TAs
• Name: 陳明軒 (包子)  
   Ming-Shiuan Chen

• Gender: Male

• Email: intere2960@cmlab.csie.ntu.edu.tw
• Name : 廖以圻 (Chi-Chi)

Chi-Chi Lao

• Gender : Male

• Email :

chichi@cmlab.csie.ntu.edu.tw
Outline

• Assembly Language
• SPIM
• MIPS
• Homework 2
Assembly Language
Assembly Language

• Assembly language
  Symbolic representation of a computer’s binary encoding

• Machine code
  Computer’s binary encoding

Assembler
  Translates assembly language into binary instructions
Assembly Language

High-level language program

Assembly language program

Program

Compiler

Assembler

Linker

Computer

FIGURE B.1.6  Assembly language either is written by a programmer or is the output of a compiler.
Why Assembly

• **A low level language**
  the code and syntax is much closer to the computer's processor

• **Direct hardware manipulation**
  device drivers, low-level embedded systems, and real-time systems

• **Speed optimization**
  performance and efficiency
• To write in assembly is to understand exactly how the processor and memory work together to "make things happen".

Sometimes to debug a higher-level language, you have to review the resulting assembly language.
What is SPIM

- **MIPS32 Simulator**
  reads and executes assembly language program written for MIPS 32-bit architecture

- **SPIM does not execute binary programs**
  provides a simple debugger and minimal set of operating system services

- **SPIM implements both a terminal**
QtSPIM Installation

SPIM: A MIPS32 Simulator

James Larus
spim@larusstone.org

Contents
- Older Versions of SPIM
- Further Information
- Changes to SPIM
- Copyrignt

Spim is a self-contained simulator that runs MIPS32 programs. It reads and executes assembly language programs written for this processor. Spim also provides a simple debugger and minimal set of operating system services. Spim does not execute binary (compiled) programs.

Spim implements almost the entire MIPS32 assembler-extended instruction set. (It omits most floating point comparisons and rounding modes and the memory system page tables.) The MIPS architecture has several variants that differ in various ways (e.g., the MIPS64 architecture supports 64-bit integers and addresses), which means that Spim will not run programs for all MIPS processors.

What's New?

QtSpim is a new user interface for Spim built on the Qt UI framework. Qt is cross-platform, so the same user interface and same code will run on Windows, Linux, and Mac OS X (yeah!). Moreover, the interface is clean and up-to-date (unlike the archaic X windows interface).

Spim has moved to SourceForge! The source code for all version of Spim are in an SVN repository and compiled version are available for download. There is also a bug tracker and discussion forum. Spim is an open source project, so please join in and contribute.

Spim comes with complete source code and documentation.

Spim implements both a terminal and windows interfaces. On Microsoft Windows, Linux, and Mac OS X, the spim program offers a simple terminal interface and the QtSpim program provides the windowing interface. The older programs xspim and PCSpim provide window interfaces for these systems as well.

Download SPIM
# QtSPIM Installation

## spim mips simulator

Brought to you by: jamasiarus

<table>
<thead>
<tr>
<th>Name</th>
<th>Modified</th>
<th>Size</th>
<th>Downloads / Week</th>
</tr>
</thead>
<tbody>
<tr>
<td>qtspim_9.1.16_linux32.deb</td>
<td>2015-08-26</td>
<td>29.7 MB</td>
<td>71</td>
</tr>
<tr>
<td>QtSpim_9.1.16_mac.mpkg.zip</td>
<td>2015-08-26</td>
<td>28.7 MB</td>
<td>248</td>
</tr>
<tr>
<td>qtspim_9.1.16_linux64.deb</td>
<td>2015-08-26</td>
<td>28.1 MB</td>
<td>137</td>
</tr>
<tr>
<td>QtSpim_9.1.16_Windows.exe</td>
<td>2015-08-26</td>
<td>34.3 MB</td>
<td>1,014</td>
</tr>
<tr>
<td>QtSpim_9.1.15_mac.mpkg.zip</td>
<td>2015-04-26</td>
<td>28.7 MB</td>
<td>128</td>
</tr>
<tr>
<td>QtSpim_9.1.13_mac.mpkg.zip</td>
<td>2014-02-05</td>
<td>33.1 MB</td>
<td>2</td>
</tr>
<tr>
<td>QtSpim_9.1.12_Windows.exe</td>
<td>2013-12-17</td>
<td>31.5 MB</td>
<td>547</td>
</tr>
<tr>
<td>qtspim_9.1.12_linux32.deb</td>
<td>2013-12-14</td>
<td>1.1 MB</td>
<td>10</td>
</tr>
<tr>
<td>qtspim_9.1.12_linux64.deb</td>
<td>2013-12-14</td>
<td>1.1 MB</td>
<td>120</td>
</tr>
<tr>
<td>qtspim_9.1.9_linux64.deb</td>
<td>2013-01-23</td>
<td>1.1 MB</td>
<td>3</td>
</tr>
</tbody>
</table>

Looking for the latest version? [Download QtSim_9.1.12_Windows.exe (31.5 MB)](https://example.com)
References of SPIM

