Nível da Arquitetura do Conjunto das Instruções

(Aula 12)

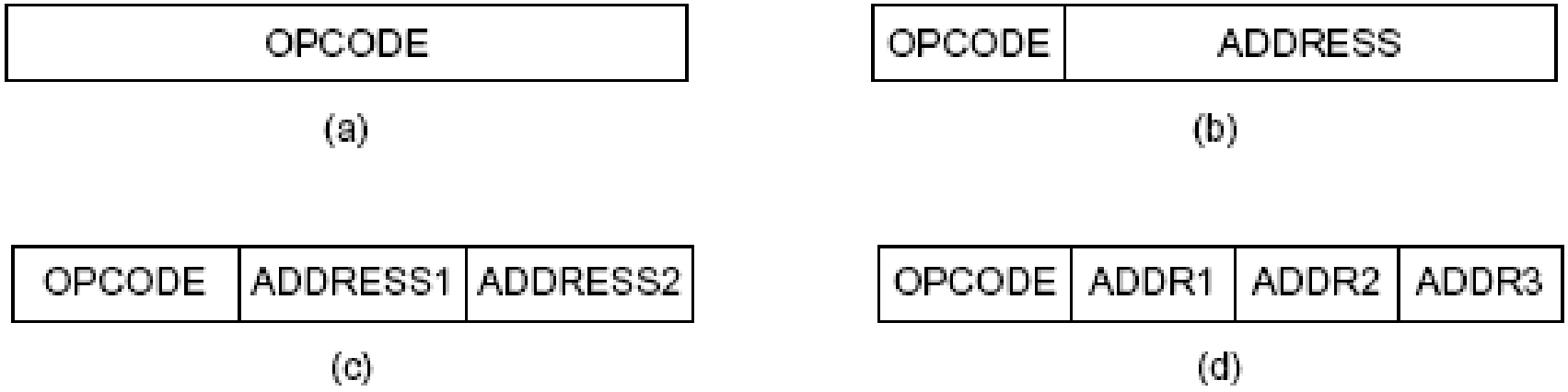
Formatos de Instruções

Modos de Endereçamento

Tipos de Instruções

Formatos de Instruções (1)

- Uma instrução é formada por:
  - um código de operação (obrigatório)
  - informações a respeito da fonte e do destino de seus operandos (facultativo)
    - Ex: um, dois ou três endereços.

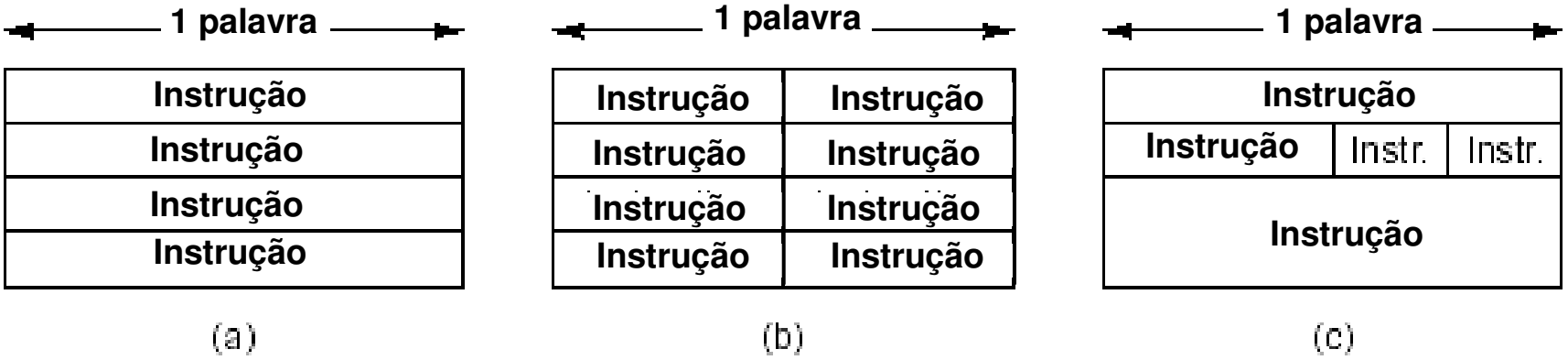


Quatro formatos de instruções muito comuns: (a) Instrução sem endereço; (b) Instrução com um endereço; (c) Instrução com dois endereços; (d) instrução com três endereços.

Formatos de Instruções (2)

- O código de operação (opcode) informa ao hardware o que deve ser feito quando de sua execução
- Em algumas máquinas, todas as instruções têm o mesmo tamanho. Em outras, esse tamanho pode variar de uma instrução para a outra
- Instruções com tamanhos diversos
  - Complica o projeto mas tem-se economia de memória
- Instruções de tamanhos iguais
  - Simplifica o projeto, mas desperdiça espaço

Formatos de Instruções (3)



- Algumas das relações possíveis entre o tamanho das instruções e o tamanho das palavras de memória

Critérios para Determinação do Formato de Instruções (1)

- Para uma arquitetura ter sucesso comercial
  - Pode-se querer que ela dure 20 anos ou mais, mantendo compatibilidades
  - O ISA deve ter a capacidade de suportar o acréscimo de novas instruções
    - Sendo capaz de explorar novidades ao longo do tempo de vida
  - Algumas decisões tomadas durante o projeto ISA podem revelar-se inadequadas ao longo do tempo, principalmente se a tecnologia for incompatível com a implementação do ISA

Critérios para Determinação do Formato de Instruções (2)

- Em geral as instruções pequenas são mais atraentes do que as grandes. Por que?
  - Um programa feito com instruções de 16 bits gasta metade do espaço de memória de um programa com instruções de 32 bits (supondo o mesmo número de instruções)
  - A execução de instruções menores tende a ser mais rápida (manipulação de um menor número de registradores)
  - Diminui-se o gargalo na banda passante da memória
    - A memória não tem capacidade de suprir instruções e dados na mesma velocidade à qual o processador é capaz de "consumi-los"
    - Se banda passante de uma cache = bd bits/s
      tamanho médio das instruções = tm bits
      a memória só disponibiliza no máximo **bd/tm** instruções por seg

Nº de bits/segundo

Critérios para Determinação do Formato de Instruções (3)

- Por outro lado, a minimização do tamanho das instruções pode dificultar muito a sua decodificação
  - O projetista é obrigado a usar um conjunto restrito de códigos para as instruções e o projeto pode não ser flexível para aumento da quantidade de instruções
- Deve haver espaço suficiente para que todas as operações desejadas possam ser expressas
  - O número de bits reservado para o opcode
  - A forma de endereçamento (operandos da instrução)
    - Ex: se tam. de endereço = 16 bits

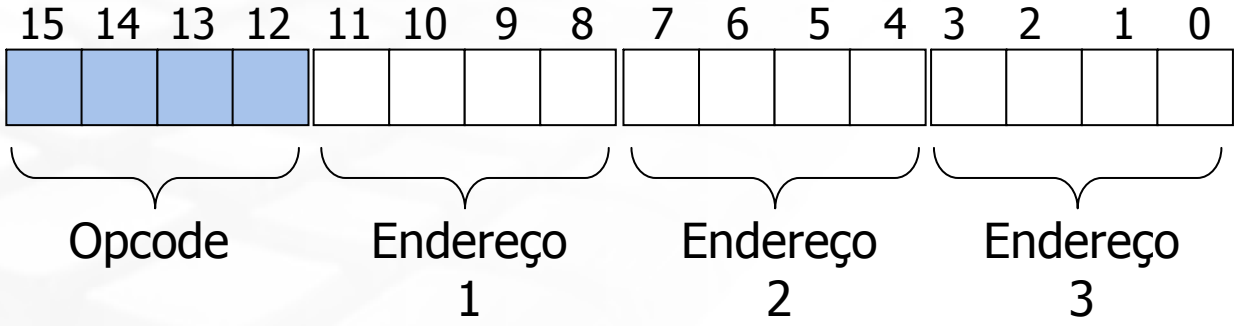
      - Um operando que seja um endereço de um byte (uma célula) em uma instrução deve ter 16 bits

      - Um operando que seja um endereço de uma palavra de 4 (2^2) bytes só precisa ter 16 - 2 bits

Critérios para Determinação do Formato de Instruções (4)

