BITS Pilani
Pilani Campus

Computer Organization and Software Systems

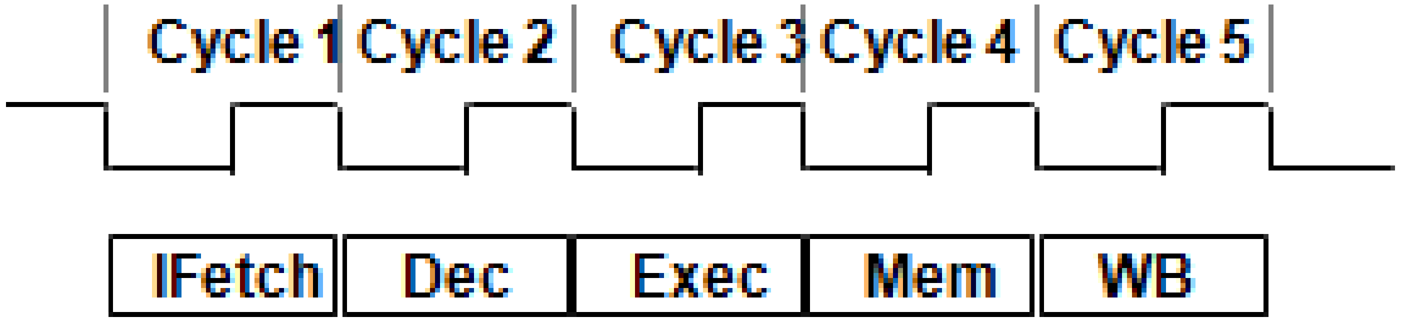
Session 7

Prof. C R Sarma
WILP.BITS-Pilani

Revision for Multi Cycle MIPS The Five Steps

RSC

- Step 1 : IF (Instruction Fetch)
- Step 2 : ID (Instruction Decode)
- Step 3 : EX (Execute)
- Step 4 : MEM (Memory)
- Step 5 : WB (Write Back)



I-Type Instruction: lw $s1,100[$s2]

Step1: IFetch

IR ← Memory [PC]

PC← PC + 4

Step 2: Dec

– read two registers corresponding to rs and rt fields

A ← Reg [IR [25:21]]

B ← Reg [IR [20:16]]

– compute branch target address with ALU

ALUout ← PC + (sign-extend (IR[15-0]) << 2)

Step 3: Exec

1. Memory reference

   ALUout ← A + sign-extend (IR[15:0])

2. Arithmetic and logical instruction

   ALUout ← A op B

3. Branch

   if (A == B) PC ← ALUout

4. Jump

   PC ← { PC[31:28], (IR [25:0] << 2)

Step 4: Mem

1. Memory Access

   MDR ← Memory [ALUout]

   or

   Memory [ALUout] ← B

2. R-Type Instruction Completion

   Reg [IR [15: 11]] ← ALUout

Step 5: WB

Memory read completion step

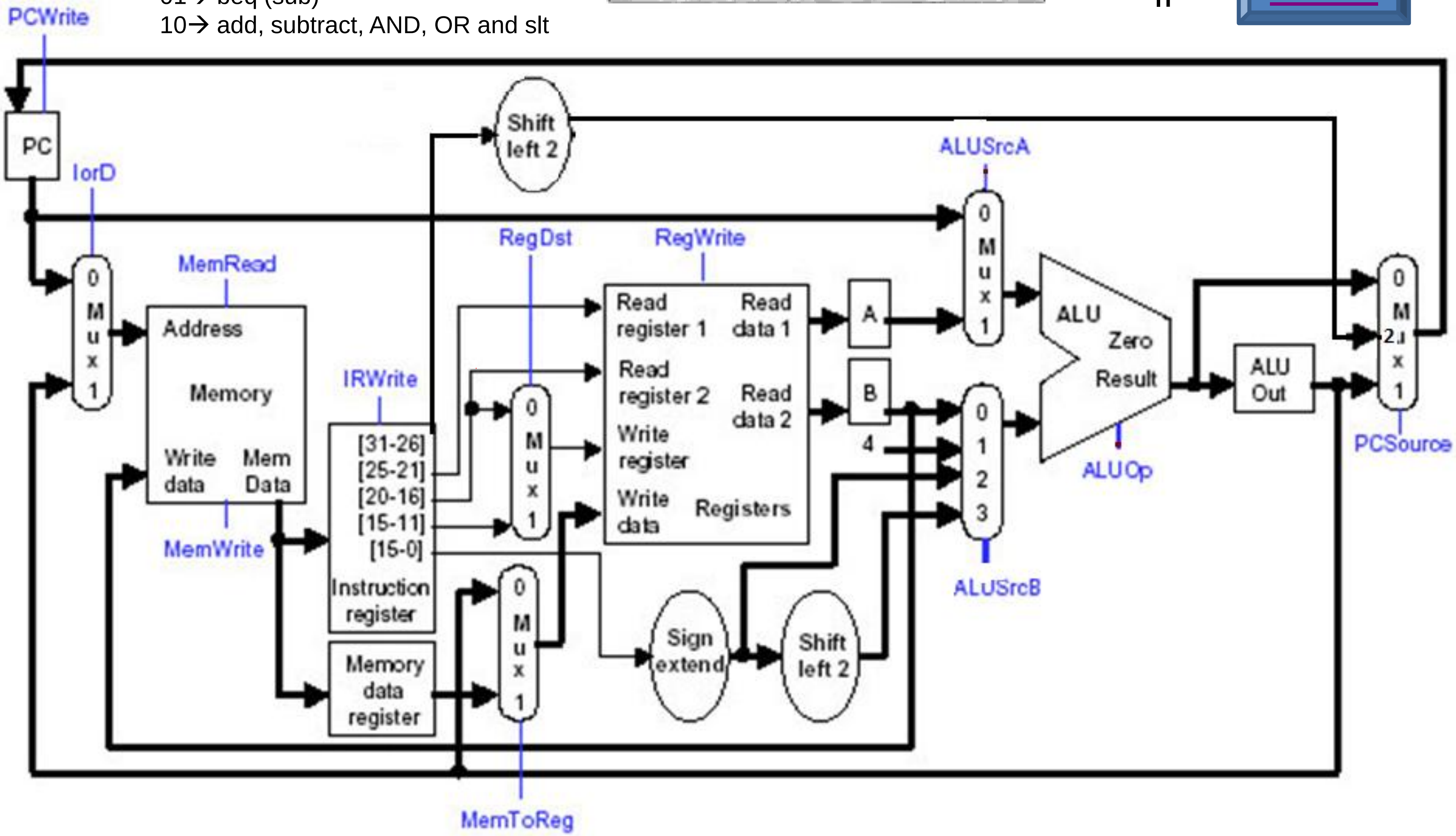
– Reg [IR [20 : 16]] ← MDR

- ALU control unit generates 4 bit ALUOperation control signal based on
  - function field
  - 2 bit control field → ALUOp
    - 00 → lw and sw
    - 01→ beq (sub)
    - 10→ add, subtract, AND, OR and slt

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

IF

BACK



Contd…

FigFig
Fig

- **Instruction Fetch :**
  - IR ← Memory [PC]:
    - 1 – MemRead
    - 1 – IRWrite
    - 0 – IorD
  - PC← PC + 4 :
    - 0 - ALUSrcA
    - 01 - ALUSrcB
    - 00 - ALUop
    - 00 - PCSource
    - 1 - PCWrite

00: Add operation
01: Subtract Operation
10:Function field of the instruction determines the ALU operation

00: PC + 4 output of ALU
01: branch target address ALUout
10:jump target address

I-Type Instruction: lw $s1,100[$s2]

Step1: IFetch

IR ← Memory [PC]

PC← PC + 4

Step 2: Dec

– read two registers corresponding to rs and rt fields

A ← Reg [IR [25:21]]

B ← Reg [IR [20:16]]

– compute branch target address with ALU

ALUout ← PC + (sign-extend (IR[15-0]) << 2)

Step 3: Exec

1. Memory reference

   ALUout ← A + sign-extend (IR[15:0])

2. Arithmetic and logical instruction

   ALUout ← A op B

3. Branch

   if (A == B) PC ← ALUout

4. Jump

   PC ← { PC[31:28], (IR [25:0] << 2)

Step 4: Mem

1. Memory Access

   MDR ← Memory [ALUout]

   or

   Memory [ALUout] ← B

2. R-Type Instruction Completion

   Reg [IR [15: 11]] ← ALUout

Step 5: WB

Memory read completion step

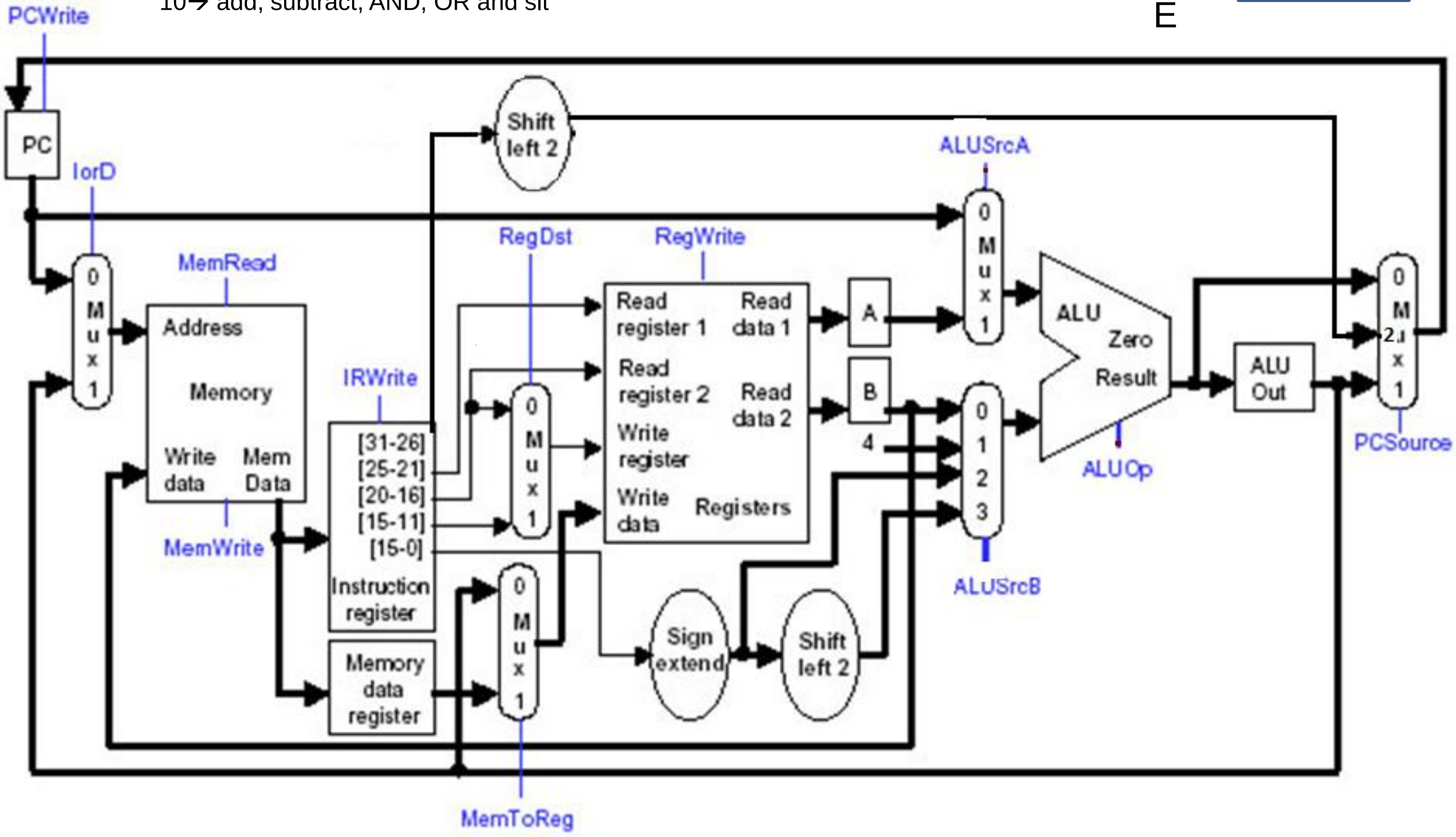
– Reg [IR [20 : 16]] ← MDR

- ALU control unit generates 4 bit ALUOperation control signal based on
  - function field
  - 2 bit control field → ALUOp
    - 00 → lw and sw
    - 01→ beq (sub)
    - 10→ add, subtract, AND, OR and slt

