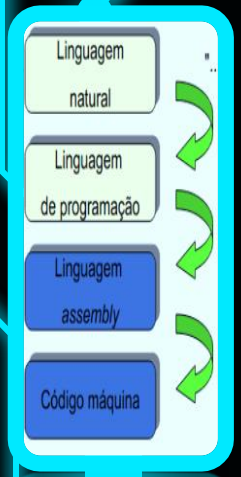
UNIVERSIDADE DE LUANDA
Instituto de Tecnologias de Informação e Comunicação



DISCIPLINA DE ARQUITECTURA DE COMPUTADORES I

Linguagem de Montagem e Linguagem de Máquina Assembly

Docente: engº. Nzuzi Rodolfo

CONCEITOS BÁSICOS

Linguagem/código de máquina

- Instruções que o processador é capaz de executar. Essas instruções, chamadas de código de máquina, são representadas por sequências de bits, normalmente limitadas pelo número de bits do registrador principal (8, 16, 32, 64 ou 128) da CPU.

Linguagem/código de máquina

- Notação legível por humanos para o código de máquina que uma arquitetura de computador específica utiliza

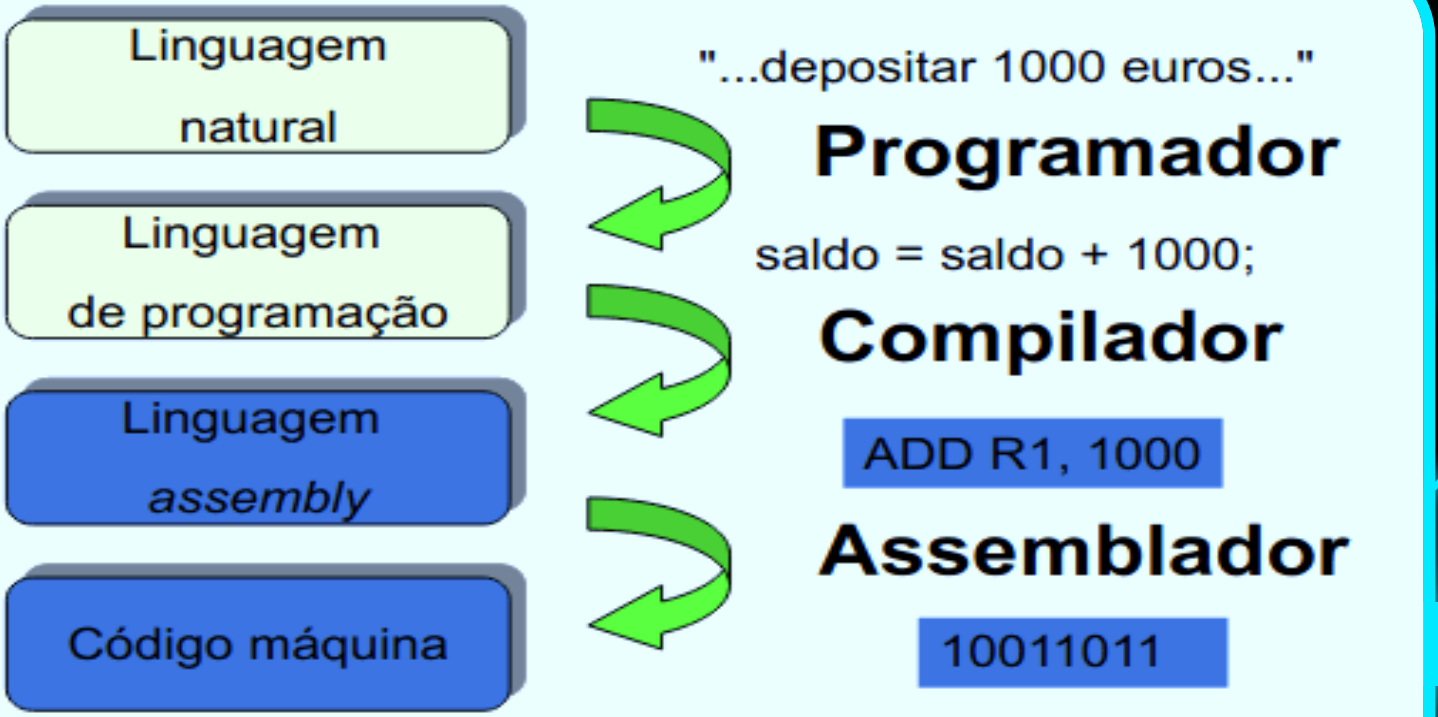
CONCEITOS BÁSICOS

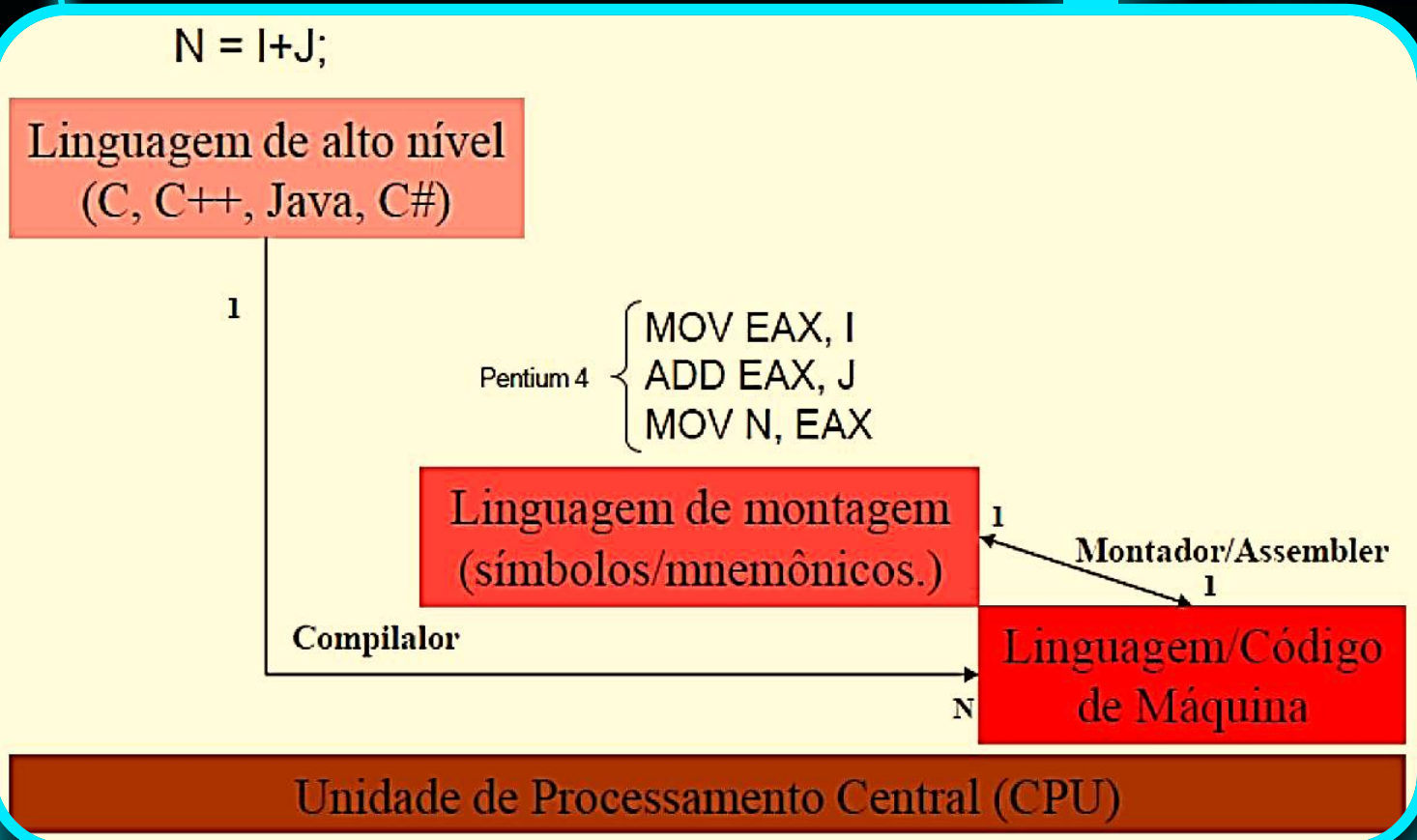
Tradutor ou compilador

- Programas que convertem um programa usuário escrito em alguma linguagem (fonte) para uma outra linguagem (alvo).

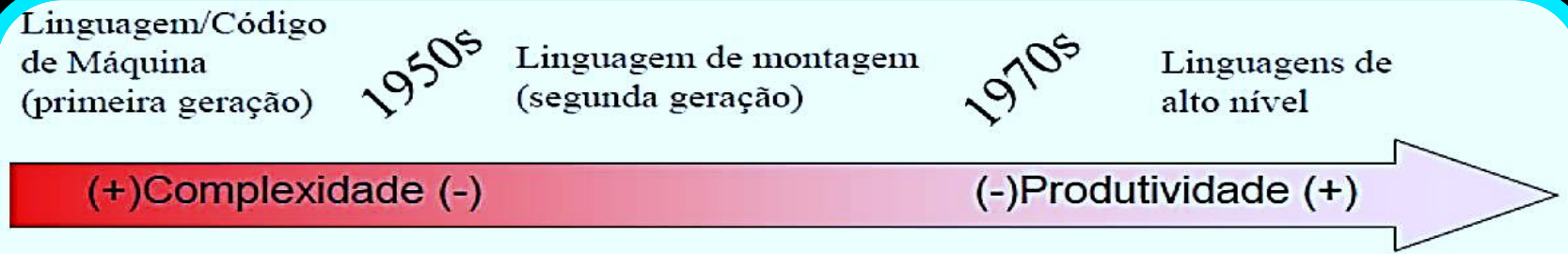
Montador/Assembler

- Um tradutor onde a linguagem fonte é a linguagem de montagem e a linguagem alvo é a linguagem de máquina