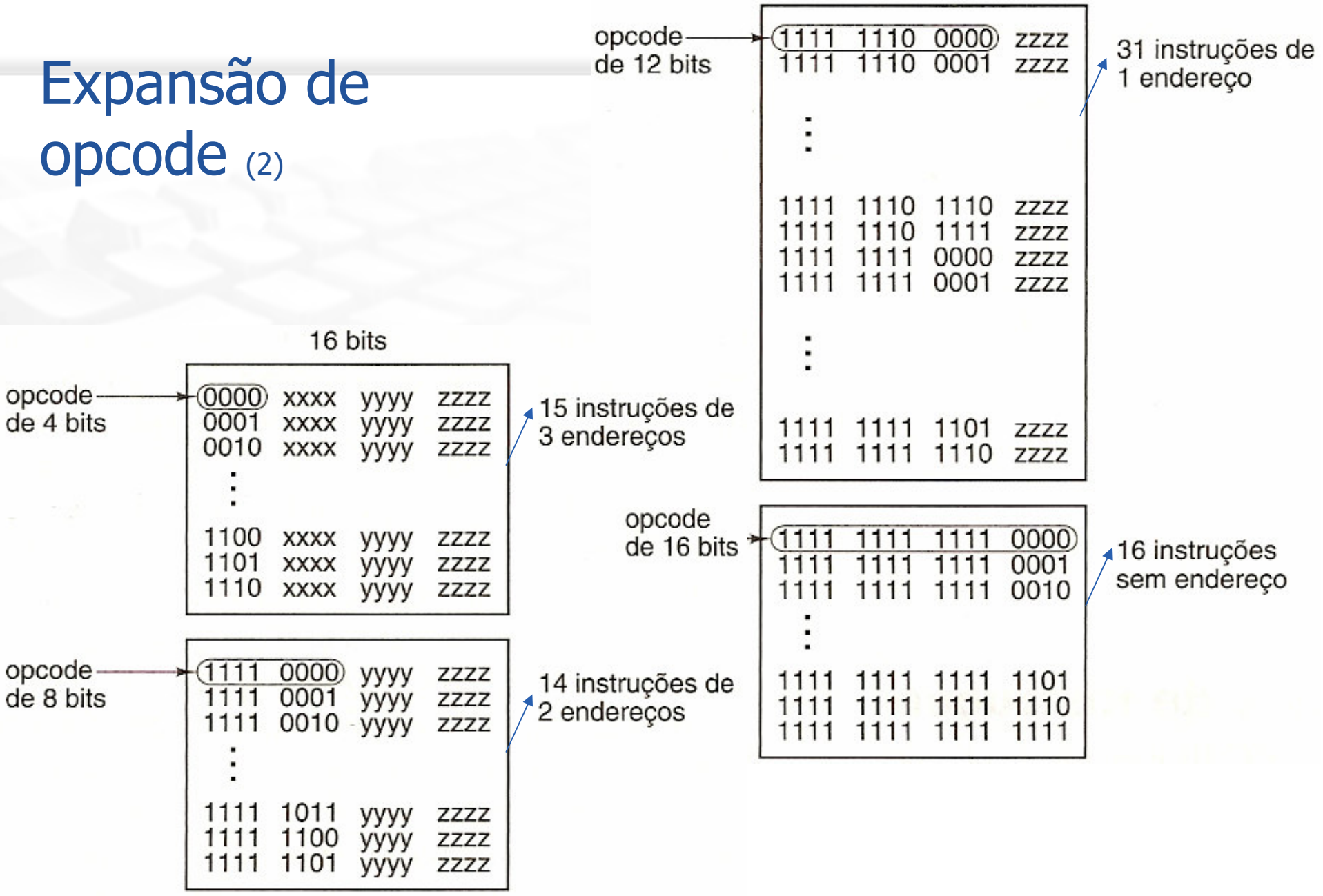
- Importante!
  - O tamanho ideal de uma instrução deve considerar, além do preço da memória, o tempo de decodificação e de execução de uma instrução
  - Como os processadores modernos são capazes de executar várias instruções no mesmo ciclo de clock, torna-se imperativo um mecanismo de busca de várias instruções em cada ciclo de clock (memórias cache)
  - Quando uma instrução tem endereços, o tamanho do endereço deve ser compatível com o tamanho máximo da memória do computador
    - Mas, memórias maiores requerem endereços mais longos resultando em instruções maiores
  - Torne mais rápido o caso mais freqüente
    - Por exemplo, valor zero no registrador $0

Expansão de opcode (1)



- 16 instruções de 3 operandos (3 endereços)
- E se precisarmos de mais instruções com diferentes formatos?

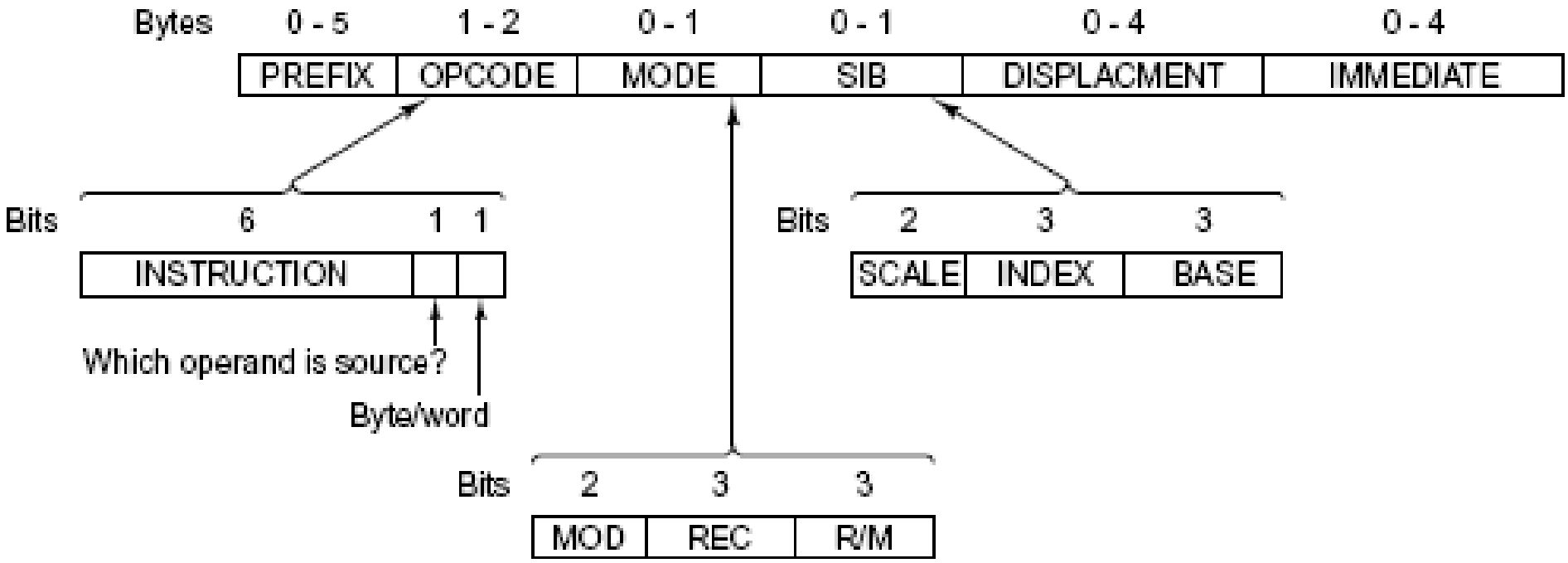
Expansão de opcode (2)



Formato de Instruções do Pentium 4 (1)

- Possuem até 6 campos de tamanhos variáveis.
- **Byte de Prefixo:** um código de operação extra, colocado à frente de uma instrução para modificar sua ação.
- **Código de Operação:** determina a operação a ser executada. Possui dois bits de mais baixa ordem para determinar o uso de um byte ou de uma palavra e o outro para indicar se o endereço de memória (quando houver) é de fonte ou destino.
- A maioria das instruções que referenciam um operando na memória possui também informações sobre esse operando.
  - **MODE, REG e R/M:** usados também para especificar registradores que contém o operando (EAX, EBX, ECX, EDX, ESI, EDI, EBP e ESP)
- **SIB (Scale, Index, Base)**: Escala, Índice, Base
- **Deslocamento:** Endereço de memória. (1, 2 ou 4 bytes)
- **Imediato:** uma constante, um operando imediato. (1, 2, 4 bytes)

Formato de Instruções do Pentium 4 (2)



Formato de Instruções do Pentium 4 (3)

Amostra das instruções internas do Pentium 4

Moves

MOV DST,SRC	Move SRC to DST
PUSH SRC	Push SRC onto the stack
POP DST	Pop a word from the stack to DST
XCHG DS1,DS2	Exchange DS1 and DS2
LEA DST,SRC	Load effective addr of SRC into DST
CMOV DST,SRC	Conditional move

