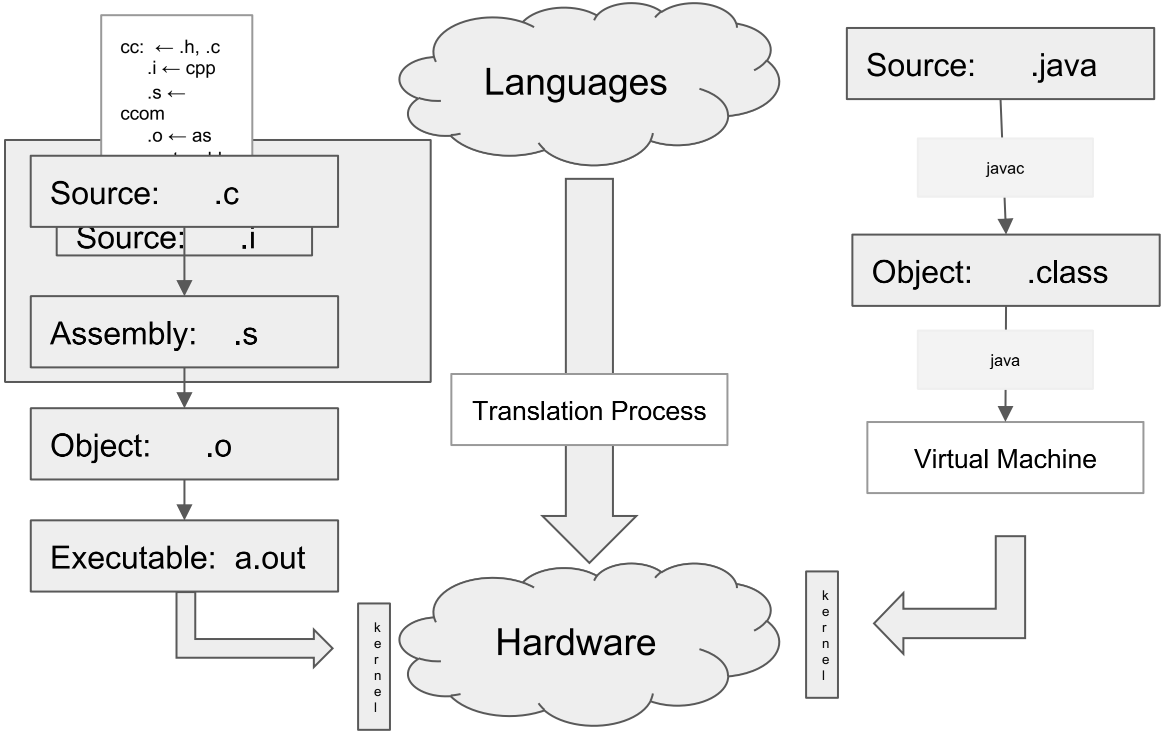
Languages, Compilers, and Hardware:

- Languages
    - Domain Specific

- Compilers & Interpreters
    - Analysis
        - lexicographic
        - syntactic
        - semantic
    - Language Optimization
    - Machine Optimization
    - Translation: TAC → MIPS

