ORGANIZAÇÃO DE COMPUTADORES

Representação da informação

O sistema binário foi escolhido para representar informações nos computadores de hoje devido a sua facilidade de implementação e confiabilidade nos dados, pois toda informação é representada a partir de dois símbolos, bastante discretos (0 e 1).

Veja os exemplos:

-Os meios magnéticos (disquete e HD`s) utilizam a polaridade magnética para representar os 0`s e 1`s pois dispõem de duas e somente duas polaridades Norte e Sul.

-Os meios ópticos representam a informação através de recepção ou não recepção da luz refletida na superfície do disco.

-Circuitos elétricos, representam informações fechando ou abrindo um contato.

-Memórias DRAM, armazenam cargas elétricas em capacitores, permitindo que seja atribuído o valor 1 (um) para capacitor carregado e 0 (zero) para capacitores descarregados.

Poderíamos citar vários exemplos para justificar a utilização do sistema binário, porém fugiria o nosso objetivo no momento.

Interessante frisar, que no momento em que a informação armazenada passa de um meio para outro (HD para memória, por exemplo) ela sempre será convertida em sinais elétricos. Estes sinais representam a informação por dois níveis bem discretos de voltagem, os quais dependendo do padrão utilizado poderão ser, por exemplo:

0 volt para “0” e 5 volts para “1”.

Ë dessa forma que a informação é transportada de um ponto para outro dentro do computador.

O conjunto de fios elétricos, utilizado no transporte e controle de dados dentro do computador, é comumente chamado de barramento e formam grupos bem distintos relacionados às funções comuns exercidas por eles. Podemos destacar 3 grupos, a saber:

-Barramento de dados- Conjunto de fios, por onde trafegam os conteúdos binários (instruções e dados).

-Barramento de endereço: Responsável por determinar o local onde uma operação (leitura ou escrita) será executada. O código binário contido neste conjunto de fios indica o endereço.

-Barramento de controle: Este conjunto de fios é utilizado para controlar o fluxo. Temporiza sincroniza e determina o tipo de operação que será executada em um determinado endereço.

Veja um exemplo:

Suponha que um resultado (X) que está no processador deverá ser armazenado no endereço $200_{(hexa)}$ de memória.

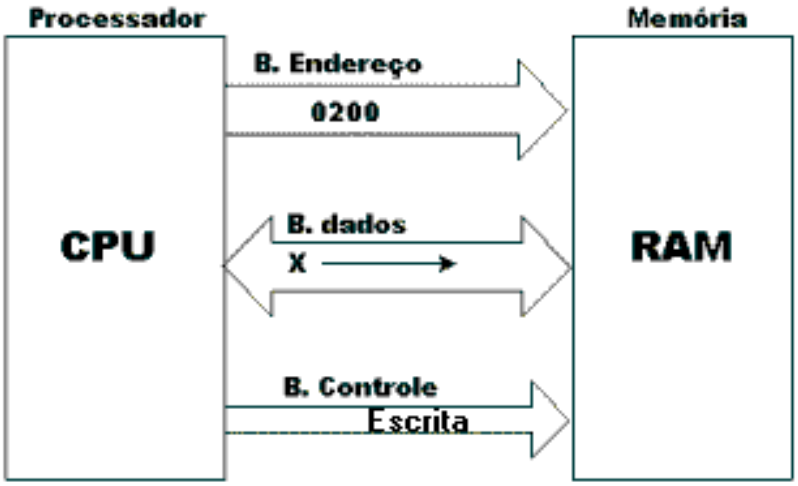
1 – O processador coloca o código $0200_{(hexa)}$ no barramento de endereços;

2 - Coloca o valor (X) a ser guardado na memória, no barramento de dados;

3 – Ativa a linha de controle para escrita;

4 – A memória responde, armazenando no endereço $0200_{(hexa)}$ o conteúdo binário correspondente a (X).

Veja figura a seguir.



