Organização e Performance de Processadores

Introdução

- Questão de desenho do conjunto de instruções
  - ✓ Número de endereços
  - ✓ Modos de endereçamento
  - ✓ Tipos de instrução
  - ✓ Formatos de instrução
- Controlo microprogramado
  - ✓ Implementação em software ou hardware
- Questões de performance

Número de Endereços

- Máquinas de 3 endereços
  - ✓ Dois para operandos
  - ✓ Um para o resultado
  - ✓ Usado nos processadores RISC
  - ✓ Exemplos:

```
add     dest,src1,src2
        ; M[dest]=[src1]+[src2]
sub     dest,src1,src2
        ; M[dest]=[src1]-[src2]
mult    dest,src1,src2
        ; M[dest]=[src1]*[src2]
```

Número de Endereços

- Exemplo
  - ✓ Código C:

```
A = B + C * D - E + F + A
```

  - ✓ Código assembly:

```
mult    T,C,D   ; T = C*D
add     T,T,B   ; T = B+C*D
sub     T,T,E   ; T = B+C*D-E
add     T,T,F   ; T = B+C*D-E+F
add     A,T,A   ; A = B+C*D-E+F+A
```

Número de Endereços

- Máquinas de 2 endereços
  - ✓ Um dos operandos serve também para guardar o resultado
  - ✓ Usado nos processadores Intel
  - ✓ Exemplos:

```
mov     dest,src ; M[dest]=[src]
add     dest,src ; M[dest]=[dest]+[src]
sub     dest,src ; M[dest]=[dest]-[src]
mult    dest,src ; M[dest]=[dest]*[src]
```

Número de Endereços

- Exemplo
  - ✓ Código C:

```
A = B + C * D - E + F + A
```

  - ✓ Código assembly:

```
mov     T,C       ; T = C
mult    T,D       ; T = C*D
add     T,B       ; T = B+C*D
sub     T,E       ; T = B+C*D-E
add     T,F       ; T = B+C*D-E+F
add     A,T       ; A = B+C*D-E+F+A
```

Número de Endereços

- Máquinas de 1 endereço
  - ✓ Apenas se denota um operador
  - ✓ Utilizam de forma implícita registos especiais nomeados acumuladores
    - ✓ Maquinas de acumulador
  - ✓ O MARIE é uma máquina de acumulador
  - ✓ Exemplos:

```
load    addr ; accum = [addr]
store   addr ; Mem[addr] = accum
add     addr ; accum = accum + [addr]
sub     addr ; accum = accum - [addr]
mult    addr ; accum = accum * [addr]
```

Número de Endereços

- Exemplo
  - ✓ Código C:

```
A = B + C * D - E + F + A
```

  - ✓ Código assembly:

```
load    C   ; Carregar C para accum
mult    D   ; accum = C*D
add     B   ; accum = C*D+B
sub     E   ; accum = B+C*D-E
add     F   ; accum = B+C*D-E+F
add     A   ; accum = B+C*D-E+F+A
store   A   ; Guardar o conteúdo de accum
            ; em A
```

Número de Endereços

- Máquinas sem endereços
  - ✓ Utiliza implícitamente uma pilha de avaliação de expressões
  - ✓ Máquinas de pilha
  - ✓ A JVM é uma máquina de pilha
  - ✓ Exemplos:

```
push    addr  ; push([addr])
pop     addr  ; Mem[addr] = pop()
add           ; push(pop() + pop())
sub           ; push(pop() - pop())
mult          ; push(pop() * pop())
```

Número de Endereços

- Exemplo
  - ✓ Código C:

```
A = B + C * D - E + F + A
```

  - ✓ Código assembly:

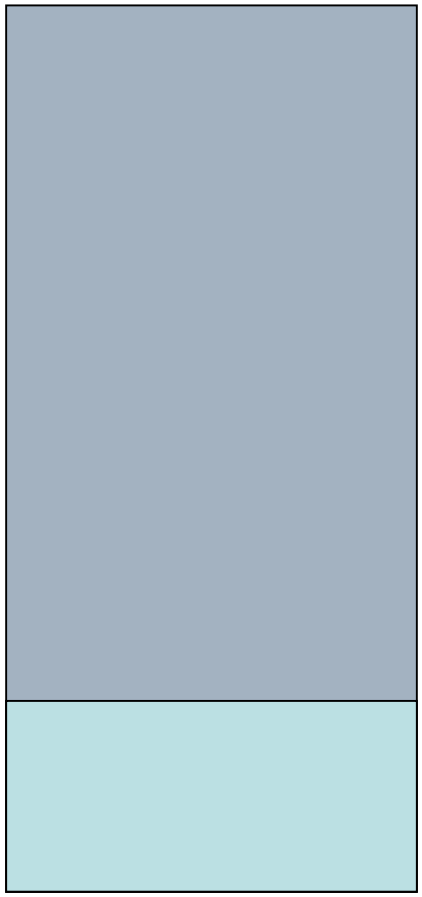
```
push    E          |   sub
push    C          |   push    F
push    D          |   add
mult               |   push    A
push    B          |   add
add                |   pop     A
```

Funcionamento de uma máquina de pilha

- Exemplo de avaliação de uma expressão:
  (3 + 5) * - 4 + 2

- Usando notação pós-fixa:
  3 5 + 4 - * 2 +

- Avaliando usando uma pilha:
  push 3
  push 5
  add
  push 4
  change Sign
  mul
  push 2
  add
  pop resultado



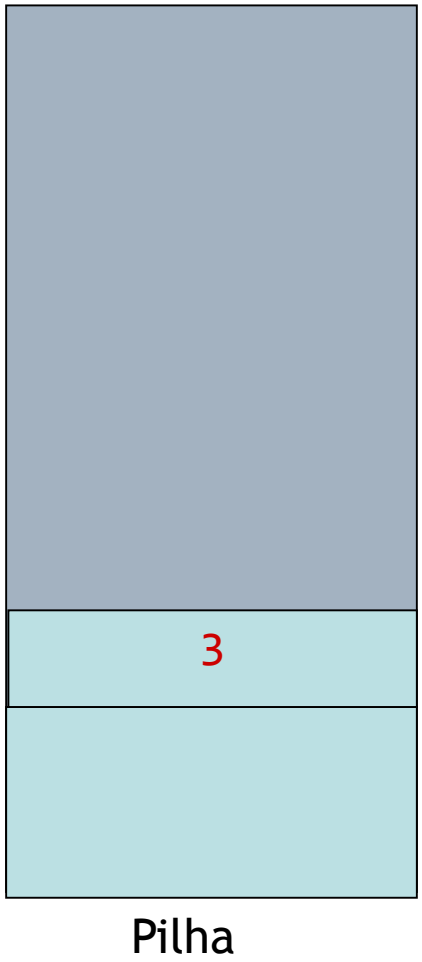
Pilha

Funcionamento de uma máquina de pilha

- Exemplo de avaliação de uma expressão:
  (3 + 5) * - 4 + 2

- Usando notação pós-fixa:
  3 5 + 4 - * 2 +

- Avaliando usando uma pilha:
  push 3
  push 5
  add
  push 4
  change Sign
  mul
  push 2
  add
  pop resultado



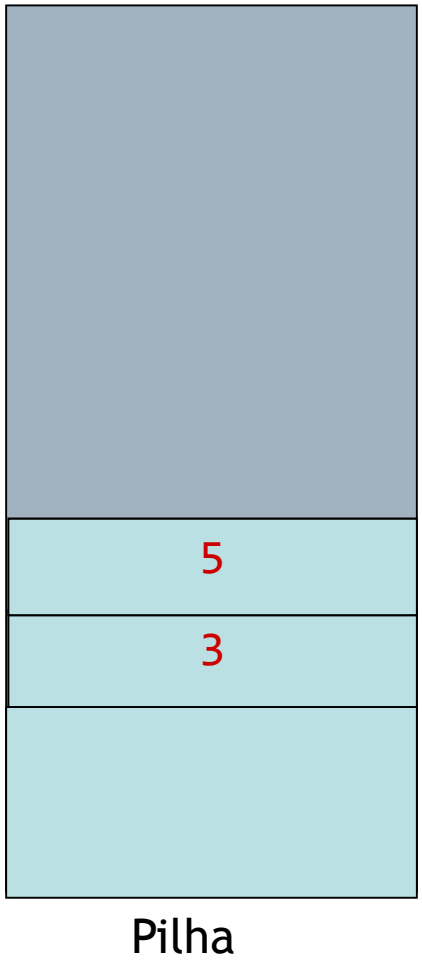
Pilha

Funcionamento de uma máquina de pilha

- Exemplo de avaliação de uma expressão:
  (3 + 5) * - 4 + 2

- Usando notação pós-fixa:
  3 5 + 4 - * 2 +

- Avaliando usando uma pilha:
  push 3
  push 5
  add
  push 4
  change Sign
  mul
  push 2
  add
  pop resultado



Pilha

Funcionamento de uma máquina de pilha

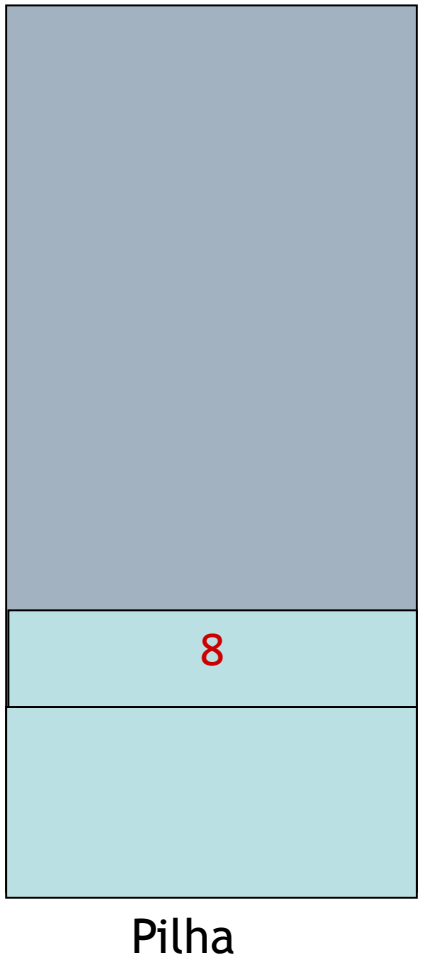
- Exemplo de avaliação de uma expressão:
  (3 + 5) * - 4 + 2

- Usando notação pós-fixa:
  3 5 + 4 - * 2 +

- Avaliando usando uma pilha:

push 3
push 5
add
push 4
change Sign
mul
push 2
add
pop resultado

Tira dois operandos da pilha e coloca lá o resultado



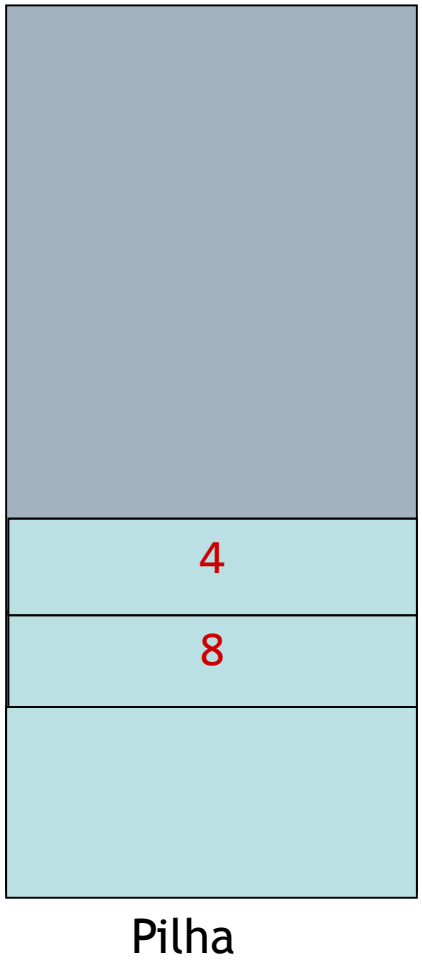
Pilha

Funcionamento de uma máquina de pilha

- Exemplo de avaliação de uma expressão:
  (3 + 5) * - 4 + 2

- Usando notação pós-fixa:
  3 5 + 4 - * 2 +

- Avaliando usando uma pilha:
  push 3
  push 5
  add
  push 4
  change Sign
  mul
  push 2
  add
  pop resultado