- Hardware
    - General Types: Registers / Stack
    - Specific CPU Controls

- CLI: compilation exercise



cc: ← .h, .c
.i ← cpp
.s ← ccom
.o ← as

Source: .c

Source: .i

Assembly: .s

Object: .o

Executable: a.out

kernel

Languages

Translation Process

Hardware

Source: .java

javac

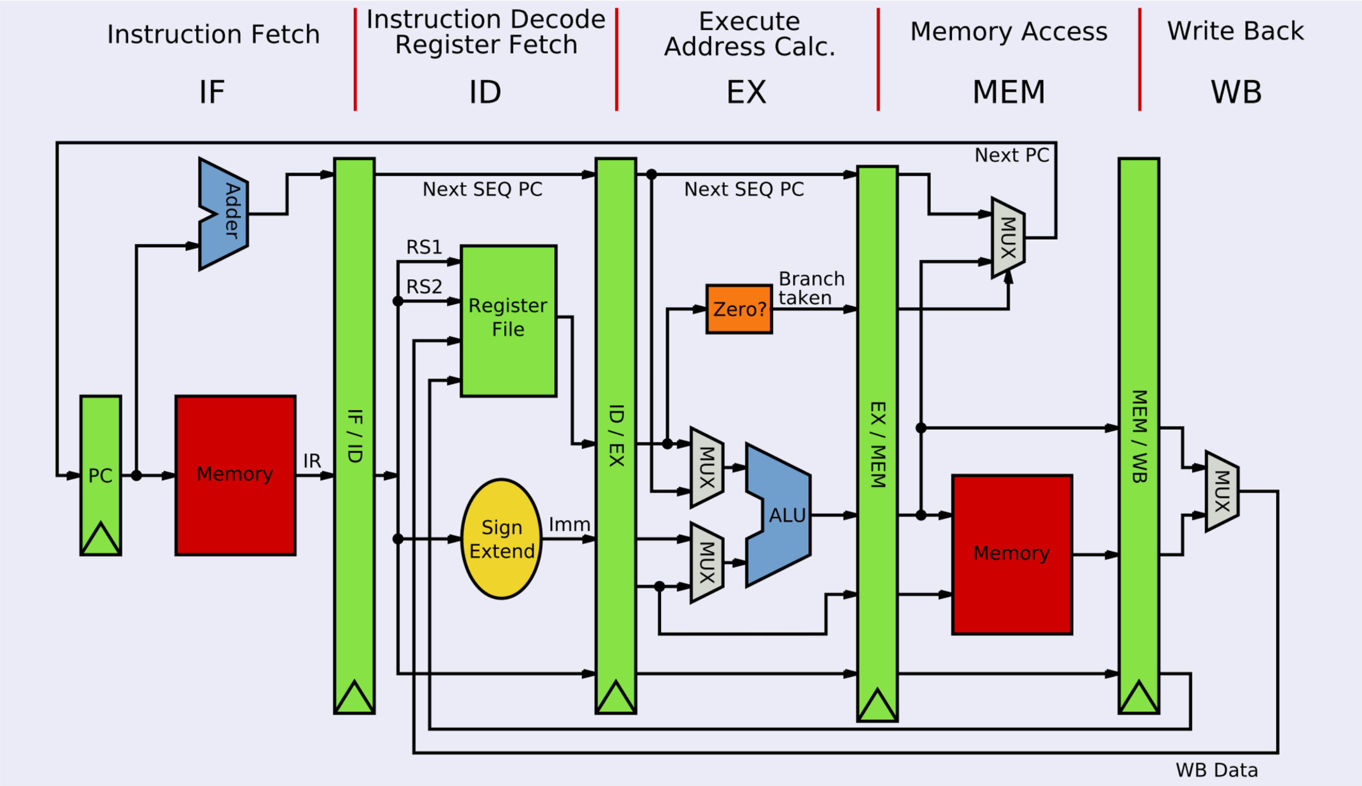
Object: .class

java

Virtual Machine

kernel

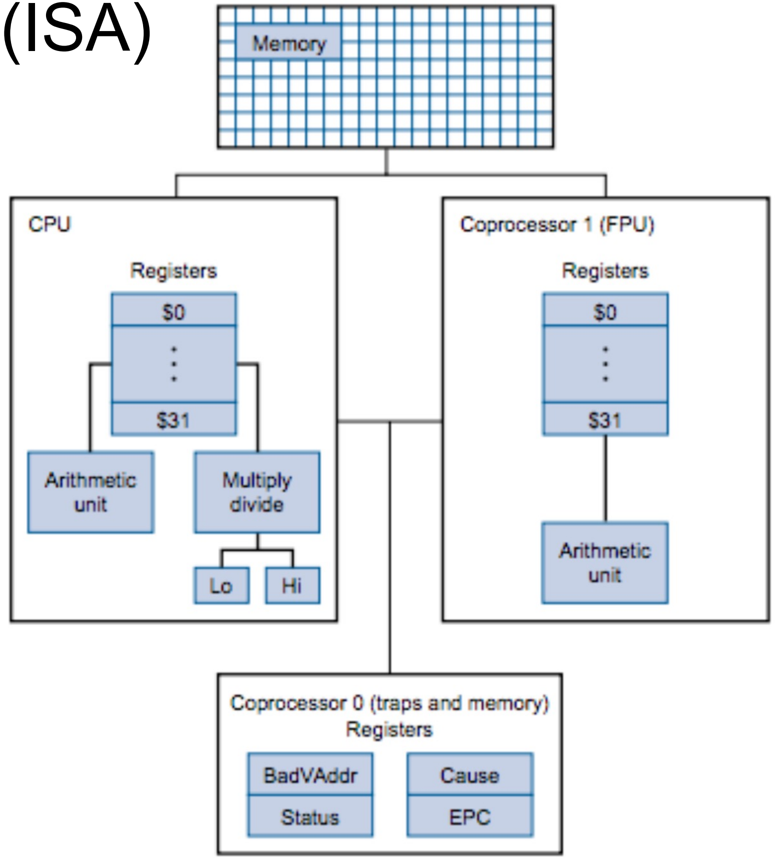
MIPS Microarchitecture



x86: has 4 general purpose registers

MIPS Instruction Set Architecture (ISA)

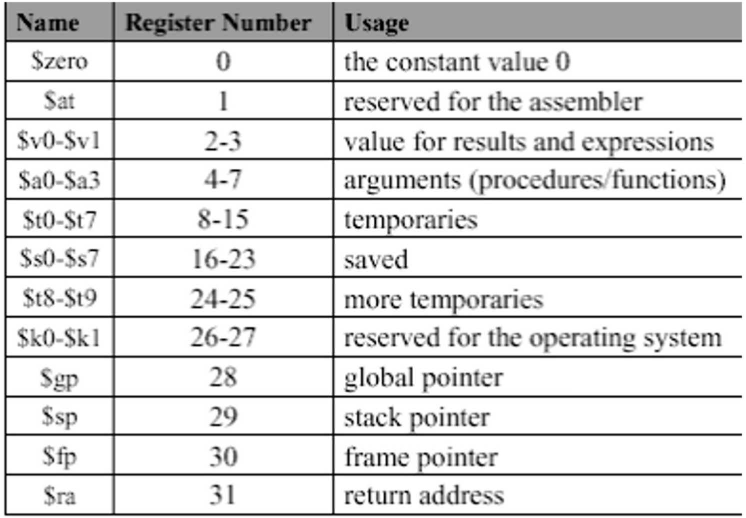
- General Architecture
  - RISC (Reduced Instruction Set Computer)
  - Simple Instructions
  - Lots of Registers
  - Remember Memory is SLOW!
- CPU
  - ALU
  - 32 general purpose registers
- Instruction Set:
  - List of Instructions Supported by the Architecture
  - [MIPS Cheat Sheet]
- Coprocessors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



MIPS ISA Architecture: Registers

- Data types:
  - byte, half, word
  - integer (signed/unsigned), binary32, binary64
- Registers:
  - 32: 32-bit integer registers
  - 32: 32-bit floating point registers
    - binary32: $fp0 .. $fp31
    - binary64: {$fp0, $fp1} .. {$fp30, $fp31}
  - 3: system registers: pc, hi, lo

