Assignment 3

Computer Architecture CS3350B

Albirawi, Zaid
250626065

March 20th, 2015

1. 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.

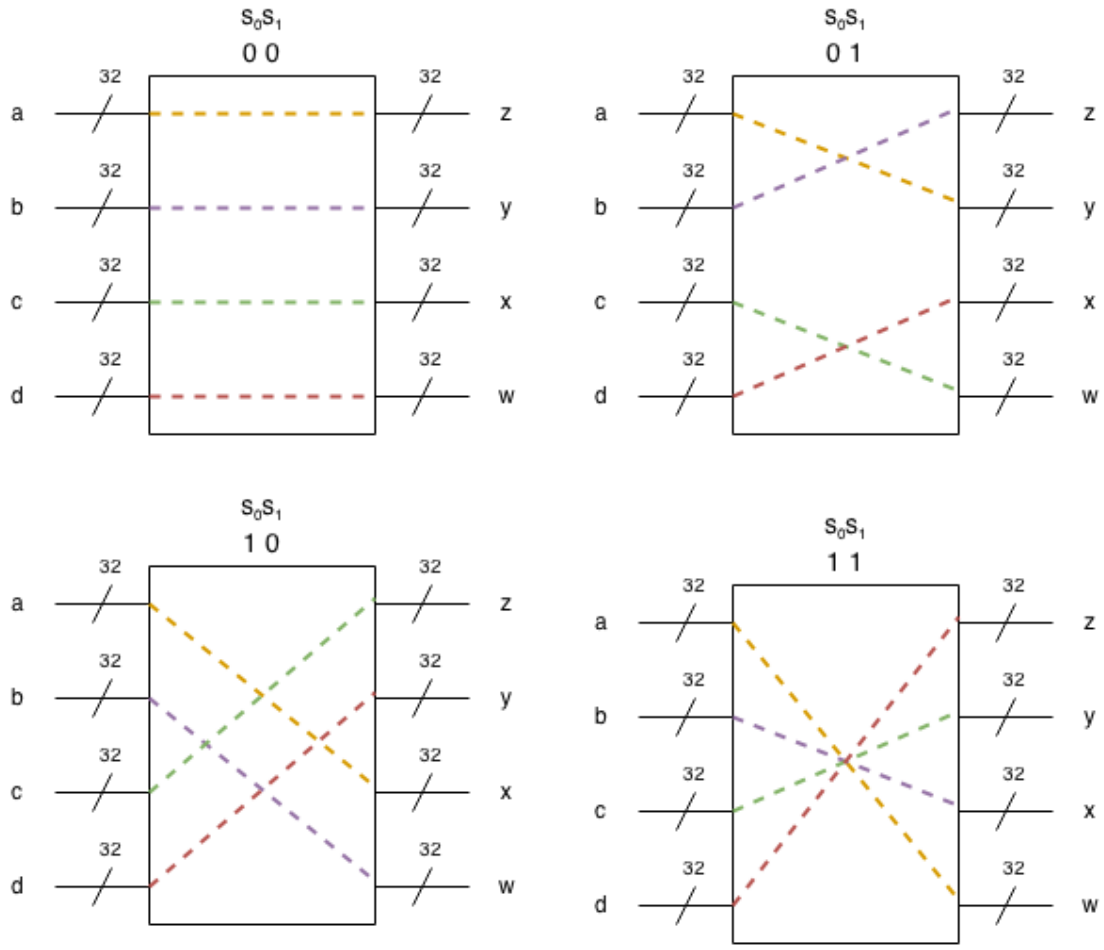
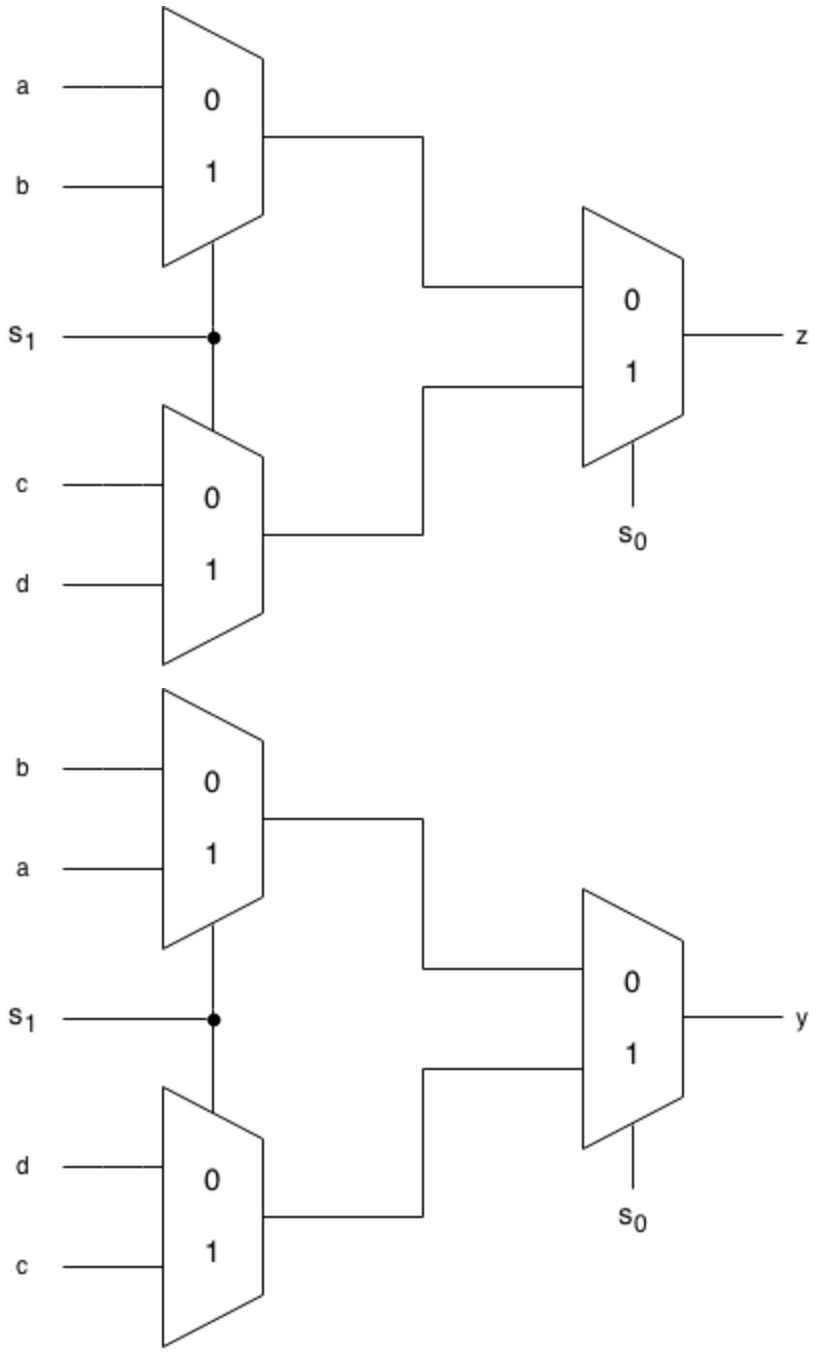


