Jacob Reed

TCSS 371: Homework 3

1. 32-bit instruction, 120 opcodes, 80 registers
   1. 2^7 = 128 **= 7 bits**
   2. 2^7 = 128, since 2^6 only covers 64. **7 bits**
   3. Opcode = 7 bits, DR = 7 bits, SR1 = 7 bits, SR2 = 7 bits

7 \* 4 = 28 bits. 32 - 28 = 4.

Unused bits = **4 bits**

1. The execution of the JMP instruction:
   1. Fetch – Loads the instruction from PC into MAR, sends read signal to memory, and then copies contents of MDR into IR.
   2. Decode – First four bits are interpreted as JMP, then decodes the remaining bits into groups, the following 3 bits are 0’s and aren’t used for the JMP instruction, and the 3 bits after that are the register address, followed by the remaining bits which are also 0’s and not used.
   3. Evaluate – Doesn’t require memory access. Skip
   4. Fetch Operands – Fetches x3000 from R3.
   5. Execute – Does nothing for JMP.
   6. Store – Stores R3’s contents (x3000) into PC, so upon next cycle IR receives x3000 from PC.
   7. This instruction will cause an infinite loop until the register or instruction is changed. Looping between x3000 and R3 which points back to x3000.
2. 1. 0001 011 010 1 00000 ;Adds R2’s value into R3 with 0
   2. 1001 011 011 111111 ;Negate R3

0001 011 011 1 00001 ;Convert to 2’s comp

0001 001 010 0 00 011 ;Add numbers and store in R1

* 1. 001 001 001 1 00000 ; ADD R1, R1, #0
  2. 0101 100 100 1 00000 ; AND 0

1. R6 = x123B or 4667

This could be replaced by: **1010 1100 0011 1111 (LDI R6, x3050)**

1. 1. If R1 = x0000 then NOT R1 = xFFFF
   2. If R0 is x0000, then adding R1 to R0 and storing result into R2. R2 = NOT(R1)
   3. Then NOT(R2) will equal what R1 was originally.
   4. **If R1 and R0 are equal then condition code is zero, looping back to x3100**
2. *See conversions.c*