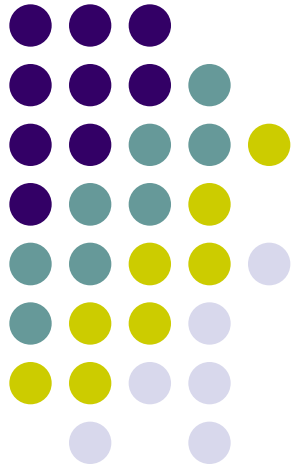
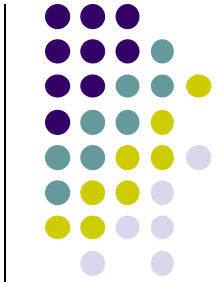
Chapter 5
Basic Processing Unit



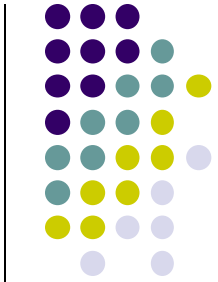
陈俊颖

Contents

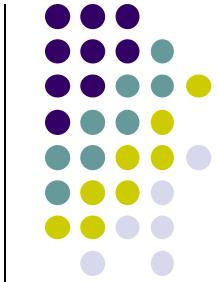
5.1 Some Fundamental Concepts



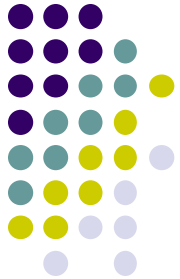
- Processing Unit
  - Instruction Set Processor or Central Processing Unit (CPU)
  - A processor is the responsible for reading program instructions from the computer's memory and executing them.
    - It fetches one instruction at a time.
    - It decodes (interprets) the instruction.
    - Then, it carries out the actions specified.



- Register Organization of Processor
  - User-Visible Registers
    - A user-visible register is one that may be referenced by means of the machine language that the CPU executes.
    - General-purpose registers
    - Data registers
      - Hold data and cannot be employed in the calculation of an operand address.

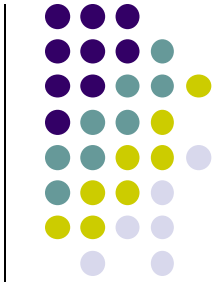


- Register Organization of Processor
  - User-Visible Registers
    - Address registers
      - Segment pointers
        - In a machine with segmented addressing, a segment register holds the address of the base of the segment.
      - Index registers
      - Stack pointer
    - Condition codes register
      - At least partially visible to the user.
      - Generally, machine instructions allow condition code bits to be read by implicit reference, but they cannot be altered by the programmer.



- Register Organization of Processor
  - Control and Status Registers
    - Most of these registers, on most machines are not visible to the user.
    - Some of them may be visible to machine instructions executed in a control or operating system mode.
    - Program Counter (PC)
    - Instruction Register (IR)
    - Memory Address Register (MAR)
    - Memory Data Register (MDR)

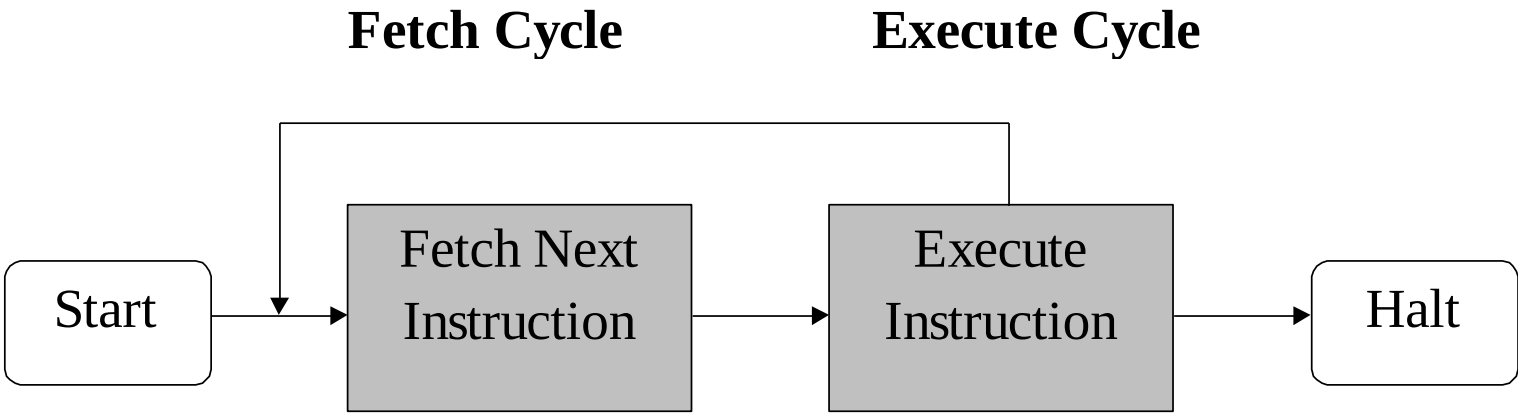
Essential to instruction execution



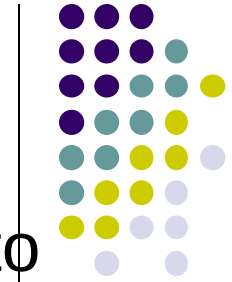
- Register Organization of Processor
  - Control and Status Registers
    - Program Status Word (PSW)
      - Include a register or a set of registers.
      - The PSW typically contains condition codes plus other status information
        - N (negative), Z (zero), V (overflow), C (carry)
        - Equal: set if a logical compare result is equality
        - Interrupt enable/disable: used to enable or disable interrupts.
        - Supervisor: indicates whether the CPU is executing in supervisor or user mode.
    - Interrupt vector register
    - Page table pointer
    - …

Related to a particular processor design

Instruction Execution

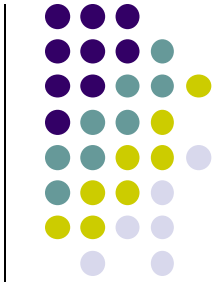


- To execute an instruction, the processor has to perform the following steps:
  - 1. Fetch the contents of the memory location pointed to by the PC. The contents of this location are loaded into the IR. IR ← [[PC]]



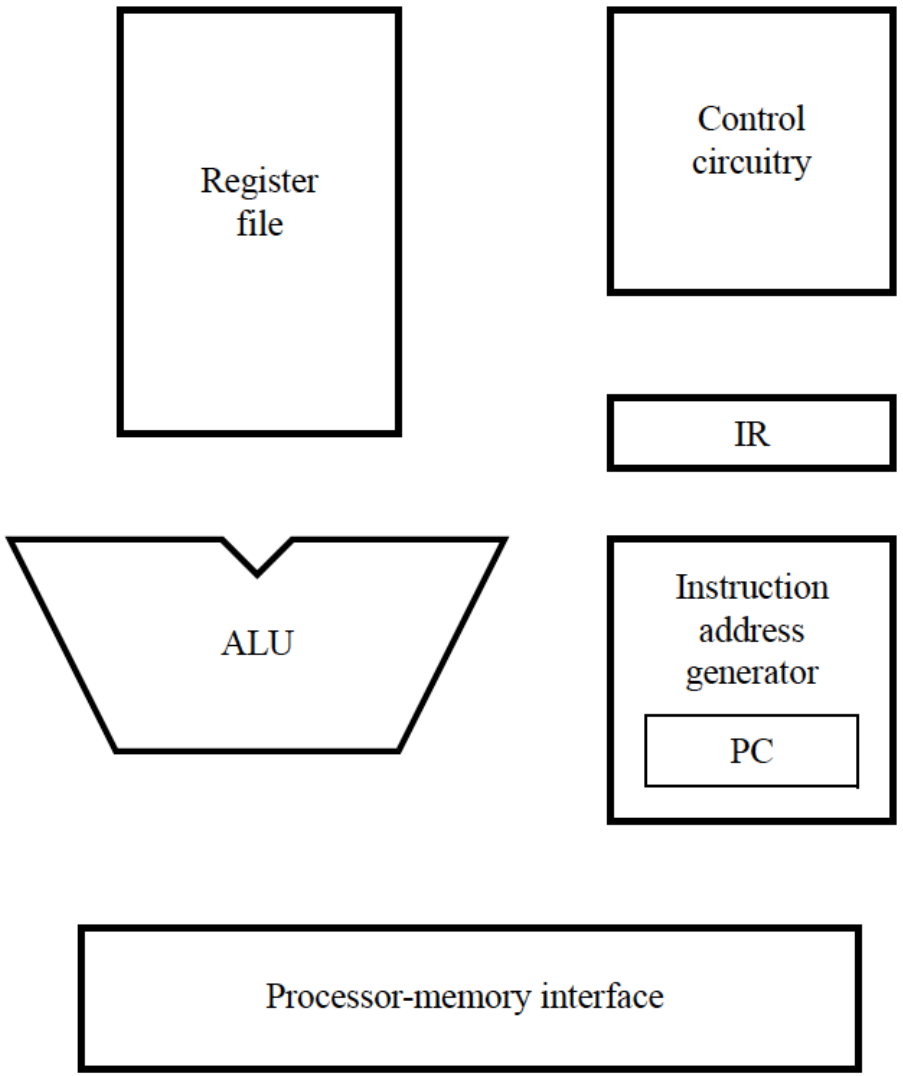
Instruction Execution

- To execute an instruction, the processor has to perform the following steps: (ctd.)
  - 2. Increment the PC to point to the next instruction. PC←[PC] + 4
    - Assuming that the memory is byte addressable, and each instruction comprises 4 bytes.
  - 3. Carry out the operation specified by the instruction in the IR.
  - (1), (2): fetch phase, (3): execution phase
  - Note
    - If an instruction occupies more than one memory word, steps1 and 2 must be repeated as many times as necessary to fetch the complete instruction.



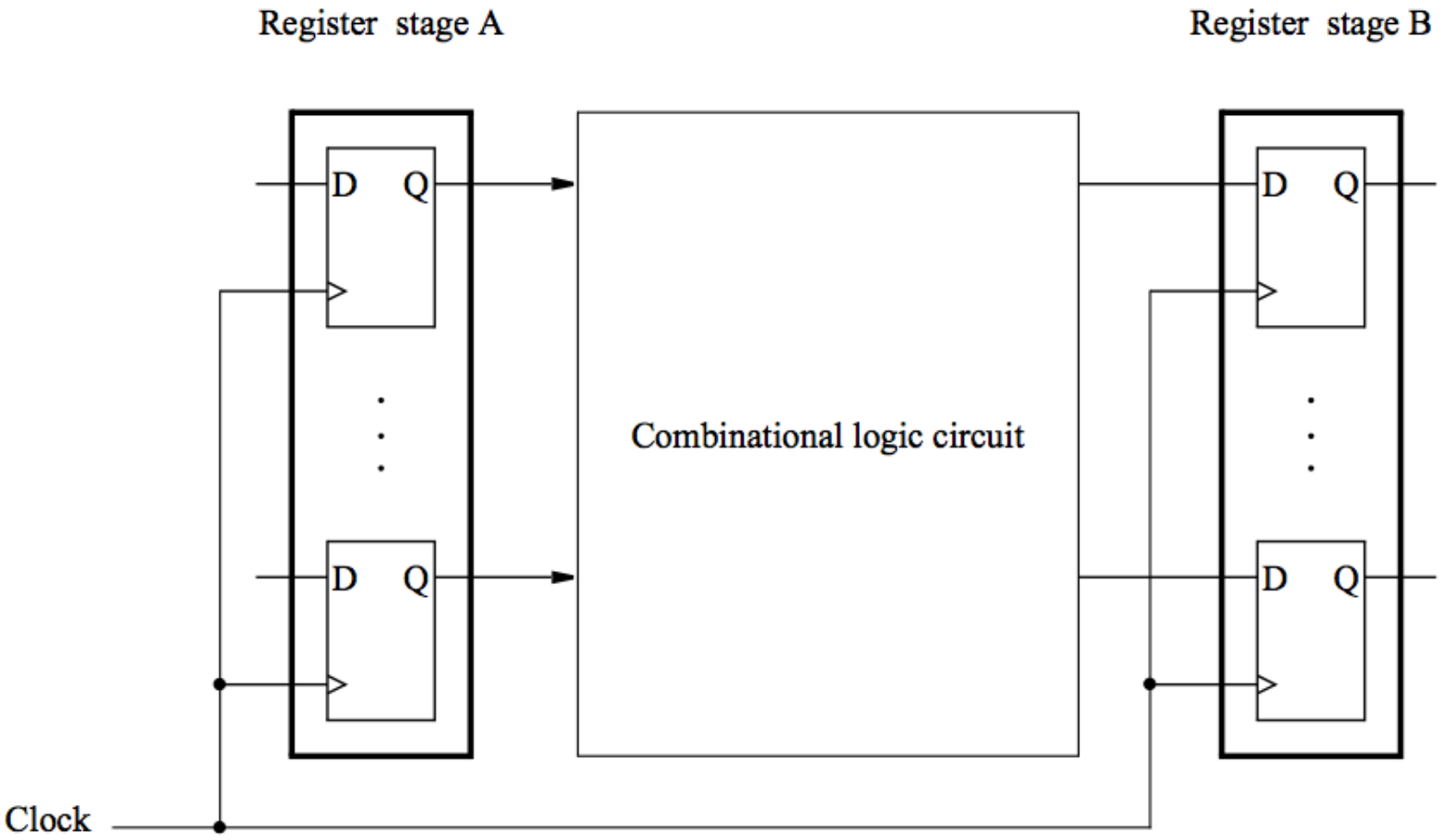
- Operations of Instruction Execution
  - With few exceptions, an instruction can be executed by performing one or more of the following operations in some specified sequence:
    - Read the contents of a given memory location and load them into a processor register.
    - Read data from one or more processor registers.
    - Perform an arithmetic or a logic operation and place the result in a processor register.
    - Store data from a processor register into a given memory location.

- Main Hardware components of Processor
  - PC provides instruction address
  - Instruction is fetched into IR
  - Instruction address generator updates PC
  - Control circuitry interpret instruction and generate control signals to perform the actions needed.



