ORGANIZAÇÃO BÁSICA DE COMPUTADORES E LINGUAGEM DE MONTAGEM

1. Conceitos básicos

Bit = BInary digiT = vale sempre 0 ou 1
elemento básico de informação

Byte = 8 bits processados em paralelo (ao mesmo tempo)

Word = n bytes (pedende do processador)

Double word = 2 words = 4 bytes = 32 bits

Nibble = 4 bits (utilidade para BCD)

Posição de bits:

Para 1 byte:

7	6	5	4	3	2	1	0
0	1	0	1	0	1	0	1

Para 1 word:

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

byte alto (high byte) | byte baixo (low byte)

Obs:
Words são armazenados em bytes consecutivos em memórias de 8 bits.
Byte baixo = byte inferior ou byte de menor ordem -> endereço N
Byte alto = byte superior ou byte de maior ordem -> endereço N+1

Memória

Memória: local do computador (hardware) onde se armazenam temporária ou definitivamente dados (números, caracteres e instruções)

Posição de memória ou endereço: localidade física da memória onde se encontra o dado.

Organização da memória:

Endereço	Conteúdo
...	...
4MB	10110101
...	...
1048576	01001010
...	...
1765	01001101
...	...
4	01010000
3	11111111
2	11101001
1	11011010
0	01100100

Obs: no 8086, a memória é organizada em bytes (conjunto de 8 bits).

1.1 Representação de números e caracteres

a) 1 byte

00100111b $= 0.2^7 + 0.2^6 + 1.2^5 + 0.2^4 + 0.2^3 + 1.2^2 + 1.2^1 + 1.2^0$
$= 0 + 0 + 32 + 0 + 0 + 4 + 2 + 1 = 39d$
$= 27h$

b) 1 word

0101011101101110b $= 0.2^{15} + 1.2^{14} + \ldots + 1.2^2 + 1.2^1 + 0.2^0$
$= 22382d$
$= 576Eh$ (mais fácil de representar!)

high byte = 0101 0111b = 57h
low byte = 0110 1110b = 6Eh

Tipo de conversão	Procedimento
Decimal => Binário	Divisões sucessivas por 2 até se obter zero no quociente; leitura dos dígitos binários de baixo para cima.
Binário = > Decimal	Soma de potências de 2 cujo expoente é a posição do bit e cujo coeficiente é o próprio bit.
Hexadecimal => Binário	Expandir cada dígito hexa em quatro dígitos binários segundo seu valor.
Binário => Hexadecimal	Compactar cada quatro dígitos binários em um único dígito hexa segundo seu valor.
Decimal => Hexadecimal	Divisões sucessivas por 16 até se obter zero no quociente; leitura dos dígitos de baixo para cima.
Hexadecimal => Decimal	Soma de potências de 16 cujo expoente é a posição do dígito e cujo coeficiente é o valor do próprio dígito hexa.

Representação sinalizada e não sinalizada de números:

Exemplo 1: 01110001b

valor não sinalizado $= 0.2^7 + 1.2^6 + 1.2^5 + 1.2^4 + 0.2^3 + 0.2^2 + 0.2^1 + 1.2^0$

$= 64 + 32 + 16 + 1 = 113d$

valor sinalizado bit de sinal = **0** => **" + "** (positivo)

$= 1.2^6 + 1.2^5 + 1.2^4 + 0.2^3 + 0.2^2 + 0.2^1 + 1.2^0$

$= 64 + 32 + 16 + 1 = 113d$ logo: +113d

Exemplo 2: 10110001b

valor não sinalizado $= 1.2^7 + 0.2^6 + 1.2^5 + 1.2^4 + 0.2^3 + 0.2^2 + 0.2^1 + 1.2^0$

$= 128 + 32 + 16 + 1 = 177d$

valor sinalizado bit de sinal = **1** => **" - "** (negativo)

10110001 <= número em complemento de 2 (C2)

11001110 <= número invertido menos bit de sinal

\+ 1

11001111 <= número em sinal e magnitude

magnitude (7 bits) $= 1.2^6 + 0.2^5 + 0.2^4 + 1.2^3 + 1.2^2 + 1.2^1 + 1.2^0$

$= 64 + 8 + 4 + 2 + 1 = 79d$ logo: - 79d

Obs: Usa-se o mesmo procedimento para palavras de 16 bits (word).

