UNIT-4

PROCESSOR CONTROL UNIT

❖ MICRO-OPERATIONS

The control unit controls the operation of the processor.

The operation of a computer, in executing a program, consists of a sequence of instruction cycles, with one machine instruction per cycle. Each instruction cycle is made up of a number of smaller units. One subdivision that we found convenient is fetch, indirect, execute, and interrupt, with only fetch and execute cycles always occurring.

We began to see that a further decomposition is possible. Each of the smaller cycles involves a series of steps, each of which involves the processor registers. We will refer to these steps as micro-operations. The prefix micro refers to the fact that each step is very simple and accomplishes very little. Figure 4.1 depicts the relationship. To summarize, the execution of a program consists of the sequential execution of instructions. Each instruction is executed during an instruction cycle made up of shorter subcycles (e.g., fetch, indirect, execute, interrupt).The execution of each sub cycle involves one or more shorter operations, that is, micro-operations.

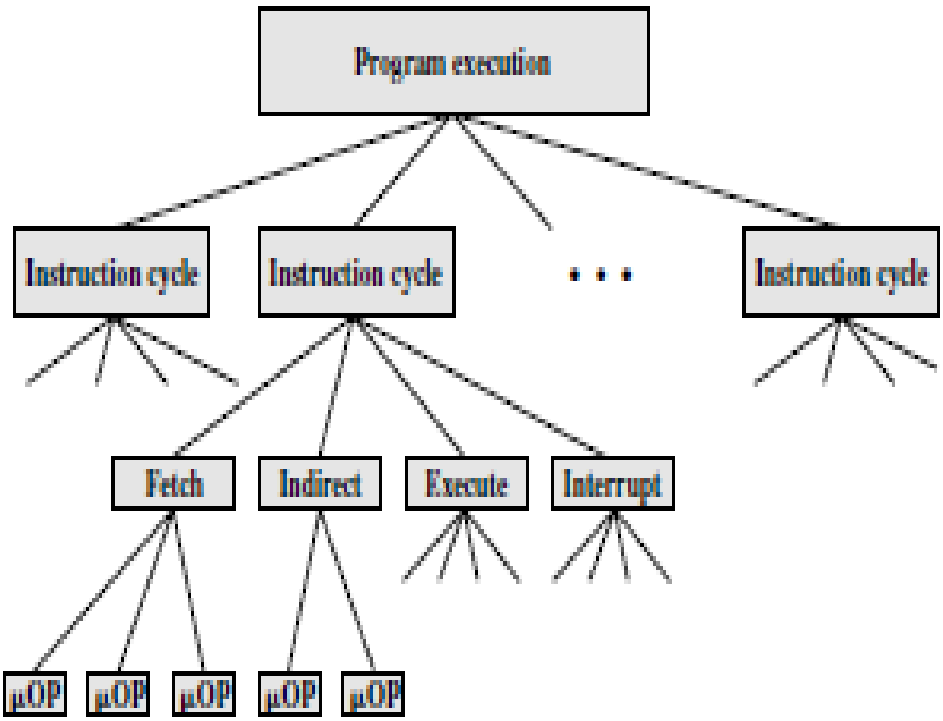


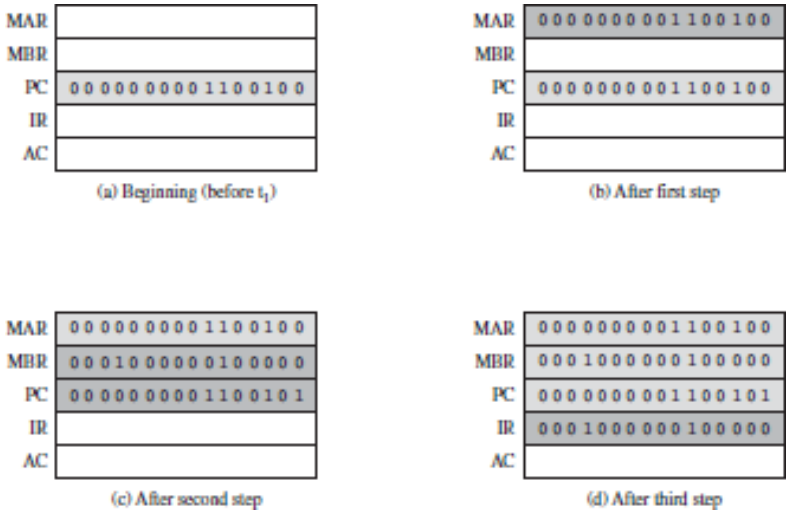
Figure 4,1 Constituent elements of a Program execution

The Fetch Cycle

We begin by looking at the fetch cycle, which occurs at the beginning of each instruction cycle and causes an instruction to be fetched from memory. Four registers are involved:

- **Memory address register (MAR):** Is connected to the address lines of the system bus. It specifies the address in memory for a read or write operation.
- **Memory buffer register (MBR):** Is connected to the data lines of the system bus. It contains the value to be stored in memory or the last value read from memory.
- **Program counter (PC):** Holds the address of the next instruction to be fetched.
- **Instruction register (IR):** Holds the last instruction fetched.

