Ch 4- Processors & Pipelining

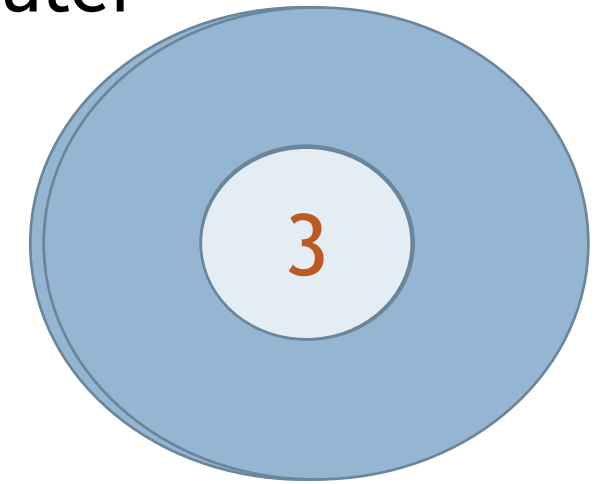
CS2400

Spring 2020

Review

- Write the main concept you learned from Chapter 1 and 2 in one or two sentence .
- Chapter 1 : Computer Abstractions and Technology
- Chapter 2 : Instructions: Language of the Computer

- You will get 3 minutes.



Introduction

- Chapter 1 (Computer Abstractions and Technology) explains that the performance of a computer is determined by three key factors:
  - Instruction count,
  - Clock cycle time
  - *Clock cycles per instruction* (CPI).

- Chapter 2 (Instructions: Language of the Computer) explains that the instruction count required for a given program is determine by
  - Compiler
  - Instruction set architecture

- The implementation of the processor determines both the clock cycle time and the number of clock cycles per instruction.

Focus on

- Data Path and Controls for implementing Processor

- The memory-reference instructions *load register* (LDR) and *store register* (STR).
- The arithmetic-logical instructions ADD, SUB, AND, and ORR.
- The instructions *compare and branch on zero* (CBZ) and branch (B).

Instruction

- For every instruction, the first two steps are identical:
- Send the program counter (PC) to the memory that contains the code and fetch the instruction from that memory.
- Read one or two registers, using fields of the instruction to select the registers to read. For the LDUR and CBZ instructions, we need to read only one register, but most other instructions require reading two registers.
- After these two steps, the actions required to complete the instruction depend on the instruction class.

Consider the first three steps for a typical LEGv8 instruction.

Fetch instruction from memory

Use ALU for address calculation

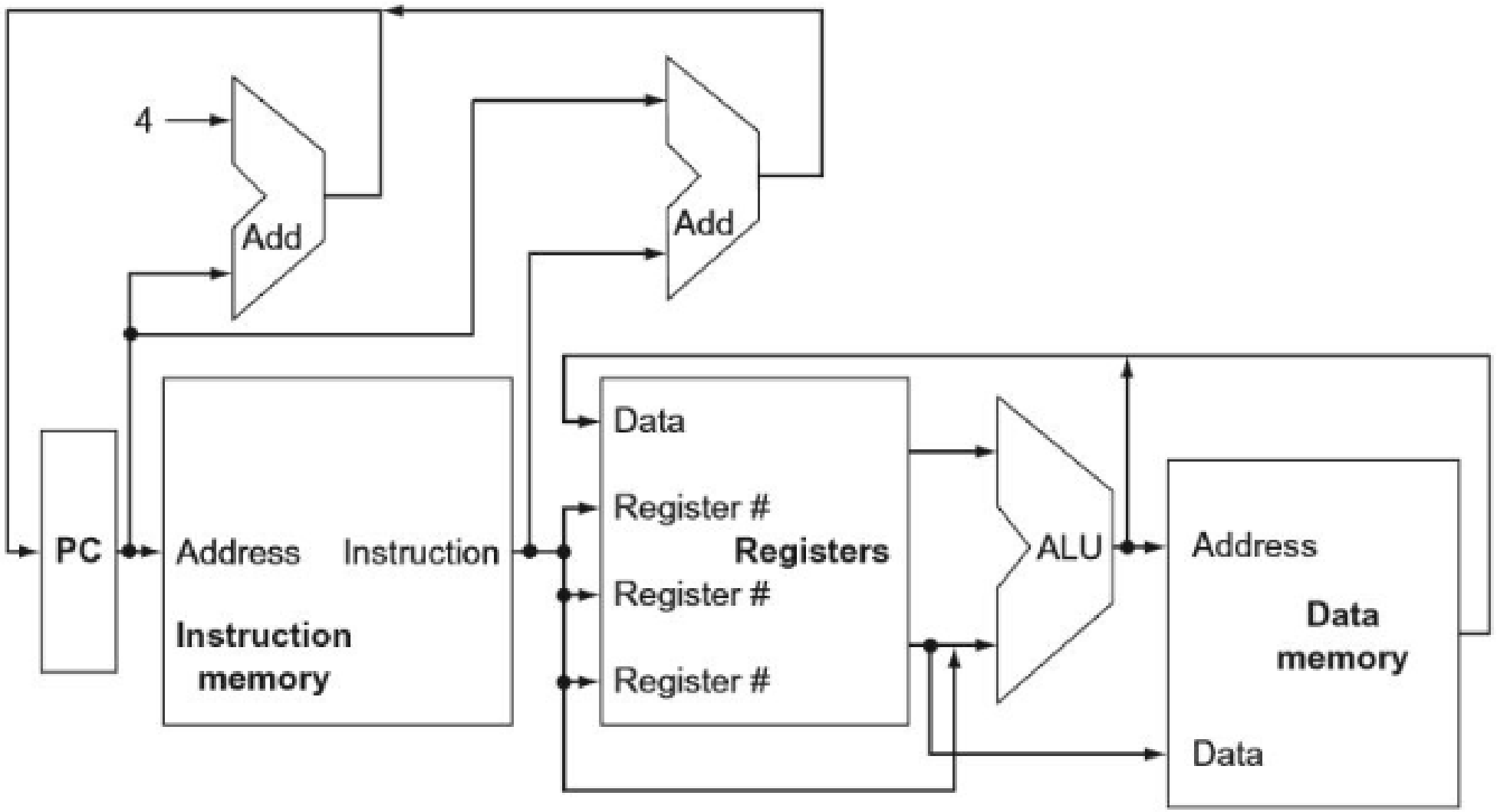
Use ALU for comparisons

Use ALU for operation execution

Read register(s)

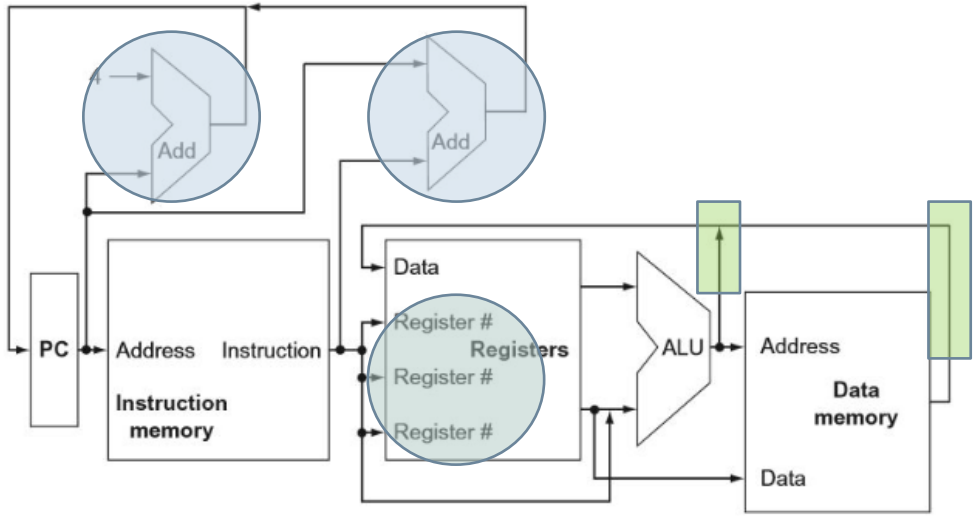
	1st step
	2nd step
	3rd step for memory-reference instructions
	3rd step for arithmetic-logic instructions
	3rd step for branch instructions

Fetch instruction from memory	1st step Every instruction must first be fetched from memory, based on the value of the program counter.
Read register(s)	2nd step Every instruction reads one or two registers.
Use ALU for address calculation	3rd step for memory-reference instructions The address calculation determines what memory address to access.
Use ALU for operation execution	3rd step for arithmetic-logic instructions The instruction's operation like add or subtract is carried out by the ALU.
Use ALU for comparisons	3rd step for branch instructions The result of the comparison determines whether the branch should be taken.


4
Add
Add
PC
Address
Instruction
Instruction memory
Data
Register #
Registers
Register #
Register #
ALU
Address
Data memory
Data

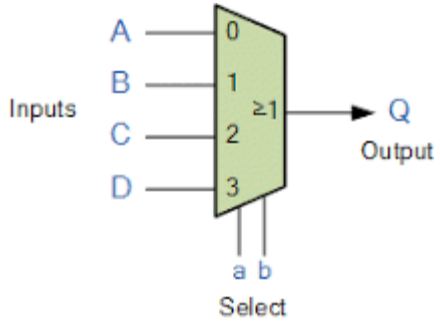
Datapath

- Data going to a particular unit as coming from two different sources.
  - The value written into the PC can come from one of two adders .
  - Data written into the register file can come from either the ALU or the data memory.
  - Second input to the ALU can come from a register or the immediate field of the instruction.

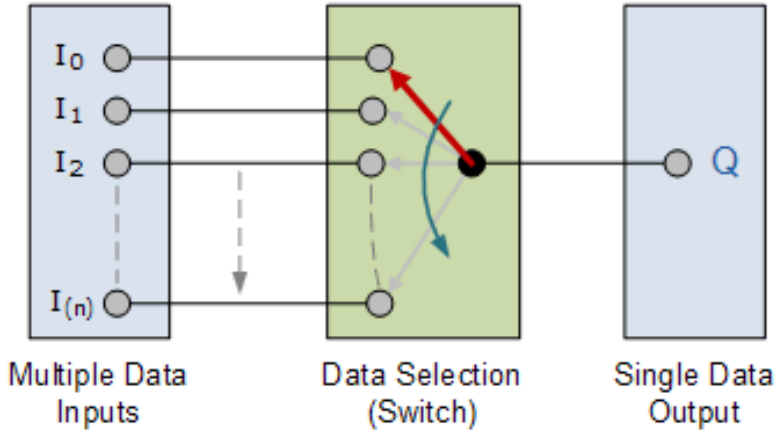


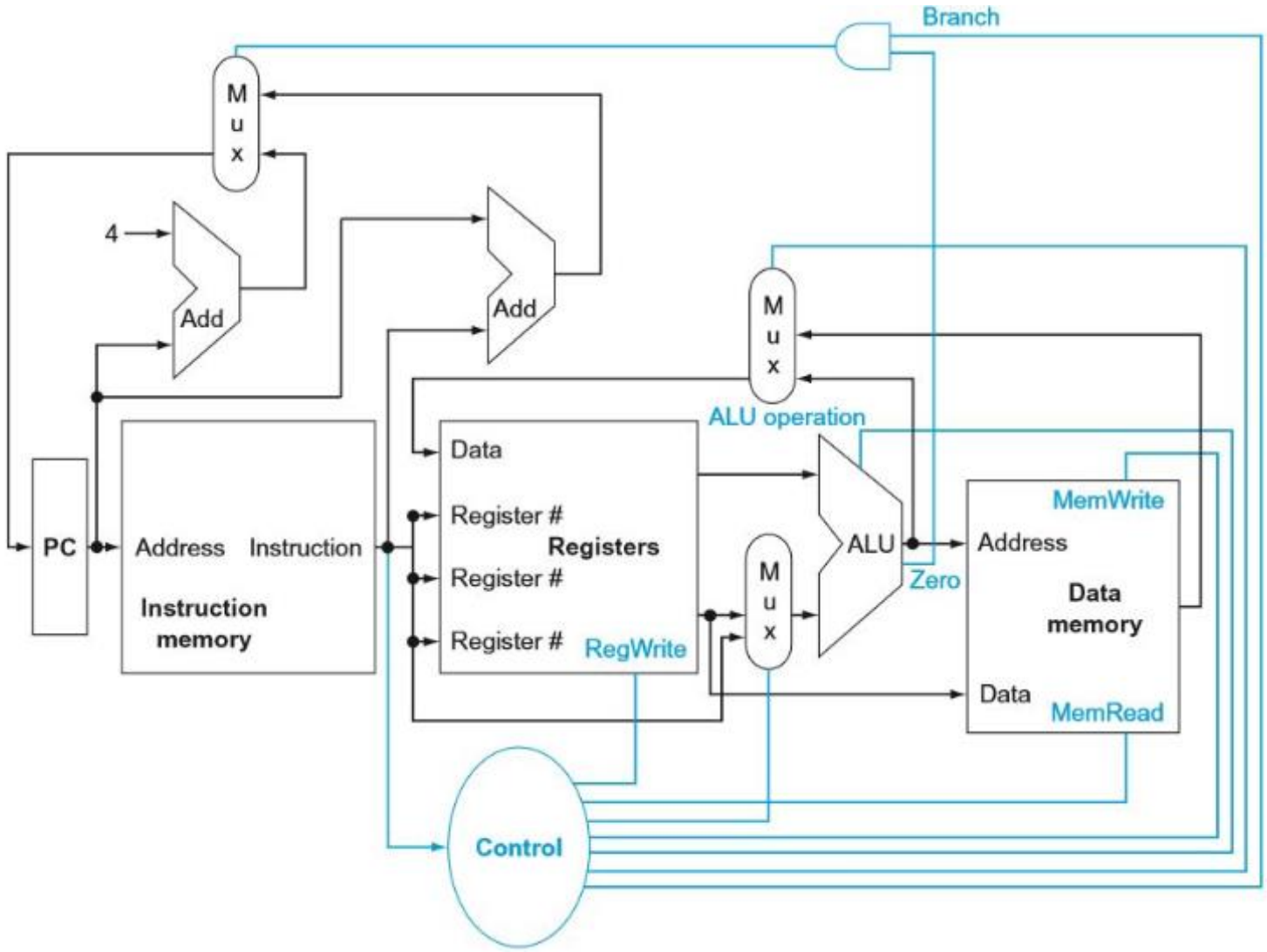
Review

- ? Which logic element can be used to choose from multiple sources.
- ? Selection
- ? Control



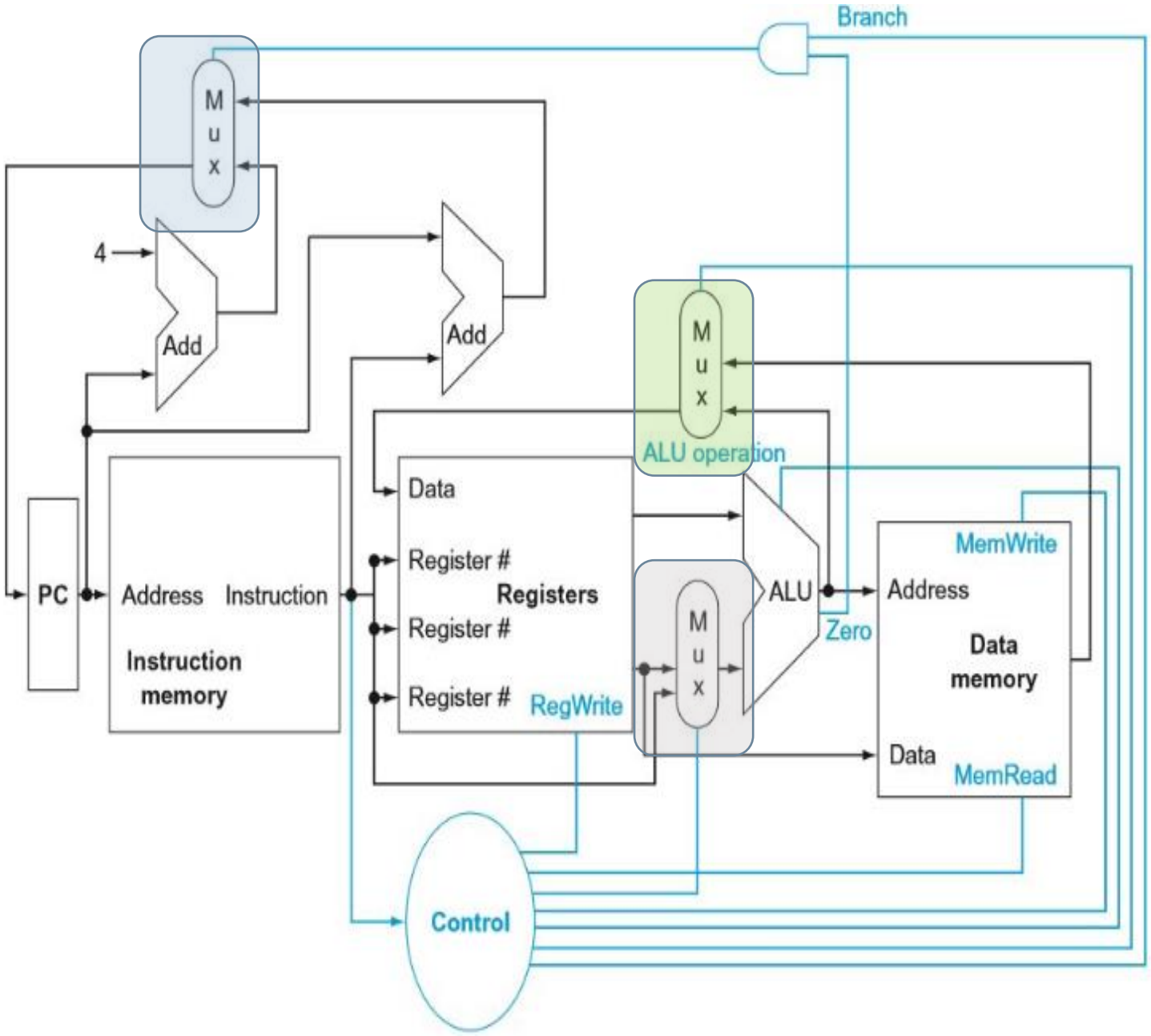
Multiplexer




Branch
M
u
x
4
Add
Add
M
u
x
ALU operation
PC
Address
Instruction
Instruction
memory
Data
Register #
Registers
Register #
Register #
RegWrite
M
u
x
ALU
Zero
MemWrite
Address
Data
memory
Data
MemRead
Control

Datapath

- What value replaces the PC (PC + 4 or the branch destination address).
- Data from ALU or the data memory.
- Second input to the ALU can come from a register or the immediate field of the instruction.



Logic design conventions

- ***Combinational element***: An operational element, that depends on current input.

