Arquitetura e Organização de Computadores
Projetando um Microprocessador

FEVEREIRO DE 2021

Prof. Dr. Marcelo Oliveira Camponez

SUMÁRIO

1 A MATEMÁTICA DO COMPUTADOR

A partir dos estudos de Boole foram surgindo as bases para o processamento matemático computacional. A partir da segunda geração de computadores a representação binária se tornou um padrão para a computação.

1.1 ARITMÉTICA BINÁRIA

Em binário são definidas as seguintes operações básicas sobre inteiros:

Tabela 1 – Adição em binário

0	+	0	=	0
0	+	1	=	1
1	+	0	=	1
1	+	1	=	0 (e vai 1)

Tabela 2 – Subtração em binário

0	-	0	=	0
0	-	1	=	1 (e vai 1)
1	-	0	=	1
1	-	1	=	0

Tabela 3 – Multiplicação em binário

0	x	0	=	0
0	x	1	=	0
1	x	0	=	0
1	x	1	=	1

Além da operação de divisão;

Exemplos:

DECIMAL	BINÁRIO	HEXADECIMAL
142 + 47 = 189	10001110 + 101111 = 10111101	8E + 2F = BD
142 - 47 = 95	10001110 - 101111 = 1011111	8E - 2F = 5F

DIVISÃO EM BINÁRIO

```
 11011 | 101
-101   |---
 ----  | 101
 00111
  -101
  ----
   010
```

MULTIPLICAÇÃO EM BINÁRIO

```
    1011
  x 1101
  ------
    1011
   0000
  1011
 1011
 --------
 10001111
```

1.2 BASES NUMÉRICAS

Desde quando se começou a registrar informações sobre quantidades, foram criados diversos métodos de representá-las.

O método com qual estamos acostumados usa um sistema de numeração posicional. Isso significa que a posição ocupada por cada algarismo em um número altera seu valor de uma potência de 10 (na base 10) para cada casa à esquerda.

Por exemplo, no sistema decimal (base 10), no número 125 o algarismo 1 representa 100 (uma centena ou 10^2) , o 2 representa 20 (duas dezenas ou $2x10^1$) e o 5 representa 5 mesmo (5 unidades ou $5x10^0$). Assim, em nossa notação,

$125 = 1x10^2 + 2x10^1 + 5x10^0$

A base de um sistema é a quantidade de algarismos disponível na representação. A base 10 é hoje a mais usualmente empregada, embora não seja a única utilizada. No comércio pedimos uma dúzia de rosas ou uma grosa de parafusos (base 12) e também marcamos o tempo em minutos e segundos (base 60).

Os computadores utilizam a base 2 (sistema binário) e os programadores, por facilidade, usam em geral uma base que seja uma potência de 2, tal como 2^4 (base 16 ou sistema hexadecimal) ou eventualmente ainda 2^3 (base 8 ou sistema octal).

Na base 10, dispomos de 10 algarismos para a representação do número: 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9. Na base 2, seriam apenas 2 algarismos: 0 e 1. Na base 16, seriam 16: os 10 algarismos aos quais estamos acostumados, mais os símbolos A, B, C, D, E e F, representando respectivamente 10, 11, 12, 13, 14 e 15 unidades. Generalizando, temos que uma base b qualquer disporá de b algarismos, variando entre 0 e (b-1).

A representação 125,3810 (base 10) significa $1x10^2 + 2x10^1 + 5x10^0 + 3x10^{-1} + 8x10^{-2}$:

Generalizando, representamos uma quantidade N qualquer, numa dada base b, com um número tal como segue:

$Nb = an.b^n + ... + a2.b^2 + a1.^{b1} + a0.b^0 + a\text{-}1.b^{-1} + a\text{-}2.b^{-2} + ... + a\text{-}n.b^{-n}$ sendo que

$an.b^n + ... + a2.b^2 + a1.b^1 + a0.b^0$ é a parte inteira e

$a\text{-}1.b^{-1} + a\text{-}2.b^{-2} + ... + a\text{-}n.b^{-n}$ é a parte fracionária.

Intuitivamente, sabemos que o maior número que podemos representar, com n algarismos, na base b, será o número composto n vezes pelo maior algarismo disponível naquela base (ou seja, b-1). Por exemplo, o maior número que pode ser representado na base 10 usando 3 algarismos será 999 (ou seja, $10^3 - 1 = 999$).

Generalizando, podemos ver que o maior número inteiro N que pode ser representado, em uma dada base b, com n algarismos (n "casas"), será $N = b^n - 1$. Assim, o maior número de 2 algarismos na base 16 será FF_{16} que, na base 10, equivale a $25510 = 16^2 - 1$.

1.2.1 - Representação Binária

Os computadores modernos utilizam apenas o sistema binário, isto é, todas as informações armazenadas ou processadas no computador usam apenas duas grandezas, representadas pelos algarismos 0 e 1. Essa decisão de projeto deve-se à maior facilidade de representação interna no computador, que é obtida através de dois diferentes níveis de tensão. Havendo apenas dois algarismos, portanto dígitos binários, o elemento mínimo de informação nos computadores foi apelidado de bit (uma contração do inglês **b**inary dig**it**).

Na base 2, o número "10" vale dois. Mas se $10_2 = 2_{10}$, então dez é igual a dois?

Não, dez não é e nunca será igual a dois!

Na realidade, "10" não significa necessariamente "dez". Nós estamos acostumados a associar "10" a "dez" porque estamos acostumados a usar o sistema de numeração decimal.

O número 10_2 seria lido "um-zero" na base 2 e vale 2_{10} (convertido para "dois" na base dez), 10_5 seria lido "um-zero" na base 5 e vale 5_{10} (convertido para "cinco" na base dez), 10_{10} pode ser lido como "um-zero" na base 10 ou então como "dez" na base dez, 10_{16} seria lido "um-zero" na base 16 e vale 16_{10} (convertido para "dezesseis" na base dez), etc.

Portanto, 10 só será igual a dez se - e somente se - o número estiver representado na base dez!

Uma curiosidade: o número "10_b" vale sempre igual à base, porque em uma dada base b os algarismos possíveis vão sempre de 0 a (b - 1)! Como o maior algarismo possível em uma dada base b é igual a (b-1), o próximo número será (b - 1 + 1 = b) e portanto será sempre 10 e assim, numa dada base qualquer, o valor da base será sempre representado por "10"!

Obs.: Toda vez que um número for apresentado sem que seja indicado em qual sistema de numeração ele está representado, estenderemos que a base é dez. Sempre que outra base for utilizada, a base será obrigatoriamente indicada.

Um dia pode ser que os computadores se tornem obrigatórios e sejamos todos forçados por lei a estudar a aritmética em binário! Mas, mesmo antes disso, quem programa computadores precisa conhecer a representação em binário! Vamos começar entendendo as potências de dois.

A representação binária é perfeitamente adequada para utilização pelos computadores. No entanto, um número representado em binário apresenta muitos bits, ficando longo e passível de erros quando manipulado por seres humanos normais como, por exemplo os programadores, analistas e engenheiros de sistemas. Para facilitar a visualização e manipulação por programadores de grandezas processadas em computadores, são usualmente adotadas as representações octal (base 8) e principalmente hexadecimal (base 16). Ressaltamos mais uma vez que o computador opera apenas na base 2 e as representações octal e hexadecimal não são usadas no computador, elas se destinam apenas à manipulação de grandezas pelos programadores.

As tabelas 4 e 5 mostram a formação dos número 7216 na base 10 e 1110 na base 2.

Tabela 4 – Representação 7216_{10}

10^3	10^2	10^1	10^0	← **Base 10**	
(1000)	(100)	(10)	(1)		
7	2	1	6	← **Nr. Decimal**	
|	|	|	|_____	6x1	= 6
|	|	|_____	_____	1x10	= 10
|	|_____	_____	_____	2x100	= 200
|_____	_____	_____	_____	7x1000	= 7000
					7216 (decimal)

Tabela 5 – Representação 1110_2

2^3	2^2	2^1	2^0	← **Base 2**	
(8)	(4)	(2)	(1)		
1	1	1	0	← **Nr. Binário**	
|	|	|	|____	1x0	= 0
|	|	|_____	_____	1x2	= 2
|	|_____	_____	_____	4x1	= 4
|_____	_____	_____	_____	8x1	= 8
					14 (Decimal)

1.2.2 - Conversões de Base

Binário para Decimal

O número decimal é obtido pela soma dos números binários, já multiplicados pelas potências de base 2, com coeficientes desde o 0 (zero) ao 9.

Tabela 6 – Conversão binário para decimal

2^3	2^2	2^1	2^0	**← Base 2**	
(8)	(4)	(2)	(1)		
1	1	1	0	**← Nr. Binário**	
|	|	|	|____		1x0 = 0
|	|	|_____	_____		1x2 = 2
|	|_____	_____	_____		4x1 = 4
|_____	_____	_____	_____		8x1 = 8
					14 (Decimal)

Decimal para Binário

O número Binário é obtido por divisão sucessiva do número Decimal por 2 (dois), até que o quociente seja menor que 2;

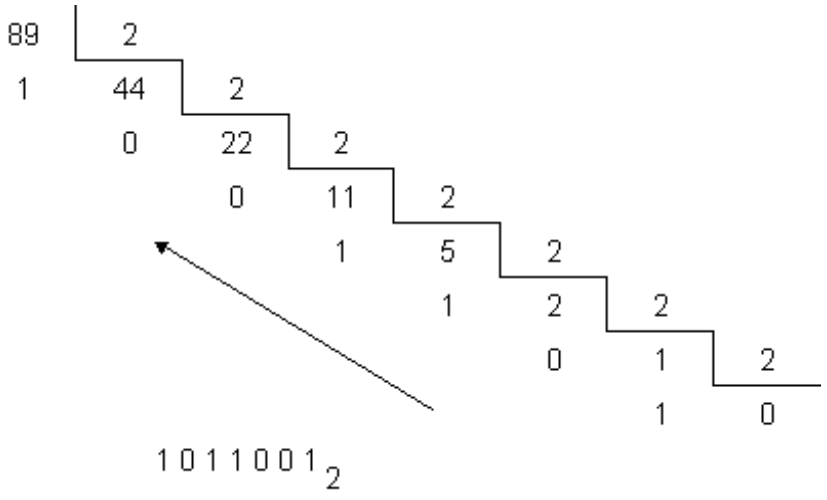


Figura 5 -Conversão de decimal para binário

Hexadecimal para Decimal

O número decimal é obtido pela soma dos números em hexadecimal, já multiplicados pelas potências de base 16, com coeficientes desde o 0 (zero) ao 9.

Tabela 7 – Conversão hexadecimal para decimal

16^3	16^2	16^1	16^0	← **Base 16**
(4096)	(256)	(16)	(1)	
4	F	6	A	← **Nr. Hexadecimal**
|	|	|	|_____	1xA ou 1x10 = 10
|	|	|_____	_____	16x6 = 96
|	|_____	_____	_____	256xF = 3840
|_____	_____	_____	_____	4096x4 = 16384
				20330 (Decimal)

Hexadecimal para Binário

Na conversão HEX-BIN, convertemos cada digito Hexadecimal separadamente em quatro dígitos, fazendo uso da tabela Hexadecimal.

