# Computer Organization and Software Systems

## Session 7

**BITS** Pilani
Pilani Campus

Pruthvi Kumar K R

# Control Unit Design

**BITS** Pilani
Pilani Campus

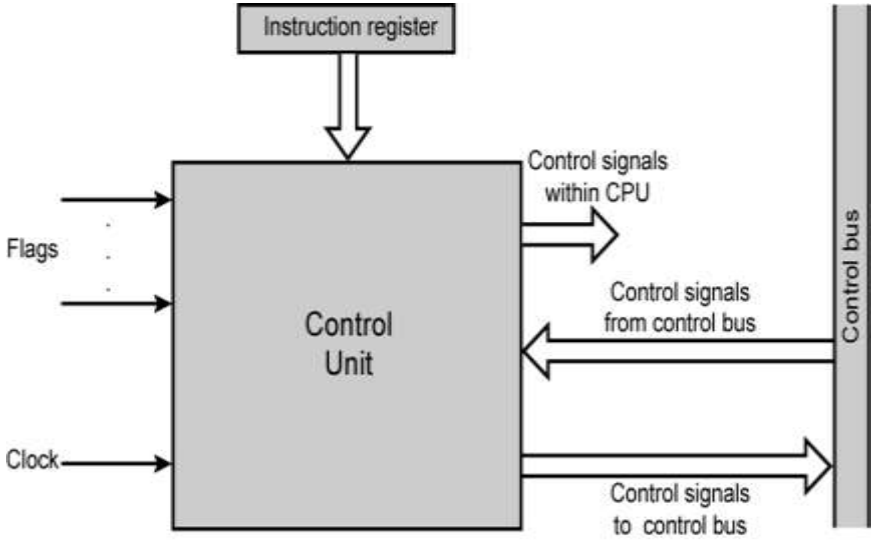
# Control Unit implementation

Hardwired control unit (RISC)

Microprogrammed control unit(CISC)

# Hardwired Implementation (1)

- Control unit inputs
  - Flags and control bus
    - Each bit means something
  - Instruction register
    - Op-code causes different control signals for each different instruction
    - Unique logic for each op-code
  - Clock
- Time efficient



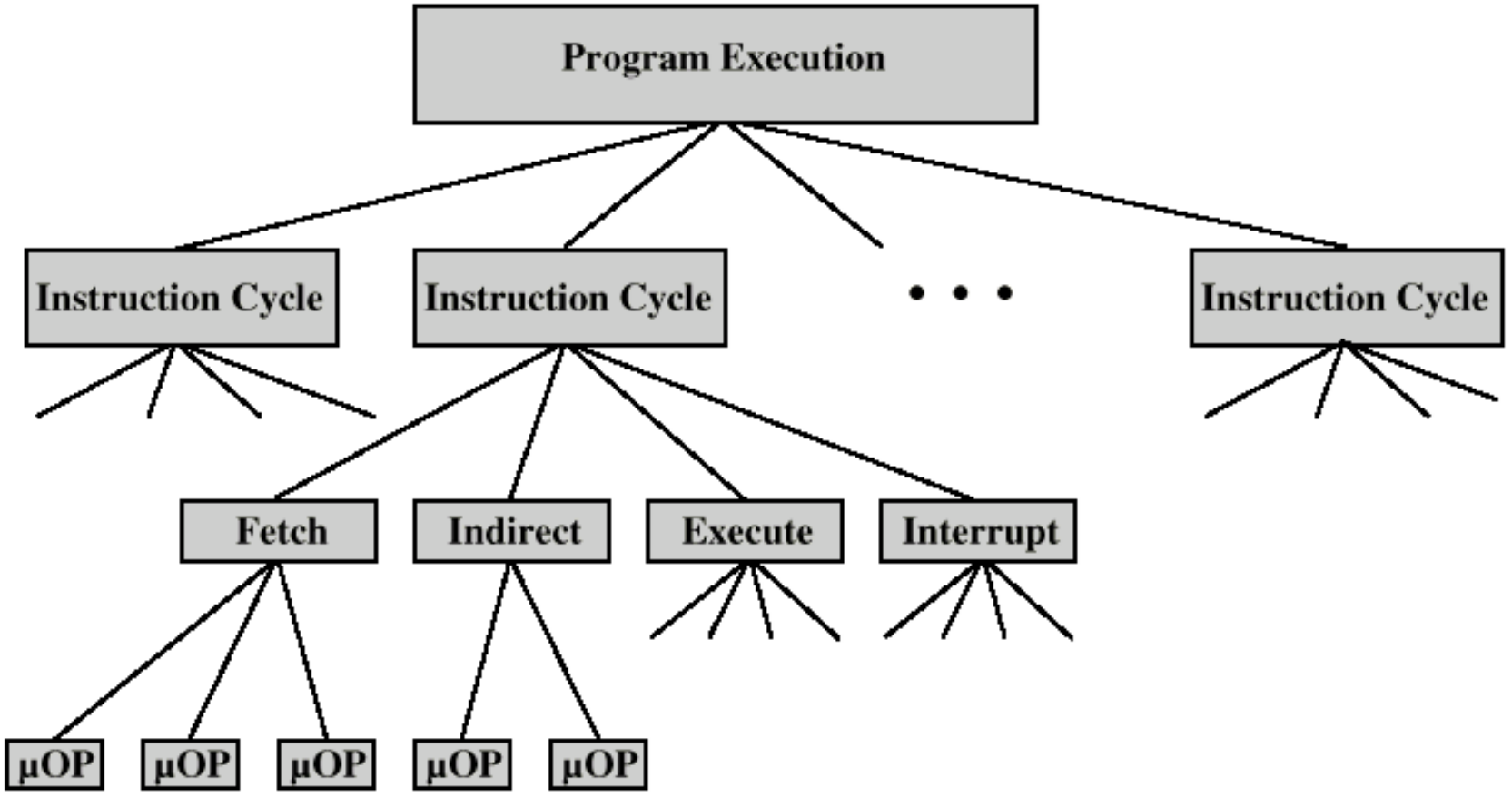
# Problems With Hard Wired Designs

- Complex sequencing & micro-operation logic
- Difficult to design and test
- Inflexible design
- Difficult to add new instructions

# Microprogrammed Control Unit

- A computer executes a program
- Fetch/execute cycle
- Each cycle has a number of steps
  - Called micro-operations
  - Each step does very little
  - Atomic operation of CPU

# Constituent Elements of Program Execution



# Example: Fetch Sequence

t1: MAR □ (PC)
t2: MBR □ (memory)
PC □ (PC) +1
t3: IR □ (MBR)

**OR**

t1: MAR <- (PC)
t2: MBR <- (memory)
t3: PC <- (PC) +1
IR <- (MBR)

# Example: Execute Cycle (ADD)

Different for each instruction

e.g. ADD R1,X - add the contents of location X to Register 1 , result in R1

t1: MAR ← ($IR_{address}$)

t2: MBR ← (memory)

