Homework 7

Rhea Mae Edwards
Student ID #
932-389-303

1) Arithmetic Logic Units

5.9 32-Bit ALU in Figure 5.15

```
module top(
    input [31:0] A, input Cout,   // input F2
    input [31:0] B,     input [1:0] F,
    ouput reg [31:0] y);

    reg [31:0] B0;
    reg [31:0] B1;
    reg [31:0] add;
    reg [31:0] and;
    reg [31:0] or ;
    reg [31:0] ran;

    assign B0 = ~B;

    assign B1 = F2? B0:B; // assign B1 = F2? B0 : B;

    always @ (*)
      begin
        add = A + B1 + Cout ; // Unsure what F2 does
        and = A & B1;
        or  = A | B;
      end

  // assign ran = whatever this gate does with add...

    always @ (*)
     begin
       case (F)
         0: y = and;
         1: y = or;
         2: y = add;
         3: y = ran;
       endcase
     end

endmodule
```

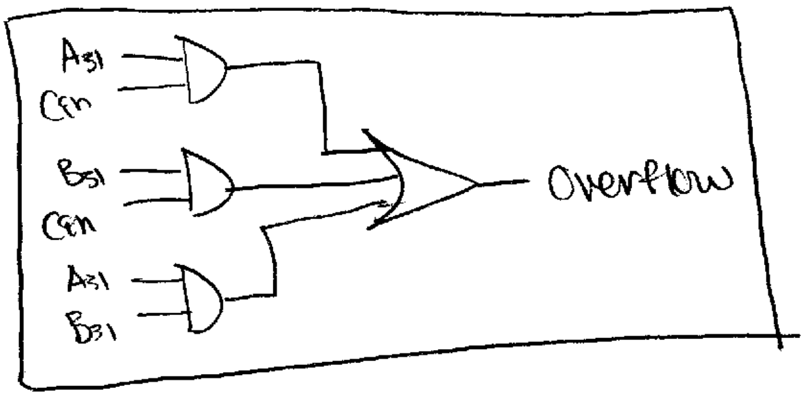
5.10 An Overflow Output, True when Adder overflows, Otherwise False

a.) Boolean Equation

$$\text{Overflow} = A_{31}\overline{B_{31}}C + \overline{A_{31}}B_{31}C + A_{31}B_{31}\overline{C} + A_{31}B_{31}C$$

$$\boxed{\text{Overflow} = ((A_{31}+B_{31})C) + (A_{31}B_{31})}$$

b.) Overflow Circuit



c.) Updated HDL

```
output overflow,
. . .
overflow = (A[31] & Cin) | (B[31] & Cin) | (A[31] & B[31]);
```

5.11 Zero Output, Outputs TRUE when Y == 0

output zero,
. . .
always @ (*)
begin
~~if(A+B+& == 0)~~
zero = 1;
else
zero = 0;
end

if (A == 0)
if (B == 0)
if (C == 0)
zero = 1;

else if (B=1)
if (C=1)
zero = 1,

2) Floating Point

5.25 16-Bit Fixed-Point Sign/Magnitude Format and Hexadecimal Format

a.) - 13.5625

1 ___ ___ 1 1 ___ 1 . 1 ___ ___ 1 ___ ___ ___

16 8 4 2 1 . 1/2 1/4 1/8 1/16 1/32

$\boxed{\text{Sign/Magnitude} = 10001101.10010000}$ 1110010.0110111 +1

1111|0011.0110|1111

$\boxed{\text{Hexadecimal} = F3.6F}$

b.) 42.3125

___ ___ 1 ___ 1 ___ 1 ___ . ___ 1 ___ 1 ___ ___ ___ ___

64 32 16 8 4 2 1 . 1/2 1/4 1/8 1/16 1/32

$\boxed{\text{Sign/Magnitude} = 00101010.01010000}$

1101|0110.1010|1111 11010101.1010111 +1

$\boxed{\text{Hexadecimal} = D6.AF}$

c.) - 17.15625

1 ___ ___ 1 ___ ___ ___ 1 . ___ ___ 1 ___ 1 ___ ___ ___

32 16 8 4 2 1 . 1/2 1/4 1/8 1/16 1/32 1/64

$\boxed{\text{Sign/Magnitude} = 10010001.00101000}$

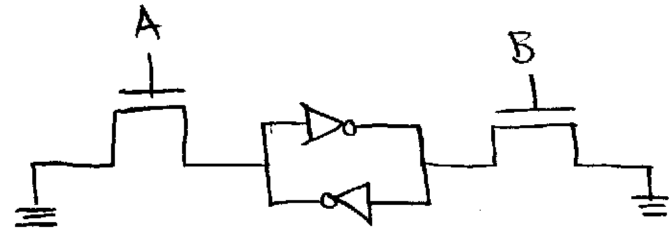
1110|1111.1101|0111 1110110.1101011 +1

$\boxed{\text{Hexadecimal} = EF.D7}$

- Question didn't actually ask to represent the numbers in floating-point format... Exercise 5.29 does though...