- ***State/Sequential element***: An operational element, that depends on current input and output . A memory element, such as a register or a memory.

Combinational or State ?

- A datapath element whose output values depend only on the present input values is a _____ element.
- A datapath element that has internal storage is called a _____ element.
- An ALU is a _____ element.
- A register is a _____ element.
- A clock input is present on a _____ element.
- A sequential element is another name for a _____ element.

Combinational or State

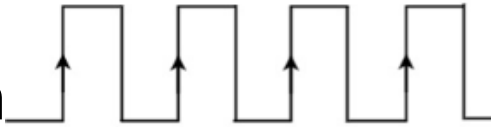
- A datapath element whose output values depend only on the present input values is called a ___C__ element.
- A datapath element that has internal storage is called a ___S__ element.
- An ALU is a __C___ element.
- A register is a __ S ___ element.
- A clock input is present on a __ S ___ element.
- A sequential element is another name for a __ S ___ element.

Clock

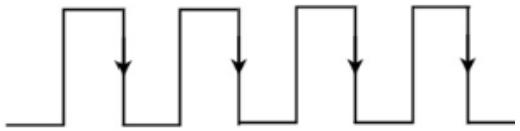
- Level triggering
  - Positive level triggering

  - Negative level triggering

- Edge triggering
  - Positive edge triggering



  - Negative edge triggering



Review

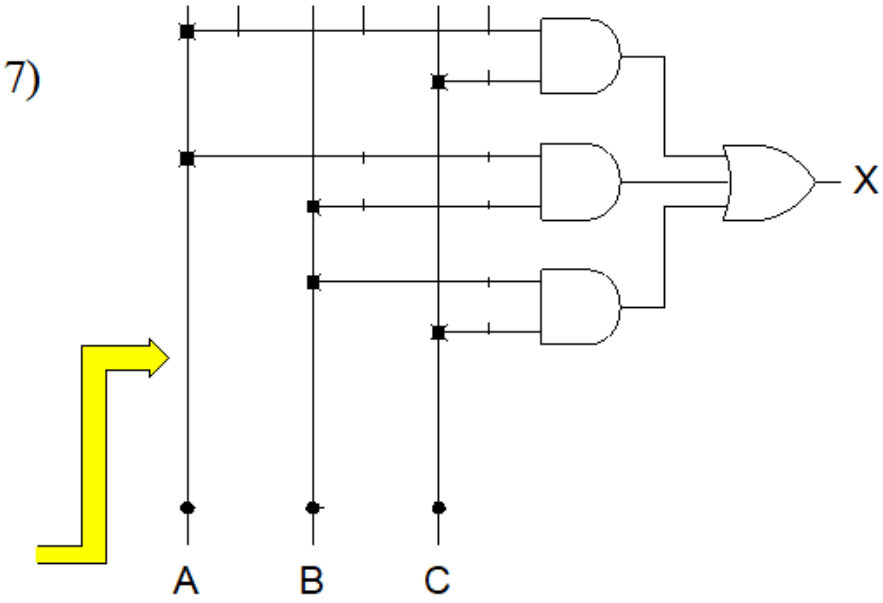
- Design a 3-input (A,B,C) digital circuit that will give at its output (X) a logic 1 only if the binary number formed at the input has more ones than zeros.

	Inputs A	B	C	Output X
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

$$X = \sum(3,5,6,7)$$

A \ BC	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$X = AC + AB + BC$$

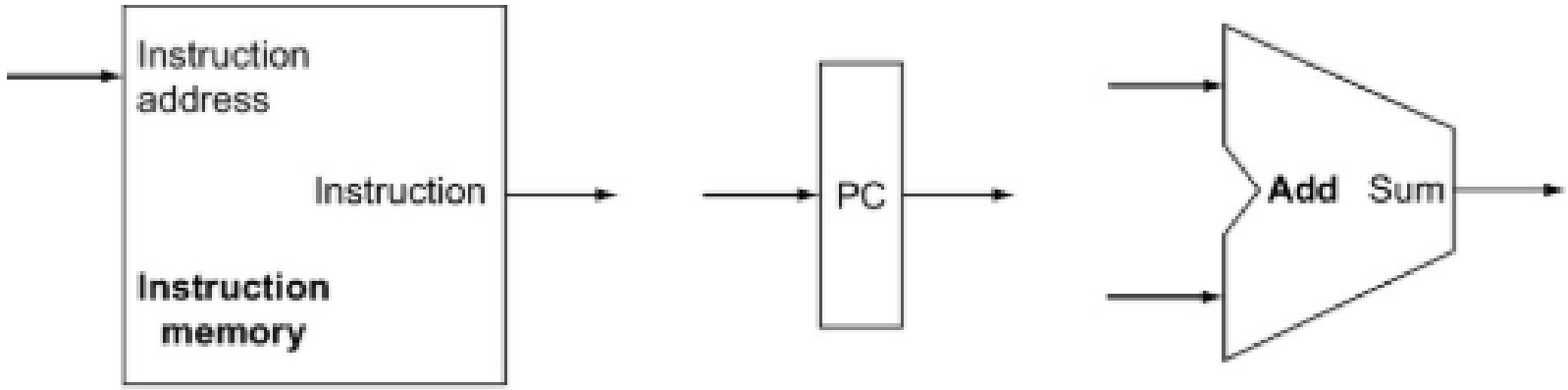


Review

- What is the basic element of a register ?
- How does it work?

Building a Datapath

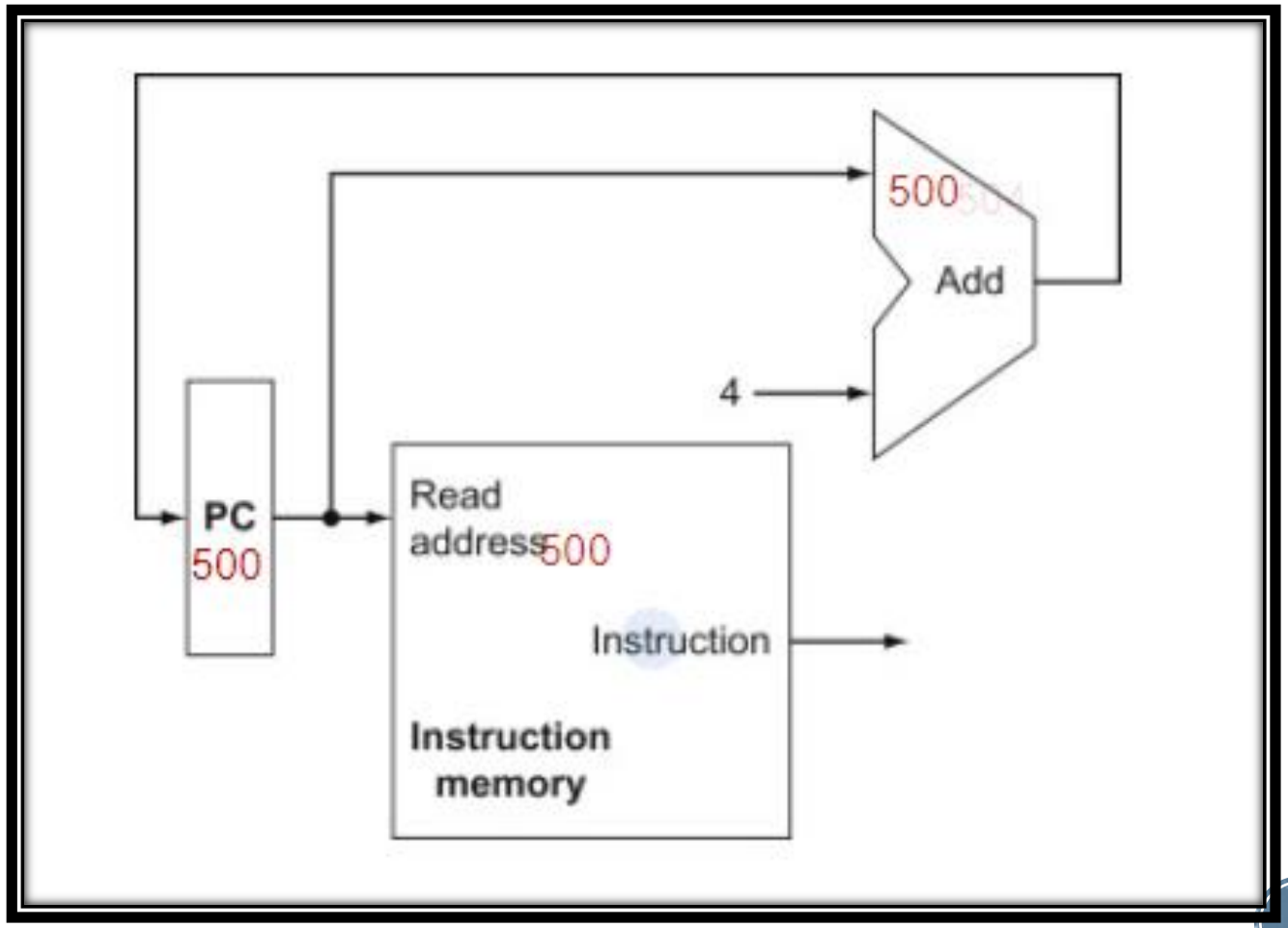
- ***Datapath element***: A unit used to operate on or hold data within a processor. The datapath elements includes the instruction and data memories, the register file, adders and the ALU .
- Below element are used to fetches instructions and increments the PC to obtain the address of the next sequential instruction.



a. Instruction memory b. Program counter c. Adder

Fetching instructions and incrementing the program counter

Program Counter (PC) : The register containing the address of the instruction in the program being executed.



R-format Instructions

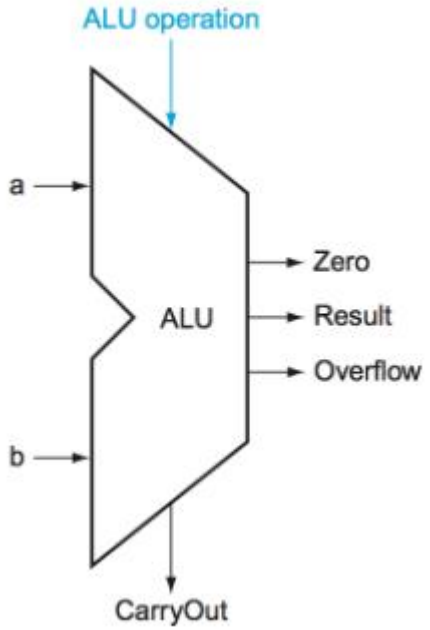
R-type instructions or arithmetic-logical instructions

- They all read two registers, perform an ALU operation on the contents of the registers, and write the result to a register.
  - Example :ADD ,SUB,AND,ORR

- The two elements needed to implement R-format ALU operations are the register file and the ALU .

ALU

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



Register File AND ALU

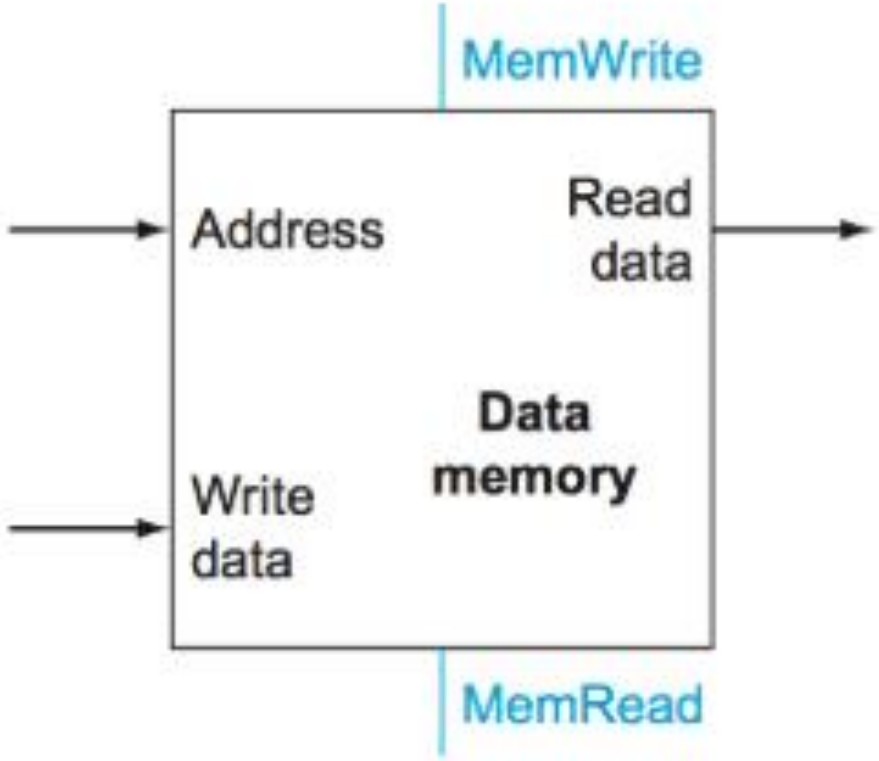
Register file: A state element that consists of a set of registers that can be read and written by supplying a register number to be accessed.

Data Path –Load /Store

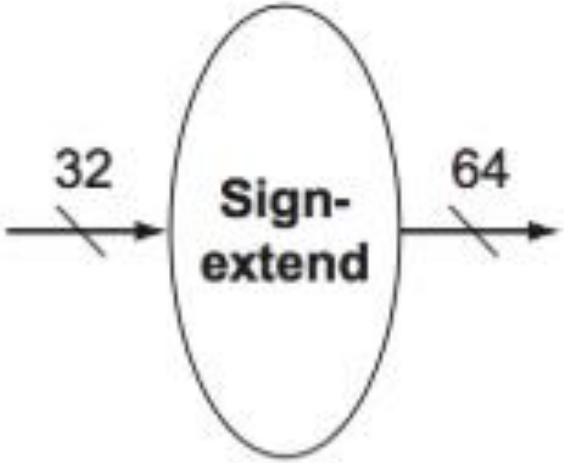
- Load register and store register instructions

```
LDR R1, [R2, offset_value]
```

- We need a unit to *sign-extend* for the offset field in the instruction to a 64-bit signed value, and a data memory unit to read from or write to.



a. Data memory unit

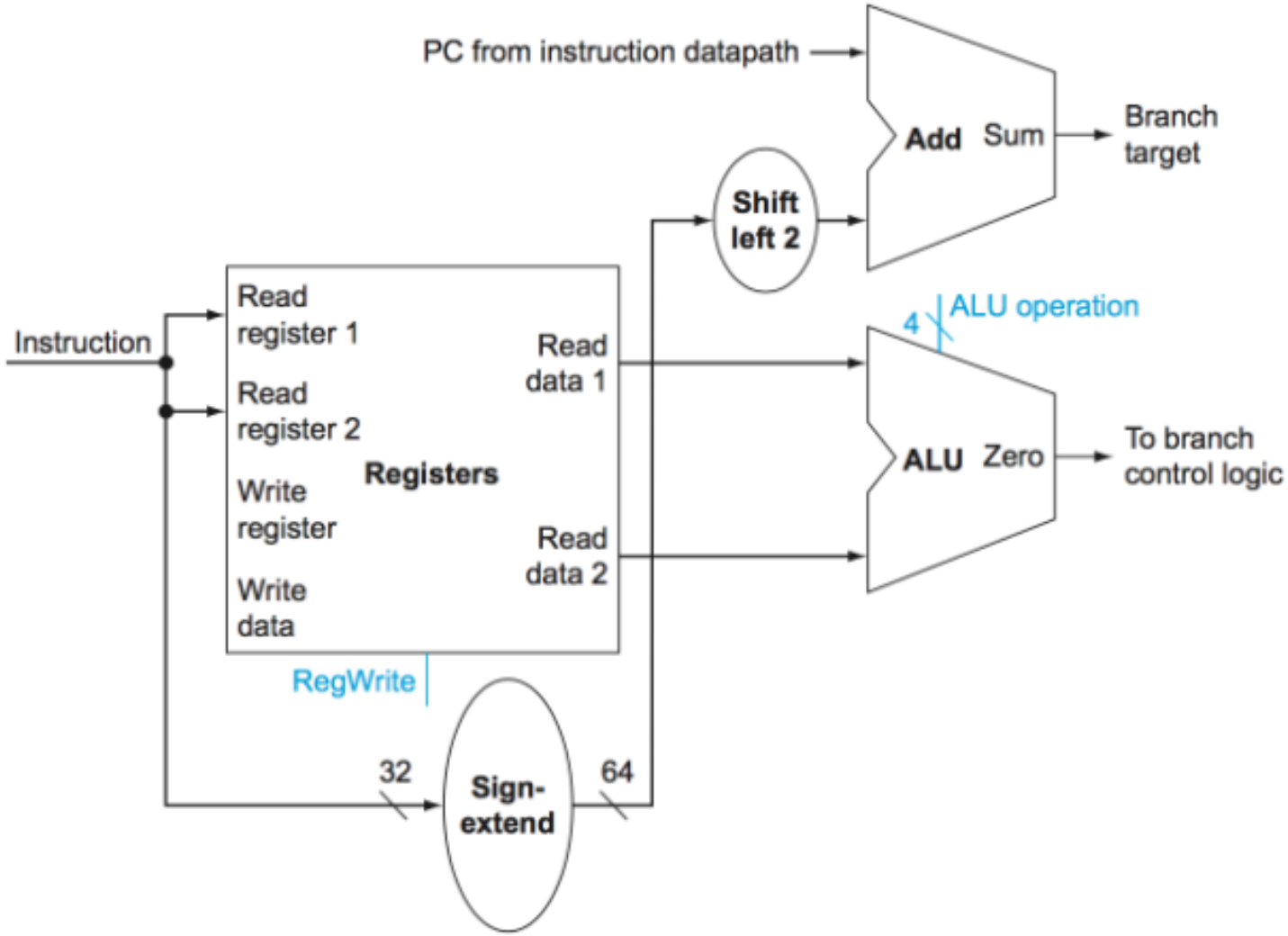


b. Sign extension unit

Data path - Branch

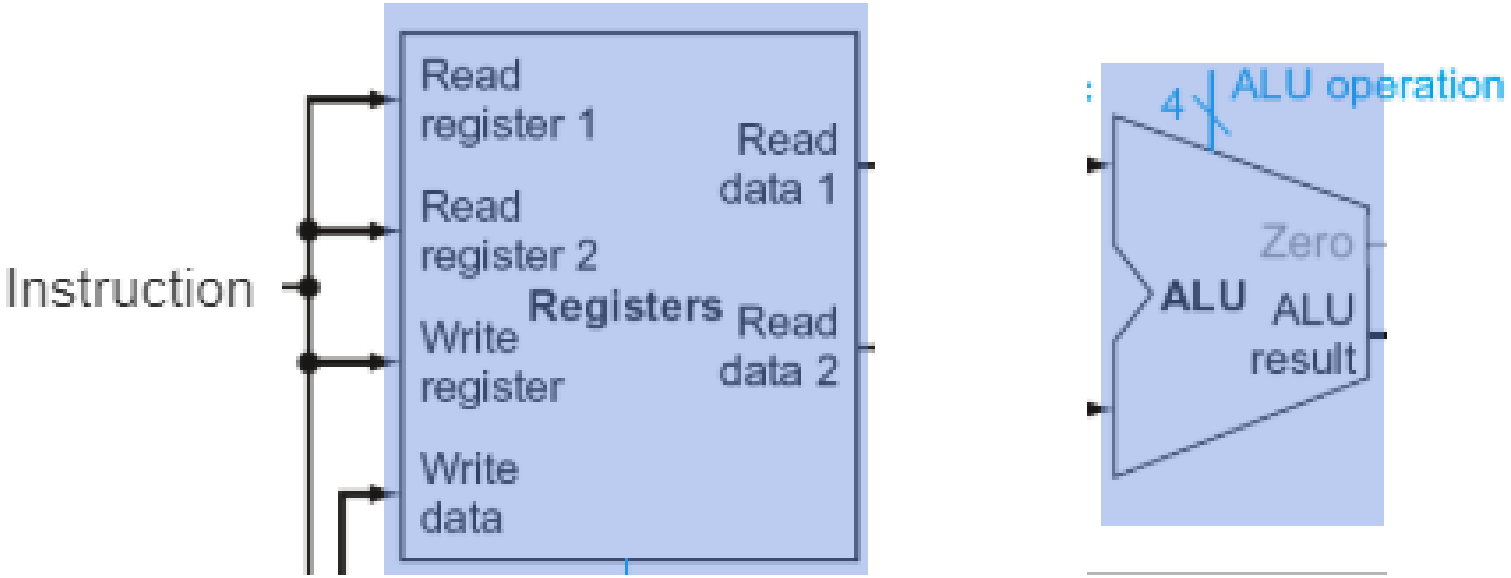
- The CBZ instruction has two operands, a register that is tested for zero, and a 19-bit offset used to compute the branch target address relative to the branch instruction address.

