Resolución del ejercicio

1. Cuando acabe el programa anterior indicar en hexadecimal:

Contenido del registro $s2	
Contenido del registro $t2	
Dirección de memoria a la que accede para leer el dato la instrucción **lw $t3,8($s2)**	
Contenido de la dirección 0x10010007	
Valor almacenado en la variable resul	

El contenido del registro $s2 será la dirección de memoria de la variable num1, es decir, 0x10010000.

En el programa proporcionado, la instrucción la $s2, num1 carga en el registro $s2 la dirección de memoria donde se encuentra la variable num1.

La variable num1 se encuentra en la sección .data del programa y está definida con la directiva .word, lo que significa que es un entero de 32 bits. Además, se encuentra en la dirección de memoria 0x10010000.

Luego, la instrucción lw $t2,4($s2) carga en el registro $t2 el contenido de la dirección de memoria que se encuentra 4 bytes después de la dirección almacenada en $s2. Es decir, $t2 contiene el contenido de la dirección de memoria 0x10010004. $t3 apunta a 0x10010008 por la instrucción lw $t3,8($s2), que suma 8 bytes a la dirección a la que apunta $s2.

La instrucción "lw $t3,8($s2)" accede a la dirección de memoria apuntada por $s2 + 8 bytes, es decir, la dirección 0x10010008.

El contenido de la dirección de memoria 0x10010007 no se utiliza en el programa.

El valor almacenado en la variable resul será el resultado de la suma entre el valor de num1 y num2. que se guarda en la variable resul con la instrucción "sw $t4,12($s2)". Por lo tanto, el valor almacenado en la dirección de memoria 0x1001000C (que es la dirección de num1 + 3 palabras) será el resultado de la suma, que es 0x7009 en hexadecimal.

Suponer que ejecutamos el siguiente programa para el procesador MIPS.
Los datos están cargados en memoria a partir de la dirección 0x10010000

```
        .data
num1:   .word   0x7000
num2:   .word   9
num3:   .word   0x54ab00
resul:  .word   0


        .text
        .globl main
main:
        la   $s2, num1
        lw   $t2,4($s2)
        lw   $t3,8($s2)
        add  $t4,$t2,$t3
        sw   $t4,12($s2)
        addi $v0, $0,10
        syscall
```

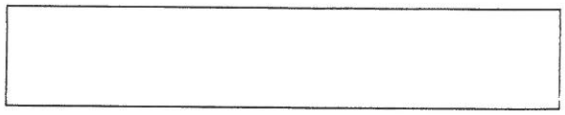
Este programa para el procesador MIPS realiza lo siguiente:

1. Se definen las variables num1, num2, num3 y resul en la sección .data de la memoria.
2. En la sección .text del programa, se define la función main.
3. La instrucción "la $s2, num1" carga la dirección de memoria de la variable num1 en el registro $s2.
4. La instrucción "lw $t2,4($s2)" carga el valor de la segunda palabra (4 bytes) después de la dirección de memoria apuntada por $s2 (que es la dirección de num1) en el registro $t2. El valor que se carga es 9.
5. La instrucción "lw $t3,8($s2)" carga el valor de la tercera palabra (8 bytes) después de la dirección de memoria apuntada por $s2 (que es la dirección de num1) en el registro $t3. El valor que se carga es 0x54ab00.
6. La instrucción "add $t4,$t2,$t3" suma los valores de $t2 (9) y $t3 (0x54ab00) y guarda el resultado en el registro $t4.
7. La instrucción "sw $t4,12($s2)" guarda el valor de $t4 en la cuarta palabra (12 bytes) después de la dirección de memoria apuntada por $s2 (que es la dirección de num1 + 3 palabras). Es decir, guarda el resultado de la suma en la variable resul.
8. La instrucción "addi $v0, $0, 10" carga el valor 10 en el registro $v0, que se utiliza para indicar al sistema operativo que se desea salir del programa.
9. La instrucción "syscall" realiza una llamada al sistema para terminar la ejecución del programa.

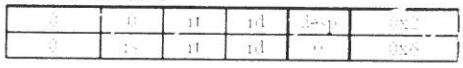
Por lo tanto, el programa carga los valores de num1 y num2 en registros, los suma y guarda el resultado en la variable resul. Luego, termina la ejecución del programa.

