Organização de Computadores II
DCC007

Aula 16 – Paralelismo em Nível de Dados
Máquinas Vetoriais / Extensões SIMD

Prof. Omar Paranaiba Vilela Neto

Taxonomia Histórica
Classificação de Flynn

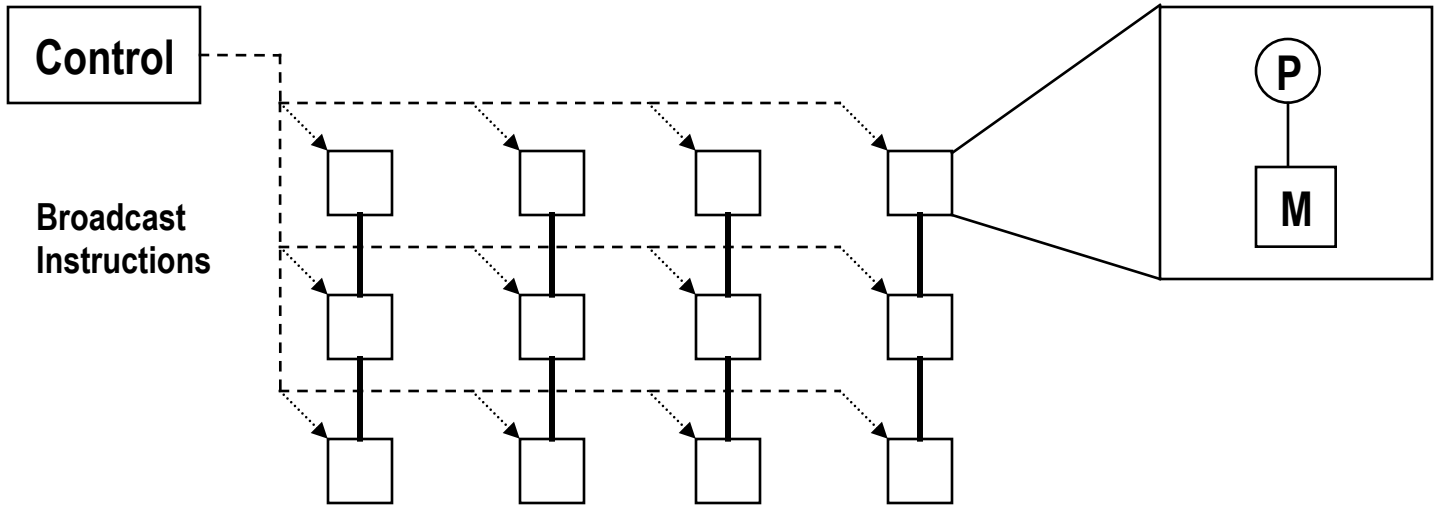
- Baseada nas noções de
  - ***Instruction streams***
  - ***Data streams***

- Organização da máquina é dada pela multiplicidade do hardware para manipular sequências independentes de dados e instruções

  - ***SISD***: Single Instruction and Single Data Stream
  - ***SIMD***: Single Instruction and Multiple Data Streams
  - ***MISD***: Multiple Instructions and Single Data Stream
  - ***MIMD***: Multiple Instructions and Multiple Data Streams

Máquinas SIMD

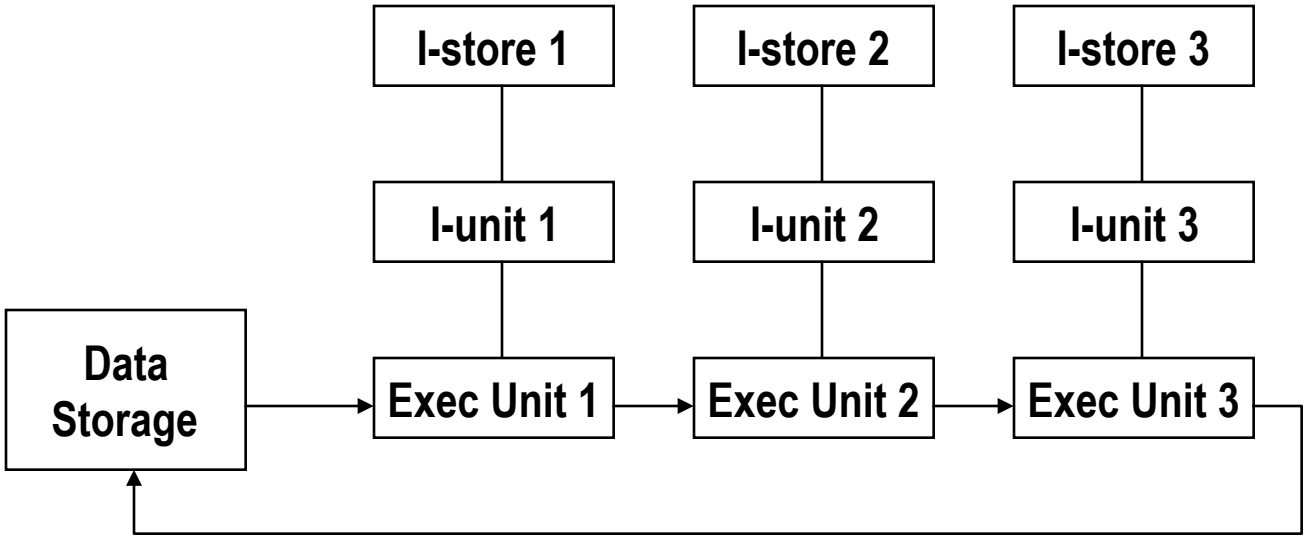
- Exemplos: GPGPU



- Vantagens: simplicidade de controle, custo, fácil de depurar, baixa latência
- Desvantagens: modelo restrito, pode desperdiçar recursos se somente poucos PEs são utilizados por instrução

Máquinas MISD

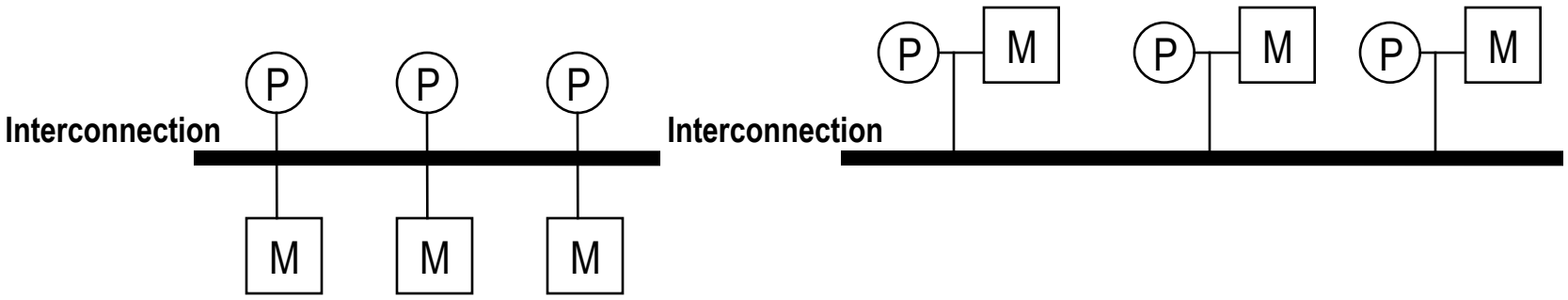
- Exemplo: arrays sistólicos (Warp de CMU)



- Vantagens: simples de projetar, custo-desempenho alto quando pode ser utilizado (processamento de sinais)
- Desvantagens: aplicabilidade limitada, difícil de programar

Máquinas MIMD

- Exemplo: SGI Challenge, SUN SparcCenter, Cray T3D, Multicores, Clusters....



- Vantagens: aplicabilidade
- Desvantagens: difícil de projetar bem, overhead de sincronização pode ser alto, programação pode ser difícil

Introdução

- Arquitetura SIMD pode explorar significativamente o **paralelismo em nível de dados** para:
  - matrix-oriented computação científica
  - media-oriented processadores de imagem e som

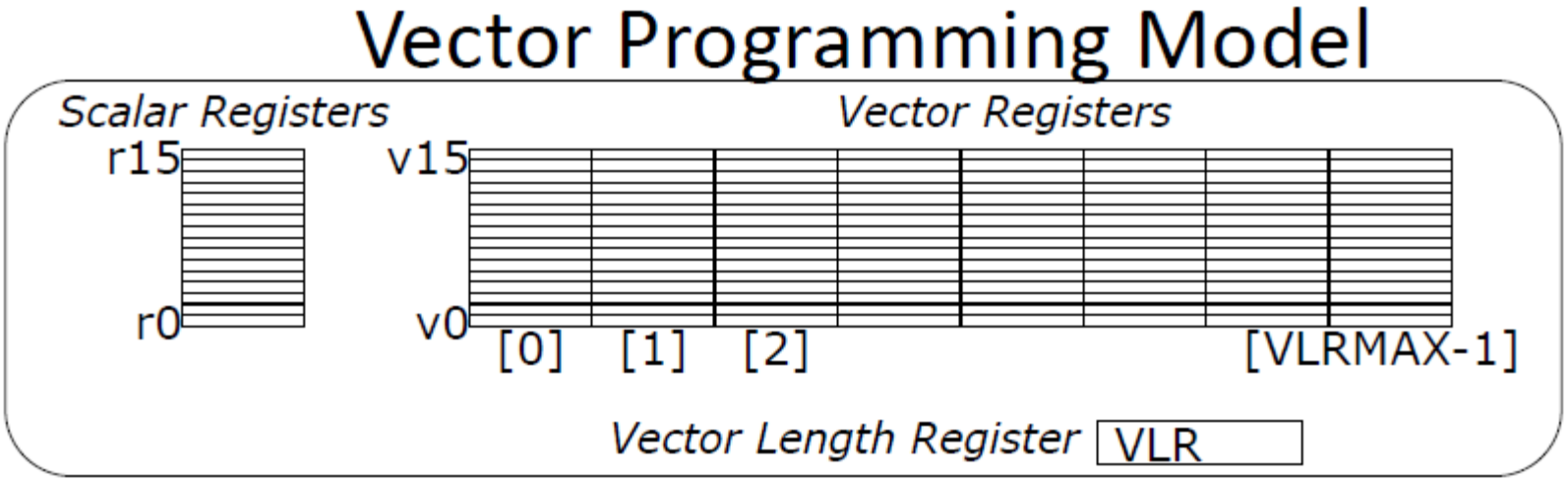
- SIMD é **energeticamente mais eficiente** que MIMD
  - Precisa apenas buscar uma instrução por operação de dados
  - É mais atrativo para aplicações embarcadas

