CS 151B: Homework 1

Jonathan Woong

804205763

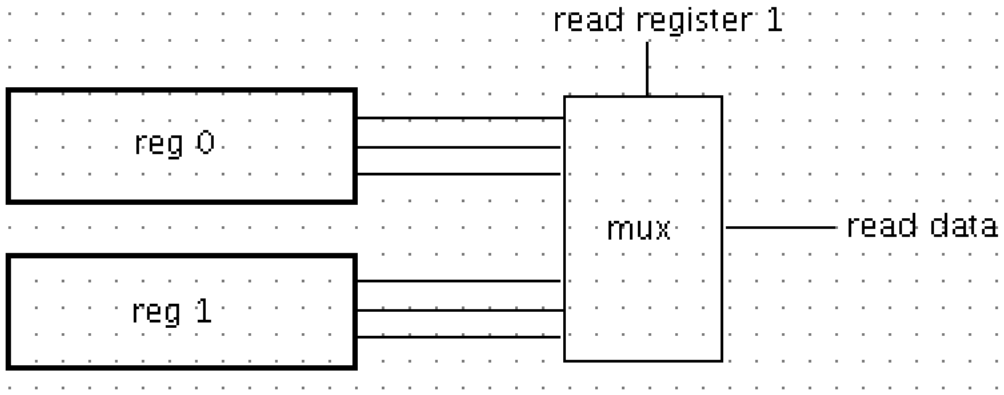
Spring 2017

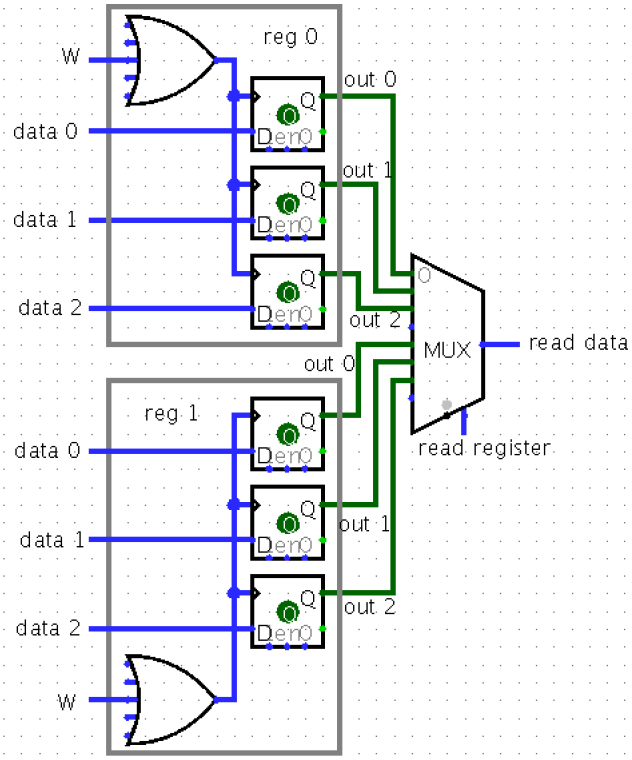
Discussion 1A

Monday 10th April, 2017

Problem 1

Figure B.8.8 on page B-55 illustrates the implementation of the read ports of the register file for the MIPS datapath. Your task is to design a new register file that has only two registers and each register has only three bits of data. This new register file has only one read port. Redraw Figure B.8.8 so that every wire in your diagram corresponds to only 1 bit of data (unlike the diagram in Figure B.8.8, in which some wires are 5 bits and some wires are 32 bits). Redraw the registers using D flip-flops. You do not need to show how to implement a D flip-flop or a multiplexer.





Problem 2

Show the minimal sequence of MIPS instructions for the following C statement:

```
b[12-i]=b[i+j]-x;
```

In the C program, `b` is declared as an array of four-byte integers and `i, j, x` are declared as four-byte integer scalars. Variables `i, j, x` are stored in, respectively, registers 5, 8, and 13. The base address of array `b` is $3,880,220_{10}$.

```
$1 is a temporary register
address of b in hex is 0x003B351C

add     $1,$5,$8        # $1 = i+j
sll     $1,$1,2         # $1 = 4(i+j)
lui     $2,0x003B       # $2 = b
ori     $2,$2,0x351C
lw      $3,0($2)        # $3 = b[0]
add     $4,$3,$1        # $4 = &b[i+j]
lw      $4,0($4)        # $4 = b[i+j]
sub     $6,$4,$13       # $6 = b[i+j]-x
li      $7,12           # $7 = 12
sub     $1,$7,$5        # $1 = 12-i
sll     $1,$1,2         # $1 = 4(12-i)
add     $9,$3,$1        # $9 = &b[12-i]
sw      $6,0($9)        # b[12-i] = b[i+j]-x
```

Problem 3

Show the minimal sequence of MIPS instructions that extract bits 11 through 16 from register $11 and use that value to replace bits 26 through 31 of register $12, without changing register $11 and without changing the other 26 bits of register $12. Note that the bit numbering is little endian.

```
srl     $2,$11,11       # $2 = $11 >> 11
sll     $2,$2,26        # $2 = $2 << 26
srl     $12,$12,6       # $12 = $12 >> 6
add     $2,$2,$12       # $2 = $2 + $12
```

