- CPU Time = CPU Clock Cycles per program × Clock Cycle Time

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

- Instruction Count (IC 指令数) Clock cycle Per Instruction (CPI 一个指令clock cycle数量)

Clock Cycles = Instruction Count × Clock Cycles per Instruction (CPI)

CPU Time = Instruction Count × CPI × Clock Cycle Time

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

- Assembly language : not portable
- Instruction Set: all commands a computer understand

Reduced Instruction Set Computer - RISC CPI_{RISC} 小, 更好.

Complex Instruction Set Computer - CISC

- Arithmetic Operations.

add a, b, c # a = b + c.

Design Principle 1: Simplicity favors regularity.

Operands in MIPS Assembly.

- register operands. P 11.

Memory Operand P 17

eg1: g = h + A[8].

```
lw $t0, 32($s3)
add $s1, $s2, $t0
```

RF ← mem

A[8]

32 ⇒ \$s3 + 32 = BA + 8 × 4

eg2: A[12] = h + A[8]

```
lw $t0, 32($s3)
add $t0, $s2, $t0
sw $t0, 48($s3)
```

Line 1: (optional)

```
add $s0, $s1, $s2
lw $s3, 4($s0)
sll $s3, $s3, 2.
```

offset = i × 4.

~~2(\$s0)~~ : (\$s0) + 2 错的

RF → mem

Immediate operands (constant)

addi \$s3, \$s3, 4 < 16 bit 2's compount.

reg reg immediate number

- **The constant Zero.** 不可覆盖

add \$t2, \$s1, \$zero.

- **Logical Operations**

(乘2) Shift left : sll

(除2) Shift right : srl

} sll/srl rd, rt, shamt

destination source 移的位数.

Bitwise AND : and, andi   and \$t0, \$t1, \$t2.   rd

Bitwise OR : or, ori   or \$t0, \$t1, \$t2.

Bitwise NOT : nor   nor \$t0, \$t1, \$zero.

- **load 32-bit constants.**

lui rt, constant   存到 rt 左边 16 bit, 把右 16 bit 变 0.

eg: lui \$s0, 61

ori \$s0, \$s0, 2304   再存到右 16bit.

- **Branch / Jump Operation**   "if" "while"

beq rs, rt, L1   branch if register equal (if ==) 跳到 L1

bne rs, rt, L1   branch if register not equal (if !=). 跳到 L1.

j L1   unconditional jump to L1.

eg: if (i==j) f=g+h;

else f = g-h;   f, g, h, i, j in \$s0, \$s1, \$s2, \$s3, \$s4.

```
bne  $s3, $s4, Else
add  $s0, $s1, $s2.
j    Exit
```

Else: sub \$s0, \$s1, \$s2.

Exit: ...

eg: while (save[i] == k) i+=1; i: \$s3, k: \$s5, BA \$s6.

```
Loop:   sll   $t1, $s3, 2        4×i
        add   $t1, $t1, $s6      BA+4×i
        lw    $t0, 0($t1)
        bne   $t0, $s5, Exit.
        addi  $s3, $s3, 1
        j     Loop
Exit: ...
```

- Conditional Operations.

slt rd, rs, rt (set less than destination - 定 0 或 1)

slti rt, rs, constant (if rs< constant, rt=1 ; else rt=0).

eg: slt \$t0, \$s1, \$s2

bne \$t0, \$zero, L } 若 \$s1 < \$s2, 值行 L

Signed comparison: slt, slti

UnSigned comparison: sltu, sltui

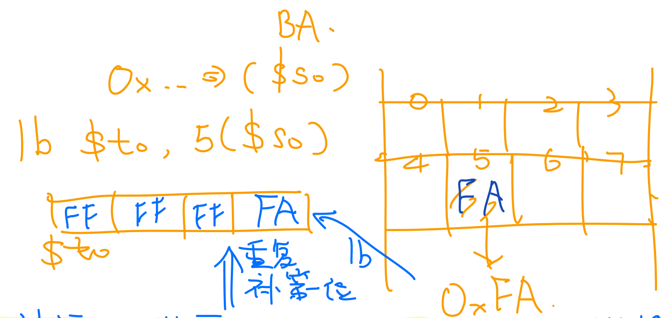
- Byte / Halfword Operations

lb rt, offset (rs) sign extend to 32 bits in rt.

lbu rt, offset (rs) zero extend to 32 bits in rt.

把 rs 中一个 byte 的内容存到 rt, 并补位, offset 可以不是 4 的倍数

lb: R[rt] = {SignExt(M[R[rs] + SignExt Imm](7:0))} 把 FA 给 t0.



16位 即补签; 16是 sign extension

负$_{10}$ ⇒ 二进：先表示正数(4n位)，再逆，再+1
二进 ⇒ 负$_{10}$：先逆，再+1，为该数(补为4n位)

- 正负号 0正 1负

2's complement negative of every bit, then plus one. -3: 3: 0011 → -3: 1100+1 = 1101

若给 1011 则先转 0100 再+1 = 0101 = 5 ∴ 1011 = -5

若给3个8/16进制2's 的数，要先转2进制看正负，然后根据2进制数再转别的

eg: $(F3A8)_{16} = 1111001110101000_2 \Rightarrow 0000\,1100\,0101\,0111 + 1 = 0000110001011000 \Rightarrow -3160_{10}$

若给3个2进制2's的数做减法，先看正负转10进制，再减，结果再转2.

eg: $(10100 - 10101001)_2$
10100 → 01011 + 1 = 01100 = -12
10101001 → 01010111 = -87
→ $(75)_{10} = (01001011)_2$

L3

- jal FunctionLabel : jump and link. (function call)
  jr $ra : jump register (function return).
- Leaf Function : don't call other functions

Function Calling Convention

应用时机：1) immediately before function called
2) In function, but before it starts executing
3) Immediately before function finishes

1) • Pass arguments to $a0 - $a3
   • save registers that should be saved by caller ($a0-$a3)
   • jal
2) • allocate memory of frame's size

- save registers that should be saved by function in frame, before be overwritten $s0-$s7, $fp, $ra
- establish $fp, $fp = $sp + frame's size - 4

3) - place result to $v0, $v1
- restore registers saved by function
- destroy stack frame by moving $sp upword
- jr $ra

* stack 由大到小. 其他由小到大

linker P33.

L4.

- R-format

6	5	5	5	5	6
op	rs	rt	rd	shamt	funct

op + funct 确定 operations of instruction

- I-format

6	5	5	16
op	rs	rt	constant or address

constant: $-2^{15} \sim 2^{15}+1$

address : offset add to base address.

* addi, lw, sw 都是先rt再rs的.

- beq rs, rt, LOOP. 16位是 relative address. (RA).

RA = (LOOP - PC - 4) / 4.

CPU 读到后 RA shift left 2 → +4 + PC → 给 PC

- J-format

6	26
op	address

Target address = PC[31:28] : address × 4 (P13).

从 0x00080014 j到 0x00080000 : 0020000 28位