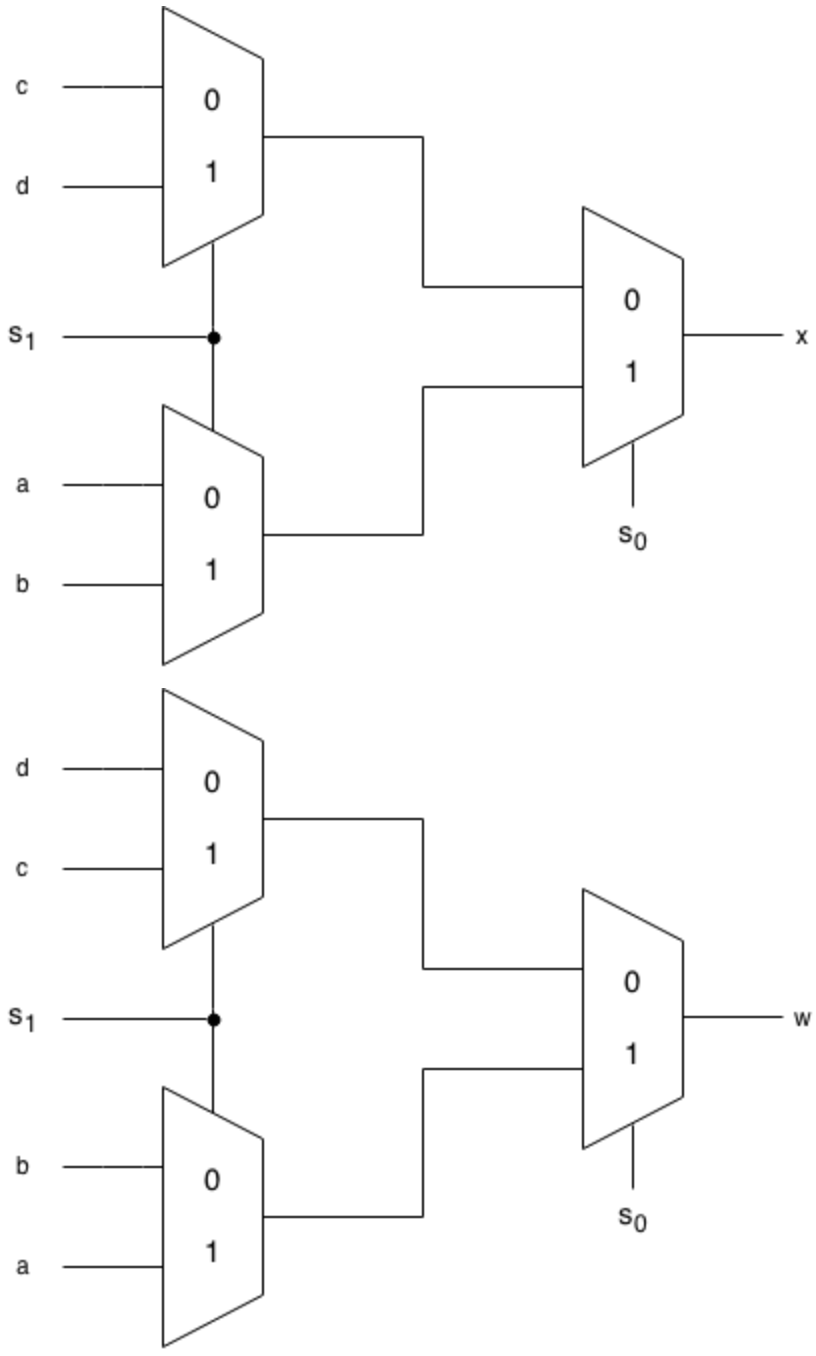
Figure 1: The switching network with the combination values of $s = s_1\ s_0$.

