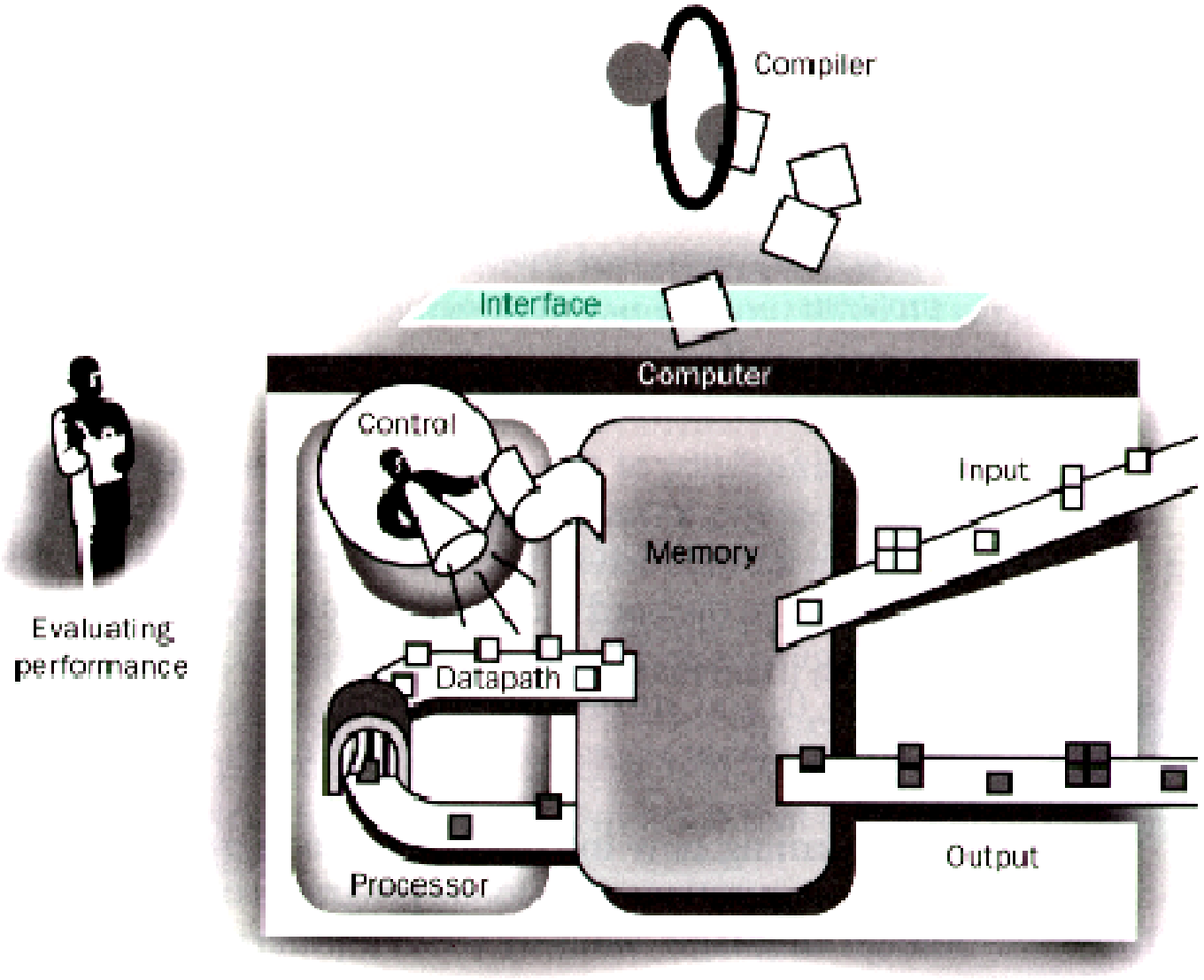
Conceitos Básicos de Arquitetura de Computadores

Capítulo 3

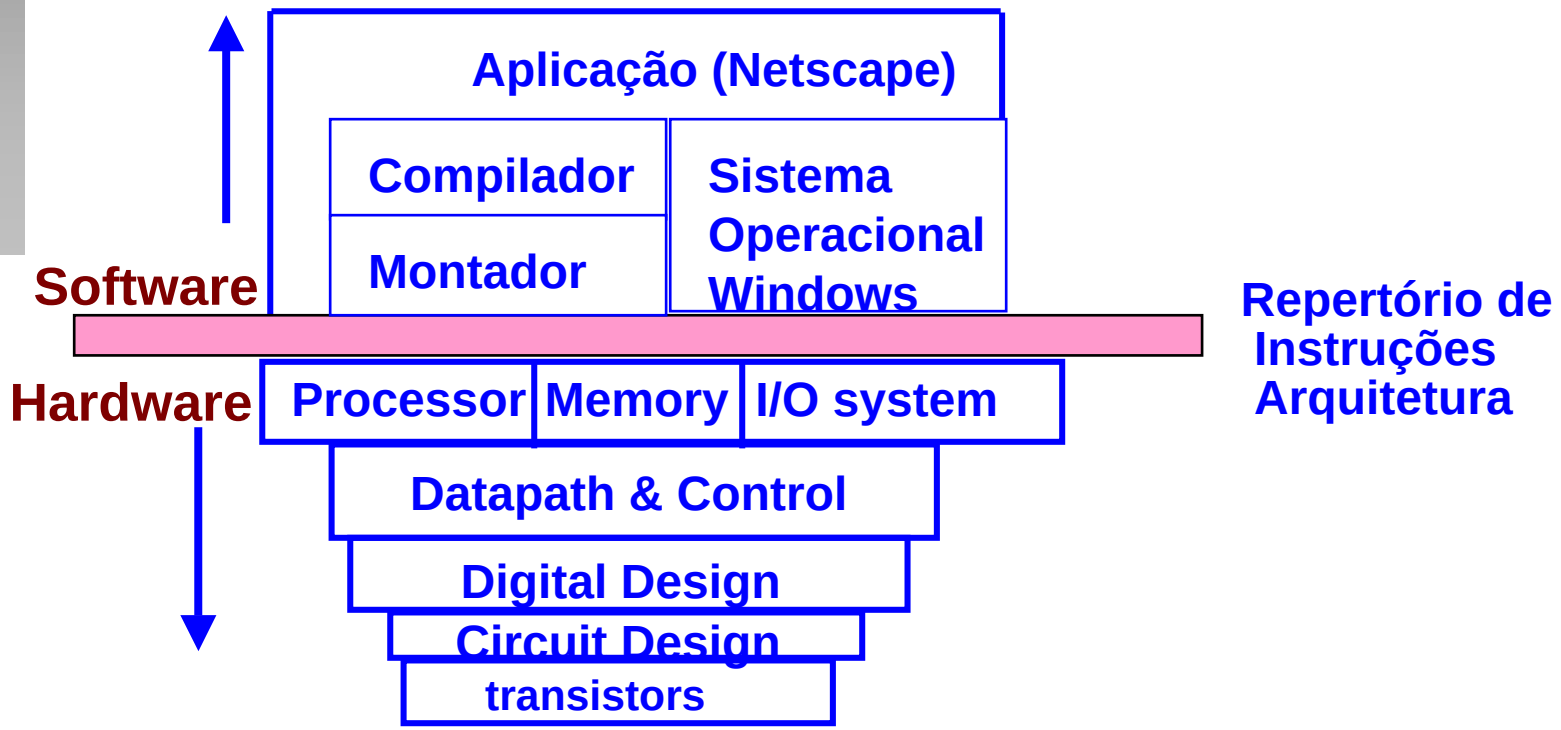
Cin - CENTRO DE INFORMÁTICA UFPE

GRECO

Computador: Hardware + Software

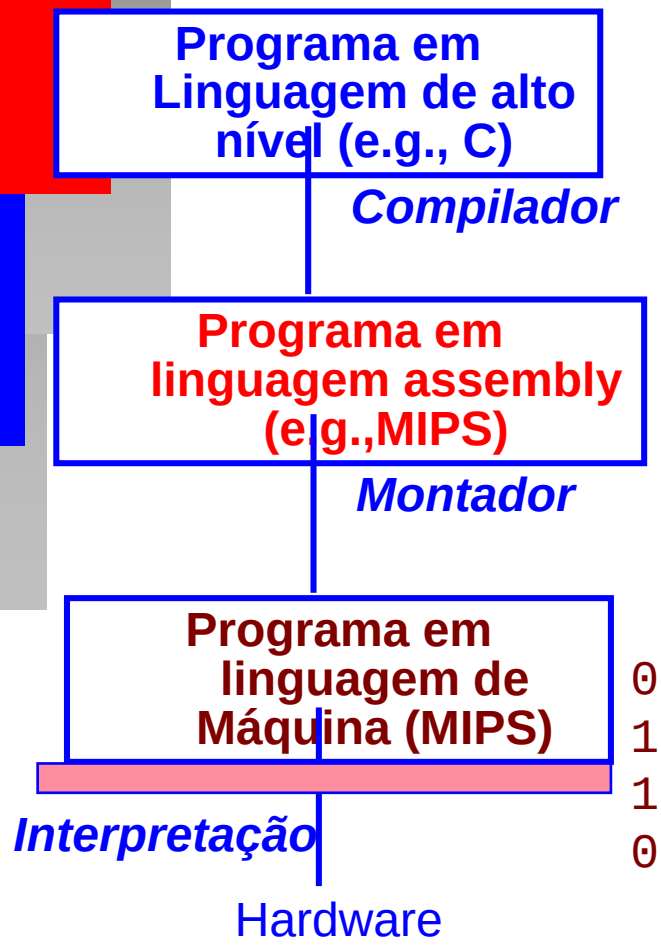


Computador: Hardware + Software



- Coordenação de muitos níveis de abstração

Representação da Informação




```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

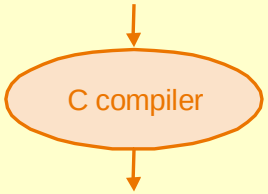
```
lw$to, 0($2)
lw$t1, 4($2)
sw     $t1, 0($2)
sw     $t0, 4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

Representação da Informação

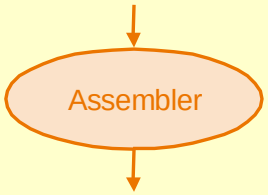
High-level language program (in C)

```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```



Assembly language program (for MIPS)

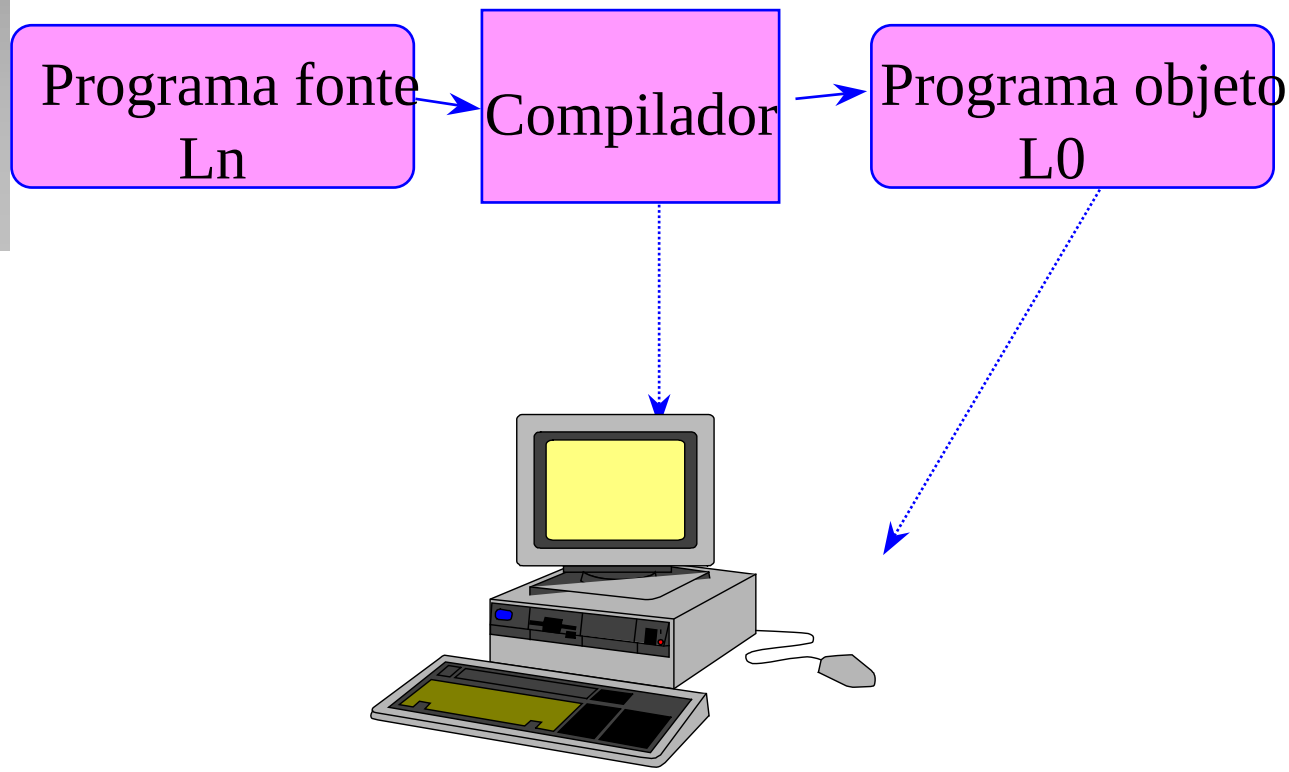
```
swap:
      muli $2, $5,4
      add  $2, $4,$2
      lw   $15, 0($2)
      lw   $16, 4($2)
      sw   $16, 0($2)
      sw   $15, 4($2)
      jr   $31
```



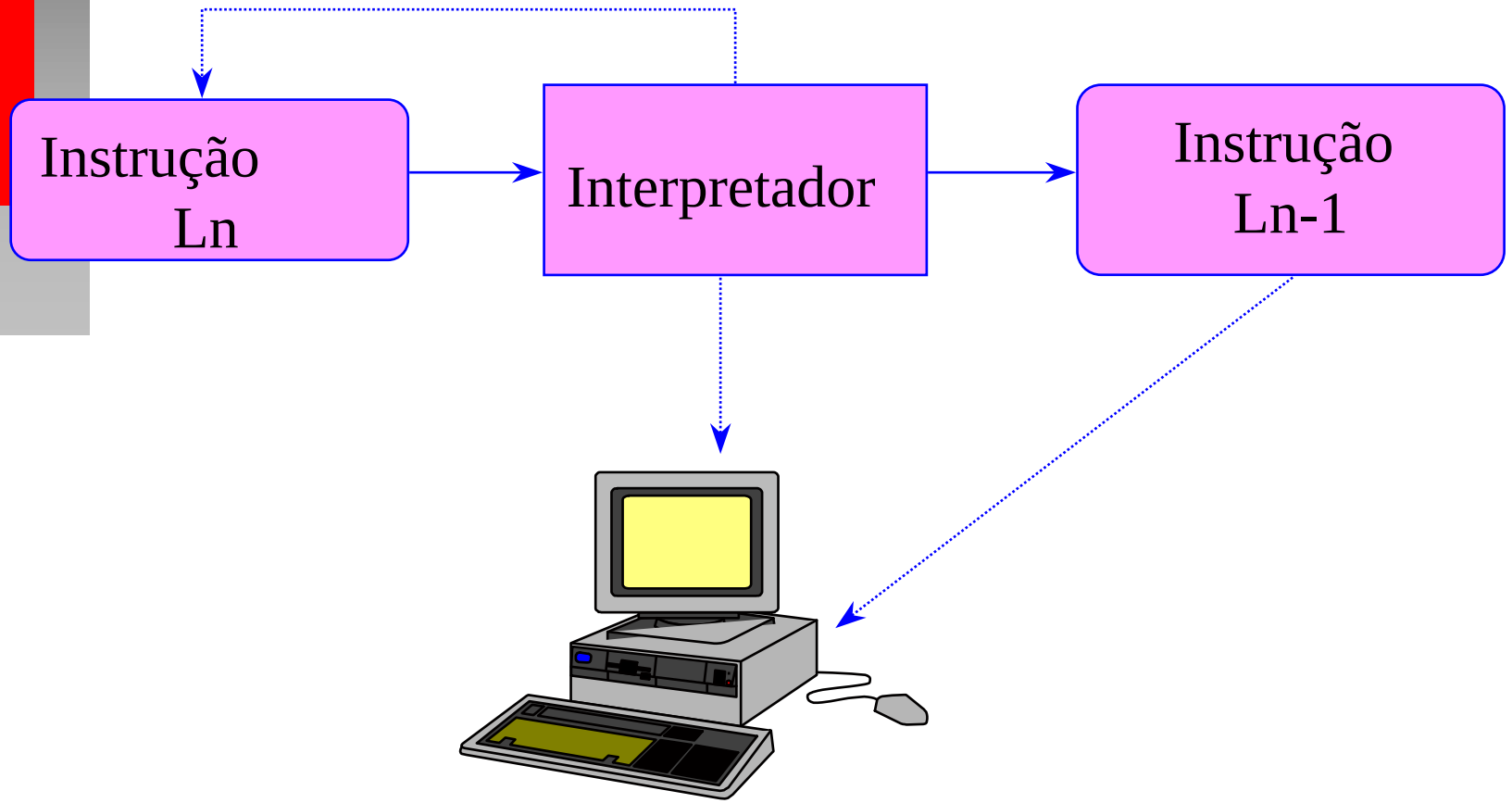
Binary machine language program (for MIPS)

```
00000000101000010000000000011000
00000000100011100001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