Arithmetic

ADD DST,SRC	Add SRC to DST
SUB DST,SRC	Subtract DST from SRC
MUL SRC	Multiply EAX by SRC (unsigned)
IMUL SRC	Multiply EAX by SRC (signed)
DIV SRC	Divide EDX:EAX by SRC (unsigned)
IDIV SRC	Divide EDX:EAX by SRC (signed)
ADC DST,SRC	Add SRC to DST, then add carry bit
SBB DST,SRC	Subtract DST & carry from SRC
INC DST	Add 1 to DST
DEC DST	Subtract 1 from DST
NEG DST	Negate DST (subtract it from 0)

Binary coded decimal

DAA	Decimal adjust
DAS	Decimal adjust for subtraction
AAA	ASCII adjust for addition
AAS	ASCII adjust for subtraction
AAM	ASCII adjust for multiplication
AAD	ASCII adjust for division

Boolean

AND DST,SRC	Boolean AND SRC into DST
OR DST,SRC	Boolean OR SRC into DST
XOR DST,SRC	Boolean Exclusive OR SRC to DST
NOT DST	Replace DST with 1's complement

Shift/rotate

SAL/SAR DST,#	Shift DST left/right # bits
SHL/SHR DST,#	Logical shift DST left/right # bits
ROL/ROR DST,#	Rotate DST left/right # bits
RCL/RCR DST,#	Rotate DST through carry # bits

Test/compare

TST SRC1,SRC2	Boolean AND operands, set flags
CMP SRC1,SRC2	Set flags based on SRC1 - SRC2

Transfer of control

JMP ADDR	Jump to ADDR
Jxx ADDR	Conditional jumps based on flags
CALL ADDR	Call procedure at ADDR
RET	Return from procedure
IRET	Return from interrupt
LOOPxx	Loop until condition met
INT ADDR	Initiate a software interrupt
INTO	Interrupt if overflow bit is set

Strings

LODS	Load string
STOS	Store string
MOVS	Move string
CMPS	Compare two strings
SCAS	Scan Strings

Condition codes

STC	Set carry bit in EFLAGS register
CLC	Clear carry bit in EFLAGS register
CMC	Complement carry bit in EFLAGS
STD	Set direction bit in EFLAGS register
CLD	Clear direction bit in EFLAGS reg
STI	Set interrupt bit in EFLAGS register
CLI	Clear interrupt bit in EFLAGS reg
PUSHFD	Push EFLAGS register onto stack
POPFD	Pop EFLAGS register from stack
LAHF	Load AH from EFLAGS register
SAHF	Store AH in EFLAGS register

Miscellaneous

SWAP DST	Change endianness of DST
CWQ	Extend EAX to EDX:EAX for division
CWDE	Extend 16-bit number in AX to EAX
ENTER SIZE,LV	Create stack frame with SIZE bytes
LEAVE	Undo stack frame built by ENTER
NOP	No operation
HLT	Halt
IN AL,PORT	Input a byte from PORT to AL
OUT PORT,AL	Output a byte from AL to PORT
WAIT	Wait for an interrupt

SRC = source
DST = destination

# = shift/rotate count
LV = # locals

Formatos de Instruções (3)

- A especificação sobre onde estão os operandos é conhecida como **Endereçamento**
- Uma instrução pode estar acompanhada por um, dois ou três endereços, ou pode não ter endereço algum presente

- Em algumas máquinas, todas as instruções têm o mesmo tamanho. Em outras, esse tamanho pode variar de uma instrução para a outra

Endereçamento (1)

- A especificação sobre onde estão os operandos de uma instrução é conhecida como **Endereçamento**
  - Para cada instrução deve-se especificar em que endereço (registrador ou memória) estará cada operando
- Existe sempre um compromisso entre o tamanho do campo necessário para endereçamento (na instrução) e os possíveis endereços existentes (na memória).

Endereçamento (2)

- A maior parte dos bits de uma instrução acaba sendo usada para indicar de onde vêm os operandos
  - Exemplo:
    - Uma instrução ADD precisa especificar três operandos: dois operandos-fonte e um operando destino
    - Se os endereços de memória são de 32 bits, é natural pensar que esta instrução precisará de 32 bits para cada operando (além dos bits para o opcode)
- Técnicas usadas para reduzir a quantidade de bits gasta na especificação dos operandos
  - Uso de registradores
  - Referenciar operandos de maneira implícita

Endereçamento (3)

- Uso de registradores
  - Se um operando precisar ser usado mais de uma vez -> carregá-lo em um registrador
  - Vantagens
    - Acesso mais rápido
    - São necessários menos bits para se endereçar o operando
      - Ex: Se a arquitetura apresenta 32 registradores, usamos 5 bits p/ especificar um deles
      - Neste caso a operação ADD (com três operandos) só gastaria 15 bits na especificação dos operandos
  - Mas isso tem um custo!
    - Se o operando estiver na memória inicialmente, devemos sempre carregá-lo em algum registrador antes de executar a instrução
      - Ex: Temos que executar duas vezes a instrução LOAD antes de ADD, para trazer os dados a serem somados para os registradores... além de haver um terceiro LOAD após o ADD p/ levar o resultado da operação p/ a memória

Endereçamento (4)

- Referenciar operandos de maneira implícita
  - Reduzir a quantidade de operandos
  - Uma técnica é definir um dos operandos-fonte da operação, como operando destino
    - Por exemplo,

      ADD com 3 operandos

      ADD FONTE1 FONTE2 DESTINO (DESTINO = FONTE1 + FONTE2)

      ADD com apenas 2 endereços (operandos)

      ADD REG_X FONTE1 (REG_X = REG_X + FONTE1)

- Conclusão
  - Diferentes projetistas fazem escolhas diferentes
  - Diferentes **Modos de Endereçamento**

Modos de Endereçamento (1)

- Modos de endereçamento:
  - Determinam como endereços/operandos são especificados pelas instruções
- Instruções especificam operandos:
  - Constante
  - Em registrador
  - Em posição de memória
- Modos de endereçamento:
  - Reduzem número de instruções do programa
  - Aumentam complexidade do hardware
  - Podem aumentar CPI das instruções

Modos de Endereçamento (2)

- **Endereçamento Imediato**
  - O campo da instrução contém o valor do operando.
  - O operando é conhecido como **Operando Imediato.**
  - No exemplo, a constante 4 é carregada no registrador R1.

MOV	R1	4

  - Não precisa de referências adicionais à memória para a busca do operando, pois o valor do operando está na própria instrução.
  - Só permite a definição de valores constantes para um operando
  - O número de constantes é limitado pelo tamanho do campo reservado a esse fim na instrução.
    - n bits para o operando -> 2^n valores diferentes

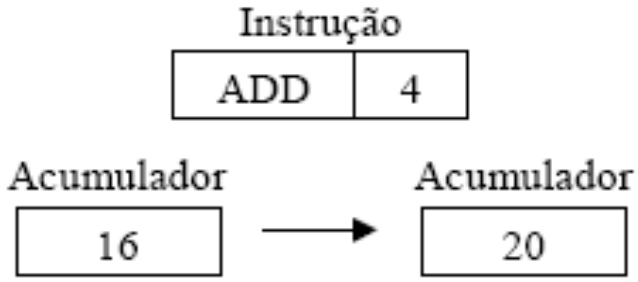
Modos de Endereçamento (3)

- **Endereçamento Imediato (cont.)**
  - Outro exemplo (c/ operando implícito):

ADD	4

ACUMULADOR:=ACUMULADOR+4;

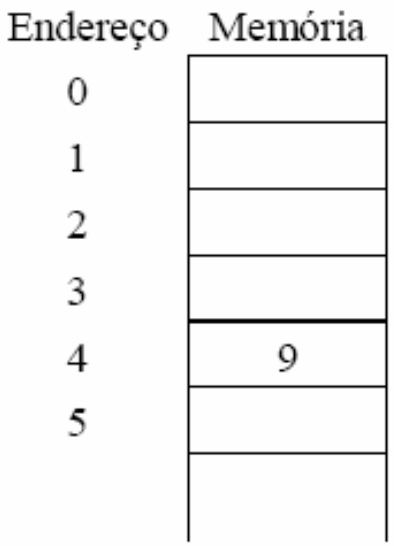
  - Se o valor inicial do acumulador for 16, o seu valor será modificado para 16 + 4 = 20, após a execução da instrução acima.

