

(6 pts) Exercise 2-8

(See number discussion in Section 2.5)

- What binary number does this hexadecimal number represent:
7fff fffa_{hex}?

- What decimal number does it represent?

(10 pts) Exercise 2-9

Show the hexadecimal representation of this MIPS instruction:

```
add $t0 , $t1 , $zero
```

(10 pts) Exercise 2-10

What MIPS instruction is represented by this binary entry:

```
1000 1101 0000 1001 0000 0000 0100 0000
```

(5 pts) Exercise 2-11

- What is the MIPS assembly code for the following:

```
if (g != j) h = g - h;
```

```
else      h = g + h;
```

Variables f to j are assigned to registers \$s0 to \$s4

```
f  $s0  
g  $s1  
h  $s2  
i  $s3  
j  $s4
```

(5 pts) Exercise 2-12

- What is the MIPS assembly code for the following:

if (j == h) g = i + j;

Variables f to j are assigned to registers \$s0 to \$s4

f \$s0
g \$s1
h \$s2
i \$s3
j \$s4

(5 pts) Exercise 2-13

- What is the MIPS assembly code for the following:

if ((j == h) && (f != i)) g = i + j;

Variables f to j are assigned to registers \$s0 to \$s4

f \$s0
g \$s1
h \$s2
i \$s3
j \$s4

(10 pts) Exercise 2-14

- What is the MIPS assembly code for the following:

```
if ( ( (g != h) && (f == i) ) ||
     (g == i) )
    g = i + j;
```

```
f $s0
g $s1
h $s2
i $s3
j $s4
```

Variables f to j are assigned to registers \$s0 to \$s4

(blank)

(20 pts) Exercise 2-18: Pseudo-instructions

- Below you will see several problems of the form:

```
li $t1, small          # $t1 = small
```
- This is an example pseudo-instruction, with its meaning given as a comment. The instruction should load 'small' into \$t1, where 'small' means a constant that fits within 16 bits. The constant is also called an 'immediate' – 'li' stands for 'load immediate'
- Your job is to write the real MIPS instruction (or sequence of instructions) that the compiler would produce for each given pseudo-instruction. For instance, the solution for the above pseudo-instruction is:

```
addi $t1, $zero, small
```
- A 'big' constant needs 32 bits to be represented. You will need some notation to talk about the upper (most significant) 16 bits and the lower (least significant bits of this number. Use UPPER(large) to refer to the most significant bits and LOWER(large) to refer to the other bits.
- To make your answers simpler, you may make use of the 'li' pseudo-instruction where helpful (except when defining li itself)

f \$s0
g \$s1
h \$s2
i \$s3
j \$s4

- ```
clear $t0 # $t0 = 0
```
- ```
beq $t1, small, L # if ($t1 == small) go to L
```
- ```
beq $t2, big, L # if ($t2 == big) go to L
```
- ```
li $t2, big       # $t2 = big
```
- ```
bge $t5, $t3, L # if ($t5 >= $t3) go to L
```
- ```
lw $t5, big($t2) # t5 = Memory[$t2+big]
```