Debug C code on Sparc Processor from Command Line

OBJECTIVE

The objective of this tutorial ia to show you how to take a C program compile it and show how to do single step debug on a Sparc processor in the command line.

HOW TO COMPILE C CODE IN UNIX

C CODE

```
main()
{
        register int i = 0;
        register int sum = 0;

        static int A[10] = {1,2,3,4,5,6,7,8,9,-1};

        Loop:  sum = sum + A[i];
               i = i+1;

               if(i != 10)
                       goto Loop;
}
```

For this tutorial to compile C code gcc will be used. So if you take the C code provided above and name it loop.c, then to compile it all you have to do is enter the following:

```
bash-2.03$ gcc -g loop.c -o loop
```

Gcc is the name of the compiler, loop.c is the file, and –o loop specifies the output file so that it is not the default output file which is a.out. If there are no errors than you will see a new line else you will get an error message and you will have to find your error(s).

For the assembly code you have to enter the following command:

```
bash-2.03$ gcc loop.c -S
```

RUN YOUR C CODE IN UNIX

Now if you want to run your program in Unix, if you have successfully compiled your code then you will have to enter the following text.

```
bash-2.03$ ./loop
```

Keep in mind if you are using the code from this tutorial you will not see an output as there is no output in the program.

DEBUG YOUR CODE

To debug your code you will first have to invoke gdb, which is a command line debugger. In order for the debugger to work you will have to have successfully compiled your code. Then you will enter the following text:

```
bash-2.03$ gdb loop
```

And the result should look as follows:

```
bash-2.03$ gdb loop
GNU gdb 5.0
Copyright 2000 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "sparc-sun-solaris2.7"...
(gdb) break main
```

Now that we are in the debugging environment lets set a break point at the beginning of the code. This will be done by entering the following text:

```
(gdb) break main
Breakpoint 1 at 0x1059c: file loop.c, line 3.
(gdb)
```

At this time we can now begin to run our program since we have inserted a break point to do this lets enter the following:

```
(gdb) run
Starting program: /home/cslab/faculty/cs342/loopprogram/loop

Breakpoint 1, main () at loop.c:3
3               register int i = 0;
```

Now your program should have started and is ready to execute line 3 which is to initialize an Int I as 0 in a register. At this point I will demonstrate how to make a single step. To do this enter the following:

```
(gdb) step
4               register int sum = 0;
```

If you now want to see the contents of I then please enter the following:

```
(gdb) print i
$1 = 0
```

Now if you want to see where you are currently at in the debug mode then you enter the following command:

```
(gdb) where
#0  main () at loop.c:4
```

If you want to see what is in the registers then enter:

```
(gdb) info registers
g0             0x0      0
g1             0xff3197cc       -13527092
g2             0x0      0
g3             0x0      0
g4             0x0      0
g5             0x0      0
g6             0x0      0
g7             0x0      0
o0             0x1      1
o1             0x1      1
o2             0x1      1
o3             0x0      0
o4             0x0      0
o5             0xff29bbd0       -14042160
sp             0xffbef5a0       -4262496
o7             0x1064c  67148
l0             0xff33e5d8       -13376040
l1             0x0      0
l2             0x0      0
l3             0x0      0
l4             0x0      0
l5             0x0      0
l6             0x0      0
---Type <return> to continue, or q <return> to quit---
```

To view the assembly code enter disassemble and you should get the following:

```
(gdb) disassemble
Dump of assembler code for function main:
0x10598 <main>: save  %sp, -112, %sp
0x1059c <main+4>:       clr  %o0
0x105a0 <main+8>:       clr  %o1
0x105a4 <main+12>:      sethi  %hi(0x20400), %o3
0x105a8 <main+16>:      or  %o3, 0x3c4, %o2     ! 0x207c4 <force_to_data>
0x105ac <main+20>:      mov  %o0, %o3
0x105b0 <main+24>:      sll  %o3, 2, %o4
0x105b4 <main+28>:      ld  [ %o2 + %o4 ], %o2
0x105b8 <main+32>:      add  %o1, %o2, %o1
0x105bc <main+36>:      inc  %o0
0x105c0 <main+40>:      cmp  %o0, 0xa
0x105c4 <main+44>:      be  0x105d4 <main+60>
0x105c8 <main+48>:      nop
0x105cc <main+52>:      b  0x105a4 <main+12>
0x105d0 <main+56>:      nop
0x105d4 <main+60>:      ret
0x105d8 <main+64>:      restore
End of assembler dump.
```

To find out what is at a given memory location enter as follows:

```
(gdb) x 0x207c0
0x207c0 <completed.4>:  0x00000000
```

But if you want to find more than just one memory location you can enter:

```
(gdb) x/10w
0x207c4 <force_to_data>:        0x00000001      0x00000002      0x00000003      0x00000004
0x207d4 <force_to_data+16>:     0x00000005      0x00000006      0x00000007      0x00000008
0x207e4 <force_to_data+32>:     0x00000009      0xffffffff
(gdb)
```

This will give you the next ten words in memory. At this point you have all the instructions needed to do a single step debug of C code in assembly.

Assignment:

Try another example. Write a program that calculates the Factorial of a number. Run it in Sparc and then on SPIM (MIPS) software.