2. Indica cuál es la codificación en hexadecimal de la siguiente instrucción:

```
srl $t0,$t2,3
```

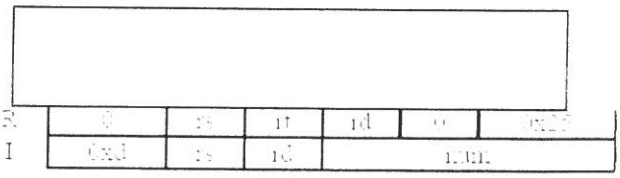


Desplazamientos lógicos y de rotación a derecha



3. ¿A qué instrucción del procesador MIPS se corresponde la siguiente palabra de 32 bits?

0x 3568fffb



Otros ejercicios

2. Indica cuál es la codificación en hexadecimal de la siguiente instrucción:

```
srl $t0,$t2,3
```

0x000a40c2

Desplazamientos lógicos y de rotación a derecha

They must add up to 32 bits

srl	rd, rt, desp	R	6	0	5	0	5 rt	5 rd	5 desp	0x6	rd = bits de rt desplazados desp posiciones
srlv	rd, rt, rs	R	0		rs		rt	rd	0	0x6	rd = bits de rt desplazados rs posiciones
srl	rdest, rori1, ori2	Ps									rdest = bits de rori1 desplazados ori2 posic
ror	rdest, rori1, ori2	Ps									rdest = bits de rori1 rotados ori2 posiciones

0000 0000 0000 1010 0100 0000 1100 0010

0 0 0 a 4 0 c 2

0x000a40c2

3. ¿A qué instrucción del procesador MIPS se corresponde la siguiente palabra de 32 bits?

0x 3568fffb

```
ori $t0,$t3,0xfffb
```

or	rd, rs, rt	R	0	rs	rt	rd	0	0x25	rd = rs or rt
ori	rd, rs, num	I	0xd	rs	rd	num			rd = rs or num

0011 0101 0111 1000 1111 1111 1111 1011

0xd B 8 F F F B

ori $t0, $t3, 0xfffb

rs rd num

sw, $ra, 0x0000($sp)

0xdeb7fffc

Resolución del ejercicio

1. Cuando acabe el programa anterior indicar en hexadecimal:

Contenido del registro $s2	0x10010000 Puntero
Contenido del registro $t2	0x9 Valor en memoria
Dirección de memoria a la que accede para leer el dato la instrucción **lw $t3,8($s2)**	0x10010008 $s2+8
Contenido de la dirección 0x10010007	0x00000000 Si ningún momento se utiliza.
Valor almacenado en la variable resul	0x54ab09

El contenido del registro $s2 será la dirección de memoria de la variable num1, es decir, 0x10010000.

En el programa proporcionado, la instrucción la $s2, num1 carga en el registro $s2 la dirección de memoria donde se encuentra la variable num1.

La variable num1 se encuentra en la sección .data del programa y está definida con la directiva .word, lo que significa que es un entero de 32 bits. Además, se encuentra en la dirección de memoria 0x10010000.

Luego, la instrucción lw $t2,4($s2) carga en el registro $t2 el contenido de la dirección de memoria que se encuentra 4 bytes después de la dirección almacenada en $s2. Es decir, $t2 contiene el contenido de la dirección de memoria 0x10010004. $t3 apunta a 0x10010008 por la instrucción lw $t3,8($s2), que suma 8 bytes a la dirección a la que apunta $s2.

La instrucción "lw $t3,8($s2)" accede a la dirección de memoria apuntada por $s2 + 8 bytes, es decir, la dirección 0x10010008.

El contenido de la dirección de memoria 0x10010007 no se utiliza en el programa.

El valor almacenado en la variable resul será el resultado de la suma entre el valor de num1 y num2, que se guarda en la variable resul con la instrucción "sw $t4,12($s2)". Por lo tanto, el valor almacenado en la dirección de memoria 0x1001000C (que es la dirección de num1 + 3 palabras) será el resultado de la suma, que es 0x7009 en hexadecimal.

