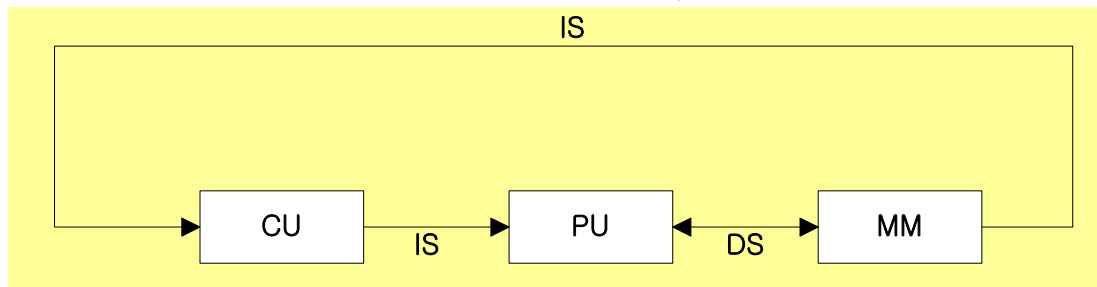
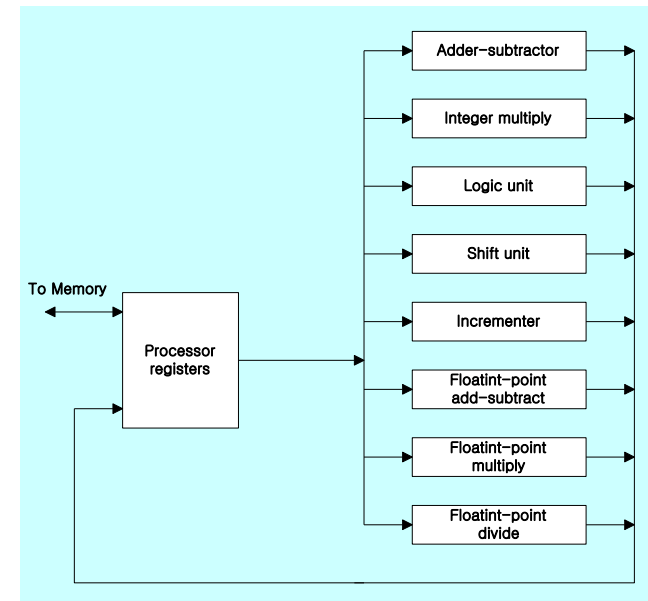


Pipeline and Vector Processing

■ Parallel Processing

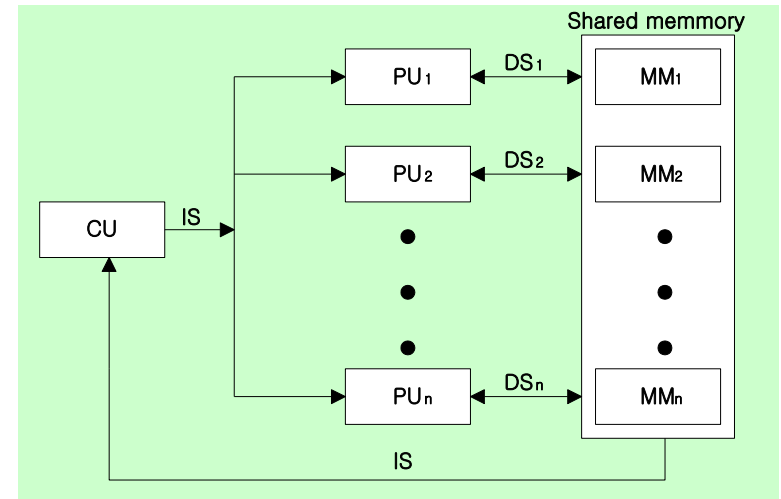
- ◆ *Simultaneous* data processing tasks for the purpose of increasing the computational speed
- ◆ Perform *concurrent* data processing to achieve faster execution time
- ◆ Multiple Functional Unit :
- ◆ *Separate the execution unit into eight functional units operating in parallel*
- ◆ Computer Architectural 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



- 2) **SIMD**

(Single Instruction - Multiple Data stream)

