

**EXAMEN ORDINARIO FUNDAMENTOS DE COMPUTADORES. Prácticas**

Apellidos y nombre:

DNI:

Grupo:

1. (0,5 puntos) Escriba en el recuadro el código MIPS equivalente al siguiente código de alto nivel:

```
for (i=0; i<5; i++)
    p[i]=p[i+1];
```

donde p es un vector de 6 enteros de 32 bits cuya dirección se encuentra en el registro \$s5. Implemente el índice i mediante el registro \$s0.

Emplee comentarios para dar las explicaciones oportunas.

```

add    $t1, $zero, 5      # $t1 ← 5 para la comparación
add    $s0, $zero, $zero  # $s0 ← 0. (i = 0)

Bucle:
beq     $s0, $t1, Salir    # Comparación del índice i con 5
addi    $t0, $s0, 1        # $t0 ← i+1
sll     $t0, $t0, 2        # $t0 ← 4*(i+1)
add     $t0, $s5, $t0      # $t0 ← p+4*(i+1) (dirección de p[i+1])
lw      $t2, 0($t0)        # $t2 ← p[i+1]
sll     $t0, $s0, 2        # $t0 ← 4*i
add     $t0, $t0, $s5      # $t0 ← p+4*i (dirección de p[i])
sw      $t2, 0($t0)        # p[i] ← $t2 = p[i+1]
addi    $s0, $s0, 1        # i = i+1
j       Bucle

Salir:
```

2. (0,3 puntos) Escriba en decimal el contenido del registro \$f0 después de ejecutarse el fragmento de programa siguiente:

```
.data
```

```
A:.double 10.0, 4.0, 8.0
```

```
B:.double 3.0, 15.0, 20.0
```

Justificación:

Contenido final del registro \$f0:

**-2,5**

```

...
la      $t3, A
l.d     $f0, 0($t3)
addi    $t3, $t3, 8
l.d     $f4, 0($t3)
sub.d   $f4, $f4, $f0
la      $t4, B
l.d     $f0, 8($t4)
div.d   $f0, $f0, $f4
```

*El directivo .double reserva espacio para números en punto flotante con doble precisión (8 bytes cada uno).*

Instrucción	Efectos
la \$t3, A	\$t3 ← Dirección A
l.d \$f0, 0(\$t3)	\$f0:\$f1 ← Contenido dirección A (10.0)
addi \$t3, \$t3, 8	\$t3 ← A+8
l.d \$f4, 0(\$t3)	\$f4:\$f5 ← Contenido dirección (A+8) (4.0)
sub.d \$f4, \$f4, \$f0	\$f4:\$f5 ← \$f4:\$f5-\$f0:\$f1 (4.0-10.0 = -6.0)
la \$t4, B	\$t4 ← Dirección B
l.d \$f0, 8(\$t4)	\$f0:\$f1 ← Contenido dirección (B+8) (15.0)
div.d \$f0, \$f0, \$f4	\$f0:\$f1 ← \$f0:\$f1/\$f4:\$f5 (15.0/-6.0 = -2.5)

3. (0,3 puntos) Escriba en decimal el contenido del registro \$s0 después de ejecutarse el fragmento de programa siguiente:

```
.data
K: .word 19, -4, -30, 5, 8, 24
    ...
la    $s5, K
lw    $s0, 12($s5)
addi  $s0, $s0, -4
```

Contenido final del registro \$s0 (decimal):

1

Justificación:

Instrucción	Efectos
la    \$s5, K	\$s5 ← Dirección del vector K
lw    \$s0, 12(\$s5)	\$0 ← Contenido de la dirección (K+12) = 5
addi  \$s0, \$s0, -4	\$s0 ← \$s0-4 = 5-4 = 1

4. (0,5 puntos) Escriba en el recuadro el fragmento de código en lenguaje ensamblador MIPS correspondiente a la sentencia de lenguaje de alto nivel  $A[j][i] = A[i][j]$ . Supóngase que A es una matriz cuadrada de 3x3 componentes enteras cuya dirección se encuentra en el registro \$s5, que el índice i se halla en el \$s0 y que el índice j se encuentra en el \$s1. Incluya también la declaración de la citada matriz inicializándola con los valores que desee.

