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 of the machine instruction in Hex. (Be sure to put down all your steps for partial credit in case you make some mistake at any steps)

addi :

beq :

j :

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)

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

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

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** instructions as possible

```
foo:




     # 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

                                   # fill me in!

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

                                   # fill me in!
          j loop
```

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).