# Computer Organization and Software Systems

## Session 8

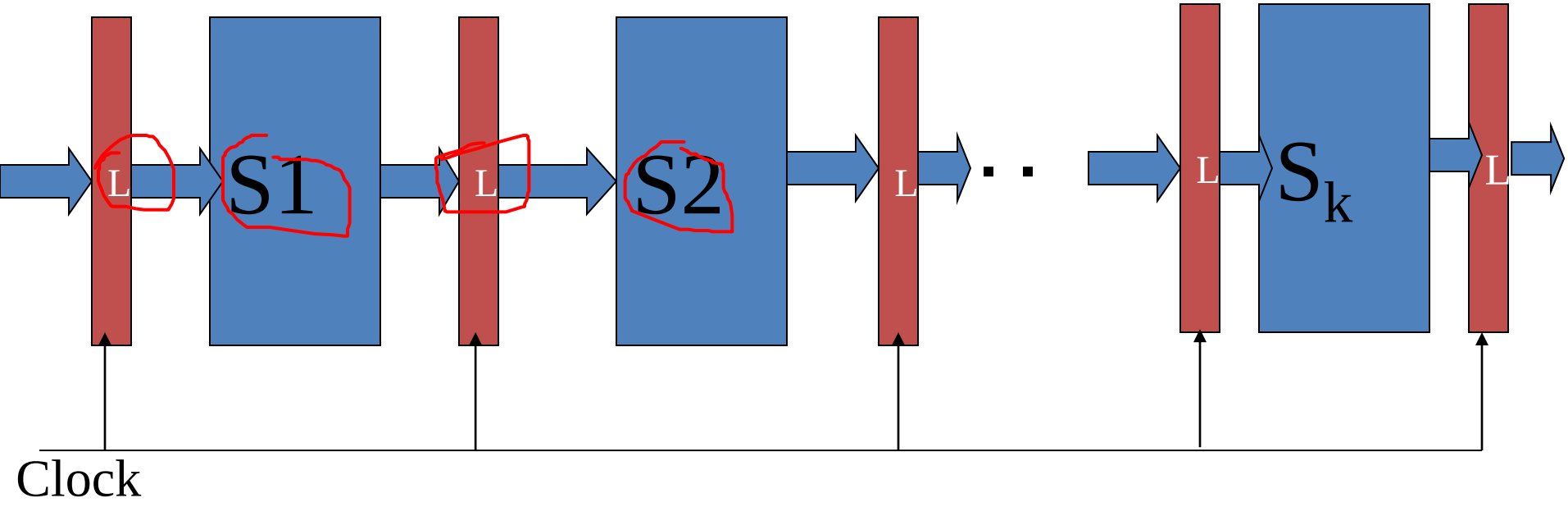
Dr. Lucy J. Gudino
WILP & Department of CS & IS

BITS Pilani
Pilani Campus

# Pipeline Continued

BITS Pilani
Pilani Campus

# 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

innovate achieve lead

- 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

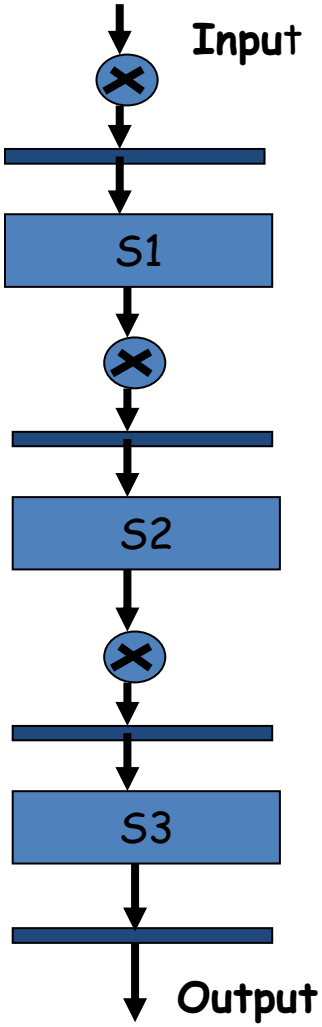
- Uni-function

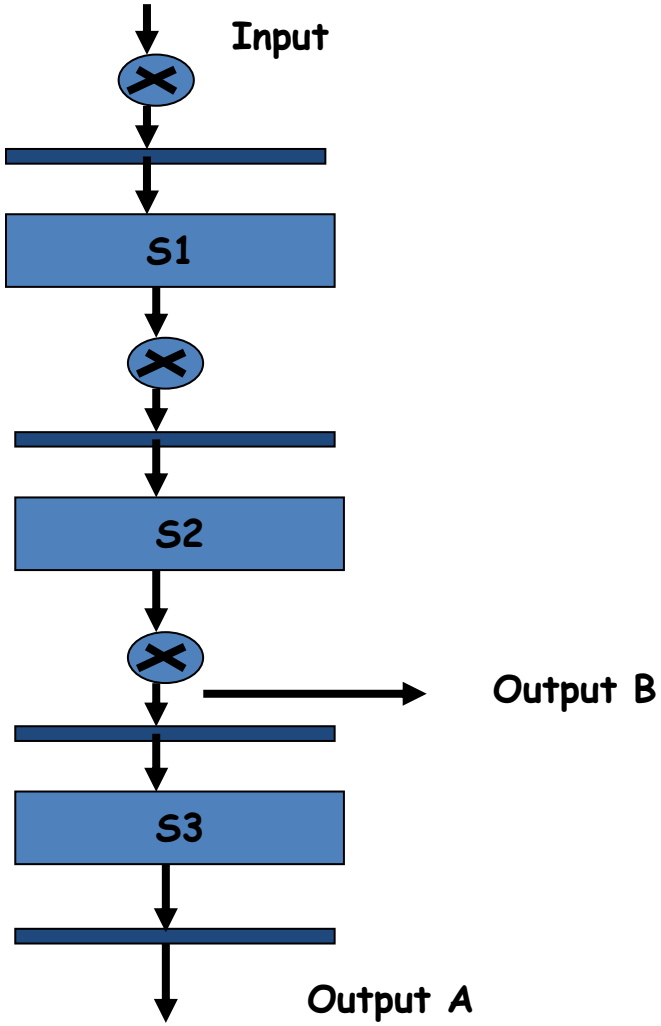
  Pipeline with a fixed and dedicated function

