OUTCOME #4: "An ability to design and implement computer logic circuits."

Multiple Choice – select the single most appropriate response for each question.

Note that none of the above MAY be a VALID ANSWER.

1. The **five-bit** sign and magnitude number, **$SM(10101)_2$**, converted to **radix** notation, is:
   (A) R $(10101)_2$
   (B) R $(01010)_2$
   (C) R $(11010)_2$
   (D) R $(11011)_2$
   (E) none of the above

2. When **subtracting** the **five-bit** signed numbers **$(10111)_2$ – $(11001)_2$** using **radix arithmetic**, the result obtained is:
   (A) $(01110)_2$
   (B) $(11111)_2$
   (C) $(11110)_2$
   (D) overflow *(invalid result)*
   (E) none of the above

3. The number of **diagonals** of full adder cells required to implement a **5x3** unsigned binary multiplier is:

   (A) 3 (B) 4 (C) 5 (D) 6 (E) none of these

4. The **total number** of **full adder cells** required to implement a **5x3** unsigned binary multiplier is:
   (A) 8
   (B) 10
   (C) 12
   (D) 24
   (E) none of the above

5. If an **N-nanosecond** PLD were used to **generate each product component bit** (using a separate macrocell for each product component bit), and **implement each full adder cell** (using a pair of macrocells to implement each full adder cell), the worst case propagation delay of a **5x3** unsigned binary multiplier array would be:
   (A) 7N
   (B) 8N
   (C) 9N
   (D) 10N
   (E) none of the above

The following Verilog module applies to questions 6-8:

```
/* 4-bit Carry Look-Ahead Adder Block – Practice Exam Version */

module cla4pe(X, Y, CIN, C, S);

  input wire [3:0] X, Y;    // Operands
  input wire CIN;           // Carry in
  output wire [3:0] C;      // Carry equations (C[3] is Cout)
  output wire [3:0] S;      // Sum outputs

  wire [3:0] G, P;
  // Generate functions
  assign G = X & Y;
  // Propagate functions
  assign P = X ^ Y;

  // Carry equations
  assign C[0] = G[0] | CIN&P[0];
  assign C[1] = G[1] | G[0]&P[1] | CIN&P[0]&P[1];
  assign C[2] = G[2] | G[1]&P[2] | G[0]&P[1]&P[2]
                | CIN&P[0]&P[1]&P[2];
  assign C[3] = G[3] | G[2]&P[3] | G[1]&P[2]&P[3]
                | G[0]&P[1]&P[2]&P[3]
                | CIN&P[0]&P[1]&P[2]&P[3];

  // Sum equations
  assign S[0] = CIN ^ P[0];
  assign S[3:1] = C[2:0] ^ P[3:1];

endmodule
```

6. If realized as coded above, the **number of product terms** required to implement the equation for **C[0]** would be:

   (A) **3** (B) **4** (C) **7** (D) **10** (E) none of the above

7. If realized as coded above, the **number of product terms** required to implement the equation for **C[1]** would be:

   (A) **3** (B) **4** (C) **7** (D) **10** (E) none of the above

8. If realized as coded above, the **number of product terms** required to implement the equation for **S[1]** would be:

   (A) **3** (B) **4** (C) **7** (D) **10** (E) none of the above

The following chart applies to questions 9-11:

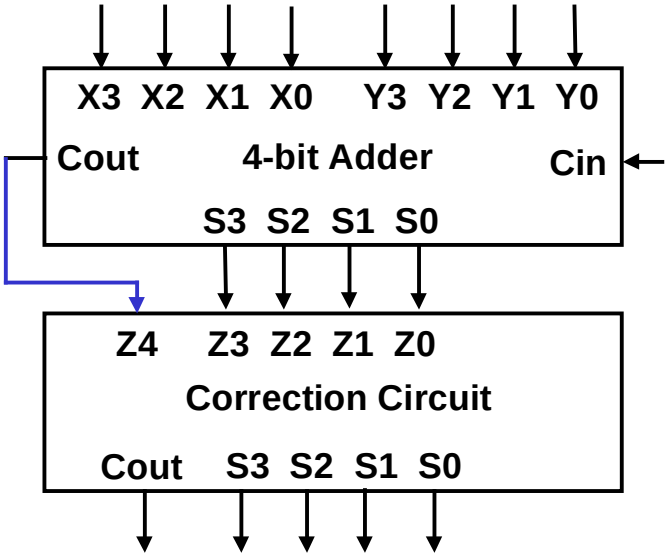
A_1	A_0	(A)	B_1	B_0	(B)	?	C	Z	N	V
0	0	0	0	0	0	(A) = (B)	1	1	0	0
0	0	0	0	1	+1	(A) < (B)	0	0	1	0
0	0	0	1	0	-2	(A) > (B)	0	0	1	1
0	0	0	1	1	-1	(A) > (B)	0	0	0	0
0	1	+1	0	0	0	(A) > (B)	1	0	0	0
0	1	+1	0	1	+1	(A) = (B)	1	1	0	0
0	1	+1	1	0	-2	(A) > (B)	0	0	1	1
0	1	+1	1	1	-1	(A) > (B)	0	0	1	1
1	0	-2	0	0	0	(A) < (B)	1	0	1	0
1	0	-2	0	1	+1	(A) < (B)	1	0	0	1
1	0	-2	1	0	-2	(A) = (B)	1	1	0	0
1	0	-2	1	1	-1	(A) < (B)	0	0	1	0
1	1	-1	0	0	0	(A) < (B)	1	0	1	0
1	1	-1	0	1	+1	(A) < (B)	1	0	1	0
1	1	-1	1	0	-2	(A) > (B)	1	0	0	0
1	1	-1	1	1	-1	(A) = (B)	1	1	0	0

9. The function for "A *equals* B" ($F_{A=B}$) can be expressed as:
   (A) $F_{A=B} = C$
   (B) $F_{A=B} = Z$
   (C) $F_{A=B} = N$
   (D) $F_{A=B} = V$
   (E) none of the above

10. The function for "A less than B" ($F_{A<B}$) can be expressed as:
   (A) $F_{A<B} = Z + N' \cdot V + N \cdot V'$
   (B) $F_{A<B} = N' \cdot V + N \cdot V'$
   (C) $F_{A<B} = N' \cdot V' + N \cdot V$
   (D) $F_{A<B} = Z' + N' \cdot V' + N \cdot V$
   (E) none of the above

11. The function for "A greater than or equal to B" ($F_{A\geq B}$) can be expressed as:
   (A) $F_{A\geq B} = Z + N' \cdot V + N \cdot V'$
   (B) $F_{A\geq B} = N' \cdot V + N \cdot V'$
   (C) $F_{A\geq B} = N' \cdot V' + N \cdot V$
   (D) $F_{A\geq B} = Z' + N' \cdot V' + N \cdot V$
   (E) none of the above

The following block diagram of a BCD full adder cell applies to questions 12-14:



12. The purpose of the "correction circuit" is to:
    (A) always add $(0110)_2$ to the direct sum (Z3 Z2 Z1 Z0)
    (B) add $(0110)_2$ to the direct sum (Z3 Z2 Z1 Z0) only if Z4=1
    (C) always subtract $(0110)_2$ from the direct sum (Z3 Z2 Z1 Z0)
    (D) add $(0110)_2$ to the direct sum (Z3 Z2 Z1 Z0) only if Z4+Z3·Z2+Z3·Z1=1
    (E) none of the above

13. If **X3..X0 = 0111, Y3..Y0 = 0011, and Cin = 1,** the value **input to** the correction circuit (Z4 Z3 Z2 Z1 Z0) will be:
    (A) 0 1 0 1 0
    (B) 0 1 0 1 1
    (C) 1 0 0 0 0
    (D) 1 0 0 0 1
    (E) none of the above

14. If **X3..X0 = 0111, Y3..Y0 = 0011, and Cin = 1,** the value **output by** the correction circuit (Cout S3 S2 S1 S0) will be:
    (A) 0 1 0 1 0
    (B) 0 1 0 1 1
    (C) 1 0 0 0 0
    (D) 1 0 0 0 1
    (E) none of the above

15. As observed in Lab 13, incrementing the program counter (PC) on the **same clock edge** that loads the instruction register (IR) **does <u>not</u> cause a problem** because:
    (A) the memory will ignore the new address the PC places on the address bus
    (B) the output buffers in the PC will not allow the new PC value to affect the address bus until the next fetch cycle
    (C) the IR will be loaded with the value on the data bus prior to the clock edge while the contents of the PC will increment after the clock edge
    (D) the value in the PC will change in time for the correct value to be output on the address bus (and fetch the correct instruction), before the IR load occurs
    (E) none of the above

