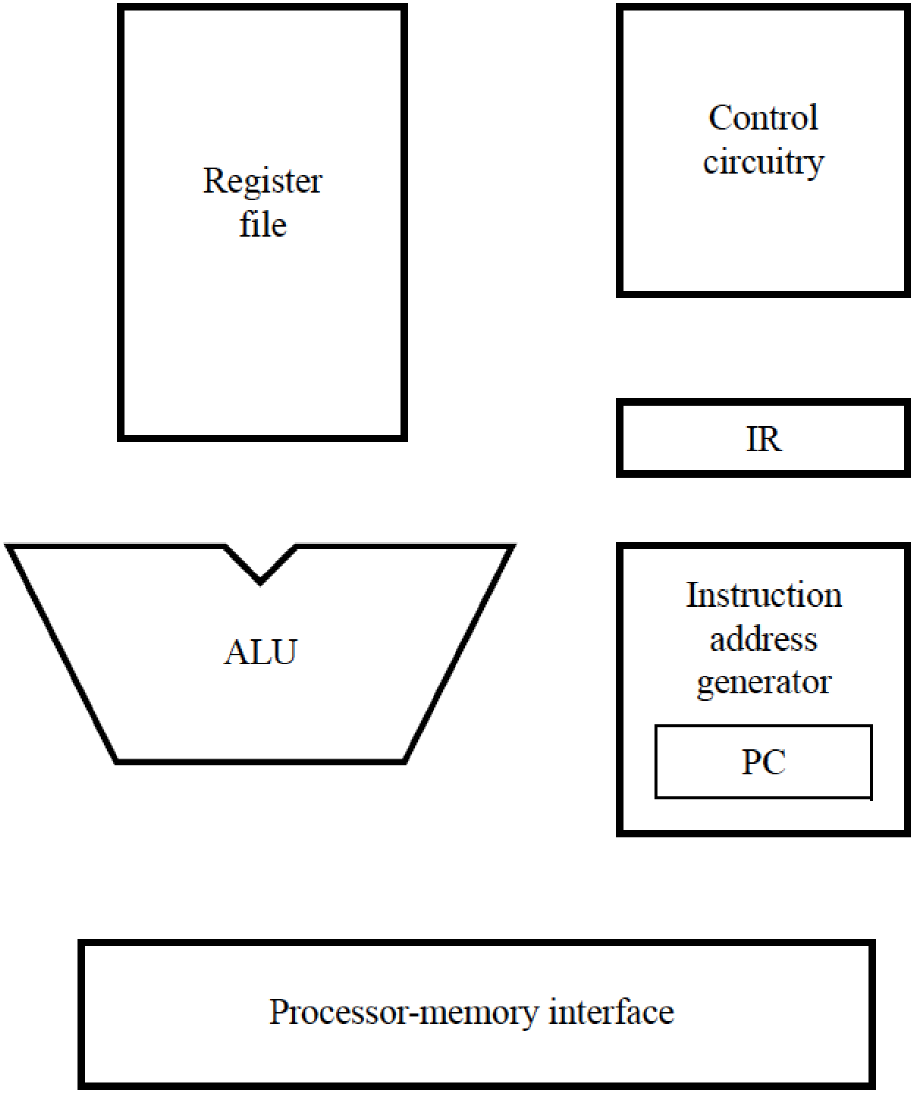
Chapter 5
Basic Processing Unit

Processing Unit

- A processor is responsible for reading program instructions from the computer’s memory and executing them.
- It fetches one instruction at a time.
- It decodes (interprets) the instruction.
- Then, it carries out the actions specified.

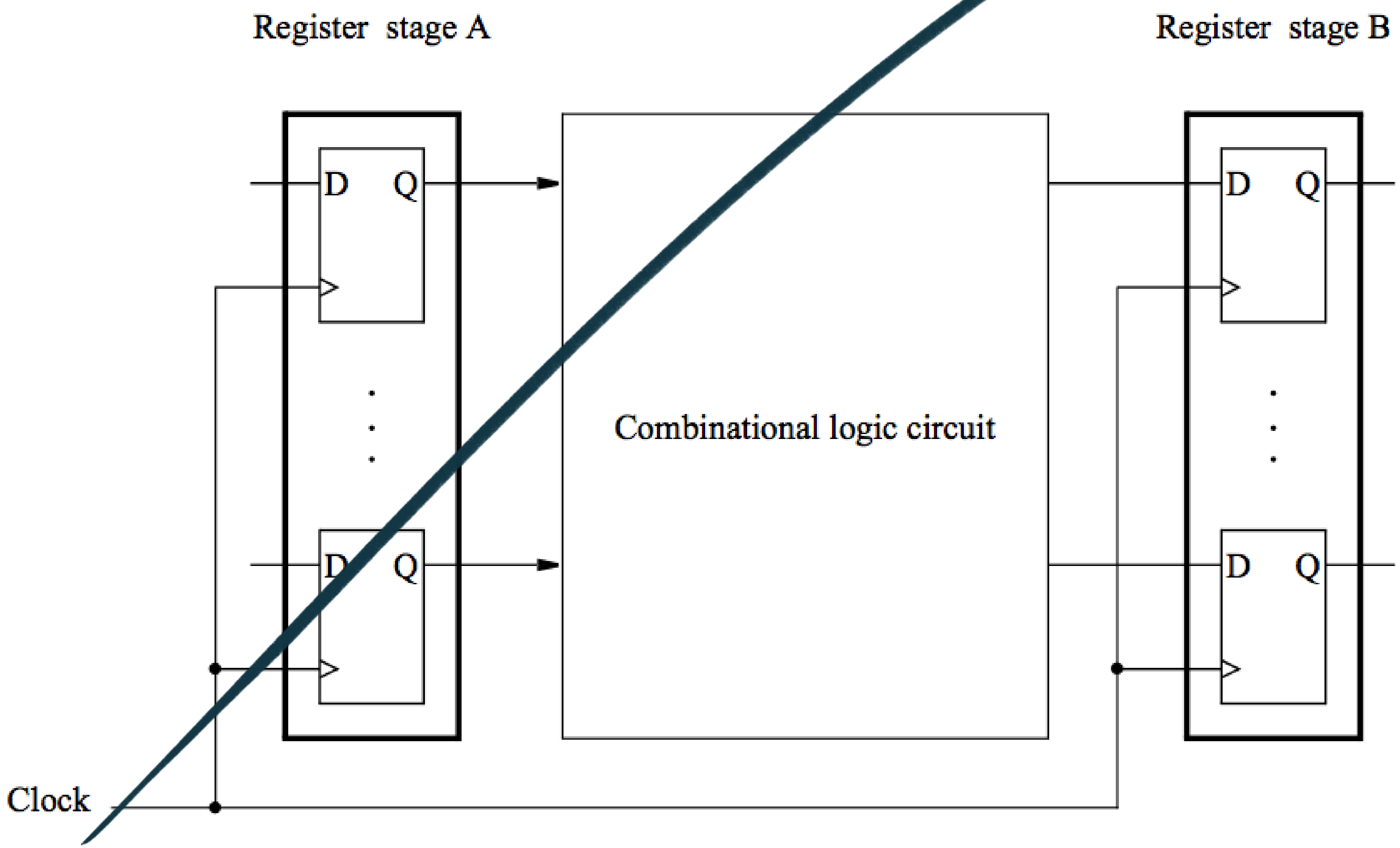
Processor’s building blocks

- PC provides instruction address.
- Instruction is fetched into IR
- Instruction address generator updates PC
- Control circuitry interprets instruction and generates control signals to perform the actions needed.

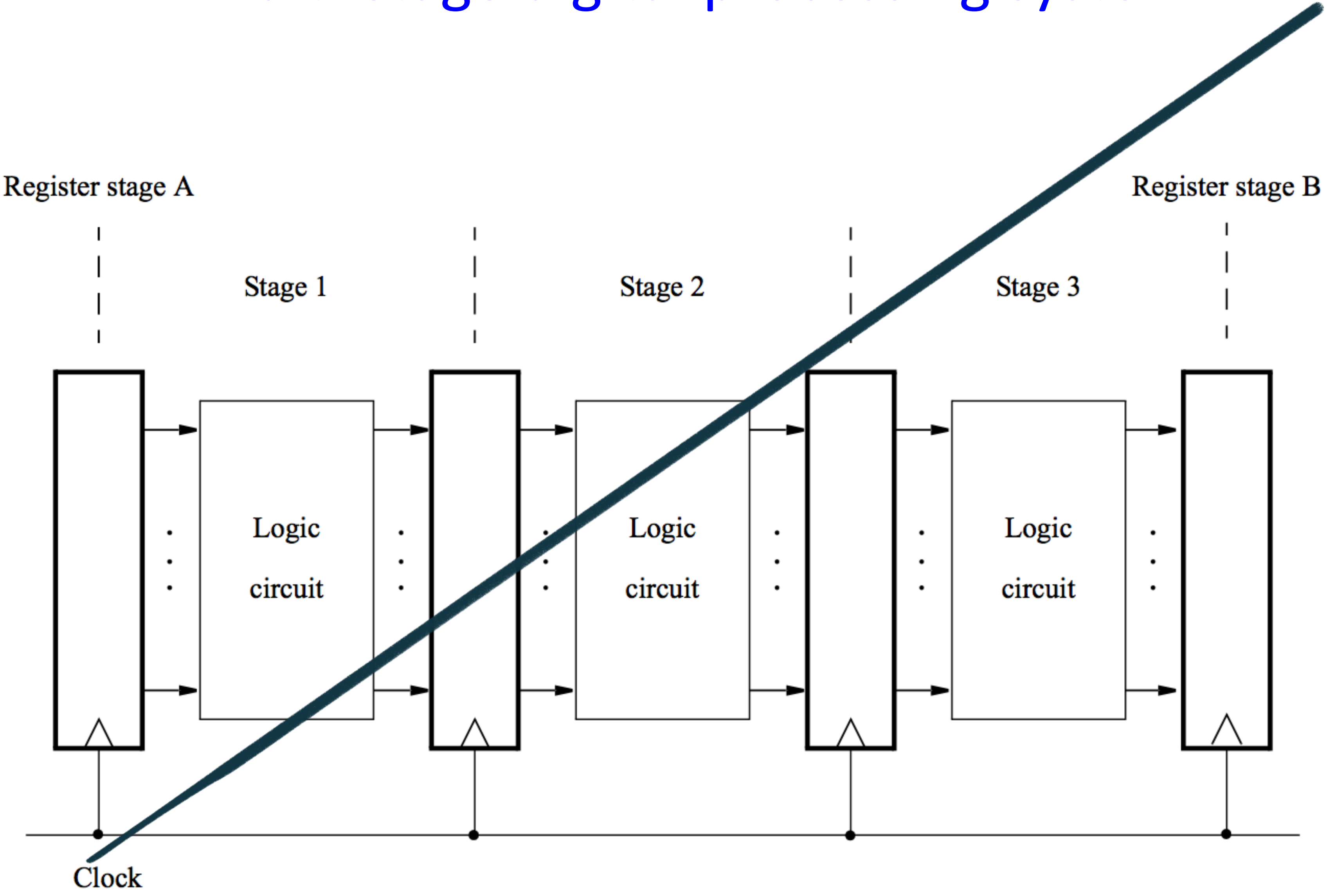


A digital processing system

Contents of register A are processed and deposited in register B.



A multi-stage digital processing system



Why multi-stage?

- Processing moves from one stage to the next in each clock cycle.
- Such a multi-stage system is known as a pipeline.
- High-performance processors have a pipelined organization.
- Pipelining enables the execution of successive instructions to be overlapped.
- Pipelining will be discussed later.

Instruction execution

- Pipelined organization is most effective if all instructions can be executed in the same number of steps.
- Each step is carried out in a separate hardware stage.
- Processor design will be illustrated using five hardware stages.
- How can instruction execution be divided into five steps?

Memory access instructions

Load R5, X(R7)

1. Fetch the instruction and increment the program counter.
2. Decode the instruction and read the contents of register R7.
3. Compute the effective address X+[R7]
4. Read the memory source operand from the memory location X+[R7].
5. Load the operand into the destination register R5.

Store R6, X(R8)

1. Fetch the instruction and increment the program counter.
2. Decode the instruction and read registers R6 and R8.
3. Compute the effective address X+[R8].
4. Store the content of register R6 into the memory location X+[R8].
5. No action

Computational instruction

Add R3, R4, R5

1. Fetch the instruction and increment the program counter.
2. Decode the instruction and read registers R4 and R5.
3. Compute the sum [R4] + [R5].
4. No action.
5. Load the result into the destination register, R3.

- *Stage 4 (memory access) is not involved in this instruction.*

Add R3, R4, #100

1. Fetch the instruction and increment the program counter.
2. Decode the instruction and read register R4.
3. Compute the sum [R4] + 100.
4. No action.
5. Load the result into the destination register, R3.

