16. Dado el siguiente programa en C, usar el conversor https://godbolt.org/ para pasarlo a RISC-V. Comparar ambos códigos.

- **¿Cuál es la principal diferencia que encuentra?**
- **¿Cómo es la equivalencia entre las instrucciones?**
- **¿Cómo es el orden de las instrucciones?**

```
#include <stdio.h>


int main() {
   char l, palabra[21];
   int i;


   printf("Teclee una palabra de menos de 20 letras:");
   scanf("%s", palabra);


   i = 0;
   while (palabra[i++] != '\0')
      ;
   l = i - 1;


   printf("%s tiene %d letras\n", palabra, l);
   printf("%s escrita al revés es: ", palabra);


   i = l;
   while (i > 0)
      printf("%c", palabra[--i]);


   return 0;
}
```

```
main:
   # Prologue
   addi   sp,sp,-32
   sw     ra,28(sp)
   sw     s0,24(sp)
   addi   s0,sp,32
```

```
    # Llamada a printf para solicitar la palabra
    lui     a0,%hi(.LC0)
    addi    a0,a0,%lo(.LC0)
    call    printf

    # Llamada a scanf para leer la palabra
    lui     a0,%hi(.LC1)
    addi    a0,a0,%lo(.LC1)
    addi    a1,s0,-21
    call    scanf

    # Inicialización de i
    sb      zero,-22(s0)

    # Bucle para calcular la longitud de la palabra
.L2:
    lb      a4,-21(s0)
    addi    a5,a4,-1
    sb      a5,-22(s0)
    addi    a5,a4,1
    addi    a5,a5,-1
    lbu     a5,0(a5)
    bnez    a5,.L2

    # Almacenar la longitud en l
    lb      a5,-22(s0)
    sb      a5,-23(s0)

    # Llamada a printf para mostrar la longitud
    lui     a0,%hi(.LC2)
    addi    a0,a0,%lo(.LC2)
    addi    a1,s0,-21
    lb      a2,-23(s0)
    call    printf

    # Llamada a printf para indicar la inversión
```

```
    lui     a0,%hi(.LC3)
    addi    a0,a0,%lo(.LC3)
    addi    a1,s0,-21
    call    printf

    # Inicialización de i para invertir la palabra
    lb      a5,-23(s0)
    sb      a5,-22(s0)

    # Bucle para imprimir la palabra invertida
.L3:
    lb      a5,-22(s0)
    blez    a5,.L4
    addi    a5,a5,-1
    sb      a5,-22(s0)
    addi    a4,s0,-21
    add     a4,a4,a5
    lbu     a0,0(a4)
    call    putchar
    j       .L3

.L4:
    # Epilogue
    lw      ra,28(sp)
    lw      s0,24(sp)
    addi    sp,sp,32
    ret
```

Comparación y análisis

1. **Principal diferencia encontrada:**
    - **Abstracción vs. Especificidad:** El código en C es más abstracto y legible, enfocándose en la lógica del programa sin preocuparse por los detalles de implementación a nivel de hardware. En contraste, el código ensamblador es específico y detallado, manejando registros, direcciones de memoria y llamadas a funciones de manera explícita.
2. **Equivalencia entre las instrucciones:**
    - **Declaraciones y asignaciones:** En C, las variables se declaran y asignan directamente. En ensamblador, esto se traduce en operaciones que ajustan el

puntero de pila (sp) y almacenan valores en registros o en la memoria relativa a sp.

- **Llamadas a funciones (printf, scanf):** En C, se llaman directamente. En ensamblador, se carga la dirección de la cadena de formato en a0, los argumentos en a1, a2, etc., y luego se utiliza la instrucción call para invocar la función.
- **Bucles (while):** En C, se expresan de forma estructurada. En ensamblador, se implementan mediante etiquetas (.L2, .L3) y saltos condicionales (bnez, blez) para controlar el flujo del programa.
- **Operaciones con cadenas:** En C, se manejan con índices y operadores. En ensamblador, se utilizan desplazamientos y operaciones de carga/almacenamiento de bytes (lb, lbu, sb) para acceder y modificar caracteres en la memoria.

3. **Orden de las instrucciones:**
   - **Secuencia lógica:** Aunque el ensamblador sigue la lógica del código en C, las instrucciones pueden reordenarse para optimizar el uso de registros y minimizar accesos a memoria, especialmente con optimizaciones del compilador.
   - **Prologue y epilogue:** El ensamblador incluye secciones al inicio y al final (prologue y epilogue) para preparar y limpiar el entorno de ejecución, manejando el puntero de pila y guardando/restaurando registros.
   - **Manejo explícito de la pila:** Las variables locales y temporales se gestionan mediante ajustes al puntero de pila y accesos a direcciones relativas, algo implícito en C pero explícito en ensamblador.

Conclusión

La traducción de un programa en C a ensamblador RISC-V revela cómo las construcciones de alto nivel se descomponen en operaciones básicas que la CPU puede ejecutar. Esta comparación destaca la eficiencia y control que ofrece el ensamblador, a costa de una mayor complejidad y menor legibilidad en comparación con el código en C.