• Official website of SPIM: http://spimsimulator.sourceforge.net/

• Assemblers, Linkers, and the SPIM Simulator: http://pages.cs.wisc.edu/~larus/HP_AppA.pdf

• MIPS Instruction Reference: http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html
MIPS
MIPS memory layout

- MIPS 32-bit CPU (all registers are 32 bits wide)
  accessible memory range: 0x00000000–0xFFFFFFFF
- Memory holds both instructions (text) and data
  If a program is loaded into SPIM, its .text segment is automatically placed at 0x00400000, its .data segment at 0x10000000
## MIPS Assembly

<table>
<thead>
<tr>
<th>Operation Code (Opcode)</th>
<th>Arithmetic Instructions</th>
<th>Data Transfer Instructions</th>
<th>Logic Instructions</th>
<th>Branch and Jump-Related Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• add, sub, addi, addu, addiu, subu</td>
<td>• lw, sw, lbu, sb, lui, ori</td>
<td>• beq, bne, slt, slti, sltu</td>
<td>• j, jr, jal</td>
</tr>
</tbody>
</table>

Computer Organization and Structure 2015
### MIPS Registers and Usage Convention

<table>
<thead>
<tr>
<th>Register</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>$zero</td>
<td>constant 0</td>
</tr>
<tr>
<td>$v0, $v1</td>
<td>expression of a function</td>
</tr>
<tr>
<td>$a0 ~ $a3</td>
<td>argument 1~4</td>
</tr>
<tr>
<td>$t0 ~ $t9</td>
<td>temporary registers</td>
</tr>
<tr>
<td>$s0 ~ $s7</td>
<td>save registers</td>
</tr>
<tr>
<td>$sp</td>
<td>stack pointer</td>
</tr>
<tr>
<td>$fp</td>
<td>frame pointer</td>
</tr>
<tr>
<td>$ra</td>
<td>return address</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
### MIPS Assembly

#### Some data types in MIPS

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.word, .half</td>
<td>32/16 bit integer</td>
</tr>
<tr>
<td>.byte</td>
<td>8 bit integer</td>
</tr>
<tr>
<td>.ascii, .asciiz</td>
<td>string</td>
</tr>
<tr>
<td>.double, .float</td>
<td>floating point</td>
</tr>
</tbody>
</table>
Assembler Syntax

- **Comment**: ( # )
  
  Everything from the sharp sign to the end of the line is ignored

- **Identifier**: (A sequence of alphanumeric characters, _, and .)
  
  Identifier are a sequence of alphanumeric characters, underscores (_), and dots (.) that do not begin with a number

- **Instruction Opcode**

  Instruction opcodes are reserved words that are not valid identifiers

- **Label**

  Labels are declared by putting them at the beginning of a line followed by a colon.
```c
int main()
{
    printf("Hello World\n");
    return 0;
}
```

### MIPS

```
.data
Mystr: .asciiz "Hello World\n"

.text
main:
    li $v0, 4
    la $a0, Mystr
    syscall
    li $v0, 10
    syscall
```
MIPS — Hello World

MIPS Architecture

Put Static Data Here

.data
Mystr: .asciiz “Hello World\n”
Yourint: .word 75
Hisarray: .word 100, 100, 100
    .word 20, 40, 60
    .word 1, 2, 3

.text
......

Put Your Code Here
MIPS — Hello World

MIPS Architecture

Put Static Data Here

.data

.text

main:
# do anything you want
......
# end of the program
li $v0, 10
syscall

Put Your Code Here
MIPS System Calls

• SPIM provides a small set of operating-system-like services through the system call instruction.

• A program loads the system call code into register $v0 and arguments into registers $a0-$a3 (or $f12 for floating-point values).

• System calls that return values put their results in register $v0 (or $f0 for floating-point results).
# MIPS System Calls

