

Informática de Gestão

Arquitectura de Computadores

Linguagem

Assembly x86 & Emu8086

Docente: eng.º Nzuzi Rodolfo

QUEM É O PROF?



Engº Nzuzi Rodolfo Henriques Manuel
E-mail: nzuzimanuel@instic.uniluanda.ao
E-mail : nzuzirodolfo9@gmail.com
Código ORCID: 0009-0007-1463-3369



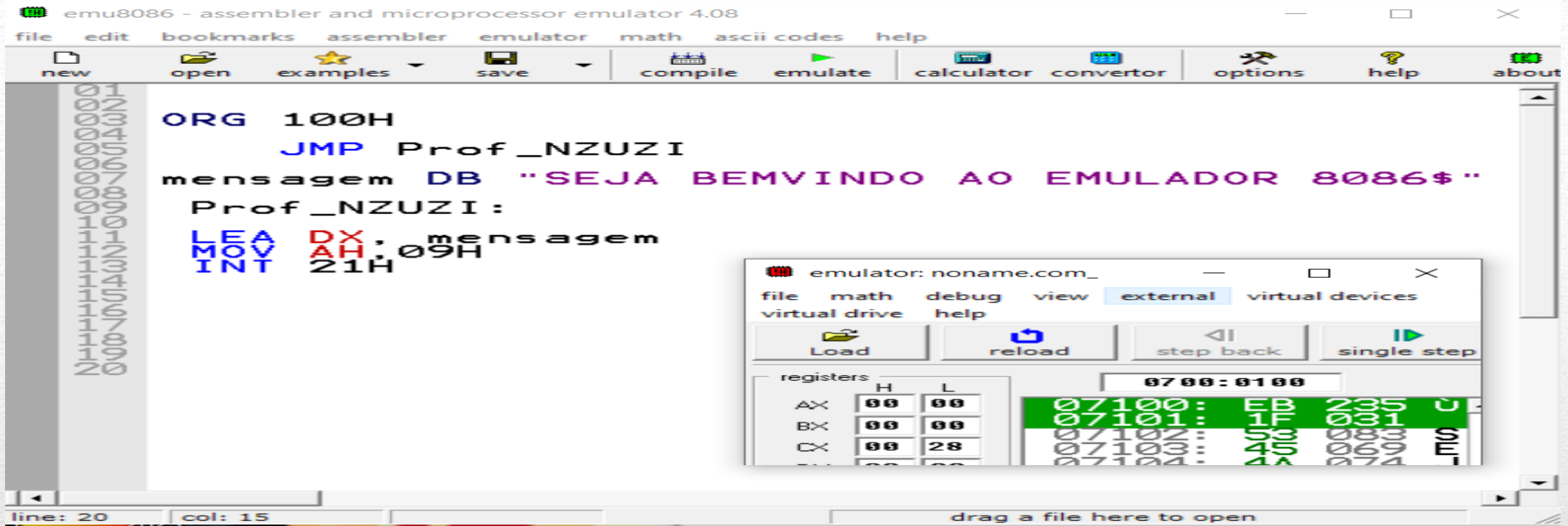
LinkedIn

APRESENTAÇÃO E INTRODUÇÃO DA DISCIPLINA

Esclarecimentos e Procedimentos

https://github.com/nzuziRodolfo/Arquitectura_Computadore_2024.git

2



EMU8086

Objetivos

- Conhecer o emulador 8086

Microprocessadores 80x86 16 bits



Microprocessadores 80x86 16 bits

ORG 100H

JMP Prof_NZUZI

mensagem DB "SEJA BEMVINDO AO EMULADOR 8086\$"

Prof_NZUZI:

LEA DX, mensagem

MOV AH, 09H

INT 21H

emulator: noname.com_

file math debug view external virtual devices
virtual drive help

Load reload step back single step

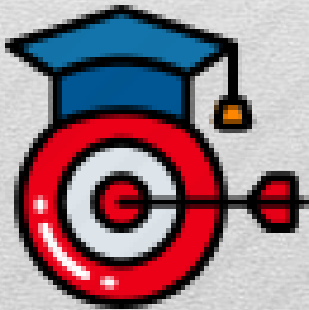
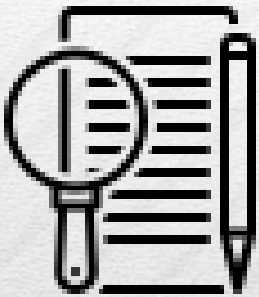
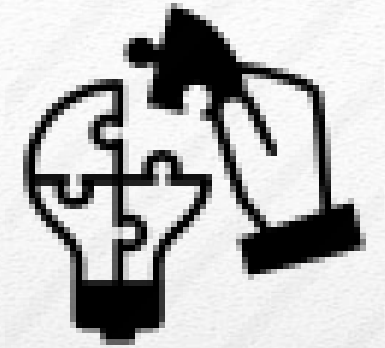
registers

	H	L
AX	00	00
BX	00	00
CX	00	28

0700:0100

07100:	EB	235	U
07101:	1F	031	
07102:	53	083	SU
07103:	45	069	
07104:	4A	074	

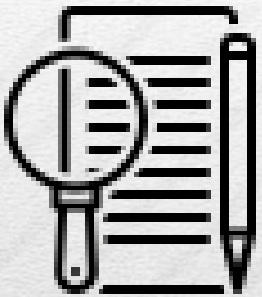
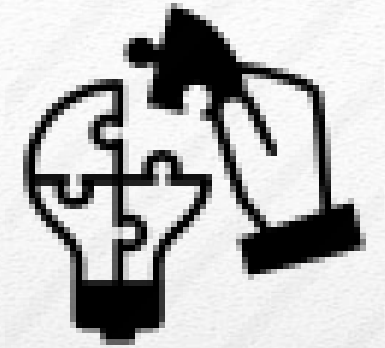
Conceitualização



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

Conceitualização



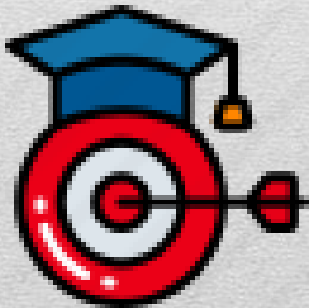
- **Tradutor ou compilador**

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.

Exemplo: nasm, tasm, and masm para x86



$N = I + J;$

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

Compilador

N

Linguagem/Código
de Máquina

Unidade de Processamento Central (CPU)

História

Linguagem/Código
de Máquina
(primeira geração)

1950s

Linguagem de montagem
(segunda geração)

1970s

Linguagens de
alto nível

(+)Complexidade (-)

(-)Produtividade (+)

Por que usar linguagem de montagem?

(+)Desempenho(-)

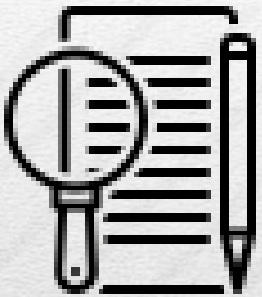
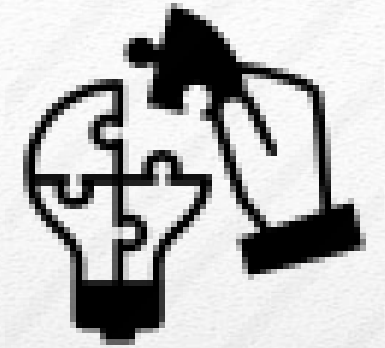
(+)acesso à máquina(-)

Código menor e mais rápido

- Cartão inteligente
- Drivers

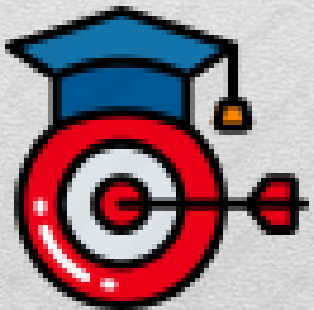
- Tratadores de interrupções de baixo nível em um S.O.
- Controladores em sistemas embutidos de tempo real

O que é a arquitetura x86 ?

