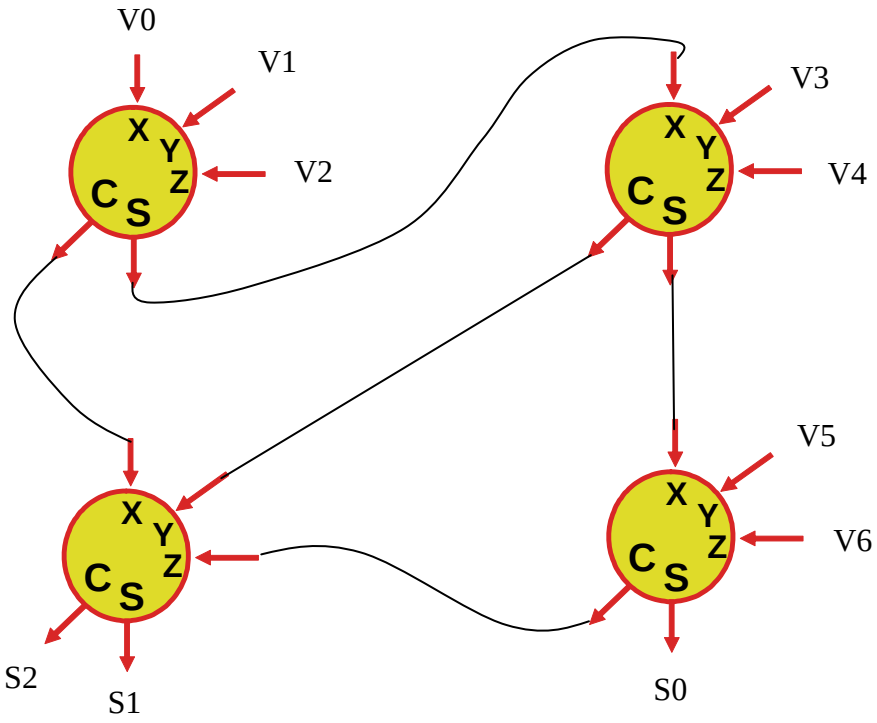
Practice Homework Solution for Module 4

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)

```
  1 0 0 1 1
+ 0 1 1 1 1
-----------
  0 0 0 1 0
```

```
  1 0 0 1 1
+ 1 0 0 1 1
-----------
  0 0 1 1 0   OVF
```

```
  0 1 1 1 1
+ 1 0 0 0 0
-----------
  1 1 1 1 1
```

```
  0 1 1 1 1
+ 0 1 0 1 0
-----------
  1 1 0 0 1   OVF
```

4. Radix subtraction (base 2)

```
  1 0 0 0 1
- 1 1 1 1 1
-----------
  1 0 0 1 0
```

```
  1 0 0 0 0
- 0 1 0 1 1
-----------
  0 0 1 0 1   OVF
```

```
  0 1 0 1 1
- 0 0 0 0 0
-----------
  0 1 0 1 1
```

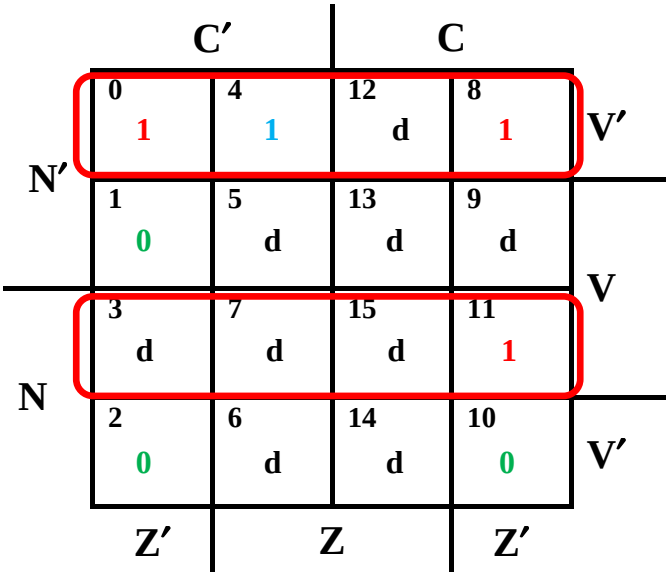
```
  0 1 0 1 1
- 1 0 0 0 1
-----------
  1 1 0 1 0   OVF
```