Enunciado

Suponer que ejecutamos el siguiente programa para el procesador MIPS.
Los datos están cargados en memoria a partir de la dirección 0x10010000

```
        .data
num1:   .word   0x7000
num2:   .word   9
num3:   .word   0x54ab00
resul:  .word   0
```

```
        .text
        .globl main
main:
        la   $s2, num1
        lw   $t2,4($s2)
        lw   $t3,8($s2)
        add  $t4,$t2,$t3
        sw   $t4,12($s2)
        addi $v0, $0,10
        syscall
```

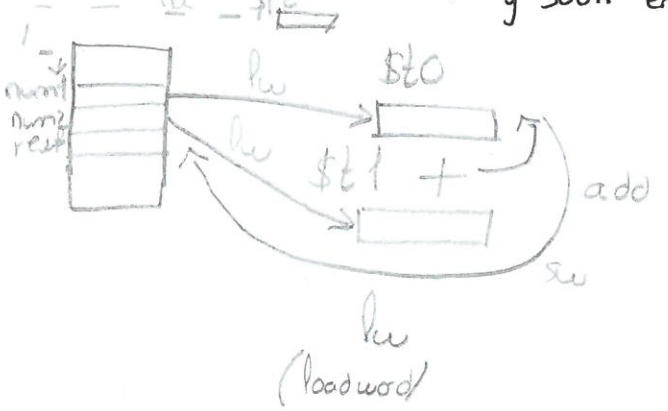
Explicación

Este programa para el procesador MIPS realiza lo siguiente:

1. Se definen las variables num1, num2, num3 y resul en la sección .data de la memoria.
2. En la sección .text del programa, se define la función main.
3. La instrucción "la $s2, num1" carga la dirección de memoria de la variable num1 en el registro $s2.
4. La instrucción "lw $t2,4($s2)" carga el valor de la segunda palabra (4 bytes) después de la dirección de memoria apuntada por $s2 (que es la dirección de num1) en el registro $t2. El valor que se carga es 9.
5. La instrucción "lw $t3,8($s2)" carga el valor de la tercera palabra (8 bytes) después de la dirección de memoria apuntada por $s2 (que es la dirección de num1) en el registro $t3. El valor que se carga es 0x54ab00.
6. La instrucción "add $t4,$t2,$t3" suma los valores de $t2 (9) y $t3 (0x54ab00) y guarda el resultado en el registro $t4.
7. La instrucción "sw $t4,12($s2)" guarda el valor de $t4 en la cuarta palabra (12 bytes) después de la dirección de memoria apuntada por $s2 (que es la dirección de num1 + 3 palabras). Es decir, guarda el resultado de la suma en la variable resul.
8. La instrucción "addi $v0, $0, 10" carga el valor 10 en el registro $v0, que se utiliza para indicar al sistema operativo que se desea salir del programa.
9. La instrucción "syscall" realiza una llamada al sistema para terminar la ejecución del programa.

Por lo tanto, el programa carga los valores de num1 y num2 en registros, los suma y guarda el resultado en la variable resul. Luego, termina la ejecución del programa.

3) 200h en memvar num1 y 300h en memvar num1




```
num1:   .word 0x200
num2:   .word 0x300
resultado: .word 0

        .text
        .globl main
main:   la $t2, num1
        lw $t0, 0($t2)
        lw $t1, 4($t2
        add $t0, $t0, $t1
        sw $t0, 8($t2)
        addi $v0, $0, 10
        syscall
```

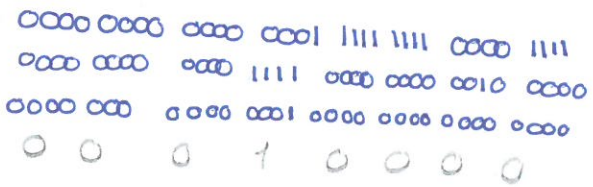
4) en $s0 esta 0x0001ff0f

$s1 0x000f0020

$s2 0x20

and

a) Si $t0, $s0, $s1 = ??