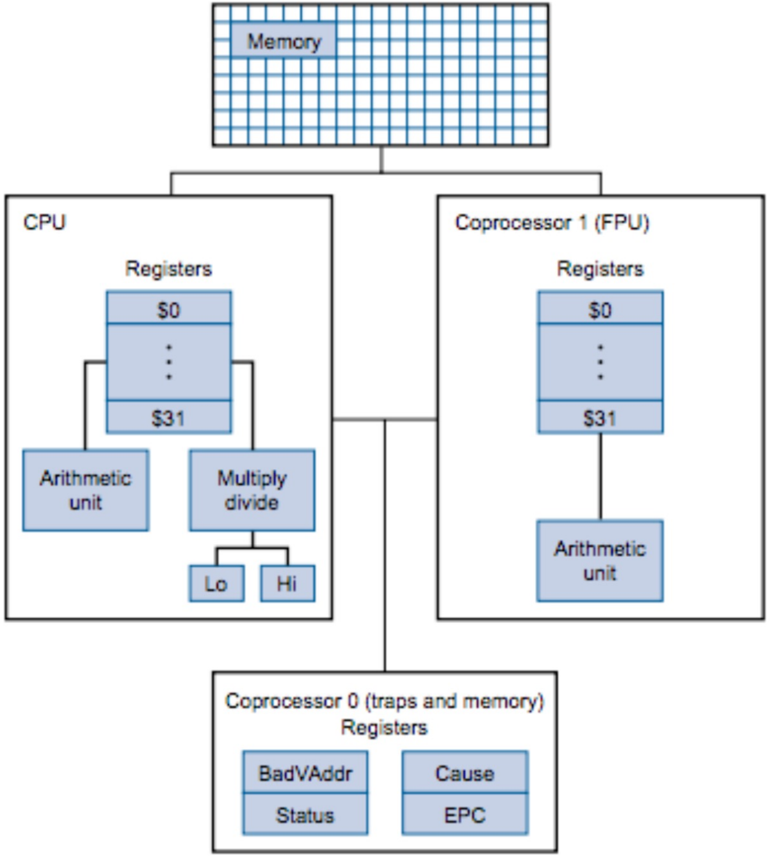
Integer Registers



Name	Register Number	Usage
$zero	0	the constant value 0
$at	1	reserved for the assembler
$v0-$v1	2-3	value for results and expressions
$a0-$a3	4-7	arguments (procedures/functions)
$t0-$t7	8-15	temporaries
$s0-$s7	16-23	saved
$t8-$t9	24-25	more temporaries
$k0-$k1	26-27	reserved for the operating system
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address

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



Category of Instructions

- native instructions:
  - defined by the ISA, directly corresponds to a hardware operation
- idioms:
  - defined by the ISA, an alternate form of a hardware instruction
  - `b label`
    - `beq $zero, $zero, label`
- pseudo instructions:
  - an alternate syntactic form of an instruction
  - pseudo instruction is textual replaced with native instruction
  - `lb $t1, label+offset`
    - `lui $at, &label`
    - `lb $t1, offset($at)`
- macros:
  - user-defined pseudo instruction
    - `average $v0, $t2, $t3`
    - `average($v0, $t2, $t3)`
- subroutines
  - user defined abstraction, resulting
    - `jal subroutine`
    - `nop`
  - a change in control-flow
  - a ownership of registers

```
.macro average(%d, %s, %t)
    addu %d, %s, %t
    srl %d, 2
.end_macro
```

```
                   .text
subroutine:             nop
                   jr $ra
```

MIPS ISA Architecture: Instructions

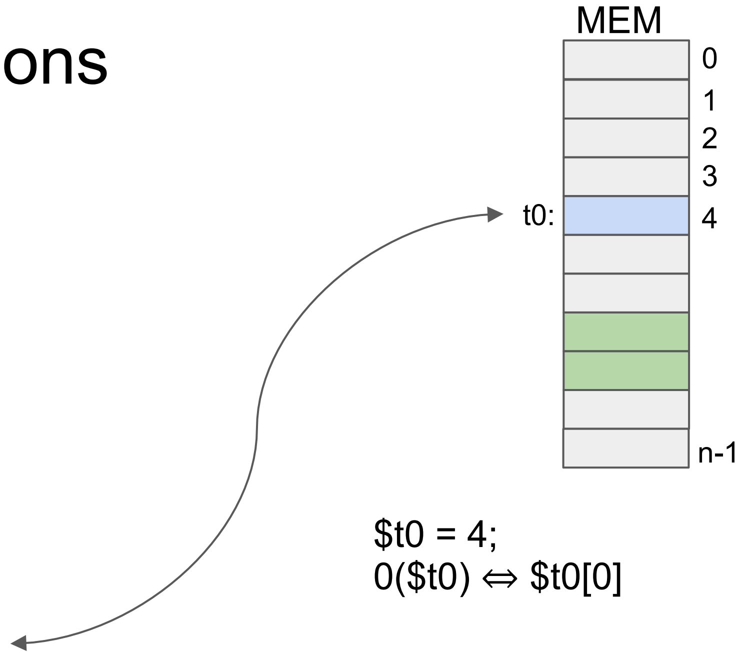
- Three basic instruction types
  - Arithmetic, bitwise logic, etc.
  - Data transfers
  - Basic control flow

