PROBLEMS

7.1 Why is the Wait-for-Memory-Function-Completed step needed when reading from or writing to the main memory?

7.2 A processor uses a control sequence similar to that in Figure 7.6. Assume that a memory read or write operation takes the same time as one internal processor step and that both the processor and the memory are controlled by the same clock. Estimate the execution time of this sequence.

7.3 Repeat Problem 7.2 for a machine in which the memory access time is equal to twice the processor clock period.

7.4 Assume that propagation delays along the bus and through the ALU of Figure 7.1 are 0.3 and 2 ns, respectively. The setup time for the registers is 0.2 ns, and the hold time is 0. What is the minimum clock period needed?

7.5 Write the sequence of control steps required for the bus structure in Figure 7.1 for each of the following instructions:

(*a*) Add the (immediate) number NUM to register R1.

(*b*) Add the contents of memory location NUM to register R1.

(*c*) Add the contents of the memory location whose address is at memory location NUM to register R1.

Assume that each instruction consists of two words. The first word specifies the operation and the addressing mode, and the second word contains the number NUM.

7.6 The three instructions in Problem 7.5 have many common control steps. However, some of these control steps occur at different counts of the control step counter. Suggest a scheme that exploits these common steps to reduce the complexity of the encoder block in Figure 7.11.

7.7 Consider the Add instruction that has the control sequence given in Figure 7.6. The processor is driven by a continuously running clock, such that each control step is 2 ns in duration. How long will the processor have to wait in steps 2 and 5, assuming that a memory read operation takes 16 ns to complete? What percentage of time is the processor idle during execution of this instruction?

7.8 The addressing modes of a 32-bit, byte-addressable machine include autoincrement and autodecrement. In these modes, the contents of an address register are either incremented or decremented by 1, 2, or 4, depending on the length of the operand. Suggest some modification to Figure 7.1 to simplify this operation.

7.9 Show a possible control sequence for implementing the instruction

MUL R1,R2

on the processor in Figure 7.1. This instruction multiplies the contents of the registers R1 and R2, and stores the result in R2. Higher-order bits in the product, if any, are discarded. Suggest additional control signals as needed, and assume that the multiplier is organized as in Figure 6.7.

7.10 Show the control steps for the Branch-on-Negative instruction for a processor that has the structure given in Figure 7.8.

7.11 Show the control steps needed to implement the Branch-to-Subroutine instruction of one of the processors described in Chapter 3. Assume that processor has the internal organization of Figure 7.1.

7.12 Repeat Problem 7.11 for the processor in Figure 7.8.

7.13 Figure 7.3 shows an edge-triggered flip-flop being used for implementing the processor registers. Consider the operation of transferring data from one register to another. Examine the timing of this operation in detail and explain any potential difficulties that may be encountered if the edge-triggered flip-flop is replaced with a simple gated latch, such as that in Figure A.27.

7.14 The multiplexer and feedback connection in Figure 7.3 eliminate the need for gating the clock input as a means for enabling and disabling register input. Using a timing diagram, explain the problems that may arise if clock gating were used.

7.15 Assume that the register file in Figure 7.8 is implemented as a RAM. At any given time, a location in this RAM can be accessed for either a read or a write operation. During the operation R1 $\leftarrow$ [R1] + [R2], register R1 is both a source and a destination. Explain how you would use additional latches at either the input or the output of the RAM to operate the file in a master-slave mode. Use a timing diagram to explain how your new design enables register R1 to be used as both a source and a destination in the same clock cycle.

7.16 The Run signal in Figure 7.11 is set to 0 to prevent the control step counter from being advanced while waiting for a memory read or write operation to be completed. Examine the timing diagram in Figure 7.5, and prepare a state diagram for a control circuit that generates this signal. Design an appropriate circuit.

7.17 The MDR_{inE} control signal is asserted following a clock cycle in which the control signal Read is asserted and is negated when the memory transfer is completed, as shown in Figure 7.5. Design a suitable circuit to generate MDR_{inE}.

7.18 Consider a 16-bit, byte-addressable machine that has the organization of Figure 7.1. Bytes at even and odd addresses are transferred on the high- and low-order 8 bits of the memory bus, respectively. Show a suitable gating scheme for connecting register MDR to the memory bus and to the internal processor bus to allow byte transfers to occur. When a byte is being handled, it should always be in the low-order byte position inside the processor.

7.19 Design an oscillator using an inverter and a delay element. Assuming that the delay element introduces a delay T, what is the frequency of oscillation?

Modify the oscillator circuit such that oscillations can be started and stopped under the control of an asynchronous input RUN. When the oscillator is stopped, the width of the last pulse at its output must be equal to T, independent of the time at which RUN becomes inactive.

7.20 Some control steps in a processor take longer to complete than others. It is desired to generate a clock signal controlled by a signal called Long/Short such that the duration of a control step is twice as long when this signal is equal to 1. Assume that the control step counter has an Enable input and that the counter is advanced on the positive edge of the clock if Enable = 1. Design a circuit that generates the Enable signal to vary the size of the control steps as needed.

7.21 The output of a shift register is inverted and fed back to its input, to form a counting circuit known as a Johnson counter.

(*a*) What is the count sequence of a 4-bit Johnson counter, starting with the state 0000?