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

BACK

D
E



Second Step

0: First operand is PC
1: First operand is register

Instruction decode and re

- read two registers
  fields
- compute branch

A ← Reg [IR [2 21]]
B ← Reg [IR 0:16]]
ALUout ← P + (sign-e d (IR[15-0]) << 2)

Second operand
00: register
01: 4
10:Sign Extd lower 16 bits of IR
11 :Sign Extd lower 16 bits of IR << 2

0	ALUSrcA
11	ALUSrcB
00	ALUop

00: Add
01: Subtract
10:Function field determines the ALU operation

I-Type Instruction: lw $s1,100[$s2]

Step1: IFetch

IR ← Memory [PC]

PC← PC + 4

Step 3: Exec

1. Memory reference

   ALUout ← A + sign-extend (IR[15:0])

2. Arithmetic and logical instruction

   ALUout ← A op B

3. Branch

   if (A == B) PC ← ALUout

4. Jump

   PC ← { PC[31:28], (IR [25:0] << 2)

Step 5: WB

Memory read completion step

– Reg [IR [20 : 16]] ← MDR

Step 2: Dec

– read two registers corresponding to rs and rt fields

  A ← Reg [IR [25:21]]

  B ← Reg [IR [20:16]]

– compute branch target address with ALU

  ALUout ← PC + (sign-extend (IR[15-0]) << 2)

Step 4: Mem

1. Memory Access

   MDR ← Memory [ALUout]

   or

   Memory [ALUout] ← B

2. R-Type Instruction Completion

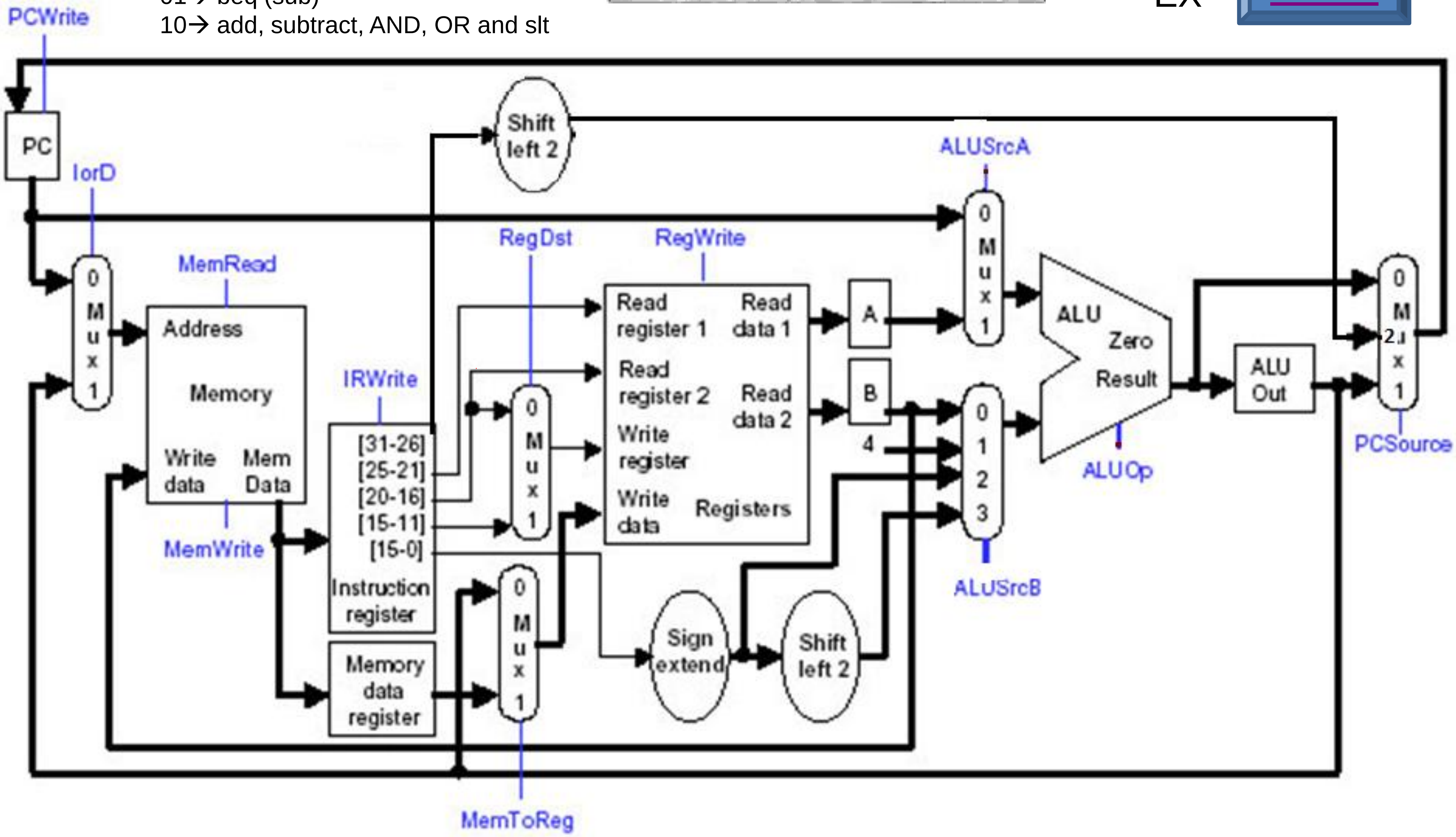
   Reg [IR [15: 11]] ← ALUout

- ALU control unit generates 4 bit ALUOperation control signal based on
  - function field
  - 2 bit control field → ALUOp
    - 00 → lw and sw
    - 01→ beq (sub)
    - 10→ add, subtract, AND, OR and slt

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

EX

BACK



Fig

Step 3: Contd…

0: First operand is PC

1: First operand is register A

1.Memory reference

ALUout ← A + sign-extend (IR[15:0])

1 - ALUSrcA

Second operand

00: register

01: 4

10:Sign Extd lower 16 bits of IR

11 :Sign Extd lower 16 bits of IR << 2

10 - ALUSrcB

00 - ALUop

00: Add operation

01: Subtract Operation

10:Function field of the instruction determines the ALU operation

I-Type Instruction: lw $s1,100[$s2]

Step1: IFetch

IR ← Memory [PC]

PC← PC + 4

Step 3: Exec

1. Memory reference

   ALUout ← A + sign-extend (IR[15:0])
2. Arithmetic and logical instruction

   ALUout ← A op B
3. Branch

   if (A == B) PC ← ALUout
4. Jump

   PC ← { PC[31:28], (IR [25:0] << 2)

Step 5: WB

Memory read completion step

– Reg [IR [20 : 16]] ← MDR

Step 2: Dec

– read two registers corresponding to rs and rt fields

A ← Reg [IR [25:21]]

B ← Reg [IR [20:16]]

– compute branch target address with ALU

ALUout ← PC + (sign-extend (IR[15-0]) << 2)

Step 4: Mem

1. Memory Access

   MDR ← Memory [ALUout]

   or

   Memory [ALUout] ← B
2. R-Type Instruction Completion

   Reg [IR [15: 11]] ← ALUout

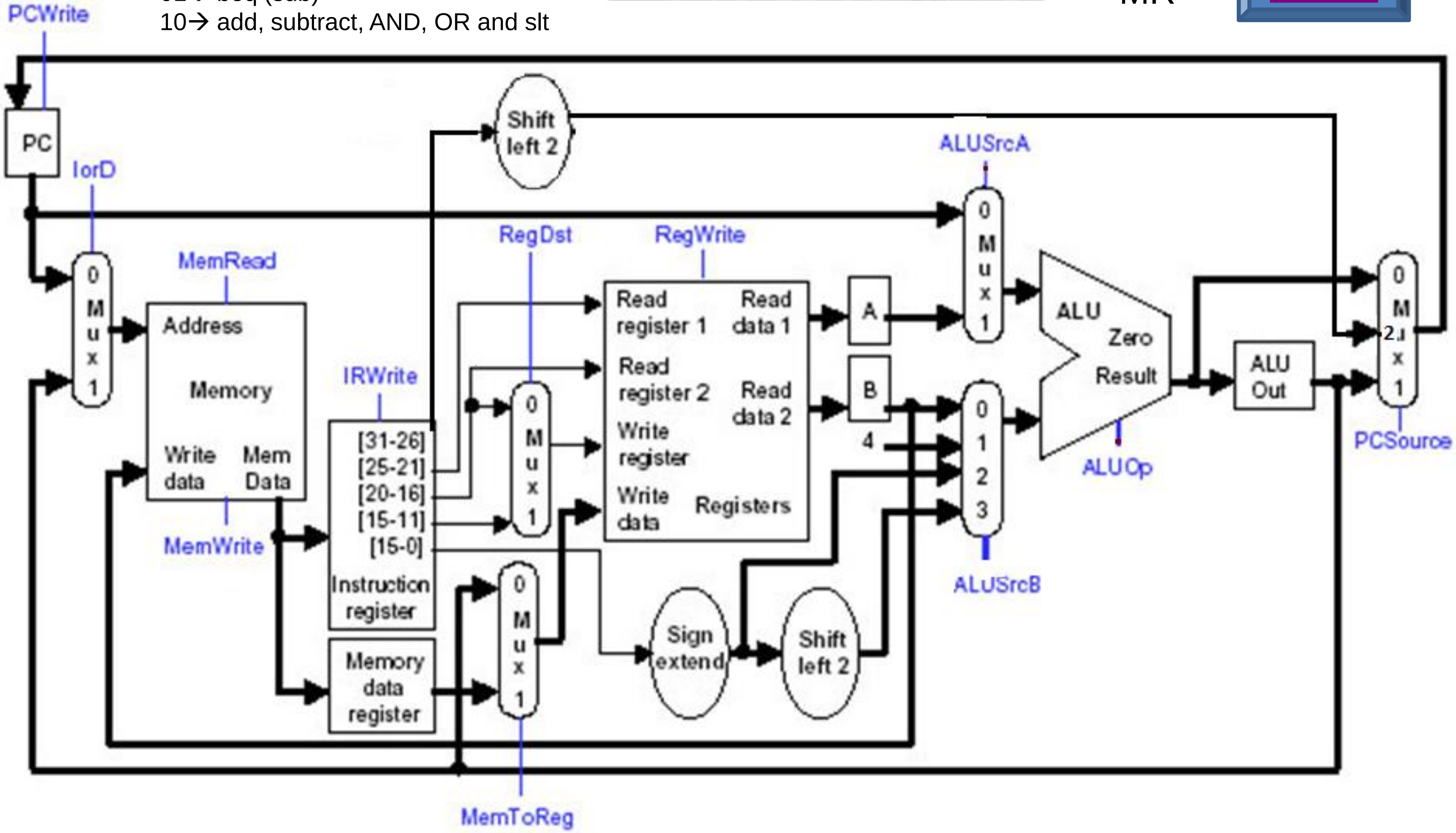
•ALU control unit generates 4 bit ALUOperation control signal based on
- function field
- 2 bit control field → ALUOp
  - 00 → lw and sw
  - 01→ beq (sub)
  - 10→ add, subtract, AND, OR and slt

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

innovate achieve lead

MR

BACK



Step 4 Contd…

FIG

Memory reference

lw

MDR 🡐 Memory [ALUout]

1 MemRead

1 IorD

sw

Memory [ALUout] 🡐 B

1 MemWrite

1 IorD

I-Type Instruction: lw $s1,100[$s2]

Step1: IFetch

IR ← Memory [PC]

PC← PC + 4

Step 3: Exec

1. Memory reference

   ALUout ← A + sign-extend (IR[15:0])

2. Arithmetic and logical instruction

   ALUout ← A op B

3. Branch

   if (A == B) PC ← ALUout

4. Jump

   PC ← { PC[31:28], (IR [25:0] << 2)

Step 5: WB

Memory read completion step

– Reg [IR [20 : 16]] ← MDR

Step 2: Dec

– read two registers corresponding to rs and rt fields

  A ← Reg [IR [25:21]]

