Parte 6
Otimizações da Arquitetura

Bibliografia

[1] Miles J. Murdocca e Vincent P. Heuring, “Introdução à Arquitetura de Computadores”

[2] Andrew S. Tanenbaum, “Modern Operating Systems”

[3] William Stallings, “Arquitetura e Organização de Computadores”

Busca pelo Desempenho

- Melhor tecnologia de circuitos integrados
  - Aumento da complexidade das instruções
  - Aumento da complexidade dos modos de endereçamento
  - Idéia: aumentar a complexidade arquitetural diminui a "lacuna semântica"
    - Distância entre a linguagem de máquina e a linguagem de alto nível
  - Porém nem sempre instruções complexas são mais eficientes...
  - Além disso, nem sempre o programador/compilador utiliza as instruções mais complexas...

Freqüência das Instruções

- **Freqüência de ocorrência de instruções para diversos tipos de linguagens. (Os percentuais não somam 100 devido a arredondamentos.) (Adaptado de Knuth, D. E., *An Empirical Study of FORTRAN Programs, Software—Practice and Experience*, 1, 105-133, 1971.)**

Statement	Average Percent of Time
Assignment	47
If	23
Call	15
Loop	6
Goto	3
Other	7

Onde Melhorar?

- ~ 50% das instruções são atribuições
- operações mais complexas ~ 7%

Deve-se otimizar as instruções que ocorrem com maior freqüência durante a execução...

Nem sempre são as instruções mais complexas...

Complexidade das Atribuições

- **Percentagens mostrando a complexidade das atribuições e das chamadas de função. (Adaptado de Tanenbaum, A., *Structured Computer Organization*, 4/e, Prentice Hall, Upper Saddle River, New Jersey, 1999.)**

	Percentage of Number of Terms in Assignments	Percentage of Number of Locals in Procedures	Percentage of Number of Parameters in Procedure Calls
0	–	22	41
1	80	17	19
2	15	20	15
3	3	14	9
4	2	8	7
≥ 5	0	20	8

Onde Melhorar? (cont.)

- Atribuições
  - Caso mais freqüente é a atribuição simples
- Existem poucas variáveis locais em cada procedimento
- Número de parâmetros nas chamadas de procedimentos em geral é baixo

- Conclusão
  - Programas são simples do ponto de vista das instruções utilizadas
- Compiladores em geral não utilizam instruções e modos de endereçamento complexos

- Motivações para máquinas RISC

Exemplo

- Atribuição
  - Ocorre com muita freqüência
- Regra de projeto: tornar o caso freqüente rápido

  = torná-lo simples
- Exemplo: forçar toda a comunicação com a memória a utilizar duas instruções
  - LOAD/STORE
    - típico de arquiteturas RISC

Speedup e Eficiência

- O *speedup* *S* é a razão entre o tempo de execução sem a melhoria (w0) e o tempo de execução com a melhoria (w).

$$S = \frac{T_{wo}}{T_w} \qquad S = \frac{T_{wo} - T_w}{T_w} \times 100$$

- O tempo de execução *T* é calculado multiplicando o número de instruções, *IC*, pelo número de ciclos por instrução, *CPI*, pelo período do clock da máquina, τ.

$$T = IC \times CPI \times \tau$$

- Substituindo *T* no cálculo do *speedup*, obtém-se:

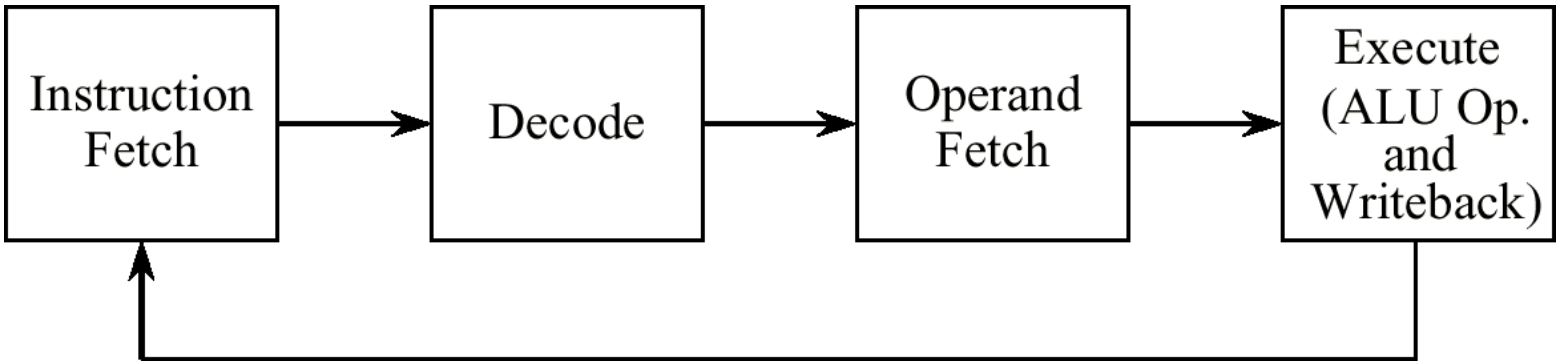
$$S = \frac{IC_{wo} \times CPI_{wo} \times \tau_{wo} - IC_w \times CPI_w \times \tau_w}{IC_w \times CPI_w \times \tau_w} \times 100$$

Exemplo

- **Estimar o *speedup* obtido pela substituição de uma CPU com CPI médio de 5 por outra CPU com CPI médio de 3.5, com período de clock aumentado de 100 ns para 120 ns.**
- **Utilizando a equação anterior:**

$$S = \frac{5 \times 100 - 3.5 \times 120}{3.5 \times 120} \times 100 = 19\%$$

Pipeline de Instruções com 4 Estágios



Comportamento do Pipeline

- **Comportamento do pipeline durante uma referência à memória e durante um desvio.**

	Time 1	2	3	4	5	6	7	8
Instruction Fetch	addcc	ld	srl	subcc	be	nop	nop	nop
Decode		addcc	ld	srl	subcc	be	nop	nop
Operand Fetch			addcc	ld	srl	subcc	be	nop
Execute				addcc	ld	srl	subcc	be
Memory Reference						ld		

Espaços de Atraso de Desvios e Leituras

- `ld` e `st` requerem 5 ciclos, mas as restantes somente 4
  - Uma instrução que sucede `ld` ou `st` não deve usar registradores em comum com esta
  - Solução segura: inserir um `nop`

    Leitura atrasada (*delayed load*)
  - Solução semelhante para o desvio

    Desvio atrasado (*delayed branch*)
- Otimização
  - O compilador pode encontrar instruções que *podem* preencher o espaço

Preenchendo o Espaço de Leitura Atrasada

- **Código SPARC, (a) com inserção de `nop`, e (b) com o `srl` deslocado para a posição do `nop`.**

```
srl    %r3, %r5
addcc  %r1, 10, %r1
ld     %r1, %r2
nop
subcc  %r2, %r4, %r4
be     label
```

(a)

```
addcc  %r1, 10, %r1
ld     %r1, %r2
srl    %r3, %r5
subcc  %r2, %r4, %r4
be     label
```

(b)

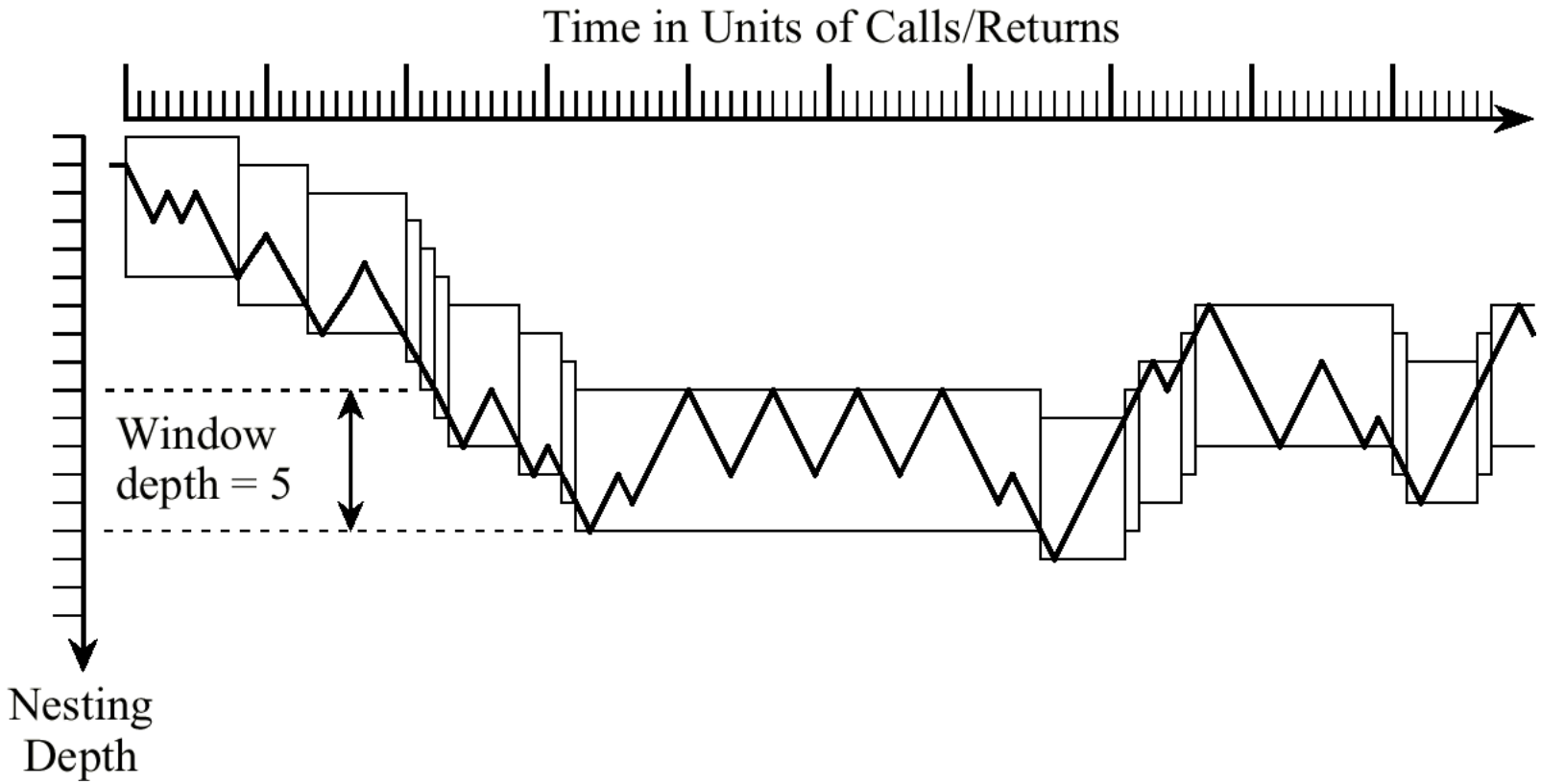
Execução Especulativa de Instruções

- Advinhar a decisão de desvio
- Desfazer efeitos colaterais se escolha errada
- Estatística
  - Laços executam com mais freqüência que são evitados
- Chute errado: fase de execução parada e pipeline esvaziado
- Custo pode ser elevado
  - Ex. Arquiteturas RISC em geral utilizam NOPs

Análise de Eficiência do Pipeline

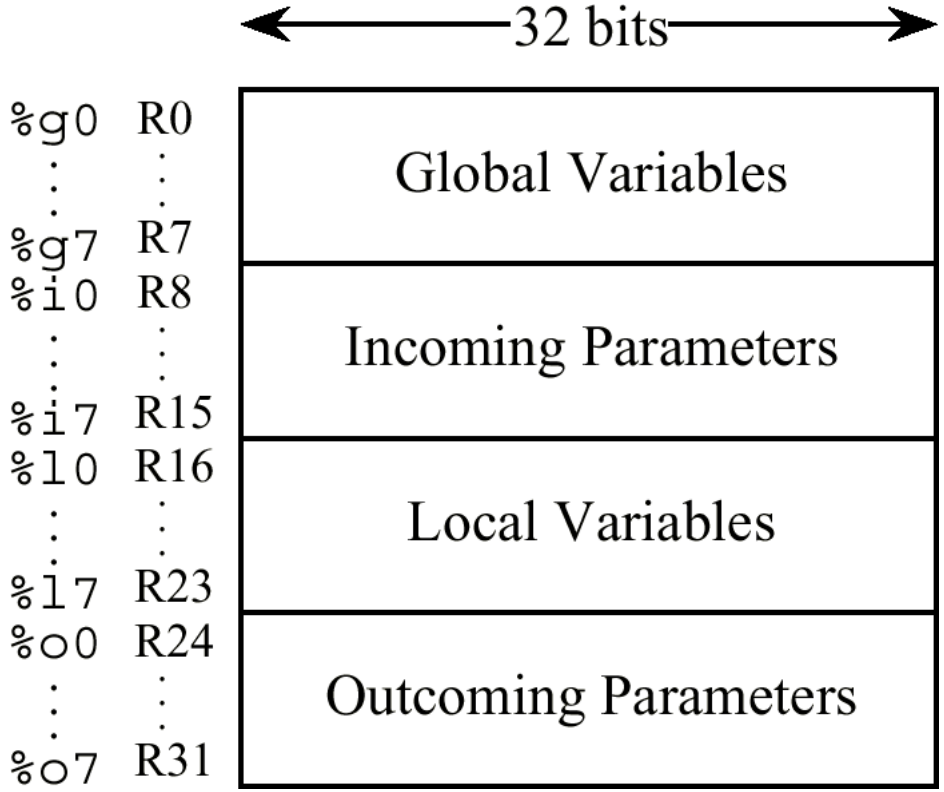
Comportamento de Chamada-Retorno

- **Comportamento de chamada-retorno em função da profundidade de aninhamento e do tempo (Adapted from Stallings, W., *Computer Organization and Architecture: Designing for Performance*, 4/e, Prentice Hall, Upper Saddle River, 1996).**

