Computer Organization and Architecture

Lecture – 17

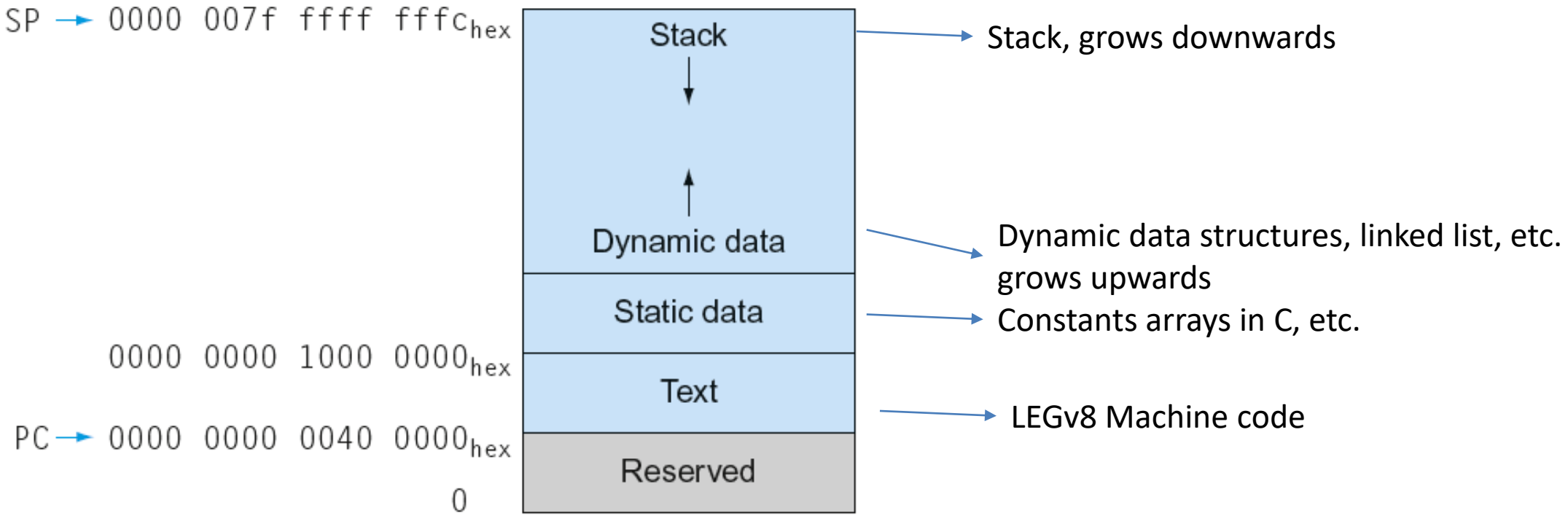
Oct 17th , 2022

Chapter – 4: The Processor

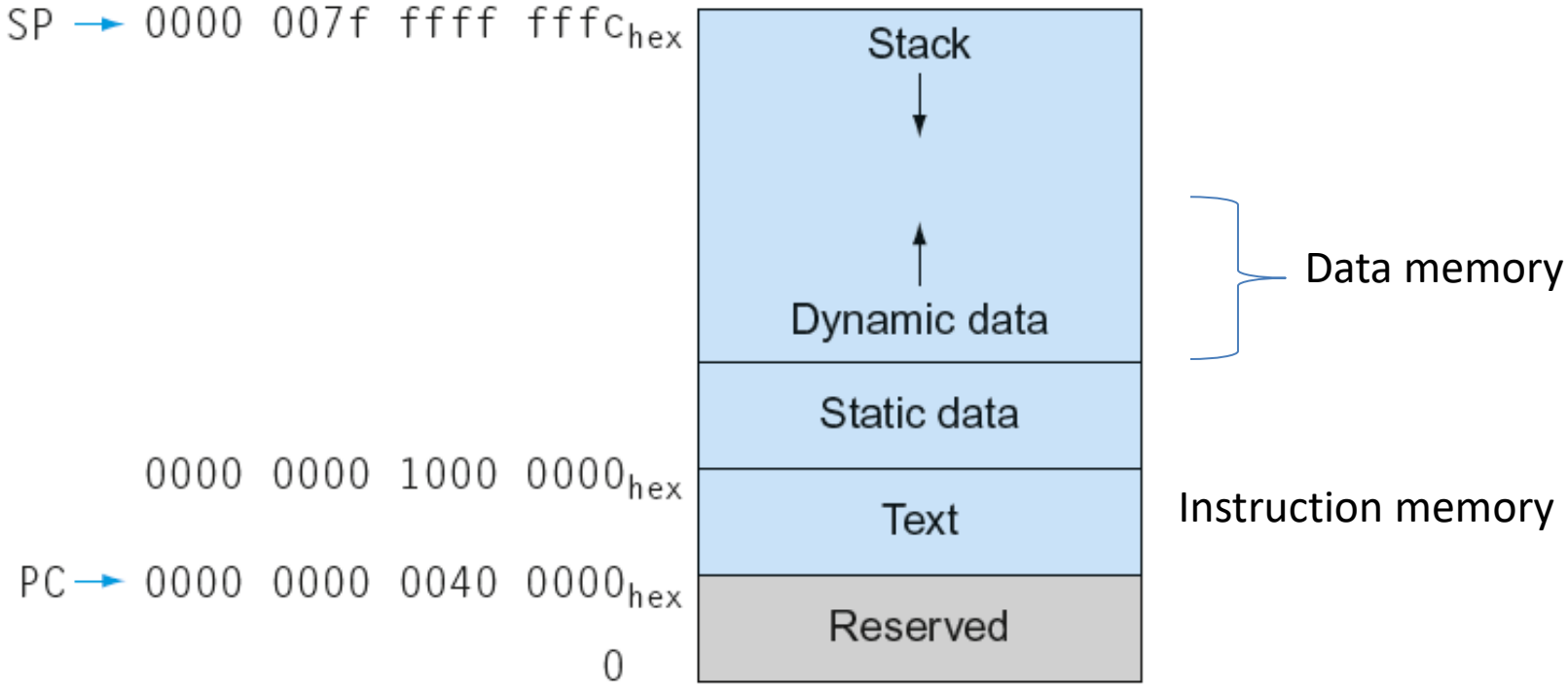
Introduction

- CPU performance factors
  - Instruction count (Chapter 2)
    - Determined by ISA and compiler
  - CPI and Cycle time (Chapter 1)
    - Determined by CPU hardware
  - Implementation of the processor CPI and cycle time.

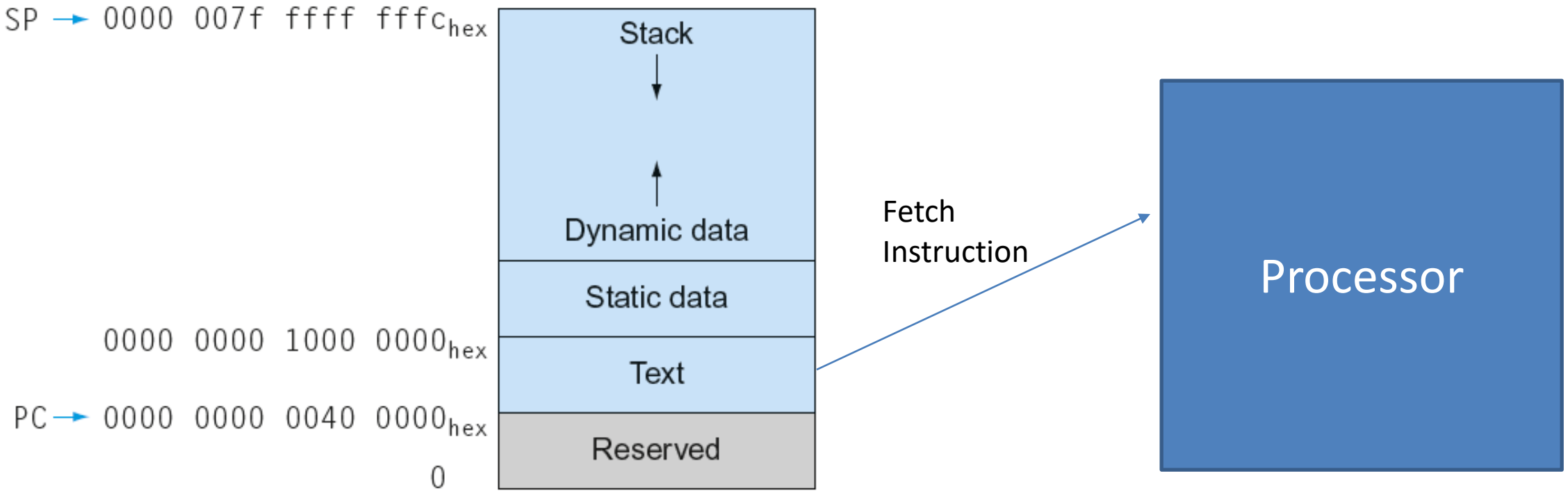
Memory Layout



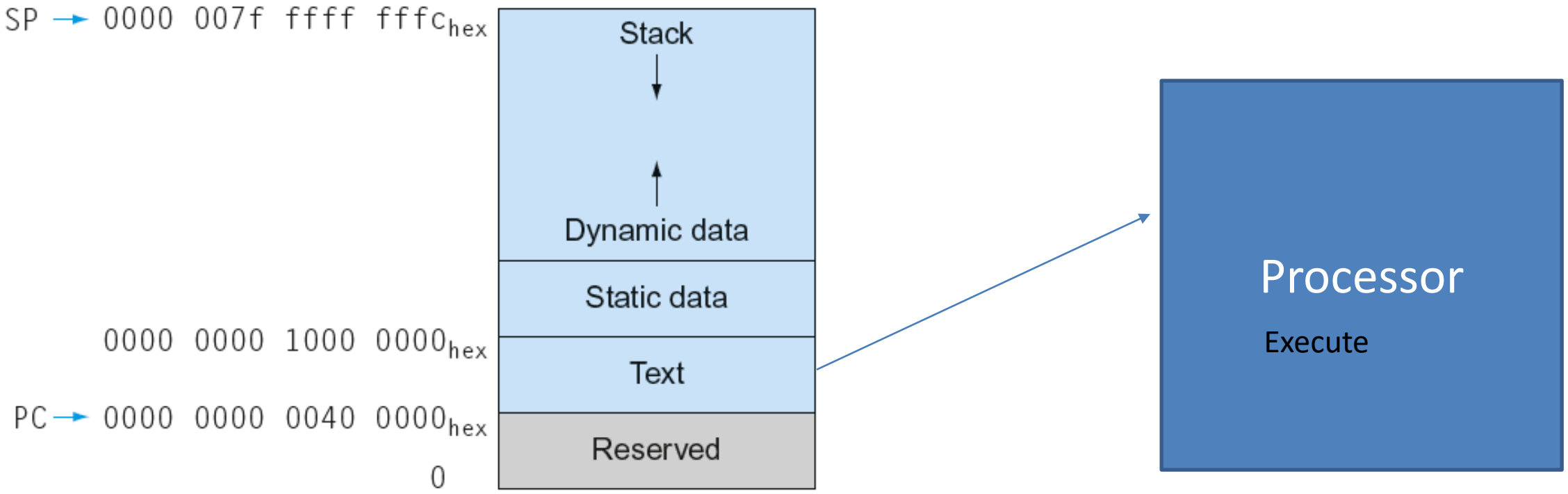
Memory Layout



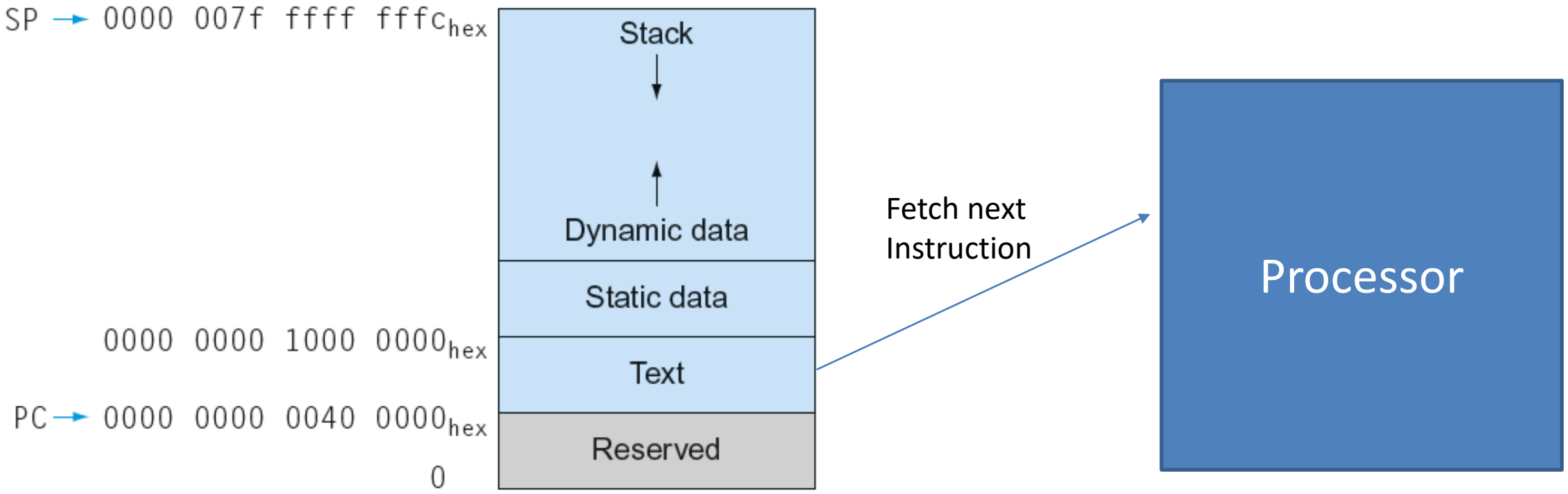
Executing instructions



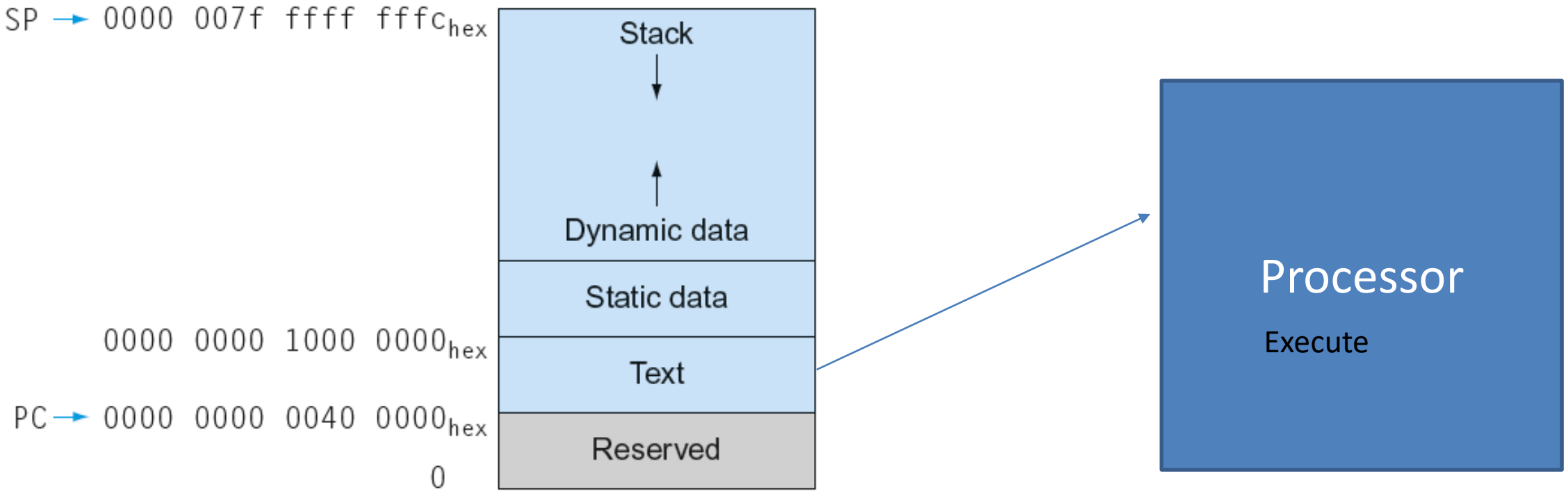
Executing instructions



Executing instructions



Executing instructions



Introduction

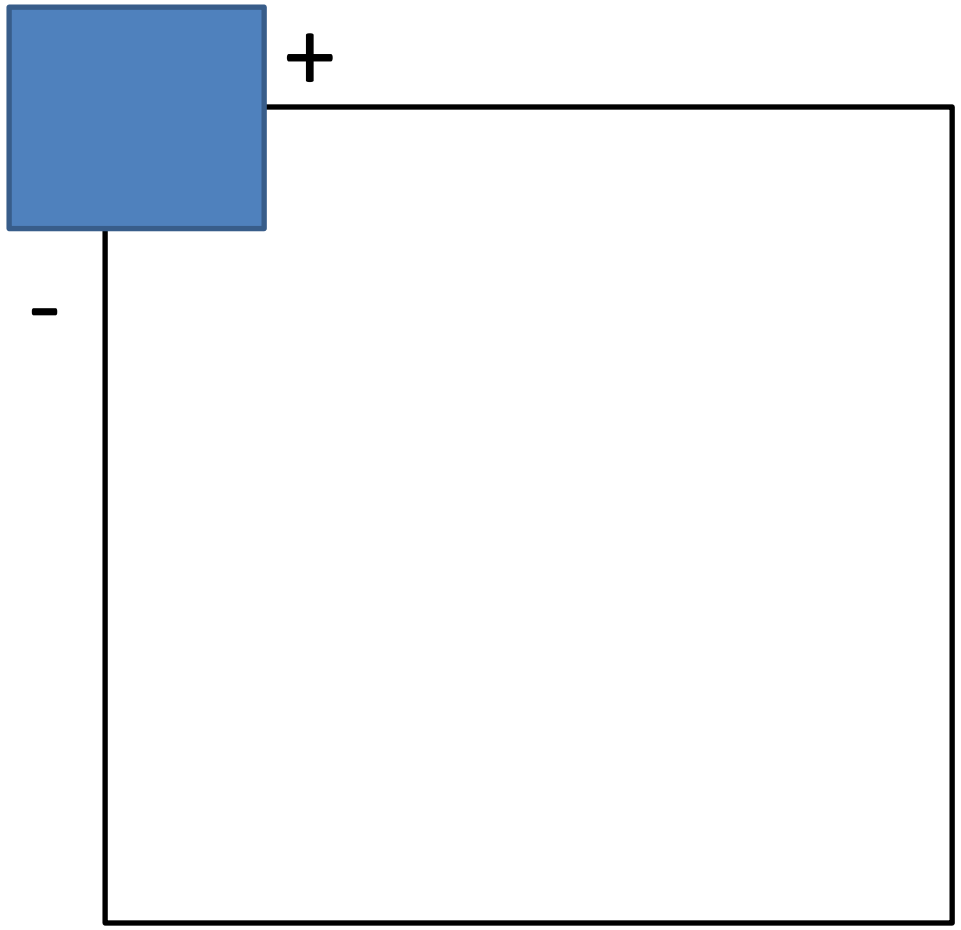
- CPU performance factors
  - Instruction count
    - Determined by ISA and compiler
  - CPI and Cycle time
    - Determined by CPU hardware
- We will examine two LEGv8 implementations
  - A simplified version
  - A more realistic pipelined version
- Simple subset, shows most aspects
  - Memory reference: LDUR, STUR
  - Arithmetic/logical: ADD, SUB, AND, ORR
  - Control transfer: CBZ , B