Exemplo 3: **70FFh** = 0111000011111111b

valor não sinalizado = $0.2^{15} + 1.2^{14} + ... + 1.2^{2} + 1.2^{1} + 1.2^{0}$

valor sinalizado bit de sinal = **0** => **" + "** (positivo)

= valor igual ao não sinalizado

Exemplo 4: **C777h** = 1100011101110111b

valor não sinalizado = $1.2^{15} + 1.2^{14} + ... + 1.2^{2} + 1.2^{1} + 1.2^{0}$

bit de sinal = **1** => **" - "** (negativo)

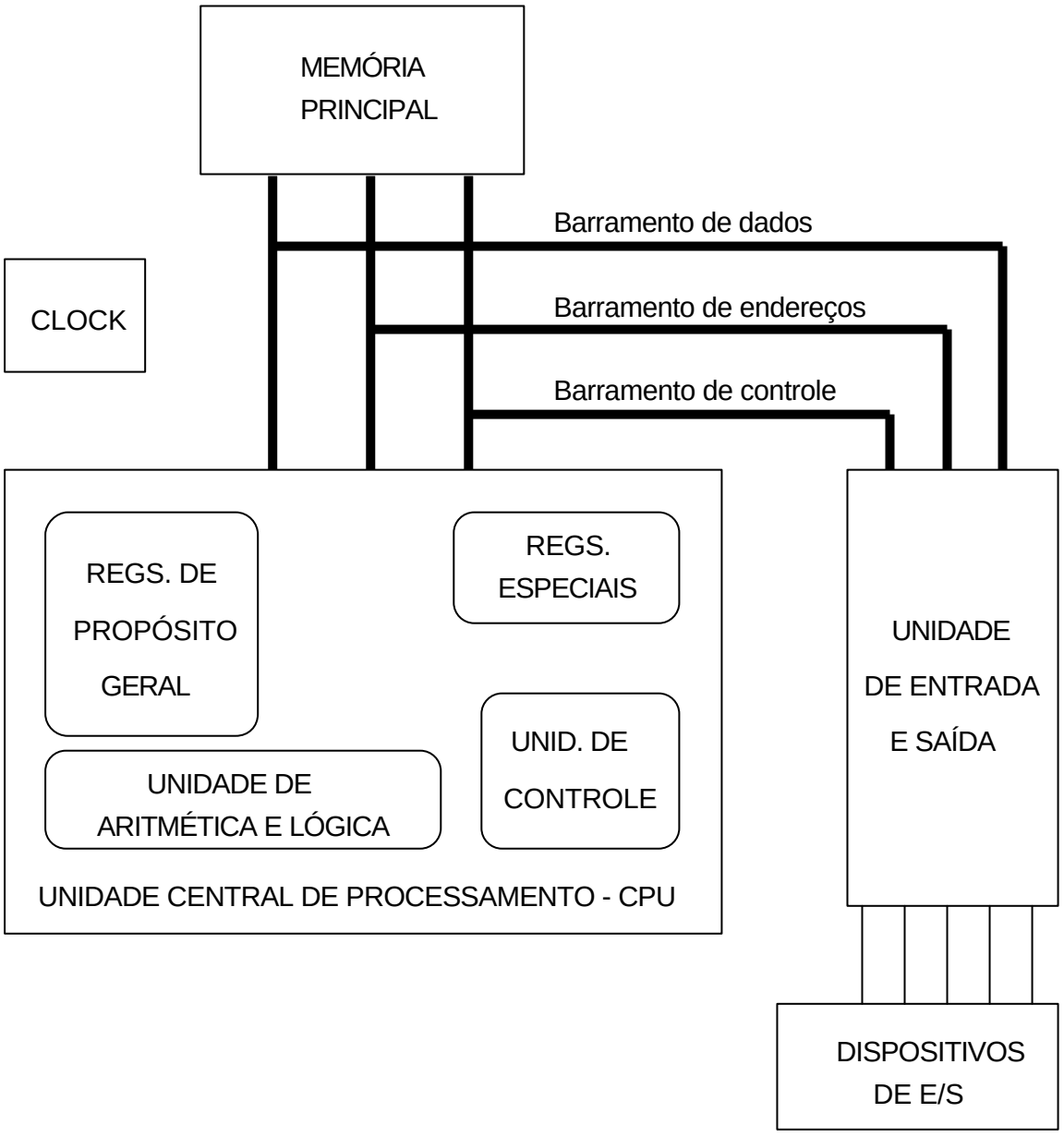
valor sinalizado = a ser calculado considerando o número em C2

