

## EC 362 Problem Set #1

- 1) Using the following MIPS program, determine the instruction format for each instruction and the HEX values of each instruction field. Fill in the table below with those values:

```

        addi  $v0, $zero, 0      # count = 0
loop:   lw    $v1, 0($a0)       # load word
        addi  $v0, $v0, 1      # count++
        sw    $v1, 0($a1)     # store word
        addi  $a0, $a0, 4      # ptr to next src
        addi  $a1, $a1, 4      # ptr to next dest
        bne   $v1, $zero, loop # loop if copied word is not 0
        addi  $v0, $v0, -1     # count--

```

Instr	Opcode	Rs	Rt	Addr/immed
Addi				
Lw				
Addi				
Sw				
Addi				
Addi				
Bne				
Addi				

- 2) Consider the following Java code segment:

```

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

```

An inefficient Java translation of this into MIPS might result in assembly code that uses *both* a conditional branch (for the conditional test) and an unconditional jump (for the repeat of the while) each time through the loop, in addition to the array bounds checking tests required by Java. For example, if the base address of the array `save` is in register `$$s6`, `i` is stored in register `$$s3`, and `k` is stored in register `$$s5`, then one possible inefficient MIPS code sequence would be

```

        lw    $t2, 4($s6)      # this is the length of "save"
loop:   slt   $t1, $s3, $zero   # 1 means i < 0
        bne   $t1, $zero, except # index out of bounds
        slt   $t1, $s3, $t2     # 1 means i < save.length
        beq   $t1, $zero, except # index out of bounds
        sll   $t1, $s3, 2       # 4*i
        add   $t1, $t1, $s6     # &save[i] - 8
        lw    $t0, 8($t1)      # load save[i]
        bne   $t0, $s5, exit    # save[i] != k?
        addi  $s3, $s3, 1       # i++
        j     loop
exit:

```

Note that Java Arrays store the length of the array in position `$$s6 + 4` and then the array itself starts at position `$$s6 + 8`.

Only poor compilers would produce code with such excessive loop overhead (3 branches + 1 jump). Write a version of the assembly code for this segment so that it uses at most two branches/jumps (total) each time through the loop. Please note that regardless of the optimization you perform, Java still requires that the array index be "bounds checked" for  $0 \leq i < \text{save.length}$  in every iteration! How many instructions are executed before and after the optimization if the number of loop iterations is 100.

- 3) Consider the following fragment of C code:

```
for (i = 0; i <= 100; i++) {
    a[i] = b[i] + c;
}
```

Assume that *a* and *b* are arrays of words and the base address of *a* is in  $\$a0$  and the base address of *b* is in  $\$a1$ . Register  $\$t0$  is associated with variable *i* and register  $\$s0$  with variable *c*. Write efficient MIPS code for this segment. How many instructions are executed during the running of this code?

- 4) Write a MIPS assembly language function to implement the C / C++ `atoi` (ASCII to integer) function. The function prototype for `atoi` is as follows:

```
int atoi(const char *p);
```

`atoi` converts an ASCII decimal string pointed to by the variable *p* into its corresponding integer value. For example, `atoi("24")` would pass a pointer to the three byte sequence 50 (the ASCII decimal value for '2'), 52 (the ASCII decimal value for '4'), 0 (the ASCII decimal value for '\0' which is the string termination value) which represents '2', '4', '\0'. The result would then be the integer 24 (decimal). The program need not handle negative numbers, but if any non-digit character appears in the string, your program should exit with a return value of -1. You may use the `MUL` pseudo-instruction if needed. Use `spim` to test your function!