

Instruções para chamada de procedimento

① Descrição para o procedimento

jal procedimento

jump and link

returning address

↑

↳ salva o end. da próxima instrução na reg. de retorno

↳ faz o diret. para o resultado procedimento

② Retorno do chamador

jr \$ra

jmp \$t0

returning

```
# int main() {
    int a = 3; → $s0
    int b = 2; → $s1
    printf("%d\n", media(a,b));
    return 0;
}
```

```
int media(int a, int b) {
    return (a+b)/2;
}

int soma(int a, int b) {
    return a+b;
}
```

↳ Assembly:

main:

li \$s0, 3

li \$s1, 2

Carrega args:

move \$a0, \$s0

move \$a1, \$s1

Chama o procedimento:

jal media

Imprime resultado

move \$a0, \$v0 } imprime
li \$v0, 1 } a média
syscall } (intóire)

média.

li \$v0, 11 } imprime

li \$a0, 10 }

syscall

addi \$sp, \$sp, -4 } salva \$ra na

sw \$ra, 0(\$sp) } pilha

jal soma

smi \$v0, \$v0, 1

lw \$ra, 0(\$sp) } restaura o valor de

addi \$sp, \$sp, 4 } \$ra

jr \$ra

soma:

add \$v0, \$a0, \$a1

jr \$ra

Formato tipo-J

op	endereço
6 bits	26 bits

É o formato das instruções j, jal e jr

Manipulação de memória

① Tabela ASCII

128 caracteres (imprimíveis e não imprimíveis)

L printf("%d\n", 2-'0'); 2

L 'a' - 'A' = 32

② Instruções de acesso à memória

lb / bu \$t0, 0(\$s0) — carrega 1 byte

sb \$t0, 0(\$s0) — escreve 1 byte

③ System calls

↳ 11 - imprime caractere

↳ 12 - le caractere

Aritmética Computacional

Representação numérica no computador

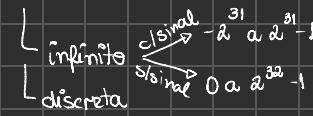
→ Adição e Subtração:

↳ overflow

→ Multiplicação e divisão

→ Representação de ponto flutuante

Ponteiros



Reais

↳ infinito

↳ contínuo

Underflow: não é possível representar números pequenos
epsilon da máquina: (E_mach)

Adição:

$$\begin{array}{r} 1 \ 1 \ 1 \\ + 0 \ 1 \ 0 \\ \hline 1 \ 0 \ 1 = 13_{10} \end{array}$$

Overflow pode acontecer:

* na soma, quando o sinal dos operandos for igual

* na subtração, quando os sinais dos operandos forem diferentes

Não há overflow

- soma, com sinal diferentes
- subtração, com mesmo sinal

Como identificar overflow

addu \$t0, \$t1, \$t2 # addu não lança exceção

xor \$t0, \$t1, \$t2 # se sinal(t1) == sinal(t2)

$$t_0 < 0$$

slt \$t0, \$t0, \$zero

bne \$t0, \$zero, sem overflow

se sinal dos op. forem iguais e o de resultado diferente, haverá um overflow.

xor \$t0, \$t0, \$t1

slt \$t0, \$t0, \$zero

bne \$t0, \$zero, overflow.

Em números sem sinal:

$$\$t_1 + \$t_2 > 2^{32} - 1 = ?$$

addu \$t0, \$t1, \$t2

nor \$t0, \$t2, \$zero # $t_0 = 2^{32} - \$t_2$

slt \$t0, \$t0, \$t1 # $2^{32} - t_2 - 1 < t_0 ?$

bne \$t0, \$zero, overflow

Multiplicação de inteiros

$$\begin{array}{r}
 1000 \quad M \\
 \times 1001 \quad Q \\
 \hline
 1000 \\
 +0000 - \\
 +0000 - \\
 +1000 - - \\
 \hline
 1001000 \quad \text{Produto}
 \end{array}$$

Algoritmo

- $P = 0$ \rightarrow ANDI Q,Q,L
- Se $Q[0] = 1$, então $P = P + M$ SLIM, M, L
- Faça um deslocamento à esquerda em M $\text{SRL P}, Q, L$
- Faça um deslocamento à direita em Q $\text{SLR P}, Q, L$
- Se não for a n-ésima iteração, volte ao passo 2.
 \hookrightarrow qd de bits

Ex:

	1	2	3	4
$M = 1000$	1000	100000	1000000	10000000
$Q = 1001$	1	10	1	0
$P = 0$	1000	1000	1000	1000000

Para executar esse algoritmo, precisamos de

M [32 bits]

Q [32 bits]

P [64 bits]

$$\begin{array}{r}
 1000 \\
 \times 1001 \\
 \hline
 1000
 \end{array}$$

1000		Q
0100		Q
0010		Q
1001		Q

M [32 bits]		Q
X [32 bits]		Q
P [64 bits]		Q

0100 | 1000

~ desloca produto p/direita e
semonda os resultados do mult.
na parte de cima

Algoritmo otimizado

- $P[63 \dots 32] = 0$ e $P[31 \dots 0] = Q$
- Se $P[0] = 1$, então $P[63 \dots 32] += M$
- Desloca P uma casa à direita
- Se não for a 32^ª iteração, volte ao passo 2.

\hookrightarrow PROVA

Ex: Multiplicar $2 \times 3 = 0010 \times 0011$

It:	Etapa	Produto
0	Salvos iniciais	0000 0011
	$P[7 \dots 4] += M$	0000 0011
1	$\text{SRL } P, P, 1$ $P[7 \dots 4] += M$	0001 1000
2	$\text{SRL } P, P, 1$ Montém P	0001 1000
3	$\text{SRL } P, P, 1$ Montém P	0000 1100
4	$\text{SRL } P, P, 1$	0000 0110

Obs: o algoritmo apresentado funciona apenas para inteiros sem sinal.
Para multiplicarmos números com sinal, basta considerar os quocientes para positivo, executar o algoritmo e aplicar regra de sinal no final.

Instruções do assembly MIPS

mult $r_1, r_2 \Rightarrow$ calcula a multiplicação e salva o resultado nos registradores específicos $r_1 \dots r_2$.

hi	lo
32 bits	32 bits

⇒ Para acessar o conteúdo desses registradores.

mfhi reg ⇒ copia o conteúdo de hi e lo para o registrador reg.

mflo from

mfhi reg ⇒ copia o conteúdo de registrador reg para hi e lo

mflo reg

move to

Obs: mul rd,rs,rt executa rs * rt e salva em rd. Funciona bem se rs e

rt tiverem, no máximo, 16 bits