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

1 0 - 0 1 _____ 0 1 C = 0 Z = 0 N = 0 V = 1	1 0 - 1 0 _____ 0 0 C = 0 Z = 1 N = 0 V = 0	1 0 - 1 1 _____ 1 1 C = 1 Z = 0 N = 1 V = 0
0 1 - 0 1 _____ 0 0 C = 0 Z = 1 N = 0 V = 0	0 1 - 1 0 _____ 1 1 C = 1 Z = 0 N = 1 V = 1	0 1 - 1 1 _____ 1 0 C = 1 Z = 0 N = 1 V = 1
0 0 - 0 1 _____ 1 1 C = 1 Z = 0 N = 1 V = 0	0 0 - 1 0 _____ 1 0 C = 1 Z = 0 N = 1 V = 1	0 0 - 1 1 _____ 0 1 C = 1 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)	0	1	0	0
0	0	0	0	1	+1	(A) < (B)	1	0	1	0
0	0	0	1	0	-2	(A) > (B)	1	0	1	1
0	0	0	1	1	-1	(A) > (B)	1	0	0	0
0	1	+1	0	0	0	(A) > (B)	0	0	0	0
0	1	+1	0	1	+1	(A) = (B)	0	1	0	0
0	1	+1	1	0	-2	(A) > (B)	1	0	1	1
0	1	+1	1	1	-1	(A) > (B)	1	0	1	1
1	0	-2	0	0	0	(A) < (B)	0	0	1	0
1	0	-2	0	1	+1	(A) < (B)	0	0	0	1
1	0	-2	1	0	-2	(A) = (B)	0	1	0	0
1	0	-2	1	1	-1	(A) < (B)	1	0	1	0
1	1	-1	0	0	0	(A) < (B)	0	0	1	0
1	1	-1	0	1	+1	(A) < (B)	0	0	1	0
1	1	-1	1	0	-2	(A) > (B)	0	0	0	0
1	1	-1	1	1	-1	(A) = (B)	0	1	0	0



AGEB = N′•V′ + N•V

7. Compile the two ABEL programs (listed below) using ispLever in order to determine the *minimum number* of P-terms required for the equations listed**.** Also, run the timing simulator (*Performance Analyst*) to determine the worst case propagation delay for each program when targeted for the same CPLD used in lab (LC4256ZE-5TN144C).

```
MODULE cla4_v1
TITLE '4-bit Carry Look-Ahead Adder – V1'

DECLARATIONS
X0..X3, Y0..Y3 pin;  " operands
CIN pin;  " carry in
S0..S3 pin istype 'com';  " sum outputs
G0 = X0&Y0;  " generate function definitions
G1 = X1&Y1;
G2 = X2&Y2;
G3 = X3&Y3;
P0 = X0$Y0;  " propagate function definitions
P1 = X1$Y1;
P2 = X2$Y2;
P3 = X3$Y3;
C0 = G0 # CIN&P0; " carry function definitions
C1 = G1 # G0&P1 # CIN&P0&P1;
C2 = G2 # G1&P2 # G0&P1&P2 # CIN&P0&P1&P2;
C3 = G3 # G2&P3 # G1&P2&P3 # G0&P1&P2&P3 # CIN&P0&P1&P2&P3;

EQUATIONS
S0 = CIN$P0;
S1 = C0$P1;
S2 = C1$P2;
S3 = C2$P3;

END
```

Worst case propagation delay:

Variable: S3

Delay: 6.40 ns

Prefit / Postfit

Number of P-terms required to implement equation for S0 = 4 / 4

Number of P-terms required to implement equation for S1 = 12 / 7

Number of P-terms required to implement equation for S2 = 28 / 15

Number of P-terms required to implement equation for S3 = 60 / 31

```
MODULE cla4_v2
TITLE '4-bit Carry Look-Ahead Adder – V2'

DECLARATIONS
X0..X3, Y0..Y3 pin;  " operands
CIN pin;  " carry in
C0..C3 pin istype 'com';  " carry functions
S0..S3 pin istype 'com';  " sum outputs
G0 = X0&Y0;  " generate function definitions
G1 = X1&Y1;
G2 = X2&Y2;
G3 = X3&Y3;
P0 = X0$Y0;  " propagate function definitions
P1 = X1$Y1;
P2 = X2$Y2;
P3 = X3$Y3;

EQUATIONS
C0 = G0 # CIN&P0;
C1 = G1 # G0&P1 # CIN&P0&P1;
C2 = G2 # G1&P2 # G0&P1&P2 # CIN&P0&P1&P2;
C3 = G3 # G2&P3 # G1&P2&P3 # G0&P1&P2&P3 # CIN&P0&P1&P2&P3;

S0 = CIN$P0;
S1 = C0$P1;
S2 = C1$P2;
S3 = C2$P3;

END
```

Worst case propagation delay:

Variable: S1, S2, S3

Delay: 10.05 ns

Prefit / Postfit

