Chapter 5: Control Unit

Micro operations

The execution of a program consists of the sequential execution of instruction. Each instruction is executed during in instruction cycle made of up of shorter sub cycles (e.g. fetch, indirect, execute, interrupt). The performance of each sub cycles involves one or more short operations called micro-operations.

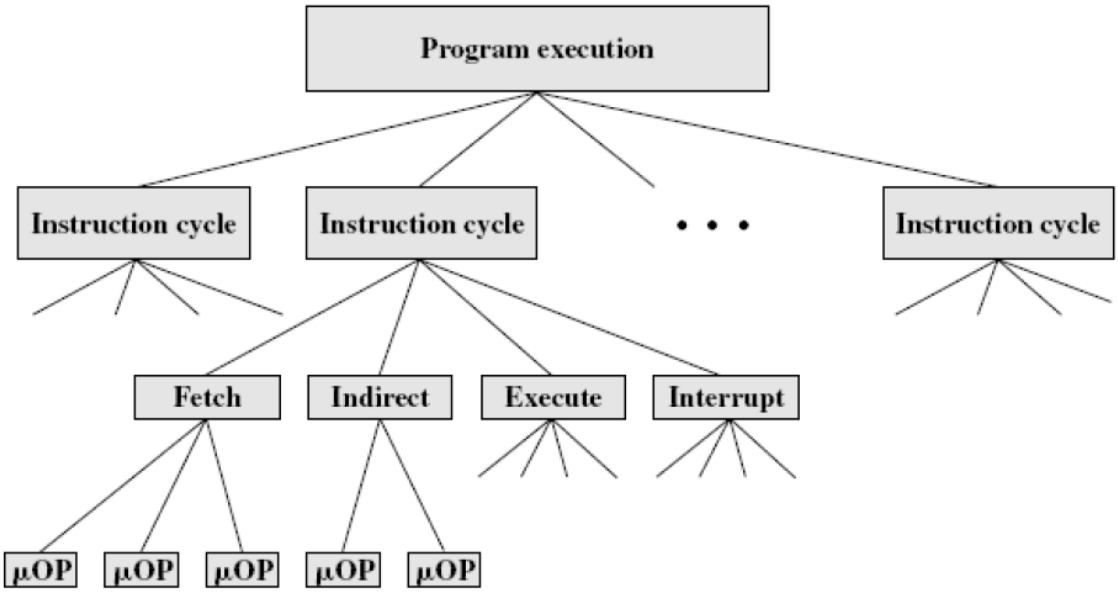


Figure 5.1 Constituents elements of a program execution

1. The Fetch Cycle

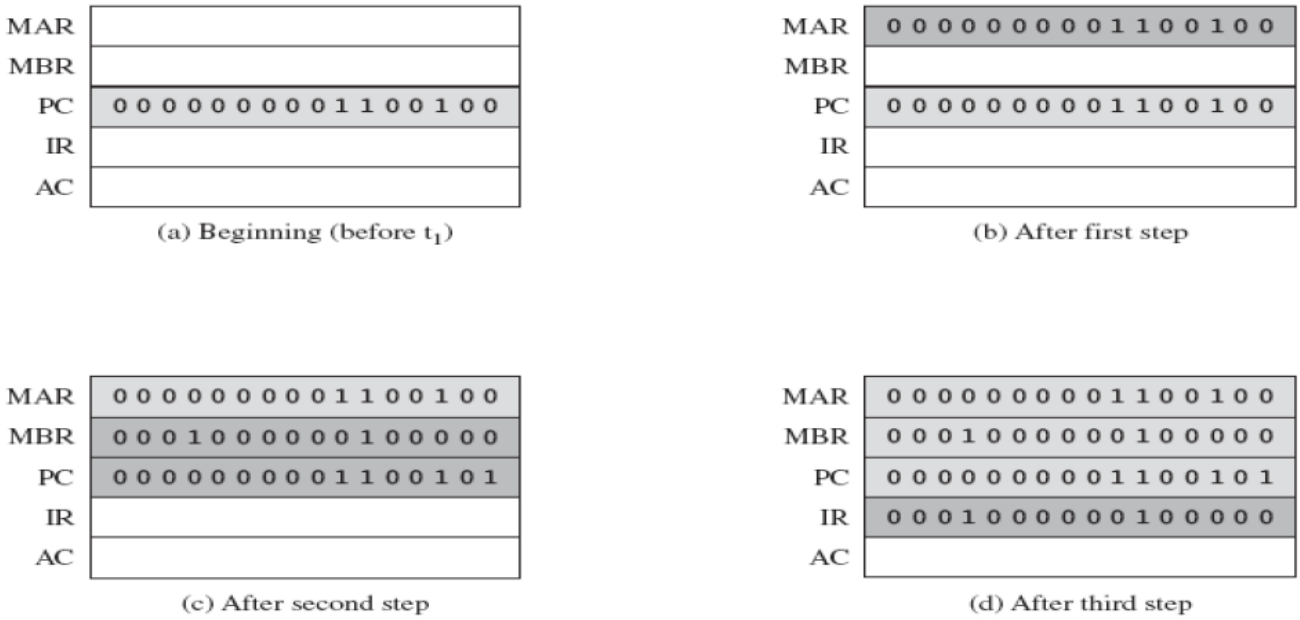
Fetch cycle occurs at the beginning of each instruction cycle and causes an instruction to be fetched from memory. Four registers are involved:

• **Memory address register (MAR):** It is connected to the address lines of the system bus. It specifies the address in memory for a read or writes operation.

• **Memory buffer register (MBR):** It 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.



Sequence of events

1. At the beginning of the fetch cycle the address of the next instruction to be executed is in PC.
2. The first step is to move the address to the MAR, because it is connected to address line.
3. Second step is to bring in the instruction. The desired address is placed on address bus, control unit issues READ command and data placed on MBR through data bus, also incremented PC by 1. Here the MBR load and PC incremented is done simultaneously without interference.
4. Third step is to move the content of MBR to the instruction register (IR).

