CHAPTER 8

8-1.

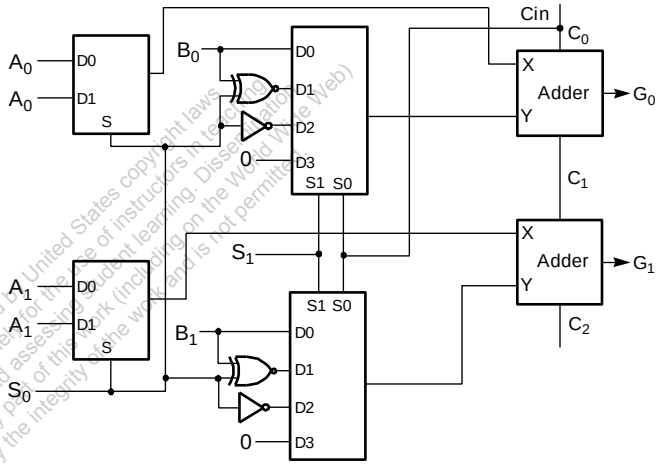
$\log_2 64 = 6$ lines/mux or decoder

8-2.*

$$C = C_8$$
$$V = C_8 \oplus C_7$$
$$Z = \overline{F_7 + F_6 + F_5 + F_4 + F_3 + F_2 + F_1 + F_0}$$
$$N = F_7$$

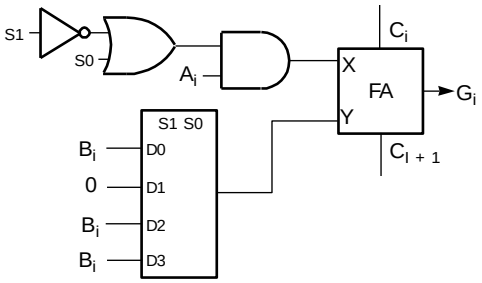
8-3.*

$$X = S_0\bar{A} + \bar{S}_0 A$$
$$Y = \bar{S}_1\bar{C}_{in}B + \bar{S}_1 S_0 B + \bar{S}_1\bar{S}_0\bar{B} + S_1\bar{S}_0\bar{C}_{in}$$



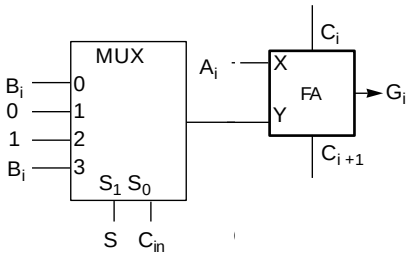
8-4.*

$$X = A\overline{S1} + AS0$$
$$Y = B\overline{S1}\,\overline{S0} + \bar{B}S_1$$



8-5.

Connect to Cin for first stage.



$00: G = A + B$ (Add)
$01: G = A + 1$ (Increment)
$10: G = A - 1$ (Decrement)
$11: G = A + \overline{B} + 1$ (Subtract)

Cascade four such stages, connecting the carries.

8-6.*

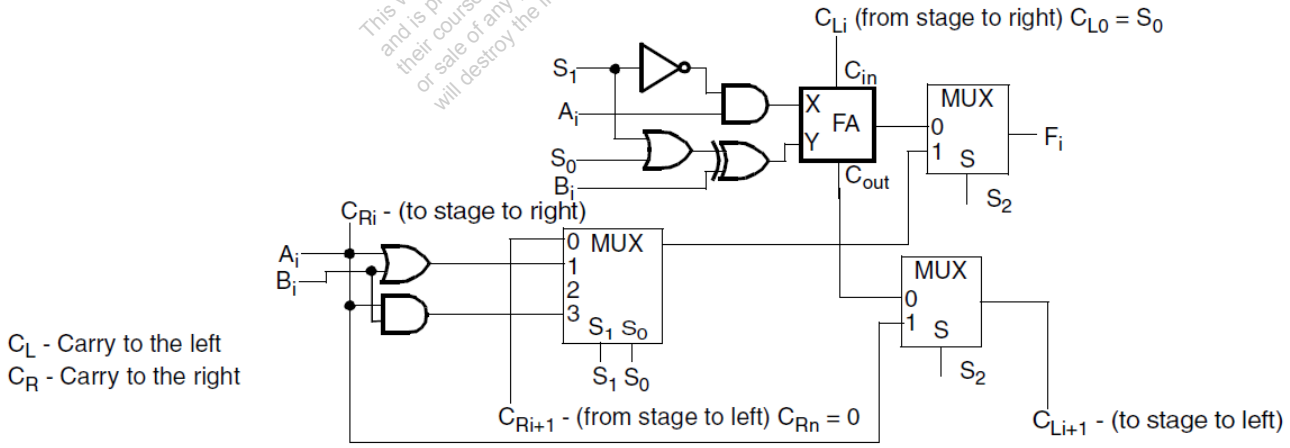
a) XOR = 00, NAND = 01, NOR = 10 XNOR = 11

