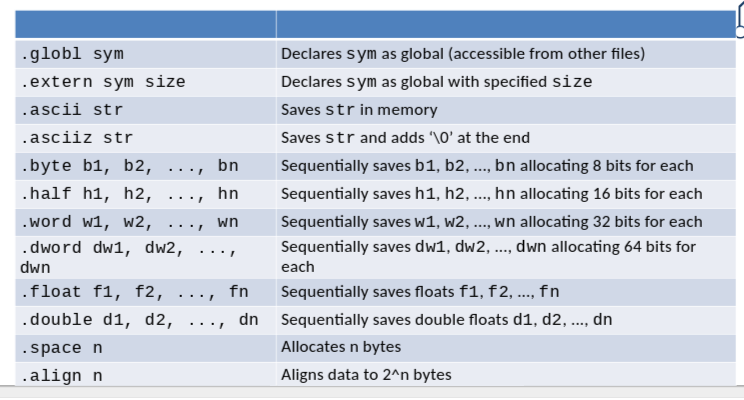
Comp lecture #13

Elements of MAL

* Directives start with ‘.’:
* .data data segment,
* .text code segment.
* Labels end with ‘:’
* v:, main:, loop: and endloop:.
* Registers start with ‘$’.
* Comments start with ‘#’.
* Instructions and macros (pseudo-instructions).
* Constants.

Basic MAL Directives

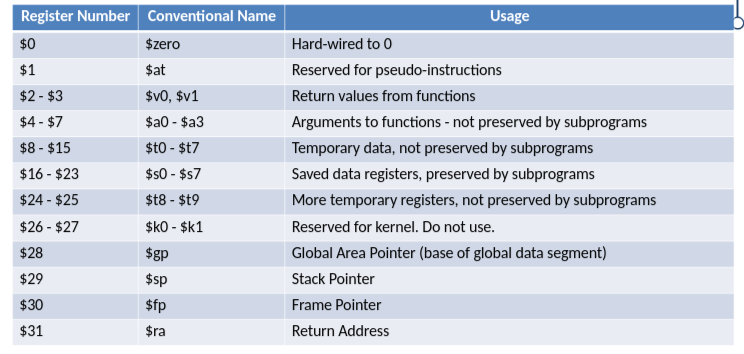


**Labels in MAL**

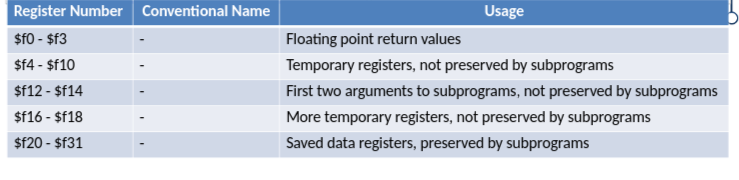
* **in the .data segment, data labels define addresses of variables.**
* **in the .text segment instruction labels define addresses of instruction.**

**MIPS Registers**

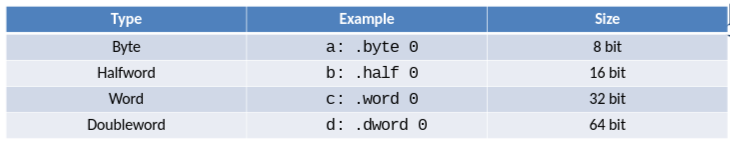
* **MIPS has 32 main register and 32 co processor registers**
* **Size of a register is 32 bits**
* **Registers saves address, data and instruction**
* **Max mem = 232 4G bytes (doesnt make sense to use 64 bit if CPU is only 32-bit)**
* **All MIPS arithmetic operations use registers.**
* **Register’s name starts with ’$’ and has two synonymic versions:** 
  + **Number**
  + **symbolic name.**



**MIPS Registers for Floating Point Operations**

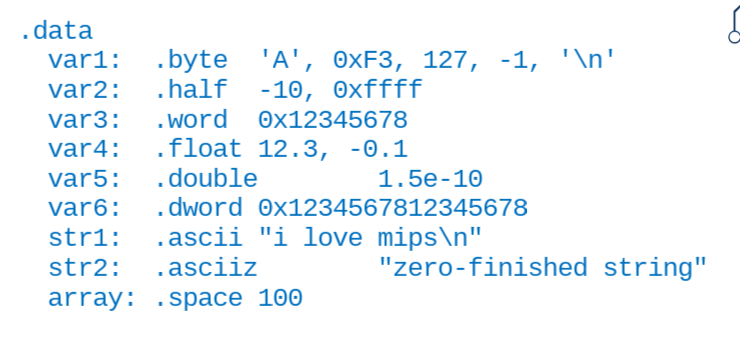


**MAL Data Types**



* **In MAL a type defines only the size of data.**
* **The interpretation of binary code depends on instruction.**

