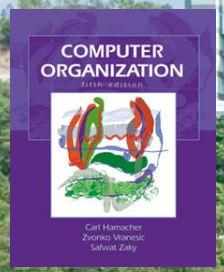
香港中文大學
The Chinese University of Hong Kong

CSCI2510 Computer Organization

Lecture 09: Basic Processing Unit

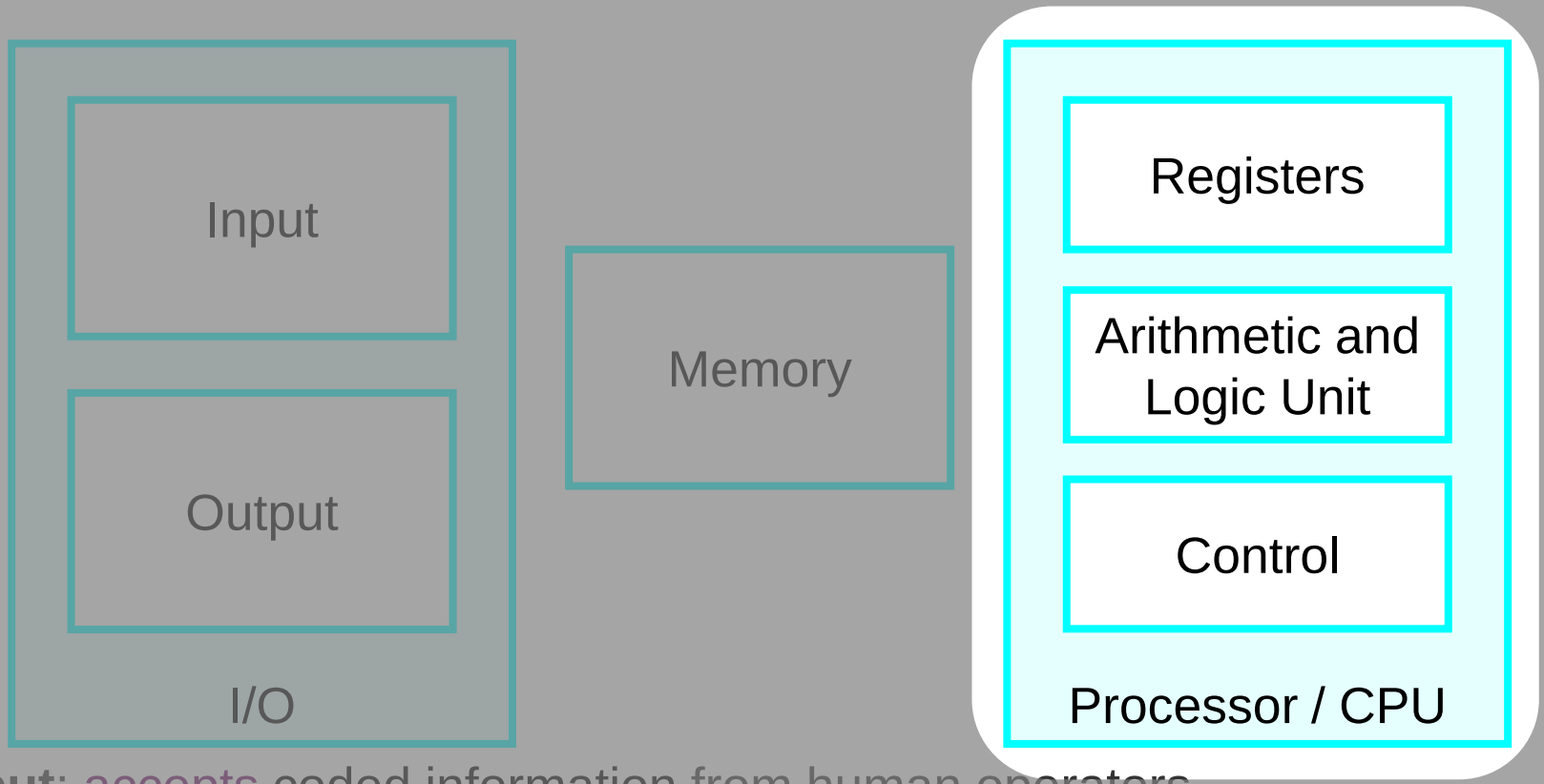
Ming-Chang YANG

mcyang@cse.cuhk.edu.hk



Reading: Chap. 7.1~7.3 (5th Ed.)

Basic Functional Units of a Computer



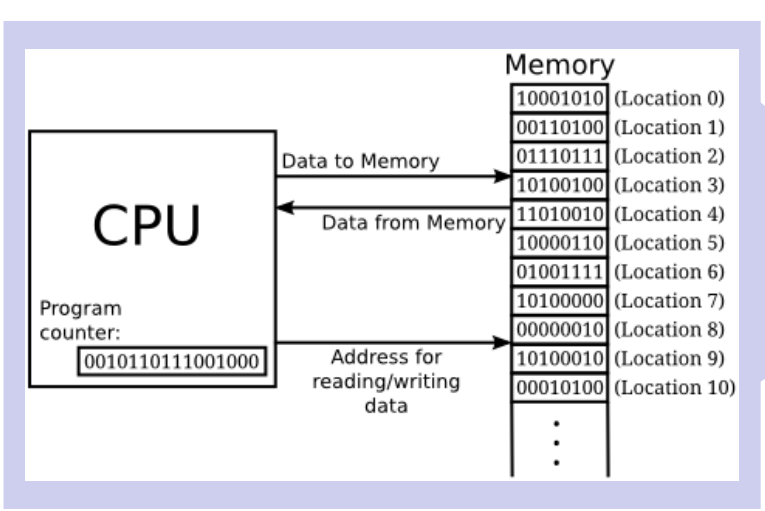
- **Input**: accepts coded information from human operators.
- **Memory**: stores the received information for later use.
- **Processor**: executes the instructions of a program stored in the memory.
- **Output**: reacts to the outside world.
- **Control**: coordinates all these actions.

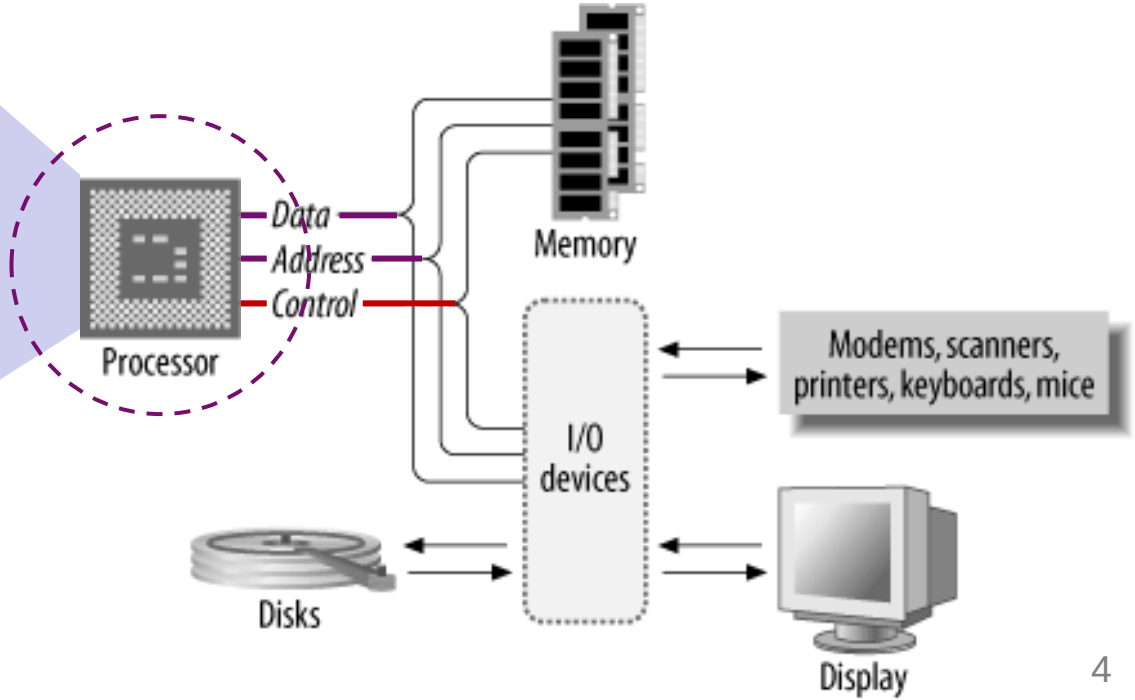
Outline

- Processor Internal Structure
- Instruction Execution
  - Fetch Phase
  - Execute Phase
- Execution of A Complete Instruction
- Multiple-Bus Organization

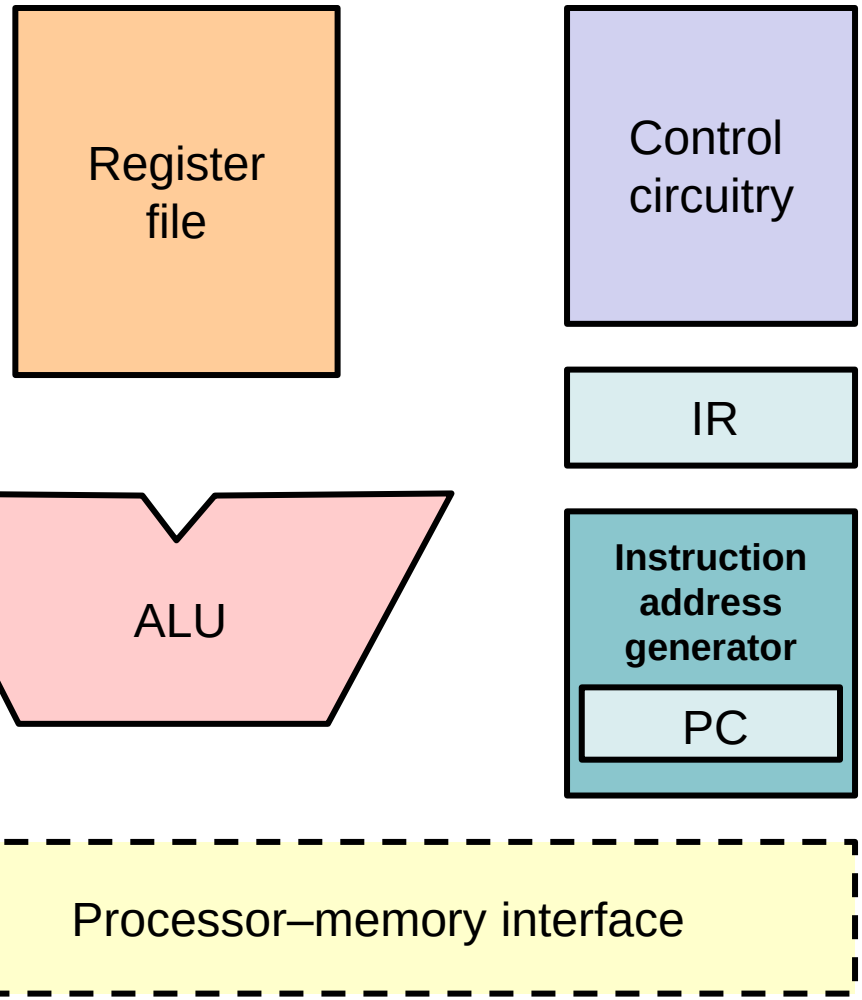
Basic Processing Unit: Processor

- Executes machine-language instructions.
- Coordinates other units in a computer system.
- Often be called the central processing unit (CPU).
  - The term "central" is no longer appropriate today.
  - Today's computers often include several processing units.
    - E.g., multi-core processor, graphic processing unit (GPU), etc.





Main Components of a Processor



Register file: a memory unit for the processor's **general-purpose registers (GPRs)**

Control circuitry: Interpret or decode the fetched instruction

IR: Hold the instruction until its execution is completed (*special purpose register*)

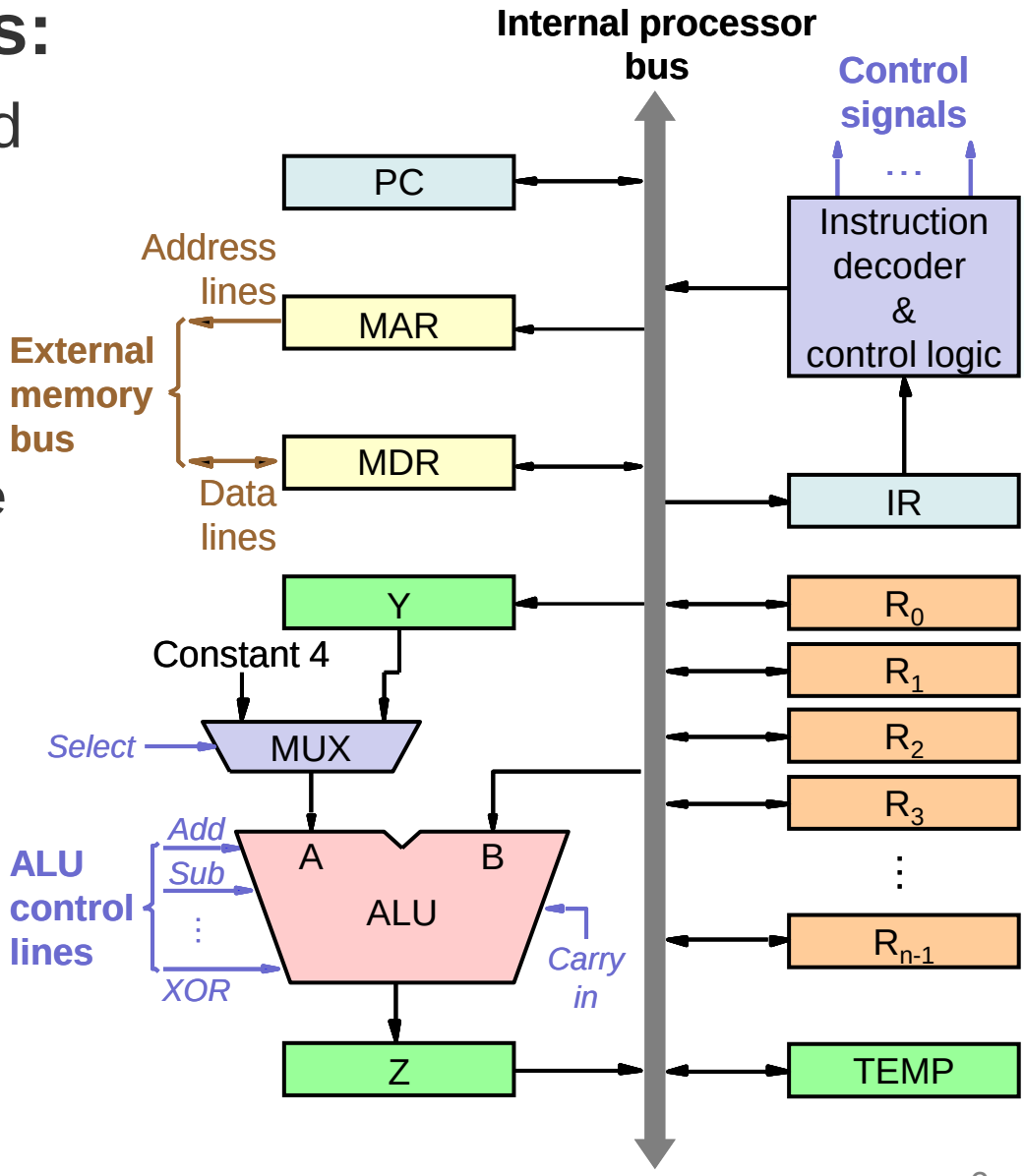
Arithmetic and Logic Unit (ALU): Perform an arithmetic or logic operation

PC: Keep track of the address of the next instruction to be fetched and executed (*special purpose register*)

Processor-memory interface: Allow the communication between processor and memory

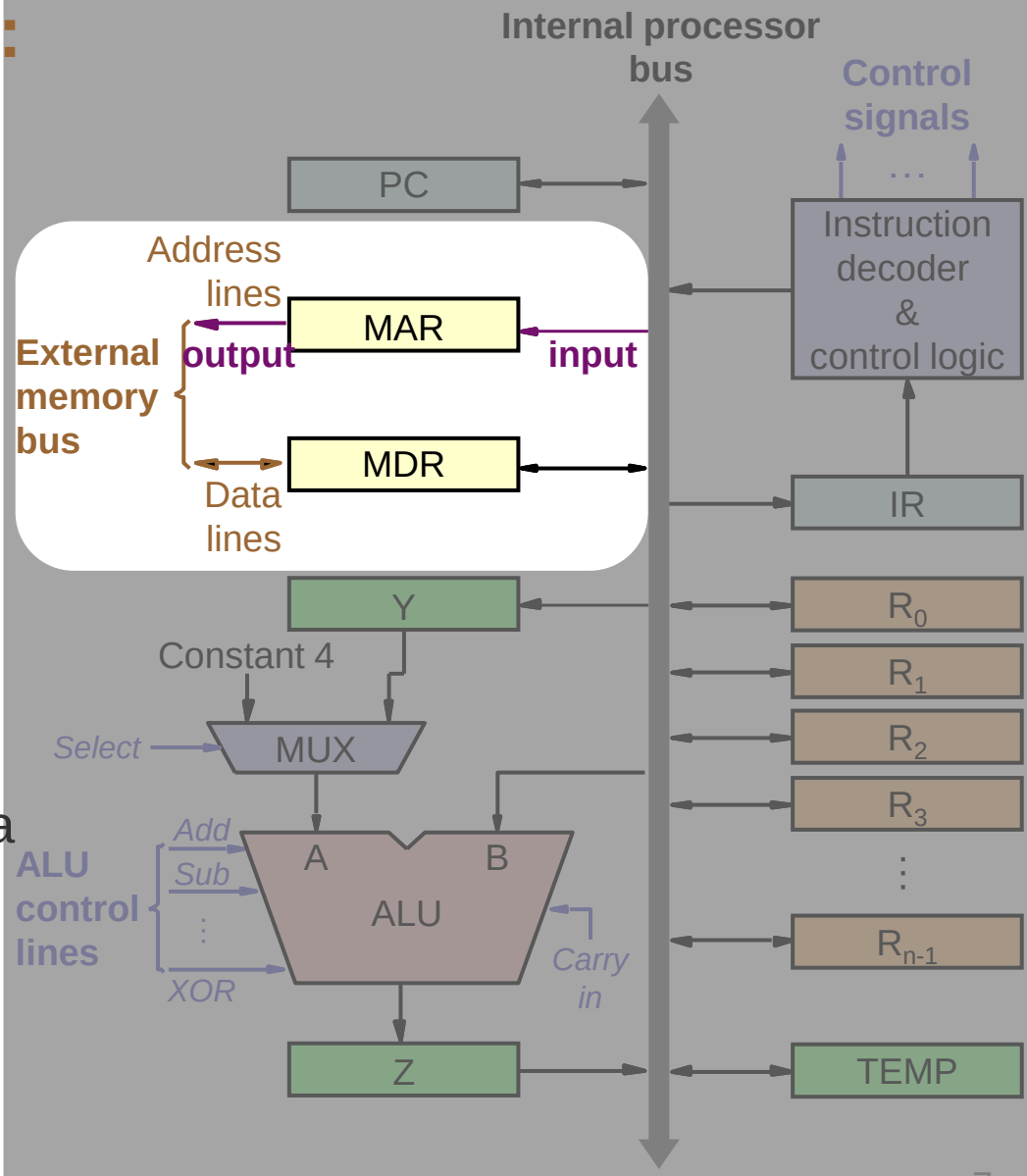
Processor Internal: Internal Bus

- **Internal Processor Bus:**
  - ALU, control circuitry, and all the registers are interconnected via a **single common bus**.
    - The bus is internal to the processor (i.e., only visible to the processor).
  - **Black** parts: data (and control) path
  - **Blue** parts: control path
- **External Memory Bus:**
  - **Brown** part: external memory path



