Test 1 Reference Sheet and Review

Historical - Turing Machine

Mathematical model of a device that can perform any computation – Alan Turing (1937)

- Ability to read/write symbols on an infinite "tape"
- State transitions, based on current state and symbol

Two Important Concepts

- Universality - All computers can compute the same thing or *all computers can compute exactly the same things, given enough time and memory*
- Layered Abstraction - We can build very complex systems from simple components

Fundamental Ideas Become Road Blocks

Moore's Law

- **Moore's Law** - transistor count (and consequentially computing performance) doubles every two years
- **Physics** - As density increased, power and heat increased. As density increases, there are diminishing returns due to inability to shrink the transistors

MOSFETs are ideal for digital electronics as they can be:

- very efficient switches, consume almost no power in their off state
- provide good conductivity when on
- can be made extremely small, (*measured in 10's nanometers*)

Microprocessor Journey to Date

- Increase speed of microprocessor (*Clock Speed*, kHz -> GHz)
- Increase capability of microprocessor (*more transistors*, 10^3 -> 10^9)
- Increase word size of microprocessor (*8bit -> 32bit -> 64bit*)
- Increase capability of machine instruction (*SSE, MMX*)
- Simplify everything and start all over (*CISC -> RISC*)
- Increase the number of processors (*2 -> 4 -> 8...144*)
- Increase the width of the memory bus (*8-bit -> 4096-bit*)

How do we represent data in a computer?

1. Presence of a voltage – we'll call this state "1"
2. Absence of a voltage – we'll call this state "0"

Text: ASCII Characters, Maps 128 characters to 7-bit binary number, Both printable and non-printable (ESC, DEL, …) characters

Unsigned Integers

An n-bit *unsigned integer* represents 2^n values, from 0 to 2^n-1

- **3-bits:** 2^3 -> 8-1 -> 0-7
- **8-bits:** 2^8 -> 256-1 -> 0-255
- **16-bits:** 2^16 -> 65,536-1 -> 0-65,535

Powers of 2, for 16 bits, 0-15

0	**1**	**2**	**3**	**4**	**5**	**6***	**7**	**8***	**9**	**10***	**11**	**12**	**13**	**14**	**15**
1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16,384	32,768

Signed Integers

- **Positive Integers** - Just like unsigned with zero in most significant bit
- **Negative Integers**, Use *Two's Complement* - for each positive number (X), assign value to its negative (-X), such that X + (-X) = 0 with "normal" addition, ignoring carry out

Number to 2's complement negative number	Subtraction
1. Start with positive number	1. Negate 2nd operand
2. Flip every bit	2. Add two numbers
3. Then add one	Both numbers **must** have same number of digits, replicate MSB if needed
4. Make most significant bit a 1	Ignore Carry Out

Overflow occurs if and only if:

After negation (the operation to make the lower number it's 2's complement)

1. The two operands have the **same sign** (both positive or both negative), **AND**
2. The result has the **opposite sign**

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

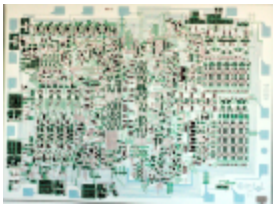
Logic Operations

Logic	Comment	Rules
AND	Useful for clearing bits	**AND** w/ 0 = 0, **AND** w/ 1 = no change
OR	Useful for setting bits	**OR** w/ zero = no change **OR** w/ one = 1
NOT	Unary operation, flips every bit	
XOR	"not equals" test	1 if bits are different, 0 if bits are same

Hexadecimal Notation

Decimal	Binary	Hexadecimal	Decimal	Binary	Hexadecimal
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

Markdown Formatting Basics

Element	Syntax	Raw	Preview
Heading 1	`# Text`	`# Heading`	# Heading
Heading 2	`## Text`	`## Subheading`	## Subheading
Bold	`**text**`	`**bold**`	**bold**
Italic	`*text*`	`*italic*`	*italic*
Code	`` `code` ``	`` `git add -A` ``	`git add -A`
Link	`[text](url)`	`[GitHub](https://github.com)`	GitHub
image	`![text](url)`	`![](./4004_layout.svg)`	

Boolean Algebra

Law	AND Example	OR Example
Idempotent	A AND A = A	A OR A = A
Annulment	A AND 0 = 0	A OR 1 = 1
Identity	A AND 1 = A	A OR 0 = A
Complement	A AND (NOT A) = 0	A OR (NOT A) = 1
Double Complement	NOT(NOT A) = A	
Commutative	A AND B = B AND A	A OR B = B OR A
Distributive	A AND (B OR C) = (A AND B) OR (A AND C)	A OR (B AND C) = (A OR B) AND (A OR C)
Absorptive	A AND (A OR B) = A	A OR (A AND B) = A
Associative	(A AND B) AND C = A AND (B AND C)	(A OR B) OR C = A OR (B OR C)
De Morgan's Theorem	NOT(A AND B) = (NOT A) OR (NOT B)	NOT(A OR B) = (NOT A) AND (NOT B)

Combination Logic

1. **1 to 4 Demultiplexer**
    - data *distributor* - same input, multiple outputs
    - passes 1 input to 1 of 4 outputs
    - register or memory selection with data
2. **2-input Multiplexer**
    - data *selector*, multiple inputs, single output
    - passes 2 inputs to 1 output
    - select a specific clock signal for timing
3. **2 input Decoder**
    - converts a n-bit value to a string of 2^n bits
    - register or memory selection
    - (*yes, a demux w/o the data*)
4. **2 bit Full Adder**
    - n-bit counter
    - program counter
    - divides by 2^n-bits