**Examples of Data Definitions**

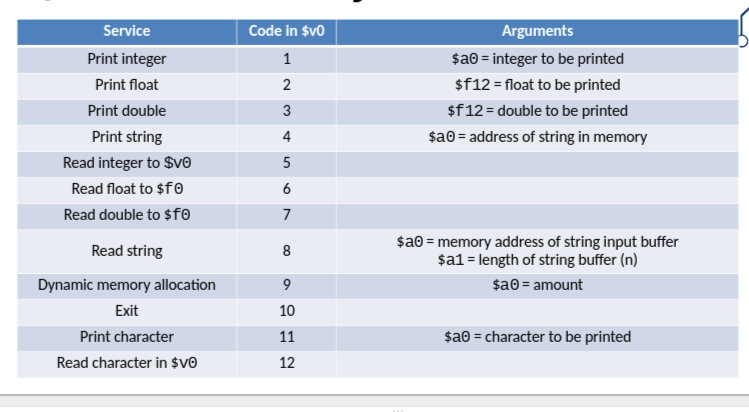


* **.asciiz differe from .ascii in that .asciiz terminates**

**Source Code**

* **Each line of code can be**
  + **Instruction (?:label)**
  + **Single directive**
  + **Empty line**
  + **Comment**
* **Comment starts:#, ends:newline**

**Special Instruction syscall**

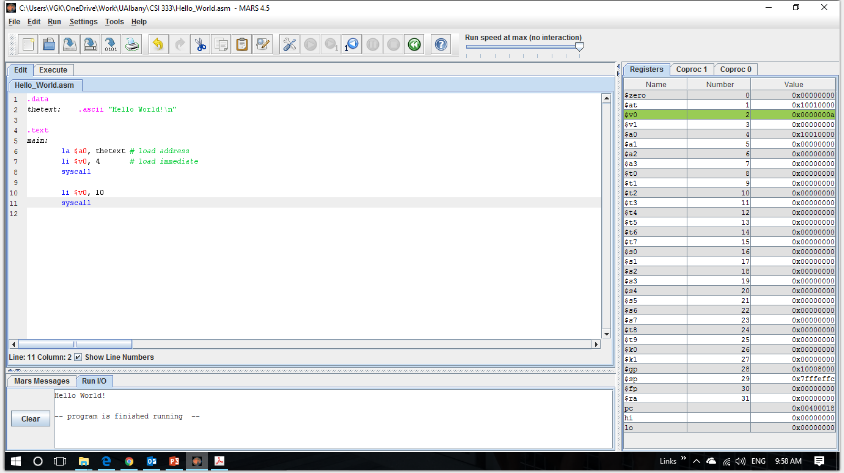


* **Lets your code commmunicate with os**
* **If something goes wrong the OS usually tells you in C**
* **If you mess up your processor, you can destroy your machine**
* **Use software (simulators) to run your mips code**

***MAL Simulators***

* Direct experiments with CPU can destroy it
* Use simulators such as SPIM and MARS to test your code

**MAL Simulators**



* **First part data that defines the strings**
* **The list to your right are 32 registers**
* **Before execution all registers are zero**
* **The values in the register change as they need to**
* **Change value of regsiter v0, to 4 becuase next instruction is syscall**
* **Goes to register v0 to find what to do, print string at register (memory location) a0**
* **In MIPS you can execute your code step by step to see what is happening**
* **Exit called done by syscall, if code does not do it itself**
* **Values placed into registers are binary**
* **Use hexadecimals to save space**
* **0x tells you if a number is hexadecimal**

**SPIM Interface**

* **Make sure code runs well with SPIM simulator**

**MAL Instructions**

* **Psuedo-instructions in the previous example** 
  + **la rdest, addr**
* **translates to**
  + **lui $at, hi(addr)**
  + **ori rdest, $at, lo(addr)**
* **Too hard to remember all instruction, but have to know where code is**
* **Use psuedo-instruction**