Pilha

Funcionamento de uma máquina de pilha

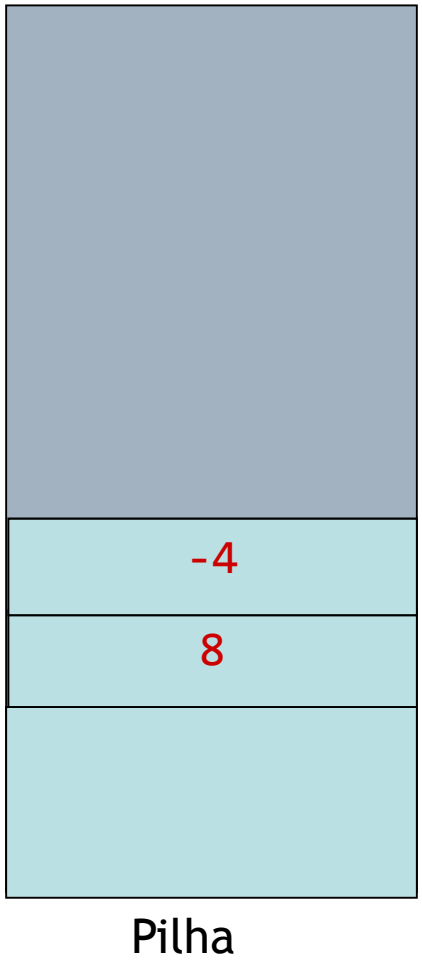
- Exemplo de avaliação de uma expressão:
  (3 + 5) * - 4 + 2

- Usando notação pós-fixa:
  3 5 + 4 - * 2 +

- Avaliando usando uma pilha:

push 3
push 5
add
push 4
change Sign
mul
push 2
add
pop resultado

Tira um operando da, o 4, pilha e coloca lá o resultado: -4



Funcionamento de uma máquina de pilha

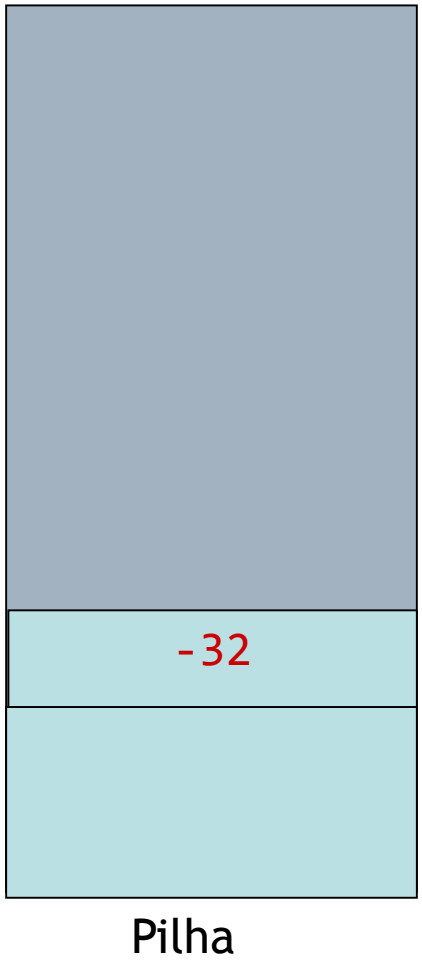
- Exemplo de avaliação de uma expressão:
  (3 + 5) * - 4 + 2

- Usando notação pós-fixa:
  3 5 + 4 - * 2 +

- Avaliando usando uma pilha:

push 3
push 5
add
push 4
change Sign
mul
push 2
add
pop resultado

Tira dois operandos da pilha e coloca lá o resultado



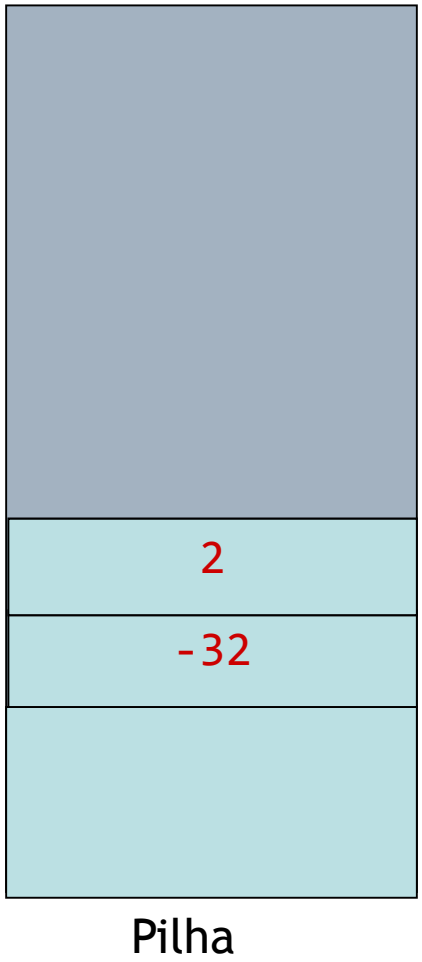
Pilha

Funcionamento de uma máquina de pilha

- Exemplo de avaliação de uma expressão:
  (3 + 5) * - 4 + 2

- Usando notação pós-fixa:
  3 5 + 4 - * 2 +

- Avaliando usando uma pilha:
  push 3
  push 5
  add
  push 4
  change Sign
  mul
  push 2
  add
  pop resultado



Pilha

Funcionamento de uma máquina de pilha

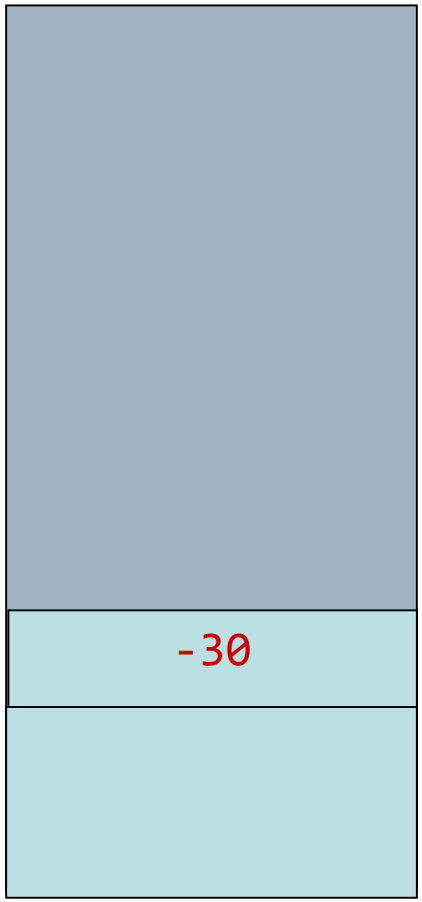
- Exemplo de avaliação de uma expressão:
  (3 + 5) * - 4 + 2

- Usando notação pós-fixa:
  3 5 + 4 - * 2 +

- Avaliando usando uma pilha:

push 3
push 5
add
push 4
change Sign
mul
push 2
add
pop resultado

O resultado final está no topo da pilha



Pilha

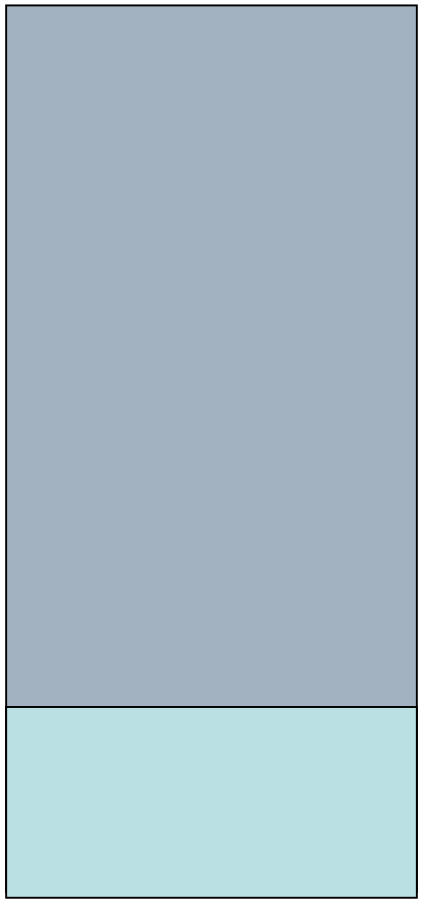
Funcionamento de uma máquina de pilha

- Exemplo de avaliação de uma expressão:
  (3 + 5) * - 4 + 2

- Usando notação pós-fixa:
  3 5 + 4 - * 2 +

- Avaliando usando uma pilha:
  push 3
  push 5
  add
  push 4
  change Sign
  mul
  push 2
  add
  pop resultado

O resultado final está no topo da pilha

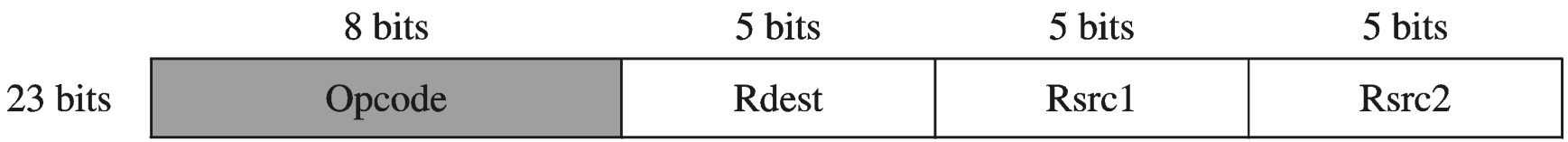


Pilha

Número de Endereços - Comparação

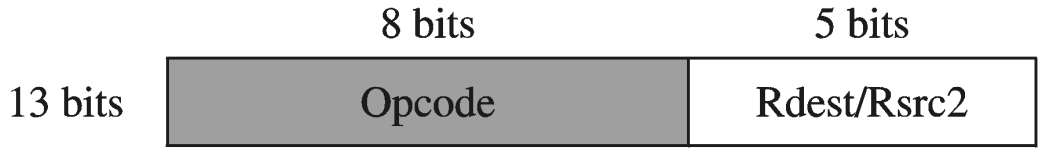
- Menos endereços resulta num código com mais instruções, o que normalmente resulta em mais acessos a memória
  - ✓ Código com 3 endereços
    - ✓ Codificação do exemplo precisa de 5 instruções
    - ✓ 4 acessos a memória por instrução → 20 acessos
    - ✓ Podemos optimizar esse número recorrendo a registos
      - ✓ Suponhamos que T passa a ser um registo em vez de um posição de memória → 12 acessos
  - ✓ Código com zero endereços
    - ✓ Codificação do exemplo precisa de 12 instruções
    - ✓ 7 destas lêem uma posição de memória
    - ✓ 19 acessos
- Mais endereços resulta em instruções mais compridas