Linguagem natural
Linguagem de programação
Linguagem assembly
Código máquina
"...depositar 1000 euros..."
Programador
saldo = saldo + 1000;
Compilador
ADD R1, 1000
Assemblador
10011011


N = I+J;
Linguagem de alto nível
(C, C++, Java, C#)
1
Pentium 4
MOV EAX, I
ADD EAX, J
MOV N, EAX
Linguagem de montagem
(símbolos/mnemônicos.)
1
Montador/Assembler
1
Compilalor
N
Linguagem/Código
de Máquina
Unidade de Processamento Central (CPU)

HISTÓRIA



Por que usar linguagem de montagem?

Código menor e mais rápido

- Cartão inteligente
- Drivers

- Tratadores de interrupções de baixo nível em um S.O.
- Controladores em sistemas embutidos de tempo real

ARQUITETURA

O que é a arquitetura x86 ?

Essa arquitetura nasceu no 8086, que foi um microprocessador da Intel que fez grande sucesso.
Daí em diante a Intel lançou outros processadores baseados na arquitetura do 8086 ganhando nomes como: 80186, 80286, 80386 etc. Daí surgiu a nomenclatura 80x86 onde o x representaria um número qualquer, e depois a nomenclatura foi abreviada para apenas x86.
Nos dias atuais a Intel e a AMD fazem um trabalho em conjunto para a evolução da arquitetura, por isso os processadores das duas fabricantes são compatíveis.

LINGUAGEM MONTADORA (ASSEMBLY)

Por que programar em Assembly ?

O código em Assembly pode ser mais rápido e menor do que o código gerado por compiladores.Assembly permite o acesso direto a recursos do hardware, o que pode ser difícil em linguagens de alto nível.
Programar em Assembly permite que se ganhe um conhecimento profundo de como os computadores funcionam.
Saber programar em Assembly é muito útil mesmo que nunca se programe diretamente nele.

REQUISITOS PARA APRENDER ASSEMBLY

- conhecimento de alguma linguagem de programação de alto nível (java, básico, c/c++, pascal...) isso pode o ajudar muito.

- •É assumido que você tem um pouco de conhecimento sobre representação de número (hex/bin), se não era recomendado para estudar numerando seminário de sistemas altamente antes de você prosseguir…

Introdução á Linguagem Montadora 8086

A sintaxe assembly do 8086:

A SINTAXE ASSEMBLY DO 8086:

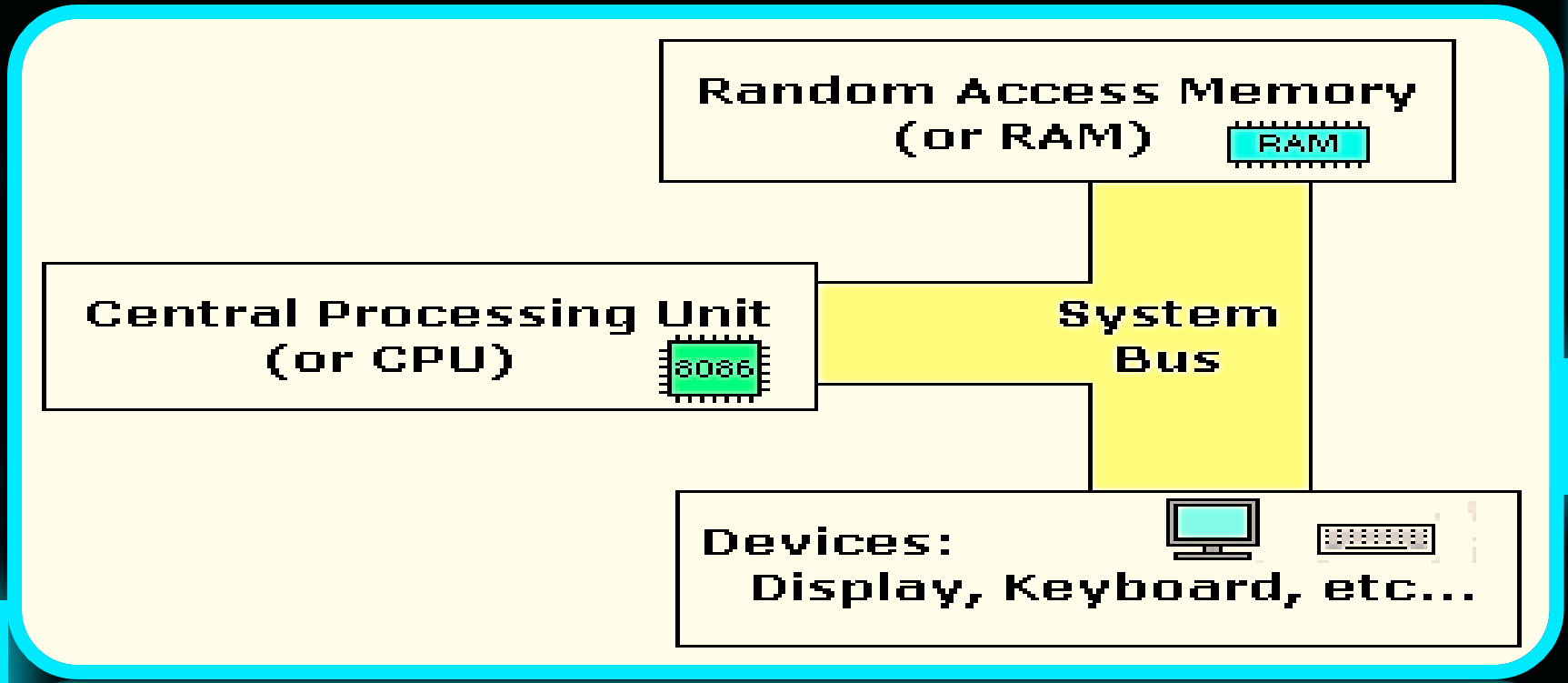
Linguagem montadora não é sensível à letra maiúscula ou minúscula
Para facilitar a compreensão do texto do programa, sugere-se.

