香港中文大學
The Chinese University of Hong Kong

CSCI2510 Computer Organization

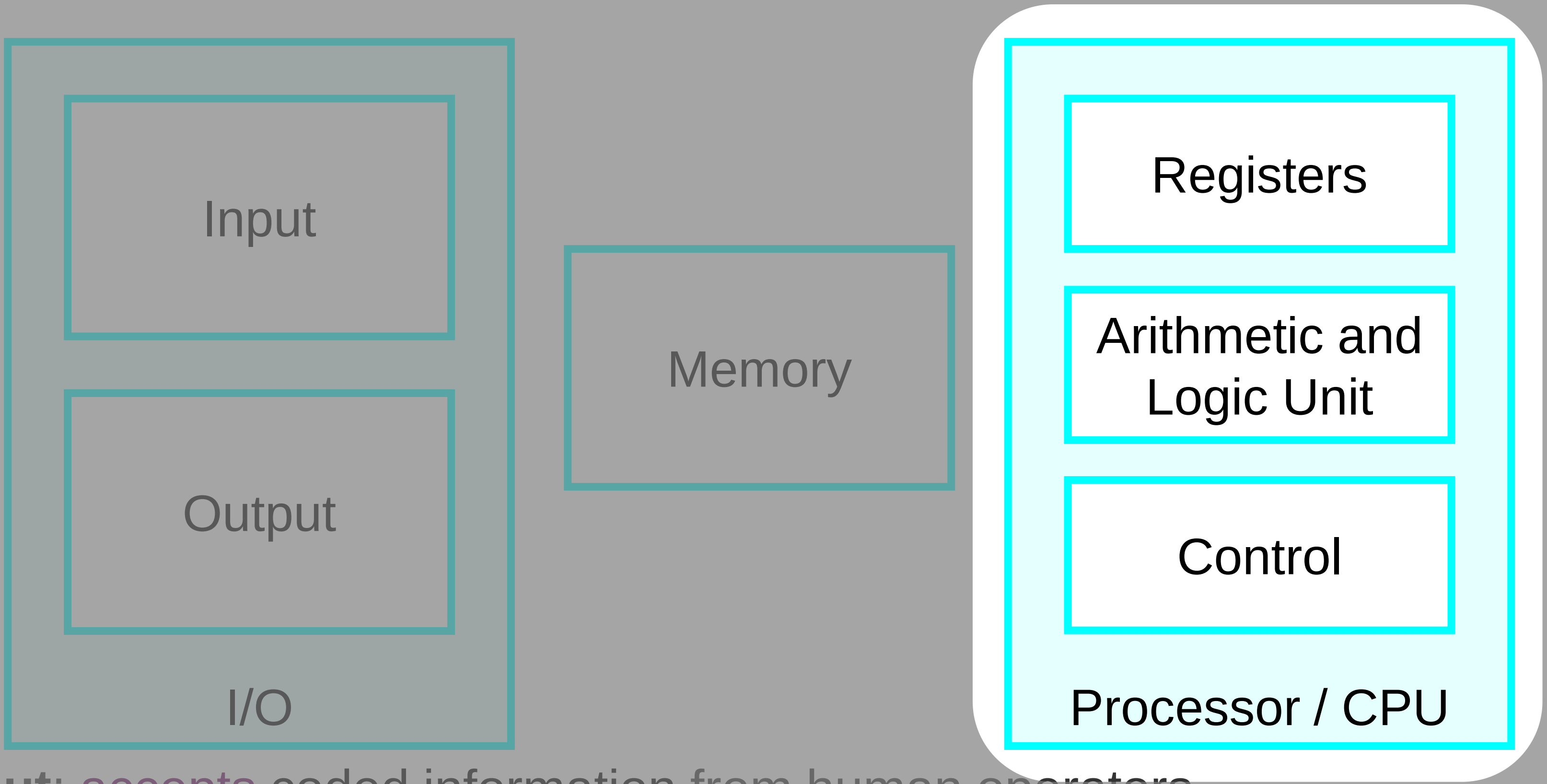
Lecture 09:
Basic Processing Unit

Ming-Chang YANG

mcyang@cse.cuhk.edu.hk

Reading: Chap. 2.13, 5

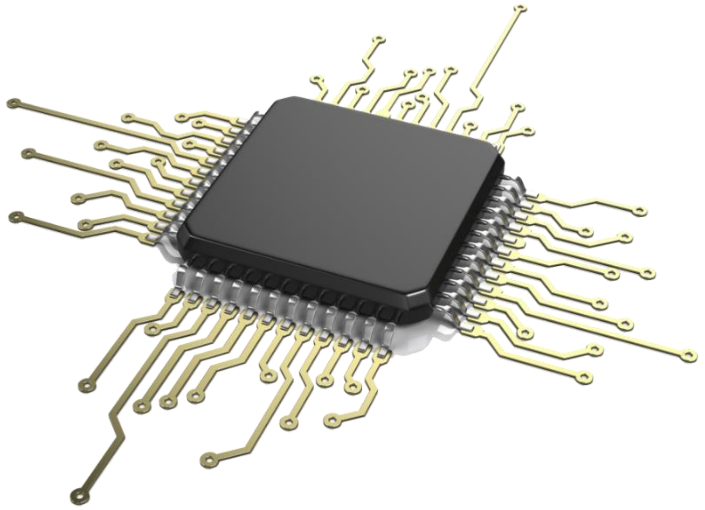
Basic Functional Units of a Computer



- **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.

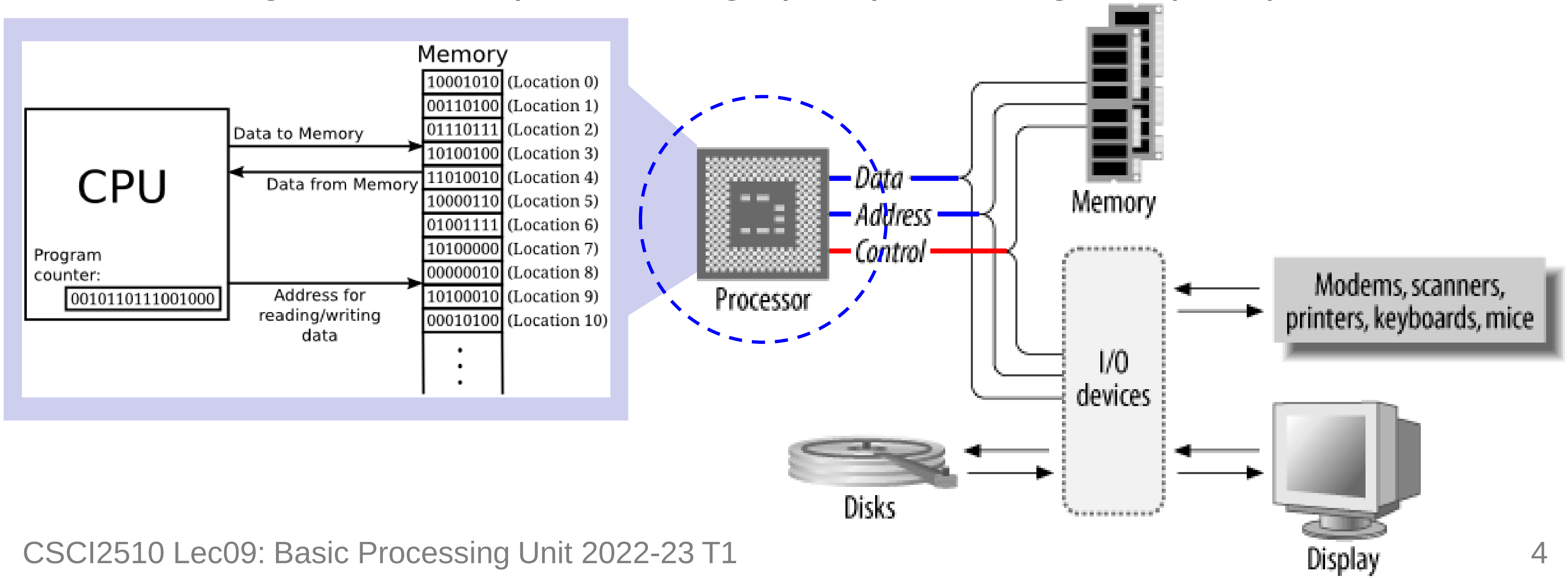
Outline

- Main Components of a Processor
- RISC-Style Processor Design
  - Five-Stage Organization
  - Instruction Execution
- CISC-Style Processor Design
  - Multi-Bus Interconnect
  - Instruction Execution
- Control Signal Generation



Basic Processing Unit: Processor

- Executing machine-language instructions.
- Coordinating other units in a computer system.
- Used to 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

Control Circuitry

Interpret/decode the fetched instruction & issue control signals to coordinate all the other units

Register File (i.e., R_0~R_{n-1})

Served as the processor's general-purpose registers

Arithmetic and Logic Unit (ALU)

Perform arithmetic or logic operations

Instruction Address Generator

Program Counter (PC)

Keep the address of the next instruction to be fetched and executed (special register)

Update the contents of PC after every instruction is fetched

Instruction Register (IR)

Hold the instruction until its execution is completed (special register)

Other Special Registers

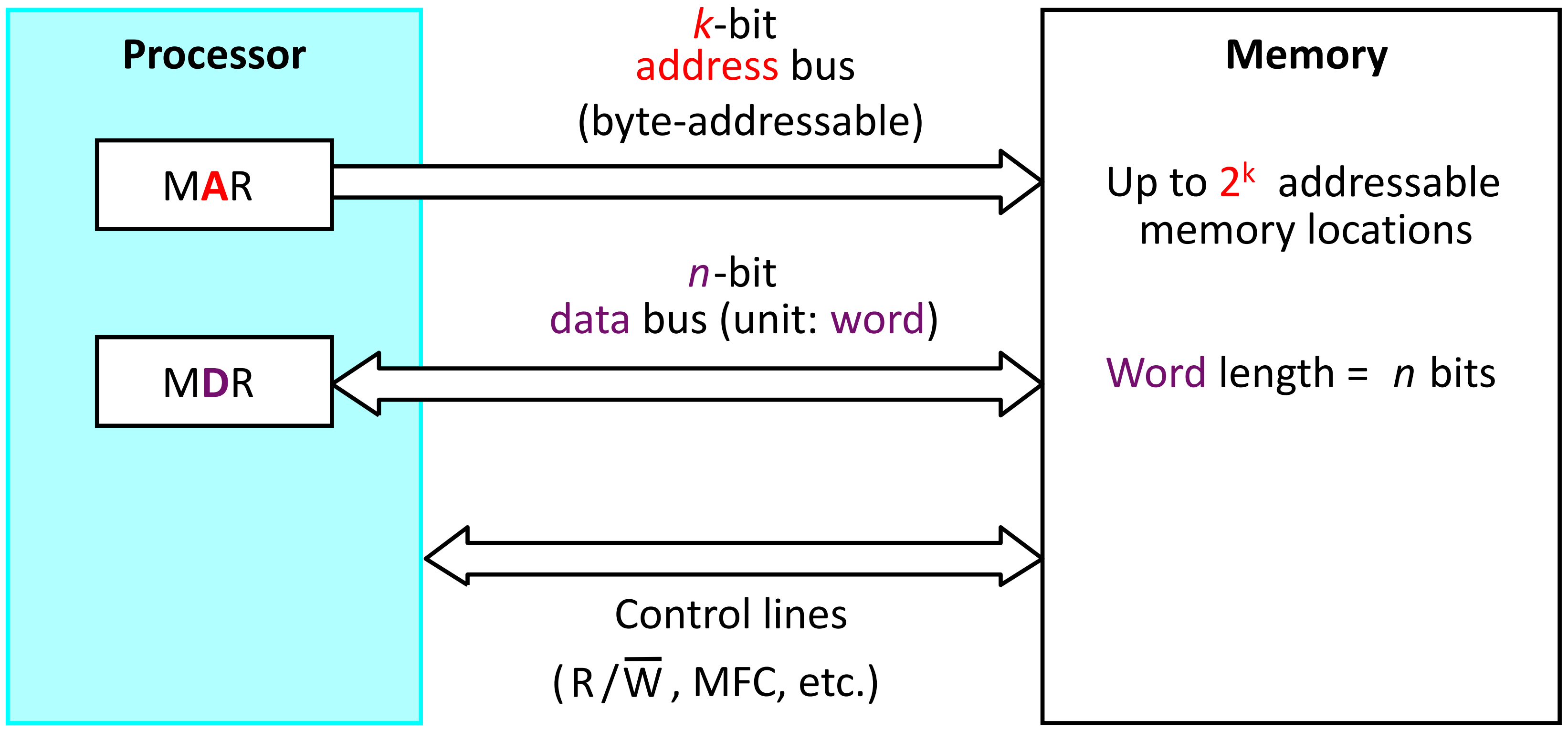
E.g., memory address register, memory data register, condition code register, stack pointer register, link register, etc.

Processor–Memory Interface

Allow communication between processor and memory through two special registers: memory address register (MAR) and memory data register (MDR)

Recall: Processor-Memory

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


Processor
MAR
MDR
k-bit address bus (byte-addressable)
n-bit data bus (unit: word)
Control lines
($R/\overline{W}$, MFC, etc.)
Memory
Up to 2^k addressable memory locations
Word length = *n* bits

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

Recall: RISC vs. CISC Styles

- There are two fundamentally different approaches in the design of instruction sets for modern computers:
  1) **Reduced Instruction Set Computer (RISC)** reduces the complexity/types of instructions for higher performance.
     - Each instruction fits in a single word in memory.
     - A load/store architecture is adopted.
       - Memory operands are accessed only using `Load/Store` instructions.
       - The operands involved in arithmetic/logic operations must be either in registers or given explicitly within the instruction.
  2) **Complex Instruction Set Computer (CISC)** allows more complicated but powerful instructions to be designed.
     - Each instruction may span more than one word in memory.
     - The operands involved in arithmetic/logic operations can be in both registers and memory or given explicitly within the instruction.
     - Two-operand format is usually used.

The processor design is affected by the instruction set design!

