CSE 31

Midterm 2 Sample

Time : 75 minutes

Name:

Problem	Points	Max Points
1		30
2		30
3		40
Total		100

1 : [30 pts] MIPS Translation

The program below is written using the MIPS instruction set. It is loaded into memory at address 0xF000000C (all instruction memory addresses are shown below).

```
F000000C loop: addi $1, $1, -1
F0000010       beq $1, $0, done
F0000014       j loop
F0000018 done:
```

Write out the number (in decimal) for each field (opcode, rs, rt etc) and the final bits representation in Hex. (Be sure to put down all your steps for partial credit in case you make some mistake at any steps)

addi : opcode = 8
Rs = 1
Rt = 1
Imm = -1

0000 0000 0000 0001
1111 1111 1111 1110 + 1 → -1

0010 0000 0010 0001 1111 1111 1111 1111
0x2021FFFF

beq : opcode = 4
Rs = 1
Rt = 0
Imm = 1
0001 0000 0010 0000 0000 0000 0000 0001
0x10200001

j : opcode = 2
Imm (26 bits) = 00 0000 0000 0000 0000 0000 0011
(F000000C = 1111 0000 0000 0000 0000 0000 0000 1100)

0000 1000 0000 0000 0000 0000 0000 0011
0x08000003

2 : [30 pts] Machine Code Translation

Translate the following machine code instructions into MIPS writing out ALL FIELDS followed by the instruction (Be sure to put down all your steps for partial credit in case you make some mistake at any steps)

`0x01048020` (Specify the opcode, rs, rt, rd, shmt and funct fields in binary followed by the MIPS instruction)

Binary 0000 0001 0000 0100 1000 0000 0010 0000

opcode = 0

rs = 01000 = 8 ($t0)

rt = 00100 = 4 ($a0)

rd = 10000 = 16 ($s0)

shmt = 0

funct = 100000 = 32 = 0x20 (add)

instruction = add $s0, $t0, $a0

`0x12110003` (Specify the opcode, rs, rt and imm fields in binary followed by the MIPS instruction)

Binary 0001 0010 0001 0001 0000 0000 0000 0011

opcode = 000100 = 4 (beq)

rs = 10000 = 16 ($s0)

rt = 10001 = 17 ($s1)

imm = 0000 0000 0000 0011 = 3 (three lines after next instruction)

instruction = beq $s0, $s1, 3 (label)

_____ <- next line

_____

_____

label: _____

`0x091A04D2` with PC: `0xA0012484` (Specify the opcode and addr fields in binary, the full label address in hex, followed by the MIPS instruction)

Binary 0000 1001 0001 1010 0000 0100 1101 0010

opcode = 2 (j)

label (target) address = 1010 0100 0110 1000 0001 0011 0100 1000

instruction = j 0xA4681348

3 : [40 pts] MIPS Coding

a) [10 points] The original MIPS processor did not support multiplication; compilers were expected to break down multiplication and division into simpler operations. Even on newer MIPS processors (that have the MUL instruction), compilers sometimes still do this to improve performance.

Consider the following C function:

```
int foo(int x) {
    return x*257; }
```

Write the corresponding MIPS assembly code below. You may **not** use any form of MUL. Your answer should use as **few** a number of instructions as possible

```
foo: # x * (256 + 1)
     # (x * 256) + x
     # (x << 8) + x
     # x is in $a0 passed as in argument

     sll $v0, $a0, 8     # x << 8
     add $v0, $v0, $a0

     # return value should be in $v0
     jr $ra
```

(b) [10 pts] Multiplication is more difficult when neither argument is known at compile time. The general procedure for achieving multiplication of two unsigned numbers is to use a series of shift and add operations (think about how long-hand multiplication works). The following assembly code multiplies two unsigned numbers, $a0 and $a1, leaving the result in $v0. Assume that the result is sufficiently small that it fits in a single register.

Fill in the missing lines .

```
          addi $v0, $zero, $zero    # clear $v0
loop:     beq $a1, $zero, done      # if $a1==0, we are done
          andi $t0, $a1, 1          # check bottom bit of $a1...
          beq $t0, $zero, skip      # ...if it is 0, skip over
                                    # the next instruction

          add $v0 $v0, $a0          # fill me in!

skip:     srl $a1, $a1, 1           # shift $a1 to the right

          sll $a0, $a0, 1           # fill me in!

          j loop
```

```
   1010
x  0101
   1010  ← first number
  00000  ← first << 1
 101000  ← first << 2
0000000
10100000
```

c) [20 pts] Below is a recursive version of the function BitCount. This function counts the number of bits that are set to 1 in an integer. Your task is to 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);
}
```

Translate this procedure into MIPS assembly language, following our standard conventions for register use (arguments in registers, not stack, whenever possible).

```
BitCount :    # version 1
          addi $sp, $sp, -8          # pro
          sw $ra, 0($sp)

          add $v0, $0, $0                 # return value is 0 add
          $t0, $a0, $0
          beq $t0, $0, epi                # if x == 0

          andi $t1 , $t0, 0x1             # $t0 contains bit
          sw $t1, 4($sp)                  # save bit
          srl $a0, $t0, 1                 # x >> 1 as first argument jal
          jal BitCount               # recursion (x >> 1)
          lw $t1, 4($sp)                  # restore bit
          add $v0, $v0, $t1     # $v0 contains return value of bitcount (x >> 1)

    epi:  lw $ra, 0($sp)
          addi $sp, $sp, +8
          jr $ra
```

```
BitCount :          # version 2

          addi $sp, $sp, -8          # pro
          sw $ra, 0($sp)
          sw $a0, 4($sp)
```

```
                add $v0, $0, $0                 # return value is 0
                beq $a0, $0, epi                 # if x == 0

                srl $a0, $a0, 1                  # x >> 1 as first argument
                jal BitCount                    # recursion (x >> 1)

                lw $a0, 4($sp)                  # restore bit
                andi $a0 , $a0, 0x1             # $t0 contains bit
                add $v0, $v0, $a0     # $v0 contains return value of bitcount (x >> 1)

epi:            lw $ra, 0($sp)
addi            $sp, $sp, +8
                jr $ra
```