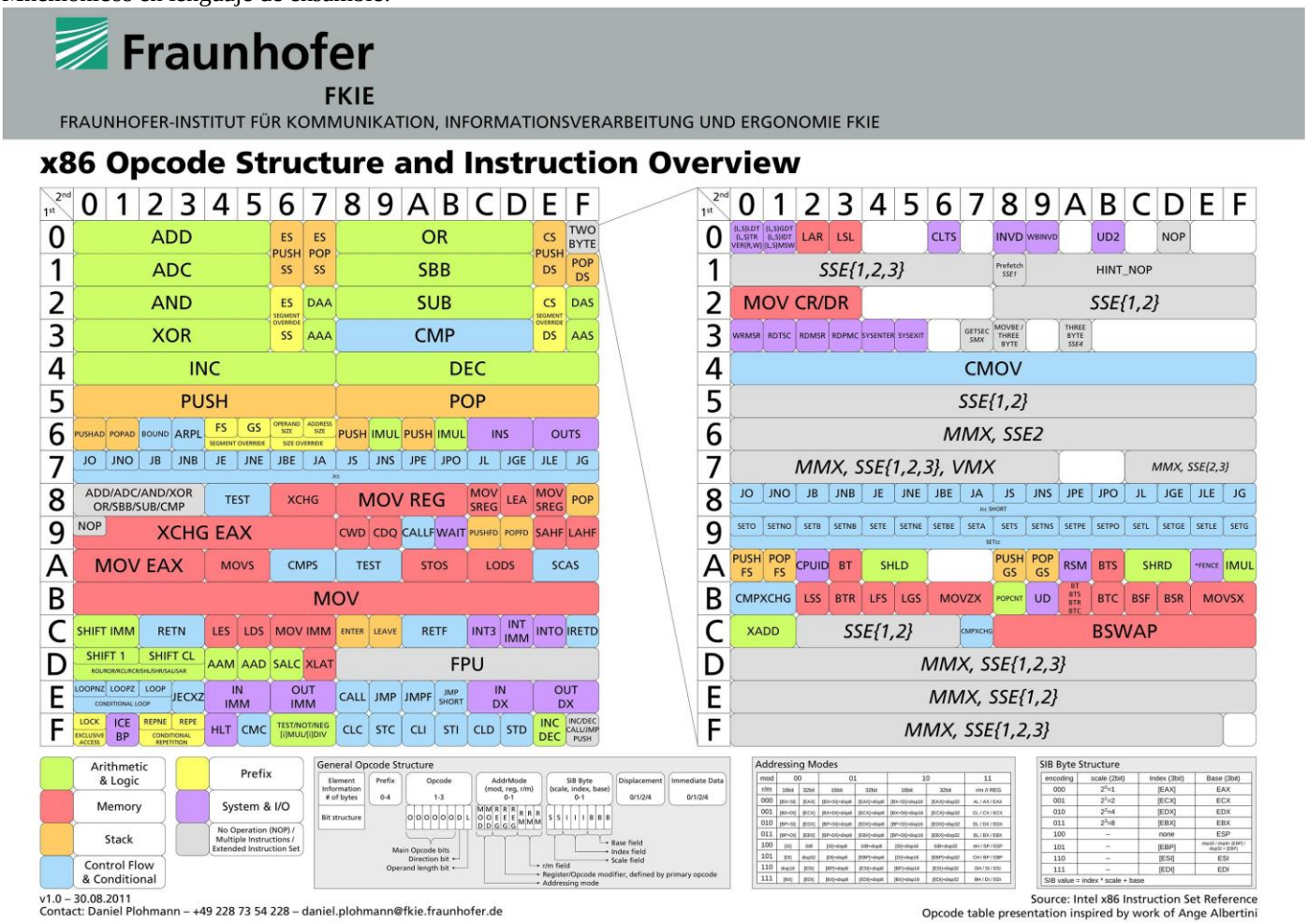
Solución Parcial Dispro

Ingenieria Inversa

Guillermo Andrés Aguilera Hurtado
Miguel Angel Cera Contreras

Pontificia Universidad Javeriana
Facultad de Ingeniería
Bogotá
Noviembre 2023

Mnemónicos en lenguaje de ensamble:



Desambladores:

- https://disasm.pro
- https://defuse.ca/online-x86-assembler.htm#disassembly2
- http://shell-storm.org/online/Online-Assembler-and-Disassembler/

Este estudio conlleva a conocer: (1) qué hace cada instrucción, (2) sobre qué registros de la CPU actúa cada instrucción y (3) cuál es el posible efecto en las banderas del "Status Register". Esto dirige la atención a revisar algunos aspectos básicos de la arquitectura x86 y a identificar si la instrucción corresponde a una de (1) control de flujo, (2) operación aritmético-lógica, (3) de transferencia de datos entre CPU y memoria y (4) entrada y salida.

- https://www.felixcloutier.com/x86/
- https://shell-storm.org/x86doc/
- https://en.wikipedia.org/wiki/X86_instruction_listings
- https://docs.oracle.com/cd/E19253-01/817-5477/817-5477.pdf

Se recomienda empezar a identificar los saltos (condicionales y no condicionales). Esta sugerencia lleva a inferir de primero, las secuencias de control de flujo que pudieron haber sido usadas en el código fuente original de Lenguaje C. Si todos los saltos son hacia adelante, el control de flujo en alto nivel se llevó a cabo mediante secuencias de Luego, se sugiere hacer una propuesta de la función y del "main()" en Lenguaje C. Para ello puede editar dicha

propuesta de código fuente, para verificar en el sitio WEB de “Compiler Explorer” con las opciones apropiadas, si el código inferido genera el código máquina objeto de este taller:

- https://godbolt.org/

Codigo 1:

```
code05.o:    file format Mach-O 64-bit x86-64

Disassembly of section __TEXT,__text:

0000000000000000 _main:
       0: 55                            pushq  %rbp
       1: 48 89 e5                      movq   %rsp, %rbp
       4: 48 83 ec 10                   subq   $16, %rsp
       8: c7 45 fc 00 00 00 00          movl   $0, -4(%rbp)
       f: bf 04 00 00 00                movl   $4, %edi
      14: be 0c 00 00 00                movl   $12, %esi
      19: e8 00 00 00 00                callq  0 <_func0>
      1e: 31 c9                         xorl   %ecx, %ecx
      20: 89 45 f8                      movl   %eax, -8(%rbp)
      23: 89 c8                         movl   %ecx, %eax
      25: 48 83 c4 10                   addq   $16, %rsp
      29: 5d                            popq   %rbp
      2a: c3                            retq
      2b: 0f 1f 44 00 00                nopl   (%rax,%rax)

0000000000000030 _func0:
      30: 55                            pushq  %rbp
      31: 48 89 e5                      movq   %rsp, %rbp
      34: 89 7d fc                      movl   %edi, -4(%rbp)
      37: 89 75 f8                      movl   %esi, -8(%rbp)
      3a: c7 45 f4 00 00 00 00          movl   $0, -12(%rbp)
      41: 8b 45 fc                      movl   -4(%rbp), %eax
      44: 89 45 f0                      movl   %eax, -16(%rbp)
      47: c7 45 ec 01 00 00 00          movl   $1, -20(%rbp)
      4e: 8b 45 fc                      movl   -4(%rbp), %eax
      51: 3b 45 f8                      cmpl   -8(%rbp), %eax
      54: 0f 8e 06 00 00 00             jle    6 <_func0+0x30>
      5a: 8b 45 f8                      movl   -8(%rbp), %eax
      5d: 89 45 f0                      movl   %eax, -16(%rbp)
      60: c7 45 f4 01 00 00 00          movl   $1, -12(%rbp)
      67: 8b 45 f4                      movl   -12(%rbp), %eax
      6a: 3b 45 f0                      cmpl   -16(%rbp), %eax
      6d: 0f 8f 39 00 00 00             jg     57 <_func0+0x7c>
      73: 8b 45 fc                      movl   -4(%rbp), %eax
      76: 99                            cltd
      77: f7 7d f4                      idivl  -12(%rbp)
      7a: 83 fa 00                      cmpl   $0, %edx
      7d: 0f 85 16 00 00 00             jne    22 <_func0+0x69>
      83: 8b 45 f8                      movl   -8(%rbp), %eax
      86: 99                            cltd
      87: f7 7d f4                      idivl  -12(%rbp)
      8a: 83 fa 00                      cmpl   $0, %edx
      8d: 0f 85 06 00 00 00             jne    6 <_func0+0x69>
      93: 8b 45 f4                      movl   -12(%rbp), %eax
      96: 89 45 ec                      movl   %eax, -20(%rbp)
      99: e9 00 00 00 00                jmp    0 <_func0+0x6e>
      9e: 8b 45 f4                      movl   -12(%rbp), %eax
      a1: 83 c0 01                      addl   $1, %eax
      a4: 89 45 f4                      movl   %eax, -12(%rbp)
      a7: e9 bb ff ff ff                jmp    -69 <_func0+0x37>
      ac: 8b 45 ec                      movl   -20(%rbp), %eax
      af: 5d                            popq   %rbp
      b0: c3                            retq
```

Fig 1. Código 1/Code 05.

Análisis:

Código MAIN:

1. pushq %rbp: **Guarda el valor actual de `rbp` (base pointer) en la pila.**
2. movq %rsp, %rbp: **Establece el valor actual del puntero de pila (`rsp`) como la nueva base (`rbp`).**
3. subq $16, %rsp: **Reserva espacio en la pila para variables locales, restando 16 bytes del puntero de pila.**
4. movl $0, -4(%rbp): **Inicializa una variable local moviendo el valor `0` a la posición `-4(%rbp)`.**
5. movl $4, %edi: **Coloca el valor `4` en el registro `%edi`.**
6. movl $12, %esi: **Coloca el valor `12` en el registro `%esi`.**
7. callq 0 <_func0>: **Llama a la función `_func0`.**
8. xorl %ecx, %ecx: **Realiza una operación XOR, estableciendo `%ecx` en `0`.**
9. movl %eax, -8(%rbp): **Guarda el resultado de la función `_func0` en la variable local.**
10. movl %ecx, %eax: **Mueve el valor de `%ecx` a `%eax`.**
11. addq $16, %rsp: **Libera el espacio reservado en la pila.**
12. popq %rbp: **Restaura el valor de la base de la pila.**
13. retq: **Retorna de la función.**

La línea `nopl (%rax, %rax)` es una instrucción que no realiza ninguna operación y puede ser ignorada en este contexto.