Tabela 8 – Conversão hexadecimal para binário

F	6	A	9	8	7	6	5	4	3	2	1	0
1111	0110	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001	0000

Binário para Hexadecimal

Para a conversão BIN-HEX faremos o processo inverso ao Hexadecimal - Binário. Se o número Binário não for divisível por 4, adicionamos os zeros necessários à esquerda, vide tabela 8.

Decimal para Hexadecimal

O número Hexadecimal é obtido por divisão sucessiva do número Decimal por 16 (dezesseis),
até que o quociente seja menor que 16. Depois recolhemos os restos em hexadecimal de baixo para cima, isto é: 4 - 15(F) - 6 - 10(A)

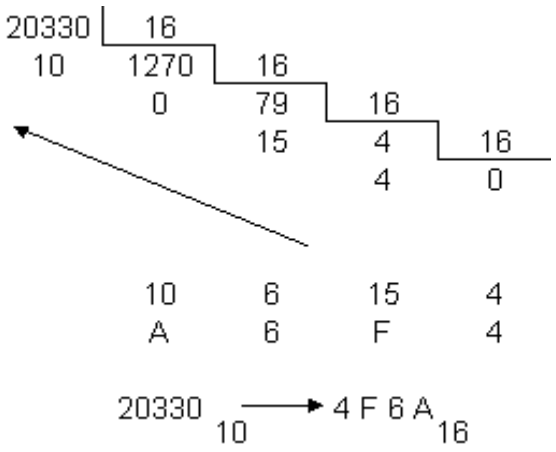


Figura 06 – Conversão Hexadecimal para decimal

1.3 REPRESENTAÇÃO DE NÚMEROS NEGATIVOS

O Complemento de dois, tabela 9, é o sistema mais usado para representação de números inteiros com sinal nos computadores modernos. O dígito mais significativo é o que informa o sinal do número. Se esse dígito for zero o número é positivo e se for um é negativo.

Os números são escritos da seguinte forma:
Positivos: sinal (bit 0) + o número em binário.
Exemplos: 0001 (+1), 0100 (+4) e 0111 (+7)

Negativos: pegamos o número em binário e "invertemos" (0100 invertendo têm-se 1011) e somamos um ao valor "invertido" (1011 + 0001 = 1100).

Pela definição temos que só existe uma representação para o zero e ela é 0000...0

Tabela 9 – Representação em complemento de 2

Número (decimal)	Representação em complemento de 2 (8 bits)
-1	11111111
2	00000010
-3	11111101
4	00000100
-5	11111011

EXERCÍCIOS

1 - Converta para o sistema decimal:

a) 100110_2
b) 011110_2
c) 111011_2
d) 1010000_2
e) 11000101_2
f) 11010110_2
g) 011001100110101_2

2 - Converta para o sistema binário:

a) 78_{10}
b) 102_{10}
c) 215_{10}
d) 404_{10}
e) 808_{10}
f) 5429_{10}
g) 16383_{10}

3 - Quantos bits necessitaríamos para representar cada um dos números decimais abaixo?

a) 512_{10}
b) 12_{10}
c) 2_{10}
d) 17_{10}
e) 33_{10}
f) 43_{10}
g) 7_{10}

4 - Converta para o sistema decimal os seguintes números hexadecimais:

a) 479_{16}
b) $4AB_{16}$
c) BDE_{16}
d) $F0CA_{16}$
e) $2D3F_{16}$

5 - Converta os seguintes números decimais em hexadecimais:

a) 486_{10}
b) 2000_{10}
c) 4096_{10}
d) 5555_{10}
e) 35479_{10}

6 - Converta para o sistema binário:

a) 84_{16} d) $47FD_{16}$

b) $7F_{16}$ e) $F1CD_{16}$

c) $3B8C_{16}$

7 - Converta para o sistema hexadecimal os seguintes números binários:

a) 10011_2 d) 11111011110010_2

b) 1110011100_2 e) 1000000000100010_2

c) 100110010011_2

8 - Efetue as operações:

a) $1000_2 + 1001_2$ d) $1110_2 + 1001011_2 + 11101_2$

b) $10001_2 + 11110_2$ e) $110101_2 + 1011001_2 + 1111110_2$

c) $101_2 + 100101_2$

9 - Represente os números $+97_{10}$ e -121_{10}, utilizando a notação sinal-módulo.

10 - Estando o número 10110010 em sinal-módulo, o que ele representa no sistema decimal?

11 - Determine o complemento de 1 de cada número binário:

a) 01110100_2

b) 11000010_2

12 - Represente os seguintes números na notação do complemento de 2:

a) -1011_2 d) -11010100_2

b) -100001_2 e) -01010011_2

c) -10111101_2

13 - Qual o equivalente em decimal do número 10110111_2, aqui representado em complemento de 2?

14 - Efetue as operações utilizando o complemento de 2:

a) $101101_2 - 100111_2$ d) $-10010011_2 + 11011010_2$

b) $10000110_2 - 110011_2$ e) $-10011101_2 - 1000101_2$

c) $111100_2 - 11101011_2$

15 - Efetue em binário as operações, utilizando a aritmética do complemento de 2:

a) $44_{16} - 3E_{16}$ c) $-BC_{16} + FC_{16}$

b) $A9_{16} - E0_{16}$ d) $-22_{16} - 1D_{16}$

2. REVISÃO ELETRÔNICA DIGITAL

2.1 PORTAS LÓGICAS

Utilizar as notas de aula.

2.2 EXPRESSÕES LÓGICAS

Utilizar as notas de aula.

2.3 TABELAS VERDADE

Utilizar as notas de aula.

2.4 CIRCUITOS SOMADORES E DECODIFICADORES

Utilizar as notas de aula.

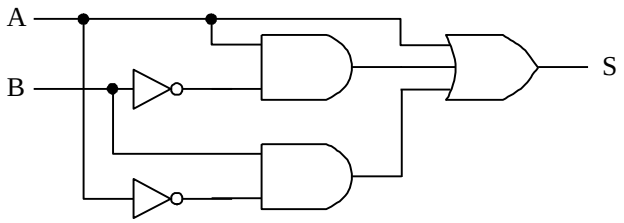
2.5 FLIP FLOP REGISTRADORES E MEMÓRIAS

Utilizar notas de aula.

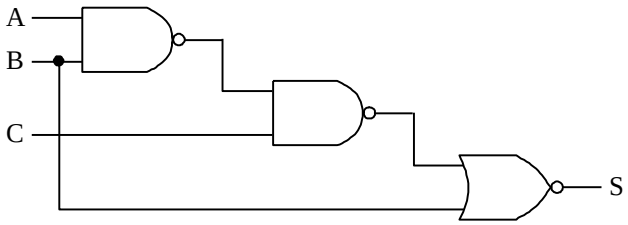
EXERCÍCIOS

1 – Determine as Expressões Booleanas de saída dos Circuitos Lógicos abaixo:

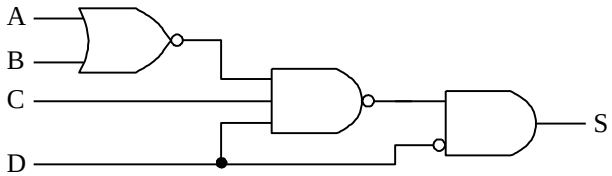
a)



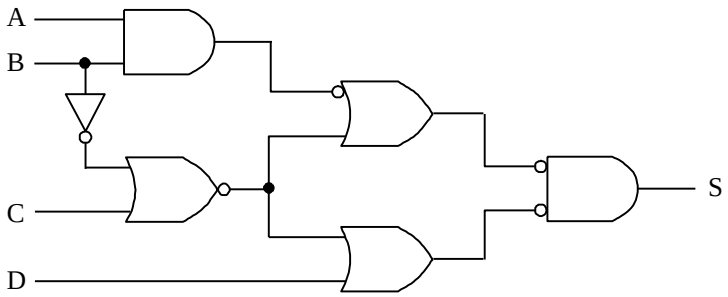
b)



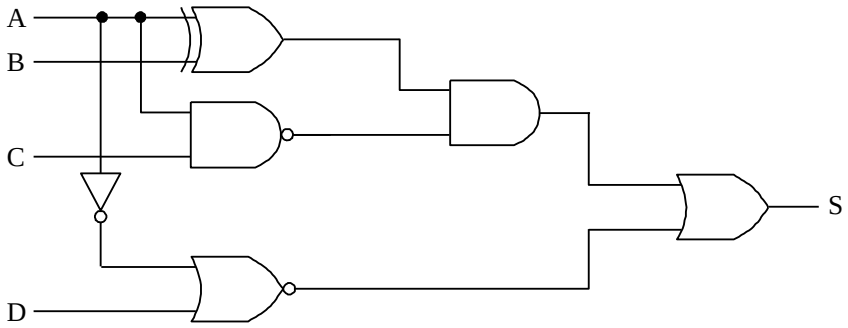
c)



d)



e)



2 – Determine os Circuitos Lógicos das seguintes Expressões Booleanas:

a) $S = \overline{\overline{A} + B}.C$

d) $S = \left(\overline{A} + \overline{B}\right).\left(\overline{\overline{A} + \overline{B}}\right)$

b) $S = \overline{\overline{A.B} + C}$

c) $S = \left(\overline{A.B + C}\right).\overline{D}$

e) $S = \left[(A \oplus B) + \overline{A}\right].\overline{C}$

f) $S = \left[\overline{B} + (A \oplus B)\right].\overline{A} + \overline{C}$

3 - Determine as Tabelas da Verdade para as Expressões Booleanas do exercício anterior.

4 – Determine as Expressões Booleanas a partir das Tabelas da Verdade:

a)

A	B	C	S
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

b)

A	B	C	S
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

c)

A	B	C	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

d)

A	B	C	S
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

e)

A	B	C	D	S
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

5) Uma memória com 1024 posições de memória precisa de quantas linhas no barramento de endereços?

6) Quais são os principais componentes de uma CPU?

7) diferenças entre arquitetura Von Newman e Harvard.

8) O que é e para que serve uma ULA?

9) Para que serve a unidade de controle?

10) O que são os registradores?

11) Para que servem as memórias RAM?

12) Suponha uma CPU que acessa uma memória RAM com 512k posições e 8 bits de largura. O chip de memória tem as seguintes linhas de CS (Chip Select); W/R; Outup Enable:

a) Quantas linhas deve ter o barramento de endereços? Porque

b) Quantas linhas deve ter o barramento de dados? E o barramento de controle? Porque?

3 ARQUITETURA DE COMPUTADORES

Existem duas principais arquiteturas usadas em processadores:

A arquitetura de **Von Newmann** que caracteriza-se por apresentar um barramento externo compartilhado entre dados e endereços. Embora apresente baixo custo, esta arquitetura apresenta desempenho limitado pelo gargalo do barramento.

A arquitetura de **Harvard** onde existem dois barramentos externos independentes (e normalmente também memórias independentes) para dados e endereços. Isto reduz de forma sensível o gargalo de barramento, que é uma das principais barreiras de desempenho, em detrimento do encarecimento do sistema como um todo (Fig.1).

