Supplement to

Logic and Computer Design Fundamentals
3rd Edition[1]

MICROPROGRAMMED CONTROL

Selected topics not covered in the third edition of *Logic and Computer Design Fundamentals* are provided here for optional coverage and for self-study if desired. This material fits well with the desired coverage in some programs but not may not fit within others due to time constraints or local preferences. This supplement from the seond edition of *Logic and Computer Design Fundamentals* is referenced in Section 8-7 as a part of the design of sequencing and control units. It introduces traditional microprogrammed control and gives a microprogrammed controlled version of the binary multiplier example.

A control unit with its binary control values stored as words in memory is called a *microprogrammed control*. Each word in the control memory contains a *microinstruction* that specifies one or more microoperations for the system. A sequence of microinstructions constitutes a *microprogram*. The latter is often fixed at the time of the system design and so is usually stored in ROM. Microprogramming involves placing some representation for combinations of values of control variables in words of ROM for use by the rest of the control logic via successive read operations. The contents of a word in ROM at a given address specify the microoperations to be performed for both the datapath and the control unit. A microprogram can also be stored in RAM. In this case, it is loaded initially at system startup from the computer console or from some form of nonvolatile storage, such as a magnetic disk. With either ROM or RAM, the memory in the control unit is called *control memory*; if RAM is used, the memory is referred to as *writable control memory.*

Figure 1 shows the general configuration of a microprogrammed control. The control memory is assumed to be a ROM within which all control information is permanently stored. The *control address register* (*CAR*) specifies the address of the microinstruction. The *control data register (CDR)*, which is optional, may hold the microinstruction currently being executed by the datapath and the control unit.

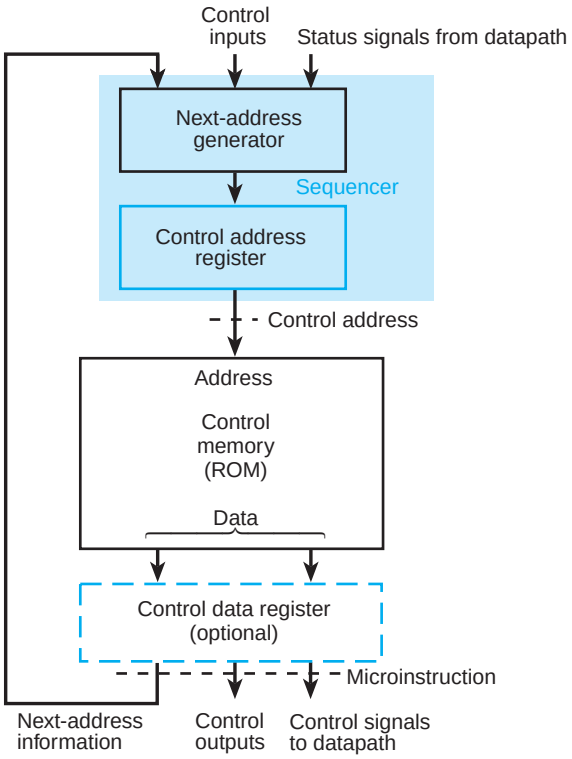

FIGURE 1
Microprogrammed Control Unit Organization

One of the functions of the control word is to determine the address of the next microinstruction to be executed. This microinstruction may be the next one in sequence, or it may be located somewhere else in the control memory. Therefore, one or more bits that specify how to determine the address of the next microinstruction must be present in the current microinstruction. The next address may also be a function of status and external control inputs. While a microinstruction is being executed, the *next-address generator* produces the next address. This address is transferred to the *CAR* on the next clock pulse and is used to read the next microinstruction to be executed from ROM. Thus, the microinstructions contain bits for activating microoperations in the datapath and bits that specify the sequence of microinstructions executed.

The next-address generator, in combination with the *CAR*, is sometimes called a microprogram *sequencer,* as it determines the sequence of instructions that is read from control memory. The address of the next microinstruction can be specified in several ways, depending on the sequencer inputs. Typical functions of a microprogram sequencer are incrementing the *CAR* by one and loading the *CAR*. Possible sources for the load operation include an address from control memory,

an externally provided address, and an initial address to start control unit operation.