Alguns números em hexadecimal, em representação sinalizada:

0FFFh positivo, 2 bytes
EF0h positivo, 3 dígitos hexa, 2 bytes, byte alto 0Eh, byte baixo F0h
1h positivo, 1 byte, 01h
80h positivo, se 2 bytes => 0000000010000000b
negativo, se 1 byte => 10000000b
8000h negativo, byte alto 80h, byte baixo 00h
C321h negativo
FFFFh negativo

1.2 Organização de um computador digital

Um computador digital pode ser descrito de uma forma simplificada segundo o diagrama de blocos:



- Unidade Central de Processamento: - UCP ou CPU
    - Unidade de Controle - UC
    - Unidade de Aritmética e Lógica - UAL ou ULA
    - Registradores de Propósito Geral - RPG
    - Registradores Específicos - RPE
        - Reg. de Dados da Memória - RDM
        - Reg. de Endereços da Memória - REM
        - Contador de Programa (*Program Counter*) - PC
        - Registrador de Instruções - RI
        - Apontador de Pilha (*Stack Pointer*) - SP
        - Outros (conforme a CPU)

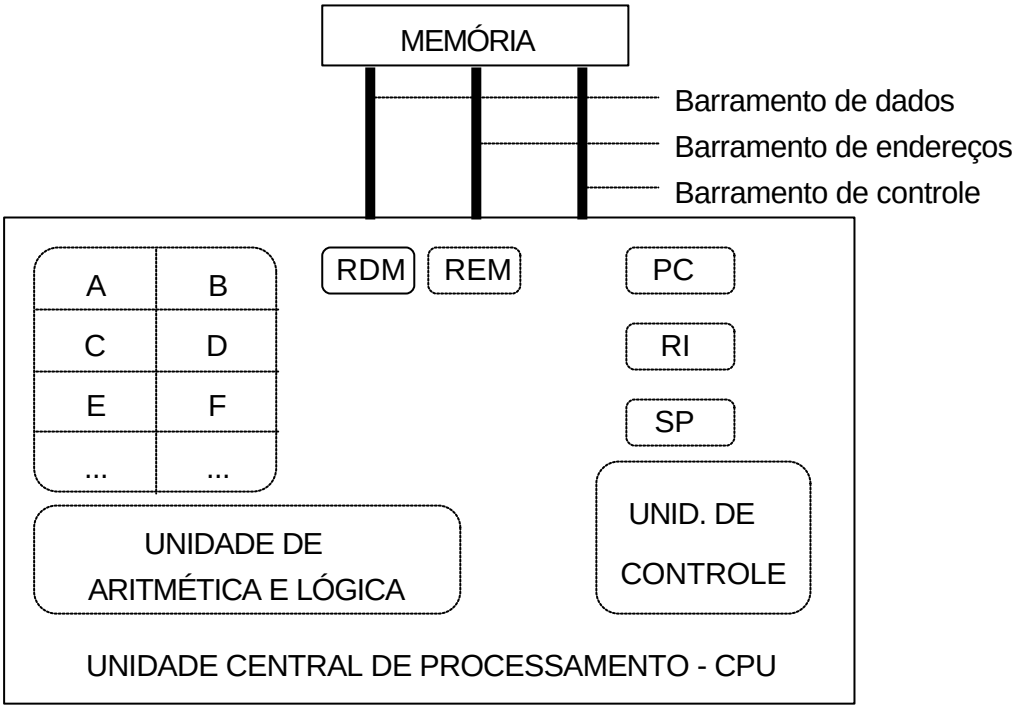
- Memória Principal

- Memória Secundária

- Unidade de E/S
    - Interfaces
    - Canais de E/S
    - Processadores de E/S

- Dispositivos de E/S

1.3 A CPU e a execução de um programa



Ciclo de busca e execução de uma instrução

1a. etapa: Busca da instrução na memória ("*FETCH*")

REM <-- PC
read
RDM <-- (REM)
PC <-- PC + n

2a. etapa: Decodificação da instrução

RI <-- RDM
ocorre a decodificação do conteúdo de RI na UC

3a. etapa: Busca dos operandos da instrução (se houver)

REM <-- endereço do operando
read
RDM <-- (REM)
PC <-- PC + n

4a. etapa: Execução da instrução – depende da instrução

Exemplo:

O que acontece com os registradores da CPU na execução da seguinte instrução: **MOV AL,5h** ?

Seqüência de acontecimentos:

PC <-- 01h

REM <-- PC fetch
RDM <-- (REM)
PC <-- PC + 2

RI <-- RDM decod.
Decodificação de MOV

REM <-- PC busca
RDM <-- (REM) operando
PC <-- PC+2

AL <-- RDM execução

Memória

Endereço	Conteúdo
01h	MOV AL
02h	5h
03h	....
04h	....
...	

1.4 Linguagem de máquina e linguagem montadora

Exemplo de um mesmo programa em linguagens de máquina e montadora:

Instrução de máquina (binário)	Operação
