CSE 4305
Computer Organization and Architecture

Control Unit Operation

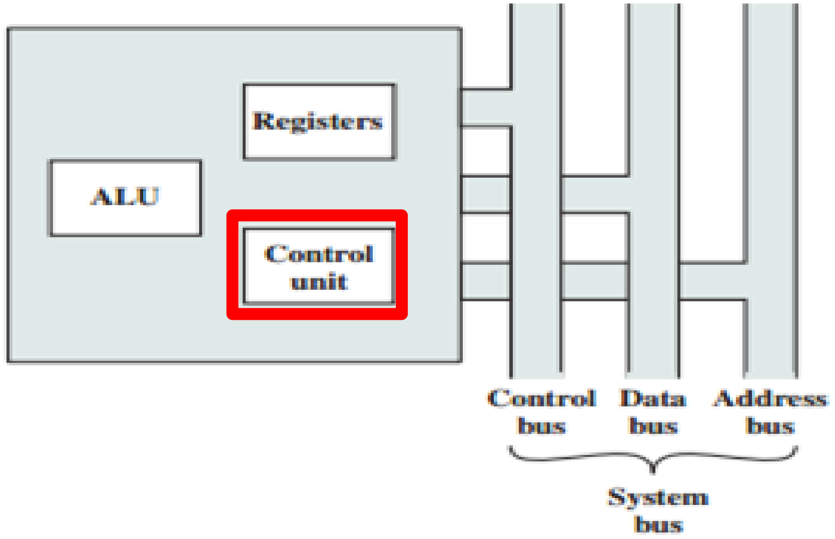
Course Teacher: Md. Hamjajul Ashmafee

Assistant Professor, CSE, IUT

Email: ashmafee@iut-dhaka.edu

Control Unit *of a Computer System*

- Now we need to know ***how*** these functions are performed or controlled by a processor –
  - So far, we have learned the answer of the question “**what**” a computer/processor performs
  - Control unit is the **engine** that runs the entire computer
    - **Answer of How?** - Control unit controls the operation of a computer based on **knowing** the instructions provided to it as well as the nature of the results of last arithmetic and logical operations
      - It never gets **into the data** how it being processed or which results being produced



Execution of Instructions

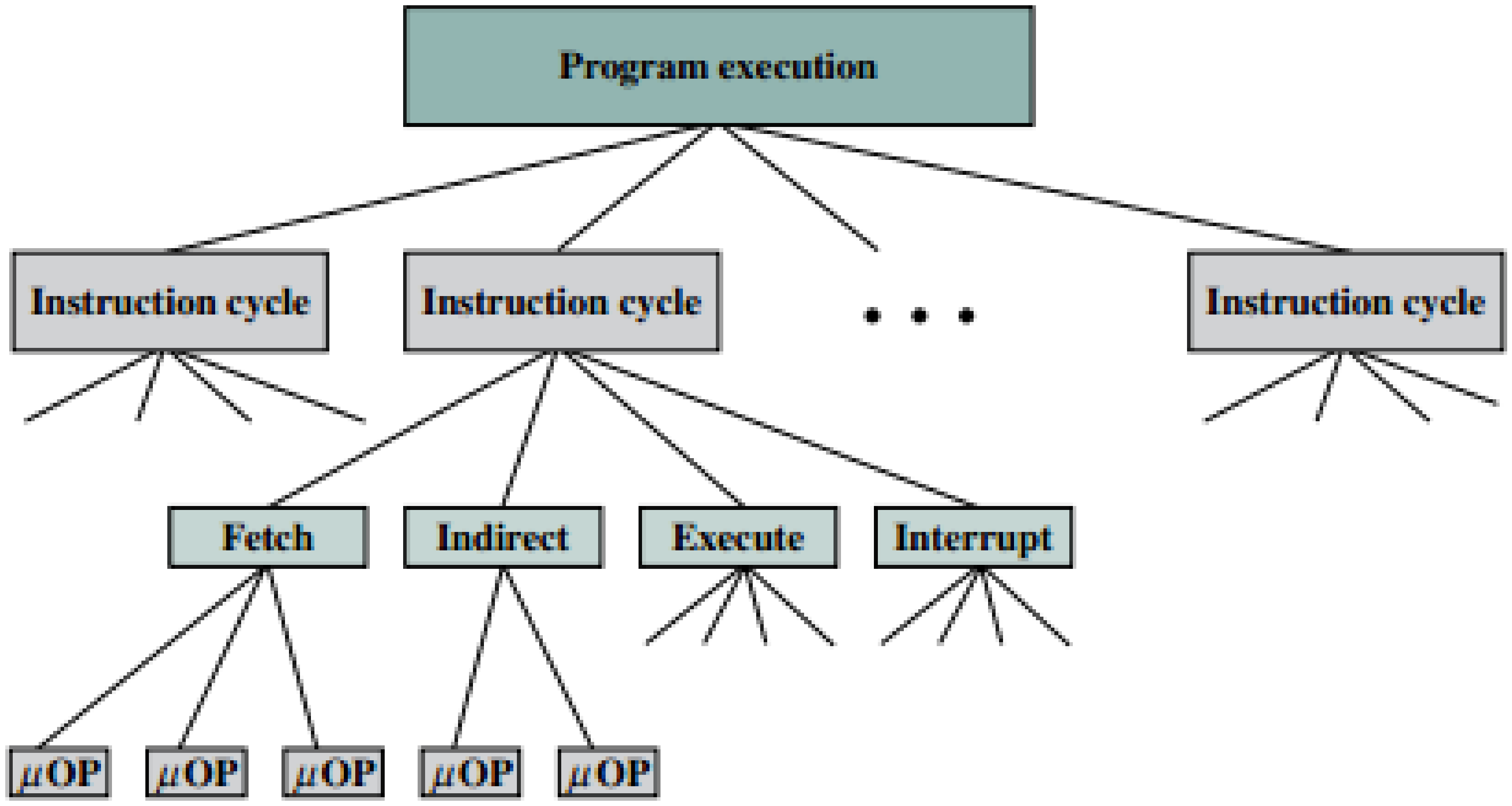
- When a **program** is executed, this operation consists of several instruction cycles
  - Through each instruction cycle, one **machine instruction** is executed and one user command will be fulfilled.
- This **sequence** of instruction cycles is not necessarily same as written sequence of instructions within a program
  - Because the written sequence of instructions of a program can be broken if a branching occurs or because of other issues (e.g., interrupts)
  - **Rather** we will refer this **sequence** of instruction cycles as ***execution time sequence*** of instructions
- Again, these instruction cycles are made of even several **smaller** **units/phases**
  - Smaller units like ***fetch***, ***indirect***, ***execute*** and ***interrupt***
  - Here, ***fetch*** and ***execute*** phases are very common for all instruction cycles

Micro-operations

- To **design** a control unit, we need to **decompose** these phases **even smaller**
- So, these sub-cycles (or phases) are made of a series of further **smaller steps**
  - These are referred to **MICRO-OPERATIONS**
  - ***"Micro"*** refers that each step is very small, simple and accomplishes a very atomic and functional task through processor
  - Each of micro-operations, involves **processor registers** to store and move temporary data

Decomposition of Execution of a Program

(With a notion of micro-operation)