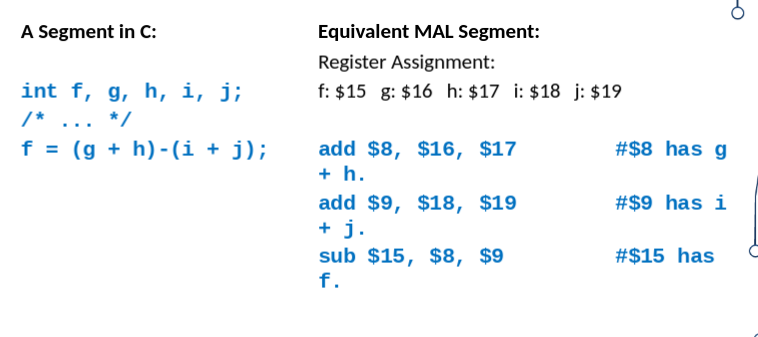
**Types of Instructions**

* **Type R(register)**
  + **Three operands,**
    - **Destination register ($rd)**
    - **First argument ($rs)**
    - **Seconde argument ($rt)**
    - **add $t2, $t0, $t1**
    - **$t0 + $t1 -> $t2**
* **Type I (immediate)**
  + **Two registers and constant**
  + **addi $t3, $t2, 12**
  + **$t2 + 12 -> $t3**
* **Type J (jump)**
  + **one operand: new address for PC.**
  + **E.g. j 128.**

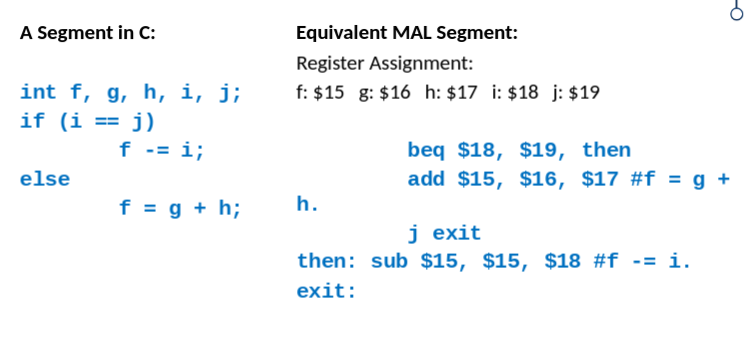
TRANSlating C segments into MAL

* Assembly language can take a long time
* To introduce MAL instructions
* To convery level of detail involved

Example 1

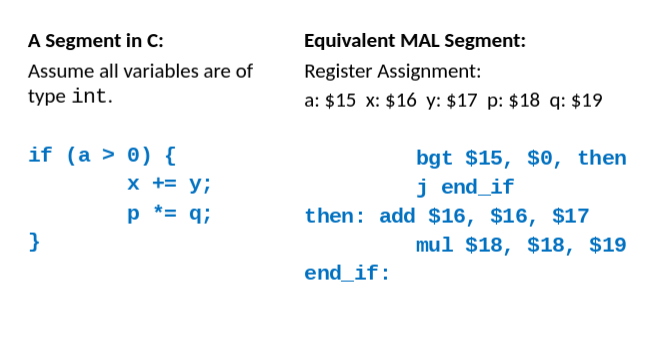


Example 2

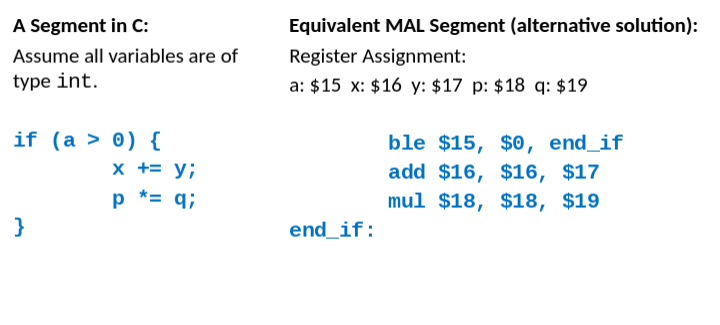


* Here the else statement comes right after tthe conditional and exits
* The then label comes to as representation of the if statement

