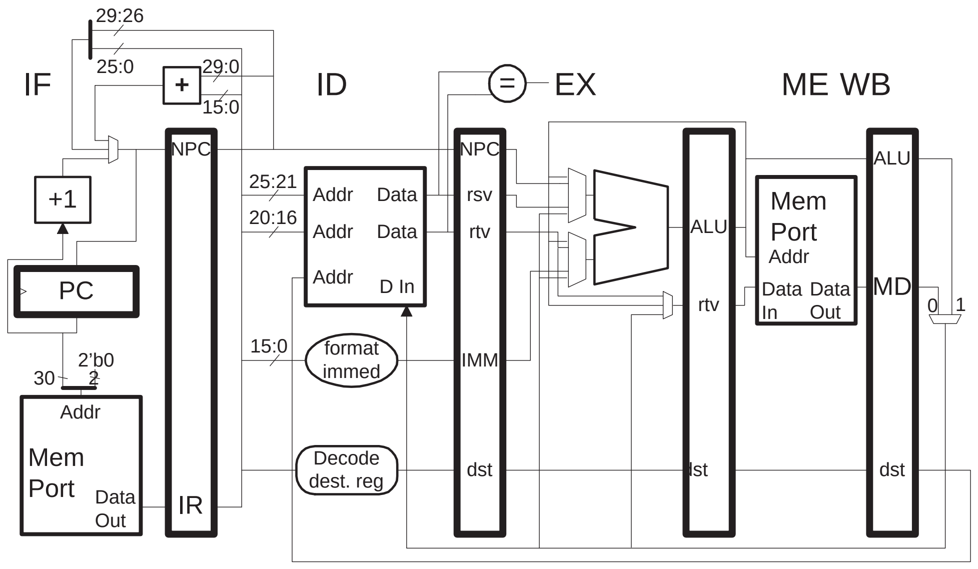
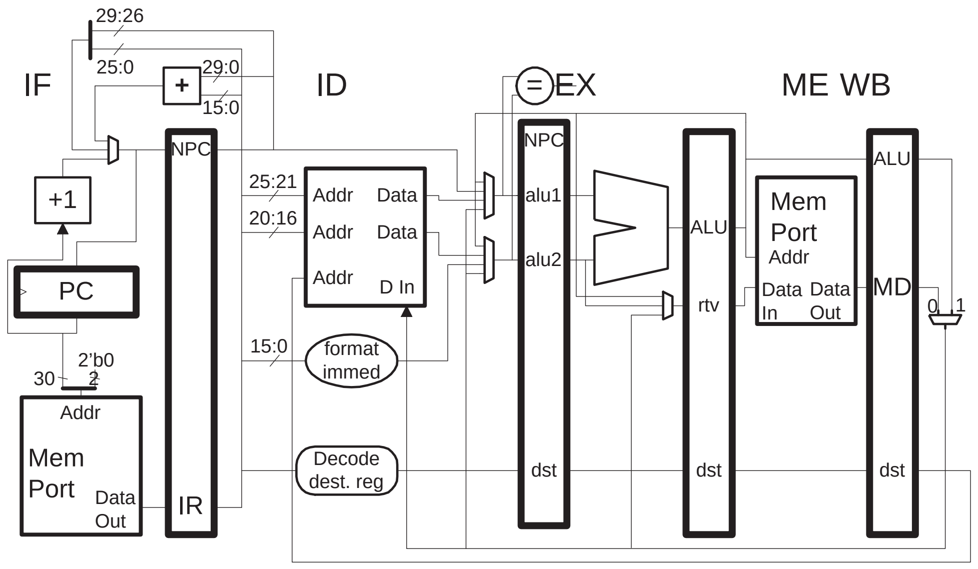
**LSU EE 4720** **Homework 3** Solution **Due: 29 October 2008**

**Problem 1:** Two MIPS implementations appear below, the first is the one presented in class, it will be called the *mux-in-EX implementation*. The second, the *mux-in-ID implementation*, has the ALU input multiplexers in the ID stage, to better balance critical paths. The clock frequency of the mux-in-EX implementation is 1 GHz and the clock frequency of the mux-in-ID implementation is 1.1 GHz.





(*a*) With this change some of the ALU multiplexer inputs are unnecessary. Show which inputs are unnecessary and explain why.

The WB bypass multiplexer inputs are unnecessary because the register file can already bypass from the data in port (which connects to WB) to the data out port.

*Problem continued on next page.*

(*b*) The code below computes the sum of the low 12 bits of elements in an integer array. Compute the performance, in array elements per second, of this code for both the mux-in-EX system and the mux-in-ID system. Assume that the array size is large and that the number of array elements is even.

Note that the code computes two array elements per loop iteration. The solution strategy is to determine the number of cycles per iteration, then use the clock frequency to compute the performance in array elements per second.