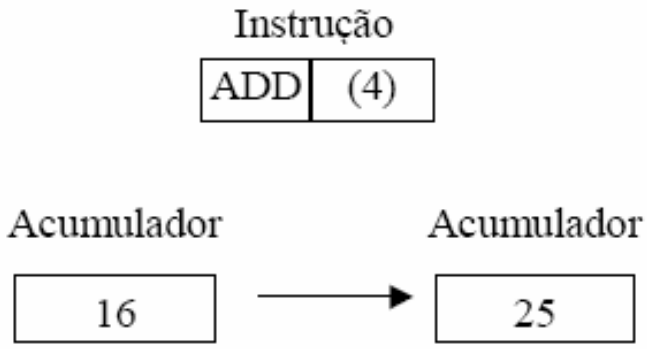


Modos de Endereçamento (4)

- **Endereçamento Direto (Absoluto)**
  - Especifica diretamente um operando armazenado na memória informando seu endereço completo.
  - A instrução usando endereçamento direto vai sempre acessar o mesmo endereço de memória toda vez que for executada
  - Problema: como saber em tempo de compilação qual o endereço exato de uma dado/variável na memória???
    - Em geral usamos

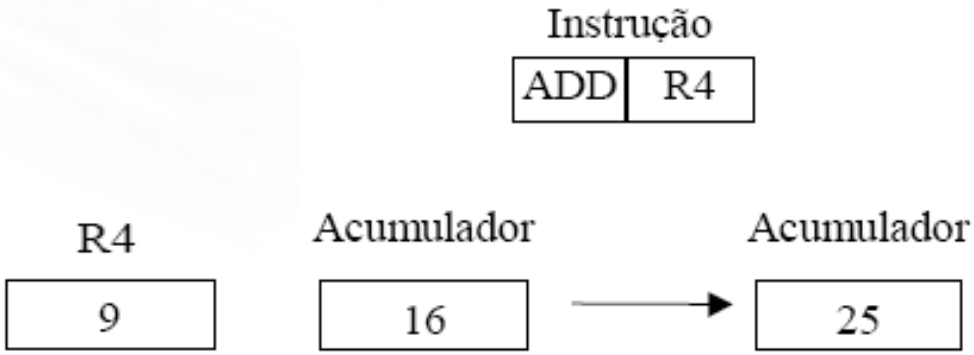
    para variáveis globais...





Modos de Endereçamento (5)

- **Endereçamento Via Registrador**
  - Conceitualmente, semelhante ao endereçamento direto, mas especifica um registrador em vez de um endereço na memória
    - O dado encontra-se no registrador
  - Ex: compiladores tentam descobrir variáveis que estão sendo acessadas muitas vezes (por exemplo dentro de um loop) e alocam um registrador para armazenar seus valores
  - Exemplo:



Modos de Endereçamento (6)

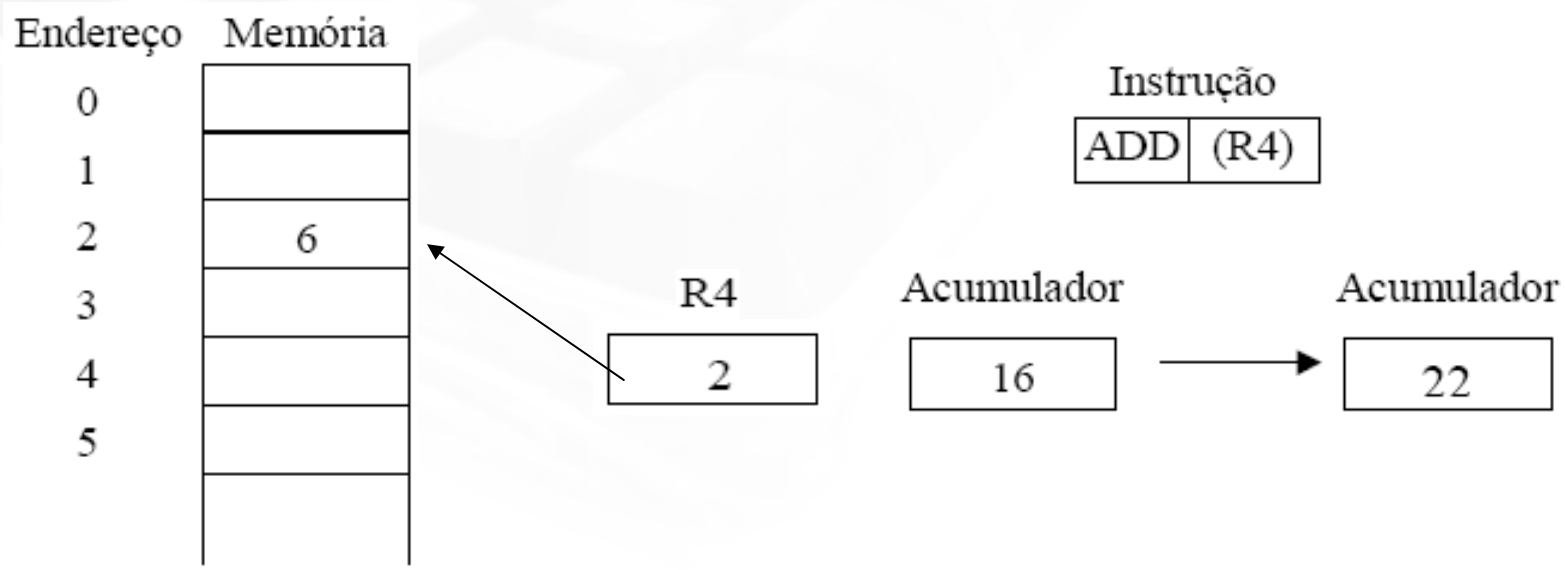
- **Endereçamento Indireto Via Registrador**
  - O operando vem da memória, ou vai para a memória, mas seu endereço não está gravado na instrução, e sim em um registrador
    - Chamamos de **ponteiro** (registrador "aponta" p/ um local na memória)
  - Torna-se possível usar diferentes palavras de memória a cada nova execução de uma mesma instrução
    - Execução de loops
  - Exemplo de instrução:

    ADD (R4)

    Significado: ACUMULADOR := ACUMULADOR + Mem[Reg[R4]]

Modos de Endereçamento (7)

- **Endereçamento Indireto Via Registrador (cont.)**
  - Exemplo:



Modos de Endereçamento (8)

- **Endereçamento Indireto Via Registrador (cont.)**
  - Programa **genérico** em linguagem de montagem para cálculo da soma dos elementos de um vetor (*array*)
    - Acessa as 1024 posições de um array de inteiros para calcular a soma desses valores, armazenando-a no registrador R1
    - no ADD, o primeiro operando recebe o resultado da soma (ñ há ACUMULADOR)
    - # indica constante ou endereço

End. via registrador
End. absoluto

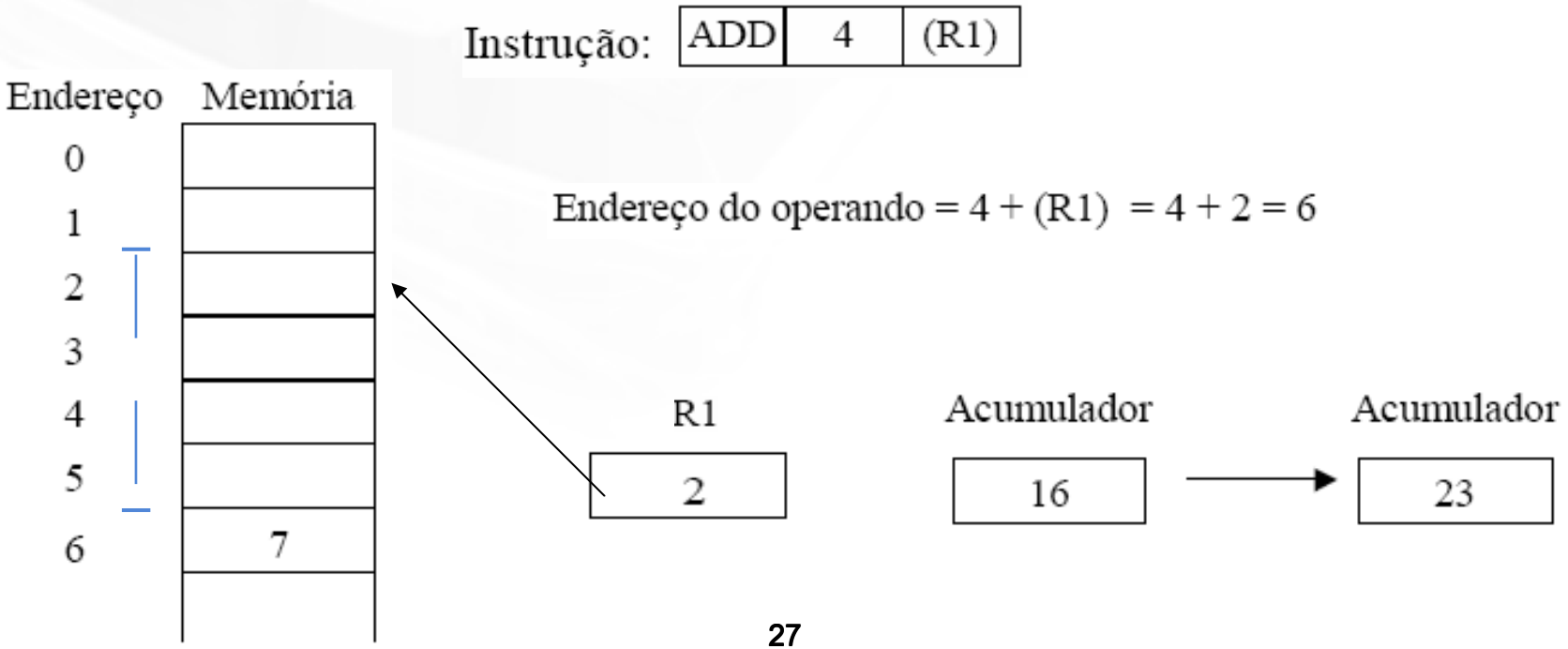
```
        MOV R1,  0          ; acumula soma em R1; valor inicial zero
        MOV R2, #A          ; R2 = endereço do vetor A
        MOV R3, #A+4096     ; R3 = endereço de 1ª palavra após A
LOOP:   ADD R1, (R2)        ; modo indireto de obter operando com base em R2
        ADD R2,  4          ; incrementa R2 de uma palavra (4 bytes)
        CMP R2, R3          ; compara R2 e R3 para testar fim de laço
        BLT LOOP            ; se R2 < R3, não terminou, continua o laço
```