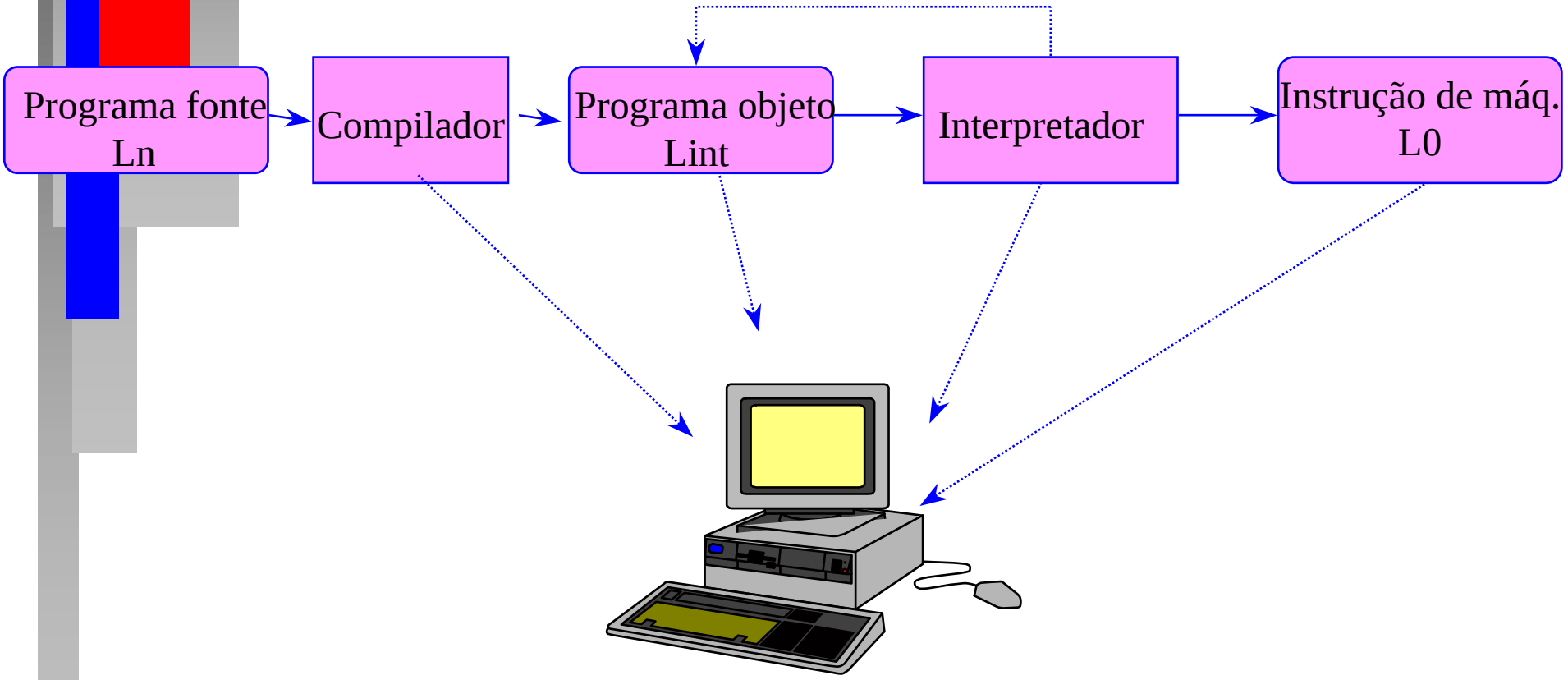
Compilação



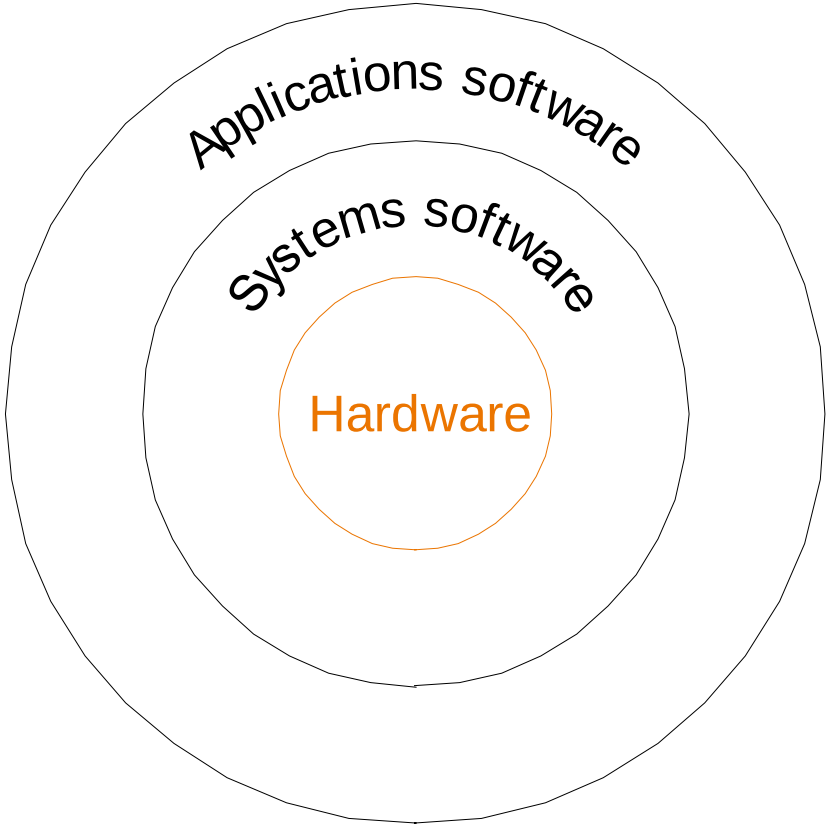
Interpretação



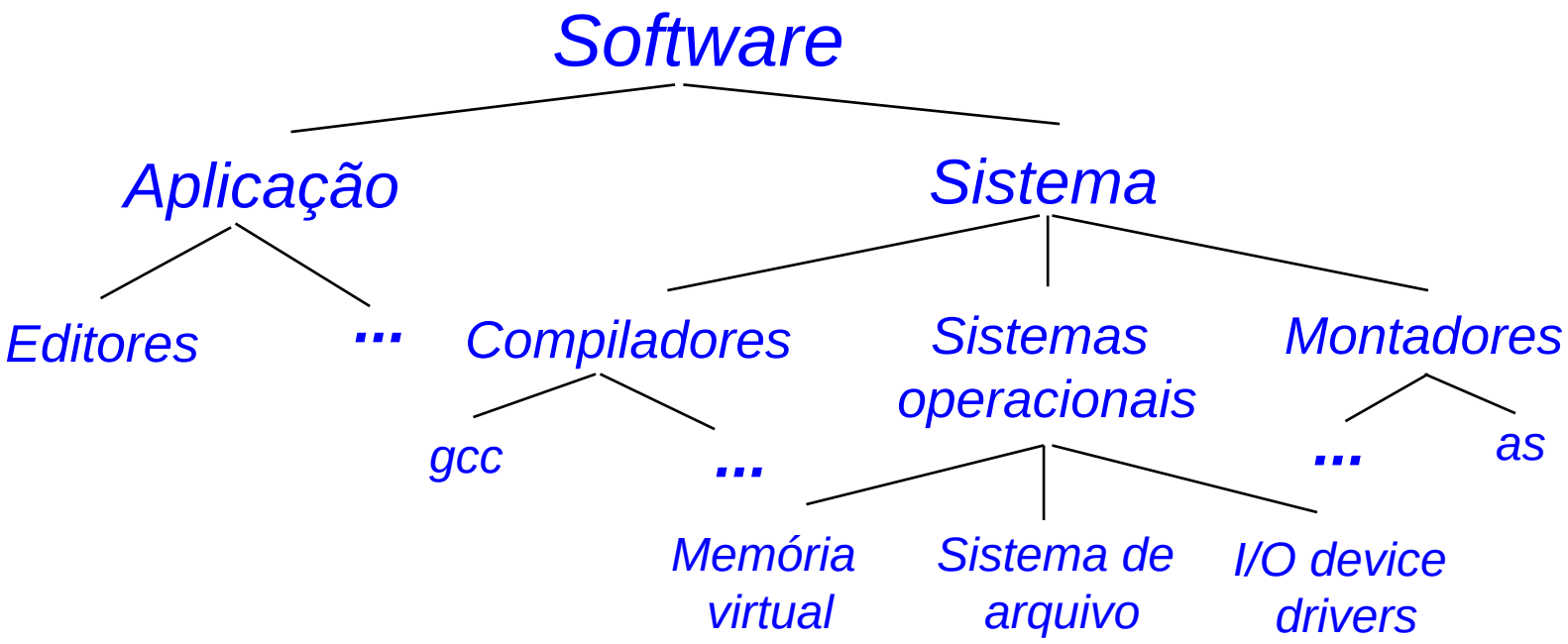
Interpretação & Compilação



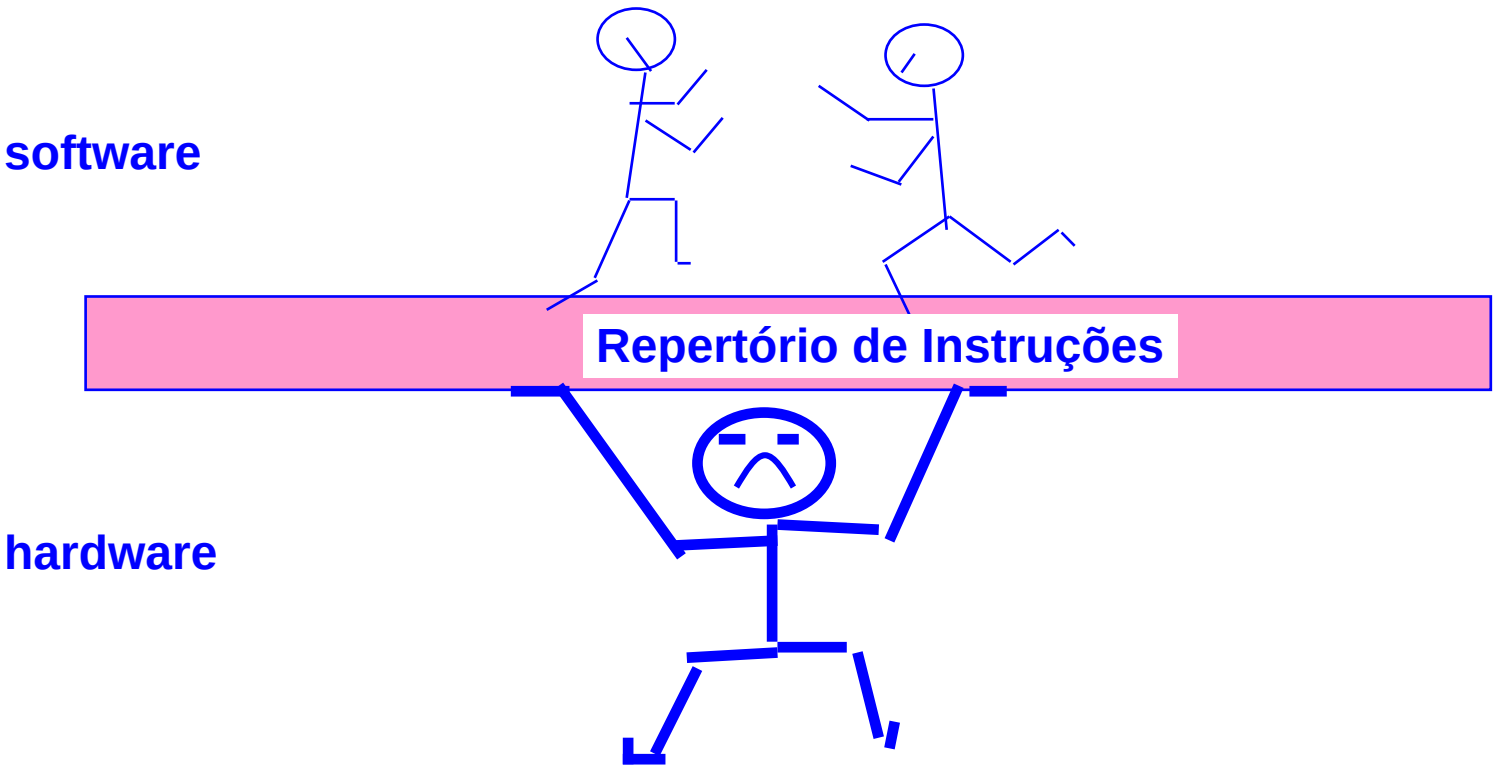
Computador: Hardware e Software



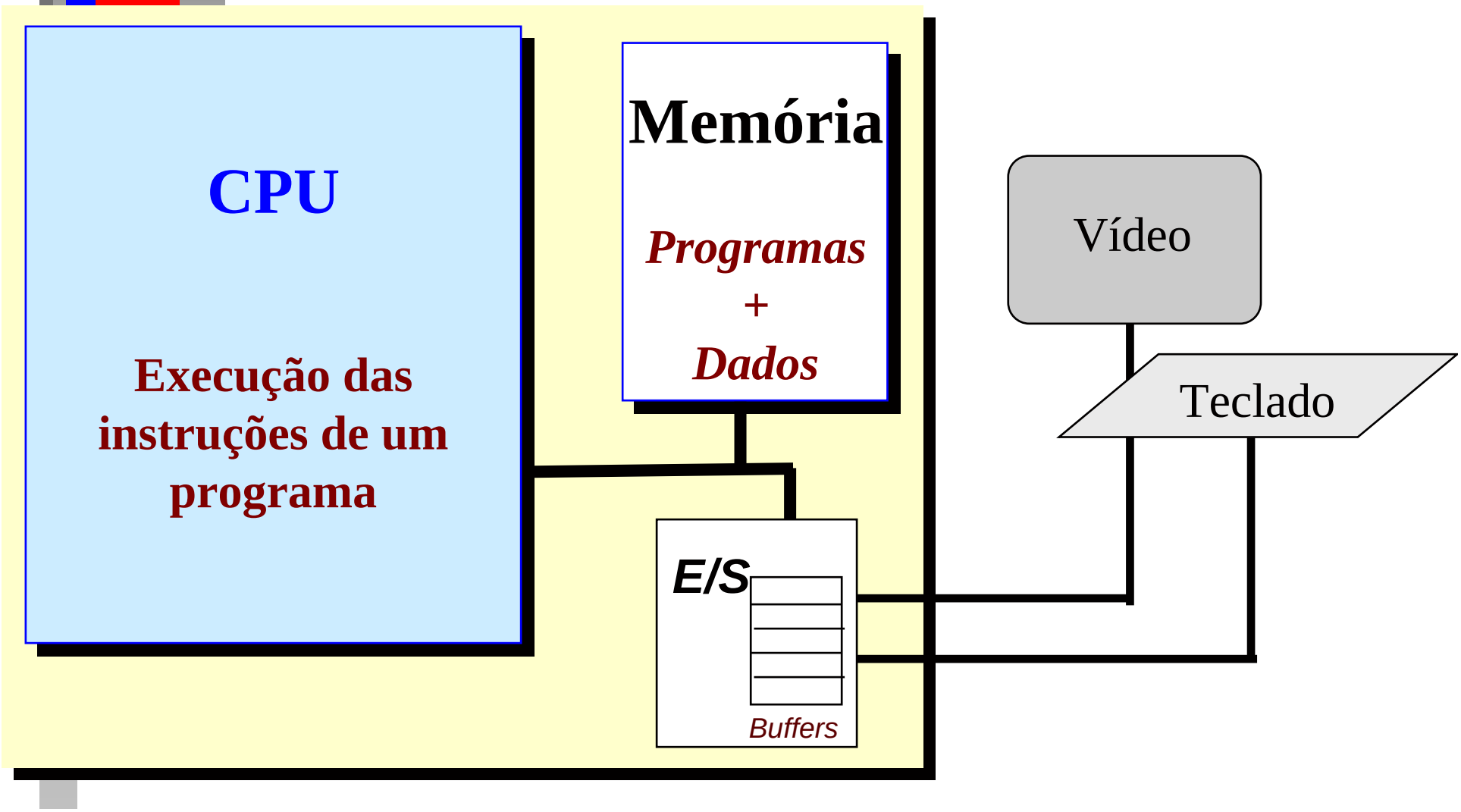
Computador: Hardware e Software



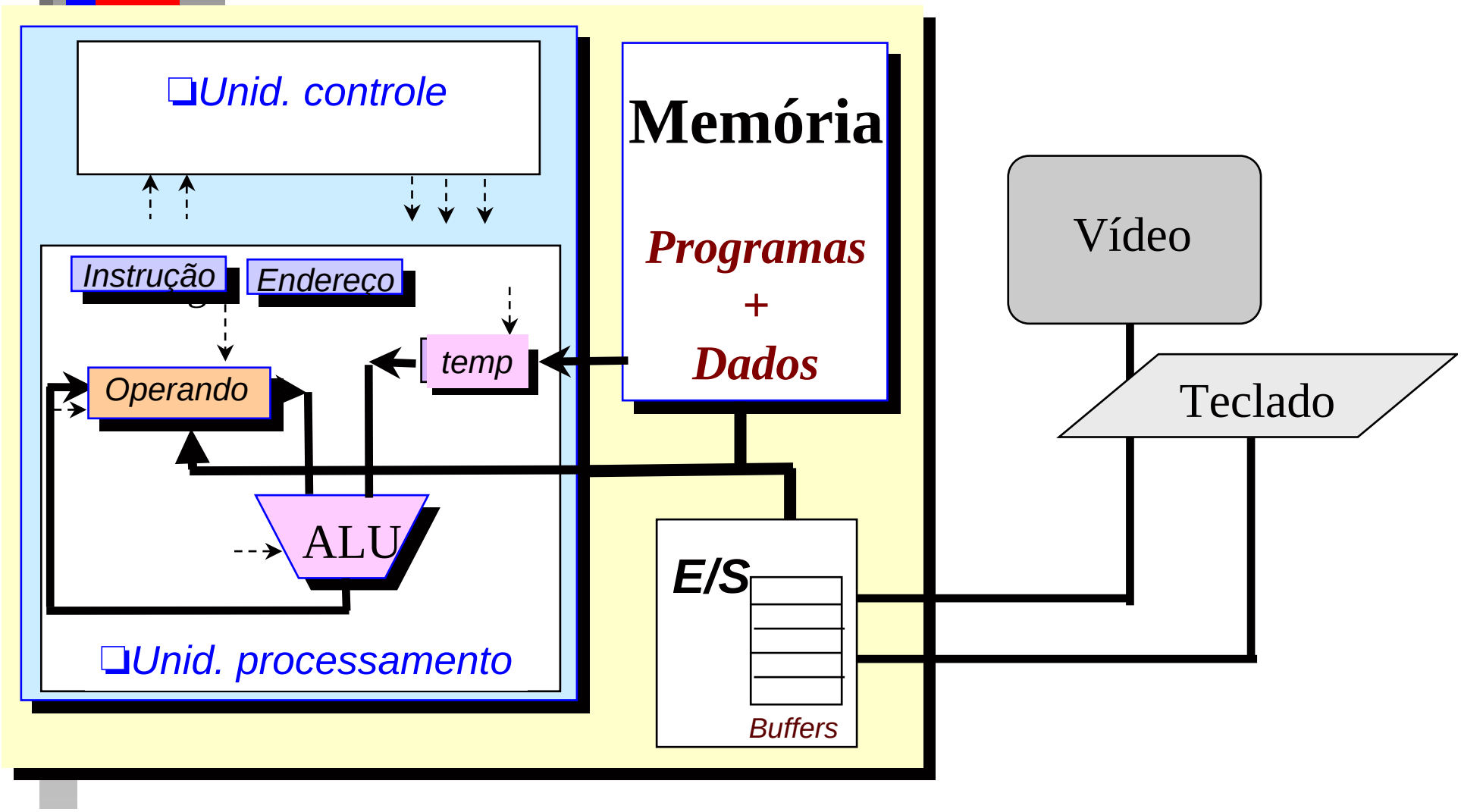
Interface entre hw e sw: Repertório de Instruções:



Como funciona um computador

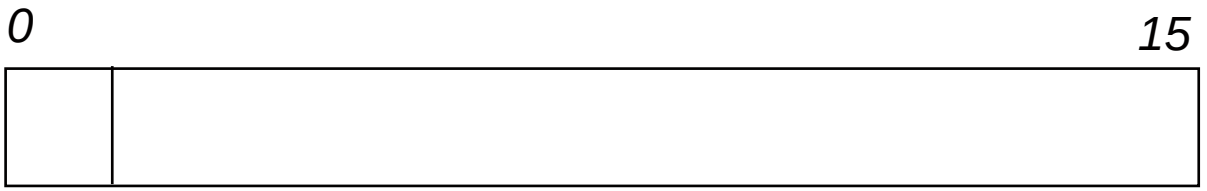


Como funciona um computador

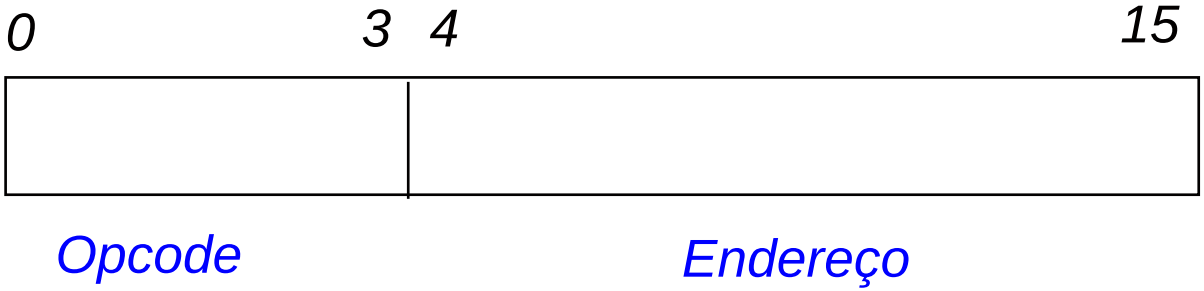


Memória

- Dado

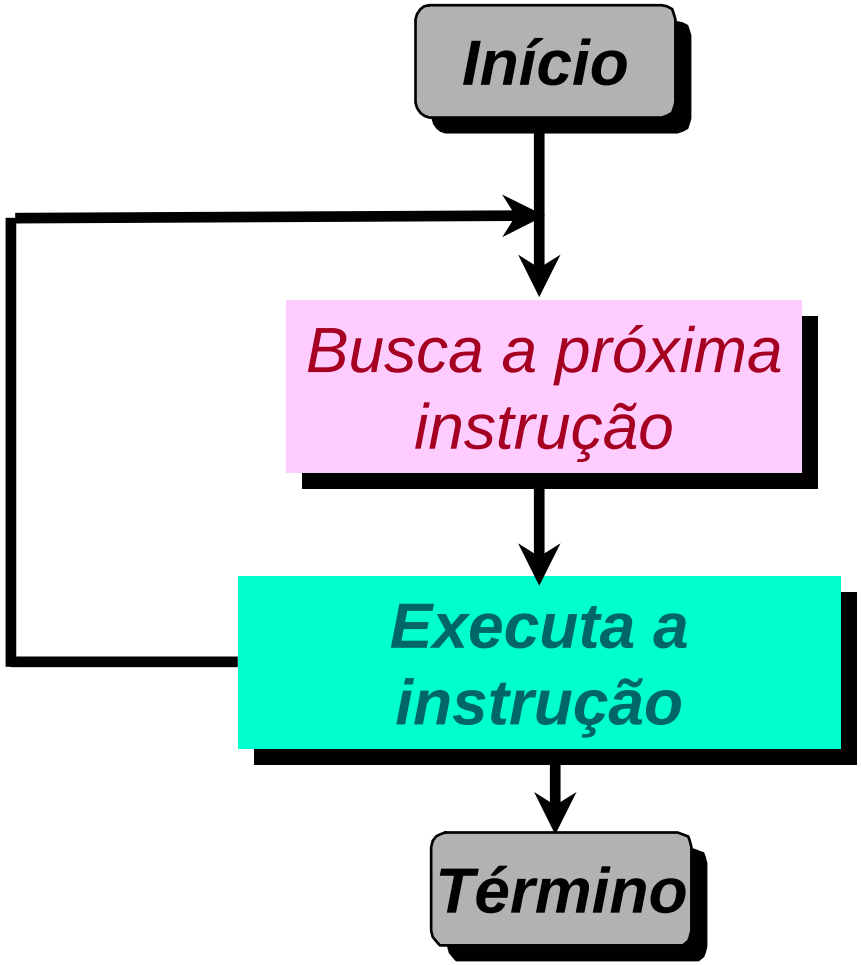


- Instrução

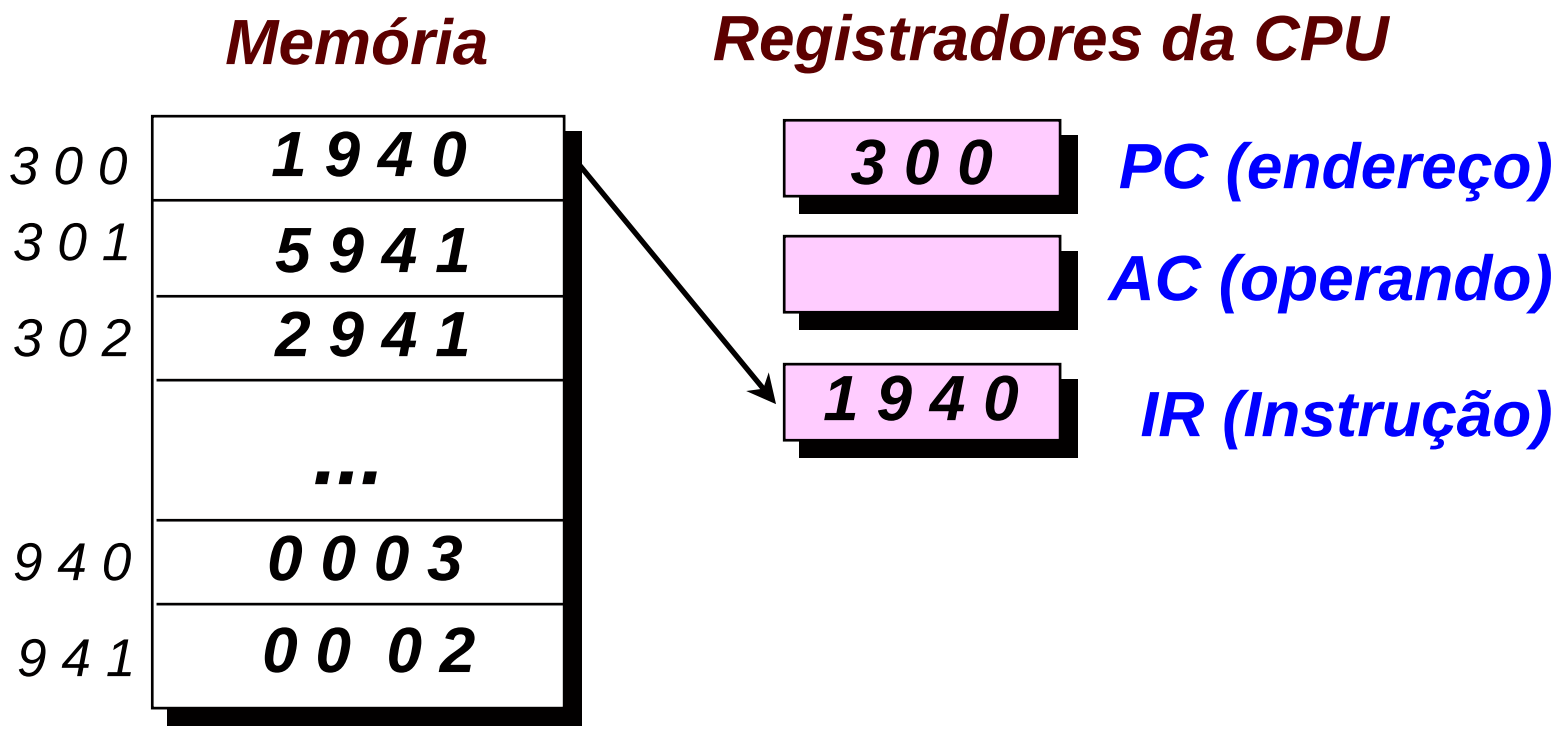


0001	AC <- Mem.
0010	Mem. <- AC
0101	AC <- AC + Mem.

Como funciona um computador?

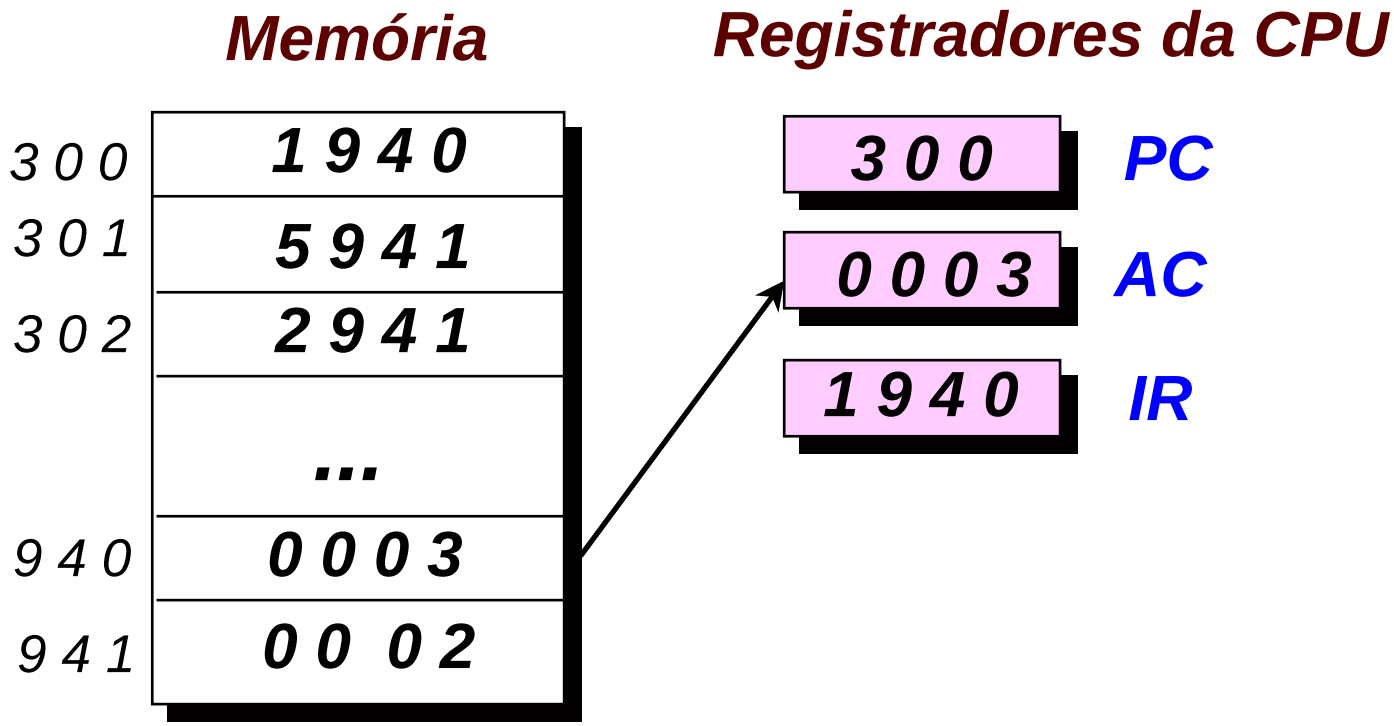


Executando um programa



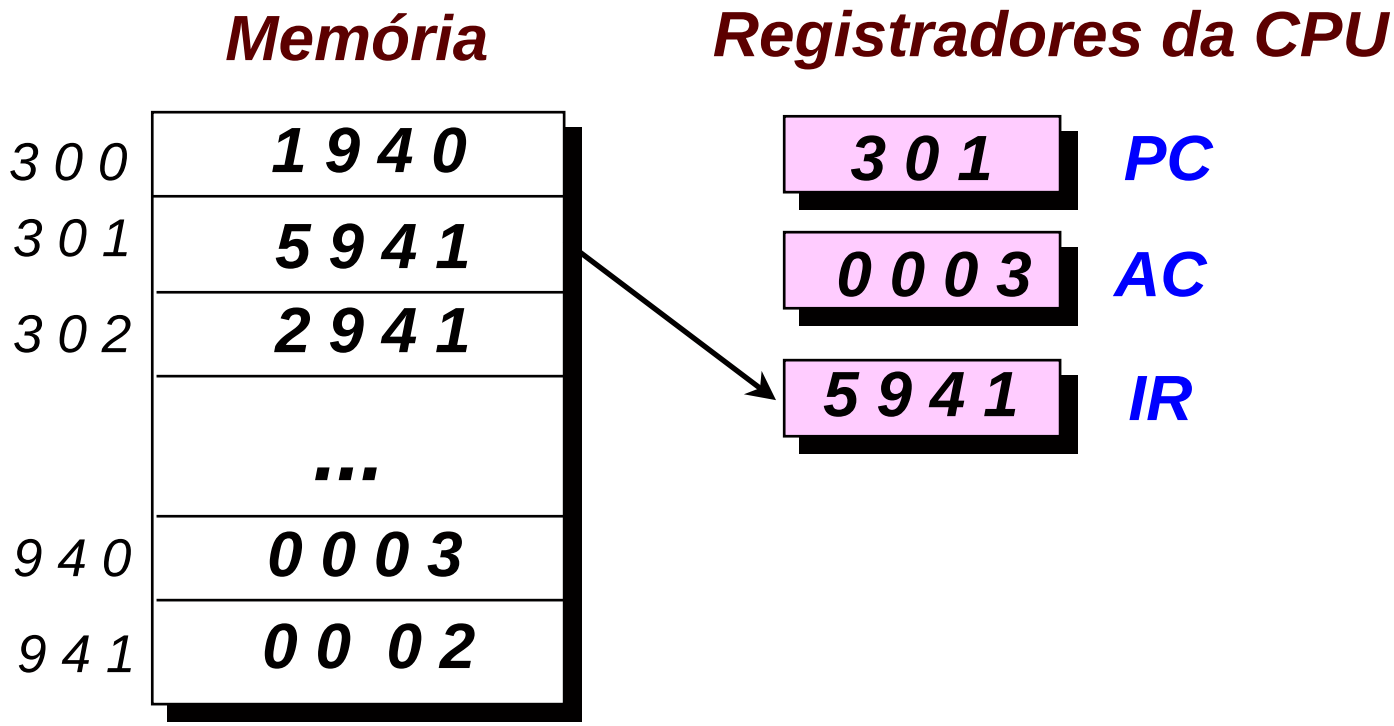
0001	AC <- Mem.
0010	Mem. <- AC
0101	AC <- AC + Mem.

Executando um programa



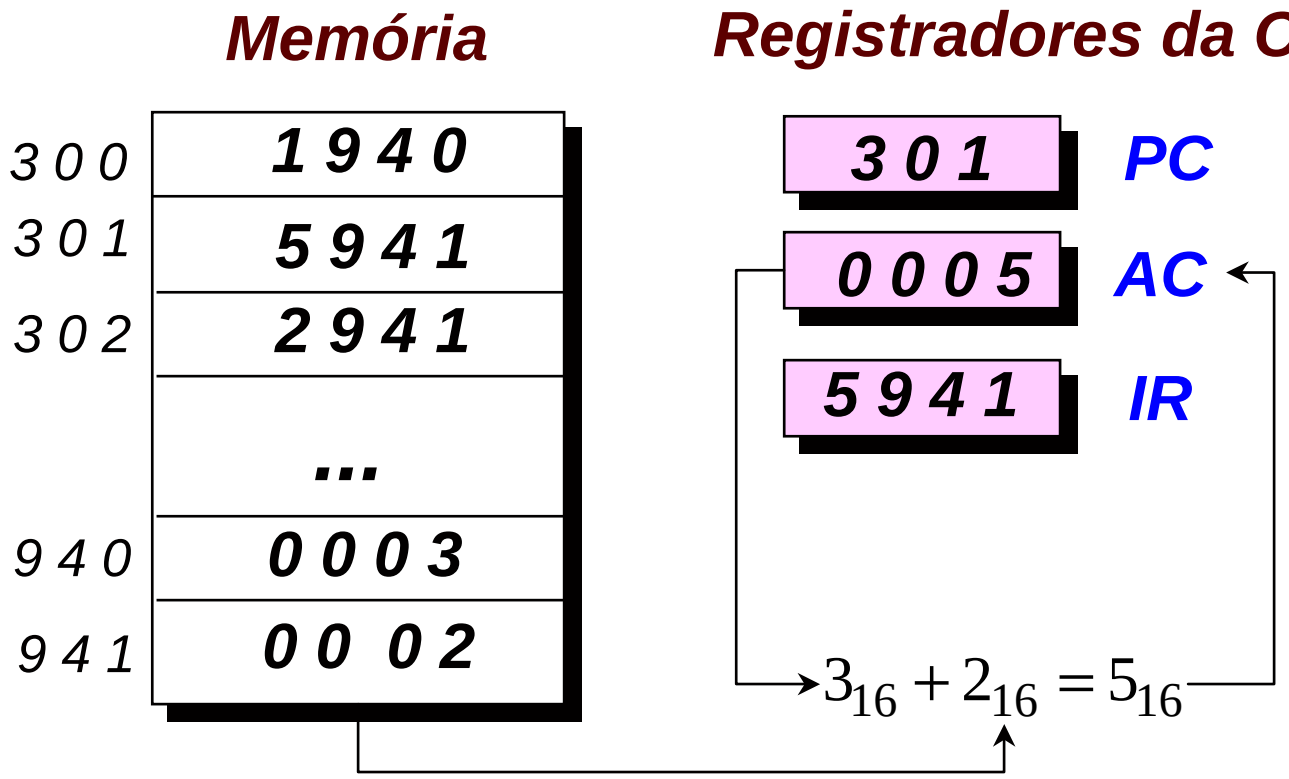
0001	AC <- Mem.
0010	Mem. <- AC
0101	AC <- AC + Mem.

Executando um programa



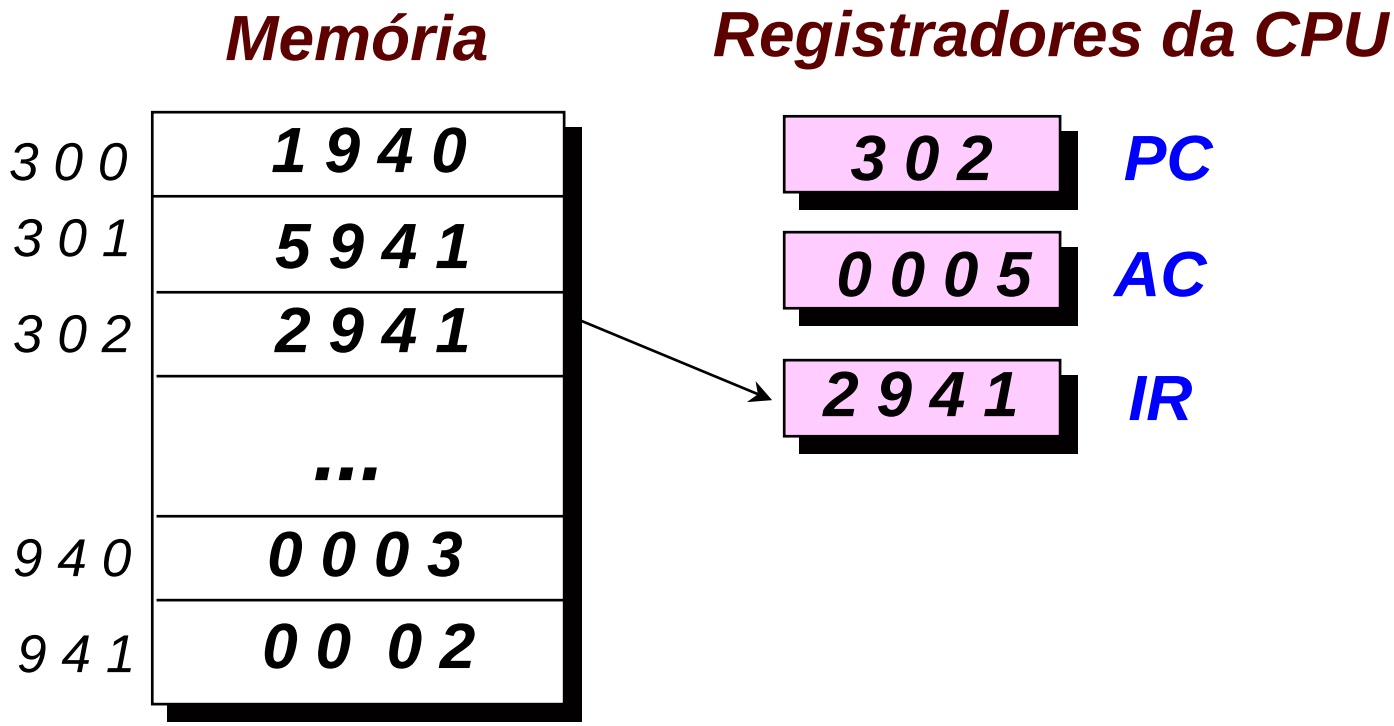
0001	AC <- Mem.
0010	Mem. <- AC
0101	AC <- AC + Mem.

