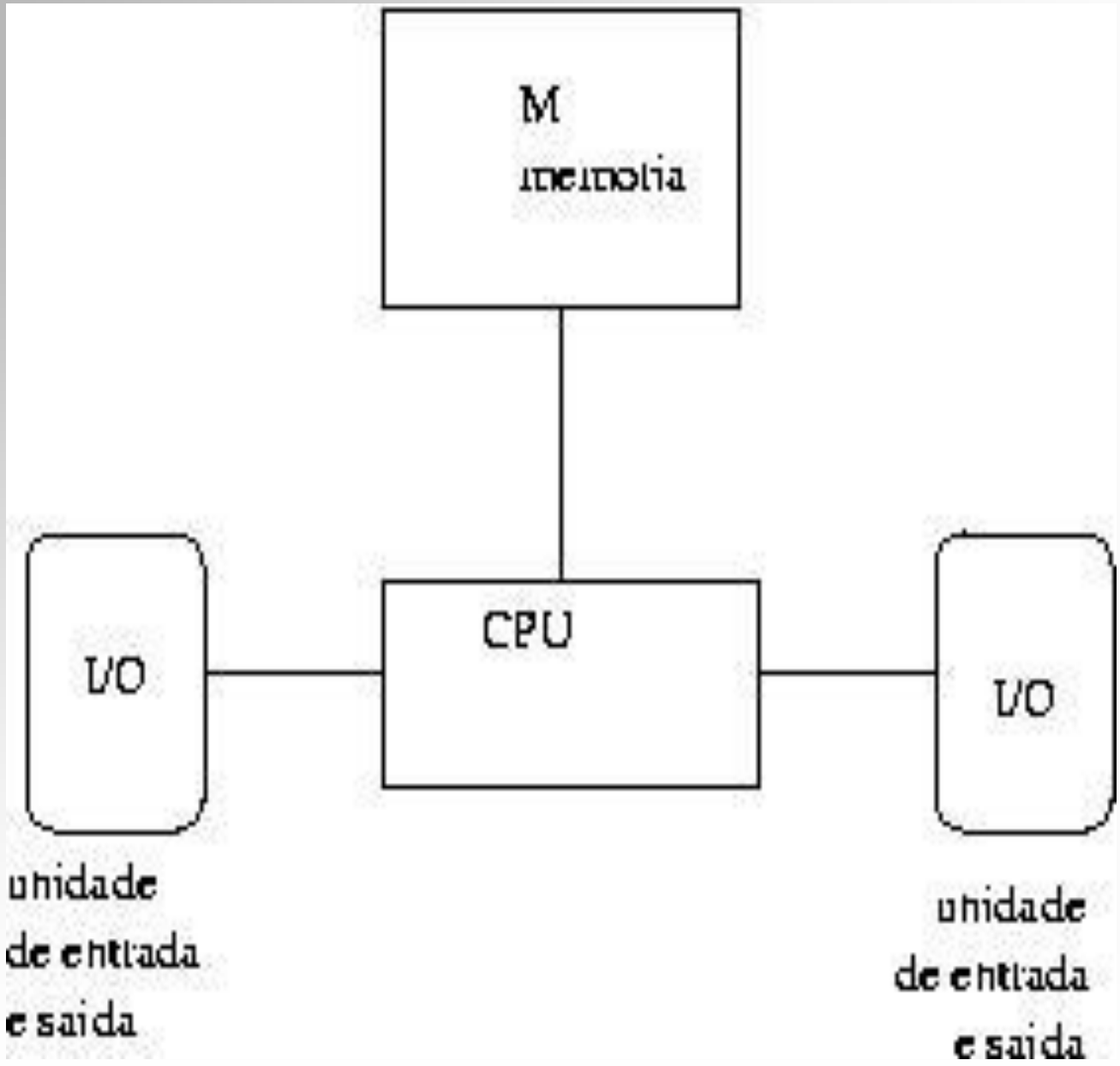
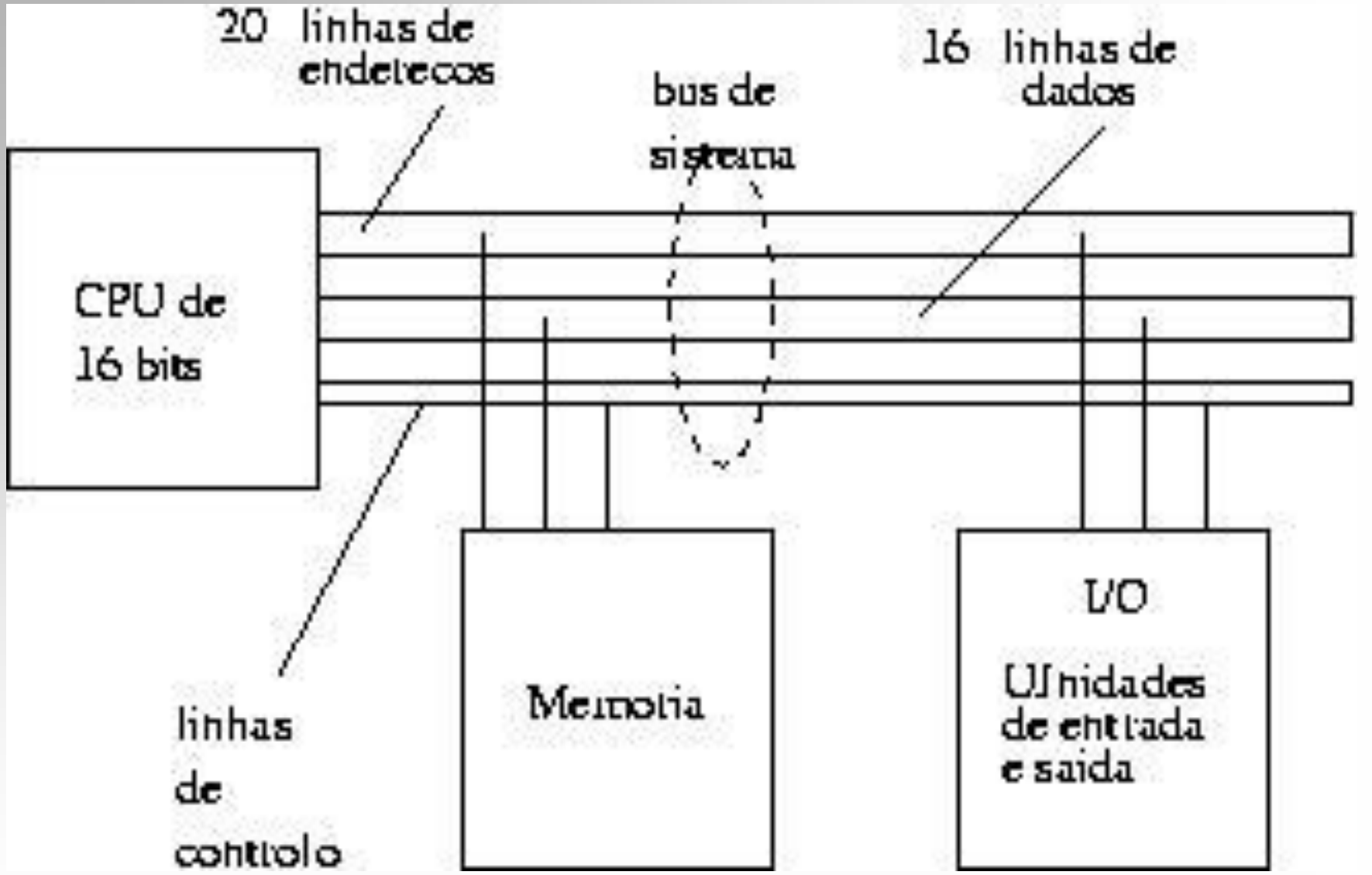
Tópicos sobre Arquitectura de Computadores -- 2

José A. Cardoso e Cunha
DI-FCT/UNL

1. Arquitectura de Von Neumann
2. Programação em linguagem ´´máquina´´
3. Sistema de entradas e saídas
4. Hierarquia das unidades de memória
5. Organização interna do processador


M
I/O
CPU
I/O
unidade
de entrada
e saida
unidade
de entrada
e saida


20 linhas de endereços
bus de sistema
16 linhas de dados
CPU de 16 bits
linhas de controlo
Memória
I/O
Unidades de entrada e saída

Capacidade de memória

- O número máximo de bits do endereço determina a capacidade máxima de memória acessível
- Endereço Capacidade

  8 bits ---------- ?

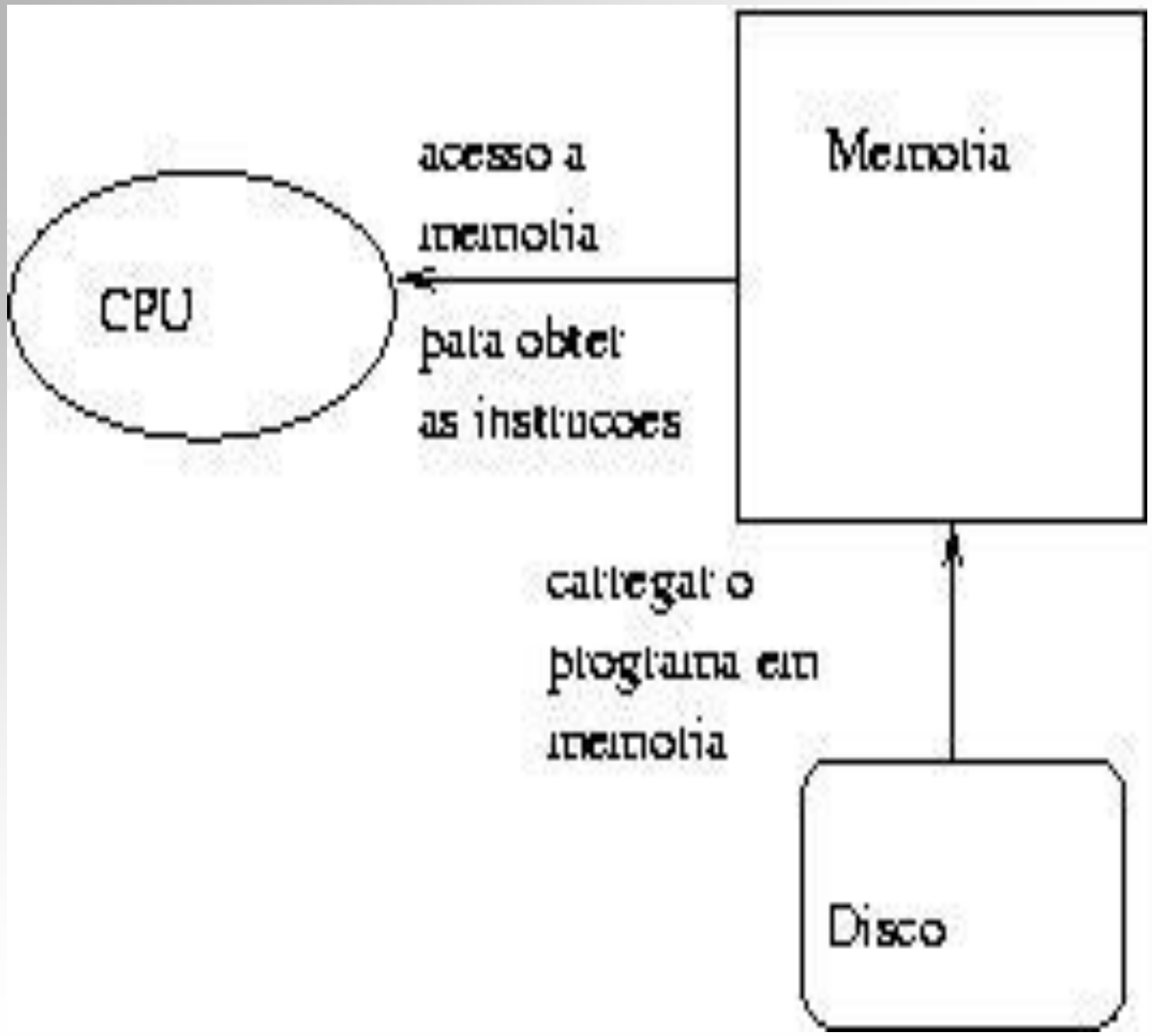
  16 bits

  20 bits

  24 bits

  30 bits

  32 bits
- Depende da memória e da arquitectura do computador


CPU
acceso a
memoria
para obtener
las instrucciones
Memoria
cargar el
programa en
memoria
Disco

Modelo de execução de Von Neumann

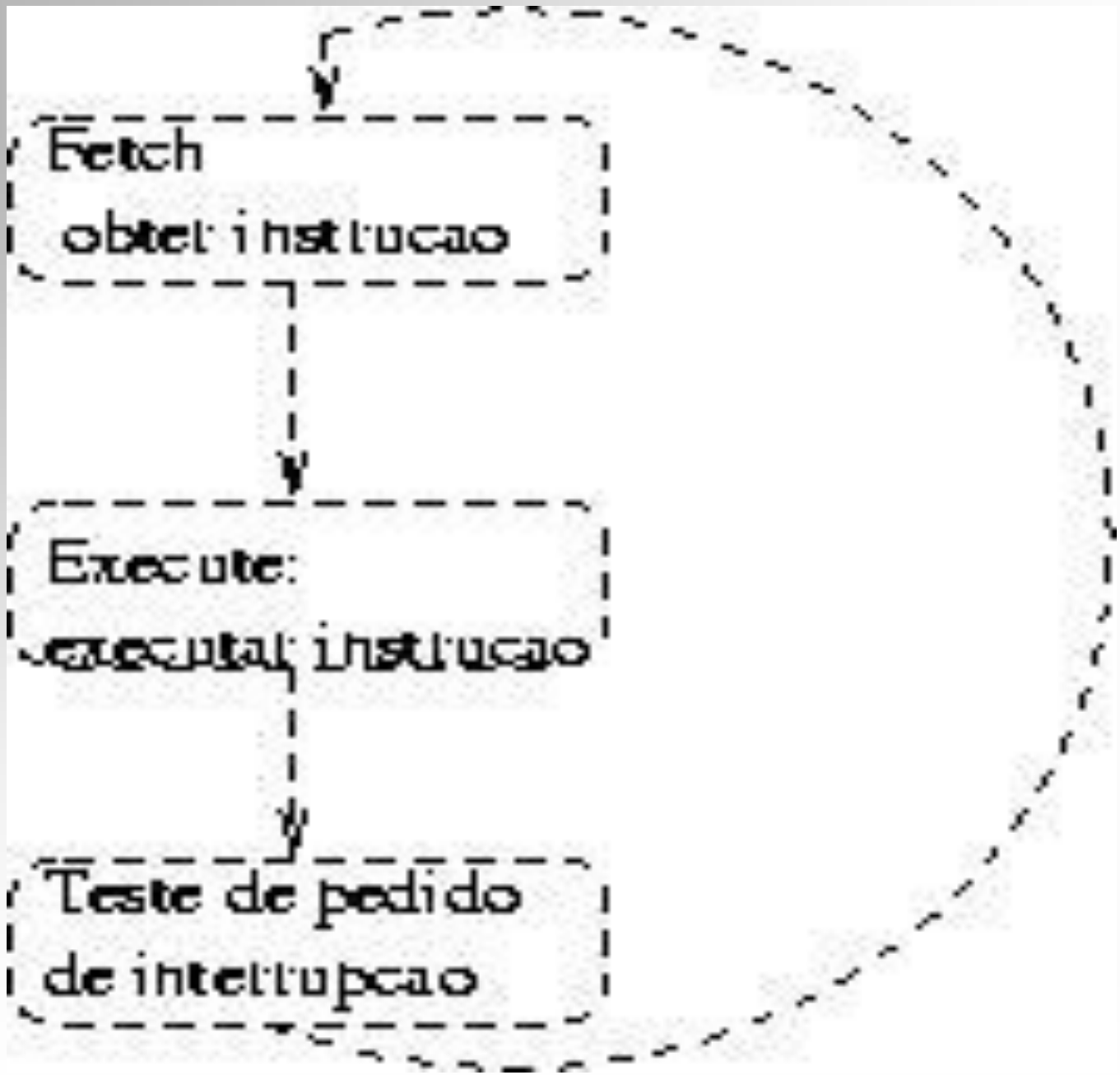
- Inicialmente,um programa é carregado em memória, como uma sequência de instruções, codificadas em binário, em posições consecutivas de memória (bytes)
- Após activado, o CPU inicia um ciclo eterno:

Enquanto ESTADO_ACTIVO fazer

(1) Obter instrução (Inst) de memória; – (*Fetch)*

(2) Executar instrução Inst; - (*Execute)*

- Ao executar a instrução *HALT*, o CPU termina o ciclo.


Fetch
obter instrucao
Execute:
executar instrucao
Teste de pedido
de interrupcao

A execução de Inst pode envolver:

- Operações aritméticas e lógicas (pela ALU)
- Transferências CPU – Memória
- Transferências CPU – Periféricos
- Controlo da sequência de execução de instruções

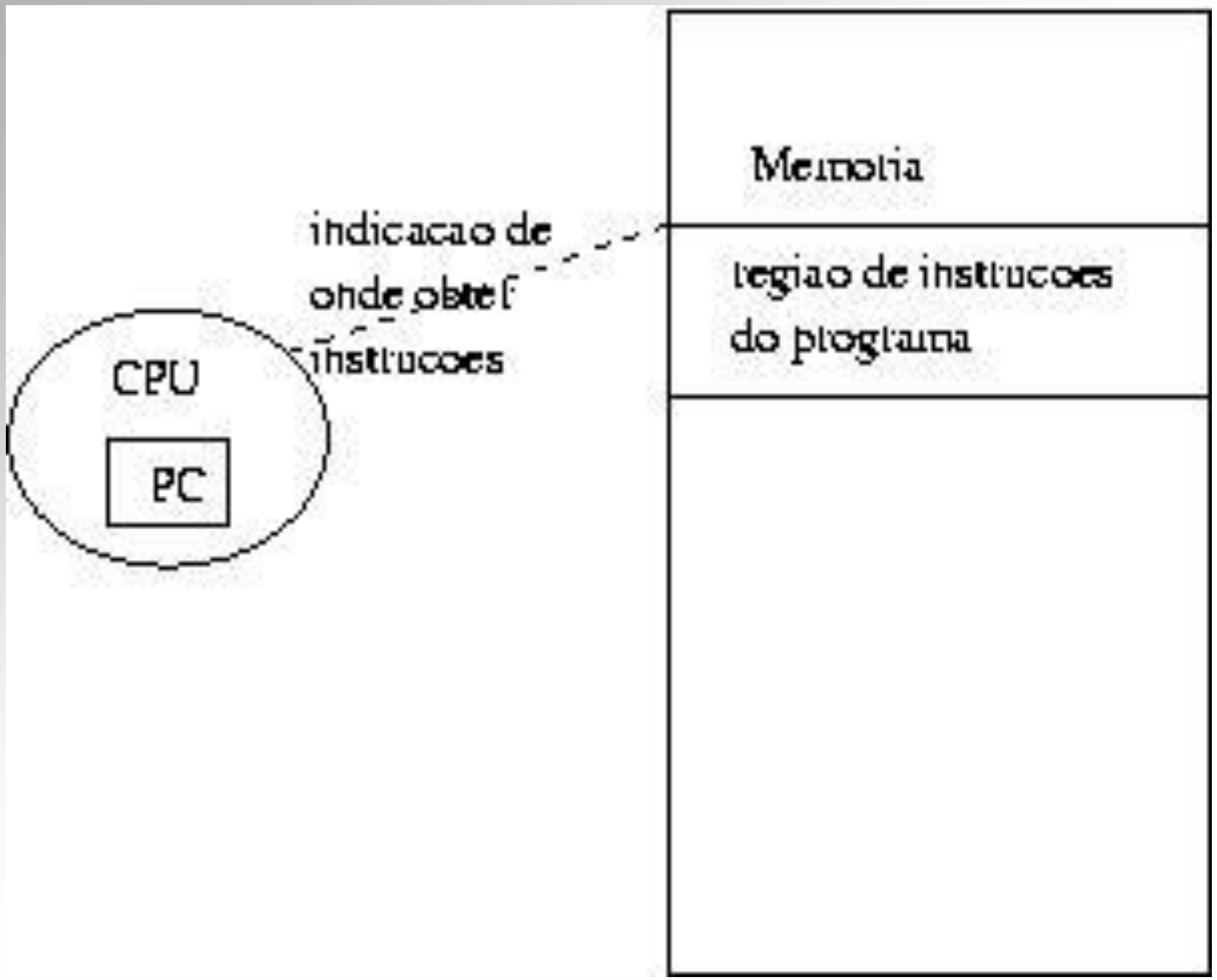
A ALU opera sobre números binários.

