**UNIVERSIDADE FEDERAL DE PELOTAS – UFPEL**
**CENTRO DE DESENVOLVIMENTO TECNOLÓGICO (CDTec)**
**CURSOS DE CIÊNCIA DA COMPUTAÇÃO E ENGENHARIA DE COMPUTAÇÃO**
**DISCIPLINA DE PROGRAMAÇÃO DE SISTEMAS**

PROFs.: Me. ANDERSON PRIEBE FERRUGEM

SEGUNDO TRABALHO
TÓPICO: MÁQUINA VIRTUAL.

O TRABALHO SERÁ UMA APRESENTAÇÃO EM VÍDEO DO GRUPO COM TODOS PARTICIPANTES COM CÓDIGO DISPONIBILIZADO VIA GITHUB;
O ENVIO É FEITO APENAS POR COMPONENTE DO GRUPO;
A DURAÇÃO MÁXIMA DO VÍDEO DEVERÁ SER DE **10 MIN** COM **TOLERÂNCIA** DE **5 MIN. (5-15)** ;
**A APRESENTAÇÃO DEVERÁ MOSTRAR:**
**1) INTERAÇÃO ENTRE OS COMPONENTES;**
**2)ARGUIÇÃO DO FUNCIONAMENTO E DAS TÉCNICAS USADAS.**

**A APRESENTAÇÃO NÃO DEVERÁ SER APENAS:**
**1) APRESENTAÇÃO DE SLIDES;**
**2) APRESENTAÇÕES INDIVIDUAIS DOS COMPONENTES DO GRUPO.**

EM CASO DE DÚVIDAS SOBRE A APRESENTAÇÃO PROCUREM POSTAR NO E-AULAS (DESTA FORMA A RESPOSTA FICA DISPONÍVEL A TODOS).

**FERRAMENTAS:**

**SOFTWARE:**
**JAVA OU C++ (Escolha do grupo)**
**Apresentação gráfica da execução !!!**

---

## Projeto máquina virtual do sistema computacional hipotético Z808

### Introdução

O trabalho descrito a seguir consiste em implementar a máquina virtual (emulador) do sistema computacional Z808 - conforme apresentado no livro Tradução de programas – Da montagem a carga. Cristian Koliver.

Tal sistema será composto de dois módulos que deverão operar de forma integrada: o executor (emulador propriamente dito) e uma interface visual. O resultado do trabalho deverá ser entregue com toda a documentação (programas fontes, programa executável, documentação formal sucinta das estruturas de dados definidas, das funções desenvolvidas e estratégias adotadas) pelo Github. A avaliação do trabalho será realizada com base nos seguintes aspectos:

- **correção do programa,**
- **adequação das definições adotadas,**
- **uso das técnicas básicas de programação,**
- **autenticidade e domínio sobre o produto gerado,**

# Descrição do emulador Z808

## 1. Memória

A memória do computador é definida pelos seguintes atributos:

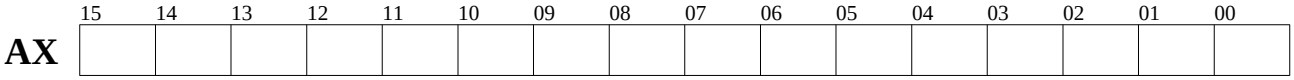
| | |
|---|---|
| **Tamanho da memória** | 64 KB (65536) |
| **Palavra de memória** | 16 bits |
| **Unidade de endereçamento** | palavra |
| **Bit de paridade** | <NA> |
| **Cache** | <NA> |
| Observações adicionais: <NA> significa "Não se aplica". | |

## 2. Registradores de dados

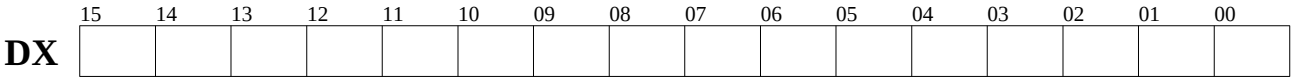
Um registrador é uma pequena porção de memória localizada no processador central. Os registradores permitem acessos muito rápidos a dados e são usados para aumentar a velocidade de execução de programas. A maioria das modernas arquiteturas de computadores opera transferindo dados da memória principal para os registradores, onde estes são processados e o resultado é devolvido à memória principal - é a chamada arquitetura de carregamento-armazenamento.