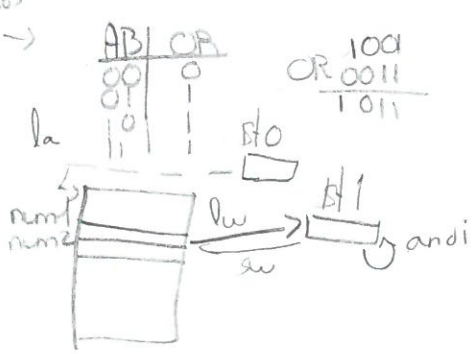
5) num1: 0x1234567f num2: 0xffffffff

Cero → AB | AND: 00→0, 01→0, 10→0, 11→1

AND 1001 / 0011 / 0001

Uno → AB | OR: 00→0, 01→1, 10→1, 11→1

OR 1001 / 0011 / 1011



✧ A cero bits 2 y 3 de num1

✧ A uno bits 3, 7 y 31 de num2

```
num1:   .word 0x1234567f
num2:   .word 0xffffffff
        .text
        .globl main
main:   la $t0, num1 -> Load address dvdo de num1
        lw $t1, 0($t0) -> Load word dvdo de num1 en $t1
        andi $t1, $t1, 0xfffffff3 -> parando en 0 2 y 3
        sw $t1, 0($t0)

        lw $t1, 4($t0) -> Lw de num2 en $t1
        li $t2, 0x7fffff77
        and $t1, $t2, $t2
        sw $t1, 4($t0)
```

```
# def variables
  .data
num1: .word 13
num2: .word 0x1500
resul: .word 0
```

RegDest => sies R=1 ; R≠0

ALUSic => Shag num=1 ; no num=0

MemtoReg =>

1/ 1 Mbyte = 1024 Kbytes?

1Mbyte = 2^{20} bytes

1024 Kbytes = $2^{10} \cdot 2^{10} = 2^{20}$ bytes.

4/ 46 Palab

46 pal. $2^2 \cdot 2^{30} = 2^{32}$ / necesito 32 bits

```
  .text
  .globl main
main:
  la $t
  lw $t2, 0($t1)
fin:
  addi $V0, $0, 10
  syscall
```

3/ 1 Tbyte = 1024 Mbytes?

1 Tbyte = 2^{40} bytes

1024 Mb = $2^{10} \cdot 2^{10} = 2^{30}$

3/ 1 Terapalabra?

10^{12} palabras

#salir del programa

4/ 64 gigapalab de 16 bits byte a byte bits necesarios?

64 Gb = 64 pal 2 bytes/pal = 64 Gbytes

126 Gbytes = $2^7 \cdot 2^{30} = 2^{37}$

1/ Suma 200h y 300h como valores inmediatos, guardado en resultado

```
  .data
resultado: .word 0      #almacena resul

  .text
  .globl main
main:
  addi $t0, $0, 0x200
  addi $t1, $t0, 0x300
  la $t2, resultado
  sw $t1, 0($t2)

  addi $v0, $0, 10
  syscall
```

#podria ser $t0 otravez ya que dejo de usarlo

2/ Suma 1024 y 30000h. guardado en "resultado"

```
  li $t0, 0x1024
  li $t1, 0x30000
  add $t0, $t0, $t1
  la $t1, resultado
  sw $t0, 0($t1)
```

0x00400024 0x0810000a ① guardar

31

000010 n nn.... n 0

0 0 0 1

0 8 1 0 0 0 0 1010

a

0 0 4 0 0 0 8

188

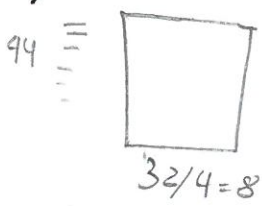
185

182

beq ⇒ salto si valor rs = rt

Exámenes prueba

1/ MAR 44 bits y MDR 32 bits. Máx cant de memoria? byte a byte



2^{44} direcciones · 1 byte/dirección $= 2^{40} + 2^4 = 16$ Tb

2/

37918S

730 539

12·50 = 600 + 4 = 604 · 10.000 = 6040000

16 millones

10

$2^{20} \cdot 2^4 = 2^{24}$ → Lineal

"Estructura de Computadores" MAYO 2019

ACTIVIDAD EVALUABLE PRACTICA CONSULTA DE ESTADO

(5% de la nota final)

Nombre:.....