Number of P-terms required to implement equation for C0 = 3 / 3

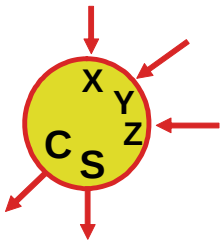
Number of P-terms required to implement equation for C1 = 7 / 6

Number of P-terms required to implement equation for C2 = 15 / 10

Number of P-terms required to implement equation for C3 = 31 / 31

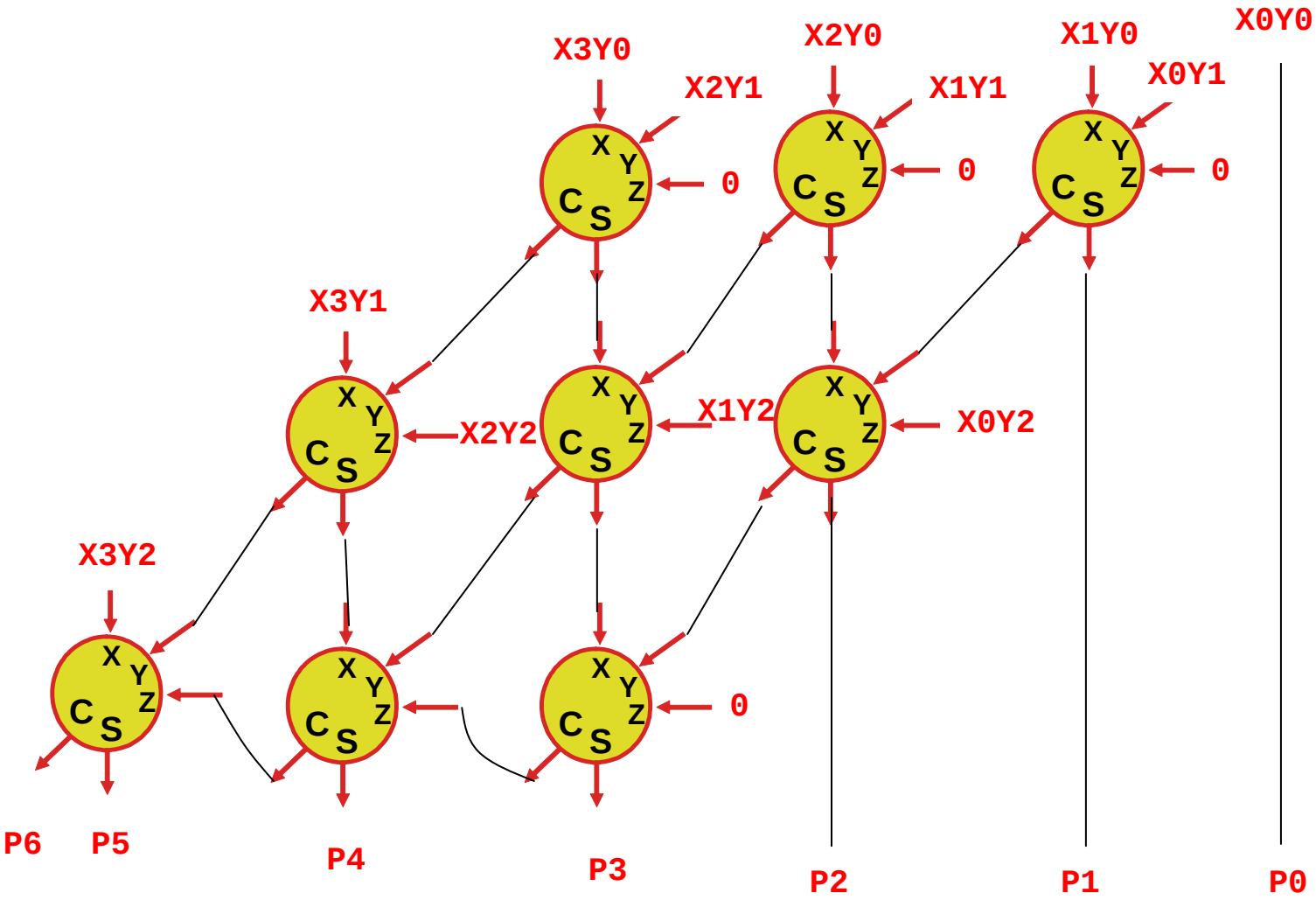
8. Draw a circuit that multiplies a 4-bit unsigned binary number X3 X2 X1 X0 by a 3-bit unsigned binary number Y2 Y1 Y0, using an array of full-adder cells. Determine the worst case propagation delay if each full adder takes 10 ns to produce its C and S outputs, and each AND gate (used to generate the product components) has 5 ns of propagation delay.

```
                X3    X2    X1    X0
              ×       Y2    Y1    Y0
        ------------------------------
                    X3Y0  X2Y0  X1Y0  X0Y0
              X3Y1  X2Y1  X1Y1  X0Y1
        X3Y2  X2Y2  X1Y2  X0Y2
        ------------------------------
P6  P5    P4    P3    P2    P1    P0
```