$$\text{Out} = S_1\overline{A}\overline{B} + \overline{S}_1\overline{A}B + \overline{S}_1A\overline{B} + S_1S_0AB + (\text{one of } S_0\overline{A}\overline{B} + \overline{S}_1S_0\overline{A})$$

b) The above is a simplest result.

8-7.+

S2	S1	S0	Operation	S2	S1	S0	Operation
0	0	0	$A + B$	1	0	0	sr A
0	0	1	$A + B + 1$	1	0	1	$A \vee B$
0	1	0	$\overline{B}$	1	1	0	sl A
0	1	1	$\overline{B} + 1$	1	1	1	$A \wedge B$



8-8.*

(a) 1010 (b) 1110 (c) 0101 (d) 1101

8-9.

	DA	AA	BA	MB	FS	MD	RW
(a)	011	- - -	- - -	-	- - - -	1	1
(b)	100	000	000	0	1010	0	1
(c)	001	100	- - -	-	1101	0	1
(d)	011	011	- - -	-	0001	0	1
(e)	010	010	- - -	-	1110	0	1
(f)	001	010	100	0	1010	0	1
(g)	111	001	011	0	0010	0	1
(h)	100	101	- - -	0	0101	0	1

8-10.*

(a) $R5 \leftarrow R4 \wedge R5$ R5 = 0000 0100

(b) $R6 \leftarrow R2 + \overline{R4} + 1$ R6 = 1111 1110

(c) $R5 \leftarrow R0$ R5 = 0000 0000

(d) $R5 \leftarrow R0$ R5 = 0000 0000

(e) $R4 \leftarrow$ *sr*Constant R4 = 0000 0011

(f) $R3 \leftarrow$ Data in R3 = 0001 1011

8-11.

$R3 \leftarrow R3 + R1,\ R3 = 01100111$

$R4 \leftarrow R4 \vee R1,\ R4 = 01110100$

$R5 \leftarrow R5 \oplus R1,\ R5 = 01101100$

$R1 \leftarrow \overline{R1},\ R1 = 11011111$

$R1 \leftarrow R1 + 1,\ R1 = 1100000$

$R6 \leftarrow R6 + \overline{R1} + 1,\ R6 = 01100001$

$R7 \leftarrow R7 + \overline{R1} + 1,\ R7 = 01101001$

$R1 \leftarrow R7,\ R1 = 01101001$

R1 ... R7 = i, D, g, t, l, a, i Unscrambled is Digital

8-12.

a) 64 b) 32 c) 0 to 65,535 d) –32,768 to +32,767

8-13.*

a) Opcode = 8 bits b) 18 bits c) 262,144 d) +131,071 and –131,072

8-14.

Distinct Opcodes = 4 + 16 + 127 = 148

8-15.