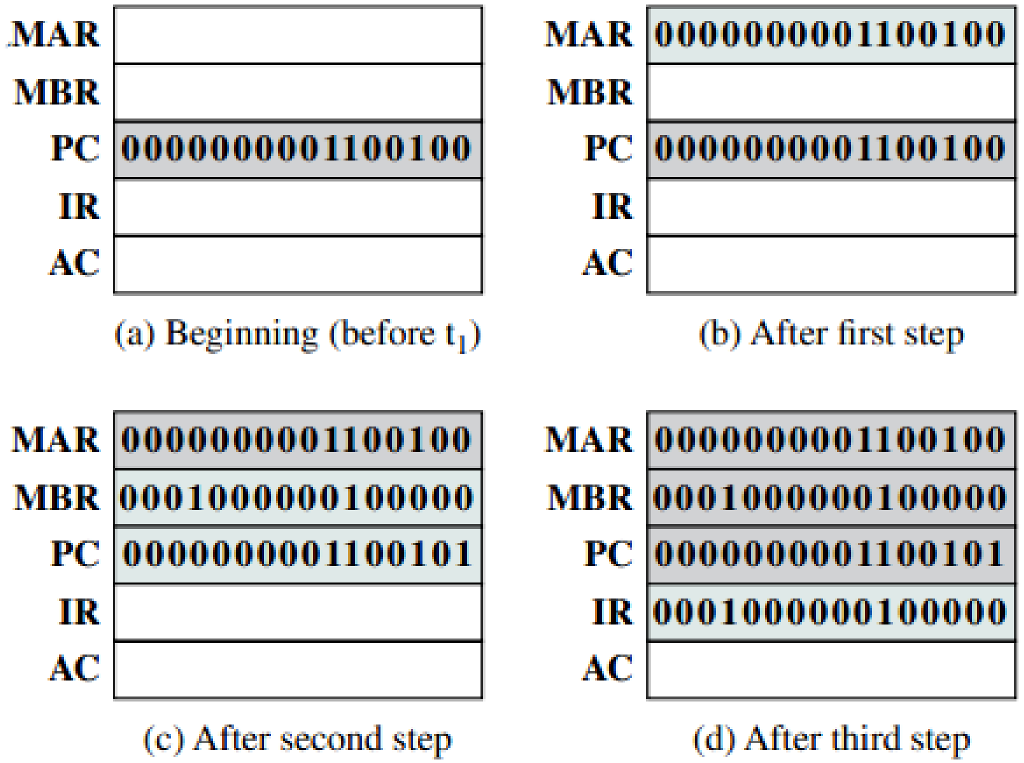
Fetch Cycle: Sequence of Events

At the beginning of the fetch cycle (***even before the instruction cycle starts***), the address of the next instruction to be executed should be in the ***program counter*** register (**PC**)

1. **First** step is to move this address from ***PC*** to ***memory address register*** (**MAR**)
   - As the MAR is the only register connected with the address lines of the system bus
2. **Next**, to bring the instruction in the processor from the memory
   - Control unit issues a **READ command** on the control bus
   - As a result, the desired data is appeared on the data lines of the system bus
   - Then, it is copied to ***memory buffer register*** (**MBR**) from data bus
3. **Meanwhile**, PC needs to be incremented by <u>unit instruction length</u> to get ready for the next instruction*****
   - **Note:** If two or more **events** do not interfere with each other, we can execute them simultaneously to save time
   - Here, event 3 and 4 can be done simultaneously or we may try it any other way
4. **Last**, to move the content of MBR to ***instruction register*** (**IR**)
   - So now MBR is free to do other tasks

Fetch Cycle: At a Glance

- This simple **fetch cycle** is made of **3 steps** and **4 micro-operations**
    - Each micro-operation causes to **move** data from or to the register(s) [details in **figure**]



MAR	
MBR	
PC	0000000001100100
IR	
AC	

(a) Beginning (before t_1)

MAR	0000000001100100
MBR	
PC	0000000001100100
IR	
AC	

(b) After first step

MAR	0000000001100100
MBR	0001000000100000
PC	0000000001100101
IR	
AC	

(c) After second step

MAR	0000000001100100
MBR	0001000000100000
PC	0000000001100101
IR	0001000000100000
AC	

(d) After third step

Fetch Cycle: In Time Sequence

- As **clock** is also available in control unit, it generates successive **clock pulses** which define time units of equal duration (t_1, t_2, t_3, …)
    - Each micro operation can be performed within single time unit (*as per expectation*)
- Fetch cycle in **symbolic form** of micro operations:

$$
\begin{array}{lll}
t_1: & MAR & \leftarrow (PC) \\
t_2: & MBR & \leftarrow \text{Memory} \\
 & PC & \leftarrow (PC) + I \\
t_3: & IR & \leftarrow (MBR)
\end{array}
$$

[Where I = unit instruction length]

Indirect Cycle

- After finishing a fetch cycle, next step is to **fetch one or more source operands** to perform the operation
- If the instruction specifies an indirect address, an indirect cycle must be executed before the execute cycle
  - **Indirect Address:** The address specified by the instruction contains another address rather than an operand
- It includes following micro–operations:
  1. The **address field** of the instruction is transferred to the **MAR**
  2. Then the effective address of desired operand is fetched from the memory address specified by **MAR** to **MBR** through a **memory READ command**
  3. Finally, the **address field** of the IR is updated with the **current content** of the **MBR**
     - It signifies that IR now contains a **direct address** (effective address) rather than an indirect one. And now it is ready to perform the execute cycle

Indirect Cycle: Micro-operations in Time Sequence

t_1: MAR $\leftarrow$ (IR(Address))
t_2: MBR $\leftarrow$ Memory
t_3: IR(Address) $\leftarrow$ MBR

Interrupt Cycle

- At the **end** of any **execute cycle**, a **test** is made to determine if any **enabled interrupt** has been occurred or not
  - If so, an interrupt cycle will occur
- During an interrupt cycle, following micro-operations will be executed:
  1. At first step, the content of **PC** is transferred to **MBR**
     - So that it can be saved to return after completing interrupt handler.
  2. Then **MAR** will be loaded with **an address** where the content of **PC** will be saved (*in stack*)
  3. **Meanwhile,** **PC** will be loaded with another **address** of the start of the interrupt service routine (**ISR**)
     - As ***event 2*** and ***3*** do not interfere each other, they both can be grouped in one time unit
  4. At last, content of **MBR** should be stored in memory (*in stack*)
     - Control unit will issue a **memory WRITE command** on the control bus
- After this interrupt cycle, processor is ready to **fetch** next instruction cycle

Interrupt Cycle: Micro-operations in Time Sequence

- But the nature of interrupt cycle varies from one machine to another
  - It may take different number of steps (micro-operations) to complete

t_1: MBR $\leftarrow$ (PC)

t_2: MAR $\leftarrow$ Save_Address

PC $\leftarrow$ Routine_Address

t_3: Memory $\leftarrow$ (MBR)

Execute Cycle

- Fetch, indirect, interrupt cycles are **simple** and **predictable** in nature
  - Those involve a small and fixed sequence of micro operations
  - In each case, same micro operations will be **repeated**
- But for **execute cycle**, the scenario is different:
  - Because of **variety of opcodes** to perform different types of operations
  - For each opcodes, there are a number of **different sequences** of micro operations
- Control unit **examines** the **opcode**
  - It **generates** the **sequence of micro operations** based on the content of the opcode field of any instruction
  - This step is known as ***instruction decoding***

Execute Cycle: Add Instruction

- To add the content of **memory location X** to the content of **register R1** and store the result in **R1** back, we need the following instruction:

```
ADD R1, X
```

- Then following sequence of micro operations will be occurred:

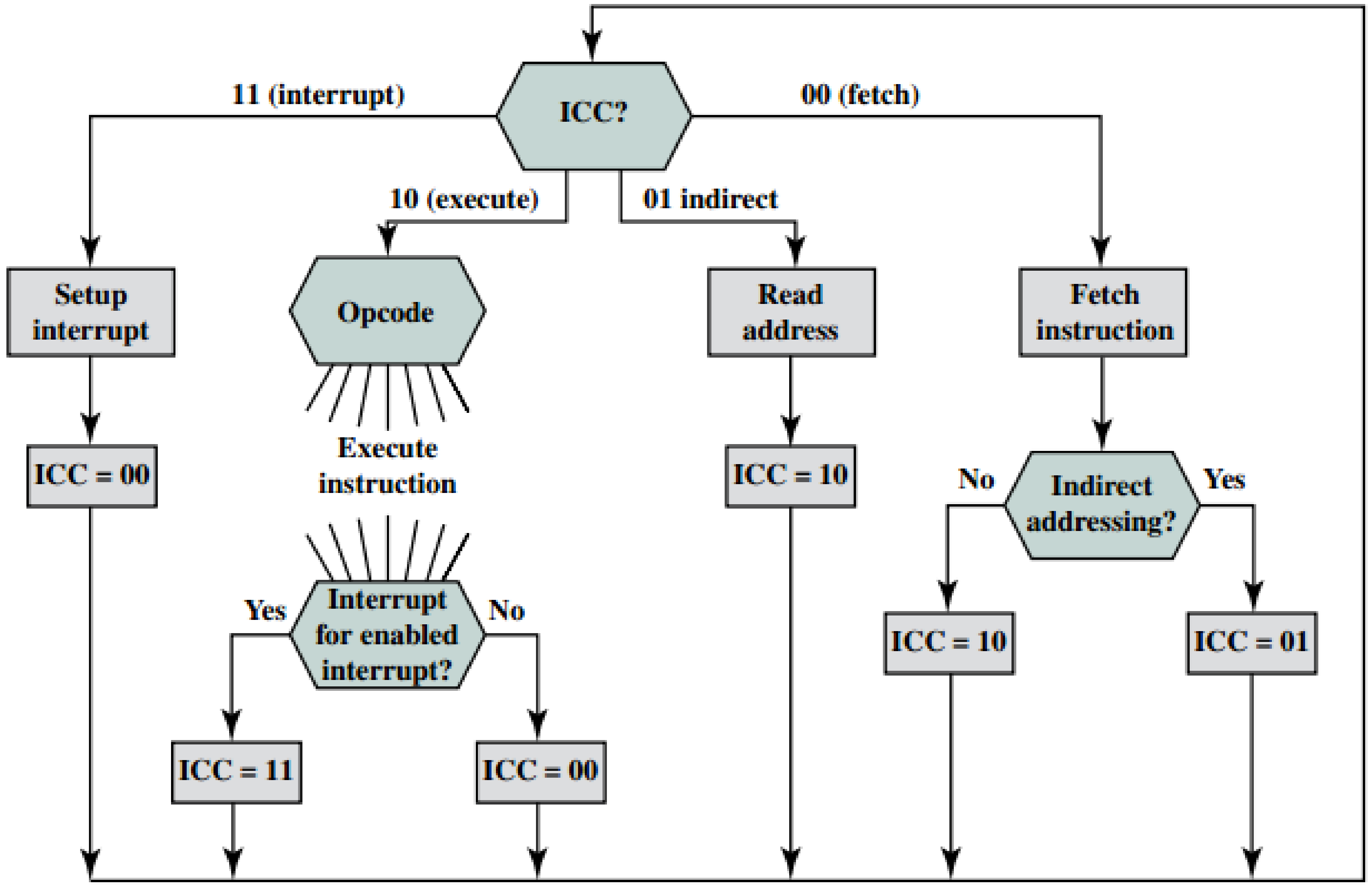
$$\begin{aligned}
t_1&: \text{MAR} \leftarrow (\text{IR(address)})\\
t_2&: \text{MBR} \leftarrow \text{Memory}\\
t_3&: \text{R1} \leftarrow (\text{R1}) + (\text{MBR})
\end{aligned}$$

Instruction Cycle's State

- Until now, we have discussed, **each phase** of the instruction cycle can be **decomposed** into a sequence of elementary micro operations
  - To complete the picture, we need to merge different sequences of micro operations together
- Assume a **2 bit register** – **I**nstruction **C**ycle **C**ode (***ICC***)
  - It designates the state of the processor in term of different **phases of an instruction cycle** like:

    00 – Fetch
    01 – Indirect
    10 – Execute
    11 – Interrupt
  - After finishing each of the cycle, ***ICC*** is **updated** appropriately by the processor

Flowchart of an Instruction Cycle



Functional Requirements for Control Unit...

- All micro operations fall into one of the following categories:
  1. Transfer data from one **register** to another **register**
  2. Transfer data from a **register** to an **external interface** (connected with the system bus)
  3. Transfer data from an **external interface** to a **register**
  4. Perform an **arithmetic** or **logic operation** using registers

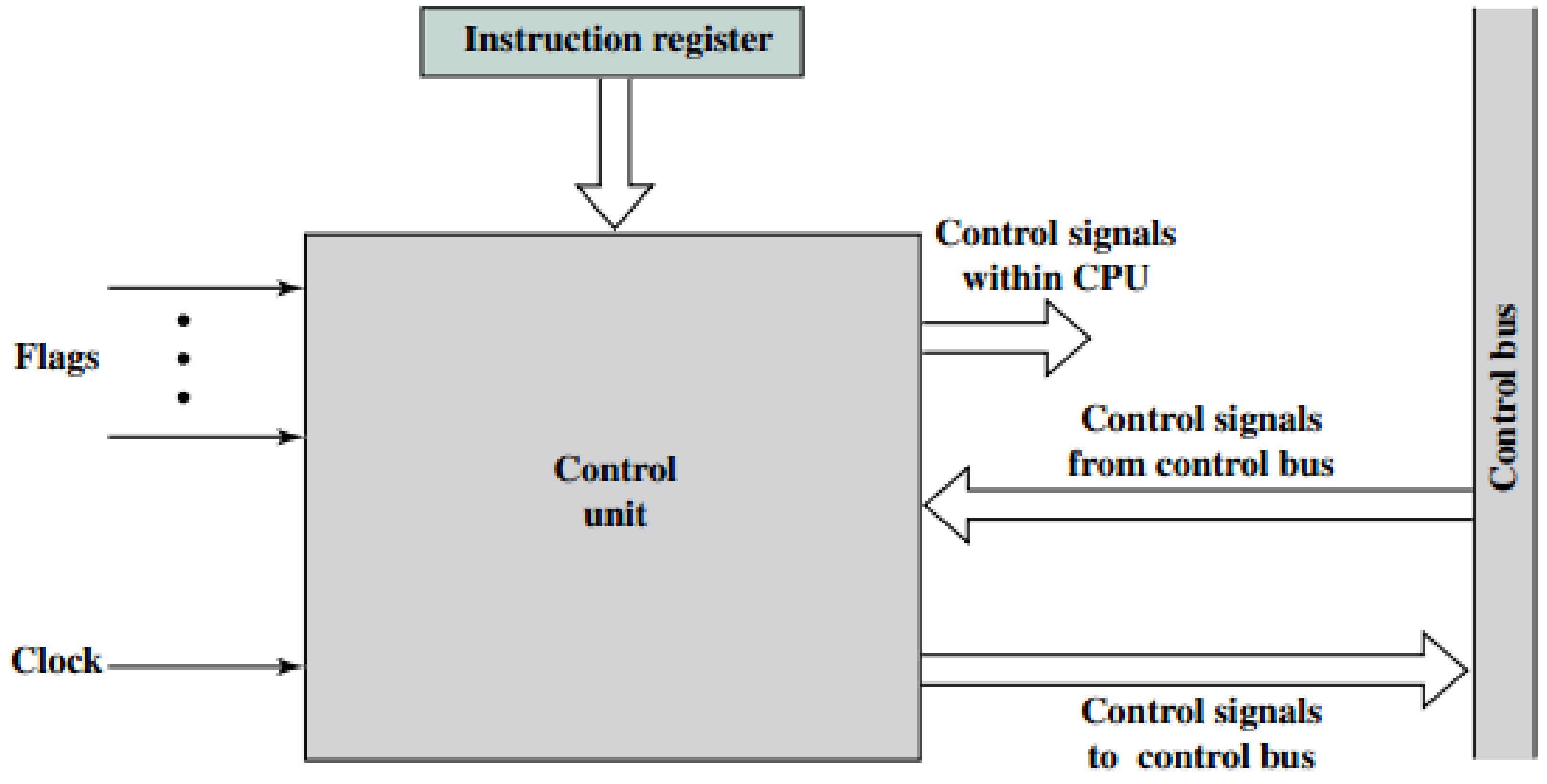
Functional Requirements for Control Unit...