- ***Branch target address***: The address specified in a branch.
- ***Branch taken***: All unconditional branches.
- ***Branch not taken*** or (***untaken branch***): A branch where the branch condition is false and the program counter (PC) becomes the address of the instruction that sequentially follows the branch.

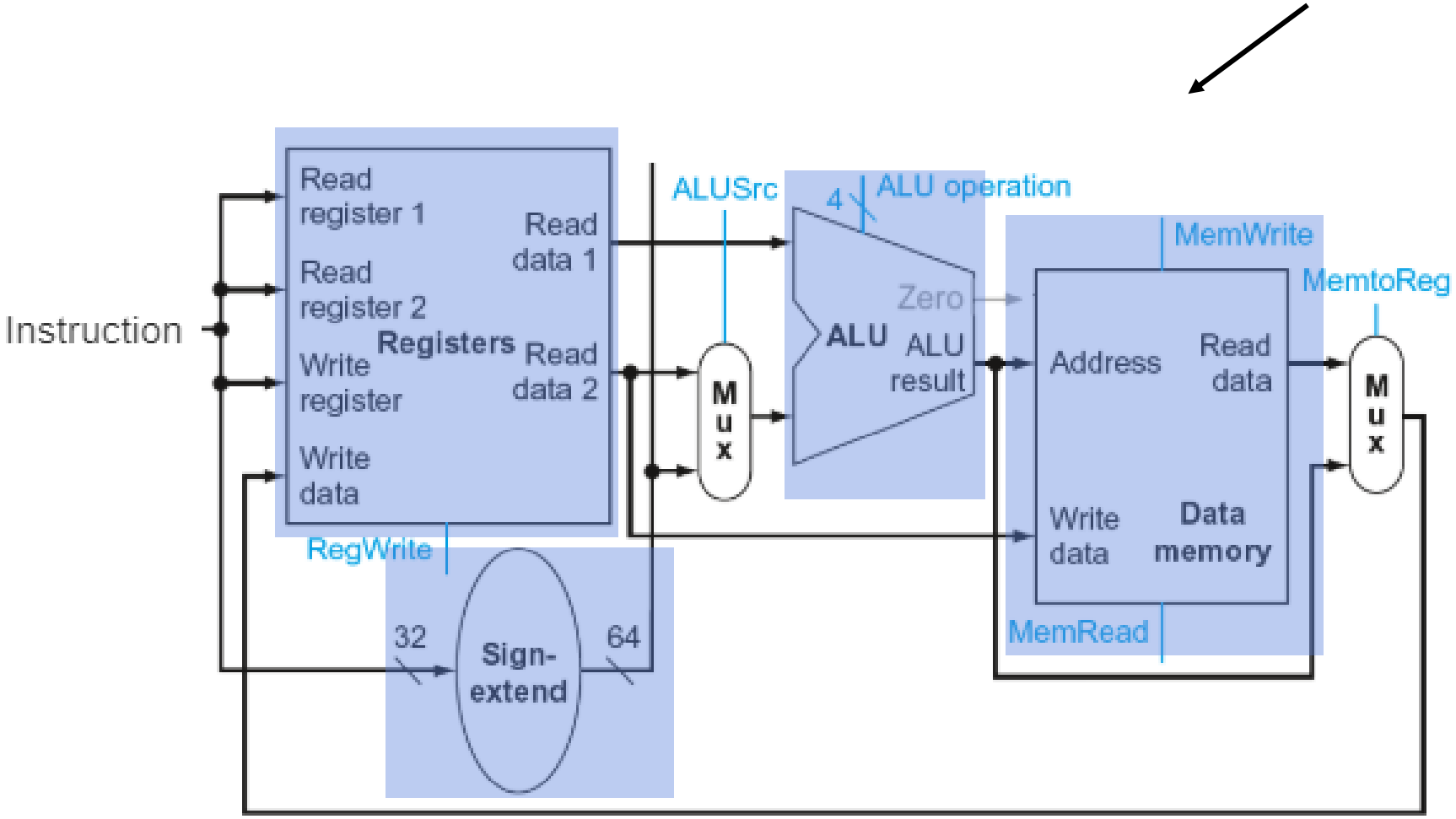

PC from instruction datapath
Add
Sum
Branch target
Shift left 2
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
4
ALU operation
ALU
Zero
To branch control logic
RegWrite
32
Sign-extend
64

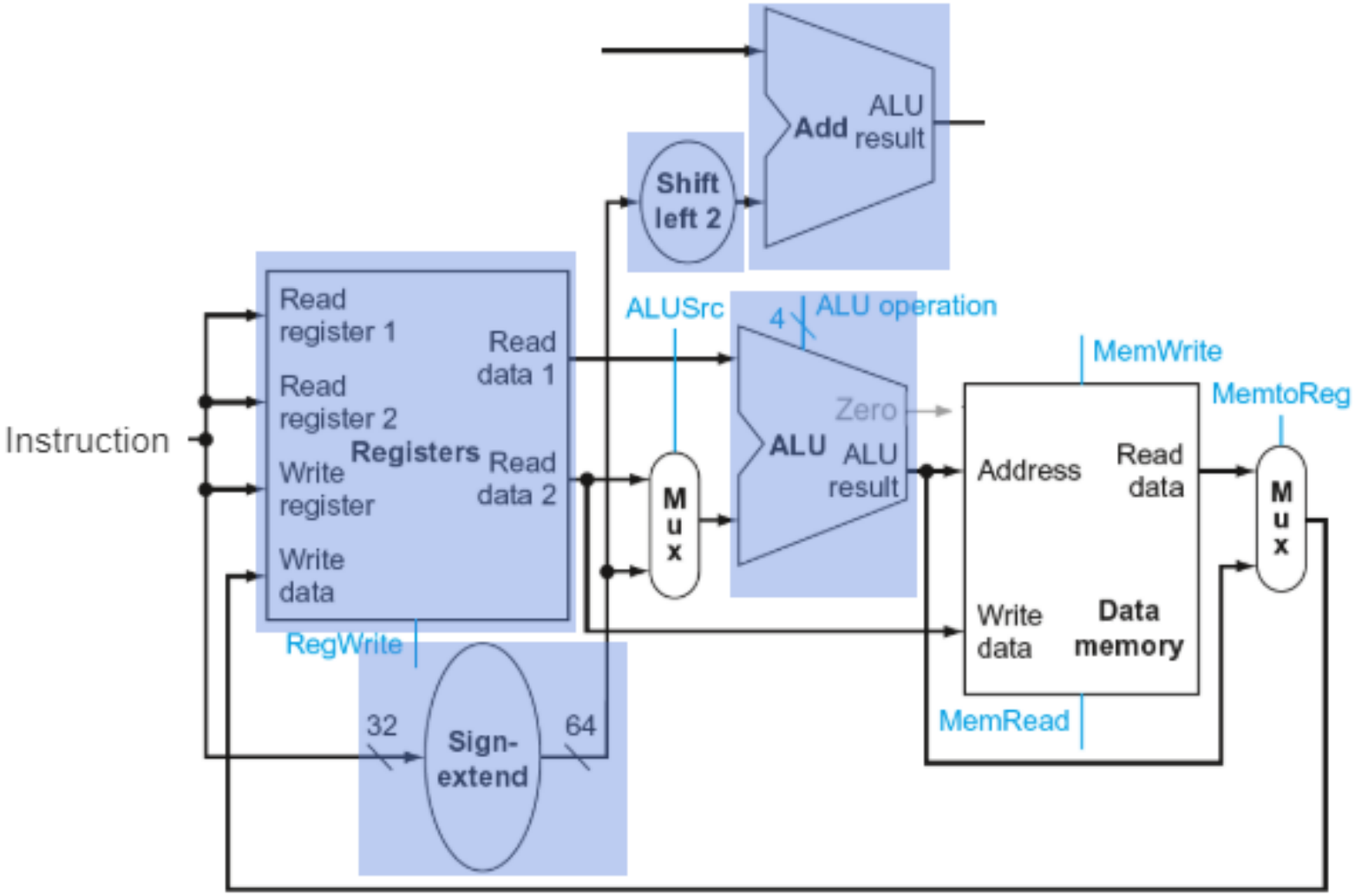
Building a Datapath

For R-format ALU operations: the register file and ALU



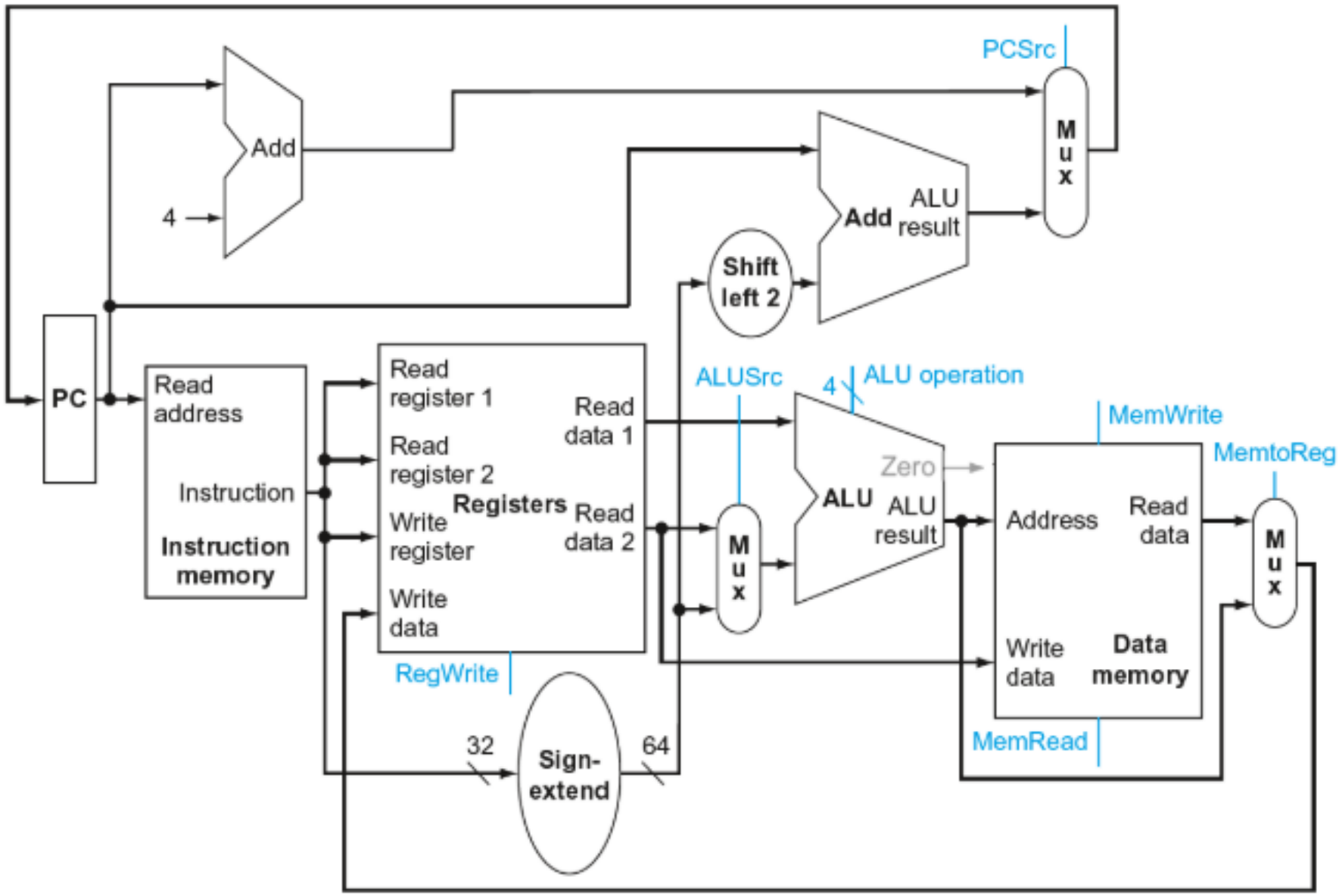
Loads and stores: register file, ALU, data memory unit, and sign extension unit.





Branch Instructions : The ALU evaluates the branch condition, and the adder computes branch target address (sign extending, then shifting left by 2).

Data path


PCSrc
Add
4
M
u
x
Add
ALU
result
Shift
left 2
PC
Read
address
Instruction
Instruction
memory
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
RegWrite
ALUSrc
4
ALU operation
MemWrite
MemtoReg
Zero
ALU
ALU
result
Address
Read
data
Write
data
Data
memory
MemRead
32
Sign-
extend
64

Find the error iN the shaded part (if any)

1) An R-type instruction like add uses three datapath units: the Register File , the ALU , and the Data Memory.

2) In addition to the register file and ALU, a load or store instruction also involves the sign extension , datamemory , and shift left units.

3) The branch datapath uses the sign extension, shift by 2 , and datamemory units.

4)The mux at the upper right either passes PC +1 (the normal case), or passes a target address from the instruction (for branches)

Solution

1) An R-type instruction like add uses three datapath units: the Register File , the ALU , and the Data Memory.

2) In addition to the register file and ALU, a load or store instruction also involves the sign extension , datamemory , and shift left units.

3) The branch datapath uses the sign extension, shift by 2 , and datamemory units.

4)The mux at the upper right either passes PC +1 (the normal case), or passes a target address from the instruction (for branches)

Instruction representation

Name	Fields							Comments
Field size		6 to 11 bits	5 to 10 bits	5 or 4 bits	2 bits	5 bits	5 bits	All LEGv8 instructions are 32 bits long
R-format	R	opcode	Rm	shamt		Rn	Rd	Arithmetic instruction format
I-format	I	opcode	immediate			Rn	Rd	Immediate format
D-format	D	opcode	address		op2	Rn	Rt	Data transfer format
B-format	B	opcode	address					Unconditional Branch format
CB-format	CB	opcode	address				Rt	Conditional Branch format

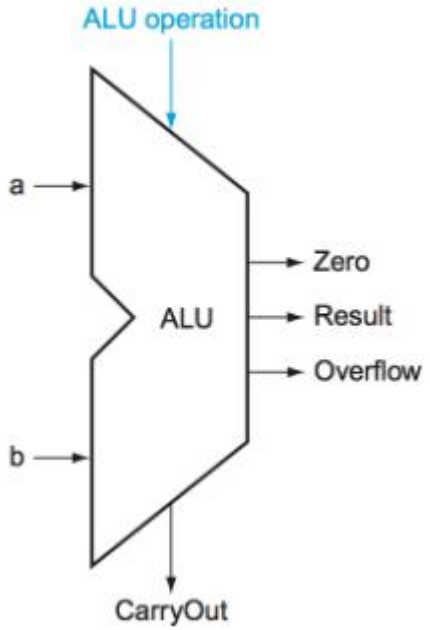
Legv Instruction

0000 = EQ
0001 = NE
0010 = HS / CS
0011 = LO / CC
0100 = MI
0101 = PL
0110 = VS
0111 = VC
1000 = HI
1001 = LS
1010 = GE
1011 = LT
1100 = GT
1101 = LE
1110 = AL
1111 = NV

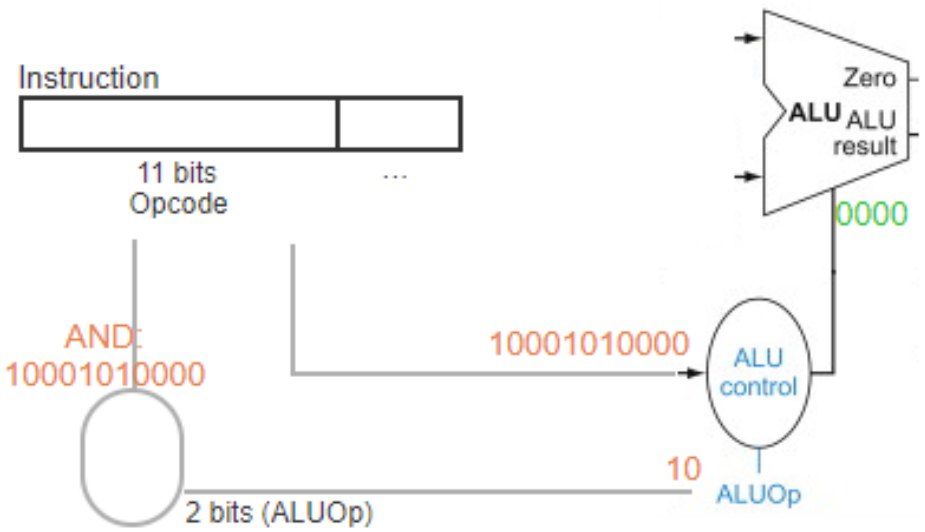
Instruction	Opcode	Opcode Size	11-bit opcode range		Instruction Format
			Start	End	
B	000101	6	160	191	B - format
STURB	00111000000	11	448		D - format
LDURB	00111000010	11	450		D - format
B.cond	01010100	8	672	679	CB - format
ORRI	1011001000	10	712	713	I - format
EORI	1101001000	10	840	841	I - format
STURH	01111000000	11	960		D - format
LDURH	01111000010	11	962		D - format
AND	10001010000	11	1104		R - format
ADD	10001011000	11	1112		R - format
ADDI	1001000100	10	1160	1161	I - format
ANDI	1001001000	10	1168	1169	I - format
BL	100101	6	1184	1215	B - format
ORR	10101010000	11	1360		R - format
ADDS	10101011000	11	1368		R - format
ADDIS	1011000100	10	1416	1417	I - format
CBZ	10110100	8	1440	1447	CB - format
CBNZ	10110101	8	1448	1455	CB - format
STURW	10111000000	11	1472		D - format
LDURSW	10111000100	11	1476		D - format
STXR	11001000000	11	1600		D - format
LDXR	11001000010	11	1602		D - format
EOR	11101010000	11	1616		R - format
SUB	11001011000	11	1624		R - format
SUBI	1101000100	10	1672	1673	I - format
MOVZ	110100101	9	1684	1687	IM - format
LSR	11010011010	11	1690		R - format
LSL	11010011011	11	1691		R - format
BR	11010110000	11	1712		R - format
ANDS	11101010000	11	1872		R - format
SUBS	11101011000	11	1880		R - format
SUBIS	1111000100	10	1928	1929	I - format
ANDIS	1111001000	10	1936	1937	I - format
MOVK	111100101	9	1940	1943	IM - format
STUR	11111000000	11	1984		D - format
LDUR	11111000010	11	1986		D - format

