In-Class Homework for Module 4 – No. 1
Monday, April 7, 2014

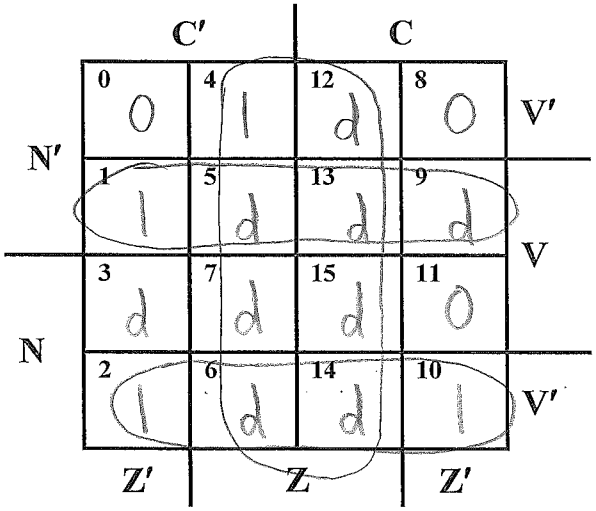
Solve all possible 2-bit Radix subtraction problems. Determine how the condition codes (C,Z,N,V) are set for each case.

1 1 (-1) 11 - 0 0 -(0) 11 (-1) +1 1 1 1 C = 0 Z = 0 N = 1 V = 0	1 1 (-1) 11 - 0 1 -(+1) 10 (-2) +1 1 1 0 C = 0 Z = 0 N = 1 V = 0	1 1 (-1) 11 - 1 0 -(-2) 01 (+1) +1 1 0 1 C = 0 Z = 0 N = 0 V = 0	1 1 (-1) 11 - 1 1 -(-1) 00 (0) +1 1 0 0 C = 0 Z = 1 N = 0 V = 0
1 0 (-2) 10 - 0 0 -(0) 11 (-2) +1 1 1 0 C = 0 Z = 0 N = 1 V = 0	1 0 (-2) 10 - 0 1 -(+1) 10 (-3) +1 1 0 1 C = 0 Z = 0 N = 0 V = 1	1 0 (-2) 10 - 1 0 -(-2) 01 (0) +1 1 0 0 C = 0 Z = 1 N = 0 V = 0	1 0 (-2) 10 - 1 1 -(-1) 00 (-1) +1 0 1 1 C = 1 Z = 0 N = 1 V = 0
0 1 (+1) 01 - 0 0 -(0) 11 (+1) +1 1 0 1 C = 0 Z = 0 N = 0 V = 0	0 1 (+1) 01 - 0 1 -(+1) 10 (0) +1 1 0 0 C = 0 Z = 1 N = 0 V = 0	0 1 (+1) 01 - 1 0 -(-2) 01 (+3) +1 0 1 1 C = 1 Z = 0 N = 1 V = 1	0 1 (+1) 01 - 1 1 -(-1) 00 (+2) +1 0 1 0 C = 1 Z = 0 N = 1 V = 1
0 0 (0) 00 - 0 0 -(0) 11 (0) +1 1 0 0 C = 0 Z = 1 N = 0 V = 0	0 0 (0) 00 - 0 1 -(+1) 10 (-1) +1 0 1 1 C = 1 Z = 0 N = 1 V = 0	0 0 (0) 00 - 1 0 -(-2) 01 (+2) +1 0 1 0 C = 1 Z = 0 N = 1 V = 1	0 0 (0) 00 - 1 1 -(-1) 00 (+1) +1 0 0 1 C = 1 Z = 0 N = 0 V = 0

In-Class Homework for Module 4 – No. 2
Wednesday, April 9, 2014

Complete the missing entries in the "condition code generation chart," below, and derive the function for "A *less than or equal to* B" ("ALEB") in its *simplest form.*