Instruction Register Transfer	DA	AA	BA	MB	FS	MD	RW	MW	PL	JB
$R[0] \leftarrow R[7] \oplus R[3]$	000	111	011	0	1010	0	1	0	0	x
$R[1] \leftarrow M[R[4]]$	001	100	xxx	x	xxxx	1	1	0	0	x
$R[2] \leftarrow R[5] + 2$	010	101	xxx	1	0010	0	1	0	0	x
$R[3] \leftarrow$ sl $R[6]$	011	xxx	110	0	1110	0	1	0	0	x
if $R[4] = 0$ then $PC \leftarrow PC +$ se PC else $PC \leftarrow PC + 1$	xxx	100	xxx	x	0000	x	0	0	1	0

Instruction Register Transfer	Operation Code	DR	SA	SB or Operand
$R[0] \leftarrow R[7] +$ R[6]	000 0010	000	111	110
$R[1] \leftarrow R[5] - 1$	000 0110	001	110	000
$R[2] \leftarrow$ sl $R[4]$	000 1110	010	000	100
$R[3] \leftarrow \overline{R[3]}$	000 1011	011	011	000
$R[4] \leftarrow R[2] \vee R[1]$	000 1001	100	010	001

8-16.

MB: B15 - Correct for table specification

MD: B13 - Correct for table specification

RW: $\overline{B14}$ - Correct for table specification

MW: $\overline{B15}$B14 - Correct for table specification

PL: B15 B14 - Correct for table specification

JB: B13 - Correct for table specification

BC: B9 - Correct for table specification

FS: The logic gives 0000 for FS for PL =1 which selects the value on the A input to the Function Unit to be evaluated for branches and blocks the value on bit 9 which is otherwise used as BC for branches. For PL = 0, the normal bits for FS are passed as required from the op code. Correct.

8-17.

Instruction	**Code**	**Registers/Memory changed**
ADD R0, R1, R2	000 0101 000 001 010	
SUB R3, R4, R5	000 001 011 100 101	R0 = 3
SUB R6, R7, R0	000 0101 110 111 000	R3 = -1
ADD R0, R0, R3	000 0101 000 000 011	R6 = 4
SUB R0, R0, R6	000 0101 000 000 110	R0 = 2
ST R7, R0	010 0000 000 111 000	R0 = -2
LD R7, R6	011 0000 111 110 000	M[7] = -2
ADI R0, R6, 2	100 0010 000 110 000	R7 = M[4]
ADI R3, R6, 3	100 0010 011 110 011	R0 = 6
		R3 = 7

8-18.

$$R[4] \leftarrow R[4] \oplus R[4]$$

V and C are produced by the arithmetic circuit. For the XOR FS code, S1 = 0, S0 = 1. and Cin = 0, giving arithmetic operation Add. For arbitrary contents in R[4], the values of C and V are a function of the sign and value of the contents of R4 and are unpredicable.

8-19.

Part	State	Opcode	VCNZ	Next State	IL	PS	DX	AX	BX	MB	FS	MD	RW	MM	MW
(a)	EX0	0000101	xxxx	EX1	0	01	0xxx	0xxx	0xxx	0	0101	0	1	0	0
(b)	EX0	0001101	xxxx	INF	0	01	1000	xxxx	1000	0	1101	0	1	0	0
(c)	EX0	1100000	xxx0	INF	0	10	xxxx	0xxx	xxxx	x	0000	0	0	0	0
	EX0	1100000	xxx1	INF	0	01	xxxx	0xxx	xxxx	x	0000	0	0	0	0
(d)	EX0	0000000	xxxx	INF	0	01	0xxx	0xxx	xxxx	x	0000	0	1	0	0

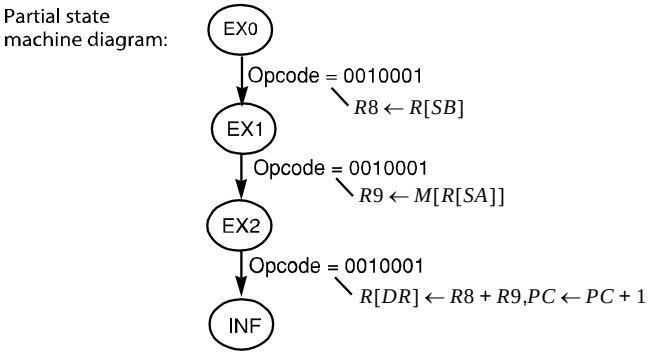
8-20.

