

Fundamentos de Arquitetura de Computadores

Tiago Alves

Faculdade UnB Gama
Universidade de Brasília



Revisão

Componentes:

- Carrega words mas endereça bytes
- Aritmética somente entre registradores ou imediato
- Tamanho padrão 32 bits

Instrução:

```
add $s1, $s2, $s3
sub $s1, $s2, $s3
addi $s1,$s2,imm
mulu $s1,$s2,imm
lw $s1, imm($s2)
sw $s1, imm($s2)
```

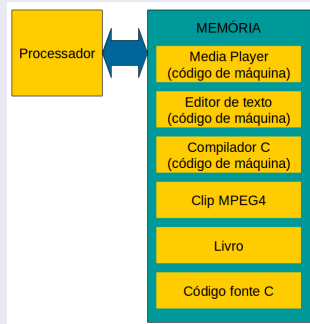
Significado:

```
$s1 = $s2 + $s3
$s1 = $s2 - $s3
$s1=$s2+imm
$s1=$s2 x imm # (pseudo-instrução)
$s1 = Memory[$s2+imm]
Memory[$s2+imm] = $s1
```



Programa armazenado

- Instruções são compostas por bits.
- Programas são armazenados na memória para serem lidos da mesma forma que os dados.



Programa armazenado

Ciclos de **busca e execução**:

- Instruções são **buscadas** e colocadas num registrador especial (IR: Instruction Register).
- Bits neste registrador “**controlam**” as ações subsequentes necessárias a execução da instrução.
- Busca a próxima instrução e continua...



Formato de Instruções MIPS

Instruções, assim como registradores e words de dados, também têm 32 bits de comprimento

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

Campos (fields):

- op: operação básica da instrução: opcode
- rs: primeiro registrador de operando origem (source)
- rt: segundo registrador de operando origem (source)
- rd: registrador de operando destino: resultado (destination)
- shamt: shift amount
- funct: variação da operação: function code



Formato de Instruções MIPS

Exemplo: add \$t0, \$s1, \$s2

- registradores são identificados por seus números (vide tabelas): \$t0=8, \$s1=17, \$s2=18

Formato de Instrução Tipo-R (add: op=0 funct=32)

Campo	op	rs	rt	rd	shamt	funct
decimal	0	17	18	8	0	32
binário	000000	10001	10010	01000	00000	100000
Tamanho	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits



Formato de Instruções MIPS

O que acontece quando uma instrução necessita de campos maiores?

- Usa o imediato.

```
addi $t0,$t1,Imm  
lw $t0,Imm($t1)
```



Formato de Instruções MIPS

Quarto Princípio de Projeto:

- **Um bom projeto exige bons compromissos.**

Compromisso do MIPS: Manter todas as instruções com o mesmo comprimento de bits, independente do tipo de formato da instrução!



Formato de Instruções MIPS

Novo tipo de formato de instrução para instruções com dados Imediatos.

- Exemplo: `lw $t0, 32($s3)`

O comprimento do quarto campo do Tipo-I é a soma dos comprimentos dos últimos três campos do Tipo-R.

Formato de Instrução Tipo-I

op	rs	rt	Imm
35	19	8	32
100011	10011	01000	00000000000100000
6 bits	5 bits	5 bits	16 bits



Compilação manual...

- Suponha que \$t1 tenha o endereço base de A e que \$s2 corresponda a h, traduza a seguinte linha em C para código de máquina MIPS:

$A[300] = h + A[300];$

- Primeiro, temos que o código em assembly correspondente é:

```
lw $t0, 1200($t1) # $t0 = A[300]
add $t0, $s2, $t0 # $t0 = h + A[300]
sw $t0, 1200($t1) # A[300] = h + A[300]
```

- Qual o código em linguagem de máquina destas 3 instruções?



Compilação manual...

	op	rs	rt	rd	endereço /shamt	funct
lw \$t0,1200(\$t1)	35	9	8	1200		
add \$t0, \$s2, \$t0	0	18	8	8	0	32
sw \$t0, 1200(\$t1)	43	9	8	1200		

op	rs	rt	rd	endereço /shamt	funct
100011	01001	01000	0000 0100 1011 0000		
000000	10010	01000	01000	00000	100000
101011	01001	01000	0000 0100 1011 0000		

Na Memória: 0x00400000 8D 28 04 B0
0x00400004 02 48 40 20
0x00400008 AD 28 04 B0



Operações Lógicas

Operação	C	Instrução MIPS	Opcode / Funct
Shift à esquerda	<<	sll	0 / 0
Shift à direita	>>	srl	0 / 2
AND	&	and andi	0 / 36 12
OR		or ori	0 / 37 13
XOR	^	xor xori	0 / 38 14
NOR		nor	0 / 39

E as outras? Ex.: not?