16. If a program contains **more PSH instructions than POP instructions,** the following is likely to occur:
    (A) stack underflow (stack collides with beginning of program space)
    (B) stack overflow (stack collides with end of program space)
    (C) program counter overflow (program counter wraps to beginning of program space)
    (D) program counter underflow (program counter wraps to end of program space)
    (E) none of the above

17. **Whether or not** a conditional branch is ***taken*** or ***not taken*** depends on:
    (A) the value of the program counter
    (B) the state of the condition code bits
    (C) the cycle of the state counter
    (D) the value in the accumulator
    (E) none of the above

18. The state counter in the "extended" machine's instruction decoder and micro-sequencer needs **<u>both</u>** a **synchronous reset** (RST) **<u>and</u>** an **asynchronous reset** (ARS) because:
    (A) we want to make sure the state counter gets reset
    (B) the ARS signal allows the state counter to be reset to the "fetch" state when START is pressed, while the RST allows the state counter to be reset when the last execute cycle of an instruction is reached
    (C) the RST signal allows the state counter to be reset to the "fetch" state when START is pressed, while ARS allows the state counter to be reset when the last execute cycle of an instruction is reached
    (D) the state counter is not always clocked
    (E) none of the above

19. When a set of control signals are said to be **mutually exclusive**, it means that:
    (A) all the control signals may be asserted simultaneously
    (B) only one control signal may be asserted at a given instant
    (C) each control signal is dependent on the others
    (D) any combination of control signals may be asserted at a given instant
    (E) none of the above

Figure 1 on the attached *Reference Sheets* applies to questions 20 through 22.

20. If the input control combination **AOE=0, ALE=1, ALX=0, ALY=0** is applied to this circuit, the function performed will be:
    (A) ADD
    (B) SUBTRACT
    (C) LOAD
    (D) NEGATE
    (E) none of the above

21. If the input control combination **AOE=0, ALE=1, ALX=1, ALY=0** is applied to this circuit, the function performed will be:
    (A) ADD
    (B) SUBTRACT
    (C) LOAD
    (D) NEGATE
    (E) none of the above

22. If the input control combination **AOE=1, ALE=1, ALX=1, ALY=1** is applied to this circuit, the function (inadvertently) performed on (A) will be equivalent to a:
    (A) logical left shift
    (B) logical right shift
    (C) rotate left
    (D) rotate right
    (E) none of the above

Figure 2 on the attached *Reference Sheets* applies to questions 23 through 25.

23. When the program stops ("halts"), the value in the **A register** will be:
    (A) 0100 1100
    (B) 1000 0000
    (C) 0000 0000
    (D) 0100 0011
    (E) none of the above

24. When the program stops ("halts"), the **condition code bits** will be:
    (A) CF = 0, NF = 0, VF = 0, ZF = 0
    (B) CF = 1, NF = 1, VF = 0, ZF = 0
    (C) CF = 0, NF = 1, VF = 1, ZF = 0
    (D) CF = 1, NF = 0, VF = 0, ZF = 1
    (E) none of the above

25. When the program stops ("halts"), the value in the **program counter** will be:
    (A) 00000
    (B) 00110
    (C) 00111
    (D) 01111
    (E) none of the above

Figure 3 on the attached *Reference Sheets* applies to questions 26 through 30.

NOTE: If there is only one execute state, then "first execute state" ≡ "last execute state".

26. The following control signal is **not** asserted on a **fetch** cycle:
    (A) IRA
    (B) IRL
    (C) PCC
    (D) POA
    (E) none of the above

27. Assuming a **standard stack convention** (in which the **SP register** points to the **top stack item**), the following **SP control signal(s)** will be asserted on the **first execute cycle** of a **JSR** instruction:
    (A) SPI
    (B) SPD
    (C) SPI and SPA
    (D) SPD and SPA
    (E) none of the above

28. Assuming a **standard stack convention** (in which the **SP register** points to the **top stack item**), the following **PC control signal** will be asserted on the **last execute cycle** of a **JSR** instruction:
    (A) POA
    (B) POD
    (C) PLA
    (D) PLD
    (E) none of the above

29. Assuming a **standard stack convention** (in which the **SP register** points to the **top stack item**), the following **SP control signal(s)** will be asserted on the **first execute cycle** of an **RTS** instruction:
    (A) SPI
    (B) SPD
    (C) SPI and SPA
    (D) SPD and SPA
    (E) none of the above

30. Assuming a **standard stack convention** (in which the **SP register** points to the **top stack item**), the following **PC control signal** will be asserted on the **last execute cycle** of an **RTS** instruction:
    (A) POA
    (B) POD
    (C) PLA
    (D) PLD
    (E) none of the above

