Facultad de Ingeniería
Departamento de Ingeniería en Sistemas
Arquitectura de Computadoras (IS-603)

Tarea Ejercicios MIPS

Presentado por:
Alma Celeste Flores Martínez 20131008875

Ciudad Universitaria, Tegucigalpa MDC, Francisco Morazán
Agosto 2016

Ejercicios

Para la realización de los siguientes ejercicios, use el simulador asignado para la arquitectura MIPS (cuando el ejercicio lo permita). Cuando sea conveniente o necesario, haga las suposiciones correspondientes. También, se permite el uso de pseudoinstrucciones.

1. Para el siguiente fragmento de código C, cuál es el correspondiente código en ensamblador MIPS

   f = g + (h - 5)

   Solución:

```
# Suponiendo que los datos f, g, h están asignados a los
# Registros $s0, $s1 y $s2, respectivamente:

 subi $t0, $s2, 5
add $s0, $s1, $t0
```

f	$s0
g	$s1
h	$s2

2. Para el siguiente fragmento de código en ensamblador MIPS, ¿Cuál es el correspondiente código en lenguaje C?

   add f, g, h
   add d, i, f
   sub b, f, d

   Solución:

```
b = (g + h) – (i + f)
```

3. Para el siguiente fragmento de código C, cuál es el correspondiente código en ensamblador MIPS

   B[8] = A[i-j]

   Solución:

```
# Suponiendo que i, j están asignados a los registros $s0
# Y $s1 respectivamente y que la dirección del primer
# Elemento de los vectores A y B llamados  A[0], B[0], están
```

```
# En $s2 y $s3 Respectivamente:

  sub $t0, $s0, $s1
  lw $t1, $t0($s2)
  sw $t1, 32($s3)
```

i	$s0
j	$s1
A[0]	$s2
B[0]	$s3

4. Para el siguiente fragmento de código en ensamblador MIPS, ¿Cuál es el correspondiente código en lenguaje C?

```
sll  $t0, $s0, 2          # St0 = f * 4
add  $t0, $s6, $t0        # $t0 = &A[f]
sll  $t1, $s1, 2          # $t1 = g * 4
add  $t1, $s7, $t1        # $t1 = &B[g]
lw   $s0, 0($t0)          # f = A[f]
addi $t2, $t0, 4
lw   $t0, 0($t2)
add  $t0, $t0, $s0
sw   $t0, 0($t1)
```

Solución:

```
sll  $t0, $s0, 2          # St0 = f * 4
add  $t0, $s6, $t0        # $t0 = &A[f]
sll  $t1, $s1, 2          # $t1 = g * 4
add  $t1, $s7, $t1        # $t1 = &B[g]
lw   $s0, 0($t0)          # f = A[f]
addi $t2, $t0, 4          # $t2 = &A[f + 1]
lw   $t0, 0($t2)          # $t0 = A[f + 1]
add  $t0, $t0, $s0        # $t0 = A[f + 1] + A[f]
sw   $t0, 0($t1)          # B[g] = A[f + 1] + A[f]
```

B[g] = A [f + 1] + A[f]; // donde f = A[f]

5. La siguiente tabla muestra un arreglo de 32 bits almacenado en memoria:

Address	Data
24	2
28	4
36	6

40	1

a. Implemente código C que ordene el arreglo
b. Escriba el código MIPS correspondiente al algoritmo en código C del apartado a

Solución:

a.

Suponiendo que el arreglo se llama A tendremos lo siguiente:

Arreglo A[]	Dirección	Dato
A[0]	24	2
A[1]	28	4
A[2]	32	3
A[3]	36	6
A[4]	40	1

```
// Suponemos que tenemos dos variables temporales t1 y t2:
:
:
  t1 = A[0];       // t1 = 2
  t2 = A[1];       // t2 = 4
  A[0] = A[4];     //A[0] = 1
  A[1] = t1;       // A[1] = 2
  A[4] = A[3];     // A[4] = 6
  A[3] = t2;       //A[3] = 4
:
:
```

El arreglo ordenado queda así:

Arreglo A[]	Dirección	Dato
A[0]	40	1
A[1]	24	2
A[2]	32	3
A[3]	28	4
A[4]	36	6

b.

```
# Suponiendo que la primera dirección del arreglo A[] esta
# Almacenada en $s6
```

```
lw $t0, 0($s6)
lw $t1, 4($s6)
lw $t2, 16($s6)
sw $t2, 0($s6)
sw $t0, 4($s6)
lw $t0, 12($s6)
sw $t0, 16($s6)
sw $t1, 12($s6)
```

$s6	A[0]	1
$s6 + 4	A[1]	2
$s6 + 8	A[2]	3
$s6 + 12	A[3]	4
$s6 + 16	A[4]	6

Prueba de Escritorio							
$t0	$t1	$t2	$s6	$s6 + 4	$s6 + 8	$s6 + 12	$s6 + 16
~~2~~	4	1	~~2~~	~~4~~	3	~~6~~	~~1~~
6	**4**	**1**	**1**	**2**	**3**	**4**	**6**

6. Muestre cómo el valor **0xabcdef12** es almacenado en memoria

   a. En un esquema de memoria little endian
   b. En un esquema de memoria big endian

Solución:

a. Little Endian:

Ordena los bytes del menos significativo al más significativo.

