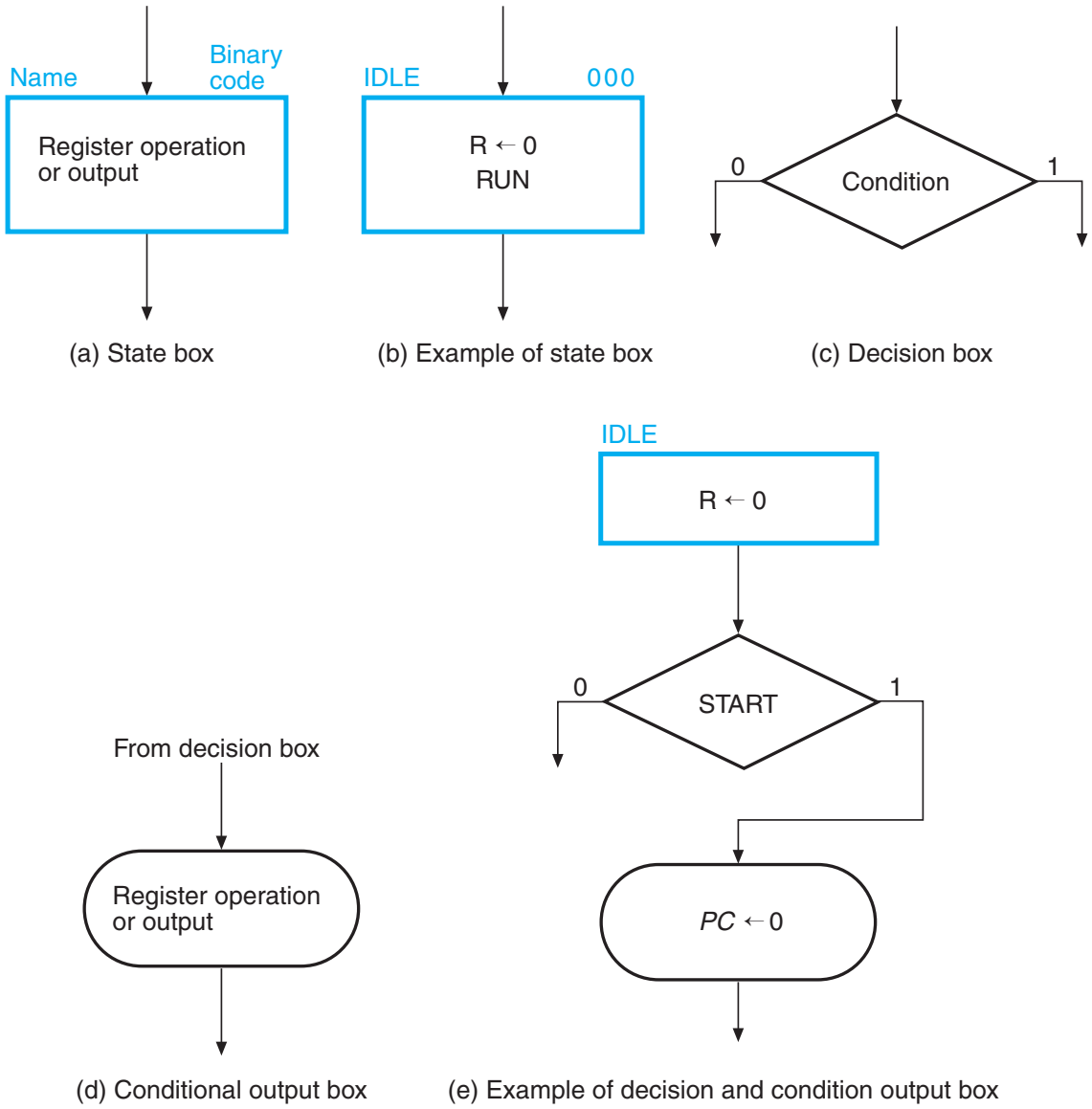

(a) State box (b) Example of state box (c) Decision box

(d) Conditional output box (e) Example of decision and condition output box

Fig. 8-1 ASM Chart Elements

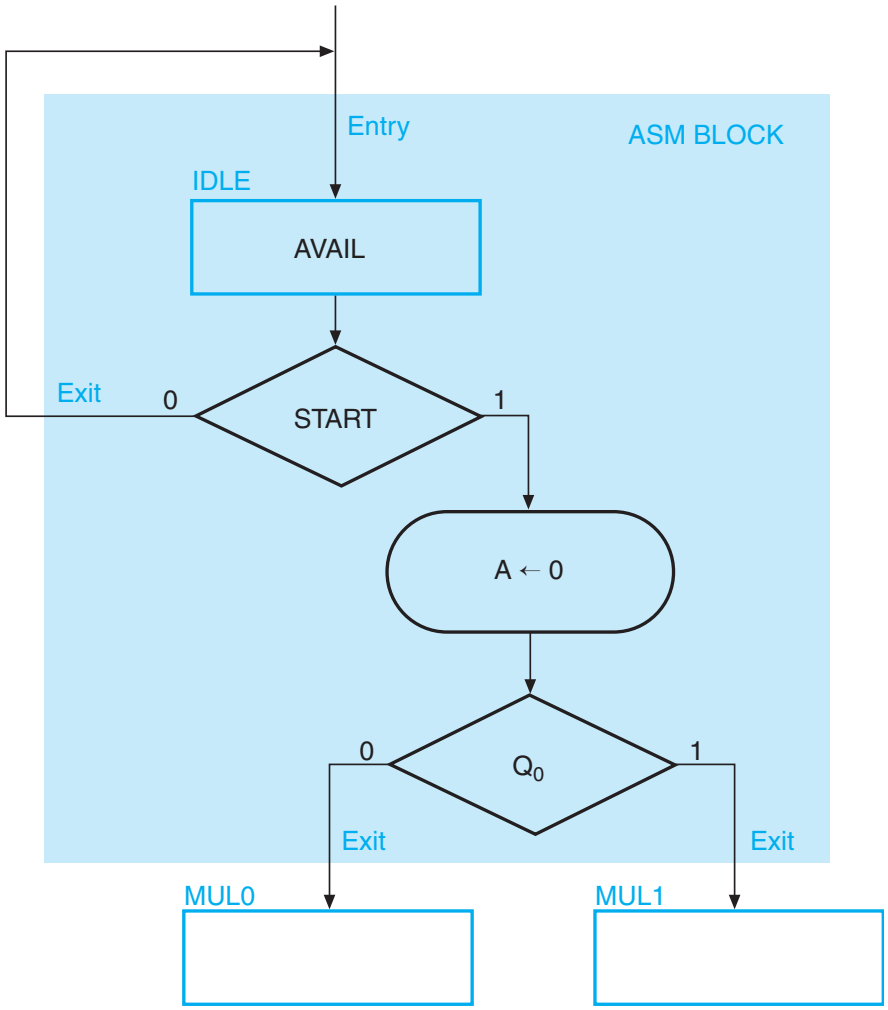

Fig. 8-2 ASM Block

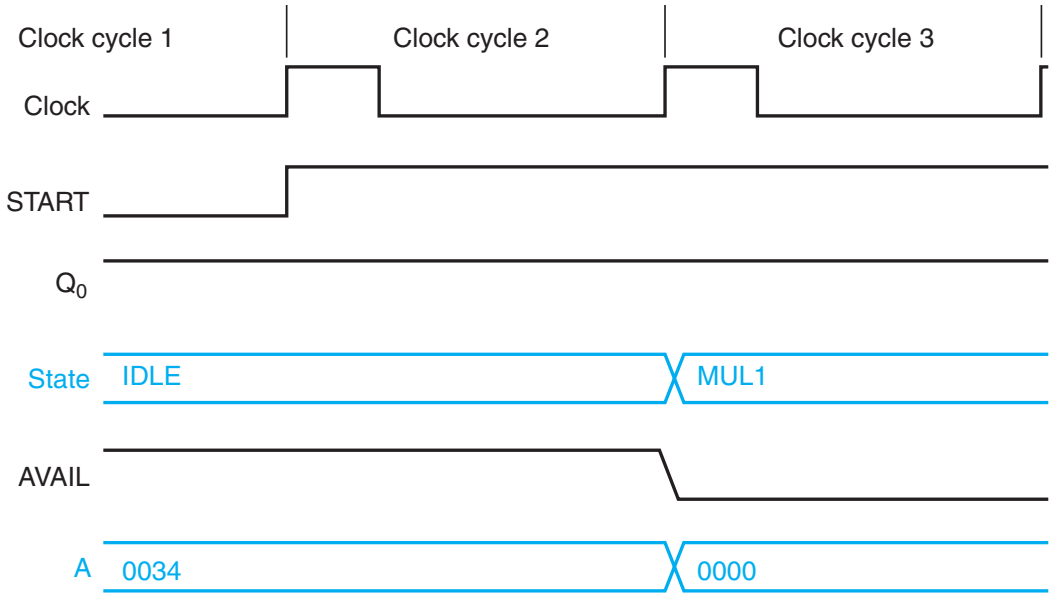

Fig. 8-3 ASM Timing Behavior

23	10111	Multiplicand
19	10011	Multiplier
	10111	
	10111	
	00000	
	00000	
	10111	
437	110110101	Product

Fig. 8-4 Hand Multiplication Example

23	10111	Multiplicand
19	10011	Multiplier
	00000	Initial partial product
	10111	Add multiplicand, since multiplier bit is 1
	10111	Partial product after add and before shift
	010111	Partial product after shift
	10111	Add multiplicand, since multiplier bit is 1
	1000101	Partial product after add and before shift[a]
	1000101	Partial product after shift
	01000101	Partial product after shift
	001000101	Partial product after shift
	10111	Add multiplicand, since multiplier bit is 1
	110110101	Partial product after add and before shift
437	0110110101	Product after final shift

a. Note that overflow temporarily occurred.

