Instituto Politécnico Nacional

Escuela Superior de Cómputo

Arquitectura de computadoras

Practica 1

Profesor:

Maria Elena Aguilar Jauregui

Alumnos:

Lechuga Canales Héctor Jair

García Quiroz Gustavo Ivan

Eduardo Alfonso Rivera Morelos

Quintero Maldonado Fabián

Grupo:

5CV1

Índice

1 Introducción

La arquitectura de computadoras representa un campo fundamental en el desarrollo de la tecnología informática moderna, donde cada instrucción ejecutada por un procesador refleja la compleja interacción entre hardware y software. En este contexto, la arquitectura RISC-V emerge como un modelo de conjunto de instrucciones (ISA) revolucionario, caracterizado por su diseño de código abierto y su flexibilidad para adaptarse a diversas necesidades computacionales.

La comprensión profunda de las instrucciones de bajo nivel es esencial para todo ingeniero en computación, ya que permite entender el funcionamiento interno de los procesadores y cómo se ejecutan las operaciones más básicas. Mediante la simulación de instrucciones como carga, almacenamiento, operaciones aritméticas y lógicas, saltos condicionales e incondicionales, y manejo de subrutinas, se logra una aproximación práctica al núcleo del funcionamiento de un procesador.

La presente práctica busca precisamente desmitificar la complejidad de las instrucciones de ensamblador, proporcionando una experiencia práctica que conecta los conceptos teóricos con su implementación concreta. A través de la programación directa en lenguaje ensamblador RISC-V, los estudiantes pueden explorar y comprender los mecanismos fundamentales que permiten la ejecución de programas en un sistema computacional, profundizando así su conocimiento sobre la arquitectura de computadoras.

2 Objetivos

- Familiarizarse con el uso de un simulador compatible con ISA RISC-V.
- Implementar instrucciones de carga y almacenamiento en un entorno de simulación.
- Ejecutar operaciones aritméticas y lógicas básicas en RISC-V.
- Analizar los resultados obtenidos y evaluar el comportamiento de las instrucciones programadas.

3 Desarrollo

Para la realización de la práctica, se utilizó el simulador VENUS, el cual permite ejecutar programas en RISC-V desde un navegador sin necesidad de instalación.
Pasos para ejecutar los programas en VENUS:

1. Acceder al simulador en VENUS Web.
2. En la ventana Editor, escribir el código del programa a ejecutar.
3. En la pestaña Simulador, presionar el botón de "assemble" para ensamblar el código.
4. Presionar "Run" para ejecutar el programa.
5. Visualizar resultados en la consola y en la ventana de registros y memoria.

3.1 Funcionamiento de los programas

3.1.1 Operaciones aritméticas

Se programaron operaciones de suma, resta, multiplicación y división. Cada operación carga valores en registros, realiza la operación correspondiente y almacena el resultado en un tercer registro.

```
# Operaciones Aritméticas en RISC-V
# Este archivo contiene ejemplos de operaciones aritméticas básicas:
# suma, resta, multiplicación y división

.text
main:
    # Programa 1: Suma de dos números
    li x5, 10       # cargar 10 en x5
    li x6, 20       # cargar 20 en x6
    add x7, x5, x6  # suma x5 + x6 y guarda el resultado en x7

    # Programa 2: Resta de dos números
```

```
    li x8, 30       # cargar 30 en x8
    li x9, 15       # cargar 15 en x9
    sub x10, x8, x9 # resta x8 - x9 y guarda el resultado en x10

    # Programa 3: Multiplicación de dos números
    li x11, 7       # cargar 7 en x11
    li x12, 6       # cargar 6 en x12
    mul x13, x11, x12  # multiplica x11 * x12 y guarda el resultado en x13

    # Programa 4: División de dos números
    li x14, 20      # cargar 20 en x14
    li x15, 4       # cargar 4 en x15
    div x16, x14, x15  # divide x14 / x15 y guarda el resultado en x16

    # Terminar programa
    li a0, 10       # código para terminar el programa
    ecall
```

