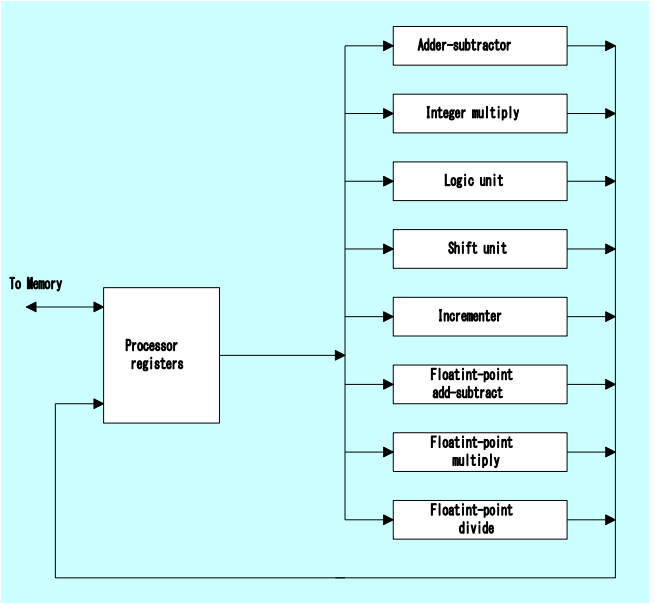
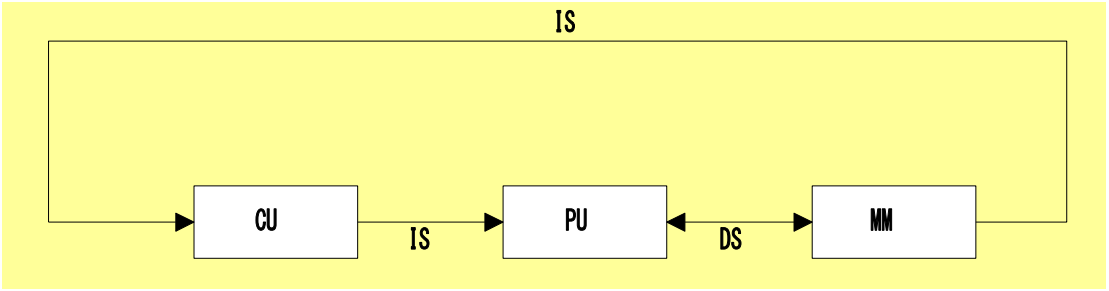
Chap. 9 Pipeline and Vector Processing

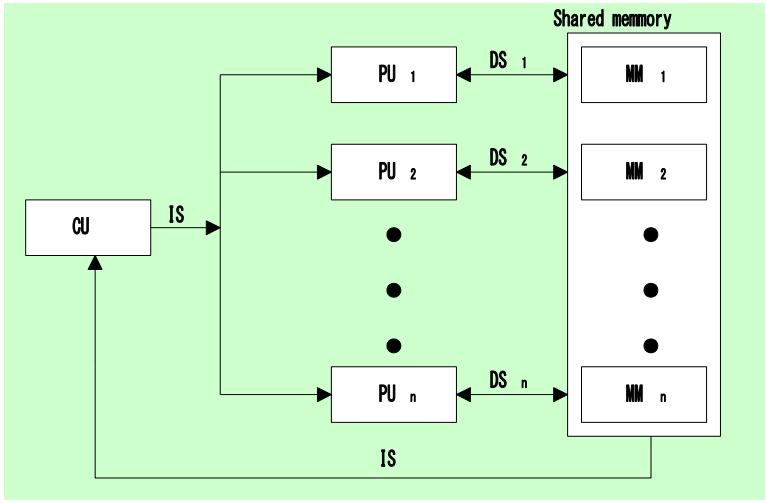
- Parallel Processing
  - = *Simultaneous* data processing tasks for the purpose of increasing the computational speed
  - = Perform *concurrent* data processing to achieve faster execution time
  - Multiple Functional Unit :
    - ***Separate the execution unit into eight functional units operating in parallel***
  - Computer Architectural Classification
    - Data-Instruction Stream : Flynn
    - Serial versus Parallel Processing : Feng
    - Parallelism and Pipelining : Händler
  - Flynn's Classification
    - 1) **SISD** (Single Instruction - Single Data stream)
      - » for practical purpose: only one processor is useful
      - » Example systems : Amdahl 470V/6, IBM 360/91