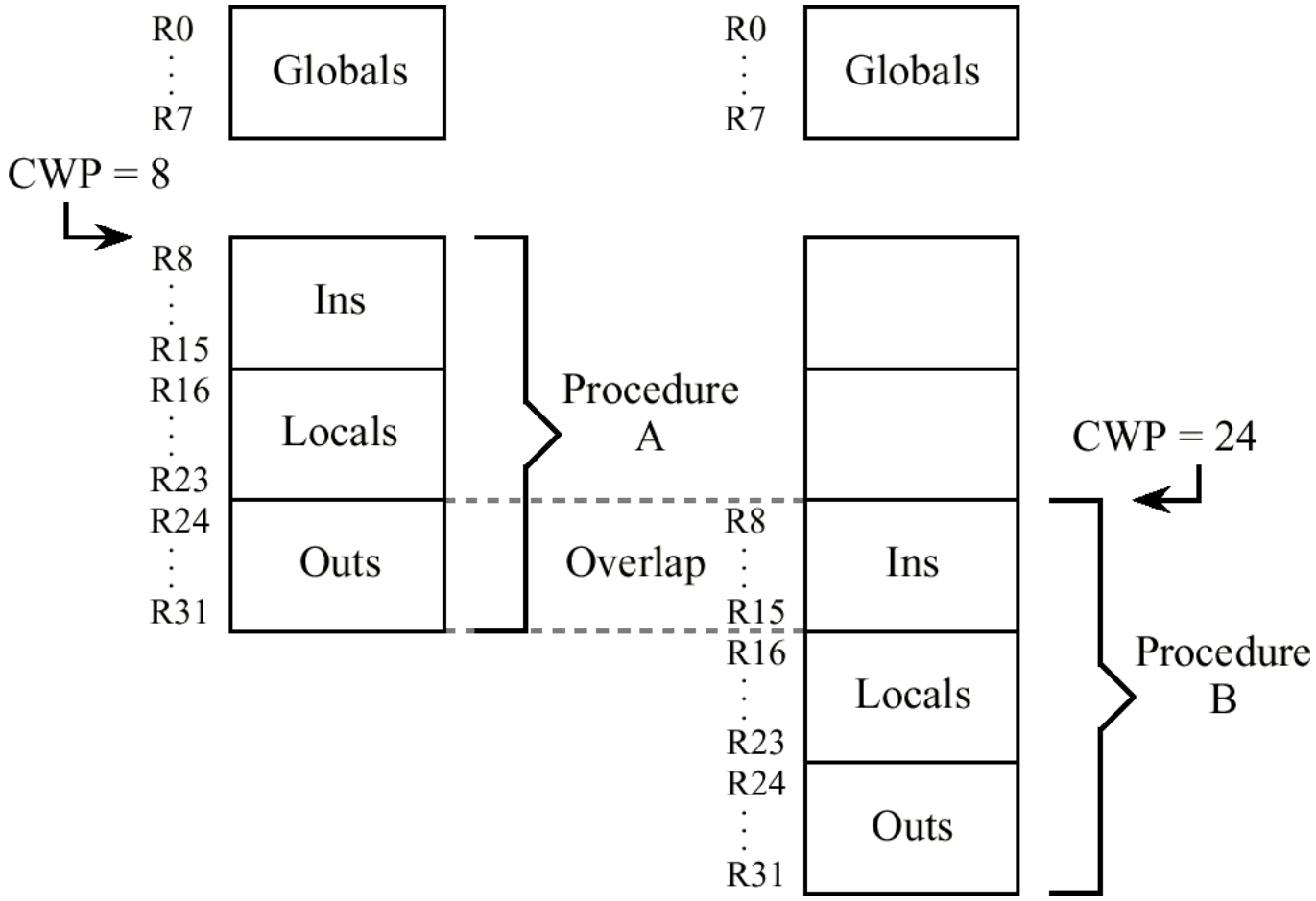


Registradores SPARC

- **Visão do usuário dos registradores RISC I.**



Sobreposição de Janelas de Registradores



Exemplo: Programa C Compilado

- **Código fonte de um programa a ser compilado com o gcc.**

```
/* Example C program to be compiled with gcc */

#include
<stdio.h>
main ()
{
     int a, b, c;

     a = 10;
     b = 4;
     c = add_two(a, b);

     printf("c = %d\n", c);
}

int add_two(a,b)
int a, b;
{
     int result;

     result = a + b;
     return(result);
}
```

Código SPARC gerado pelo gcc

```
! Output produced by gcc compiler on Solaris (Sun UNIX)
! Annotations added by author

.file   add.c   ! Identifies the source program
.section        ".rodata"       ! Read-only data for routine main
        .align 8        ! Align read-only data for routine main on an
                        ! 8-byte boundary
.LLC0
        .asciz  "c = %d\n"      ! This is the read-only data
.section        "text"  ! Executable code starts here
        .align 4  ! Align executable code on a 4-byte (word) boundary
        .global main
        .type   main,#function
        .proc   04
main:           ! Beginning of executable code for routine main
        !#PROLOGUE#     0
        save %sp, -128, %sp  ! Create 128 byte stack frame. Advance
                             ! CWP (Current Window Pointer)
        !#PROLOGUE#     1
        mov 10, %o0  ! %o0 <- 10. Note that %o0 is the same as %r24.
    ! This is local variable a in main routine of C source program.
        st %o0, [%fp-20]     ! Store %o0 five words into stack frame.
        mov 4, %o0   ! %o0 <- 4. This is local variable b in main.
        st %o0, [%fp-24]     ! Store %o0 six words into stack frame.
        ld [%fp-20], %o0     ! Load %o0 and %o1 with parameters to
        ld [%fp-24], %o1     ! be passed to routine add_two.
        call add_two, 0      ! Call routine add_two
        nop     ! Pipeline flush needed after a transfer of control
        st %o0, [%fp-28]     ! Store result 67 words into stack frame.
                             ! This is local variable c in main.
        sethi %hi(.LLC0), %o1 ! This instruction and the next load
        or %o1, %lo(.LLC0), %o0 ! the 32-bit address .LLC0 into %o0
        ld [%fp-28], %o1  ! Load %o1 with parameter to pass to printf
```

Código SPARC gerado pelo gcc (cont.)

```
        call printf, 0
        nop     ! A nop is needed here because of the pipeline flush
                ! that follows a transfer of control.
.LL1
        ret     ! Return to calling routine (Solaris for this case)
        restore ! The complement to save. Although it follows the
            ! return, it is still in the pipeline and gets executed.
.LLfe1
        .size   main, .LLfe1-main ! Size of
        .align 4
        .global add_two
        .type   add_two, #function
        .proc 04
add_two:
         !#PROLOGUE# 0
         save %sp, -120, %sp
         !#PROLOGUE# 1
         st %i0, [%fp+68] !Same as %o0 in calling routine (variable a)
         st %i1, [%fp+72] !Same as %o1 in calling routine (variable b)
         ld [%fp+68], %o0
         ld [%fp+72], %o1
         add %o0, %o1, %o0 ! Perform the addition
         st %o0, [%fp-20]  ! Store result in stack frame
         ld [%fp-20], %i0  ! %i0 (result) is %o0 in called routine
         b .LL2
         nop
.LL2:
         ret
         restore
.LLfe2:
         .size   add_two, .LLfe2-add_two
         .ident  "GCC: (GNU) 2.5.8"
```

Efeito da Otimização pelo Compilador

- **Código SPARC gerado com a flag de otimização -O:**

```
! Output produced by -O optimiziation for gcc compiler

.file   "add.c"
.section        ".rodata"
        .align 8
.LLC0:
        .asciz  "c = %d\n"
.section        ".text"
        .align 4
        .global main
        .type    main,#function
        .proc   04
main:
        !#PROLOGUE# 0
        save %sp,-112,%sp
        !#PROLOGUE# 1
        mov 10,%o0
        call add_two,0
        mov 4,%o1
        mov %o0,%o1
        sethi %hi(.LLC0),%o0
        call printf,0
        or %o0,%lo(.LLC0),%o0
        ret
        restore
.LLfe1:
        .size    main,.LLfe1-main
        .align 4
        .global add_two
        .type    add_two,#function
        .proc   04
add_two:
        !#PROLOGUE# 0
        !#PROLOGUE# 1
        retl
        add %o0,%o1,%o0
.LLfe2:
        .size    add_two,.LLfe2-add_two
        .ident  "GCC: (GNU) 2.7.2"
```

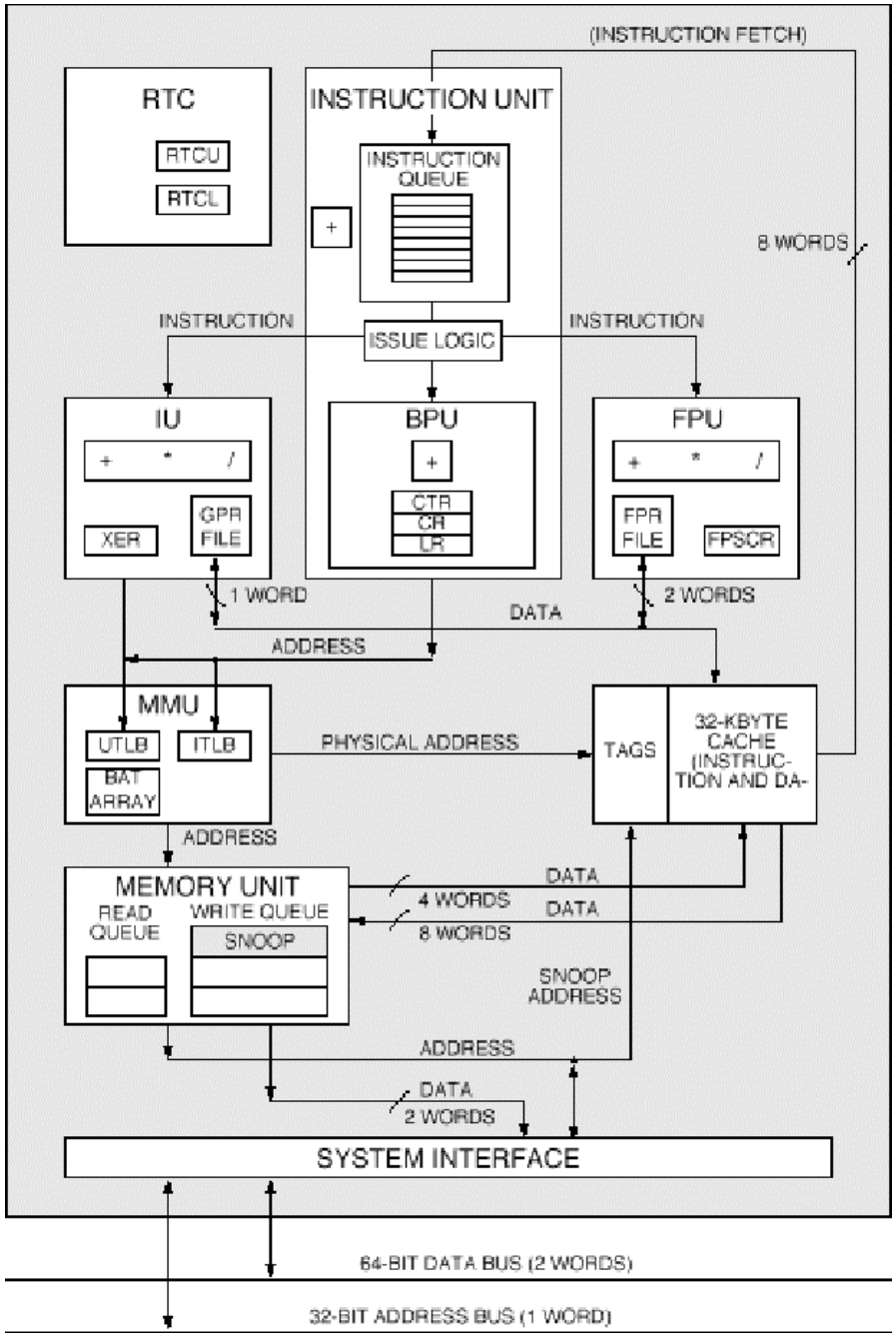
Máquinas de Início Múltiplo de Instruções (Superescalares)

