


## Assemblers Linkers and SPIM

---

- See Appendix A (on CD ROM in old 3<sup>rd</sup> edition)
- Installation
- Machine Language
- Symbolic Instructions
  - Mnemonic operations and register numbers
- Symbolic Languages
  - Register names, mnemonic labels, and pseudos
- Symbolic Languages with Comments
- Symbolic Languages with Mnemonic Registers

Fall, 2008
Charlie Shub - CS216 - Spim - Appendix A
Page 1




## Why / Why Not Use Them

---

- Use when you need
  - Speed
  - Size
  - Access to features
- Contra-indicators
  - Machine Specific
  - Programs Longer
  - Programs more error prone

Fall, 2008
Charlie Shub - CS216 - Spim - Appendix A
Page 2




## Major Features

---

- Labels (names for memory cells)
  - Local (invisible outside)
  - Global (visible outside)
- Macros
  - Define a sequence
  - Invoke Definition
  - Assembler expands
- Pseudo Operations

Fall, 2008
Charlie Shub - CS216 - Spim - Appendix A
Page 3




## Pseudo Operations

---

- Data Generating
 

.byte	values
.word	values
.ascii	string
- Assembler Directives
  - .data
  - .text
- Pseudo-instructions
  - Later
- Complete List
  - Found on CD Rom

Fall, 2008
Charlie Shub - CS216 - Spim - Appendix A
Page 4




## SPIM

---

- Assembler and Simulator
  - Install it
  - Write Code For it
  - Load Code
  - Run Code
- Doing I/O
  - Use System Calls
  - Listed on CD Rom

Fall, 2008
Charlie Shub - CS216 - Spim - Appendix A
Page 5




## Installation From CD (3<sup>rd</sup> Edition)

---

- Insert the CD that comes with your text into your CD drive.
- If the autorun sequence on the CD does not start up your browser, open the file **index.html** in the top level directory of your CD.
- Select **Software** in the left panel.
- Select the **Spim, PCSpim and xspim** link in the right panel.
- Select the **Click Here** link under **Windows**
- Double click on the **Setup.exe** entry.
- It may be convenient for you to create a shortcut to **pcspim.exe** and then move the shortcut to your desktop.

Fall, 2008
Charlie Shub - CS216 - Spim - Appendix A
Page 6




## Alternate SPIM Versions

---

- There are links on the class page to
  - The original PC Spim from the University of Wisconsin
  - A copy of the Zipfile on the CD
  - An older version from the University of Texas that has resizable windows.

Fall, 2008
Charlie Shub - CS216 - Spim - Appendix A
Page 7




## Configuring SPIM

---

- Run **PCSPIM**.
- Select **Simulator** and then **Settings**.
- The following settings are recommended
  - Save window positions
  - General registers in hexadecimal
  - Allow pseudo instructions
  - Load exception file
- You may want to deselect the **Mapped I/O** box so there is *not* a check mark in that box.
- The default "exception file" is located in the folder where you put all the extracted files.

Fall, 2008
Charlie Shub - CS216 - Spim - Appendix A
Page 8




## Highlights from CD Rom (3<sup>rd</sup> edition)

---

- Assembler Directives (pseudo operations)
- System Calls
- Pseudo Instructions
- Procedure Call Register Use Convention
- Object File Format

Fall, 2008
Charlie Shub - CS216 - Spim - Appendix A
Page 9




## Assembler Directives

---

<pre>.align n .ascii str .asciiz str .byte b1,...,bn .data .extern sym .globl sym .space n .text .word</pre>	<pre>Align the next datum on a 2<sup>n</sup> byte boundary. .align 2 aligns the next value on a word boundary. Store the string str in memory Store the string str in memory and null-terminate it. Store the n values in successive bytes of memory. Subsequent items are stored in the data segment. Declare that sym is defined externally. Declare that label sym is global and can be referenced from other files. (using this adds sym to symbol table. this is useful for breakpoints) Allocate n bytes of space in the current data segment Subsequent items are put in the user text segment. like byte, but stores 32 bit words.</pre>
--	--

Fall, 2008
Charlie Shub - CS216 - Spim - Appendix A
Page 10




## Using System Calls

---

- Load the registers with the call arguments (see next pages)
- Execute a `syscall` instruction
- Save any return values
- For example
 

```
li    $v0,1    #code for print_int
li    $a0,5    #integer to print
syscall      #print it
```

Fall, 2008
Charlie Shub - CS216 - Spim - Appendix A
Page 11




## System Calls

---

<pre>print_int print_float print_string read_int read_float read_string Exit print_char read_char</pre>	<pre>01  \$a0 = integer 02  \$f12 = float 04  \$a0 = string 05  returns integer (in \$v0) 06  returns float (in \$f0) 08  \$a0 = buffer, \$a1 = length 10 11  \$a0 = char 12  char (in \$a0)</pre>
---	--

Fall, 2008
Charlie Shub - CS216 - Spim - Appendix A
Page 12



## Variables in SPIM


---

- In procedure languages all variables and arrays are declared.
- In SPIM that must happen too
- int a, b, c in a high level language would be done as

```

c:  .space 4 # allocate 4 bytes
b:  .word 0 # allocate 4 bytes and
      # populate them with zero
a:  .word 0 # populate a word with zero
    
```

Fall, 2008 Charlie Shub - CS216 - Spim - Appendix A Page 13




## Arrays in SPIM

---

- In procedure languages all variables and arrays are declared.
- In SPIM that must happen too
- int ray[100];  
Implemented as  
ray: .space 400 #100 words is 400 bytes
- char str [128]  
Implemented as  
str: .space 128 #128 bytes

