Chapter 7

Input/Output

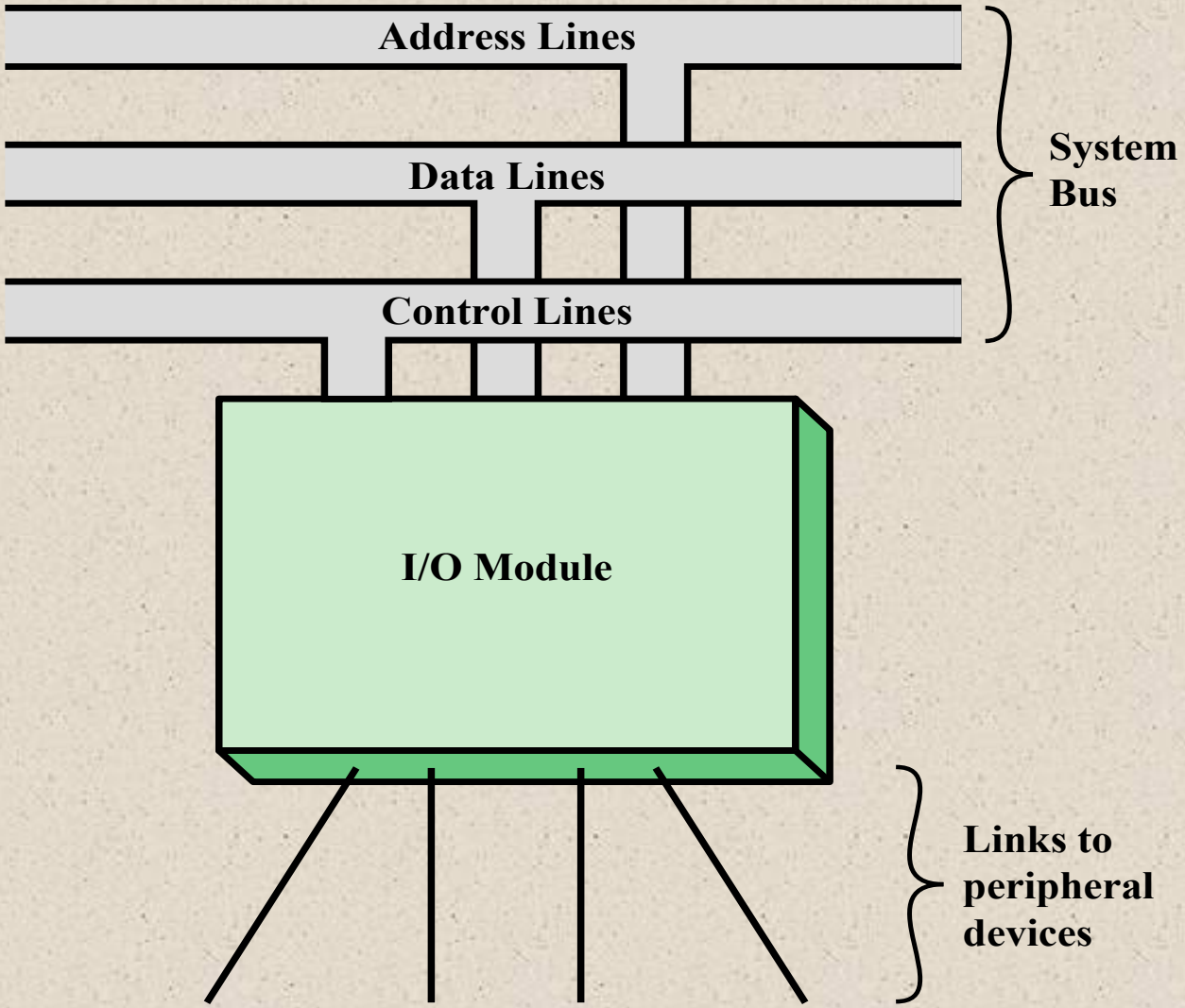


Figure 7.1 Generic Model of an I/O Module

External Devices

- Provide a means of **exchanging data** between the external environment and the computer
- Attach to the computer by a **link to an I/O module**
  - The link is used **to exchange control, status, and data** between the I/O module and the external device
- ***Peripheral device***
  - **An external device** connected to an I/O module

Three categories:

- **Human readable**
  - Suitable for communicating with the computer user
  - Video display terminals **(VDTs), printers**
- **Machine readable**
  - Suitable for communicating with equipment
  - **Magnetic disk and tape systems, sensors and actuators**
- **Communication**
  - Suitable for **communicating with remote devices** such as a terminal, a machine readable device, or another computer

