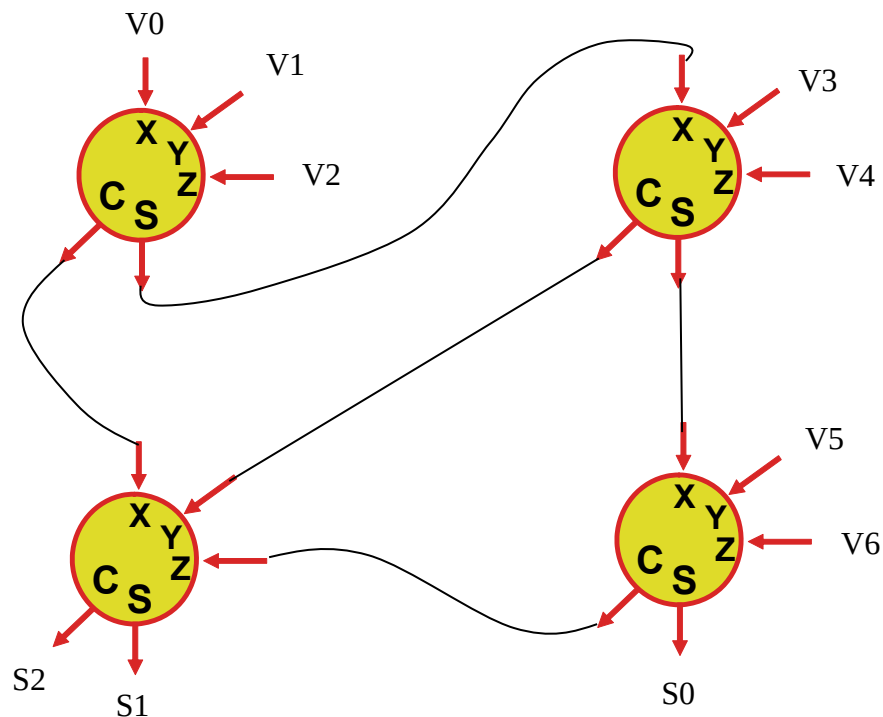


Practice Homework Solution for Module 4

1. Tired of writing the names of those you want “kicked off the island” on cards, you wish to modernize the voting scheme used on *Digital Survivor*. Specifically, you would like to design a circuit that tabulates the “stay on the island”/“kick off the island” (yes/no) votes of up to 7 fellow contestants. Armed with only a bucket of full-adder cells (plus 7 switches, some resistors, 3 LEDs, a breadboard, some wires, a battery, and a large pepperoni pizza), you start your *Amazing Digital Race*. Assume that the tribunal can correctly interpret 3-bit binary numbers.

HINT: This is sometimes called a “population counting” circuit – here we simply want to tabulate the number of inputs that are “1”.



2. Signed conversions:

$$R(010001)_2 \rightarrow SM(010001)_2 \rightarrow DR(010001)_2$$

$$DR(101110)_2 \rightarrow R(101111)_2 \rightarrow SM(110001)_2$$

$$SM(101110)_2 \rightarrow DR(110001)_2 \rightarrow R(110010)_2$$

3. Radix addition (base 2)

$$\begin{array}{r} 1\ 0\ 0\ 1\ 1 \\ +\ 0\ 1\ 1\ 1\ 1 \\ \hline 0\ 0\ 0\ 1\ 0 \end{array}$$

$$\begin{array}{r} 1\ 0\ 0\ 1\ 1 \\ +\ 1\ 0\ 0\ 1\ 1 \\ \hline 0\ 0\ 1\ 1\ 0 \quad \text{OVF} \end{array}$$

$$\begin{array}{r} 0\ 1\ 1\ 1\ 1 \\ +\ 1\ 0\ 0\ 0\ 0 \\ \hline 1\ 1\ 1\ 1\ 1 \end{array}$$

$$\begin{array}{r} 0\ 1\ 1\ 1\ 1 \\ +\ 0\ 1\ 0\ 1\ 0 \\ \hline 1\ 1\ 0\ 0\ 1 \quad \text{OVF} \end{array}$$

