LSU EE 4720 Homework 3 Solution Due: 20 March 2006

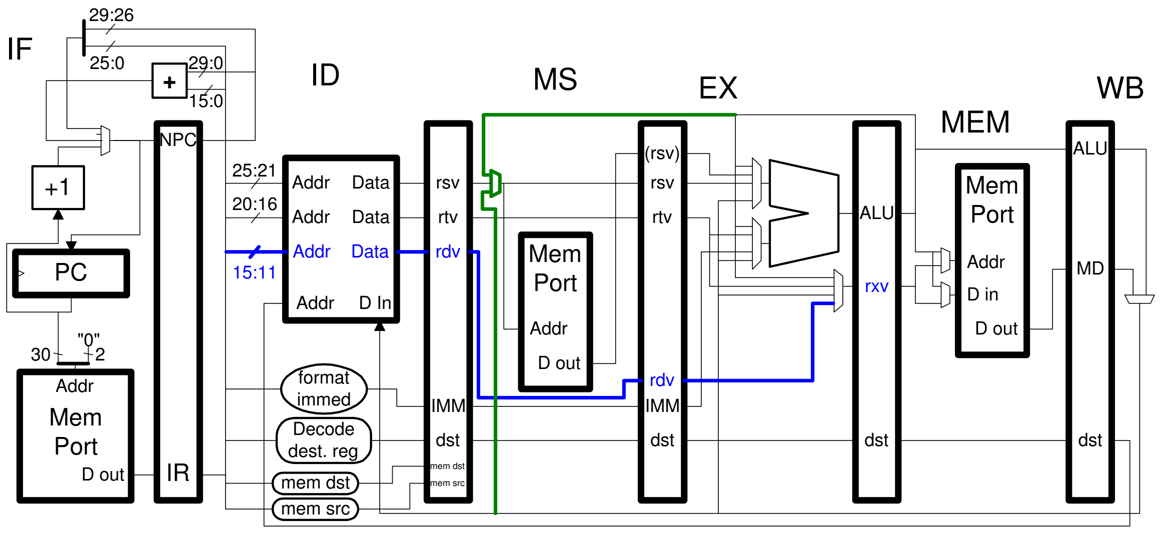
Review Fall 2004 Final Exam Problem 2, which was discussed in class on Monday, 13 March 2006.

Problem 1: Using the solution to Fall 2004 Final Exam problem 2 parts a, b, and d (but not c) as a starting point, make changes to implement a new two-source register MM instruction `add.mmr` which operates as shown in the example below. *Hint: The solution requires a register file modification.*

```
add.mmr   (r1), (r2), r3     # Mem[r1] = Mem[r2] + r3
```

Solution shown below. The `add.mmr` instructions use three register source operands and so a third read port must be added to the register file, that is shown below in blue. The "new" register value, `rdv`, is used as the store address for the sum.

The diagram below also shows, in green, the bypass connections used in the solution to Problem 3 (but not the branch condition bypasses used in Problem 4).



Problem 2: Your boss, a stuck-in-the-twentieth-century RISC true believer who only grudgingly agreed to include `add.mm`, `add.mr`, `add.rm`, and `add.mmr` in MMMIPS, flies into an incoherent rage when you suggest also adding `add.mmm` to MMMIPS. What pushed your boss over the edge? (That is, why is `add.mmm` much harder to add to the implementation in the Fall 2004 exam than `add.mmr`.) Instruction `add.mmm` operates as shown below:

```
add.mmm   (r1), (r2), (r3)     # Mem[r1] = Mem[r2] + Mem[r3]
```

Unlike the other memory-memory instructions, `add.mmm` must read two source operands from memory. To do that without stalling the pipeline would require a second memory port in the MS stage, which is expensive. The alternative is having `add.mmm` spend two cycles in MS, but that would mean stalling the pipeline which is not something you want to do for reasons other than dependencies.

Problem 3: Write a pair of programs intended to show the benefit of MMMIPS. Both programs should do the same thing, program A should use ordinary MIPS instructions and run on the MIPS pipeline shown below. Program B should use MMMIPS instructions and run on the implementation shown in the exam solution. Reasonable bypass connections may be added, including those needed for branches.

(a) Show the programs.

Two pairs of programs are shown below, each program adds 7 to all the elements of an array. Program A-1 has 5 instructions in a loop body and executes at 1.2 CPI; program B-1 has 3 instructions and executes at 1.67 CPI. Since the programs are different (albeit accomplishing the same thing) CPI cannot be used to compare them. Instead, performance will be measured in cycles per element. (Think of it as execution time divided by the number of iterations in the array.) Both programs handle one element per iteration, A-1 completes an iteration in 6 cycles and B-1 completes an iteration in 5 cycles, so that B-1 is faster despite having a higher CPI.