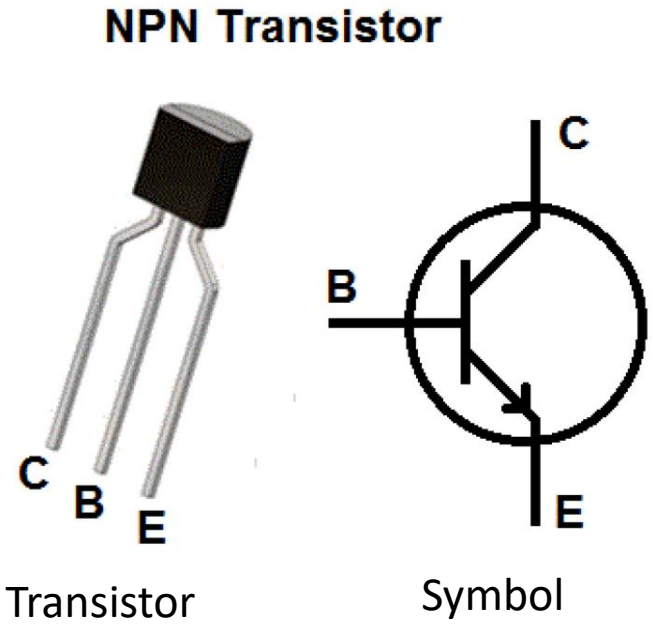
Building a Datapath

- Combine various circuits elements to create a processor
  - Combinational elements
  - State elements

Combinational Elements

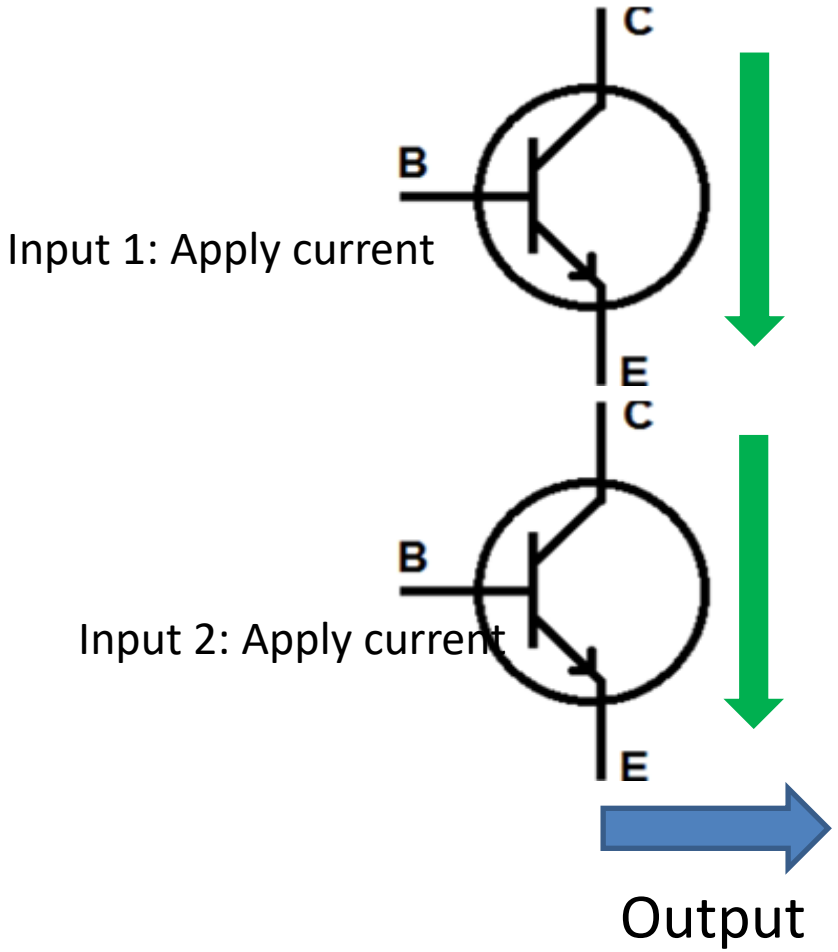
Control the flow of current using transistors





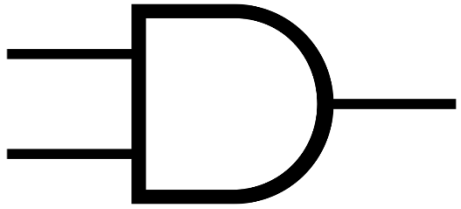
Transistor Symbol

Two transistors in parallel: Logic Gate



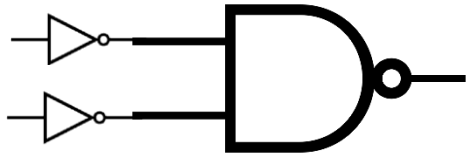
Input 1	Input 2	Output
0	0	0
1	0	0
0	1	0
1	1	1

AND Gate

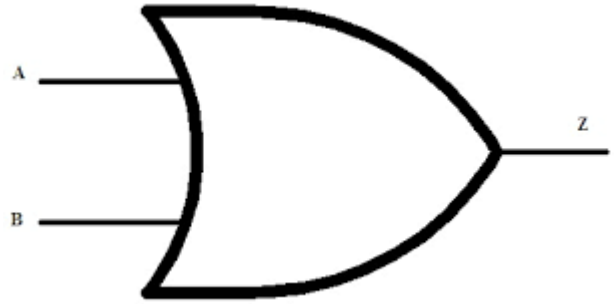


Symbol

OR Gate

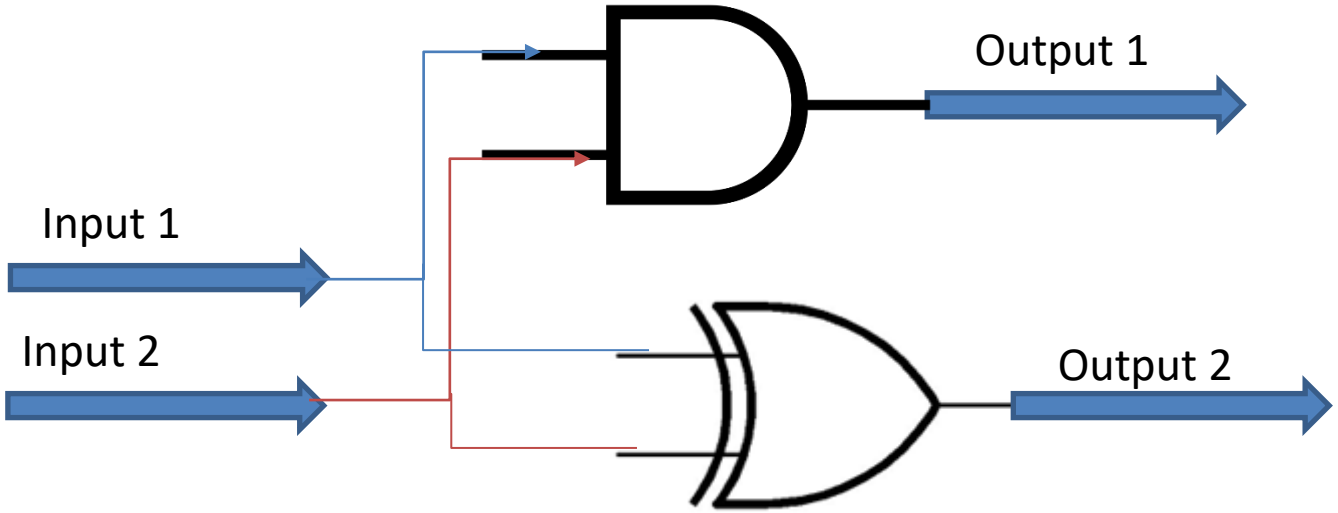


Input 1	Input 2	Output
0	0	0
1	0	1
0	1	1
1	1	1



OR Gate

Design an Adder



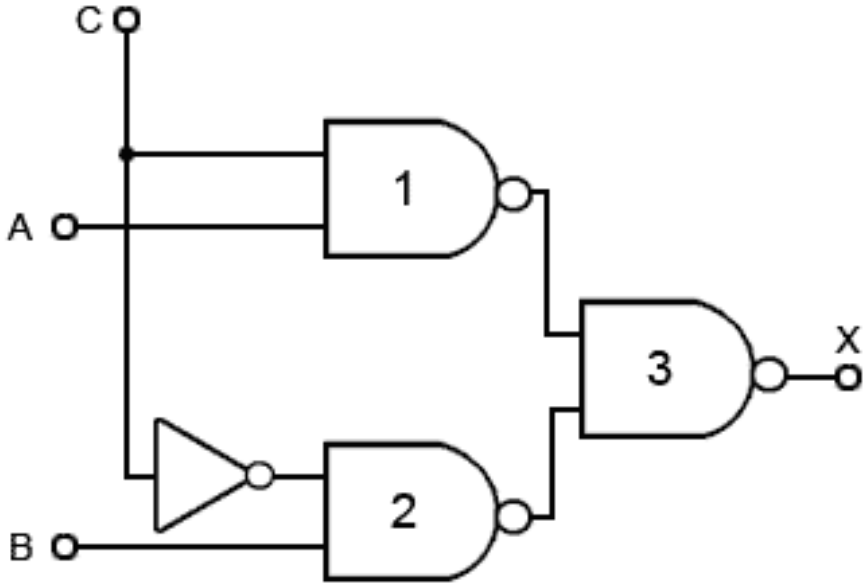
XOR Gate

It looks similar to OR Gate.
Except the last row is inverted where both inputs are 1, the result in inverted.

Input 1	Input 2	Output 1	Output 2
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Binary

Multiplexer (Data selector)

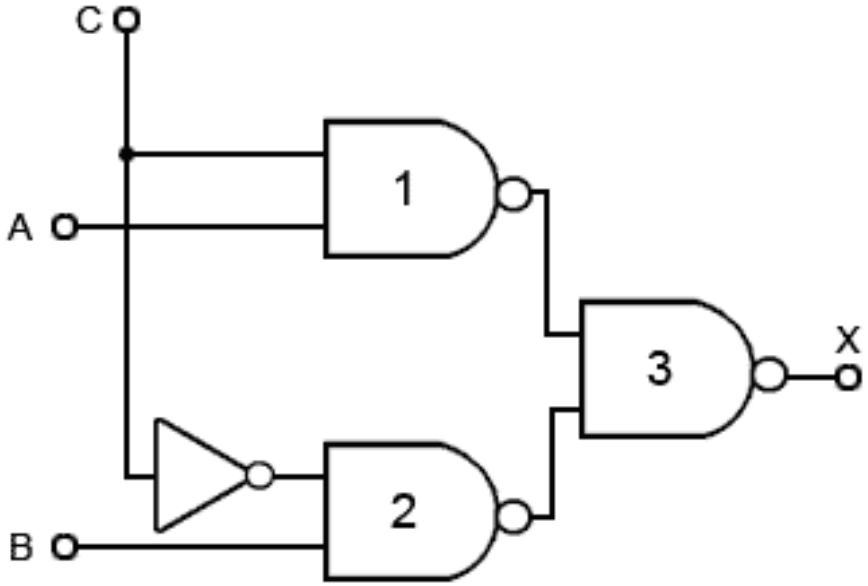


A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

C	B	A	X
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

https://learnabout-electronics.org/Digital/dig42.php

Multiplexer (Data selector)



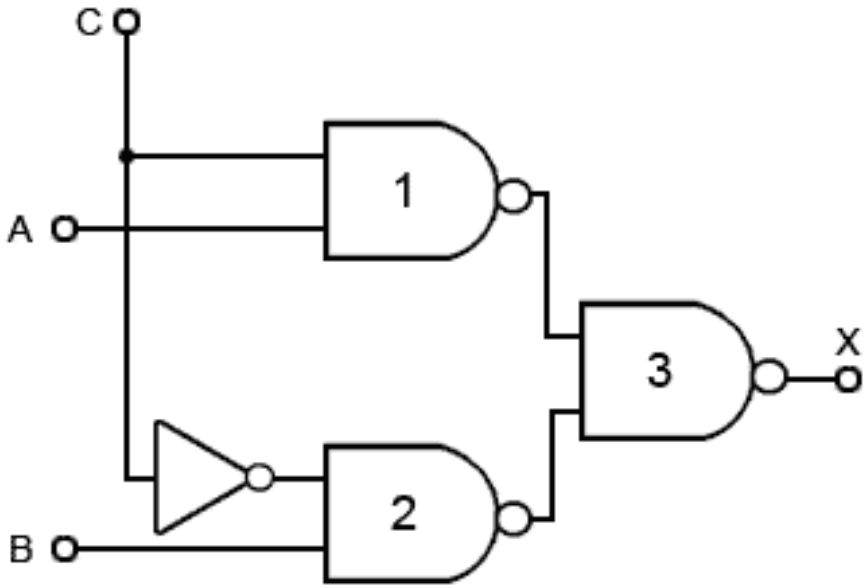
C = 0

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

C	B	A	X
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

https://learnabout-electronics.org/Digital/dig42.php

Multiplexer (Data selector)



C = 1

Selector

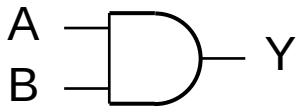
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

C	B	A	X
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

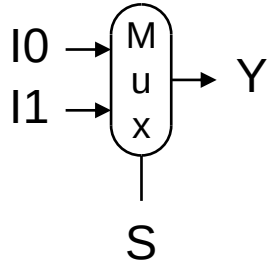
https://learnabout-electronics.org/Digital/dig42.php

Combinational Elements

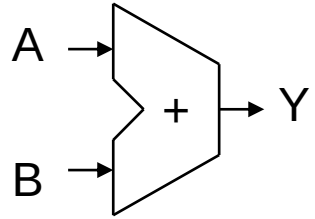
- AND-gate
  - Y = A & B



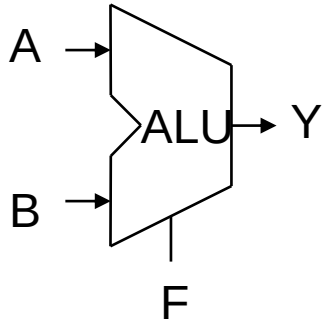
- Multiplexer
  - Y = S ? I1 : I0



- Adder
  - Y = A + B



- Arithmetic/Logic Unit
  - Y = F(A, B)



ALU

- Combines adder and And/OR logic gate

b. ALU

ALU

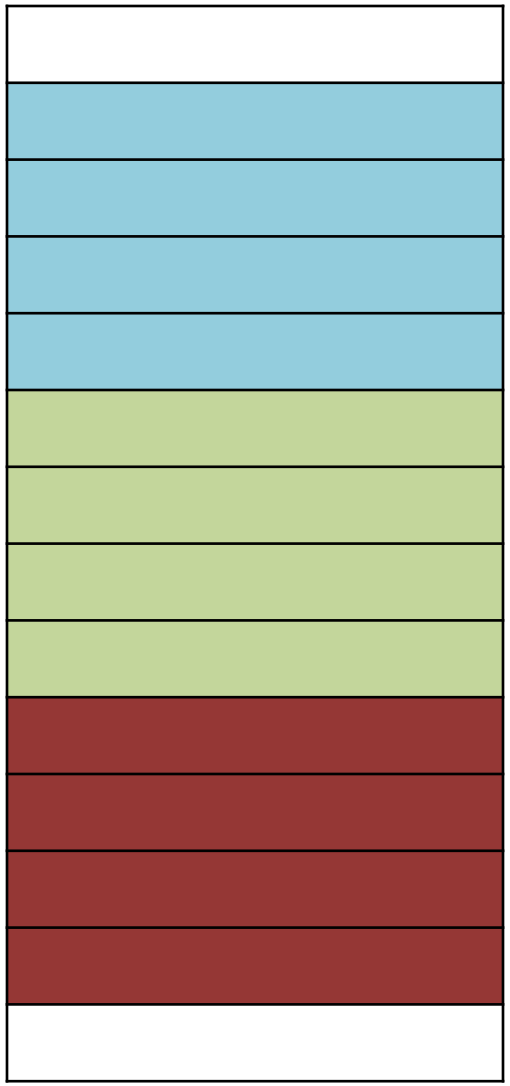
- Combines adder and And/OR logic gate

b. ALU

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

State Elements

Instruction Memory



ADD X0, X1, X2
32-bit

LDUR X3, [X2, #16]
32-bit

CBZ X0, Loop
32-bit

10

Memory (Byte address)

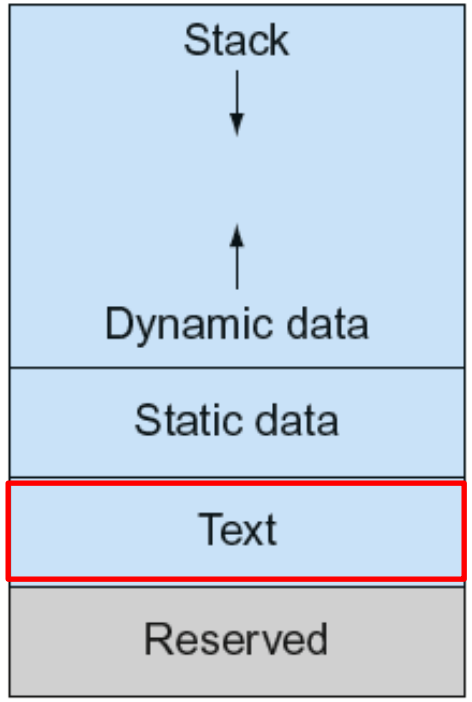
Instructions stored in memory

SP → 0000 007f ffff fffc$_{hex}$

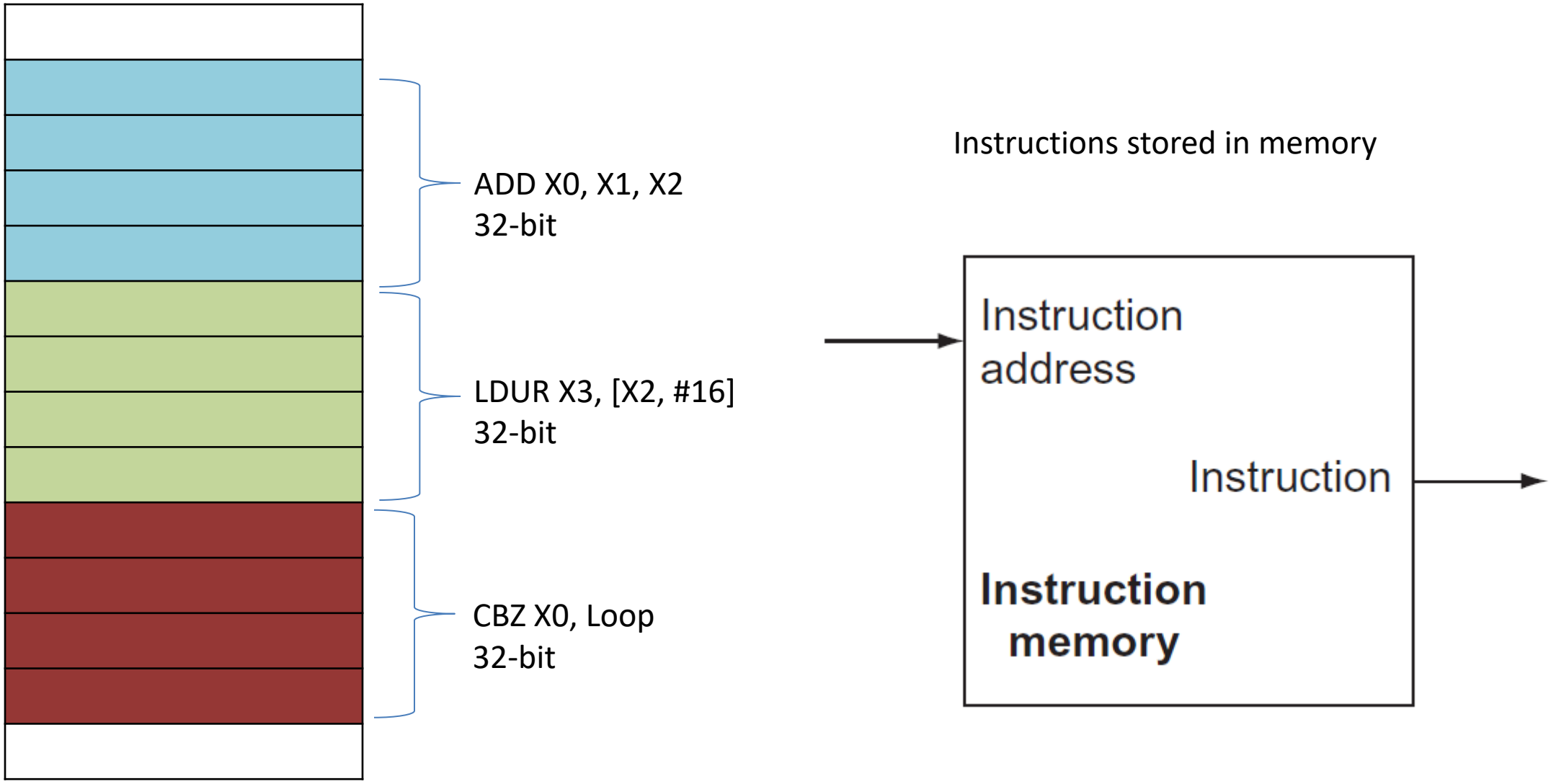
0000 0000 1000 0000$_{hex}$

PC → 0000 0000 0040 0000$_{hex}$

0



Instruction Memory



Instruction Memory

ADD X0, X1, X2
32-bit

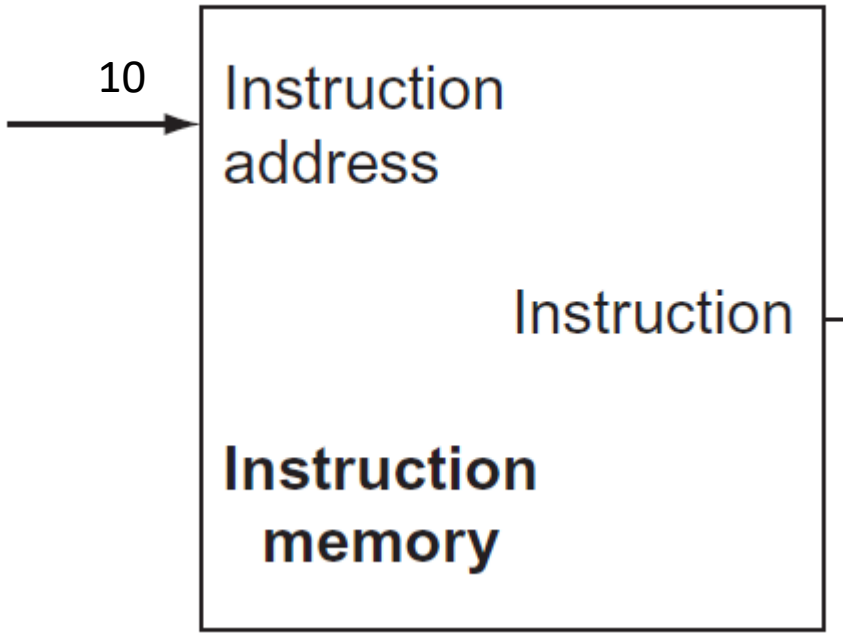
LDUR X3, [X2, #16]
32-bit

CBZ X0, Loop
32-bit

10

Memory (Byte address)