Thus, the simple fetch cycle actually consists of three steps and four micro-operations. Each micro-operation involves the movement of data into or out of a register. So long as these movements do not interfere with one another, several of them can take place during one step, saving time. Symbolically, we can write this sequence of events as follows:

$$t_1: \quad MAR \leftarrow (PC)$$
$$t_2: \quad MBR \leftarrow Memory$$
$$\quad\quad PC \leftarrow (PC) + I$$
$$t_3: \quad IR \leftarrow (MBR)$$

Where I is the instruction length. We assume that a clock is available for timing purposes and that it emits regularly spaced clock pulses. Each clock pulse defines a time unit. Thus, all time units are of equal duration. Each micro-operation can be performed within the time of a single time unit. The notation represents successive time units. In words, we have (t1, t2, t3).

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:

$$t_1: \text{MAR} \leftarrow (\text{PC})$$
$$t_2: \text{MBR} \leftarrow \text{Memory}$$
$$t_3: \text{PC} \leftarrow (\text{PC}) + I$$
$$\text{IR} \leftarrow (\text{MBR})$$

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

i. 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.
ii. 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.

2. 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. It includes the following micro-operations:

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

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.

3. The Interrupt Cycle (how interrupt is handled)

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:

$$t_1: \text{MBR} \leftarrow (\text{PC})$$
$$t_2: \text{MAR} \leftarrow \text{Save\_Address}$$
$$\text{PC} \leftarrow \text{Routine\_Address}$$
$$t_3: \text{Memory} \leftarrow (\text{MBR})$$

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. However, because most processors provide multiple types and/or levels of interrupts, it may take one or more additional micro-operations to obtain the Save_Address and the Routine_Address before they can be transferred to the MAR and PC, respectively. In any case, once this is done, 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.

4. 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:

t_1: MAR ← (IR(address))
t_2: MBR ← Memory
t_3: 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. Again, this is a simplified example. Additional micro-operations may be required to extract the register reference from the IR and perhaps to stage the ALU inputs or outputs in some intermediate registers.

5. The Instruction Cycle:

Instruction cycle can be decomposed into a sequence of elementary micro operation. The complete picture with sequence of micro operations together is shown as in figure.

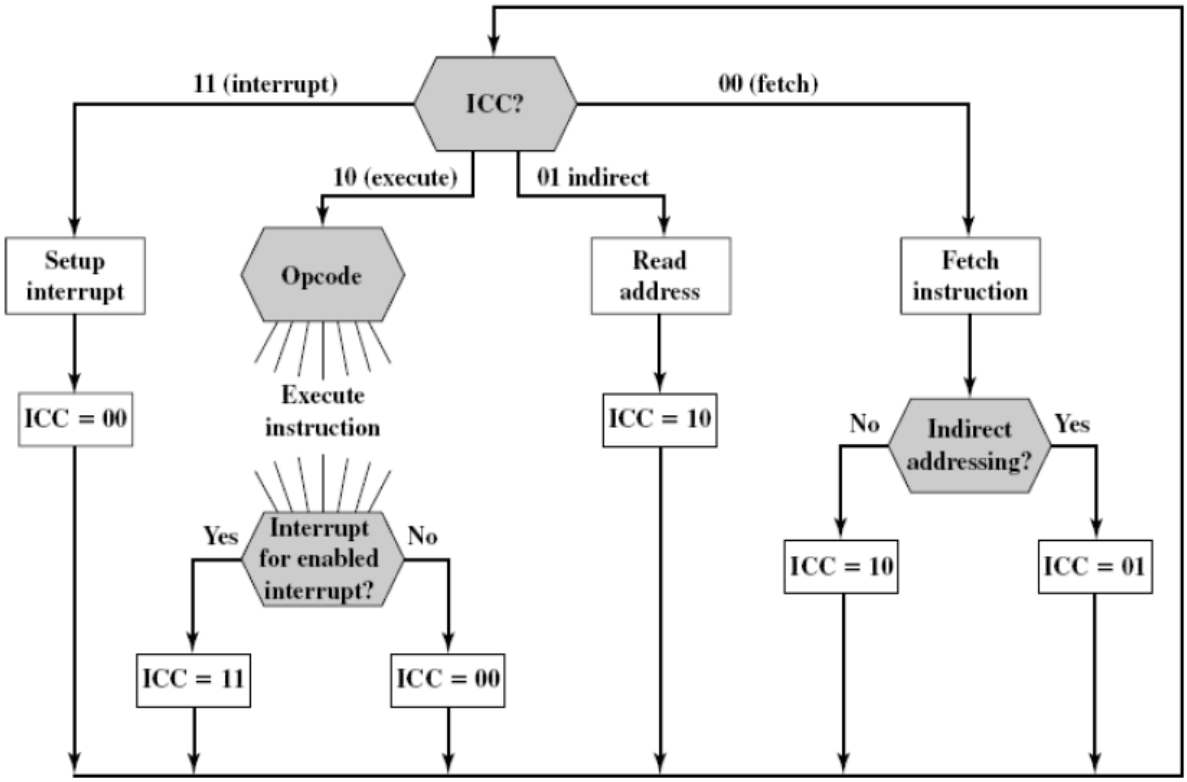


Figure 5.3: flowchart for instruction cycle.

We assume a new 2 bit register called instruction cycle code (ICC). The ICC designates the state of the processor in terms of which portion of the cycle it is:

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.

CONTROL OF THE PROCESSOR

The behavior or functioning of the processor into elementary operations, called micro-operations. 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.* A definition of these functional requirements is the basis for the design and implementation of the control unit. 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

The basic functional elements of the processor are the following:

- ALU
- Registers
- Internal data paths
- External data paths
- Control unit

The execution of a program consists of operations involving these processor elements. As we have seen, these operations consist of a sequence of micro-operations. 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.