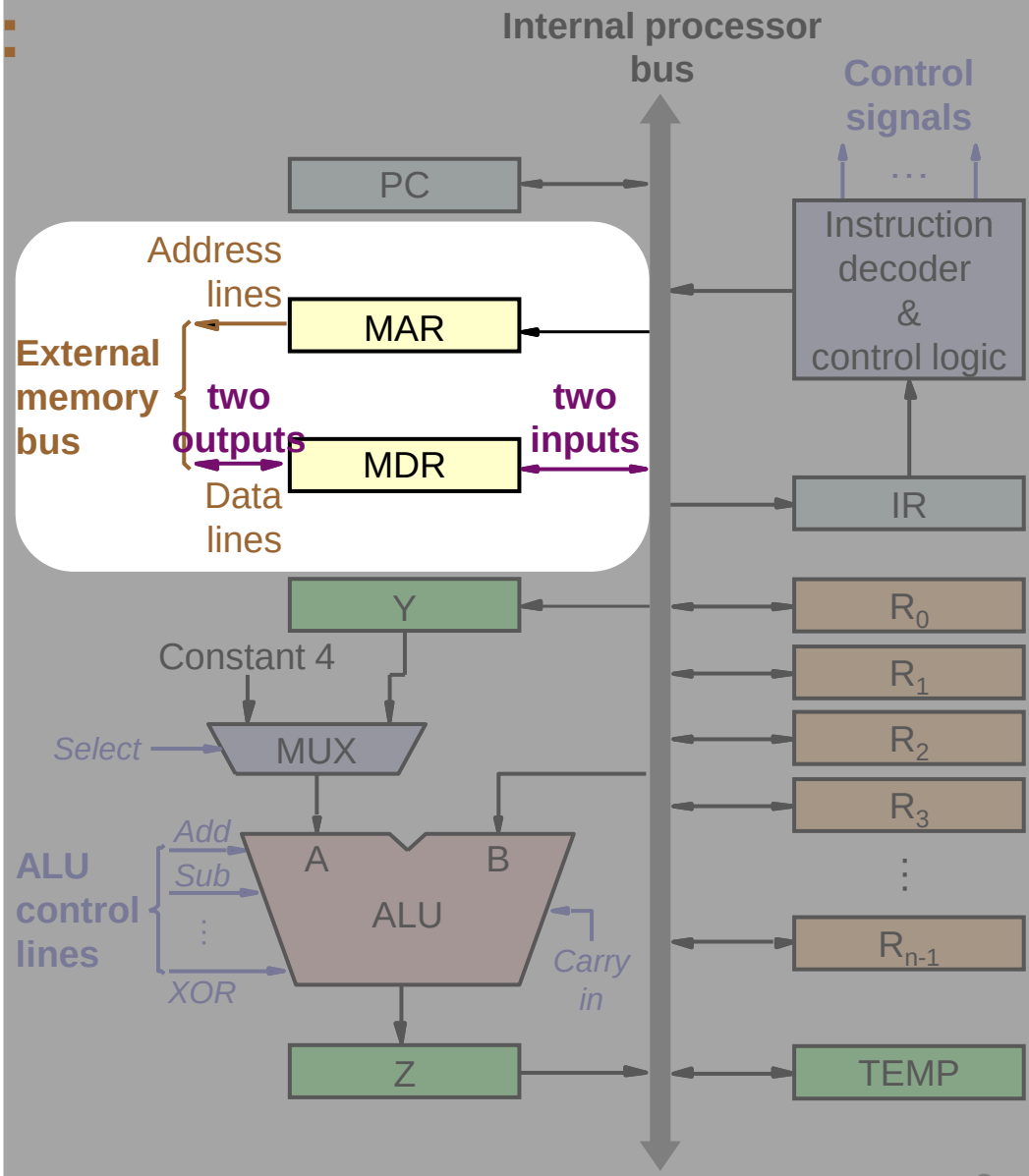
Processor Internal: External Bus (1/2)

- **External Memory Bus:**
  - **Processor-memory interface**: External memory bus are controlled through MAR and MDR.
  - **MAR**: Specify the requested memory address
    - **Input**: Address is specified by processor via internal processor bus.
    - **Output**: Address is send to the memory via external memory bus.



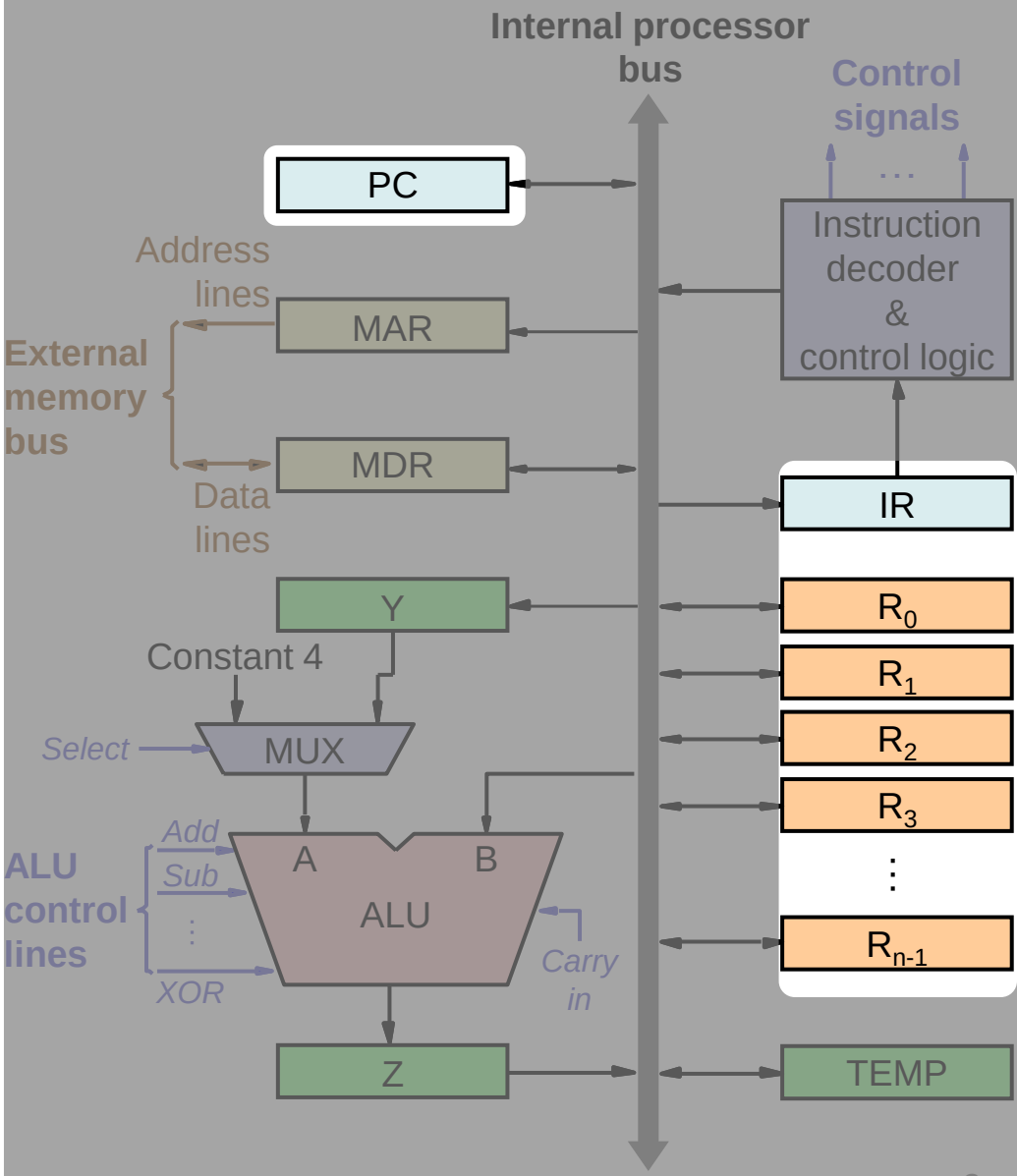
Processor Internal: External Bus (2/2)

- **External Memory Bus:**
  - **MDR**: Keep the content of the requested memory address
    - There are two inputs and two outputs for MDR.
    - **Inputs**: Data may be placed into MDR either
      - From the internal processor bus or
      - From the external memory bus.
    - **Outputs**: Data stored in MDR may be loaded from either bus.



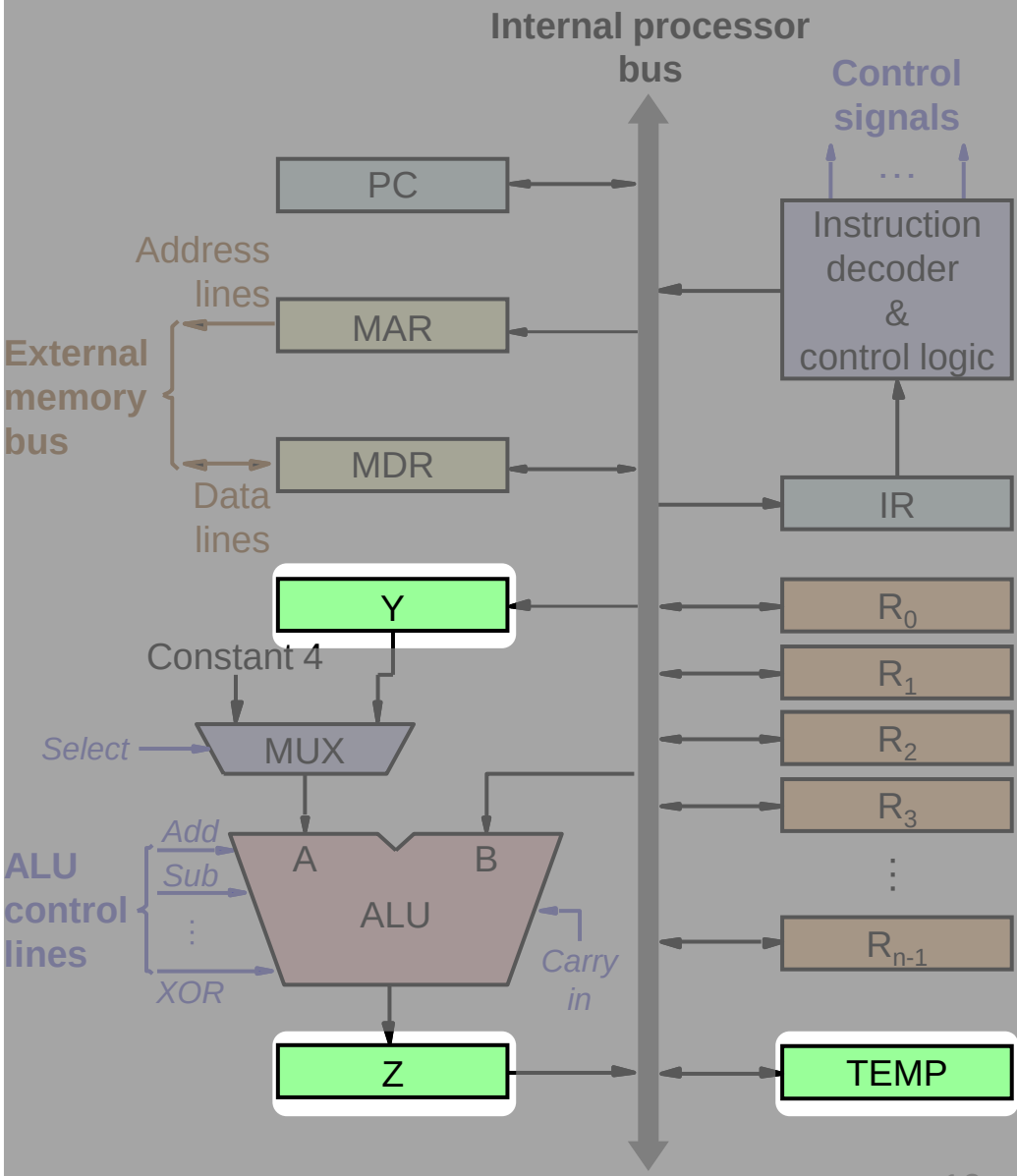
Processor Internal: Register (1/2)

- **General-Purpose Registers:**
  - R_0 through R_{n-1}
    - *n* varies from one processor to another.
- **Special Registers:**
  - **Program Counter**
    - Keep track of the address of the next instruction to be fetched and executed.
  - **Instruction Register**
    - Hold the instruction until the current execution is completed.



Processor Internal: Register (2/2)

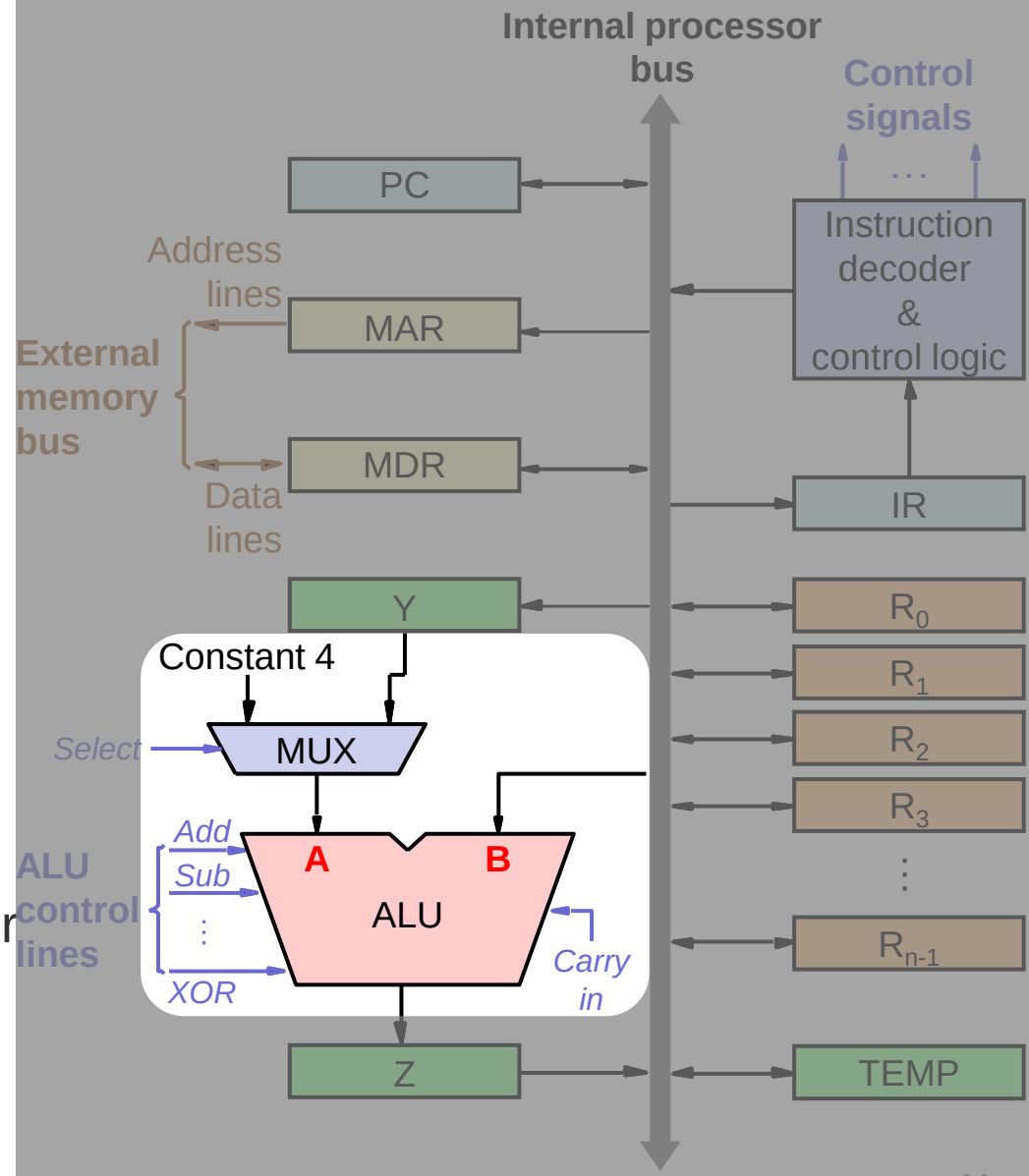
- **Special Registers: Y, Z, & TEMP**
  - Transparent to the programmer.
  - **Used** by the processor for temporary storage during execution of some instructions.
  - **Never used** for storing data generated by one instruction for later use by another instruction.
  - We will discuss their functionalities later.



