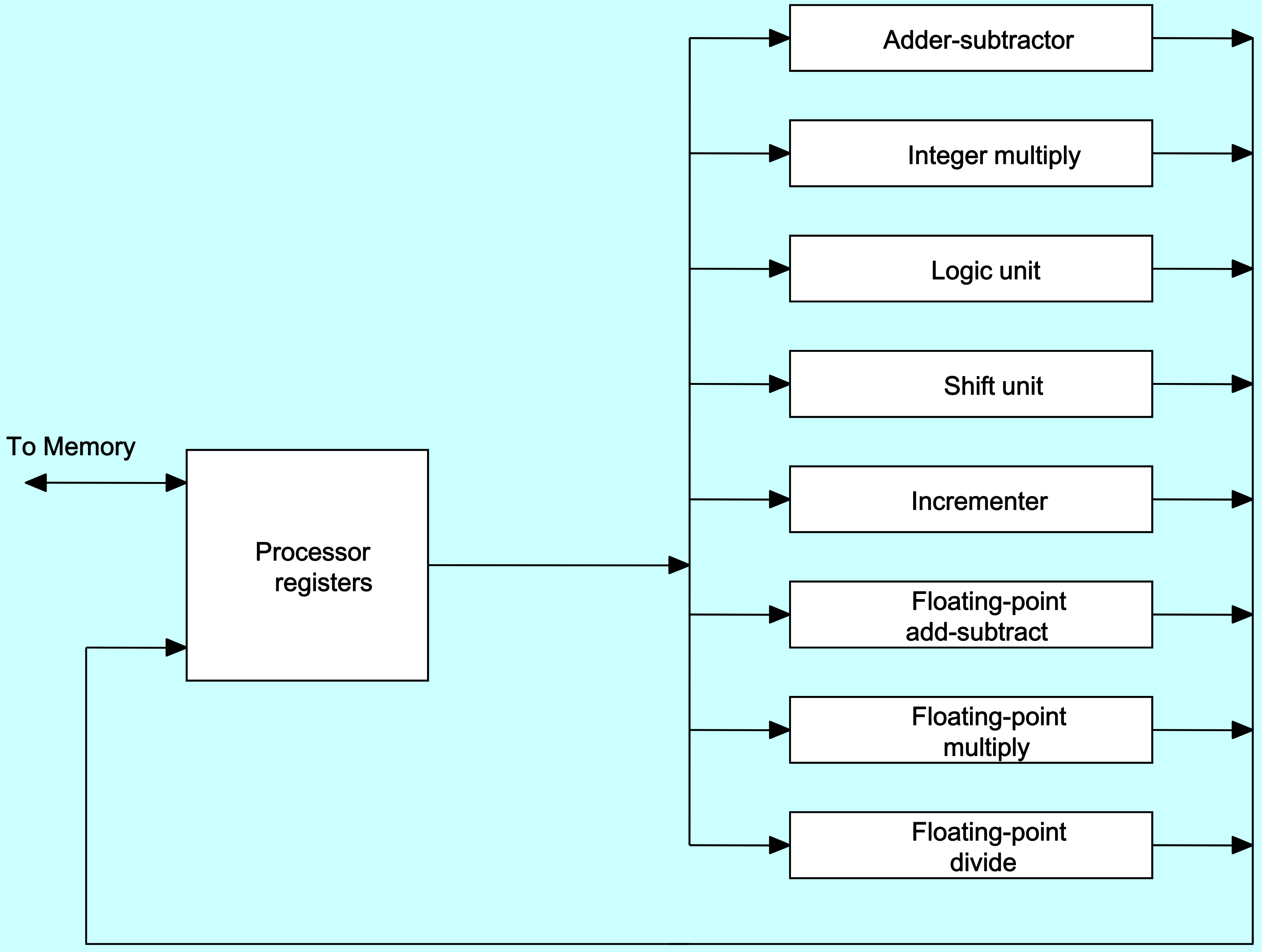
Pipeline

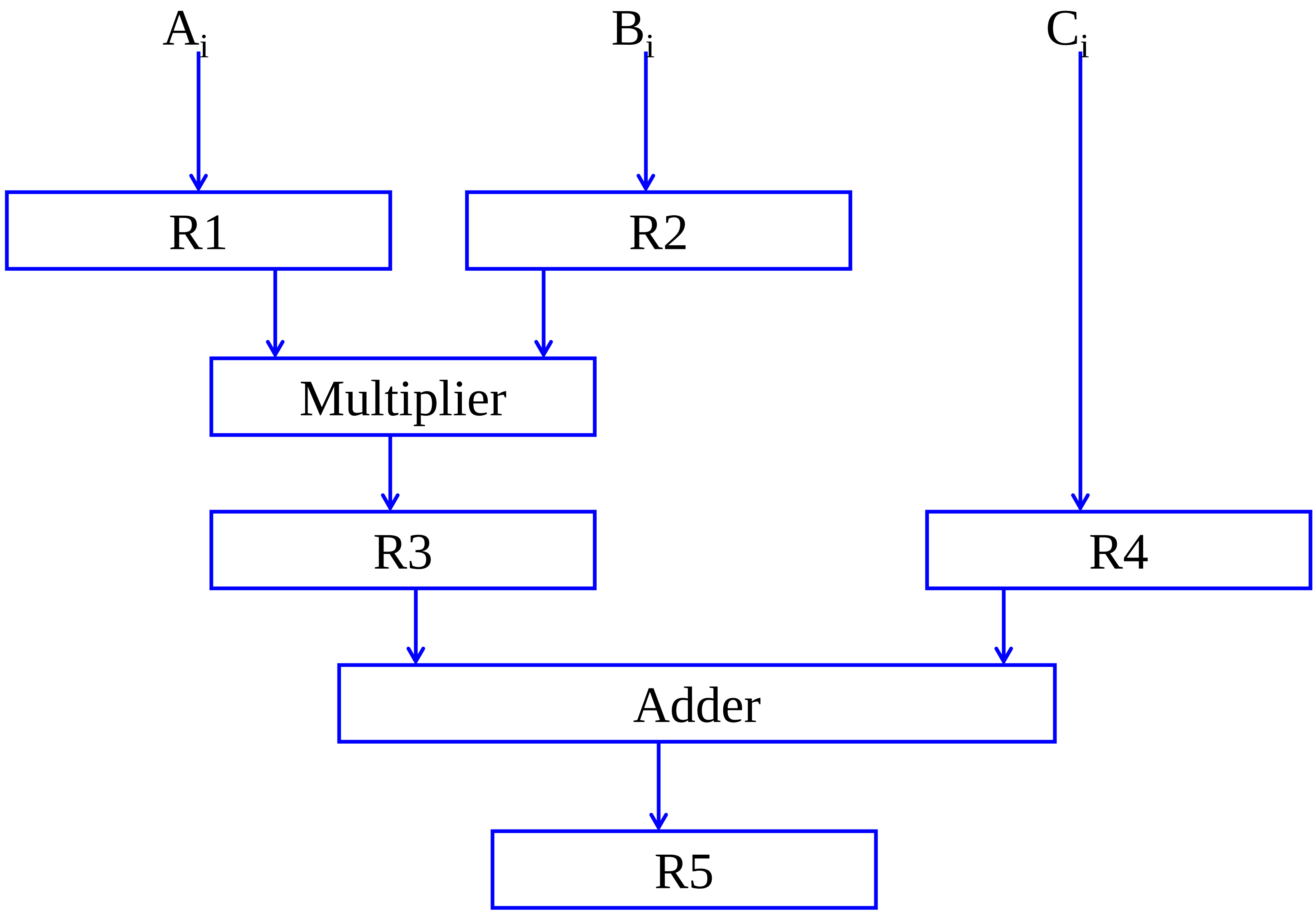
- Parallel Processing
  - *Simultaneous* data processing tasks for the purpose of increasing the computational speed
  - Perform *concurrent* data processing to achieve faster execution time
  - Multiple Functional Unit :
    - ***Separate the execution unit into eight functional units operating in parallel***


To Memory
Processor registers
Adder-subtractor
Integer multiply
Logic unit
Shift unit
Incrementer
Floating-point add-subtract
Floating-point multiply
Floating-point divide

- **Pipelining :** it is the process of Decomposing a sequential process into suboperations with Each subprocess is executed in a special dedicated segment concurrently with all other segments.
- It is a collection of processing segments through which binary information flows. Where each segment performs partial processing dedicated by the way the task is partioned.

