CS 302 Homework 1 – due 3:30 p.m., Monday, April 12, 1999

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

This assignment must be done individually, not with your team partner.

Write an assembly language subroutine `qs`, suitable for calling from C, that sorts an array of double precision 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.

For example, if your routine is linked with the following main program:

```
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
```

In addition to the assembly code itself, you *must* turn in pseudo-code or C code for your subroutine, 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 a line

```
x = t
```

the assembly code might contain the following commented lines:

```
ld [%fp-36],%i4       ! fetch t
st %i4,[%fp-40]       ! store into x
```

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 subroutine must be in a separate file `qs.s`; don't use `gcc`'s special mechanisms for including in-line assembly code.

Strive for correctness, clarity, concision, and efficiency, in that order. For full credit, you must address *all* these criteria!

Quicksort

There are many published descriptions of quicksort; I suggest the one given on pp. 153-155 of Cormen, Leiserson, and Rivest, *Introduction to Algorithms*, MIT Press, 1990, or the one in (any edition of) Sedgewick, *Algorithms*, Addison-Wesley. *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.

Hints

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. This can be very useful for seeing what instructions the compiler chooses, exactly what the function calling conventions are, etc. You may want to write `qs` in C first (a pointer-based version will 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.

Integer register conventions on SPARC are described in SPARC documentation handed out in class. Note that you *must* begin your routine by creating a stack frame of at least 96 bytes, using an instruction of the form `save %sp,-96,%sp`, and you must conclude the routine with a `restore` instruction; otherwise, don't mess with `%sp`. After the `save` instruction has been executed, you'll find the input arguments in `%i0` and `%i1`, the frame pointer in `%fp` (which is the same as `%i6`) and the call address in `%i7` (the return address is 8 more than the call address). You can use the other integer registers freely, bearing in mind that the `%o` registers aren't saved if you call another procedure.

You can assume that the caller saves any floating point registers, so you can use them freely.

Early versions of the SPARC did not implement the integer multiply instructions (`umul` and `smul`), so most compilers don't use them, but you can on the CS dept. machines. Remember, there are especially easy ways to multiply by powers of 2.

Don't forget that memory locations referenced in double-word moving instructions must be aligned on 8-byte boundaries.

Organization and Submission

Submit your fully commented qs.s file as a bundle to `cs302acc`.