Arquitetura simplificada da memória

A memória armazena conteúdos na forma binária, na grande maioria dos casos em cada endereço de memória são armazenados 8 bits (1 Byte). Logo, uma memória com 256 endereços, é capaz de armazenar 256 bytes.

É muito importante conhecermos a diferença entre o endereço de memória e o conteúdo deste endereço. Veja na tabela abaixo:

Memória	
Endereço (16 bits)	**Conteúdo** (8 bits)
0001111110010111	01001001
0001111110011000	11100101
0001111110011001	00000001
0001111110011010	11110110

Código binário no barramento de endereços

Conteúdo armazenado no endereço solicitado

Com o objetivo de simplificar a visualização e manipulação destes códigos, pois são números muito extensos, adotamos o código hexadecimal.

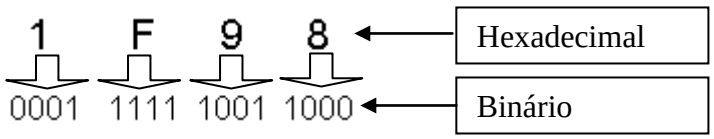
Desta forma podemos expressar a tabela acima da seguinte maneira:

Memória	
Endereço	Conteúdo
1F97	49
1F98	E5
1F99	01
1F9A	F6

Logo podemos dizer que o conteúdo do endereço $1F98_{(hexa)}$ é $E5_{(hexa)}$.

A seguir veja a conversão de hexadecimal p/ binário.

Observe que cada dígito Hexadecimal é representado Por 4 bits (dígitos binários)



Conteúdos de memória.

Já sabemos que os conteúdos dos endereços de memória são apenas códigos binários. O que significa cada um destes códigos armazenados na memória do computador?

O computador é um equipamento destinado a executar programas, e para que isto aconteça é necessário que este programa esteja armazenado na memória. Desta maneira, ao iniciar a execução de um programa, o processador irá até a memória e numa operação de leitura, trará para seu interior um código binário, que representa uma instrução (comando que define uma operação a ser executada).

Assim sabemos que as instruções que um processador executa são puramente códigos binários armazenados na memória. Porém, como praticamente todas as instruções executadas pelo processador são para manipular dados, estes dados também devem estar armazenados na memória permitindo que o processador o traga para seu interior para ser manipulado.

Concluímos então que programas são compostos de instruções que manipulam dados e que o conteúdo de uma posição de memória poderá ser um **dado** ou uma **instrução**.

Executando um programa.

Genericamente podemos dizer que um programa é executado em pequenos e seqüenciais passos conforme descrição a seguir:

1 A CPU coloca no barramento de endereços, o endereço da primeira instrução

2 A CPU ativa a linha de controle para leitura na memória

3 A memória responde colocando no barramento de dados o conteúdo do endereço solicitado.

4 A CPU captura este conteúdo e encaminha para a decodificação para determinar a operação a ser executada.

5 Se a operação requer um dado que está na memória, então calcula o endereço no qual o dado se encontra e coloca no barramento de endereços, ativa a linha de leitura, a memória responde colocando no barramento de dados o conteúdo solicitado.

6 A CPU então captura o dado e executa a instrução.

7 Terminada a execução da instrução, inicia nova faze indo até a memória buscar a próxima instrução.

Obs: *Ao iniciar a execução de um programa, o primeiro conteúdo binário buscado na memória será sempre uma instrução.*

Nomenclaturas

Com objetivo de visualizarmos a execução de instruções, utilizaremos um simulador onde poderemos escrever programas em linguagem de máquina ou de montagem e observar o seu desenvolvimento passo a passo .