- *Stage 4 (memory access) is not involved in this instruction.*

Summary – Actions to implement an instruction

1. Fetch an instruction and increment the program counter.

2. Decode the instruction and read registers from the register file.

Fetch → Decode → ALU → memory access → Destination

3. Perform an ALU operation.

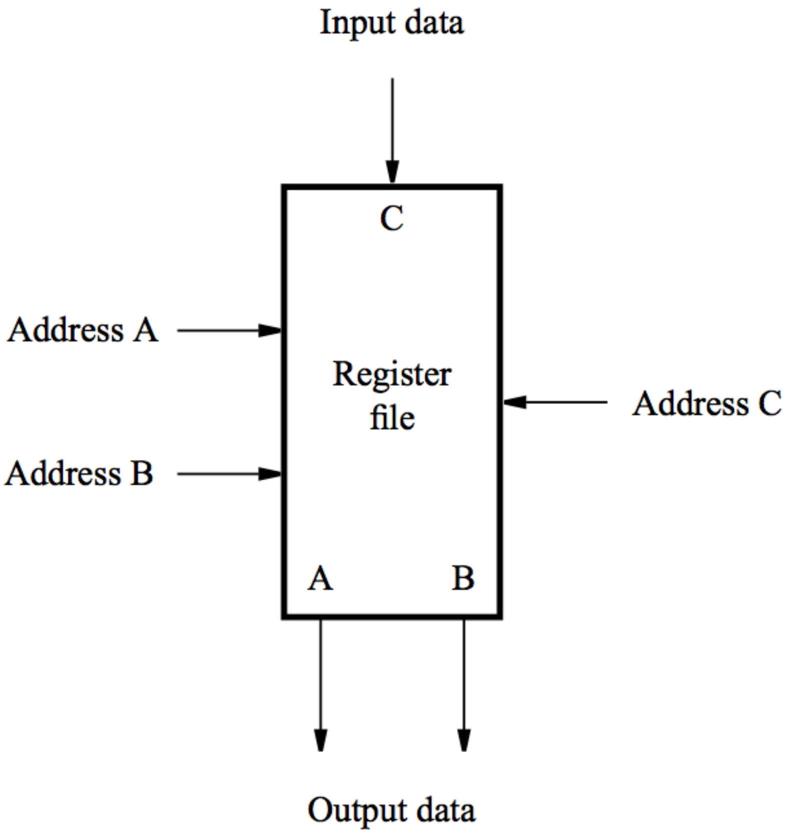
4. Read or write memory data if the instruction involves a memory operand.

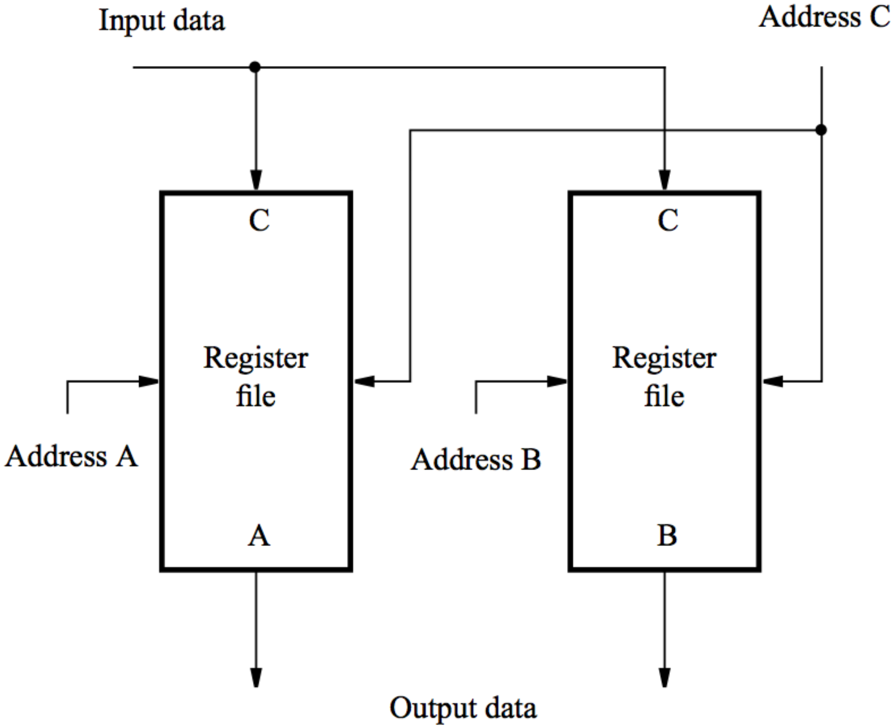
5. Write the result into the destination register.

- This sequence determines the hardware stages needed.

Hardware Components: Register file

- A 2-port register file is needed to read the two source registers at the same time.
  - May be implemented using a 2-port memory.
  - Or using two single-ported memory blocks.



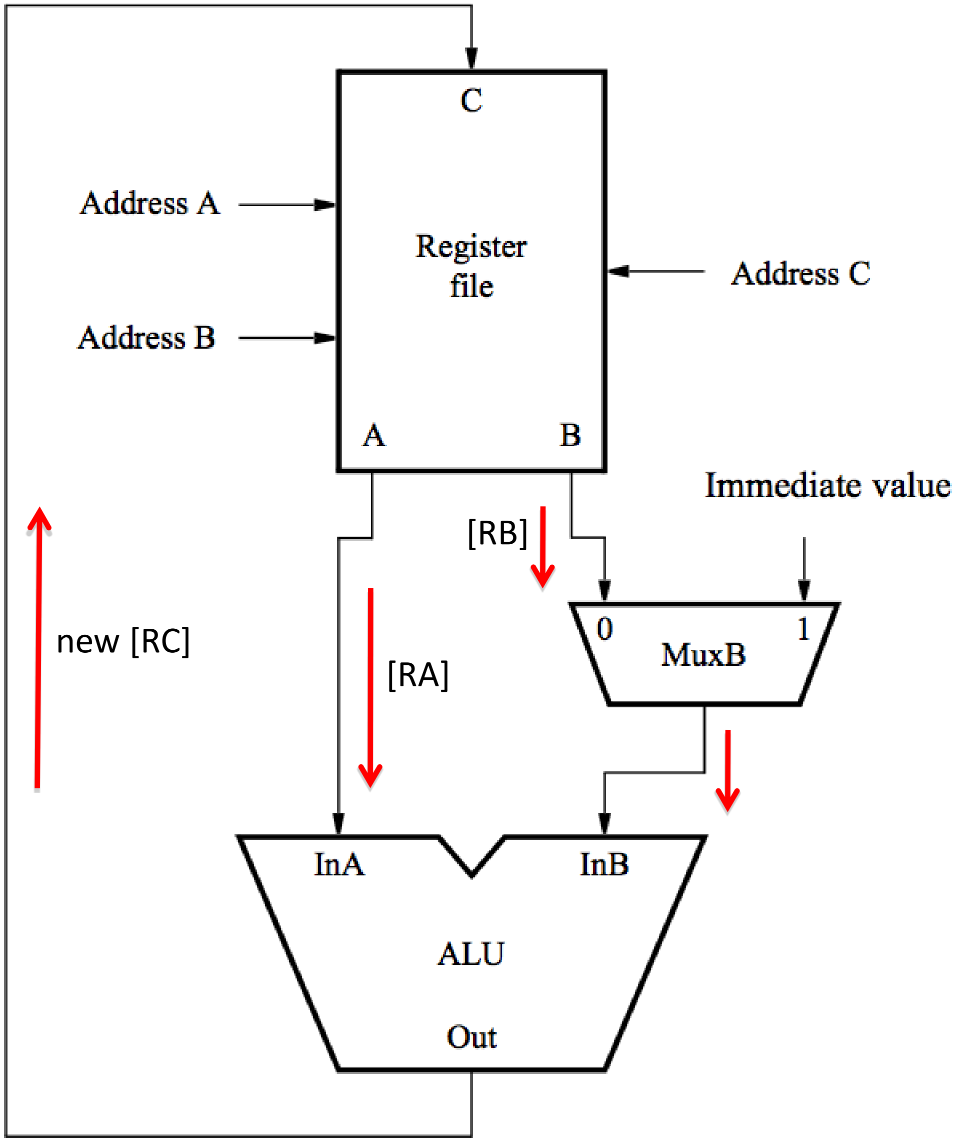


A conceptual view – computational instructions

- Both source operands and the destination location are in the register file.

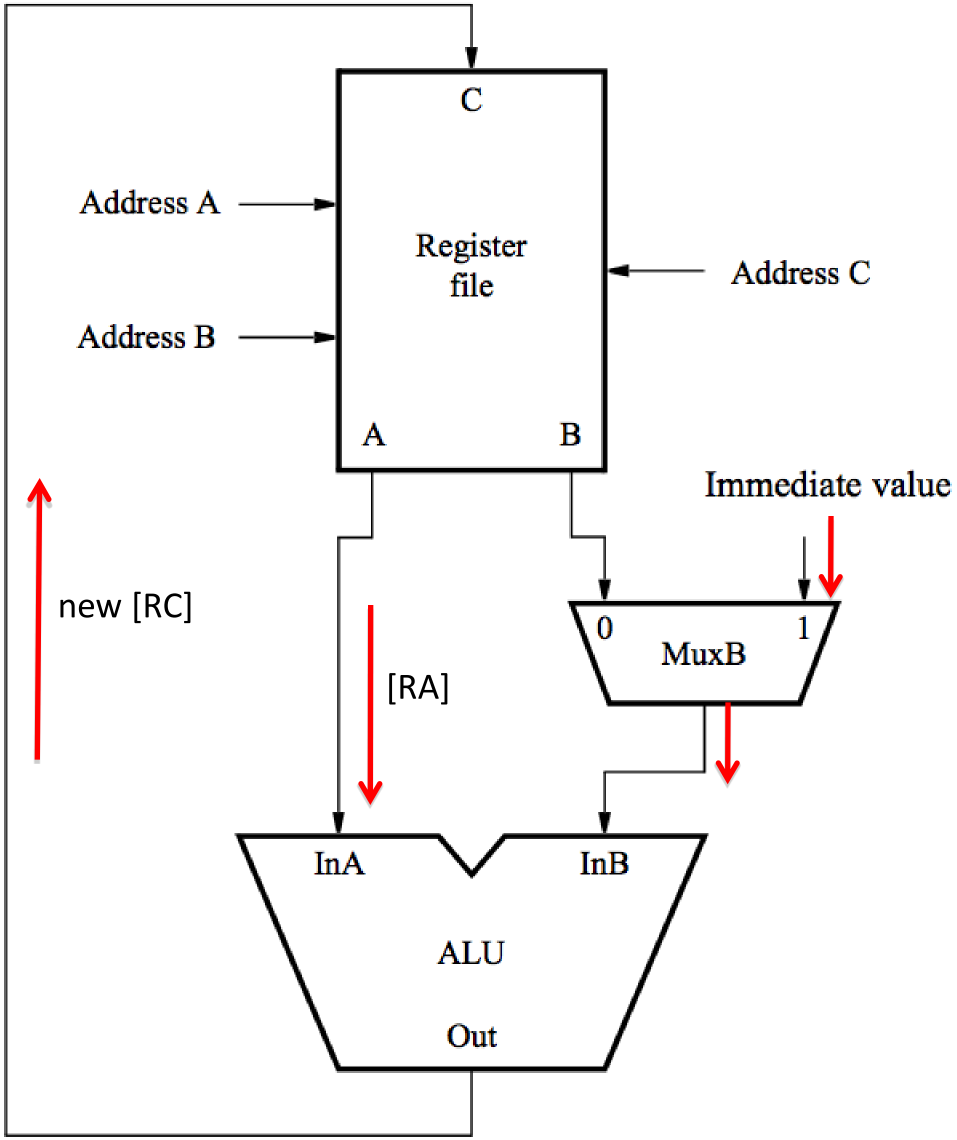
  [RA] and [RB] denote values of registers that are identified by addresses A and B

  new [RC] denotes the result that is stored to the register identified by address C



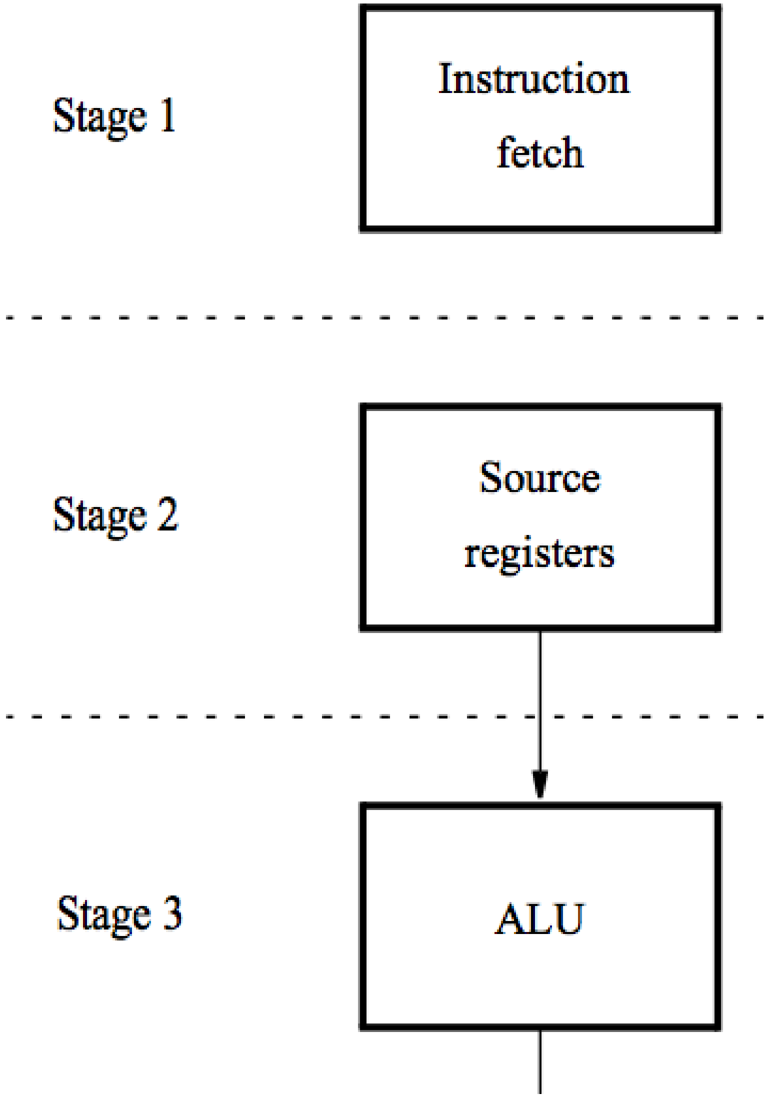
A conceptual view – immediate instructions

- One of the source operands is the immediate value in the IR.