(*b*) Show how you can use a Johnson counter to generate the timing signals T_1, T_2, and so on in Figure 7.11, assuming there is a maximum of 10 timing intervals.

7.22 An ALU of a processor uses the shift register shown in Figure P7.1 to perform shift and rotate operations. Inputs to the control logic for this register consist of

ASR	Arithmetic Shift Right
LSR	Logic Shift Right
SL	Shift Left

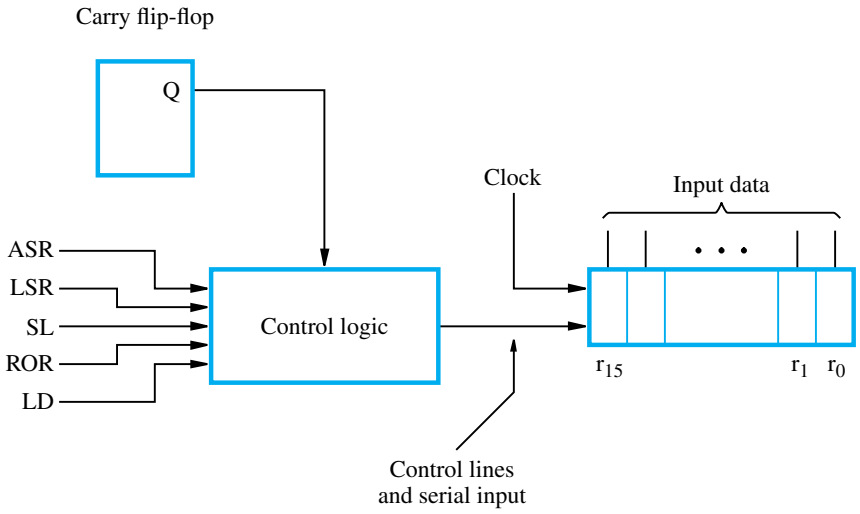


Figure P7.1 Organization of shift-register control for Problem 7.22.

ROR Rotate Right

LD Parallel Load

All shift and load operations are controlled by one clock input. The shift register is implemented with edge-triggered D flip-flops. Give a complete logic diagram for the control logic and for bits r_0, r_1, and r_{15} of the shift register.

7.23 The digital controller in Figure P7.2 has three outputs, X, Y, and Z, and two inputs, A and B. It is externally driven by a clock. The controller is continuously going through the following sequence of events: At the beginning of the first clock cycle, line X is set to 1. At the beginning of the second clock cycle, either line Y or Z is set to 1, depending on whether line A was equal to 1 or 0, respectively, in the previous clock cycle. The controller then waits until line B is set to 1. On the following positive edge of the clock, the controller sets output Z to 1 for the duration of one clock cycle, then resets all output signals to 0 for one clock cycle. The sequence is repeated, starting at the next positive edge of the clock. Draw a state diagram and give a suitable logic design for this controller.

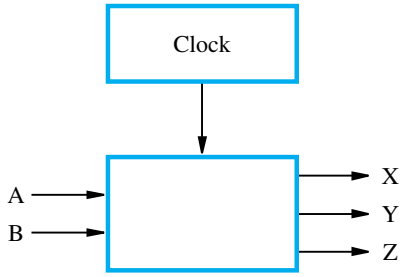


Figure P7.2 Digital controller in Problem 7.23.

7.24 Write a microroutine, such as the one shown in Figure 7.21, for the instruction

MOV X(Rsrc),Rdst

where the source and destination operands are specified in indexed and register addressing modes, respectively.

7.25 A BGT (Branch if > 0) machine instruction has the expression $Z + (N \oplus V) = 0$ as its branch condition, where Z, N, and V are the zero, negative, and overflow condition flags, respectively. Write a microroutine that can implement this instruction. Show the circuitry needed to test the condition codes.

7.26 Write a combined microroutine that can implement the BGT (Branch if > 0), BPL (Branch if Plus), and BR (Branch Unconditionally) instructions. The branch conditions for the BGT and BPL instructions are $Z + (N \oplus V) = 0$ and $N = 0$, respectively. What is the total number of microinstructions required? How many microinstructions are needed if a separate microroutine is used for each machine instruction?

7.27 Figure 7.21 shows an example of a microroutine in which bit-ORing is used to modify microinstruction addresses. Write an equivalent routine, without using bit-ORing, in

which conditional branch microinstructions are used. How many additional microinstructions are needed? Assume that the conditional branch microinstructions can test some of the bits in the IR.

7.28 Show how the microprogram in Figure 7.20 should be modified to implement the 68000 microprocessor instruction

ADD src,Rdst

7.29 Explain how the flowchart in Figure 7.20 can be modified to implement the general instruction

MOVE src,dst

in which both the source and the destination can be in any of the five address modes shown.