Parallel Processing Example

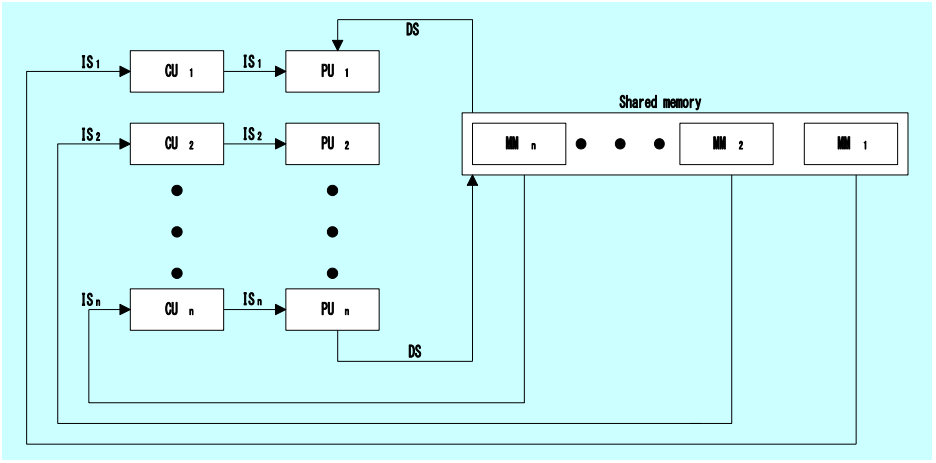




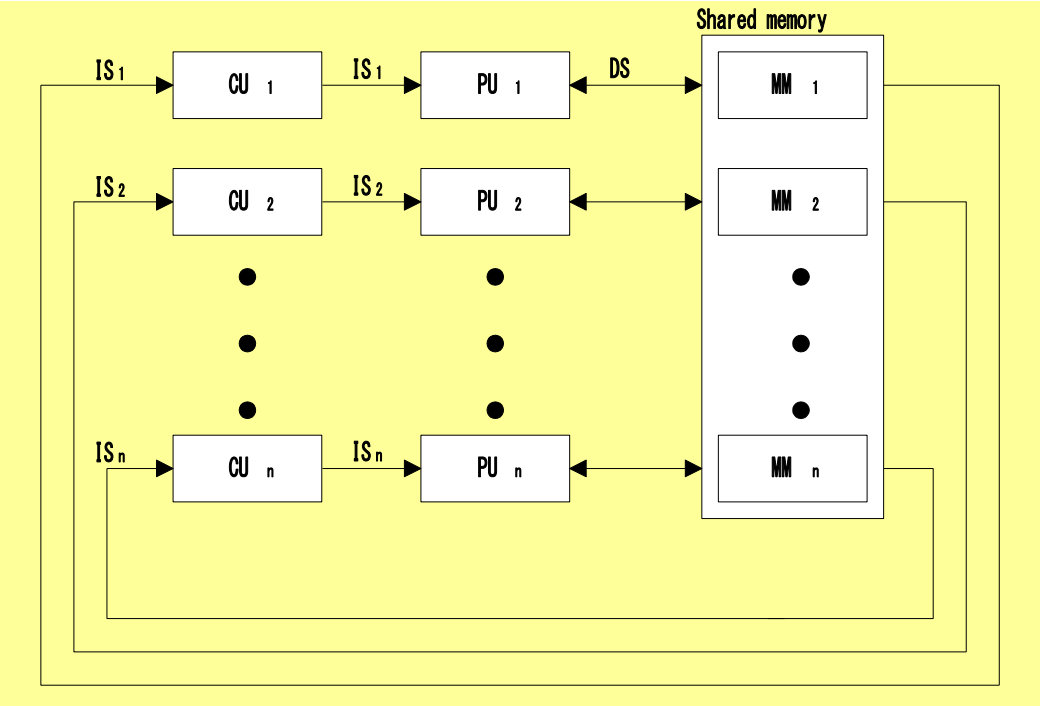
- 2) **SIMD**
  (Single Instruction - Multiple Data stream)
  - » vector or array operations
    - one vector operation includes many operations on a data stream
  - » Example systems : CRAY -1, ILLIAC-IV