- SIMD permite que o programador continue **pensando sequencialmente**

Paralelismo SIMD

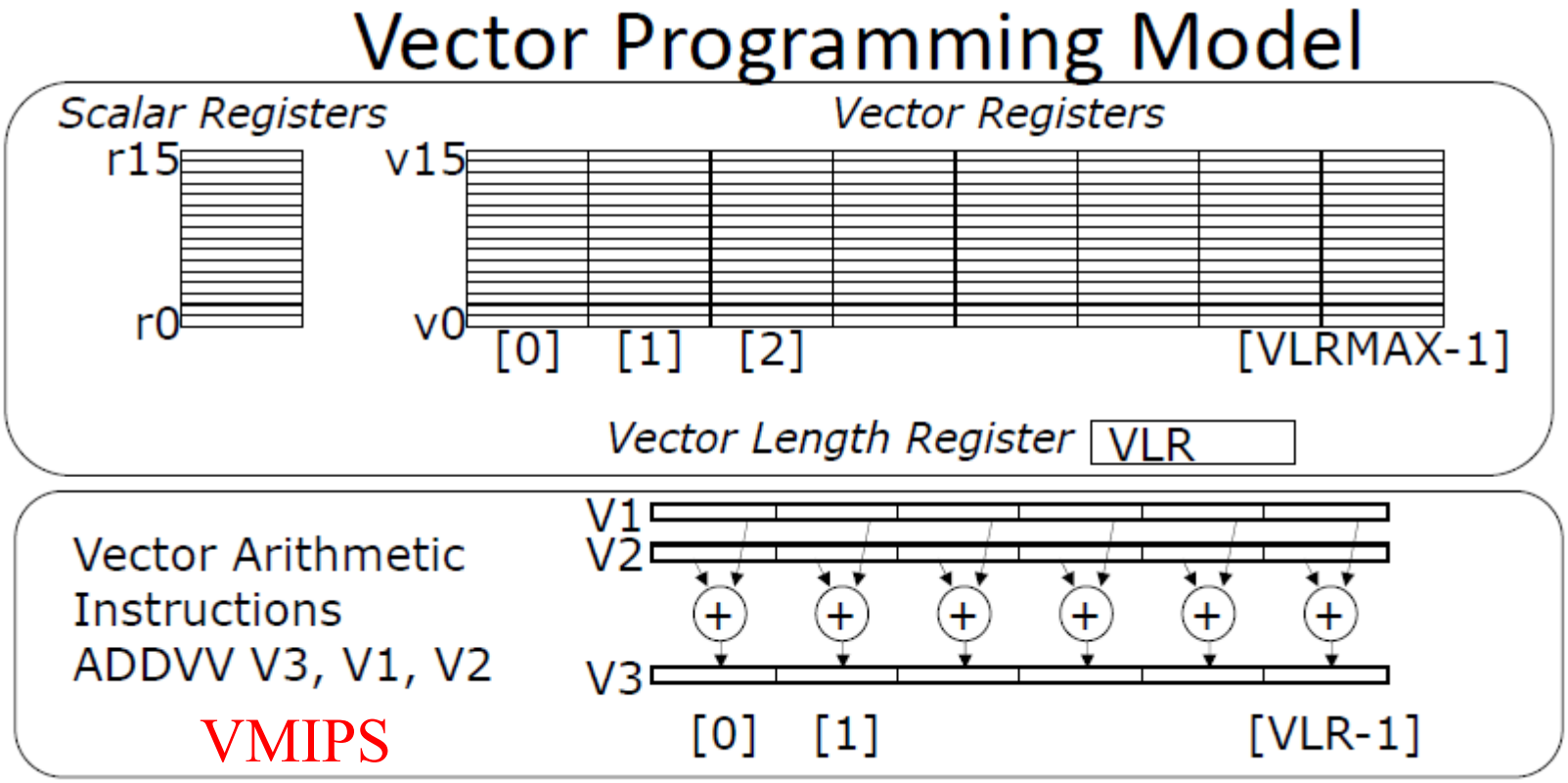
- Arquiteturas Vetoriais
- Extensões SIMD
- Graphics Processor Units (GPUs)

Introdução - Software

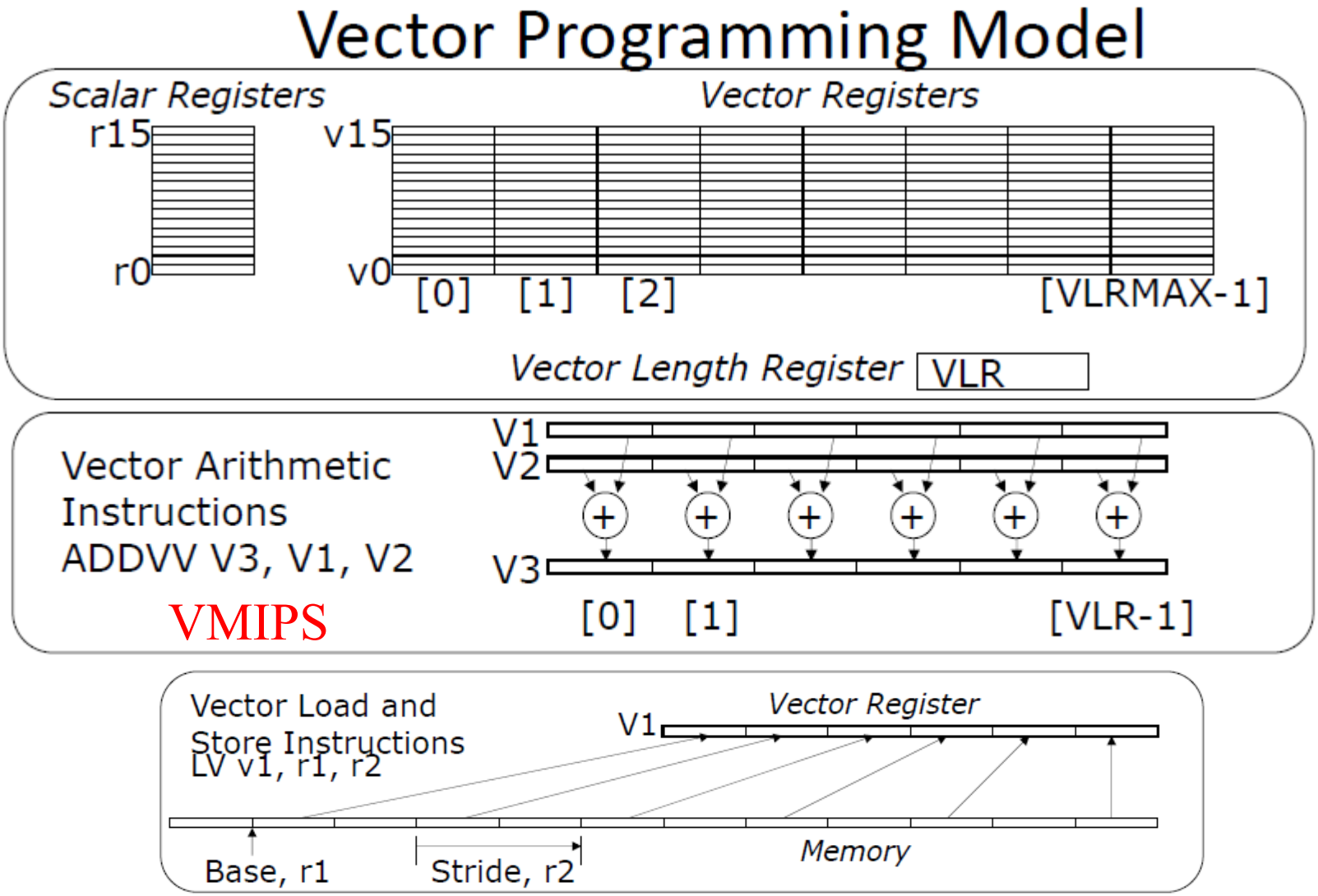


2 Tipos de Registradores

Introdução - Software



Introdução - Software



Código Vetorial

Multiplicação

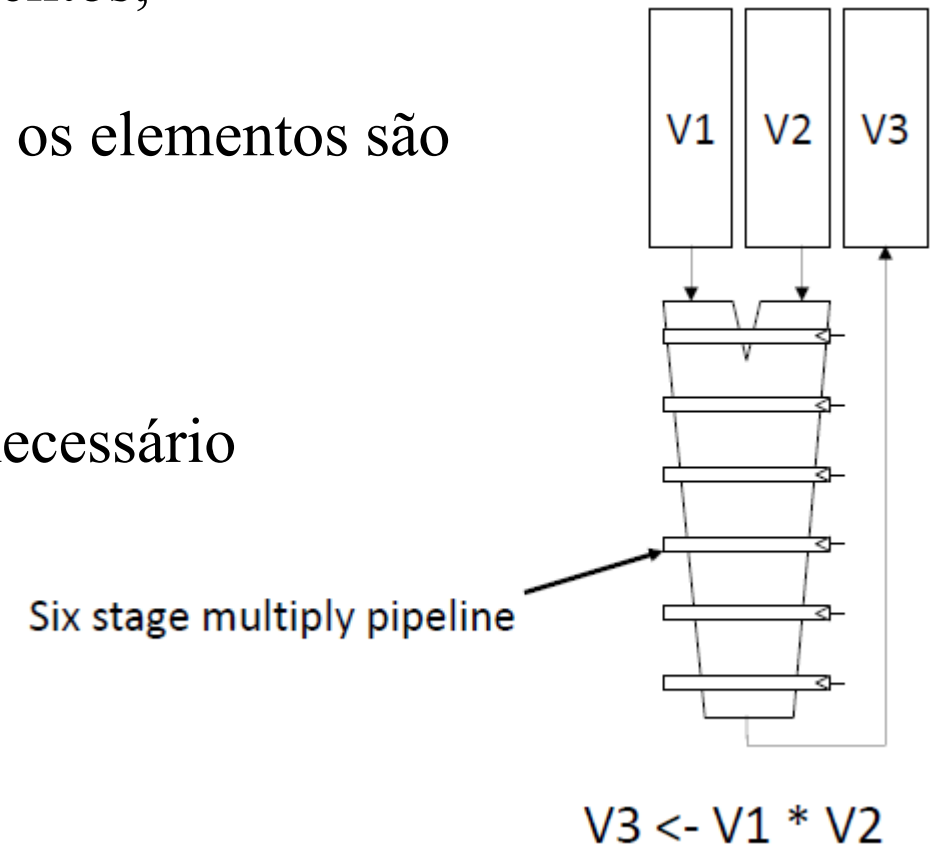
```
# C code
for (i=0; i<64; i++)
  C[i] = A[i] * B[i];
```

```
# Scalar Assembly Code
  LI R4, 64
loop:
  L.D F0, 0(R1)
  L.D F2, 0(R2)
  MUL.D F4, F2, F0
  S.D F4, 0(R3)
  DADDIU R1, 8
  DADDIU R2, 8
  DADDIU R3, 8
  DSUBIU R4, 1
  BNEZ R4, loop
```

```
# Vector Assembly Code
  LI VLR, 64
  LV V1, R1
  LV V2, R2
  MULVV.D V3, V1, V2
  SV V3, R3
```

