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

```assembly
.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 $14, 28($29)
        add $8, $14, 1
        sw $8, 20($29)
        b $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 (Frame Pointer)
- SP (Stack Pointer)
- ARUG (Argument Register)
- Local Variables
- callee save registers
- dynamic area
- grow and shrink during expression evaluation
- high mem
- low mem
- arguments and local variables at fixed offset from FP

MIPS Register Naming Conventions

- 0 zero constant
- 1 at reserved for assembler
- 2 v0 expression evaluation &
- 3 v1 function results
- 4 a0 arguments
- 5 a1
- 6 a2
- 7 a3
- 8 t0 temporary: caller saves
- 9 t1
- 10 t0 temporary: callee saves
- 11 t2
- 12 t3
- 13 t4
- 14 t5
- 15 t6
- 16 s0 callee saves
- 17 s1
- 18 temporary (cont'd)
- 19 t9
- 20 t8
- 21 t7
- 22 k0 reserved for OS kernel
- 23 k1
- 24 k0
- 25 gp Pointer to Global Area
- 26 sp Stack Pointer
- 27 fp frame Pointer
- 28 ra Return Address (HW)
**Example: Factorial**

Stack

<table>
<thead>
<tr>
<th>Old $r1</th>
<th>Old $fp</th>
<th>Old $r0</th>
<th>Old $r9</th>
<th>Old $fp</th>
<th>Old $r8</th>
<th>Old $fp</th>
<th>Old $r7</th>
<th>Old $fp</th>
<th>Old $r6</th>
<th>Old $fp</th>
<th>Old $r5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Main</td>
<td></td>
<td>fact(10)</td>
<td></td>
<td>fact(9)</td>
<td></td>
<td>fact(8)</td>
<td></td>
<td>fact(7)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stack</td>
<td>Stack grows</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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 $t0.

**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 $a0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>print</td>
<td>float $f12</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>print</td>
<td>double $f12</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>print</td>
<td>string $a0, string address</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>read</td>
<td>int integer in $v0</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>read</td>
<td>float float in $f0</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>read</td>
<td>double double in $f0</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>read</td>
<td>string $a0=buffer, $a1=length</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>sbrk</td>
<td>$a0=amount address in $v0</td>
<td></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
```

**Example 2**

```
# Example for CPS 104
# Program to add together array of 3 numbers.
.text
.globl main
main:
    subu $sp, 40
    sw $ra, 36($sp)
    sw $s3, 32($sp)
    sw $s2, 28($sp)
    sw $s1, 24($sp)
    sw $s0, 20($sp)
    move $v0, $0
    move $s1, $0
    la $s0, list  # Initialization
    la $s2, msg
    la $s3, list+36
```

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

```
main:
    jal main
```

```
.globl main
main:
    subu $sp, 40  # Push the stack
    sw $ra, 36($sp)
    sw $s3, 32($sp)
    sw $s2, 24($sp)
    sw $s1, 20($sp)
    move $v0, $0
```

```
move $a0, $v0
move $a1, $a0
move $a2, $a0
```

```
```
Example 2 (Continued)

Main code segment

`again: # Begin main loop
lw  $t6, 0($s0)     #\  Actual "work"
addu $s1, $s1, $t6   #/ SPIM I/O
#  SPIM I/O
li      $v0, 4          #\  Print a string
move    $a0, $s2        # >
syscall                 #/  Print a number
li      $v0, 1          #\  Print a string (eol)
move    $a0, $s1        # >
syscall                 #/  end of loop
addu    $s0, $s0, 4       #\  index update and
bne     $s0, $s3, again   #/  end of 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\(^n\) \times index)]
  - scale = 0,1,2,3
  - base any GPR, Index not ESP
- Base + scaled Index + displacement
  - mem[reg + (2\(^n\) \times index) + displacement]
  - scale = 0,1,2,3
  - base any GPR, Index not ESP

Condition Codes

- Both PowePC 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
  - bzt 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
  - memory = reg or reg = memory
  - move EBX, [EDI + 40]
  - move [EDI + 40], EBX

Summary

- Procedure calls
- SPIM
- powerPC, Intel 80x86 ISA
- Next Time
- Boolean Algebra, Logic Gates
- Reading
- Appendix B