PRÁCTICA 1

Esquema de contenidos

Introducir el lenguaje ensamblador y el manejo del simulador.
Contenidos básicos:

- Operaciones aritméticas sencillas con datos en registros
- Utilización de la memoria
- Llamadas al sistema: entrada/salida en consola, fin de ejecución de un programa

Parte 1: Operaciones aritméticas con datos cargados en registros

1. Introducir el código ensamblador equivalente al siguiente código C

```c
int main(void)
{
    register int a =  4660, b = 65244;
    printf("%d", a+b);
}
```

```
# segmento de texto
.text
.globl main

main:
    addi $a0, $zero, 4660   # Pone el primer entero en el registro $a0
    addi $a1, $zero, 65244  # Pone el segundo entero en el registro $a1

    add $a0, $a0, $a1       # Hace $a0 = $a0+$a1

    addi $v0, $zero, 1   # Codigo syscall 1 = imprime el entero en $a0
    syscall              # Llamada al sistema

    addi $v0, $zero, 10  # Codigo syscall 10 = Finaliza ejecución
    syscall              # Llamada al sistema
```

---

2. Ejecutarlo paso a paso y ver como se modifican los registros Modificanse $a0, $a1, $v0, PC e IR
3. Especificar el número de ciclos de ejecución 7 ciclos de ejecución
4. Comprobar que el entero que se imprime es el que está en el registro `$a0`

Pasando de hexadecimal a decimal o valor gardado no rexistro despois da execución da suma vemos que é o valor de saída da suma.

Parte 2: Operaciones aritméticas con datos en memoria

1. Introducir el código ensamblador equivalente al siguiente código C

```c
int a =  305419896, b = -19088744, c = 0;
int main(void)
{
    c = a + b;
    printf("%d", c);
}
```

```
# segmento de datos
.data
a:    .word 305419896
b:    .word -19088744
c:    .word 0

# segmento de texto
.text
.globl main

main:
    la $t0, a            # Carga la dirección de a en $t0
    lw $a1, 0($t0)       # Carga el valor contenido en a en $a1
    la $t0, b            # Carga la dirección de b en $t0
    lw $a2, 0($t0)       # Carga el valor contenido en b en $a2
    la $t0, c            # Carga la dirección de c en $t0
    lw $a0, 0($t0)       # Carga el valor contenido en c en $a0

    add $a0, $a1, $a2    # Hace $a0 = $a1+$a2

    sw $a0, 0($t0)       # Guarda el valor contenido en $a0 en
                         # la dirección indicada por $t0

    addi $v0, $zero, 1   # Codigo syscall 1 = imprime el entero en $a0
    syscall              # Llamada al sistema

    addi $v0, $zero, 10  # Codigo syscall 10 = Finaliza ejecución
    syscall              # Llamada al sistema
```

---

2. Análisis de la pseudo-instrucción *la*: Indicar la codificación equivalente con dos instrucciones especificando su significado

3. Cubrir la siguiente tabla para las 4 primeras instrucciones ejecutadas (las filas en blanco están reservadas para las dos instrucciones equivalentes a *la*):

A pseudo-instrucción la carga no rexistro indicado a variable almacenada na dirección de memoria indicada. Divídese en 2 intrucións lui e ori. Lui almacena os 16 bits + significativos e pon os 16 menos significativos a 0 e ori almacena os menos significativos e fai un or cos 16 bits menos significativos.

Foto para terminar la tabla

Código	Contenido PC	Contenido $t0	Contenido $a1	Contenido $a2
lui $t0, 0x1001	00400004			
ori $t0,$t0, 0x0000	00400008			
`lw $a1, 0($t0)`	0040000c			
lui $t0, 0x1001				
ori $t0, $t0 0x0004				
`lw $a2, 0($t0)`				

4. Analizar qué hace la instrucción `sw $a0, 0($t0)`. ¿En qué dirección de memoria se guarda el resultado de la suma?

Almacena en memoria en la posición obtenida de sumarle el desplazamiento a la dirección del registro $2, la palabra del registro $1. La dirección debe ser múltiplo de 4.
Na memoria almacenanse de 4 en 4 bytes e podense direccionar de 1 en 1 bytes. O 0 é o desprazamento e pódense poñer só multiplos de 4.

Parte 3: Impresión de strings

1. Introducir el código ensamblador equivalente al siguiente código C

```c
int main(void)
{
    printf("Hola mundo!");
}
```

```
#
# segmento de datos:
.data
cadena:  .asciiz "Hola Mundo!"

#
# segmento de texto
.text
.globl main

main:
     la $a0, cadena      # Dirección de la cadena en $a0
     addi $v0, $0, 4     # Codigo syscall 4 = imprimir cadena de texto
     syscall             # Llamada al sistema.

     addi $v0, $0, 10    # Codigo syscall 10 = Finaliza ejecución.
     syscall             # Llamada al sistema
```

---

2. Obtener los requerimientos de almacenamiento de la cadena de datos: número de palabras de 32 bits y posiciones de memoria utilizadas. Dibujar en papel el mapa de memoria correspondiente.

3. Interpretar el contenido de las direcciones de memoria. Para ello utilizar la siguiente tabla de códigos ASCII:

ASCII	H	o	l	a		M	u	n	d	o	!	\0
Decimal	72	111	108	97	32	77	117	110	100	111	33	0
Hexadecimal	48	6f	6c	61	20	4d	75	6e	64	6f	21	00

Parte 4: Entrada por teclado

Escribe un programa ensamblador equivalente al siguiente código C

```c
char s[10];
int main(void)
{
  scanf("%s", s);
  printf("%s", "Hola");
  printf("%s", s);
}
```

Es decir, pide un nombre por teclado e imprime primero "Hola" y después el nombre. Colocar la cadena de texto por teclado en un espacio de memoria (de 10 bytes) reservado con *.space*. Abordar los siguientes objetivos:

1. Analizar almacenamiento en memoria. Dibujar en papel el mapa de utilización de memoria.

2. En el segmento de datos, coloca ahora primero el string "Hola" y después el espacio de 10 bytes, pero definiendo "Hola" con *.ascii* en vez de con *.asciiz*. ¿Qué se imprime? ¿por qué?