Instructions stored in memory



CBZ X0, Loop
32-bits

Register File

a. Registers

Register file:
1. Read values from registers
2. Write values to registers

Gated Latch

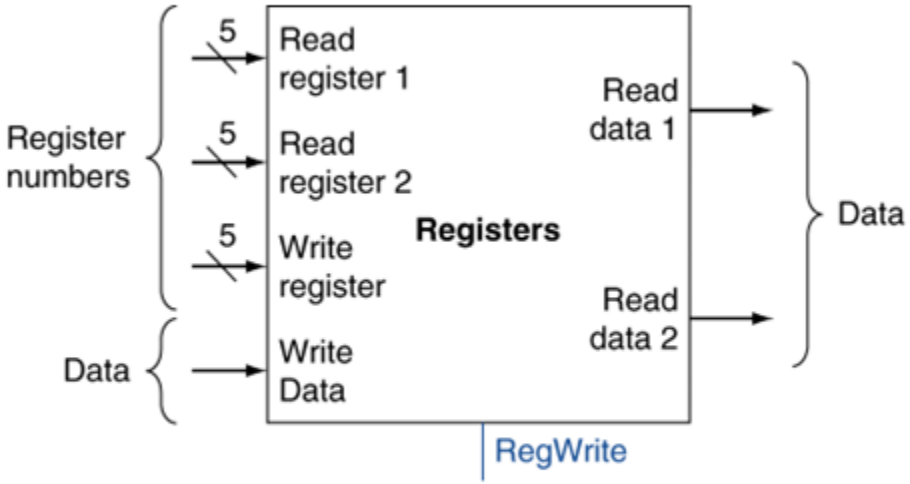
I	W	O
0	0	0
1	0	0 (No update)
0	**1**	0 (Same as I)
1	**1**	1 (Same as I)
0	0	1 (No update)
1	0	1 (No update)

Register File

a. Registers

Register file:
1. Read values from registers
2. Write values to registers

Register File



a. Registers

Register values

X0	1000
X1	1100
X2	1001
X3	
..	

Register file:
1. Read values from registers
2. Write values to registers

Register File

a. Registers

Register values

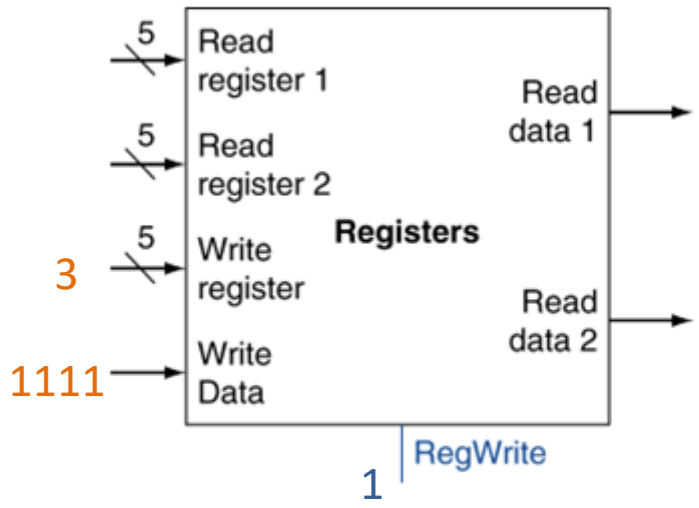
X0	1000
X1	1100
X2	1001
X3	
..	

Register file:

1. Read values from registers

2. Write values to registers

Register File



a. Registers

Register values

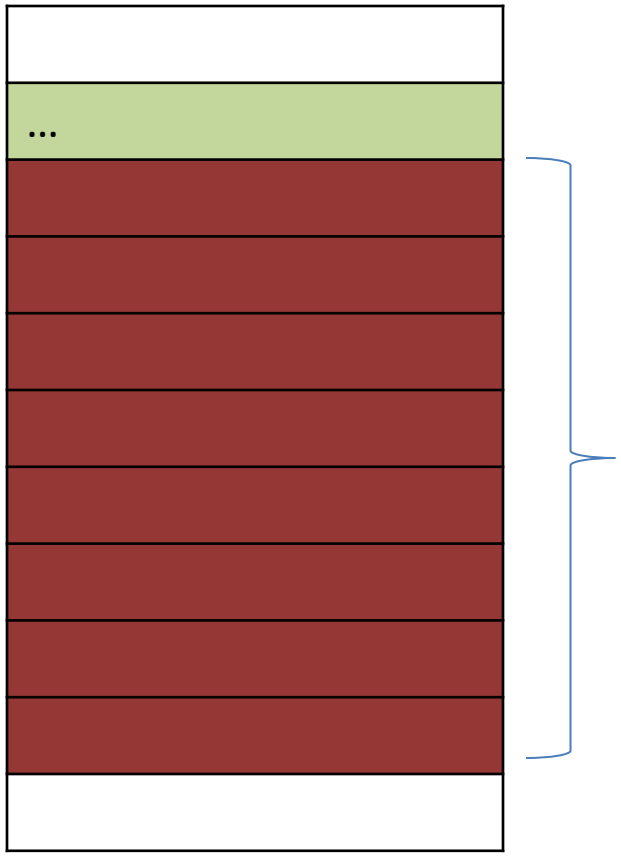
X0	1000
X1	1100
X2	1001
X3	1111
..	

Register file:

1. Read values from registers
2. **Write values to registers**

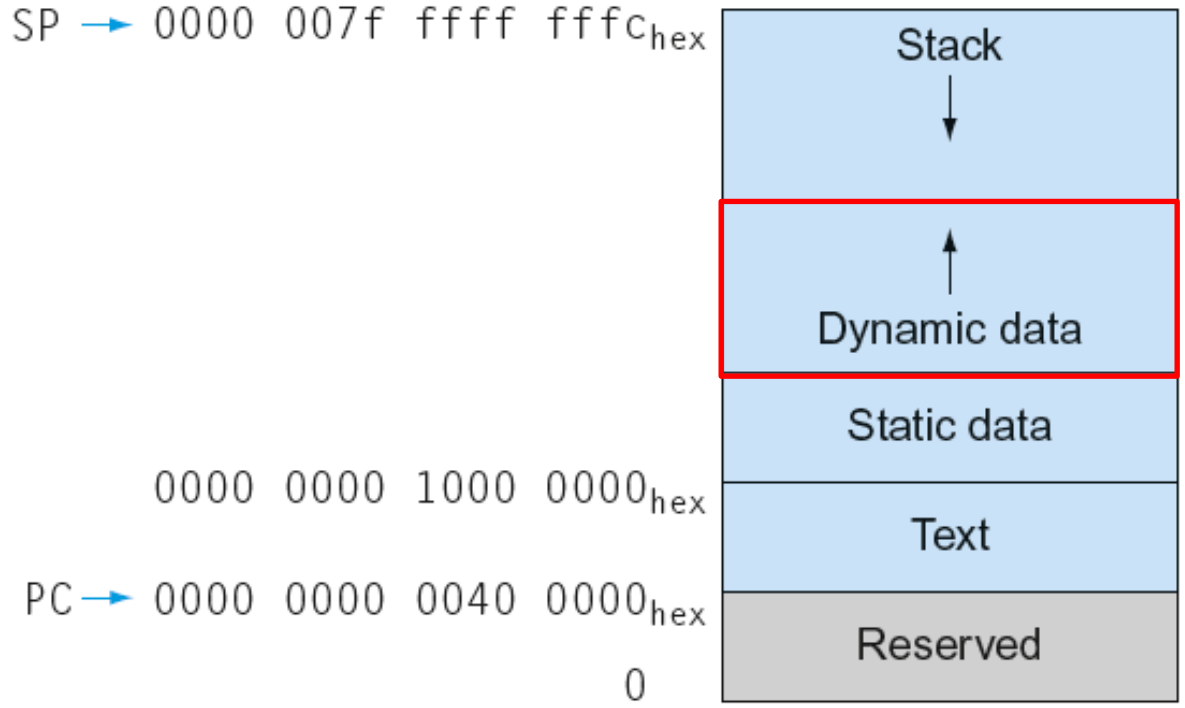
Data Memory

Data Memory



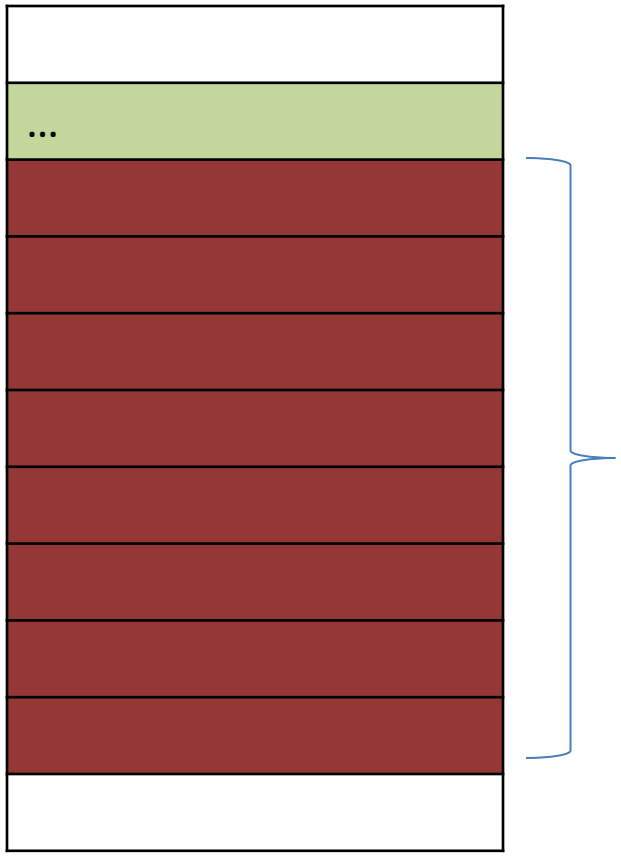
Integer 24
64-bits

Data stored in memory



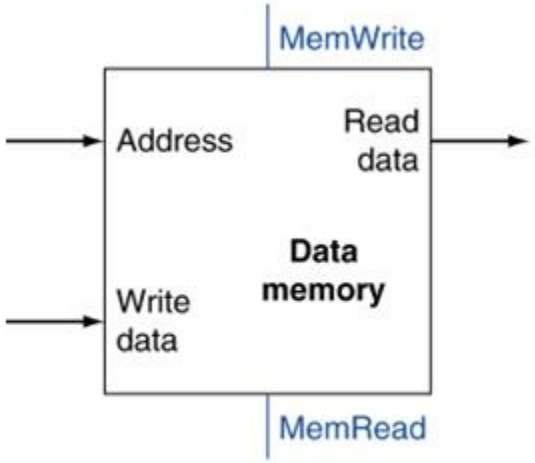
DataMemory (Byte address)

Data Memory



Integer 24
64-bits

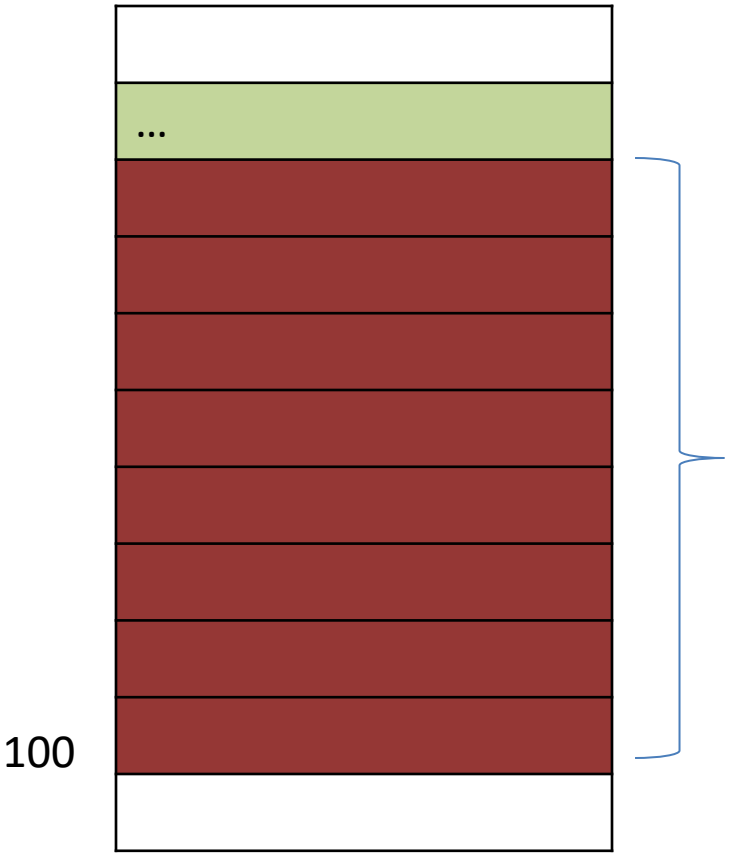
Data stored in memory



a. Data memory unit

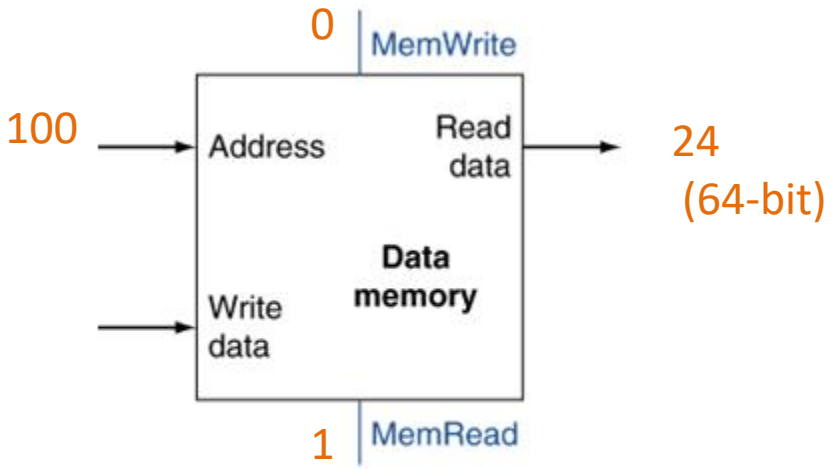
DataMemory (Byte address)

Data Memory



Integer 24
64-bits

Data stored in memory



a. Data memory unit

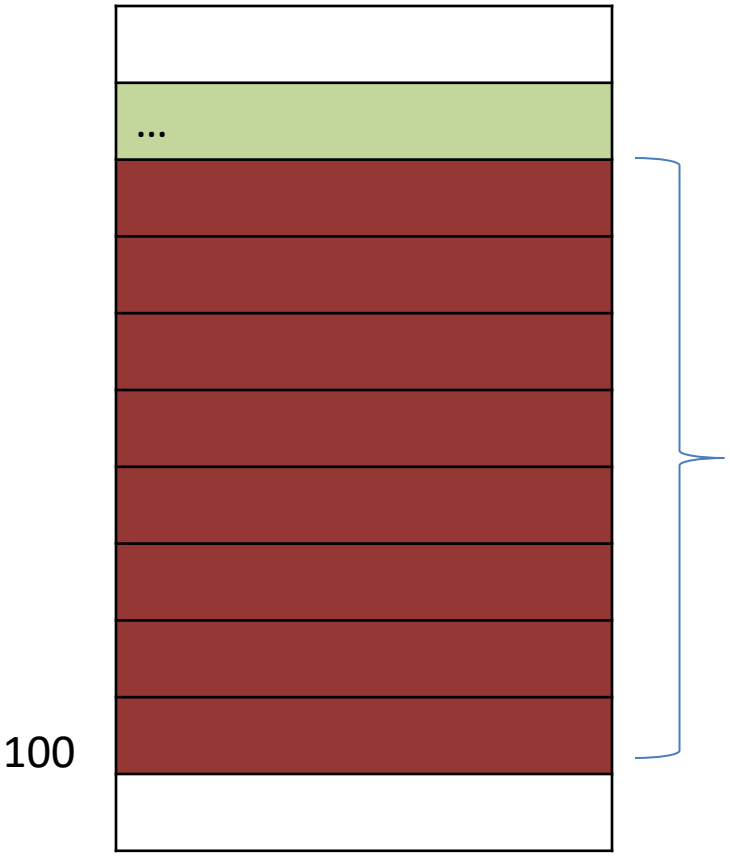
Register file:

1. Read values from memory

2. Write values to Memory

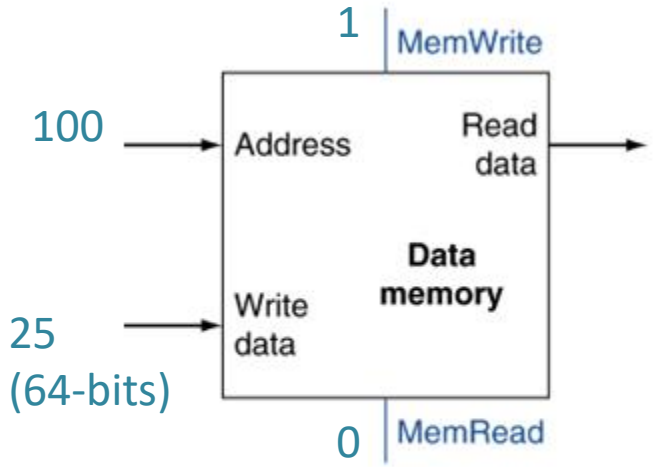
DataMemory (Byte address)