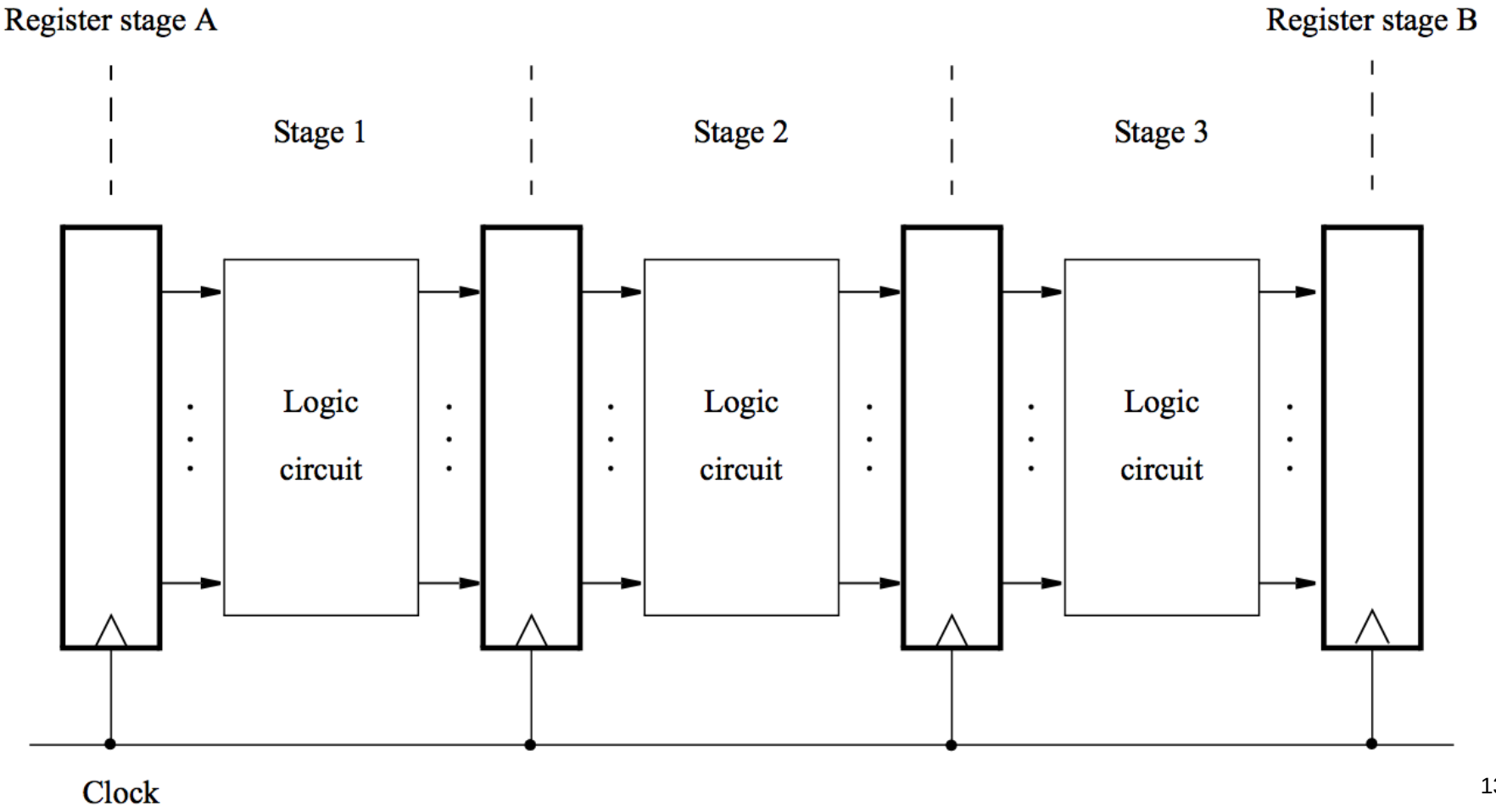
Structure of Data Processing System

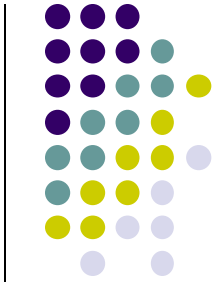
Contents of register A are processed and deposited in register B.



Structure of Data Processing System

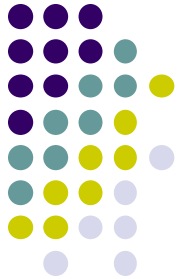
- The operation performed by the combinational block can often be broken down into several simpler steps, where each step is performed by a subcircuit of the original circuit.



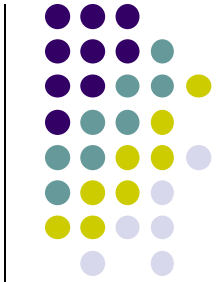


- Structure of Data Processing System
  - Why multi-stage?
    - Processing moves from one stage to the next in each clock cycle.
    - Such a multi-stage system is known as a pipeline.
    - High-performance processors have a pipelined organization.
    - Pipelining enables the execution of successive instructions to be overlapped.
    - Pipelining will be discussed in Chapter 6.

5.2 Instruction Execution



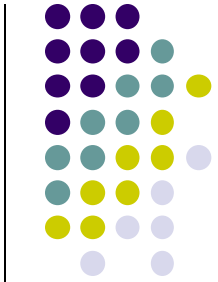
- Pipelined organization is most effective if all instructions can be executed in the same number of steps.
- Each step is carried out in a separate hardware stage.
- Processor design will be illustrated using five hardware stages.
- How can instruction execution be divided into five steps?



- A memory access instruction:

  Load R5, X(R7)

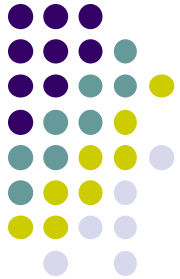
  - Load a word of data from memory location X+[R7] into R5 using Index addressing mode.
  - Five steps required for execution of this instruction
    1. Fetch the instruction and increment the program counter.
    2. Decode the instruction and read the contents of register R7 in the register file.
    3. Compute the effective address X + [R7].
    4. Read the memory source operand.
    5. Load the operand into the destination register, R5.



- A computational instruction:
  Add R3, R4, R5
  - The Add instruction does not require access to an operand in the memory, so it can be completed in four steps.
  - It is advantageous to use the same multi-stage processing hardware for all instructions. We can add a step in which no action takes place.
    1. Fetch the instruction and increment the program counter.
    2. Decode the instruction and read register R4 and R5.
    3. Compute the sum [R4] + [R5].
    4. No action.
    5. Load the result into the destination register, R3.

- A memory access instruction:
  Store R6, X(R8)
  - The instruction does not require access to the final step of loading the result into a destination register.
    1. Fetch the instruction and increment the program counter.
    2. Decode the instruction and read register R6 and R8.
    3. Compute the effective address X + [R8].
    4. Store the contents of register R6 into memory location X + [R8].
    5. No action.

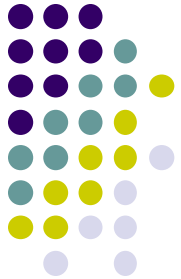
- The following five-step sequence of actions is suitable for all instructions in a RISC-style instruction set.



Step	Action
1	Fetch an instruction and increment the program counter.
2	Decode the instruction and read registers from the register file.
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, if needed.

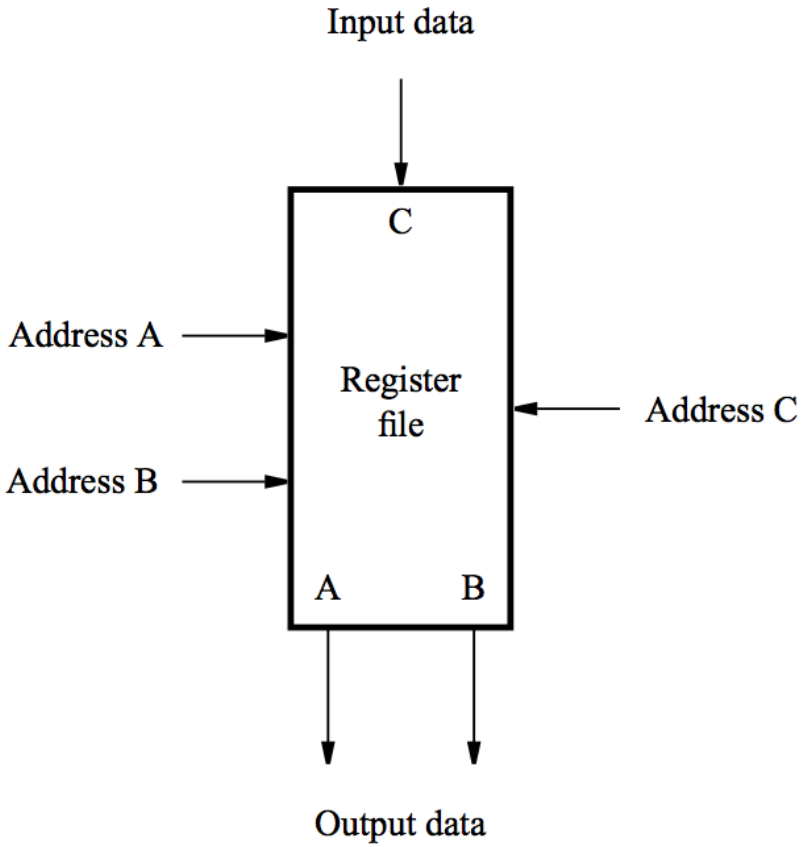
Figure 5.4 A five-step sequence of actions to fetch and execute an instruction.

5.3 Hardware Components



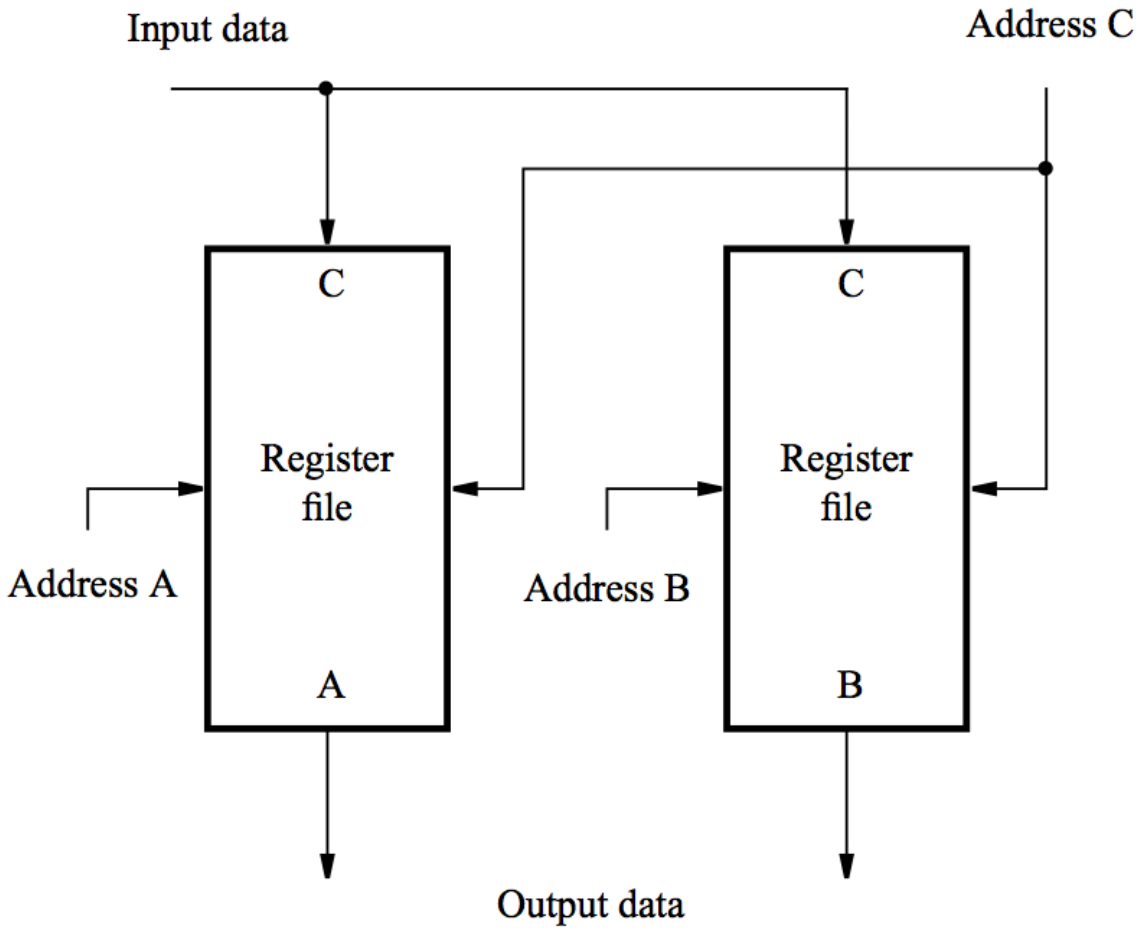
- Consider the components in Figure 5.1.
- How these components may be organized in the multi-stage structure of Figure 5.3.

- Register file
  - A 2-port register file is needed to read the two source registers at the same time.
  - Two separate outputs (A and B), and two address inputs to select the two registers to be read. They are connected to the fields in the IR that specify the source registers.
  - Also has a data input, C, and a corresponding address input to select the destination register, which is connected to the IR field that specifies the destination register.