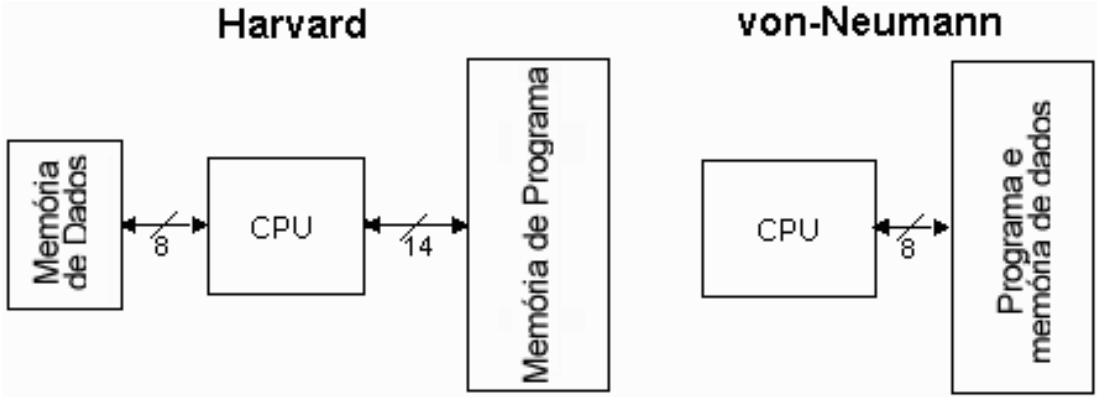


Figura 1 – Arquiteturas Von Neumann e Harvard

A figura 2 mostra um exemplo de um processador de arquitetura VON NEUMANN destacando seus principais componentes

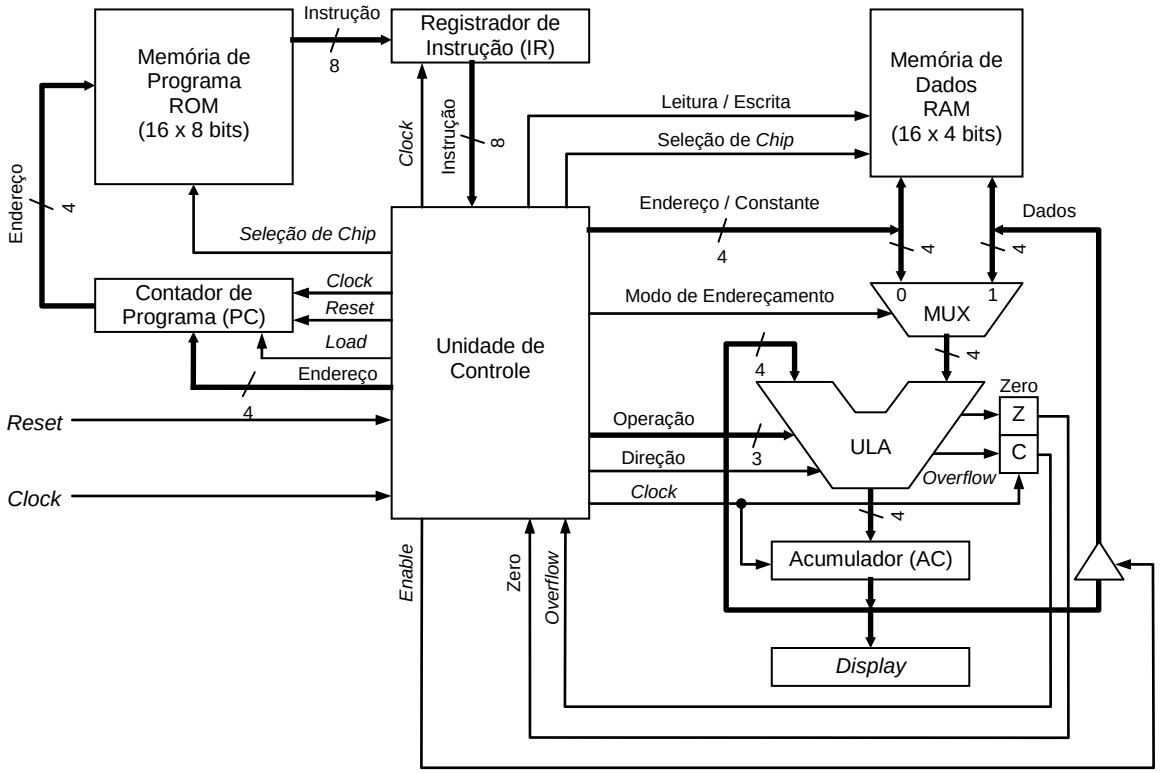


Figura 2 – Arquiteturas Von Neumann principais componentes

As CPU ou processadores são compostos basicamente de:

- **Unidade de Controle** - Responsável por gerar todos os sinais que controlam as operações no exterior do CPU, e ainda por dar todas as instruções para o correto funcionamento interno do CPU;

- Unidade lógica e aritmética - O componente principal, ela realiza todas as operações lógicas e de cálculo que serão usadas para executar uma tarefa.

- **Registradores -** Os registradores são pequenas memórias velozes que armazenam comandos ou valores que utilizados no controle e processamento de cada instrução. Os registradores mais importantes são: - Contador de Programa (PC) – Sinaliza para a próxima instrução a ser executada; - Registrador de Instrução (IR) – Registra a instrução da execução; Os outros realizam o armazenamento de resultados intermediários, como o Acumulador (AC).

3.1 UNIDADE DE CONTROLE

A Unidade de Controle (Fig. 3) é responsável por gerar todos os sinais que controlam as operações no exterior do CPU, e ainda por dar todas as instruções para o correto funcionamento interno do processador.

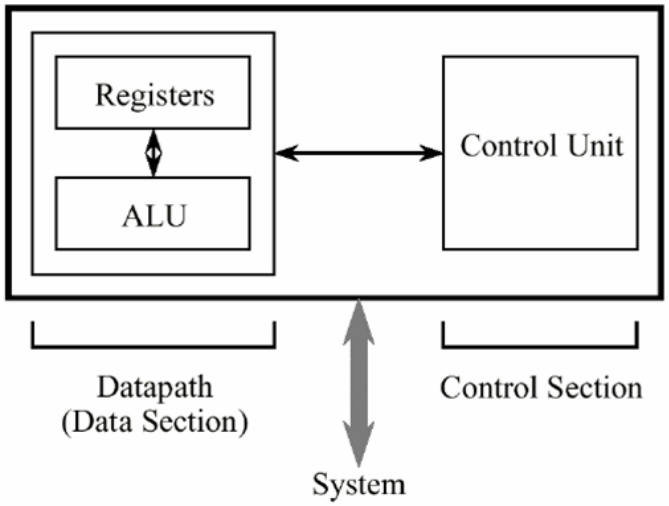


Figura 3 – Unidade de Controle

A unidade de controle executa três ações básicas intrínsecas e pré-programadas pelo próprio fabricante do processador, são elas: busca (fetch), decodificação e execução de instruções.

Assim sendo, todo processador, ao iniciar sua operação, realiza uma operação cíclica, também chamada de ***ciclo de instruções***, tendo como base essas três ações.

3.2 UNIDADE LÓGICA E ARTITMÉTICA

A Unidade lógica e aritmética (ULA) ou em inglês *Arithmetic Logic Unit* (ALU) é a unidade central do processador (Central Processing Unit, ou simplesmente CPU), que realmente executa as operações aritméticas e lógicas referenciadas pelas instruções.

A ULA executa as principais operações lógicas e aritméticas do computador. Ela soma, subtrai, divide, determina se um número é positivo ou negativo ou se é zero. Além de executar funções aritméticas, uma ULA deve ser capaz de determinar se uma quantidade é

menor ou maior que outra e quando quantidades são iguais. A ULA pode executar funções lógicas com letras e com números.

Em suma, a ULA executa:

- Operações aritméticas com inteiros
- Operações lógicas bit a bit And, Not, Or, XOR
- Operações de deslocamento de bits (deslocamento, rotação por um número específico de bits para esquerda ou direita, com ou sem sinal).

-Também toma decisões lógicas, resolvendo sintaxes lógicas em uma programação.

As entradas para a ULA são os dados a serem operados (chamados operandos) e o código da unidade de controle indicando as operações para executar. As saídas são os resultados da computação. Na Figura 7 "A" e "B" são operandos, "R" é a saída, "F" é a entrada da unidade de controle e "D" é a saída de status.

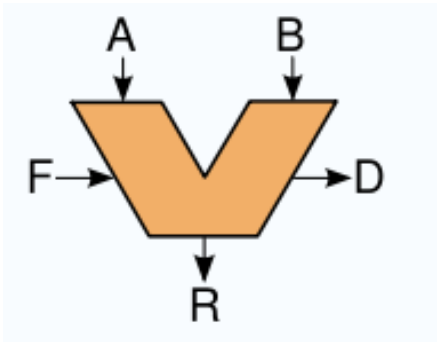


Figura 4 - Um símbolo esquemático típico para uma ULA

Em muitos projetos a ULA também leva ou gera as entradas ou saídas um conjunto de códigos de condições de ou para um registrador de status. Esses códigos são usados para indicar casos como vai-um (empresta-um), overflow, divisão-por-zero.

3.3 REGISTRADORES

Um registrador (Fig. 5) de uma unidade central de processamento é um tipo de memória de pequena capacidade porém muito rápida, contida no CPU, utilizada no armazenamento

temporário durante o processamento. Os registradores estão no topo da hierarquia de memória, sendo assim são o meio mais rápido e caro de se armazenar um dado.

São utilizados na execução de programas de computadores, disponibilizando um local para armazenar dados. Na maioria dos computadores modernos, quando da execução das instruções de um programa, os dados são movidos da memória principal para os registradores, então as instruções que utilizam estes dados são executadas pelo processador, e finalmente, os dados são movidos de volta para a memória principal.

Registradores de dados são utilizados para armazenar valores, tais como inteiros e pontos flutuante. Em algumas CPUs antigas e mais baratas, é um registrador de dados especial, conhecido como acumulador, e é utilizado implicitamente em muitas operações.

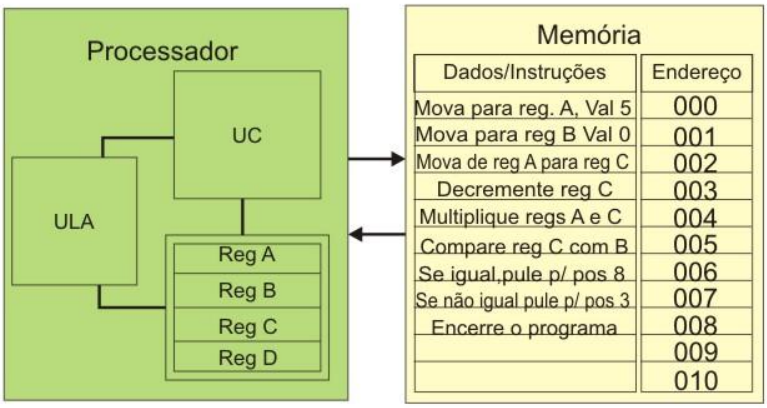


Figura 5 – Registradores

3 EXECUTANDO UM PROGRAMA

3.1 CICLO DE EXECUÇÃO DE INSTRUÇÕES

Uma **INSTRUÇÃO** é um número em binário em uma posição da memória RAM que indica a operação que dever ser feita pelo processador. Um DADO, da mesma forma, é um número em binário na memória RAM que representa uma variável de um programa;

Um **PROGRAMA** é um conjunto de instruções organizadas e orientadas para a resolução de um problema. O processador, ao executar um programa, segue um algoritmo chamado de ciclo de execução de instruções (Figs. 6 e 7). Esse ciclo contém os seguintes passos básicos: Busca de Instrução; Decodifica Instrução; Busca de Operandos; Execução de Instruções; Salvar Resultados; Incrementar o PC.

