Universidade Federal de Santa Catarina

Centro Tecnológico – Departamento de Informática e de Estatística

INE 5607 – Organização e Arquitetura de Computadores

Prof. Laércio Lima Pilla

Gabarito das questões introdutórias de linguagem de montagem.

1) Compile o seguinte código de linguagem de alto nível:

```
c = 2*a+b;

d = a-b/2;
```

Use o seguinte mapeamento de variáveis para registradores: (a,b,c,d) -> ($s0,$s1,$s2,$s3)

Solução:

```
1. sll $t0, $s0, 1      # temp <- a << 1, o que equivale a a*2
2. add $s2, $t0, $s1    # c <- temp + b
3. srl $t1, $s1, 1      # temp <- b >> 1, o que equivale a b/2
4. sub $s3, $s0, $t1    # d <- a - temp
```

Importante: as operações intermediárias não modificaram os valores originais de a e b.

2) Considere o seguinte trecho de código em linguagem de montagem:

```
1. and $t0, $s0, $s1
2. and $t1, $s0, $s2
3. or  $t2, $t0, $t1
4. nor $t3, $s2, $zero
5. and $t4, $s1, $t3
6. or  $s3, $t2, $t4
```

Sabendo que os registradores $s0, $s1, $s2 e $s3 correspondem as variáveis x, y, z e resposta, respectivamente, escreva o código em linguagem de alto nível que gerou o código apresentado.

Solução:

Uma forma prática de solucionar esse problema é começar substituindo registradores por variáveis e mnemônicos de instruções por operações. Um exemplo de tradução é apresentado abaixo

```
1. and $t0, $s0, $s1     ->    temp_0 = x & y
2. and $t1, $s0, $s2     ->    temp_1 = x & z
3. or  $t2, $t0, $t1     ->    temp_2 = temp_0 | temp_1
4. nor $t3, $s2, $zero   ->    temp_3 = ~z
5. and $t4, $s1, $t3     ->    temp_4 = y & temp_3
6. or  $s3, $t2, $t4     ->    resposta = temp_2 | temp_4
```

O próximo passo é começar a substituir os valores temporários na saída pelas subexpressões anteriores.

```
resposta = temp_2 | temp_4

resposta = ( temp_0 | temp_1 ) | temp_4

resposta = ( (x & y) | temp_1 ) | temp_4

resposta = ( (x & y) | (x & z) ) | temp_4

resposta = ( (x & y) | (x & z) ) | ( y & temp_3 )

resposta = ( (x & y) | (x & z) ) | ( y & ~z )
```

Respostas plausíveis para esta questão são

```
resposta = ((x & y) | (x & z)) | ( y & ~z )
```

e

```
resposta = (x & y) | (x & z) | ( y & ~z )
```

3) Compile o seguinte código de linguagem de alto nível:

```
Array_A[ 2 ] = Array_B[ 10 ] + Array_B[ Array_A[ 5 ] + 17 ]
```

Considere que o endereço inicial de Array_A está armazenado em $a0 e o endereço inicial de Array_B está armazenado em $a1.

Solução:

O código acima pode ser decomposto em partes menores como ilustrado abaixo:

```
1. Temp_1 = Array_A[ 5 ]
2. Temp_2 = Temp_1 + 17
3. Temp_3 = Array_B[ Temp_2 ]
4. Temp_4 = Array_B[ 10 ]
5. Temp_5 = Temp_3 + Temp_4
6. Array_A[ 2 ] = Temp_5
```

Instruções no estilo temp = array [valor imediato] podem ser transformadas em loads simples, enquanto a instrução 3 necessita de um cálculo de endereço mais elaborado. O código em linguagem de montagem é apresentado abaixo.

```
1. lw   $t1, 20($a0)
2. addi $t2, $t1, 17
3. sll  $t2, $t2, 2
4. add  $t2, $t2, $a1
5. lw   $t3, 0($t2)
6. lw   $t4, 40($a1)
7. add  $t5, $t3, $t4
8. sw   $t5, 8($a0)
```

O que fazem as instruções de 2 a 4? Elas calculam o índice a ser acessado no Array_B como se em alto nível, depois multiplicam o valor por 4 porque acessamos bytes no nível de linguagem de montagem e, por fim, somam esse deslocamento com o endereço base de Array_B.

Por exemplo, a instrução Temp_1 = Array_A[5] pode ser feita como apresentada na instrução 1 do código simbólico acima ou através do seguinte conjunto de instruções:

```
1. addi $t0, $zero, 5
2. sll $t0, $t0, 2
3. add $t0, $t0, $a0
4. lw $t1, 0($t0)
```