Antes definiremos alguns termos utilizados:

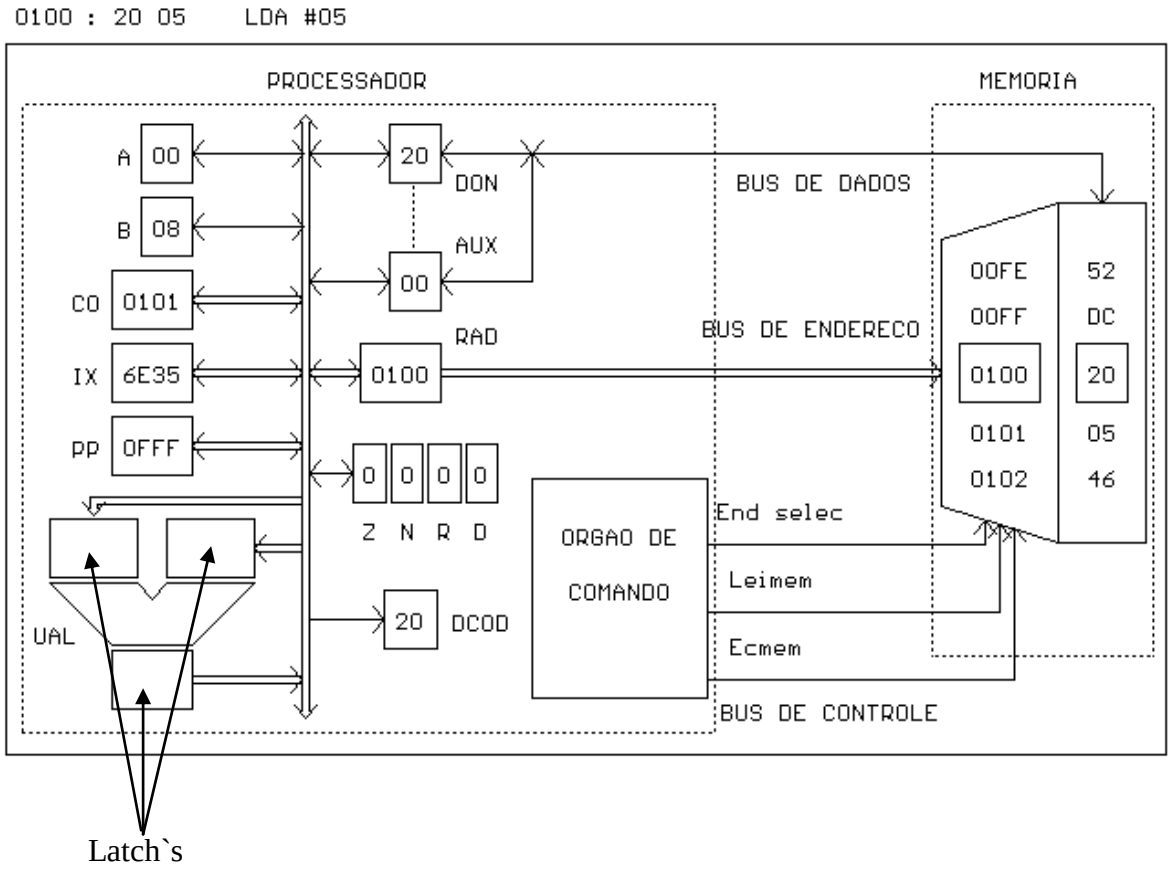
Código de operação (op code) : É o código binário de uma instrução convertido para hexadecimal. Este código define qual é a instrução e como ela será executada.

Mnemônico : Para simplificar o trabalho dos programadores, um software chamado compilador converte os comandos escritos na forma de abreviaturas(MNEMÔNICO) para códigos binários armazenando-os na memória. É mais fácil do que trabalhar com op code.

Operando : Conteúdo (dado) que será manipulado.

Modos de endereçamento : Maneira com que se busca um operando na memória ou seja as formas utilizadas para se determinar o endereço do operando. (consulte a apostila sobre instruções e endereçamento disponíveis no MICRO3).

MICRO 3



A arquitetura do MICRO 3 .

