Computer Organization

CPU Organization - Functions and Interconnections

Chapters (3 + 12) + Lecture Notes

Program Concept

- Hardwired systems are inflexible
- General purpose hardware can do different tasks, given correct control signals
- Instead of re-wiring, supply a new set of control signals

What is a program?

- A sequence of steps
- For each step, an arithmetic or logical operation is done
- For each operation, a different set of control signals is needed

CPU Basics

- A typical CPU has three major components:
  - Register Set,
    - The register set is usually a combination of general-purpose and special-purpose registers.
    - General-purpose registers are used for any purpose.
    - Special-purpose registers have specific functions within the CPU.
  - Arithmetic Logic Unit, and
  - Control Unit (CU).

CPU Basics

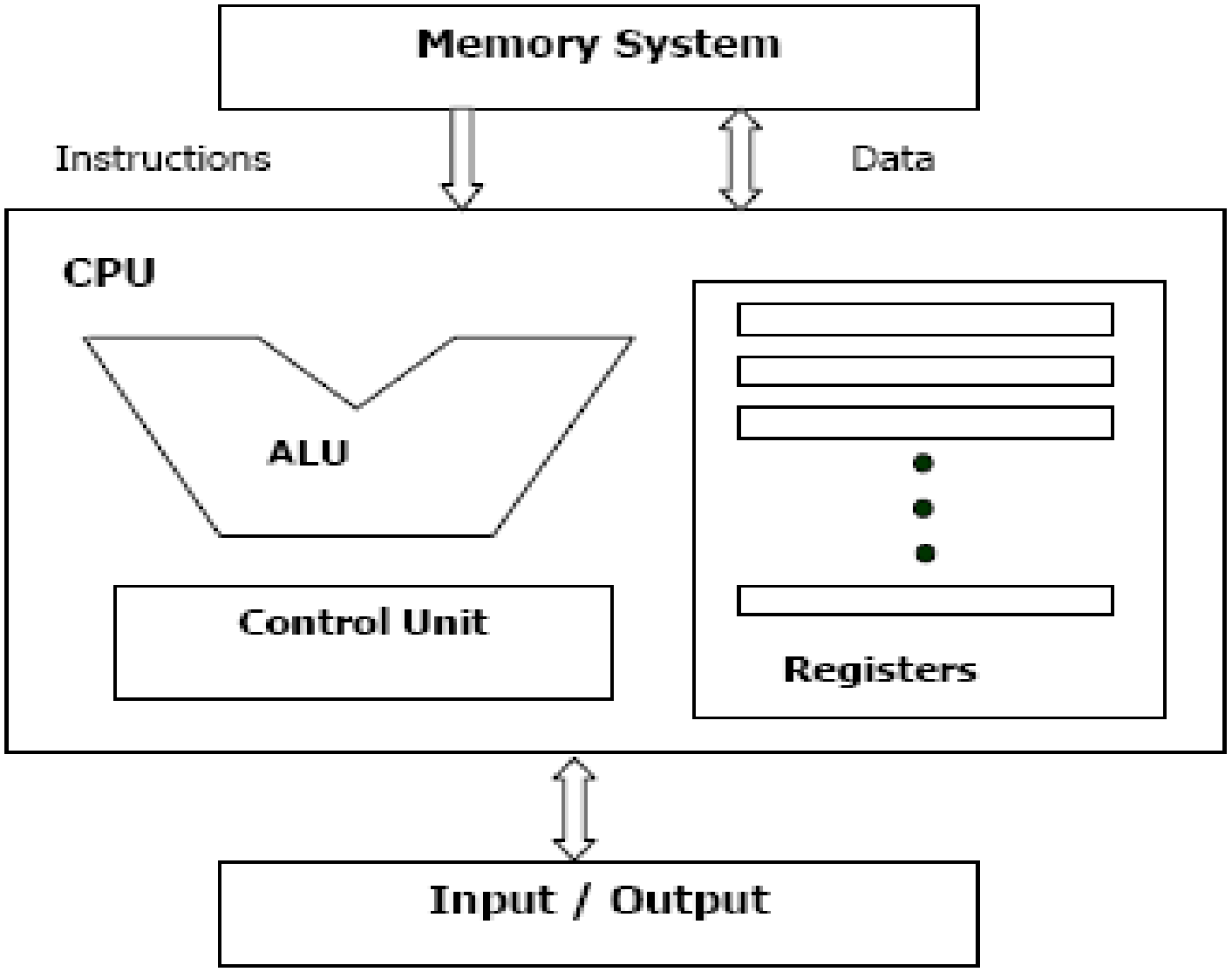
- The Control Unit and the Arithmetic and Logic Unit constitute the Central Processing Unit
- Data and instructions need to get into the system and results out
  - Input/output
- Temporary storage of code and results is needed
  - Main memory

Function of Control Unit

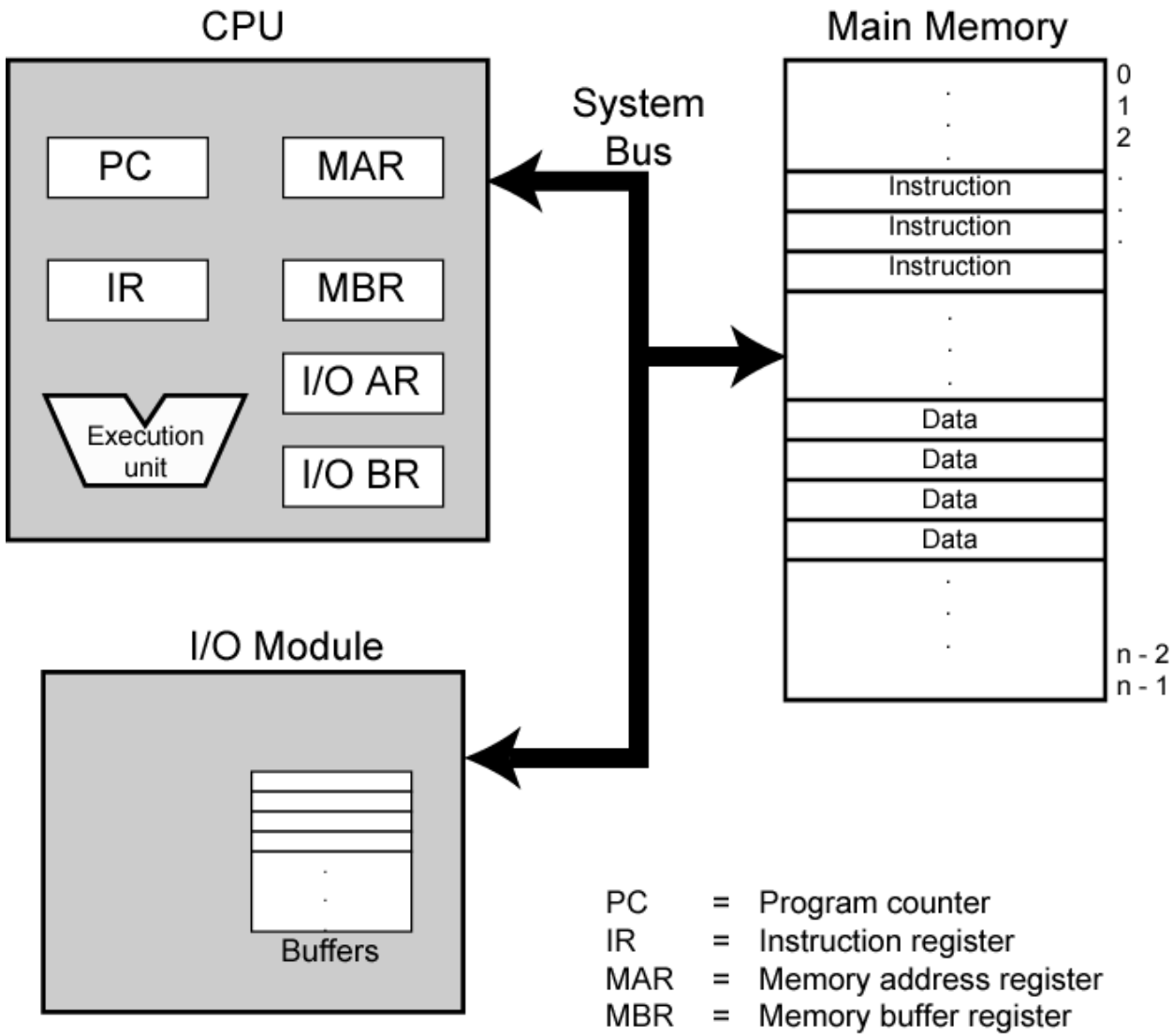
- For each operation a unique code is provided
  - e.g. ADD, MOVE
- A hardware segment accepts the code and issues the control signals

- We have a computer!

CPU Basics


Memory System
Instructions
Data
CPU
ALU
Control Unit
Registers
Input / Output

Computer Components: Top Level View

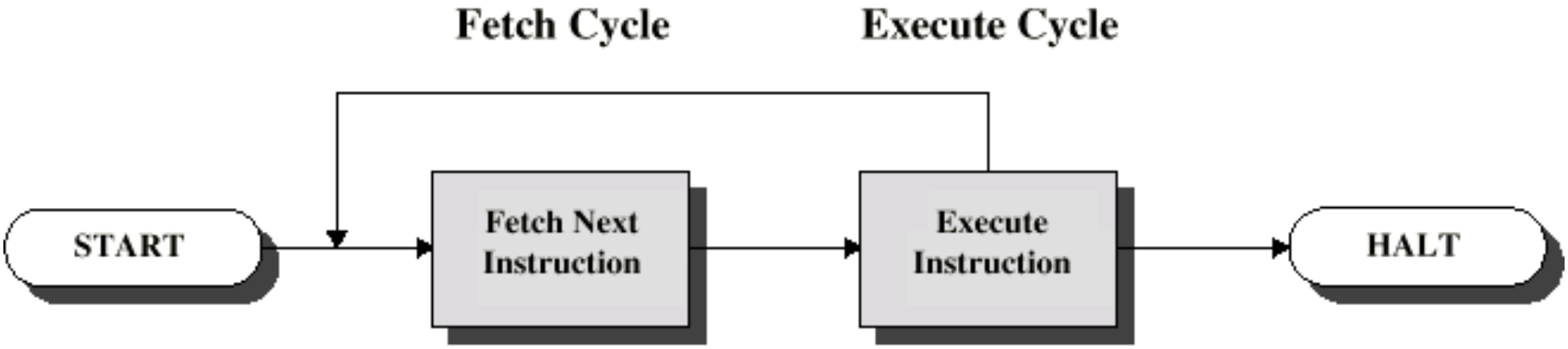


CPU Basics: Instruction Cycle

- A typical and simple execution cycle in a CPU is as follows:
  - The next instruction to be executed, whose address is obtained from the PC, is fetched from the memory and stored in the IR.
  - Instruction is decoded.
  - Operands are fetched from the memory and stored in CPU registers, if needed.
  - Instruction is executed.
  - Results are transferred from CPU registers to the memory, if needed.
- The execution cycle is repeated as long as there are more instructions to execute.
- A check for pending interrupts is usually included in the cycle.

Instruction Cycle

- Two steps:
  - Fetch
  - Execute



Registers

- CPU must have some working space (temporary storage) Called registers
- Number and function vary between processor designs
- One of the major design decisions
- Top level of memory hierarchy
- User Visible Registers
  - General Purpose
  - Data
  - Address
  - Condition Codes

Register Set

- Memory Access Registers
  - Two registers are essential in memory write and read operations:
    - *memory data register* (*MDR*) and
    - *memory address register* (*MAR*).
  - The *MDR* and *MAR* are used exclusively by the CPU and are not directly accessible to programmers.
  - In order to perform a write operation into a specified memory location, the MDR and MAR are used as follows:
    - The word to be stored into the memory location is first loaded by the CPU into *MDR*
    - The address of the location into which the word is to be stored is loaded by the CPU into a *MAR*.

Register Set

- Memory Access Registers
  - Similarly, to perform a memory read operation, the MDR and MAR are used as follows:
    - The address of the location from which the word is to be read is loaded into the MAR.
    - The required word will be loaded by the memory into the MDR ready for use by the CPU.
- Instruction Fetching Registers
  - Two main registers are involved in fetching an instruction for execution:
    - the *program counter* (PC) and
    - the *instruction register* (IR).

Register Set

- Condition Registers
  - Condition registers, or flags, are used to maintain status information.
  - Some architectures contain a special Program Status Word (PSW) register.
  - The PSW contains bits that are set by the CPU to indicate the current status of an executing program.
  - These indicators are typically for arithmetic operations, interrupts, memory protection information, or processor status.

Sign of last result, Zero, Carry, Equal, Overflow, Interrupt enable/disable, Supervisor

CPU Instruction Cycle

- **Fetch Instructions**
  - The sequence of events in fetching an instruction can be summarized as follows:
    - The contents of the PC are loaded into the MAR.
    - The value in the PC is incremented. (This operation can be done in parallel with a memory access).
    - As a result of a memory read operation, the instruction is loaded into the MDR.
    - The contents of the MDR are loaded into the IR.