ALU Control

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



ALU CONTROL



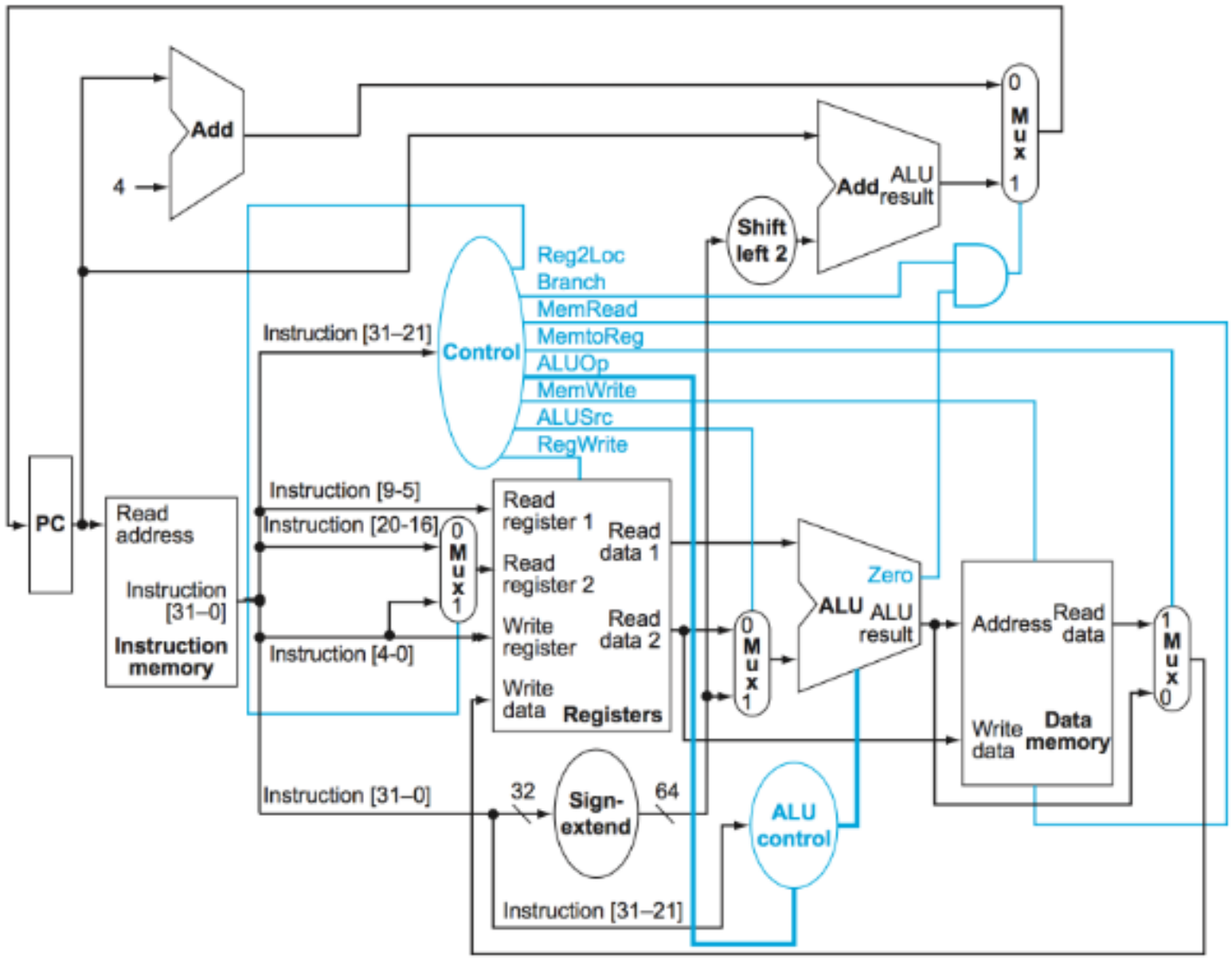
ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	pass input b
1100	NOR

Instruction	ALUOp	Instruction operation	Opcode field	Desired ALU action	ALU control input
LDUR	00	load register	XXXXXXXXXXX	add	0010
STUR	00	store register	XXXXXXXXXXX	add	0010
CBZ	01	compare and branch on zero	XXXXXXXXXXX	pass input b	0111
R-type	10	ADD	10001011000	add	0010
R-type	10	SUB	11001011000	subtract	0110
R-type	10	AND	10001010000	AND	0000
R-type	10	ORR	10101010000	OR	0001

ALU Truth Table

ALUOp		Opcode field											Operation
ALUOp1	ALUOp0	I[31]	I[30]	I[29]	I[28]	I[27]	I[26]	I[25]	I[24]	I[23]	I[22]	I[21]	
0	0	X	X	X	X	X	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	X	X	X	X	X	0111
1	X	1	0	0	0	1	0	1	1	0	0	0	0010
1	X	1	1	0	0	1	0	1	1	0	0	0	0110
1	X	1	0	0	0	1	0	1	0	0	0	0	0000
1	X	1	0	1	0	1	0	1	0	0	0	0	0001

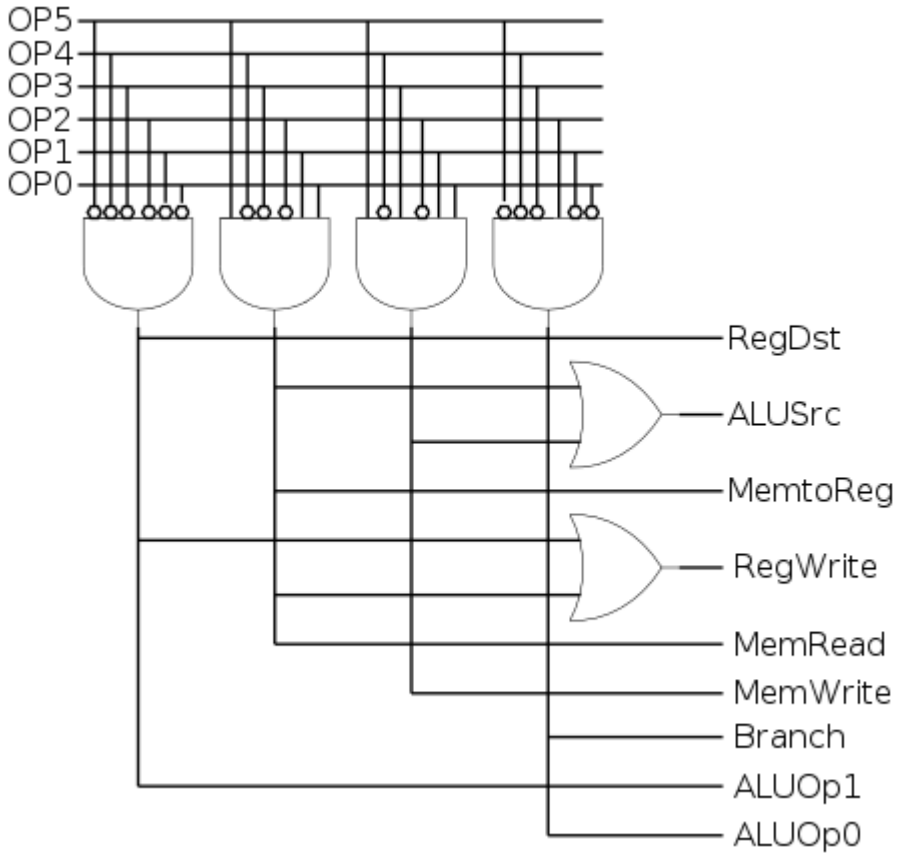
Data Path with Control



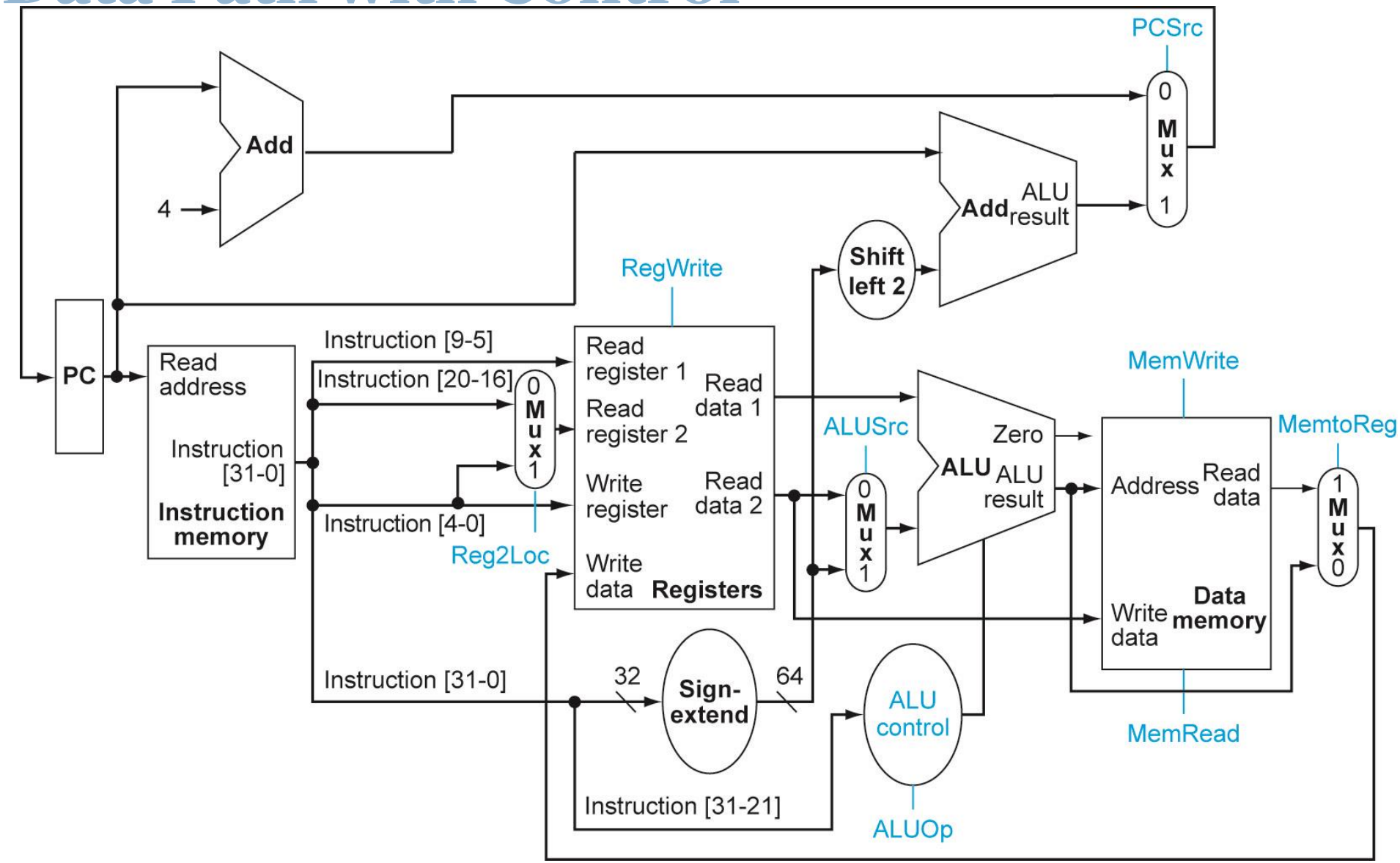
Classwork

- Convert ADD R1,R2,R3 into 32 bit binary format and draw the flow of instruction as bits in the Datapath.

6 bit control unit



Data Path with Control



Control Signals - Q1 in worksheet

Signal	Function When asserted	Function When deasserted
Reg2Loc	The register number for Read register 2 comes from the Rt field(bits 4:0)	The register number for Read register 2 comes from the Rm field(bits 20:16)
AluSrc		
MemtoReg		
RegWrite		
MemRead		
MemWrite		
PC Src		

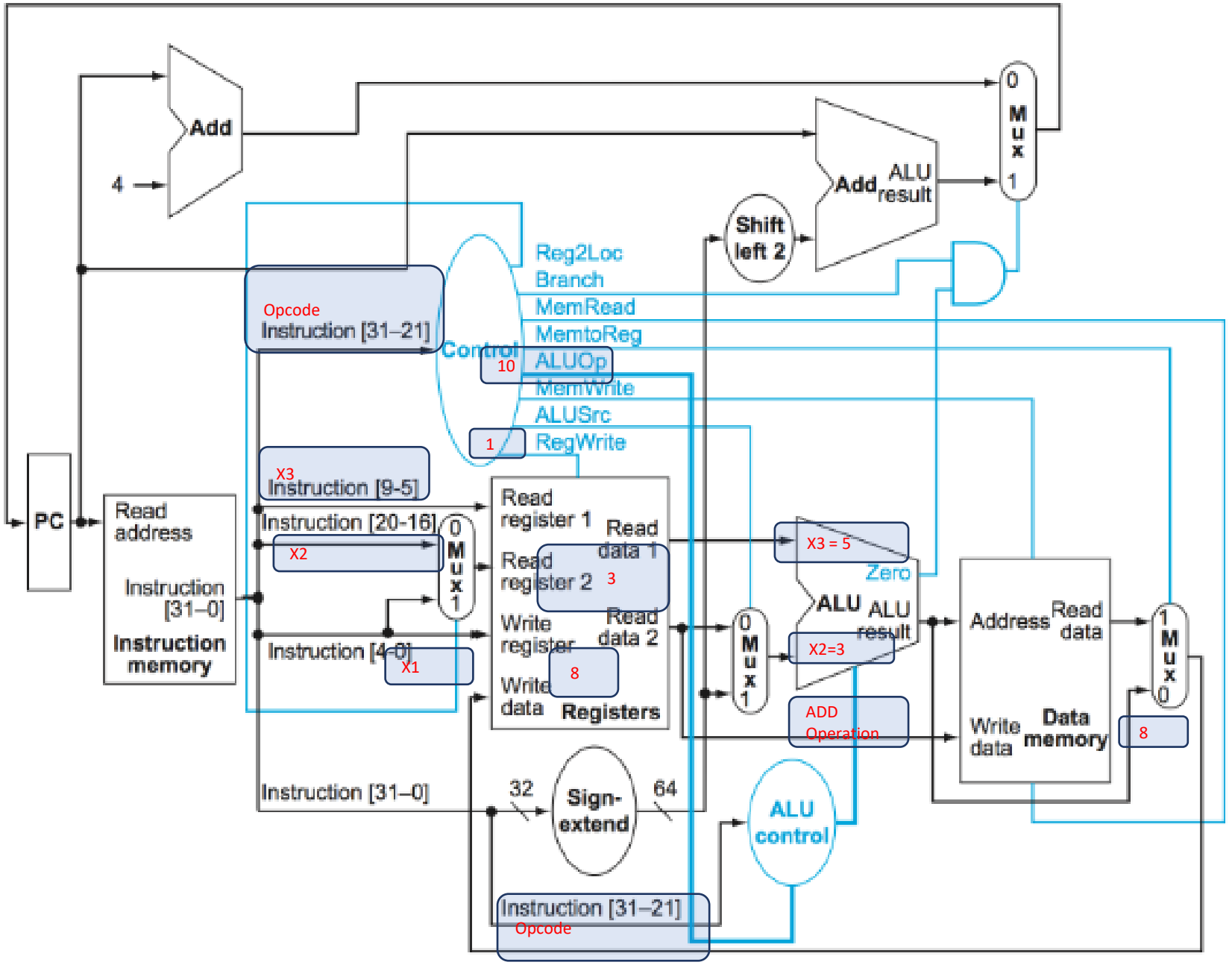
Control Signals

Signal	Function When asserted	Function When deasserted
Reg2Loc	The register number for Read register 2 comes from the Rt field(bits 4:0)	The register number for Read register 2 comes from the Rm field(bits 20:16)
AluSrc	Choose second ALU operand –the sign extended bits of instruction	Choose second ALU operand –the second register (Read data2)
MemtoReg	Value writing to the register file – Write data from memory.	Value writing to the register file – Write data from ALU.
RegWrite	Register in write register is written with value in Write data input.	NONE
MemRead	Data in the given address is send to Read Data.	NONE
MemWrite	Data goes to write data to be written in the given address.	NONE
PC Src	PC is replaced by the output of the adder that computes the branch target	Adder that computes the PC+4 for next instruction.