- 3) **MISD**
  (Multiple Instruction - Single Data stream)
  - » Data Stream Bottle neck



- 4) **MIMD**
  (Multiple Instruction - Multiple Data stream)
  - » Multiprocessor System



◆ Main topics in this Chapter

- Pipeline processing :
  - » Arithmetic pipeline :
  - » Instruction pipeline :
- Vector processing :adder/multiplier pipeline
- Array processing : array processor
  - » Attached array processor :
  - » SIMD array processor :

} ← Large vector, Matrices, Array Data

Pipelining

Pipelining

- Decomposing a sequential process into suboperations
- Each subprocess is executed in a special dedicated segment concurrently

Pipelining Example

- Multiply and add operation : $Ai * Bi + Ci$ (for i = 1, 2, …, 7)
- 3 Suboperation Segment

»1) $R1 \leftarrow Ai, R2 \leftarrow Bi$: Input Ai and Bi

»2) $R3 \leftarrow R1 * R2, R4 \leftarrow Ci$: Multiply and input Ci

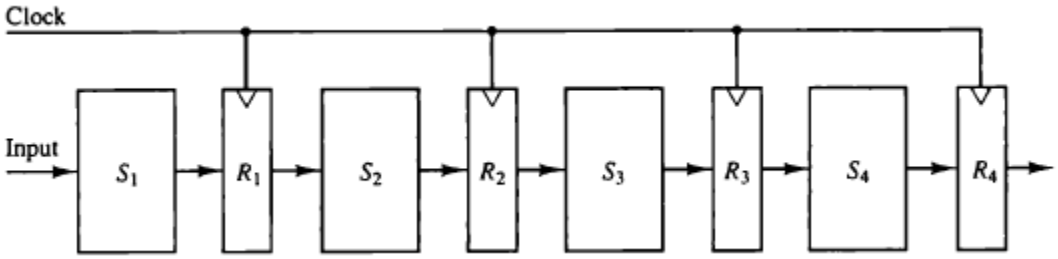
»3) $R5 \leftarrow R3 + R4$: Add Ci

Clock Pulse Number	Segment 1		Segment 2		Segment 3
	$R1$	$R2$	$R3$	$R4$	$R5$
1	A_1	B_1	—	—	—
2	A_2	B_2	$A_1 * B_1$	C_1	—
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
6	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
7	A_7	B_7	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
8	—	—	$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
9	—	—	—	—	$A_7 * B_7 + C_7$