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



$ALEB = Z + N' \cdot V + N \cdot V'$

$= Z + N \oplus V$

00 − 01 = 00 + 10 + 1 = 011

10 − 00 = 10 + 11 + 1 = 110

10 − 01 = 10 + 10 + 1 = 101

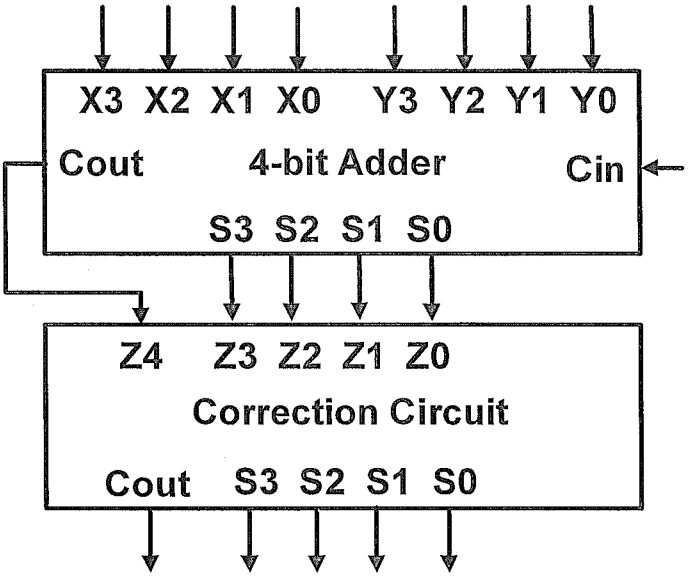
10 − 11 = 10 + 00 + 1 = 011

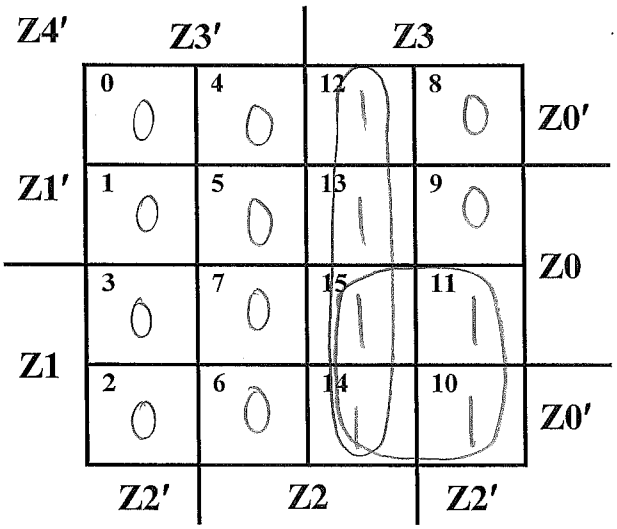
11 − 00 = 11 + 11 + 1 = 111

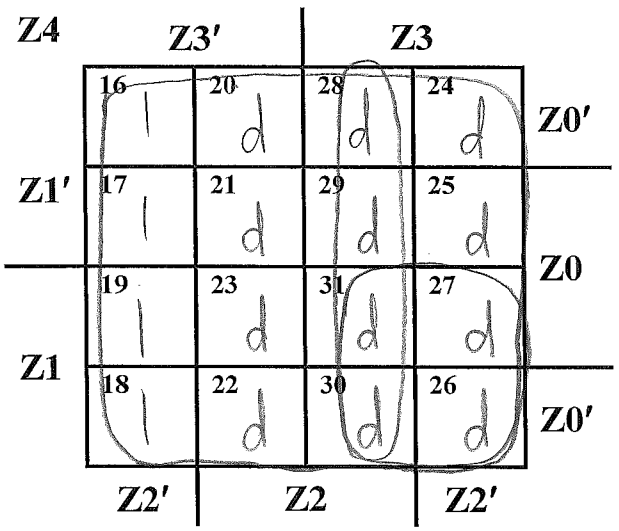
11 − 01 = 11 + 10 + 1 = 110

In-Class Homework for Module 4 – No. 3
Monday, April 14, 2014

In the BCD full adder illustrated below, the correction circuit adds 0110_2 to the "direct sum" ($Z_3Z_2Z_1Z_0$) under certain conditions to produce a valid BCD result. Derive the function, $F(Z_4Z_3Z_2Z_1Z_0)$, that is "1" when 0110_2 is to be added to the direct sum to produce a valid BCD result ($S_3S_2S_1S_0$), and is "0" otherwise.