4. Radix subtraction (base 2)

$$\begin{array}{r} 1\ 0\ 0\ 0\ 1 \\ -\ 1\ 1\ 1\ 1\ 1 \\ \hline 1\ 0\ 0\ 1\ 0 \end{array}$$

$$\begin{array}{r} 1\ 0\ 0\ 0\ 0 \\ -\ 0\ 1\ 0\ 1\ 1 \\ \hline 0\ 0\ 1\ 0\ 1 \quad \text{OVF} \end{array}$$

$$\begin{array}{r} 0\ 1\ 0\ 1\ 1 \\ -\ 0\ 0\ 0\ 0\ 0 \\ \hline 0\ 1\ 0\ 1\ 1 \end{array}$$

$$\begin{array}{r} 0\ 1\ 0\ 1\ 1 \\ -\ 1\ 0\ 0\ 0\ 1 \\ \hline 1\ 1\ 0\ 1\ 0 \quad \text{OVF} \end{array}$$

5. Determine how the condition codes (C, Z, N, V) are set for 2-bit Radix subtraction.
Show all work.

$\begin{array}{r} 1\ 0 \\ - 0\ 1 \\ \hline 0\ 1 \end{array}$ C = 1 Z = 0 N = 0 V = 1	$\begin{array}{r} 1\ 0 \\ - 1\ 0 \\ \hline 0\ 0 \end{array}$ C = 1 Z = 1 N = 0 V = 0	$\begin{array}{r} 1\ 0 \\ - 1\ 1 \\ \hline 1\ 1 \end{array}$ C = 0 Z = 0 N = 1 V = 0
$\begin{array}{r} 0\ 1 \\ - 0\ 1 \\ \hline 0\ 0 \end{array}$ C = 1 Z = 1 N = 0 V = 0	$\begin{array}{r} 0\ 1 \\ - 1\ 0 \\ \hline 1\ 1 \end{array}$ C = 0 Z = 0 N = 1 V = 1	$\begin{array}{r} 0\ 1 \\ - 1\ 1 \\ \hline 1\ 0 \end{array}$ C = 0 Z = 0 N = 1 V = 1
$\begin{array}{r} 0\ 0 \\ - 0\ 1 \\ \hline 1\ 1 \end{array}$ C = 0 Z = 0 N = 1 V = 0	$\begin{array}{r} 0\ 0 \\ - 1\ 0 \\ \hline 1\ 0 \end{array}$ C = 0 Z = 0 N = 1 V = 1	$\begin{array}{r} 0\ 0 \\ - 1\ 1 \\ \hline 0\ 1 \end{array}$ C = 0 Z = 0 N = 0 V = 0