Outline

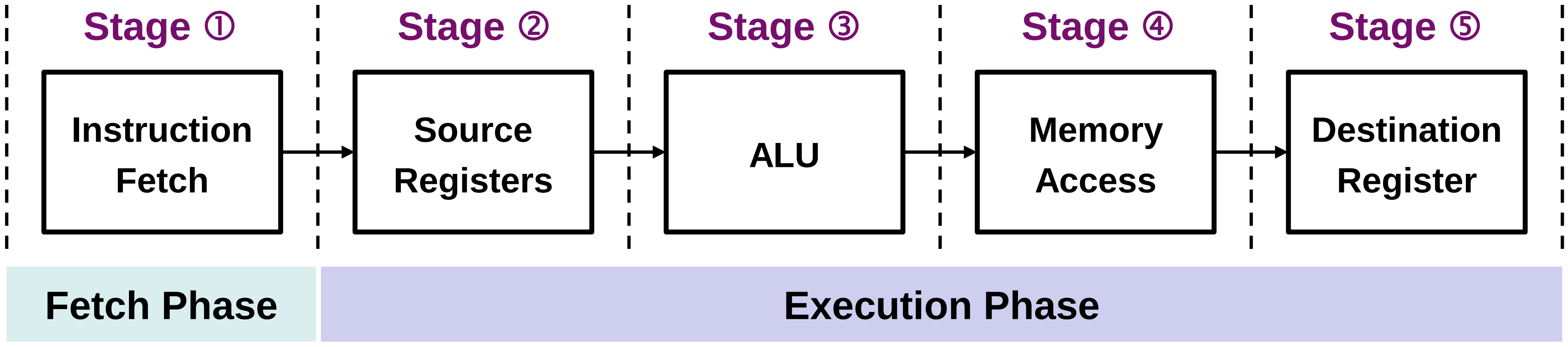
- Main Components of a Processor
- RISC-Style Processor Design
  - Five-Stage Organization
  - Instruction Execution
- CISC-Style Processor Design
  - Multi-Bus Interconnect
  - Instruction Execution
- Control Signal Generation

Five-Stage Organization (RISC CPU)

- The execution of **RISC instructions** can be *generally* organized into a five-stage sequence of actions:

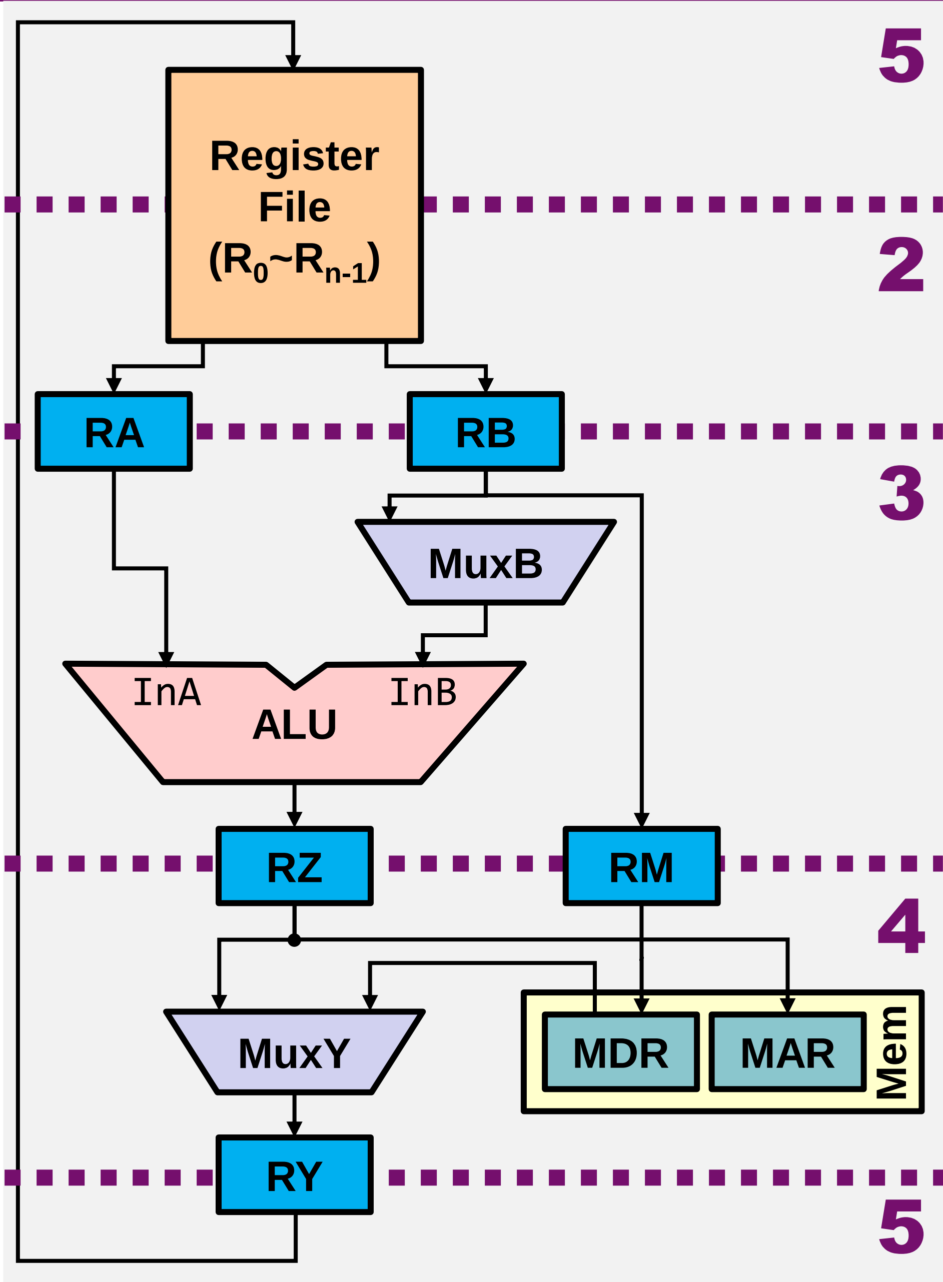
① Fetch an instruction and increment the PC.
② Decode the instruction & read source registers.
③ Perform an ALU operation.
④ Read or write memory data if memory operand is involved.
⑤ Write the result into the destination register.

Note: Not all these actions have to be carried out by every instruction.



Multi-Stage Structure: Datapath

- The processor's hardware can also be organized into multiple stages.
  - The actions taken in each stage can be completed independently and in one clock cycle (hopefully).
- It is necessary to insert inter-stage registers to:
  - Hold the produced results;
  - Work as inputs to the next.
- This multi-stage structure is often called datapath.

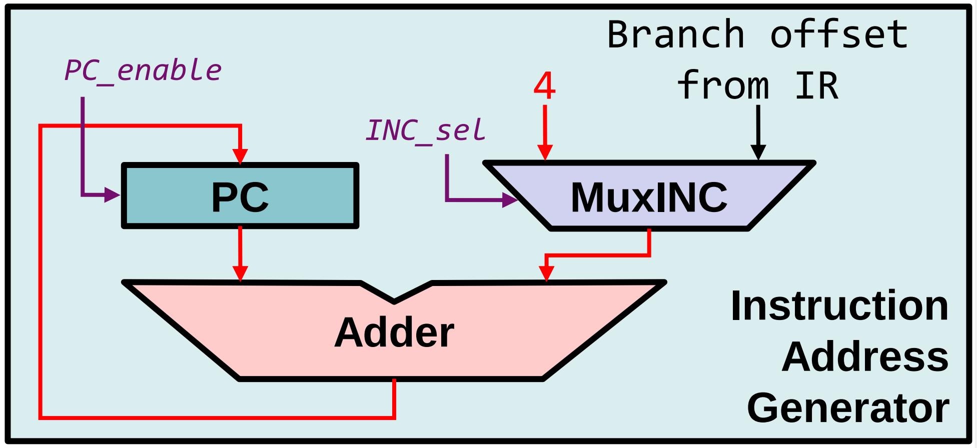


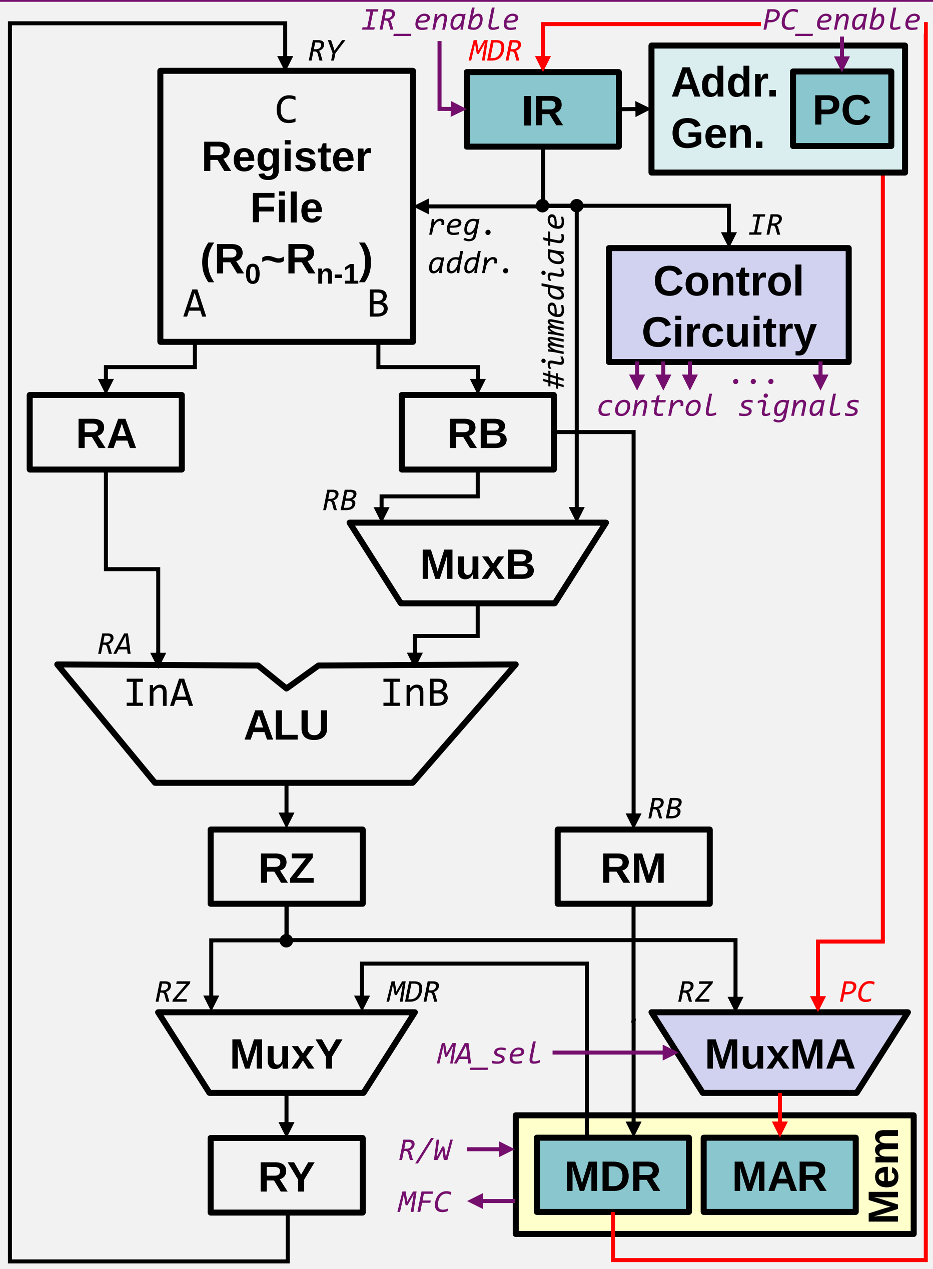
Outline

- Main Components of a Processor
- RISC-Style Processor Design
  - Five-Stage Organization
  - Instruction Execution
- CISC-Style Processor Design
  - Multi-Bus Interconnect
  - Instruction Execution
- Control Signal Generation

Fetch Phase: Stage ①

- The **fetch phase** includes two major actions:
  1) **IR ← [[PC]]**
     - Load the memory contents pointed to by **PC** into **IR**.
     - The **MuxMA** is set to select the address from **PC**.
     - **IR_enable** must be set.
  2) **PC ← [PC] + 4**
     - **PC_enable** must be set.



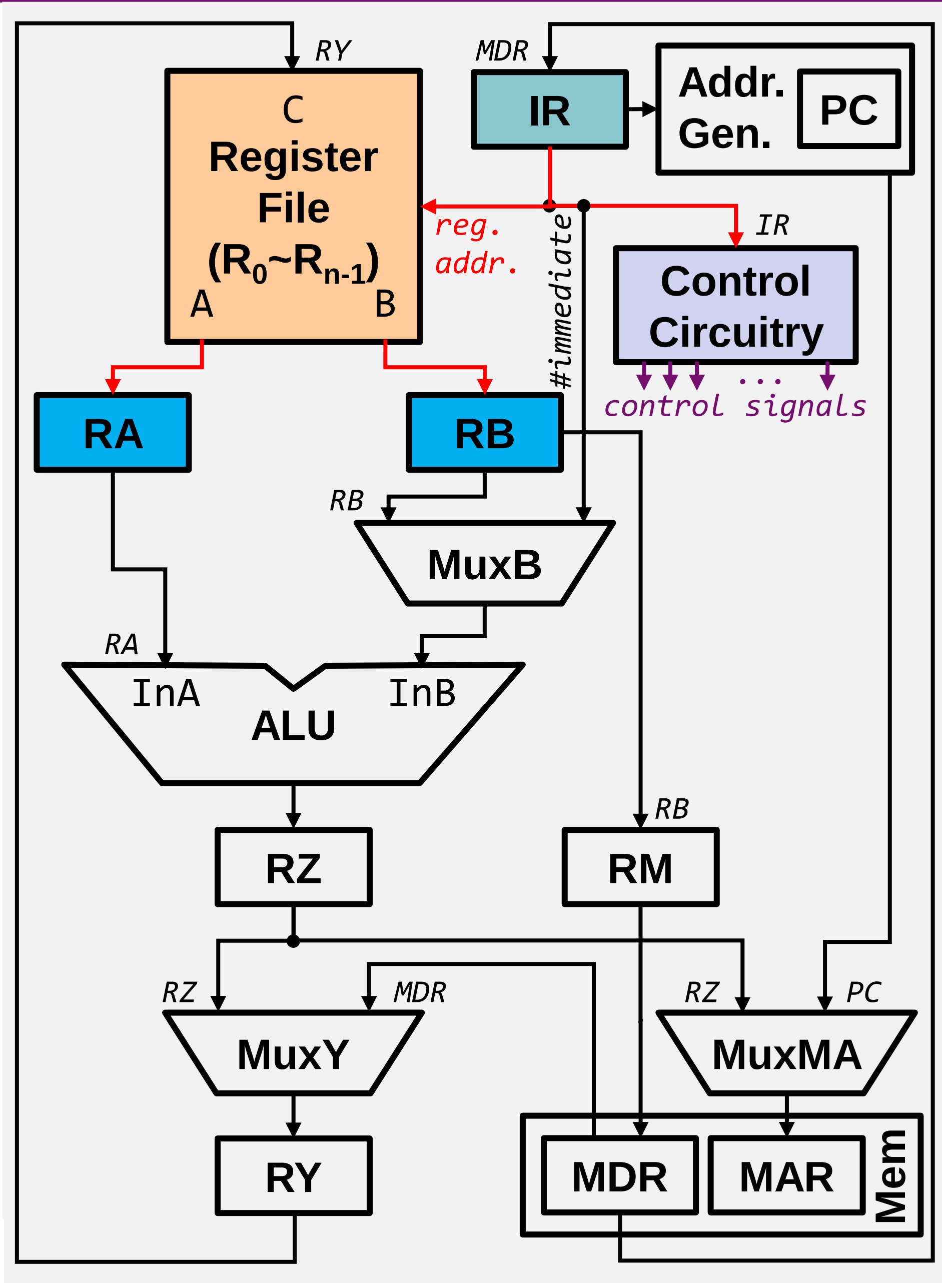


Execution Phase: Stage ②

- The **execution phase** is generally of four stages:

  ② Decode the instruction & read source registers.