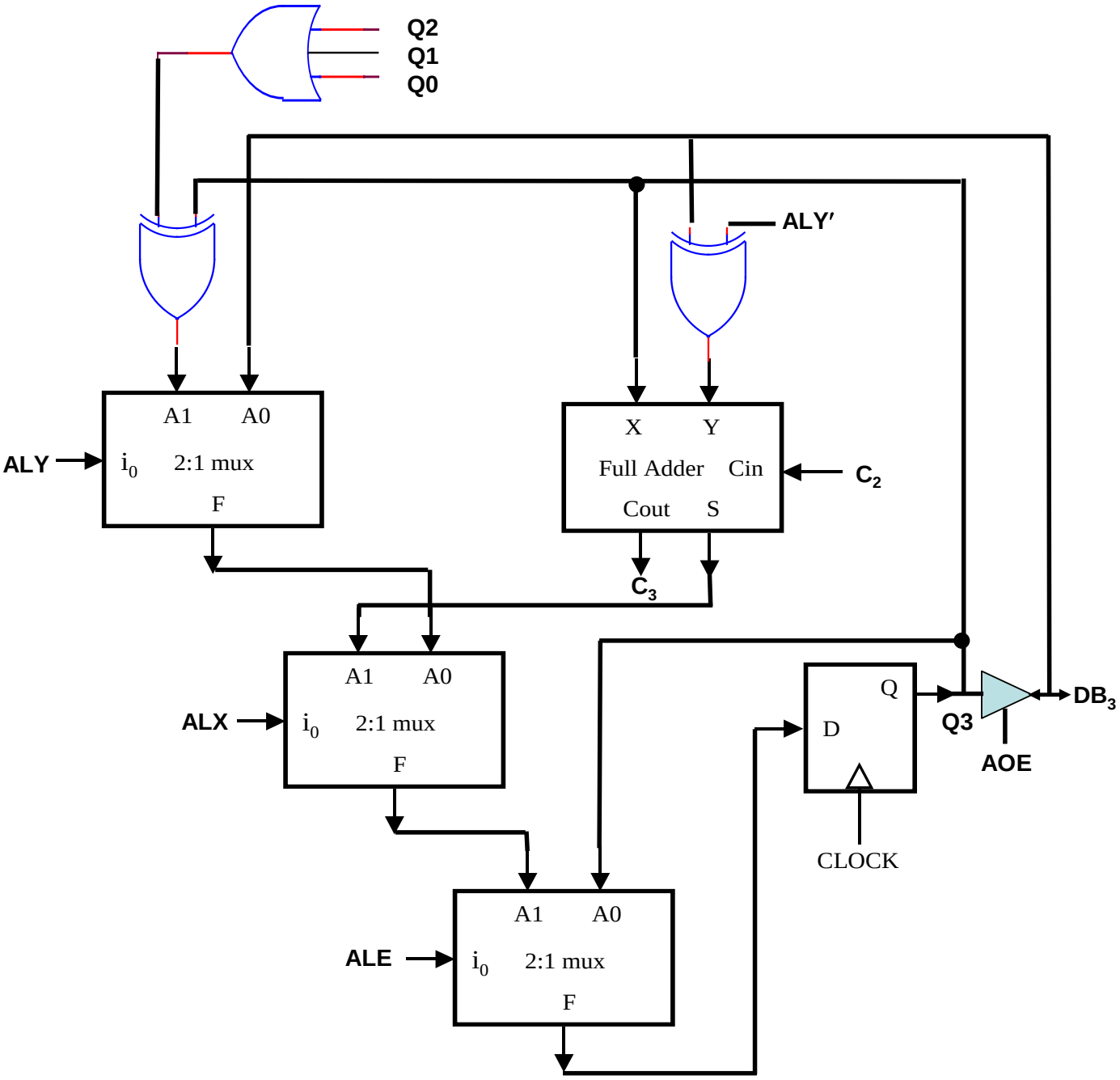


Figure 1. Block Diagram for Bit 3 of a Simple Computer ALU (applies to questions 20 through 22).