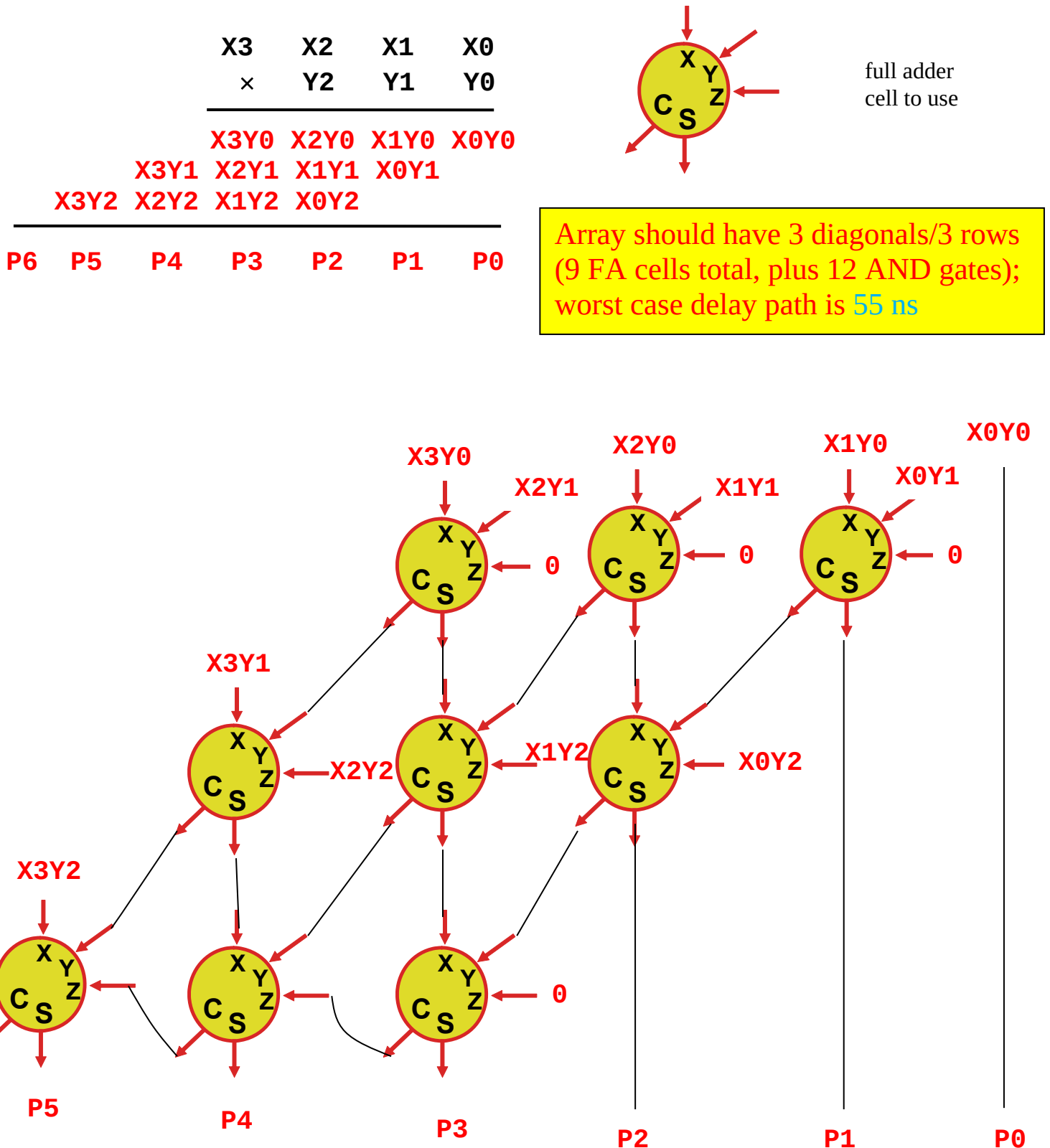
6. Complete the missing entries in the “condition code generation chart,” below, and derive the function for “A greater than or equal to B” (“AGEB”) in its simplest (minimal) form. **Show all work.**

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

	C'		C		
N'	0	4	12	8	V'
	1	d	1	1	
N	1	5	13	9	V
	d	d	d	0	
N	3	7	15	11	V'
	1	d	d	d	
N	2	6	14	10	V'
	0	d	d	0	
	Z'	Z	Z'		

