



Reconocimiento-CompartirIgual 3.0
España (CC BY-SA 3.0 ES)

Introducción a la Ciberseguridad

Práctica 1: Ensamblador

Marta Beltrán Pardo

Miguel Calvo Matalobos

Agradecimientos (versiones anteriores): Isaac Martín de Diego y Alberto Fernández Isabel

Contenidos

Contenidos.....	2
1. Introducción.....	3
2. Material de la práctica	6
3. Normativa y evaluación	7
4. Consejos para hacer una memoria	8
5. Enunciado de la práctica	10
Instrucciones	10
Ejercicio 1.....	13
Ejercicio 2.....	14
Ejercicio 3.....	15
Ejercicio 4.....	17
5. Anexo I.....	18

1. Introducción

Ya hemos estudiado en las primeras unidades de la asignatura que los computadores ejecutan programas (cuyo objetivo es resolver un problema determinado). Estos programas, están compuestos por secuencias de instrucciones y se escriben utilizando lenguajes de programación. Las secuencias de instrucciones pueden ser escritas o bien por personas (programadores), o generados de forma automática mediante las herramientas adecuadas.

Los programas se encuentran cargados en memoria principal. En la Figura 1 puede observarse el proceso de traducción desde un programa en un lenguaje de alto nivel (por ejemplo, C), hasta que es cargado en la memoria para su ejecución. A modo resumen, podemos decir que **un lenguaje de programación es una notación formal para describir algoritmos o funciones que serán ejecutados por un computador.**

A su vez, **los lenguajes de programación pueden dividirse en lenguajes de alto nivel** (los cuales facilitan la tarea de los programadores, ya que se encuentran más próximos a la forma de pensar y de expresarse de los humanos) y en **lenguajes de bajo nivel** (ceranos a la arquitectura de la máquina y, normalmente, más difíciles de comprender para los programadores). Dentro de los lenguajes de bajo nivel encontramos el lenguaje máquina, el cual es el único que la circuitería de la máquina puede interpretar; sus instrucciones se encuentran codificadas en binario (0s y 1s).

El **lenguaje ensamblador** es un lenguaje de bajo nivel diseñado para sacar el máximo partido a las características físicas de un computador. Sus principales peculiaridades son:

- Dependencia absoluta de la arquitectura del computador.
- Imposibilidad de transportar programas entre distintas máquinas, salvo que sean de la misma familia o compatibles entre sí.
- Programas muy largos (con gran número de instrucciones en comparación con los programas escritos en lenguajes de alto nivel).
- Códigos de operación, datos y referencias expresados en binario.