1) El siguiente programa corresponde a una posible solución de la práctica de consulta de estado. Indicar:
   A) En qué número de línea está la instrucción que lee el dato introducido del teclado? __27____
   B) En qué número de línea está la instrucción que escribe el dato leido del teclado en el puerto del monitor? ____29____

2) Se ha rediseñado el sistema y las nuevas direcciones de los puertos son:
   - puerto de estado teclado 0xf2200 (bit estado es bit 6)(activo si 1)
   - puerto de datos teclado 0x0xf2204
   - puerto de estado monitor 0xf2208 (bit estado bit 9(activo si 1)
   - puerto de datos monitor 0xf220c

Indicar qué habría qué cambiarle al programa. Para ello al lado de las instrucciones que hay que modificar escribir las nuevas instrucciones

```
1  .data
2  cadena:      .space 100
3
4          .text
5          .globl main
6  main:
7          la $a0, cadena
8          jal leer_cadena
9
10         la $a0, cadena
11         add $a1, $v0, $0
12         jal mostrar_cadena
13
14         addi $v0, $0. 10
15         syscall
16
18 leer_cadena:
19         li $t0. 0xffff0000                  li $t0. 0xf2200
20         addi $t2,$0, 0x3b
21         addi $v0,$0. 0

22 kbready:
23         lw $t3. 0($t0)
24         andi $t3, $t3, 1                    andi $t3, $t3, 0x40
25         beq $t3, $0, kbready
26
27         lb $t3, 4($t0)
28         beq $t3, $t2, finteclado
29         sb $t3, 0($a0)
30         addi $a0. $a0, 1
31         addi $v0, $v0, 1
32         j kbready
33
34 finteclado:
```

```
35      jr $ra
36
37 mostrar_cadena:
38      li $t0, 0xffff0008                  li $t0,0xf2208
39 seguir:
40      beq
   $a1,$0,finmostrarcadena

41      addi $a1, $a1, -1
42 moready:
43      lw $t2, 0($t0)
44      andi $t2, $t2, 1                    andi $t2, $t2, 0x200
45      beq $t2, $0, moready
46
47      lb $t3, 0($a0)
48      addi $a0, $a0, 1
49      sb $t3, 4($t0)
50      j seguir
51 finmostrarcadena:
52      jr $ra
```

3) ¿Cuántas subrutinas tiene el programa? _7___
Indicar el nombre y número de línea donde empieza y donde acaba cada subrutina.
Leer_cadena:18-21
Kbready:22-32
Fin_teclado:34-35
Mostrar_cadena: 37-38
Seguir: 40-41
Moready: 42-50
Finmostrarcadena:51-52

4) Modificar la rutina mostrar_cadena para que sólo saque por el monitor aquellos caracteres que no sean el número 5 (código ASCII 0X35) y deje en el registro $v1 el número de caracteres que no ha mostrado. Recuadrar las nuevas instrucciones.

```
mostrar_cadena:
      li $t0, 0xffff0008
     li$v1,0
      li $t7,0x35
seguir:
      beq
$a1,$0,finmostrarcadena

      addi $a1, $a1, -1
moready:
      lw $t2, 0($t0)
      andi $t2, $t2, 1
      beq $t2, $0, moready

      lb $t3, 0($a0)
      addi $a0, $a0, 1
beq $t3, $t7.nomostrar
      sb $t3, 4($t0)
      j seguir

nomostrar:
```

```
        .globl main
main:
        la $a0, cadena
        jal leer_cadena

        la $a0, cadena
        add $a1, $v0, $0
        jal mostrar_cadena

        addi $v0, $0, 10        # salir del programa
        syscall
# rutina para leer caracteres del teclado

leer_cadena:
        li $t0, 0xffff0000      # dirección base de los puertos de E/S del teclado
        addi $t2,$0, 0x3b       # código de ESC
        addi $v0,$0, 0          # caracteres leidos
kbready:
        lw $t3, 0($t0)
        andi $t3, $t3, 1
        beq $t3, $0, kbready

        lb $t3, 4($t0)          # recoger el carácter del teclado
        beq $t3, $t2, finteclado
        sb $t3, 0($a0)          # almacenarlo en memoria
        addi $a0, $a0, 1
        addi $v0, $v0, 1        # incrementar el num. de caracteres leidos

        j kbready       # volver a esperar otro caracter
finteclado:
        jr $ra

# rutina para mostrar por pantalla una cadena de caracteres

mostrar_cadena:
        li $t0, 0xffff0008      # dirección base de los puertos de E/S del monitor
seguir:
        beq $a1,$0, finmostrarcadena  #termino cuando ya no quedan caracteres
        addi $a1, $a1, -1       #decremento contador de caracteres
moready:
        lw $t2, 0($t0)          #
        andi $t2, $t2, 1
        beq $t2, $0, moready

        lb $t3, 0($a0)          # leer el siguiente caracter a mostrar
        addi $a0, $a0, 1
        sb $t3, 4($t0)          # mostrar el carácter por pantalla
        j seguir

finmostrarcadena:
        jr $ra
```