Código_func0:

1. pushq %rbp: Guarda el valor actual de rbp en la pila.
2. **movq %rsp, %rbp**: Establece el valor actual del puntero de pila (rsp) como la nueva base (rbp).
3. **movl %edi, -4(%rbp)**: Guarda el primer argumento de la función en la posición -4(%rbp).
4. **movl %esi, -8(%rbp)**: Guarda el segundo argumento de la función en la posición -8(%rbp).
5. **movl $0, -12(%rbp)**: Inicializa una variable local en la posición -12(%rbp) con el valor 0.
6. **movl -4(%rbp), %eax**: Mueve el valor del primer argumento a %eax.
7. **movl %eax, -16(%rbp)**: Guarda el valor del primer argumento en la posición -16(%rbp).
8. **movl $1, -20(%rbp)**: Inicializa una variable local en la posición -20(%rbp) con el valor 1.
9. **cmpl -8(%rbp), %eax**: Compara el segundo argumento con el valor en %eax.
10. **jle 6 <_func0+0x30>**: Salta a la dirección 6 si la comparación es menor o igual (<=).
11. **movl -8(%rbp), %eax**: Mueve el valor del segundo argumento a %eax.
12. **movl %eax, -16(%rbp)**: Guarda el valor del segundo argumento en la posición -16(%rbp).
13. **movl $1, -12(%rbp)**: Establece la variable local en -12(%rbp) en 1.
14. **cmpl -16(%rbp), %eax**: Compara el valor almacenado en -16(%rbp) con %eax.
15. **jg 57 <_func0+0x7c>**: Salta a la dirección 57 si la comparación es mayor (>).
16. **movl -4(%rbp), %eax**: Mueve el valor de la variable local almacenada en -4(%rbp) a %eax.
17. **cltd**: Convierte el contenido de %eax a un valor de doble palabra extendido en %edx:%eax.
18. **idivl -12(%rbp)**: Divide %edx:%eax por el valor en -12(%rbp). El cociente se guarda en %eax y el residuo en %edx.
19. **cmpl $0, %edx**: Compara el residuo con cero.
20. **jne 22 <_func0+0x69>**: Salta a la dirección 22 si el resultado de la comparación no es igual a cero

El patrón se repite para las siguientes instrucciones (26-30, 36-40, 47-4a), cada una realiza una operación de división y salto condicional basado en el residuo. Estas instrucciones están relacionadas con un bucle que continúa hasta que el residuo es cero.

21. **movl -12(%rbp), %eax**: Mueve el valor de la variable local almacenada en -12(%rbp) a %eax.
22. addl $1, %eax: Aumenta el valor de %eax en 1.
23. **movl %eax, -12(%rbp)**: Guarda el nuevo valor de %eax en la variable local.
24. **jmp -69 <_func0+0x37>**: Salta a la dirección -69 (retroceso) para repetir el bucle.

25. **movl -20(%rbp), %eax**: Mueve el valor de la variable local almacenada en -20(%rbp) a %eax.

26. **popq %rbp**: Restaura el valor de la base de la pila.

27. **retq**: Retorna de la función.

Codigo en C:

```
int func0(int a, int b);
int main() {
   int result = func0(4, 12);
   return 0;
}

int func0(int a, int b) {
   int temp1 = 0;
   int temp2 = a;
   int temp3 = 1;

   if (a > b) {
      temp2 = b;
   }

   for (temp1 = 1; temp1 <= temp2;) {
      if (a % temp1 == 0) {
         if (b % temp1 == 0) {
            temp3 = temp1;
         }
      }
      temp1++;
   }
   return temp3;
}
```

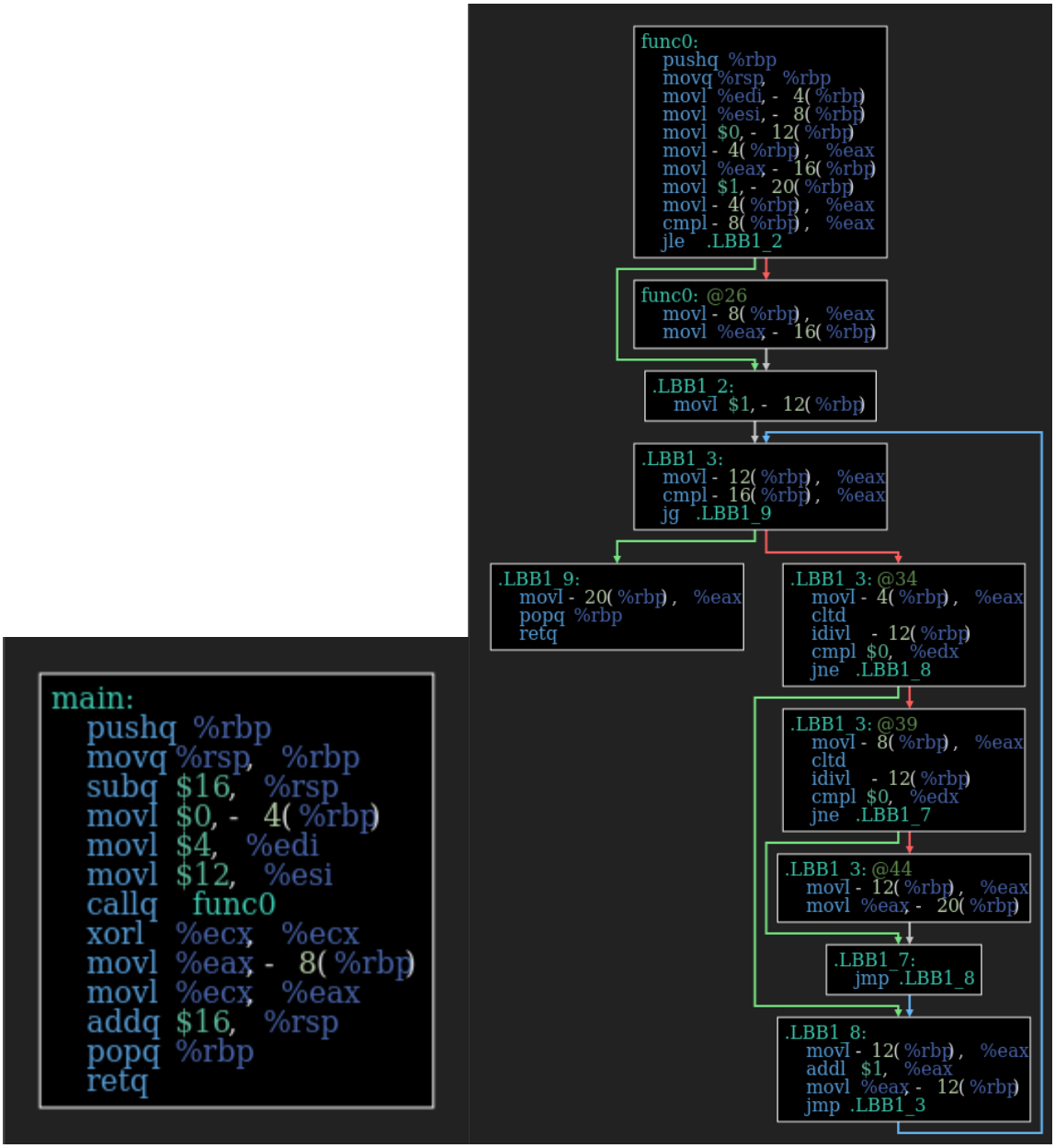
Explicacion: función llamada func0 que calcula el máximo común divisor (MCD) de dos números enteros (a y b) utilizando un enfoque iterativo con un bucle for y condicionales.

La función func0 toma dos parámetros enteros (a y b) y devuelve el MCD de ambos. En el programa principal (main), se llama a func0 con los valores 4 y 12, y el resultado se almacena en la variable result.

La implementación utiliza tres variables temporales (temp1, temp2, y temp3) para gestionar los cálculos. La lógica del código incluye:

14. Inicialización de las variables temporales (temp1, temp2, y temp3).
15. Comparación de a y b para asegurarse de que temp2 almacene el menor de ambos.
16. Uso de un bucle for para iterar desde 1 hasta temp2.
17. Dentro del bucle, se verifica si temp1 es un divisor común de a y b. Si es así, se actualiza temp3 con el valor de temp1.
18. Incremento de temp1 al final de cada iteración.
19. Finalmente, la función retorna el valor de temp3, que representa el máximo común divisor de a y b.