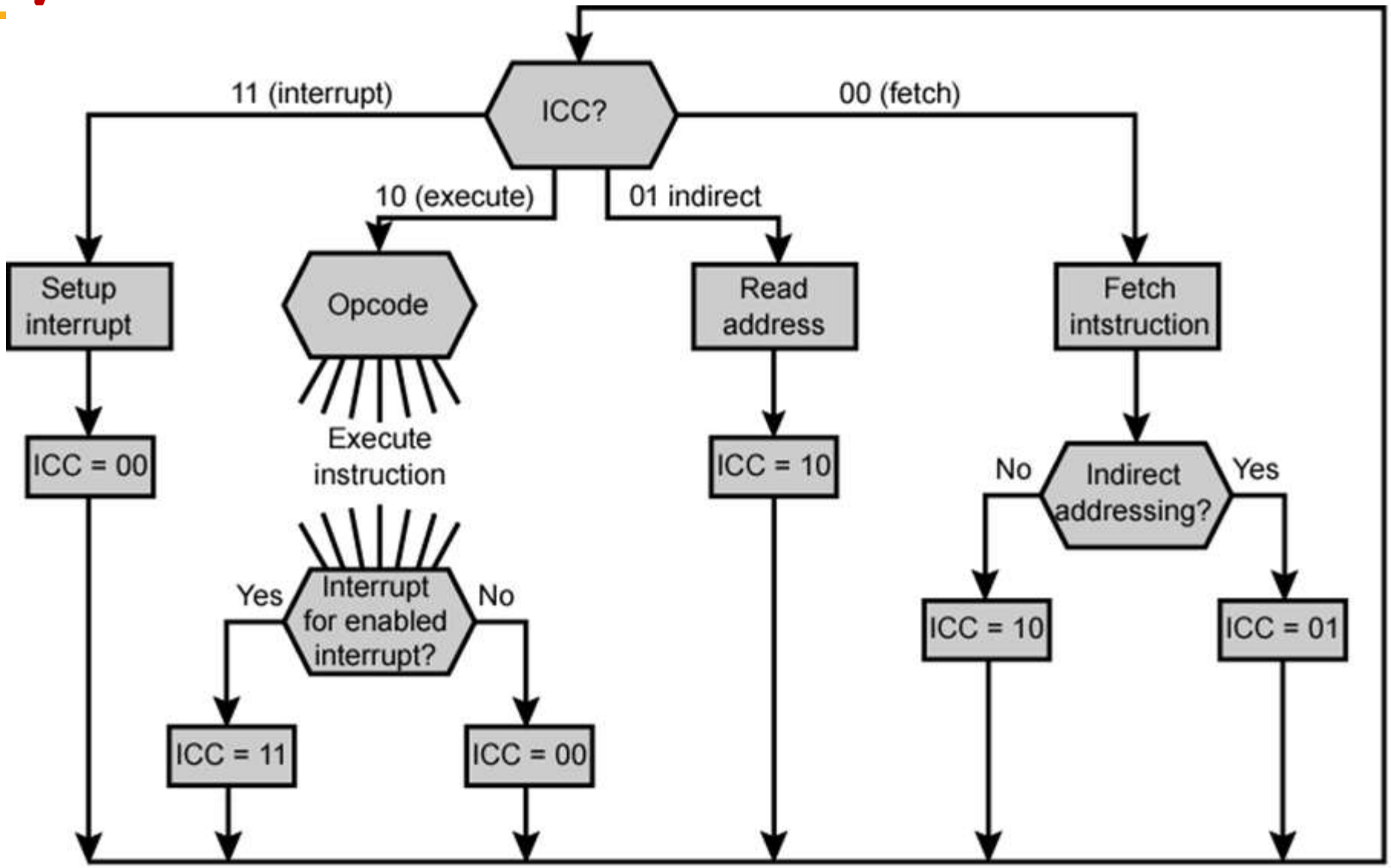
t3: R1 ← R1 + (MBR)

Note no overlap of micro-operations

# Instruction Cycle

- Each phase decomposed into sequence of elementary micro-operations
- E.g. fetch, indirect, and interrupt cycles
- Execute cycle
  - One sequence of micro-operations for each opcode
- Need to tie sequences together
- Assume new 2-bit register
  - Instruction cycle code (ICC) designates which part of cycle processor is in
    - 00: Fetch
    - 01: Indirect
    - 10: Execute
    - 11: Interrupt

# Flowchart for Instruction Cycle



# Functions of Control Unit

- The control unit performs two basic tasks:
  - Sequencing
    - Causing the CPU to step through a series of micro-operations
  - Execution
    - Causing the performance of each micro-op
- This is done using Control Signals

Example: ADD R1, X

t1: MAR □ (PC)
t2: MBR □ (memory)
PC □ (PC) +1
t3: IR □ (MBR)

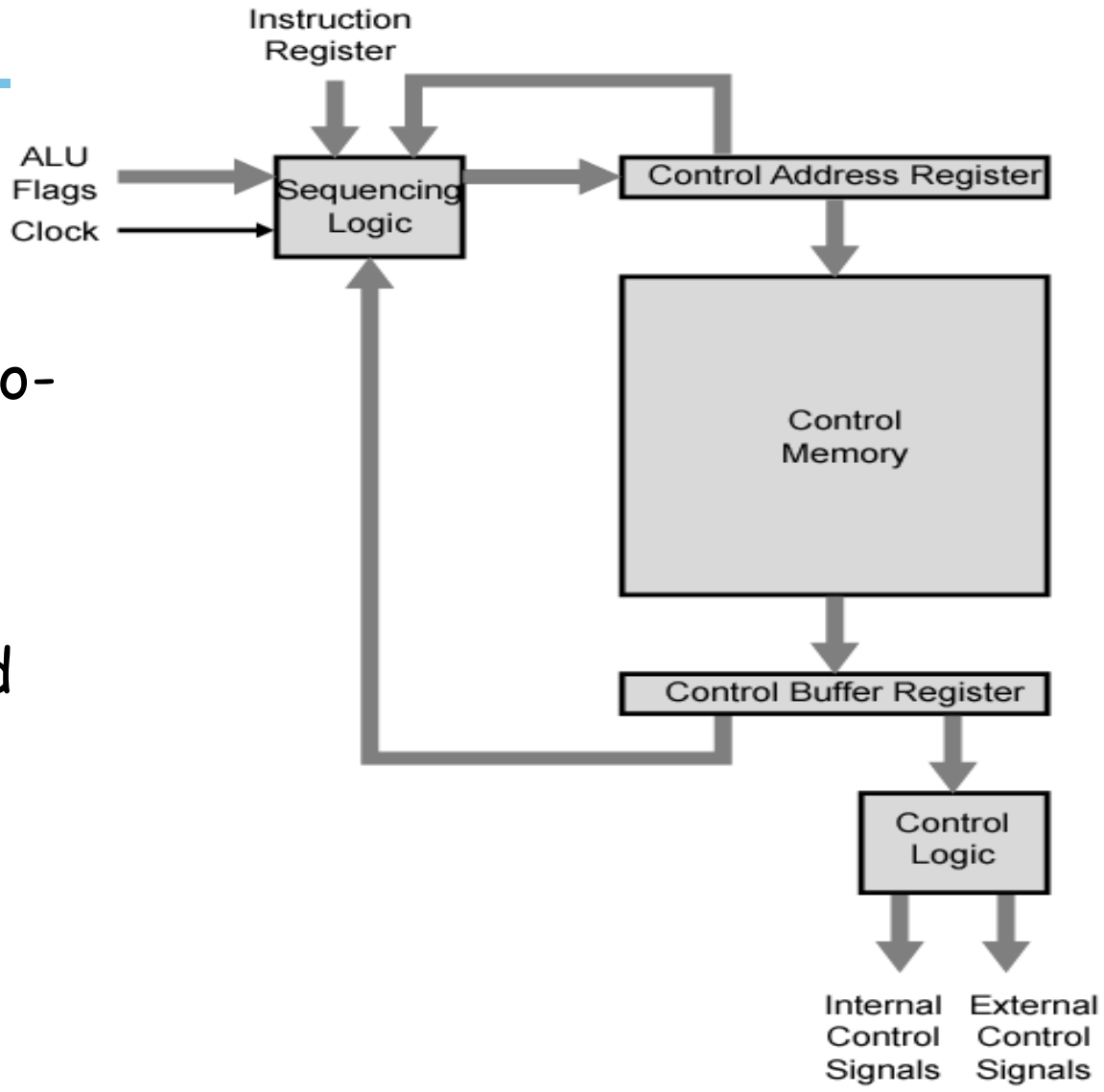
t4: MAR □ ($IR_{address}$)
t5: MBR □ (memory)
t6: R1 □ R1 + (MBR)

