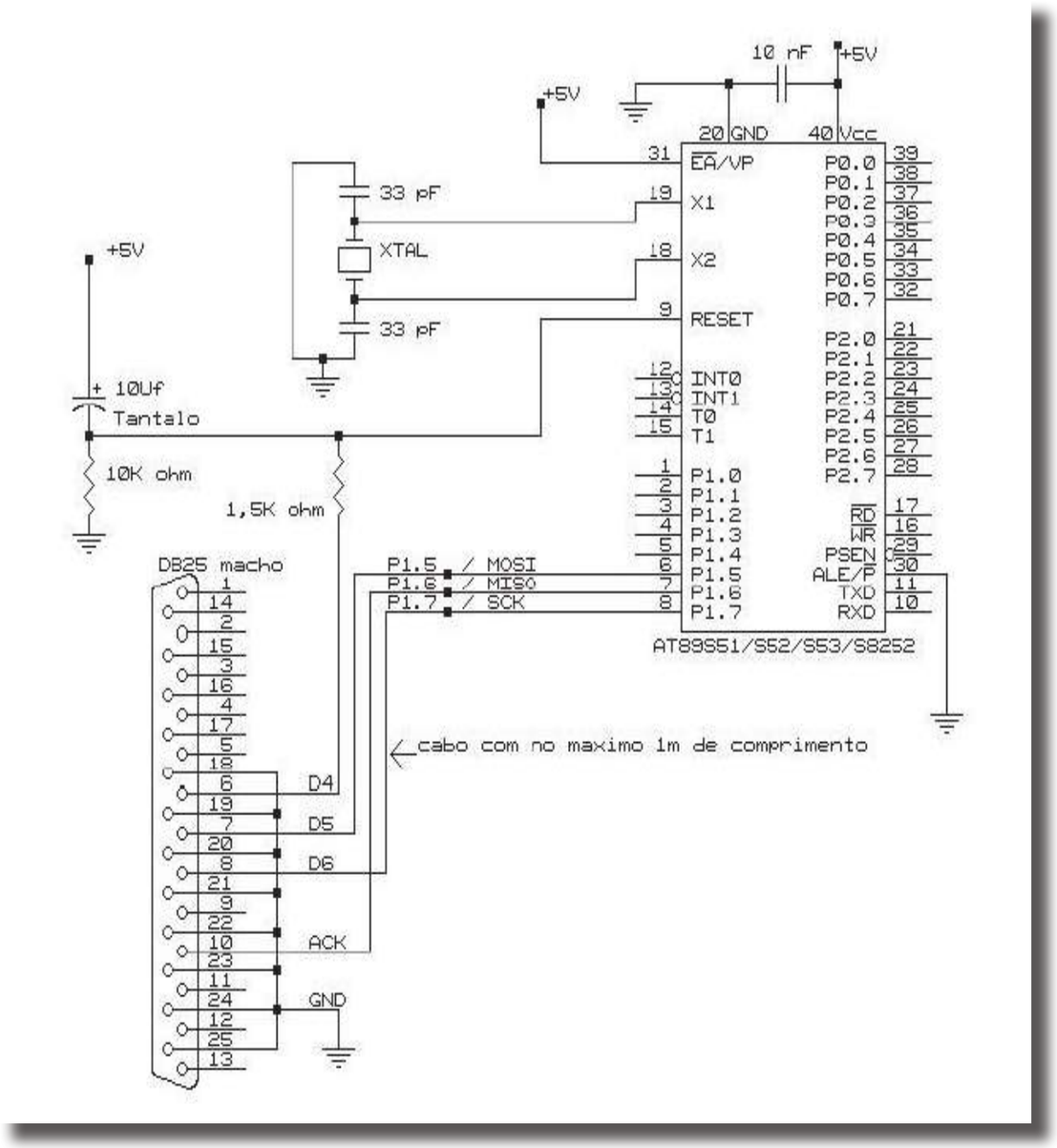
Data limite para apresentação do projeto: _____