- **uso de letra maiúscula para código**
- **uso de letra minúscula para comentários**

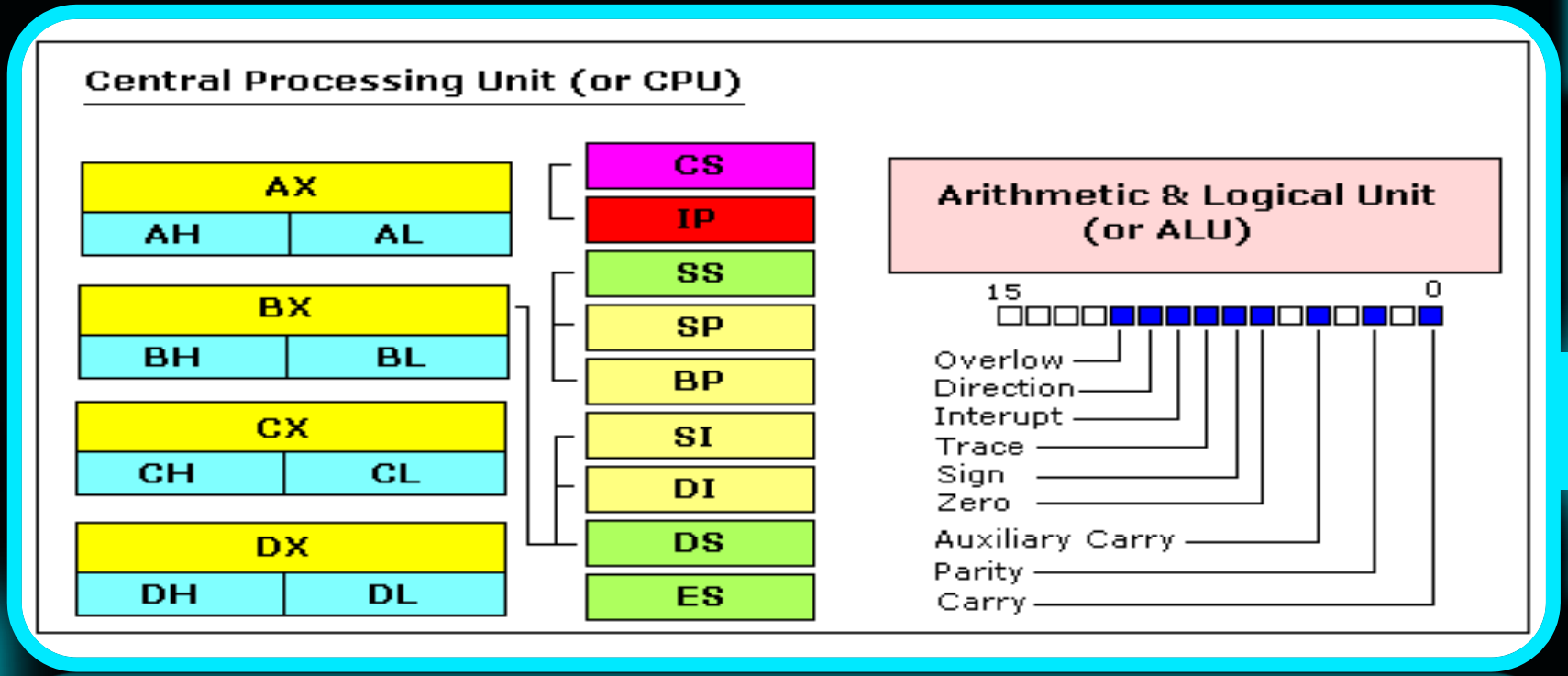
Para construirmos os programas em Assembly, devemos estruturar o fonte da seguinte forma (usando emu8086 como montador)

```
;Engenheiro Nzuzi Rodolfo

; Meu programa

org 100h ;

; Todo Codigo escreva aqui



ret
```

