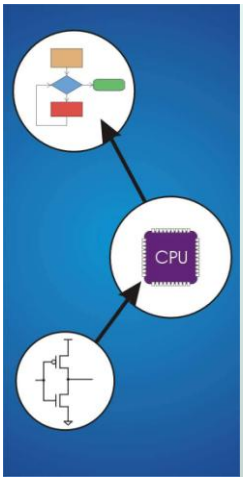


Midterm Exam
Review Slides

Original slides from Gregory Byrd, North Carolina State University

Modified slides by Chris Wilcox, Colorado State University

Review Topics

- Number Representation
- Computer Arithmetic
- Transistors and Gates
- Combinational Logic
- Sequential Circuits
- Finite State Machines
- C Programming
- gdb Debugging
- LC-3 Architecture

Number Representation
What can a binary number mean?

- Interpretations of a 32-bit memory location:
  - 32-bit floating point (IEEE)
  - 32-bit unsigned/signed integer
  - 16-bit unsigned/signed integer (2)
  - 8-bit unsigned/signed bytes (4)
  - ASCII characters (4)
  - RISC instruction
  - Control or status register
  - .jpg. .mpg, .mp3., .avi, …

Number Representation
Hexadecimal to Binary Conversion

- Method: Convert hexadecimal digits to binary using table.
- Question: What is hexadecimal 0xFEBD4570 in binary?

```
F    E    B    D    4    5    7    0
1111 1110 1011 1101 0100 0101 0111 0000
```

- Answer:

```
11111110101111010100010101110000
```

Hexadecimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Number Representation

Binary to Hexadecimal Conversion

- Method: Group binary digits, convert to hex digits using table.
- Question: What is binary 1100110111101111000100100011000 in hexadecimal?

```
1100 1101 1110 1111 0001 0010 0011 0000
C    D    E    F    1    2    3    0
```

- Answer: `0xCDEF1230`

Hexadecimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Number Representation

Decimal to Binary Conversion

- Method: Convert decimal to binary with divide by 2, check odd/even.
- Question: What is decimal 49 in binary?

```
49 is odd, prepend a '1'                1
49 / 2 = 24 is even, prepend a '0'     01
24 / 2 = 12 is even, prepend a '0'    001
12 / 2 = 6 is even, prepend a '0'    0001
6 / 2 = 3 is odd, prepend a '1'     10001
3 / 2 = 1 is odd, prepend a '1'    110001
```

Answer: `110001`

2^n	Decimal
2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256
2^9	512
2^{10}	1024

Number Representation

Binary to Decimal Conversion

- Method: Convert binary to decimal by multiplying by 2, add 1 if bit set.
- Question: What is binary 110101 in binary?

```
Start with 0                            0
Left bit set, multiply by 2, add 1      1
Left bit set, multiply by 2, add 1      3
Left bit clear, multiply by 2           6
Left bit set, multiply by 2, add 1     13
Left bit clear, multiply by 2          26
Left bit set, multiply by 2, add 1     53
```

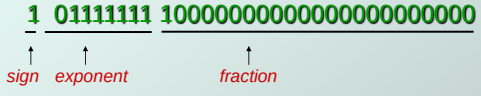
Answer: `53`

2^n	Decimal
2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256
2^9	512
2^{10}	1024

Number Representation

Binary to Floating Point Conversion

- Single-precision IEEE floating point number:



  - Sign is 1 – number is negative.
  - Exponent field is 01111111 = 127 – 127 = 0 (decimal).
  - Fraction is 1.100000000000… = 1.5 (decimal).

- Value = $-1.5 \times 2^{(127-127)} = -1.5 \times 2^0 =$ **-1.5**

Number Representation
Floating Point to Binary Conversion

- Value = 6.125
  - Number is positive – sign is 0
  - Fraction is 110.001 (binary), normalize to 1.10001 * 2^2
  - Exponent is 2 + 127 = 129 (decimal) = 10000001
- Single-precision IEEE floating point number:

0 10000001 10001000000000000000000

sign exponent fraction

Number Representation
Hexadecimal to ASCII Conversion

- Method: Convert values to ASCII by table lookup.
- Each two (hex) digits is a single character.
- Question: What is hex 0x42454144 in binary?

