Organização e Arquitetura de Computadores I

Linguagem de Montagem

Operações Lógicas

- Embora os primeiros computadores se concentrassem em *words* completas, logo ficou claro que era útil atuar sobre campos de bits dentro de uma *word* ou até mesmo sobre bits individuais.

Operadores em C e Java e sua instruções MIPS correspondentes

Operações Lógicas	Operadores C	Operadores Java	Instruções MIPS
shift à esquerda	<<	<<	Sll
shift à direita	>>	>>>	Srl
AND bit a bit	&	&	and, andi
OR bit a bit	|	|	or, ori
NOT bit a bit	~	~	nor

Operações Lógicas

- A primeira classe dessas operações é chamada de *shifts* (deslocamentos). Elas movem todos os bits de uma word para a esquerda ou direita, preenchendo os bits que ficaram vazios com 0s.
  - Exemplo:
    - Se o registrador $s0 tivesse o seguinte valor:

      0000 0000 0000 0000 0000 0000 0000 1001 = 9
    - Executando “sll $t2,$s0,4”, que desloca 4 bits à esquerda, temos:

      0000 0000 0000 0000 0000 0000 1001 0000 = 144

Operações Lógicas

- O complemento de um deslocamento à esquerda é um deslocamento à direita. Os nomes reais das duas instruções de deslocamento no MIPS são ***s****hift* ***l****eft* ***l****ogical* (sll) e ***s****hift* ***r****ight* ***l****ogical* (srl).

srl $t2, $s0, 4

op = 0x00

funct = 0x02

shamt = 4

rt = $s0 = 16

rd = $t2 = 10

op	*rs*	*rt*	*rd*	*shamt*	*funct*
000000	00000	10000	01010	00100	000010

Operações Lógicas

- Exemplo:
  - Supondo que o registrador $t2 tenha:

0000 0000 0000 0000 0000 0000 1101 0000 0000

  - Supondo que o registrador $t1 tenha:

0000 0000 0000 0000 0000 0011 1100 0000 0000

  - Executando a instrução "**and $t0,$t1,$t2**", resultará em:

0000 0000 0000 0000 0000 0000 1100 0000 0000

Operações Lógicas

- Exemplo:
  - Supondo que o registrador $t2 tenha:

0000 0000 0000 0000 0000 0000 1101 0000 0000

  - Supondo que o registrador $t1 tenha:

0000 0000 0000 0000 0000 0011 1100 0000 0000

  - A Execução da instrução "**or $t0,$t1,$t2**" resultará em:

0000 0000 0000 0000 0000 0011 1101 0000 0000

Operações Lógicas

- Exemplo:
  - Supondo que o registrador $t1 tenha:

0000 0000 0000 0000 0000 0011 1100 0000 0000

  - Supondo que o registrador $t3 tenha:

0000 0000 0000 0000 0000 0000 0000 0000 0000

  - A Execução da instrução "**nor $t0, $t1, $t3**" resultará em:

1111 1111 1111 1111 1111 1100 0011 1111 1111

Exercício de Fixação

- Refaça o trecho de código em C usando deslocamentos quando a multiplicação for necessária:
  - Dado o código abaixo:

g = h + A[i];

  - Suponha que A é um array de 100 elementos cujo endereço-base está no registrador $s3 e que o compilador associa as variáveis g, h e i aos registradores $s1, $s2 e $s4. Qual o código gerado para o MIPS, correspondente ao comando C acima?

Instruções para Tomada de Decisões

- A tomada de decisão normalmente é representada nas linguagens de programação usando a instrução *if*, às vezes combinadas com instruções *go to* e rótulos (*labels*).
- O assembler do MIPS inclui duas instruções para tomada de decisões, semelhantes a uma instrução *if* com um *go to*.
- Essas instruções são chamadas de instruções de desvio condicional.