O Z808 possui **dois registradores de dados AX** (Acumulador) e **DX** (Registrador de dados) de **16 bits**. Os bits recebem um designação numérica de 0 a 15, da direita para esquerda, sendo o bit 0 o de mais baixa ordem ou menos significativo.

**AX** : é o chamado ACUMULADOR (o X refere-se a “eXtended”). O AX pode ser dividido em dois registradores de 8 bits: AL e AH, sendo AL formado pelo byte menos significativo (D0 a D7), enquanto AH é constituído pelo byte mais significativo (D8 a D15), podendo cada um deles ser acessado separadamente. **O código de endereçamento de AX é C0.**



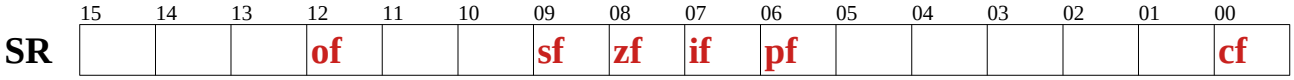

**DX** também pode ser dividido em DL e DH. pode ser dividido em DL e DH. É chamado de registrador de DADOS, pois ele pode ser usado como uma extensão do AX em operações de multiplicação e divisão.**O código de endereçamento de DX é C2.**




### 3. Demais Registradores:

A lista seguinte mostra os demais registradores implementados no computador hipotético e sua descrição.

| Registrador | Tipo | Tamanho (bits) | Descrição |
|---|---|---|---|
| **SP** | Pilha<br>STACK POINTER | **16** | Aponta para o topo da memória do tipo pilha. Usado pelas instruções push e pop. |
| **SI** | Registrador de índice<br>SOURCE INDEX | **16** | Aponta para a origem dos dados que serão movimentados. É usado para indexação de tabelas no endereçamento indireto. |
| **IP** | Apontador de instrução<br>*Instruction pointer* | **16** | Contém durante a execução de um programa o endereço na memória da próxima instrução a ser executada |
| **SR** | Registrador de estado<br>*Status register* | **16** | Contém seis flags de um bit. Usados para indicar várias condições durante a execução do programa |




Os "flags" de estado podem ser testados após algumas operações terem sido realizadas e cujos resultados poderia ser usados para decisões de desvios condicionais. Os "flags" são inicializados da seguinte maneira:
**CE:** é setado (recebe valor 1) se uma operação de adição resulta cm **"carry"** (vai-um) ou se uma operação de subtração resulta cm "borrow" (vem-um). Se nenhum "carry" ou "borrow ocorrer após urna adição ou subtração, ele é resetado (recebe 0). A execução da sequência de instruções

```
MOV AX,1111111111111111b
add AX,1
```

onde a letra b ao lado da constante indica que está sendo usada a notaçao binária, faria CF ser setado;

**PF**: é o "flag" de paridade. É setado quando um valor contêm um número par de "bits" no estado 1 (caso o número contenha um número impar de 1's. ele é resetado). A execução da sequência de instruções
mov AX, 0

add AX, 0111011101110111b

seta PF, pois o operando-destino (AX) passa a conter um número par de 1's;

**IF**: 'flag" de interrupção. Quando ele está setado, o Z808 ignora (desabilita) as interrupções. Ele é alterado pelo usuário, quando este deseja desabilitar interrupções (**popf pushr**)
**ZF**: é o flag zero.Setado quando resultado de uma operação é zero (1). Para resultados diferentes do zero ele é resetado(0).
**SF**: flag de sinal, usado para indicar se o número é positivo ou negativo.
**OF**: Overflow Flag, indica um estouro da capacidade de armazenamento de um registrador.

