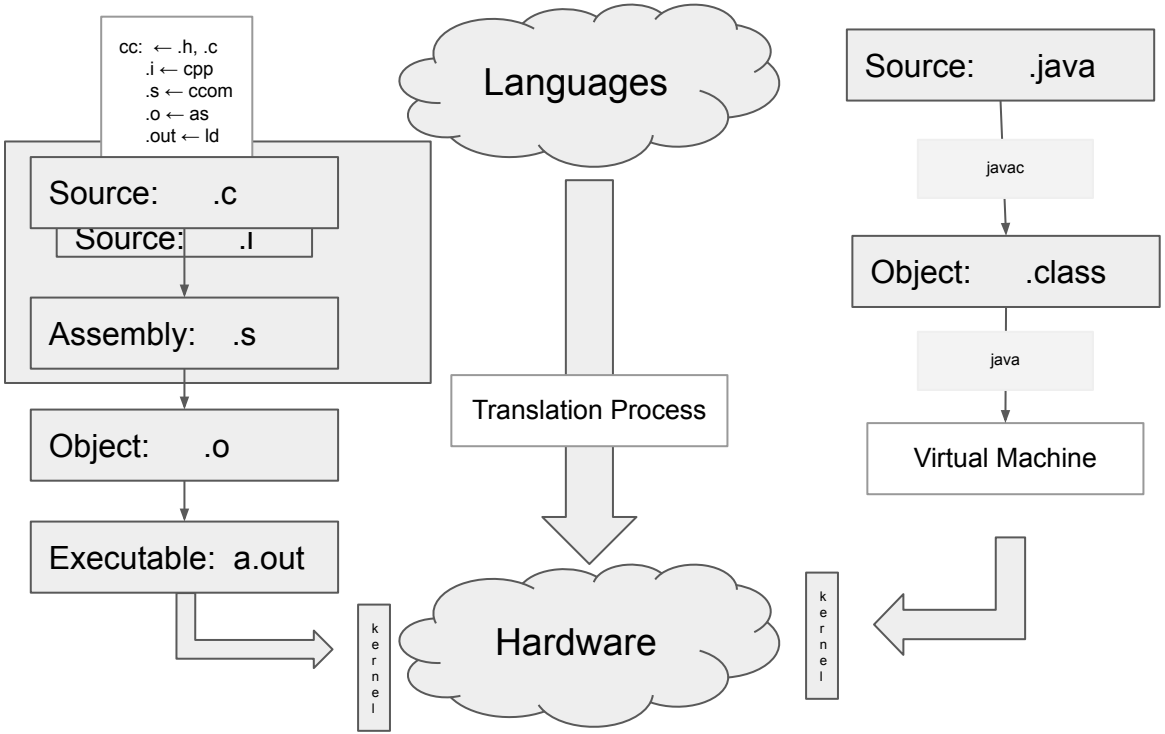
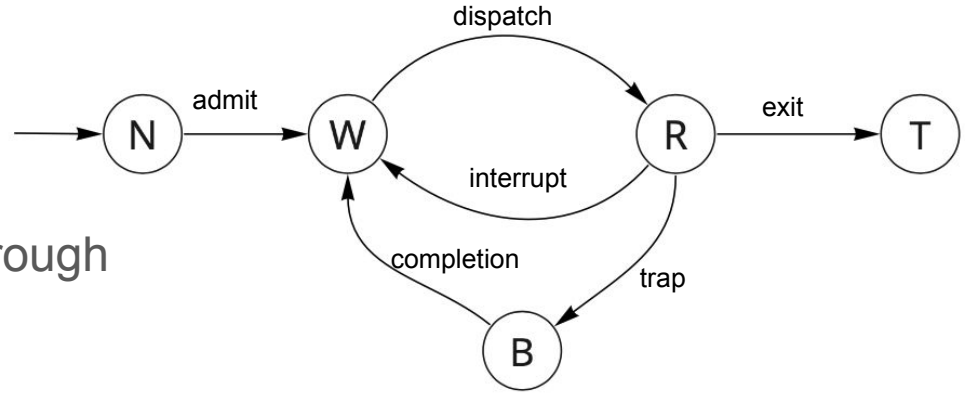
Languages, Compilers, and Hardware:

- Languages
  - Domain Specific
- Compilers & Interpreters
  - Analysis
    - lexicographical
    - syntaxical
    - semantics
  - Language Optimization
  - Machine Optimization
  - Translation: TAC → MIPS
- Hardware
  - General Types: Registers / Stack
  - Specific CPU Controls
- CLI: compilation exercise



Process Status Diagram



- Control of the computer moves through a well-defined cycle

- Transitions:
  - admit: A request is made to allow your program to content for control
  - dispatch: Your program is given control
  - exit: Your program asserts that it is done
  - interrupt: The OS seizes control
  - trap: Your program (implicitly or explicitly) requests a service to be performed
  - completion: The request is satisfied

- Traps are calls to the Kernel (the OS)

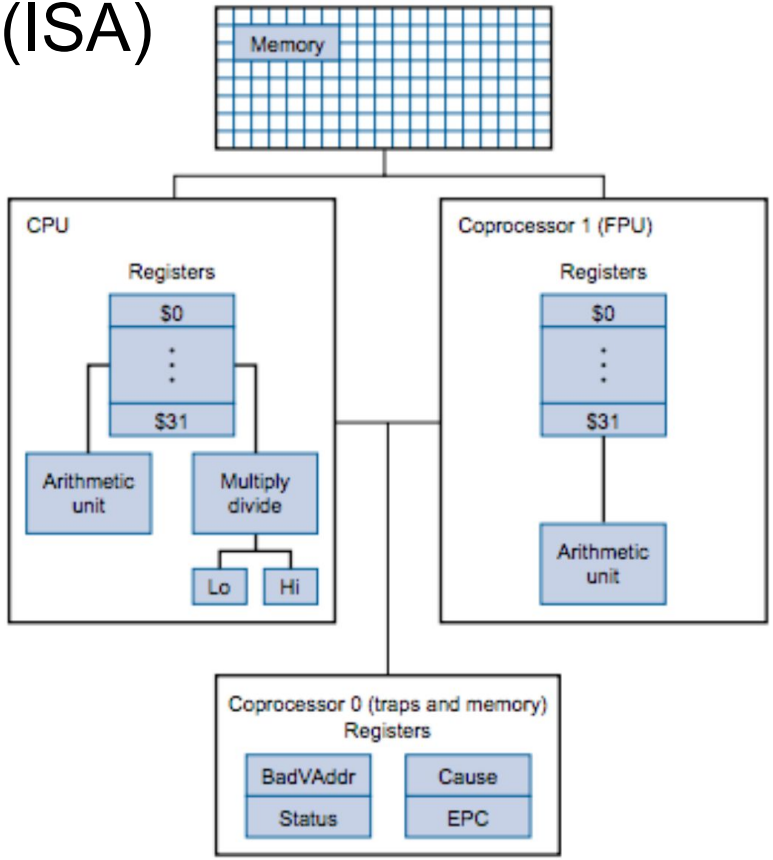
MIPS System Calls: SystemCallAPI.png

1. print integer
2. print float
3. print double
4. print string
5. read integer
6. read float
7. read double
8. read string
9. allocate memory
10. terminate
11. print character
12. read character
13. file open
14. file read
15. file write
16. file close