En el ejemplo proporcionado, al llamar a func0 con los valores 4 y 12, se calculará el MCD de ambos, que es 4, y el resultado se almacenará en la variable result.


main:
pushq %rbp
movq %rsp, %rbp
subq $16, %rsp
movl $0, -4(%rbp)
movl $4, %edi
movl $12, %esi
callq func0
xorl %ecx, %ecx
movl %eax, -8(%rbp)
movl %ecx, %eax
addq $16, %rsp
popq %rbp
retq
func0:
pushq %rbp
movq %rsp, %rbp
movl %edi, -4(%rbp)
movl %esi, -8(%rbp)
movl $0, -12(%rbp)
movl -4(%rbp), %eax
movl %eax, -16(%rbp)
movl $1, -20(%rbp)
movl -4(%rbp), %eax
cmpl -8(%rbp), %eax
jle .LBB1_2
func0: @26
movl -8(%rbp), %eax
movl %eax, -16(%rbp)
.LBB1_2:
movl $1, -12(%rbp)
.LBB1_3:
movl -12(%rbp), %eax
cmpl -16(%rbp), %eax
jg .LBB1_9
.LBB1_9:
movl -20(%rbp), %eax
popq %rbp
retq
.LBB1_3: @34
movl -4(%rbp), %eax
cltd
idivl -12(%rbp)
cmpl $0, %edx
jne .LBB1_8
.LBB1_3: @39
movl -8(%rbp), %eax
cltd
idivl -12(%rbp)
cmpl $0, %edx
jne .LBB1_7
.LBB1_3: @44
movl -12(%rbp), %eax
movl %eax, -20(%rbp)
.LBB1_7:
jmp .LBB1_8
.LBB1_8:
movl -12(%rbp), %eax
addl $1, %eax
movl %eax, -12(%rbp)
jmp .LBB1_3

Código 2:

```
code06.o:    file format Mach-O 64-bit x86-64


Disassembly of section __TEXT,__text:

0000000000000000 _main:
       0: 55                            pushq  %rbp
       1: 48 89 e5                      movq   %rsp, %rbp
       4: 48 83 ec 10                   subq   $16, %rsp
       8: c7 45 fc 00 00 00 00          movl   $0, -4(%rbp)
       f: bf 04 00 00 00                movl   $4, %edi
      14: be 0c 00 00 00                movl   $12, %esi
      19: e8 00 00 00 00                callq  0 <_func0>
      1e: 31 c9                         xorl   %ecx, %ecx
      20: 89 45 f8                      movl   %eax, -8(%rbp)
      23: 89 c8                         movl   %ecx, %eax
      25: 48 83 c4 10                   addq   $16, %rsp
      29: 5d                            popq   %rbp
      2a: c3                            retq
      2b: 0f 1f 44 00 00                nopl   (%rax,%rax)

0000000000000030 _func0:
      30: 55                            pushq  %rbp
      31: 48 89 e5                      movq   %rsp, %rbp
      34: 48 83 ec 10                   subq   $16, %rsp
      38: 89 7d f8                      movl   %edi, -8(%rbp)
      3b: 89 75 f4                      movl   %esi, -12(%rbp)
      3e: 83 7d f8 00                   cmpl   $0, -8(%rbp)
      42: 0f 85 0b 00 00 00             jne    11 <_func0+0x23>
      48: 8b 45 f4                      movl   -12(%rbp), %eax
      4b: 89 45 fc                      movl   %eax, -4(%rbp)
      4e: e9 14 00 00 00                jmp    20 <_func0+0x37>
      53: 8b 45 f4                      movl   -12(%rbp), %eax
      56: 99                            cltd
      57: f7 7d f8                      idivl  -8(%rbp)
      5a: 8b 75 f8                      movl   -8(%rbp), %esi
      5d: 89 d7                         movl   %edx, %edi
      5f: e8 cc ff ff ff                callq  -52 <_func0>
      64: 89 45 fc                      movl   %eax, -4(%rbp)
      67: 8b 45 fc                      movl   -4(%rbp), %eax
      6a: 48 83 c4 10                   addq   $16, %rsp
      6e: 5d                            popq   %rbp
      6f: c3                            retq
```

Fig 2. Código 2/Code 06.

Análisis:

Código MAIN:

1. **pushq %rbp**: Guarda el valor actual del puntero de la base de la pila (**%rbp**) en la pila.
2. **movq %rsp, %rbp**: Establece el puntero de la base de la pila (**%rbp**) al valor actual del puntero de la pila (**%rsp**).
3. **subq $16, %rsp**: Reserva espacio en la pila al restar 16 bytes del puntero de la pila (**%rsp**).
4. **movl $0, -4(%rbp)**: Mueve el valor 0 al espacio de 4 bytes (entero) debajo del puntero de la base de la pila (**%rbp**).
5. **movl $4, %edi**: Mueve el valor 4 al registro **%edi**.
6. **movl $12, %esi**: Mueve el valor 12 al registro **%esi**.
7. **callq 0 <_func0>**: Realiza una llamada a la función **_func0**. El valor de retorno se coloca en el registro **%eax**.
8. **xorl %ecx, %ecx**:operación XOR entre el registro **%ecx** y sí mismo, estableciendo **%ecx** en 0.
9. **movl %eax, -8(%rbp)**: Mueve el valor en **%eax** al espacio de 8 bytes debajo del puntero de la base de la pila (**%rbp**).
10. **movl %ecx, %eax**: Mueve el valor en **%ecx** al registro **%eax**.
11. **addq $16, %rsp**: Libera el espacio reservado en la pila al sumar 16 bytes al puntero de la pila (**%rsp**).
12. **popq %rbp**: Restaura el valor del puntero de la base de la pila (**%rbp**) desde la pila.
13. **retq**: Retorna de la función **_main**.

Código_func0:

1. **pushq %rbp**: Guarda el valor actual del puntero de la base de la pila (**%rbp**) en la pila.
2. **movq %rsp, %rbp**: Establece el puntero de la base de la pila (**%rbp**) al valor actual del puntero de la pila (**%rsp**).
3. **subq $16, %rsp**: Reserva espacio en la pila al restar 16 bytes del puntero de la pila (**%rsp**).
4. **movl %edi, -8(%rbp)**: Mueve el valor del primer argumento (**%edi**) al espacio de 8 bytes debajo del puntero de la base de la pila (**%rbp**).
5. **movl %esi, -12(%rbp)**: Mueve el valor del segundo argumento (**%esi**) al espacio de 12 bytes debajo del puntero de la base de la pila (**%rbp**).
6. **cmpl $0, -8(%rbp)**: Compara el valor en el espacio de 8 bytes debajo de **%rbp** con 0.
7. **jne 11 <_func0+0x23>**: Salta a la etiqueta 11 si la comparación no es igual.
8. **movl -12(%rbp), %eax**: Mueve el valor en el espacio de 12 bytes debajo de **%rbp** al registro **%eax**.
9. **movl %eax, -4(%rbp)**: Mueve el valor en **%eax** al espacio de 4 bytes debajo del puntero de la base de la pila (**%rbp**).
10. **jmp 20 <_func0+0x37>**: Salta a la etiqueta 20.
11. Etiqueta 11: Aquí comienza el bloque de código si la comparación en la línea 6 fue verdadera.
12. **movl -12(%rbp), %eax**: Mueve el valor en el espacio de 12 bytes debajo de **%rbp** al registro **%eax**.
13. **cltd**: Estende el signo del registro **%eax** al registro de doble tamaño **%edx:%eax**.
14. **idivl -8(%rbp)**: Divide el valor en **%edx:%eax** por el valor en el espacio de 8 bytes debajo de **%rbp**. El cociente se coloca en **%eax** y el residuo en **%edx**.
15. **movl -8(%rbp), %esi**: Mueve el valor en el espacio de 8 bytes debajo de **%rbp** al registro **%esi**.
16. **movl %edx, %edi**: Mueve el valor en **%edx** al registro **%edi**.
17. **callq -52 <_func0>**: Llama a una función desconocida (offset -52 desde la siguiente instrucción).
18. **movl %eax, -4(%rbp)**: Mueve el valor de retorno de la función al espacio de 4 bytes debajo del puntero de la base de la pila (**%rbp**).
19. **addq $16, %rsp**: Libera el espacio reservado en la pila al sumar 16 bytes al puntero de la pila (**%rsp**).
20. **popq %rbp**: Restaura el valor del puntero de la base de la pila (**%rbp**) desde la pila.
21. **retq**: Retorna de la función **_func0**.

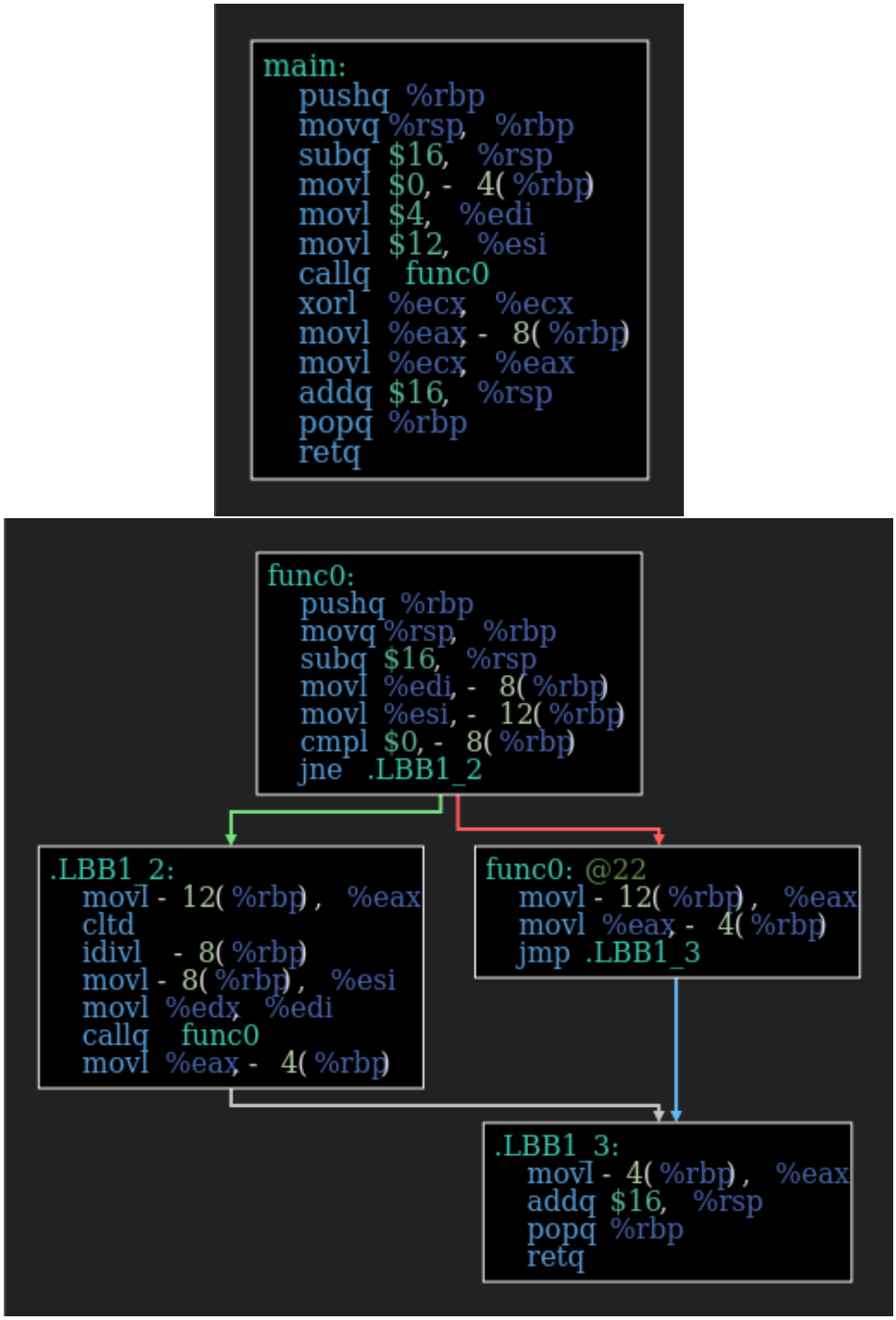
Este codigo realiza operaciones aritméticas y de control de flujo basadas en las comparaciones y saltos condicionados. Finalmente retorna un valor.

