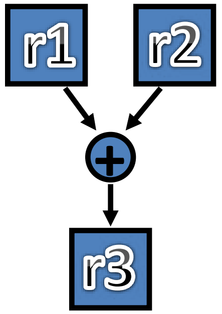
Vector Processing

Alternative Model:Vector Processing

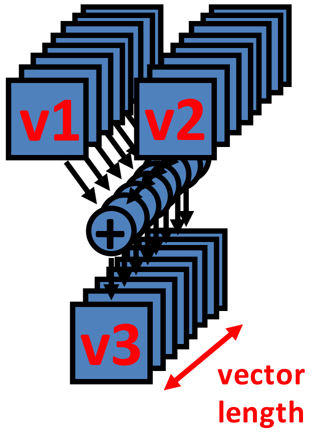
- Vector processors have high-level operations that work on linear arrays of numbers: "vectors"

SCALAR
(1 operation)



```
add r3, r1, r2
```

VECTOR
(N operations)



```
add.vv v3, v1, v2
```

3.4.1 Characteristics of Vector Processing

A vector operand contains an ordered set of n elements, where n is called the *length* of the vector. Each element in a vector is a scalar quantity, which may be a floating-point number, an integer, a logical value, or a character (byte). Vector instructions can be classified into four primitive types:

$$
\begin{aligned}
f_1 &: V \rightarrow V \\
f_2 &: V \rightarrow S \\
f_3 &: V \times V \rightarrow V \\
f_4 &: V \times S \rightarrow V
\end{aligned}
\tag{3.20}
$$

where V and S denote a vector operand and a scalar operand, respectively. The mappings f_1 and f_2 are unary operations and f_3 and f_4 are binary operations. As

Vector Instruction

- Given V- a Vector(operand)
  - An ordered set of n elements(n : the *length* of vector)
    - Elements are scalar : floating point number, integer, logical value, character(byte)
- Given S- a scalar (operand)

- Unary Operation- f1 and f2:
  - f1: $V \longrightarrow V$ example

f_1 VSQR Vector square root : $B(I) \leftarrow \sqrt{A(I)}$

Vector Instruction contd….

- Unary Operation-f2(Vector Reduction Instruction):
  - f2: $V \longrightarrow S$ example:

f_2	VSUM	Vector summation:	$S = \sum_{I=1}^{N} A(I)$

- $V_i \times V_j \longrightarrow S$ (dot product)
- Binary Operation- f3 and f4:
  - f3: $V \times V \longrightarrow V$ example:

f_3	VADD	Vector add:	$C(I) = A(I) + B(I)$

  - f4: $V \times S \longrightarrow V$ example:

f_4	SADD	Vector-scalar add:	$B(I) = S + A(I)$

Table 3.5 Some representative vector instructions

Type	Mnemonic	Description ($I = 1$ through N)	
f_1	VSQR	Vector square root:	$B(I) \leftarrow \sqrt{A(I)}$
	VSIN	Vector sine:	$B(I) \leftarrow \sin(A(I))$
	VCOM	Vector complement:	$A(I) \leftarrow \overline{A(I)}$
f_2	VSUM	Vector summation:	$S = \sum_{I=1}^{N} A(I)$
	VMAX	Vector maximum:	$S = \max_{I=1,N} A(I)$
f_3	VADD	Vector add:	$C(I) = A(I) + B(I)$
	VMPY	Vector multiply:	$C(I) = A(I) * B(I)$
	VAND	Vector and:	$C(I) = A(I)$ and $B(I)$
	VLAR	Vector larger:	$C(I) = \max(A(I), B(I))$
	VTGE	Vector test $>$:	$C(I) = 0$ if $A(I) < B(I)$
			$C(I) = 1$ if $A(I) > B(I)$
f_4	SADD	Vector-scalar add:	$B(I) = S + A(I)$
	SDIV	Vector-scalar divide:	$B(I) = A(I)/S$

Vector Instruction contd..

- Vector – Memory Instruction
  - f5 : $M \longrightarrow V$ #Vector Load
  - f6 : $V \longrightarrow M$ #Vector Store
  - f7: $M \longrightarrow V_1 \times V_0$ # Vector Gather
  - f6: $V_1 \times V_0 \longrightarrow M$ # Vector Scatter

  - Ref Chapter 8 (Hwang) : page 403

Applications

- Vector processing techniques operate in video-game console hardware and in graphics accelerators, weather related data,
- Multimedia Processing (compress., graphics, audio synth, image proc.)
  - Standard benchmark kernels (Matrix Multiply, FFT, Convolution, Sort)
  - Lossy Compression (JPEG, MPEG video and audio) Lossless
  - Compression (Zero removal, RLE, Differencing, LZW)
- Cryptography (RSA, DES/IDEA, SHA/MD5)
- Speech and handwriting recognition
- Operating systems/Networking (memcpy, memset, parity, checksum)
- Databases (hash/join, data mining, image/video serving)

Vector Processors Contd….

Definitions

- Vector
  - A set of scalar items all of same type
- Stored in memory, vector elements are ordered to have addressing increment between successive elements , that is called stride
- Hardware resources
  - Vector registers
  - Register counters and scalar registers
  - Functional Pipelines
    - (Vector processing occurs when arithmetic and logical operations are applied to vectors)

Vector Processing

- Advantage
  - Faster , more efficient than scalar processing
  - Reduces software overhead in maintenance of looping control
  - Reduces memory access conflicts
  - Matches with pipelining and segmentation concepts to generate one result per clock cycle continuously.

Vector Processing

- Disdvantage
  - Increased hardware cost
  - Compiler(Vectorizer) capable of vectorization(conversion of scalar code to vector code) has higher cost

Vector Operand

- Entries in matrix may be store in row major or column major way in the memory
  - Each row, column, diagonal may be used as a vector
- (Note:
- Row elements are stored in contiguous locations with a unit stride
- Column elements must be stored with stride n (n is matrix order)
- Diagonal of matrix separated by a stride of n+1

Pipelined Vector Processing Methods

- Vector processing involves processing large arrays of data
  - Example:

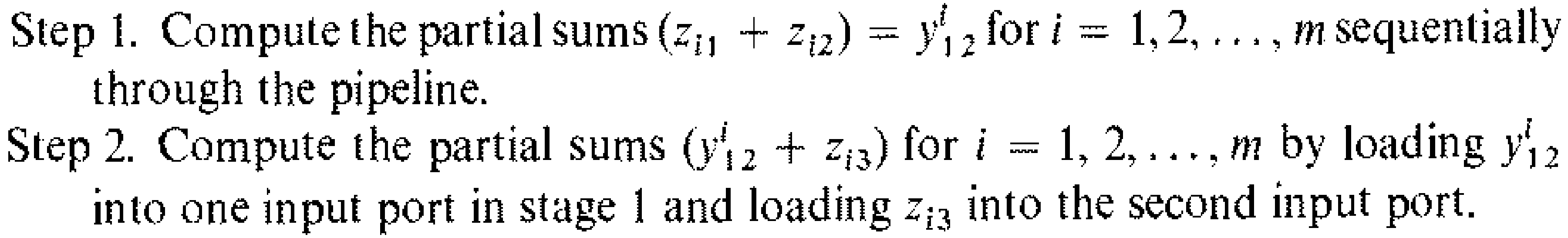
$$\begin{aligned} y_1 &= z_{11} + z_{12} + \cdots + z_{1n} \\ y_2 &= z_{21} + z_{22} + \cdots + z_{2n} \\ &\cdots\cdots\cdots\cdots\cdots\cdots \\ y_m &= z_{m1} + z_{m2} + \cdots + z_{mn} \end{aligned}$$

- 3 Types of Vector Processing Methods
  - 1) Horizontal Vector Processing
  - Vector y calculated in sequential order $y_{i,\ i=1,\ldots,m}$
  - Not useful for vector processing

$y_i = \sum_{j=1}^{n} z_{ij}$ involving $(n-1)$ additions must be completed before switching to the evaluation of the next summation $y_{i+1} = \sum_{j=1}^{n} z_{i+1,j}$. To evaluate each y_i

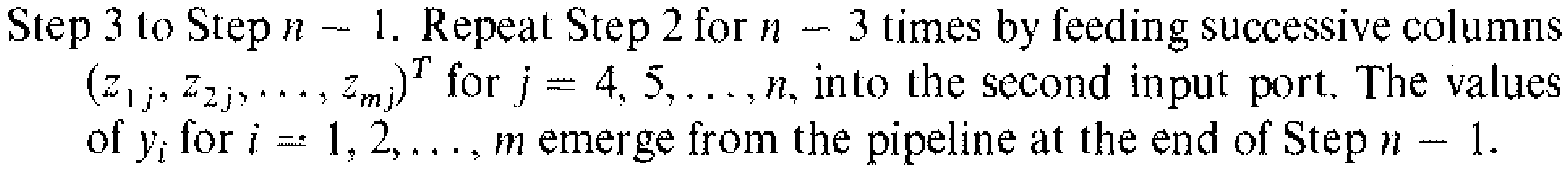
Pipelined Vector Processing Methods contd….

- 2) Vertical Vector Processing
- Number of vector component m is unrestricted, many intermediate results have to be stored
- useful for vector processing, but vectors are very large in size



Step 1. Compute the partial sums $(z_{i1} + z_{i2}) = y^i_{12}$ for $i = 1, 2, \ldots, m$ sequentially through the pipeline.

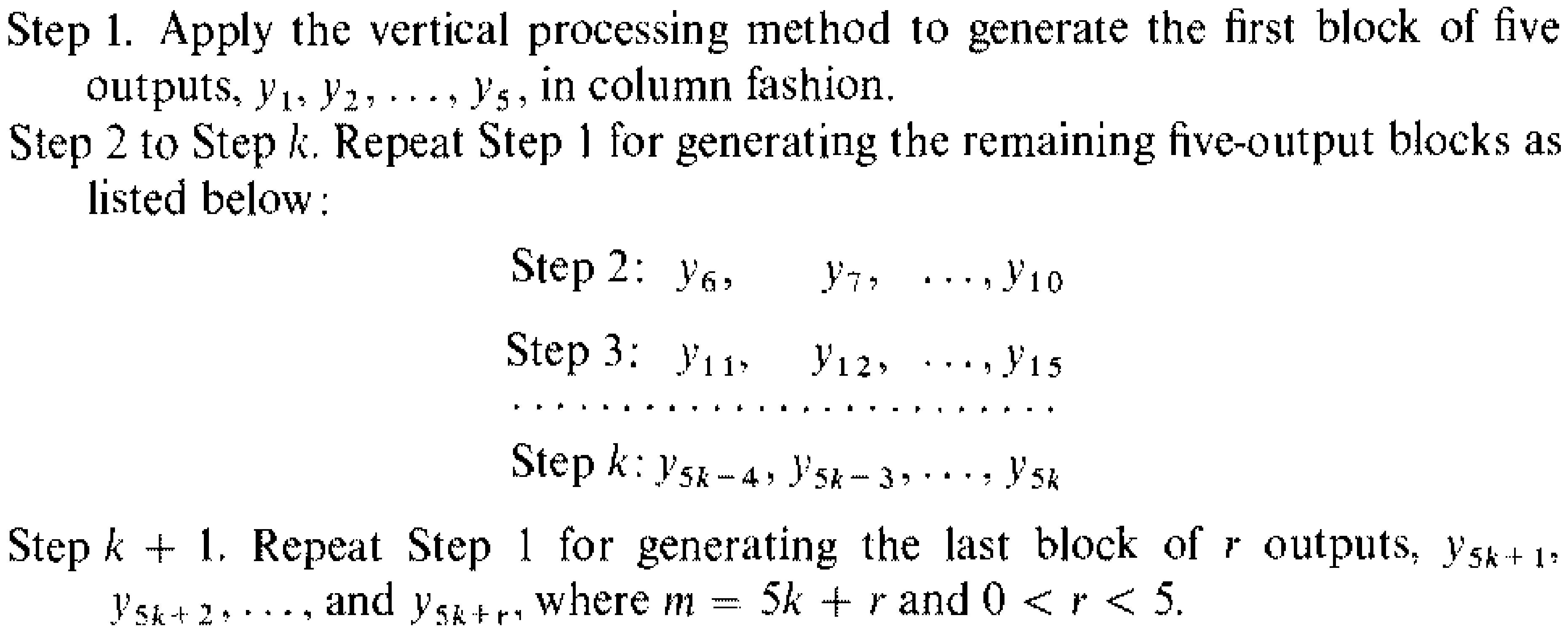
Step 2. Compute the partial sums $(y^i_{12} + z_{i3})$ for $i = 1, 2, \ldots, m$ by loading y^i_{12} into one input port in stage 1 and loading z_{i3} into the second input port.