O MODELO DE COMPUTADOR SIMPLES



DENTRO DA CPU



REGISTRADORES DE USO GERAL

AX	BX	CX	DX	(16 bits)
AH/AL	BH/BL	CH/CL	DH/DL	(8 bits)

- AX: Acumulador ->dividido em AH / AL Usado em operações aritméticas.
- BX: Base ->dividido em BH / BL Usado para indexar tabelas de memória (ex.: índice de vetores).
- CX: Contador -> dividido em CH / CL Usado como contador de repetições em loop e movimentação repetitiva de dados.
- DX: Dados ->dividido em DH / DL Uso geral

REGISTRADORES DE USO GERAL

AX	BX	CX	DX	(16 bits)
AH/AL	BH/BL	CH/CL	DH/DL	(8 bits)

Ligadas a bx índice de vetores.

- SI - registrador de índice de fonte.
- DI - registrador de índice de destino.
- BP - ponteiro báse.
- SP - ponteiro pilha.
- Apesar do nome de um registro, é o programador que determina o uso para cada registro de propósito geral. O propósito principal de um registro é armazenar um número (variável).

REGISTRADORES DE SEGMENTO

- **CS** - pontos ao segmento que contém o programa atual.
- **DS** - geralmente pontos a segmento onde variáveis estão definidas.
- **ES** - registradores de segmento extra, está até um coder definir seu uso.
- **SS** - pontos ao segmento que contém a pilha

Embora é possível armazenar qualquer dados no registrador de segmento , esta nunca é uma idéia boa. Os registradores de segmento têm um propósito muito especial - apontando a blocos acessíveis de memória.).

REGISTRADORES DE SEGMENTO

Registradores de Segmento trabalham junto com registrador de propósito geral para aceder qualquer valor na memória

REGISTRADORES DE PROPÓSITO ESPECIAIS

- IP - o ponteiro de instrução.
- Registrado flags - determina o estado atual do microprocessador.

Registrador IP sempre trabalha junto com o registrador de segmento CS e aponta a instrução a executar actualmente.

Registrador flags é modificado automaticamente pelo CPU depois de operações matemáticas, isto permite determinar o tipo do resultado, e determinar condições para transferir o controle de outras partes do programa

TIPOS DE DADOS

- Para dizer ao compilador sobre os tipos de dados, estes prefixos devem ser usados
- byte ptr - para byte.
- word ptr - para palavra (dois bytes).
- por exemplo:
- byte ptr [BX]; acesso ao byte .
- ou
- word ptr [BX]; acesso a palavra.
- Montador suporta prefixos mais curtos como:
- b. - para byte ptr
- w. - para palavra ptr

Usando Instruções de transferência de dados

- MOV Destino,Fonte

- Cópia o segundo operando (fonte) para o primeiro operando (destino).

- Ambos operandos devem ser o mesmo tamanho que pode ser um byte ou uma palavra

Modos de endereçamento

- Directo : endereço é dado na instrução

```
MOV AX, Var1
:
```

Usando Instruções de transferência de dados

Modos de endereçamento

- Indirecto : endereço é lido de um registo base (BX ou BP) ou Index (SI ou DI) MOV AX, [BX]

- Indexado : identico ao anterior mas usando registrador index (SI ou DI) MOV AX,[SI+10]

- Imediato : MOV var, 12

Usando Instruções de transferência de dados

Transferencia permitidas

MOV REG, memória,

MOV memória , REG,

MOV REG, REG,

MOV memória , imediato,

MOV REG, imediato,

REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP,,,. 2

memória: [BX], [BX+SI+7], variável

imediato: 5, -24, 3Fh, 10001101b

VARIÁVEIS

Variável é um local na memória.

Nosso compilador suporta dois tipos de variáveis: BYTE e word.

Sintaxe para declaração de uma variável:

nome DB valor

nome DW valor

DB - serve para Definir Byte.