Codigo en C:

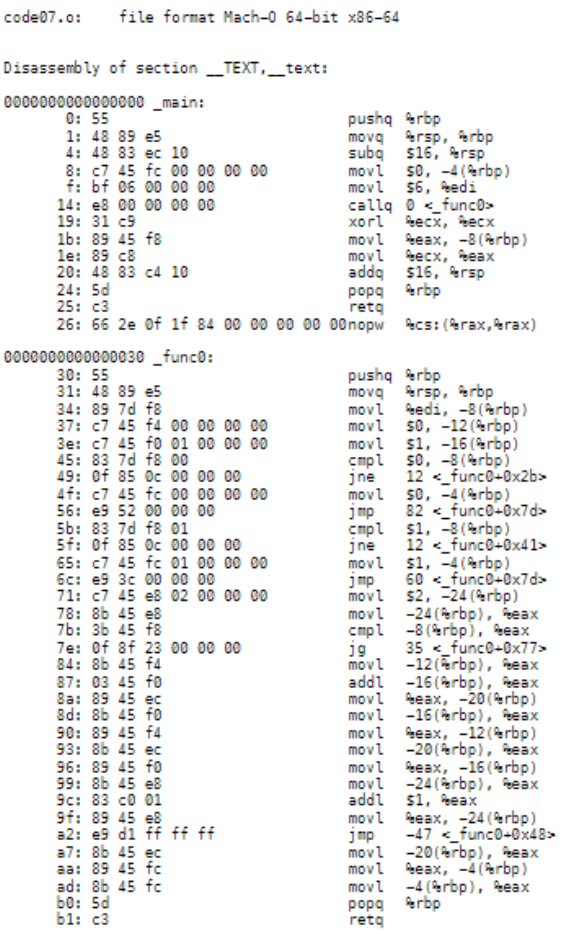
```
int func0(int a, int b);

int main() {
   int result = func0(4, 12);
   return 0;
}
int func0(int a, int b) {
   if (a == 0) return b;
   return func0(b % a, a);
}
```

Explicacion: El código proporcionado implementa un programa en el lenguaje de programación C para calcular el máximo común divisor (MCD) de dos números enteros utilizando el algoritmo de Euclides. El programa define una función llamada func0 que toma dos parámetros enteros (a y b) y devuelve el MCD de ambos. La función main del programa invoca func0 con los valores 4 y 12, y almacena el resultado en la variable result.

La función func0 realiza la operación recursiva del algoritmo de Euclides. Si el primer parámetro (a) es igual a cero, la función retorna el segundo parámetro (b), estableciendo así la condición base de la recursión. En caso contrario, la función se llama a sí misma con los argumentos modificados, siendo el nuevo valor de a el residuo de la división de b entre a, y b toma el valor original de a. Este proceso continúa hasta que a alcanza cero, momento en el cual se retorna el valor de b, que representa el MCD buscado.



Código 3:



```
code07.o:    file format Mach-O 64-bit x86-64

Disassembly of section __TEXT,__text:

0000000000000000 _main:
       0: 55                            pushq  %rbp
       1: 48 89 e5                      movq   %rsp, %rbp
       4: 48 83 ec 10                   subq   $16, %rsp
       8: c7 45 fc 00 00 00 00          movl   $0, -4(%rbp)
       f: bf 06 00 00 00                movl   $6, %edi
      14: e8 00 00 00 00                callq  0 <_func0>
      19: 31 c9                         xorl   %ecx, %ecx
      1b: 89 45 f8                      movl   %eax, -8(%rbp)
      1e: 89 c8                         movl   %ecx, %eax
      20: 48 83 c4 10                   addq   $16, %rsp
      24: 5d                            popq   %rbp
      25: c3                            retq
      26: 66 2e 0f 1f 84 00 00 00 00 00nopw   %cs:(%rax,%rax)

0000000000000030 _func0:
      30: 55                            pushq  %rbp
      31: 48 89 e5                      movq   %rsp, %rbp
      34: 89 7d f8                      movl   %edi, -8(%rbp)
      37: c7 45 f4 00 00 00 00          movl   $0, -12(%rbp)
      3e: c7 45 f0 01 00 00 00          movl   $1, -16(%rbp)
      45: 83 7d f8 00                   cmpl   $0, -8(%rbp)
      49: 0f 85 0c 00 00 00             jne    12 <_func0+0x2b>
      4f: c7 45 fc 00 00 00 00          movl   $0, -4(%rbp)
      56: e9 52 00 00 00                jmp    82 <_func0+0x7d>
      5b: 83 7d f8 01                   cmpl   $1, -8(%rbp)
      5f: 0f 85 0c 00 00 00             jne    12 <_func0+0x41>
      65: c7 45 fc 01 00 00 00          movl   $1, -4(%rbp)
      6c: e9 3c 00 00 00                jmp    60 <_func0+0x7d>
      71: c7 45 e8 02 00 00 00          movl   $2, -24(%rbp)
      78: 8b 45 e8                      movl   -24(%rbp), %eax
      7b: 3b 45 f8                      cmpl   -8(%rbp), %eax
      7e: 0f 8f 23 00 00 00             jg     35 <_func0+0x77>
      84: 8b 45 f4                      movl   -12(%rbp), %eax
      87: 03 45 f0                      addl   -16(%rbp), %eax
      8a: 89 45 ec                      movl   %eax, -20(%rbp)
      8d: 8b 45 f0                      movl   -16(%rbp), %eax
      90: 89 45 f4                      movl   %eax, -12(%rbp)
      93: 8b 45 ec                      movl   -20(%rbp), %eax
      96: 89 45 f0                      movl   %eax, -16(%rbp)
      99: 8b 45 e8                      movl   -24(%rbp), %eax
      9c: 83 c0 01                      addl   $1, %eax
      9f: 89 45 e8                      movl   %eax, -24(%rbp)
      a2: e9 d1 ff ff ff                jmp    -47 <_func0+0x48>
      a7: 8b 45 ec                      movl   -20(%rbp), %eax
      aa: 89 45 fc                      movl   %eax, -4(%rbp)
      ad: 8b 45 fc                      movl   -4(%rbp), %eax
      b0: 5d                            popq   %rbp
      b1: c3                            retq
```

Fig 3. Código 3/Code 07.

Análisis:
Código Main:

1. **pushq %rbp**: Guarda el valor actual de **%rbp** en la pila.
2. **movq %rsp, %rbp**: Establece el puntero de marco de pila (**%rbp**) al valor actual del puntero de pila (**%rsp**).
3. **subq $16, %rsp**: Reserva espacio en la pila para variables locales.
4. **movl $0, -4(%rbp)**: Inicializa una variable local a 0.
5. **movl $6, %edi**: Prepara el argumento para la llamada a la función.
6. **callq 0 <_func0>**: Llama a la función **_func0**.
7. **xorl %ecx, %ecx**: Inicializa **%ecx** a 0.
8. **movl %eax, -8(%rbp)**: Almacena el resultado de la función en una variable local.

9. **movl %ecx, %eax**: Devuelve 0 (el resultado de la función) en **%eax**.
10. **addq $16, %rsp**: Libera el espacio de la pila reservado para variables locales.
11. **popq %rbp**: Restaura el valor original de **%rbp** desde la pila.
12. **retq**: Retorna de la función.

La linea 26 es una instruction nopw que no realiza ninguna operación y se utiliza para llenar el espacio de memoria. En este caso no parece ser una secuencia de operación.

Código_fun0:

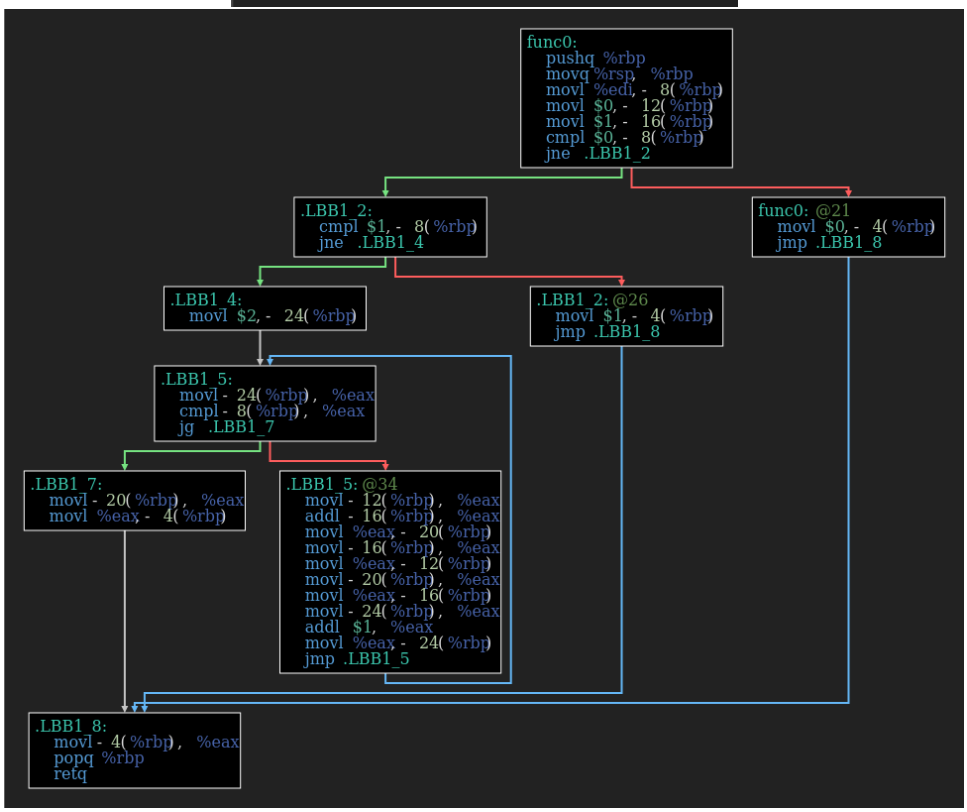
1. **pushq %rbp**: Guarda el valor actual del registro de base de pila (**rbp**) en la pila.
2. **movq %rsp, %rbp**: Establece el registro de base de pila (**rbp**) con el valor actual del puntero de pila (**rsp**).
3. **movl %edi, -8(%rbp)**: Mueve el contenido del registro **edi** al espacio reservado en la pila a 8 bytes desde **rbp**.
4. **movl $0, -12(%rbp)**: Inicializa una variable en la pila con el valor 0.
5. **movl $1, -16(%rbp)**: Inicializa otra variable en la pila con el valor 1.
6. **cmpl $0, -8(%rbp)**: Compara el contenido de la variable en la pila a 8 bytes desde **rbp** con 0.
7. **jne 12 <_func0+0x2b>**: Salta a la dirección 12 si la comparación anterior no es igual (no es cero).
8. **movl $0, -4(%rbp)**: Si la comparación fue igual (cero), se inicializa otra variable en la pila con el valor 0.
9. **jmp 82 <_func0+0x7d>**: Salto incondicional a la dirección 82, omitiendo el código restante de la función.
10. **cmpl $1, -8(%rbp)**: Compara el contenido de la variable en la pila a 8 bytes desde **rbp** con 1.
11. **jne 12 <_func0+0x41>**: Salta a la dirección 12 si la comparación anterior no es igual (no es cero).
12. **movl $1, -4(%rbp)**: Si la comparación fue igual (cero), se inicializa otra variable en la pila con el valor 1.
13. **jmp 60 <_func0+0x7d>**: Salto incondicional a la dirección 60, omitiendo el código restante de la función.
14. Las líneas de la dirección 71 a la 99 realizan operaciones y comparaciones en función de las variables en la pila, actualizando los valores de las variables y ejecutando saltos condicionales según los resultados de las comparaciones.
15. **jmp -47 <_func0+0x48>**: Salto incondicional a la dirección -47, repitiendo el ciclo de ejecución de las líneas 71 a 99.
16. **movl -20(%rbp), %eax**: Mueve el contenido de la variable en la pila a 20 bytes desde **rbp** al registro **eax**.
17. **movl %eax, -4(%rbp)**: Guarda el valor en **eax** en otra variable en la pila a 4 bytes desde **rbp**.
18. **movl -4(%rbp), %eax**: Mueve el contenido de la variable en la pila a 4 bytes desde **rbp** al registro **eax**.
19. **popq %rbp**: Restaura el valor anterior del registro de base de pila (**rbp**) desde la pila.
20. **retq**: Retorna de la función.

Codigo en C:

```
int func0(int a);
int main() {
   int result = func0(6);

   return 0;
}

int func0(int a) {
   int b = 0;
   int c = 1;
   if (a == 0) {
      return 0;

   } else if (a == 1) {
      return 1;
```

```
    } else {
        int x, y = 2;

        while (y <= a) {
            x = b + c;
            b = c;
            c = x;
            y = y + 1;
        }

        return x;
    }
}
```

```
main:
    pushq  %rbp
    movq %rsp,   %rbp
    subq  $16,   %rsp
    movl  $0, -  4(%rbp)
    movl  $6,   %edi
    callq   func0
    xorl   %ecx,   %ecx
    movl  %eax, -  8(%rbp)
    movl  %ecx,   %eax
    addq  $16,   %rsp
    popq  %rbp
    retq
```



Explicacion: Este código en C define una función llamada func0 que calcula el término en la posición a de la secuencia de Fibonacci. La secuencia de Fibonacci es una serie de números en la que cada número es la suma de los dos anteriores (comenzando con 0 y 1).

La función func0 toma un solo parámetro a, que representa la posición en la secuencia de Fibonacci. La función principal (main) llama a func0 con el valor 6 y almacena el resultado en la variable result.

La implementación de func0 utiliza un bucle while para iterar desde 2 hasta a, calculando los términos de la secuencia de Fibonacci mediante una actualización de variables (x, b, y c). Si a es igual a 0, la función retorna 0; si a es igual a 1, retorna 1. En los demás casos, la función calcula el término en la posición a de la secuencia de Fibonacci y retorna ese valor. En este caso, el resultado de llamar func0(6) sería 8, ya que el término en la posición 6 de la secuencia de Fibonacci es 8.

Codigo 4:

```
code08.o:    file format Mach-O 64-bit x86-64

Disassembly of section __TEXT,__text:

0000000000000000 _main:
       0: 55                            pushq  %rbp
       1: 48 89 e5                      movq   %rsp, %rbp
       4: 48 83 ec 10                   subq   $16, %rsp
       8: c7 45 fc 00 00 00 00          movl   $0, -4(%rbp)
       f: bf 06 00 00 00                movl   $6, %edi
      14: e8 00 00 00 00                callq  0 <_func0>
      19: 31 c9                         xorl   %ecx, %ecx
      1b: 89 45 f8                      movl   %eax, -8(%rbp)
      1e: 89 c8                         movl   %ecx, %eax
      20: 48 83 c4 10                   addq   $16, %rsp
      24: 5d                            popq   %rbp
      25: c3                            retq
      26: 66 2e 0f 1f 84 00 00 00 00 00nopw   %cs:(%rax,%rax)

0000000000000030 _func0:
      30: 55                            pushq  %rbp
      31: 48 89 e5                      movq   %rsp, %rbp
      34: 48 83 ec 10                   subq   $16, %rsp
      38: 89 7d f8                      movl   %edi, -8(%rbp)
      3b: 83 7d f8 00                   cmpl   $0, -8(%rbp)
      3f: 0f 85 0c 00 00 00             jne    12 <_func0+0x21>
      45: c7 45 fc 00 00 00 00          movl   $0, -4(%rbp)
      4c: e9 3b 00 00 00                jmp    59 <_func0+0x5c>
      51: 83 7d f8 01                   cmpl   $1, -8(%rbp)
      55: 0f 85 0c 00 00 00             jne    12 <_func0+0x37>
      5b: c7 45 fc 01 00 00 00          movl   $1, -4(%rbp)
      62: e9 25 00 00 00                jmp    37 <_func0+0x5c>
      67: 8b 45 f8                      movl   -8(%rbp), %eax
      6a: 83 e8 01                      subl   $1, %eax
      6d: 89 c7                         movl   %eax, %edi
      6f: e8 bc ff ff ff                callq  -68 <_func0>
      74: 8b 4d f8                      movl   -8(%rbp), %ecx
      77: 83 e9 02                      subl   $2, %ecx
      7a: 89 cf                         movl   %ecx, %edi
      7c: 89 45 f4                      movl   %eax, -12(%rbp)
      7f: e8 ac ff ff ff                callq  -84 <_func0>
      84: 8b 4d f4                      movl   -12(%rbp), %ecx
      87: 01 c1                         addl   %eax, %ecx
      89: 89 4d fc                      movl   %ecx, -4(%rbp)
      8c: 8b 45 fc                      movl   -4(%rbp), %eax
      8f: 48 83 c4 10                   addq   $16, %rsp
      93: 5d                            popq   %rbp
      94: c3                            retq
```

Fig 4. Código 4/Code 08.

Análisis:

Código _main:

1. **pushq %rbp:** Se realiza una operación de push en la pila, guardando el valor actual del puntero de la base de la pila (**%rbp**).
2. **movq %rsp, %rbp:** Establece el puntero de la base de la pila (**%rbp**) al valor actual del puntero de la pila (**%rsp**).
3. **subq $16, %rsp:** Reserva espacio en la pila para variables locales, restando 16 bytes del puntero de la pila (**%rsp**).
4. **movl $0, -4(%rbp):** Inicializa una variable en la pila con el valor 0.
5. **movl $6, %edi:** Prepara el argumento para la llamada a la función **_func0**.
6. **callq 0 <_func0>:** Llama a la función **_func0**.
7. **xorl %ecx, %ecx:** Inicializa el registro **%ecx** a 0.
8. **movl %eax, -8(%rbp):** Guarda el resultado de la función en la pila.
9. **movl %ecx, %eax:** Pone 0 en el registro **%eax**.
10. **addq $16, %rsp:** Libera el espacio reservado en la pila.

11. **popq %rbp:** Restaura el registro de base de la pila.
12. **retq:** Retorna de la función.

Código _func0:

1. **pushq %rbp:** Se realiza una operación de push en la pila, guardando el valor actual del puntero de la base de la pila (**%rbp**).
2. **movq %rsp, %rbp:** Establece el puntero de la base de la pila (**%rbp**) al valor actual del puntero de la pila (**%rsp**).
3. **subq $16, %rsp:** Reserva espacio en la pila para variables locales, restando 16 bytes del puntero de la pila (**%rsp**).
4. **movl %edi, -8(%rbp):** Guarda el argumento en la pila.
5. **cmpl $0, -8(%rbp):** Compara el argumento con 0.
6. **jne 12 <_func0+0x21>:** Salta a la dirección 12 si el resultado de la comparación es distinto de cero.
7. **movl $0, -4(%rbp):** Inicializa una variable en la pila con el valor 0.
8. **jmp 59 <_func0+0x5c>:** Salta a la dirección 59, evitando las instrucciones siguientes.
9. **cmpl $1, -8(%rbp):** Compara el argumento con 1.
10. **jne 12 <_func0+0x37>:** Salta a la dirección 12 si el resultado de la comparación es distinto de cero.
11. **movl $1, -4(%rbp):** Inicializa una variable en la pila con el valor 1.
12. **jmp 37 <_func0+0x5c>:** Salta a la dirección 37, evitando las instrucciones siguientes.
13. **movl -8(%rbp), %eax:** Mueve el argumento a **%eax**.
14. **subl $1, %eax:** Resta 1 de **%eax**.
15. **movl %eax, %edi:** Guarda **%eax** como argumento.
16. **callq -68 <_func0>:** Llama a la función **_func0** de forma recursiva.
17. **movl -8(%rbp), %ecx:** Mueve el argumento a **%ecx**.
18. **subl $2, %ecx:** Resta 2 de **%ecx**.
19. **movl %ecx, %edi:** Guarda **%ecx** como argumento.
20. **callq -84 <_func0>:** Llama a la función **_func0** de forma recursiva.
21. **movl -12(%rbp), %ecx:** Mueve el resultado de la primera llamada a **%ecx**.
22. **addl %eax, %ecx:** Suma el resultado de la segunda llamada.
23. **movl %ecx, -4(%rbp):** Guarda la suma en la pila.