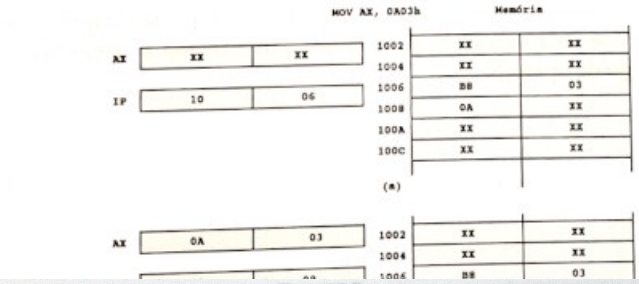
## 4. Modos de Endereçamento

### 2.3. MODOS DE ENDEREÇAMENTO

Os modos de endereçamento (modos como o processador obtém os dados) empregados pelo Z808 são cinco: imediato, via registrador, direto, indireto e indexado.

#### 2.3.1. ENDEREÇAMENTO IMEDIATO

No endereçamento imediato, o operando é uma constante numérica que segue o código da instrução. Existem três modos de referenciar-se a constantes no Z808: diretamente, através de um número, através de um nome simbólico de uma constante declarada com a diretiva **EQU** (vide adiante) ou através de um nome simbólico precedido da diretiva **OFFSET** (o conceito de diretiva será explicado posteriormente). A figura 2.4 ilustra um exemplo do uso do modo de endereçamento imediato (B8 é o código de operação da instrução).




#### 2.3.2. ENDEREÇAMENTO DIRETO

No modo de endereçamento direto ou absoluto, o operando localiza-se em um endereço especificado na instrução (imediatamente após o código de operação). A figura 2.5 ilustra um exemplo do modo de endereçamento direto. A instrução do exemplo (**mov AX,Oito**, onde 'Oito' é um nome simbólico - variável - do endereço 1002) faz um acesso a mais à memória do que a anterior (um acesso para pegar o código de operação, um acesso para pegar o endereço do dado e um outro para pegar o conteúdo daquele endereço). **Oito** deve ter sido declarado como uma variável (inicializada ou não) da seguinte forma:

```
Oito DW 8;
```

(se **Oito** tivesse sido declarado como uma constante através da diretiva **EQU**,

```
Oito EQU 8;
```

o modo de endereçamento seria o imediato).

#### 2.3.3. ENDEREÇAMENTO VIA REGISTRADOR

No modo de endereçamento via registrador, o(s) operando(s) é (são) o(s) conteúdo(s) de um registrador(es), conforme o exemplo da figura 2.6.

O modo de endereçamento via registrador é o mais rápido (exige menos ciclos de UCP) uma vez que faz um único acesso à memória, para pegar o código de operação.

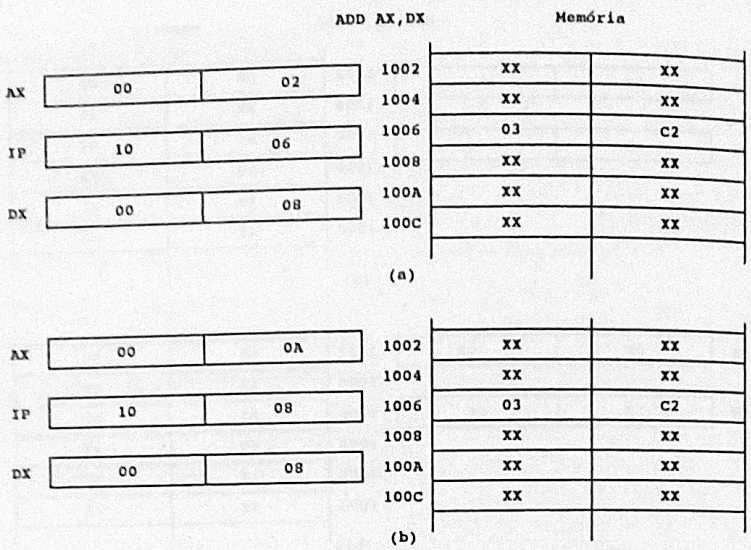



**Figura 2.6 Exemplo do modo de endereçamento via registrador: (a) antes da execução da instrução; (b) após a execução da instrução**

### 2.3.5. ENDEREÇAMENTO INDEXADO

No modo de endereçamento indexado, o registrador **SI** é usado como um deslocamento (índice) em relação à uma base. Essa base é um endereço. É importante ressaltar que o deslocamento é dado em "bytes".