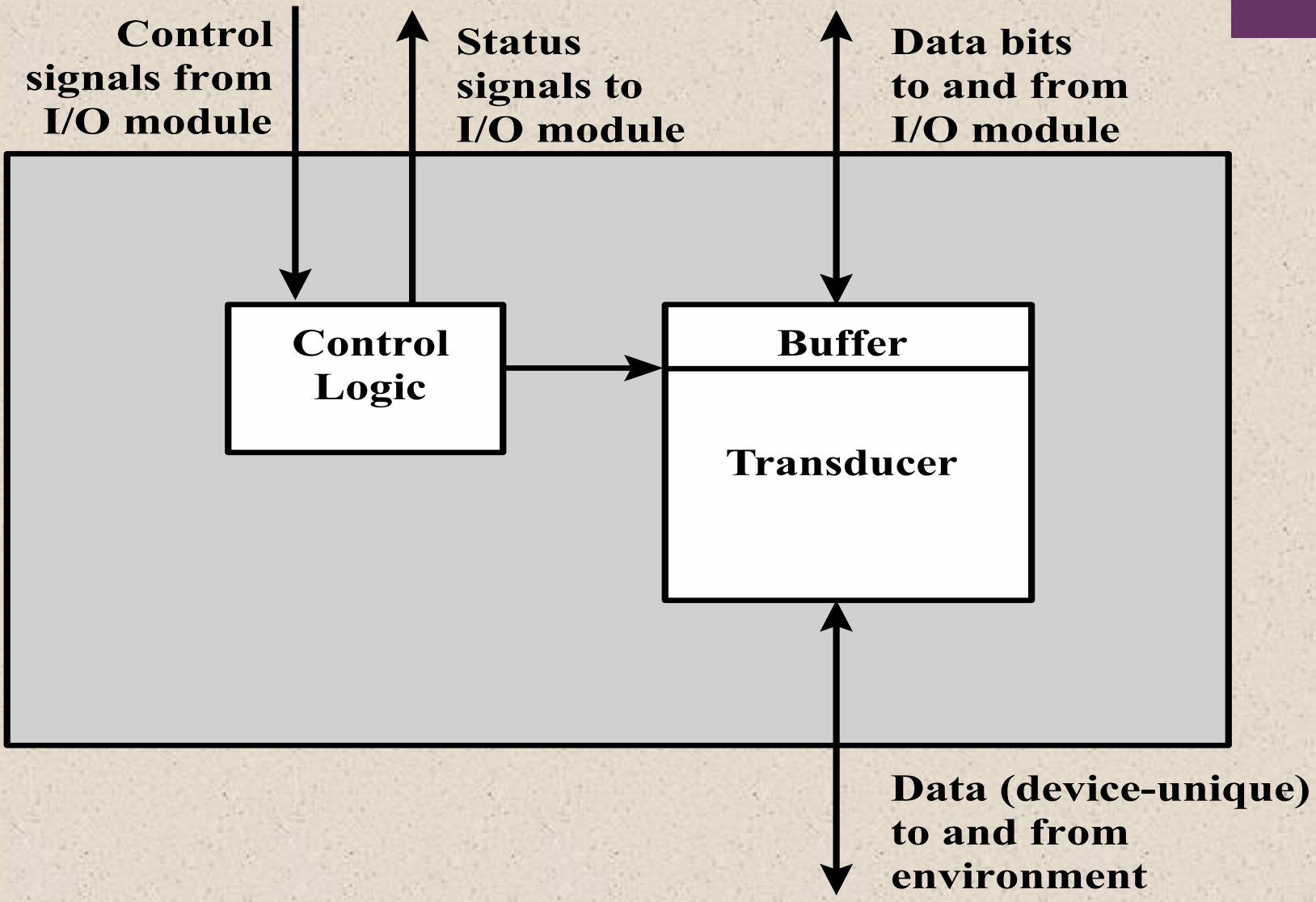


Figure 7.2 Block Diagram of an External Device

Keyboard/Monitor

Most common means of computer/user interaction

User provides input through the keyboard

The monitor displays data provided by the computer

International Reference Alphabet (IRA)

- **Basic unit of exchange is the character**
  - Associated with each character is a code
  - **Each character** in this code is represented by a unique **7-bit binary code**
    - **128 different characters** can be represented
- **Characters are of two types:**
  - **Printable**
    - Alphabetic, numeric, and special characters that can be printed on paper or displayed on a screen
  - **Control**
    - Have to do with **controlling the printing or displaying of characters**
    - Example is **carriage return**
    - Other control characters are concerned with communications procedures

Keyboard Codes

- When the user depresses a key it generates an electronic signal that is interpreted by the **transducer in the keyboard** and **translated into the bit pattern** of the corresponding IRA code
- This **bit pattern is transmitted to the I/O module** in the computer
- On output, IRA code characters are transmitted to an external device from the I/O module
- The transducer interprets the code and sends the required electronic signals to the output device either to display the indicated character or perform the requested control function

The major functions for an I/O module fall into the following categories:

Control and timing

- Coordinates the flow of traffic between internal resources and external devices

Processor communication

- Involves command decoding, data, status reporting, address recognition

Device communication

- Involves commands, status information, and data

Data buffering

- Performs the needed buffering operation to balance device and memory speeds

Error detection

- Detects and reports transmission errors

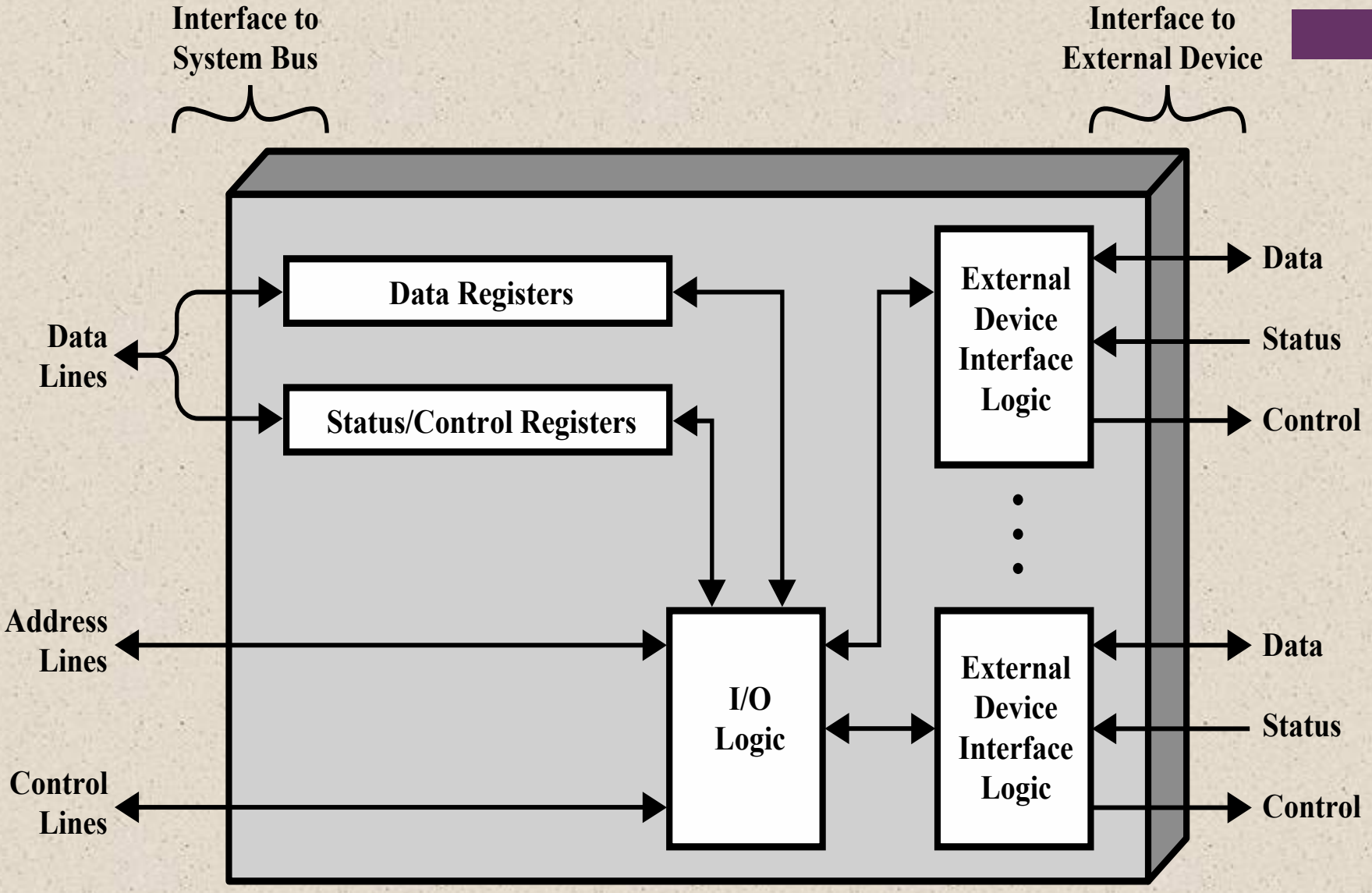


Figure 7.3 Block Diagram of an I/O Module

Programmed I/O

Three techniques are possible for I/O operations:

- **Programmed I/O**
  - Data are exchanged between the processor and the I/O module
  - Processor executes a program that gives it direct control of the I/O operation
  - When the processor issues a command it must wait until the I/O operation is complete
  - If the processor is faster than the I/O module this is wasteful of processor time
- **Interrupt-driven I/O**
  - Processor issues an I/O command, continues to execute other instructions, and is interrupted by the I/O module when the latter has completed its work
- **Direct memory access (DMA)**
  - The I/O module and main memory exchange data directly without processor involvement

Table 7.1

I/O Techniques

	No Interrupts	Use of Interrupts
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct memory access (DMA)

I/O Commands

- There are **four types of I/O commands that an I/O module may receive when it is addressed by a processor**:

  1) **Control**

  - used to activate a peripheral and tell it what to do

  2) Test

  - used to test various status conditions associated with an I/O module and its peripherals

  3) **Read**

  - causes the I/O module to obtain an item of data from the peripheral and place it in an internal buffer

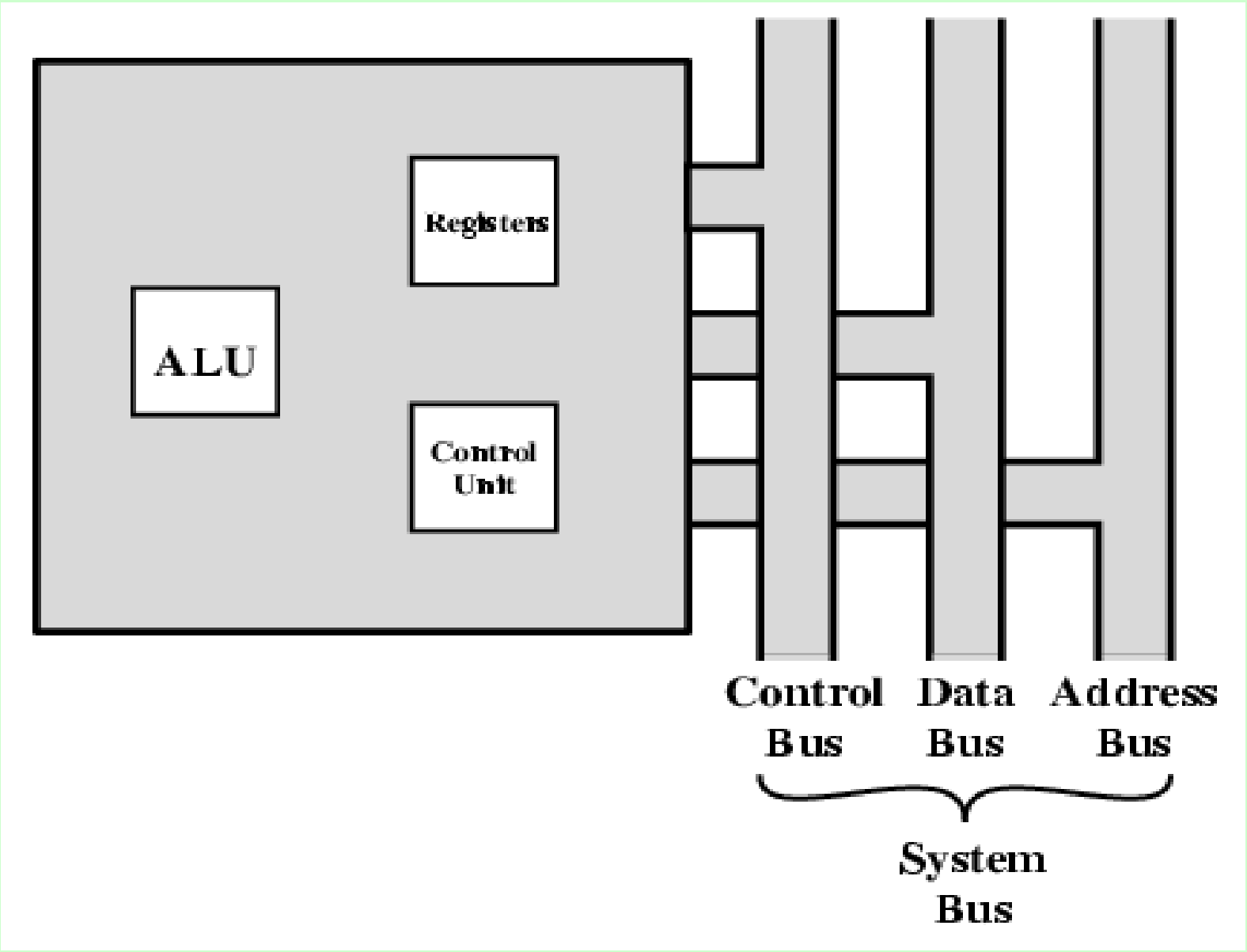
  4) **Write**

  - causes the I/O module to take an item of data from the data bus and subsequently transmit that data item to the peripheral