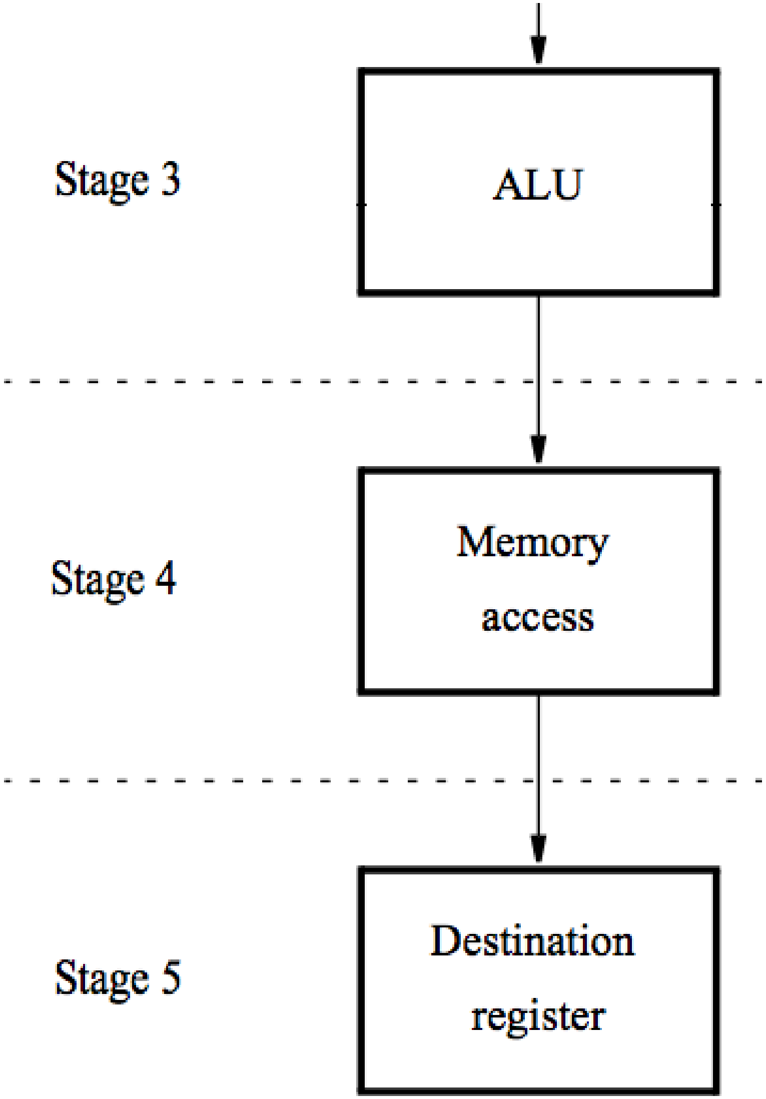
A 5-stage implementation of a RISC processor

- Instruction processing moves from stage to stage in every clock cycle, starting with fetch.
- The instruction is decoded and the source registers are read in stage 2.
- Computation takes place in the ALU in stage 3.



A 5-stage implementation of a RISC processor

- If a memory operation is involved, it takes place in stage 4.
- The result of the instruction is stored in the destination register in stage 5.



The datapath – Stages 2 to 5

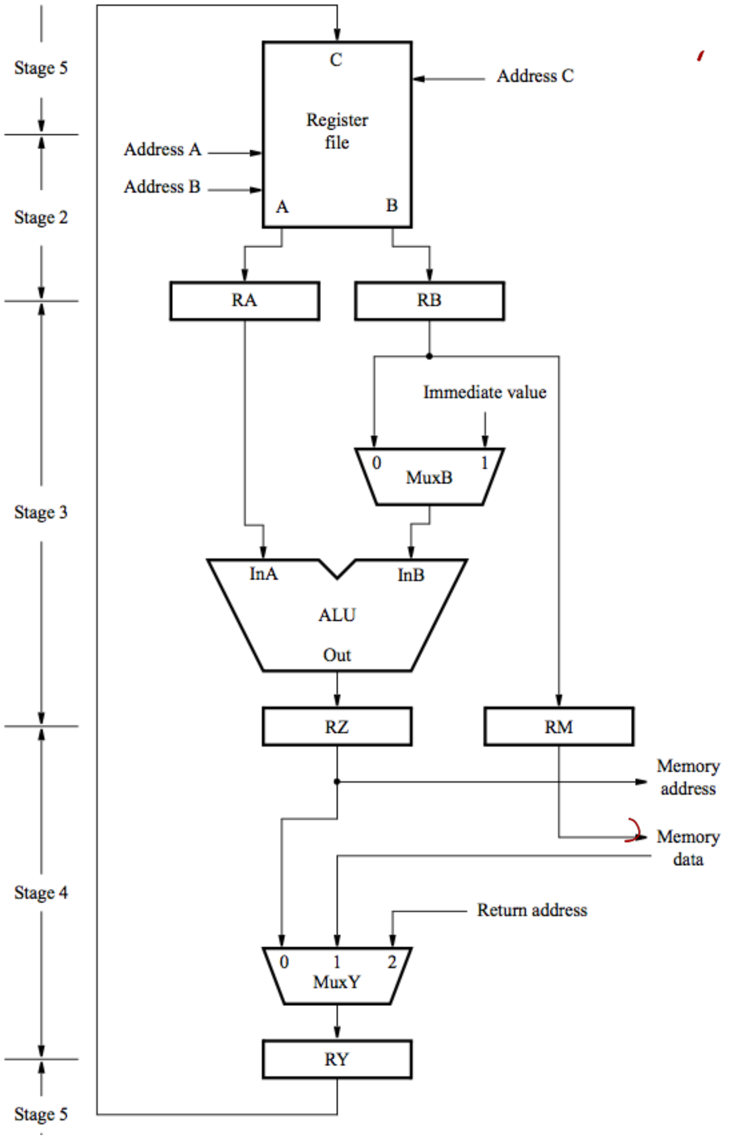
- Register file,
  used in stages 2 and 5

  (Inter-stage registers RA, RB, RZ, RY needed to carry data from one stage to the next.)

- ALU stage

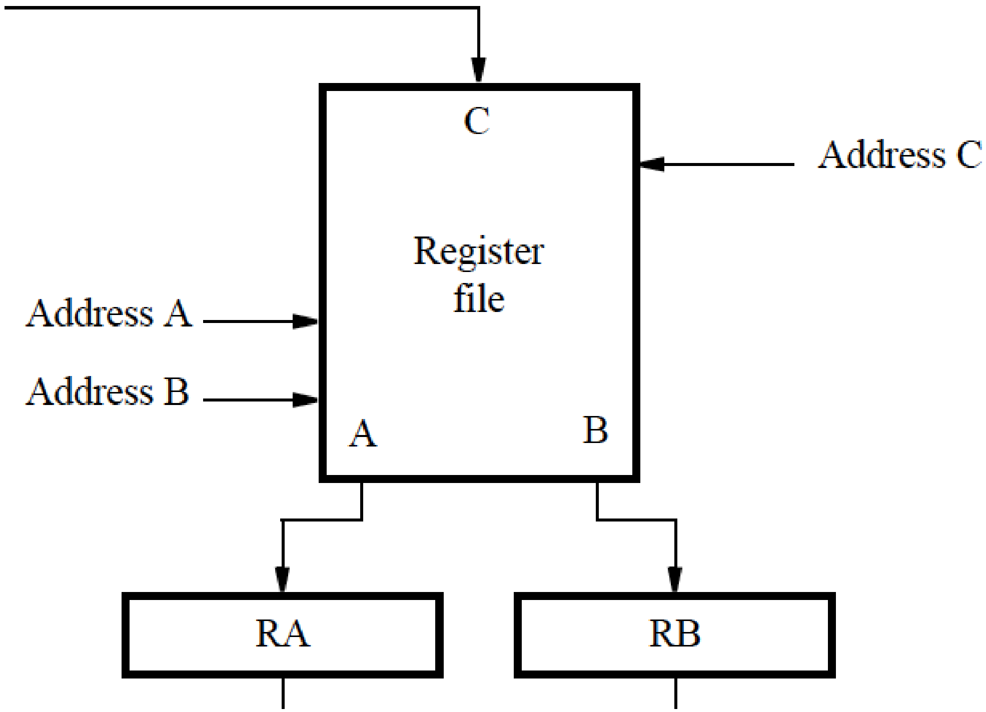
- Memory stage

- Final stage to store result to the register file



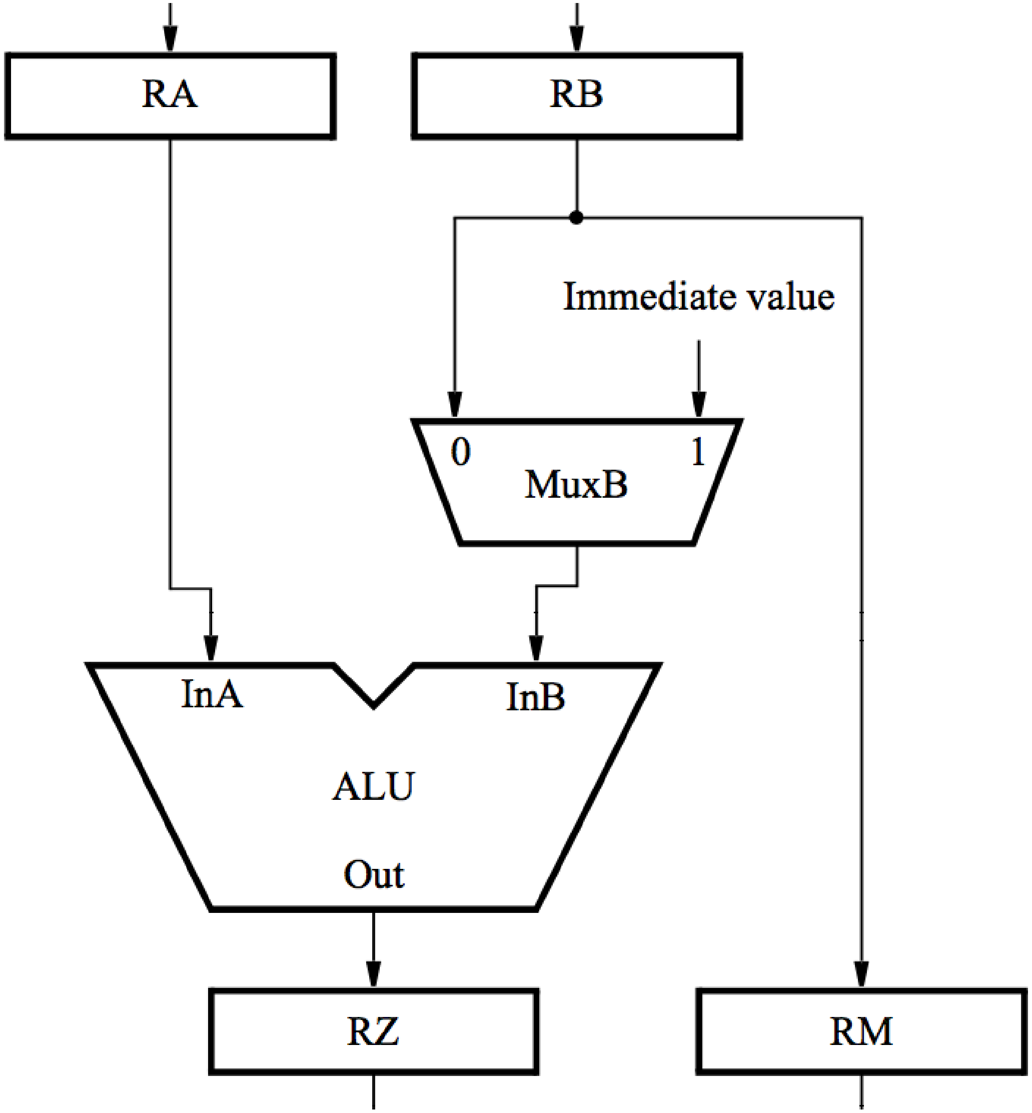
Register file – Stages 2 & 5

- Address inputs are connected to the corresponding fields in IR.
- Source registers are read in stage 2; their contents are stored in RA and RB.
- In stage 5, the result of the instruction is stored in the destination register selected by address C.