  B ← Reg [IR [20:16]]

– compute branch target address with ALU

  ALUout ← PC + (sign-extend (IR[15-0]) << 2)

Step 4: Mem

1. Memory Access

   MDR ← Memory [ALUout]

   or

   Memory [ALUout] ← B

2. R-Type Instruction Completion

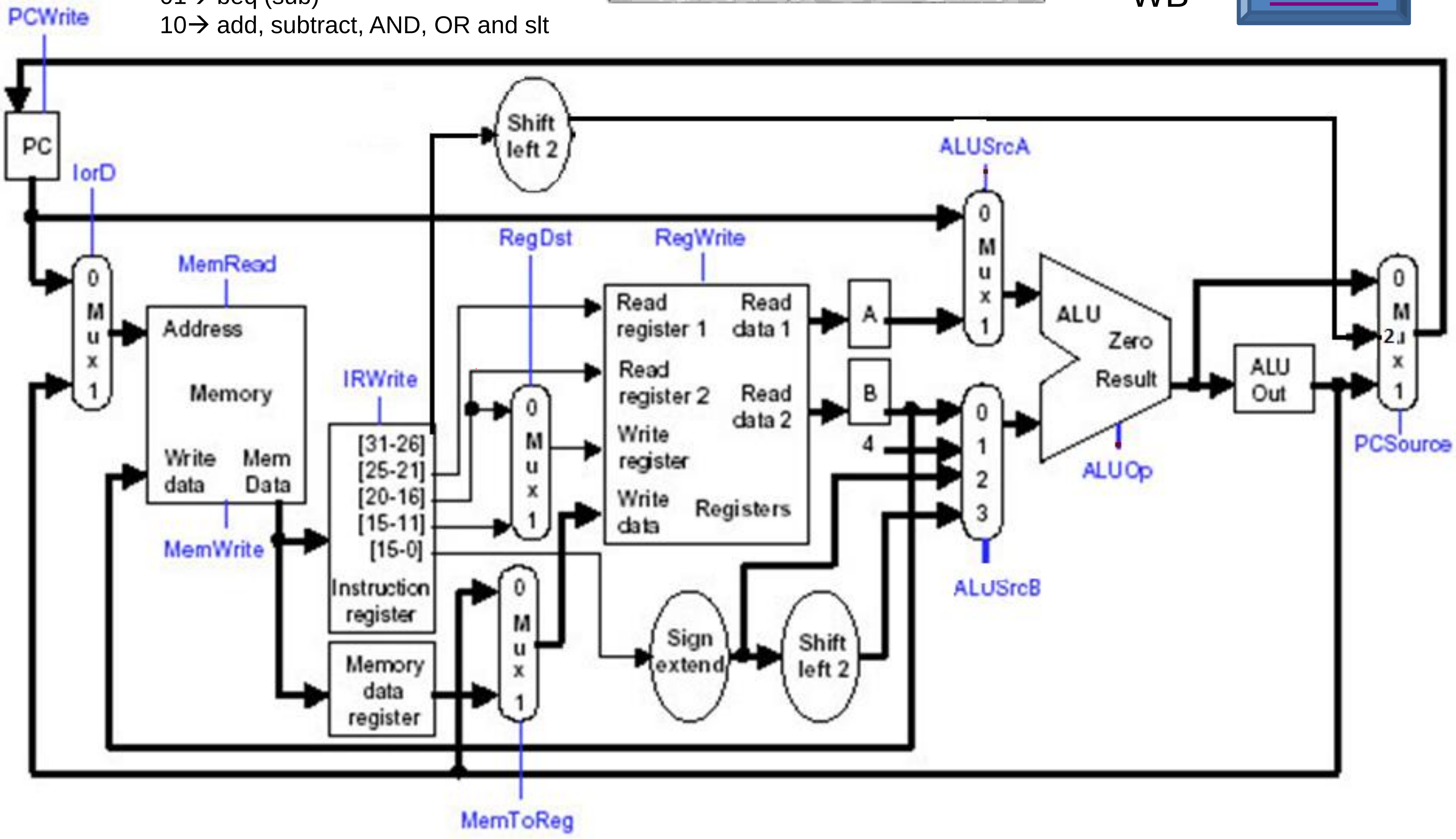
   Reg [IR [15: 11]] ← ALUout

- ALU control unit generates 4 bit ALUOperation control signal based on
  - function field
  - 2 bit control field → ALUOp
    - 00 → lw and sw
    - 01→ beq (sub)
    - 10→ add, subtract, AND, OR and slt

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

WB

BACK



Step 5

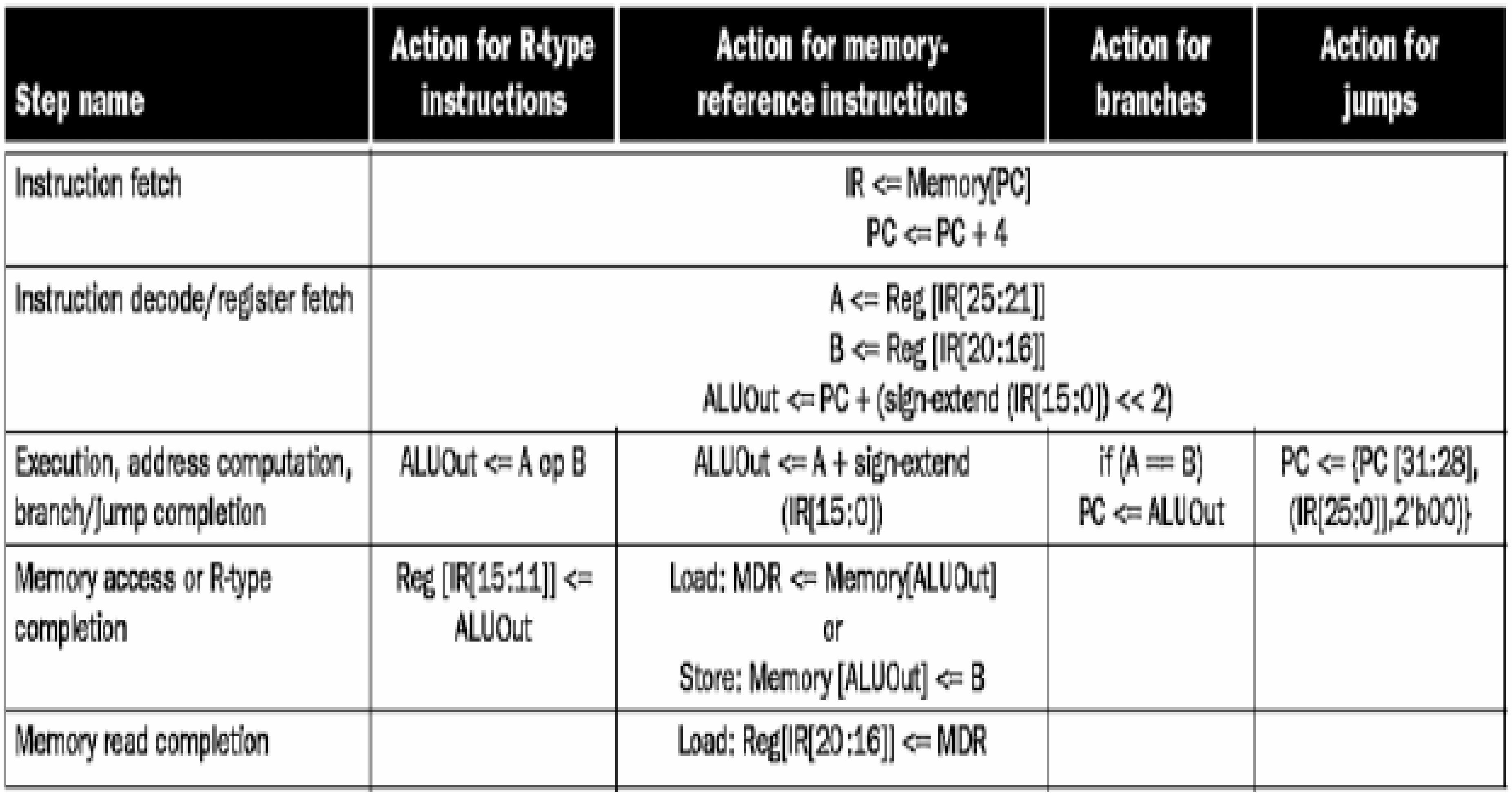
Fig

Write back or Memory read completion step

- Reg [IR [20 : 16]] 🡐 MDR

MemtoReg	1
RegWrite	1
RegDst	0

Summary



Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
Instruction fetch	IR <= Memory[PC] PC <= PC + 4			
Instruction decode/register fetch	A <= Reg [IR[25:21]] B <= Reg [IR[20:16]] ALUOut <= PC + (sign-extend (IR[15:0]) << 2)			
Execution, address computation, branch/jump completion	ALUOut <= A op B	ALUOut <= A + sign-extend (IR[15:0])	if (A == B) PC <= ALUOut	PC <= {PC [31:28], (IR[25:0]],2'b00)}
Memory access or R-type completion	Reg [IR[15:11]] <= ALUOut	Load: MDR <= Memory[ALUOut] or Store: Memory [ALUOut] <= B		
Memory read completion		Load: Reg[IR[20:16]] <= MDR		

BITS Pilani
Pilani Campus

Control Unit Design

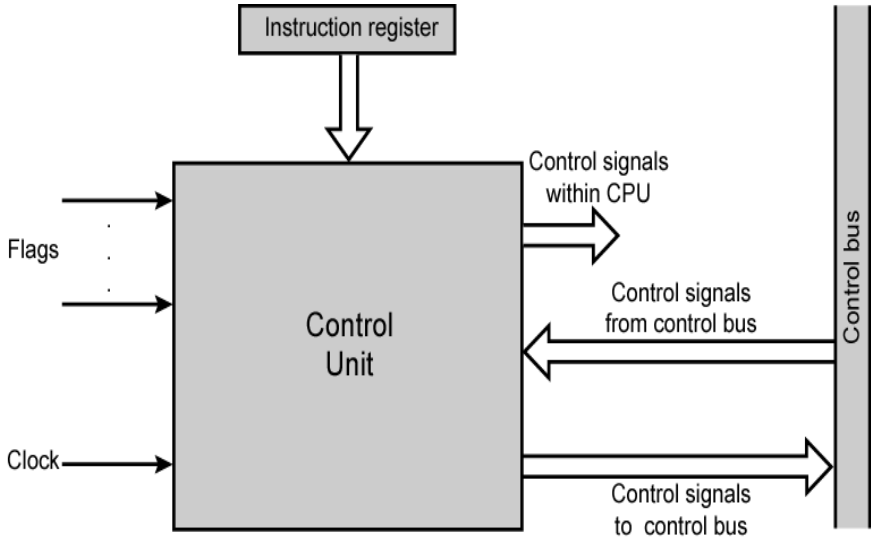
Control Unit implementation

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



Instruction register
Flags
Clock
Control Unit
Control signals within CPU
Control signals from control bus
Control signals to control bus
Control bus