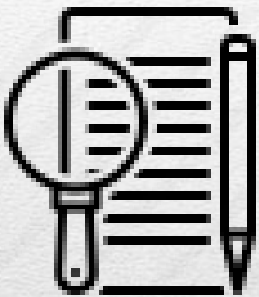
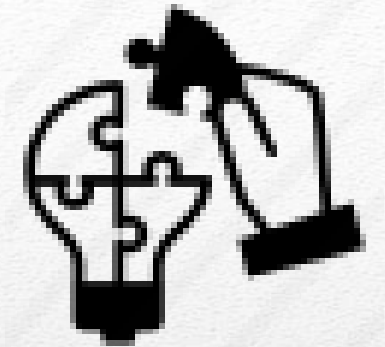


Essa arquitetura nasceu no **8086**, que foi um microprocessador da Intel que fez grande sucesso.

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 máquina.

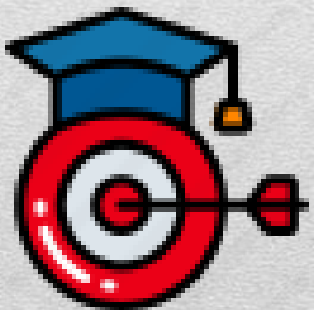


O que é a arquitetura x86 ?

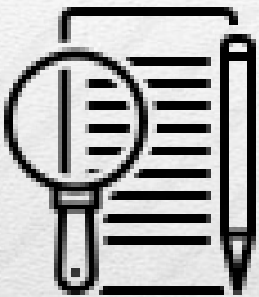


Essa arquitetura nasceu no **8086**, que foi um microprocessador da Intel que fez grande sucesso.

- ❑ O **avô** é o microprocessador **8086 de 16 bits** lançado em 1978;
- ❑ Os **pais** são os processadores de 32 bits: 80386, 80486, Pentium e Pentium IV;
- ❑ A **geração atual de processadores** são processadores de 64 bits: Intel Core i3, i5, i7

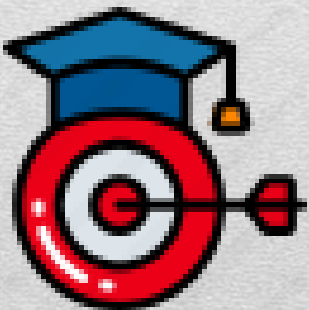


REQUISITOS PARA APRENDER ASSEMBLY



Conhecimento de alguma linguagem de programação de alto nível (**java**, **c#**, **c/c++**,...) isso pode o ajudar muito.

É assumido que você tem um pouco de conhecimento sobre representação de número (**hex/bin**).