Processor Internal: Internal Bus

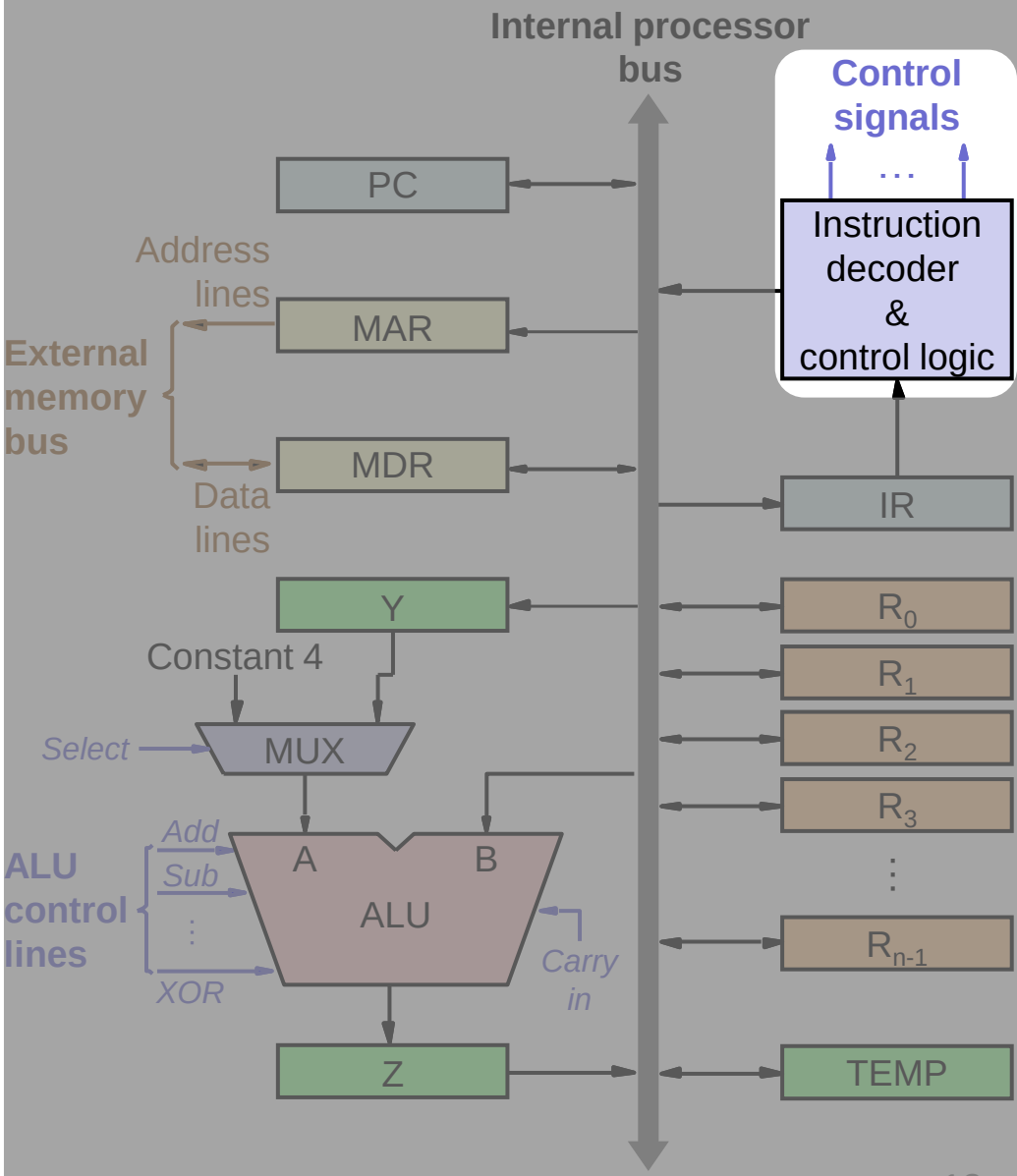
- **Arithmetic and Logic Unit (ALU):**
  - Perform arithmetic or logic operation

    Z = A operator B
    - Two inputs **A** and **B**
    - One output to register **Z**
- **Multiplexer (MUX):**
  - The **input A** of ALU: Select (*ctrl line*) either
    - The output of register Y or
    - A constant value 4 (for incrementing PC).



Processor Internal: Control Circuitry

- **Instruction decoder**:
  - Interpret the fetched instruction stored in the IR register.
- **Control logic**:
  - Issue control signals to control the all the units inside the processor.
    - E.g., ALU control lines, select-signal for MUX, carry-in for ALU, etc.
  - Interact with the external memory bus.



Outline

- Processor Internal Structure
- Instruction Execution
  - Fetch Phase
  - Execute Phase
- Execution of A Complete Instruction
- Multiple-Bus Organization

Recall: Register Transfer Notation

- Register Transfer Notation (RTN) describes the *data transfer from one location in computer to another.*
  - Possible locations: memory locations, processor registers.
    - Locations can be identified symbolically with names (e.g. LOC).

Ex.

R2 ← [LOC]

- *Transferring the contents of memory LOC into register R2.*

① **Contents of any location**: denoted by placing square brackets **[ ]** around its location name (e.g. **[**LOC**]**).

② **Right-hand side** of RTN: always denotes a value

③ **Left-hand side** of RTN: the name of a location where the value is to be placed (by overwriting the old contents)

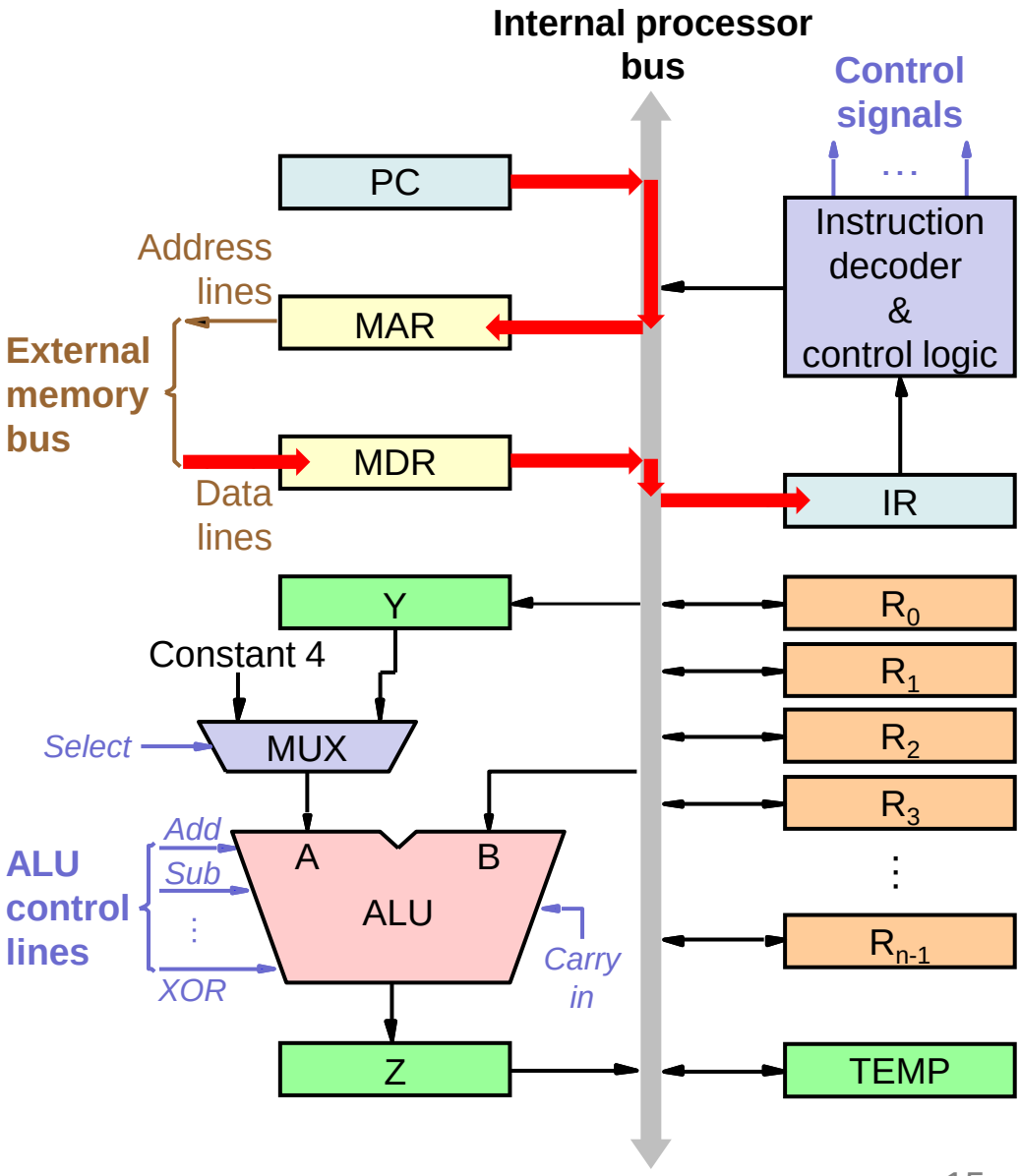
Instruction Execution (1/3)

1) Fetch Phase

- IR ← [[PC]]
  - Fetch the contents of the memory location pointed to by PC, and load into IR
- PC ← [PC]+4
  - Increment the contents of PC by 4.
    - Why 4? Instruction is 32 bits (4B) and memory is byte addressable.

2) Execute Phase

- Decode instruction in IR
- Perform the operation(s)



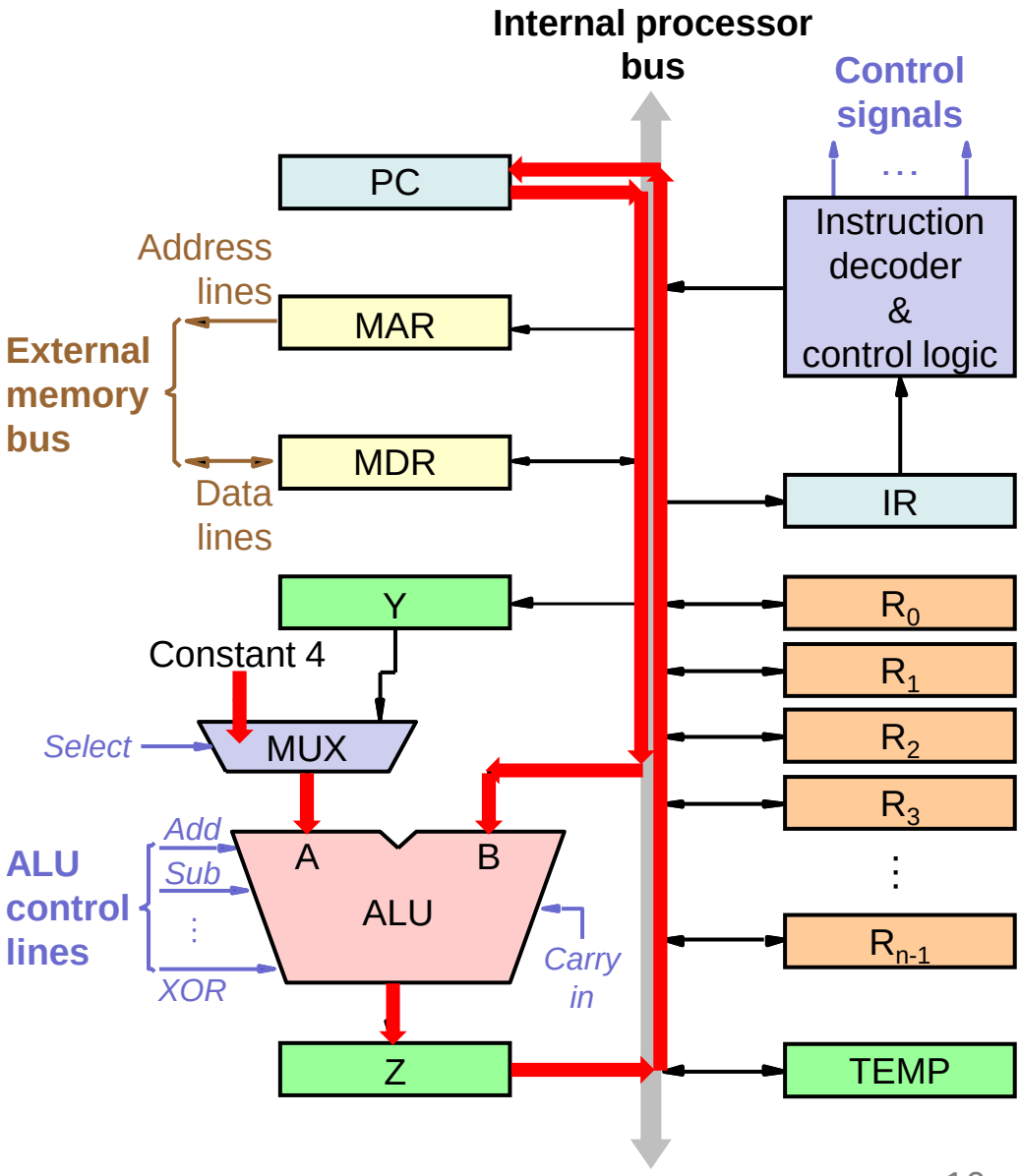
Instruction Execution (2/3)

1) Fetch Phase

- IR ← [[PC]]
  - Fetch the contents of the memory location pointed to by PC, and load into IR
- **PC ← [PC]+4**
  - Increment the contents of PC by 4.
    - Why 4? Instruction is 32 bits (4B) and memory is byte addressable.

2) Execute Phase

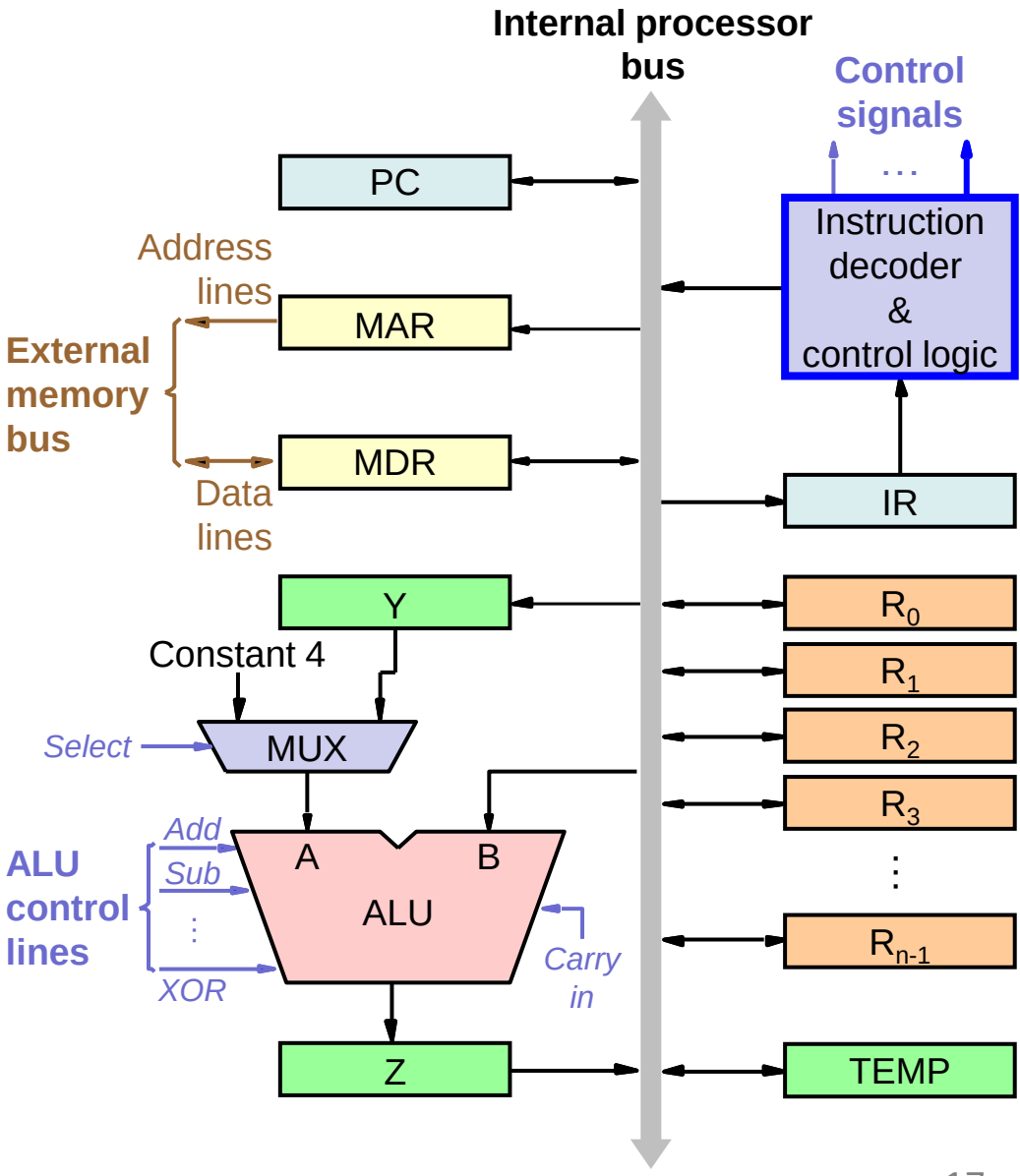
- Decode instruction in IR
- Perform the operation(s)



Instruction Execution (3/3)

1) Fetch Phase
   - IR ← [[PC]]
     - Fetch the contents of the memory location pointed to by PC, and load into IR
   - PC ← [PC]+4
     - Increment the contents of PC by 4.
       - Why 4? Instruction is 32 bits (4B) and memory is byte addressable.