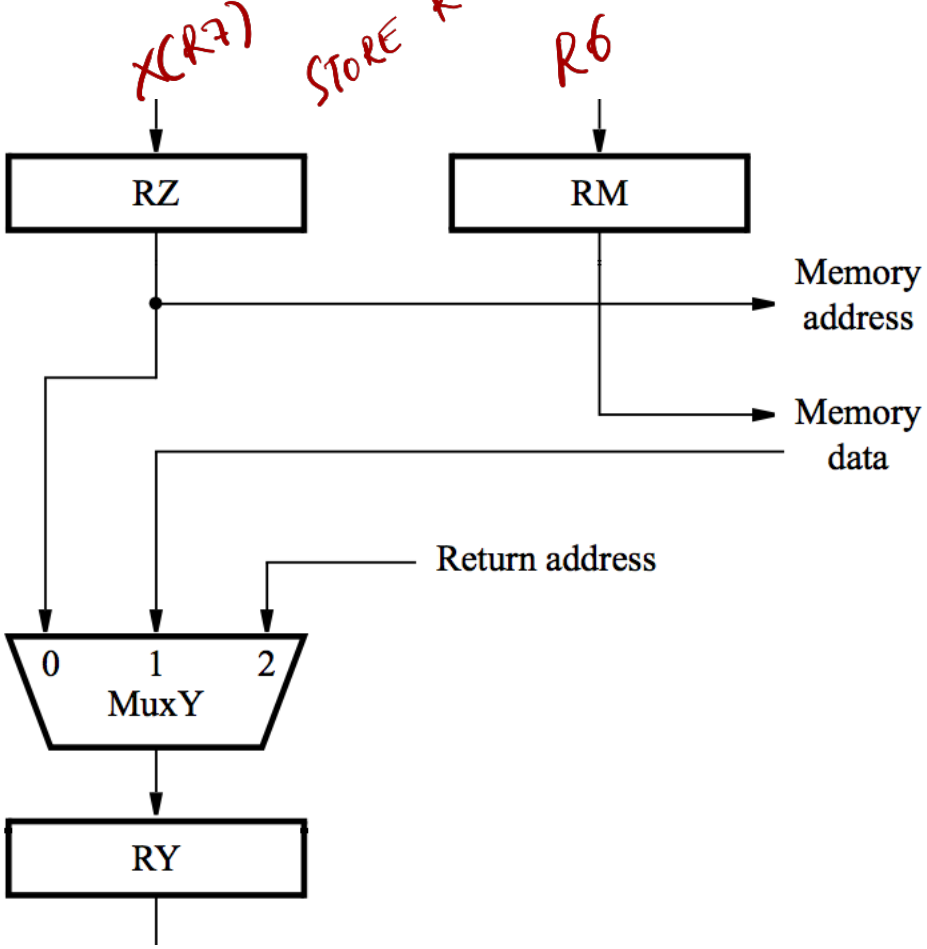
ALU stage

- ALU performs calculation specified by the instruction.
- Multiplexer MuxB selects either RB or the Immediate field of IR.
- Results stored in RZ.
- Data to be written in the memory are transferred from RB to RM.



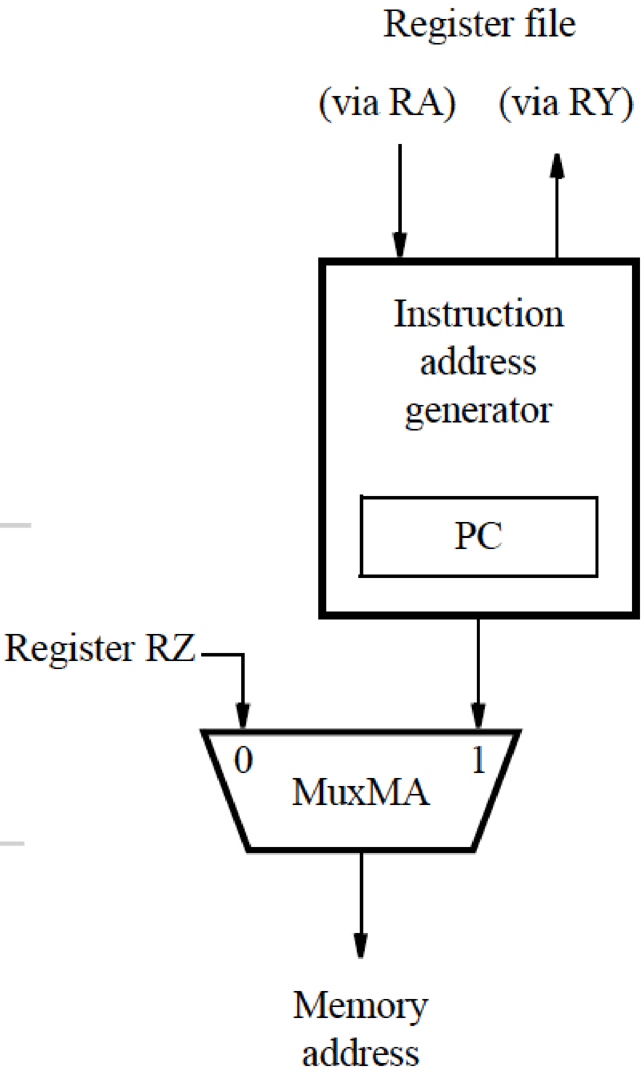
Memory stage

- For a memory instruction, RZ provides memory address, and MuxY selects read data to be placed in RY.
- RM provides data for a memory write operation.
- For a calculation instruction, MuxY selects [RZ] to be placed in RY.
- Input 2 of MuxY is used in subroutine calls.



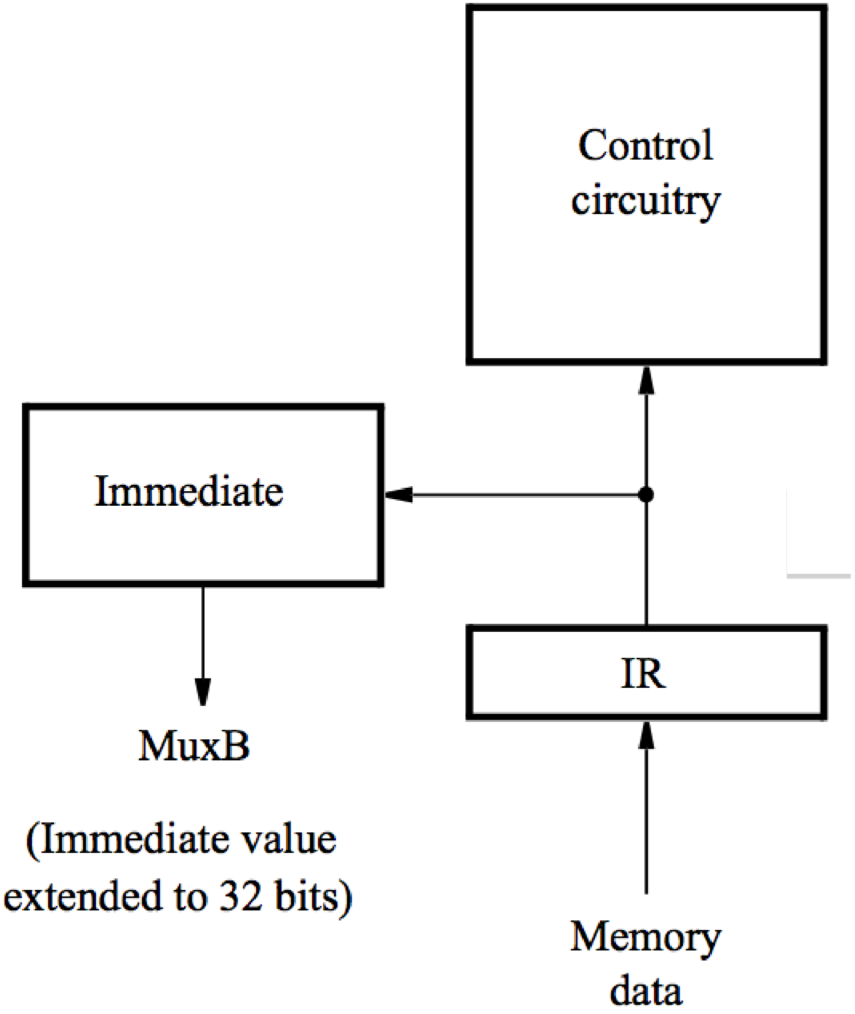
Memory address generation

- MuxMA selects the PC when fetching instructions.
- The Instruction address generator increments the PC after fetching an instruction.
- It also generates branch and subroutine addresses.
- MuxMA selects RZ when reading/writing data operands.



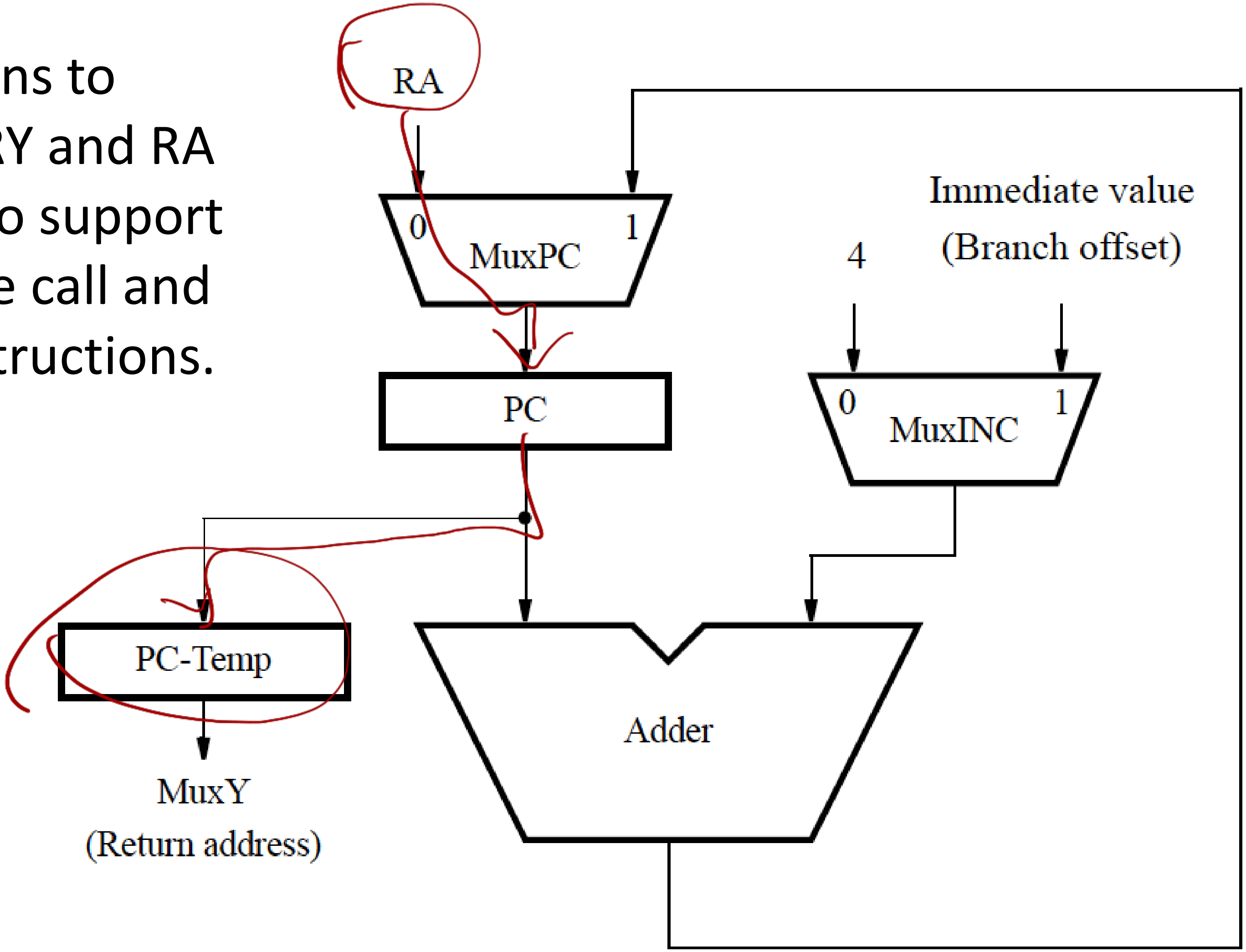
Processor control section

- When an instruction is read, it is placed in IR.
- The control circuitry decodes the instruction.
- It generates the control signals that drive all units.
- The Immediate block extends the immediate operand to 32 bits, according to the type of instruction.



Instruction address generator

- Connections to registers RY and RA are used to support subroutine call and return instructions.



Example: Add R3, R4, R5

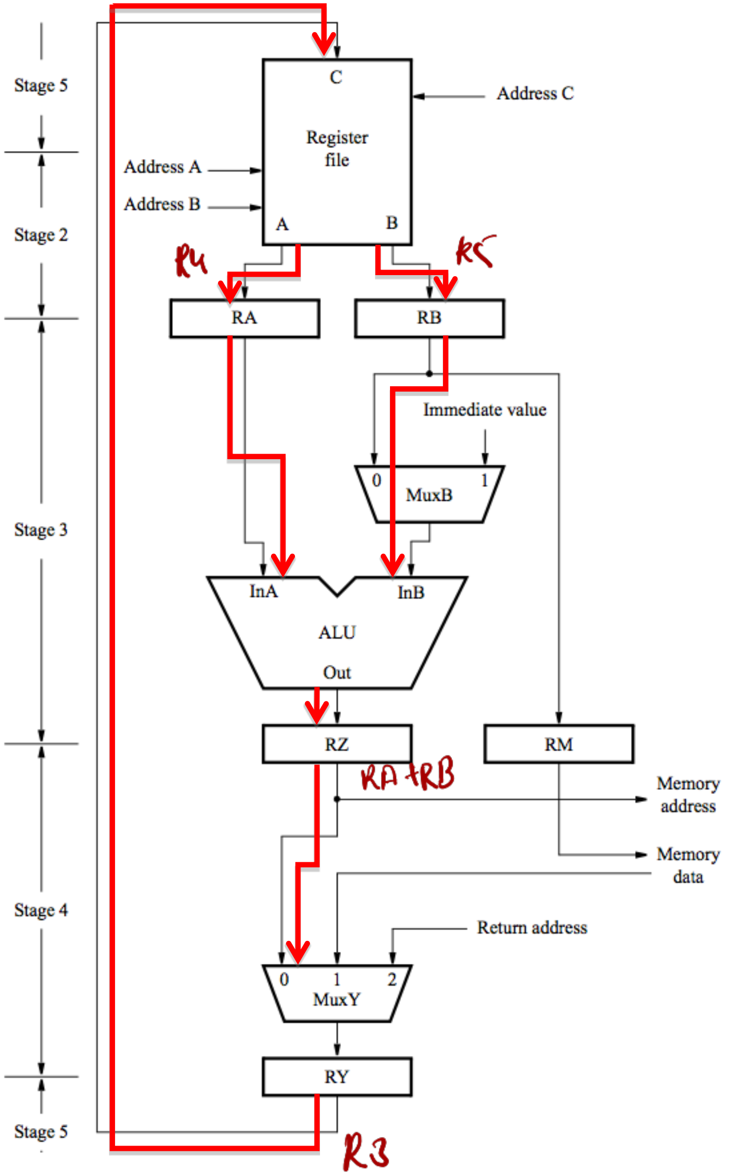
1. Memory address ← [PC], Read memory, IR ←Memory data, PC ← [PC] + 4