Data Memory



Integer **25**
64-bits

Data stored in memory



a. Data memory unit

Register file:
1. Read values from memory
2. Write values to Memory

DataMemory (Byte address)

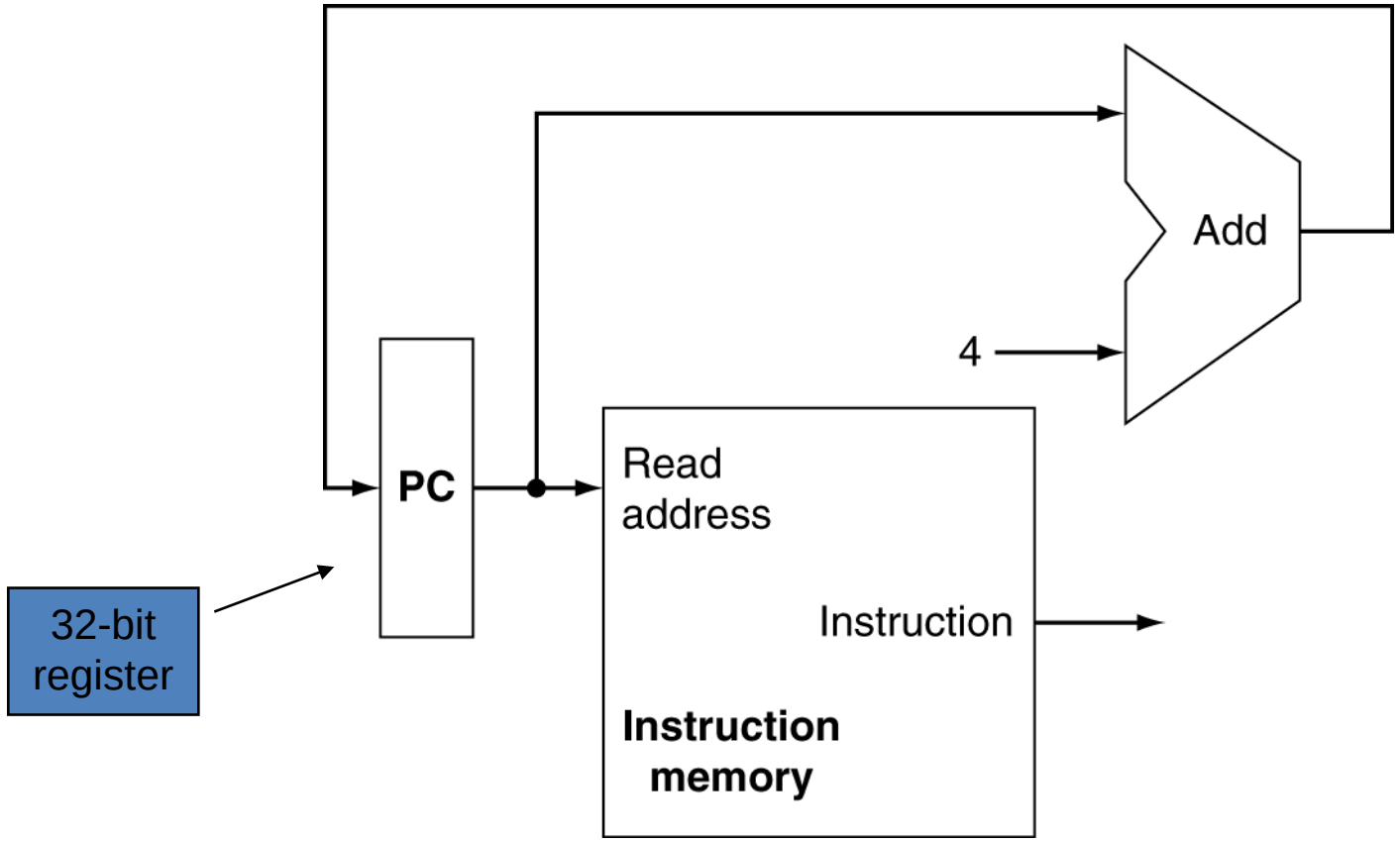
Building a Datapath

- Datapath
  - Elements that process data and addresses in the CPU
    - Registers, ALUs, mux's, memories, …
- We will build a LEGv8 datapath incrementally
  - **Fetch Instruction**
  - Execute Instruction
    - ADD, LDUR/STUR, CBZ

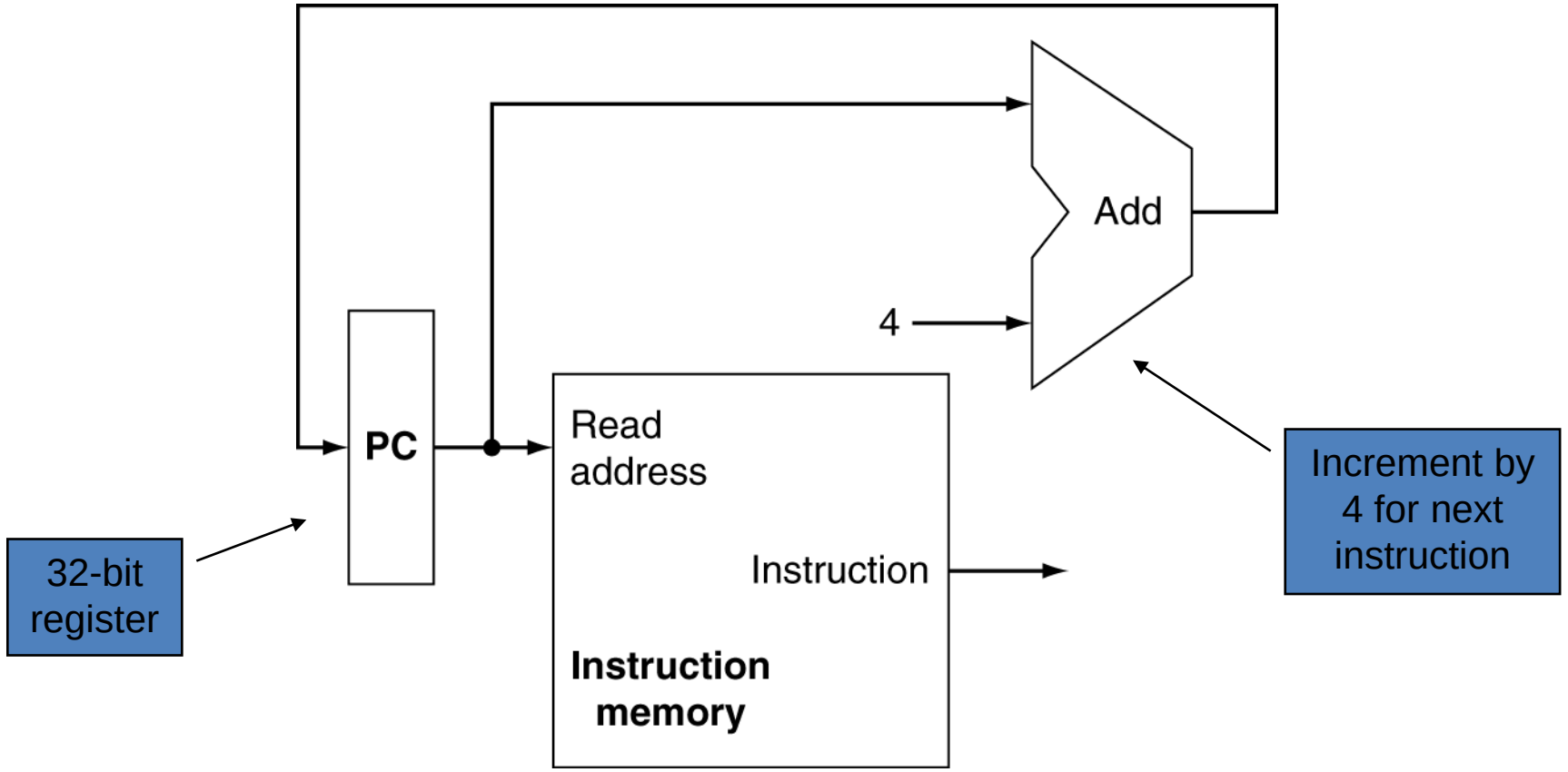
What Registers used?

- X0 – X7: procedure arguments/results
- X9 – X15: temporaries registers
- X19 – X27: saved registers
- X28 (SP): stack pointer (address of the most recently allocated stack)
- X29 (FP): frame pointer
- **X30 (PC): Program Counter**
  - **Address of the current Instructions**
  - **LR, link register**