Barramentos:

- *B. de endereço* de 16 bits possibilitando acessar 2^{16} (64K) de posições de memória
- *B de dados* de 8 bits, permitindo a transferência de 1 byte (8bits) de cada vez entre memória e processador.
- *B. de controle:*

  -End selec: Indica que o endereço selecionado é válido
  -Leimem : Ativada quando a operação na memória é de leitura
  -Ecmem : Ativada quando a operação a ser realizada na memória é de escrita.

CPU : A CPU é composta de :

- Registradores:
  - **A** : Registrador de 8 bits - Armazena temporariamente um operando ou resultado de uma operação.
  - **B** : Registrador de 8 bits - Semelhante ao **A.**
  - **Co** : Contador de programa – Registrador de 16 bits - Seu conteúdo binário indica o endereço onde será buscada a próxima instrução - Normalmente o seu conteúdo é transferido para o registrador de endereço (RAD) para ser colocado no respectivo barramento.
  - **IX** : Registrador indexador – Registrador de 16 bits - Utilizado para determinar o endereço do operando no modo de endereçamento indexado.

- **PP** : Apontador de Pilha –Registrador de 16 bits – Seu conteúdo determina o topo da pilha nas operações com stack.
- **DON** : Registrador de dados – Registrador de 8 bits – Armazena temporariamente os conteúdos (dados e instruções) que entram e saem do processador.
- **AUX** : Registrador de dados – Registrador de 8 bits – Utilizado para formar juntamente com DON, um conteúdo de 16 bits antes de transferi-lo para outro registrador (de 16 bits)
- **RAD** : Registrador de endereço – Registrador de 16 bits – Armazena o código do endereço de memória que deverá ser acessado . No caso do MICRO 3 também serve para concatenar os conteúdos vindos de DON + AUX antes de ser transferido para outro registrador de 16 bits.
- **Código de condições: (flags)** registradores de 1 bit que funcionam sinalizando a ocorrência de eventos a saber:
  - **Z-** Indica quando o resultado é zero [z] = 1 quando resultado é zero
  - **N-** Indica resultado negativo
  - **R-** Indica a ocorrência de "cary" (vai um) após operação aritmética
  - **D** – Indica ocorrência de overflow (transbordo) em operações aritméticas

- Unidade de Controle : Neste modelo a unidade de controle está representada por:
  - **DCOD -** Decodificador de instrução – Tem como objetivo interpretar o código binário que representa a instrução, determinando como será executada.
  - **ORGÃO DE COMANDO** : Gera os sinais de controle para leitura ou escrita na memória.

- UAL : Unidade de Aritmética e Lógica – Local onde são executadas as operações de cálculo. A UAL possui 3 registradores de 16 bits chamados latchs, sendo 2 para os operandos de entrada e um para saída de resultado.

Memória

A memória do MICRO 3 consiste em 64K (65536) endereços, sendo que em cada endereço armazena 1 Byte(8 bits).

Dependendo do programa, o conteúdo de um determinado endereço poderá ser:

- Uma instrução
- Um operando
- O endereço de um operando (veja modos de endereçamento)

MEMÓRIA	
Endereço (hexa)	Conteúdo (hexa)
0100	**10**
0101	**30**
0102	**90**

Se o contador de programas aponta para o endereço 0100, então o conteúdo deste endereço é uma instrução (LDA direto) e o conteúdo dos endereços 0101 e 0102 informam o endereço do operando (3090).

Escrevendo programas para o MICRO 3.