x86: has 4 general purpose registers

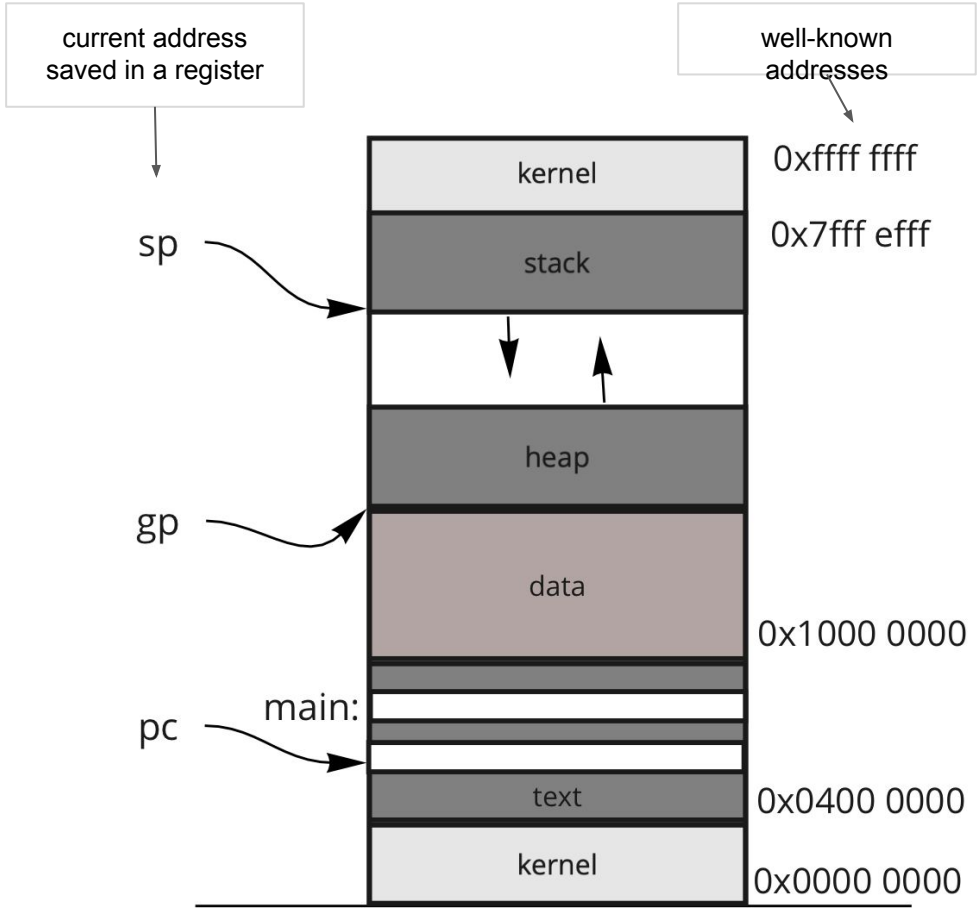
MIPS Instruction Set Architecture (ISA)

- General Architecture
  - RISC (Reduced Instruction Set Computer)
  - Simple Instructions
  - Lots of Registers
  - Remember Memory is SLOW!
- Instruction Set:
  - List of Instructions Supported by the Architecture
  - MIPS Cheat Sheet
- Memory
- CPU
  - ALU
  - 32 general purpose registers
- Coprocessors
  - Floating point
  - Traps, Exceptions, Interrupts