3.1.2 Operaciones lógicas

Se incluyeron operaciones AND, OR, XOR, desplazamientos y comparaciones. Estas operaciones permiten manipular bits y evaluar condiciones.

```
# Operaciones Lógicas en RISC-V
# Este archivo contiene ejemplos de operaciones lógicas básicas:
# AND, OR, XOR, desplazamiento a la izquierda, desplazamiento a la derecha y comparación

.text
main:
    # Programa 1: Operación AND lógica
```

```
li x5, 12       # cargar 12 en x5 (binario: 1100)
li x6, 10       # cargar 10 en x6 (binario: 1010)
and x7, x5, x6  # AND lógico entre x5 y x6, guarda en x7

# Programa 2: Operación OR lógica
li x8, 12       # cargar 12 en x8 (binario: 1100)
li x9, 10       # cargar 10 en x9 (binario: 1010)
or x10, x8, x9  # OR lógico entre x8 y x9, guarda en x10

# Programa 3: Operación XOR lógica
li x11, 12      # cargar 12 en x11 (binario: 1100)
li x12, 10      # cargar 10 en x12 (binario: 1010)
xor x13, x11, x12  # XOR lógico entre x11 y x12, guarda en x13

# Programa 4: Desplazamiento a la izquierda (shift left)
li x14, 5       # cargar 5 en x14 (binario: 0101)
li x15, 2       # desplazar 2 bits
sll x16, x14, x15  # desplazar x14 a la izquierda x15 bits, guarda en x16

# Programa 5: Desplazamiento a la derecha lógico (shift right logical)
li x17, 20      # cargar 20 en x17
li x18, 2       # desplazar 2 bits
srl x19, x17, x18  # desplazar x17 a la derecha x18 bits, guarda en x19

# Programa 6: Comparación "menor que" (set less than)
li x20, 15      # cargar 15 en x20
li x21, 20      # cargar 20 en x21
slt x22, x20, x21  # si x20 < x21, entonces x22 = 1, sino x22 = 0

# Terminar programa
li a0, 10       # código para terminar el programa
```

```
    ecall
```

3.1.3 Carga y almacenamiento

Se trabajó con instrucciones lw y sw para manipular valores en memoria. Un valor se cargó desde memoria a un registro, se modificó y se almacenó nuevamente en memoria.

```
# Ejemplo: Instrucciones de carga y almacenamiento en RISC-V
.data
    valor: .word 42    # Reserva espacio para un valor entero (42)

.text
main:
    li x5, 25          # Cargar valor inmediato 25 en x5
    la x6, valor       # Cargar la dirección de 'valor' en x6
    lw x7, 0(x6)       # Cargar el contenido de la dirección en x6 a x7 (x7 = 42)
    sw x5, 0(x6)       # Almacenar el contenido de x5 en la dirección en x6
    lw x8, 0(x6)       # Cargar el nuevo valor en x8 (ahora x8 = 25)

    li a0, 10          # Código para terminar el programa
    ecall
```

3.1.4 Instrucciones de corrimiento, salto condicional e incondicional y llamado a rutina.

En el programa principal, se cargan los valores 5 y 10 en los registros a0 y a1, se realiza un desplazamiento lógico a la izquierda de a0 por 2 posiciones (resultando en 20), se llama a la subrutina suma que suma estos valores, luego se usa una instrucción de salto condicional bge para potencialmente saltar a la etiqueta

