Winter ’12 CIS 314 Homework 4 – 160/100 points – Due Friday, November 30, 11:59 PM

1. [30] P&H 4.1.1a-4.1.3a, p.410. Assume that the resource blocks are the Instruction Memory, Registers, ALU, and Data Memory.

2. [30] P&H 4.12.1a-4.12.3a, p.419. Assume that the pipeline is full prior to issuing each instruction (for pipelined cases).


4. [30] P&H P&H 4.22.1a-4.22.2a, p.430. Assume that the branch target address is calculated during the IF stage.

5. [+40] Tail recursion is a method for writing simple recursive functions such that the recursive call is the last expression in the body of the function. This way, code can be compiled into assembly without using the stack. Not even the $ra register needs to be stored on the stack, because we can call the function recursively using a j instruction. Note that because of the register conventions, we cannot use the S registers within the function body because they aren’t stored on the stack for later restoration. Tail recursion is obviously more efficient because there’s no function-call overhead, but is limited to recursive functions with a single recursive call in the body of the function. Here is a high-level version of a recursive factorial function written in tail-recursion form:

```plaintext
fact(n) {  
  if (n == 1)  
    return 1;  
  else  
    return n*fact(n - 1);  
}
```

Write a MIPS program which prompts the user for a single positive integer and then calls a function `fact()` written in tail-recursion form. To clarify, you should call your function initially with a jal instruction and your function should ultimately return using a jr $ra instruction, but the function should call itself using a j instruction so that it doesn’t overwrite the return address in $ra. Hint: you’ll probably want to use a second argument register which initially holds the value 1.