  – Ex: Floating point adder
- Multifunction

  – Pipeline may perform different functions

# Uni-function Vs Multifunction



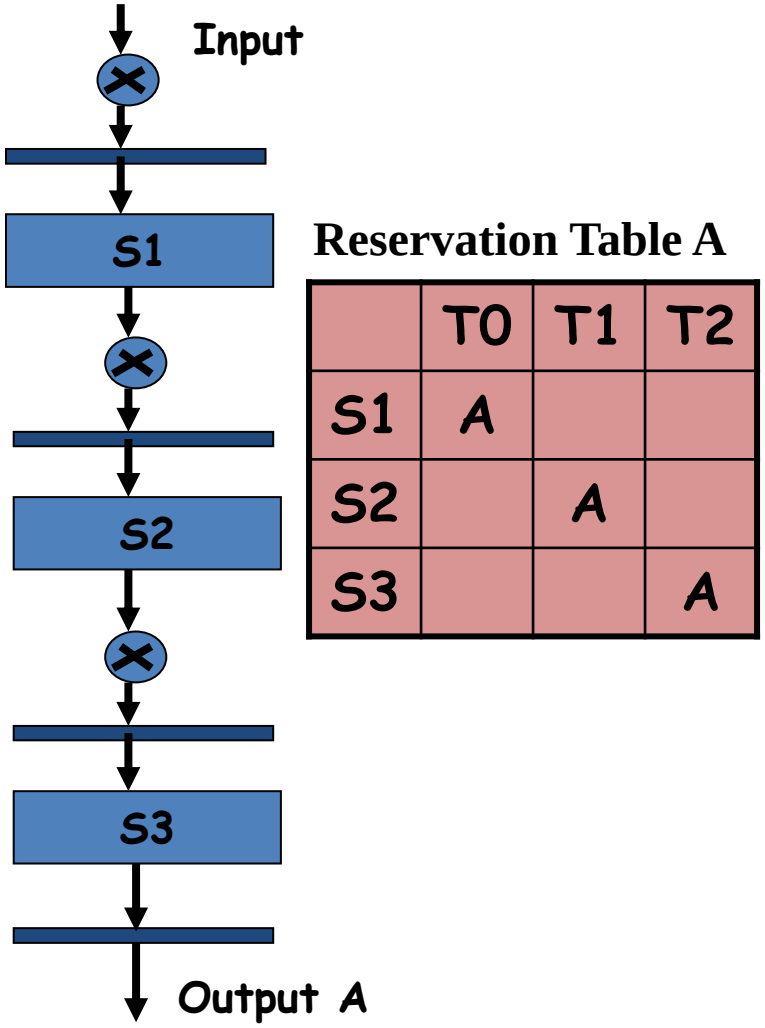
# Reservation Table

innovate achieve lead

Is a two dimensional chart

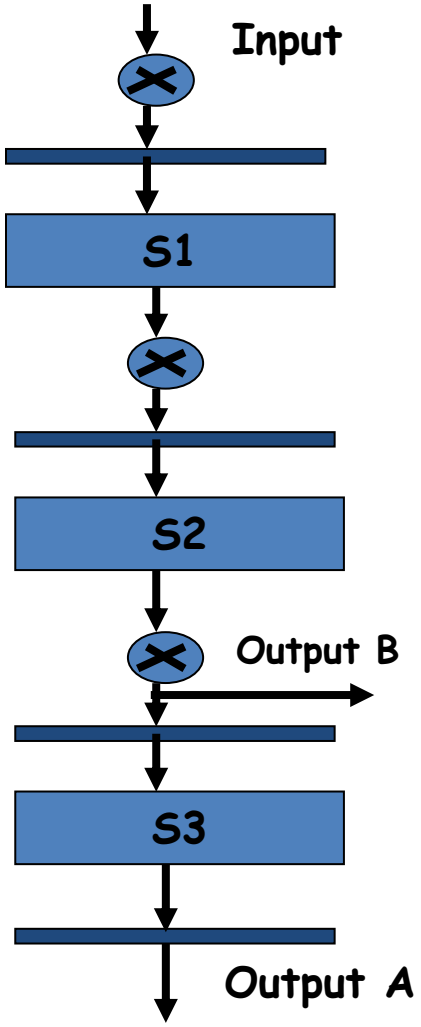
Used to show how successive pipeline stages are utilized or reserved

# Uni-function Vs Multifunction




Reservation Table A

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




Reservation Table A

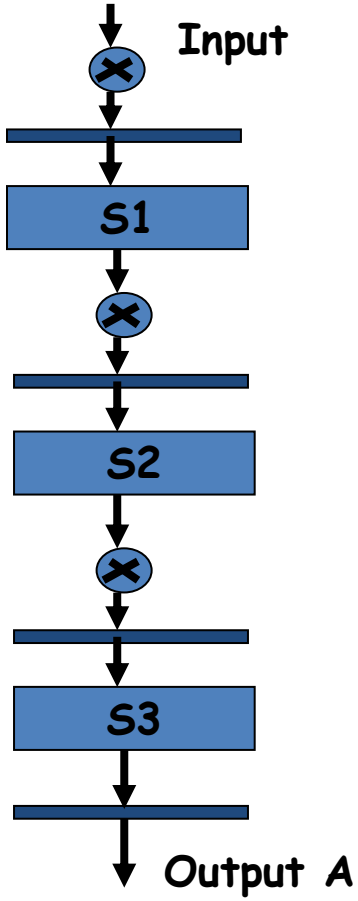
| | T0 | T1 | T2 |
|---|---|---|---|
| S1 | A | | |
| S2 | | A | |
| S3 | | | A |