# Control Signals

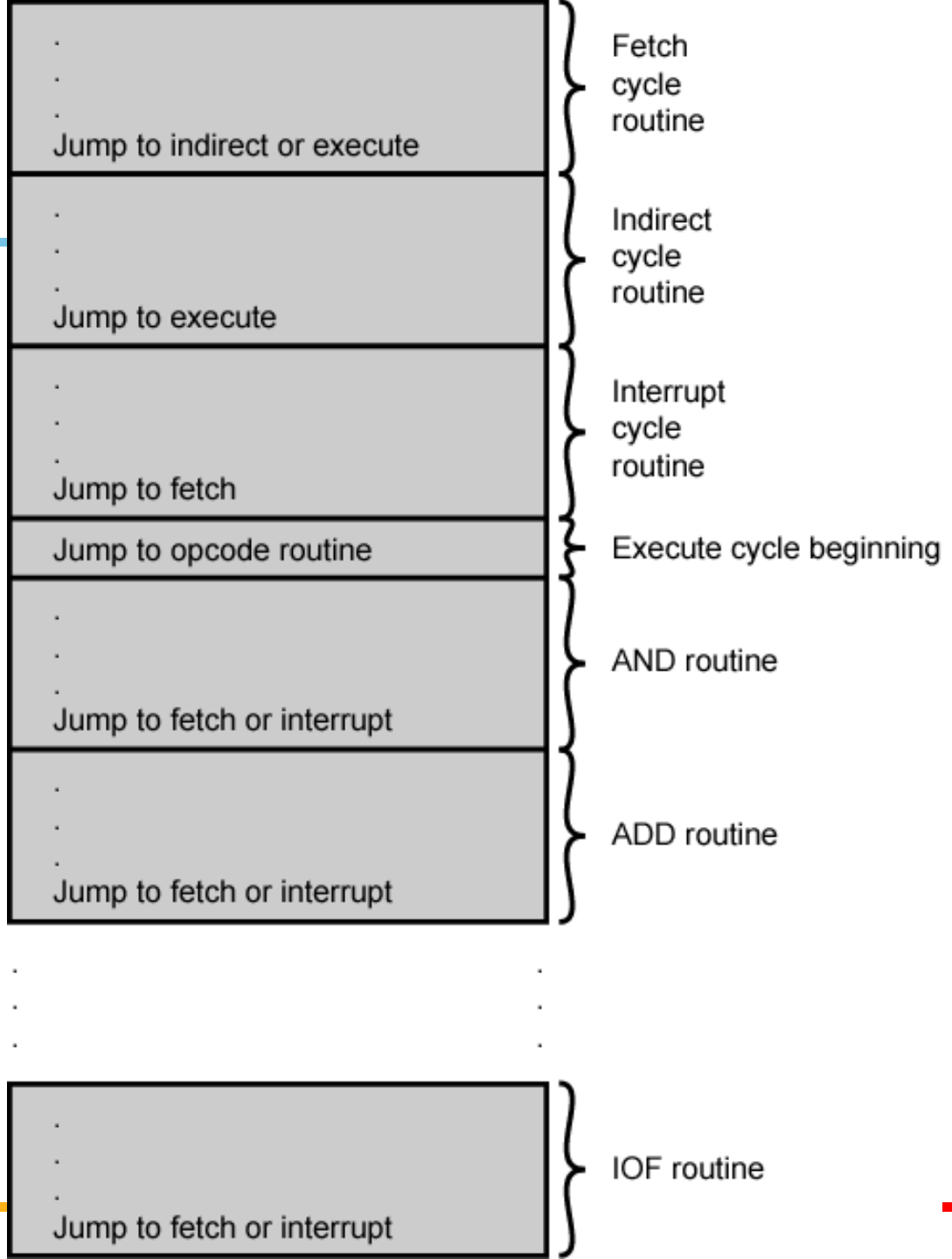
- Inputs to the control unit
  - Clock
    - One micro-instruction (or set of parallel micro-instructions) per clock cycle
  - Instruction register
    - Op-code for current instruction
    - Determines which micro-instructions are performed
  - Flags
    - State of CPU
    - Results of previous operations
  - From control bus
    - Interrupts
    - Acknowledgements

# Microprogrammed Control Unit

- Use sequences of micro-instructions to control complex operations called micro-programming or firmware
- Firmware is midway between hardware and software