Fig. 8-5 Hardware Multiplication Example

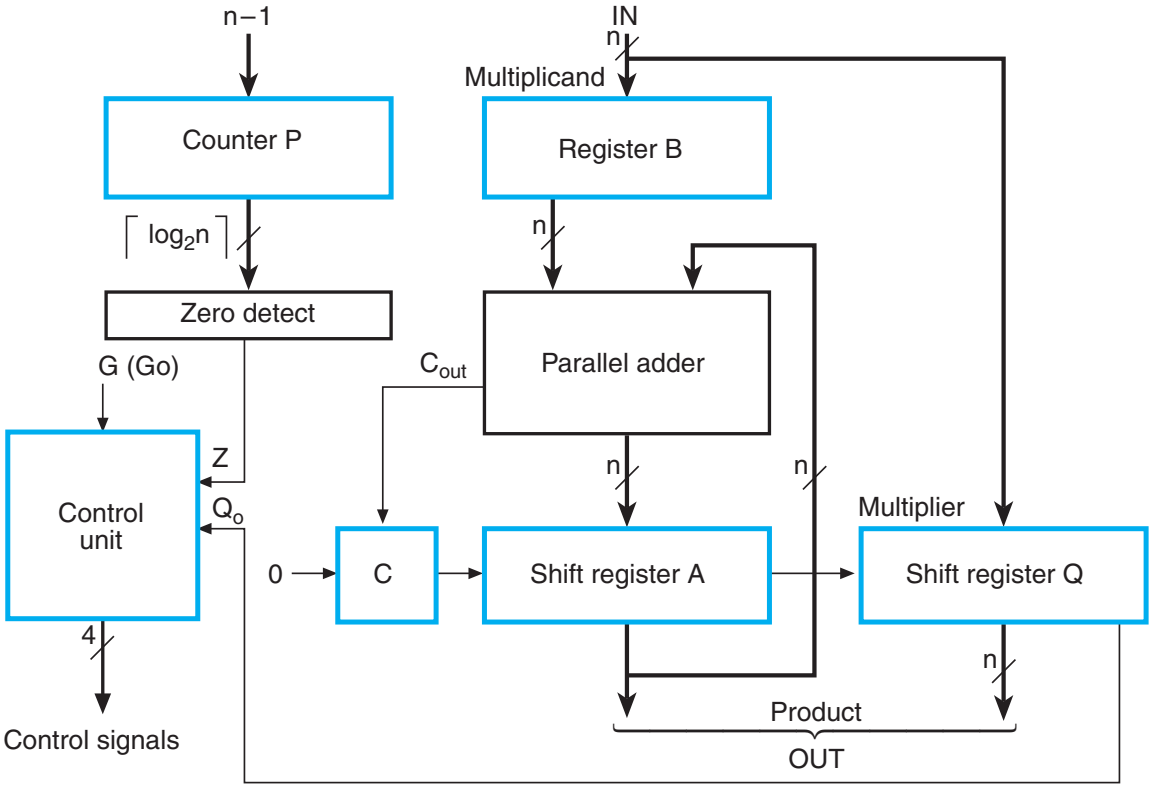

Fig. 8-6 Block Diagram for Binary Multiplier

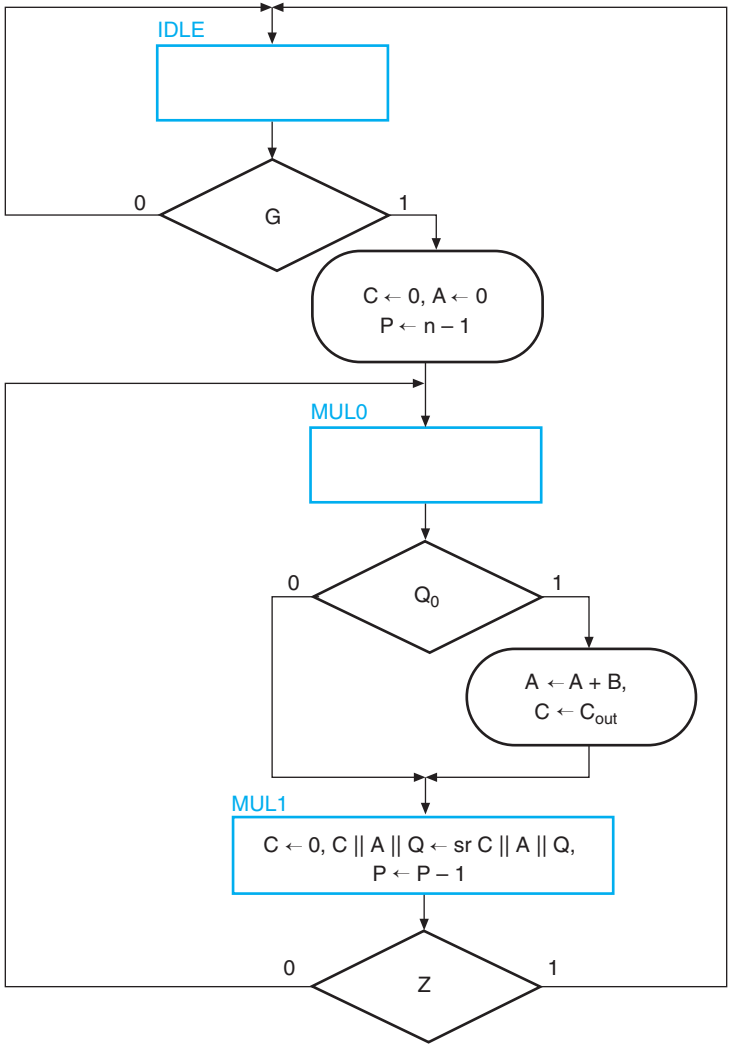

Fig. 8-7 ASM Chart for Binary Multiplier

TABLE 8-1
Control Signals for Binary Multiplier

Block Diagram Module	Microoperation	Control Signal Name	Control Expression
Register A:	$A \leftarrow 0$	Initialize	$\text{IDLE} \cdot G$
	$A \leftarrow A + B$	Load	$\text{MUL0} \cdot Q_0$
	$C \| A \| Q \leftarrow \text{sr}\ C \| A \| Q$	Shift_dec	MUL1
Register B:	$B \leftarrow IN$	Load_B	LOADB
Flip-Flop C:	$C \leftarrow 0$	Clear_C	$\text{IDLE} \cdot G + \text{MUL1}$
	$C \leftarrow C_{\text{out}}$	Load	—
Register Q:	$Q \leftarrow IN$	Load_Q	LOADQ
	$C \| A \| Q \leftarrow \text{sr}\ C \| A \| Q$	Shift_dec	—
Counter P:	$P \leftarrow n - 1$	Initialize	—
	$P \leftarrow P - 1$	Shift_dec	—

Table 8-1 Control Signals for Binary Multiplier

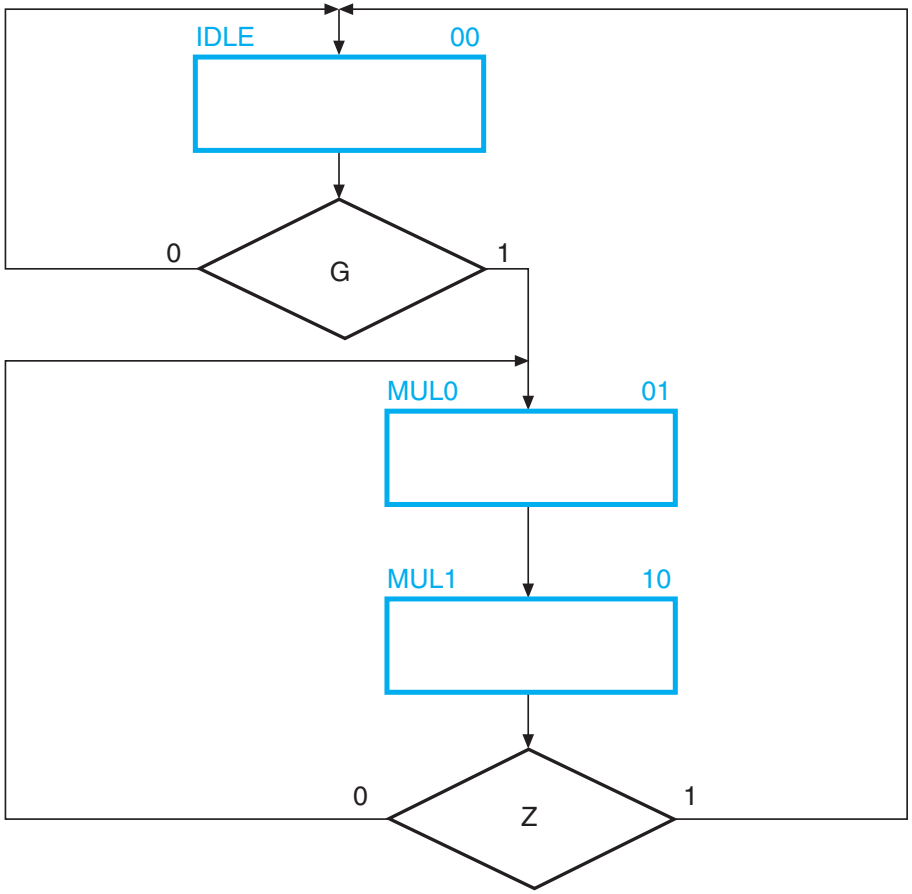

Fig. 8-8 Sequencing Part of ASM Chart for the Binary Multiplier

TABLE 8-2
State Table for Sequence Register and Decoder Part of Multiplier Control Unit

Present state			Inputs		Next state		Decoder Outputs		
Name	M_1	M_0	G	Z	M_1	M_0	IDLE	MUL0	MUL1
IDLE	0	0	0	×	0	0	1	0	0
	0	0	1	×	0	1	1	0	0
MUL0	0	1	×	×	1	0	0	1	0
MUL1	1	0	×	0	0	1	0	0	1
	1	0	×	1	0	0	0	0	1
—	1	1	×	×	×	×	×	×	×

Table 8-2 State Table for Sequence Register and Decoder Part of Multiplier Control Unit

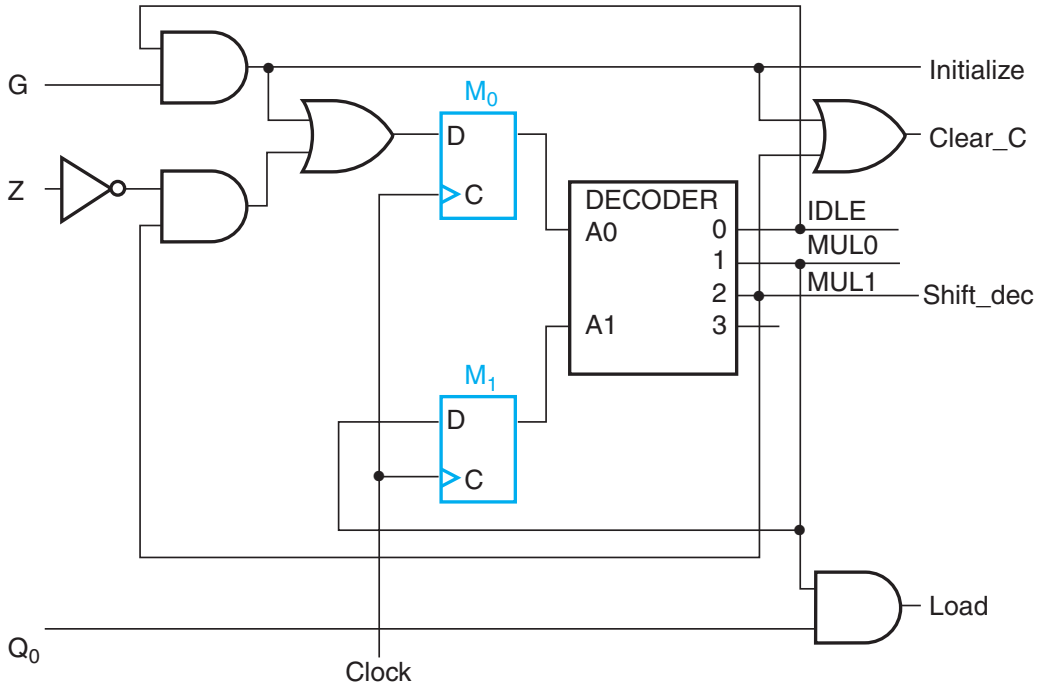

Fig. 8-9 Control Unit for Binary Multiplier Using a Sequence Register and a Decoder

