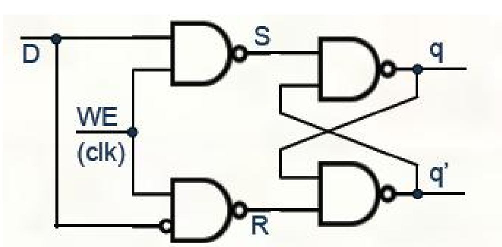
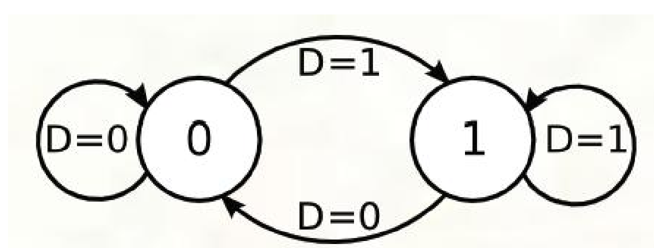
Lecture 9-11 – 2/7/11 – 2/11/11

- Announcements
  - Hwk 2 due Thursday; Hwk 3 posted tonight
  - Test 1: Tuesday, Feb 22nd 5:45 -7PM through Lecture 11
- Last Week (P&P 3.4-3.6)
  - Storage
  - Sequential Logic
  - Clocks
- This Week (P&P 3.6-3.7; 2)
  - Finite State Machines
  - LC-3 Datapath
  - Representation
- Next Week
  - LC-3

Finite State Machine (FSM)

- A mechanism for <u>describing</u> a system that includes storage and computation
  - Output is a function of the inputs as well as history
  - Implemented by sequential logic
- Represented by a state diagram
  - States (Circles)
  - Transitions (Arcs)
- Example: D-Latch





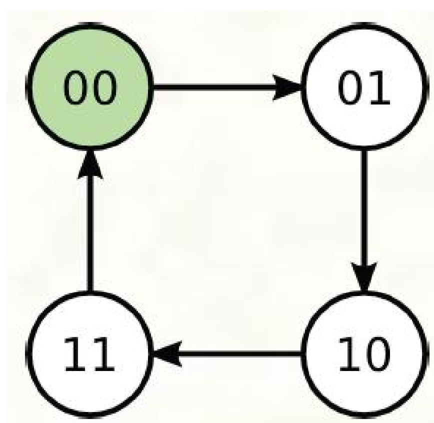
FSM: Definition

- An FSM has the following components:
  - A set of states
  - A set of inputs
  - A set of outputs
  - A state transition function (of the states and inputs)
  - An output function
    - Moore machine: of the states only
    - Mealy machine: of the states and inputs
- An FSM is *synchronous* if all changes to memory (state) occur at the same time determined by a global system clock
- Represented by a state diagram
  - States (Circles, labeled with output (Moore))
  - Transitions (Arcs, labeled with input values and output (Mealy))
  - Clock is typically not shown

Example 1: 2-Bit Counter

- Counter starts at 0 (green), increments each time the clock cycles, overflowing back to 0 when it gets to 3

H_{old}	L_{old}	H_{new}	L_{new}
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

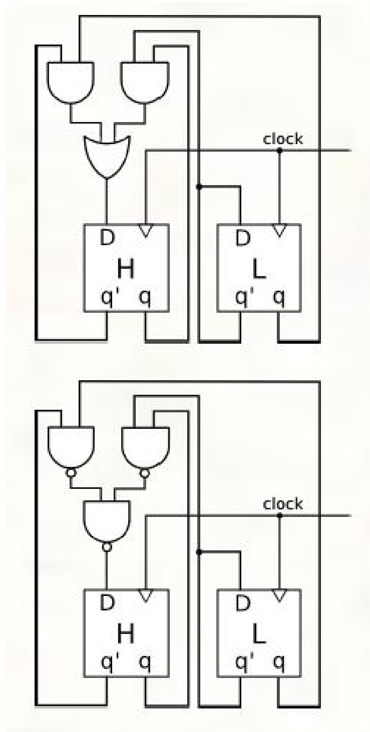


Example 1: 2-Bit Counter (cont')

H_{old}	L_{old}	H_{new}	L_{new}
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