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

3. RZ ← [RA] + [RB]

4. RY ← [RZ]

5. R3 ← [RY]



Example: Load R5, X(R7)

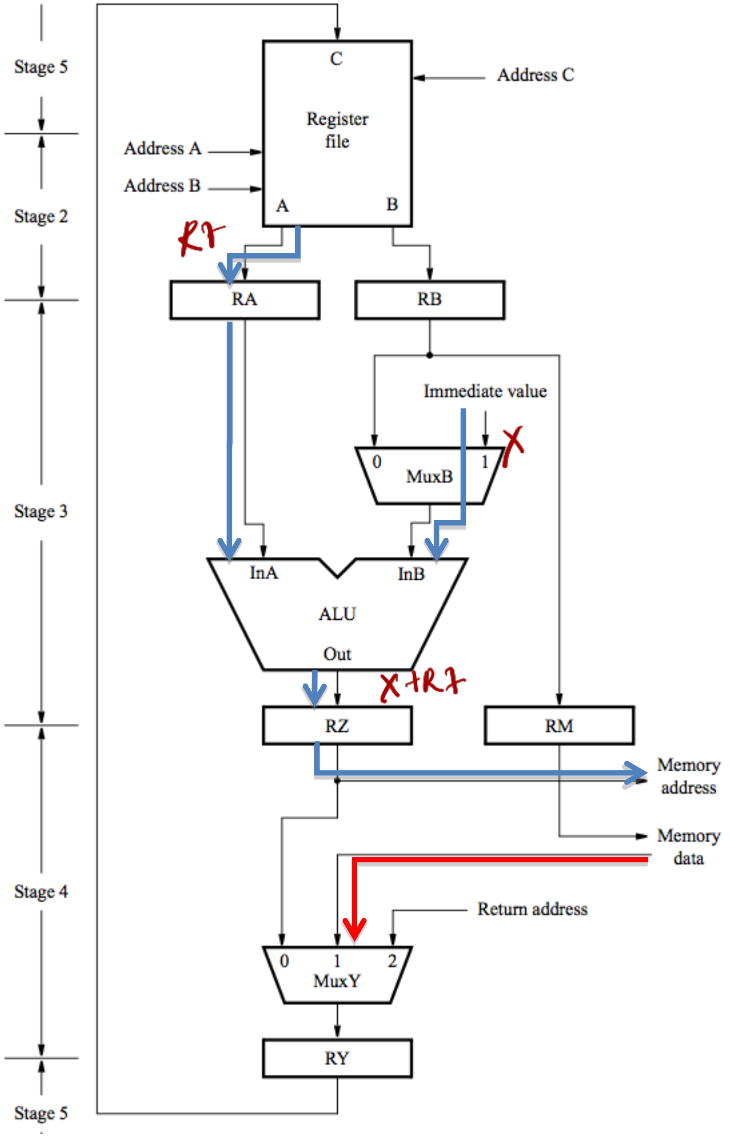
1. Memory address ← [PC], Read memory, IR ← Memory data, PC ←[PC] + 4

2. Decode instruction, RA ←[R7]

3. RZ ←[RA] + Immediate value X

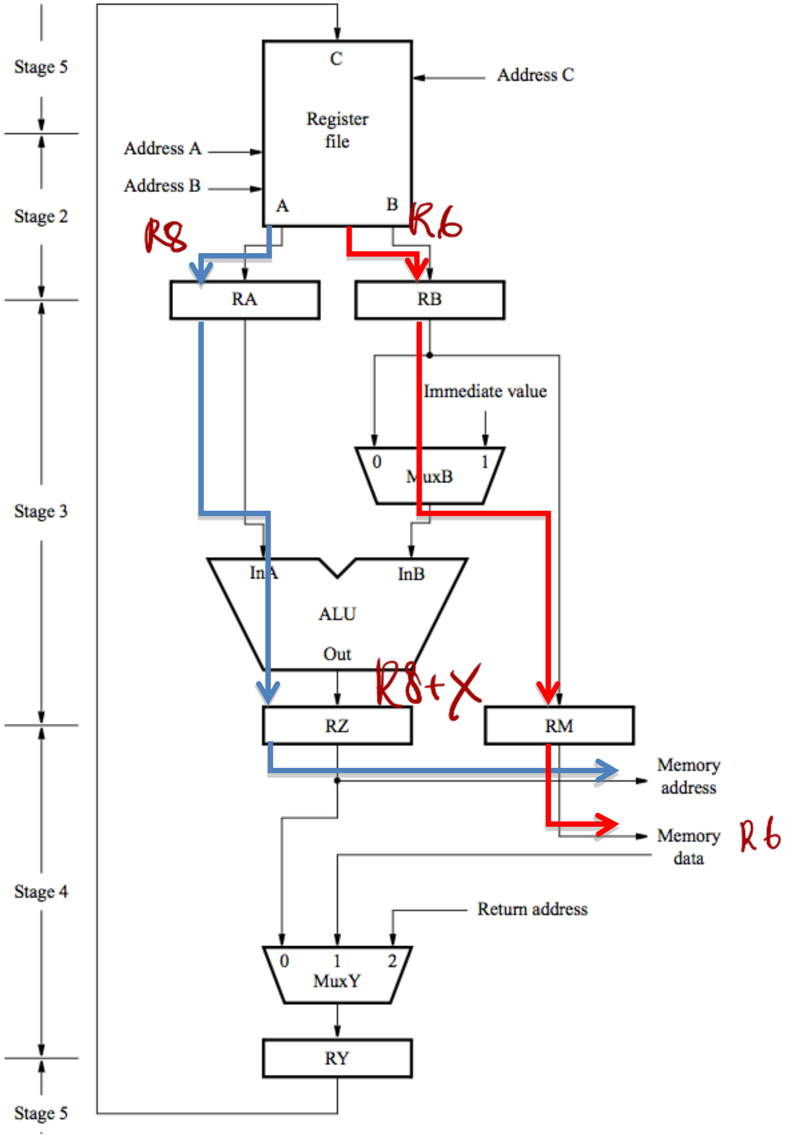
4. Memory address ←[RZ], Read memory, RY ← Memory data

5. R5 ←[RY]



Example: Store R6, X(R8)

1. Memory address ←[PC], Read memory, IR ← Memory data, PC ← [PC] + 4
2. Decode instruction, RA ←[R8], RB ←[R6]
3. RZ ←[RA] + Immediate value X, RM ←[RB]
4. Memory address ←[RZ], Memory data ←[RM], Write memory
5. No action



Unconditional branch

1. Memory address $\leftarrow$[PC], Read memory, IR $\leftarrow$ Memory data, PC $\leftarrow$[PC] $+$ 4
2. Decode instruction
3. PC $\leftarrow$[PC] $+$ Branch offset
4. No action
5. No action

Conditional branch: Branch_if_[R5]=[R6] LOOP

1. Memory address ← [PC], Read memory, IR ← Memory data, PC ← [PC] + 4
2. Decode instruction, RA ← [R5], RB ← [R6]
3. Compare [RA] to [RB],
   If [RA] = [RB], then PC ← [PC] + Branch offset
4. No action
5. No action

Subroutine call with indirection: Call_register R9

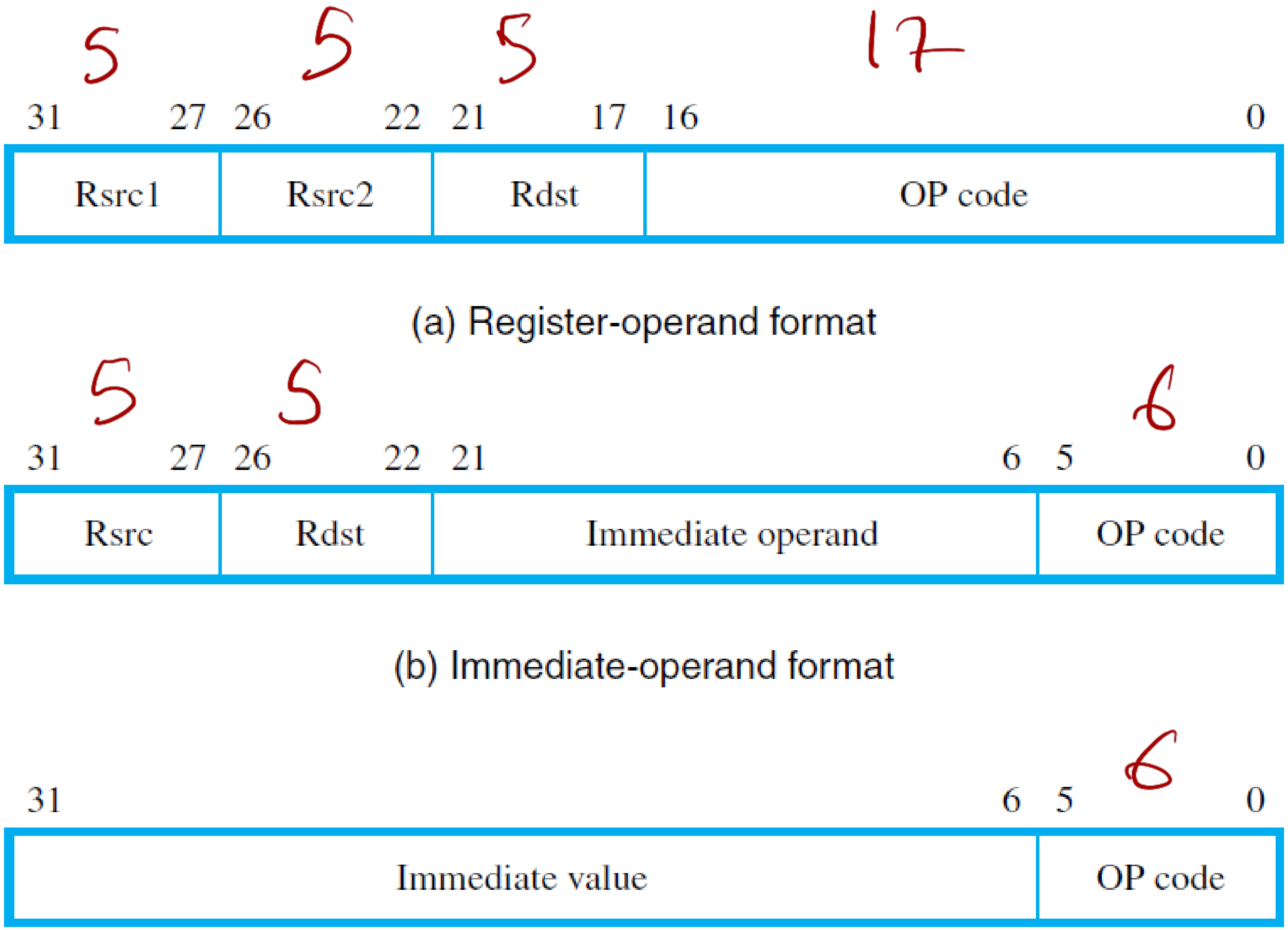
1. Memory address ←[PC], Read memory, IR ← Memory data, PC ←[PC] + 4
2. Decode instruction, RA ←[R9]
3. PC-Temp ←[PC], PC ←[RA]
4. RY ← [PC-Temp]
5. Register LINK ← [RY]

What about memory access delay??

Control signals

- Select multiplexer inputs to guide the flow of data.
- Set the function performed by the ALU.
- Determine when data are written into the PC, the IR, the register file, and the memory.
- Inter-stage registers are always enabled because their contents are only relevant in the cycles for which the stages connected to the register outputs are active.