Emplee comentarios para dar las explicaciones oportunas.

Declaración de la matriz

```
A: .word 1, 2, 3, 4, 5, 6, 7, 8, 9
```

Fragmento de programa

```
addi  $t1, $zero, 3  # $t1 ← 3 (para las multiplicaciones, 3 = número de columnas)
mul    $t0, $s0, $t1 # $t0 ← Número de columnas*i
add    $t0, $t0, $s1 # $t0 ← Número de columnas*i+j
sll    $t0, $t0, 2    # $t0 ← 4*(Número de columnas*i+j)
add    $t0, $t0, $s5 # $t0 ← 4*(Número de columnas*i+j)+A (dirección de A[i][j])
lw     $t2, 0($t0)    # $t2 ← A[i][j]
mul    $t0, $s1, $t1 # $t0 ← Número de columnas*j
add    $t0, $t0, $s0 # $t0 ← Número de columnas*j+i
sll    $t0, $t0, 2    # $t0 ← 4*(Número de columnas*j+i)
add    $t0, $t0, $s5 # $t0 ← 4*(Número de columnas*j+i)+A (dirección de A[j][i])
sw     $t2, 0($t0)    # A[j][i] ← $t2 = A[i][j]
```

5. (0,5 puntos) Escriba en hexadecimal el contenido del registro \$s0 después de ejecutarse el programa siguiente:

```
.data
c:      .word    52, 49, 55, 48
        .text
        .globl  __start
```

Contenido final del registro \$s0 (hex.):

**0x34313730**

**Justificación:**

```
__start:
    la    $a0, c
    jal   X
    add   $s0, $v0, $zero
    addi  $v0, $zero, 10
    syscall

X:      addi $t0, $zero, 4
        add  $t1, $a0, $zero
        add  $v0, $zero, $zero
        addi $v0, $zero, 8
        lw   $t2, 0($t1)
        or   $v0, $v0, $t2
        addi $t1, $t1, 4
        addi $t0, $t0, -1
        bne  $t0, $zero, bucle
        jr   $ra

bucle:  sll  $v0, $v0, 8
        lw   $t2, 0($t1)
        or   $v0, $v0, $t2
        addi $t1, $t1, 4
        addi $t0, $t0, -1
        bne  $t0, $zero, bucle
        jr   $ra

X:      la    $a0, c           # $a ← Dirección c
        jal   X               # Llamada a función X
        add   $s0, $v0, $zero # $s0 ← $v0 (Valor de retorno)
        addi  $v0, $zero, 10  # Salida
        syscall
        addi  $t0, $zero, 4   # Función X: $t0 ← 4 (contador)
        add   $t1, $a0, $zero # $t1 ← $a0 = Dirección c
        add   $v0, $zero, $zero # $v0 ← 0
        sll   $v0, $v0, 8     # desplaza $v0 8 veces a la izqda
        lw    $t2, 0($t1)     # $t2 ← ($t1) (52 en 1a. pasada)
        or    $v0, $v0, $t2   # $v0 ← $v0 or $t2 (52 en 1a. p.)
        addi  $t1, $t1, 4     # Incrementa en 4 el apuntador $t1
        addi  $t0, $t0, -1    # Decrementa en 1 el contador $t0
        bne   $t0, $zero, bucle # Vuelve a bucle si $t0 != 0
        jr    $ra            # Retorno de la función
```

En la siguiente table se puede ver un resumen de la ejecución de la función (valores al final de cada pasada)

Número de pasada	Inicial	1	2	3	4
\$t0 (Contador)	4	3	2	1	0
\$t1 (Apuntador)	c	c+4	c+8	c+12	c+16
\$t2 (decimal)	-	52	49	55	48
\$t2 (hex.)	-	33	31	37	30
\$v0 (Resultado, hex.)	0	34	3431	343137	34313730

6. (0,4 puntos) Escriba en el recuadro un programa en ensamblador MIPS para solicitar un número entero por la consola mediante la frase "Introduzca un número entero: " y luego imprima en pantalla el cuádruplo de ese número.

