# **CHAPTER EIGHT**

# MICROPROGRAMMED CONTROL

### 8.1. Introduction

In any digital computer, the function of the control unit is to initiate sequences of microoperations. The number of different types of microoperations that are available in a given system is finite.

Generally, control unit of a digital computer may be designed using one of the following techniques:

### **1. Hardwired Control Unit**

In this type, the control signals are generated by hardware using conventional logic design techniques.

#### 2. Microprogrammed Control Unit

In this type, the control variables stored in memory at any given time can be represented by a string of 1's and 0's called a control word. As such, control words can be programmed to perform various operations on the components of the system.

Each word in control memory contains within it a **microinstruction**. Generally, a microinstruction specifies one or more microoperations. A sequence of microinstructions forms what is called a

microprogram.

Generally, a computer that employs a microprogrammed control unit will have two separate memories:

#### 1. The main memory

This memory is available to the user for storing programs. The user's program in main memory consists of machine instructions and data.

#### 2. The control memory

This memory contains a fixed microprogram that cannot alter by the occasional user. The microprogram consists of microinstructions that specify various internal control signals for execution of register microoperations.

Each instruction initiates a series of microinstructions in control memory. These microinstructions generate the microoperations to:

- 1. Fetch the instruction from main memory.
- 2. Evaluate the effective address.
- **3.** Execute the operation specified by the instruction.
- **4.** Finally, return the control to the fetch phase in order to repeat the cycle for the next instruction.

Figure 8.1 shows the general block diagram of a microprogrammed control unit which assumed to be a ROM.





The function of the **control address register** is to specify the address of the microinstruction, while the function of **control data register** is to holds the microinstruction read from memory.

The microinstruction contains a control word that specifies one or more microoperations for the data processor. Once these operations are executed, the control must determine the next address. The location of the next microinstruction may be the one next in sequence, or it may be located somewhere else in the control memory. For this reason it is necessary to use some bits of the present microinstruction to control the generation of the address of the next microinstruction. The next address may also be a function of external input conditions. While the microoperations are being executed, the next address is computed in the next address generator circuit and then transferred into the control address register to read the next microinstruction.

Thus a microinstruction contains bits for initiating microoperations in the data processor part and bits that determine the address sequence for the control memory.

The next address generator is sometimes called a microprogram sequencer, as it determines the address sequence that is read from control memory.

# Depending on the sequencer inputs, the address of the next microinstruction can be specified in several ways:

- 1. By incrementing control address register by one.
- **2.** Loading the control address register an address from control memory.
- 3. Transferring an external address.
- 4. Loading an initial address to start the control operations.

The control data register holds the present microinstruction while the next address is computed and read from memory. The data register is sometimes called a pipeline register. It allows the execution of the microoperations specified by the control word simultaneously with the generation of the next microinstruction. This configuration requires a two-phase clock, one clock applied to the address register and the other to the data register.

The system can operate without the control data register by applying a single-phase clock to the address register. The control word and next-address information are taken directly from the control memory.

It must be realized that a ROM operates as a combinational circuit, with the address value as the input and the corresponding word as the output. The content of the specified word in ROM remains in the output wires as long as its address value remains in the address register. No read signal is needed as in a random-access memory. Each clock pulse will execute the microoperations specified by the control word also transfer a new address to the control address register.

In the example that follows we assume a single-phase clock and therefore we do not use a control data register, in this way the address register is the only component in the control system that receives clock pulses. The other two components: the sequencer and the control memory are combinational circuits and do not need a clock.

The main advantage of the microprogrammed control is the fact that once the hardware configuration is established; there should be no need for further hardware or wiring changes. If we want to establish a different control sequence for the system, all we need to do is specify a different set of microinstructions for control memory (i.e. different microprogram residing in control memory).

## 8.2. Address Sequencing

Each computer has a set of instructions; each instruction has its own microprogram routine in control memory which generates the microoperations that execute it.

When the computer is turned on, an initial address is loaded into the control address register. This address usually presents the first address of the microinstruction that activates the instruction fetch routine. The fetch routine may be sequenced by incrementing the control address register through the rest of its microinstructions. At the end of the fetch routine, **the instruction is loaded in the instruction register**.

The next step is to determine the effective address of the operand. The effective address computation routine in control memory can be reached through a branch microinstruction, which is conditioned on the status of the mode bits of the instruction. When the effective address computation routine is completed, the address of the operand is available in the memory address register.

The following step is to generate the microoperations that execute the instruction fetched from memory. Each instruction has its own microprogram routine stored in a given location of control memory.

The process that transforms the instruction code bits to an address in control memory where the routine is located is referred as a **mapping process**. Once the required routine is reached, the microinstructions that execute the instruction may be sequenced by incrementing the control address register, but sometimes the sequence of microoperations will depend on values of certain status bits in processor registers. Microprograms that employ subroutines will require an external register to store the return address, since the return addresses cannot be stored in ROM.

When the execution of the instruction is completed, control must return to the fetch routine. This is accomplished by executing an unconditional branch microinstruction to the first address of the fetch routine. In summary, the address sequencing capabilities required in a control memory are:

- 1. Incrimination of the control address register.
- 2. Unconditional branch or conditional branch, depending on status bit conditions.
- 3. A mapping process from the bits of the instruction to an address for control memory.
- 4. A facility for subroutine call and return.

Figure 8.2 shows a block diagram of a control memory and the associated hardware needed for selecting the next microinstruction address.

#### Each microinstruction in control memory contains:

- 1. A set of bits to initiate microoperations in computer registers.
- **2.** A set of bits to specify the method by which the next microinstruction address is obtained.



