The UNIVERSITY *of* NORTH CAROLINA *at* CHAPEL HILL

Comp 411 Computer Organization
Spring 2012

Problem Set #5 Solutions

Problem 1. "Simplified Shifts"

a) Since Lori's implementation is intended to replace the original MIPS shift instructions, only minor changes are needed. Instead of feeding bits 25:21 into the ASEL mux, bits 10:6 would be used. Also, the constant '16' no longer needs to be input to the ASEL mux. The Control Logic would need to be slightly modified in order to correctly support the new instructions.

b) Since the variable shift instructions are separate instructions from the shamt shift instructions, they will still work as normal. Since they use values from registers, control will set the ASEL mux to 0 (as they were originally), while Lori's instructions will have ASEL set to 1.

c) No, the modifications in part a) are sufficient.

d) The new `lsi` instruction can load the given immediate value into any part of the word. The old `lui` instruction is a subset of `lsi`, as its functionality can be completely emulated.

e) It does not impact the data path at all, as the hardware to sign-extend the immediate value is already present. By using a signed immediate value, the resulting values would be both positive and negative. This could be helpful when computing memory offsets.

Problem 2. "Out of Control"

Opcode	PCSEL	WASEL	SEXT	BSEL	WDSEL	ALUFN				Wr	WERF	ASEL
						Sub	Bool	Shft	Math			
subu	0	0	-	0	1	1	-	0	1	0	1	0
xor	0	0	-	0	1	-	00	0	0	0	1	0
addiu	0	1	1	1	1	0	–	0	1	0	1	0
sll	0	0	-	0	1	-	00	1	0	0	1	1
andi	0	1	0	1	1	-	00	0	0	0	1	0
lw	0	1	1	1	2	0	–	0	1	0	1	0
sw	0	1	1	1	-	0	–	0	1	1	0	0
j	2	-	-	-	-	-	–	-	-	0	0	-
jal	2	2	-	-	0	-	–	-	-	0	1	-
lui	0	1	-	1	1	-	00	1	0	0	1	2

Problem 3. Delayed Decisions

(A) What instruction format would the `abnz` instruction use?

It should be encoded as an I-format

(B)

PCSEL	if Z 0 else 1
WASEL	1
SEXT	1
BSEL	0
WDSEL	1
ALUFN	Sub = 0 / Bool =XX / Shft=0 / Math=1
Wr	0
WERF	1
ASEL	0

(C)

We cannot subtract $rs from $rt using the current datapath.
It would have to be modified to support this "reverse" subtraction.

(D)

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.

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

(E)

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. It also requires more complex (slower) logic.

(F)

```
standard:
      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:


abnzversion:
      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:
```

Problem 4. Flexible Pipes

(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 5. Stage Three

(A)

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

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

(B)

One bypass path from the output of WDSEL MUX to the input of ASEL MUX, and WDSEL 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
nop
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.