$$\text{AGEB} = N' \cdot V' + N \cdot V$$

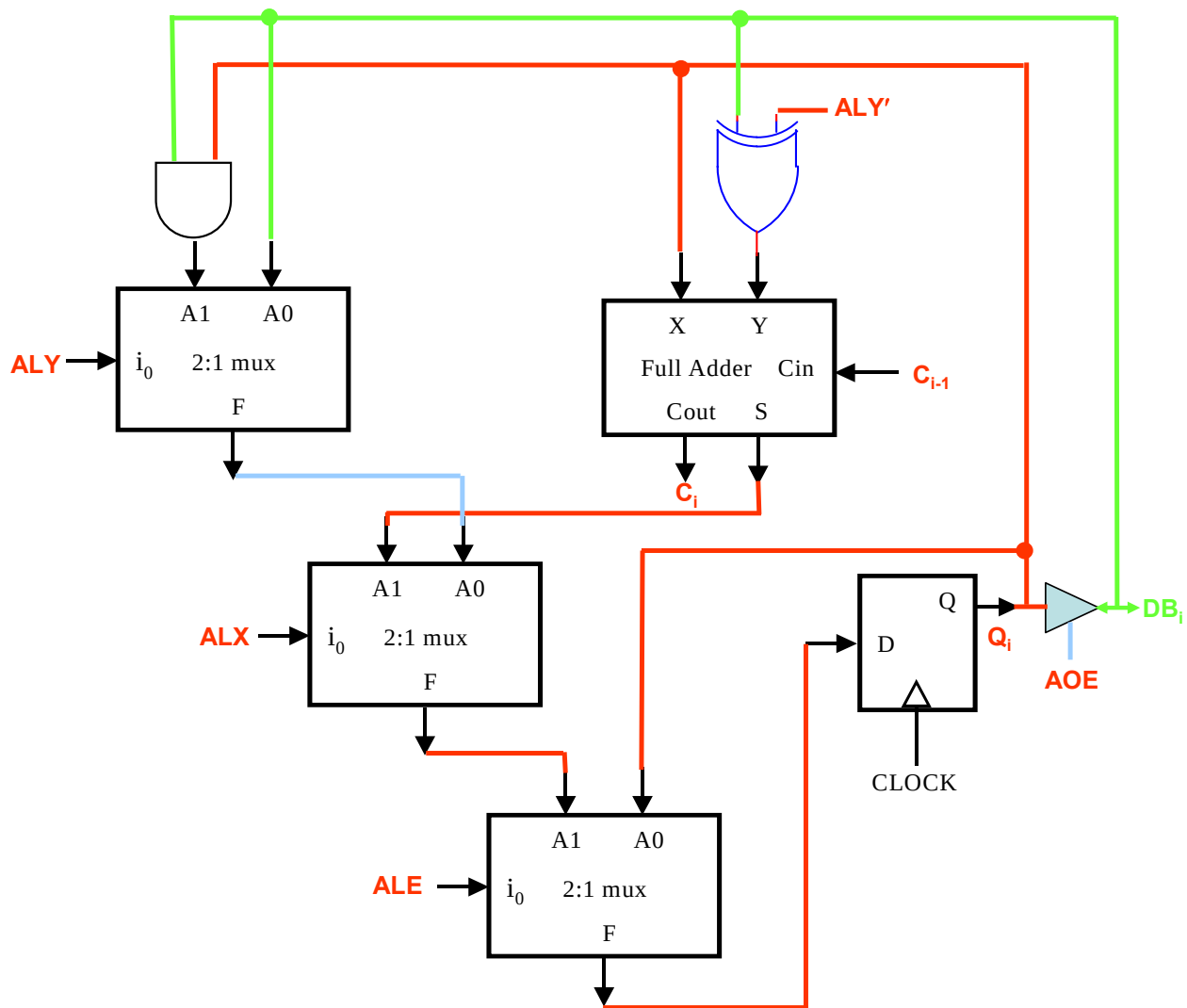
7. Draw a circuit that multiplies a 4-bit unsigned binary number $X_3 X_2 X_1 X_0$ by a 3-bit unsigned binary number $Y_2 Y_1 Y_0$, using an array of full-adder cells. Determine the worst case propagation delay if each full adder takes **10 ns** to produce its C and S outputs, and each AND gate (used to generate the product components) has **5 ns** of propagation delay.