$F(Z_4Z_3Z_2Z_1Z_0) =$ $Z2 \cdot Z3 + Z1 \cdot Z3 + Z4$

In-Class Homework for Module 4 – No. 2a
Wednesday, April 16, 2014

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	AND addr	(A) ← (A)∩(addr)
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 01110
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

Location	*Contents*
00000	001 01101
00001	011 01110
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	0011 0010
01101	0111 1011
01110	1011 0110
01111	0110 1110

AFTER 1st BLOCK

CF = 0

NF = 0

VF = 0

ZF = 0

0111 1011
∧ 1011 0110
0011 0010

AFTER 2nd BLOCK

Location	*Contents*
00000	001 01101
00001	011 01110
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	0011 0010
01101	0111 1011
01110	1011 0110
01111	0110 1110

CF = 0

NF = 0

VF = 1

ZF = 0

1011 0110
0110 1110

1011 0110
1001 0001
+ 1
01 001000

Location	*Contents*
00000	001 01101
00001	011 01110
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	0011 0010
01101	0111 1011
01110	1011 0110
01111	0110 1110

AFTER 3rd BLOCK

CF = 1

NF = 0

VF = 0

ZF = 0

0110 1110
+ 1011 0110
1 0010 0100

In-Class Homework for Module 4 – No. 2b
Wednesday, April 16, 2014

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

0111 1011

1000 0100

\+ 1

1000 0101

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

1011 0110

-0110 1110

1011 0110

1001 0001

\+ 1

10010010

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

0110 1110

+1011 0110

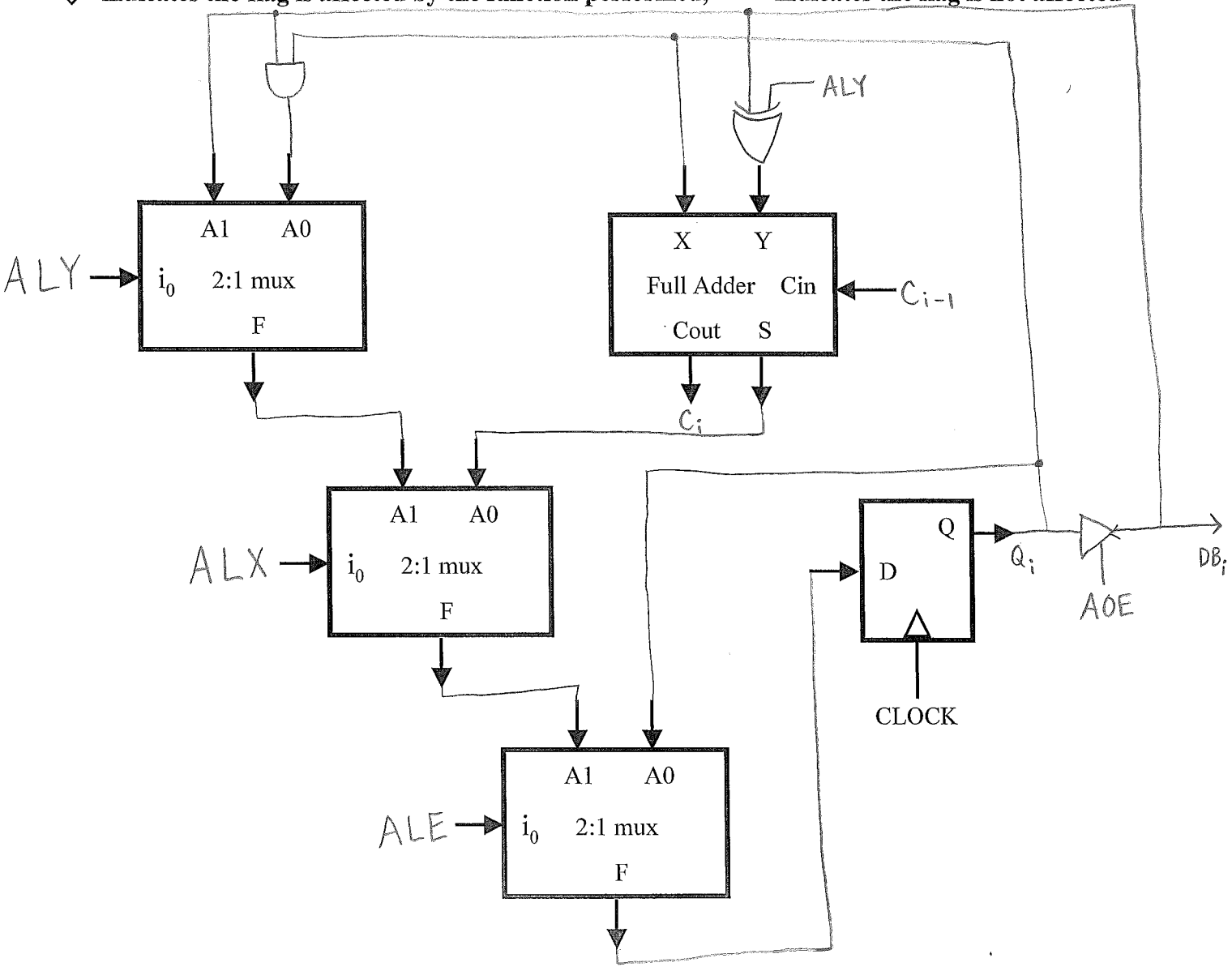
10010 0000

In-Class Homework for Module 4 – No. 4c
Wednesday, April 16, 2014

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	ADD: [Q3..Q0] ← [Q3..Q0] + [D3..D0]	⇳	⇳	⇳	⇳
0	1	0	1	SUB: [Q3..Q0] ← [Q3..Q0] – [D3..D0]	⇳	⇳	⇳	⇳
0	1	1	0	AND: [Q3..Q0] ← [Q3..Q0] ∩ [D3..D0]	•	⇳	⇳	•
0	1	1	1	LDA: [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



