Lecture # 38

MIPS Review (Post Take Home Exam)

MIPS registers

register	assembly name	Comment
r0	$zero	Always 0
r1	$at	Reserved for assembler
r2-r3	$v0-$v1	Stores results
r4-r7	$a0-$a3	Stores arguments
r8-r15	$t0-$t7	Temporaries, not saved
r16-r23	$s0-$s7	Contents saved for use later
r24-r25	$t8-$t9	More temporaries, not saved
r26-r27	$k0-$k1	Reserved by operating system
r28	$gp	Global pointer
r29	$sp	Stack pointer
r30	$fp	Frame pointer
r31	$ra	Return address

C Conditional Operator	MIPS Assembly Instruction
a == b	beq $t0, $t1, then
a != b	bne $t0, $t1, then
a < b	blt $t0, $t1, then
a > b	bgt $t0, $t1, then
a <= b	ble $t0, $t1, then
a >= b	bge $t0, $t1, then
a == 0	beqz $t0, then

```
int i;
for( i=0;i<10;i++ ) {
     loop body
}

int i = 0;
while( i < 10 ) {
     loop body
     i++;
}
```

```
li        $t0, 10          # t0 is a constant 10
li        $t1, 0           # t1 is our counter (i)
loop:
beq  $t1, $t0, end  # if t1 == 10 we are done
loop body
addi $t1, $t1, 1    # add 1 to t1
j         loop             # jump back to the top
end:
```

```
int strlen( char* string ) {
     int count = 0;

     while( *string != '\0' ) {
          string++;
          count++;
     }

     return count;
}
```

```
strlen:
li      $t0, 0          # initialize the count to zero
loop:
lbu     $t1, 0($a0)     # load the next character into t1
beqz    $t1, exit       # check for the null character
addi    $a0, $a0, 1     # increment the string pointer
addi    $t0, $t0, 1     # increment the count
j       loop            # return to the top of the loop
exit:
```

```
int max( int* array, int size ) {
     int maximum = array[0];

     for( int i=1;i<size;i++ )
          if( array[i] > maximum )
               maximum = array[i];

     return maximum;
}
```

```
max:
lw   $t0, 0($a0)     # load the first array value into t0
li   $t1, 1          # initialize the counter to one
loop:
beq  $t1, $a1, exit # exit if we reach the end of the array
addi $a0, $a0, 4     # increment the pointer by one word
addi $t1, $t1, 1     # increment the loop counter
lw   $t2, 0($a0)     # store the next array value into t2
ble  $t2, $t0, end_if
move $t0, $t2        # found a new maximum, store it in t0
end_if:
j    loop            # repeat the loop
exit:
```

Translate HLL into x86 into MIPS

Selection sort carries out a sequence of passes over the table. At the first pass an entry is selected on some criteria and placed in the correct position in the table. The possible criteria for selecting an element are to pick the smallest or pick the largest. If the smallest is chosen then, for sorting in ascending order, the correct position to put it is at the beginning of the table. Now that the correct entry is in the first place in the table the process is repeated on the remaining entries. Once this has been repeated n-1 times the n-1 smallest entries are in the first n-1 places which leaves the largest element in the last place. Thus only n-1 passes are required. The algorithm can be described as follows:

```
for (i = 0; i < n-1; i++){
   p = i;
   for (j = i+1; j < n; j++)
     if (a[j]<a[p])
        p = j;
   t = a[p];
   a[p] = a[i];
   a[i] = t;
 }
```

```
.data
    a:          .space 80
.text
    MAIN:

    addi $s0, $zero, 0  # i = 0
    add $t0, $zero, $a1 # $t0 = n
    addi $t0, $t0, -1   # $t0 = n - 1

FOR_1:
    slt $t1, $s0, $t0   # i < $t0 = n - 1 continue
    beq $t1, $zero, SORT_EXIT # if !(i < n - 1) branch out of loop

    add $s3, $zero, $s0     # p = i
    addi $t1, $s0, 1    # $t1 = i + 1
    add $s1, $zero, $t1     # j = $t1 = i + 1

FOR_2:
    slt $t1, $s1, $a1   # j < n continue
    beq $t1, $zero, IF_1    # if !(j < n) branch out of loop

IF_2: # "FIND MIN"
```

```
    # get value at a[ j ] store in $t3
    add $t2, $zero, $s1     # calculate index $t2 = j
    sll $t2, $t2, 2      # offset = $t2 * 4
    add $t2, $t2, $a0    # add offset to base address
    lw $t3, 0($t2)       # load value at a[ j ] into $t3

     # get value at a[p] store in$t5
    add $t4, $zero, $s3     # calculate index $t4 = p
    sll $t4, $t4, 2      # offset = $t4 * 4
    add $t4, $t4, $a0    # add offset to base address
    lw $t5, 0($t4)       # load value at a[p] into $t5

    slt $t1, $t3, $t5    # if(a[ j ] < a[p]) continue
    beq $t1, $zero, LOOP_2  # if !(a[ j ] < a[p]) branch out of if stmt
    add $s3, $zero, $s1     # p = j

LOOP_2:
    addi $s1, $s1, 1     # j++
    j FOR_2

IF_1: # "SWAP"
    beq $s3, $s0, LOOP_1    # if(p == i) branch out of if stmt (jump to LOOP_1)
```

```
    # tmp = a[p]
    add $t2, $zero, $s3     # calculate index $t2 = p
    sll $t2, $t2, 2      # offset = $t2 * 4
    add $t2, $t2, $a0    # add offset to base address
    lw $s2, 0($t2)       # $s2 = tmp = a[p]

    # a[p] = a[ i ]
    add $t3, $zero, $s0     # calculate index $t3 = i
    sll $t3, $t3, 2      # offset = $t2 * 4
    add $t3, $t3, $a0    # add offset to base address
    lw $t0, 0($t3)       # $t0 = a [ i ]

    sw $t0, 0($t2)       # store value at a[ i ] in a[p]

    # a[ i ] = tmp
    sw $s2, 0($t3)       # store tmp value in a[ i ]

LOOP_1:
    addi $s0, $s0, 1     # i++
    j FOR_1

SORT_EXIT:
```