Operações Lógicas

NOT

- Realizado utilizando o NOR.
- Utiliza-se o \$zero (Hardwired) de preferência!
- Expressão Booleana: $A \text{ NOR } B = \text{NOT } (A \text{ OR } B)$.
- Se $B = 0$: $\text{NOT } (A \text{ OR } 0) = \text{NOT}(A)$.



Operações Lógicas

sll e srl (deslocamentos lógicos ou *logical shifts*)

- Instrução do Tipo-R.
- Utiliza o campo shamt (*shift amount*)
 - Quantidade de Deslocamento (de acordo com a direção!)
 - Ex.: sll \$t2, \$s0, 4 # \$t2 = \$s0 << 4 bits

op	rs	rt	rd	shamt	funct
0	0	16	10	4	0



Operações Lógicas

AND e OR

- Mesma atribuição do AND e OR booleanos
- Realizados Bit-a-bit

AND: Operação de “mascaramento” (ocultar bits) e “chave lógica”.

OR: “Junção de sinais lógicos”

MIPS: Existem os equivalentes para AND e OR com Imediatos (`ori` e `andi`) .



Controle de Fluxo

Desvio **Incondicional**

- Registrador Especial PC (Program Counter): indica qual o endereço da próxima instrução a ser buscada na memória
- Instruções MIPS

```
jr  $t0      # Jump Register: PC=[ $\$t0$ ]           Obs.: Tipo-R !
j   Label    # Jump Label: PC=Label
jal Label    # Jump and Link:  $\$ra=PC+4$ ; PC=Label
```

Formato de instrução Tipo-J: Ex.: j 1200

op	Endereço
2	1200
000010	000000000000000010010110000
6 bits	26 bits

Controle de Fluxo

Desvio Condicional:

```
bne $t0, $t1, Label    # Branch if Not Equal: $t0!= $t1 ? PC=Label
beq $t0, $t1, Label    # Branch if Equal: $t0== $t1 ? PC=Label
```

C:

```
if (i!=j)
    h=i+j;
else
    h=i-j;
```

Assembly:

```
beq $s4, $s5, Label1
add $s3, $s4, $s5
j Label2
Label1: sub $s3, $s4, $s5
Label2: ...
```

Exercício: Implementar um Loop: `for(i=0;i!=10;i++) {...}`



Comparações

No MIPS, são implementadas as comparações: `==` e `!=` Como implementar: `<`, `>`, `<=`, `>=` ?

Instrução MIPS: Set on Less Than

- `slt $t0,$t1,$t2` # `$t0=1` se `$t1<$t2`; `$t0=0` caso contrário
- `slti $t0,$t1,Imm` # `$t0=1` se `$t1<Imm`; `$t0=0` caso contrário
- Apenas com estas instruções podemos montar várias estruturas de controle.
- Ao montador, é reservado o registrador `$1 ($at)` para essa tarefa

Ex.: Construa a pseudo-instrução Branch If Less Than : `blt $t0,$t1,Label`

```
slt $at, $t0, $t1
bne $at, $zero, Label
```



Constantes

Constantes pequenas são usadas muito frequentemente (50% dos operandos).

```
A = A + 5;  
B = B + 1;  
C = C - 18;
```

Como tratar constantes no MIPS?

- coloque constantes típicas na memória e carregue-as.
- crie registradores hardwired (como \$zero) para constantes como um (1).

Instruções com imediato MIPS:

```
addi $29, $29, 4  
slti $8, $18, 10  
andi $29, $29, 6  
ori $29, $29, 4;
```

Ex.: Implemente a pseudo instrução: `li $t0,Imm #(Load Immediate)`

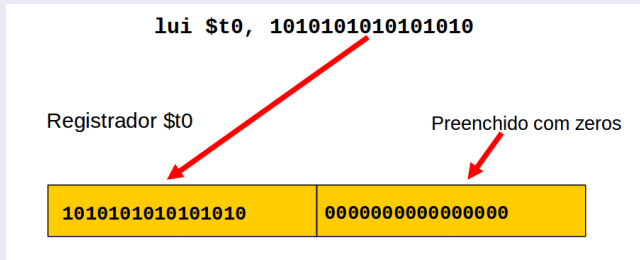
Princípio de Projeto

Agilizar o caso mais comum!