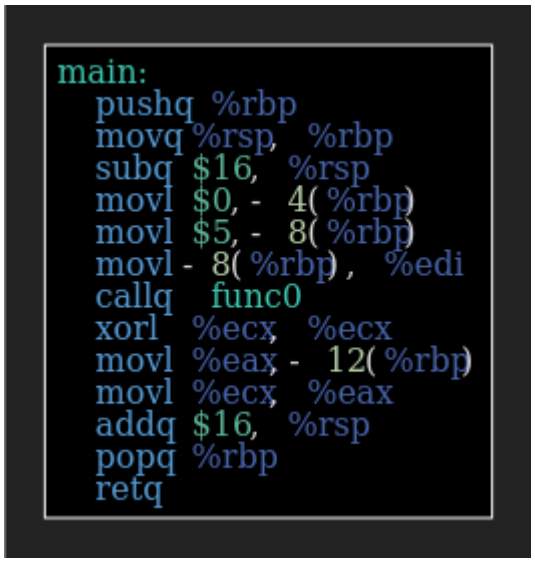
Codigo en C:

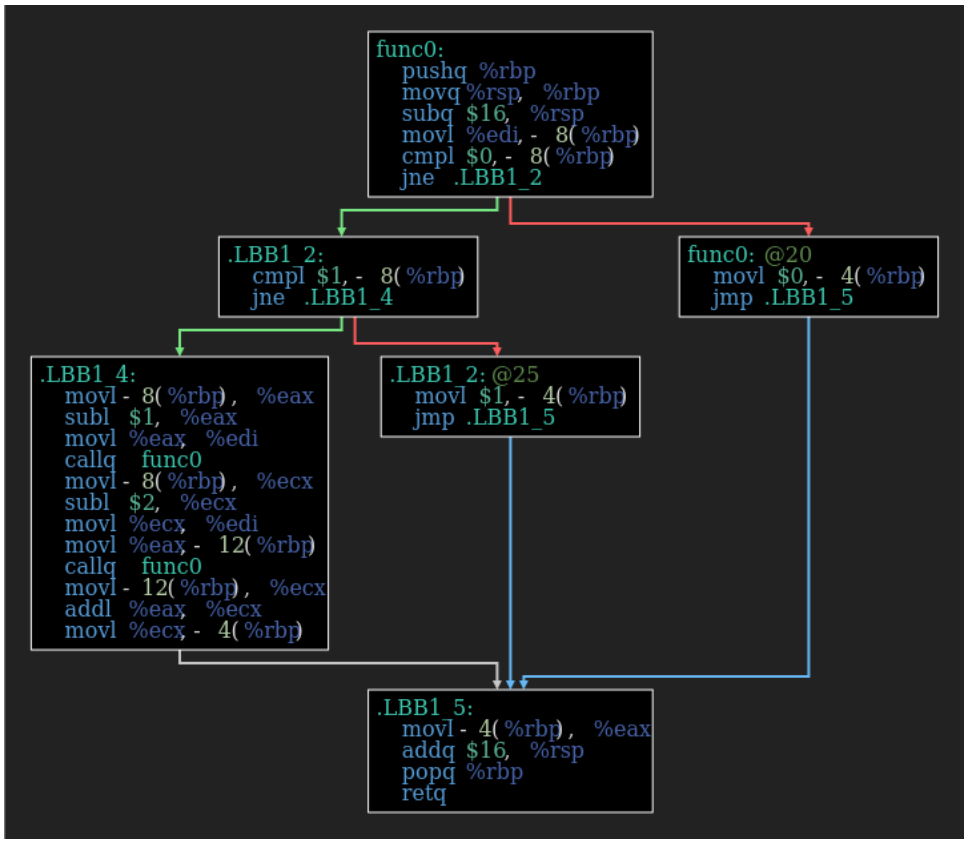
```
int func0(int n);

int main() {
   int result;
   result = func0(6);
   return 0;
}

int func0(int n) {
   if (n == 0) return 0;
   if (n == 1) return 1;
   return func0(n - 1) + func0(n - 2);
}
```

Explicacion: Este código en C implementa una función recursiva para calcular el término en la posición n de la secuencia de Fibonacci. La secuencia de Fibonacci es una serie de números en la que cada número es la suma de los dos anteriores (comenzando con 0 y 1).

La función func0 toma un solo parámetro n, que representa la posición en la secuencia de Fibonacci. La función principal (main) llama a func0 con el valor 6 y almacena el resultado en la variable result.

La implementación de func0 utiliza una estructura recursiva para calcular el término en la posición n de la secuencia de Fibonacci. Si n es igual a 0, la función retorna 0; si n es igual a 1, retorna 1. En otros casos, la función retorna la suma de los términos en las posiciones n-1 y n-2 de la secuencia de Fibonacci, calculados recursivamente llamando a func0 con esos valores. En este caso, el resultado de llamar func0(6) sería 8, ya que el término en la posición 6 de la secuencia de Fibonacci es 8. Es importante notar que esta implementación utiliza una recursión ingenua y puede volverse ineficiente para valores grandes de n debido a la duplicación de cálculos.




func0:
pushq %rbp
movq %rsp, %rbp
subq $16, %rsp
movl %edi, -8(%rbp)
cmpl $0, -8(%rbp)
jne .LBB1_2
.LBB1_2:
cmpl $1, -8(%rbp)
jne .LBB1_4
func0: @20
movl $0, -4(%rbp)
jmp .LBB1_5
.LBB1_4:
movl -8(%rbp), %eax
subl $1, %eax
movl %eax, %edi
callq func0
movl -8(%rbp), %ecx
subl $2, %ecx
movl %ecx, %edi
movl %eax, -12(%rbp)
callq func0
movl -12(%rbp), %ecx
addl %eax, %ecx
movl %ecx, -4(%rbp)
.LBB1_2: @25
movl $1, -4(%rbp)
jmp .LBB1_5
.LBB1_5:
movl -4(%rbp), %eax
addq $16, %rsp
popq %rbp
retq

Codigo 5:

```
code09.o:    file format Mach-O 64-bit x86-64

Disassembly of section __TEXT,__text:

0000000000000000 _main:
       0: 55                            pushq  %rbp
       1: 48 89 e5                      movq   %rsp, %rbp
       4: 48 83 ec 10                   subq   $16, %rsp
       8: c7 45 fc 00 00 00 00          movl   $0, -4(%rbp)
       f: c7 45 f8 05 00 00 00          movl   $5, -8(%rbp)
      16: 8b 7d f8                      movl   -8(%rbp), %edi
      19: e8 00 00 00 00                callq  0 <_func0>
      1e: 31 c9                         xorl   %ecx, %ecx
      20: 89 45 f4                      movl   %eax, -12(%rbp)
      23: 89 c8                         movl   %ecx, %eax
      25: 48 83 c4 10                   addq   $16, %rsp
      29: 5d                            popq   %rbp
      2a: c3                            retq
      2b: 0f 1f 44 00 00                nopl   (%rax,%rax)

0000000000000030 _func0:
      30: 55                            pushq  %rbp
      31: 48 89 e5                      movq   %rsp, %rbp
      34: 89 7d fc                      movl   %edi, -4(%rbp)
      37: c7 45 f8 00 00 00 00          movl   $0, -8(%rbp)
      3e: 8b 45 fc                      movl   -4(%rbp), %eax
      41: 89 45 f4                      movl   %eax, -12(%rbp)
      44: 83 7d f4 00                   cmpl   $0, -12(%rbp)
      48: 0f 8e 17 00 00 00             jle    23 <_func0+0x35>
      4e: 8b 45 f4                      movl   -12(%rbp), %eax
      51: 03 45 f8                      addl   -8(%rbp), %eax
      54: 89 45 f8                      movl   %eax, -8(%rbp)
      57: 8b 45 f4                      movl   -12(%rbp), %eax
      5a: 83 c0 ff                      addl   $-1, %eax
      5d: 89 45 f4                      movl   %eax, -12(%rbp)
      60: e9 df ff ff ff                jmp    -33 <_func0+0x14>
      65: 8b 45 f8                      movl   -8(%rbp), %eax
      68: 5d                            popq   %rbp
      69: c3                            retq
```

Fig 5. Código 5/Code 09.

Análisis:

Código Main:

1. **pushq %rbp**: Guarda el valor actual del registro de base de pila (rbp) en la pila.
2. **movq %rsp, %rbp**: Establece el registro de base de pila (rbp) con el valor actual del puntero de pila (rsp).
3. **subq $16, %rsp**: Reserva espacio en la pila para variables locales al restar 16 del puntero de pila.
4. **movl $0, -4(%rbp)**: Inicializa una variable local en -4(%rbp) con el valor 0.
5. **movl $5, -8(%rbp)**: Inicializa otra variable local en -8(%rbp) con el valor 5.
6. **movl -8(%rbp), %edi**: Mueve el valor de la variable local en -8(%rbp) al registro %edi, que es usado para pasar argumentos a funciones.
7. **callq 0 <_func0>**: Llama a la función **_func0**.
8. **xorl %ecx, %ecx**: XOR lógico de %ecx consigo mismo, estableciéndolo en 0.
9. **movl %eax, -12(%rbp)**: Mueve el resultado de la función **_func0** a una variable local en -12(%rbp).
10. **movl %ecx, %eax**: Mueve 0 a %eax.
11. **addq $16, %rsp**: Libera el espacio reservado en la pila.
12. **popq %rbp**: Restaura el registro de base de pila.
13. **retq**: Retorna de la función principal.