Pipelining

General considerations

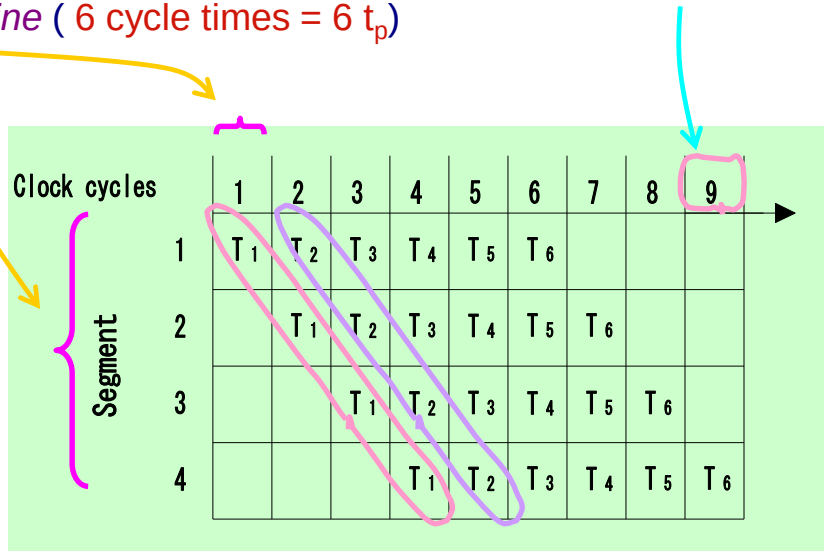
- 4 segment pipeline :
  - » **S** : Combinational circuit for Suboperation
  - » **R** : Register(intermediate results between the segments)
- Space-time diagram :
  - » Show segment utilization as a function of time
- Task : T1, T2, T3,…, T6
  - » Total operation performed going through all the segment



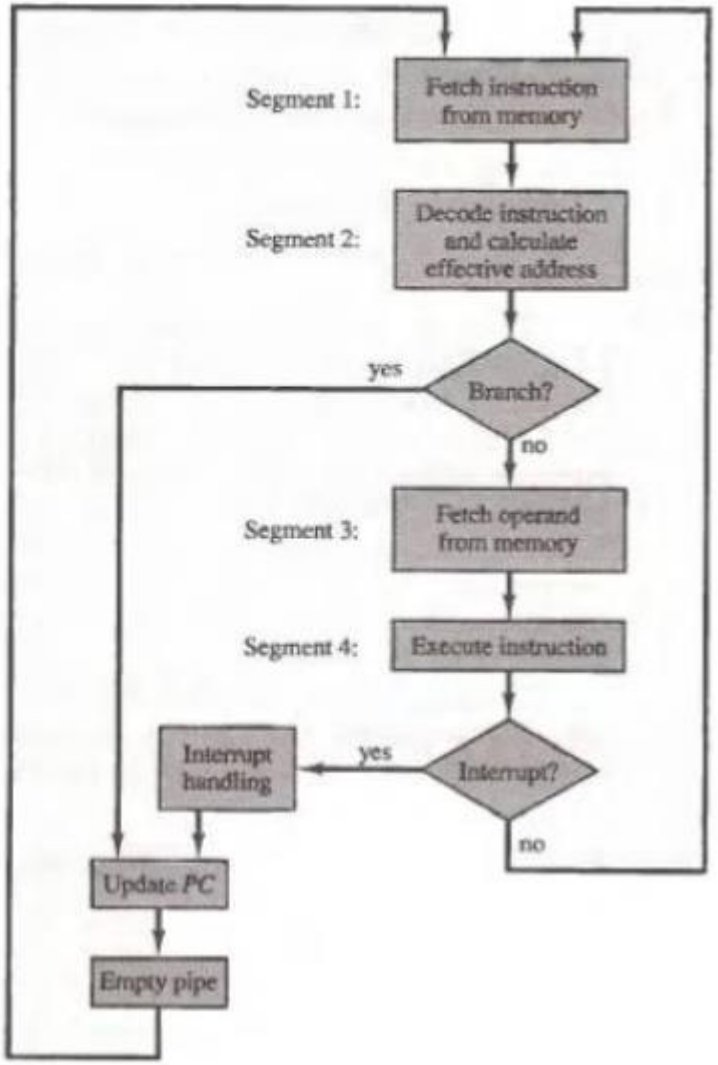
Clock cycles	1	2	3	4	5	6	7	8	9
Segment 1	T_1	T_2	T_3	T_4	T_5	T_6			
Segment 2		T_1	T_2	T_3	T_4	T_5	T_6		
Segment 3			T_1	T_2	T_3	T_4	T_5	T_6	
Segment 4				T_1	T_2	T_3	T_4	T_5	T_6