Alternative implementation of 2-port Register file

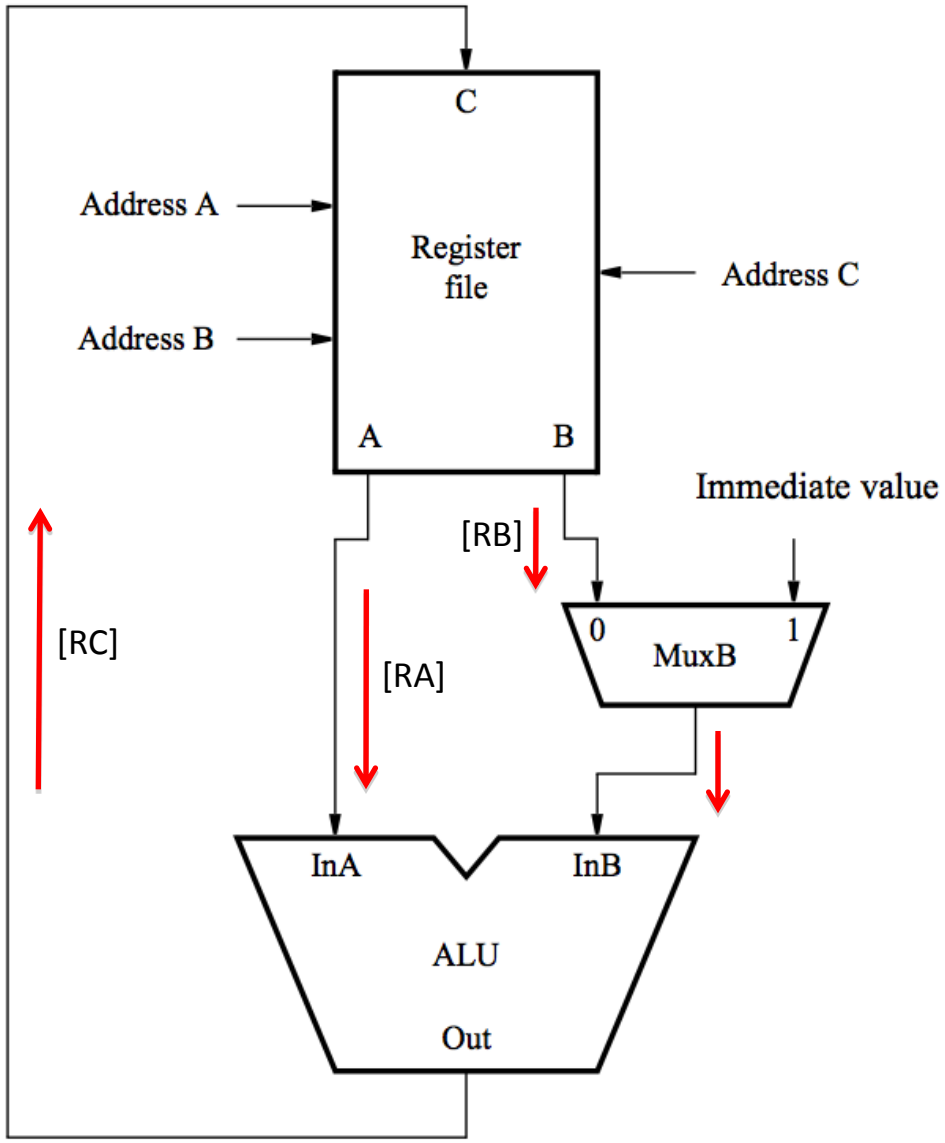
- Using two single-ported memory blocks



ALU

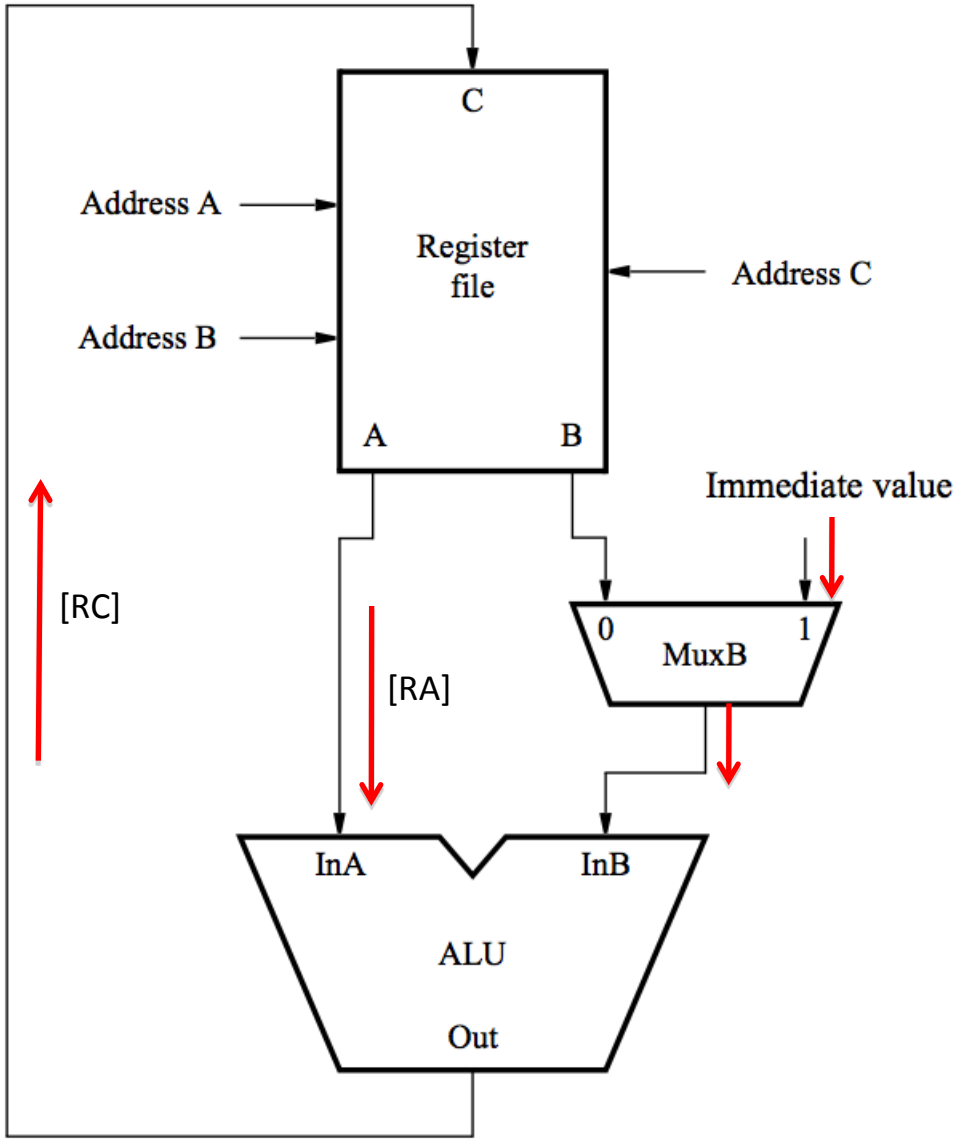
- ALU is used to manipulate data. It performs arithmetic operations (addition and subtraction) and logic operations (AND, OR, XOR).

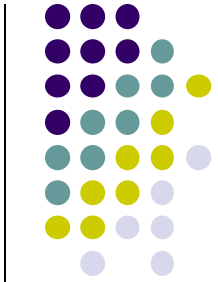
- Both source operands and the destination location are in the register file.



- ALU

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



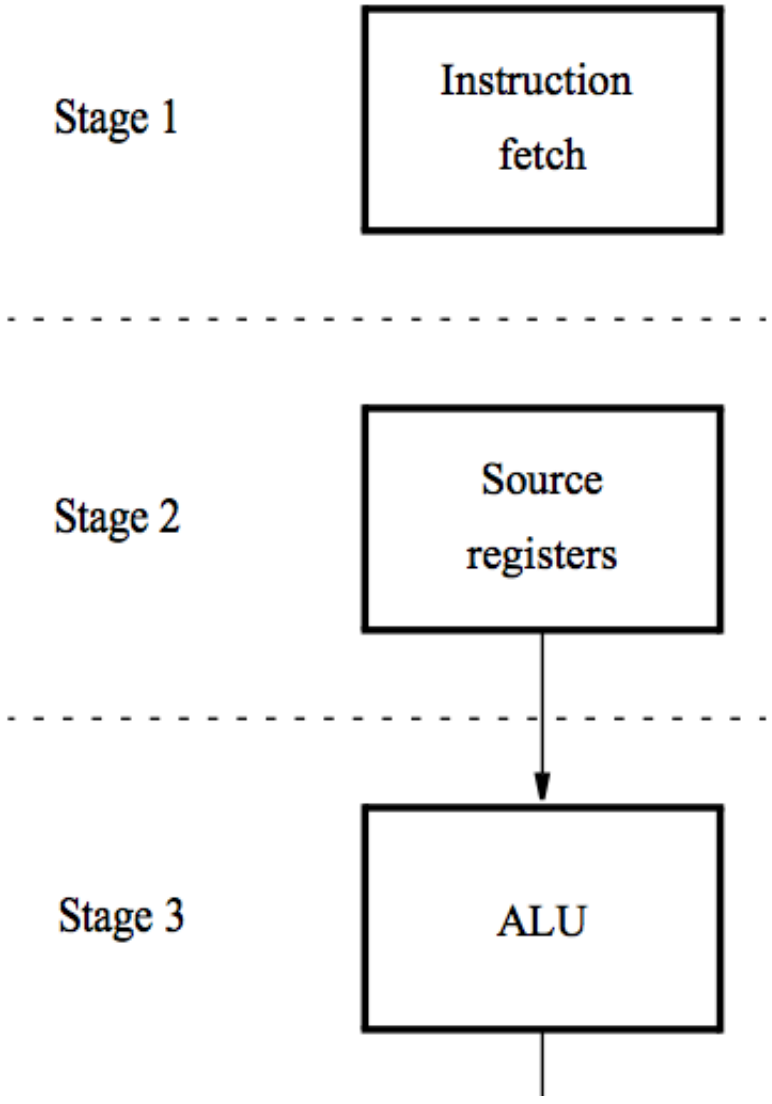


Datapath

- Instruction processing consists of two phases:
  - The fetch phase
  - The execution phase
- Divide the processor hardware into two corresponding sections.
  - The section that fetches instructions is also responsible for decoding them and for generating the control signals that cause appropriate actions to take place in the execution section
  - The execution section reads the data operands specified in an instruction, performs the required computations, and stores the results.

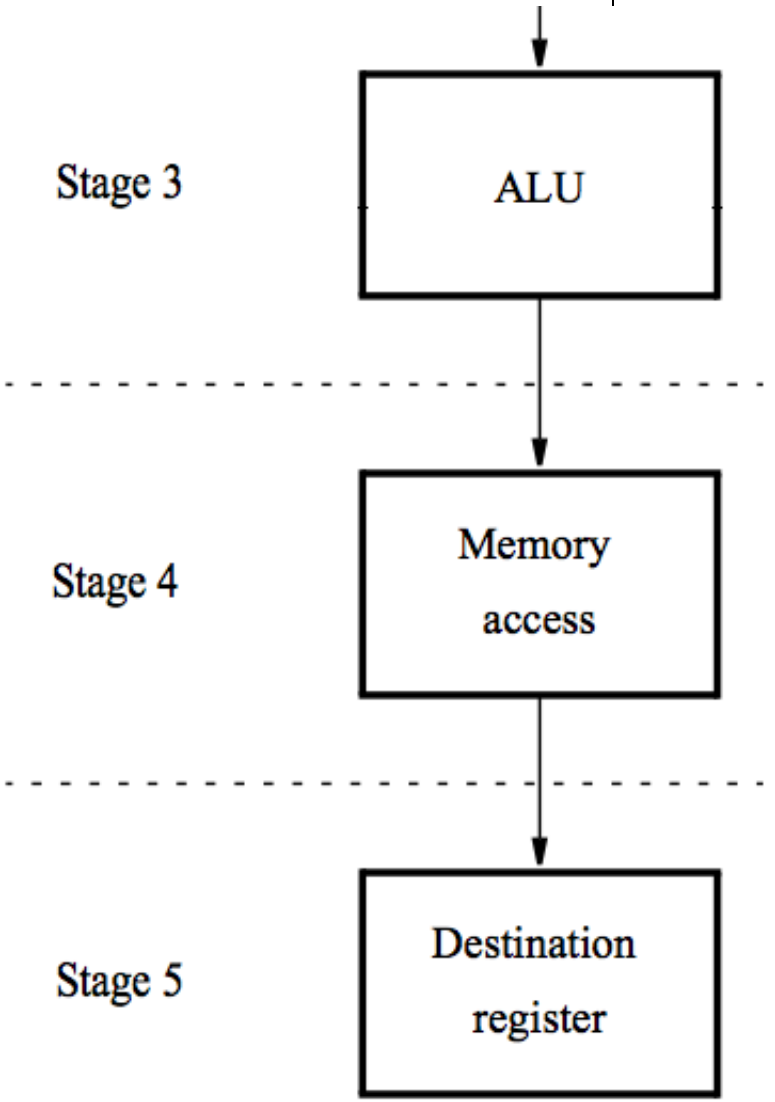
A 5-stage implementation of a RISC processor

- Instruction processing moves from each stage to the next in every clock cycle.
- An instruction is fetched and placed into IR in stage 1.
- 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.



Datapath (stages 2 to 5)

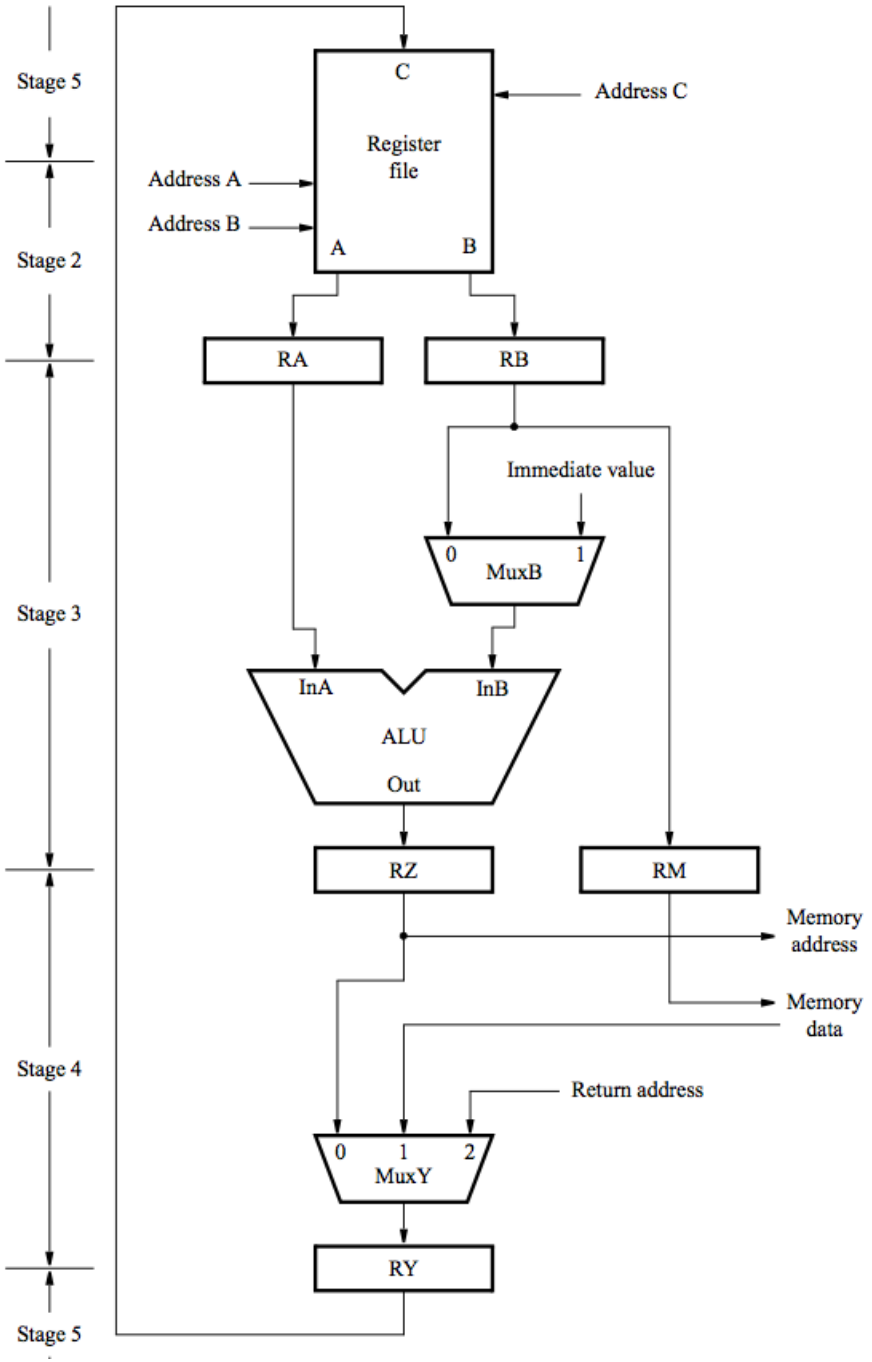
- Register file

  Inter-stage registers needed to carry data from one stage to the next.