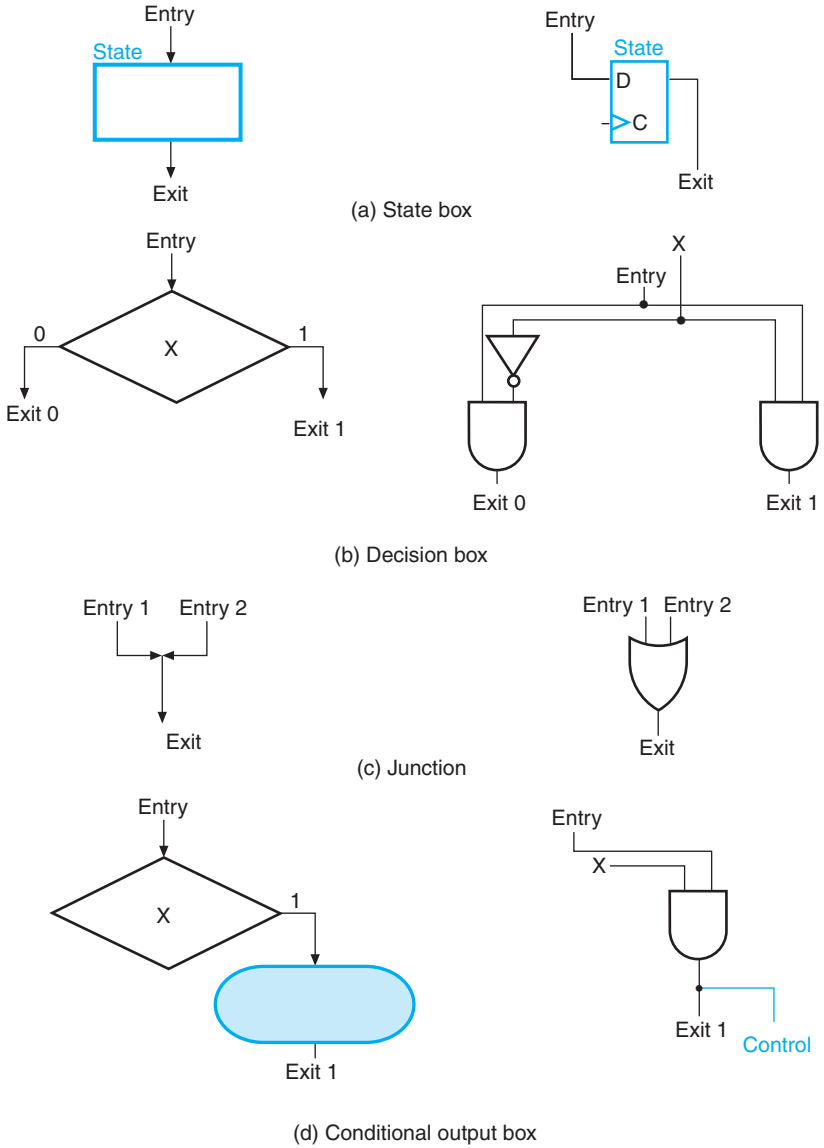

Fig. 8-10 Transformation Rules for Control Unit with One Flip-Flop per State

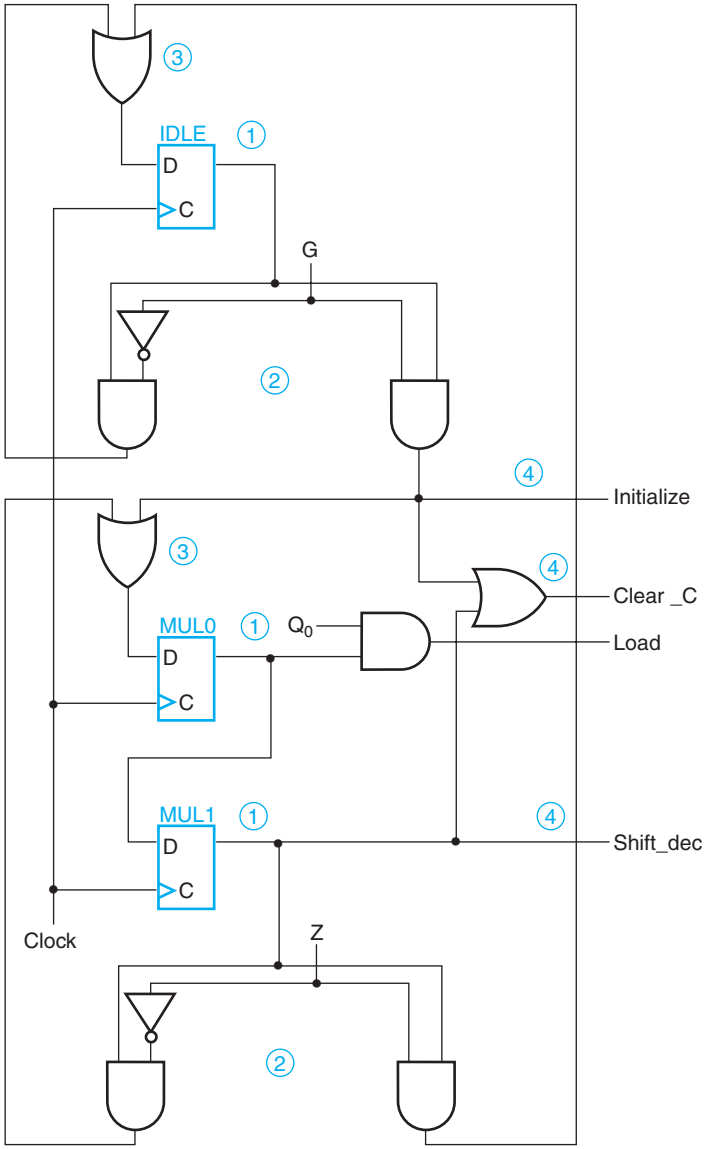

Fig. 8-11 Control Unit with One Flip-Flop per State for the Binary Multiplier

```
-- Binary Multiplier with n = 4: VHDL Description
-- See Figures 8-6 and 8-7 for block diagram and ASM Chart
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity binary_multiplier is
  port(CLK, RESET, G, LOADB, LOADQ: in std_logic;
    MULT_IN: in std_logic_vector(3 downto 0);
    MULT_OUT: out std_logic_vector(7 downto 0));
end binary_multiplier;

architecture behavior_4 of binary_multiplier is
  type state_type is (IDLE, MUL0, MUL1);
  signal state, next_state : state_type;
  signal A, B, Q: std_logic_vector(3 downto 0);
  signal P: std_logic_vector(1 downto 0);
  signal C, Z: std_logic;
begin
  Z <= P(1) NOR P(0);
  MULT_OUT <= A & Q;

  state_register: process (CLK, RESET)
  begin
    if (RESET = '1') then
      state <= IDLE;
    elsif (CLK'event and CLK = '1') then
      state <= next_state;
    end if;
  end process;

  next_state_func: process (G, Z, state)
  begin
    case state is
      when IDLE =>
        if G = '1' then
          next_state <= MUL0;
        else
          next_state <= IDLE;
        end if;
      when MUL0 =>
        next_state <= MUL1;
      when MUL1 =>
        if Z = '1' then
          next_state <= IDLE;
        else
          next_state <= MUL0;
        end if;
```

Fig. 8-12 VHDL Description of a Binary Multiplier

```
    end case;
  end process;

  datapath_func: process (CLK)
  variable CA: std_logic_vector(4 downto 0);
  begin
    if (CLK'event and CLK = '1') then
      if LOADB = '1' then
        B <= MULT_IN;
      end if;
      if LOADQ = '1' then
        Q <= MULT_IN;
      end if;
      case state is
        when IDLE =>
          if G = '1' then
            C <= '0';
            A <= "0000";
            P <= "11";
          end if;
        when MUL0 =>
          if Q(0) = '1' then
            CA := ('0' & A) + ('0' & B);
          else
            CA := C & A;
          end if;
            C <= CA(4);
            A <= CA(3 downto 0);
        when MUL1 =>
          C <= '0';
          A <= C & A(3 downto 1);
          Q <= A(0) & Q(3 downto 1);
          P <= P - "01";
      end case;
    end if;
  end process;
end behavior_4;
```

Fig. 8-13 VHDL Description of a Binary Multiplier (Continued)

```
// Binary Multiplier with n = 4: Verilog Description
// See Figures 8-6 and 8-7 for block diagram and ASM Chart

module binary_multiplier_v (CLK, RESET, G, LOADB, LOADQ,
       MULT_IN, MULT_OUT);
input CLK, RESET, G, LOADB, LOADQ;
input [3:0] MULT_IN;
output [7:0] MULT_OUT;
reg [1:0] state, next_state, P;
parameter IDLE = 2'b00, MUL0 = 2'b01, MUL1 = 2'b10;
reg [3:0] A, B, Q;
reg C;
wire Z;

assign Z = ~| P;
assign MULT_OUT = {A,Q};

//state register
always@(posedge CLK or posedge RESET)
begin
  if (RESET == 1)
    state <= IDLE;
  else
  state <= next_state;
end

//next state function
always@(G or Z or state)
begin
  case (state)
    IDLE:
      if (G == 1)
        next_state <= MUL0;
      else
        next_state <= IDLE;
    MUL0:
      next_state <= MUL1;
    MUL1:
      if (Z == 1)
        next_state <= IDLE;
      else
        next_state <= MUL0;
  endcase
end

//datapath function
always@(posedge CLK)
```

Fig. 8-14 Verilog Description of a Binary Multiplier

```
begin
  if (LOADB == 1)
    B <= MULT_IN;
  if (LOADQ == 1)
    Q <= MULT_IN;
  case (state)
    IDLE:
      if (G == 1)
        begin
          C <= 0;
          A <= 4'b0000;
          P <= 2'b11;
        end
    MUL0:
      if (Q[0] == 1)
        {C, A} = A + B;
    MUL1:
      begin
        C <= 1'b0;
        A <= {C, A[3:1]};
        Q <= {A[0], Q[3:1]};
        P <= P - 2'b01;
      end
  endcase
end
endmodule
```

Fig. 8-15 Verilog Description of a Binary Multiplier (Continued)