2) Execute Phase
- Decode instruction in IR
- Perform the operation(s)



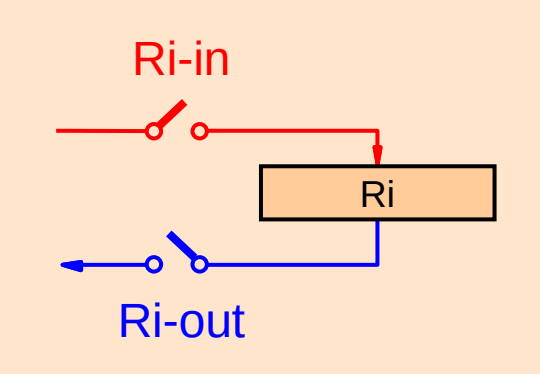
Instruction Execution: Execute Phase

- An instruction can be executed by performing one or more of the following operation(s):
  1) **Transfer data** from a register to another register or to the ALU
  2) **Perform arithmetic** (or logic) **operations** and store the result into the special register Z
  3) **Load content** of a memory location to a register
  4) **Store content** of a register to a memory location

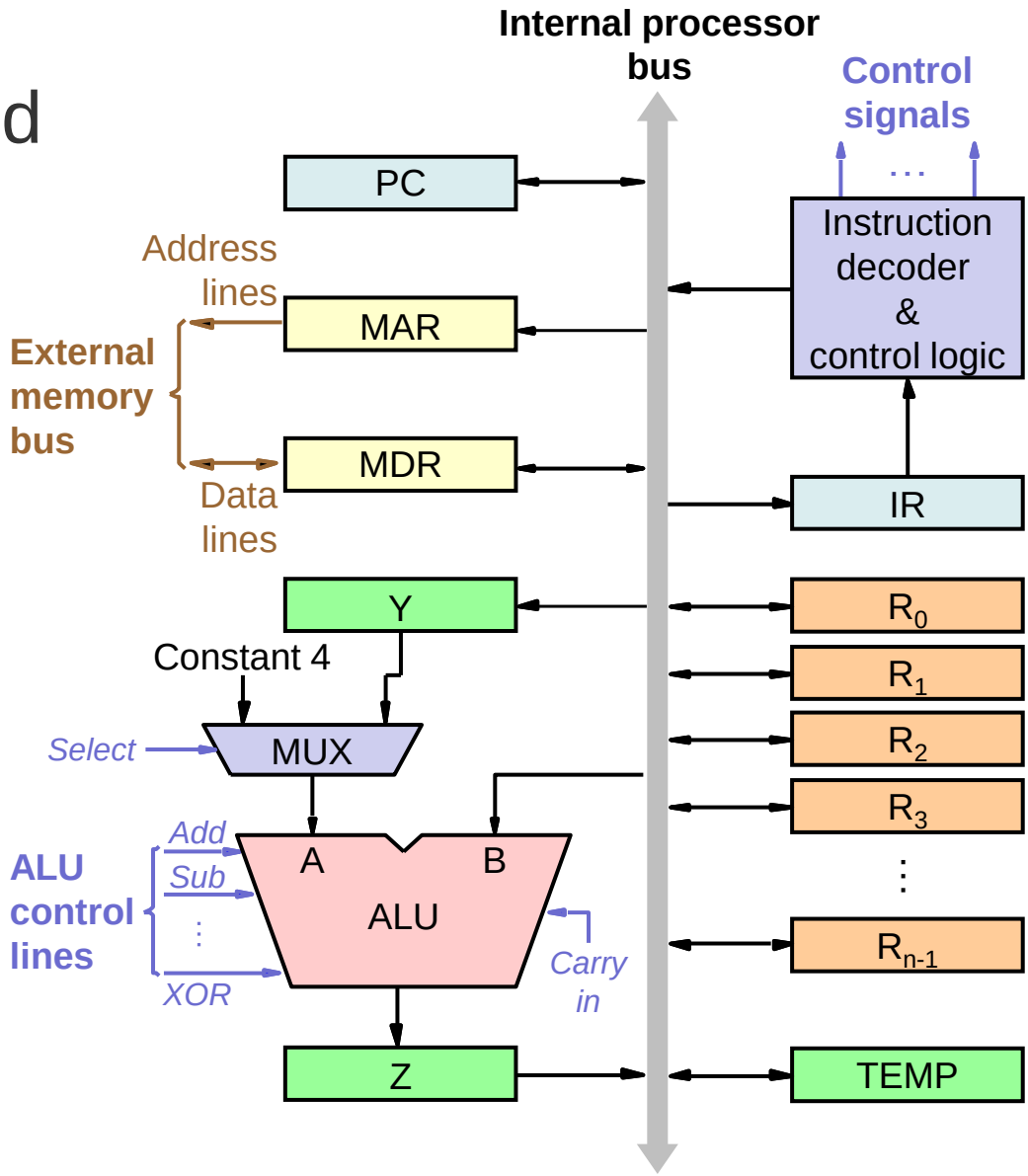
- **Sequence of Control Steps**: Describes how these operations are performed in processor step by step.

1) Register Transfer

- Input and output of register Ri are controlled by switches (—⁄o—):
  - Ri-in: Allow data to be transferred into Ri



  - Ri-out: Allow data to be transferred out from Ri



1) Register Transfer (Cont'd)

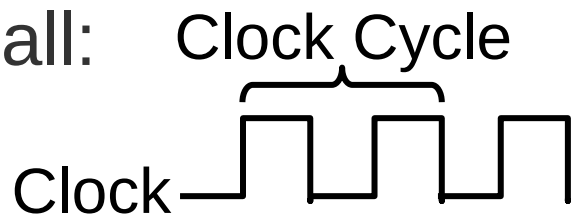
- Ex: **R3 ← [R1]**
  - ① Clock 1: R1-out, R3-in
    - Set R1-out to 1
    - Set R3-in to 1
    - Set all others to 0
  - ② Clock 2:
    - Reset R1-out to 0
    - Reset R3-in to 0

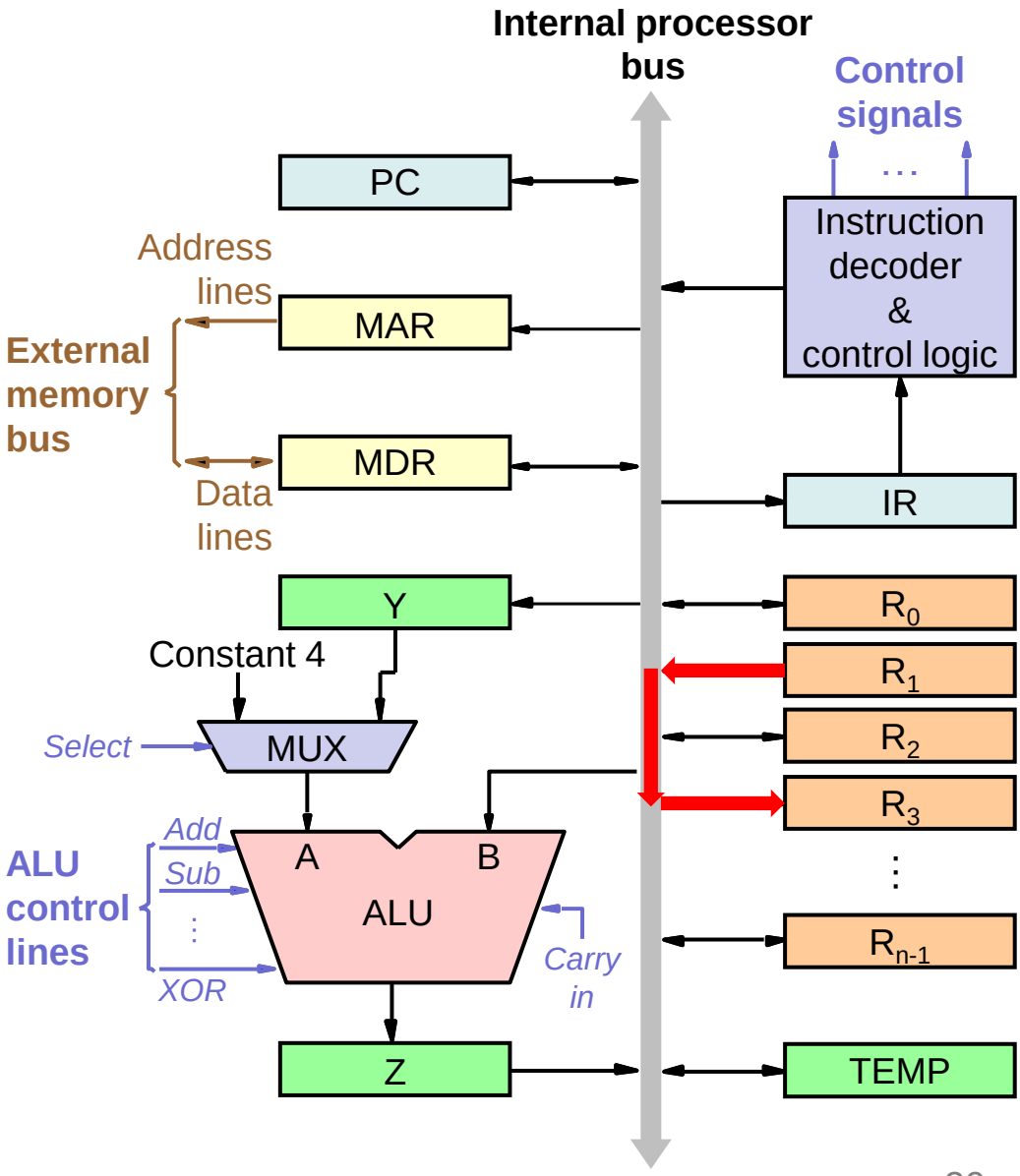
Sequence of Steps:

① R1-out, R3-in

*Note: Only state "**set**" for short.*

- Recall:



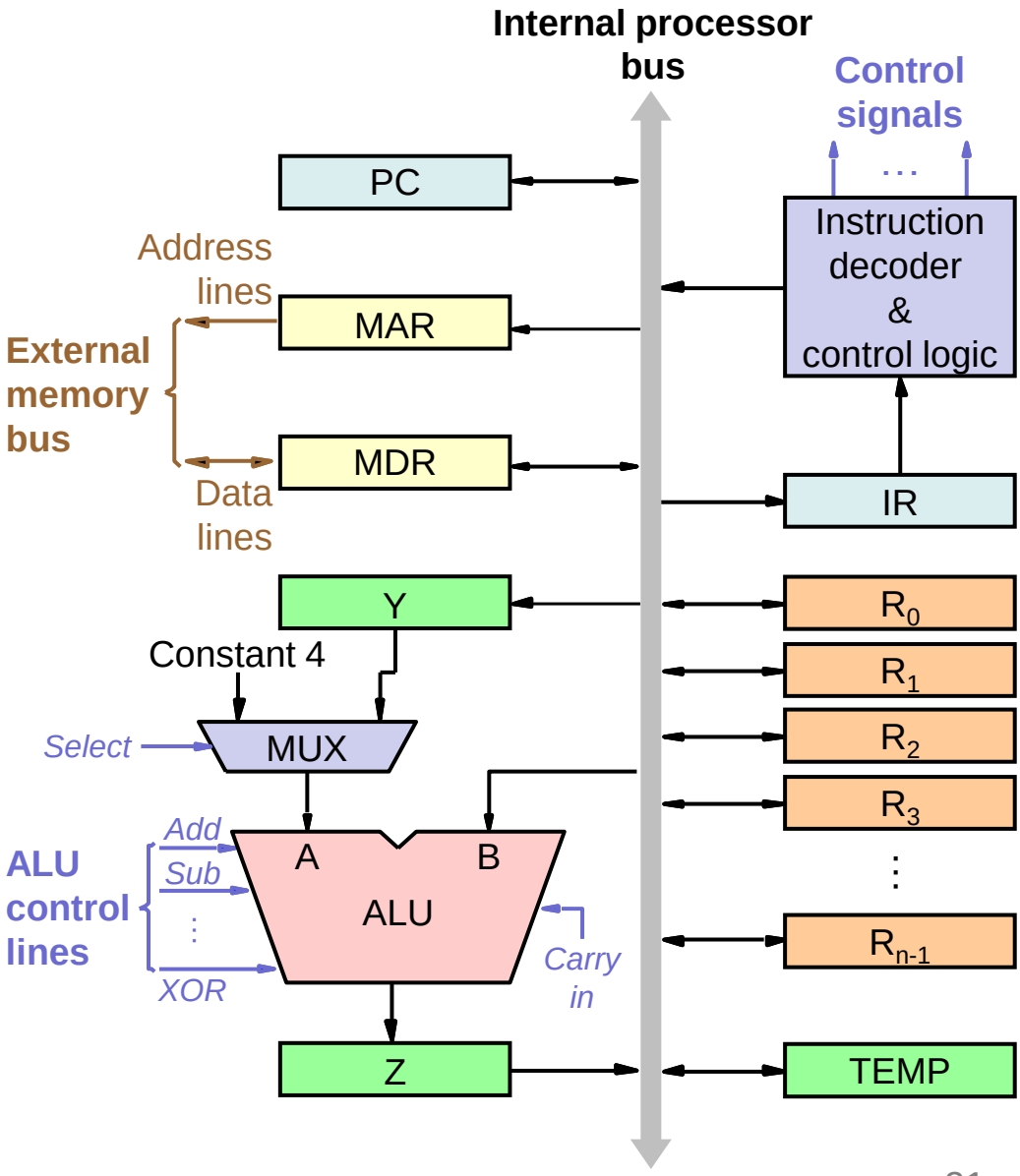


Class Exercise 10.1

Student ID: _____ Date:
Name: _____ _____

- What is the sequence of steps for the following operation?

R1 ← [R3]



2) Arithmetic or Logic Operation

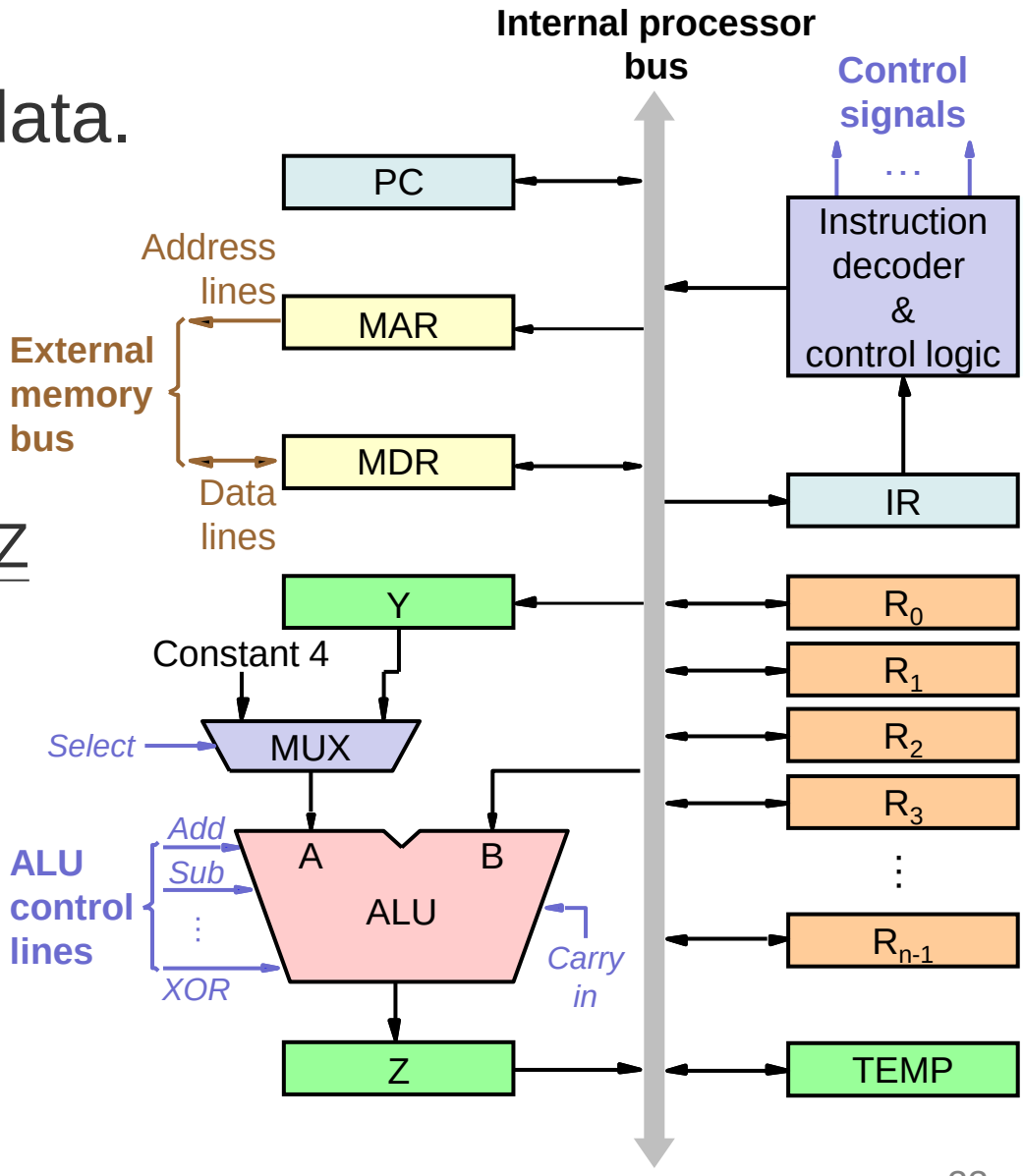
- **ALU**: A circuit without storage to manipulate data.
  - **Two inputs**: from A & B
    - **A**: #4 or register Y
    - **B**: Any other register
  - **ALU**: Perform operation
  - **One output**: to register Z
- Ex: **R3 ← [R1] + [R2]**

Sequence of Steps:

① R1-out, Y-in

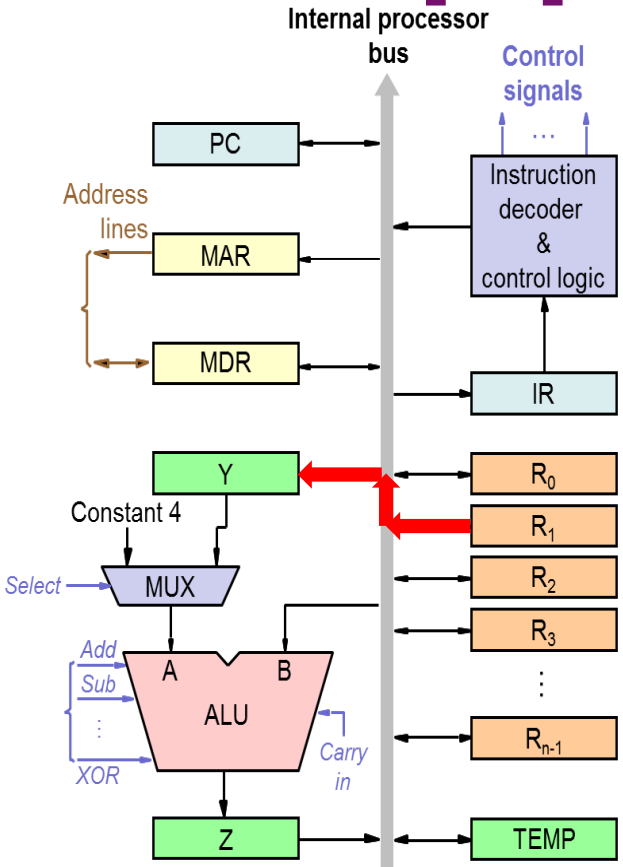
② Select-Y, R2-out, B-in, Add, Z-in

③ Z-out, R3-in

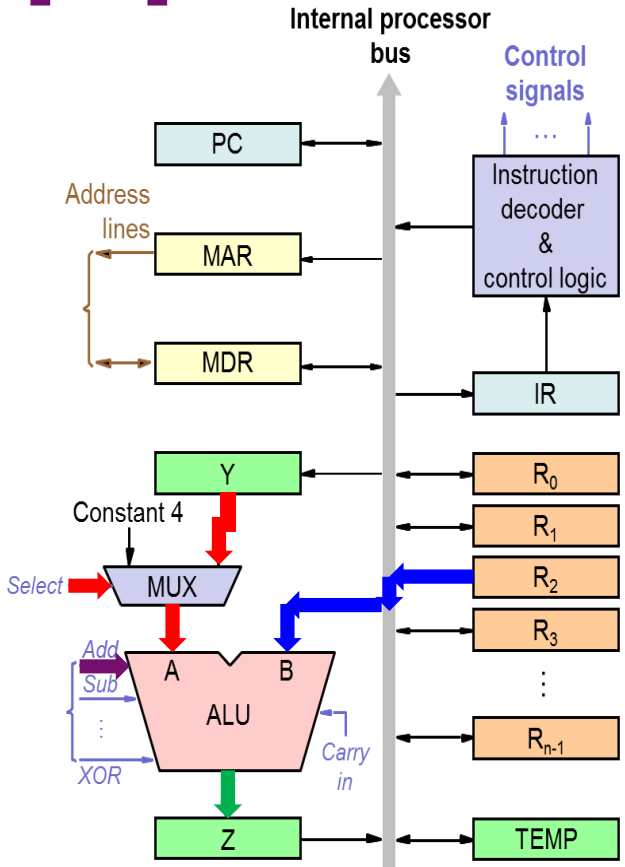


2) Arithmetic or Logic Operation (Cont'd)