Instruções para Tomada de Decisões

- A primeira instrução significa ir até a instrução rotulada por “L1” se o valor no “registrador1” **for igual** ao valor no “registrador2”. O mnemônico beq significa ***b****ranch if* ***eq****ual* (desviar se igual).

beq registrador1, registrador2, L1

Instruções para Tomada de Decisões

- A segunda instrução significa ir até a instrução rotulada por “L1” se o valor no “registrador1” **não for igual** ao valor no “registrador2”. O mnemônico bne significa ***b****ranch if* ***n****ot* ***e****qual* (desviar se não igual).

bne registrador1, registrador2, L1

Instruções para Tomada de Decisões

- Exemplo:
  - Analise o código a seguir:

    if (i == j)

    f = f – i;

    else

    { f = g + h;

    f = f – i; }
  - Supondo que as cinco variáveis de f até j correspondam aos cinco registradores de $s0 a $s4, qual o código MIPS gerado pelo compilador?

Instruções para Tomada de Decisões

- Resposta:

```
beq $s3, $s4, L1          # Desvia para L1 se i = j
add $s0, $s1, $s2         # f = g + h (executa se i ≠ j)
L1: sub $s0, $s0, $s3     # f = f – i (sempre executado)
```

Instruções de Desvio

- Nas instruções de tomada de decisão, o salto depende de uma condição. Uma importante instrução de desvio é o salto incondicional.
- É o equivalente ao *go to*,

j L1

o "j" é o mnemônico de ***j****ump* (pulo). A instrução acima significa, ir até a instrução rotulada por "L1".

Instruções de Desvio

- Exemplo:
  - Dado o código em C abaixo:

```
if ( i == j )
  f = g + h;
else
  f = g – h;
```

  - Sabendo que as variáveis f, g, h, i, j serão associadas aos registradores de $s0 a $s4, qual código em assembler MIPS poderia substituir o código acima?

Instruções de Desvio

- Resposta:

```
bne $s3,$s4,Else        # vá para Else se i <> j
add $s0,$s1,$s2         # f = g + h (ignorada se i <> j)
j Exit                  # vá para Exit
Else:sub $s0,$s1,$s2    # f = g + h (ignorada se i == j)
Exit:                   # rótulo Exit
```

Instruções de Desvio

- Exemplo:
  - Dado o código em C abaixo:

```
While (save [i] == k)
    i += 1;
```

  - Sabendo que i e k correspondem aos registradores $s3 e $s5 e a base do vetor save esteja em $s6. Qual código em assembler MIPS corresponde ao código acima?

Instruções de Desvio

- Resposta:

```
sll St1,$s3,2          # $t1 = 4 * i
add $t1,$t1,$s6        # $t1= endereço-base de save[i]
Loop:        # while
lw  $t0,0($t1)         # $t0 = save[i]
bne $t0,$s5,Exit       # vá para Exit se save[i] <> k
addi $t1,$t1,4         # i = i + 1
j Loop       # vá para Loop
Exit:        # rótulo Exit
```

Loops

- O teste de igualdade ou desigualdade provavelmente é o teste mais popular, mas às vezes é útil ver se uma variável é menor do que outra variável. Essas comparações são realizadas com uma instrução que compara dois registradores e atribui 1 a um terceiro registrador se o primeiro for menor do que o segundo; caso contrário é atribuído 0.

Loops

- A instrução MIPS é chamada ***s****et on* ***l****ess* ***t****han* (atribuir se menor que) ou slt.

slt $t0, $s3, $s4

- A instrução atribuirá 1 ao registrador “$t0” se o valor no registrador “$s3” for menor do que o valor no registrador “$s4”; caso contrário é atribuído 0 ao registrador “$t0”.

Loops

- Operadores constantes são populares nas comparações. **Como o registrador "$zero" sempre tem 0**, já podemos comparar com 0. Para comparar com outros valores, existe uma versão imediata da instrução "slt".

slti $t0,$s2,10

- A instrução "slti" funciona tal qual a "slt", a diferença é que compara o registrador "$s2" com uma constante.

