CSSE2010 Course Notes

Paddy Maher

August 5, 2021

1 Introduction

Following on from the lecture notes as prescribed by the CSSE2010 course, the course notes will be outputting through looking at the levels of abstraction of a computer.

[insert graph of abstraction in computers]

2 Digital logic level

2.1 Bytes and bits

- Computers represent everything in **binary**
- **Bit** = binary digit (0 or 1)
- **Byte** = 8 bits
- Modern computers deal with words which are usually a power of 2 number of bytes. For example:
    - 1, 2, 4 or 8 bytes = 8, 16, 32 or 64 bits

2.2 Representing whole unsigned numbers in binary

2.2.1 Conversion from binary to decimal and vice versa

Converting binary to decimal

- Add values of each position where bit is 1
- Example: 1010011 = 128 + 32 + 4 + 2 + 1 = 167

Converting Decimal to Binary

Method 1

- Rewrite 'n' as the sum of powers of 2 *(by repeating subtracting largest powers of 2 not greater than n)*
- Assemble binary number from 1's in bit positions corresponding to those powers of 2, 0's elsewhere

Method 2

Building up bits from the right (least significant bit *(LSB)* to most *(MSB)*)

- Divide n by 2
- Remainder of division *(0 or 1)* is next bit
- Repeat with n = quotient

2.2.2 Least and Most Significant Bits

The most significiant bit is denoted as **MSB**, likewise, the least significant bit is denoted as **LSB**. These bits can be found on a binary number typically as the leftmost and rightmost bits respectively.

[arrow pointing to example of MSB and LSB in binary number]

2.3 Basic Digital Logic

2.3.1 Digital circuits and Logic gates

Digital Circuits

- Only two logical levels present (i.e. binary)
  - Logic '0'; usually small voltage *(e.g. around 0 volts)*
  - Logic '1'; usually larger voltage *(e.g. 0.8 to 5 volts, depending on the "logic family", i.e. type/size of transistors)*

Logic gates

- Are the building blocks of computers
- Each gate has
  - One or more inputs
  - Exactly one ouput
- Perform logic operations *or functions*
  - 7 basic types: **NOT**, **AND**, **OR**, **NAND**, **NOR**, **XOR XNOR**
  - Inputs and outputs can have only two states **1** and **0**; can be called "true" and "false" respectively.
  - Logic symbol, truth table, boolean expression, timing diagram

2.3.2 Basic Logic Gates

[complete logic gates]

2.4 Boolean Logic

2.4.1 Boolean Logic Functions

- Logic functions can be expressed as expressions involving:
  - variables **(literals)**, e.g A, B, X
  - functions, e.g $+$, $.$, $\oplus$, $\overline{X}$
- Rules about how this works are called **Boolean algebra**
- Variables and functions can only take on values 0 or 1

2.4.2 Boolean Algebra conventions

Conventions

- Inverstion: [insert overline] (overline)
  - e.g. NOT(A) = $\overline{A}$ *(*A bar)
- AND: dot(.) or implied *(*by adjacency)
  - e.g. AND(A,B) = AB = A.B
- OR: plus sign (+)
  - e.g. OR(A,B,C) = A+B+C

Other examples

- XOR(A,B) = A $\oplus$ B = $\overline{A}$B + A$\overline{B}$
- NAND(A,B,C) = $\overline{ABC}$
- NOR(A,B) = $\overline{A+B}$

2.4.3 Summary of logic function representations

There are four representations of logic functions *(assume function of **n** inputs)*

- Truth
  - Lists output for all 2^n combinations of inputs (Best to list inputs in a systematic way)
- Boolean function (or equation)
  - Describes the conditions under which the function output is 1
- Logic Diagram
  - Combination of logic symbols joined by wires
- Timing Diagram

2.4.4 Logic function implementation

- Any logic function can implemented as the **OR** of **AND** combinations of the inputs
  - Called 'sum of products'
- Example:

  - Consider truth table

A	B	C	M
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

  - For each '1' in the output column, write down the **AND** combination of inputs that give 1
  - **OR** these together

Equivalent functions

- Sum of products does not necessarily produce circuit with minimum number of gates
- Can '*manipulater*' boolean functions to give an equivalent function
  - Use rules of boolean algebra
- Example: $\mathbf{Z} = \mathbf{AB} + \mathbf{AC} = \mathbf{A}(\mathbf{B}+\mathbf{C})$