- Examples:

```
○ add  $v0, $v0, $a0    #  $v0 = $v0 + $a0
○ addi $v0, $v0, 2      #  $v0 = $v0 + 2
○ srl  $a0, $a1, 4      #  $a0 = $a1 >>> 4
○ li   $t0, 4           #  $t0 = 4

○ move  $t1, $t2        #  $t1 = $t2
○ lb   $s0, 0($t0)      #  $s0 = MEM[$t0]
  lh   $s1, 3($t0)      #  $s1 = concat(MEM[$t0+3+1],MEM[$t0+3+0])

○ beq  $t3, $t5, label  # if ($t3 == $t5) goto label
○ jal  proc             # method()
```



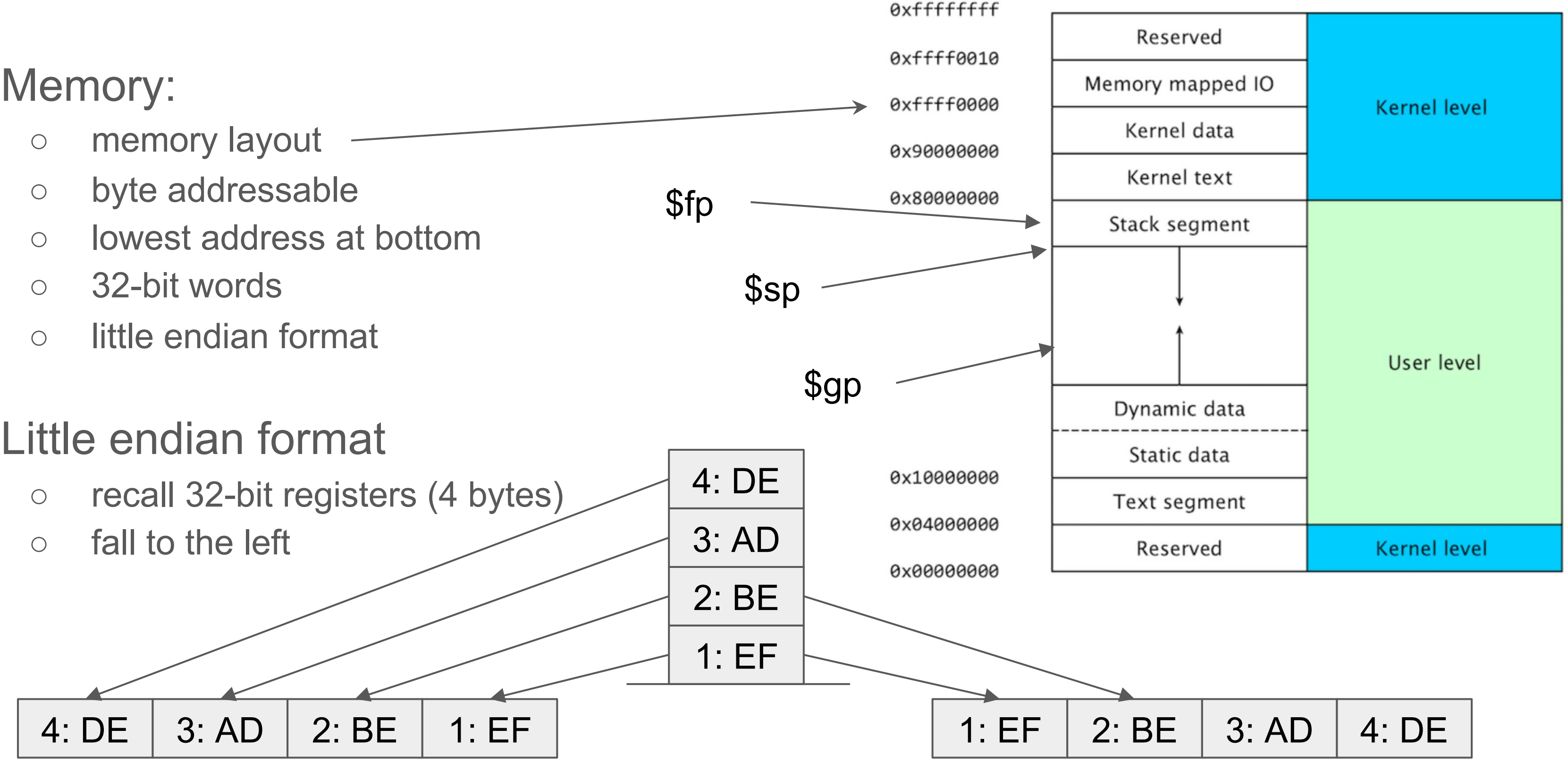
$t0 = 4;
0($t0) ⇔ $t0[0]