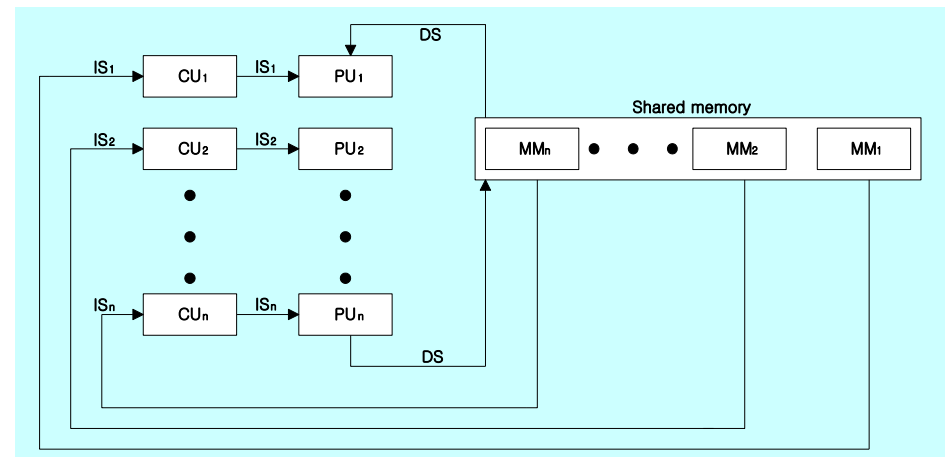
- » vector or array operations one vector operation includes many operations on a data stream
- » Example systems : CRAY -1, ILLIAC-IV



- 3) **MISD**

(Multiple Instruction - Single Data stream)

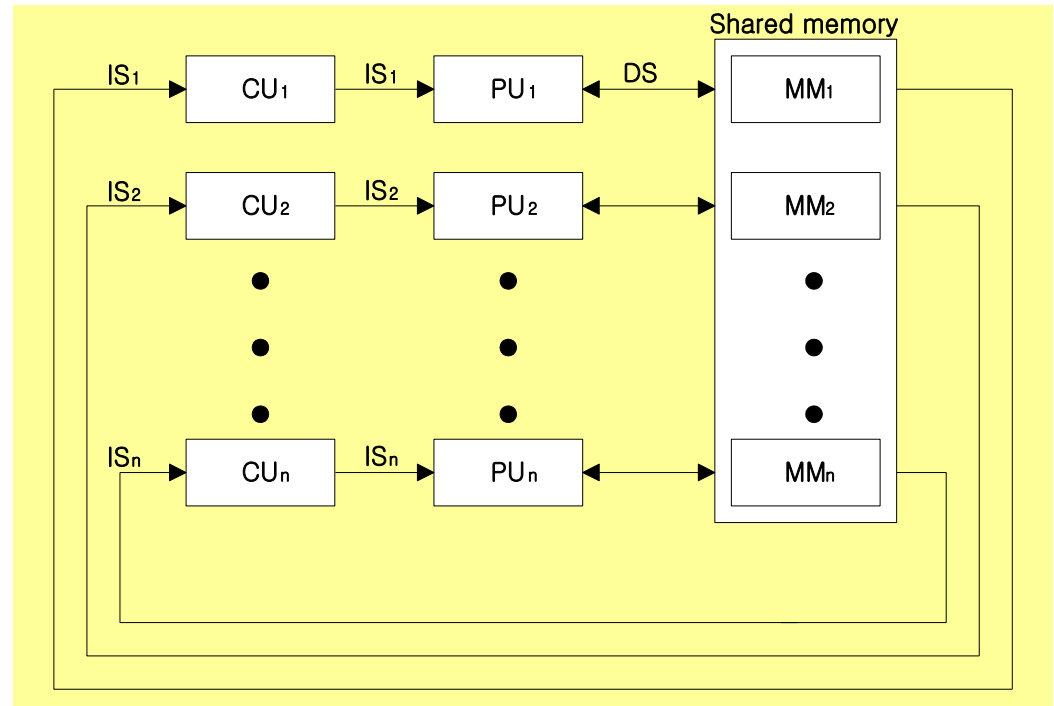
- » Data Stream Bottle neck



- 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

- Multiply and add operation : $A_i * B_i + C_i$ (for $i = 1, 2, \dots, 7$)
- 3 Suboperation Segment
 - » 1) $R1 \leftarrow A_i, R2 \leftarrow B_i$: Input A_i and B_i
 - » 2) $R3 \leftarrow R1 * R2, R4 \leftarrow C_i$: Multiply and input C_i
 - » 3) $R5 \leftarrow R3 + R4$: Add C_i

