Procedure Calls, SPIM, Other ISAs

CPS 104
Lecture 7

Administrivia

- Homework #2, Due Tuesday September 24
- Software Project: Simulate MIPS

Outline
- Review
- SPIM
- powerPC, Intel 80x86 ISA

Reading
- Appendix A

Next Lecture
- Logic Design
- Reading Appendix B.1-B.3
# Review: The C code

```c
#include <iostream.h>

int main ( )
{
    int i;
    int sum = 0;
    for(i=0; i <= 100; i++)
        sum = sum + i*i ;
    cout << "The answer is " << sum << endl;
}

Let's write the assembly ... :)
```

---

## Review: Assembly Language Example 1

```
.text
.align 2
main:
    subu $29, $29, 32
    sw $31, 20($29)
    sd $4, 32($29)
    sw $0, 24($29)
    sw $0, 28($29)
loop:
    lw $14, 28($29)
    mul $15, $14, $14
    lw $24, 24($29)
    addu $25, $24, $15
    sw $25, 24($29)
    addu $8, $14, 1
    sw $8, 28($29)
    ble $8, 100, loop
  la $4, str
  lw $5, 24($29)
  jal cout
  move $2, $0
  lw $31, 20($29)
  addu $29, $29, 32
  jr $31

.data
.align 0
str:
  .asciiz "The answer is "
```

---
Call-Return Linkage: Stack Frames

- FP: Points to the top of the stack frame.
- SP: Points to the start of the stack frame.
- ARGS: Register for arguments.
- Callee Save Registers: Fixed offset from FP.
- Local Variables: Grows and shrinks during expression evaluation.
- Dynamic area:_argument 5, Argument 6, Callee Save Registers, (old FP, RA).

MIPS Register Naming Conventions

- 0: zero constant 0
- 1: at reserved for assembler
- 2: v0 expression evaluation & function results
- 3: v1 function results
- 4: a0 arguments
- 5: a1
- 6: a2
- 7: a3
- 8: t0 temporary: caller saves
- 15: t7
- 16: s0 callee saves
- 23: s7
- 24: t8 temporary (cont’d)
- 25: t9
- 26: k0 reserved for OS kernel
- 27: k1
- 28: gp Pointer to global area
- 29: sp Stack pointer
- 30: fp frame pointer
- 31: ra Return Address (HW)
Example: Factorial

Stack

<table>
<thead>
<tr>
<th>Old $ra</th>
<th>Main</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>fact(10)</td>
</tr>
</tbody>
</table>

Stack grows

<table>
<thead>
<tr>
<th>Old $ra</th>
<th>Old $fp</th>
</tr>
</thead>
<tbody>
<tr>
<td>fact(9)</td>
<td></td>
</tr>
<tr>
<td>fact(8)</td>
<td></td>
</tr>
<tr>
<td>fact(7)</td>
<td></td>
</tr>
<tr>
<td>fact(6)</td>
<td></td>
</tr>
</tbody>
</table>

System Call Instruction

- System call is used to communicate with the operating system and do simple I/O.
- Load system call code into Register $v0
- Load arguments (if any) into registers $a0, $a1 or $f12 (for floating point).
- do: syscall
- Results returned in registers $v0 or $f0.
### SPIM System Call Support

<table>
<thead>
<tr>
<th>Code</th>
<th>Service</th>
<th>Arguments</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>print</td>
<td>int</td>
<td>$a0</td>
</tr>
<tr>
<td>2</td>
<td>print</td>
<td>float</td>
<td>$f12</td>
</tr>
<tr>
<td>3</td>
<td>print</td>
<td>double</td>
<td>$f12</td>
</tr>
<tr>
<td>4</td>
<td>print</td>
<td>string</td>
<td>$a0 (string address)</td>
</tr>
<tr>
<td>5</td>
<td>read</td>
<td>integer</td>
<td>integer in $v0</td>
</tr>
<tr>
<td>6</td>
<td>read</td>
<td>float</td>
<td>float in $f0</td>
</tr>
<tr>
<td>7</td>
<td>read</td>
<td>double</td>
<td>double in $f0</td>
</tr>
<tr>
<td>8</td>
<td>read</td>
<td>string</td>
<td>$a0=buffer, $a1=length</td>
</tr>
<tr>
<td>9</td>
<td>sbrk</td>
<td></td>
<td>$a0=amount address in $v0</td>
</tr>
<tr>
<td>10</td>
<td>exit</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Echo number and string

```
.text
main:
    li $v0, 5       # code to read an integer
    syscall         # do the read (invokes the OS)
    move $a0, $v0   # copy result from v0 to a0

    li $v0, 1       # code to print an integer
    syscall         # print the integer

    li $v0, 4       # code to print string
    la $a0, nln     # address of string (newline)
    syscall
```
Echo Continued

li $v0, 8  # code to read a string
la $a0, name  # address of buffer (name)
li $a1, 8  # size of buffer (8 bytes)
syscall
la $a0, name  # address of string to print
li $v0, 4  # code to print a string
syscall

jr $31  # return

.data
.align 2
name: .word 0,0
nln: .asciiz "\n"