The diagram shows four different paths from which the control address register (CAR) receives the address of the next instruction.

- **1.** By incrementer, which increments the content of the control address register by one.
- 2. By branching; this is achieved by specifying the branch address in one of the fields of the microinstruction. Conditional branching is obtained by using part of the microinstruction to select a specific status bit in order to determine its condition.
- **3.** An external address via a mapping logic circuit is transferred into control memory.
- **4.** The return address for a subroutine is stored in a special register whose value is then used when the microprogram wishes to return from the subroutine.

# **Conditional Branching**

The branch logic of Fig. 8.2 provides decision-making capabilities in the control unit.

# In every digital system, the status conditions are special bits that provide parameter information such as:

- 1. The carry-out of an adder.
- 2. The sign bit of a number.
- 3. The mode bits of an instruction.
- 4. Input or output status conditions.

The field which specifies a branch address in the microinstruction, together with the status bits, controls the conditional branch decisions generated in the branch logic.

The simplest way to implement the branch logic hardware is to test the specified condition and branch to the indicated address if the condition is met; otherwise, the address register is incremented.

This can be implemented using a multiplexer. For example, suppose that there are eight status bit conditions in the system. Therefore, three bits in the microinstruction are used to specify one of the eight status bit conditions. These three bits provide the selection variables for the multiplexer. If the selected status bit is in the 1 state, the output of the multiplexer is 1; otherwise, it is 0. A 1 output in the multiplexer generates a control signal to transfer the branch address from the microinstruction into the control address register. A 0 output in the multiplexer causes the address register to be incremented. In this configuration, the microprogram follows one of two possible paths, depending on the value of the selected status bit.

An unconditional branch microinstruction can be implemented by loading the branch address from control memory into the control address register via the multiplexer. This can be accomplished by fixing the value of one status bit at the input of the multiplexer, so it is always equal to 1. A reference to this bit by the status bit select lines from control memory causes the branch address to be loaded into the control address register unconditionally.

## Mapping of Instruction

A special type of branch exists when a microinstruction specifies a branch to the first word in control memory where a microprogram routine for an instruction is located. The status bits for this type of branch are the bits in the operation code part of the instruction.

**For example**, suppose a computer with an instruction format as shown in Fig. 8.3. four bits are specified for the operation code which specifies up to 16 distinct instructions. Assume further that the control memory has 128 words, requiring an address of seven bits. As mentioned before, the control memory includes a microprogram routine for each operation code that executes the corresponding instruction.

Computer Instruction	100	1	1	0	1		Address
Mapping Bits	0	x	x	x	x	0	0
Microinstruction Address	0	1	1	0	1	0	0
		Figure 8.3					e 8.3

One simple mapping process that converts the 4-bit operation code to a 7-bit address for control memory is shown in Fig. 8.3. This mapping consists of placing a 0 in the most significant bit of the address, transferring the four-operation code bits, and cleaning the two least significant bits of the control address register.

This provides for each computer instruction a microprogram routine with a capacity of four microinstructions, if the routine needs more than four microinstructions, it can use addresses 1000000 through 111111. If it uses fewer than four microinstructions, the unused memory locations would be available for other routines.

A more general mapping rule can be achieved by using a ROM to specify the mapping function. In this configuration, the bits of the instruction specify the address of a mapping ROM. The contents of the mapping ROM give the bits for the control address register.

#### The advantages of this mapping process are:

- 1. The microprogram routine that executes the instruction can be placed in any desired location in control memory.
- 2. It provides flexibility for adding new subroutines for new instructions in the control memory as the need arises.

## **Subroutines**

# Subroutines are programs that are used by other routine to accomplish a particular task.

An example of these subroutines is the subroutine needed to generate the effective address of the operand for an instruction. This is common to all memory reference instructions.

A subroutine can be called from any point within the main body of the microprogram.

## 8.3. Microprogram Example

When the configuration and the microprogrammed control unit of the computer are established, the designer's task is to generate the microcode for the control memory. This code generation is called microprogramming.

# **Computer Configuration**

To explain the microprogramming process, we present a simple digital computer similar but not identical to the basic computer introduced in chapter 6.

## Figure 8.4 shows the block diagram of this computer.



### F<u>igure 8.4</u>

### This computer consists of:

- 1. A main memory (2048 × 16) for storing instructions and data.
- 2. A control memory (128 × 20) for storing the microprogram.
- **3.** Four registers associated with the processor unit; there function is similar to the corresponding registers in the basic computer introduced in chapter 6. These registers are :
  - a. The Accumulator Register (AC).
  - **b.** The Program Counter Register (PC).
  - c. The Address Register (AR).
  - d. The Data Register (DR).
- 4. Two registers associated with control unit:
  - a. The Subroutine Register (SBR).
  - **b.** The Control Address Register (CAR).
- 5. The Arithmetic Logic and Shift Unit.
- **6.** Two multiplexers for transfer of information among the registers and the registers and main memory.

From the diagram, it's clear that **DR** can receive information from **AC**, **PC**, or main memory. **AR** can receive information either from **PC** or **DR**. **PC** receives information from **AR** only. The arithmetic, logic, and shift unit performs microoperations with data from **AC** and **DR** and places the result in **AC**. The memory receives its address from **AR**. Data written to memory comes from **DR**, while **DR** receives the data read from memory.

# **Instruction Format**

Figure 8.5 presents the instruction format for the assumed computer.

## The instruction format consists of three fields:

- 1. The 1-bit field specified for addressing mode symbolized by I.
- 2. The 4-bit field specified for operation code (Opcode).
- 3. The 11-bit field specified for the address.