0x42 = 'B'
0x45 = 'E'
0x41 = 'A'
0x44 = 'D'

- Answer: "BEAD"

Char	ASCII Code	Char	ASCII Code
'A'	0x41	'0'	0x30
'B'	0x42	'1'	0x31
'C'	0x43	'2'	0x32
'D'	0x44	'3'	0x33
'E'	0x45	'4'	0x34
'F'	0x46	'5'	0x35
'G'	0x47	'6'	0x36

Computer Arithmetic
Signed Integer Representations

Binary Number	Signed Magnitude	1's Complement	2's Complement
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	-0	-7	-8
1001	-1	-6	-7
1010	-2	-5	-6
1011	-3	-4	-5
1100	-4	-3	-4
1101	-5	-2	-3
1110	-6	-1	-2
1111	-7	-0	-1

Computer Arithmetic
2's Complement Arithmetic

- Binary Arithmetic (unsigned integers):

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

- Hex Equivalent:

0x92 + 0x35 = 0xC7

- Decimal Equivalent:

146 + 53 = 199

- Binary Arithmetic (signed integers):

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

- Hex Equivalent:

0x92 + 0x35 = 0xC7

- Decimal Equivalent:

-110 + 53 = -57

Computer Arithmetic
Bitwise Logical Operations

- Bitwise AND (&):

```
  1 1 1 1 0 0 0 0
& 0 0 1 1 0 1 0 1
  0 0 1 1 0 0 0 0
```

- Hex Equivalent:

0xF0 & 035 = 0xC0

- Bitwise OR (|):

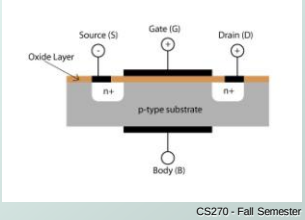
```
  1 1 1 1 0 0 0 0
| 0 0 1 1 0 1 0 1
  1 1 1 1 0 1 0 1
```

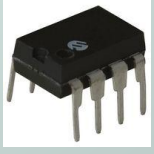
- Hex Equivalent:

0xF0 | 035 = 0xF5

Transistors and Gates
Transistor Basics (p-type and n-type)

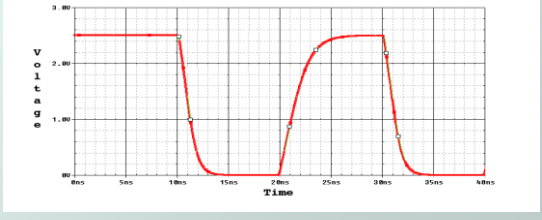
- Gate voltage determines current flow between source and drain.
  - P-type: 0V closes circuit, 2.9V opens circuit.
  - N-type: 2.9V closes circuit, 0V opens circuit.



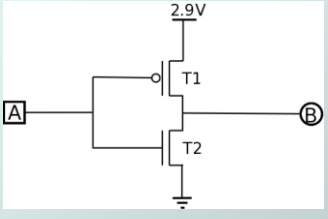


Transistors and Gates
Transistor Basics (p-type and n-type)

- Transistors are switches which have a propagation delay, waveform is not ideal, and voltage transition is not instantaneous!



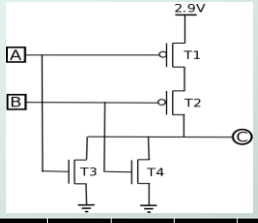
Transistors and Gates
NOT Gate



A	T1	T2	B
0	Closed	Open	1
1	Open	Closed	0

Transistors and Gates

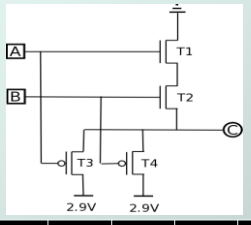
NOR Gate



A	B	T1	T2	T3	T4	C
0	0	Closed	Closed	Open	Open	1
0	1	Closed	Open	Open	Closed	0
1	0	Open	Closed	Closed	Open	0
1	1	Open	Open	Closed	Closed	0

Transistors and Gates

NAND Gate

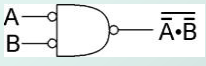


A	B	T1	T2	T3	T4	C
0	0	Open	Open	Closed	Closed	1
0	1	Open	Closed	Closed	Open	1
1	0	Closed	Open	Open	Closed	1
1	1	Closed	Closed	Open	Open	0

Transistors and Gates

De Morgan's Law

- Converting AND to OR (with some help from NOT)
- Consider the following gate:



A	B	$\bar{A}$	$\bar{B}$	$\bar{A} \cdot \bar{B}$	$\overline{\bar{A} \cdot \bar{B}}$
0	0	1	1	1	0
0	1	1	0	0	1
1	0	0	1	0	1
1	1	0	0	0	1

Same as A OR B!