DW - serve para Definir word

VARIÁVEIS

- Variável é um local na memória.
- nome - pode ser qualquer letra ou uma combinação de dígito, entretanto deve começar com uma letra.
- valor - pode ser algum valor numérico em qualquer sistema de numeração (hexadecimal, binário, ou decimal), ou "? " símbolo para variáveis que não são inicializadas.

```
Vejamos  exemplo com instrução de MOV:
 ORG 100h
MOV AL, var1
MOV BX, var2
RET   ; stops the program.
VAR1 DB 7
var2 DW 1234h
```

Exemplo de instrução MOV

- JAVA

```
main(){
int x, y, z;
x = 7;
y = 13;
z = x;
```

- Assembly

```
ORG 100h
mov x,7
mov y,13
mov DX,x
mov z,DX
RET
x dw ?
y dw ?
z dw ?
```

Códigos ASCII		
Caracteres	**HEX**	**DEC**
espaço	20H	32
0 a 9	30H a 39H	48 a 57
A até Z	41H a 5AH	65 a 90
a até z	61H l 7AH	97 a 122
Enter	13H	18

Conjunto de instruções

Movimiento de datos.

MOV, PUSH, POP, XCHG, IN, OUT

Aritméticas.

ADD, SUB, INC, DEC, MUL, DIV, CMP

Lógicas. (Trabalham a nível de bit.)

AND, OR, XOR, TEST

Deslocamento e rotação. (Trabalham a nível de bit.)

SHL, SHR, ROL, ROR

Conjunto de instruções

Transferência de control.

Saltos incondicional.

JMP

Saltos condicionales.

JE, JA, JB, JNE, JNA, JNB, JC, JZ, JNC, JNZ

Control de laços.

LOOP

Conjunto de instruções

PUSH: transfere 2 ou 4 bytes ate a pilha

POP faz o inverso

O objetivo destas instrução é de guardar em um momento determinado o valor de registrador e logo retira-lo quando os necessite.

XCHG

Troca o conteúdo de um registrador com conteúdo de outro registrador ou uma localização de memoria; não se pode executar em registros de segmento.

Conjunto de instruções

XCHG

Conjunto de instruções

Arrays

Podem ser vistas como cadeias de variáveis. Um string de texto é um exemplo de um array de byte, cada caráter é apresentado como um código ASCII valor (0 ..255).

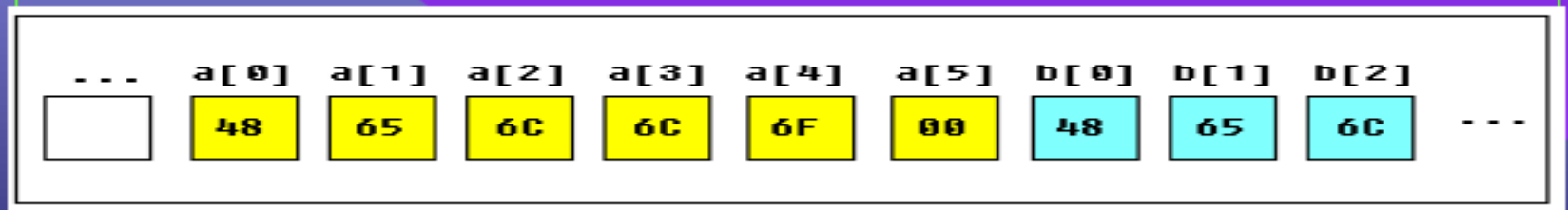
Aqui são alguns exemplos de definição de array:

```
um DB 48h, 65h, 6Ch, 6Ch, 6Fh, 00h,
b DB 'Oi', 0
```

Conjunto de instruções

Arrays

b é uma cópia exata do um array, quando o compilador vê um string automaticamente dentro de citações converte isto para conjunto de bytes. Este quadro mostra uma parte da memória onde estas são declaradas arrays:



Conjunto de instruções

Arrays

Você pode acessar o valor de qualquer elemento no array usando colchetes, por exemplo,:

```
MOV AL, a[3]
```

Você também pode usar qualquer registrador índice de memória BX, SI, DI, BP, por exemplo,:

```
MOV SI, 3,
MOV AL, a[SI]
```

Conjunto de instruções

Arrays

Se você precisa declarar uma array grande que você pode usar o operador DUP.

A sintaxe para DUP:

numero DUP (value(s))

número - número de duplicações a fazer (qualquer valor constante).

valor - expressão que DUP duplicará.

por exemplo:

C DB 5 DUP(9)

é um modo alternativo de declarar:

C DB 9, 9, 9, 9, 9,

CONSTANTES

- Constantes são como as variáveis, mas eles só existem até que seu programa é compilado (montador).Segundo a definição de uma constante seu valor não pode ser mudado. Para Definir constantes é usado diretiva de EQU :

```
nome EQU <qualquer expressão>
Por exemplo:
K EQU 5
MOV AX, k
O  exemplo acima é funcionalmente idêntico a código:
MOV AX, 5
```

INSTRUÇÕES ARITMÉTICA E LÓGICA

Há 3 grupos de instruções.

