OUTCOME #4: "an ability to analyze and design 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.

Place answers on the supplied BUBBLE SHEET only – nothing written here will be graded.

1. Assuming the **CF** condition code bit is **initially cleared,** a **sequence of arithmetic operations** that could be performed on the Raulmatic 716 to verify that **CF** was properly **being set and subsequently cleared** is:
   (A) 1111 – 1110 followed by 1111 + 1110
   (B) 0010 – 0011 followed by 0010 + 0011
   (C) 0010 – 0001 followed by 0010 + 0001
   (D) 0001 – 1110 followed by 0001 + 1110
   (E) none of the above

2. If the **output port pins** of the Raulmatic 716 had **not** been latched**,** data written to the output port **would remain on its pins:**
   (A) until another OUT instruction writes different data to the port
   (B) only during the execute cycle of an OUT instruction
   (C) only while the clock signal is *high* during the execute cycle of an OUT instruction
   (D) until the next instruction is executed
   (E) none of the above

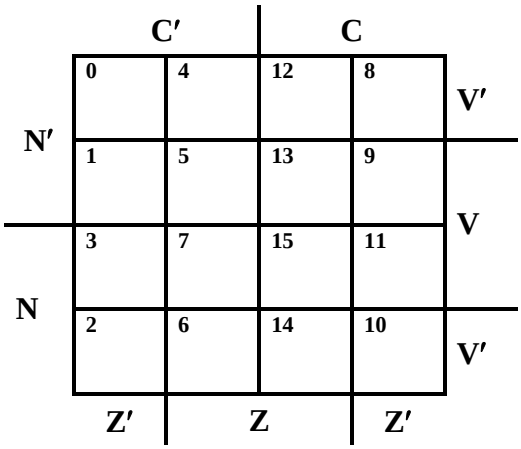
3. If a 4-bit adder/subtractor was realized as shown, the **minimum number of product terms** required to realize the equation for **C[0]** *(when all the nodes are collapsed)* would be:
   (A) 2
   (B) 4
   (C) 6
   (D) 8
   (E) none of the above

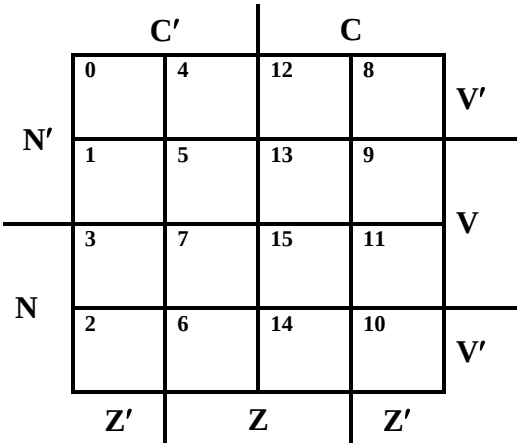
4. If a 4-bit adder/subtractor was realized as shown, the **minimum number of product terms** required to realize the equation for **SD[0]** *(when all the nodes are collapsed)* would be:
   (A) 2
   (B) 4
   (C) 6
   (D) 8
   (E) none of the above

```
module add_sub_4 (X, Y, M, C, SD);

  // 4-bit adder/subtractor
  // M = 0 perform add, M = 1 perform subtract

  input wire [3:0] X, Y;      // operands
  input wire M;               // add/sub mode
  output wire [3:0] C;        // carry
  output wire [3:0] SD;       // sum/difference

  wire [3:0] G, P;

  // CLA Generate functions
  assign G = X & (Y ^ {4{M}});

  // CLA Propagate functions
  assign P = X ^ (Y ^ {4{M}});

  // CLA Carry function definitions
  assign C[0] = G[0] | M&P[0];
  assign C[1] = G[1] | G[0]&P[1] | M&P[0]&P[1];
  assign C[2] = G[2] | G[1]&P[2] | G[0]&P[1]&P[2]
                     | M&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]
                     | M&P[0]&P[1]&P[2]&P[3];

  // CLA Sum/Difference equations
  assign SD[0] = M ^ P[0];
  assign SD[3:1] = C[2:0] ^ P[3:1];

endmodule
```

The following chart applies to questions 5 and 6:

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

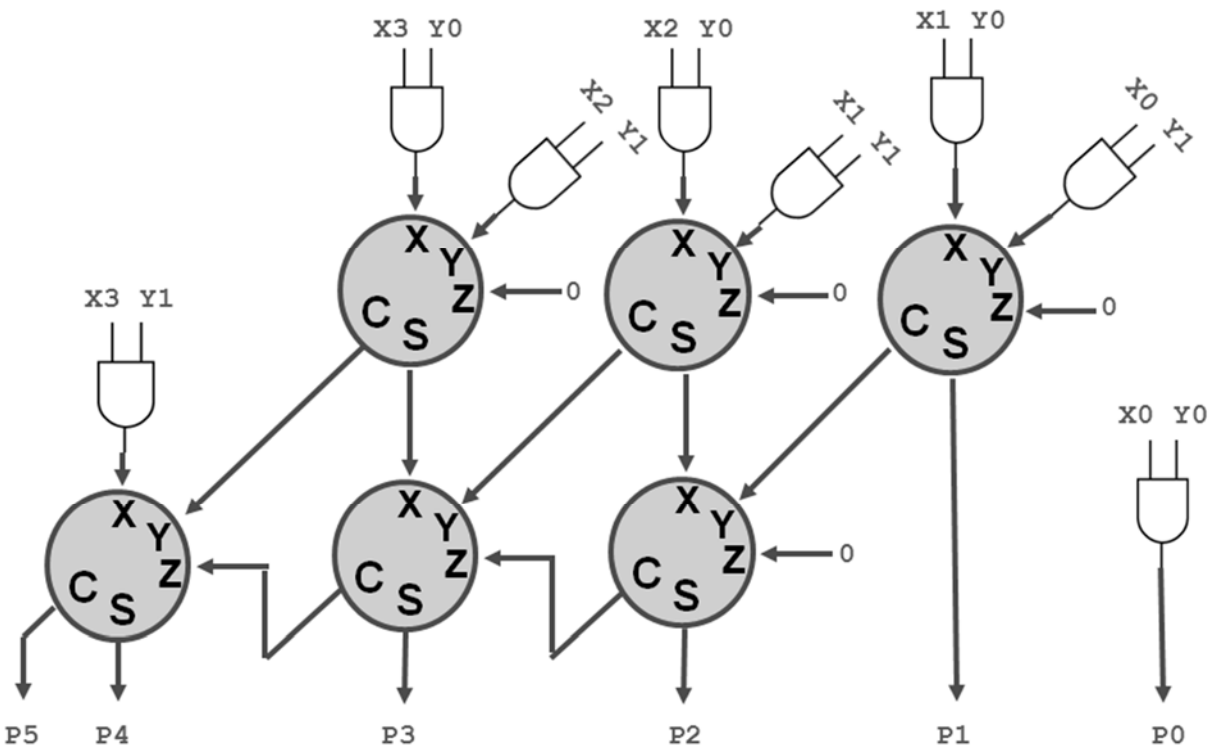




5. The function for "A less than or equal to B" ($F_{A\le B}$) can be expressed as:
   (A) $\mathbf{F_{A\le B} = C \cdot Z'}$
   (B) $\mathbf{F_{A\le B} = C' + Z}$
   (C) $\mathbf{F_{A\le B} = N' \cdot V + N \cdot V'}$
   (D) $\mathbf{F_{A\le B} = N' \cdot V' + N \cdot V}$
   (E) none of the above

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

The following circuit (using full-adder cells) applies to questions 7 through 9:



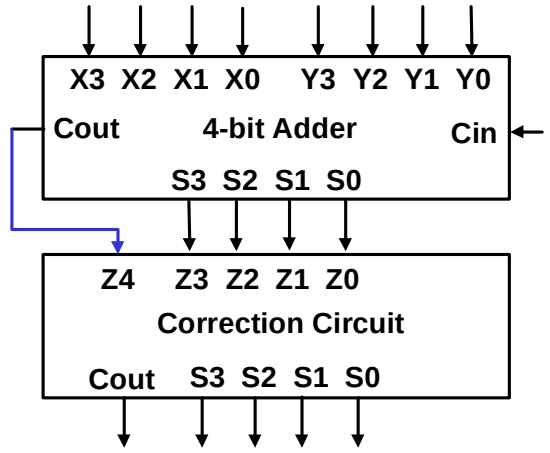
7. The function performed by this circuit is:
   (A) multiply a 2-bit unsigned binary number by a 3-bit number
   (B) multiply a 3-bit unsigned binary number by a 3-bit number
   (C) multiply a 4-bit unsigned binary number by a 2-bit number
   (D) multiply a 5-bit unsigned binary number by a 2-bit number
   (E) none of the above

