Chapter 2

QTSpim Simulator

QTSpim Introduction

- What is QTSpim?
  - A simulator that runs assembly programs for MIPS R2000/R3000 RISC computers.

- What does QTSpim do?
  - Reads MIPS assembly language files and translates to machine language.
  - Executes the machine language instructions.
  - Shows contents of registers and memory.
  - Works as a debugger (supports break-points and single-stepping).
  - Provides basic Operating System - like services, like simple I/O.
Template for a MIPS Assembly Language Program

# Comment giving name of program and description of function
# Template.s
# Bare-bones outline of MIPS assembly language program

.data
# variable declarations here
# ...

.text

main: # indicates start of code (first instruction to execute)
# remainder of program code here
# ...
# ...

MIPS Assembly Language Program Structure

- QTSpim source files are just a plain text file.
- Source file name should end in suffix .s or .asm
- .data directive
  - Placed in section of program identified with the assembler directive .data
  - Declares variable names used in program; storage allocated in main memory (RAM).
- .text directive
  - Placed in section of program identified with the assembler directive .text
  - Contains program code (instructions).
  - Starting point for code execution given label main:
  - Ending point of main code should use exit system call (covered later under System Calls).
Example Program: add2numbersProg1.asm

## Program adds 10 and 11

```
.text                   # text section
.globl main            # call main by SPIM

main:
  ori     $8,$0,0xA       # load "10" into register 8
  ori     $9,$0,0xB       # load "11" into register 9
  add     $10,$8,$9       # add registers 8 and 9, put result
                     # in register 10
```

Common QTSpim Directives

- `.globl` `sym`
  Declare that the symbol `sym` is global and can be referenced from other files.

- `.word` `w1, ..., wn`
  Store n 32-bit quantities in successive memory words.

- `.byte` `b1, ..., bn`
  Store n 8-bit quantities in successive memory bytes.

- `.ascii` `str`
  Store the string in memory but do not null-terminate it:
  - Characters are represented in single quotes ‘a’
  - Strings are represented in double-quotes “str”
  - Special characters, e.g. \n, \t, follow C convention.

- `.asciiz` `str`
  Store the string in memory and null-terminate it.
Less Common Directives

- `.float` f1, ..., fn
  - Store n floating point single precision numbers in successive memory locations.
- `.double` d1, ..., dn
  - Store n floating point double precision numbers in successive memory locations.
- `.space` n
  - Reserves n successive bytes of space.
- `.align` n
  - Align the next datum on a 2^n byte boundary. For example, `.align 2` aligns next value on a word boundary. `.align 0` turns off automatic alignment of `.half`, `.word`, etc. till next `.data` directive.

QTSpim Pseudoinstructions

- Most assembler instructions represent machine instructions one-to-one.
- Pseudoinstructions examples:
  - `move $t0, $t1` → `add $t0, $zero, $t1`
  - `li $t4, 4` → `ori $t4, $0, 4`
  - `la $t2, message` → `lui $t2, (upper 16 bits) ori $t2, $t2, (lower 16)`
  - `blt $t0, $t1, L` → `slt $at, $t0, $t1 bne $at, $zero, L`
- There is a good chance that QTSpim will use register 1 ($at) as a temporary location.
QTSpim System Calls

- System Calls (syscall)
  - Operating systems-like services.
- Method
  - Load system call code into register $v0 (see following table for codes).
  - Load arguments into registers $a0, ..., $a3
  - Call system with SPIM instruction `syscall`
  - After call, return value is in register $v0, or $f0 for floating point results.

QTSpim System Call Codes

<table>
<thead>
<tr>
<th>Service</th>
<th>Code (put in $v0)</th>
<th>Arguments</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>print_int</td>
<td>1</td>
<td>$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=address 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_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>addr in $v0</td>
</tr>
<tr>
<td>exit</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example QTSpim Print Program

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

.text
main:
  li  $v0, 4  # Load immediate; 4 is the code for print_string
  la  $a0, str  # The print_string syscall expects the string
  syscall  # address as the argument; la is the instruction
  # to load the address of the operand (str).
  syscall  # QTSpim will now invoke syscall #4
  li  $v0, 1  # syscall #1 corresponds to print_int
  li  $a0, 5  # print_int expects the integer as its argument
  syscall  # QTSpim will now invoke syscall #1
```

Example MIPS Program

```
.text
.globl main
main:
  li  $v0, 4
  la  $a0, str1
  syscall
  li  $v0, 5
  syscall
  add  $t0, $v0, $zero
  li  $v0, 5
  syscall
  add  $t1, $v0, $zero
  li  $v0, 4
  la  $a0, str2
  syscall
  li  $v0, 1
  add  $a0, $t1, $t0
  syscall
```

Write an assembly program to prompt the user for two numbers and print the sum of the two numbers.
QTSpim Example Program: systemCalls.asm

```assembly
## Enter two integers in
## console window
## Sum is displayed
.text
.globl main
main:
  la $t0, value
  li $v0, 5
  syscall
  sw $v0, 0($t0)
  li $v0, 5
  syscall
  sw $v0, 4($t0)
  lw $t1, 0($t0)
  lw $t2, 4($t0)
  add $t3, $t1, $t2
  sw $t3, 8($t0)
  li $v0, 4
  la $a0, msg1
  syscall
  li $v0, 1
  move $a0, $t3
  syscall
  li $v0, 10
  syscall
.data
value: .word 0, 0, 0
msg1: .asciiz "Sum = "
```

More QTSpim Example Programs

- On the course web page, you will find a link to an examples zip file that has 18 simple and well-documented MIPS assembly programs. Run the code in the order below of increasing complexity:

  1. add2numbersProg1
  2. add2numbersProg2
  3. storeWords
  4. swap2memoryWords
  5. branchJump
  6. systemCalls
  7. overflow
  8. averageOfBytes
  9. printLoop
  10. sumOfSquaresProg1
  11. sumOfSquaresProg2
  12. sumOfSquaresProg3
  13. procCallsProg1
  14. procCallsProg1Modified
  15. procCallsProg2
  16. addFirst100
  17. factorialNonRecursive
  18. factorialRecursive


**Using QTSpim**

- **Loading source file**
  - Use *File -> Load file*
    - Any grammatical errors will be flagged when you open your file.

- **Simulation**
  - *Simulator -> Settings*:
    - *Accept pseudo instructions* is the only box that should be checked.
  - *Simulator -> Run Parameters*: to load the *Program Counter* with the address of the first instruction:
    - *Enter Address Value* as "0x00400000"
  - *Simulator -> Run/Continue or Single Step* to run a loaded program.
  - *Simulator -> Stop*: stop execution.
  - *Simulator -> Clear Registers and Reinitialize*: clean-up before new run.

- **Breakpoints**
  - *Break points* are set/cleared by right-clicking on an instruction.

---

**Conclusions**

- The code presented so far should get you started in writing your own MIPS assembly.
- Remember the only way to master the MIPS assembly language – in fact, any computer language – is to *write lots and lots of code*.
- For anyone aspiring to understand modern computer architecture it is rather helpful to *master MIPS assembly* as all modern computers (since the mid-80’s) have been inspired by, if not based fully or partly on, the MIPS instruction set architecture.