Problem 4

Before the MIPS processor executes the following code segment, the value in register 5 ($5) is 29. What will be the values in register 1 and in register 2 after the entire code segment is executed? Show the values in hex. Explain your answer.

```
lui     $5, 414         # $5 = 0x19E0000 = 0000 0001 1001 1110 0000 0000 0000 0000
sw      $5, 24($0)      # M[$0 + 24] = 0x19E0000
lbu     $1, 25($0)      # $1 = M[$0 + 25] = 0000 = 0x0
lbu     $2, 26($0)      # $2 = M[$0 + 26] = 0000 = 0x0
```

Register 1 will contain 0x0 and register 2 will contain 0x0. This is because M[$0 + 24] is the start of the sequence of bytes that represent 0x019E0000 (little endian).

Problem 5

Translate the following MIPS assembly instruction into machine code.

sh $4, 36($27)

101001	00100	11011	0000000000100100

Problem 6

Consider the following C code segment:

```
while (a[i] != 33) {
   if (a[i] > y)
      z = z + a[i];
   else
      z = z - y;
   i++;
}
```

Assume that all the variables are declared as four-byte integers. `i,y,z` are stored in registers $5,$7,$12 respectively. The base of array `a` is stored in register $11. The program was compiled into the assembly to the right. Convert the assembly program to machine code.

```
        .text   452
        addi    $15,$0,33
loop:   sll     $4$,$5,2
        add     $4,$11,$4
        lw      $4,0($4)
        beq     $4,$15,next
        addi    $5,$5,1
        slt     $3,$7,$4
        beq     $3,$0,notlt
        add     $12,$12,$4
        j       loop
notlt:  sub     $12,$12,$7
        j       loop
next:
```

insn	address	op	rs	rt	rd	shamt	func
addi $15,$0,33	0001 1100 0100	001000	00000	01111	0000 0000 0010 0001		
sll $4,$5,2	0001 1100 1000	000000	00000	00101	00100	00010	000000
add $4,$11,$4	0001 1100 1100	000000	01011	00100	00100	00000	100000
lw $4,0($4)	0001 1101 0000	100011	00100	00100	0000 0000 0000 0000		
beq $4,$15,next	0001 1101 0100	000100	00100	01111	0000 0001 1111 1000		
addi $5,$5,1	0001 1101 1000	001000	00101	00101	0000 0000 0000 0001		
slt $3,$7,$4	0001 1101 1100	000000	00111	00100	00011	00000	101010
beq $3,$0,notlt	0001 1110 0000	000100	00011	00000	0000 0001 1111 0000		
add $12,$12,$4	0001 1110 0100	000000	01100	00100	01100	00000	100000
j loop	0001 1110 1000	000010	00 0000 0000 0000 0001 1100 1000				
sub $12,$12,$7	0001 1110 1100	000000	01100	00111	01100	0000	100010
j loop	0001 1111 0000	000010	00 0000 0000 0000 0001 1100 1000				

Problem 7

Consider the `seq` pseudoinstruction (page A-59). Produce efficient assembly code implementation of

```
sleu    $8, $13, $22
```

Use only real instructions. Do not use any labels.

```
slt     $1,$13,$22      # $1=1 if $13<$22, $1=0 otherwise
seq     $2,$13,$22      # $2=1 if $13==$22, $2=0 otherwise
ori     $8,$1,$2        # $8=($1 OR $2)
```

Assumption: the value in register 2 before the execution of sleu is not required by the program after the execution of sleu.

Problem 8

Assume that the `lhu` instruction does not work (you also cannot use the `lh` instruction). Assume that MIPS is a big endian processor.

Add a new pseudoinstruction to the MIPS assembly language. The new pseudoinstruction is `lhwrdu` (load a half-word from memory, unsigned). This pseudoinstruction requires the specification of a destination register number, a base register number, and an offset. The contents of the base register are added to the offset and the sum is the address of a (two byte) half word in memory which is loaded, unsigned, into the destination register. Thus, the semantics are exactly the same as for the original lhu instruction). Show an efficient assembly code implementation of

```
lhwrdu   $7,240($14)
```

using only real MIPS instructions. Minimize the use of registers.

```
addi $1,$14,0xF0        # $1 = $14 + 240 = address
lw       $1,0($1)       # $1 = word at address
srl  $1,$1,0x10         # right shift $1 by 16, store in $1
sll  $8,$1,0x10         # left shift $1 by 16, store in $8
```

Problem 9

Write a MIPS assembly program that will produce different results depending on whether the processor is big endian or little endian. Specifically, your program must store the value 0 into the byte at address 149 if the processor is little endian but store the value 1 into the byte at address 149 if the processor is big endian. If necessary, your program may modify bytes 6-13 in main memory as well as registers $7, $8, and $9. Your program may not modify any other memory locations or registers.

```
li  $7,0xF0       # 0xF0 = 1111 0000
sw  $7,100($0)    # M[$0 + 100] = 0xF0
lb  $8,100($0)    # $8 = F if little endian, $8 = 0 if big endian
lb  $9,101($0)    # $9 = 0 if little endian, $9 = F if big endian
slt $149,$8,$9    # $149 = 0 if $8>$9, $149$ = 1 if $8<$9
```