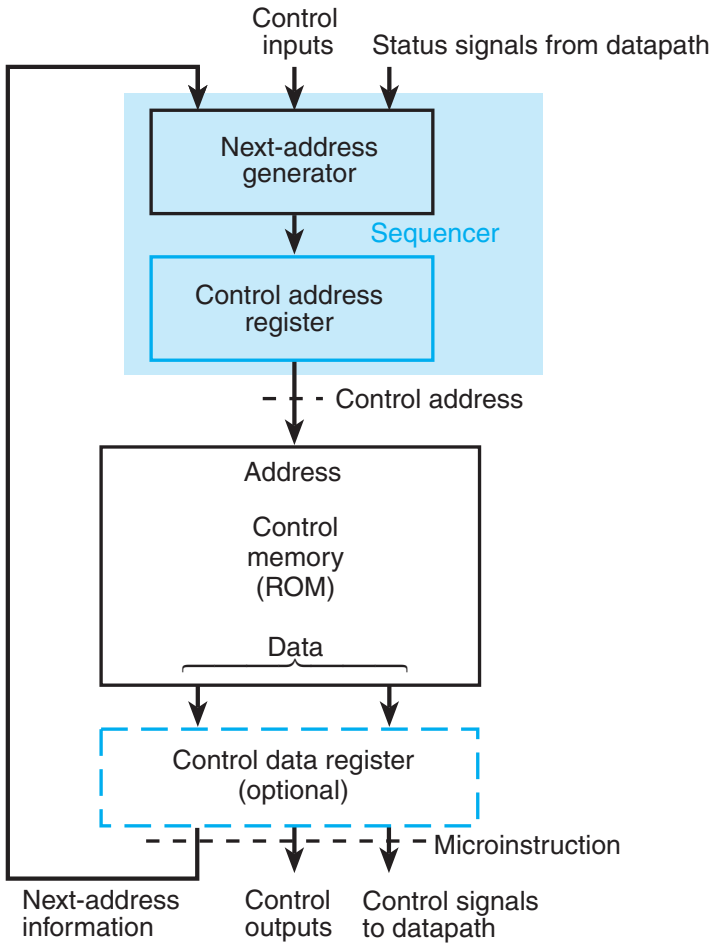

Fig. 8-16 Microprogrammed Control Unit Organization

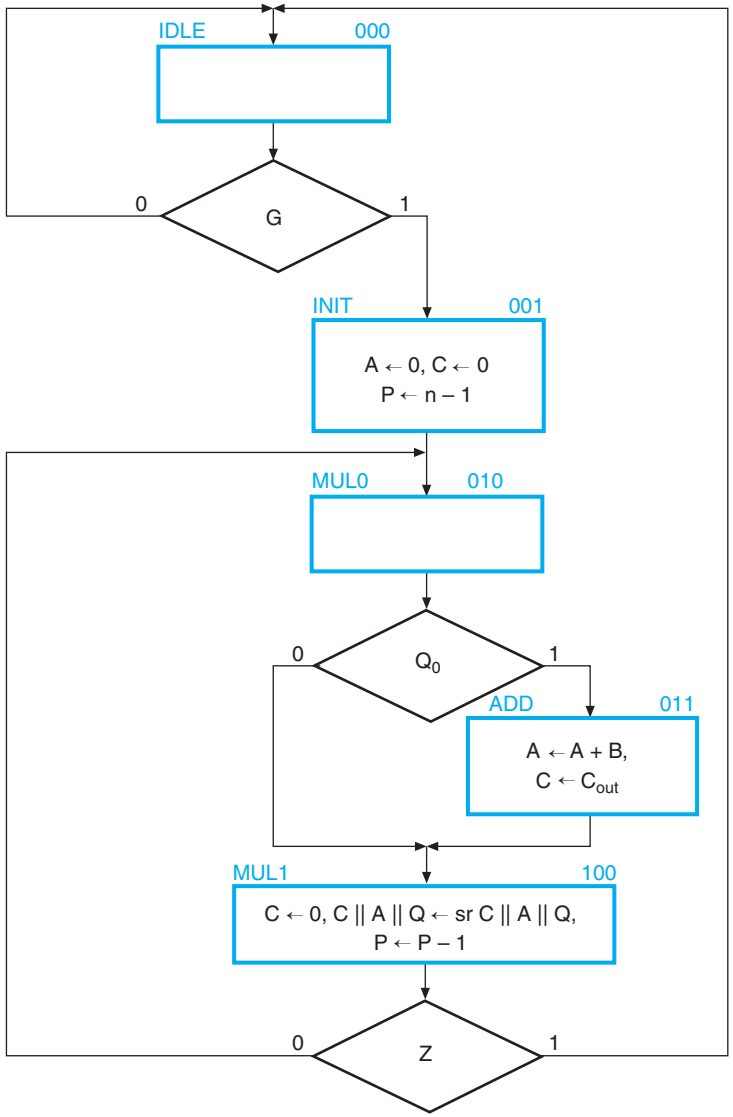

Fig. 8-17 ASM Chart for Microprogrammed Binary Multiplier Control Unit

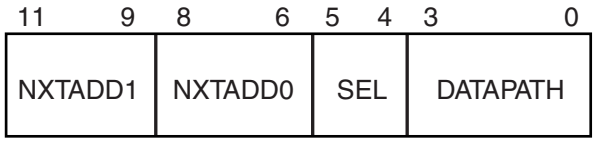

Fig. 8-18 Microinstruction Control Word Format

TABLE 8-3
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	$A \leftarrow A + B, C \leftarrow C_{out}$	ADD	1	LD
Clear_C	$C \leftarrow 0$	INIT, MUL1	2	CC
Shift_dec	$C \| A \| Q \leftarrow \text{sr } C \| A \| Q, P \leftarrow P-1$	MUL1	3	SD

Table 8-3 Control Signals for Microprogrammed Multiplier Control

TABLE 8-4
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: $CAR \leftarrow NXTADD1$

Table 8-4 SEL Field Definition for Binary Multiplier Control Sequencing

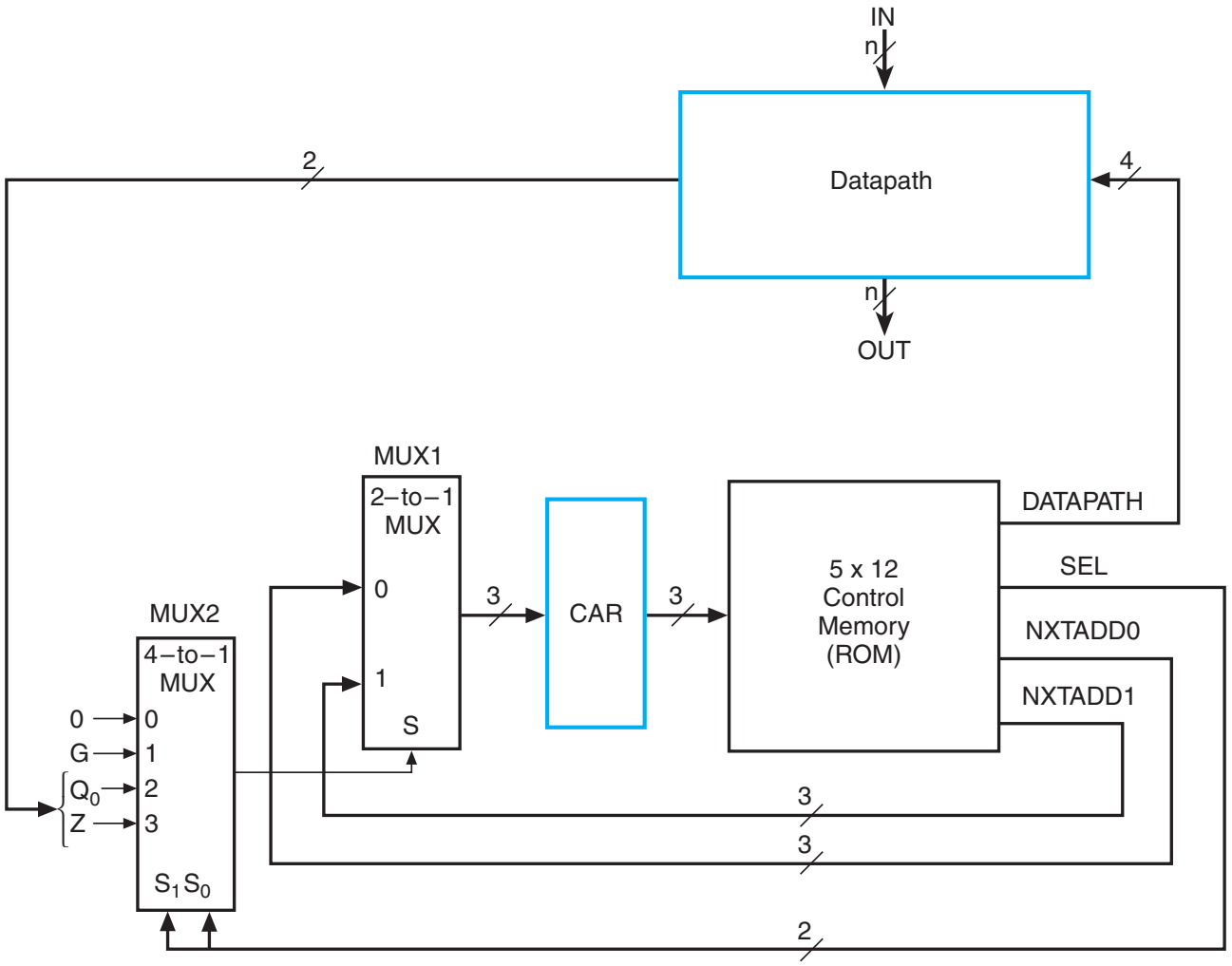

Fig. 8-19 Microprogrammed Control Unit for Multiplier

TABLE 8-5
Register Transfer Description of Binary Multiplier Microprogram

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

Table 8-5 Register Transfer Description of Binary Multiplier Microprogram

TABLE 8-6
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

Table 8-6 Symbolic Microprogram and Binary Microprogram for Multiplier

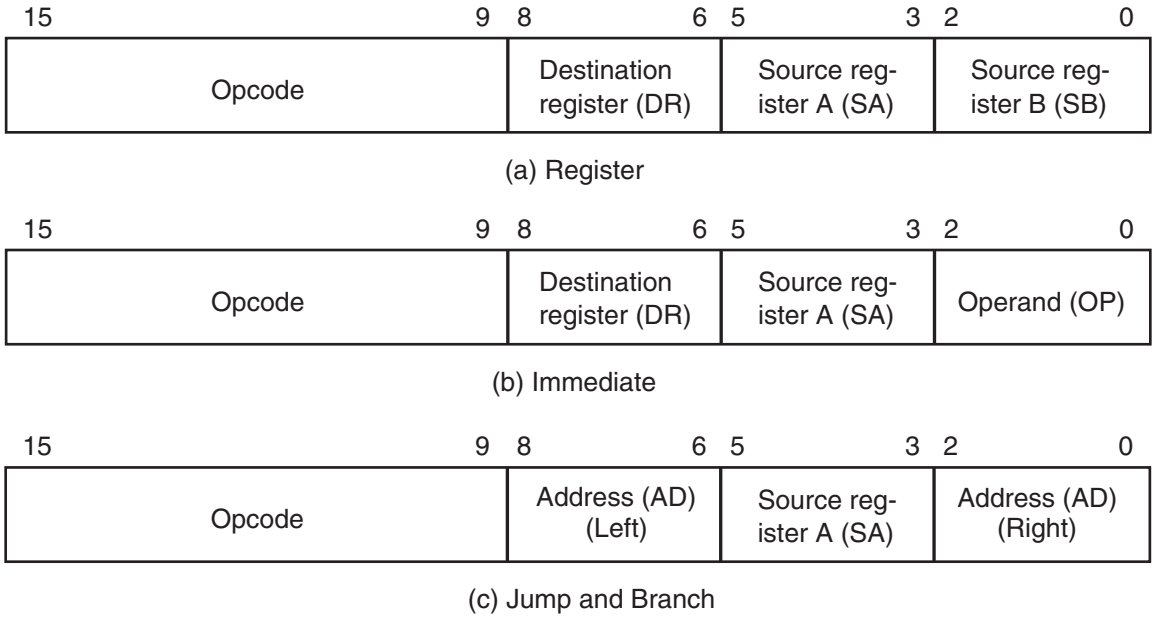

Fig. 8-20 Three Instruction Formats

Decimal address	Memory contents	Decimal opcode	Other specified fields	Operation
25	0000101 001 010 011	5 (Subtract)	DR:1, SA:2 SB:3	R1 ← R2 − R3
35	0100000 000 100 101	32 (Store)	SA:4 SB:5	M [R4] ← R5
45	1000010 010 111 011	66 (Add Immediate)	DR:2 SA:7 OP:3	R2 ← R7 + 3
55	1100011 101 110 100	96 (Branch on zero)	AD: 44 SA:6	If R6 = 0, PC ← PC − 20
70	0000000 011 000 000	Data = 192. After execution of instruction in 35, Data = 80.		

