USING A SIMULATOR

QUICK INTRODUCTION
From various sources
For cs470

INTRODUCTION
MARS INTERFACE
SIMULATOR USAGE
MIPS ASSEMBLER LANGUAGE
PROGRAM EXAMPLE
Introduction

MARS IDE

- A simulator for the MIPS processor
- Pete Sanderson, Ken Vollmar
- Similar to SPIM discussed in Appendix A

Functions

- Text editor to writing code
- **Loads and executes** MIPS assembly language files (.asm or .s) immediately
- Debugger (**breakpoints**, **single-stepping**)
- Provides **Console** for input/output
INTERFACE

Registers window
- values of all registers Int/FP

Text segment window
- user and kernel instructions
- Addresses, code in binary and assembly

Data segment window
- User data, stack, kernel data

Messages window
- messages (include error messages)

Console interface
Input/output
Simulator Settings
Execute Program

![SPIM Simulator Screenshot](image)

- **PC**: 400002
- **HI**: 0
- **LO**: 0
- **IC**: 0

Your assembly code starts:

- **R16 (x0)** = 0
- **R17 (xl)** = 5
- **R18 (x2)** = 0
- **R19 (x3)** = 0
- **R20 (x4)** = 0
- **R21 (x5)** = 0
- **R22 (x6)** = 0
- **R23 (x7)** = 400002
- **R24 (t8)** = 0
- **R25 (t9)** = 0
- **R26 (t10)** = 0
- **R27 (t11)** = 0
- **R28 (t12)** = 0
- **R29 (t13)** = 0
- **R30 (t14)** = 0

Register change resulting from previous instruction:

- **R15 (t7)** = 0
- **R16 (x0)** = 0
- **R17 (xl)** = 5
- **R18 (x2)** = 0
- **R19 (x3)** = 0
- **R20 (x4)** = 0
- **R21 (x5)** = 0
- **R22 (x6)** = 0
- **R23 (x7)** = 400002
- **R24 (t8)** = 0
- **R25 (t9)** = 0
- **R26 (t10)** = 0
- **R27 (t11)** = 0
- **R28 (t12)** = 0
- **R29 (t13)** = 0
- **R30 (t14)** = 0

Current instruction:

- **User Text Segment** [00400000]...[00440000]

Kernel Text Segment [80000000]...[80001000]
Program Text and Data segments
Set a break point
Simulator Usage

- The icon runs the program to completion. Using this icon, you should observe the yellow highlight showing the program’s progress and the values of the Fibonacci sequence appearing in the Data Segment display.

- The icon resets the program and simulator to initial values. Memory contents are those specified within the program, and register contents are generally zero.

- The icon is “single-step.” Its complement is , “single-step backwards” (undoes each operation).
# MIPS Assembly Layout

## Program Layout

<table>
<thead>
<tr>
<th>Code Section</th>
<th>Data Section</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.text</code></td>
<td><code>.data</code></td>
</tr>
<tr>
<td><code>.globl main</code></td>
<td><code>.data_type</code></td>
</tr>
</tbody>
</table>

- **.text**  #code section
- **.globl main**  #starting point: must be global
- **main:**  # user program code
- **label:**  `.data_type`  *list_of_data*
  #data loc + data type + data

- **.text**  #code section
- **label:**  #function label
  #user functions
MIPS Assembler Directives

Data Types

- **.word, .half** - 32/16 bit integer
- **.byte** - 8 bit integer (similar to ‘char’ type in C)
- **.asci, .asciiz** - string (asciiz is null terminated)
  - Strings are enclosed in double-quotas ("")
  - Special characters in strings follow the C convention
    - newline(\n), tab(\t), quote("")
- **.double, .float** - floating point

Other Directives

- **.text** - Indicates that following items are stored in the user text segment
- **.data** - Indicates that following data items are stored in the data segment
- **.globl sym** - Declare that symbol sym is global and can be referenced from other files
SPIM Program Example I

#sample example 'load, add, store'
.text
.globl main
main:  la $t0, value
     lw $t1, 0($t0)
     lw $t2, 4($t0)
     add $t3, $t1, $t2
     sw $t3, 8($t0)
     li $v0, 10
     syscall
.data
value: .word 10, 20, 0
#this ends the program.

# text section
# call main by SPIM
# load address 'value' into $t0
# load word 0(value) into $t1
# load word 4(value) into $t2
# add two numbers into $t3
# store word $t3 into 8($t0)
# 10 exit syscall.
# do the syscall.
# data section
# data for addition
# Common Pseudoinstructions

<table>
<thead>
<tr>
<th>Pseudoinstruction</th>
<th>Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mov $rt, $rs</code></td>
<td><code>add $rt, $rs, $0</code></td>
</tr>
<tr>
<td><code>li $rs, small</code></td>
<td><code>addi $rs, $0, small</code></td>
</tr>
</tbody>
</table>
| `li $rs, big`           | `lui $rs, upper( big )
  ori $rs, $rs, lower( big )`                    |
| `la $rs, big`           | `lui $rs, upper( big )
  ori $rs, $rs, lower( big )`                    |
| `lw $rt, big($rs)`     | `lui $t0, upper( big )
  ori $t0, $t0, lower( big )
  add $t0, $rs, $t0
  lw $rt, 0($t0)`                            |
SPIM System Calls

System Calls (syscall): OS services

- Load system call code into register $v0
- Load arguments into registers $a0…
- After call, return value (if any) is in register $v0

