MIPS assembler directives

(From Computer Organization and Design - The Hardware/Software Interface by Dave Patterson and John Hennessy, 3rd edition) SPIM supports a subset of the assembler directives provided by the actual MIPS assembler:

Syntax	Description
`.align n`	Align the next datum on a 2^n byte boundary. For example, `.align 2` aligns the next value on a word boundary. `.align 0` turns off automatic alignment of `.half`, `.word`, `.float`, and `.double` directives until the next `.data` or `.kdata` directive.
`.ascii str`	Store the string in memory, but do not null-terminate it.
`.asciiz str`	Store the string in memory and null-terminate it.
`.byte b1,..., bn`	Store the n values in successive bytes of memory.
`.data <addr>`	The following data items should be stored in the data segment. If the optional argument addr is present, the items are stored beginning at address addr.
`.double d1,...,dn`	Store the n floating point double precision numbers in successive memory locations.
`.extern sym size`	Declare that the datum stored at sym is size bytes large and is a global symbol. This directive enables the assembler to store the datum in a portion of the data segment that is efficiently accessed via register `$gp`.
`.float f1,..., fn`	Store the n floating point single precision numbers in successive memory locations.
`.globl sym`	Declare that symbol sym is global and can be referenced from other files.
`.half h1,..., hn`	Store the n 16-bit quantities in successive memory halfwords.
`.kdata <addr>`	The following data items should be stored in the kernel data segment. If the optional argument addr is present, the items are stored beginning at address addr.
`.ktext <addr>`	The next items are put in the kernel text segment. In SPIM, these items may only be instructions or words (see the `.word` directive below). If the optional argument addr is present, the items are stored beginning at address addr.
`.space n`	Allocate n bytes of space in the current segment (which must be the data segment in SPIM).
`.text <addr>`	The next items are put in the user text segment. In SPIM, these items may only be instructions or words (see the .word directive below). If the optional argument addr is present, the items are stored beginning at address addr.
`.word w1,..., wn`	Store the n 32-bit quantities in successive memory words. SPIM does not distinguish various parts of the data segment (`.data`, `.rdata` and `.sdata`).

Control Flow Instructions in MIPS

Type	Instruction	Description
Unconditional Jump	`j Label`	This instruction jumps unconditionally to the instruction followed by the label `Label`. It is equivalent to the `goto` statement in C.

Conditional Branch on Equality	`beq rs, rt, Label`	This instruction branches to label `Label`, if registers `rs` and `rt` have the same values. In Register Transfer Notation (RTN), it means:
		`if ( R[rs] == R[rt] )`
		`  goto Label;`
		
		If the registers have different values, the processor proceeds to the next instruction.

Conditional Branch on Inequality	`bne rs, rt, Label`	This instruction branches to label `Label`, if registers `rs` and `rt` have the same values.

Set Register Based on Relation	`slt rd, rs, rt`	This instruction sets register `rd` to `1` if `R[rs]<R[rt]`. Otherwise it sets `rd` to `0`. In RTN, it means:
		`if ( R[rs] < R[rt] )`
		`  R[rd] = 1;`
		`else`
		`  R[rd] = 0;`
		
		There is an immediate version of the `slt` instruction in which the 3rd argument is a 16-bit signed integer. "`slti rd, rs, 0x0002`" sets `rd` to 1 if `R[rs] < 0x0002`.

I/O Manipulation –MIPS System Calls (syscall)

SPIM provides a set of operating-system-like services through the system call (syscall) instructions. Basic input and output can be managed and implemented by system calls.

To request a service, a program loads the system call code into register $v0 and arguments into registers $a0, $a1, $a2, and $a3.

Here is a summary of the system calls for your reference.

Service	**Call Code in $v0**	**Arguments**	**Results**
Print Integer	1	$a0=number(to be printed) Example Code: li $a0, 89 li $v0, 1 syscall	
Print Float	2	$f12=number (to be printed) Example Code: .data flabel: .float 3.14 l.s $f12, flabel li $v0, 2 syscall	
Print double	3	$f12=number (to be printed) Example Code: .data dlabel: .double 3.1415926 l.d $f12, dlabel li $v0, 3 syscall	
Print String	4	a0=address (of string in memory) Example Code: .data str1: .asciiz "Hi There!" la $a0, str1 li $v0, 4 syscall	
Read Integer	5	Example Code: li $v0, 5 syscall	number in $v0
Read Float	6	Example Code: li $v0, 6 syscall	number in $f0
Read double	7	Example Code: li $v0, 7 syscall	
Read String	8	$a0=address (of input string in memory) $a1=length of buffer(n bytes) Example Code: .data str1: .space 80 la $a0, str1 li $a1, 80 li $v0, 8 syscall	

Sbrk (Dynamically allocate n-byte of memory)	9	$a0=n-byte (to allocate) Example Code: li $a0, 80 li $v0, 9 syscall # Get memory move $a0, $v0 li $a1, 80 li $v0, 8 syscall # Read String li $v0, 4 syscall # Print String	address in $v0
Exit	10	Example Code: li $v0, 10 syscall	

Service	Call Code in $v0	Arguments	Results
Print Integer	1	$a0=number (to be printed) Example Code: li $a0, 89 li $v0, 1 syscall	
Print Float	2	$f12=number (to be printed) Example Code: l.s $f12, flabel li $v0, 2 syscall .data flabel: .float 3.14	
Print double	3	$f12=number (to be printed) Example Code: l.d $f12, dlabel li $v0, 3 syscall .data dlabel: .double 3.1415926	
Print String	4	$a0=address (of string in memory) Example Code: la $a0, str1 li $v0, 4 syscall .data str1: .asciiz "Hi There!"	
Read Integer	5	Example Code: li $v0, 5 syscall	number in $v0
Read Float	6	Example Code: li $v0, 6 syscall	number in $f0
Read double	7	Example Code:	number in $f0

		li $v0, 7 syscall	
Read String	8	$a0=address (of input string in memory) $a1=length of buffer(n bytes) Example Code: la $a0, str1 li $a1, 80 li $v0, 8 syscall .data str1: .space 80	
Sbrk (Dynamically allocate n-byte of memory)	9	$a0=n-byte (to allocate) Example Code: li $a0, 80 li $v0, 9 syscall # Get memory move $a0, $v0 li $a1, 80 li $v0, 8 syscall # Read String li $v0, 4 syscall # Print String	address in $v0
Exit	10	Example Code: li $v0, 10 syscall	