7.30 Figure P7.3 gives part of the microinstruction sequence corresponding to one of the machine instructions of a microprogrammed computer. Microinstruction B is followed

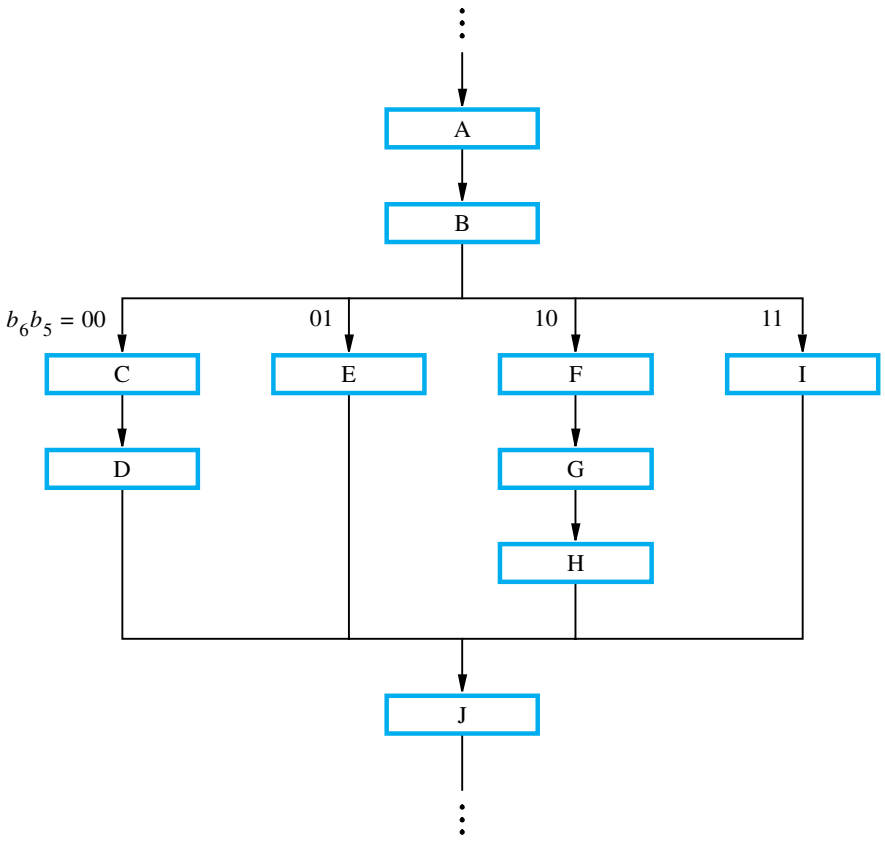


Figure P7.3 A microinstruction-sequence pattern used in Problem 7.30.

by C, E, F, or I, depending on bits b_6 and b_5 of the machine instruction register. Compare the three possible implementations described below.

(*a*) Microinstruction sequencing is accomplished by means of a microprogram counter. Branching is achieved by microinstructions of the form

If b_6b_5 branch to X

where b_6b_5 is the branch condition and X is the branch address.

(*b*) Same as Part *a* except that the branch microinstruction has the form

Branch to X, OR

where X is a base branch address. The branch address is modified by bit-ORing of bits b_5 and b_6 with the appropriate bits within X.

(*c*) A field in each microinstruction specifies the address of the next microinstruction, which has bit-ORing capability.

Assign suitable addresses for all microinstructions in Figure P7.3 for each of the implementations in Parts *a* through *c*. Note that you may need to insert branch instructions in some cases. You may choose arbitrary addresses, as long as they are consistent with the method of sequencing used. For example, in Part *a*, you could choose addresses as follows:

Address	Microinstruction
00010	A
00011	B
00100	If $b_6b_5 = 00$ branch to XXXXX
...	...
XXXXX	C

7.31 It is desired to reduce the number of bits needed to encode the control signals in Figure 7.19. Suggest a new encoding that reduces the number of bits by two. How does the new encoding affect the number of control steps needed to implement an instruction?

7.32 Suggest a new encoding for the control signals in Figure 7.19 that reduces the number of bits needed in a microinstruction to 12. Show the effect of the new encoding on the control sequences in Figures 7.6 and 7.7.

7.33 Suggest a format for microinstructions, similar to Figure 7.19, if the processor is organized as shown in Figure 7.8.

7.34 What are the relative merits of horizontal and vertical microinstruction formats? Relate your answer to the answers to Problems 7.31 and 7.32.

7.35 What are the advantages and disadvantages of hardwired and microprogrammed control?

Chapter 7 – Basic Processing Unit

7.1. The WMFC step is needed to synchronize the operation of the processor and the main memory.

7.2. Data requested in step 1 are fetched during step 2 and loaded into MDR at the end of that clock cycle. Hence, the total time needed is 7 cycles.

7.3. Steps 2 and 5 will take 2 cycles each. Total time = 9 cycles.

7.4. The minimum time required for transferring data from one register to register Z is equal to the propagation delay + setup time
= 0.3 + 2 + 0.2 = 2.5 ns.