State	Instruction	R8	R9	Z	Next State
EX0	$R8 \leftarrow R[SA]$			0	EX1
EX1	$R9 \leftarrow zfOP$	0101100111000111	x	0	EX2
EX2	$R8 \leftarrow srR8$	0101100111000111	5	0	EX3
EX3	$R9 \leftarrow R9 - 1$	0010110011100011	5	0	EX2
EX2	$R8 \leftarrow srR8$	0010110011100011	4	0	EX3
EX3	$R9 \leftarrow R9 - 1$	0001011001110001	4	0	EX2
EX2	$R8 \leftarrow srR8$	0001011001110001	3	0	EX3
EX3	$R9 \leftarrow R9 - 1$	0000101100111000	3	0	EX2
EX2	$R8 \leftarrow srR8$	0000101100111000	2	0	EX3
EX3	$R9 \leftarrow R9 - 1$	0000010110011100	2	0	EX2
EX2	$R8 \leftarrow srR8$	0000010110011100	1	0	EX3
EX3	$R9 \leftarrow R9 - 1$	0000001011001110	1	1	EX4
EX4	$R[DR] \leftarrow R8$	0000001011001110	0	0	INF
INF	-	0000001011001110	0	-	-

8-21.+

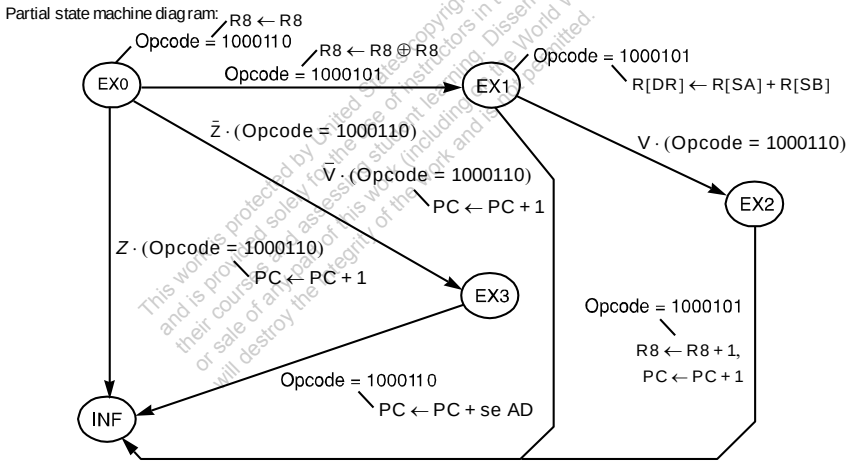
Removal of the two decisions on zero operations does not affect the number of states in the state machine diagram. The reason for this is that the same states are required because the datapath of the computer only supports one register transfer per clock cycle. The transfers required by the instruction execution force the state structure. As a consequence, there can be no reduction of the number of clock cycles required to execute the instructions. The new state machine diagram is actually a worst case in terms of execution time for the instruction execution. The two decisions can only improve the execution times. So the analysis suggested is unnecessary.

8-22.



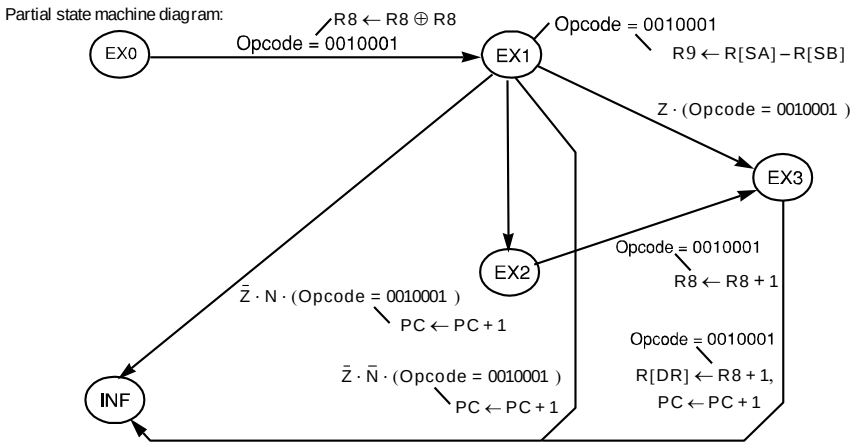
State	Opcode	VCNZ	Next State	IL	PS	DX	AX	BX	MB	FS	MD	RW	MM	MW
EX0	0010001	xxxx	EX1	0	00	1000	xxxx	0xxx	0	1100	0	1	0	0
EX1	0010001	xxxx	EX2	0	00	1001	0xxx	xxxx	x	xxxx	1	1	0	0
EX2	0010001	xxxx	INF	0	01	0xxx	1000	1001	0	0010	1	0	0	0

8-23.