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.

$$z = a \cdot \neg s_0 \cdot \neg s_1 + b \cdot \neg s_0 \cdot s_1 + c \cdot s_0 \cdot \neg s_1 + d \cdot s_0 \cdot s_1$$
$$y = b \cdot \neg s_0 \cdot \neg s_1 + a \cdot \neg s_0 \cdot s_1 + d \cdot s_0 \cdot \neg s_1 + c \cdot s_0 \cdot s_1$$
$$x = c \cdot \neg s_0 \cdot \neg s_1 + d \cdot \neg s_0 \cdot s_1 + a \cdot s_0 \cdot \neg s_1 + b \cdot s_0 \cdot s_1$$
$$y = d \cdot \neg s_0 \cdot \neg s_1 + c \cdot \neg s_0 \cdot s_1 + b \cdot s_0 \cdot \neg s_1 + a \cdot s_0 \cdot s_1$$

1.2. Draw the gate diagram with hierarchical multiplexers, as on Slide 8 of the PDF version of the set of Slides 5.4.




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

2. Consider the data-path shown in Figure 2, which is taken 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.

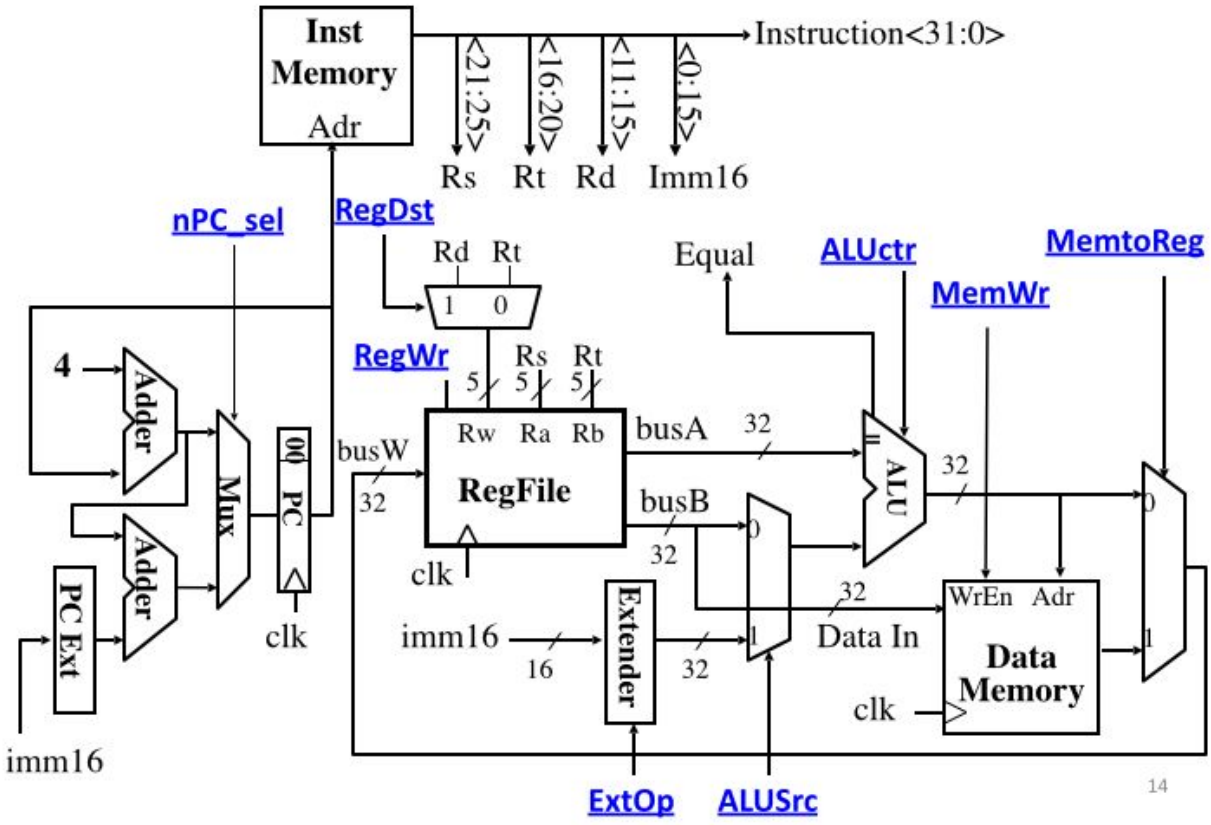


Figure 2: The basic implementation of the data-path and control for a MIPS instruction.

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

- Fetch Instruction
  - Fetches the instruction word from the memory.
  - Increments the PC. PC = PC + 4
- Instruction Decode
  - Reads the opcode, sll.
  - Reads the data from $s1.
- Arithmetic-Logic Unit
  - Takes the value in $s1 and shifts it left it by the immediate 2.
- Memory Access
  - No memory access.
- Register Write

- Writes the results from the ALU to the $s0 register.

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

- **PC**: performs the PC interementing process.
- **Instruction Memory**: fetches the instruction word from the memory.
- **RegFile**: Delivers the $s1 data to the ALU block and writes the result to the $s0 register.
- **Extender**: will extend the immediate value of 2 and pass it on to the ALU block.
- **ALU**: will perform the arithmetic shifting operation and pass the result to the RegFile block.

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?

- **PC Extender**: will produce a value that will not be used because the nPC_sel signal is not on.
- **Data Memory**: will not have any outputs for this instruction.

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.

Signal	nPC_sel	RegDst	RegWr	ExtOp	ALUSrc	ALUctr	MemWr	MemtoReg
Value	0	1	1	1	1	Add	0	0

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

   Reg[Rw] ← Mem[Reg[Ra] + Reg[Rb]];
   PC ← PC + 4;

   **Assuming that Reg[Ra] + Reg[Rb] will evaluate to a memory location without the need of an immediate number(the offset).

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

   - **PC**: performs the PC interementing process.
   - **Instruction Memory**: fetches the instruction word from the memory.
   - **RegFile**: Delivers the data stored into the Ra and Rb registers to the ALU and stores the result into the Rw register.
   - **ALU**: will perform the addition operation between the values of the Ra and Rb registers and pass the result to the address port in the Data Memory block.
   - **Data Memory**: will fetch the data from the memory and pass it onto the RegFile block.

   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?

   No new signals. The signals that we have a sufficient enough.