$L_{new} = H_{old}'L_{old}' + H_{old}L_{old}' = L_{old}'$

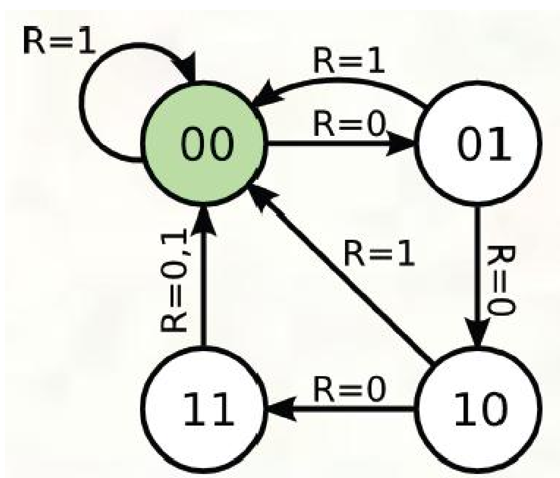
$H_{new} = H_{old}'L_{old} + H_{old}L_{old}'$



Example 2: 2-Bit Counter With Reset

- Counter starts at 0 (green), increments each time the clock cycles, overflowing back to 0 when it gets to 3. Resets to 00 when R=1.

R	H_{old}	L_{old}	H_{new}	L_{new}
0	0	0	0	1
0	0	1	1	0
0	1	0	1	1
0	1	1	0	0
1	X	X	0	0



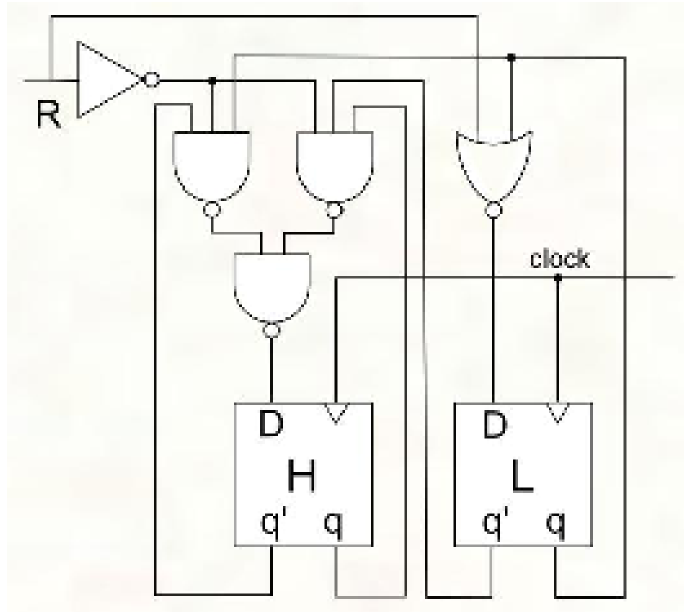
$$L_{new} = R'H_{old}'L_{old}' + R'H_{old}L_{old}'$$
$$= R'L_{old}' = (R + L_{old})'$$

$$H_{new} = R'H_{old}'L_{old} + R'H_{old}L_{old}'$$
$$= R'(H_{old}'L_{old} + H_{old}L_{old}')$$

Example 2: 2-Bit Counter With Reset (cont')

$L_{new} = (R + L_{old})'$

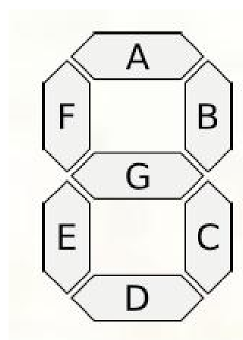
$H_{new} = R'(H_{old}'L_{old} + H_{old}L_{old}')$



Example 3: 2-Bit Counter With Display

- Each segment in the display can be lit independently to allow all ten decimal digits to display

R	H_{old}	L_{old}	H_{new}	L_{new}	A	B	C	D	E	F	G
0	0	0	0	1	1	1	1	1	1	1	0
0	0	1	1	0	0	1	1	0	0	0	0
0	1	0	1	1	1	1	0	1	1	0	1
0	1	1	0	0	1	1	1	1	0	0	1
1	X	X	0	0	0	0	0	0	0	0	0

