Machine Architecture
Assignment 1

Casper B. Hansen
University of Copenhagen
Department of Computer Science
fvx507@alumni.ku.dk

September 16, 2013

Translate this C-program into MIPS assembly

```c
#include <stdio.h>

void qsort(int v[], int left, int right)
{
    int i, last, tmp;

    if (left >= right)
        return;

    tmp = v[left];
    v[left] = v[(left + right) / 2];
    v[(left + right) / 2] = tmp;
    last = left;

    for (i = left + 1; i <= right; i++)
    {
        if (v[i] < v[left]) {
            ++last;
            tmp = v[last];
            v[last] = v[i];
            v[i] = tmp;
        }
    }

    tmp = v[left];
    v[left] = v[last];
    v[last] = tmp;

    qsort(v, left, last-1);     // recurse on left-hand side
    qsort(v, last+1, right);    // recurse on right-hand side
}
```

```
.globl main

.data

array: .space 32        # reserve 32 bytes for the test array (8 ints)

.text

# main procedure
main:
    # populate the array with random numbers
    la $a0, array       # supply the array address argument
    li $a1, 8           # supply the array size argument
    li $a2, 31          # supply the upper-bound argument
    jal rand_pop        # call the random number population routine
    nop

    # for debugging, print the initial array
    la $a0, array       # supply the array address argument
    li $a1, 8           # supply the array size argument
    jal print_array     # call the array printing routine
    nop

    # call the quicksort routine
    la $a0, array       # int v[]
    li $a1, 0           # int left
    li $a2, 7           # int right
    jal qsort           # call qsort
    nop

    # for debugging, print the processed array
    la $a0, array       # supply the array address argument
    li $a1, 8           # supply the array size argument
    jal print_array     # call the array printing routine

    li $v0, 10          # load system call code for termination
    syscall             # execute the system call

# quicksort: subroutine
qsort:
    # $a0:  v[]
    # $a1:  left
    # $a2:  right

#   addi $sp, $sp, -36  # push the stack

    # l >= r is logically the same as if r < l
    blt $a2, $a1, exit  # if (right < left) simply exit

    # initialize saved temporaries
    move $s0, $zero     # int i;
    move $s1, $zero     # int last;
    move $s2, $zero     # int tmp;

    # precalculated the offsets needed
    #   $t1 = left
    #   $t2 = right
    sll $t1, $a1, 2     # calculate the left offset into the array
    sll $t2, $a2, 2     # calculate the right offset into the array

    # precalculate the addresses needed
    #   $t3 = &v[left]
    #   $t4 = &v[right]
    #   $t5 = &v[ (left + right) / 2 ]
    add $t3, $t1, $a0   # add the left offset to the address
    add $t4, $t2, $a0   # add the right offset to the address
    add $t5, $a1, $a2   # left + right
    srl $t5, $t5, 1     # divide it by 2
    sll $t5, $t5, 2     # calculate offset
#   sll $t5, $t5, 1     # look into this: divide by 2 and calculate offset in one go
    add $t5, $t5, $a0   # add the address of the array

#   lw $t0, 0($t4)
#   move $a0, $t0
#   li $v0, 1
#   syscall

    # tmp = v[left];
    lw $s2, 0($t3)      # load the first element of the sub-array

    # v[left] = v[(left + right) / 2];
    lw $t0, 0($t5)      # load the element at index (left + right) / 2
    sw $t0, 0($t3)      # and store it into v[left]

    # v[(left + right) / 2] = tmp;
    sw $s2, 0($t5)      # store tmp back into the middle of the array

    # last = left;
    move $s1, $a1

    # for-loop
    beg_for:
    beq $s0, $a2, end_for   # exit condition

    # ...

    add $s0, $s0, 1     # increment the counter
    end_for:

    # tmp = v[left];
    lw $s2, 0($t3)      # ...

    # v[left] = v[last];
    sll $t6, $s1, 2     # calculate the offset of last
    add $t6, $t6, $a0   # add to that the address of the array
    lw $t0, 0($t6)      # load the last element
    sw $t0, 0($t3)      # store it into the left-most

    # v[last] = tmp;
    sw $s2, 0($t6)      # store tmp into last

    # store the index arguments somewhere safe
    move $t8, $a1
    move $t8, $a2

    # qsort(v, left, last-1);
    add $a2, $s1, 1     # correct the right argument
#   jal qsort

    # qsort(v, last+1, right);
#   jal qsort

    exit:
    jr $ra              # return to caller

rand_pop:
    # populate the array with an unordered list of integers
    la $t0, ($a0)       # load the address of the array
    li $t1, 0           # initialize the indexing register
    move $t2, $a1       # initialize the limiting register
    mul $t2, $t2, 4     # calculate the actual limit

    # generate random numbers and use those to populate the array
    beg_rand:
    beq $t1, $t2, end_rand  # exit condition

    move $a1, $a2       # set the upper bound on the random numbers
    li $v0, 42          # load the system call code for random number generation
    syscall             # fetch the random number

    sw $a0, 0($t0)      # store the random number

    add $t1, $t1, 4     # increment the address offset counter
    add $t0, $t0, 4
    j beg_rand          # loop back
    end_rand:

    jr $ra

# print_array: prints a fancily formatted array, for debugging purposes
print_array:

    la $t0, ($a0)       # load the address of the array
    li $t1, 0           # initialize the indexing register
    move $t2, $a1           # initialize the limiting register
    mul $t2, $t2, 4     # calculate the actual limit

    li $a0, '['
    li $v0, 11
    syscall

    # print each element of the array
    beg_print:
    beq $t1, $t2, end_print # exit condition

    lw $a0, 0($t0)      # set the integer to be printed
    li $v0, 1           # load the system call code for random number generation
    syscall             # fetch the random number

    add $t1, $t1, 4     # increment the address offset counter
    add $t0, $t0, 4

    beq $t1, $t2, no_comma
    li $a0, ','
    li $v0, 11
    syscall

    no_comma:
    j beg_print         # loop back
    end_print:

    li $a0, ']'
    li $v0, 11
    syscall

    li $a0, '\n'
    li $v0, 11
    syscall

    jr $ra
```