  - **[IR]** must be firstly decoded by **Control Circuity.**
    - It is for generating signals to control all the hardware.
  - Two source registers can be read from **Register File** at the same time.
    - How? The source register addresses are supplied by **IR** directly (w/o decoding).
    - Two source registers are always read and placed into **RA** and **RB** (no matter whether they are needed).



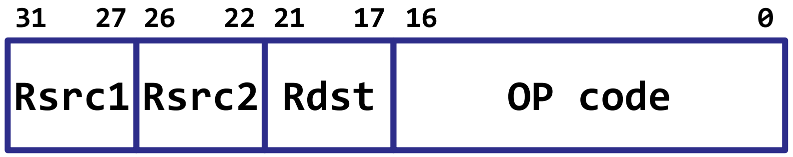
Instruction Encoding

- Consider a RISC-style processor that
  - Has 32 general-purpose registers;
  - Represents every instruction by a 32-bit word.

- Representative encoding formats include:

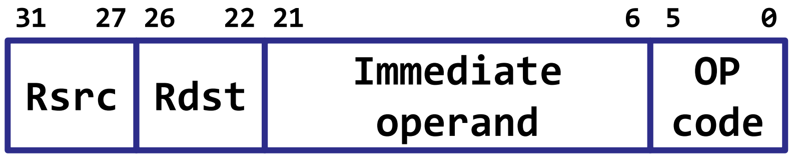
① **Three-Operand Format**

- E.g., **`Add, Rdst, Rsrc1, Rsrc2`**



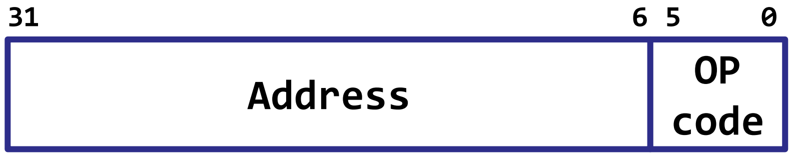
② **Immediate-Operand Format**

- E.g., **`Add Rdst, Rsrc, #Value`**
- E.g., **`Load/Store`** instruction using <u>register indirect</u> or <u>index</u> modes
- E.g., **`Branch`** instruction using offset



③ **Address-Operand Format**

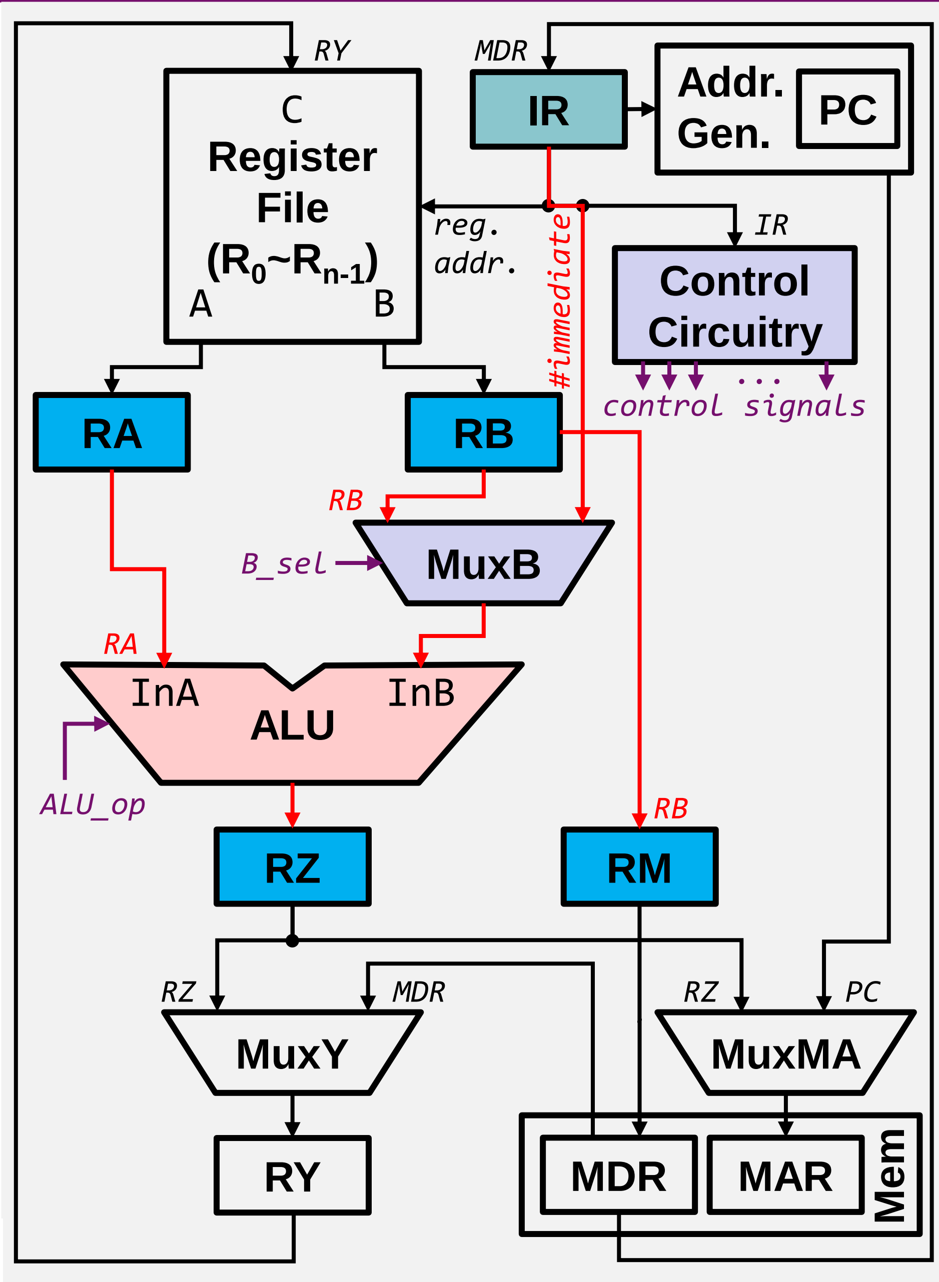
- E.g., **`Branch`** instruction
- E.g., **`Call`** instruction



The instruction encoding varies (a lot) from machine to machine!

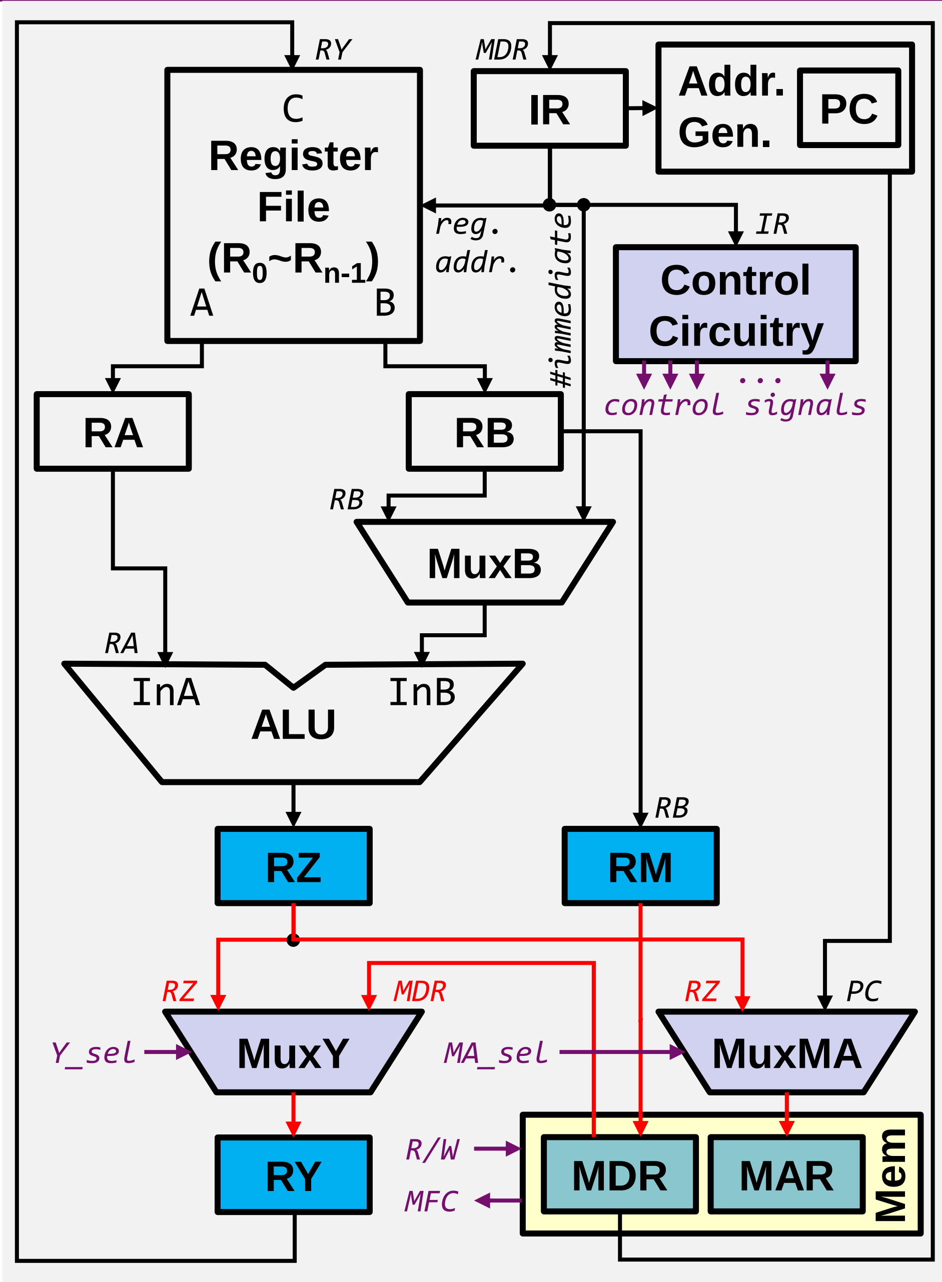
Execution Phase: Stage ③

- The **execution phase** is generally of four stages:
  ③ Perform ALU operation.
  - The input `InA` of **ALU** is supplied by **RA**.
  - The input `InB` of **ALU** is supplied by the multiplexer **MuxB** which forwards:
    - Either the contents of **RB**;
    - Or the immediate value specified in **IR**.
  - **ALU** performs the operation.
  - The computed result is placed in **RZ**.
  - Note: **[RB]** is always forwarded to **RM** (though it's only needed by **`Store`**).



Execution Phase: Stage ④

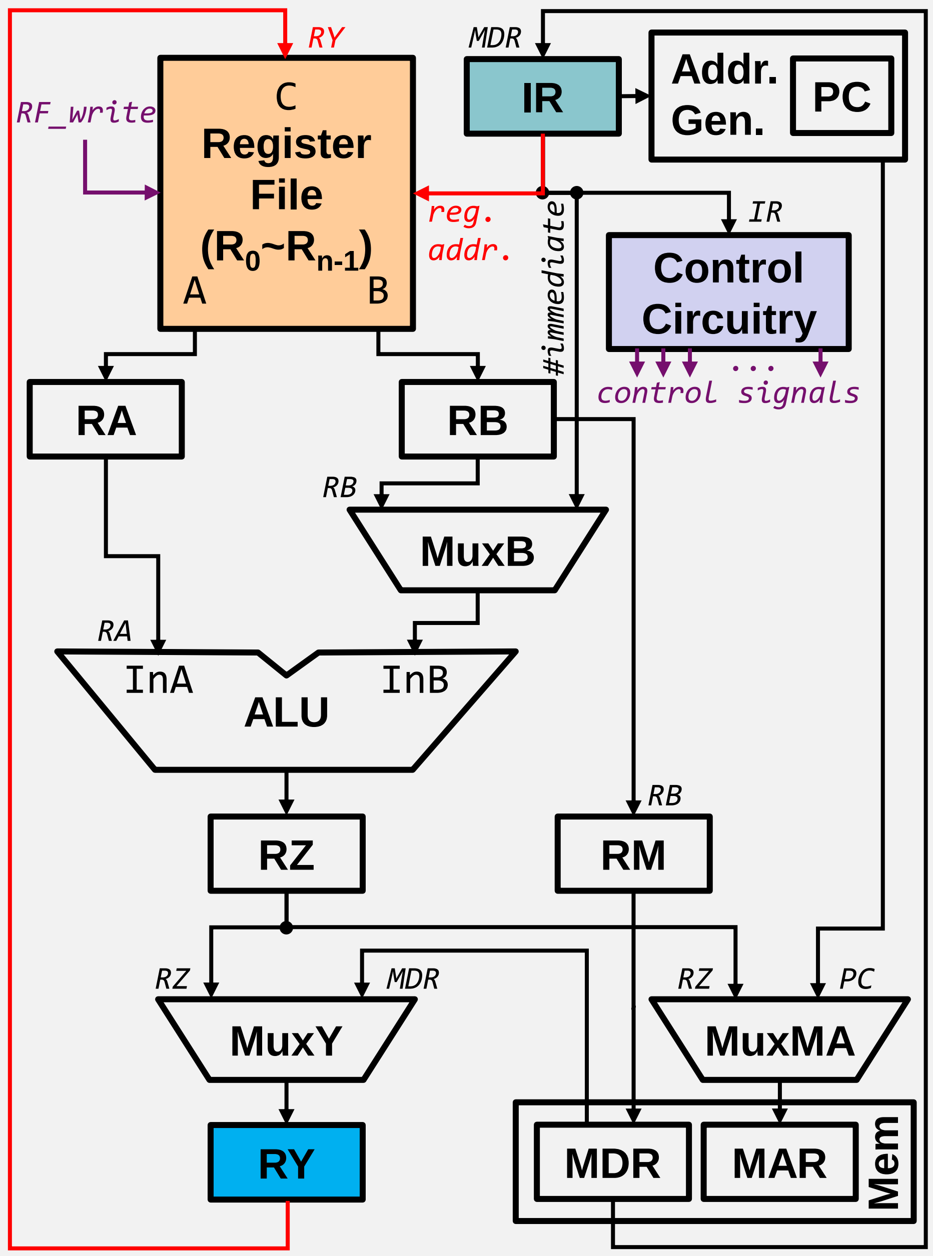
- The **execution phase** is generally of four stages:
  - ④ Read/write memory data.
    - The memory read/write takes place via Processor-Mem Interface.
      - The effective address is derived by **ALU** and kept in **RZ** in Stage ③.
      - The "loaded" data are put into **RY** (with the multiplexer **MuxY** properly set).
      - The "to-be-stored" data are available in **RM**.
    - Note: For non-**Load** and non-**Store** instructions, the data in **RZ** are forwarded to **RY**.



