Faculdade de Computação

Arquitetura e Organização de Computadores 1

Prática 2 - Laboratório de Programação MIPS

Prof. Cláudio C. Rodrigues

Problemas:

P1) Traduza os códigos de máquina a seguir para sentenças MIPS assembly:

a) 0000 0000 0000 1001 0111 1101 1000 0000
b) 1000 1101 0000 1000 0000 0000 0100 0000
c) 0x3C12F000

P2) Qual é o valor armazenado no registrador **$t2** após a execução da sequência de instruções dos itens abaixo? Considere que o valor armazenado no registrador **$t0** é 0x55555555 e, no registrador **$t1** é 0x12345678.

a)
```
sll $t2, $t0, 4
or $t2, $t2, $t1
```
b)
```
sll $t2, $t0, 4
andi $t2, $t2, -1
```
c)
```
srl $t2, $t0, 3
andi $t2, $t2, 0xFFEF
```

P3) Considere que você é um projetista da ARM e deve projetar um novo conjunto de instruções com apenas 16 registradores de 32 bits. Isto significa que precisamos apenas de campos de 4 bits para identificar operandos registradores em nossas instruções.

a) Quantos bits extras obteremos para serem utilizados em outros campos nos seguintes formatos: **R**, **J** e **I**?

b) Para instruções R-formato, você atribuiria os bits extras para **opcode**, **shamt** ou **funct**? Explique sua escolha em uma ou duas frases.

c) Para instruções do formato-I, você, naturalmente, atribuiria os bits extras para o campo imediato, resultando no seguinte formato:

```
[ opcode (6) | rs (4) | rt (4) | immediate (18) ]
```

Qual a fração do espaço de endereçamento de memória, podemos alcançar com uma instrução de desvio (*branch*)?

d) Assuma que o *Program Counter* (PC) contém atualmente o endereço **0x08000000**. Qual é o menor endereço (em hexadecimal), podemos alcançar com uma instrução de desvio?

P4) Apresente uma razão para que o processador MIPS não disponibilize uma instrução que realize subtração com valor imediato?

P5) Explique por que uma codificação em *assembly* apresentaria problemas ao implementar a instrução de desvio na sequência de código a seguir:

```
here: beq $s0, $s2, there
…
there: add $s0, $s0, $s0
```

Mostre como este código poderia ser escrito em *assembly* de forma a resolver estes problemas.

P6) Elabore em linguagem de montagem (*assembly*) uma sequência mínima de instruções MIPS para realizar as ações das pseudoinstruções abaixo:

```
a) ble $t3, $t5, L      ---->    # if ($t3<=$t5) goto L

b) bgt $t4, $t5, L      ---->    # if ($t4>$t5)  goto L

c) if ( $t0 >= 0x12345678 ) {$t2=0;}
```

P7) 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");
```

P8) 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?

P9) Considere as seguintes definições de dados:

```
      .data
var1: .byte 3, -2, 'A'
var2: .half 1, 256, 0xffff
var3: .word 0x3de1c74, 0xff
      .align 3
str1: .asciiz "COE308"
```

a) Mostre em código hexadecimal como a memória alocada seria configurada com as declarações acima. Considere que a ordenação de bytes "*Little Endian*" é aplicada para os bytes das palavras (.word) e meias palavras (.half). Os caracteres 'A'e 'O' estão codificados em ASCII.

P10) Considere as pseudo-instruções abaixo, produza uma sequência mínima de instruções MIPs reais, para que se obtenha o mesmo efeito. Utilize o registrador $at como um registrador temporário.

```
b) addiu $s1, $s2, imm32              # imm32 is a 32-bit immediate
c) bge $s1, imm32, Label              # imm32 is a 32-bit immediate
d) rol $s1, $s2, 5                    # rol = rotate left $s2 by 5 bits
```

P11) 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?

f) Como são retornados valores pelas funções?

P12) Comente o código MIPS a seguir e descreva em uma sentença que ele computa. 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
```

P13) 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];}
}
```

P14) 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];
     }
}
```

P15) O seguinte fragmento de código processa um *array* e produz dois importantes valores nos registradores **$v0** e **$v1**. Assuma que o *array* consiste de 5000 palavras indexadas de 0 a 4999, e seu endereço base está armazenado em **$a0** e seu tamanho (5000) em **$a1**. Descreva em uma sentença o que este código faz. Especificamente, o que será retornado em **$v0** e **$v1**?

```
          add $a1, $a1, $a1
          add $a1, $a1, $a1
          add $v0, $zero, $zero
          add $t0, $zero, $zero
outer:    add $t4, $a0, $t0
          lw $t4, 0($t4)
          add $t5, $zero, $zero
          add $t1, $zero, $zero
inner:    add $t3, $a0, $t1
          lw $t3, 0($t3)
          bne $t3, $t4, skip
          addi $t5, $t5, 1
skip:     addi $t1, $t1, 4
          bne $t1, $a1, inner
          slt $t2, $t5, $v0
          bne $t2, $zero, next
          add $v0, $t5, $zero
          add $v1, $t4, $zero
next:     addi $t0, $t0, 4
          bne $t0, $a1, outer
```

P16) 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.

P17) 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?

P18) Converta a função abaixo, escrita em linguagem C, para a linguagem do *MIPS assembly*. Considere que **$a0** e **$a1** guardam os ponteiros para os inteiros, troque os valores para os quais eles apontam usando a pilha.

```
void swap(int *a, int *b) {
  int tmp = *a;
  *a = *b;
  *b = tmp;
}
```

P19) 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) {
   suma += N
   N--;
}
return soma;
```

P20) Converta o seguinte fragmento de código, escrito em linguagem C, para a linguagem MIPS assembly.

```
int sumton(unsigned int n){
     if(n==0) return 0;
     else return n + sumton(n-1);
}
```

P21) Converta o fragmento de código abaixo, escrito em linguagem C, para a linguagem do MIPS assembly.

```
// Strcpy:
// $s1 -> char s1[] = “Hello!”;
// $s2 -> char *s2 =
// malloc(sizeof(char)*7);
int i=0;
do {
  s2[i] = s1[i];
  i++;
} while(s1[i] != '\0');
s2[i] = '\0';
```

P22) 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';
         }
    }
}
```

P23) 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;
}
```

P24) Converta o fragmento de código abaixo, escrito em linguagem C, para a linguagem do MIPS assembly.

```
/** Return the number of odd numbers in a number array */
uint32_t number_odds(uint32_t *numbers, uint32_t size) {
    uint32_t odds = 0;
    for (uint32_t i = 0; i < size; i++)
         odds += *(numbers+i) & 1;      // odds += numbers[i] & 1;
    return odds;
}
```

P25) 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;
```

P26) 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.