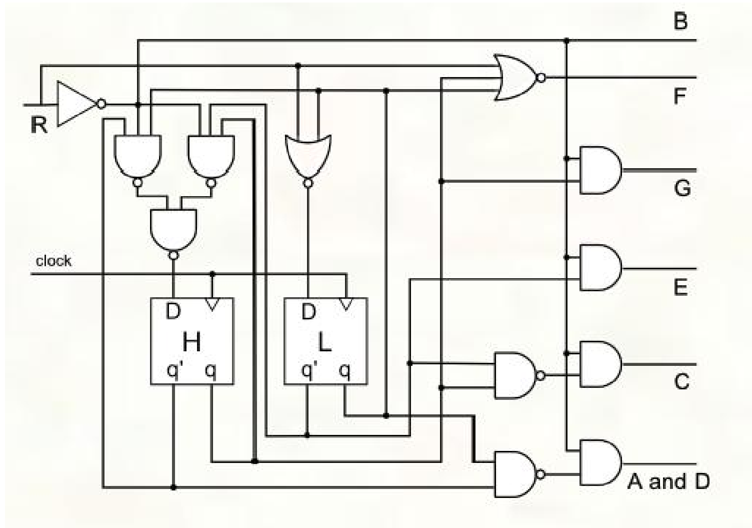


$L_{new} = (R + L_{old})'$ $H_{new} = R'(H_{old}'L_{old} + H_{old}L_{old}')$

$A = D = R'(H_{old}'L_{old})'$ $B = R'$ $C = R'(H_{old}L_{old}')'$

$E = R'L_{old}'$ $F = (R + H_{old} + L_{old})'$ $G = R'H_{old}$

Example 3: Display Logic



Example 4: Pattern Recognition

- Output a “1” when three consecutive “1” inputs have been seen; “0” at all other times