- Unidades de Execução Separadas
  - Várias instruções em várias fases de execução ao mesmo tempo
- Uma ou mais
  - Unidades Inteiras (IU)
  - Unidades de Ponto Flutuante (FPU)
  - Unidades de Processamento de Desvios (BPU)

Máquinas Superescalares

- Instruções escalonadas nas unidades de execução
- Podem ser executadas fora de ordem
  - Precisam ser examinadas antes de iniciadas
- Unidade de Instrução
  - Lê instruções em uma fila
  - Determina tipos e dependências
  - Escalona para execução

A Arquitetura PowerPC 601



Máquinas VLIW

- *Very Long Instruction Word* (VLIW)
- Múltiplas operações compactadas em uma instrução
  - 128 bits ou mais...
- Em geral possui várias unidades de execução como as superescalares
- Compilador deve organizar várias operações em uma instrução
  - Deve ser pessimista nas escolhas de dependências
  - Software deve ser recompilado

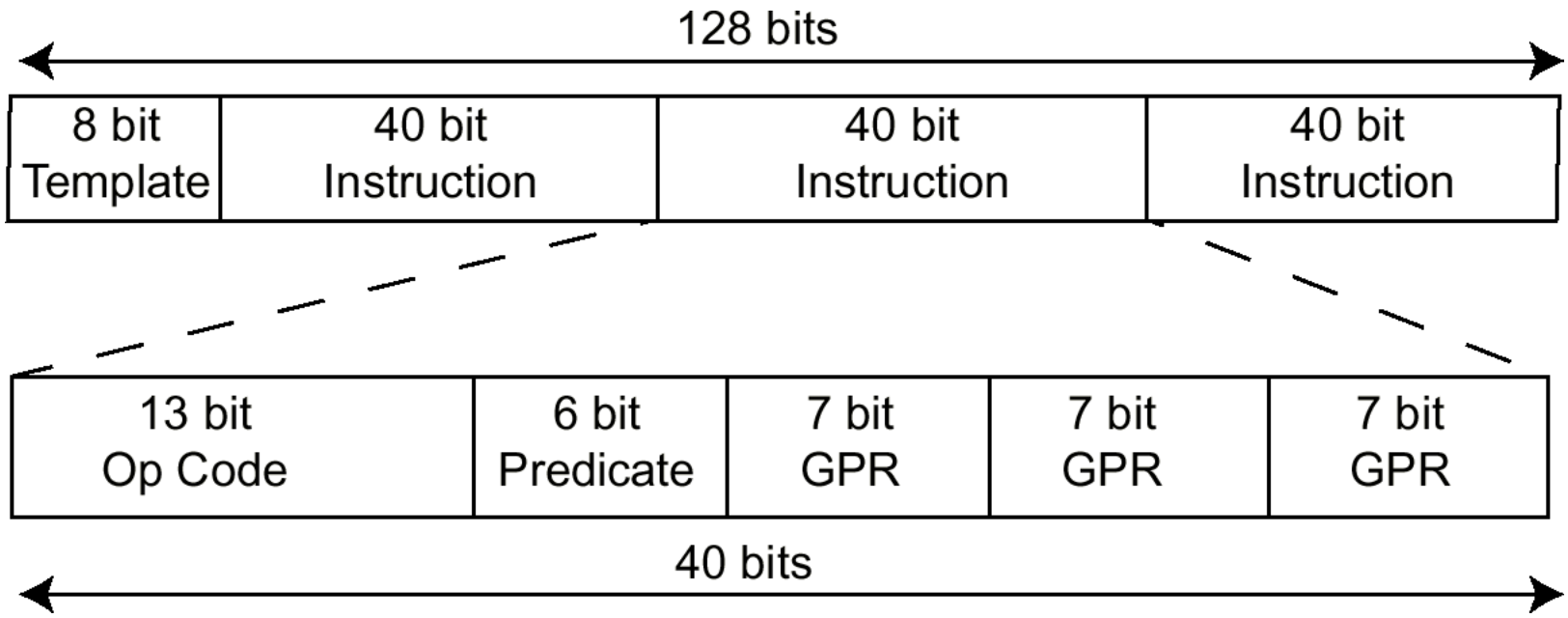
A Arquitetura IA-64

- EPIC (*Explicit Parallel Instruction Computing*)

- Palavra de Instrução
  - 3 operações compactadas em 128 bits
  - “padrão” define dependência/paralelismo

- Predicação de Desvios
  - Ambas as opções de desvio executadas em paralelo
- Leituras Especulativas
  - Analisar seqüência de instruções
  - Ler da memória antes do tempo, *especulando...*

Palavra de Instrução de 128 bits da Arquitetura IA-64



Arquitetura Paralela

- Processamento Paralelo
  - Vários processadores
  - Programa distribuído em vários processadores
- Características
  - Número de elementos de processamento (PEs)
  - Rede de interconexão entre os PEs
  - Organização da memória

Medidas de Desempenho

- Tempo paralelo
  - Tempo absoluto de execução do programa no processador paralelo
- *Speedup*
  - Razão entre o tempo no seqüencial e o tempo no paralelo
    - (melhor implementação em cada máquina)
- Nem todo o programa é paralelizável...
  - Eficiência
  - *Troughput*

Speedup Paralelo e Lei de Amdahl

- ***Speedup*** **do processamento paralelo em relação ao seqüencial:**

$$S = \frac{T_{Sequential}}{T_{Parallel}}$$

- **Lei de Amdahl, para *p* processadores e uma fração *f* de código não-paralelisável:**

$$S = \frac{1}{f + \frac{1-f}{p}}$$

  - **Por exemplo, se *f* = 10% das operações precisam ser feitas sequencialmente, então o speedup não pode ser maior que 10%, independente do número de processadores utilizado:**

$$S = \frac{1}{0.1 + \frac{0.9}{10}} \cong 5.3 \qquad S = \frac{1}{0.1 + \frac{0.9}{\infty}} = 10$$

$$p = 10 \; processors \qquad p = \infty \; processors$$

Eficiência

- ***Razão*** **entre o** ***speedup*** **e o número de processadores usado**

- **Exemplo**
  - ***Speedup*** **de 5.3 com 10 processadores**
  - **A eficiência é:**

$$\frac{5.3}{10} = .53, \textit{ or } 53\%$$

Throughput

- **Quantidade de cálculo realizada por unidade de tempo**
  - **Importante para aplicações limitadas por E/S e que usam pipeline**
- **Exemplo**
  - **Pipeline de 4 estágios que permanece cheio**
  - **Cada estágio do pipeline completa sua tarefa em 10 ns**
  - **Tempo médio para completar uma operação = 10 ns**
    - **Embora execução de qualquer operação = 40 ns**
  - **Vazão**

$$0.1 \frac{operation}{ns} = 10^8 \text{ operations per second.}$$

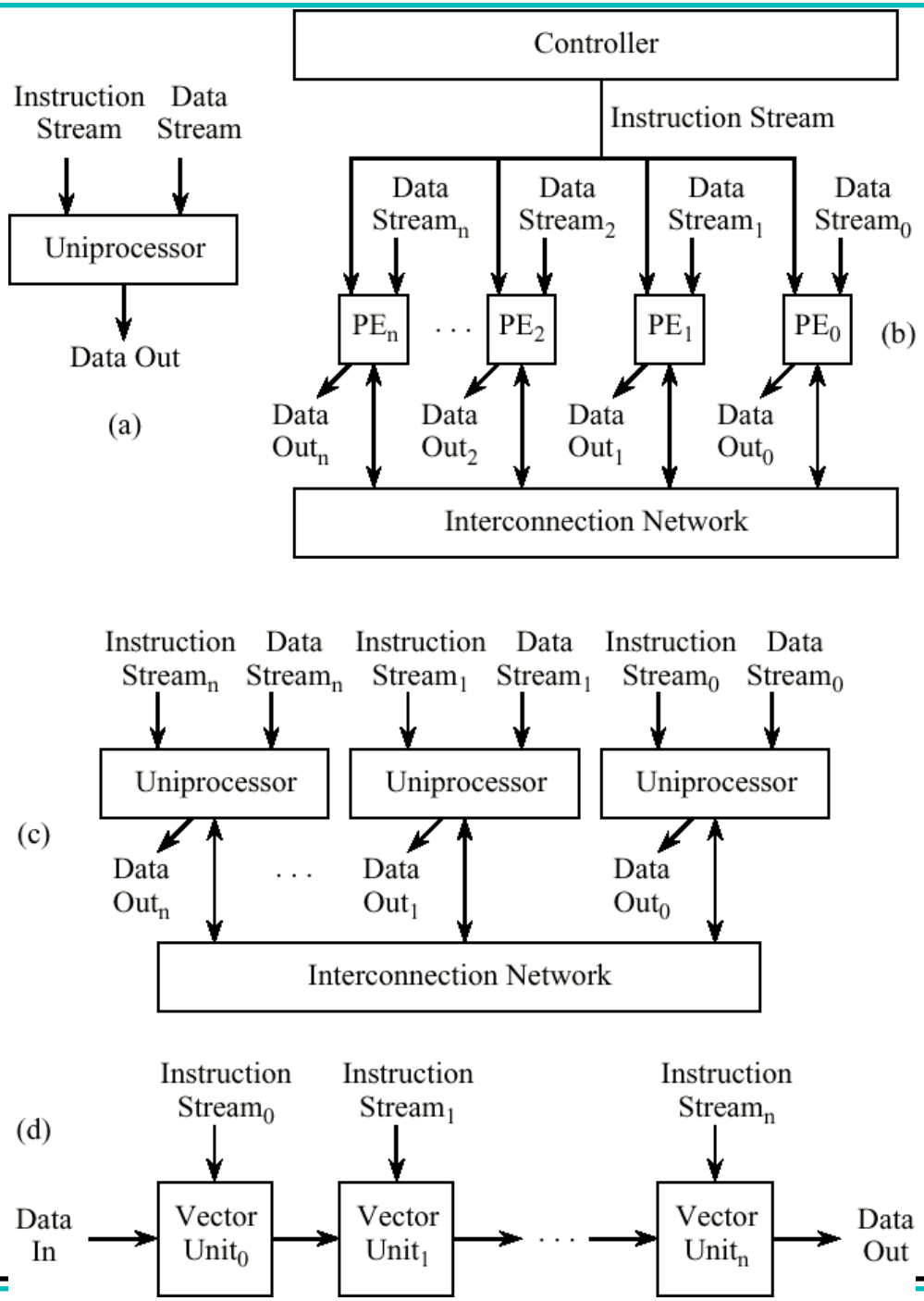
Taxonomia de Flynn

(a) SISD (Single Instruction Single Data stream);

(b) SIMD (Single Instruction Multiple Data stream;

(c) MIMD (Multiple Instruction Single Data stream;

(d) MISD (Multiple Instruction Single Data stream).



Topologias de Redes

(a) crossbar;

(b) barramento;

(c) anel;

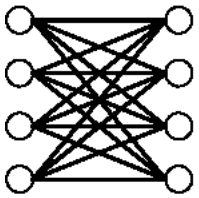
(d) malha;

(e) estrela;

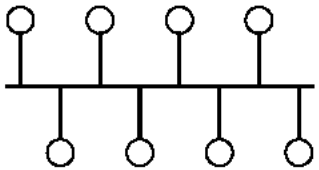
(f) árvore;

(g) embaralhamento perfeito;

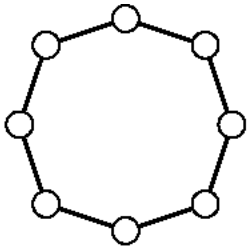
(h) hipercubo.