End. indireto via registrador

Modos de Endereçamento (9)

- **Endereçamento Indexado**
  - Referencia palavras de memória situadas a uma distância conhecida a partir do conteúdo de um registrador.
    - O endereço do dado é obtido somando o valor no campo de endereço com o valor contido em um registrador de índice.

Voltando ao Exemplo do ADD usando o registrador Acumulador!!!



Modos de Endereçamento (10)

- **Endereçamento Indexado (cont.)**
  - Exemplo: resultado da soma de dois vetores A e B, cada um com 1024 elementos.
    - Elementos inteiros, palavras de 4 bytes.
    - Elementos são somados um a um em laço.
    - Registrador R1 armazena a soma acumulada dos elementos.
    - Registrador R2 = índice i usado para percorrer os vetores.
    - Registrador R3 armazena o primeiro índice fora dos vetores.
    - Registrador R4 (rascunho) armazena a soma A(i) + B(i).
    - Instrução MOV R4, A(R2): fonte usa modo indexado, com A sendo o deslocamento a ser somado ao conteúdo do registrador de índice R2; destino endereçado via registrador R4
    - Instrução ADD R4, B(R2): idem, com B = deslocamento.

Modos de Endereçamento (11)

- **Endereçamento Indexado (cont.)**

```
        MOV R1,  0        ; Acumula soma de elementos em R1, valor inicial zero
        MOV R2,  0        ; R2 = índice i da soma corrente A(i) + B(i)
        MOV R3,  4096     ; valor do 1º valor inválido do índice
LOOP:   MOV R4,A(R2)      ; R4 = A(i)
        ADD R4,B(R2)      ; R4 = A(i) + B(i)
        ADD R1,R4         ; soma acumulada dos elementos de A(i) + B(i)
        ADD R2,  4        ; incrementa índice do tamanho da palavra: i = i + 4
        CMP R2,R3         ; compara R2 e R3 para testar fim de laço
        BLT LOOP          ; se R2 < R3, não Terminou, continua o laço
```

End. indexado

Modos de Endereçamento (12)

- **Endereçamento Base-Indexado**
  - O endereço de memória é calculado somando-se o conteúdo de dois registradores com um deslocamento (opcional).
  - O conteúdo da base (Registrador-Base) é fixo, enquanto que o conteúdo do índice (Registrador-Índice) é incrementado, como no caso anterior.
  - No exemplo anterior, as instruções indexadas podem ser substituídas por base-indexadas, com os endereços de A em R5 e B em R6:

```
LOOP:  MOV R4,A(R2)          LOOP:  MOVE R4,(R2+R5)
       ADD R4,B(R2)                 ADD R4, (R2+R6)
```

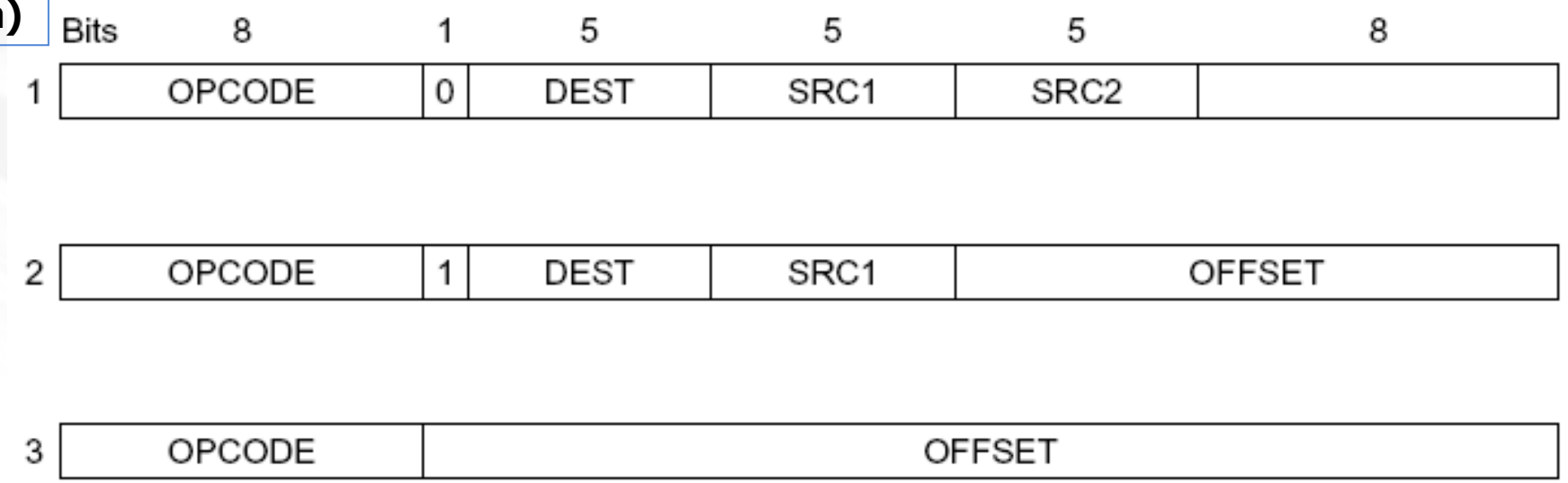
Modos de Endereçamento (13)

Addressing mode	Pentium II	UltraSPARC II	JVM
Immediate	×	×	×
Direct	×		
Register	×	×	
Register indirect	×		
Indexed	×	×	×
Based-indexed		×	

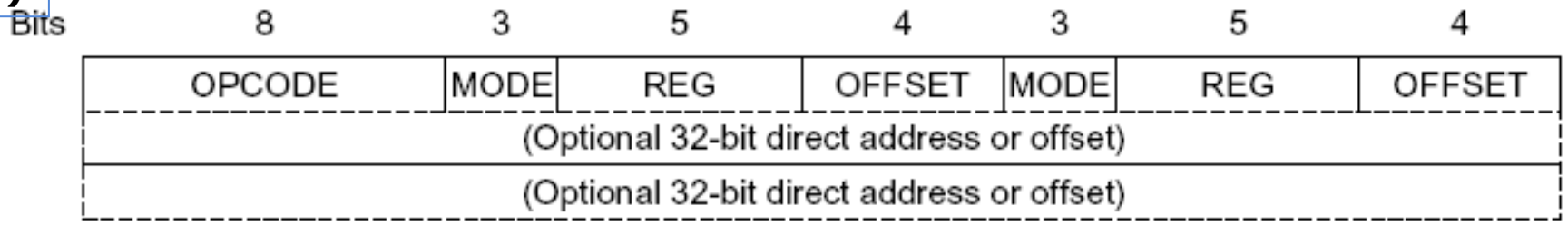
Modos de Endereçamento (14)

Diferentes projetos de formatos de instruções.