10100001 00000000 00000000	Busca o conteúdo da palavra de memória 0 e o coloca no reg. AX
00000101 00000100 00000000	Adiciona 4 ao reg. AX
10100011 00000000 00000000	Armazena o conteúdo de AX na palavra de memória de endereço 0

Instrução em linguagem montadora	Comentários
MOV AX,A	; busca o conteúdo da posição de ; memória dada por A e o coloca no ; reg. AX
ADD AX,4h	; adiciona 4 a AX, resultado em AX ;
MOV A, AX	; armazena o conteúdo de AX na ; posição de memória definida por A

Observa-se que:

- para cada instrução em linguagem montadora corresponde apenas uma instrução em linguagem de máquina;
- uma instrução em linguagem de máquina pode corresponder a mais de um byte;
- a programação em linguagem de máquina é tediosa e suscetível a erros.

Linguagens de alto nível

- a Linguagem Montadora tem o seu conjunto de instruções muito primitivo;
- torna-se impossível entender os algorítmos, a menos que se façam comentários;

Linguagens de Alto Nível nasceram para expandir as possibilidades de programação, numa forma textual mais próxima a humana.

Necessitam de tradução para a linguagem de máquina, por meio de **compiladores** (mais comum) ou **interpretadores**.

Exemplos: Pascal, C, Fortran (utilizam compiladores)
Prolog, Basic. (utilizam interpretadores)

Vantagens das linguagens de alto nível:

- próximas às linguagens naturais (facilidade de conversão de algorítmos);
- menor tempo de codificação e menor quantidade de linhas de programa;
- um programa em linguagem de alto nível roda em qualquer computador, desde que haja um compilador ou interpretador adequado (**portabilidade**).

Vantagens da linguagem montadora

- alta eficiência: produz programas em linguagem de máquina que podem ser mais rápidos e menores do que os obtidos por outras linguagens
- algumas operações são facilitadas, como escrever em posições de memória específicas e portas de E/S, sendo praticamente impossível em linguagens de alto nível;
- partes críticas de um programa podem ser escritas em *Assembly*;

- somente pela linguagem montadora se pode ter uma noção de como o computador "pensa" e porque certas coisas acontecem.