◆ 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 : $T_1, T_2, T_3, \dots, T_6$
 - » Total operation performed going through all the segments

Segment
versus
clock-cycle

◆ Speedup S : Nonpipeline / Pipeline

- $S = n \cdot t_n / (k + n - 1) \cdot t_p = 6 \cdot 6 t_n / (4 + 6 - 1) \cdot t_p = 36 t_n / 9 t_n = 4$

- » 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)

- If $n \rightarrow \infty$ $S = t_n / t_p$

- task

$$\text{nonpipeline} (t_n) = \text{pipeline} (k \cdot t_p)$$

$$S = t_n / t_p = k \cdot t_p / t_p = k$$

(**segment**)

Pipeline = 9 clock cycles

Clock cycles	1	2	3	4	5	6	7	8	9
1	T ₁	T ₂	T ₃	T ₄	T ₅	T ₆			
2		T ₁	T ₂	T ₃	T ₄	T ₅	T ₆		
3			T ₁	T ₂	T ₃	T ₄	T ₅	T ₆	
4				T ₁	T ₂	T ₃	T ₄	T ₅	T ₆

◆ Pipeline Arithmetic Pipeline, Instruction Pipeline

■ Arithmetic Pipeline

◆ Floating-point Adder Pipeline Example

- Add / Subtract two normalized floating-point binary number

- » $X = A \times 2^a = 0.9504 \times 10^3$

- » $Y = B \times 2^b = 0.8200 \times 10^2$

- 4 segments suboperations

- » 1) Compare exponents by subtraction :

