



CS202

Lab1:Introduction



Simulator of MIPS



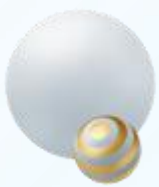
- BlackBoard site:
 - Computer Organization Spring 2022
- QQ group:
 - 869456435





Assignments Submission Rules

- All assignments **MUST be submitted to BlackBoard site**, any other forms of submission are NOT accepted.
- If the submission is delayed **for one day, 20% discount** on the total score. If it is **delayed for more than a week, any submission is NOT ACCEPTABLE!** This assignment is 0 point.
- In the case of plagiarism: at the 1st time, the assignment was 0 for all concerned students and at the 2nd time, the grade of the experimental course is 0 for all concerned students.
- Reminder:
 - Assignment scoring and the score publication would be completed within two weeks after the publication of assignment. **If you have any question about the score, please email the relevant reviewer in one week after the score publication.**



Experimental tool kits

- Learn and practice **MIPS** (QtSpim / Mars)
- Design and implement an **CPU**
(vivado + FPGA based Development Board)



- QtSpim
 - **QtSpim is the newest version of Spim that currently being actively maintained, runs on Microsoft Windows, Mac OS X, and Linux.**
 - Spim, A self-contained simulator that runs MIPS32 program developed by Prof. James Larus from University of Wisconsin-Madison
 - **QtSpim reads and executes programs written in assembly language (汇编语言) for a MIPS computer. QtSpim does not execute binary (compiled) programs.** To simplify programming, QtSpim provides a simple debugger and small set of operating system services
 - **QtSpim implements MOST of the MIPS32 assembler-extended instruction set.** (It omits the floating point comparisons and rounding modes and the memory system page tables.) which means that QtSpim will **NOT run programs for ALL MIPS processors.**

Name	Modified	Size
QtSpim_9.1.23_Windows.msi	2021-12-06	21.5 MB
qtspim_9.1.23_linux64.deb	2021-12-06	18.5 MB
QtSpim_9.1.23_mac.mpkg.zip	2021-12-06	22.2 MB

QtSpim (2)

QtSpim

load file clear register run the simulation in different ways

File Simulator Registers Text Segment Data Segment Window Help

FP Regs Int Regs [16] Data Text

Int Regs [16]

PC	= 0
EPC	= 0
Cause	= 0
BadVAddr	= 0
Status	= 3000ff10
HI	= 0
LO	= 0
R0 [r0]	= 0
R1 [at]	= 0
R2 [v0]	= 0
R3 [v1]	= 0
R4 [a0]	= 1
R5 [a1]	= 7ffff53c
R6 [a2]	= 7ffff544
R7 [a3]	= 0
R8 [t0]	= 0
R9 [t1]	= 0
R10 [t2]	= 0
R11 [t3]	= 0
R12 [t4]	= 0
R13 [t5]	= 0
R14 [t6]	= 0
R15 [t7]	= 0
R16 [s0]	= 0
R17 [s1]	= 0
R18 [s2]	= 0
R19 [s3]	= 0
R20 [s4]	= 0
R21 [s5]	= 0

display integer/floating-point register

Text

User Text Segment [00400000]..[00440000]

```
[00400000] lw $4, 0($29)
[00400004] addiu $5, $29, 4
[00400008] addiu $6, $5, 4
[0040000c] sll $2, $4, 2
[00400010] addu $6, $6, $2
[00400014] jal 0x00000000 [main]
[00400018] nop
[0040001c] ori $2, $0, 10
[00400020] syscall
```

Kernel Text Segment [80000000]..[80010000]

```
[80000180] addu $27, $0, $1
[80000184] lui $1, -28672
[80000188] sw $2, 512($1)
[8000018c] lui $1, -28672
[80000190] sw $4, 516($1)
[80000194] mfc0 $26, $13
[80000198] srl $4, $26, 2
[8000019c] andi $4, $4, 31
[800001a0] ori $2, $0, 4
[800001a4] lui $4, -28672 [ __m1_ ]
[800001a8] syscall
[800001ac] ori $2, $0, 1
[800001b0] srl $4, $26, 2
[800001b4] andi $4, $4, 31
[800001b8] syscall
[800001bc] ori $2, $0, 4
[800001c0] andi $4, $26, 60
[800001c4] lui $1, -28672
[800001c8] addu $1, $1, $4
[800001cc] lui $4, -28672
```

display instruction / data segment

Console

Memory and registers cleared

SPIM Version 9.1.20 of August 29, 2017
Copyright 1990-2017 by James Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
QtSPIM is linked to the Qt library, which is distributed under the GNU Lesser General Public License version 3 and version 2.1.

display status of Qtspim



QtSpim (3)

Load file and Do the simulation

Step1: loading a file (assembly code file usually have the extension “.s”)

- Open file : **File->Load File** or **File ->Reinitialize and Load File**
- Tips:
 - **Reinitialize and Load File** : **Clears** all the changes made by a program, deleting all of its instructions, then **reload** the last file
 - **Before doing the simulation, make sure clear registers**, you can do it by :

Simulator->Clear Registers or by related tools.