Part	State	Opcode	VCNZ	Next State	IL	PS	DX	AX	BX	MB	FS	MD	RW	MM	MW
AOV	EX0	1000101	xxxx	EX1	0	00	0xxx	1000	1000	0	1010	0	1	x	0
	EX1	1000101	1xxx	EX2	0	00	0xxx	0xxx	0xxx	0	0010	0	1	x	0
	EX1	1000101	0xxx	INF	0	01	0xxx	0xxx	0xxx	0	0010	0	1	x	
	EX2	1000101	xxxx	INF	0	01	1000	1000	xxxx	0	0001	0	1	x	0
BRV	EX0	1000110	xxx0	EX3	0	00	1000	1000	xxxx	x	0000	0	1	x	0
	EX0	1000110	xxx1	INF	0	01	1000	1000	xxxx	x	0000	0	1	x	0
	EX3	1000110	xxxx	INF	0	10	xxxx	xxxx	xxxx	x	xxxx	0	0	x	0

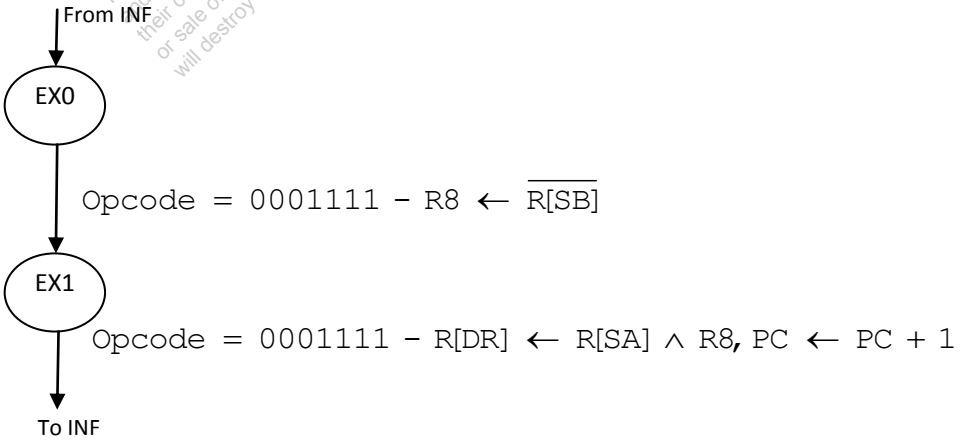
8-24.



State	Opcode	VCNZ	Next State	IL	PS	DX	AX	BX	MB	FS	MD	RW	MM	MW
EX0	0010001	xxxx	EX1	0	00	1000	1000	1000	0	1010	0	1	0	0
EX1	0010001	xxx1	EX3	0	00	1001	0xxx	0xxx	0	0101	0	1	0	0
EX1	0010001	xx00	EX2	0	00	1001	0xxx	0xxx	0	0101	0	1	0	0
EX1	0010001	xx10	INF	0	01	1001	0xxx	0xxx	0	0101	0	1	0	0
EX2	0010001	xxxx	EX3		00	1000	1000	xxxx	x	0001	0	1	0	0
EX3	0010001	xxxx	INF	0	01	0xxx	1000	xxxx	x	0001	0	1	0	0

8-25.

Partial state machine diagram, assuming the opcode for the new instruction is 0001111:



8-26.

The operation code used for SMR and these instructions is 0111111 which is easy to generate by complementing all 0's. The word to be stored in memory is built in a register by complementing all 0's and ANDing the result with the value of SB from the original instruction. The register value generated is incremented, stored in memory and loaded into the IR after the execution of each transfer from a register to a memory location.