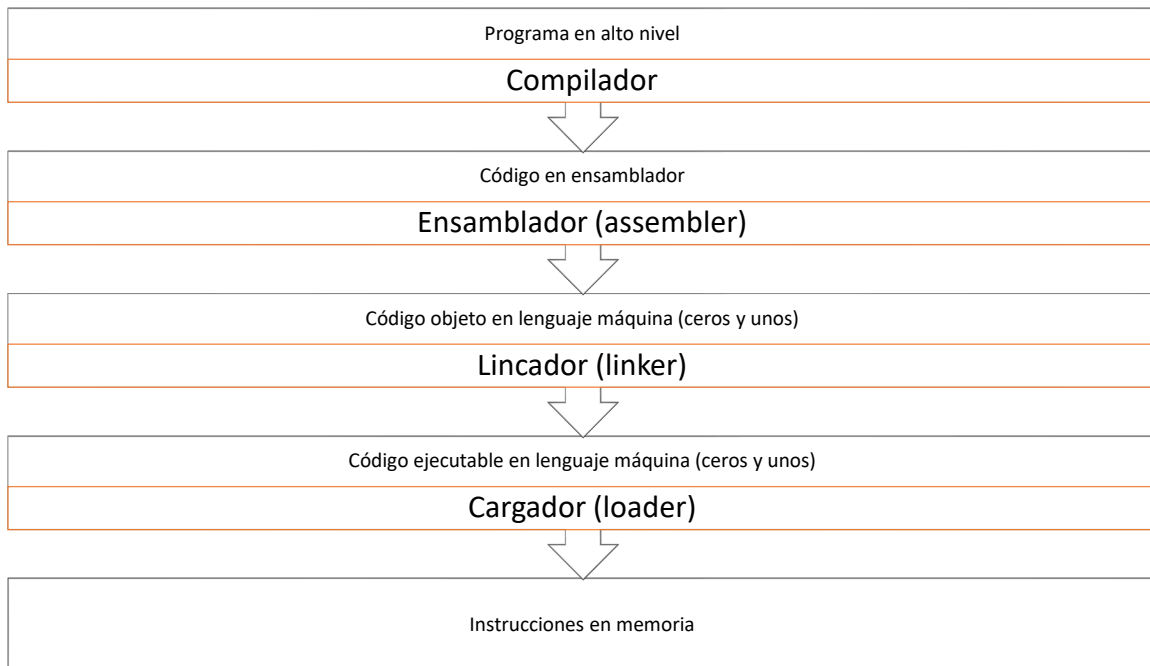


Figura 1. Proceso de traducción (programa en lenguaje alto nivel a memoria).

Por otro lado, el MIPS (Microprocessor without Interlocked Piepelines Stages) es un tipo específico de microprocesador desarrollado por MIPS Technologies. La arquitectura del MIPS es RISC (Reduced Instruction Set Computer), cuyas características principales son:

- Sus instrucciones son de tamaño fijo y se presentan en un reducido número de formatos.
- Sólo las instrucciones de carga y almacenamiento acceden a la memoria.
- Suelen disponer de muchos registros de propósito general.

Además, el MIPS puede encontrarse con un tamaño de registro de 32 bits (MIPS32) o con un tamaño de registro de 64 bits (MIPS64). Los procesadores MIPS se han utilizado en gran cantidad de productos como sistemas empuotrados, dispositivos para Windows CE, routers CISCO, videoconsolas, etc., y, además, suelen utilizarse en el ámbito universitario para el aprendizaje de los

lenguajes ensamblador. El nanoMIPS, que hemos estudiado en clase, es sólo una versión pedagógica de estos procesadores con un repertorio de instrucciones muy reducido.

Para facilitar el desarrollo y el aprendizaje sobre MIPS32, la Universidad de Missouri State ha desarrollado el simulador MARS.

2. Material de la práctica

A continuación, se exponen los archivos necesarios para la realización de esta práctica, que pueden descargarse desde el Aula Virtual:

- **Practica1_Guion.pdf:** este documento. Se corresponde con el guion de la práctica y en él están contenidas todas las explicaciones necesarias para su realización. Además, puede consultarse el apartado "Consejos para hacer una memoria" para
- **Practica1_CodigoEjemplos.zip:** programas sencillos (con extensión .asm y .c) sobre los cuáles se deberá trabajar en esta práctica. En este mismo guion, en cada uno de los ejercicios propuestos del apartado "Enunciado de la práctica", se indica el archivo o archivos necesarios para su realización.
 - `Practica1_example1.asm`, `Practica1_example1.c`, `Practica1_example1.7_resuelto.c`. Estos archivos deben utilizarse en la realización del ejercicio 1 de esta práctica.
 - `Practica2_example2.asm`, `Practica2_example2.c` y `Practica1_example2.4_resuelto.c`. Estos archivos deben utilizarse en la realización del ejercicio 2 de esta práctica.
 - `Practica3_example3.1.asm` y `Practica3_example3.2.asm`. Estos archivos deben utilizarse en la realización del ejercicio 2 de esta práctica.
- **Practica1_MIPS32.pdf:** hoja de instrucciones del MIPS32 (con su repertorio completo).

Además, será necesario descargar el programa MARS en su última versión. Este programa, servirá para ejecutar los códigos de ejemplo proporcionados junto al enunciado de esta práctica (los que tienen extensión ".asm") y para realizar las diferentes tareas solicitadas. Puede descargarse en su última versión desde el [siguiente enlace](#). Para ejecutarlo, será necesario tener instalado el entorno de ejecución de Java en el ordenador (puedes descargarse [aquí](#)).