R Type instruction - in Datapath

- ADD X1,X2,X3

- 10001011000 00011 000000 00010 00001
- 11 bits 5 bits 5 bits 6 bits 5 bits
- 31-21 4-0 9-5 15-10 20-16


Add
4
0
M
u
x
1
Add
ALU result
Shift left 2
Reg2Loc
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Opcode
Instruction [31–21]
Control
10
1
X3
Instruction [9-5]
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [20-16]
X2
Instruction [4-0]
X1
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
3
8
Registers
X3 = 5
Zero
ALU
ALU result
X2=3
ADD
Operation
Address
Read data
Write data
Data memory
8
Instruction [31–0]
32
Sign-extend
64
ALU control
Instruction [31–21]
Opcode

D Type

- LDR X1,[X2,#offset]
- Example LDR X10, [X11, #0]

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1	0	1	0
opcode											DT Address									Op		Rn					Rt					

Control Line table Q 2 and 3 in worksheet

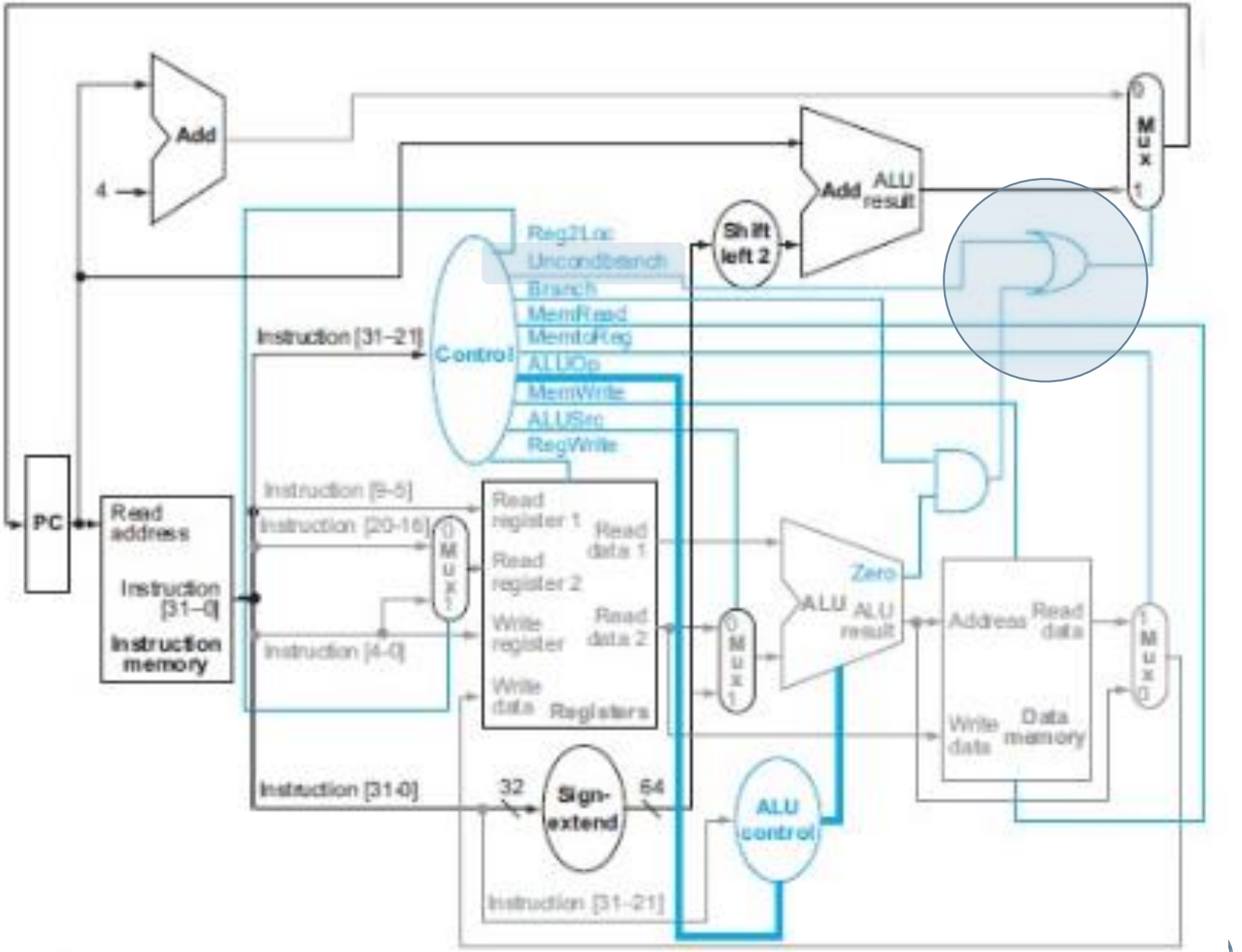
Instruction	Reg2Loc	AluSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	AluOp1	AluOp2
R type									
LDUR									
STUR									
CBZ									

0 : if reset
1 : if set
X : if don't care

Instruction	Reg2Loc	AluSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	AluOp1	AluOp2
R type	0	0	0	1	0	0	0	1	0
LDUR	X	1	1	1	1	0	0	0	0
STUR	1	1	X	0	0	1	0	0	0
CBZ	1	0	X	0	0	0	1	0	1

0 : if reset
1 : if set
X : if don't care

Unconditional Branch



Q 4 in Worksheet

- Consider the table

R-type	I-Type	LDUR	STUR	CBZ	B
24%	28%	25%	10%	11%	2%

- What fraction of all instructions use data memory?
- What fraction of all instructions use instruction memory?
- What fraction of all instructions use the sign extend?
- What is the sign extend doing during cycles in which its output is not needed?

Solution

- Fraction of data memory utilized =25%+ 10% =35%
- Fraction of instruction -memory utilized: All
- Fraction of sign-extend employed. The instructions I-type. LDUR, STUR. CBZ and B use sign-extend. The fraction of sign-extend used = 28% + 25% + 10%+ 11% + 2% =76% .
- Sign-extend is computed on every clock cycle.

Q 5 and 6

- Form a team of 2
- Answer question 5 and 6 in a separate sheet and turn it in.

Going virtual - NOT viral !!!!!!

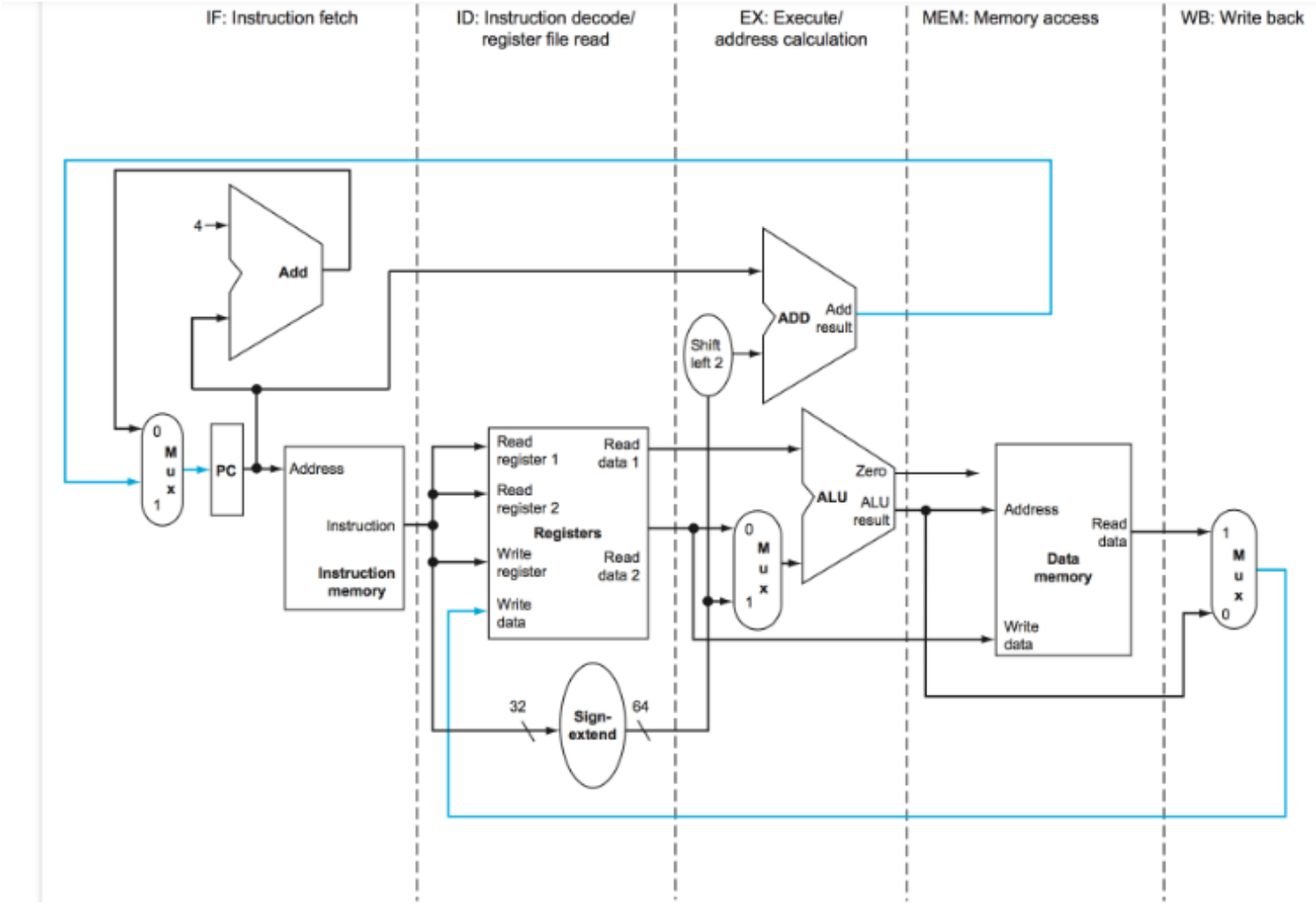
- From next week we are moving online
  - Please check the email you got from me through Microsoft team and sign in with your MSU email id.
  - All course materials, assignments and quizzes will be online through Moodle.
  - Please check your email and Moodle announcements everyday.
  - MS Teams will be used for live lectures or office hours if necessary.
  - Email me for any questions regarding the course any time.
  - Be self motivated to be online during the time of the class.
  - Attendance will be counted based on log during class time (starting 6 pm) if its late 15 minutes you wont get the days attendance.
  - Make use of chat in MS Team or discussion forum in Moodle for any questions during class hours .

Pipelining

- ***Pipelining***: An implementation technique in which multiple instructions are overlapped in execution, much like an assembly line.

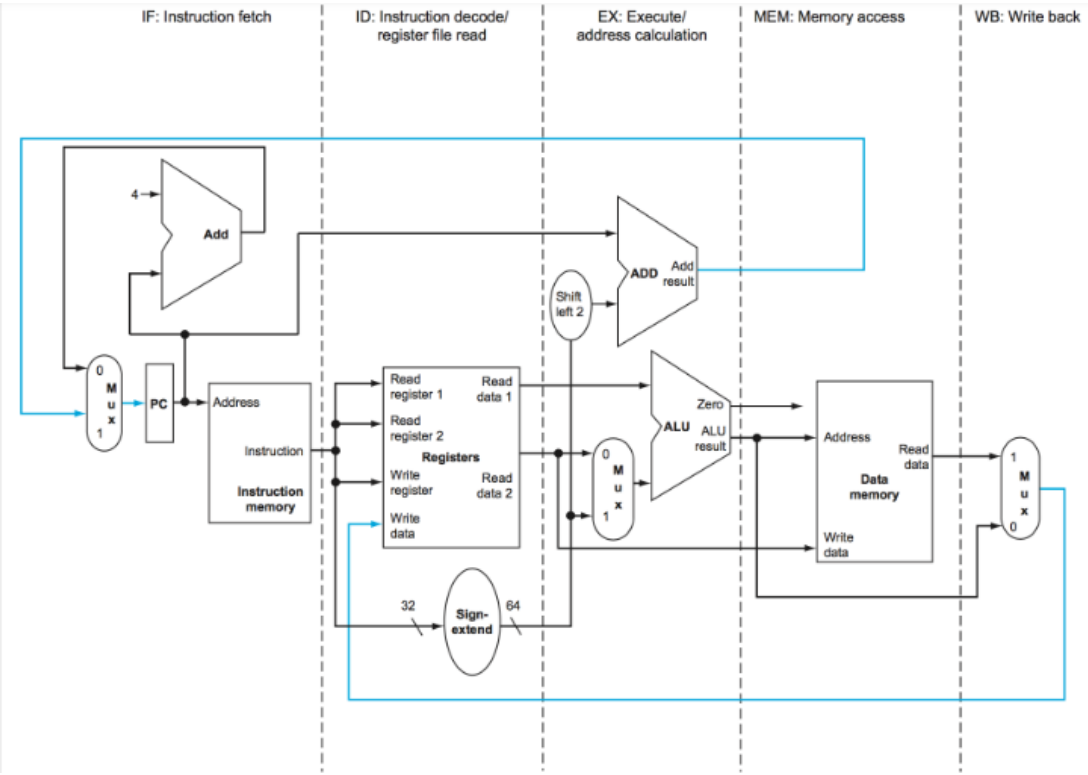
Pipelining

- Instructions classified into 5 steps.
  - Fetch instruction from memory.
  - Read registers and decode the instruction.
  - Execute the operation or calculate an address.
  - Access an operand in data memory (if necessary).
  - Write the result into a register (if necessary).



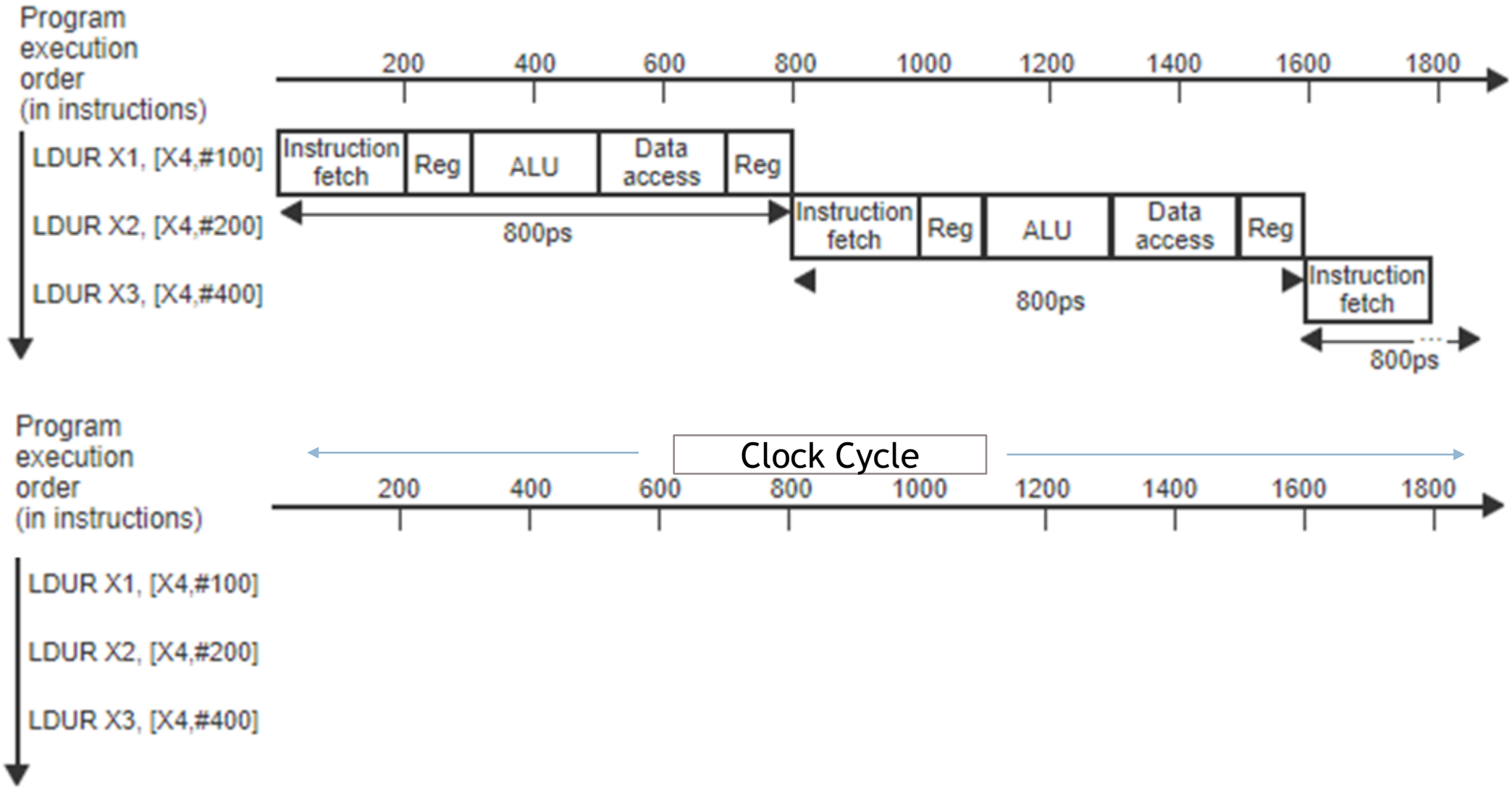
Pipelining

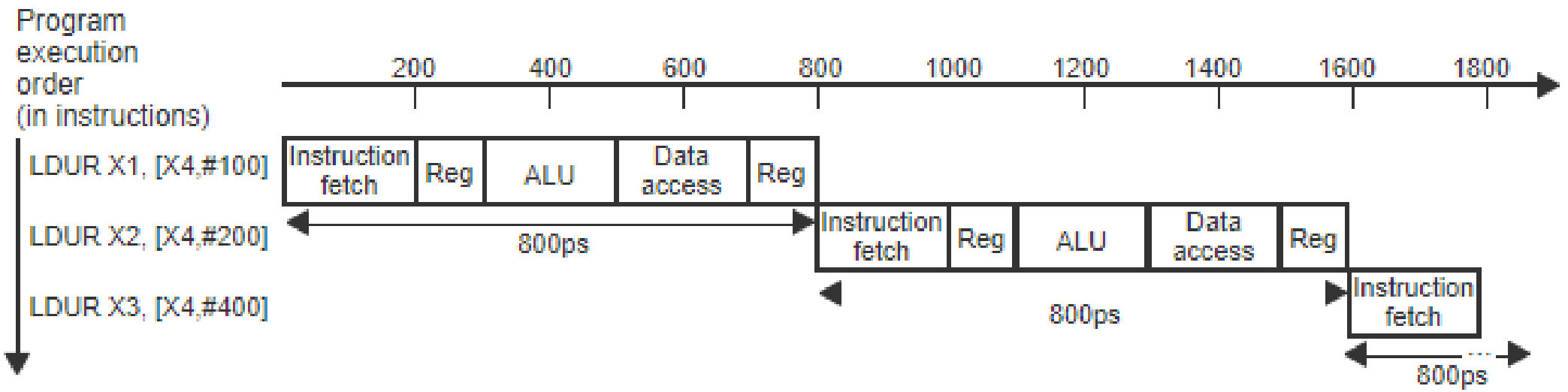
- Study limited to seven instructions:
  - load register (LDUR),
  - store register (STUR),
  - add (ADD),
  - subtract (SUB),
  - AND (AND), OR (ORR),
  - and compare and branch on zero (CBZ).