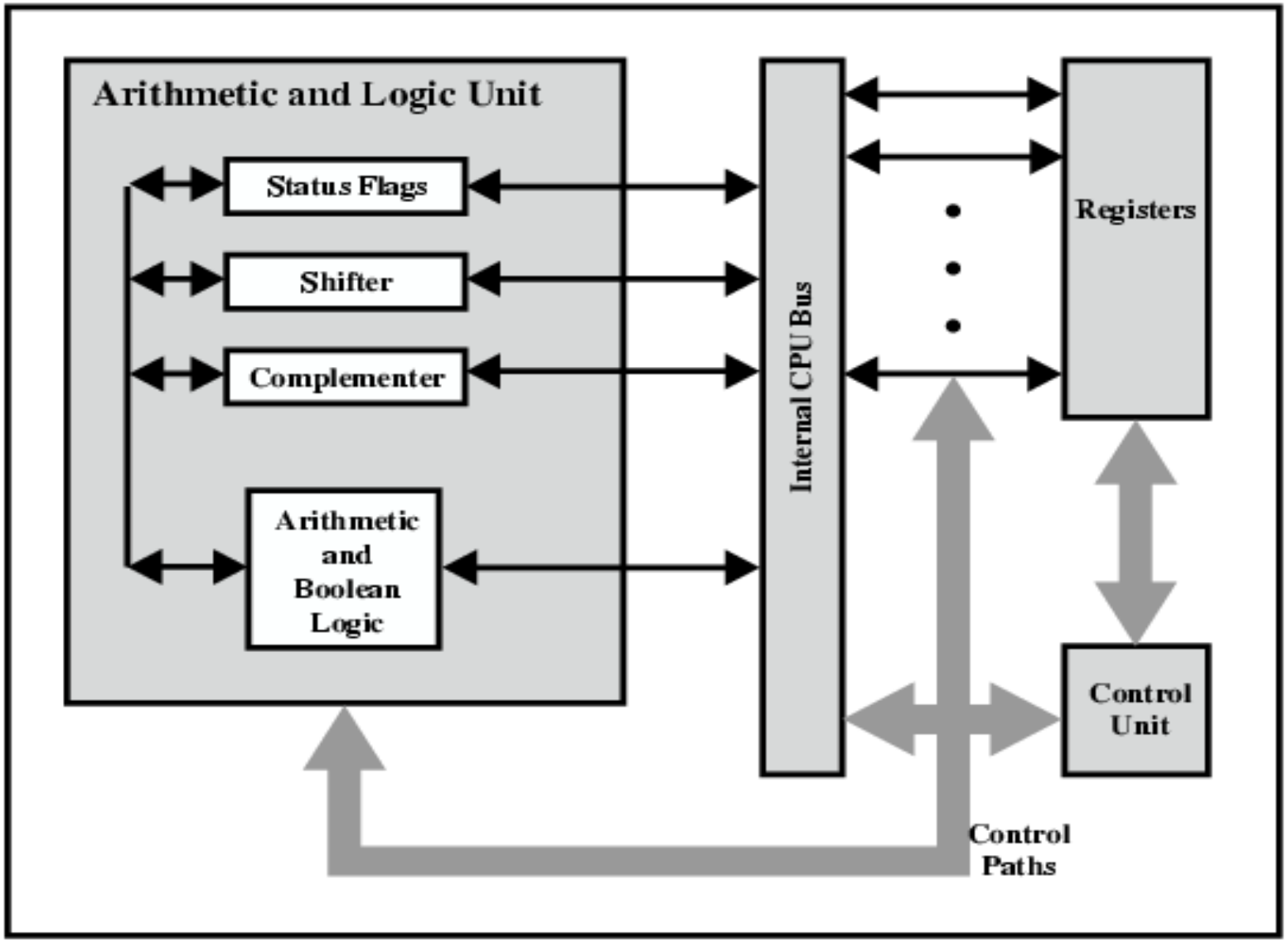
Chapter 15

Control Unit Operation

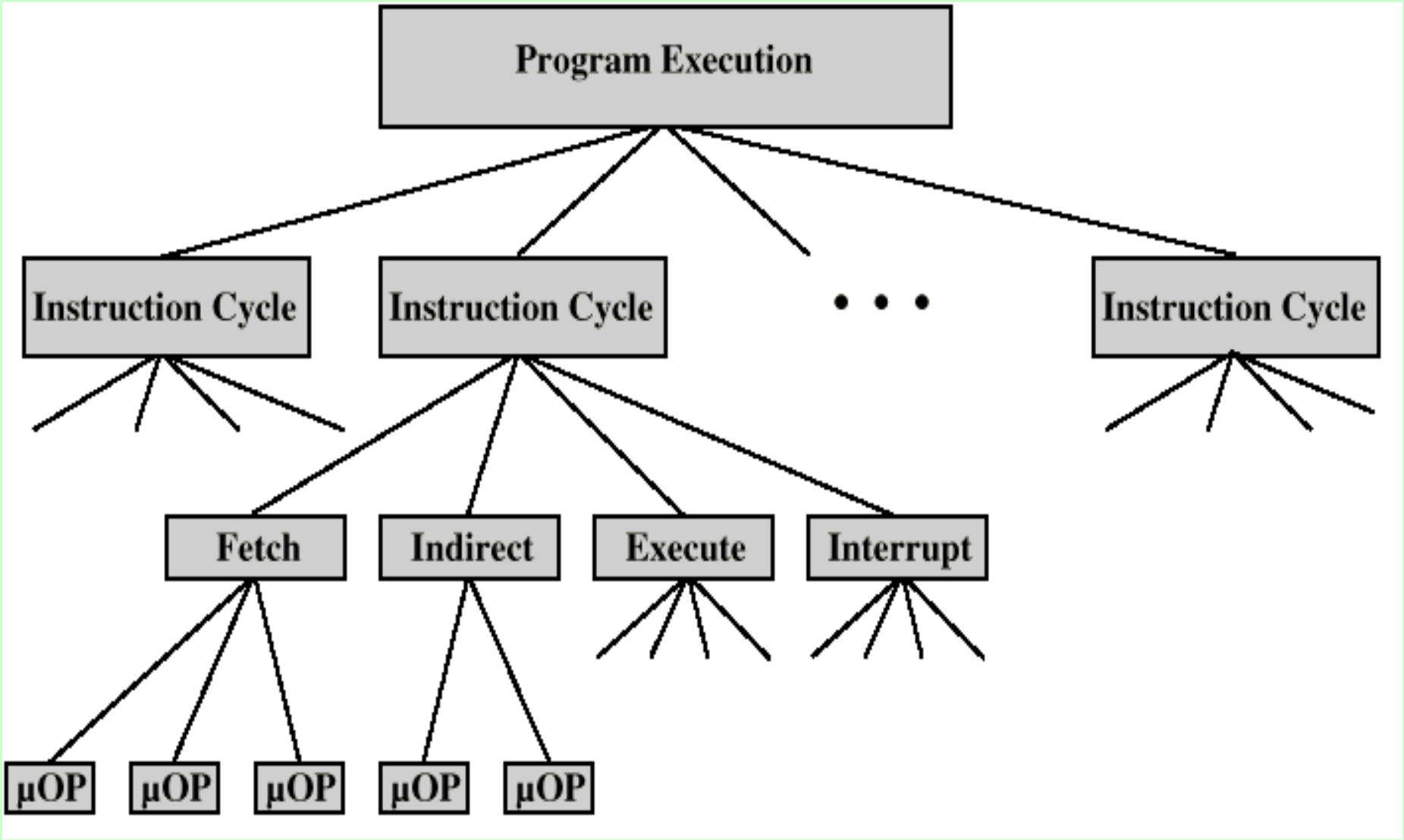
CPU With Systems Bus



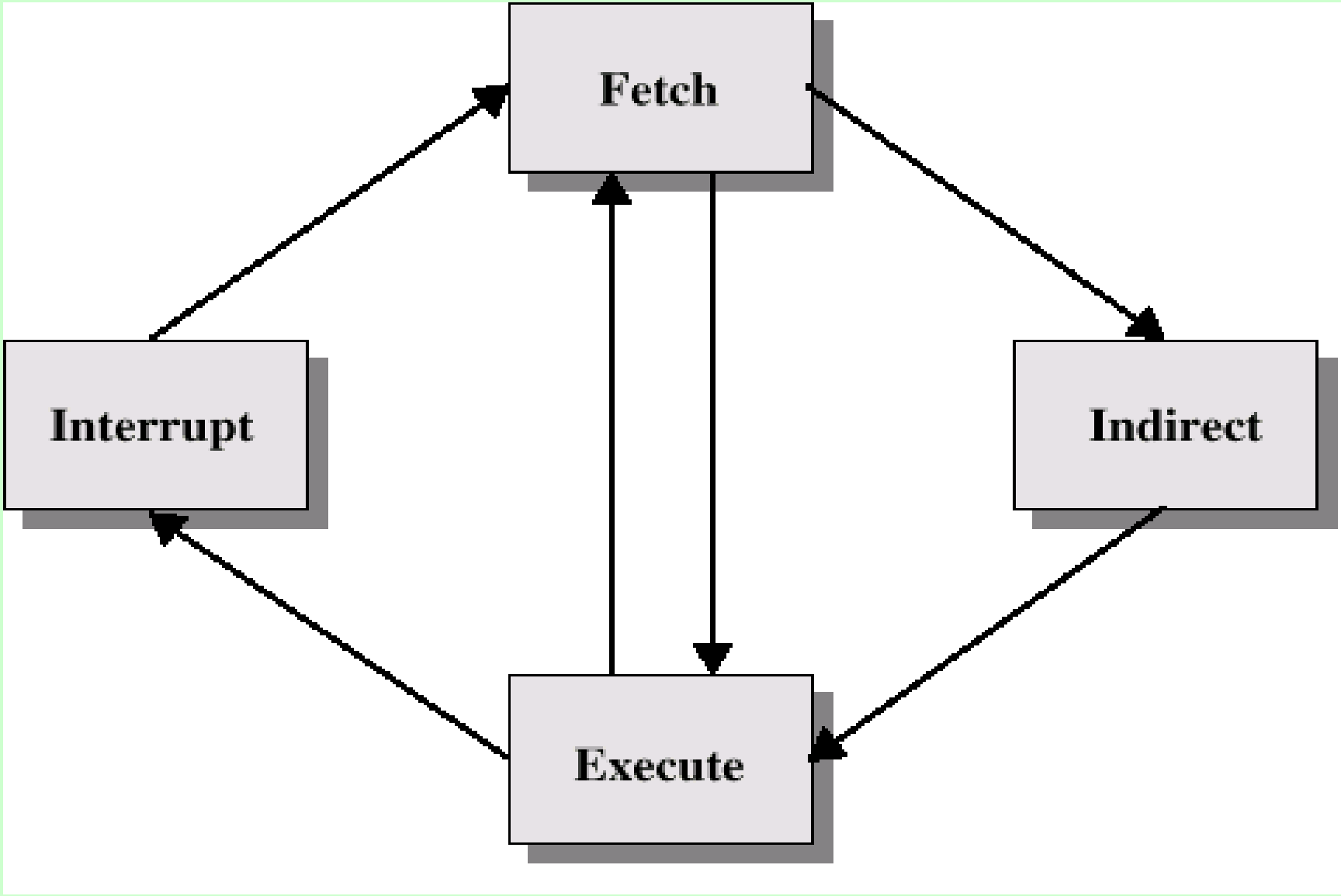
CPU Internal Structure



Constituent Elements of Program Execution



Instruction Cycle with Indirect



Fetch - 4 Registers

- **Memory Address Register (MAR)**
  - —Connected to address bus
  - —Specifies address for read or write op
- **Memory Buffer Register (MBR)**
  - —Connected to data bus
  - —Holds data to write or last data read
- **Program Counter (PC)**
  - —Holds address of next instruction to be fetched
- **Instruction Register (IR)**
  - —Holds last instruction fetched

Data Flow (Instruction Fetch)

- Fetch
  - PC contains address of next instruction
  - Address moved to MAR
  - Address placed on address bus
  - Control unit requests memory read
  - Result placed on data bus, copied to MBR, then to IR
  - Meanwhile PC incremented by 1