O modo de endereçamento indexado é bastante conveniente para o processamento de vetores e matrizes.

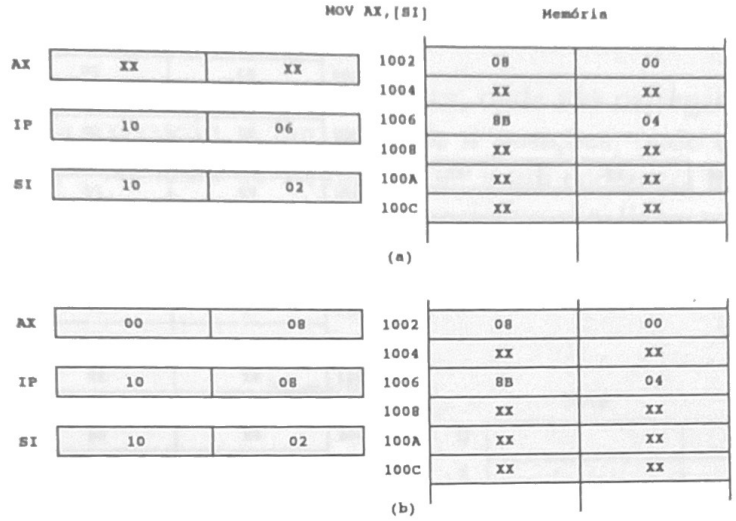



**Figura 2.7 Exemplo do modo de endereçamento indireto: (a) antes da execução da instrução; (b) após a execução da instrução**

A figura 2.8 ilustra um exemplo de tal modo de endereçamento. **Tabela** é o nome simbólico do endereço 0A02 que representa a base de um vetor de dez posições (cada posição com o tamanho de uma palavra) inicializadas com valores primos, conforme a declaração abaixo:

```
Tabela      DW 01
            DW 03
            DW 05
            DW 07
            DW 11
            DW 13
            DW 17
            DW 19
            DW 23
            DW 29
```