Register Transfer	State	Opcode	VCNZ	Next State	IL	PS	DX	AX	BX	MB	FS	MD	RW	MM	MW
R8 ← R[SA]	EX0	0111111	xxxx	EX1	0	00	1000	0xxx	xxxx	0	0000	1	1	0	0
R9 ← R9 ⊕ R9 (R9 ← 0)	EX1	0111111	xxxx	EX2	0	0	1001	1001	1001	0	1010	0	1	0	0
R9 ← $\overline{R9}$	EX2	0111111	xxxx	EX3	0	0	1001	1001	xxxx	x	1011	0	1	0	0
R10 ← sr R9	EX3	0111111	xxxx	EX4	0	0	1010	1001	xxxx	x	1101	0	0	0	0
R9 ← R10 ∧ zf SB	EX4	0111111	xxxx	EX5	0	0	1001	1010	xxxx	1	1000	0	1	0	0
R11 ← zf SB	EX5	0111111	xxxx	EX6	0	0	1011	xxxx	xxxx	1	1100	0	1	0	0
M[R10] ← R9	EX6	0111111	xxxx	EX7	0	0	xxxx	1010	1001	0	xxxx	x	0	0	1
IR ← M[R10]	EX7	0111111	xxxx	EX8	1	0	xxxx	1010	xxxx	x	xxxx	x	0	0	0
M[R8] ← R[SB]	EX8	0111111	xxxx	EX9	0	0	xxxx	1000	0xxx	0	xxxx	1	1	0	0
R8 ← R8 + 1	EX9	0111111	xxxx	EX10	0	0	1000	1000	xxxx	x	0001	0	1	0	0
R9 ← R9 + 1	EX10	0111111	xxxx	EX11	0	0	1001	1001	xxxx	x	0001	0	1	0	0
R11 – zf SB, $\overline{Z}$:→ EX5	EX11	0111111	xxx0	EX5	0	0	xxxx	1011	xxxx	1	0101	x	0	0	0
R11 – zf SB, Z :→ INF, inc PC	EX11	0111111	xxx1	INF	0	01	xxxx	1011	xxxx	1	0101	x	0	0	0

8-27.

Since the condition codes are not fully available to the programmer (only N and Z are used by instructions, not V and C), the approach of problem 6-1, part c, for a signed comparison using the N and V bits is not possible. Instead the program must make the comparisons based upon the signs of the current minimum value and the current array element.

```
// Assembly for the solution to problem 8-27
// Logic and Computer Design Fundamentals, 5th edition
//
// The problem specifies that the pointer to the array is in memory address 0,
// the length is in address 1, and the result should be stored in address 2.
//
// Register usage
// R0: Pointer to current location in array
// R1: Loop variable for the array length remaining to be checked
// R2: Temporary variable for current array element value
// R3: The current minimum values
// R4: Pointer to where the result will be stored
// R5: Temporary variable for comparisons of the current minimum value and current array
element
//
// Using labels for locations to make the targets of branches and jumps more clear


    LDI R0, 0       // Read in pointer to array and its length
    LDI R1, 1
    LD R0, R0
    LD R1, R1
    LDI R4, 2       // Set R4 to point to the location for storing the result
    LD R3, R0       // R3 = array[0]
    DEC R1, R1
    BRZ R1, done    // handling the case where the array is only one element
    INC R0, R0
loop: LD R2, R0     // temporary = array[i]
    XOR R5, R3, R2  // Compare the signs of current minimum and temporary
    BRN R5, different_signs
    SUB R5, R3, R2  // If same sign, min = temporary if result of subtraction
                    //is positive or zero
    BRN no_new_min
new_min: MOV R3, R2
no_new_min: DEC R1, R1 // Update loop variable
    BRZ R1, done
    INC R0, R0      // Point to the next array element
    JMP loop
different_signs:
    BRN R3, no_new_min  // If different signs and current minimum is negative,
                        // it is still the minimum
    JMP new_min
done:
    ST R4, R3       // Save the minimum to memory
end: JMP end        // Infinite loop because there's no halt instruction
```