- ALU stage

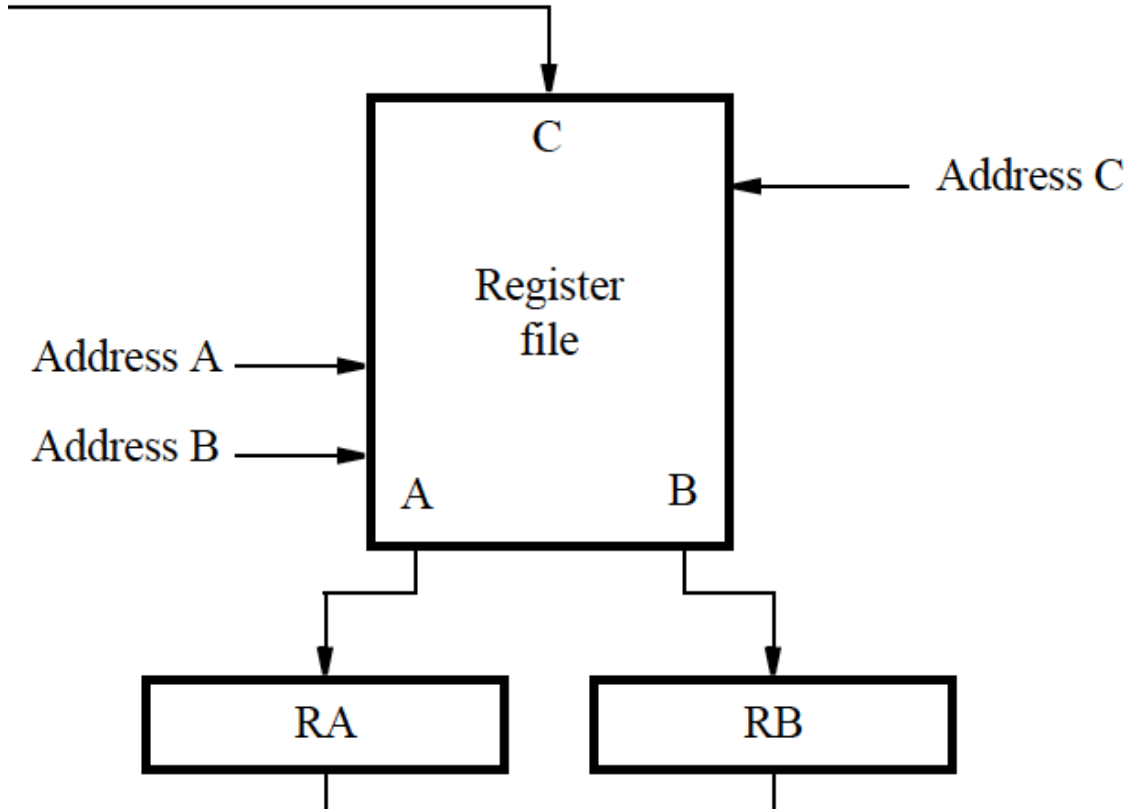
- Memory stage

- Back to register file



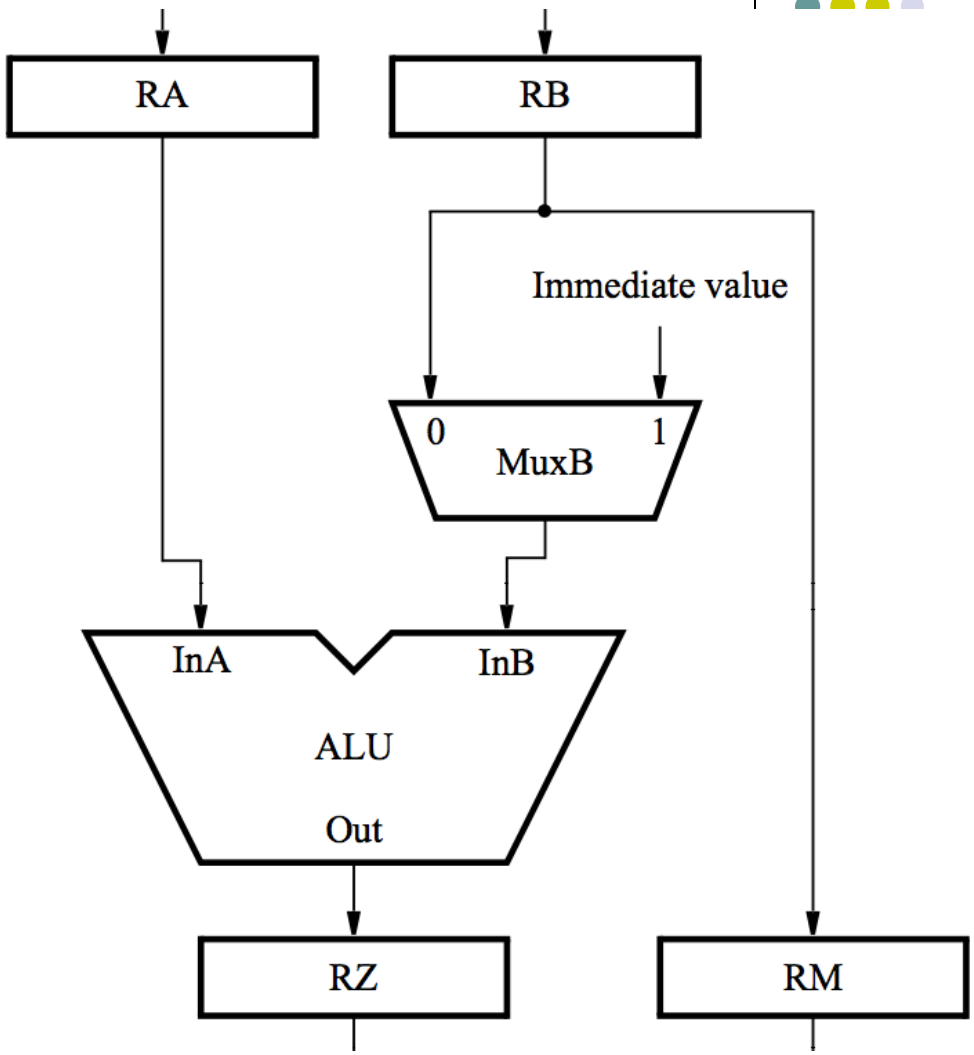
Datapath: Register file (stages 2 & 5)

- Address inputs connected to corresponding fields in IR.
- Source registers are read and their contents stored in RA and RB.
- The result of the instruction is stored in the destination register selected by address.



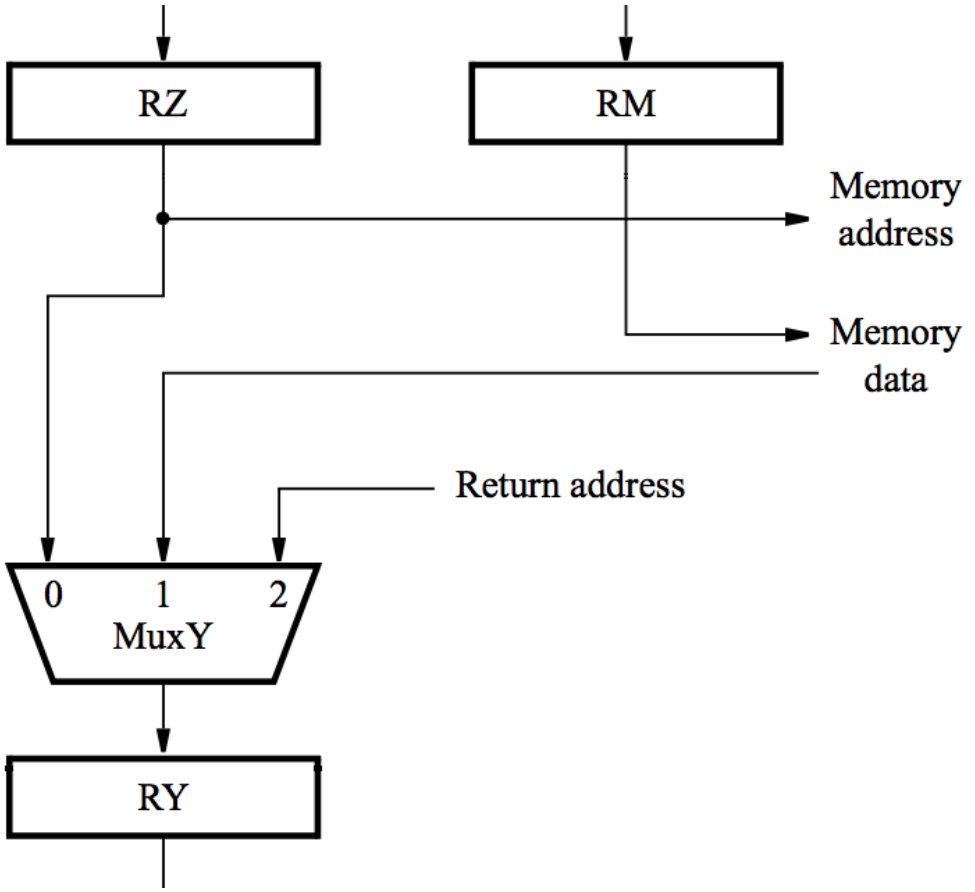
Datapath: ALU stage

- 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.



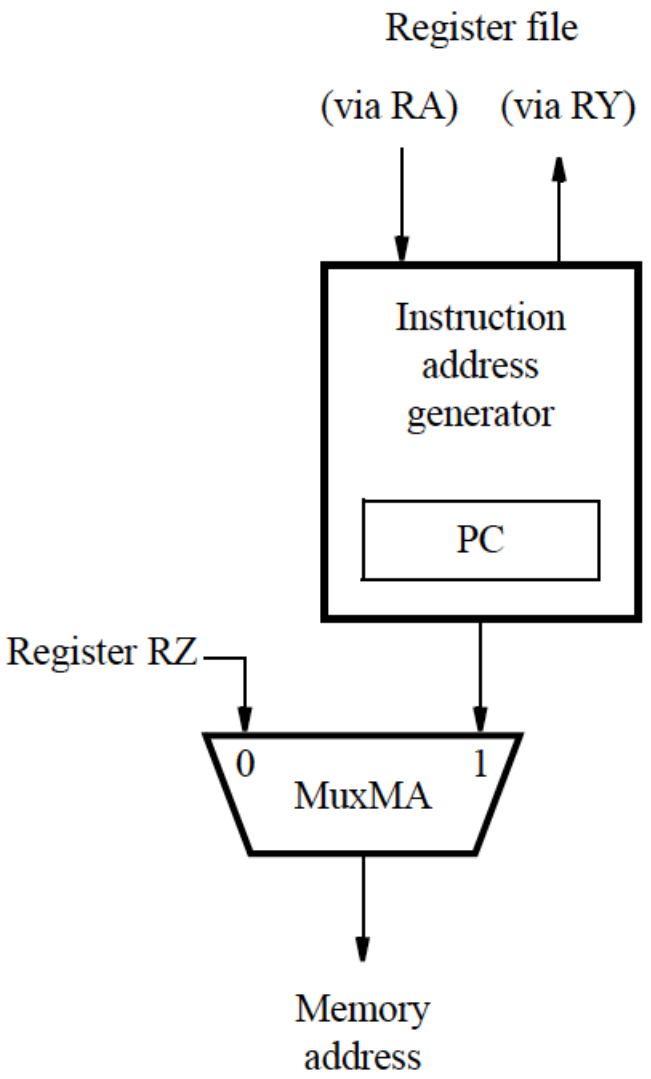
Datapath: Memory stage

- For a memory instruction, RZ provides memory address, and MuxY selects data read 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.



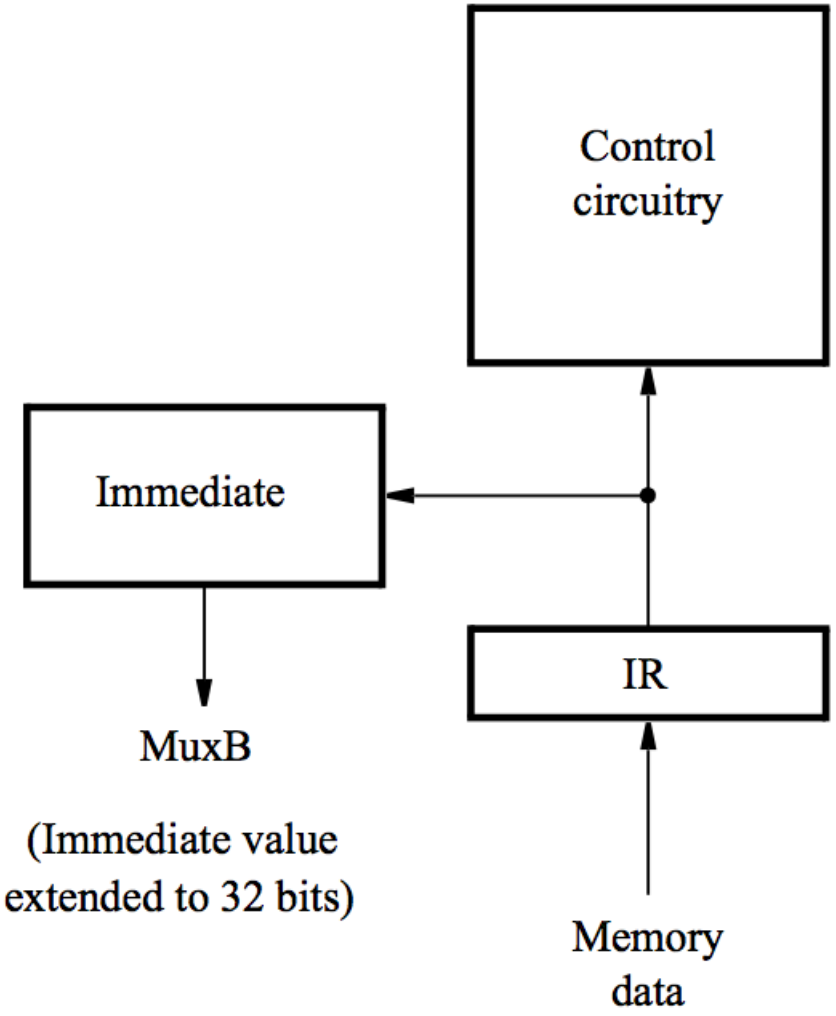
Instruction fetch section

- 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 or writing data operands.