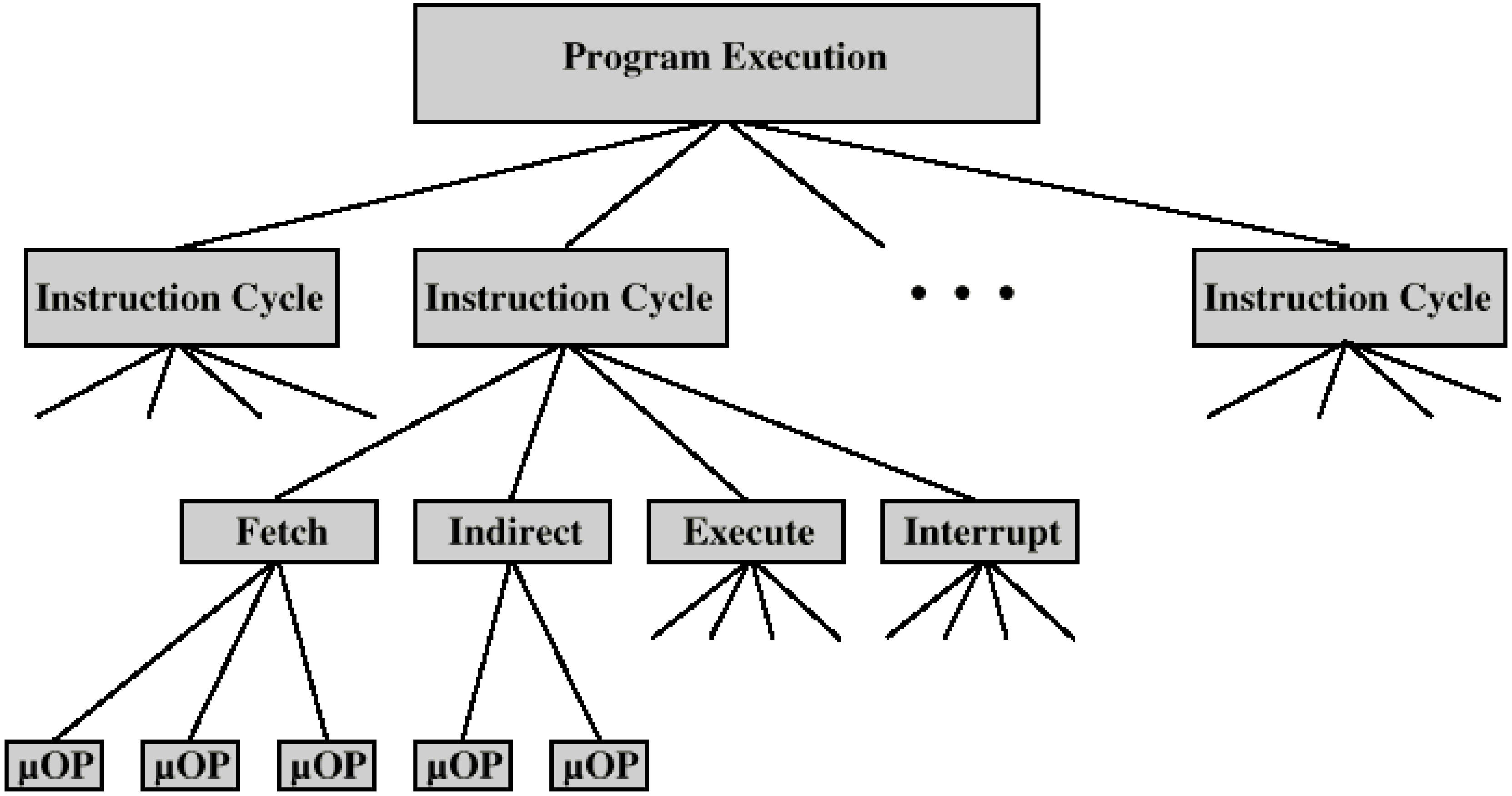
Problems With Hard Wired Designs

- 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 $\leftarrow$ (PC)

t2: MBR $\leftarrow$ (memory)

PC $\leftarrow$ (PC) +1

t3: IR $\leftarrow$ (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: $MAR \leftarrow (IR_{address})$

t2: $MBR \leftarrow (memory)$

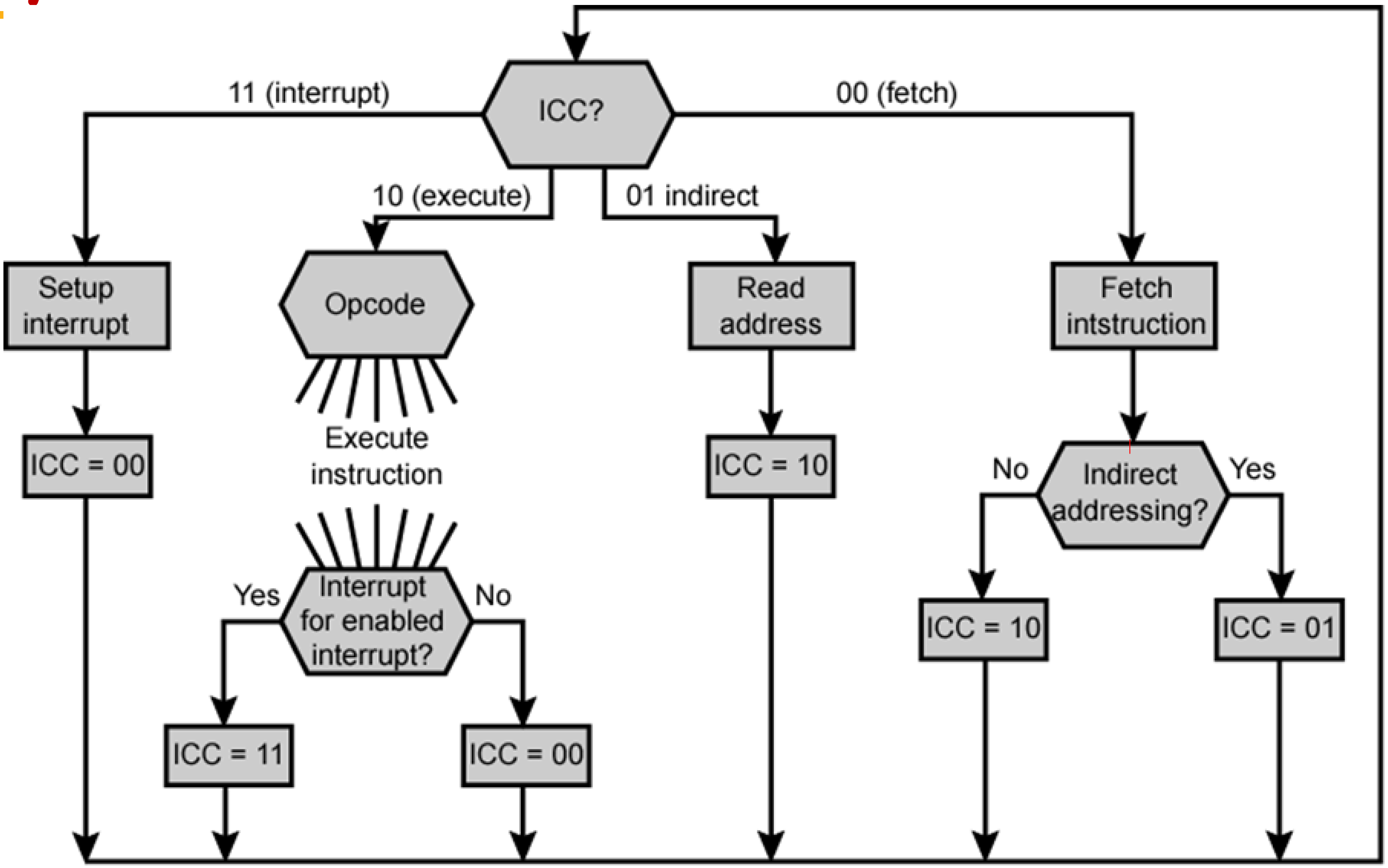
t3: $R1 \leftarrow R1 + (MBR)$

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

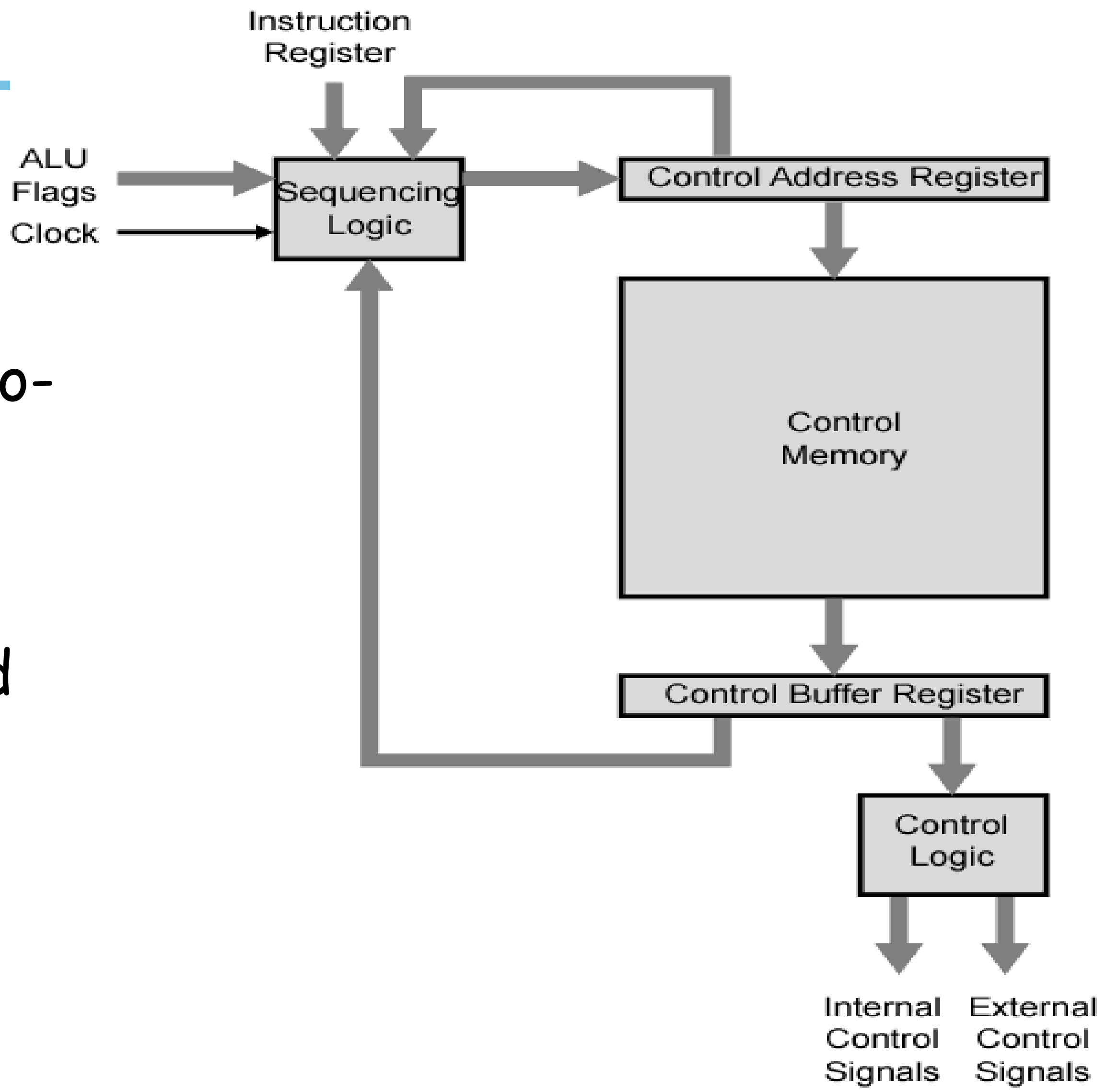
t1:	MAR ← (PC)
t2:	MBR ← (memory)
	PC ← (PC) +1
t3:	IR ← (MBR)
t4:	MAR ← ($IR_{address}$)
t5:	MBR ← (memory)
t6:	R1 ← R1 + (MBR)