PRÁCTICA 4.1 (usar la versión 4.12 de Simula3MS)
ENTRADA/SALIDA POR CONSULTA DE ESTADO

IMPORTANTE: Si habéis actualizado Java después de instalar el simulador tendréis que volver a instalar el simulador porque si no, no funciona el simulador con entrada/salida.
Antes de ejecutar el programa debemos configurar el simulador para entrada salida por encuesta

Objetivos:

- Aplicar los conceptos de entrada/salida.
- Aplicar el concepto de consulta de estado.

Desarrollo / Comentario:

La arquitectura MIPS utiliza E/S mapeada en memoria, en la que se emplean varias direcciones de memoria para referirse a los puertos de los dispositivos periféricos.

En Simula3MS la dirección base de la E/S mapeada en memoria es 0xffff0000. Dispone de dos dispositivos de E/S: el teclado (Entrada) y el monitor (Salida). Los puertos correspondientes a estos dispositivos son:

Teclado: Puerto de estado/control à 0xffff0000
Puerto de datos à 0xffff0004

Monitor: Puerto de estado/control à 0xffff0008
Puerto de datos à 0xffff000c

El bit 0 del registro de estado/control del teclado (llamado *ready*) cambia automáticamente de 0 a 1 cuando se introduce un carácter en el teclado. Este bit se pone automáticamente a 0 en el momento que el procesador recoge el carácter introducido (mediante una lectura del puerto de datos del teclado).

El bit 0 del registro de estado/control del monitor (llamado *ready*) indica si se ha terminado de escribir el carácter enviado a la pantalla. Si la pantalla está ocupada escribiendo el último carácter enviado, este bit estará a 0. Si ya se ha escrito el último carácter en la pantalla, el bit *ready* pasará a 1.

Primera parte: Observar el siguiente programa que lee un carácter introducido por el teclado y lo visualiza en el monitor

```
.text
        .globl main
main:

        li $t0, 0xffff0000                  # dirección base de los puertos de E/S

kbready:
        lw $t1, 0($t0)                      # leemos puerto de estado del teclado
        andi $t1, $t1, 1                    # aislamos el bit de estado del teclado
        beq $t1, $0, kbready                # preguntamos si esta activo
        lb $t2, 4($t0)                      # recoger el caracter del teclado (leemos del puerto de datos del teclado)

moready:
        lw $t3, 8($t0)                      # leemos puerto de estado del monitor
        andi $t3, $t3, 1                    # aislamos el bit de estado del monitor
        beq $t3, $0, moready                # preguntamos si está activo

        sb $t2, 12($t0)                     # mostrar el carácter por pantalla

        addi $v0, $0, 10                    # salir del programa
        syscall
```

a) ¿En qué registro de la CPU se almacena el dato leído del teclado? Sol: En $t2

b) Ejecuta el programa con el simulador. Pulsa la tecla 7. ¿qué valor se almacena en el registro de datos del monitor? Sol: 0x37

c) ¿Qué significa ese número? Sol: Es el codigo ASCII del numero 7.

d) ¿Cuántas teclas puedes visualizar? Sol: Solo 1

e) ¿Cómo modificaríamos el programa para poder seguir pulsando teclas y visualizarlas en el monitor de manera infinita? Sol: Pondría j kbready detras de sb $t2, 12($t0).

f) Suponer que rediseñamos el sistema hardware y el bit de estado del teclado es el bit 5 en vez de ser el bit 0 ¿qué tendríamos que modificar en el programa anterior?