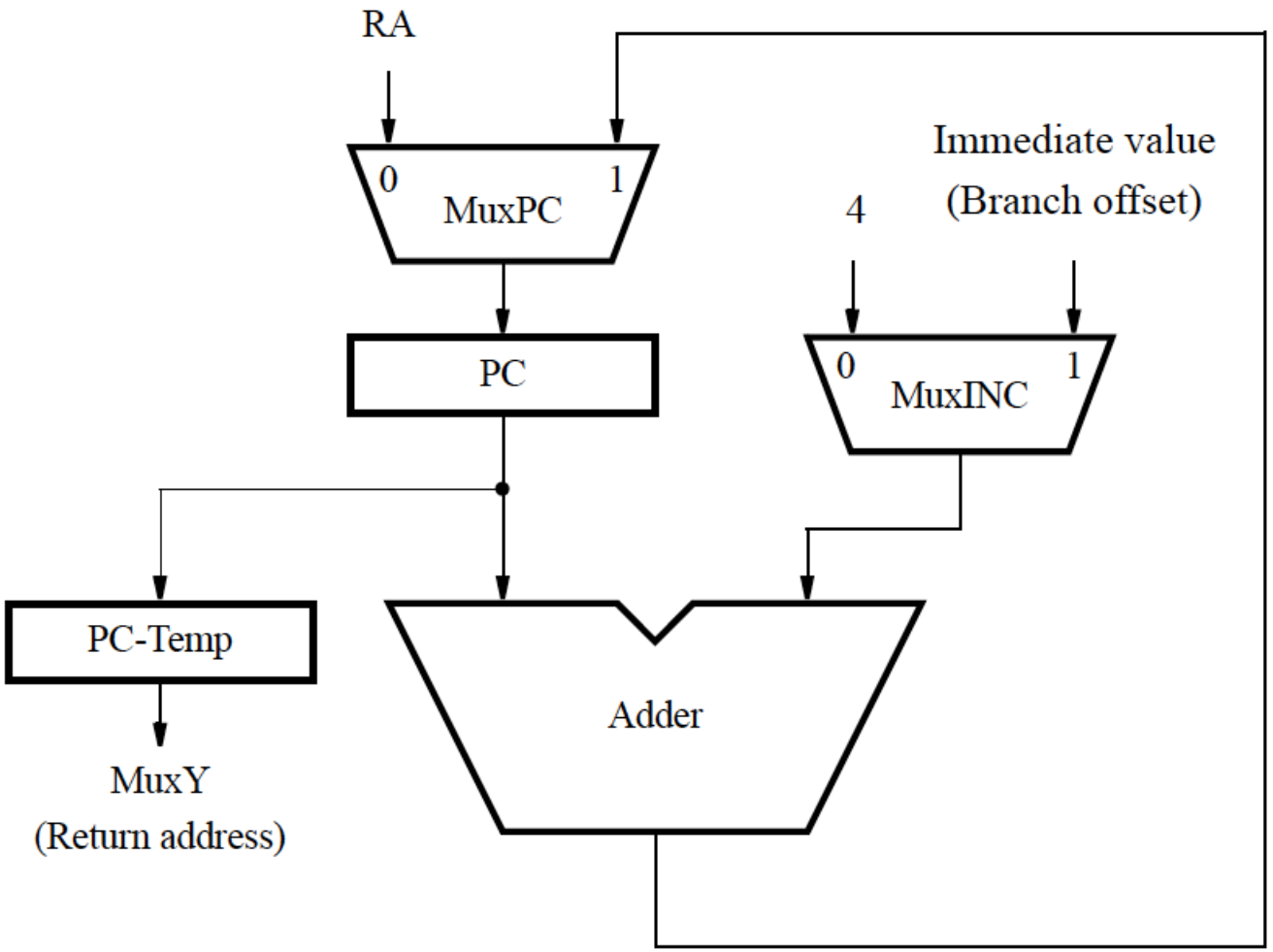


Instruction fetch section

- 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 as specified in the instruction.



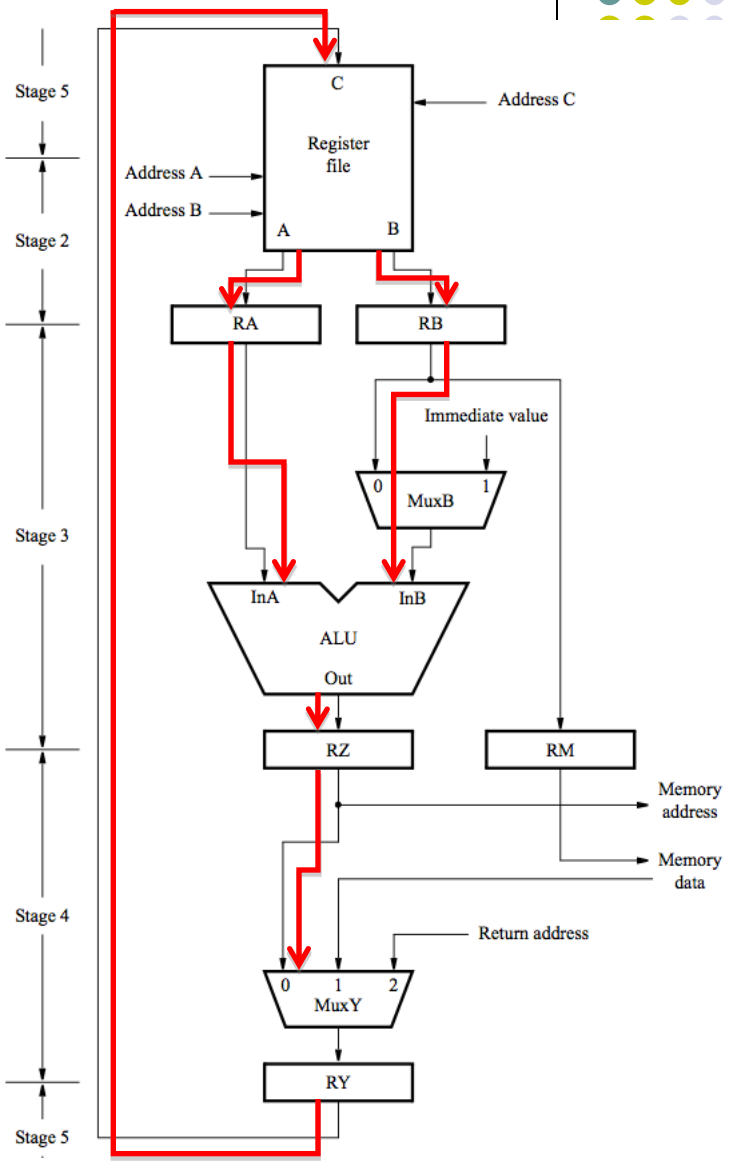
- Instruction fetch section
  - Instruction address generator
    - Connections to RY and RA are used for subroutine call and return.

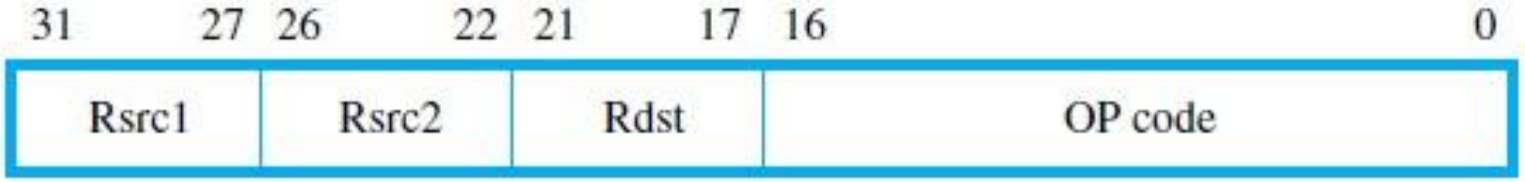


5.4 Instruction Fetch and Execution Steps

- Example1: Add R3, R4, R5

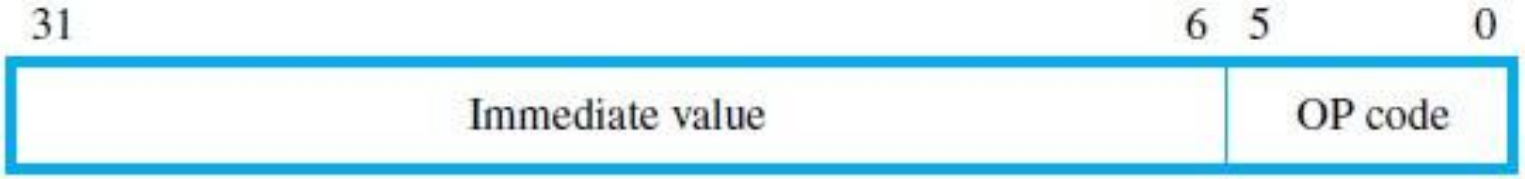
1. Memory address ← [PC], Read memory, IR ←Memory data, PC ← [PC] + 4
2. Decode instruction, RA ← [R4], RB ← [R5]
3. RZ ← [RA] + [RB]
4. RY ← [RZ]
5. R3 ← [RY]





(a) Register-operand format

(b) Immediate-operand format

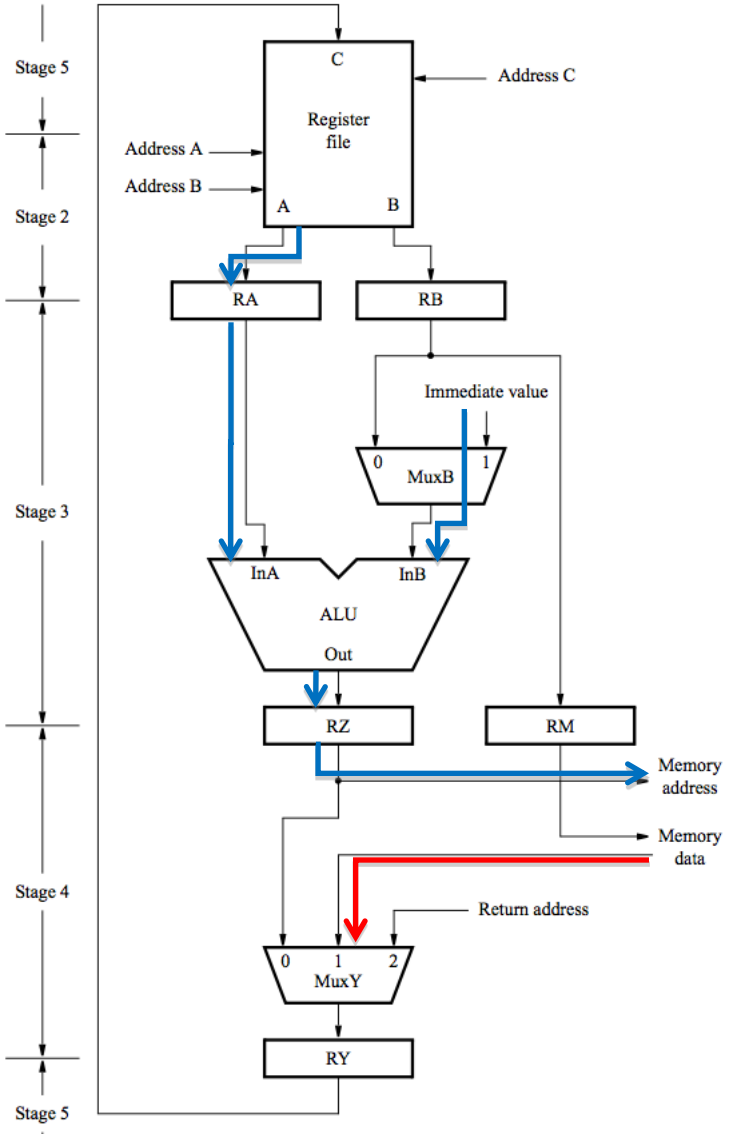


(c) Call format

Figure 5.12 Instruction encoding.

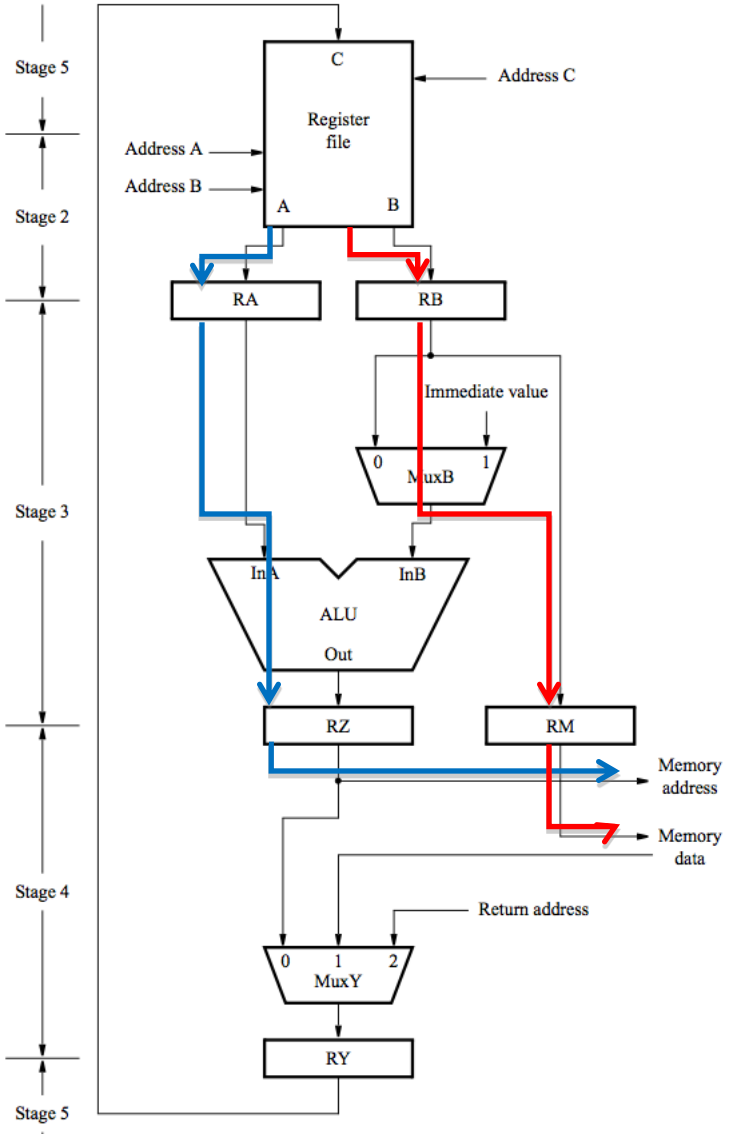
- Example2: Load R5, X(R7)

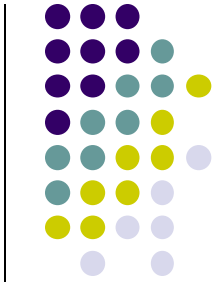
1. Memory address ← [PC], Read memory, IR ←Memory data, PC ← [PC] + 4
2. Decode instruction, RA ← [R7]
3. RZ ← [RA] + Immediate value X
4. Memory address ←[RZ], Read memory, RY ← Memory data
5. R5 ← [RY]



- Example3: Store R6, X(R8)

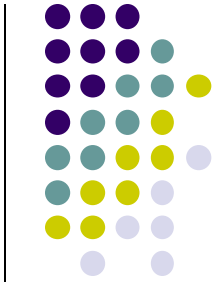
1. Memory address ← [PC], Read memory, IR ←Memory data, PC ← [PC] + 4
2. Decode instruction, RA ← [R8], RB ← [R6]
3. RZ ← [RA] + Immediate value X, RM ←[RB]
4. Memory address ←[RZ], Memory data ← [RM], Write memory
5. No action





- Unconditional Branch

1. Memory address ← [PC], Read memory, IR ←Memory data, PC ← [PC] + 4
2. Decode instruction
3. PC ← [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

The comparison can be done by performing [RA] – [RB], but it is time consuming.

We can use a simpler and faster comparator.

- 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]

Wait for memory

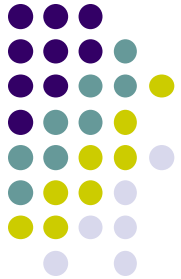
- Most of the time, the instruction or data referenced in memory Read and Write operations are found in the cache.
  - The operation is completed in one clock cycle.
- When the requested information is not in the cache and has to be fetched from the main memory, several clock cycles may be needed.
- The processor-memory interface will generate a Memory Function Completed (MFC) signal, which is asserted when the requested memory Read or Write operation has been completed.
  - When the requested data are in the cache, the MFC signal is generated before the end of the same clock cycle in which the memory request is issued (the step is completed in one clock cycle).
  - If access to the main memory is required, the MFC response is delayed until the operation is completed (the step is extended to several clock cycles).

