# CSE 30321 – Computer Architecture I – Fall 2009
## Homework 01 – Introduction to Programmable Processors
**Assigned:** August 27, 2009 **– Due:** September 8, 2009

## Problem 1: (12 points)
Machine code is a useful abstraction in that it gives you a feel for how some pseudo code might actually be executed on a given microprocessor, yet also abstracts away some of the "gory detail" associated with the process (i.e. a "1s" and "0s" representation). In this question we start with machine code, "look up" in Part A, and "look down" in Part B.

Assume that you have the following sequence of instructions from the 3-instruction processor:

1. MOV 17, R1
2. ADD R1, R1, R1
3. ADD R1, R1, R1
4. MOV R1, 18
5. MOV R2, 18
6. ADD R1, R2, R2
7. MOV 19, R1

Part A: (6 points)
After instructions (1)-(5) have been executed, what has been done?
(please answer in terms similar to: d(x) = d(y) + d(z))

Part B: (4 points)
Write the machine code for each of the 7 instructions listed above.

## Problem 2: (14 points)
The four questions listed below all require short answers. These questions are included to give you more practice working with instruction mnemonics. The primary purpose of these questions is to tie the notion of an "instruction" to a block diagram that is representative of the hardware that's actually necessary to execute an instruction. You might want to refer to the datapath diagram for the 3-instruction and 6-instruction processor in your notes.

Question A: (3 points)
For the 3-instruction processor, is it possible to multiply a number by 16 in less than 10 clock cycles (CCs)? (Note: be sure to include the time to fetch and decode the instruction in your answer.)

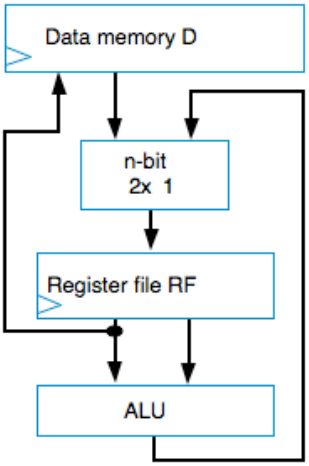
Question B: (4 points)
For the 6-instruction processor, what is the minimum number of clock cycles required to add together (i) the constant 5, (ii) a piece of data in memory, (iii) a piece of data already in a register and then to save the result in 3 different memory locations. Again, include the time spent fetching and decoding.

Question C: (4 points)
For the operation described in Question B, how might you modify the 6-instruction datapath to reduce the number of CCs required? By how much would you improve performance? (Be quantitative and explain.)

Question D: (3 points)
For the 3-instruction processor, how many instructions are required to swap the data in R1 with the data in R4? (Show your sequence of instructions for partial credit.)

**Problem 3: (6 points)**

For the 3-instruction processor, assume that you wanted to have instructions that (i) always adds the constant 1 to a register (i.e. Add R4, R5, #1 or Add R5, R5, #1) and (ii) allowed you to add 2 numbers together in memory and write the result back to register file. How would you modify the 3-instruction datapath so that the least amount of hardware is required? Illustrate below.




**Problem 4: (8 points)**

In CSE 30321 you'll frequently do assignments and readings that are centered on machine instructions. However, it's important to remember that all datapaths do not have the same hardware, all datapaths might not look like that those for the 3-instruciton or 6-instruction processor, etc. By looking at the sequence of instructions that might evolve from pseudo code, you can actually make some inferences as to the capability of the machine itself. The purpose of this question is to get you thinking about this concept.

Let's assume that you have 3 pieces of data – A, B, and C.

- A is stored in memory location #1
- B is stored in memory location #10
- C is stored in memory location #100

Let's also assume that (i) you've written some C code that will multiply A and B, add the result to data in memory location C, and store the final result in C, and (ii) that you compile the C-code on 2 different machines (read: two different computer architectures) and get the sequences of instructions below:

Machine 1:
Load* R1, A
Load R2, B
Load R3, C
Mult R4, R1, R2
Add R3, R4, R3
Store C, R3

Machine 2:
Load R1, A
Mult R2, R1, B
Add R6, R2, C
Store C, R6

*Note: Now instead of MOV, we're using LOAD – essentially this instruction is doing the same thing as the MOV instruction associated with the 3-instruction processor. Only the name is different.

What can you infer about the datapath associated with each processor? (A 2-3 sentence explanation is sufficient.)

**Problem 5: (10 points)**

As you'll see over the course of the semester, knowing something about a processor's architecture can actually help to make you a better programmer. In this question, we look at a simple example as to why this might be true.

Assume that you've written the following C-code:

```
for (i=0; i<10; i++) {
      a = a * 4;
      b = c + d;
}
```

For the processor that you will compile this code on, both the add and the multiply instruction that will result from the above code take 1 clock cycle to fetch and 1 clock cycle to decode. However, it turns out that multiply instructions take 5 clock cycles to execute and add instructions take 1 clock cycle to execute. Why is this? If you look at the Boolean gate representation for a 32-bit multiplier and a 32-bit adder, you'll see that the complexity of the multiplier is greater than that of the adder. As a result, the multiply simply takes longer.

If your compiler **always** uses a multiply instruction if it sees the operation a * b, would it be worthwhile to rewrite your C-code? If so, how? You may assume that a, b, c, and d are all in registers.

(Hint: think about the number of clock cycles it takes to run each instruction).