Pipelining

- 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
- Content of registers in pipeline example :


A_i
B_i
C_i
R1
R2
Multiplier
R3
R4
Adder
R5

	Segment1		Segment 2		Segment 3
Clock pulse Number	R1	R2	R3	R4	R5
1	A1	B1	-	-	-
2	A2	B2	A1*B1	C1	-
3	A3	B3	A2*B2	C2	A1*B1+C1
4	A4	B4	A3*B3	C3	A2*B2+C2
5	A5	B5	A4*B4	C4	A3*B3+C3
6	A6	A6	A5*B5	C5	A4*B4+C4
7	A7	A7	A6*B6	C6	A5*B5+C5
8	-	-	A7*B7	C7	A6*B6+C6
9	-	-	-	-	A7*B7+C7

◆General considerations

- 4 segment pipeline : the operand pass through all four segments in a fixed sequence. Each segment consists of a combinational ckt Si that performs a sub operation over the data stream. The segments are separated by the registers to hold the intermediate results.

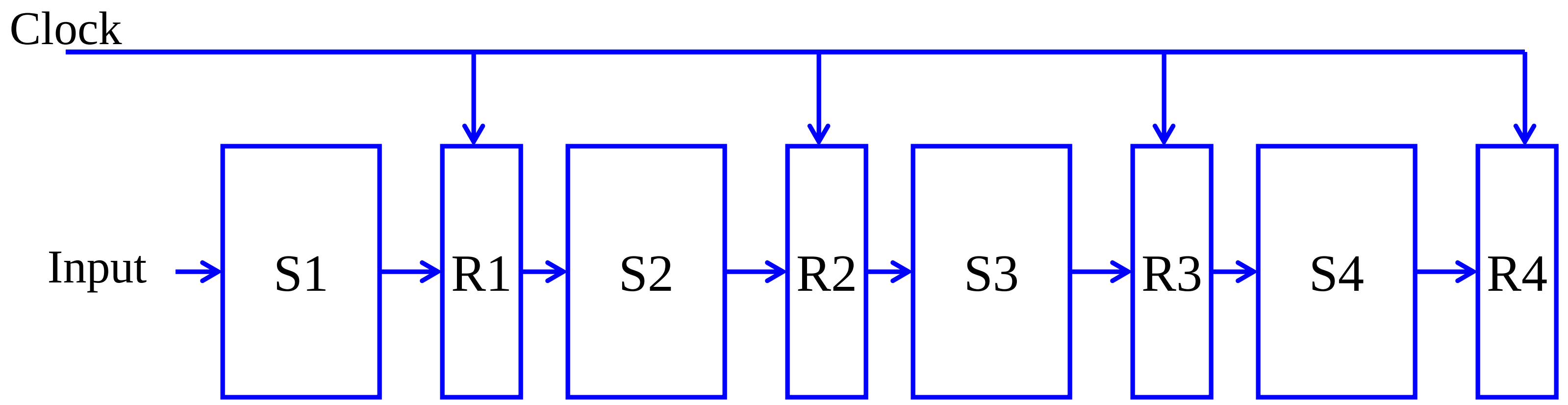


Fig.: Four Segment pipeline

- Space-time diagram :
  - »Show segment utilization as a function of time
- Task : T1, T2, T3,…, T6 executed in four segments.
  - »Total operation performed going through all the segment

Pipeline= 9 clock cycles

	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			
	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

- Speedup S : Nonpipeline / Pipeline
  - $\mathbf{S = n \cdot t_n / (k + n - 1) \cdot t_p} = 6 \cdot 6\, t_n / (4 + 6 - 1) \cdot t_p = 36\, t_n / 9\, t_n = 4$
    - » 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)

$k + n - 1 \approx n$

  - If $n \rightarrow \infty$ 이면, $\mathbf{S = t_n / t_p}$
  - If we assume that the time it takes to process a task is the same in the pipeline and nonpipeline circuits then we have

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

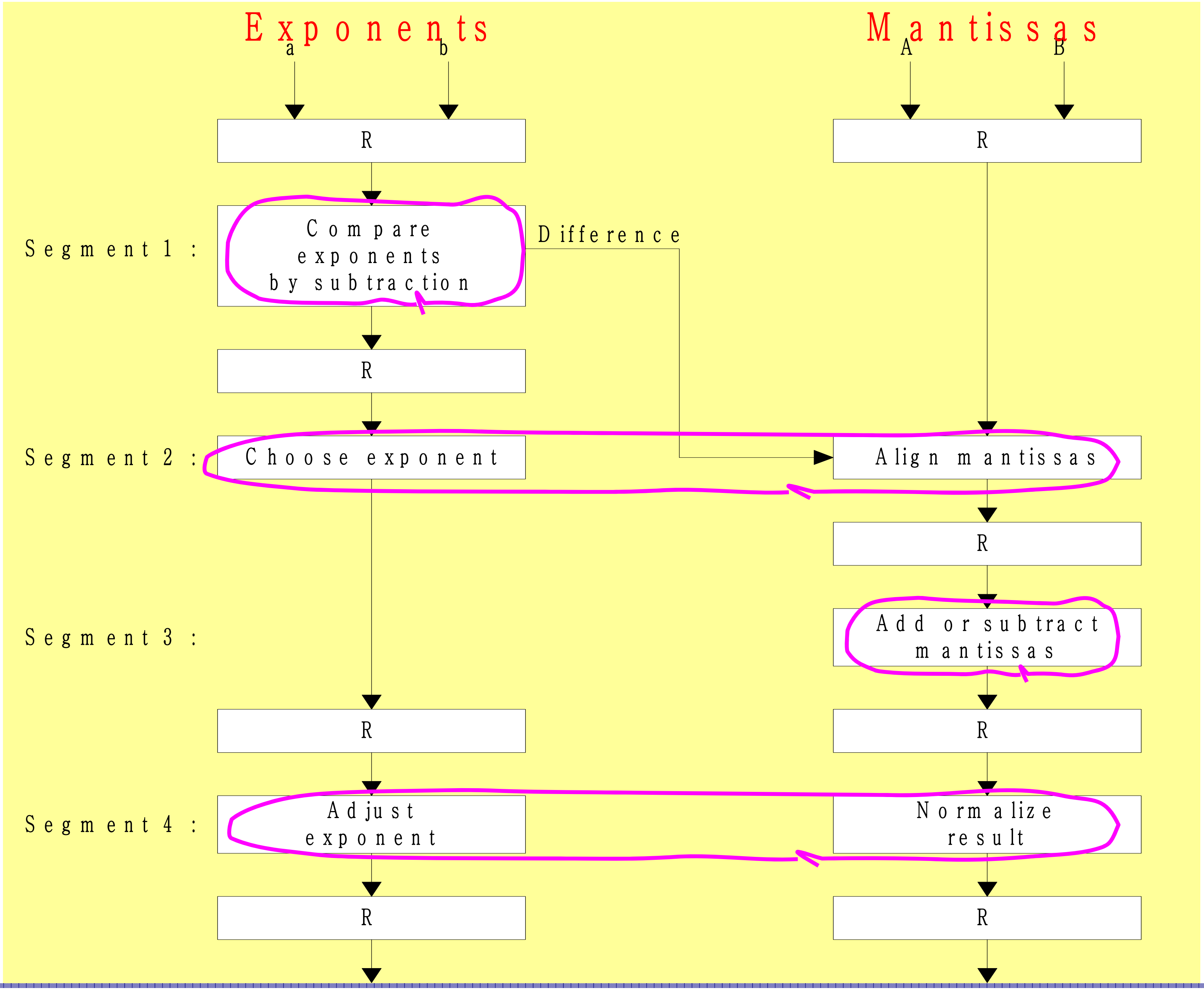
  $\mathbf{S = t_n / t_p = k \cdot t_p / t_p = k}$

  Where k is the number of segments.