Control Signals

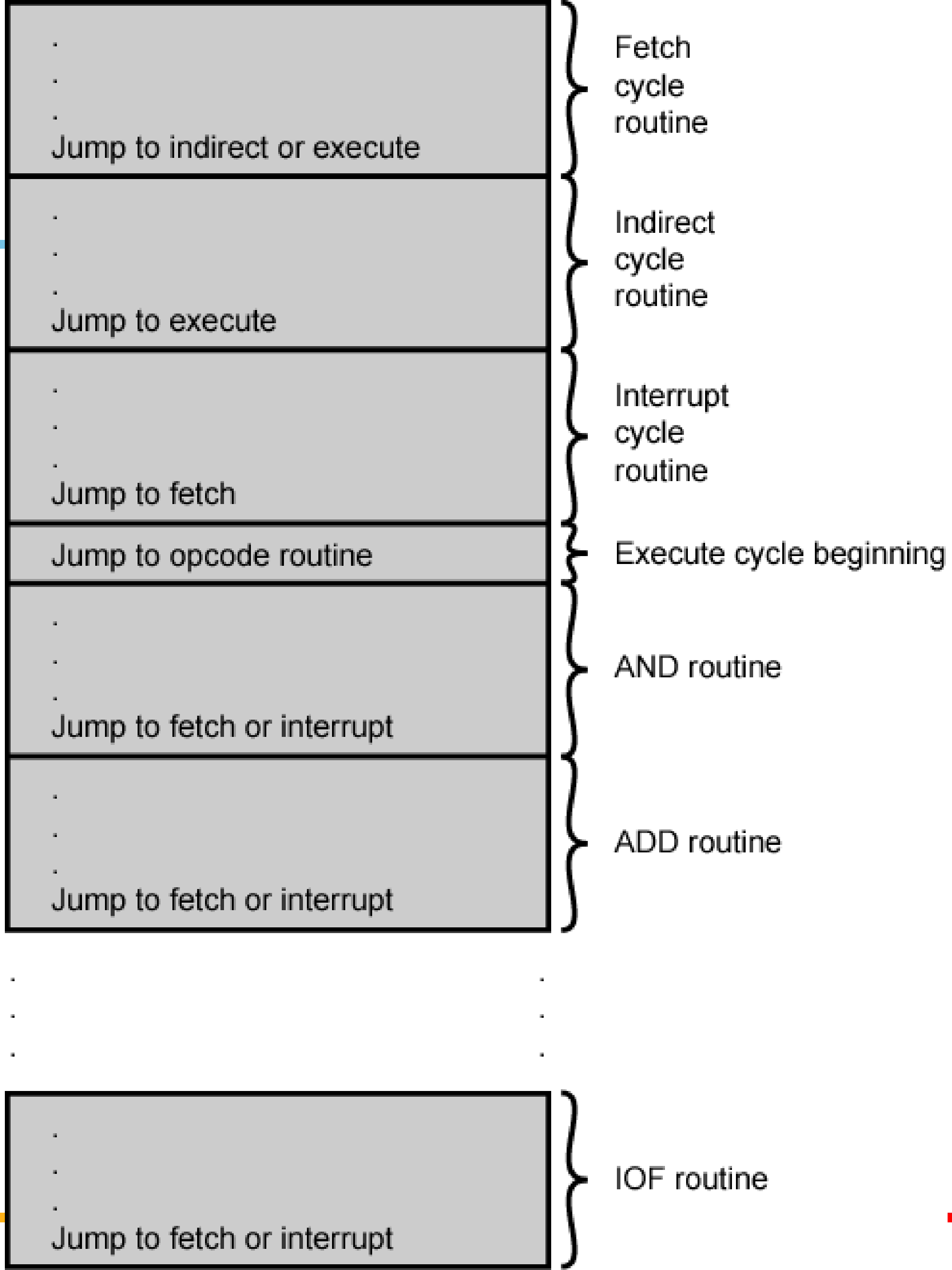
- Inputs to the control unit
  - Clock
    - One micro-instruction (or set of parallel micro-instructions) per clock cycle
  - Instruction register
    - Op-code for current instruction
    - Determines which micro-instructions are performed
  - Flags
    - State of CPU
    - Results of previous operations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