(a)



(b)



Tipos de Instruções (1)

- Instruções de Transferências
- Instruções aritméticas
- Instruções lógicas
- Instruções de manipulação de Strings
- Instruções de Entrada/Saída
- **Instruções de comparação de valores**
- **Instruções de desvio**
- **Instruções de Chamada a Procedimento**
- **...**

Fluxo de Controle das Instruções

Tipos de Instruções (2)

- **Instruções de Transferências (Movimento de Dados)**
  - Criam uma cópia de um dado armazenado em algum lugar (fonte) e armazenam essa cópia em outro lugar (destino);
  - Precisam especificar o endereço da fonte e do destino;

Instruções de movimentação de dados	
`MOV DST, SRC`	Move SRC para DST
`PUSH SRC`	Coloca SRC na pilha
`POP DST`	Tira valor da pilha e armazena em DST
`XCHG DS1, DS2`	Troca DS1 com DS2

Tipos de Instruções (3)

- **Instruções Aritméticas**
  - Adição, subtração, multiplicação e divisão

Instruções aritméticas	
`ADD DST, SRC`	Soma DST (destino) com SRC (fonte)
`SUB DST, SRC`	Subtrai DST de SRC
`MUL SRC`	Multiplica o valor de EAX por SRC
`INC DST`	Soma uma unidade a DST
`DEC DST`	Subtrai uma unidade de DST
`NEG DST`	Nega DST (subtrai seu valor de 0)

Tipos de Instruções (4)

Instruções Lógicas

Instruções booleanas	
`AND DST, SRC`	AND booleano entre SRC e DST
`OR DST, SRC`	OR booleano entre SRC e DST
`XOR DST, SRC`	EXCLUSIVE OR booleano entre SRC e DST

Instruções de deslocamento/rotação

Instruções de deslocamento/rotação	
`SAL/ DST, #N` `SAR`	Desloca DST para esquerda/direita #N bits
`ROL/ DST, #N` `ROR`	Rotaciona DST para esquerda/direita #N bits

Entrada/Saída (1)

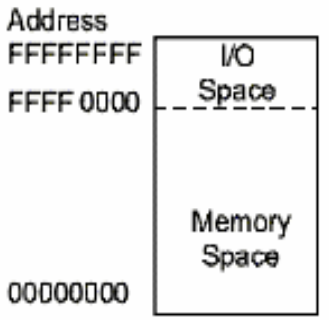
- Existem 3 esquemas diferentes para realizações de E/S em computadores pessoais:
  - **1 – E/S Programada com Espera Ocupada**
  - **2 – E/S Dirigida por Interrupção**
  - **3 – E/S com Acesso Direto à Memória (DMA)**

Entrada/Saída (2)

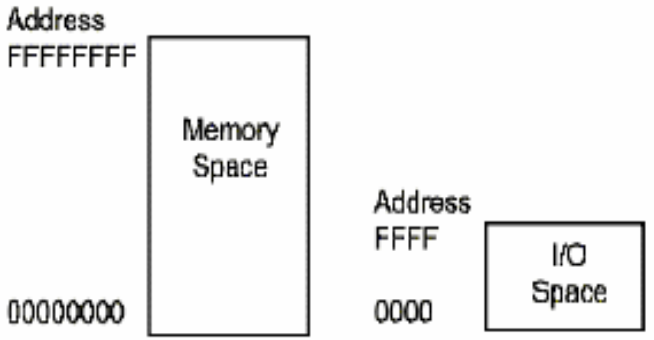
- **E/S Programada com Espera Ocupada**
  - Comumente implementados em microprocessadores de baixa performance (sistemas embarcados, sistemas de tempo real, ...)
  - Há, em geral, uma única **instrução de entrada** e uma única **instrução de saída**. Cada uma dessas instruções seleciona um dos dispositivos de E/S do sistema. Como?
    - Os controladores dos dispositivo de E/S definiem um conjunto de registradores
    - Durante uma operação de E/S a CPU realiza escrita e leitura nesses registradores
      - Assim como a CPU pode fazer leitura e escrita em endereços de memória...
    - Portanto a questão é: Como endereçar esses registradores?

Entrada/Saída (3)

- E/S mapeada em memória
  - São usadas as mesmas instruções que realizam escrita e leitura em endereços de memória (LOAD/STORE)
    - Ex: LOAD EAX , FFFF000D
  - O sistema identifica se é acesso a memória ou a um dispositivo de E/S pelo endereço



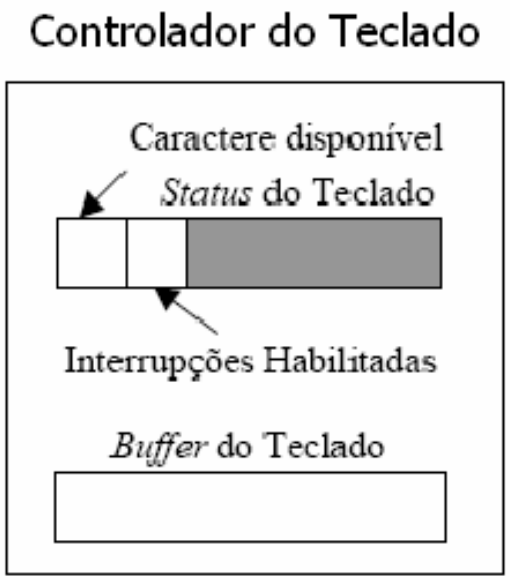
- Espaços de endereçamento separados
  - Os registradores dos controladores de E/S têm endereços específicos (independente dos endereços de memória)
  - São usadas instruções de máquina específicas para ler ou escrever nesses registradores (IN/OUT)
    - Ex: IN EAX , 000D



- Na operação de entrada ou de saída, um único caractere é transmitido entre um **registrador fixo do processador** e um registrador **de um dispositivoo de E/S selecionado** na instrução.

Entrada/Saída (3)

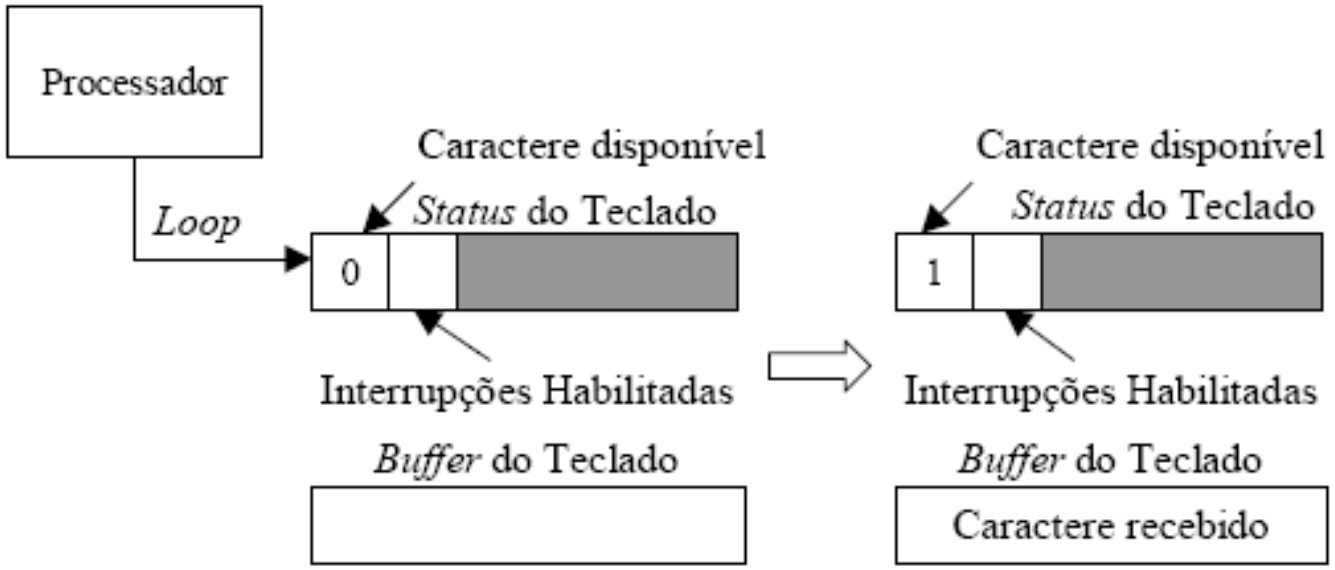
- **E/S Programada com Espera Ocupada:**
  - **Exemplo: Entrada de caracteres do teclado**
    - O bit mais à esquerda do registrador ***Status do Teclado*** indica se existe algum caracter disponível no Registrador ***Buffer do Teclado***
    - Sempre que o usuário digita uma tecla, o codigo ASCII da mesma é armazenado pelo Controlador no Registrador ***Buffer do Teclado***