full adder cell to use

Array should have 3 diagonals/3 rows (9 FA cells total, plus 12 AND gates); worst case delay path is 55 ns



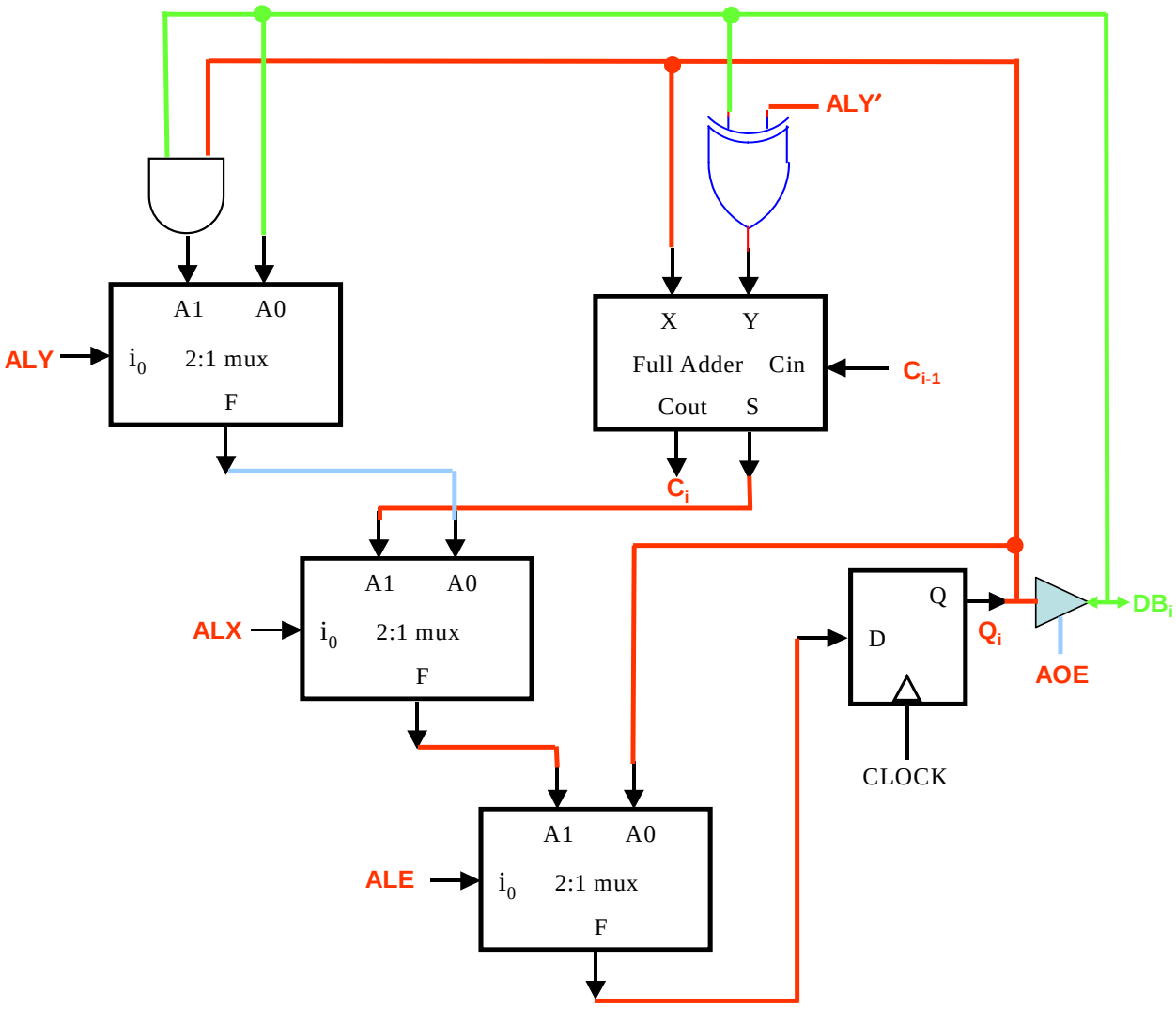
9. Write an ABEL program that multiplies a 4-bit unsigned binary number X3 X2 X1 X0 by a 3-bit unsigned binary number Y2 Y1 Y0. Attach a printed copy of your program along with a synopsis of the worst case timing analysis as determined by ispLever's Performance Analyst.

```
MODULE mul4x3
TITLE '4x3 Combinational Multiplier'

DECLARATIONS
X3..X0, Y2..Y0 pin;  " multiplicand, multiplier
P7..P0 pin istype 'com';  " product bits
P = [P6..P0];

" Definition of product components
PC1 = Y0 & [ 0, 0, 0,X3,X2,X1,X0];
PC2 = Y1 & [ 0, 0,X3,X2,X1,X0, 0];
PC3 = Y2 & [ 0,X3,X2,X1,X0, 0, 0];

EQUATIONS
" Form unsigned sum of product components
P = PC1 + PC2 + PC3;
END


Worst case propagation delay for LC4256ZE-5TN144C is 6.05 ns
```

10. 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] ← [D3..D0]	•	⇳	⇳	•
0	1	0	1	**AND:** [Q3..Q0] ← [Q3..Q0] ∩ [D3..D0]	•	⇳	⇳	•
0	1	1	0	**SUB:** [Q3..Q0] ← [Q3..Q0] – [D3..D0]	⇳	⇳	⇳	⇳
0	1	1	1	**ADD:** [Q3..Q0] ← [Q3..Q0] + [D3..D0]	⇳	⇳	⇳	⇳
1	0	d	d	**OUT:** [D3..D0] ← [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



11. 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) ← -(A)
0 0 1	LDA addr	(A)←(addr)	1 0 0	SUB addr	(A) ← (A)–(addr)
0 1 0	STA addr	(addr)←(A)	1 0 1	ADD addr	(A) ← (A)+(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 = 0