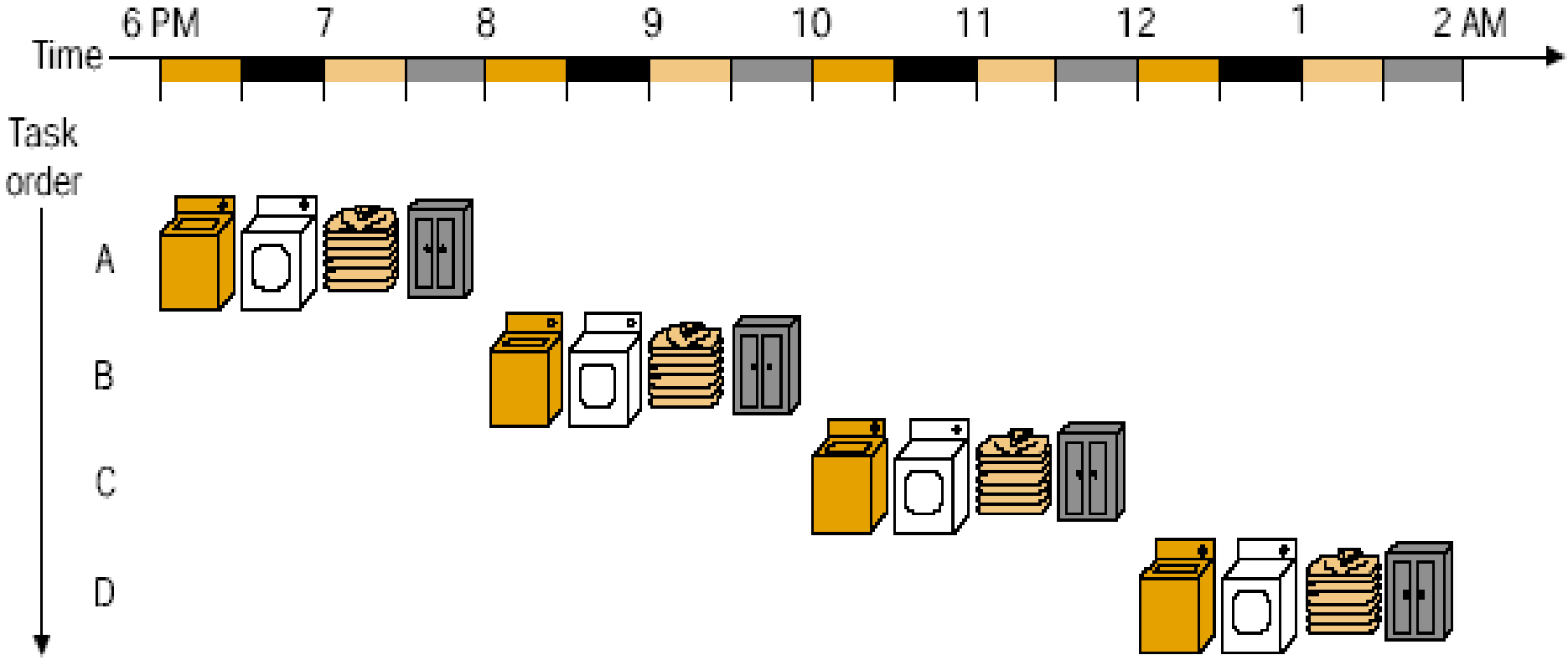
# Organization of Control Memory



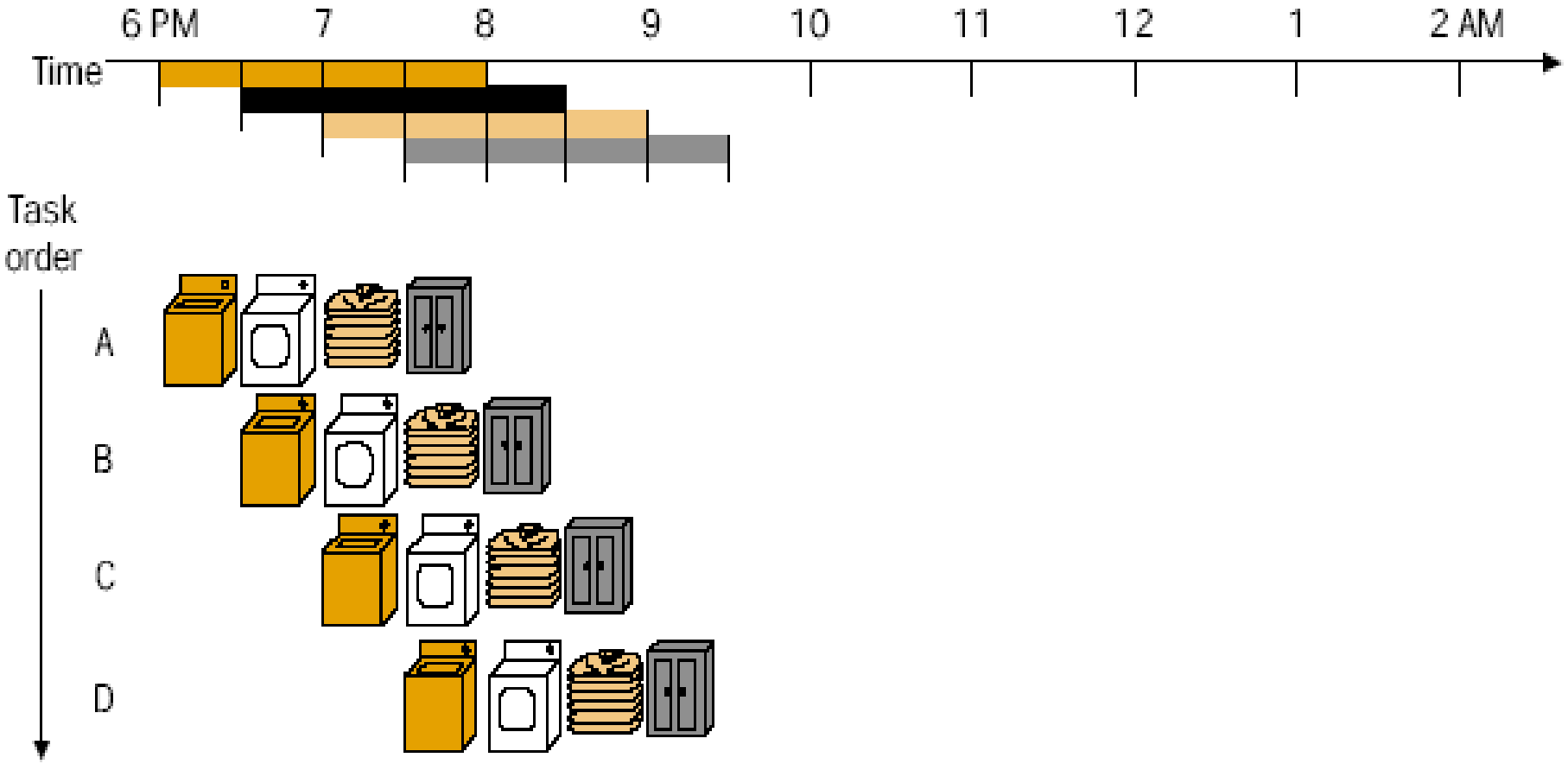
# Pipeline

**BITS** Pilani
Pilani Campus

# Laundry System



# Laundry System……



# Pipelining

- The process of accepting new inputs at one end before previously accepted inputs appear as outputs at the other end.

- To apply this concept to instruction execution, we must recognize that, in fact, an instruction has a number of stages

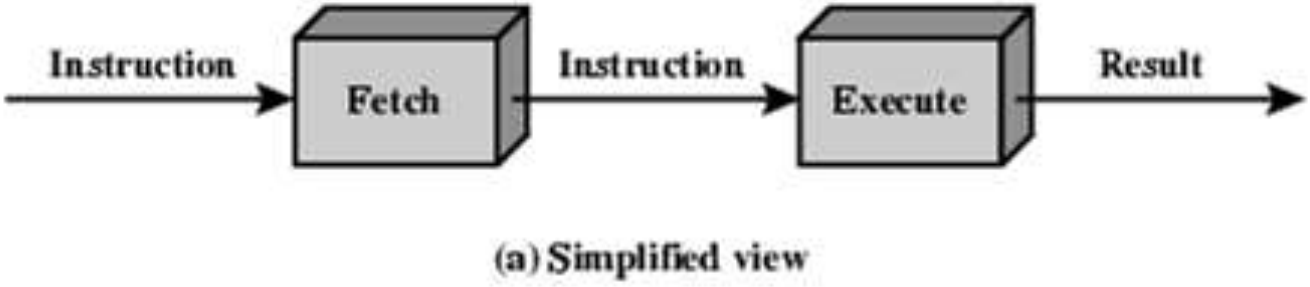
# Pipelining

- An overlapped parallelism: overlapped execution of multiple operations
- Pipelining
  - Subdivide the input task into a sequence of subtasks
  - Specialized hardware stage for each task
  - Concurrent operation of all the stages

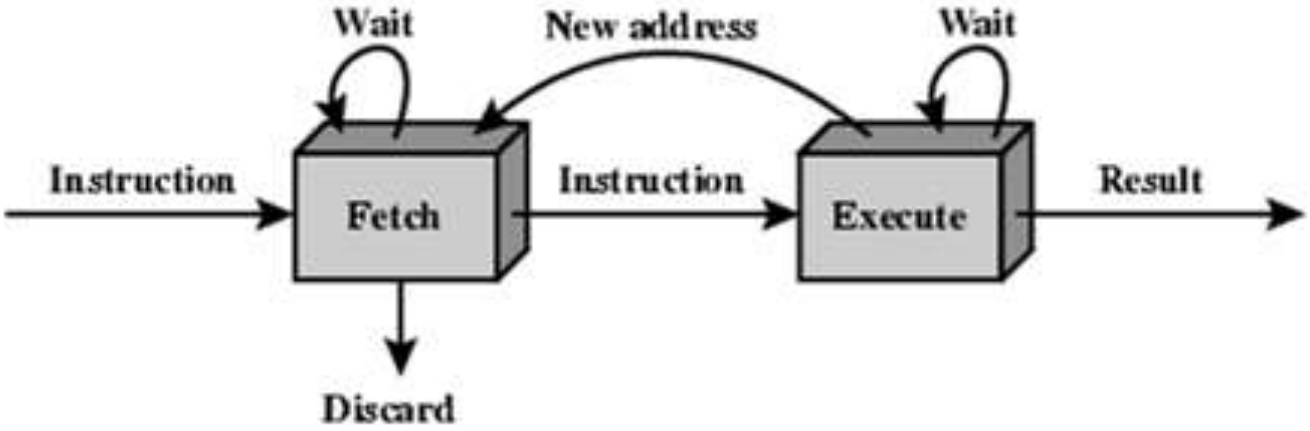
# Two segment instruction pipeline

- Contains
  - Instruction Fetch (IF)
  - Execute (EX)
- Example: 8086 microprocessor
- Instruction Fetch unit is implemented by means of first in first out buffer (Queue)

# Two Stage Pipeline




(a) Simplified view




(b) Expanded view

# Issues

1. The execution time will generally be longer than the fetch time
2. A conditional branch instruction makes the address of the next instruction to be fetched unknown.