Let us look at the sequence of events for the fetch cycle from the point of view of its effect on the processor registers. An example appears in Figure 4.2.



4.2 Sequence of events, Fetch cycle

At the beginning of the fetch cycle, the address of the next instruction to be executed is in the program counter (PC); in this case, the address is 1100100.

- The first step is to move that address to the memory address register (MAR) because this is the only register connected to the address lines of the system bus.
- The second step is to bring in the instruction. The desired address (in the MAR) is placed on the address bus, the control unit issues a READ command on the control bus, and the result appears on the data bus and is copied into the memory buffer register (MBR). We also need to increment the PC by the instruction length to get ready for the next instruction. Because these two actions (read word from memory, increment PC) do not interfere with each other, we can do them simultaneously to save time.
- The third step is to move the contents of the MBR to the instruction register (IR). This frees up the MBR for use during a possible indirect cycle.

Thus, the simple fetch cycle actually consists of three steps and four micro-operations. Symbolically, we can write this sequence of events as follows:

t1: MAR ← (PC)
t2: MBR ← Memory
PC ← (PC) + I
t3: IR ← (MBR)

where I is the instruction length. Each micro-operation can be performed within the time of a single time unit.The notation (t1, t2, t3)represents successive time units.In words,we have

- **First time unit:** Move contents of PC to MAR.
- **Second time unit:** Move contents of memory location specified by MAR to MBR. Increment by I the contents of the PC.
- **Third time unit:** Move contents of MBR to IR.

Note that the second and third micro-operations both take place during the second time unit. The third micro-operation could have been grouped with the fourth without affecting the fetch operation:

t1: MAR ← (PC)
t2: MBR ← Memory
t3: PC ← (PC) + I
IR ← (MBR)

The groupings of micro-operations must follow three simple rules:

1 The proper sequence of events must be followed. Thus (MAR ; (PC)) must precede (MBR ; Memory) because the memory read operation makes use of the address in the MAR.

2 Conflicts must be avoided. One should not attempt to read to and write from the same register in one time unit, because the results would be unpredictable. For example, the micro-operations (MBR ; Memory) and (IR ; MBR) should not occur during the same time unit.

3 A final point worth noting is that one of the micro-operations involves an addition. To avoid duplication of circuitry, this addition could be performed by the ALU.

The Indirect Cycle

Once an instruction is fetched, the next step is to fetch source operands. If the instruction specifies an indirect address, then an indirect cycle must precede the execute cycle and includes the following micro-operations:

t1: MAR ← (IR(Address))

t2: MBR ← Memory

t3: IR(Address) ← (MBR(Address))

The address field of the instruction is transferred to the MAR.This is then used to fetch the address of the operand. Finally, the address field of the IR is updated from the MBR, so that it now contains a direct rather than an indirect address. The IR is now in the same state as if indirect addressing had not been used, and it is ready for the execute cycle.

The Interrupt Cycle

At the completion of the execute cycle, a test is made to determine whether any enabled interrupts have occurred. If so, the interrupt cycle occurs. The nature of this cycle varies greatly from one machine to another. We present a very simple sequence of events,

t1: MBR ← (PC)
t2: MAR ← Save_Address
PC ← Routine_Address
t3: Memory ← (MBR)

In the first step, the contents of the PC are transferred to the MBR, so that they can be saved for return from the interrupt. Then the MAR is loaded with the address at which the contents of the PC are to be saved, and the PC is loaded with the address of the start of the interrupt-processing routine. These two actions may each be a single micro-operation. to the MAR and PC, respectively. The final step is to store the MBR, which contains the old value of the PC, into memory. The processor is now ready to begin the next instruction cycle.

The Execute Cycle

The fetch, indirect, and interrupt cycles are simple and predictable. Each involves a small, fixed sequence of micro-operations and, in each case the same micro-operations are repeated each time around.

This is not true of the execute cycle. Because of the variety opcodes, there are a number of different sequences of micro-operations that can occur.

Let us consider several hypothetical examples.

First, consider an add instruction: ADD R1, X

which adds the contents of the location X to register R1. The following sequence of micro-operations might occur:

t1: MAR ← (IR(address))

t2: MBR ← Memory

t3: R1 ← (R1) + (MBR)

We begin with the IR containing the ADD instruction. In the first step, the address portion of the IR is loaded into the MAR. Then the referenced memory location is read. Finally, the contents of R1 and MBR are added by the ALU.

Let us look at two more complex examples.