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
    - $Z = 0.1324 \times 10^4$


Exponents
Mantissas
a
b
A
B
R
R
Segment 1 :
Compare exponents by subtraction
Difference
R
Segment 2 :
Choose exponent
Align mantissas
R
Segment 3 :
Add or subtract mantissas
R
R
Segment 4 :
Adjust exponent
Normalize result
R
R

Instruction Pipeline

- Instruction Cycle

  1) Fetch the instruction from memory

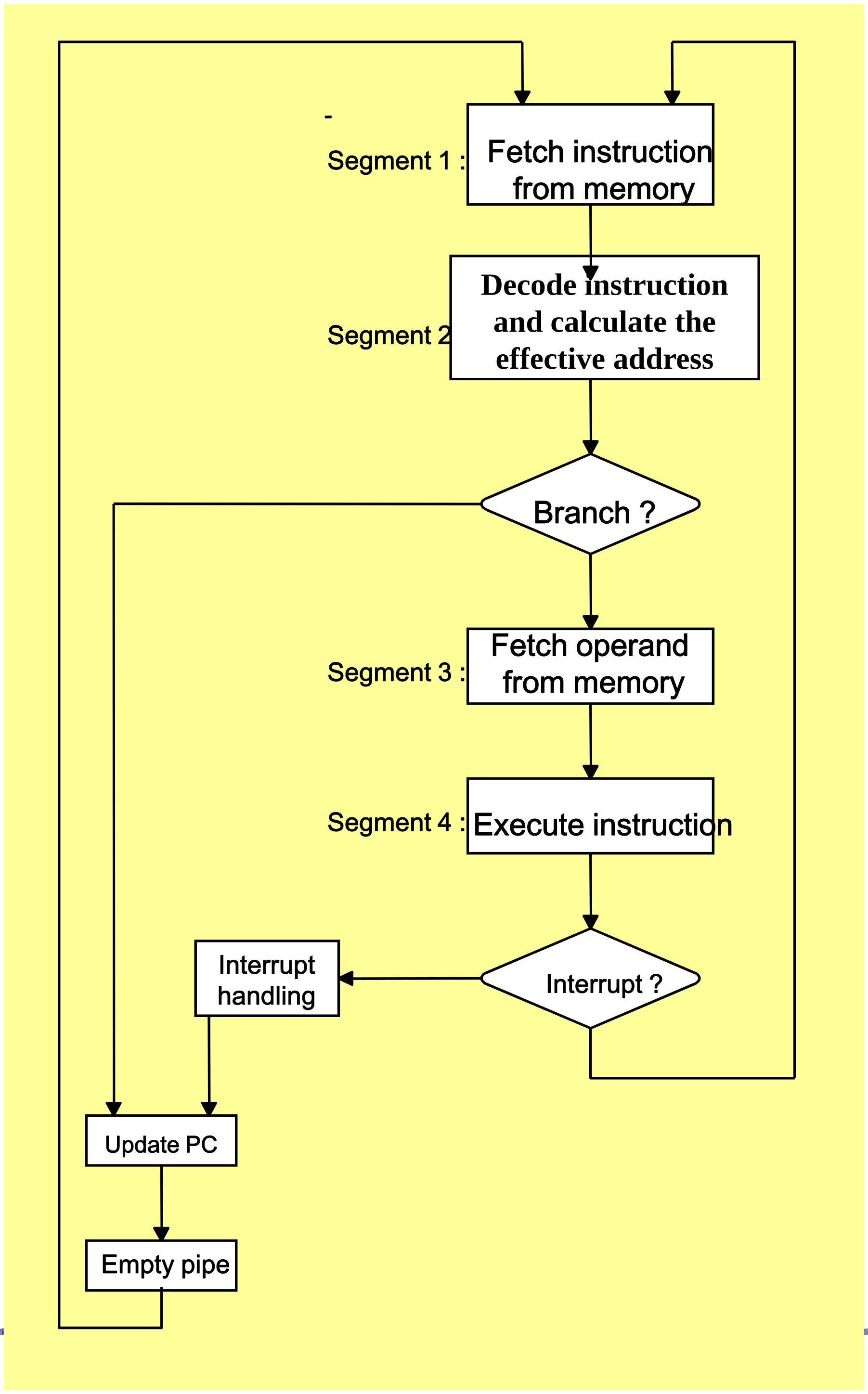
  2) Decode the instruction

  3) Calculate the effective address

  4) Fetch the operands from memory

  5) Execute the instruction

  6) Store the result in the proper place


-
Segment 1 :
Fetch instruction from memory
Segment 2
Decode instruction and calculate the effective address
Branch ?
Segment 3 :
Fetch operand from memory
Segment 4 :
Execute instruction
Interrupt handling
Interrupt ?
Update PC
Empty pipe

◆ 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 :

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.
    - » Can be avoided by using separate instruction stream and data memories.
  - 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 instruction의 결과가 나올 때 까지 Hardware 적인 Delay를 강제 삽입
    - » Operand Forwarding
      - previous instruction의 결과를 곧바로 ALU 로 전달 (정상적인 경우, register를 경유함)
  - Software 적인 방법
    - » Delayed Load
      - previous instruction의 결과가 나올 때 까지 No-operation instruction 을 삽입

Introduction: Multiprocessor

The slowdown in uniprocessor performance and growing concern over power generated a new era in computer architecture is known as multiprocessor.

Mainstream of multiprocessor design: multiprocessors with small to medium numbers of processors (4 to 32)

