## Outline

- ISAs in General
- MIPS Assembly Programming
- Other Instruction Sets



## But first: SPIM

- SPIM is a program that simulates the behavior of MIPS32 computers
  - Can run MIPS32 assembly language programs
  - You will use SPIM to run/test the assembly language programs you write for homeworks in this class
- Two flavors of same thing:
  - spim: command line interface
  - xspim: xwindows interface



## MIPS Assembly Language

- One instruction per line
- Numbers are base-10 integers or Hex with leading 0x
- Identifiers: alphanumeric, _, . string starting in a letter or _
- Labels: identifiers starting at the beginning of a line followed by ":"
- Comments: everything following # until end-of-line
- Instruction format: Space and "," separated fields
  - [Label:] <op> reg1, [reg2], [reg3] [# comment]
  - [Label:] <op> reg1, offset(reg2) [# comment]
  - .Directive [arg1], [arg2], . . .



## MIPS Pseudo-Instructions

- Pseudo-instructions: extend the instruction set for convenience
- Examples

```
move $2, $4           # $2 = $4, (copy $4 to $2)
Translates to:
add $2, $4, $0

li $8, 40             # $8 = 40, (load 40 into $8)
addi  $8, $0, 40

sd $4, 0($29)         # mem[$29] = $4; Mem[$29+4] = $5
sw $4, 0($29)
sw $5, 4($29)

la $4, 0x1000056c     # Load address $4 = <address>
lui $4, 0x1000        # load upper immediate (lui)
ori $4, $4, 0x056c    # or immediate (ori)
```

## Assembly Language (cont.)

- Directives: tell the assembler what to do
- Format "."<string> [arg1], [arg2] . . .

- Examples

| | |
|---|---|
| .data [address] | # start a data segment |
| .text [address] | # start a code segment |
| .align n | # align segment on $2^n$ byte boundary |
| .ascii <string> | # store a string in memory |
| .asciiz <string> | # store null-terminated string in memory |
| .word w1, w2, . . . , wn | # store n words in memory |

Let's see how these get used in programs …



## A Simple Program

- Add two numbers x and y:

```
  .text             # declare text segment
  .align 2          # align it on 4-byte (word) boundary
main:               # label for main
  la   $3, x        # load address of x into R3 (pseudo-inst)
  lw   $4, 0($3)    # load value of x into R4
  la   $3, y        # load address of y into R3 (pseudo-inst)
  lw   $5, 0($3)    # load value of y into R5
  add  $6, $4, $5   # compute x+y
  jr   $31          # return to calling routine

  .data             # declare data segment
  .align 2          # align it on 4-byte boundary
x:.word 10          # initialize x to 10
y:.word 3           # initialize y to 3
```

*Note: program doesn't obey register conventions*



## Another example: The C / C++ code

```
#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 …**



## Assembly Language Example 1

```
  .text
  .align   2
main:
  move $14, $0 # i = 0
  move $15, $0 # tmp = 0
  move $16, $0 # sum = 0
loop:
  mul $15, $14, $14 # tmp = i*i
  add $16, $16, $15 # sum = sum + tmp
  addi $14, $14, 1   # i++
  ble $14, 100, loop # if i < 100, goto loop

# how are we going to print the answer here?
# and how are we going to exit the program?
```

## System Call Instruction

- System call is used to communicate with the operating system and request services (memory allocation, I/O)
  - syscall instruction in MIPS
- SPIM supports "system-call-like"

1. Load system call code into register $v0
   - Example: if $v0==1, then syscall will print an integer
2. Load arguments (if any) into registers $a0, $a1, or $f12 (for floating point)
3. syscall

- Results returned in registers $v0 or $f0




## SPIM System Call Support

| code | service | ArgType | Arg/Result |
|---|---|---|---|
| 1 | print | int | $a0 |
| 2 | print | float | $f12 |
| 3 | print | double | $f12 |
| 4 | print | string | $a0 (string address) |
| 5 | read | integer | integer in $v0 |
| 6 | read | float | float in $f0 |
| 7 | read | double | double in $f0 & $f1 |
| 8 | read | string | $a0=buffer, $a1=length |
| 9 | sbrk | $a0=amount | address in $v0 |
| 10 | exit | | |




## 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

# code continues on next slide …
```




## 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"
```