Laboratório 2 Comparações, *Opcodes*, Pseudoinstruções

Comparações

Além do comando bne, podemos usar o seu inverso, beq (*branch if equal*). Outros saltos condicionais no assembly MIPS puro fazem comparação com zero (exemplo: `bgtz $reg,destino` – *branch if greater than zero*, salta se maior do que zero).

Para fazer comparações entre dois valores, somos obrigados a utilizar `slt $regD, $reg1,$reg2` *(set if less than*) e `slti $regD,$reg1,constante` *(set if less than immediate*). Essa instrução atribui 1 a regD caso reg1<reg2 (reg1<constante) ou 0 caso contrário.

A razão para isso é que uma operação como `bgt $reg1,$reg2,destino` *(branch if greater than)* iria ser relativamente lenta, possivelmente reduzindo o clock máximo atingível pelo processador; é preferível ter duas instruções rápidas ao invés disso.

Para a conveniência do programador, o conjunto de operações estendidas inclui operações não nativas como bgt, blt e outras; são as pseudoinstruções.

▶ Usando *apenas instruções nativas* do assembly MIPS, faça um programa que classifica $s1 e $s2 em seis diferentes possibilidades, indicadas por $s3 ao final, de acordo com o seguinte:

$s1	$s2	$s3
$s1>=0	$s2>32	1
	$s2==32	2
	$s2<32	3

$s1	$s2	$s3
$s1<0	$s2>32	4
	$s2==32	5
	$s2<32	6

Para saber quais são as operações nativas, consulte o Help do MARS (aba MIPS – *Basic Instructions*).

Código de Máquina

O montador traduz a linguagem *assembly* para o chamado código de máquina, uma sequência de comandos em binário que o processador compreende. Cada instrução é associada a um número, e cada processador possui seu conjunto de instruções possíveis e regras de formação.

Exemplo: a instrução `add $s0,$s1,$s2` corresponde ao número de 32 bits 0x02328020.

O MIPS possui apenas três formatos: R (*register*), I (*immediate*) e J *(jump).* As instruções possuem sempre 32 bits e os campos são regulares.

Regras de Codificação

O formato R dita a codificação das instruções que fazem operações exclusivamente entre registradores. Abaixo, os números entre parênteses indicam a quantidade de bits dos campos.

Formato R	000000 (6)	rs (5)	rt (5)	rd (5)	sa (5)	function (6)

O campo *sa* ou *shamt* significa *shift amount* e é usado para instruções de deslocamento de bits, que não serão vistas. O campo *function* ou *funct* definirá a operação específica. Os seis zeros iniciais são o campo *opcode* (*operation code*) e indicam se tratar de uma instrução tipo R.

Exemplo: a instrução `sub $t6,$s5,$t9` faz `t6←s5−t9`, e é codificada como 0x02B97022*:*

000000	10101	11001	01110	00000	100010

Para entender o mecanismo, consultamos a tabela ao final da folha: *sub* tem *function* 100010_2, $t6 (rd) é $14 ($01110_2$), $s5 (rs) é $21 ($10101_2$) e $t9 (rt) é $25 ($11001_2$). *Atenção para a ordem inversa do rd!*

O formato J é usado apenas para as instruções de salto *j* (jump), que realiza um salto incondicional para o label especificado, e a *jal* (chamada de funções), e será visto depois.

O formato I é para instruções que incluem uma constante de 16 bits, dita *imediata* (pois está

embutida na instrução), compreendendo *addi*, *branches* e instruções de memória.

Formato I	opcode (6)	rs (5)	rt (5)	constante imediata (16)

Como exemplo, a instrução `addi $s1,$zero,0x45` vai gerar o código de máquina 0x20110045, sendo rs=$zero e rt=$s1. A codificação dos *branches* será vista depois.

Pseudoinstruções

Para nossa conveniência, o montador reconhece extensões simples da linguagem assembly e as traduz para o conjunto básico. Abaixo estão listadas algumas delas, sendo *cte* uma constante:

Pseudoinstrução	Significado
li $reg,cte	addi $reg,$zero,cte
move $reg1,$reg2	add $reg1,$zero,$reg2
beq $reg,cte,label	addi $at,$zero,cte beq $reg,$at,label

Pseudoinstrução	Significado
bgt $reg,cte,label	addi $at,$zero,cte slt $at,$at,$reg bne $at,$zero,label
mulo $reg1,$reg2,$reg3	mult $reg2,$reg3 mflo $reg1 *[+5 instruções para estouro]*

Para executar algumas pseudoinstruções, o montador precisa de um registrador auxiliar de seu uso exclusivo, o *$at*, que não deve ser usado explicitamente pelo programador.

Acesso à Memória

A instrução `lw $reg1,cte($reg2)` *(load word)* irá ler 32 bits do endereço de memória dado por $reg2+cte e vai armazenar esse valor em $reg1. A instrução `sw $reg1,cte($reg2)` *(store word)* irá escrever o valor de $reg1 no endereço de memória dado por $reg2+cte.

Exemplos: `lw $s4,0x400($zero)` carrega para $s4 o valor presente no endereço 0x400; `sw $t2,8($s2)` armazena o valor de $t2 no endereço 8+$s2.

Deve-se utilizar a área de memória convencionada para RAM. No MARS, podemos usar a partir do endereço 0x1000 0000 ou 0x1001 0000. Note que a constante `cte` tem apenas 16 bits (!), ou seja: *não temos* uma instrução nativa como *sw $t6,0x12345678($zero)*.

Importante: cada endereço de memória armazena 8 bits (1 byte), o que é uma convenção quase universal. Portanto, *cada dado de 32 bits ocupa 4 endereços de memória*.

Exercícios (exercícios extras em outro arquivo – não deixem de fazer os ★!)

Construa e teste programas que acessam a memória, colocando números para teste através da janela *Data Segment* (a do meio) na tela de *debug* do MARS.

1. Ordene 20 números inteiros.
2. Transfira 400 words de 32 bits a partir do endereço 0x1000 0100 para a região de memória iniciada em 0x1000 1100.
3. **[Entregar na próxima aula]** Idem, mas invertendo a ordem dos dados no destino.
4. Calcule a soma de todos os dados entre 0x1000 0000 e 0x1000 FFFC (inclusive).
5. Armazene uma sequência crescente de 1000 valores na memória a partir do endereço 0x1000 1234.

Convenções de Registradores

$0	$zero
$1	$at
$2 - $3	$v0 - $v1
$4 - $7	$a0 - $a3
$8 - $15	$t0 - $t7
$16 - $23	$s0 - $s7
$24 - $25	$t8 - $t9
$26 - $27	$k0 - $k1
$28 - $29	$gp - $sp
$30 - $31	$fp - $ra

Opcodes – Formato I

Instruction		Opcode
addi	rt, rs, immediate	001000
andi	rt, rs, immediate	001100
beq	rs,rt,label	000100
bne	rs, rt, label	000101
lui	rt,immediate	001111
lw	rt, immediate(rs)	100011
ori	rt,rs,immediate	001101
sw	rt, immediate(rs)	101011

Valores do campo *function* para instruções formato R (o campo *opcode* é fixo em 000000_2 para todas estas instruções)

Instruction		Function
add	rd, rs, rt	100000
addu	rd, rs, rt	100001
and	rd, rs, rt	100100
div	rs, rt	011010
jr	rs	001000
sub	rd, rs, rt	100010
syscall		001100