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

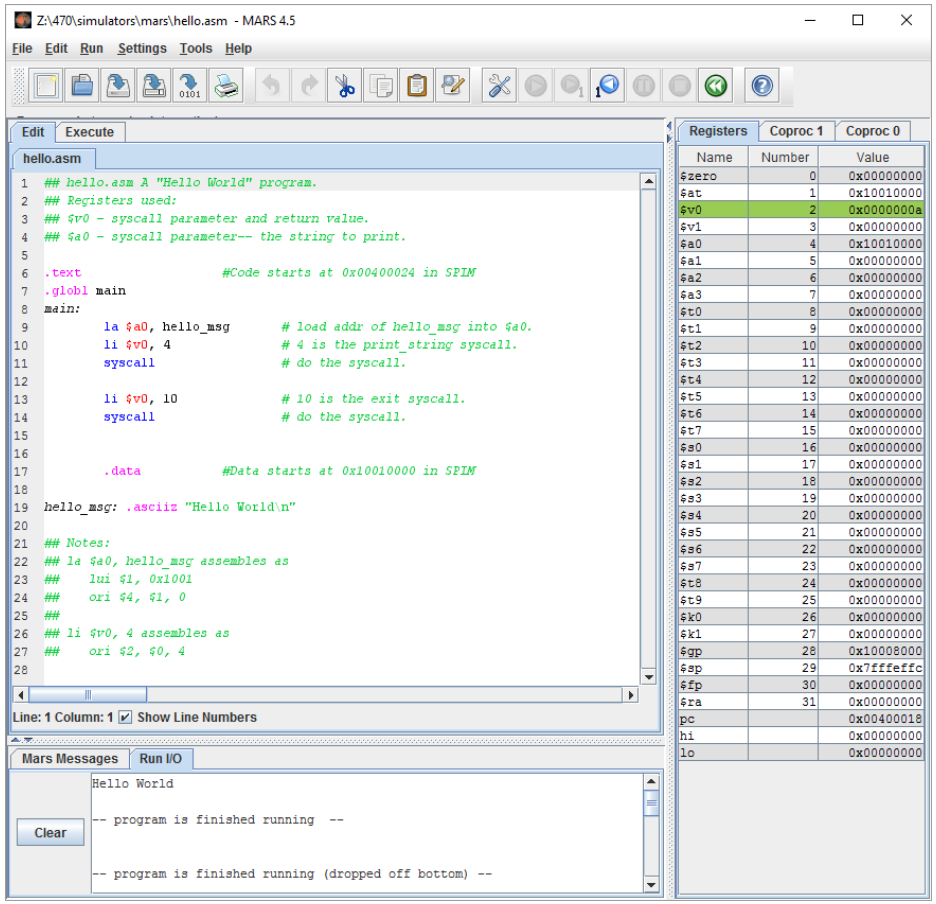
- User data, stack, kernel data

Messages window

- messages (include error messages)

Console interface

Input/output



Simulator Settings

Z:\470\simulators\mars\hello.asm - MARS 4.5

File Edit Run Settings Tools Help

- Show Labels Window (symbol table)
- Program arguments provided to MIPS program
- Popup dialog for input syscalls (5,6,7,8,12)
- Addresses displayed in hexadecimal
- Values displayed in hexadecimal
- Assemble file upon opening
- Assemble all files in directory
- Assembler warnings are considered errors
- Initialize Program Counter to global 'main' if defined
- Permit extended (pseudo) instructions and formats
- Delayed branching
- Self-modifying code
- Editor...
- Highlighting...
- Exception Handler...
- Memory Configuration...