Speedup S : Nonpipeline / Pipeline

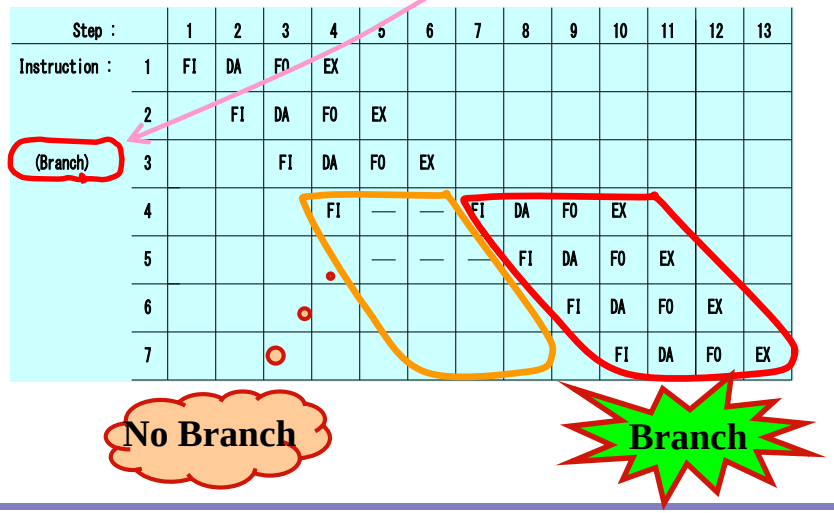
- With pipeline: k-segment pipeline with a clock time t_p to execute n tasks
- Without pipeline: Each task takes t_n
- **$S = n \cdot t_n / (k + n - 1) \cdot t_p$** $= 6 \cdot 6\, t_n / (4 + 6 - 1) \cdot t_p = 36\, t_n / 9\, t_n = 4$
  - » n : *task number* (6)
  - » t_n : *time to complete each task in nonpipeline* (6 cycle times = 6 t_p)
  - » t_p : *clock cycle time* (1 clock cycle)
  - » k : *segment number* (4)