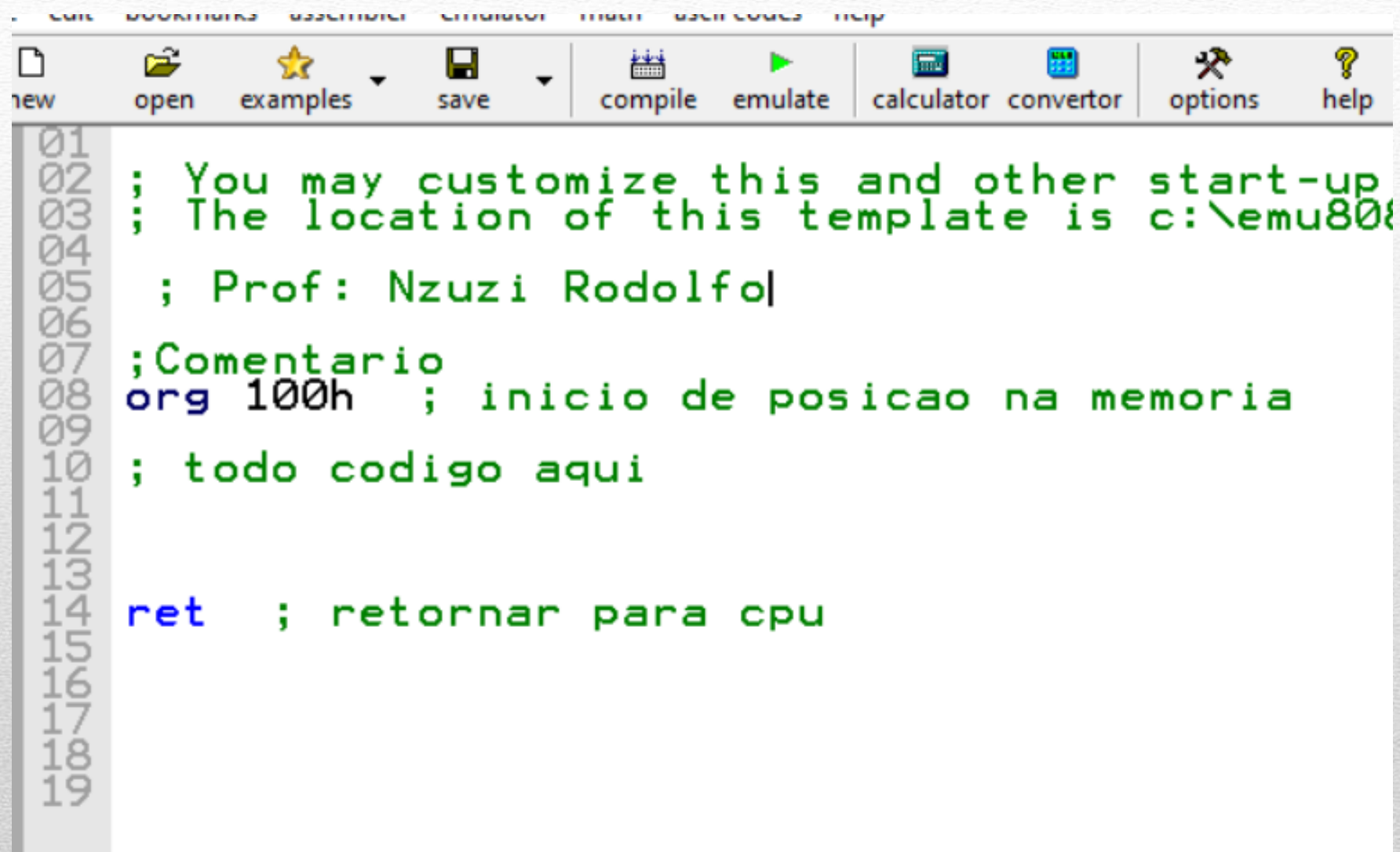
A SINTAXE ASSEMBLY DO 8086

Linguagem montadora não é sensível à letra maiúscula ou minúscula

Para facilitar a compreensão do texto do programa, sugere-se.

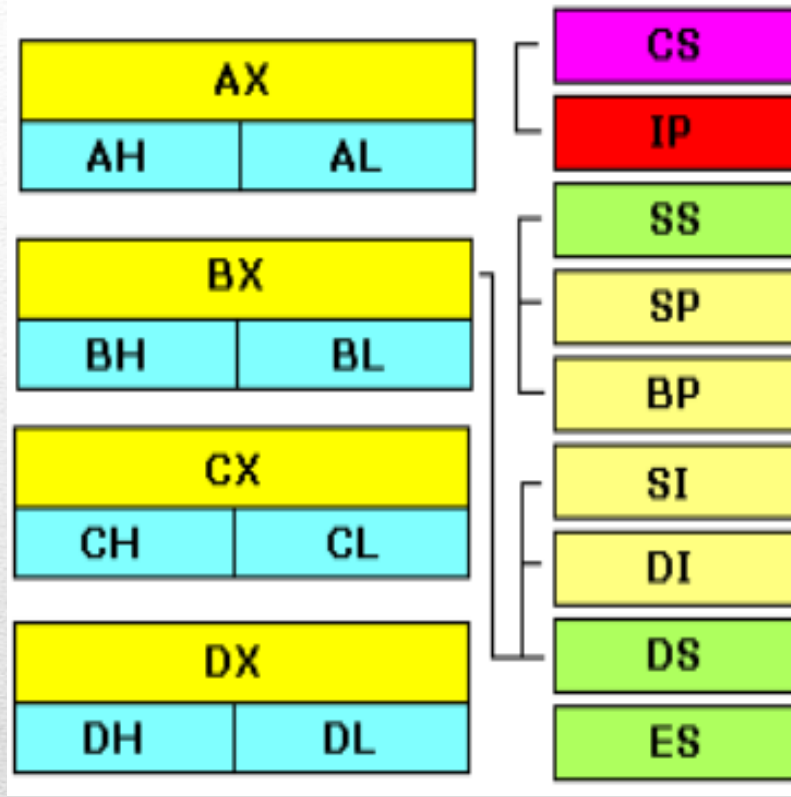
- Uso de letra maiúscula para código
- Uso de letra minúscula para comentários