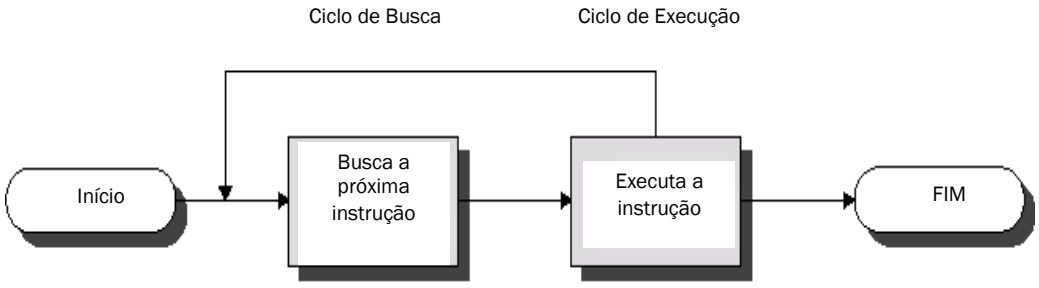


Figura 6 – Ciclo de Instruções simplificado

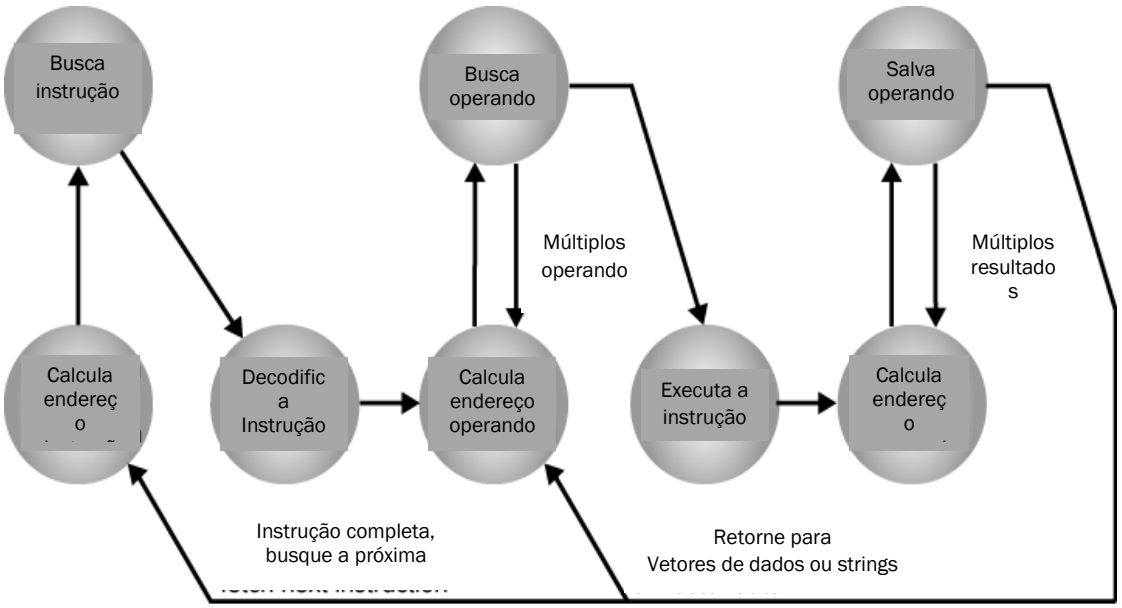


Figura 7 – Ciclo de Instruções detalhado

4 MEMÓRIAS

Todo computador é dotado de uma quantidade de memória, que varia de máquina para máquina, que se constitui de um conjunto de circuitos capazes de armazenar os dados e os programas a serem executados pela máquina.

É incrível a quantidade de memórias eletrônicas encontradas atualmente. Muitas delas se tornaram parte integral de nosso vocabulário: RAM; ROM; Cache; RAM dinâmica; RAM estática; Flash; Stick; Virtual; de Vídeo; BIOS.

4.1 HIERARQUIA DE MEMÓRIAS

Historicamente as memórias sempre foram o gargalo da computação. Os processadores sempre foram muito mais rápidos e boa parte da perda do desempenho do sistema se deve aos acessos à memórias. Por causa disso, há nos computadores modernos uma hierarquia de memórias de forma a equalizar custo, quantidade de bytes disponíveis e velocidade (Fig. 8).

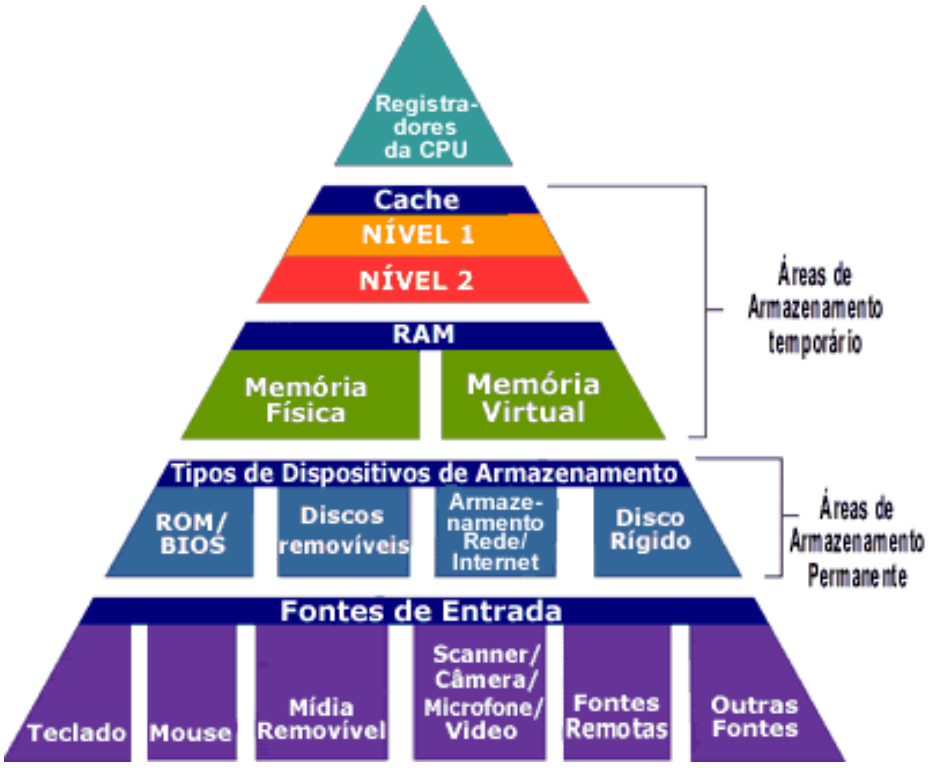


Figura 8 – Hierarquia de memórias

Dessa forma, há diversos tipos e tecnologias diferentes de memórias distribuídas em vários locais do computador, assim:

1) Registradores: as memórias mais rápidas disponíveis estão dentro do processador e existem em pequenas quantidades;

2) Memória Cache: memórias muito rápidas e caras de serem implementadas. Podem estar dentro ou fora do processador;

3) Memória RAM: São memórias rápidas, em termo de custo não são baratas e nem tão caras quanto as caches. Ficam em um slot específico da placa mãe;

4)Memórias de Massa: são exemplos o Disco Rígido; os CDs; os DVDs; Blue-Ray. São muito lentas quando comparadas às anteriores mas tem um custo por bit muito baixo;

5)Outras: BIOS, Memórias de placas de Rede, de vídeo, de audio, etc...

4.2 CAMADAS DE MEMÓRIAS

Todos os componentes no seu computador, como a CPU, o disco rígido e o sistema operacional, trabalham juntos como um time. Nele, a memória desempenha uma das funções essenciais. Desde o momento em que o computador é ligado até a hora de desligá-lo, a sua CPU está constantemente usando a memória. Vamos ver um cenário típico ao ligarmos um computador:

1) o computador carrega os dados da memória apenas de leitura (ROM) e executa um auto-teste de energia (POST - Power-On Self Test) para ter certeza de que a maioria dos componentes principais está funcionando corretamente. Uma parte desse teste, o controle de memória, checa todos os endereços de memória com um rápido processo de leitura/escrita, operação executada para certificar-se de que não há erros nos chips de memória. Esse processo permite que os dados sejam gravados em um bit e então lidos a partir desse bit;

2) o computador carrega o sistema básico de entrada/saída (BIOS - Basic Input/Output System) da memória ROM. A BIOS fornece a maioria das informações básicas sobre os dispositivos de armazenamento, seqüência de boot, segurança, plug-and-play (auto reconhecimento de dispositivo), capacidade e alguns outros itens;

3) o computador carrega o sistema operacional (SO) do disco rígido no sistema de memória RAM. Geralmente, as partes críticas do sistema operacional são mantidas na memória RAM enquanto o computador é ligado. Isso permite que a CPU tenha acesso imediato ao sistema operacional, o que aumenta a performance e a funcionalidade do sistema como um todo;

4) quando você abre um aplicativo, ele é carregado na memória RAM. Para preservar o uso dessa memória, muitos aplicativos são carregados, inicialmente apenas em suas partes essenciais, sendo outras partes carregadas conforme a necessidade;

5) depois que um aplicativo está carregado, qualquer arquivo aberto para uso no aplicativo é carregado na memória RAM;

6) quando você salva um arquivo e fecha o aplicativo, o arquivo é gravado no disco de armazenamento e, então, ele e o aplicativo são removidos da memória RAM.

Como observado na lista acima, cada vez que algo é carregado ou aberto é então colocado na memória RAM. Isso significa que eles são simplesmente colocados na área temporária de armazenamento do seu computador, para que a CPU possa acessar a informação mais facilmente. A CPU requisita os dados de que necessita da memória RAM. Novos dados, que já foram processados e gravados, voltam para a RAM em um ciclo contínuo. Na maioria dos computadores, essa troca de dados entre a CPU e RAM acontece milhões de vezes por segundo. Quando os aplicativos são fechados, eles e outros arquivos relacionados são normalmente removidos (deletados) da memória RAM, para dar lugar a novos dados. Se os arquivos alterados não forem salvos em uma unidade de armazenamento permanente antes de serem removidos, serão definitivamente perdidos.

CPUs rápidas e potentes precisam de acesso ágil e fácil a uma grande quantidade de dados para maximizar sua performance. Se a CPU não pode pegar os dados de que precisa, ela literalmente pára e espera por eles. As CPUs modernas com velocidade na faixa de 1 gigahertz podem manipular uma grande quantidade de dados, potencialmente bilhões de bytes por segundo. O problema enfrentado pelos projetistas é que uma memória capaz de acompanhar a velocidade de uma CPU de 1 gigahertz é extremamente cara para a comercialização em grandes quantidades.

Os projetistas resolveram esse problema de custo por meio de "camadas" de memória, usando memória cara em pequenas quantidades e então reforçando-a com grandes quantidades de memória mais baratas.

A forma mais barata de memória de leitura/escrita, amplamente utilizada hoje, é o disco rígido. Os discos rígidos fornecem grandes quantidades de armazenamento permanente e barato. Você pode comprar discos rígidos a um custo de centavos por megabyte, porém, pode levar um bom tempo (aproximadamente um segundo) para ler um megabyte do disco rígido. Devido ao espaço de armazenamento em um disco rígido ser tão barato e abundante, ele representa o estágio final da hierarquia da memória da CPU, chamado memória virtual.