Fig. 8-21 Memory Representation of Instructions and Data

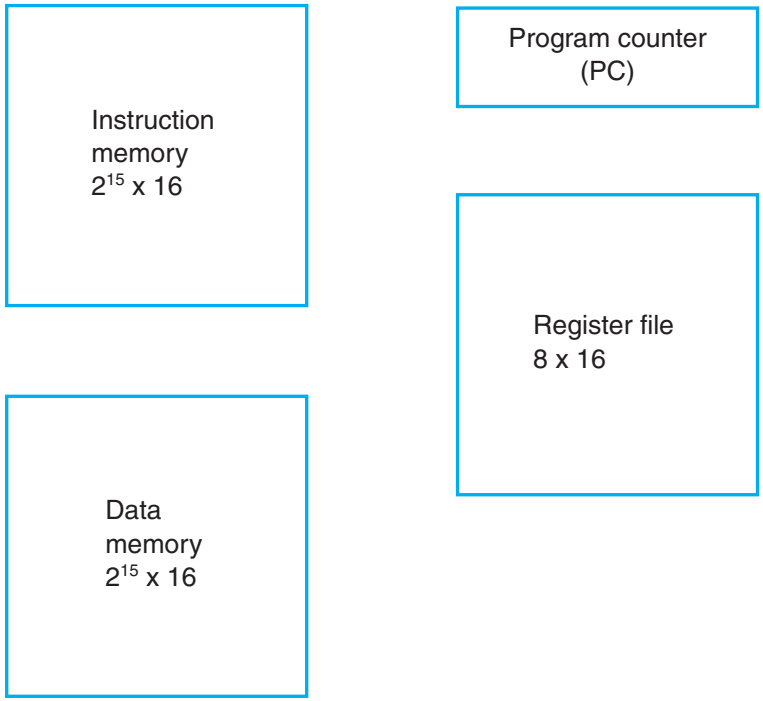

Fig. 8-22 Storage Resource Diagram for a Simple Computer

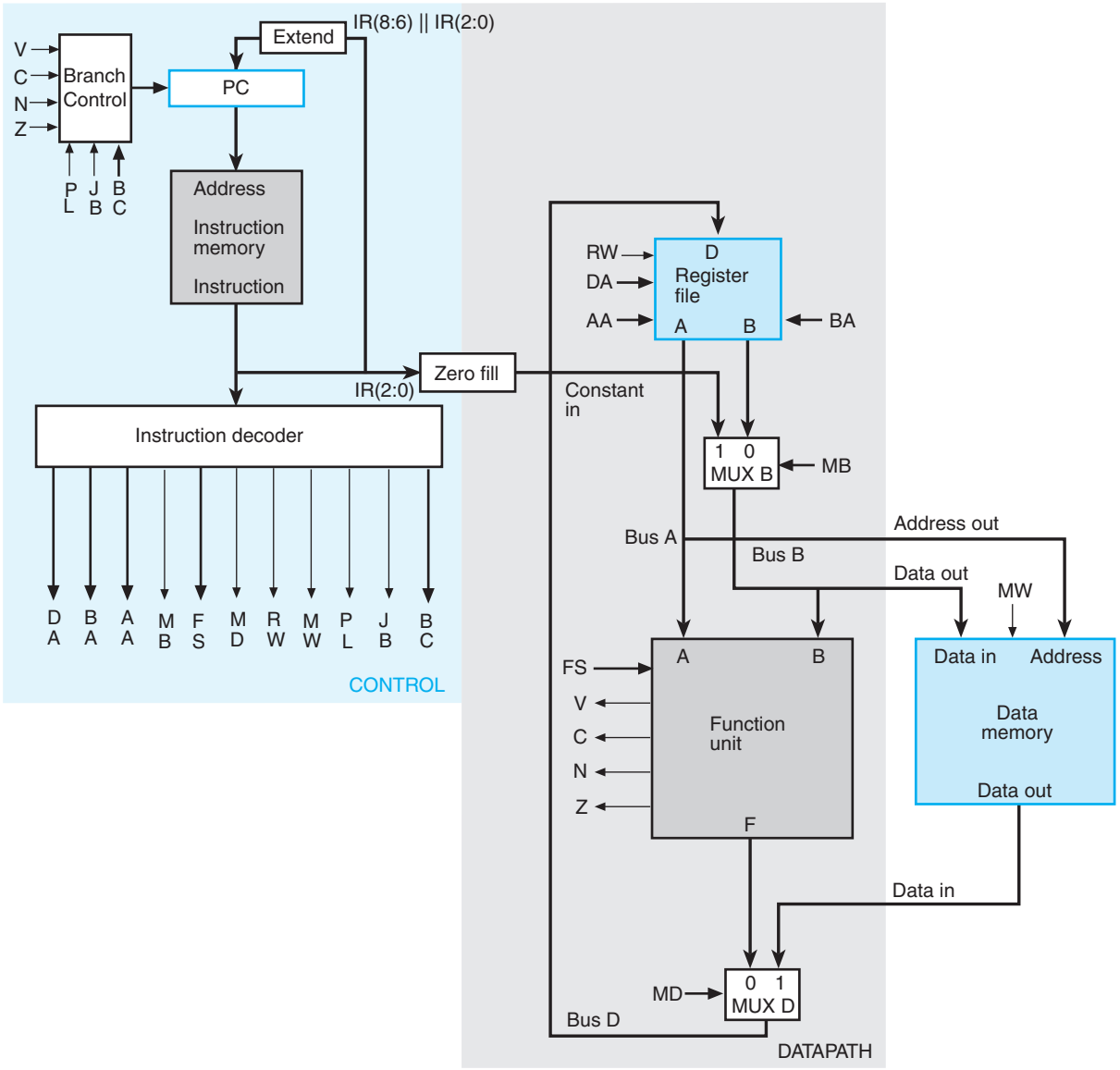

Fig. 8-23 Block Diagram for a Single-Cycle Computer

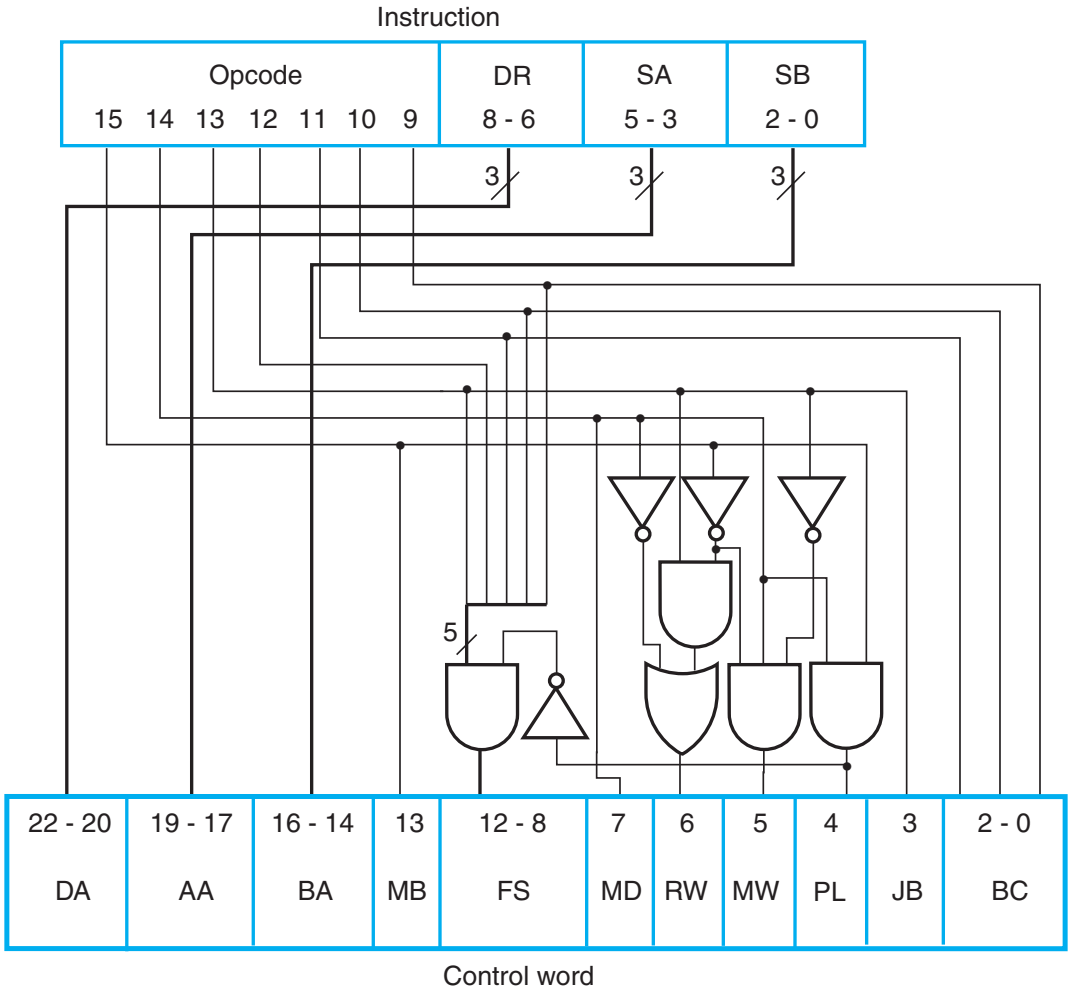

Fig. 8-24 Diagram of Instruction Decoder

TABLE 8-7
Truth Table for Instruction Decoder Logic

	Instruction Bits			Control Word Bits					
Instruction Function Type	**Bit 15**	**Bit 14**	**Bit 13**	**MB**	**MD**	**RW**	**MW**	**PL**	**JB**
ALU function using registers	0	0	0	0	0	1	0	0	X
Shifter function using registers	0	0	1	0	0	1	0	0	X
Memory write using register data	0	1	0	0	X	0	1	0	X
Memory read using register data	0	1	1	0	1	1	0	0	X
ALU operation using a constant	1	0	0	1	0	1	0	0	X
Shifter function using a constant	1	0	1	1	0	1	0	0	X
Conditional Branch	1	1	0	X	X	0	0	1	0
Unconditional Jump	1	1	1	X	X	0	0	1	1