Primeiro grupo: ADD, SUB,CMP, E, TESTE, OU, XOR

ADD - soma segundo operando ao primeiro.

SUB - Subtraia segundo operando ao primeiro.

CMP - só Subtraia segundo operando do primeiro somente para flags(compara).

AND - Lógico AND entre todos os bits de dois operandos. Estas regras se aplicam

INSTRUÇÕES ARITMÉTICA E LÓGICA

Há 3 grupos de instruções.

Segundo grupo: MUL, IMUL, DIV, IDIV

MUL - Não assinado multiplique

IMUL - Assinou multiplique

DIV - Não assinado dividir:

IDIV - Assinou divida:

INSTRUÇÕES ARITMÉTICA E LÓGICA

Há 3 grupos de instruções.

Terceiro grupo: INC, DEC, NÃO, NEG

INC incrementa 1 : INC AX -> AX = AX + 1

DEC decrementa 1: DEC AX -> AX = AX – 1

NOT - Contrária cada bit do operando

NEG - Faça operando negar (dois complemento). De fato inverte cada bit de operando e então soma 1 a isto. Por exemplo 5 se tornarão -5, e -2 se tornarão 2

CONTROLE DE FLUXO DE PROGRAMA

saltos incondicionais .

A instrução básica que transfere controlo a outro ponto no programa é JMP

A sintaxe básica de instrução de JMP:

```
JMP rotulo
org 100h
 mov ax, 5                    ; fixe em ax a 5.
mov bx, 2                   ; bx fixo para 2.
jmp   calc                     ; vá 'calculo.'
atrás:  jmp para             ; vá 'parada.'
 calc:
```

CONTROLE DE FLUXO DE PROGRAMA

saltos incondicionais .

A instrução básica que transfere controlo a outro ponto no programa é JMP

```
add ax, bx                ; some bx para ax.
jmp  atrás                 ; vá 'atrás.'

parada:
ret                       ; volte a sistema operacional.
```

CONTROLE DE FLUXO DE PROGRAMA

Saltos Condicionais.

JE salta se for igual negação JNE

Instruction	Description	Condition	Opposite Instruction
JZ , JE	Jump if Zero (Equal).	ZF = 1	JNZ, JNE
JC , JB, JNAE	Jump if Carry (Below, Not Above Equal).	CF = 1	JNC, JNB, JAE
JS	Jump if Sign.	SF = 1	JNS
JO	Jump if Overflow.	OF = 1	JNO
JPE, JP	Jump if Parity Even.	PF = 1	JPO
JNZ , JNE	Jump if Not Zero (Not Equal).	ZF = 0	JZ, JE
JNC , JNB, JAE	Jump if Not Carry (Not Below, Above Equal).	CF = 0	JC, JB, JNAE
JNS	Jump if Not Sign.	SF = 0	JS
JNO	Jump if Not Overflow.	OF = 0	JO
JPO, JNP	Jump if Parity Odd (No Parity).	PF = 0	JPE, JP

CONTROLE DE FLUXO DE PROGRAMA

Saltos Condicionais.

JE salta se for igual negação JNE

JL salta se menor que < negação JNL

JG salta se maior que > negação JNG

JGE >= JNGE

JLE =< JNLE

CONTROLE DE FLUXO DE PROGRAMA

Loops.

Decrementa cx, e salta na etiqueta se cx não for 0

LOOP etiqueta:

```
Ex.             MOV CX,10
COMEÇO: ;Segmento a repetir
                LOOP COMEÇO
```

INTERRUPÇÕES DE SOFTWARE

- **Interrupções podem ser visto como várias funções. Estas funções tornam a programação muito mais fácil, em vez de escrever um código para imprimir um caráter você simplesmente pode chamar o interrupções e fará tudo para você**

- **Interrupções também é ativado através de diferente hardware, estes são chamados interrupções de hardware. Atualmente nós estamos interessados em software só interrupções.**

- **Para fazer uma interrupção de software há uma instrução INT, tem sintaxe muito simples:**
- **INT valor**

INTERRUPÇÕES DE SOFTWARE

- Interrupções de software podem ser ativadas diretamente por nossos programas assembly
- Dois tipos de interrupções

  - Interrupções do Sistema Operacional DOS
  - Interrupções da BIOS

- Para gerar interrupções do DOS use: INT 21h
- Quando usamos esta instrução, o DOS chama uma rotina de tratamento específica, dependendo do tipo de interrupção
- O tipo de interrupção será definido em função do valor que estiver armazenado no registrador AL

INTERRUPÇÕES DE SOFTWARE

- INT 21h (Entrada: AH <- 01h , Saída: caracter ->AL)
- Lê um caracter da console e coloca o seu código ASCII no registrador AL

Como gerar este tipo de Interrupção

- 1. Copie o valor 08h dentro do registrador AH
- 2. Chame a instrução "INT 21h"
- Após esta chamada, assim que for digitado um caracter, seu código ASCII
- será colocado dentro de AL

INTERRUPÇÕES DE SOFTWARE

- INT 21h (Entradas: AH <- 02h , DL<- caracter)
- imprime o caracter ou o executa.