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

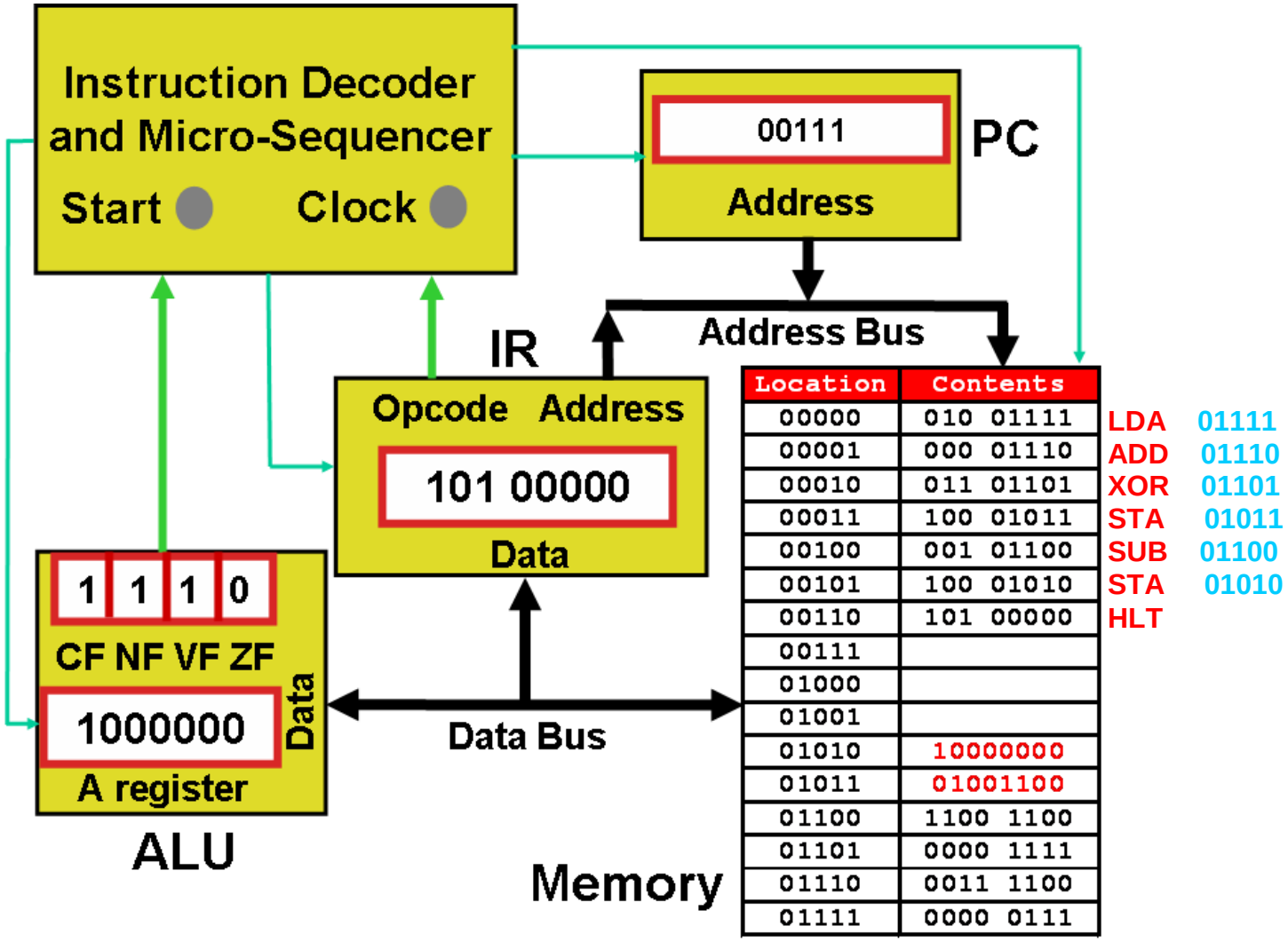
VF = 0

ZF = 0

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

Opcode	Mnemonic	Function Performed
0 0 0	ADD *addr*	Add contents of *addr* to contents of A
0 0 1	SUB *addr*	Subtract contents of *addr* from contents of A
0 1 0	LDA *addr*	Load A with contents of location *addr*
0 1 1	XOR *addr*	XOR contents of *addr* with contents of A
1 0 0	STA *addr*	Store contents of A at location *addr*
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.



13. To implement transfer-of-control instructions in the simple computer requires modification of the program counter.

Briefly describe the modification that is necessary:

The Program Counter (PC) needs to be able to be loaded from the Instruction register (IR) via the address bus.