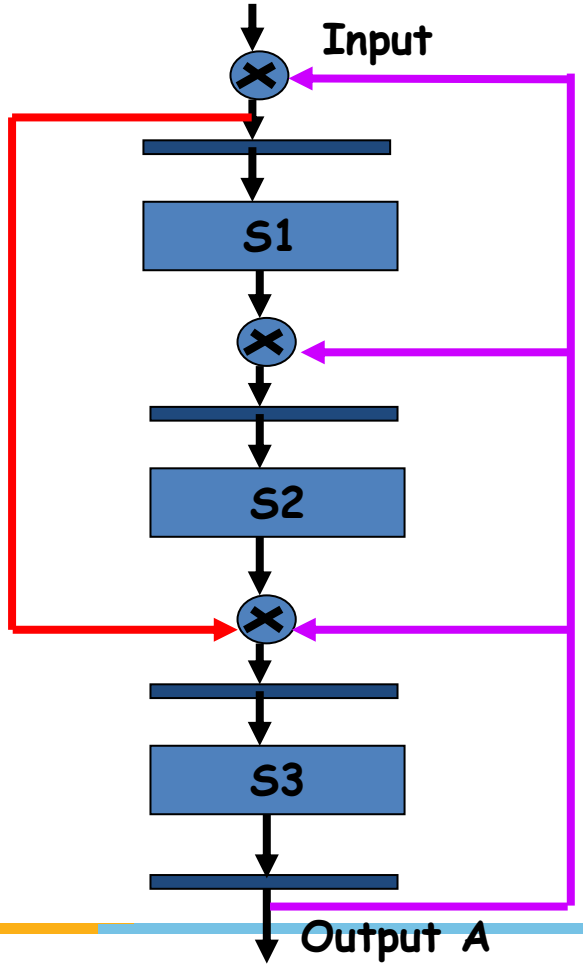
Reservation Table B

| | 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

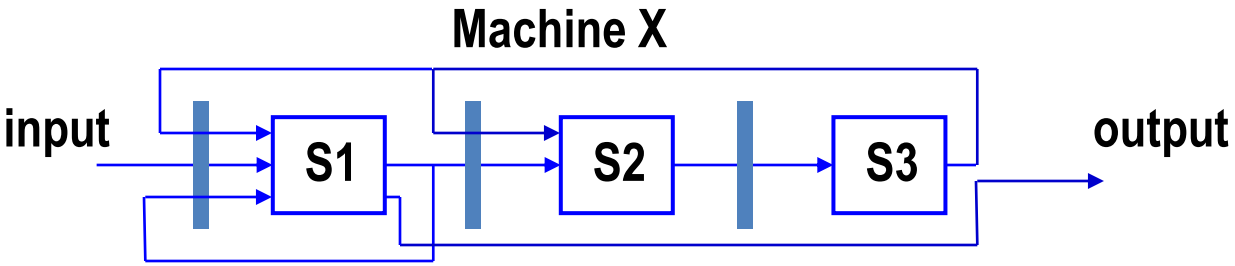


# Static and Dynamic pipelining

- 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



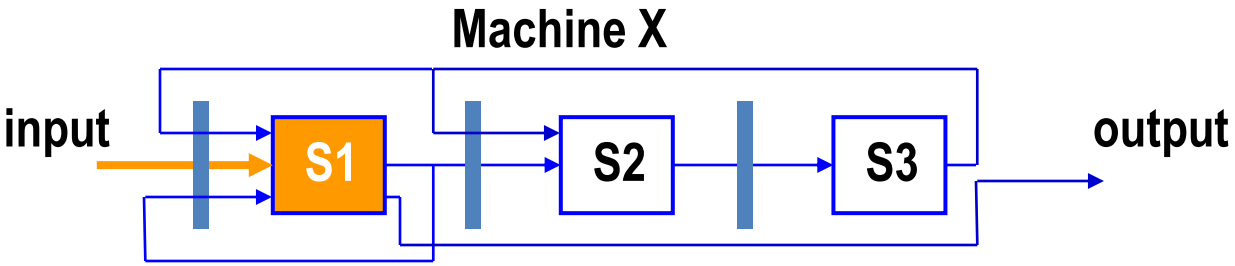

## Reservation Table

Time →

Stage

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

# Reservation Table



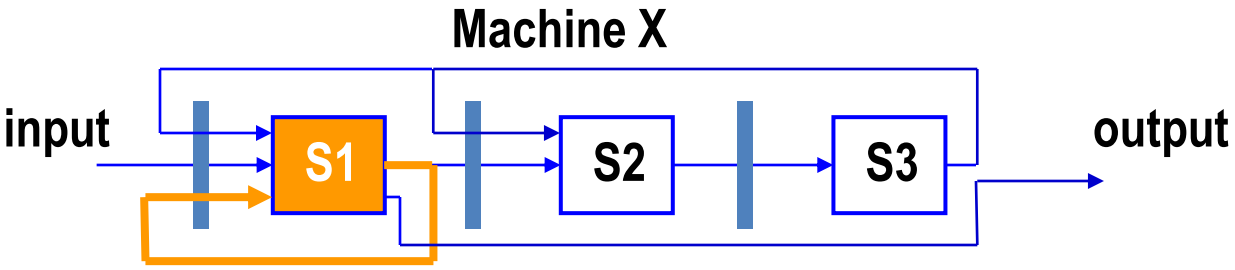

## Reservation Table

Time →

Stage

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

# Reservation Table



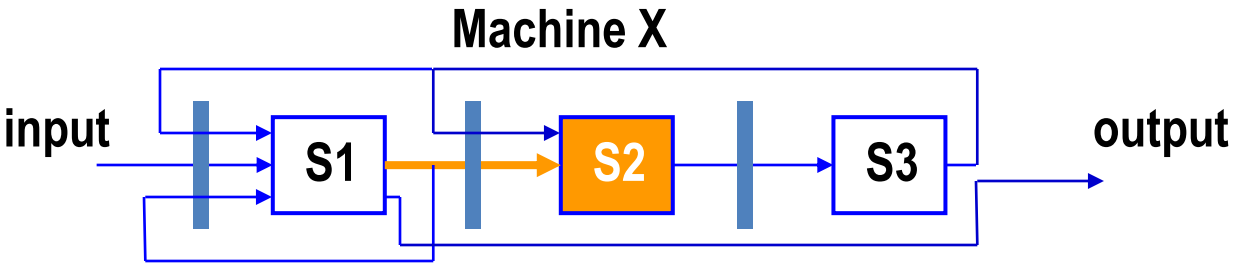

## Reservation Table

Time →

Stage

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

# Reservation Table



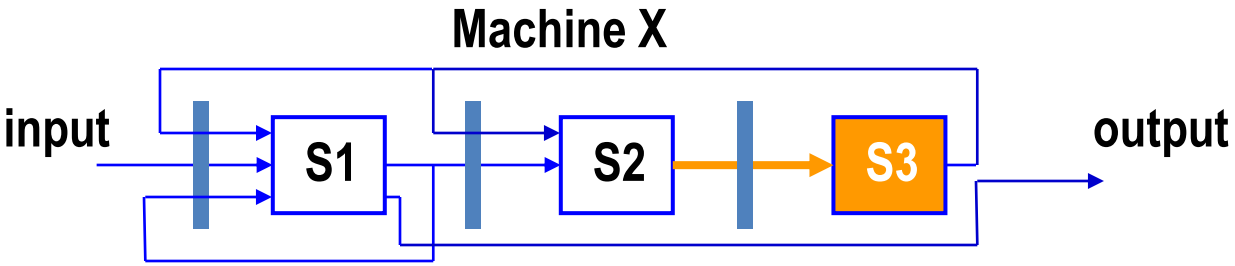

## Reservation Table

Time →

Stage

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

# Reservation Table



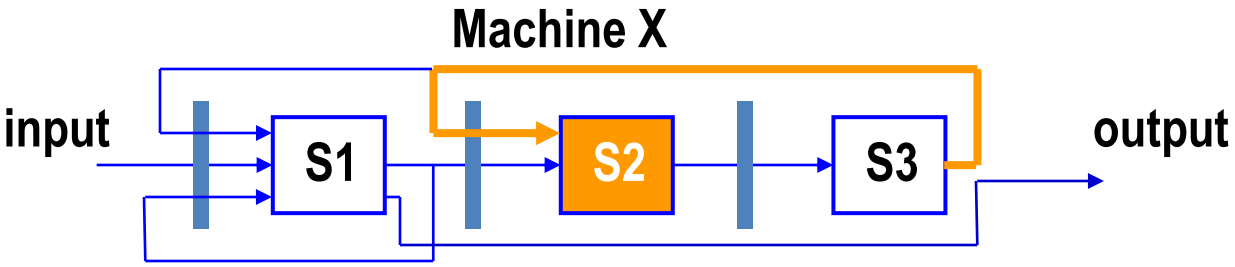

## Reservation Table

Time →

Stage