Sol: Cambiaríamos la instruccion andi $t1, $t1, 1 por andi $t1, $t1, 0x20

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
Si pasamos esto a hexadecimal me da 0x20

Segunda parte: Empleando E/S por consulta de estado, se pide:

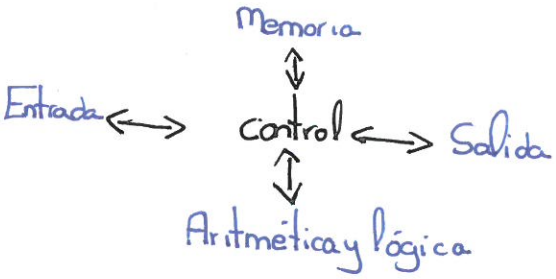
1. Escribe una función que lea una cadena de caracteres introducidos desde el teclado, los escriba en memoria a partir de la posición indicada en el registro $a0, y finalmente devuelva en el registro $v0 el número de caracteres leídos. La cadena termina cuando se pulsa la tecla ESC (código 0x3b).
2. Escribe una función que muestre por pantalla los datos que hay a partir de la dirección de memoria indicada en el registro $a0, teniendo en cuenta que la longitud de la cadena a mostrar está en el registro $a1.
3. Escribe un programa que llame a las dos funciones anteriores. El programa debe pasar al registro $a1 como argumento de la función del apartado 2 la salida $v0 de la función del apartado1.

Solución:

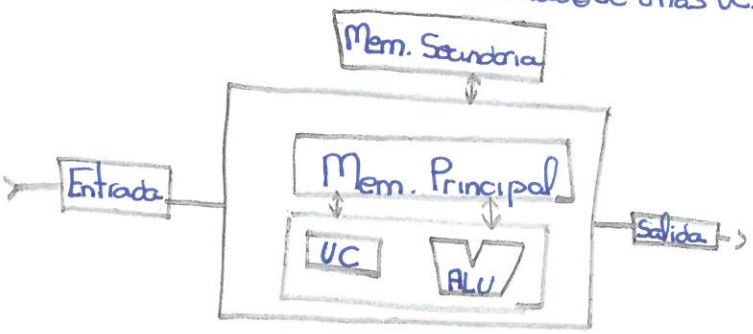
```
        .data
cadena: .space 100

        .text
```

Arquitectura Von Neumann (1945)



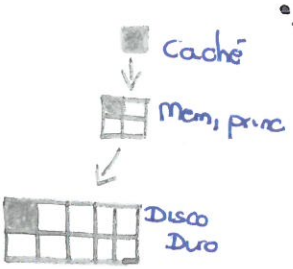
- Entrada/salida: Datos, envía mensajes. Monitores, discos duros, Wi-Fi
- Memoria: Almacena datos, programas... (Principal / Secundaria)
- ALU: Circuitos con operaciones aritméticas (Suma, resta...) y lógicas (AND, OR...)
- Unidad de control (UC): Recibe señales de estado de otras UCs, también envía

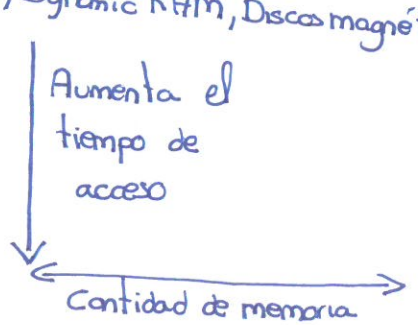


- Kilobyte: $(2^{10})^1$
- Megabyte: $(2^{10})^2$
- Gigabyte: $(2^{10})^3$
- Terabyte: $(2^{10})^4$

Unidad de memoria

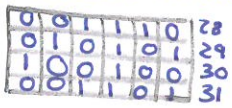
- Modo de acceso
  - → Aleatorio, llega directamente
  - → Secuencial, debe pasar por todas
  - → Asociativa, se usa el dato a buscar
- Volatilidad
  - → RAM, aleatorio, volátil (se va si se apaga) y permite operaciones
  - → ROM, ¡Sólo lectura!, aleatorio, no volátil
    - Ejemplos: Static RAM, Dynamic RAM, Discos magnéticas y ópticas
- Jerarquía





- Estructura interna

  Celdas → Almacenan un bit → En tamaño fijo (palabras), identificados con una dirección



18/ CPU 500 MHz ¿t encuentre tras 10 ciclos?