Step 3 to Step $n - 1$. Repeat Step 2 for $n - 3$ times by feeding successive columns $(z_{1j}, z_{2j}, \ldots, z_{mj})^T$ for $j = 4, 5, \ldots, n$, into the second input port. The values of y_i for $i = 1, 2, \ldots, m$ emerge from the pipeline at the end of Step $n - 1$.

Pipelined Vector Processing Methods contd….

- 3) Vector Looping Method
- Not restricted by vector length m
- Intermediate result appear in small blocks of data, therefore suitable for register-register architecture



Step 1. Apply the vertical processing method to generate the first block of five outputs, $y_1, y_2, \ldots, y_5$, in column fashion.

Step 2 to Step k. Repeat Step 1 for generating the remaining five-output blocks as listed below:

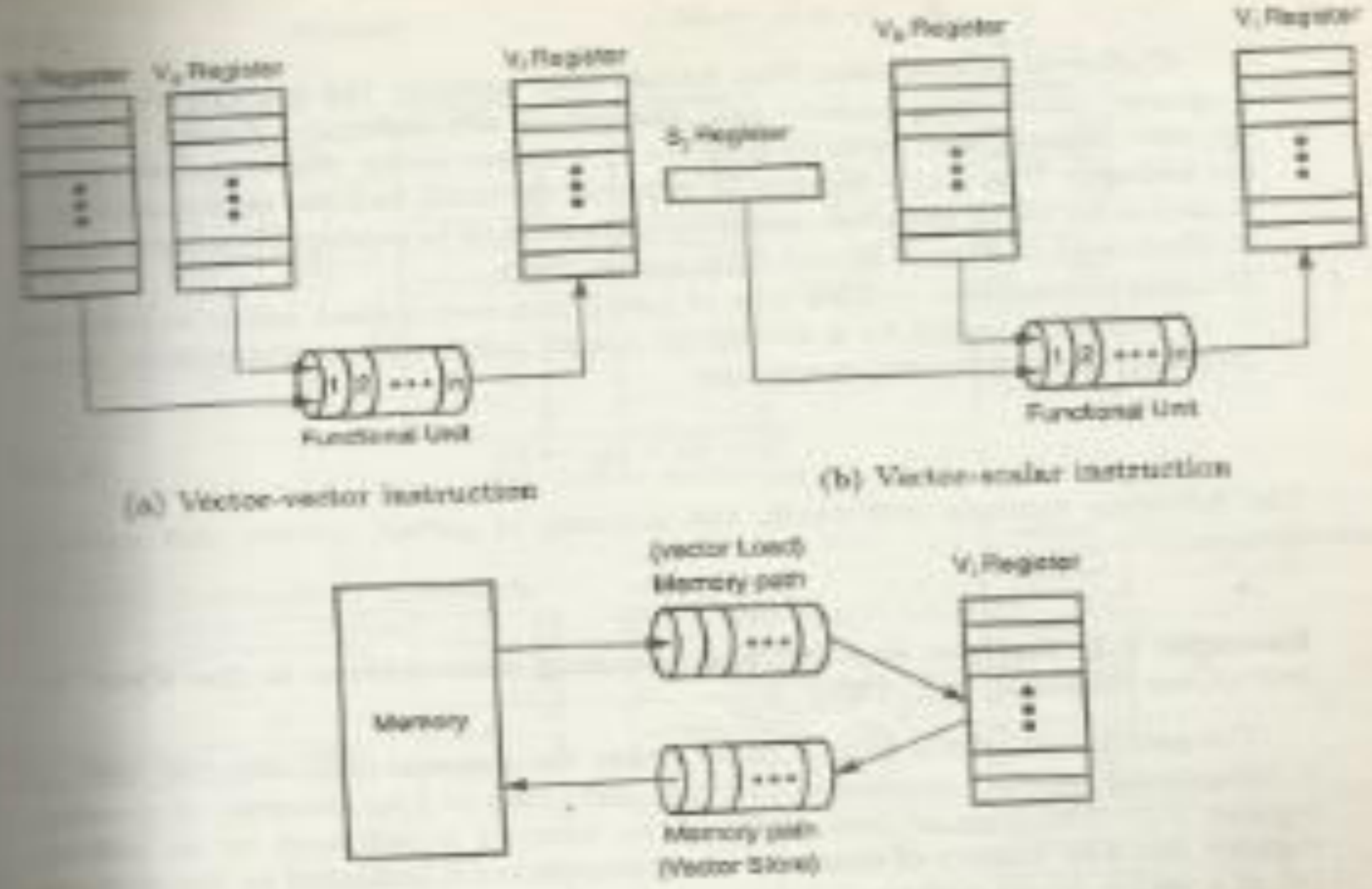
$$\begin{array}{llll}
\text{Step 2:} & y_6, & y_7, & \ldots, y_{10} \\
\text{Step 3:} & y_{11}, & y_{12}, & \ldots, y_{15} \\
\cdots & \cdots & \cdots & \cdots \\
\text{Step } k: & y_{5k-4}, & y_{5k-3}, & \ldots, y_{5k}
\end{array}$$

Step $k+1$. Repeat Step 1 for generating the last block of r outputs, y_{5k+1}, $y_{5k+2}, \ldots$, and y_{5k+r}, where $m = 5k + r$ and $0 < r < 5$.

Vector Operand

- For accessing vector from memory
  - Base address is needed
  - Stride is needed
  - Length of the vector is needed

  (All the above are scalar registers that are accessed for getting vector from memory to vector registers of the processor)