Número de Endereços - Comparação

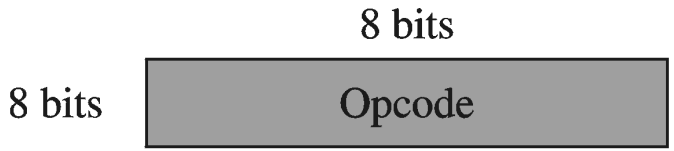


3-address format

2-address format



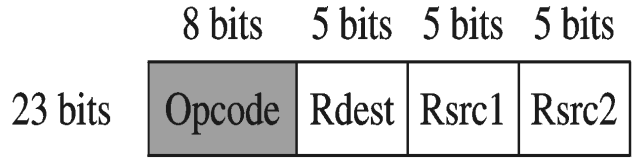
1-address format



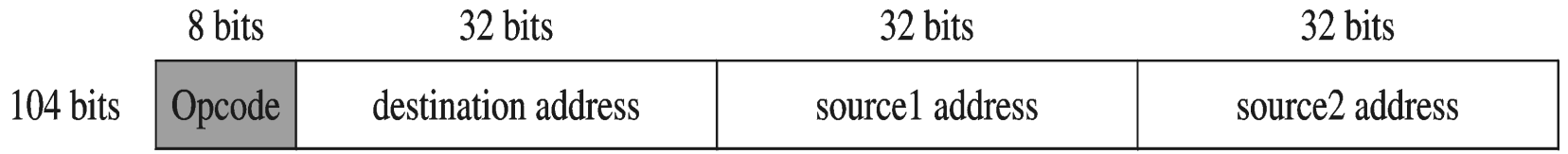
0-address format

Arquitecturas LOAD/STORE

- As únicas instruções que operam sobre memória são o LOAD e o STORE
- As restantes operam sobre registos
- Utilizado em arquitecturas RISC e vectoriais
- Reduz o comprimento da instrução
- Exemplo com endereços de 32 bits e 32 registos



Register format



Memory format

Arquitecturas LOAD/STORE

- Exemplos de instruções

```
load    Rd,addr      ; Rd = [addr]
store   addr,Rs      ; Mem[addr] = Rs
add     Rd,Rs1,Rs2   ; Rd = Rs1 + Rs2
sub     Rd,Rs1,Rs2   ; Rd = Rs1 - Rs2
mult    Rd,Rs1,Rs2   ; Rd = Rs1 * Rs2
```

Arquitecturas LOAD/STORE

- Exemplo
  - ✓ Código C:

```
A = B + C * D – E + F + A
```

  - ✓ Código assembly:

```
load    R1,B          mult    R2,R2,R3
load    R2,C          add     R2,R2,R1
load    R3,D          sub     R2,R2,R4
load    R4,E          add     R2,R2,R5
load    R5,F          add     R2,R2,R6
load    R6,A          store   A,R2
```

Fluxo de Execução

- Por omissão o fluxo de execução é sequencial

- No entanto, existem instruções que alteram este comportamento
  - ✓ Modificam o valor do *Program Counter*
  - ✓ Saltos
    - ✓ Incondicionais
    - ✓ Condicionais
    - ✓ Com atraso
  - ✓ Chamadas a subrotinas
    - ✓ Com atraso

Fluxo de Execução – Saltos Incondicionais

- Salto para endereço absoluto
  - ✓ jump target → PC = target
  - ✓ Exemplos:
    - ✓ NASM IA-32 → jump 1000
    - ✓ MIPS → j 1000

- Salto relativo ao PC
  - ✓ jump target → PC = PC + target
  - ✓ A vantagem é que o código é recolocável
    - ✓ Pode ser colocado em qualquer posição na memória
  - ✓ Exemplo:
    - ✓ MIPS → b 1000

Fluxo de Execução – Saltos Condicionais

- A execução do salto depende da avaliação de uma condição
- Existem duas abordagens:
  - ✓ Set-then-Jump
    - ✓ A avaliação da condição está separada do salto
    - ✓ Registos de condição guardam o resultado da avaliação
    - ✓ Usado no IA-32 e AMD64/Intel64. Exemplo:

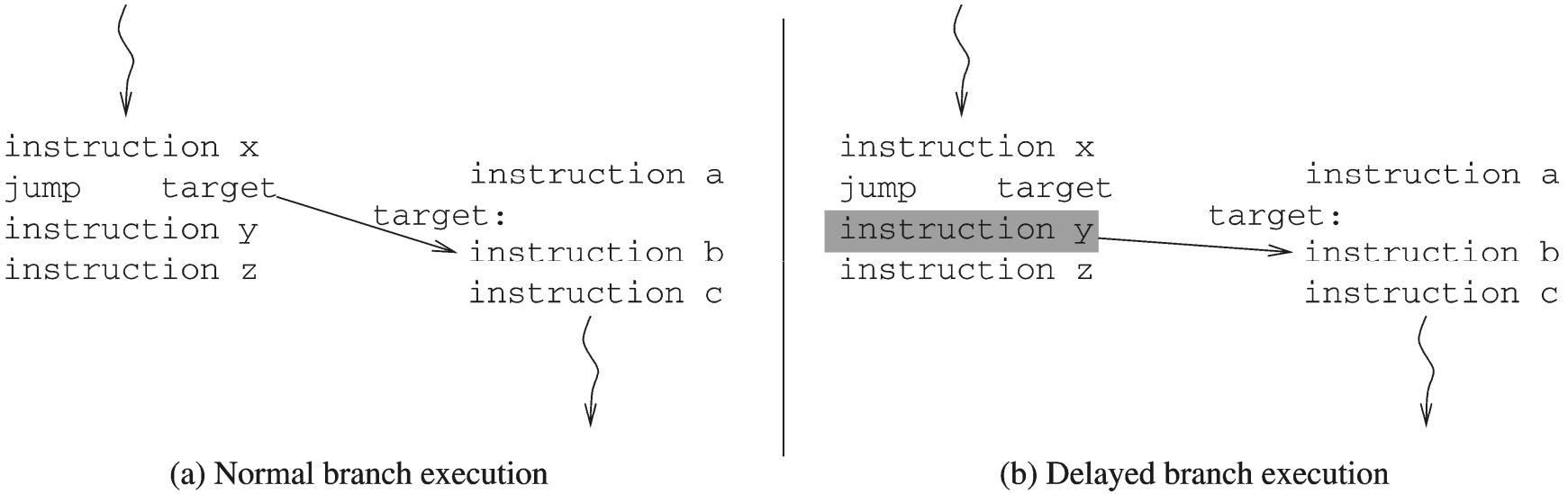
```
cmp     AX,BX
je      target ; salta se AX == BX
```

  - ✓ Test-then-Jump
    - ✓ Uma única instrução para testar a condição e saltar
    - ✓ Usado no MIPS. Exemplo:

```
beq     R1,R2,target ; salta se R1 == R2
```

Fluxo de Execução – Saltos com Atraso

- A transferência de controlo só é efectuada depois da execução da instrução a seguir ao salto



- Útil em processadores com pipelining
  - ✓ Quando se executa o jump a `instrução y` já foi carregada da memória
- Podemos ter vários slots de atraso para instruções

Fluxo de Execução - Subrotinas

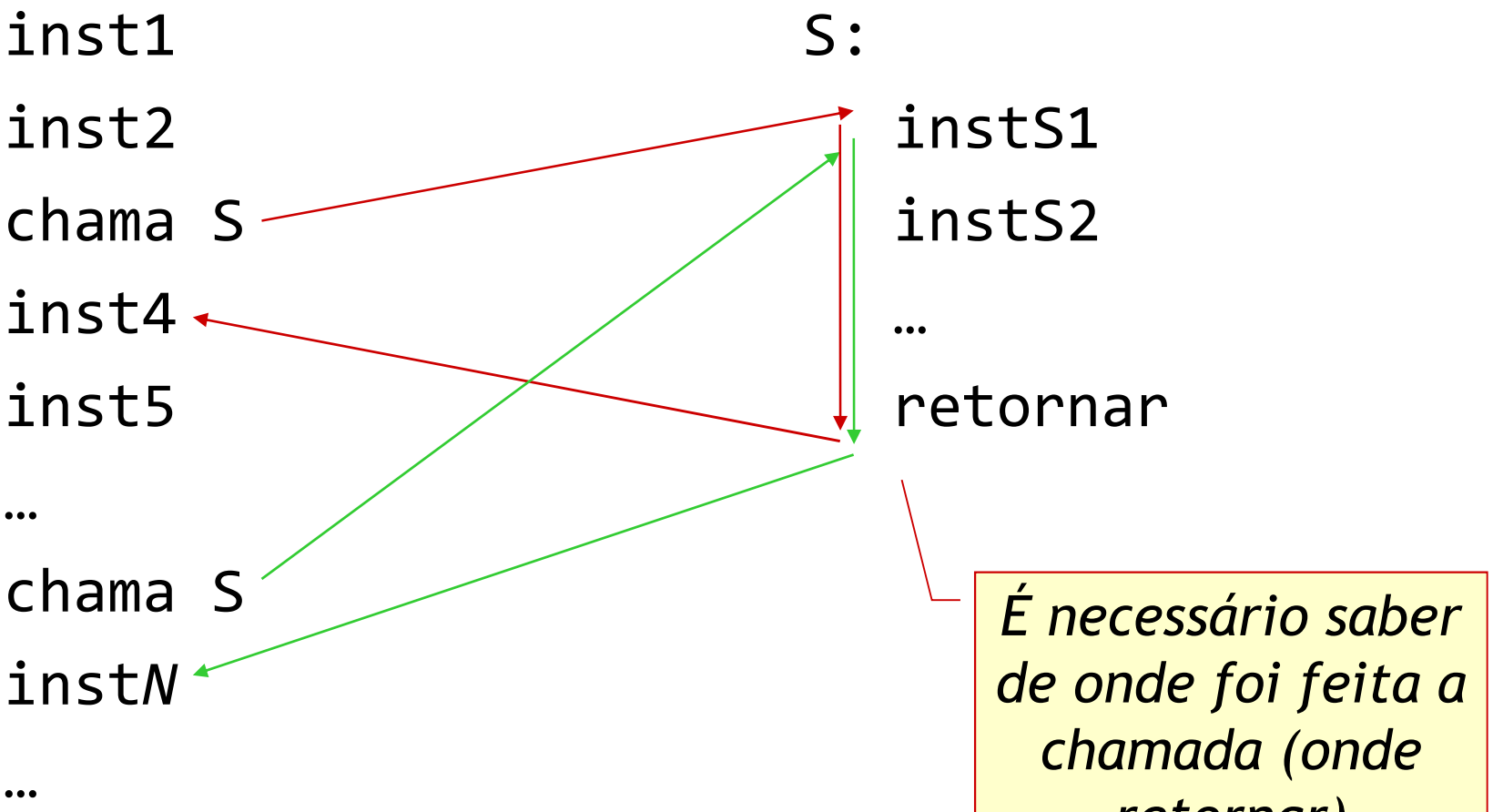
- Subrotina: uma sequência de instruções, que pode ser chamada múltiplas vezes, a partir de diferentes pontos de um programa

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

Vantagens das Subrotinas

- Permite implementar procedimentos e funções ou métodos
  - ✓ Estruturação do programa:
    - ✓ Maior clareza e modularidade…
    - ✓ Desenvolvimento e teste incrementais…
  - ✓ Redução do tamanho do programa:
    - ✓ Evita a repetição de sequências muito utilizadas

Subrotinas – Chamada e Retorno



Subrotinas – Chamada e Retorno

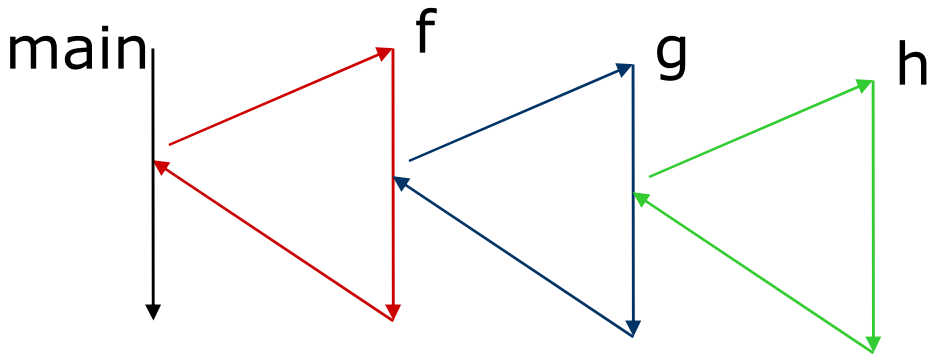
- Requer uma instrução para a chamada
  - ✓ Não é igual a um salto, pois um salto só altera o PC
  - ✓ Numa chamada a uma subrotina não basta saltar, há que saber regressar
    - ✓ A chamada guarda o valor actual do PC e depois salta para o endereço dado (tal como um salto)
  - ✓ Exemplo IA-32: call target

- Requer uma outra instrução para regressar
  - ✓ Salta para o endereço guardado
    - ✓ No fim da subrotina repõe o PC, saltando para o endereço que a chamada guardou
  - ✓ Exemplo IA-32: ret

Subrotinas – Endereço de Retorno

- Como salvaguardar o endereço de retorno:
  - ✓ Numa célula pré-definida de memória?
  - ✓ Num registo dedicado do CPU?
  - ✓ Numa estrutura de dados especial em memória?

