Computer Architecture — Due on Friday, March 20, 2015

Assignment 3

CS3350B — *University of Western Ontario*

Submission instructions. With each assignment, you are required to hand in an Assignment Submission Form, on which you certify that the material you've handed in is exclusively your own work. Put this form inside a 9-by-12-inch envelope along with your assignment, bearing your name and the course number CS3350b, which will be used to return your assignment. Hand in your envelope (contained with your completed assignment and the Assignment Submission Form) in class, or drop it in the CS3350b locker (locker #308, on the third floor of the Middlesex College building) by the midnight on the due date.

PROBLEM 1. [25 points] We would like to design a *switching network* that has four 32-bit input words `a`, `b`, `c` and `d`, and four 32-bit output words `z`, `y`, `x` and `w` as well as a 2-bit control input `s`. Figure 1 specifies how the switching network passes the input words to the output depending on the value of the control input.

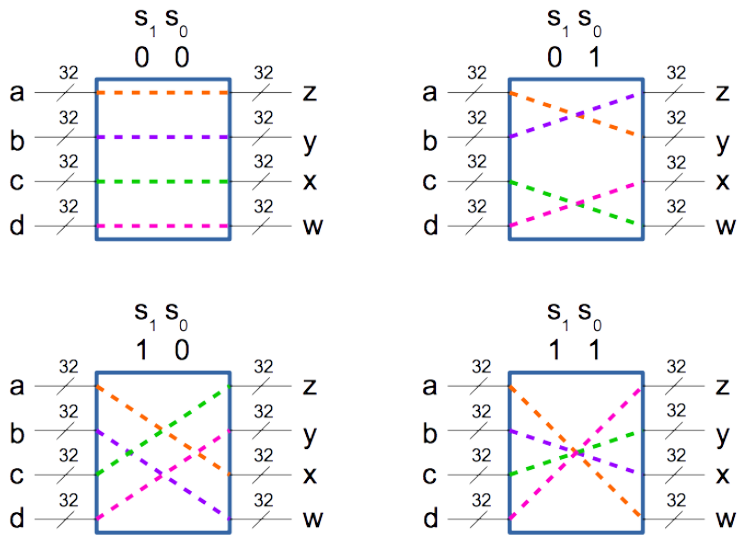

Figure 1: The switching network with the combination values of `s` = s_1 s_0.

1.1 Write the Boolean expressions for `z`, `y`, `x` and `w`, respectively. These expressions are functions of `a`, `b`, `c`, `d`, s_1 and s_0.

s_1	s_0	z	x	y	w
0	0	a	b	c	d
0	1	b	a	d	c
1	0	c	d	a	b
1	1	d	c	b	a

$z = \overline{s_1} \cdot \overline{s_0} \cdot a + \overline{s_1} \cdot s_0 \cdot b + s_1 \cdot \overline{s_0} \cdot c + s_1 \cdot s_0 \cdot d$
$y = \overline{s_1} \cdot s_0 \cdot a + \overline{s_1} \cdot \overline{s_0} \cdot b + s_1 \cdot s_0 \cdot c + s_1 \cdot \overline{s_0} \cdot d$
$x = s_1 \cdot \overline{s_0} \cdot a + s_1 \cdot s_0 \cdot b + \overline{s_1} \cdot \overline{s_0} \cdot c + \overline{s1} \cdot s0 \cdot d$
$w = s_1 \cdot s_0 \cdot a + s_1 \cdot \overline{s_0} \cdot b + \overline{s_1} \cdot s_0 \cdot c + \overline{s_1} \cdot \overline{s_0} \cdot d$

1.2 Draw the gate diagram with hierarchical multiplexers, as on Slide 8 of the PDF version of the set of Slides 5.4. **Note**: a given input signal (either a, b, c or d) may appear at several places on the diagram without wires connecting them. This will keep diagram gates less messy.

See the network.circ.

PROBLEM 2. [25 points] Consider the data-path shown in Figure 2, which is taken

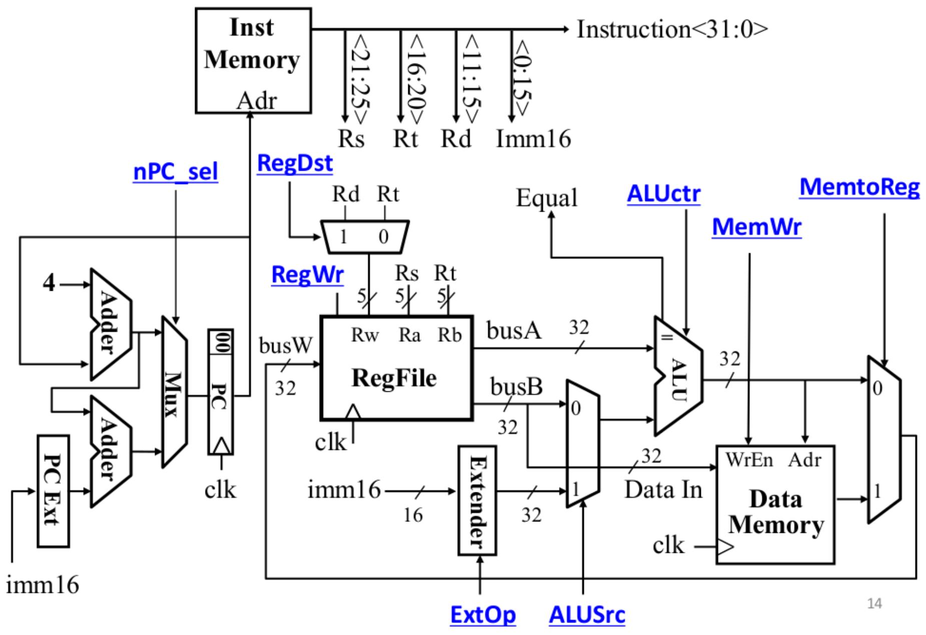

Figure 2: The basic implementation of the data-path and control for a `MIPS` instruction.