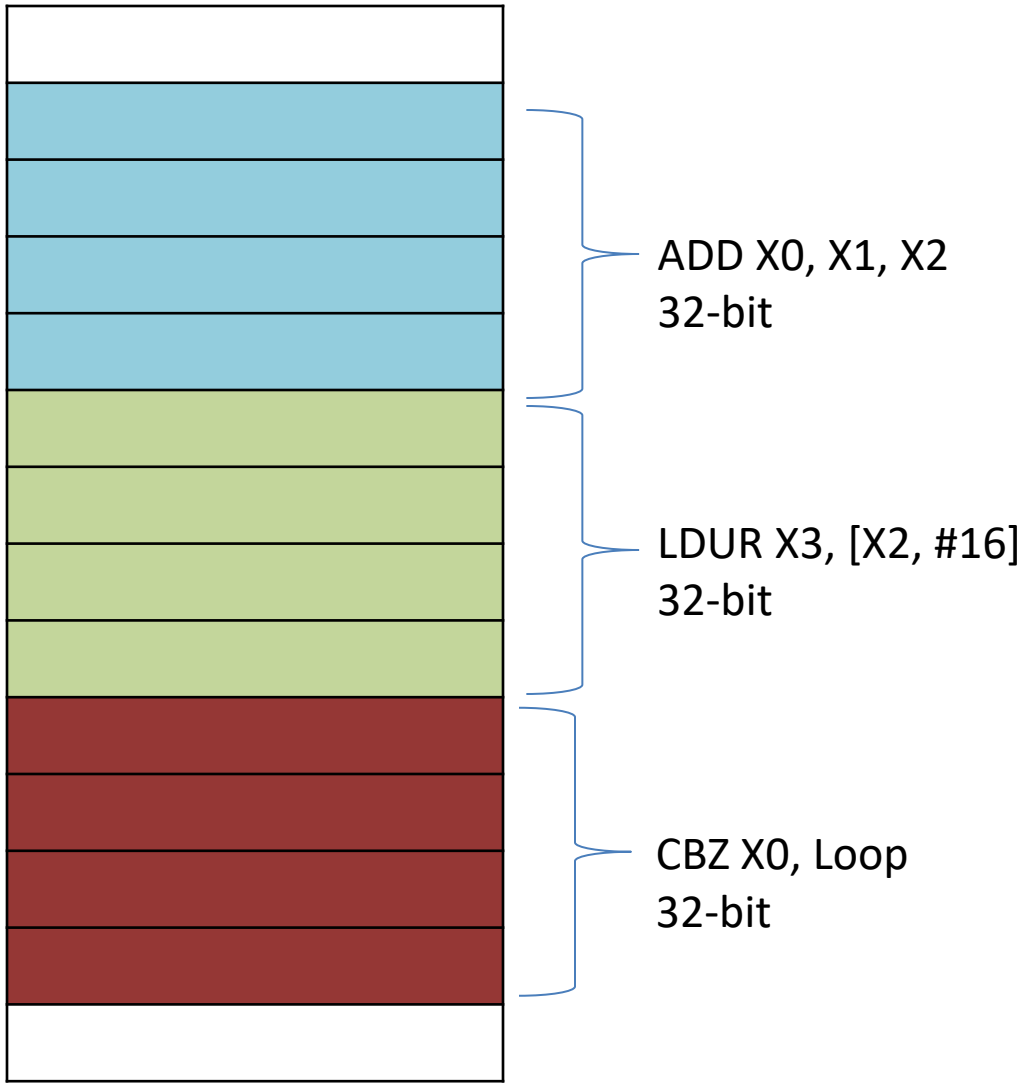
Instruction Fetch

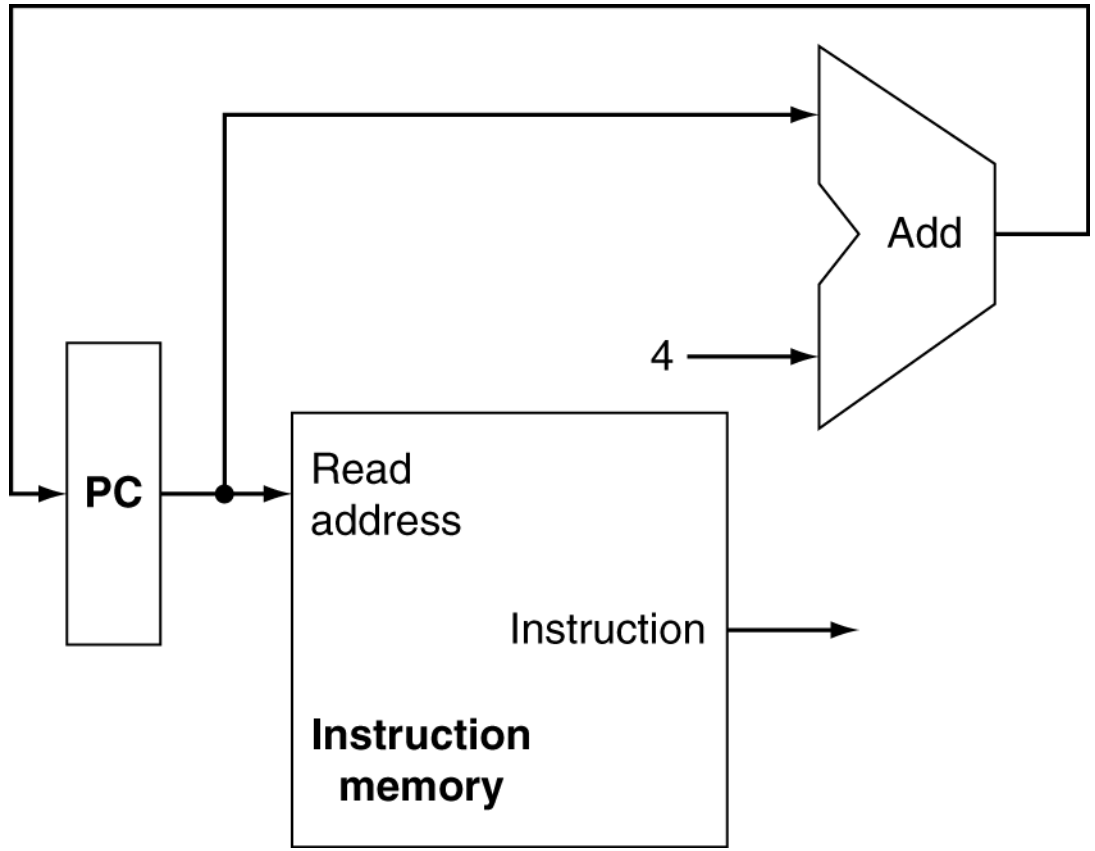


Instruction Fetch

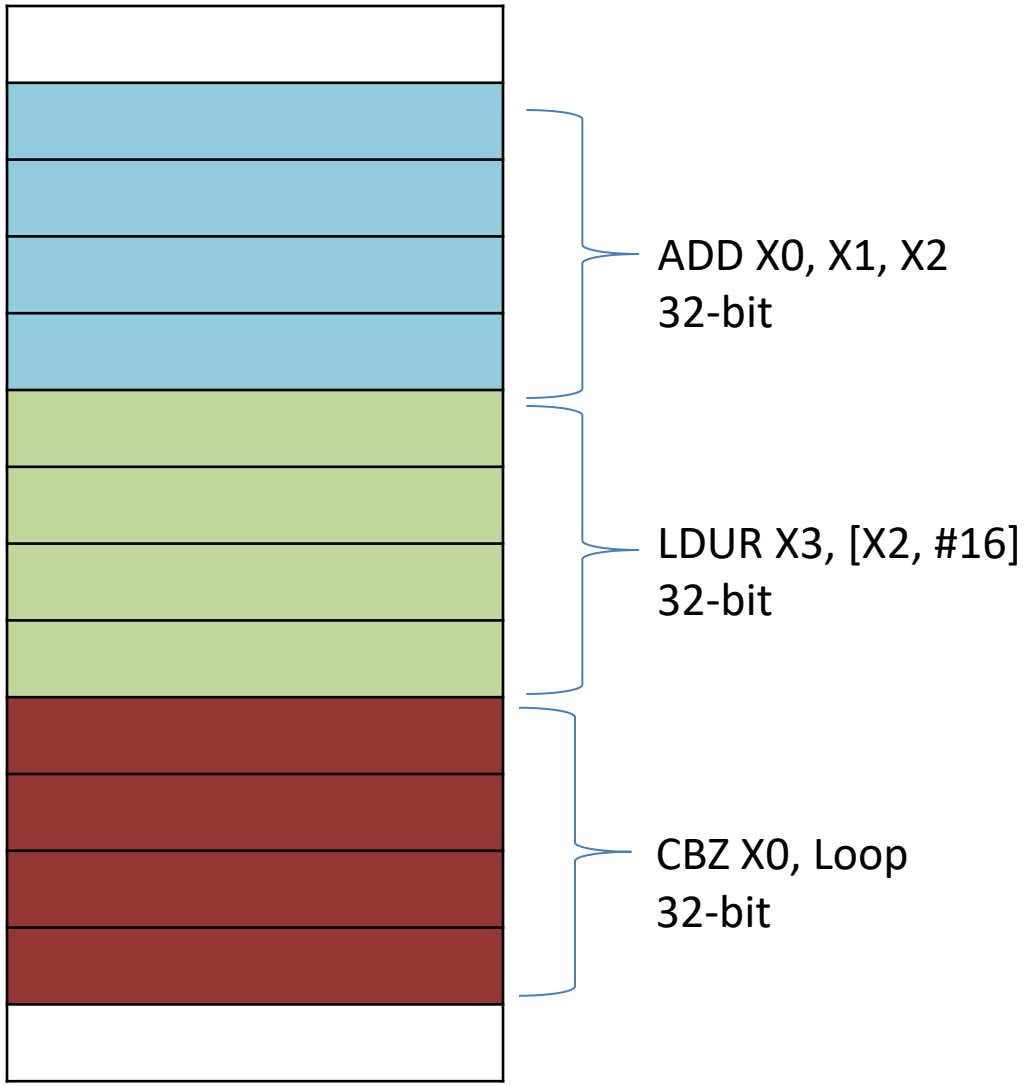


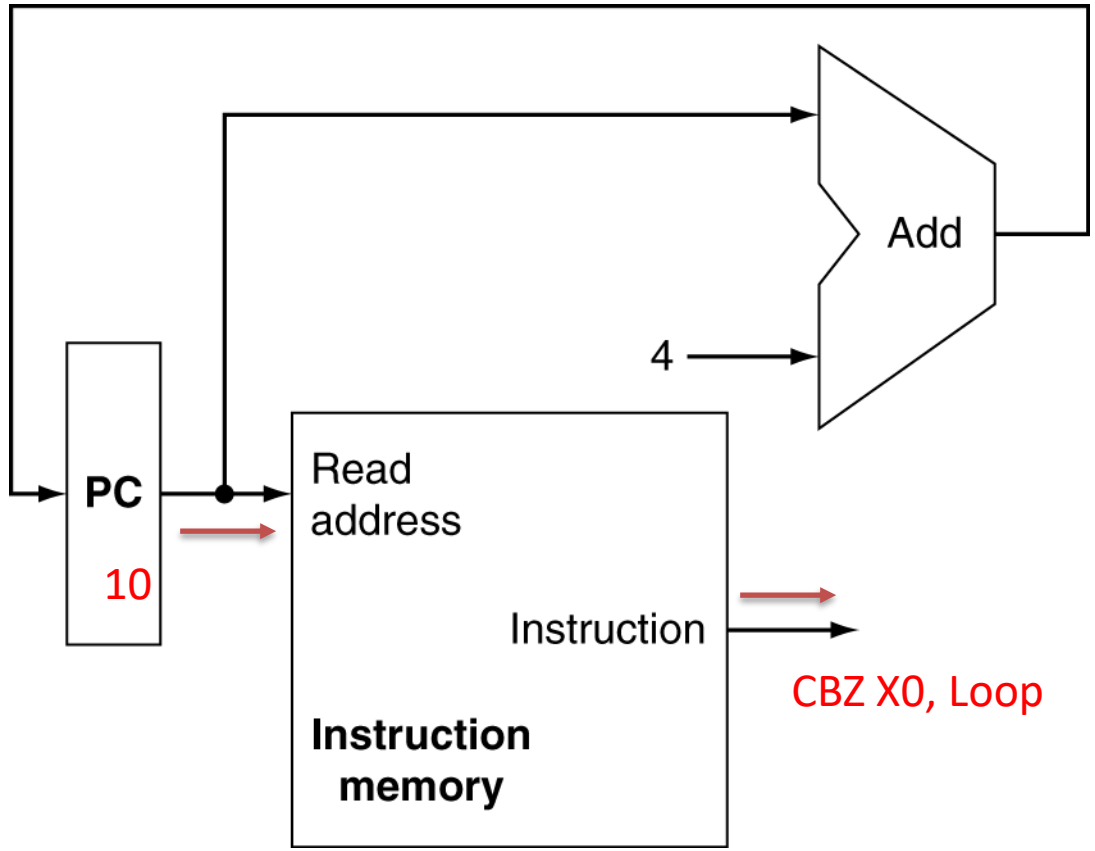
Fetch Instruction



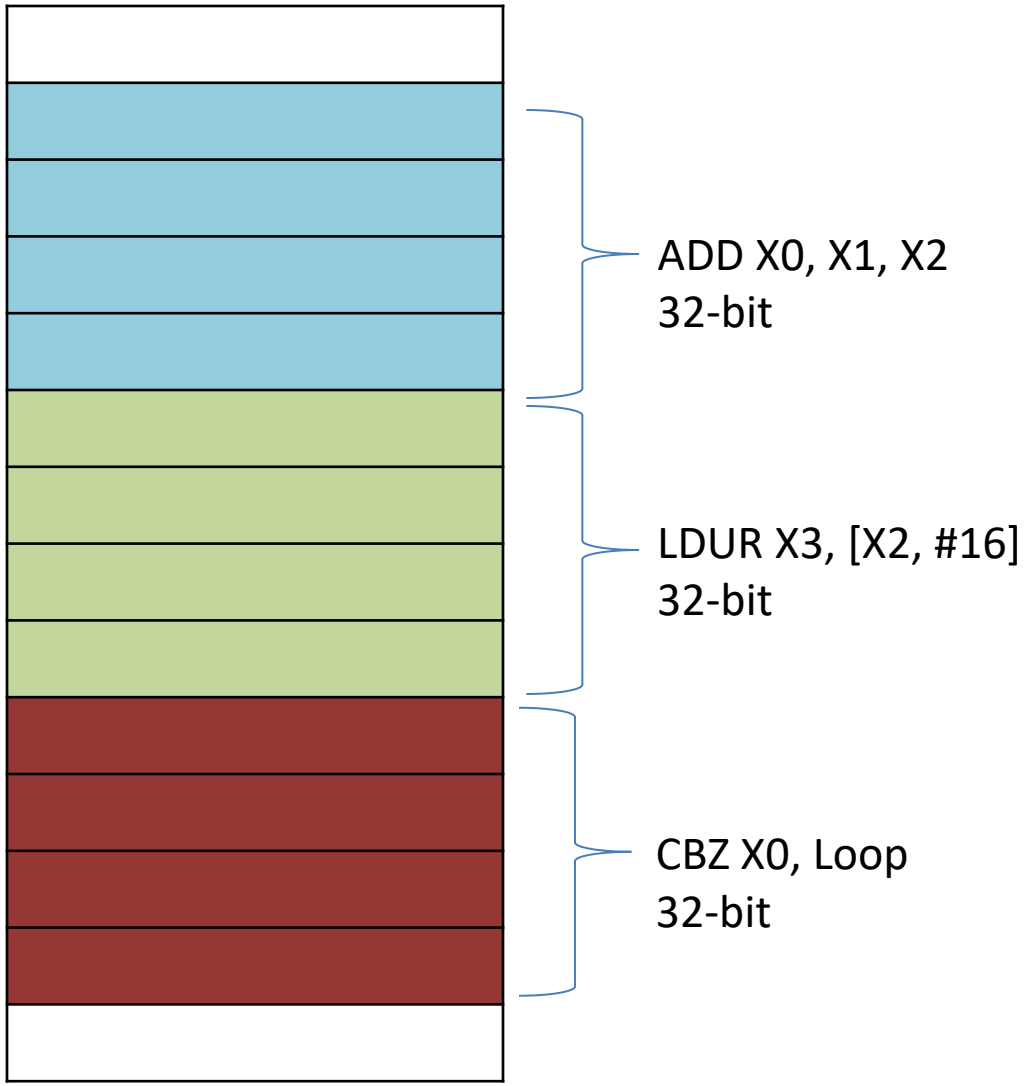


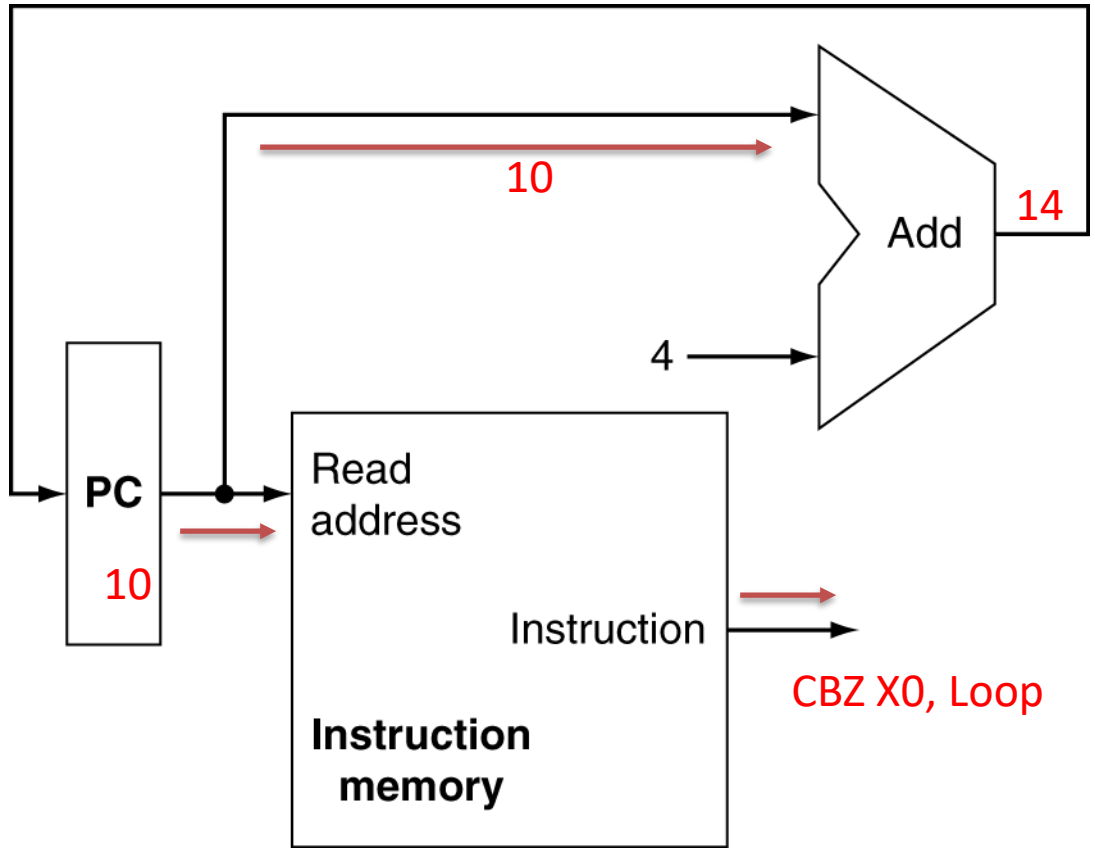
Fetch Instruction



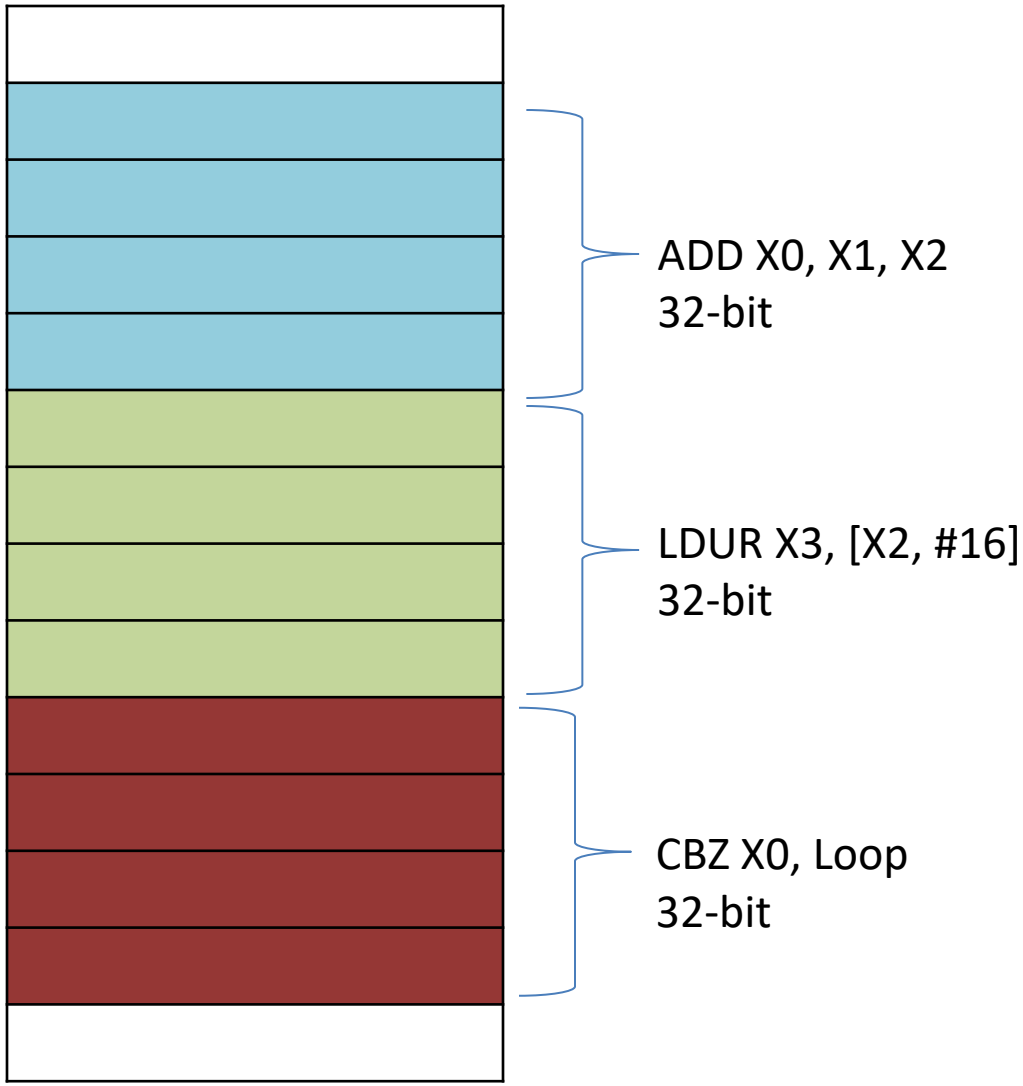


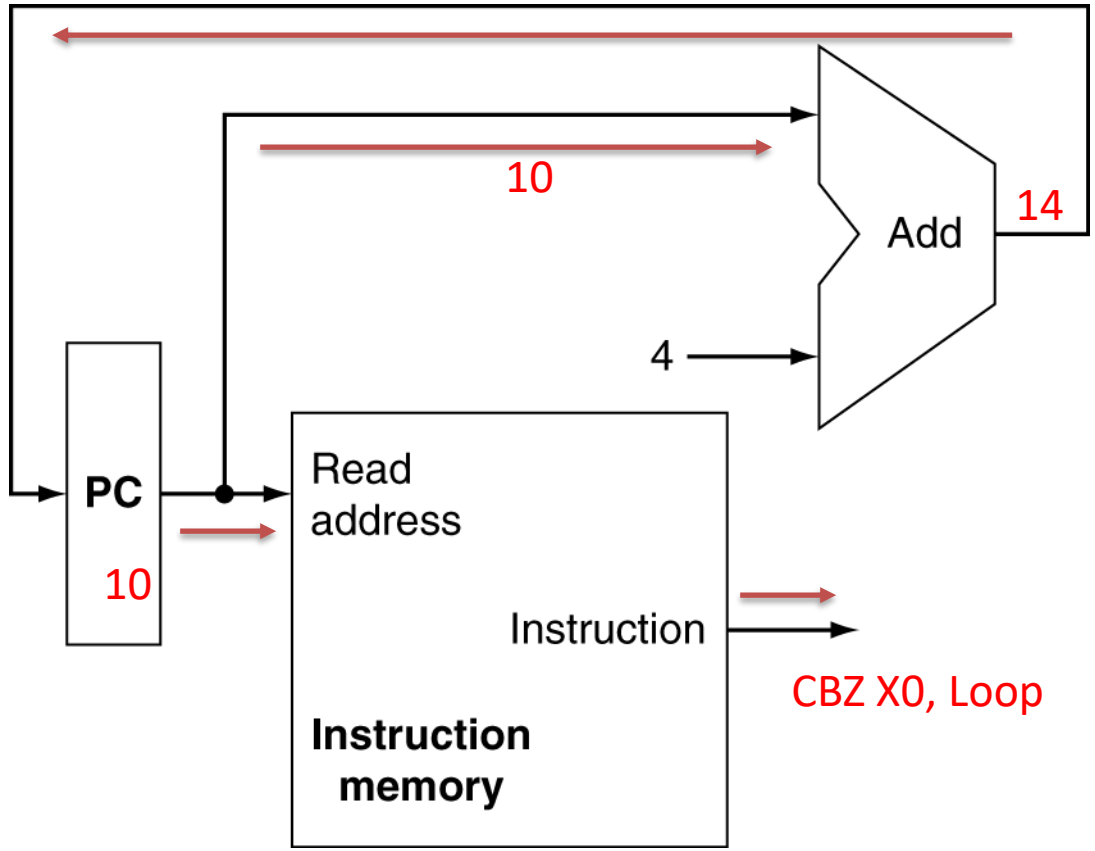
Fetch Instruction



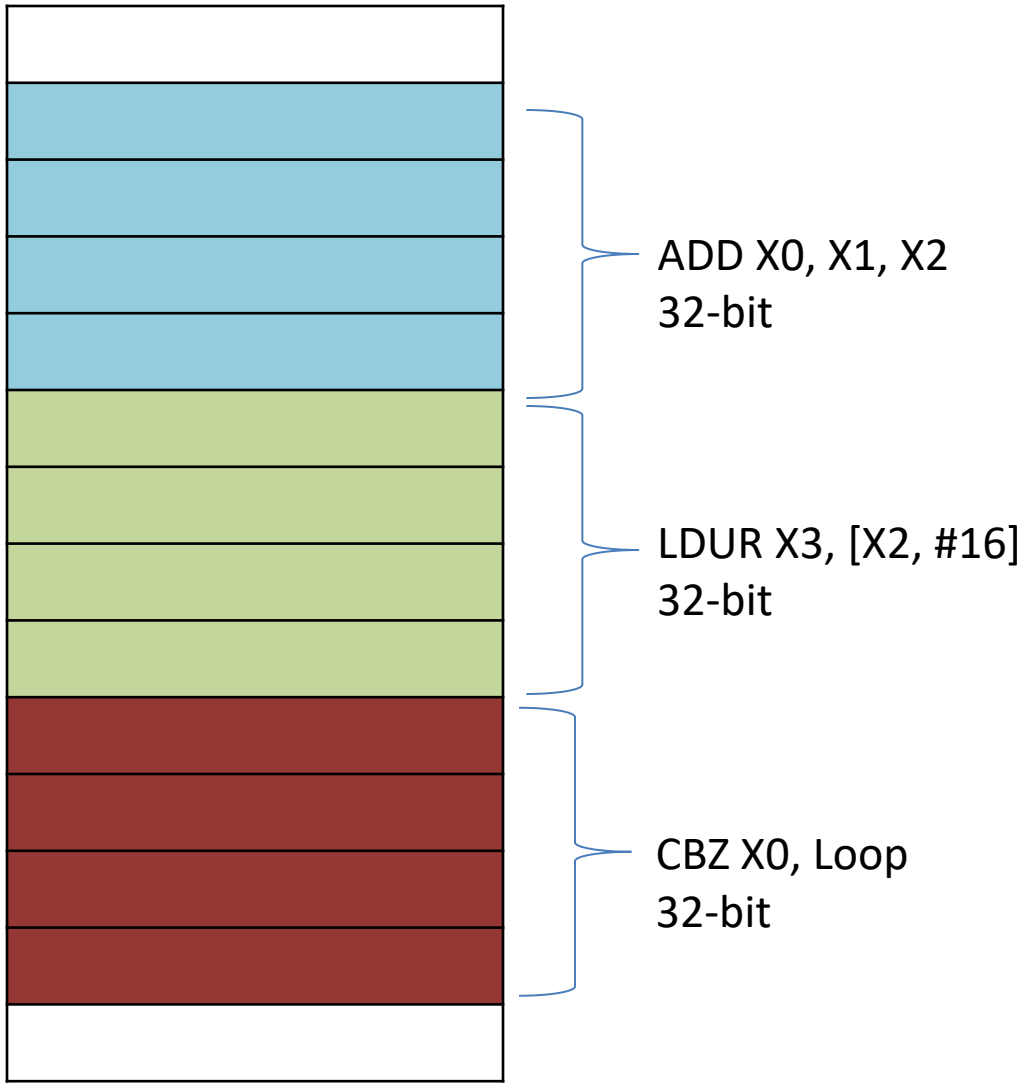


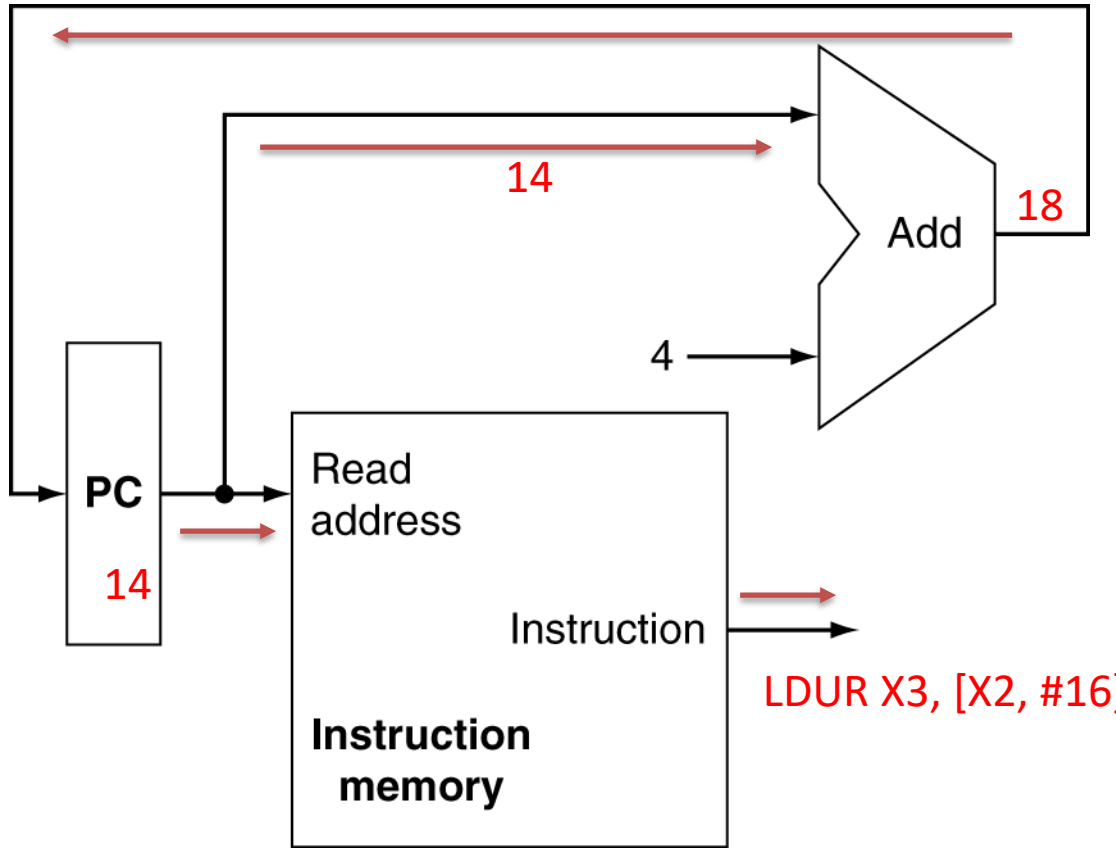
Fetch Instruction



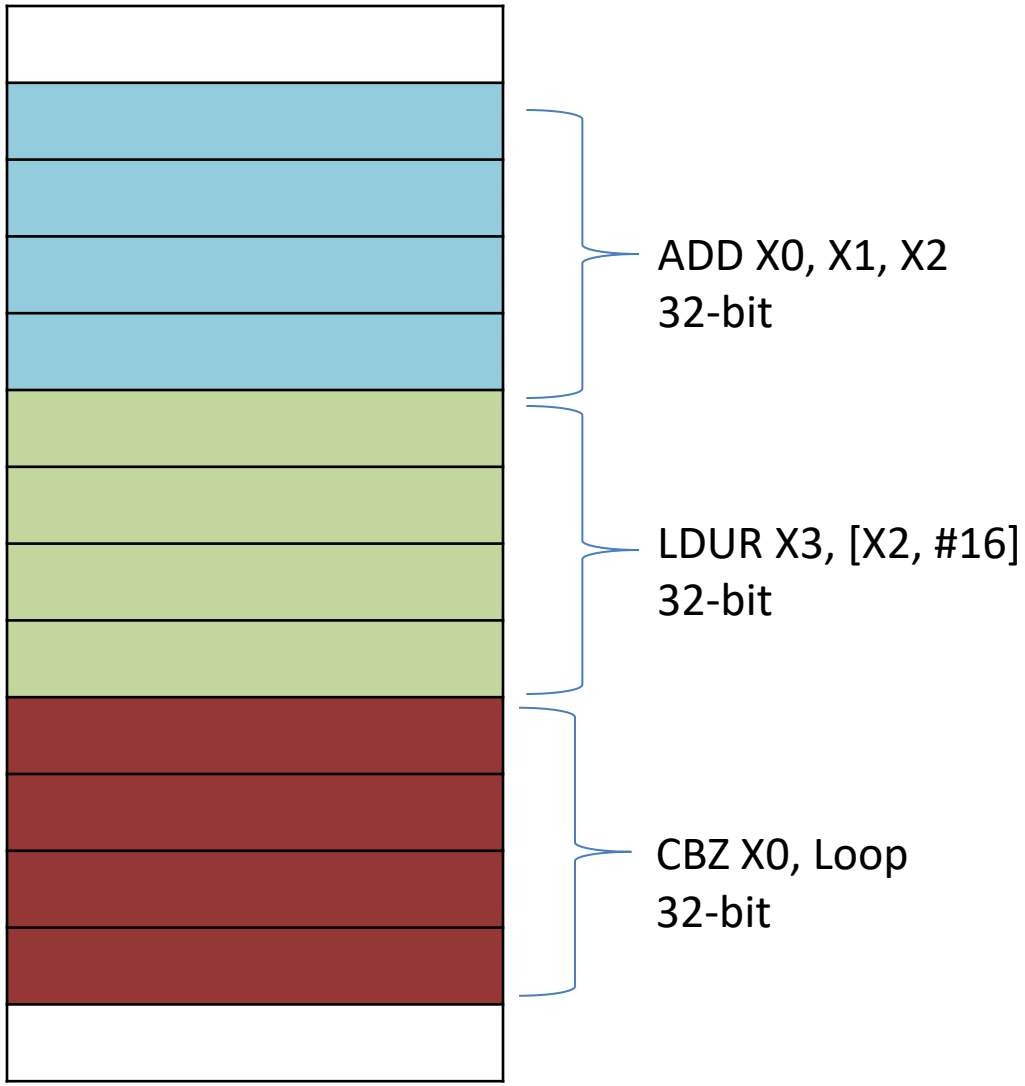


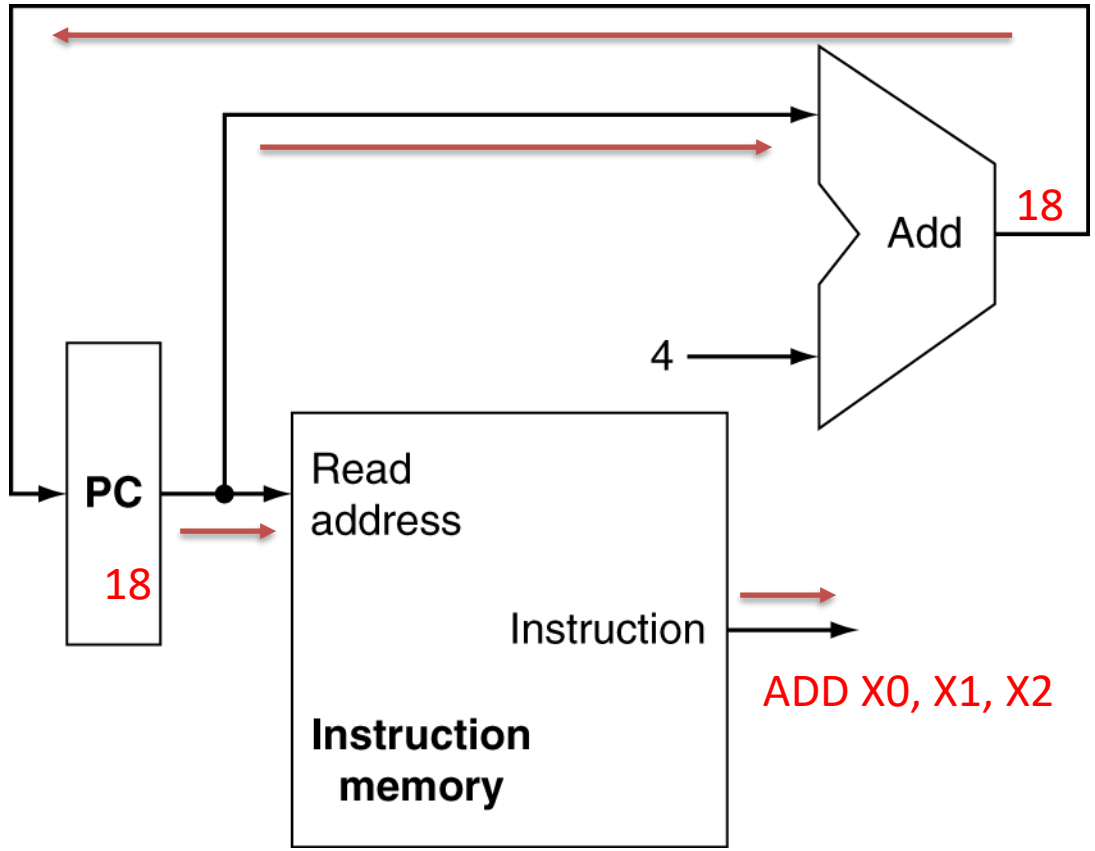
Fetch Instruction





Fetch Instruction

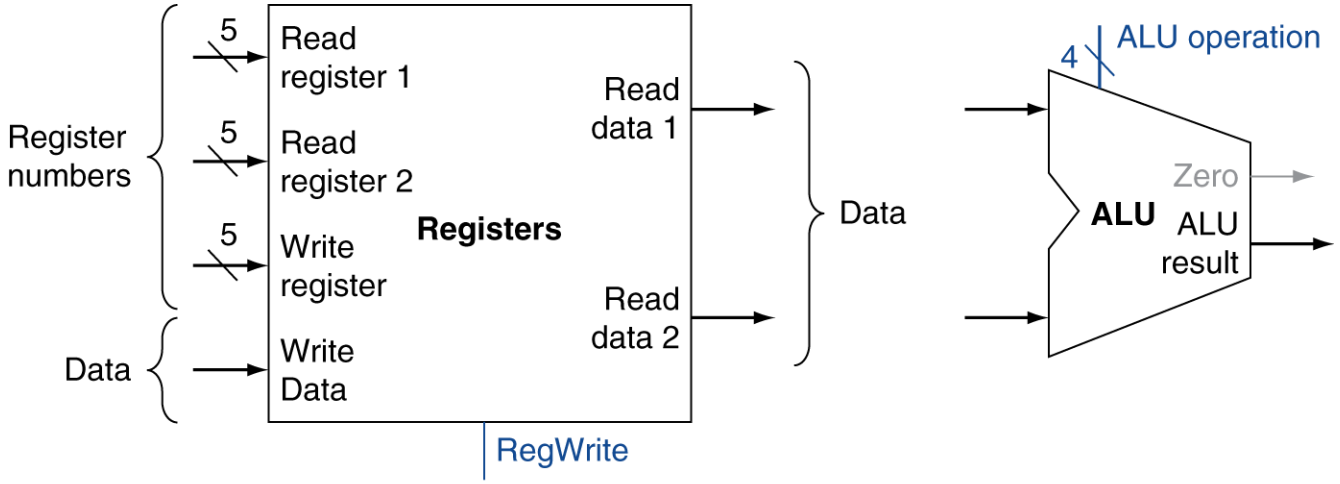




ADD (R-Type) Instruction

ADD (R-Type) Instruction

- EG. `ADD X3, X1, X2`
- Read two register operands
- Perform arithmetic/logical operation
- Write register result



a. Registers

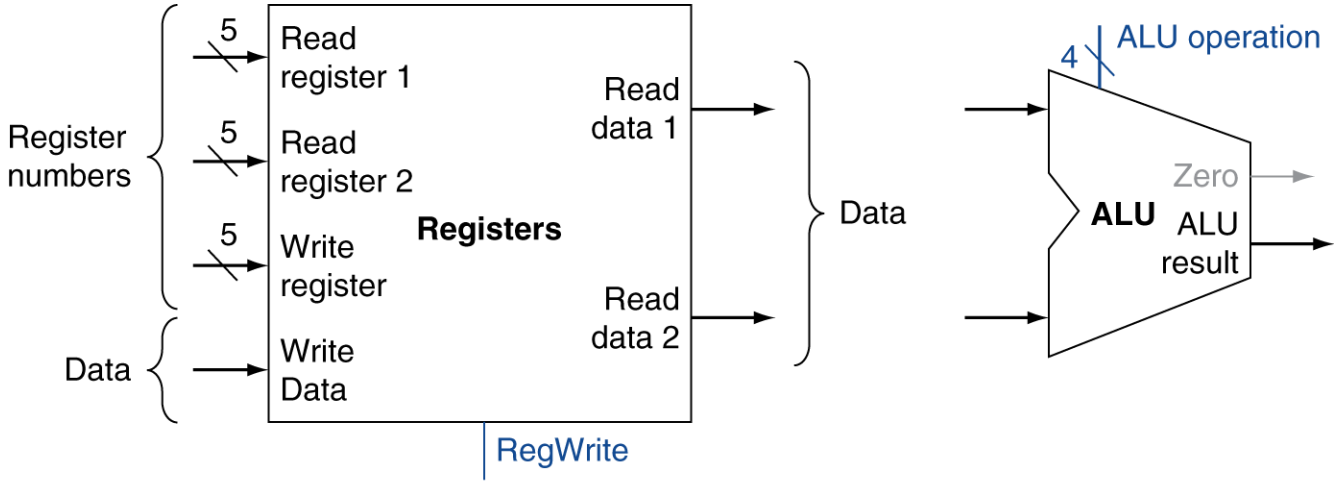
b. ALU

ADD (R-Type) Instruction

- EG. `ADD X3, X1, X2`
- Read two register operands