Opcode	*Mnemonic*	*Function Performed*	*CC Affected*
0 0 0	ADD *addr*	Add contents of *addr* to contents of A	CF, NF, VF, ZF
0 0 1	SUB *addr*	Subtract contents of *addr* from contents of A	CF, NF, VF, ZF
0 1 0	LDA *addr*	Load A with contents of location *addr*	NF, ZF
0 1 1	XOR *addr*	XOR contents of *addr* with contents of A	NF, ZF
1 0 0	STA *addr*	Store contents of A at location *addr*	none
1 0 1	HLT	Halt – Stop, discontinue execution	none

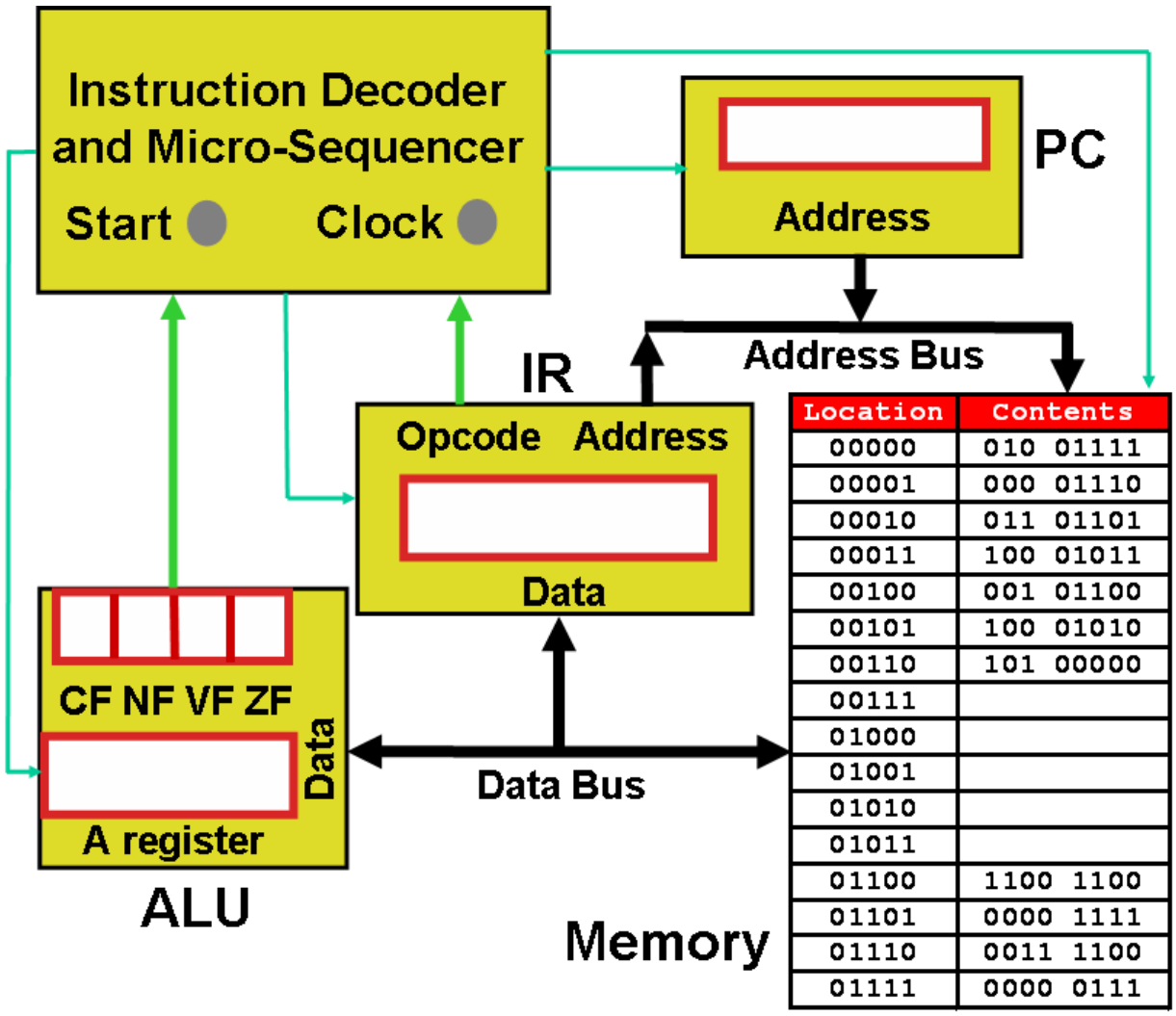


Figure 2. Simple Computer Instruction Set, Block Diagram, and Memory Contents (applies to questions 23 through 25).