8. Using the parts provided plus any additional logic gates you deem necessary, complete the **block diagram** for **one bit (“i”) of the ALU** defined as follows:

AOE	ALE	ALX	ALY	Function Performed	CF	ZF	NF	VF
0	1	0	0	LDA : $[Q3..Q0] \leftarrow [D3..D0]$	•	↕	↕	•
0	1	0	1	AND : $[Q3..Q0] \leftarrow [Q3..Q0] \cap [D3..D0]$	•	↕	↕	•
0	1	1	0	SUB : $[Q3..Q0] \leftarrow [Q3..Q0] - [D3..D0]$	↕	↕	↕	↕
0	1	1	1	ADD : $[Q3..Q0] \leftarrow [Q3..Q0] + [D3..D0]$	↕	↕	↕	↕
1	0	d	d	OUT : $[D3..D0] \leftarrow [Q3..Q0]$	•	•	•	•
0	0	d	d	(no operation – retain state)	•	•	•	•

“↕” indicates the flag is affected by the function performed, “•” indicates the flag is not affected



9. Given the “snapshot” of memory shown, calculate the result stored in memory along with the condition codes generated when each “shaded” region of machine code is executed.

Opcode	Mnemonic	Description	Opcode	Mnemonic	Description
0 0 0	HLT	Stop execution	0 1 1	NGA	$(A) \leftarrow -(A)$
0 0 1	LDA addr	$(A) \leftarrow (\text{addr})$	1 0 0	SUB addr	$(A) \leftarrow (A) - (\text{addr})$
0 1 0	STA addr	$(\text{addr}) \leftarrow (A)$	1 0 1	ADD addr	$(A) \leftarrow (A) + (\text{addr})$

INITIAL CONTENTS

Location	Contents
00000	001 01101
00001	011 00000
00010	010 01100
00011	001 01110
00100	100 01111
00101	010 01011
00110	001 01111
00111	101 01110
01000	010 01010
01001	000 00000
01010	
01011	
01100	
01101	0111 1011
01110	1011 0110
01111	0110 1110

CF = 0

NF = 0

VF = 0

ZF = 0

AFTER 1st BLOCK

Location	Contents
00000	001 01101
00001	011 00000
00010	010 01100
00011	001 01110
00100	100 01111
00101	010 01011
00110	001 01111
00111	101 01110
01000	010 01010
01001	000 00000
01010	
01011	
01100	1000 0101
01101	0111 1011
01110	1011 0110
01111	0110 1110

CF = 0

NF = 1

VF = 0

ZF = 0

AFTER 2nd BLOCK

Location	Contents
00000	001 01101
00001	011 00000
00010	010 01100
00011	001 01110
00100	100 01111
00101	010 01011
00110	001 01111
00111	101 01110
01000	010 01010
01001	000 00000
01010	
01011	0100 1000
01100	1000 0101
01101	0111 1011
01110	1011 0110
01111	0110 1110

CF = 1

NF = 0

VF = 1

ZF = 0

AFTER 3rd BLOCK

Location	Contents
00000	001 01101
00001	011 00000
00010	010 01100
00011	001 01110
00100	100 01111
00101	010 01011
00110	001 01111
00111	101 01110
01000	010 01010
01001	000 00000
01010	0010 0100
01011	0100 1000
01100	1000 0101
01101	0111 1011
01110	1011 0110
01111	0110 1110