Execution Phase: Stage ⑤

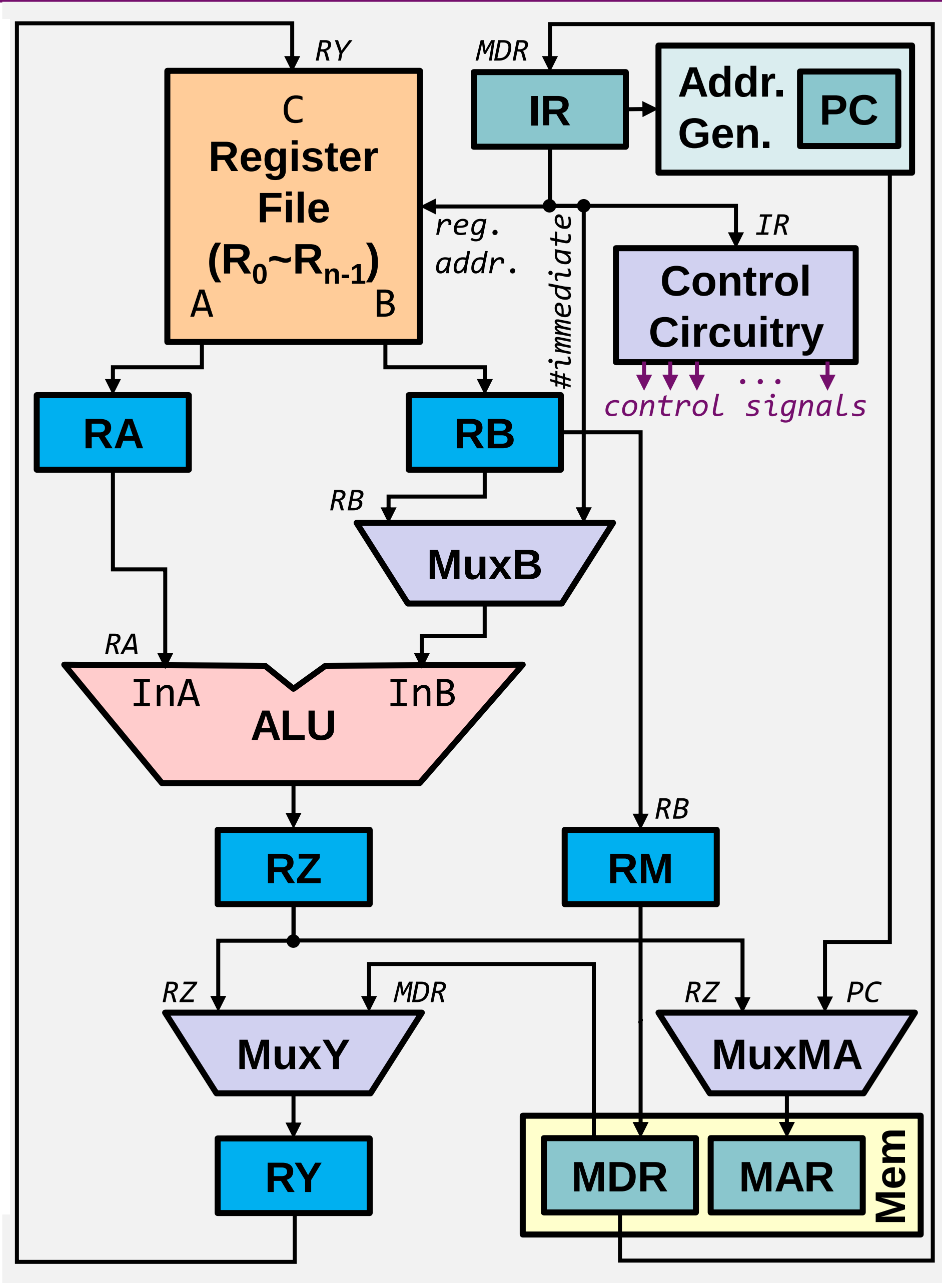
- The **execution phase** is generally of four stages:
  - ⑤ Write the result into the destination register.
    - The data kept in **RY**, which can be:
      - Either the result computed by **ALU** in Stage ③ and forwarded to **RY** in Stage ④;
      - Or the data loaded from the memory in Stage ④.

    are written into **Register File** if needed.
      - The dest. reg. address is from **IR** but is determined by **Control Circuitry.**
      - **RF_write** must be set.



Observations

- The datapath is designed to be independent and versatile.
- But not all actions/stages have to be carried out by every instructions.
- Let's examine the actual execution of the following typical instructions:
  - Add R3, R4, R5
  - Load R5, X(R7)
  - Store R6, X(R8)
  - Branch



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.

$$R2 \leftarrow [LOC]$$

– *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)

Ex 1: Add R3, R4, R5

31 27	26 22	21 17	16 0
Rsrc1	Rsrc2	Rdst	OP code

① MAR ← [PC], Read memory, Wait_MFC, IR ← [MDR], PC ← [PC] + 4 (shown here)

② Decode instruction, RA ← [R4], RB ← [R5]

- The source register addresses are available in **IR**$_{31-27}$ and **IR**$_{26-22}$.
- As a result, **[R4]** and **[R5]** can be read into **RA** and **RB** in Stage ②.

③ RZ ← [RA] + [RB]

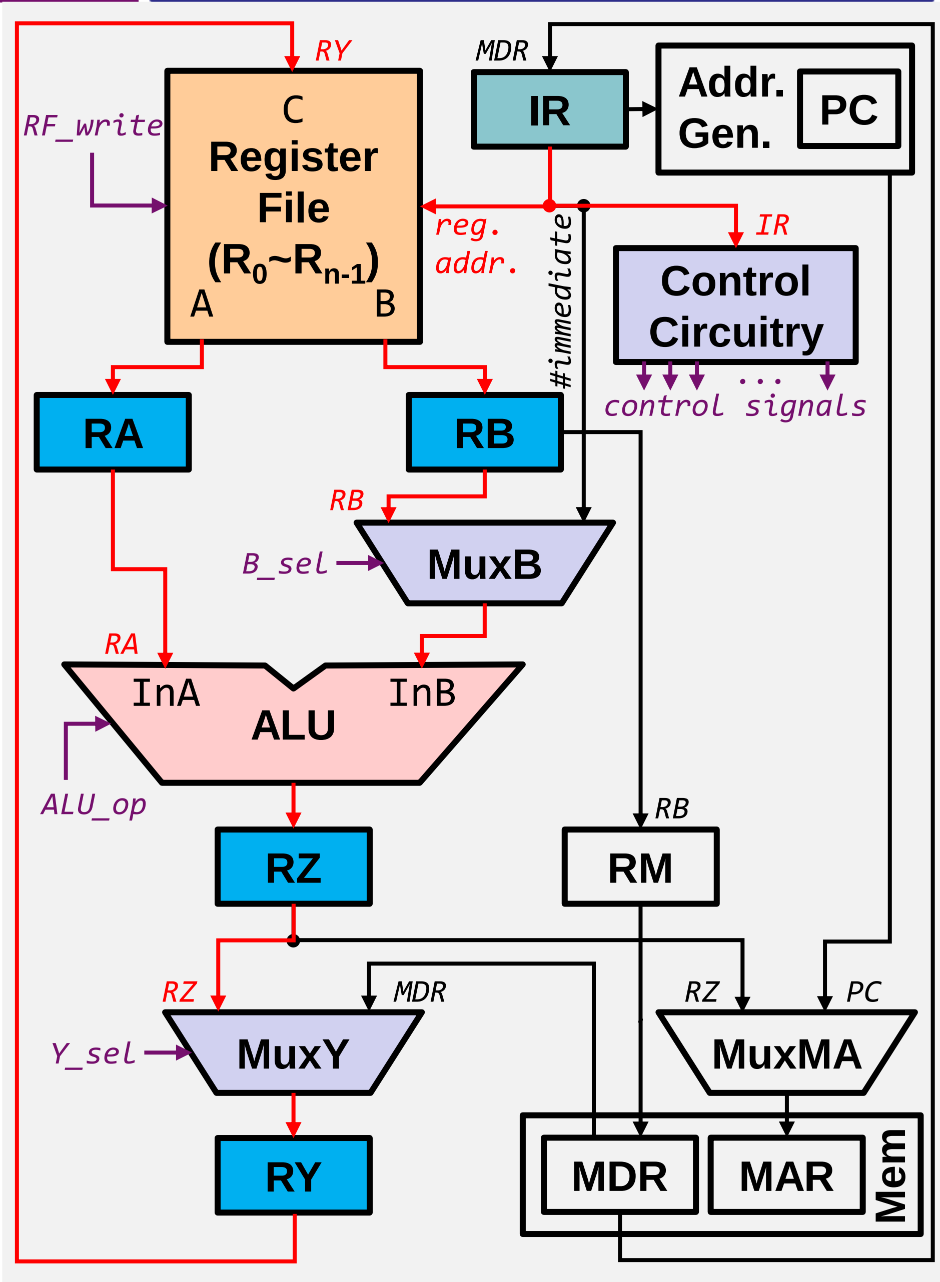
- **MuxB** is set to select input from **RB**.
- **ALU** is set to perform an **Add**.

④ RY ← [RZ]

- **MuxY** is set to select input from **RZ**.

⑤ R3 ← [RY]

- The dest. reg. address is in **IR**$_{21-17}$.
- **RF_write** is set to allow writing **R3**.



Class Exercise 9.1

- Assume **R3**, **R4**, and **R5** originally hold the values **0**, **40**, and **60**, respectively.
- Considering **Add R3, R4, R5**, show the registers' contents "after" the completion of Stages ② to ⑤:
  - *Note: Fill in "?" for those blanks that cannot be determined.*

	RA	RB	RZ	RY	R3
②					
③					
④					
⑤					

Ex 2: Load R5, X(R7)

31 27	26 22	21 6	5 0
Rsrc	Rdst	Immediate operand	OP code

① MAR ← [PC], Read memory, Wait_MFC, IR ← [MDR], PC ← [PC] + 4 (shown here)

② Decode instruction, RA ← [R7]

- The src. reg. address is in $\mathbf{IR_{31-27}}$.

③ RZ ← [RA] + X

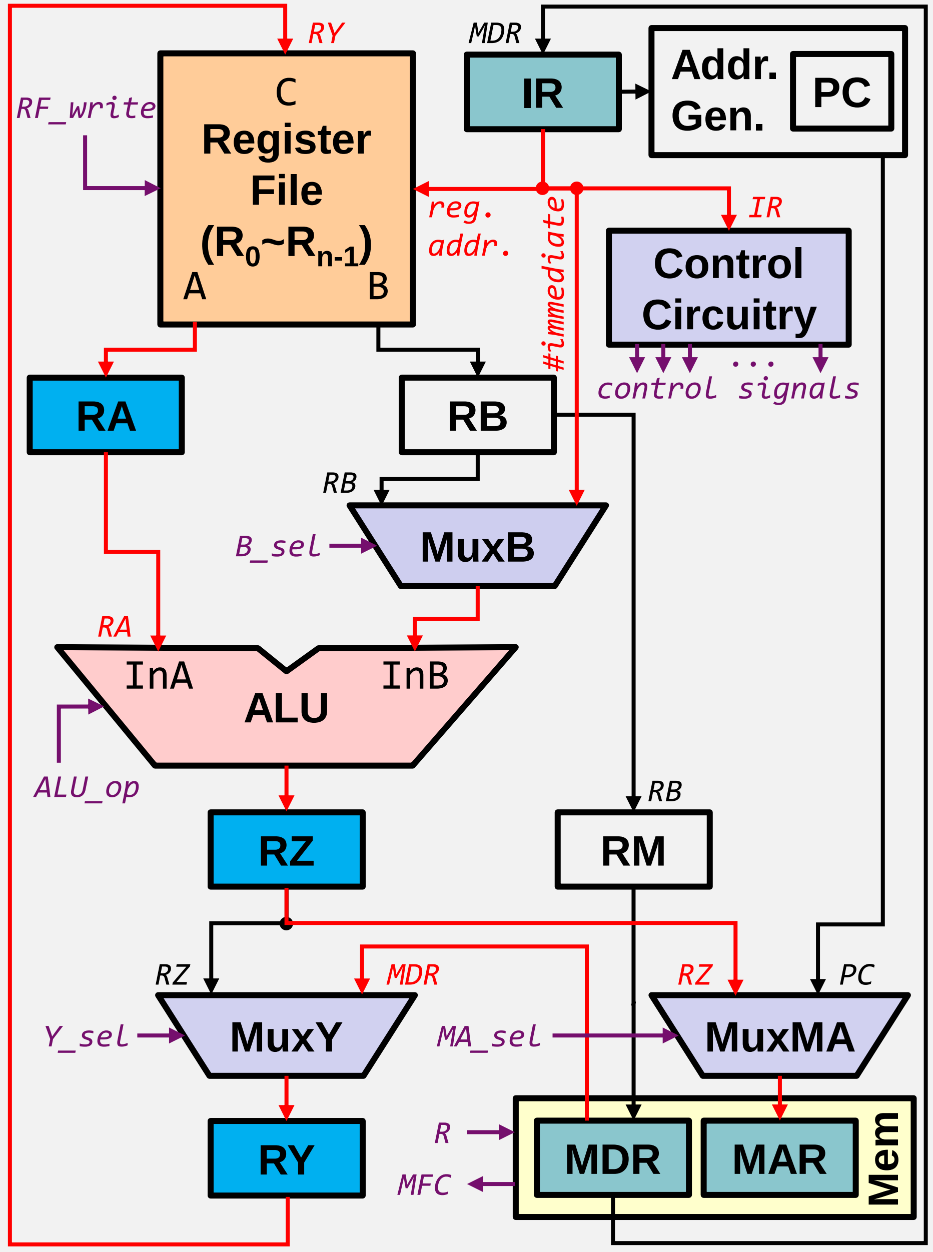
- The immediate value **X** is from **IR**.
- **MuxB** is set to select input from **IR**.
- **ALU** is set to perform an **Add**.