Executando um programa



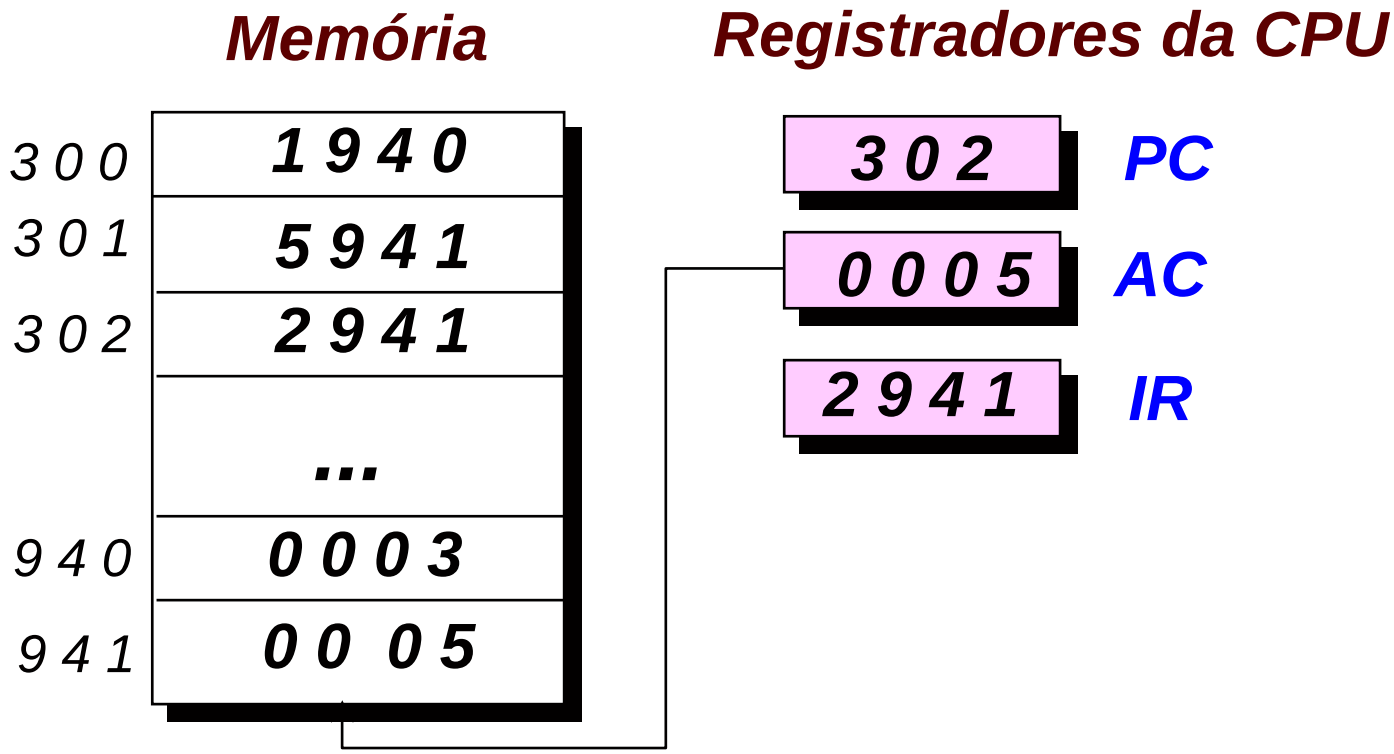
0001	AC <- Mem.
0010	Mem. <- AC
0101	AC <- AC + Mem.

Executando um programa



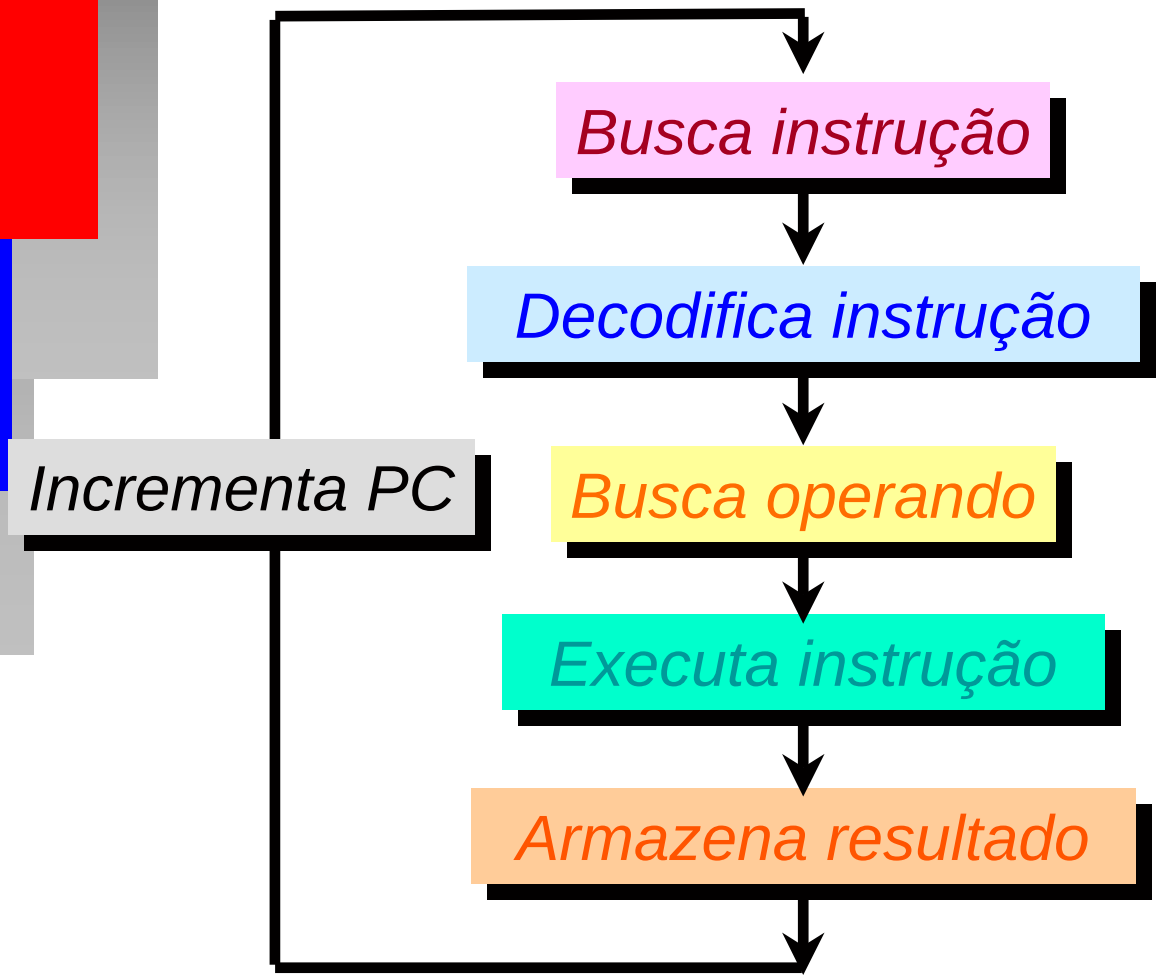
0001	AC <- Mem.
0010	Mem. <- AC
0101	AC <- AC + Mem.

Executando um programa



0001	AC <- Mem.
0010	Mem. <- AC
0101	AC <- AC + Mem.

Executando um programa



Infra-Estrutura de Hardware

- ***Arquitetura***
  - *Conjunto de registradores*
  - *Tipos de Dados*
  - *Acesso à memória*
  - *Formato e Repertório de instruções*

- ***Organização***
  - *Tecnologia de memória*
  - *Interfaces*
  - *Implementação das instruções*
  - *interconexões*

Operações aritméticas

- *Aritméticas*
  - *add a,b,c* — *a = b + c*
  - *a = b + c + d + e?*
  - *sub a,b,c* — *a = b - c*

Todas as instruções aritméticas possuem 3 operandos: destino, fonte 1, fonte 2

A simplicidade é favorecida pela regularidade

Expressões Aritméticas

- *f = (g + h) - (i + j)*
  - *Variáveis auxiliares: t0 e t1*
  - *add t0,g,h*
  - *add t1,i,j*
  - *sub f,t0,t1*

Operandos no Hardware

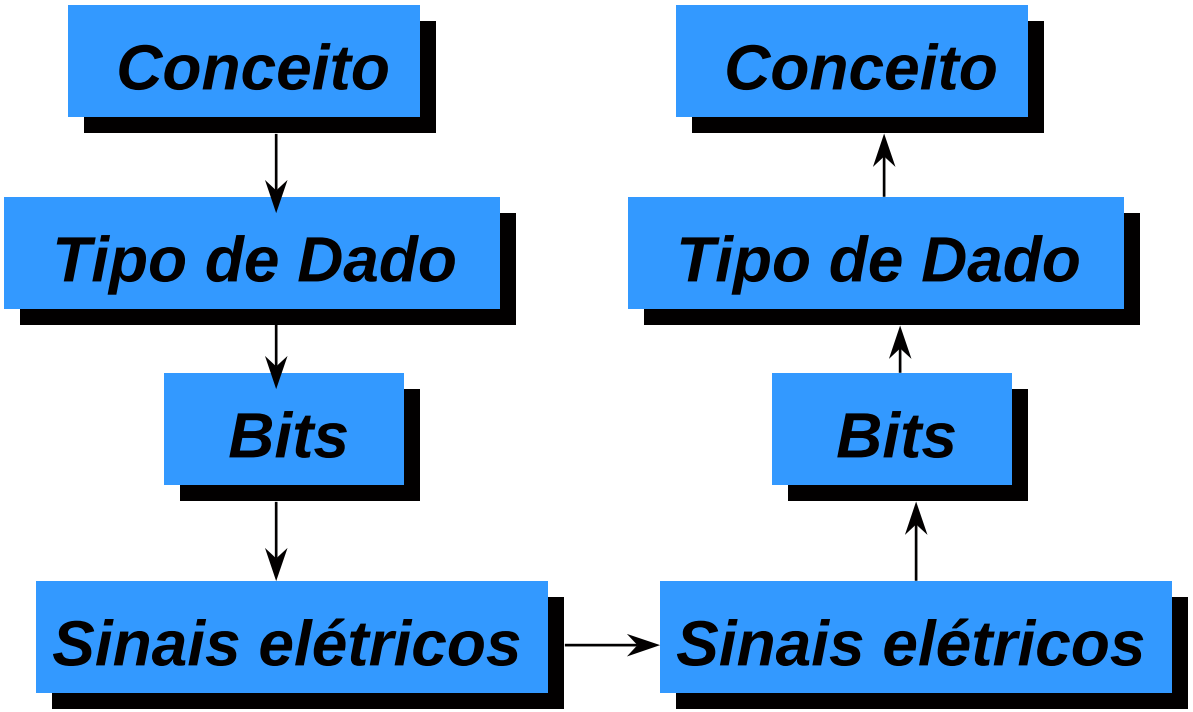
- *Para se garantir o desempenho....*
- *Operandos em registradores*
- *Vantagens:*
  - *leitura e escrita em registradores são muito mais rápidas que em memória*

Quanto menor mais rápido

Operandos no Hardware

- *Registradores*
  - *$s0, $s1, ... : armazenam variáveis do programa*
  - *$t0, $t1, ... : armazenam variáveis temporárias*
- *f = (g + h) - (i + j)?*
  - *Variáveis g,h,i e j estão armazenadas nos registradores $s0, $s1, $s2 e $s3*

Tipos de Dados



Conceitual

Ling. Progr.

Arquitetura

Físico

Tipos de Dados

- *Escalar*
  - *números*
    - *Inteiros*
    - *Ponto-Flutuante (real)*
  - *caracteres*
    - *ASCII*
    - *EBCDIC*
  - *dados lógicos*

Inteiros

- Representação binária
  - sinal-magnitude

0000000000001010	1000000000001010
+ 10	- 10

  - complemento a 1

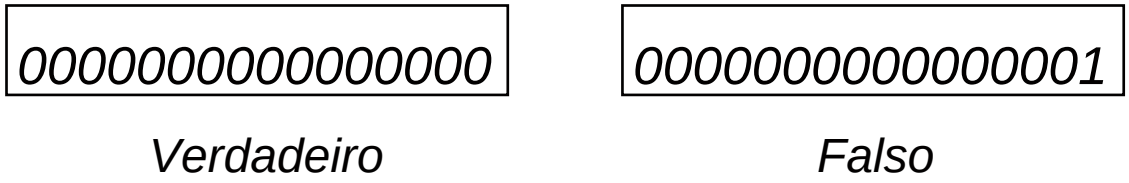
0000000000001010	1111111111110101
+ 10	- 10

  - complemento a 2

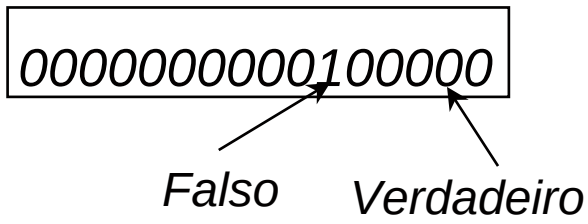
0000000000001010	1111111111110110
+ 10	- 10

Dados lógicos

- Representação
  - Uma palavra



  - Um bit



Ponto Flutuante

Representação

$$3{,}14 = 0{,}34 \times 10^{1} = 3{,}14 \times 10^{0}$$

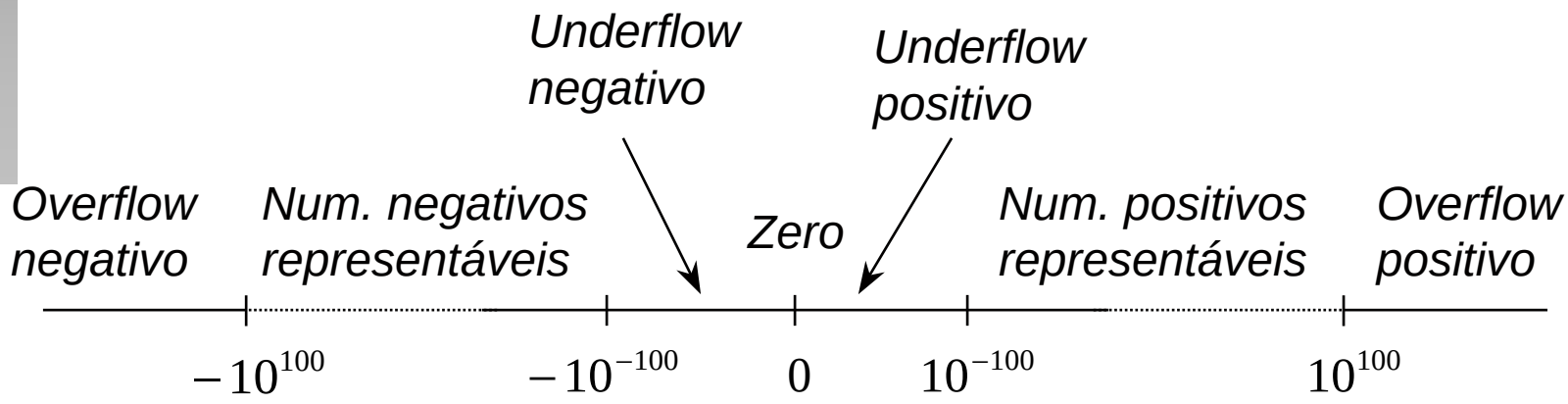
$$0{,}000001 = 0{,}10 \times 10^{-5} = 1{,}00 \times 10^{-6}$$

$$n = f \times 10^{e}$$

sinal	expoente	mantissa (fração)

Ponto Flutuante

Padrão IEEE



Item	Precisão simples	Precisão dupla
Sinal	**1**	**1**
Expoente	**8**	**11**
Mantissa	**23**	**52**
Total	**32**	**64**

Tipos de Dados

- *Escalar*
  - *números*
    - *Inteiros*
    - *Ponto-Flutuante (real)*
  - *caracteres*
    - *ASCII*
  - *dados lógicos*
- *Estruturas (Estático)*
  - *Array*
  - *Record*
- *Listas, Árvores (Dinâmico)*

Operandos na Memória

- *Manipulando arrays:*
  - *Armazenados na memória*
- *Instruções que permitam transferência de informação entre memória e registrador*

Instruções de Transferência de Dados
load word - lw

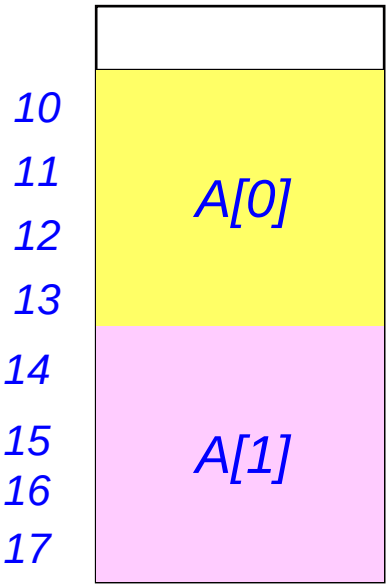
Array: endereço inicial de memória
elemento a ser transferido

Operandos na Memória

- *Arrays no MIPS:*
  - *endereço inicial:*
    - *registrador*
  - *elemento:*
    - *valor na instrução*
- *g = h + A[8] onde g e h estão nos registradores $s1 e $s2*
  - *lw $t0, 8($s3)*
  - *add $s1, $s2, $t0*

Operandos na Memória

- *MIPS*
  - *Inteiros com 32 bits*
  - *Memória endereçada por byte*
  - *Endereços múltiplos de 4*



End (A[0]) = 10
End (A[1]) = 14

End(A[i]) = End-inicial + i x 4

Operandos na Memória

- *Escrita em Memória*
  - *Store word, sw*
- *A[12] = h + A[8]*
  - *endereço inicial de A em $s3 e h em $s2*
  - *lw $t0, 32($s3)*
  - *add $t0, $s2, $t0*
  - *sw $t0, 48($s3)*

Operandos na Memória

- *Array com variável de indexação*
- *g = h + A[i]*
  - *endereço inicial de A em $s3 e g, h e i estão em $s1, $s2 e $s4*
  - *add $t1, $s4, $s4*
  - *add $t1, $t1, $t1*
  - *add $t1, $t1, $s3*
  - *lw $t0, 0($t1)*
  - *add $s1, $s2, $t0*

Resumo

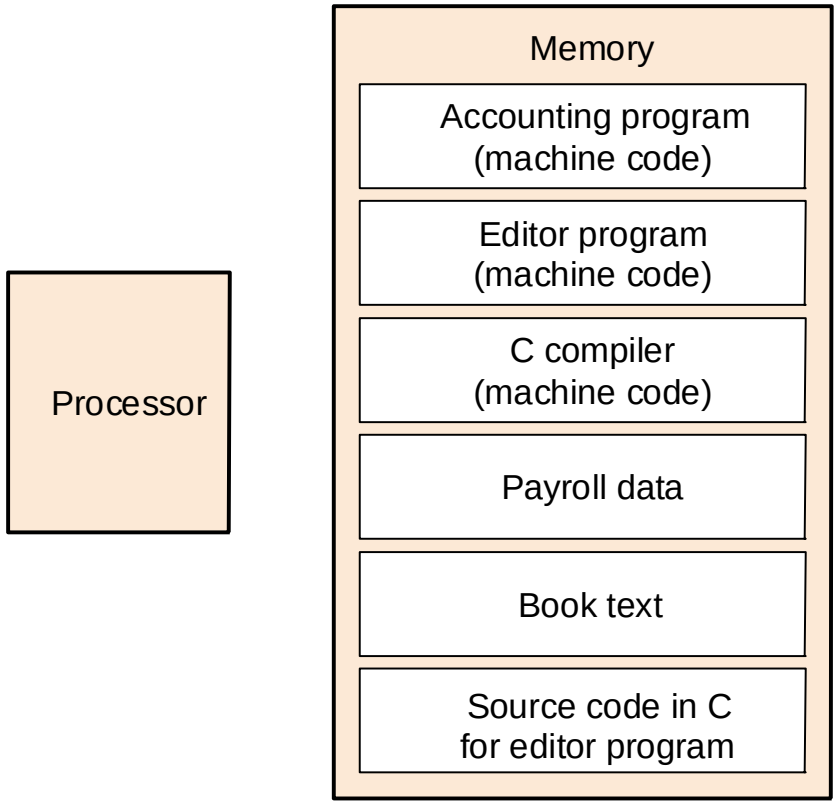
- *Operandos no MIPS*
  - *32 registradores*
    - *$s0, $s1,...*
    - *$t0, $t1,...*
  - *2^{30} palavras de memória*
    - *palavras de 32 bits*
    - *endereçamento por byte*