Dirección	Dato
12	ab
8	cd
4	ef
0	12

b. Big Endian:

Ordena los bytes del más significativo al menos significativo.

Dirección	Dato
12	12
8	ef
4	cd
0	ab

7. Traslade el valor **0xabcdef12** a decimal.

Solución:

= (16e0 * 2) + (16e1 * 1) + (16e2 * 15) + (16e3 * 14) + (16e4 * 13) + (16e5 * 12) + (16e6 * 11) + (16e7 * 10)
= 2 + 16 + 3840 + 57344 + 851968 + 12582912 + 184549376 + 268435456
0xabcdef12 = 2882400018

8. Traslade el siguiente código MIPS a código C

```
addi    $t0, $s6, 4          # $t0 = &A[f + 1]
add     $t1, $s6, $zero      # $t1 = &A[ f ]
sw      $t1, 0($t1)          # $t1=  A[ f ]
lw      $t0, 0($t0)          # $t0 =  A[f+1]
add     $s0, $t1, $t0        # B[g] = A[ f ] + A[f + 1]
```

Solución:

```
//Suponemos que $s6 y $s0, corresponde a los vectores llamados: A[ f ] y B[ g ]
//respectivamente
:
:
B[ g ] = A[ f ] + A[f + 1];
:
:
```

9. Traduzca el código MIPS del ejercicio anterior en su correspondiente versión de código máquina, identificando cada uno de sus campos.

addi $t0, $s6, 4

TIPO I

Código Operación	Base	Destino	Desplazamiento
8	22	8	4
001000	10110	01000	0000000000000100

add $t1, $s6, $zero

TIPO R

OP	Operando1	Operando2	Destino	Shamt	Función
0	22	0	9	0	32
000000	10110	00000	01001	00000	100000

sw $t1, 0($t1)

TIPO I

Código Operación	Base	Destino	Desplazamiento
43	9	9	0
101011	01001	01001	0000000000000000

lw $t0, 0($t0)

TIPO I

Código Operación	Base	Destino	Desplazamiento
35	8	8	0
100011	01000	01000	0000000000000000

add $s0, $t1, $t0

TIPO R

OP	Operando1	Operando2	Destino	Shamt	Función
0	9	8	16	0	32
000000	01001	01000	10000	00000	100000

El código máquina completo seria:

```
0010 0010 1100 1000 0000 0000 0000 0100    // addi  $t0, $s6, 4
0000 0010 1100 0000 0100 1000 0010 0000    // add   $t1, $s6, $zero
1010 1101 0010 1001 0000 0000 0000 0000    // sw    $t1, 0($t1)
1000 1101 0000 1000 0000 0000 0000 0000    // lw    $t0, 0($t0)
0000 0001 0010 1000 1000 0000 0010 0000    // add   $s0, $t1, $t0
```

10. Asuma que los registros $s0 y $s1 contienen respectivamente los valores 0x80000000 y 0xD0000000:

a. ¿Cuál es el valor de $t0 para el siguiente código ensamblador

add $t0, $s0, $s1

Solución:

Al utilizar el simulador la ejecución termina con errores. $t0 = 00000000

b. ¿El contenido de $t0 es el resultado de la operación, o hay overflow?

Solución:

Hay overflow, esto se debe a que al realizar la operación y sumar $s0 con $s1 el resultado se pasa de los 32 bits.

c. ¿Cuál es el contenido de $t0 para el siguiente código MIPS

```
add   $t0, $s0, $s1
add   $t0, $t0, $s0
```

Solución:

$t0 = 00000000. Hay overflow.