A Unidade de controlo do CPU:

- Obtém a próxima instrução de memória;
- Descodifica a instrução
- Emite os sinais de controlo, na micro-arquitectura, correspondentes ao encadeamento de acções necessárias para executar a instrução corrente e as transferências de informação necessárias.

Localizar a instrução em memória

- A que célula de memória deve o CPU ir buscar a instrução, no passo 1 (FETCH)?
- O programa foi previamente colocado em Memória, numa zona de células consecutivas, cujo endereço inicial é então conhecido.
- O CPU consulta um Registador especial – Program Counter (PC), que deve ter sido inicializado com o endereço de memória onde está a primeira instrução do programa:


Memoria
indicacao de
onde obter
instrucoes
regiao de instrucoes
do programa
CPU
PC

Obter uma instrução - FETCH

- Faz uma cópia da representação binária da instrução, a partir da memória, transferindo-a, pelo bus de sistema, para um registador especial do CPU: *Instruction Register (IR)*
- São acções automáticas da Unidade de controlo, no passo 1 do ciclo de execução
- A operação FETCH equivale a IR := Mem[PC]
- A instrução manter-se-á em IR durante toda a sua execução

Instruções máquina

- Um grupo de bits codifica cada instrução e é reconhecido pelo CPU
- Sequências de instruções guardadas em memória são interpretadas pelo ciclo de execução do CPU

A memória central armazena:

- O Programa: sequência de instruções em células consecutivas de memória, codificadas em binário
- ´´Dados´´: valores contidos em células de memória, em binário, que são interpretadas pelo CPU como os operandos das instruções máquina, ou os resultados da sua execução

Coexistem na mesma memória física…

Então:

11101111

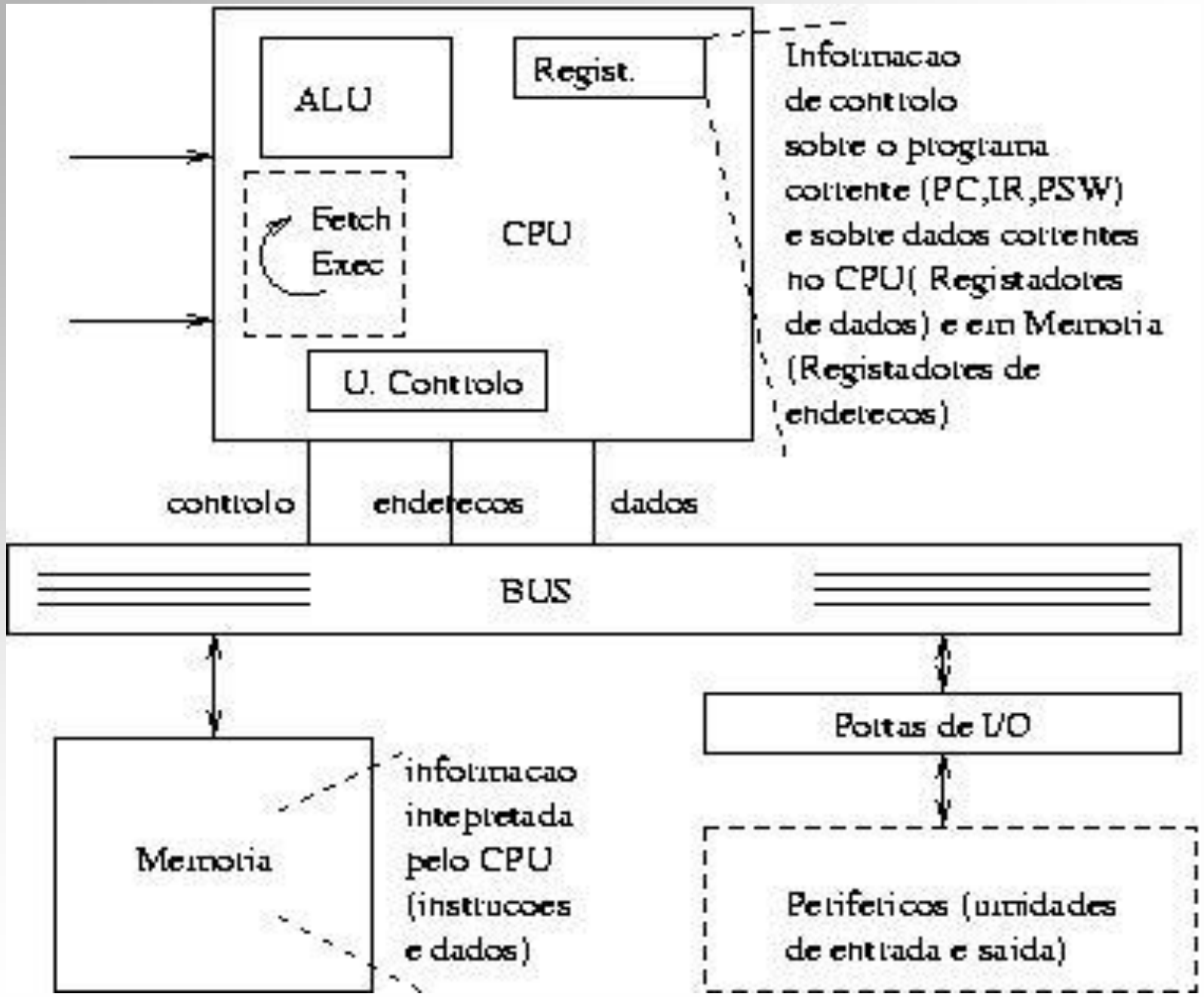
Será uma instrução ou um dado?

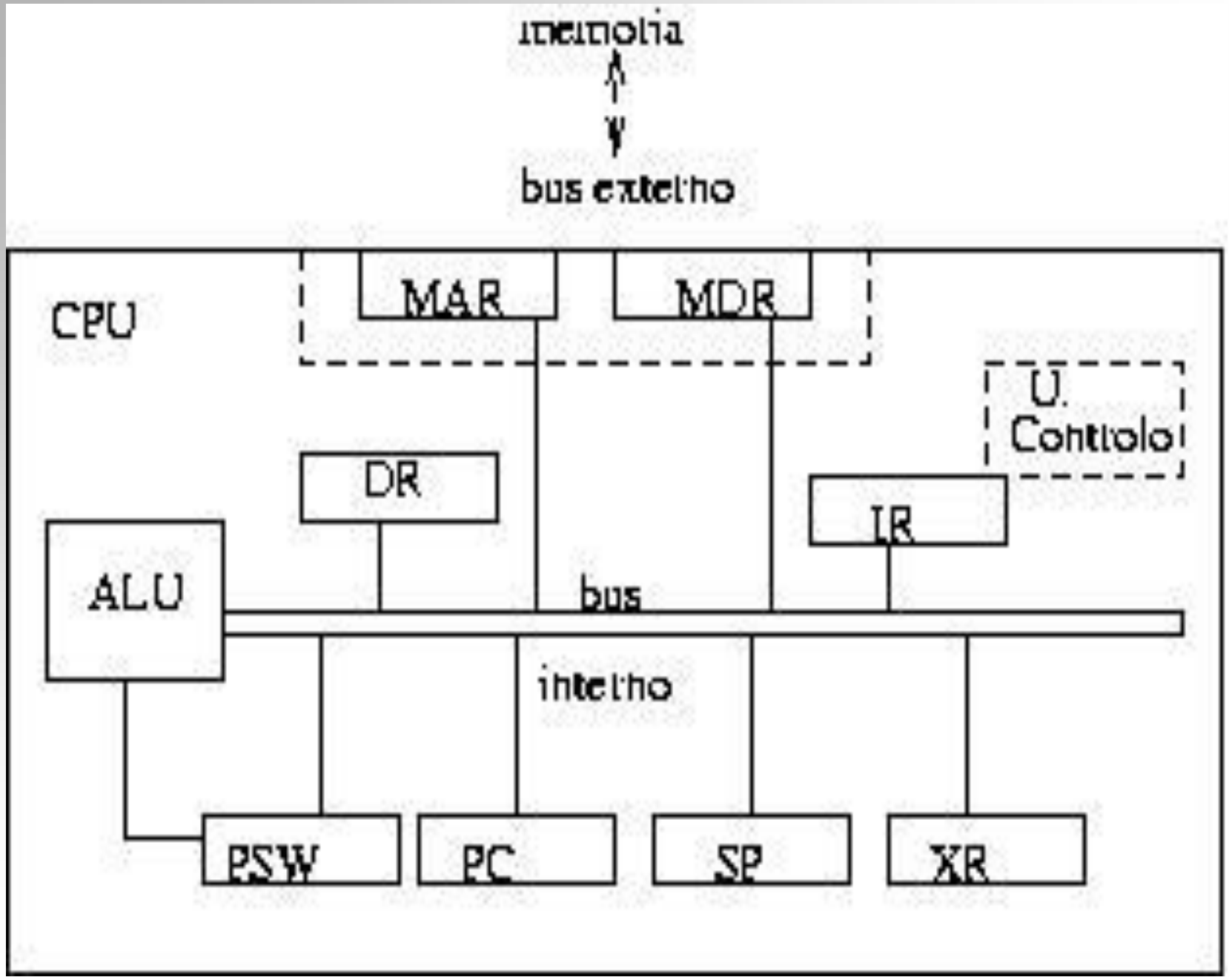
Arquitectura de um computador

Caracterizada por:

- Conjunto de instruções do processador
- Modelo de memória (alcance de um programa, alcance que o bus permite, capacidade de memória)
- Estrutura interna do processador (conjuntos de registadores, etc)
- Topologias de interligação dos componentes CPU, memória e periféricos (estruturas de buses)

- Um mesmo conjunto de instruções pode ter distintas realizações: noção de *família* de processadores, com custos e desempenhos variáveis


ALU
Regist.
Fetch
Exec
CPU
U. Controlo
Informacao
de controlo
sobre o programa
corrente (PC,IR,PSW)
e sobre dados correntes
no CPU(Registadores
de dados) e em Memoria
(Registadores de
endereços)
controlo
endereços
dados
BUS
Portas de I/O
Memoria
informacao
interpretada
pelo CPU
(instrucoes
e dados)
Perifericos (unidades
de entrada e saida)


memoria
bus esterno
CPU
MAR
MDR
U. Controllo
DR
IR
ALU
bus
interno
PSW
PC
SP
XR

Conjunto de instruções do CPU

- Programa: sequência de instruções, em células consecutivas de memória
- Instrução: definição de uma operação elementar, capaz de ser executada pelo CPU

1. Que tipo de instruções se devem incluir na arquitectura?
2. Que informação deve incorporar-se no código de cada instrução?

As classes de instruções necessárias são poucas:

- Aritméticas e lógicas
- Transferências
- Controlo da sequência de execução

Os objectivos a considerar são muitos:

- Aproximar das linguagens de alto nível
- Reduzir o tamanho dos programas
- Oferecer instruções poderosas e eficientes
- Oferecer diversos modos de endereçar as variáveis em memória (ex. int, arrays, records)
- Reduzir o número de acessos a memória central

- Instruções simples,que facilitem realização eficiente
- Oferecer só as instruções de facto úteis
- Promover uma abordagem em que instruções mais complexas não são oferecidas, mas podem ser implementadas, por programas, de forma eficiente

Para as boas soluções, contribuem:

- A definição do conjunto de instruções
- E todos os outros elementos da arquitectura…

A solução boa é também fruto de compromissos:

- Custo
- Funcionalidade
- Desempenho
- Leis do mercado e circunstâncias tecnológicas…

Que instruções incluir: (lista não exaustiva)

- Multiplicação e divisão
- Aritmética de vírgula flutuante
- Aritmética de dupla precisão
- Chamada-retorno de funções/métodos
- Execução eficiente de ciclos e iterações
- Contar e temporizar acções
- Operar sobre cadeias de caracteres, sobre listas, sobre filas, sobre records, sobre matrizes
- Mover e recolocar programas em memória
- Suspensão e reactivação de programas
- ????

A que nível: na arquitectura do computador ou em níveis superiores da hierarquia?

Informação a incluir na instrução

Depende do tipo de instrução:

- O código da operação
- A especificação dos operandos ou sua localização
- A especificação dos resultados (localização)

Exemplo: formato de instrução com 4 campos:

código – end1 -- end2 – end3

Para operações com dois operandos (em end1 e end2), deixando o resultado em end3)

´´end´´ pode indicar uma célula de memória ou um registador do CPU

Quantos bits para o código?

Depende do número de instruções distintas

Exemplo: 8 bits → 256 instruções possíveis

Quantos bits para os endereços?

Depende de:

- onde estão os operandos,
- o modo como se endereça a memória
- a capacidade do espaço de endereços

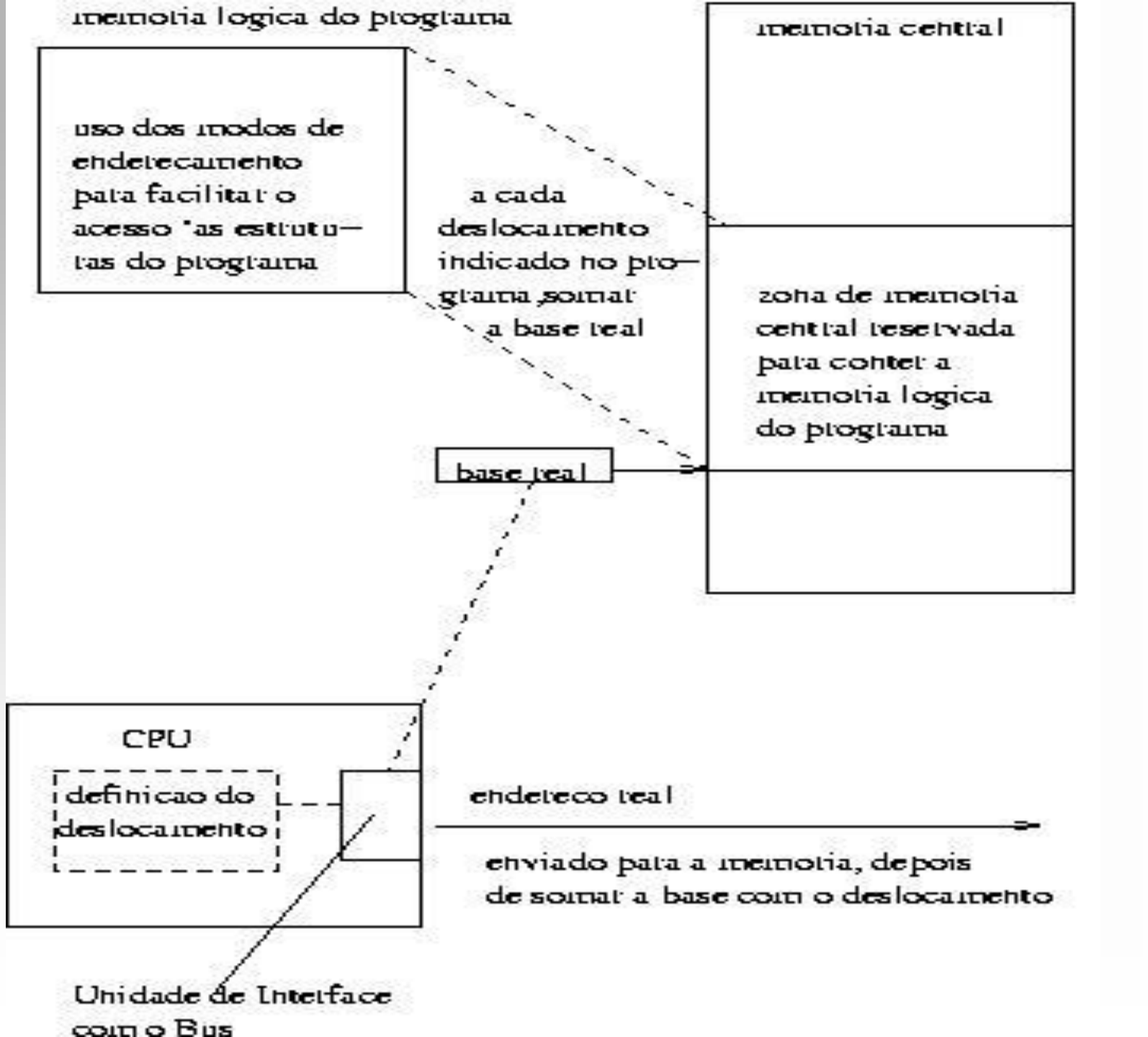
Exemplo:

- Operandos e resultado em memória
- Memória de 64 KiloBytes = 2^{16} Bytes

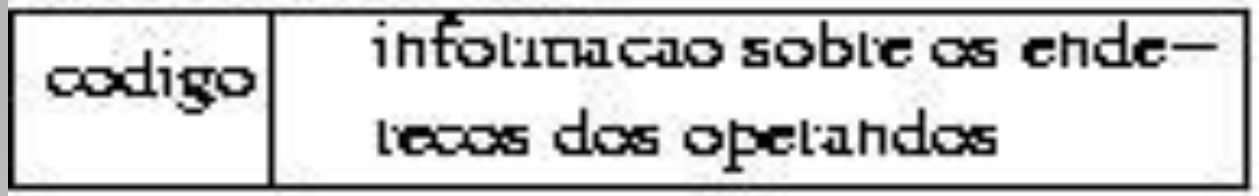
Então: 8 bits + 3 * 16 = 56 bits -- uma instrução

Que alternativas?