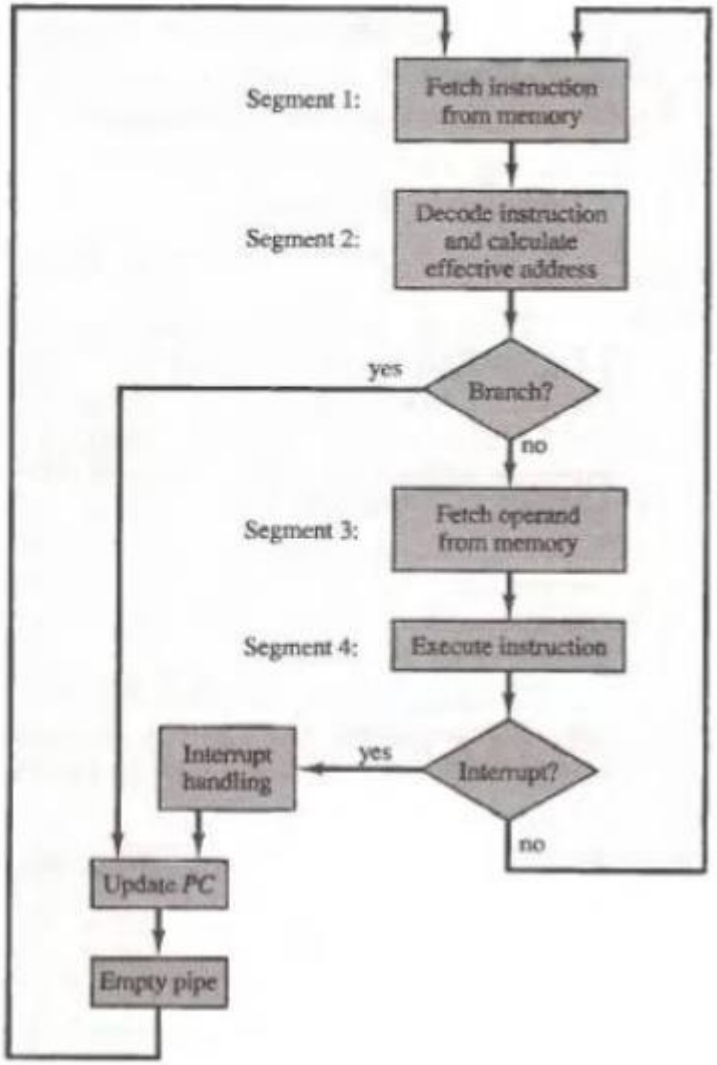


- Instruction Pipeline
  - Instruction Cycle
    1) Fetch the instruction from memory
    2) Decode the instruction
    3) Calculate the effective address
    4) Fetch the operands from memory
    5) Execute the instruction
    6) Store the result in the proper place



- Instruction Pipeline
  - Example : Four-segment Instruction Pipeline
    - Four-segment CPU pipeline :
      - » 1) **FI** : Instruction Fetch
      - » 2) **DA** : Decode Instruction & calculate EA
      - » 3) **FO** : Operand Fetch
      - » 4) **EX** : Execution
    - Timing of Instruction Pipeline :
      - » **Instruction 3** **Branch**





- Pipeline Conflicts : 3 major difficulties
  - 1) Resource conflicts
    - » memory access by two segments at the same time
  - 2) Data dependency
    - » when an instruction depend on the result of a previous instruction, but this result is not yet available
  - 3) Branch difficulties
    - » branch and other instruction (interrupt, ret, ..) that change the value of PC
- Data Dependency
  - Hardware
    - » Hardware Interlock
      - previous instruction Hardware Delay
    - » Operand Forwarding
      - previous instruction
  - Software
    - » Delayed Load
      - previous instruction No-operation instruction

- Delayed Branch
  - » 1) No-operation instruction
  - » 2) Instruction Rearranging

Clock cycles :	1	2	3	4	5	6	7	8	9	10
1. Load	I	A	E							
2. Increment		I	A	E						
3. Add			I	A	E					
4. Subtract				I	A	E				
5. Branch to X					I	A	E			
6. No-operation						I	A	E		
7. No-operation							I	A	E	
8. Instruction in X								I	A	E

(a) Using no-operation instructions

Clock cycles :	1	2	3	4	5	6	7	8
1. Load	I	A	E					
2. Increment		I	A	E				
3. Branch to X			I	A	E			
4. Add				I	A	E		
5. Subtract					I	A	E	
6. Instruction in X						I	A	E

(b) Rearranging the instructions

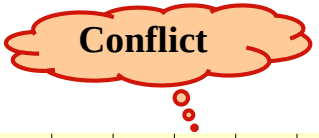
9-5 RISC Pipeline

RISC CPU

- Instruction Pipeline
- Single-cycle instruction execution
- Compiler support

Example : Three-segment Instruction Pipeline

- 3 Suboperations Instruction Cycle
  - » 1) **I** : Instruction fetch
  - » 2) **A** : Instruction decoded and ALU operation
  - » 3) **E** : Transfer the output of ALU to a register, memory, or PC
- Delayed Load :
  - » Instruction(**ADD R1 + R3**) Conflict
  - » Delayed Load
    - No-operation
- Delayed Branch :



Clock cycles :	1	2	3	4	5	6
1. Load R1	I	A	E			
2. Load R2		I	A	E		
3. Add R1+R2			I	A	E	
4. Store R3				I	A	E

(a) Pipeline timing with data conflict

Clock cycles :	1	2	3	4	5	6	7
1. Load R1	I	A	E				
2. Load R2		I	A	E			
3. No-operation			I	A	E		
4. Add R1+R2				I	A	E	
5. Store R3					I	A	E

(b) Pipeline timing with delayed load

9-5 RISC Pipeline

Example : Three-segment Instruction Pipeline

- 3 Suboperations Instruction Cycle
  - » 1) **I** : Instruction fetch
  - » 2) **A** : Instruction decoded and ALU operation
  - » 3) **E** : Transfer the output of ALU to a register, memory, or PC
- Delayed Branch :