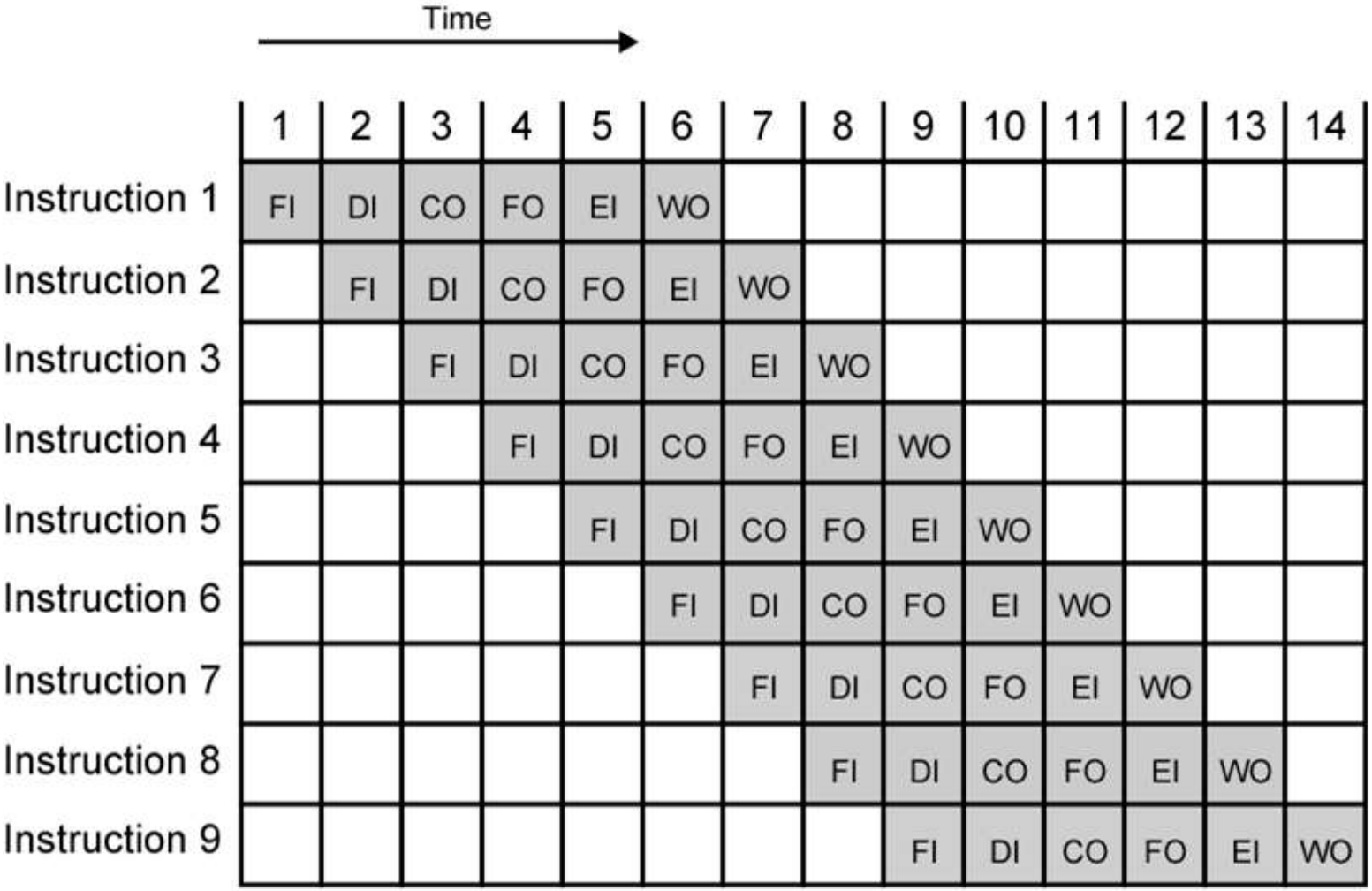
# Four Segment instruction pipeline

- Contains
  - FI : fetch instruction
  - DA : Decode instruction and calculate effective address
  - FO :Fetch operand
  - EX: Execute instruction

# Six stage pipeline

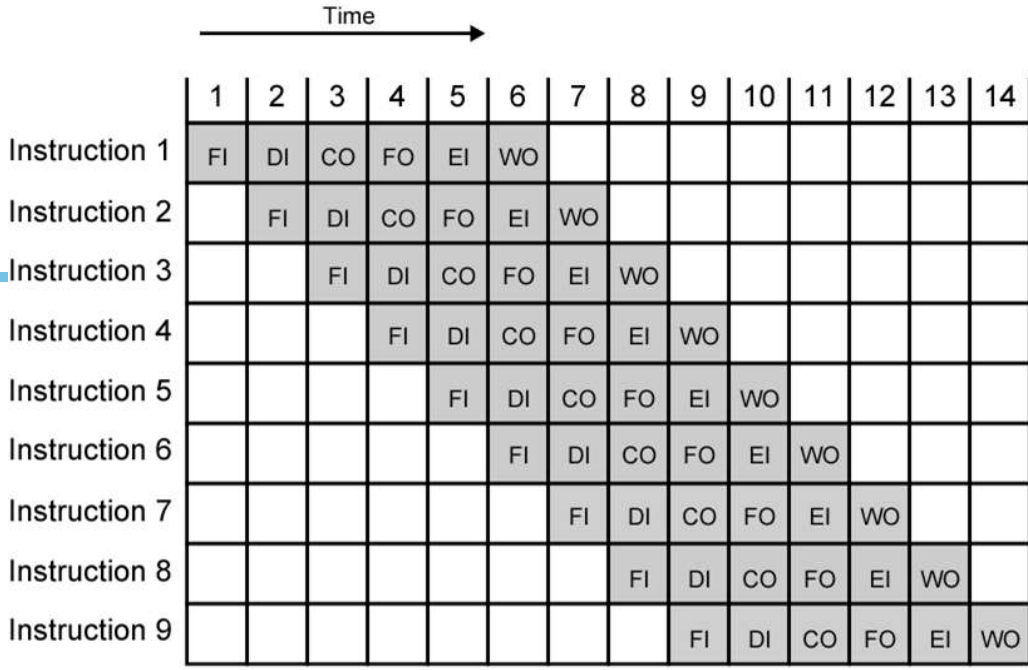
- Contains
  - FI : fetch instruction
  - DI : Decode instruction
  - CO : calculate effective address
  - FO :Fetch operand
  - EI : Execute instruction
  - WO : Write Operand

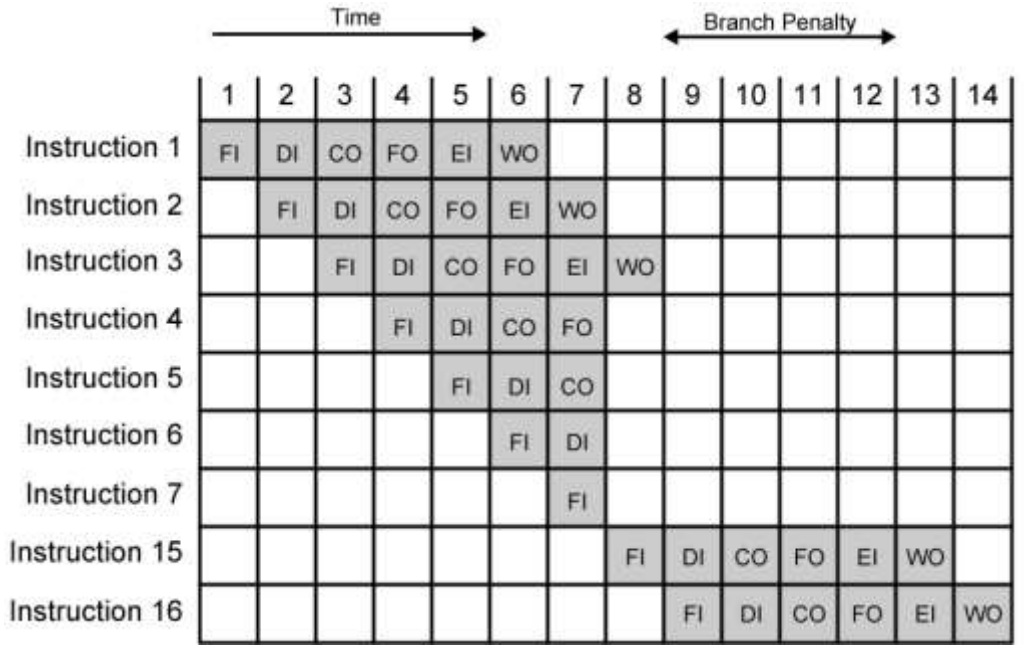
# Timing Diagram for Instruction Pipeline Operation



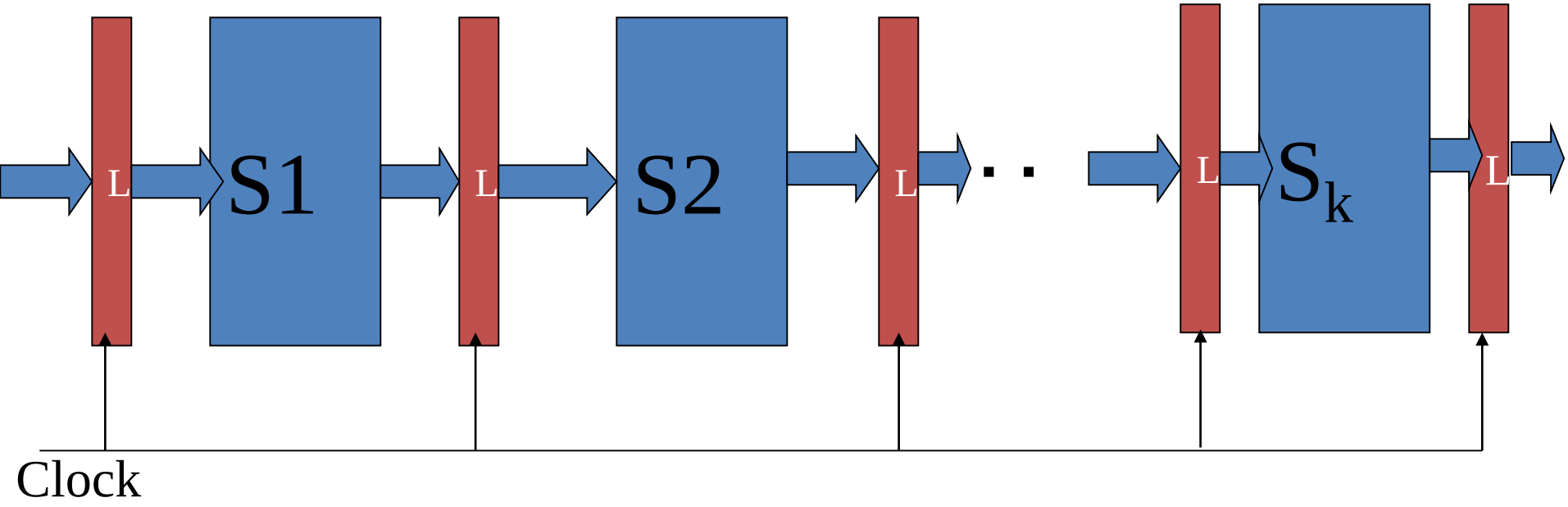
# Important Points to be noted

- Do all the instructions need all the stages?
- At t=6, WO, FO and FI accesses the memory. Is there any issue?
- Is there any implication on having different time duration for different stages?
- Any issues with conditional branch?
- Dependency of CO stage on register used in previous stage