mayor_igual, y finalmente se llama a la subrutina imprimir que imprimiría un mensaje predefinido seguido del resultado de la suma. El programa concluye con un salto incondicional a la etiqueta terminar, donde se realiza una llamada al sistema para finalizar la ejecución, utilizando el código de syscall 10.

```
.data
mensaje: .string "La suma es: "
.text
.globl main

main:
    li a0, 5           # Cargar valor 5 en a0
    li a1, 10          # Cargar valor 10 en a1

    # Ejemplo de corrimiento (shift)
    slli a2, a0, 2     # Desplazamiento lógico a la izquierda: a2 = a0 << 2 (5*4 = 20)

    jal suma           # Llamado a rutina suma (salto y enlace)

    # Salto condicional
    bge a3, a2, mayor_igual  # Si a0 >= a2, salta a mayor_igual
    mv s1, zero        # Esto no se ejecutará si el salto se toma
mayor_igual:
    mv s0, a0          # Guardar resultado en s0

    jal imprimir       # Llamado a rutina imprimir

    # Ejemplo de salto incondicional
    j terminar         # Salto incondicional a la etiqueta terminar

    # Este código no se ejecutará debido al salto anterior
    li a7, 10
```

```
    ecall

terminar:
    li a7, 10          # Código de salida del sistema
    ecall              # Llamada al sistema para terminar

suma:
    add a3, a0, a1     # Suma a3 = a0 + a1
    jr ra              # Retorno de rutina

imprimir:
    li a7, 4           # Código para imprimir string
    la a0, mensaje     # Cargar dirección del mensaje
    #ecall             # Imprimir mensaje

    li a7, 1           # Código para imprimir entero
    mv a0, s0          # Cargar valor a imprimir
    ecall              # Imprimir número

    li a7, 11          # Código para imprimir carácter
    li a0, 10          # Cargar carácter de nueva línea
    ecall              # Imprimir nueva línea

    jr ra              # Retorno de rutina
```

4 Resultados

4.1 Operaciones Aritméticas

En el desarrollo de las operaciones aritméticas básicas, se implementaron exitosamente cuatro programas que demuestran las capacidades fundamentales de procesamiento numérico en RISC-V. El primer programa realizó una suma de los valores 10 y 20, almacenando el resultado 30 en el registro x7. Seguidamente, se ejecutó una resta entre 30 y 15, obteniendo el valor 15 en el registro x10. La multiplicación se llevó a cabo con los números 7 y 6, produciendo el resultado 42 en el registro x13. Finalmente, la división de 20 entre 4 generó el cociente 5, almacenado en el registro x16, evidenciando la capacidad del procesador para realizar operaciones matemáticas elementales de manera eficiente.

