EXAMEN ORDINARIO FUNDAMENTOS DE COMPUTADORES. Prácticas

Apellidos y nombre:	DNI:	Grupo:

1. (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*

2. (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]
```

3. (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:
```

4. (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
        ...
  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
```

Justificación:

Contenido final del registro `$f0`: **-2,5**

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)`

5. (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
```

6. (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
__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
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
```

Contenido final del registro `$s0` (hex.): **0x34313730**

Justificación:

```
            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
X:          addi $t0, $zero, 4      # Función X: $t0 ← 4 (contador)
            add  $t1, $a0, $zero    # $t1 ← $a0 = Dirección c
            add  $v0, $zero, $zero  # $v0 ← 0
bucle: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

MIPS Reference Data

CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION (in Verilog)	OPCODE / FUNCT (Hex)
Add	add	R R[rd] = R[rs] + R[rt]	(1) 0 / 20hex
Add Immediate	addi	I R[rt] = R[rs] + SignExtImm	(1,2) 8hex
Add Imm. Unsigned	addiu	I R[rt] = R[rs] + SignExtImm	(2) 9hex
Add Unsigned	addu	R R[rd] = R[rs] + R[rt]	0 / 21hex
And	and	R R[rd] = R[rs] & R[rt]	0 / 24hex
And Immediate	andi	I R[rt] = R[rs] & ZeroExtImm	(3) chex
Branch On Equal	beq	I if(R[rs]==R[rt]) PC=PC+4+BranchAddr	4hex
Branch On Not Equal	bne	I if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4) 5hex
Jump	j	J PC=JumpAddr	(5) 2hex
Jump And Link	jal	J R[31]=PC+8;PC=JumpAddr	(5) 3hex
Jump Register	jr	R PC=R[rs]	0 / 08hex
Load Byte Unsigned	lbu	I R[rt]={24'b0,M[R[rs]+SignExtImm](7:0)}	(2) 24hex
Load Halfword Unsigned	lhu	I R[rt]={16'b0,M[R[rs]+SignExtImm](15:0)}	(2) 25hex
Load Linked	ll	I R[rt] = M[R[rs]+SignExtImm]	(2,7) 30hex
Load Upper Imm.	lui	I R[rt] = {imm, 16'b0}	fhex
Load Word	lw	I R[rt] = M[R[rs]+SignExtImm]	(2) 23hex
Nor	nor	R R[rd] = ~ (R[rs] | R[rt])	0 / 27hex
Or	or	R R[rd] = R[rs] | R[rt]	0 / 25hex
Or Immediate	ori	I R[rt] = R[rs] | ZeroExtImm	(3) dhex
Set Less Than	slt	R R[rd] = (R[rs] < R[rt]) ? 1 : 0	0 / 2ahex
Set Less Than Imm.	slti	I R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2) ahex
Set Less Than Imm. Unsigned	sltiu	I R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6) bhex
Set Less Than Unsig.	sltu	R R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6) 0 / 2bhex
Shift Left Logical	sll	R R[rd] = R[rt] << shamt	0 / 00hex
Shift Right Logical	srl	R R[rd] = R[rt] >> shamt	0 / 02hex
Store Byte	sb	I M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2) 28hex
Store Conditional	sc	I M[R[rs]+SignExtImm] = R[rt]; R[rt] = (*atomic*) ? 1 : 0	(2,7) 38hex
Store Halfword	sh	I M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2) 29hex
Store Word	sw	I M[R[rs]+SignExtImm] = R[rt]	(2) 2bhex
Subtract	sub	R R[rd] = R[rs] - R[rt]	(1) 0 / 22hex
Subtract Unsigned	subu	R R[rd] = R[rs] - R[rt]	0 /23hex

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1'b0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct	
	31 26 25	21 20	16 15	11 10	6 5		0
I	opcode	rs	rt	immediate			
	31 26 25	21 20	16 15				0
J	opcode	address					
	31 26 25						0

ARITHMETIC CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION	OPCODE / FMT, FT / FUNCT (Hex)
Branch On FP True	bc1t	FI if(FPcond)PC=PC+4+BranchAddr (4)	11/8/1/--
Branch On FP False	bc1f	FI if(!FPcond)PC=PC+4+BranchAddr(4)	11/8/0/--
Divide	div	R Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	0/--/--/1a
Divide Unsigned	divu	R Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt] (6)	0/--/--/1b
FP Add Single	add.s	FR F[fd] = F[fs] + F[ft]	11/10/--/0
FP Add Double	add.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}	11/11/--/0
FP Compare Single	c.x.s*	FR FPcond = (F[fs] *op* F[ft]) ? 1 : 0	11/10/--/y
FP Compare Double	c.x.d*	FR FPcond = ({F[fs],F[fs+1]} *op* {F[ft],F[ft+1]}) ? 1 : 0	11/11/--/y
* (x is eq, lt, or le) (*op* is ==, <, or <=) (*y* is 32, 3c, or 3e)			
FP Divide Single	div.s	FR F[fd] = F[fs] / F[ft]	11/10/--/3
FP Divide Double	div.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}	11/11/--/3
FP Multiply Single	mul.s	FR F[fd] = F[fs] * F[ft]	11/10/--/2
FP Multiply Double	mul.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]}	11/11/--/2
FP Subtract Single	sub.s	FR F[fd]=F[fs] - F[ft]	11/10/--/1
FP Subtract Double	sub.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}	11/11/--/1
Load FP Single	lwc1	I F[rt]=M[R[rs]+SignExtImm] (2)	31/--/--/--
Load FP Double	ldc1	I F[rt+1]=M[R[rs]+SignExtImm+4] (2)	35/--/--/--
Move From Hi	mfhi	R R[rd] = Hi	0/--/--/10
Move From Lo	mflo	R R[rd] = Lo	0/--/--/12
Move From Control	mfc0	R R[rd] = CR[rs]	10/0/--/0
Multiply	mult	R {Hi,Lo} = R[rs] * R[rt]	0/--/--/18
Multiply Unsigned	multu	R {Hi,Lo} = R[rs] * R[rt] (6)	0/--/--/19
Shift Right Arith.	sra	R R[rd] = R[rt] >>> shamt	0/--/--/3
Store FP Single	swc1	I M[R[rs]+SignExtImm] = F[rt] (2)	39/--/--/--
Store FP Double	sdc1	I M[R[rs]+SignExtImm] = F[rt]; (2) M[R[rs]+SignExtImm+4] = F[rt+1]	3d/--/--/--

FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fmt	ft	fs	fd	funct
	31 26 25	21 20	16 15	11 10	6 5	0
FI	opcode	fmt	ft	immediate		
	31 26 25	21 20	16 15			0

PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	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

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