- Instruções com menor número de endereços
- Instruções em que só figura uma parte do endereço
- Instruções mais simples terão menos argumentos
- Instruções com operandos e resultado em registadores do CPU: ´´end´´ terá menos bits
- Instruções podem assumir que os operandos estão em células pré-definidas


memoria logica do programa
memoria central
uso dos modos de endereçamento para facilitar o acesso as estruturas do programa
a cada deslocamento indicado no programa somar a base real
zona de memoria central reservada para conter a memoria logica do programa
base real
CPU
definicao do deslocamento
endereco real
enviado para a memoria, depois de somar a base com o deslocamento
Unidade de Interface com o Bus

Instrucao maquina



conforme o modo de endereçamento

produz o endereco efectivo

envia-o para a unidade de interface com o bus externo (Memory Address Register)

Instruções de 3 endereços

código – op1 -- op2 – dest

end1 = mod(op1)

end2 = mod(op2)

end3 = mod(dest)

(end3) ← código((end1),(end2))

Instruções de 2 endereços

código – op1 – dest

end1 = mod(op1)

end2 = mod(dest)

Ou só tem 1 operando, ou tem 2 operandos mas esmaga o segundo:

(end2) ← código((end1),(end2))

Instruções de 1 endereço

código – op

end = mod(op)

Se tiver 2 operandos, assume que um deles foi antes carregado num registador do CPU, implícito (Acumulador AC)

(AC) ← código((end),(AC))

Instruções sem endereços

código

Assumem uma estrutura implícita, na memória ou no processador, sobre a qual operam.

Exemplo: operações sobre uma PILHA (*stack*)

Empilhar Registador: (*push*)

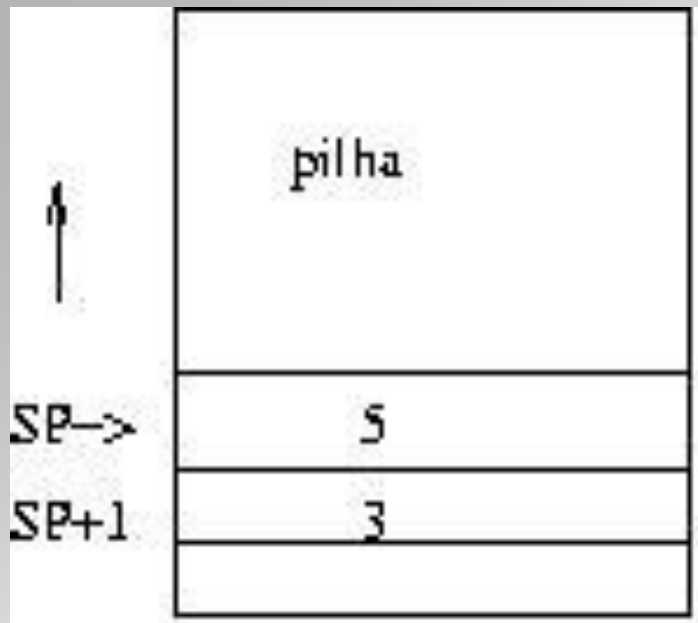
- Cria uma nova posição sobre o Topo da Pilha
- Copia (Registador) para o novo Topo da Pilha

Desempilhar Registador (*pop*)

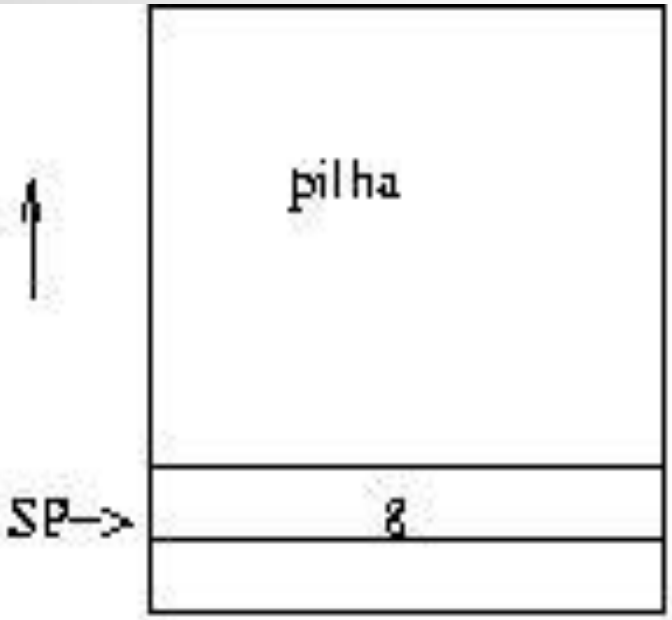
- Copia o valor do Topo da Pilha para o Registador
- e elimina esse valor do Topo da Pilha

Somar

- Desempilha, sucessivamente, os dois elementos no Topo da Pilha e empilha o resultado



(a)



(b)

Empilhar AC:

decrementa SP;

M[SP] := (AC)

sentido decrescente do valor de SP

Desempilhar AC:

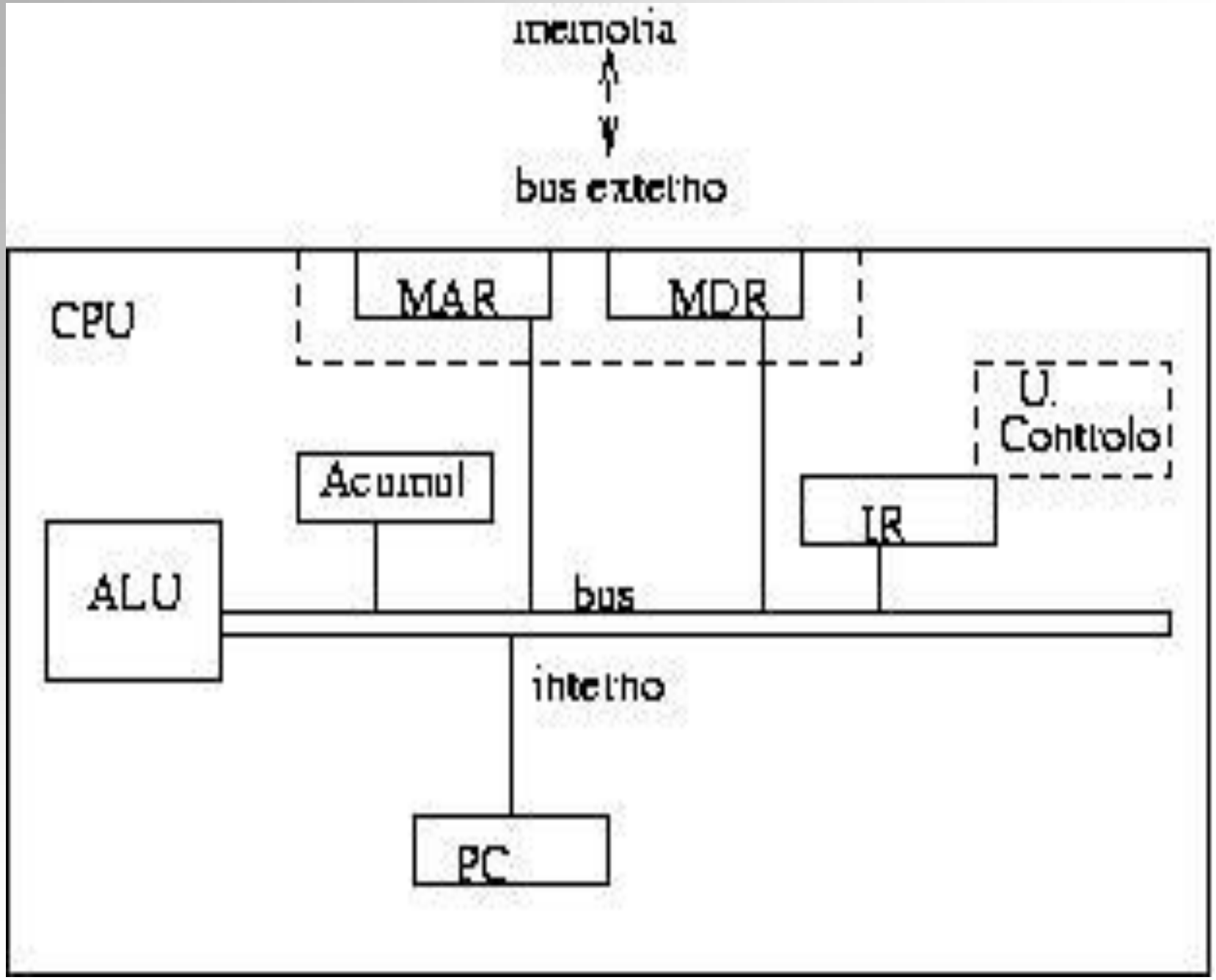
(AC) := M[SP];

incrementa SP;

operacao:

M[SP+1] :=codigo(M[SP], M[SP+1])

Exemplo de um CPU simplificado


memória
bus externo
CPU
MAR
MDR
U. Controlo
Acumul
IR
ALU
bus
interno
PC

Instruções com um formato de 8 bits:

Código I endereço

Código: 3 bits, codifica 8 instruções

Endereço: 5 bits, permite endereçar directamente até 32 posições de memória (bytes)

No CPU:

ALU opera sobre números de 8 bits

PC de 5 bits

IR de 8 bits

Registador de trabalho AC (acumulador) de 8 bits

MAR de 5 bits

MDR de 8 bits

: Na Memória central

Capacidade de 32 células, cada contém 1 byte

Bus de sistema:

5 linhas de endereços

8 linhas de dados

linhas de controlo

Instruções de 1 endereço

load n ----- AC := Mem[n] --- mov AC,n

store n ----- Mem[n] := AC --- mov n, AC

add n ----- AC := AC + Mem[n]

sub n ----- AC := AC – Mem[n]

jump n ----- PC := n

jpos n ----- IF AC > 0 THEN PC := n;

jzero n ----- IF AC == 0 THEN PC := n;

halt ----- Indicador de Paragem := TRUE

em que n é um endereço de memória

Assume-se que:

- Inicialmente PC := 0
- O Programa é carregado em posições consecutivas de memória, a partir do endereço 0

O ciclo do CPU é:

```
WHILE NOT Indicador de Paragem DO
   BEGIN
        IR :=  Mem[PC];   --- FETCH
        PC := PC + 1;       --- prepara próxima I. (!)
        ExecutarInstrução (IR)   --- executa I. corrente
   END;
```

Transferências (´´moves´´)

load n – mov AC, n

carregar AC com o valor de Mem[n]

MAR := n;

Bus de controlo := pedir-leitura

Aguardar;

MDR := valor no Bus de dados

AC := MDR

store n – mov n, AC

carregar o valor de AC em Mem[n]

MAR := n;

MDR := AC

Bus de controlo := pedir-escrita

Instruções Aritméticas

add n -acumular em AC a soma de AC e Mem[n]

MAR := n;

Bus de controlo := pedir-leitura

Aguardar;

MDR := valor no Bus de dados

Activar ALU:

    res := AC + MDR

AC := res

Instruções de salto

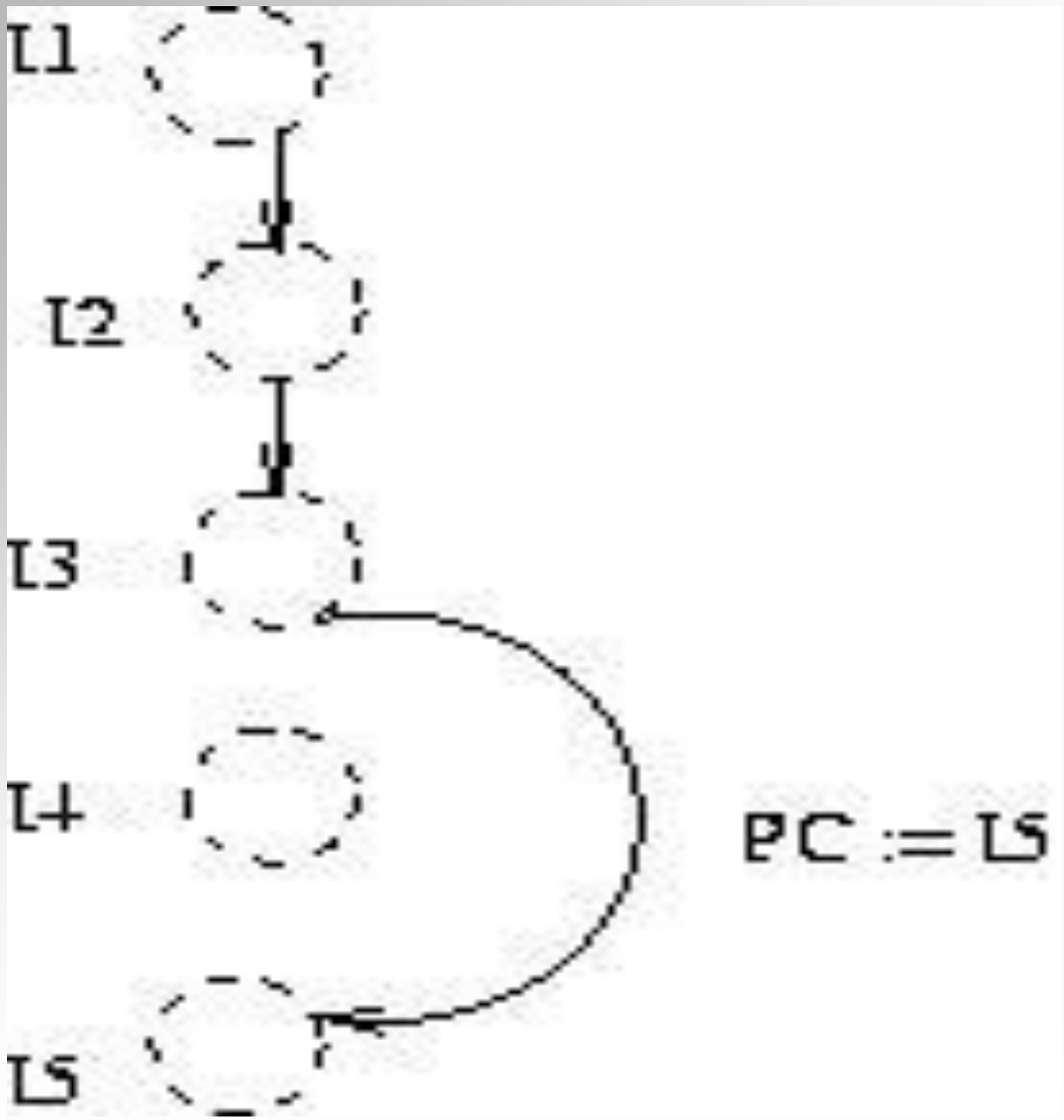
jump n ----- PC := n

jpos n ----- IF AC > 0 THEN PC := n;

jzero n ----- IF AC == 0 THEN PC := n;

Afectam o conteúdo do PC que determina o endereço onde o CPU irá obter a próxima instrução

Se jpos n e jzero n não alterarem o valor do PC, então este valor aponta o endereço seguinte da zona de memória onde está o programa.


L1
L2
L3
L4
L5
PC := L5

Exemplo de programa

Multiplicar os conteúdos de 2 células de memória de endereços de memória E1 e E2 e colocar o resultado na célula de endereço E3.

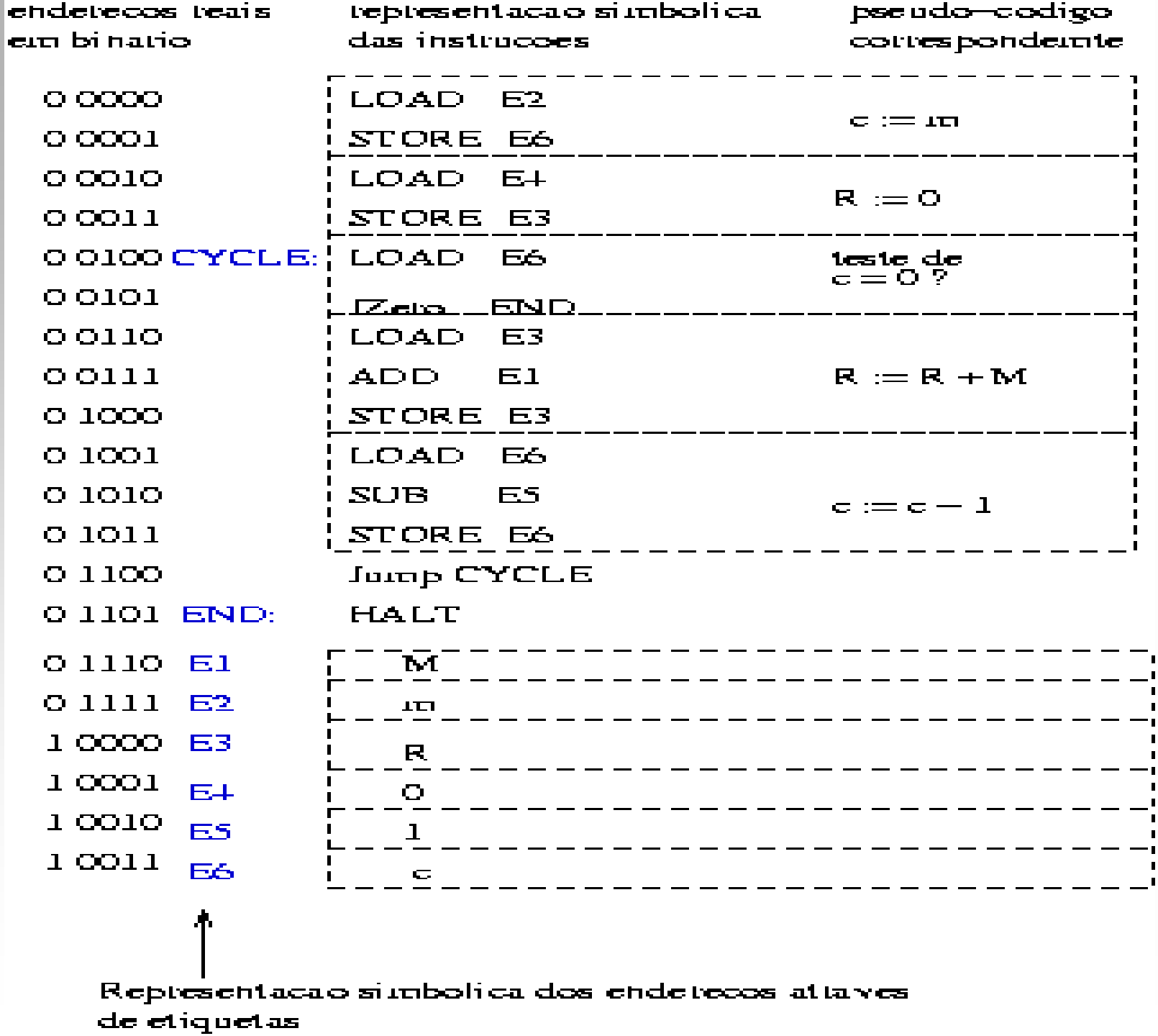
M = Mem[E1]