④ MAR ← [RZ], Read memory, Wait_MFC, RY ← [MDR]

- **MuxMA** is set to select input from **RZ**.
- **MuxY** is set to select input from **MDR**.

⑤ R5 ← [RY]

- The dest. reg. address is in $\mathbf{IR_{26-22}}$.
- **RF_write** is set to allow writing **R5**.



Class Exercise 9.2

- Assume **R5** and **R7** originally hold the values **0** and **4**, respectively, and the contents of main memory are shown as follows:

0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	…
9	8	7	6	5	4	3	2	1	0	…

- Considering **Load <u>R5</u>, 4(R7)**, show the registers' contents "after" the completion of Stages ② to ⑤:
  - *Note: Fill in "?" for those blanks that cannot be determined.*

	RA	RB	RZ	RY	MAR	MDR	R5	R7
②								
③								
④								
⑤								

Ex 3: Store R6, X(R8)

31 27	26 22	21 6	5 0
Rsrc	Rdst	Immediate operand	OP code

① MAR ← [PC], Read memory, Wait_MFC, IR ← [MDR], PC ← [PC] + 4 (shown here)

② Decode instruction, RA ← [R8], RB ← [R6]

- The src. reg. address is in **IR**$_{31-27}$.
- The dest. reg. address is in **IR**$_{26-22}$.

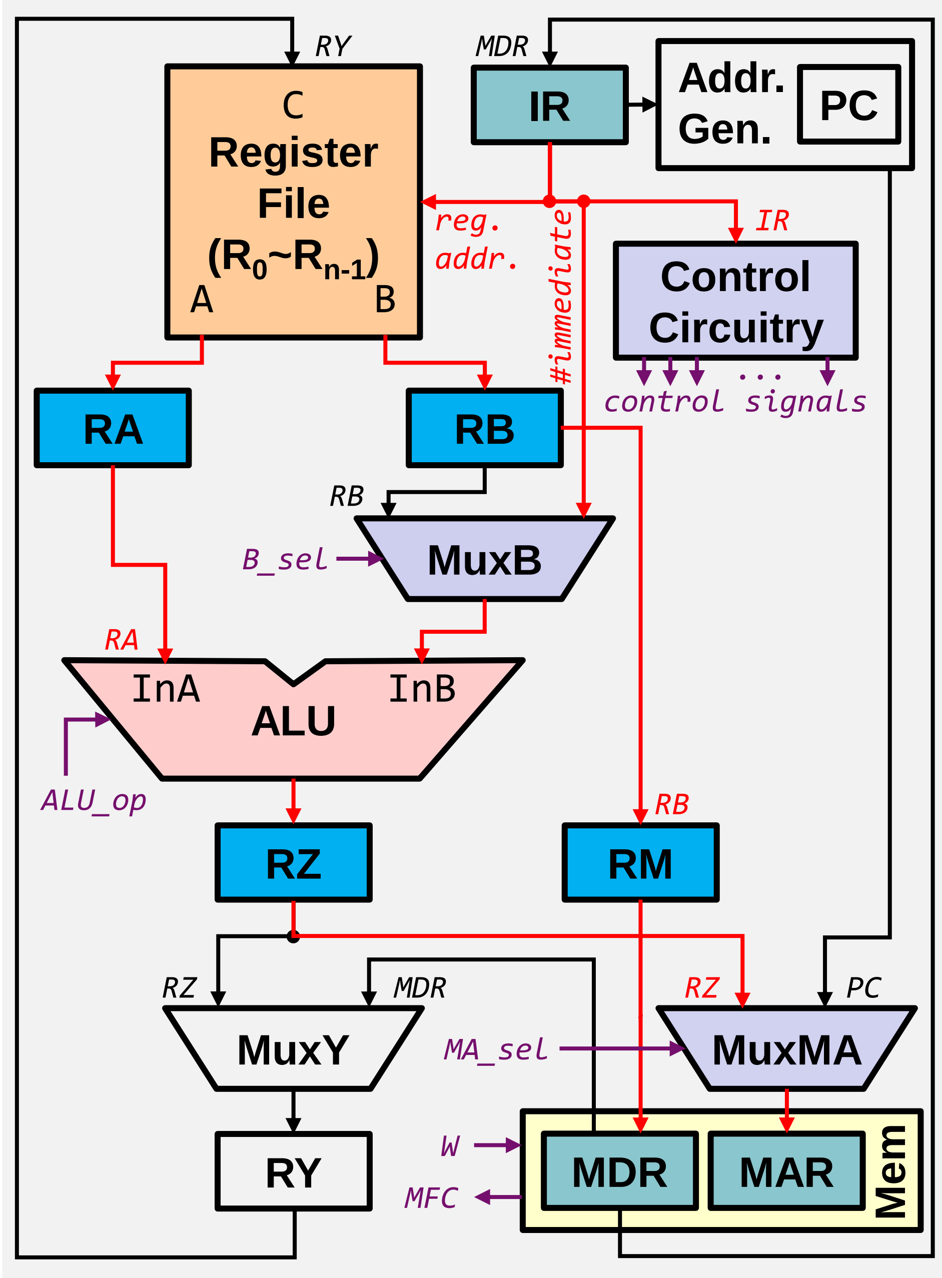
③ RZ ← [RA] + X, RM ← [RB]

- The immediate value **X** is from **IR**.
- **[R6]** is forwarded to ④ via **RM**.

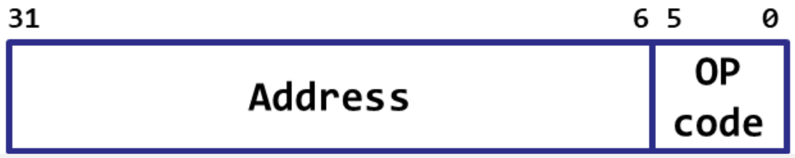
④ MAR ← [RZ], MDR ← [RM], Write memory, Wait_MFC

- **MuxMA** is set to select input from **RZ**.
- The written data are forwarded from **RM** to **MDR**.

⑤ No action



Ex 4: Branch



① MAR ← [PC], Read memory, Wait_MFC, IR ← [MDR], PC ← [PC] + 4 (shown here)

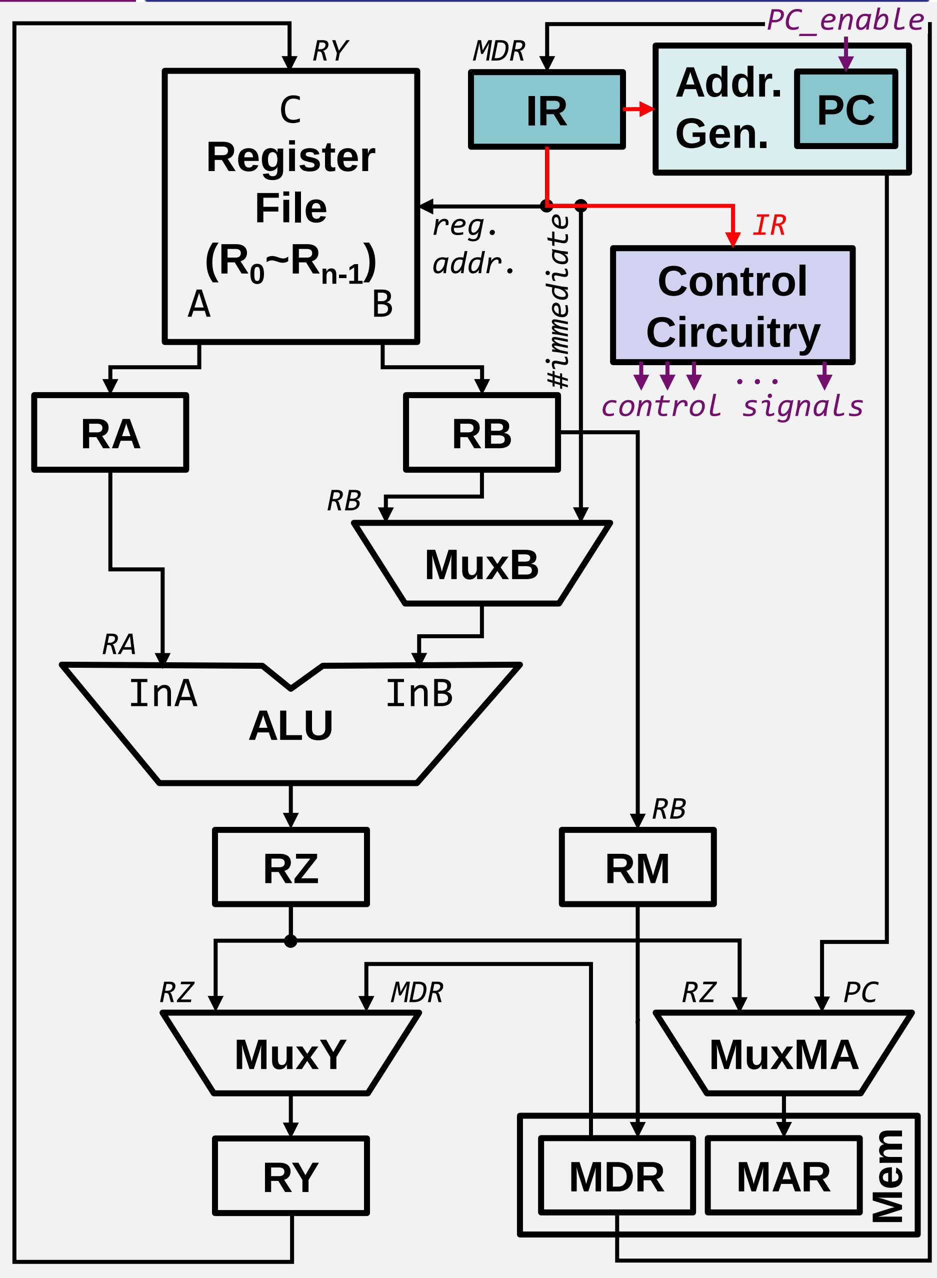
② Decode instruction

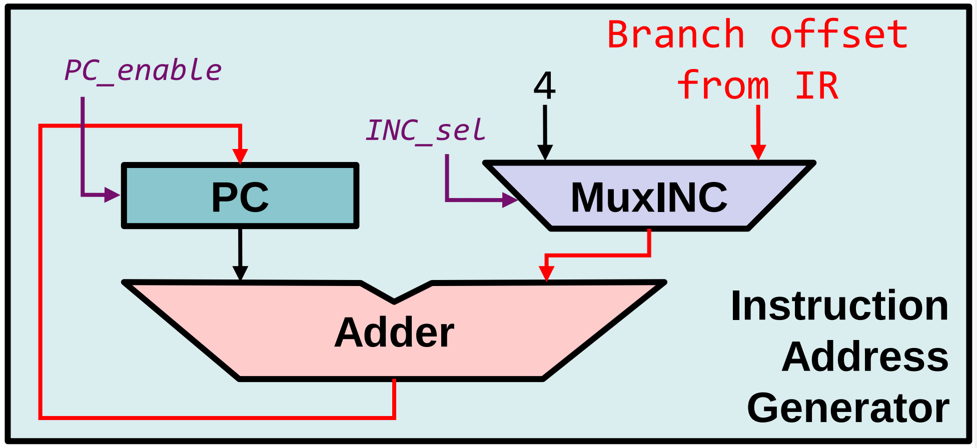
③ PC ← [PC] + branch offset

- The **branch offset** is from **IR**.
- **MuxINC** (in **Instruction Address Generator**) is set to select **offset**.