Step	Micro-operation
t_0	MAR ← (PC); PC ← (PC) + 4
t_1	MDR ← Mem[MAR]
t_2	IR ← (MDR)

CPU Instruction Cycle

- **Execute Simple Arithmetic Operation**
  - Add *R*1, *R*2, *R*0
    - This instruction adds the contents of source registers *R*1 and *R*2, and stores the results in destination register *R*0. This addition can be executed as follows:
      - The registers *R*0, *R*1, *R*2, are extracted from the IR.
      - The contents of *R*1 and *R*2 are passed to the ALU for addition.
      - The output of the ALU is transferred to *R*0.

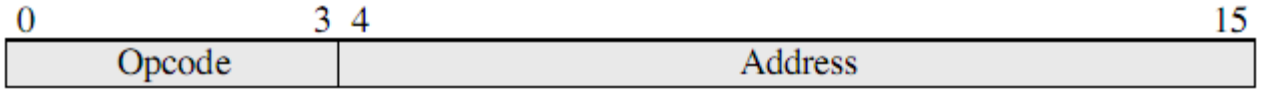
Step	Micro-operation
t_0	$R_0 \leftarrow (R_1) + (R_2)$

CPU Instruction Cycle

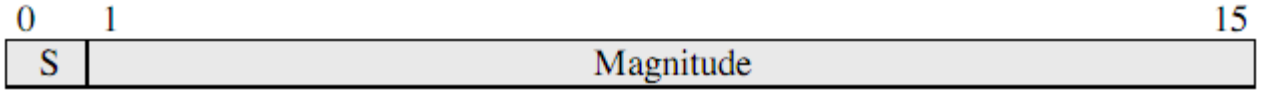
- Execute Simple Arithmetic Operation
  - Add *X*, *R*0
    - This instruction adds the contents of memory location *X* to register *R*0 and stores the result in *R*0. This addition can be executed as follows:
      - The memory location *X* is extracted from IR and loaded into MAR.
      - As a result of memory read operation, the contents of *X* are loaded into MDR.
      - The contents of MDR are added to the contents of *R*0.

Step	Micro-operation
t_0	MAR ← X
t_1	MDR ← Mem[MAR]
t_2	$R_0 \leftarrow (R_0) + (MDR)$

Format of Instructions and Data



(a) Instruction format



(b) Integer format

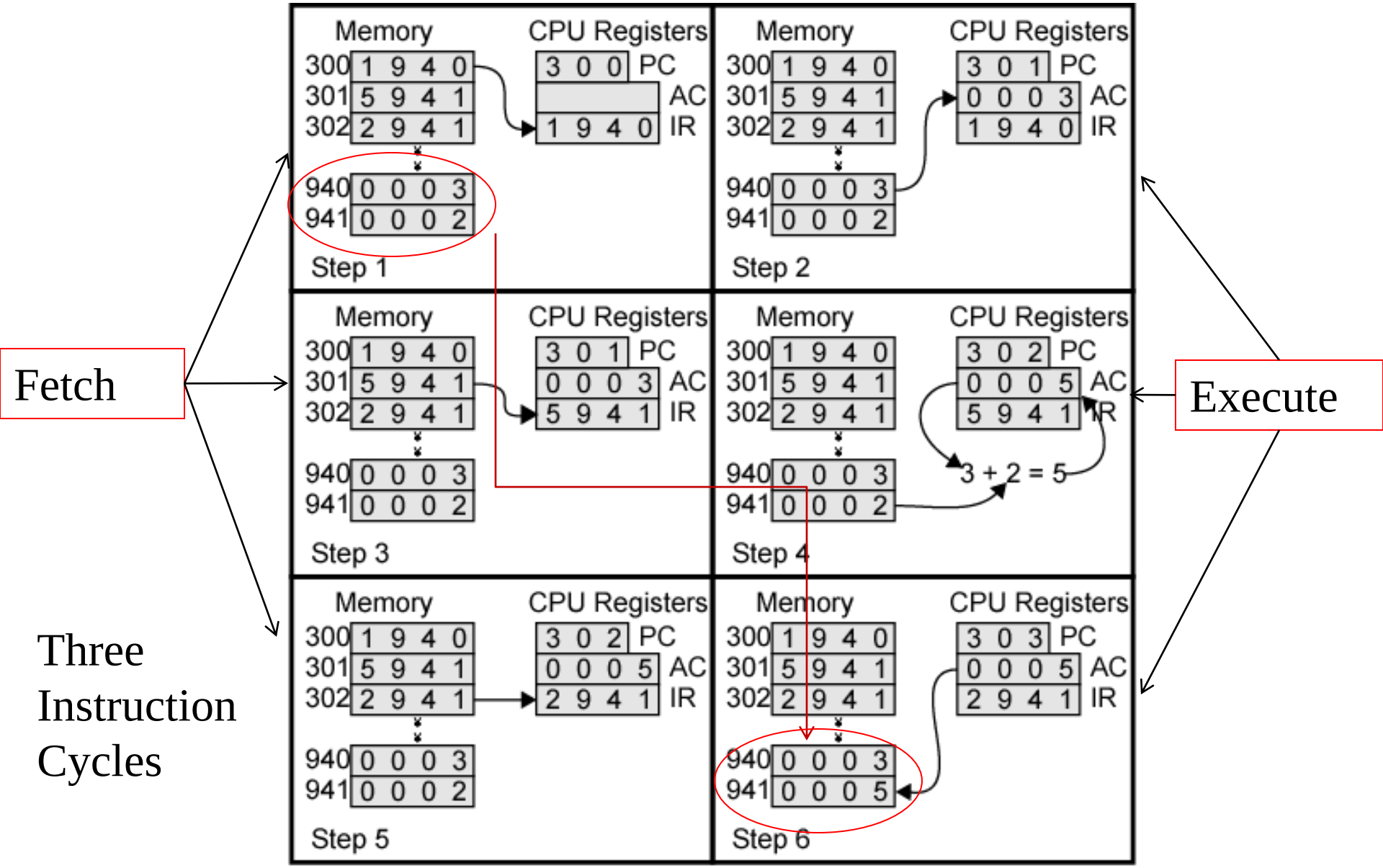
Program Counter (PC) = Address of instruction
Instruction Register (IR) = Instruction being executed
Accumulator (AC) = Temporary storage

(c) Internal CPU registers

0001 = Load AC from Memory
0010 = Store AC to Memory
0101 = Add to AC from Memory

(d) Partial list of opcodes

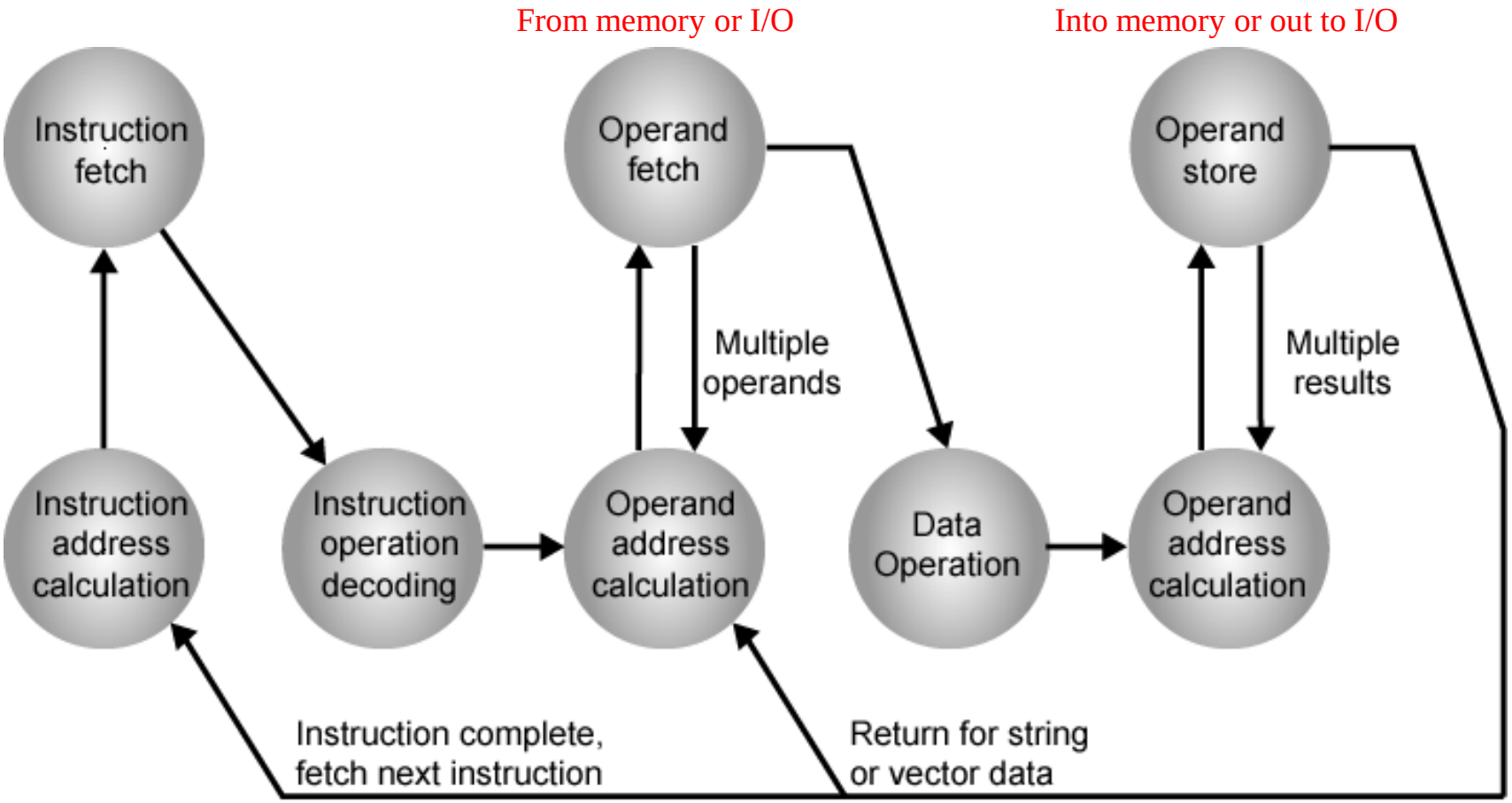
Example of Program Execution



Example of Program Execution

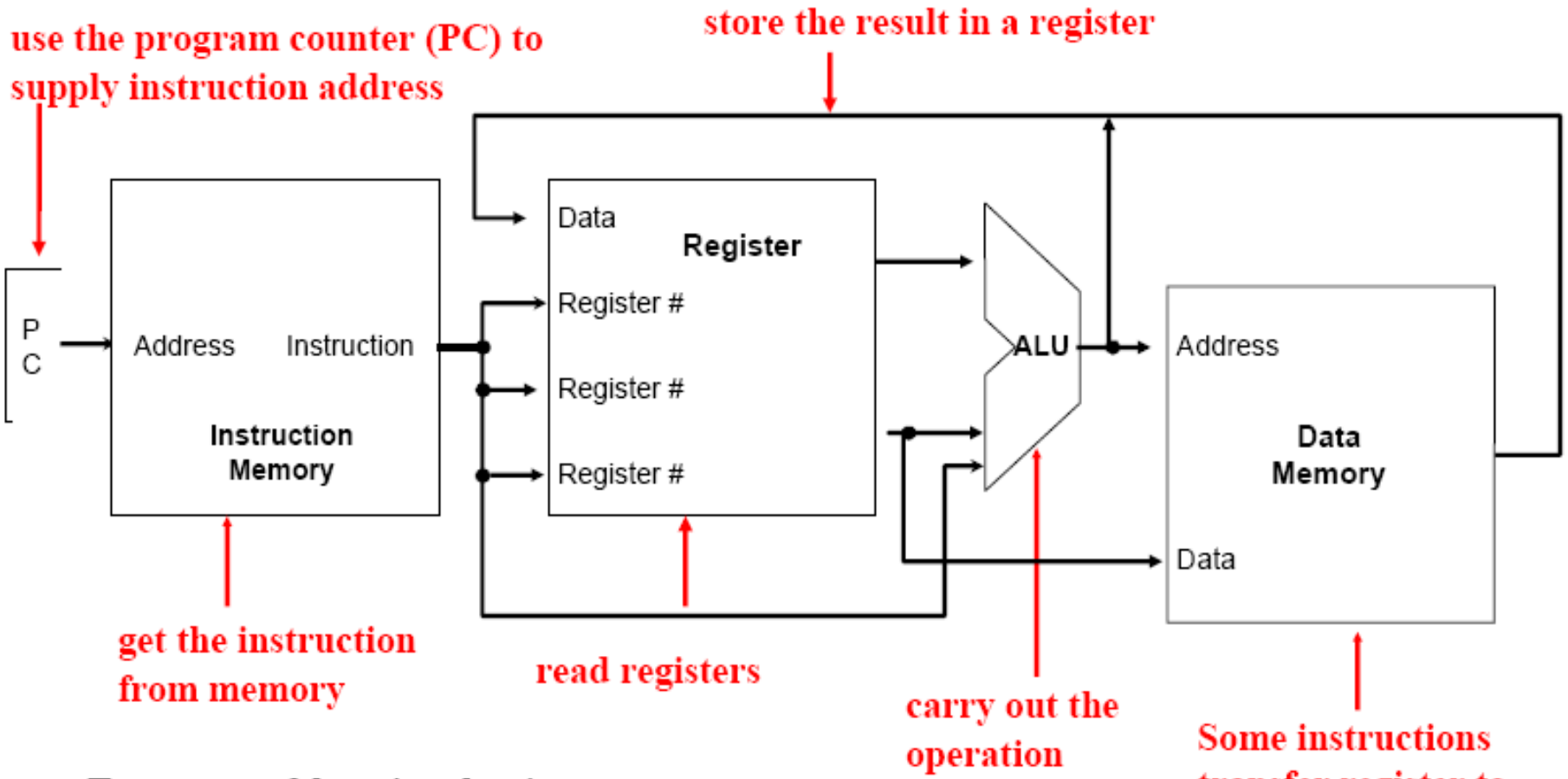
1. PC contains address of first instruction: 300h
2. PC value moved to AR or MAR
3. Instruction 1940h (hex) is loaded into IR (IR ← M[AR])
4. Increment PC.
5. The “1” value in IR means LOAD AC.
6. The “940” value is the memory address for data to be read and loaded to AC: AC ← M[AR]
7. AC gets the value of 3h. Increment PC
8. Next inst. is 5941h: the value “5” means add to AC from memory location “941”. Increment PC
9. Next inst. is 2941h: the value “2” means store AC to memory location “941”