- Basically, **control unit** performs **two basic tasks** through **control signals**:
  1. **Sequencing**
     - Thus the processor can execute a series micro operations in the **proper sequence** based on the program being executed
  2. **Execution (request)**
     - Thus the control unit can **cause** each micro operation to be executed using the **triggering functions**

Control Signals

- We have already learned:
    - The elements that make up the processor – ALU, CU, registers, data paths
    - The micro operations that are **executed** by the processor
- But ***how*** will the processor execute them?
    - With the help of **control unit**
  - Now, ***how*** will the control unit perform its job?
    - It will take **inputs** that **determine** the **current state** of the system
    - It will also provide **output signal** that control the behavior of the system
- Control unit has its **logic** to perform its **sequencing** and **execution** functions
  - Based on that it will interact with other elements of the processor to determine a **proper decision**

Block Diagram of Control Unit


Instruction register
Control signals within CPU
Flags
Control signals from control bus
Control unit
Control bus
Clock
Control signals to control bus

Control Unit: Inputs

- **Clock**
    - This is how control unit keeps time for **synchronization**
- **Instruction Register**
    - From this register, the **opcode** and **addressing mode** of the current instruction is determined to select proper micro operation(s) to be performed during execute cycle
- **Flags**
    - Needed for the control unit itself to know about the status of the processor and the outcome of previous ALU operations
    - **Example:** ***ISZ X***, here based on the updated/incremented value of **X**, current PC will be skipped if value of X is zero which has been actually reflected by the zero flag
- **Control Signals from Control Bus**
    - Control bus from the system bus receives **necessary signal** from outside to the control unit

Control Unit: Outputs

- Control signals **within** the processor
  - There are **two** types of signals to the processor:
    1. Signals that cause data to be **moved** from one register to another
    2. Signals that activate specific **ALU functions**
- Control signals **to** control bus
  - There are also **two** types of signals to the control bus:
    1. Control signal to the memory to **move data**
    2. Control signal to I/O modules to perform specific **I/O operation**

Types of Control Output Signals

Issued by a CU (echoed from the last slide)

1. Signals that activate an <u>ALU function</u>
   - **Example:** ***Add*** through a specific logic
2. Signals that activate an <u>internal data path</u>
   - **Example:** ***move*** data from ***MBR*** to ***IR*** during instruction fetch
3. Signals that are provided to <u>external system bus</u> or other <u>external interface</u>
   - **Example:** memory ***READ*** or ***WRITE***

These signals generated by the CU are ultimately applied as binary inputs to individual logic gates of other components of the computer

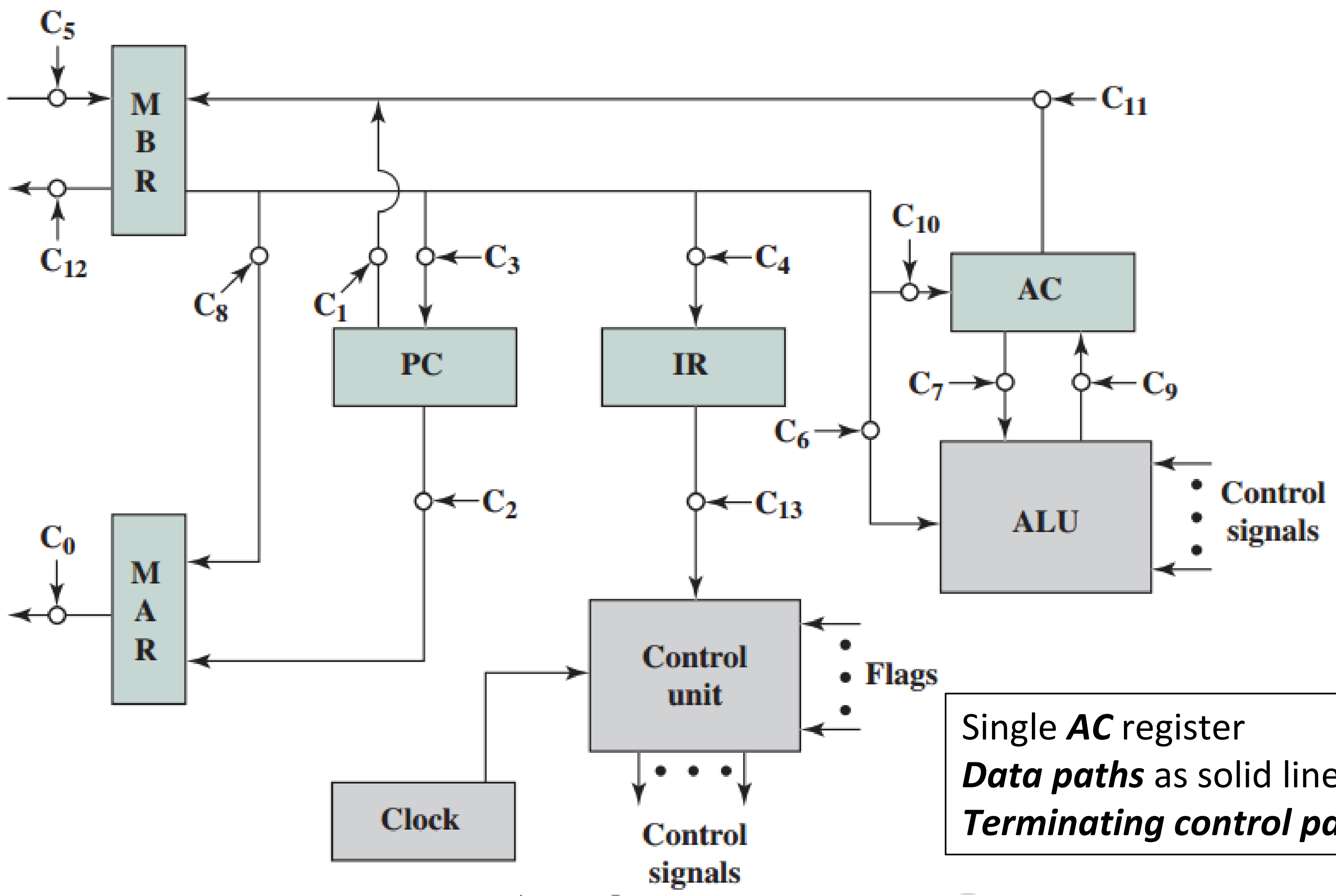
Example: Control Signals during Fetch Cycle

Issued by a CU to maintain the fetch cycle

- The content of **PC** needs to be transferred to **MAR**
  - Control unit opens the gates between **PC** and **MAR** (type 2)
- Next, read a word from memory (type 3) and update PC (type 1) in parallel
  - Control unit signals to open the gates between **MAR** and the address bus of the system bus
  - A **memory read** control signal to the control bus
  - A control signal to open the gates between **MBR** and the **data bus** of the system bus to allow the content of data bus storing to MBR
  - Control signal to the **ALU logic** that adds 1 to the content of **PC** and stores it back to the **PC**
- After that, control unit sends a control signal to open the gates between the **MBR** and **IR**

[N.B: what would be the next sub cycle e.g. indirect or execute that also should be decided in this cycle]