O próximo estágio da hierarquia é a memória RAM. O tamanho em bit de uma CPU lhe diz quantos bytes de informação ela pode acessar da RAM ao mesmo tempo. Por exemplo, uma CPU de 16 bits pode processar 2 bytes ao mesmo tempo (1 byte = 8 bits, então 16 bits = 2 bytes) e uma CPU de 64 bits pode processar 8 bytes ao mesmo tempo.

Sozinho, o sistema de memória RAM do computador não é rápido o suficiente para estar compatível com a velocidade da CPU. É por isso que ele precisa de um cache (discutido mais adiante). Entretanto, quanto mais rápida a memória RAM, melhor. A maioria dos chips hoje opera com um ciclo de acesso à memória de 50 a 70 nanossegundos. A velocidade de leitura/escrita é tipicamente proveniente do tipo de memória RAM usada como DRAM, SDRAM, RAMBUS.

4.3 CLASSIFICAÇÃO DE MEMÓRIAS

Podemos identificar, quanto à função, diferentes categorias de memória:

- a **memória principal**, ou memória de trabalho, onde normalmente devem estar armazenados os programas e dados a serem manipulados pelo processador;

- a **memória secundária** que permitem armazenar uma maior quantidade de dados e instruções por um período de tempo mais longo; o disco rígido é o exemplo mais evidente de memória secundária de um computador, mas podem ser citados outros dispositivos menos recentes como as unidades de fita magnética e os cartões perfurados;

- a **memória cache**, que se constitui de uma pequena porção de memória com curto tempo de resposta, normalmente integrada aos processadores e que permite incrementar o desempenho durante a execução de um programa.

Quanto a tecnologia utilizada para a fabricação dos circuitos, as memórias podem ser classificadas como:

-**RAM** (memória de leitura e escrita): são chips de memória que podem ser lidos e gravados pela CPU a qualquer instante. A CPU usa a RAM para armazenar e executar programas vindos do disco, para ler e gravar os dados que estão sendo processados. Uma outra característica da RAM, é que se trata de uma memória VOLÁTIL. Isso significa que quando o computador é desligado, todos os seus dados são apagados. Por essa razão, é necessário que os programas e dados fiquem gravados no disco, que é uma memória permanente.

Existem vários tipos de RAM com diversas características e para diversas aplicações. A mais conhecida é a **DRAM (dinâmica)** e a **SRAM (estática)** e suas evoluções. Estes tipos serão detalhados mais adiante.

-**ROM**: São chips de memória que podem ser lidos pela CPU a qualquer instante, mas não podem ser gravados pela CPU. Sua gravação é feita apenas pelo fabricante do computador,

ou pelo fabricante de memórias. Os dados armazenados nela já saem prontos de fábrica e são produzidas em larga escala na indústria. A característica importante de ROM é que trata-se de uma memória permanente. Seu conteúdo nunca é perdido, mesmo com o computador desligado Portanto este tipo de memória é usado para armazenar programas estáticos (que não alteram) e produzidos em massa. Esse tipo de memória foi usado para armazenar o BIOS, que se localiza na placa-mãe.

-**PROM**: Significa Programmable ROM, ou seja, ROM programável. Trata-se de uma espécie de ROM que é produzida apagada. O fabricante pode programá-las, ou seja, gravar seu programa. Esta gravação pode ser feita apenas um vez, pois utiliza um processo irreversível. Por isso, usa-se o termo queimar a PROM quando se grava nesta memória.

-**EPROM**: Significa Eraseable PROM, ou seja, uma ROM programável e apagável. Assim como ocorre com a PROM, a EPROM pode ser programada e a partir daí, comporta-se como uma ROM comum, mantendo os dados armazenados mesmo sem corrente elétrica, e permitindo apenas operações de leitura. A grande diferença é que a EPROM pode ser apagada com raios ultravioleta de alta potência. Possuem uma "janela de vidro", através da qual os raios ultravioleta podem incidir nas operações de apagamento. Nota-se que essa janela de vidro fica sempre coberta por um adesivo que tampa a passagem de luz. É fácil identificar um chip EPROM na placa mãe justamente pela presença desse adesivo.

-**EEPROM**: Significa Electrically Erasable Programmable ROM (EEPROM ou E2PROM). Esta é o tipo de memória ROM mais flexível, que pode ser apagada sob o controle de *software*. Este é o tipo que se usa para armazenar as BIOS atuais. Dessa forma, o usuário pode realizar atualizações no BIOS, fornecidas pelo fabricante da placa de CPU. Quando se ouve falar em “flash BIOS” ou “fazendo um upgrade de BIOS”, isto se refere a reprogramação do BIOS EEPROM com um programa de software especial.

4.4 TIPOS DE MEMÓRIAS RAM

A memória RAM (Random Access Memory) é a forma mais conhecida de memória de computador. A memória RAM é considerada de "acesso aleatório" porque é possível acessar diretamente qualquer célula da memória se você conhece a linha e a coluna que cruzam essa célula.

Semelhante a um microprocessador, um chip de memória é um circuito integrado (CI), feito de milhões de transistores e capacitores. Na forma mais comum de memória de computador, a memória de acesso aleatório dinâmico (DRAM), um transistor e um capacitor são unidos para criar uma célula de memória, que representa um único bit de dados. O capacitor mantém o bit de informação: um 0 ou um 1. O transistor age como uma chave que permite ao circuito de controle no chip de memória ler o capacitor ou mudar seu estado.

Um capacitor é como um pequeno balde capaz de armazenar elétrons. Para armazenar um 1 na célula de memória, o balde é preenchido com elétrons (Fig. 9). Para armazenar um 0, ele é esvaziado. O problema com o balde do capacitor é que ele tem um vazamento. Em questão de poucos milésimos de segundos, um balde cheio fica vazio. Portanto, para a memória dinâmica funcionar, a CPU ou o controlador de memória tem de carregar todos os capacitores mantendo um 1 antes que eles descarreguem. Para isto, o controlador de memória lê a memória e então grava nela de volta. Esta operação de atualização (mais conhecida como refrescamento) acontece automaticamente, milhares de vezes por segundo.

Figura 9 – Memória Dinâmica

O nome DRAM vem desta operação de refrescamento. A memória DRAM tem de ser refrescada de forma dinâmica, constantemente, ou perde o que está guardando. O aspecto negativo de todo esse refrescamento é que leva tempo e deixa a memória lenta.

As células de memória são gravadas em uma pastilha de silício em uma série de colunas (bitlines) e linhas (wordlines). O cruzamento de um bitline e um wordline constitui o endereço da célula de memória (Fig. 10).

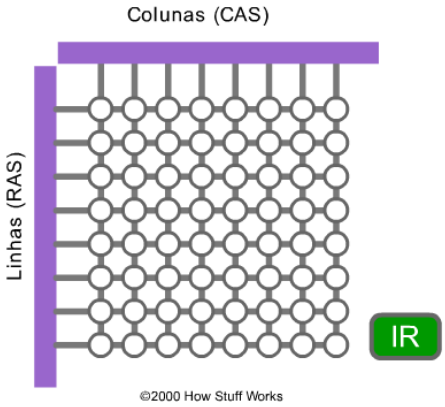


Figura 10 – CAS e RAS

A memória DRAM funciona enviando uma carga através da coluna apropriada (CAS) para ativar o transistor de cada bit na coluna. Ao gravar, as linhas contêm o estado que o capacitor deve assumir. Ao ler, um amplificador de sinal, determina o nível de carga no capacitor. Se for maior que 50%, ele o lê como um 1, caso contrário, ele o lê como um 0. Um contador guarda a seqüência de refrescamento baseado na ordem na qual as linhas foram acessadas. A duração de tempo necessária para fazer tudo isso é tão curta que é expressada

em nanosegundos (bilionésimos de um segundo). Um chip de memória de 70 ns leva 70 nanosegundos para ler e recarregar completamente cada célula.

As células de memória sozinhas seriam inúteis se não houvesse alguma maneira de obter e inserir informações nelas. As células de memória têm uma estrutura inteira de apoio composto por outros circuitos especializados. Esses circuitos realizam funções como:

- identificar cada linha e coluna (selecionar o endereço da linha e selecionar o endereço da coluna);
- manter atualizada a seqüência de refrescamento (contador);
- ler e rearmazenar o sinal de uma célula (amplificador de sinal);
- dizer a uma célula se deve levar uma carga ou não (habilitador de gravação);

Outras funções do controlador de memória abrangem uma série de tarefas como identificação do tipo, velocidade e quantidade de memória e a verificação de erros.

A RAM estática usa uma tecnologia totalmente diferente. Nela, uma forma de flip-flop contém cada bit de memória. Um flip-flop para uma célula de memória utiliza 4 ou 6 transistores mais alguns fios, mas nunca tem de ser refrescado. Isto torna a RAM estática significativamente mais rápida que a RAM dinâmica. Entretanto, como ela tem mais componentes, ocupa também muito mais espaço em um chip que uma célula de memória dinâmica. Portanto, você pode ter menos memória por chip, o que torna a RAM estática muito mais cara.

A RAM estática é rápida e cara, enquanto a DRAM é mais barata e mais lenta. A RAM estática é usada para se criar cache de velocidade compatível com a CPU, enquanto a DRAM se constitui no grande sistema de memória RAM.

Em geral, os chips de memória estão disponíveis apenas como parte de uma placa chamada módulo. Você provavelmente já viu uma memória descrita como 8x32 ou 4x16. Estes números representam o número de chips multiplicado pela capacidade individual de cada chip, medida em megabits (Mb), ou um milhão de bits. Pegue o resultado e divida-o por 8

para chegar ao número de megabytes nesse módulo. Por exemplo, 4x32 significa que o módulo tem 4 chips de 32 megabits. Multiplique 4 por 32 e obterá 128 megabits. Já que sabemos que um byte tem 8 bits, precisamos dividir nosso resultado de 128 por 8. Nosso resultado é 16 megabytes.

EXERCÍCOS

1) Cite o maior número possível de lugares do computador onde há memórias?

2) Porque as memórias são consideradas o gargalo da computação?

3) Qual a diferença entre memórias RAM e ROM?

4) Quais os tipos de memória ROM? Escreva um texto falando as características de cada uma delas.

5) Explique o que é uma memória RAM Dinâmica?

6) O que é o refresh? Porque ele é necessário?

7) Explique o que é uma memória RAM Estática?

10) Por que o computador precisa de tantos sistemas de memória diferentes?

11) Diferencia arquitetura Havard e Von Newman.

.

5 CPU

5.1 FORMATOS DE INSTRUÇÃO

As instruções de um programa compilado são armazenadas na memória sob a forma de um código em binário, ou código de instrução. Um código de instrução é logicamente formado por campos de bits, que contém as informações necessárias à execução da instrução. Esses campos de bits indicam, por exemplo, qual a operação a ser realizada e quais os operandos a serem usados. A Figura 11 mostra um exemplo de código de instrução com seus campos de bits.

0101110000	0010	0010	00011
Opcod	opf1	opf2	opdst

Figura 11 – Formatos de Instruções

Neste código de instrução, o campo opcod contém o código da operação a ser realizada, enquanto os campos opf1, opf2 e opdst indicam os operandos fonte e destino, respectivamente. Suponha que o código 01011100 no campo opcod indique uma operação de adição, e que os valores 00001, 00010 e 00011 nos campos opf1, opf2 e opdst indiquem os registradores R1, R2 e R3, respectivamente. Assim, este código de instrução é a representação binária da instrução

ADD R1,R2,R3.