The *CDR* holds the present microinstruction while the next address is being computed and the next microinstruction is being read from memory. The *CDR* breaks up a long combinational delay path through the control memory and the datapath. Insertion of this register is just like inserting a pipeline platform, as in Section 7-11; it allows the system to use a higher clock frequency and hence perform processing faster. The inclusion of a *CDR* in a system, however, complicates the sequencing of microinstructions, particularly when decision making based on status bits is involved. Hence, for simplicity, we omit the *CDR* and take the microinstructions directly from the ROM outputs. The ROM operates as a combinational circuit, with the address as the input and the corresponding microinstruction as the output. The contents of the specified word in ROM remain on the output lines of the ROM as long as the address value is applied to the inputs. No read/write signal is needed, as it is with RAM. Each clock pulse executes the microoperations specified by the microinstruction and also transfers a new address to the *CAR*, which, in this case, is the only component in the control that receives clock pulses and stores state information. The next-address generator and the control memory are combinational circuits. Thus, the state of the control unit is given by the contents of the *CAR*.

The status bits enter the next-address generator and affect the determination of the next state. Unless the status bits bypass the control unit and directly control the microoperations being executed in the datapath, they can do no more than select the next microoperation by affecting the address generated by the next-address generator. This has a profound effect on the structure of the ASM charts for microprogrammed controls. The sequential circuits must be Moore-type sequential circuits, and as a consequence, conditional output boxes are *not* permitted in the ASM charts. This often means that more states will be required in the ASM for a given hardware algorithm. An ASM chart for the binary multiplier, developed under the restriction that the system contain no conditional output boxes, is given in Figure 2. Compared to the ASM chart in Figure 8-7 in the text, this chart has two more states, INIT and ADD, that have been added where originally conditional output boxes were used. Besides being a Moore-type circuit, this ASM has only single decision boxes determining the sequencing between states. Although next-state decisions based on multiple values are possible, they are often excluded in simpler next-address generator designs.

Binary Multiplier Example

To illustrate microprogrammed control design, we consider again the binary multiplier. We need to determine three things: the bits in the control word for the microinstructions, the sizes of the ROM and the *CAR*, and the structure of the next-address generator. Then we can proceed to design the microsequencer and write the microprogram for binary multiplication.

We focus on the microoperations needed to perform the multiplications, ignoring those that load the multiplicand into register *B* and the multiplier into

register Q. From Table 8-1 in the text, we find that only four control signals are needed for the datapath to perform the multiplication: Initialize, Load, Clear_C, and Shift_dec. These control signals are repeated in Table 1. In addition, the corresponding register transfers are copied from Table 8-1 in the text. By comparing these transfers with those in the states in the microprogrammed control ASM chart for the multiplier in Figure 2 we can list the states in which each control signal is active. These states appear in the third column of Table 1. This information forms the foundation for designing the part of the microinstruction that controls the datapath.

The four control signals can be used as given or can be encoded to reduce the number of bits in the microinstruction. If the control signals are not encoded, four