- Perform arithmetic/logical operation
- Write register result

Register values

X0	1000
X1	1100
X2	1001
X3	
..	



a. Registers

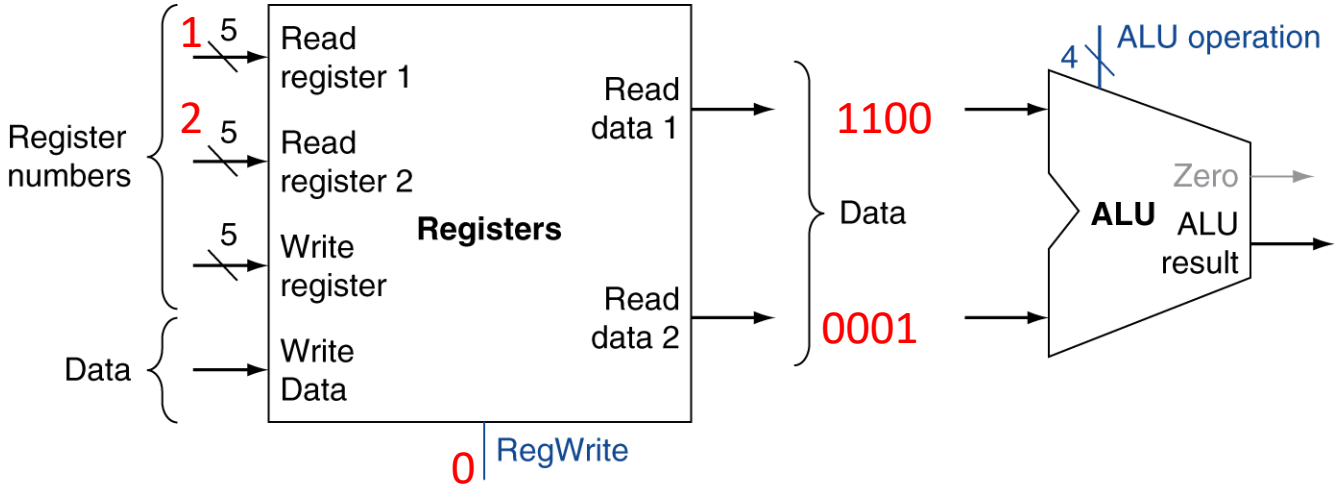
b. ALU

ADD (R-Type) Instruction

- EG. `ADD X3, X1, X2`
- Read two register operands
- Perform arithmetic/logical operation
- Write register result

Register values

X0	1000
X1	1100
X2	0001
X3	
..	



a. Registers

b. ALU

ALU

- Combines adder and And/OR logic gate

b. ALU

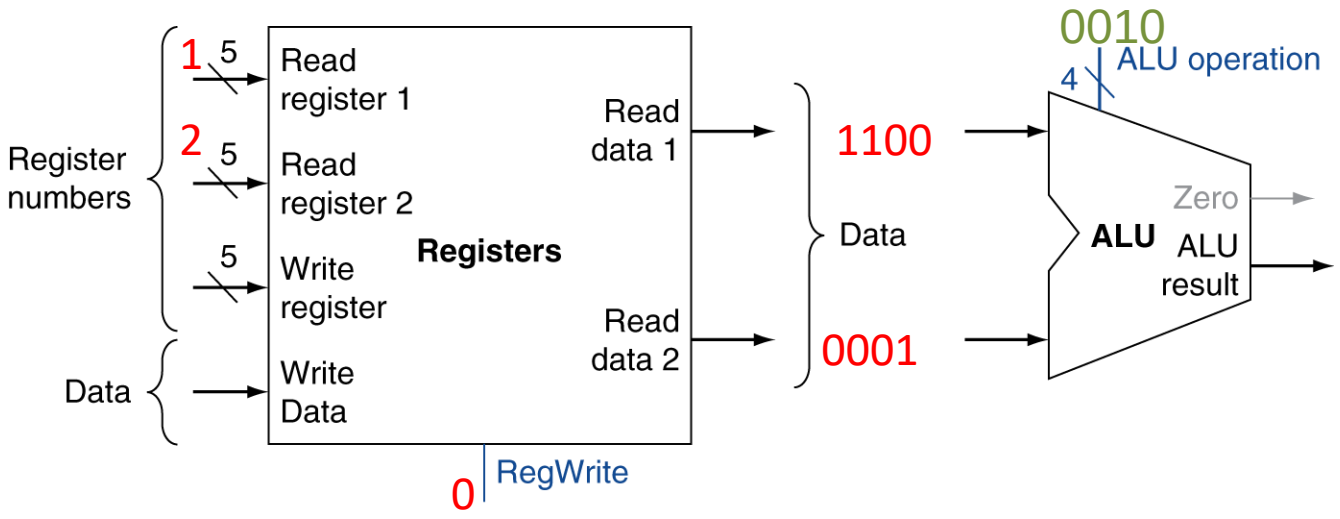
ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	pass input b
1100	NOR

ADD (R-Type) Instruction

- EG. `ADD X3, X1, X2`
- Read two register operands
- Perform arithmetic/logical operation
- Write register result

Register values

X0	1000
X1	1100
X2	0001
X3	
..	



a. Registers

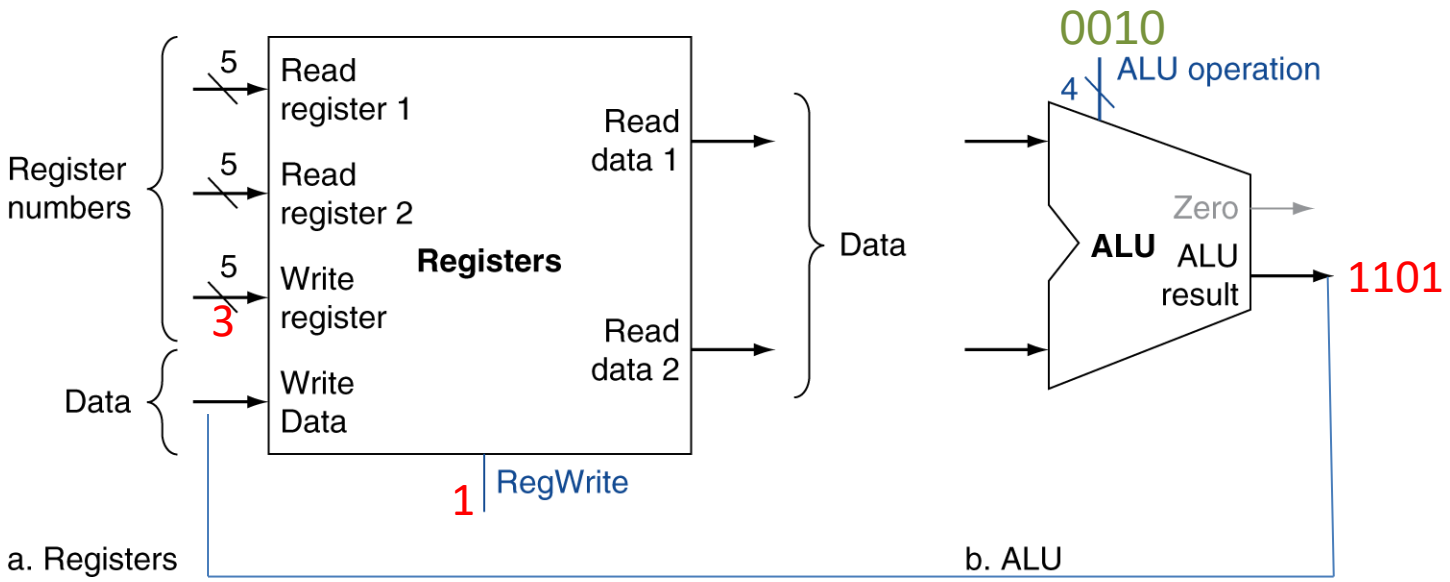
b. ALU

ADD (R-Type) Instruction

- EG. `ADD X3, X1, X2`
- Read two register operands
- Perform arithmetic/logical operation
- Write register result

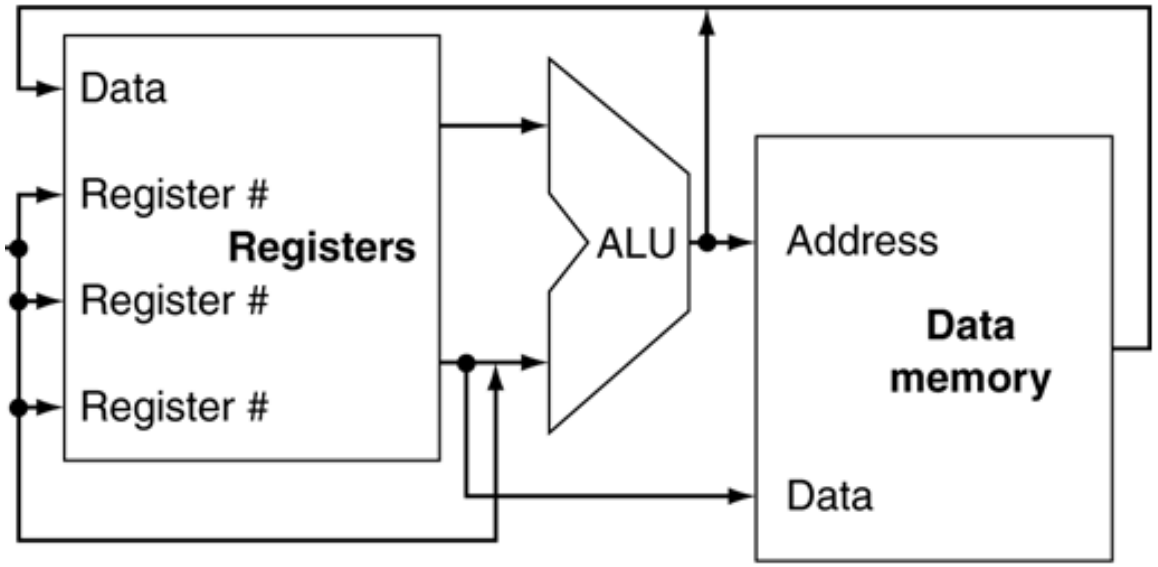
Register values

X0	1000
X1	1100
X2	0001
X3	1101
..	