O formato de instrução refere-se as características do código de instrução tais como tamanho do código, tipos de campos de bits e localização dos campos de bits dentro do código. Uma arquitetura se caracteriza por apresentar instruções com formato irregular ou com formato regular. No primeiro caso, as instruções podem apresentar códigos com tamanhos diferentes, e um certo campo de bits pode ocupar posições diferentes nas instruções onde aparece.

Em uma arquitetura com instruções regulares, todos os códigos de instrução possuem o mesmo tamanho, e um certo campo de bits sempre ocupa a mesma posição nas instruções onde aparece.

As arquiteturas com formatos de instrução irregular possibilitam programas com menor tamanho de código executável. Isto acontece porque aqueles campos de bits não necessários a uma instrução são eliminados, economizando espaço de armazenamento na memória.

Por outro lado, arquiteturas com formatos de instrução regular apresentam uma grande vantagem quanto à simplicidade no acesso às instruções. Se todas as instruções possuem um tamanho de n bits, basta que o processador realize um único acesso de n bits à memória principal para obter uma instrução completa. Considere agora um processador com códigos de instrução com tamanho variável. Neste caso, o processador não sabe, a priori, quantos bits deve buscar para obter uma instrução completa. Após realizar um acesso, torna-se necessário que o processador interprete parcialmente o código da instrução para determinar se deve realizar um outro acesso à memória para completar a busca da instrução.

A decodificação parcial e o acesso adicional podem comprometer o desempenho ao aumentar o tempo de execução da instrução.

A segunda vantagem de instruções regulares é a simplicidade na decodificação das instruções. Em instruções regulares, um certo campo de bits sempre ocupa a mesma posição. Isto permite, por exemplo, que os operandos da instrução sejam acessados ao mesmo tempo em que o código de operação é interpretado, já que o processador conhece antecipadamente onde encontrar as informações sobre os operandos. Em instruções irregulares, os campos que indicam os operandos podem aparecer em qualquer posição dentro do código da instrução.

Assim, é necessário antes interpretar o código de operação, para determinar as posições dos campos de operando dentro daquele código de instrução em particular. Agora, a decodificação e o acesso aos operandos são realizados seqüencialmente, o que contribui para aumentar o tempo de execução das instruções.

5.2 MODOS DE ENDEREÇAMENTO

Uma característica de um conjunto de instruções é o número de operandos explicitamente indicados em uma instrução aritmética ou lógica. Em algumas arquiteturas, estas instruções referenciam explicitamente três operandos, dois operandos-fonte e um operando-destino, como por exemplo em:

ADD R1, R2, R3

onde R1 e R2 são os operandos-fonte e R3 é o operando-destino. Em outras arquiteturas, instruções aritméticas/lógicas especificam apenas dois operandos. Neste caso, um dos operandos-fonte é também o operando-destino. Por exemplo, na instrução:

ADD R1, R2

R2 contém um dos operandos-fonte e também é usado como operando-destino. Quanto à localização dos operandos especificados por uma instrução aritmética/lógica, podemos encontrar arquiteturas onde podem ser realizados acessos aos operandos diretamente a partir da memória principal. Por exemplo, nestas arquiteturas podemos ter instruções tais como:

ADD M1,R1,R2
ADD M1,M2,R1
ADD M1,M2,M3

onde M1, M2 e M3 são endereços de locações de memória. Em um outro extremo, existem arquiteturas onde todos os operandos encontram-se apenas em registradores. As instruções aritméticas/lógicas são todas do tipo:

ADD R1,R2,R3
ADD R1,R2

A partir do número de operandos explicitamente referenciados e da localização destes operandos, podemos classificar as arquiteturas nos seguintes tipos:

• **arquiteturas memória-memória**: as instruções aritméticas/lógicas usam três operandos e todos os operandos podem estar na memória;

• **arquiteturas registrador-memória**: as instruções aritméticas/lógicas usam dois operandos, sendo que apenas um deles pode residir na memória;

• **arquiteturas registrador-registrador**: as instruções aritméticas/lógicas usam três operandos, todos em registradores. Neste caso, apenas duas instruções acessam diretamente a memória: LOAD e STORE. A instrução LOAD carrega em um registrador um dado armazenado na memória e instrução STORE armazena na memória o conteúdo de um registrador.

Arquiteturas memória-memória e registrador-memória apresentam como vantagem um menor número de instruções no código do programa, já que não é necessário carregar previamente em registradores os operandos-fonte de uma instrução aritmética/lógica, como acontece em uma arquitetura registrador-registrador. Por outro lado, a existência de instruções aritméticas/lógicas mais poderosas torna mais complexa a implementação da arquitetura. As arquiteturas Intel 80x86 e Motorola MC680x0 são do tipo registrador-memória.

Dentre as arquiteturas memória-memória podemos citar o DEC VAX 11.

Os operandos de uma instrução podem encontrar-se em registradores, na memória principal ou ainda embutidos na própria instrução. O modo de endereçamento refere-se à maneira como uma instrução especifica a localização dos seus operandos. Existem três modos de endereçamento básicos:

• modo registrador: a instrução indica o número de um registrador de dados onde se encontra um operando (fonte ou destino);

• modo imediato: a instrução referencia um operando que se encontra dentro do próprio código da instrução;

• modo implícito: a localização do operando não está explicitamente indicada na instrução.

Por exemplo, nas chamadas arquiteturas acumulador, um dos operandos-fonte e o operando-destino nas instruções aritméticas/lógicas encontra-se sempre em um registrador especial, o acumulador. Assim, não é necessário que este registrador seja explicitamente referenciado pela instrução.

A Figura 4.3 mostra exemplos de instruções que usam os modos de endereçamento implícito, registrador e imediato.

Modo	Exemplo	Significado
Implícito		
	ADD R1	Ac ← Ac + R1
Registrador		
	ADD R1,R2	R2 ← R1 + R2
Imediato		
	ADD R1,#4	R1 ← R1 + 4

Os modos de endereçamento citados referenciam apenas operandos que se encontram em registradores ou na instrução. Existem ainda os modos de endereçamento usados para referenciar dados armazenados na memória principal. Entre as diferentes arquiteturas, existe uma enorme variedade de modos de endereçamento referentes à memória principal, e que formam, na realidade, uma classe de modos de endereçamento à parte.

Um modo de endereçamento referente à memória indica como deve ser obtido o endereço da locação de memória onde se encontra o dado que será acessado. Este endereço é chamado endereço efetivo. Apesar da variedade mencionada, é possível identificar alguns modos de endereçamento referentes à memória que são oferecidos pela maioria das arquiteturas. Estes modos de endereçamento mais comuns estão relacionados a seguir:

Tabela 10 – Modos de Endereçamento

Modo	**Exemplo**	**Significado**	**Uso**
Direto	ADD (100),R1	R1 ← M[100] + R1	acesso à variáveis estáticas
Indireto	ADD (R1),R2	R2 ← M[R1] + R2	acesso via ponteiros
Relativo à base	ADD 100(R1),R2	R2 ← M[100+R1] + R2	acesso a elementos em estruturas
Indexado	ADD (R1+R2),R3	R3 ← M[R1+R2] + R3	acesso a elementos em um vetor

No modo direto, o endereço efetivo é um valor imediato contido no código da instrução. Por exemplo, na instrução. Por exemplo, na instrução ADD (100),R1, um dos operandos encontra-se na locação de memória com endereço 100. O modo de endereçamento direto é usado principalmente no acesso às variáveis estáticas de um programa, cujo endereço em memória pode ser determinado durante a compilação do programa.

No modo indireto, o endereço efetivo encontra-se em um registrador. Por exemplo, na instrução ADD (R1),R2, um dos operandos encontra-se na locação de memória cujo endereço está no registrador R1. Ou seja, o operando na memória é indicado indiretamente, através de um registrador que contém o endereço efetivo. Este modo de endereçamento é

usado no acesso a variáveis dinâmicas, cujo endereço na memória é conhecido apenas durante a execução do programa. O acesso a uma variável dinâmica é realizado através de um ponteiro, que nada mais é do que o endereço da variável. Para realizar o acesso à variável dinâmica, o ponteiro é carregado em um registrador, e a instrução que acessa a variável usa este registrador com o modo de endereçamento indireto.

No modo relativo à base, o endereço efetivo é a soma do conteúdo de um registrador, chamado endereço-base, com um valor imediato contido na instrução, chamado deslocamento. Por exemplo, na instrução ADD 100(R1),R2, R1 contém o endereço-base e 100 é o deslocamento. O endereço efetivo do operando em memória é a soma do conteúdo de R1 com o valor 100. O modo relativo à base é usado no acesso a componentes de variáveis dinâmicas estruturadas (por exemplo, record em Pascal ou struct em C).

No modo indexado, o endereço efetivo é dado pela soma de um índice com um endereço-base, ambos armazenados em registradores. Por exemplo, na instrução ADD (R1+R2),R3, R1 contém o endereço-base, e R2 o índice. O modo indexado é normalmente usado no acesso aos elementos de um vetor. A Figura 4.6 mostra como é calculado o endereço efetivo no modo de endereçamento indexado.

EXERCÍCIOS

1) A respeito de Formato de Instruções, o que é opcode? O que são operandos?

2) O que é um Modo de endereçamento?

3) Cite os modos de endereçamento citados na apostila, explicando cada um deles.

6. PROJETO DE MICROARQUITETURA

6.1 INTRODUÇÃO

A Unidade Central de Processamento (CPU) é um dos principais elementos que compõem a estrutura de um computador. Segundo Tanenbaum (1992)[1] a CPU é composta pela Unidade de Controle, Unidade Lógica e Aritmética e Registradores. A Unidade de Controle é responsável pela busca e decodificação de instruções, bem como pelo controle dos demais elementos que compõem a CPU. A Unidade Lógica e Aritmética faz operações lógicas como E (And), OU (Or), entre outras e operações aritméticas como adição e subtração. Os Registradores são pequenas áreas de armazenamento que funcionam como memórias de rascunho de informações, durante o processamento. Outro elemento importante que constitui a estrutura de um computador é a Memória, cuja função é armazenar programas e dados.

Dois registradores importantes são o Contador de Programa (*Program Counter* – PC) e o Registrador de Instrução (*Instruction Register* – IR). O PC aponta para a próxima instrução a ser executada, ou seja, ele contém o endereço de memória da próxima instrução. O IR tem a função de armazenar a instrução que está sendo executada.

O ciclo de execução de uma instrução de dado por uma série de passos:

1. Busca a próxima instrução da memória para o registrador de instrução;
2. Atualiza o contador de programa para que ele aponte para a instrução seguinte;
3. Determina o tipo de instrução;
4. Se a instrução usa dados da memória, determina onde eles estão;
5. Busca os dados, se houver algum, para registradores internos da CPU;
6. Executa a instrução;
7. Armazena os resultados em locais apropriados;

[1] TANENBAUM, Andrew S. **Organização Estruturada de Computadores**. Rio de Janeiro: LTC, 1992.