CPU para um sistema mínimo



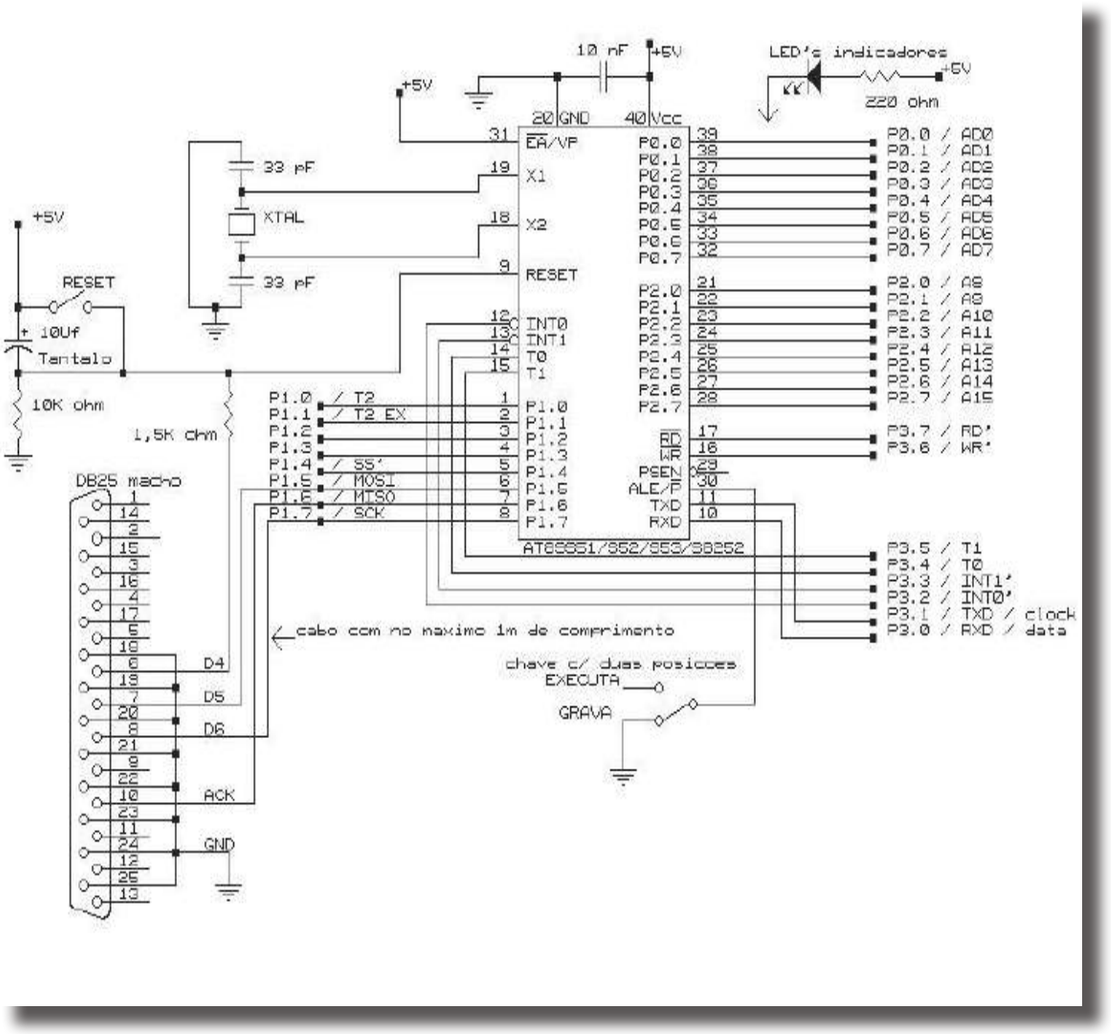
Circuito do gravador

(por exemplo para as cpu's AT89S51, AT89S52, AT89S53 e AT89S8252)



CPU do sistema mínimo e gravador

(por exemplo para as cpu's AT89S51, AT89S52, AT89S53 e AT89S8252)



Bibliografia

GONÇALVES, Victor, Sistemas Electrónicos com Microcontroladores. ETEP. (s.d.).

JR, Vidal Pereira da Silva, Aplicações Práticas do Microcontrolador 8051. Editora Eriça, 12ª Edição 2004.

NICOLOSI, Denys E. C., Microcontrolador 8051. Editora Eriça, 5ª Edição 2004.

Notas