Execução Aritmética Vetorial

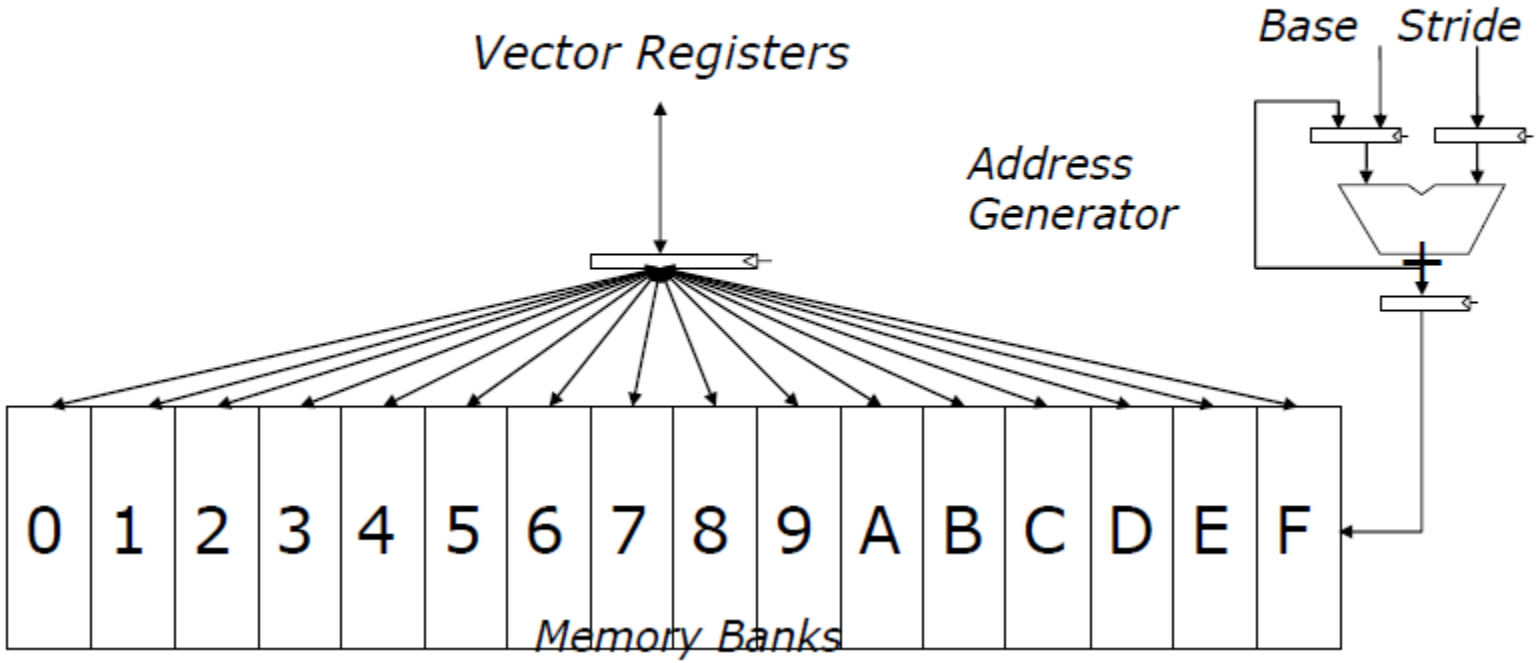
- Usa **Pipeline Profundo** (Clock rápido) para executar operações com elementos;
- **Simplifica o controle** porque os elementos são independentes
  - Sem hazards de dados;
  - Encaminhamento não necessário



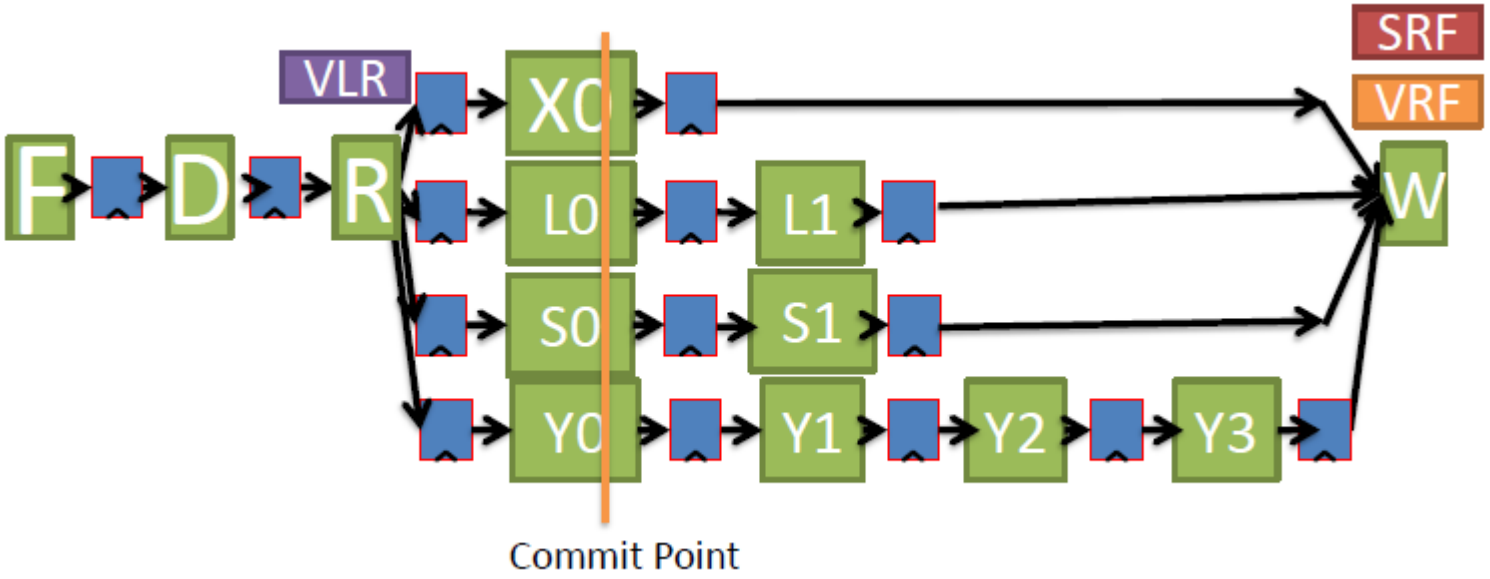
Sistema de Memória Intercalado

Cray-1, 16 banks, 4 cycle bank busy time, 12 cycle latency

- *Bank busy time*: Time before bank ready to accept next request



Exemplo de Arquitetura Vetorial

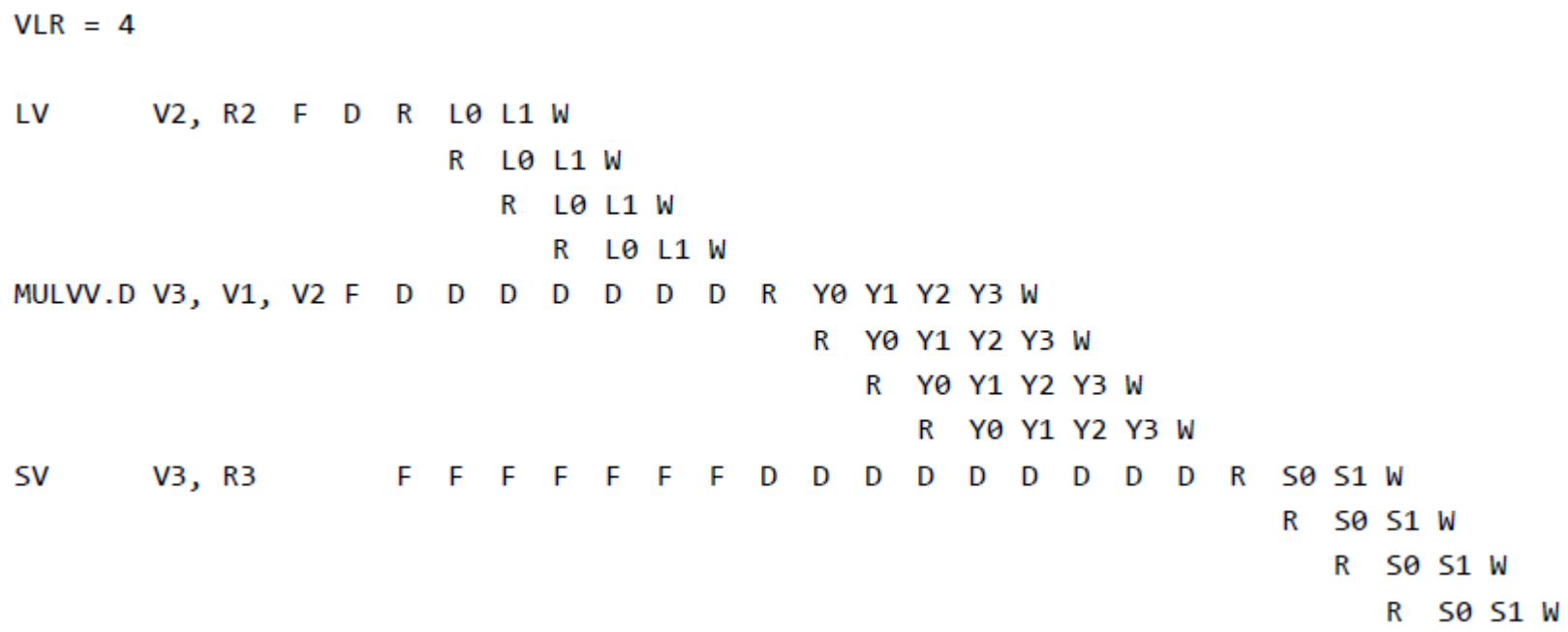


- SRF: Scalar Register File
- VRF: Vector Register File
- VLR: Vector Length Register

Exemplo de Arquitetura Vetorial

```
# C code
for (i=0; i<4; i++)
  C[i] = A[i] * B[i];
```

```
# Vector Assembly Code
  LI VLR, 4
  LV V1, R1
  LV V2, R2
  MULVV.D V3, V1, V2
  SV V3, R3
```



