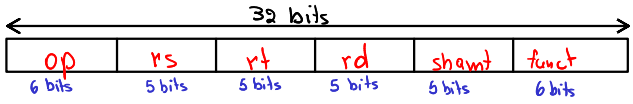
Registradores: são porções de memória na CPU as quais podemos utilizar para realizar operações, no MIPS temos 32 registradores de 32 bits cada. São 5 bits que referenciam os registradores.

nº decimal	Nome reg.	Descrição
0	$zero, $r0	Sempre contém zero
1	$at	Montador
2 e 3	$v0 e $v1	Valores de retorno
4, ..., 7	$a0, ..., $a3	Argumentos de função
8, ..., 15	$t0, ..., $t7	Para cálculos temporários
16, ..., 23	$s0, ..., $s7	Registradores salvos
24 e 25	$t8 e $t9	Mais reg. temporários
26 e 27	$k0 e $k1	Reservados para o Kernel
28	$gp	Apontador de memória global
29	$sp	Ponteiro de pilha
30	$fp	Ponteiro de quadro
31	$ra	Endereço de retorno

Instruções: as instruções no MIPS ocupam 32 bits, no Assembly utilizamos mnemônicos, ao invés dos bits diretamente para representar uma instrução. As instruções possuem campos de larguras prédefinidas, o que é usado em quais instruções depende do formato da instrução.

↳ Tipo-R:

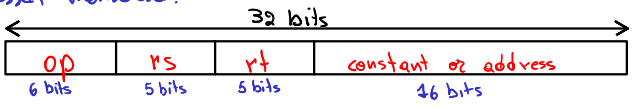
* op: código básico da instrução, opcode.
* rs: reg. do primeiro operando, register source
* rt: reg. do segundo operando, register target
* rd: reg. destino.
* shamt: Shift Amount
* funct: variante de função.



Ex: 000000 10001 10010 01000 00000 10000

add $t0, $s1, $s0 — não usado

↳ Tipo I: tipo imediato, para carregar constantes e para acessar memória.



Acessando Memória:

↳ Utilizamos loads e stores, instruções do tipo-I.

↳ lw $regDestino, deslocamento ($regBase)

* $regDestino = MEM ($regBase + deslocamento)

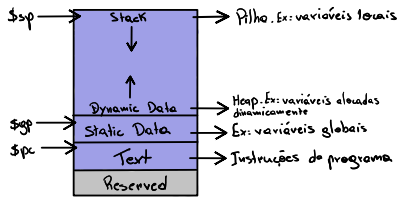
↳ sw $regFonte, deslocamento ($regBase)

* MEM ($regBase + deslocamento) = $regFonte

↳ Ex: lw $t0, 32($s3) # carregue para $t0 o valor armazenado na posição indicada por $s3 deslocada 32 bytes.

Entrada e Saída: usamos syscall, colocamos o código da operação desejada em $v0, o syscall devolve o controle ao S.O., que olha para o $v0 e faz o requisitado.

Convenção da Memória:



Contador de Programa: O processador carrega o endereço apontado pelo reg. PC. A primeira coisa que o processador faz é acrescentar +4 ao PC, já que cada instrução ocupa 4 bytes em no MIPS 32.

Branches: são desvios, instrução utilizada para tomada de decisão (if, while, for), instruções do tipo-I

↳ Ex: beq, $s0, $s1, ENDEREÇO # salte se $s0 == $s1

op rs rt const and address

↳ O endereço efetivo do salto é: PC = PC+4 + ENDEREÇO×4

⚠ ao executar a instrução na linha x o PC está na linha x+1, ainda o endereço é em base 10.

Rótulos: são utilizados para facilitar os desvios, ao invés de utilizar o endereço usamos um label para o endereço, e usamos no branch o label, o montador substitui o rótulo pelo endereço.

Comparações: instruções do tipo-R, atribui a um registrador 1 se a comparação for verdadeiro e 0 caso contrário.

↳ Ex: slt $s0, $s1, $s2 # if($s1 < $s2) $s0 = 1; else $s0 = 0;

Saltos incondicionais

↳ J: jump (salte para o endereço). PC = ENDEREÇO << 2

↳ Instrução do tipo-J.

Ex: J 4000

op endereço base 10

Chamadas de Funções:

① Passagem de parâmetros: usamos os reg. $a0, ..., $a3, caso precisar mais usamos a pilha.

② Transferir controle para a função: usamos jal (jump-and-link), instrução do tipo-J em que faz o jump e salva o endereço da próxima instrução no reg. $ra (return address).

③ Adquirir recursos de armazenamento: salvar na pilha o conteúdo que iremos sobrescrever em $s0, ..., $s7. Ex: vamos utilizar $s0 e $s1, então:

```
addi, $sp, $sp, -8 # desloca o topo da pilha em 8 bytes
sw $s0, 0($sp)
sw $s1, 4($sp)
```

④ Realizar a tarefa: realizamos a tarefa usando $s0 e $s1, e armazenando o(s) valor(es) de retorno em $v0 ou $v1, caso precisar de mais pode ser usado a pilha.