- Podem existir chamadas em cascata e recursivas
  - ✓ exemplo: main() → f() → g() → h()



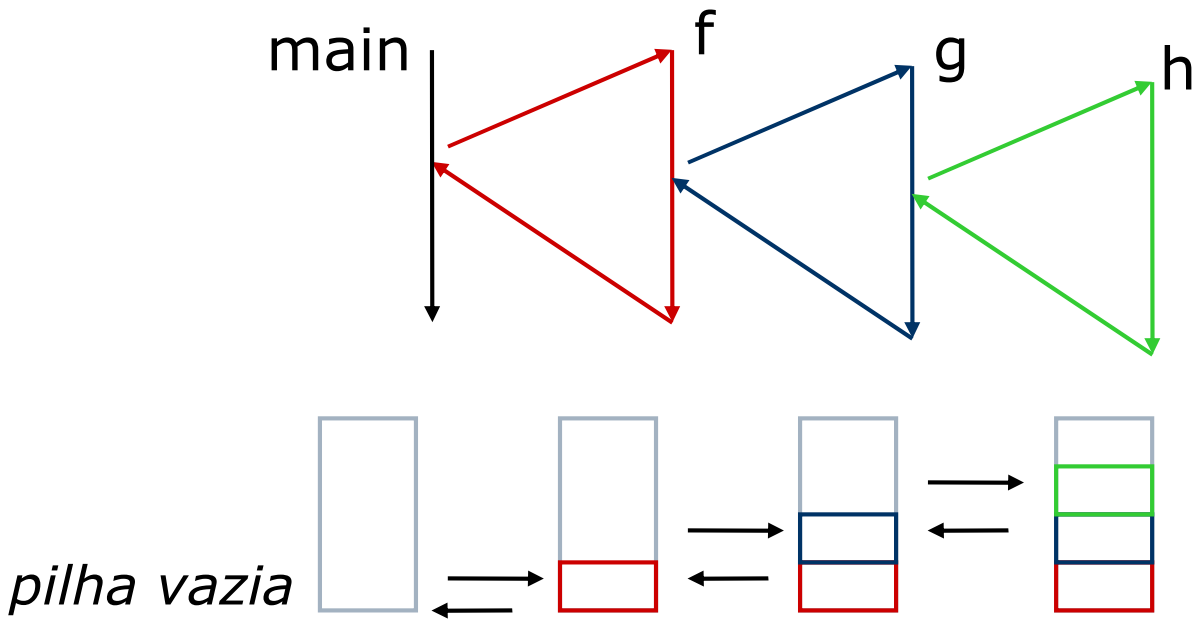
Subrotinas – Endereço de Retorno

- A solução exige manter todos os endereços das chamadas activas
  - ✓ Registos ou células de memória pré-definidas é limitado (ou impraticável)
- Usa-se uma estrutura de dados em pilha
  - ✓ Permite que se empilhe sempre mais um endereço, ficando no topo o último empilhado
  - ✓ No IA-32 a própria chamada à subrotina guarda o endereço de retorno na pilha
  - ✓ Em processadores com ISA RISC, como o MIPS, a chamada guarda o endereço de retorno num registo
    - ✓ É da responsabilidade do programador guardá-lo na pilha, caso o queira

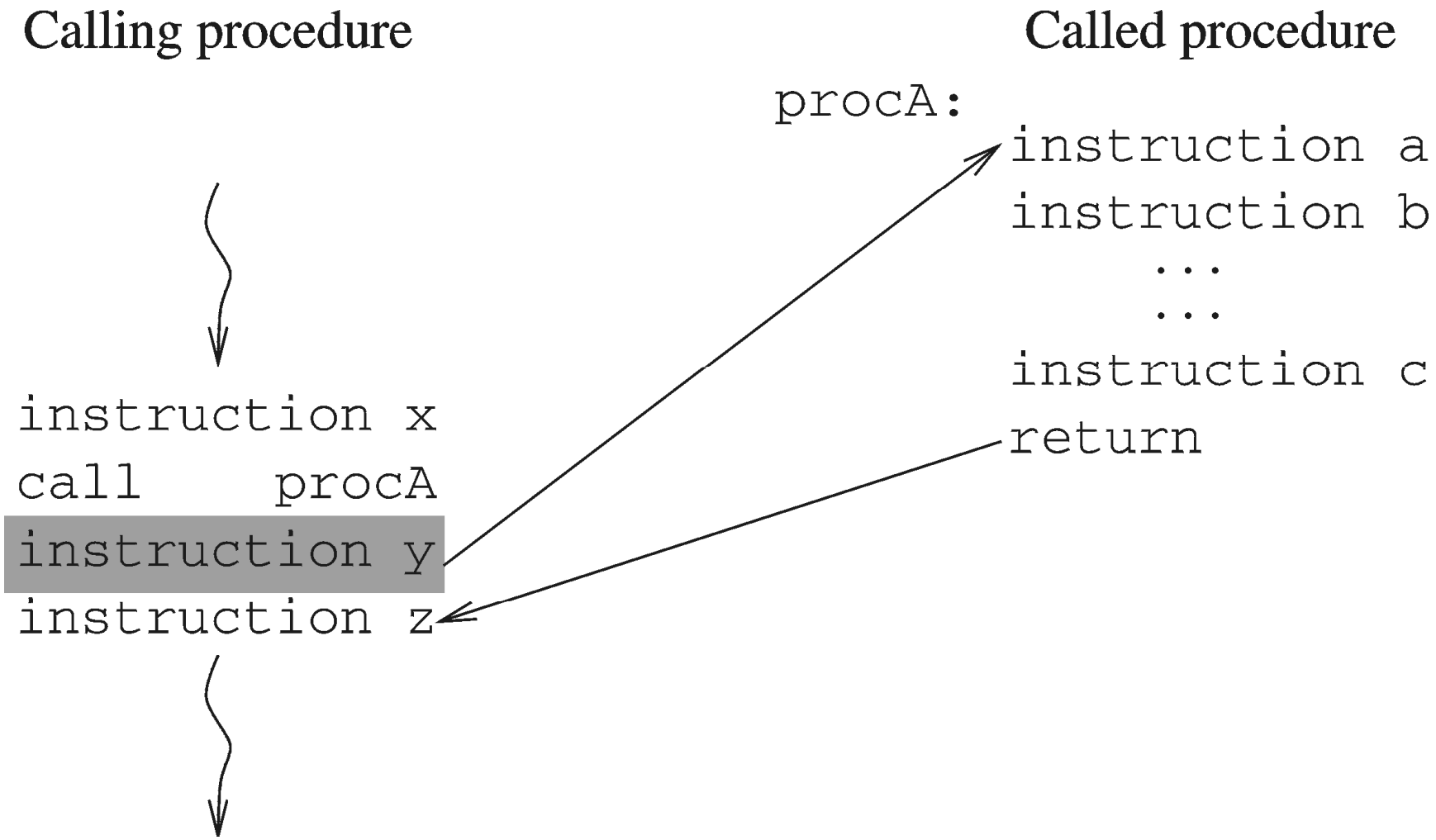
Subrotinas – Endereço de Retorno

- A solução exige manter todos os endereços das chamadas activas

- Usa-se uma estrutura de dados em pilha



Subrotinas – Chamadas com Atraso



Subrotinas – Passagem de Argumentos

- Duas técnicas base
  - ✓ Baseada em registos
    - ✓ Utiliza um conjunto de registos
    - ✓ Mais rápido pois não precisa de aceder a memória
    - ✓ Limita o número de argumentos
    - ✓ Abordagem utiliza sempre que possível no MIPS e no PowerPC
  - ✓ Baseada na pilha
    - ✓ Utiliza a pilha
    - ✓ Mais geral
    - ✓ Abordagem normalmente seguida no IA-32
- Janelas de registos
  - ✓ Técnica mais recente usada pelo SPARC e Itanium (IA-64)

Tipo de Operandos

- Tipos básicos
  - ✓ Caracteres
    - ✓ Em arquitecturas com memória endereçável ao byte não requer qualquer tipo de tratamento extra
  - ✓ Números inteiros
  - ✓ Números em vírgula flutuante

- *Overloading* de instruções
  - ✓ Exemplo: NASM/IA-32

```
mov     AL,address   ; load de um valor em 8 bits
mov     AX,address   ; load de um valor em 16 bits
mov     EAX,address  ; load de um valor em 32 bits
```

Tipo de Operandos

- Instruções dedicadas
  - ✓ Exemplo: MIPS

```
lb     Rdest,address  ; load de um byte
lh     Rdest,address  ; load de uma halfword
                      ; (16 bits no MIPS)
lw     Rdest,address  ; load de uma word
                ; (32 bits no MIPS)
ld     Rdest,address  ; load de uma doubleword
                      ; (64 bits no MIPS)
lbu     Rdest,address ; load sem sinal de um byte
```

- Load com sinal estende o sinal do valor até à capacidade do registo
- Load sem sinal preenche o resto do registo com zeros

Modos de Endereçamento

- Podem ser colocados em 3 sítios
  - ✓ Registos
    - ✓ Endereçamento por registo
  - ✓ Instrução
    - ✓ Endereçamento imediato
      - ✓ Constantes
  - ✓ Memória
    - ✓ Endereçamento directo
      - ✓ O endereço de memória é codificado na instrução
    - ✓ Endereçamento indirecto
      - ✓ O sítio onde está o endereço é codificado na instrução

Modos de Endereçamento

- RISC
  - ✓ Segue o modelo LOAD/STORE
  - ✓ Suportam normal 2 tipos de endereçamento indirecto a memória:
    - ✓ endereço ← registo + constante
      - ✓ Exemplo: `lb R2, 10(R1)  ; R2 ← Mem[R1+10]`
    - ✓ endereço ← registo + registo
      - ✓ Exemplo: `lb R2, R3(R1)  ; R2 ← Mem[R1+R3]`
- CISC
  - ✓ Suporta muito modos de endereçamento indirecto a memória em quase todas as instruções
  - ✓ Forma geral para endereçamento indirecto no IA-32: [Rb+Ri*S+D], com S e D constantes
    - ✓ Exemplo: add [EAX+EBX*2+10], 4

Tipos de Instrução

- Movimento de dados
  - ✓ Directo
    - ✓ IA-32: `mov dest, src`
  - ✓ Indirecto

```
add  Rdest, Rsrc, 0 ; Rdest = Rsrc+0
```

- Aritméticas e lógicas
  - ✓ Aritméticas
    - ✓ Inteiro e vírgula flutuante, com e sem sinal
    - ✓ **`add, sub, mult, div, …`**
  - ✓ Lógicas
    - ✓ **`and, or, not, xor, …`**

Tipos de Instrução - Aritméticas e Lógicas

- Alteram implícita ou explicitamente os bits de condição
  - ✓ S: Sign bit (0 = +, 1= -)
  - ✓ Z: Zero bit (0 = nonzero, 1 = zero)
  - ✓ O: Overflow bit (0 = no overflow, 1 = overflow)
  - ✓ C: Carry bit (0 = no carry, 1 = carry)
- As instruções aritméticas e lógicas do IA-32 alteram implícitamente os bits de condição
  - ✓ Atenção: o **`mov`** não altera os bits de condição
- No PowerPC o programador tem de utilizar instruções especiais
  - ✓ **`add`** - Não altera os bits de condição
  - ✓ **`addcc`** - Altera os bits de condição