7.5. For the organization of Figure 7.1:

(*a*) 1. PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2. Z_{out}, PC_{in}, Y_{in}, WMFC
3. MDR_{out}, IR_{in}
4. PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
5. Z_{out}, PC_{in}, Y_{in}
6. $R1_{out}$, Y_{in}, WMFC
7. MDR_{out}, SelectY, Add, Z_{in}
8. Z_{out}, $R1_{in}$, End

(*b*) 1-4. Same as in (*a*)
5. Z_{out}, PC_{in}, WMFC
6. MDR_{out}, MAR_{in}, Read
7. $R1_{out}$, Y_{in}, WMFC
8. MDR_{out}, Add, Z_{in}
9. Z_{out}, $R1_{in}$, End

(*c*) 1-5. Same as in (*b*)
6. MDR_{out}, MAR_{in}, Read, WMFC
7-10. Same as 6-9 in (*b*)

7.6. Many approaches are possible. For example, the three machine instructions implemented by the control sequences in parts *a*, *b*, and *c* can be thought of as one instruction, Add, that has three addressing modes, Immediate (Imm), Absolute (Abs), and Indirect (Ind), respectively. In order to simplify the decoder block, hardware may be added to enable the control step counter to be conditionally loaded with an out-of-sequence number at any time. This provides a ”branching” facility in the control sequence. The three control sequences may now be merged into one, as follows:

1-4. Same as in (*a*)
5. Z_{out}, PC_{in}, If Imm branch to 10

6. WMFC
7. MDR$_{out}$, MAR$_{in}$, Read, If Abs branch to 10
8. WMFC
9. MDR$_{out}$, MAR$_{in}$, Read
10. R1$_{out}$, Y$_{in}$, WMFC
11. MDR$_{out}$, Add, Z$_{in}$
12. Z$_{out}$, R1$_{in}$, End

Depending on the details of hardware timing, steps, 6 and 7 may be combined. Similarly, steps 8 and 9 may be combined.

7.7. Following the timing model of Figure 7.5, steps 2 and 5 take 16 ns each. Hence, the 7-step sequence takes 42 ns to complete, and the processor is idle 28/42 = 67% of the time.

7.8. Use a 4-input multiplexer with the inputs 1, 2, 4, and Y.

7.9. With reference to Figure 6.7, the control sequence needs to generate the Shift right and Add/Noadd (multiplexer control) signals and control the number of additions/subtractions performed. Assume that the hardware is con gured such that register Z can perform the function of the accumulator, register TEMP can be used to hold the multiplier and is connected to register Z for shifting as shown. Register Y will be used to hold the multiplicand. Furthermore, the multiplexer at the input of the ALU has three inputs, 0, 4, and Y. To simplify counting, a counter register is available on the bus. It is decremented by a control signal Decrement and it sets an output signal Zero to 1 when it contains zero. A facility to place a constant value on the bus is also available.

After fetching the instruction the control sequence continues as follows:

4. Constant=32, Constant$_{out}$, Counter$_{in}$
5. R1$_{out}$, TEMP$_{in}$
6. R2$_{out}$, Y$_{in}$
7. Z$_{out}$, if TEMP$_0 = 1$ then SelectY else Select0, Add, Z$_{in}$, Decrement
8. Shift, if Zero=0 then Branch 7
9. Z$_{out}$, R2$_{in}$, End

7.10. The control steps are:

1-3. Fetch instruction (as in Figure 7.9)
4. PC$_{out}$, Offset- eld-of-IR $_{out}$, Add, If N = 1 then PC$_{in}$, End

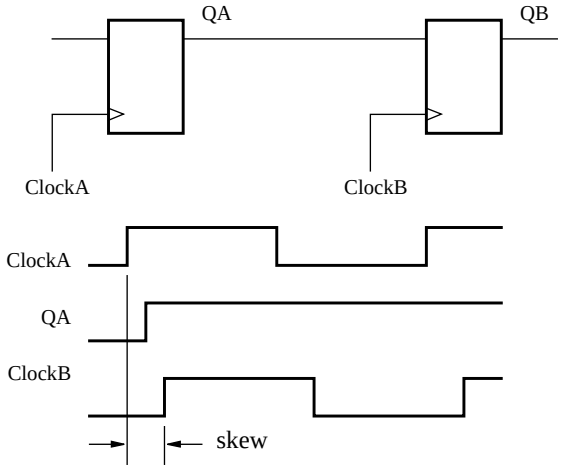
7.11. Let SP be the stack pointer register. The following sequence is for a processor that stores the return address on a stack in the memory.

1-3. Fetch instruction (as in Figure 7.6)
4. SP_{out}, Select4, Subtract, Z_{in}
5. Z_{out}, SP_{in}, MAR_{in}
6. PC_{out}, MDR_{in}, Write, Y_{in}
7. Offset- eld-of-IR $_{out}$, Add, Z_{in}
8. Z_{out}, PC_{in}, End, WMFC

7.12. 1-3. Fetch instruction (as in Figure 7.9)
4. SP_{outB}, Select4, Subtract, SP_{in}, MAR_{in}
5. PC_{out}, R=B, MDR_{in}, Write
6. Offset- eld-of-IR $_{out}$, PC_{out}, Add, PC_{in}, WMFC, End

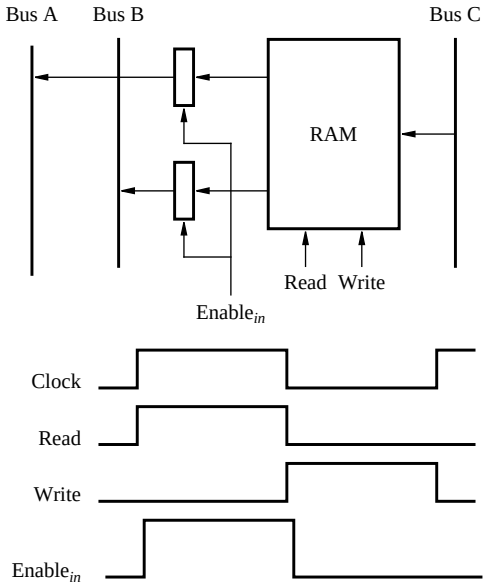
7.13. The latch in Figure A.27 cannot be used to implement a register that can be both the source and the destination of a data transfer operation. For example, it cannot be used to implement register Z in Figure 7.1. It may be used in other registers, provided that hold time requirements are met.