Instruction Cycle State Diagram



Execution cycle may reference memory more than once. The operation could be an I/O type. For some instruction, some states may be null and others may be accessed more than once.

An Abstract (Simplified) View:



- Two types of functional units:
  - elements that operate on data values (combinational)
  - elements that contain states (sequential)

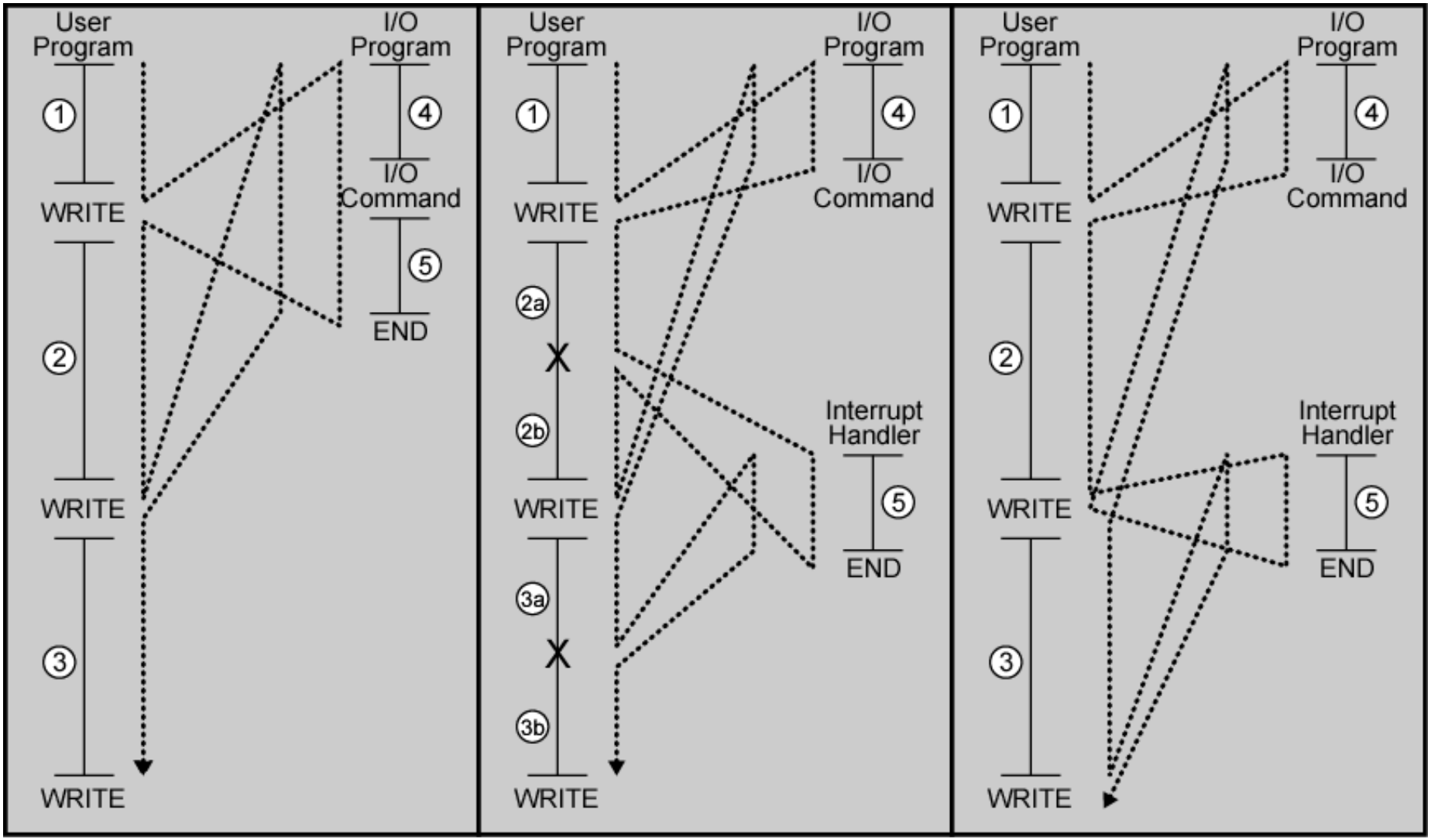
Interrupts

- Mechanism by which other modules (e.g. I/O) may interrupt normal sequence of processing. *Improves process efficiency*.

Classes:

- Program (condition occurs as a result of instruction execution)
  - —e.g. Arithmetic overflow, division by zero
- Timer
  - —Generated by internal processor timer
  - —Used in pre-emptive multi-tasking
- I/O (from I/O controller)
  - —to signal normal completion or error
- Hardware failure
  - —e.g. memory parity error, power failure

Program Flow Control

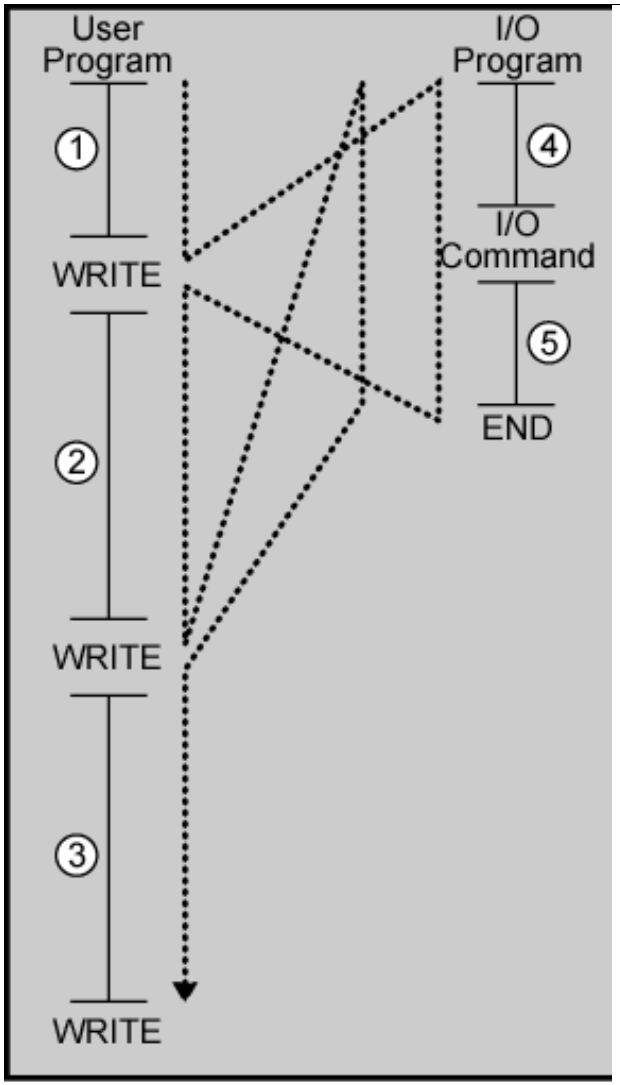


(a) No interrupts

(b) Interrupts; short I/O wait

(c) Interrupts; long I/O wait

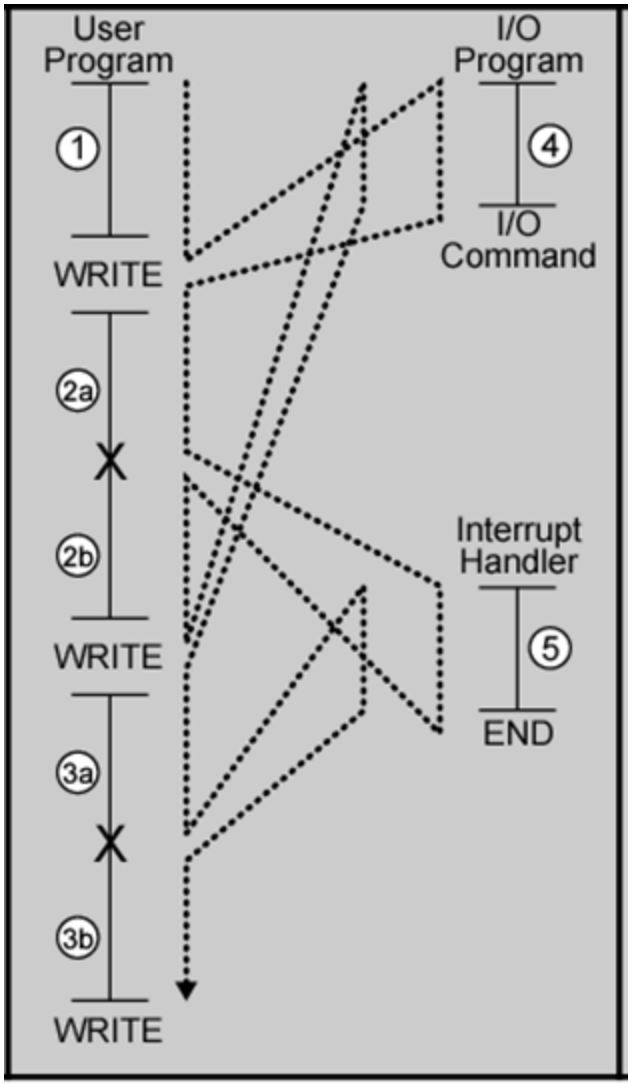
Program Flow Control



(a) No interrupts

1. User program executes codes 1, 2, and 3 that do not involve the I/O.
2. It interleaves the codes with WRITE calls to an I/O program.
3. The IO program has a sequence of instructions, 4, to prep for the I/O operation. It has the actual IO command, and a sequence of instructions, 5, to complete the operation (i.e. set flag for success or failure.
4. Since no interrupt, the IO command may take long, and the program has to wait for the IO device to perform what it was asked to do.
5. In this case, the IO program is hung up waiting, and the user program is stopped at WRITE call location.