3. Normativa y evaluación

En este apartado se detalla el formato de entrega de la práctica y la forma en la que se evaluará la misma:

- El porcentaje de la nota final de la asignatura al que corresponde esta práctica puede consultarse en la Guía docente de la propia asignatura.
- La práctica deberá realizarse, de forma obligatoria, en grupos de dos personas. Para la asignación de los grupos se deberán seguir las indicaciones del profesor.
- Cada grupo deberá:
 - Realizar una única memoria (puede utilizarse esta misma guía como plantilla) en la que responda las preguntas planteadas y/o en la que se exponga, de forma argumentada, las decisiones tomadas para la realización de la práctica.
 - Desarrollar y/o modificar tantos archivos de código como se soliciten en los diferentes ejercicios que forman la práctica.
- El resultado de la realización de la práctica consistirá en un fichero .zip llamado **Practica1.zip** que deberá entregarse a través del Aula Virtual en el espacio habilitado para ello y en la fecha límite allí indicada. Este fichero debe contener:
 - La memoria en formato PDF. En ella debe indicarse, en la primera página y de forma clara, el nombre y apellidos de los integrantes del grupo.
 - Los archivos de código resultantes de la realización de los ejercicios solicitados:
 - Practica1_example1.7_resuelto.asm
 - Practica1_example2.3_resuelto.asm
 - Practica1_example2.4_resuelto.asm
 - Practica1_example3.1_resuelto.asm
 - Practica1_example3.2_resuelto.asm
 - Practica1_example4.2_resuelto.asm

4. Consejos para hacer una memoria

A continuación, se exponen algunos consejos que ayudarán a mejorar el resultado final de un documento de memoria de un trabajo y/o práctica:

- **PORTADA.** En ella debe detallarse, de forma clara, el título de la práctica o el trabajo, el nombre y apellidos de la persona o personas que realizan el trabajo/práctica, los estudios que cursa y la asignatura, el curso académico, etc.
- **TÍTULOS.** Los títulos y subtítulos de las diferentes secciones o capítulos deben ser claros, concisos y descriptivos. Idealmente, los títulos comenzarán en nueva página, no siendo esto necesario para los subtítulos. Microsoft Word ofrece diferentes estilos de título (“Inicio” → “Estilos”).
- **ÍNDICE.** Un documento de memoria debe llevar un índice o tabla de contenido. Esta, suele situarse justo después de la portada y en ella se indican las diferentes secciones o capítulos (títulos y subtítulos) y las páginas en las que se encuentran. Microsoft Word ofrece la generación automática de índices (“Referencias” → “Tabla de contenido”).
- **PAGINACIÓN.** El documento debe tener sus páginas numeradas (salvo la portada). Para ello, puede utilizarse, en Microsoft Word, la opción “Número de página” (“Insertar” → “Número de página”).
- **TEXTO.** El texto de la memoria debe estar justificado, con un buen espaciado entre párrafos y, para mejorar la lectura, una tipografía, un tamaño de letra y un interlineado adecuados (por ejemplo, Arial 11 e interlineado a 1,2). También debe tenerse en cuenta que, si el texto está escrito en español, cualquier palabra en otro idioma diferente deberá escribirse en cursiva. Además, no deberían existir errores ortográficos y/o gramaticales a lo largo del documento.
- **FIGURAS Y TABLAS.** Las figuras/imágenes y las tablas deben numerarse y rotularse (para ello, Microsoft Word ofrece la opción “Referencias” → “Insertar título”). Además, idealmente, tanto las figuras/imágenes como las tablas se referenciarán a lo largo del texto de la memoria (por

ejemplo: “En la figura X puede observarse el resultado de este experimento”), dándole de esta forma un sentido y una explicación textual a la propia figura/imagen/tabla.

- **CITAS y REFERENCIAS.** Si se consultan recursos (artículos, revistas, libros, páginas web, apuntes, etc.) y/o se debe indicar en el texto del documento de dónde se ha extraído cierta información (por ejemplo, una figura, un párrafo, etc.), estos deben citarse utilizando un estilo de cita normalizado (por ejemplo, mediante la herramienta “Insertar Cita” de Microsoft Word con estilo IEEE o APA). En caso de que en el documento se incluyan párrafos literales extraídos de alguna de las referencias, estos deberán entrecomillarse (por ejemplo: “Todo debería ser hecho lo más simple posible, pero no más simple.”).
- **BIBLIOGRAFÍA.** En relación con el punto anterior, todas las citas y referencias deben aparecer en un apartado de bibliografía (que suele situarse al final del documento). En él se mostrarán los detalles de cada una de esas citas/referencias que han ido insertándose a lo largo de la memoria. Microsoft Word permite insertar una bibliografía automáticamente (siempre y cuando las citas y referencias se hayan insertado tal y como se ha indicado en el punto anterior); para ello: “Referencias” → “Citas y bibliografía” → “Bibliografía”.
- **DOCUMENTO FINAL.** Para facilitar la lectura en cualquier sistema operativo y/o software sin pérdida de elementos de formato y/o desconfiguraciones/modificaciones inesperadas, la memoria debe presentarse en PDF (salvo que se indique lo contrario).