Resumo

- *Linguagem de montagem do MIPS*
  - *Aritméticas*
    - *add regi, regj, regk*
      - *regi = regj + regk*
    - *sub regi, regj, regk*
      - *regi = regj - regk*
  - *Transferência de dados*
    - *lw regi, desl(reg_base)*
      - *regi = mem(reg_base+desl)*
    - *sw regi, desl(reg_base)*
      - *mem(reg_base+desl) = regi*

Computador de Programa Armazenado

- *Instruções e Dados possuem uma representação numérica na memória*



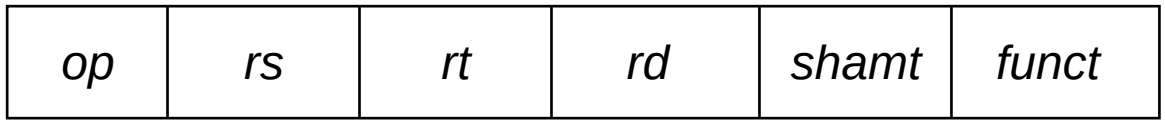
Representação das instruções

- *Informação tem uma representação numérica na base 2*
  - *codificação das instruções*
  - *mapeamento de nomes de registradores para números*
    - *$s0 a $s7 : 16 a 23*
    - *$t0 a $t7 : 8 a 15*
  - *add $t0, $s1, $s2*

00000	10001	10010	01000	00000	100000
6	5	5	5	5	6

Representação das instruções

- Instruções do MIPS
  - Aritméticas



  - Transferência de informação

Linguagem de Montagem vs. Linguagem de Máquina

- *A[300] = h + A[300]*
  - *lw $t0, 1200($t1)*
  - *add $t0, $s2, $t0*
  - *sw $t0, 1200($t1)*

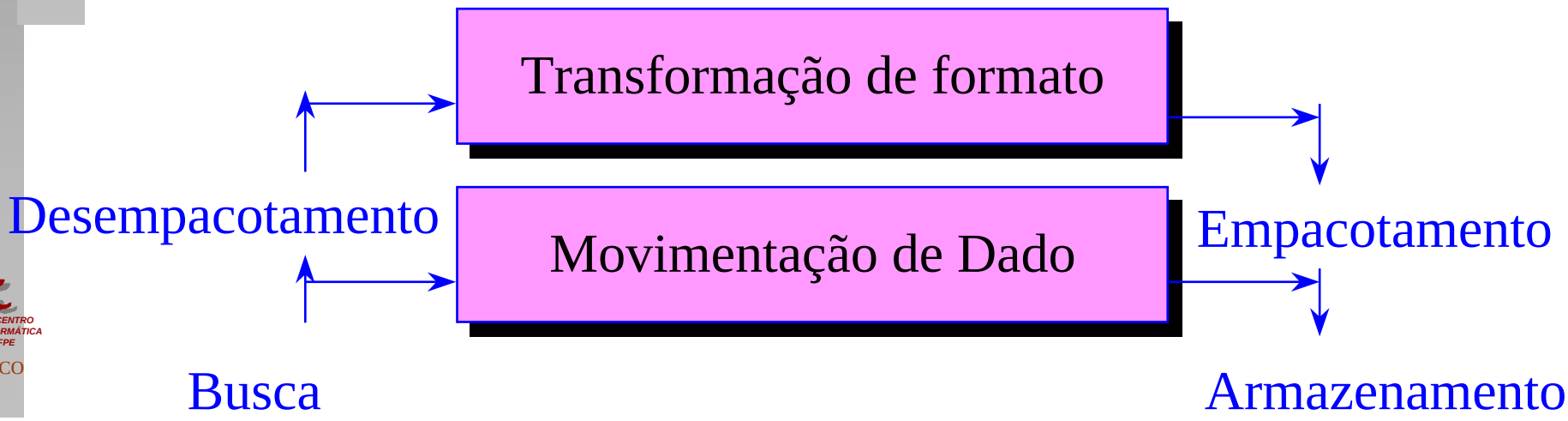
op	rs	rt	rd	Shamt/end	funct
35	9	8	1200		
0	18	8	8	0	32
43	9	8	1200		

op	rs	rt	rd	Shamt/end	funct
100011	01001	01000	0000 0100 1011 0000		
000000	10010	01000	01000	00000	100000
101011	01001	01000	0000 0100 1011 0000		

Exemplo: uma CPU simples...

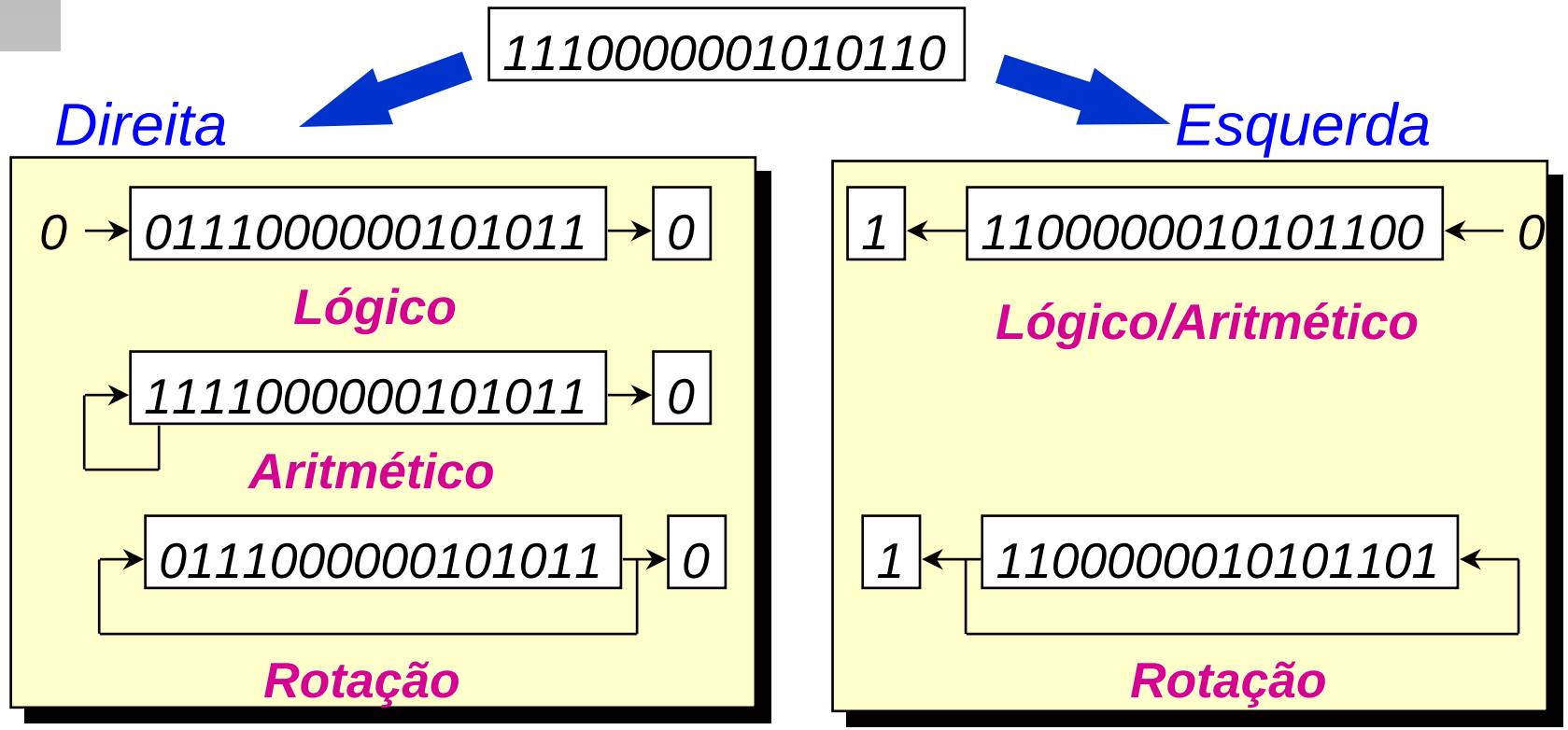
Instrução	Descrição
nop	No operation
lw reg, end(reg_base)	reg. = mem (reg_base+end)
sw reg, end(reg_base)	Mem(reg_base+end) = reg
add regi, regj,regk	Regi. <- Regj. + Regk
sub regi, regj, regk	Regi. <- Regj. – Regk

Mais operações sobre dados

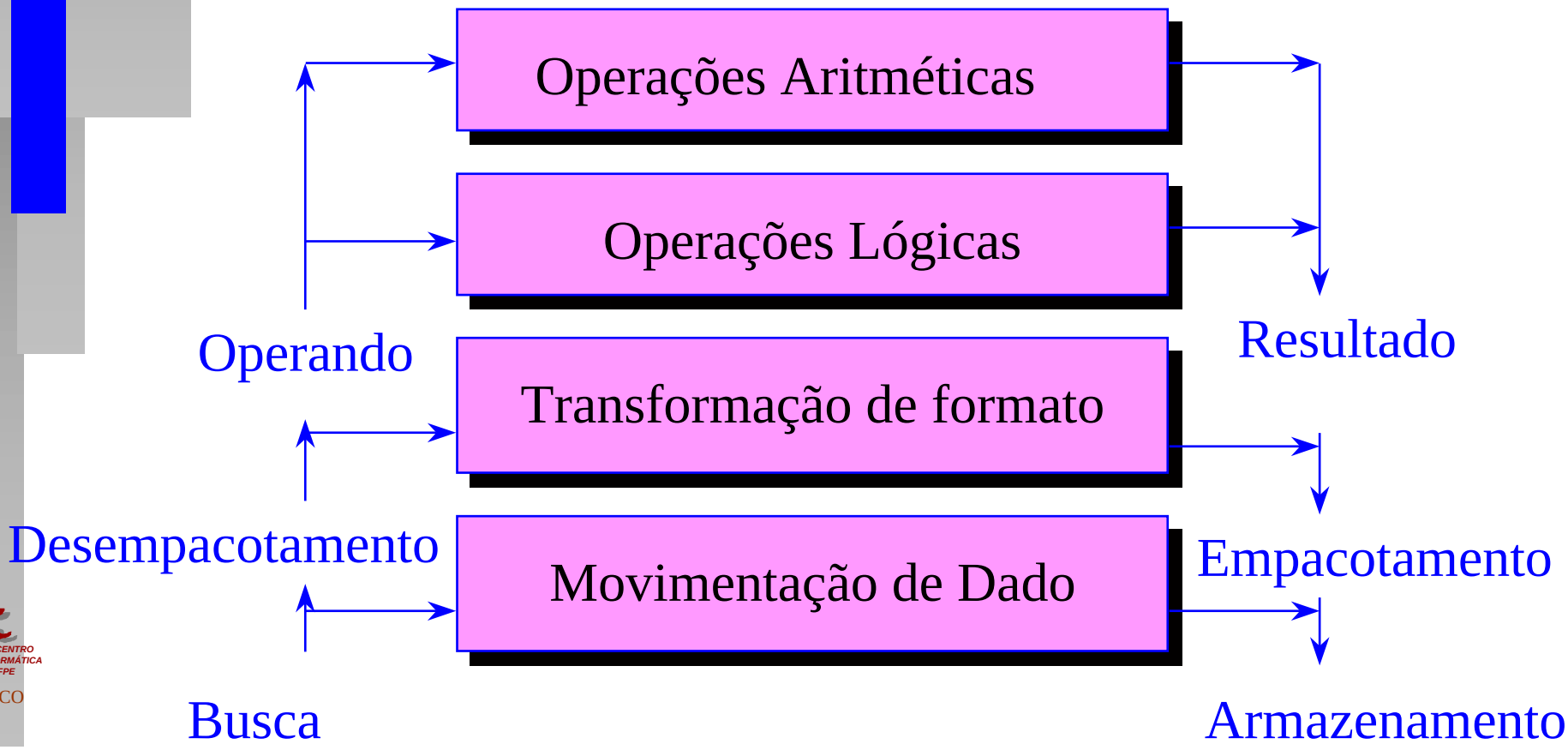


Transformação de Formato

- Afetam a localização dos bits
- Deslocamento e rotação

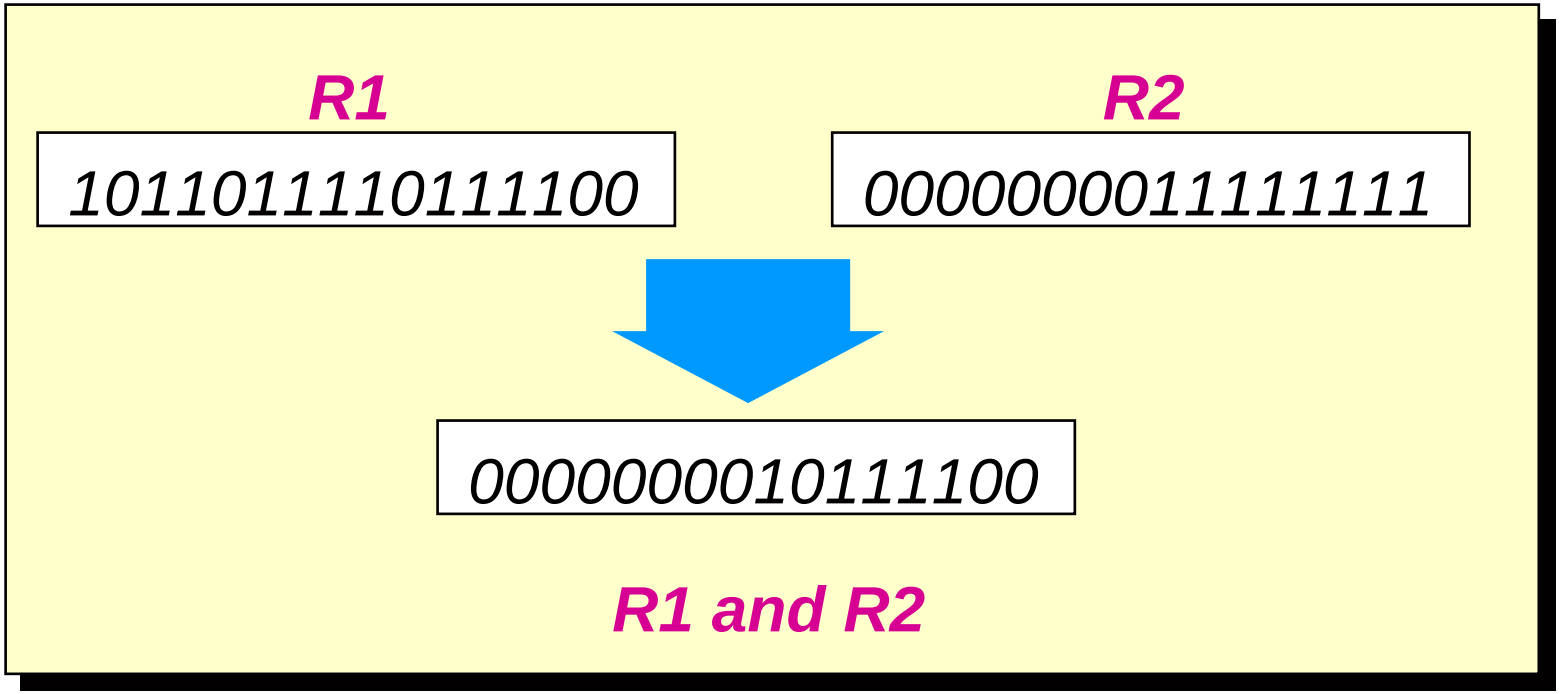


Operações sobre Dados



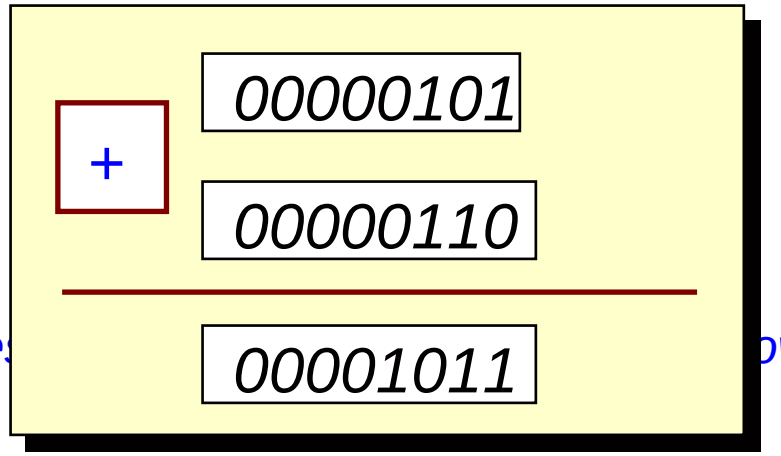
Operações Lógicas

- AND, OR, XOR, NOT
- Extração de grupos de bits



Operações Aritméticas

- Soma, subtração:
  - Tipos de dados
    - inteiros
    - ponto-flutuante
  - Exemplo:



✱ Pode gerar res ow e underflow)

Operações Aritméticas

- *Multiplicação*
  - *resultado: tamanho duplo*
- *Divisão*
  - *dividendo: tamanho duplo*
  - *resultados:*
    - *quociente*
    - *resto*

Controle de Fluxo

- *Alterar a sequência de execução das instruções:*

Ling. alto nível

- ***If ...then ...else***
- ***case***
- ***loop***
- ***go to***

Linguagem máquina

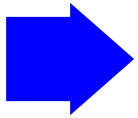
- ***Desvio incondicional***
- ***Desvio condicional a comparações entre variáveis e/ou valores***

Desvios no MIPS

- Realizado por duas instruções
  - jump endereço
  - branch if equal reg1, reg2, endereço

Ling. alto nível

```
If a=b then
   a:= a+c
else
   a: a-c
end if;
```



Linguagem máquina

```
        lw $S0,a
        lw $S1,b
        lw $S2,c
        beq $S0, $S1, end1
        sub $S0, $S0, $S2
        j end2
end1: add $S0, $S0, $S2
end2: sw $S0, a
```

Desvios condicionais no MIPS

- *Instruções de comparação e desvio:*
  - *Beq regd, regs, end_desvio*
    - *PC = PC + (desvio*4) se regd = regs, senão PC = PC +1*
  - *Bne regd, regs, end_desvio*
    - *PC = PC + (desvio*4) se regd ≠ regs, senão PC = PC +1*

Desvios condicionais no MIPS

- Instruções de Comparação:
  - set less than
  - slt regd, regs1, regs2
    - Regd = 1 se regs1 menor que regs2, caso contrário regd = 0
- Registrador com constante zero
- Desvio se maior ou igual???
  - slt $t0, $s0, $s1
  - beq $t0, $0, end

Desvios no MIPS

- *Instruções de desvio:*
  - *j endereço*
    - *PC = endereço*
- *Instruções de desvio indireto:*
  - *j reg*
    - *PC = (reg)*

MIPS

Instrução	Descrição
nop	No operation
lw reg, end(reg_base)	reg. = mem (reg_base+end)
sw reg, end(reg_base)	Mem(reg_base+end) = reg
add regi, regj,regk	Regi. <- Regj. + Regk
sub regi, regj, regk	Regi. <- Regj. – Regk
and regi, regj,regk	Regi. <- Regj. and Regk
srl regd, regs, n	Desloca regs para direita logico n vezes e armazena em regd
sra regd, regs, n	Desloca regs para dir. aritm. N vezes e armazena em regd
sll regd, regs, n	Desloca regs para esquerda n vezes
ror regd, regs, n	Rotaciona regs para direita n vezes
rol regd, regs, n	Rotaciona regs para esquerda n vezes
beq regi, regj , desl	PC=PC+desl*4 se regi = regj
bne regi, regj, end	PC=PC+desl*4 se regi <> regj
slt regi, regj, regk	Regi =1 se regj < regk senão regi=0
j end	Desvio para end
j regd	Desvio para endereço em regd

Subrotinas

- Implementação de procedimentos e funções

```
Ling. alto nível
    Programa principal
    Var i,j,k: integer
    Procedure A (var x: integer);
    ...
    Begin
    ...(corpo do procedimento)
    End;
    Begin
    ...
    A(k); (chamada do procedimento)
    ....
    End;
```

Retorno após a chamada

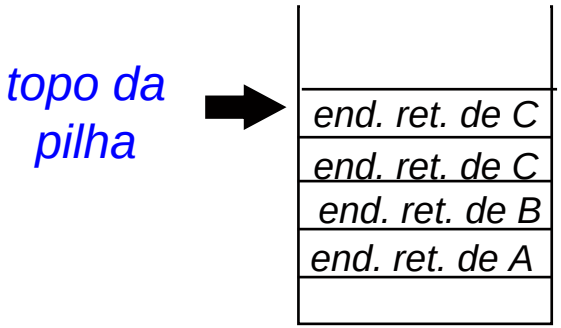
O endereço de retorno deve ser salvo...

mas onde?