Table 8-7 Truth Table for Instruction Decoder Logic

TABLE 8-8
Six Instructions for the Single-Cycle Computer

Operation code	Symbolic name	Format	Description	Function	MB	MD	RW	MW	PL	JB
1000010	ADI	Immediate	Add immediate operand	$R[\text{DR}] \leftarrow R[\text{SA}] + \text{zf } I(2{:}0)$	1	0	1	0	0	0
0110000	LD	Register	Load memory content into register	$R[\text{DR}] \leftarrow M[R[\text{SA}]]$	0	1	1	0	0	1
0100000	ST	Register	Store register content in memory	$M[R[\text{SA}]] \leftarrow R[\text{SB}]$	0	1	0	1	0	0
0011000	SL	Register	Shift left	$R[\text{DR}] \leftarrow \text{sl}R[\text{SB}]$	0	0	1	0	0	1
0001110	NOT	Register	Complement register	$R[\text{DR}] \leftarrow \overline{R[\text{SA}]}$	0	0	1	0	0	0
1100000	BRZ	Jump/Branch	If R[SA] = 0, branch to PC + se AD	If R[SA] = $0, PC \leftarrow PC + \text{se}AD$, If R[SA] $\neq 0, PC \leftarrow PC + 1$	1	0	0	0	1	0

Table 8-8 Six Instructions for the Single-Cycle Computer

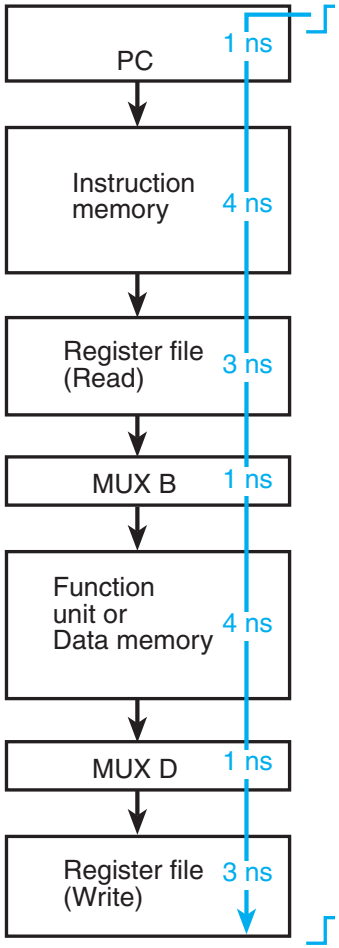

Fig. 8-25 Worst Case Delay Path in Single-Cycle Computer

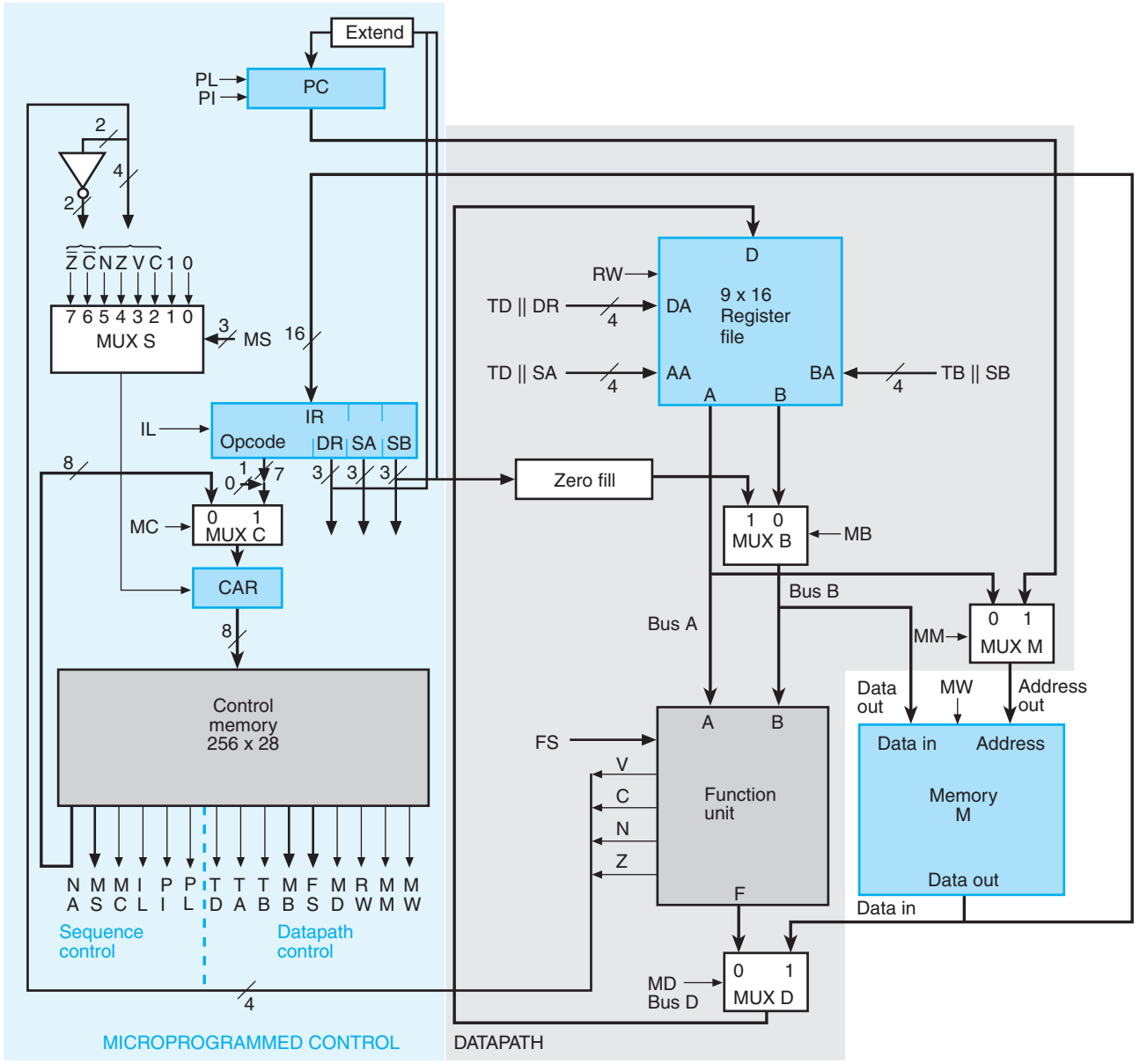

Fig. 8-26 Multiple-Cycle Microprogrammed Computer

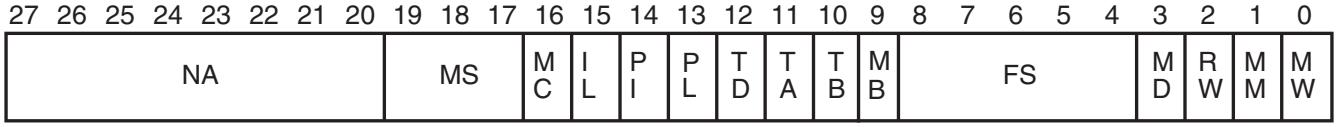

Fig. 8-27 Format for Microinstruction

TABLE 8-9
Control Word Information for Datapath

TD	TA	TB	MB		FS		MD	RW	MM	MW	
Select	Select	Select	Select	Code	Function	Code	Select	Function	Select	Function	Code
R[DR]	R[SA]	R[SB]	Register	0	$F = A$	00000	FnUt	No write (NW)	Address	No write (NW)	0
R8	R8	R8	Constant	1	$F = A + 1$	00001	Data In	Write (WR)	PC	Write (WR)	1
					$F = A + B$	00010					
					$F = A + B + 1$	00011					
					$F = A + \overline{B}$	00100					
					$F = A + \overline{B} + 1$	00101					
					$F = A - 1$	00110					
					$F = A$	00111					
					$F = A \wedge B$	01000					
					$F = A \vee B$	01010					
					$F = A \oplus B$	01100					
					$F = \overline{A}$	01110					
					$F = B$	10000					
					$F = \text{sr } B$	10100					
					$F = \text{sl } B$	11000					

Table 8-9 Control Word Information for Datapath

TABLE 8-10
Control Information for Sequence Control Fields

MS			MC		IL		PI		PL		
Action	**Symbolic Notation**	**Code**	**Select**	**Symbolic Notation**	**Action**	**Symbolic Notation**	**Action**	**Symbolic Notation**	**Action**	**Symbolic Notation**	**Code**
Increment *CAR*	CNT	000	NA	NXA	No load	NLI	No load	NLP	No load	NLP	0
Load *CAR*	NXT	001	Opcode	OPC	Load instr.	LDI	Increment PC	INP	Load PC	LDP	1
If $C = 1$, load *CAR*; else increment *CAR*	BC	010									
If $V = 1$, load *CAR*; else increment CAR	BV	011									
If $Z = 1$, load *CAR*; else increment *CAR*	BZ	100									
If $N = 1$, load *CAR*; else increment *CAR*	BN	101									
If $C = 0$, load *CAR*; else increment *CAR*	BNC	110									
If $Z = 0$, load *CAR*, else increment *CAR*	BNZ	111									