5. Enunciado de la práctica

En este apartado se describirán las distintas actividades, programas y ejercicios que deberá realizar cada grupo de prácticas, así como la información a completar y/o rellenar en la memoria.

Instrucciones

Antes de comenzar a realizar la práctica, es necesario ejecutar los códigos de ejemplo proporcionados junto al enunciado desde el simulador [MARS](#). Se trata de que te familiarices con el simulador y de que comprendas cómo se ejecutan las diferentes instrucciones, que etapas atraviesan, qué resultados producen, etc.

Para ello, una vez abierto el simulador, en el menú “File”, con la opción “Open”, debe seleccionarse el archivo .asm que se quiere abrir. El código en ensamblador se mostrará en la pestaña “Edit” y los registros del procesador pueden encontrarse a la derecha de la pantalla. Véase la Figura 2.

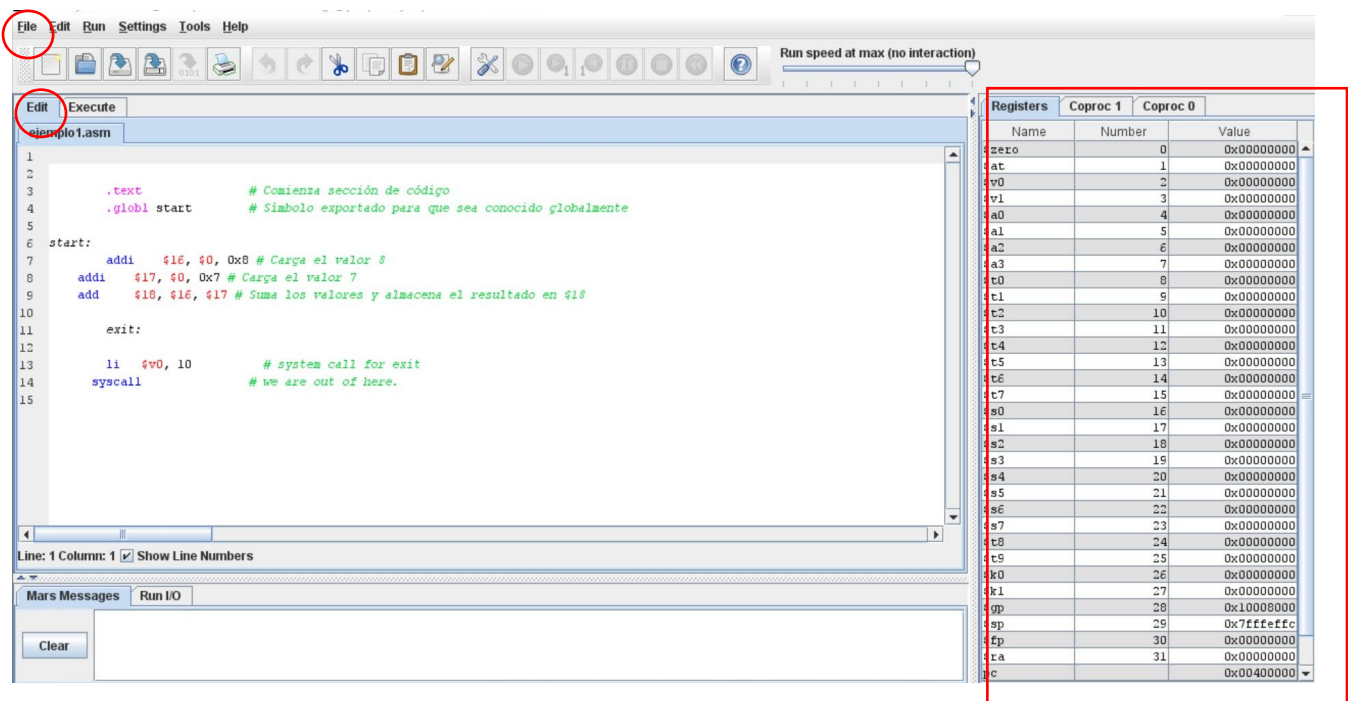


Figura 2. Abrir un archivo .asm desde el simulador MARS.