Onde salvar o endereço de retorno

- *registradores*
  - *só permite chamadas seriais*
- *pilha*
  - *permite aninhamento e recursividade*

```
...
Procedure A
begin
   Procedure B
    begin
       Procedure C
        begin
           ...
           C
           ...
        end
    ...
```

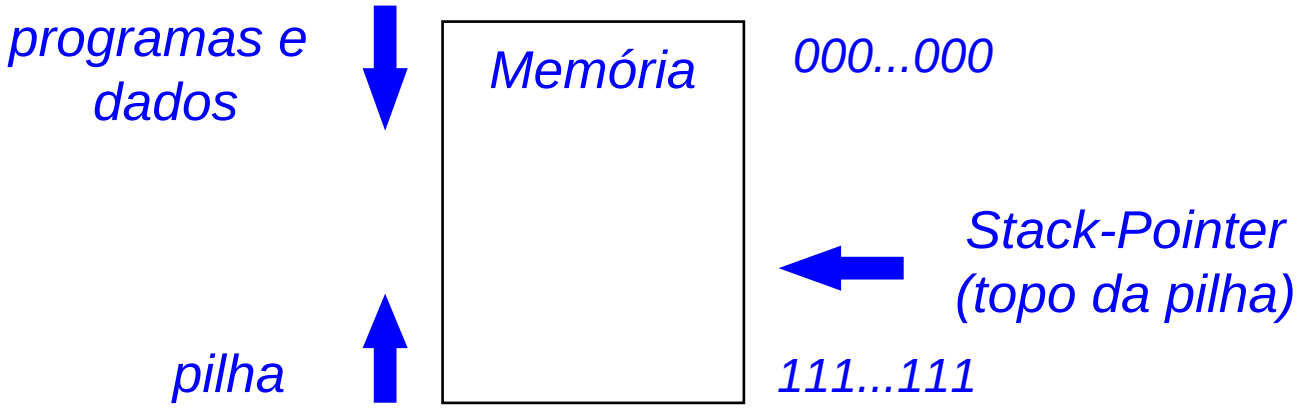


Chamada de subrotina - MIPS

- *Instruções:*
  - *jal*
    - *guarda endereço de retorno em $ra (reg. 31)*
    - *muda fluxo de controle*
  - *jr - jump register*
    - *recupera endereço de retorno de $ra*

Implementando a pilha

- Utiliza parte da memória como pilha



  - MIPS: stack-pointer (Reg 29)

Implementando a pilha

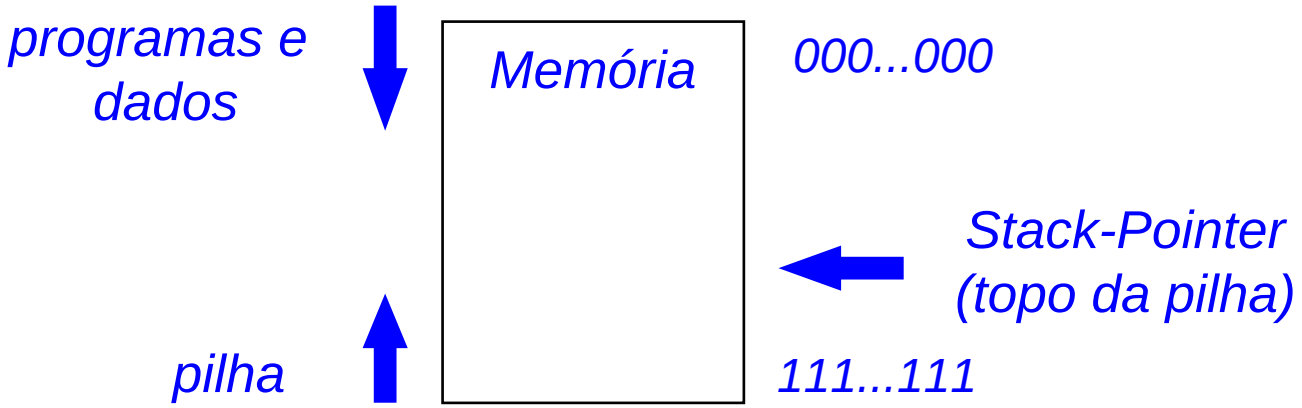
- *Instruções:*
  - *coloca registrador na pilha:*
    - *lw $28, UM*
    - *sw reg, 0($29)*
    - *sub $29, $29, $28*
  - *retira registrador da pilha*
    - *lw $28, UM*
    - *add $29, $29, $28*
    - *lw reg, 0($28)*

Chamada de subrotina em outras arquiteturas

- *Instruções:*
  - *call*
    - *empilha endereço de retorno*
    - *muda fluxo de controle*
  - *ret*
    - *recupera endereço de retorno*
- *Outras instruções de suporte...*
  - *Salvar todos registradores na pilha*
  - *alocar parte da pilha para armazenar variáveis locais e parâmetros*

Usando a pilha em outras arquiteturas

- Utiliza parte da memória como pilha



  - SP: Registrador adicional
- Instruções adicionais:
  - push reg: mem(SP) ← reg ; SP = SP - 1
  - pop reg: SP = SP + 1, reg ← mem(SP);

MIPS vs. Outras arquiteturas

- *Endereço de retorno:*
  - *MIPS: registrador*
  - *Outras: Memória*
  - *Melhor desempenho*
- *Acesso à Pilha:*
  - *MIPS: instruções lw e sw*
  - *Outras: instruções adicionais*
  - *Menor complexidade na implementação*
  - *Compilador mais complexo*

MIPS vs. Outras arquiteturas

- *Chamadas aninhadas ou recursivas*
  - *MIPS: implementada pelo compilador*
  - *Outras: suporte direto da máquina*
  - *Compilador mais complexo*

MIPS

Instrução	Descrição
nop	No operation
lw reg, end(reg_base)	reg. = mem (reg_base+end)
sw reg, end(reg_base)	Mem(reg_base+end) = reg
add regi, regj,regk	Regi. <- Regj. + Regk
sub regi, regj, regk	Regi. <- Regj. – Regk
and regi, regj,regk	Regi. <- Regj. and Regk
xor regi, regj, regk	Regi = regj xor regk
srl regd, regs, n	Desloca regs para direita n vezes sem preservar sinal, armazena valor deslocado em regd .
sra regd, regs, n	Desloca regs para dir. n vezes preservando o sinal, armazena valor deslocado em regd.
sll regd, regs, n	Desloca regs para esquerda n vezes, armazena valor deslocado em regd.
ror regd, regs, n	Rotaciona regs para direita n vezes, armazena valor deslocado em regd.
rol regd, regs, n	Rotaciona regs para esquerda n vezes, armazena valor deslocado em regd.
beq regi, regj, desl	PC = PC + desl*4 se regi = regj
bne regi, regj, desl	PC = PC + desl *4 se regi <> regj
slt regi, regj, regk	Regi =1 se regj < regk senão regi=0
j end	Desvio para end
Jr reg	Pc = reg
Jal end	Reg15 = pc, pc = endereço
break	Para a execução do programa

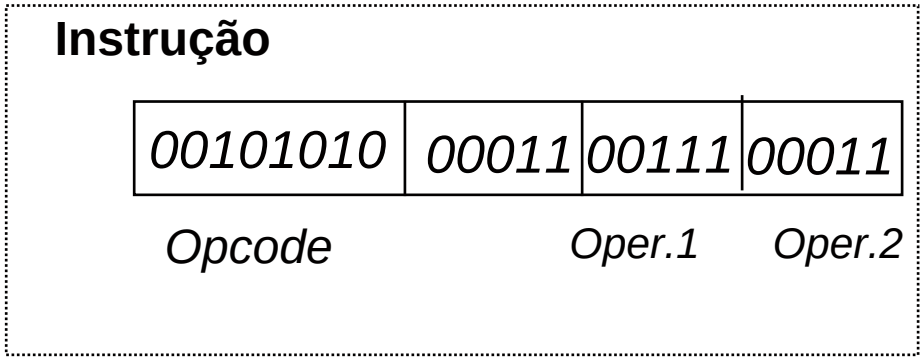
Outros Modos de Endereçamento

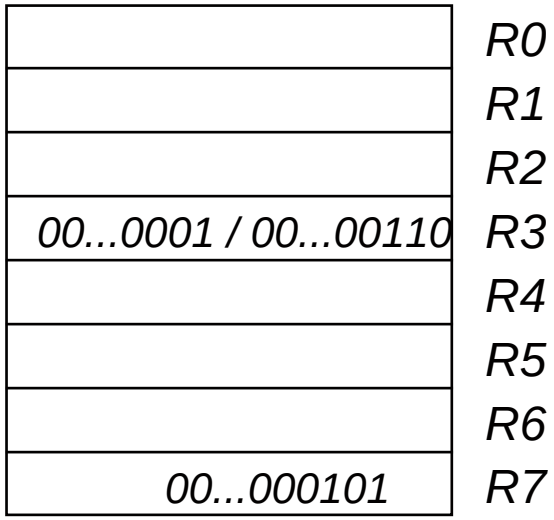
Operandos no MIPS

- *Aritméticas*
  - *Registradores*
- *Load, store*
  - *Memória*

Endereçamento de registrador

- Operações aritméticas:
  - O operando está em um registrador e a instrução contem o número do registrador
    - ADD R3, R3, R7
      - R3 <- R3 + R7

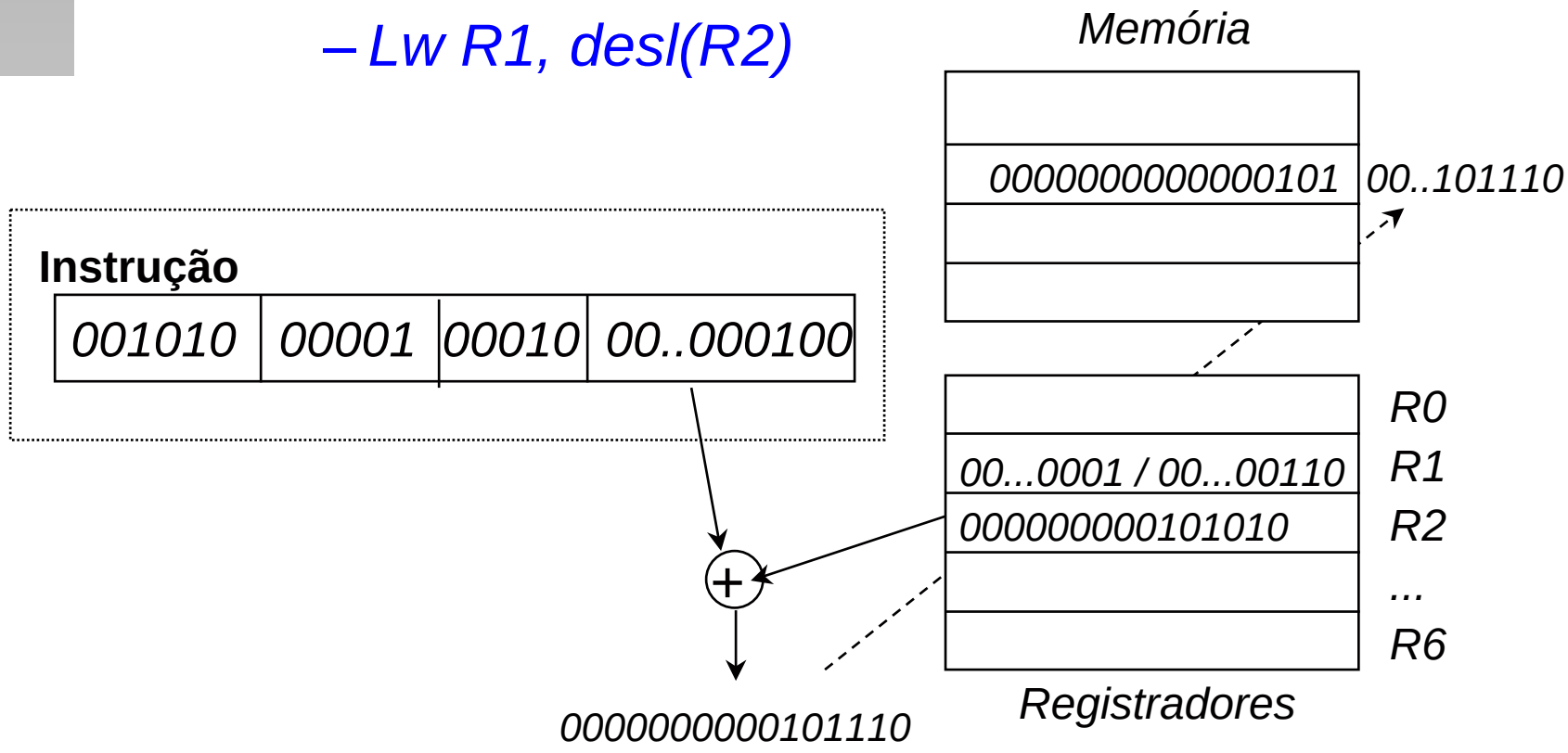




Registradores

Endereçamento base

- Instruções de acesso à memória:
  - Instrução:deslocamento
  - Registrador de base:end- inicial
    - Lw R1, desl(R2)

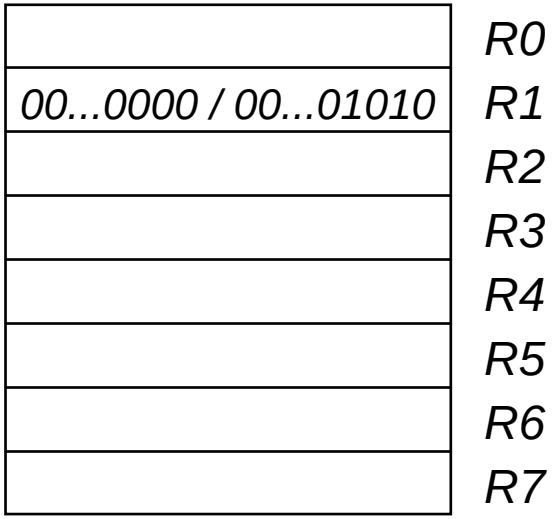


Endereçamento imediato

- Operações aritméticas e de comparação:
  - O operando é especificado na instrução
    - ADDI R1, R2, #A

Instrução

001011	00010	00001	000...01010



Registradores

- Modo bastante frequente
- Operando constante

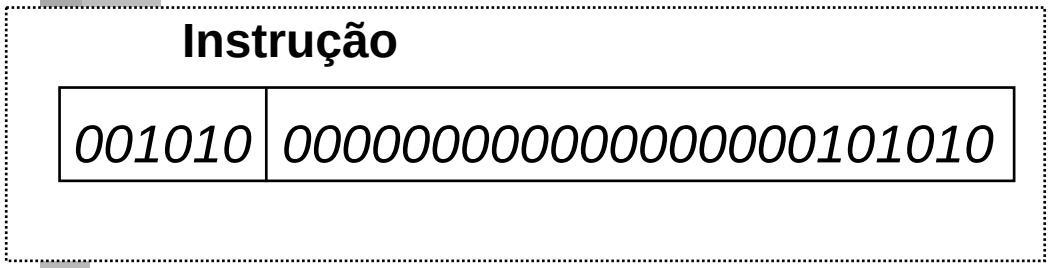
Endereços no MIPS

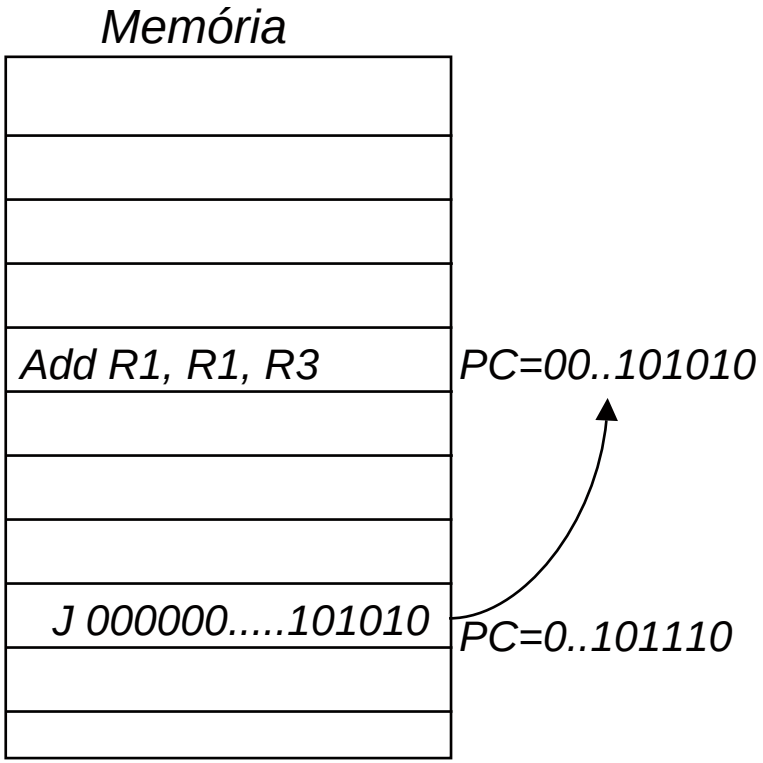
- *Endereço na Instrução*
  - *J endereço*
  - *PC = endereço*
- *Endereço em Registrador*
  - *Jr reg*
  - *PC = (reg)*
- *Endereço relativo a registrador*
  - *Beq deslocamento*
  - *PC = PC + deslocamento*4*

Endereçamento (Pseudo)Direto

Instrução de Desvio:

- o endereço da próxima instrução é especificado na instrução
  - J end1
  - PC <- end1