7.14. The presence of a gate at the clock input of a ip- op introduces clock skew. This means that clock edges do not reach all ip- ops at the same time. For example, consider two ip- ops A and B, with output QA connected to input DB. A clock edge loads new data into A, and the next clock edge transfers these data to B. However, if clock B is delayed, the new data loaded into A may reach B before the clock and be loaded into B one clock period too early.

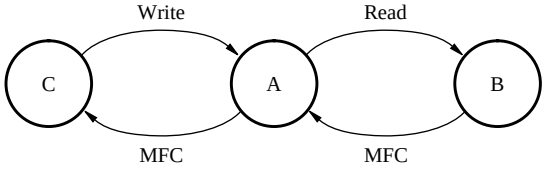


In the absence of clock skew, ip- op B records a 0 at the rst clock edge. However, if Clock B is delayed as shown, the ip- op records a 1.

7.15. Add a latch similar to that in Figure A.27 at each of the two register file outputs. A read operation is performed in the RAM in the first half of a clock cycle and the latch inputs are enabled at that time. The data read enter the two latches and appear on the two buses immediately. During the second phase of the clock the latch inputs are disabled, locking the data in. Hence, the data read will continue to be available on the buses even if the outputs of the RAM change. The RAM performs a write operation during this phase to record the results of the data transfer.

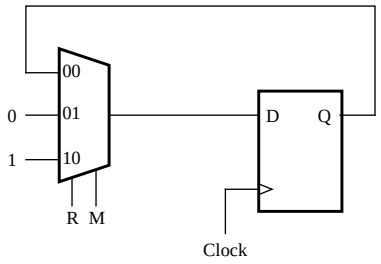


7.16. The step counter advances at the end of a clock period in which Run is equal to 1. With reference to Figure 7.5, Run should be set to 0 during the first clock cycle of step 2 and set to 1 as soon as MFC is received. In general, Run should be set to 0 by WMFC and returned to 1 when MFC is received. To account for the possibility that a memory operation may have been already completed by the time WMFC is issued, Run should be set to 0 only if the requested memory operation is still in progress. A state machine that controls bus operation and generates the run signal is given below.

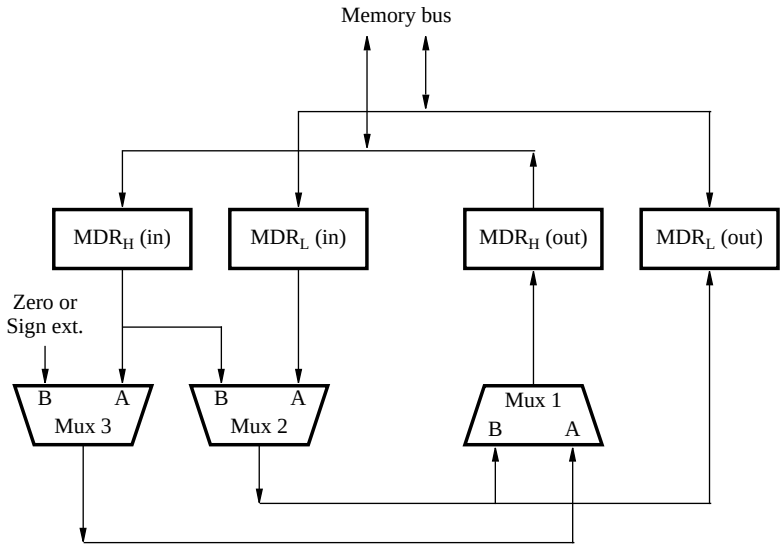


Run = WNFC · (B + C)

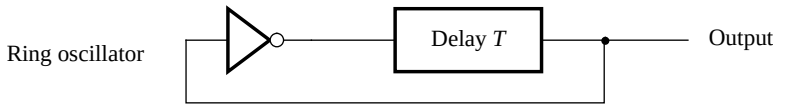
7.17. The following circuit uses a multiplexer arrangement similar to that in Figure 7.3.

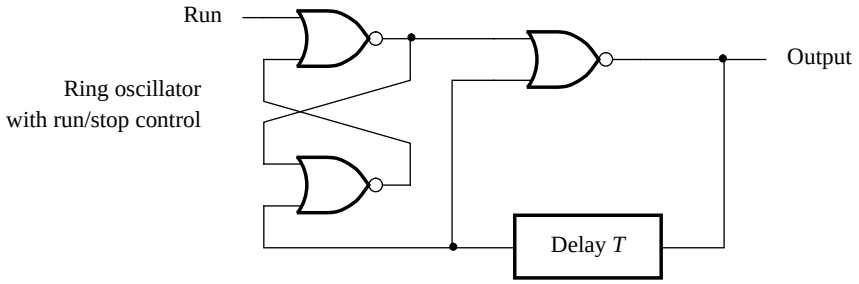


7.18. A possible arrangement is shown below. For clarity, we have assumed that MDR consists of two separate registers for input and output data. Multiplexers Mux-1 and Mux-2 select input B for even and input A for odd byte operations. Mux 3 selects input A for word operations and input B for byte operations. Input B provides either zero extension or sign extension of byte operands. For sign-extension it should be connected to the most-signi cant bit output of multiplexer Mux-2.