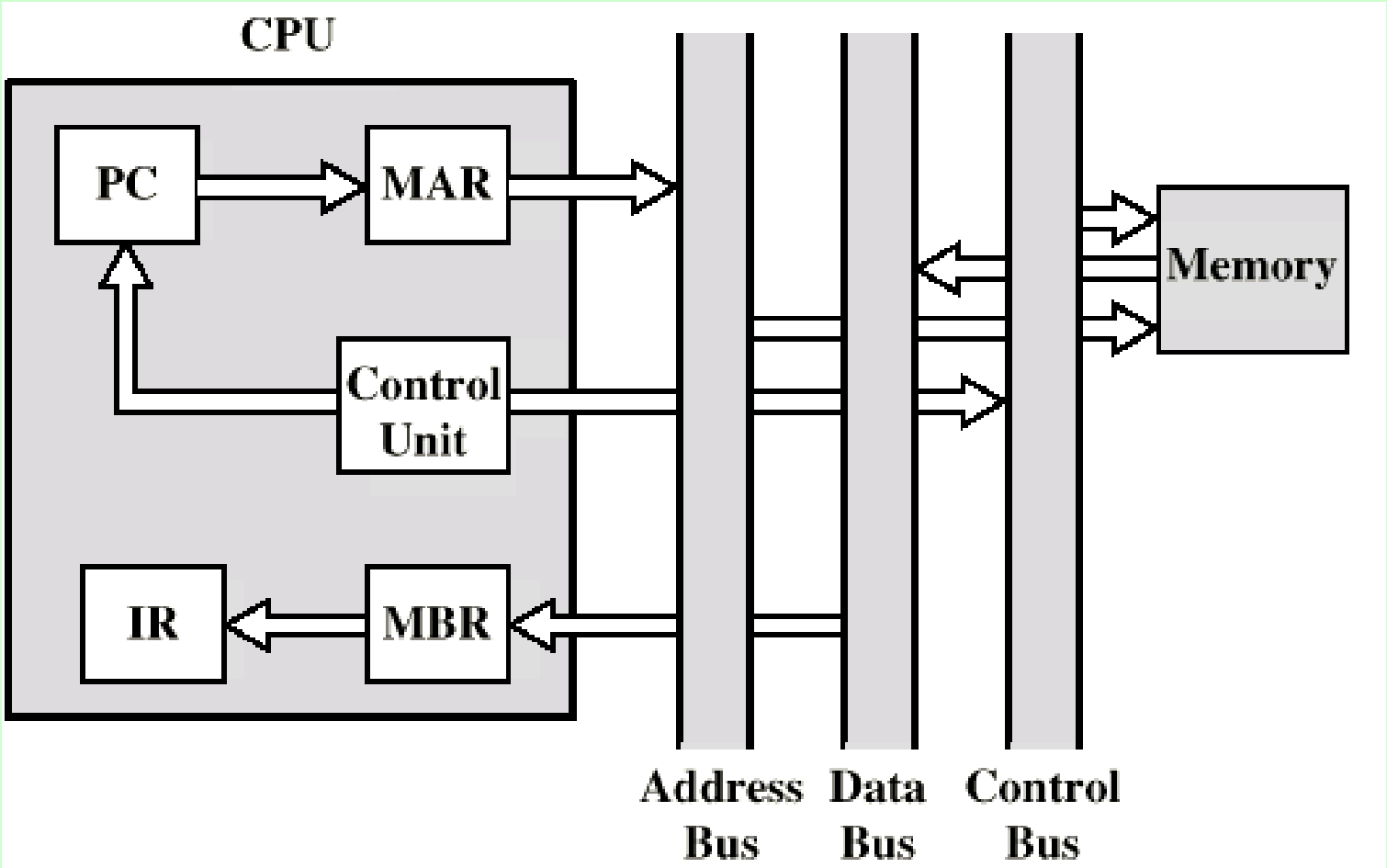
Fetch Sequence

- **Address of next instruction is in PC**
- **Address (MAR) is placed on address bus**
- Control unit issues **READ command**
- Result (**data from memory) appears on data bus**
- **Data from data bus copied into MBR**
- **PC incremented by 1** (in parallel with data fetch from memory)
- **Data (instruction) moved from MBR to IR**
- MBR is now free for further data fetches

Fetch Sequence (symbolic)

- **t1:MAR <- (PC)**
- **t2:MBR <- (memory)**
- **PC <- (PC) +1**
- **t3:IR <- (MBR)**
  - (tx = time unit/clock cycle)
  - or
    - t1: MAR <- (PC)
    - t2: MBR <- (memory)
    - t3: PC <- (PC) +1
    - IR <- (MBR)

Data Flow (Fetch Diagram)



MBR = Memory buffer register
MAR = Memory address register
IR = Instruction register
PC = Program counter

Rules for Clock Cycle Grouping

- **Proper sequence** must be followed
  - **MAR <- (PC) must precede MBR <- (memory)**
- **Conflicts** must be **avoided**
  - **Must not read & write same register at same time**
  - **MBR <- (memory) & IR <- (MBR) must not be in same cycle**
- Also: PC <- (PC) +1 involves addition
  - Use ALU
  - May need additional micro-operations

Indirect Cycle

- **May require memory access to fetch operands**
- **Indirect addressing requires more memory accesses**
- **Can be thought of as additional instruction subcycle**

Indirect Cycle

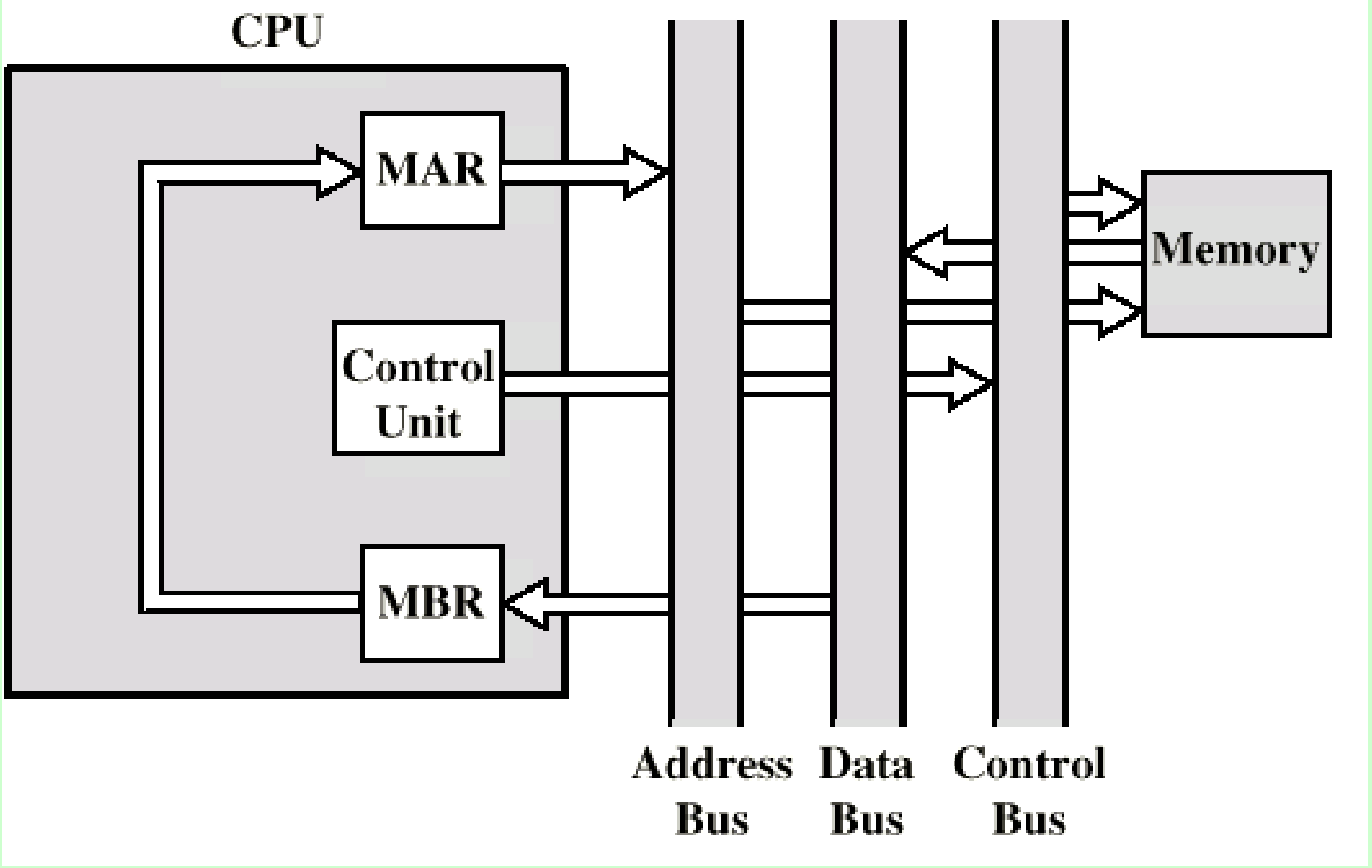
- **MAR <- ($IR_{address}$) - address field of IR**
- **MBR <- (memory)**
- **$IR_{address}$ <- ($MBR_{address}$)**