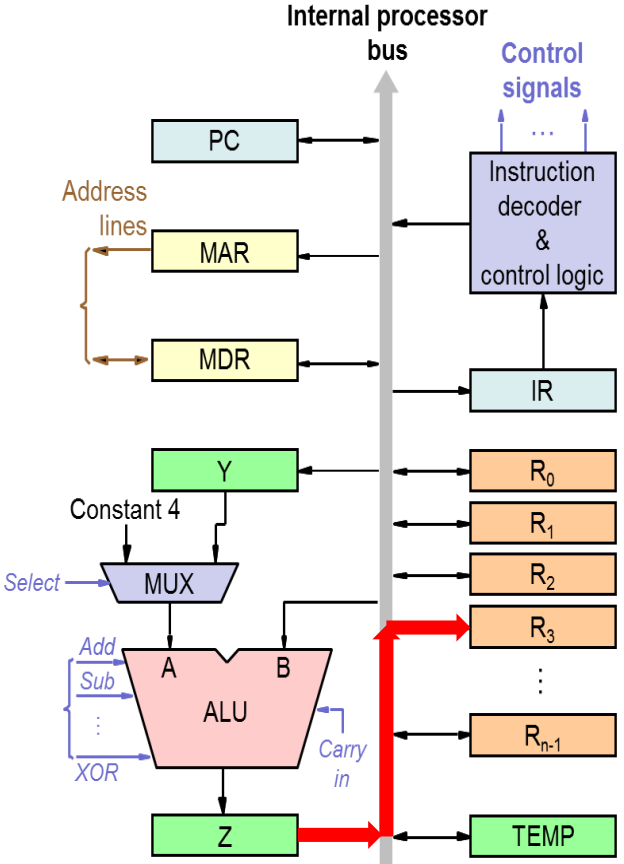
- Ex: **R3 ← [R1] + [R2]**



① R1-out, Y-in



② Select-Y
R2-out, B-in,
Add, Z-in



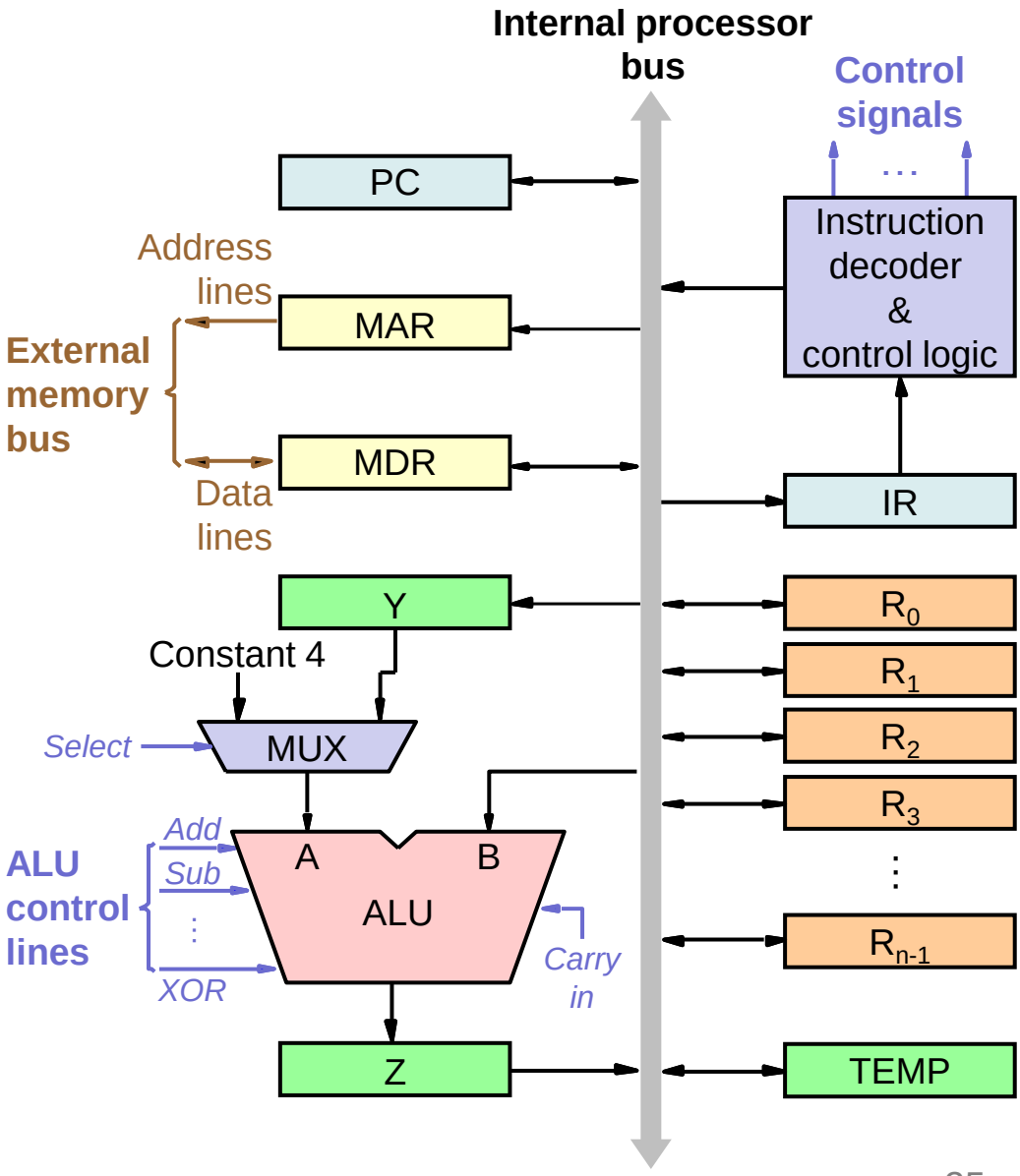
③ Z-out, R3-in

Question: Why to first transfer R1 to the special register Y?

Class Exercise 10.2

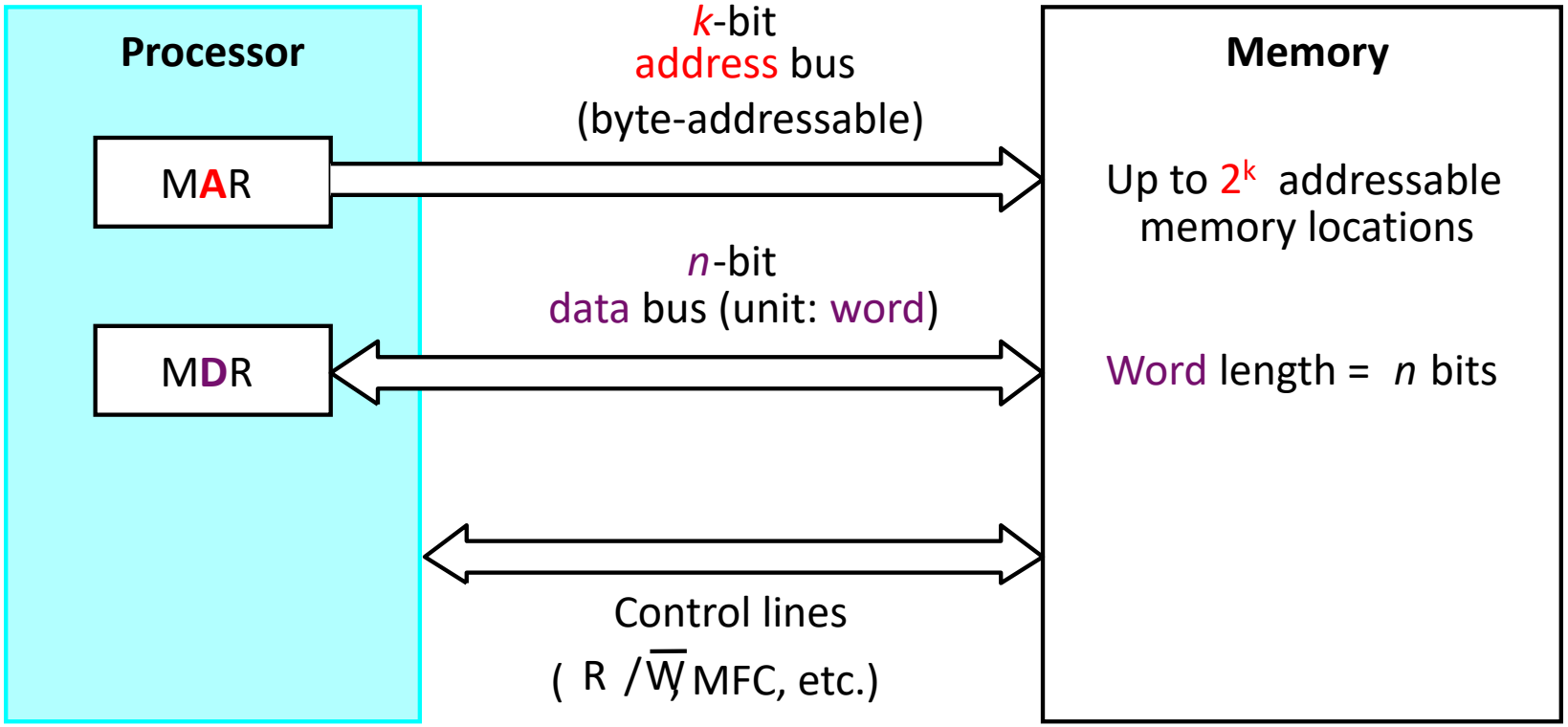
- What is the sequence of steps for the following operation?

  R6 ← [R4] – [R5]



Recall: Processor-Memory Interface

- Data transferring takes place through MAR and MDR.
  - **MAR**: Memory Address Register
  - **MDR**: Memory Data Register



**MFC (Memory Function Completed): Indicating the requested operation has been completed.*

Recall: Assembly-Language Notation

- Assembly-Language Notation is used to represent machine instructions and programs.
  - An instruction must specify an operation to be performed and the operands involved.
  - **Ex.** The instruction that causes the transfer from memory location LOC to register R2:

```
Load R2, LOC
```

Load: *operation;*

LOC: *source operand;*

R2: *destination operand.*

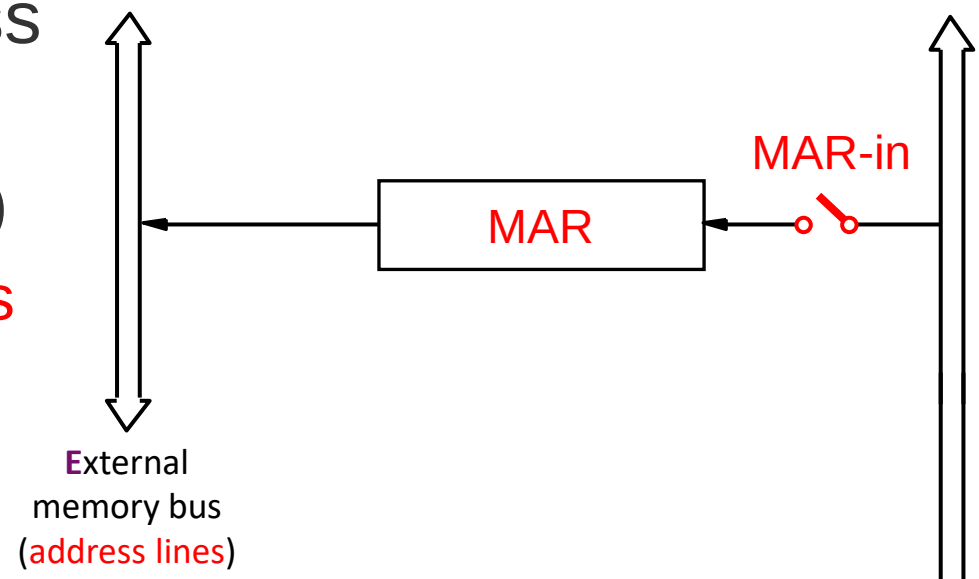
Some machines may put destination last:

```
operation src, dest
```

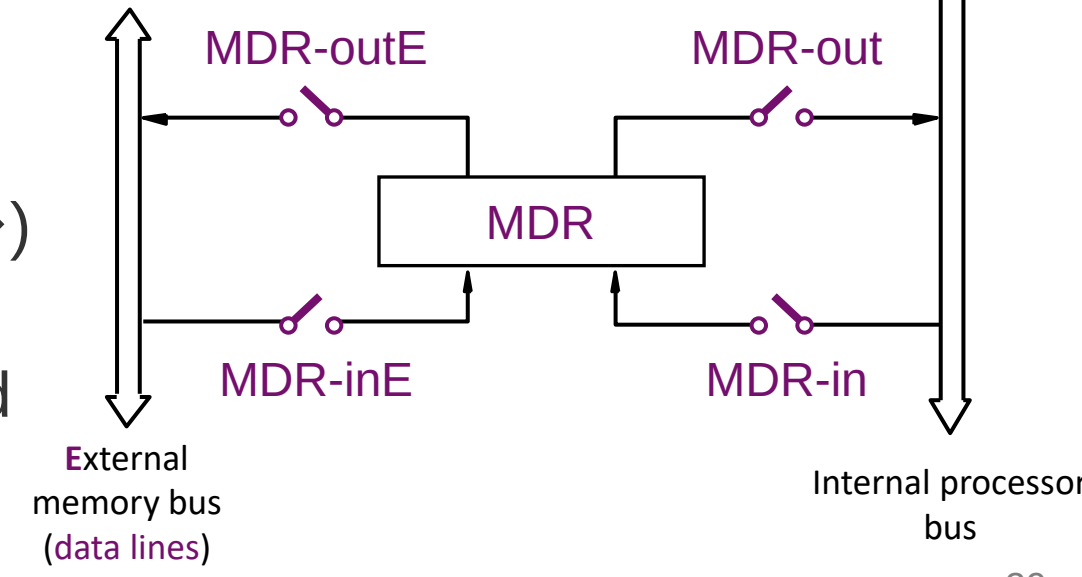
  - Sometimes operations are defined by using mnemonics.
    - **Mnemonics**: abbreviations of the words describing operations
    - E.g. **Load** can be written as **LD**, **Store** can be written as **STR** or **ST**.

3) Loading Word from Memory

- **MAR**: Memory Address Register
  - Uni-directional bus (←)
  - Connect to the address lines directly



- **MDR**: Memory Data Register
  - Bi-directional bus (←→)
  - MDR connections to buses are all controlled by switches ( ).



3) Loading Word from Memory (Cont'd)

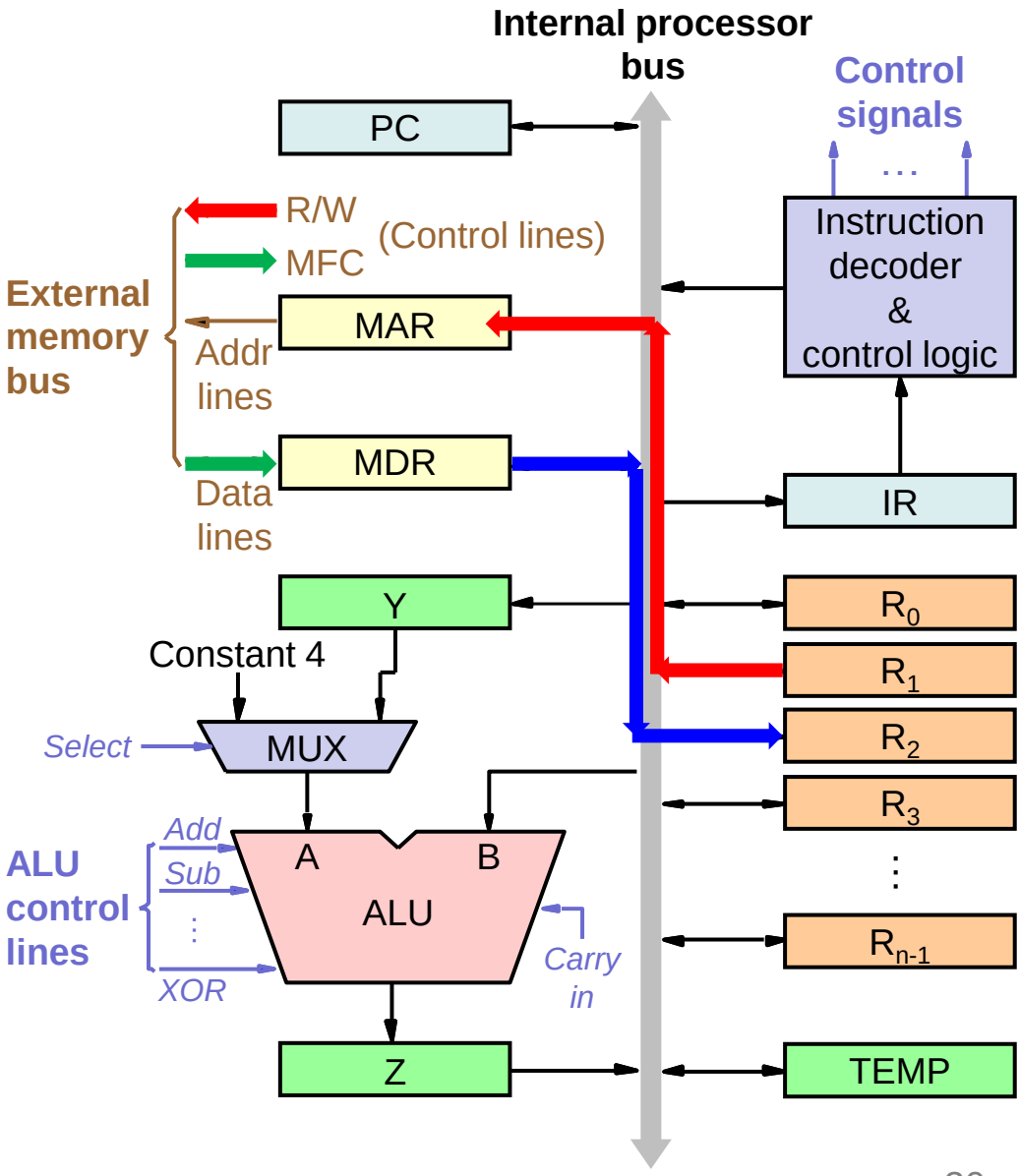
- Ex: **Mov R2, (R1)**

Sequence of Steps:

①➡R1-out,
MAR-in,
Read *(start to load a word from memory)*

②➡MDR-inE,
WaitMFC *(wait until the loading is completed)*

③➡MDR-out,
R2-in



3) Loading Word from Memory (Cont'd)

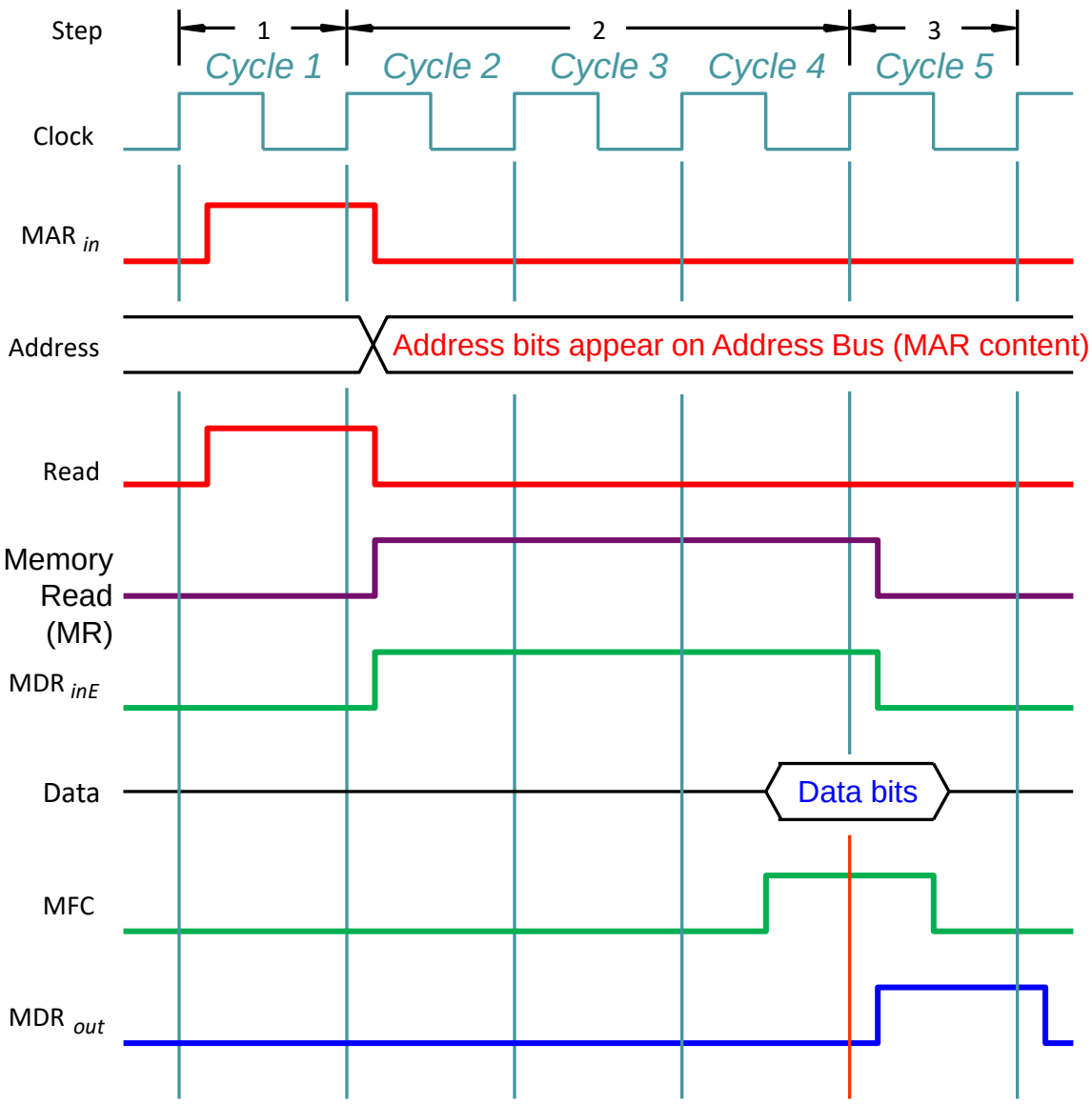
- **Timing Sequence:**

①➡R1-out *(not shown)*, MAR-in, Read *(start to read a word from memory)*

=== assume memory read takes 3 cycles ===

②➡MDR-inE, WaitMFC *(wait until the loading is completed)*

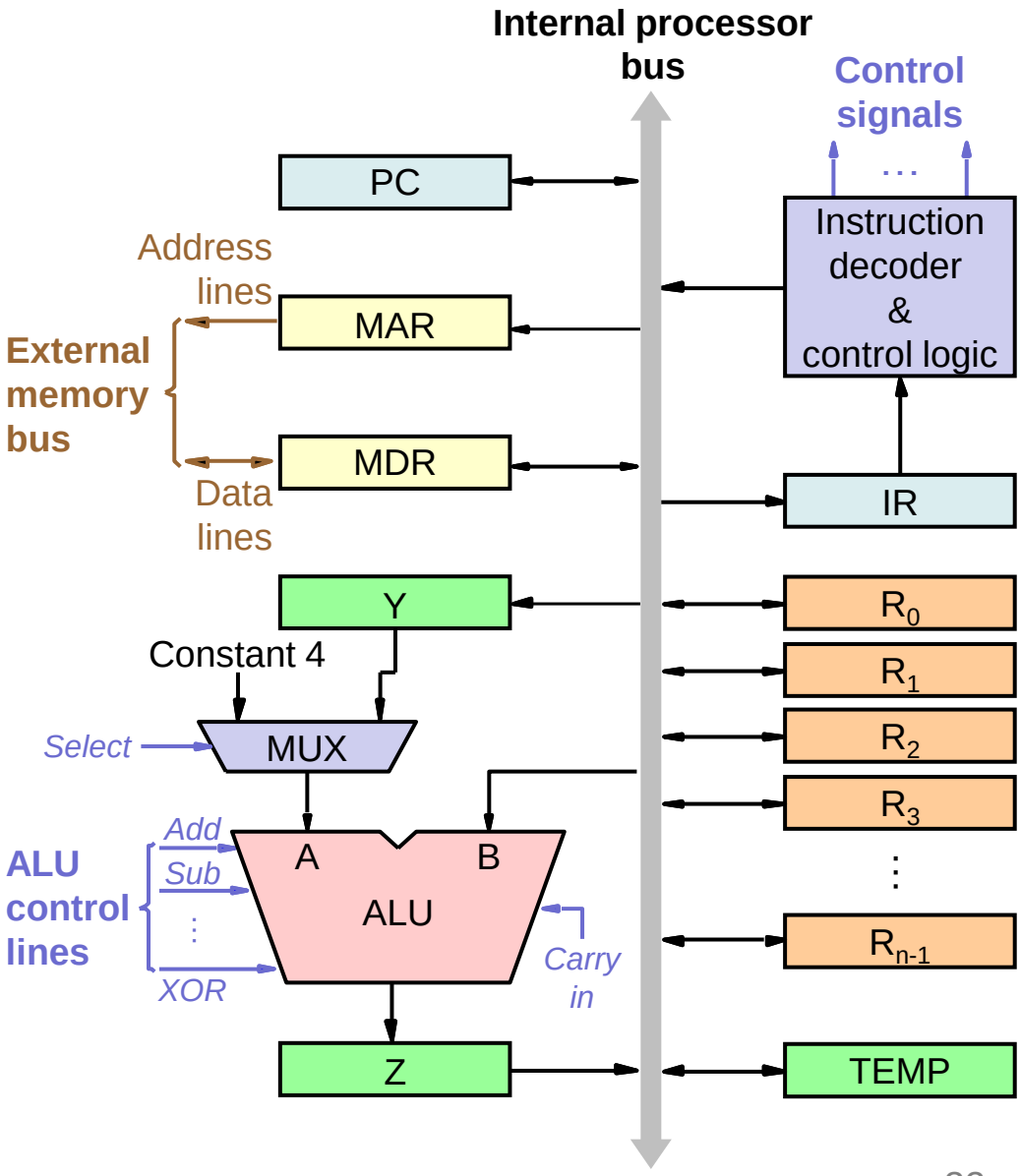
③➡MDR-out, R2-in *(not shown)*



Class Exercise 10.3

- What is the sequence of steps for the following operation?

  Mov R4, (R3)



4) Storing Word to Memory

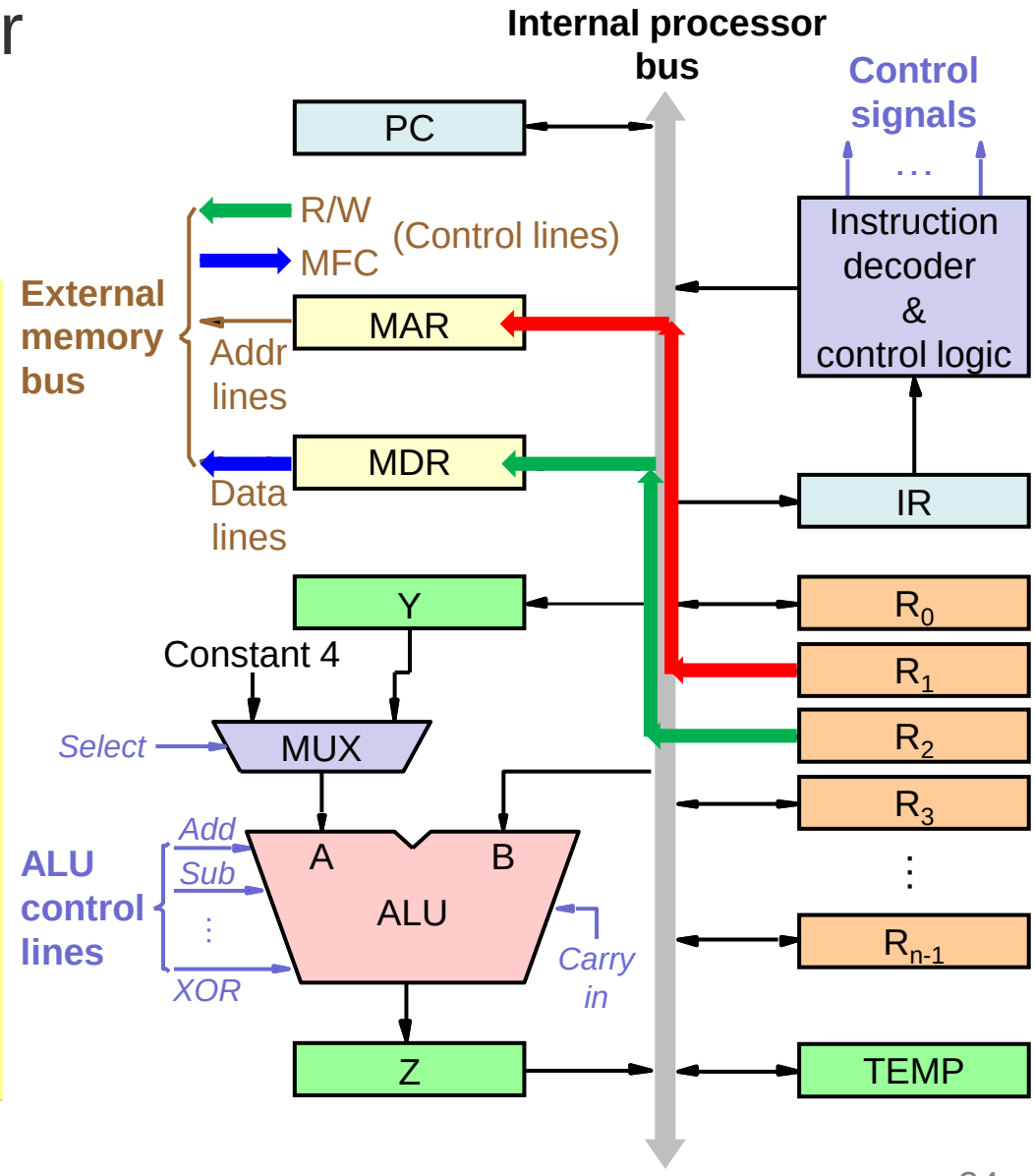
- This operation is similar to the previous one.
- Ex: **Mov (R1), R2**

Sequence of Steps:

①➡R1-out, MAR-in

②➡R2-out, MDR-in, Write *(start to store a word into memory)*

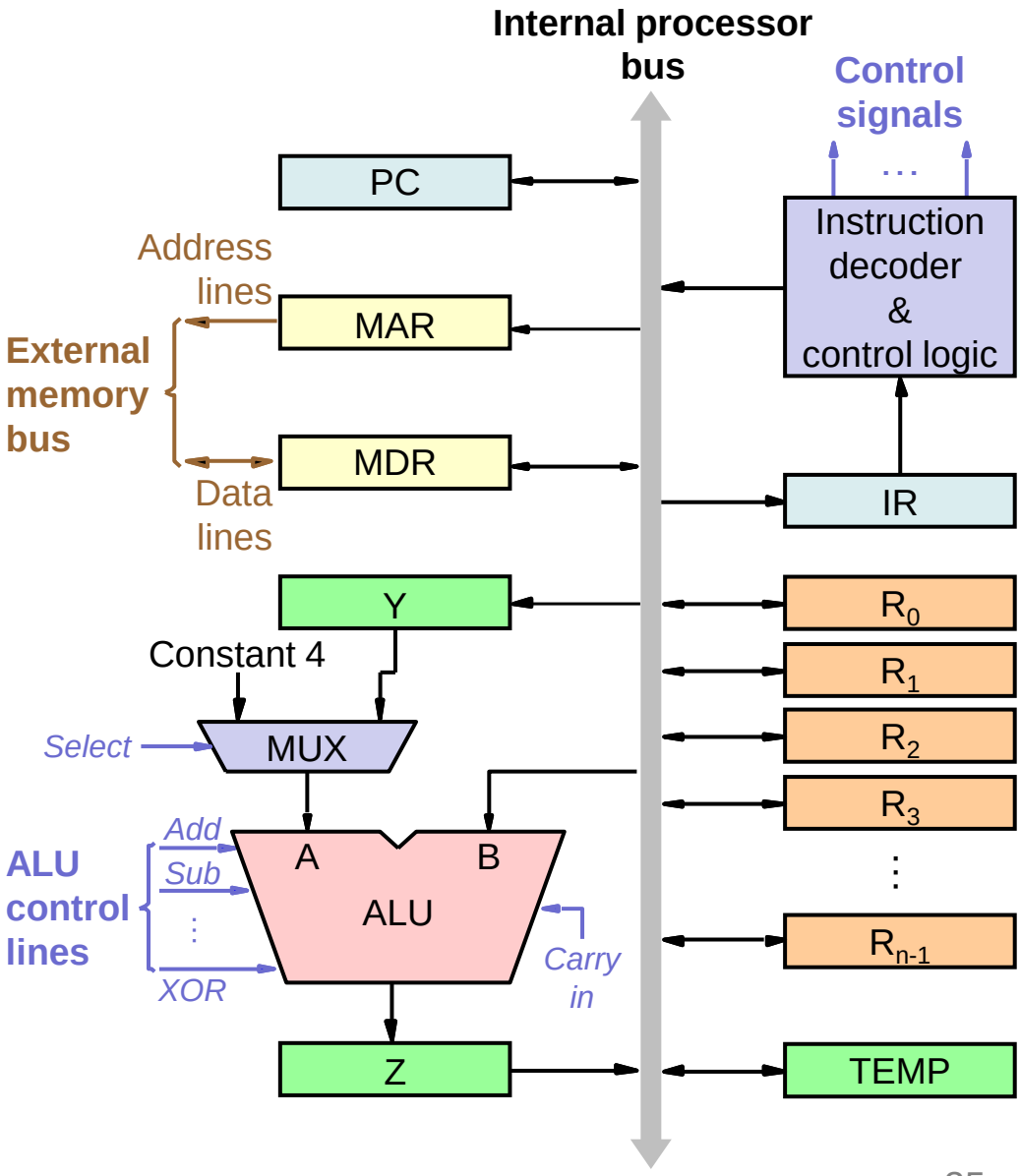
③➡MDR-outE, WaitMFC *(wait until the storing is completed)*



Class Exercise 10.4

- What is the sequence of steps for the following operation?