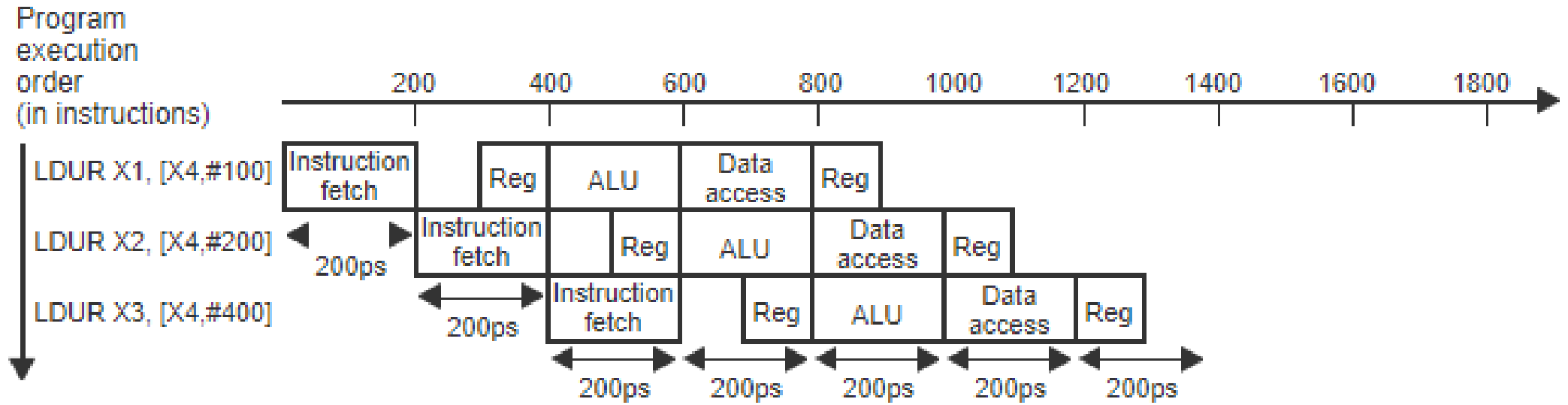


Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load register (LDUR)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store register (STUR)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (ADD, SUB, AND, ORR)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (CBZ)	200 ps	100 ps	200 ps			500 ps

Pipelining


Program execution order (in instructions)
200
400
600
800
1000
1200
1400
1600
1800
LDUR X1, [X4,#100]
LDUR X2, [X4,#200]
LDUR X3, [X4,#400]
Instruction fetch
Reg
ALU
Data access
Reg
800ps
800ps
800ps
Clock Cycle


Program execution order (in instructions)
200
400
600
800
1000
1200
1400
1600
1800
LDUR X1, [X4,#100]
LDUR X2, [X4,#200]
LDUR X3, [X4,#400]
Instruction fetch
Reg
ALU
Data access
Reg
800ps
Instruction fetch
Reg
ALU
Data access
Reg
800ps
Instruction fetch
800ps

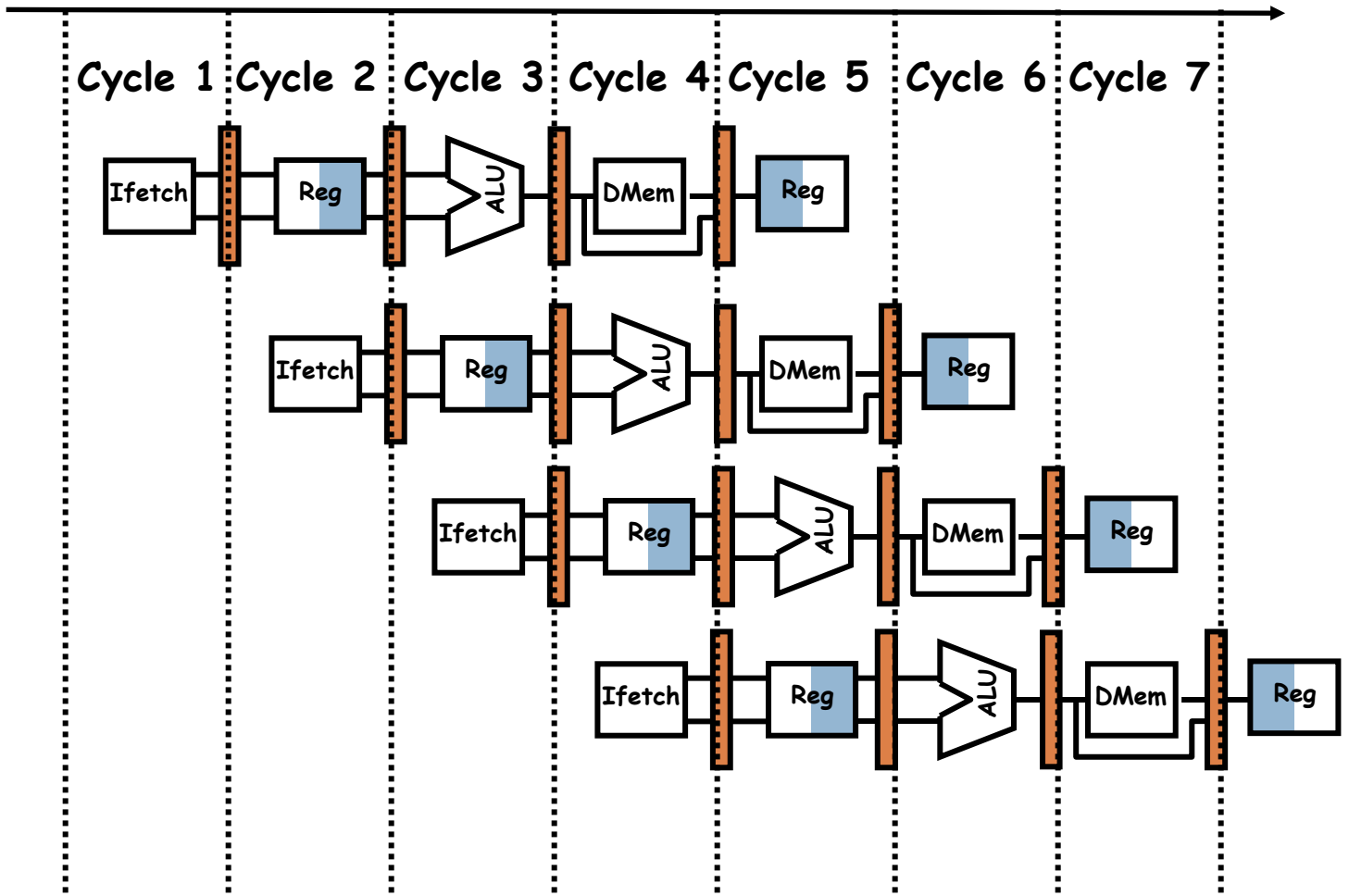

Program execution order (in instructions)
200
400
600
800
1000
1200
1400
1600
1800
LDUR X1, [X4,#100]
LDUR X2, [X4,#200]
LDUR X3, [X4,#400]
Instruction fetch
Reg
ALU
Data access
Reg
200ps
Instruction fetch
Reg
ALU
Data access
Reg
200ps
Instruction fetch
Reg
ALU
Data access
Reg
200ps
200ps
200ps
200ps
200ps

Pipelining

- Time between instructions $_{\text{Pipelined}} = \frac{\textit{Time between instructions}_{\textit{NonPipelined}}}{No.of\ pipe\ stages}$

- Stages of instruction execution:
  - **IF**: Instruction fetch
  - **ID**: Instruction decode and register file read
  - **EX**: Execution or address calculation
  - **MEM**: Data memory access
  - **WB**: Write back

Pipelining


Cycle 1
Cycle 2
Cycle 3
Cycle 4
Cycle 5
Cycle 6
Cycle 7
Ifetch
Reg
ALU
DMem
Reg
Ifetch
Reg
ALU
DMem
Reg
Ifetch
Reg
ALU
DMem
Reg
Ifetch
Reg
ALU
DMem
Reg

Pipeline Hazards

- There are situations in pipelining when the next instruction cannot execute in the following clock cycle
- Structural hazard
- Data hazards
- Control Hazards

Pipeline hazards

- **Structural Hazards**
  - The hardware cannot support the combination of instructions that we want to execute in the same clock cycle.

- **Data Hazards**
  - When a planned instruction cannot execute in the proper clock cycle because data that is needed to execute the instruction are not yet available.
    - ***Forwarding /Bypassing***. A method of resolving a data hazard by retrieving the missing data element from internal buffers rather than waiting for it to arrive from programmer-visible registers or memory.
    - **Stall :** inserting one or more **"bubbles"** in the pipeline

Data Hazard

- Example If R1=5, R2=3,R0=4. R3=1

ADD R1, R2, R0

SUB R5, R1, R3

Instruction\ Cycle	1	2	3	4	5	6	7
I_1	IF	ID	EX	DM	WB		
I_2		IF	ID	EX	DM	WB	

Data Hazard

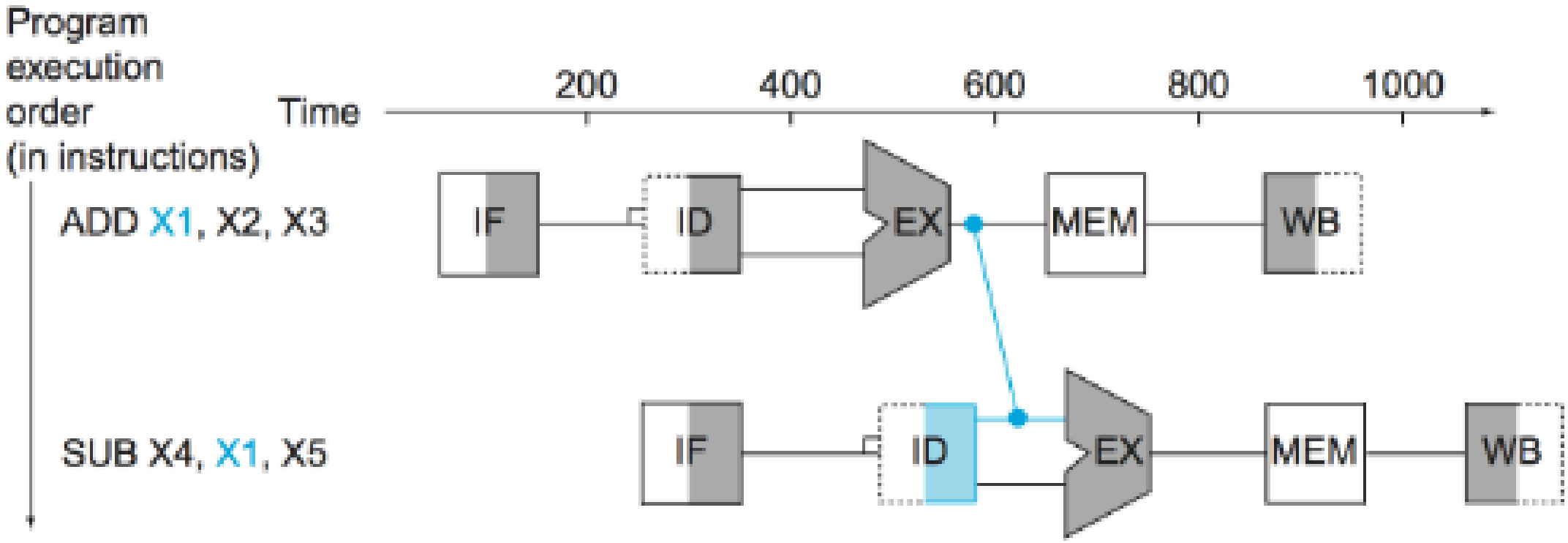
Will these instructions cause data hazard?

ADD R1, R2, R0

SUB R5, R2, R3

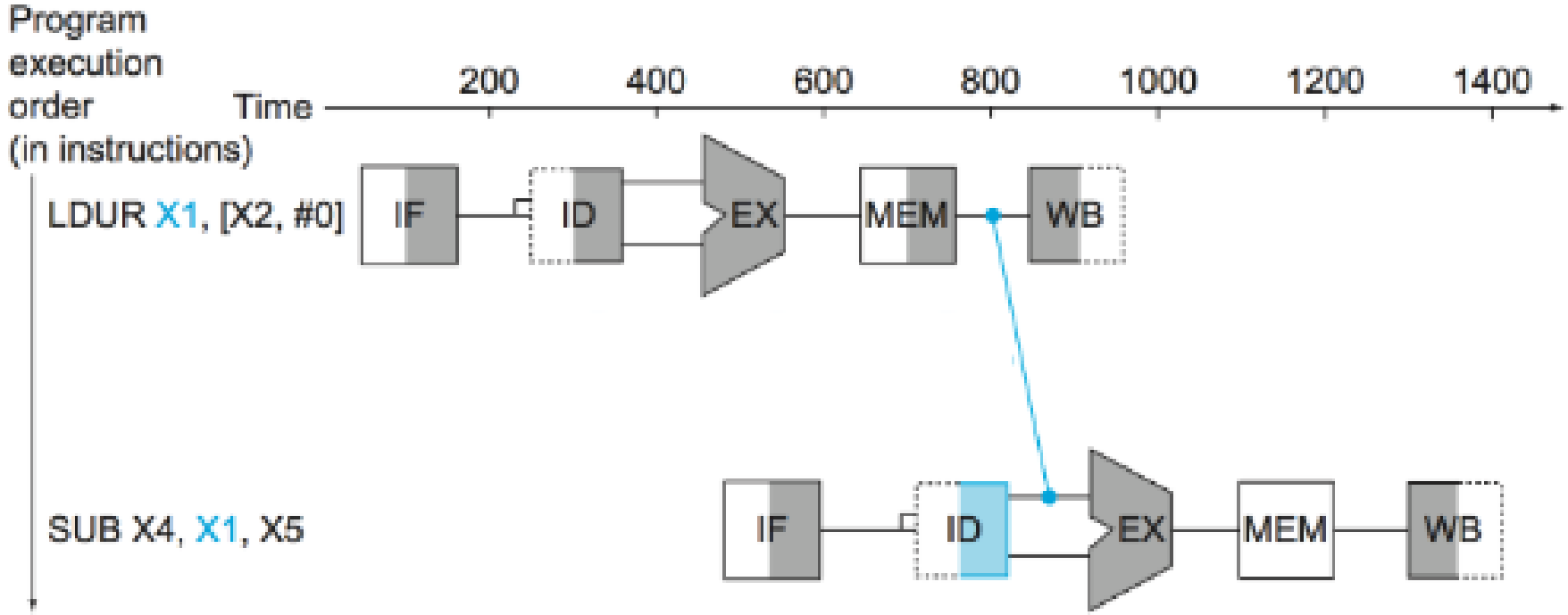
No Data Hazard

Bypassing OR *Forwarding*



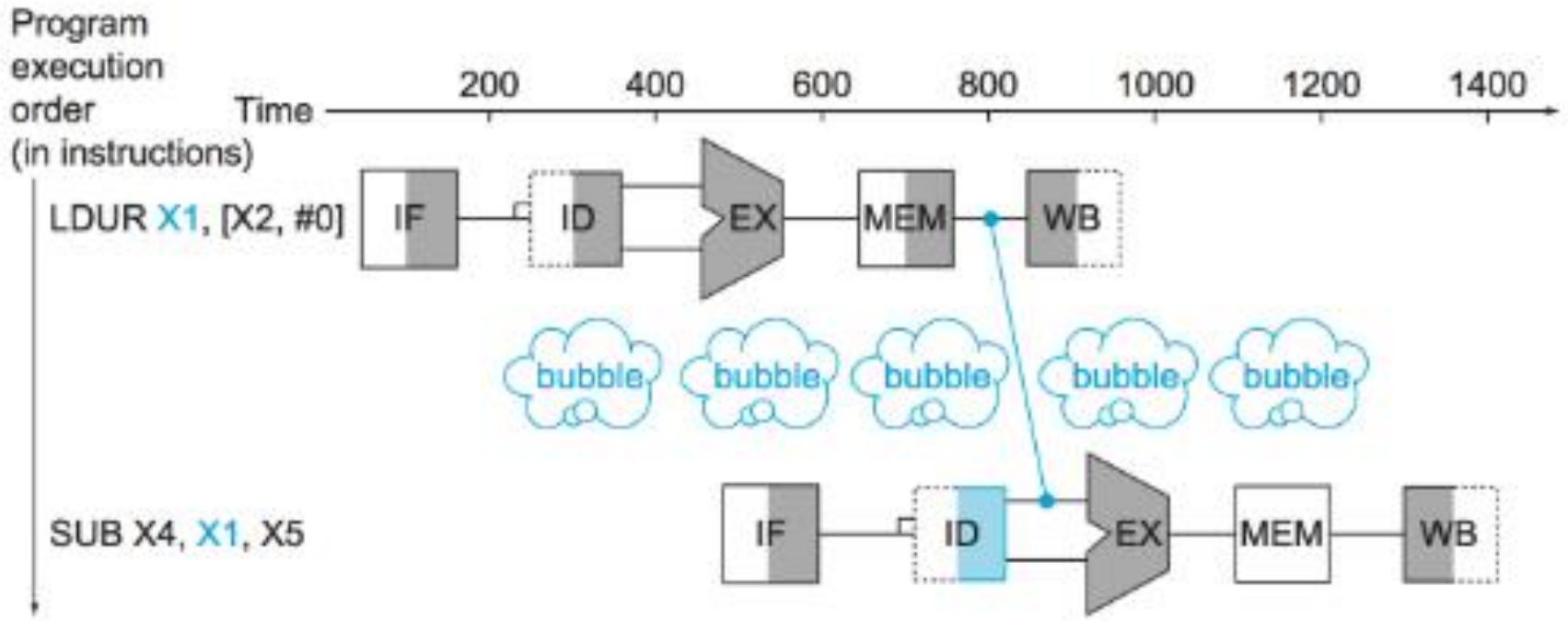
Data Hazards *Example*

- ***Load-use data hazard***: A specific form of data hazard in which the data being loaded by a load instruction has not yet become available when it is needed by another instruction.