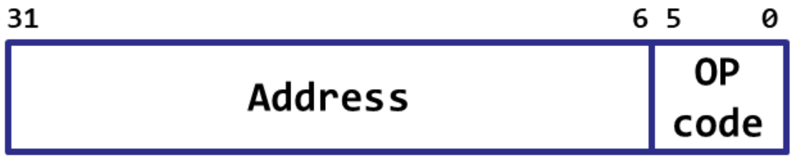
④ No action

⑤ No action





Class Exercise 9.3

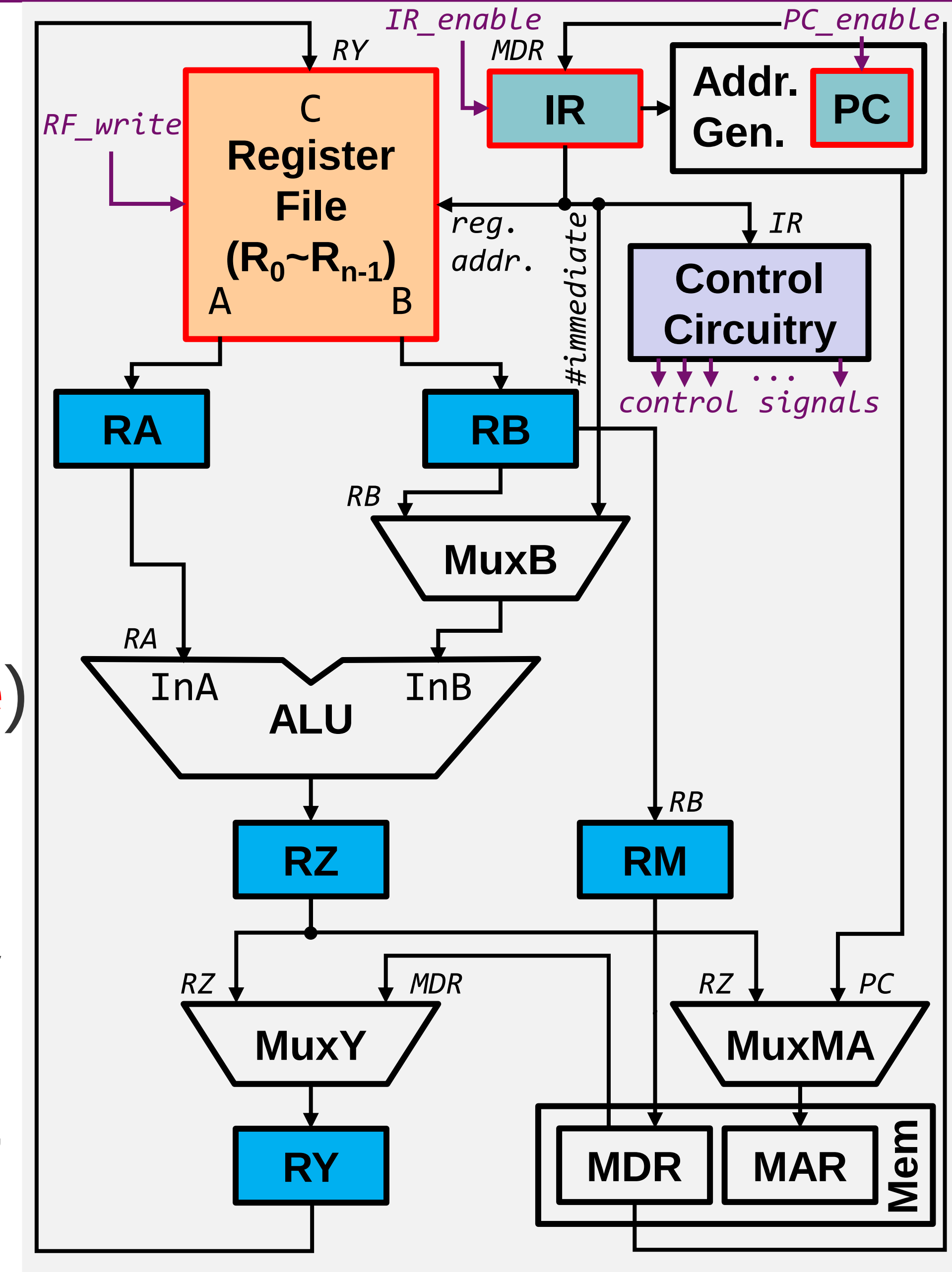


- **Branch** instructions typically use the **address field** to specify an offset from the current instruction to the branch target.
- Given the program, what is the required offset for **Branch LOOP** at the memory address **140**?
  - *Note: This program finds out the smallest number in a list.*

LABEL	ADDR.	OPCODE	OPERAND
	100	Move	R2, addr LIST
	104	Clear	R3
	108	Load	R4, N
	112	Load	R5, (R2)
LOOP:	116	Subtract	R4, R4, #1
	120	Branch_if_[R4]=0	DONE
	124	Add	R3, R3, #4
	128	Load	R6, (R2,R3)
	132	Branch_if_[R6]≥[R5]	LOOP
	136	Move	R5, R6
	140	**Branch**	**LOOP**
DONE:	144	Store	R5, RESULT

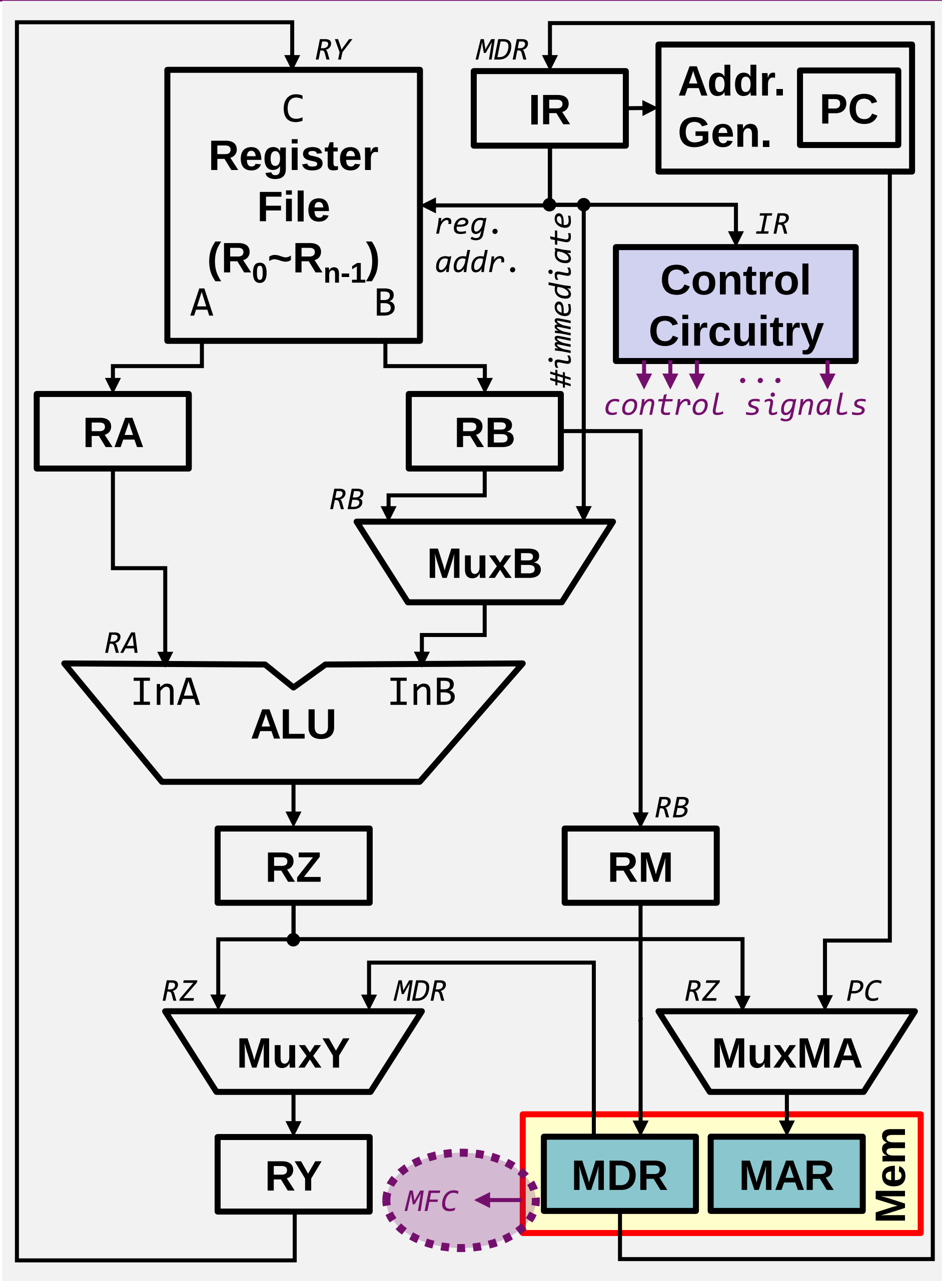
Remark 1: Register Enabling

- **Inter-stage registers** (i.e., **RA**, **RB**, **RZ**, **RM**, & **RY**) are always enabled.
  - The results from one stage are always transferred to the next for simplicity.
- The other registers (e.g., **PC**, **IR**, and **Register File**) must not be changed in every stage.
  - They must be enabled only at certain times.
    - How? By setting **PC_enable**, **IR_enable** and **RF_write**.



Remark 2: Memory Function Completed

- If data are in cache, the stage can be completed in one clock cycle.
- If data are not in cache, the stage may take several clock cycles.
  - To handle such uncertainty:
    - **Processor-Memory Interface** generates the MFC signal upon the completion of a memory operation.
    - **Control Circuitry** checks the MFC signal during any stage involving memory to delay subsequent stage(s).

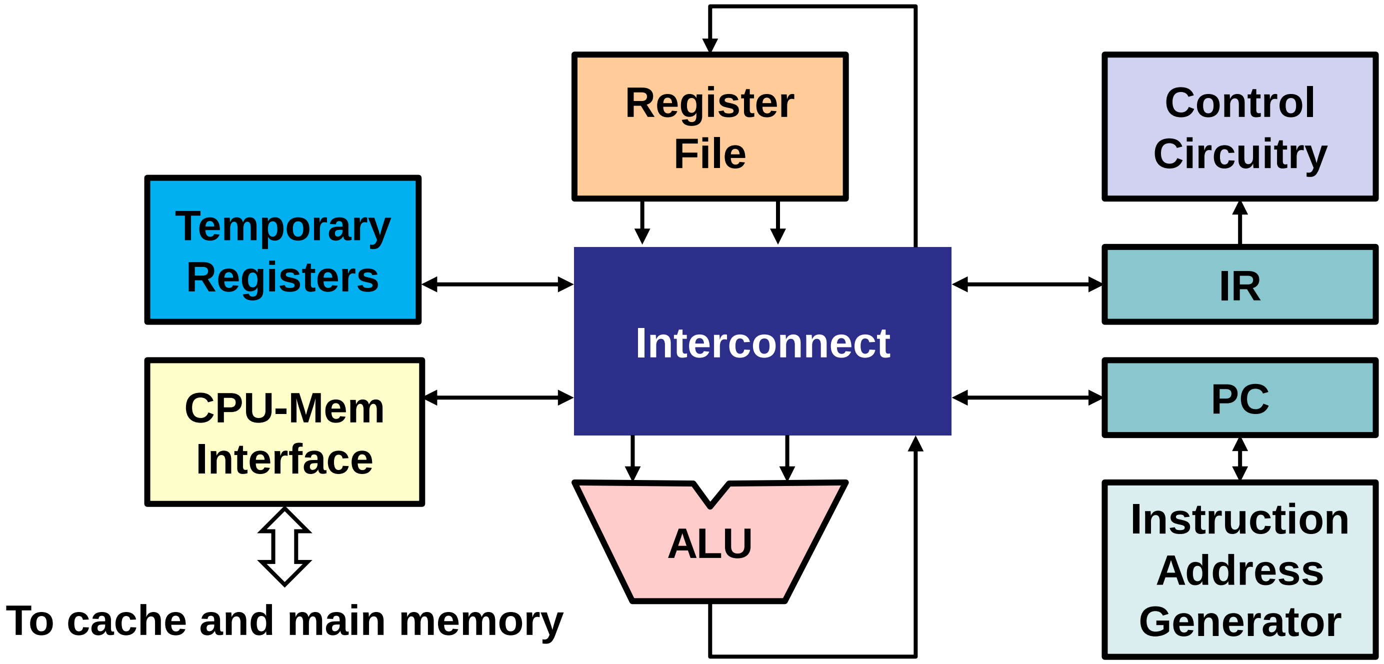


Outline

- Main Components of a Processor
- RISC-Style Processor Design
  - Five-Stage Organization
  - Instruction Execution
- CISC-Style Processor Design
  - Multi-Bus Interconnect
  - Instruction Execution
- Control Signal Generation

Interconnect (CISC CPU)

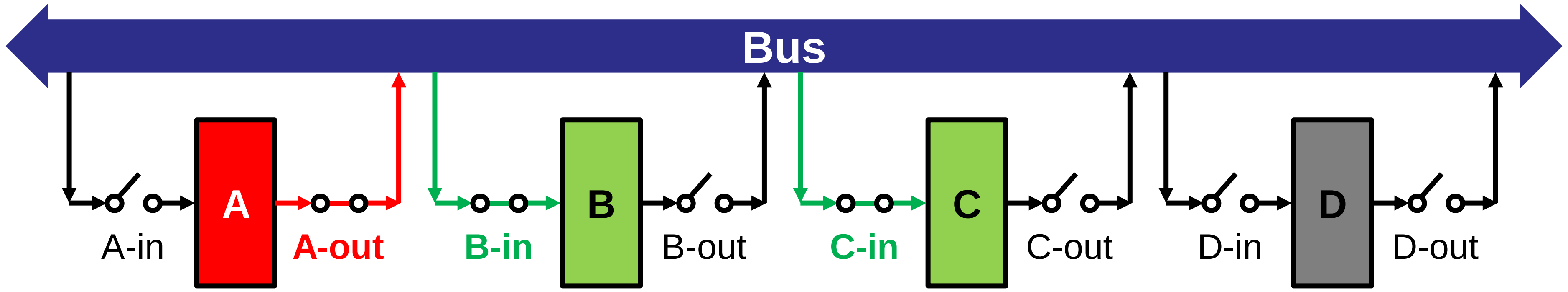
- CISC-style instructions require a different and more flexible organization of the processor hardware:
  - **Interconnect** provides interconnections among other units but does not prescribe any pattern of data flow.
  - Inter-stage registers are not needed, but it is still necessary to have some **Temporary Registers**.



