

# Register Machines

Lower-level & more verbose than stack machines, but:

- closer to modern CPUs (RISC architecture)
- closer to control-flow graphs
- simpler than stack machine (but register set is finite)

Examples:

[ARM architecture](#)

RISC V: <http://riscv.org/>

Directly Addressable  
RAM  
large - slow even with cache

**A few fast  
registers**

R0,R1,...,R31

# Basic Instructions of Register Machines

$R_i \leftarrow \text{Mem}[R_j]$       load

$\text{Mem}[R_j] \leftarrow R_i$       store

$R_i \leftarrow R_j * R_k$       compute: for an operation \*

Efficient register machine code uses as few loads and stores as possible.

# State Mapped to Register Machine

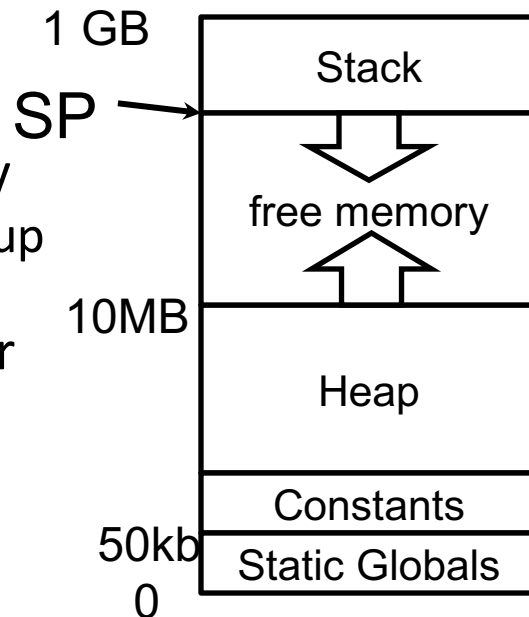
Both dynamically allocated heap and stack expand

Heap is **more general**:

- Can allocate, read/write, deallocate, in any order
- Garbage Collector does deallocation automatically
  - Must be able to find free space among used one, group free blocks into larger ones (compaction),...

Stack is efficient: top of stack pointer (SP) is a register

- allocation is simple: increment, decrement
- to allocate N bytes on stack (**push**):  $SP := SP - N$
- to deallocate N bytes on stack (**pop**):  $SP := SP + N$



Exact picture varies  
depend on hardware,  
OS, language runtime

# WASM vs General Register Machine Code

## Naïve Correct Translation

**WASM:**

`imul.32`

**Register Machine:**

$R1 \leftarrow \text{Mem}[\text{SP}]$

$\text{SP} = \text{SP} + 4$

$R2 \leftarrow \text{Mem}[\text{SP}]$

$R2 \leftarrow R1 * R2$

$\text{Mem}[\text{SP}] \leftarrow R2$

# Register Allocation

# How many variables?

x,y,z,xy,xz,res1

Do we need 7 distinct registers if we wish to avoid load and stores?

x = m[0]      7 variables:  
y = m[1]      x,y,z,xy,yz,xz,res1

xy = x \* y

z = m[2]

yz = y\*z

xz = x\*z

res1 = xy + yz

m[3] = res1 + xz

x = m[0]      can do it with 5 only!

y = m[1]

xy = x \* y

z = m[2]

yz = y\*z

y = x\*z      // reuse y

x = xy + yz      // reuse x

m[3] = x + y