m = Mem[E2] R : = M * m

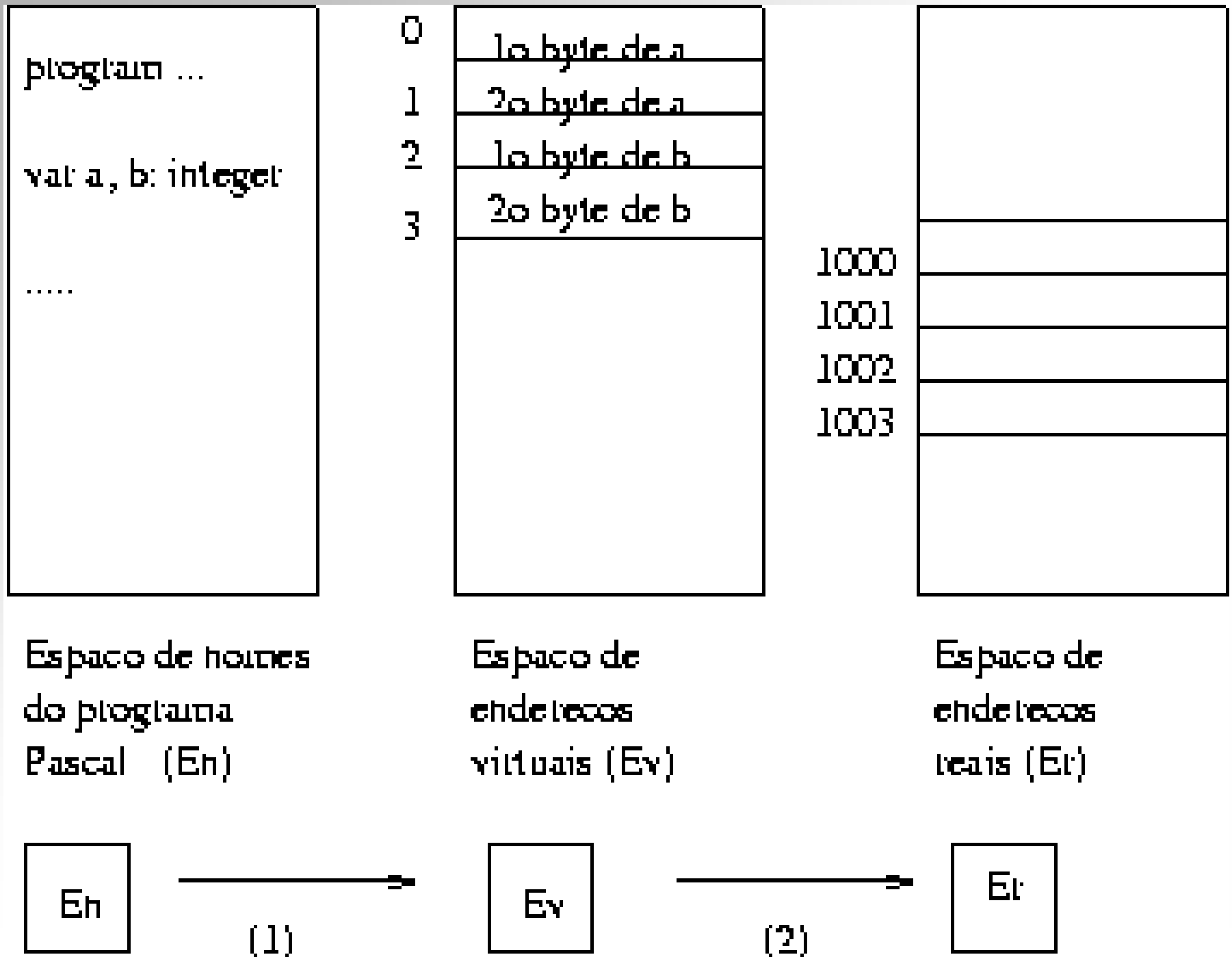
R = Mem[E3]

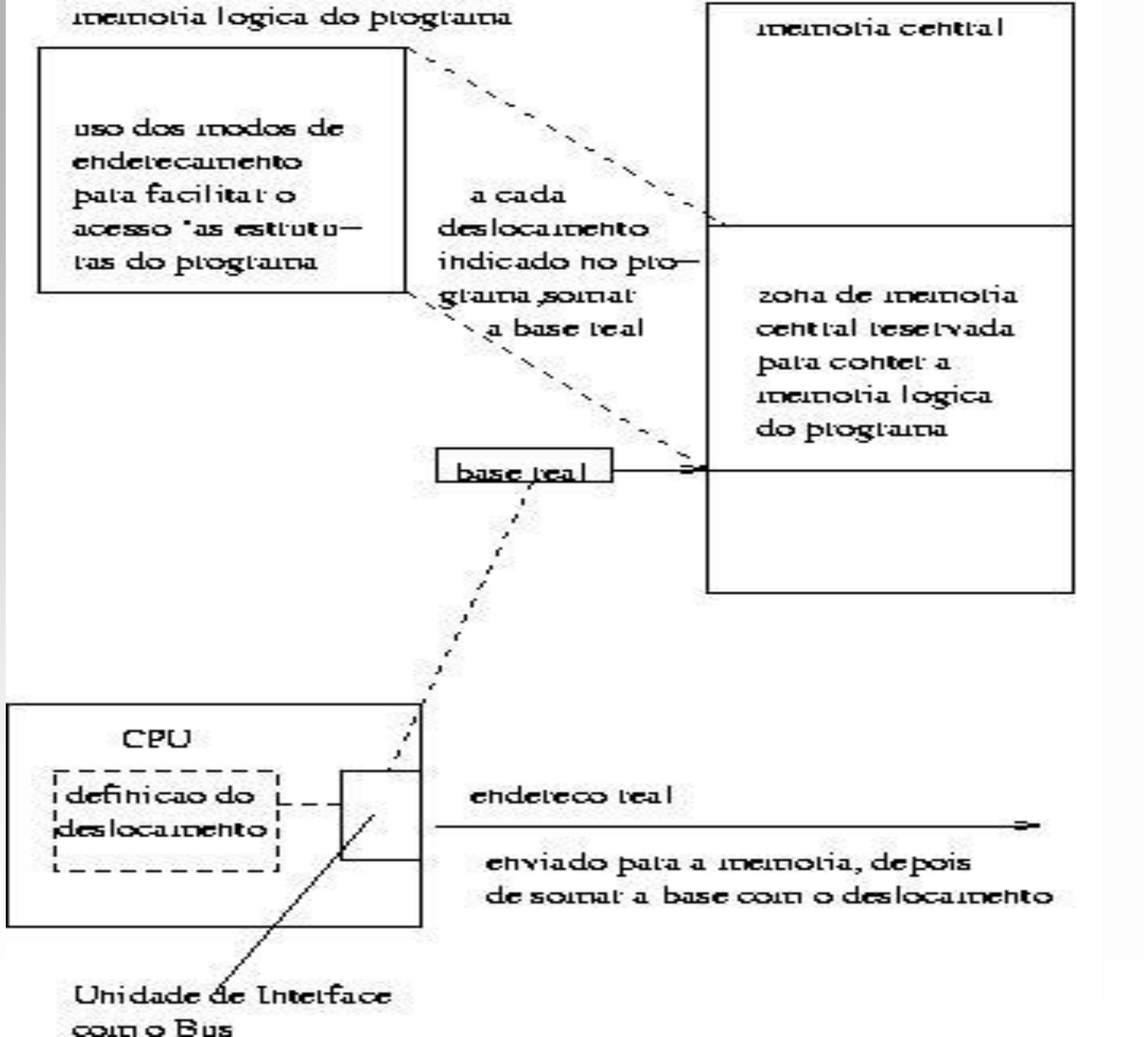
Algoritmo baseado em somas sucessivas:

```
c := m; R := 0;
Enquanto c <> 0 fazer
        c := c – 1;
        R := R + M
fimfazer;
```


endereços reais em binário
representação simbólica das instruções
pseudo-código correspondente
0 0000
0 0001
LOAD E2
STORE E6
c := m
0 0010
0 0011
LOAD E4
STORE E3
R := 0
0 0100 CYCLE:
0 0101
LOAD E6
Zero END
teste de c = 0 ?
0 0110
0 0111
0 1000
LOAD E3
ADD E1
STORE E3
R := R + M
0 1001
0 1010
0 1011
LOAD E6
SUB E5
STORE E6
c := c - 1
0 1100
Jump CYCLE
0 1101 END:
HALT
0 1110 E1
0 1111 E2
1 0000 E3
1 0001 E4
1 0010 E5
1 0011 E6
M
m
R
0
1
c
Representação simbólica dos endereços através de etiquetas

Modos de endereçamento de memória


program ...
var a, b: integer
.....
0
1
2
3
1o byte de a
2o byte de a
1o byte de b
2o byte de b
1000
1001
1002
1003
Espaco de nomes do programa Pascal (En)
Espaco de endereços virtuais (Ev)
Espaco de endereços reais (Er)
En
(1)
Ev
(2)
Er


memoria logica do programa
memoria central
uso dos modos de endereçamento para facilitar o acesso as estruturas do programa
a cada deslocamento indicado no programa somar a base real
zona de memoria central reservada para conter a memoria logica do programa
base real
CPU
definicao do deslocamento
endereco real
enviado para a memoria, depois de somar a base com o deslocamento
Unidade de Interface com o Bus

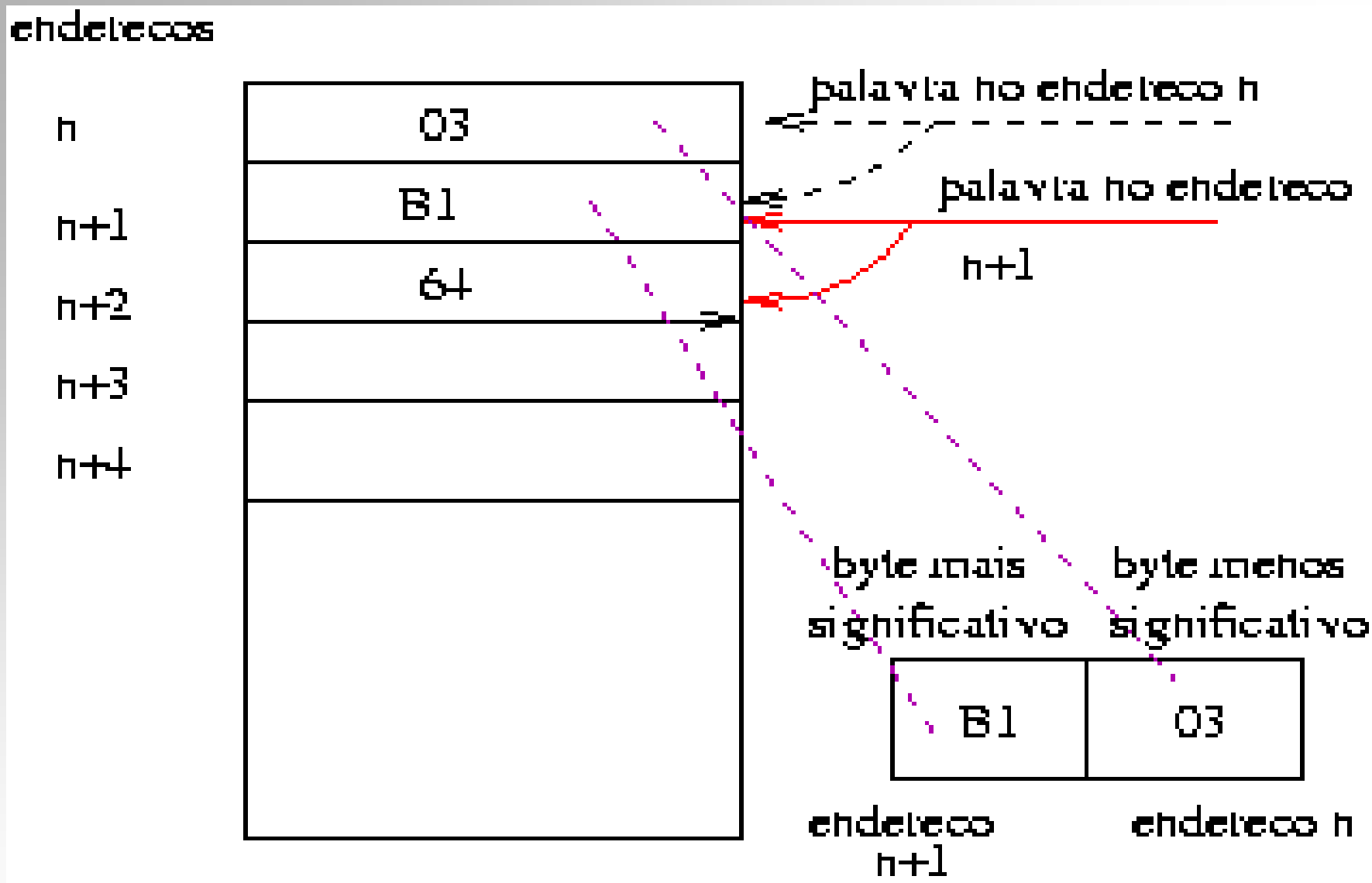
Tamanho da palavra

- Externa: definida pelo Bus externo ao CPU (bus de sistema): o número máximo de bytes transferíveis em cada transacção entre CPU e Memória → número de linhas de dados
- Espaço de Endereços Reais: alcance máximo de posições de memória central, permitido pelo Bus → número de linhas de endereços
- Interna ao CPU: definida pelo tamanho dos buses internos ao CPU e pelo máximo tamanho de operandos manipulados pela ALU

Para o mesmo computador, estes tamanhos podem ser diferentes.

Organização da Memória Central

- Na generalidade dos computadores, o tamanho da célula básica endereçável pelo CPU é o byte: compromisso entre uma representação compacta dos dados e um acesso eficiente.
- A Memória é ´´vista´´ como um vector de bytes
- O CPU envia ´´endereços de bytes´´ para a Memória



Organizacao little-endian: o byte menos significativo fica no endereco inferior de memoria e o byte mais significativo fica no endereco superior (n+1)

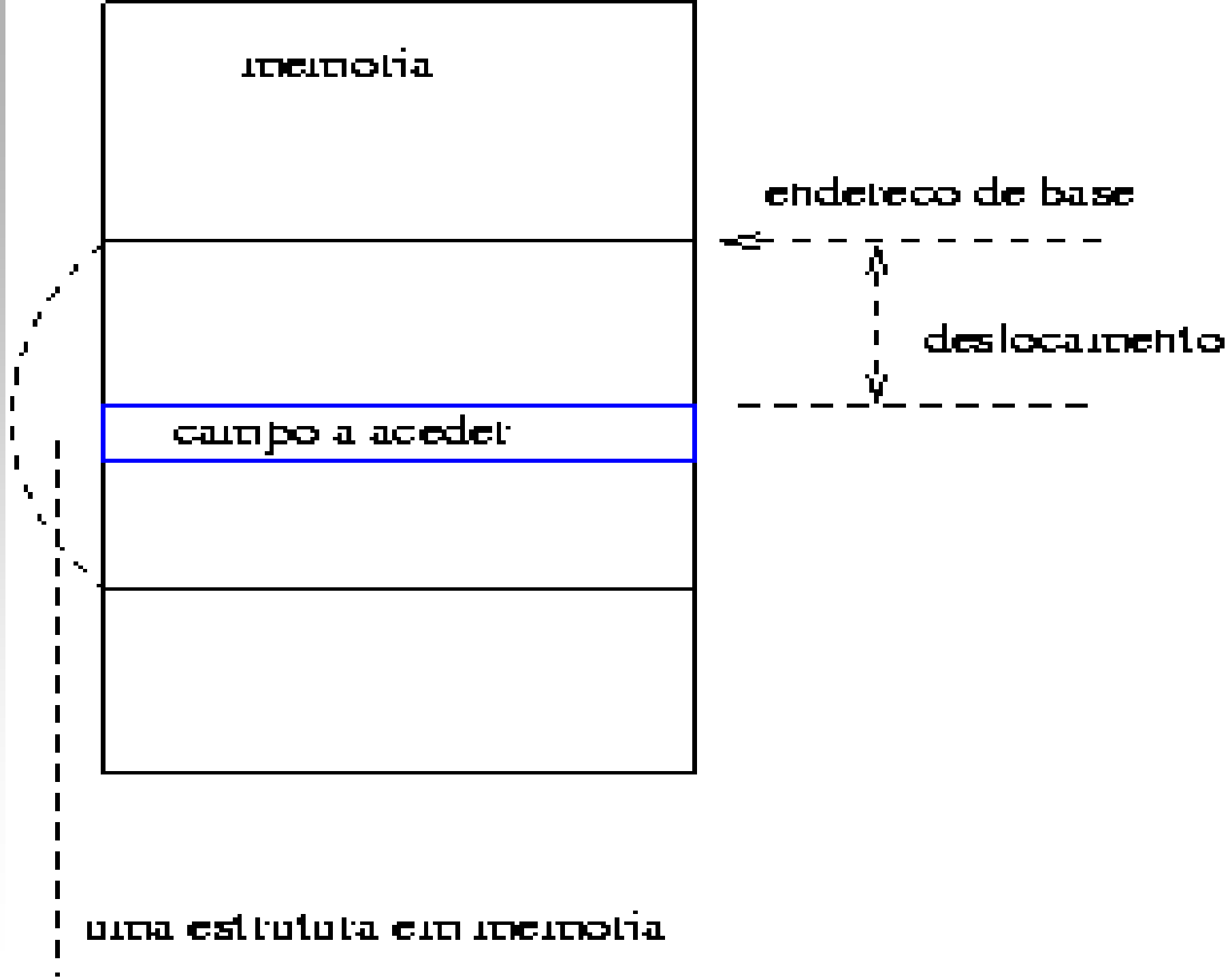
Modos de endereçamento da memória

- Tipos de dados: o computador dá suporte a um certo tipo de dados se manipular operandos desse tipo com eficiência…
  - Instruções máquina: sobre dados com representação adequada
  - Modos de endereçamento: para obter acesso eficiente aos operandos em memória central