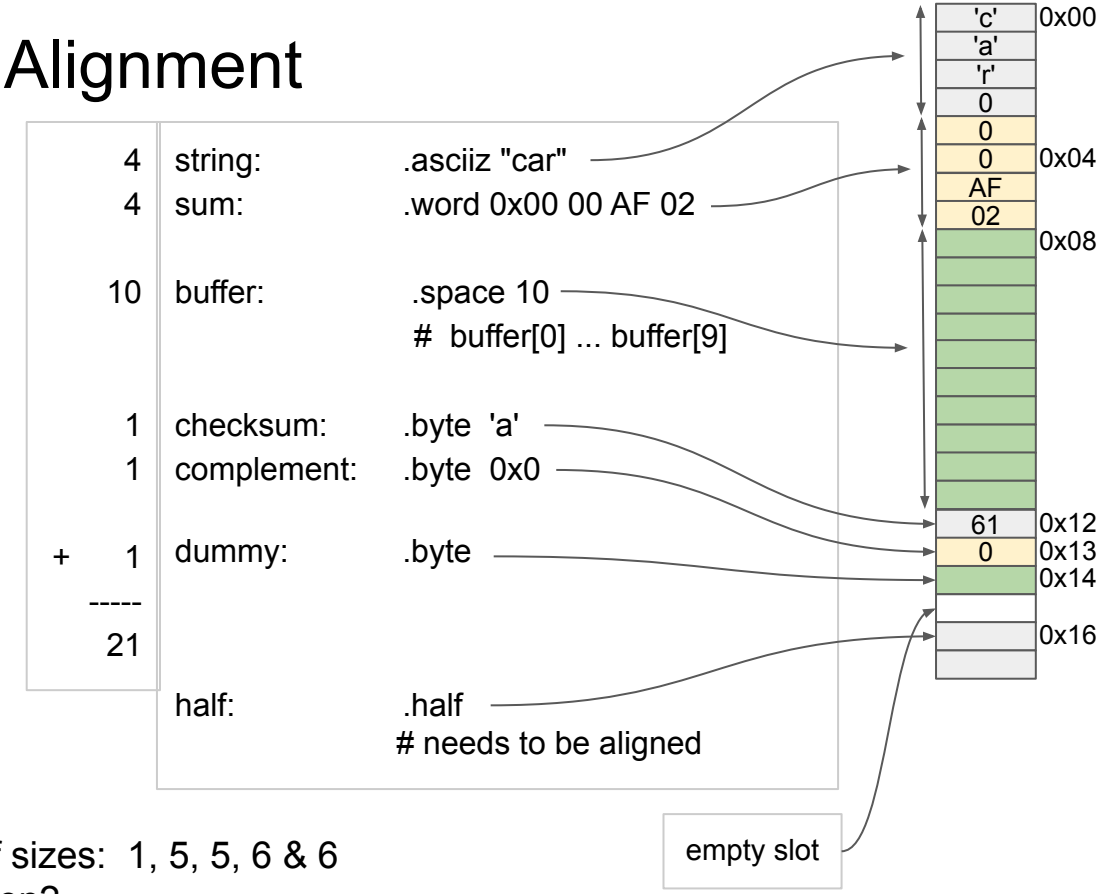
Main Memory

- View & Orientation
  - Array of Bytes: (i.e., byte addressable)
- Data Segments: (to name a few)
  - .text
  - .data
    - .lit4, .lit8 (4 and 8 byte literals)
    - .bss (block storage)
  - heap
  - stack
- Data Declarations and Sizes
  - .byte, .half, .word
  - .ascii, .asciiz
  - .float, .double
  - .space
- Alignment
- Endianness