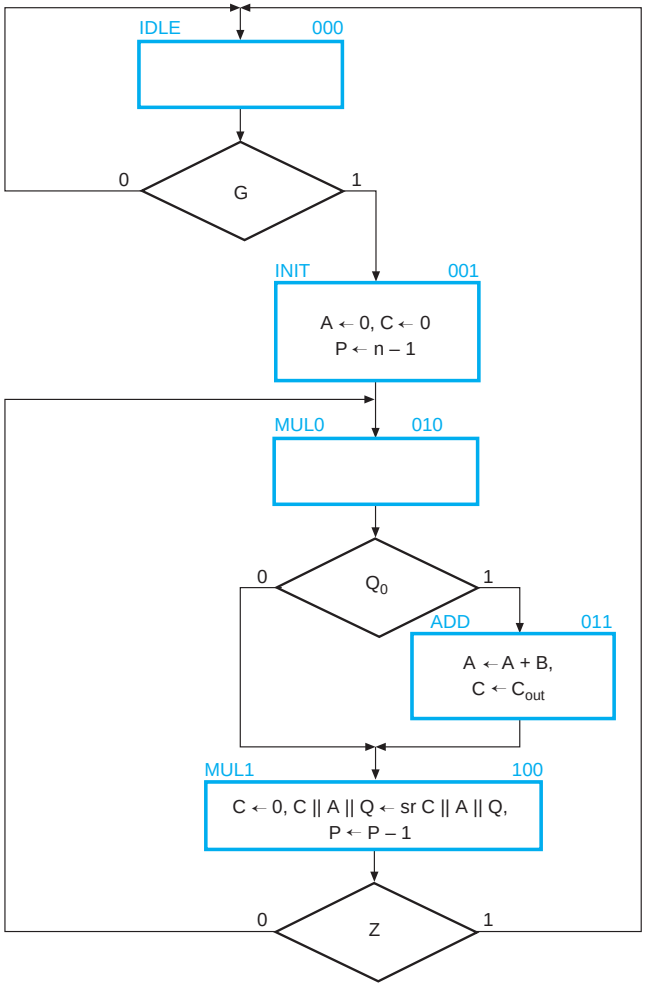

FIGURE 2
ASM Chart for Microprogrammed Binary Multiplier Control Unit

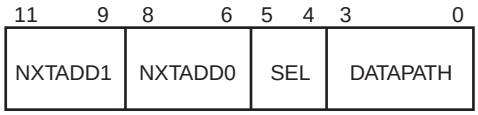

FIGURE 3
Microinstruction Control Word Format

bits, one for each of the control signals, are required in the DATAPATH field of the microinstruction. If the control signals are to be encoded in some fashion, there must be sufficient flexibility to encode all possible distinct combinations of the control signals needed. Suppose that we wish to use a single code word for each such combination. From Table 1, the three control signal combinations used are (Initialize, Clear_C) in state INIT, (Load) in state ADD and (Clear_C, Shift_dec) in state MUL1. In addition, for states IDLE and MUL1, we need the combination with no control signals active, giving a total of four distinct combinations. With encoding, the number of bits in the field of the microinstruction for controlling the datapath can be reduced to two, since four combinations can be represented using two bits. If this is done, then a decoder will be required at the ROM output to regenerate the original control signals. In that case, since only two bits are saved and the number of states is small, the saving in the size of ROM is unlikely to be sufficient to justify adding the decoder. As a consequence, we will not encode the control signals.

The bit position in the microinstruction format to be occupied by each control signal is given in the fourth column of Table 1. The fifth column lists the symbolic designation for each of the microoperations for use in writing microprograms. The format for the microinstruction control word for the binary multiplier control is given in Figure 3. The 4-bit codes based on Table 1 are used in the DATAPATH field of the word.

TABLE 1
Control Signals for Microprogrammed Multiplier Control

Control Signal	Register Transfers	States in Which Signal is Active	Micro-instruction Bit Position	Symbolic Notation
Initialize	$A \leftarrow 0, P \leftarrow n-1$	INIT	0	IT
Load	$A \leftarrow A+B, C \leftarrow C_{\text{out}}$	ADD	1	LD
Clear_C	$C \leftarrow 0$	INIT, MUL1	2	CC
Shift_dec	$C \Vert A \Vert Q \leftarrow \text{sr } C \Vert A \Vert Q, P \leftarrow P-1$	MUL1	3	SD

The remainder of the microinstruction control word is devoted to the sequencing of the control unit. There are many different ways to design the sequencer. The design method determines the fields needed in the microinstruc-

tion for sequencing. As a first step in this design, we consider the sequencing requirement defined by the ASM chart in Figure 2. In states INIT and ADD, the next state does not depend on status signals or inputs. In state IDLE, the next state depends on the value of *G*. In state MUL0, the next state depends on the value of Q_0. Finally, in state MUL1, the next state depends on the value of *Z*. For the cases where the next state depends on a status or input value, a pair of address values is needed, one for the input value equal to 0 and one for the input value equal to 1.

The approach used to define the addresses is a major decision in the sequencer design. There are many possible approaches, but two are most typical. One method includes the two addresses in the microinstruction controlling the decision. Based on the value of the decision variable, one of the two address values is loaded into the *CAR*. This method permits the arbitrary assignment of addresses to states and ensures that no states need to be added to provide the desired sequencing. But it requires two addresses in each microinstruction, potentially resulting in a long microinstruction word and wide ROM. The other method uses a counter with parallel load as the *CAR*. One of the two addresses is obtained from the microinstruction, but the other is obtained by simply counting up the *CAR*. This method requires at most one address per microinstruction word. But the assignment of the addresses to the states can be problematic, and states may have to be added to provide the desired sequencing. These states can slow the operation of the system due to the added clock cycles needed to pass through them.