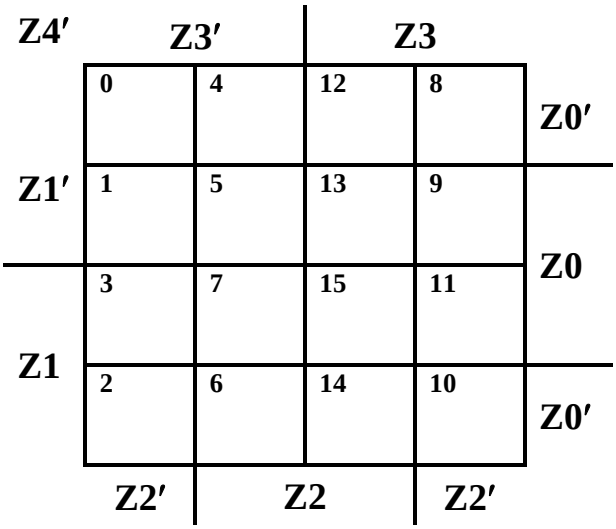
8. If each **AND** gate produces its output in **N** nanoseconds, and each full adder cell produces its **carry (C)** output in **2N** nanoseconds and its **sum (S)** output in **3N** nanoseconds, then the **worst case propagation delay** for the entire circuit (in nanoseconds) will be:

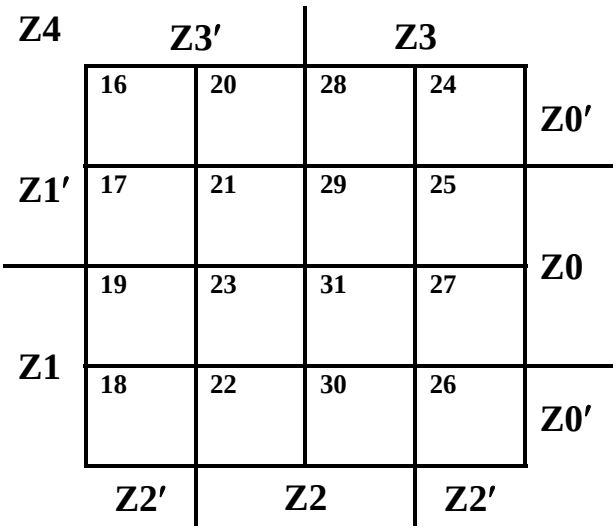
   (A) 10 N (B) 11 N (C) 12 N (D) 13 N (E) none of these

9. If all of the Xi and Yi inputs are set to 1, the output $\mathbf{P_5P_4P_3P_2P_1P_0}$ produced will be:
   (A) **010010**
   (B) **000110**
   (C) **101101**
   (D) **011110**
   (E) none of the above

The following block diagram for a BCD full adder applies to questions 10 and 11:



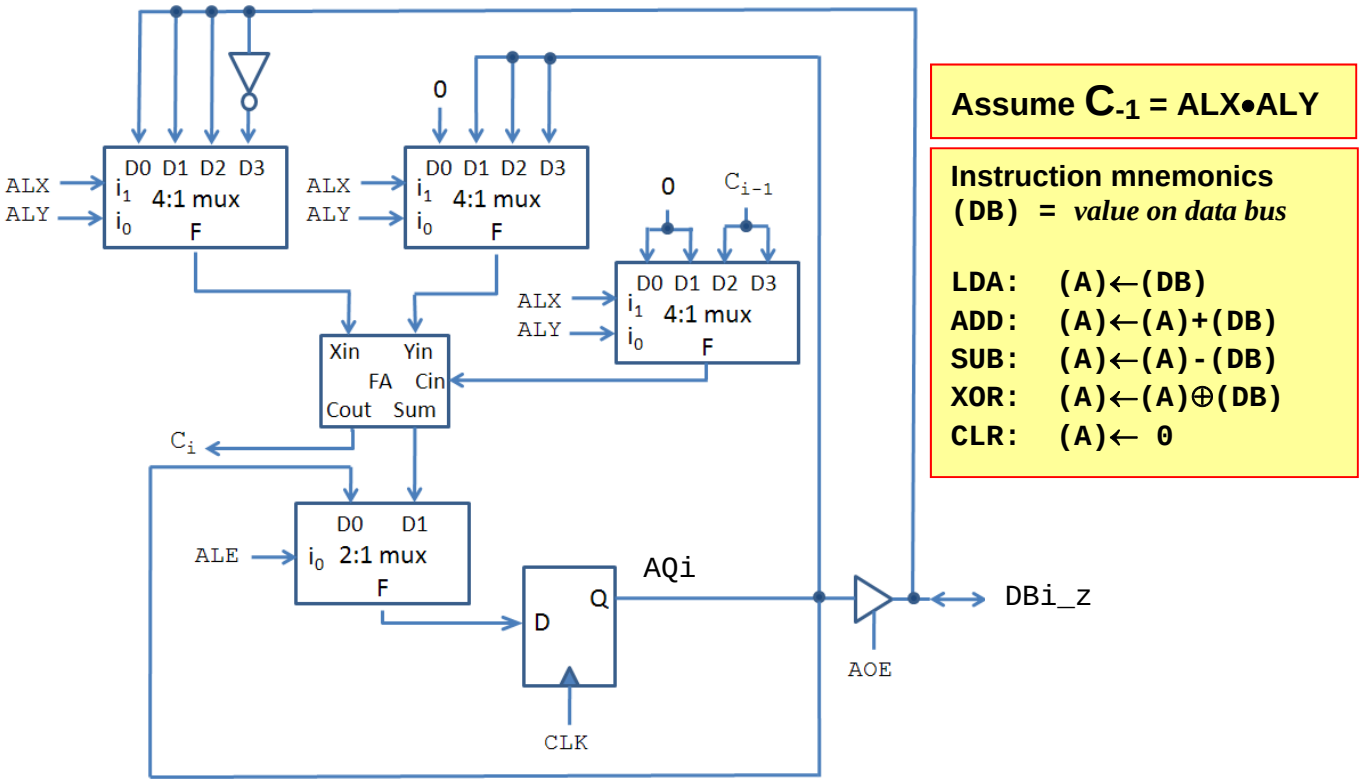




10. **Correction circuit** output **Cout** is equal to:
    (A) **Z4′ + Z3•Z2 + Z3•Z1**
    (B) **Z4 + Z3•Z2 + Z3•Z1**
    (C) **Z4 + Z3′•Z2′ + Z3•Z1**
    (D) **Z4 + Z3•Z2 + Z3′•Z1′**
    (E) none of the above

11. If **X[3:0] = 0111, Y[3:0] = 0110, and Cin = 1,** the value **output by** the correction circuit **{Cout S[3:0]}** will be:
    (A) **0 1 0 1 0**
    (B) **0 1 0 1 1**
    (C) **1 0 0 0 0**
    (D) **1 0 1 0 0**
    (E) none of the above

The following figure applies to questions 12 through 15. It represents a single bit "i" of an n-bit ALU. Assume the least significant bit carry in **(C_{-1})** is set equal to **ALX•ALY.**



Assume C_{-1} = ALX•ALY

Instruction mnemonics
(DB) = ***value on data bus***

```
LDA:  (A)←(DB)
ADD:  (A)←(A)+(DB)
SUB:  (A)←(A)-(DB)
XOR:  (A)←(A)⊕(DB)
CLR:  (A)← 0
```

12. If the input control combination **AOE=0, ALE=1, ALX=0, ALY=0** is applied to this circuit, the function performed will be:

    (A) LDA (B) ADD (C) SUB (D) XOR (E) CLR

13. If the input control combination **AOE=0, ALE=1, ALX=0, ALY=1** is applied to this circuit, the function performed will be:

    (A) LDA (B) ADD (C) SUB (D) XOR (E) CLR

14. If the input control combination **AOE=0, ALE=1, ALX=1, ALY=1** is applied to this circuit, the function performed will be:

    (A) LDA (B) ADD (C) SUB (D) XOR (E) CLR

15. The equation realized by the multiplexer generating the Xin input to the full adder can be expressed as a dataflow assignment in Verilog as:

    (A) `Xin = DBi_z & (~ALX | ~ALY) | ( ALX & ALY & ~DBi_z);`
    (B) `Xin = DBi_z & ( ALX | ~ALY) | (~ALX & ALY & ~DBi_z);`
    (C) `Xin = DBi_z & (~ALX | ALY) | ( ALX & ~ALY & ~DBi_z);`
    (D) `Xin = DBi_z & ( ALX | ALY) | (~ALX & ~ALY & ~DBi_z);`
    (E) none of the above

The following tables apply to questions 16 through 19:

Opcode	*Mnemonic*	*Description*	*Opcode*	*Mnem*	*Description*
0 0 0	HLT	Stop execution	1 0 0	PPA	(A)←(A)+((SP)), (SP)←(SP)+1
0 0 1	LDA addr	(A)←(addr)	1 0 1	PPS	(A)←(A)-((SP)), (SP)←(SP)+1
0 1 0	STA addr	(addr)←(A)	1 1 0	PPX	(A)←(A)⊕((SP)), (SP)←(SP)+1
0 1 1	JMP addr	(PC)← addr	1 1 1	PSH	(SP)←(SP)-1, ((SP))←(A)

Location	*Contents*	*Mnemonic*
00000	001 01101	LDA data1
00001	111 00000	PSH
00010	001 01110	LDA data2
00011	111 00000	PSH
00100	001 01111	LDA data3
00101	111 00000	PSH
00110	110 00000	PPX
00111	010 10000	STA res1
01000	100 00000	PPA
01001	010 10001	STA res2
01010	101 00000	PPS
01011	010 10010	STA res3
01100	000 00000	HLT
01101	1111 1011	(data1)
01110	1011 1111	(data2)
01111	0110 1110	(data3)
10000		(res1)
10001		(res2)
10010		(res3)

Work area for calculations:

16. The value stored at location 10000 (`res1`) will be:
    (A) 0000 0000 (B) 1111 1111 (C) 1011 0110 (D) 1011 1011 (E) none of these

17. The value stored at location 10001 (`res2`) will be:
    (A) 0000 0000 (B) 1111 1111 (C) 1011 0110 (D) 1011 1111 (E) none of these

18. The value stored at location 10010 (`res3`) will be:
    (A) 0000 0000 (B) 1100 0100 (C) 1011 0110 (D) 1011 1011 (E) none of these

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

The following narrative and ALU function table apply to questions 20 through 22:

Assume the ALU function table is modified as shown, in particular the manner in which the CF and VF condition codes are affected by LDA and AND instructions. Note that CF is cleared to 0 when an LDA instruction is executed, and set to 1 if execution of an AND instruction yields a result of 1111. Also note that VF is cleared to 0 when an LDA instruction is executed, but is unaffected by execution of an AND instruction. Recall that `CY[3:0]` are the carries produced by each position, while `ALU[3:0]` are the "next" values loaded in the A register (`AQ[3:0]`) when the ALU is enabled (`ALE=1`).

Assume the equations you are asked to identify below are included in an `always` block with an appropriate sensitivity list.

Modified ALU function table (changes highlighted):

AOE	ALE	ALX	ALY	Function Performed	CF	ZF	NF	VF
0	1	0	0	LDA: AQ[3:0] ← DB_z[3:0]	0	⇳	⇳	0
0	1	0	1	AND: AQ[3:0] ← AQ[3:0] ∩ DB_z[3:0]	⇳*	⇳	⇳	-
0	1	1	0	SUB: AQ[3:0] ← AQ[3:0] – DB_z[3:0]	⇳	⇳	⇳	⇳
0	1	1	1	ADD: AQ[3:0] ← AQ[3:0] + DB_z[3:0]	⇳	⇳	⇳	⇳
1	0	d	d	OUT: DB_z[3:0] ← AQ[3:0]	-	-	-	-
0	0	d	d	(no operation – retain state)	-	-	-	-

* CF = 1 if AND yields result of 1111; else, CF = 0

20. To implement the modification for how the condition code bits are affected, the Verilog equation for **`next_CF`** should be:
    (A) `next_CF = ALE ? (ALX ? ALY&ALU[3]&ALU[2]&ALU[1]&ALU[0] : CY[3]) : CF;`
    (B) `next_CF = ALE ? (ALY ? ALX&ALU[3]&ALU[2]&ALU[1]&ALU[0] : CY[3]) : CF;`
    (C) `next_CF = ALE ? (ALY ? CY[3] : ALX&ALU[3]&ALU[2]&ALU[1]&ALU[0]) : CF;`
    (D) `next_CF = ALE ? (ALX ? CY[3] : ALY&ALU[3]&ALU[2]&ALU[1]&ALU[0]) : CF;`
    (E) none of the above