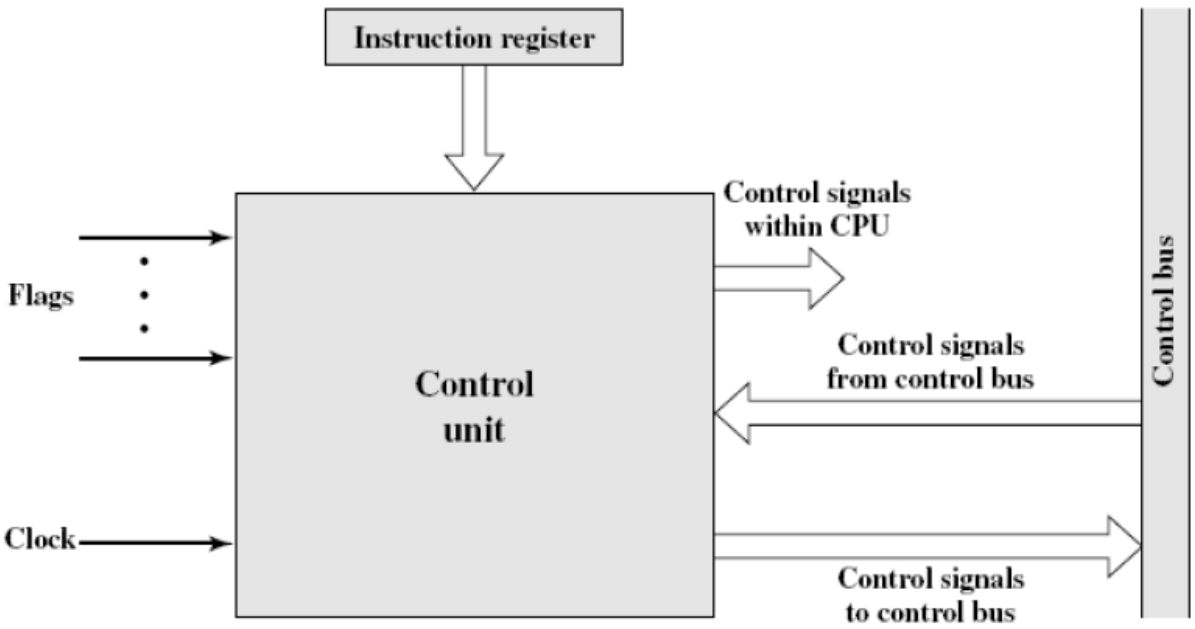


Figure 5.4 Block Diagram of Control Unit

Figure 5.4 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. Such as interrupt signal and acknowledgements.

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.

A Control Signals Example

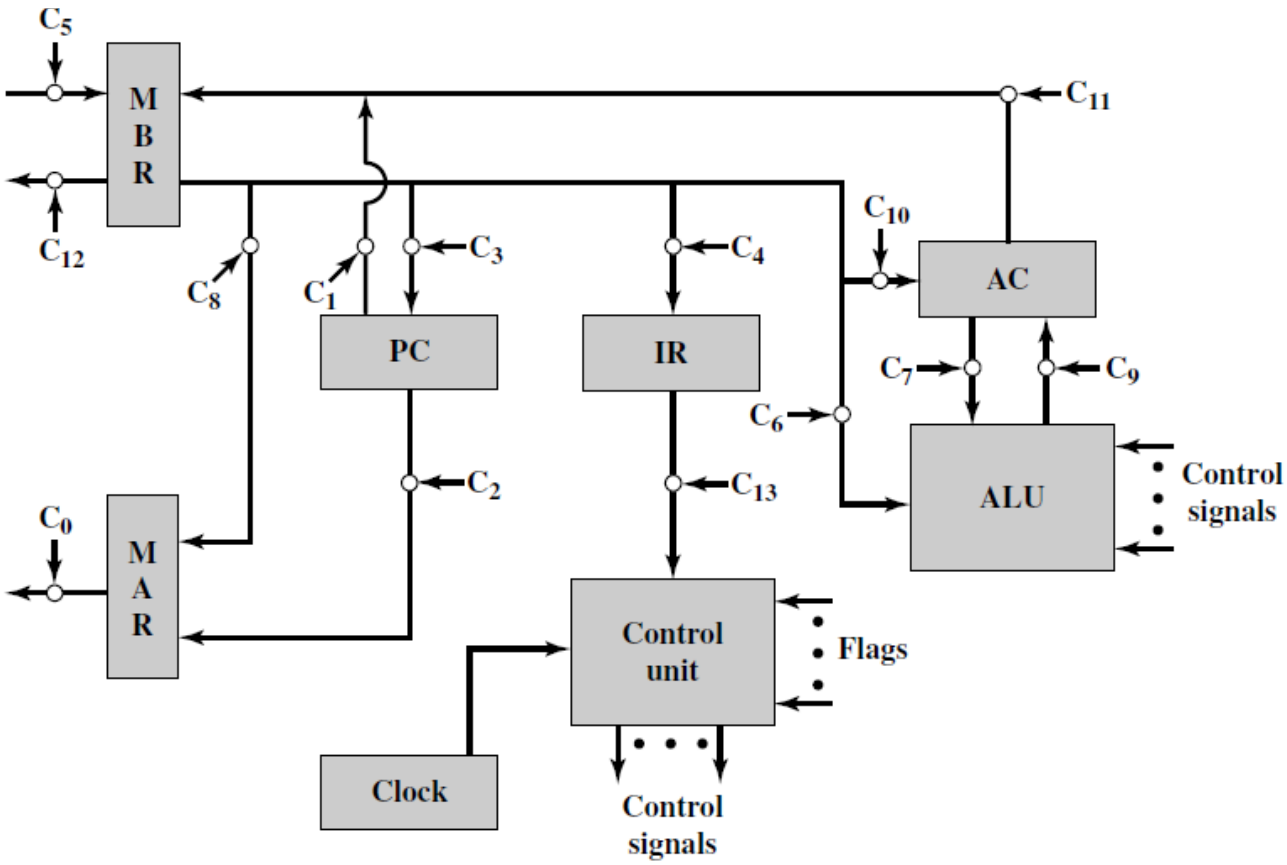


Figure 5.5 Data paths and Control signal

To illustrate the functioning of the control unit, let us examine a simple example. Figure 5.5 illustrates the example. 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. 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).

The control unit must maintain knowledge of where it is in the instruction cycle. Using this knowledge, and by reading all of its inputs, the control unit emits a sequence of control signals that causes micro-operations to occur. It uses the clock pulses to time the sequence of events, allowing time between events for signal levels to stabilize.
Table 5.1 indicates the control signals that are needed for some of the micro-operation sequences described earlier. For simplicity, the data and control paths for incrementing the PC and for loading the fixed addresses into the PC and MAR are not shown.

