MIPS assembler syntax

Comments begin with a sharp sign (#). Everything from the sharp sign to the end of the line is ignored.

Identifiers are a sequence of alphanumeric characters, underbars (_), and dots (.) that do not begin with a number.

Registers are denoted by $0 - $31 or $t0, $t1, $a0 etc. (see the register window).

Instruction opcodes are reserved words that cannot be used as identifiers.

Labels are declared by putting themat the beginning of a line followed by a colon, for example:

```
                .data
        item:   .word 1

                .text
                .globl __start   # Must be global
    __start:  lw $t0, item
```

Numbers are base 10 by default. If they are preceded by 0x, they are interpreted as hexadecimal. Hence, 256 and 0x100 denote the same value.

Strings are enclosed in double quotes (”). Special characters in strings follow the C convention:

- newline \n
- tab   \t
- quote \”

MIPS assembler directives

.ascii str	# Store the string str in memory, but do not null-terminate it.
.asciiz str	# Store the string str in memory and null-terminate it.
.byte b_1,..., b_n	# Store the n values in successive bytes of memory.
.data	# Subsequent items are stored in the data segment.
.double d_1, ..., d_n	# Store the n floating-point double precision numbers # in successive memory locations.
.float f_1,..., f_n	# Store the n floating-point single precision numbers # in successive memory locations.
.globl sym	# Declare that label sym is global and can be referenced from other files.
.half h_1, ..., h_n	# Store the n 16-bit quantities in successive memory halfwords.
.space n	# Allocate n bytes of space in the current segment (which # must be the data segment in SPIM).
.text	# Subsequent items are put in the user text segment. In SPIM, these items # may only be instructions or words (see the .word directive).
.word w_1,..., w_n	# Store the n 32-bit quantities in successive memory words.

Instruction format, pseudo-instructions

```
opcode [operand1 [, operand2 [, operand3] ] ]

operand ::= register | label | number | number(register) | label(register)
```