<table>
<thead>
<tr>
<th>Service</th>
<th>System call code</th>
<th>Arguments</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>print_int</td>
<td>1</td>
<td>$a0 = integer</td>
<td></td>
</tr>
<tr>
<td>print_float</td>
<td>2</td>
<td>$f12 = float</td>
<td></td>
</tr>
<tr>
<td>print_double</td>
<td>3</td>
<td>$f12 = double</td>
<td></td>
</tr>
<tr>
<td>print_string</td>
<td>4</td>
<td>$a0 = string</td>
<td></td>
</tr>
<tr>
<td>read_int</td>
<td>5</td>
<td></td>
<td>integer (in $v0)</td>
</tr>
<tr>
<td>read_float</td>
<td>6</td>
<td></td>
<td>float (in $f0)</td>
</tr>
<tr>
<td>read_double</td>
<td>7</td>
<td></td>
<td>double (in $f0)</td>
</tr>
<tr>
<td>read_string</td>
<td>8</td>
<td>$a0 = buffer, $a1 = length</td>
<td></td>
</tr>
<tr>
<td>sbrk</td>
<td>9</td>
<td>$a0 = amount</td>
<td>address (in $v0)</td>
</tr>
<tr>
<td>exit</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>print_char</td>
<td>11</td>
<td>$a0 = char</td>
<td></td>
</tr>
<tr>
<td>read_char</td>
<td>12</td>
<td>$a0 = filename (string), $a1 = flags, $a2 = mode</td>
<td>char (in $a0)</td>
</tr>
<tr>
<td>open</td>
<td>13</td>
<td>$a0 = file descriptor, $a1 = flags, $a2 = mode</td>
<td>file descriptor (in $a0)</td>
</tr>
<tr>
<td>read</td>
<td>14</td>
<td>$a0 = file descriptor, $a1 = buffer, $a2 = length</td>
<td>num chars read (in $a0)</td>
</tr>
<tr>
<td>write</td>
<td>15</td>
<td>$a0 = file descriptor, $a1 = buffer, $a2 = length</td>
<td>num chars written (in $a0)</td>
</tr>
<tr>
<td>close</td>
<td>16</td>
<td>$a0 = file descriptor</td>
<td></td>
</tr>
<tr>
<td>exit2</td>
<td>17</td>
<td>$a0 = result</td>
<td></td>
</tr>
</tbody>
</table>
MIPS System Calls - Example

```
data
    str:
        .asciiz "the answer = 
.text
    li $v0, 4      # system call code for print_str
    la $a0, str   # address of string to print
    syscall       # print the string
    li $v0, 1      # system call code for print_int
    li $a0, 5      # integer to print
    syscall       # print it
    li $v0, 10     # system call code for exit
    syscall
```

Output: “the answer = 5”
Execute Program in

1. Write your own assembly program, and save it as .sfile
2. Simulator - Reinitialize Simulator
3. Open your .s file
4. Simulator - Clear Registers
5. Simulator - Run / Continue
Homework 2

• This is an individual assignment

• Plagiarism will be heavily punished

• Write the following three programs in MIPS assembly language. (Must run correctly on SPIM)

<table>
<thead>
<tr>
<th>Find out all prime numbers</th>
<th>Tower of Hanoi</th>
<th>Greatest Common Divisor</th>
</tr>
</thead>
</table>

• One bonus program: Binary Search
Documentation (20%)

• Detailed documentation for each program is required
• The following parts must be included:
  ■ Your name, student ID, and email address
  ■ Explanation of the design or the flow of each program
  ■ What you’ve learnt from writing the programs
• Problems or difficulties you’ve encountered during writing the programs are nice to be included in the document
Problem 1: Find out all prime numbers

• **Input:**
a positive integer $n > 1$

• **Output:**
All prime numbers which is smaller than $n$

• **Requirements:**
  1. Print the correct answer.
  2. Can do many calculations iteratively
  3. The file name is *FindPrime.s*
Problem 2: Tower of Hanoi

- A hanoi tower with 3 rods A, B, C and n disks. Move all the disks from A to C.
- **Input:**
a positive integer n, $1 < n \leq 5$
- **Output:**
Print all the steps
- **Requirements:**
Print the correct step.
The file name is `Hanoi.s`
Problem 3: GCD

- **Input:**
  two positive integers $x, y > 1$

- **Output:**
  the greatest common divisor of $x$ and $y$. Output $\text{gcd}(x, y)$

- **Requirements:**
  1. Print the correct answer
  2. Can do many calculations iteratively
  3. The file name is GCD.s
Optional: Binary Search

• **Input:**
  1. *n* positive integers, where *n* < 8
  2. value to search

• **Output:**
  1. If value can be find, print “Find it.”.
  2. If value not in the sequence, print “The value is not in this sequence.”.

• **Requirements:**
  Print the correct answer.

The file name is *Binary_Search.s*.
Submission

• Deadline: 11:59 PM, Monday, Oct. 19, 2015
• You must submit at least the following files:
  - FindPrime.s
  - Hanoi.s
  - GCD.s
  - Binary_Search.s (Optional)
  - (Your student id)_hw2_document.pdf

• The attach filename should be like b03xxxxxx.zip
• Email your zipped file to TA:
  intere2960@cmlab.csie.ntu.edu.tw
## Grading Guidelines

<table>
<thead>
<tr>
<th>Description</th>
<th>For Each Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program runs without error messages</td>
<td>10%</td>
</tr>
<tr>
<td>Program executes correctly</td>
<td>60%</td>
</tr>
<tr>
<td>Documentation and description</td>
<td>20%</td>
</tr>
<tr>
<td>Implementation Detail</td>
<td>10%</td>
</tr>
</tbody>
</table>
Deadline

• Late submission policy

• 10% off from your total score each day
Contact Information

• **TA Hours @ 管院 一館五樓 503-C**

Chi-Chi Liao(廖以圻) Fir. 8:00~10:00

Ming-Shiuan Chen(陳明軒) Tus. 13:00~15:00

• **Contact us if you have any problem**

Chi-Chi: chichi@cmlab.csie.ntu.edu.tw

包子: interes2960@cmlab.csie.ntu.edu.tw
Thank You For
Your Attention