Control Unit with Control Signals & Data Paths



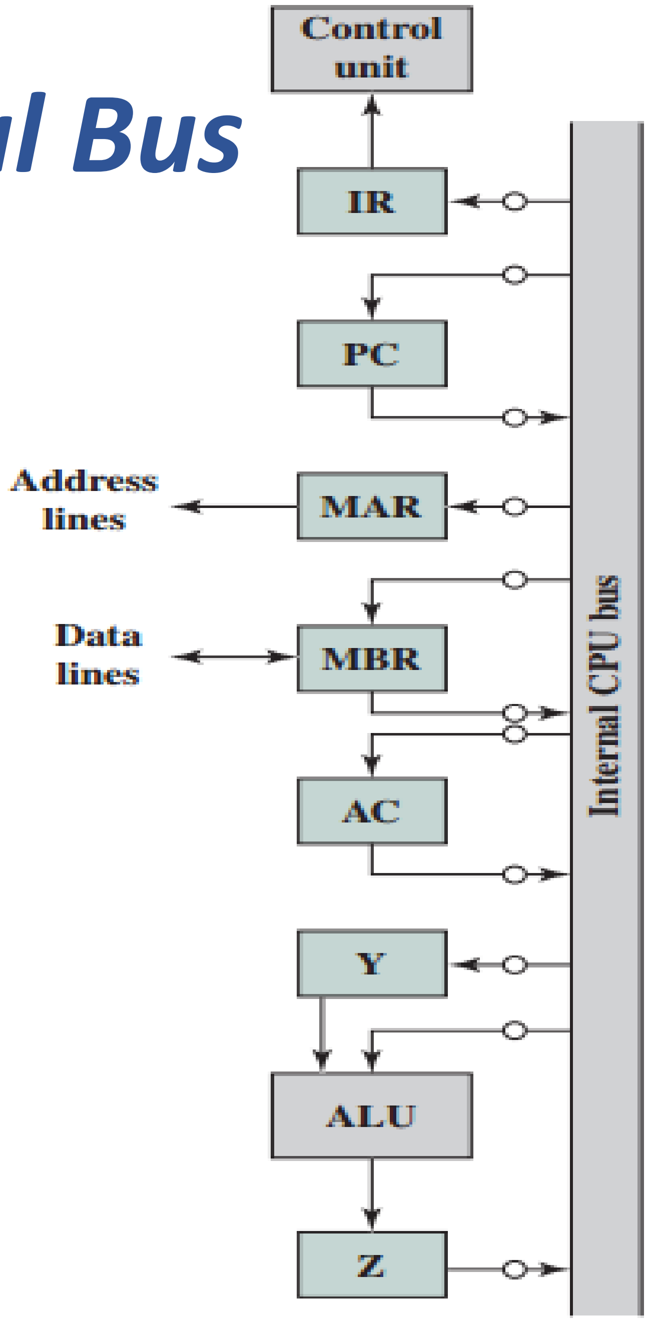
Single ***AC*** register
Data paths as solid lines
Terminating control paths as ***circle*** and ***C_i***

Internal Processor Organization

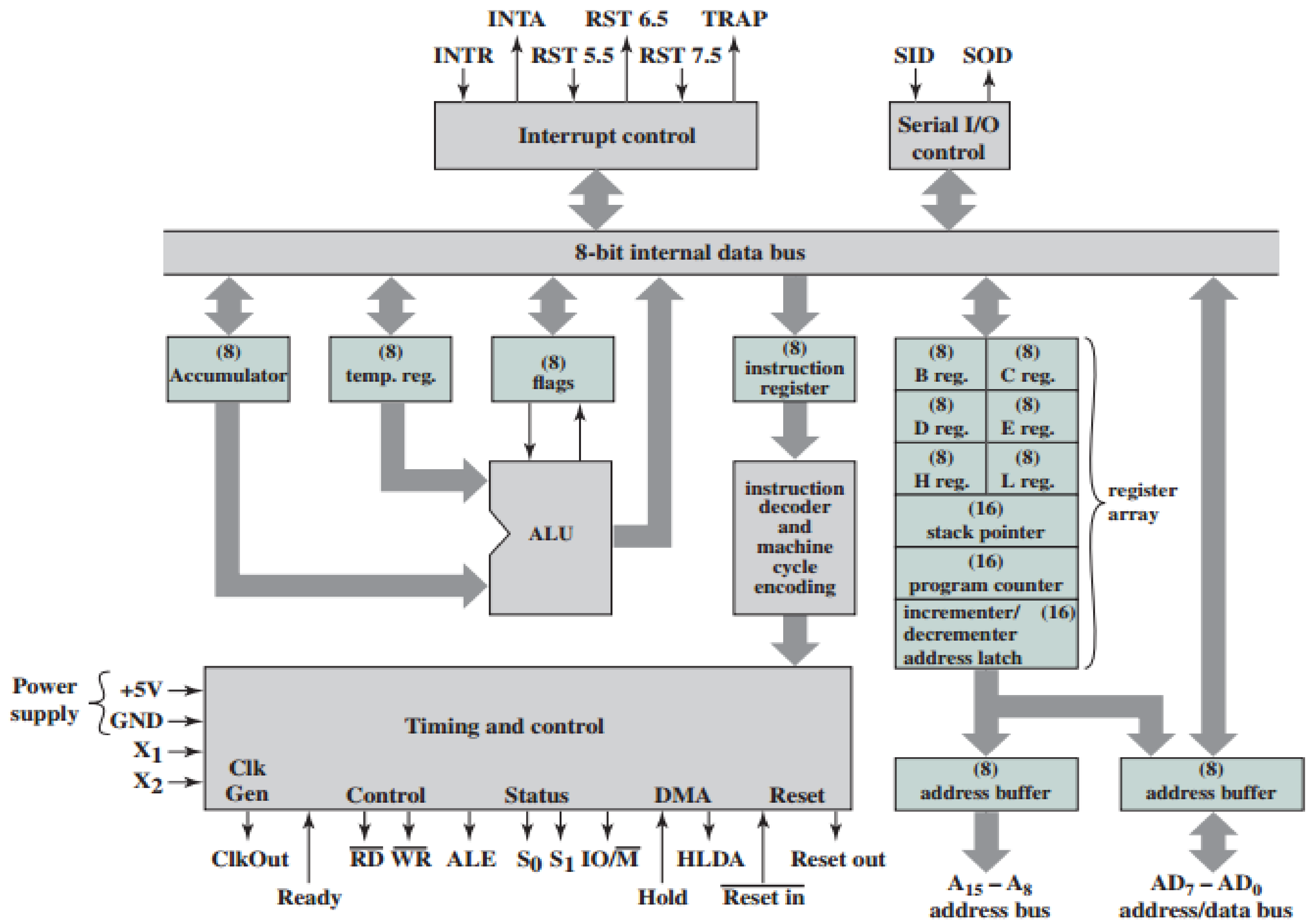
- If variety of data paths (p2p) are used, it will increase the complexity of the organization
  - Rather an **internal processor bus** is used which connects the ALU and all processor register
  - Gates and control signals are provided for data movement onto and off the bus from each register
  - Additionally two registers (labeled Y and Z) are added for proper operation of ALU
    - Temporary storage as second operand and output during arithmetic and logical operations respectively (e.g. adding a value from memory to the AC)
  - It is required for simplicity of interconnection layout and saving space

t_1: MAR $\leftarrow$ (IR(address))
t_2: MBR $\leftarrow$ Memory
t_3: Y $\leftarrow$ (MBR)
t_4: Z $\leftarrow$ (AC) + (Y)
t_5: AC $\leftarrow$ (Z)