8. Volta ao passo 1 para iniciar a execução da próxima instrução.

6.2 OBJETIVO

O objetivo deste trabalho é desenvolver, documentar e simular no ***LogiSim*** o projeto de um processador de 4 bits, e elaborar programas que sejam executados pelo mesmo. A documentação deverá conter as formas de onda dos sinais de controle, as tabelas da verdade, mapas de Karnaugh, expressões lógicas simplificadas e circuitos lógicos. A simulação deverá conter macros para os diversos elementos que compõem o processador.

6.3 ARQUITETURA

A arquitetura do processador está apresentada na Figura 1. As setas largas indicam barramentos de dados. O traço oblíquo que corta a seta é acompanhado de um valor que indica o número de linhas do barramento, ou seja, o número de bits que trafegam simultaneamente no mesmo. As linhas estreitas indicam linhas de controle de apenas 1 bit. As setas indicam o sentido de tráfego das informações, identificando o transmissor e o receptor, bem como a existência de barramento bidirecional.

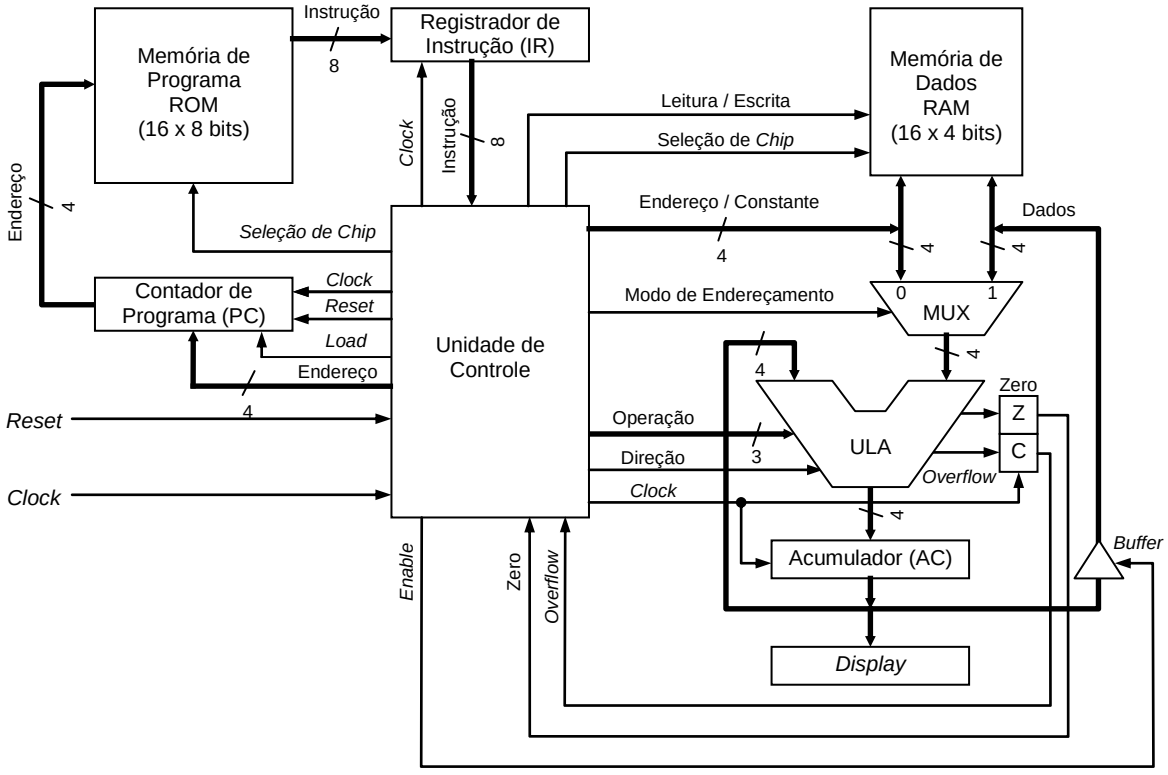


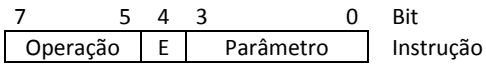
Figura 1 – Arquitetura do processador.

6.4 CARACTERÍSTICAS

Cada instrução do processador será composta por 8 bits, representando a operação a ser executada e os parâmetros necessários para sua execução. A quantidade de bits referentes à operação pode variar, assim como a quantidade e o tipo de parâmetros requeridos.

6.1.1 Formato das instruções lógicas e aritméticas

As instruções lógicas e aritméticas do processador terão o formato apresentado na Figura 2.



Onde:
Operação – é o código da operação a ser executada (3 bits);
E – é o modo de endereçamento a ser utilizado (1 bit);
Parâmetro – é o operando exigido pela operação especificada (4 bits).

Figura 2 – Formato das instruções lógicas e aritméticas.

As instruções lógicas e aritméticas suportam dois tipos de endereçamento: imediato e direto. No endereçamento imediato o próprio parâmetro da instrução será utilizado como dado (constante) para executar a operação. No endereçamento direto o parâmetro da instrução será um endereço da memória RAM que contém o dado a ser processado.

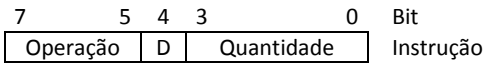
A tabela a seguir mostra o conjunto de instruções lógicas e aritméticas.

Código de Operação	Representação simbólica		Descrição simplificada
000	move	X, E	AC ← OP
001	add	X, E	AC ← AC + OP
010	sub	X, E	AC ← AC – OP
011	or	X, E	AC ← AC or OP
100	not	X, E	AC ← not OP

Onde:
AC – é o registrador acumulador da ULA;
OP – é o operando utilizado na operação;
X – é o parâmetro de 4 bits que compõe a instrução;
M(X) – é o conteúdo do endereço X da memória RAM;
E – se for igual a 0 indica que o parâmetro X é uma constante (ou seja, OP = X), se for igual a 1 indica que o parâmetro X é um endereço de memória RAM (ou seja, OP = M(X)).

6.1.2 Formato da instrução de deslocamento

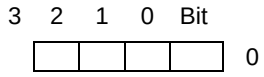
A instrução de deslocamento do processador terá o formato apresentado na Figura 3.



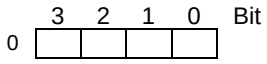
Onde:
Operação – é o código da operação a ser executada (3 bits);
D – é a direção do deslocamento (1 bit);
Quantidade – é a quantidade de bits que será deslocado (4 bits).

Figura 3 – Formato da instrução de deslocamento.

A instrução possibilita o deslocamento de um número para a direita ou para a esquerda. No deslocamento para a esquerda zeros são acrescentados como bit menos significativo do número, comportando-se como uma multiplicação do número na potência de 2 (veja figura 4a). No caso do deslocamento para a direita zeros são acrescentados como bit mais significativo do número, comportando-se como uma divisão inteira do número inicial por um número na potência de 2 (veja figura 4b).



(a) Deslocamento para esquerda.



(b) Deslocamento para direita.

Figura 4 – Deslocamento.

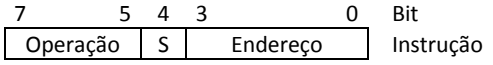
A tabela a seguir mostra a instrução de deslocamento do processador.

Código de Operação	Representação simbólica		Descrição simplificada
101	shift	Q, D	AC ← AC << Q, se D=0 AC ← AC >> Q, se D=1

Onde:
AC – é o registrador acumulador da ULA;
Q – quantidade de bits a ser deslocado;
D – se for igual a 0 promove um deslocamento para a esquerda, se for igual a 1 promove um deslocamento para a direita.

6.1.3 Formato da instrução de salto condicional

A instrução de salto condicional do processador terá o formato apresentado na Figura 5.



Onde:
Operação – é o código da operação a ser executada (3 bits);
S – é o tipo de salto condicional (1 bit);
Endereço – é o endereço da memória ROM que contém a instrução que será executada no próximo ciclo (4 bits).

Figura 5 – Formato da instrução de salto condicional.

A instrução suporta dois tipos de salto condicional, alterando o conteúdo do registrador PC, caso uma condição seja satisfeita. Nesta instrução será possível selecionar entre o salto caso o valor do registrador acumulador seja igual a zero ou caso seja não negativo.

A tabela a seguir mostra a instrução de salto condicional do processador.

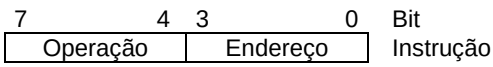
Código de Operação	Representação simbólica	Descrição simplificada
110	jump X, S	PC ← X (se S = 0 e AC = 0 ou S = 1 e O = 1)

Onde:
PC – registrador contador de programa;
AC – é o registrador acumulador da ULA;
X – é o endereço de 4 bits da memória ROM;
S – o salto ocorre se S e AC forem iguais a 0 ou se S for igual a 1 e ocorreu overflow

O Apêndice B mostra como efetuar comparações.

6.1.4 Formato da instrução de salto incondicional

A instrução de salto incondicional do processador terá o formato apresentado na Figura 6.



Onde:
Operação – é o código da operação a ser executada (4 bits);
Endereço – é o endereço da memória ROM que contém a instrução que será executada no próximo ciclo (4 bits).

Figura 6 – Formato da instrução de salto incondicional.

A instrução efetua um salto incondicional, alterando o conteúdo do registrador PC, que aponta para a próxima instrução a ser executada.

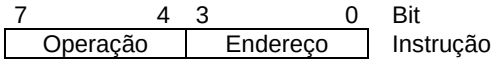
A tabela a seguir mostra a instrução de salto incondicional do processador.

Código de Operação	Representação simbólica		Descrição simplificada
1110	goto	X	PC ← X

Onde:
PC – registrador contador de programa;
X – é o endereço de 4 bits da memória ROM.

6.1.5 Formato da instrução de armazenamento

A instrução de armazenamento do processador terá o formato apresentado na Figura 7.



Onde:
Operação – é o código da operação a ser executada (4 bits);
Endereço – é o endereço da memória RAM receberá o valor a ser armazenado (4 bits).

Figura 7 – Formato da instrução de armazenamento.

A instrução efetua o armazenamento em uma determinada posição da memória RAM.

A tabela a seguir mostra a instrução de armazenamento.

Código de Operação	Representação simbólica		Descrição simplificada
1111	store	X	M(X) ← AC

Onde:
AC – é o registrador acumulador da ULA;
X – é o endereço de 4 bits da memória RAM;
M(X) – é o conteúdo do endereço X da memória RAM.

Uma descrição detalhada de cada instrução poderá ser vista no Apêndice A.

O Ciclo de Execução do processador ocorrerá em dois ciclos de *clock*. Veja a descrição completa do ciclo de execução na Figura 8.

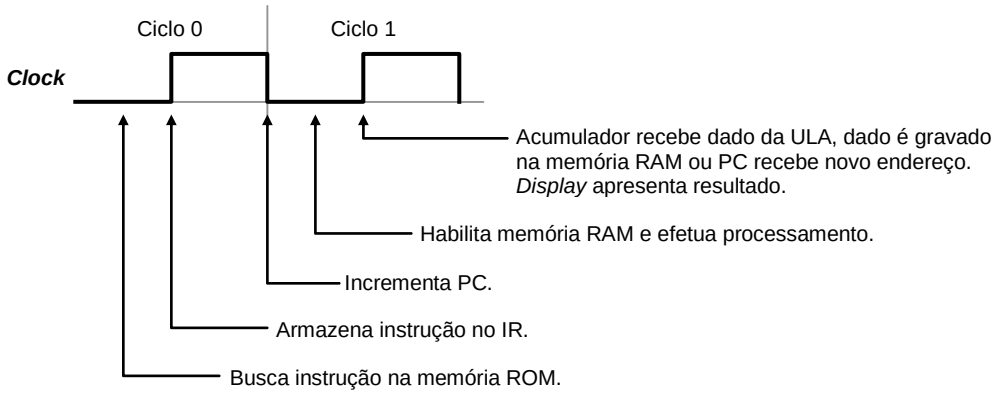


Figura 8 – Ciclo de execução do processador.