Buses: Key to Transferring Data

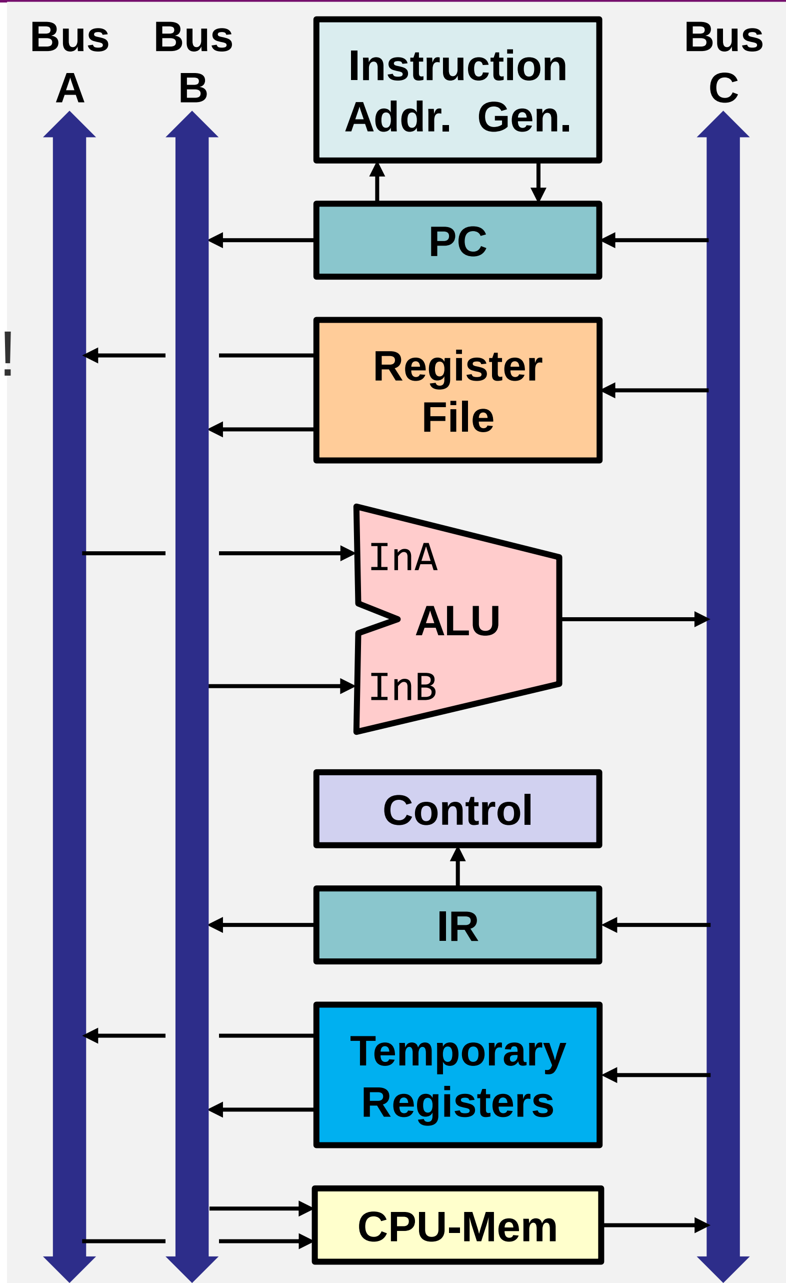
- It is typical to use **buses** to implement **Interconnect**.
  - A bus consists of a set of lines that enable data transferring from any one device to any other (connected to the bus).

- There may be multiple devices connected to the bus:
  - Only one can drive the bus at any given time.
  - More than one can receive data from the bus at the same t.
  - For this reason, **switches** (→⌀ o→) are often needed to allow data to be transferred into or transferred out from a device.



Three-Bus Implementation

- It is common to interconnect the processor hardware via three buses:
  - Why 3? Typical instruction format!
  - **Bus A** and **Bus B** allow the data transfer of two source operands to **ALU** simultaneously.
  - **Bus C** allows transferring the result (computed by **ALU**) to the destination operand.
  - *Note: Addresses for the three ports of **Register File** are generated by **Control Circuity** (not shown in the figure).*



Outline

- Main Components of a Processor
- RISC-Style Processor Design
  - Five-Stage Organization
  - Instruction Execution
- CISC-Style Processor Design
  - Multi-Bus Interconnect
  - Instruction Execution
- Control Signal Generation

Instruction Fetching and Decoding

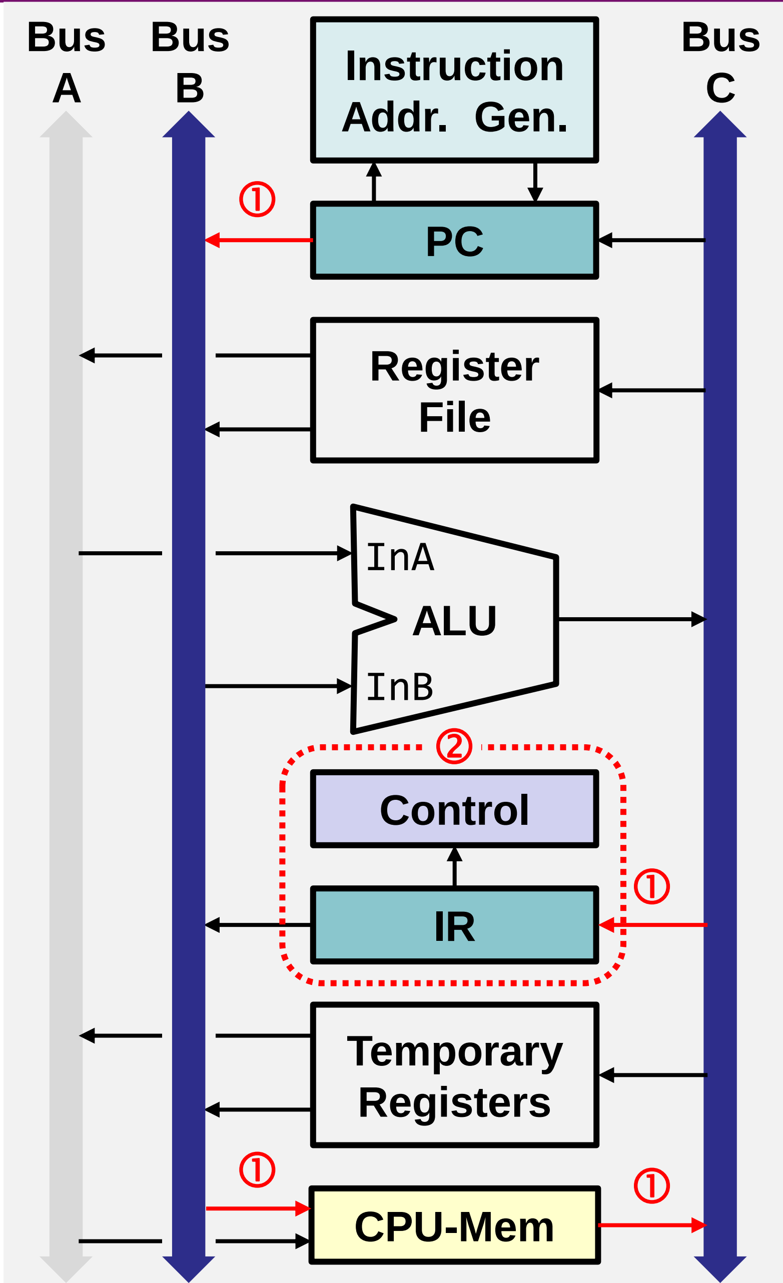
- All the instructions share the same actions of **fetching** and **decoding**:

① `MAR ← [PC], Read memory, Wait_MFC, IR ← [MDR], PC ← [PC] + 4`

- **`Bus B`** is used to send **`[PC]`** to **`CPU-Memory Interface`** to fetch the instruction.
- The data read from the memory are sent to **`IR`** over **`Bus C`**.
- Note: **`Wait_MFC`** is needed since the data may be read from the main memory.

② `Decode instruction`

- **`[IR]`** is decoded by **`Control Circuity`** to generate control signals (not shown).



Ex 1: Add R5, R6

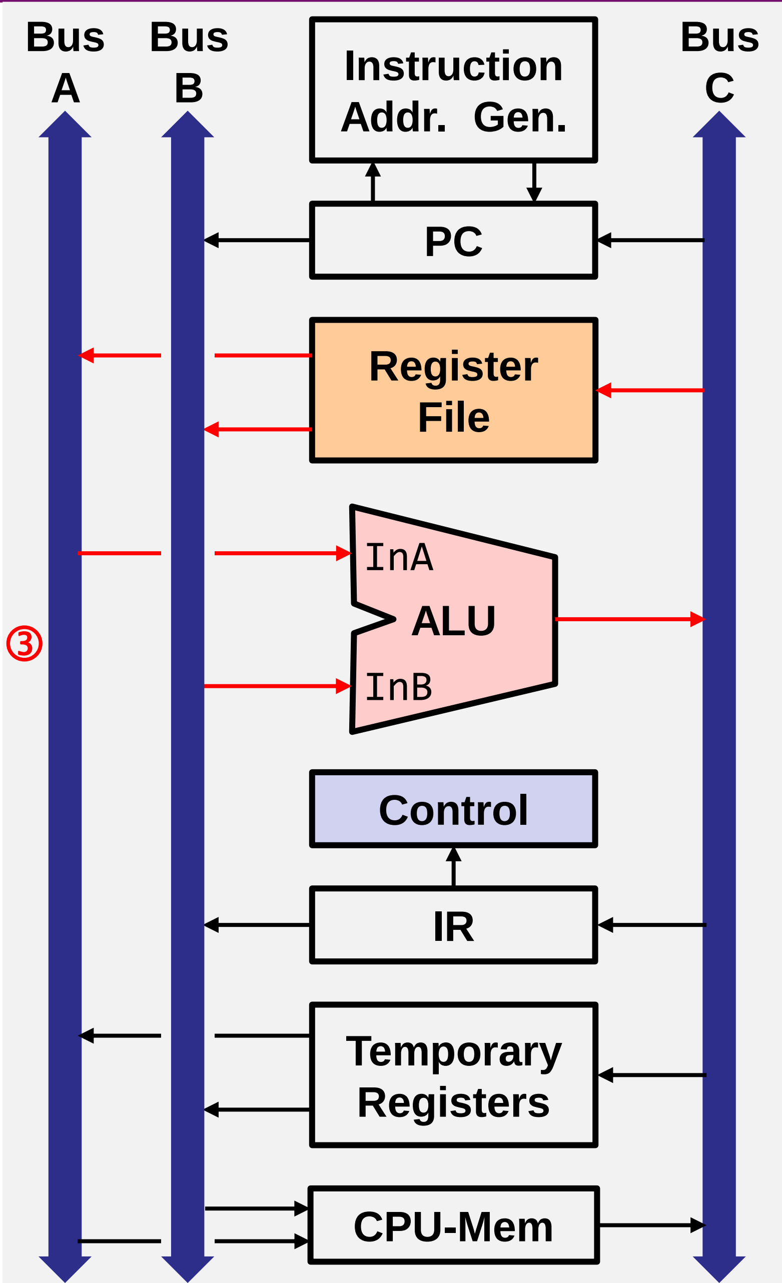
① `MAR ← [PC], Read memory, Wait_MFC, IR ← [MDR], PC ← [PC] + 4` (shown here)

② `Decode instruction` (shown here)

③ `R5 ← [R5] + [R6]`

- **`[R5]`** is read from **`Register File`** and transferred to **`InA`** of **`ALU`** via **`Bus A`**.
- **`[R6]`** is read from **`Register File`** and transferred to **`InB`** of **`ALU`** via **`Bus B`**.
- **`ALU`** is set to perform an **`Add`**.
- The sum is stored back into **`R5`** via **`Bus C`**.

- *Note 1: Reading source registers cannot proceed in parallel with the decoding, since CISC-style instructions do not always use the same fields to specify reg. addresses.*
- *Note 2: Control Circuitry must carefully coordinate how to read and write Register File within the same step.*



Ex 2: And X(R7), R9

- Assume the index offset is a 32-bit value given as the second word of the instruction.

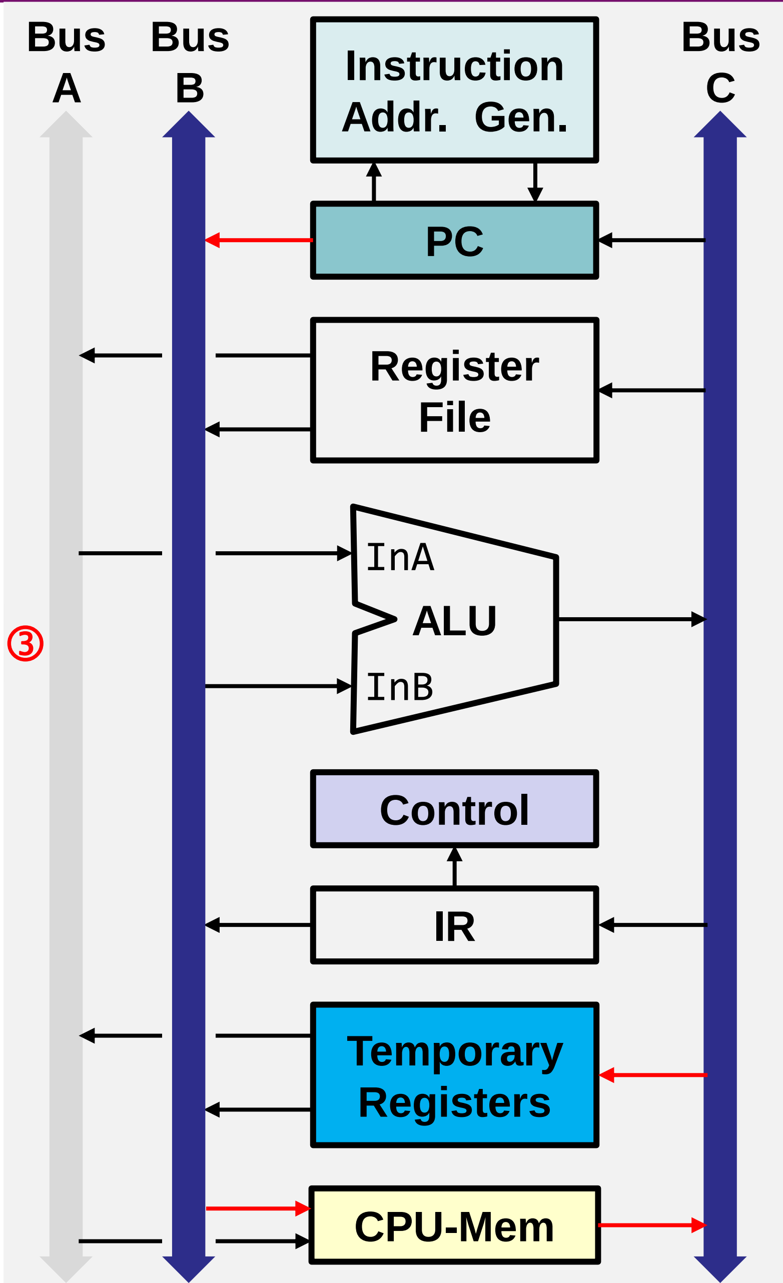
31			0
OP code	Source	Dest.	Other
Index Offset (X)			

① MAR ← [PC], Read memory, Wait_MFC, IR ← [MDR], PC ← [PC] + 4 (shown here)

② Decode instruction (shown here)

③ MAR ← [PC], Read memory, Wait_MFC, Tmp1 ← [MDR], PC ← [PC] + 4

- The second word (i.e., **X**) is fetched from memory into the temporary register Tmp1.