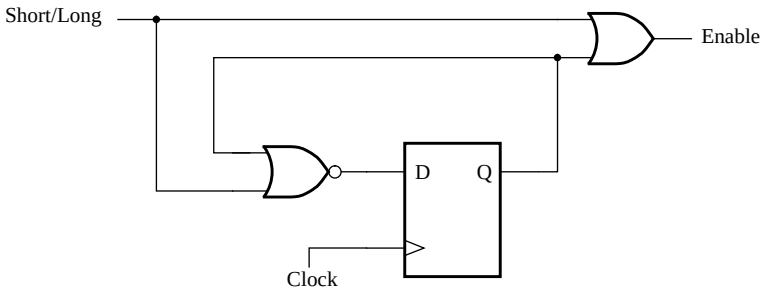


7.19. Use the delay element in a ring oscillator as shown below. The frequency of oscillation is 1/(2T). By adding the control circuit shown, the oscillator will run only while Run is equal to 1. When stopped, its output A is equal to 0. The oscillator will always generate complete output pulses. If Run goes to 0 while A is 1, the latch will not change state until B goes to 1 at the end of the pulse.

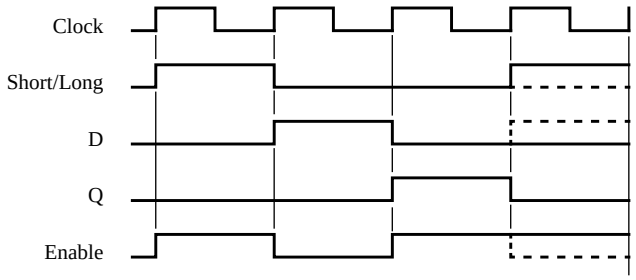




7.20. In the circuit below, Enable is equal to 1 whenever Short/$\overline{\text{Long}}$ is equal to 1, indicating a short pulse. When this line changes to 0, Enable changes to 0 for one clock cycle.



Short/Long	Q	D
0	0	1
0	1	0
1	0	0
1	1	0



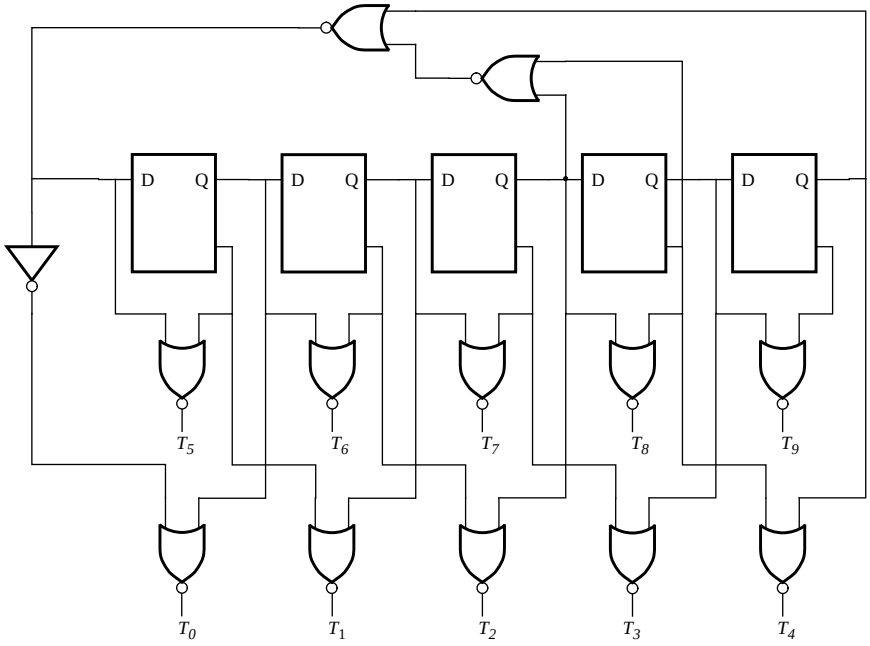
7.21. (*a*) Count sequence is: 0000 1000 1100 1110 1111 0111 0011 0001 0000

(*b*) A 5-bit Johnson counter is shown below, with the outputs Q_1 to Q_5 decoded to generate the signals T_1 to T_{10}. The feed back circuit has been modi ed to make the counter self-starting. It implements the function

$$D_1 = \overline{Q_5 + \overline{Q_3 + \overline{Q4}}}$$

This circuit detects states that have $Q_3Q_4Q_5$ = 010 and changes the feedback value from 1 to 0. Without this or a similar modi cation to the feedback circuit, the counter may be stuck in sequences other than the desired one above.

The advantage of a Johnson counter is that there are no glitches in decoding the count value to generate the timing signals.