Tipos de Instrução

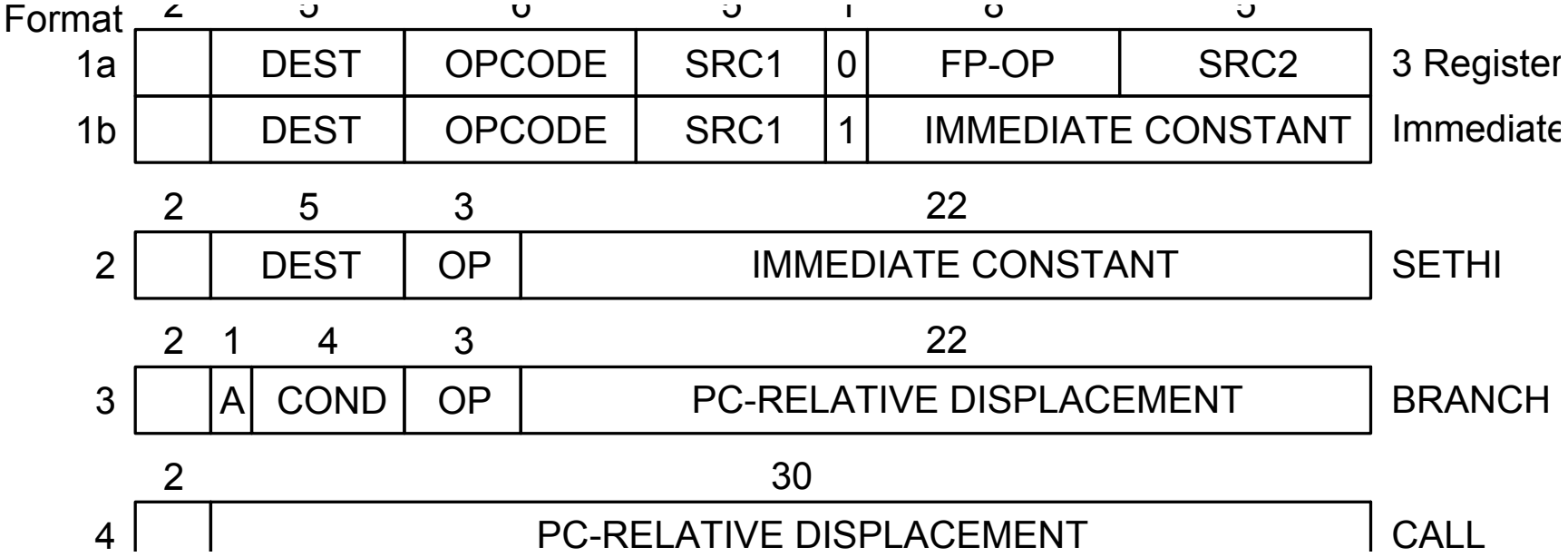
- Fluxo de controlo
  - ✓ Salto
  - ✓ Subrotinas
  - ✓ Interrupções por software

- Programação de Entradas/Saídas
  - ✓ Memory-mapped I/O
    - ✓ Instruções usais de acesso a memória
  - ✓ Espaço próprio
    - ✓ Instruções próprias
    - ✓ Exemplo IA-32:

```
in    AX,io_port   ; lê de um porto de I/O
out   io_port,AX   ; escreve para um porto I/O
```

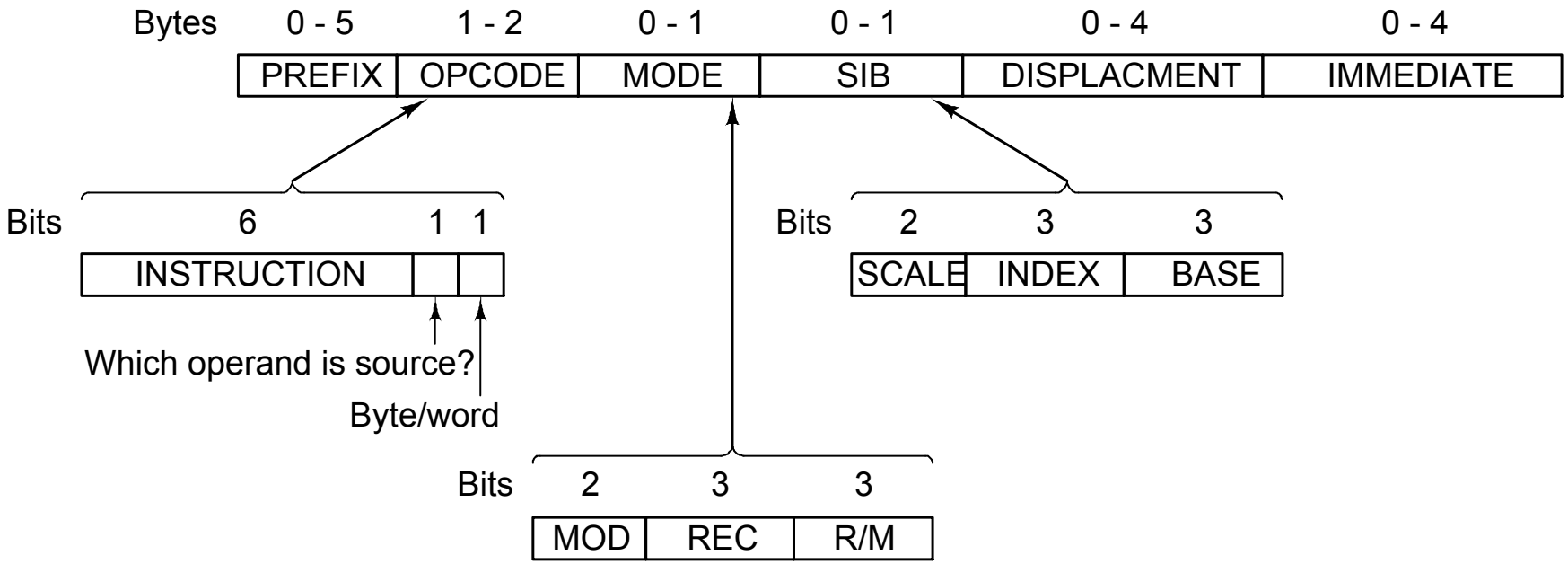
Formatos de Instrução – Tamanho Fixo

- Abordagem das máquina RISC
- Exemplo: SPARC 32 bits



Formatos de Instrução – Tamanho Variável

- Abordagem das máquina CISC
- Exemplo: Pentium (IA-32)

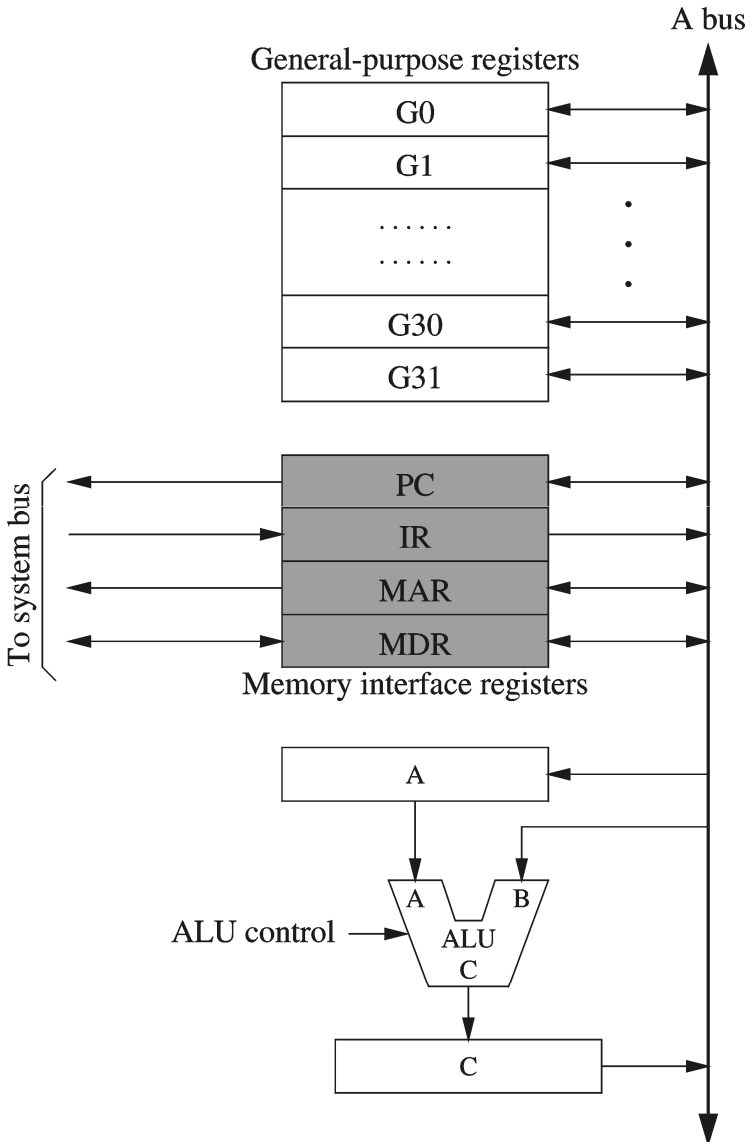


Controlo Microprogramado

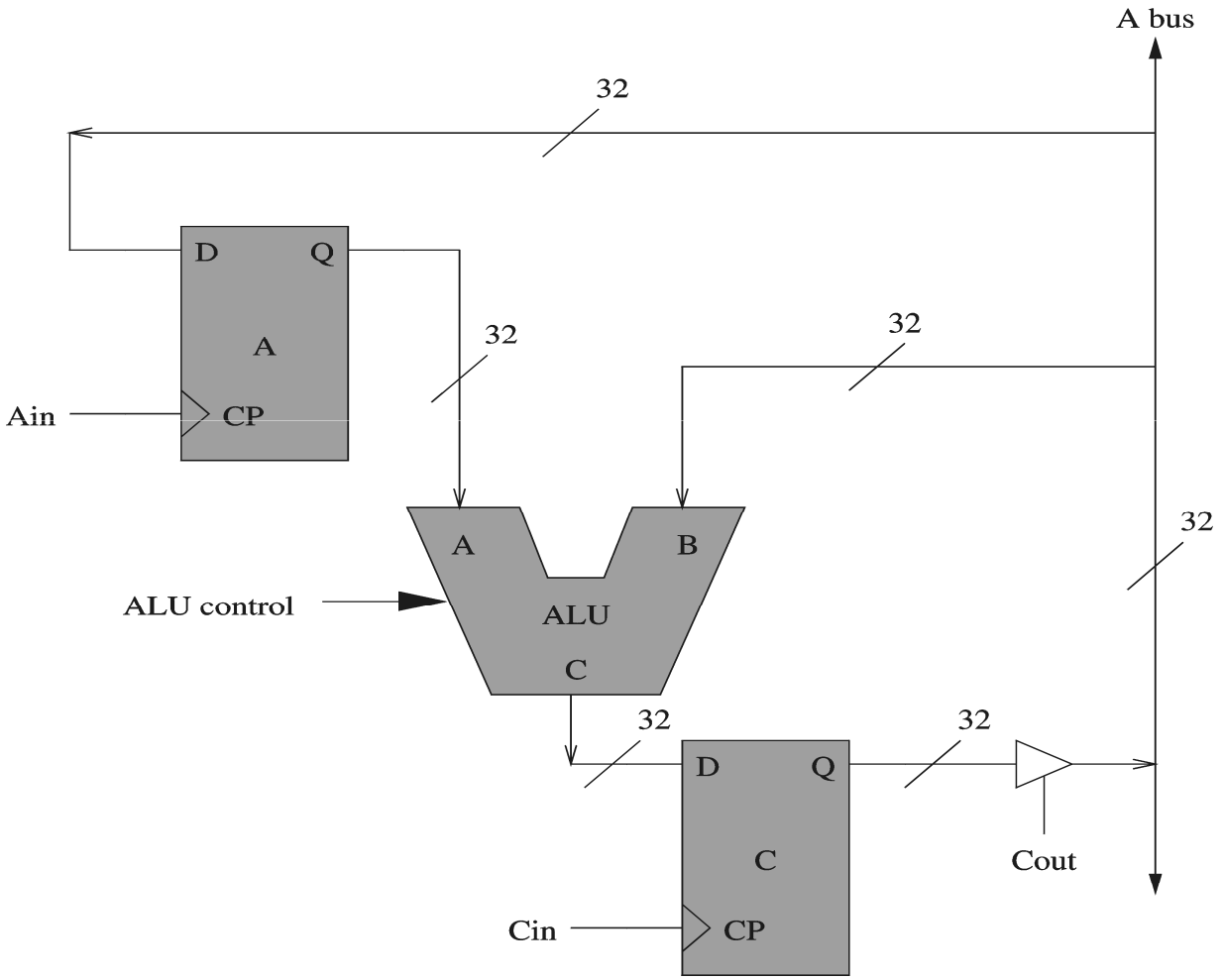
- Como é que as instruções vistas são executadas pelo hardware?
- Nos acetatos seguintes assume-se:
  - ✓ Um só bus de 32 bits para interligar as diversas entidades
  - ✓ As operações são a 32 bits
  - ✓ Os seguintes registos de 32 bits:

Nome	Uso
G0 a G31	Geral
PC	Program Counter
IR	Instruction Register
MAR	Memory Address Register
MDR	Memory Data Register

Controlo Microprogramado



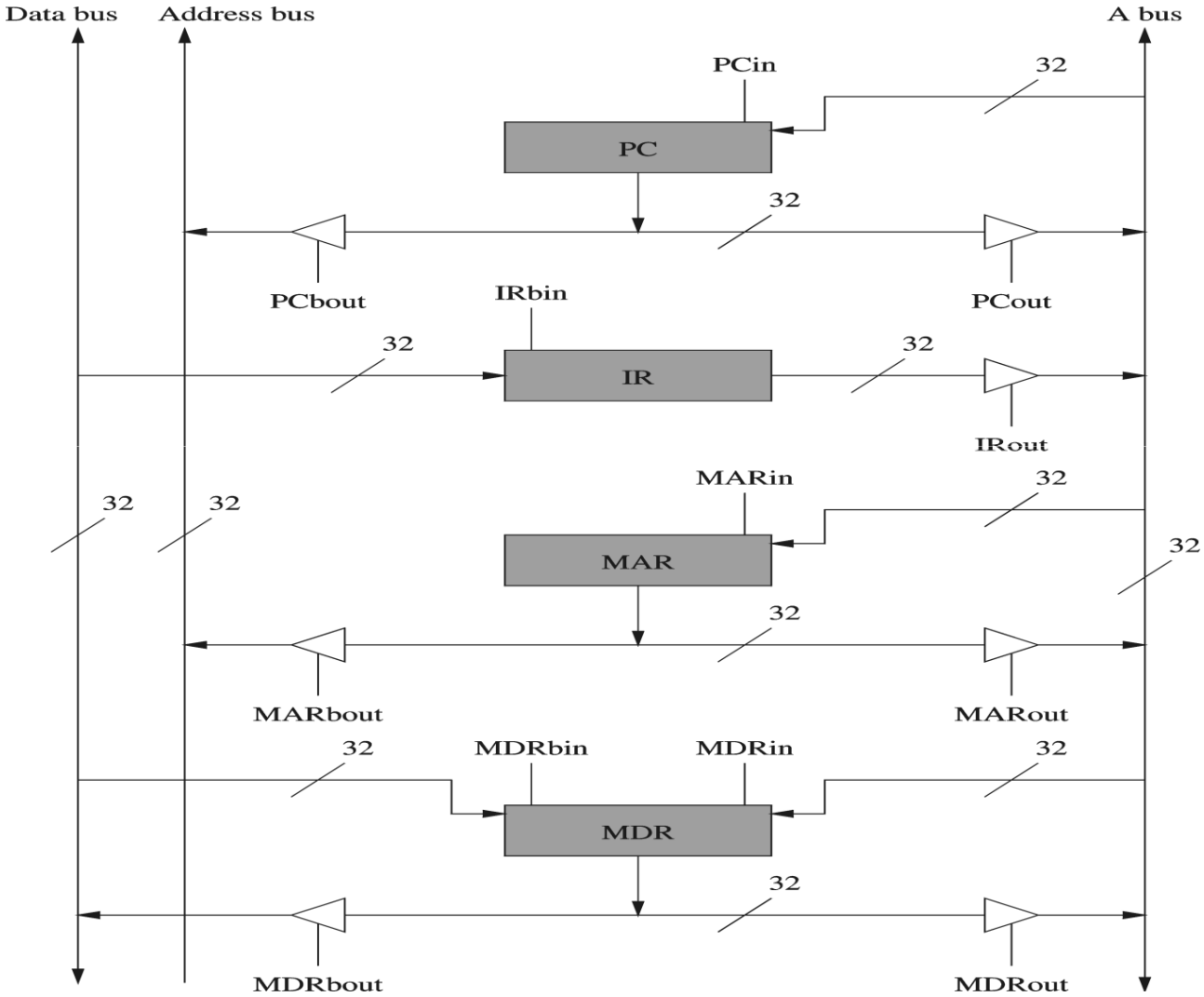
Controlo Microprogramado - ALU



Controlo Microprogramado – Registos

Nome	Uso	Sinais de controlo
G0 a G31	Geral	GXin e GXout
PC	Program Counter	PCin, PCout e PCbout
IR	Instruction Register	IRin e IRout
MAR	Memory Address Register	MARin, MARout e MARbout
MDR	Memory Data Register	MDRin, MDRout e MDRbin, MDRbout

Interface com o Bus de Sistema



Implementação de `add G9,G5,G7`

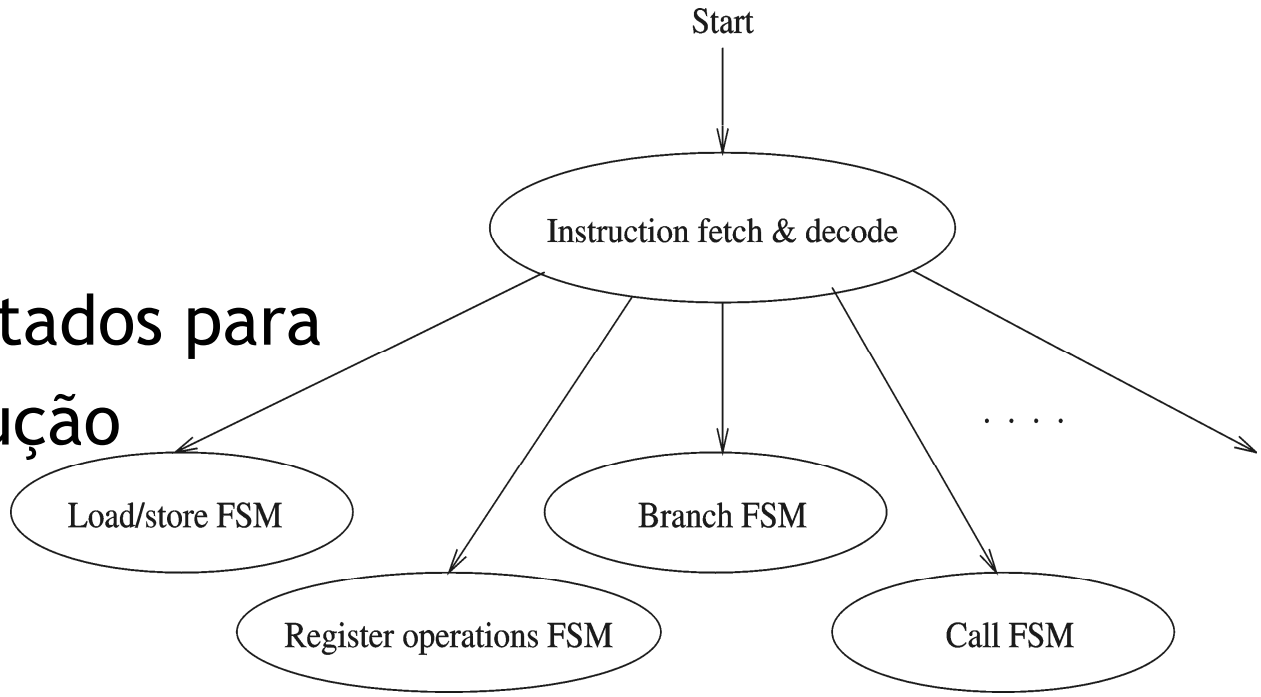
1. Transferir o conteúdo de G5 para o registo A
   - ✓ Activar G5out e Ain
2. Colocar o conteúdo de G7 no bus
   - ✓ Activar G7out
- Indicar à ALU que deve realizar a operação
  - ✓ Activar os sinais de controlo da ALU
- Colocar o resultado no registo C
  - ✓ Activar Cin
3. Transferir o conteúdo de C para G9
   - ✓ Activar Cout e G9in

Implementação do fetch

1. Colocar PC no bus de sistema e no bus local (para calcular-se o novo valor de PC)
   - ✓ Activar PCbout e PCout
- Indicar à ALU que deve realizar a operação **`add4`**
  - ✓ Activar os sinais de controlo da ALU
- Colocar o resultado no registo C
  - ✓ Activar Cin
2. Esperar 1 ciclo pela resposta da memória
   - ✓ Durante a espera transferir o conteúdo de C para PC
     - ✓ Activar Cout e PCin
3. Colocar o resultado no IR
   - ✓ Activar IRin

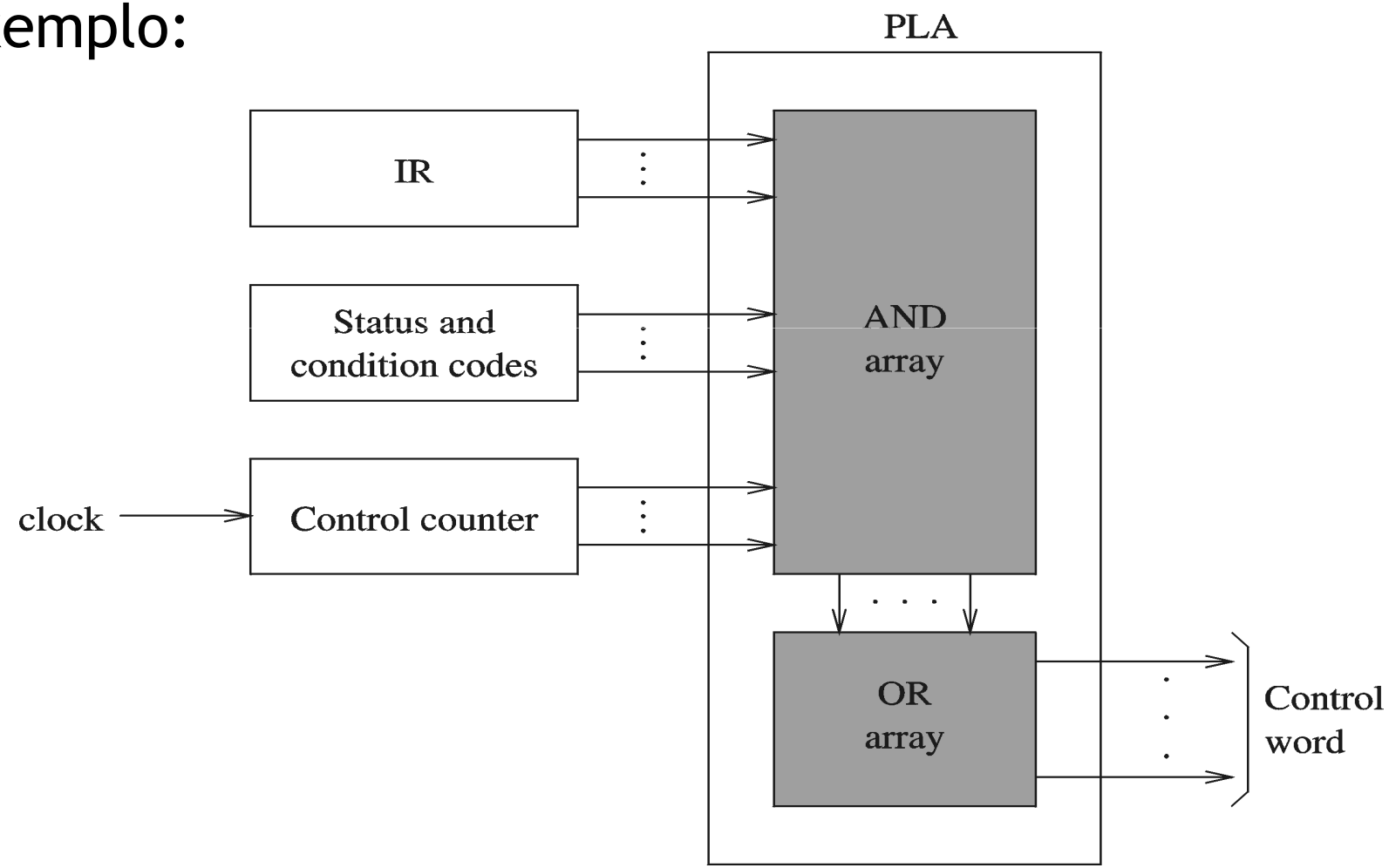
Grupos de Instruções

- Grupos de Instruções
  - ✓ Load/Store - Operam sobre memória
  - ✓ Registo - Operam apenas sobre registos
  - ✓ Salto
  - ✓ Call
  - ✓ …
- Máquina de estados para definir a execução da instrução



