CS 250 Spring 2017 Homework 05
Due 11:58pm Wednesday, February 22, 2017
Submit your typewritten file in PDF format to Blackboard.

1. Why are general purpose registers important?
   General purpose registers are important because they are available to store any transient data required by the program, thus acting as a temporary storage and can store/retrieve data in between successive runs. Overall the use of many general purpose registers can increase the speed at which a chip does work which is very important for large-scale CPUs and other hardware.

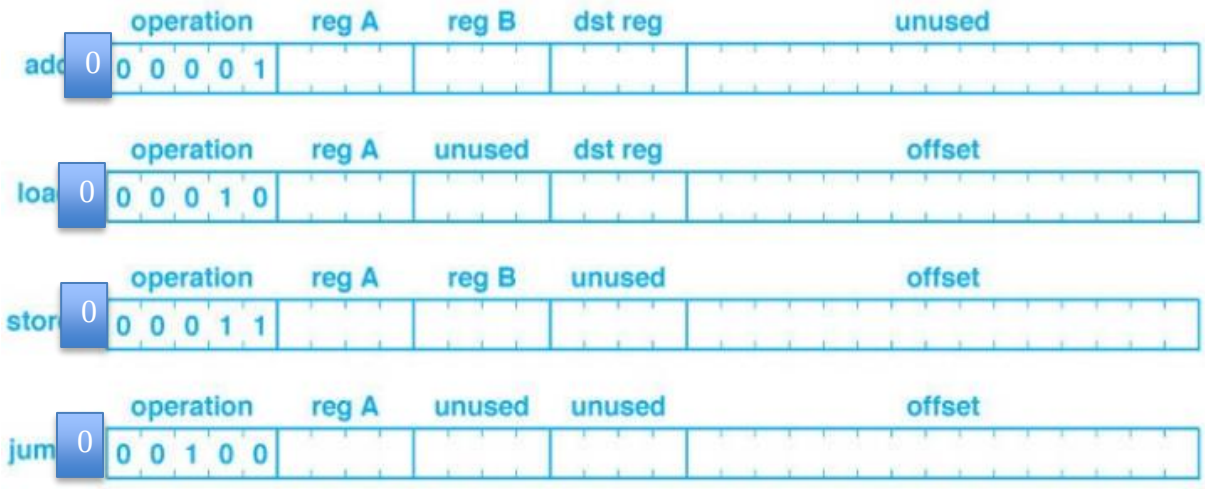
2. Refer to textbook Figure 5.9.
   a. What part of the ISA specifies the type of operands?
      On the left side of the ISA, in between the block lists of the actual instructions, there are category titles that read 'Arithmetic', 'Logical (Boolean)', 'Data Transfer', 'Conditional Branch', and 'Unconditional Branch'. These are the types of operands specified here.
   b. Name an instruction that shows a register can hold a binary string that is interpreted as a (likely) 2's complement integer.
      The instruction 'add' would likely hold a binary string interpreted as 2's complement.
   c. Name an instruction that shows a register can also hold a meaningless binary string.
      The 'store word' instruction, as that binary string will have no meaning stored until it is interpreted again and given meaning.
   d. Name an instruction that overrides the work of the circuit in Figure 6.4.
      The 'add immediate' instruction, as it takes a constant and adds another signed integer, just like the circuit it figure 6.4
   e. Name the instruction that corresponds to the C language code *if(a == b){ }*.
      The 'branch equal' instruction, as this determines if two registers are equal.
   f. How many bits are necessary for the opcode field for the instructions in Figure 5.9?
      6 bits are necessary, as there are 32 individual instructions. These range from the bit string address of '000001' to '100000'.
   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)*.

```
1       cmp     a,0         ; Compare registry holding 'a' with value 0
2       be      isEq        ; Branch to isEq if they are equal
3       mov     b,5         ; Set registry holding 'b' to value 5 (this is the 'else')
4       jmp     Cont        ; Jump to the continue point
5   isEq:
6       mov     b,4         ; Set registry holding 'b' to value 4 (this is the 'then')
7   Cont:
```

3. To obtain the highest performance in an instruction execution pipeline what operating condition must be maintained?

   It is necessary to operate under overlapping conditions. This can be maintained by avoiding any stalls and 'bubbles' in the pipeline. A programmer should avoid introducing any unnecessary branch instructions and perhaps can further optimize the code to delay when references are called rather than immediate calls when they are not needed.

4. 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?



This negatively impacts the expressiveness because now there is an extra bit to manage for the upper 32 general purpose registers. This puts new restrictions on how to call the addresses in memory as they are now different indexes.

5. What is the cardinality of the set of bit strings that are equivalent within the context of Figure 6.3?

   There are three registers of the same *size* bit string, however there are zero equivalent bit strings so the cardinality of equivalent bit strings is: **ZERO**.

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 ← 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.

7. Is Figure 6.9 of a Von Neumann architecture? Why or why not?

   No this is not a Von Neumann architecture. A Von Neumann architecture utilizes the same memory for both data and instructions. This is a **Harvard Architecture** as it has *separate* memory for both instructions and data.

8. 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.

Instruction	**Input M1**	**Input M2**	**Input M3**
add	X	Upper	Upper
load	upper	X	upper
store	X	Lower	Upper
jump	X	Lower	lower

9. How many gates were required for the "bit bucket" in Lab 04?

   Three NAND gates are required in order to implement the bit bucket in Lab 04