- Cadeias de caracteres: vectores de bytes
  - Operações: comparações, cópias, pesquisa
  - Endereçadas por:
    - endereço do primeiro ou último elemento
    - tamanho de cada elemento
  - Mantidas em memória e não em registadores doCPU
    - mas registadores do CPU guardam informação para o acesso: endereço + contador de bytes
- Arrays: modos especiais de endereçamento facilitam o acesso a elementos
  - Dimensões
  - Número de bytes por elemento
  - Arranjo consecutivo em memória (por linhas, por colunas)

- Records: modos de endereçamento para facilitar o acesso a campos individuais
  - Calcular o endereço inicial de cada campo do record
  - Seleccionar campos individuais

- De um modo geral: facilitar o acesso às estruturas de dados armazenadas em células consecutivas de memória


memória
endereço de base
deslocamento
campo a aceder
uma estrutura em memória

Modos de endereçamento

Permitem que as instruções especifiquem o acesso a células de memória, podendo os seus endereços serem calculados durante a execução das instruções

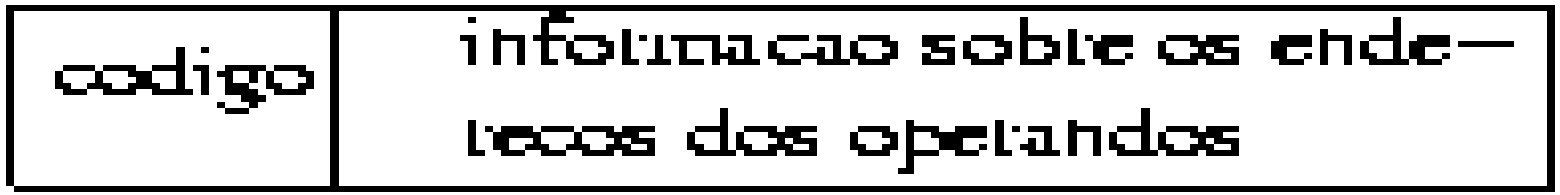
código I modo I endereço I

Para:

1. Acesso eficiente a estruturas de dados: arrays, records, listas, pilhas, etc.
2. Reduzir o número de bits do campo ´´endereço´´
3. Calcular endereços relativos à posição da instrução corrente (PC) ou a outra posição

(para obter programas recolocáveis)

Instrucao maquina



conforme o modo de endereçamento

produz o endereço efectivo

envia-o para a unidade de interface com o bus externo (Memory Address Register)

Endereçamento imediato

código I valor-do-operando

O operando faz parte da própria instrução, sendo trazido para o CPU, quando este faz FETCH da instrução (para o IR (*Instruction Register*))

Exemplo: mov R1, 5 -- R1:=5

Serve para especificar constantes:

para fazer inicializações ou como operando em operações aritméticas e lógicas

Não é absolutamente necessário, mas:

pode reduzir o tamanho do programa e aumentar a rapidez da sua execução

Endereçamento por registador

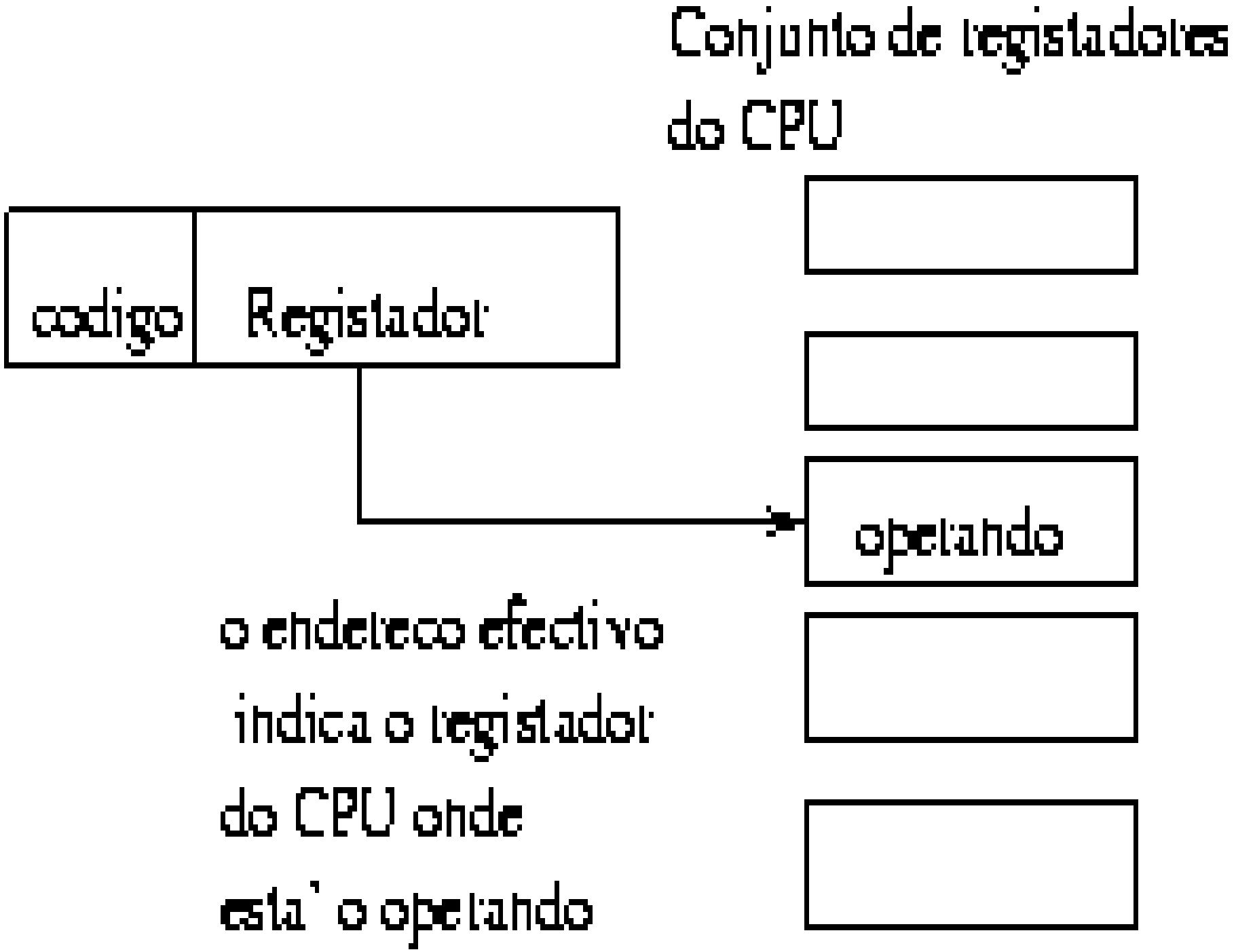
código I registador-do-CPU

O operando está contido num dos registadores do CPU

Exemplo: mov R1, R2 -- R1:= R2

Pressupõe que uma instrução anterior carrega o registador com o valor desejado

- Acesso muito rápido ao operando
- Limitado número de registadores do CPU


Conjunto de registadores
do CPU
codigo
Registador
operando
o endereco efectivo
indica o registador
do CPU onde
esta' o operando

Endereçamento directo

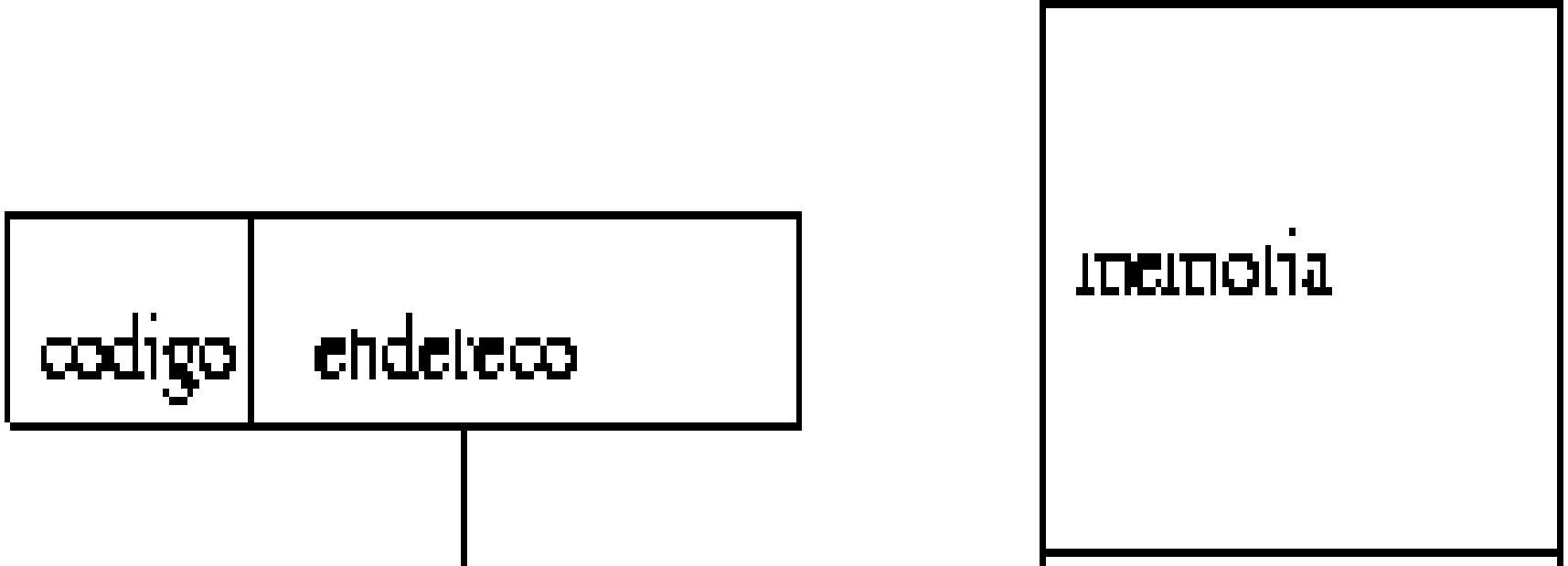
código I endereço-de-memória

O endereço efectivo está contido na instrução.

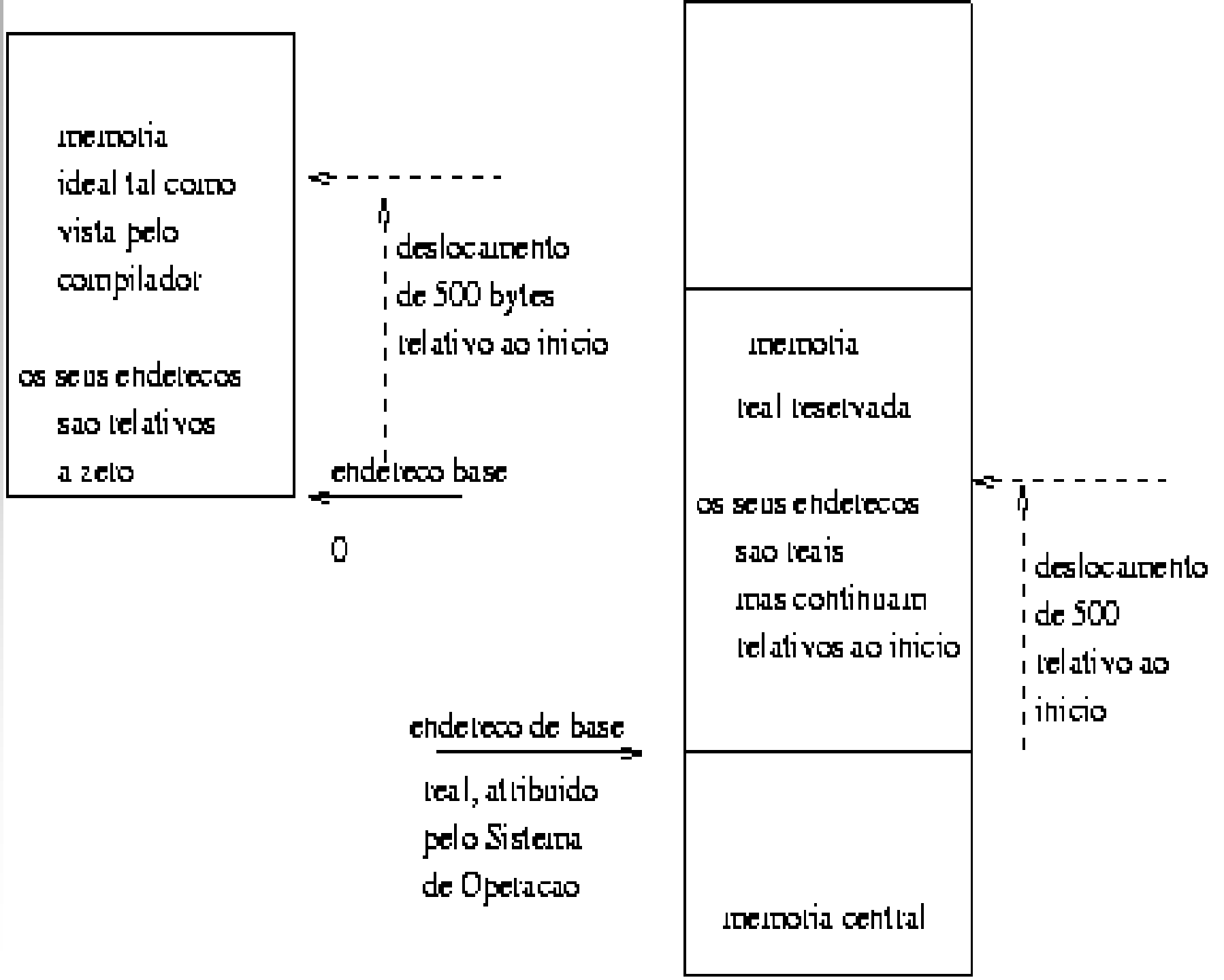
Exemplo: mov R1, [100] -- R1:= Mem[100]

Se o endereço indica a posição real de memória onde está o operando:

- É obrigatório conhecer essa posição quando se escreve o programa:
  - Nem sempre se sabe
  - Obriga o programa a ser carregado em zonas pré-definidas de memória
- O campo de ´´endereço´´ deve ter o número de bits suficiente para alcançar toda a memória real



o endereco efectivo contido na propria instrucao e indica onde esta' o operando em memoria


memoria
ideal tal como
vista pelo
compilador
os seus endereços
sao relativos
a zero
deslocamento
de 500 bytes
relativo ao inicio
endereco base
0
endereco de base
real, atribuido
pelo Sistema
de Operacao
memoria real reservada
os seus endereços
sao reais
mas continuam
relativos ao inicio
deslocamento
de 500
relativo ao
inicio
memoria central

Endereçamento relativo (ao PC)

código I deslocamento (´´offset´´)

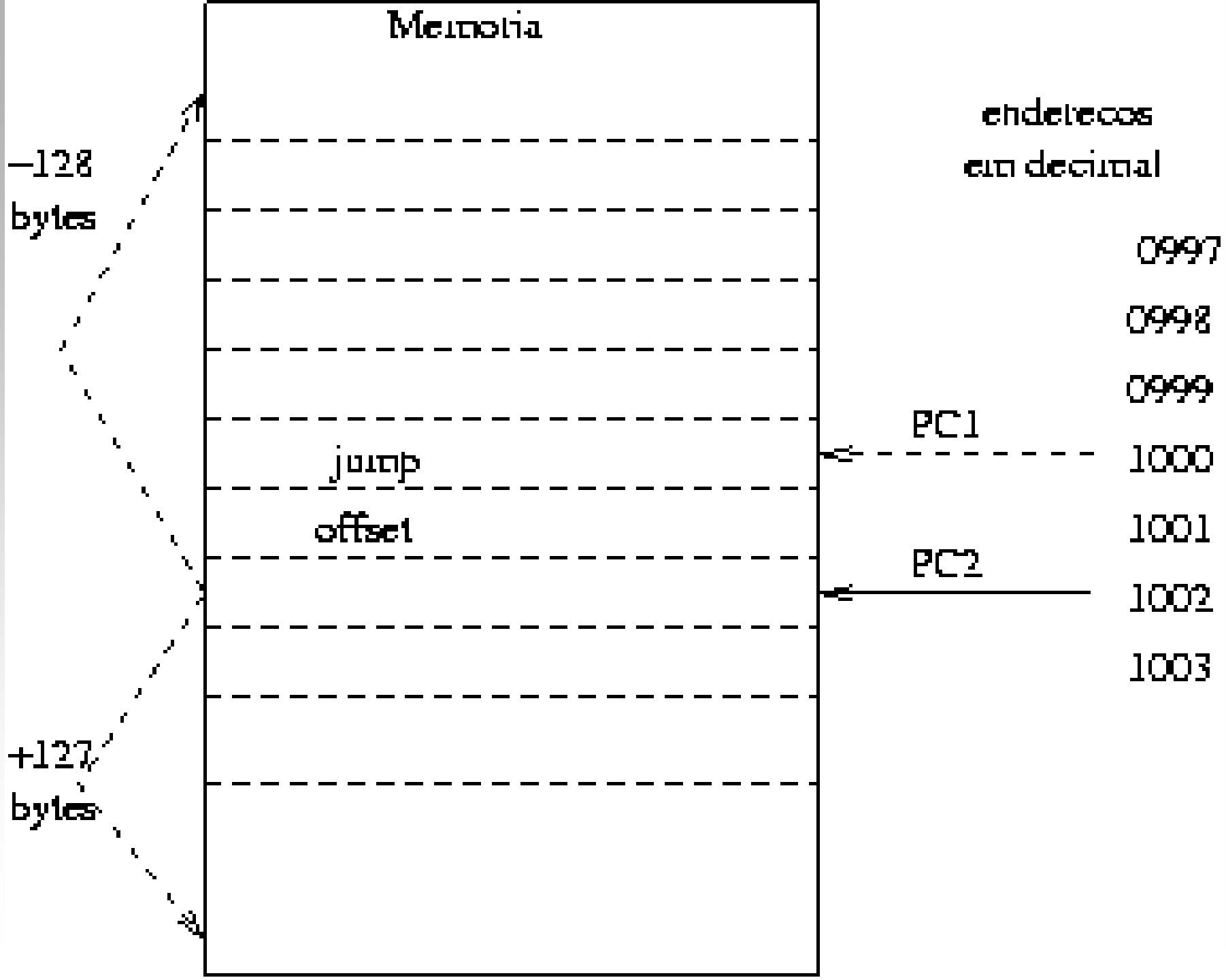
Um deslocamento, indicado na instrução, é somado ao valor corrente do Program Counter, obtendo-se o endereço efectivo.