Implementação em Hardware

- Abordagem das máquinas RISC
- Exemplo:



Implementação em Software

- Abordagem das máquinas CISC
  - ✓ Implementação em hardware é complexa e cara
    - ✓ Só é exequível para conjuntos de instruções simples
  - ✓ Traduzir a instrução num programa que gera os sinais de controlo

- Exemplo: **`add G9,G5,G7`**
  - ✓ 3 passos – assumindo um ciclo por passo

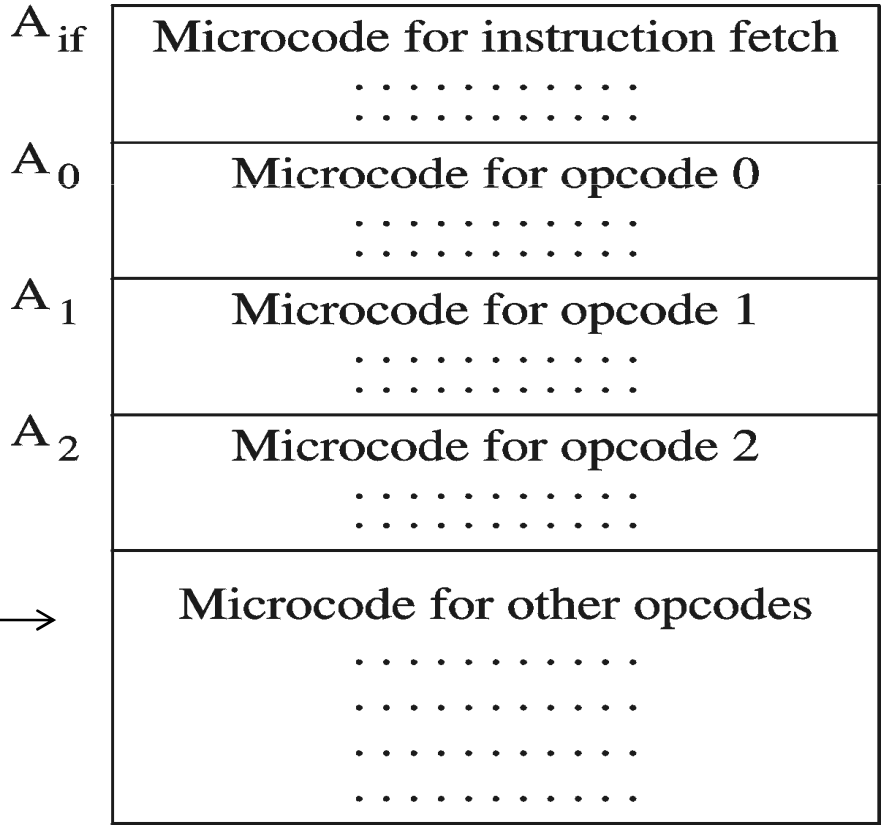
```
S1    G5out: Ain;
S2    G7out: ALU=add: Cin;
S3    Cout: G9in: end;
```

Implementação em Software

- Microinstrução - codificação os sinais de cada passo
- Microrotina – sequência de microinstruções que implementa a instrução original

- O microprograma implementa a máquina De estados

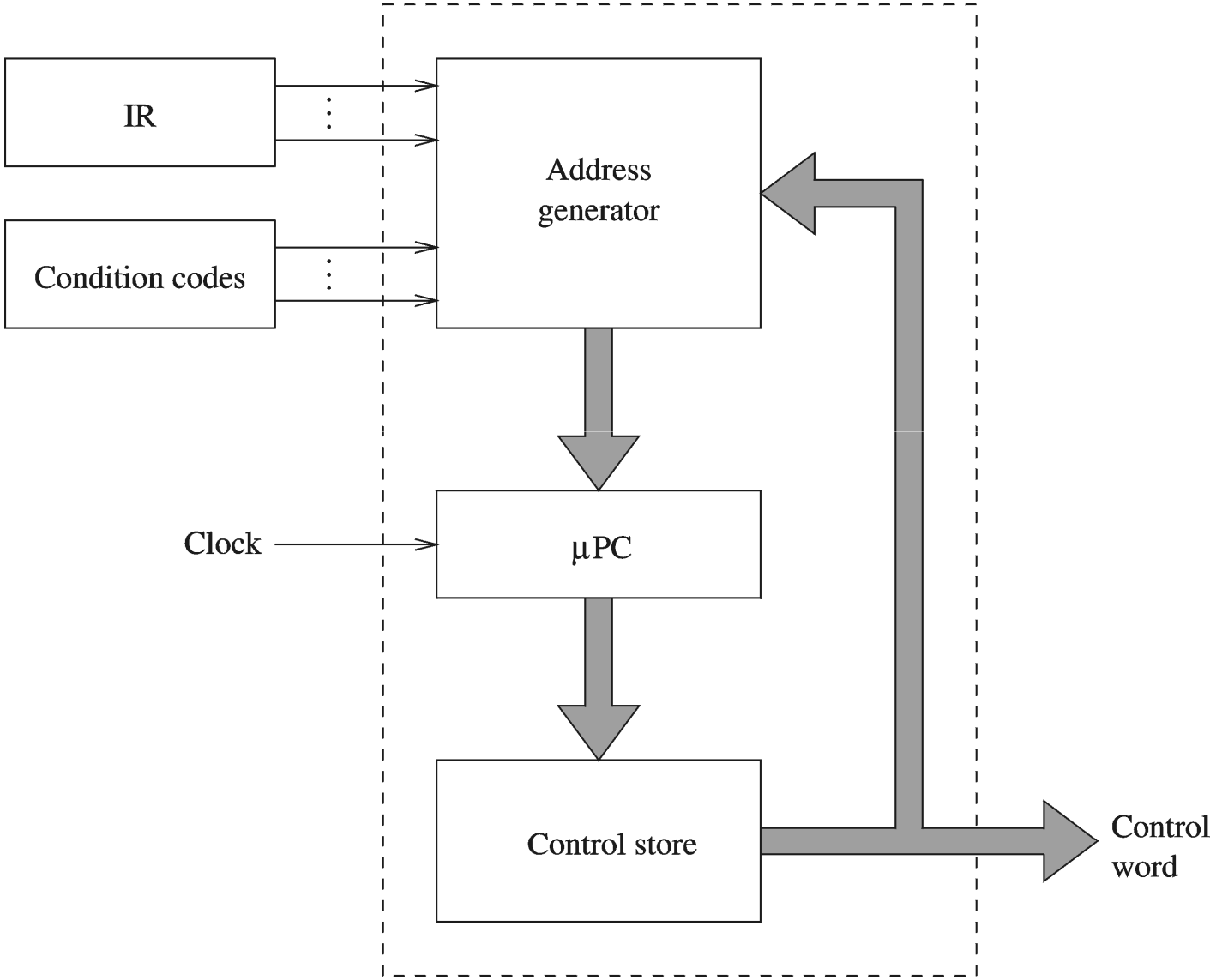
- Exemplo de uma organização linear →



Implementação em Software

- Um microcontrolador pode executar o microprograma e gerar os sinais de controlo
  - ✓ *Control store* → Guarda o microprograma
  - ✓ µPC → Program counter do microprograma
  - ✓ *Address generator* → Dado o conteúdo do IR e do registo (bits) de condição indica onde está a microrotina a executar, ou seja, qual é o seu endereço inicial

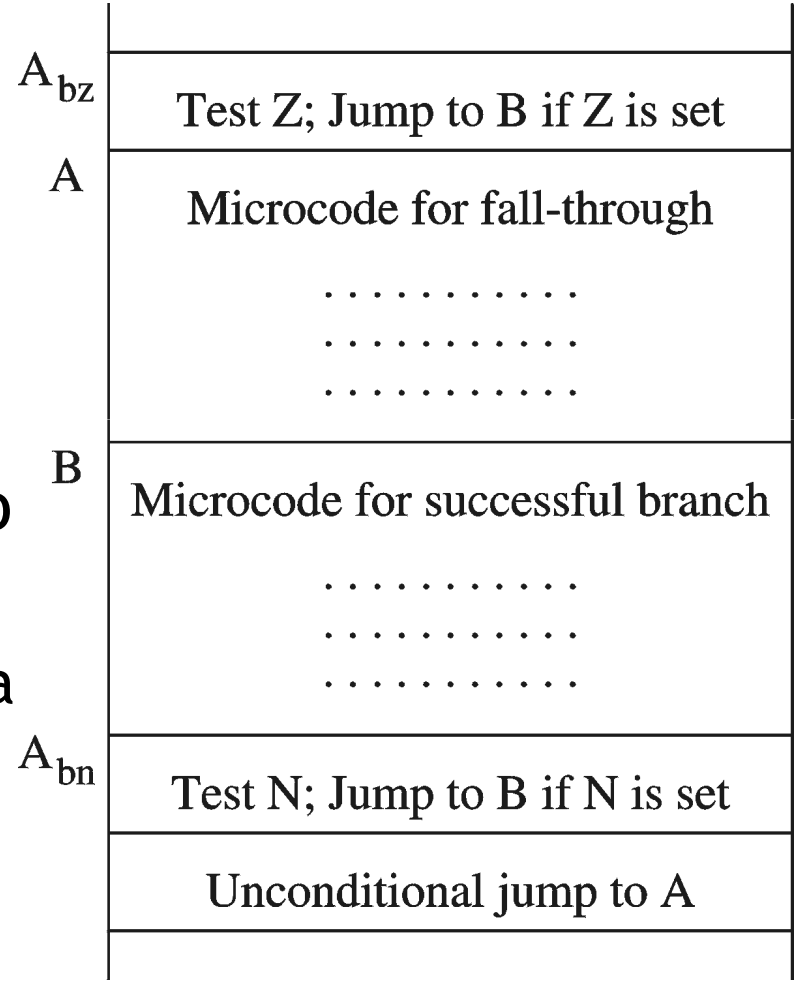
Implementação em Software



Implementação em Software

- A abordagem linear não factoriza código comum a várias microrotinas

- Uma abordagem mais eficiente consiste em ter apenas uma cópia do código comum e usar saltos
  - ✓ Requer endereço da próxima microinstrução a executar