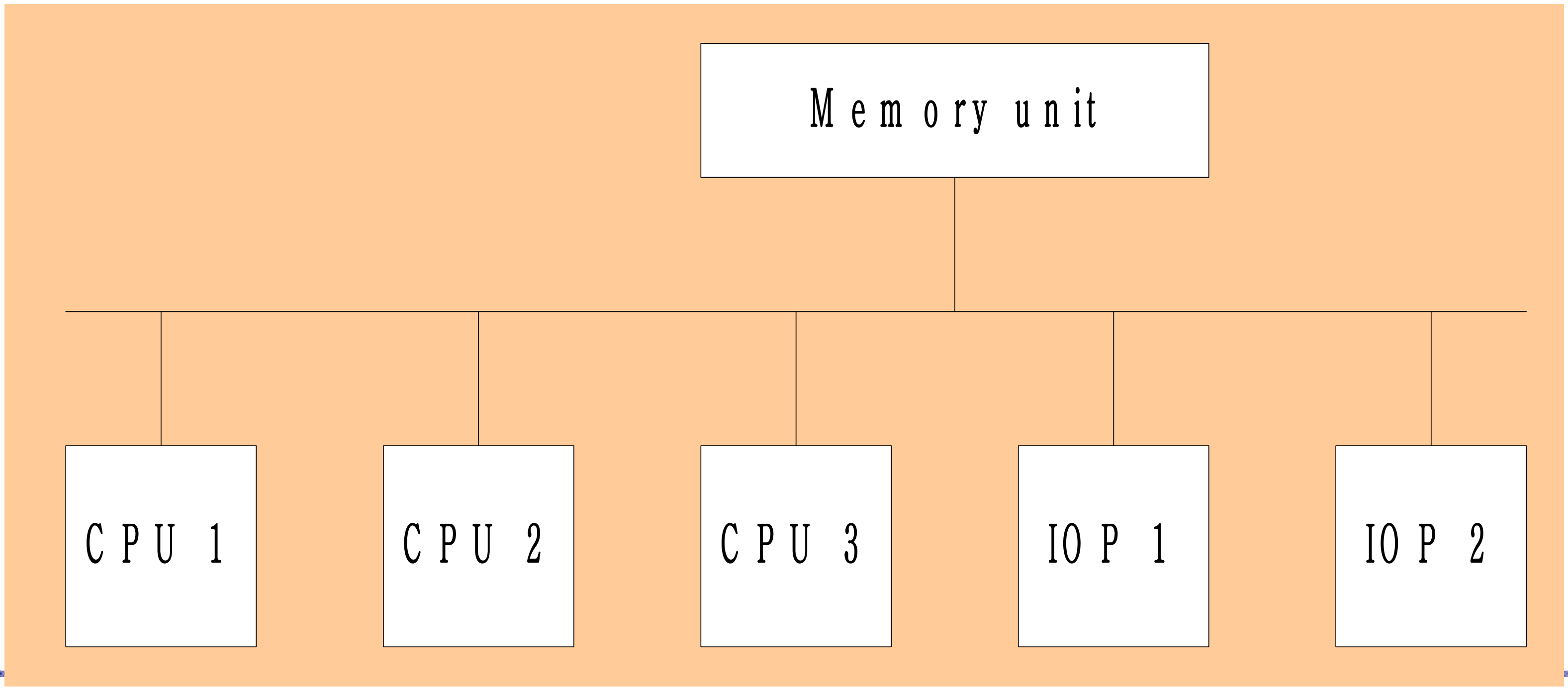
Characteristics of Multiprocessors

- Multiprocessors System = MIMD
  - An interconnection of two or more CPUs with memory and I/O equipment
    - » **a single CPU** and **one or more IOPs** is usually not included in a multiprocessor system
      - Unless the IOP has computational facilities comparable to a CPU
- Computation can proceed in parallel in one of two ways
  - 1) Multiple independent jobs can be made to operate in parallel
  - 2) A single job can be partitioned into multiple parallel tasks
- Classified by the memory Organization
  - 1) Shared memory or Tightly-coupled system: Provides a cache memory with each CPU and there is Global common memory that all CPU can access.
    - » Local memory + Shared memory
  - 2) Distribute memory or Loosely-coupled system
    - » Local memory + message passing scheme (I.e. the processors are tied together by a switching scheme designed to route information from one processor to another though message passing scheme.

- Interconnection Structure: the components that form the multiprocessor system are CPU ,IOP connected to I/O devices , and memory unit that may be portioned into a number of separate modules.
  - Multiprocessor System Components
    - 1) Time-shared common bus
    - 2) Multi-port memory
    - 3) Crossbar switch
    - 4) Multistage switching network
    - 5) Hypercube system

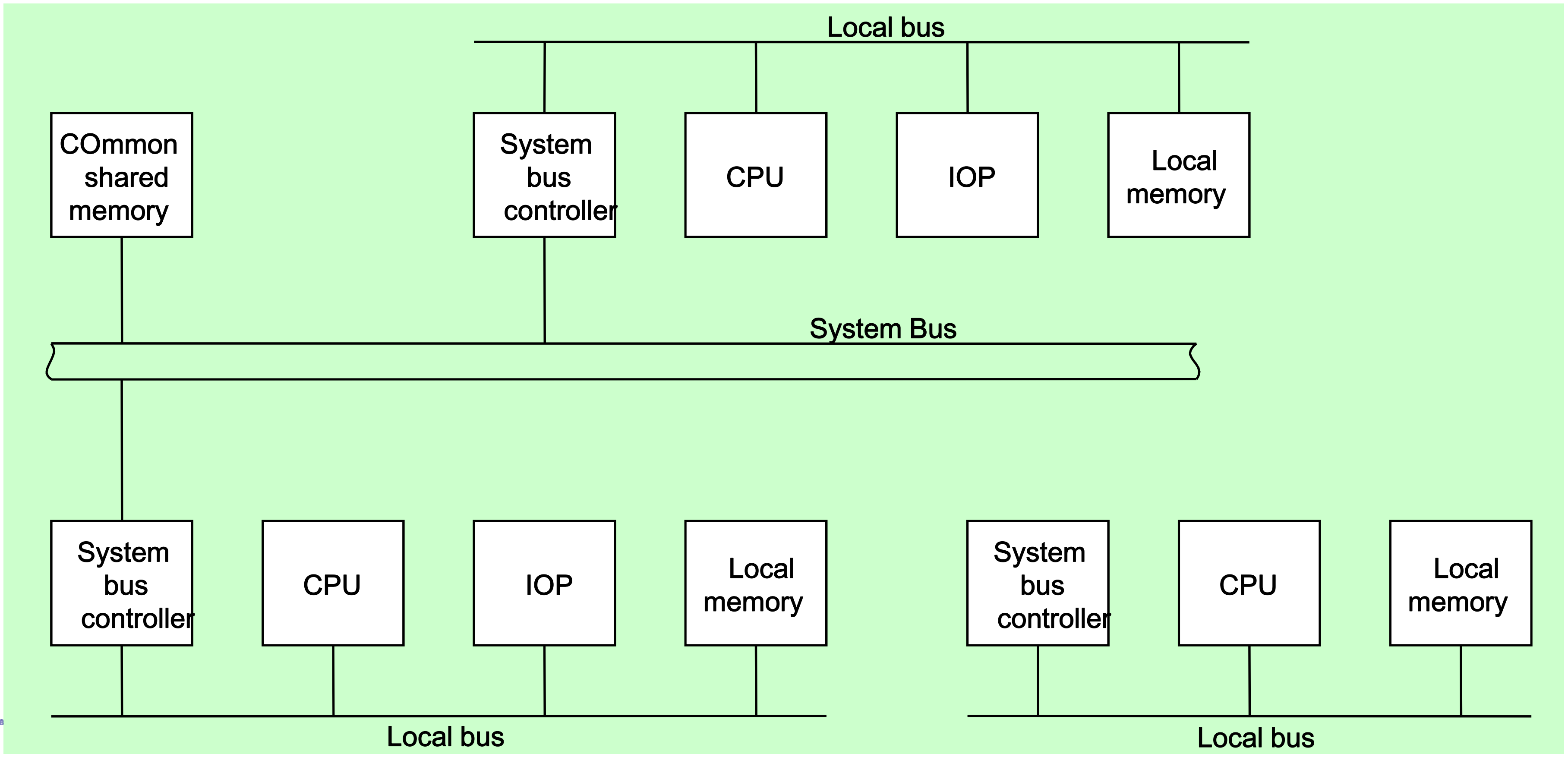
Time-shared Common Bus

- Time-shared single common bus system :
  - » Only one processor can communicate with the memory or another processor at any given time
    - when one processor is communicating with the memory, all other processors are either *busy with internal operations* or must be *idle waiting for the bus*