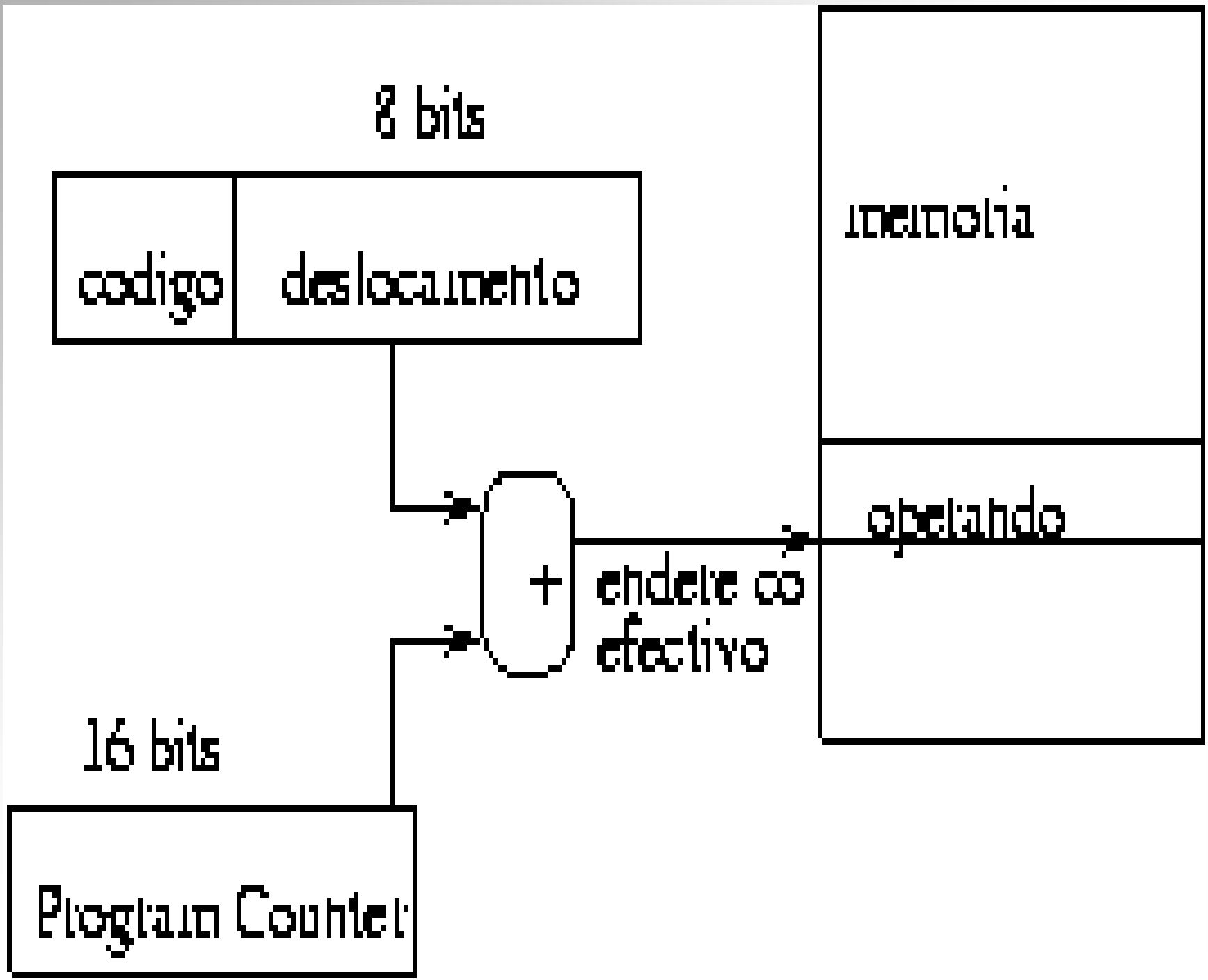
Exemplo: jump [PC+34] -- PC:= PC+34

Exemplo: mov R1, [PC-30] -- R1:= Mem[PC-30]

NOTA: o valor do PC é o valor corrente (após incrementado, já apontando a instrução seguinte)

Obtêm-se programas recolocáveis, que são independentes das posições de memória onde são carregados.


Memoria
endereços
em decimal
−128
bytes
0997
0998
0999
PC1
1000
jump
offset
1001
PC2
1002
1003
+127
bytes


8 bits
codigo
deslocamento
memoria
operando
+
endereco efectivo
16 bits
Program Counter

Supondo: PC – registador de 16 bits

Deslocamento – um número de 8 bits, inteiro com sinal.

Endereços

(base 10)

127 jump [PC+4] -- ocupa 2 bytes

129 .....

PC = 0000 0000 1000 0001 = 129(base 10)

0000 0000 0000 0100 = 4(base 10)

0000 0000 1000 0101 = 133(base 10)

Extensão do bit de sinal

Supondo: PC – registador de 16 bits

Deslocamento – um número de 8 bits, inteiro com sinal.

Endereços

(base 10)

127 jump [PC-4] -- ocupa 2 bytes

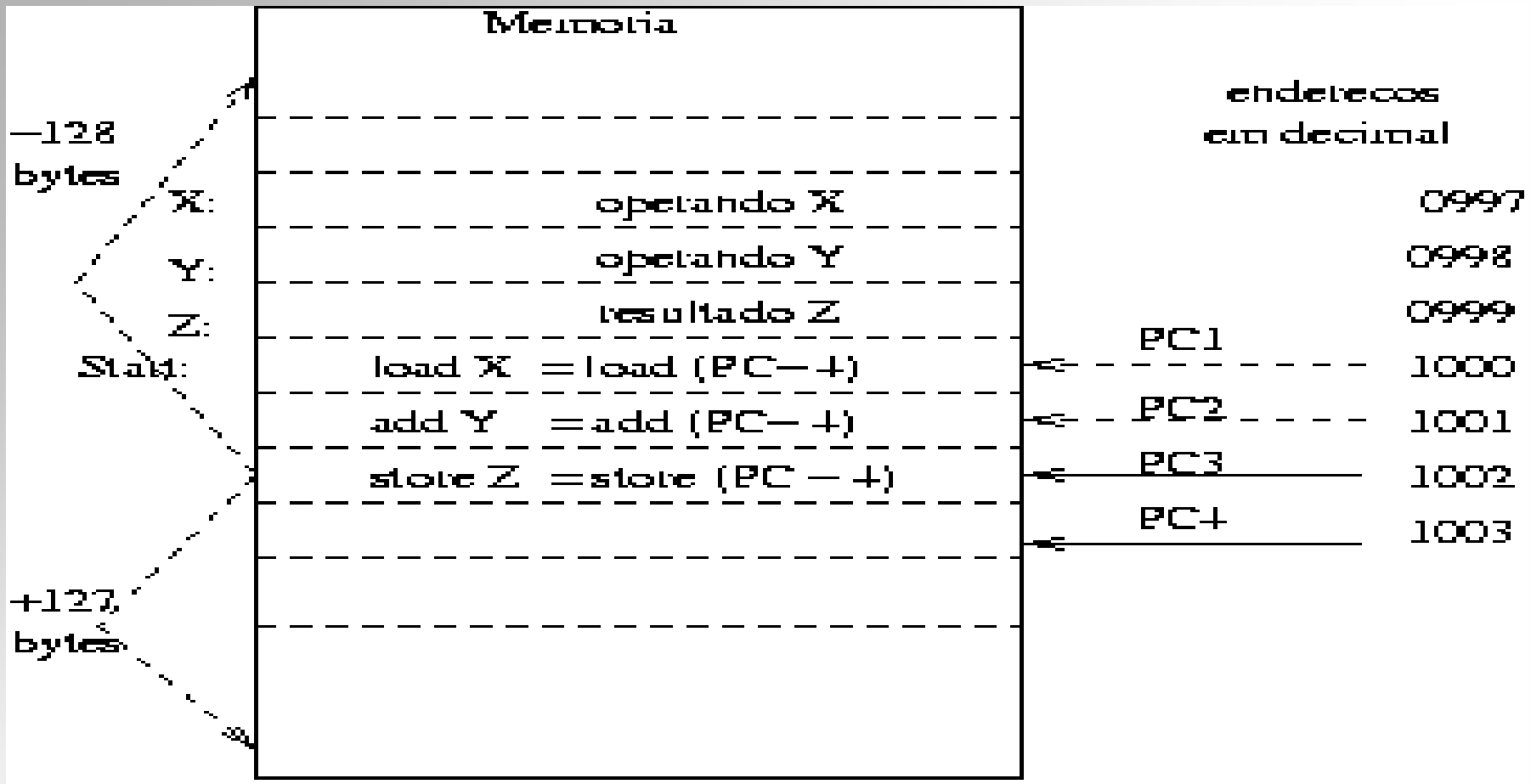
129 .....

PC = 0000 0000 1000 0001 = 129(base 10)

1111 1111 1111 1100 = -4(base 10)

0000 0000 0111 1101 = 125(base 10)

Extensão do bit de sinal



Admita que cada instrucao ocupa 1 byte.
Admita que se tem um unico acumulador no CPU
e que a instrucao `load X equivale a AC := Mem[X]
e a instrucao add Y equivale a AC:= AC+Mem[Y]
e store Z equivale a Mem[Z] := AC

Com enderecos relativos, o deslocamento X vale +,
bem como o deslocamento Y e o deslocamento Z.

Endereçamento baseado

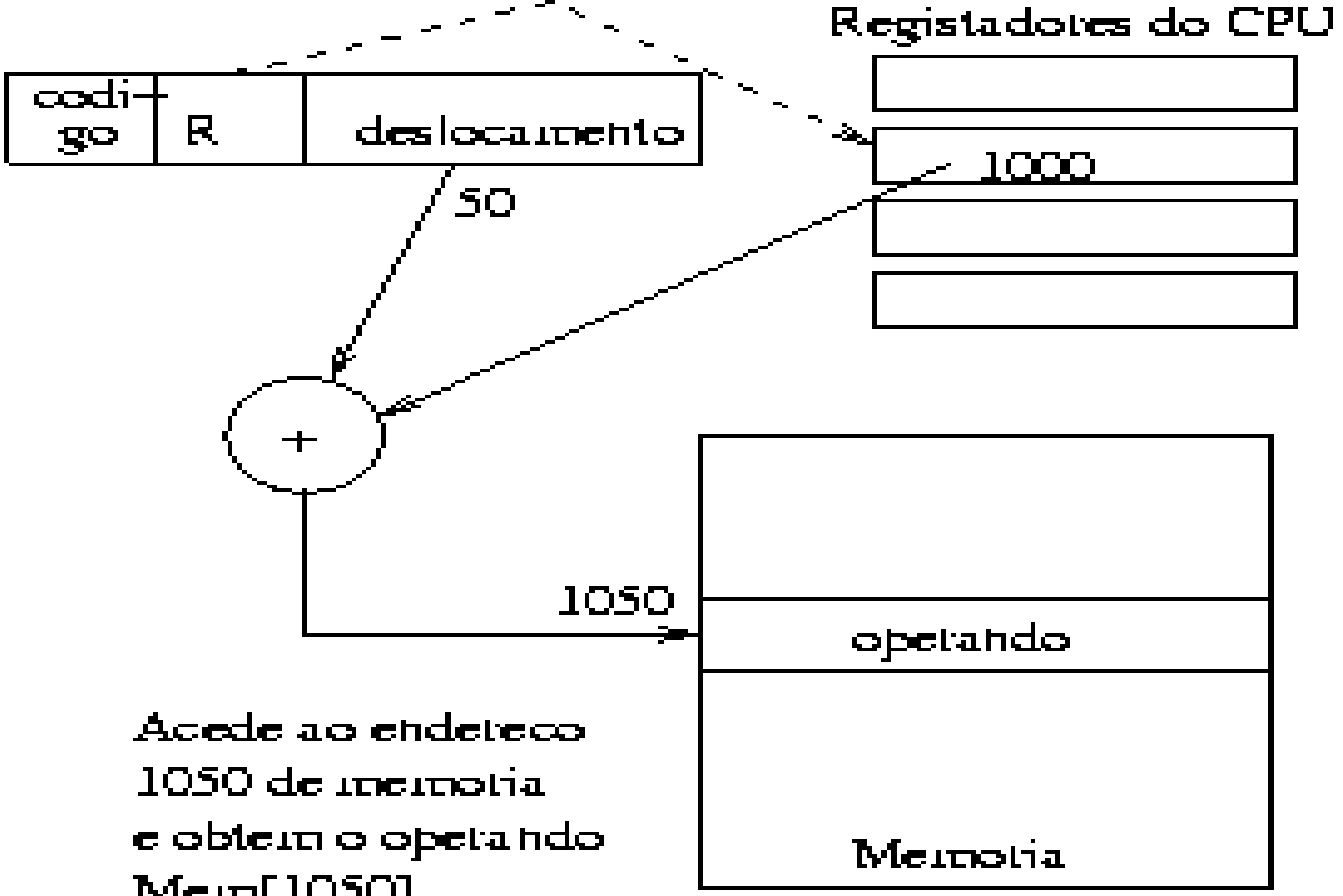
código I registador I deslocamento

Um deslocamento, indicado na instrução, é somado ao valor corrente do ´´registador de base´´ indicado, obtendo-se o endereço efectivo.

Exemplo: mov R1, [R_b+50] -- R1:= Mem[R_b+50]

O ´´registador´´ é usado como um ´´apontador´´ para a ´´base´´ de uma zona de memória, relativamente à qual se localizam os operandos através do ´´deslocamento´´.

Pressupõe que Rb foi previamente carregado.



Acede ao endereco
1050 de memoria
e obtem o operando
Mem[1050]

O registador R contem o valor 1000
O deslocamento tem o valor 50

Pode ser utilizado para:

- Aceder a dados numa zona de parâmetros cujo endereço de base é passado a uma subrotina
- Aceder a elementos de uma pilha
- Aceder a dados de modo independente da posição de memória em que estão carregados
- Aceder a campos particulares de estruturas de dados (records, elementos de listas)

Endereçamento indexado

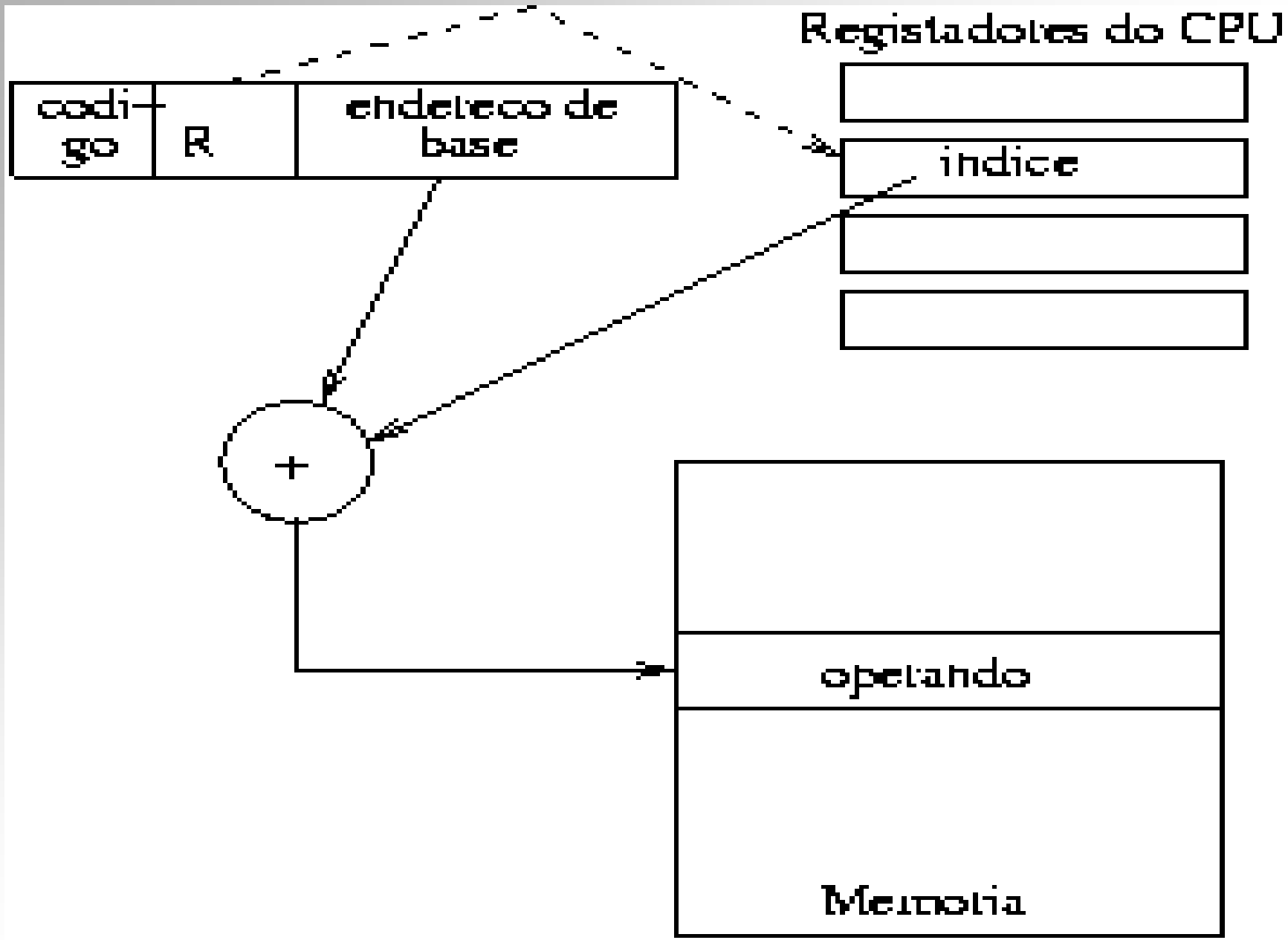
código I registador I ender.base

Um endereço de base, indicado na instrução, é somado ao valor corrente do ´´registador de índice´´ indicado, obtendo-se o endereço efectivo.

Exemplo: mov R1, [1000 + Ri] -- R1:= Mem[1000+Ri]

O ´´registador´´ é usado como um ´´índice´´ de uma zona de memória, cuja base foi especificada na instrução através do´´endereço base´´.