11. Traduzca la siguiente instrucción máquina a ensamblador ¿De qué tipo es?

$$0000\ 0010\ 0001\ 0000\ 1000\ 0000\ 0010\ 0000_2$$

Solución:

Tipo R

OP	Operando1	Operando2	Destino	Shamt	Función
0	16	16	16	0	32
000000	10000	10000	10000	00000	100000

En código MIPS: add $s0, $s0, $s0

12. Proporcione la representación hexadecimal de la siguiente instrucción

```
sw    $t1, 32($t2)
```

TIPO I

Código Operación	Base	Destino	Desplazamiento
43	9	10	32
101011	01001	01010	0000000000100000

1010	10	0000	0
1101	13	0000	0
0010	2	0010	2
1010	10	0000	0

Notación Hexadecimal: 0xad2a0020

13. Proporcione el tipo, instrucción en lenguaje ensamblador, y la representación binaria de la instrucción descrita por los siguientes campos de instrucción MIPS:

```
op = 0, rs = 3, rt = 2, rd = 3, shamt = 0, funct = 34
```

Solución:

Tipo R

OP	Operando1	Operando2	Destino	Shamt	Función
0	3	2	3	0	34
000000	00011	00010	00011	00000	000001

Notación código ensamblador: sub $v1, $v1, $v0

14. Proporcione el tipo, instrucción en lenguaje ensamblador, y la representación binaria de la instrucción descrita por los siguientes campos de instrucción MIPS:

```
op = 0x23, rs = 1, rt = 2, const = 0x4
```

TIPO I

Código Operación	Base	Destino	Desplazamiento
35	1	2	4
100011	00001	00010	0000000000000100

Notación código ensamblador: lw $v0, 4($at)

15. Asuma que en un procesador MIPS, se expande la cantidad de registros de 32 a 128

a. ¿Cómo afectaría este cambio el tamaño de los campos de una instrucción MIPS de tipo **Registro?**

Solución:

Entre más registros, habran más bits por registro, también podría aumentar el tamaño del código. Menos overflow y posiblemente menos instrucciones.

16. Asuma los siguientes contenidos de registros:

```
$t0 = 0xAAAAAAAA, $t1 = 0x12345678
```

c. ¿Cuál es el contenido de **$t2** después de operar las siguientes instrucciones?

```
sll $t2, $t0, 8
or $t2, $t2, $t1
```

Solución:

Al utilizar el simulador $t2 = 0xbabefe78.

d. ¿Cuál es el contenido de **$t2** después de operar las siguientes instrucciones?

```
sll   $t2, $t0, 4
andi  $t2, $t2, −1
```

Solución:

Al utilizar el simulador $t2 = 0xaaaaaaa0.

17. Asuma que **$t0** contiene el valor **0x00101000.** ¿Cuál es el valor de **$t2** después de ejecutar las siguientes instrucciones?

```
        ...
        slt   $t2, $0, $t0
        bne   $t2, $0, ELSE
        j     DONE
ELSE:
        addi  $t2, $t2, 2
DONE:
        ...
```

Solución:

Al utilizar el simulador $t2 = 0x00000003.

18. Considere el siguiente ciclo MIPS:

```
        ...
LOOP:
        slt  $t2, $0, $t1
        beq  $t2, $0, DONE
        subi $t1, $t1, 1
        addi $s2, $s2, 2
        j    LOOP
DONE:
        ...
```

a. Asuma que el registro **$t1** es inicializado a **10**. ¿Cuál es el valor del registro **$s2** asumiendo que **$s2** es inicializado a cero?

Solución:

Al utilizar el simulador $s2 = 0x00000020.

19. Implemente en código ensamblador el siguiente código escrito en C:

```
int fib(int n){
      if (n==0)
            return 0;
      else if (n == 1)
            return 1;
      else
            return fib(n−1) + fib(n−2);
```

Solución:

```
.data
.text

fib: addi $sp, $sp, -12
    sw $ra, 8($sp)
    sw $s0, 4($sp)
    sw $a0, 0($sp)
    bgt $a0, $0, prueba
    add $v0, $0, $0
    j rtn
```

```
prueba: addi $t0, $0, 1
            bne $t0, $a0, gen
            add $v0, $0, $t0
            j rtn
gen: subi $a0, $a0,1
       jal fib
       add $s0, $v0, $0
       sub $a0, $a0,1
       jal fib
       add $v0, $v0, $s0
rtn: lw $a0, 0($sp)
     lw $s0, 4($sp)
     lw $ra, 8($sp)
     addi $sp, $sp, 12
     jr $ra
```

-También podríamos calcular el n-esimo término de la serie de Fibonacci de la siguiente manera:

```
# Asumiendo que:
# $s0 = n = 5
# $s1 = f
# $s2 = fant
# $s3 = i
# $s4 = faux
li $s0,5
li $s2,1
li $s1,1
li $s3,2
WHILE: bgt $s3,$s0,END
          move $s4,$s1
          add $s1,$s1,$s2
         move $s2,$s4
         addi $s3,$s3,1
         j WHILE
END:
```

f	f(n-1)
fant	f(n-2)
faux	Valor Auxiliar

FIN ☺