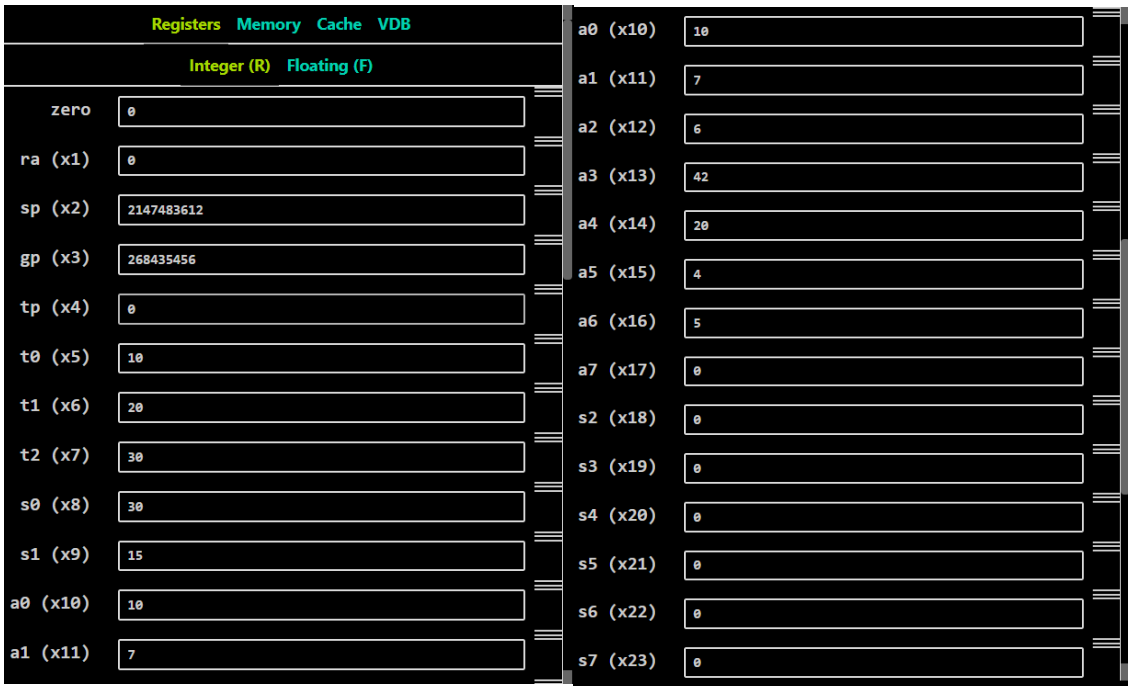

Figura 1 Operaciones Aritméticas

4.2 Operaciones Lógicas

Las operaciones lógicas implementadas demostraron la capacidad de manipulación de bits a nivel de registro. En la operación AND entre los valores binarios 1100 (12) y 1010 (10), se obtuvo el resultado 1000 (8), ilustrando cómo esta operación preserva los bits activos en ambos operandos. La operación OR entre los mismos valores produjo 1110 (14), mostrando la unión de bits activos. La operación XOR generó el valor 0110 (6), resaltando su propiedad de revelar diferencias bit a bit. Adicionalmente, se realizaron operaciones de desplazamiento: un corrimiento a la izquierda de 5 por 2 posiciones resultó en 20, mientras que un desplazamiento a la derecha de 20 por 2 posiciones produjo 5. La instrucción de comparación "menor que" se probó entre 15 y 20, generando un resultado de 1, confirmando la capacidad de comparación lógica.

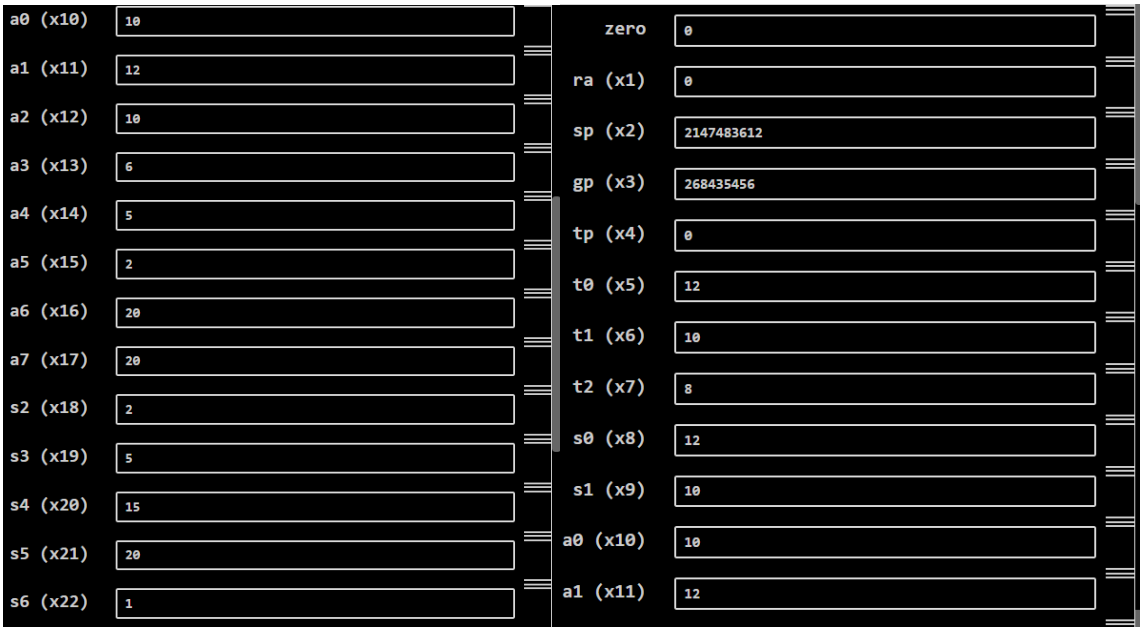

Figura 2 Operaciones Lógicas

4.3 Carga y Almacenamiento

La sección de carga y almacenamiento demostró la interacción entre registros y memoria. Inicialmente, se cargó un valor predefinido de 42 en memoria. Posteriormente, se cargó el valor 25 en el registro x5 y se utilizaron las instrucciones de carga (lw) y almacenamiento (sw) para manipular este valor. La dirección de memoria del valor se cargó en x6, permitiendo la transferencia del valor 25 desde el