Regras



The image shows a screenshot of the NEMU8086 emulator's main window. The window has a menu bar with options: File, Bookmarks, Assembly, Emulator, Math, User Codes, and Help. Below the menu bar is a toolbar with icons for new, open, examples, save, compile, emulate, calculator, convertor, options, and help. The main area displays assembly code with line numbers 01 through 19 on the left. The code is as follows:

```
01  
02 ; You may customize this and other start-up  
03 ; The location of this template is c:\emu8086  
04  
05 ; Prof: Nzuzi Rodolfo  
06  
07 ;Comentario  
08 org 100h ; inicio de posicao na memoria  
09  
10 ; todo codigo aqui  
11  
12  
13  
14 ret ; retornar para cpu  
15  
16  
17  
18  
19
```



DE PROPOSITO GERAL

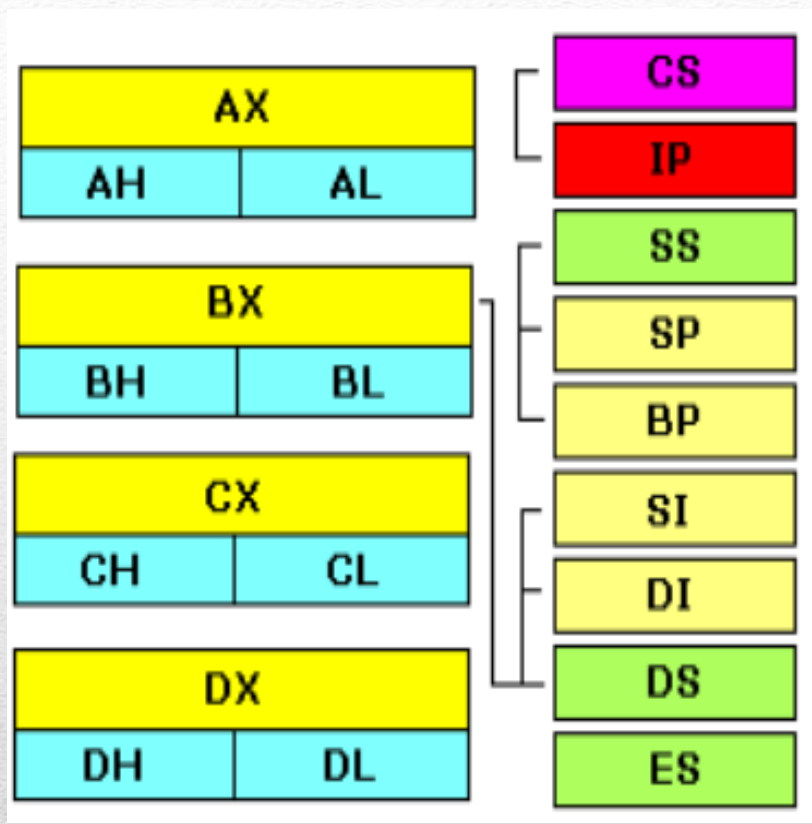
AX: Acumulador -> dividido em AH / AL Usado em operações aritméticas.

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

Registadores



DE PROPOSITO GERAL

Ligadas a bx índice de vetores.

SI - registrador de índice de fonte.

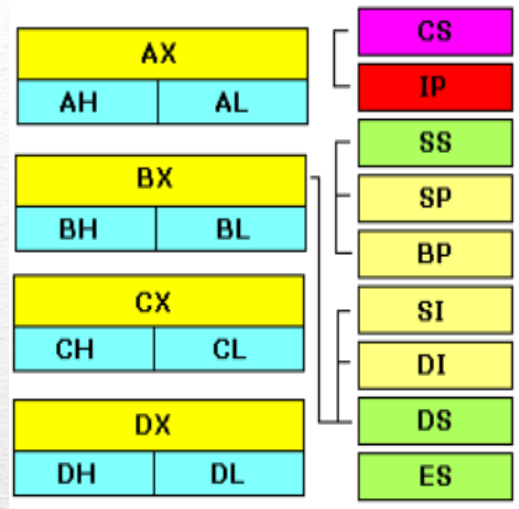
DI - registrador de índice de destino.

BP - ponteiro base.

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).