Complete the ABEL file below that implements the modified program counter:

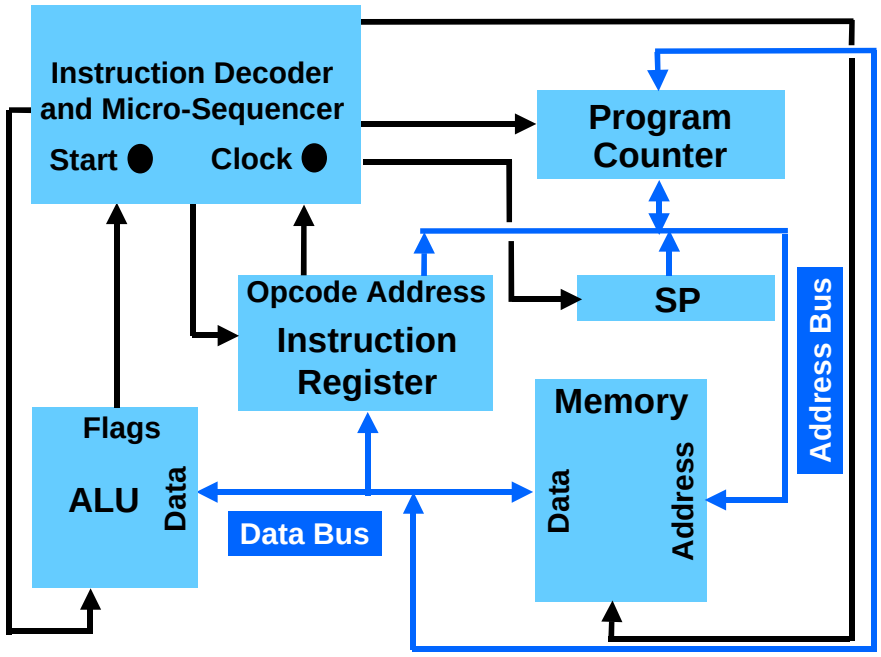
```
MODULE pcwl
TITLE  'Program Counter with Load Capability'

DECLARATIONS
CLOCK pin;
PC0..PC4 pin istype 'reg_D,buffer';
PCC pin; " PC count enable
ARS pin; " asynchronous reset (connected to START)
POA pin; " PC output on address bus tri-state enable
PLA pin; " PC load from address bus enable
" Note: Assume PCC and PLA are mutually exclusive

EQUATIONS
"          retain state       load           count up by 1
PC0.d = !PCC&!PLA&PC0.q # PLA&PC0.pin # PCC&!PC0.q;

PC1.d = !PCC&!PLA&PC1.q # PLA&PC1.pin # PCC&(PC1.q$PC0.q);

PC2.d = !PCC&!PLA&PC2.q # PLA&PC2.pin # PCC&(PC2.q$(PC1.q&PC0.q));

PC3.d = !PCC&!PLA&PC3.q # PLA&PC3.pin # PCC&(PC3.q$(PC2.q&PC1.q&PC0.q));

PC4.d = !PCC&!PLA&PC4.q # PLA&PC4.pin # PCC&(PC4.q$(PC3.q&PC2.q&PC1.q&PC0.q));


[PC0..PC4].oe = POA;

[PC0..PC4].ar = ARS;

[PC0..PC4].clk = CLOCK;

END
```

14. Complete the ABEL file, below, that implements a latched input/output port on the Simple Computer. ***Note the port address specified.***

```
MODULE iol

TITLE    'Input/Output Port 10110 - With Output Latch'

DECLARATIONS

DB0..DB7 pin istype 'com';   " data bus
AD0..AD4 pin;                " address bus
IN0..IN7 pin;                " input port
OUT0..OUT7 pin istype 'com'; " output port
IOR pin; " Input port read
IOW pin; " Output port write

" Port select equation for port address 10110

PS = AD4&!AD3&AD2&AD1&!AD0;

EQUATIONS

[DB0..DB7] = [IN0..IN7];

[DB0..DB7].oe = IOR&PS;

" Transparent latch for output port

[OUT0..OUT7] = !(IOW&PS)&[OUT0..OUT7] # IOW&PS&[DB0..DB7];

END
```

15. 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) ← (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	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

16. 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																		