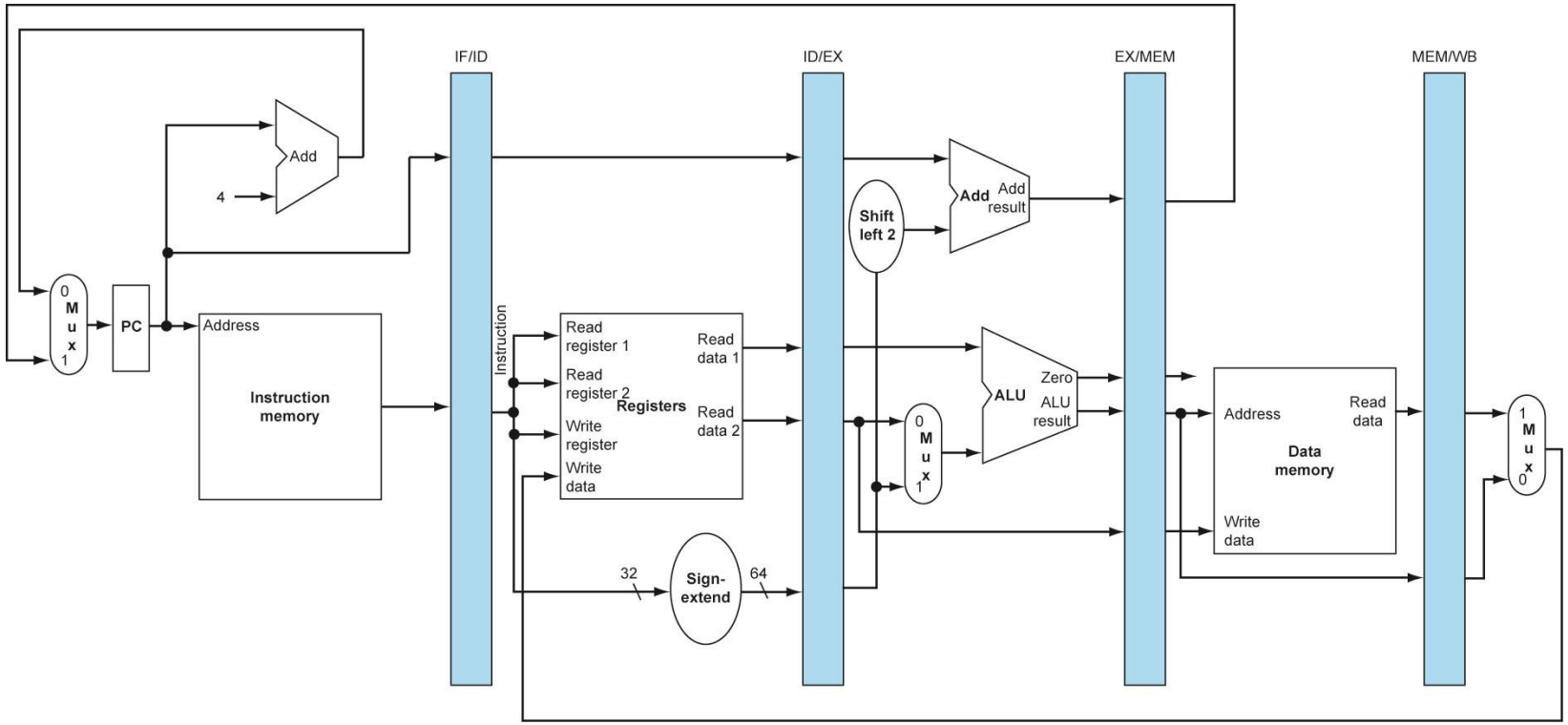
Data Hazards

- Common solution is to **stall** the pipeline until the hazard is resolved, inserting one or more "**bubbles**" in the pipeline. Programmers use NOP instruction



Pipeline registers

- Need registers between stages
  - To hold information produced in previous cycle



Q

- a = b + e;
  c = b + f;
- The generated LEGv8 code for the above C code segment, is given below: (Assuming all variables are in memory and are addressable as offsets from X0). Find the hazards in the code segment and reorder the instructions to avoid any pipeline stalls.

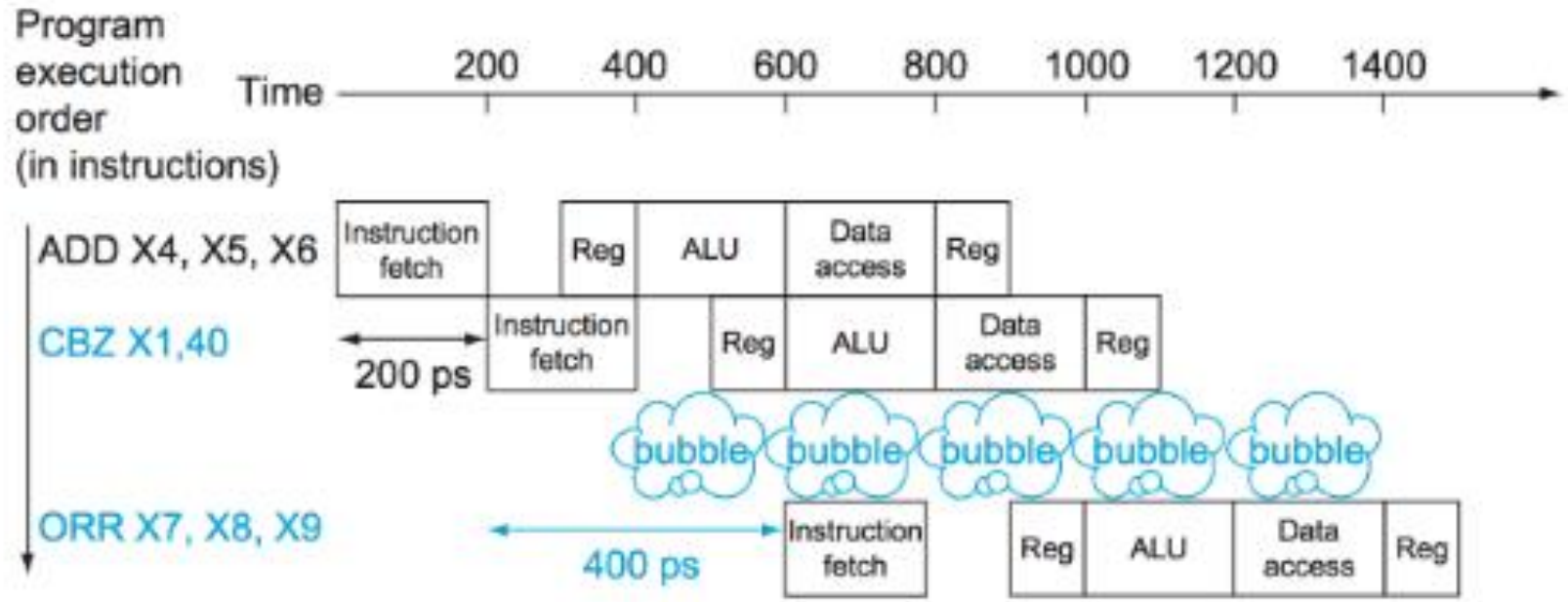
```
LDUR   X1, [X0,#0]   // Load b
LDUR   X2, [X0,#8]   // Load e
ADD    X3, X1, X2    // b + e
STUR   X3, [X0,#24]  // Store a
LDUR   X4, [X0,#16]  // Load f
ADD    X5, X1, X4    // b + f
STUR   X5, [X0,#32]  // Store c
```

Q

```
LDUR   X1, [X0,#0]   // Load b
LDUR   X2, [X0,#8]   // Load e
ADD    X3, X1, X2    // b + e
STUR   X3, [X0,#24]  // Store a
LDUR   X4, [X0,#16]  // Load f
ADD    X5, X1, X4    // b + f
STUR   X5, [X0,#32]  // Store c
```

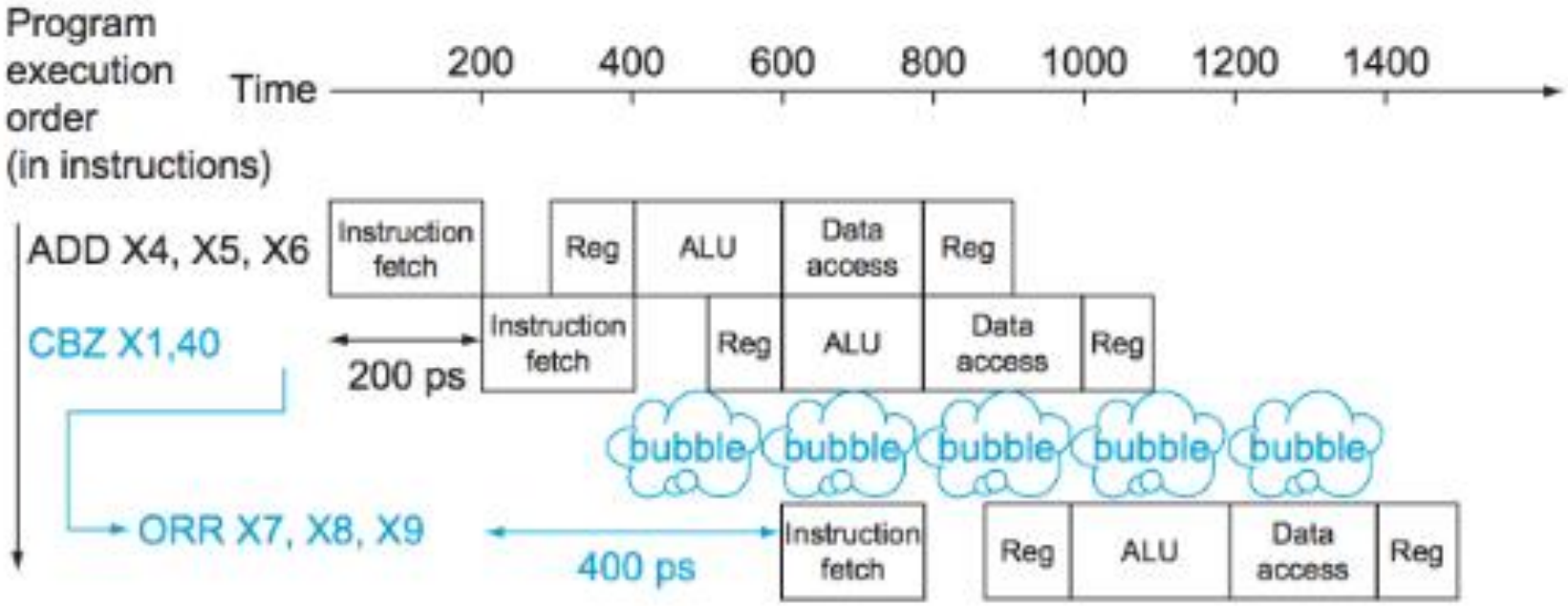
Control Hazards

- Control Hazards /Branch hazard
  - Arising from the need to make a decision based on the results of one instruction while others are executing.



Control Hazards

- ***Branch prediction***: A method of resolving a branch hazard that assumes a given outcome for the conditional branch and proceeds from that assumption rather than waiting to ascertain the actual outcome.



Control Hazard

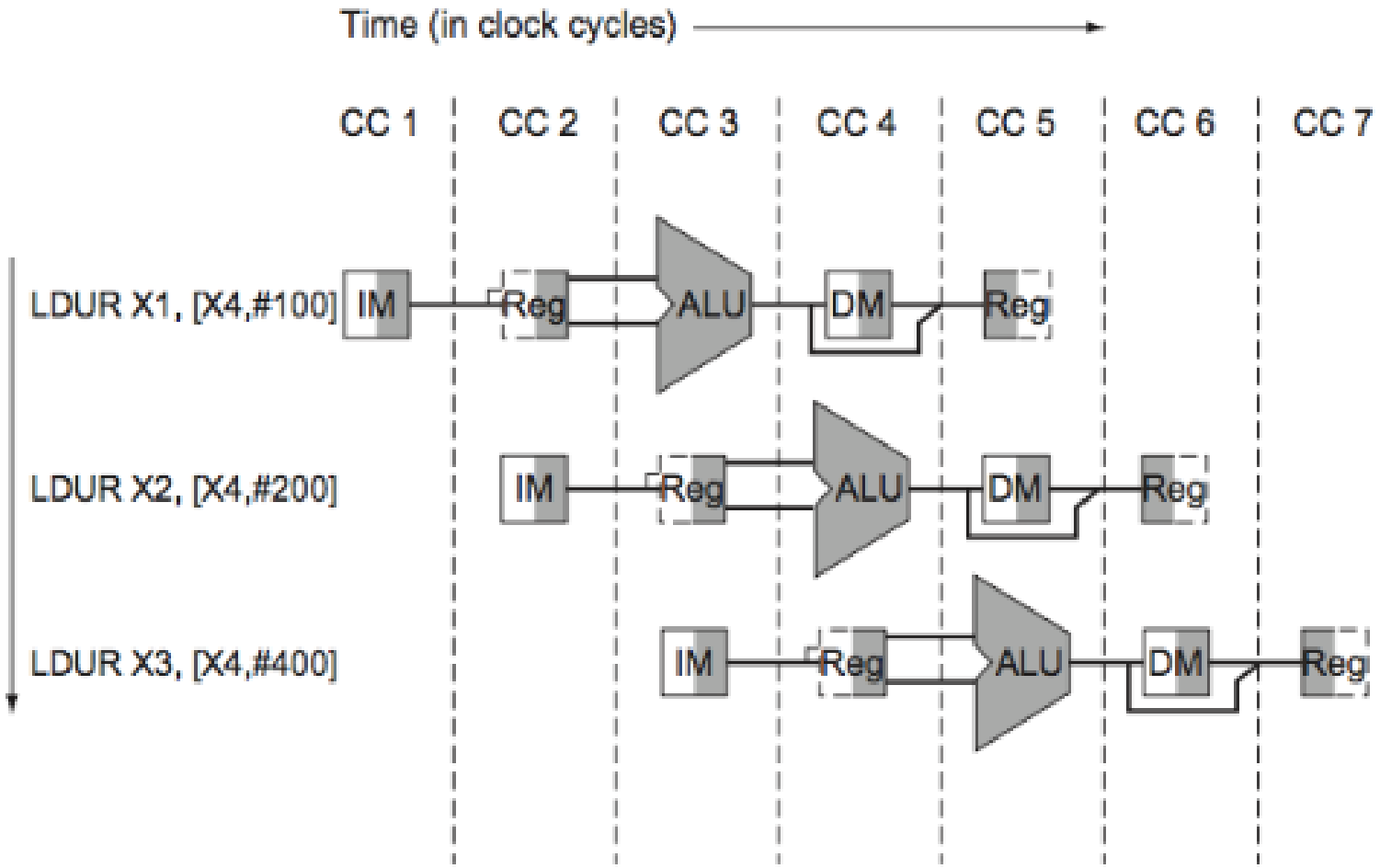
- One improvement over branch stalling is to **predict** that the conditional branch will not be taken and thus continue execution down the sequential instruction stream. If the conditional branch is taken, the instructions that are being fetched and decoded must be discarded.

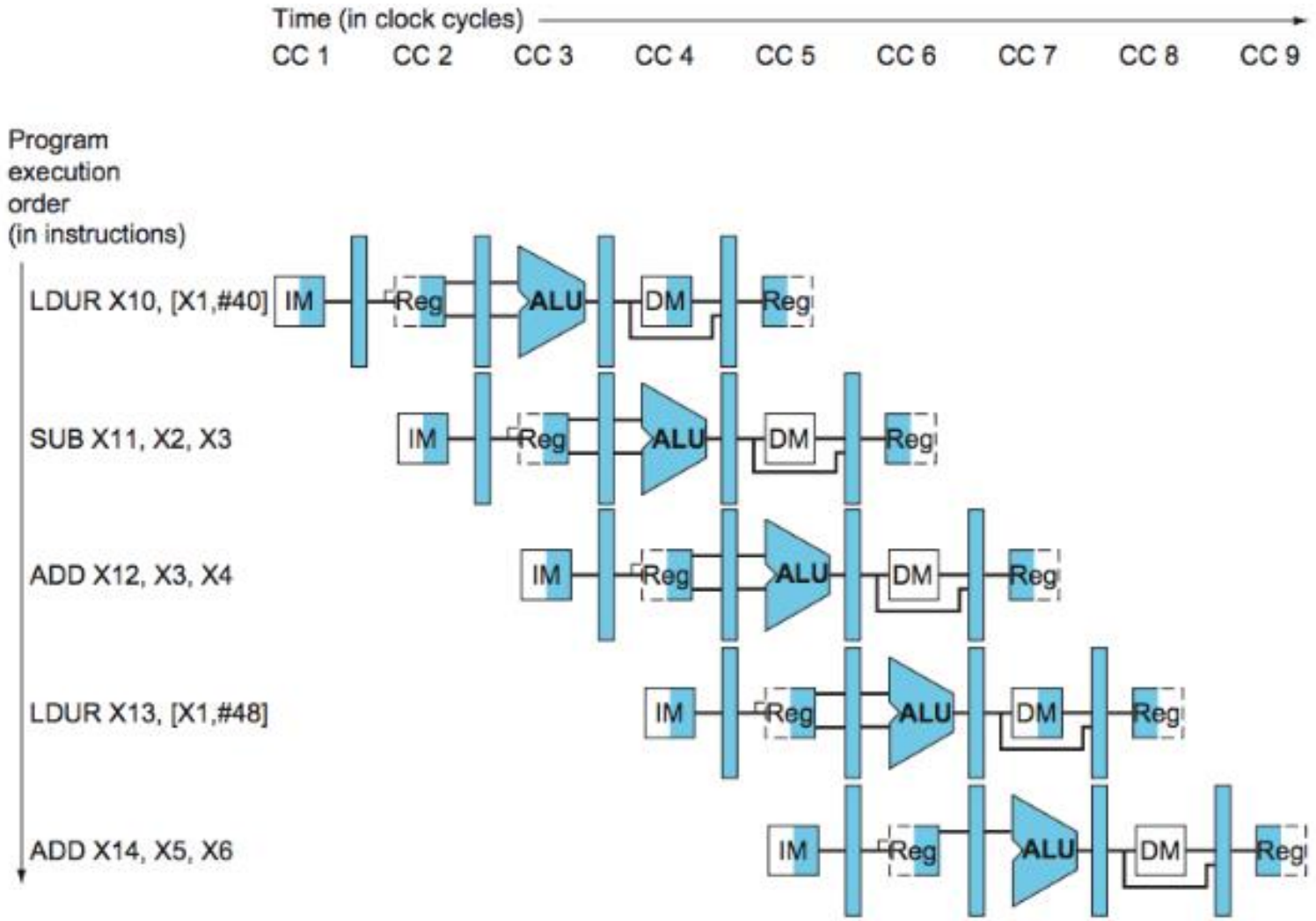
- ***Flush***: To discard instructions in a pipeline, usually due to an unexpected event.

