Project #7: SPARC Assembly Programming

Due Date: Tuesday, January 24, 2006, Noon

Overview

This assignment is an exercise in assembly programming, which will provide useful background for the last component of your compiler project, the code generator.

Your goal is to create a subroutine **scan(a,n)** which takes as input a one dimensional array **a** of size **n**, and returns a new array of size **n + 1**, such that the **i**-th element of the new array (for **i** ≠ **n**) equals the sum of the first **i** elements of **a**, and the last element represents the maximal element of **a**. For instance, if the input array consists of four elements [4; 6; 8; 5], then the output array should have five elements [4; 10; 18; 23; 8].

Write two versions of this subroutine in the SPARC assembly language, one for an integer array and one for a double array. These two routines should be made suitable for calling from C with the following interface:

```
int * iscan (int *a, int n)
double * dscan (double *a, int n)
```

where **a** is a pointer to the input array and **n** is the size of the array. Each subroutine should allocate a fresh array (of size **n** + 1) on the heap to hold the result, and return a pointer to this array after filling it in. Each subroutine should also check two error cases, "**n** ≤ 0" and "No room available in the heap"; in both cases, the routine should return the null pointer.

For example, if your routines are linked with the following **main** program:

```
int a[] = {3, 4, 5, 6, 7, -8};
double b[] = {3.0, 4.0, 5.0, 6.0, 7.0, -8.0};

main() {
    int i, *c = iscan (a, 6);
    double *d = dscan (b, 6);

    printf ("c = [");
    for (i=0; i<7; i++)
        printf (" %4d", c[i]);
    printf ("]\n");
    printf ("d = [");
    for (i=0; i<7; i++)
        printf (" %4.1f", d[i]);
    printf ("]\n");
}
```

it should produce the following output:

```
c = [    3    7   12   18   25   17    7]
d = [  3.0  7.0 12.0 18.0 25.0 17.0  7.0]
```

Implementation Requirements

- Name your two routines **iscan** and **dscan** and place them in a single file, **scan.s**.
- Write "C" code for these two routines and include them in **scan.s** as block comments.
- For each routine, include a block comment listing the purpose of all registers that are used in the routine.
- Provide a comment for every line of assembly code indicating what it does. This comment must be expressed in terms of the "C" code. For example, for the two lines in "C" code:

```
x = 9;
y = (x - 1) * (x - 7) / (x - 33);
```

the assembly code might contain the following commented lines:

```
...                     ! x is stored in %l0
...                     ! y is stored in %l1
...                     ...
mov     9, %l0          ! Initialize x to 9
sub     %l0, 1, %o0     ! Move (x - 1) into %o0
sub     %l0, 7, %o1     ! Move (x - 7) into %o1
smul    %o0,%o1,%o0     ! Multiply, putting result in %o0
sub     %l0, 33, %o1    ! Move divisor (x - 33) into %o1
wr      %g0,%y          ! Divide, putting result in %o0
sdiv    %o0,%o1,%o0     ! .
mov     %o0, %l1        ! Store final result in y
```

For full credit, you must address the following criteria: correctness, clarity, concision, and efficiency. Correctness is the main criterion. There are several extremal cases that are not discussed here, but which must be handled correctly. Do not worry about overflow from the addition that must be done.

In time permits, you should strive to optimize your code by reducing the number of SPARC instructions as much as possible. Only try to fill delay slots after first making sure your program works perfectly.

Restrictions

Do not use the "-S" switch of "gcc" to generate your answer. Also, do not use the "-S" switch of "gcc" to generate a program which you then modify and optimize. (That would be considered cheating!) Instead, you must write the SPARC code from scratch.

Files

The following files can be found via the class web page or FTPed from:

```
~harry/public_html/compilers/p7
```

main.c
This is a driver routine that will call your code. This is NOT the driver routine that will be used to test your program. We will use a more thorough routine that will exercise your **iscan** and **dscan** on several combinations of data.

makefile
This is a short Unix "makefile". You may use it if you wish. See the Unix "`make`" command for more details.

Program Submission

Please email your **scan.s** file as a plain-text attachment to:

```
???@cs.pdx.edu
```

Please use a subject like:

```
Proj 7 - John Doe
```

Remember:

- Include only a single file (**scan.s**).
- Include the file as an attachment.
- Do not include *anything else* in the email.
- The e-mail "subject" should say
  Proj 7 – *Your Name*
- Do not submit more than once, without prior approval.
- If instructed to re-submit, the subject should be
  Proj 7 – *Your Name* – Second submission
- Send all other email queries using a different subject (e.g., "**CS322 Question**")
- Be sure to keep an unmodified copy of your file on the PSU Solaris with the timestamp intact, in case there are "issues" later about what you emailed.
- Work independently: you must write this program by yourself.

Hints / How to Get Started

If you are having difficulty, try the following steps:

1. Type in and run the "**myHello.s**" program (from the "SPARC Demo Programs" document).

2. Write a small routine (in SPARC assembler) that is passed 2 integer arguments. This routine will add them together and return the result (call it "**adder.s**"). Write a simple "**main.c**" to test it. Assemble and link these together and run it.

Here is a "C" version of **adder**; translate it into SPARC:

```
int adder (int x, int y) {
  return x + y;
}
```

This step teaches you how to link a subroutine with C routines, and how to pass arguments and return results.

3. Write a SPARC routine that is passed a pointer to an array of integers and a number (like in the project assignment). However, it will simply return an integer. This routine will run through the array passed to it, find the maximal element, and return it. Begin by writing a "C" version of this routine, to make sure you understand the general algorithm for searching an array. This step will make sure you can deal with pointers.

4. Write a version of "**iscan**" in "C" and make sure it does exactly what the project assignment requires. In this code, use only the simplest of "C" code:

- Use only "while(1) {}" for looping. Use "if ... break;" to exit.
- Do not use "for" loops.
- Use very simple tests in "if" statements, like "if (x>=n)..."
- Break complex statements (like "if (x>=n && i>0)...") into multiple statements.

This will make sure you understand the algorithm you will be using when you write your SPARC version. Using simple statements will make translating the algorithm into SPARC easier.