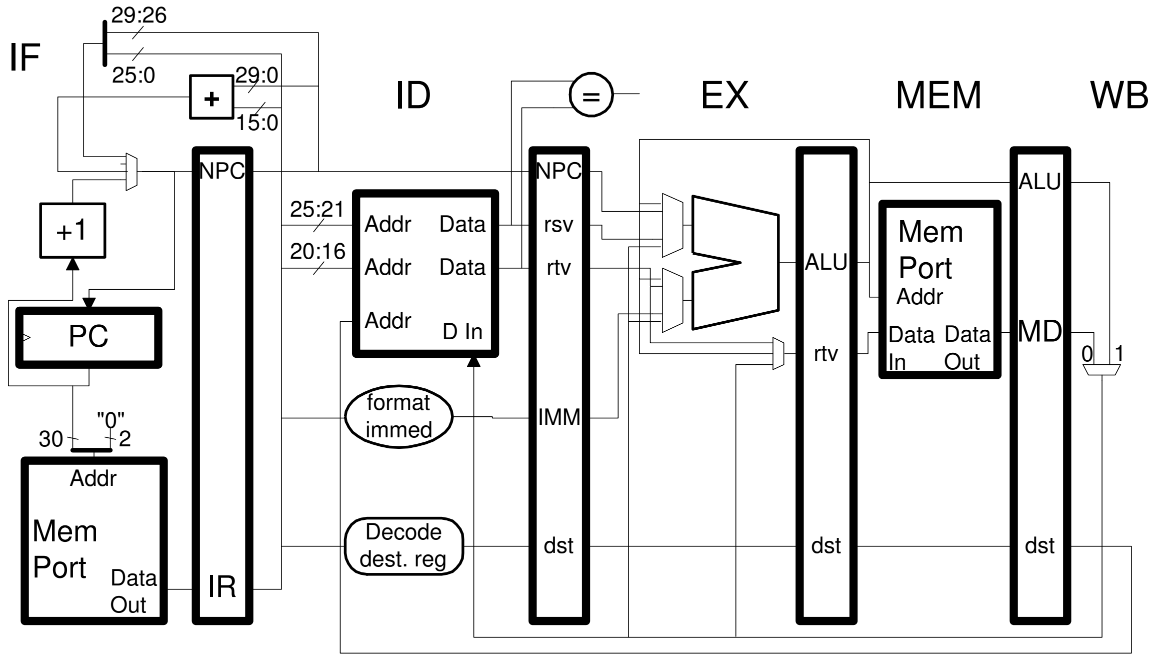
(Program A-1 stalls in cycle 5 so that the branch can get r2; program B-1 stalls in 5 so that `add.mm` can get r2 and again in cycle 7 so the branch can get r2.)

Programs A-2 and B-2 perform the same function but operate on two elements per iteration (using a technique called loop unrolling). This eliminates all stalls and so both run at a CPI of 1, however the margin of CPE of B-2 over A-1 is now higher (since B-2 had more stalls to eliminate).

Both pairs of programs show an advantage for MMMIPS.

(b) Compute the execution time (in cycles) of each program. The comparison should be fair so each program should be producing the same result.

See above.



Programs on next page.

```
## A-1: Regular MIPS. Loop handles one element per iteration.
 #
 # Cycles per instruction: 6 / 5 = 1.2
 # Cycles per element:     6 / 1 = 6
 #
LOOP:
# Cycle               0  1  2  3  4  5  6  7  8  9  10
 lw r1,0(r2)          IF ID EX ME WB
 addi r2, r2, 4          IF ID EX ME WB
 addi r1, r1, 7             IF ID EX ME WB
 bneq r2, r9, LOOP             IF ID -> EX ME WB
 sw r1, -4(r2)                    IF -> ID EX ME WB
 # Second iteration below.
 lw r1,0(r2)                            IF ID EX ME WB
 addi r2, r2, 4                            IF ID EX ME WB
# Cycle               0  1  2  3  4  5  6  7  8  9  10


## B-1: Memory-memory MIPS (MMMIPS). Loop handles one element per iteration.
 #
 # Assumes bypass from ME to MS to avoid stalls. (Not on exam soln.)
 #
 # Cycles per instruction: (8-3)/3 = 5/3 = 1.67
 # Cycles per element:     (8-3)/1 = 5/1 = 5
 #
LOOP:
# Cycle               0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17
 add.mm (r2),(r2), 7  IF ID MS EX ME WB
 bneq r2, r9, LOOP       IF ID MS EX ME WB
 addi r2, r2, 4             IF ID MS EX ME WB
 # Second iteration below.
 add.mm (r2),(r2), 7           IF ID -> MS EX ME WB
 bneq r2, r9, LOOP                IF -> ID -> MS EX ME WB
 addi r2, r2, 4                         IF -> ID MS EX ME WB
 # Third iteration below.
 add.mm (r2),(r2), 7                          IF ID -> MS EX ME WB
 bneq r2, r9, LOOP                               IF -> ID -> MS EX ME WB
 addi r2, r2, 4                                        IF -> ID MS EX ME WB
# Cycle               0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17
```