	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 5.1: Micro-operations and Control Signals

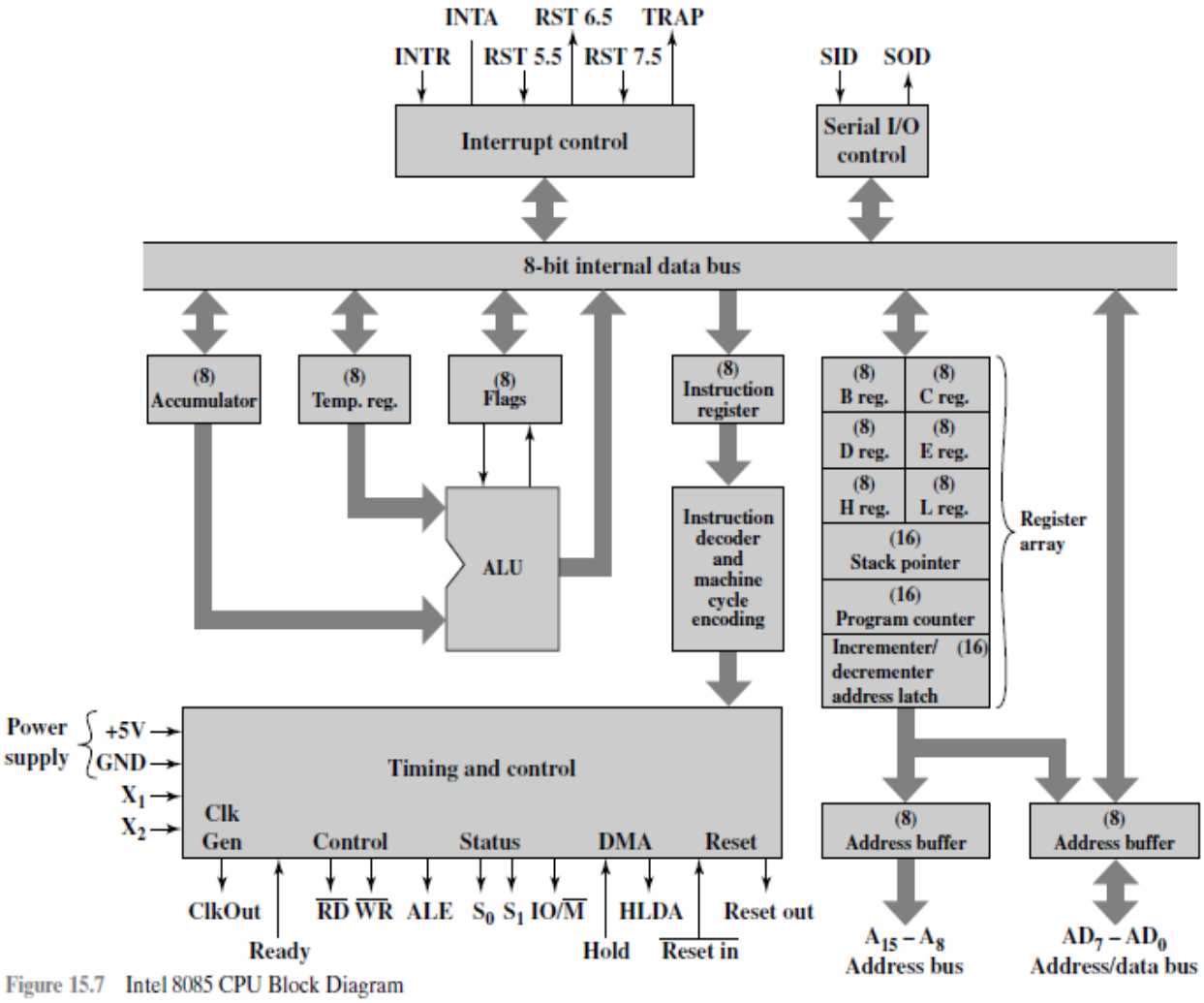


Figure 15.7 Intel 8085 CPU Block Diagram

Figure: 5.6 Intel 8085 CPU Block Diagram

Types of control unit

Till now we have discussed control unit in terms of inputs and function, now we focus our attention on the technique used for implementation.

1. Hardwired implementation
2. Micro programmed implementation

1. Hardwired implementation

In hardwired implementation, the control unit is essentially a combinational circuit. i.e. its input logic signals are transferred into a set of output logic signals which are control signal

Control unit inputs

The key inputs 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. Overflow).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 op-code 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, which takes an encoded input and produces a single output. In general, a decoder will have n binary inputs and 2^n binary outputs. Each of the 2^n different input patterns will activate a single unique output. The de-coder for a control unit will typically have to be more complex than that, to account for variable-length opcodes. The clock portion of the control unit issues a repetitive sequence of pulses. This is useful for measuring the duration of micro-operations. Essentially, the period of the clock pulses must be long enough to allow the propagation of signals along data paths and through processor circuitry. However, as we have seen, the control unit emits different control signals at different time units within a single instruction cycle. Thus, we would like a counter as 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. T_1.