Exemplo de Arquitetura Vetorial

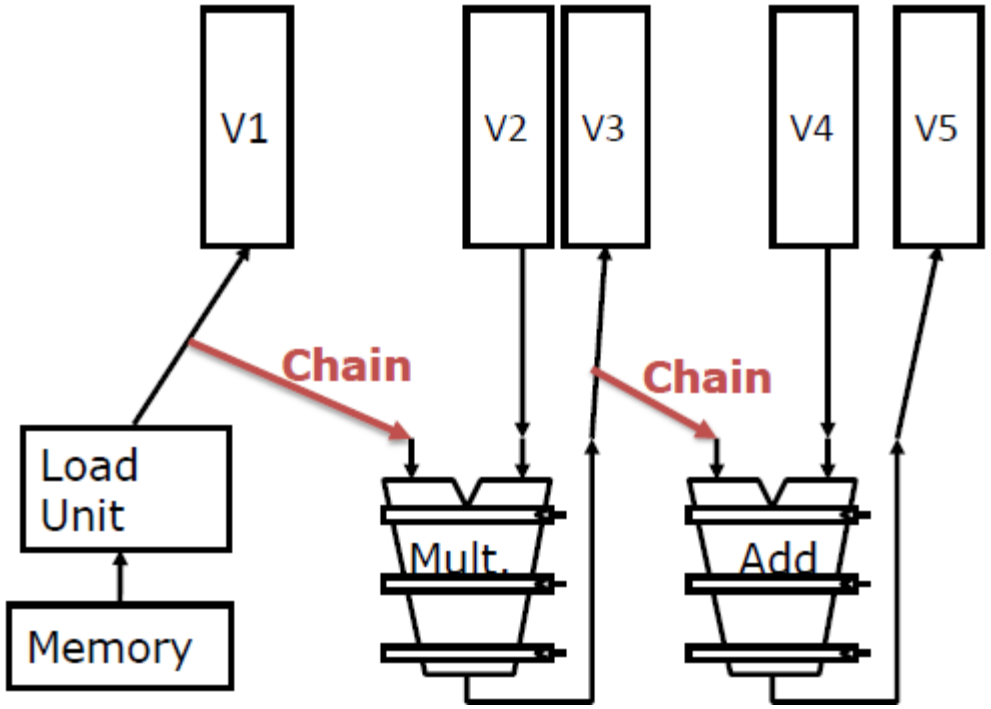
```
# C code
for (i=0; i<4; i++)
  C[i] = A[i] * B[i];
```

```
# Vector Assembly Code
  LI VLR, 4
  LV V1, R1
  LV V2, R2
  MULVV.D V3, V1, V2
  SV V3, R3
```

Otimizando o Hardware – Encadeamento

- Encaminhamento na versão vetorial
  - Introduzida no Cray-1

```
LV    V1
MULVV V3,V1,v2
ADDVV V5,V3, v4
```



Otimizando o Hardware – Encadeamento: Vantagem

- **Sem encadeamento**: tem que esperar até o último elemento.

- **Com encadeamento**: pode começar assim que o primeiro dado é disponibilizado..

Otimizando o Hardware – Encadeamento: Vantagem

```
# C code
for (i=0; i<4; i++)
  C[i] = A[i] * B[i];
```

Exemplo antigo

```
# Vector Assembly Code
  LI VLR, 4
  LV V1, R1
  LV V2, R2
  MULVV.D V3, V1, V2
  SV V3, R3
```

```
VLR = 4

LV      V2, R2   F  D  R  L0 L1 W
                        R  L0 L1 W
                           R  L0 L1 W
                              R  L0 L1 W
MULVV.D V3, V1, V2 F  D  D  D  D  D  D  D  R  Y0 Y1 Y2 Y3 W
                                              R  Y0 Y1 Y2 Y3 W
                                                 R  Y0 Y1 Y2 Y3 W
                                                    R  Y0 Y1 Y2 Y3 W
SV      V3, R3         F  F  F  F  F  F  F  D  D  D  D  D  D  D  D  D  R  S0 S1 W
                                                                        R  S0 S1 W
                                                                           R  S0 S1 W
                                                                              R  S0 S1 W
```

Otimizando o Hardware – Encadeamento: Register File

```
# C code
for (i=0; i<4; i++)
  C[i] = A[i] * B[i];
```

Exemplo novo

```
# Vector Assembly Code
  LI VLR, 4
  LV V1, R1
  LV V2, R2
  MULVV.D V3, V1, V2
  SV V3, R3
```

```
VLR = 4

LV      V2, R2   F   D   R   L0 L1 W
                         R   L0 L1 W
                             R   L0 L1 W
                                 R   L0 L1 W
MULVV.D V3, V1, V2 F   D   D   D   D   R   Y0 Y1 Y2 Y3 W
                                       R   Y0 Y1 Y2 Y3 W
                                           R   Y0 Y1 Y2 Y3 W
                                               R   Y0 Y1 Y2 Y3 W
SV      V3, R3         F   F   F   F   D   D   D   D   D   D   R   S0 S1 W
                                                               R   S0 S1 W
                                                                   R   S0 S1 W
                                                                       R   S0 S1 W
```

Otimizando o Hardware – Encadeamento: Encaminhamento

```
# C code
for (i=0; i<4; i++)
  C[i] = A[i] * B[i];
```

Exemplo novo

```
# Vector Assembly Code
  LI VLR, 4
  LV V1, R1
  LV V2, R2
  MULVV.D V3, V1, V2
  SV V3, R3
```

```
VLR = 4

LV      V2, R2   F   D   R   L0  L1  W
                         R   L0  L1  W
                             R   L0  L1  W
                                 R   L0  L1  W
MULVV.D V3, V1, V2 F   D   D   D   D   R   Y0  Y1  Y2  Y3  W
                                       R   Y0  Y1  Y2  Y3  W
                                           R   Y0  Y1  Y2  Y3  W
                                               R   Y0  Y1  Y2  Y3  W
SV      V3, R3         F   F   F   F   D   D   D   D   D   D   R   S0  S1  W
                                                               R   S0  S1  W
                                                                   R   S0  S1  W
                                                                       R   S0  S1  W
```

Sem Alterações

Otimizando o Hardware – Encadeamento: Encaminhamento

```
# C code
for (i=0; i<4; i++)
  C[i] = A[i] * B[i];
```

Exemplo novo
Mais portas RF

```
# Vector Assembly Code
  LI VLR, 4
  LV V1, R1
  LV V2, R2
  MULVV.D V3, V1, V2
  SV V3, R3
```

```
   VLR = 4

   LV      V2, R2   F   D   R   L0 L1 W
                        R   L0 L1 W
                            R   L0 L1 W
                                R   L0 L1 W
   MULVV.D V3, V1, V2 F   D   D   R   Y0 Y1 Y2 Y3 W
                                R   Y0 Y1 Y2 Y3 W
                                    R   Y0 Y1 Y2 Y3 W
                                        R   Y0 Y1 Y2 Y3 W
   SV      V3, R3         F   F   D   D   D   D   R   S0 S1 W
                                                R   S0 S1 W
                                                    R   S0 S1 W
                                                        R   S0 S1 W
```

Boa melhora

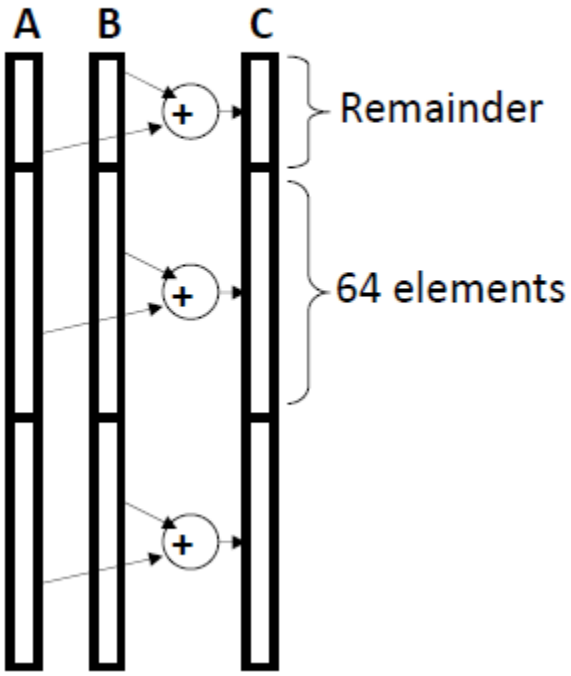
Otimizando o Hardware – Encadeamento: Encaminhamento

```
VLR = 8
LV      V2, R2   F  D  R  L0 L1 W
                        R  L0 L1 W
                           R  L0 L1 W
                              R  L0 L1 W
                                 R  L0 L1 W
                                    R  L0 L1 W
                                       R  L0 L1 W
                                          R  L0 L1 W
MULVV.D V3, V1, V2 F  D  D  R  Y0 Y1 Y2 Y3 W
                              R  Y0 Y1 Y2 Y3 W
                                 R  Y0 Y1 Y2 Y3 W
                                    R  Y0 Y1 Y2 Y3 W
                                       R  Y0 Y1 Y2 Y3 W
                                          R  Y0 Y1 Y2 Y3 W
                                             R  Y0 Y1 Y2 Y3 W
                                                R  Y0 Y1 Y2 Y3 W
SV      V3, R3         F  F  D  D  D  D  R  S0 S1 W
                                          R  S0 S1 W
                                             R  S0 S1 W
                                                R  S0 S1 W
                                                   R  S0 S1 W
                                                      R  S0 S1 W
                                                         R  S0 S1 W
                                                            R  S0 S1 W
```