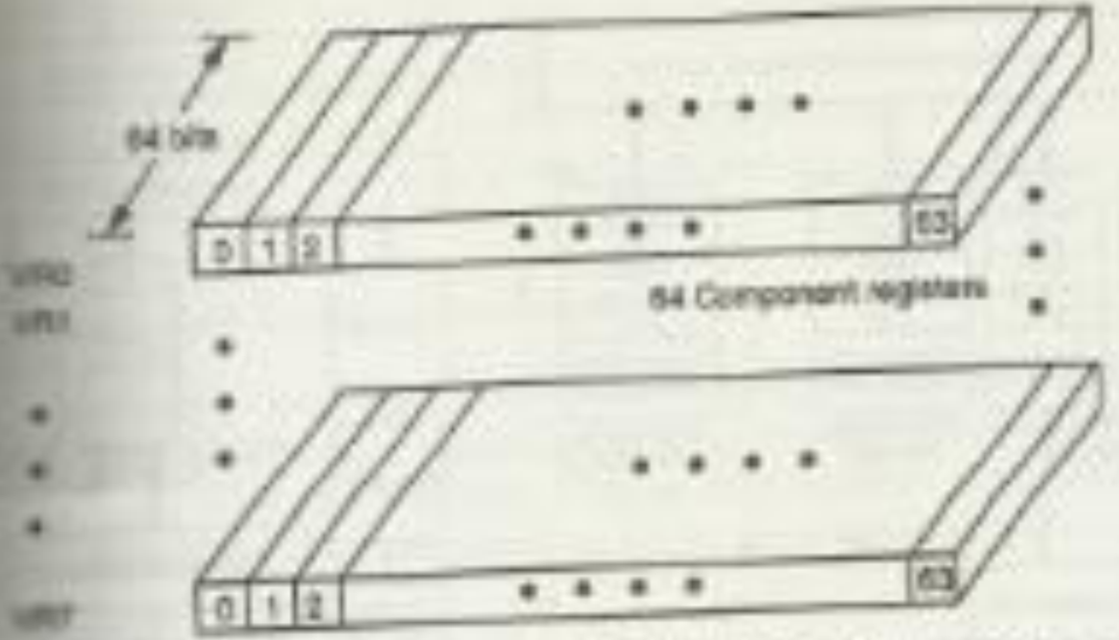


(a) Vector-vector instruction

(b) Vector-scalar instruction

(c) Vector-memory instructions

Figure 8.1 Vector instruction types in Cray-like computers.



(a) Eight vector registers (8 × 64 × 64 bits) on Cray machines

Figure 8.14 Vector register file in Cray and Fujitsu supercomputers.

Styles of Vector Architectures

- *memory-memory vector processors*: all vector operations are memory to memory
- *vector-register processors*: all vector operations between vector registers (except load and store)
  - Vector equivalent of load-store architectures
  - Includes all vector machines since late 1980s: Cray, Convex, Fujitsu, Hitachi, NEC
  - We assume vector-register for rest of lectures

Vector-register processors is same as Register-register processor

Vector Processors Contd….

Vector Operand

- Each vector register has a fixed number of component registers
  - (Very long vector need to be segmented and processed one segment at a time)

VECTOR LENGTH

- vector length of 64(Maximum Vector Length-MVL).
  - 1. In real world applications vector lengths are not exactly 64.
    - adding just first n elements of a vector ,
      - Vector Length register(VLR) used for this
    - VLR controls the length of any vector operation by defining their length.
    - value cannot be greater than the length of the vector registers. (64 in this case)

  - 2. In real world applications, data in vectors in memory can be greater than the MVL of the processor.
    - we use a technique called Strip Mining

STRIP MINING

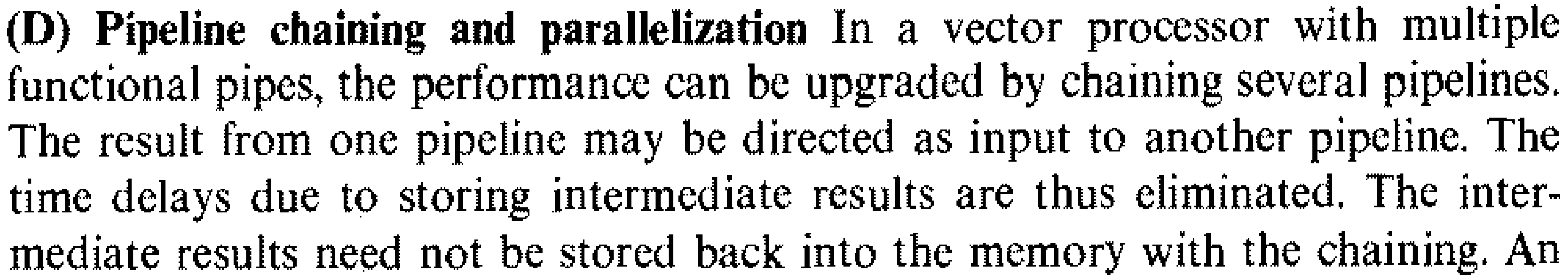
- Splitting data : each vector operation is done for a size less than or equal to MVL(Maximum Vector Length).
  - Done by a simple loop with MOD operator as control point.

  (VL- Vector Length)

```
low = 0;
 VL = (n % MVL); /*find odd-size piece using modulo op % */
 for (j = 0; j <= (n/MVL); j=j+1) {       /*outer loop*/
for (i = low; i < (low+VL); i=i+1)       /*runs for length VL*/
Y[i] = a * X[i] + Y[i] ;                        /*main operation*/
 low = low + VL;                               /*start of next vector*/
VL = MVL;                                    /*reset the length to MVL*/
}
```

Vector Pipeline Chaining

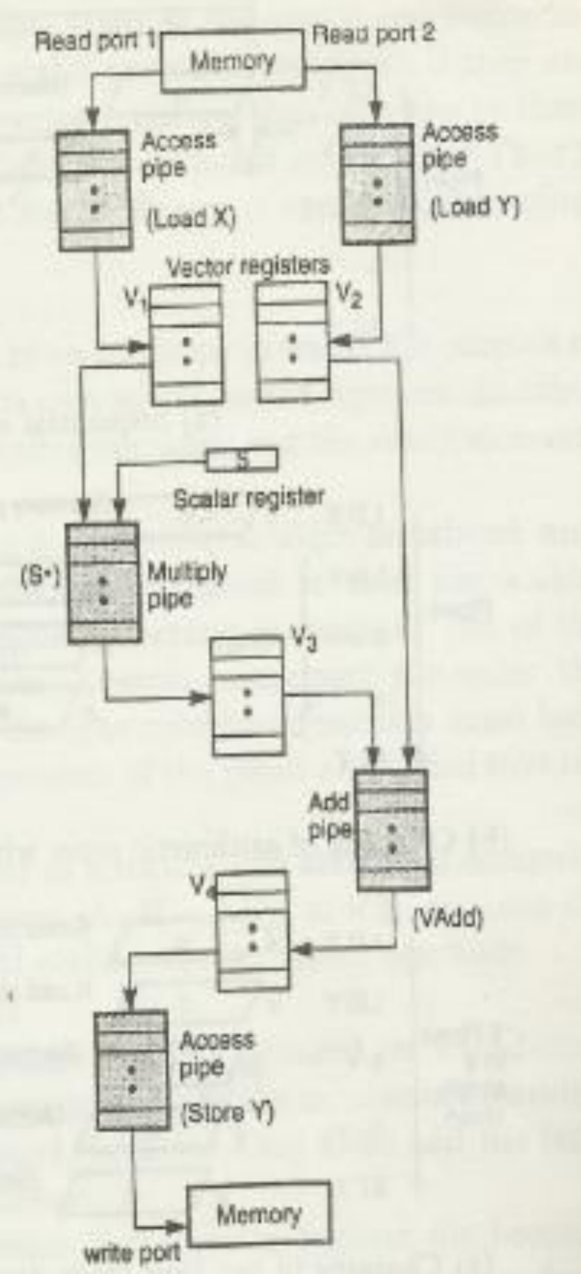
Hwang and Briggs Page 310



(D) Pipeline chaining and parallelization In a vector processor with multiple functional pipes, the performance can be upgraded by chaining several pipelines. The result from one pipeline may be directed as input to another pipeline. The time delays due to storing intermediate results are thus eliminated. The intermediate results need not be stored back into the memory with the chaining. An