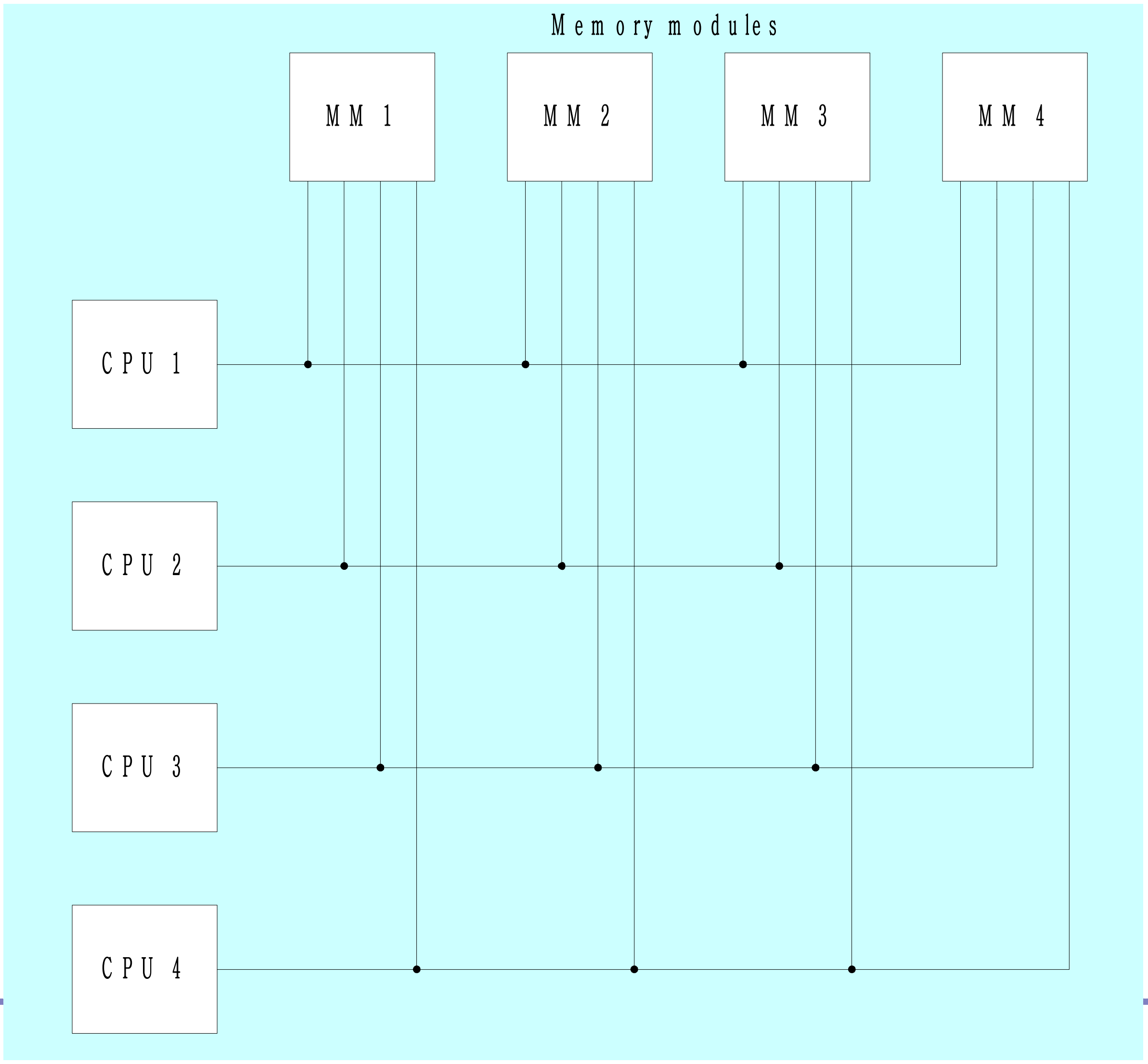
- Dual common bus system :
  - System bus + Local bus
  - Shared memory
    - the memory connected to the common system bus is shared by all processors
  - System bus controller
    - Link each local but to a common system bus

Tightly coupled system



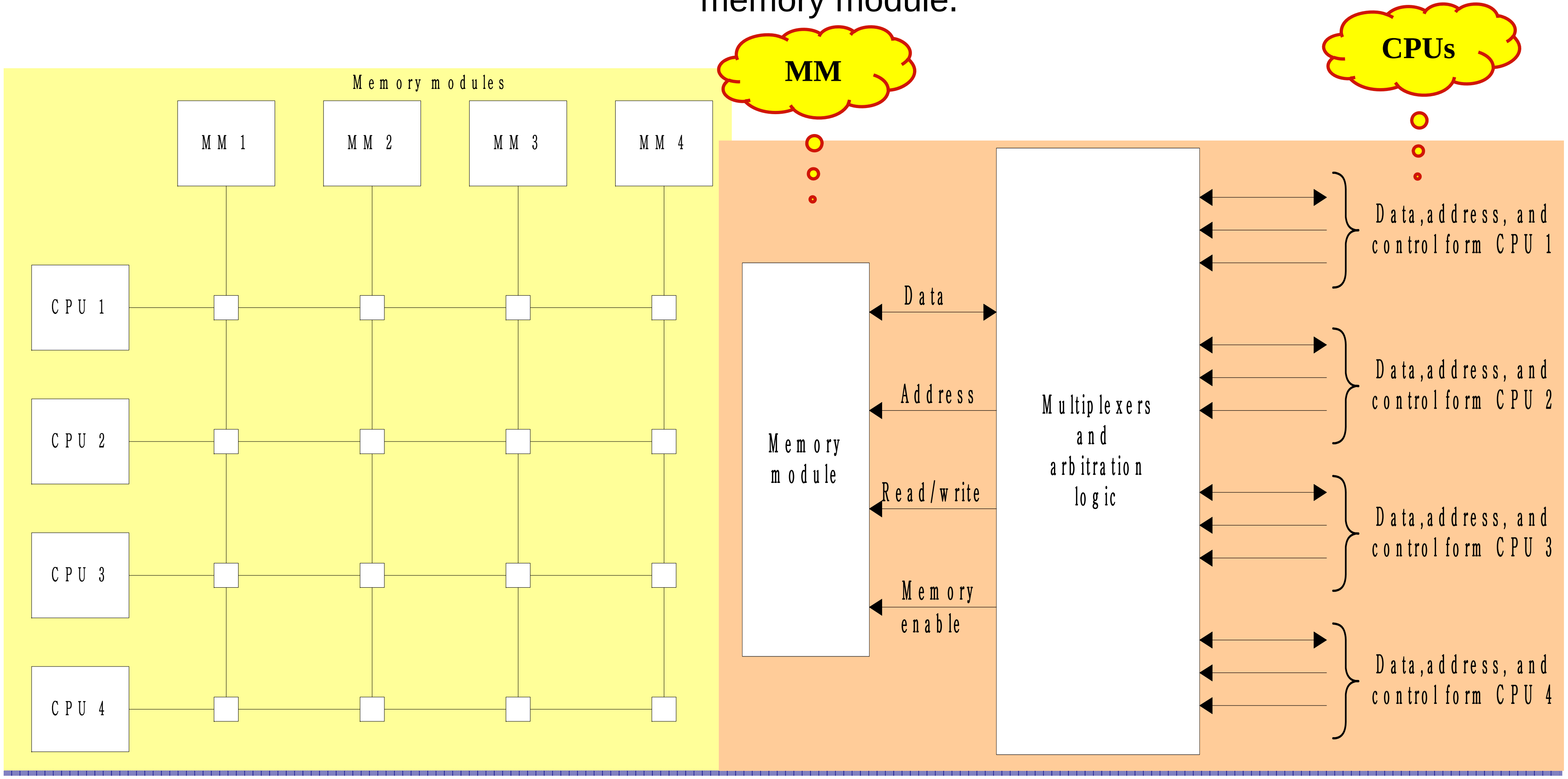
◆Multi-port memory :

- multiple paths between processors and memory
  - » Advantage : high transfer rate can be achieved
  - » Disadvantage : expensive memory control logic / large number of cables & connectors



Crossbar Switch :

- Crosspoints are placed at intersections between processor buses and memory module paths.
- A small square in each crosspoints is a switch that determines the path from a processor to a memory module.
- Each switch point has a control logic to set up the transfer path between processor and memory module.