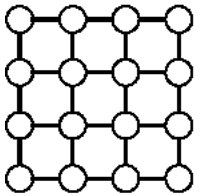
(a)



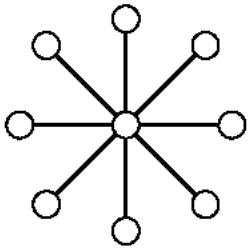
(b)



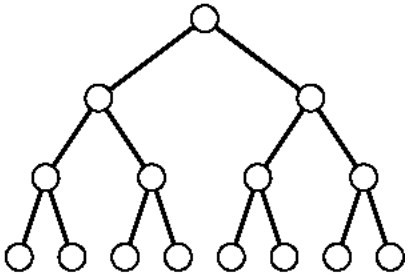
(c)



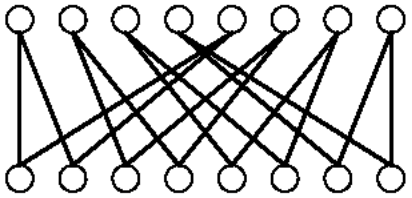
(d)



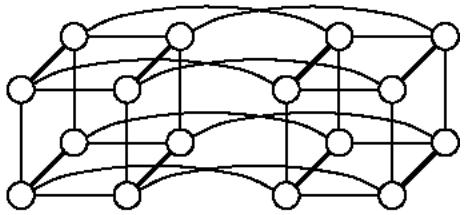
(e)



(f)



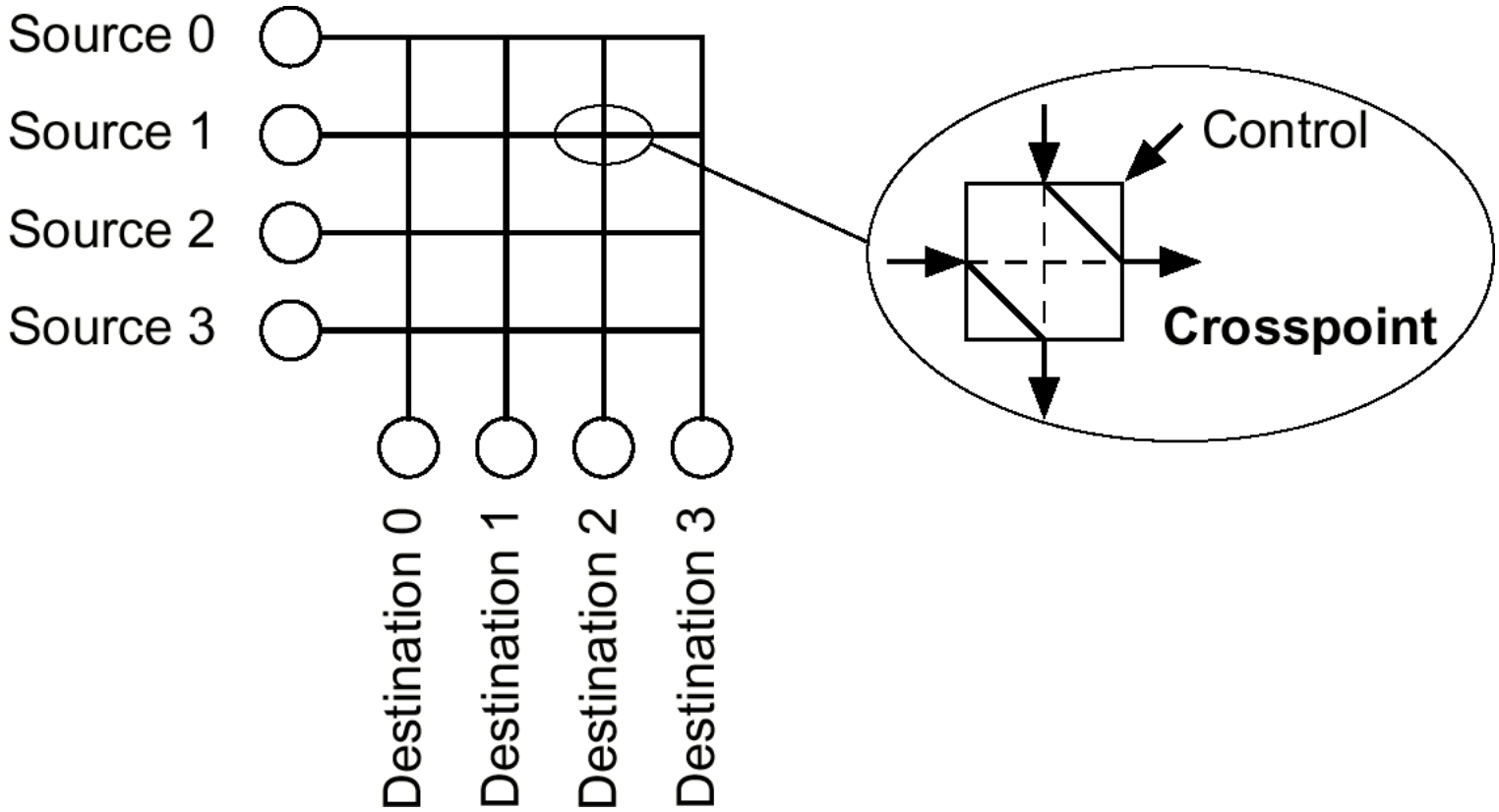
(g)



(h)

Crossbar

- **Organização interna de um *crossbar*.**

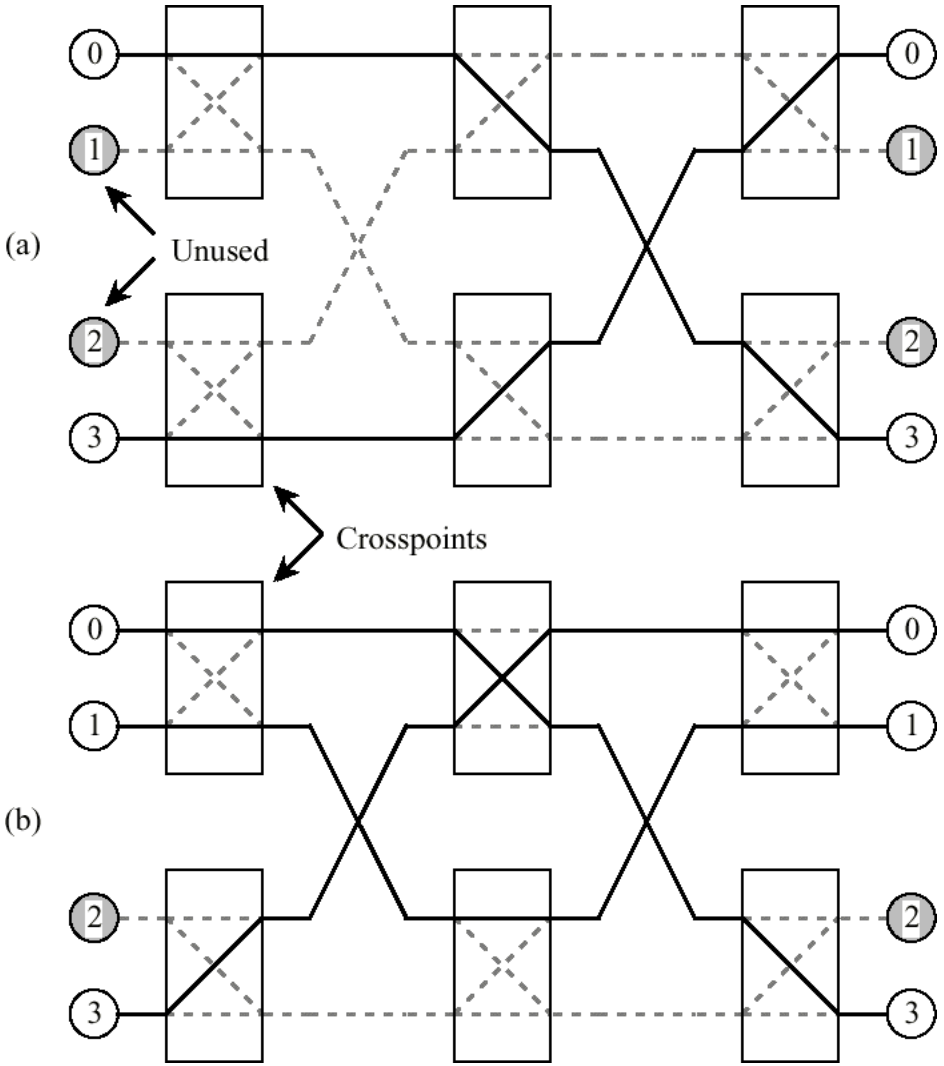


Decisões nos Cruzamentos

Cada cruzamento passa as entradas direto para as saídas, ou as inverte.

(a) decisões nos cruzamentos para conexões 0 → 3 e 3 → 0;

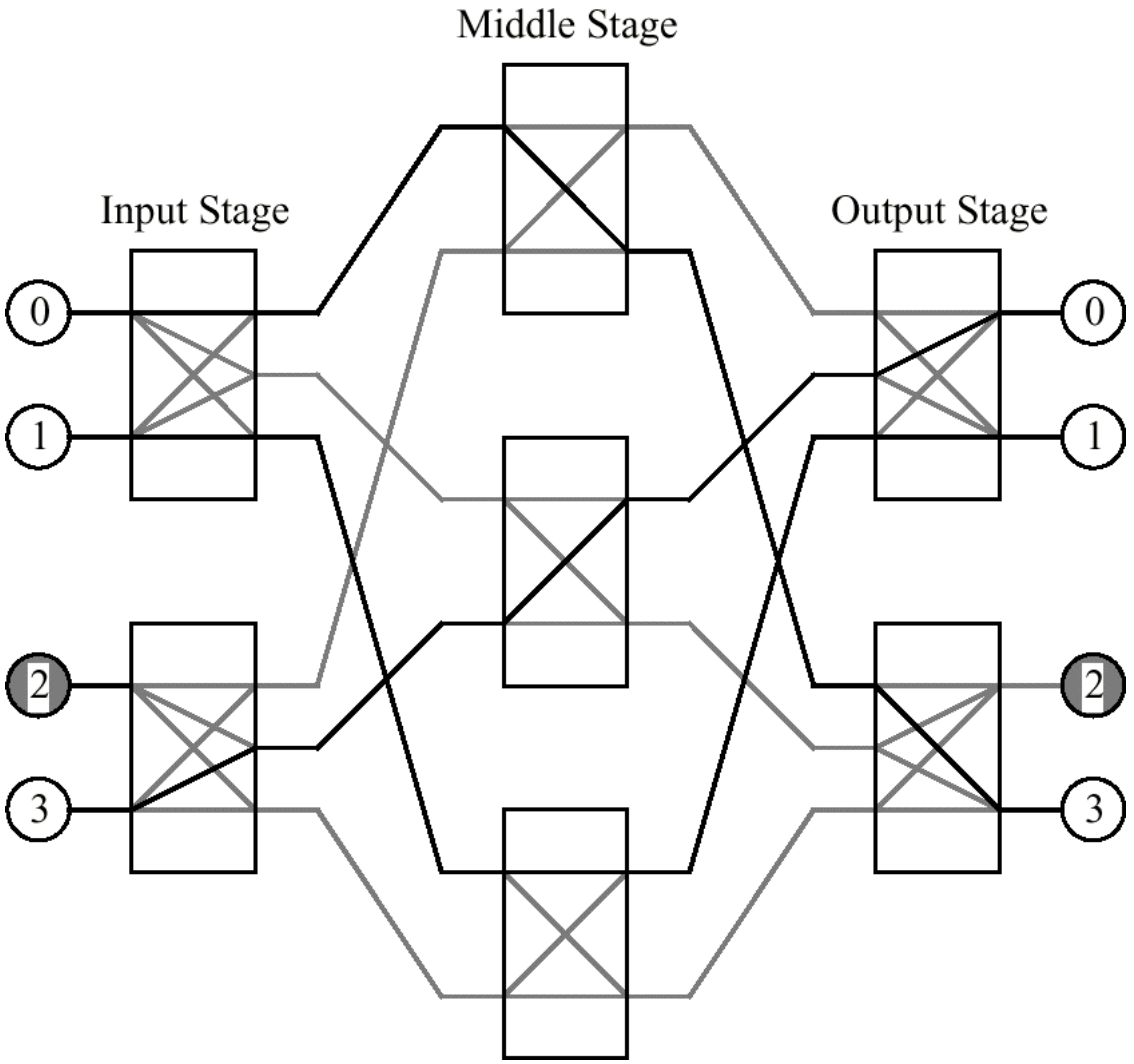
(b) ajustado para acomodar a conexão 1 → 1.



