CSE 360-Computer Architecture

Lecture-SPIM

Dr. Shamim Akhter

Computer Science and Engineering

EAST WEST UNIVERSITY

Overview of SPIM – the MIPS simulator

Introduction

- What is SPIM?
  - a *simulator* that runs *assembly programs* for *MIPS* R2000/R3000 Reduce Instruction Set Computing-RISC computers
- What does SPIM 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 *OS-like services*, like simple I/O

MIPS & SPIM Resources

- " Computer Organization & Design: The Hardware/Software Interface", by Patterson and Hennessy: Chapter 3 and Appendix A.9-10

- Other on-line resources

PCSpim Installation

- Windows Installation
  - download the file *http://www.cs.wisc.edu/~larus/SPIM/pcspim.zip* and save it on your machine.
  - unzip the file
  - run the *setup.exe* program

Using SPIM

- Loading source file
  - Use *File* -> *Open* menu
- Simulation
  - *Simulator* -> *Settings…* :
    - in the *Display* section check only the first two items *Save window positions* and *General registers in hexadecimal*
    - in the *Execution* section check only *Allow pseudo instruction*
  - *Simulator* -> *Set Value…* : to load PC with address of first instruction
    - enter *Address or Register Name* as "PC" and enter *Value* as "0x00400000"
      - reason: the *text area* of memory, where programs are stored, starts here
  - *Simulator* -> *Go* : run loaded program
    - Click the *OK* button in the *Run Parameters* pop-up window if the *StartingAddress:* value is "0x00400000"
  - *Simulator* -> *Break* : stop execution
  - *Simulator* -> *Clear Registers* and *Reinitialize* : clean-up before new run

Important!!

Using SPIM

- *Simulator -> Reload* : load file again after editing
- *Simulator -> Single Step* or *Multiple Step* : stepping to debug
- *Simulator -> Breakpoints* : set breakpoints

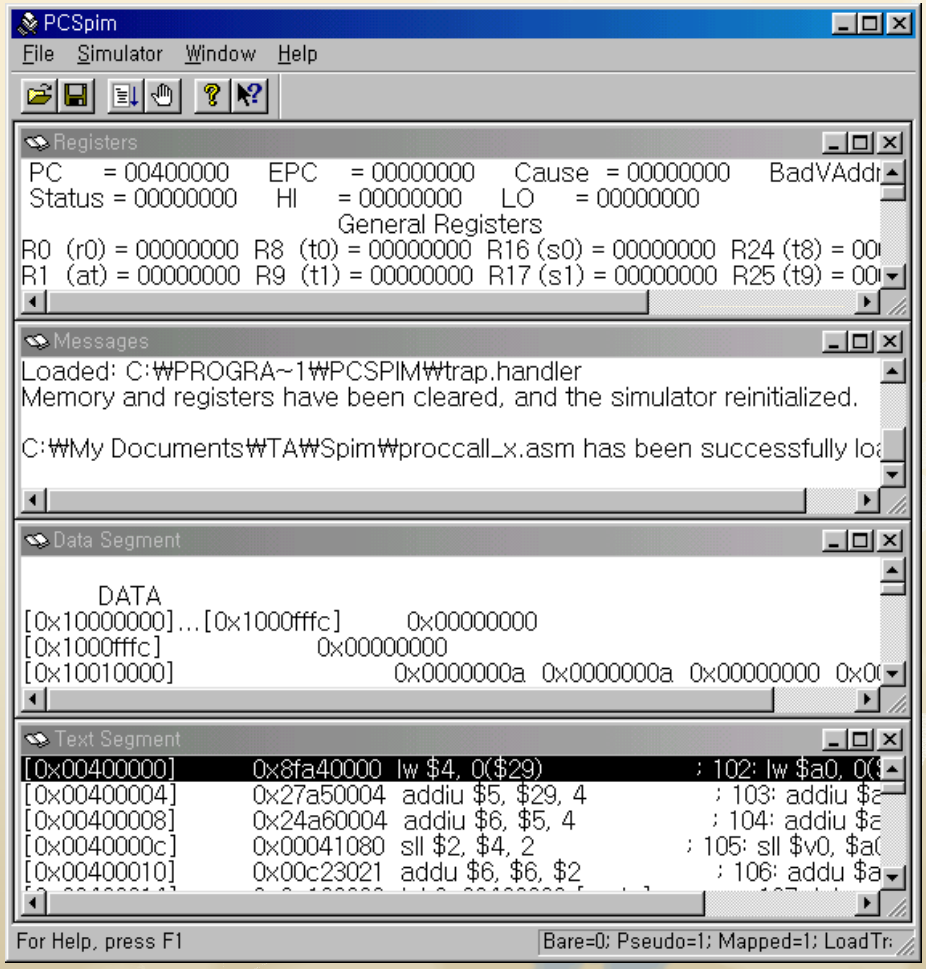
- Notes:
  - text segment window of SPIM shows assembly and corresponding machine code
    - *pseudo-instructions each expand to more than one machine instruction*
  - if *Load trap file* is checked in *Simulator -> Settings…* then text segment shows additional trap-handling code
  - if *Delayed Branches* is checked in *Simulator -> Settings…* then `statementx` will execute before control jumps to `L1` in following code – to avoid insert `nop` before `statementx`:

```
    jal L1
    statementx
    …
L1: …
```