TABLE 2
SEL Field Definition for Binary Multiplier Control Sequencing

SEL		
Symbolic notation	**Binary Code**	**Sequencing Microoperations**
NXT	00	$CAR \leftarrow NXTADD0$
DG	01	$\overline{G}$: $CAR \leftarrow NXTADD0$ G: $CAR \leftarrow NXTADD1$
DQ	10	$\overline{Q_0}$: $CAR \leftarrow NXTADD0$ Q_0: $CAR \leftarrow NXTADD1$
DZ	11	$\overline{Z}$: $CAR \leftarrow NXTADD0$ Z: $CAR \leftarrow NXTADD1$

Using the information in Table1, Table 2, and Figure 2, we now design the control unit. Based on the length of the microinstruction, the length of the ROM control words is 12 bits. Since there are only five states in Figure 2, the ROM contains 5 words. We need a 3-bit address to address 5 words and use a simple 3-bit

parallel load register as the *CAR*. The addresses to be loaded into the *CAR* come from NXTADD0 and NXTADD1 in the microinstruction on the ROM output. A quad 2–to–1 multiplexer on the data input to the *CAR* selects between these two address sources. The select signal *S* for this address multiplexer is determined on the basis of Table 2. A 4–to–1 multiplexer can be used to select the constant or decision variable required for each of the four SEL codes.

The resulting multiplier block diagram containing the components discussed, as well as the datapath, appears in Figure 4. The control memory is a ROM with a capacity of five 12-bit microinstructions. Four of the output bits of the ROM go to the datapath control inputs, while the remaining 8 bits determine the next address for the *CAR*. The SEL bits control the 4–to–1 multiplexer MUX2. For SEL at 00, MUX2 selects input 0, which has the value 0. A 0 on *S* of MUX1 selects NXTADD0 as the next address, as specified in Table 2. For SEL at 01, MUX 2 selects input 1, which is *G*. If *G* is 0, then the *S* on MUX1 is 0, selecting NXTADD0