- (like data forwarding in scalar pipelinging)
- Same functional unit cannot be assigned to execute more than one instruction in the same chain
  - Functional Units NEED to be Independent

Hwang Page 439 Figure



(b) Complete chaining using three memory-access pipes in the Cray X-MP

The result vector of one functional pipeline is made the operand vector of the next functional pipeline, instead of wasting time in copying result to operand vector intelligent compiler should have the capability of detecting sequences of operation that can be chained together. We have seen some chaining operations of multipl pipelines in Section 3.4.1. Figure 4.37 shows the parallel use of two load-store pipe which are chained with the *multiply pipe* and then the *add pipe* in the VP-200.

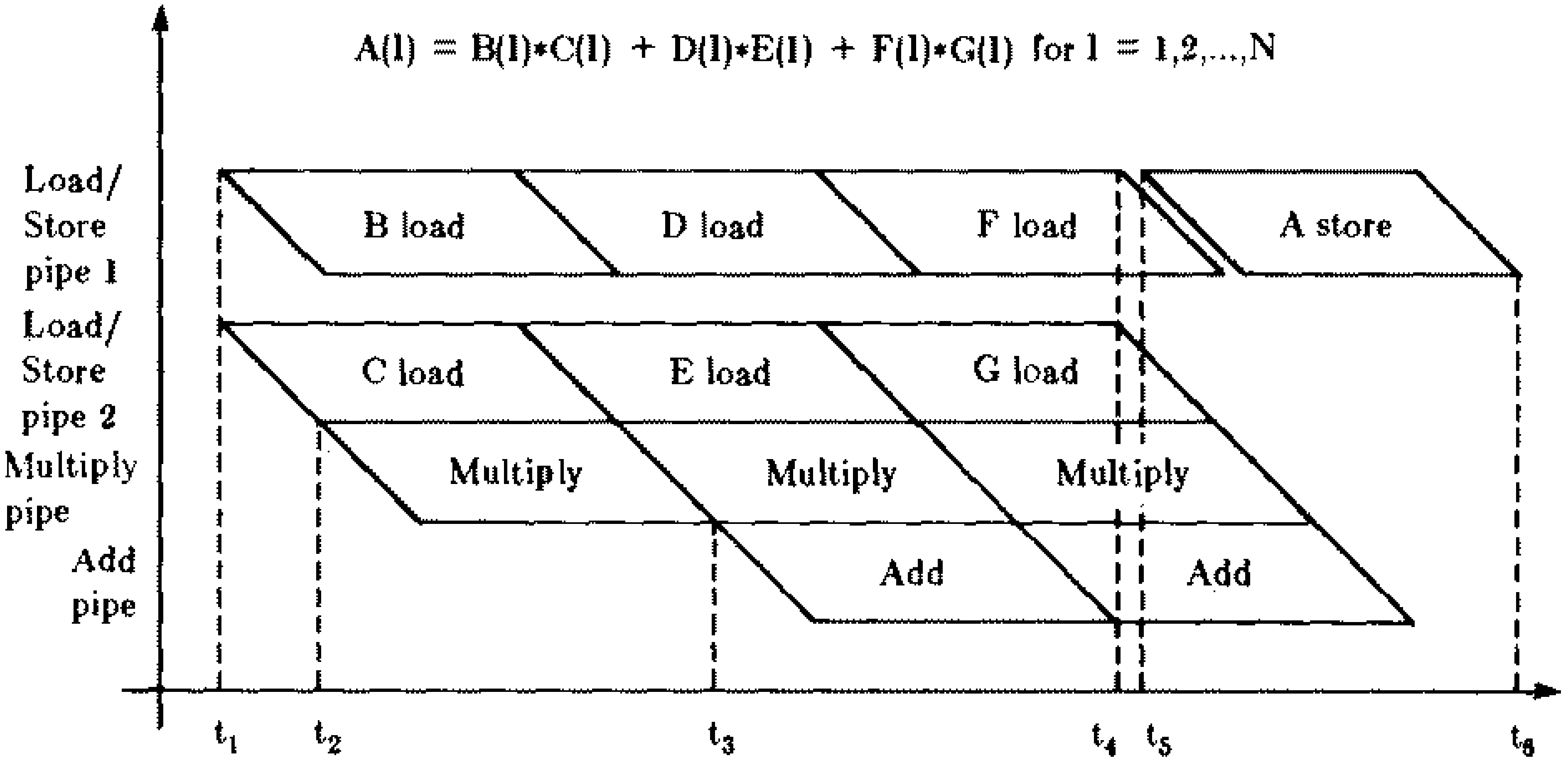


Figure 4.37 Pipeline chaining and parallelization for linked vector operations. (Courtesy of Fujits Limited, Japan.)

Superscalar processor architectures Hwang pages 177-178 , 4.2.1,Chapter 6 pages 310 – 311,

Superscalar Processor

- Scalar Processor:
  - 1 instruction per cycle
  - 1 instruction completion expected from pipeline per cycle
- Superscalar Processor:
  - Multiple instructions issued per cycle(Exploits more instruction level parallelism in user programs)
    - Special compilers used to find instruction level parallelism
  - Independent instructions needed
    - Can be executed without causing a wait state(stall)
    - Average number of independent instructions found in code for parallel operation is 2
      - Therefore instruction issue degree limited to 2 to 5 in superscalar processor