En el menú "Run", mediante la opción "Assemble", puede ensamblarse el código. Si existe algún error, este aparecerá detallado en la ventana de mensajes "Mars Messages", en la parte inferior del simulador.

Para ejecutar el código, una vez ensamblado, debe elegirse la opción "Go", también dentro del menú "Run" (si quieren ejecutarse todas las instrucciones) o "Step" (si quiere ejecutarse instrucción por instrucción para poder observar qué sucede con la ejecución de cada una de ellas). Véase la Figura 3.

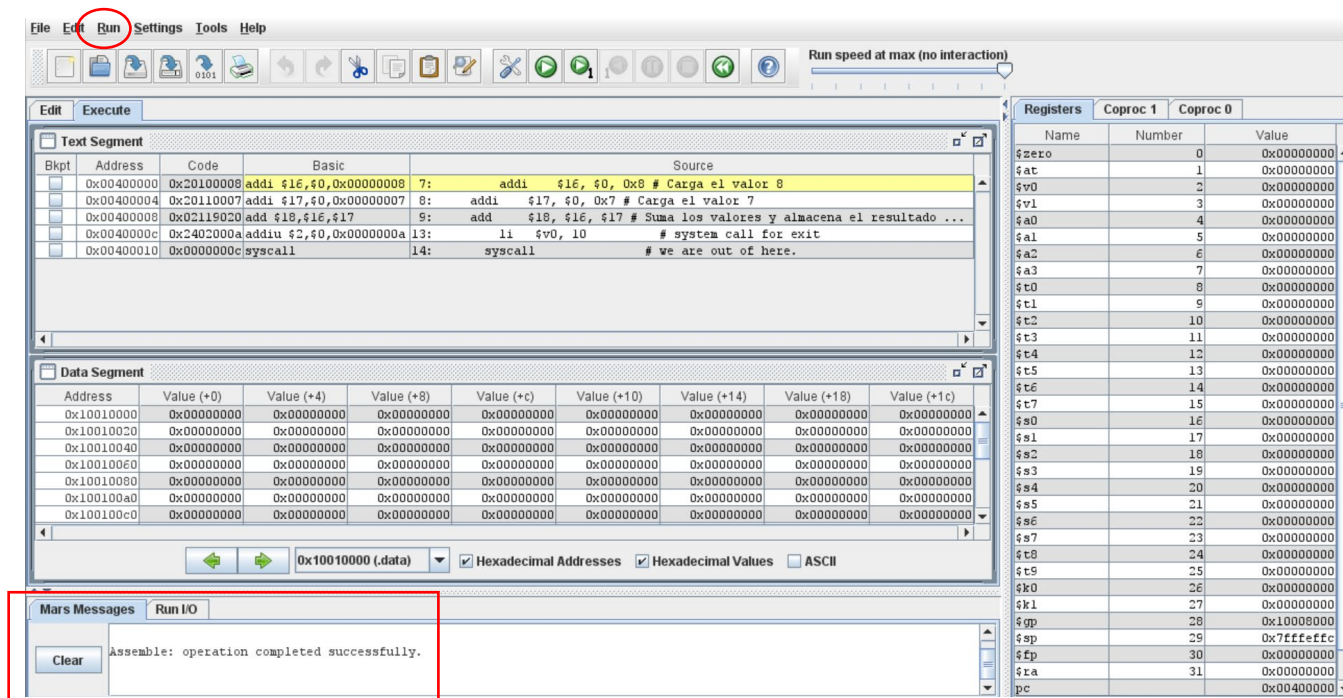


Figura 3. Ensamblado y ejecución en el simulador MARS.

Al ensamblar el código, se mostrará una nueva pestaña (en lugar de la pestaña "Edit"), "Execute", en la que puede verse el contenido de la memoria de instrucciones y la memoria de datos (recuerda que en esta arquitectura están separadas). Pueden verse las direcciones de memoria y sus contenidos en diferentes formatos estándar, según sea más cómodo.

Además, si el código ejecutado tiene algo de entrada/salida (E/S en español, I/O en inglés) como uso de teclado o de ratón, por ejemplo, esta información se mostrará en la pestaña "Run I/O", en la parte inferior del simulador (al lado de "Mars Messages").