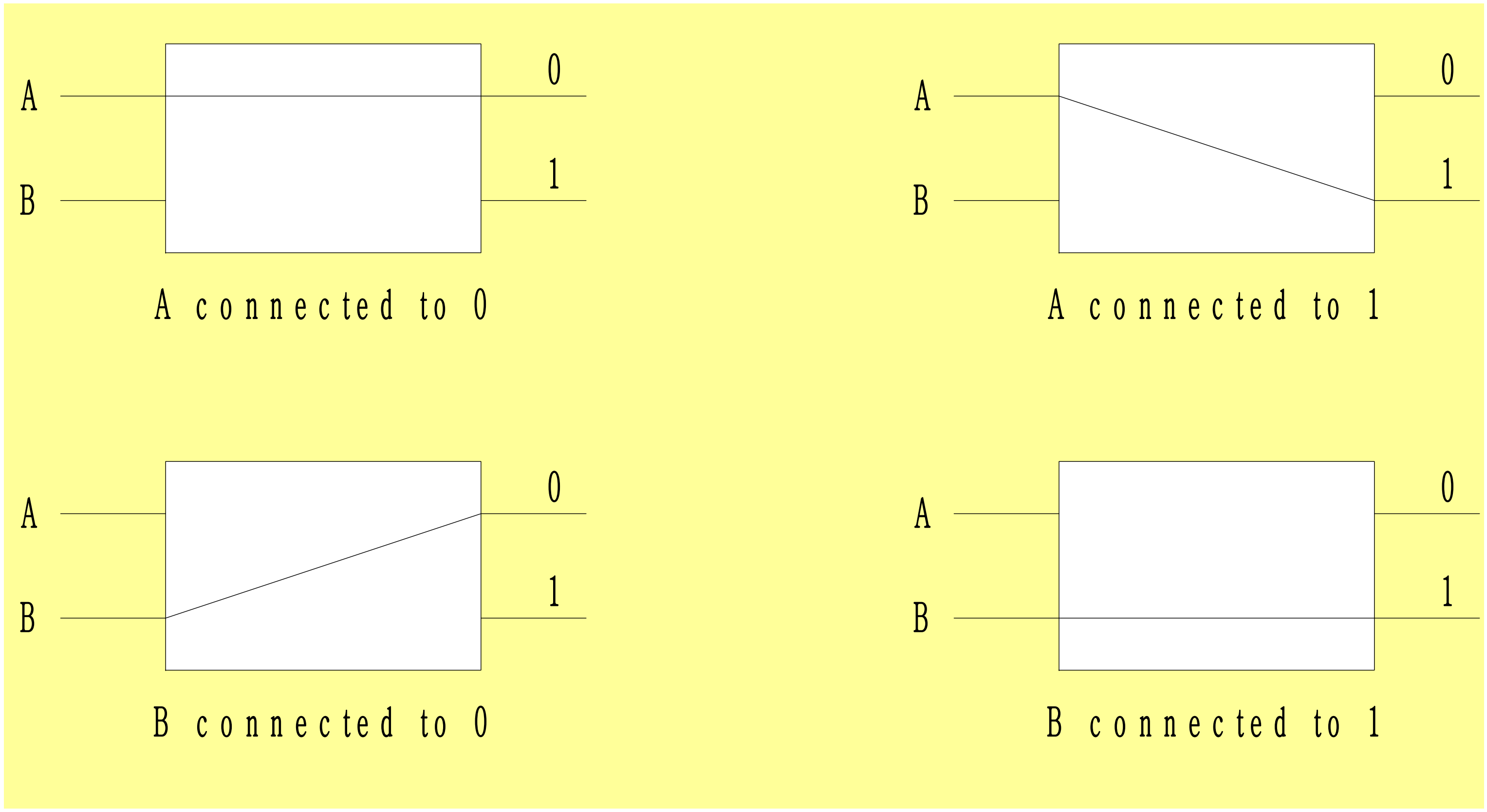


MM
CPUs
Memory modules
MM 1
MM 2
MM 3
MM 4
CPU 1
CPU 2
CPU 3
CPU 4
Memory module
Data
Address
Read/write
Memory enable
Multiplexers and arbitration logic
Data,address, and control form CPU 1
Data,address, and control form CPU 2
Data,address, and control form CPU 3
Data,address, and control form CPU 4