In-Class Homework for Module 4 – No. 5
Monday, April 21, 2014

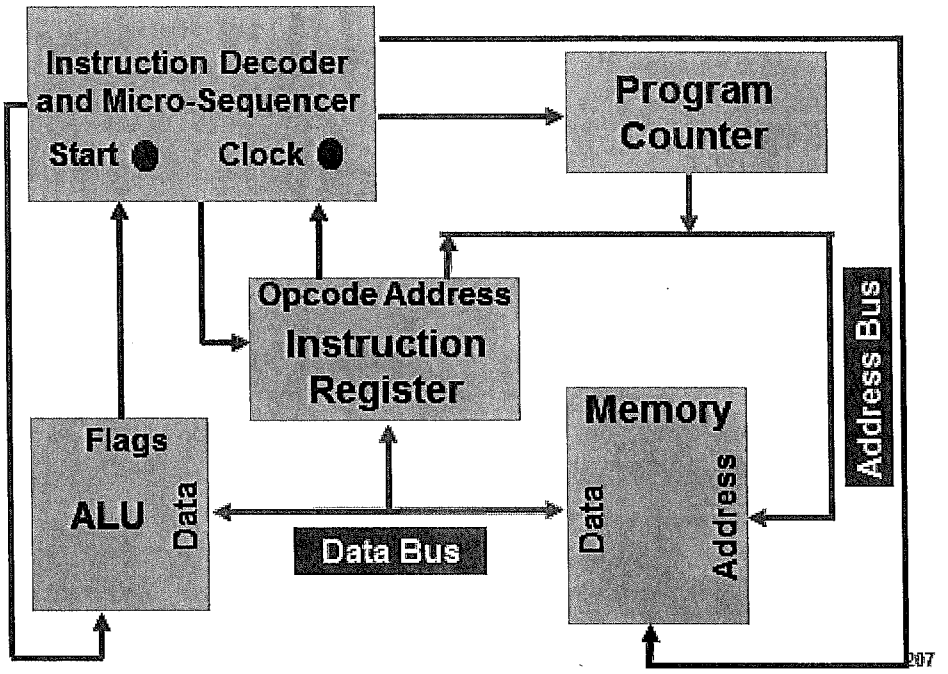
Simple Computer Functional Block Review

Before class on April 21st, view the online video for Module 4-I. Complete the skeletal section that follows for your assigned functional block. Be prepared to present and discuss the completed skeletal section of your functional block and any ABEL code that may be associated with it.

Circle the functional block you have been assigned:

Memory Program Counter (PC) Instruction Register (IR)

Arithmetic Logic Unit (ALU) Instruction Decoder Microsequencer (IDMS)



Memory

- Function - a place to store the program, operands, and computation results
- Modes of operation
  - Read - An address is placed on the address lines while CS and OE are asserted; the latch outputs for the selected location are output on the data lines
  - Write - An address is placed on the address lines, data is placed on the data lines, then CS and WE are asserted; the latches of the selected location open, and the data is stored
- RWM - read/write memory is given to memory arrays in which we can store and retrieve information at any time
- RAM - random-access memories means that the time it takes to read or write a bit of memory is independent of the bit's location in the RAM
- Volatile memory – data that is lost if power is removed
- Control inputs
  - a chip select (CS) signal that serves as the overall enable for the memory chip
  - an output enable (OE) signal that tells the memory chip to drive the data output lines with the contents of the memory location specified on its address lines
  - a write enable (WE) signal that tells the memory chip to write the data supplied on its data input lines at the memory location specified on its address lines
- Control signals
  - MSL: Memory SeLect
  - MOE: Memory Output Enable
  - MWE: Memory Write Enable
- Other notable information

Program Counter (PC)

- Function - a way to keep track of which instruction is to be executed next. The program counter (PC) is basically a binary "up" counter with tri-state outputs
- Control signals
  - ARS: Asynchronous ReSet
  - PCC: Program Counter Count enable
  - POA: Program counter Output on Address bus tri-state buffer enable
- PC in ABEL
- Other notable information

Instruction Register (IR)

- Function - a place to temporarily "stage" an instruction while it is being executed. The instruction register (IR) is basically an 8-bit data register, with tri-state outputs on the lower 5 bits.
- Control signals
  - IRL: Instruction Register Load enable
  - IRA: Instruction Register Address field tri-state output enable
- IR in ABEL
- Other notable information
  - Note that the upper 3 bits (opcode field) are output directly to the instruction decoder and micro-sequencer

Arithmetic Logic Unit (ALU)

- Function - a way to perform arithmetic and logic operations. ALU is a multi-function register that performs all the arithmetic and logical (Boolean) operations necessary to implement the instruction set.
- Control signals
    - ALE: ALU Enable
    - ALX: ALU "X" function select
    - ALY: ALU "Y" function select
    - AOE: A register tri-state Output Enable
- ALU in ABEL
- Other notable information

