C335 Homework #3 Solutions

Part I (10 points)

In the class lecture, we have compiled the following C code:

```
while (save[i] == k)
i += 1;
```

into the following assembly language:

```
Loop:  sll    $t1, $s3, 2
       add    $t1, $t1, $s6
       lw     $t0, 0($t1)
       bne    $t0, $s5, Exit
       addi   $s3, $s3, 1
       j      Loop
Exit:  . . .
```

Assume the memory address for the first instruction (Loop: …) is 0x00400020.

Now, you are expected to assemble the program (except the last line of the assembly code) into machine code. Your solution should contain three columns: the first column being the memory address of the instruction, the second column being the binary representation of the machine code, and the third column being the hexadecimal representation of the machine code.

Memory Address	Machine Code (Bin)	Machine Code (Hex)
0x00400020	000000, 00000, 10011, 01001, 00010, 000000	0x00134880
0x00400024	000000, 01001, 10110, 01001, 00000,100000	0x01364820
0x00400028	100011, 01001, 01000, 0000 0000 0000 0000	0x8d280000
0x0040002c	000101, 01000, 10101, 0000 0000 0000 0010	0x15150002
0x00400030	001000, 10011, 10011, 0000 0000 0000 0001	0x22730001
0x00400034	000010, 0000 0100 0000 0000 0000 0010 00	0x08100008

Part II (9 points)

For the following C code segment, write a code segment in MIPS assembly language to do the same thing. Assume **i** is in $s0, **x** is in $s1, and **y** is in $s2. Don't forget to comment you code.

```
for (i=0; i<x; i=i+1)
     y = y + i;
```

Version #1

```
       add $s0, $zero, $zero      # Initialize i
Loop:  slt $t0, $s0, $s1          # compare i and x
       beq $t0, $zero, Exit       # if (i >= x) jump to Exit
       add $s2, $s2, $s0          # y = y + i
       addi $s0, $s0, 1           # i = i +1
       j  Loop                    # Loop
Exit:  . . .
```

Version #2

```
       add $s0, $zero, $zero      # Initialize i
```

```
        slt $t0, $s0, $s1          # compare i and x
        beq $t0, $zero, Exit       # if (i >= x) jump to Exit
Loop:   add $s2, $s2, $s0          # y = y + i
        addi $s0, $s0, 1           # i = i +1
        slt $t0, $s0, $s1          # compare i and x
        bne $t0, $zero, Loop       # if (i < x) Loop
Exit:   . . .
```

Part III (11 points)

We have given some examples of pseudoinstructions in the lecture notes. For each pseudoinstruction in the following table, produce a minimal sequence of actual MIPS instructions to accomplish the same thing. You may need to use **$at** for some of the sequences. In the following table, ***big*** refers to a specific number that requires 32 bits to represent and ***small*** to a number that can fit in 16 bits.

(**Hint:** you could use upper(big) / lower(big) to represent the upper/lower half of the immediate big respectively.)

Pseudoinstruction	What it accomplishes	Equivalent MIPS instructions
move $t1, $t2	$t1 = $t2	add $t1, $t2, $zero
clear $t0	$t0 = 0	add $t0, $zero, $zero
li $t1, small	$t1 = small	addi $t1, $zero, small
li $t2, big	$t2 = big	lui $t2, upper(big) ori $t2, $t2, lower(big)
addi $t0, $t2, big	$t0 = $t2 + big	lui $at, upper(big) ori $at, $at, lower(big) add $t0, $t2, $at
beq $t2, big, L	If($t2 = big) go to L	lui $at, upper(big) ori $at, $at, lower(big) beq $t2, $at, L
beq $t1, small, L	If ($t1 = small) go to L	addi $at, $zero, small beq $t1, $at, L
ble $t3, $t5, L	If ($t3 <= $t5) go to L	slt $at, $t5, $t3 beq $at, $zero, L
bgt $t4, $t5, L	If ($t4 > $t5) go to L	slt $at, $t5, $t4 bne $at, $zero, L
bge $t5, $t3, L	If ($t5 >= $t3) go to L	slt $at, $t5, $t3 beq $at, $zero, L

lw $t5, big(t2)	$t5 = Memory[$t2 + big]	lui $at, upper(big)
		ori $at, $at, lower(big)
		addu $t2, $t2, $at
		lw $t5, 0($t2)

Part IV (10 points)

In the class lecture, we have compiled the following C code:

```
int fact (int n) {
        if (n < 1) return 1;
                else return (n * fact (n-1)); }
```

into the following assembly language:

```
fact:   addi    $sp, $sp, -8            #adjust stack pointer
        sw      $ra, 4($sp)             #save return address
        sw      $a0, 0($sp)             #save argument n
        slti    $t0, $a0, 1             #test for n < 1
        beq     $t0, $zero, L1          #if n >=1, go to L1
        addi    $v0, $zero, 1           #else return 1 in $v0
        addi    $sp, $sp, 8             #adjust stack pointer
        jr      $ra                     #return to caller
L1:     addi    $a0, $a0, -1            #n >=1, so decrement n
        jal     fact                    #call fact with (n-1)
        #this is where fact returns
bk_f:   lw      $a0, 0($sp)             #restore argument n
        lw      $ra, 4($sp)             #restore return address
        addi    $sp, $sp, 8             #adjust stack pointer
        mul     $v0, $a0, $v0           #$v0 = n * fact(n-1)
        jr      $ra                     #return to caller
```

Assume the memory address for the first instruction (fact: …) is 0x00400020.

Now, you are expected to assemble the program (***SKIP the instruction: mul $v0, $a0, $v0***) into machine code. Your solution should contain three columns: the first column being the memory address of the instruction, the second column being the binary representation of the machine code, and the third column being the hexadecimal representation of the machine code.

Memory Address	Machine Code (Bin)	Machine Code (Hex)
0x00400020	001000, 11101, 11101, 1111 1111 1111 1000	0x23bdfff8
0x00400024	101011, 11101, 11111, 0000 0000 0000 0100	0xafbf0004
0x00400028	101011, 11101, 00100, 0000 0000 0000 0000	0xafa40000
0x0040002c	001010, 00100, 01000, 0000 0000 0000 0001	0x28880001
0x00400030	000100, 01000, 00000, 0000 0000 0000 0011	0x11000003
0x00400034	001000, 00000, 00010, 0000 0000 0000 0001	0x20020001
0x00400038	001000, 11101 11101, 0000 0000 0000 1000	0x23bd0008
0x0040003c	000000, 11111, 00000, 00000, 00000, 001000	0x03e00008
0x00400040	001000, 00100, 00100, 1111 1111 1111 1111	0x2084ffff
0x00400044	000011, 0000 0100 0000 0000 0000 0010 00	0x0c100008
0x00400048	100011, 11101, 00100, 0000 0000 0000 0000	0x8fa40000
0x0040004c	100011, 11101, 11111, 0000 0000 0000 0100	0x8fbf0004
0x00400050	001000, 11101, 11101, 0000 0000 0000 1000	0x23bd0008
0x00400054		
0x00400058	000000, 11111, 00000, 00000, 00000, 001000	03e000008