

Comp 120 Computer Organization
Spring 2005

Solutions to Problem Set #8

Problem 1. “Delayed Decision” [35 points]

(A)

PCSEL	0 if Z, 1 otherwise
RGEN	0X
SEXT	X
BSEL	0
WDSEL	1
ALUFN	0XX01
Wr	0
WERF	1
ASEL	0

(B) We cannot subtract \$rs from \$rt using the current datapath. It would have to be modified to support this “reverse” subtraction.

(C) Steps sum1 takes is $(2+3N+2)$ while sum2 takes $(3+2N+2)$. For sum2 to be at least 25% faster than sum1, N must be no less than 8.

$$\frac{(3N + 4) - (2N + 5)}{3N + 4} \geq 0.25 \Rightarrow N \geq 8$$

(D) The branch decision is made at ALU stage. A straightforward implementation requires 2 delay slots. We need an adder and a comparator to compute the early branch decision. This implementation is complex when compared to bne and beq. Also it is time consuming.

(E)

Standard:

```
addi $sp, $sp, -24
addi $t0, $t0, 0    //sum stored in $t0
addi $t1, $t1, 0    // i = 0
addi $t2, $0, N     // N stored in $t2
slt  $t2, $t1, $t2
beq  $t2, $0, end
loop: sll $t1, $t1, 2
      lw  $t3, x($t1)
      add $t0, $t0, $t3    //sum = sum + x[i]
      addi $t1, $t1, 1    // i++
      addi $t2, $0, N     // N stored in $t2
      slt  $t2, $t1, $t2
      bne $t2, $0, loop
end:  addi $sp, $sp, 24
```

```

Abnz version:
    addi $sp, $sp, -24
    addi $t0, $t0, 0    //sum stored in $t0
    addi $t1, $t1, 0    // i = 0
    addi $t2, $0, N
    beq $t2, $t1, end
loop: subi $t2, $0, N // -N stored in $t2
    sll $t1, $t1, 2
    lw $t3, x($t1)
    add $t0, $t0, $t3    //sum = sum + x[i]
    addi $t1, $t1, 1    // i++
    abnz $t1, $t2, loop
end: addi $sp, $sp, 24

```

Problem 2. “Flexible Pipes” [35 points]

(A) In the original pipelined miniMIPS it will take 5 clock periods (T) to complete the first add and then 999 T to complete the rest. Thus the total time to process 1000 adds will be 1004T. In the modified miniMIPS it will take 4T for the first add and 999T for the rest. So the total time for this ‘improved’ version to complete 1000 adds will be 1003T, not much improvement over the regular pipelined miniMIPS.

(B) One instruction per clock period T.

(C) If an instruction that uses all 5 stages (e.g. lw/sw) is right before an instruction which only uses 4 stages (e.g. add/sub), then the second instruction will have to be stalled.

For example:

```

    lw $t0, x
    add $t1, $t2, $t3

```

To execute this sequence correctly the pipeline diagram must look like this:

Pipe Stage	t1	t2	t3	t4	t5	t6
IF	lw	add				
RF		lw	add			
ALU			lw	add	add	
MEM				lw		
WB					lw	add

The stall happens when the add instruction does not move to the next stage between t4 and t5.

(D) If all we ever executed were single 4-stage instructions such as add, then Bud’s idea would improve performance by 20%. In reality, however, we execute programs with many instructions. If these instructions are all 4-stage instructions, then, as shown in part (A) the performance improvement is insignificant. If these instructions are

intermixed 4-stage and 5-stage instructions, then, as part (C) showed, there isn't going to be any performance improvement.

Problem 3. "Stage Three" [30 points]

(A) `addi $at, $0, offset`
`lw $t0, ($at)`

`addi $at, $0, offset`
`sw $t0, ($at)`

(B) One bypass path from the output of WDSSEL MUX to the input of ASEL MUX, and WDSSEL MUX to the input of BSEL MUX.

The following sequence uses both two bypass paths:

`lw $t0, ($at)`
`add $t1, $t0, $t0`

We need one more bypass path to support jal operation, which comes from PC^{REG} to the inputs of ASEL and BSEL MUXs:

`jal loop`
`noop`
`add $t0, $t1, $t1`

(C) It does not require interlock because memory instruction (lw/sw) is only offset from the next instruction by one step.