Constantes (maiores)

Para carregar uma constante de 32 bits num registrador são necessárias duas instruções. Nova instrução: load upper immediate

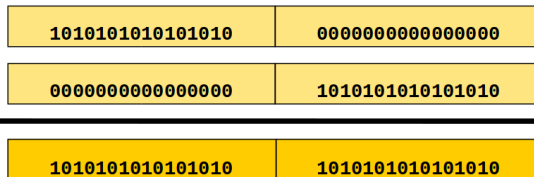


Constantes (maiores)

Carga dos bits menos significativos...

```
ori $t0, $t0, 1010101010101010
```

ori



Linguagem Assembly vs. Linguagem de Máquina

O assembly fornece uma representação simbólica conveniente

- muito mais fácil do que escrever números binários
- por exemplo, destino primeiro
- Pode-se usar Labels em vez de endereços numéricos

A linguagem de máquina é realidade subjacente

- Por exemplo, o destino não é mais o primeiro
- Labels são convertidos em números apropriados

O assembly pode fornecer “pseudo-instruções”

- por exemplo, `move $t0, $t1` existe apenas no assembly
- pode ser implementada usando `add $t0,$t1,$zero`

Ao considerar o desempenho, você deve contar as instruções reais.



Outras questões

Abordadas no Livro e Apêndices do Livro texto:

- suporte para procedimentos;
- linkers, carregadores, layout da memória;
- pilhas, frames, recursão;
- manipulação de strings e ponteiros;
- interrupções e exceções;
- chamadas de sistema e convenções.



Resumo...

Instruções simples, todas de 32 bits

Bastante estruturada

Somente três formatos de instruções

R	op	rs	rt	rd	shamt	funct
I	op	rs	rt	16 bits imediato		
J	op	26 bits endereço				

Contar com o compilador para obter desempenho

Auxiliar o compilador onde for possível

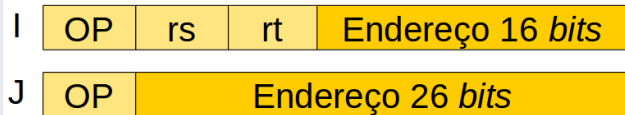


Endereços (do Contador de Programa PC) em desvios

Instruções

```
bne $t4, $t5, Label      # Próxima instrução em Label se $t4 != $t5
beq $t4, $t5, Label      # Próxima instrução em Label se $t4 == $t5
j Label # Próxima instrução em Label
jal Label # $ra=PC+4; Próxima Instrução em Label
```

Formatos



Endereços não têm 32 bits!



Linguagem Assembly vs. Linguagem de Máquina

Pode-se especificar um registrador (como lw e sw) e adicioná-lo ao endereço (Endereço Relativo)

- utilizar Instruction Address Register (PC = program counter)
- maioria dos desvios são locais (Princípio da Localidade)

Instruções tipo-J, j e jal , utilizam os 4 bits de alta ordem do PC e concatenam ao endereço do (Label (26)<<2)(28) totalizando os 32 bits:

- limites de endereço de 256 MB (64M instruções).
- O montador e o linker precisam cuidar disso!



Endereços (do Contador de Programa PC) em desvios

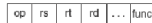
1. Endereçamento imediato



Ex.:

`addi $t0,$t1,Imm`

2. Endereçamento em registrador

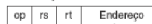


Registadores
Registrador

`add $t0,$t1,$t2`

`jr $t0`

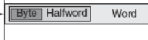
3. Endereçamento de base



Registrador

+

Memória

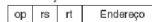


`lw $t0,Imm($t1)`

`lhu $t0,Imm($t1)`

`lbu $t0,Imm($t1)`

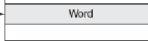
4. Endereçamento relativo ao PC



PC

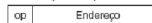
+

Memória



`beq $t0,$t1,Label`

5. Endereçamento pseudodireto



PC

+

Memória



`j Label`

`jal Label`

Endereços (do Contador de Programa PC) em desvios

Exemplo: `while(save[i]==k) i++;`

Loop: `sll $t1,$s3,2`
`add $t1,$t1,$s6`
`lw $t0,0($t1)`
`bne $t0,$s5, Exit`
`addi $s3,$s3,1`
`j Loop`

Exit:

80000	0	0	19	9	2	0
80004	0	9	22	9	0	32
80008	35	9	8	0		
80012	5	8	21	2		
80016	8	19	19	1		
80020	2	20000				
80024	...					



Endereços (do Contador de Programa PC) em desvios

Exemplo: `while(save[i]==k) i++;`

- MIPS: Possuem endereços em bytes, diferenciando em 4 bytes (word);
- A instrução `bne` acrescenta 2 words (8 bytes) na instrução seguinte, especificando o endereço de desvio ($8 + 80016$) e não em relação à instrução atual ($12 + 80012$), ou ao endereço completo/absoluto (80024).
- O jump utiliza o endereço completo ($20000 \times 4 = 80000$)

Loop: `sll $t1,$s3,2`
`add $t1,$t1,$s6`
`lw $t0,0($t1)`
`bne $t0,$s5, Exit`
`addi $s3,$s3,1`
`j Loop`

Exit:

80000	0	0	19	9	2	0
80004	0	9	22	9	0	32
80008	35	9	8	0		
80012	5	8	21	2		
80016	8	19	19	1		
80020	2	20000				
80024	...					