21. To implement the modification for how the condition code bits are affected, the Verilog equation for **`next_VF`** should be:
    (A) `next_VF = ALE ? ALY&VF : (ALX ? (CY[3] ^ CY[2])) : VF;`
    (B) `next_VF = ALE ? (ALX ? (CY[3] ^ CY[2]) : ALY&VF) : VF;`
    (C) `next_VF = ALE ? ALX&VF : (ALY ? (CY[3] ^ CY[2])) : VF;`
    (D) `next_VF = ALE ? (ALX ? (CY[3] | CY[2]) : ALY&VF) : VF;`
    (E) none of the above

22. The Verilog equation for **`next_AQ`** should be:
    (A) `next_AQ = ALE ? ALU : AQ;`
    (B) `next_AQ = ALE ? AQ : ALU;`
    (C) `next_AQ = ALU ? ALE : AQ;`
    (D) `next_AQ = AQ ? ALE : ALU;`
    (E) none of the above

The Reference Sheet on the page that follows applies to questions 23 through 30:

23. A “load A” (LDA) instruction is performed by the opcode:
    (A) 000 (B) 001 (C) 010 (D) 011 (E) none of these

24. A “store A” (STA) instruction is performed by the opcode:
    (A) 000 (B) 001 (C) 010 (D) 011 (E) none of these

25. The instruction performed by opcode 011 is:
    (A) **PSH** (push the contents of A onto the stack)
    (B) **POP** (pop the top stack item and load it into A)
    (C) **STA** (store the contents of A)
    (D) **HLT** (halt execution)
    (E) none of the above

26. The instruction performed by opcode 100 is:
    (A) **POP** (add the contents of the memory location to A)
    (B) **PPA** (pop the top stack item and add it to A)
    (C) **PPS** (pop the top stack item and subtract it from A)
    (D) **RTS** (return from subroutine)
    (E) none of the above

27. The instruction performed by opcode 110 is:
    (A) **POP** (add the contents of the memory location to A)
    (B) **PPA** (pop the top stack item and add it to A)
    (C) **PPS** (pop the top stack item and subtract it from A)
    (D) **RTS** (return from subroutine)
    (E) none of the above

28. The instruction performed by opcode 111 is:
    (A) **PSH** (push the contents of A onto the stack)
    (B) **POP** (pop the top stack item and load it into A)
    (C) **JSR** (jump to subroutine)
    (D) **RTS** (return from subroutine)
    (E) none of the above

29. The stack convention used by this computer is:
    (A) stack pointer points to top stack item
    (B) stack pointer points to next available location
    (C) stack pointer points to first item pushed on stack
    (D) stack pointer points to last item pushed on stack
    (E) none of the above

30. Changing the stack convention used by this computer (to the “other one”) would:
    (A) increase the number of execute states required by 1
    (B) decrease the number of execute states required by 1
    (C) improve the performance (speed of execution)
    (D) reduce the performance (speed of execution)
    (E) none of the above

Tables and Figures that Apply to Questions 23 through 30

State	Opcode	MSL	MOE	MWE	IRL	IRA	AOE	ALE	ALX	ALY	PCC	POA	PLA	POD	PLD	SPI	SPD	SPA	RST
S0	—	H	H		H						H	H							
S1	000																		
S1	001	H	H			H		H	H										H
S1	010	H		H		H	H												H
S1	011	H		H			H										H	H	H
S1	100															H			
S1	101															H			
S1	110															H			
S1	111	H		H										H			H	H	
S2	100	H	H												H			H	H
S2	101	H	H					H										H	H
S2	110	H	H					H		H								H	H
S2	111					H							H						H

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

AOE	ALE	ALX	ALY	Function	CF	ZF	NF	VF
0	1	0	0	Add	⇳	⇳	⇳	⇳
0	1	0	1	Subtract	⇳	⇳	⇳	⇳
0	1	1	0	Load	•	⇳	⇳	•
1	0	d	d	Output	•	•	•	•
0	0	d	d	<none>	•	•	•	•