

CS/ECE 3810
Solutions – Homework #2
Fall 2009

1. Exercise 2.10:

2.10.1 (4 points): To solve, first extract the opcode bits to determine each instruction type. Then, look up the instruction on page B-50:

	Highest 6 bits	Corresponding opcode
(a)	$101011_2 = 43_{10}$	sw
(b)	$100011_2 = 35_{10}$	lw

Both these instructions are I-type instructions (see pages B-69 and B-67). Since they are memory access instructions, they will take the form: `OPCODE rt, immediate(rs)`. To find the registers and the immediate, write the bits into this instruction format:

	Opcode bits	rs bits	rt bits	Immediate bits
(a)	$101011_2 = 43_{10}$	$10000_2 = 16_{10}$	$01011_2 = 11_{10}$	$0000000000000100_2 = 4_{10}$
(b)	$100011_2 = 35_{10}$	$01000_2 = 8_{10}$	$01000_2 = 8_{10}$	$0000000001000000_2 = 64_{10}$

The opcodes have already been figured out - all that remains is to look up the registers and put the parts of the instruction together. Using the chart on page B-24, register 8 is \$t0, register 11 is \$t3, and register 16 is \$s0. The solution is:

	Instruction
(a)	sw \$t3, 4(\$s0)
(b)	lw \$t0, 64(\$t0)

2.10.2 (2 points): As found above, they are both **I-type** instructions.

2.10.3 (2 points): Convert each 32-bit word four bits at a time:

(a)	1010	1110	0000	1011	0000	0000	0000	0100	0xAE0B0004
	A	E	0	B	0	0	0	4	
(b)	1000	1101	0000	1000	0000	0000	0100	0000	0x8D080040
	8	D	0	8	0	0	4	0	

I double-checked my work by typing in the instructions in part 2.10.1 into the Mars MIPS simulator. I then assembled the program and checked the hexadecimal output to make sure it matched part 2.10.3. This helped me discover that I mixed up rs and rt in part (a). I fixed it, and the correct answers are above.

2.10.4 (4 points): For each instruction, look up the opcode and register values, determine the instruction type, and convert to binary, then assemble the bits.

(a)	add \$t0, \$t0, \$zero	R-type
	Opcode:	$00_{16} = 000000_2$
	Function:	$20_{16} = 100000_2$
	rs:	$\$t0 = 8_{10} = 01000_2$
	rt:	$\$zero = 0_{10} = 00000_2$
	rd:	$\$t0 = 8_{10} = 01000_2$

Write down the bits for each field, then regroup them and convert.

(a)	opcode	rs	rt	rd	shamt	func		
	000000	01000	00000	01000	00000	100000		
	0000	0001	0000	0000	0100	0000	0010	
	0	1	0	0	4	0	2	
								0x01004020

Repeat again for (b).

(b)	lw \$t1, 4(\$s3)	I-type
	Opcode:	$23_{16} = 100011_2$
	rs:	$\$s3 = 19_{10} = 10011_2$
	rt:	$\$t1 = 9_{10} = 01001_2$
	Immediate:	$4_{10} = 0000000000000100_2$

Write down the bits for each field, then regroup them and convert.

(b)	opcode	rs	rt	immediate				
	100011	10011	01001	0000000000000100				
	1000	1110	0110	1001	0000	0000	0000	0100
	8	E	6	9	0	0	0	4
								0x8E690004

2.10.5 (2 points): As found above, part (a) is an R-type instruction, and part (b) is an I-type instructions.

2.10.6 (4 points): The fields were determined in part 2.10.4. Simply rewrite the binary for each field as hexadecimal:

(a)	Opcode:	$000000_2 = 0_{16}$
	Function:	$100000_2 = 20_{16}$
	rs:	$01000_2 = 8_{16}$
	rt:	$00000_2 = 0_{16}$
	rd:	$01000_2 = 8_{16}$

(b)	Opcode:	$100011_2 = 23_{16}$
	rs:	$10011_2 = 13_{16}$
	rt:	$01001_2 = 9_{16}$
	Immediate:	$0000000000000100_2 = 4_{16}$

Again, I double-checked my answers using Mars. While you cannot use Mars during the exams, I highly recommend using it when you study.

2. Exercise 2.11:

2.11.1 (2 points): Simply write down the bits for each digit.