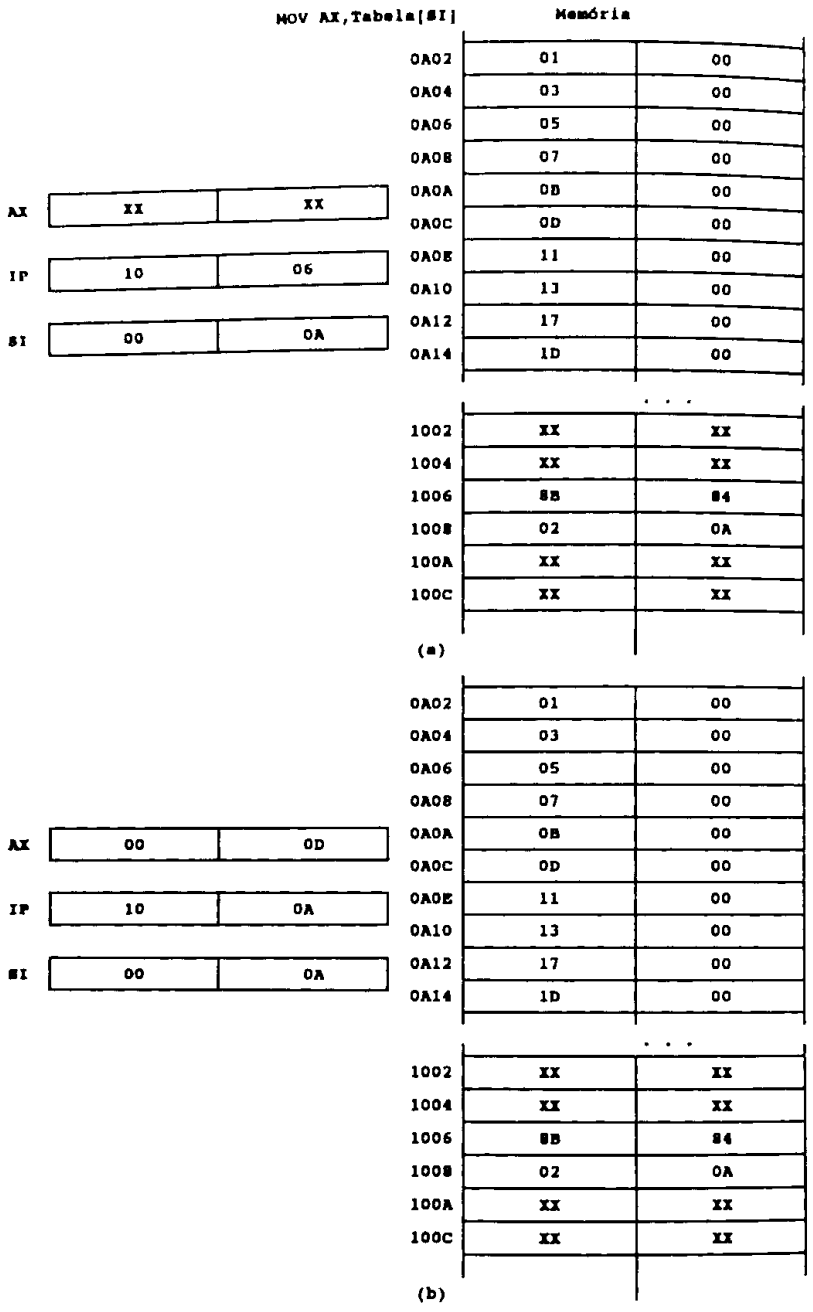



**Figura 2.8 Exemplo do modo de endereçamento indexado**

Deseja-se mover para **AX** o conteúdo da sexta posição do vetor ou posição de índice 5 (que contém o valor 13 ou 000Dh). Como tal posição representa um deslocamento de 10 "bytes" em relação à base, SI foi carregado com 0A (10 em hexadecimal).

## 5. Conjunto de Instruções

A seguir está definido o conjunto de instruções reconhecido pelo computador, acompanhado de todas as informações necessárias para sua implementação.

Cada código de instrução (*opcode*) e operando (opd1 ou opd2) ocupa uma palavra de memória. As ações dizem respeito aos registradores, conforme identificação definida na tabela de registradores e endereços de memória referenciados. As observações sinalizadas se são descritas na legenda abaixo do quadro.