Example 2

# Example for CPS 104
# Program to add together array of 9 numbers.
.text  # Code
.align 2
.globl main  # MAIN procedure Entrance
main:
subu $sp, 40  #\ Push the stack
sw $ra, 36($sp)  # \ Save return address
sw $s3, 32($sp)  # \
sw $s2, 28($sp)  # > Entry Housekeeping
sw $s1, 24($sp)  # / save registers on stack
sw $s0, 20($sp)  # /
mv $v0, $0  #/ initialize exit code to 0
mv $s1, $0  #\
la $s0, list  # \ Initialization
la $s2, msg  # /
la $s3, list+36  #/
# Example 2 (Continued)

Main code segment

again:
```
lw $t6, 0($s0)     #
addu $s1, $s1, $t6   #/ Actual "work"
               #   SPIM I/O
li $v0, 4         #
move $a0, $s2      # > Print a string
syscall            #/
li $v0, 1          #
move $a0, $s1      # > Print a number
syscall            #/
li $v0, 4          #
l a $a0, n ln      # > Print a string (eol)
syscall            #/
```

```
addu $s0, $s0, 4       #\ index update and
bne $s0, $s3, again   #/ end of loop
```

Exit Code

```
move $v0, $0          #
lw $s0, 20($sp)       #
lw $s1, 24($sp)       #
lw $s2, 28($sp)       # \ Closing Housekeeping
lw $s3, 32($sp)       # / restore registers
lw $ra, 36($sp)       # / load return address
addu $sp, 40          # / Pop the stack
jr $ra                # exit(0) ;
.end main             # end of program
```

Data Segment

```
.data          # Start of data segment
list: .word 35, 16, 42, 19, 55, 91, 24, 61, 53
msg: .asciiz "The sum is 
ln: .asciiz \n"
```
Example 2

• Let’s write it recursively

Miscellaneous MIPS Instructions

break   A breakpoint trap occurs, transfers control to exception handler
syscall  A system trap occurs, transfers control to operating system
coprocessor instrs  Support for floating point.
TLB instructions  Support for virtual memory: discussed later
restore from exception  Restores previous interrupt mask & kernel/user mode bits into status register
load word left/right  Supports unaligned word loads
store word left/right  Supports unaligned word stores
SPIM Demo

- spim: command line interface
- xspim: xwindows interface

PowerPC ISA

- Very similar to MIPS
- Indexed Addressing (register+register)
  \[ \text{lw } t1, a0, s3 \quad t1 = \text{mem}[a0+s3] \]
- Update Addressing
  \[ \text{lw } t1, 4(a0) \quad t1 = \text{mem}[a0+4]; \quad a0 += 4; \]
- Load/Store Multiple
- Counter Register
  \[ \text{bc loop, ctr } != 0 \]
  \[ \text{decrement ctr, if ctr } != 0 \text{ go to loop} \]
Intel 80x86 ISA

- Long history
- Binary compatibility
- 1978: 8086, 16-bit, registers have dedicated uses
- 1980: 8087, added floating point (stack)
- 1982: 80286, 24-bit
- 1985: 80386, 32-bit, new insts -> GPR almost
- 1989-95: 80486, Pentium, Pentium II
- 1997: Added MMX
- 1999: Pentium III
- 2002: Pentium 4

80x86 Registers and Addressing Modes

- eight 32-bit GPRs
  - EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI
- six 16-bit Registers for code, stack, & data
- 2 address ISA
  - one operand is both source and destination
- Not Load/Store
  - One operand can be in memory
80x86 Addressing Modes

- **Register Indirect**
  - mem[reg]
  - not ESP or EBP

- **Base + displacement (8 or 32 bit)**
  - mem[reg + const]
  - not ESP or EBP

- **Base + scaled Index**
  - mem[reg + (2<sup>scale</sup> x index)]
  - scale = 0,1,2,3
  - base any GPR, Index not ESP

- **Base + scaled Index + displacement**
  - mem[reg + (2<sup>scale</sup> x index) + displacement]
  - scale = 0,1,2,3
  - base any GPR, Index not ESP

Condition Codes

- Both PowerPC and x86 ISA have condition codes
- Special HW register, that has values set as side effect of instruction execution

- **Example conditions**
  - Zero
  - Negative

- **Example use**
  subi $t0, $t0, 1
  bz loop
80x86 Instruction Encoding

• Variable Size 1-byte to 17-bytes
• Jump (JE) 2-bytes
• Push 1-byte
• Add Immediate 5-bytes
• W bit says 32-bits or 8-bits
• D bit indicates direction
  \( \text{memory} \rightarrow \text{reg or reg} \rightarrow \text{memory} \)
  \( \text{movw} \text{EBX, [EDI + 45]} \)
  \( \text{movw [EDI + 45]}, \text{EBX} \)

Summary

• Procedure calls
• SPIM
• powerPC, Intel 80x86 ISA

Next Time
• Boolean Algebra, Logic Gates

Reading
• Appendix B