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

# Reservation Table



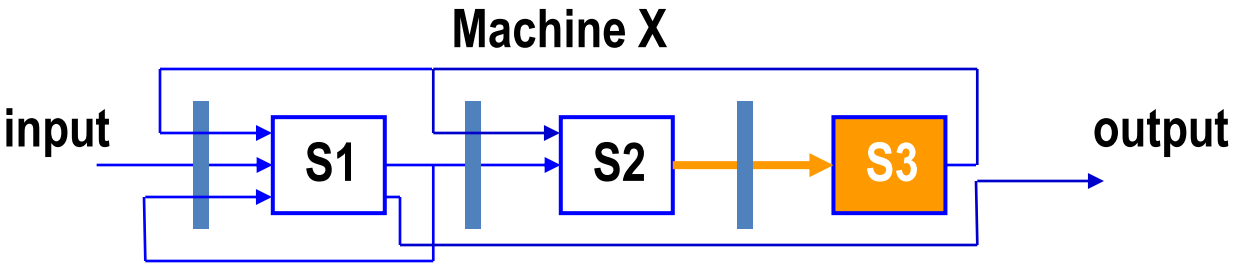

## Reservation Table

Time →

Stage

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

# Reservation Table



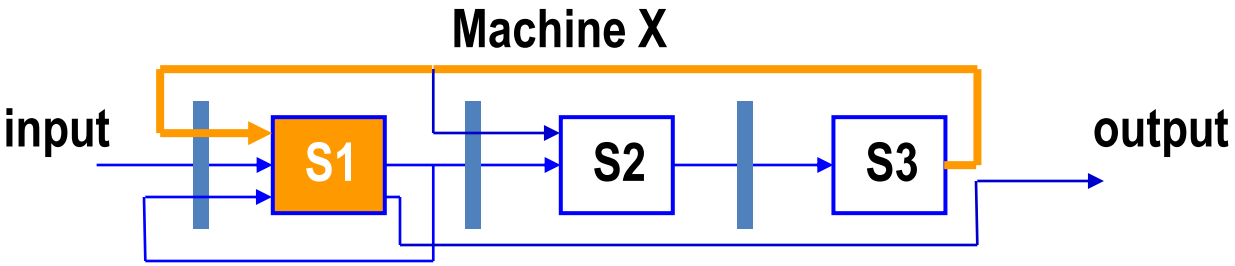

## Reservation Table

Time →

Stage

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

# Reservation Table



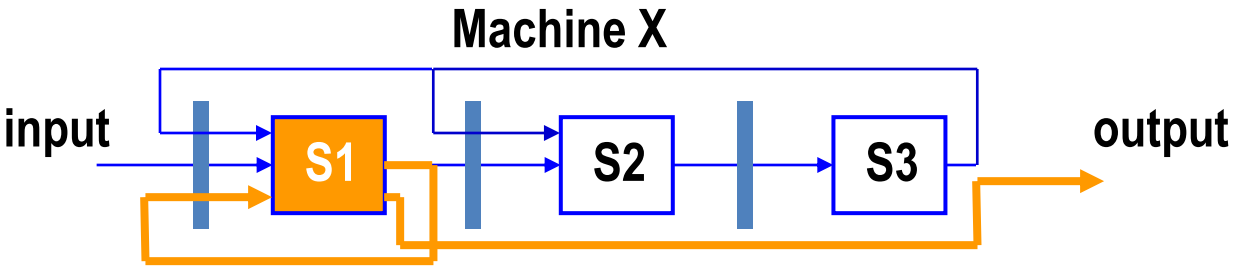

## Reservation Table

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.

# Important Terms

Clock period: $\tau$ Tau

$\tau_i$: time delay of $S_i$ stage

$\tau_L$: time delay of latch

$$\tau = \max\{\tau_i\} + \tau_L$$

20 + 2

= 22 ns

Pipeline processor frequency $f = 1/\tau$

= 1/22 Hz

F I 5

D I 10

F O 5

E T 20

+10 10

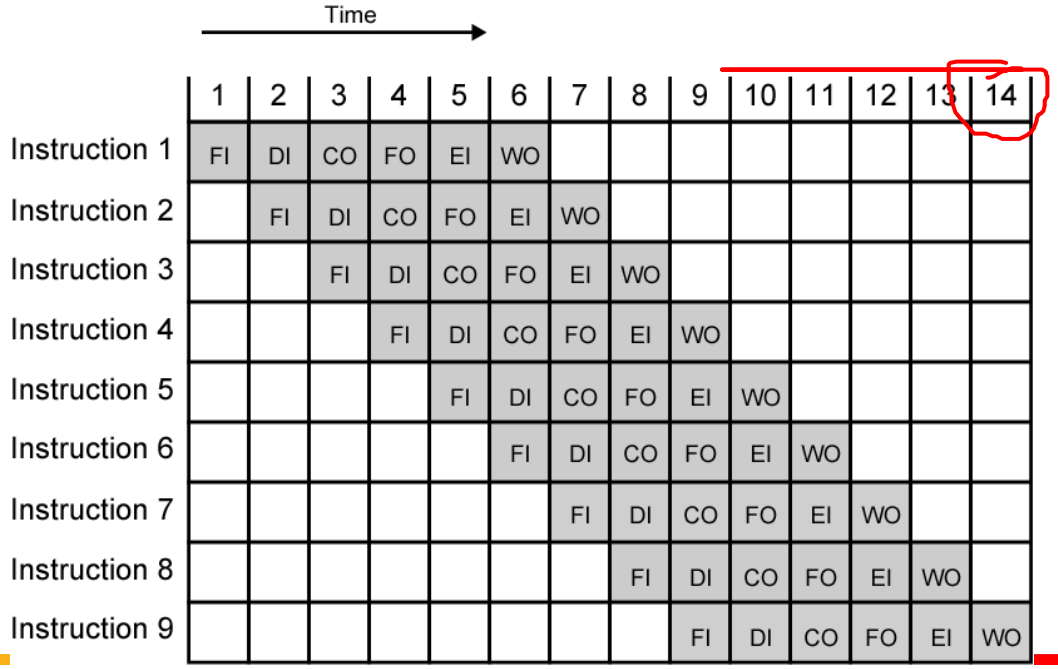
# Important Terms

Time taken to complete n tasks by k stage pipeline is

$$T_k = [k + (n-1)]\tau$$

Time taken by the nonpipelined processor

?