Clock cycles:	1	2	3	4	5	6	7	8	9	10
1. Load	I	A	E							
2. Increment		I	A	E						
3. Add			I	A	E					
4. Subtract				I	A	E				
5. Branch to *X*					I	A	E			
6. No-operation						I	A	E		
7. No-operation							I	A	E	
8. Instruction in *X*								I	A	E

(a) Using no-operation instructions

Clock cycles:	1	2	3	4	5	6	7	8
1. Load	I	A	E					
2. Increment		I	A	E				
3. Branch to *X*			I	A	E			
4. Add				I	A	E		
5. Subtract					I	A	E	
6. Instruction in *X*						I	A	E

9-6 Vector Processing

Science and Engineering Applications

- Long-range weather forecasting, Petroleum explorations, Seismic data analysis, Medical diagnosis, Aerodynamics and space flight simulations, Artificial intelligence and expert systems, Mapping the human genome, Image processing

Vector Operations

- Arithmetic operations on large arrays of numbers
- Conventional scalar processor
  - » Machine language

```
      Initialize I = 0
20    Read A(I)
      Read B(I)
      Store C(I) = A(I) + B(I)
      Increment I = I + 1
      If  I ≤ 100 go to 20
      Continue
```

  - » Fortran language

```
      DO  20  I = 1, 100
20    C(I) = A(I) + B(I)
```

- Vector processor
  - » Single vector instruction

```
C(1:100) = A(1:100) + B(1:100)
```

Vector Instruction Format :

Operation code	Base address source 1	Base address source 2	Base address destination	Vector length
ADD	A	B	C	100

Matrix Multiplication

- 3 x 3 matrices multiplication : **n^2 = 9** inner product

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

» $c_{11} = a_{11} b_{11} + a_{12} b_{21} + a_{13} b_{31}$: inner product **9**

- Cumulative multiply-add operation : **n^3 = 27** multiply-add

$c = c + a \times b$

» $c_{11} = c_{11} + a_{11} b_{11} + a_{12} b_{21} + a_{13} b_{31}$: multiply-add

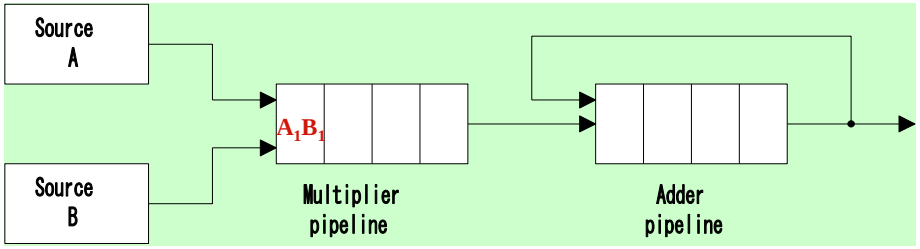
① ① ② ② ③ ③ 9 X 3 multiply-add = **27**

Initialize C_{11} = 0

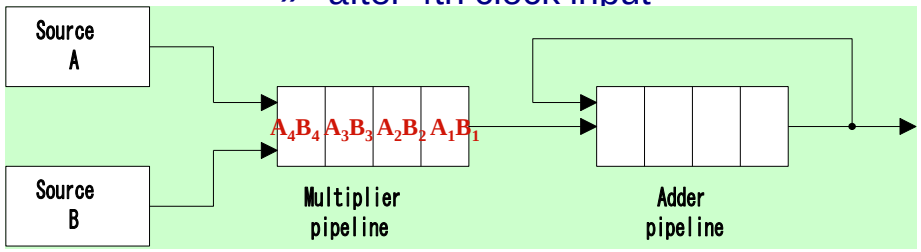
Pipeline for calculating an inner product :

- Floating point multiplier pipeline : 4 segment
- Floating point adder pipeline : 4 segment
- $C = A_1B_1 + A_2B_2 + A_3B_3 + \cdots + A_kB_k$