- Wait for memory
  - Step 1 of any instruction includes a Wait for MFC command:

    Memory address ← [PC], Read memory, Wait for MFC,

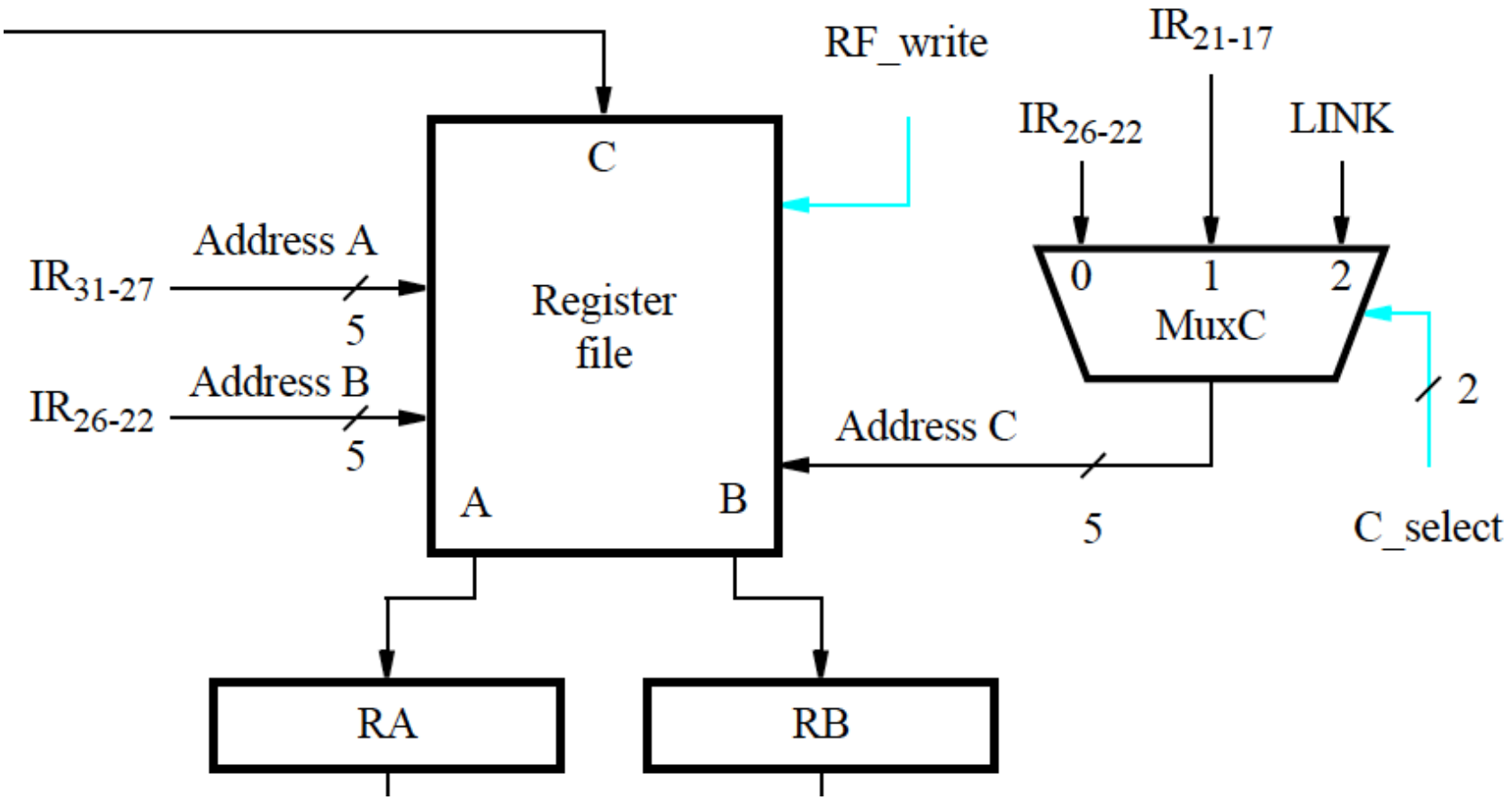
    IR ←Memory data, PC ← [PC] + 4
  - The Wait for MFC command is also needed in step 4 of Load and Store instructions.

5.5 Control Signals

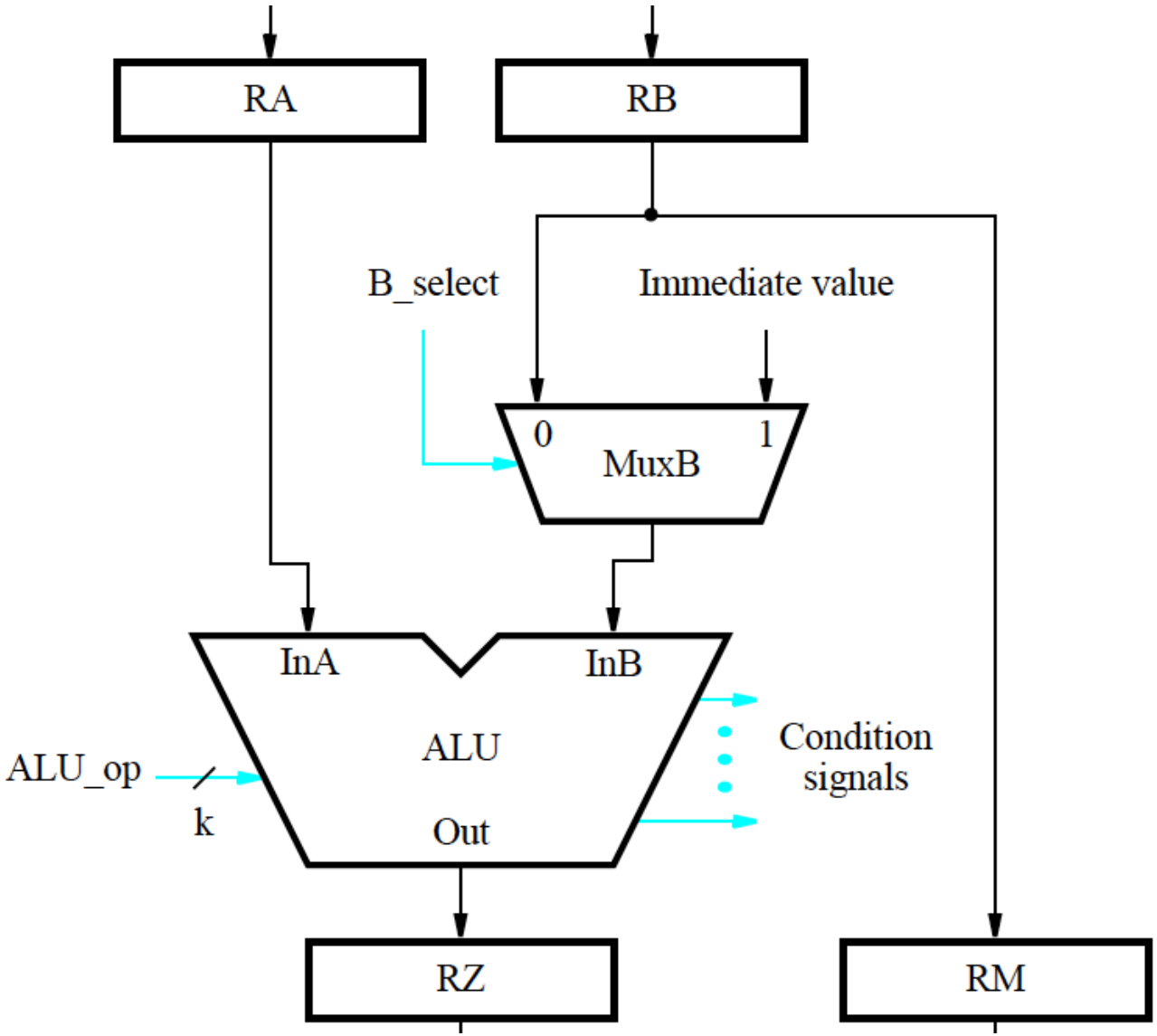


- The operation of the processor's hardware components is governed by control signals.
  - Control multiplexer selection to guide the flow of data.
  - Set the function performed by the ALU.
  - Determine when data are written into the PC, the IR, the register file and the memory.
- Inter-stage registers are always enabled.