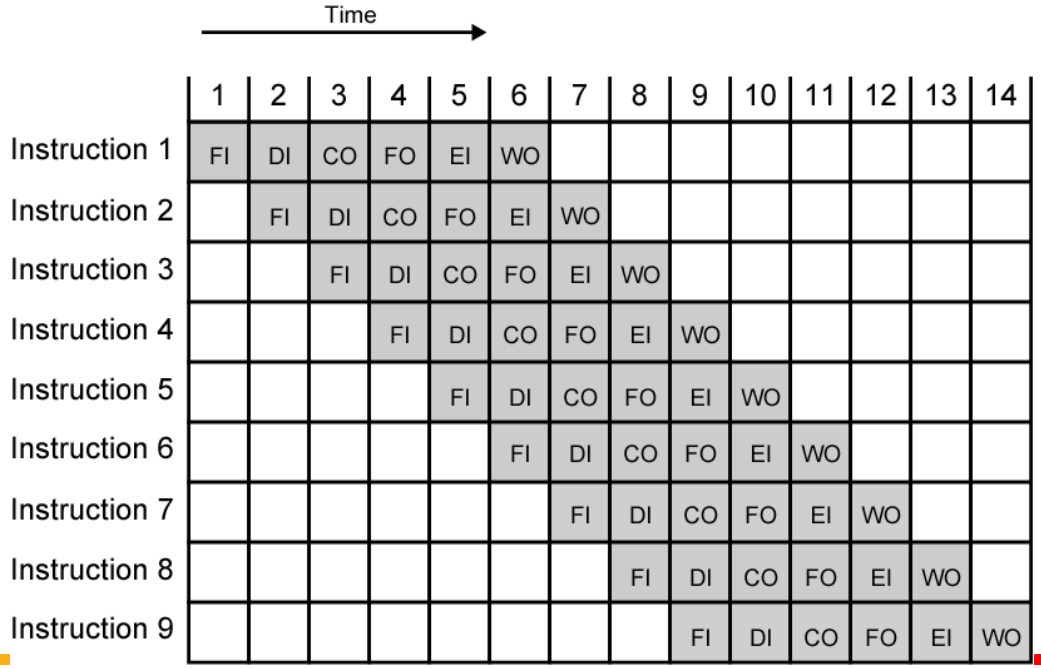
# Important Terms

Time taken to complete n tasks by k stage pipeline is

$$T_k = [k + (n-1)]\tau$$

Time taken by the nonpipelined processor

$$T_1 = k^* n^* \tau$$



# Important Terms



Speedup: speedup of a k-stage linear-pipeline over an equivalent non pipelined processor

$$S_k = \frac{T_1}{T_k}$$

$$= \frac{n*k*\tau}{[k+(n-1)]\tau}$$

$$= \frac{n*k}{[k+(n-1)]}$$

# Important Terms……

The maximum speedup is $S_k \rightarrow k$ when $n \rightarrow INF$

Maximum speedup is very difficult to achieve because of data dependencies between successive tasks, program branches, interrupts etc.

# Important Terms



Efficiency: the ratio of actual speedup to ideal speedup k

$$\eta = \frac{n.k}{k.\left[k + (n-1)\right]}$$

$$= \frac{n}{\left[k + (n-1)\right]}$$

# Contd…

- Maximum efficiency

$$\eta \to 1 \text{ as } n \to \infty$$

- Implies that the larger the number of tasks flowing through the pipeline, the better is its efficiency
- In steady state of a pipeline, we have n >> k, then efficiency should approach 1
- However, this ideal case may not hold all the time because of program branches and interrupts and data dependencies

# Important Terms



**Throughput : The number of tasks that can be completed by a pipeline per unit time**

$$H_k = \frac{n}{[k + (n-1)]\tau} = \frac{nf}{[k + (n-1)]} = \eta f$$

# Problem 1

Draw a space-time diagram for a six segment pipeline showing the time it takes to process eight tasks

Determine the number of clock cycles that it takes to process 200 tasks in a six segment pipeline

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| T1 | S1 | S2 | S3 | S4 | S5 | S6 | | | | | | | |
| T2 | | S1 | S2 | S3 | S4 | S5 | S6 | | | | | | |
| T3 | | | S1 | S2 | S3 | S4 | S5 | S6 | | | | | |
| T4 | | | | S1 | S2 | S3 | S4 | S5 | S6 | | | | |
| T5 | | | | | S1 | S2 | S3 | S4 | S5 | S6 | | | |
| T6 | | | | | | S1 | S2 | S3 | S4 | S5 | S6 | | |
| T7 | | | | | | | S1 | S2 | S3 | S4 | S5 | S6 | |
| T6 | | | | | | | | S1 | S2 | S3 | S4 | S5 | S6 |

# Problem 1



Draw a space-time diagram for a six segment pipeline showing the time it takes to process eight tasks

Determine the number of clock cycles that it takes to process 200 tasks in a six segment pipeline

# Problem 2

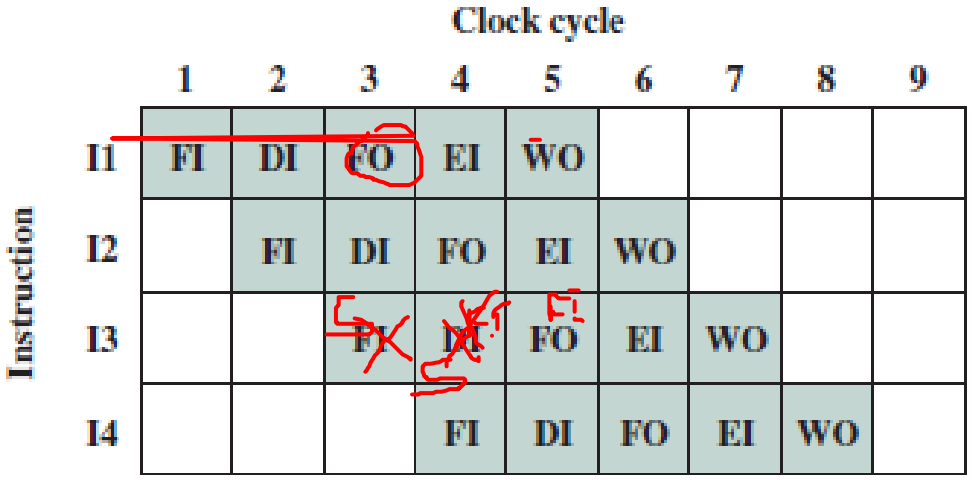
Assume each task is subdivided in to 6 subtasks and clock cycle is 10 microseconds.

