Faculdade de Computação

Arquitetura e Organização de Computadores 1

Atividade de Programação MIPS Assembly 01

Prof. Cláudio C. Rodrigues

Problemas:

P1) Apresente uma razão para que o processador MIPS não disponibilize uma instrução que realize subtração com valor imediato?

P2) Considere as declarações de uma variável string (str) realizadas conforme as sentenças abaixo. Explique de forma objetiva como o espaço de endereçamento de memória será configurado, para cada uma das declarações.

a) `static char str[] = "thing";`
b) `char str[] = "thing";`
c) `char *str = malloc(6); strcpy(str, "thing");`

P3) Modos de Endereçamento do MIPS:

Temos vários modos de endereçamento para o acesso à memória (imediato não listados):

- endereçamento base deslocamento.
- endereçamento relativo ao PC.
- endereçamento pseudo-direto.
- endereçamento por registrador.

a) Uma determinada solução de programação em *MIPS assembly* necessita de uma instrução para executar um salto para um endereço 2^{28} + 4 bytes distante da atual posição do PC. Como você faria para resolver? Considere que o endereço de destino será conhecido em tempo de compilação.

b) Uma determinada solução de programação em *MIPS assembly* necessita de uma instrução para executar um desvio para um endereço 2^{17} + 4 bytes distante da atual posição do PC, quando $t0 é igual a 0. Considere que não saltaremos para um endereço superior a 2^{28} bytes. Como você faria para resolver?

P4) Considere as *pseudoinstruções* abaixo, Elabore em linguagem de montagem (*assembly*) uma sequência mínima de instruções MIPS nativas, para que se obtenha o mesmo efeito. Utilize o registrador $at como um registrador temporário.

```
a) addiu $s1, $s2, imm32              # imm32 is a 32-bit immediate
b) bge $s1, imm32, Label              # imm32 is a 32-bit immediate
a) rol $s1, $s2, 5                    # rol = rotate left $s2 by 5 bits
b) ble $t3, $t5, L          ---->     # if ($t3<=$t5) goto L
c) bgt $t4, $t5, L          ---->     # if ($t4>$t5)  goto L
```

P5) Convenções

a) Como deve o registrador $sp ser utilizado? Quando é que devemos adicionar ou subtrair $sp?
b) Quais registradores necessitam ser salvos ou restaurados antes de executarmos a instrução **jr** para retornar de uma função?
c) Quais registradores necessitam ser salvos antes de executar a instrução **JAL**?
d) Como podemos passar parâmetros para funções?
e) O que devemos fazer se houvera necessidade de passar mais do que quatro parâmetros para uma função? Como são retornados valores pelas funções?

P6) Traduza o fragmento de código C abaixo para MIPS assembly. Considere que os arrays inteiros **a** e **b** tem seus endereços base nos registradores **$a0** e **$a1**, respectivamente. O valor de **n** está no registrador **$a2**.

```
for (i=0; i<n; i++) {
     if (i > 2) {
          a[i] = a[i-2] + a[i-1] + b[i];
     } else {
          a[i] = b[i];
     }
}
```

P7) Comente o código MIPS a seguir e descreva em uma sentença qual é a atividade que ele realiza. Assuma que **$a0** é utilizado para a entrada e inicialmente contém **n**, um inteiro positivo. Assuma que **$v0** é utilizado para a saída.

```
begin: addi $t0, $zero, 0
       addi $t1, $zero, 1
loop:  slt $t2, $a0, $t1
       bne $t2, $zero, finish
       add $t0, $t0, $t1
       addi $t1, $t1, 2
       j loop
finish: add $v0, $t0, $zero
```

P8) Converta o seguinte fragmento de código, escrito em linguagem C, para a linguagem MIPS assembly. Considere que as variáveis **i**, **j**, **x** e **y** foram atribuidas aos registradores **$t0**, **$t1**, **$a1** e **$a2** respectivamente. Considere que o endereço base do array foi guardado no registrador $a0.