Edit Execute

hello.asm

```
## hello.
## Regist
## $v0 -
## $a0 -

.text
.globl ma
main:
        l                                   r into $a0.
        l                                   syscall.
        s

        l                                   ..
        s


        .data           #Data starts at 0x10010000 in SPIM

hello_msg: .asciiz "Hello World\n"

## Notes:
## la $a0, hello_msg assembles as
##     lui $1, 0x1001
##     ori $4, $1, 0
##
## li $v0, 4 assembles as
##     ori $2, $0, 4
```

Line: 1 Column: 1 Show Line Numbers

Mars Messages Run I/O

```
Hello World

-- program is finished running  --


-- program is finished running (dropped off bottom) --
```

Clear

Registers Coproc 1 Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x10010000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x10010000
$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	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400018
hi		0x00000000
lo		0x00000000

Execute Program

QtSpim

File Simulator Registers Text Segment Data Segment Window Help

FP Regs | Int Regs [16] | Data | Text

Int Regs [16]

```
PC       = 40002c
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 4
R3  [v1] = 0
R4  [a0] = 1
R5  [a1] = 7ffff638
R6  [a2] = 7ffff640
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] = 5
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 400018
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
```

your assembly code starts

register change resulting from previous instruction

Text

```
                                    User Text Segment [00400000]..[00440000]
[00400000] 8fa40000  lw $4, 0($29)              ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004  addiu $5, $29, 4           ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004  addiu $6, $5, 4            ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080  sll $2, $4, 2              ; 186: sll $v0 $a0 2
[00400010] 00c23021  addu $6, $6, $2            ; 187: addu $a2 $a2 $v0
[00400014] 0c100009  jal 0x00400024 [main]      ; 188: jal main
[00400018] 00000000  nop                        ; 189: nop
[0040001c] 3402000a  ori $2, $0, 10             ; 191: li $v0 10
[00400020] 0000000c  syscall                    ; 192: syscall # syscall 10 (exit)
[00400024] 001fb821  addu $23, $0, $31          ; 3: addu $s7, $0, $ra #save the return address in a global register
[00400028] 20110005  addi $17, $0, 5            ; 5: addi $s1, $0, 5 #g = 5
[0040002c] 2012ffec  addi $18, $0, -20          ; 6: addi $s2, $0, -20 #h = -20
[00400030] 2013000d  addi $19, $0, 13           ; 7: addi $s3, $0, 13 #i = 13
[00400034] 20140003  addi $20, $0, 3            ; 8: addi $s4, $0, 3 #j = 3
[00400038] 02324020  add $8, $17, $18           ; 9: add $t0, $s1, $s2 #register $t0 contains g + h
[0040003c] 02744820  add $9, $19, $20           ; 10: add $t1, $s3, $s4 #register $t1 contains i + j
[00400040] 01098022  sub $16, $8, $9            ; 11: sub $s0, $t0, $t1 #f = (g + h) - (i + j)
[00400044] 34020004  ori $2, $0, 4              ; 17: li $v0, 4 #print_str (system call 4)
[00400048] 3c041001  lui $4, 4097 [message]     ; 18: la $a0, message # takes the address of string as an argument
[0040004c] 0000000c  syscall                    ; 19: syscall
[00400050] 34020001  ori $2, $0, 1              ; 21: li $v0, 1 #print_int (system call 1)
[00400054] 00102020  add $4, $0, $16            ; 22: add $a0, $0, $s0 #put value to print in $a0
[00400058] 0000000c  syscall                    ; 23: syscall
[0040005c] 0017f821  addu $31, $0, $23          ; 26: addu $ra, $0, $s7 #restore the return address
[00400060] 03e00008  jr $31                     ; 27: jr $ra #return to the main program
[00400064] 00000020  add $0, $0, $0             ; 28: add $0, $0, $0 #nop

                                    Kernel Text Segment [80000000]..[80010000]
[80000180] 0001d821  addu $27, $0, $1           ; 90: move $k1 $at # Save $at
[80000184] 3c019000  lui $1, -28672             ; 92: sw $v0 s1 # Not re-entrant and we can't trust $sp
[80000188] ac220200  sw $2, 512($1)
[8000018c] 3c019000  lui $1, -28672             ; 93: sw $a0 s2 # But we need to use these registers
[80000190] ac240204  sw $4, 516($1)
```

current instruction

SPIM Version 9.1.4 of September 4, 2011
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.