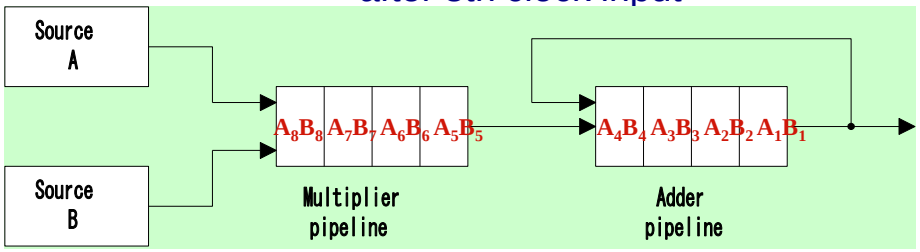
» after 1st clock input



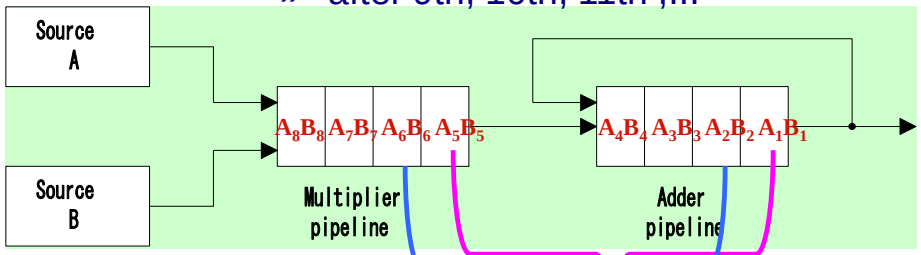
» after 4th clock input



» after 8th clock input



» after 9th, 10th, 11th ,...



» Four section summation

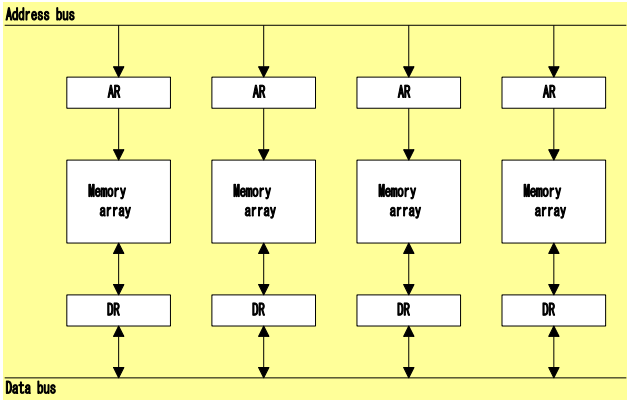
$$
\begin{aligned}
C = {} & A_1B_1 + A_5B_5 + A_9B_9 + A_{13}B_{13} + \cdots \\
& + A_2B_2 + A_6B_6 + A_{10}B_{10} + A_{14}B_{14} + \cdots \\
& + A_3B_3 + A_7B_7 + A_{11}B_{11} + A_{15}B_{15} + \cdots \\
& + A_4B_4 + A_8B_8 + A_{12}B_{12} + A_{16}B_{16} + \cdots
\end{aligned}
$$

$A_2B_2 + A_6B_6$ $A_1B_1 + A_5B_5$

, , , ⑩ ⑨

Memory Interleaving :

- *Simultaneous* access to memory from two or more source using *one memory bus system*
- Even / Odd Address Memory Access



Supercomputer

- Supercomputer = Vector Instruction + Pipelined floating-point arithmetic
- Performance Evaluation Index
  - » **MIPS** : Million Instruction Per Second
  - » **FLOPS** : Floating-point Operation Per Second
    - megaflops : 10^6, gigaflops : 10^9
- Cray supercomputer : Cray Research
  - » Clay-1 : 80 megaflops, 4 million 64 bit words memory
  - » Clay-2 : 12 times more powerful than the clay-1
- VP supercomputer : Fujitsu
  - » VP-200 : 300 megaflops, 32 million memory, 83 vector instruction, 195 scalar instruction
  - » VP-2600 : 5 gigaflops