## CS 322 Homework 2 – due 1:30 p.m., Thursday, May 3, 2012

The goal of this assignment is to re-acquaint you with assembly language programming in general and with the x86-64 architecture in particular. It is not directly connected with your **fab** compiler project, but should prove useful background for your code generator.

Write an X86-64 assembly language subroutine `qs`, suitable for calling from C, that sorts an array of double precision floating point numbers into ascending order using quicksort. The calling interface is given by

```
void qs(int n, double *a)
```

where `a` is the array to be sorted, and `n` is its size. The array should be sorted in place. Of course, you may choose to implement `qs` using additional, private subroutines if you wish.

For example, if your routine is linked with the following main program (compiled for X86-64):

```
#include <stdio.h>

double a[] = {1.1,5.5,3.3,2.2,4.4};

main () {
   int i;
   qs(5,a);
   for (i = 0; i < 5; i ++)
           printf ("%f ", a[i]);
   printf("\n");
}
```

it should produce the following output:

```
1.1 2.2 3.3 4.4 5.5
```

Of course, your code should also work when linked against more interesting main programs (like the private one we'll use as a test driver)!

In addition to the assembly code itself, you *must* turn in pseudo-code or C code corresponding to your assembly code, and you must comment *every* line of the assembly code indicating what it does in terms of the pseudo-code. For example, if the pseudo-code contained the line

```
if (i < j) *i = *j;
```

the assembly code might contain the following commented lines:

```
    cmpq %rax, %rbx       # compare i vs. j
    jbe  .L77             # branch if j <= i
    movsd (%rbx),%xmm2    # fetch *j
    movsd %xmm2,(%rax)    # store into *i
.L77:
```

It is also useful to include a comment explaining where each pseudo-code variable is held (i.e., which register or stack frame slot).

Your code must be in a separate file `qs.s`; don't use `gcc`'s special mechanisms for including in-line assembly code. Your explanatory C code or psuedo-code should be included as a block comment in this file.

Strive for correctness, clarity, concision, and efficiency, in that order. For full credit, you must address *all* these criteria! Extra credit will be given for especially fast code, but don't try for this until you are certain that the first three criteria are met.

## Quicksort

There are many published descriptions of quicksort; I suggest the one given in Cormen, et al., *Introduction to Algorithms*, MIT Press (any edition), or in Sedgewick, *Algorithms*, Addison-Wesley (any edition). *Don't* bother to compute the partition element location in some fancy way like "median of three;" just use the first or last element of the (sub)array. (When we test your code for speed, we'll use a randomly generated array, so the average case $O(n \ln n)$ behavior should apply.)

## Hints

For this homework, you will need to use the CS department linuxlab machines or an equivalent platform (64-bit X86 hardware; recent linux OS; GNU tool chain). You can choose to develop your code on MacOS, but if so, you will still need to port your `.s` file to the linuxlab machines and make sure that it assembles and runs correctly there; doing so will probably require a small amount of surgery on your assembler input file.

See the course web page for pointers to various resources on X86-64 assembly language programming.

Use the `-m64` option to `gcc` or `cc` to get X86-64 output for your driver program. (This might not be essential, but it can't hurt.)

Giving the `-S` option to `cc` or `gcc` together with a `.c` file produces a file with `.s` extension containing the assembly code generated by the compiler for that `.c` file. Again, use the `-m64` option to get X86-64 output. This can be very useful for seeing what instructions the compiler chooses, exactly what the function calling conventions are, etc. You will probably want to write `qs` in C first (a pointer-based version will generally compile to better code), and use the generated code as the basis for your own routine. This approach won't help you with the explanatory comments, though! And you will probably need to improve this code further by hand to avoid being penalized for obvious stupidities in the compiler-generated assembler. Incidentally, also specifying `-O2` to `gcc` is usually a good idea, because the resulting code will be both more efficient and clearer.

## Organization and Submission

Place your solution in a single file called `qs.s` and mail it to `cs322-01@cs.pdx.edu`. Your C code or pseudocode should be included in a block comment at the top of this file. You must mail this file as a plain text *attachment*; the contents of the message itself don't matter. You mail subject line should contain the word "`HW2`". We should then be able to assemble your program on a linuxlab machine by creating a fresh directory, saving your attachment, and typing

```
gcc -c -o qs.o qs.s
```

or

```
as -o qs.o qs.s
```

Assuming that `main.c` contains a test driver (such as the one described above) we should be able to produce a linked executable `qs` containing both the driver and your code by copying `main.c` into the directory and then typing

```
gcc -m64 main.c qs.o -o qs
```

Note that we will be using automated mechanisms to read, compile, and test your programs, so adherence to this naming and mailing policy is important! You may lose points if you fail to submit your program in the correct way.