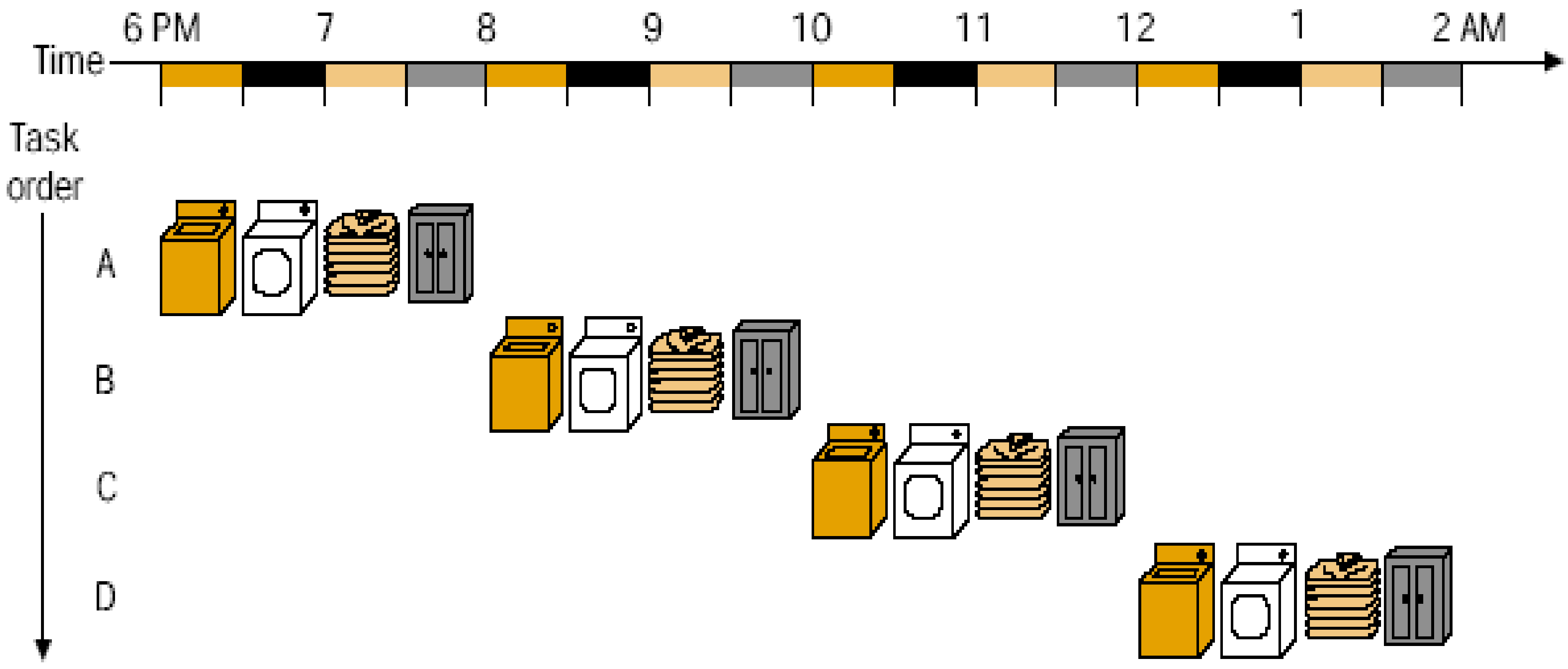
Organization of Control Memory



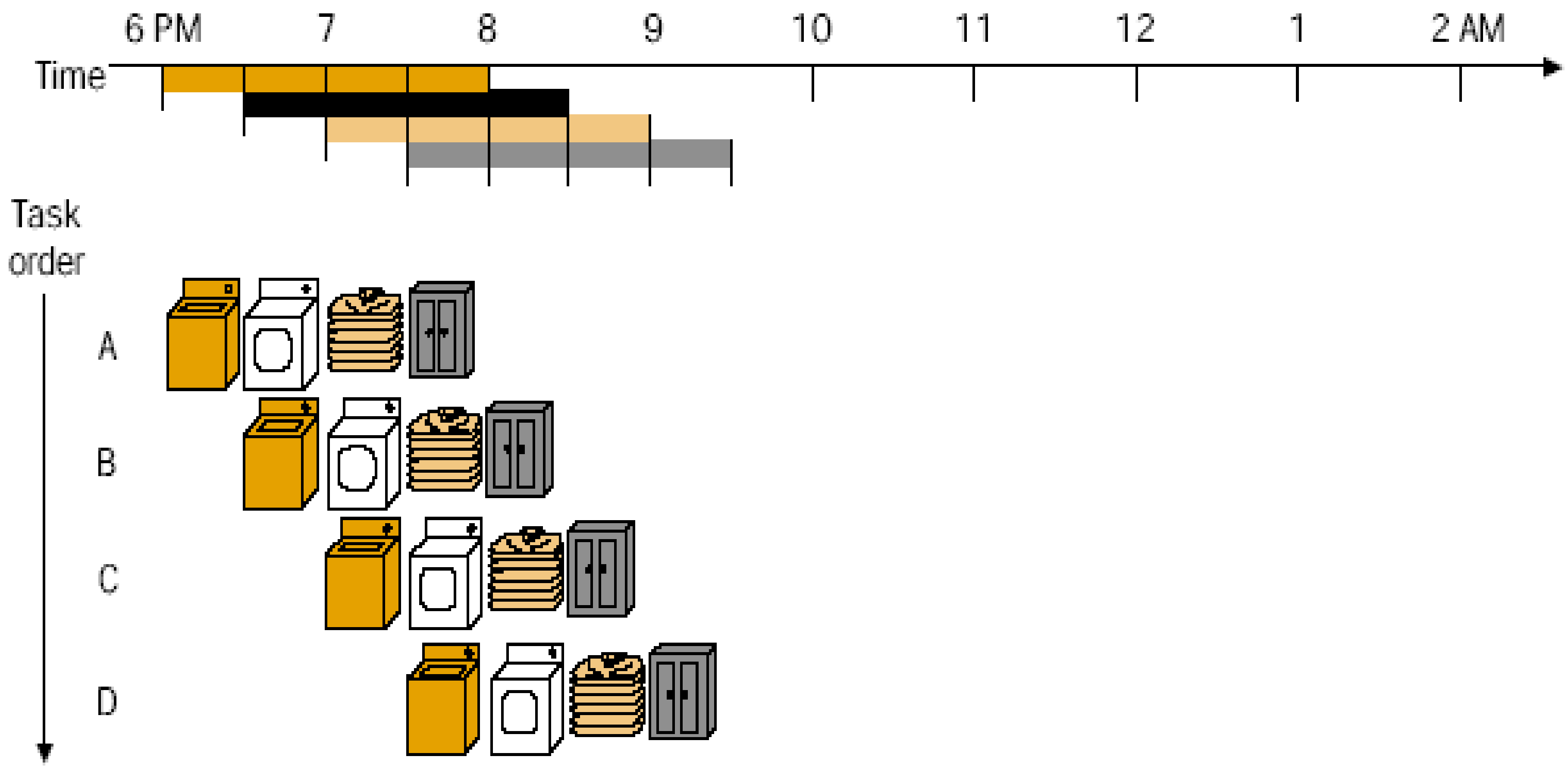
BITS Pilani
Pilani Campus

Pipeline

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

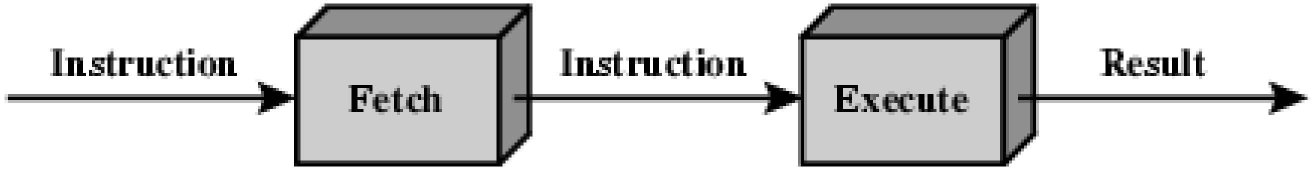
Pipelining

- 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

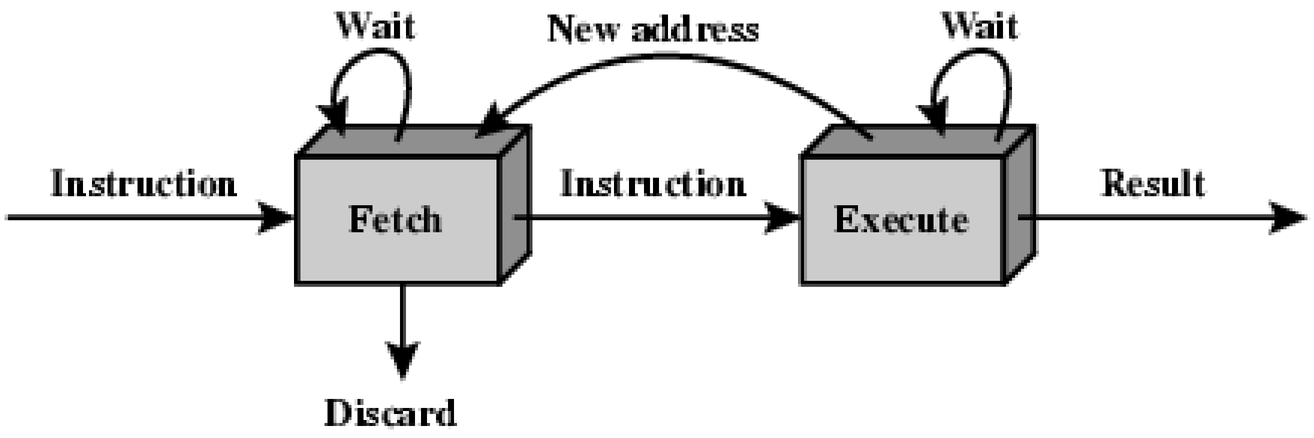
Two segment instruction pipeline

- 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

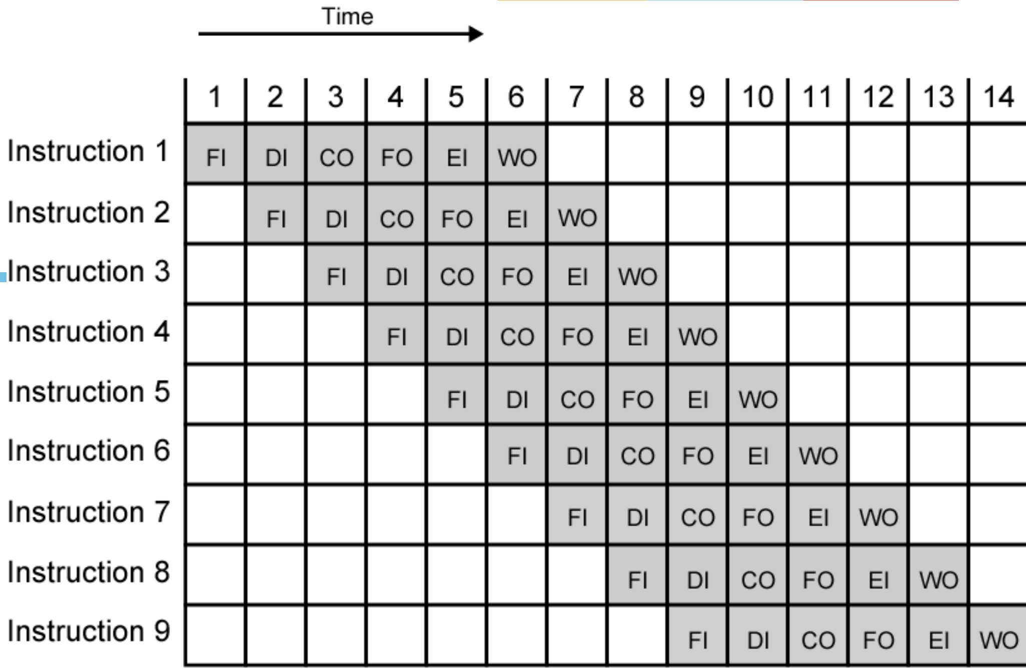
Timing Diagram for Instruction Pipeline Operation

Time →

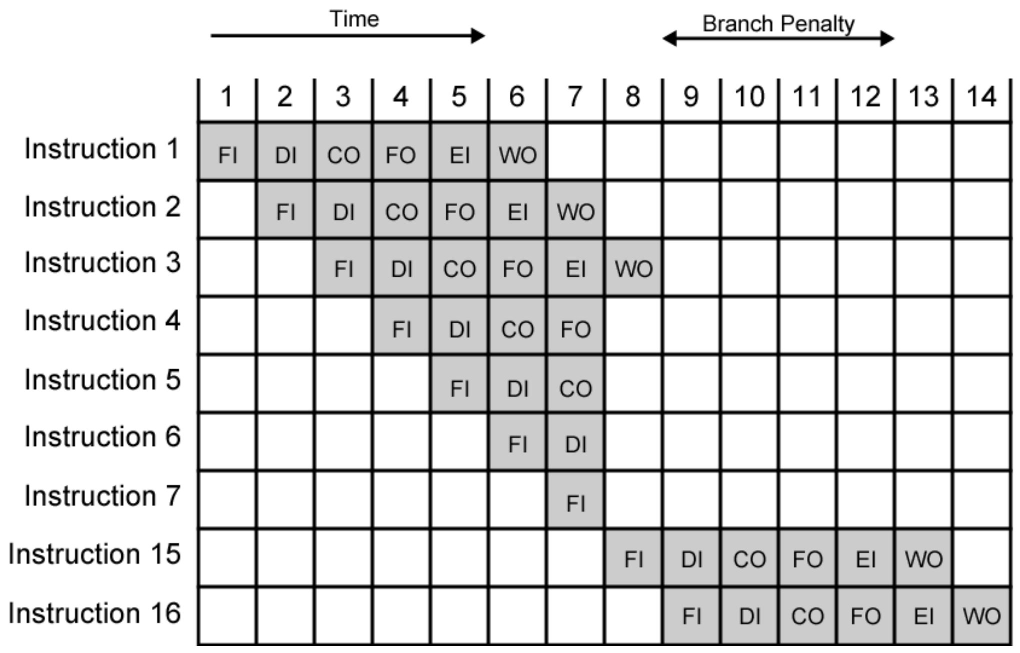
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

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

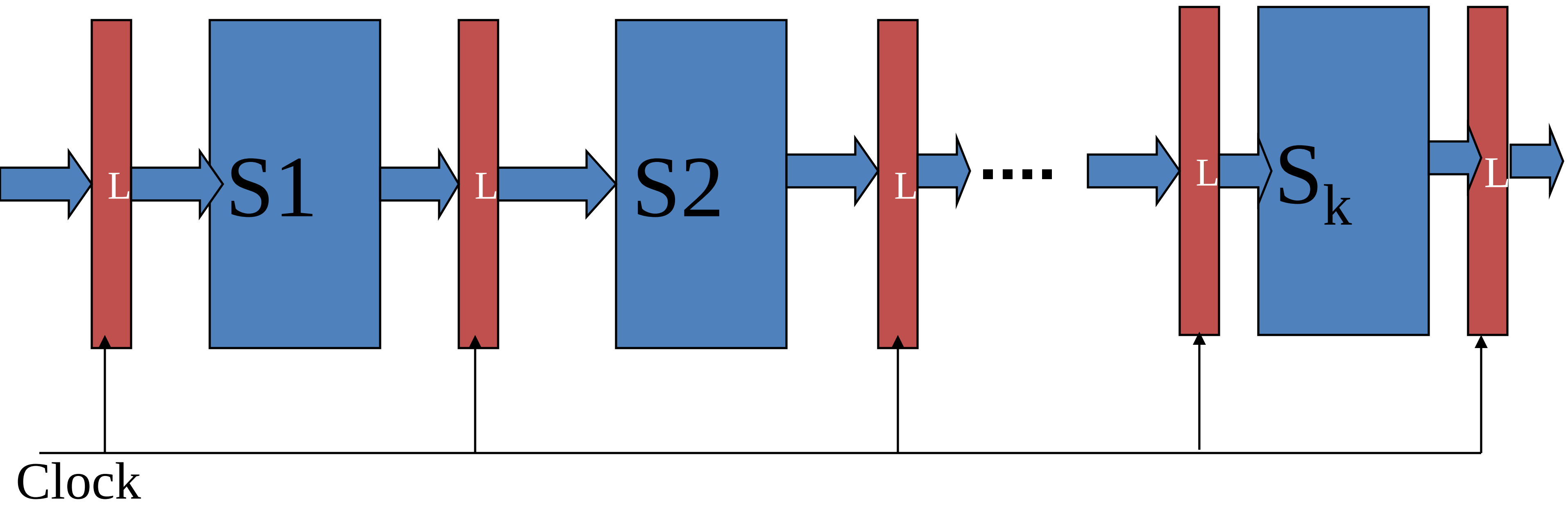


	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO


Time
Branch Penalty

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO							
Instruction 5					FI	DI	CO							
Instruction 6						FI	DI							
Instruction 7							FI							
Instruction 15								FI	DI	CO	FO	EI	WO	
Instruction 16									FI	DI	CO	FO	EI	WO

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

$$\underline{X} = \underline{A}*2^a$$
$$Y = \underline{B}*2^b$$

Instruction pipelining

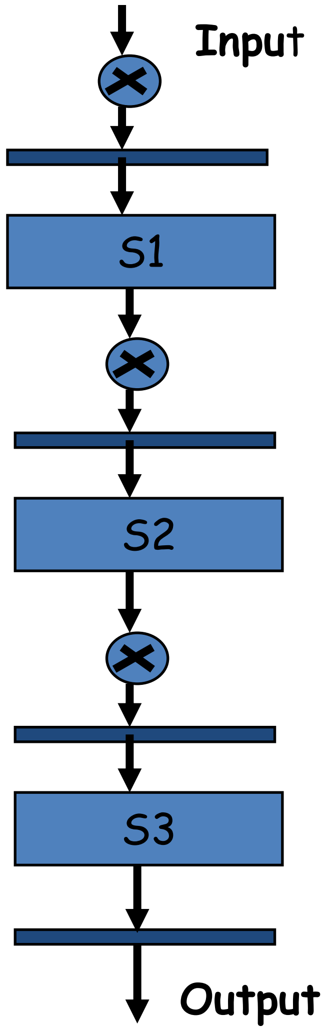
- The execution of a stream of instructions can be pipelined by overlapping the execution of the current instruction with the fetch, decode……of subsequent instructions
- Sequence of steps followed in most general purpose computer to process instruction

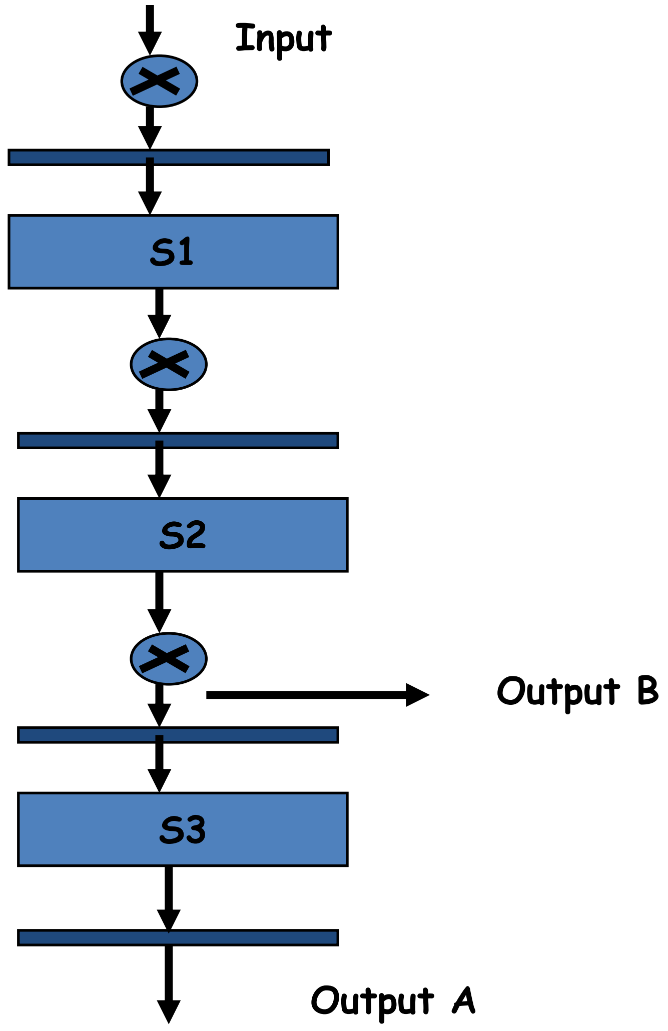
1. Fetch the instruction from memory
2. Decode the instruction
3. Calculate the effective address
4. Fetch the operands from memory
5. Execute the instruction
6. Store the result in the proper place