endiance: the order of bytes within a word

- big: 1,2,3,4 (yy/mm/dd)
- little: 4,3,2,1 (dd/mm/yy)
- middle: 3,4,1,2 (mm/dd/yy)

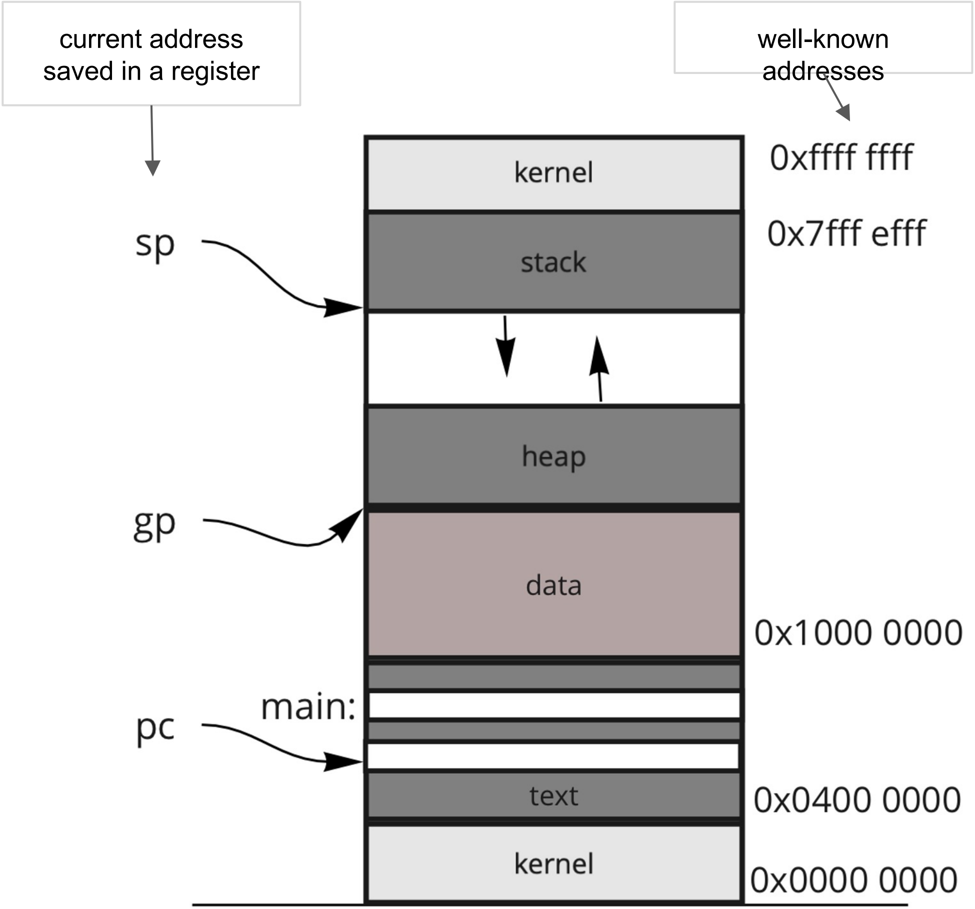
MIPS ISA (Architecture) Memory Layout

- Memory:
  - memory layout
  - byte addressable
  - lowest address at bottom
  - 32-bit words
  - little endian format
- Little endian format
  - recall 32-bit registers (4 bytes)
  - fall to the left



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



Memory Organization (Java program)