(a)	A	E	0	B	F	F	F	C
	1010	1110	0000	1011	1111	1111	1111	1100
	10101110000010111111111111111100							

(b)	8	D	0	8	F	F	C	0
	1000	1101	0000	1000	1111	1111	1100	0000
	10001101000010001111111111000000							

2.11.2 (2 points): For part (a) the 32-bit word is negative. First, since converting a positive number is generally easier, I will negate the 32-bit word in part (a) using the shortcut from page 91:

(a)	number =	10101110000010111111111111111100
	- number =	01010001111101000000000000000100

Next, sum up the positive parts and remember that the number is negative:

$$(a) \quad -(2^{30} + 2^{28} + 2^{24} + 2^{23} + 2^{22} + 2^{21} + 2^{20} + 2^{18} + 2^2) = -1374945284$$

For part (b), since it is unsigned, just sum up the decimal equivalents for each bit:

$$(b) \quad 2^{31} + 2^{27} + 2^{26} + 2^{24} + 2^{19} + 2^{15} + 2^{14} + 2^{13} + 2^{12} + 2^{11} + 2^{10} + 2^9 + 2^8 + 2^7 + 2^6 = 2366177216$$

2.11.3 (2 points): You could solve this in the same way as 2.10.1 above. Start by finding the opcode:

	Highest 6 bits	Corresponding opcode
(a)	$101011_2 = 43_{10}$	sw
(b)	$100011_2 = 35_{10}$	lw

However, at this point, notice that the highest 16 bits (highest four hexadecimal digits) of each word are *identical* to those in problem 2.10.1. The only differences are the immediate values. Compute the immediate values (signed 16-bit values), then use them as the offset.

	Immediate bits	Corresponding value
(a)	1111111111111100	-4
(b)	1111111111000000	-64

(Use the negation trick again to make the conversion easy.)

	Instruction
(a)	sw \$t3, -4(\$s0)
(b)	lw \$t0, -64(\$t0)

I again double-checked these results in Mars.

3. Exercise 2.17:

2.17.1 (2 points): The given instructions are not included because they are redundant; they can be implemented with other instructions. Including them would add unnecessary complexity to the MIPS design.

2.17.2 (2 points): For part (a), there are two register operands. Either I-type or R-type will do.

For part (b), there are three register operands. The instruction is an R-type instruction.

2.17.3 (6 points): For part (a), move the value to the destination register. Then, if the value in the register is negative, negate it. If it is positive, do nothing:

```
add $t2, $zero, $t3
bgtz $t2, Skip
sub $t2, $zero, $t2
Skip:
```

Another approach is to use the \$at register as a temporary register, and use logic and arithmetic to negate the result *only if* the original number was negative.

```
sra $at, $t3, 31 # $at will be -1 if value was negative,
                # 0 if positive.
xor $t2, $t3, $at # Invert value and
sub $t2, $t2, $at # add 1 if the value was negative.
```

For part (b), simply reverse the order of the operands and use the slt instruction.

```
slt $t1, $t3, $t2
```

Normally, reversing the order of an arithmetic inequality changes a 'less than' into a 'greater than or equal to'. However, for this instruction reversing the order of the operands *does not change* the way equal values are treated. (This is a key insight!)

4. Decimal to Hex Converter (16 points):

In this problem, I gave you source code that was missing instructions. Here are the missing instructions. Note that I have truncated the comments – see the problem for the full comments.

```
li $s1, 8          # Set up a loop counter
Loop:
rol $s0, $s0, 4    # Roll the bits left by four bits...
and $t0, $s0, 15   # Mask off low bits...
slti $t1, $t0, 10  # Determine if the bits...
beq $t1, $zero, MakeHighDigit # If not...

MakeLowDigit:
addi $t0, $t0, 48  # Combine it with...
j DigitOut        # Go output the digit

MakeHighDigit:
subi $t0, $t0, 10  # Subtract 10 from low bits
addi $t0, $t0, 65  # Add them to the code for 'A' (65) ...
```

```
DigitOut:
    move $a0, $t0      # Output the ASCII character
    li $v0, 11
    syscall

    subi $s1, $s1, 1  # Decrement loop counter
    bne $s1, $zero, Loop # Keep looping if...
```