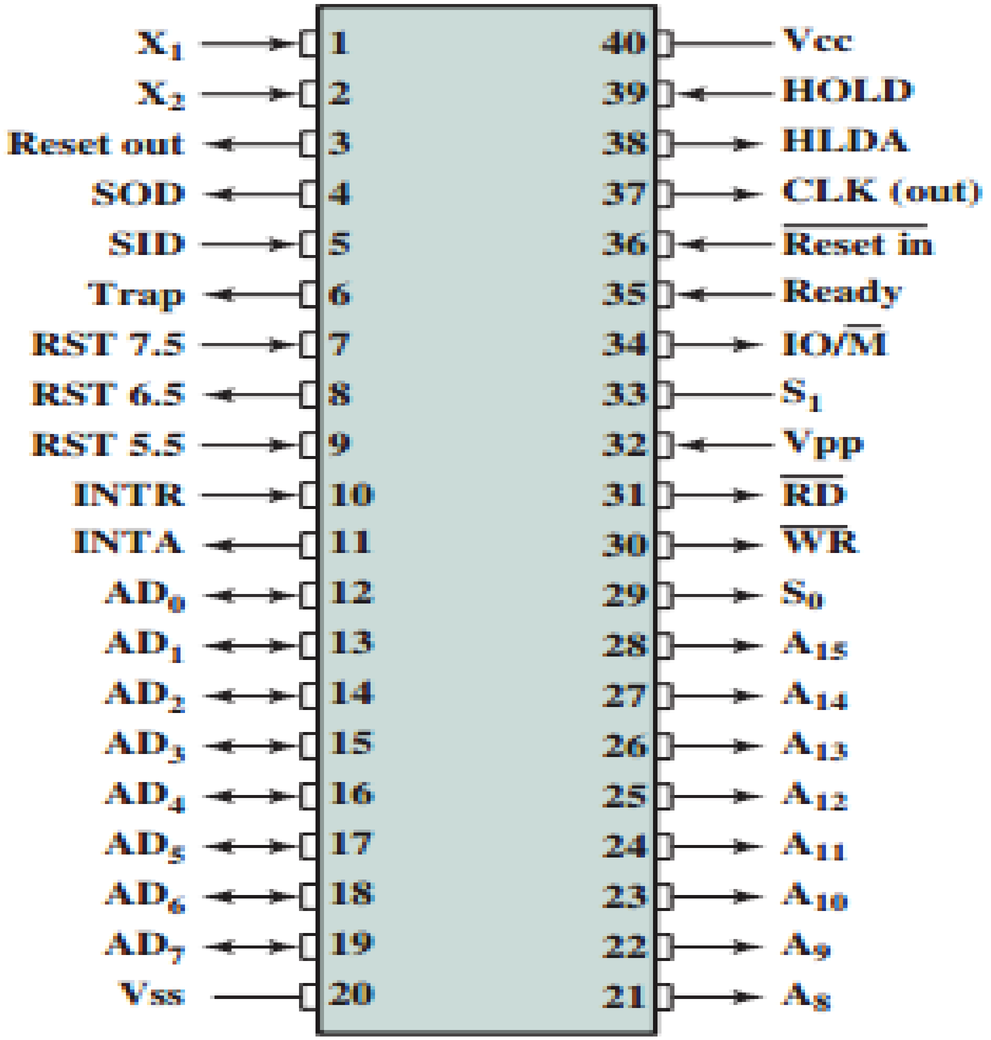
CPU with Internal Bus



Example: Intel 8085 CPU Block Diagram



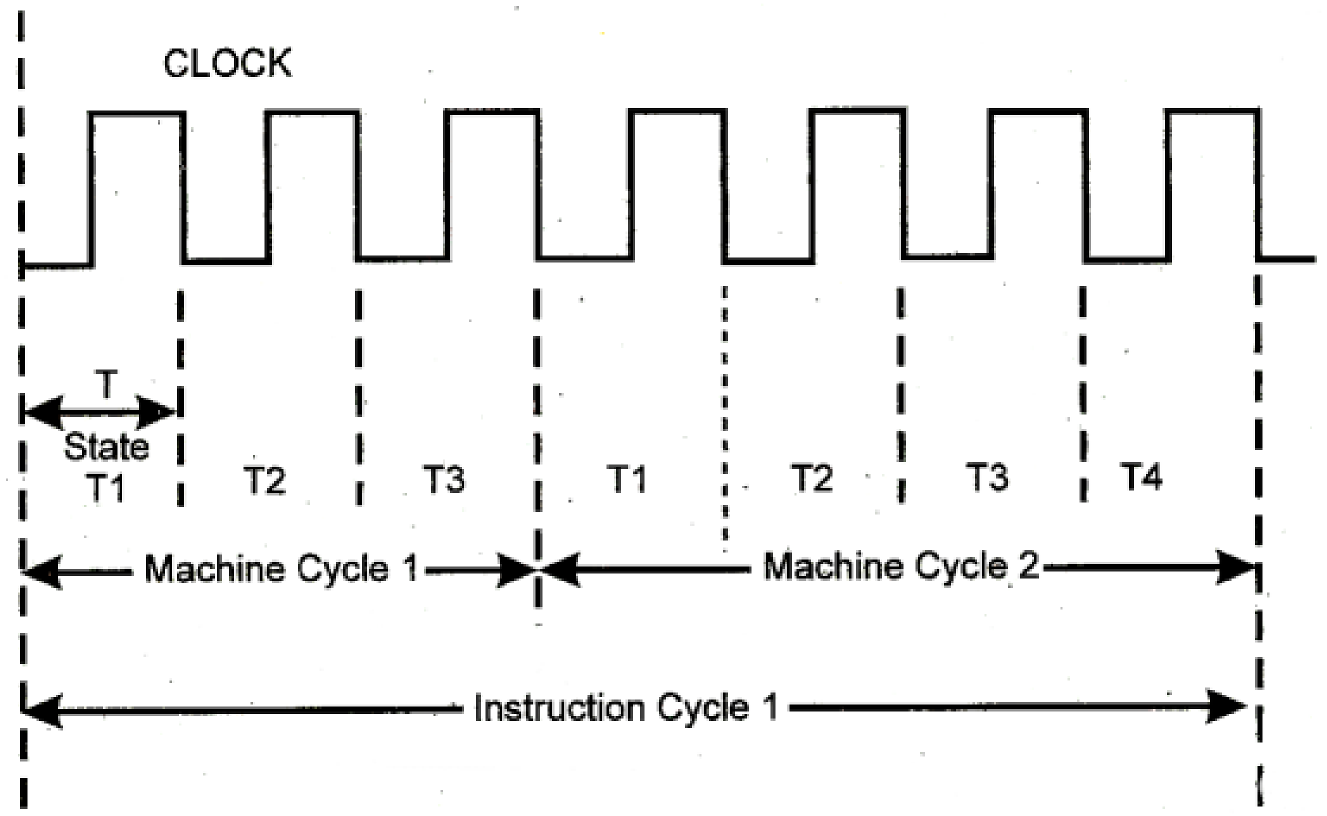
Intel 8085 Pin Configuration
[Interface between Processor and External System]



Different Level of Cycles ***[During Instruction Execution]***

- Operations of the processor –
    - Controlled by the control unit through control signals
    - **Also their timing is synchronized by the clock**
- Each **instruction cycle** is divided into from one to five **machine cycles**
    - Each **machine cycles** in turn divided into from three to five **states (also known as t-state or clock cycles)**
        - Each **state** lasts one **clock cycle**
            - During a **state**, the processor performs <u>one or a set of simultaneous micro operations</u> as determined by the control signals
- The number of machine cycles is fixed for a given instruction but varies from one instruction to another
- **Machine cycles** are defined to be equivalent to **bus access**
    - Thus the number of machine cycles for an instruction depends on the number of times the processor must **communicate** with **external devices**