Instruction Set:

Opcode	*Mnemonic*	*Description*	*Opcode*	*Mnemonic*	*Description*
0 0 0	HLT	Stop execution	1 0 0	ADD addr	(A) ← (A)+(addr)
0 0 1	LDA addr	(A)←(addr)	1 0 1	SUB addr	(A) ← (A)–(addr)
0 1 0	STA addr	(addr)←(A)	1 1 0	JSR addr	Call subroutine at location addr
0 1 1	ASR	Arithmetic Shift Right (A)	1 1 1	RTS	Return from subroutine

ALU Function Table:

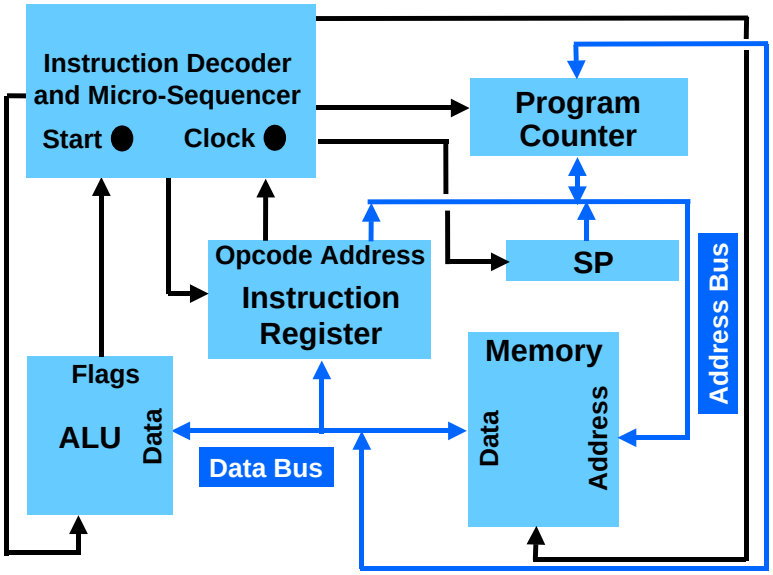
AOE	ALE	ALX	ALY	Function	CF	ZF	NF	VF
0	1	0	0	Add	X	X	X	X
0	1	0	1	Subtract	X	X	X	X
0	1	1	0	Load	•	X	X	•
0	1	1	1	Arithmetic Shift Right*	X	X	X	•
1	0	d	d	Output	•	•	•	•
0	0	d	d	<none>	•	•	•	•

X → flag affected, • → flag not affected
*For Arithmetic Shift Right, CF = bit shifted out of accumulator

Signal Names:

Name	Description
START	Asynchronous Machine Reset
MSL	Memory Select
MOE	Memory Output Tri-State Enable
MWE	Memory Write Enable
PCC	Program Counter Count Enable
POA	Program Counter Output on Address Bus Tri-State Enable
PLA	Program Counter Load from Address Bus Enable
POD	Program Counter Output on Data Bus Tri-State Enable
PLD	Program Counter Load from Data Bus Enable
IRL	Instruction Register Load Enable
IRA	Instruction Register Output on Address Bus Tri-State Enable
AOE	A-register Output on Data Bus Tri-State Enable
ALE	ALU Function Enable
ALX	ALU Function Select Line "X"
ALY	ALU Function Select Line "Y"
SPI	Stack Pointer Increment
SPD	Stack Pointer Decrement
SPA	Stack Pointer Output on Address Bus Tri-State Enable
RST	Synchronous State Counter Reset
RUN	Machine Run Enable

Block Diagram:



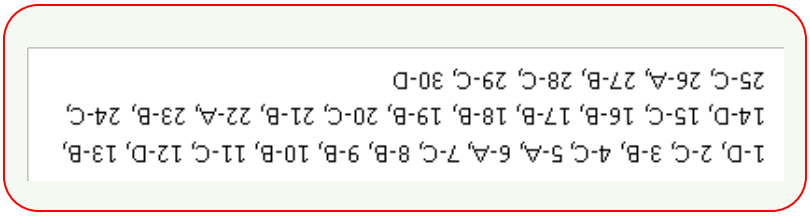


Figure 3. Simple 8-bit Computer Instruction Set, Signal Names, and Block Diagram (applies to questions 26 through 30).

1-D, 2-C, 3-B, 4-C, 5-A, 6-A, 7-C, 8-B, 9-B, 10-B, 11-C, 12-D, 13-B, 14-D, 15-C, 16-B, 17-B, 18-B, 19-B, 20-C, 21-B, 22-A, 23-B, 24-C, 25-C, 26-A, 27-B, 28-C, 29-C, 30-D