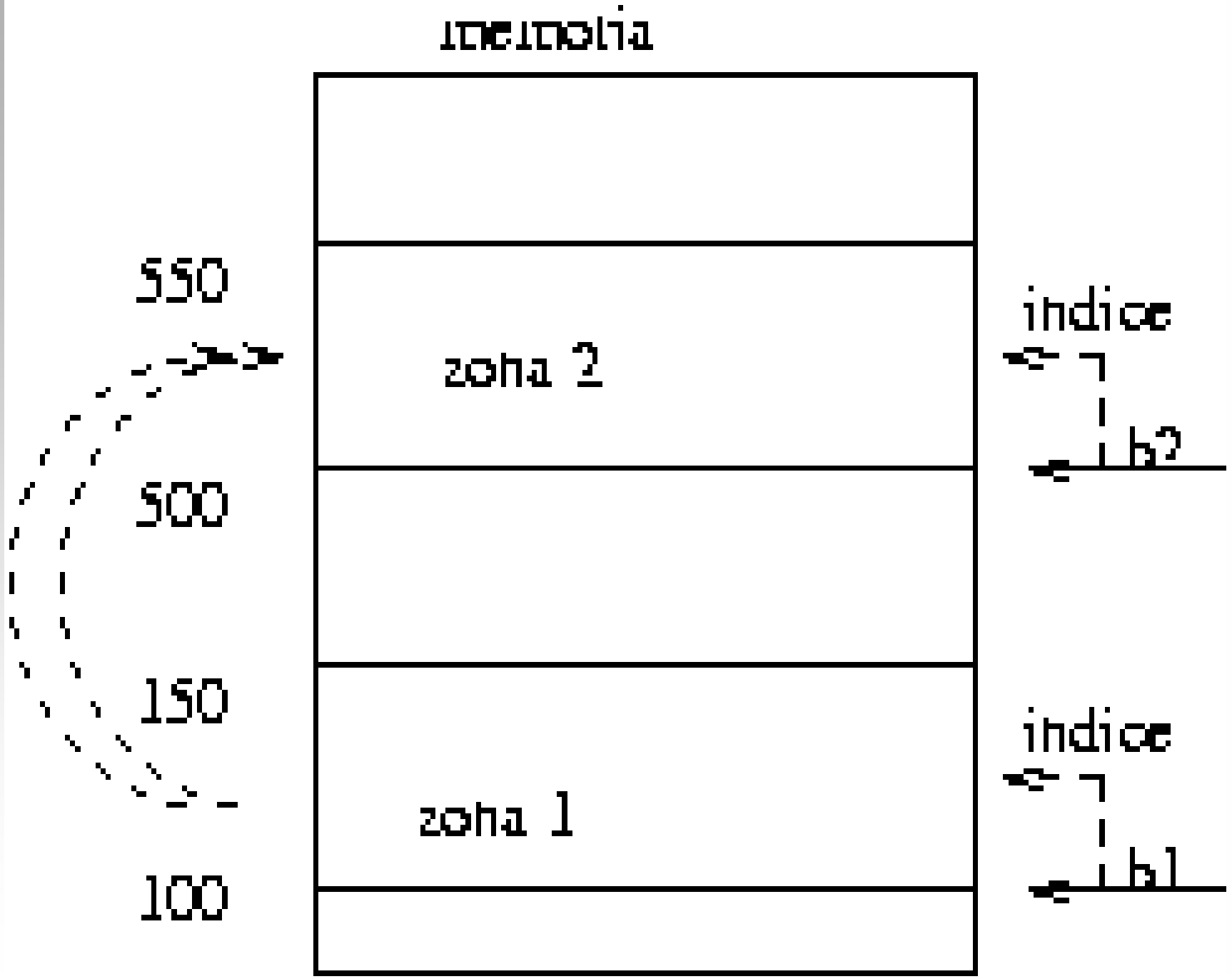
Pressupõe que Ri foi previamente inicializado e vai ser utilizado para aceder a sucessivas posições da zona memória (incrementado/ decrementado)



O registador R contem o valor do indice

O endereco de base vem na instrucao

Serve para aceder a ´´array´´e tabelas: o endereço de base (fixo na instrução) dá a base da tabela e o ´´registador de índice´´ dá o índice o elemento corrente.


memoria
550
500
150
100
zona 2
zona 1
indice
b2
indice
b1

Exemplo:

```
   mov R1, 0          -- R1:=0
   mov R2, 0          -- R2:=0
REPEAT
   mov [500+R1], [100+R2]     -- Mem[500+R1]:=Mem[100+R2]
   inc R1                     -- R1:=R1+1
   inc R2                     -- R2:=R2+1
UNTIL Fim
```

Usa R1 e R2 como registadores de índice (index register)

Endereçamento baseado e indexado

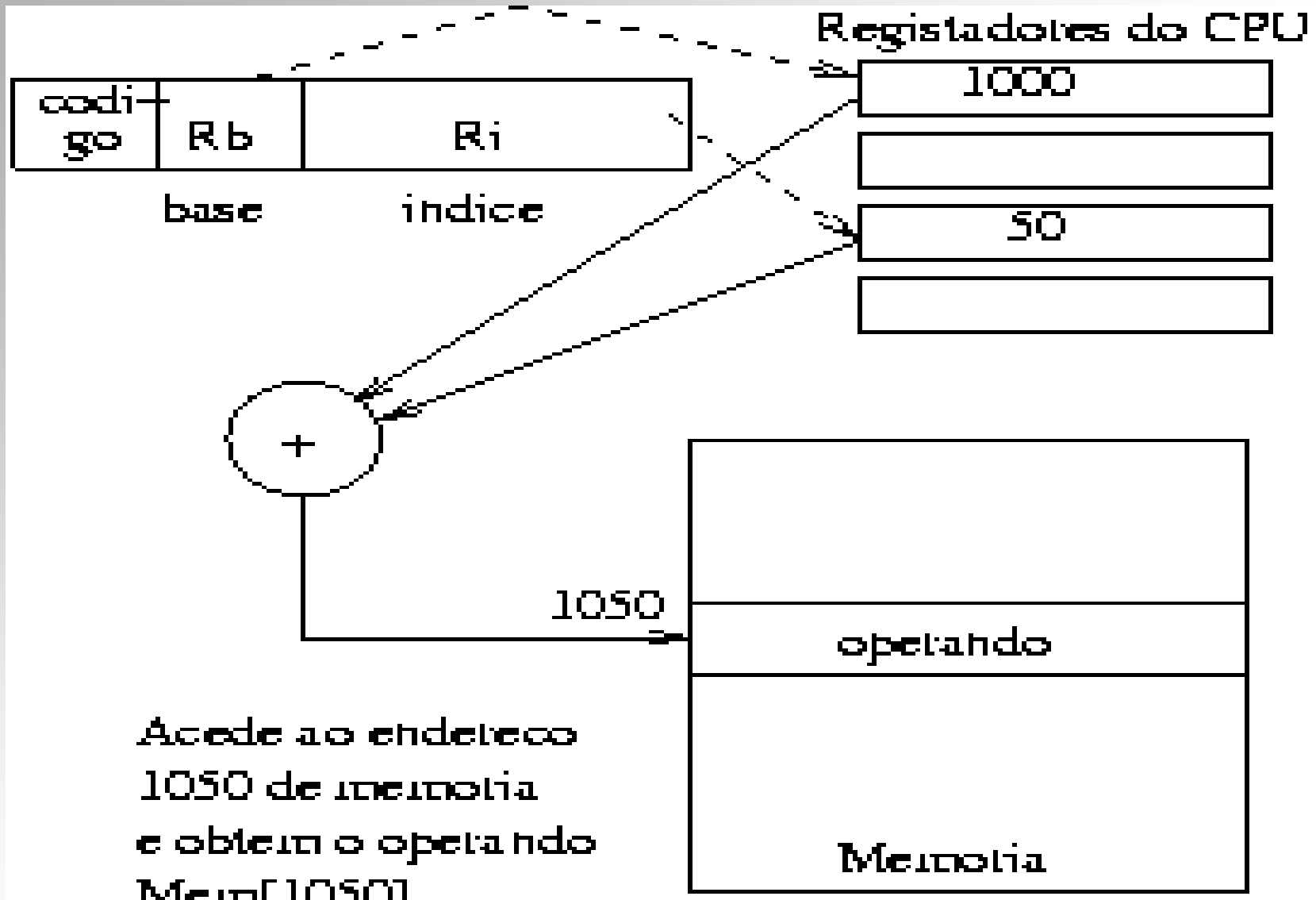
código I reg.base I reg.índice

O valor corrente do registador de base, indicado na instrução, é somado ao valor corrente do ´´registador de índice´´ indicado, obtendo-se o endereço efectivo.

Exemplo: mov R1, [Rb+Ri] -- R1:= Mem[Rb+Ri]

O programa carrega Rb com o endereço de base de uma zona de memória, e acede a células individuais daquela zona, indicando os seus índices em Ri.

Rb e Ri podem ser definidos durante a execução do programa.



Acede ao endereco
1050 de memoria
e obtem o operando
Mem[1050]

O registador Rb contem o valor 1000
O registador Ri contem o valor 50

Endereçamento indirecto

código I @ endereço

O endereço indicado na instrução é usado para localizar uma posição de memória que contém o endereço efectivo.

Exemplo: mov R1, @ End -- R1:= Mem[Mem[End]]

A posição indicada pelo ´´endereço´´ é utilizada como ´´apontador´´ para localizar o operando.

Exemplo: estruturas de listas de elementos ligados, em memória.

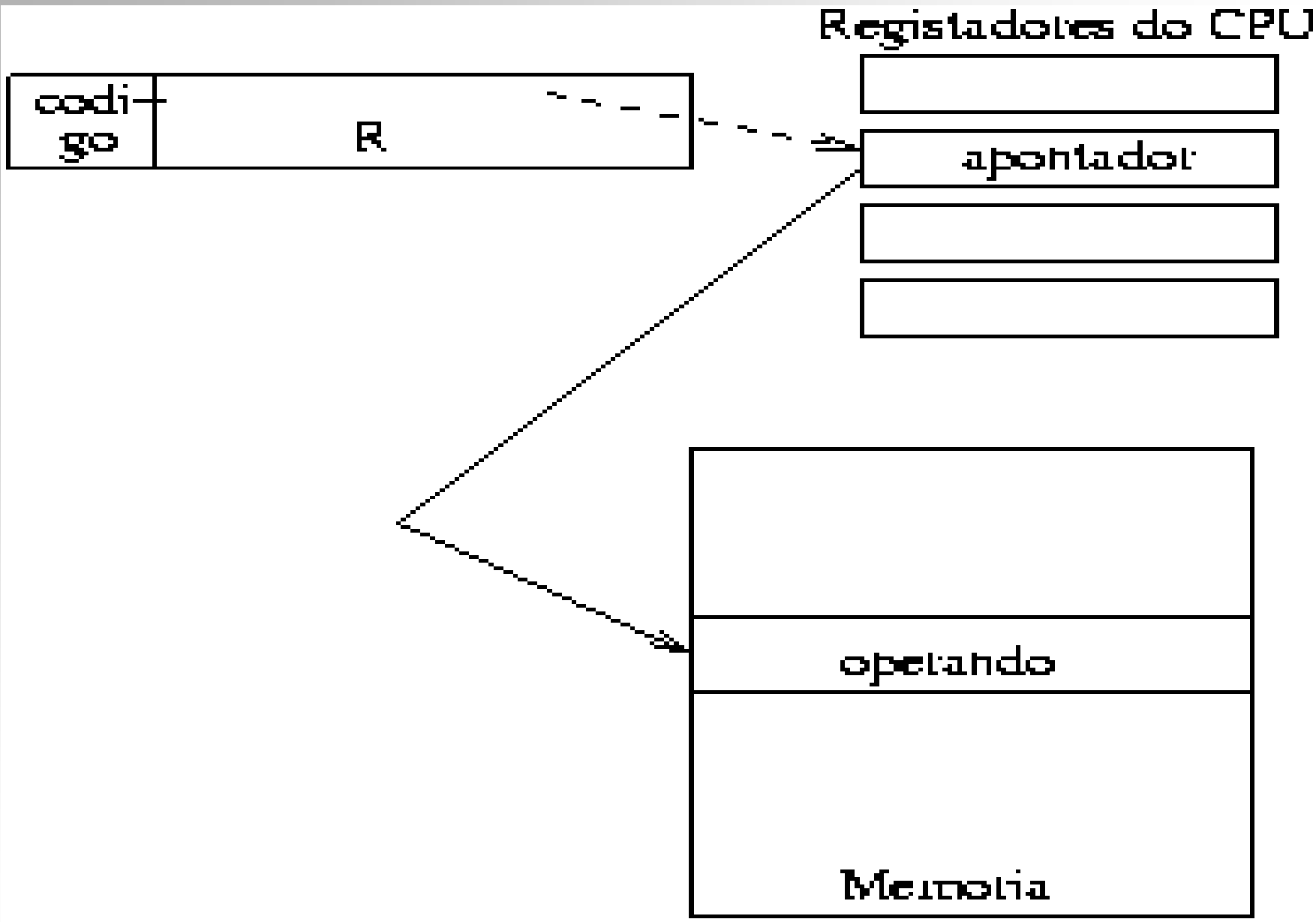
Endereçamento indirecto, por registador

código I @ registador

O registador indicado na instrução contém um valor que é interpretado como o endereço efectivo de memória.

Exemplo: mov R1, [Rx] -- R1:= Mem[Rx]

O ´´registador´´ é utilizado como ´´apontador´´ para localizar o operando



O registador R contem o endereco do operando

Exemplo:

```
   mov R1, 500          -- R1:=500
   mov R2, 100          -- R2:=100
REPEAT
   mov [R1], [R2]       -- Mem[R1]:=Mem[R2]
   inc R1               -- R1:=R1+1
   inc R2               -- R2:=R2+1
UNTIL Fim
```

Usa R1 e R2 como apontadores.

Endereçamento indirecto, por registador, com funções de auto–incremento ou -decremento

código I @ **+/-**registador**+/-**.

mov R1, [Rx]**+** -- R1:= Mem[Rx]; **Rx:=Rx+1** -- pós-auto-inc

mov R1, [Rx]**-** -- R1:= Mem[Rx]; **Rx:=Rx-1** -- pós-auto-dec

mov R1, **+**[Rx] -- **Rx:=Rx+1**; R1:= Mem[Rx] -- pré-auto-inc

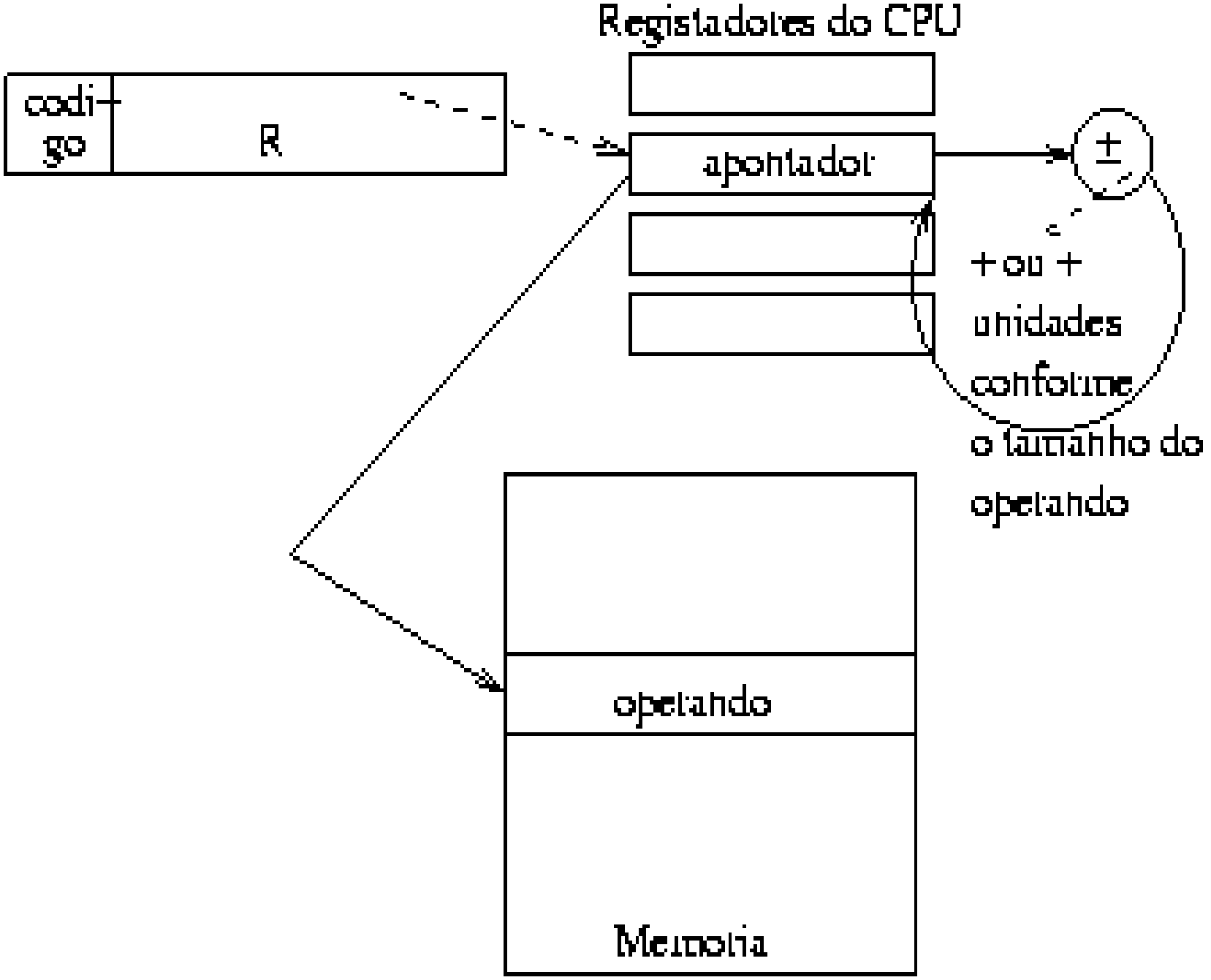
mov R1, **-**[Rx] -- **Rx:=Rx-1**; R1:= Mem[Rx] -- pré-auto-dec

O registador, antes/depois de usado como apontador, é automaticamente incrementado ou decrementado…

Exemplo:

```
   mov R1, 500          -- R1:=500
   mov R2, 100          -- R2:=100
REPEAT
   mov [R1]+, [R2]+
                        -- Mem[R1]:=Mem[R2]; R1:=R1+1; R2:=R2+1
UNTIL Fim
```

Usa R1 e R2 como apontadores e incrementa-os automaticamente, após utilizados.