  Mov (R3), R4



Loading Word vs Storing Word

- **Loading Word**
- Ex: **Mov R2, (R1)**

① R1-out,
MAR-in,
Read

② **MDR-inE**,
WaitMFC

③ **MDR-out**,
R2-in

- **Storing Word**
- Ex: **Mov (R1), R2**

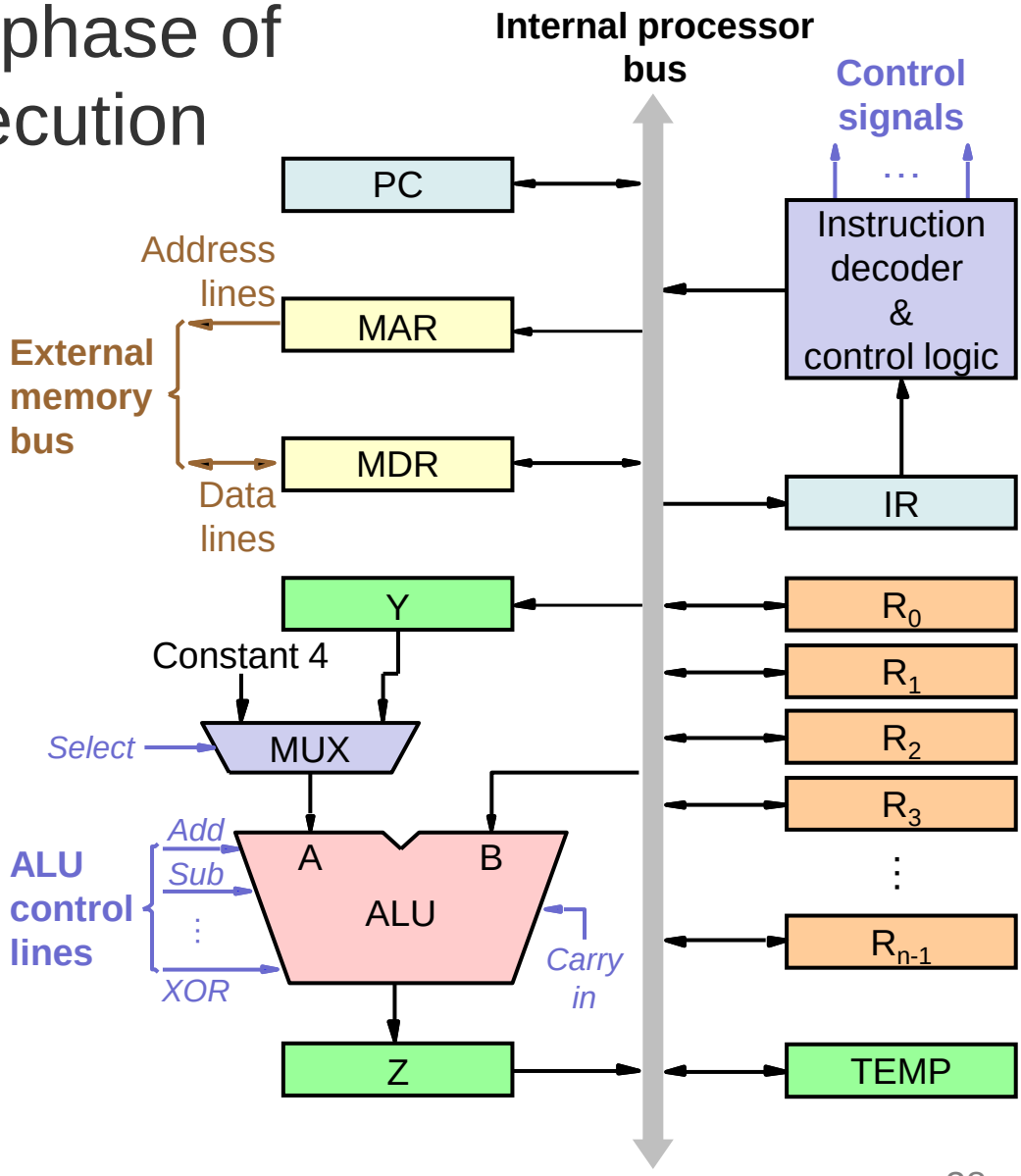
① R1-out,
MAR-in

② R2-out,
MDR-in,
Write

③ **MDR-outE**,
WaitMFC

Revisit: Fetch Phase

- **Fetch Phase**: The first phase of machine instruction execution
  - **IR ← [[PC]]**
    - Fetch the instruction from the memory location pointed to by PC, and load it into IR
  - **PC ← [PC]+4**
    - Increment the contents of PC by 4
- What is the sequences of steps for the fetch phase with the highest parallelism?



Fetch Phase (1/3)

- Ex: **Fetch Phase**

Sequence of Steps:

① PC-out, MAR-in, Read
Select-4, B-in,
Z-in, Add

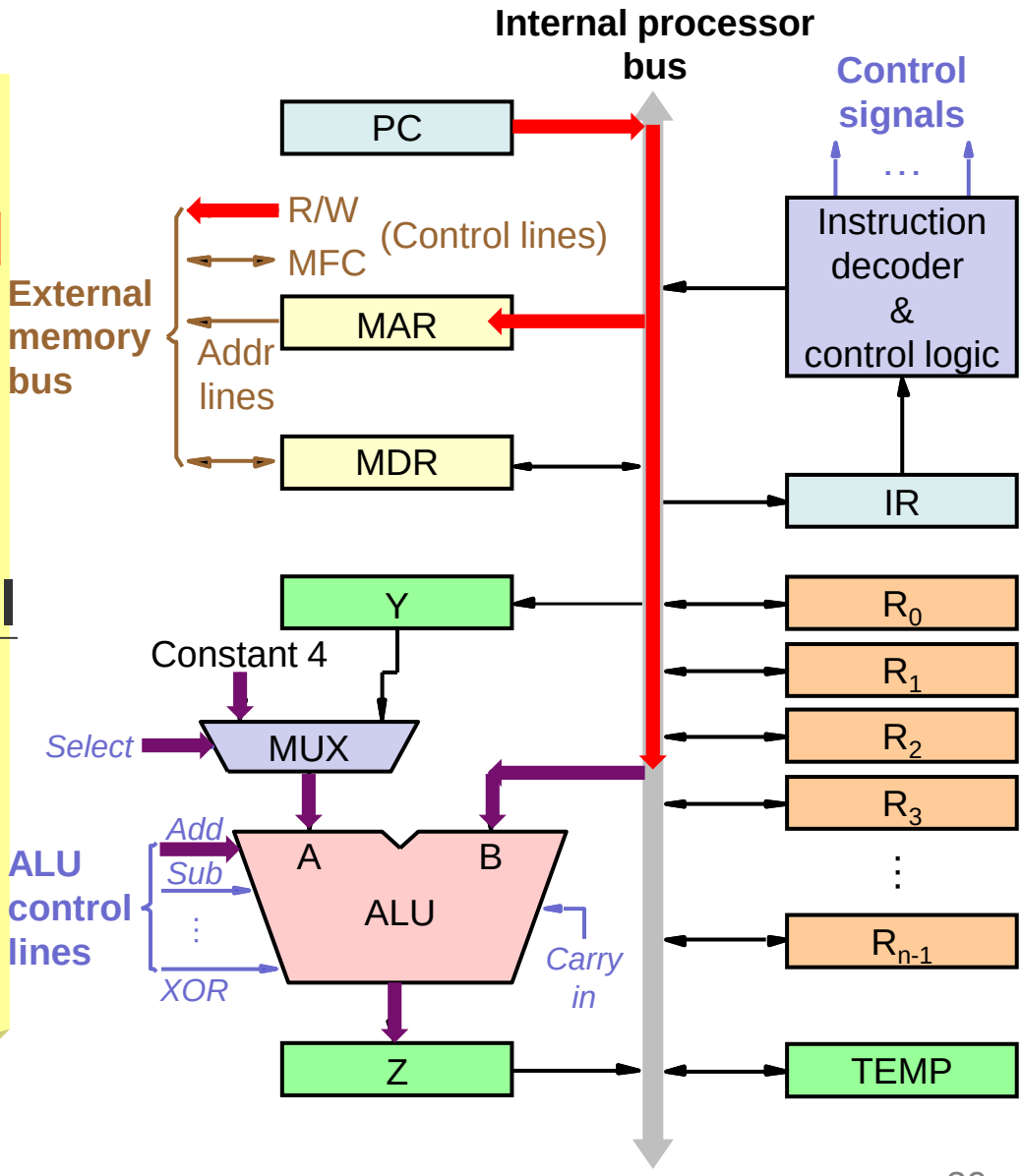
➡ Fetch the instruction

➡ Increment PC **in parallel**

② MDR-inE, WaitMFC
Z-out, PC-in, Y-in

– Y-in is for branch (discuss later).

③ MDR-out, IR-in



Fetch Phase (2/3)

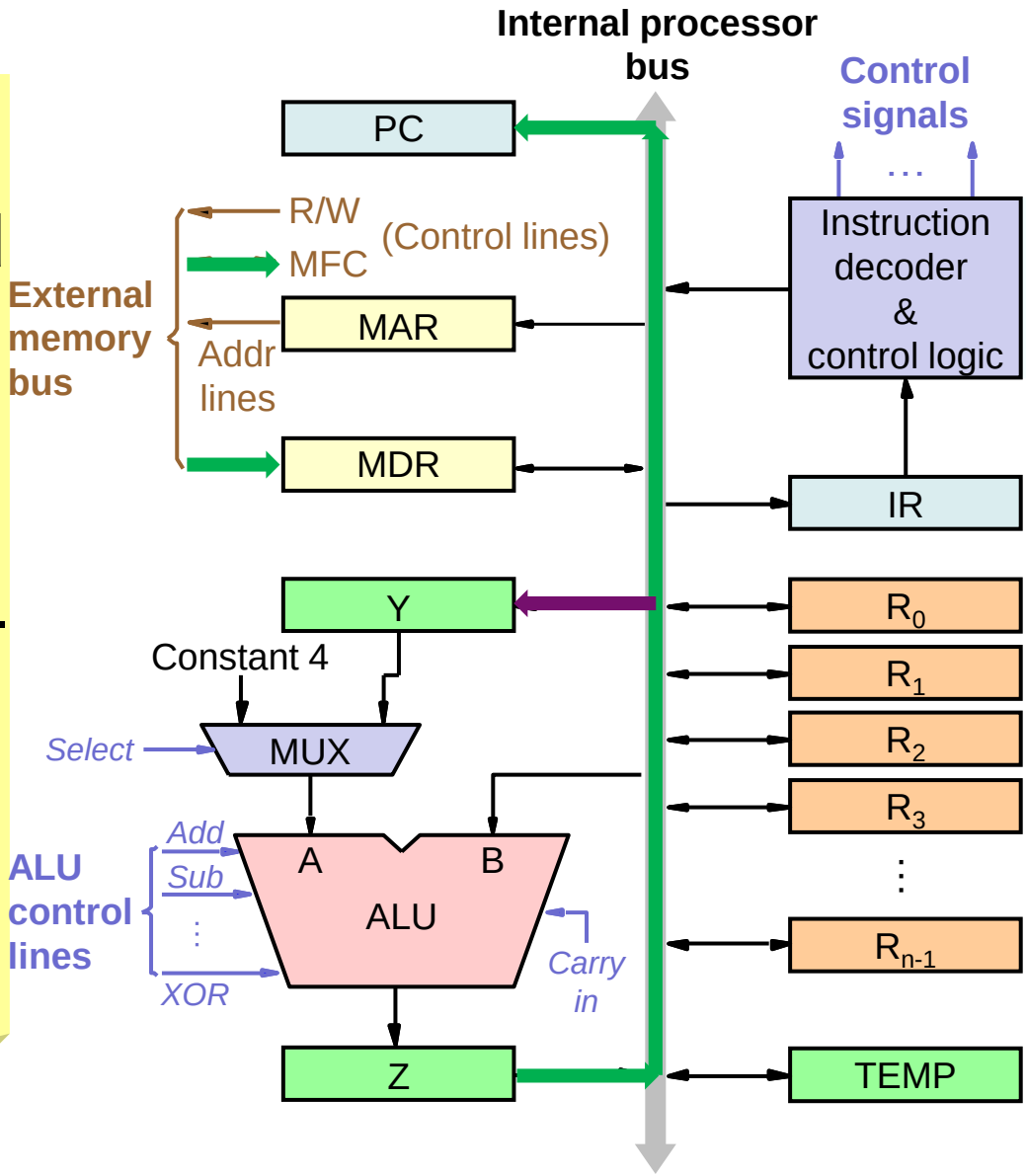
- Ex: **Fetch Phase**

Sequence of Steps:

① PC-out, MAR-in, Read
Select-4, B-in,
Z-in, Add
  - Fetch the instruction
  - Increment PC in parallel.

② MDR-inE, WaitMFC
Z-out, PC-in, Y-in
  - Y-in is for branch (discuss later).

③ MDR-out, IR-in



Fetch Phase (3/3)

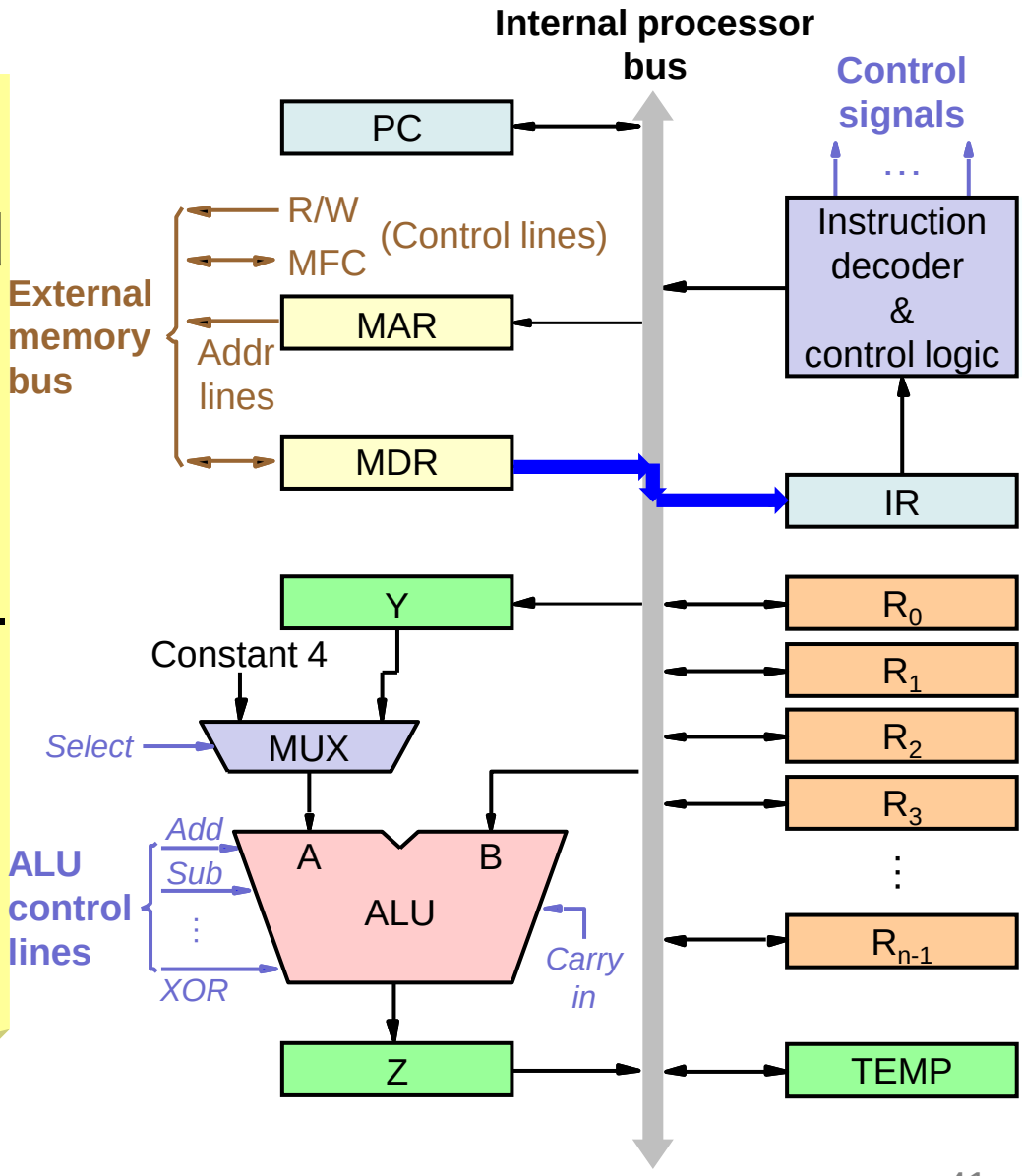
- Ex: **Fetch Phase**

Sequence of Steps:

① PC-out, MAR-in, Read Select-4, B-in, Z-in, Add
  - Fetch the instruction
  - Increment PC in parallel.

② MDR-inE, WaitMFC Z-out, PC-in, Y-in
  - Y-in is for branch (discuss later).

③ MDR-out, IR-in



Observations and Insights

- The internal processor bus and the external memory bus can be operated independently (concurrently).
  - Since the separation provided by MAR and MDR.