Step2:

- **Run file** : **Simulator->Run / Continue**
- The value of register in left window will be refreshed, the program's output will appear in the Console window.



<http://courses.missouristate.edu/kenvollmar/mars/download.htm>

D:\计算机组成原理\2020s\mips1.asm - MARS 4.5

File Edit Run Settings Tools Help

new open assembler Run speed at max (no interaction) show Registers' information

Edit Execute Edit MIPS program

mips1.asm

```
1 .data
2     str .asciiz "welcome to MIPS world:)"      #"str" is a lable ?
3
4 .text
5 main:                                          #"main" is a lable
6     li $v0,4
7     la $t0,str                                # which register will be used for syscall, $to or $?
8     syscall                                   #invoke a system service "Print String" with service index 4
9
10    li $v0,10
11    syscall                                   #invoke a system service "EXIT" with service index 10
12
```

Line: 12 Column: 1 Show Line Numbers

Mars Messages Run I/O I/O window

show Mars Messages

Clear

Assemble: assembling D:\计算机组成原理\2020s\mips1.asm

Error in D:\计算机组成原理\2020s\mips1.asm line 2 column 10: ".asciiz" is not a valid integer constant or label

Error in D:\计算机组成原理\2020s\mips1.asm line 2 column 21: ""welcome to MIPS world:)" is not a valid integer constant

Error in D:\计算机组成原理\2020s\mips1.asm line 2 column 2: Symbol "str" not found in symbol table.

Assemble: operation completed with errors.

Coproc 1 Coproc 0

Registers

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

MARS is a **lightweight** interactive development environment (IDE) for programming in **MIPS** assembly language, intended for educational-level use with Patterson and Hennessy's Computer Organization and Design.

MARS requires Java J2SE 1.5 (or later) SDK installed on your computer.

Mars(2)

run one step at a time
run the program
clear memory and registers

File Edit **Run** Settings Tools Help

run the program

run one step at a time

clear memory and registers

Registers

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x24020004	addiu \$2,\$0,0x0...	6: li \$v0,4
	0x00400004	0x3e011001	lui \$1,0x0001001	7: la \$a0,str
	0x00400008	0x34240000	ori \$4,\$1,0x000...	
	0x0040000c	0x0000000c	syscall	8: syscall
	0x00400010	0x2402000a	addiu \$2,\$0,0x0...	10: li \$v0,10
	0x00400014	0x0000000c	syscall	11: syscall

Data Segment

Address	Value (+...	Value (+...	Value (+...	Value (+...	Value (+...	Value (+...	Value (+...	Value (+...
0x100...	0x6c6...	0x6f7...	0x006...	0x000...	0x000...	0x000...	0x000...	0x000...
0x100...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...
0x100...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...
0x100...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...
0x100...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...
0x100...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...	0x000...

Mars Messages

Run I/O

Assemble state info

Assemble: assembling D:\计算机组成原理\2020s\LAB\lab1\mips1.asm

Assemble: operation completed successfully.

Clear

Run Settings Tools Help

Assemble F3

Go F5

Run Settings Tools Help

Assemble F3

Go F5

Mars Messages

Run I/O

Assemble: assembling D:\计算机组成原理\2020s\LAB\lab1\mips1.asm

Assemble: operation completed successfully.

Go: running mips1.asm

Go: execution completed successfully.

Clear

Mars Messages

Run I/O

Hello, world

— program is finished running —

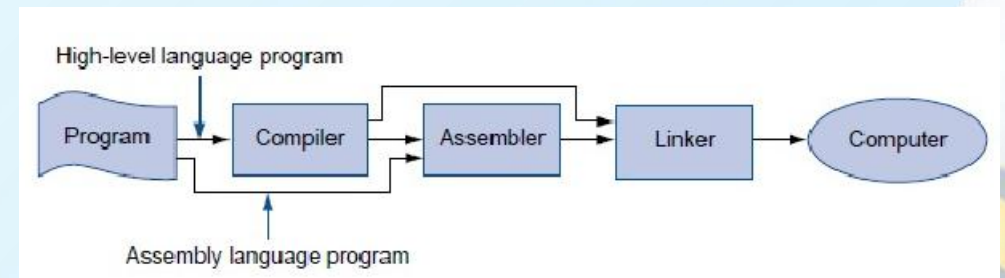
Clear



Assembly Language(汇编语言) Overview

- Assembly language
 - The main body of assembly language (汇编语言) is assembly instruction (汇编指令) .
 - Assembly instructions(汇编指令) and Machine instructions(机器指令) differ in instruction expression. **Assembly instruction is a human-friendly writing format of machine instruction.** Assembly language is the mnemonic of machine language. For example
 - machine instruction “1000100111011000” means that send the contents of register BX to AX.
 - The assembly instructions are written as “mov ax, bx”. Such writing is similar to human language and easy to read and remember.
 - Feature: programs written in assembly language are inherently machine-specific and must be totally rewritten to run on another computer architecture