Es importante señalar que el código **nopl (%rax,%rax)** no realiza ninguna operación y es simplemente un lugar de no operación.

Código_func0:

1. **pushq %rbp**: Guarda el valor actual del registro de base de pila (rbp) en la pila.
2. **movq %rsp, %rbp**: Establece el registro de base de pila (rbp) con el valor actual del puntero de pila (rsp).
3. **movl %edi, -4(%rbp)**: Mueve el valor del registro %edi (argumento de la función) a una variable local en -4(%rbp).
4. **movl $0, -8(%rbp)**: Inicializa otra variable local en -8(%rbp) con el valor 0.
5. **movl -4(%rbp), %eax**: Mueve el valor de la variable local en -4(%rbp) al registro %eax.
6. **movl %eax, -12(%rbp)**: Mueve el contenido de %eax a una variable local en -12(%rbp).
7. **cmpl $0, -12(%rbp)**: Compara el valor de la variable local en -12(%rbp) con 0.
8. **jle 23 <_func0+0x35>**: Salta a la etiqueta 23 si el resultado de la comparación es menor o igual a cero.
9. **movl -12(%rbp), %eax**: Mueve el valor de la variable local en -12(%rbp) al registro %eax.
10. **addl -8(%rbp), %eax**: Suma el contenido de la variable local en -8(%rbp) a %eax.
11. **movl %eax, -8(%rbp)**: Mueve el resultado de la suma a la variable local en -8(%rbp).
12. **movl -12(%rbp), %eax**: Mueve el valor de la variable local en -12(%rbp) a %eax.
13. **addl $-1, %eax**: Resta 1 al contenido de %eax.
14. **movl %eax, -12(%rbp)**: Mueve el resultado de la resta a la variable local en -12(%rbp).
15. **jmp -33 <_func0+0x14>**: Salto incondicional a la etiqueta -33, reiniciando el ciclo.
16. **movl -8(%rbp), %eax**: Mueve el contenido de la variable local en -8(%rbp) al registro %eax.
17. **popq %rbp**: Restaura el registro de base de pila.
18. **retq**: Retorna de la función **_func0**.

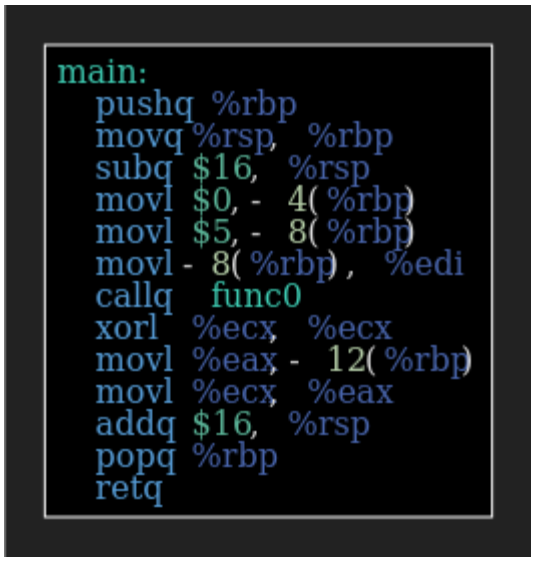
Codigo en C:

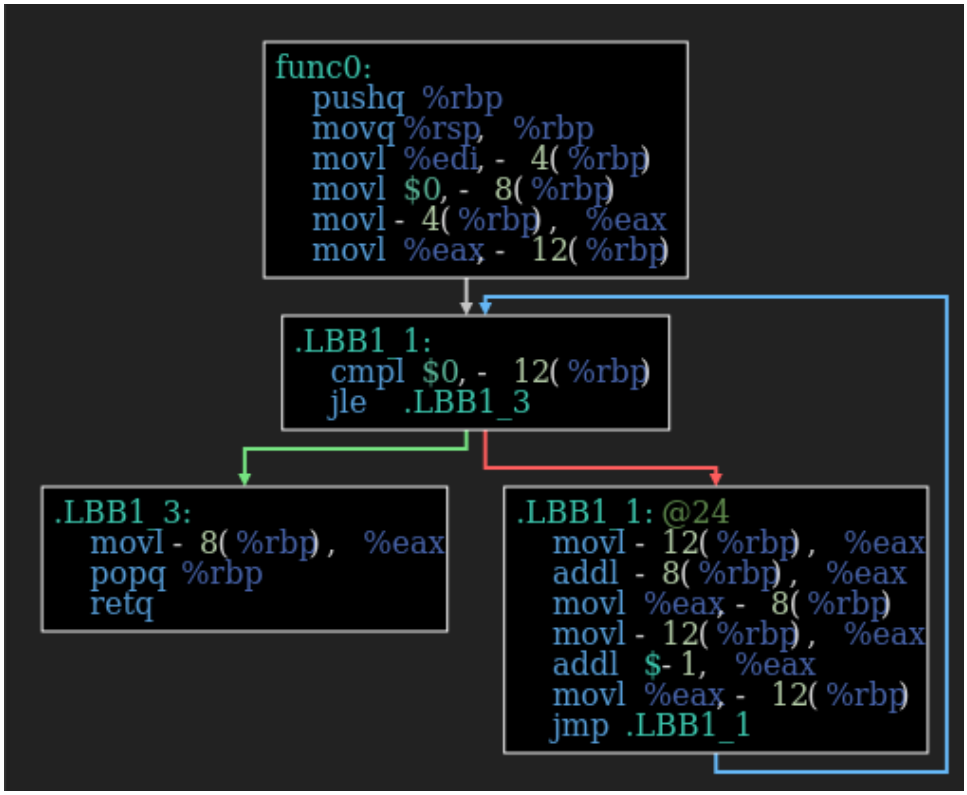
```
int func0(int n);

int main() {
   int result = 0, n = 5;
   int x = func0(n);
}

int func0(int n) {
   int result = 0, a = n;
   while (a > 0) {
      result += a, a--;
   }
   return result;
}
```

Explicacion: Este código en C define una función llamada func0 que calcula la suma de los números desde n hasta 1 mediante un bucle while. Luego, en la función principal (main), se llama a func0 con n igual a 5 y el resultado se almacena en la variable x.

La función func0 toma un solo parámetro n, que representa el número hasta el cual se realizará la suma. Dentro de la función, se utiliza un bucle while que acumula la suma de los valores de n a 1 en la variable result. Después de cada iteración, el valor de a se decrementa en 1. La función retorna la suma total.

En el ejemplo proporcionado, al llamar a func0 con n igual a 5, se realizará la suma $5+4+3+2+1$5+4+3+2+1, y el resultado se almacenará en la variable x. El resultado final será 15.


main:
pushq %rbp
movq %rsp, %rbp
subq $16, %rsp
movl $0, -4(%rbp)
movl $5, -8(%rbp)
movl -8(%rbp), %edi
callq func0
xorl %ecx, %ecx
movl %eax, -12(%rbp)
movl %ecx, %eax
addq $16, %rsp
popq %rbp
retq


func0:
pushq %rbp
movq %rsp, %rbp
movl %edi, -4(%rbp)
movl $0, -8(%rbp)
movl -4(%rbp), %eax
movl %eax, -12(%rbp)
.LBB1_1:
cmpl $0, -12(%rbp)
jle .LBB1_3
.LBB1_3:
movl -8(%rbp), %eax
popq %rbp
retq
.LBB1_1: @24
movl -12(%rbp), %eax
addl -8(%rbp), %eax
movl %eax, -8(%rbp)
movl -12(%rbp), %eax
addl $-1, %eax
movl %eax, -12(%rbp)
jmp .LBB1_1

Código 6:

```
code10.o:    file format Mach-O 64-bit x86-64


Disassembly of section __TEXT,__text:

0000000000000000 _main:
       0: 55                            pushq  %rbp
       1: 48 89 e5                      movq   %rsp, %rbp
       4: 48 83 ec 10                   subq   $16, %rsp
       8: c7 45 fc 00 00 00 00          movl   $0, -4(%rbp)
       f: c7 45 f8 05 00 00 00          movl   $5, -8(%rbp)
      16: 8b 7d f8                      movl   -8(%rbp), %edi
      19: e8 00 00 00 00                callq  0 <_func0>
      1e: 31 c9                         xorl   %ecx, %ecx
      20: 89 45 f4                      movl   %eax, -12(%rbp)
      23: 89 c8                         movl   %ecx, %eax
      25: 48 83 c4 10                   addq   $16, %rsp
      29: 5d                            popq   %rbp
      2a: c3                            retq
      2b: 0f 1f 44 00 00                nopl   (%rax,%rax)

0000000000000030 _func0:
      30: 55                            pushq  %rbp
      31: 48 89 e5                      movq   %rsp, %rbp
      34: 48 83 ec 10                   subq   $16, %rsp
      38: 89 7d f8                      movl   %edi, -8(%rbp)
      3b: 83 7d f8 00                   cmpl   $0, -8(%rbp)
      3f: 0f 84 20 00 00 00             je     32 <_func0+0x35>
      45: 8b 45 f8                      movl   -8(%rbp), %eax
      48: 8b 4d f8                      movl   -8(%rbp), %ecx
      4b: 83 e9 01                      subl   $1, %ecx
      4e: 89 cf                         movl   %ecx, %edi
      50: 89 45 f4                      movl   %eax, -12(%rbp)
      53: e8 d8 ff ff ff                callq  -40 <_func0>
      58: 8b 4d f4                      movl   -12(%rbp), %ecx
      5b: 01 c1                         addl   %eax, %ecx
      5d: 89 4d fc                      movl   %ecx, -4(%rbp)
      60: e9 06 00 00 00                jmp    6 <_func0+0x3b>
      65: 8b 45 f8                      movl   -8(%rbp), %eax
      68: 89 45 fc                      movl   %eax, -4(%rbp)
      6b: 8b 45 fc                      movl   -4(%rbp), %eax
      6e: 48 83 c4 10                   addq   $16, %rsp
      72: 5d                            popq   %rbp
      73: c3                            retq
```

Fig 6. Código 6/Code 10.