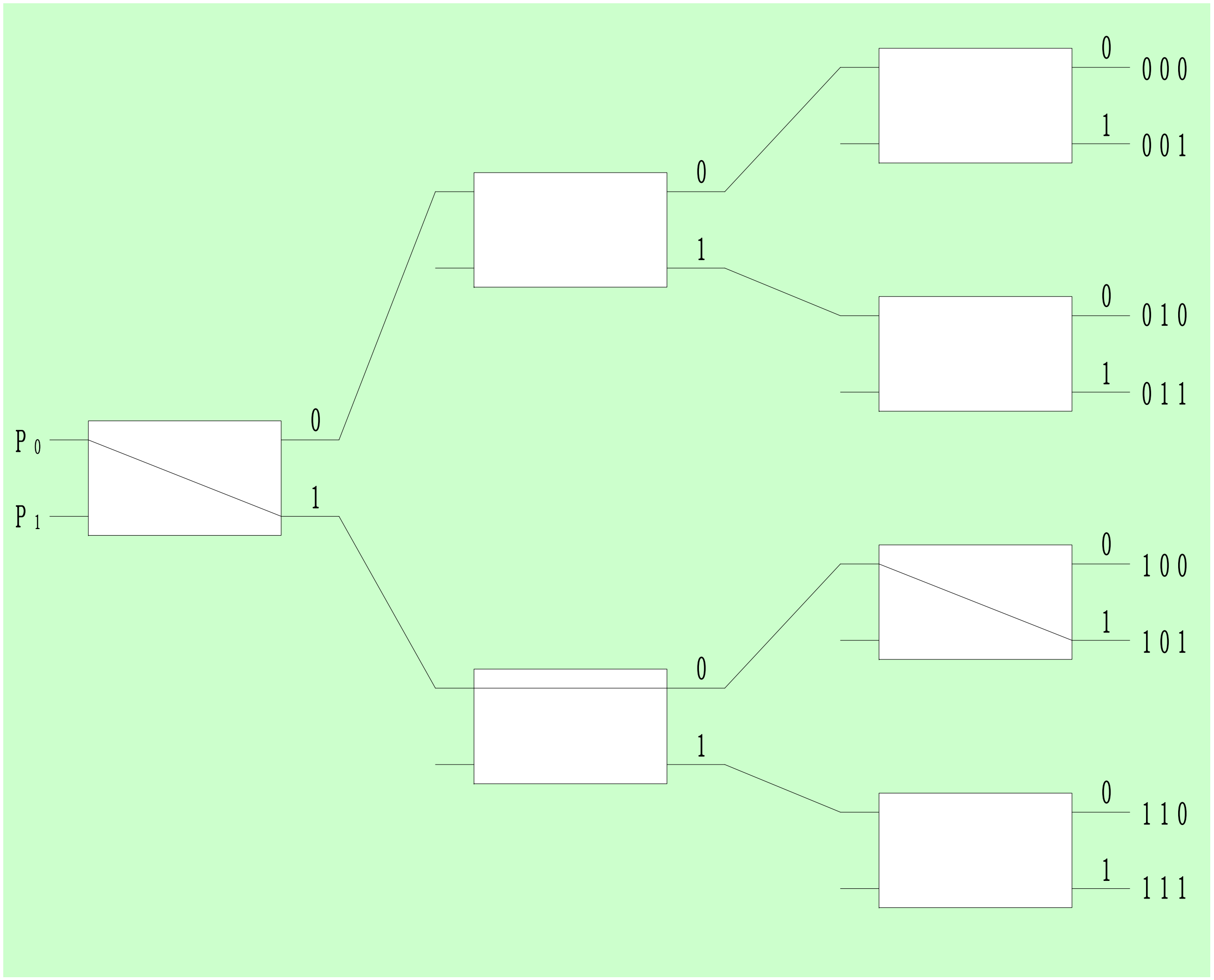
◆Multistage Switching Network

- Control the communication between a number of sources and destinations
  - » Tightly coupled system : **PU** $\longleftrightarrow$ **MM**
  - » Loosely coupled system : **PU** $\longleftrightarrow$ **PU**
- Basic components of a multistage switching network are

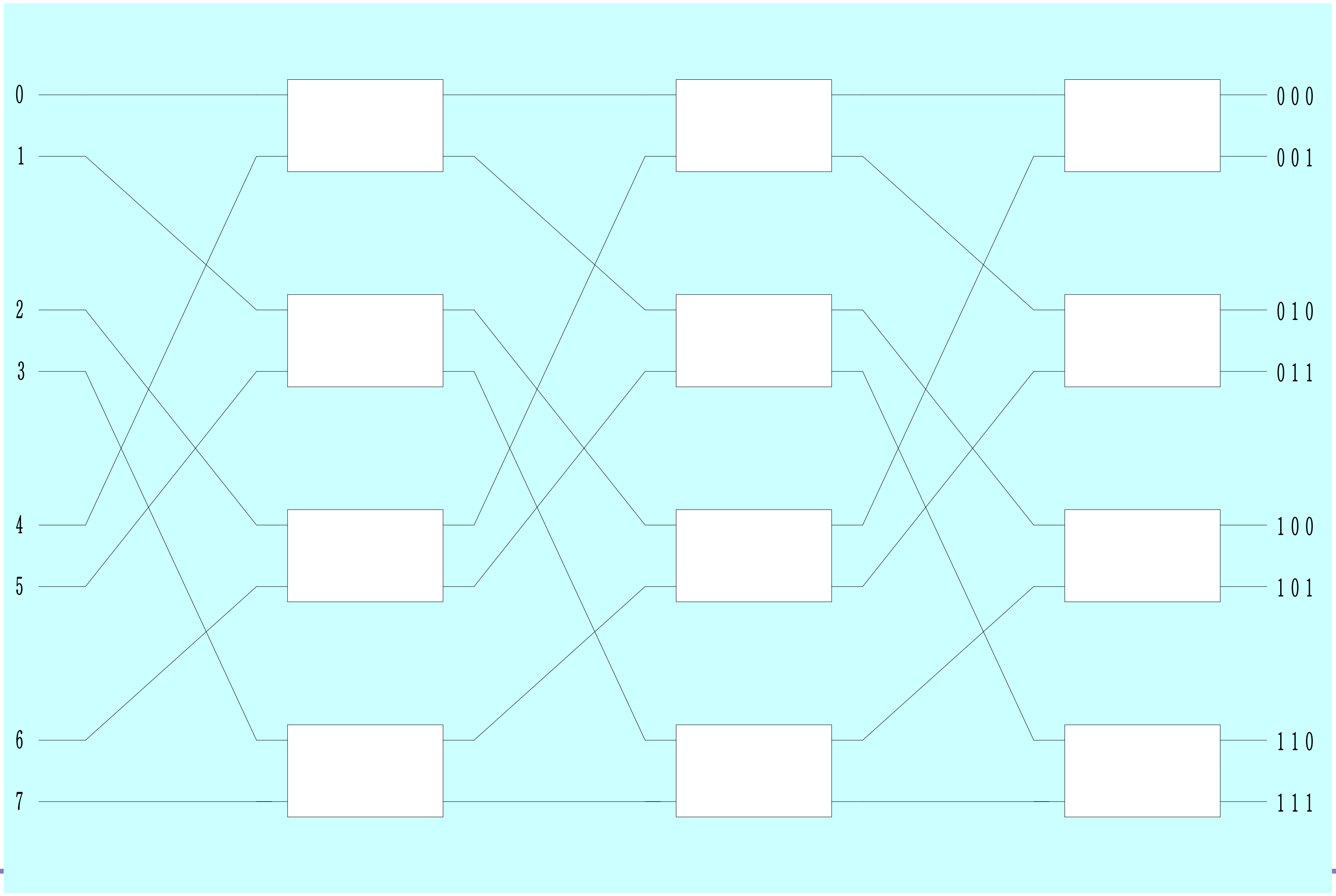
  two-input, two-output interchange switch : the two Input has labled A & B and two output labeled 0 & 1.



- It is possible to build a multistage network to control the communication between number of sources and destinations.
- The 2 Processor (P1 and P2) are connected through switches to 8 memory modules marked by (000 - 111)