# Structure of a pipeline



# Classification

- Arithmetic pipelining
- Instruction pipelining
- Processor pipelining
- Unifunction and multifunction pipelining
- Static and Dynamic pipelining
- Scalar and Vector pipelining

# Arithmetic pipelining

- Arithmetic and logic units of a computer can be segmentized for pipeline operations
- Usually found in high speed computers
- Example:
  - Star 100 – 4 stage
  - TI-ASC – 8 stage
  - Cray-1 – 14 stage
  - Cyber 205 – 26 stages
  - Intel Cooper Lake (3rd Gen Intel Xeon) = 14 stages
- Floating point adder pipeline

$$X = A*2^a$$

$$Y = B*2^b$$

# Instruction pipelining

- The execution of a stream of instructions can be pipelined by overlapping the execution of the current instruction with the fetch, decode……of subsequent instructions
- Sequence of steps followed in most general purpose computer to process instruction
  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

# Unifunction and multifunction pipelining

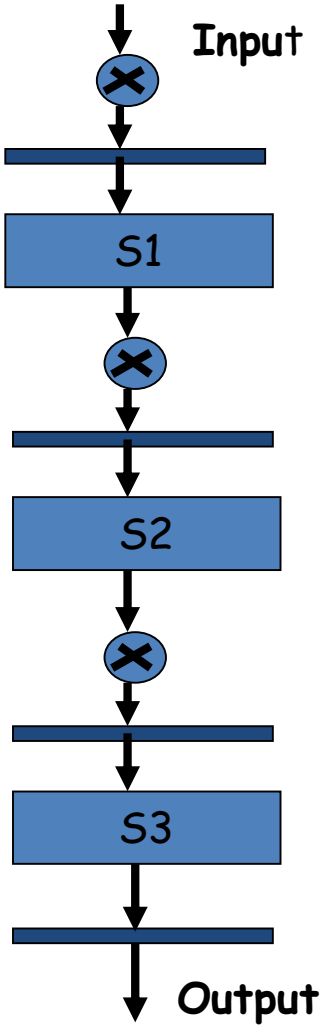
- Uni-function

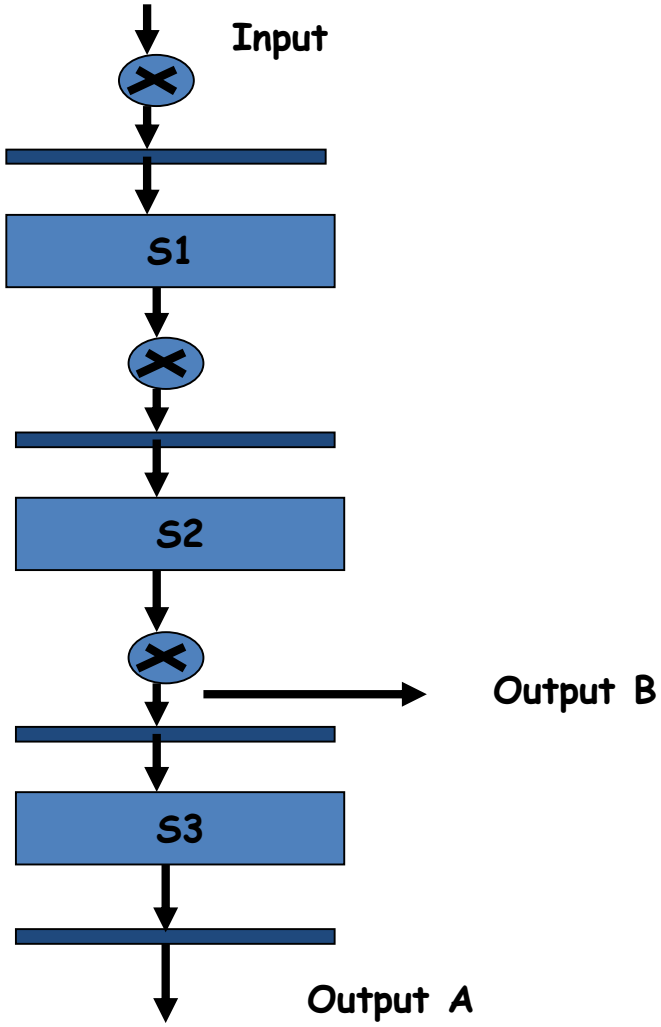
  Pipeline with a fixed and dedicated function

  – Ex: Floating point adder
- Multifunction

  – Pipeline may perform different functions

# Uni-function Vs Multifunction



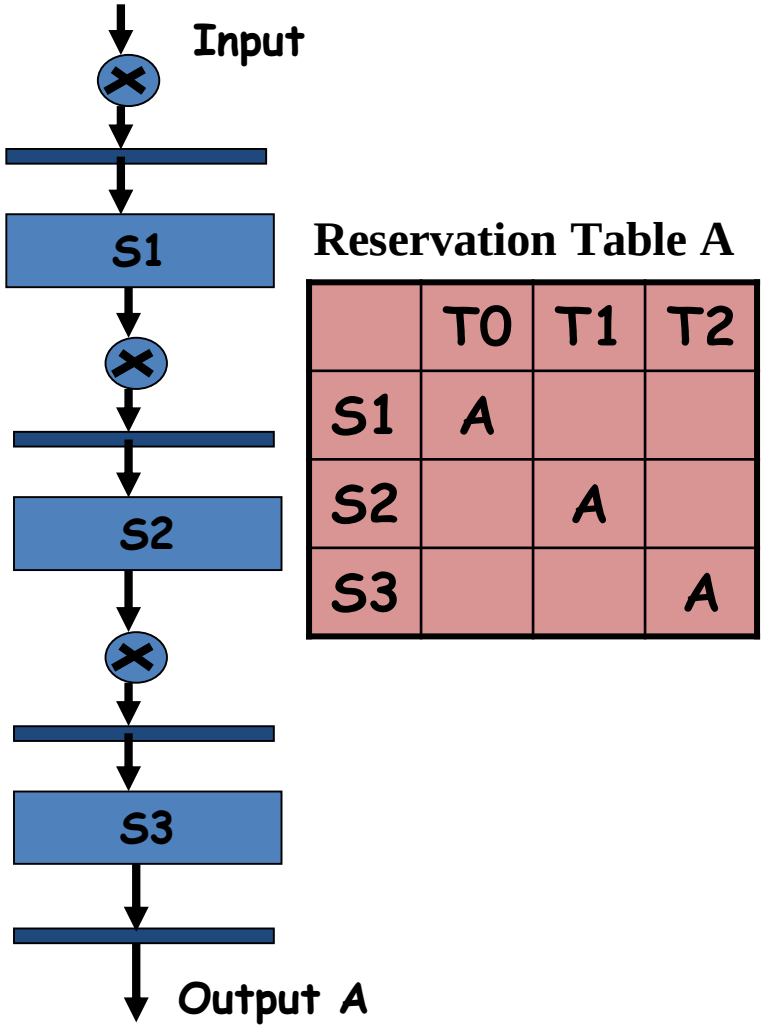



# Reservation Table

Is a two dimensional chart

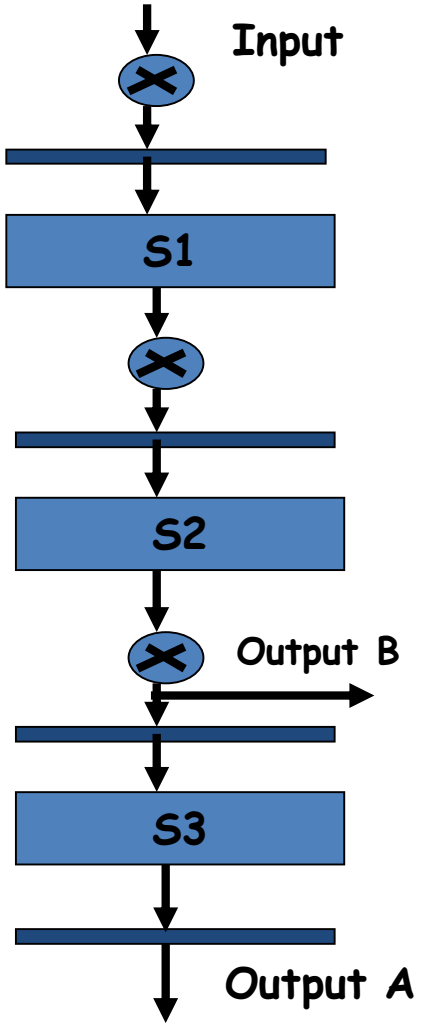
Used to show how successive pipeline stages are utilized or reserved