Bloqueio em Redes de Interconexão

- Rearranjável não-blocante
  - Conexões usadas são modificadas para acomodar novas conexões
- Estritamente não-blocante
  - Não há necessidade de rearranjar conexões nos cruzamentos para acrescentar novas conexões
  - Ex. *Crossbar*, Rede de Clos

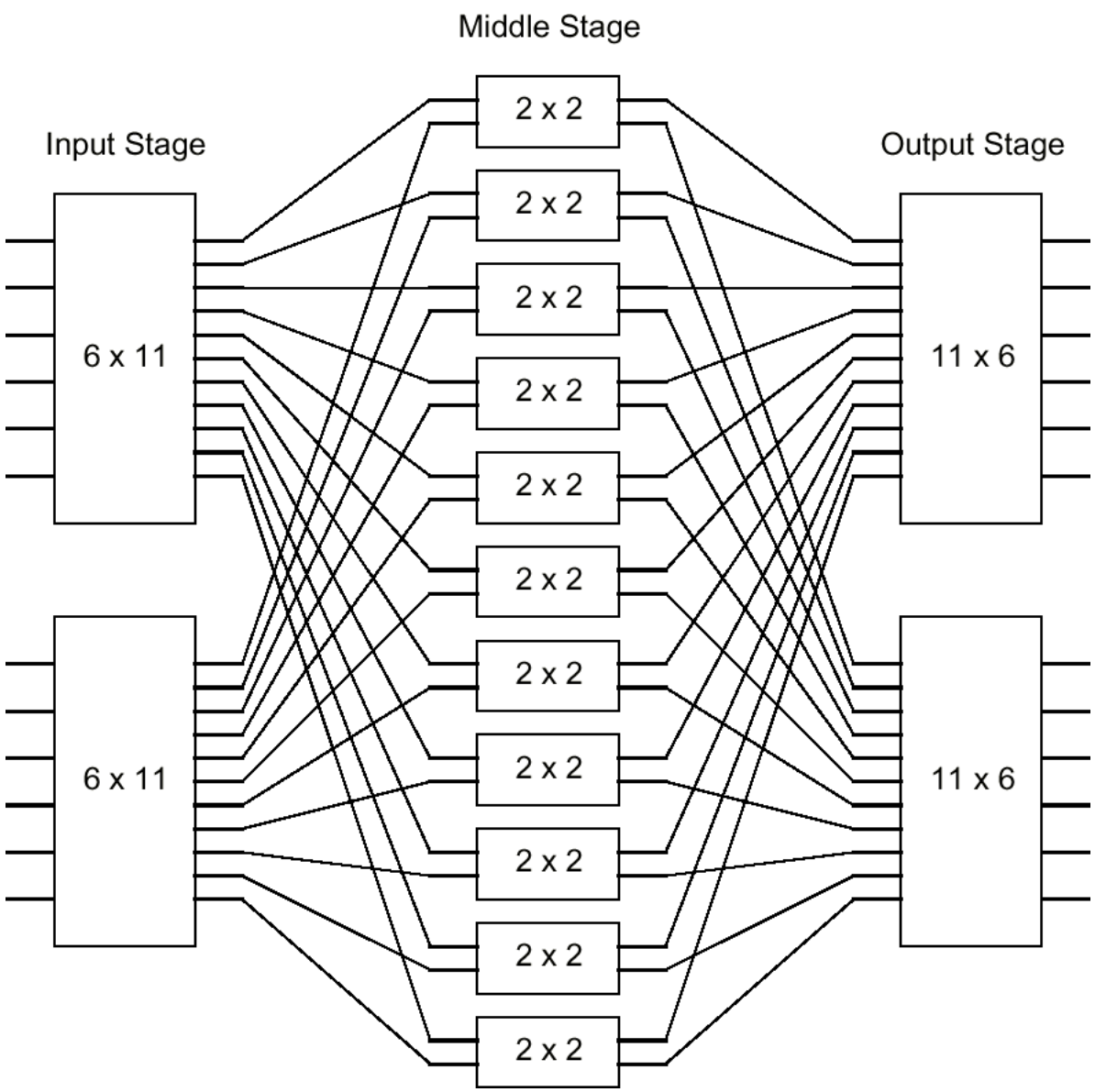
Rede de Clos de 3 Estágios



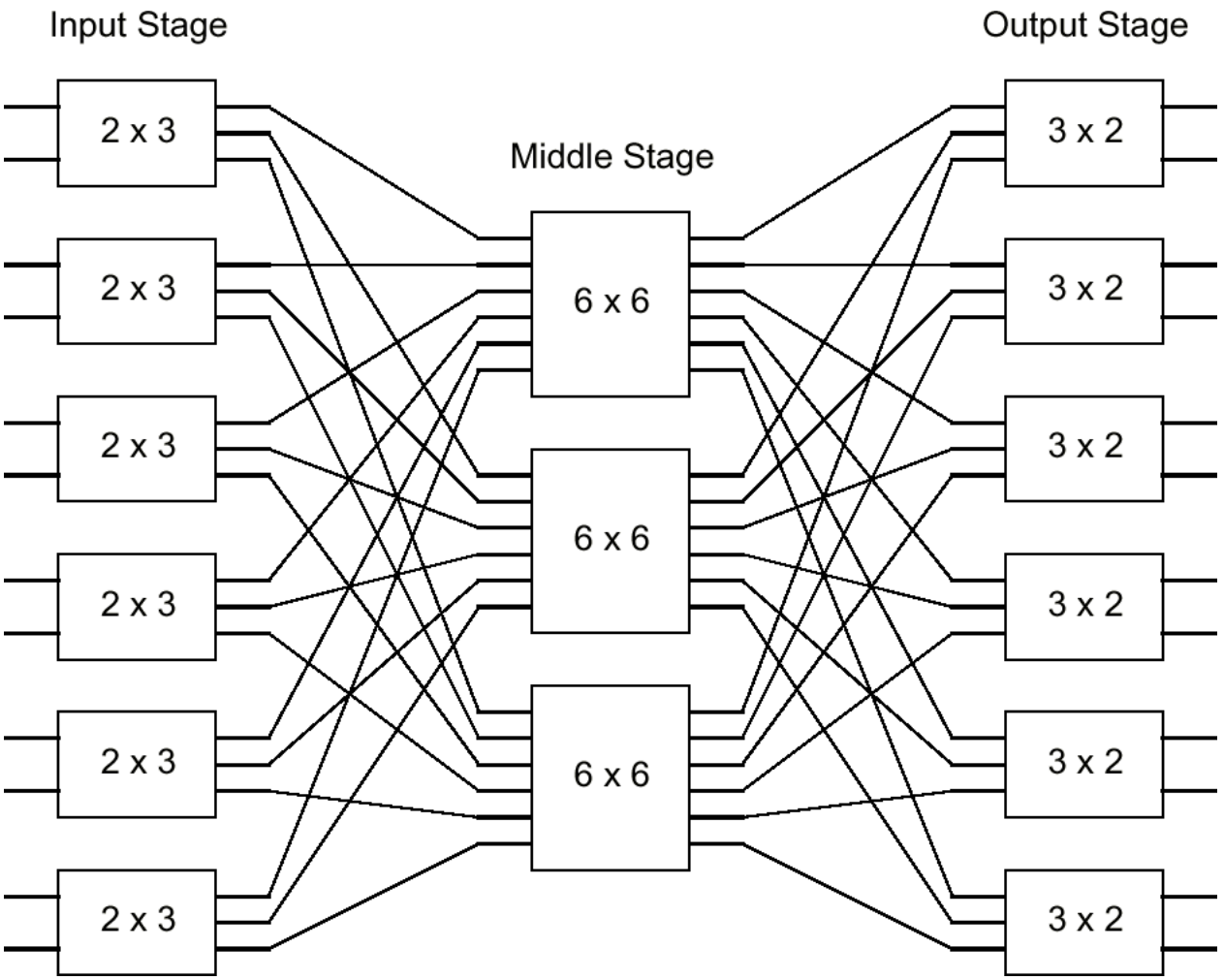
Parâmetros da Rede de Clos

- Dado: Número de canais (entradas e saídas)
- Escolhe-se
  - n – número de entradas de um cruzamento de entrada
  - p – número de saídas de um cruzamento de saída
- Resultados
  - Número de cruzamentos necessários no estágio intermediário
    - (n-1) + (p-1) + 1 = n + p – 1
  - Complexidade máxima de qualquer cruzamento

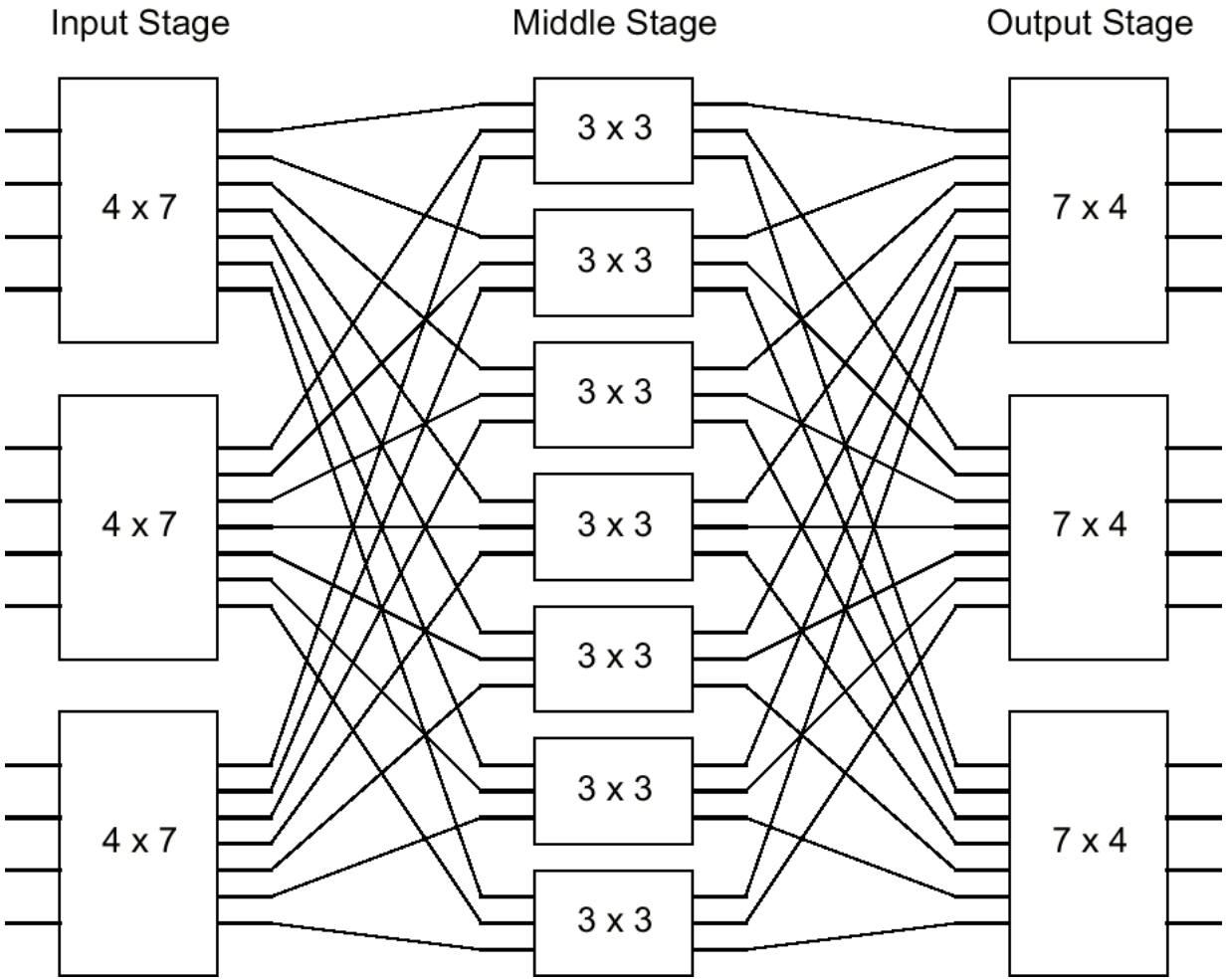
Rede de Clos de 12 Canais e 3 Estágios com $n = p = 6$