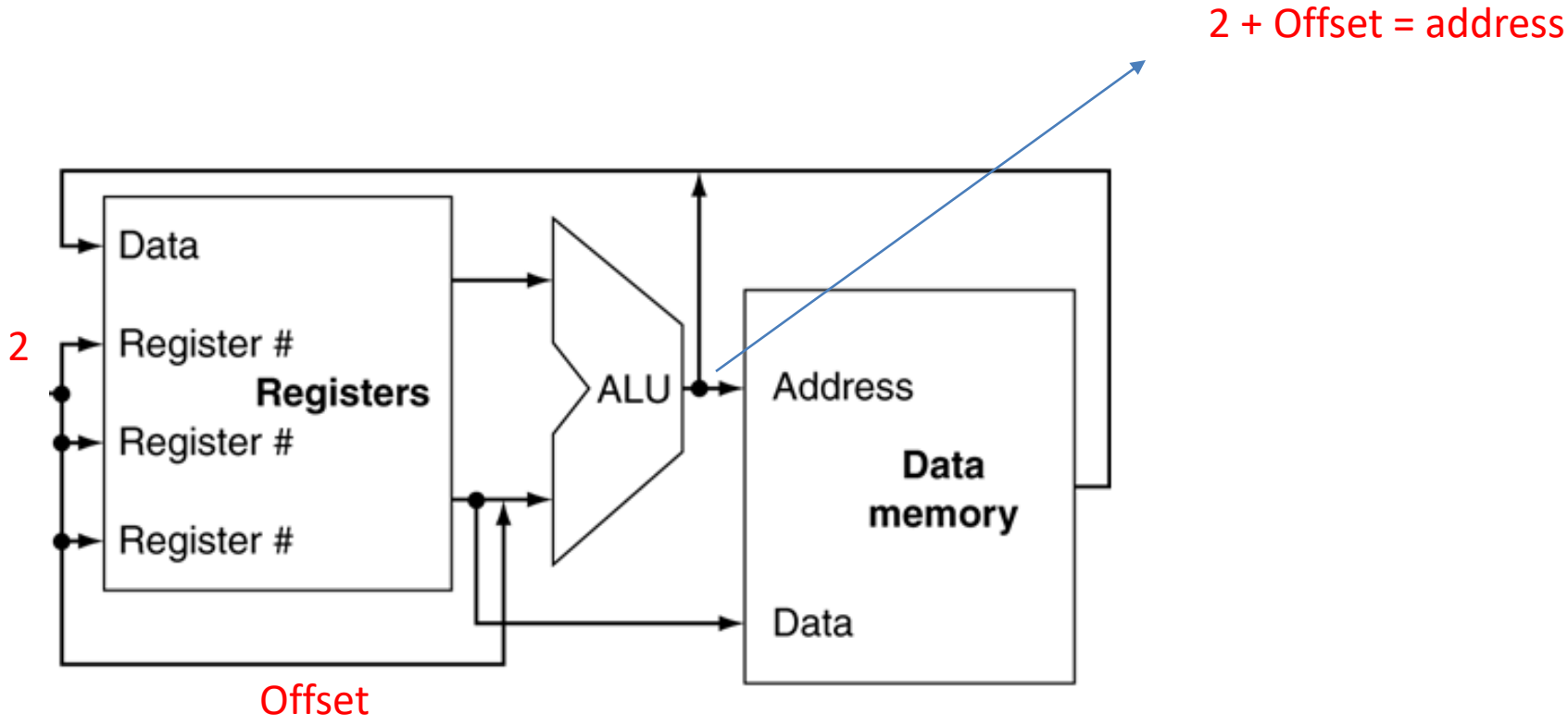
Load/Store Instruction (D-Type)

- `LDUR X1, [X2, #offset]`



Load/Store Instruction (D-Type)

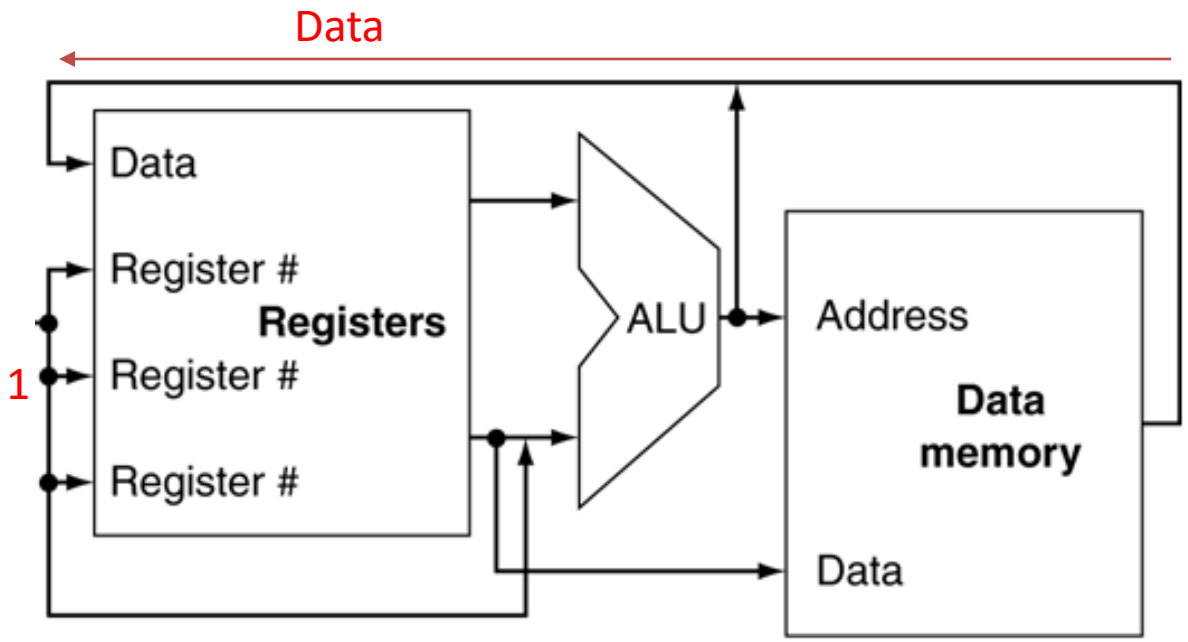
- `LDUR X1, [X2, #offset]`



Needs to be sign extended

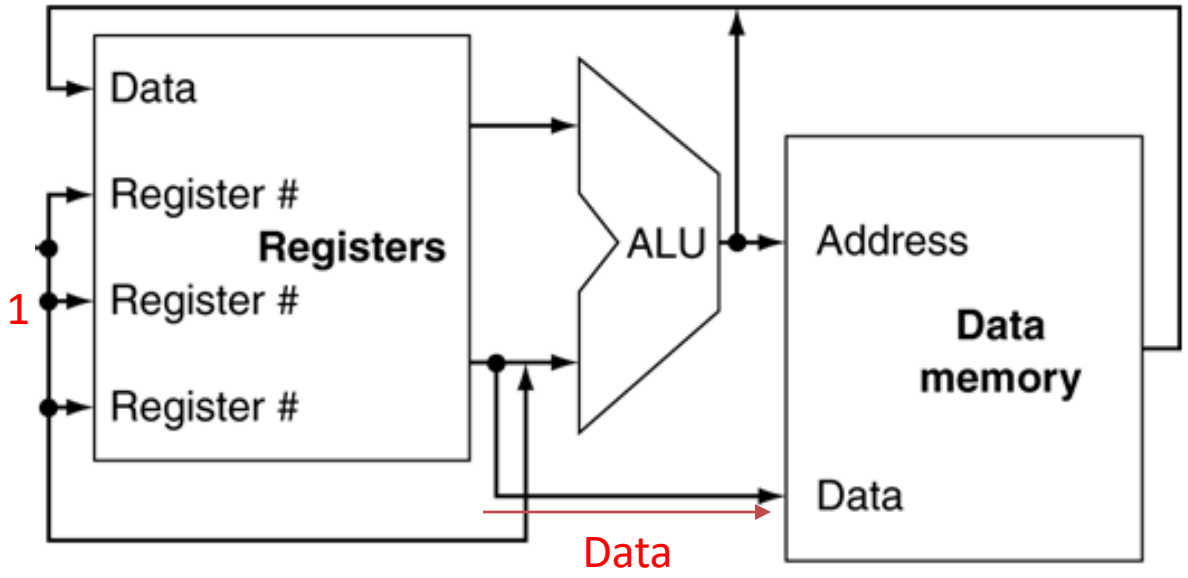
Load/Store Instruction (D-Type)

- `LDUR X1, [X2, #offset]`

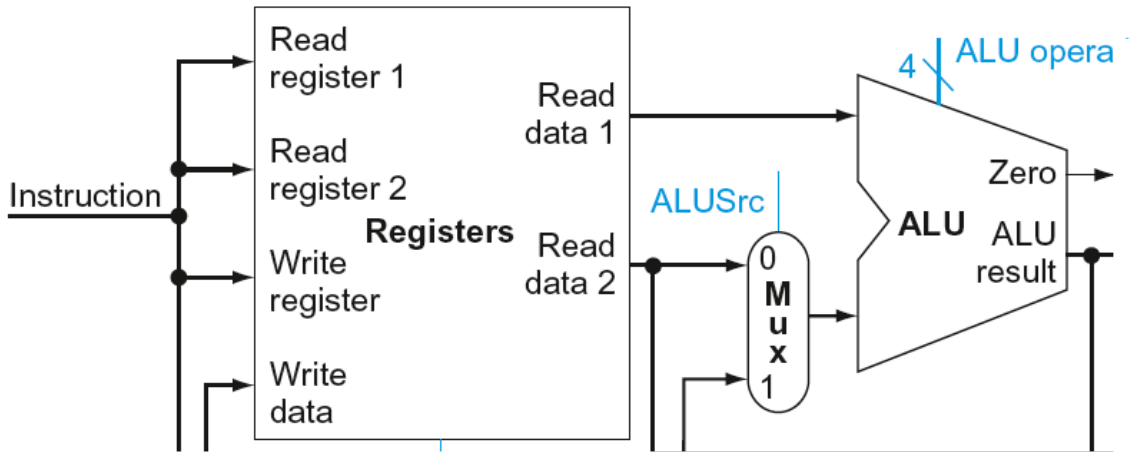


Load/Store Instruction (D-Type)

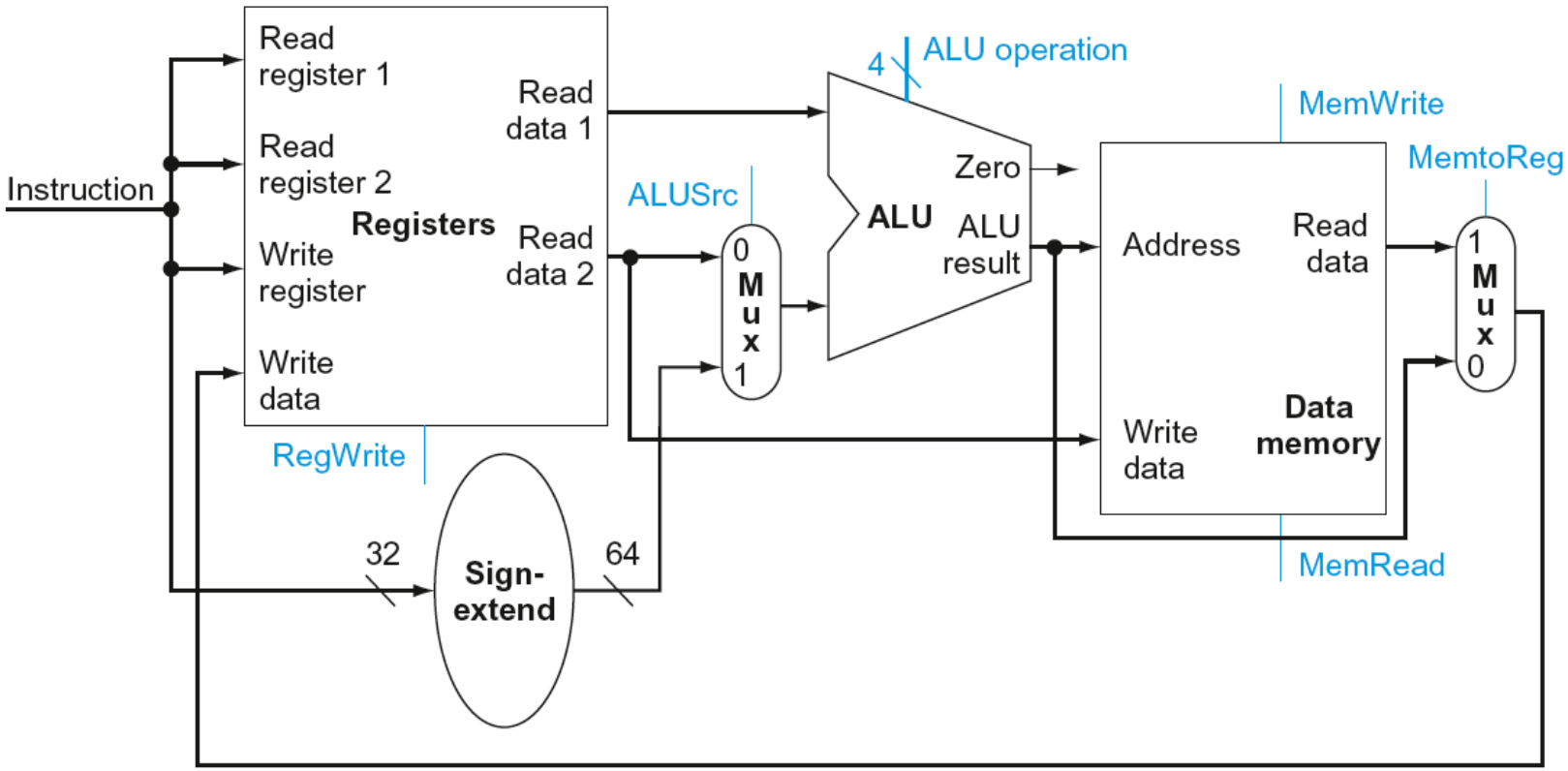
- `STUR X1, [X2, #offset]`



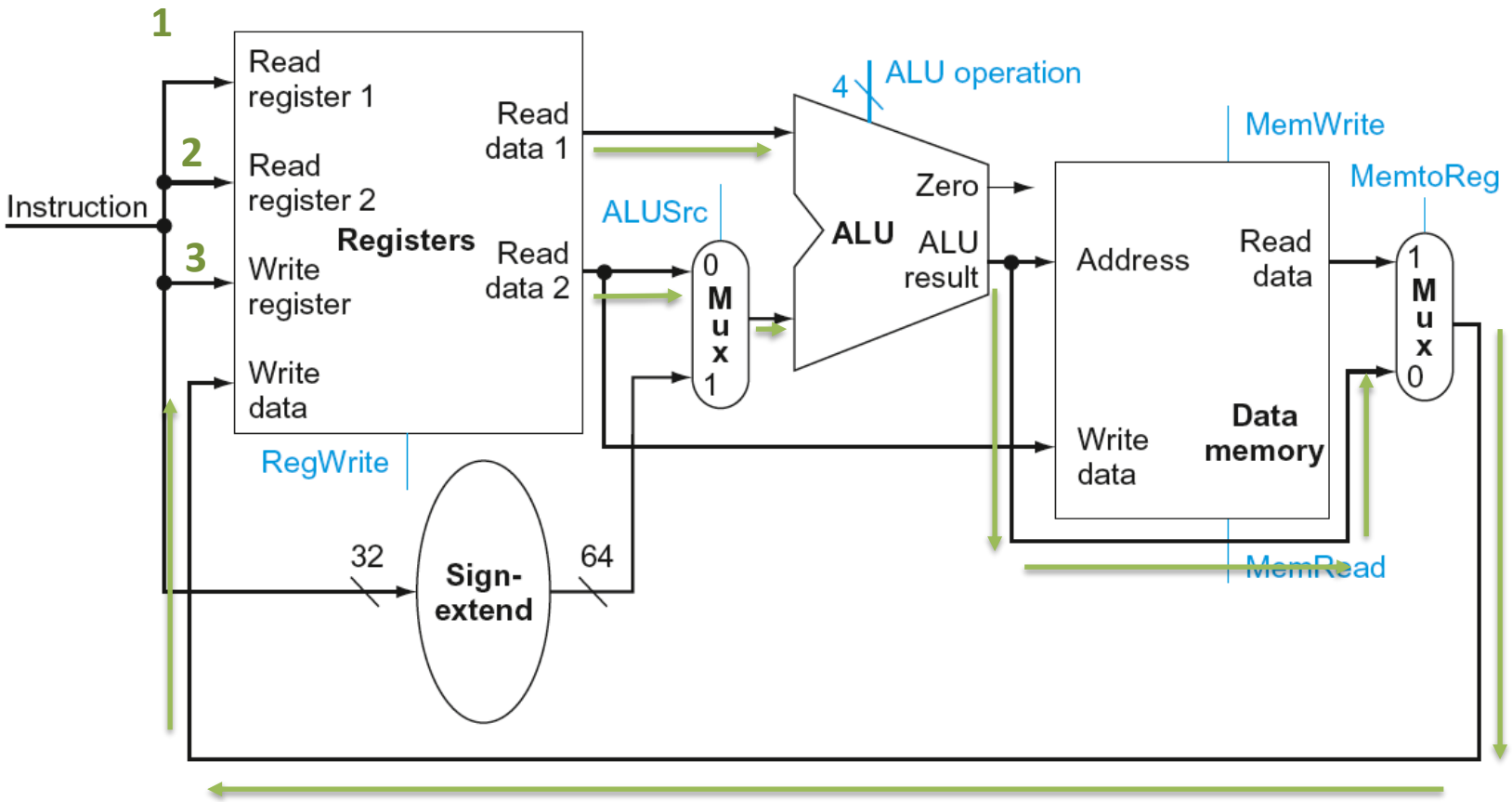
R-Type/Load/Store Datapath



R-Type/Load/Store Datapath



R-Type/Load/Store Datapath



R-type
ADD X3, X1, X2

RegWrite ➔ 0
ALUSrc ➔ 0
ALU operation ➔ 0010
MemWrite ➔ 0
MemRead ➔ 0
MemtoReg ➔ 0