registro a la ubicación de memoria. Al volver a cargar el valor, se confirmó el cambio, mostrando la capacidad de modificar y recuperar datos en la memoria del sistema.

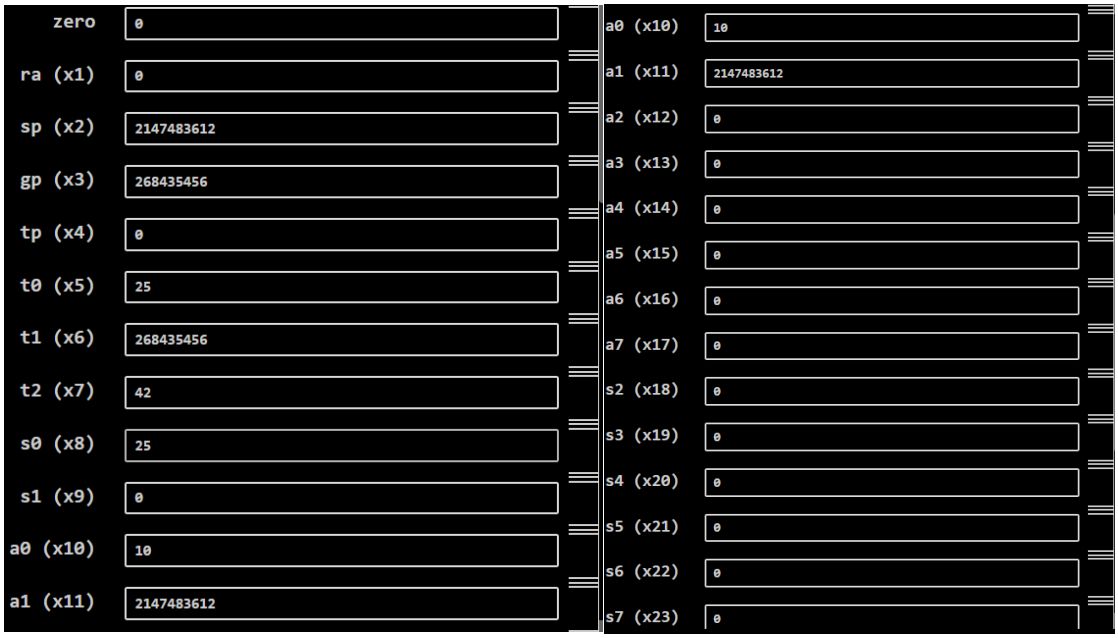

Figura 3 Carga y Almacenamiento

4.4 Instrucciones de Corrimiento, Salto y Subrutinas

En esta sección más compleja, se implementaron múltiples técnicas de control de flujo y manipulación de datos. Se realizó un desplazamiento lógico a la izquierda del valor 5 por 2 posiciones, resultando en 20. Una subrutina de suma se implementó para adicionar los valores 5 y 10, generando el resultado 15. Se utilizaron instrucciones de salto condicional (bge) para potencialmente modificar el flujo del programa basado en comparaciones. La subrutina de impresión demostró la capacidad de imprimir mensajes de texto y valores numéricos, incluyendo un salto de línea. El programa concluyó con un salto incondicional a la etiqueta de terminación, ilustrando el control preciso del flujo de ejecución en lenguaje ensamblador.