A common instruction is increment and skip if zero:

ISZ X

The content of location X is incremented by 1. If the result is 0, the next instruction is skipped. A possible sequence of micro-operations is

t1: MAR ← (IR(address))
t2: MBR ← Memory
t3: MBR ← (MBR) + 1
t4: Memory ← (MBR)
If ((MBR) = 0) then (PC ← (PC) + I)

The new feature introduced here is the conditional action.The PC is incremented if (MBR) = 0. This test and action can be implemented as one micro-operation.

Finally,consider a subroutine call instruction.As an example,consider a branchand-save-address instruction:

BSA X

The address of the instruction that follows the BSA instruction is saved in location X, and execution continues at location X + I. The saved address will later be used for return.The following micro-operations suffice:

t1: MAR ← (IR(address))
MBR ← (PC)
t2: PC ← (IR(address))
Memory ← (MBR)
t3: PC ← (PC) + I

The address in the PC at the start of the instruction is the address of the next instruction in sequence. This is saved at the address designated in the IR. The latter address is also incremented to provide the address of the instruction for the next instruction cycle.

The Instruction Cycle

We have seen that each phase of the instruction cycle can be decomposed into a sequence of elementary micro-operations. There is one sequence each for the fetch, indirect, and interrupt cycles, and, for the execute cycle, there is one sequence of micro-operations for each opcode.
We assume a new 2-bit register called the *instruction cycle code* (ICC). The ICC designates the state of the processor in terms of which portion of the cycle it is in:

00: Fetch

01: Indirect

10: Execute

11: Interrupt

At the end of each of the four cycles, the ICC is set appropriately. The indirect cycle is always followed by the execute cycle. The interrupt cycle is always followed by the fetch cycle. For both the fetch and execute cycles, the next cycle depends on the state of the system.

Thus, the flowchart of Figure 4.3 defines the complete sequence of micro-operations, depending only on the instruction sequence and the interrupt pattern.

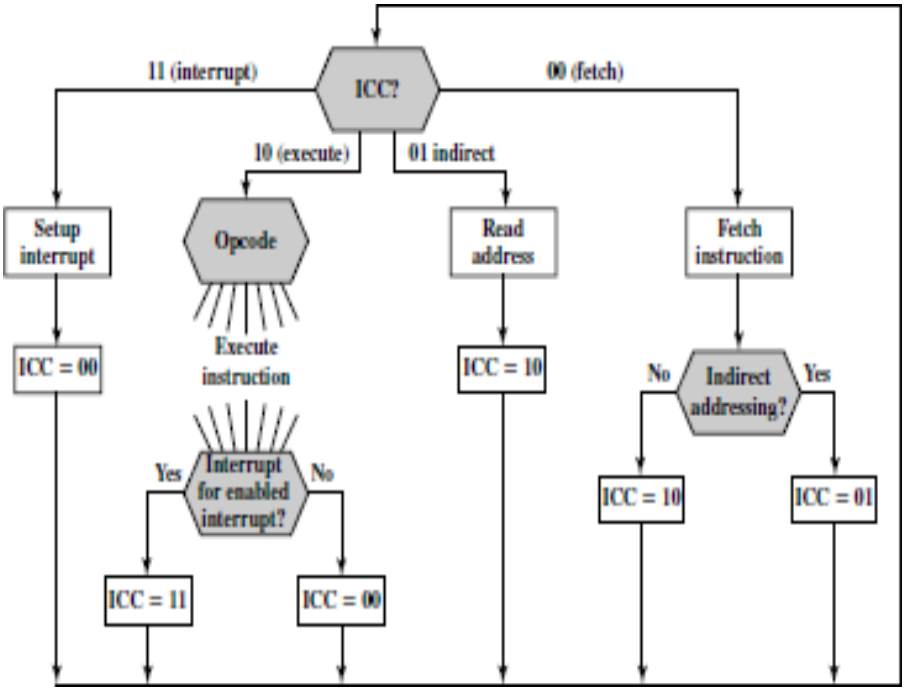


Figure 4.3 Flowchart for Instruction Cycle

❖ CONTROL OF THE PROCESSOR

By reducing the operation of the processor to its most fundamental level, we are able to define exactly what it is that the control unit must cause to happen. Thus, we can define the *functional requirements* for the control unit: those functions that the control unit must perform

With the information at hand, the following three-step process leads to a characterization of the control unit:

1 Define the basic elements of the processor.
2 Describe the micro-operations that the processor performs.
3 Determine the functions that the control unit must perform to cause the micro-operations to be performed.

First, the basic functional elements of the processor are the following:

1. ALU **2.** Registers **3.** Internal data paths **4.** External data paths **5.**Control Unit

The ALU is the functional essence of the computer. Registers are used to store data internal to the processor. Some registers contain status information needed to manage instruction sequencing (e.g., a program status word). Others contain data that go to or come from the ALU, memory, and I/O modules. Internal data paths are used to move data between registers and between register and ALU. External data paths link registers to memory and I/O modules, often by means of a system bus. The control unit causes operations to happen within the processor.