Análisis:

Código Main:

1. **movq %rsp, %rbp**: Establece el registro de base de pila (**rbp**) al valor actual del puntero de pila (**rsp**), creando un nuevo marco de pila.
2. **subq $16, %rsp**: Reserva espacio en la pila para variables locales restando 16 bytes del puntero de pila (**rsp**).
3. **movl $0, -4(%rbp)**: Inicializa una variable local colocando el valor 0 en la posición de memoria 4 bytes antes del registro de base de pila (**rbp**).
4. **movl $5, -8(%rbp)**: Inicializa otra variable local colocando el valor 5 en la posición de memoria 8 bytes antes del registro de base de pila (**rbp**).
5. **movl -8(%rbp), %edi**: Mueve el valor de la variable local (5) al registro **edi**.
6. **callq 0 <_func0>**: Llama a la función **_func0**.
7. **xorl %ecx, %ecx**: Hace un XOR lógico del registro **ecx** consigo mismo, estableciéndolo en 0.
8. **movl %eax, -12(%rbp)**: Guarda el resultado de la función **_func0** en una variable local.

9. **movl %ecx, %eax**: Mueve el valor del registro **ecx** al registro **eax**.
10. **addq $16, %rsp**: Libera el espacio en la pila reservado anteriormente.
11. **popq %rbp**: Restaura el valor del registro de base de pila (**rbp**) desde la pila.
12. **retq**: Retorna de la función principal (**main**).
13. **nopl (%rax,%rax)**: No realiza ninguna operación (no-operation).

Código _fun0:

14. **pushq %rbp**: Guarda el valor actual del registro de base de pila (**rbp**) en la pila.
15. **movq %rsp, %rbp**: Establece el registro de base de pila (**rbp**) al valor actual del puntero de pila (**rsp**), creando un nuevo marco de pila.
16. **subq $16, %rsp**: Reserva espacio en la pila para variables locales restando 16 bytes del puntero de pila (**rsp**).
17. **movl %edi, -8(%rbp)**: Guarda el valor del registro **edi** (que contiene el argumento de la función) en una variable local en la pila.
18. **cmpl $0, -8(%rbp)**: Compara el valor de la variable local con 0.
19. **je 32 <_func0+0x35>**: Salta a la dirección 32 si la comparación anterior fue verdadera (igual a cero).
20. **movl -8(%rbp), %eax**: Mueve el valor de la variable local al registro **eax**.
21. **movl -8(%rbp), %ecx**: Mueve el valor de la variable local al registro **ecx**.
22. **subl $1, %ecx**: Resta 1 al registro **ecx**.
23. **movl %ecx, %edi**: Mueve el valor del registro **ecx** al registro **edi**.
24. **movl %eax, -12(%rbp)**: Guarda el valor original de la variable local en una nueva posición en la pila.
25. **callq -40 <_func0>**: Llama recursivamente a la función **_func0**.
26. **movl -12(%rbp), %ecx**: Recupera el valor original de la variable local en el registro **ecx**.
27. **addl %eax, %ecx**: Suma el valor actual de la variable local (en **eax**) al valor original (en **ecx**).
28. **movl %ecx, -4(%rbp)**: Guarda el resultado de la suma en una nueva variable local en la pila.
29. **jmp 6 <_func0+0x3b>**: Salta a la dirección 6, reiniciando el bucle.
30. **movl -8(%rbp), %eax**: Mueve el valor de la variable local al registro **eax**.
31. **movl %eax, -4(%rbp)**: Guarda el valor de la variable local en una nueva posición en la pila.
32. **addq $16, %rsp**: Libera el espacio en la pila reservado anteriormente.
33. **popq %rbp**: Restaura el valor del registro de base de pila (**rbp**) desde la pila.
34. **retq**: Retorna de la función **_func0**.

Este código representa una función recursiva que realiza algunas operaciones aritméticas y condicionales. El bucle continúa hasta que el valor de la variable local alcanza cero.

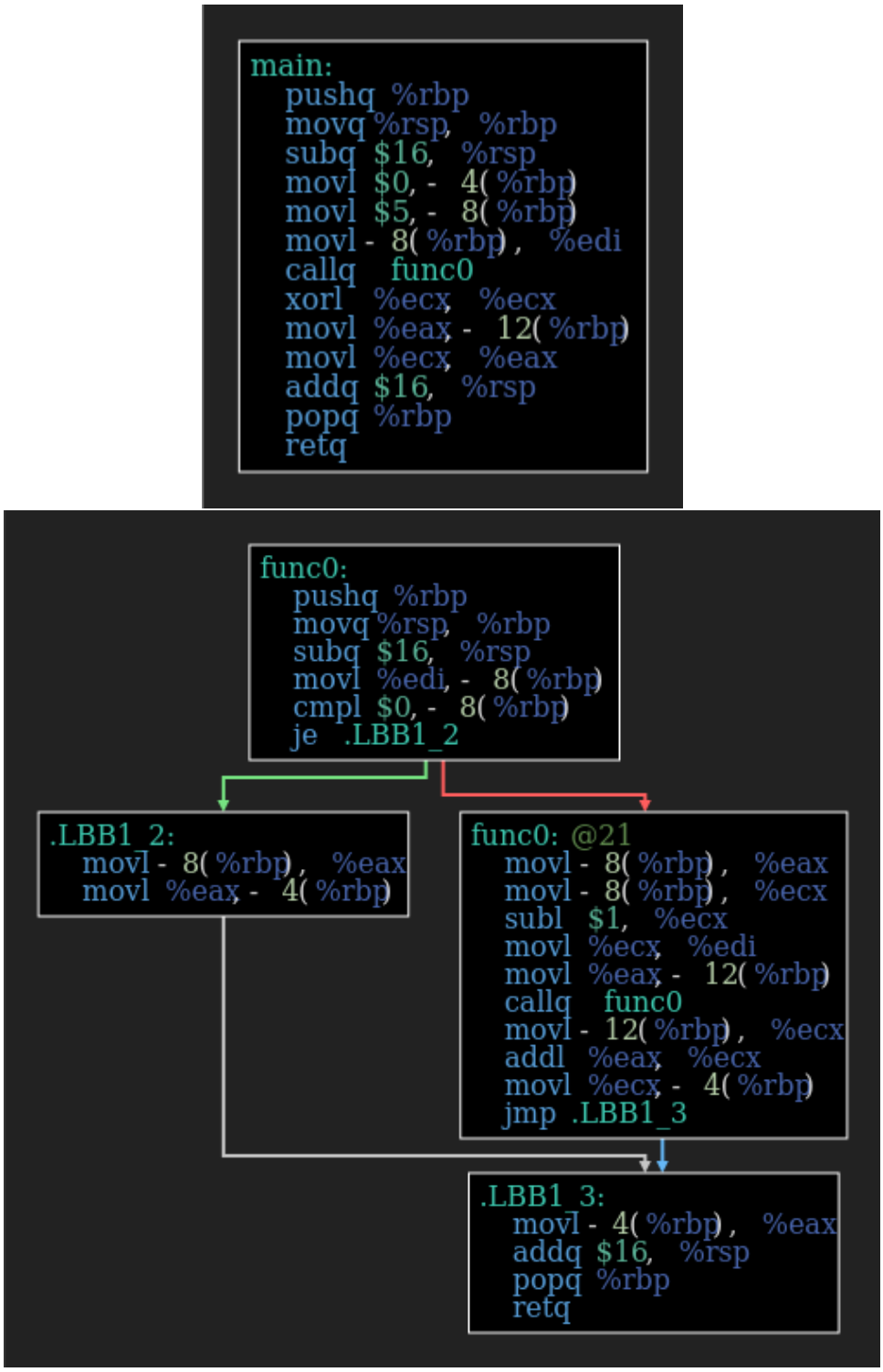
Código en C:

```
int func0(int x);

int main() {
   int result = 5, arg;
   arg = func0(result);
   return 0;
}

int func0(int x) {
   if (x != 0) {
      return x + func0(x - 1);
   }
   return x;
}
```

Explicacion: función recursiva llamada func0 que calcula la suma de los números desde x hasta 0. En el programa principal (main), se llama a func0 con el valor 5, y el resultado se almacena en la variable arg.

La función func0 toma un solo parámetro x. La implementación utiliza una estructura recursiva para calcular la suma de los números desde x hasta 0. La condición base para la recursión es cuando x es igual a 0, en cuyo caso la función retorna 0. En otros casos, la función retorna la suma de x y la llamada recursiva a func0 con el valor decrementado de x.

En el ejemplo proporcionado, al llamar a func0 con x igual a 5, se realizará la suma $5+4+3+2+1+0$5+4+3+2+1+0, y el resultado se almacenará en la variable arg. El resultado final será 15. Es importante señalar que esta implementación utiliza una recursión directa y puede volverse ineficiente para valores grandes de x debido a la profundidad de la recursión.


main:
pushq %rbp
movq %rsp, %rbp
subq $16, %rsp
movl $0, -4(%rbp)
movl $5, -8(%rbp)
movl -8(%rbp), %edi
callq func0
xorl %ecx, %ecx
movl %eax, -12(%rbp)
movl %ecx, %eax
addq $16, %rsp
popq %rbp
retq
func0:
pushq %rbp
movq %rsp, %rbp
subq $16, %rsp
movl %edi, -8(%rbp)
cmpl $0, -8(%rbp)
je .LBB1_2
.LBB1_2:
movl -8(%rbp), %eax
movl %eax, -4(%rbp)
func0: @21
movl -8(%rbp), %eax
movl -8(%rbp), %ecx
subl $1, %ecx
movl %ecx, %edi
movl %eax, -12(%rbp)
callq func0
movl -12(%rbp), %ecx
addl %eax, %ecx
movl %ecx, -4(%rbp)
jmp .LBB1_3
.LBB1_3:
movl -4(%rbp), %eax
addq $16, %rsp
popq %rbp
retq