PCSpim Windows Interface

- Registers window
  - shows the values of all registers in the MIPS CPU and FPU
- Text segment window
  - shows assembly instructions & corresponding machine code
- Data segment window
  - shows the data loaded into the program's memory and the data of the program's stack
- Messages window
  - shows PCSpim messages

Separate console window appears for I/O



SPIM 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
```

MIPS Assembly Code Layout

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

MIPS Assembler Directives

- Top-level Directives:
  - **.text**
    - indicates that following items are stored in the user text segment, typically instructions
  - **.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 Example Program: add2numbersProg2.asm

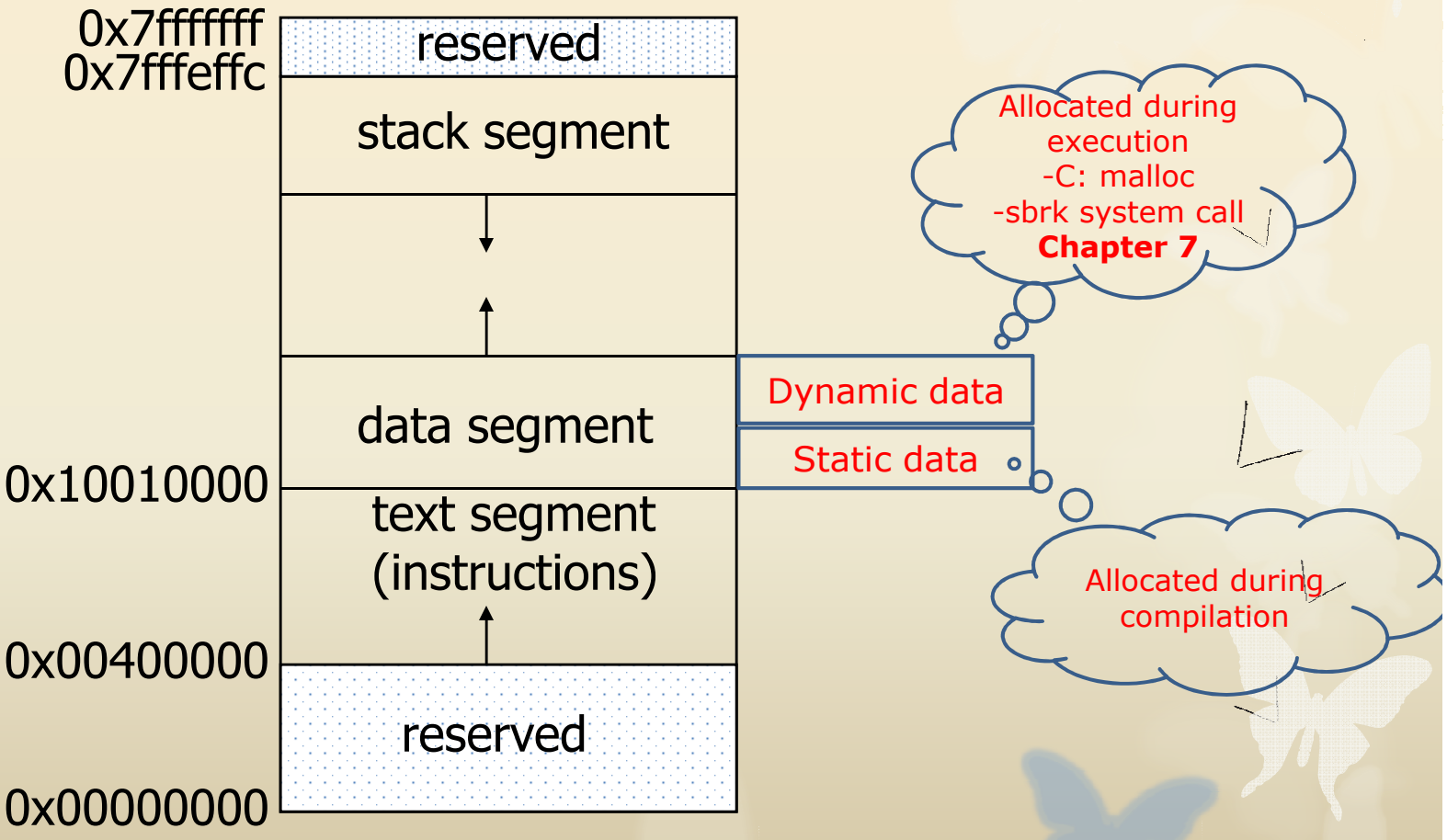
```
# Program adds 10 and 20