Instruction Encoding



5	5	5	17
31 27	26 22	21 17	16 0
Rsrc1	Rsrc2	Rdst	OP code

(a) Register-operand format

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

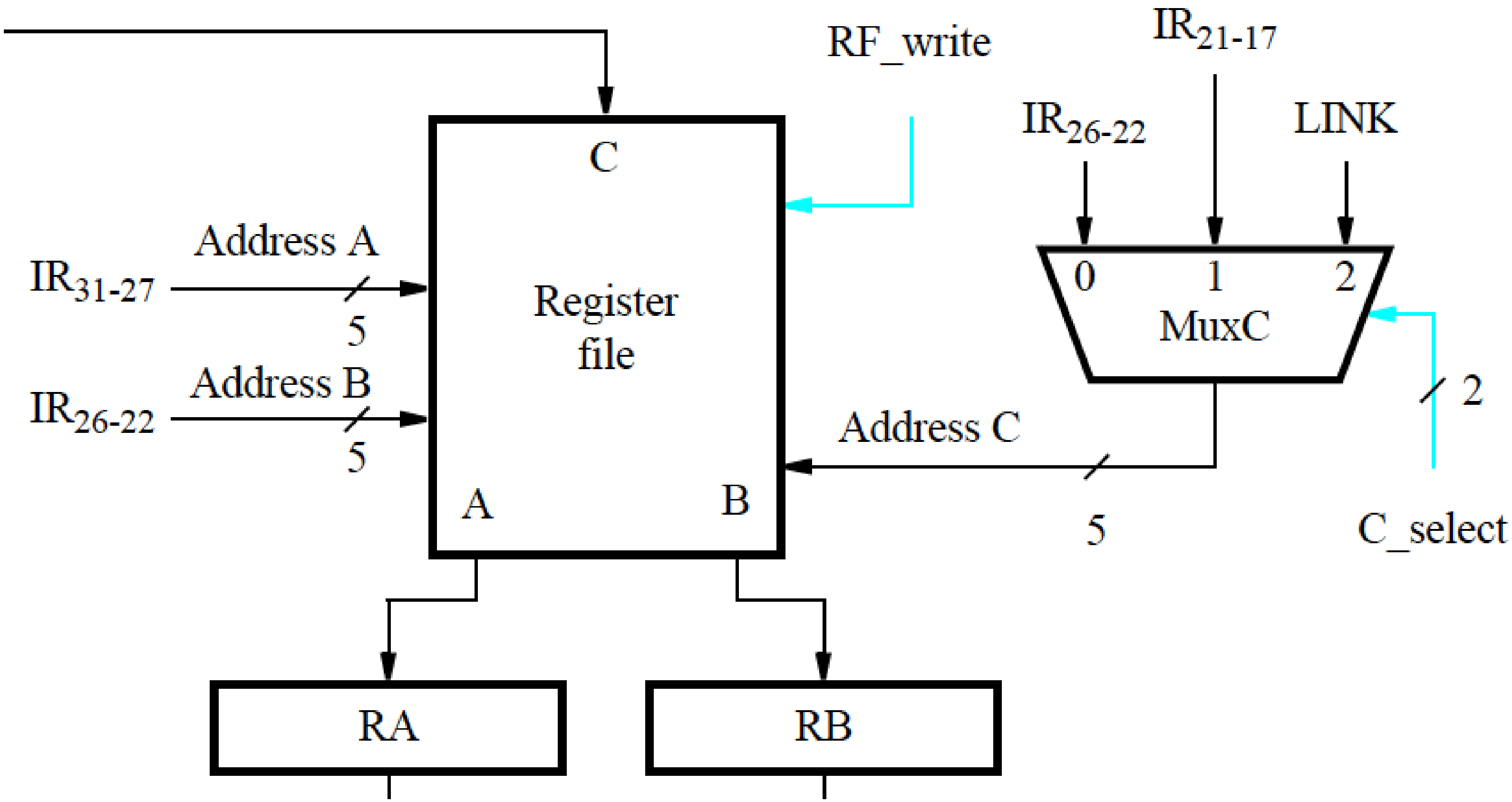
(b) Immediate-operand format

	6
31 6	5 0
Immediate value	OP code

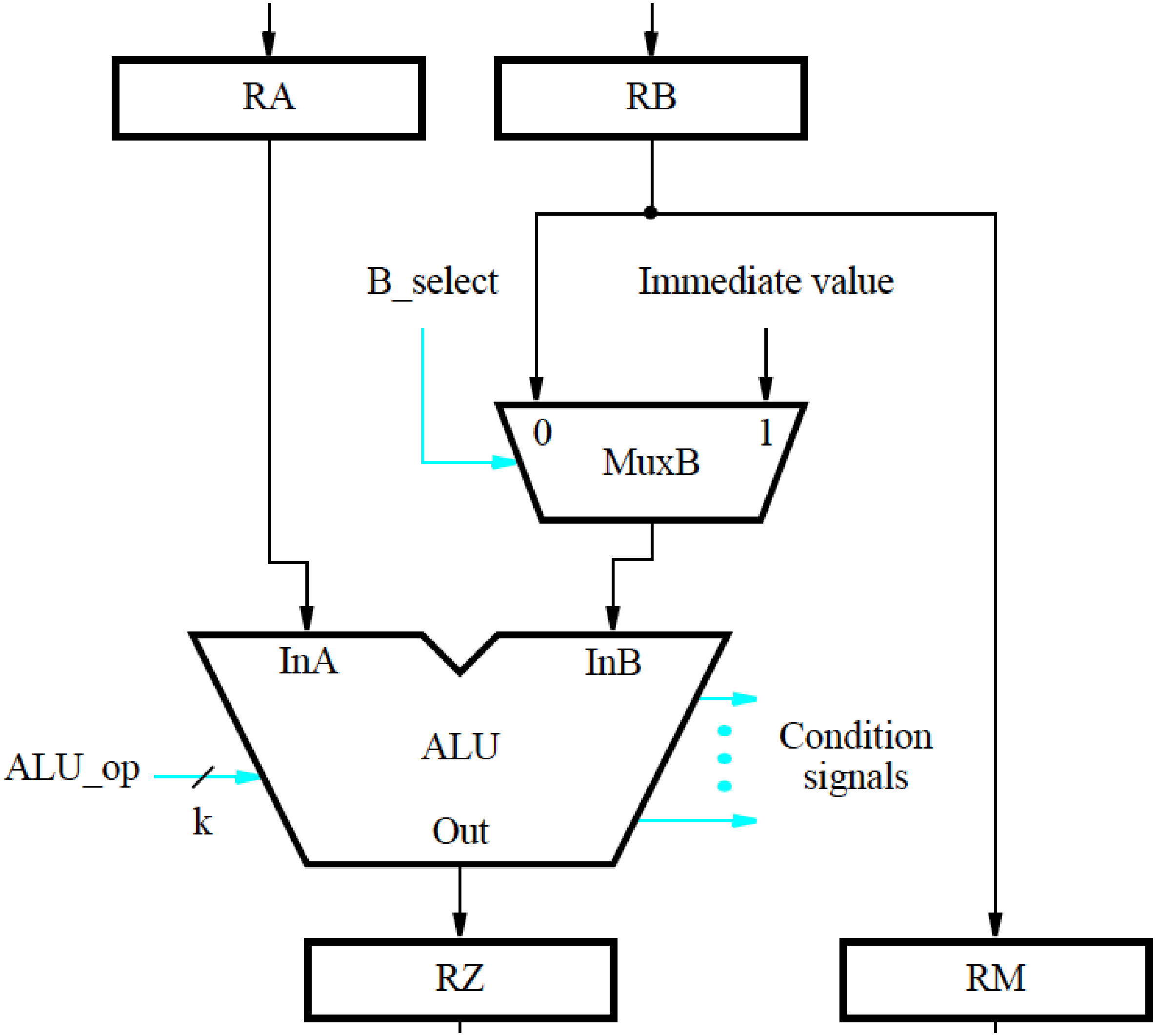
(c) Call format

Instruction encoding.