$t = 10 \cdot \frac{1}{500 \cdot 10^6} = 0{,}02 \cdot 10^{-6} = 0{,}02\ \mu s$ (c)

19/ 1. Falso
2. Falso, tmb datos
3. Falso
4. V
5. Falso, falta la fase de decodificación
6. V
7. F, faltan buses y registros
8. F
9. V
10. 64 direcciones = 2^{64} F

20/ Fases de ejecución de las instrucciones máquina
1. Captación
2. Decodificación
3. Ejecución

21/ Diferencia palabra de un computador y el tamaño del bus
ALU ↔ ↳ Memoria

22/ 20 bits → 2^{20} = 1 Mebibyte

64 kibites = $= 2^6 \cdot 2^{10} \cdot 2^{16}$

0000 0000 0000 0000
0 0 0 0
⋮
F F F F

F F F F F H
- F F F F H
F 0 0 0 0

$2^{32} = 2^2 \cdot 2^{30} = 2$ Gibyt

srl $t0, $t2, 3

0 0 rt rd desp 0x2

0000 0000 0000 1010 0100 000010

0x000a4

2

0x 35290000

0011 0101 1001 0000 0000 0000 0000

add $t3, $t1, $t2

0 rs rt rd 0 0x20

00 0000 01001 01010 01011 00000 10000

0 1 2 a 5 8 2 0

addi $t3 $t3 ffff

0x8 rs rd num

001000 01011 01011 1111 1111 1111 1111

0x 2 1 6 6 f f f f

sw $t0, 0($t1) sw rt, num(rs)

0x2b rs rt num

101011 01001 01000 0000 0000 0000 0000

0xa b28 0000

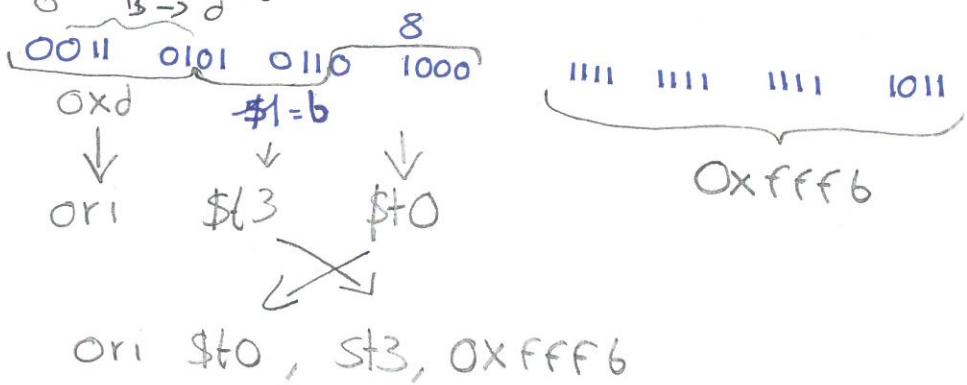
Ensamblador MIPS

Registros

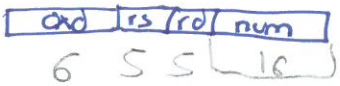
Suma	add a, b, c	a = b+c
Resta	sub a, b, c	a = b-c
Lectura	lw $s1 100($s2)	$s1 = Memoria [$s2 +100]
Escritura	sw $s1, 100($s2)	Memoria [$s2 + 100] = $s1

Cada instrucción MIPS = 32 bits de 0's y 1's

0x3568fffb



Ori rd, rs, num

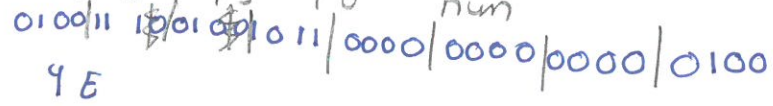


0x012a5820

0000 0001 0010 1010 0101 1000 0010 0000

lw $t3, 8($s2) lw rd, num(rs)

0x23



4 E

0x21290004

0010 0001 0010 1001 0000 0000 0000 0100

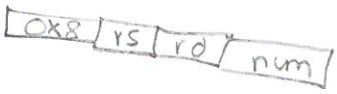
0x8 9 9

0x0004

addi $t1 $t1

addi $t1, $t1, 0x0004

addi rd, rs, num



0x8d2a0000

2 3 9 a 0 0 0 0

000010001101001010100000000000000000

0x23 $t1 $t2 0 0 0 0

$0

lw $t2, $0($t1)

lw rd, num(rs)

addi $t1, $t1, 4

001000 01001 01001 0000 0000 0000 0100

6 5 5 16

0x21290004

add $t0 $s1