- Determine the number of clock cycles that is taken to process 50 tasks in six stage pipeline
- Determine the number of clock cycles that is taken to process 50 tasks in non- pipeline processor
- Compute speed up, efficiency and throughput

# Pipeline Hazards / Conflicts

- Resource Hazard: access to same resource by two segments at the same time
- Data Hazard : an instruction depends on the result of a previous instruction, but this result is not yet available
- Control Hazard: arise from branch and other instructions that change the value of PC

# Resource Hazard




(a) Five-stage pipeline, ideal case

# Problem



Consider the following code. Assume that initial contents of all the registers is zero.

```
START:    MOV R3, #1
          MOV R1, #2
          MOV R2, #3
          ADD R3, R1, R2
          SUB R4, R3, R2
          HLT
```

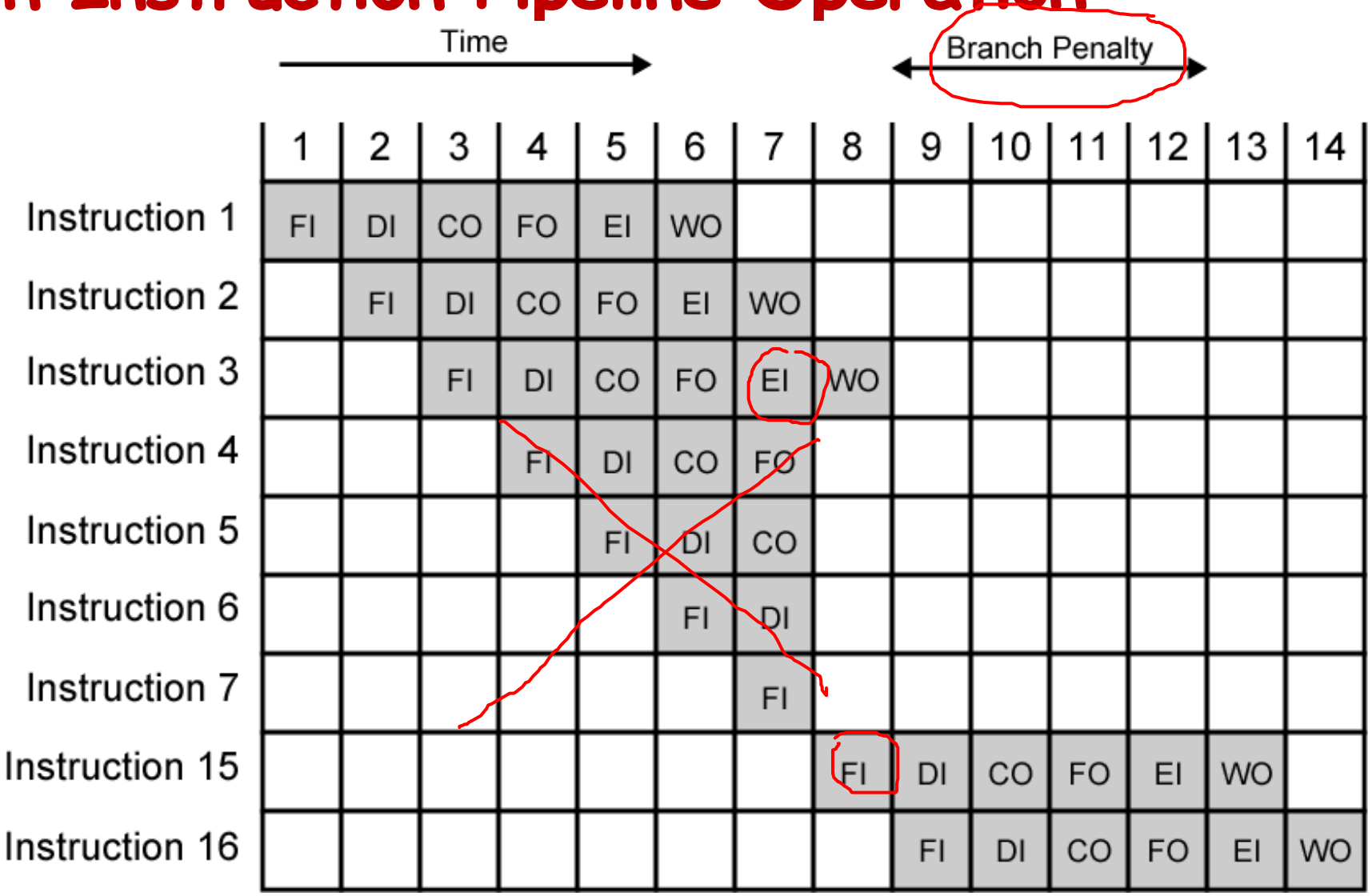
- Write timing of instruction pipeline with FIVE stages
- What is the content of various registers after the execution of program?

# Data Dependency Conflict

```
I1  MOV R3, #1
2   MOV R1, #2
3   MOV R2, #3
4   ADD R3, R1, R2
5   SUB R4, R3, R2
6   HLT
```

| time | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
|---|---|---|---|---|---|---|---|---|---|---|
| I1 | **FI** | **DI** | **FO** | **EI** | **WO** | | | | | |
| I2 | | **FI** | **DI** | **FO** | **EI** | **WO** | | | | |
| I3 → | | | **FI** | **DI** | **FO** | **EI** | **WO** | | | |
| I4 → | | | | **FI** | **DI** | **FO** | **EI** | **WO** | | |
| I5 → | | | | | **FI** | **DI** | **FO** | **EI** | **WO** | |
| I 6 | | | | | | **FI** | **DI** | **FO** | **EI** | **WO** |

# The Effect of a Conditional Branch on Instruction Pipeline Operation



# Dealing with Branches



- Multiple Streams
- Prefetch Branch Target
- Loop buffer
- Branch prediction
- Delayed branching

# Multiple Streams



- Have two pipelines
- Pre-fetch each branch into a separate pipeline
- Use appropriate pipeline
- Used by IBM 370/168 and the IBM 3033
- Issues :
  - Leads to bus & register contention
  - Multiple branches lead to further pipelines being needed