Ejercicio 1

Para la realización de este ejercicio será necesario el código de ejemplo número 1 (`Practica1_example1.asm` y `Practica1_example1.7_resuelto.c`), proporcionado junto al enunciado de esta práctica. Este código calcula el área de un rectángulo y muestra el resultado.

Para ayudar a la comprensión de este código, además, puede observarse el fichero `Practica1_example1.c`, que se corresponde con el código proporcionado pero escrito en lenguaje C.

1. ¿Qué diferencia hay entre las instrucciones “add”, “addi” y “li”?
2. ¿Qué significan en el simulador las notaciones “\$” y “0x”?
3. ¿Se guarda alguna información en memoria? Justifica tu respuesta tanto si crees que no se guarda información en memoria como si consideras que sí lo hace. Además, si la respuesta es sí, ¿qué información se almacena en memoria y en qué zona y dirección? ¿por qué?
4. Ejecuta el código instrucción por instrucción e intenta comprender cómo van cambiando los valores de los diferentes registros en cada paso. Si se sustituyen en el código “\$s0”, “\$s1”, “\$s2” por “\$16”, “\$17” y “\$18” respectivamente, ¿Existe algún cambio en los valores que se almacenan en los registros y/o en el resultado final? ¿Por qué ocurre esto?
5. ¿Qué es una llamada al sistema o *syscall*? ¿Cómo se ejecuta en ensamblador? ¿Cuál es la llamada al sistema con código 1 en el MIPS32? ¿Qué código de llamada al sistema debería utilizarse para leer un entero?
6. ¿Qué es el registro pc y qué almacena? ¿Puedes modificarlo? ¿Qué es el registro zero y qué almacena? ¿Puedes modificarlo?
7. Modifica el código proporcionado para que, en lugar de calcular el área de un rectángulo, calcule el área de un triángulo: $(\text{base} \times \text{altura}) / 2$, tal y como lo hace el ejemplo proporcionado en C (`Practica1_example1.7_resuelto.c`). El fichero que debes entregar se llamará `Practica1_example1.7_resuelto.asm`.

Ejercicio 2

Para la realización de este ejercicio será necesario el código de ejemplo número 2 (`Practica1_example2.asm`), proporcionado junto al enunciado de esta práctica.

Para ayudar a la comprensión de este código, además, puede observarse el fichero `Practica1_example2.c`, que se corresponde con el código proporcionado pero escrito en lenguaje C.

1. ¿Qué secuencia de control implementa el programa `Practica1_example2.asm`?, ¿Qué hace el código proporcionado (`Practica1_example2.asm`)? ¿Cuál es el resultado devuelto por pantalla? ¿Y si se modifica el contenido de la variable “max” (\$s1) con el valor 101?
2. ¿Se guarda alguna información en memoria? Justifica tu respuesta tanto si crees que no se guarda información en memoria como si consideras que sí lo hace. Además, si la respuesta es sí, ¿qué información se almacena en memoria y en qué zona y dirección? ¿por qué?
3. Intenta entender cómo van cambiando los valores de los diferentes registros al ir ejecutando el código instrucción por instrucción. Una vez comprendido el funcionamiento del código, modifícalo para que, en lugar de sumar los números pares, sume los números impares. El fichero que debes entregar con los resultados de este ejercicio se llamará `Practica1_example2.3_resuelto.asm`.
4. Trata de modificar el código para que, en lugar de sumar los valores de los números pares, el código devuelva la cantidad de números impares y de números pares que hay (en dos sumas diferentes), tal y como lo hace el ejemplo proporcionado en C (`Practica1_example2.4_resuelto.c`). El fichero que debes entregar con los resultados de este ejercicio se llamará `Practica1_example2.4_resuelto.asm`.

Ejercicio 3

Para la realización de este ejercicio será necesario el código de ejemplo número 3 (Practica1_example3.1.asm y Practica1_example3.2.asm), proporcionados junto al enunciado de esta práctica. Con este programa, aprenderás el manejo de arrays en lenguaje ensamblador.

1. En primer lugar, y tras analizar el código del programa facilitado para este primer apartado del ejercicio (Practica1_example3.1.asm), trata de explicar, de forma detallada y apoyándote en el pseudocódigo, lo que hace dicho programa. Añade comentarios también al código original (el fichero que debes entregar se llamará Practica1_example3.1_resuelto.asm).