Ex 2: And X(R7), R9 (Cont'd)

④ `Tmp2 ← [Tmp1] + [R7]`

- **`[Tmp1]`** and **`[R7]`** are sent to **`ALU`** over **`Buses A`** and **`B`**, and the effective address is placed into the temporary register `Tmp2`.

⑤ `MAR ← [Tmp2], Read memory, Wait_MFC, Tmp1 ← [MDR]`

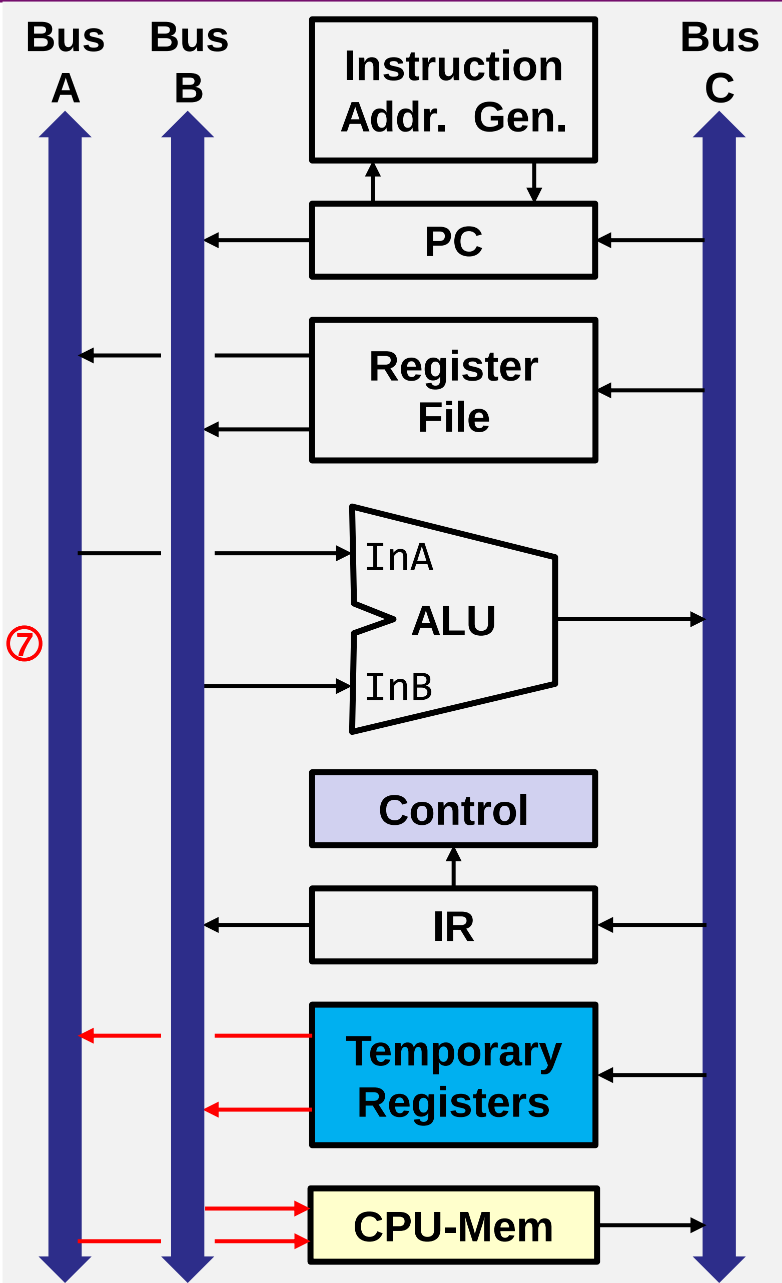
- The contents of memory address **`X(R7)`** are read and placed into `Tmp1`.

⑥ `Tmp1 ← [Tmp1] AND [R9]`

- The **`AND`** computation is performed, and the result is placed into `Tmp1`.

⑦ `MAR ← [Tmp2], MDR ← [Tmp1], Write memory, Wait_MFC`

- The result is stored into the memory at the address **`X(R7)`**, which is "still" available in `Tmp2` since the completion of ④.



Class Exercise 9.4

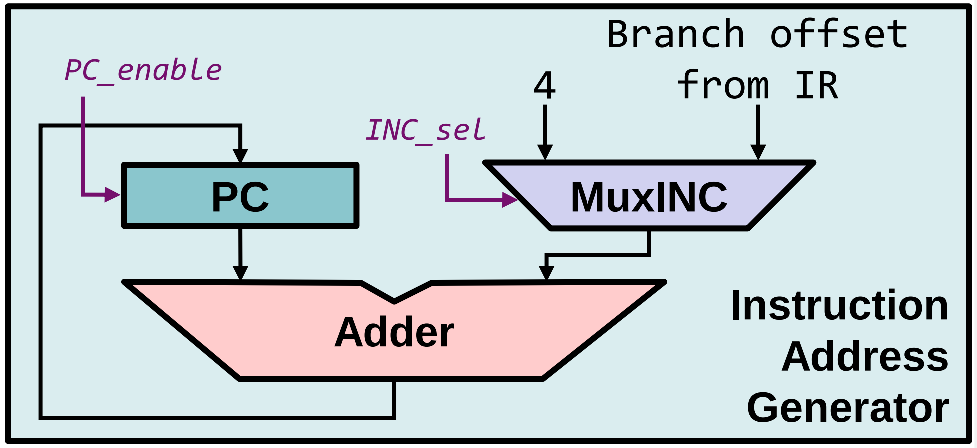
- Consider the three-bus implementation of a CISC-style processor design.
- How many times of memory accesses are involved in **Add R5, R6** and **And X(R7), R9**, respectively?

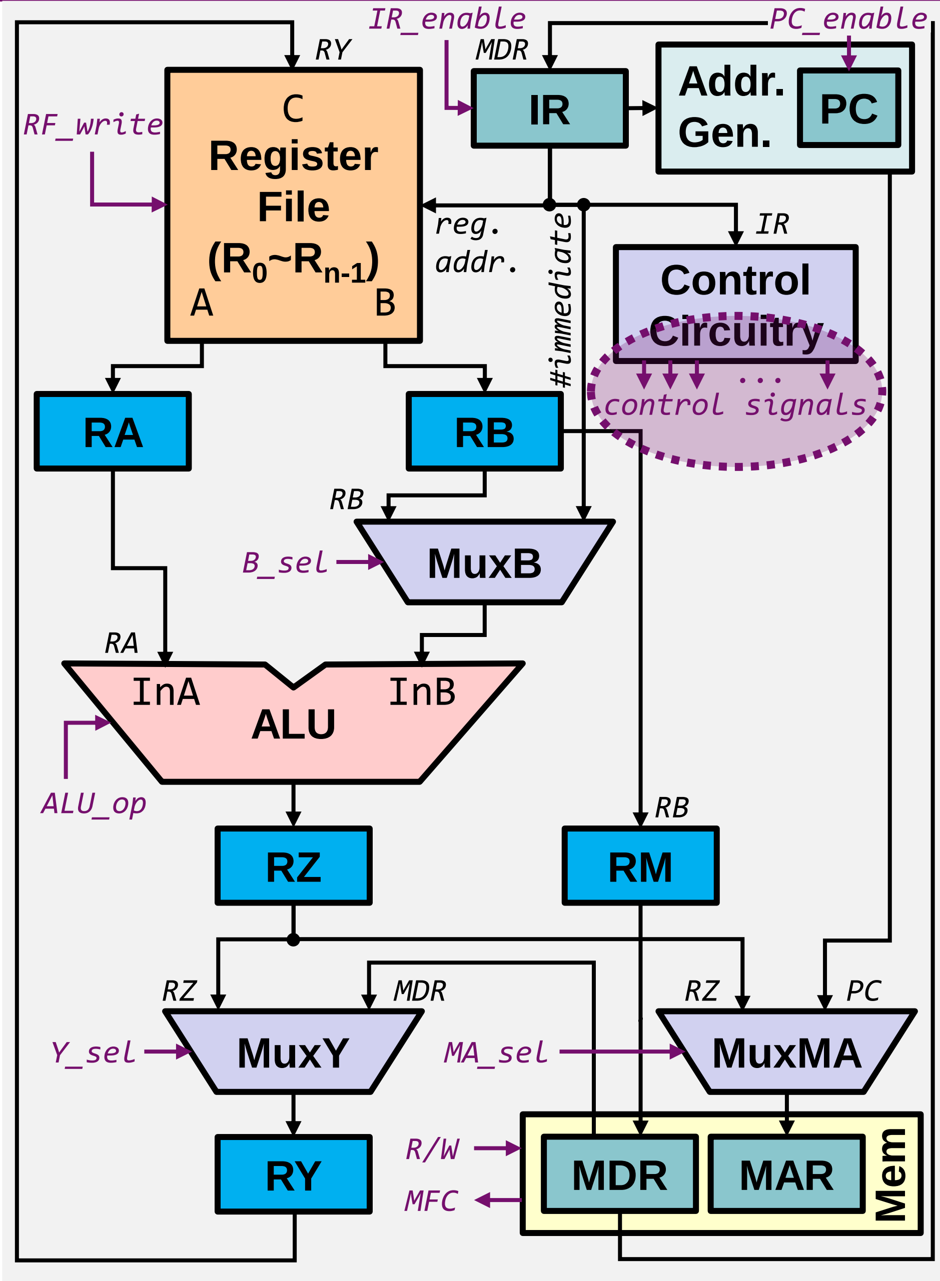
Outline

- Main Components of a Processor
- RISC-Style Processor Design
  - Five-Stage Organization
  - Instruction Execution
- CISC-Style Processor Design
  - Multi-Bus Interconnect
  - Instruction Execution
- Control Signal Generation

Recall: Control Signals

- The processor's hardware is governed by control signals that determine:
  - What is the input data appearing at a **Multiplexer**'s output?
  - What will be the operation performed by **ALU**?
  - Will the data be loaded into the selected **register**?
  - Etc.



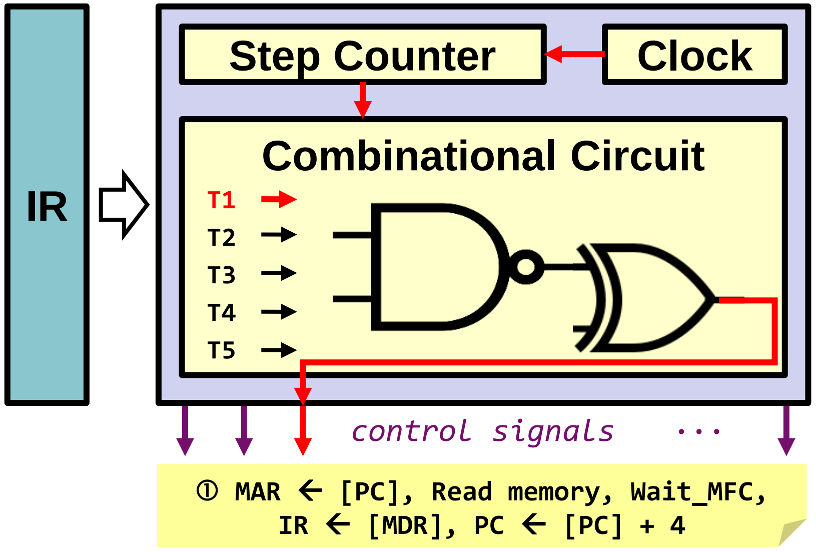


Control Signal Generation

- There're two typical ways to generate control signals:

1) Hardwired Control:

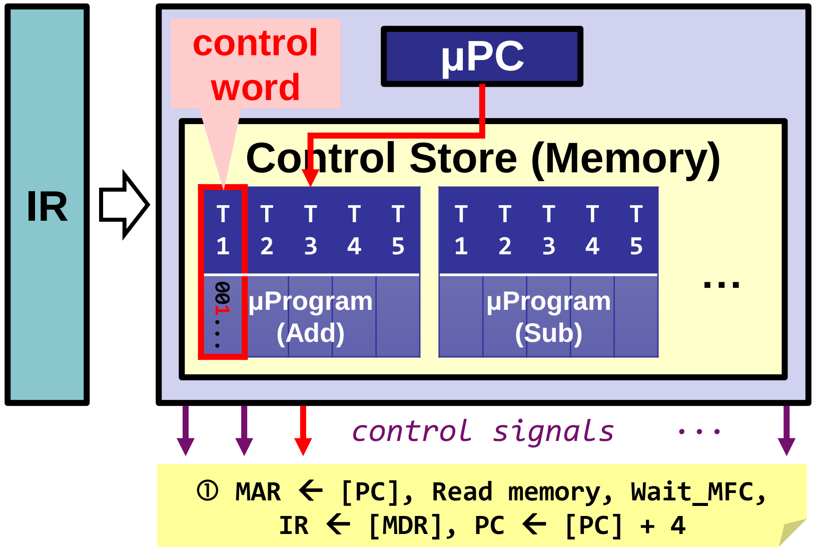
- The **combinational circuit** is used to "**hard code**" the generation of control signals with **logic gates**.
  - The **clock** and **counter** specify the current step (e.g., T1).
- It is a "hardware approach" and can operate at high speed.



① MAR ← [PC], Read memory, Wait_MFC, IR ← [MDR], PC ← [PC] + 4

2) Microprogrammed Control:

- Control signals are specified by "**micro-programs**" in **Control Store**.
  - The **micro-Program Counter** (μPC) always points to the next **control word** (or **μ-instruction**).
- It is a "software approach" and can support a complex instruction set.



① MAR ← [PC], Read memory, Wait_MFC, IR ← [MDR], PC ← [PC] + 4

Summary

- Main Components of a Processor
- RISC-Style Processor Design
  - Five-Stage Organization
  - Instruction Execution
- CISC-Style Processor Design
  - Multi-Bus Interconnect
  - Instruction Execution
- Control Signal Generation