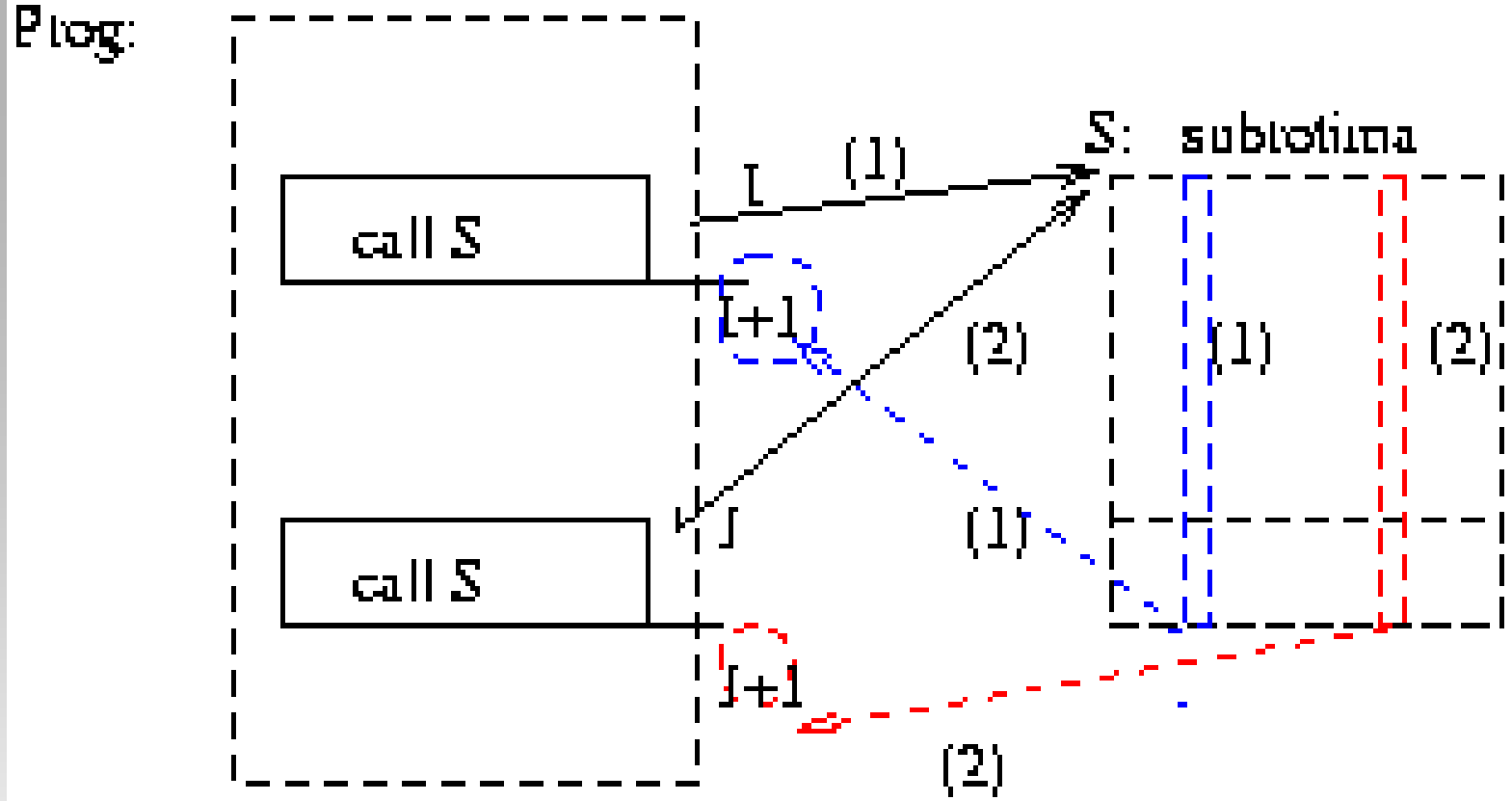
O registador R contem o endereco do operando

Instruções suportando subrotinas

Subrotinas

Subrotina: uma sequência de instruções máquina, que pode ser chamada múltiplas vezes, com diferentes argumentos, a partir de diferentes pontos de um programa

O suporte, a nível da arquitectura do computador, de uma funcionalidade sobre a qual assentam as funções e procedimentos de linguagens de ´´alto-nível´´.



(1) primeira chamada da subrotina S
a partir do ponto I do programa
com retorno ao ponto I+1

(2) segunda chamada da subrotina S
a partir do ponto J do programa
com retorno ao ponto J+1

(admite-se que as
instruções call S
ocupam 1 byte

Vantagens:

- Estruturação do programa:
  - Maior clareza e modularidade…
  - Desenvolvimento e teste incrementais…
- Redução do tamanho do programa:
  - Evita repetição de sequências muito utilizadas

A sua programação exige cuidados particulares:

- Na especificação dos parâmetros de entrada e de saída e como são passados à subrotina
  - Por valor (cópia) ou por referência (endereço)
- Que registadores do CPU são utilizados e se são ou não salvaguardados no início e repostos no fim
- Em que condições assume o estado da pilha de execução e em que estado a deixa, no retorno.

call S não é exactamente igual a jump S

jump S ------- PC := S

call S: não basta saltar; há que saber regressar

Se não existisse o ´´call s´´: ter-se-ia de executar:

No programa:

jump S --- para fazer a chamada

No fim da subrotina:

jump ´´I+1´´ – para regressar ao 1º ponto de chamada

jump ´´J+1´´ – para regressar ao 2º ponto de chamada

…

Após o ´´fetch´´ da instrução ´´call S´´, o Program Counter já foi incrementado e ´´aponta´´ o endereço de retorno: I+1, J+1, ….

call S:

1º -- salvaguarda o valor corrente do PC

2º -- PC:= S -- faz um salto absoluto

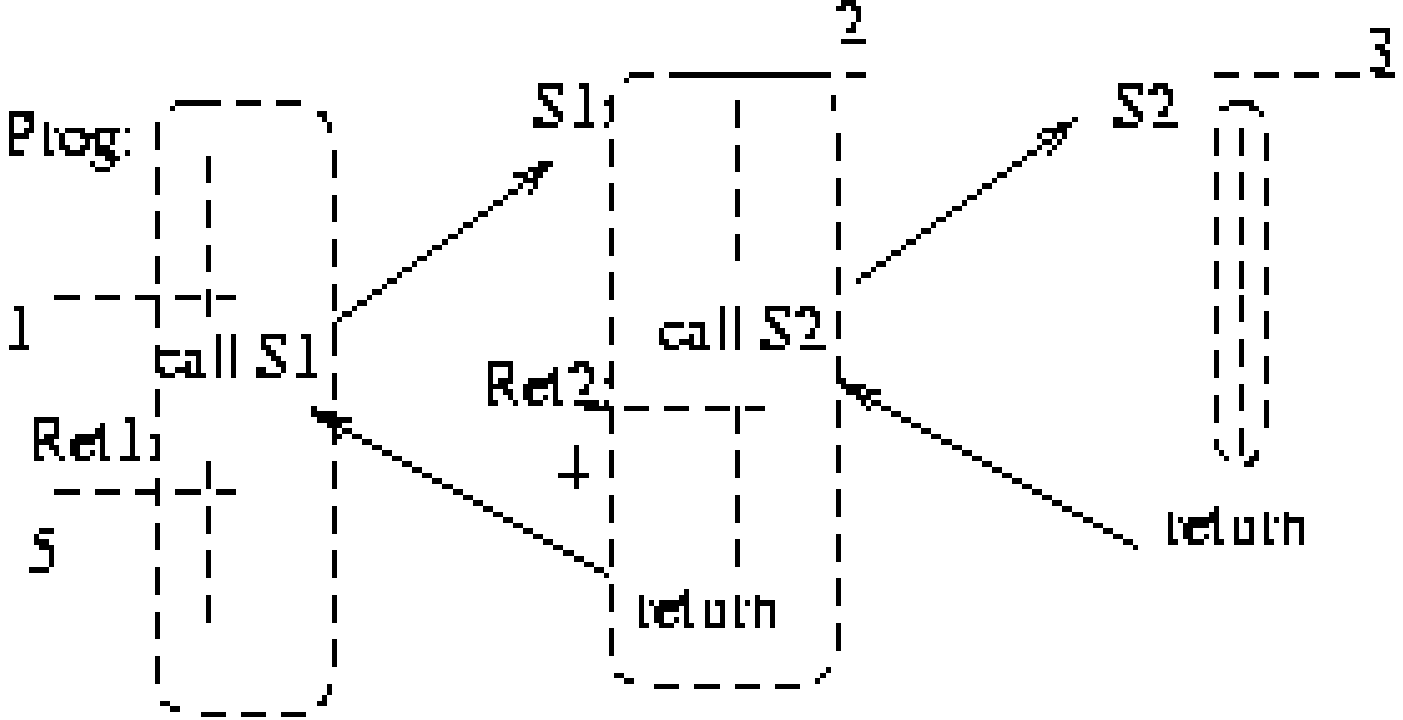
return:

-- repõe o valor original do PC, fazendo assim um salto absoluto para o ponto de retorno

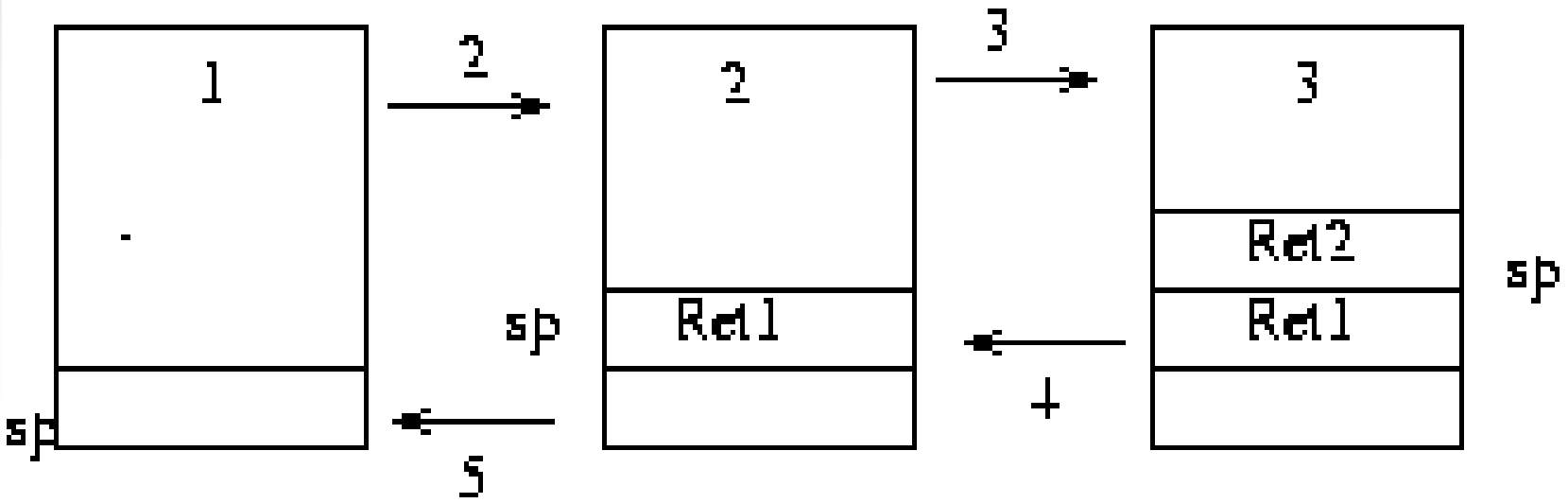
1ª micro-acção do ´´call s´´:

Salvaguarda do endereço de retorno:

- Numa célula pré-definida de memória?
- Num registador dedicado, do CPU?
- Numa zona de pilha em memória?



estados sucessivos da pilha em memoria



Salvaguarda dos registadores

No programa invocador?

- Antes de executar a instrução ´´call´´
- Exige que o programa saiba quais deve guardar, ou então teria de os guardar todos…

No início da subrotina?

- Salvaguarda só os que vai utilizar e repõe no fim

Onde?

Passagem de parâmetros

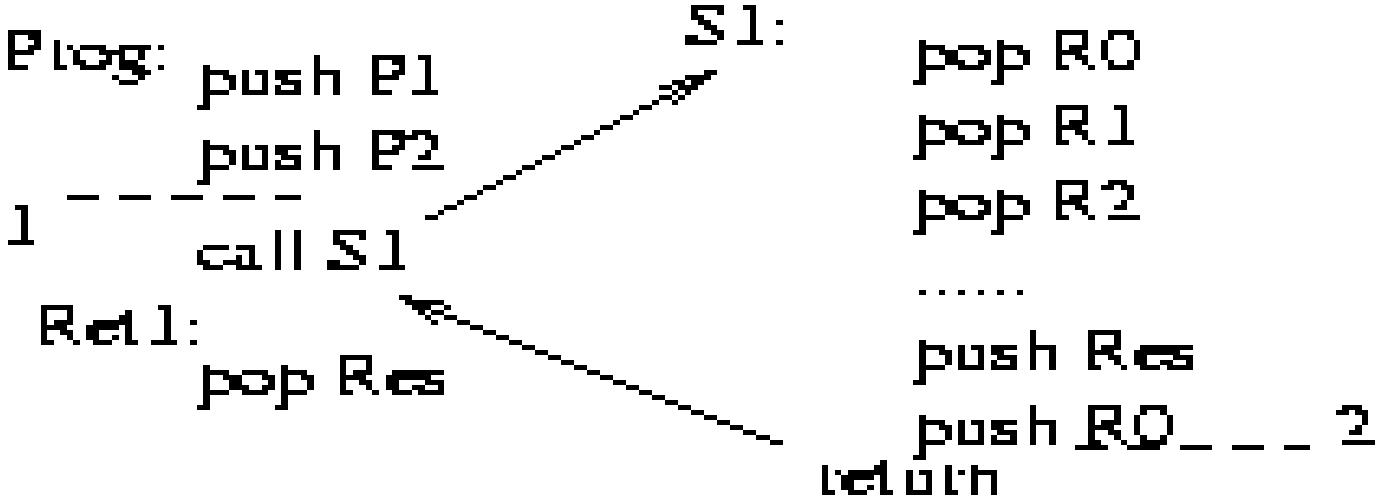
Modo:

- Por cópia ou valor
- Por referência ou endereço

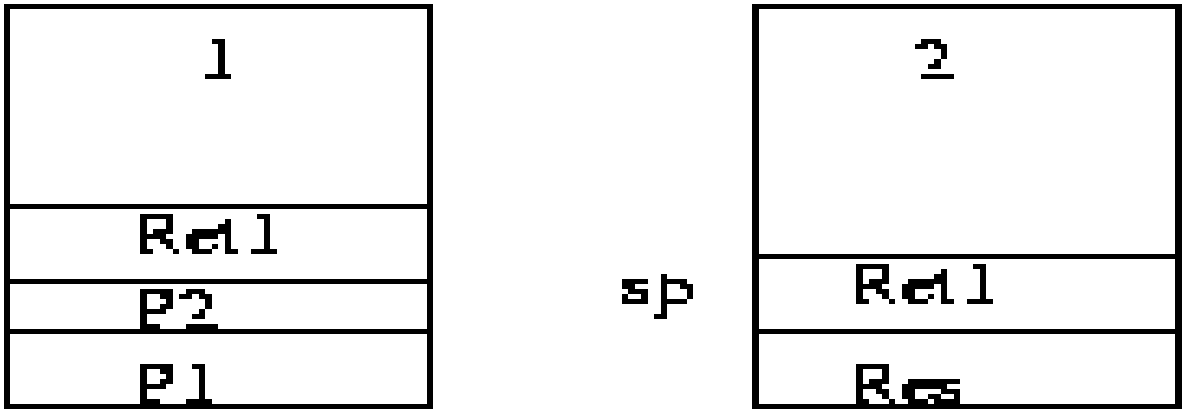
Onde:

- Em posições pré-definidas de memória
- Em registadores do CPU
- Através da pilha de execução

Não esquecer a especificação dos parâmetros de entrada e dos de saída.



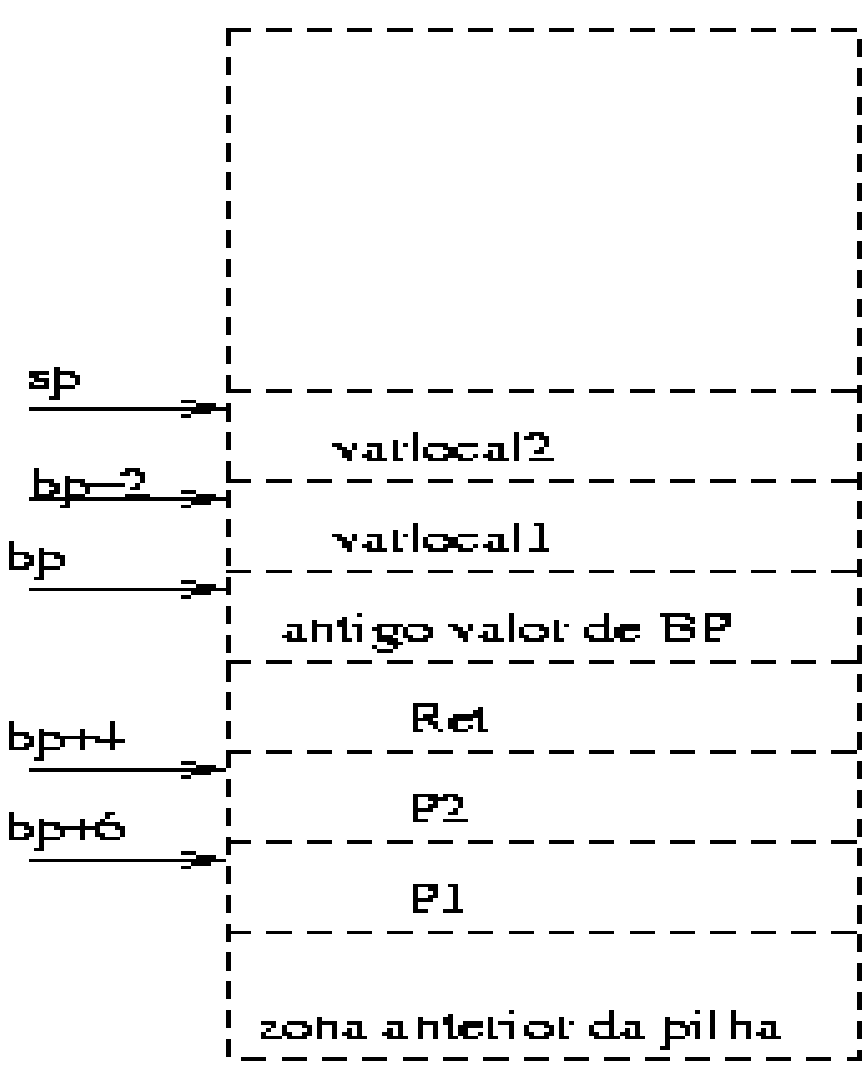
estados sucessivos da pilha em memoria



R0, R1, R2 — registadores do CPU

Res— o resultado da subrotina

P1, P2 —os parametros de entrada



Subrot:

```
push BP
mov bp sp
sub sp, 4
.....
mov bx, [bp+4]
mov si, [bp+6]
dec si
mov al , [bx+si]
.....
mov [bp-2], dx
....
mov sp,bp
pop bp
ret
```

B

C

Programa:

```
push P1
push P2
```

A

```
call Subrot
add  sp,4
```

D

.....