27 Foreign Language Interfaces, Cont.

Outline

Announcements

Example: Factorial — a recursive function

27.1 Announcements

- Review Session

27.2 Example: Factorial — a recursive function

Now we'll look at the stack development of a recursive function, `factorial()`.

Factorial is implemented as follows:

$$\text{fact}(x) = \begin{cases} 1 & \text{if } x = 0 \\ x \cdot \text{fact}(x-1) & \textit{otherwise} \end{cases}$$

Since factorial is recursive, we have will be making nested function calls and will have to take that into consideration.

(Yes, you could optimize this, but the idea is to demonstrate a self-contained recursive function.) The code for the assembly and C parts is given in figure 57.

A representative stack frame for a call to `fact()` is given here:

$fp_1, sp_1 \longrightarrow$	saved ra_1	0
	saved fp_0	4
	a0 shadow	8
	a1 shadow	12
	a2 shadow	16
$sp_0 \longrightarrow$	a3 shadow	20

An unwound version for `fact(6)` is shown in figure 59.

27.3 Help with printf()

Once again we'll run over printf for those having problems with the homework.

```
        .globl fact
        .text
        .align 2
        a0shadow = 8
        ras = 0
        fps = 4
fact:   sub  $sp,$sp,8          # preserve callee-save registers (fp,ra)
        sw   $31,ras($sp)       # save return address
        sw   $fp,fps($sp)       # save frame pointer
        move $fp,$sp            # set the frame pointer
        blez $4,base            # if x <= 0, return 1
        sw   $4,a0shadow($fp)   # preserve the argument (a0 to stack)
        sub  $4,$4,1            # set up n-1 as an arg
        sub  $sp,$sp,16
        jal  fact               # recurse
        add  $sp,$sp,16
        lw   $4,a0shadow($fp)   # restore own argument
        mult $4,$2              # mult n(a0) and fact(n-1)(v0)
        mflo $2                 # return this value in v0
        mfhi $8
        beq  $8,$0,end          # finish on no overflow
        li   $2,0               # return 0 if overflow
        j    end
base:   li   $2,1
end:    move $sp,$fp            # restore stack from frame pointer
        lw   $31,ras($sp)       # restore return address
        lw   $fp,fps($sp)       # restore frame pointer
        add  $sp,$sp,8
        j    $31                # return
```

```
#include <stdio.h>

extern int fact(int x);

int main(int argc, char *argv[]){
  int i;
  for(i=1;i<argc;i++){
    printf("Fact(%d) = %d\n",atoi(argv[i]),fact(atoi(argv[i])));
  }
}
```

Figure 57: fact.s and main.c

```
elysium% fact 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Fact(0) = 1
Fact(1) = 1
Fact(2) = 2
Fact(3) = 6
Fact(4) = 24
Fact(5) = 120
Fact(6) = 720
Fact(7) = 5040
Fact(8) = 40320
Fact(9) = 362880
Fact(10) = 3628800
Fact(11) = 39916800
Fact(12) = 479001600
Fact(13) = 0
Fact(14) = 0
Fact(15) = 0
```

Figure 58: Output from `fact` program

$fp_7, sp_7 \longrightarrow$	saved ra_7	0x400644	0	fact(0)
	saved fp_6	fp_6	4	
	a0 shadow	0	8	
	a1 shadow	$<unk>$	12	
	a2 shadow	$<unk>$	16	
	a3 shadow	$<unk>$	20	
$fp_6 \longrightarrow$	saved ra_6	0x400644	0	fact(1)
	saved fp_5	fp_5	4	
	a0 shadow	1	8	
	a1 shadow	$<unk>$	12	
	a2 shadow	$<unk>$	16	
	a3 shadow	$<unk>$	20	
$fp_5 \longrightarrow$	saved ra_5	0x400644	0	fact(2)
	saved fp_4	fp_4	4	
	a0 shadow	2	8	
	a1 shadow	$<unk>$	12	
	a2 shadow	$<unk>$	16	
	a3 shadow	$<unk>$	20	
$fp_4 \longrightarrow$	saved ra_4	0x400644	0	fact(3)
	saved fp_3	fp_3	4	
	a0 shadow	3	8	
	a1 shadow	$<unk>$	12	
	a2 shadow	$<unk>$	16	
	a3 shadow	$<unk>$	20	
$fp_3 \longrightarrow$	saved ra_3	0x400644	0	fact(4)
	saved fp_2	fp_2	4	
	a0 shadow	4	8	
	a1 shadow	$<unk>$	12	
	a2 shadow	$<unk>$	16	
	a3 shadow	$<unk>$	20	
$fp_2 \longrightarrow$	saved ra_2	0x400644	0	fact(5)
	saved fp_1	fp_1	4	
	a0 shadow	5	8	
	a1 shadow	$<unk>$	12	
	a2 shadow	$<unk>$	16	
	a3 shadow	$<unk>$	20	
$fp_1 \longrightarrow$	saved ra_1	0x4006cc	0	fact(6)
	saved fp_0	$<unk>$	4	
	a0 shadow	6	8	
	a1 shadow	$<unk>$	12	
	a2 shadow	$<unk>$	16	
	a3 shadow	$<unk>$	20	
$sp_0 \longrightarrow$	Original TOS			

Return address in fact: 0x400644
Return address in main: 0x4006cc

Figure 59: Fact(6) unwound.