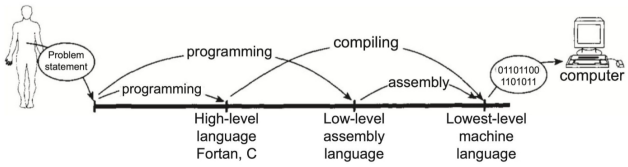
Assembly language ISA Instruction Set Architecture . How hard ⇔ soft

↳ instruction in human readable format (⇔ bin.)



E.

```
gcd:
    beq  $a0, $a1, exit  # if a = b, go to exit
    slt  $v0, $a1, $a0   # is b > a?
    bne  $v0, $0, suba   # if yes, goto suba

    subu $a0, $a0, $a1   # subtract b from a
    b    gcd             # and repeat

suba:
    subu $a1, $a1, $a0   # subtract a from b
    b    gcd             # and repeat

exit:
    move $v0, $a0        # return a
    jr   $ra             # go back to caller
```

labels (go-to)

opcodes operands (data) Comments

MIPS is an RISC (reduced instruction set computer) architecture : simple / regular instructions

32-bit MIPS : word size, register, instruction are 32-bits.

Load-store architecture : only operate on data in register (32 FFs), not mem.

For mem, need to load → store

destination (1st) sources

1. Add	a = b + c;	add $a, $b, $c	a,b,c all registers
Sub	a = b - c;	sub ~	# simplicity :)

Immediate operands (16-bit) : small const. embedded in instruction (immediately available)

E. addi , $t0, $t0, 1 → imm. (range: -32768 ~ 32767)

32-bit const. needs to be loaded to imm. in 2 steps

lui $t1, 0xAAAA → $t1 [AAAA | 0000] (1.)

bitwise OR ori $t1, $t1, 0xBBBB → $t1 [AAAA | BBBB] (2.)

li $t1, 0xAAAABBBB : pseudo-instruction, assembler converts to

Reserved registers

$ prefix

Name	Number	Usage	Preserved across function calls?
$zero	0	Constant zero (can't be changed)	
$at	1	Reserved for assembler	
$v0-$v1	2-3	Function result(s)	
$a0-$a3	4-7	Function argument(s)	
$t0-$t7	8-15	Temporaries	
$s0-$s7	16-23	Saved Temporaries	yes
$t8-$t9	24-25	More temporaries	
$k0-$k1	26-27	Reserved for OS	
$gp	28	Global pointer	yes
$sp	29	Stack pointer	yes
$fp	30	Frame pointer	yes
$ra	31	Return address	yes

(saved temp)

32 register files

Memory Load/store : lbu (byte unsigned), lhu (halfword), ll (linked), lw (word)

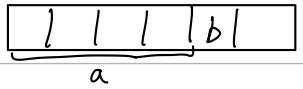
Pointer (address) for every byte E. word : 4 bytes, always +4

Array (E. of int (4 bytes)) : a, a+4, a+8, ...

E. linked list: not cont. add.

addr. = base ptr + imm. offset E. 4($t1) → address = reg[$1] + 4

E. struct { int a; char b; }; (4, 1)



&a = base &b = base + 4

E. increment value in mem.

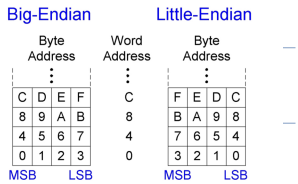
```
lw $s0, 4($t1)    # load 4($t1) to $s0
addi $s0, $s0, 1
sw $s0, 4($t1)    # store
```

Load byte (8b → 32b) lb $t0, 4$(t1) → reg[$t0] = 0xFFFFFFF0 fill MSB

lbu ~~~~ = 0x000000F0 fill 0

Number bytes - Little-endian : byte starts at LS end

Big ~~~~ MS end



Control - branch (conditional E. if, loop) (b) beq (beq, $0, $0, <add> (always =)) bne (≠)

jump (uncond. E. fucn. call/return) j jal (and link) jr (register)

E. jal jumps to label and notes current position (where left off) in $ra