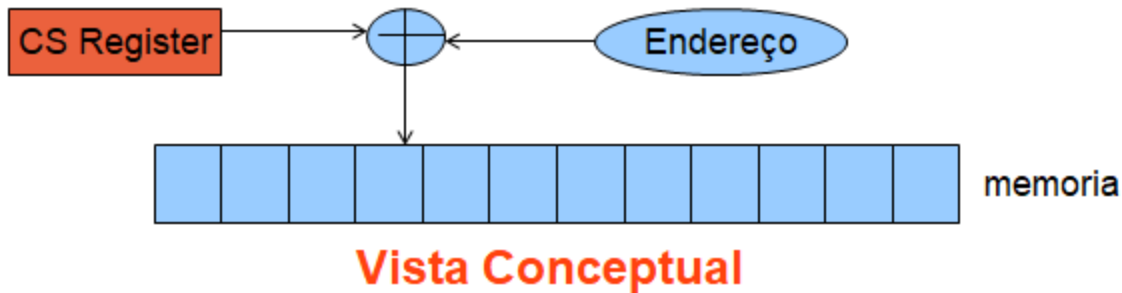
Registadores



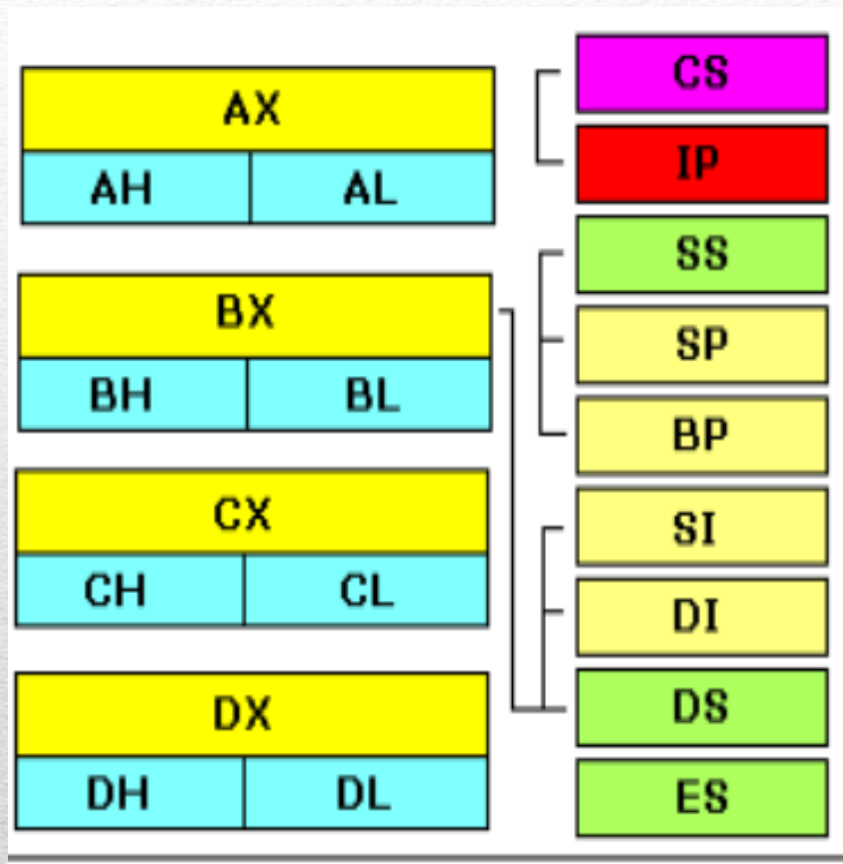
REGISTRADORES DE SEGMENTO

- ❖ **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.).



Registadores



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

Registadores

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

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

❖ **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**

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

Vejamos exemplo com instrução de MOV:

ORG 100h

MOV AL, var1

MOV BX, var2

RET ; stops the program.

VAR1 DB 7

var2 DW 1234h

• **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 a 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.

Salto incondicional.

JMP

Salto condicionais.

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 instruções 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.

**boq6 ex6cnpai em ledigp02 q6 26dhwento
onplo ledigp02oi on nua9 loca9izacão q6 memoria; n9o 26
p02oi o con9en9o q6 nua9 ledigp02oi con9 con9en9o q6**



Busque auxílio em livro,
não pare por aqui!

Docente: eng.º Nzuzi Rodolfo