Hwang page 178

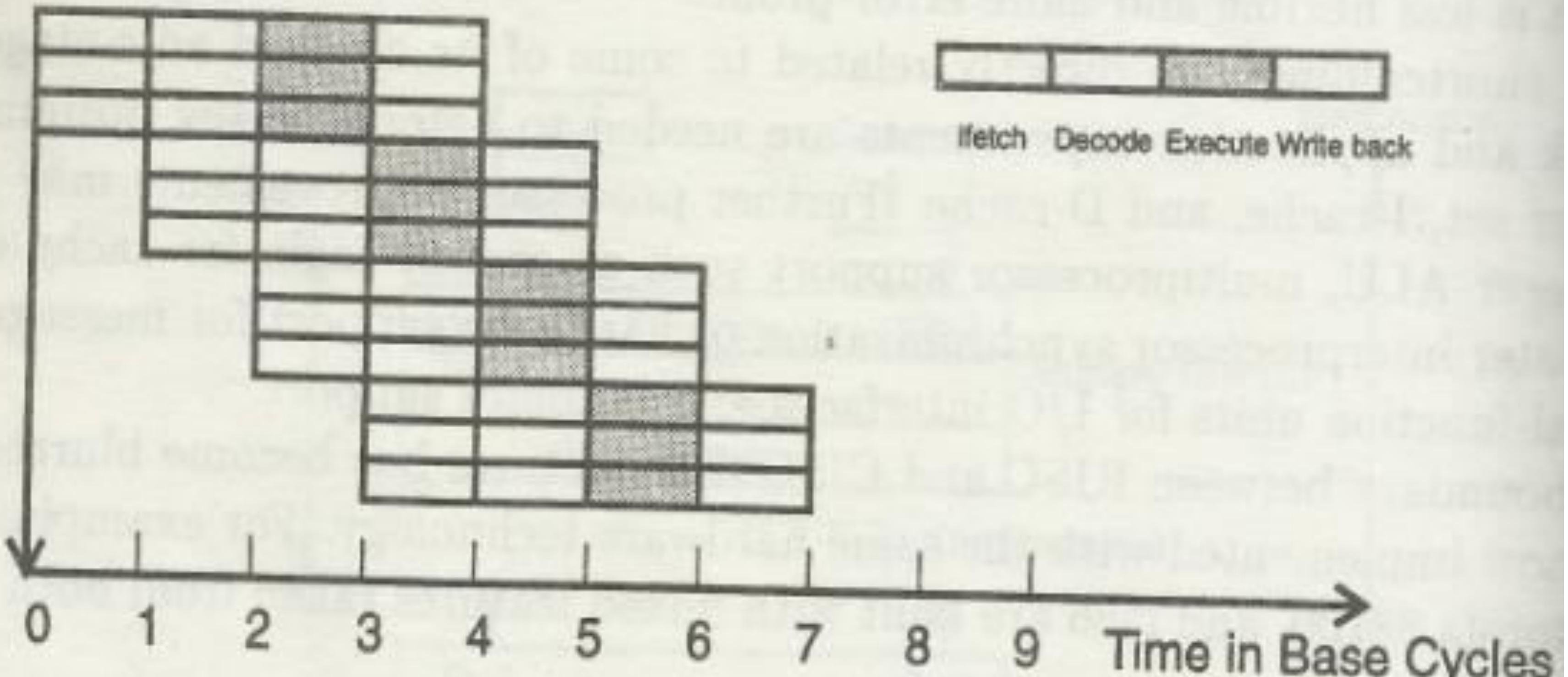


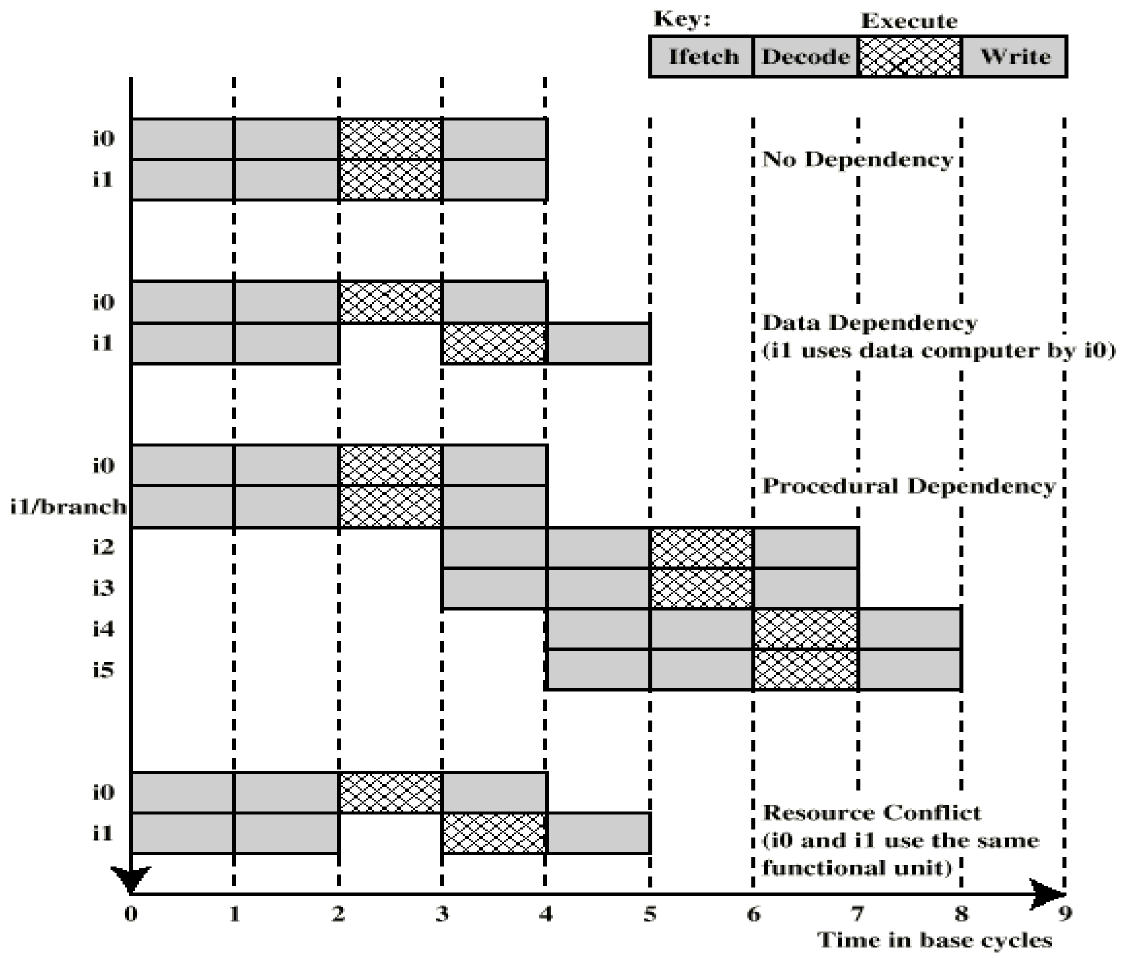
Figure 4.11 A superscalar processor of degree m = 3.

