MAL Directives

Directives are instructions to the assembler, specifying an action to be taken during the assembly process. One important use of directives is declaring or reserving memory variables. In addition, directives are used to break up the program into sections. The operation name of a MAL directive always begins with a period (".").

Usually, a `.data` section only contains data directives and a `.text` section only contains machine instructions. A MAL program can have multiple `.data` and `.text` sections. An assembler groups all of the `.data` sections together in memory and groups all of the `.text` sections together in a different place in memory.

Directive	Operand Syntax	Meaning
.globl	label { , label }*	Declare labels to be global
.data	none	Start a data declaration section
.text	none	Start an instruction section
.word	integer [: non-negative integer]	Declare a C int variable
.asciiz	string	Declare a string variable

The braces and asterisk are not part of the assembly language code. They are markup notation indicating that the contents inside the braces can be repeated 0 or more times. This means that the operands for the `.globl` directive can be a list of labels separated by commas.

Th Mars simulator requires that the `main` label be declared with `.globl` in order for the program to start executing at the correct address. This is not needed if the `main` label is at the beginning of the first text segment.

The first operand for `.word` specifies the initial value for the variable. The second operand specifies the number of repetitions. The brackets are not part of the assembly language code. They only indicate that the second operand is optional.

MAL Registers

Simple programs should only use the following registers:

- $zero for the constant 0
- $s0 - $s8 for main program variables
- $t0 - $t8 for subprogram variables
- $a0 - $a3 for subprogram and syscall parameters
- $v0, $v1 for subprogram return values and syscall codes and return values

For more complex programs, there are register usage conventions that specify how main programs and subprograms should coordinate their use of registers.

Alternate Name	Register Name	Use
$zero	$0	constant value 0
$s0 - $s8	$16 - $23, $30	saved values - preserved across calls
$sp	$29	stack pointer - preserved across calls
$ra	$31	return address - not preserved across calls
$a0 - $a3	$4 - $7	the first four parameters - not preserved across calls
$t0 - $t9	$8 - $15, $24 - $25	temporaries - not preserved across calls
$v0 - $v1	$2 - $3	expression evaluation and subprogram return value - not preserved across calls
$at	$1	reserved by the assembler - dangerous to use
$gp	$28	global pointer - dangerous to use
$k0 - $k1	$26 - $27	reserved by the operating system - dangerous to use

The usage convention designations have significant implications for both subprograms and their callers.

- When a register is designated as "preserved across calls", it means that the caller can count on the register having the same contents before and after a subprogram call. If the subprogram uses one of these registers, it should take measures to save the register value before changing it and restore the value before returning.
- If a register is designated as "not preserved across calls", it means that the caller cannot count on the register having the same contents before and after a subprogram call. Thus the subprogram can use the register freely. If the caller puts a value into one of these registers before a subprogram call and needs the value after the call, then the caller has the responsibility of saving and restoring the value.
- If a register is designated as "dangerous to use", it means that the register is used either by the operating system or the assembler. Most programs should avoid the use of these registers.