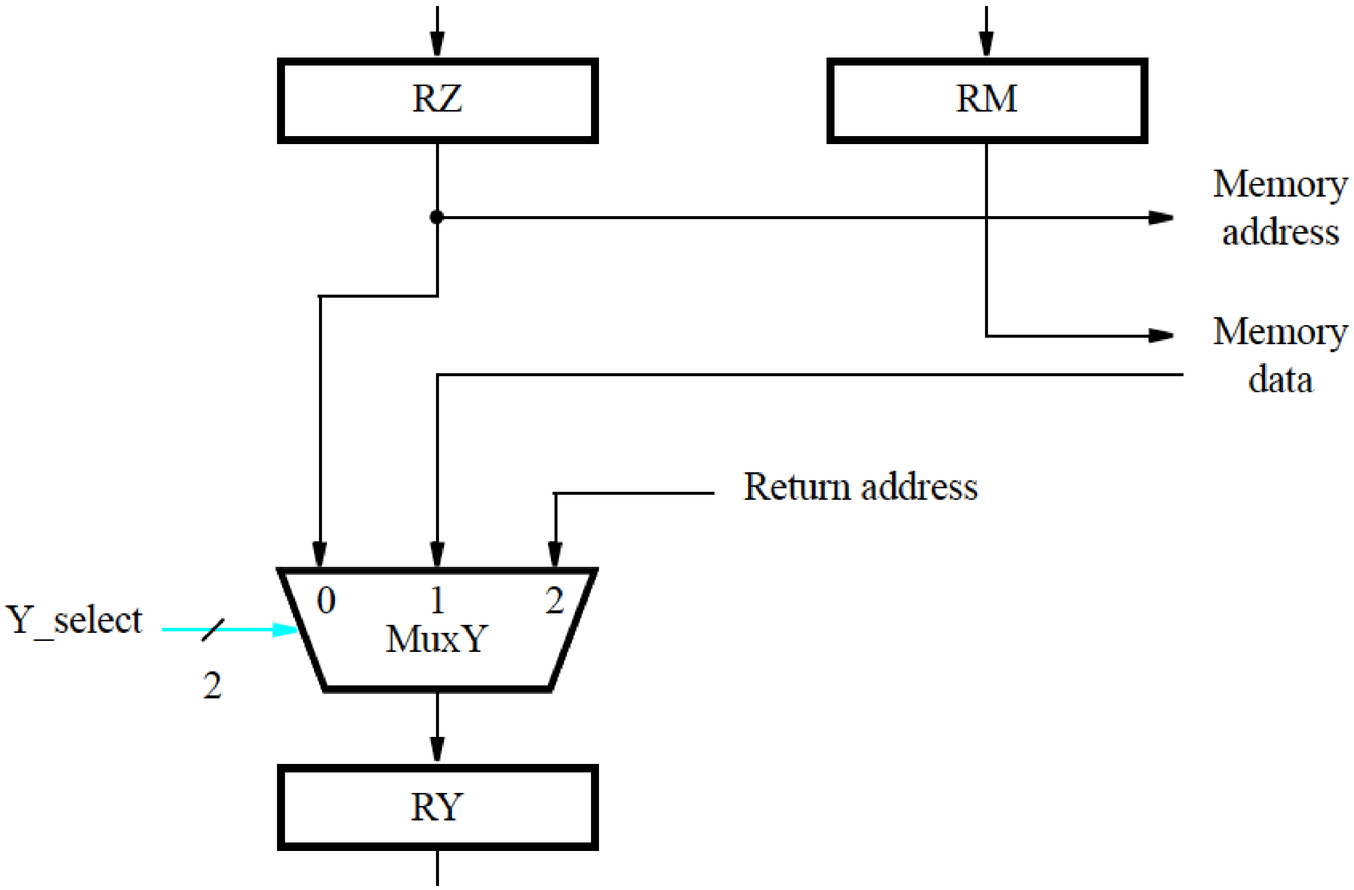
Register file control signals



ALU control signals



Result selection

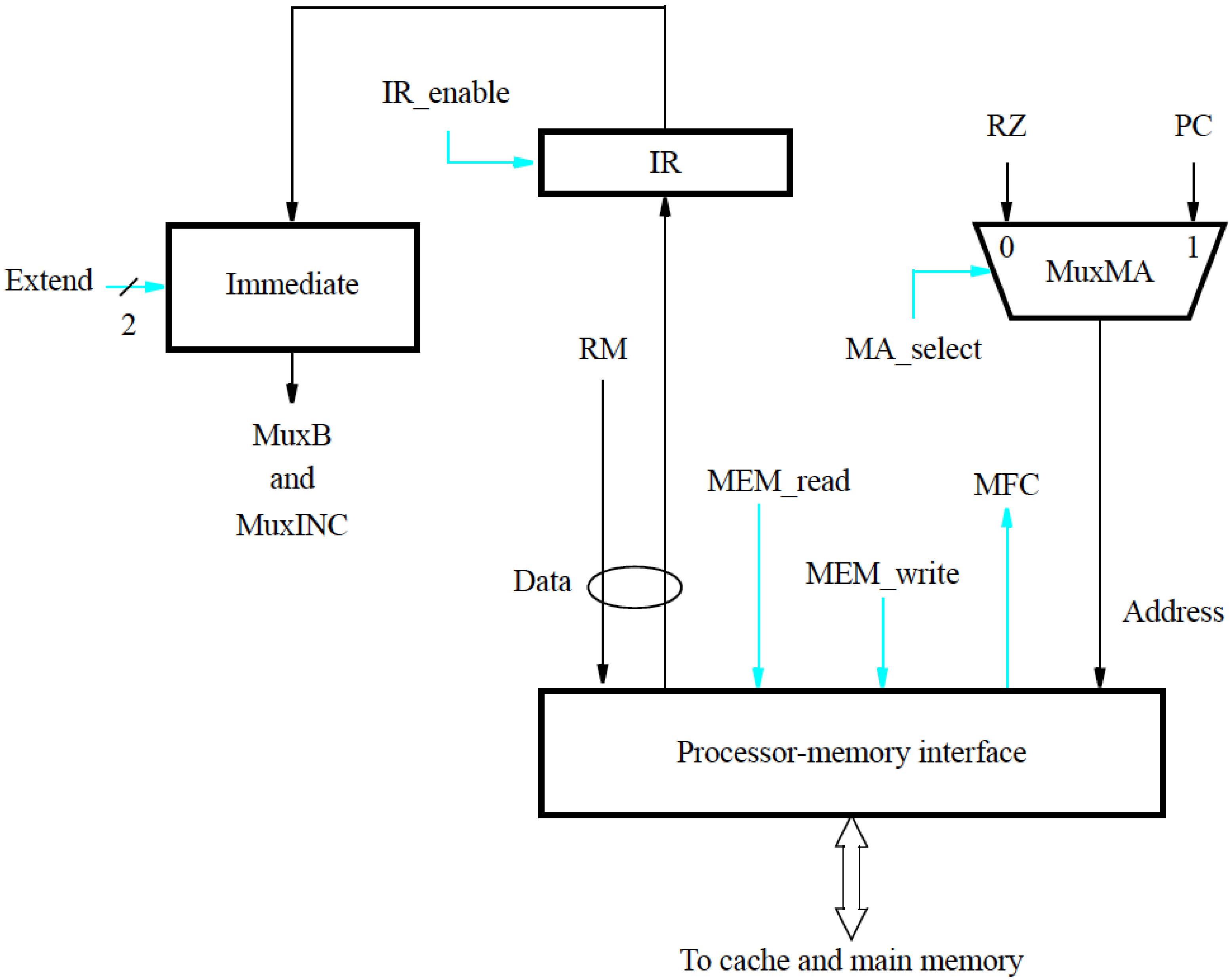


Memory access

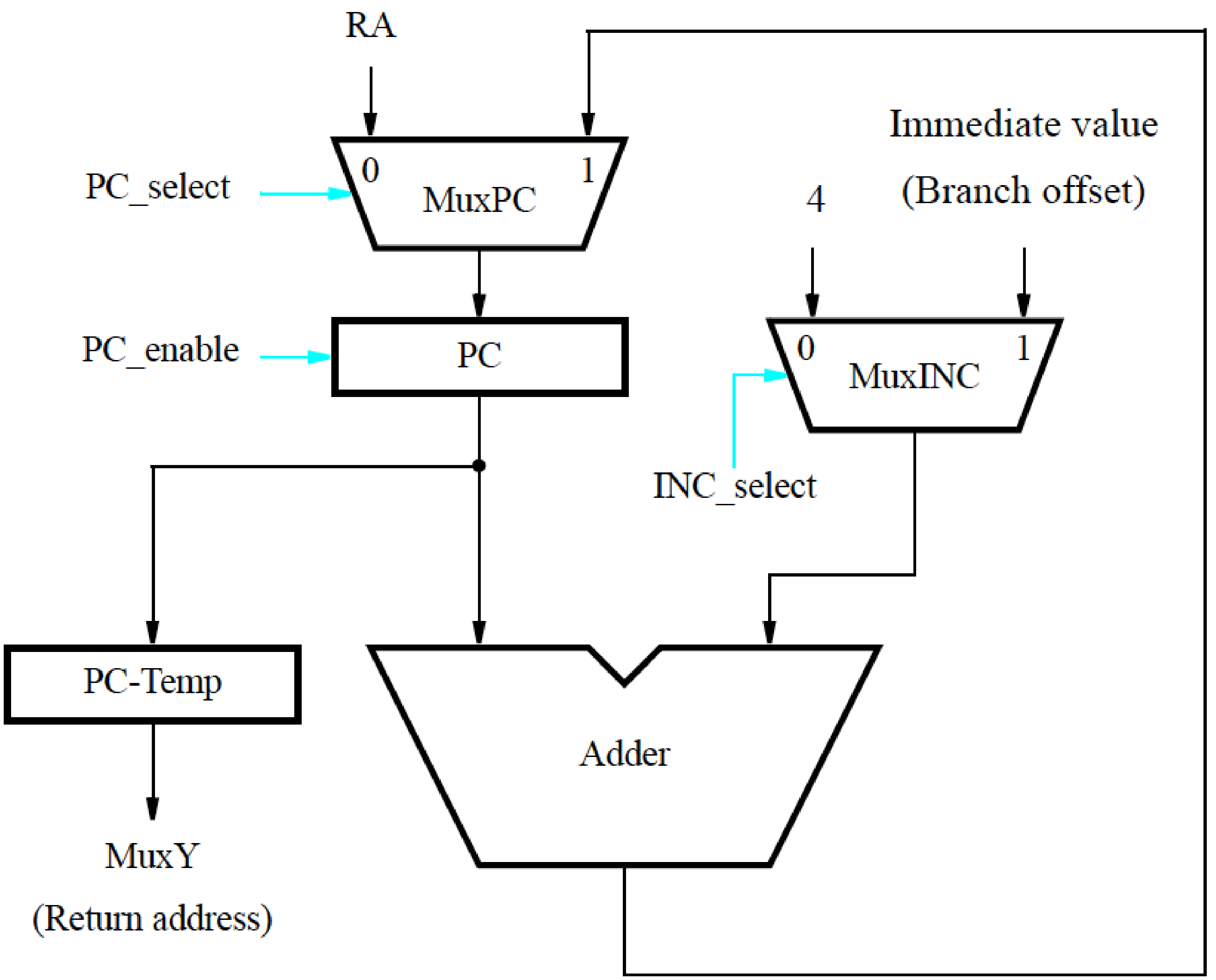
- Cache memory described earlier as faster and smaller storage that is an adjunct to the larger and slower main memory.
- When data are found in the cache, access to memory can be completed in one clock cycle.
- Otherwise, read and write operations may require several clock cycles to load data from main memory into the cache.
- A control signal is needed to indicate that memory function has been completed (MFC). E.g., for step 1:

1. Memory address $\leftarrow$ [PC], Read memory, Wait for MFC,
   IR $\leftarrow$ Memory data, PC $\leftarrow$[PC] $+$ 4

Memory and IR control signals



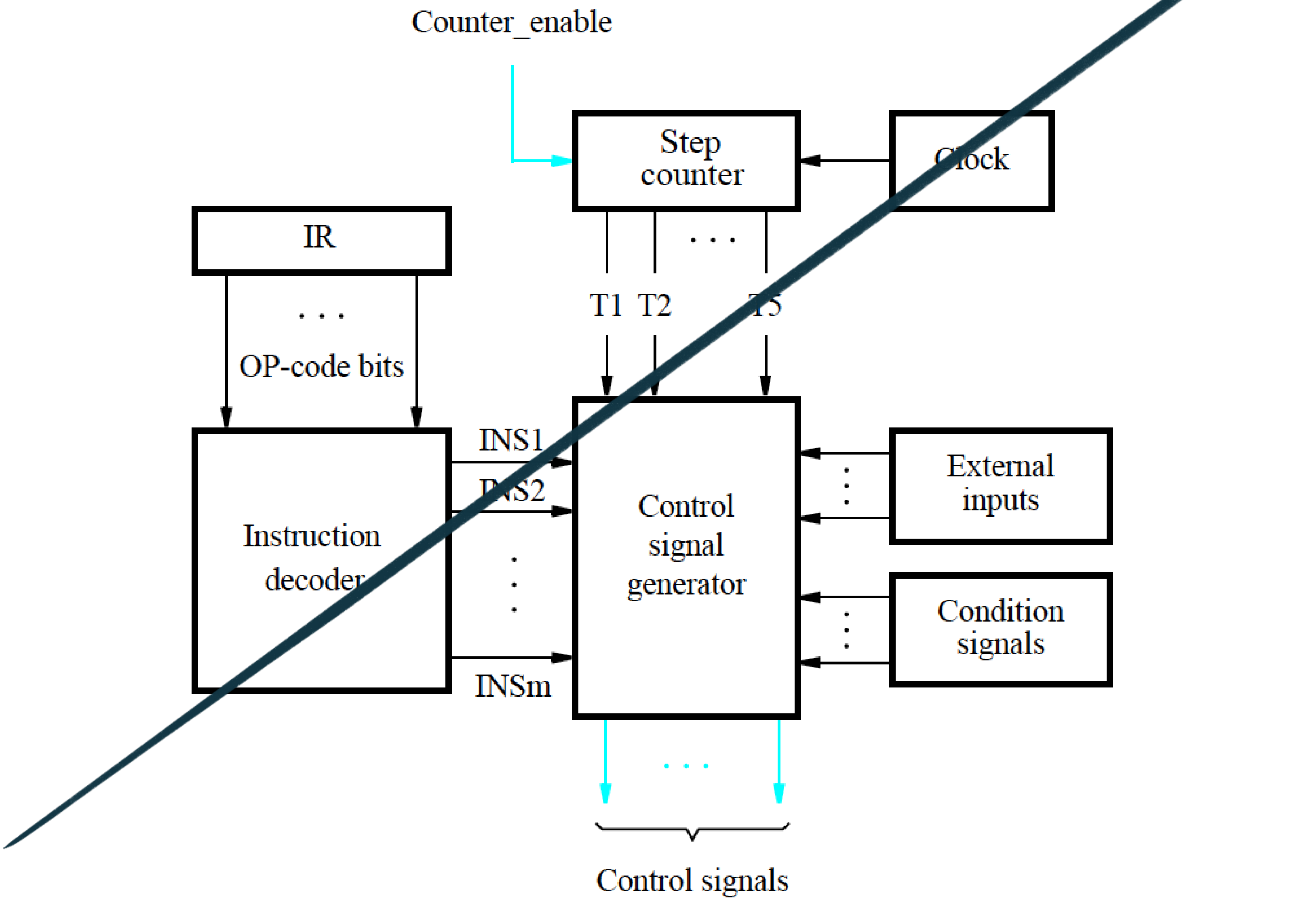
Control signals of instruction address generator



Control signal generation

- Actions to fetch & execute instructions have been described.
- The necessary control signals have also been described.
- Circuitry must be implemented to generate control signals so actions take place in correct sequence and at correct time.
- There are two basic approaches:
  hardwired control and microprogramming
- Hardwired control involves implementing circuitry that considers step counter, IR, ALU result, and external inputs.
- Step counter keeps track of execution progress,
  one clock cycle for each of the five steps described earlier
  (unless a memory access takes longer than one cycle).

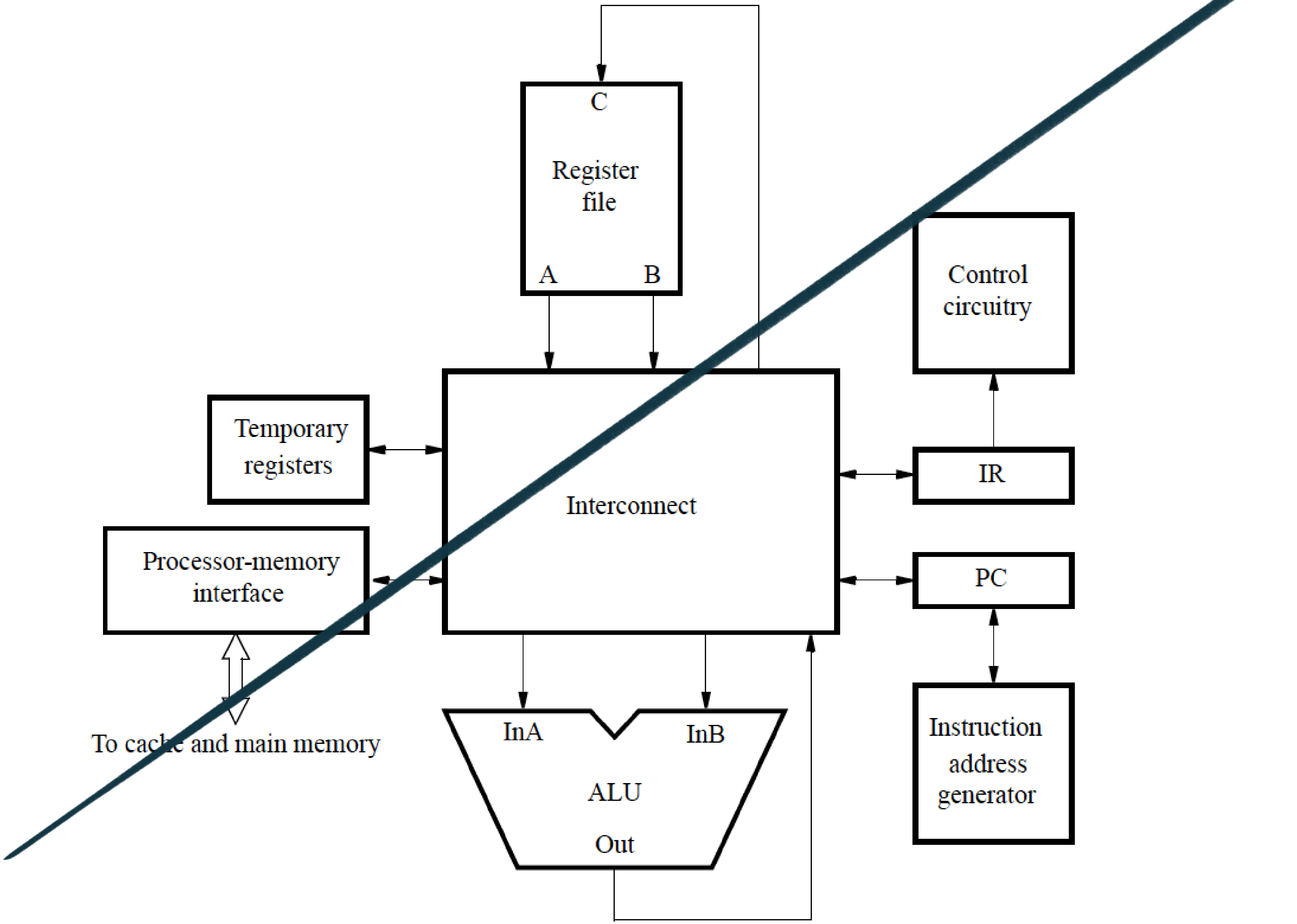
Hardwired generation of control signals



CISC processors

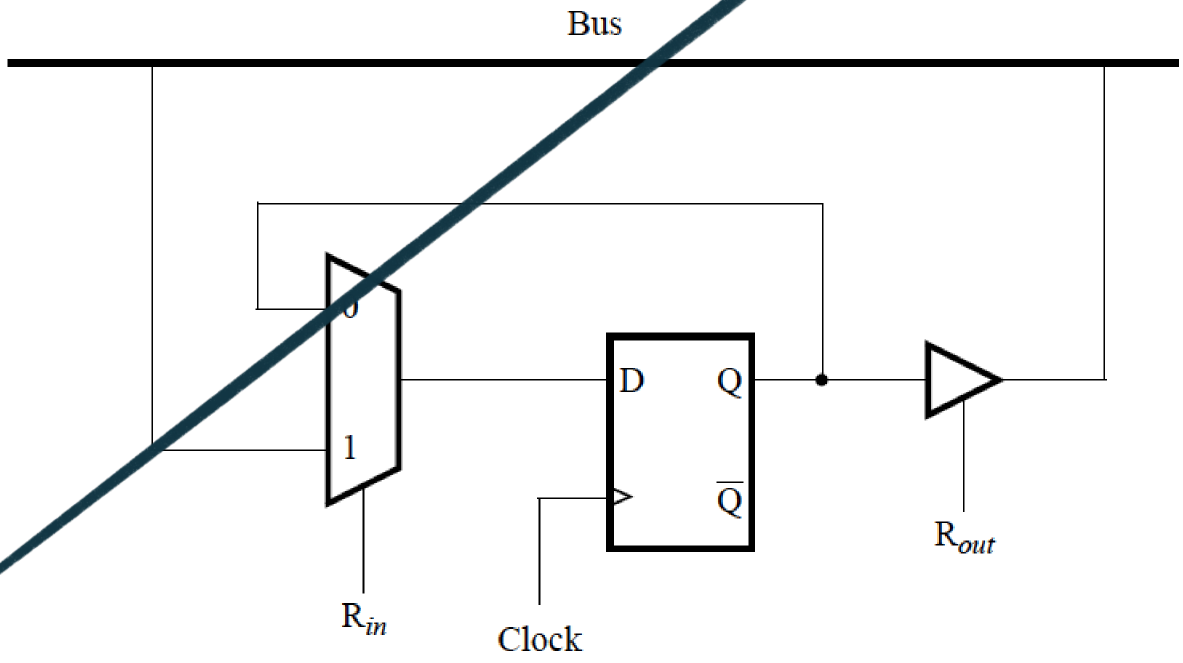
- CISC-style processors have more complex instructions.
- The full collection of instructions cannot all be implemented in a fixed number of steps.
- Execution steps for different instructions do not all follow a prescribed sequence of actions.
- Hardware organization should therefore enable a flexible flow of data and actions to accommodate CISC.