Program Flow Control



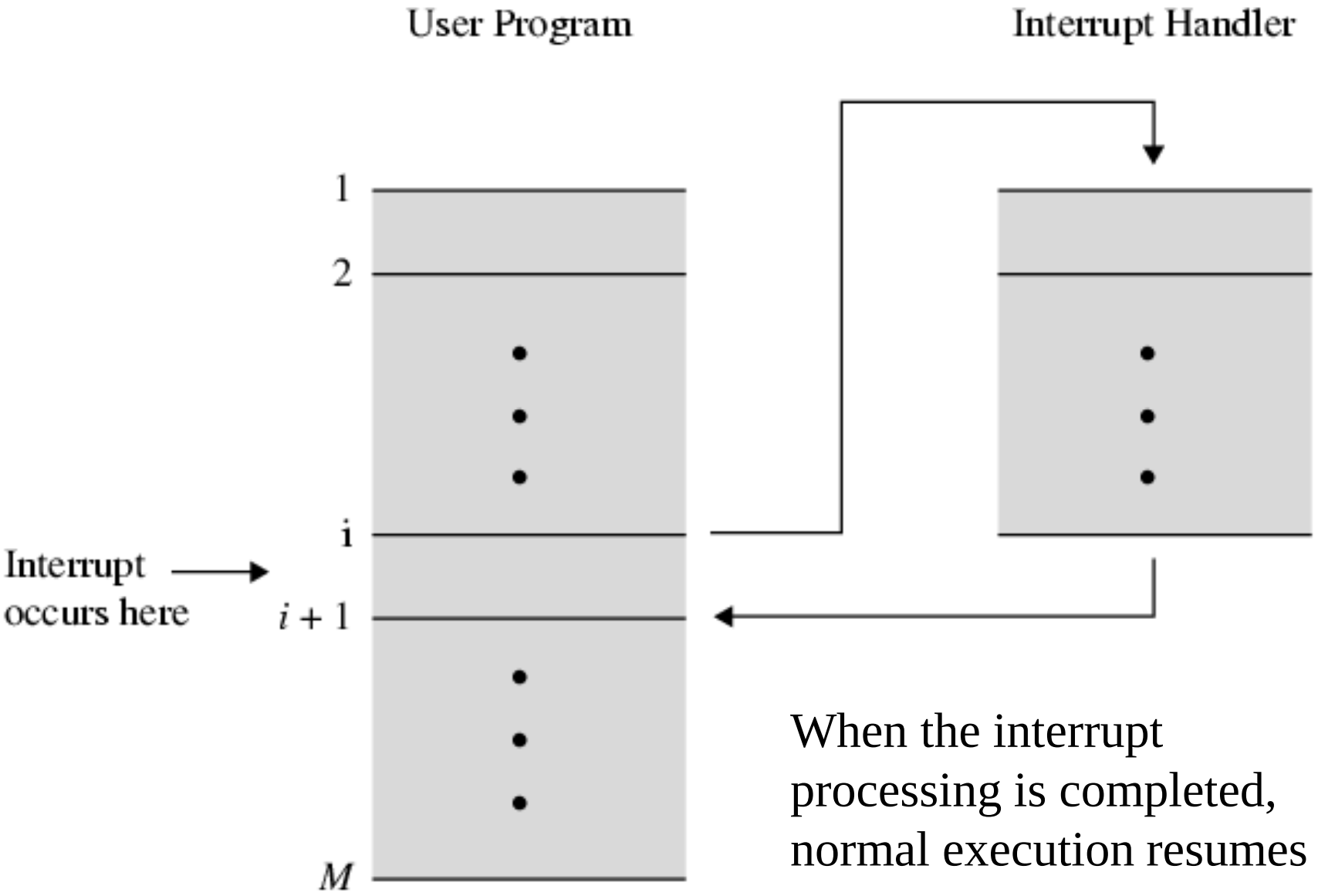
(b) Interrupts; short I/O wait

1. With interrupts, Processor can execute other inst. While an IO operation is in progress.
2. A WRITE call is made and the IO preparation code,4, and the I/O command are executed.
3. User program resumes execution while the external device is busy doing what it was told to do via the I/O command (print data).
4. When external device is ready to be serviced again, its I/O module sends an Interrupt Request signal to the processor.
5. Processor suspends operations to handle the Interrupt (point X) (interrupt handler). Normal operation then resumes.
6. Interrupt handling code is not part of the user code.
7. Interrupt handling is the responsibility of the Processor and OS.

Interrupt Cycle

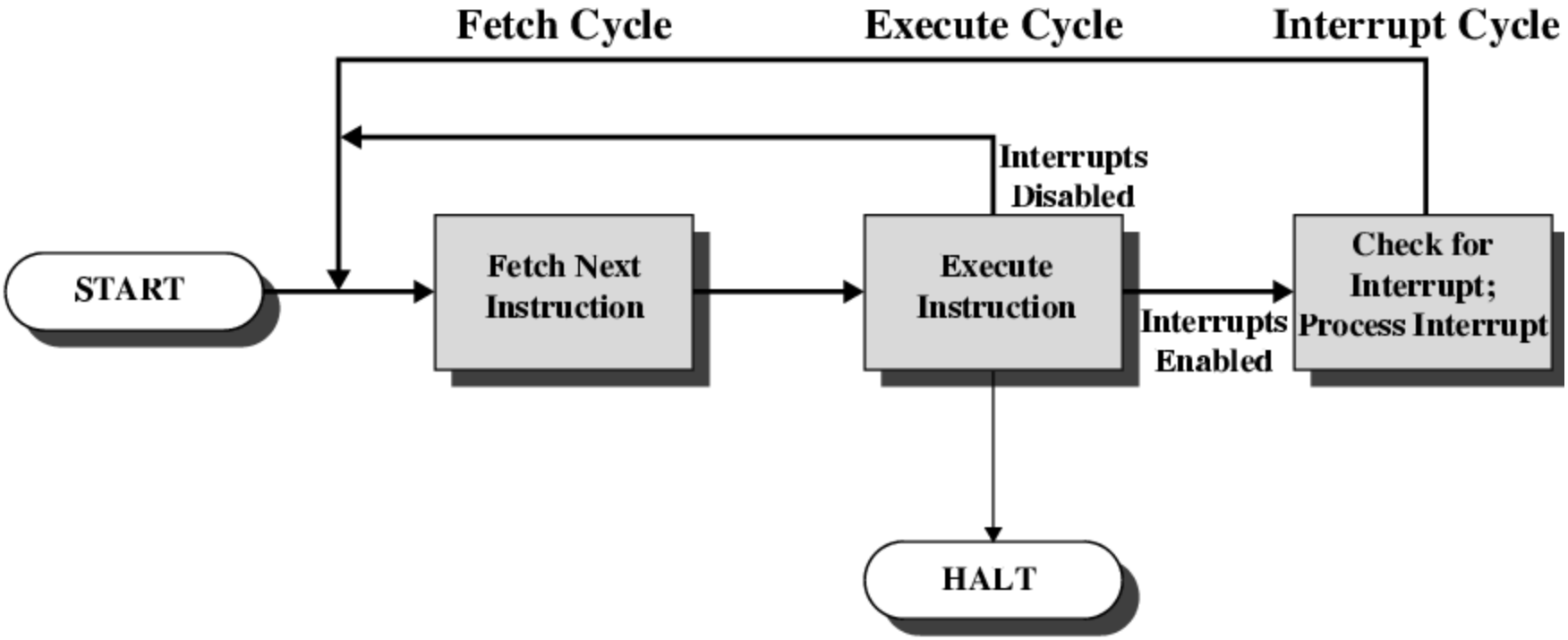
- Added to instruction cycle
- Processor checks for interrupt
  - Indicated by an interrupt signal
- If no interrupt, fetch next instruction
- If interrupt pending:
  - Suspend execution of current program
  - Save context
  - Set PC to start address of interrupt handler routine
  - Process interrupt
  - Restore context and continue interrupted program

Transfer of Control via Interrupts



When the interrupt processing is completed, normal execution resumes

Instruction Cycle with Interrupts

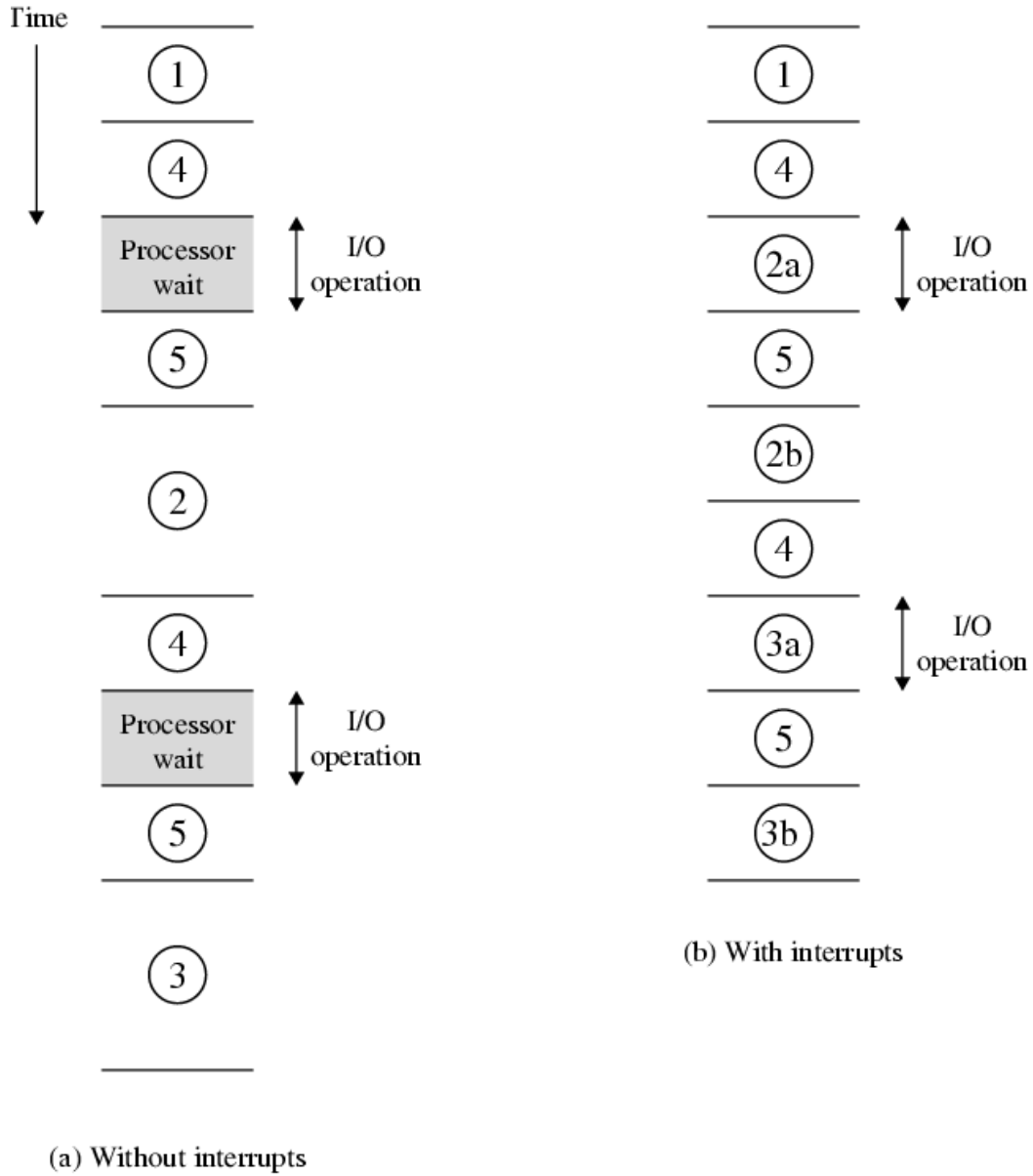


If Interrupt is pending:

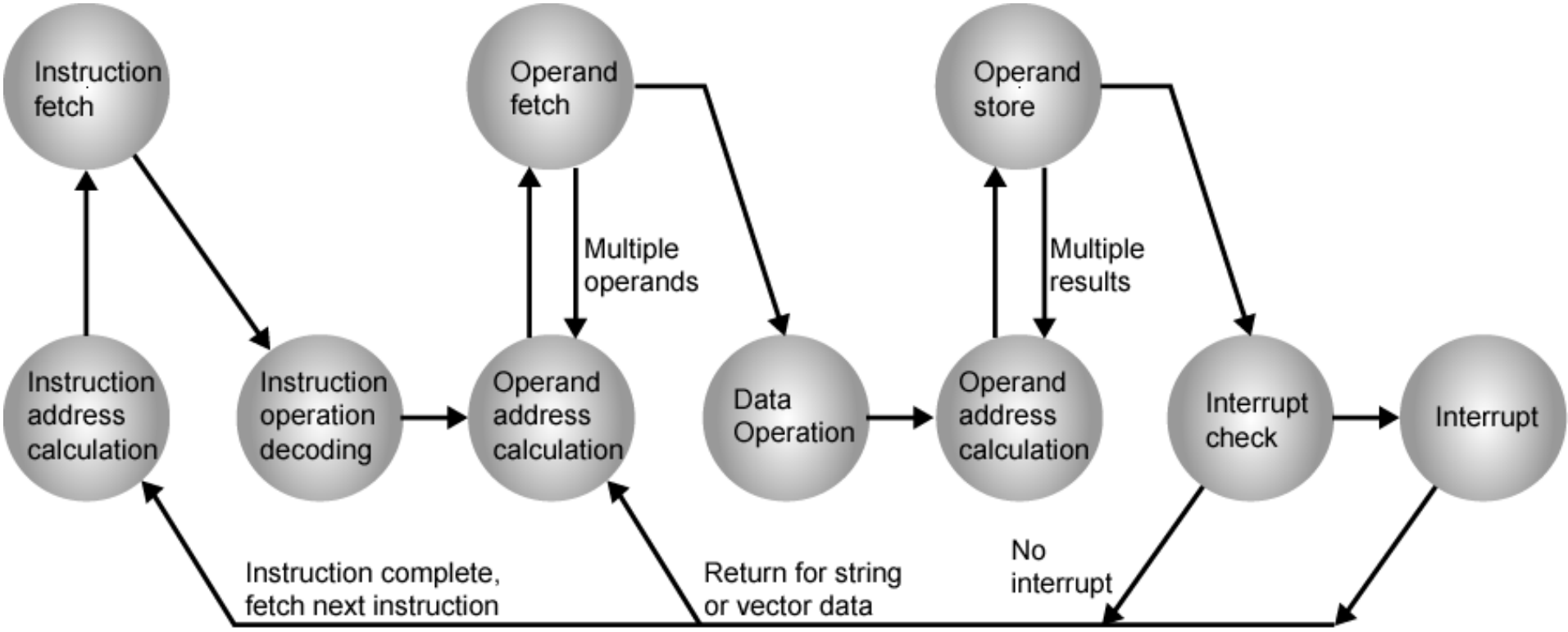
-Suspend execution of current program and save address of next instruction.