Data Declarations and Alignment

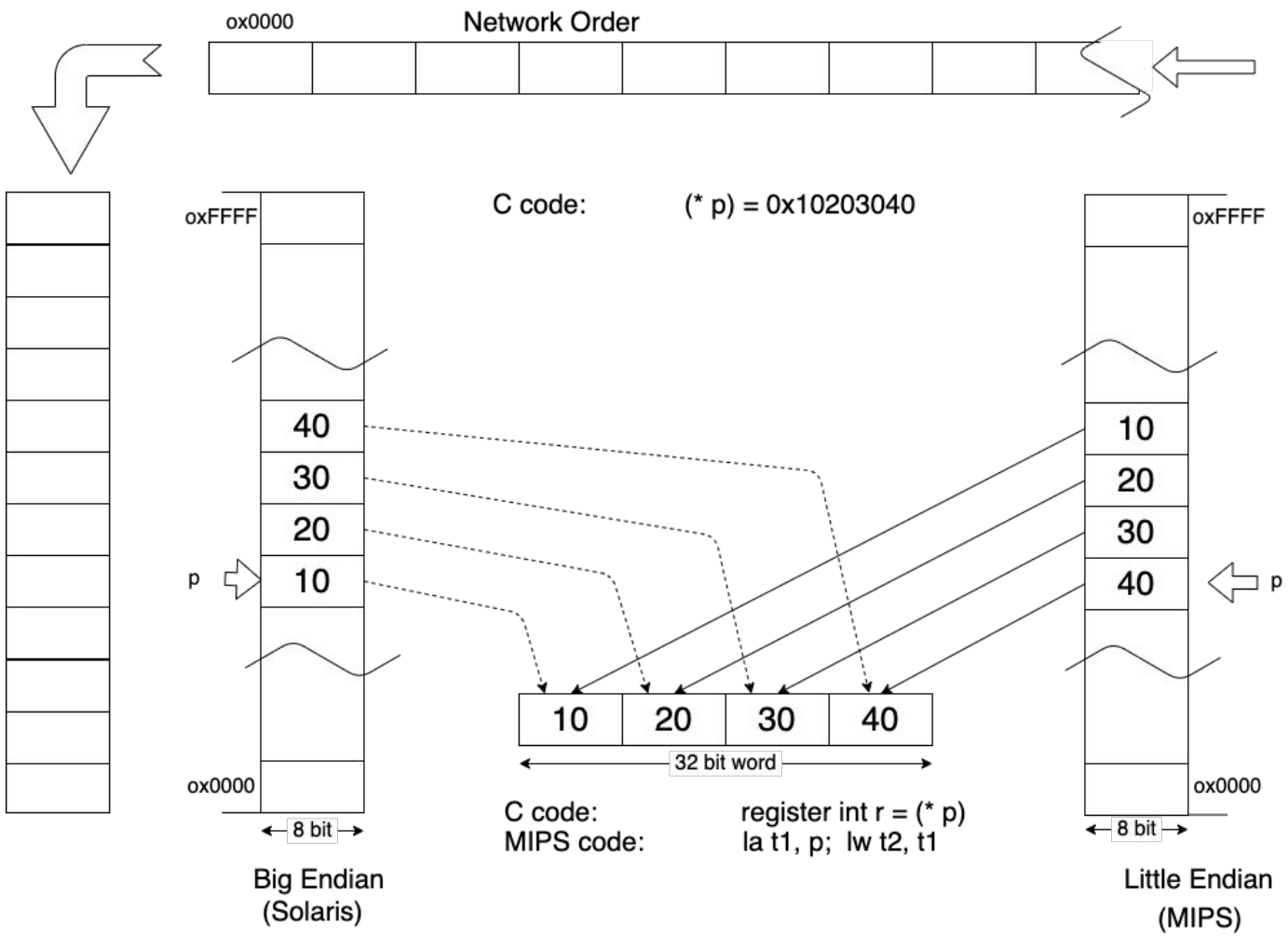
- Alignments:
  - byte aligned (1)
  - half aligned (2)
  - word aligned (4)
  - paragraph aligned (16)
  - page aligned (64K)
- Foreshadow
  - Virtual Memory
  - Paging Systems
  - Fragmentation
    (wasted space)



You have a carton of size 14, and boxes of sizes: 1, 5, 5, 6 & 6
How many boxes can you place in the carton?

Endianness

- The order of bytes within a word
- Network Order
  - Big Endian
- Host Order
  - Big Endian
  - Little Endian



Addressing Modes

```
var1:   .word  0xface
var2:   .half  0x22
lst:    .space 10
```

- Registers are prepend with a dollar sign, literals are not
  - ARM: literals are appended by a sharp, registers are not.
- immediate: is when the actual value is one of the operands.