Linguagem de montagem: repertório de instruções

- instruções de entrada e saída
  - leitura
  - escrita
- instruções de transferência de dados
  - movimentação
  - troca
- instruções aritméticas
  - soma
  - subtração
  - multiplicação
  - divisão
- instruções lógicas
  - AND
  - OR
  - NOT
  - XOR
- instruções de desvios
  - loop
  - incondicionais
  - condicionais, se:
    - maior
    - menor
    - igual
    - maior ou igual
    - menor ou igual
- instruções de deslocamento e rotação
- pseudo-instruções (diretivas ou operadores)

Elementos básicos da linguagem de montagem

- Rótulo (*Label*): necessários para identificação de saltos;
  devem ser alfanuméricos começando por letras;

- Código de operação (**mnemônico**): especifica o tipo de instrução;

- Operandos (argumentos):

  Operandos:
  - registradores
  - endereços de memória
  - dados

- Comentário: forma de declarar a natureza da idéia codificada.

Exemplo de linhas de programa *assembly* do 8086:

[Rótulo:]	**[Cod. oper.]**	**[Operando(s)]**	**[;Comentário]**
loop:	MOV	AX,25h	; inicializa AX com 25h
	ADD	AX,AX	; AX <-- AX + AX
	DEC	contador	; contador <-- contador - 1
	JNZ	loop	

Instruções de N operandos

- Instruções de 2 operandos:

ADD AX,BX -> Soma o conteúdo do registrador AX com o do registrador BX, o resultado permanecendo em AX.

- instruções de 1 operando:

INC CX -> O conteúdo do registrador CX é incrementado de 1 unidade.

- Instruções de 0 operandos:

HLT -> (halt) Pára completamente o processador.

Obs: Existem linguagens assembly de outros processadores que permitem instruções com 3 operandos:

MPY E1,E2,E3 -> Multiplica o conteúdo do endereço E2 com o conteúdo do endereço E3, armazenando o resultado no endereço E1.

1.5 Introdução aos montadores (*assemblers*)

- **Função:**

Gerar a linguagem de máquina de um programa escrito em uma linguagem simbólica (linguagem de montagem, onde os símbolos são os mnemônicos).

- **Características desejáveis:**

  - Permitir representações simbólicas de rótulos, códigos de instrução e operandos.

  - Aceitar comentários, para facilitar a documentação.

  - Aceitar representação de números em várias bases numéricas.

  - Aceitar **pseudo-instruções**, que definem localizações (rótulos), reservam memória, equacionam símbolos, etc., mas não geram código de máquina.

- **Processo de montagem:**

  - Conversão de códigos mnemônicos em seus equivalentes códigos binários.

  - Conversão de números decimais, hexadecimais e octais em seus equivalentes binários.

  - Atribuição de endereços para as instruções do programa principal e das subrotinas.

- Atribuição de endereços na memória para as palavras de dados.