a)
```
int i = 0;
int j = -1;
while(i < 10){
     if((i & 0x0001)==1) //se i for impar, adicione-o ao j
          j += i;
     i++;
}
```

b)
```
int fun(char [] a, int x, int y){
  if (a[x] > a[y]){
    a[x+1] = a[y];
    return a[x] + a[y];}
  else{
    return a[x] - a[y];}
}
```

P9) O programa a seguir tenta copiar palavras de um endereço no registrador **$a1**, contando o número de palavras copiadas no registrador **$v0**. O programa para de copiar quando encontra uma palavra igual a **0**. Você não tem que preservar o conteúdo dos registradores **$v1**, **$a0** e **$a1**. Esta palavra de terminação deve ser copiada, mas não contada.

```
loop:     lw $v1, 0($a0)          # read next word from source
          addi $v0, $v0, 1        # increment count words copied
          sw $v1, 0($a1)          # write to destination
          addi $a0, $a0, 1        # advance pointer to next source
          addi $a1, $a1, 1        # advance pointer to next dest
          bne $v1, $zero, loop    # loop if word copied != zero
```

Existem múltiplos erros neste programa MIPS; conserte-os e torne este programa *bug-free*. O modo mais fácil de escrever programas MIPS é utilizar o simulador MARS disponível no sítio da disciplina. Você pode efetuar o download do simulador através dos *links* na página do curso.

P10) Considere o seguinte fragmento de código C:

```
for (i=0;i<=100;i=i+1) {
     a[i] = b[i] + c;
}
```

Assuma que **a** e **b** são *arrays* de inteiros (.word) e que o endereço base de **a** está em **$a0** e que o endereço base de **b** está em **$a1**. O registrador **$t0** está associado a variável **i** e o registrador **$s0** a variável **c**. Escreva o código utilizando o conjunto de instruções MIPS. Quantas instruções são executadas durante a execução deste código? Quantas referências de dados na memória serão feitas durante a execução?

P11) Converta o fragmento de código abaixo, escrito em linguagem C, para a linguagem do MIPS assembly.

```
/** Returns the number of bytes in S before, but not counting, the null
terminator. */
size_t string_length(char *s) {
    char *s2 = s;
    while(*s2++);
    return s2 - s - 1;
}
```

P12) Converta o fragmento de código abaixo, escrito em linguagem C, para a linguagem do MIPS assembly.

```
/** Converts the string S to lowercase */
void string_to_lowercase(char *s) {
    for(char c = *s; (c=*s) != '\0'; s++) {
         if(c >= 'A' && c <= 'Z') {
              *s += 'a' - 'A';
         }
    }
}
```

P13) Converta o fragmento de código abaixo, escrito em linguagem C, para a linguagem do *MIPS assembly*. O fragmento calcula a soma de números de 0 a N. Considere que **$s0** mantem N (N>=0), **$s1** mantem a soma. Transforme a solução em estrutura de função.

```
int soma= 0
if (N == 0) return 0;
   while (N != 0) {
   soma += N
   N--;
}
return soma;
```

P14) Escreva em linguagem MIPS assembly um programa denominado *contadígitos* que leia do dispositivo padrão de entrada (teclado) um valor inteiro **n**. O programa deve imprimir na tela de saída o valor de **n** e o **número de algarismos** que possui.

P15) Escreva, em linguagem Assembly do MIPS, uma função denominada **amplitude** que receba uma relação de números armazenados em um vetor **v** de **n** elementos reais e, retorne como resposta a sua amplitude. A amplitude de uma relação de números reais é a diferença entre o maior e o menor valores da relação.
Por exemplo, a amplitude da relação {5, 7, 15, 2, 23 21, 3, 6} é 23 - 2 = 21.

P16) Converta o fragmento de código abaixo, escrito em linguagem C, para a linguagem do MIPS assembly.

```
// Nth_Fibonacci(n):
// $s0 -> n, $s1 -> fib
// $t0 -> i, $t1 -> j
// Assume fib, i, j are these values
int fib = 1, i = 1, j = 1;
if (n==0) return 0;
else if (n==1) return 1;
n -= 2;
while (n != 0) {
  fib = i + j;
  j = i;
  i = fib;
  n--;
}
return fib;
```