The execution of a program consists of operations involving these processor elements. All micro-operations fall into one of the following categories:

- Transfer data from one register to another.
- Transfer data from a register to an external interface (e.g., system bus).
- Transfer data from an external interface to a register.
- Perform an arithmetic or logic operation, using registers for input and output.

The control unit performs two basic tasks:

- **Sequencing:** The control unit causes the processor to step through a series of micro-operations in the proper sequence, based on the program being executed.
- **Execution:** The control unit causes each micro-operation to be performed.

The key to how the control unit operates is the use of control signals.

Control Signals

For the control unit to perform its function, it must have inputs that allow it to determine the state of the system and outputs that allow it to control the behaviour of the system. These are the external specifications of the control unit. Internally, the control unit must have the logic required to perform its sequencing and execution functions.

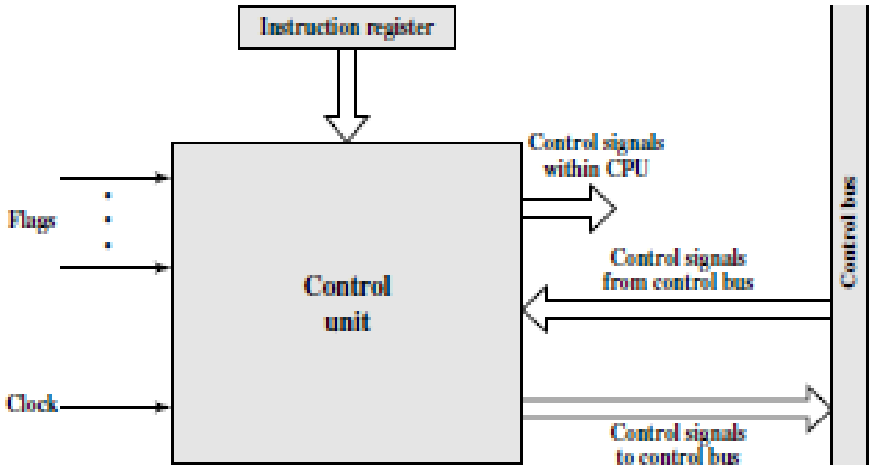


Figure 4.4 Block Diagram of Control Unit

Figure 4.4 is a general model of the control unit, showing all of its inputs and outputs. The inputs are

- **Clock:** This is how the control unit "keeps time." The control unit causes one micro-operation (or a set of simultaneous micro-operations) to be performed for each clock pulse. This is sometimes referred to as the processor cycle time, or the clock cycle time.
- **Instruction register:** The opcode and addressing mode of the current instruction are used to determine which micro-operations to perform during the execute cycle.
- **Flags:** These are needed by the control unit to determine the status of the processor and the outcome of previous ALU operations. For example, for the increment-and-skip-if-zero (ISZ) instruction, the control unit will increment the PC if the zero flag is set.
- **Control signals from control bus:** The control bus portion of the system bus provides signals to the control unit.

The outputs are as follows:

- **Control signals within the processor:** These are two types: those that cause data to be moved from one register to another, and those that activate specific ALU functions.
- **Control signals to control bus:** These are also of two types: control signals to memory, and control signals to the I/O modules.

Three types of control signals are used: those that activate an ALU function, those that activate a data path, and those that are signals on the external system bus or other external interface.

Let us consider again the fetch cycle to see how the control unit maintains control. The control unit keeps track of where it is in the instruction cycle. At a given point,it knows that the fetch cycle is to be performed next.The first step is to transfer the contents of the PC to the MAR. The control unit does this by activating the control signal that opens the gates between the bits of the PC and the bits of the MAR. The next step is to read a word from memory into the MBR and increment the PC. The control unit does this by sending the following control signals simultaneously:

- A control signal that opens gates, allowing the contents of the MAR onto the address bus
- A memory read control signal on the control bus
- A control signal that opens the gates, allowing the contents of the data bus to be stored in the MBR
- Control signals to logic that add 1 to the contents of the PC and store the result back to the PC