Nota: Un pseudocódigo expresa los pasos que se siguen para resolver un problema sin hacerlo en un lenguaje de programación concreto. Un ejemplo de pseudocódigo puede verse en la Figura 4.

```
inicio
  leer(n)
  pares = 0
  impares = 0
  i = 1
  hacer
    inicio
      si i%2 == 0
        pares = pares + 1
      sino
        impares = impares + 1
      i = i + 1
    fin
  mientras i <= n
  imprimir(pares, impares)
fin
```

Figura 4. Ejemplo de pseudocódigo.

2. El siguiente apartado consiste en analizar y comprender el segundo programa facilitado para la realización de este ejercicio (Practica1_example3.2.asm), que cuenta el número de caracteres de una frase. A este programa, para poder funcionar, le faltan únicamente dos líneas de código (indicadas mediante un comentario en el fichero .asm facilitado).

Debes completarlas y explicar el motivo de tu elección. El fichero que debes entregar, correspondiente al programa que cuenta el número de caracteres de una frase totalmente funcional, se llamará `Practica1_example3.2_resuelto.asm`.

Ejercicio 4

Para la realización de este ejercicio no se proporciona ningún código de ejemplo. Este ejercicio consiste en realizar, desde cero, un programa que:

- Pida, una a una, tres notas numéricas de un alumno (tres números enteros: n_1 , n_2 y n_3). Estos números serán solicitados por el programa al usuario, que los introducirá por teclado en cada ejecución.
- Calcule la media de esas tres notas: $((n_1 + n_2 + n_3) / 3)$.
- Muestre por pantalla, mediante un texto, si el alumno está aprobado (cuando la media sea mayor o igual a 5) o suspenso (cuando la media sea menor que 5).

1. **Escribe el pseudocódigo del programa propuesto.**
2. **Traduce el pseudocódigo creado para el punto anterior a código ensamblador. No olvides comentar el código. El fichero que debes entregar se llamará `Practica1_example4.2_resuelto.asm`.**
3. **Prueba el funcionamiento del programa (introduciendo diferentes notas, notas no numéricas, decimales, etc.). No olvides adjuntar pantallazos de la demostración.**

5. Anexo I

En este apartado se incluyen los códigos, scripts y útiles necesarios para la realización de los ejercicios de esta práctica.

Practica1_example1.c

```
#include <stdio.h>

int main()
{
    int base=8;           // Carga el valor 8 - base
    int altura=5;         // Carga el valor 5 - altura
    int areaRect=base*altura; // Multiplica los valores y almacena el resultado en $s2 - área rectángulo

    printf("%d",areaRect);
}
```

Practica1_example1.asm

```
.text                # Comienza sección de código
.globl start         # Símbolo exportado para que sea conocido globalmente

start:
    li $s0, 0x8       # Carga el valor 8 - base
    li $s1, 0x5       # Carga el valor 5 - altura
    mul $s2, $s0, $s1 # Multiplica y almacena el resultado en $s2 - área rectángulo

    move $a0, $s2      # Se carga el area para mostrarlo
    li $v0, 1          # Se carga el número de llamada al sistema "print integer"
    syscall            # Llamada al sistema

exit:

    li $v0, 10         # Se carga el número de llamada al sistema para salir
    syscall            # Llamada al sistema
```

Practica1_example1.7_resuelto.c

```
#include <stdio.h>

int main()
{
    int base=8;           // Carga el valor 8 - base
    int altura=5;         // Carga el valor 5 - altura
    int areaRect=base*altura; // Multiplica los valores y almacena el resultado en $s2 - área rectángulo.

    int areaTrian=areaRect/2; // Divide el área del rectángulo entre 2 - área triángulo.

    printf("%d",areaTrian);
}
```

Practica1_example2.c

```
#include <stdio.h>

int main(){
    int i = 0;
    int suma = 0;
    int resto = 0;

    for(i=0; i < 51; i = i+1){
        resto = i % 2;
        if (resto == 0){
            suma = suma+i;
        }
    }
    printf("%d ",suma);
    return 0;
}
```