Fall, 2008 Charlie Shub - CS216 - Spim - Appendix A Page 14




## File I/O System Calls

---

```

open 13  $a0 = filename (string), $a1 =
          flags, $a2 = mode
          returns file descriptor (in $a0)
read  14  $a0 = file descriptor, $a1 =
          buffer, $a2 = length
          returns num chars read (in $a0 )
write 15  $a0 = file descriptor, $a1 = buffer,
          $a2 = length
          returns num chars written (in $a0 )
close 16  $a0 = file descriptor
exit2 17  $a0 = result
    
```

Fall, 2008 Charlie Shub - CS216 - Spim - Appendix A Page 15




## Pseudo-Instructions

---

```

li  rdest,imm pseudoinstruction
la  rdest,address pseudoinstruction
move rdest,rsrc pseudoinstruction
abs  rdest,rsrc pseudoinstruction
neg  rdest,rsrc pseudoinstruction
negu rdest,rsrc pseudoinstruction
not  rdest,rsrc pseudoinstruction
rol  rdest,rsrc1,rsrc2 pseudoinstruction
ror  rdest,rsrc1,rsrc2 pseudoinstruction
    
```

Fall, 2008 Charlie Shub - CS216 - Spim - Appendix A Page 16




## Set Pseudo-Instructions

---

- seq rdest,rsrc1,rsrc2 pseudoinstruction
- sge rdest,rsrc1,rsrc2 pseudoinstruction
- sgeu rdest,rsrc1,rsrc2 pseudoinstruction
- sgt rdest,rsrc1,rsrc2 pseudoinstruction
- sgtu rdest,rsrc1,rsrc2 pseudoinstruction
- sle rdest,rsrc1,rsrc2 pseudoinstruction
- sleu rdest,rsrc1,rsrc2 pseudoinstruction
- sne rdest,rsrc1,rsrc2 pseudoinstruction

Fall, 2008 Charlie Shub - CS216 - Spim - Appendix A Page 17




## Branch Pseudo-Instructions

---

- b label pseudoinstruction
- beqz rsrc,label pseudoinstruction
- bge rsrc1,rsrc2,label pseudoinstruction
- bgeu rsrc1,rsrc2,label pseudoinstruction
- bgt rsrc1,src2,label pseudoinstruction
- bgtu rsrc1,src2,label pseudoinstruction
- ble rsrc1,src2,label pseudoinstruction
- bleu rsrc1,src2,label pseudoinstruction
- blt rsrc1,rsrc2,label pseudoinstruction
- bltu rsrc1,rsrc2,label pseudoinstruction
- bnez rsrc,label pseudoinstruction

Fall, 2008 Charlie Shub - CS216 - Spim - Appendix A Page 18




## Arithmetic Pseudo-Instructions

---

```

mul  rdest,rsrc1,src2 pseudoinstruction
div  rdest,rsrc1,src2 pseudoinstruction
divu rdest,rsrc1,src2 pseudoinstruction
rem  rdest,rsrc1,src2 pseudoinstruction
remu rdest,rsrc1,src2 pseudoinstruction
mulo rdest,rsrc1,src2 pseudoinstruction
mulou rdest,rsrc1,src2 pseudoinstruction
    
```

Fall, 2008 Charlie Shub - CS216 - Spim - Appendix A Page 19




## Procedure Call Conventions Part 1

---

- The MIPS CPU contains 32 general-purpose registers that are numbered 0–31.
- Register \$0 always contains the hardwired value 0.
- Registers \$at (1), \$k0 (26), and \$k1 (27) are reserved for the assembler and operating system and should not be used.
- Registers \$a0 –a3 (4–7) are used to pass the first four arguments to routines (remaining arguments are passed on the stack).
- Registers \$v0 and \$v1 (2, 3) are used to return values from functions.

Fall, 2008 Charlie Shub - CS216 - Spim - Appendix A Page 20




## Procedure Call Conventions Part 2

---

- Registers \$t0 –t9 (8–15, 24, 25) are **caller-saved registers** that are used to hold temporary quantities that need not be preserved across calls (see Section 2.7 in Chapter 2).
- Registers \$s0 –s7 (16–23) are **callee-saved registers** that hold long-lived values that should be preserved across calls.
- Register \$gp (28) is a global pointer that points to the middle of a 64K block of memory in the static data segment.
- Register \$sp (29) is the stack pointer, which points to the last location on the stack.
- Register \$fp (30) is the frame pointer.
- Register \$ra (31) is the return address register. The jal instruction writes register \$ra during a procedure call.

Fall, 2008 Charlie Shub - CS216 - Spim - Appendix A Page 21




## Object File Format Part 1

---

- The *object file header* describes the size and position of the other pieces of the file.
- The **text segment** contains the machine language code for routines in the source file. These routines may be unexecutable because of unresolved references.
- The **data segment** contains a binary representation of the data in the sourcefile. The data also may be incomplete because of unresolved references to labels in other files.
- The **relocation information** identifies instructions and data words that depend on **absolute addresses**. These references must change if portions of the program are moved in memory.

Fall, 2008 Charlie Shub - CS216 - Spim - Appendix A Page 22



## Object File Format Part 2

---

- The *symbol table* associates addresses with external labels in the source file and lists unresolved references.
- The *debugging information* contains a concise description of the way in which the program was compiled, so a debugger can find which instruction addresses correspond to lines in a source file and print the data structures in readable form.
- The assembler produces an object file that contains a binary representation of the program and data and additional information to help link pieces of a program together.

Fall, 2008 Charlie Shub - CS216 - Spim - Appendix A Page 23