| **Mnemônico** | **Cód. de Máq.* (opcode)** | **Tam. da Instrução (bytes - 8bits)** | **N° de Operandos** | **Ação (comentário) **** | **Modos de endereçamento (D/In/Im)** | **Flags afetados** | |
|---|---|---|---|---|---|---|---|
| **add AX,reg** | **03 C0**<br>**03 C2** | **2** | **1** | AX← AX + AX<br>AX← AX + DX | **&** | **CF,PF,ZF,SF,OF** | |
| **add AX,opd** | **05 opd** | **3** | **1** | AX←AX + **opd** | **D/In/Im** | **CF,PF,ZF,SF,OF** | |
| **div SI** | **F7 F6** | **2** | **1** | AX←AX **div** SI<br>DX←AX **mod** SI | **&** | **CF,PF,ZF,SF,OF** | |
| **div AX** | **F7 C0** | **2** | **1** | AX←AX **div** AX<br>DX←AX **mod** AX | **&** | **CF,PF,ZF,SF,OF** | |
| **sub AX,reg** | **2B C0**<br>**2B C2** | **2** | **1** | AX← AX – AX<br>AX← AX - DX | **&** | **CF,PF,ZF,SF,OF** | |
| **sub AX,opd** | **25 opd** | **3** | **1** | AX←AX -**opd** | **D/In/Im** | **CF,PF,ZF,SF,OF** | |
| **mul SI** | **F7 F6** | **2** | **1** | AX←AX * SI<br>DX←AX * SI (parte alta da multiplicação se houver overflow; caso contrario DX=0) | **&** | **CF,PF,ZF,SF,OF** | |
| **mul AX** | **F7 F0** | **2** | **1** | AX←AX * AX<br>DX←AX * AX<br>(parte alta da multiplicação se houver overflow caso contrario DX=0)) | **&** | **CF,PF,ZF,SF,OF** | |
| **cmp AX,opd** | **3D opd** | **3** | **1** | ZF←1, se AX = **opd**<br>ZF←0, se AX ≠ **opd** | **D/In/Im** | **CF,PF,ZF,SF,OF** | |
| **cmp AX,DX** | **3B C2** | **2** | **1** | ZF←1, se AX = DX<br>ZF←0, se AX ≠ DX | **&** | **CF,PF,ZF,SF,OF** | |
| **and AX,AX** | **23 C0**<br>**23 C2** | **2** | **1** | AX←AX and AX<br>AX←AX and DX | **&** | **CF,PF,ZF,SF,OF** | # |
| **and AX,opd** | **25 cte** | **3** | **1** | AX←AX and **opd** | **D/In/Im** | **CF,PF,ZF,SF,OF** | # |
| **not AX** | **F8 C0** | **2** | **1** | AX←not(AX) | **&** | **CF,PF,ZF,SF,OF** | # |
| **or AX,AX** | **0B C0**<br>**0B C2** | **2** | **1** | AX←AX or AX<br>AX←AX or DX | **&** | **CF,PF,ZF,SF,OF** | # |
| **or AX,opd** | **0D cte** | **3** | **1** | AX←AX or **opd** | **D/In/Im** | **CF,PF,ZF,SF,OF** | # |
| **xor AX,AX** | **33 C0**<br>**33 C2** | **2** | **1** | AX←AX xor AX<br>AX←AX xor DX | **&** | **CF,PF,ZF,SF,OF** | # |
| **xor AX,cte** | **35 cte** | **3** | **1** | AX←AX xor cte | **D/In/Im** | **CF,PF,ZF,SF,OF** | # |
| **jmp opd** | **EB opd** | **3** | **1** | IP ←**opd** | **D/In/Im** | **inalterados** | |
| **jz opd** | **74 opd** | **3** | **1** | IP ←**opd**, se ZF = 1 | **D/In/Im** | **inalterados** | |
| **jnz opd** | **75 opd** | **3** | **1** | IP ←**opd**, se ZF ≠1 | **D/In/Im** | **inalterados** | |