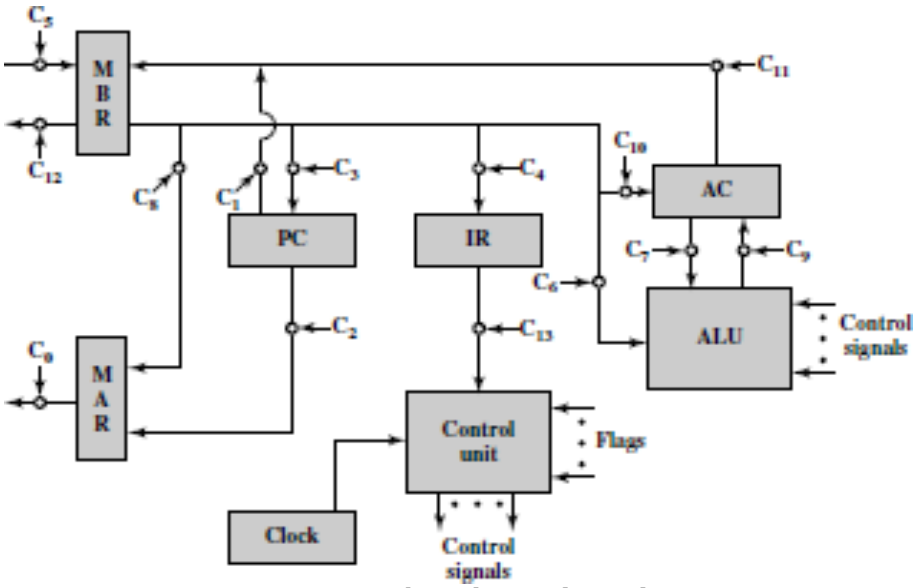
Following this, the control unit sends a control signal that opens gates between the MBR and the IR.

This completes the fetch cycle except for one thing: The control unit must decide whether to perform an indirect cycle or an execute cycle next. To decide this, it examines the IR to see if an indirect memory reference is made.

The indirect and interrupt cycles work similarly. For the execute cycle, the control unit begins by examining the opcode and, on the basis of that, decides which sequence of micro-operations to perform for the execute cycle.

A Control Signals Example

To illustrate the functioning of the control unit, let us examine Figure 4.5.This is a simple processor with a single accumulator (AC). The data paths between elements are indicated. The control paths for signals emanating from the control unit are not shown, but the terminations of control signals are labeled Ci and indicated by a circle.



4.5 Data paths and control signals

The control unit receives inputs from the clock, the instruction register, and flags. With each clock cycle, the control unit reads all of its inputs and emits a set of control signals. Control signals go to three separate destinations:

- **Data paths:** The control unit controls the internal flow of data. For example, on instruction fetch, the contents of the memory buffer register are transferred to the instruction register. For each path to be controlled, there is a switch (indicated by a circle in the figure). A control signal from the control unit temporarily opens the gate to let data pass.
- **ALU:** The control unit controls the operation of the ALU by a set of control signals. These signals activate various logic circuits and gates within the ALU.
- **System bus:** The control unit sends control signals out onto the control lines of the system bus (e.g., memory READ).

Table 4.1 indicates the control signals that are needed for some of the micro-operation sequences :

	Micro-operations	Active Control Signals
Fetch:	t_1: MAR ← (PC)	C_2
	t_2: MBR ← Memory PC ← (PC) + 1	C_5, C_R
	t_3: IR ← (MBR)	C_4
Indirect:	t_1: MAR ← (IR(Address))	C_8
	t_2: MBR ← Memory	C_5, C_R
	t_3: IR(Address) ← (MBR(Address))	C_4
Interrupt:	t_1: MBR ← (PC)	C_1
	t_2: MAR ← Save-address PC ← Routine-address	
	t_3: Memory ← (MBR)	C_{12}, C_W

C_R = Read control signal to system bus.
C_W = Write control signal to system bus.

Table 4.1 Micro-Operations and Control Signals

❖ 8085 ARCHITECTURE

The Intel 8085 organization is shown in Figure 4.6. Several key components that may not be self-explanatory are:

- **Incrementer/decrementer address latch:** Logic that can add 1 to or subtract 1 from the contents of the stack pointer or program counter. This saves time by avoiding the use of the ALU for this purpose.
- **Interrupt control:** This module handles multiple levels of interrupt signals.
- **Serial I/O control:** This module interfaces to devices that communicate 1 bit at a time.