Dois registradores de 1 byte cada

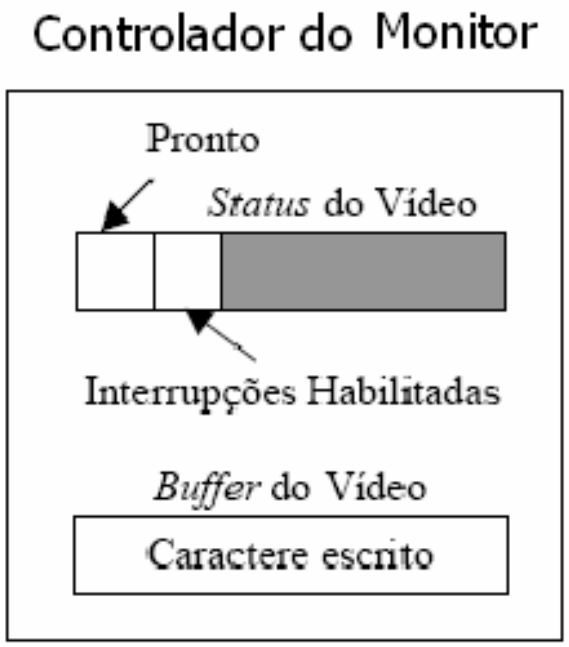
Entrada/Saída (4)

- **Exemplo: Entrada de caracteres do teclado** (cont.)
  - Processador fica em loop lendo o registrador ***Status do Teclado*** até que o bit ***Caractere disponível*** seja ligado pelo dispositivo.
  - Quando esse bit é ligado significa que o teclado acaba de escrever um novo caractere no registrador buffer de dados do teclado.
  - Quando isso ocorre, o programa lê o caractere do registrador de dados do teclado, desligando o bit Caractere disponível



Entrada/Saída (5)

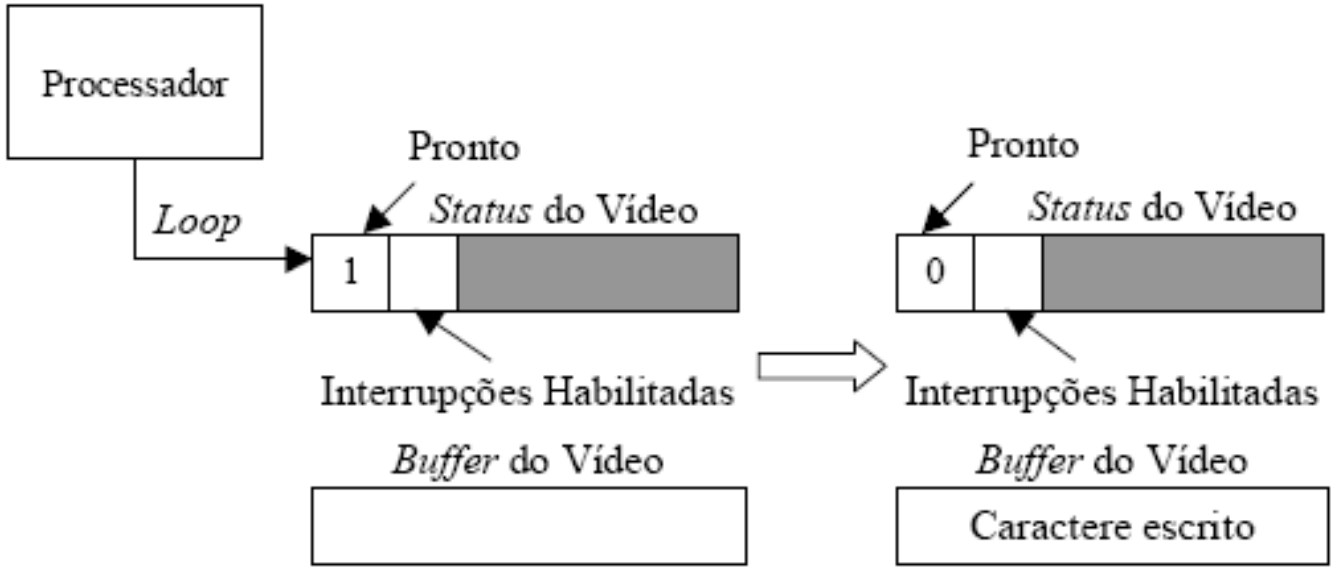
- **E/S Programada com Espera Ocupada:**
  - **Exemplo: <u>Saída</u> de caracteres para o monitor**
    - O bit mais à esquerda do registrador ***Status do Vídeo*** indica que o vídeo está pronto para receber um novo caractere
    - Sempre que a CPU quiser enviar um caractere para o monitor, ela deve escrever o seu código ASCII no Registrador ***Buffer do Vídeo*** e deve Zerar o bit ***Pronto*** do registrador ***Status do Vídeo***



Dois registradores de 1 byte cada

Entrada/Saída (6)

- **Exemplo: <u>Saída</u> de caracteres para o monitor (cont.)**
  - Processador fica em loop lendo registrador ***Status do Vídeo*** até que o bit ***Pronto*** seja ligado pelo dispositivo.
  - Quando esse bit é ligado significa que o vídeo está pronto para receber um novo caractere.
  - Quando isso ocorre, o programa coloca o caractere no registrador de dados do vídeo, desligando o bit ***Pronto***.



Entrada/Saída (7)

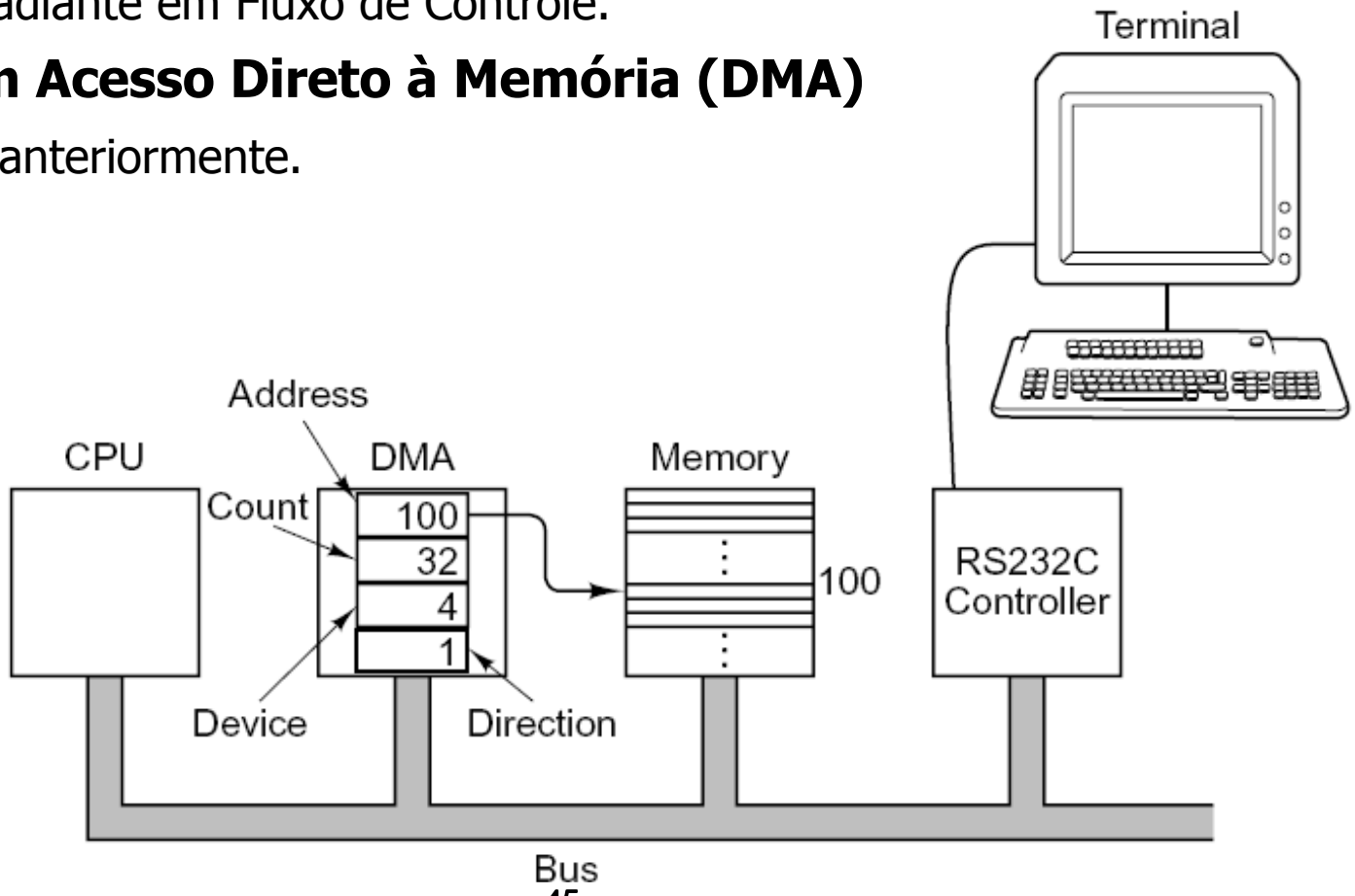
- **Exemplo: Saída de caracteres para o monitor** (cont.)

```
public static void output_buffer (char buf[], int count)
   int status, i , ready;

   for (i=0 ; i< count; i++){
      do{
         status = in(display_status_reg);  //display_status_reg é  o end.
                                             //do registrador Status do Vídeo
         ready = (status >> 7) & 0x01;    //isola o bit Pronto
      } while (ready!= 1);
      out (display_buffer_reg, buf[i])      //display_status_reg é  o end. do end.
                                            // do registrador Buffer do Vídeo
}
```