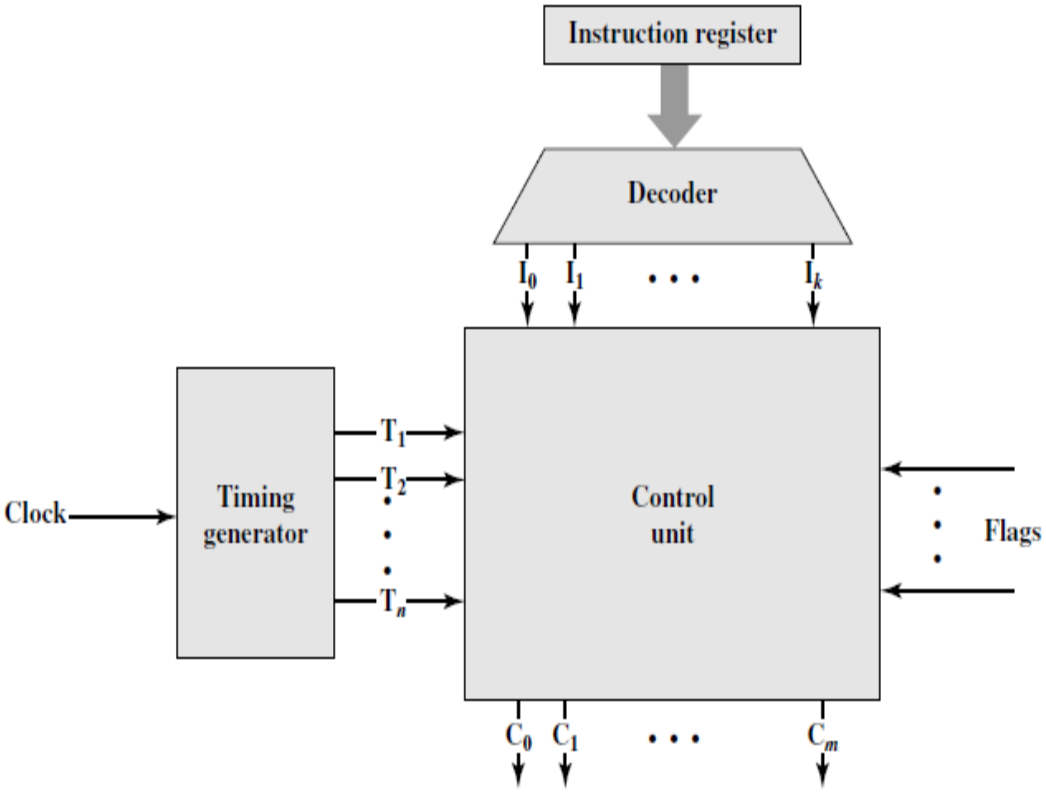


Figure 5.6: Control unit with decoded inputs.

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. 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 5.1. Let us define two new control signals, P and Q that have the following interpretation:

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

Then the following Boolean expression defines C5: as That is, the control signal C5 will be asserted during the second time unit of both the fetch and indirect cycles.

$$C_5 = \overline{P} \cdot \overline{Q} \cdot T_2 + \overline{P} \cdot Q \cdot T_2$$

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.

Problems with hardwired approach

- Complex sequencing and micro operations logic
- Difficult to design and test.
- Inflexible design.
- Difficult to add new instructions.

Micro programmed control unit

The concept of micro-programming was developed by Maurice wilkes in 1951, using diode matrices for memory element. A micro-program consists of sequence of micro-instruction in a micro- programming. Micro-programmed control unit is a relatively logic circuit that is capable of sequencing through micro-instruction and generating control signals to execute each micro-instruction. Consider that for each micro-operation all that the control unit is allowed is to do is generate a set of control signals. i.e. each control lines of control unit are either on or off. This condition can be represented by a binary digit for each control lines. So we would construct a control word in which each bit represents one control line. Then each micro-operation would be represented by a different pattern of 1's and 0's in control word. Now the sequence of control word or micro-operations is not fixed, so we put the control words in a memory location called control memory.

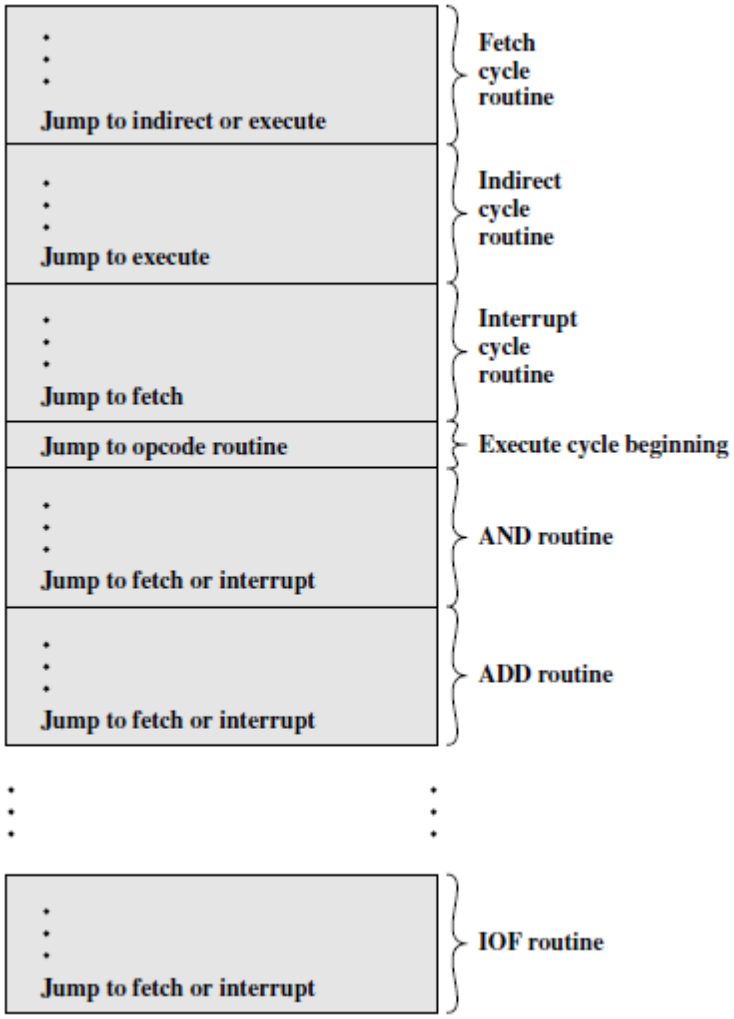


Figure 5.7: Organization of Control Memory

Figure shows the sequence of micro-operations to be performed during each cycle.

Working of micro-programmed implementation

The control memory of Figure5.7 contains a program that describes the behavior of the control unit. It follows that we could implement the control unit by simply executing that program. Figure 5.7 shows the key elements of such an implementation. The set of microinstructions is stored in the *control memory.* The *control address register* contains the address of the next microinstruction to be read. When a microinstruction is read from the control memory, it is transferred to a *control buffer register.*

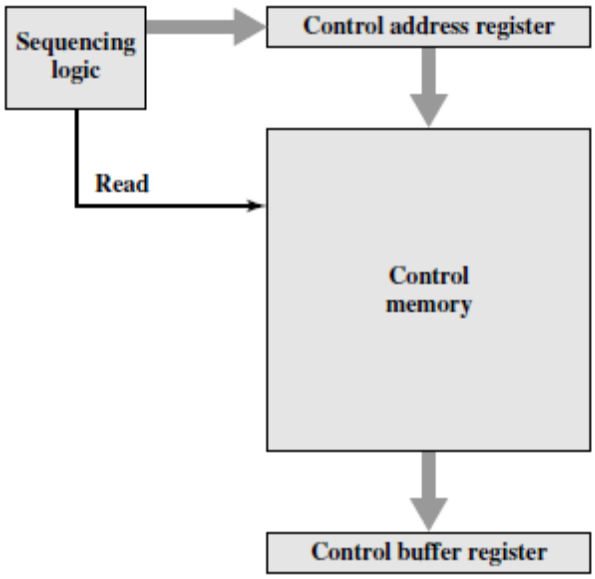


Figure 5.7: Control Unit micro architecture

Thus, reading a microinstruction from the control memory is the same as *executing* that microinstruction. The third element shown in the figure is a sequencing unit that loads the control address register and issues a read command. Let us examine this structure in greater detail, as depicted in Figure 16.4.Comparing this with Figure 16.4, we see that the control unit still has the same inputs (IR,ALU flags, clock) and outputs (control signals).The control unit functions as follows:

1. To execute an instruction, the sequencing logic unit issues a READ command to the control memory.
2. The word whose address is specified in the control address register is read into the control buffer register.
3. The content of the control buffer register generates control signals and next address information for the sequencing logic unit.
4. The sequencing logic unit loads a new address into the control address register based on the next-address information from the control buffer register and the ALU flags.

All this happens during one clock pulse. The last step just listed needs elaboration. At the conclusion of each microinstruction, the sequencing logic unit loads a new address into the Control address register. Depending on the value of the ALU flags and the control buffer register, one of three decisions is made:

• **Get the next instruction:** Add 1 to the control address register.
• **Jump to a new routine based on a jump microinstruction:** Load the address field of the control buffer register into the control address register.
• **Jump to a machine instruction routine:** Load the control address register based on the opcode in the IR.

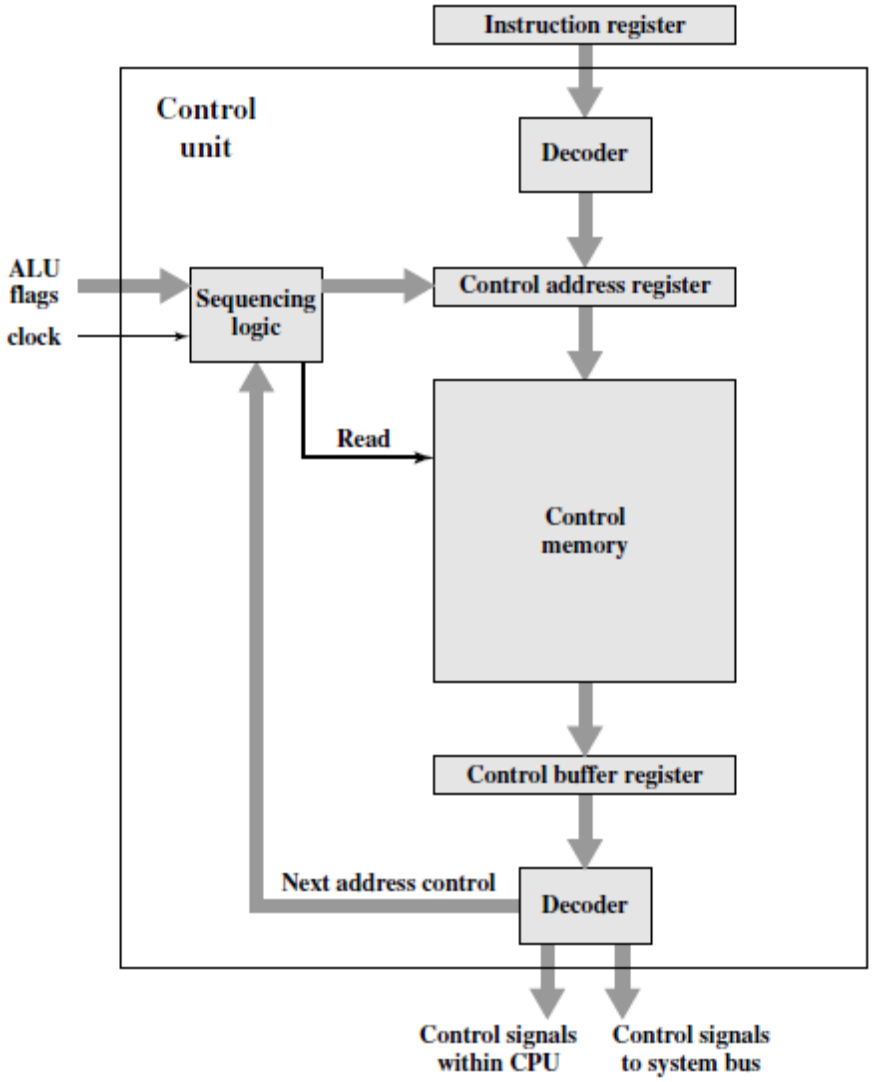


Figure 5.8: Functioning of Micro-programmed Control Unit

Advantage and Disadvantage

The principle advantage of the use of micro-programming is to implement a control unit is that it simplifies the design of control unit, so cheaper and less error prone. A hardwired control unit must contain complex logic for sequencing through the many micro-operations of instruction cycle. The decoder and sequencing logic unit of micro-programmed control unit are very simple piece of logic. Principle disadvantage of micro-programmed unit is that it will be slower than hardwired unit. Micro-programmed is a dominant technique for implementing control units in pure CICS architecture and hardware control unit in RISC architecture.

RISC: Reduced Instruction Set Computer
CISC: Complex Instruction Set Computer

The two basic tasks performed by a micro-programmed control unit are as follows:

1) Micro instruction sequencing: the micro-programmed control unit gets the next microinstruction from control memory.

2) Micro-instruction execution: the micro-programmed control unit generated the control signal needed to execute the micro- instruction. The control unit design must consider both which affect the format of the micro-instruction and the timing of control unit.

Micro- instruction sequencing

Two concerns are involved in the design of a microinstruction sequencing technique, the size of the microinstruction and the address-generation time. The first concern is obvious; minimizing the size of the control memory reduces the cost of that component. The second concern is simply a desire to execute microinstructions as fast as possible. In executing a micro-program, the address of the next microinstruction to be executed is in one of these categories:

- Determined by instruction register
- Next sequential address
- Branch

First occurs only once per instruction cycle, just after instruction is fetched. Second is the most common; however the design cannot be optimized just for sequential access. Third branches both conditional and unconditional are necessary and tend to occur more often.