Como gerar este tipo de Interrupção

- 1. Copie o valor 02h dentro do registrador AH
- 2. Copie o código ASCII do caracter que deseja imprimir dentro do registrador DL
- 3. Chame a instrução "INT 21h"L

INTERRUPÇÕES DE SOFTWARE

- Funções para mostrar informações no vídeo.

02H Exibe um caracter. O uso da função 40H é recomendado ao invés desta função

09H Exibe uma cadeia de caracteres

40H Escreve num dispositivo/arquivo

- Funções para ler informações do teclado.

01H Ler um caracter do teclado e mostrá-lo

0AH Entrada do teclado usando buffer

3FH Leitura de um dispositivo/arquivo

INTERRUPÇÕES DE SOFTWARE

- Interrupção 10h

Propósito: Chamar uma diversidade de funções do BIOS

Sintaxe:

Int 10H

Esta interrupção tem várias funções, todas para entrada e saída de vídeo. Para aceder cada uma delas é necessário colocar o número da função correspondente no registrador AH.

Veremos apenas as funções mais comuns da interrupção 10H.

Função 02H, seleciona a posição do cursor

Função 09H, exibe um caracter e o atributo na posição do cursor

Função 0AH, exibe um caracter na posição do cursor

Função 0EH, modo alfanumérico de exibição de caracteres

INTERRUPÇÕES DE SOFTWARE

Funções da interrupção 16h

- Função 00H, lê um caracter do teclado.
- Função 01H, lê o estado atual do teclado.

- Exemplos:

```
MOV AH,01h          ; Função 1 do DOS (leitura de caractere)
INT 21h       ; lê 1o caracter, retorna código ASCII ao registrador AL
MOV BL,AL ; move o código ASCII para o registrador BL por enquanto
INT 21h       ; lê 2 o caracter, retorna código ASCII ao registrador AL
```

INTERRUPÇÕES DE SOFTWARE

- ; --- Impressão dos 2 caracteres, na ordem invertida ---
- MOV AH,02h ; Função 2 do DOS (escrita de caractere)
- MOV DL,AL ; move o código ASCII do 2o caractere lido p/ DL
- INT 21h ; imprime o caractere cujo código está em DL
- MOV DL,BL ; move o código ASCII do 1o caracter lido p/ DL
- MOV AH,2h ; função 2h, imprime caracter
- INT 21h ; imprime o caractere cujo código está em DL

PROCEDIMENTOS

Procedimento é uma parte de código que pode ser chamado de seu programa para fazer alguma tarefa específica

A sintaxe para declaração de procedimento:

```
nome PROC

    ; aqui vai o código
    ; do procedimento...

RET
nome ENDP
```

PROCEDIMENTOS

nome - é o nome de procedimento, o mesmo nome deveria estar no topo e o fundo, isto é usado para conferir fechando correto de procedimentos

Exempolo:

```
ORG   100h
CALL  m1
MOV   AX, 2
RET               ; retorna ao sistema opacional.m1    PROC
MOV   BX, 5
RET               ; return to caller.
m1    ENDP

END
```

PROCEDIMENTOS

nome - é o nome de procedimento, o mesmo nome deveria estar no topo e o fundo, isto é usado para conferir fechando correto de procedimentos

Exempolo:

```
ORG   100h
CALL  m1
MOV   AX, 2
RET             ; retorna ao sistema opacional.m1    PROC
MOV   BX, 5
RET             ; return to caller.
m1    ENDP

END
```

EXERCÍCIOS

1- Escreva un fragmento de codigo em linguagem assembly que determine a quantidade de consoantes que se encontram no registo AX

2- Escreva un fragmento de codigo em linguagem assembly que determine a quantidade de letras mayusculas que se encontram numa lista de caracteres

a) Conhece-se os N elementos

b) Identifica-se o fim da corrente com @

3- Escreva un fragmento de codigo em linguagem assembly que dado uma lista de 10 numeros onde os mesmo não excedem de 255.

a) Determine o valor maior da lista

b) Determine a posicion do menor numero da lista

c) Quantos numeros são menores que 64

4- Escreva un fragmento de codigo em linguagem assembly que dado um texto determinado, calcule quantas palavras tem.


UNIVERSIDADE
DE LUANDA
Instituto de Tecnologias de
Informação e Comunicação
Linguagem de Montagem e
Linguagem de Máquina
Docente: Engº Nzuzi Rodolfo
Conclusão
2022 © nzuzi rodolfo