7.22. We will generate a signal called Store to recirculate data when no external action is required.

$$\begin{aligned}
\text{Store} &= \overline{(\text{ARS} + \text{LSR} + \text{SL} + \text{LLD})} \\
D_{15} &= \text{ASR} \cdot Q_{15} + \text{SL} \cdot Q_{14} + \text{ROR} \cdot \text{Carry} + \text{LD} \cdot D_{15} + \text{Store} \cdot Q_{15} \\
D_1 &= (\text{ASR} + \text{LSR} + \text{ROR}) \cdot Q_2 + \text{SL} \cdot Q_0 + \text{LD} \cdot D_1 + \text{Store} \cdot Q_1 \\
D_0 &= (\text{ASR} + \text{LSR} + \text{ROR}) \cdot Q_1 + \text{LD} \cdot D_0 + + \text{Store} \cdot Q_0
\end{aligned}$$

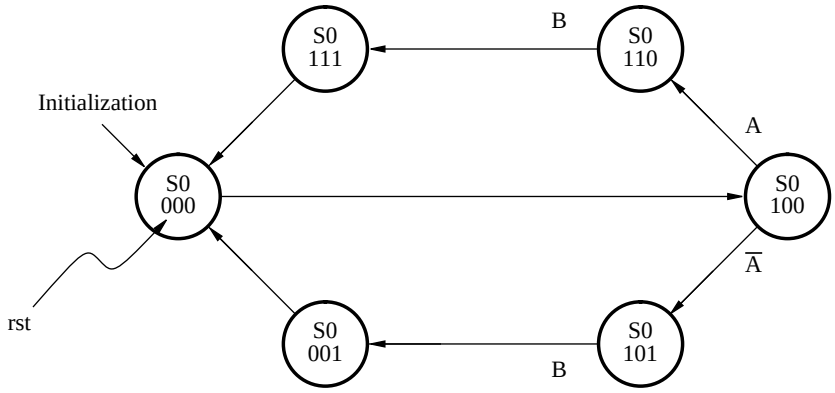
7.23. A state diagram for the required controller is given below. This is a Moore machine. The output values are given inside each state as they are functions of the state only.

Since there are 6 independent states, a minimum of three ip- ops r, s, and t are required for the implementation. A possible state assignment is shown in the diagram. It has been chosen to simplify the generation of the outputs X, Y, and Z, which are given by

$$X = r + s + t \qquad Y = s \qquad Z = t$$

Using D ip- ops for implementation of the controller, the required inputs to the ip- ops may be generated as follows

$$\begin{aligned} D(r) &= s\,\overline{t}\,B + \overline{s}\,\overline{t} \\ D(s) &= \overline{s}\,\overline{t}\,A + s\,\overline{t}\,B \\ D(t) &= s\,\overline{t}\,B + \overline{s}\,\overline{t}\,\overline{A} + \overline{s}\,t\,B \end{aligned}$$



7.24. Microroutine:

Address (Octal)	**Microinstruction**
000-002	Same as in Figure 7.21
300	μBranch $\{\mu\mathrm{PC} \leftarrow 161$
161	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
162	Z_{out}, PC_{in}, WMFC
163	MDR_{out}, Y_{in}
164	Rsrc_{out}, SelectY, Add, Z_{in}
165	Z_{out}, MAR_{in}, Read
166	μBranch $\{\mu\mathrm{PC} \leftarrow 170$; $\mu\mathrm{PC}_0 \leftarrow \overline{[\mathrm{IR}_8]}\}$, WMFC
170-173	Same as in Figure 7.21

7.25. Conditional branch

Address (Octal)	**Microinstruction**
000-002	Same as in Figure 7.21
003	μBranch $\{\mu\mathrm{PC} \leftarrow 300$
300	if Z+(N$\oplus$V = 1 then μBranch $\{\mu\mathrm{PC} \leftarrow 304\}$
301	PC_{out}, Y_{in}
302	$\mathrm{Address}_{out}$, SelectY, Add, Z_{in}
303	Z_{out}, PC_{in}, End

7.26. Assume microroutine starts at 300 for all three instructions. (Altenatively, the instruction decoder may branch to 302 directly in the case of an unconditional branch instruction.)

Address (Octal)	**Microinstruction**
000-002	Same as in Figure 7.21
003	μBranch $\{\mu\mathrm{PC} \leftarrow 300\}$
300	if Z+(N$\oplus$V = 1) then μBranch $\{\mu\mathrm{PC} \leftarrow 000\}$
301	if (N = 1) then μBranch $\{\mu\mathrm{PC} \leftarrow 000\}$
302	PC_{out}, Y_{in}
303	Offset- eld-of-IR $_{out}$, SelectY, Add, Z_{in}
304	Z_{out}, PC_{in}, End

7.27. The answer to problem 3.26 holds in this case as well, with the restriction that one of the operand locations (either source or destination) must be a data register.