Single Step

Program Text and Data segments

Z:\470\simulators\mars\hello.asm - MARS 4.5

File Edit Run Settings Tools Help

Edit | Execute

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c011001	lui $1,0x00001001	9: la $a0, hello_msg # ...
	0x00400004	0x34240000	ori $4,$1,0x00000000	
	0x00400008	0x24020004	addiu $2,$0,0x00000004	10: li $v0, 4 # ...
	0x0040000c	0x0000000c	syscall	11: syscall # ...
	0x00400010	0x2402000a	addiu $2,$0,0x0000000a	13: li $v0, 10 # ...
	0x00400014	0x0000000c	syscall	14: syscall # ...

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x6c6c6548	0x6f57206f	0x0a646c72	0x00000000	0x00000000	0x00000000	0x00000000	0x0000...
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0000...
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0000...
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0000...
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0000...
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0000...
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0000...

0x10010000 (.data) ☑ Hexadecimal Addresses ☑ Hexadecimal Values ☐ ASCII

Mars Messages | Run I/O

```
Hello World
```

Clear

Coproc 1 | Coproc 0

Registers

Name	Numb...	Value
$zero	0	0x00000000
$at	1	0x10010000
$v0	2	0x00000004
$v1	3	0x00000000
$a0	4	0x10010000
$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	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400010
hi		0x00000000
lo		0x00000000

Set a break point

Z:\470\simulators\mars\hello.asm - MARS 4.5

File Edit Run Settings Tools Help

Edit | Execute

Text Segment

Bkpt	Address	Code	Basic	Source
☐	0x00400000	0x3c011001	lui $1,0x00001001	9: la $a0, hello_msg # ...
☐	0x00400004	0x34240000	ori $4,$1,0x00000000	
☐	0x00400008	0x24020004	addiu $2,$0,0x00000004	10: li $v0, 4 # ...
☑	0x0040000c	0x0000000c	syscall	11: syscall # ...
☐	0x00400010	0x2402000a	addiu $2,$0,0x0000000a	13: li $v0, 10 # ...
☑	0x00400014	0x0000000c	syscall	14: syscall # ...

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x6c6c6548	0x6f57206f	0x0a646c72	0x00000000	0x00000000	0x00000000	0x00000000	0x0000...
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0000...
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0000...
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0000...
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0000...
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0000...
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x0000...

0x10010000 (.data) ☑ Hexadecimal Addresses ☑ Hexadecimal Values ☐ ASCII

Mars Messages | Run I/O

Clear

```
Go: running hello.asm

Go: execution paused at breakpoint: hello.asm
```

Coproc 1 | Coproc 0

Registers

Name	Numb...	Value
$zero	0	0x00000000
$at	1	0x10010000
$v0	2	0x00000004
$v1	3	0x00000000
$a0	4	0x10010000
$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	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x0040000c
hi		0x00000000
lo		0x00000000

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

```
        .text           #code section
        .globl main     #starting point: must be global
main:
        # user program code
```

```
        .data           #data section
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)

.ascii, .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                          # text section
.globl main                    # call main by SPIM
main:   la $t0, value          # load address 'value' into $t0
   lw $t1, 0($t0)              # load word 0(value) into $t1
   lw $t2, 4($t0)              # load word 4(value) into $t2
   add $t3, $t1, $t2           # add two numbers into $t3
   sw $t3, 8($t0)              # store word $t3 into 8($t0)

   li $v0, 10                  # 10 exit syscall.
   syscall                     #   do the syscall.
.data                          # data section
value:  .word 10, 20, 0        # data for addition
#this ends the program.
```

Common Pseudoinstructions

Pseudoinstruction	Translation
mov $rt, $rs	add $rt, $rs, $0
li $rs, small	addi $rs, $0, small
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

Service	Code($v0)	Arg	Result
Print_int	1	$a0 (int to print)	
Print_string	4	$a0 (addr of string)	
Read_int	5		Int in $v0
Read_string	8	$a0: addr, $a1:bytes	

SPIM Program Example II

```
#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

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