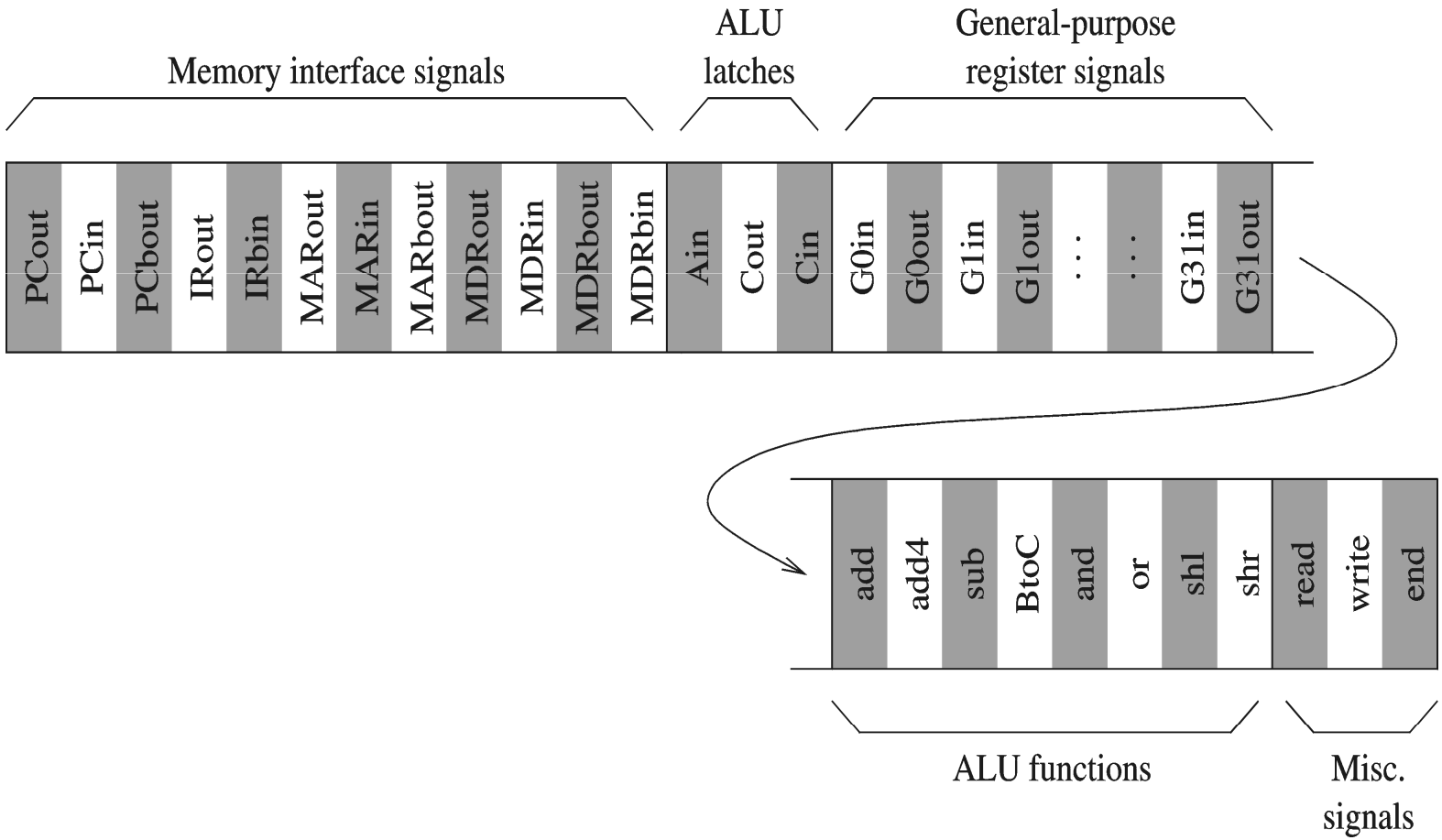
Implementação em Software

- Formato das microinstruções
  - ✓ Organização horizontal
    - ✓ Um bit para cada sinal
    - ✓ Muito flexível
    - ✓ Microinstruções longas
    - ✓ Suponhamos que a ALU do exemplo dado implementa 8 operações, ao todo são precisos 90 bits
      - ✓ 8 para as instruções
      - ✓ 3 para os registos A e B
      - ✓ 12 para os registos PC, IR, MAR e MDR
      - ✓ 64 para os registos de uso geral
      - ✓ 3 para diversos

Implementação em Software

Formato das microinstruções

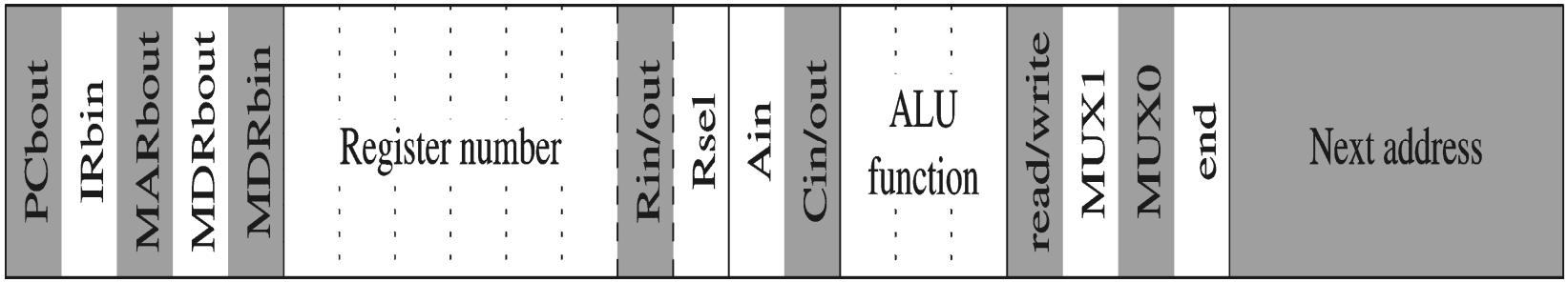
- ✓ Organização horizontal



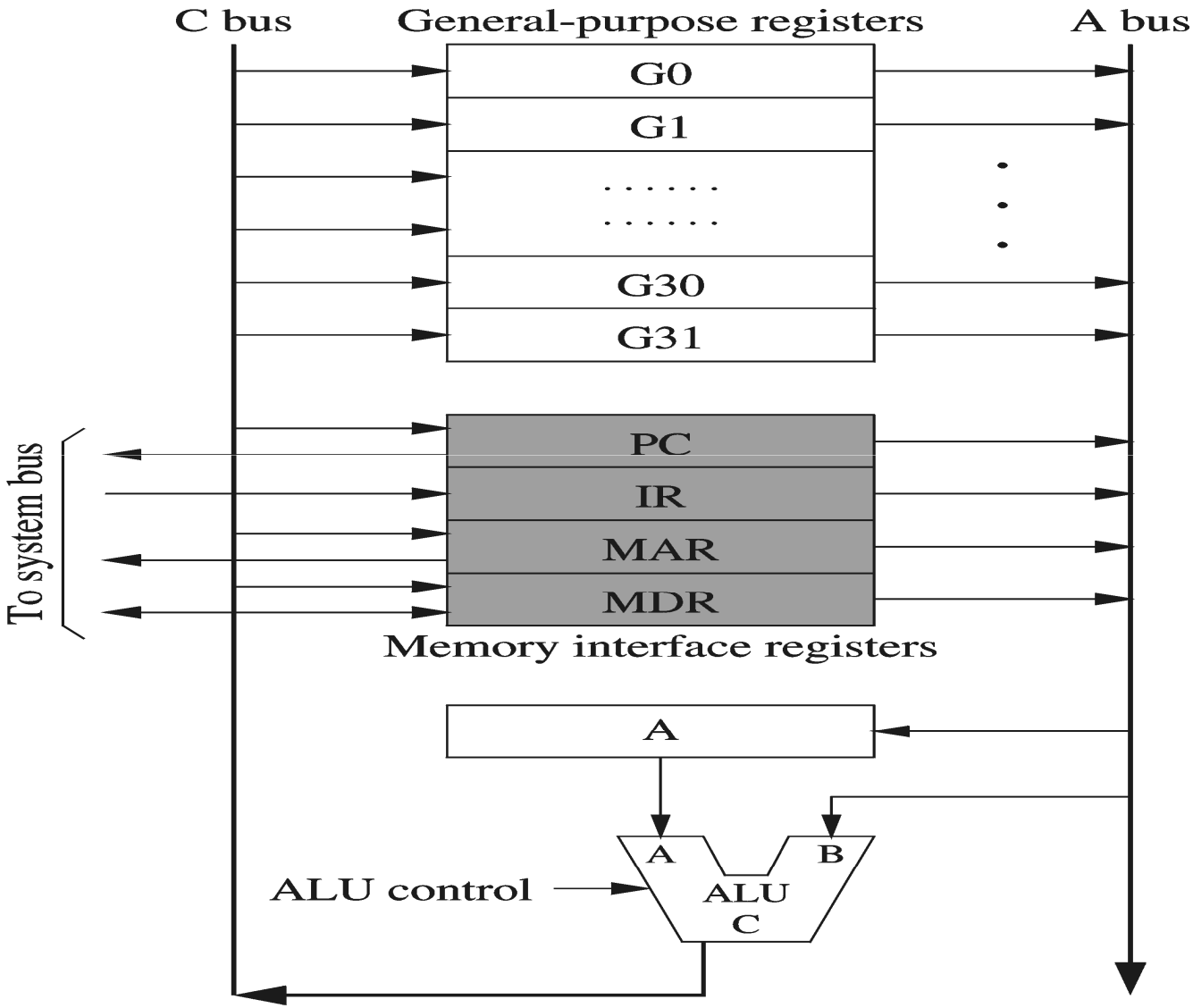
Implementação em Software

Formato das microinstruções

- ✓ Organização vertical
  - ✓ Reduzir o tamanho da instrução
  - ✓ Por exemplo usar apenas 5 bits para codificar os 32 registos e mais 1 para distinguir entre in e out
  - ✓ Precisa de hardware dedicado para a descodificação
  - ✓ Não se pode codificar movimento de dados entre registos numa só microinstrução



Adicionar mais um bus



Adicionar mais um bus

- Diminui o tempo de execução das instruções
  - ✓ Elimina a multiplexagem
  - ✓ O resultado da ALU fica imediatamente disponível no bus C
  - ✓ Exemplo: **`add G9,G5,G7`**
    - ✓ Precisa de 3 passos com 1 bus

```
S1    G5out: Ain;
S2    G7out: ALU=add: Cin;
S3    Cout: G9in: end;
```

    - ✓ Precisa apenas de 2 com 2 buses

```
S1    G5out: Ain;
S2    G7out: ALU=add: G9in;
```

Performance

- Métricas usuais
  - ✓ Tempo de resposta
    - ✓ Orientado para o utilizador
  - ✓ Throughput
    - ✓ Orientado para o sistema

- Performance de componentes
  - ✓ Processadores, memória, discos, …
  - ✓ Métricas usuais
    - ✓ MIPS – Million Instructions per Second
    - ✓ MFLOPS – Million Floating point Operations per Second

Performance

- Um programa pode executar mais rápido se o CPU o executar em menos tempo:

Tempo de execução = nº de instruções do programa *
nº de ciclos por instrução *
duração de cada ciclo

Performance = 1/(Tempo de execução)

Performance - Médias

- Média aritmética
- Média aritmética ponderada
- Média geométrica
- Média geométrica ponderada

	Resp. time on machine				Normalized values		
	REF	A	B	Ratio	A	B	Ratio
Program 1	10	11	12		1.1	1.2	
Program 2	40	49.5	60		1.24	1.5	
Arith. mean		30.25	36	1.19	1.17	1.35	1.16
Geo. mean		23.33	26.83	1.15	1.167	1.342	1.15

Performance - Médias

- A média aritmética não goza a seguinte propriedade:

$$\frac{média(a_i)}{média(b_i)} = média(\frac{a_i}{b_i})$$

- A média geométrica é boa para analisar razões mas não valores absolutos

	Resp. time on machine	
	A	B
Program 1	20	200
Program 2	50	5
Arith. mean	35	102.5
Geo. mean	31.62	31.62

Performance – Benchmark suites

- Benchmarks – conjunto de programas que avaliam a performance de um ou vários componentes

- Os benchmark SPEC (Standard Performance Evaluation Corporation) são dos mais utilizados
  - ✓ SPEC CPU2006 – CPU, memória, compilador
  - ✓ SPECapc – Aplicações 3D conhecidas
  - ✓ SPECjvm2008 – JVM