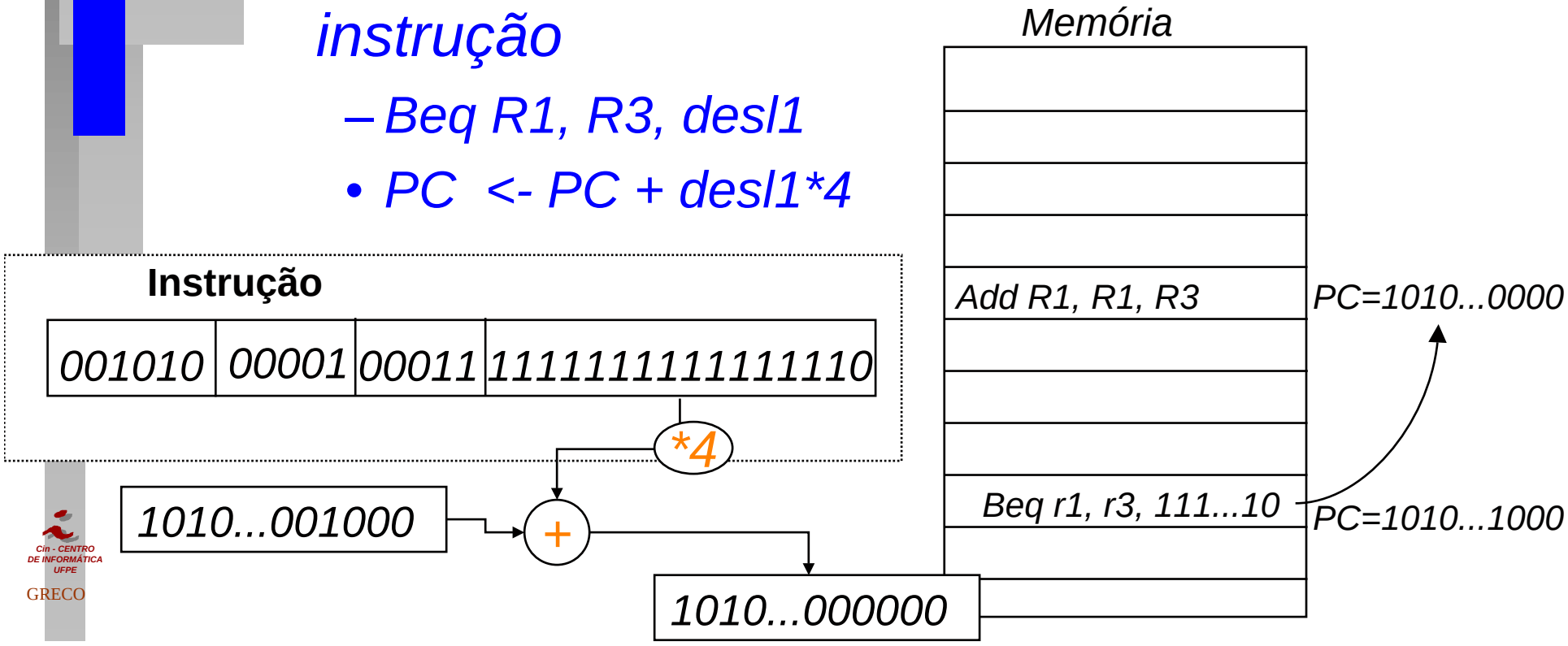




Endereçamento Relativo a PC

Instrução de Branch:

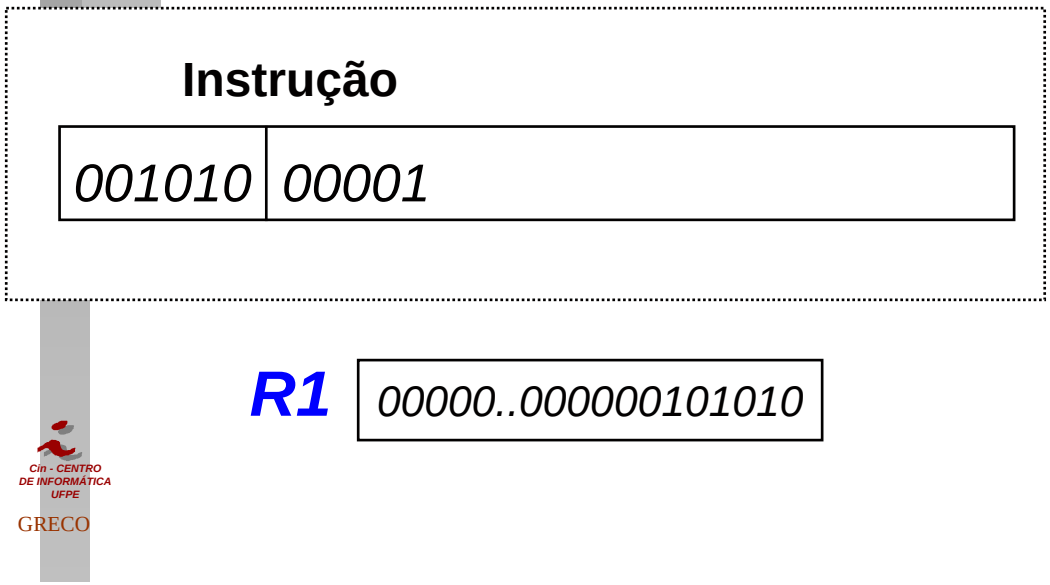
- o número de instruções a serem puladas a partir da instrução é especificado na instrução
  - Beq R1, R3, desl1
  - PC <- PC + desl1*4

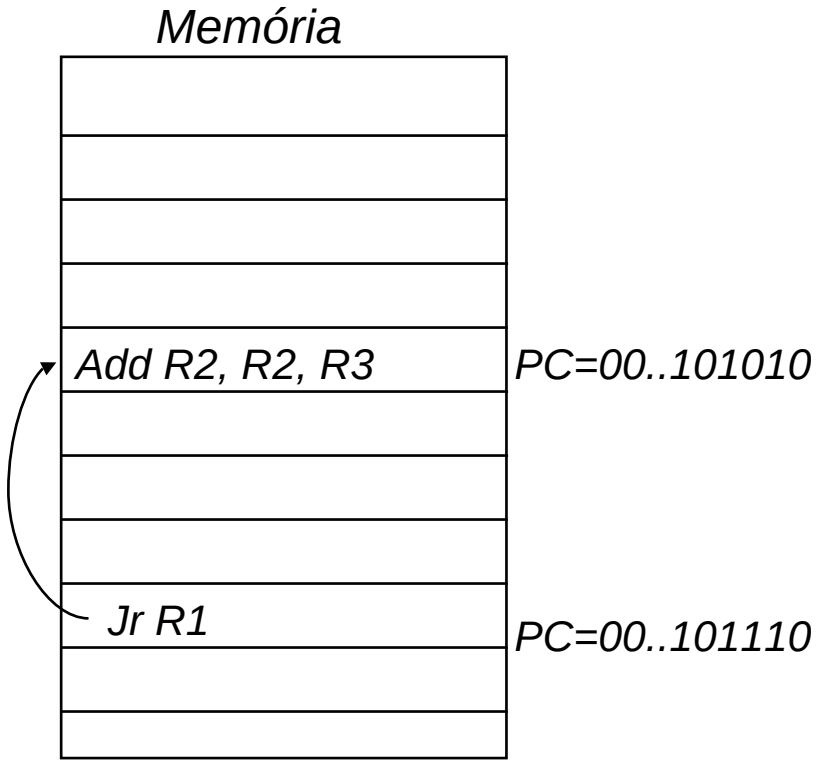


Endereçamento de Registrador

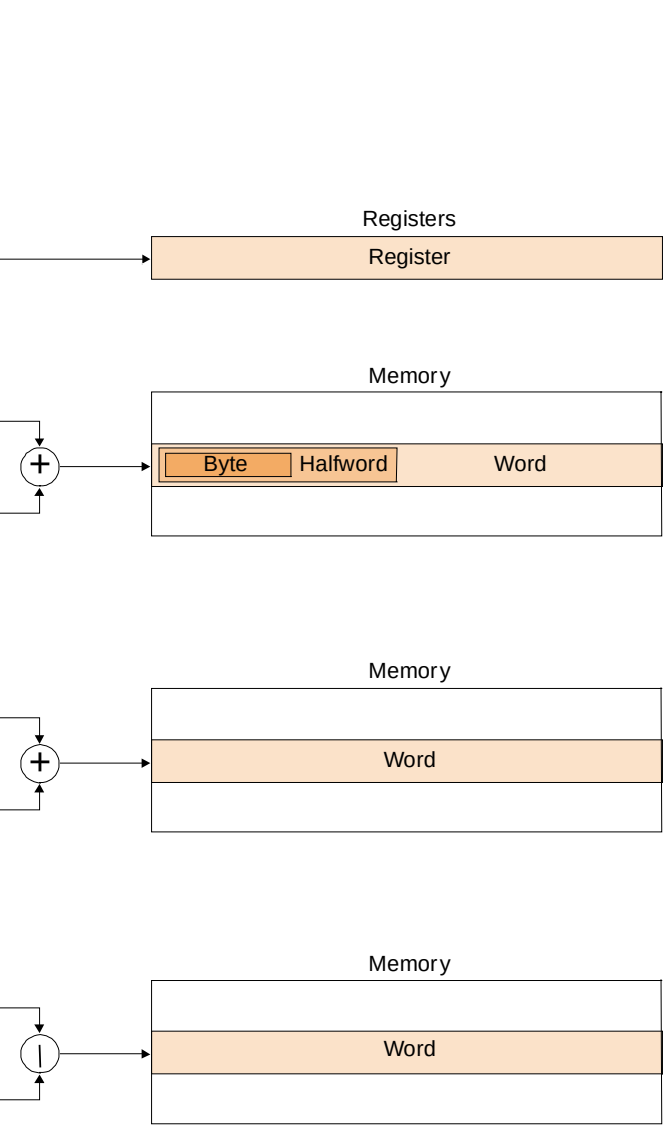
Instrução de Desvio:

- o endereço do operando na memória é especificado em um registrador
  - Jr R1
    - PC <- (R1)





Modos do MIPS



Em geral...

- *Aritméticas:*
  - *Operandos em registradores*
  - *Operandos em memórias*
  - *...*
- *Vários modos de endereçamento*

Como especificar na instrução onde está o operando e como este pode ser acessado ?

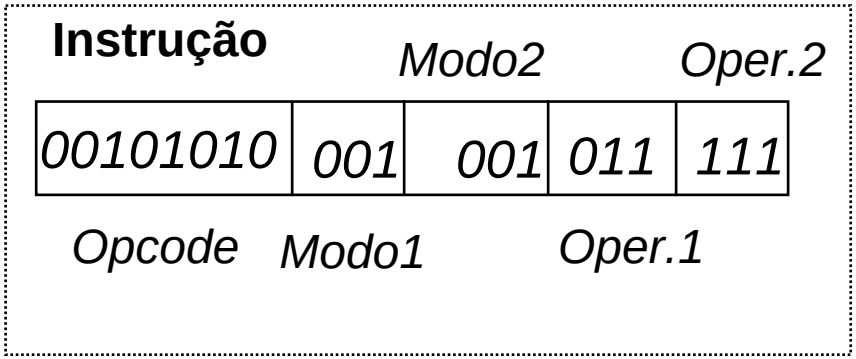
Campo na Instrução

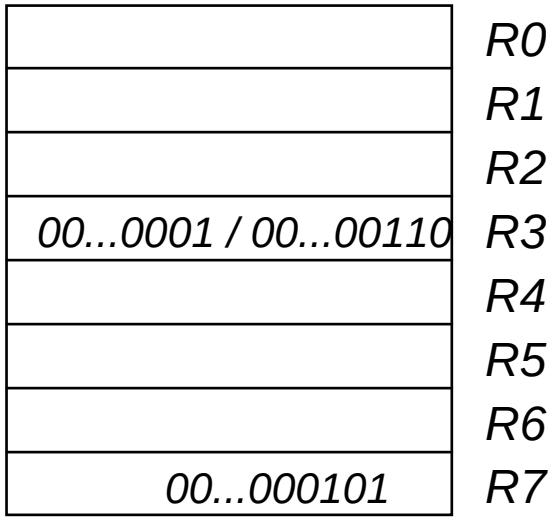
Endereçamento em outras arquiteturas

- *Aritméticas*
  - *Registrador-Registrador*
  - *Registrador - Memória*
  - *Memória - registrador*
  - *Memória - Memória*
- *Exemplo: Adição*

Endereçamento de registrador

- O operando está em um registrador e a instrução contem o número do registrador
  - ADD R3, R7
    - R3 <- R3 + R7





Registradores

Endereçamento Direto

- O endereço do operando na memória é especificado na instrução
  - ADD R1, end2
    - R1<- R1+ [end2]

Instrução

	Modo1	Modo2	
00101010	001	010	001
0000000000101010			

- instruções maiores

Registradores

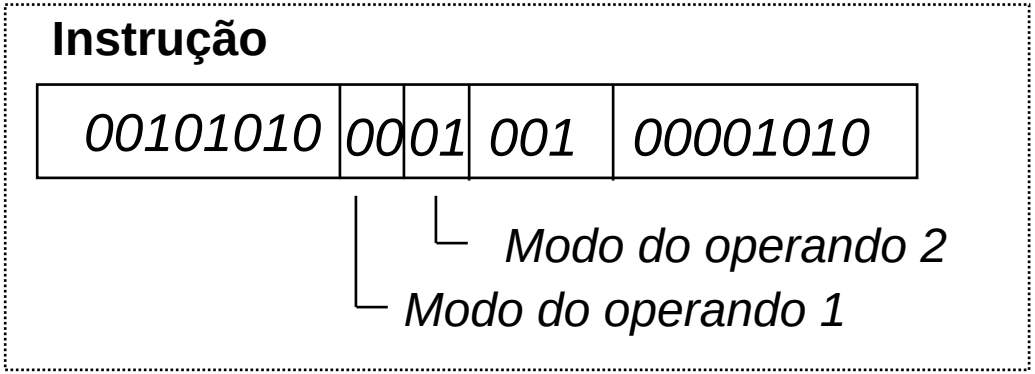
	R0
00...0001 / 00...00010	R1
	R2
	R3
	R4
	R5
	R6
	R7

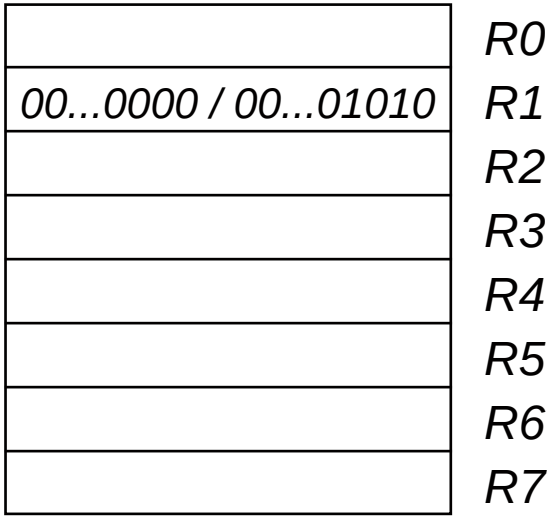
Memória

00000...0000001	00..101010

Endereçamento imediato

- O operando é especificado na instrução
  - ADD R1, #A



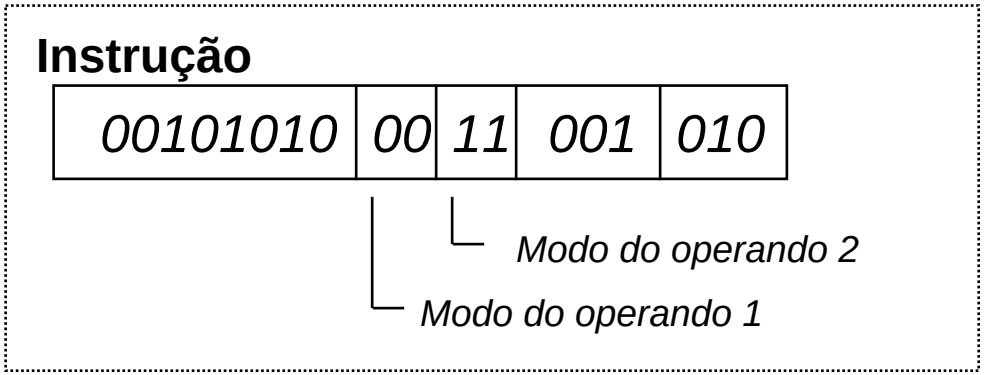


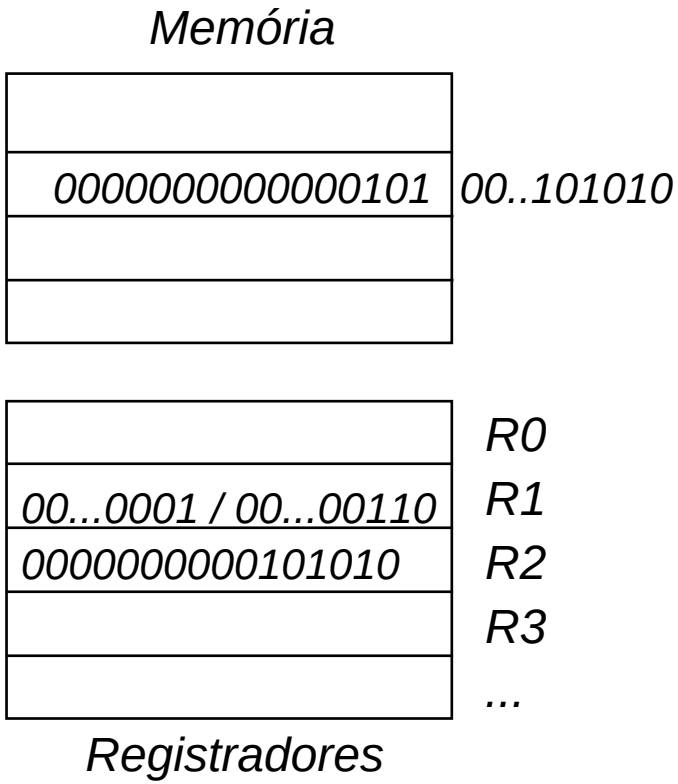
Registradores

- Modo bastante frequente
- Operando constante

Endereçamento indireto

- O endereço (reg. ou memória) contem o endereço do operando
  - ADD R1,(R2)
    - R1 <- R1 + mem(R2)