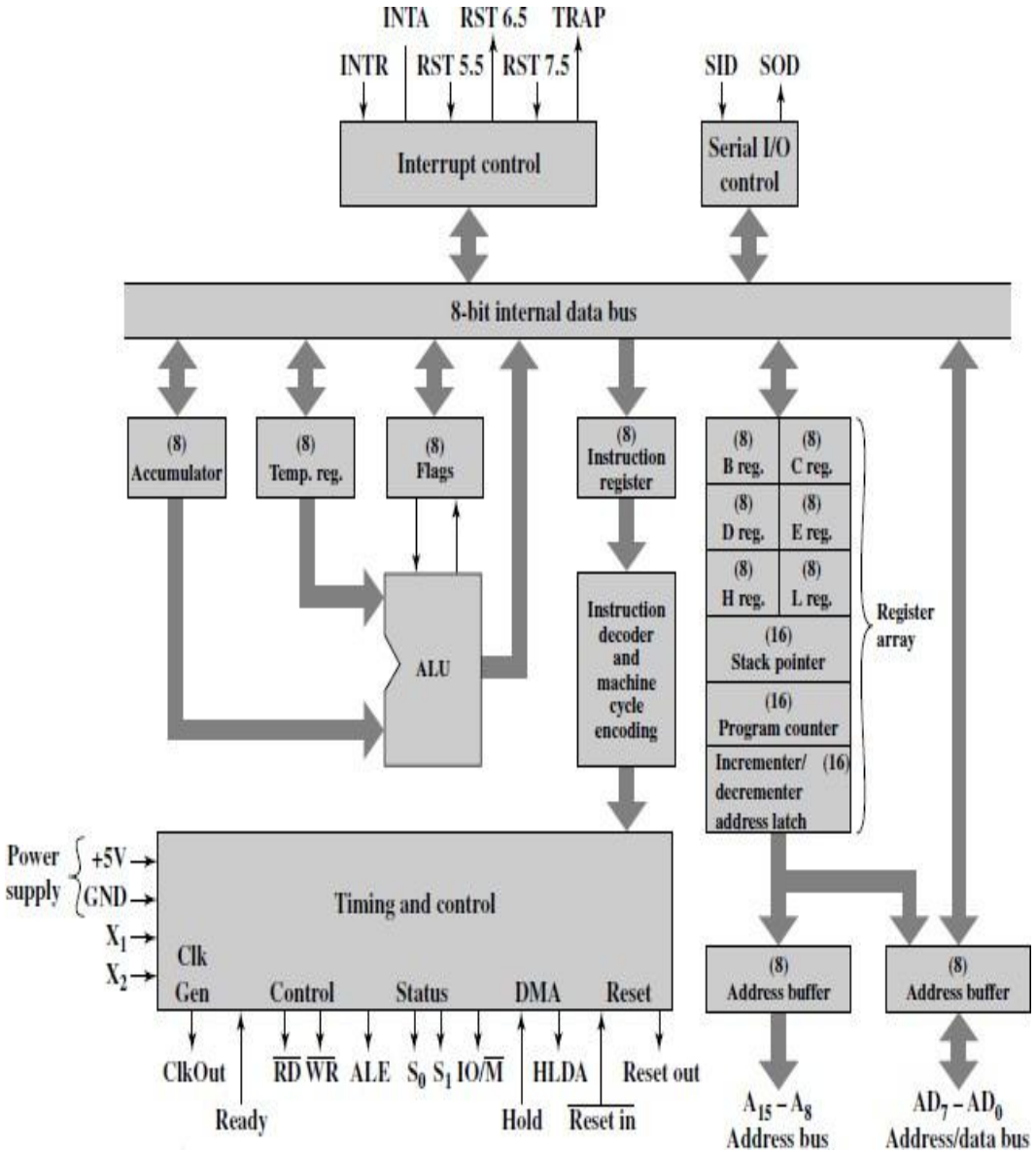


Figure 4.6 Intel 8085 CPU Block Diagram

Table 4.2 describes the external signals into and out of the 8085. These are linked to the external system bus. These signals are the interface between the 8085 processor and the rest of the system.

Address and Data Signals

High Address (A15–A8)
The high-order 8 bits of a 16-bit address.

Address/Data (AD7–AD0)
The lower-order 8 bits of a 16-bit address or 8 bits of data. This multiplexing saves on pins.

Serial Input Data (SID)
A single-bit input to accommodate devices that transmit serially (one bit at a time).

Serial Output Data (SOD)
A single-bit output to accommodate devices that receive serially.

Timing and Control Signals

CLK (OUT)
The system clock. The CLK signal goes to peripheral chips and synchronizes their timing.

X1, X2
These signals come from an external crystal or other device to drive the internal clock generator.

Address Latch Enabled (ALE)
Occurs during the first clock state of a machine cycle and causes peripheral chips to store the address lines. This allows the address module (e.g., memory, I/O) to recognize that it is being addressed.

Status (S0, S1)
Control signals used to indicate whether a read or write operation is taking place.

IO/M
Used to enable either I/O or memory modules for read and write operations.

Read Control (RD)
Indicates that the selected memory or I/O module is to be read and that the data bus is available for data transfer.

Write Control (WR)
Indicates that data on the data bus is to be written into the selected memory or I/O location.

Memory and I/O Initiated Symbols

Hold
Requests the CPU to relinquish control and use of the external system bus. The CPU will complete execution of the instruction presently in the IR and then enter a hold state, during which no signals are inserted by the CPU to the control, address, or data buses. During the hold state, the bus may be used for DMA operations.

Hold Acknowledge (HOLDA)
This control unit output signal acknowledges the HOLD signal and indicates that the bus is now available.

READY
Used to synchronize the CPU with slower memory or I/O devices. When an addressed device asserts READY, the CPU may proceed with an input (DBIN) or output (WR) operation. Otherwise, the CPU enters a wait state until the device is ready.