Superscalar Processor

- Multiple instruction pipelines used
- Instruction cache supplies multiple instruction per fetch
- Multiple functional units
  - integer units and floating point units

Note: Actual number of instruction issued to various functional units vary in each cycle constrained by data dependencies and resource conflicts

Effect of Dependencies on Superscalar Operation



Notes:

1) Superscalar operation is double impacted by a stall.

2) CISC machines typically have different length instructions and need to be at least partially decoded before the next can be fetched – not good for superscalar operation

Superscalar Processor

- Superscalar Performance
  - Time required by scalar base machine to execute N independent instruction through pipeline

  T(1,1) = k + N-1(base cycle)

  - Ideal execution time required by m-issue superscalar machine

  T(m,1)= k + (N – m)/m(base cycles)

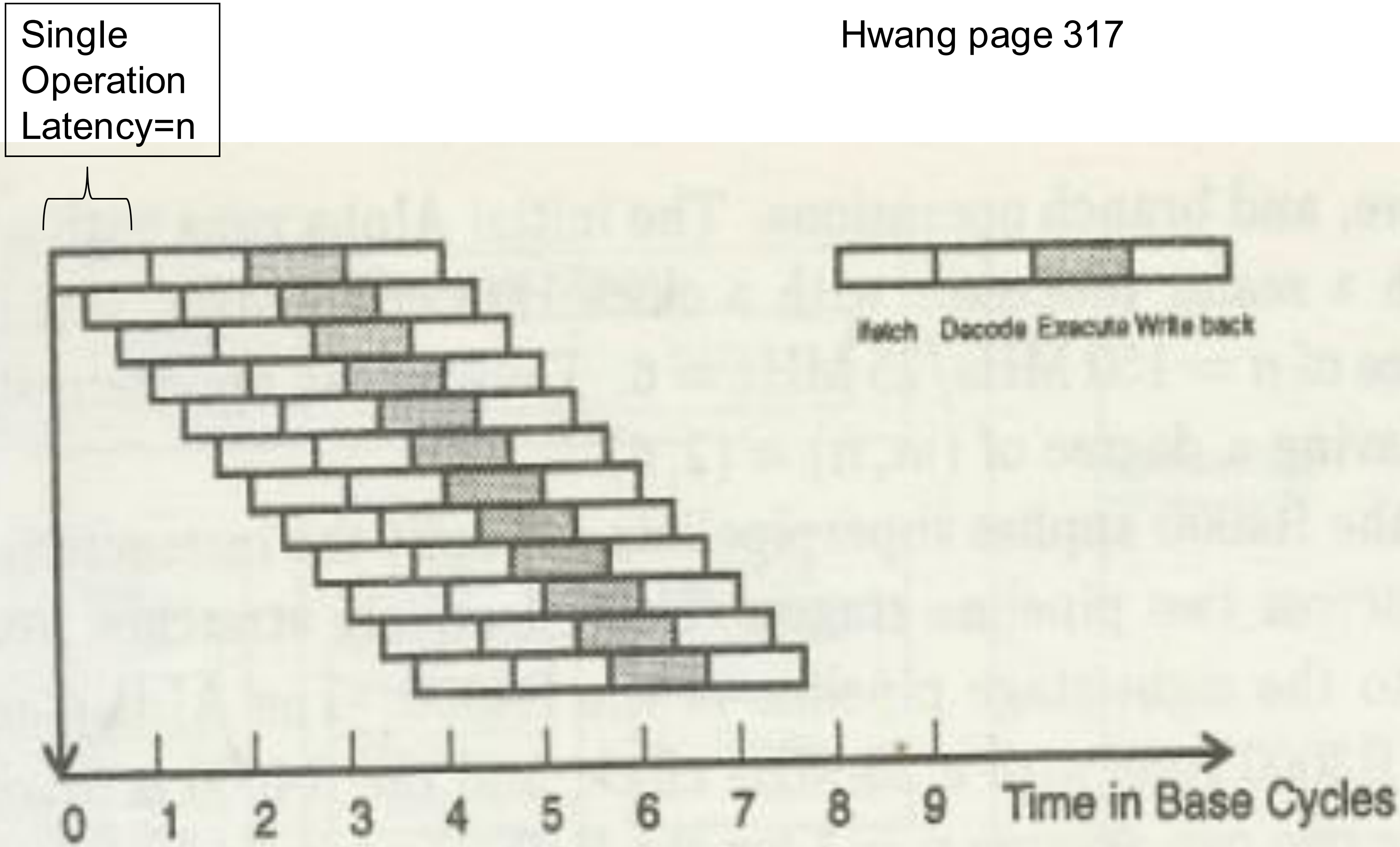
  - k - time required to execute first m instructions through m pipelines simultaneously
- Ideal speedup of superscaler over base machine
  - S(m,1)= T(1,1)/T(m,1) = (N+k-1) /(N/m +k-1)= m(N+k-1)/(N+m(k-1)
- Speedup S(m,1) → m as N→ ∞

Superpipelined processor architectures
Hwang pages 316-318

Superpipelined Processor

- In Superpipelined Processor of degree n
  - Pipeline cycle time is I/n of base cycle
    Superpipelined machine of degree n=3 will issue 1 instruction per cycle – cycle time 1/3 of base cycle time
    - Single operation latency = n pipeline cycles ≈ 1 base cycle
    - ILP required to fully utilize the machine = n instructions
  - High speed clock mechanism required

Hwang page 317



(a) Superpipelined execution with degree $n = 3$

Superpipelined Processor

- Superpipelined Performance
  - Minimum time required to execute N instructions in degree n machine with k stages in pipeline

  T(1,n)= k + 1/n(N-1)

  - Speedup(1,n)= T(1,1)/T(1,n)= (k+ N-1)/(k+(N-1)/n))

  = n(k +N-1)/ (nk + N-1)

  Speedup S(1,n) → n as N→ ∞

Superscalar vs Superpipeline

- Superscalar design
  - Spatial parallelism
    - Multiple operations running concurrently on separate hardware (execution units and register file ports), more transistors required (CMOS technology suitable)
- Superpipelined design
  - Temporal parallelism
    - Overlapping multiple operations on common piece of hardware(more deeply pipelined execution units with faster clock cycles(faster transistors required)