Table 8-10 Control Information for Sequence Control Fields

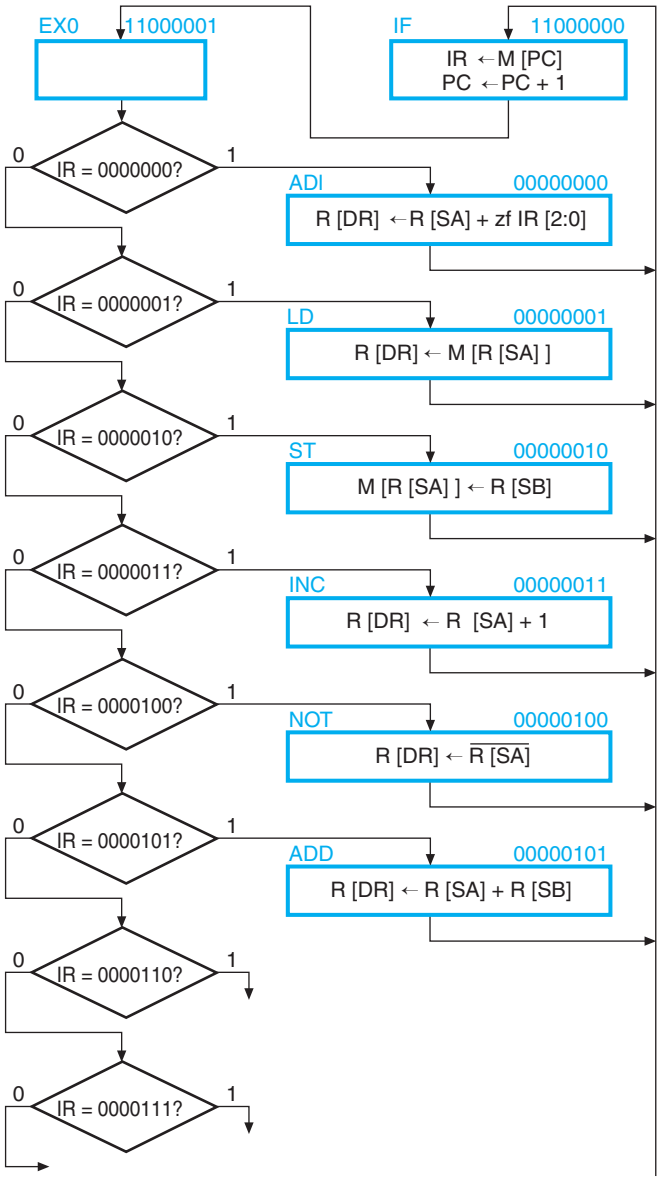

Fig. 8-28 ASM Chart for Multiple-Cycle Microprogrammed Computer

TABLE 8-11
Symbolic Microprogram for Fetch and Execution of Six Instructions

Address	NXT ADD	MS	MC	IL	PI	PL	TD	TA	TB	MB	FS	MD	RW	MM	MW
IF	EX0	CNT	—	LDI	INP	NLP	—	—	—	—	—	—	NW	PC	NW
EXO	—	NXT	OPC	NLI	NLP	NLP	—	—	—	—	—	—	NW	—	NW
ADI	IF	NXT	NXA	NLI	NLP	NLP	DR	SA	—	Constant	$F = A + B$	FnUt	WR	—	NW
LD	IF	NXT	NXA	NLI	NLP	NLP	DR	SA	—	—	—	Data	WR	MA	NW
ST	IF	NXT	NXA	NLI	NLP	NLP	—	SA	SB	Register	—	—	NW	MA	WR
INC	IF	NXT	NXA	NLI	NLP	NLP	DR	SA	—	—	$F = A + 1$	FnUt	WR	—	NW
NOT	IF	NXT	NXA	NLI	NLP	NLP	DR	SA	—	—	$F = \overline{A}$	FnUt	WR	—	NW
ADD	IF	NXT	NXA	NLI	NLP	NLP	DR	SA	SB	Register	$F = A + B$	FnUt	WR	—	NW

Table 8-11 Symbolic Microprogram for Fetch and Execution of Six Instructions

TABLE 8-12
Binary Microprogram for Fetch and Execution of Six Instructions

Address	NXT ADD	MS	MC	IL	PI	PL	TD	TA	TB	MB	FS	MD	RW	MM	MW
192	193	000	0	1	1	0	0	0	0	0	00000	0	0	1	0
193	000	001	1	0	0	0	0	0	0	0	00000	0	0	0	0
000	192	001	0	0	0	0	0	0	0	1	00010	0	1	0	0
001	192	001	0	0	0	0	0	0	0	0	00000	1	1	0	0
002	192	001	0	0	0	0	0	0	0	0	00000	0	0	0	1
003	192	001	0	0	0	0	0	0	0	0	00001	0	1	0	0
004	192	001	0	0	0	0	0	0	0	0	01110	0	1	0	0
005	192	001	0	0	0	0	0	0	0	0	00010	0	1	0	0