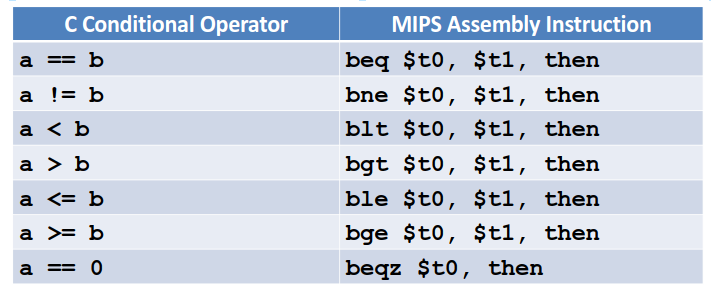
Example 3



alt

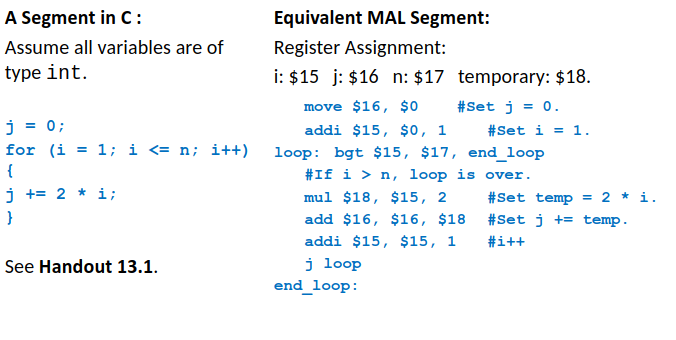


**C and Assembly Conditional Operators**

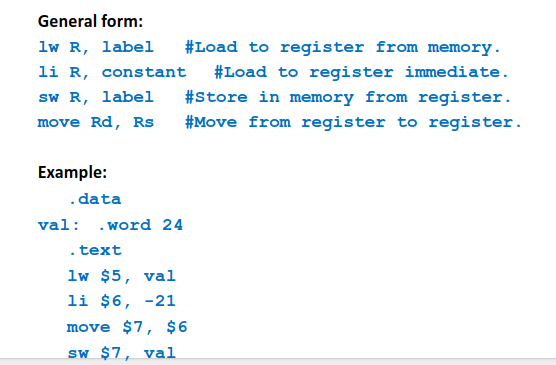


**Loops**

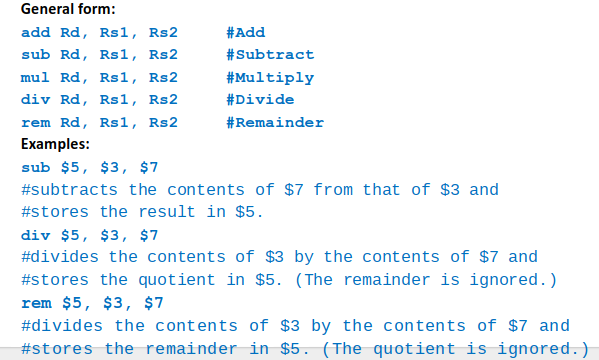
**Translating C Segments into MAL – For Loop**



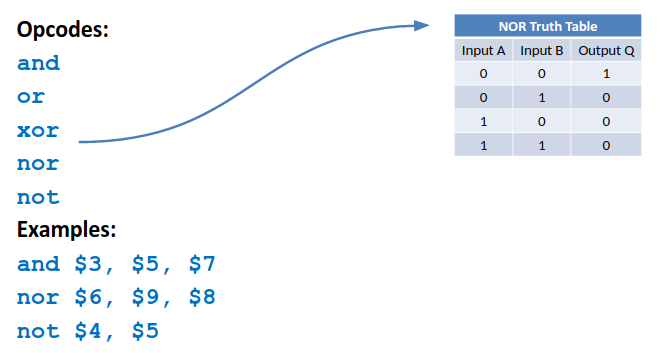
**Important MAL Instructions: Load, Store and Move**



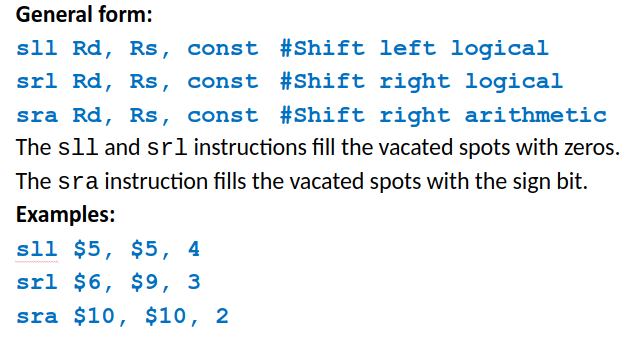
**Important MAL Instructions: Arithmetic Instructions**