Otimizando o Hardware – Stripmining

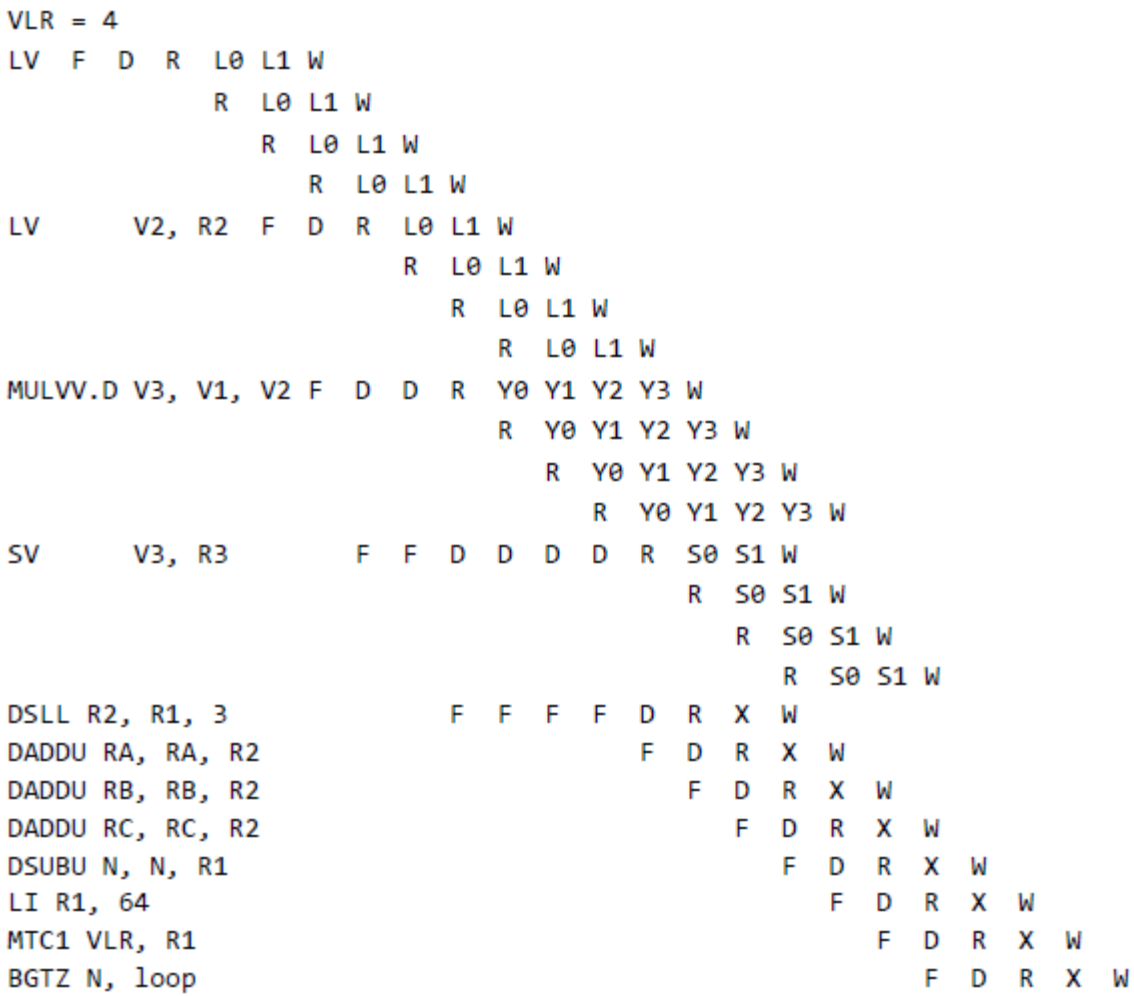
- **Problema**: Registradores Vetoriais têm tamanho finito
- **Solução**: Quebra os loops em pedaços que caibam nos registradores.

```
for (i=0; i<N; i++)
    C[i] = A[i]*B[i];
```



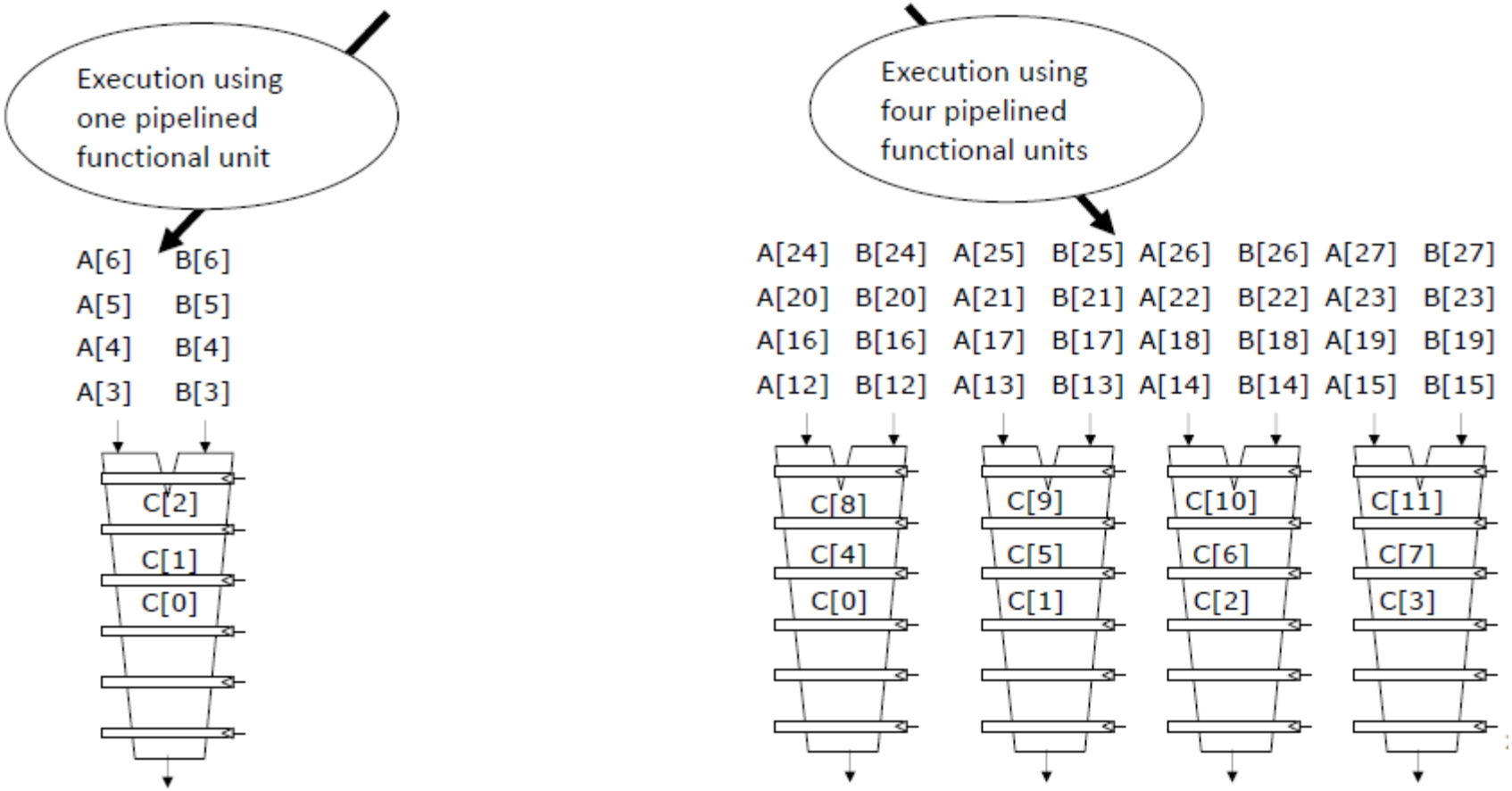

```
 ANDI R1, N, 63    # N mod 64
 MTC1 VLR, R1      # Do remainder
loop:
 LV V1, RA
 LV V2, RB
 MULVV.D V3, V1, V2
 SV V3, RC
 DSLL R2, R1, 3 # Multiply by 8
 DADDU RA, RA, R2 # Bump pointer
 DADDU RB, RB, R2
 DADDU RC, RC, R2
 DSUBU N, N, R1 # Subtract elements
 LI R1, 64
 MTC1 VLR, R1    # Reset full length
 BGTZ N, loop    # Any more to do?
```