- MBR contains an address
- IR is now in same state as if direct addressing had been used
- (What does this say about IR size?)

Data Flow (Data Fetch)

- IR is examined
- If indirect addressing, indirect cycle is performed
  - Right most N bits of MBR transferred to MAR
  - Control unit requests memory read
  - Result (address of operand) moved to MBR

Data Flow (Indirect Diagram)



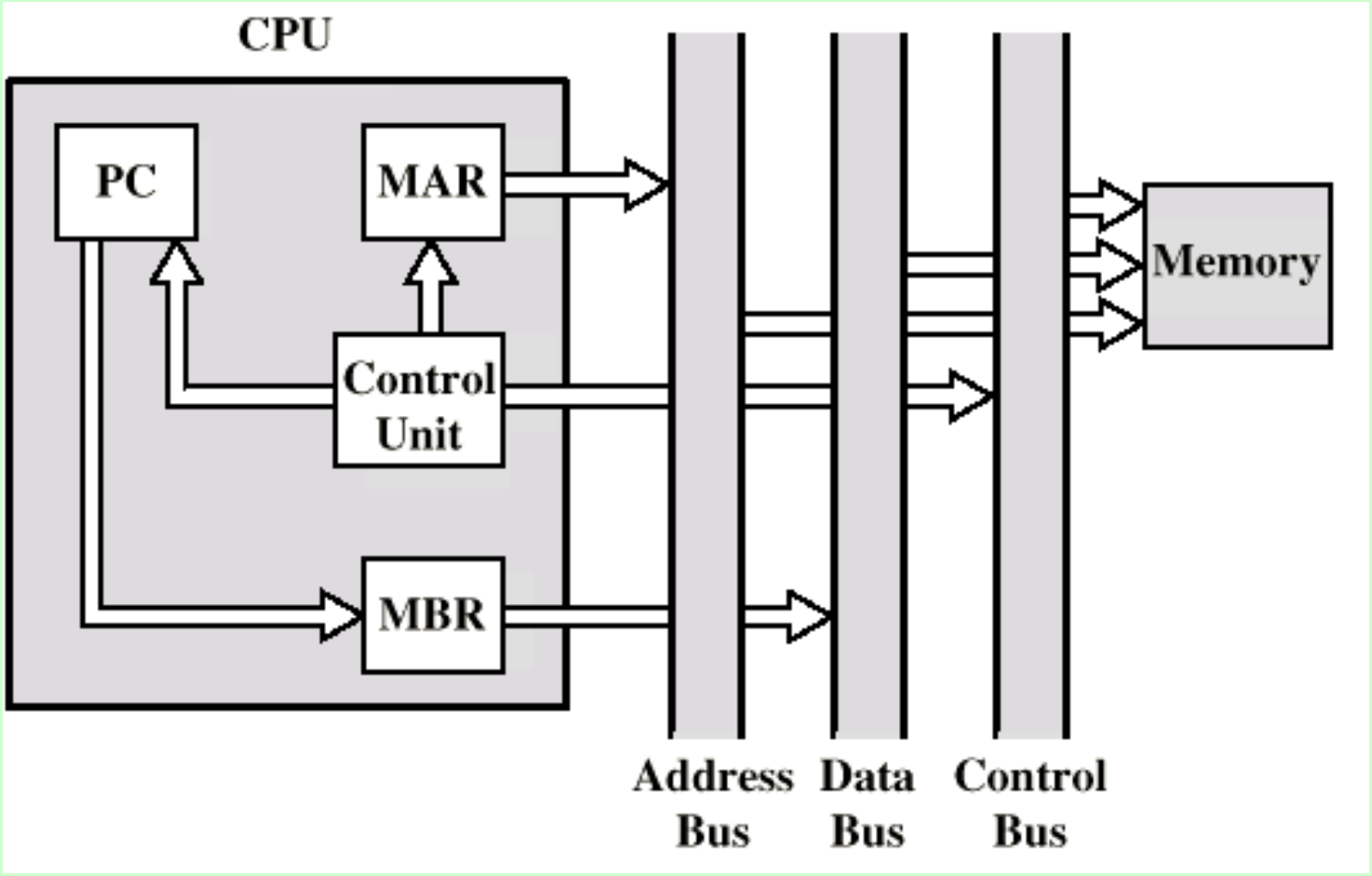
Interrupt Cycle

- **t1:MBR <-(PC)**
- **t2:MAR <- save-address**
- **PC <- routine-address**
- **t3:memory <- (MBR)**
- This is a minimum
  - —May be additional micro-ops to get addresses
  - —N.B. saving context is done by interrupt handler routine, not micro-ops

Data Flow (Interrupt)

- Simple
- Predictable
- Current PC saved to allow resumption after interrupt
- Contents of PC copied to MBR
- Special memory location (e.g. stack pointer) loaded to MAR
- MBR written to memory
- PC loaded with address of interrupt handling routine
- Next instruction (first of interrupt handler) can be fetched

