CS 250 Spring 2017 Homework 05 SOLUTION
Due 11:58pm Wednesday, February 22, 2017
Submit your typewritten file in PDF format to Blackboard.
Answers that will be scored are shown in bold. Max score = 15.

1. Why are general purpose registers important?
   Answer: General purpose registers are the fastest form of storage for bit strings in a computer, and they are visible to a programmer, which means that they can be referred to in a program.

2. **[1 pt. each, 2 pts. total]** Refer to textbook Figure 5.9.
   a. What part of the ISA specifies the type of operands? Answer: The instruction. Each instruction in an instruction set specifies the operand type(s) that it requires because the circuit that computes the result of the instruction from its operand(s) is constructed with an assumption as to the type(s) of the operands.
   b. Name an instruction that shows a register can hold a binary string that is interpreted as a (likely) 2's complement integer. **Answer: Any of add, subtract, add immediate, but definitely not add unsigned, subtract unsigned, or add immediate unsigned because instructions clearly do not take 2's complement operands.**
   c. Name an instruction that shows a register can also hold a meaningless binary string. Answer: Any of and, or, and load word.
   d. Name an instruction that overrides the work of the circuit in Figure 6.4. **Answer:** Figure 6.4 computes the default_next_instruction_pointer value. This is **overridden by any of branch equal, branch not equal, jump, jump register, and jump and link.**
   e. Name the instruction that corresponds to the C language code *if(a == b){ }*. Answer: branch not equal. Not equal is the condition for which the code within { } should be skipped.
   f. How many bits are necessary for the opcode field for the instructions in Figure 5.9? Answer: There is a total of 32 instructions in Fig. 5.9, so the opcode field must have at least 5 bits for at least 2^5 = 32 possible patterns of bits.
   g. Using Section 5.22 of the text and Figure 5.11 as guide, write the single assembly language instruction that implements the test in the C language conditional expression code *(a == 0? b = 4 : b = 5)*. Answer:
      set_on_less_than reg_A, 1
      Comment: this instruction works with signed integers because there is an unsigned version of it in Figure 5.11 and because there are separate floating point (FP) instructions. One is the signed integer immediately greater than zero, so setting the immediate = 1 will perform a comparison with identical truth value to performing a==0 directly. Finally, we assume here that reg_A holds the integer a.

3. **[2 pts.]** To obtain the highest performance in an instruction execution pipeline, what operating condition must be maintained?
   Answer: **There should be no stalls**, which can be achieved with straight line execution of independent instructions. Straight line execution of independent instructions is rare in real

programs.

4. **[3 pts.]** Modify Figure 6.2 to support a re-design of the processor of Chapter 6 to support 32 general purpose registers. What significant negative impact does this change have on the expressiveness of the ISA?
Answer: Each of the register pointer fields must now have 5 bits, meaning that 3 bits must be re-purposed from elsewhere in the instruction format, which will reduce the expressiveness of those aspects of the ISA. If these bits are taken in whole or in part from the operation field, then the number of possible operations for this computer will decrease from 32 to 16 (1 bit taken from the field) to 8 (two bits taken) to 4 (all 3 bits taken). Reducing the number of operations makes an ISA less expressive. If, instead, bits are taken from the offset field then the range of offset values decreases by a factor of two for each bit removed. Decreased offset range is also a form of decreased expressiveness of an ISA.

5. **[3 pts.]** What is the cardinality of the set of bit strings that are equivalent within the context of Figure 6.3?
Answer: Figure 6.3 shows a bit string for the instruction `add r4, r2, r3`. From Figure 6.2, the add instruction does not use the 15 bits in the offset field. Therefore, any pattern of 15 bits can be placed in this field without affecting the meaning of the instruction `add r4, r2, r3`. There are 2^15 such strings, thus the cardinality of the set of equivalent bit strings is **2^15 = 32 K strings = 32,768 strings** all meaning `add r4, r2, r3`.

6. Imagine the new instruction SUB, meaning subtract, has been added to the ISA of Figure 6.2. The assembly language instruction SUB R1, R2, R3 is defined as R1 $\leftarrow$ R2 – R3 where R2 is the minuend (the operand being decreased) and R3 is the subtrahend (the amount of the decrease). Write a descriptor in the format shown in Figure 6.2 for all bit strings that mean SUB R1, R2, R3. For any field in the binary representation of this assembly language instruction that is not a unique bit string, write a simple specification of all acceptable bit strings.
Answer: The operation field for SUB is not a unique 5-bit string; it just must not equal a string already in use, namely 00001, 00010, 00011, or 00100. There are 32-4=28 possibilities. The register fields must be correctly specified by their purpose. We are not given which ALU input, reg_A or reg_B, is to be the subtrahend. Let us assume that reg_A is the subtrahend; the answer will change only by swapping the pointers filling the reg_A and reg_B fields. Finally, the offset is not used, so any bit pattern is valid, to be indicated using X, don't care symbols. The result is for [operation][reg_A][reg_B][dst_reg][offset] is
[0, 5-31 as a 5-bit unsigned integer][0011][0010][0001][XXXXXXXXXXXXXXX].

7. **[3 points]** Is Figure 6.9 of a Von Neumann architecture? Why or why not?
Answer: **Figure 6.9 shows the Harvard Architecture, rather than a Von Neumann Architecture because there are separate instruction and data memories.**

8. **[1 pt. each, 2 pts. total]** Make a table showing for each instruction in Figure 6.2 which input (upper or lower (closer to the figure caption) for each the multiplexers M1, M2, and M3 must be selected. If either multiplexer input is a correct selection, then enter X for don't care in that cell of the table.

Answer:

Instruction	M1	M2	M3
Add	Lower; next instruction is at default location	Upper; adding reg_B	Lower; result is from ALU
Load	Lower; same reason as Add	Lower; adding offset	Upper; result is from data memory
Store	Lower; same reason as Add	Lower; adding offset	**X**; result is written to data memory, so M3 does not control write back action
Jump	Upper; must override default next instruction address	Lower; adding offset	**Lower**; next instruction address was computed by ALU

9. How many gates were required for the "bit bucket" in Lab 04?
Answer: Zero. A bit bucket is a place to send unwanted bits. Interestingly, no dedicated hardware is required to accomplish this task, thus no gates are required.