Otimizando o Hardware – Stripmining



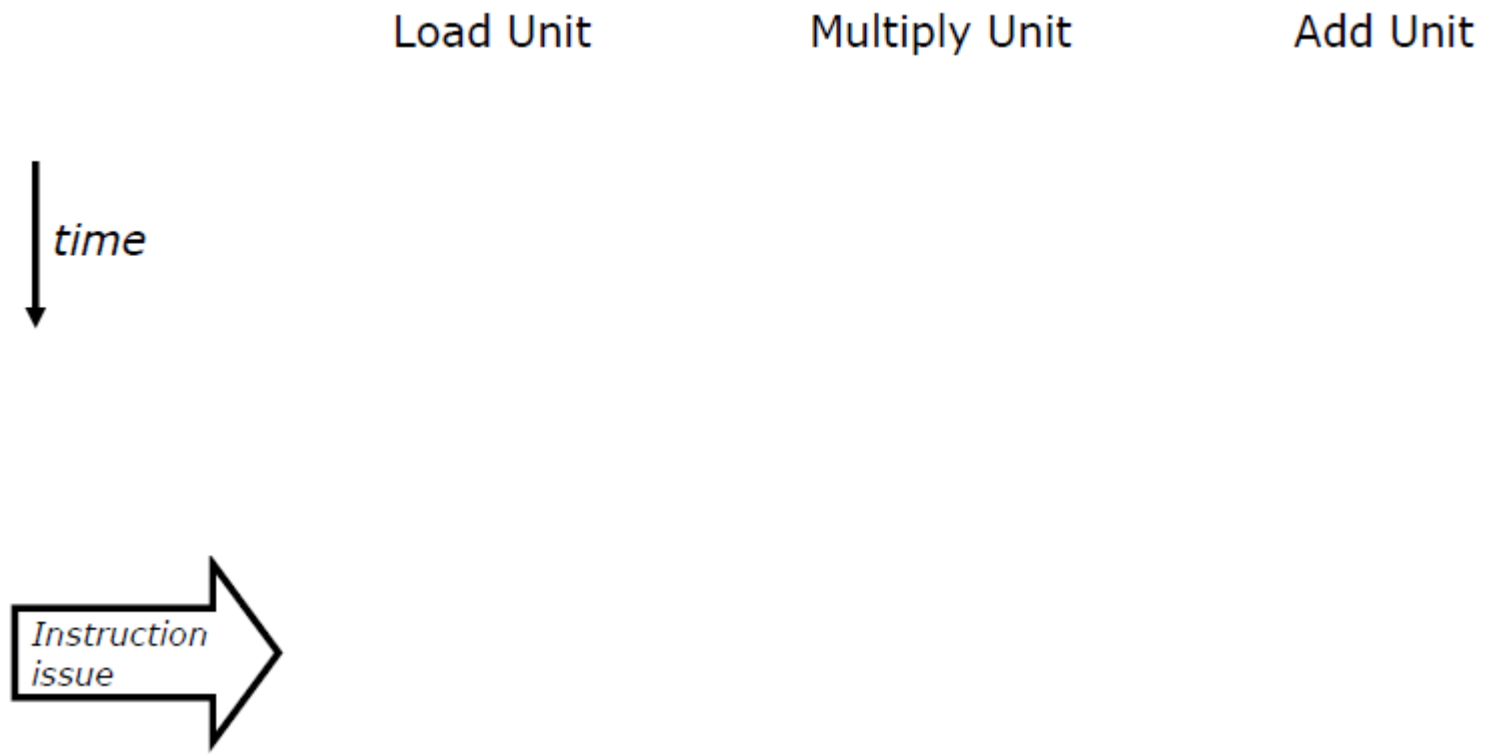
Paralelismo de Instrução Vetorial

MULVV C,A,B



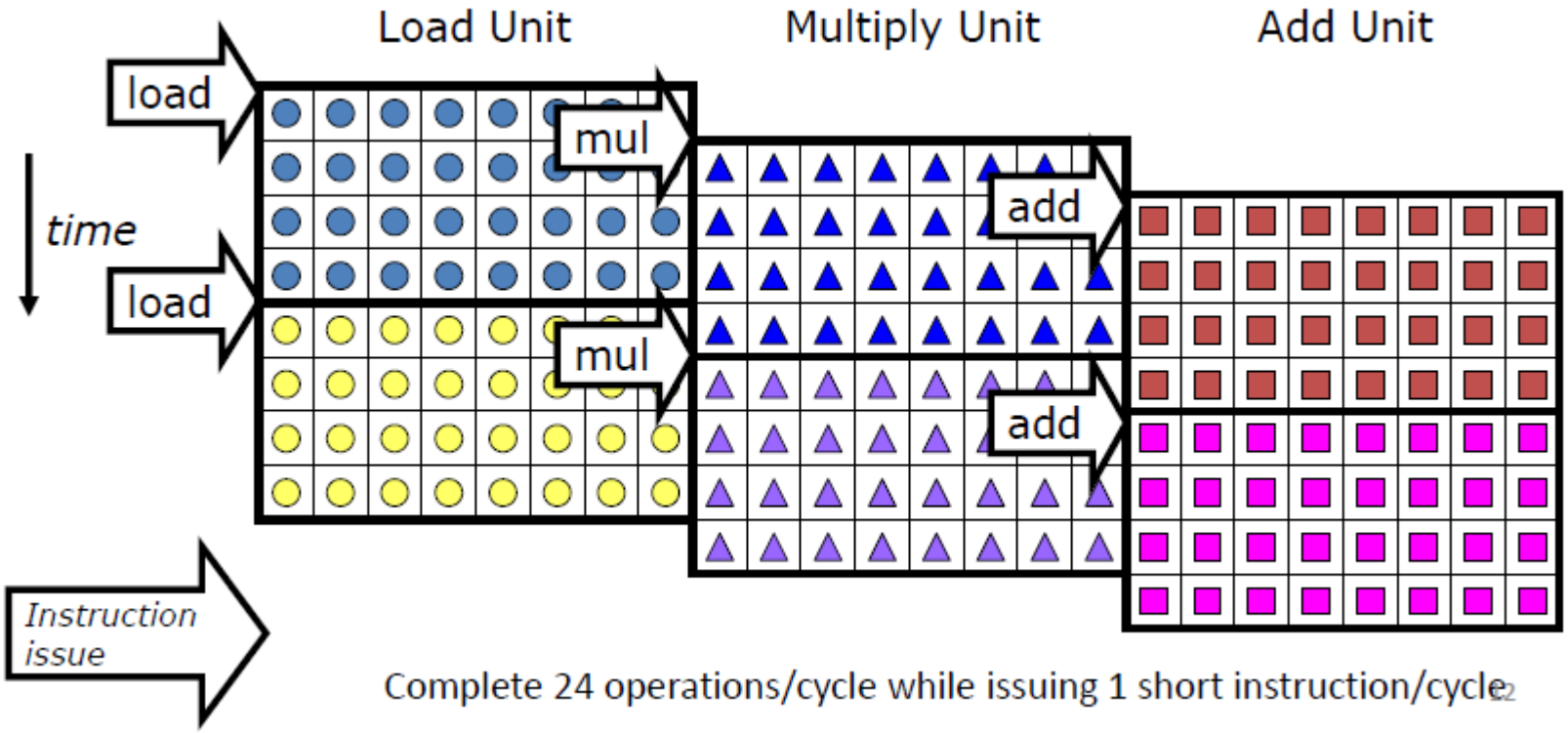
Paralelismo de Instrução Vetorial

- Pode sobrepor múltiplas instruções vetoriais
  - Exemplo: máquina tem 32 elementos por registrador de vetor e 8 raias.

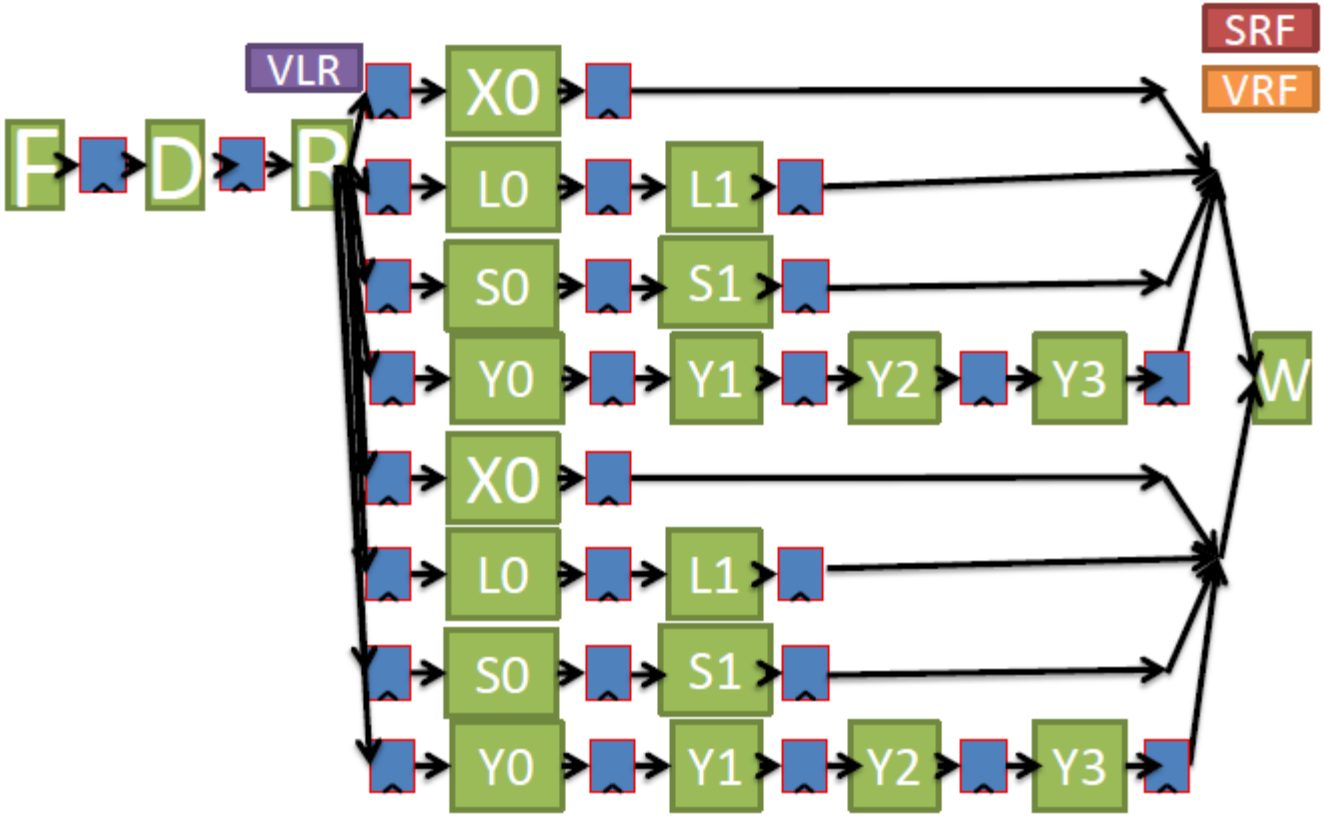


Paralelismo de Instrução Vetorial

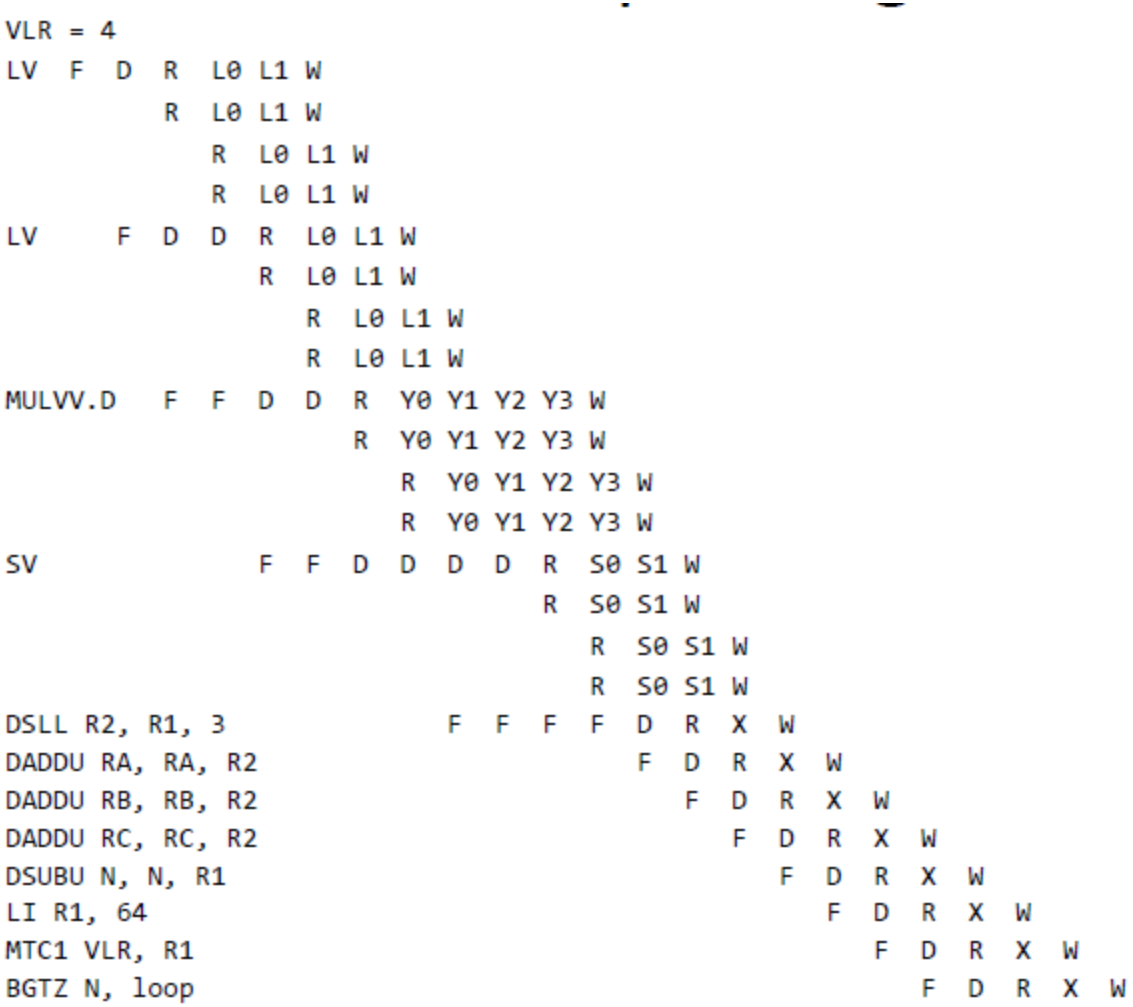
- Pode sobrepor múltiplas instruções vetoriais
  - Exemplo: máquina tem 32 elementos por registrador de vetor e 8 raias.