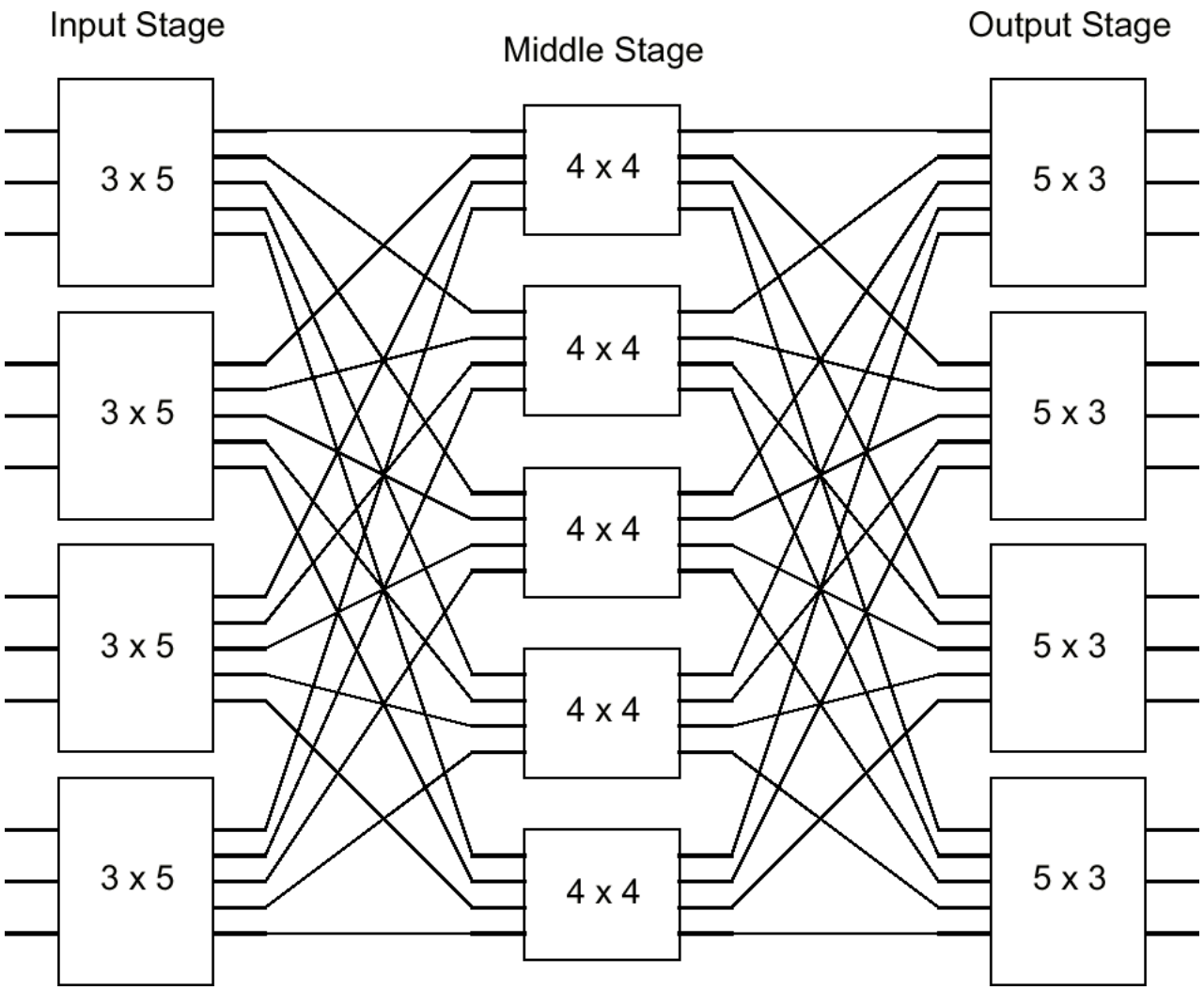
Rede de Clos de 12 Canais e 3 Estágios com $n = p = 2$



Rede de Clos de 12 Canais e 3 Estágios com $n = p = 4$



Rede de Clos de 12 Canais e 3 Estágios com $n = p = 3$



Mapeamento de um Algoritmo numa Arquitetura Paralela

- Análise de dependências
  - Seqüência de controle do programa
- Grafo de dependências para o programa
  - Cada operação = nodo do grafo
  - Cada seta = nodo que gera o resultado para nodo que precisa do mesmo

Função em C para Calcular $(x^2 + y^2) \times y^2$

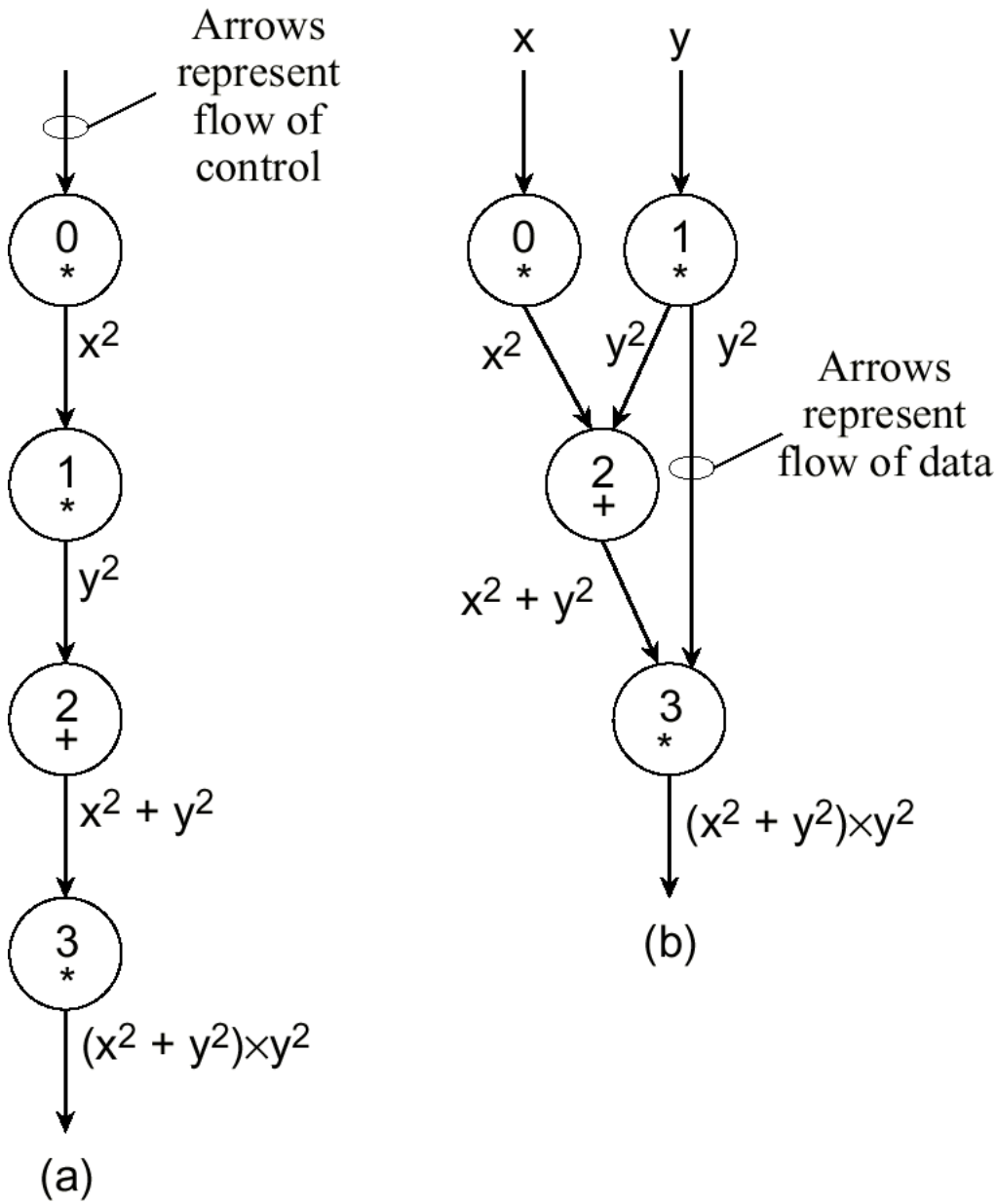
Operation numbers

```
func(x, y)  /* Compute (x² + y²) × y² */
int x, y;
{
int temp0, temp1, temp2, temp3;

0  temp0 = x * x;
1  temp1 = y * y;
2  temp2 = temp1 + temp2;
3  temp3 = temp1 * temp2;

return(temp3);
}
```

Grafo de Dependência

(a) seqüência de controle para o programa C;

(b) grafo de dependência para o programa C.



Multiplicação de Matrizes

(a) $$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

(b)

$$b_0 = \overset{0}{\boxed{a_{00}x_0}} + \overset{4}{\boxed{a_{01}x_1}} + \overset{6}{\boxed{a_{02}x_2}} + \overset{5}{\boxed{a_{03}x_3}}$$

$$b_1 = \overset{7}{\boxed{a_{10}x_0}} + \overset{11}{\boxed{a_{11}x_1}} + \overset{13}{\boxed{a_{12}x_2}} + \overset{12}{\boxed{a_{13}x_3}}$$

$$b_2 = \overset{14}{\boxed{a_{20}x_0}} + \overset{18}{\boxed{a_{21}x_1}} + \overset{20}{\boxed{a_{22}x_2}} + \overset{19}{\boxed{a_{23}x_3}}$$

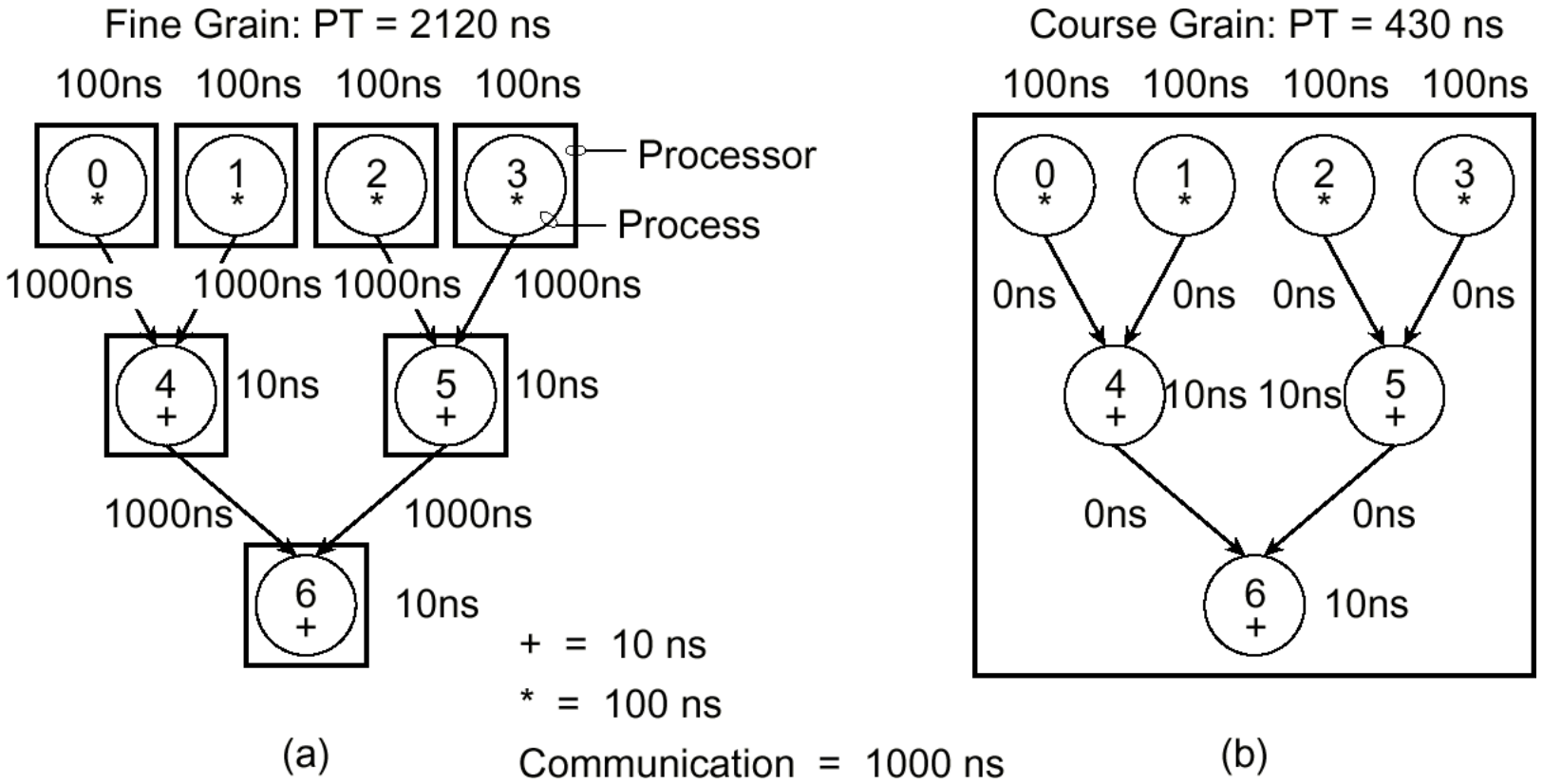
$$b_3 = \overset{21}{\boxed{a_{30}x_0}} + \overset{25}{\boxed{a_{31}x_1}} + \overset{27}{\boxed{a_{32}x_2}} + \overset{26}{\boxed{a_{33}x_3}}$$

(a) multiplicação a ser realizada: Ax = b;

(b) equações para calcular os b_i.