- **Diretivas (ou Diretrizes) ou pseudo-instruções:**

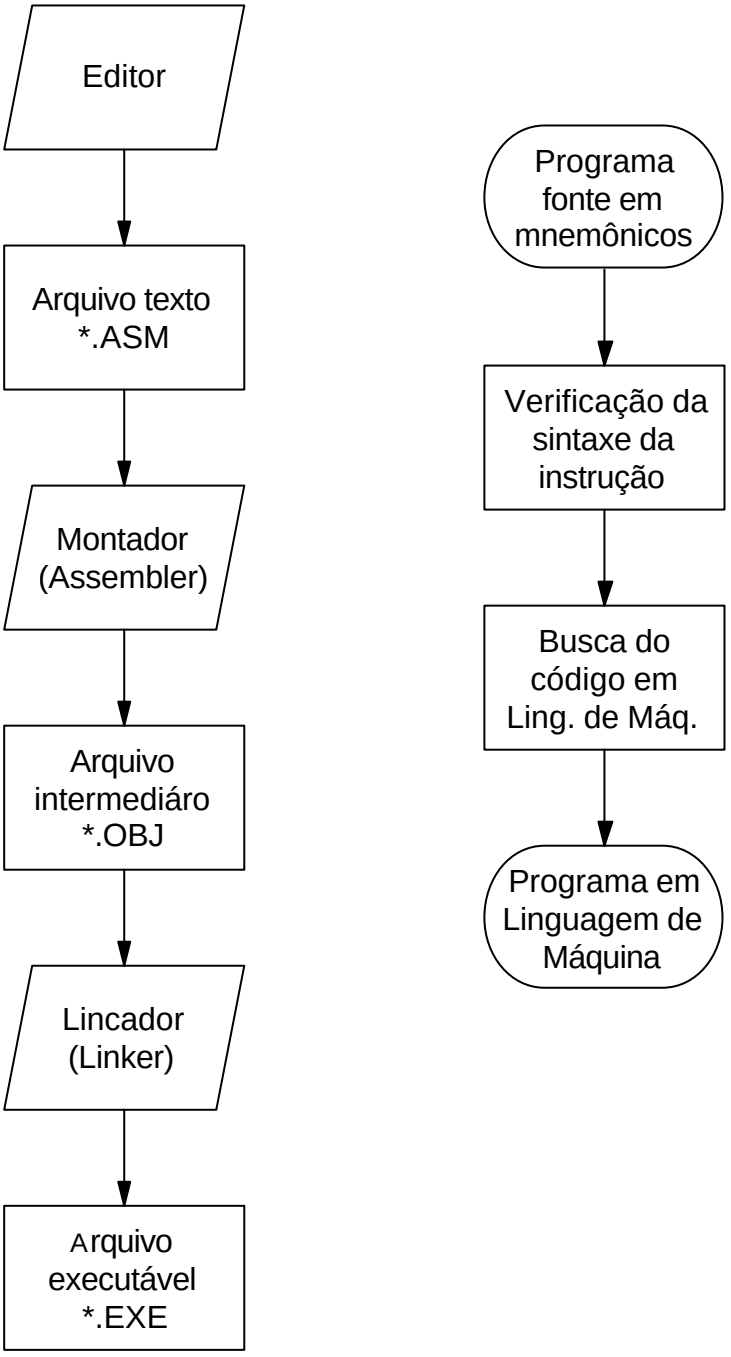
Códigos especiais para o montador, que não geram código de máquina, sendo geralmente instruções para:

- Definição de formato de dados
- Definição de espaços de memória (dados, pilha)
- Criação de subrotinas
- Determinação do tipo do processador e co-processador

- **Macro instruções:**

São instruções simbólicas definidas pelo usuário, que englobam várias instruções em linguagem de montagem. Quando invocadas, são traduzidas pelo montador (expandidas) em mais de um código de máquina.

O processo de montagem



Exemplo de um programa em linguagem montadora

```
TITLE   PGM1_1: PROGRAMA DE AMOSTRA
.MODEL  SMALL
.STACK  100H
.DATA
A       DW      2       ;definicao e inicializacao da variavel A
B       DW      5       ;definicao e inicializacao da variavel B
SUM     DW      ?       ;definicao da variavel SUM
.CODE
;inicializacao de DS
        MOV AX,@DATA
        MOV DS,AX       ;inicializa DS
;soma dos numeros
        MOV AX,A        ;AX recebe o conteudo de A
        ADD AX,B        ;AX contem A+B
        MOV SUM,AX      ;variavel SUM recebe o conteudo de AX
;conversao de hexa para ASCII
        ADD SUM,30H     ;somando 30h para compatibilizar a
                        ;quatidade em SUM com o caracter ASCII
;visualizacao do resultado na tela
        MOV AH,02H      ;funcao DOS para exibicao de caracter
        MOV DX,SUM      ;dado pronto para sair em DL
        INT 21H         ;exibe na tela
;saida do DOS
        MOV AH,4CH      ;funcao de retorno para o DOS
        INT 21H         ;saida para o DOS
        END
```