Sequencing Techniques

Based on the current microinstruction, condition flags, and the contents of the instruction register, a control memory address must be generated for the next microinstruction. A wide variety of techniques have been used. We can group them into three general categories, as illustrated in Figures 16.6 to 16.8. These categories are based on the format of the address information in the microinstruction:

- Two address fields
- Single address field
- Variable format

Two address fields

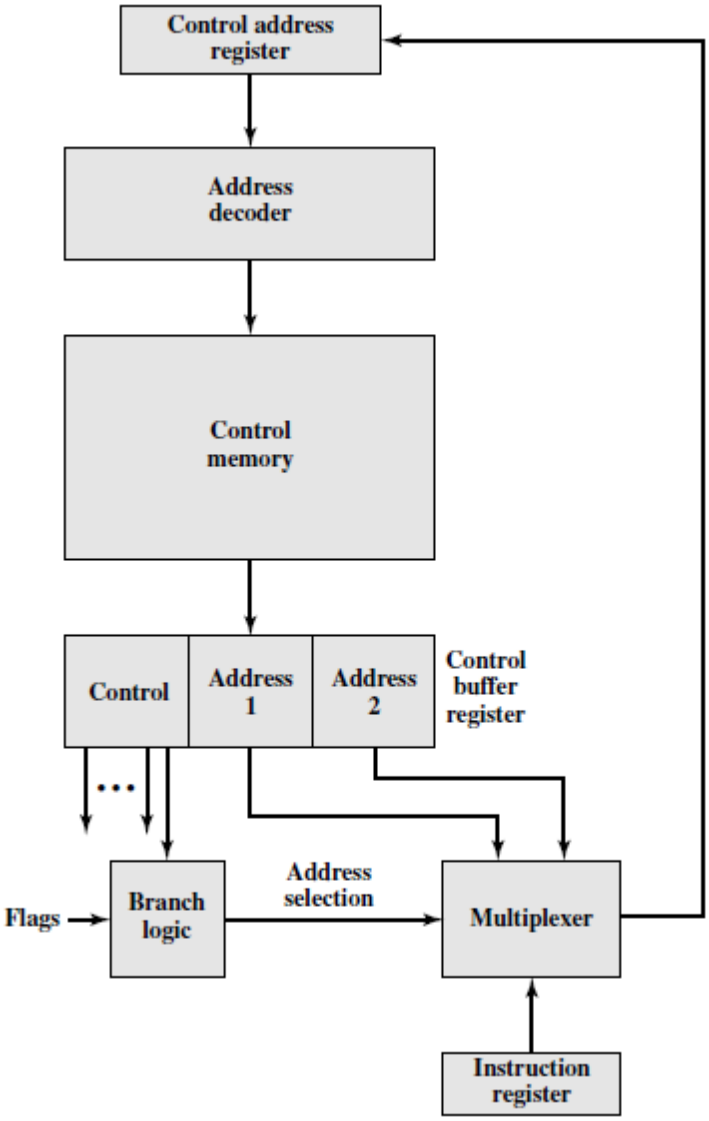


Figure 5.8: Branch Control Logic: Two Address field.

The simplest approach is to provide two address field in each micro-instruction. A multiplexer is provided that serves as a destination for both address fields plus the instruction register. Based on the address selection input multiplexer transmits either the op-code or one of the two address to Control Access Register (CAR). The CAR subsequently decodes to produce next micro-instruction address. The address selection signals are provided by a branch logic module whose input consists of control unit flags plus from control portion of micro-instruction.

Single address field

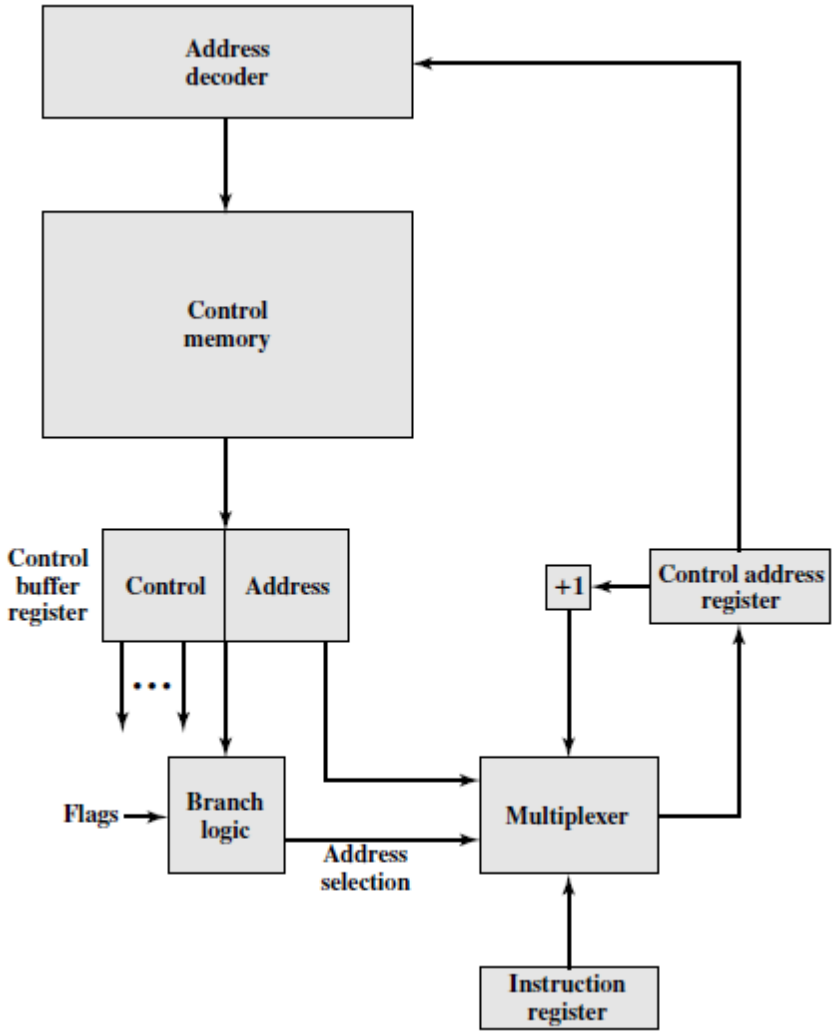


Figure 5.9: Branch Control Logic: Single Address field

A common approach is to use single address field, with this the next address can be specified by
- Address field
- Instruction register code
- Next sequential address

The address) selection signal determines which option is selected. This approach reduces the number of address fields to one.