Podemos escrever programas p/ o micro 3 de 2 maneiras diferentes:

- Utilizando MNEMÔNICOs:
  - Neste caso iniciamos com comando **C** seguido do endereço onde será gravada a primeira instrução. Veja exemplo:
    - C100
    - 0100 : LDA#8

Será armazenado no endereço 0100 o op code da instrução LDA (imediato) e no endereço 0101 será armazenado 08 (operando)

- Escrever em hexadecimal
  - Iniciamos com o comando **I** seguido do endereço de memória onde deve ser armazenado o código da instrução. Como normalmente os endereços de memória já possuem algum conteúdo gravado, este será exibido na frente do endereço e logo em seguida um sinal de ponto e vírgula (;). Após este sinal você edita o novo conteúdo hexadecimal do endereço. Exemplo:
    - I100
    - 0100 : D5 ; **20**
    - 0101 : 4F ; **08**

Os valores D5 e 4F são os conteúdos encontrados nos respectivos endereços. O valor **20** corresponde ao novo valor e que é o op code de LDA# e **08** o operando.

Exercício:

1. O mapa de memória mostra o conteúdo de alguns endereços. Qual o MNEMÔNICO da primeira instrução e o modo de endereçamento, sabendo que:
   a. **CO** = 0122 ?
   b. **CO =** 0123 ?
   c. **CO** = 0124 ?
   d. **CO** = 0125 ?

MEMÓRIA	
End. (hexa)	Conteúdo (h)
0122	50
0123	00
0124	20
0125	43

Respostas:

a. No end 0122 o conteúdo é 50 que corresponde ao op code de LDB direto. Neste caso o conteúdo dos end 0123 e 0124 informará o endereço do operando ou seja LDA 0020.
b. Se CO (contador de programa) está apontando para 0123 então o op code da próxima instrução será 00 que corresponde a instrução NOP (não opere) apenas gasta tempo e vai para a próxima instrução.
c. Com CO = 0124 o op code será 20 e a instrução LDA imediato, logo o conteúdo do end 0125 será o operando.
d. Para CO = 0125 teremos o op code 43 que corresponde a instrução PHB que é instrução de manipulação de pilha transfere o conteúdo do registrador B para memória no end indicado por **PP.**

Concluindo: Na memória temos apenas números binários.

O que eles significam, depende de como o programa está sendo executado, lembrando que o ponto de partida para executar um programa é o endereço informado pelo contador de programa que sempre aponta para próxima instrução a ser executada.

PILHA (STACK)

O micro 3 tem um registrador de 16 bits (PP) que como já foi mencionado seu conteúdo indica o endereço do topo da pilha.

A pilha não é nada mais que uma forma de ler ou gravar um conteúdo na memória no endereço referenciado por um apontador **PP.**

Operação de manipulação da pilha

- LDP : LOAD P (Carregar ponteiro) – Esta instrução atribui um valor a PP definindo o endereço do topo da pilha.
- PHA : PUSH A (Empurrar A) – Comando que copia o conteúdo do registrador A para o topo da pilha ou seja o conteúdo do registrador A será colocado na memória, no endereço indicado por PP.
    - Vale a pena frisar que cada vez que um conteúdo é empurrado para a pilha, ela cresce de tamanho sendo necessário um novo endereço para o topo sendo assim a instrução PHA além de copiar o conteúdo do registrador A para a memória ela decremento PP definindo um novo topo.

Ex: **PP** = 01F9 , **A** = D5

Antes do comando PHA		
	Endereço	conteúdo
	01F6	
	01F7	
	01F8	
Topo do stack	01F9	
	01FA	
	01FB	

Após o comando PHA		
	Endereço	conteúdo
	01F6	
	01F7	
Topo do stack	01F8	
	01F9	D5
	01FA	
	01FB	

$A \rightarrow M_{[PP]}$; $PP - 1 \rightarrow PP$

$A \rightarrow M_{[PP]}$: O conteúdo do registrador A será transferido para a posição de memória indicada por PP

$PP - 1 \rightarrow PP$ O Registrador PP será decrementado

- PPA : POP A (Puxar para A) – Comando que copia o último conteúdo empilhado para o registrador A .
    - Como toda vez que um conteúdo é empilhado, o PP é decrementado, significa que para obtermos o último conteúdo empilhado devemos primeiramente incrementar PP.

$PP + 1 \rightarrow PP$; $M_{[PP]} \rightarrow A$