- Set the PC to the starting address of an Interrupt Handler Routine and then fetch the first instruction in the handler program

Program Timing
Short I/O Wait



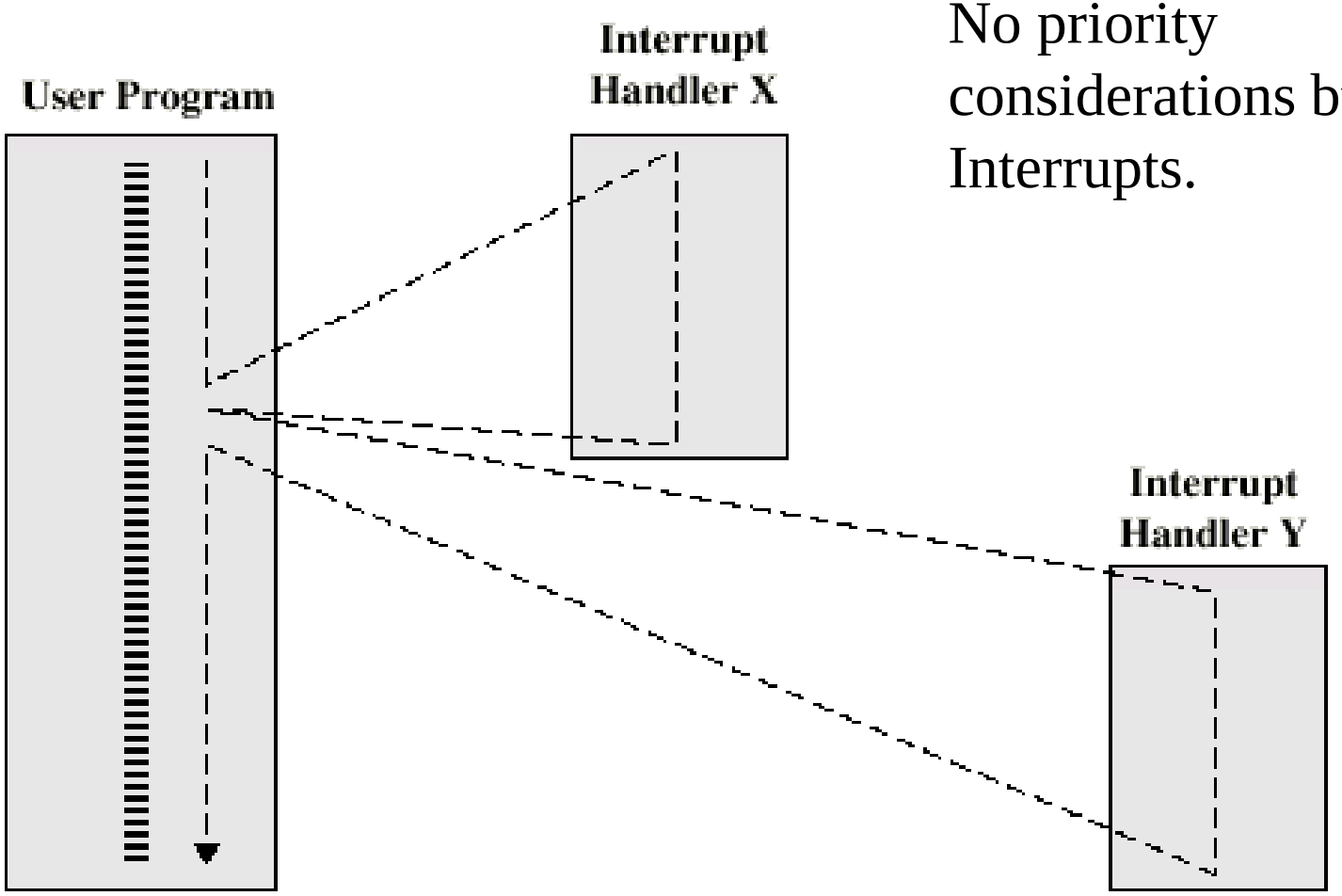
Instruction Cycle (with Interrupts) - State Diagram



Multiple Interrupts

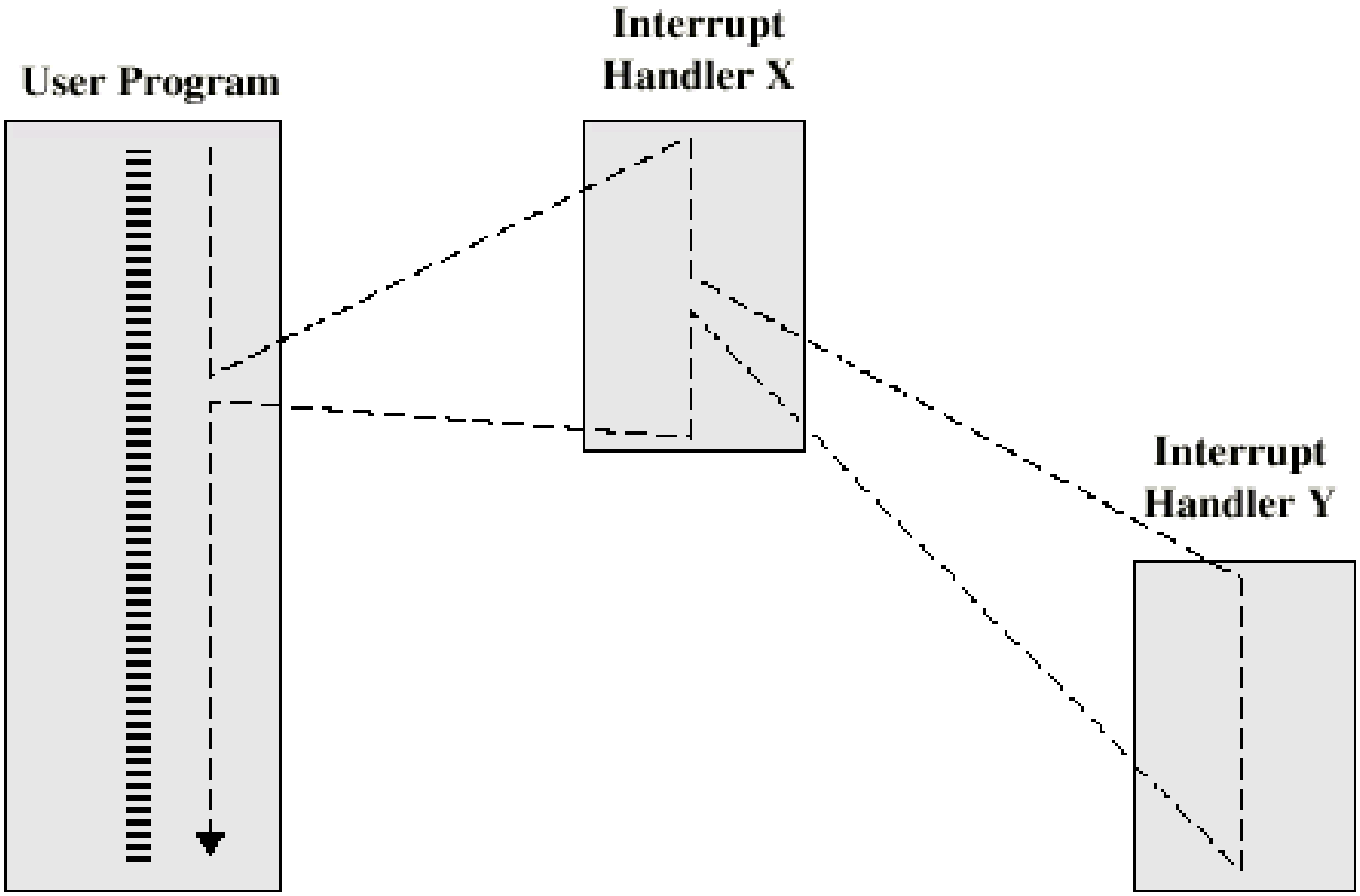
- Disable interrupts
  - Processor will ignore further interrupts whilst processing one interrupt
  - Interrupts remain pending and are checked after first interrupt has been processed
  - Interrupts handled in sequence as they occur
- Define priorities
  - Low priority interrupts can be interrupted by higher priority interrupts
  - When higher priority interrupt has been processed, processor returns to previous interrupt

Multiple Interrupts - Sequential

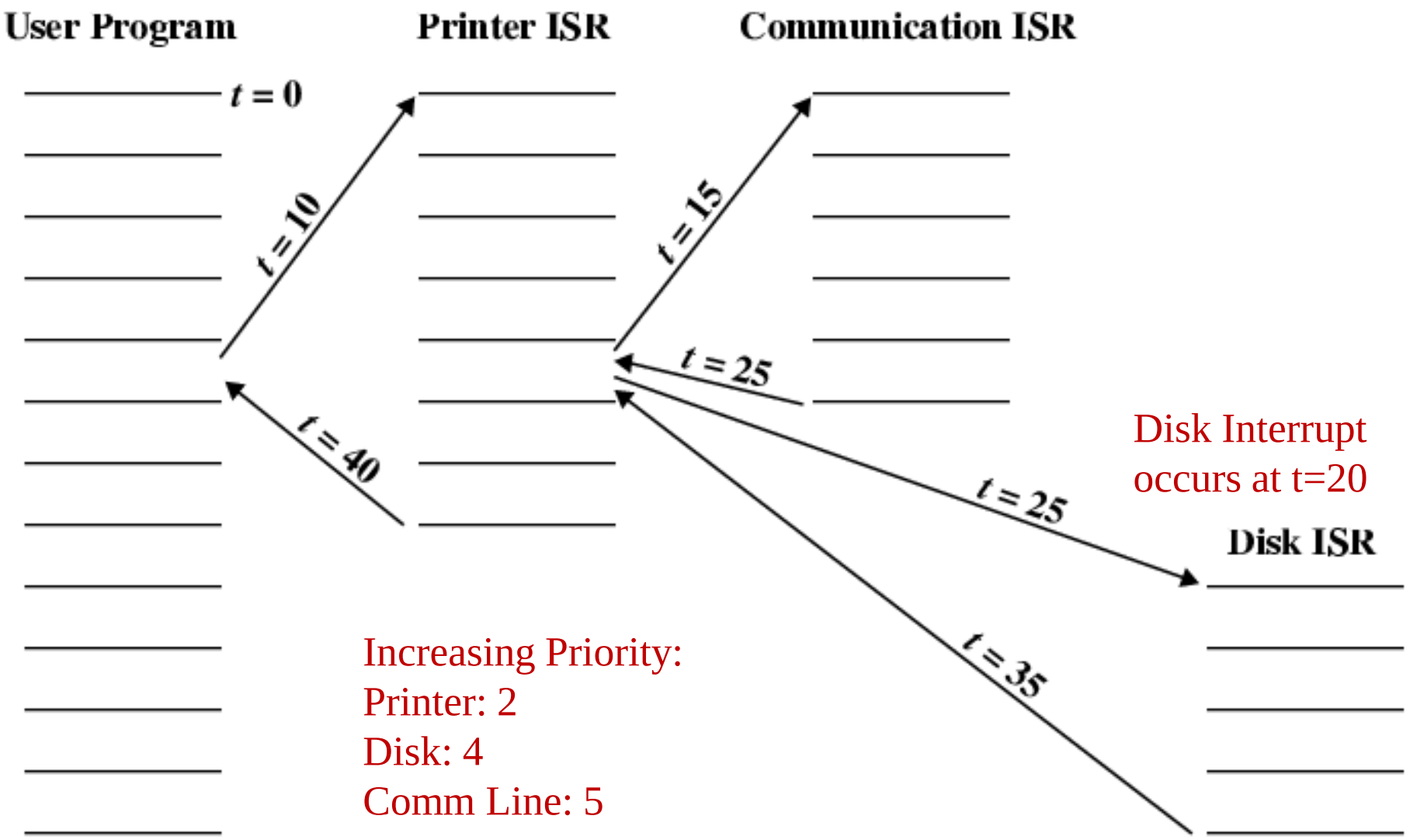


No priority considerations btw Interrupts.