Branch Prediction

- Static branch prediction
  - Based on typical branch behavior
  - Example: loop and if-statement branches
    - Predict backward branches taken
    - Predict forward branches not taken
- Dynamic branch prediction
  - Hardware measures actual branch behavior
    - e.g., record recent history of each branch
  - Assume future behavior will continue the trend
    - When wrong, stall while re-fetching, and update history

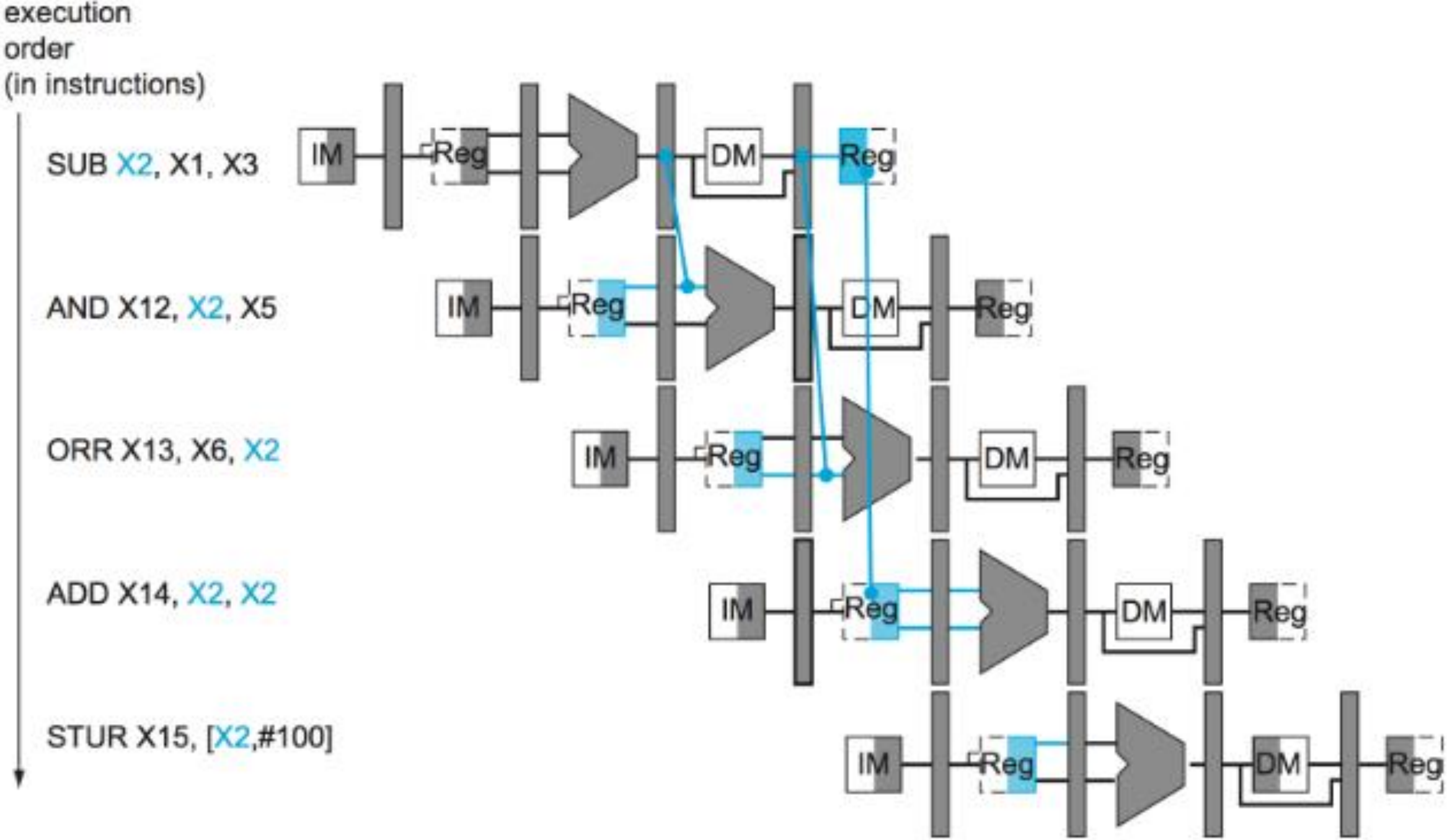
Pipeline Datapath




Time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
CC 8
CC 9
Program execution order (in instructions)
LDUR X10, [X1,#40]
SUB X11, X2, X3
ADD X12, X3, X4
LDUR X13, [X1,#48]
ADD X14, X5, X6
IM
Reg
ALU
DM
Reg

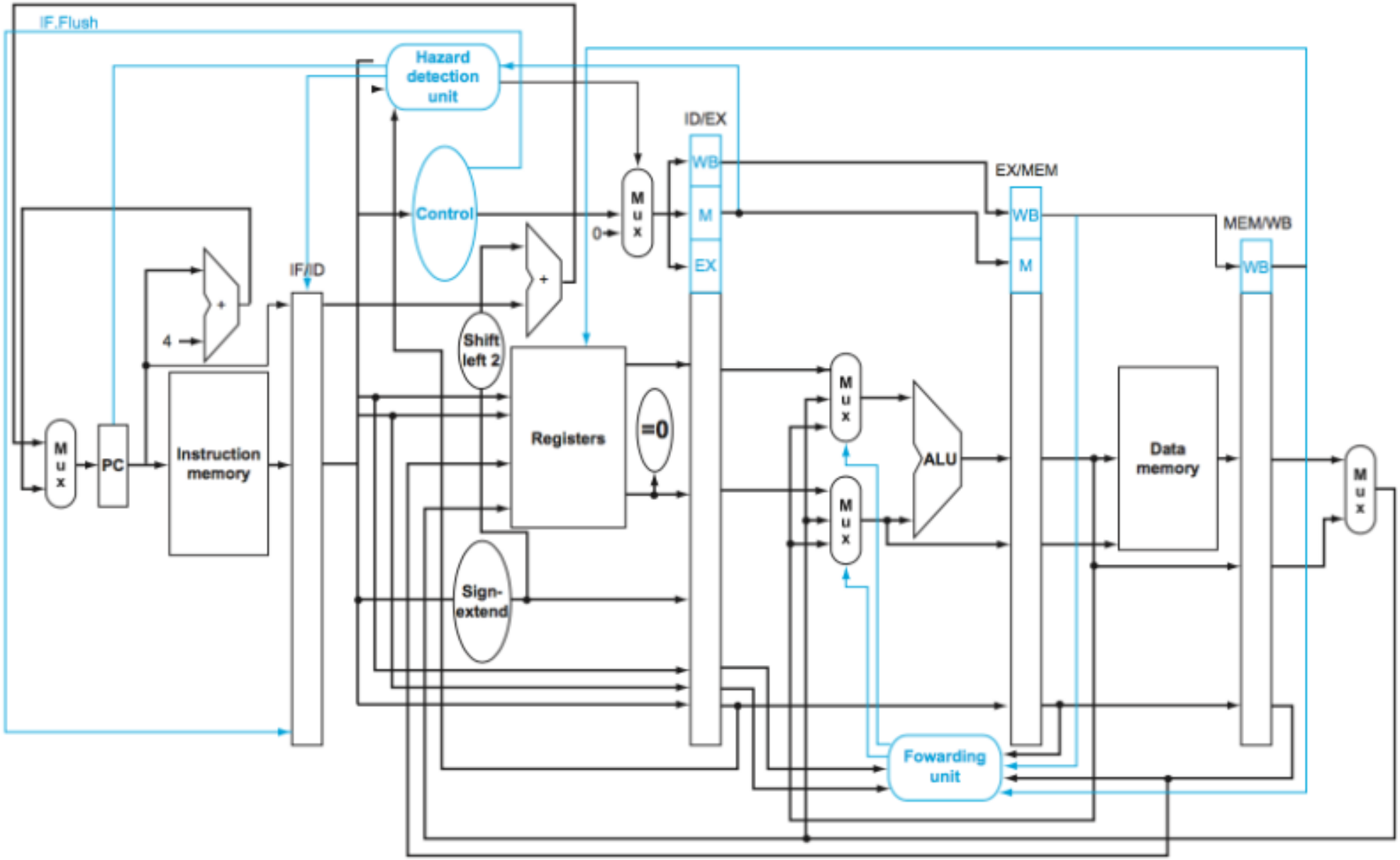
Q

```
SUB     X2,X1,X3
AND     X12, X2,X5
ORR     X13,X6,X2
ADD     X14,X2,X2
STUR    X15,[X2,#100]
```


execution
order
(in instructions)
SUB X2, X1, X3
AND X12, X2, X5
ORR X13, X6, X2
ADD X14, X2, X2
STUR X15, [X2,#100]
IM
Reg
DM
Reg

Latency

- ***Latency*** (***pipeline***): The number of stages in a pipeline or the number of stages between two instructions during execution.


IF.Flush
Hazard detection unit
Control
ID/EX
WB
M
EX
EX/MEM
WB
M
MEM/WB
WB
M
u
x
0
+
4
IF/D
Shift left 2
Registers
=0
PC
Instruction memory
Sign-extend
ALU
Data memory
Fowarding unit

EXCEPTIONS

- ***Exception***: Also called ***interrupt***. An unscheduled event that disrupts program execution.

- ***Interrupt***: An exception that comes from outside of the processor.

Exception Types

Type of event	From where?	ARMv8 terminology
System reset	External	Exception
I/O device request	External	Interrupt
Invoke the operating system from user program	Internal	Exception
Floating-point arithmetic overflow or underflow	Internal	Exception
Using an undefined instruction	Internal	Exception
Hardware malfunctions	Either	Exception or interrupt

Causes

Undefined instruction

Floating-point overflow and underflow

A hardware malfunction

Action

The processor must save the address of the unfortunate instruction in the *exception link register* (ELR) and then transfer control to the operating system at some specified address.

Exceptions Handling

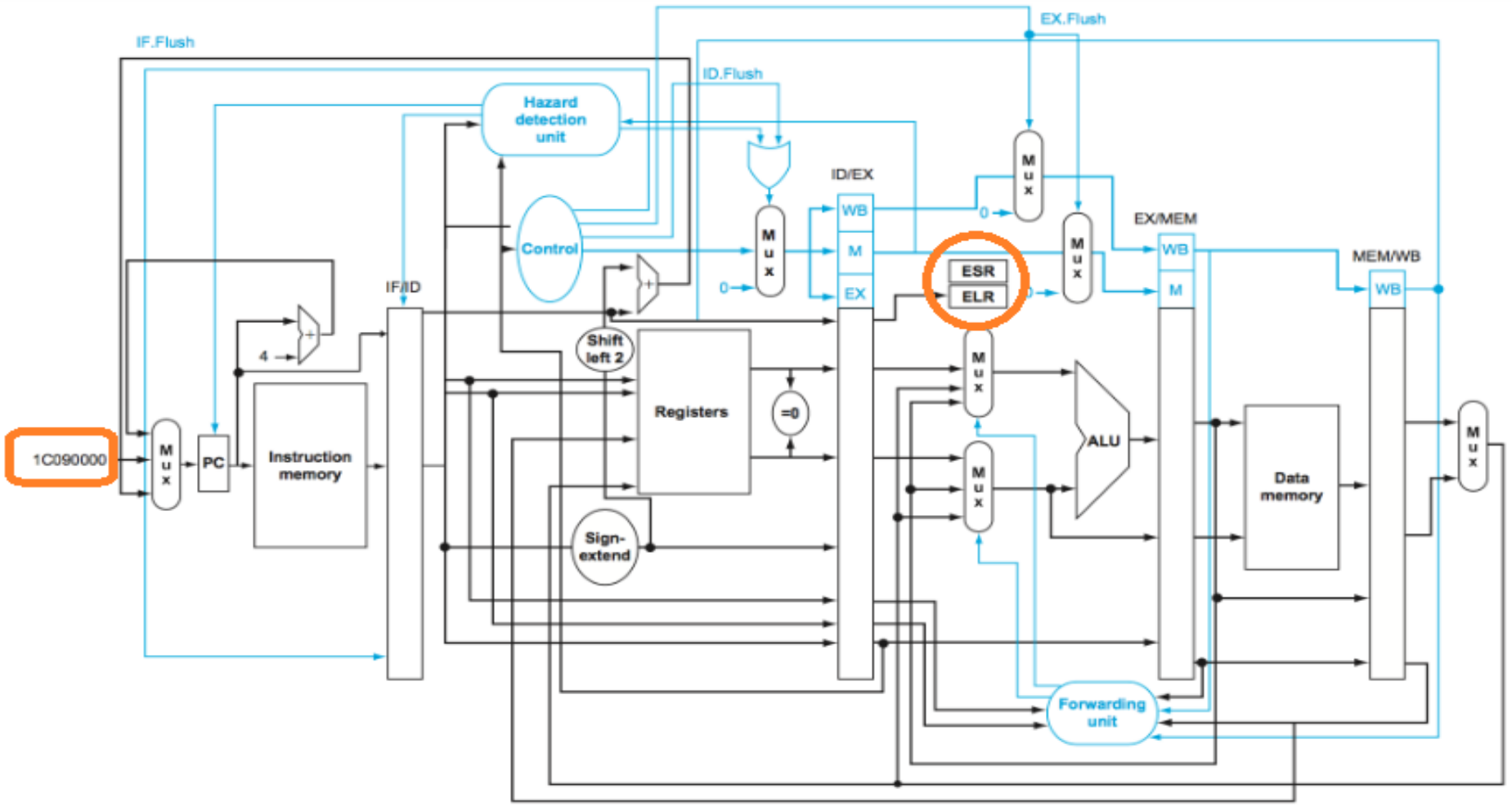
- Operating system to handle the exception, it must know
  - The reason for the exception .
  - The instruction that caused it .

  - Register (called the *Exception Syndrome Register* or *ESR*), which holds a field that indicates the reason for the exception.
  - *Vectored interrupts*

Exception type	Exception vector address to be added to a Vector Table Base Register
Unknown Reason	$00\ 0000_{two}$
Floating-point arithmetic exception	$10\ 1100_{two}$
System Error (hardware malfunction)	$10\ 1111_{two}$

Exceptions Handling

- *ELR*: A 64-bit register used to hold the address of the affected instruction.
- *ESR*: A register used to record the cause of the exception. In the LEGv8 architecture, this register is 32 bits.



Q

A five-stage pipeline (IF, ID, EX, MEM, WB) executes the following instruction sequence:
Which will be recognized first ?

```
XXX X1, X2, X1 // undefined instruction

SUB X1, X2, X1 // hardware error
```

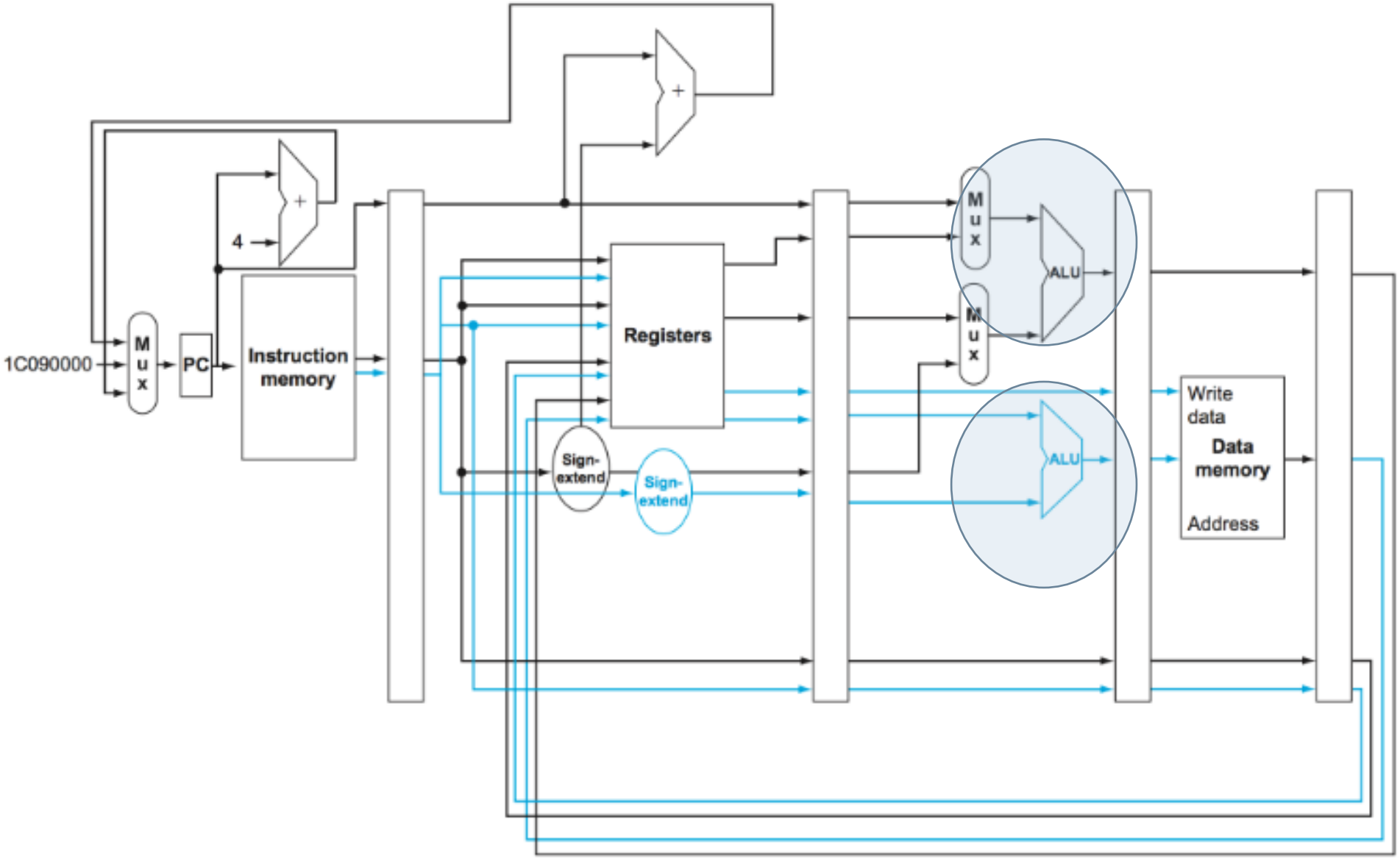
Parallelism via instructions

- ***Instruction-level parallelism***: The parallelism among instructions.
  - Increase depth of pipeline
  - Multiple instructions are launched in one clock cycle
    - **Static** multiple-issue processor where many decisions are made by the compiler before execution.
    - **Dynamic** multiple –issue processor where many decisions are made during execution by the processor.

Multiple issue:

- Packaging instructions into *issue slots*
- Dealing with data and control hazards.
- Example for a 2 issue slot Pipe line

Instruction type	Pipe stages							
ALU or branch instruction	IF	ID	EX	MEM	WB			
Load or store instruction	IF	ID	EX	MEM	WB			
ALU or branch instruction		IF	ID	EX	MEM	WB		
Load or store instruction		IF	ID	EX	MEM	WB		
ALU or branch instruction			IF	ID	EX	MEM	WB	
Load or store instruction			IF	ID	EX	MEM	WB	
ALU or branch instruction				IF	ID	EX	MEM	WB
Load or store instruction				IF	ID	EX	MEM	WB


1C090000
M
u
x
PC
Instruction
memory
4
+
+
Registers
Sign-
extend
Sign-
extend
M
u
x
ALU
M
u
x
ALU
Write
data
Data
memory
Address

Topics in Parallelism via instruction in detail will be included with Chapter 6 Parallel Processing