Laboratório 3 Utilidades, Representação de Constantes

Chamadas de Sistema

O assembly MIPS possui incorporada a instrução `syscall`, projetada para acessar as funções básicas do sistema (no PC, estaríamos falando das funções da BIOS), tais como acesso a tela, teclado e arquivos. O MARS simula várias funções úteis com *syscalls*. Por exemplo, o trecho a seguir lê um inteiro do teclado e o coloca em $t0.

```
li $v0,5        # v0 é carregado para executar a chamada 5
syscall         # faz a chamada (lê o inteiro da interface)
move $t0,$v0    # o inteiro foi colocado em v0; copiamos em t0
```

Veja a lista completa de chamadas de sistema na ajuda do MARS, incluindo um exemplo complexo.

Constantes String e Lista de Dados

Para inserirmos constantes no nosso programa de forma elegante, utilizamos *diretivas assembly*, que são código que não é compilado para instruções MIPS.

- `.data` indica que as linhas a seguir são apenas dados (constantes ou espaço reservado);
- `.text` indica as linhas a seguir são instruções MIPS;
- `.asciiz` serve para inserir uma string terminada em zero (caracteres de *um byte cada*);
- `.word` inclui uma lista de valores de 32 bits, a partir do próximo endereço múltiplo de 4.

Por exemplo:

```
        .data                       # início segmento (trecho) dados
frase:.asciiz "Frase a imprimir!" # \0 inserido automaticamente
dados:.word 34,-12,0,0x5FFE0D23   # 4 words de 32 bits

        .text           # início de segmento de programa
        la $s1,dados    # carrega o endereço dos dados em $s1
        lw $s2,12($s1) # carrega da memória o valor 0x5FFE0D23 em $s2
        la $a0,frase
        li $v0,4
        syscall         # imprime a frase na tela
```

O comando `la` carrega o endereço de um label no registrador indicado.

▶	Faça um programa que obtém vinte números do teclado e os imprime na tela na ordem inversa à de entrada. Pode usar a diretiva `.space` se quiser, mas `.word` serve também. **Entregar semana que vem.**

Tamanho de Operandos – Constantes de 32 bits

Uma vez que as instruções têm no máximo 32 bits e às vezes precisamos trabalhar com constantes deste tamanho, um artifício é usado: o montador quebra estas constantes ao meio, utilizando duas instruções para efetivar a carga (*load upper immediate* e *or immediate*):

```
 li $s0,0x12345678      =>   lui $s0,0x1234    (carrega parte alta)
(instrução original)         ori $s0,$s0,0x5678 (seta parte baixa)
```

Para labels, é o montador que decide onde colocá-los na memória (etapa de *linkagem* ou linkedição), portanto ele sabe o valor final exato, algo difícil para o programador fazer sozinho.

Codificação de Saltos

Temos três tipos de saltos no MIPS: com endereço absoluto (*j* e *jal*), com endereço relativo (todos os *branches*) e destino via registrador (*jr*). Notamos que o endereço de destino de um desvio deve ter, obrigatoriamente, 32 bits e não temos isso disponível na instrução para um operando.

Os *branches* são instruções do tipo I, o que nos dá 16 bits para especificar o endereço. Simplesmente iremos *somar* o valor imediato *sinalizado* ao valor do PC (Contador de Programa). Forçando que todas as instruções estejam em endereços múltiplos de 4 (dados alinhados em 32

bits, que é o tamanho das instruções), podemos simplesmente supor os dois bits LSB do destino como 00_2, totalizando 18 bits.

Já com endereço absoluto usamos o formato J, visto a seguir, simplesmente saltando para o endereço especificado na constante. Infelizmente, temos 26 bits disponíveis; somados aos dois zeros LSB, ficamos com 28 bits para indicar o destino do salto. Os últimos 4 bits (MSB) são copiados do valor atual do PC. O opcode da instrução *j* é 0x02.

Formato J	opcode (6)	endereço destino (26)

Como exemplo, a instrução `addi $s1,$zero,0x45` vai gerar o código de máquina 0x00110045,

Se precisarmos de um salto que cubra necessariamente toda a memória, deveremos carregar o seu destino num registrador de 32 bits e usar a instrução *jr* (jump register), que saltará para o valor especificado pelo registrador em questão.

Uma última complicação: o PC é incrementado logo após a instrução ser lida (*opcode fetch*) e antes dela ser executada (*instruction decode* e *instruction execute*), portanto o valor de referência para ser somado aos *branches* (ou tomados os 4 bits MSB para *jumps*) é *PC+4*.

Teste no MARS o programa abaixo e explique *cada bit* da codificação binária resultante:

```
start:    j final
          beq $zero,$zero,final
          la $t6,final
          jr $t6
final:    bne $0,$0,final
```

Assuma que o endereço de `start` é 0x0040 0000. A codificação de *jr* está no Lab #2. **Mostrar/entregar até semana que vem.**

Números Sinalizados

Podemos assumir, para todos os efeitos, que todos os números que lidamos no MIPS são sinalizados, o que significa dizer que seus valores são representados em complemento de 2.

Obviamente, é preciso fazer extensão de sinal: `addi $s0,$zero,-1` deve carregar 0xFFFF FFFF em $s0 (-1 em 32 bits), mas a constante de 16 bits é 0xFFFF; a "extensão de sinal" é apenas repetir o bit MSB para todos os bits à esquerda.

Deve-se notar que a instrução *addiu* (*add immediate unsigned*) na verdade não quer dizer que o número não é sinalizado; apenas está indicando que o processador não deve se preocupar caso haja um estouro de 32 bits na soma. A constante imediata terá seu sinal estendido.

Exercícios (entregar os ★ até semana que vem)

1. ★ Percebe-se que temos limites para o destino de saltos, dados conforme a instrução usada. Quais são estes limites, ou seja, quão longe pode estar nosso destino em relação à instrução de salto? Exprima este valor tanto em *endereços* quanto em *número de instruções*.
2. Prove que estender o sinal é equivalente a copiar o bit MSB para todos os bits à esquerda.
3. Um assim chamado *modo de endereçamento* indica uma maneira possível para o processador acessar a memória. No MIPS, temos endereçamento imediato (constantes, como em addi), endereçamento de registradores (como em add) e o indexado de lw e sw.
   É usual termos, em microprocessadores simples, o modo direto (colocamos o endereço a acessar como constante na instrução), o indireto simples (o endereço está num registrador) e o indexado (uma constante é somada a um valor indireto).
   Tanto o x86 quanto o ARM possuem muitos outros modos. Cite vantagens e desvantagens.
4. ★ No 8051, é o programador que decide se o número de 8 bits é sinalizado ou não. Dada a conta: $11001110_2+11000111_2$ com resultado num registrador de 8 bits, quais são os valores decimais envolvidos? Houve estouro da capacidade de 8 bits ou não?
5. ★ O comando addiu $s0,$zero,0x8000 é interpretado como uma pseudoinstrução. Por quê?
6. Faça um programa que descobre e imprime *x* tal que o fatorial de *x* estoura 31 bits.