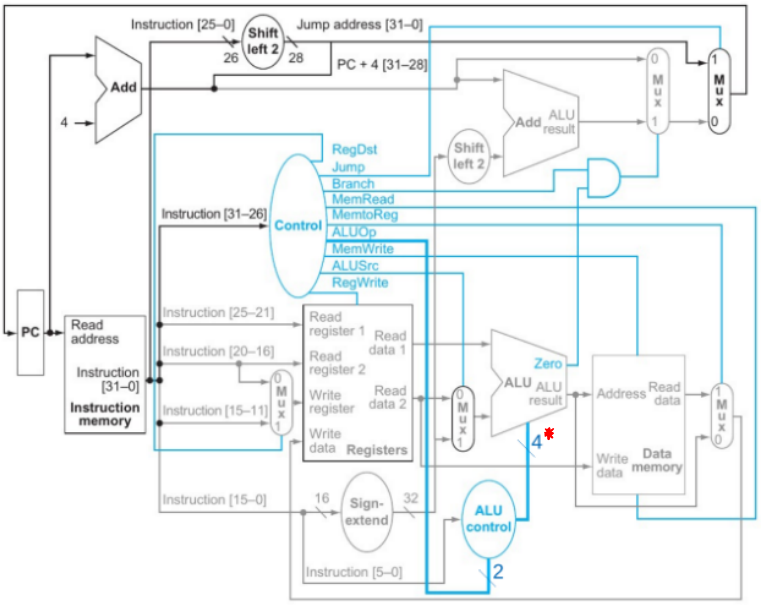
⑤ Liberar recursos e limpar rastros: restauramos os valores salvos na pilha e ajustamos a pilha.

```
lw $s0, 0($sp)
lw $s1, 4($sp)
addi $sp, $sp, 8
```

(VI) Retornamos o controle ao chamador: usamos jr (jump reg.) para voltarmos para o chamador: jr $ra

Funções não folha: devem conter:
(I) condição de parada
(II) Salvar contexto
(III) Realizar a tarefa e adicionar a $v0 e/ou $v1
(IV) Liberar recursos e restros
(V) Retornar para a função chamadora

Montando a CPU:



Nome	Formato	Tipo	Desc
add	rd,rs,rt	R 0/20	rd = rs + rt
sub	rd,rs,rt	R 0/22	rd = rs - rt
addi	rt,rs,imm	I 8	rt = rs + imm(2)
mult	rs,rt	R 0/18	(hi,lo) = rs * rt
div	rs,rt	R 0/1A	lo = rs/rt, hi = rs % rt
mfhi	rd	R 0/10	rd = hi
mflo	rd	R 0/12	rd = lo
and	rd,rs,rt	R 0/24	rd = rs & rt
or	rd,rs,rt	R 0/25	rd = rs | rt
nor	rd,rs,rt	R 0/27	rd = ~(rs | rt)
andi	rt,rs,imm	I C	rt = rs & imm
ori	rt,rs,imm	I D	rt = rs | imm
sll	rd,rt,sh	R 0/0	rd = rt << sh
slr	rd,rt,sh	R 0/0	rd = rt >> sh
lui	rt,imm	I F	rt = imm << 16
lw	rt,imm(rs)	I 23	rt = MEM(rs+imm)
sw	rt, imm(rs)	I 2B	MEM(rs+imm) = rt
la	rx, label	—	Load address
li	rx, imm	—	rx = imm
bge	rx,ry, imm	—	rx ≥ ry → imm
bgt	rx,ry,imm	—	rx > ry → imm
ble	rx,ry,imm	—	rx ≤ ry → imm
blt	rx,ry,imm	—	rx < ry → imm

ALU control lines*	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

- PC: percorre endereços na memória.
- Instruction Memory: lê um endereço na memória e devolve a instrução.
- Registers: banco de reg., contêm os 32 reg. do MIPS, recebe o endereço dos registradores e devolve os dados dos registradores, assim como escreve os dados após alguma operação.
- Data memory: recebe um endereço de memória e devolve o dado ou sobrescreve em memória.
- Sign extend: dado um sinal de 16 bits, gera o seu correspondente em 32 bits, que leva em consideração o seu complemento 2 para gerar.
- ALUop: 00_2 → adição (para lw e sw)
  01_2 → subtração (para beq/bne)
  10_2 → operação definida pelo campo funct (Tipo-R)

Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

Syscall codes

1 - print integer	5 - read integer	10 - exit
2 - print float	6 - read float	11 - print character
3 - print double	7 - read double	12 - read character
4 - print string	8 - read string	