| Mnemônico | Cód. de Máq.* (opcode) | Tam. da Instrução (bytes - 8bits) | N° de Operandos | Ação (comentário) ** | Modos de endereçamento (D/In/Im) | Flags afetados | |
|---|---|---|---|---|---|---|---|
| **jp opd** | **7A opd** | **3** | **1** | IP ←**opd**, se SF = 0 | **D/In/Im** | **inalterados** | |
| **call opd** | **E8 opd** | **3** | **1** | [SP]←IP<br>(desvio para sub-rotina opd1) | **D/In/Im** | **inalterados** | |
| **ret** | **EF** | **1** | **0** | IP←[SP]<br>(retorno de sub-rotina) | **&** | **inalterados** | |
| **hlt** | **EE** | **1** | **0** | término (fim) de execução | **&** | **inalterados** | |
| **pop reg** | **58 C0**<br>**58 C2** | **2** | **1** | AX←[SP]<br>DX←[SP]<br>Pega valor de 16 bits do topo da pilha e armazena no registrador | **&** | **inalterados** | |
| **pop opd** | **59** | **3** | **1** | opd←[SP]<br>Pega valor de 16 bits do topo da pilha e armazena no endereço | **D/In/Im** | **inalterados** | |
| **popf** | **9D** | **1** | **0** | SR←[SP]<br>Pega valor de registrador SR de 16 bits do topo da pilha e atualiza flags | **&** | **CF,PF,ZF,SF,OF** | |
| **push reg** | **50 C0**<br>**50 C2** | **2** | **1** | [SP]←AX<br>[SP]←DX<br>Pega valor de 16 bits do registrador armazena no topo da pilha | **&** | **inalterados** | |
| **pushf** | **9C** | **1** | **0** | [SP]←SR<br>Pega valor de registrador SR de 16 bits do topo da pilha e coloca na pilha | **&** | **inalterados** | |
| **store reg** | **07 C0**<br>**07 C2** | **2** | **1** | opd1← AX<br>opd1← DX | **D/In/Im** | **inalterados** | |
| **read opd** | **12 opd** | **3** | **1** | opd1←*input stream* | **D/In/Im** | **inalterados** | |
| **write opd** | **08 opd** | **3** | **1** | *Output stream*←opd1 | **D/In/Im** | **inalterados** | |

**Legenda**

** A referência "[SP]" representa um elemento a ser retirado da pilha (operação ***pop***) e a referência "[SP]←" representa uma operação ***push*** na pilha.

(#) Executa a intrução bit-a-bit.

(%) As instruções CALL e RET utilizam a "Pilha do Sistema" para tratamento dos endereços de retorno, conforme descrito no item específico sobre este tema.

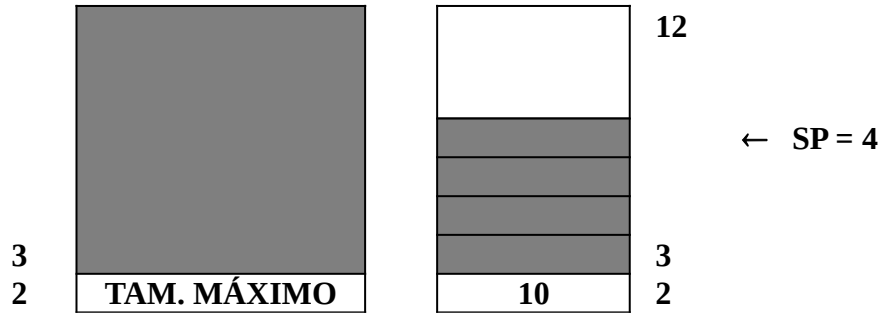
(&) Não se aplica.

## 6. Pilha do Sistema

Uma pilha é utilizada pelo sistema para armazenar os endereços de retornos de sub-rotinas, conforme indicado na seção sobre o "Conjunto de Instruções". Esta pilha do sistema é endereçada (acessada) através do registrador **SP** (ponteiro da pilha).

A pilha do sistema está localizada no início da memória física, a partir do **endereço 2** (**endereço base da pilha**), cujo conteúdo não pode se desempilhado e deve manter o seu tamanho máximo (*Stack Limit*). O valor inicial do SP é implicitamente carregado com zero ao "ligar a máquina virtual". O ponteiro da pilha somente pode crescer incrementando até seu limite, causando um desvio para o endereço 0 (zero), caracterizada como uma exceção de *"Stack Overflow"*, caso haja uma tentativa de empilhar com a pilha cheia.

A estrutura da pilha é a seguinte:




## Bibliografia

CALINGAERT, Peter. **Assemblers, Compilers, and Program Translation.** Potomac: Computer Science Press, Inc, 1979.
STALLINGS, Willian. **Computer Organization and Architecture**. 5.ed. New Jersey: Prentice Hall, 1999.
TANENBAUM, Andrew. **Structured Computer Organization.** 4.ed. New Jersey: Prentice Hall, 1999.
KOLIVER, Cristian. **Tradução de programas – Da montagem a carga.** 1.ed. Caxias do Sul: EDUCS, 1996.