Suporte a Procedimentos

- Um procedimento ou função é uma ferramenta que os programadores utilizam para estruturar programas, tanto para torná-los mais fáceis de entender quanto para permitir que o código seja reutilizado.
- Os procedimentos permitem que o programador se concentre em apenas uma parte da tarefa de cada vez, com os parâmetros atuando com uma barreira entre o procedimento e o restante do programa e dos dados, permitindo que sejam passados valores e resultados de retorno.

Suporte a Procedimentos

- Na execução de um procedimento, o programa precisa seguir a seis etapas seguintes:
  1. Colocar parâmetros em um lugar onde o procedimento possa acessá-los;
  2. Transferir o controle para o procedimento;
  3. Adquirir os recursos de armazenamento necessários para o procedimento;
  4. Realizar a tarefa desejada;
  5. Colocar o valor do retorno em um lugar onde o programa que o chamou possa acessá-lo;
  6. Retornar o controle para o ponto de origem, pois um procedimento pode ser chamado de vários pontos em um programa.

Suporte a Procedimentos

- O software do MIPS utiliza a seguinte convenção na alocação de seus 32 registradores para a chamada de procedimentos:
  - **$a0 - $a3**: quatro registradores de argumento, para passar parâmetros (R4 – R7);
  - **$v0 - $v1**: dois registradores de valor, para valores de retorno (R2 e R3);
  - **$ra**: um registrador de endereço de retorno, para retornar ao ponto de origem (R31).

Suporte a Procedimentos

- Além de alocar esses registradores, o assembler MIPS inclui uma instrução apenas para os procedimentos:

  jal EndereçoProcedimento

  - A instrução jal (***j****ump-****a****nd-****l****ink*) desvia para um endereço "EndereçoProcedimento" e simultaneamente salva o endereço da instrução seguinte no registrador $ra *(****r****eturn* ***a****ddress*).
- Um registrador mantém o endereço da instrução atual, o registrador PC (***P****rogram* ***C****ounter*).

Suporte a Procedimentos

- A instrução “jal” salva o endereço PC + 4 no registrador “$ra” para o *link* com a instrução seguinte, a fim de preparar o retorno do procedimento.
- O MIPS utiliza também uma instrução de ***j**ump **r**egister* – jr (salto de registro), significando um desvio incondicional para o endereço especificado em um registrador:

jr $ra

Suporte a Procedimentos

- A instrução “jr” pula para o endereço armazenado no registrador “$ra”. Assim, o programa que chama (***caller***) coloca os valores de parâmetro em $a0-$a3 e utiliza “jal X” para desviar para o procedimento X (***callee***). O *callee*, então, realiza os cálculos, coloca os resultados em $v0-$v1 e retorna o controle para o *caller* usando “jr $ra”.

Suporte a Procedimentos

- Existe a necessidade de preservar os dados contidos nos registradores antes do procedimento ser chamado, para que possam ser restaurados e o programa principal seguir sem problemas. Para tanto usa-se uma pilha (uma fila do tipo "último a entrar, primeiro a sair" - ***Last In First Out***).
- Uma pilha precisa de um ponteiro para o endereço alocado mais recentemente na pilha, $sp – R29 (sp é um mnemônico de ***s****tack* ***p****ointer*).
- O ponteiro da pilha é ajustado em uma *word* para cada registrador salvo ou restaurado.

Suporte a Procedimentos

- O ato de colocar dados na pilha, chama-se "***push***" e o ato de retirar dados da pilha, chama-se "***pop***".
- As pilhas crescem em sentido inverso, do endereço de memória maior para o endereço de memória menor.
- Quando adicionamos dados na pilha, temos que subtrair o valor correspondente ao tamanho do que adicionamos ao ponteiro "$sp", e quando somamos um valor, retiramos da pilha.

Suporte a Procedimentos

- Exemplo:
  - Converter o procedimento escrito na linguagem de alto nível C para o código assembler MIPS:

```
int exemplo (int g, int h, int i, int j) {
    int f;
    f = (g + h) – (i + j);
    return f; }
```

  - Supondo que as variáveis de parâmetros g, h , i e j correspondem aos registradores $a0, $a1, $a2 e $a3, e f corresponde a $s0.