Superpipelined Superscalar Performance

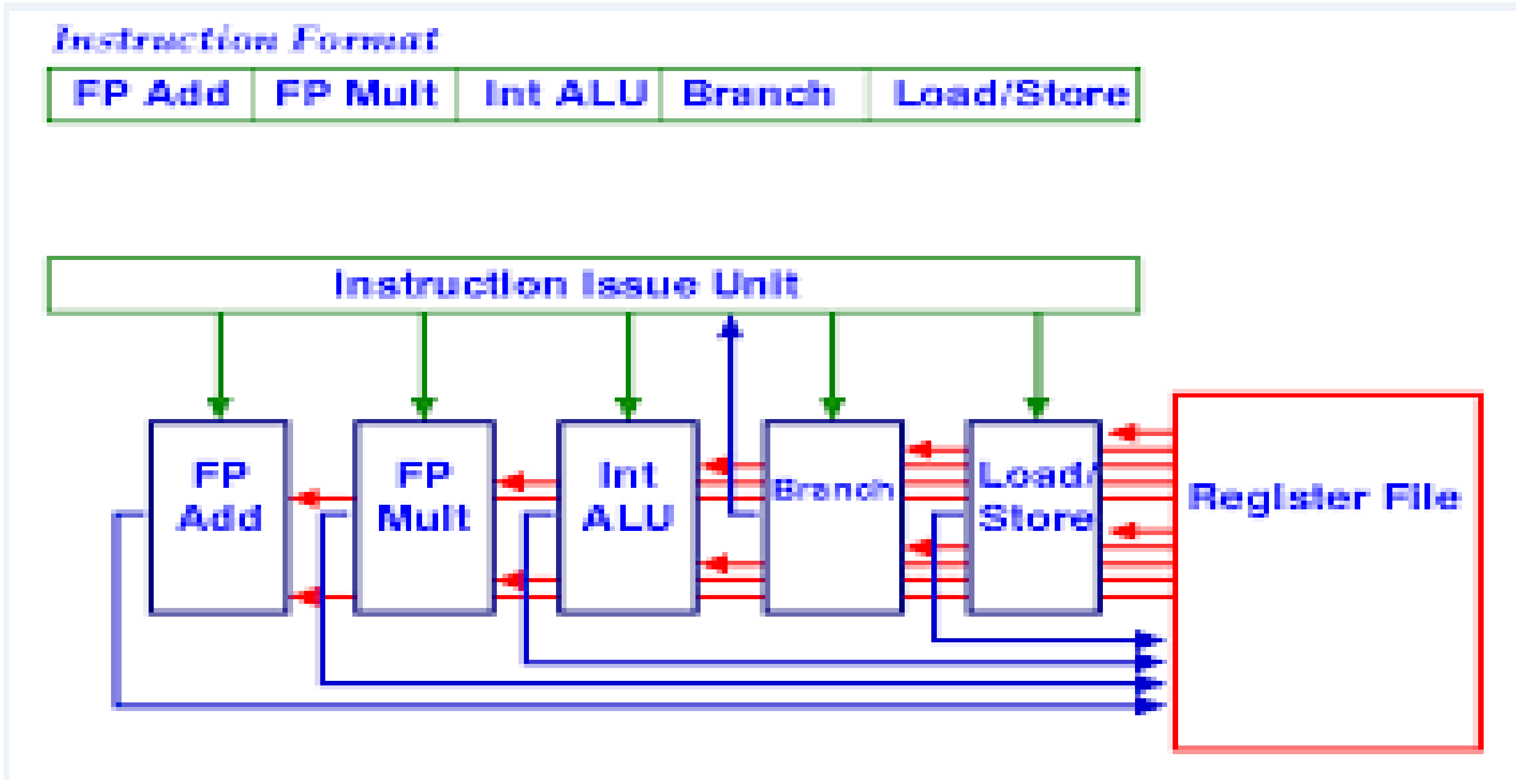
- Minimum time needed to execute N independent instructions on Superpipelined Superscalar machine of degree(m,n)
  - $T(m,n)=k+(N-m)/(mn)$ (base cycles)

- Speedup over base machine
  - $S(m,n)=T(1,1)/T(m,n)= (k+N-1)/ (k+(N-m)/(mn))= mn(k+N-1)/(mnk + N-m)$
- Speedup limit $S(m,n) \rightarrow mn$ as $N \rightarrow \infty$

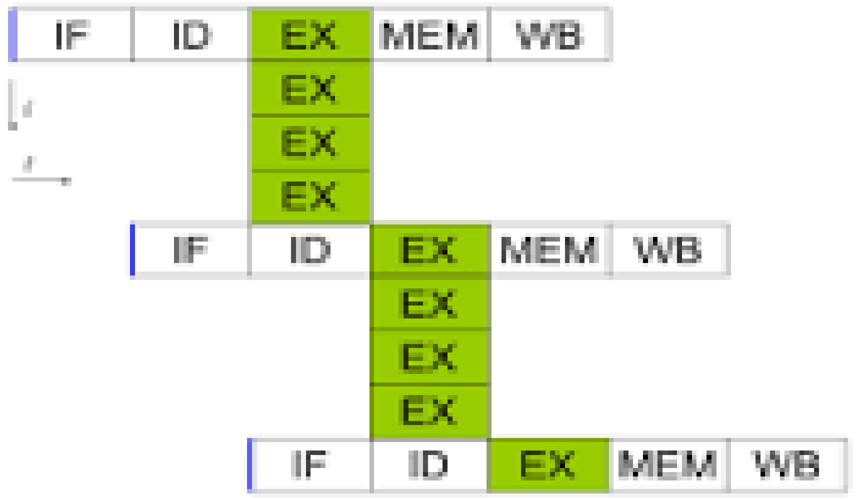
VLIW processor architectures Hwang 4.2.2 page 182 to 184

Very Long Instruction Set Word (100s of bits in length)

- A typical VLIW processor and instruction format, degree=3



Degree=4,The effective clocks/Instruction (CPI)= ¼= . 25



- For VLIW architecture
  - Instruction parallelism and data movement completely specified at compile time
    - ✓ Runtime resource scheduling and synchronization are completely eliminated
    - ✓CPI can be lower than superscalar processor

Superscalar vs VLIW

1) Decoding of VLIW instruction is easier
2) Code density of superscalar machine is better, because fixed VLIW format includes bits for non-executable operations while superscalar issues only executable operations
3) Superscalar machines
   - Object code compatible with non parallel machines

- VLIW m/c exploiting different amounts of parallelism require different instruction set