# Uni-function Vs Multifunction




Reservation Table A

| | T0 | T1 | T2 |
|---|---|---|---|
| S1 | A | | |
| S2 | | A | |
| S3 | | | A |




Reservation Table A

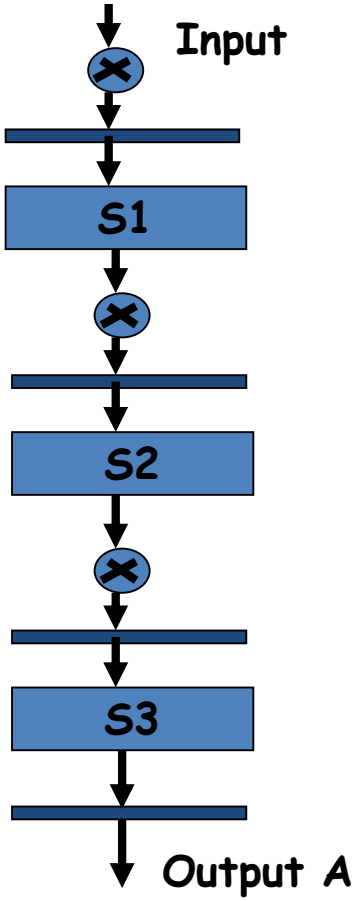
| | T0 | T1 | T2 |
|---|---|---|---|
| S1 | A | | |
| S2 | | A | |
| S3 | | | A |

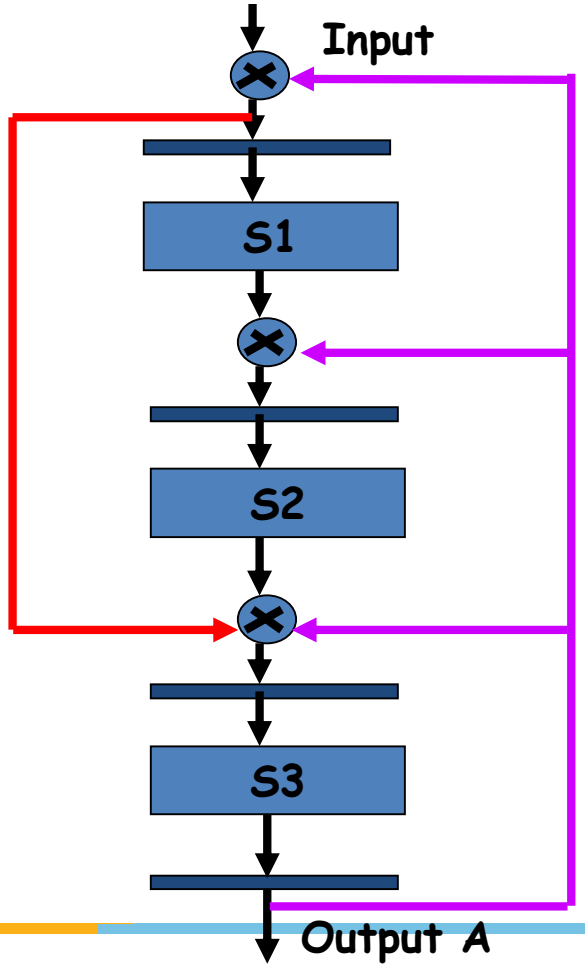
Reservation Table B

| | T0 | T1 |
|---|---|---|
| S1 | B | |
| S2 | | B |

# Linear and Nonlinear Pipelines

- Linear Pipeline: Without feed forward and feed back connection
- Nonlinear Pipeline with feed forward and/or feed back connection

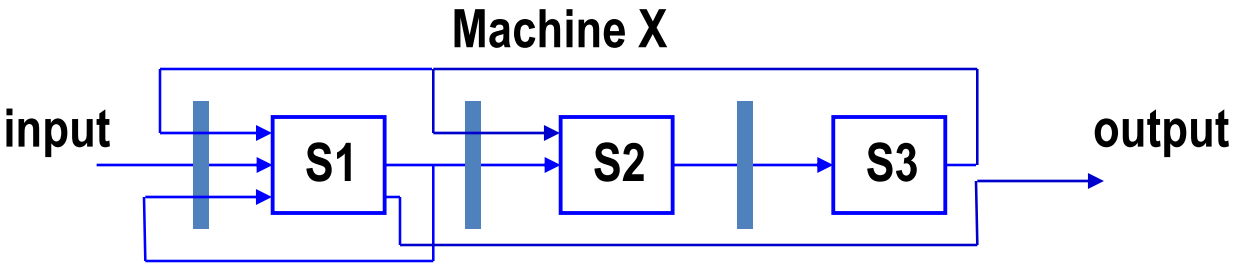


# Static and Dynamic pipelining

- Based on the configuration i.e. the interconnection pattern between its stages
- A static pipeline assumes only one functional configuration at a time
- Useful when instructions of the same type can be streamed for execution

# Reservation Table



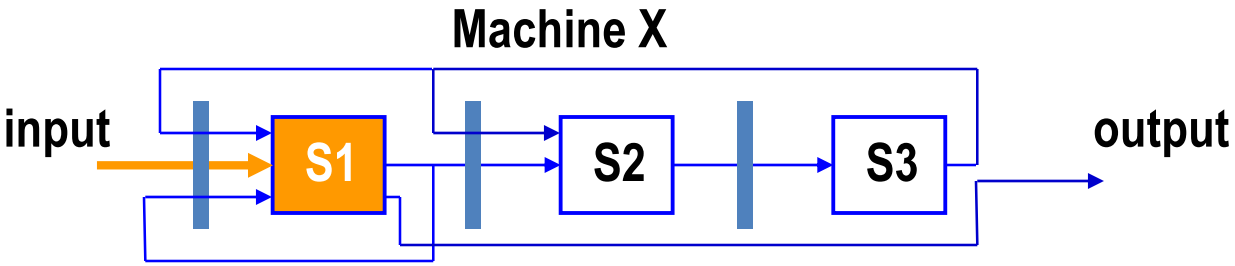

## Reservation Table

Time →

Stage

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| S1 | X | X | | | | | X | X |
| S2 | | | X | | X | | | |
| S3 | | | | X | | X | | |

# Reservation Table



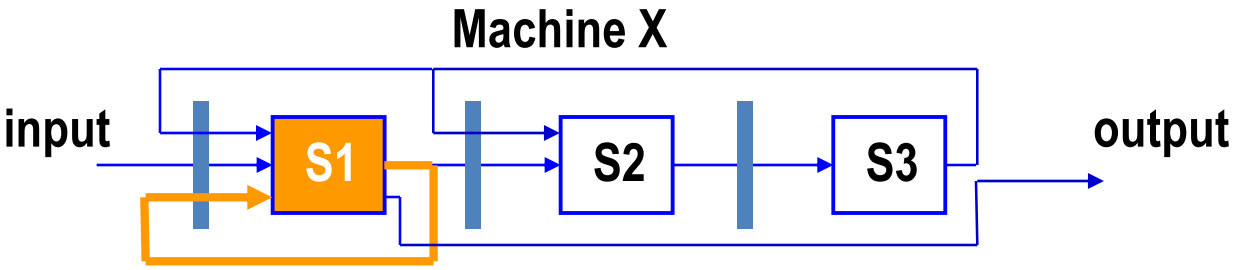

## Reservation Table

Time →

Stage

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| S1 | X | X | | | | | X | X |
| S2 | | | X | | X | | | |
| S3 | | | | X | | X | | |