Multiple Interrupts – Nested

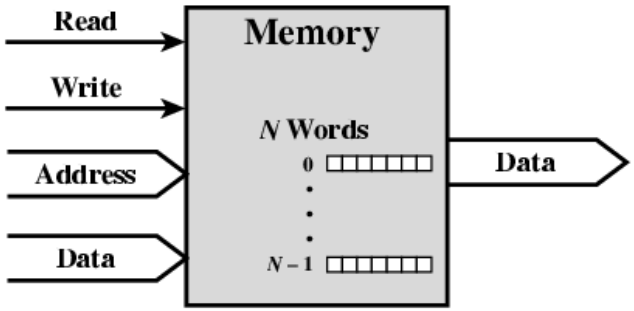


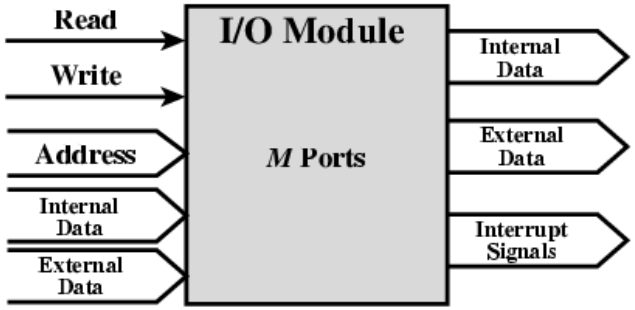
Time Sequence of Multiple Interrupts

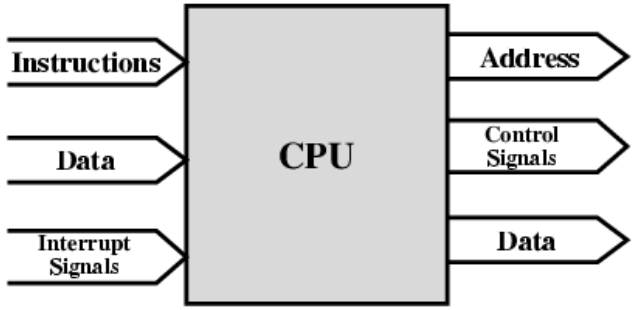


Connecting

- All the units must be connected
- Different type of connection for different type of unit
  - Memory
  - Input/Output
  - CPU







CPU Connection

- Reads instruction and data
- Writes out data (after processing)
- Sends control signals to other units
- Receives (& acts on) interrupts

Memory Connection

- Receives and sends data
- Receives addresses (of locations)
- Receives control signals
  - Read
  - Write
  - Timing

Input/Output Connection

- Similar to memory from computer's viewpoint
  - Output
    - Receive data from computer
    - Send data to peripheral
  - Input
    - Receive data from peripheral
    - Send data to computer
- Receive control signals from computer
- Send control signals to peripherals
  - e.g. spin disk
- Receive addresses from computer
  - e.g. port number to identify peripheral
- Send interrupt signals (control)

Buses

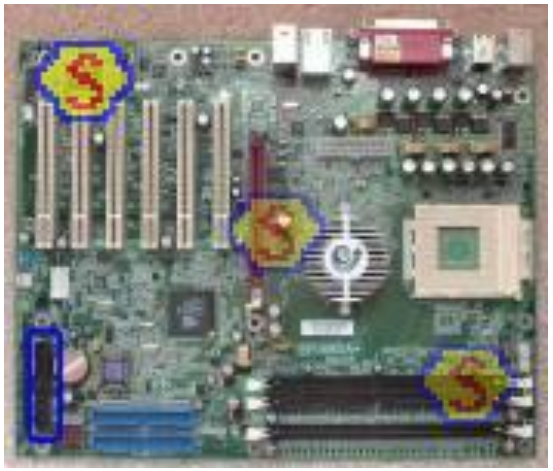
- There are a number of possible interconnection systems
- Single and multiple BUS structures are most common
- e.g. Control/Address/Data bus (PC)
- e.g. Unibus (DEC-PDP)

What is a Bus?

- A communication pathway connecting two or more devices
- Usually broadcast
- Often grouped
  - A number of channels in one bus
  - e.g. 32 bit data bus is 32 separate single bit channels
- Power lines may not be shown

Computer System Buses

- What do buses look like?
  - Parallel lines on circuit boards
  - Ribbon cables
  - Strip connectors on mother boards
    - e.g. PCI
  - Sets of wires



Bus Types

- Dedicated
  - Separate data & address lines
- Multiplexed
  - Shared lines
  - Address valid or data valid control line
  - Advantage - fewer lines
  - Disadvantages
    - More complex control
    - Ultimate performance

Data Bus

- Carries data
  - Remember that there is no difference between "data" and "instruction" at this level
- Width is a key determinant of performance
  - 8, 16, 32, 64 bit

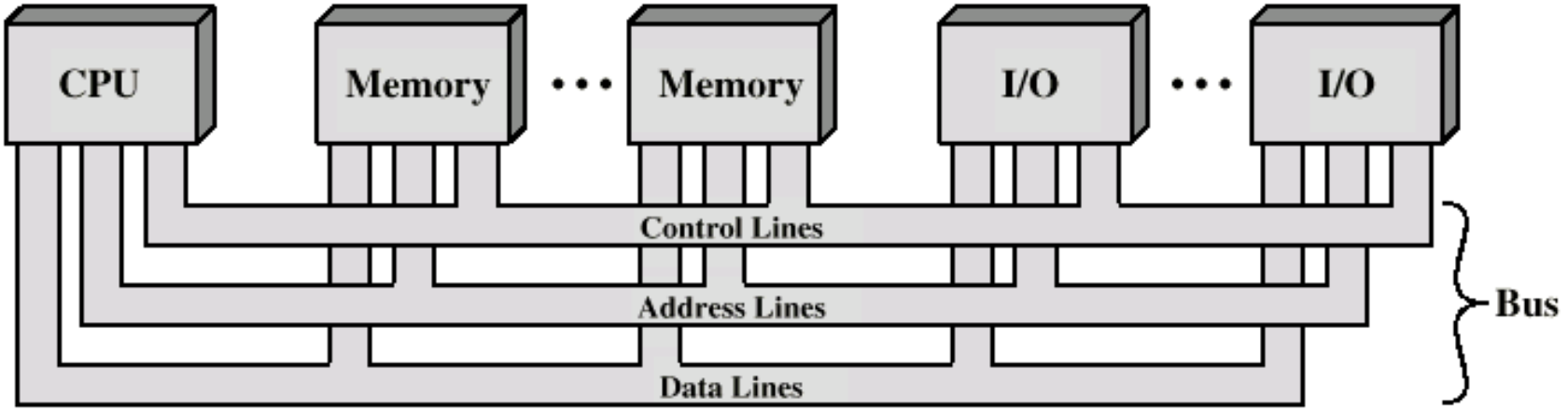
Address bus

- Identify the source or destination of data
- e.g. CPU needs to read an instruction (data) from a given location in memory
- Bus width determines maximum memory capacity of system
  - e.g. 8080 has 16 bit address bus giving 64k address space

Control Bus

- Control and timing information
  - —Memory read/write signal
  - —Interrupt request
  - —Clock signals

Bus Interconnection Scheme



Big and Yellow? School Buses !

- What do buses look like?
  - Parallel lines on circuit boards
  - Ribbon cables
  - Strip connectors on mother boards
    - e.g. PCI
  - Sets of wires

CPU local (internal) Bus Organization

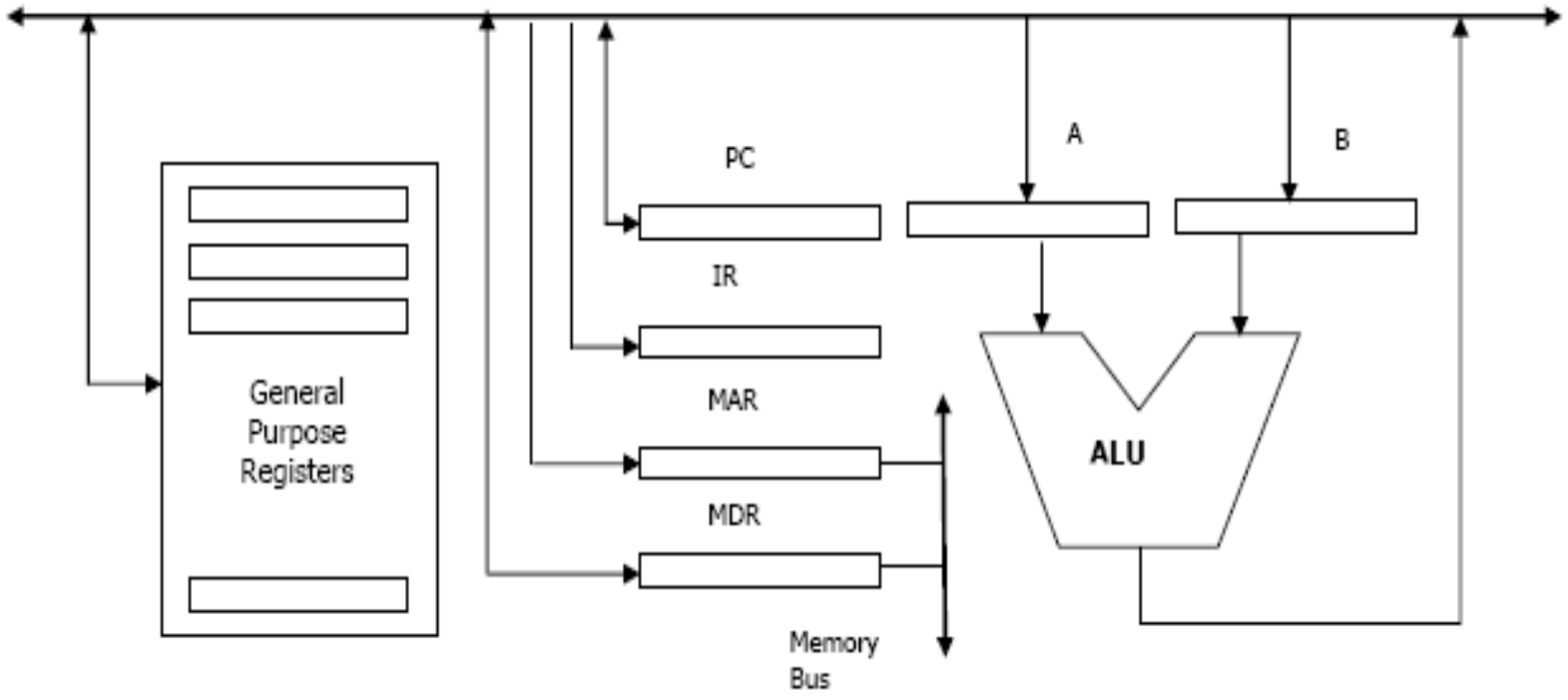
- **One-Bus Organization**
  - —Using one bus, the CPU registers and the ALU use a **single bus** to move outgoing and incoming **data**.
  - —Since a bus can handle only **a single data** movement within one **clock cycle**, **two-operand** operations will need **two cycles** to fetch the operands
  - —**Additional registers** may be needed to buffer data for the ALU
  - —This bus organization is the **simplest** and **least expensive**, but it **limits** the amount of **data transfer** that can be done in the same clock cycle, which will **slow down** the overall **performance.**

Single Bus Problems

- Lots of devices on one bus leads to:
  - Propagation delays
    - **Long data paths** mean that co-ordination of bus use can adversely affect performance
    - If aggregate data transfer approaches bus capacity
- Most systems use **multiple buses** to overcome these problems