# Prefetch Branch Target

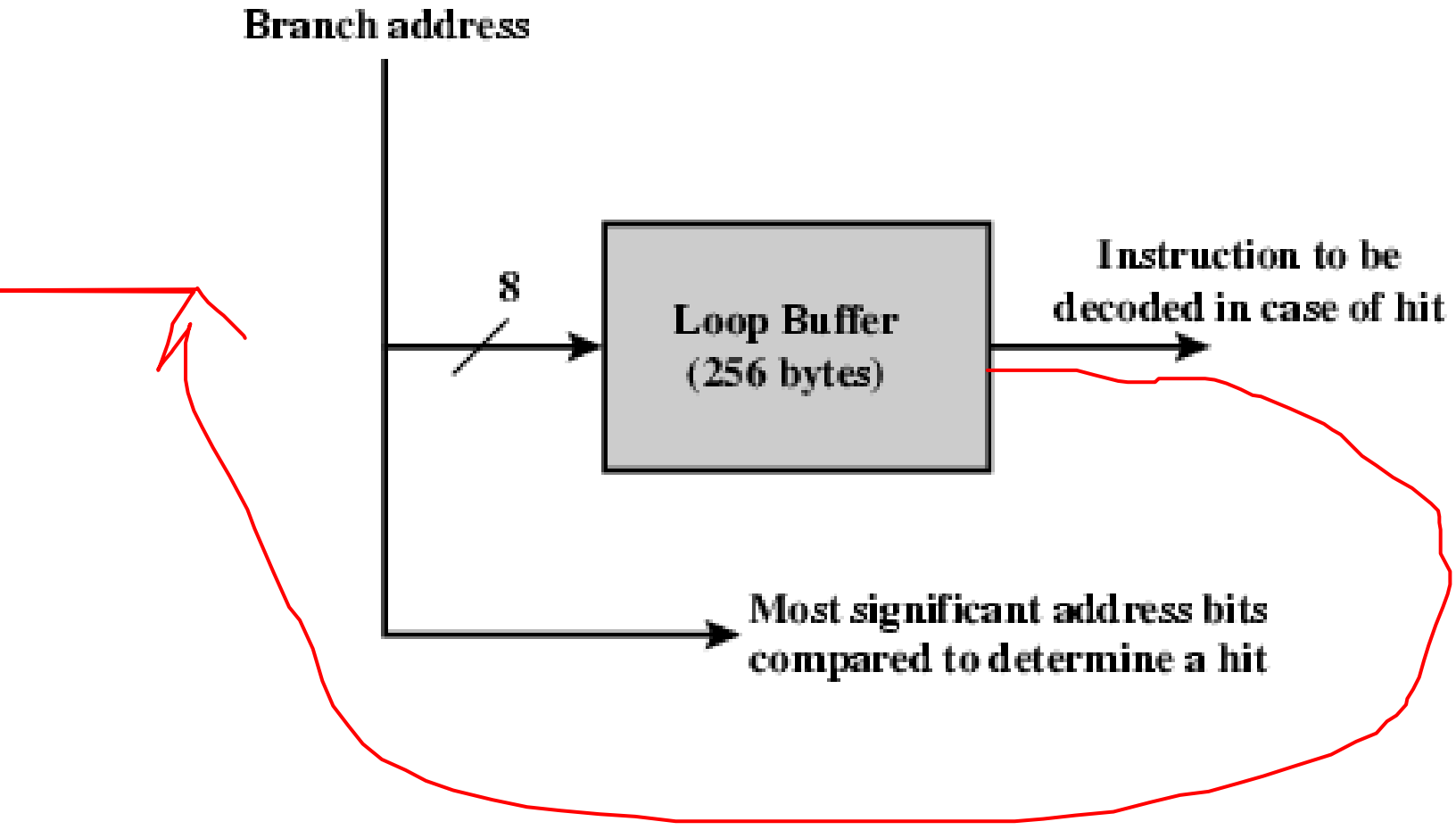
- Target of branch is prefetched in addition to instructions following branch
- Keep target until branch is executed
- Used by IBM 360/91

# Loop Buffer



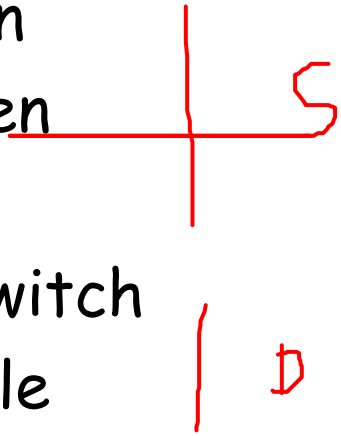
- Very fast memory
- Maintained by fetch stage of pipeline
- Check buffer before fetching from memory
- Very good for small loops or jumps
- Working principle is similar to cache
- Used by CRAY-1

# Loop Buffer Diagram



# Branch Prediction (1)

- Predict never taken
- Predict always taken
- Predict by opcode
- Taken/not taken switch
- Branch history table



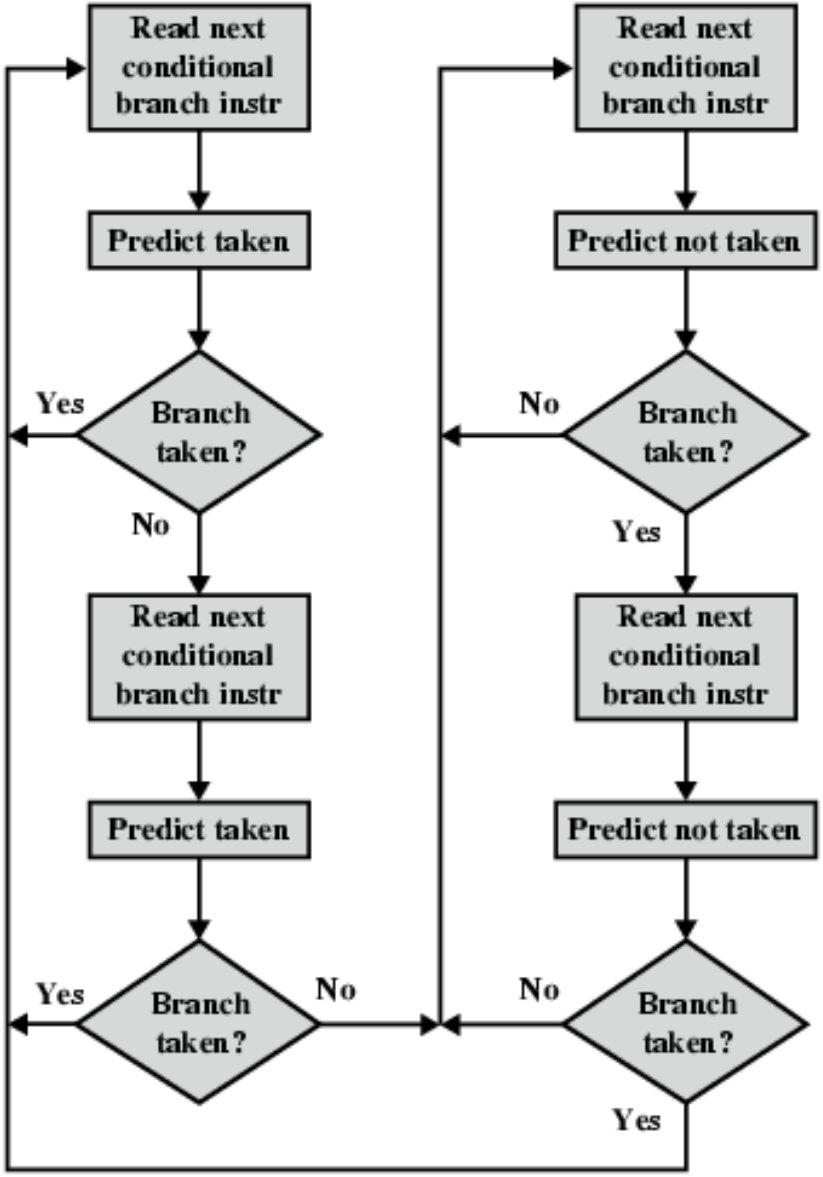
# Branch Prediction (2)

- Predict never taken
  - Assume that jump will not happen
  - Always fetch next instruction
- Predict always taken
  - Assume that jump will happen
  - Always fetch target instruction
- Predict by Opcode
  - Some instructions are more likely to result in a jump than others
  - Can get up to 75% success