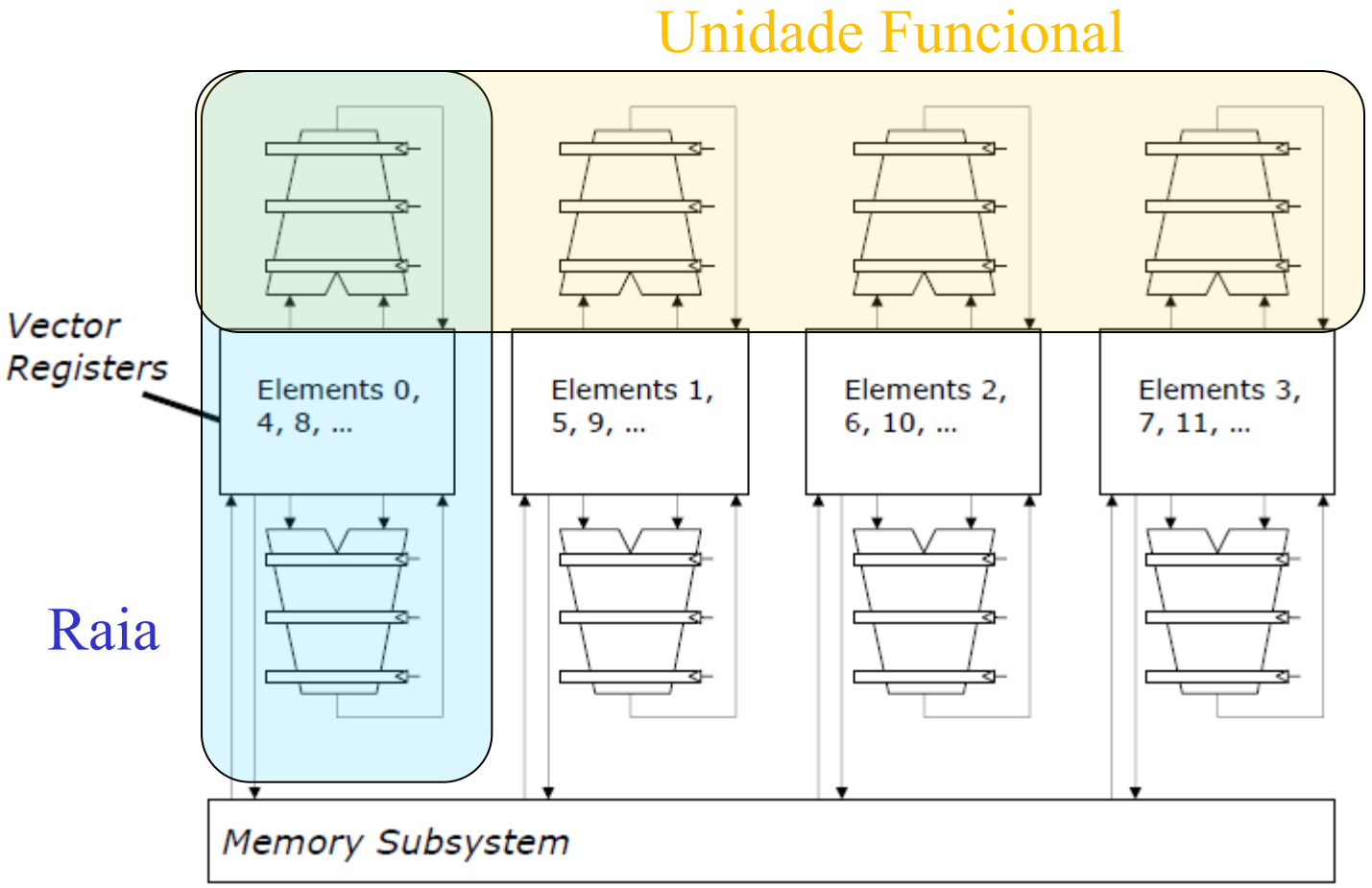
Exemplo de Arquitetura Vetorial – 2 raias



Exemplo de Arquitetura Vetorial – 2 raias



Estrutura da Unidade Vetorial



Estrutura da Unidade Vetorial

- T0 Vector Microprocessor (UCB/ICSI, 1995)

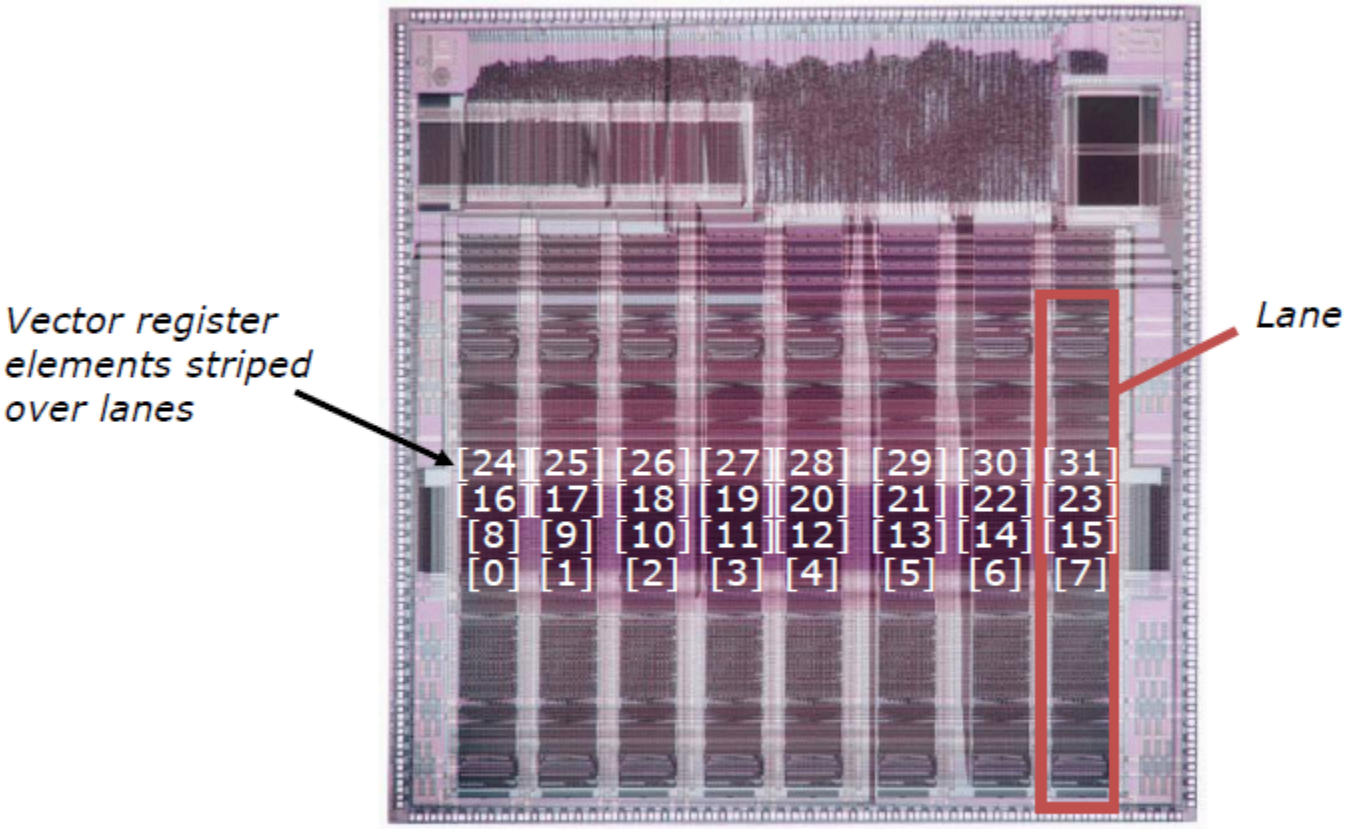


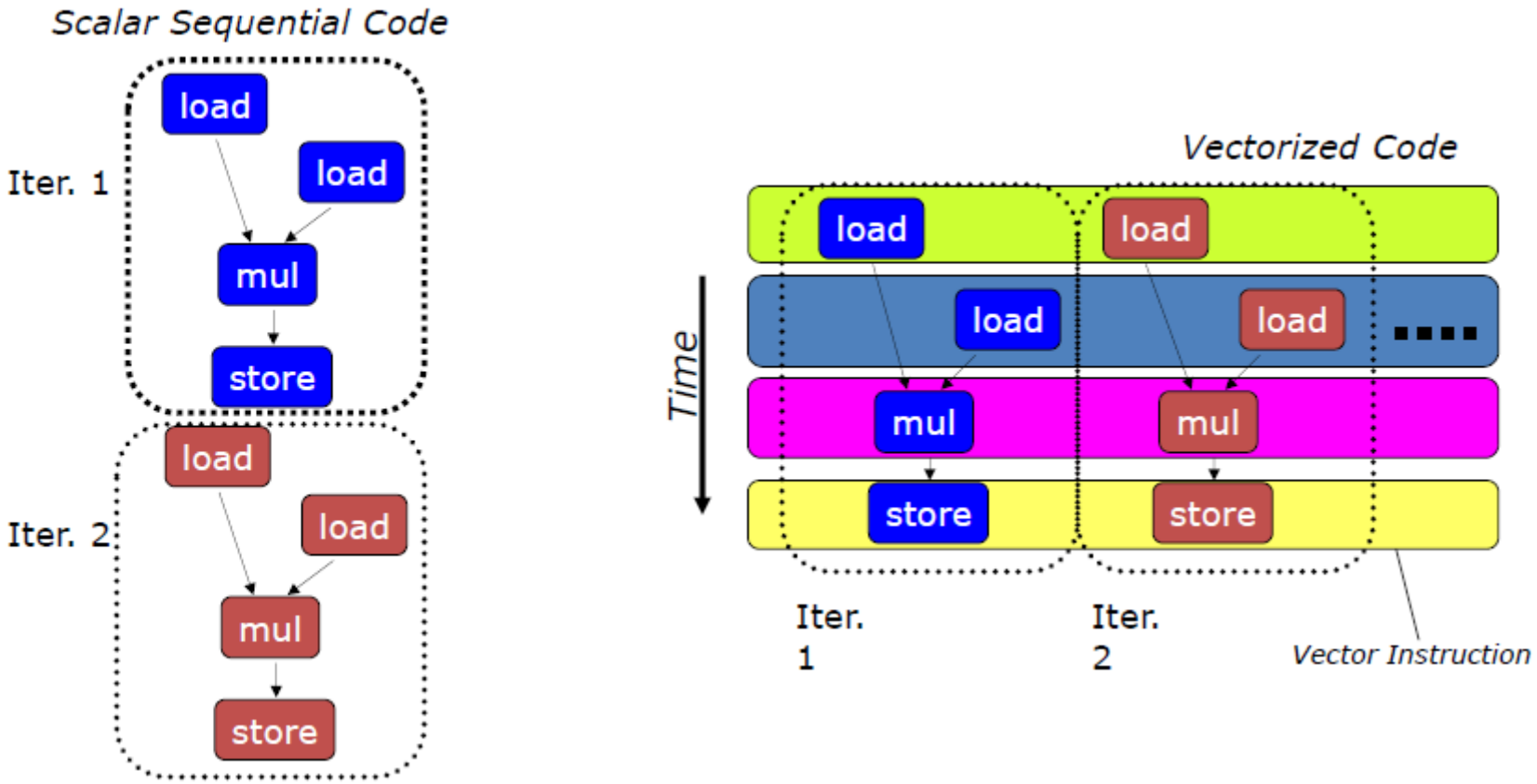
Photo of Berkeley T0, © University of California (Berkeley)
http://www1.icsi.berkeley.edu/Speech/spert/t0die.jpg