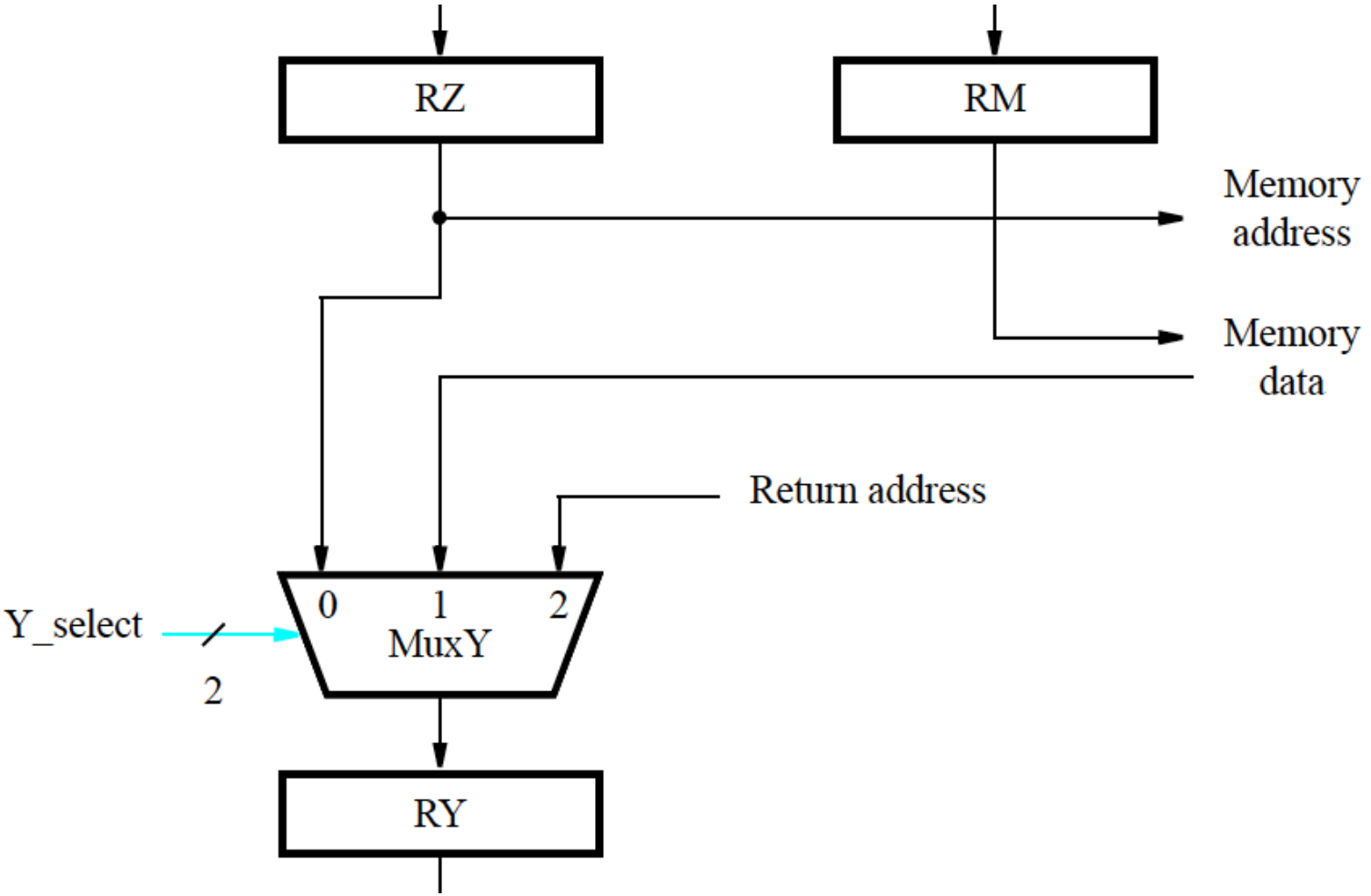
- Register file control signals

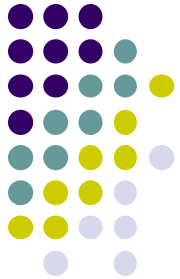


- ALU control signals



- Result selection

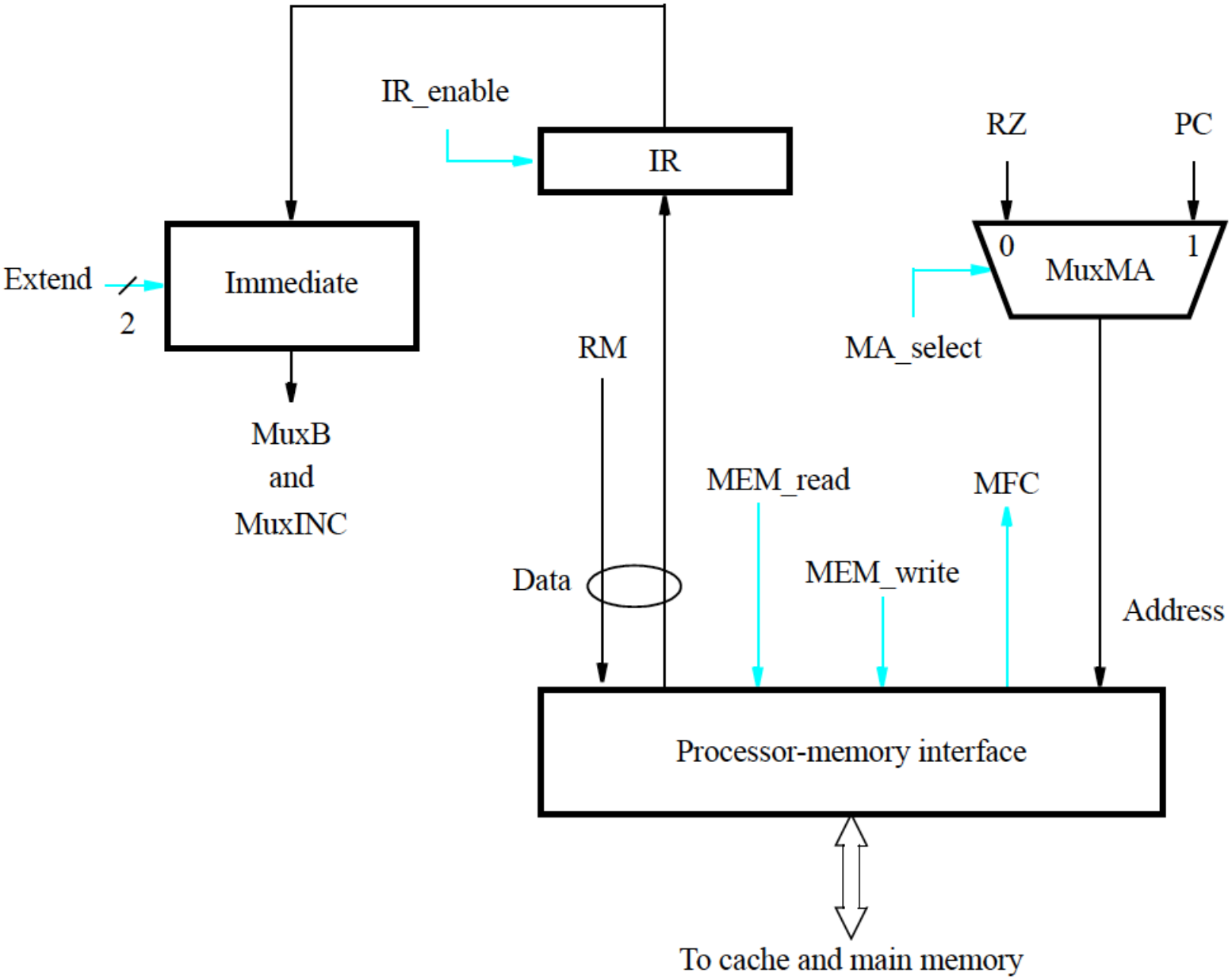




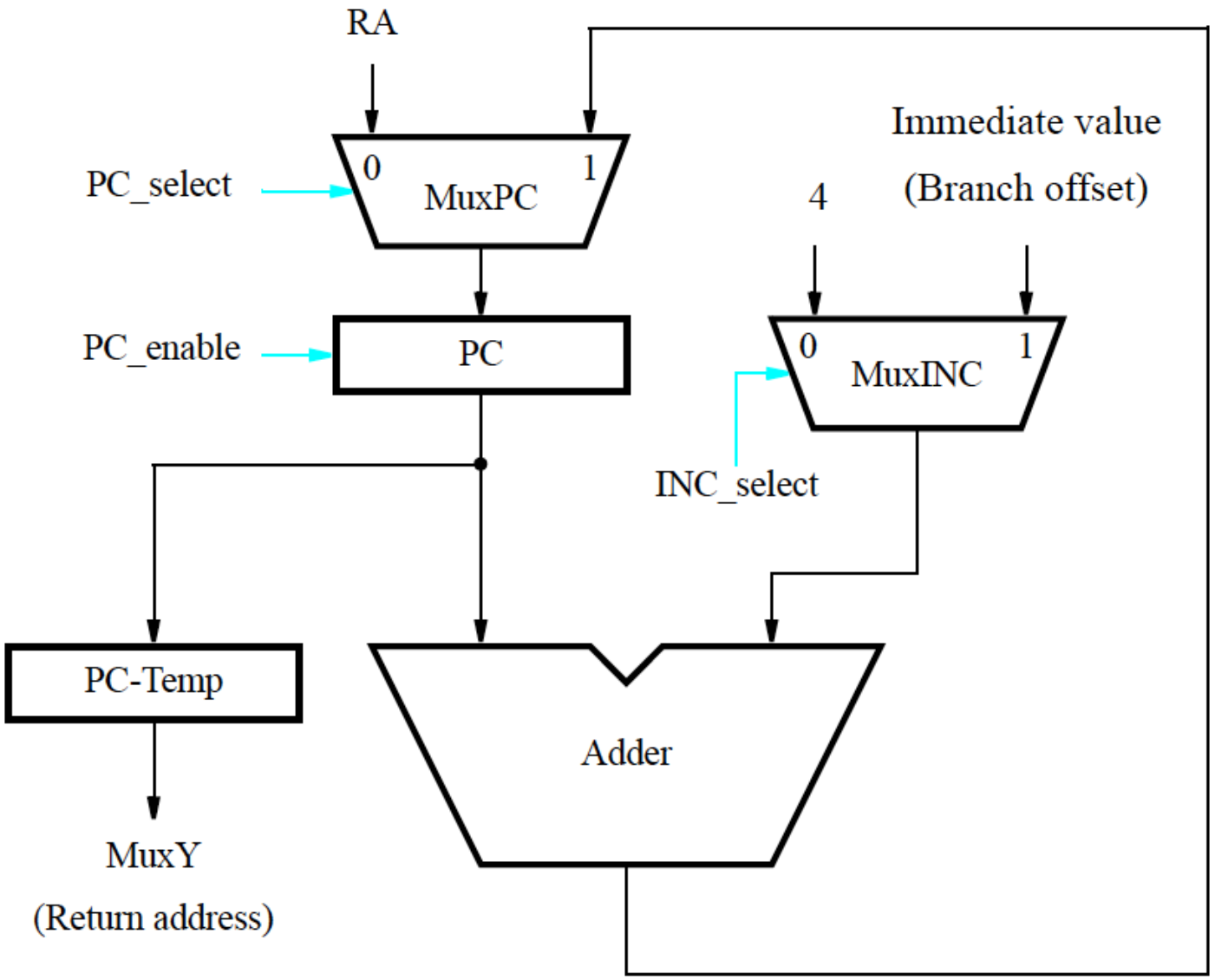
- Memory access
  - Cache memory described earlier as faster and smaller storage that is an adjunct to the larger and slower main memory.
  - When data are found in the cache, access to memory can be completed in one clock cycle.
  - Otherwise, read and write operations may require several clock cycles 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 ← [PC], Read memory, Wait for MFC, IR ← Memory data, PC ←[PC] + 4

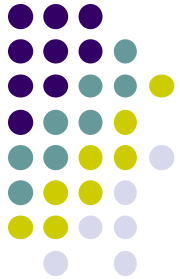
- Memory and IR control signals



- Control signals of instruction address generator



5.6 Hardwired Control

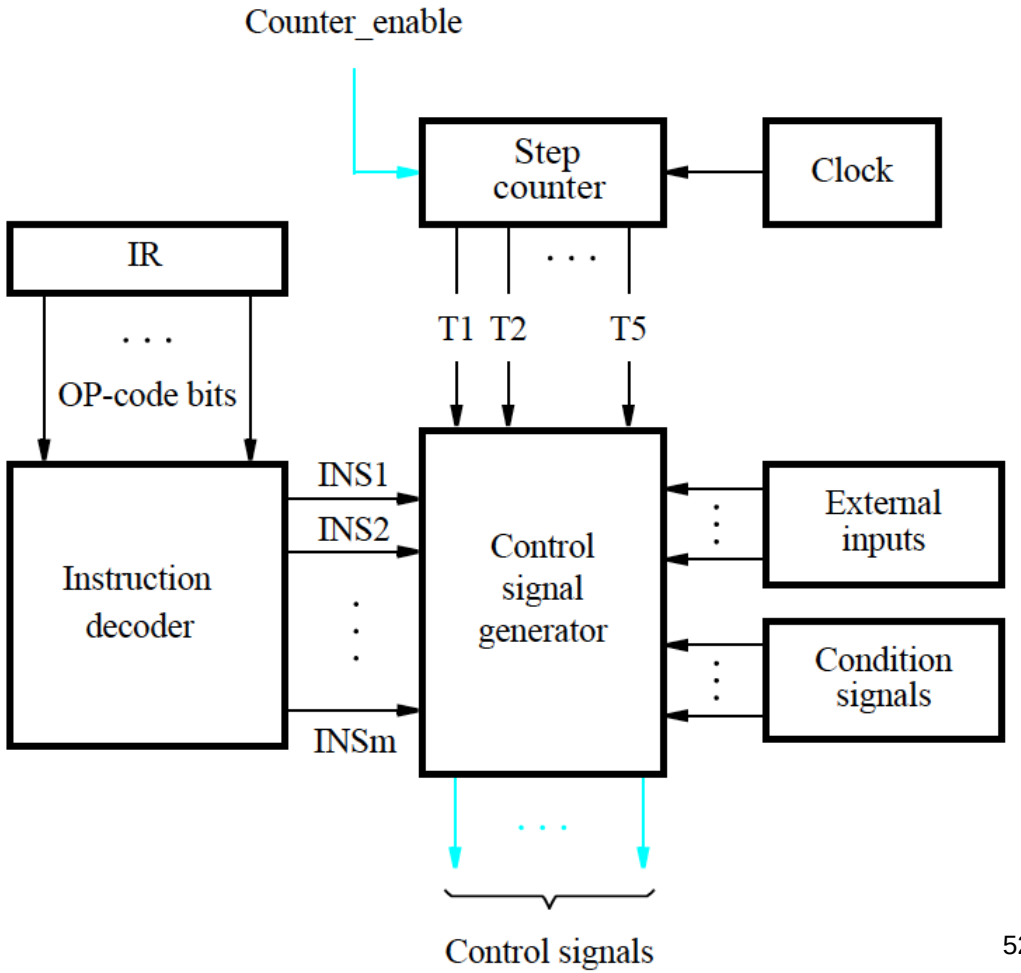


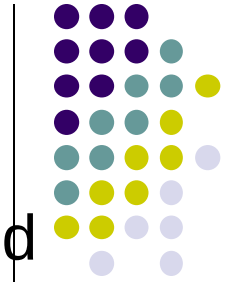
- We have described:
  - Actions to fetch & execute instructions
  - The necessary control signals
- How to generate control signals that cause the actions to take place in correct sequence and at correct time?
- There are two basic approaches:
  - Hardwired control
  - Microprogramming

Hardwired generation of control signal

- Hardwired control involves implementing circuitry that considers step counter, IR, ALU result, and external inputs (such as MFC, interrupt requests).

- Step counter keeps track of execution progress, one clock cycle for each of the five steps described earlier.





Hardwired generation of control signal

- The instruction decoder interprets the OP-code and addressing mode information in the IR and sets to 1 the corresponding INSi output.
- During each clock cycle, one of the outputs T1 to T5 of the step counter is set to 1 to indicate which of the five steps is being carried out.
- The control signal generator is a combinational circuit that produces the necessary control signals based on all its inputs.
- Function of Hardwired Control
  - Inputs: INS_m, T_n, E_k, F_j
  - Output: $C_i = f(INS_m, T_n, E_k, F_j)$

- Hardwired generation of control signal
  - Consider step 1 in the instruction execution process:
    - A new instruction is fetched. It is identified by signal T1 being asserted.
    - MA_select in Figure 5.19 is set to 1 to select the PC as the source of the memory address.
    - MEM_read is activated to initiate a memory Read operation.
    - The instruction is loaded into the IR by activating IR_enable when MFC is asserted.
    - The PC is incremented by 4, by setting the INC_select in Figure 5.20 to 0 and PC_select to 1.
    - The PC_enable is activated to load the new value into the PC at the positive edge of the clock marking of the end of step T1.

Datapath control signals

- The desired setting of various control signals can be determined by examining the actions taken in each execution step of every instruction.
- For example, RF_write signal is set to 1 in step T5. It may be generated by the logic expression:

$$RF\_write = T5 \cdot (ALU + Load + Call)$$

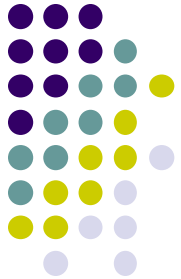
  - RF_write signal is a function of both the instruction and the timing signals.
- The setting of some of the multiplexers need not change from one timing step to another. For example,

$$B\_select = Immediate$$

  - Immediate stands for all instructions that use an immediate value in the IR.

Dealing with Memory Delay

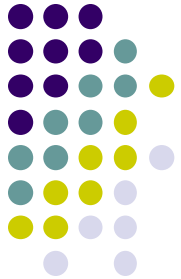
- The timing signals T1 to T5 are asserted in sequence as the step counter is advanced.
- Most of the time, the step counter is incremented at the end of every clock cycle.
- However, a step in which a MEM_read or a MEM_write command is issued does not end until the MFC signal is asserted.
- Disable the step counter to extend the duration of an execution step to more than one clock cycle.
  - Counter_enable = $\overline{WMFC}$ + MFC
  - Counter_enable is set to 1 when WMFC is not asserted or MFC is asserted.



Dealing with Memory Delay

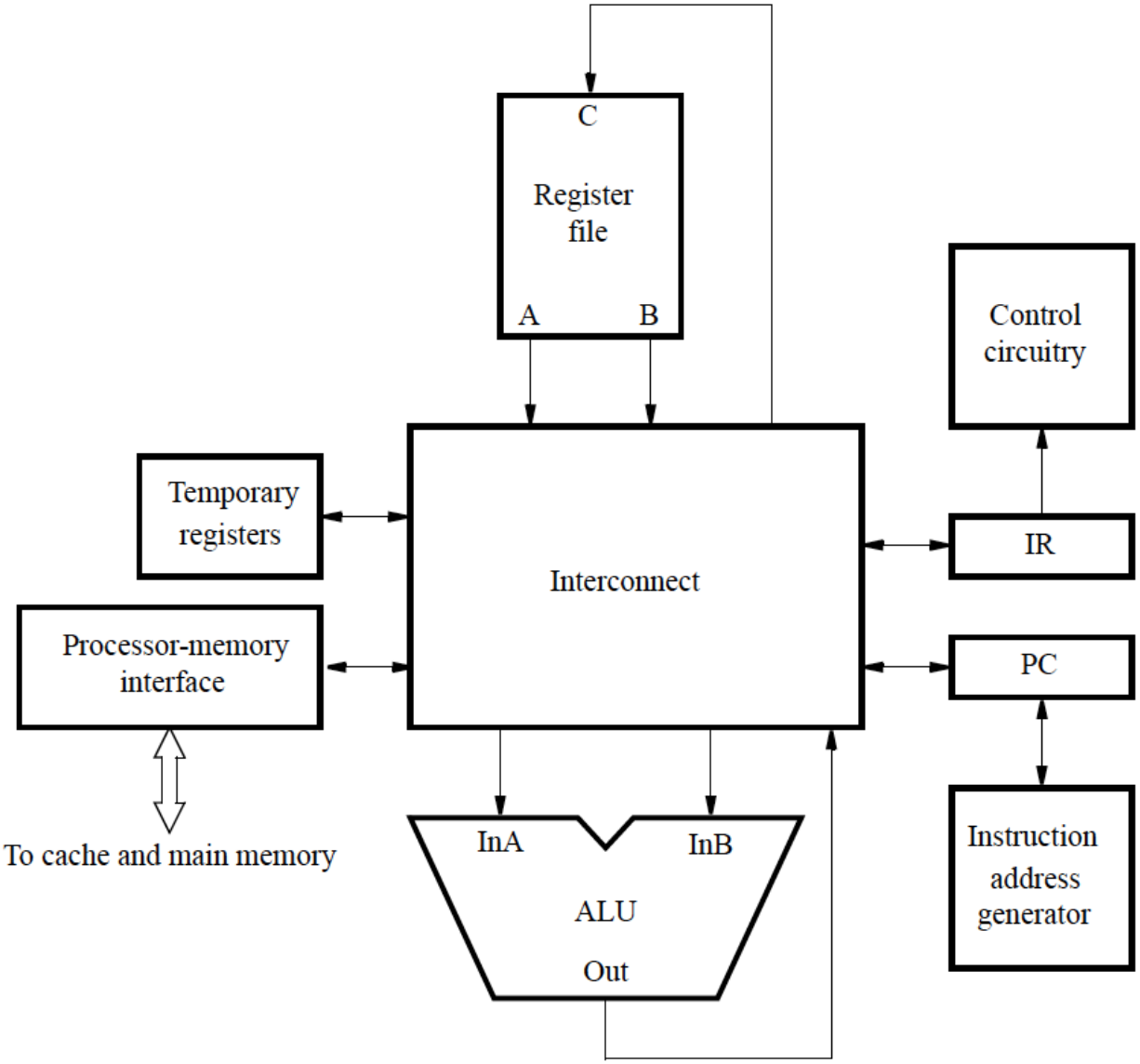
- In addition, we must ensure that the PC is incremented only once when an execution step is extended for more than one clock cycle.
- When fetching an instruction, the PC should be enabled only when MFC is received.
- PC is also enabled in step 3 of instructions that cause branching (BR).
  - PC_enable = T1 · MFC + T3 · BR