        .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)

        .data                       # data section
value: .word 10, 20, 0          # load data integers. Default data
                                # start address 0x10010000(= value)
```

MIPS Memory Usage as viewed in SPIM



MIPS Assembler Directives

- **Common Data Definitions:**
  - **.word** w1, …, wn
    - store n 32-bit quantities in successive memory words
  - **.half** h1, …, hn
    - store n 16-bit quantities in successive memory halfwords
  - **.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
      - strings are represented in double-quotes “str”
      - special characters, eg. \n, \t, follow C convention
  - **.asciiz** str
    - store the string in memory and null-terminate it

MIPS Assembler Directives

- **Common Data Definitions:**

Chapter 4: Computer Arithmetic

- **.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

MIPS: Software Conventions for Registers

0	zero	constant 0
1	at	reserved for assembler
2	v0	results from callee
3	v1	returned to caller
4	a0	arguments to callee
5	a1	from caller: caller saves
6	a2	
7	a3	
8	t0	temporary: caller saves
. . .		(callee can clobber)
15	t7	
16	s0	callee saves
. . .		(caller can clobber)
23	s7	
24	t8	temporary (cont'd)
25	t9	
26	k0	reserved for OS kernel
27	k1	
28	gp	pointer to global area
29	sp	stack pointer
30	fp	frame pointer
31	ra	return Address (HW): caller saves

SPIM System Calls

- System Calls (syscall)
  - OS-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

SPIM System Call Codes

Service	Code (put in $v0)	Arguments	Result
print_int	1	$a0=integer	
print_float	2	$f12=float	
print_double	3	$f12=double	
print_string	4	$a0=addr. of string	
read_int	5		int 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	addr in $v0
exit	10		

SPIM Example Program: systemCalls.asm

```
## 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 = "
```

system call code for read_int

result returned by call

system call code for print_string

argument to print_string call

system call code for print_int

argument to print_int call

system call code for exit