To convert AND to OR (or vice versa), invert inputs and output.

NOT(NOT(A) AND NOT(B)) = A OR B
NOT(NOT(A) OR NOT(B)) = A AND B

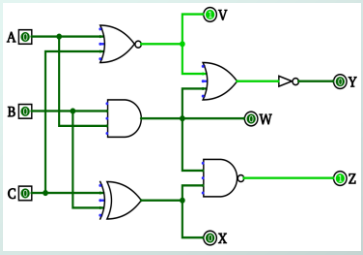
Transistors and Gates

Logical Completeness

1. AND/OR/NOT are logically complete, if you have enough gates you can build any truth table.
2. NAND/NOR are logically complete, same as above, so only these gates are sufficient!

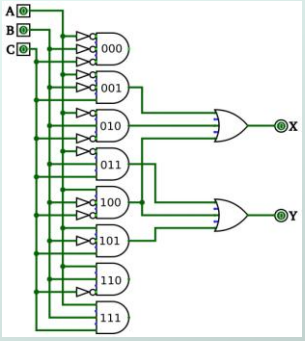
- Proof 1: Programmable logic array proves that any truth table can be built from AND/OR/NOT.
- Proof 2: Can synthesize AND/OR/NOT from NAND/NOR, though it may take more gates.

Combinational Logic
Combinational Circuit to Truth Table



A	B	C	V	W	X	Y	Z
0	0	0	1	0	0	0	1
0	0	1	0	0	1	1	1
0	1	0	1	0	1	0	1
0	1	1	0	0	0	1	1
1	0	0	0	0	0	1	1
1	0	1	0	0	1	1	1
1	1	0	0	1	1	0	0
1	1	1	0	1	0	0	1

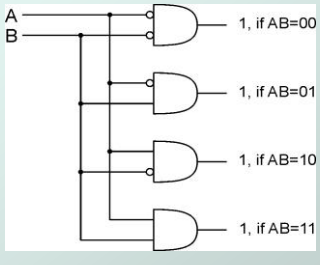
Combinational Logic
Truth Table to Combinational Circuit



A	B	C	X	Y
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	0	0

Combinational Logic
Decoder Circuit

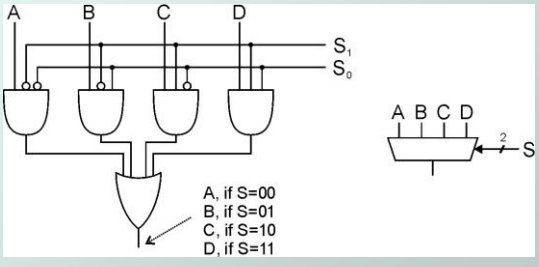
- *n* inputs, 2^n outputs
  - exactly one output is 1 for each input pattern



A	B	O00	O01	O10	O11
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

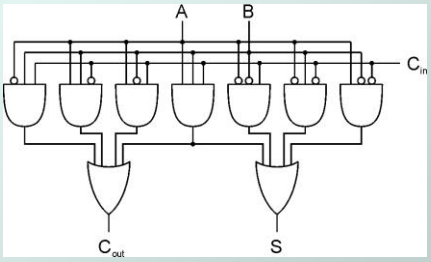
Combinational Logic
Multiplexer Circuit

- *n*-bit selector and 2^n inputs, one output
  - output equals one of the inputs, depending on selector



Combinational Logic
Full Adder Circuit

- Add two bits and carry-in, produce one-bit sum and carry-out.



A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

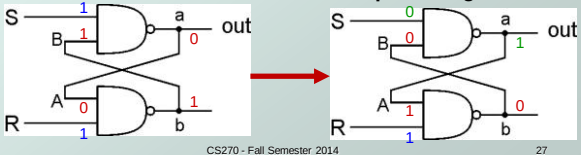
Sequential Circuits
Difference from Combinational

- Sequential circuits differ from combinational circuits because they have persistent state.
  - For a combinational circuit, the outputs depend only on the inputs.
  - For a sequential circuit, the outputs depend on the inputs and the state.
  - Sequential circuits can be used to implement a finite state machine.

Sequential Circuits
S-R Latch Circuit

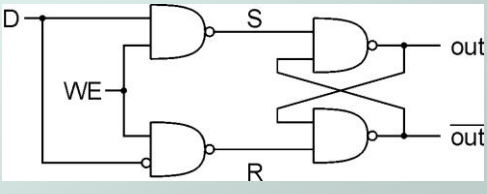
- Suppose we start with output = 0, then change S to zero (Set), latch state will change to 1.
- Or we start with output = 1, then change R to zero (Reset), latch state will change to 0.
- Setting S or R back to 1 makes latch quiescent, never do S = R = 0!