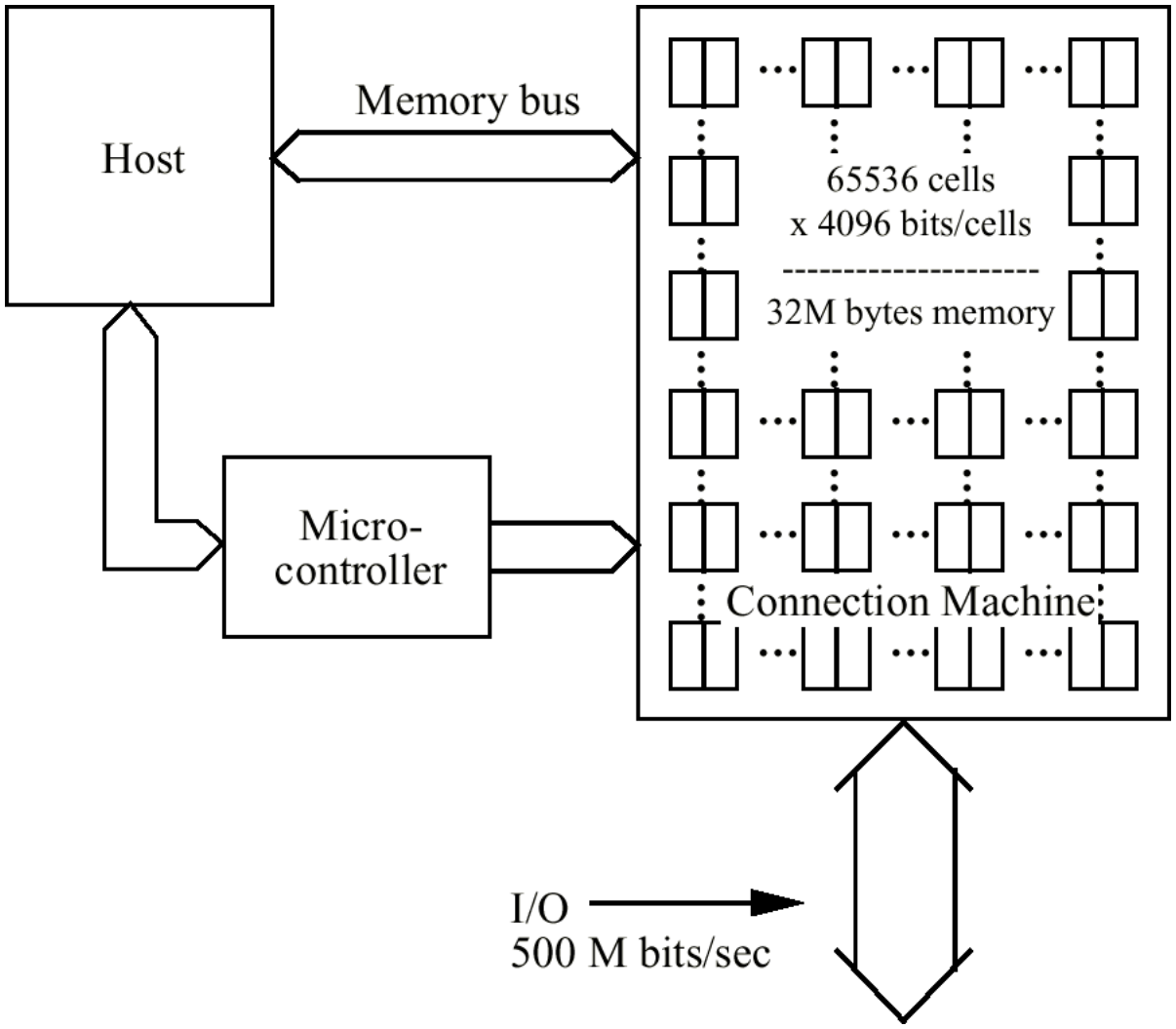
Grafo de Dependências para a Multiplicação de Matrizes

$$\frac{T_{Sequential}}{T_{Parallel}} = \frac{1720}{430} = 4$$



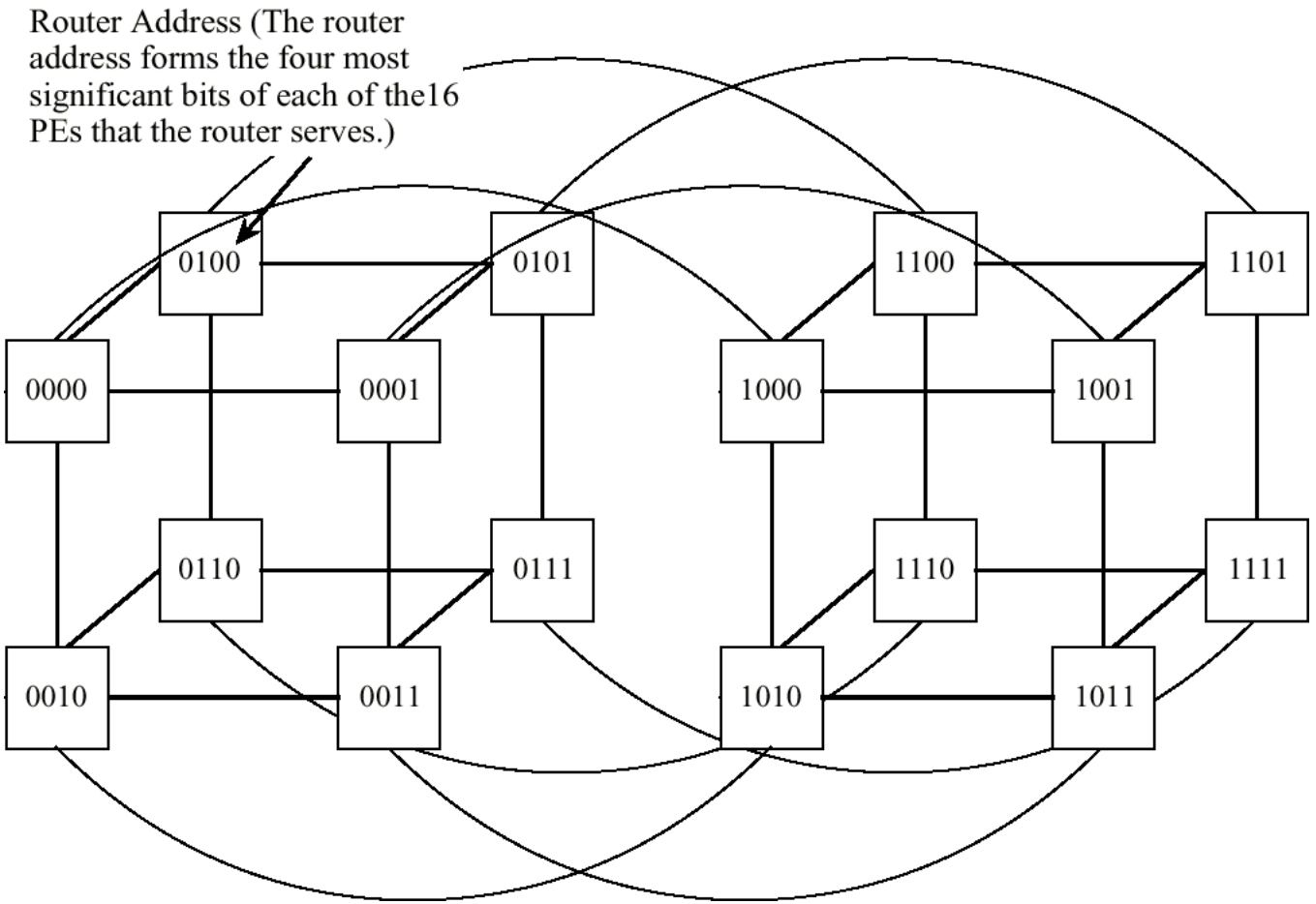
A Connection Machine CM-1

- **Diagrama de Blocos do CM-1 (Adaptado de Hillis, W. D., *The Connection Machine*, The MIT Press, 1985).**