Practica1_example2.asm

```
.text                                # Comienza sección de código
.globl start                         # Símbolo exportado para que sea conocido globalmente

start:
    li    $s0, 0                    # i = 0
    li    $s1, 51                   # max = 51
    lw    $t0, sumatorio            # suma = 0

loop:
    addi $s0, $s0, 1                # i = i + 1
    beq $s0, $s1, end              # cuando i > max, termina

    div $s3, $s0, 2                 # i / 2 (para ver si es par).
    mfhi $t1                        # Se guarda el resto de la división.
    beq $t1, $zero, suma           # Si el resto es 0 (par), lo suma.

    j     loop                     # Vuelta al bucle.

suma:
    add    $t0, $t0, $s0            # sum = sum + i
    sw $t0, sumatorio              # Se carga el nuevo valor del sumatorio.
    j     loop                     # Vuelta al bucle.

end:
    lw $a0, sumatorio              # Se carga el sumatorio para mostrarlo
    li $v0, 1                      # Se carga el número de llamada al sistema "print integer"
    syscall                        # Llamada al sistema

exit:
    li $v0, 10                     # Se carga el número de llamada al sistema para salir
    syscall                        # Llamada al sistema

.data
sumatorio: .word 0
```

Practica1_example2.4_resuelto.c

```
#include <stdio.h>

int main(){
    int i = 0;
    int contP = 0;
    int contI = 0;
    int resto = 0;

    for(i=1; i < 51; i = i+1){
        resto = i % 2;
        if (resto == 0){
            contP = contP+1;
        } else {
            contI = contI+1;
        }
    }
    printf("%d ",contP);
    printf("%d ",contI);
    return 0;
}
```

Practica1_example3.1.asm

```
.text
.globl start

start:
    # Inicialización de registros.
    la $s0, size          # $s1 = size
    lw $s1, 0($s0)
    ori $t0, $0, 0         # $t0 = cont
    ori $s2, $0, 0         # $s2 = sum
    ori $s3, $0, 0         # $s3 = pos
    ori $s4, $0, 0         # $s4 = neg

    # <init>
    ori $s5, $0, 0         # $s5 = i
    la $s6, array          # $s6 = &array

for:
    # if (<cond>)
    bge $s5, $s1, done

    # <for-body>
    lw $s7, 0($s6)
    add $s2, $s2, $s7
    add $t0, $t0, 1
    blez $s7, negativos
    add $s3, $s3, $s7
    j update

negativos:
    bgez $s7, update
```

```
        add $s4, $s4, $s7

update:
    # <update>
    addi $s5, $s5, 1
    addi $s6, $s6, 4          # Mover puntero del array
    j for

done:
    move  $a0, $t0
    li    $v0, 1
    syscall

    li $v0, 4
    la $a0, newline
    syscall

    move  $a0, $s2
    li    $v0, 1
    syscall

    li $v0, 4
    la $a0, newline
    syscall

    move  $a0, $s3
    li    $v0, 1
    syscall

    li $v0, 4
    la $a0, newline
    syscall

    move  $a0, $s4
    li    $v0, 1
    syscall

exit:
    li    $v0, 10
    syscall

# Inicialización de los datos
.data
    size:    .word 15
    array:   .word 7, 3, 1, -25, 99, 54, 7, 9, -25, -123, 98, 56, 0, 34, -39
    newline: .asciiz "\n"
```

Practica1_example3.2.asm

```

.text
.globl start

start:
    # Inicialización de registros.
    la $t0, array          # $t0: array
    li $t1, 0              # $t1: cont = 0

for:
    # if (<cond>)
    lb $a0, 0($t0)         # Carga el elemento actual del array.
    beqz $a0, done         # Comprueba si el recorrido del array ha terminado.

    li $v0, 11             # "Print" del caracter actual (que está cargado en $a0).
    syscall

    li $v0, 4              # "Print" del Salto de línea
    la $a0, newline
    syscall

    # FALTA                # Mover puntero del array
    # FALTA                # $t1: cont += 1

    j     for

done:
    li $v0, 4              # "Print" del texto.
    la $a0, txt
    syscall

    li $v0, 1              # "Print" de "cont".
    add $a0, $0,$t1
    syscall

exit:
    li $v0, 10             # Carga el número de llamada al sistema para salir
    syscall               # Llamada al sistema

# Inicialización de los datos
.data
    array: .asciiz "Hola don Pepito, hola don José"
    newline: .asciiz "\n"
    txt: .asciiz "El número de caracteres de la frase es: "

```