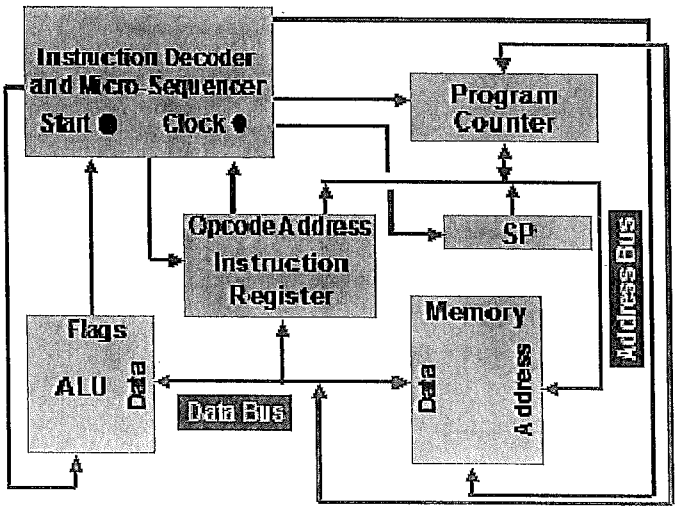
Instruction Decoder and Microsequencer (IDMS)

- Function - a way to coordinate and sequence the functions of the machine. The instruction decoder and microsequencer (IDMS) is a state machine that orchestrates the activity of all the other functional blocks.
- There are two basic steps involved in "processing" each instruction of a program (called a micro-sequence):
  - fetching the instruction from memory (at the location pointed to by the PC), loading it into the IR, and incrementing the PC
  - executing the instruction staged in the IR based on the opcode field and the operand address field
- The control signals that need to be asserted during the fetch cycle include:
  - POA: turn on PC output buffers
  - MSL: select memory
  - MOE: turn on memory output buffers
  - IRL: enable IR load
  - PCC: enable PC count
- The control signals that need to be asserted during an execute cycle for the synchronous ALU functions (ADD, SUB, LDA, AND) are:
  - IRA: turn on operand address output buffers
  - MSL: select memory
  - MOE: turn on memory data output buffers
  - ALE: enable ALU operation
  - ALX, ALY: select ALU function
- The control signals that need to be asserted during an execute cycle for STA are:
  - IRA: turn on operand address output buffers
  - MSL: select memory
  - MWE: enable memory write
  - AOE: turn on A register output buffers
- IDMS in ABEL
- Other notable information

In-Class Homework for Module 4 – No. 6
Wednesday, April 23, 2014

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 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:

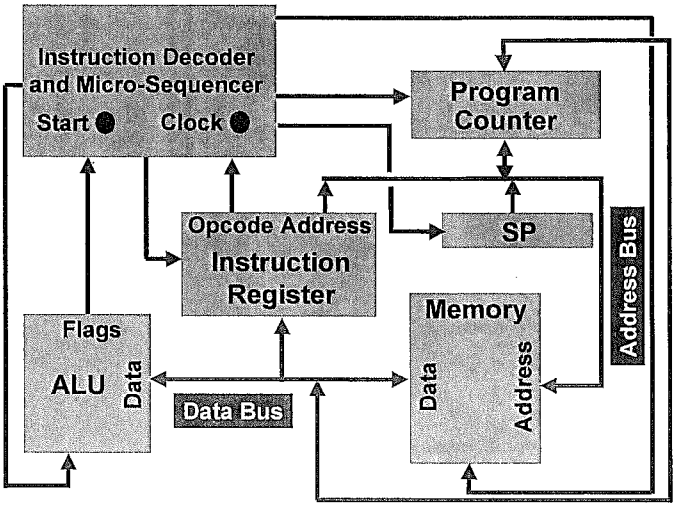
AOE	ALE	ALX	ALY	Function	CF	ZF	NF	VF
0	1	0	0	Add	X	X	X	X
0	1	0	1	Subtract	X	X	X	X
0	1	1	0	Load	•	X	X	•
0	1	1	1	Arithmetic Shift Right*	X	X	X	•
1	0	d	d	Output	•	•	•	•
0	0	d	d	<none>	•	•	•	•

X → flag affected, • → flag not affected

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

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



SIMPLE COMPUTER REFERENCE SHEET

In-Class Homework for Module 4 – No. 7
Monday, April 28, 2014

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