# Reservation Table



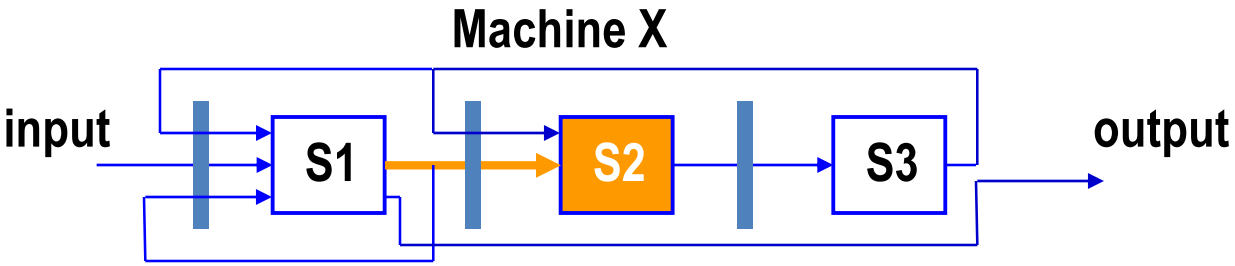

## Reservation Table

Time →

Stage

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| S1 | X | X | | | | | X | X |
| S2 | | | X | | X | | | |
| S3 | | | | X | | X | | |

# Reservation Table



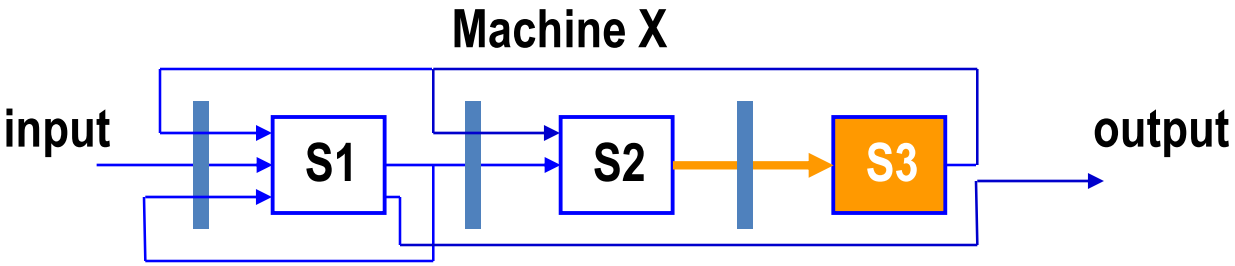

## Reservation Table

Time →

Stage

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| S1 | X | X | | | | | X | X |
| S2 | | | X | | X | | | |
| S3 | | | | X | | X | | |

# Reservation Table



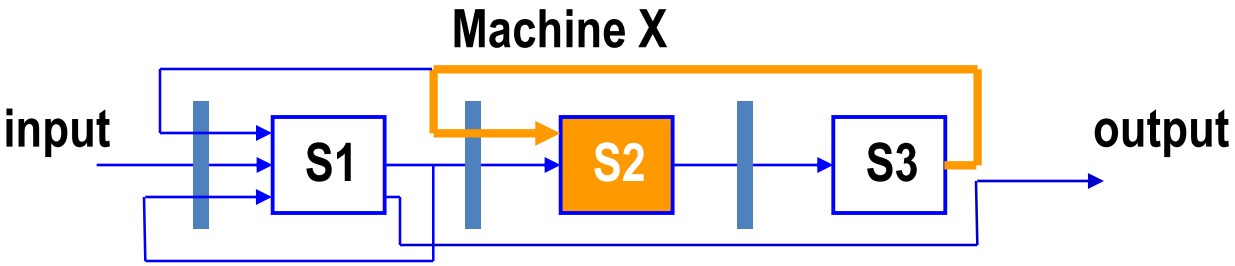

## Reservation Table

Time →

Stage

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| S1 | X | X | | | | | X | X |
| S2 | | | X | | X | | | |
| S3 | | | | X | | X | | |

# Reservation Table



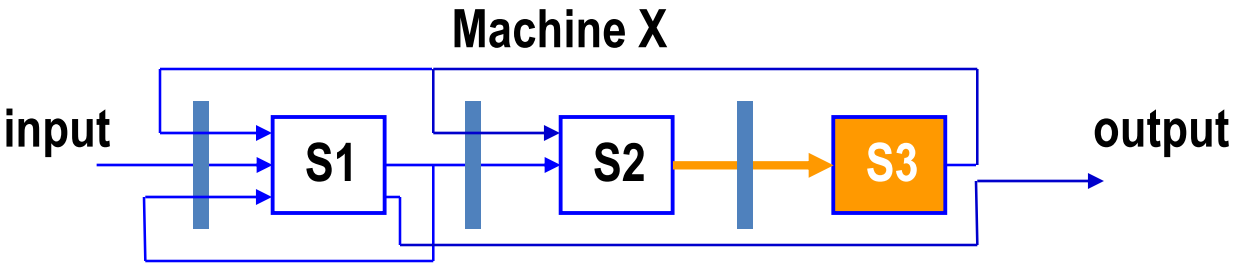

## Reservation Table

Time →

Stage

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| S1 | X | X | | | | | X | X |
| S2 | | | X | | X | | | |
| S3 | | | | X | | X | | |

# Reservation Table



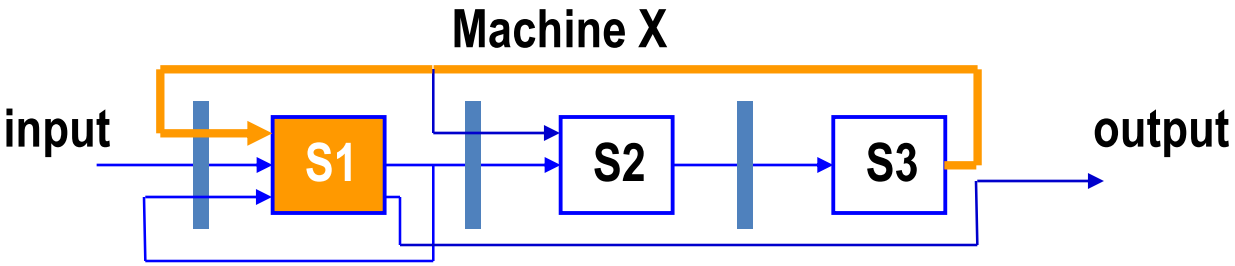

## Reservation Table

Time →

Stage

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| S1 | X | X | | | | | X | X |
| S2 | | | X | | X | | | |
| S3 | | | | X | | X | | |

# Reservation Table



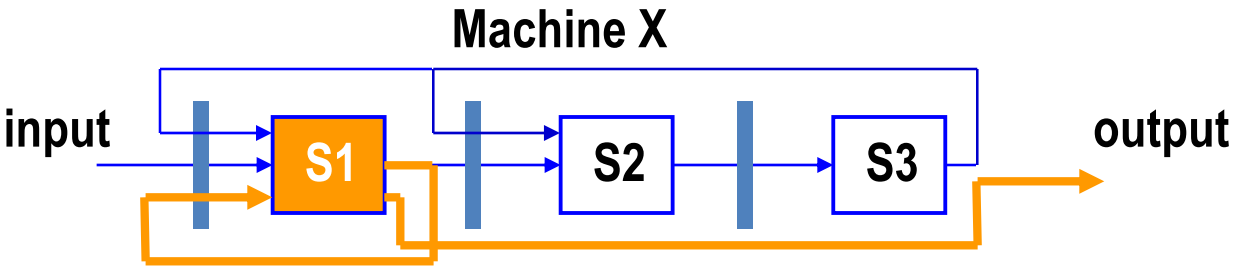

## Reservation Table

Time →

Stage

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| S1 | X | X | | | | | X | X |
| S2 | | | X | | X | | | |
| S3 | | | | X | | X | | |

# Reservation Table




## Reservation Table

Time →

Stage

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| S1 | X | X | | | | | X | X |
| S2 | | | X | | X | | | |
| S3 | | | | X | | X | | |

# Static and Dynamic pipelining

- Dynamic pipeline allows more frequent changes in its configuration
- Require more elaborate sequencing and control mechanisms

# Scalar and Vector pipelining

- Based on the operand types or instruction type
- Scalar pipeline processes scalar operands
- Vector pipeline operate on vector data and instructions.