Considere ‘A’ uma variável de 4 bits, alocada na memória RAM, no endereço 4h. O programa deve efetuar a seguinte operação:

A = A + 5;

Ou seja,

M(4h) = M(4h) + 5;

A tabela a seguir mostra como o programa seria escrito em binário, além de sua alocação na memória ROM.

Endereço da ROM	Instrução em Binário	Representação simbólica	Descrição simplificada
00h	00010100	move 4h, 1	AC ← M(4h)
01h	00100101	add 5h, 0	AC ← AC + 5
02h	11110100	store 4h	M(4h) ← AC

6.5 ESTRUTURA DO PROCESSADOR

A ULA deste processador deverá ser capaz de executar operações com números de 4 bits, tendo como resultado outro número de 4 bits, mais uma indicação de estouro do número de bits (*overflow*). O item *3.1 Formato de Instruções* estabelece o código de operações que a ULA deverá interpretar, disponibilizando o resultado em sua saída. A ULA deverá determinar se o último resultado calculado é igual a zero (Z = 1) ou é diferente de zero (Z = 0). Portanto, a ULA receberá dois números de 4 bits e o código da operação (3 bits) a ser executada, fornecendo o resultado, um número de 4 bits, além do bit de Zero (Z) e do bit de *Overflow* (C). A Figura 1 apresenta a estrutura da ULA.

Este projeto prevê a utilizado de três registradores: PC, IR e AC. Os registradores PC e IR estão definidos no item *0. Introdução*. O valor do registrador PC será a saída de um contador crescente de 4 bits, que deverá efetuar a contagem a partir de um sinal de *clock*. O valor inicial deste contador será zero, devendo funcionar de modo cíclico indefinidamente. Seu valor poderá ser alterado para efetuar saltos de programa, para isto a unidade de controle deverá aplicar um endereço na entrada do registrador e habilitar o sinal de *load*. Este contador poderá ser zerado através de uma entrada de *reset*. O registrador IR terá 8 bits, possibilitando o armazenamento de uma instrução, sendo construído com flip-flops tipo D, que deverão receber um sinal de *clock* para armazenar a instrução. O registrador AC é o acumulador de resultados da ULA, ou registrador de saída da ULA. O registrador AC também tem a função de fornecer dados para a ULA, fazendo o papel de registrador de entrada da ULA. O dado será armazenado pelo acumulador a partir de um sinal de *clock*, que também será fornecido aos flip-flops D que armazenarão os bits de Zero e *Overflow*.

A Memória ROM tem a finalidade de armazenar o programa. Ela terá 4 bits de endereço e 8 bits de dados, sendo capaz de armazenar uma instrução em cada uma das 16 posições (2^4 = 16) de memória. Uma entrada de seleção de *chip* será utilizada para habilitar a memória.

A Memória RAM tem a finalidade de armazenar dados. Ela terá 4 bits de endereço e 4 bits de dados, que serão armazenados em cada uma das 16 posições de memória. Uma entrada de seleção de *chip* será utilizada para habilitar a memória e outra entrada será utilizada para selecionar se a operação é de leitura ou escrita de dados.

O Multiplexador de 2 canais de 4 bits será utilizado para selecionar a origem do dado aplicado à ULA. Tal escolha poderá ser feita através de uma entrada seletora de 1 bit, que definirá se o dado virá da memória RAM ou da instrução.

O *Buffer* de 4 bits, composto por elementos *tri-state*, será utilizado para transferir o conteúdo do registrador acumulador (AC) para a memória RAM. Uma entrada *enable* fará a habilitação do *buffer*, aplicando o valor de sua entrada em sua saída.

A Unidade de Controle comanda os diversos sinais de controle dos elementos que compõem o processador e as memórias, tais como sinais de *clock*, *enable* e seleção. Ela recebe o sinal de *clock* e a partir desta informação estabelece o ciclo de execução do processador, apresentado no item 3.2. Esta unidade receberá um sinal de *reset* para permitir que o usuário reinicie o funcionamento do processador, de tal forma que a instrução armazenada no endereço 0h da memória ROM seja executada. Os sinais de controle geralmente dependem do *clock*, do ciclo de execução e da operação. Veja o diagrama de blocos da figura 9.

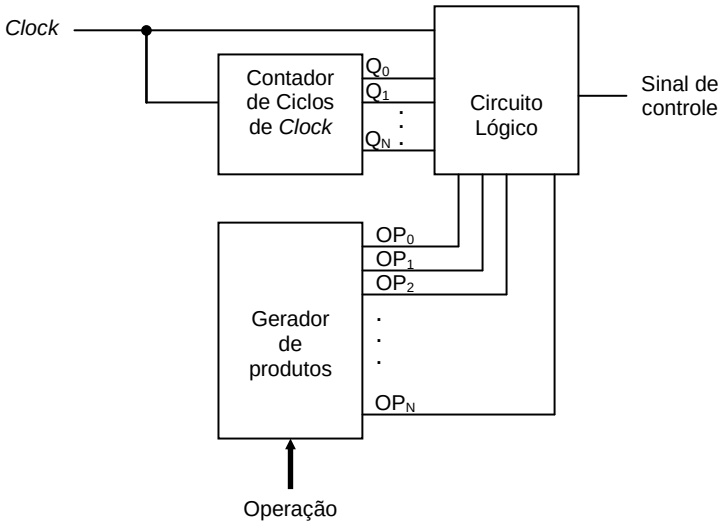


Figura 9 – Estrutura básica de circuito para geração de um sinal de controle.

Apêndice A

Descrição detalhada das instruções do processador.

move X, E

Descrição:	Efetua a operação de movimentação (cópia) de uma constante X, caso E seja igual a 0, ou do conteúdo do endereço X da memória RAM, para o registrador AC.
Limites:	0 ≤ X ≤ 15 (4 bits)
	E = 0 (OP = X) ou E = 1 (OP = M(X))
Operação:	AC ← OP
Bits afetados:	Z e O
Codificação:	000E XXXX

add X, E

Descrição:	Efetua a operação aritmética de adição entre AC e uma constante X, caso E seja igual a 0, ou a adição entre AC e o conteúdo do endereço X da memória RAM. O resultado sempre será armazenado em AC.
Limites:	0 ≤ X ≤ 15 (4 bits)
	E = 0 (OP = X) ou E = 1 (OP = M(X))
Operação:	AC ← AC + OP
Bits afetados:	Z e O
Codificação:	001E XXXX

sub X, E

Descrição:	Efetua a operação aritmética de subtração entre AC e uma constante X, caso E seja igual a 0, ou a subtração entre AC e o conteúdo do endereço X da memória RAM. O resultado sempre será armazenado em AC expresso em complemento de 2.
Limites:	0 ≤ X ≤ 15 (4 bits)
	E = 0 (OP = X) ou E = 1 (OP = M(X))
Operação:	AC ← AC – OP
Bits afetados:	Z e O
Codificação:	010E XXXX

or X, E

Descrição:	Efetua a operação lógica ‘OU’ (or) entre AC e uma constante X, caso E seja igual a 0, ou entre AC e o conteúdo do endereço X da memória RAM. O resultado sempre será armazenado em AC.
Limites:	0 ≤ X ≤ 15 (4 bits)
	E = 0 (OP = X) ou E = 1 (OP = M(X))
Operação:	AC ← AC or OP
Bits afetados:	Z e O
Codificação:	011E XXXX

xor X, E

Descrição:	Efetua a operação lógica ‘OU EXCLUSIVO’ (xor) de uma constante X, caso E seja igual a 0, ou do conteúdo do endereço X da memória RAM. O resultado sempre será armazenado em AC.
Limites:	0 ≤ X ≤ 15 (4 bits)

E = 0 (OP = X) ou E = 1 (OP = M(X))

Operação: AC ← AC xor OP

Bits afetados: Z e O

Codificação: 100E XXXX

shift X, D

Descrição: Efetua um deslocamento para direita ou para esquerda do acumulador Q bits. Caso D seja igual a 0 o deslocamento será para a esquerda, caso contrário para a direita. O resultado sempre será armazenado em AC.

Limites: 0 ≤ Q ≤ 15 (4 bits)

D = 0 (deslocamento para a esquerda) ou D = 1 (deslocamento para a direita)

Operação: AC ← AC << Q, para D = 0 ou

AC ← AC >> Q, para D = 1

Bits afetados: Z

Codificação: 101 D QQQQ

jmp X, E

Descrição: Efetua um salto condicional de programa de forma que a próxima instrução executada seja aquela alocada no endereço X ou no endereço já apontado pelo PC, em função da condição estabelecida. O salto é feito com a alteração do conteúdo do PC por X, se a condição for verdadeira. O salto pode ocorrer se AC for igual a zero (S = 0) ou se AC for não negativo (S = 1).

Limites: 0 ≤ X ≤ 15 (4 bits)

S = 0 (salta se AC = 0) ou S = 1 (salta se AC ≥ 0)

Operação: PC ← X, para S = 0 e se AC = 0 (ou seja, Z = 1) ou para S = 1 e O = 1 (ou seja,ocorreu overflow)

Bits afetados: nenhum

Codificação: 110 S XXXX

goto X

Descrição: Efetua um salto incondicional de programa para que a próxima instrução executada seja aquela alocada no endereço X da memória ROM. O salto é feito com a alteração do conteúdo do PC por X.

Limites: 0 ≤ X ≤ 15 (4 bits)

Operação: PC ← X

Bits afetados: nenhum

Codificação: 1110 XXXX

store X

Descrição: Efetua a operação de armazenamento do conteúdo do registrador AC para o endereço X da memória RAM.

Limites: 0 ≤ X ≤ 15 (4 bits)

Operação: M(X) ← AC

Bits afetados: nenhum

Codificação: 1111 XXXX

Apêndice B

Subtração
A subtração será realizada utilizando a instrução Sub cujo resultado é atribuído ao registrador AC.

Caso o resultado da subtração seja positivo ou zero, sempre existirá um *carry*. Caso o resultado seja negativo, não existirá *carry*.

Considerando uma subtração entre A e B temos:

A < B		A = B		A > B	
C	Z	C	Z	C	Z
1	0	0	1	0	0

O transporte de saída de um circuito subtrator de 4 bits é invertido em relação ao *flag* de *carry*.

Comparações
Uma comparação poderá ser realizada a partir de uma subtração, para ser utilizada em saltos de programas na criação de estruturas condicionais (Ex.: *if*) e estruturas de repetição (Ex.: *while*) a partir da operação *Jump*. Como o resultado de uma operação é armazenado no registrador AC então o salto poderá ocorrer se os valores envolvidos na subtração forem iguais (AC = 0) ou poderá ocorrer se os valores proporcionarem um resultado positivo (AC ≥ 0), para obter a comparação desejada deve-se estabelecer corretamente o valor de E.