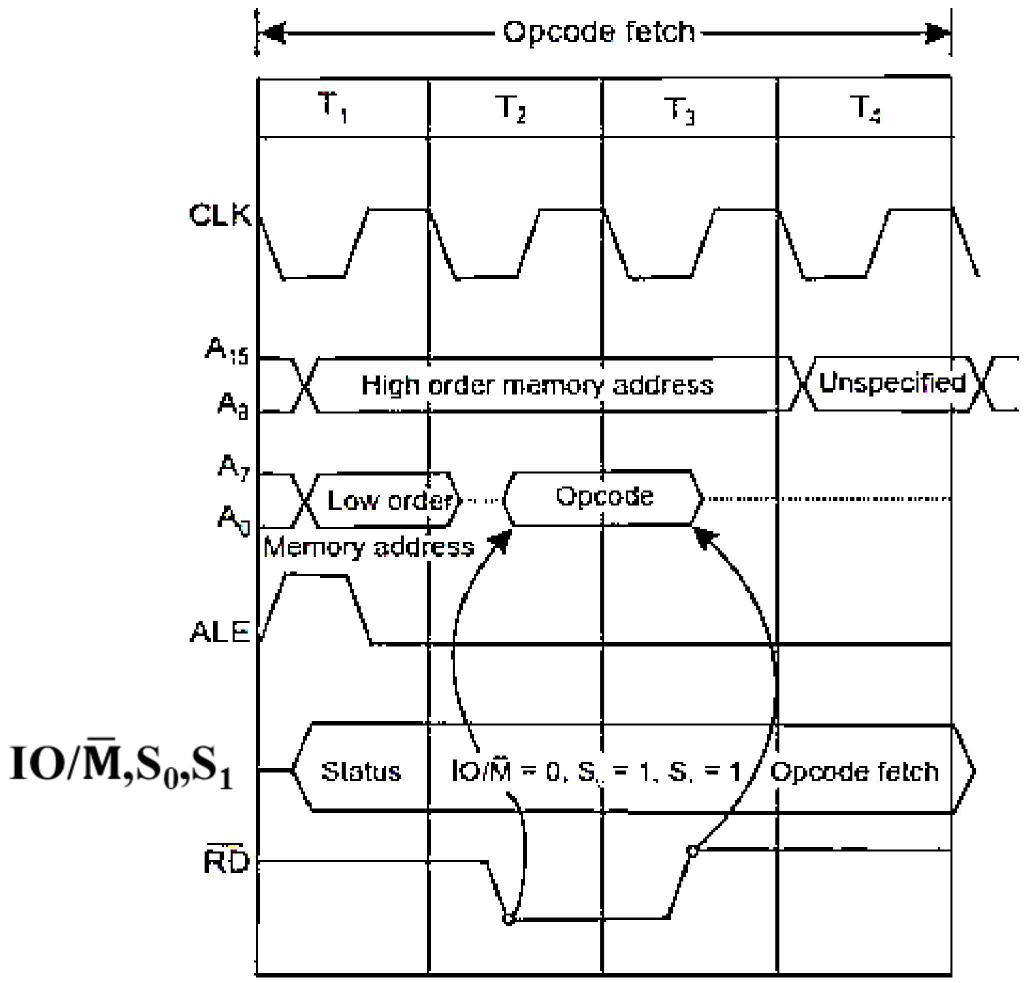
Timing Diagram of an Instruction Execution



Machine Cycles

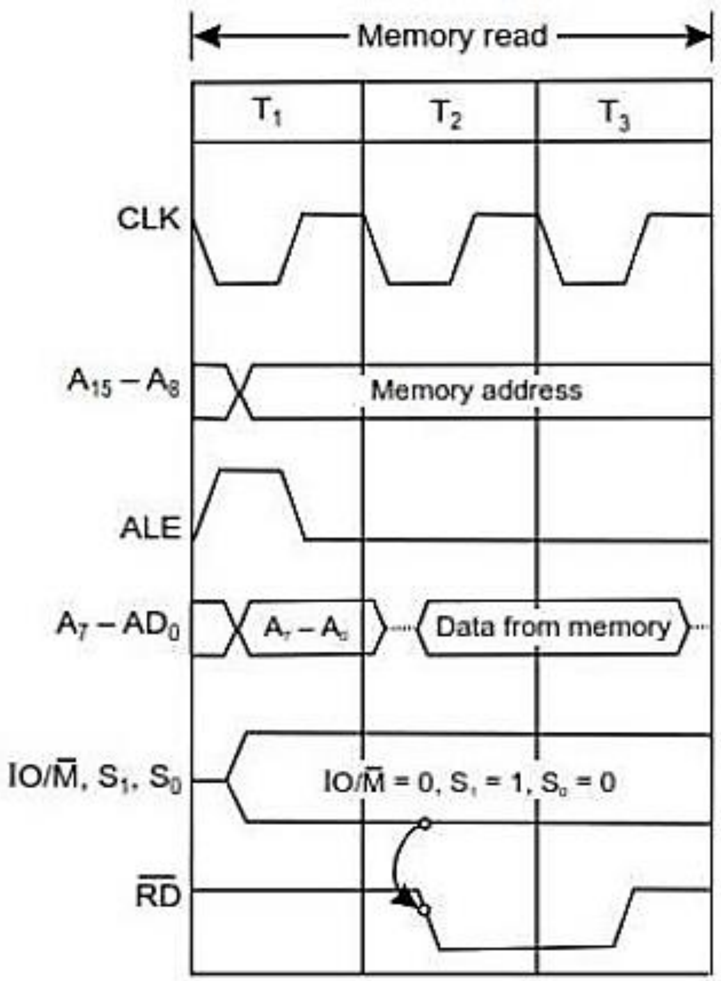
- Some basic **machine cycles** are for Intel 8085:
    - Opcode Fetch Cycle (4T)
    - Memory Read Cycle (3T)
    - Memory Write Cycle (3T)
    - I/O Read Cycle (3T)
    - I/O Write Cycle (3T)

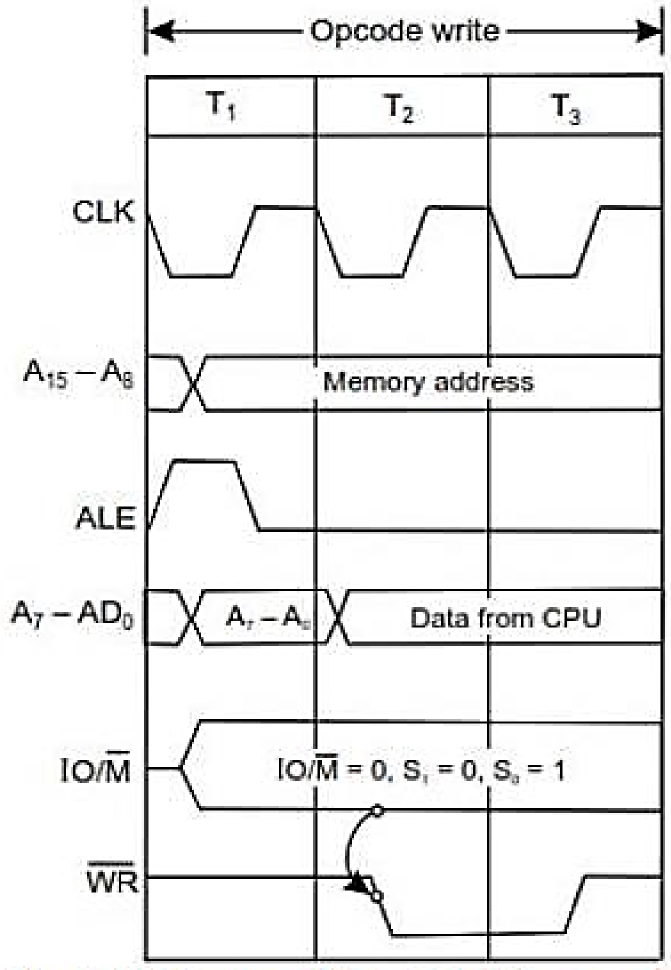
Opcode Fetch Machine Cycle of Intel 8085



- Each of the machine cycle starts with ALE pulse signals from the control unit
    - It alerts the external circuits
- During the opcode fetch cycle, last state remains unspecified for decoding the instruction
- During the first state the address of the instruction (PC) should be spread over A_8-A_{15} (PC $_{\text{Higher Order Byte}}$) and A_0/D_0-A_7/D_7 (PC $_{\text{Lower Order Byte}}$)
    - For later states, A_0/D_0-A_7/D_7 will be freed for data (functions as data bus)
- As it is a read operation from memory,
    - $\overline{RD} = 0$
    - $IO/\overline{M} = 0$
    - $S_0=1, S_1=1$