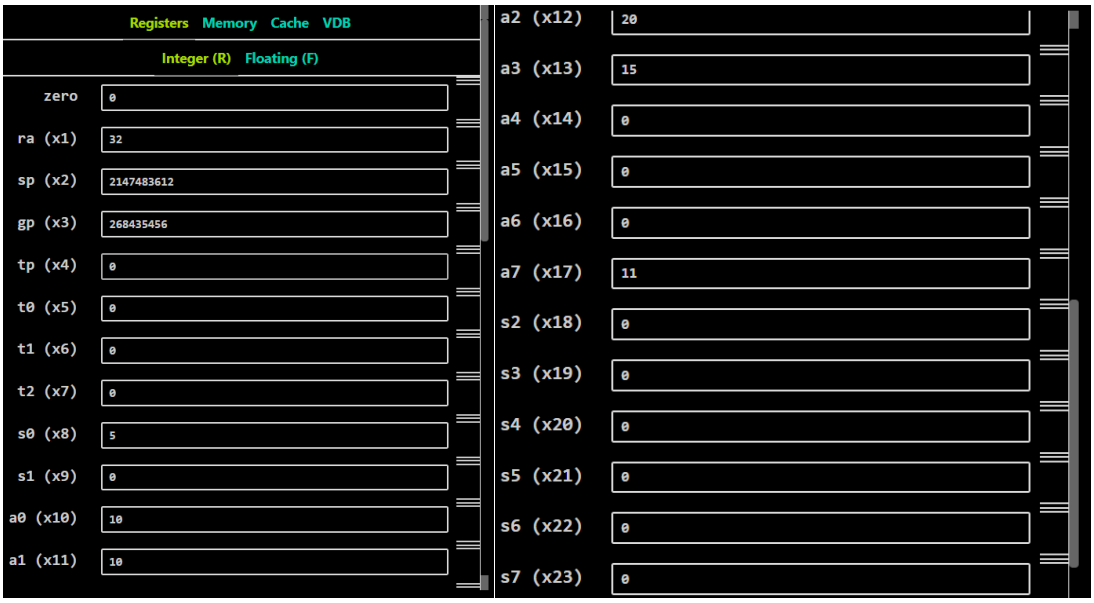

Figura 4 Instrucciones de Corrimiento, Salto y Subrutinas

5 Conclusiones

El uso del simulador VENUS permitió observar el funcionamiento detallado de las instrucciones de carga, almacenamiento, aritméticas y lógicas en la arquitectura RISC-V. Se logró una comprensión más profunda del procesamiento de datos en esta arquitectura y su aplicación en sistemas embebidos y computación de bajo consumo. Esta práctica sienta las bases para el desarrollo de programas más complejos en RISC-V y su implementación en hardware real.