Frequently used system calls

<table>
<thead>
<tr>
<th>Service</th>
<th>Code($v0)</th>
<th>Arg</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Print_int</td>
<td>1</td>
<td>$a0 (int to print)</td>
<td></td>
</tr>
<tr>
<td>Print_string</td>
<td>4</td>
<td>$a0 (addr of string)</td>
<td></td>
</tr>
<tr>
<td>Read_int</td>
<td>5</td>
<td></td>
<td>Int in $v0</td>
</tr>
<tr>
<td>Read_string</td>
<td>8</td>
<td>$a0: addr, $a1:bytes</td>
<td></td>
</tr>
</tbody>
</table>
```asm
# sample example 'system call'
.text
.globl main

main:
  la $t0, value
  li $v0, 5    # get input
  syscall
  sw $v0, 0($t0)

  li $v0, 5    # get input
  syscall
  sw $v0, 4($t0)

lw $t1, 0($t0)
lw $t2, 4($t0)
add $t3, $t1, $t2
sw $t3, 8($t0)

li $v0, 4    # print message
la $a0, msg1
syscall

li $v0, 1    # print result
move $a0, $t3
syscall

.data
value: .word 0, 0, 0
msg1: .asciiz "Result = "
```
### Psuedoinstructions

<table>
<thead>
<tr>
<th>Name</th>
<th>instruction syntax</th>
<th>Real instruction translation</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move</td>
<td>move $rt,$rs</td>
<td>addi $rt,$rs,0</td>
<td>$R[rt]=R[rs]$</td>
</tr>
<tr>
<td>Clear</td>
<td>clear $rt</td>
<td>add $rt,$zero,$zero</td>
<td>$R[rt]=0$</td>
</tr>
<tr>
<td>Load Address</td>
<td>la $rd, LabelAddr</td>
<td>lui $rd, LabelAddr[31:16];</td>
<td>$rd = Label Address</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ori $rd,$rd, LabelAddr[15:0];</td>
<td></td>
</tr>
<tr>
<td>Load Immediate</td>
<td>li $rd, IMMED[31:0]</td>
<td>lui $rd, IMMED[31:16];</td>
<td>$rd = 32 bit Immediate value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ori $rd,$rd, IMMED[15:0];</td>
<td></td>
</tr>
<tr>
<td>Branch unconditionally</td>
<td>b Label</td>
<td>beq $zero,$zero,Label</td>
<td>if(R[rs]==R[rt]) PC=Label</td>
</tr>
<tr>
<td>Branch and link</td>
<td>bal $rs,Label</td>
<td>bgezal $zero,Label</td>
<td>if(R[rs]&gt;=0) PC=Label</td>
</tr>
<tr>
<td>Branch if greater than</td>
<td>bgt $rs,$rt,Label</td>
<td>slt $at,$rt,$rs;</td>
<td>if(R[rs]&gt;R[rt]) PC=Label</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bne $at,$zero,Label</td>
<td></td>
</tr>
<tr>
<td>Branch if less than</td>
<td>blt $rs,$rt,Label</td>
<td>slt $at,$rs,$rt;</td>
<td>if(R[rs]&lt;R[rt]) PC=Label</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bne $at,$zero,Label</td>
<td></td>
</tr>
<tr>
<td>Branch if greater than or</td>
<td>bge $rs,$rt,Label</td>
<td>slt $at,$rs,$rt;</td>
<td>if(R[rs]&gt;=R[rt]) PC=Label</td>
</tr>
<tr>
<td>equal</td>
<td></td>
<td>bne $at,$zero,Label</td>
<td></td>
</tr>
<tr>
<td>Branch if less than or equal</td>
<td>ble $rs,$rt,Label</td>
<td>slt $at,$rt,$rs;</td>
<td>if(R[rs]&lt;=R[rt]) PC=Label</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bne $at,$zero,Label</td>
<td></td>
</tr>
<tr>
<td>Branch if greater than</td>
<td>bgtu $rs,$rt,Label</td>
<td>slt $at,$rt,$rs;</td>
<td>if(R[rs]&gt;R[rt]) PC=Label</td>
</tr>
<tr>
<td>unsigned</td>
<td></td>
<td>beq $at,$zero,Label</td>
<td></td>
</tr>
<tr>
<td>Branch if greater than zero</td>
<td>bgtz $rs,$rt,Label</td>
<td>slt $at,$rs,$rt;</td>
<td>if(R[rs]&gt;0) PC=Label</td>
</tr>
<tr>
<td>Multiplies and returns only</td>
<td>mul $d, $s, $t</td>
<td>mult $s, $t; mflo $d</td>
<td>$d = $s * $t</td>
</tr>
<tr>
<td>first 32 bits</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Divides and returns quotient</td>
<td>div $d, $s, $t</td>
<td>div $s, $t; mflo $d</td>
<td>$d = $s / $t</td>
</tr>
<tr>
<td>Divides and returns remainder</td>
<td>rem $d, $s, $t</td>
<td>div $s, $t; mfhi $d</td>
<td>$d = $s % $t</td>
</tr>
</tbody>
</table>

---

*Note: The 'instruction syntax' column assumes a simplified instruction set architecture (ISA) similar to that used in the MIPS architecture, where `$rt`, `$rs`, `$rd`, `$at`, `$zero`, `$IMMED`, and `$LabelAddr` represent register and immediate values.*

---