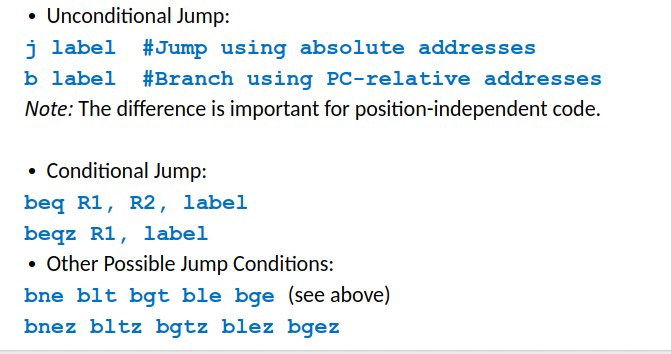
**Important MAL Instructions: Logical Bitwise Operators**



**Important MAL Instructions: Shift Instructions**



**Important MAL Instructions: Jump Instructions**



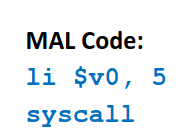
**A Simple MAL Program**

**For 13.3 notes**

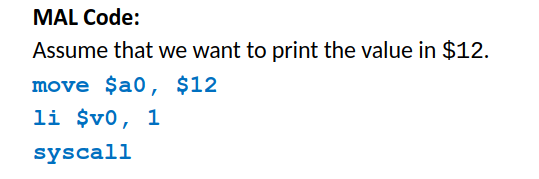
**The syscall Instruction**

* **Used for I/0 and for stopping the program**
* **The operation to be carried out is specified as a command (in register $v0) to syscall.**
* **$v0 is a synonym for $2. This register must contain the command for syscall. This register also contains the return value (if any) produced by executing the command.**
* **$a0 and $a1 are synonyms for $4 and $5 respectively. These registers must contain suitable values if such values are needed for the command**

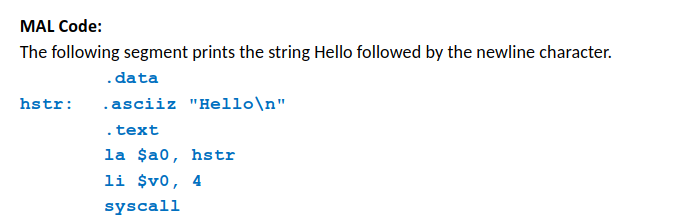
**Using syscall to Read an Integer**



**Using syscall to Print an Integer**

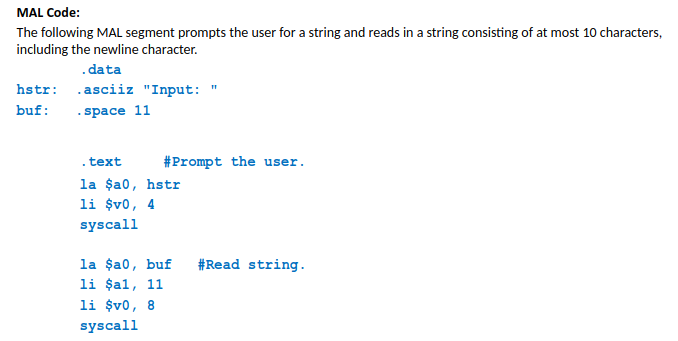


**Using syscall to Print a String**

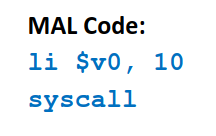


**Using syscall to Read a String, cont’d**

|  |  |  |
| --- | --- | --- |
| Service | $v0 | Arguments |
| Print integer | 1 | $a0 = integer to be printed |
| Print float | 2 | $f12 = float to be printed |
| Print double | 3 | $f12 = double to be printed |
| Print string | 4 | $a0 = address of string in memory |
| Read integer to $v0 | 5 |  |
| Read float to $f0 | 6 |  |
| Read double to $f0 | 7 |  |
| Read string | 8 | $a0 = memory address of string input buffer $a1 = length of string buffer (n) |
| Dynamic memory allocation | 9 | $a0 = amount |
| Exit | 10 |  |
| Print character | 11 | $a0 = character to be printed |
| Read character in $v0 | 12 |  |



**Using syscall to Stop a Program**



*Saving and Restoring Registers*