from Slide 14 of the PDF version of the set of Slides 5.7. In this problem, we study the following `MIPS` instruction: `sll $s0, $s1, 2`. Note: this datapath can decode both R-type and I-type `MIPS` instructions. For R-type, the `Rd`, `Shamt` and `Funct` fields are encoded in the `Imm16`. Moreover, we shall assume that `Reg[Rd]` can be read respectively from `busA`.

2.1 Describe the five stages of the data-path for this instruction.

Stage 1: fetch `sll` *instruction, increment PC*
Stage 2: decode to determine it is the shift left operation, then read register $s1 to `rt` *and set* `imm16` *to 2*
Stage 3: shift the value of $s1 by 2 bits in the ALU unit
Stage 4: idle (nothing to write to memory)
Stage 5: write the result of Stage 3 into register $s0

2.2 Which blocks in Figure 2 perform a useful function for this instruction?

PC, Inst memory, RegFile and ALU.

2.3 Which blocks in Figure 2 produce outputs that are not used for this instruction and which blocks produce no outputs for this instruction?

Data memory produces outputs that are not used.

2.4 What are the values of the control signals generated by the controller in Figure 2 for this instruction? Review the set of Slides 5.7 for this question.

nPC_sel = 0, RegDst = 1, RegWr = 1, ExtOp = x, ALUSrc = 1, ALUctr = "sll", MemWr = 0, and MemtoReg = 0.

PROBLEM 3. [20 points]

In this exercise, we consider a new MIPS instruction which is specified in the RTL language:

```
Reg[Rd] ← Mem[Reg[Rt]+Reg[Rs]];

PC ← PC + 4;
```

3.1 Which existing blocks in Figure 2 (if any) can be used for this instruction?

All of them.

3.2 Which new functional blocks (if any) do we need for this instruction?

None.

3.3 What new signals do we need (if any) from the control unit to support this instruction?

None.

PROBLEM 4. [10 points] Consider the following Boolean expression:

$$\overline{x}\,y\,z \;+\; x\,\overline{y}\,z \;+\; x\,y\,\overline{z} + \overline{x}\,y\,\overline{z} \;+\; x\,\overline{y}\,\overline{z} \;+\; \overline{x}\,\overline{y}\,z.$$

4.1 Using the simplification rules in Slide 18 on the PDF version of the set of slides 5.3, show that the above expression is logically equivalent to

$$x\,\overline{y} + y\,\overline{z} + z\,\overline{x}.$$

$$\begin{aligned}
& \overline{x}\,y\,z \;+\; x\,\overline{y}\,z \;+\; x\,y\,\overline{z} + \overline{x}\,y\,\overline{z} \;+\; x\,\overline{y}\,\overline{z} \;+\; \overline{x}\,\overline{y}\,z \\
= \; & (\overline{x}\,y\,z + \overline{x}\,\overline{y}\,z) + (x\,\overline{y}\,z + x\,\overline{y}\,\overline{z}) + (x\,y\,\overline{z} + \overline{x}\,y\,\overline{z}) \\
= \; & \overline{x}\,z\,(y + \overline{y}) + x\,\overline{y}\,(z + \overline{z}) + y\,\overline{z}\,(x + \overline{x}) \\
= \; & \overline{x}\,z\,(1) + x\,\overline{y}\,(1) + y\,\overline{z}\,(1) \\
= \; & x\,\overline{y} + y\,\overline{z} + z\,\overline{x}
\end{aligned}$$

4.2 Give a Boolean circuit (using the logic gates `AND` and `NOT` implementing the latter expression.

See prob4.circ.

PROBLEM 5. [10 points] Below is a recursive version of the function `BitCount`. This function counts the number of bits that are set to 1 in an integer.

Translate this function into MIPS assembly code. The parameter `x` is passed to your function in register `$a0`. Your function should place the return value in register `$v0`.

```
int BitCount(unsigned x) {
   int bit;
   if (x == 0)
      return 0;
   bit = x & 0x1;
   return bit + BitCount(x >> 1);
}
```

See 'bitCount.asm'.

PROBLEM 6. [10 points] Add comments to the following MIPS code and describe what it computes using C code. Assume that `$a0` and `$a1` are used for the input and both initially contain the integers `a` and `b`, respectively. Assume that `$v0` is used for the output.

```
         add $t0, $zero, $zero    # Set $t0 = 0
 loop:   beq $a1, $zero, finish   # if $a1 == 0, go to finish
         add $t0, $t0, $a0        # $t0 += $a0
         sub $a1, $a1, 1          # $a1 -= 1
         j loop                   # go to loop
 finish: addi $t0, $t0, 100       # $t0 += 100
         add $v0, $t0, $zero      # Set return value $v0 = $t0
```

The corresponding `C` *code is as follows. It adds* `a` *by* `b` *times and adds a* `100` *in the end. Thus, it computes* `a*b+100`*.*

```
t = 0;
while (b != 0) {
  t += a;
  b -= 1;
}
t += 100;
```