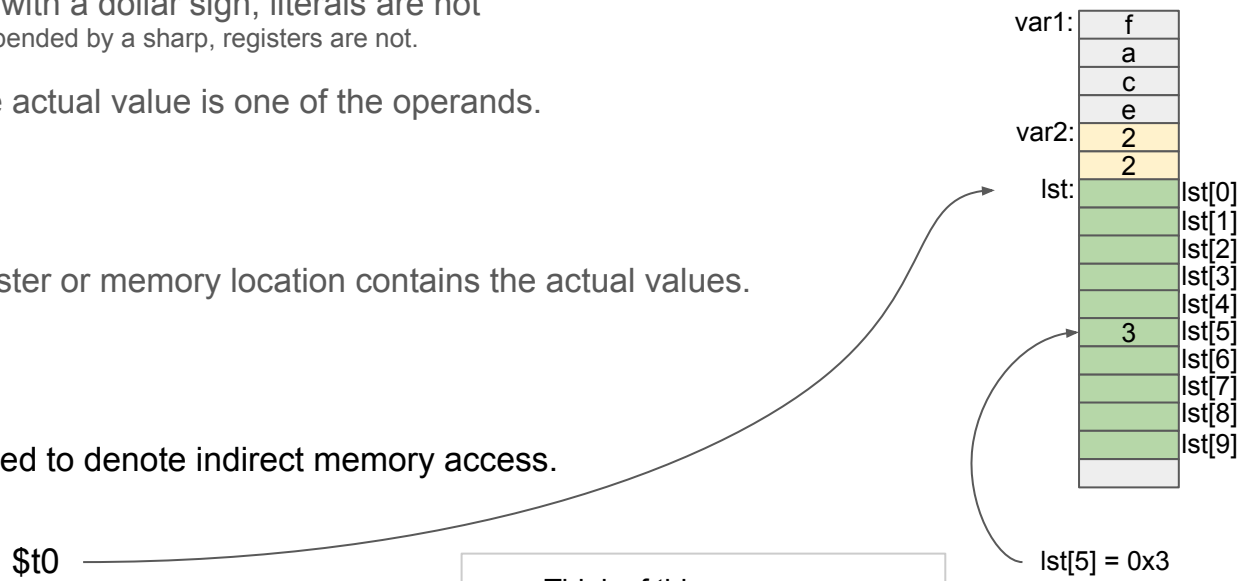
```
li $t0, 57
add $t0, $t0, 57
```

- direct: is when the register or memory location contains the actual values.

```
lw $t0, var1
lh $t1, var2
```

- indirect: the ()'s are used to denote indirect memory access.

```
li $s1, 0x3
la $t0, lst
sb $s1, 5($t0)
```