- Check out the “traffic Sign” state diagram in Section 3.6.4

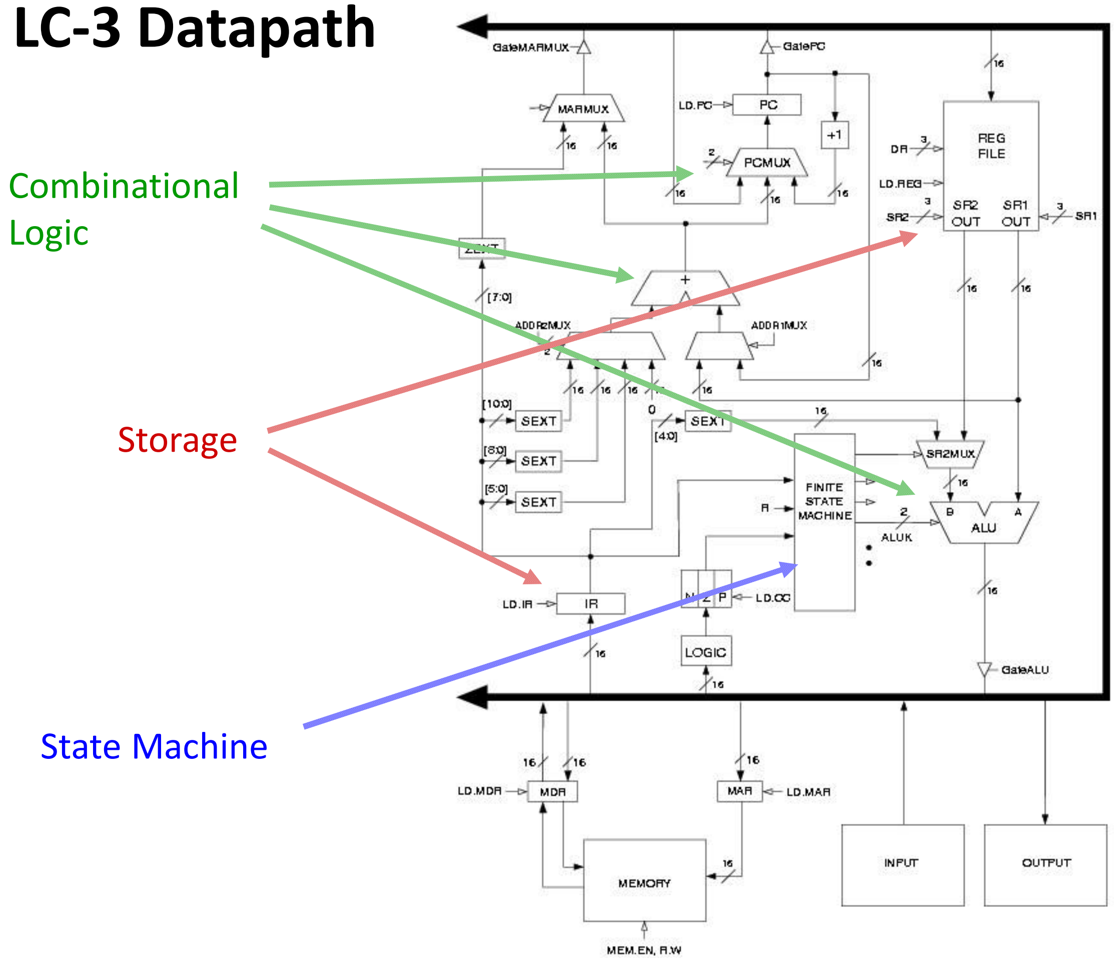
Shift Registers

- Several uses for shift operations
  - A cheap multiply operation (when the multiplier or multiplicand is a power of 2)
  - Get access to a specific bit

From Logic to Datapath

- Datapath: All the logic in a processor used to process data
- Combinational logic
  - Decoders: Convert instructions into control signals
  - MUXes: Select inputs and outputs
  - ALU (Arithmetic Logic Unit): Perform operations on data
- Sequential logic
  - State machines: Control sequencing of control signals and data movements
  - Registers and latches: Store stuff

LC-3 Datapath



Integers 1

- Binary Coded Decimal (BCD)
  - Four bits to encode each decimal digit + four bits for the sign
    - $0000_2 - 1001_2$ for 0-9
    - 1010_2 for “+” 1011_2 for “–”
  - Difficult to do arithmetic efficiently
- Signed magnitude
  - Use one bit to represent the sign
  - Two values of zero!
    - Complicates circuitry
- One’s complement
  - Leading bit indicates sign
  - Magnitude computed by inverting rest of the bits
  - Still have a negative zero

Integers 2

- Two's complement format
  - Leading bit indicates sign (like one's complement)
  - Magnitude computed by inverting rest of the bits and adding 1
  - Eliminates negative zero
  - For an n-bit number, range is: $-2^{n-1} – 2^{n-1}-1$
- Overflow detection
  - Add two numbers of the same sign and get the wrong sign
- How do we subtract?
  - A – B = A + -B = A + B' + 1
- How do we operate on numbers that are of unequal length?
  - Sign extension

Text

- Need an encoding for each characters
  - A string is just an array of characters
- ASCII (American Standard Code for Information Interchange)
  - 8 bits (one byte): 256 encodings
    - Example: 'D' = 0x44 = 010001002
    - Example: ';' = 0x3B = 001110112
  - String example: 'hello' = 0x68 0x65 0x6c 0x6c ox6f 0x00
    - Note the use of a Null (0x00) to terminate the string
- EBCDIC (Extended Binary Coded Decimal Interchange Code)
  - Developed by IBM in the 60s concurrently with ASCII
- Unicode
  - An extensible coding scheme that facilitates encoding characters from languages other than English

ASCII Table

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Bits Are Bits

- Suppose we're using 6-bit numbers, then
  - x2B 43 if unsigned
    - -11 if signed magnitude
    - -20 if one's complement
    - -21 if two's complement
    - '+' if ASCII character
- How we interpret bits is crucial!
- Instructions operate on bits
  - Compiler/assembly language programmer is responsible for knowing what is being represented and using appropriate instructions/operations