- Independent operations imply the possibility of performing some steps in parallel.
  - E.g., memory access and PC increment, instruction decoding and reading source register

- During memory access, processor waits for MFC.
  - There is NOTHING TO DO BUT WAIT for few cycles.
  - *Question: Any way to improve this situation?*

Outline

- Processor Internal Structure
- Instruction Execution
  - Fetch Phase
  - Execute Phase
- Execution of A Complete Instruction
- Multiple-Bus Organization

Example 1) ADD R1, (R3) (1/3)

- Instruction Execution: Fetch Phase & Execute Phase

1) Fetch the instruction

2) Decode the instruction

3) Load the operand [R3] from memory

4) Perform the addition

5) Store result to R1

Sequence of Steps:

① PC-out, MAR-in, Read
Select-4, B-in, Z-in, Add

② MDR-inE, WaitMFC
Z-out, PC-in, Y-in

③ MDR-out, IR-in

④ **DecodeInstruction**

⑤ R3-out, MAR-in, Read

⑥ R1-out, Y-in, MDR-inE, WaitMFC

⑦ MDR-out, SelectY, Add, Z-in, B-in

⑧ Z-out, R1-in

Example 1) ADD R1, (R3) (2/3)

Sequence of Steps:

① PC-out, MAR-in, Read
Select-4, B-in, Z-in, Add

② MDR-inE, WaitMFC
Z-out, PC-in, Y-in

③ MDR-out, IR-in

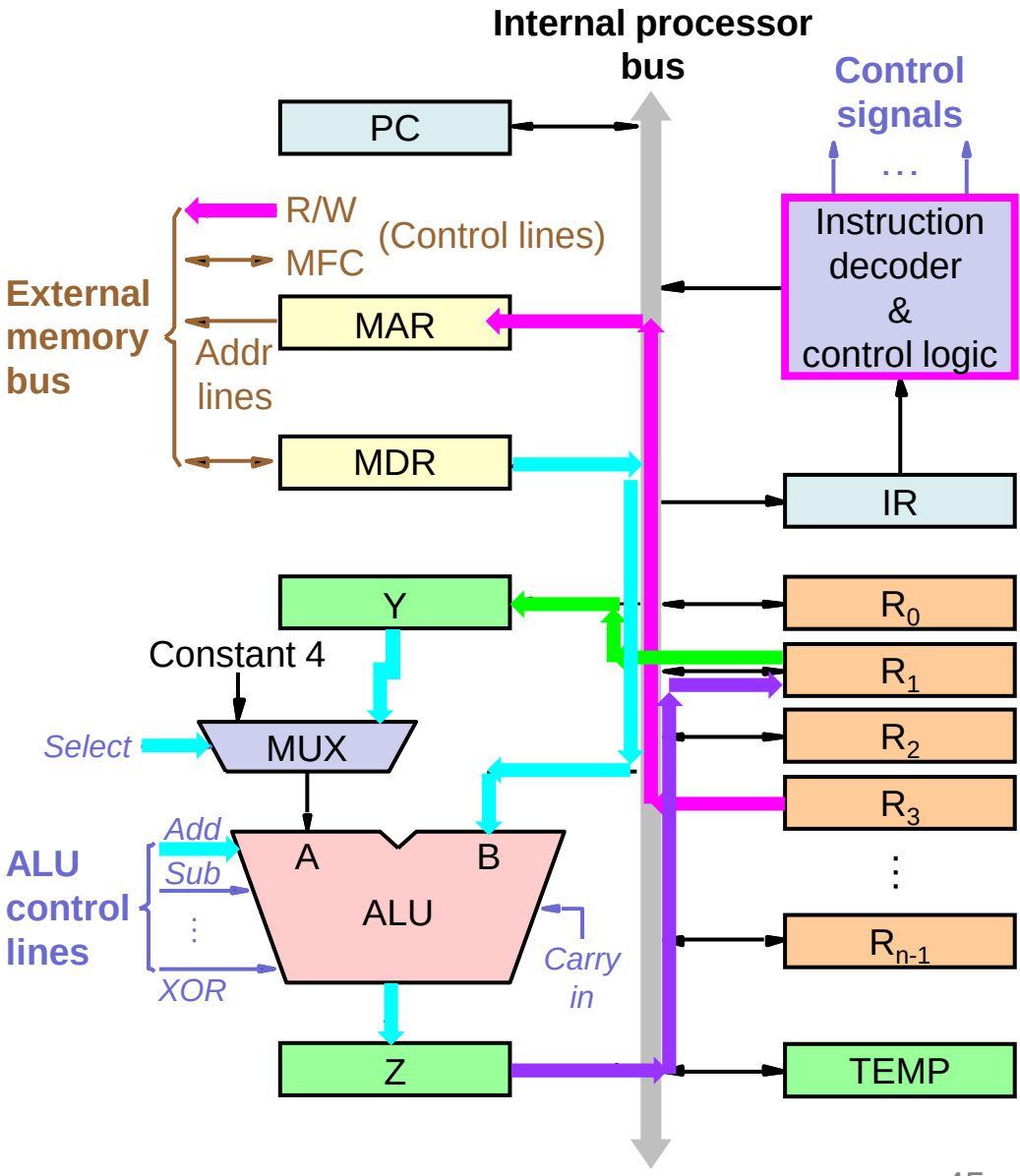
④ **DecodeInstruction**

⑤ R3-out, MAR-in, Read

⑥ R1-out, Y-in, MDR-inE, WaitMFC

⑦ MDR-out, SelectY, Add, Z-in, B-in

⑧ Z-out, R1-in



Example 1) ADD R1, (R3) (3/3)

- Detailed Explanation for Sequence of Steps:
  - ① PC loaded into MAR, read request to memory, MUX selects 4, added to PC (B-in) in ALU, store sum in Z
  - ② Z moved to PC (and Y) while waiting for memory
  - ③ Word fetched from memory and loaded into IR
  - ④ **Instruction Decoding**: Figure out what the instruction should do and set control circuitry for steps 4 – 7
  - ⑤ R3 transferred to MAR, read request to memory
  - ⑥ Content of R1 moved to Y while waiting for memory
  - ⑦ Read operation completed, the loaded word is already in MDR and copied to B-in of ALU, SelectY as second input of ALU, add performed
  - ⑧ Result is transferred to R1

Example 2) Branch Instruction (1/2)

- Instruction Execution: Fetch Phase & Execute Phase

1) Fetch the instruction

2) Decode the instruction

3) Add the offset specified in the instruction (**Offset-field-of-IR**) to the PC

4) Update the PC

Sequence of Steps:

① PC-out, MAR-in, Read Select-4, B-in, Z-in, Add

② MDR-inE, WaitMFC Z-out, PC-in, **Y-in**

③ MDR-out, IR-in

④ **DecodeInstruction**

⑤ **Offset-field-of-IR-out**, SelectY, Add, Z-in, B-in

⑥ Z-out, PC-in

Example 2) Branch Instruction (2/2)

Sequence of Steps:

① PC-out, MAR-in, Read Select-4, B-in, Z-in, Add

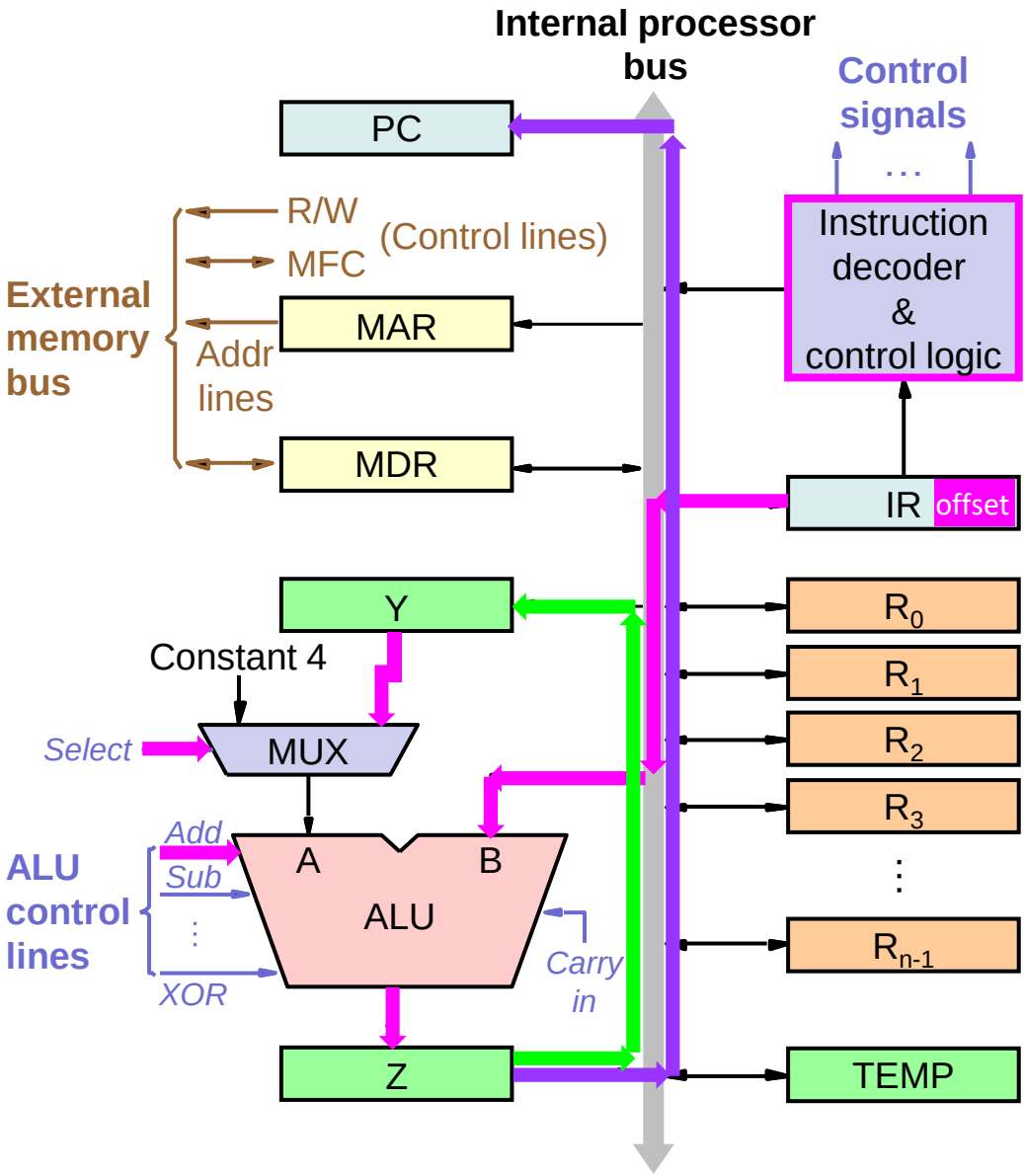
②➡MDR-inE, WaitMFC Z-out, PC-in, **Y-in**

③ MDR-out, IR-in

④ **DecodeInstruction**

⑤➡**Offset-field-of-IR-out**, SelectY, Add, Z-in, B-in

⑥➡Z-out, PC-in

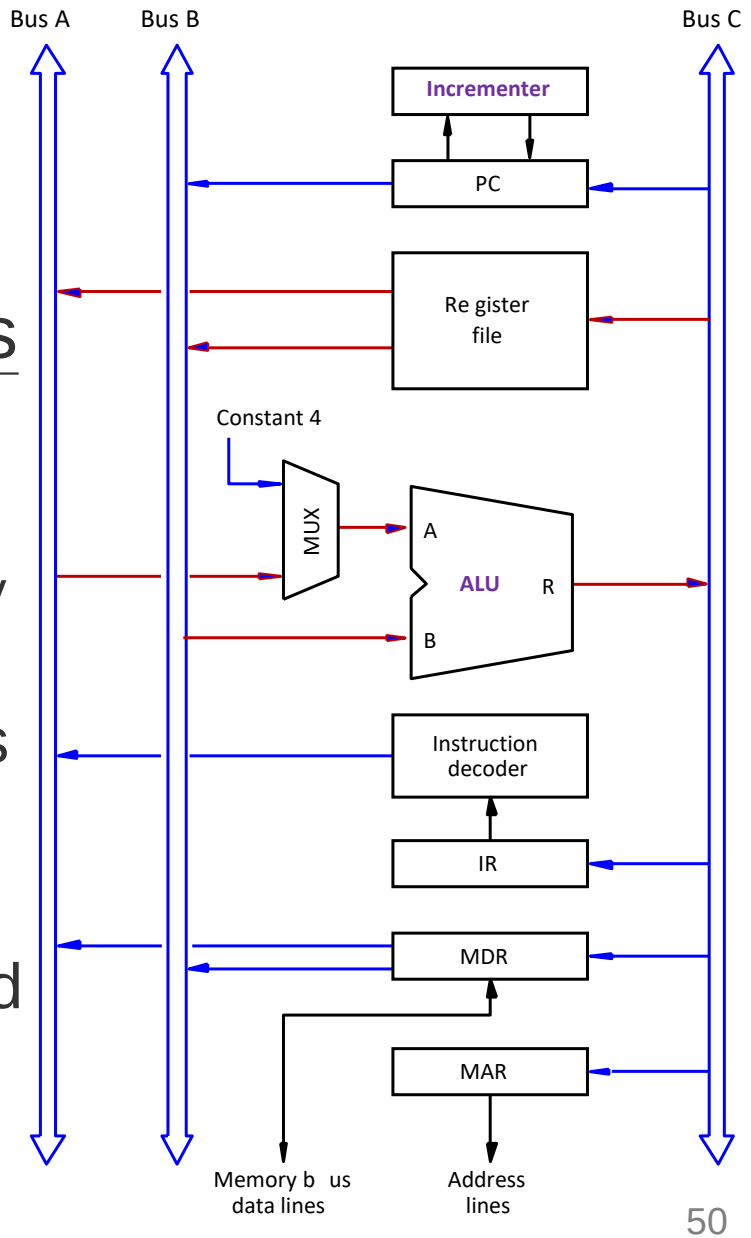


Outline

- Processor Internal Structure
- Instruction Execution
  - Fetch Phase
  - Execute Phase
- Execution of A Complete Instruction
- Multiple-Bus Organization

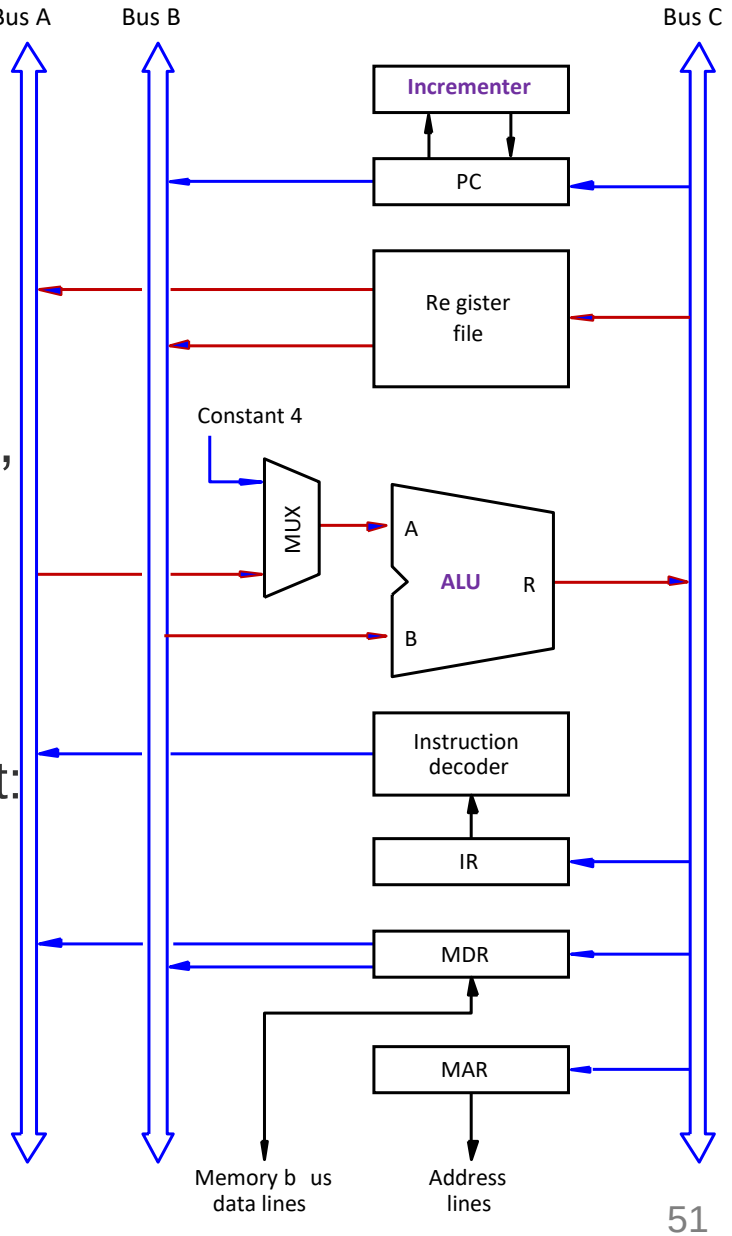
Multiple Internal Buses (1/2)

- Disadvantage of single bus: Only one data item can be transferred internally at a time.
- Solution: Multiple Internal Buses
  - All registers combined into a register file with 3 ports
    - TWO out-ports and ONE in-port (Why 3? Instruction format!).
  - Buses A and B allow simultaneous transfer of the two operands for the ALU.
  - Bus C can transfer data into a third register during the same clock cycle.



Multiple Internal Buses (2/2)

- Solution: Multiple Internal Buses
  - ALU is able to just pass one of its operands to output R
    - E.g. **R=A** or **R=B**
  - Employ an additional "Incrementer" unit to compute [PC]+4 (**IncPC**)
    - ALU is not used for incrementing PC.
    - ALU still has a Constant 4 input for other instructions (e.g., post-increment: [SP]++ for stack push).



Class Exercise 10.5

- Can you tell what does the following execution do?

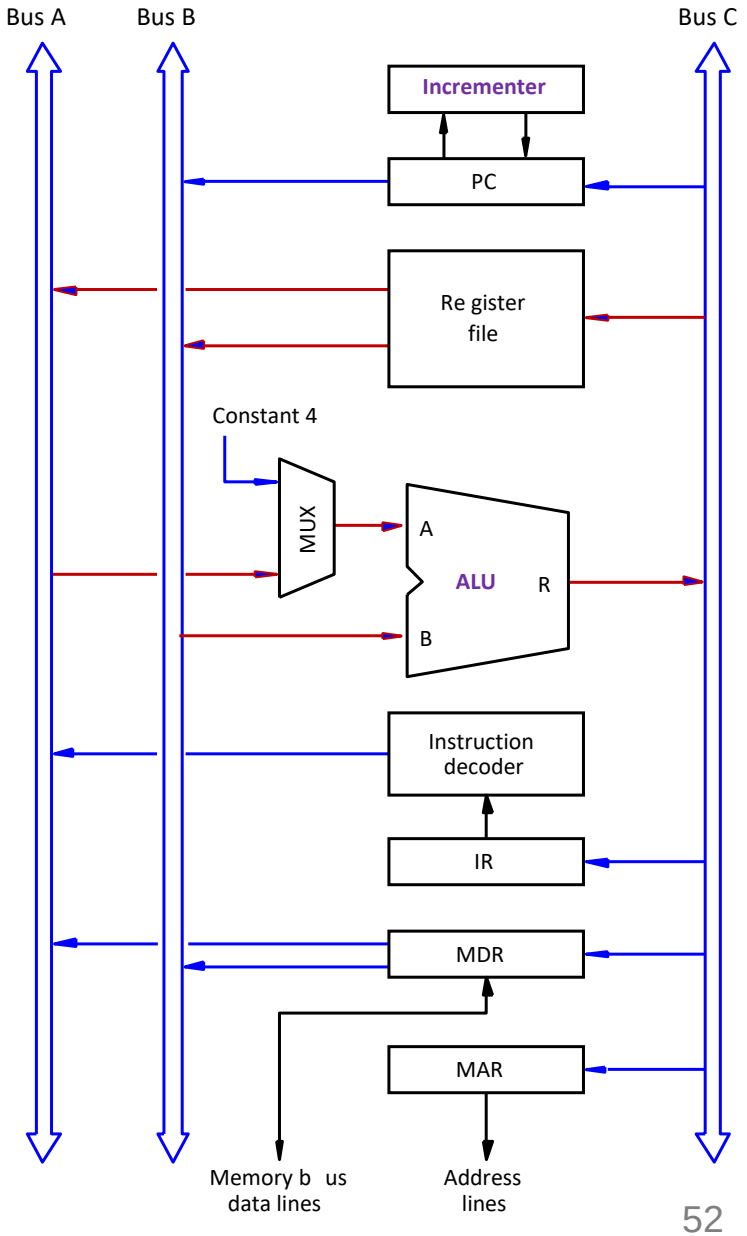
① PC-out, MAR-in, Read, **R=B**

② MDR-inE, WaitMFC, **IncPC**

③ MDR-out, IR-in, **R=B**

④ DecodeInstruction

⑤ **R4-outA, R5-outB, SelectA, Add, R6-in**



Summary

- Processor Internal Structure
- Instruction Execution
  - Fetch Phase
  - Execute Phase
- Execution of A Complete Instruction
- Multiple-Bus Organization