Table 8-12 Binary Microprogram for Fetch and Execution of Six Instructions

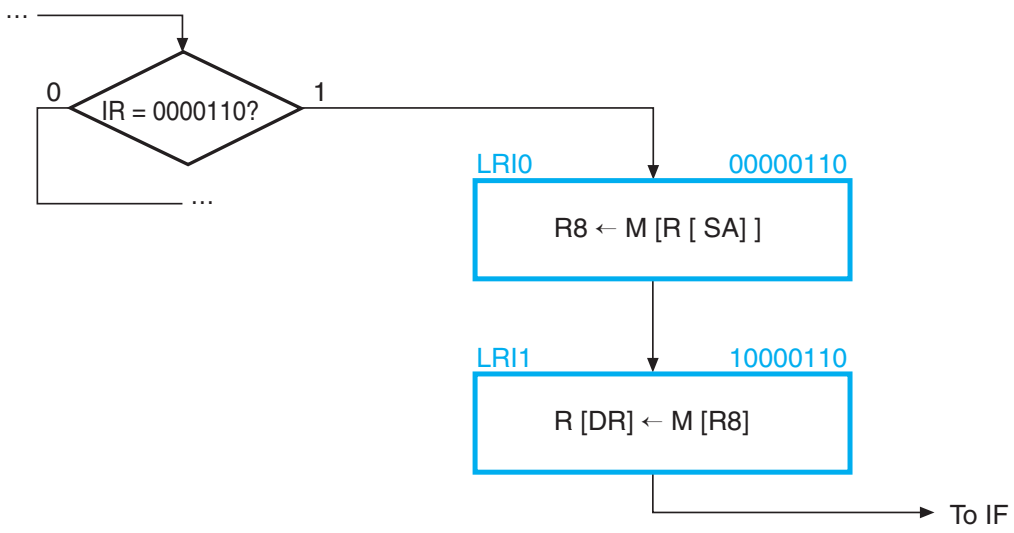

Fig. 8-29 ASM Chart for Register Indirect Instruction

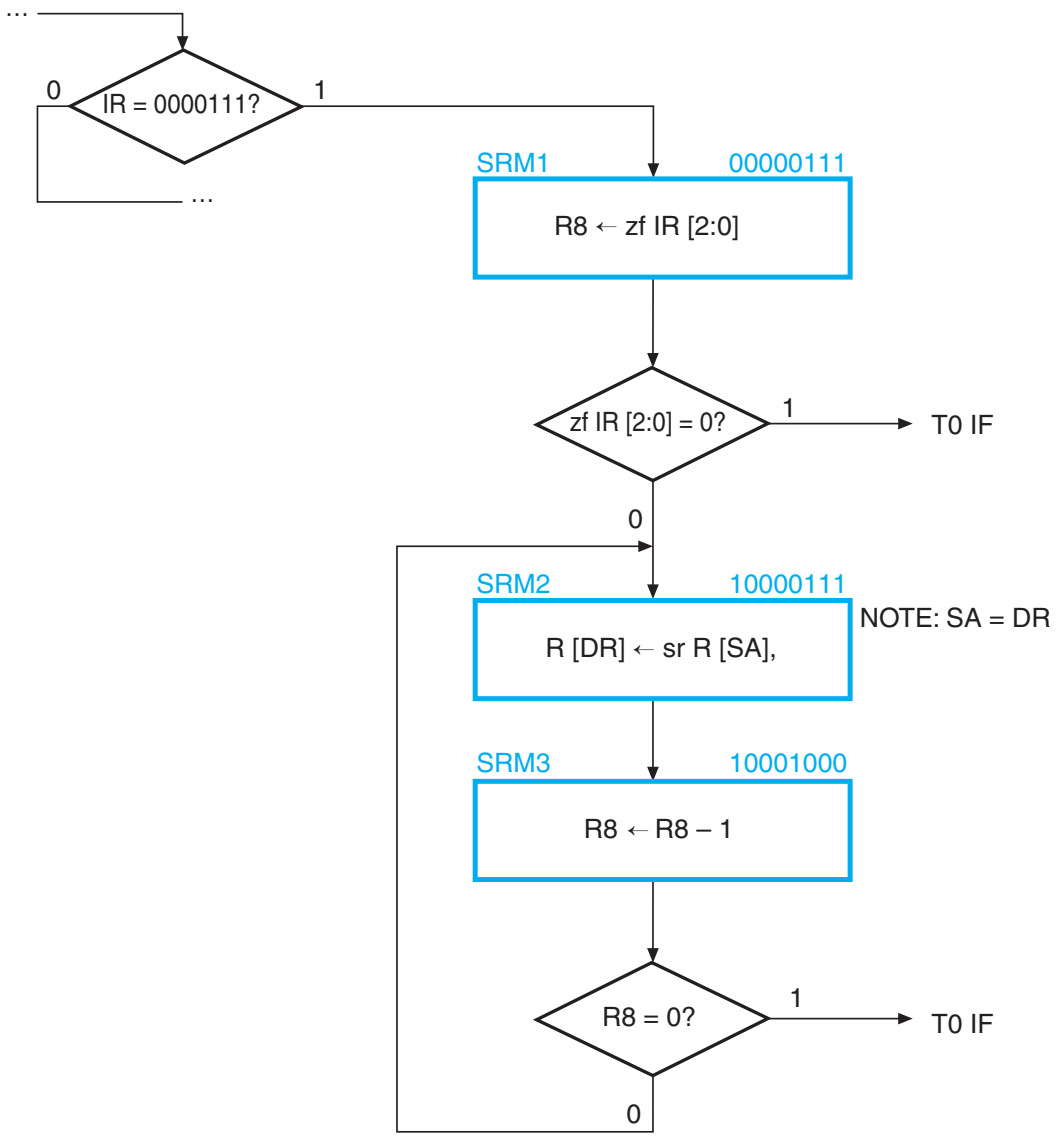

Fig. 8-30 ASM Chart for Right-Shift Multiple Instruction

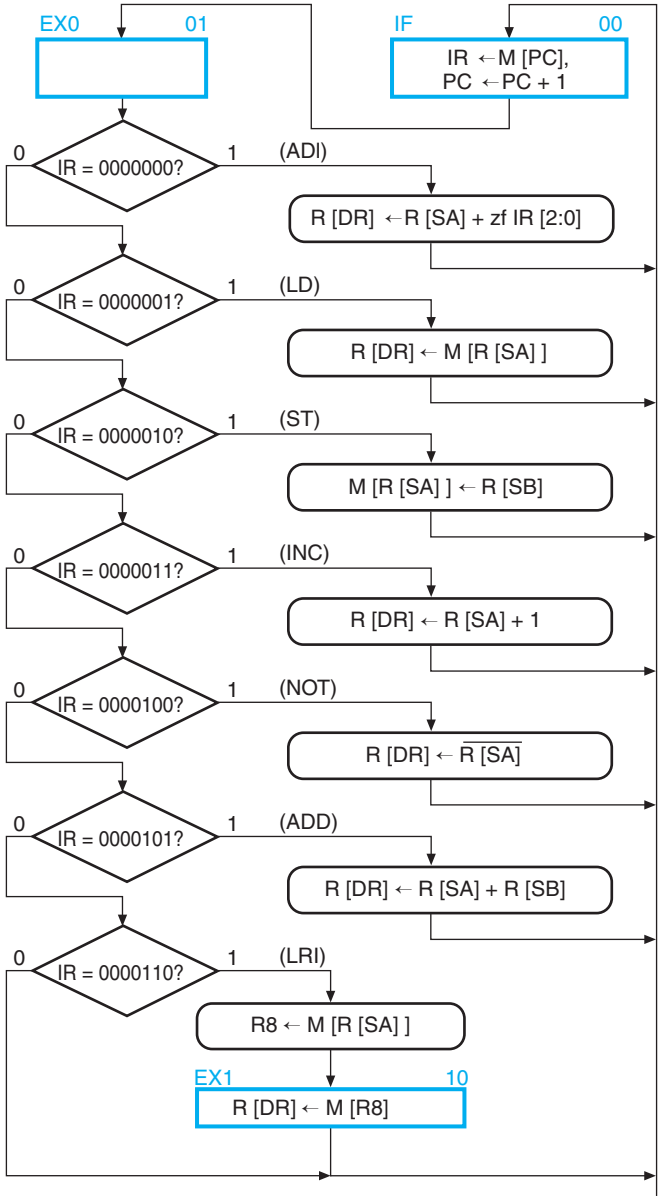

Fig. 8-31 ASM Chart for Multiple-Cycle, Decoder-Based Computer

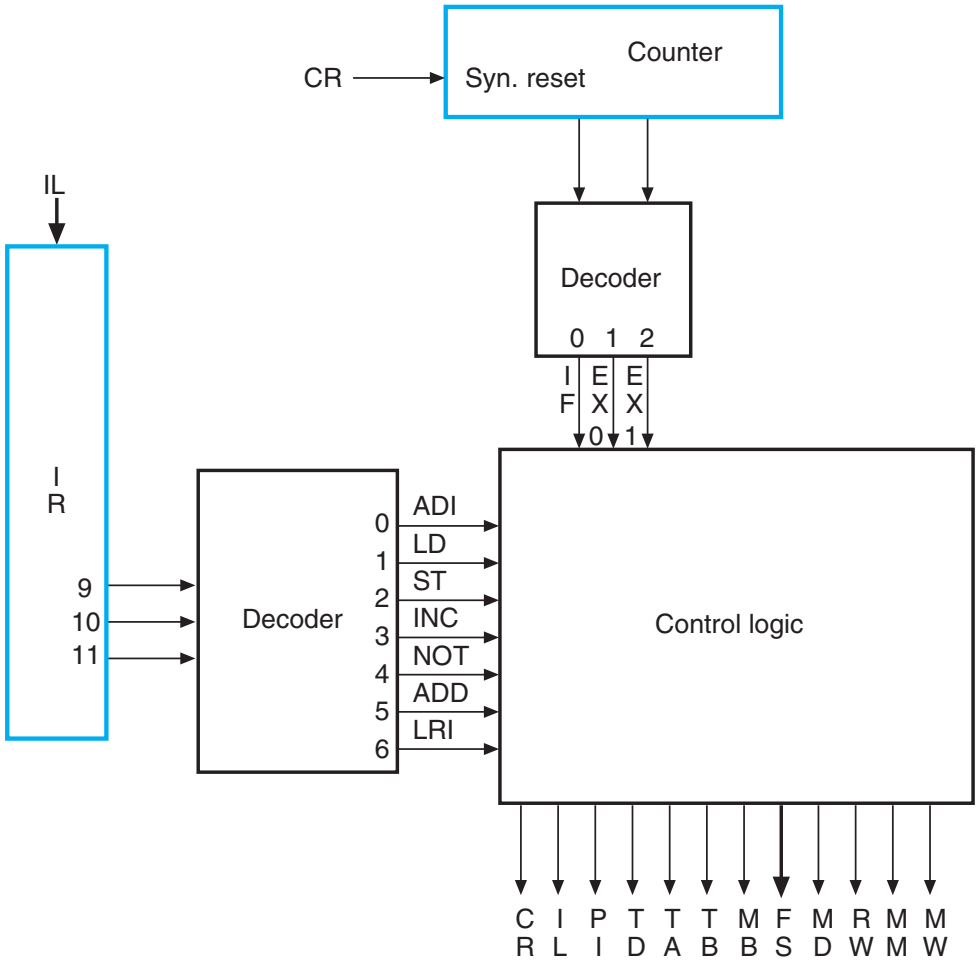

Fig. 8-32 Block Diagram of Hardwired Counter and Decoder-Based, Multiple-Cycle Control Unit

Fig. 8-33 Assembly Line Analogy to Computer Pipeline

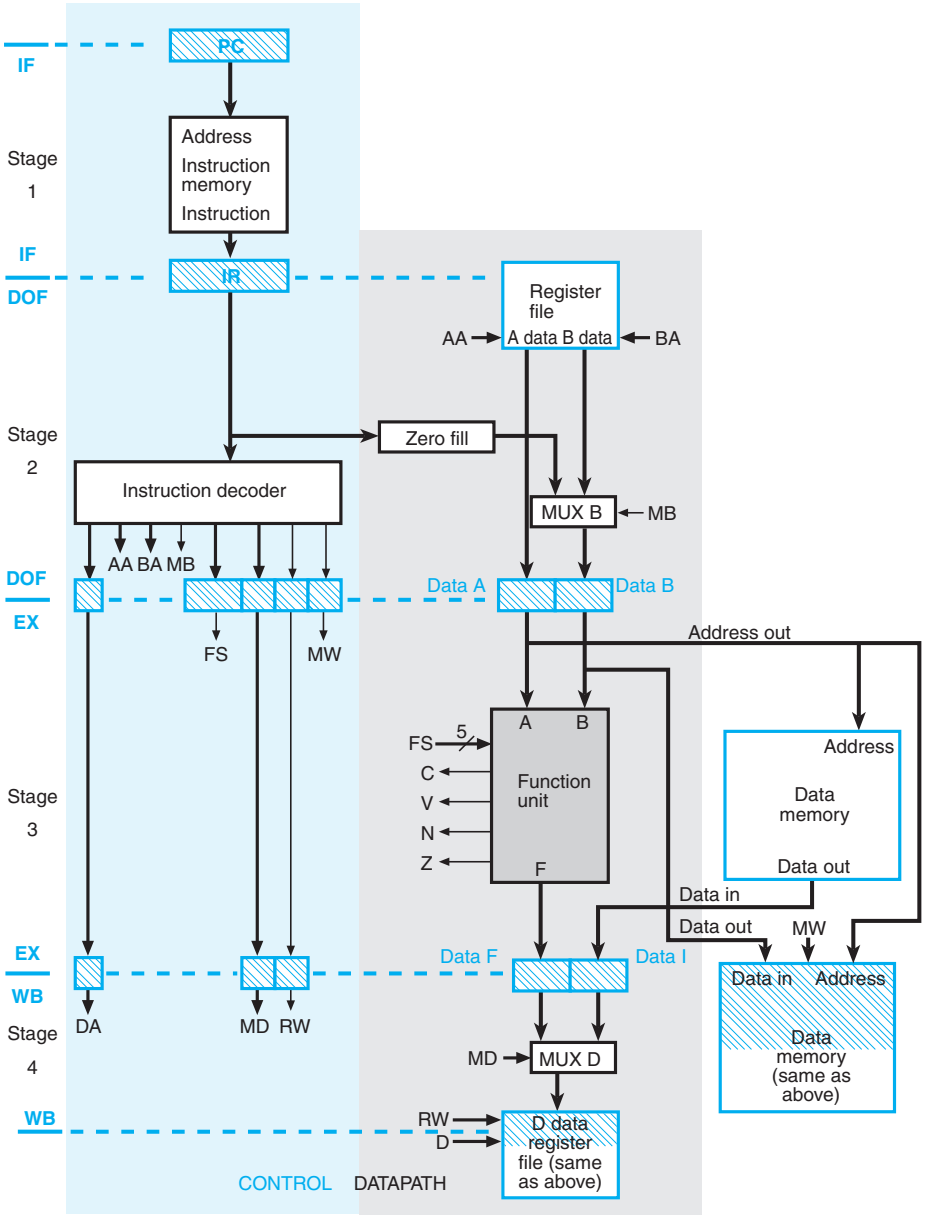

Fig. 8-34 Block Diagram of Pipelined Computer

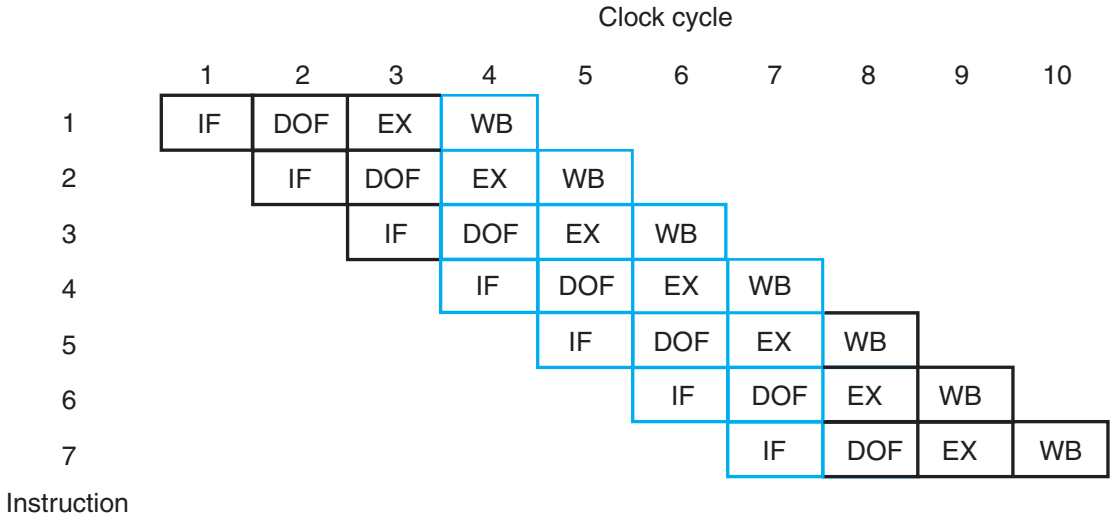

Fig. 8-35 Pipeline Execution Pattern of Register Number Program

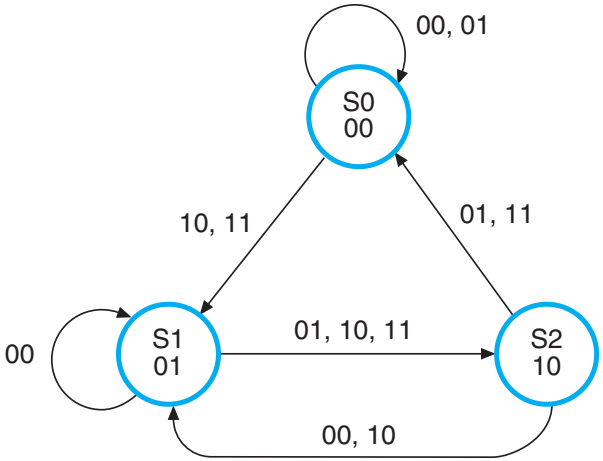

Fig. 8-36 State Diagram for Problem 8-1