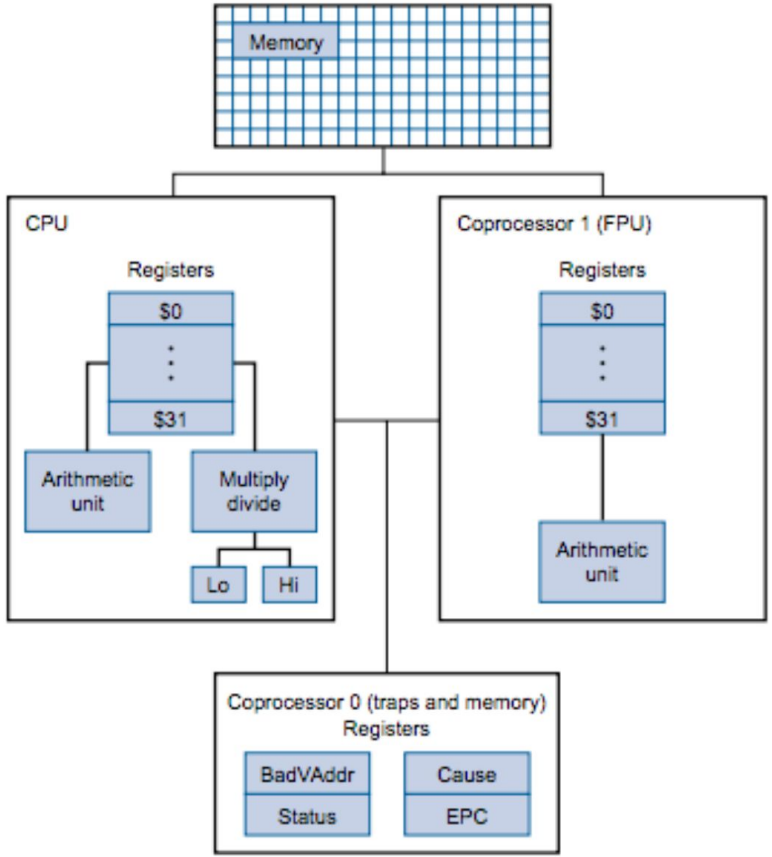


Think of this as an array,
but with a different syntax:

list[5] ⇔ 5(list)
$t0[4] ⇔ 4($t0)

Registers

- System
  - PC: Program Counter
  - IR: Instruction Register
  - BadVAddr: memory address where exception occurred
  - Status: Interrupt mask, enable bits and status when exception occurred
  - Cause: Type of exception
  - EPC: Address of instruction that caused the exception
- Reserved (Don't use!)
  - $at: reserved for the Assembler
  - $k1, $k2: reserved for the Kernel
  - $gp: global pointer defined by the compiler
- Special (Access via specific instructions)
  - PC: program counter
  - hi, lo: used double word results
    - (hi, lo) = val1 * val2
    - (hi, lo) = val1 % val2
- General Purpose
  - 32 32-bit integer registers: $0..$31
  - 32 32-bit floating point registers: $f0..$f31

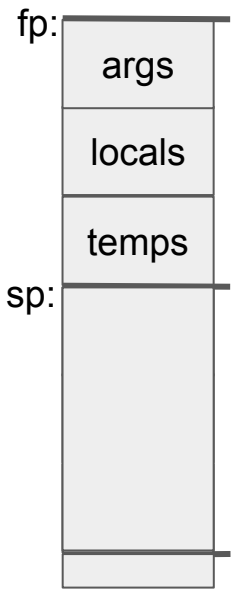


General Purpose Registers

Name	Register Number	Usage
`$zero`	0	constant 0 (hardware)
`$v0 - $v1`	2 - 3	subroutine return values
`$a0 - $a3`	4 - 7	subroutine arguments
`$t0 - $t7`	8 - 15	temporaries
`$s0 - $s7`	16 - 23	saved temporaries
`$t8 - $t9`	34 - 35	temporaries
`$sp`	29	stack pointer
`$fp`	30	frame pointer
`$ra`	31	return address (hardware)

Subroutine Calls

- Subroutine: generic term for: function, procedure, method, whatever!
- If you are a good programmer, you have
  - many or few subroutines?
- Frame of Memory for a subroutine
  - input arguments are placed onto the stack
  - local variables are placed onto the stack
  - any tempories are placed onto the stack
- Where does the return value go?




```
int f (int A1, int A2, int A3);
int g (int A1);

int f(int A1,int A2, int A3) {
    int L1;

     L1 = <some calculation>
   {
     int L3 ;

     L3 = A1 * L1 + g(A2+L2);
     L1 = L3 | A3

   }
   return (L1);

}
```

- Remember Memory is SLOW!

CPU General Purpose Registers: ($0 -- $31)

- $t0 - $t9: temporary registers
- $zero: holds the value 0
  - All needed literals must be stored in memory
  - But memory is slow, so keep 0 in a register
- $s0 - $7: saved registers
- Subroutine Specific
  - $v0 - $v1: return values
  - $a0 - $a3: input arguments
- Special Usage:
  - $sp: stack pointer
  - $fp: frame pointer
  - $ra: return address

x = f(a, b)