The pipeline diagrams appear below. Execution is shown until a repeating pattern is encountered (by examining the pipeline state present at the first instruction in an iteration). For the mux-in-EX system there are no stalls, the mux-in-ID system has several stalls.

The code for the mux-in-EX system enjoys smooth, stall-free execution and so takes 8 cycles per iteration, 4 cycles per array element, and computes at a rate of $\frac{1.0\times10^9}{\frac{1}{2}\times(8-0)} = 250\times10^6$ array elements per second.

The code for the mux-in-ID system suffers dependence stalls. From the pipeline execution diagram below one can see that it takes $23-11=12$ cycles per iteration or 6 cycles per element. It computes at a rate of $\frac{1.1\times10^9}{\frac{1}{2}\times(23-11)} = 183.3\times10^6$ elements per second. The benefit of the higher clock frequency has been undermined by the stalls *for this code.*

```
# Performance on mux-in-EX system

LOOP:   # Cycle      0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
lw $t0, 0($a0)       IF ID EX ME WB
lw $t5, 4($a0)          IF ID EX ME WB
andi $t2, $t0, 0xfff       IF ID EX ME WB
add $v0, $v0, $t2             IF ID EX ME WB
andi $t7, $t5, 0xfff             IF ID EX ME WB
add $v0, $v0, $t7                   IF ID EX ME WB
bne $a0, $t1  LOOP                     IF ID EX ME WB
addi $a0, $a0, 8                          IF ID EX ME WB
# Cycle              0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
lw $t0, 0($a0)                               IF ID EX ME WB

# Performance on mux-in-ID system

LOOP:   # Cycle      0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
lw $t0, 0($a0)       IF ID EX ME WB
lw $t5, 4($a0)          IF ID EX ME WB
andi $t2, $t0, 0xfff       IF ID -> EX ME WB
add $v0, $v0, $t2             IF -> ID -> EX ME WB
andi $t7, $t5, 0xfff                IF -> ID EX ME WB
add $v0, $v0, $t7                         IF ID -> EX ME WB
bne $a0, $t1  LOOP                           IF -> ID EX ME WB
addi $a0, $a0, 8                                   IF ID EX ME WB
# Cycle              0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
lw $t0, 0($a0)                                        IF ID -> EX ME WB
lw $t5, 4($a0)                                           IF -> ID EX ME WB
andi $t2, $t0, 0xfff                                           IF ID -> EX ME WB
add $v0, $v0, $t2                                                 IF -> ID -> EX ME WB
andi $t7, $t5, 0xfff                                                    IF -> ID EX ME WB
add $v0, $v0, $t7                                                             IF ID -> EX ME WB
bne $a0, $t1  LOOP                                                               IF -> ID EX ME WB
addi $a0, $a0, 8                                                                       IF ID EX ME WB
# Cycle              0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
lw $t0, 0($a0)                                                                            IF ID -> EX..■
```

(*c*) If, after double-checking your work, the performance of the mux-in-ID system is faster than the old mux-in-EX system inform the professor that there is a mistake in this problem. Otherwise, schedule (re-arrange instructions) the code above so that it performs faster (while still performing the same computation) on the mux-in-ID system.

The solution appears below. The instructions can easily be rearranged to avoid the stalls. Now the system computes at a rate of 275 million array elements per second, outperforming the mux-in-EX system.

```
LOOP:   # Cycle      0  1  2  3  4  5  6  7  8  9  10 11 12
lw $t0, 0($a0)       IF ID EX ME WB
lw $t5, 4($a0)          IF ID EX ME WB
addi $a0, $a0, 8           IF ID EX ME WB
andi $t2, $t0, 0xfff          IF ID EX ME WB
andi $t7, $t5, 0xfff             IF ID EX ME WB
add $v0, $v0, $t2                   IF ID EX ME WB
bne $a0, $t1  LOOP                     IF ID EX ME WB
add $v0, $v0, $t7                         IF ID EX ME WB
# Cycle              0  1  2  3  4  5  6  7  8  9  10 11 12
lw $t0, 0($a0)                                IF ID EX ME WB
```

**Problem 2:** You are in an alternate universe where you work for MIPS at a time when its first implementation (mux-in-EX) has been very successful and is in the hands of customers of all types. You are deciding on whether to make mux-in-ID the second implementation to be marketed.

(*a*) What role do compiler writers have in the success of mux-in-ID? Explain.

They must be able to write optimizers that can successfully schedule the code to avoid the "new" stalls. A compiler writer of average skill should be able to schedule away the stalls in the sample code above. Other situations are more difficult.

(*b*) If mux-in-ID is faster than mux-in-EX using the old compilers, do compilers still need to be re-written? Explain.

Yes. The old code might be faster because fewer than 10% of instructions use a source register produced by the immediately preceding instruction. Suppose that number were 5%. Then re-writing the compiler could reduce or eliminate stalls due to these instructions, yielding further performance gains.