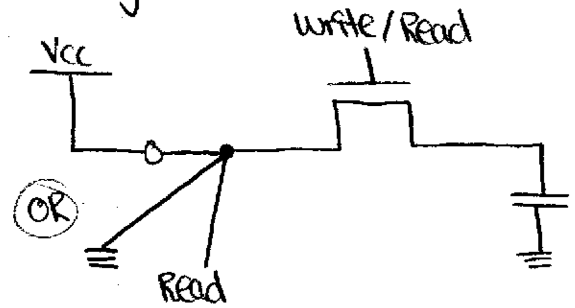
3) Memory

How many transistors are used to store a bit of SRAM?



$\boxed{\text{6 Transistors}}$

How many transistors are used to store a bit of DRAM?



$\boxed{\text{1 Transistor}}$

5) Interview Questions

5.2 Why Processors Might Use Binary Coded Decimal Representation

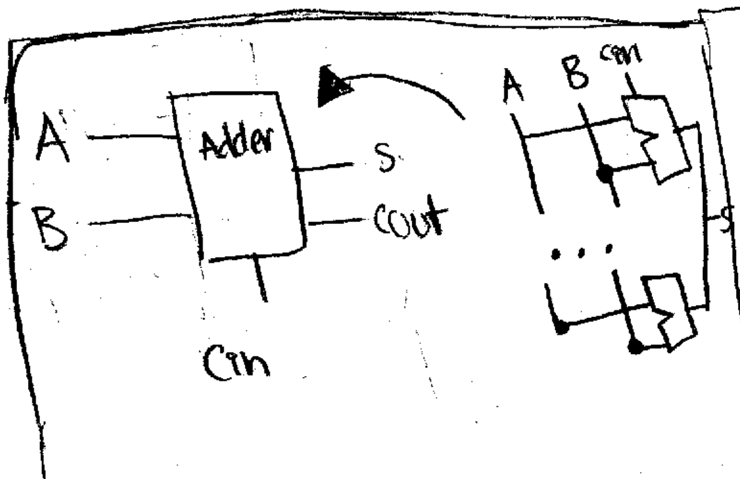
Processors might use binary coded decimal representation, because it may be easier to figure out if a certain operation will cause an overflowing bit, changing the need bits of certain number representations. Also having a number represented in its binary form may be an easier visual and process to describe for certain operations.

5.3 Hardware Add Two 8-Bit Unsigned BCD Numbers

15	0001	0101
+ 23	+ 0010	0011
38	0011	1000

Inputs A^{8}, B^{8}, Cin^{1}

Outputs $S^{8}, Cout^{1}$




```
module bcd_adder(
    input [7:0] A, B,
    input cin,
    output [7:0] S,
    output cout );

    assign S = A + B + Cin;

    always@(*)
      begin
        if((A[7] + B[7])|(A[ ]|B[ ]+Cin
          cout = 1;
      end
endmodule
```

4) Memory

Flash Memory before Your Eyes

Along with humans' simply two characteristics of short-term memory and long-term memory, who knew less complex objects can have many more than two different types of memory itself. There are a various types of memory that can be programmed and built into pieces are hardware such as static random access memory (SRAM), dynamic random access memory (DRAM), and flash memory. Even though SRAM and DRAM have highly interesting aspects to look into, flash memory will be the main focus for now with a bit of comparisons with the other types of memory; so SRAM and DRAM will not be completely forgotten.

So without completely cutting out SRAM and DRAM in this research, flash memory differs from SRAM and DRAM in a sense that it is capable to store memory even when it is not powered by electricity. Flash memory is also known as a solid-state storage device since it is not built with the components of moving pieces of hardware such as with SRAM and DRAM. Flash memory even reads the data it contains differently when it erases and writes than SRAM and DRAM. SRAM and DRAM takes its time and reads content at a byte level which is by one byte at a time, whereas flash memory deals with its information faster by working with entire blocks of memory at a time.[1] Which could have been a reason why it was given the name flash memory, the fastest memory alive; well that may not be true, but when compared only to SRAM and DRAM it can be. It is the thought that counts.

Now focusing more on just flash memory itself, there are even two different types of flash memories- NOR flash memory and NAND flash memory. The most visual difference between NOR flash memory and NAND flash memory is that NOR flash memory is mostly built into microcontrollers, and NAND flash memory are mostly used in pieces of hardware such as USB thumb drives and SD cards which can store large video and audio files. NOR flash memory is slower with its erase and write operations compared to NAND flash memory, and it is larger in size, but it can read data marginally faster than NAND flash memory. Since NOR flash memory is larger in size though, it does have quite an endurance with its read and write cycles; generally it will last longer than NAND flash memory.[1] Switching the main focus a bit from NOR flash memory to NAND flash memory, NAND flash memory can be given the credit why another known name for USB thumb drives is flash drives. But either way, no matter which one is used, they are both still quick little thinkers overall.

[1] "Flash Memory Basics and Its Interface to a Processor." *EE Herald*. Electronics Engineering Herald, 2006. Web. 03 Dec. 2015. <http://www.eeherald.com/section/design-guide/esmod16.html>.