- CISC (intel x86) vs **RISC** (ARM, **MIPS**, RISC-V)





Assembly Program Structure

data declarations + program code

- **part1: Data Declarations**
 - placed in section of program identified with assembler directive(汇编说明/汇编器指示符): **.data**
 - **declare** variable names used in program; storage allocated in main **memory** (RAM)
- **part2: Program Code**
 - placed in section of text identified with assembler directive: **.text**
 - contains program code (instructions)
 - starting point of code e.g. execution given label **main**:
 - ending point of code should use exit system call (see below in “System Calls” part)
- **part3: Comments (suggested)**
 - anything following # on a line
This stuff would be considered as comment

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

        .data        # variable declarations follow this line
                # ...

        .text        # instructions follow this line

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

# End of program, leave a blank line afterwards to make SPIM happy
```


System Calls

- **System Calls**
 - **SPIM** provides a small set of **operating-system-like services** through the system call (**syscall**) instruction.
 - To request a service, 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).

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

A demo

The QtSpim MIPS simulator interface is shown. The 'Registers' window on the left lists 32 registers. The 'Text' window in the center displays assembly code for a 'User Text Segment' and a 'Kernel Text Segment'. The 'Console' window at the bottom right shows the output 'hello Internet'. The status bar at the bottom indicates 'Memory and registers cleared'.

Registers:

- PC = 400038
- EPC = 0
- Cause = 0
- BadVAddr = 0
- Status = 3000fff10
- HI = 0
- LO = 0
- R0 [r0] = 0
- R1 [at] = 10010000
- R2 [v0] = a
- R3 [v1] = 0
- R4 [a0] = 10010000
- R5 [a1] = 7ffff53c
- R6 [a2] = 7ffff544
- R7 [a3] = 0
- R8 [t0] = 0
- R9 [t1] = 0
- R10 [t2] = 0
- R11 [t3] = 0
- R12 [t4] = 0
- R13 [t5] = 0
- R14 [t6] = 0
- R15 [t7] = 0
- R16 [s0] = 0
- R17 [s1] = 0
- R18 [s2] = 0
- R19 [s3] = 0
- R20 [s4] = 0
- R21 [s5] = 0

Text Segment:

User Text Segment [00400000]..[00400000]

```
[00400000] lw $4, 0($29)
[00400004] addiu $5, $29, 4
[00400008] addiu $6, $5, 4
[0040000c] sll $2, $4, 2
[00400010] addu $6, $6, $2
[00400014] jal 0x00400024 [main]
[00400018] nop
[0040001c] ori $2, $0, 10
[00400020] syscall
[00400024] lui $1, 4097 [str]
[00400028] ori $4, $1, 0 [str]
[0040002c] ori $2, $0, 4
[00400030] syscall
[00400034] ori $2, $0, 10
[00400038] syscall
```

Kernel Text Segment [80000000]..[80010000]

```
[80000180] addu $27, $0, $1
[80000184] lui $1, -28672
[80000188] sw $2, 512($1)
[8000018c] lui $1, -28672
[80000190] sw $4, 516($1)
[80000194] mfc0 $26, $13
[80000198] srl $4, $26, 2
[8000019c] andi $4, $4, 31
[800001a0] ori $2, $0, 4
[800001a4] lui $4, -28672 [__m1_]
[800001a8] syscall
[800001ac] ori $2, $0, 1
[800001b0] srl $4, $26, 2
```

Console:

```
hello Internet
```

```
# text segment
.text
.globl main

main:
    la $a0, str          # execution starts here
    li $v0, 4            # put string address into a0
    syscall              # system call to print a string
    li $v0, 10
    syscall              # system call to exit

#data segment
.data

str: .asciiz "hello Internet\n"
```

Update the demo by deleting the line “.globl main”, rerun it on the Qtspim, what will happen?

Run the updated demo on the Mars, what will happen?



Practice & thinking

- 1. Install QtSpim/Mars on your PC and refer to the 'help' page to find:
 - Is the function of '**syscall**' in Qtspim the same with Mars?
 - What's the name and ID of the **registers** used while invoke the "**print string**" **syscall**?
- 2. **Make an assembly file, load and run** it by QtSpim/Mars:
 - The assembly code should **get** your ID and name, and **print** them out
 - Is there any **object file** or **executable file** generated while doing the simulation?
 - **Where** is your ID and name **stored** while the assembly code get them? **register** or **memory**?
 - (Optional) What's the **Machine code** of the MIPS instruction "**li \$v0,10**"
- 3. (Optional) Find an assembler for MIPS from the Internet, use it to assemble the code of question 2 to generate an executable file, Can the executable file run on your computer and explain the reason.