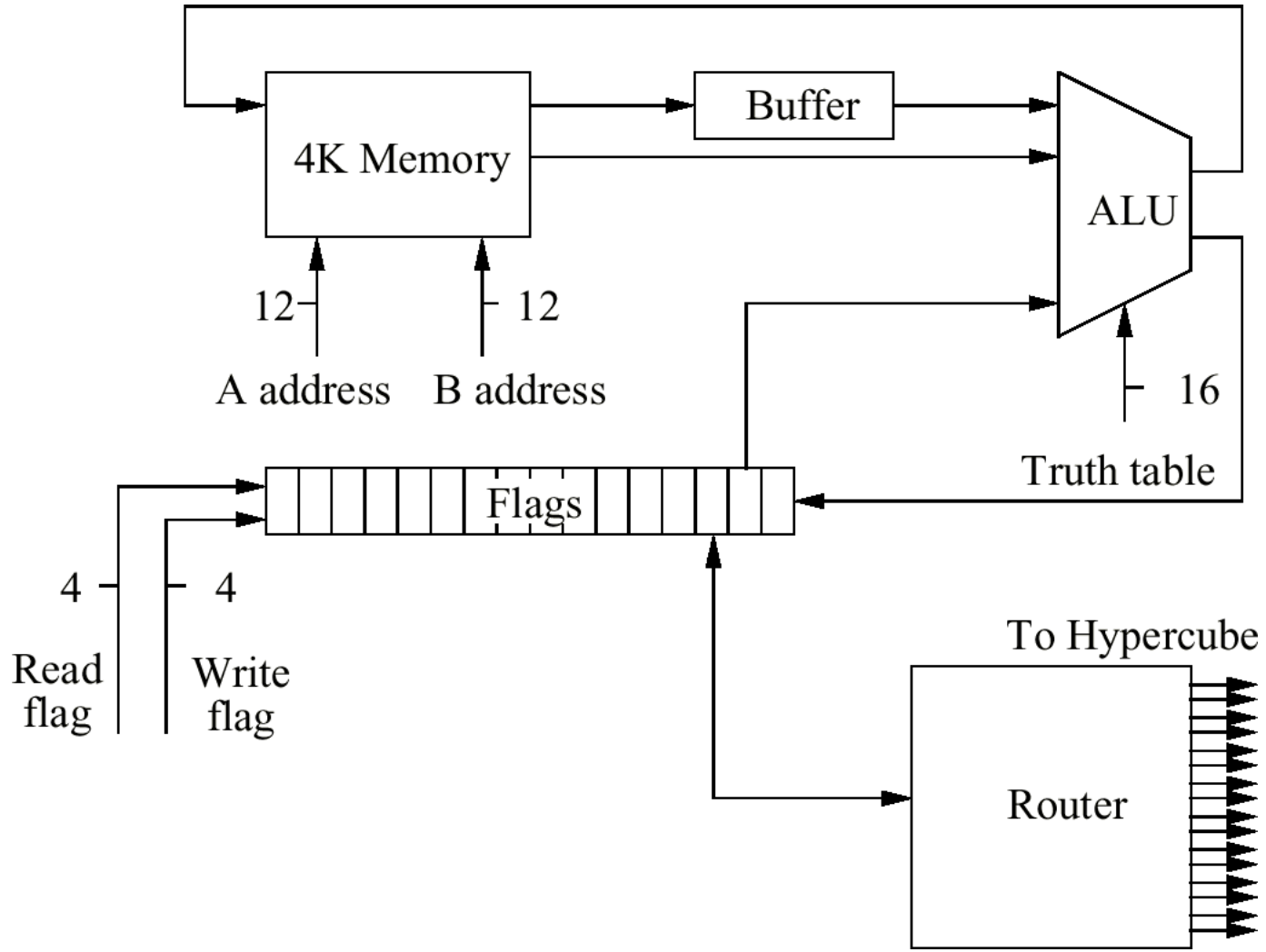


Rede de Roteadores do CM-1

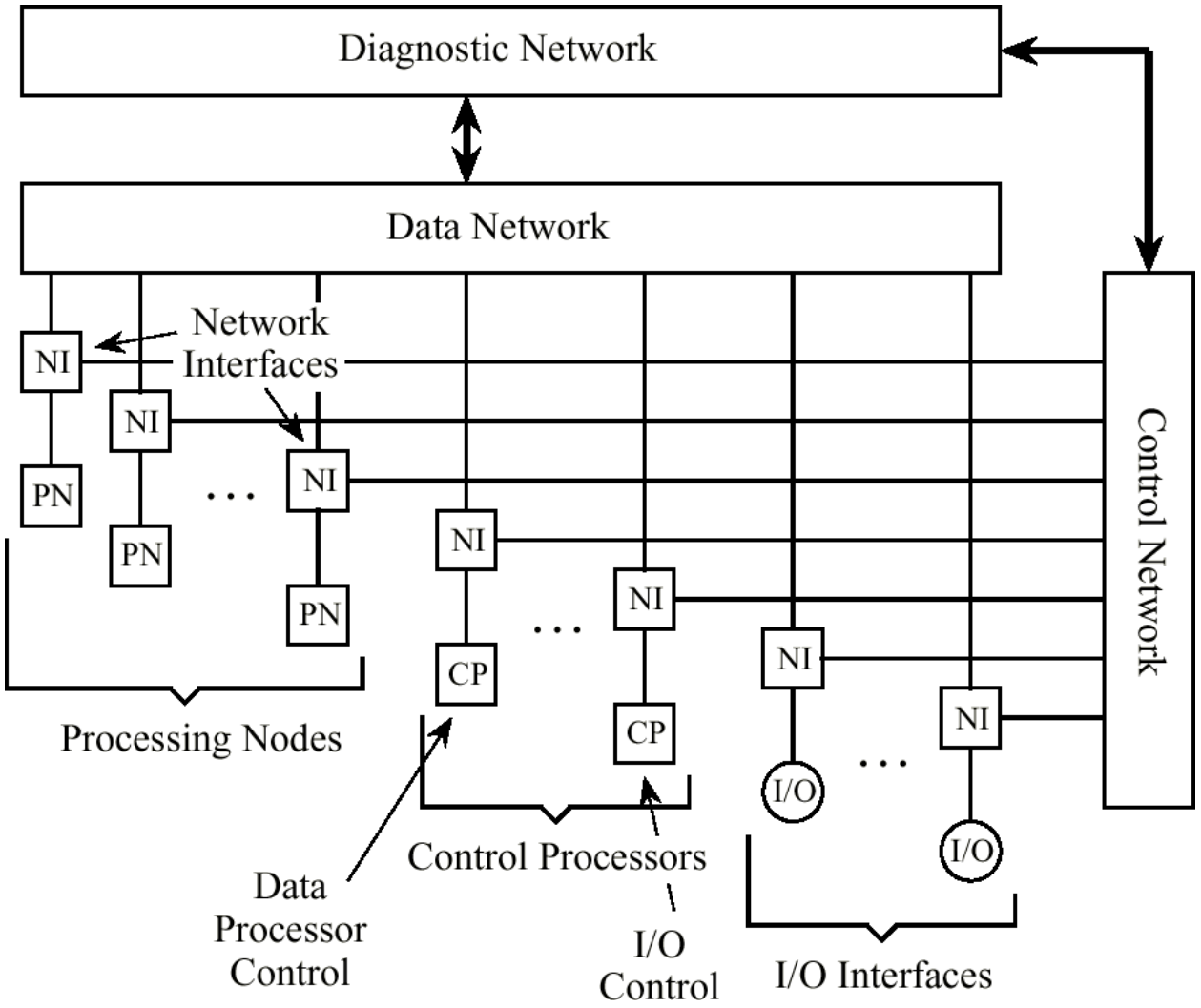
- **Hiper-cubo de 4 dimensões para a rede de roteadores.**



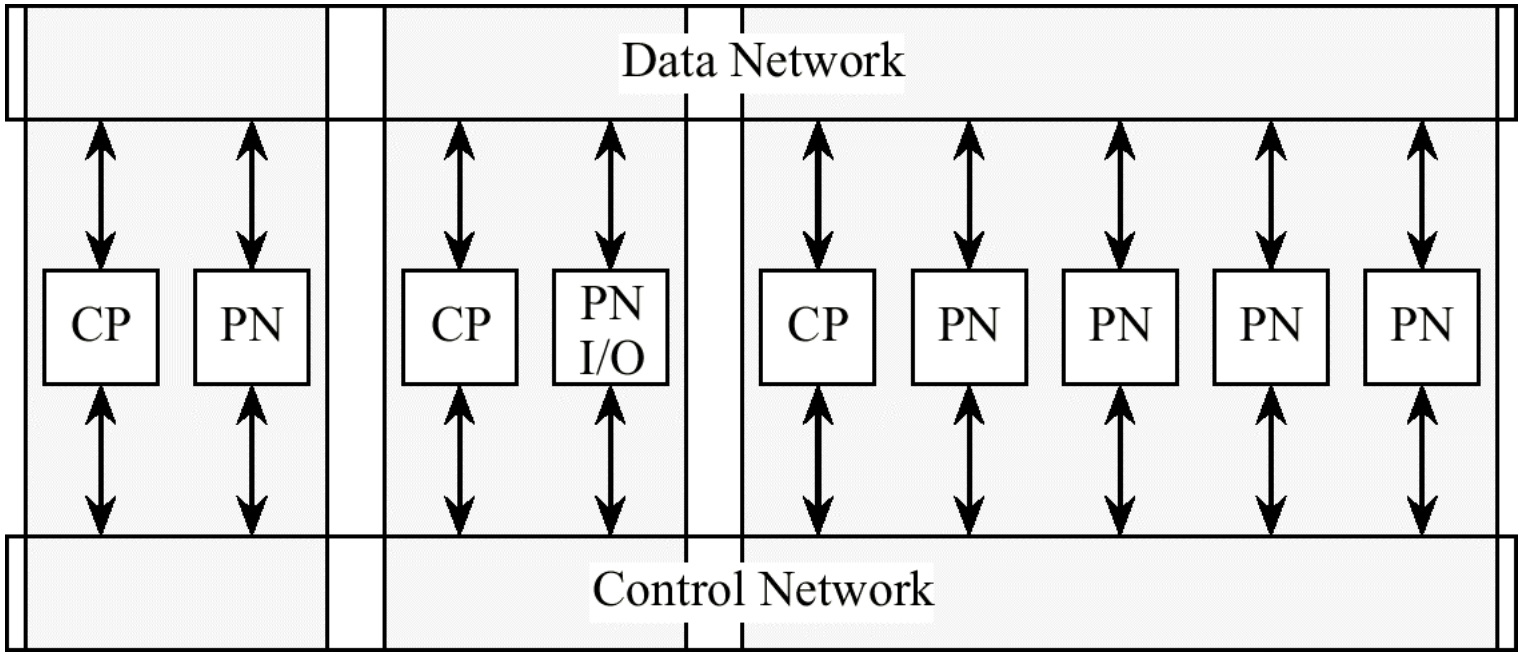
Elemento de Processamento CM-1



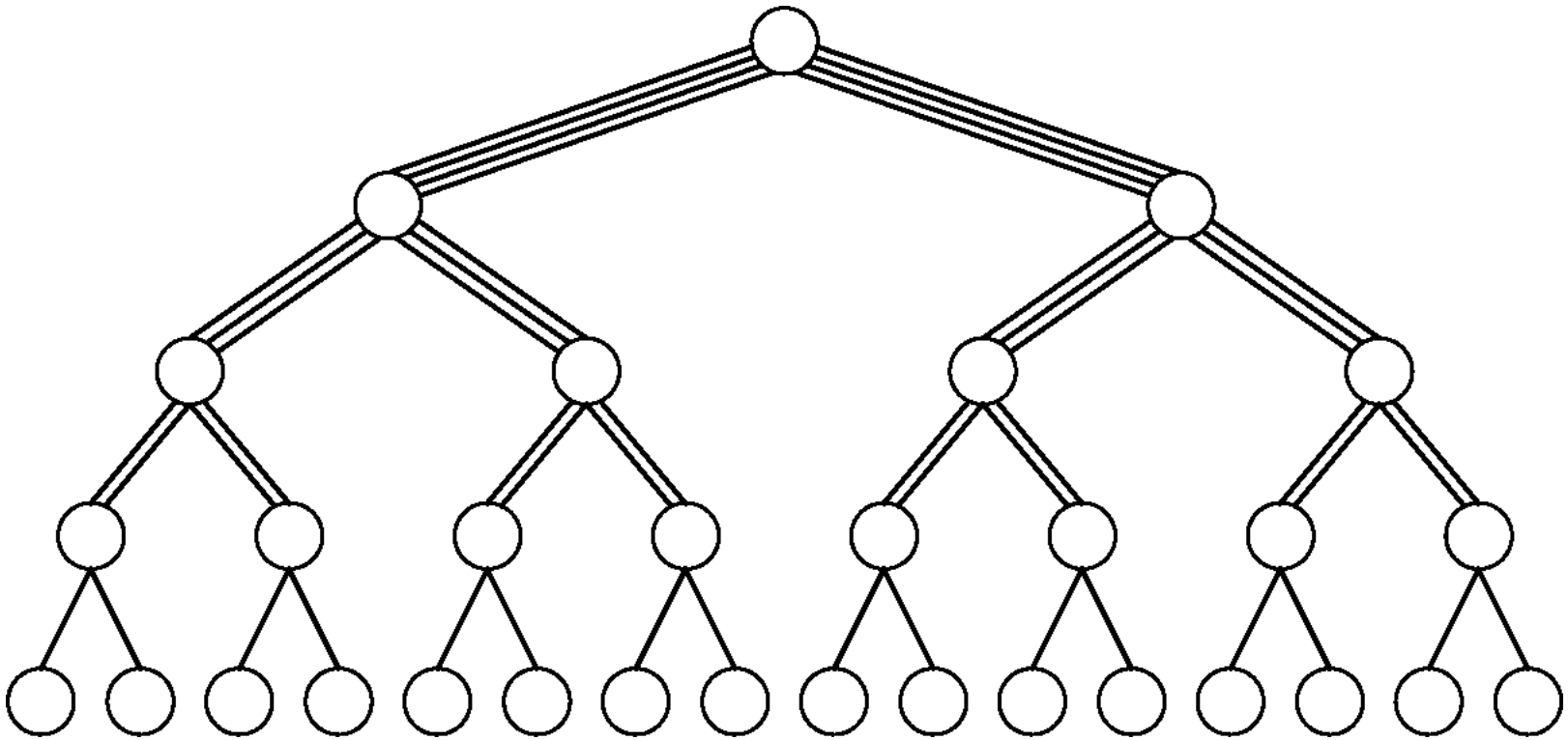
A Connection Machine CM-5



Partições no CM-5

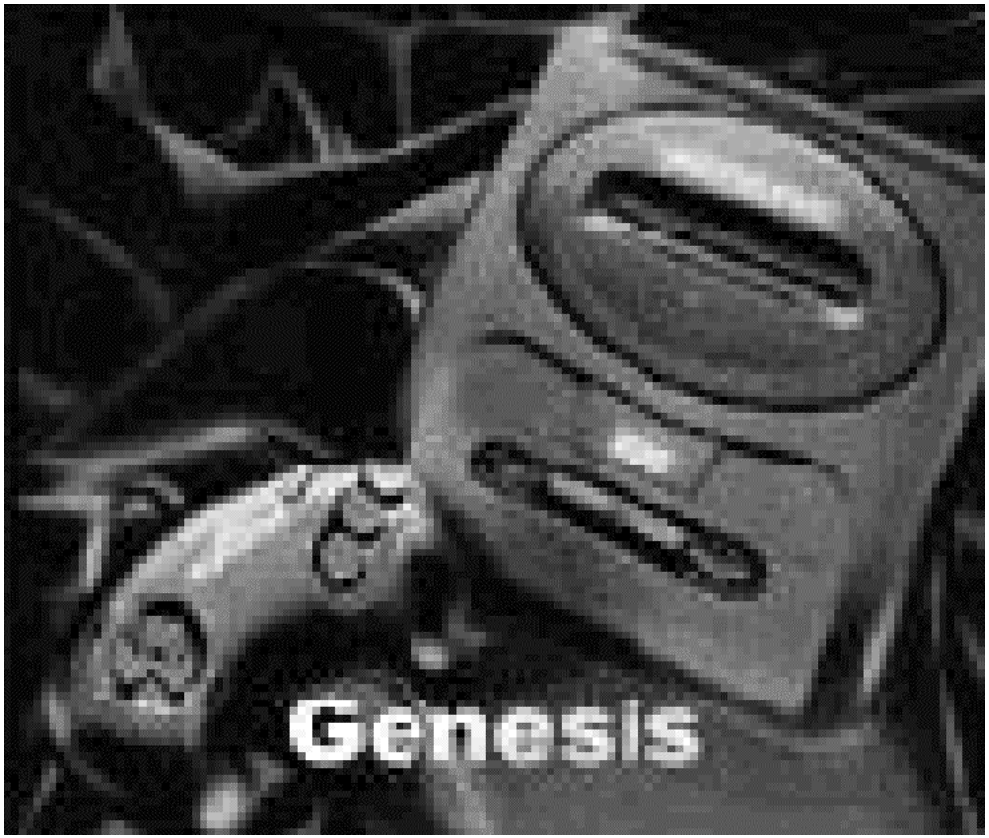


Árvore Gorda



Processamento Paralelo no Sega Genesis

- External view of the Sega Genesis home video game system.



A Arquitetura Sega Genesis

- **External view of the Sega Genesis home video game system.**