# Branch Prediction (3)

- Taken/Not taken switch
  - Based on previous history
  - Good for loops

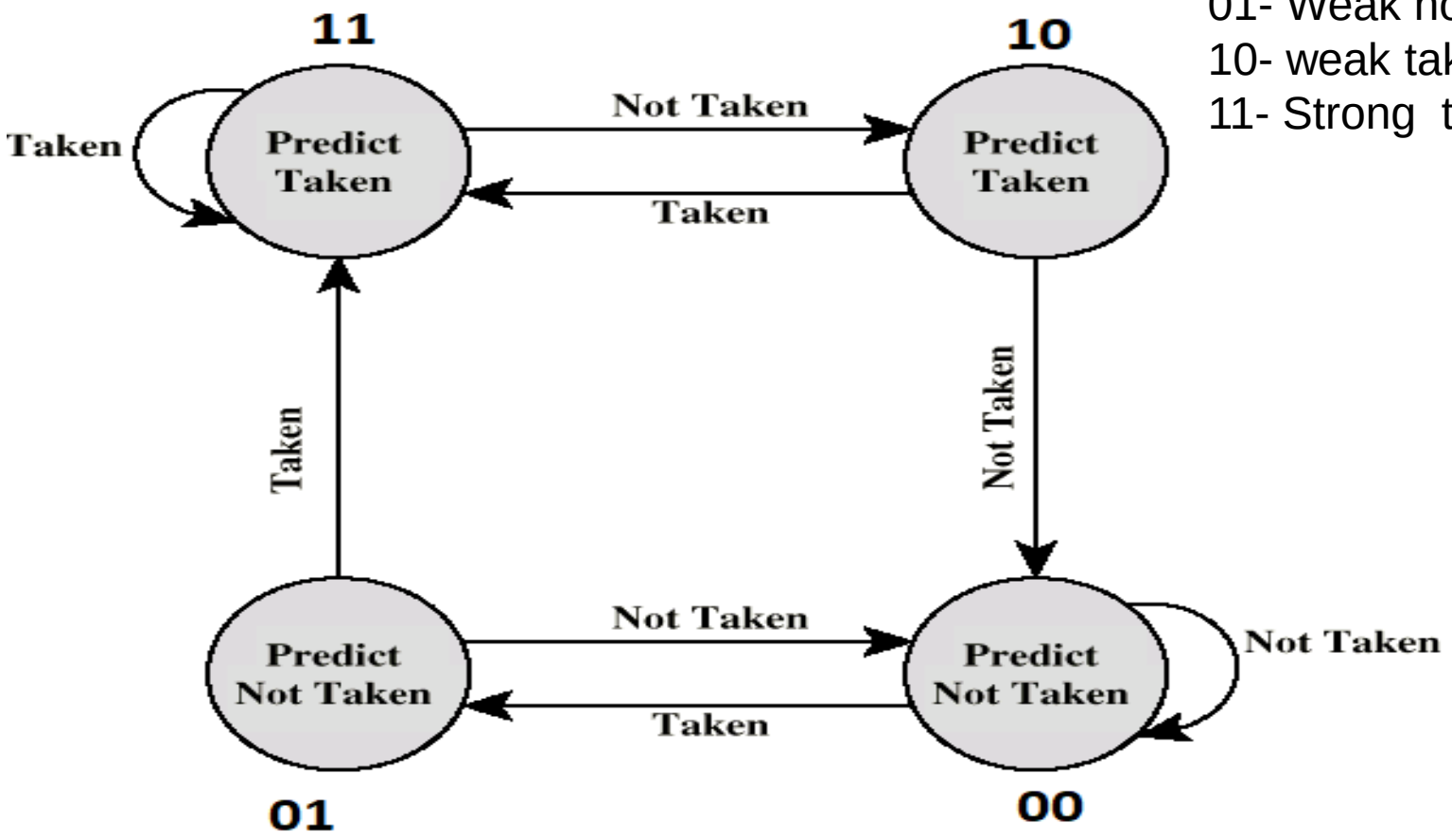
# Branch Prediction Flowchart



# Branch Prediction State Diagram

First bit is prediction bit and second bit is conviction bit

00 – strong not taken
01- Weak not taken
10- weak taken
11- Strong taken



# Branch Prediction (4)

- Delayed Branch
  - Do not take jump until you have to
  - Rearrange instructions

# Problem 3

innovate achieve lead

Consider a computer system that requires four stages to execute an instruction. The details of the stages and their time requirements are listed below:

Instruction Fetch (IF) stage: 30 ns,

Instruction Decode (ID) stage: 9 ns

Execute (EX) stage: 20 ns

Store Results (WO stage): 10 ns.

Assume that inter-stage delay is 1 ns and every instruction in the program must proceed through the stages in sequence.

a) What is the frequency of the Processor?

b) What is the minimum time for 10 instructions to complete in non-pipelined manner?

c) What is the minimum time for 10 instructions to complete in pipelined manner?

# Problem 4



A computer program is developed to run the DBSCAN algorithm on a processor with L1 cache and main memory organization with hit ratio of 67%. The processor supports 64-bit address bus, an 8-bit data bus and has a 256 KByte data cache memory with 4-way set associativity. The main memory is logically divided into a block size of 256 Bytes. Each cache line entry contains, in addition to address tag, 1 dirty bit.

i. What is the number of bits in the tag field of an address?

ii. If the associativity in the above problem, is changed to 8-way, do you see any performance gain and/ or drawback? Comment.

# Cont.

i. What is the number of bits in the tag field of an address?

Number of lines in the cache = 256 KB / 256 B = 2^10 = 1 K lines

4 way = 4 lines per set

Therefore Number of sets = 1K/ 4 = 256 sets = 2^8
Number of bits needed to address sets = 8 bits
Number bits needed to address a word in the block : 256 = 2^8→ 8 bits

Tag bit = 64 - 8 -8 = 48 bits

# Cont.

ii. If the associativity in the above problem, is changed to 8-way, do you see any performance gain and/ or drawback?

There is a performance gain in terms number of HITS
Tag comparison circuit becomes complex

# Problem 5

A RISC based CPU is to be designed to have 32 opcodes, source and destination operands referring to 64 registers, and a displacement of value such as 2ABCH. Specify the instruction format mentioning the various fields and bits required by them.

| Opcode [5 bits] | Source operand [6bits] | Destination operand [6 bits] | Displacement [16 Bits] |
|---|---|---|---|

# Problem 6



A company named CacheFul is designing a machine with a byte addressable main memory. The size of the main memory is 218 Bytes and block size is 16 Bytes. The machine employs a direct mapping cache consisting of 32 lines. This machine is specifically configured to run a classification algorithm

$(0.8 \times 3) + (1 - 0.8) \times 33)$

$= 1.4 + 6.6 = 9$ ns

i. When the algorithm is run it is noted that a memory access to main memory on a cache "miss" takes 30 ns and memory access to the cache on a cache "hit" takes 3 ns. If 80% of the processor's memory requests result in a cache "hit", how much time does it take to access memory on average?

# Cont.



ii. Recommended average memory access time for the algorithm to perform optimally is not more than 5 ns, what option as a developer, do you have to keep it near to 5ns?

$5ns = 3h + (1-h)*33$

$5ns = 3h + 33 - 33h$

$5ns = -30h + 33$

$30h = 28$

$H = 28/30 => 93\%$