Later
RegWrite ➔ 1

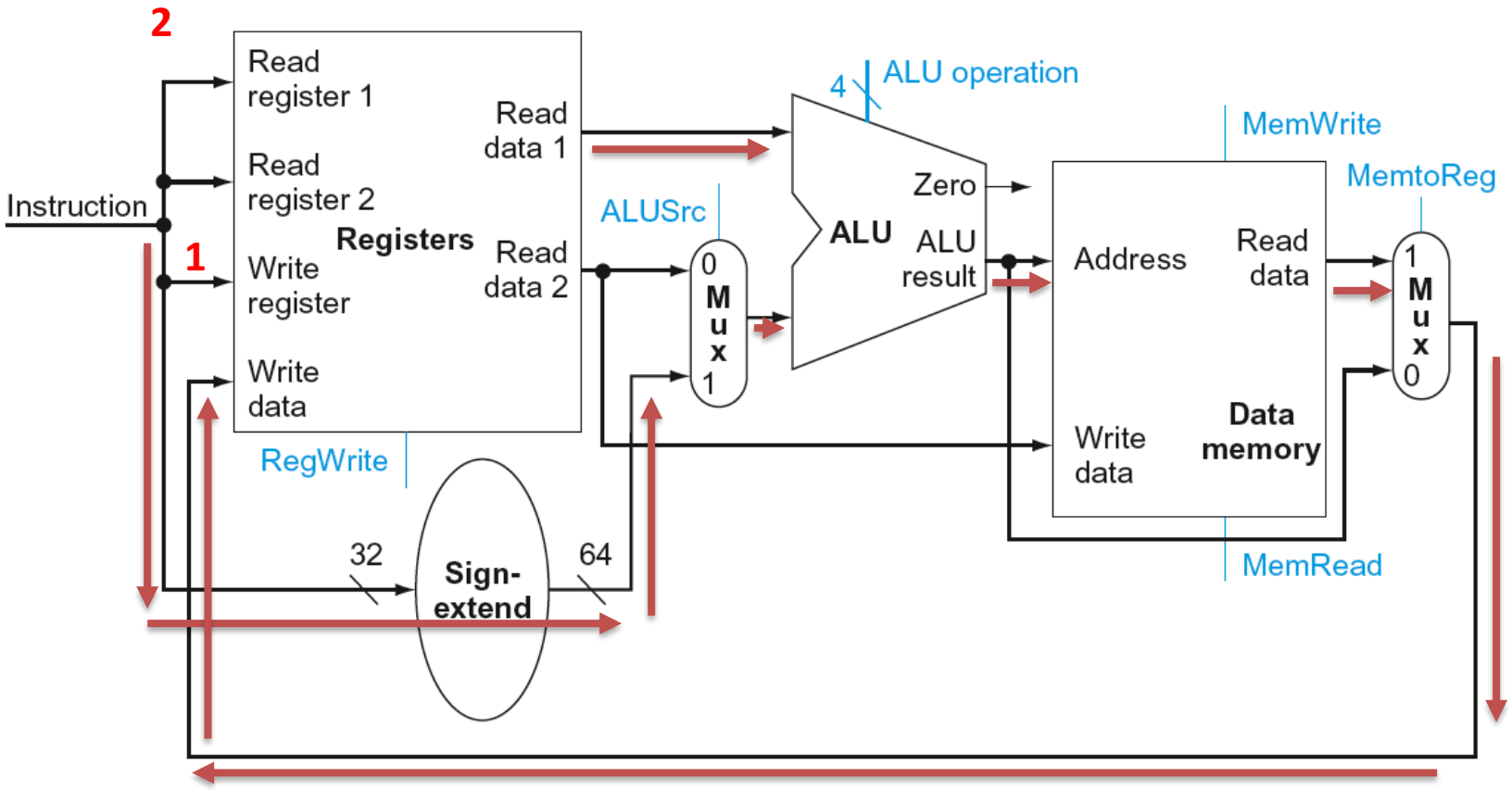
ALU

- Combines adder and And/OR logic gate

b. ALU

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

R-Type/Load/Store Datapath



D-type
`LDUR X1, [X2, #offset]`

RegWrite ➔ 0
ALUSrc ➔ 1
ALU operation ➔ 0010
MemWrite ➔ 0
MemRead ➔ 1
MemtoReg ➔ 1

Later
RegWrite ➔ 1

ALU

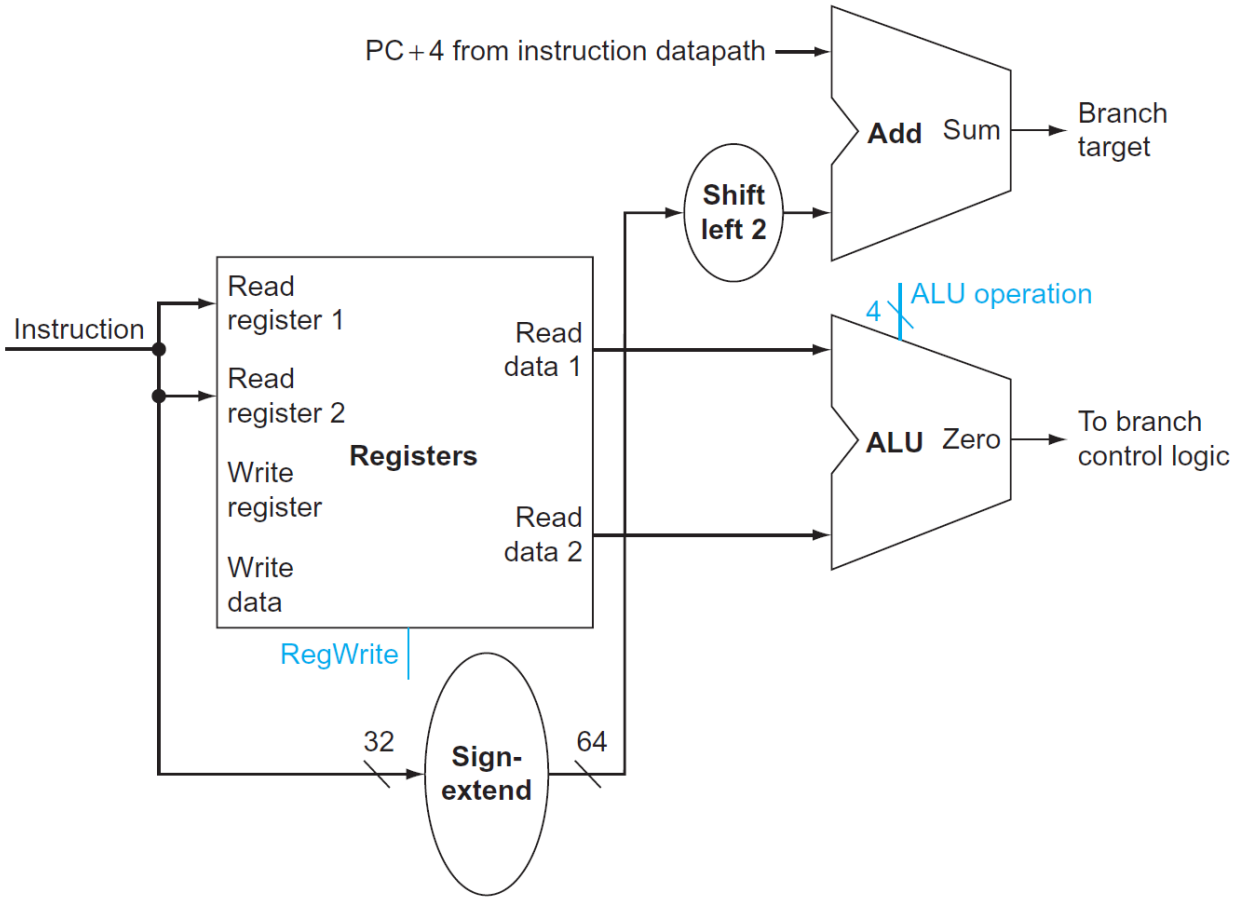
- Combines adder and And/OR logic gate

b. ALU

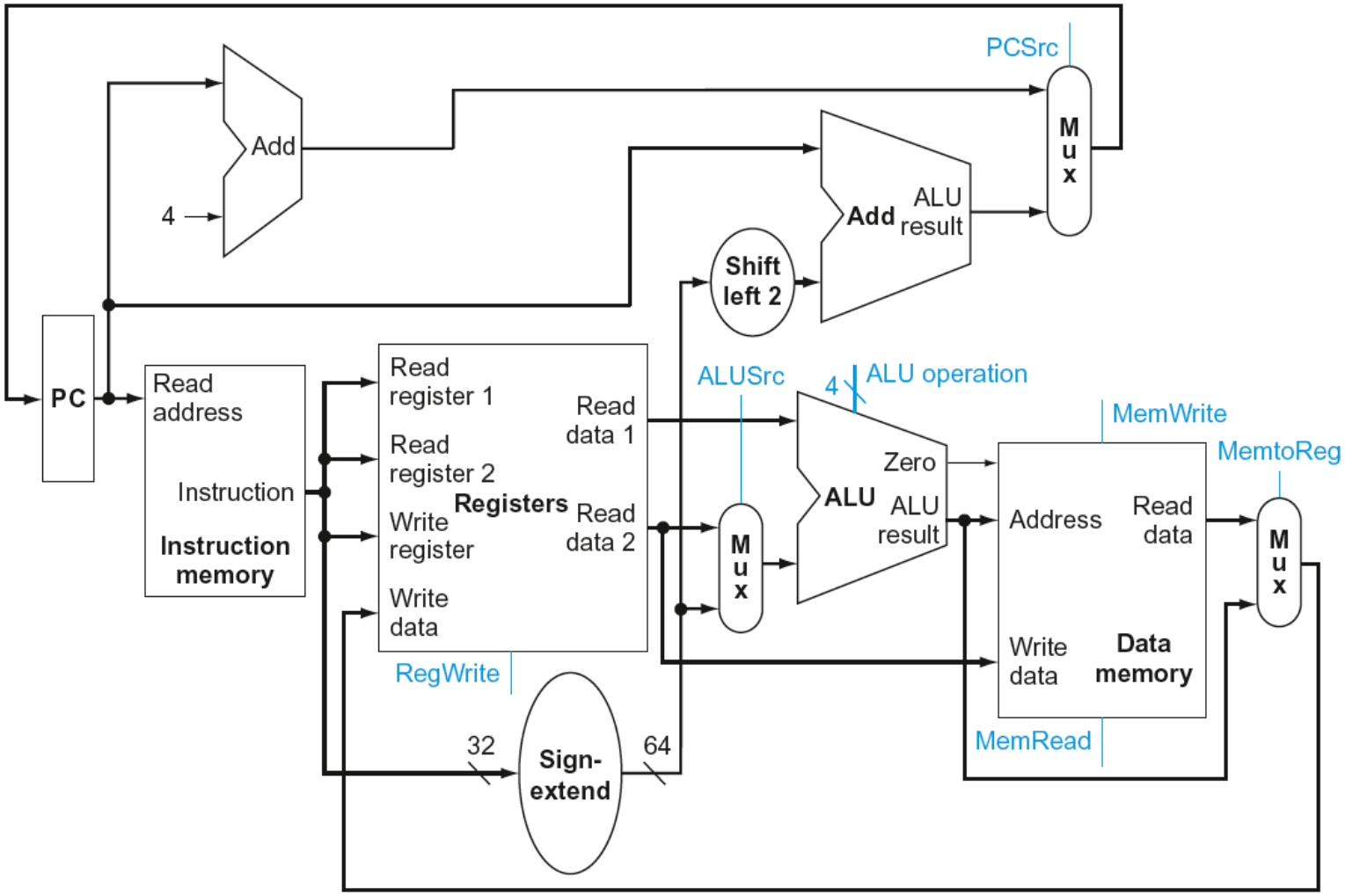
ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	pass input b
1100	NOR

Branch Instructions

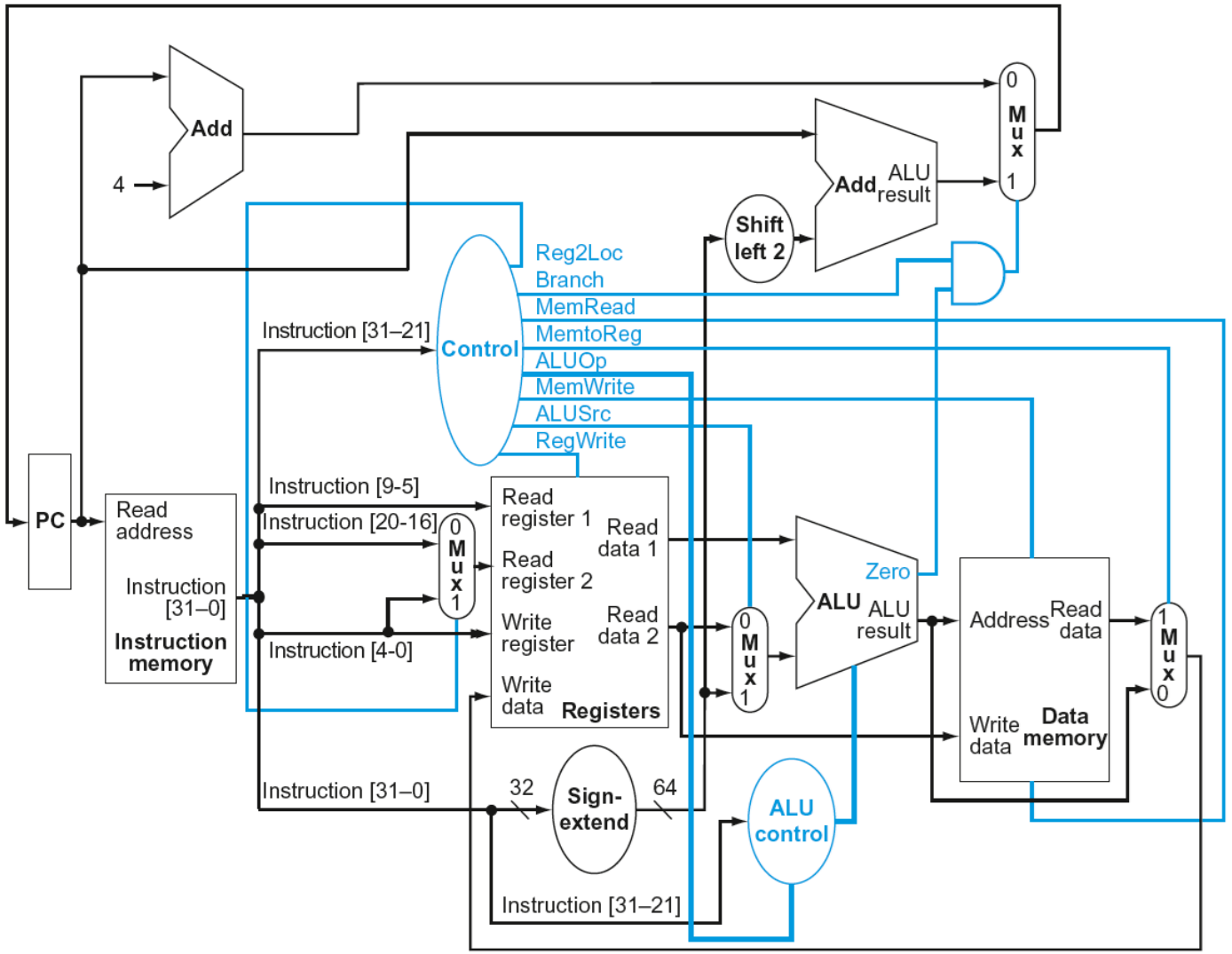
CBZ X0, address(offset)



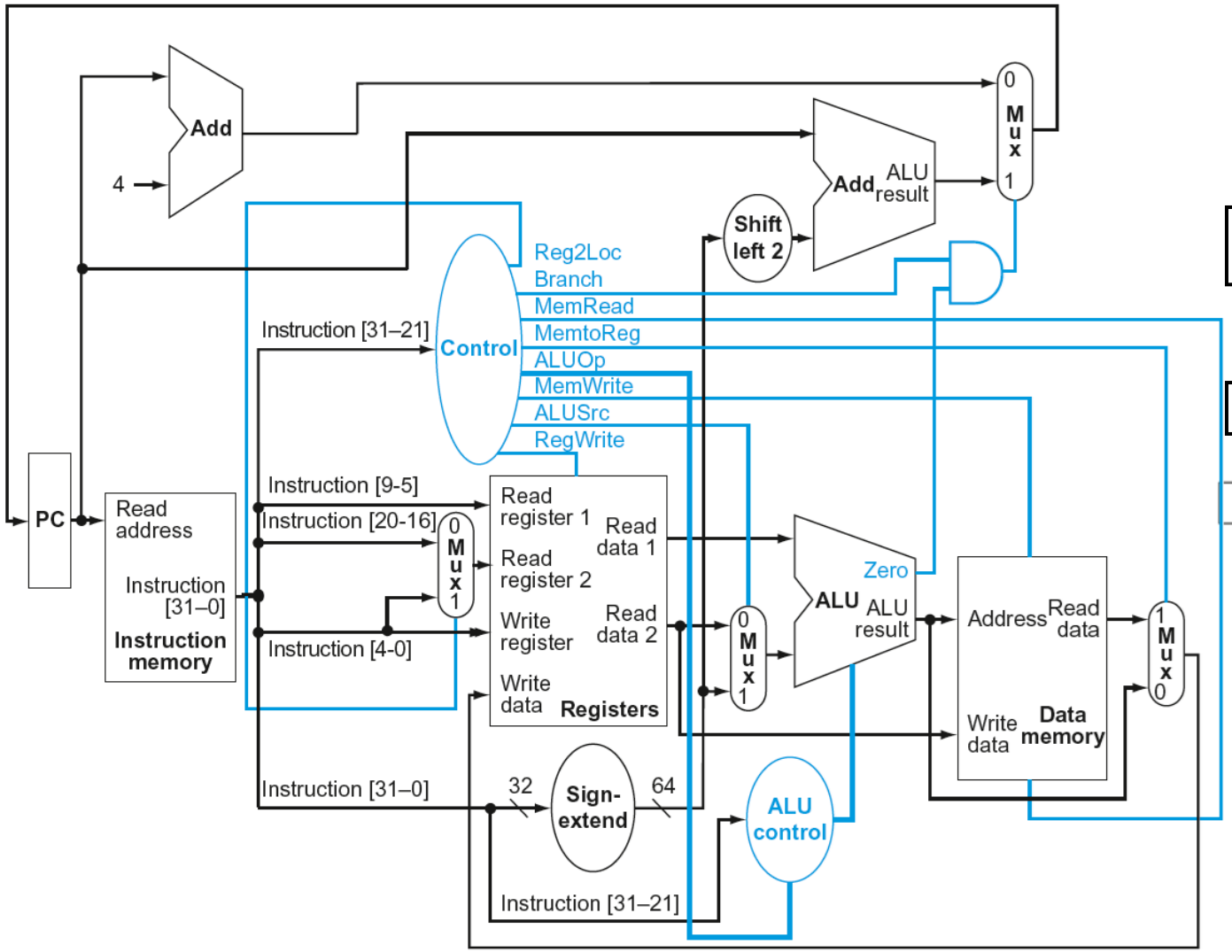
Full Datapath



Datapath With Control



Datapath With Control (R-Type)



ADD X9,X20,X21

1112_{ten}	21_{ten}	0_{ten}	20_{ten}	9_{ten}
opcode	Rm	shamt	Rn	
10001011000_{two}	10101_{two}	000000_{two}	10100_{two}	01001_{two}

opcode	Rm	shamt	Rn	Rd
31:21	20:16	15:10	9:5	4:0