Variable format

Another is to provide for two entirely different micro-instruction formats. One bit designates which format is being used, in one format the remaining bits are used to activate control signals. In other format some bits drive the branch logic module and the remaining bits provide the address. For the first format the next address is either the next sequential address or an address derived from the instruction register. With the second format either a conditional or unconditional branch is being specified. One dis-advantage of this approach is that one entire cycle is consumed with each branch micro-instruction.

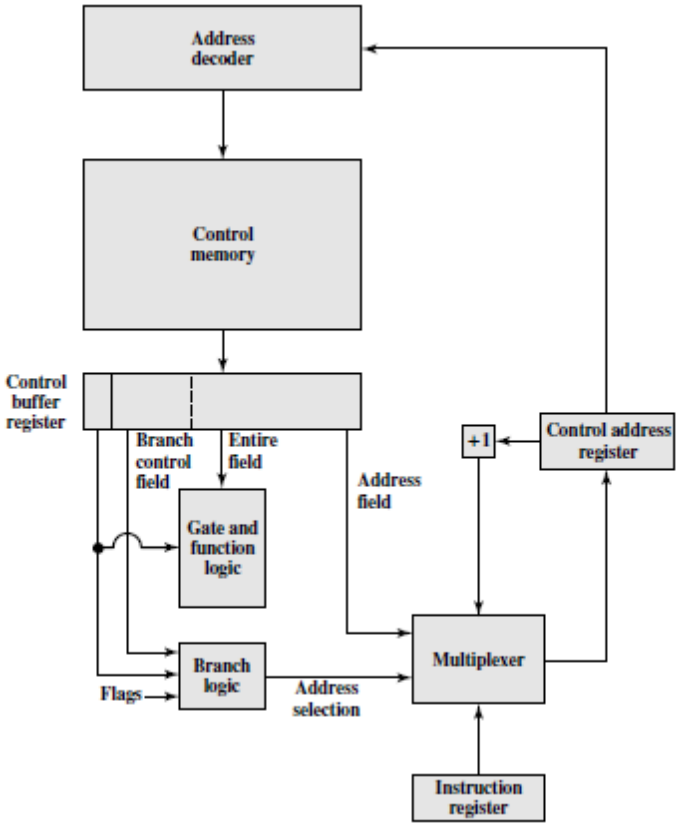


Figure 5.10: Branch Control Logic: Variable Format

Micro-Instruction execution

The micro-instruction is made up of two parts fetch and execute. The fetch is to fetch data and execute is to perform instruction. The fetch is to fetch data and execute is to perform instruction. The effect of the execution of a micro-instruction is to generate control signals for both internal control to processor and the external control to processor. An organization of control unit is:

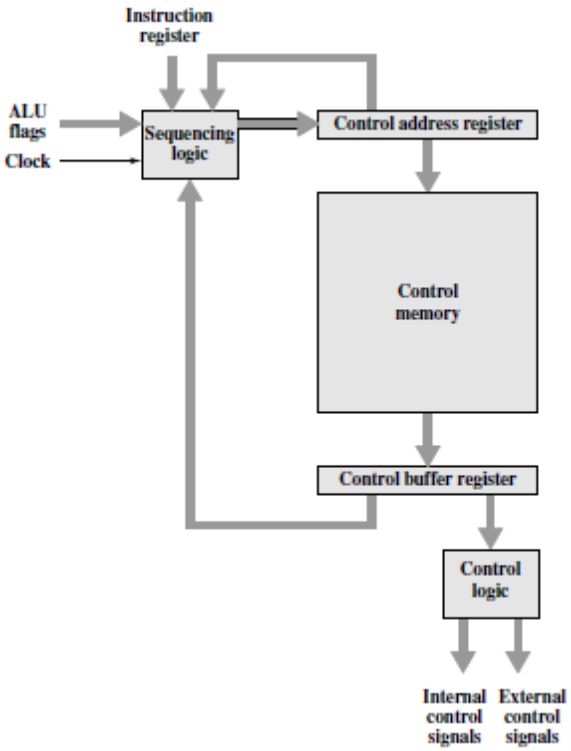


Figure 5.11: Control Unit Organization

Here are the major modules of control unit. The sequencing logic module contains the logic to perform the function. It generates address of the next micro-instruction using input as instruction register, ALU, flags, CAR and control buffer register. The last may provide an actual address control bits or both. The module is driven by clock that determines the timing of the micro-instruction cycle. Control logic module generates control signal as a function of some of bits in the micro-instruction. It should be clear that the format and content of the micro-instruction will determine the complexity of the control logic module.

Micro programming applications

For typically large microprocessor system today, there are many instruction and associated register level hardware. The are many control points to be manipulated.

Emulation:
- The use of a micro-program on one machine to execute programs originally written to run on another machine.

- By changing the microcode of a machine can execute software from another machine.

Classification of micro-instruction

- Vertical micro-programming
- Horizontal micro-programming

** Assignment to submit classification on micro-instruction

Reduced Instruction Set Computer (RISC)

In early 1980s a number of computer designers recommended that computers users use fewer instructions with a simple construct so that they can be executed much faster within the CPU without having to use memory as often. The concept of RISC architecture involves attempt to reduce the execution time by simplifying the instruction set of the computer.
The major characteristics of RISC processor are:

1. Relatively few instruction.
2. Few and simple addressing modes.
3. Memory access limited to load and store instruction.
4. All operations done within the register of CPU.
5. Fixed length, easily decoded instruction format.
6. Single cycle instruction execution.
7. Hardwired rather than micro programmed control.

Complex Instruction Set Computer (CISC)

The essential goal of a CISC architecture is to attempt to provide a single machine instruction for each statement that is written in high level language. The major characteristics of CICS architecture are:

1. Large number of instruction typically from hundreds to 250 instructions.
2. Some instruction that perform specialized tasks and are used infrequently.
3. A large variety of addressing modes typically from 5 to 20 different modes.
4. Variable length instruction formats.
5. Instruction that manipulate operands in memory.
6. Simple compilers will do the job.
7. Multiple cycle instruction (single HLL instruction is broken into smaller instruction).
8. Micro programmed implementation.
9. Simple control unit.