Signal	nPC_sel	RegDst	RegWr	ExtOp	ALUSrc	ALUctr	MemWr	MemtoReg
Value	0	1	1	0	0	Add	0	1

4. Consider the following Boolean expression:

$$\neg xyz + x\neg yz + xy\neg z + \neg xy\neg z + x\neg y\neg z + \neg x\neg yz$$

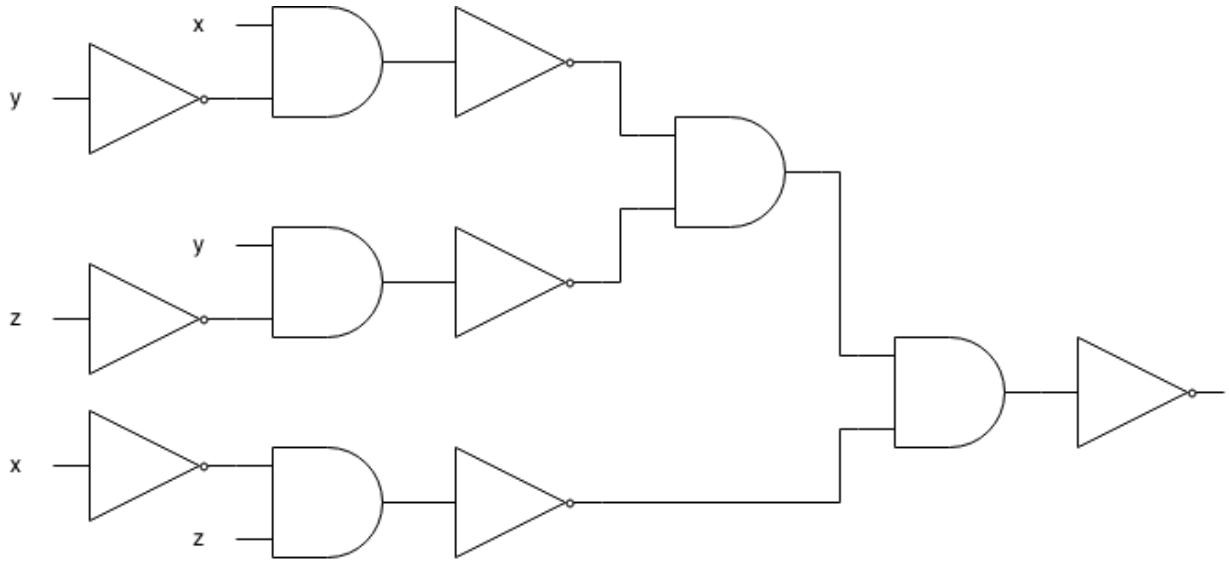
4.1. Using the simplification rules in Slide 18 on the PDF version of the set of slides 5.3, show that the above expression is logically equivalent to

$$x\neg y + y\neg z + \neg xz$$

$$= \neg xyz + x\neg yz + xy\neg z + \neg xy\neg z + x\neg y\neg z + \neg x\neg yz$$
$$= (x\neg yz + x\neg y\neg z) + (xy\neg z + \neg xy\neg z) + (\neg xyz + \neg x\neg yz)$$
$$= x\neg y \cdot (z + \neg z) + y\neg z \cdot (x + \neg x) + \neg xz \cdot (y + \neg y)$$
$$= x\neg y \cdot 1 + y\neg z \cdot 1 + \neg xz \cdot 1$$
$$= x\neg y + y\neg z + \neg xz$$

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

$$= \neg\neg(x\neg y + y\neg z + \neg xz)$$
$$= \neg[\neg(x\neg y) \cdot \neg(y\neg z) \cdot \neg(\neg xz)]$$



5. 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);
}
```

```
bitCount:
beq         $a0, 0, BASE        #if (x == 0)

andi        $s0, $a0, 0x1       #bit = x & 0x1

addi        $sp, -8             #allocates space on the stack for $ra, $s0
sw          $ra, 4($sp)         #stores the return address number on the stack
sw          $s0, 0($sp)         #stores the bit value on the stack

srl         $a0, $a0, 1         #(x >> 1)
jal         bitCount            #recursive call

BASE:
lw          $s0, 0($sp)         #loads the bit value
lw          $ra, 4($sp)         #loads the return address

add         $v0, $v0, $s0       #bit + BitCount(x >> 1);

addi        $sp, $sp, 8         #frees the stack space
jr          $ra                 #returns to the line it was called from
```

6. 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 to zero
loop:
beq     $a1, $zero, finish      #if ($a1 == 0) then finish
add     $t0, $t0, $a0           #add $a0 to $t0
sub     $a1, $a1, 1             #decrement $a1
j       loop                    #loop
finish:
addi    $t0, $t0, 100           #add 100 to the value of $t0
add     $v0, $t0, $zero         #save the return value in $v0
```

```
int problem_6(int a, int b) {
        int x = 0;
        for ( ; b != 0; b--)
                x += a;
        x += 100;
        return x;
}
```