Interrupt-Related Signals

TRAP
Restart Interrupts (RST 7.5, 6.5, 5.5)

Interrupt Request (INTR)
These five lines are used by an external device to interrupt the CPU. The CPU will not honor the request if it is in the hold state or if the interrupt is disabled. An interrupt is honored only at the completion of an instruction. The interrupts are in descending order of priority.

Interrupt Acknowledge
Acknowledges an interrupt.

CPU Initialization

RESET IN
Causes the contents of the PC to be set to zero. The CPU resumes execution at location zero.

RESET OUT
Acknowledges that the CPU has been reset. The signal can be used to reset the rest of the system.

Voltage and Ground

VCC
+5-volt power supply

VSS
Electrical ground

Table 4.2 Intel 8085 External Signals

The control unit is identified as having two components labeled (1) instruction decoder and machine cycle encoding and (2) timing and control.

The essence of the control unit is the timing and control module.This module includes a clock and accepts as inputs the current instruction and some external control signals. Its output consists of control signals to the other components of the processor plus control signals to the external system bus.

The timing of processor operations is synchronized by the clock and controlled by the control unit with control signals. Each instruction cycle is divided into from one to five *machine cycles;* each machine cycle is in turn divided into from three to five *states.* Each state lasts one clock cycle. During a state, the processor performs one or a set of simultaneous micro-operations as determined by the control signals.

Figure 4.7 gives an example of 8085 timing, showing the value of external control signals. The diagram shows the instruction cycle for an OUT instruction. Three machine cycles (M1,M2,M3) are needed.

During the first, the OUT instruction is fetched.

The second machine cycle fetches the second half of the instruction, which contains the number of the I/O device selected for output.

During the third cycle, the contents of the AC are written out to the selected device over the data bus.

The Address Latch Enabled (ALE) pulse signals the start of each machine cycle from the control unit.The ALE pulse alerts external circuits.During timing state T1 of machine cycle M1, the control unit sets the IO/M signal to indicate that this is a memory operation.Also,the control unit causes the contents of the PC to be placed on the address bus (A15 through A8) and the address/data bus (AD7 through AD0). With the falling edge of the ALE pulse, the other modules on the bus store the address.

During timing state T2, the addressed memory module places the contents of the addressed memory location on the address/data bus.

The control unit sets the Read Control (RD) signal to indicate a read, but it waits until T3 to copy the data from the bus. This gives the memory module time to put the data on the bus and for the signal levels to stabilize. The final state, T4, is a *bus idle* state during which the processor decodes the instruction. The remaining machine cycles proceed in a similar fashion

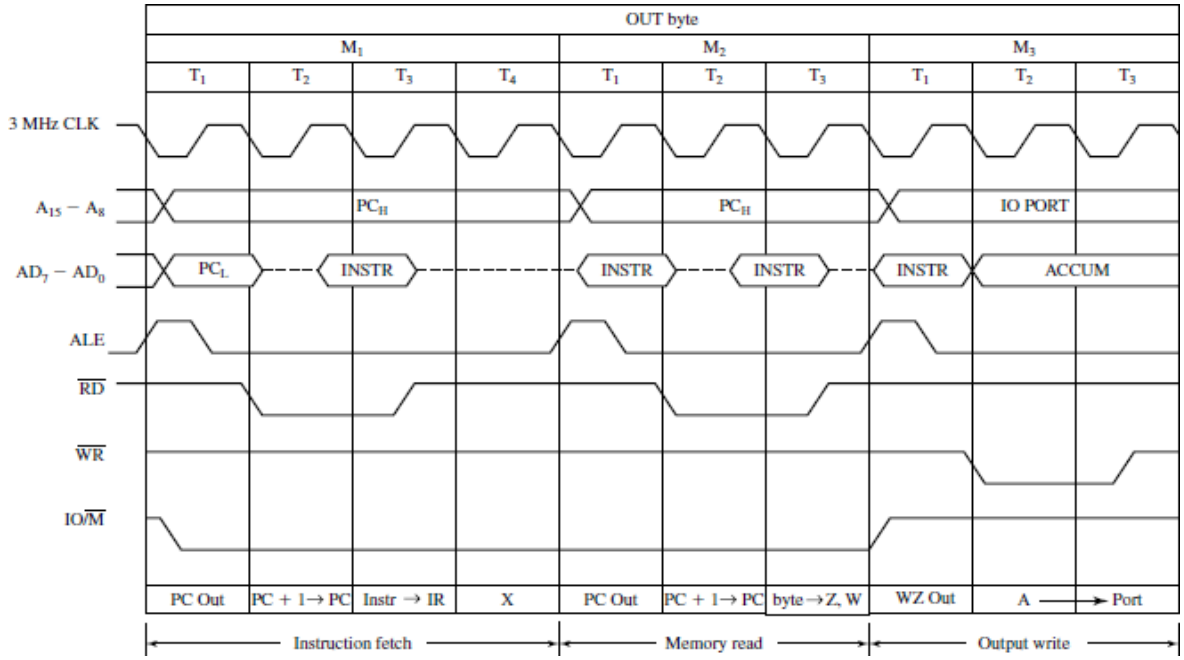


Figure 4.7 Timing Diagram for Intel 8085 OUT Instruction

❖ HARDWIRED IMPLEMENTATION

` A wide variety of techniques have been used for control unit implementation. Most of these fall into one of two categories:

- Hardwired implementation
- Microprogrammed implementation

In a hardwired implementation, the control unit is essentially a state machine circuit. Its input logic signals are transformed into a set of output logic signals, which are the control signals.