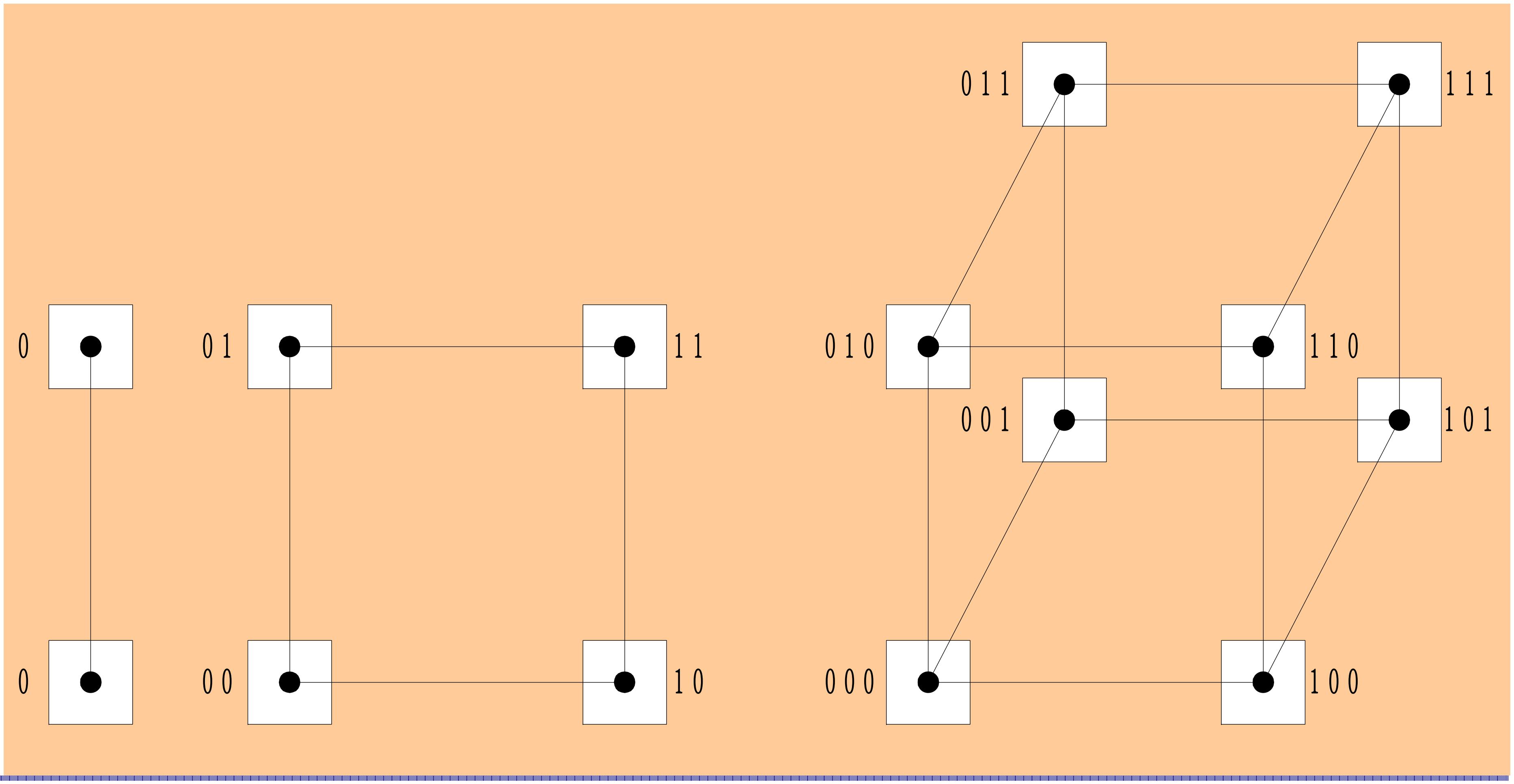


●Omega Network : 2 x 2 Interchange switch with N input x N output network topology



◆Hypercube Interconnection :

- The hypercube or binary n-cube multiprocessor structure is a Loosely coupled system composed of $N=2^n$ processors interconnected in an n-dimensional binary cube. Each processor forms a node for the cube.
- Hypercube Architecture : Intel iPSC (n = 7, 128 node)



■13-3 Interprocessor Arbitration : Bus Control

●**System bus** : Bus that connects CPUs, IOPs, and Memory in multiprocessor system

System Bus - A Backplane level bus

- Printed Circuit Board

- Connects CPU, IOP, and Memory

- Each of CPU, IOP, and Memory board can be plugged into a slot in the backplane(system bus)

- Bus signals are grouped into 3 groups

Data, Address, and Control(plus power)

- Only one of CPU, IOP, and Memory can be granted to use the bus at a time

e.g. IEEE standard 796 bus

- 86 lines

Data:	**16(multiple of 8)**
Address:	**24**
Control:	**26**
Power:	**20**

SYNCHRONOUS & ASYNCHRONOUS DATA TRANSFER

Synchronous Bus

Each data item is transferred over a time slice known to both source and destination unit

- Common clock source

- Or separate clock and synchronization signal is transmitted periodically to synchronize the clocks in the system

Asynchronous Bus

*** Each data item is transferred by *Handshake* mechanism**

- Unit that transmits the data transmits a control signal that indicates the presence of data

- Unit that receiving the data responds with another control signal to acknowledge the receipt of the data

*** Strobe pulse - supplied by one of the units to indicate to the other unit when the data transfer has to occur**

BUS SIGNALS

Bus signal allocation

- address
- data
- control
- arbitration
- interrupt
- timing
- power, ground

IEEE Standard 796 Multibus Signals

Data and address	
Data lines (16 lines)	DATA0 - DATA15
Address lines (24 lines)	ADRS0 - ADRS23
Data transfer	
Memory read	MRDC
Memory write	MWTC
IO read	IORC
IO write	IOWC
Transfer acknowledge	TACK (XACK)
Interrupt control	
Interrupt request	INT0 - INT7
interrupt acknowledge	INTA

IEEE Standard 796 Multibus Signals (Cont'd)

Miscellaneous control	
Master clock	CCLK
System initialization	INIT
Byte high enable	BHEN
Memory inhibit (2 lines)	INH1 - INH2
Bus lock	LOCK
Bus arbitration	
Bus request	BREQ
Common bus request	CBRQ
Bus busy	BUSY
Bus clock	BCLK
Bus priority in	BPRN
Bus priority out	BPRO
Power and ground (20 lines)	

Amdahl's law

Speedup *S(n)*: ratio of total execution time *T(1)* on a sequential computer to the corresponding execution time *T(n)* on the computer whose degree of parallelism is *n.*

$$S(n) = T(1)/T(n)$$

Efficiency *E(n)* : with which the speedup per degree of parallelism. *E(n)* is also an indication of processor utilization.

$$E(n) = S(n)/n$$

In General speedup and efficiency provides rough estimates of the performance changes that can be expected in a parallel processing system by increasing the parallelism degree *n,* I.e. by adding more processors

AMDAHL’S LAW

This theorem was first described by Gene Amdahl in his 1967 paper titled “***Validity of the single processor approach to achieving large scale computing capability”.*** The statement was

If F is the fraction of a calculation that is sequential, and (1-F) is the fraction that can be parallelized , then the maximum speedup that can be achieved by using P processors is 1/(F+(1-F)/P)

Given a program

f **: Fraction of time that represents operations that must be performed serially**

Maximum Possible Speedup: *S*

$$S \leq \frac{1}{f + (1 - f)/p} \quad \text{, with p processors}$$

$$S < 1/f \qquad \text{, with unlimited number of processors}$$

- Ignores possibility of new algorithm, with much smaller *f*

- Ignores possibility that more of program is run from higher speed memory such as Registers, Cache, Main Memory

- Often problem is scaled with number of processors, and *f* is a function of size which may be decreasing (Serial code may take constant amount of time, independent of size)

Ex. Of Amdahl’s Law

If 90% of a calculation can be parallelized (I.e. is sequential) than the maximum speed-up which can be achieved on 5- processor is 1/(0.1+ (1-0.1)/5) or roughly 3.6 (I.e. the program can theoretically run 3.6 times faster on five processors than one.

APPLICATION

STAMP is a new benchmark suite designed for Transactional Memory research. It currently consists of eight benchmarks with plans for more.

bayes: Bayesian network learning
genome: gene sequencing
intruder: network intrusion detection
kmeans: K-means clustering
labyrinth: maze routing
ssca2: graph kernels
vacation: client/server travel reservation system
yada: Delaunay mesh refinement (Ruppert's algorithm)

Scope of Research in Multiprocessor

The objective of research on the MIDAS project is to demonstrate the viability of a multiprocessor approach to computing and to develop a general purpose, extensible architecture which can be used to address the growing computational requirements of the scientific community. To be successful in this endeavor, however, requires more than simply designing, or even constructing, new hardware structures. To achieve high performance on future systems will require software approaches which can exploit parallel architectures as fully as possible. A critical issue, therefore, is to understand the functional requirements of a large class of applications. The effective utilization of highly parallel architectures, however, raises many new questions. Traditional operating system structures must, for example, be re-examined. Fault-tolerant environments, with error recovery capability, become increasingly important as the number of components and processors increases. Language extensions to support parallel operations are obviously required, and ultimately new languages are needed to explicitly exploit parallel constructs. A new generation of development and debugging tools will be necessary in order to examine programn performance and operation in an asynchronous parallel environment.