$$3 - 2 = 1$$

- $X = 0.9504 \times 10^3$

- $Y = 0.8200 \times 10^2$

- » 2) Align mantissas

- $X = 0.9504 \times 10^3$

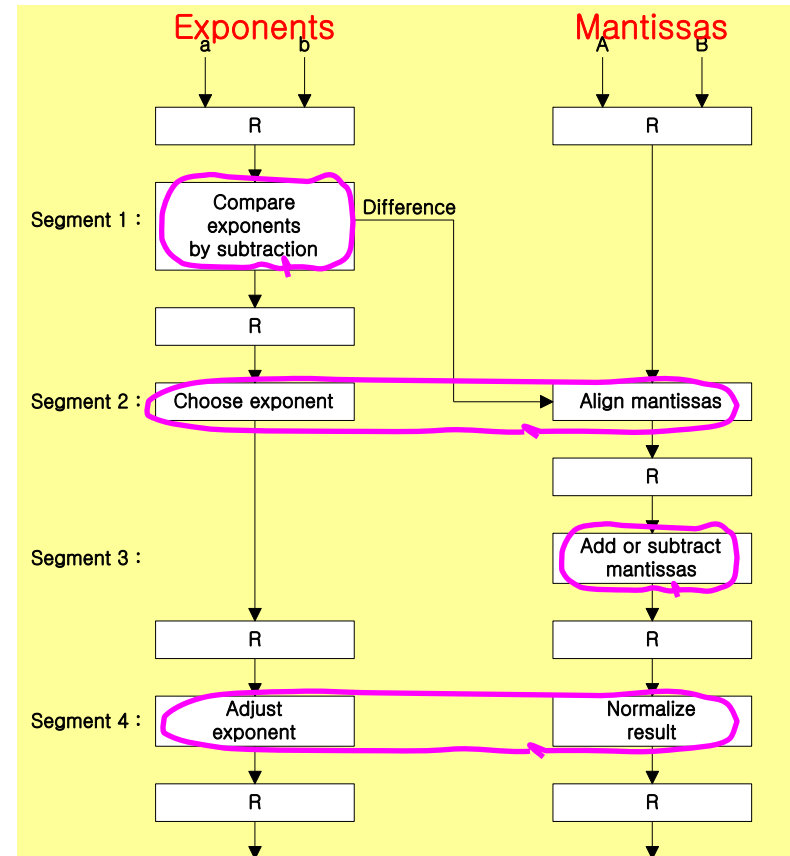
- $Y = 0.08200 \times 10^3$

- » 3) Add mantissas

- $Z = 1.0324 \times 10^3$

- » 4) Normalize result

- $Z = 0.1324 \times 10^4$



■ 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

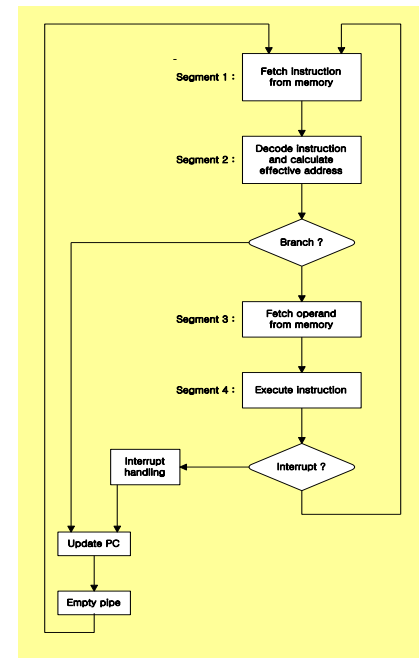
◆ Example : Four-segment Instruction Pipeline

- 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**



Step :	1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction : 1	FI	DA	FO	EX									
2		FI	DA	FO	EX								
(Branch) 3			FI	DA	FO	EX							
4				FI	—	—	FI	DA	FO	EX			
5					—	—	—	FI	DA	FO	EX		
6									FI	DA	FO	EX	
7										FI	DA	FO	EX

No Branch

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
 - » Operand Forwarding
 - previous instruction
- Software
 - » Delayed Load
 - previous instruction No-operation instruction

◆ Handling of Branch Instructions

- Prefetch target instruction
 - » Conditional branch에서 branch target instruction instruction fetch

- Branch Target Buffer : **BTB**

- » 1) Associative memory branch target address instruction.
- » 2) 만약 branch instruction

- Loop Buffer

- » 1) small very high speed register file (RAM)
- » 2) loop

Loop Buffer load access

- Branch Prediction

- » Branch predict하는 additional hardware logic

◆ Delayed Branch

- branch instruction

pipeline operation

- » 1) No-operation instruction
- » 2) Instruction Rearranging : **Compiler**

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

■ 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
 - 4 clock cycle Instruction (**LOAD R2**)
 - 3 instruction **R2**
 - » 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

■ 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
<i>ADD</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>100</i>

◆ 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

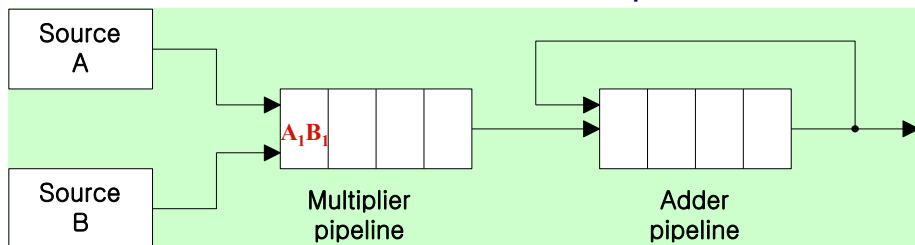
$C_{11} = 0$

◆ Pipeline for calculating an inner product :