**Emplee comentarios para dar las explicaciones oportunas.**

```
.data
Pet:      .asciiz  "Introduzca un número entero: "
Res1:     .asciiz  "\nEl cuádruplo de "
Res2:     .asciiz  " es "
        .globl  __start
        .text

__start:
    la    $a0, Pet           # $a0 ← Dirección cadena a imprimir (Pet)
    li    $v0, 4             # $v0 ← Código de llamada print-string (4)
    syscall
    li    $v0, 5             # $v0 ← Código de llamada read-int (5)
    syscall
    add   $s0, $v0, $zero    # $s0 ← $v0 = Valor del entero leído
    sll   $s1, $v0, 2        # $s1 ← $v0*4 = Valor del entero leído * 4
    la    $a0, Res1         # $a0 ← Dirección cadena a imprimir (Res1)
    li    $v0, 4             # $v0 ← Código de llamada print-string (4)
    syscall
    li    $v0, 1             # $v0 ← Código de llamada print-int (1)
    add   $a0, $s0, $zero    # $a0 ← $s0 = Entero leído para imprimirlo
    syscall
    la    $a0, Res2         # $a0 ← Dirección cadena a imprimir (Res2)
    li    $v0, 4             # $v0 ← Código de llamada print-string (4)
    syscall
    add   $a0, $s1, $zero    # $a1 ← $s1 = Entero leído * 4 para imprimirlo
    li    $v0, 1             # $v0 ← Código de llamada print-int (1)
    syscall
```

# MIPS Reference Data



①

## ARITHMETIC CORE INSTRUCTION SET

②

## OPCODE

/FMT FT /FUNCT

(Hex)

FOR- MAT OPERATION

NAME, MNEMONIC

Branch On FP True b21t FI if(FPcond)PC=PC+4+BranchAddr (4) 11/81/-

Branch On FP False b21f FI if(!FPcond)PC=PC+4+BranchAddr (4) 11/81/-

Divide div R Lo-R(s)/R(r); Hi-R(s)%R(r) 0/-/-/1a

Divide Unsigned divu R Lo-R(s)/R(r); Hi-R(s)%R(r) 0/-/-/1b

FP Add Single add.s FR F[r] = F[r] + F[r] 11/10/-0

FP Add Double add.d FR F[r] = F[r] + F[r] 11/11/-0

FP Compare Single c.s FR FPcond = (F[r] op F[r]) ? 1 : 0 11/10/-0

FP Compare Double c.d FR FPcond = (F[r] op F[r]) ? 1 : 0 11/11/-0

FP Divide Single div.s FR F[r] = F[r] / F[r] 11/10/-3

FP Divide Double div.d FR F[r] = F[r] / F[r] 11/11/-3

FP Multiply Single mul.s FR F[r] = F[r] \* F[r] 11/10/-2

FP Multiply Double mul.d FR F[r] = F[r] \* F[r] 11/11/-2

FP Subtract Single sub.s FR F[r] = F[r] - F[r] 11/10/-1

FP Subtract Double sub.d FR F[r] = F[r] - F[r] 11/11/-1

Load FP Single lwc1 I F[r] = M[R(r)+SignExtmm] (2) 31/-/-/-

Load FP Double ldc1 I F[r] = M[R(r)+SignExtmm] (2) 35/-/-/-

Move From Hi mfh1 R R[r] = Hi 0/-/-/10

Move From Lo mfl0 R R[r] = Lo 0/-/-/12

Move From Control mfc0 R R[r] = CR[r] 10/0/-/0

Multiply Unsigned mult R (Hi,Lo) = R[s] \* R[r] 0/-/-/18

Shift Right Arith. sra R (Hi,Lo) = R[r] &gt;&gt;&gt; shamt (6) 0/-/-/19

Store FP Single swc1 I M[R(r)+SignExtmm] = F[r] (2) 39/-/-/3

Store FP Double swd1 I M[R(r)+SignExtmm+4] = F[r] (2) 3d/-/-/-

Floating-Point Instruction Formats

Pseudoinstruction Set

Register Name, Number, Use, Call Convention

Reserved Across A Call?

Zero 0 The Constant Value 0 N.A.