CPU local Bus Organization

- One-Bus Organization

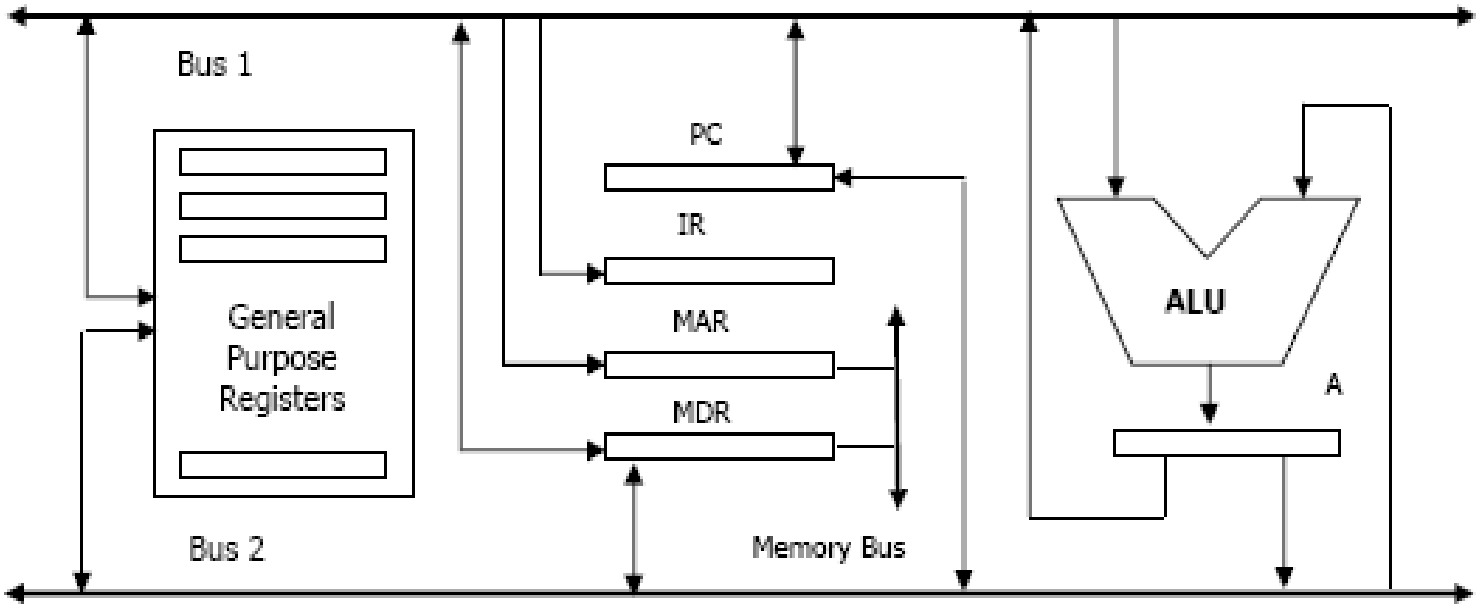


CPU local Bus Organization

- **Two-Bus Organization**
  - Using two buses is a **faster** solution than the one-bus organization.
  - In this case, general-purpose registers are connected to both buses.
  - **Data** can be **transferred** from **two** different **registers** to the ALU at the **same time**.
  - Therefore, a **two operand** operation can fetch both operands in the **same clock cycle**.
  - An **additional buffer** register may be needed to hold the **output** of the **ALU** when the **two buses are busy** carrying the two operands.
  - In some cases, **one** of the buses may be dedicated for moving **data into** registers (**in-bus**), while the **other** is dedicated for transferring **data out** of the registers (**out-bus**).

CPU local Bus Organization

- Two-Bus Organization



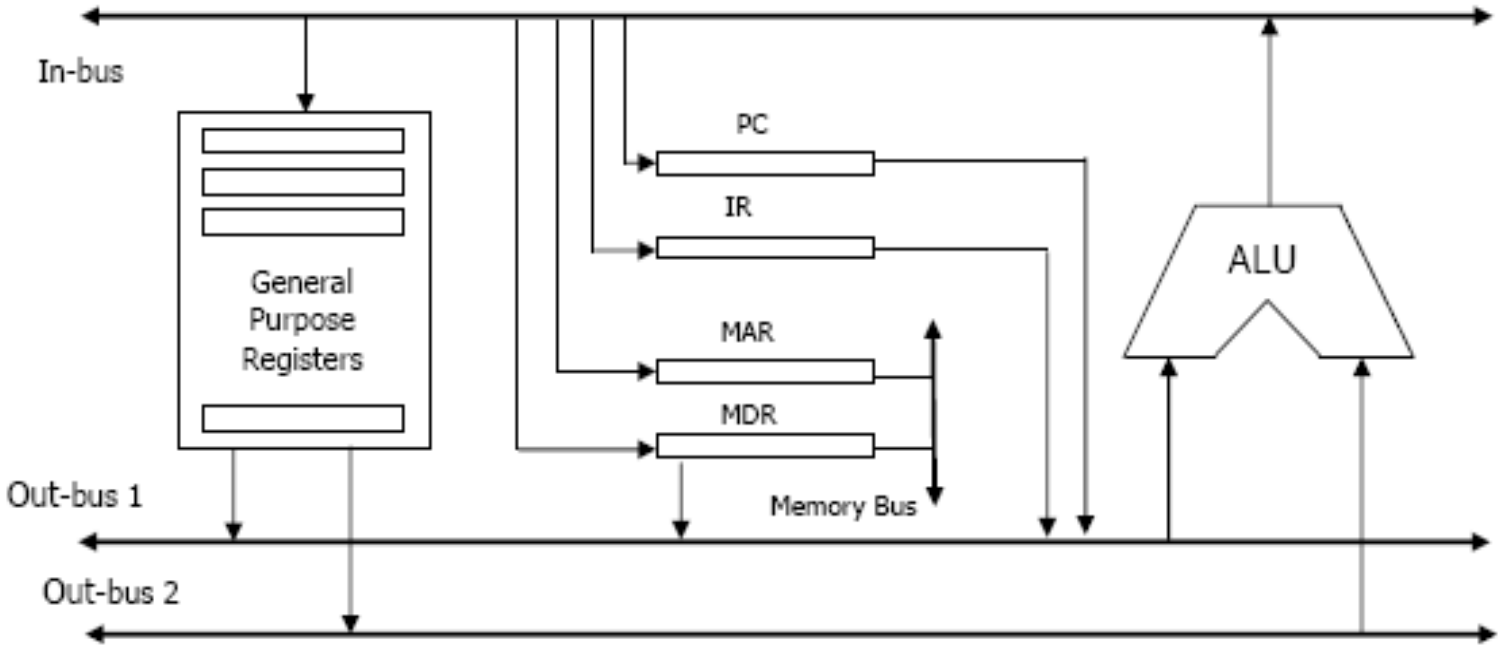
Two-Bus Datapath

CPU local Bus Organization

- **Three-Bus Organization**
  - —In a three-bus organization, **two buses** may be used as **source** buses while the **third** is used as **destination**.
  - —The **source** buses move **data out** of registers (**out-bus**), and the **destination** bus may move data **into** a register (**in-bus**).
  - —Each of the two out-buses is connected to an ALU input.
  - —The output of the ALU is connected directly to the in-bus.
  - —The **more buses** we have, the **more data** we can move **within a single** clock cycle.
  - — However, increasing the number of buses will also increase the **complexity** of the hardware.

CPU local Bus Organization

- Three-Bus Organization



Three-Bus Datapath

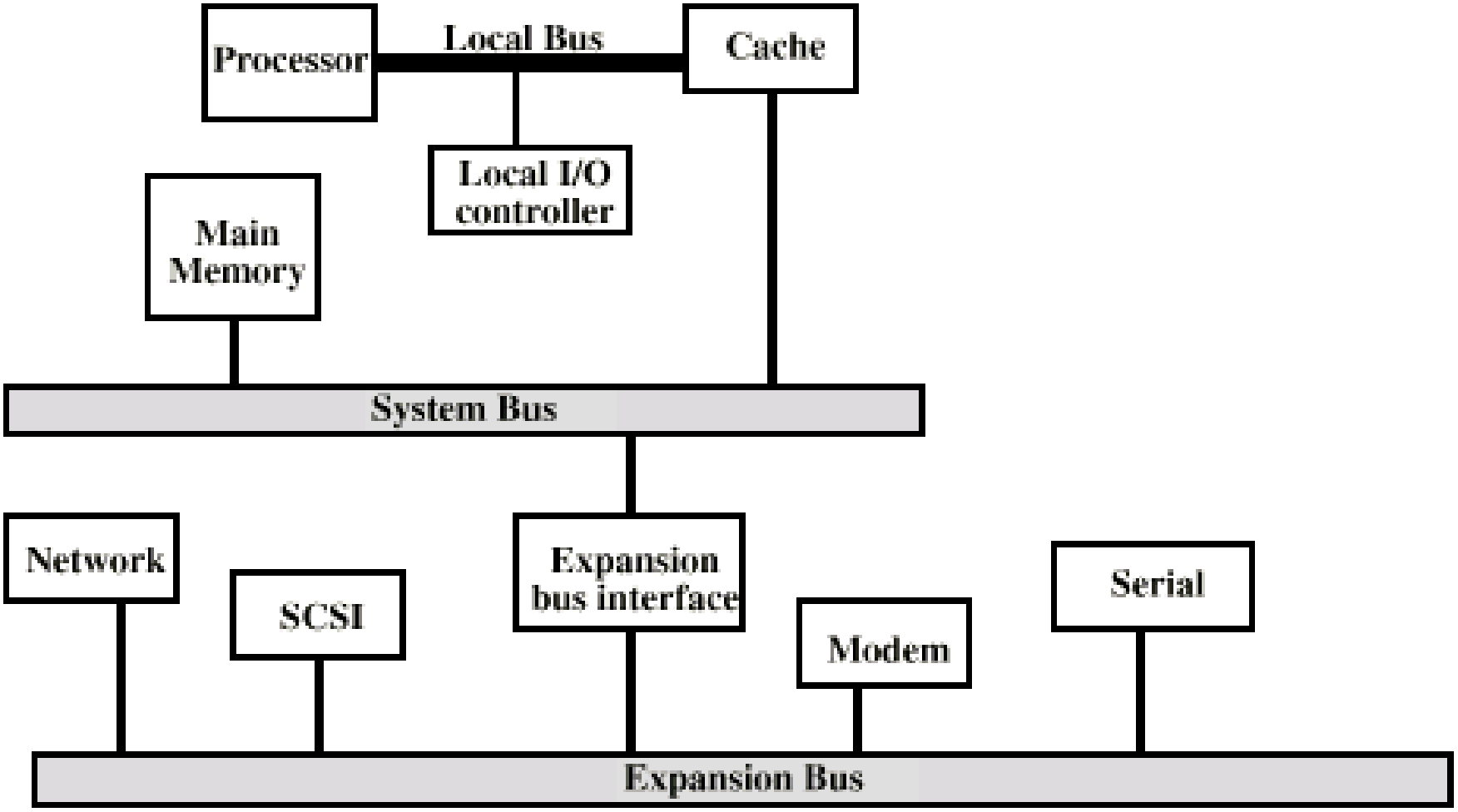
Bus Arbitration

- More than one module controlling the bus
  - e.g. CPU and DMA controller
- Only one module may control bus at one time
- Arbitration may be centralised or distributed
  - ***Centralised*** :Only one module (*bus controller or arbiter*) may control bus at one time
  - ***Distributed*** :More than one module controlling the bus
    - e.g. CPU and DMA controller

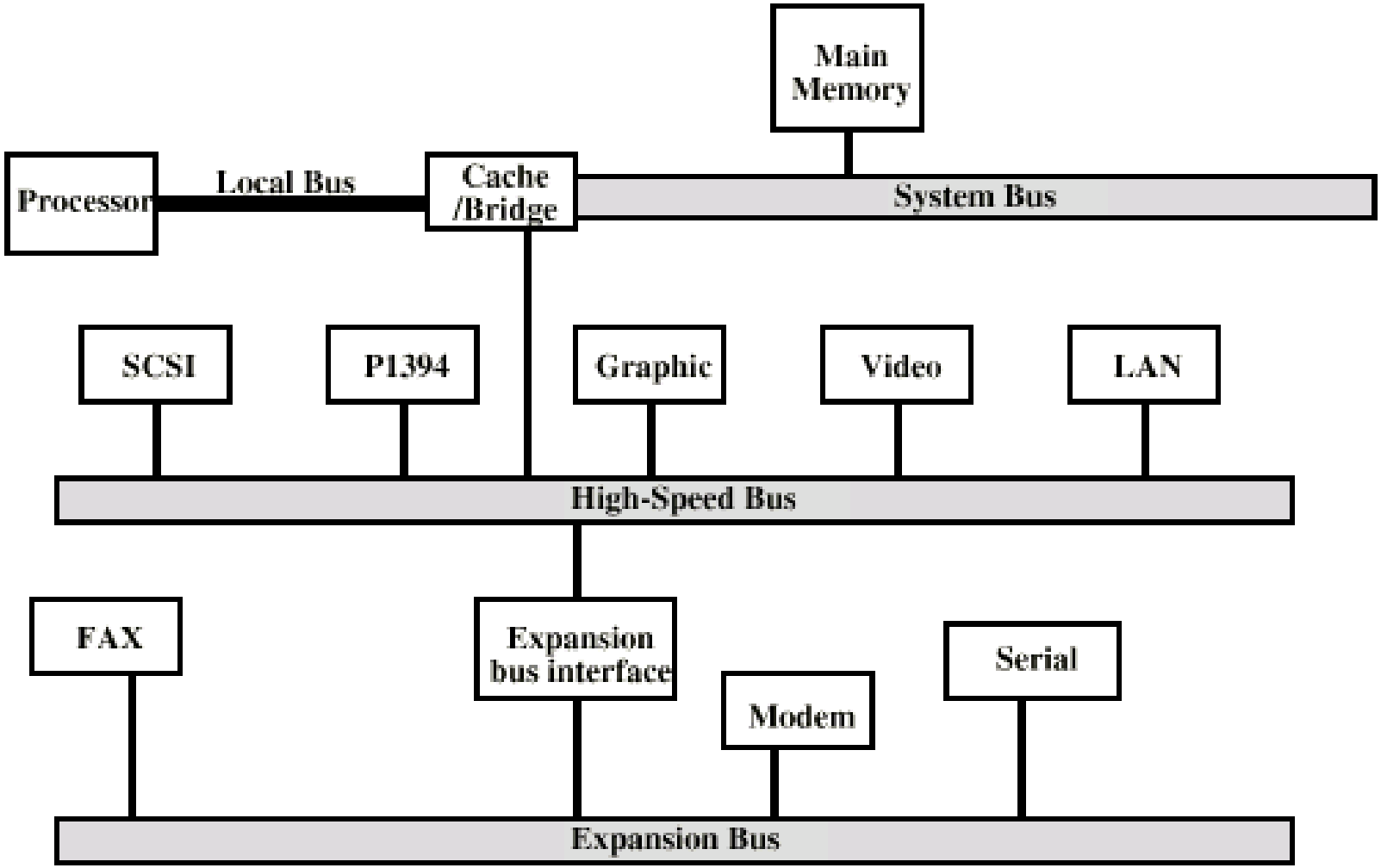
Timing

- Co-ordination of events on bus
- Bus use either *synchronous* or *asynchronous* timing.
- **Synchronous**
  - —Events determined by clock signals
  - —Control Bus includes ***clock line***
  - —A single 1-0 transition is referred to as is a ***bus cycle or clock cycle***
  - —All devices on the bus can read clock line
  - —Usually sync on leading ***edge***
  - —Usually a single cycle for an ***event***