Entrada/Saída (8)

- **E/S Dirigida por Interrupção**
  - Mais adiante em Fluxo de Controle.
- **E/S com Acesso Direto à Memória (DMA)**
  - Visto anteriormente.



Instruções de comparação e desvio condicional (1)

- Alguns programas precisam testar seus dados e alterar a seqüência de instruções executadas de acordo com o teste feito
- Ex: cálculo do fatorial de um número (n!)

```
if ( n >= 0 ){
      .
      .    // instruções para calcular n!
      .
}
else
     printf("ERRO: número negativo!\n");
```

Instruções de comparação e desvio condicional (2)

- Instruções em linguagem de máquina geralmente:
  - **testam** algum(s) bit(s) da máquina (ex: do registrador PSW) e
  - **desviam** para um endereço de acordo com o valor do bit testado;

Compara os conteúdos, calculando a subtração.

Programa em linguagem de alto nível	Programa em linguagem de montagem

`if (A!=B) {`
`. //instruções`
`. para A!=B`
`.`
`}`
`else{`
`.`
`. //instruções`
`. para A==B`
`}`
	`CMP A,B;`
	`JZS ELSE;`
	`. //instruções`
	`. para A!=B`
	`.`
	`ELSE: .`
	`. //instruções`
	`. para A==B`

Label em ling. de montagem

Instruções de comparação e desvio (condicional) (3)

CMP SRC1, SRC2	Liga os flags com base em SRC1 e SRC2
JMP ADDR	Desvia para ADDR
Jxx ADDR	Desvia condicionalmente para ADDR

- Ex: Instruções de comparação e desvio no Pentium IV

- Também usadas para realizar o controle de loops (como no exemplo em linguagem de montagem que vimos anteriormente)

Linguagem de Montagem de uma máquina hipotética

```
        LOAD R1, #1;
L1      CMP #20, R1;
        JLT L2;
        1º instrução do laço;
                .
                .
                .
        Última instrução do laço;
        INC R1;
        JMP L1
L2      1ª instrução após o laço;
```

Instruções de Chamada a Procedimento (1)

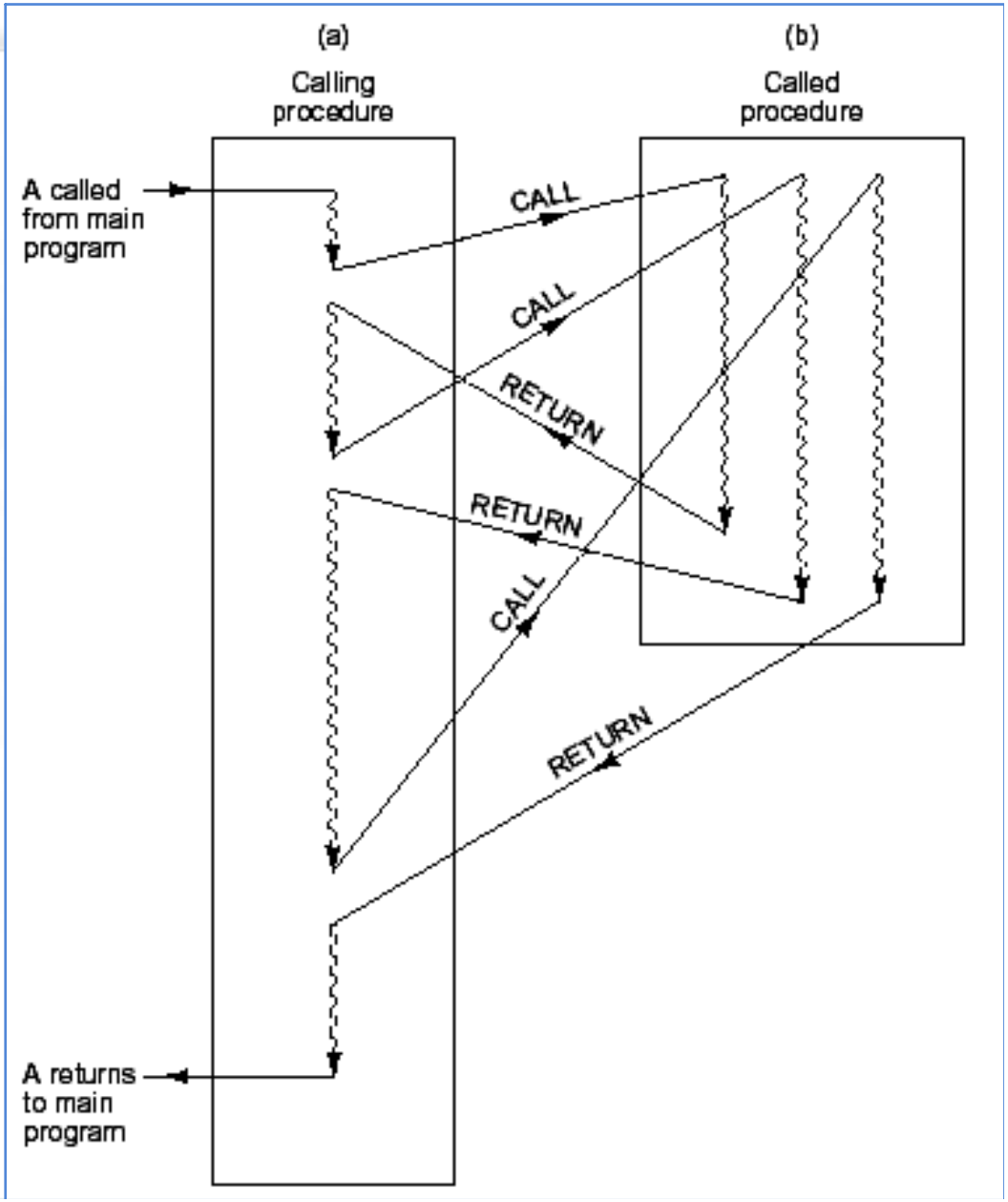
- **Procedimento (Sub-rotina):** um grupo de instruções que realizam uma determinada tarefa
- Podem ser chamados várias vezes de qualquer parte do programa
- Quando um procedimento é chamado através de instrução de chamada de procedimento (CALL), o programa é desviado para a primeira instrução do procedimento
- Quando o procedimento termina sua tarefa o programa é desviado para a instrução imediatamente seguinte a instrução de sua chamada através de instrução de retorno de procedimento (RET).

CALL ADDR	Chama o procedimento em ADDR
RET	Retorno de procedimento

Instruções de procedimento no Pentium 4

Instruções de Chamada a Procedimento (2)

- Essa figura ilustra o funcionamento de um procedimento A que chama B várias vezes
- Sempre que B é chamado ele é executado do início ao fim
- Depois o controle é retornado ao procedimento A
  - Na instrução seguinte à chamada de B



Instruções de Chamada a Procedimento (3)

- Quando um procedimento termina sua tarefa, a execução deve retornar ao comando imediatamente seguinte àquele que chamou o procedimento...
- Deve-se, portanto, **armazenar o endereço de retorno**
- Uma solução muito utilizada é **Armazenagem na Pilha**
  - Ao término da execução do procedimento, o endereço de retorno é retirado da pilha e armazenado no PC

Referências

- Andrew S. Tanenbaum, ***Organização Estruturada de Computadores***, 5ª edição, Prentice-Hall do Brasil, 2007.