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

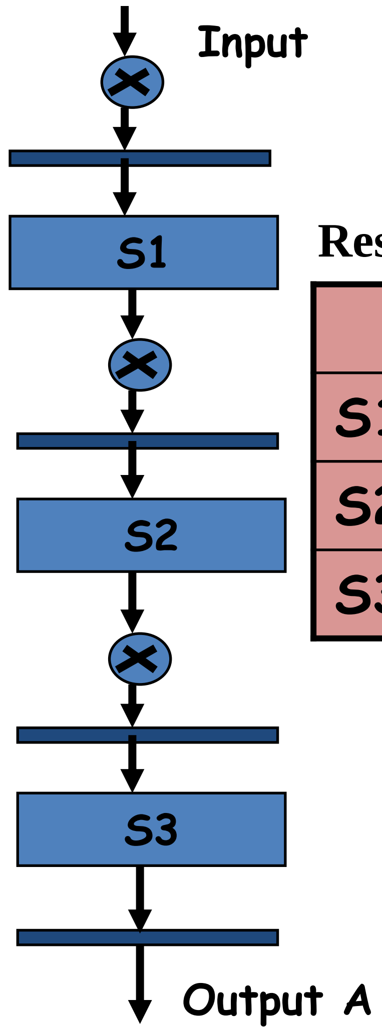




Reservation Table

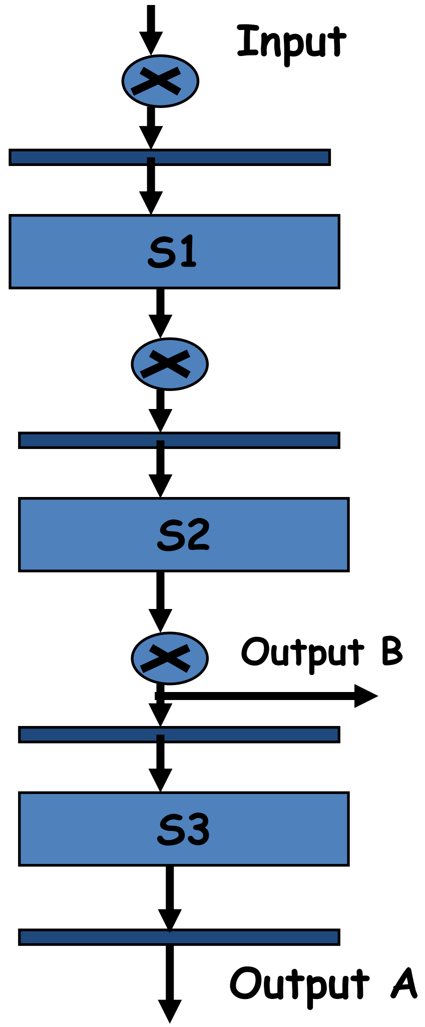
- Is a two dimensional chart
- Used to show how successive pipeline stages are utilized or reserved