Address (Octal)	**Microinstruction**
000-002	Same as in Figure 7.21
003	μBranch $\{\mu\mathrm{PC} \leftarrow 010\}$
010	if ($\mathrm{IR}_{10-8} = 000$) then μBranch $\{\mu\mathrm{PC} \leftarrow 101\}$
011	if ($\mathrm{IR}_{10-8} = 001$) then μBranch $\{\mu\mathrm{PC} \leftarrow 111\}$
012	if ($\mathrm{IR}_{10-9} = 01$) then μBranch $\{\mu\mathrm{PC} \leftarrow 121\}$
013	if ($\mathrm{IR}_{10-9} = 10$) then μBranch $\{\mu\mathrm{PC} \leftarrow 141\}$
014	μBranch $\{\mu\mathrm{PC} \leftarrow 161\}$
121	Rsrc_{out}, MAR_{in}, Read, Select4, Add, $\mathrm{Z}in$
122	Z_{out}, Rsrc_{in}
123	if ($\mathrm{IR}_8 = 1$) then μBranch $\{\mu\mathrm{PC} \leftarrow 171\}$
124	μBranch $\{\mu\mathrm{PC} \leftarrow 170\}$
170-173	Same as in Figure 7.21

7.28. There is no change for the ve address modes in Figure 7.20. Absolute and Immediate modes require a separate path. However, some sharing may be possible among absolute, immediate, and indexed, as all three modes read the word following the instruction. Also, Full Indexed mode needs to be implemented by adding the contents of the second register to generate the effective address. After each memory access, the program counter should be updated by 2, rather than 4, in the case of the 16-bit processor.

7.29. The same general structure can be used. Since the dst operand can be speci ed in any of the ve addressing modes as the src operand, it is necessary to replicate the microinstructions that determine the effective address of an operand. At microinstruction 172, the source operand should placed in a temporary register and another tree of microinstructions should be entered to fetch the destination operand.

7.30. (*a*) A possible address assignment is as follows.

Address	Microinstruction
0000	A
0001	B
0010	if (b_6b_5) = 00) then μBranch 0111
0011	if (b_6b_5) = 01) then μBranch 1010
0100	if (b_6b_5) = 10) then μBranch 1100
0101	I
0110	μBranch 1111
0111	C
1000	D
1001	μBranch 1111
1010	E
1011	μBranch 1111
1100	F
1101	G
1110	H
1111	J

(*b*) Assume that bits b_{6-5} of IR are ORed into bit μPC_{3-2}

Address	Microinstruction
0000	A
0001	B; $\mu PC_{3-2} \leftarrow b_{6-5}$
0010	C
0011	D
0100	μBranch 1111
0101	E
0110	μBranch 1111
0111	F
1011	G
1100	H
1101	μBranch 1111
1110	I
1111	J

(*c*)

Address	**Microinstruction**	
	Next address	Function
0000	0001	A
0001	0010	B; $\mu\text{PC}_{3-2} \leftarrow b_{6-5}$
0010	0011	C
0011	1111	D
0110	1111	E
1010	1011	F
1011	1100	G
1100	1111	H
1110	1111	I
1111	–	J

7.31. Put the Y_{in} control signal as the fourth signal in F5, to reduce F3 by one bit. Combine elds F6, F7, and F8 into a single 2-bit eld that represents:

00: Select4
01: SelectY
10: WMFC
11: End

Combining signals means that they cannot be issued in the same microinstruction.

7.32. To reduce the number of bits, we should use larger elds that specify more signals. This, inevitably, leads to fewer choices in what signals can be activated at the same time. The choice as to which signals can be combined should take into account what signals are likely to be needed in a given step.

One way to provide e xibility is to de ne control signals that perform multiple functions. For example, whenever MAR is loaded, it is likely that a read command should be issued. We can use two signals: MAR_{in} and $\text{MAR}_{in}\cdot$ Read. We activate the second one when a read command is to be issued. Similarly, Z_{in} is always accompanied by either Select Y or Select4. Hence, instead of these three signals, we can use $\text{Z}_{in} \cdot \text{Select4}$ and $\text{Z}_{in} \cdot \text{SelectY}$.

A possible 12-bit encoding uses three 4-bit elds FA, FB, and FC, which combine signals from Figure 7.19 as follows:

FA: F1 plus, $\text{Z}_{out}\cdot$ End, $\text{Z}_{out}\cdot$ WMFC. (11 signals)

FB: F2, F3, Instead of Z_{in}, MAR_{in}, and MDR_{in} use $\text{Z}_{in}\cdot\text{Select4}$, $\text{Z}_{in}\cdot\text{SelectY}$, MAR_{in}, $\text{MAR}_{in} \cdot \text{Read}$, and $\text{MDR}_{in}\cdot$ Write. (13 signals)

FC: F4 (16 signals)

With these choices, step 5 in Figure 7.6 must be split into two steps, leading to an 8-step sequence. Figure 7.7 remains unchanged.

7.33. Figure 7.8 contains two buses, A and B, one connected to each of the two inputs of the ALU. Therefore, two elds are needed instead of F1; one eld to provide gating of registers onto bus A, and another onto bus B.

7.34. Horizontal microinstructions are longer. Hence, they require a larger microprogram memory. A vertical organization requires more encoding and decoding of signals, hence longer delays, and leads to longer microprograms and slower operation. With the high-density of today's integrated circuits, the vertical organization is no longer justi ed.

7.35. The main advantage of hardwired control is fast operation. The disadvantages include: higher cost, in e xibility when changes or additions are to be made, and longer time required to design and implement such units.

Microprogrammed control is characterized by low cost and high e xibility. Lower speed of operation becomes a problem in high-performance computers.