Suporte a Procedimentos

Suporte a Procedimentos

- Resposta:

```
Exemplo_folha:          # rótulo, início do procedimento
addi  $sp,$sp,-12       # cria espaço na pilha p/ 3 itens
sw $t1, 8($sp)          # salva $t1 para usar depois
sw $t0, 4($sp)          # salva $t0 para usar depois
sw $s0, 0($sp)          # salva $s0 para usar depois
add $t0,$a0,$a1         # $t0 = g + h
add $t1,$a2,$a3         # $t1 = i + j
sub $s0,$t0,$t1         # $s0 = $t0 - $t1 ou f = (g + h) - (i + j)
add $v0,$s0,$zero       # retorna f ($v0 = $s0 + 0)
lw $s0, 0($sp)          # restaura $s0
lw $t0, 4($sp)          # restaura $t0
lw $t1, 8($sp)          # restaura $t1
addi $sp,$sp,12         # ajusta a pilha para remover os 3 itens
jr $ra                  # desvia de volta à rotina que chamou (caller)
```

Suporte a Procedimentos

- Na prática, existem alguns registradores que são usados de forma temporária e não há a necessidade de preservarmos o seu valor, enquanto outros precisão ser salvos, são eles:
  - **$t0-$t9**: são 10 registradores temporários que não são preservados pelos procedimento chamado;
  - **$s0-$s7**: são 8 registradores que precisam ser preservados em uma chamada de procedimento, se forem ser usados pelo procedimento.

Procedimentos aninhados

- Os procedimentos que não chamam outros são denominados **procedimentos folha**.
- Procedimentos podem chamar outros procedimentos e a si mesmos (recursivos).
- Suponha um procedimento A com um argumento 3 armazenado em “$a0” e que chame outro procedimento B com um argumento 7, também colocado em “$a0”. **Como A ainda não terminou sua tarefa, existe um conflito com relação ao uso do registrador “$ra” e “$a0”.**

Procedimentos aninhados

- Para evitar conflitos, o *caller* empilha quaisquer registradores de argumento ($a0-$a3) e/ou registradores temporários ($t0-$t9) que sejam necessários após a chamada.
- O *callee* empilha o registrador de endereço de retorno “$ra” e quaisquer registradores salvos ($s0-$s7) usados por ele.

Procedimentos aninhados

- Exemplo:
  - Dado o procedimento recursivo em C abaixo:

```
int fatorial ( int n ){
    if ( n < 1 )
        return (1);
    else
        return ( n * fatorial ( n-1 ) )}
```

  - O parâmetro n corresponde ao registrador de argumentos “$a0”. O programa compilado começa com o rótulo do procedimento e depois salva dois registradores na pilha e o endereço de retorno e “$a0”. Qual poderia ser o código em assembler do MIPS?

Suporte a Procedimentos

- Resposta:

```
fatorial:       # rótulo, início do procedimento
addi $sp,$sp,-8         # ajusta a pilha para adicionar 2 itens
sw $ra, 4($sp)          # salva o endereço de retorno
sw $a0, 0($sp)          # salva o argumento n
slti $t0,$a0,1 # teste para n < 1
beq $t0,$zero,L1        # se n >= 1, vai para L1
addi $v0,$zero,1        # retorna 1
addi $sp,$sp,8          # retira 2 itens da pilha
jr $ra          # retorna para o caller
L1:addi $a0,$a0,-1      # n >= 1: argumento recebe (n - 1)
jal factorial   # chama factorial com (n - 1)
lw $a0, 0($sp)          # ponto de retorno, restaura o argumento n
lw $ra, 4($sp)          # restaura o endereço de retorno
addi $sp,$sp,8          # ajusta a pilha para retirar 2 itens
mul $v0,$a0,$v0         # registro de retorno ($v0) = n * fact (n - 1)
jr $ra          # retorna para o caller
```