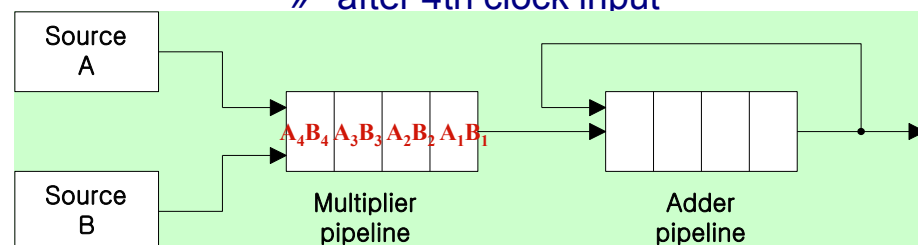
- Floating point multiplier pipeline : 4 segment
- Floating point adder pipeline : 4 segment

$$C = A_1B_1 + A_2B_2 + A_3B_3 + \dots + A_kB_k$$

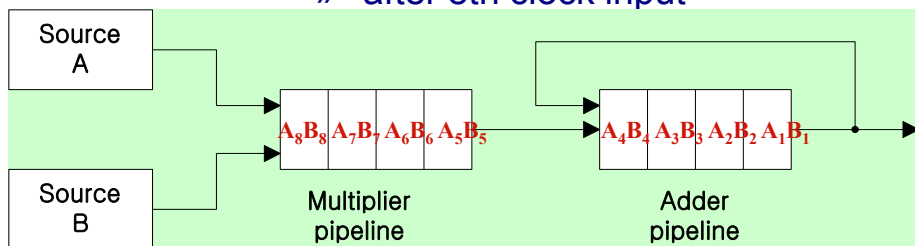
» after 1st clock input



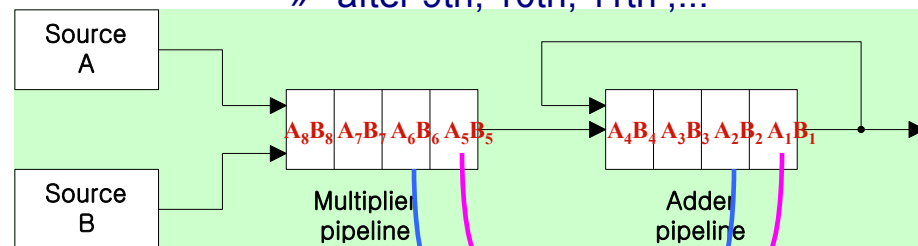
» after 4th clock input



» after 8th clock input



» after 9th, 10th, 11th ,...



» Four section summation

$$C = A_1B_1 + A_5B_5 + A_9B_9 + A_{13}B_{13} + \dots$$

$$+ A_2B_2 + A_6B_6 + A_{10}B_{10} + A_{14}B_{14} + \dots$$

$$+ A_3B_3 + A_7B_7 + A_{11}B_{11} + A_{15}B_{15} + \dots$$

$$+ A_4B_4 + A_8B_8 + A_{12}B_{12} + A_{16}B_{16} + \dots$$

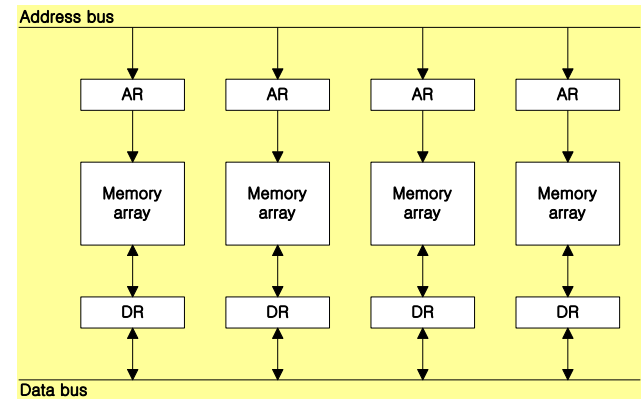
$$A_2B_2 + A_6B_6$$

,,, ⑩ ⑨

$$A_1B_1 + A_5B_5$$

◆ Memory Interleaving :

- *Simultaneous* access to memory from two or more source using *one memory bus system*
- AR 2 bit 4 memory module
- 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

■ Array Processors

◆ Performs computations on large arrays of data

Vector processing : Adder/Multiplier pipeline
Array processing : array processor

◆ Array Processing

- Attached array processor
 - » Auxiliary processor attached to a general purpose computer
- SIMD array processor
 - » Computer with multiple processing units operating in parallel

$$\begin{aligned} \blacksquare \text{ Vector } C = A + B \quad c_i = a_i + b_i \\ \text{PE}_i, 0 \end{aligned}$$