Sat 1 Assembler Temporary No

\$v0-\$v1 2-3 Values for Function Results No

\$a0-\$a3 4-7 Arguments No

\$t0-\$t7 8-15 Temporaries No

\$s0-\$s7 16-23 Saved Temporaries Yes

\$t8-\$t9 24-25 Temporaries No

\$k0-\$k1 26-27 Reserved for OS Kernel No

\$gp 28 Global Pointer Yes

\$sp 29 Stack Pointer Yes

\$fp 30 Frame Pointer Yes

\$ra 31 Return Address Yes

Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy, Computer Organization and Design, 4th ed.

CORE INSTRUCTION SET

FOR- MAT OPERATION (in Verilog)

NAME, MNEMONIC

Add add R R[r] = R[s] + R[r] (1) 0/20hex

Add Immediate addi I R[r] = R[s] + SignExtmm (1,2) 8hex

Add Imm. Unsigned addiu I R[r] = R[s] + SignExtmm (2) 9hex

And and R R[r] = R[s] &amp; R[r] 0/21hex

And Immediate andi I R[r] = R[s] &amp; ZeroExtmm 0/24hex

Branch On Equal beq I if(R[s]==R[r]) PC=PC+4+BranchAddr (4) 4hex

Branch On Not Equal bne I if(R[s]!=R[r]) PC=PC+4+BranchAddr (4) 5hex

Jump j PC=JumpAddr (5) 2hex

Jump And Link jal J R[31]=PC-8; PC=JumpAddr (5) 3hex

Jump Register jr R PC=R[s] (5) 0/08hex

Load Byte Unsigned lbu I R[r] = (24'b0.M[R(r)] + SignExtmm)(7:0) (2) 24hex

Load Halfword Unsigned lhu I R[r] = (16'b0.M[R(r)] + SignExtmm)(15:0) (2) 25hex

Load Linked ll I R[r] = M[R(r)+SignExtmm] (2,7) 30hex

Load Upper Imm. lui I R[r] = (imm, 16'b0) fhex

Load Word lw I R[r] = M[R(r)+SignExtmm] (2) 23hex

Nor nor R R[r] = ~ (R[s] | R[r]) 0/27hex

Or or R R[r] = R[s] | R[r] 0/25hex

Or Immediate ori I R[r] = R[s] | ZeroExtmm (3) dhex

Set Less Than slt I R[r] = (R[s] &lt; R[r]) ? 1 : 0 0/24hex

Set Less Than Imm. slti I R[r] = (R[s] &lt; SignExtmm) ? 1 : 0 (2) 24hex

Set Less Than Imm. Unsigned sltiu I R[r] = (R[s] &lt; SignExtmm) ? 1 : 0 (2,6) bhex

Set Less Than Unsig. sltu R R[r] = (R[s] &lt; R[r]) ? 1 : 0 (6) 0/2bhex

Shift Left Logical sll R R[r] = R[r] &lt;&lt; shamt 0/00hex

Shift Right Logical srl R R[r] = R[r] &gt;&gt; shamt 0/02hex

Store Byte sb I M[R(r)+SignExtmm](7:0) = R[r](7:0) 28hex

Store Conditional sc I M[R(r)+SignExtmm] = R[r]; R[r] = (atomic) ? 1 : 0 (2,7) 38hex

Store Halfword sh I M[R(r)+SignExtmm](15:0) = R[r](15:0) 29hex

Store Word sw I M[R(r)+SignExtmm] = R[r] (2) 2bhex

Subtract sub R R[r] = R[s] - R[r] (1) 0/22hex

Subtract Unsigned subu R R[r] = R[s] - R[r] 0/23hex

BASIC INSTRUCTION FORMATS

R opcode rs rt rd shamt funct

I opcode rs rt immediate

J opcode address

Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy, Computer Organization and Design, 4th ed.

## Llamadas al sistema mediante syscall

Servicio	Código de llamada	Argumentos	Resultado
print-int	1	\$a0=entero	
print-float	2	\$f12=valor en FP32	
print-double	3	\$f12=valor en FP64	
print-string	4	\$a0=buffer	
read-int	5		Entero en \$v0
read-float	6		FP32 en \$f0
read-double	7		FP64 en \$f0
read-string	8	\$a0=buffer, \$a1=longitud	
exit	10		
print-char	11	\$a0=carácter	
read-char	12		Carácter en \$v0