Output changes to one.



Sequential Circuits
D-Latch Circuit

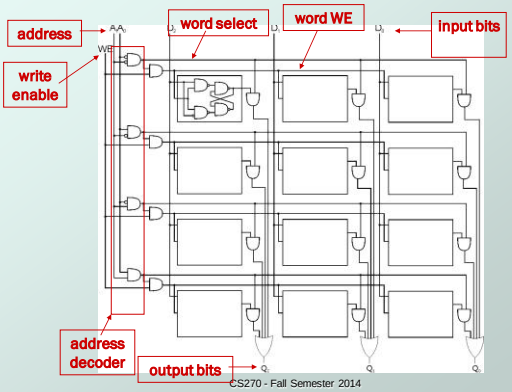
- Two inputs: D (data) and WE (write enable)
  - when **WE = 1**, latch is set to **value of D**
    - S = NOT(D), R = D
  - when **WE = 0**, latch holds **previous value**
    - S = R = 1



Sequential Circuits
Exhaustive Testing

- How many test cases for combinational logic?
  - 2^n, where n is the number of input bits
  - Example: 4-bit decoder requires 16 test cases
- How many test cases for sequential logic?
  - $2^n * 2^m$, where m is number of states
  - Example: 1-bit D-latch requires 8 test cases

Sequential Circuits - Memory Architecture



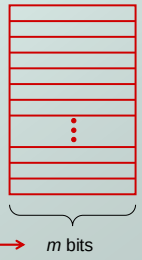
Sequential Circuits
Memory Address Space and Width

- Now that we know how to store bits, we can build a memory – a logical $k \times m$ array of stored bits.

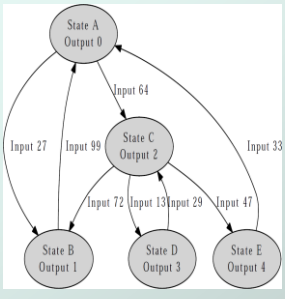
Address Space:
number of locations
(usually a power of 2)

$k = 2^n$ locations

Address Width (Addressability):
number of bits per location
(e.g., byte-addressable)

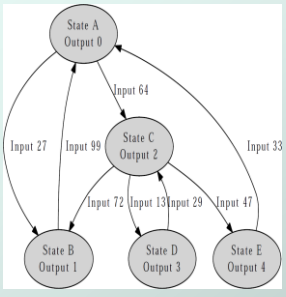


Finite State Machines
Finding States from Inputs



- Just follow the arrows, for example:
- Starting state is 'A'
- Inputs given:
  64, 13, 29, 47
- States visited:
  C, D, C, E
- State outputs:
  2, 3, 2, 4

Finite State Machines
Finding Inputs from States



- Just follow the arrows, for example:
- Starting state is 'C'
- States visited:

  E, A, B, A, C
- Inputs given:

  47, 33, 27, 99, 64
- **Not all paths are possible!**

C Programming
Bit Manipulation

- C code to read or write a bit:

```
int readBit(int value, int bit) {
      return (value >> bit) & 01;
      // return !!(value >> bit);
}

void writeBit(int *value, int bit) {
      *value |= 1<<bit;
}
```


C Programming
Control Structures

- C conditional and iterative statements
  - if statement

```
if (value == 0x12345678)
    printf("value matches 0x12345678\n");
```

  - for loop

```
for (int i = 0; i < 8; ++i)
    printf("i = %d\n", i);
```

  - while loop

```
int j = 6;
while (j--)
    printf("j = %d\n", j);
```


C Programming
Pointers and Arrays

- C pointers and arrays

```
void foo(int *pointer)
{
    *(pointer+0) = pointer[2] = 0x1234;
    *(pointer+1) = pointer[3] = 0x5678;
}
int main(int argc, char *argv[])
{
   int array[]= {0, 1, 2, 3};
   foo(array);
   for (int i = 0; i <= 3; ++i)
      printf("array[%d] = %x\n", i, array[i]);
}
```