Vantagens do ISA vetorial

- **Compacta**
  - Uma instrução curta codifica N operações
- **Expressiva**: diz ao hardware que as N instruções são:
  - Independentes;
  - Usam a mesma unidade funcional;
  - Acessam registradores distintos;
  - Acessam como instruções anteriores;
  - Acessam um bloco contínuo de memória;
  - Acessam memória em um padrão conhecido.
- **Escalável**
  - Pode rodar o mesmo código em máquinas mais paralelas (raias).

Vetorização automática de códigos

```
for (i=0; i < N; i++)
    C[i] = A[i] * B[i];
```



Vetorização – execução condicional

- **Problema**: vetorizar loops com condições

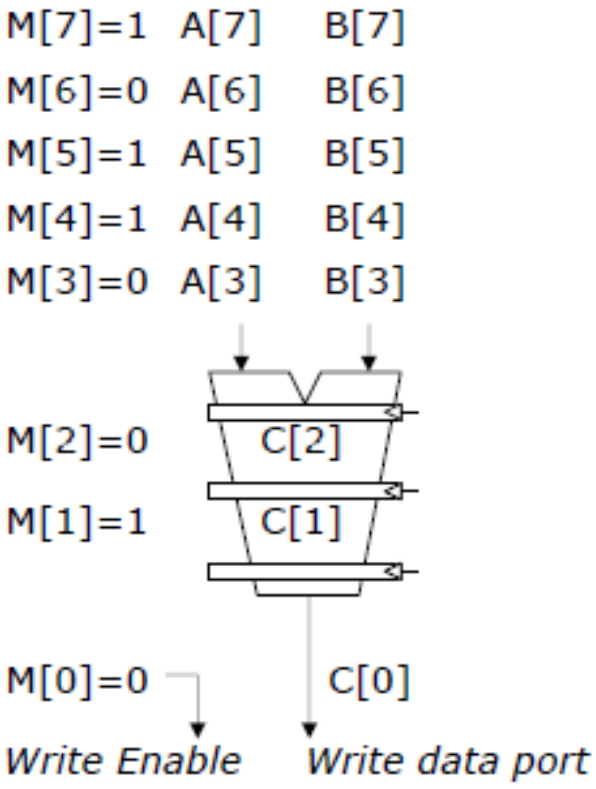
```
for (i=0; i<N; i++)
    if (A[i]>0) then
        A[i] = B[i];
```

- **Solução**: adicionar um registrador de vetores de máscaras
  - Instrução se torna NOP se máscara for falsa

```
CVM             # Turn on all elements
LV VA, RA       # Load entire A vector
SGTVS.D VA, F0  # Set bits in mask register where A>0
LV VA, RB       # Load B vector into A under mask
SV VA, RA       # Store A back to memory under mask
```

Vetorização – execução condicional

- **Solução**: executa todas as instruções. Não escreve se máscara não permitir.



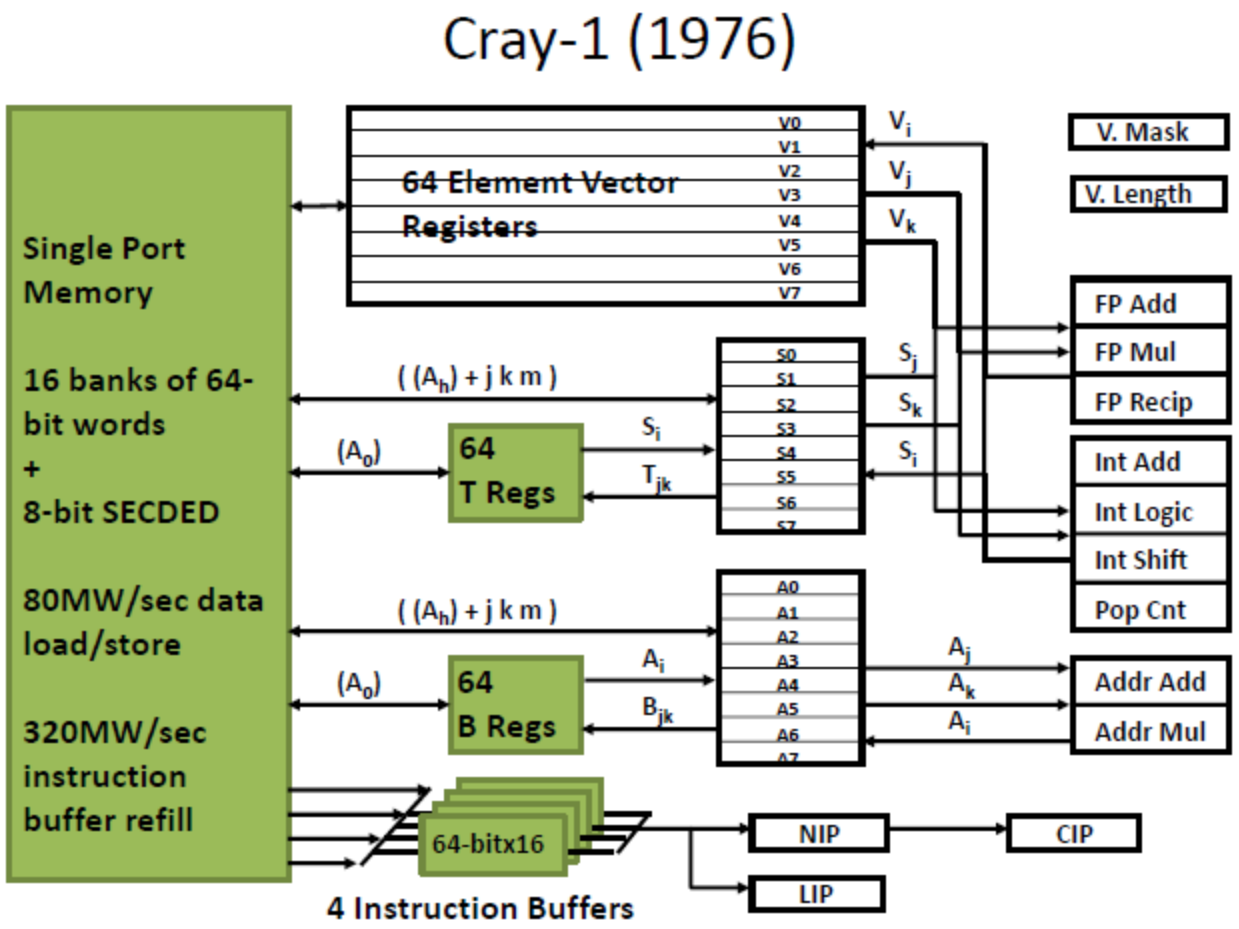
Supercomputador Vetorial

Cray-1 , 1976

- Unidade Escalar
  - Arquitetura load/store
- Extensão Vetorial
  - Vector Register;
  - Vector Instruction;
- Implementação
  - Controle *hardwired*.
  - Unidade funcionais altamente pipelined;
  - Sistema de memória intercalada;
  - Sem cache;
  - Sem memória virtual.

Cray 1 at The Deutsches Museum
Image Credit: Clemens Pfeiffer
http://en.wikipedia.org/wiki/File:Cray-1-deutsches-museum.jpg

Supercomputador Vetorial



memory bank cycle 50 ns *processor cycle* 12.5 ns (80MHz)

Extensões SIMD

- Aplicações de Media **operam em tipos de dados menores** que as palavras nativas.

- Limitações **comparadas às instruções vetoriais**:
  - Número de operandos de dados fixo no opcode
  - Não há modo de endereçamento sofisticado;
  - Sem registrador de máscara.

Implementações SIMD

- Implementações:
  - Intel MMX (1996)
    - Oito 8-bit ops ou quatro 16-bit ops
  - Streaming SIMD Extensions (SSE) (1999)
    - Oito16-bit ops
    - Quatro 32-bit ops or dois 64-bit ops
  - Advanced Vector Extensions (AVX) (2010)
    - Quatro 64-bit ops
  - Operandos devem ser posições consecutivas e alinhadas de memória

Exemplo de código SIMD

Example DXPY:

```
        L.D     F0,a        ;load scalar a
        MV      F1, F0      ;copy a into F1 for SIMD MUL
        MO      F2, F0      ;copy a into F2 for SIMD MUL
        MOV     F3, F0      ;copy a into F3 for SIMD MUL
        DADDIU  R4,Rx,#512  ;last address to load
Loop:   L.4D    F4,0[Rx]    ;load X[i], X[i+1], X[i+2], X[i+3]
        MUL.4D  F4,F4,F0    ;a×X[i],a×X[i+1],a×X[i+2],a×X[i+3]
        L.4D    F8,0[Ry]    ;load Y[i], Y[i+1], Y[i+2], Y[i+3]
        ADD.4D  F8,F8,F4    ;a×X[i]+Y[i], ..., a×X[i+3]+Y[i+3]
        S.4D    0[Ry],F8    ;store into Y[i], Y[i+1], Y[i+2], Y[i+3]
        DADDIU  Rx,Rx,#32   ;increment index to X
        DADDIU  Ry,Ry,#32   ;increment index to Y
        DSUBU   R20,R4,Rx   ;compute bound
        BNEZ    R20,Loop    ;check if done
```

Roofline Performance Model

- Intensidade Aritmética:
  - Operações de pontos flutuante por byte lido.