Hardware organization for a CISC computer

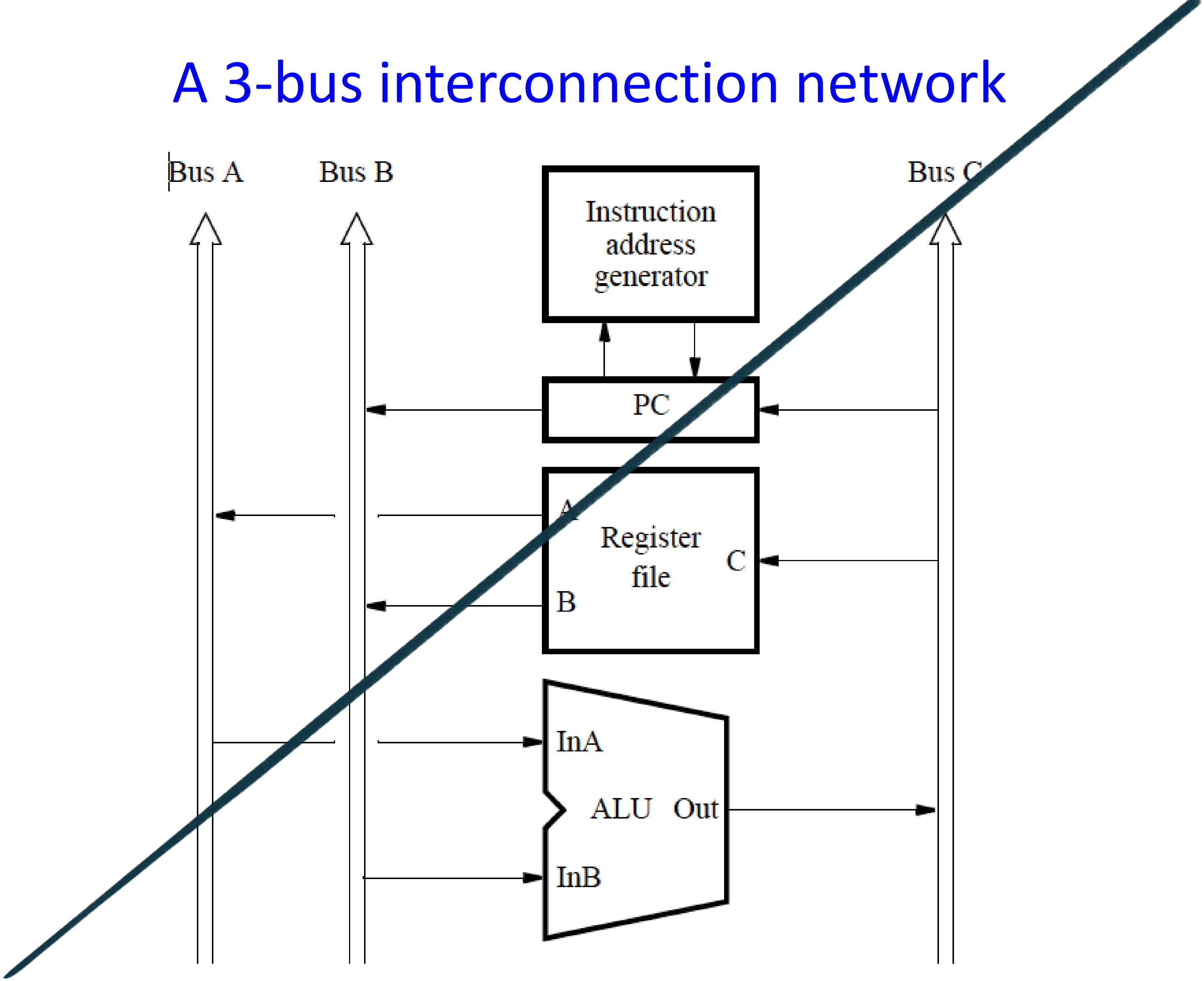


Bus

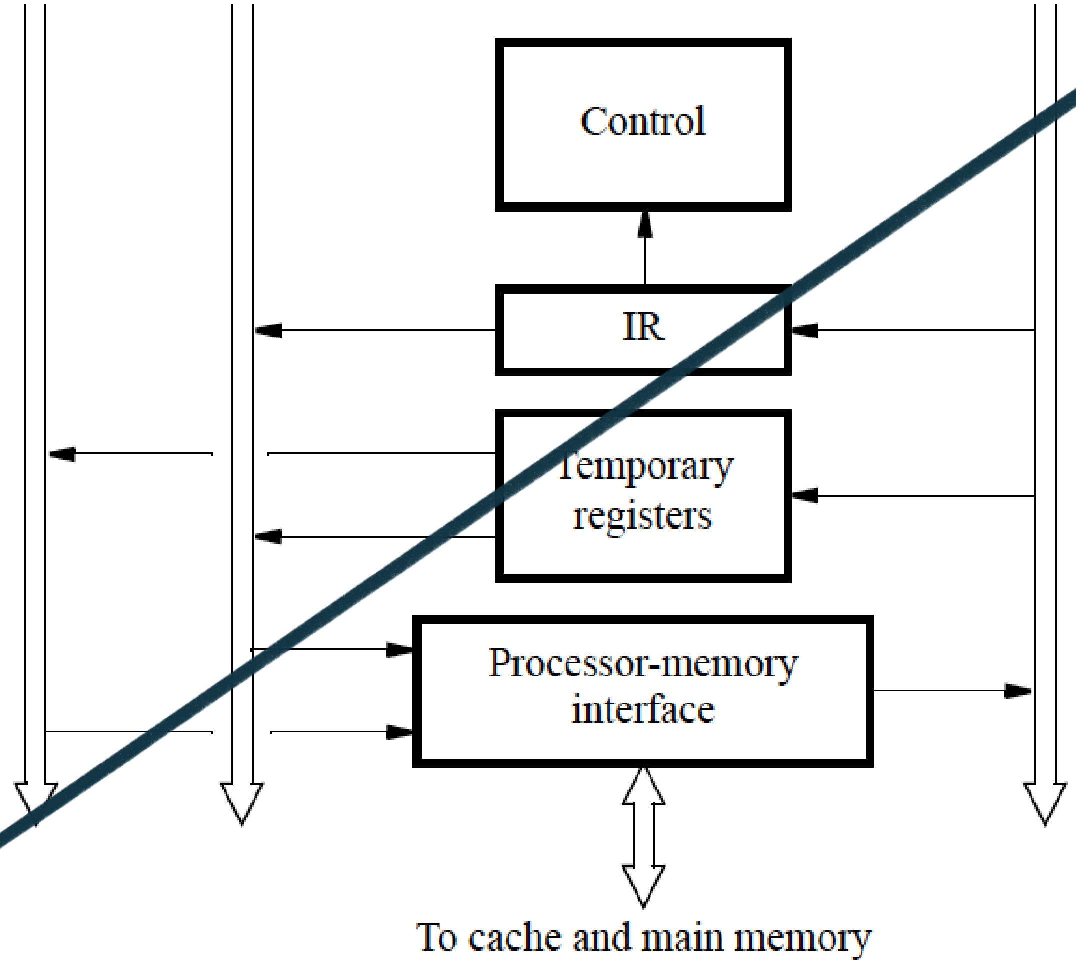
- An example of an interconnection network.
- When functional units are connected to a common bus, tri-state drivers are needed.



A 3-bus interconnection network



A 3-bus interconnection network (contd.)



Example: Add R5, R6

1. Memory address ← [PC], Read memory, Wait for MFC,
   IR ← Memory data, PC ← [PC] + 4
2. Decode instruction
3. R5 ← [R5] + [R6]

- In step 1, bus B is used to send PC contents to the processor-memory interface, which initiates a memory Read operation. Fetched Instruction is sent to the IR over bus C.
- Instruction decoded in step 2 and the control circuitry begins reading the source registers, R5 and R6.
- Contents of the registers do not become available at the A and B outputs of the register file until step 3. They are sent to the ALU using buses A and B. The ALU performs the addition operation. Sum is sent back by the ALU over bus C, to be written into register R5 at the end of the clock cycle.

Example: AND X(R7), R9

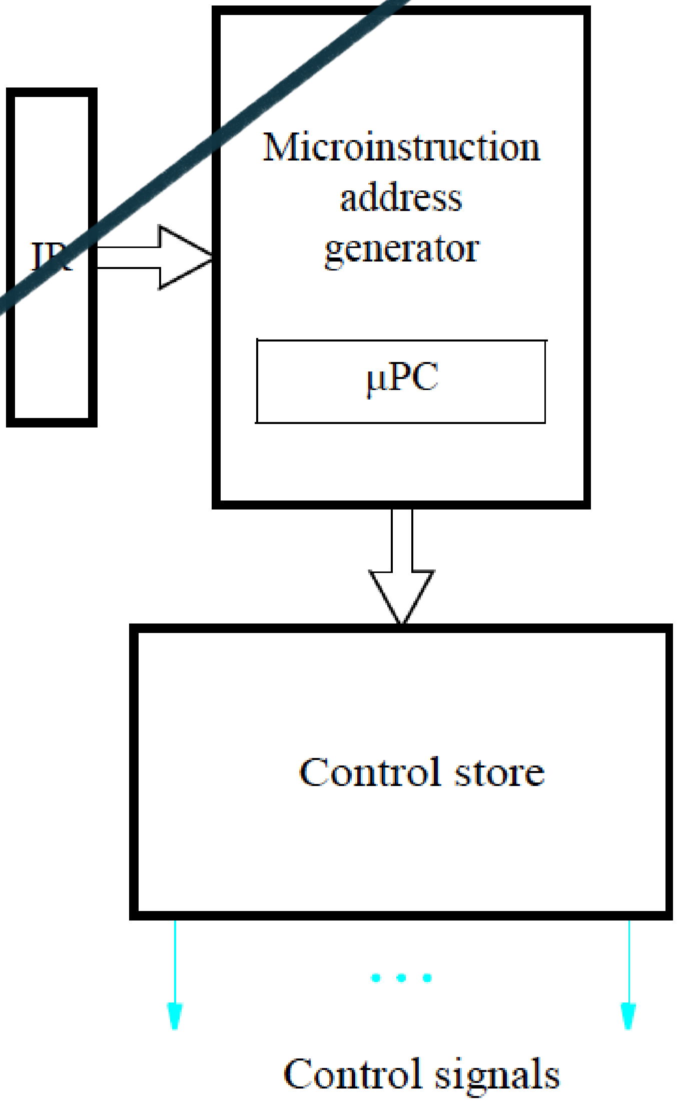
Performs AND on the contents of R9 and memory location X + [R7], stores the result back in the same memory location

[X is a 32 bit value given as the second word of the instruction]

1. Memory address ←[PC], Read memory, Wait for MFC,
   IR ← Memory data, PC ←[PC] + 4
2. Decode instruction
3. Memory address ←[PC], Read memory, Wait for MFC,
   Temp1 ← Memory data, PC ←[PC] + 4
4. Temp2 ←[Temp1] + [R7]
5. Memory address ←[Temp2], Read memory, Wait for MFC,
   Temp1 ← Memory data
6. Temp1 ←[Temp1] AND [R9]
7. Memory address ←[Temp2], Memory data ←[Temp1],
   Write memory, Wait for MFC

Microprogramming

- Microprogramming is a software-based approach
  for the generation of control signals.
- The values of the control signals for each clock period are stored in a microinstruction (control word).
- A processor instruction is implemented by a sequence of microinstructions (together called a microroutine) that are placed in a control store.
- From decoding of an instruction in IR, the control circuitry executes the corresponding sequence of microinstructions.
- μPC maintains the location of the current microinstruction.



Microprogramming

- Microinstructions for Steps 1 and 2 of all instructions common
- During step 2, microinstruction address generator decodes the instruction in IR to obtain the starting address of the corresponding microroutine and loads that address into the μPC.
- This address is used in the following clock cycle to read the micro instruction corresponding to step 3.
- As execution proceeds, the microinstruction address generator increments the μPC to read microinstructions from successive locations in the control store.
- One bit in the microinstruction, which we will call End, is used to mark the last microinstruction in a given microroutine.
- When End is equal to 1, address generator returns to the microinstruction corresponding to step 1.
- A new machine instruction is again fetched.

Microprogramming

- Microprogramming provides the flexibility needed to implement more complex instructions in CISC processors.
- However, reading and executing microinstructions incurs undesirably long delays in high-performance processors.

Sections to Read
(From Hamacher's Book)

- Chapter on Basic Processing Unit
  - All sections and sub-sections