More programs on next page.

```
## A-2: Regular MIPS. Loop handles two elements per iteration.
 #
 # Cycles per instruction: 8 / 8 = 1
 # Cycles per element:     8 / 2 = 4
 #
LOOP: # 8 / 2 = 4
# Cycle           0  1  2  3  4  5  6  7  8  9  10
 lw r1,0(r2)      IF ID EX ME WB
 lw r11,4(r2)        IF ID EX ME WB
 addi r2, r2, 8         IF ID EX ME WB
 add r1, r1, 7             IF ID EX ME WB
 add r11, r11, 7              IF ID EX ME WB
 sw r1, -8(r2)                   IF ID EX ME WB
 bneq r2, r9, LOOP                  IF ID EX ME WB
 sw r1, -4(r2)                         IF ID EX ME WB
 # Second iteration below.
 lw r1,0(r2)                              IF ID EX ME WB
# Cycle           0  1  2  3  4  5  6  7  8  9  10 11
```

```
## B-2: Memory-memory MIPS (MMMIPS). Loop handles two elements per iteration.
 #
 # Assumes bypass from ME and WB to MS to avoid stalls. (Not on exam soln.)
 #
 # Cycles per instruction: 5/5 = 1
 # Cycles per element:     5/2 = 2.5
 #
LOOP: # CPE:  5/2 = 2.5
# Cycle               0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17
 add.mm (r2),(r2), 7  IF ID MS EX ME WB
 add.mm (r12),(r12), 7   IF ID MS EX ME WB
 addi r2, r2, 8             IF ID MS EX ME WB
 bneq r12, r9, LOOP            IF ID MS EX ME WB
 addi r12, r12, 8                 IF ID MS EX ME WB
 # Second iteration below.
 add.mm (r2),(r2), 7                 IF ID MS EX ME WB
 add.mm (r12),(r12), 7                  IF ID MS EX ME WB
 addi r2, r2, 8                            IF ID MS EX ME WB
 bneq r12, r9, LOOP                           IF ID MS EX ME WB
 addi r12, r12, 8                                IF ID MS EX ME WB
# Cycle               0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17
```

Problem 4: Show a program that will run slower on the MMMIPS implementation that the ordinary MIPS implementation. That program, of course, should not use MMMIPS instructions. Reasonable bypass connections can be added, including those needed for branches. *Hint: Branches are important.*

The program is a single loop, the key feature being that the branch depends upon the immediately preceding instruction. It is assumed that the implementations have a bypass from ME to ID. With this the MIPS only needs one stall for the branch to resolve the condition. MMMIPS needs two stalls because its EX stage is one stage more distant from ID.

```
## Program on regular MIPS.
 #
 # Bypass from ME to ID for branches assumed.
 #
 # Cycles per instruction: 4 / 3 = 1.333
 #
 #
LOOP:
# Cycle             0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
 xor r1, r1, r2     IF ID EX ME WB
 bneq r1, r3, LOOP     IF ID -> EX ME WB
 sll r1, r1, 3            IF -> ID EX ME WB
 # Second iteration.
 xor r1, r1, r2                 IF ID EX ME WB
 bneq r1, r3, LOOP                 IF ID -> EX ME WB
 sll r1, r1, 3                        IF -> ID EX ME WB
# Cycle             0  1  2  3  4  5  6  7  8  9  10 11 12 13 14

## Program on MMMIPS.
 #
 # Bypass from ME to ID for branches assumed.
 #
 # Cycles per instruction: 5 / 3 = 1.667
 #
# Cycle             0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
 xor r1, r1, r2     IF ID MS EX ME WB
 bneq r1, r3, LOOP     IF ID ----> MS EX ME WB
 sll r1, r1, 3            IF ----> ID MS EX ME WB
 # Second iteration.
 xor r1, r1, r2                    IF ID MS EX ME WB
 bneq r1, r3, LOOP                    IF ID ----> MS EX ME WB
 sll r1, r1, 3                           IF ----> ID MS EX ME WB
# Cycle             0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
```