6 Anexo

Código del programa

```
# Operaciones Aritméticas en RISC-V
# Este archivo contiene ejemplos de operaciones aritméticas básicas:
# suma, resta, multiplicación y división

.text
main:
    # Programa 1: Suma de dos números
    li x5, 10       # cargar 10 en x5
    li x6, 20       # cargar 20 en x6
    add x7, x5, x6  # suma x5 + x6 y guarda el resultado en x7

    # Programa 2: Resta de dos números
    li x8, 30       # cargar 30 en x8
    li x9, 15       # cargar 15 en x9
    sub x10, x8, x9 # resta x8 - x9 y guarda el resultado en x10

    # Programa 3: Multiplicación de dos números
    li x11, 7       # cargar 7 en x11
    li x12, 6       # cargar 6 en x12
    mul x13, x11, x12  # multiplica x11 * x12 y guarda el resultado en x13

    # Programa 4: División de dos números
    li x14, 20      # cargar 20 en x14
    li x15, 4       # cargar 4 en x15
    div x16, x14, x15  # divide x14 / x15 y guarda el resultado en x16

    # Terminar programa
    li a0, 10       # código para terminar el programa
```

```
    ecall
# Operaciones Lógicas en RISC-V
# Este archivo contiene ejemplos de operaciones lógicas básicas:
# AND, OR, XOR, desplazamiento a la izquierda, desplazamiento a la derecha y comparación

.text
main:
    # Programa 1: Operación AND lógica
    li x5, 12       # cargar 12 en x5 (binario: 1100)
    li x6, 10       # cargar 10 en x6 (binario: 1010)
    and x7, x5, x6  # AND lógico entre x5 y x6, guarda en x7

    # Programa 2: Operación OR lógica
    li x8, 12       # cargar 12 en x8 (binario: 1100)
    li x9, 10       # cargar 10 en x9 (binario: 1010)
    or x10, x8, x9  # OR lógico entre x8 y x9, guarda en x10

    # Programa 3: Operación XOR lógica
    li x11, 12      # cargar 12 en x11 (binario: 1100)
    li x12, 10      # cargar 10 en x12 (binario: 1010)
    xor x13, x11, x12  # XOR lógico entre x11 y x12, guarda en x13

    # Programa 4: Desplazamiento a la izquierda (shift left)
    li x14, 5       # cargar 5 en x14 (binario: 0101)
    li x15, 2       # desplazar 2 bits
    sll x16, x14, x15  # desplazar x14 a la izquierda x15 bits, guarda en x16

    # Programa 5: Desplazamiento a la derecha lógico (shift right logical)
    li x17, 20      # cargar 20 en x17
    li x18, 2       # desplazar 2 bits
```

```
    srl x19, x17, x18  # desplazar x17 a la derecha x18 bits, guarda en x19

    # Programa 6: Comparación "menor que" (set less than)
    li x20, 15      # cargar 15 en x20
    li x21, 20      # cargar 20 en x21
    slt x22, x20, x21  # si x20 < x21, entonces x22 = 1, sino x22 = 0

    # Terminar programa
    li a0, 10       # código para terminar el programa
    ecall
# Ejemplo: Instrucciones de carga y almacenamiento en RISC-V
.data
    valor: .word 42    # Reserva espacio para un valor entero (42)

.text
main:
    li x5, 25          # Cargar valor inmediato 25 en x5
    la x6, valor       # Cargar la dirección de 'valor' en x6
    lw x7, 0(x6)       # Cargar el contenido de la dirección en x6 a x7 (x7 = 42)
    sw x5, 0(x6)       # Almacenar el contenido de x5 en la dirección en x6
    lw x8, 0(x6)       # Cargar el nuevo valor en x8 (ahora x8 = 25)

    li a0, 10          # Código para terminar el programa
    ecall
.data
mensaje: .string "La suma es: "
.text
.globl main

main:
    li a0, 5           # Cargar valor 5 en a0
```

```
    li a1, 10          # Cargar valor 10 en a1

    # Ejemplo de corrimiento (shift)
    slli a2, a0, 2     # Desplazamiento lógico a la izquierda: a2 = a0 << 2 (5*4 = 20)

    jal suma           # Llamado a rutina suma (salto y enlace)

    # Salto condicional
    bge a3, a2, mayor_igual  # Si a0 >= a2, salta a mayor_igual
    mv s1, zero        # Esto no se ejecutará si el salto se toma
mayor_igual:
    mv s0, a0          # Guardar resultado en s0

    jal imprimir       # Llamado a rutina imprimir

    # Ejemplo de salto incondicional
    j terminar         # Salto incondicional a la etiqueta terminar

    # Este código no se ejecutará debido al salto anterior
    li a7, 10
    ecall

terminar:
    li a7, 10          # Código de salida del sistema
    ecall              # Llamada al sistema para terminar

suma:
    add a3, a0, a1     # Suma a3 = a0 + a1
    jr ra              # Retorno de rutina

imprimir:
```

```
li a7, 4          # Código para imprimir string
la a0, mensaje    # Cargar dirección del mensaje
#ecall            # Imprimir mensaje

li a7, 1          # Código para imprimir entero
mv a0, s0         # Cargar valor a imprimir
ecall             # Imprimir número

li a7, 11         # Código para imprimir carácter
li a0, 10         # Cargar carácter de nueva línea
ecall             # Imprimir nueva línea

jr ra             # Retorno de rutina
```