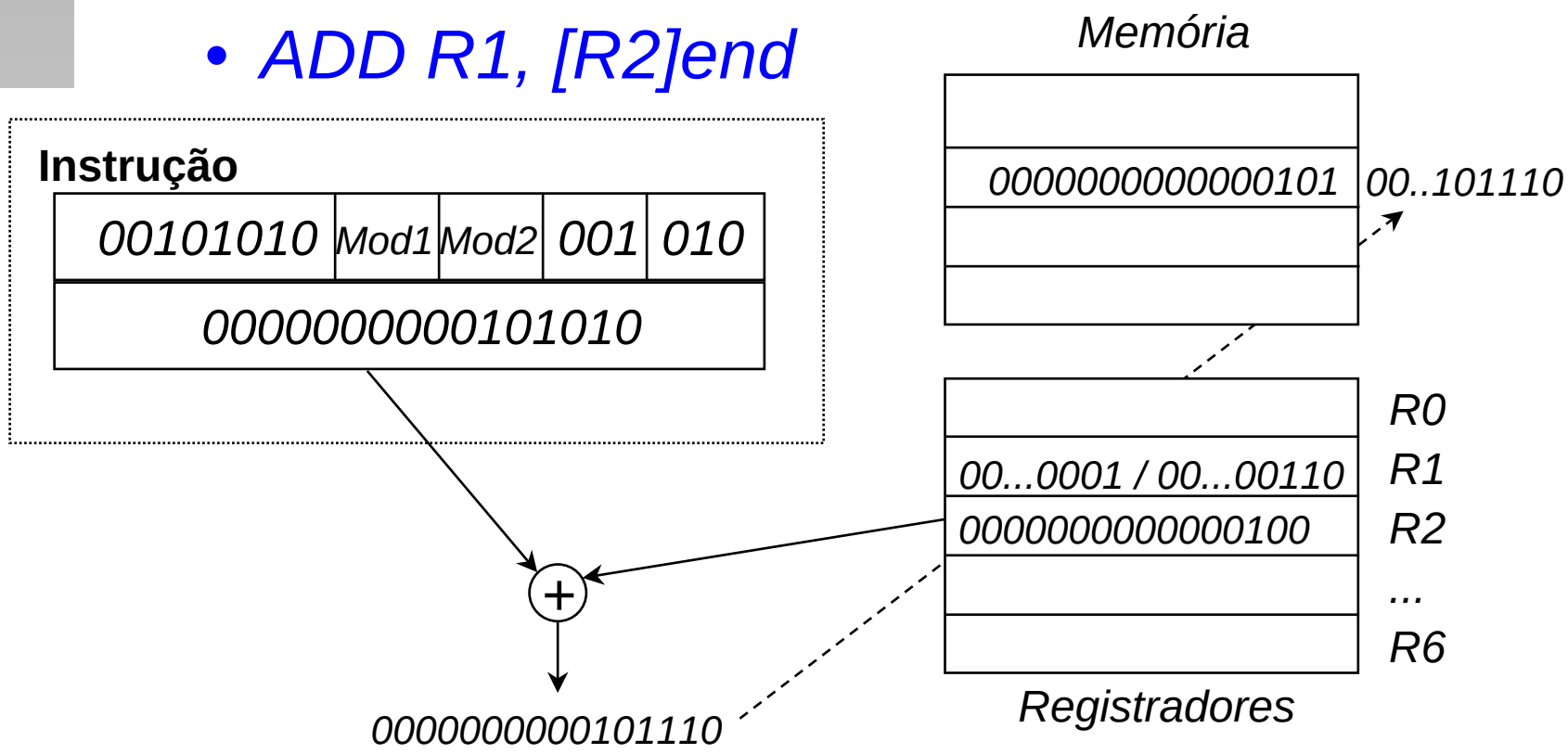




  - Endereço variável
    - ponteiros

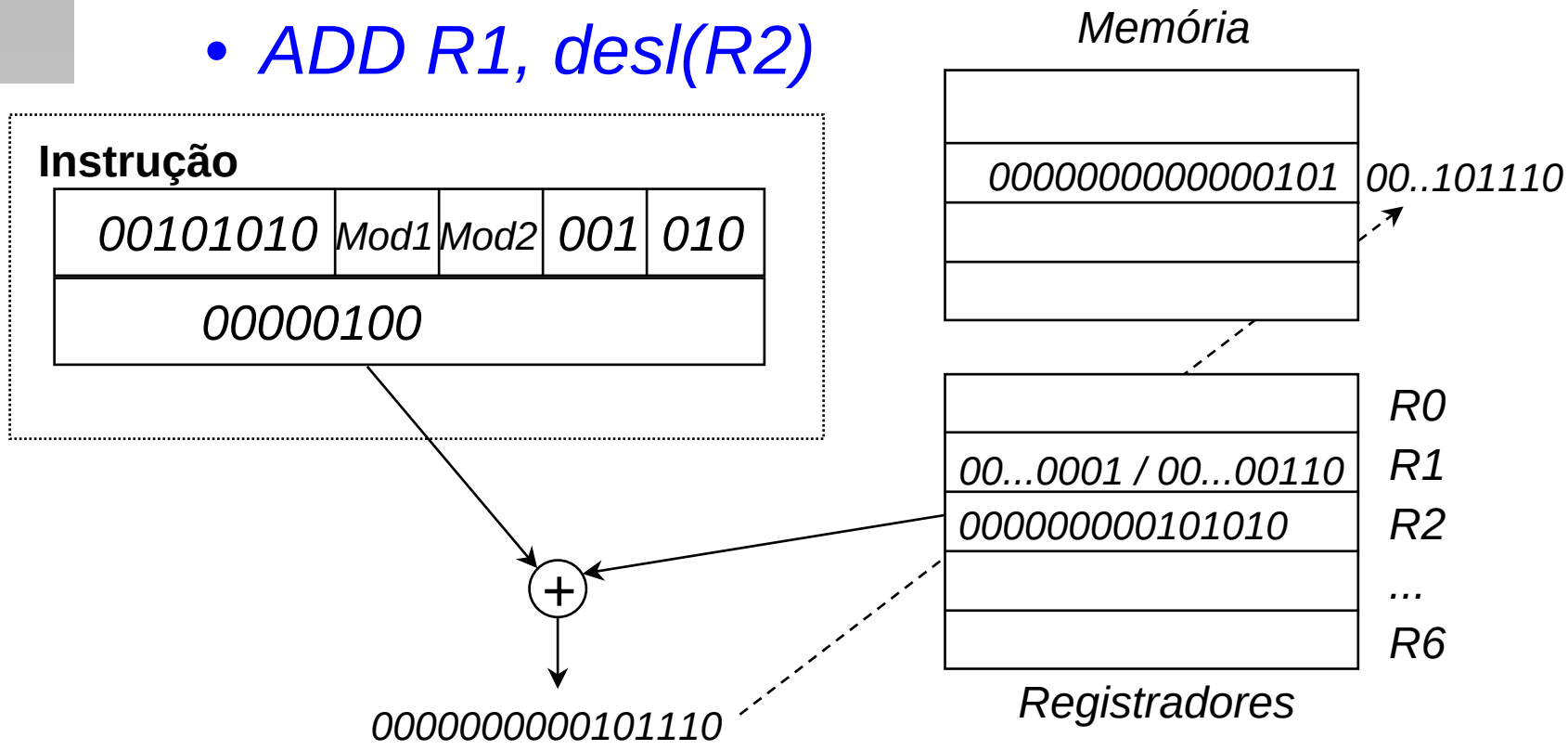
Endereçamento indexado

- Instrução: endereço inicial do array
- Registrador de índice: deslocamento
  - ADD R1, [R2]end



Endereçamento base

- Instrução:deslocamento
- Registrador de base:end- inicial
  - ADD R1, desl(R2)



Modos de Endereçamento

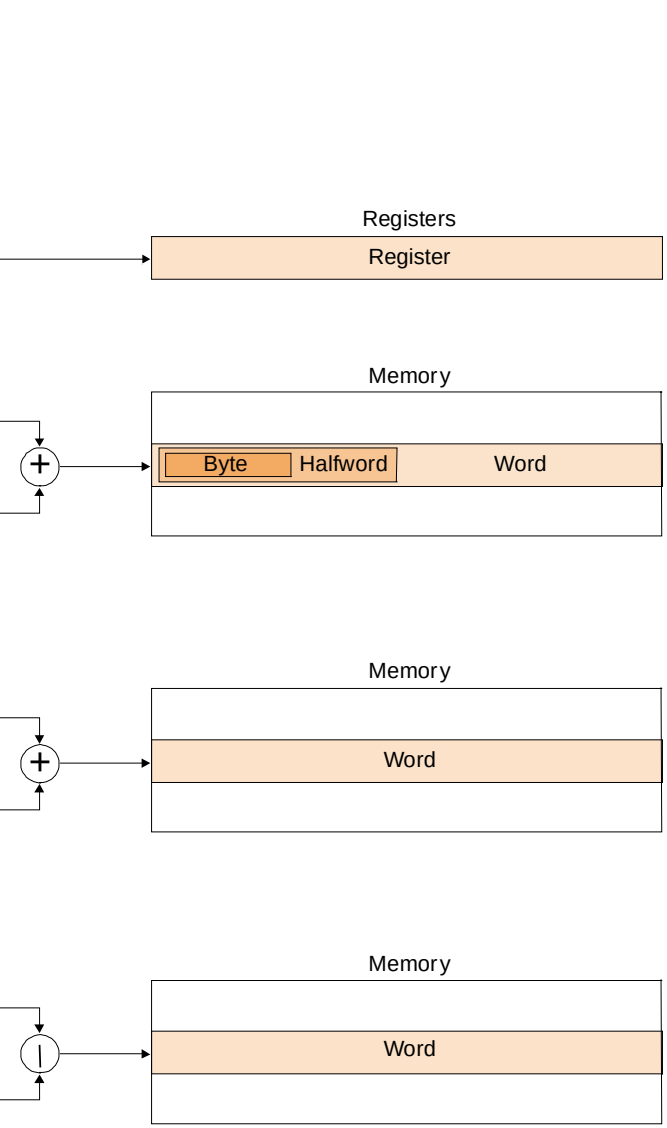
Addressing mode	Example	Meaning
Register	Add R4,R3	R4← R4+R3
Immediate	Add R4,#3	R4 ← R4+3
Displacement	Add R4,100(R1)	R4 ← R4+Mem[100+R1]
Register indirect	Add R4,(R1)	R4 ← R4+Mem[R1]
Indexed / Base	Add R3,(R1+R2)	R3 ← R3+Mem[R1+R2]
Direct or absolute	Add R1,(1001)	R1 ← R1+Mem[1001]
Memory indirect	Add R1,@(R3)	R1 ← R1+Mem[Mem[R3]]
Auto-increment	Add R1,(R2)+	R1 ← R1+Mem[R2]; R2 ← R2+d
Auto-decrement	Add R1,–(R2)	R2 ← R2–d; R1 ← R1+Mem[R2]
Scaled	Add R1,100(R2)[R3]	R1 ← R1+Mem[100+R2+R3*d]

Why Auto-increment/decrement? Scaled?

Endereçamento de desvio

- *Especificação do endereço de desvio:*
  - *Absoluto:*
    - ***PC ⟵ Endereço de Desvio***
    - *restrito a algumas funções do S.O.*
    - *implícito: vetor de interrupções*
  - *Relativo (PC = endereço base):*
    - ***PC ⟵ PC + Deslocamento (instrução)***
    - *permite relocação*
    - *codificação econômica (poucos bits para o deslocamento)*

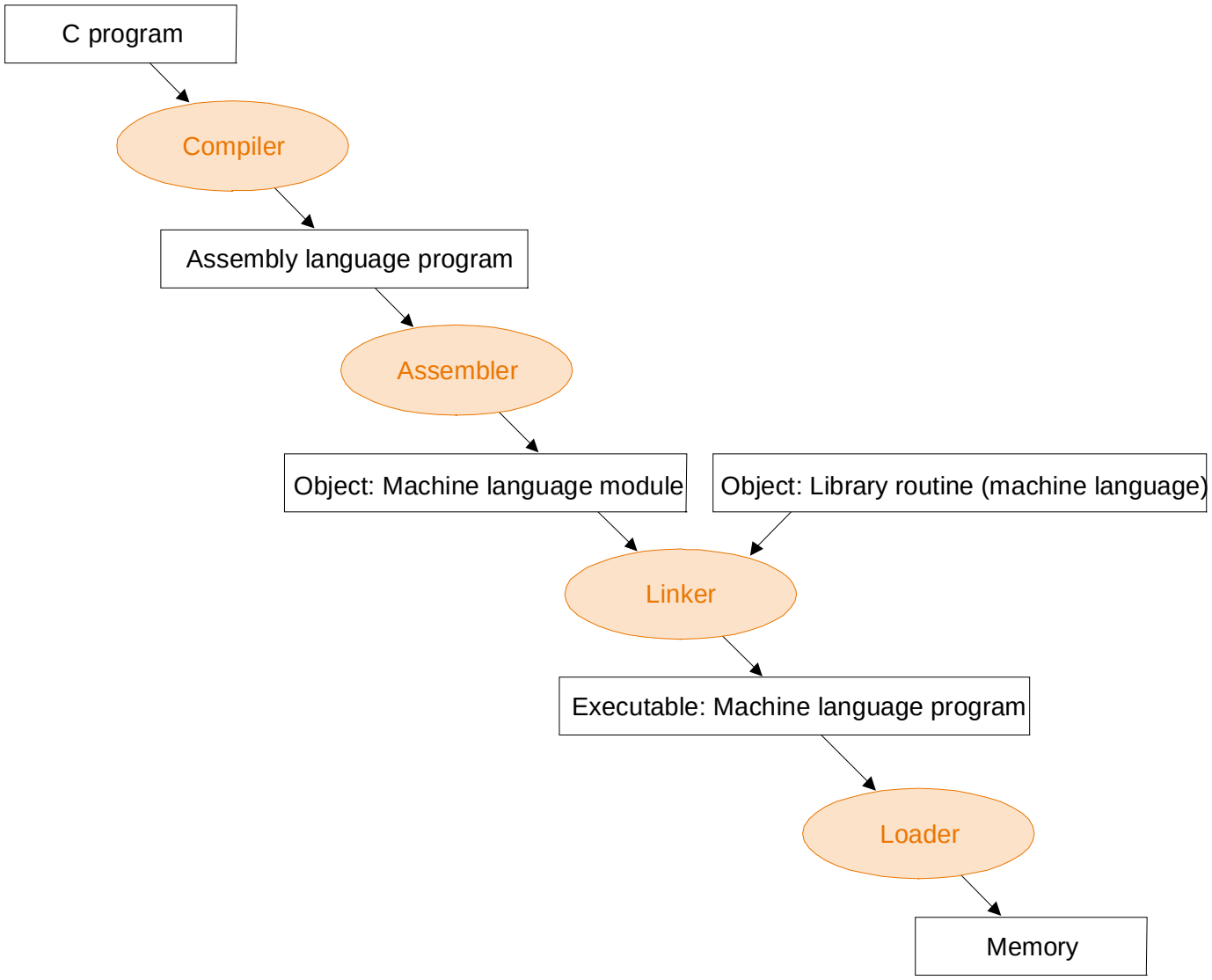
Modos do MIPS



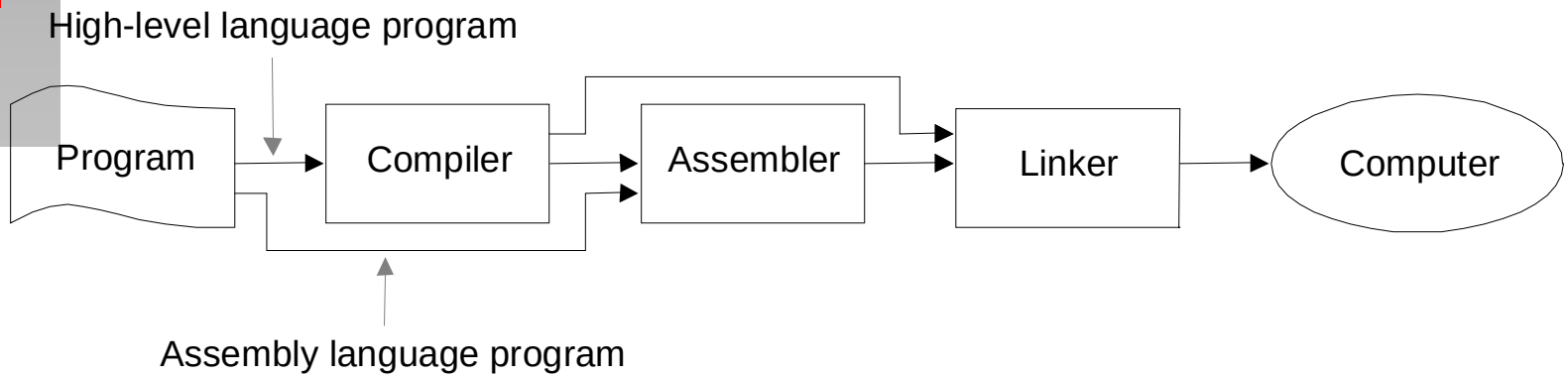
MIPS

Instrução	Descrição
nop	No operation
lw reg, end(reg_base)	reg. = mem (reg_base+end)
sw reg, end(reg_base)	Mem(reg_base+end) = reg
li reg, constante	reg = constante
add regi, regj,regk	Regi. <- Regj. + Regk
addi regi, regj, cte	Regi = regj + cte
sub regi, regj, regk	Regi. <- Regj. – Regk
and regi, regj,regk	Regi. <- Regj. and Regk
andi regi, regj, cte	Regi = regj and cte
shfrl regd, regs, n	Desloca regs para direita n vezes (Lógico) e armazena valor deslocado em regd .
shfra regd, regs, n	Desloca regs para dir. n vezes (aritmético), armazena valor deslocado em regd.
shfl regd, regs, n	Desloca regs para esquerda n vezes, armazena valor deslocado em regd.
rotr regd, regs, n	Rotaciona regs para direita n vezes, armazena valor deslocado em regd.
rotl regd, regs, n	Rotaciona regs para esquerda n vezes, armazena valor deslocado em regd.
beq regi, regj, end	Desvia para end. se regi = regj
bne regi, regj, end	Desvia para end se regi <> regj
slt regi, regj, regk	Regi = 1 se regj < regk senão regi= 0
slti regi, regj, cte	Regi = 1 se regj < cte senão regi= 0
j end	Desvio para end
j regi	PC = (regi)
jal end	R31 = PC; PC = end
break	Para a execução do programa

Executando um programa



Executando um programa



Usando o Simulador MIPSIT

- *Simulador MIPSIT*
  - *Local: P:\\cin04\apps*
  - *Executar mipsit.exe*
  - *Criando um projeto:*
    - *File/new -> criar project*
  - *Editando um programa*
    - *File/new -> criar arquivo e editar texto*

Exemplo de Programa em Linguagem de Montagem

```
.data
Item: .word 5
a:    .word 10

.text
.globl start
.ent start
start: lw $8, item
       lw $9, a
       add $10, $9, $8
      sw $10,a
.end start
```

Usando o Simulador MIPSIT

- Simulador MIPSIT
  - Compilando um programa
    - build -> build
    - Códigos objetos são armazenados na pasta object
  - Carregando código executável para o simulador
    - Executar o simulador mips.exe
    - Carregar usando mipsit:
      - Build/upload -> to simulator

Usando o Simulador MIPSIT

- *Simulando um programa*
  - *O que é visível:*
    - *Registradores (CPU)*
      - *Conteúdo em hexadecimal*
      - *Associação nome e número*
    - *Memória (RAM)*
      - *Quatro colunas:*
        - *Endereço*
        - *Conteúdo (hexa)*
        - *Rótulos*
        - *Instrução de máquina*

Usando o Simulador MIPSIT

- *Simulando um programa*
  - *Iniciando a execução:*
    - *Indo para rótulo inicial*
      - *Jump To Symbol -> start*
  - *Executando passo a passo*
      - *CPU -> step*
      - *Botao: seta para quardrado azul*
- *Para usar nomes simbólicos:*
  - *Inclua no programa em linguagem de montagem: #include ,iregdef.h>*

Resumindo

- *Organização de um computador*
- *Definição de arquitetura*
  - *Tipos de Dados*
    - *Inteiros*
    - *Booleanos*
    - *Ponto-Flutuante*
  - *Formato das instruções*
  - *Conjunto de registradores*

Resumindo

- *...Definição de Arquitetura*
  - *Repertório de instruções*
    - *sobre o dado*
      - *movimentação*
      - *transformação*
      - *codificação*
      - *aritméticas*
      - *lógicas*
    - *alteração do fluxo de execução*
      - *desvios condicionais*
      - *desvios incondicionais*
      - *Subrotinas*

Resumindo

- *...Definição de Arquitetura*
  - *Modos de Endereçamento: Dados*
    - *Operações Aritméticas e de Comparação:*
      - *Registrador*
      - *Imediato*
    - *Load/Store:*
      - *Base*
  - *Modos de Endereçamento: Instruções*
    - *Desvio Incondicional*
      - *(pseudo) direto*
      - *Indireto de registrador*
    - *Desvio Condicional*
      - *Relativo ao PC*
    - *Subrotina*
      - *(pseudo) direto*

This document was created with Win2PDF available at http://www.daneprairie.com.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.