Operations

- Arithmetic
  - Overflow
- Shift/Rotate
- Logical
- Comparison

Digression: Standards

- “Without standards there’d be no computing”
  - Just look at the “Bits Are Bits” slide!
- Standards organizations
  - ISO: International Standards Organization
  - IEC: International Electrotechnical Commission
  - ITC: International Telecommunication Union
- Domain-specific
  - OpenSocial: facilitates access to and interaction between social networking sites
  - SATA (Serial Advanced Technology Attachment): protocol for interactions between mass storage and a host

Floating Point: Some Set Up

- Scientific notation for representing numbers
  - (signed) mantissa x $10^{exponent}$
  - Mantissa is always 1 <= Mantissa < 10
  - So, 7732.34 = 7.73234 x 10^3
- Fractions in binary
  - 10.011 is:
    $= 1 * 2^1 + 0 * 2^0 + 0 * 2^{-1} + 1 * 2^{-2} + 1 * 2^{-3}$
    $= 1 * 2 + 0 + 0 + 1 * ¼ + 1 * 1/8$
    = 2 3/8
  - Can also be represented as $1.0011 * 2^1$
    - Like scientific notation, just in base 2
  - If we force the mantissa to always be: 1 <= Mantissa < 2 then we can save a bit in the representation
  - So is M is the value stored in the mantissa, then the real value is 1.M

Floating Point Representation 1

- Floating point representation (IEEE 754)

	32-bit (Single Precision)	64-bit (Double Precision)
Sign	1	1
Exponent	8	11
Mantissa	23	52

- Exponent is in excess-127 representation (single-precision)
  - An unsigned number: 0 – 255 from which we subtract 127
  - So, exponent ranges from -127 to 128.
  - End values (-127 & 128) are special (0 & infinity)
  - So, -126 <= exponent <= 127
- Given S, E & M the value represented is:
  if (E > 0 and E < 255) value = $(-1)^S * 2^{(E-127)} * 1.M$

Floating Point Representation 2

- Given S, E & M the value represented is:
  if (E > 0 and E < 255) value = $(-1)^S * 2^{(E-127)} * 1.M$
- Example: Represent -3/4
- S = 1 (the number is negative)
- M:
  - ¾ = ½ + ¼ or 0.11_2, but the mantissa must be in the form 1.xyz…
  - So, normalize 0.11_2. Express it as $1.1 * 2^{-1}$
  - Therefore the mantissa is $10000000000000000000000_2$(23 digits) or 0x400000
- E:
  - Showed it to be -1
  - Express -1 in excess-127 notation = -1 + 127 = 126 = 01111110_2
- -3/4 = 1 01111110 10000000000000000000000
  or 0xBF400000

Floating Point Representation 3

- Given S, E & M the value represented is:
  if (E > 0 and E < 255) value = $(-1)^S * 2^{(E-127)} * 1.M$
- Special case 1: E = 0 (-127 in excess-127 notation)
  - $(-1)^S * 2^{-127} * 0.M$
  - Represents zero as well as very small numbers
- Special case 2: E = oxFF (128 in excess-127 notation)
  - M = 0x000000 (all zeros), encodes +/- infinity
  - M != 0x000000, encodes NaN (Not a Number)
    - Arises when the result of an operation is indeterminant
    - Eg, infinity - infinity

Floating Point Addition 1

- Four steps

1. Adjust Mantissa
   a. Choose number with smaller exponent
   b. Shift its mantissa right the number of places in the difference
2. Adjust Exponent
   a. Set the smaller exponent to the value of the larger exponent
3. Add/Subtract
   a. Perform addition/subtraction on the mantissas
   b. Determine the value of the sign
4. Readjust Mantissa & Exponent
   a. Normalize the resultant mantissa until the bit to the left of the decimal is 1 (may require left or right shift)
   b. Adjust the exponent of the result accordingly decreasing/increasing it by the number of places shifted to the left/right

Floating Point Addition 2

- Example: 21.5 + 2.25

Floating Point Multiplication

- Three steps

1. Add Exponents
   a. Add the two exponents and subtract 127
2. Multiply
   a. Perform binary multiplication on the mantissas
   b. Determine the value of the sign
3. Readjust Mantissa & Exponent
   a. Normalize the resultant mantissa until the bit to the left of the decimal is 1 (may require left or right shift)
   b. Adjust the exponent of the result accordingly decreasing/increasing it by the number of places shifted to the left/right