Boolean identities

Name	**AND** form	**OR** form
Identity law	$1\mathbf{A} = \mathbf{A}$	$0 + \mathbf{A} = \mathbf{A}$
Null law	$0\mathbf{A} = \mathbf{A}$	$1 + \mathbf{A} = 1$
Idempotent law	$\mathbf{AA} = \mathbf{A}$	$\mathbf{A} + \mathbf{A} = \mathbf{A}$
Inverse law	$\mathbf{A}\overline{A} = 0$	$\mathbf{A} + \overline{\mathbf{A}} = 1$
Commutative law	$\mathbf{AB} = \mathbf{BA}$	$\mathbf{B} + \mathbf{A} = \mathbf{B} + \mathbf{A}$
Associative law	$(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$	$(\mathbf{A} + \mathbf{B}) + \mathbf{C} = \mathbf{A} + (\mathbf{B} + \mathbf{C})$
Distributive law	$\mathbf{A}+\mathbf{BC} = (\mathbf{A}+\mathbf{B}).(\mathbf{A}+\mathbf{C})$	$\mathbf{A}.(\mathbf{B} + \mathbf{C}) = \mathbf{AB} + \mathbf{AC}$
Absorption law	$\mathbf{A}(\mathbf{A}+\mathbf{B}) = \mathbf{A}$	$\mathbf{A} + \mathbf{AB} = \mathbf{A}$
De Morgan's law	$\overline{\mathbf{AB}} = \overline{\mathbf{A}} + \overline{\mathbf{B}}$	$\overline{\mathbf{A}} + \overline{\mathbf{B}} = \overline{\mathbf{AB}}$

2.4.5 Number bases

[Flow diagram of Binary(base 2), Decimal(base 10), Hex(base 16), Octal(base 8)] [May need to add further details of the table (in regard to the individual binary representations)]

MSB				**LSB**	
$2^{(}n-1)$	$2^{(}n-2)$	..	2^1	2^0	Excess-$2^{(}n-1)$; $-2^{(}n-1) \leq$ x $\leq (2^{(}n-1)-1)$
$-2^{(}n-1)$	$2^{(}n-2)$	..	2^1	2^0	2's comp ; $-2^{(}n-1) \leq$ x $\leq (2^{(}n-1)-1)$
$-2^{(}n-1)-1$	$2^{(}n-2)$	..	2^1	2^0	1's comp ; $-(2^{(}n-1)-1) \leq$ x $\leq (2^{(}n-1)-1)$
+/-	$2^{(}n-2)$	..	2^1	2^0	Sign-Mag ; $-(2^{(}n-1)-1) \leq$ x $\leq (2^{(}n-1)-1)$
$2^{(}n-1)$	$2^{(}n-2)$	..	2^1	2^0	Unsigned ; $0 \leq$ x $\leq 2^n - 1$

[fix peculiar exponential layout]

2.4.6 Equivalent Circuits

- All circuits can be constructed from **NAND** or **NOR** gates
  - These are called 'complete' gates
- Examples: [insert NOT AND OR logic gates]
- Reason: Easier to build **NAND** and **NOR** gates from transistors

2.5 Binary Arthmetic

2.5.1 Binary addition

- Addition is quite simple in binary
-

Addend	0	0	1	1
Augend	0	1	0	1
	0	1	1	0
	0	0	0	1

- Above ignores carry in

[make sense of the table above]

Decimal	8-bit unsigned	Decimal	2's complement
10	00001010	10	00001010
+ 243	+ 11110011	+ (-13)	+ 11110011
253	11111101	-3	11111101

- Format matters upon interpreting the number
- Whatever the format is the bit-wise addition (which leads to the hardware circuit) is the same.
- Two's complement; there is no need to do anything with the carry out from the **MSB** to get the correct result
- But in one's complement, the carry out from the MSB will have to be added back to the result to get the correct answer; this is one drawback of one's complement representation

  Overflow in binary addition

Decimal	8-bit unsigned	Decimal	2's complement
15	00001111	125	01111101
+ 243	+ 11110011	+ 4	+ 00000100
258	00000010	129	10000001

  Overflow: Not enough bits to represent the answer. The result goes out of range thus outputting an incorrect answer.

  - Unisgned: carry-out from the **MSB** $\rightarrow$ overflow

    - 2's comp: carry-in to the **MSB** and carry-out from the **MSB** are different $\rightarrow$ overflow
    - Equivalently, overflow occurs if (in 2's comp)
        * Two negatives added together give a positive, or;
        * Two positives added together give a negative

2.5.2 Adders and addition of binary words

A device which adds 2 bits *(with no carry-in)* is called a 'half-adder' [insert half adder diagrams]

Addition of binary words

- Have to be able to deal with carry-in
-

A	B	Cin	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

- $\mathbf{S} = \overline{\mathbf{AB}}\mathrm{C} + \overline{\mathbf{A}}\mathbf{B}\overline{\mathbf{C}} + ...$
- $\mathbf{S} = \mathbf{A} \oplus \mathbf{B} \oplus \mathbf{Cin}$

[insert full adder diagram with relevant sum and carry-out equation]

Binary Adder

- Can cascade full adders to make binary adder
    - Example: for 4 bits ... [insert 4 bit binary adder diagram]
- This is a ripple-carry adder

2.5.3 Binary subraction

- **A** - **B**; usually implemented as **A** + (-**B**)
    - **A** and **B** are multi-bit quantities
    - '+' in this case means addition (not **OR**)
    - -**B** means negative **B** - the two's complement of **B**
- Two's complement of **B** can be calculated by flipping bits and adding 1.

[add relevant equations and interpretations]

3 Microarchitecture level

4 Instruction set architecture level

5 Assembly language level

6 Problem-oriented language level