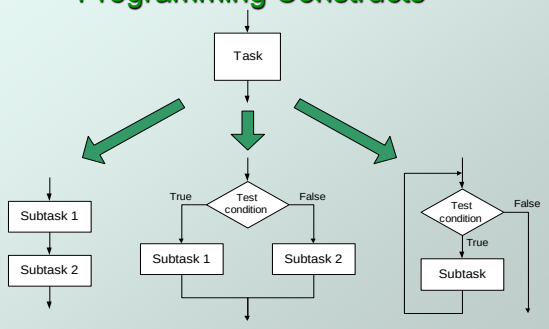
gdb Debugger
Basic Commands

- How to debug a program using *gdb*:

```
$ gdb a.out          // debug a program
(gdb) break main     // set breakpoint on function
(gdb) break 23       // set breakpoint in file
(gdb) run            // run program
(gdb) list 20        // list current file
(gdb) step           // single step
(gdb) print v        // display value of variable
(gdb) print *p       // deference pointer and display
(gdb) quit           // quit debugger
```

- Commands can be single letters (b, r, l, s, p, q)

Programming Basics
Programming Constructs

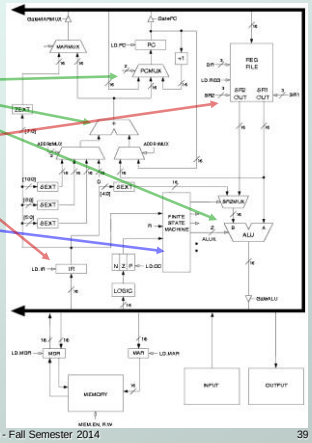


Sequential ***Conditional*** ***Iterative***

Combinational Logic

Storage

State Machine



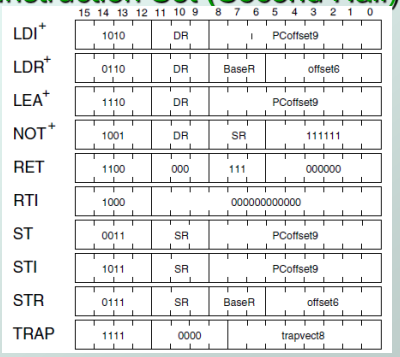
LC-3 Architecture
System Architecture

LC-3 Architecture
Instruction Set (First Half)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD+	0001			DR			SR1				0	0	0	SR2		
ADD+	0001			DR			SR1				1	imm5				
AND+	0101			DR			SR1				0	0	0	SR2		
AND+	0101			DR			SR1				1	imm5				
BR	0000			n		z		p		PCoffset9						
JMP	1100			000			BaseR			000000						
JSR	0100			1		PCoffset11										
JSRR	0100			0		0	0	BaseR			000000					
LD+	0010			DR			PCoffset9									

LC-3 Architecture

Instruction Set (Second Half)



LC-3 Architecture

Addressing Modes

- Load -- read data **from memory to register**
  - **LD:** PC-relative mode
  - **LDR:** base+offset mode
  - **LDI:** indirect mode
- Store -- write data **from register to memory**
  - **ST:** PC-relative mode
  - **STR:** base+offset mode
  - **STI:** indirect mode
- Load pointer: **compute address, save in register**
  - **LEA:** immediate mode
  - *does not access memory*

LC-3 Architecture

Machine Code to Assembly

- What is the assembly code for machine instruction `0101010010111101`?
- Step 1) Identify opcode: `0101` = AND
- Step 2) Parse entire instruction (use reference)
- Step 3) Get values from each field

OPCODE	DR	SR	1	imm5
15:12	11:9	8:6	5	4:0
0101	010	010	1	11101
AND	R2	R2		-3

- Step 4) Translate to mnemonics: `AND R2,R2,#-3`

LC-3 Architecture

Assembly to Machine Code

- What is the machine code for assembly instruction `NOT R7,R6`?
- Step 1) Identify opcode: NOT = `1001`
- Step 2) Put values into each field:

NOT	R7	R6	
OPCODE	DR	SR	111111
15:12	11:9	8:6	5:0
1001	111	110	111111

- Step 3) Build machine instruction: `1001111110111111`

LC-3 Architecture

Assembly Code Syntax

```
        .ORIG   x3000
MAIN    AND     R0,R0,#0      ; Initialize Sum
        JSR COMPUTE           ; Call function
        ST      R0, SUM       ; Store Sum
        HALT                  ; Program complete
COMPUTE LD      R1,OPERAND1   ; Load Operand1
        LD      R2,OPERAND2   ; Load Operand2
        ADD     R0,R1,R2      ; Compute Sum
        RET                   ; Function return
;; Input data set
OPERAND1 .FILL     x1234      ; Operand1
OPERAND2 .FILL     x4321      ; Operand2
SUM     .BLKW      1          ; Sum
        .END
```