Memory Read and Write Machine Cycle

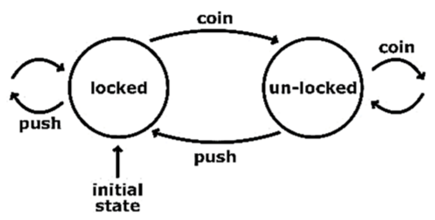




Control Unit Implementation

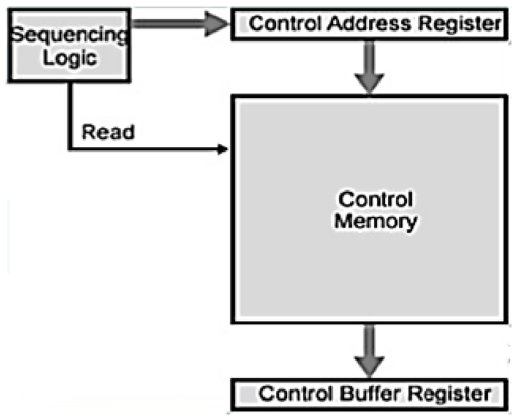
- There are two techniques to implement the control unit:

  1. **Hardwired implementation**



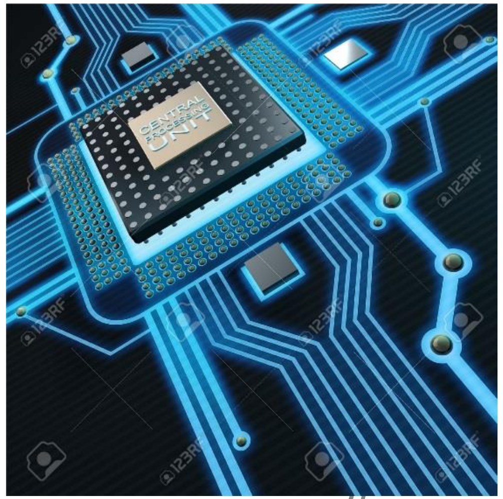
     - Designed based a **state machine**. Depending on the current state of the control unit and input control (logic) signals, control unit will switch to another state and produce a set of control signals
     - Generally, when the control signals are generated by hardware using conventional logic design techniques, the control unit is called to **be hardwired**.

  2. **Microprogrammed implementation**

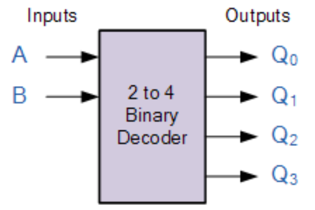


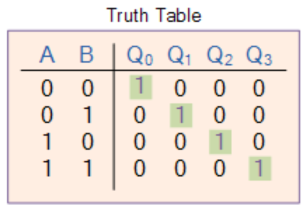
     - All controls that can be activated simultaneously are grouped together to form the **control words** which are inaccessible by the programmer
     - These control words are specially stored in **control memory**

Hardwired Implementation



Control Unit Inputs (x_i)

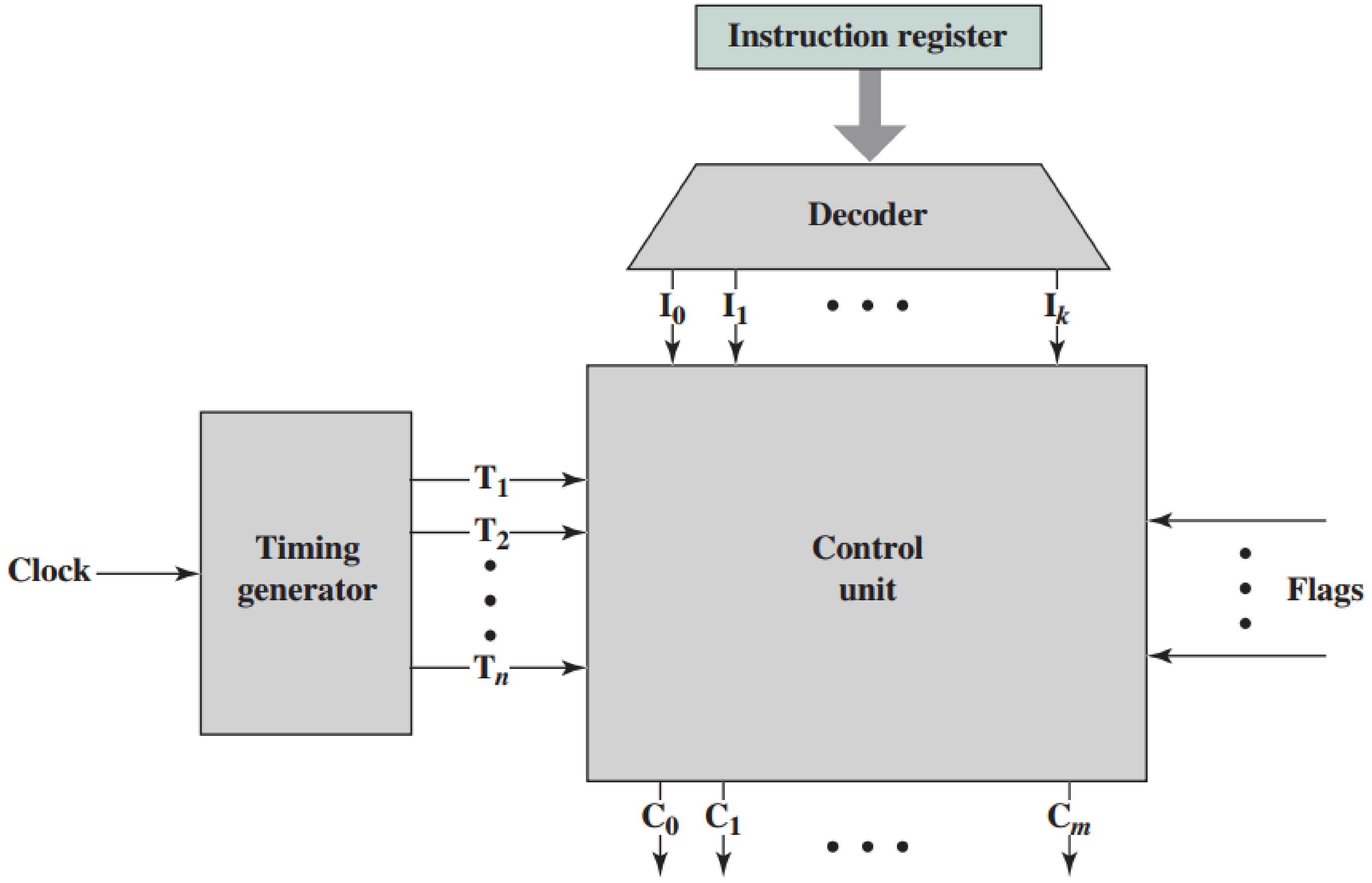



Truth Table

A	B	Q_0	Q_1	Q_2	Q_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

- **Key inputs are:**
  - **IR**
    - Control unit makes use of the opcode and perform different actions for instructions
    - Each opcode has a unique logic input, can be interpreted as ***decoder*** which takes an encoded input and produces a single output (n inputs; 2^n outputs)
  - **Clock**
    - It issues a repetitive sequences of pulses which is required to measure the duration of micro operations
  - **Flags**
    - Each individual bit has some special meaning reflected after performing each ALU operation
  - **Control bus signals (External)**
    - Each individual bit has its own essence to the control unit issued from different external components

Control Unit with Decoded Inputs



Control Unit Logic (Logic Circuit Implementation)

- Control unit can be implemented as a logic circuit that produces output control signals as a function of its input control signals
    - For each output control signal, it is required to derive a Boolean expression for that signal as a function of inputs
    - Example: Control signal, C_5 = Read data from external data bus into MBR (Memory Read) [See Slide]
        - In different phases/cycles, this control signal can be applied. So we can define them as:

$PQ = 00$	Fetch Cycle
$PQ = 01$	Indirect Cycle
$PQ = 10$	Execute Cycle
$PQ = 11$	Interrupt Cycle

        - Now, C_5 can be expr $C_5 = \overline{P} \bullet \overline{Q} \bullet T_2 + \overline{P} \bullet Q \bullet T_2$

And you could make it rather complex…

Cont….