5.7 CISC-style processors

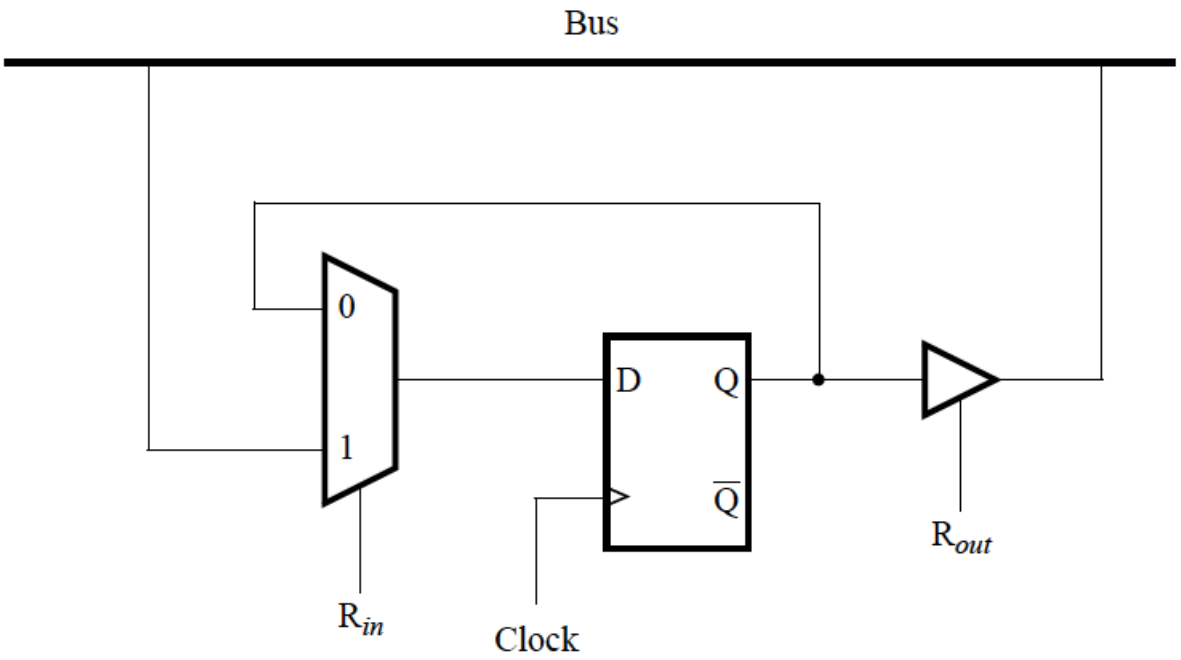


- CISC-style processors have more complex instructions.
- The full collection of instructions cannot all be implemented in a fixed number of steps.
- Execution steps for different instructions do not all follow a prescribed sequence of actions.
- Hardware organization should therefore enable a flexible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

- Hardware organization for a CISC computer

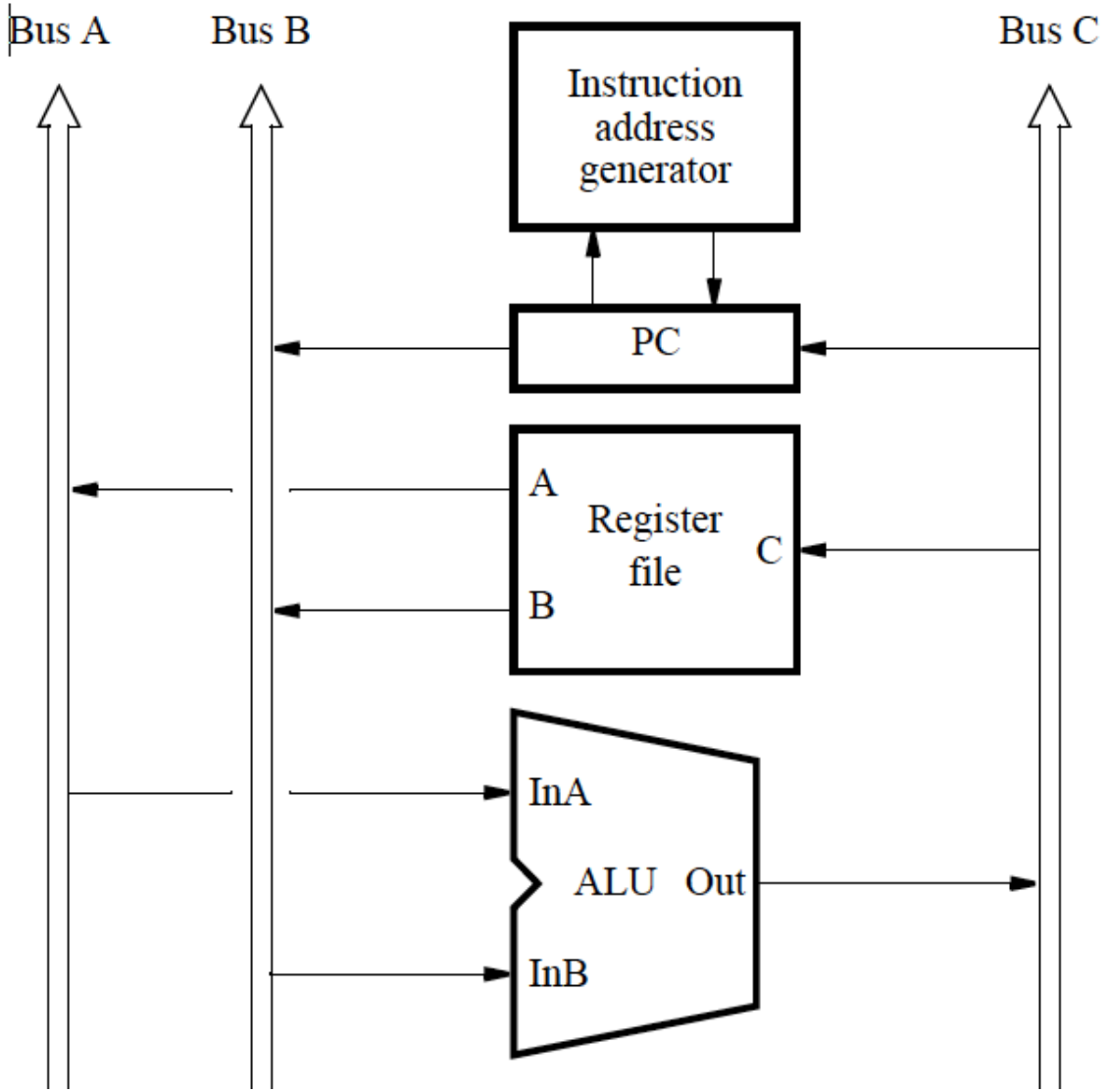


- Bus
  - An example of an interconnection network.
  - When functional units are connected to a common bus, tri-state drivers are needed.

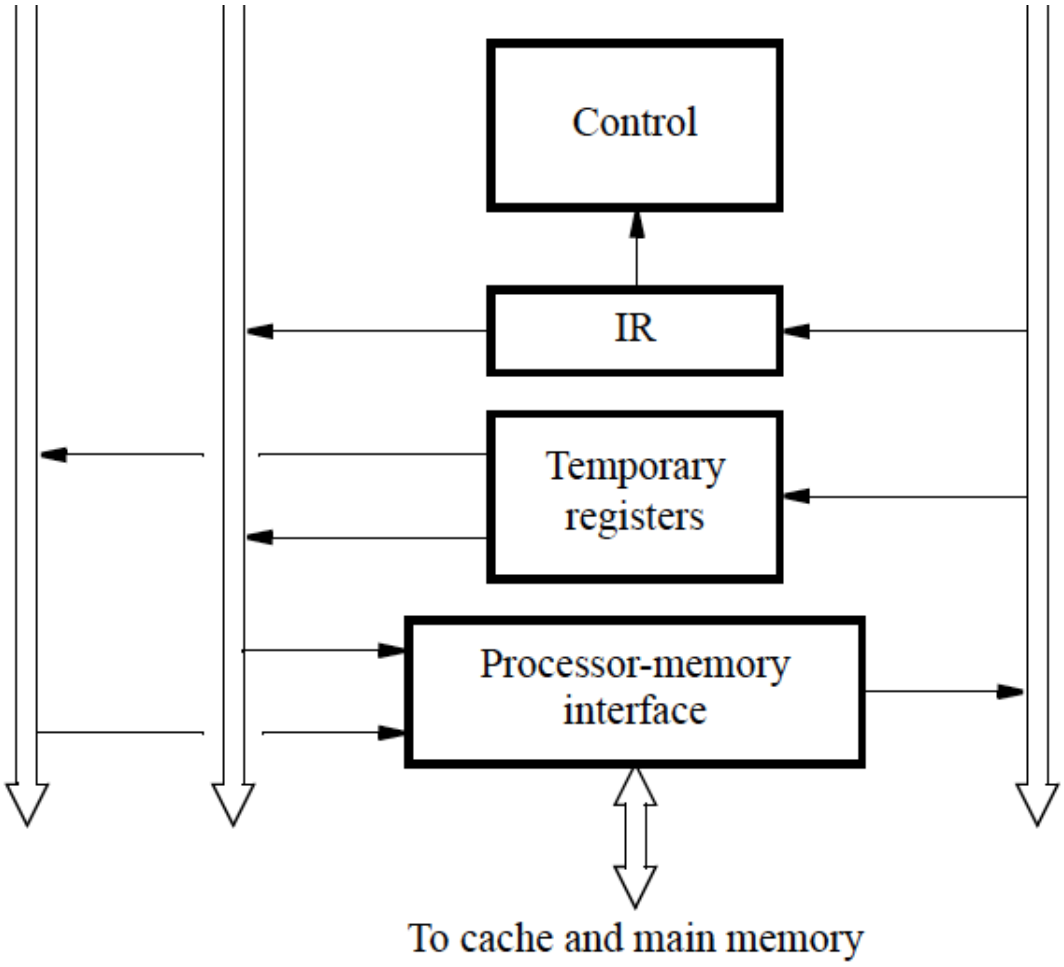


- Bus
  - When R_{in} = 1, the multiplexer selects the data on the bus line to be loaded into the flip-flop.
  - The output of the flip-flop is connected to the bus line through a tri-state gate, which is turned on When R_{out} = 1.

- A 3-bus interconnection network



- A 3-bus interconnection network



- Example: Add R5, R6
  - Fetching and executing this instruction using the hardware in Figure 5.24 can be performed in three steps:
    1. Memory address $\leftarrow$[PC], Read memory, Wait for MFC, IR $\leftarrow$ Memory data, PC $\leftarrow$[PC] + 4
    2. Decode instruction
    3. R5 $\leftarrow$[R5] + [R6]

Example: And X(R7), R9

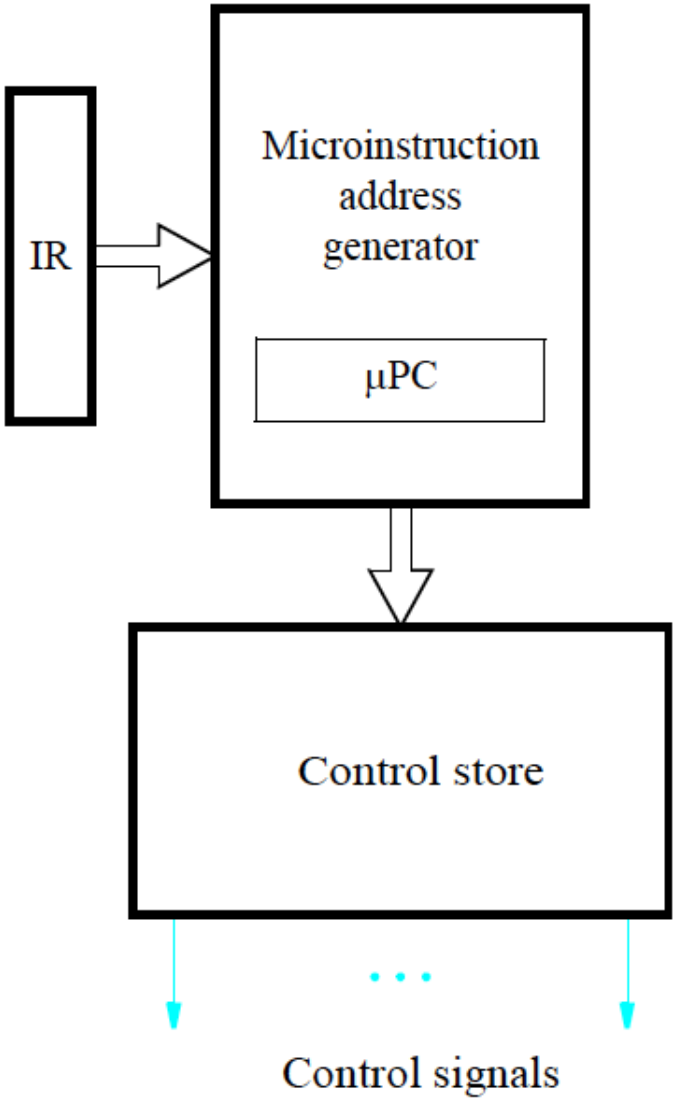
- Assume that the offset X is a 32-bit value given as the second word of the instruction.
- First, the OP-code word is fetched.
- Then, when the instruction decoding circuit recognizes the Index addressing mode, the offset X is fetched.

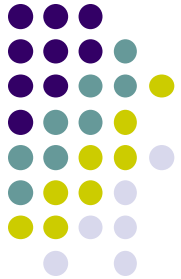
1	Memory address ← [PC], Read memory, Wait for MFC, IR ← Memory data, PC ← [PC] + 4
2	Decode instruction
3	Memory address ← [PC], Read memory, Wait for MFC, Temp1 ← Memory data, PC ← [PC] + 4
4	Temp2 ← [Temp1] + [R7]
5	Memory address ← [Temp2], Read memory, Wait for MFC, Temp1 ← Memory data
6	Temp1 ←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

Microprogramming

- Microprogramming is a software-based approach to the generation of control signals.
- The values of the control signals for each clock period are stored in a microinstruction (control word)
  - Each control signal is represented by a bit in an n-bit control word.
- A processor instruction is implemented by a sequence of microinstructions 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.

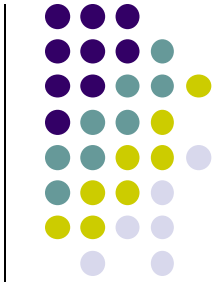
- Control signals generated from a microprogram





Microprogramming

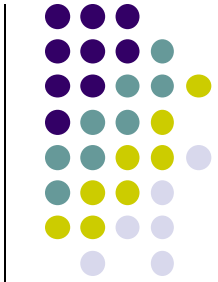
- Microprogramming provides the flexibility needed to implement more complex instructions in CISC processors.
- However, reading and executing microinstructions incurs undesirably long delays in high-performance processors.



Summary

- Hardware components
- Actions to fetch & execute instructions
- Control signals
- Approaches to generate control signals:
  - Hardwired control
  - Microprogramming
    - Microinstruction (Control word)
    - Microprogram (Microroutine)
    - Control store

Homework



- 5.4
- 5.9
- 5.15
- 5.23
- 5.25
- Give the sequence of actions for a Return-from-subroutine instruction in a RISC processor. Assume that the address LINK of the general-purpose register in which the subroutine return address is stored is given in the instruction field connected to address A of the register file ($IR_{31–27}$).