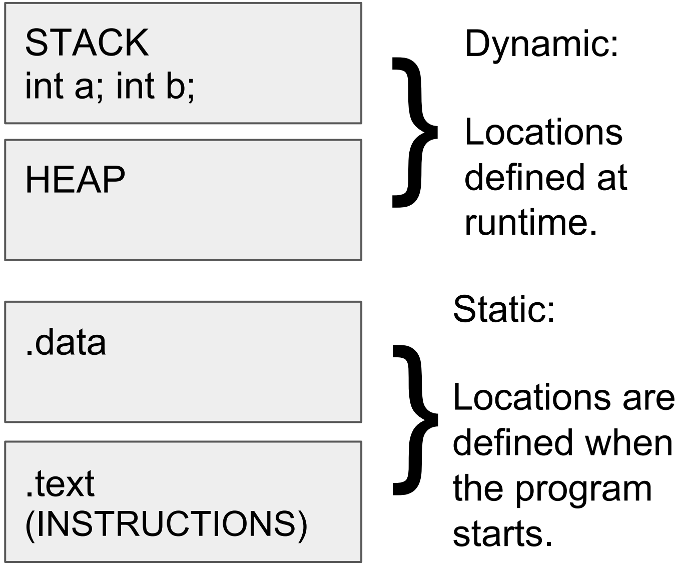
```
class Main {

 public static int x = 5;
 int y = 7;

 public int addNumbers(int a, int b) {
    int sum = a + b;
    return sum;
  }

  public static void main(String[] args) {
       int num1 = 25;
       int num2 = 15;

   // create an object of Main
    Main obj = new Main();
    int result = obj.addNumbers(num1, num2);
    System.out.println("Sum is: " + result);
  }
}
```



STACK
int a; int b;

HEAP

Dynamic:

Locations defined at runtime.

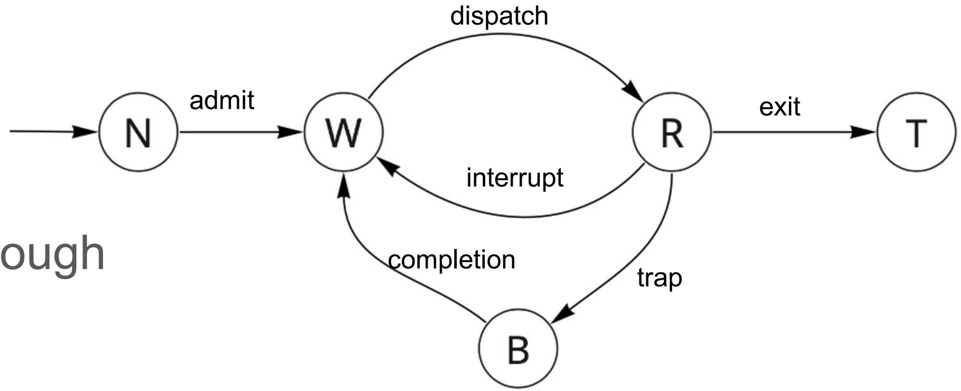
.data

.text
(INSTRUCTIONS)

Static:

Locations are defined when the program starts.

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

Three Address Code (TAC)

- A generic assembly language in which all instructions have at most three addresses
- An address references either
  - a register location
  - a memory location
- Immediate values are stored in a location within memory

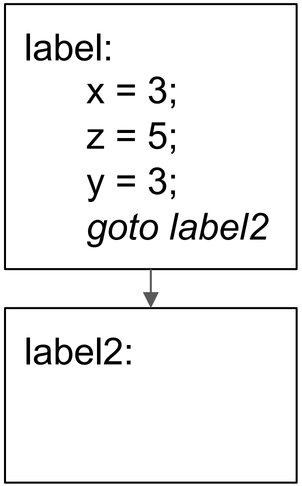
Examples:

1. a = y + x
2. a = y
3. a = x + 2
4. b = d * 2 + y
   - t0 = d * 2
   - t1 = t0 + y
   - b = t1

Assumption: the assembly language is for a register-based machine, with an infinite number of registers.

Basic Blocks

- A number of instructions in which there is
  - a single entry point (via a label), and
  - a single exit point (via a goto)
- All programs can be broken down into a set of basic blocks
- A control flow graph determines which a basic block is executed.
- Standard control flow graphs
  - if-then-else and all other variants (e.g., switch)
  - while, do-while and all other variants
  - for loop and all other variants
  - call-return



```
label:
      x = 3;
      z = 5;
      y = 3;
      goto label2
```

```
label2:
```