Control Unit Inputs

In Figure 4.4 the key inputs of the control unit are the instruction register, the clock, flags, and control bus signals. In the case of the flags and control bus signals, each individual bit typically has some meaning (e.g., over-flow). The other two inputs, however, are not directly useful to the control unit.

First consider the instruction register. The control unit makes use of the opcode and will perform different actions (issue a different combination of control signals) for different instructions. To simplify the control unit logic, there should be a unique logic input for each opcode. This function can be performed by a *decoder*

The clock portion of the control unit issues a repetitive sequence of pulses. This is useful for measuring the duration of micro-operations. The control unit emits different control signals at different time units within a single instruction cycle. So the input to the control unit,with a different control signal being used for T1, T2, and so forth. At the end of an instruction cycle, the control unit must feed back to the counter to reinitialize it at T1.

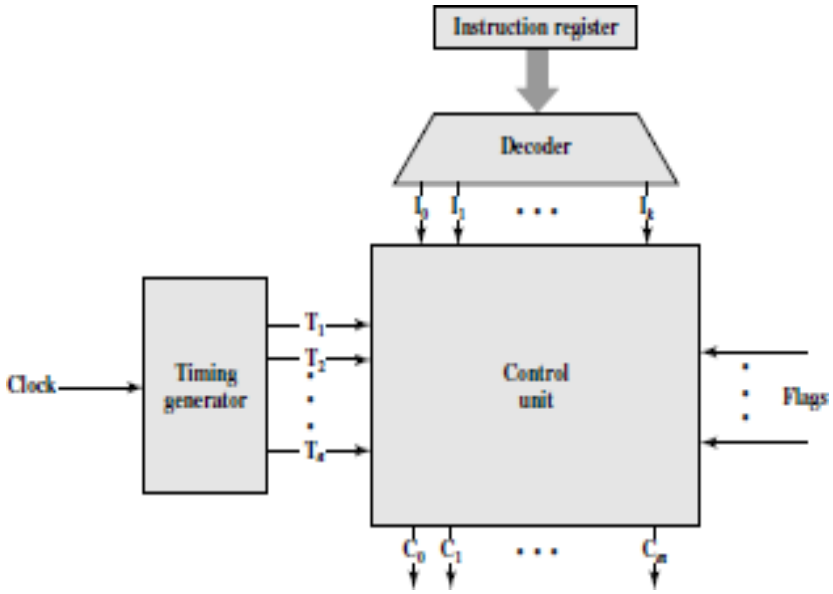


Figure 4.8 Control Unit with Decoded Inputs

With these two refinements, the control unit can be depicted as in Figure 4.8.

Control Unit Logic

To define the hardwired implementation of a control unit, all that remains is to discuss the internal logic of the control unit that produces output control signals as a function of its input signals. Essentially, what must be done is, for each control signal, to derive a Boolean expression of that signal as a function of the inputs.

Let us consider again our simple example illustrated in Figure 4.5. We saw in Table 4.1 the micro-operation sequences and control signals needed to control three of the four phases of the instruction cycle. Let us consider a single control signal, C5. This signal causes data to be read from the external data bus into the MBR. We can see that it is used twice in Table 4.1. Let us define two new control signals, P and Q, that have the following interpretation:

PQ = 00 Fetch Cycle

PQ = 01 Indirect Cycle

PQ = 10 Execute Cycle

PQ = 11 Interrupt Cycle

Then the following Boolean expression defines $C_5 = \overline{P} \cdot \overline{Q} \cdot T_2 + \overline{P} \cdot Q \cdot T_2$

That is, the control signal C5 will be asserted during the second time unit of both the fetch and indirect cycles.

This expression is not complete. C5 is also needed during the execute cycle. For our simple example, let us assume that there are only three instructions that read from memory: LDA, ADD, and AND. Now we can define C5 as

$$C_5 = \overline{P} \cdot \overline{Q} \cdot T_2 + \overline{P} \cdot Q \cdot T_2 + P \cdot \overline{Q} \cdot (LDA + ADD + AND) \cdot T_2$$

This same process could be repeated for every control signal generated by the processor. The result would be a set of Boolean equations that define the behavior of the control unit and hence of the processor.