Data Flow (Interrupt Diagram)



Data Flow (Execute)

- May take many forms
- Depends on instruction being executed
- May include
  - Memory read/write
  - Input/Output
  - Register transfers
  - ALU operations

Execute Cycle (ADD)

- Different for each instruction
- e.g. ADD R1,X - add the contents of location X to Register 1 , result in R1
- t1: MAR <- ($IR_{address}$)
- t2: MBR <- (memory)
- t3: R1 <- R1 + (MBR)
- Note no overlap of micro-operations

Execute Cycle (ISZ)

- ISZ X - increment and skip if zero
  - t1: MAR <- ($IR_{address}$)
  - t2: MBR <- (memory)
  - t3: MBR <- (MBR) + 1
  - t4: memory <- (MBR)
  - if (MBR) == 0 then PC <- (PC) + 1
- Notes:
  - if is a single micro-operation
  - Micro-operations done during t4

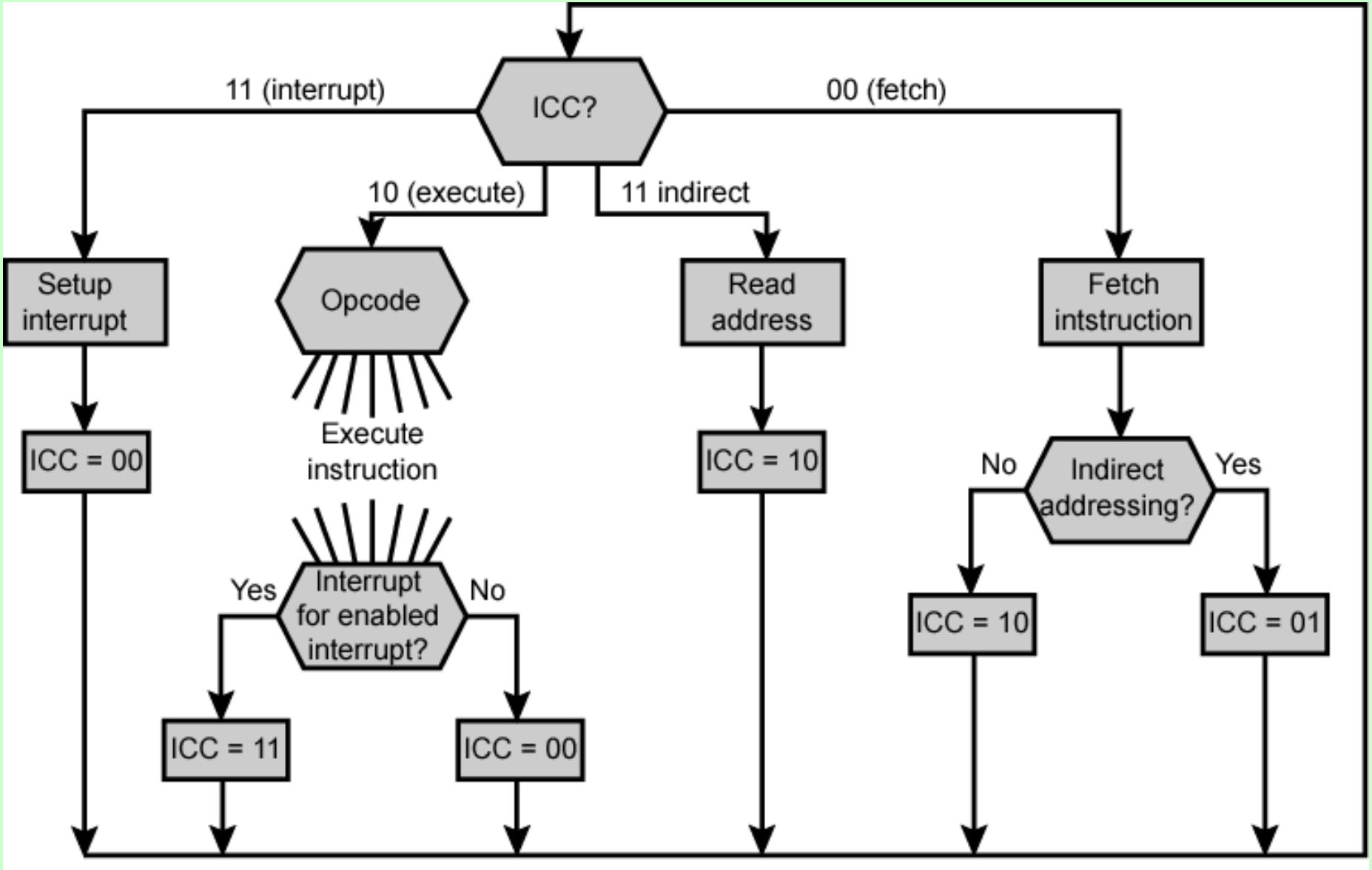
Execute Cycle (BSA)

- BSA X - Branch and save address
  - Address of instruction following BSA is saved in X
  - Execution continues from X+1
  - t1: MAR <- ($IR_{address}$)
  - MBR <- (PC)
  - t2: PC <- ($IR_{address}$)
  - memory <- (MBR)
  - t3: PC <- (PC) + 1

Instruction Cycle

- **Each phase decomposed into sequence of elementary micro-operations**
- **E.g. fetch, indirect, and interrupt cycles**
- **Execute cycle**
  - **One sequence of micro-operations for each opcode**
- Need to tie sequences together
- **Assume new 2-bit register**
  - **Instruction cycle code (ICC)** designates which part of cycle processor is in
    - **00: Fetch**
    - **01: Indirect**
    - **10: Execute**
    - **11: Interrupt**

Flowchart for Instruction Cycle



Types of Micro-operation

- Transfer data between registers
- Transfer data from register to external
- Transfer data from external to register
- Perform arithmetic or logical ops

Functions of Control Unit

- **Sequencing**
  - —Causing the CPU to step through a series of micro-operations
- **Execution**
  - —Causing the performance of each micro-op
- **This is done using Control Signals**

Control Signals

- **Clock**
  - **One micro-instruction** (or set of parallel micro-instructions) **per clock cycle**
- **Instruction register**
  - **Op-code for current instruction**
  - Determines which micro-instructions are performed
- **Flags**
  - **State of CPU**
  - **Results of previous operations**
- **From control bus**
  - **Interrupts**
  - **Acknowledgements**

Model of Control Unit