Traditional (ISA) (with cache)



High Performance Bus



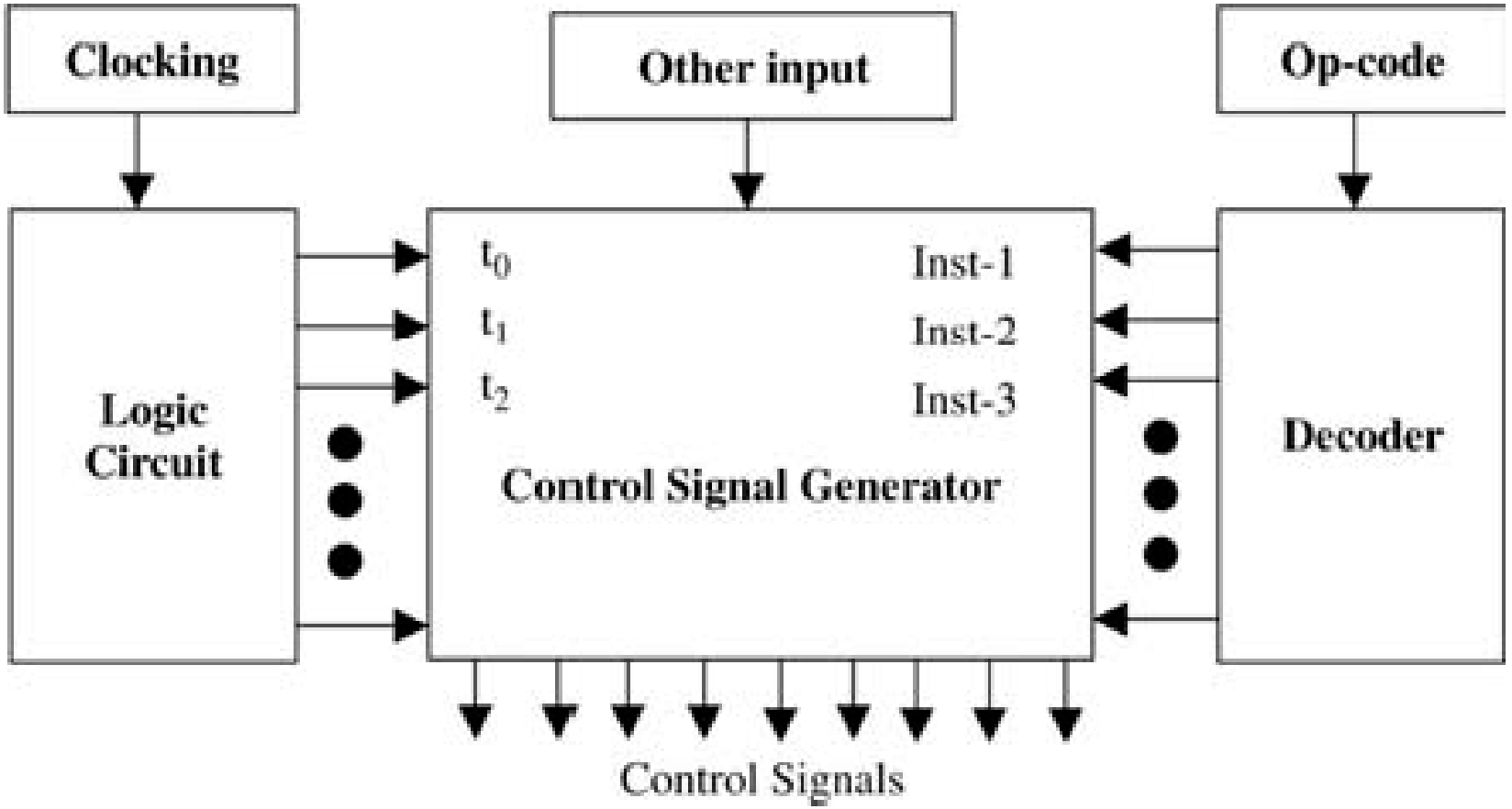
Control Unit

- The control unit is the main component that directs the system operations by **sending control signals** to the datapath.
- ***Datapath:*** The data section, which contains the registers and the ALU.
- These signals control the flow of data within the CPU and between the CPU and external units such as memory and I/O.
- Control buses generally carry signals between the control unit and other computer components in a clock-driven manner.
- The system clock produces a continuous sequence of pulses (timing signals) in a specified duration and frequency.

Control Unit

- A sequence of steps t0 , t1 , t2 , . . . , (t0 < t1 < t2 , . . .) are used to execute a certain instruction.
- The op-code field of a fetched instruction is decoded to provide the control signal generator with information about the instruction to be executed.
- Step information generated by a **logic circuit module** is used with other inputs to generate control signals.
- The signal generator can be specified simply by a set of **Boolean equations** for its output in terms of its inputs.

Control Signal Generator



Control Unit

- There are mainly two different types of control units:
  - Microprogrammed
    - The control signals associated with operations are stored in special memory units inaccessible by the programmer as control words.
  - Hardwired
    - Fixed logic circuits that correspond directly to the Boolean expressions are used to generate the control signals.

Hardwired Implementation

- In hardwired control, a direct implementation is accomplished using logic circuits.
- For each control line, one must find the Boolean expression in terms of the input to the control signal generator
- Let us explain the implementation using simple example.

Hardwired Implementation example

- Assume that the instruction set of a machine has the three instructions: Inst-x, Inst-y, and Inst-z;
- and A, B, C, D, E, F, G, and H are control lines.
- The following table shows the control lines that should be activated for the three instructions at the three steps t0 , t1 , and t2 .

Step	Inst-x	Inst-y	Inst-z
t_0	D, B, E	F, H, G	E, H
t_1	C, A, H	G	D, A, C
t_2	G, C	B, C	

Hardwired Implementation example

Step	Inst-x	Inst-y	Inst-z
t_0	D, B, E	F, H, G	E, H
t_1	C, A, H	G	D, A, C
t_2	G, C	B, C	

The Boolean expressions for control lines A, B, and C can be obtained as follows:

$$\text{A} = \text{Inst-x} \cdot t_1 + \text{Inst-z} \cdot t_1 = (\text{Inst-x} + \text{Inst-z}) \cdot t_1$$

$$\text{B} = \text{Inst-x} \cdot t_0 + \text{Inst-y} \cdot t_2$$

$$\text{C} = \text{Inst-x} \cdot t_1 + \text{Inst-x} \cdot t_2 + \text{Inst-y} \cdot t_2 + \text{Inst-z} \cdot t_1$$

$$= (\text{Inst-x} + \text{Inst-z}) \cdot t_1 + (\text{Inst-x} + \text{Inst-y}) \cdot t_2$$

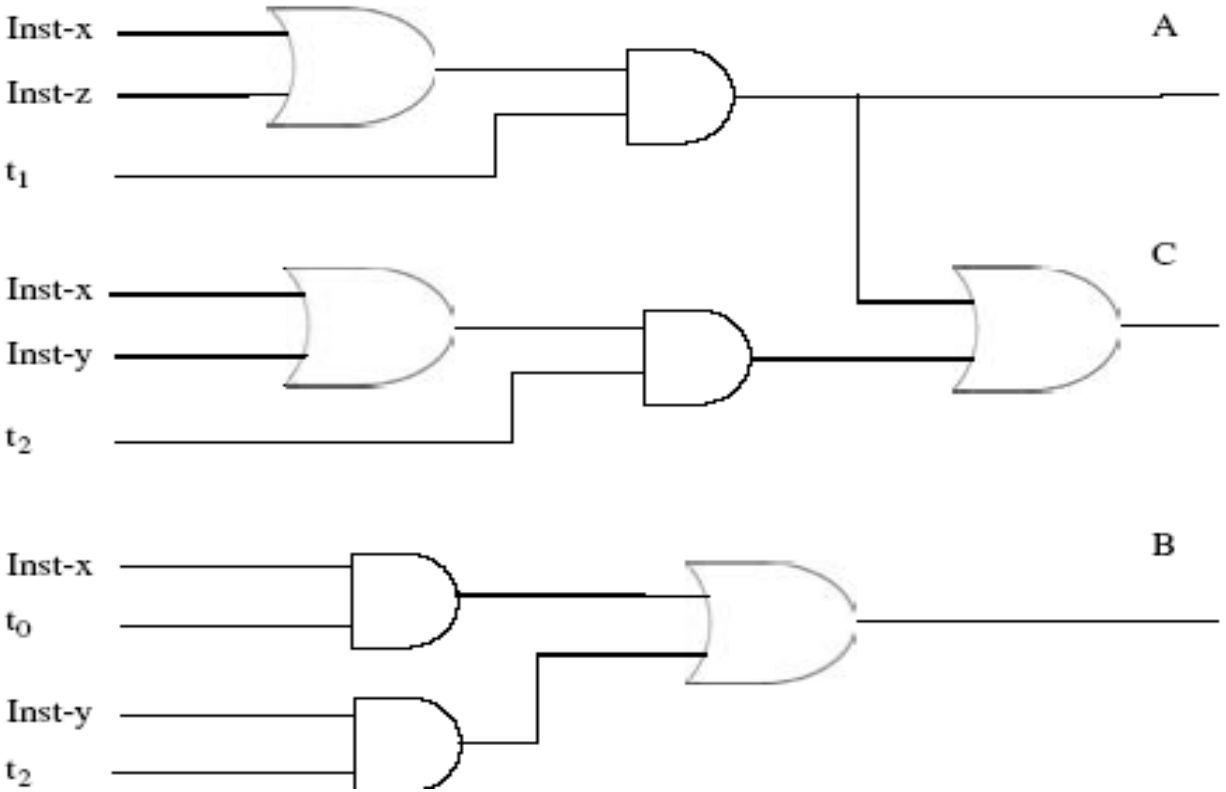
Boolean expressions for the rest of the control lines can be obtained in a similar way.

Hardwired Implementation example

The Boolean expression for control lines A, B and C

$$A = \text{Inst-x} \cdot t_1 + \text{Inst-z} \cdot t_1 = (\text{Inst-x} + \text{Inst-z}) \cdot t_1$$
$$B = \text{Inst-x} \cdot t_0 + \text{Inst-y} \cdot t_2$$
$$C = \text{Inst-x} \cdot t_1 + \text{Inst-x} \cdot t_2 + \text{Inst-y} \cdot t_2 + \text{Inst-z} \cdot t_1$$
$$= (\text{Inst-x} + \text{Inst-z}) \cdot t_1 + (\text{Inst-x} + \text{Inst-y}) \cdot t_2$$

Logic Circuit for control lines A, B and C



Microprogrammed Control Unit

- Microprogramming was motivated by the desire to **reduce the complexities** involved with hardwired control.
- An **instruction** is implemented using a **set of micro-operations**.
- Associated with each **micro-operation** is a **set of control lines** that must be **activated** to carry out the corresponding microoperation.
- The idea of microprogrammed control is to **store** the **control signals** associated with the implementation of a certain instruction as a microprogram in a **special memory** called a control memory (CM).

Microprogrammed Control Unit

- A microprogram consists of a sequence of microinstructions.
  - A microinstruction is a **vector of bits**, where each bit is a control signal, condition code, or the address of the next microinstruction.
  - Microinstructions are fetched from CM the same way program instructions are fetched from main memory

- When an instruction is fetched from memory, the **op-code** field of the instruction will **determine which microprogram** is to be executed.