CF = 1

NF = 0

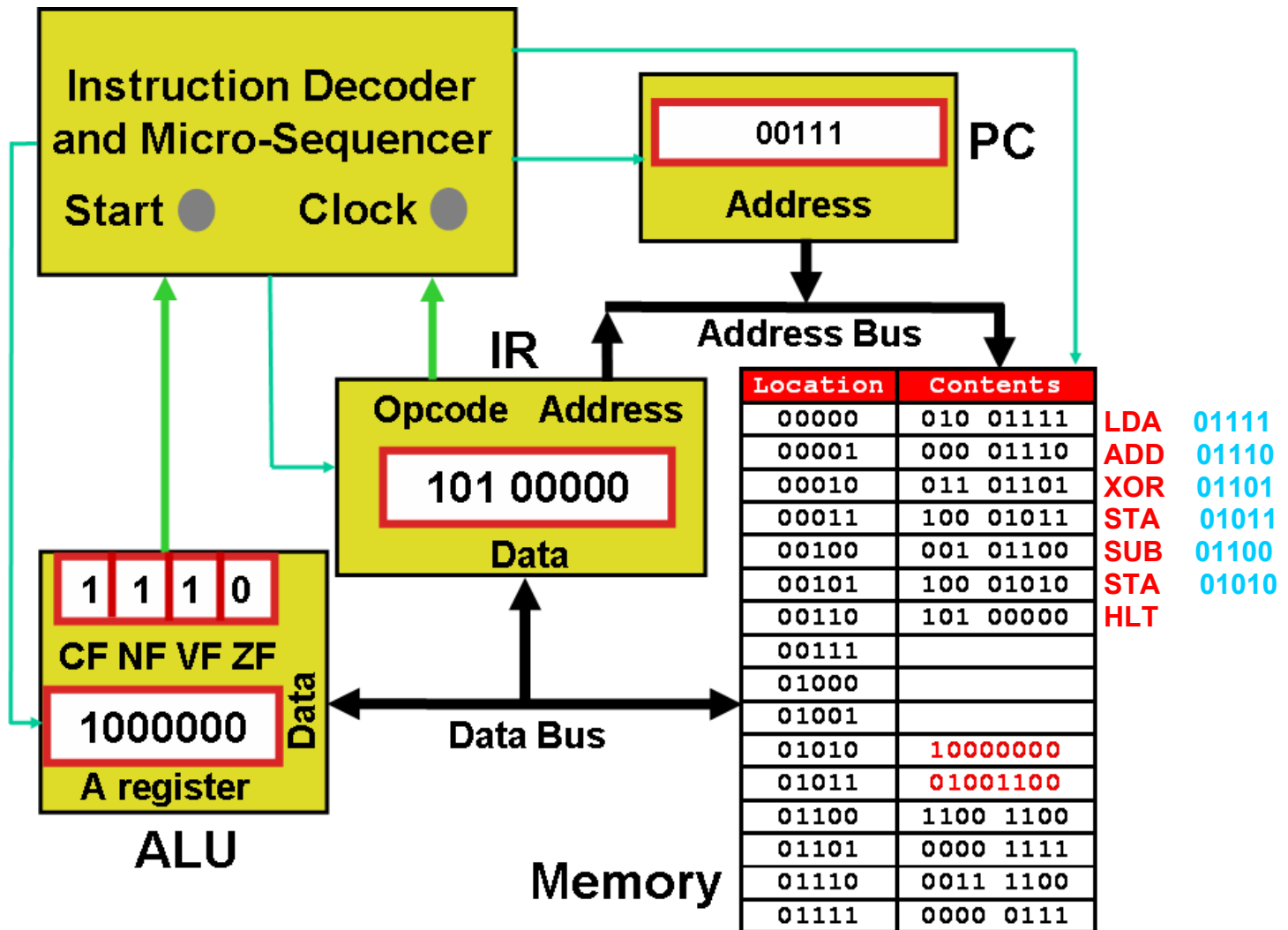
VF = 0

ZF = 0

10. Assume the simple computer instruction set has been changed to the following:

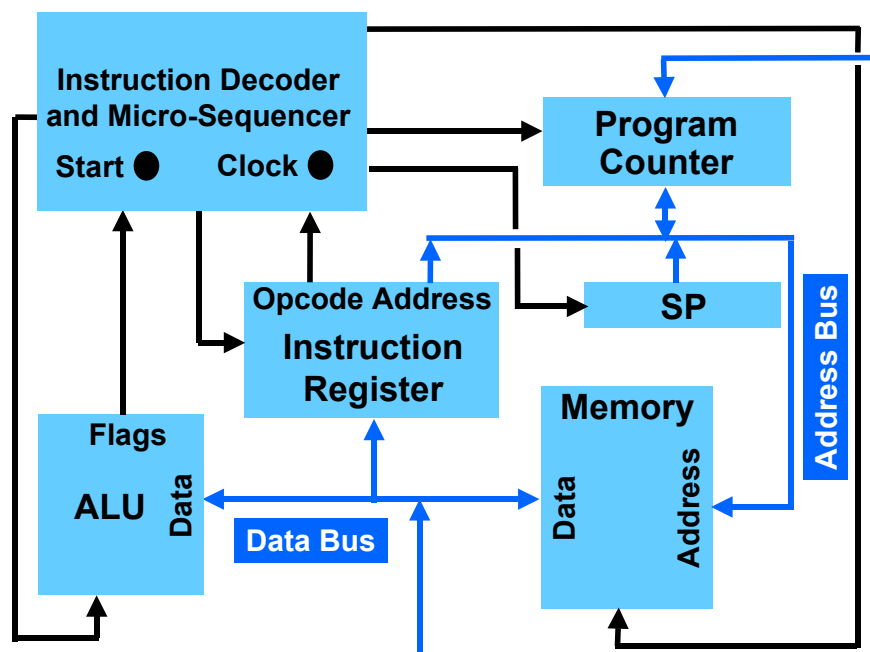
Opcode	Mnemonic	Function Performed
0 0 0	ADD <i>addr</i>	Add contents of <i>addr</i> to contents of A
0 0 1	SUB <i>addr</i>	Subtract contents of <i>addr</i> from contents of A
0 1 0	LDA <i>addr</i>	Load A with contents of location <i>addr</i>
0 1 1	XOR <i>addr</i>	XOR contents of <i>addr</i> with contents of A
1 0 0	STA <i>addr</i>	Store contents of A at location <i>addr</i>
1 0 1	HLT	Halt – Stop, discontinue execution

On the instruction trace worksheet, below, show the *final result* of executing the program stored in memory *up to and including* the HLT instruction.



11. List the signals asserted on each cycle for the Simple Computer described on the *Reference Sheet*. Assume that the stack pointer points to the next available location.

Decoded State	Instruction Mnemonic	Signals Asserted on Each Cycle
S0	Fetch	POA, MSL, MOE, IRL, PCC
S1	LDA	IRA, MSL, MOE, ALE, ALX, RST
S1	STA	IRA, MSL, MWE, AOE, RST
S1	ASR	ALE, ALX, ALY, RST
S1	ADD	IRA, MSL, MOE, ALE, RST
S1	SUB	IRA, MSL, MOE, ALE, ALY, RST
S1	PSH	SPA, MSL, MWE, AOE, SPD, RST
S1	POP	SPI
S2	PSH	□
S2	POP	SPA, MSL, MOE, ALE, ALX, RST



Simple Computer Reference Sheet for Problem #15

Simple Computer Instruction Set:

Opcode	Mnemonic	Description	Opcode	Mnemonic	Description
0 0 0	HLT	Stop execution	1 0 0	ADD addr	$(A) \leftarrow (A) + (\text{addr})$
0 0 1	LDA addr	$(A) \leftarrow (\text{addr})$	1 0 1	SUB addr	$(A) \leftarrow (A) - (\text{addr})$
0 1 0	STA addr	$(\text{addr}) \leftarrow (A)$	1 1 0	PSH	Push (A) on to stack
0 1 1	ASR	Arithmetic Shift Right	1 1 1	POP	Pop (A) off of stack

Simple Computer ALU Function Table:

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

↕ → flag affected, • → flag not affected

*For Arithmetic Shift Right, CF = bit shifted out of accumulator

Simple Computer 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

12. Show how the system control table for the JSR and RTS instructions would *change* for the Simple Computer in the notes if the alternate stack convention (where SP points to *next available location*) were used. Use the *minimum number of execute states possible* for each instruction.

Dec. State	Instr. Mnem.	MSL	MOE	MWE	PCC	POA	IRL	IRA	AOE	ALE	ALX	ALY	PLA	POD	PLD	SPI	SPD	SPA	RST
S0	—	H	H		H	H	H												
S1	LDA	H	H					H		H	H								H
S1	STA	H		H				H	H										H
S1	ADD	H	H					H		H									H
S1	SUB	H	H					H		H		H							H
S1	AND	H	H					H		H	H	H							H
S1	HLT	L			L		L			L									
S1	JSR	H		H										H			H	H	
S1	RTS															H			
S2	JSR							H					H						H
S2	RTS	H	H												H			H	H
S3	JSR																		
S3	RTS																		