Uni-function Vs Multifunction



Reservation Table A

	T0	T1	T2
S1	A		
S2		A	
S3			A



Reservation Table A

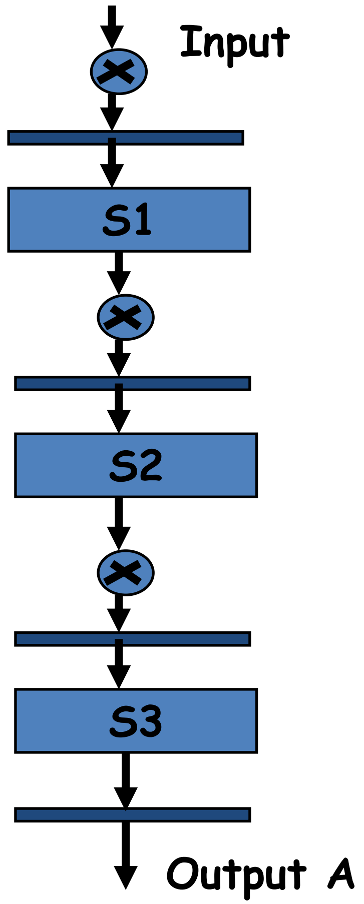
	T0	T1	T2
S1	A		
S2		A	
S3			A

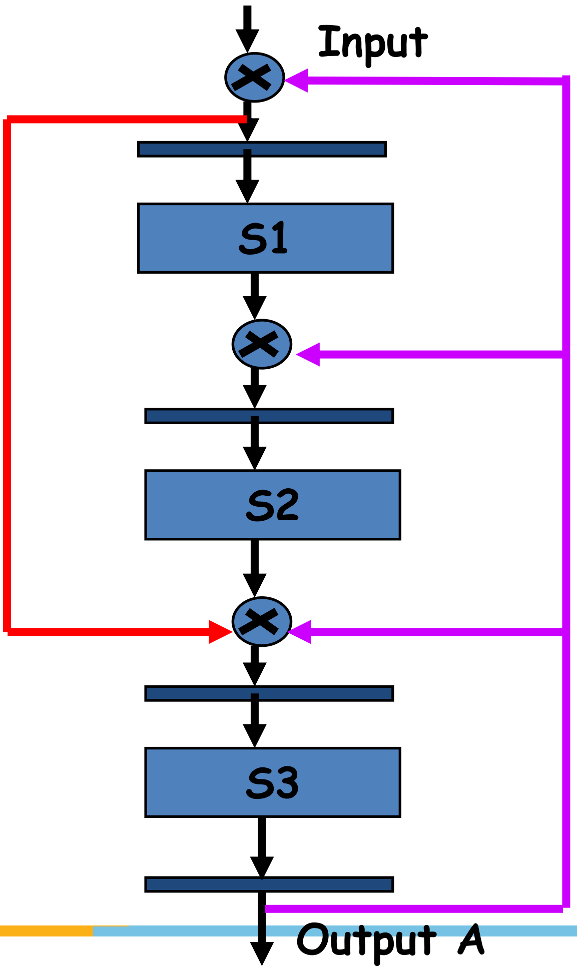
Reservation Table B

	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

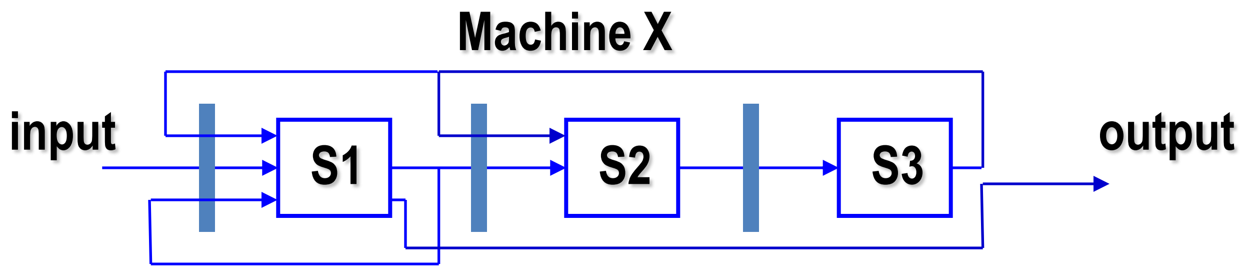




Static and Dynamic pipelining

- 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



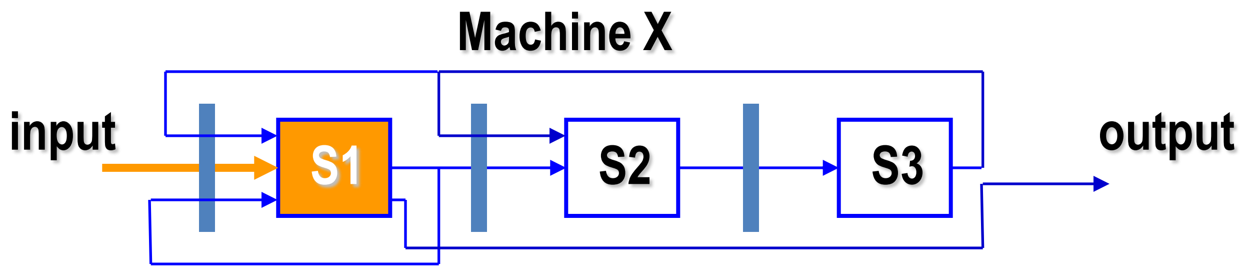
Reservation Table

Time →

Stage ↓

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

Reservation Table



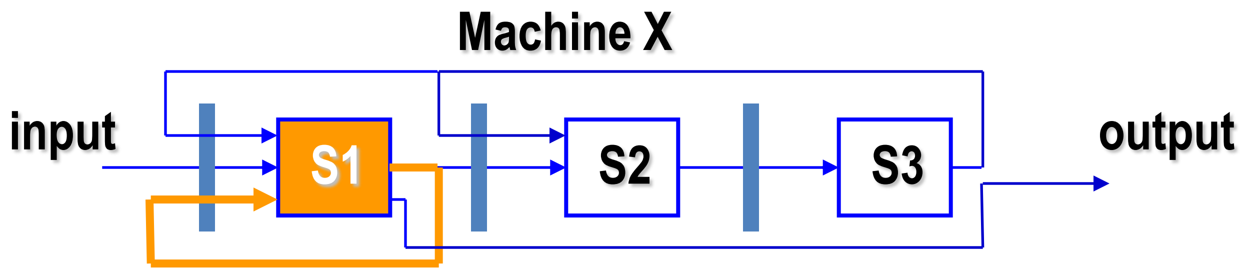
Reservation Table

Time →

Stage ↓

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

Reservation Table



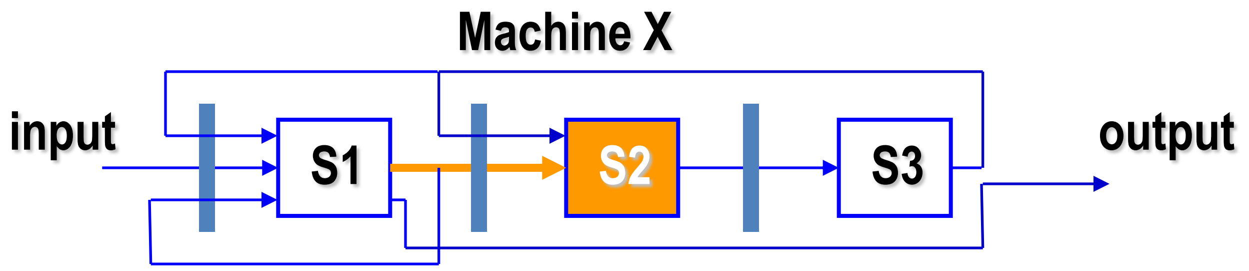
Reservation Table

Time →

Stage ↓

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

Reservation Table



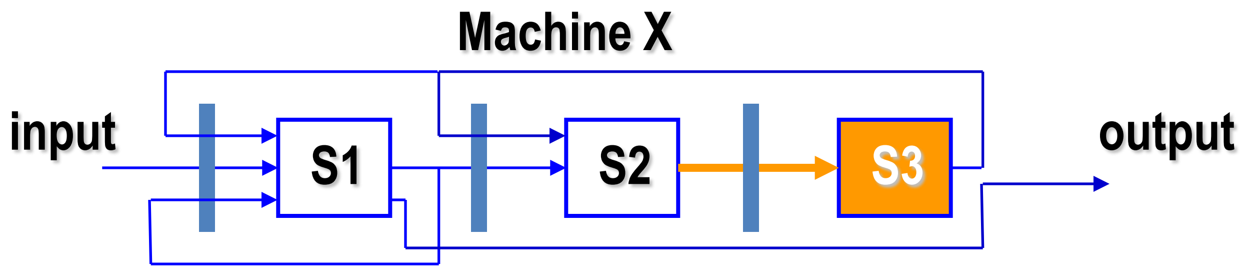
Reservation Table

Time →

Stage ↓

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

Reservation Table



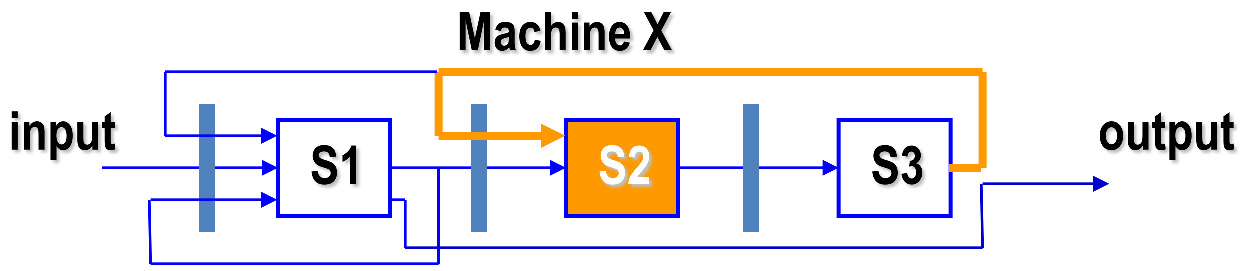
Reservation Table

Time →

Stage ↓

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

Reservation Table



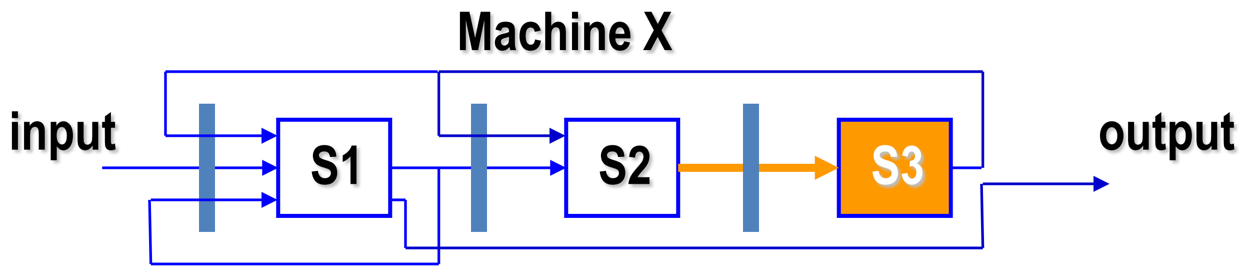
Reservation Table

Time →

Stage ↓

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

Reservation Table



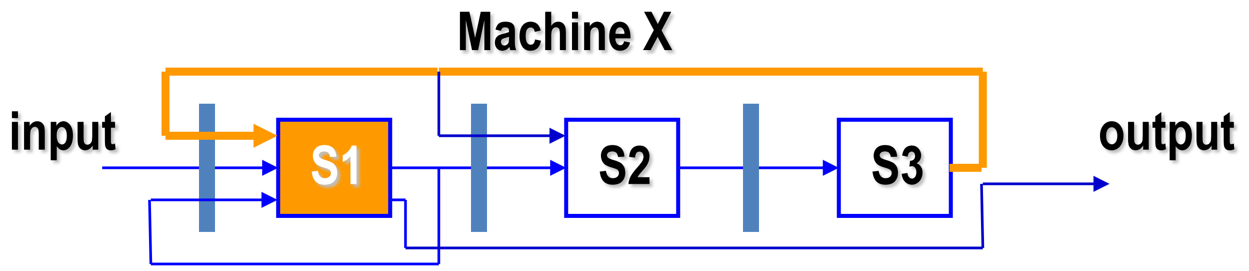
Reservation Table

Time →

Stage ↓

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

Reservation Table



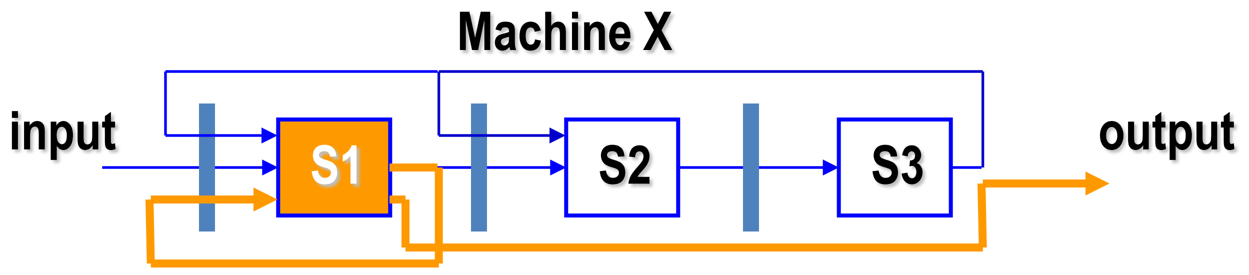
Reservation Table

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		

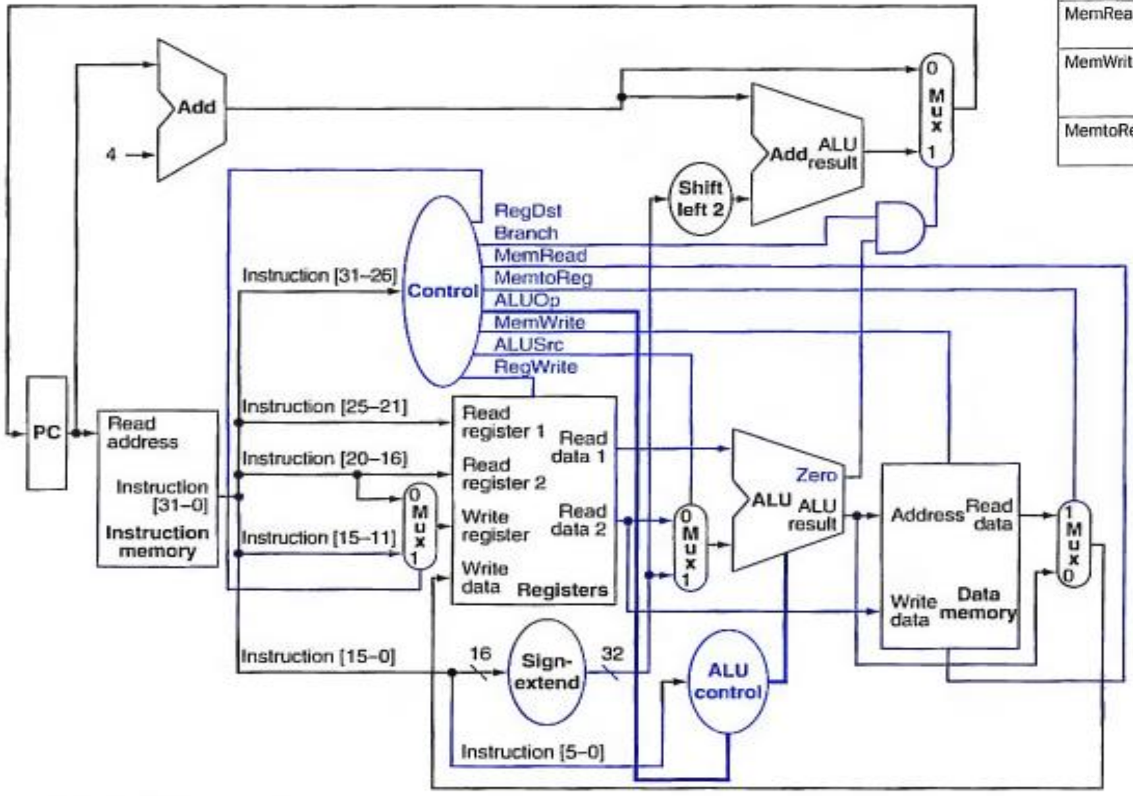
Static and Dynamic pipelining

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

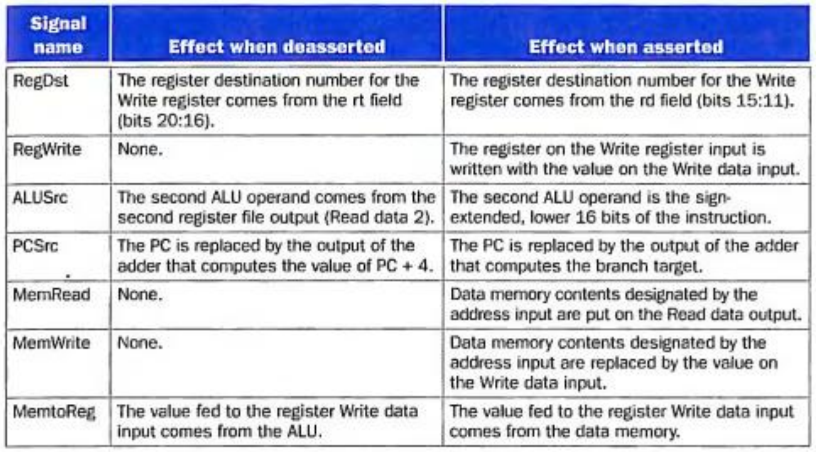
Scalar and Vector pipelining

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

ALUop	Instructions
00	lw, sw
01	Beq
10	add, sub, and, or, slt



The simple datapath with the control unit.



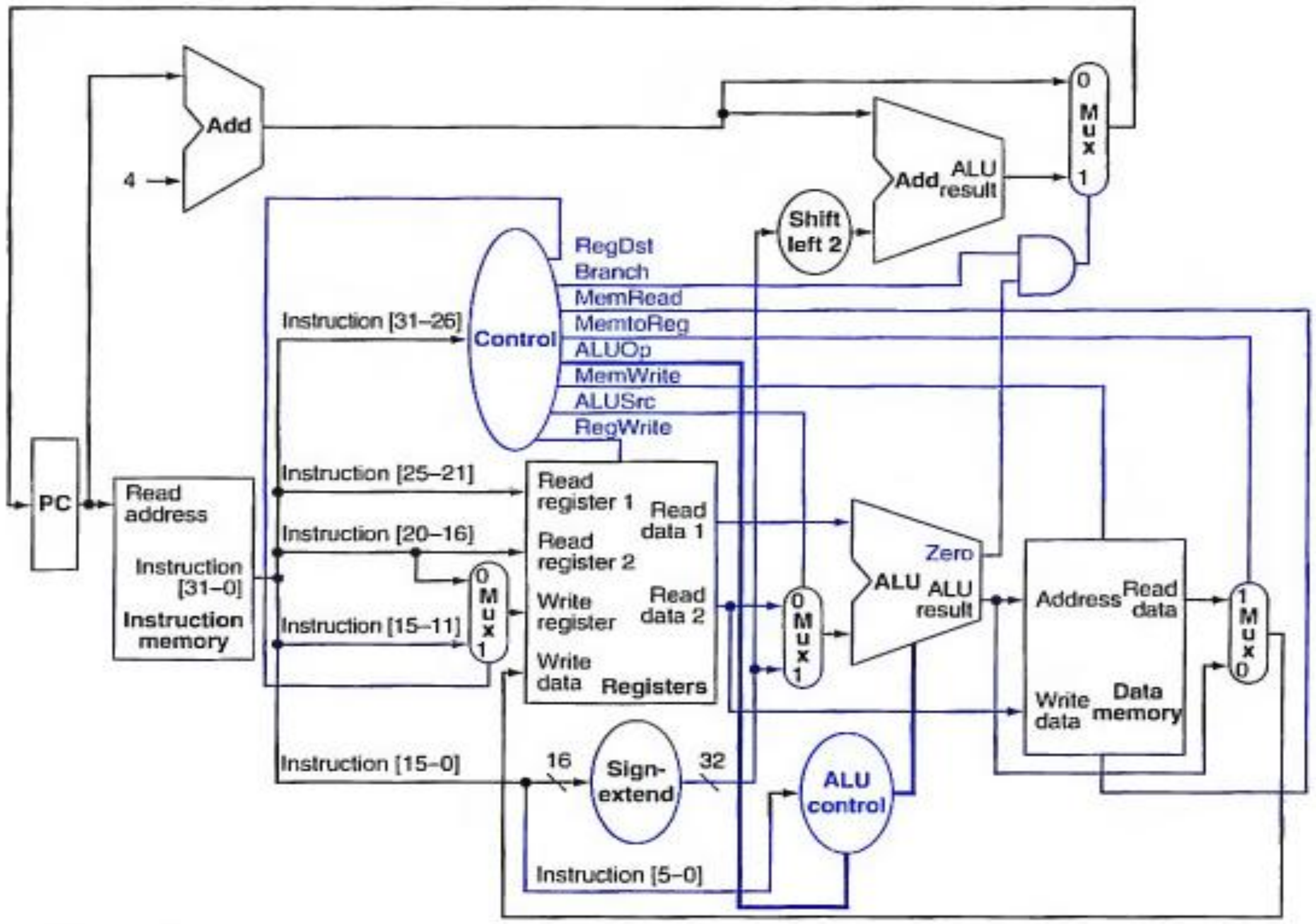
Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.

RegDst	
Branch	
MemRead	
MemtoReg	
ALUop	
MemWriteALUsrc	
RegWrite	
X: Disabled-Don't Care	

Complete Data Path

Name	Format	Example				Comments
lw	I	35	18	17	100	lw $s1, 100($s2)

RMC



The simple datapath with the control unit.