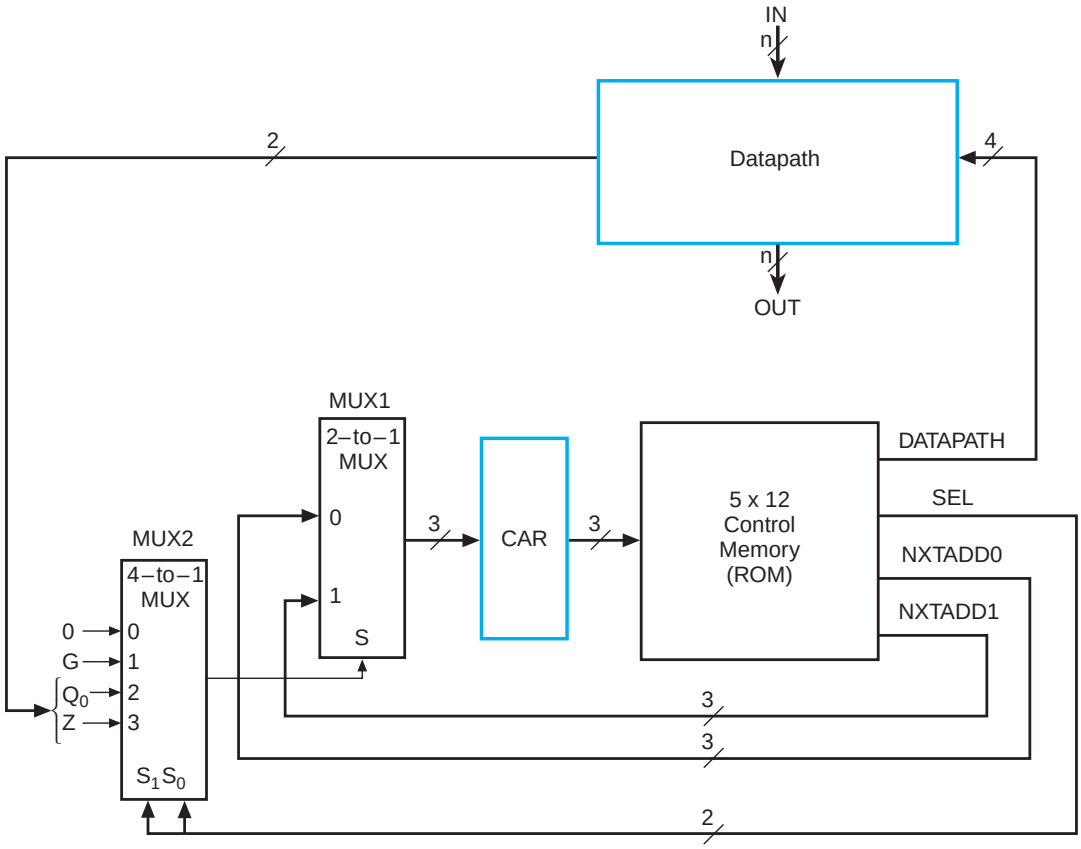

FIGURE 4
Microprogrammed Control Unit for Multiplier

as the address; if G is 1, the address becomes NXTADD1. The other two decision variables, Q_0 and Z, operate in a similar manner.

Table 4 gives a microprogram for the binary multiplier in register transfer notation. This description corresponds to the ASM in Figure 2. Note that there is a

TABLE 3
Register Transfer Description of Binary Multiplier Microprogram

Address	Symbolic transfer statement
IDLE	G: $CAR \leftarrow$ INIT, $\overline{G}$: $CAR \leftarrow$ IDLE
INIT	$C \leftarrow 0, A \leftarrow 0, P \leftarrow n-1, CAR \leftarrow$ MUL0
MUL0	Q_0: $CAR \leftarrow$ ADD, $\overline{Q_0}$: $CAR \leftarrow$ MUL1
ADD	$A \leftarrow A + B, C \leftarrow C_{out}, CAR \leftarrow$ MUL1
MUL1	$C \leftarrow 0, C \Vert A \Vert Q \leftarrow$ sr $C \Vert A \Vert Q$, Z: $CAR \leftarrow$ IDLE, $\overline{Z}$: $CAR \leftarrow$ MUL0, $P \leftarrow P-1$

TABLE 4
Symbolic Microprogram and Binary Microprogram for Multiplier

Address	NXTADD1	NXTADD0	SEL	DATAPATH	Address	NXTADD1	NXTADD0	SEL	DATAPATH
IDLE	INIT	IDLE	DG	None	000	001	000	01	0000
INIT	—	MUL0	NXT	IT, CC	001	000	010	00	0101
MUL0	ADD	MUL1	DQ	None	010	011	100	10	0000
ADD	—	MUL1	NXT	LD	011	000	100	00	0010
MUL1	IDLE	MUL0	DZ	CC, SD	100	000	010	11	1100

microinstruction in the microprogram that corresponds to each of the states in the ASM chart. On the chart, the binary code for each state is the contents of the CAR for that state. In the left half of Table 4, the register transfer microprogram is converted to a symbolic microprogram by replacing each register transfer with the symbolic names for the operation and using symbolic (state) names for the addresses. In the right half of the table, the symbolic microprogram is converted to a binary microprogram by replacing symbolic names with the corresponding binary codes from Figure 2, Table 1, and Table 2. In the binary microprogram, we have chosen to change unspecified symbolic entries to all zeros.

With the preceding introduction to hardwired and microprogrammed control unit design, we are now prepared to consider more complex control units for programmable digital systems. Our specific focus is simple computers, thereby building a basis for studying CPU designs in Chapters 9 through 12.

REFERENCES

1. MANO, M. M. AND C. R. KIME. *Logic and Computer Design Fundamentals, 3rd* ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2004.
2. MANO, M. M. AND C. R. KIME. *Logic and Computer Design Fundamentals, 2nd* ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2000.

PROBLEMS

The plus (+) indicates a more advanced problem.

8–1. A microprogrammed control unit is similar to the one in Figure 4, except that multiplexer MUX2 has 15 input status bits instead of 3; and the contro1 memory has 1,024 words. Formulate the microinstruction format, and specify the number of bits in each field. How many bits are required in each control memory word and in the entire control memory?

8–2. Multiply the two unsigned binary numbers 100110 (multiplicand) and 110101 (multiplier) by using both the hand method and the microprogrammed control in Figure 4 with the microprogram in Table 3 and Table 4. Give the *CAR* contents for each step, as well as the contents of the datapath registers.