

Trabajo Integrador

Arquitectura de Computadoras

Procesador Monociclo en Logisim Evolution

Docentes de la cátedra:

Titular: Lic. Claudio Biale

Jefe T.P.: Ing. Roberto A. Miño

Aux: 1ra.: Lic Viviana Arenhardt Aux.

Aux: 2da.: Melissa Kolb

Integrantes:

Ramirez Silvia Beatriz

Garcia Mariano Joaquin

Da Rosa Marcos Gabriel ,

Frias Esteban Hernan ,

Rojas Del Monaco Matias

Carreras:

Analista en Sistemas de Computación

Profesorado Universitario en Computación

Licenciatura en Sistemas de Información

Actividad ..

Se debe implementar una arquitectura Harvard monociclo simplificada que toma como referencia el conjunto de instrucciones de la arquitectura MSP430, con los siguientes componentes principales:

PC (16 bits)

Banco de registros (R4-R15, 12 registros de propósito general)

ALU (16 bits)

ROM instrucciones (16 bits x N palabras)

RAM datos (16 bits x M palabras)

Unidad de control (decodificador)

Registro de flags: Z, N, C, V

Opcode de Instrucciones

Tipo	Opcode
ADD	0000
SUB	0001
OR	0010
AND	0011
CMP	0100
MOV_R a R	0101
MOV-carga desde memoria	0110
MOV-guarda en memoria	0111
MOV -carga constante	1000
ADD-más inmediato	1001
JZ	1010
JNZ	1011
JMP	1100

Flags de la ALU

Flag	Nombre	Activación
Z	Zero	El resultado=0
C	Carry +	El resultado =1
C	Carry -	El resultado= 0
S	Signo	Indica si es positivo (0) o negativo (1)
V	Overflow	Indica si la operación se salió del límite de rango representativo

Formato Tipo R (Registro-Registro)

Suma:

Opcode	Registro Destino	Registro Fuente 1	Registro Fuente 2
4 bits	4 bits	4 bits	4 bits

Resta:

Opcode	Registro Destino	Registro Fuente 1	Registro Fuente 2
4 bits	4 bits	4 bits	4 bits

OR

Opcode	Registro Destino	Registro Fuente 1	Registro Fuente 2
4 bits	4 bits	4 bits	4 bits

AND

Opcode	Registro Destino	Registro Fuente 1	Registro Fuente 2
4 bits	4 bits	4 bits	4 bits

CMP

Opcode	Registro Destino	Registro Fuente 1	Registro Fuente 2
4 bits	4 bits	4 bits	4 bits

Formato Tipo I (Inmediato)

MOV-RaR

Opcode	Registro Destino	Registro Origen	no usado
4	4	4	4

MOV-Carga desde memoria

Opcode	Registro Destino	DirecciónM	no usada

MOV Guarda en memoria

Opcode	Registro Destino	DirecciónM	—

MOV-Carga constante

Opcode	Registro Destino	inmediato	—

Formato Tipo J (Salto)

JZ

Opcode	Registro Destino	Registro Fuente 1	Registro Fuente 2

JNZ

Opcode	Registro Destino	Registro Fuente 1	Registro Fuente 2

JMP

Opcode	Registro Destino	Registro Fuente 1	Registro Fuente 2

Unidad Aritmético Lógica (ALU)

Sumador completo de un bit:

Cin	A	B	Cout	Suma
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Para la Resta:

Cin	A	B	Cout	Suma
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

A modo ilustrativo, no hay mucho por realizar ya que se utilizara una compuerta OR.

A	B	Salida
0	0	0
0	1	1
1	0	1
1	1	1

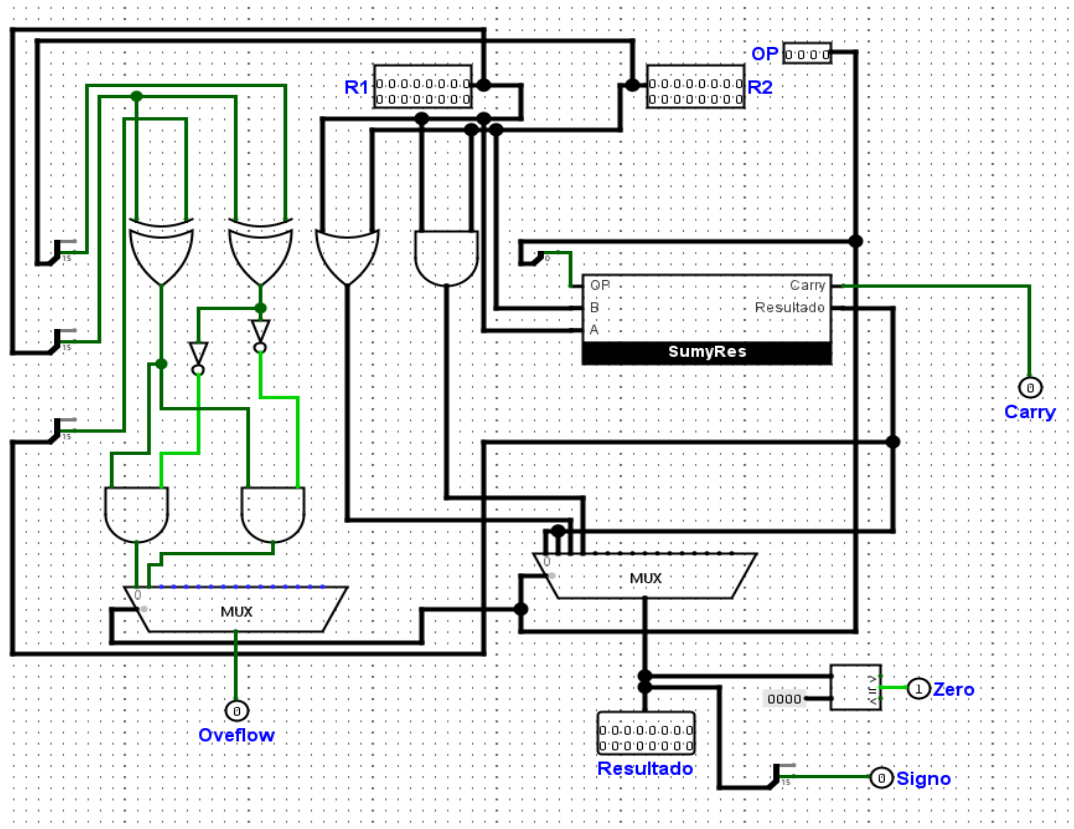
Lo mismo para compuerta AND.

A	B	Salida
0	0	0
0	1	0
1	0	0
1	1	1

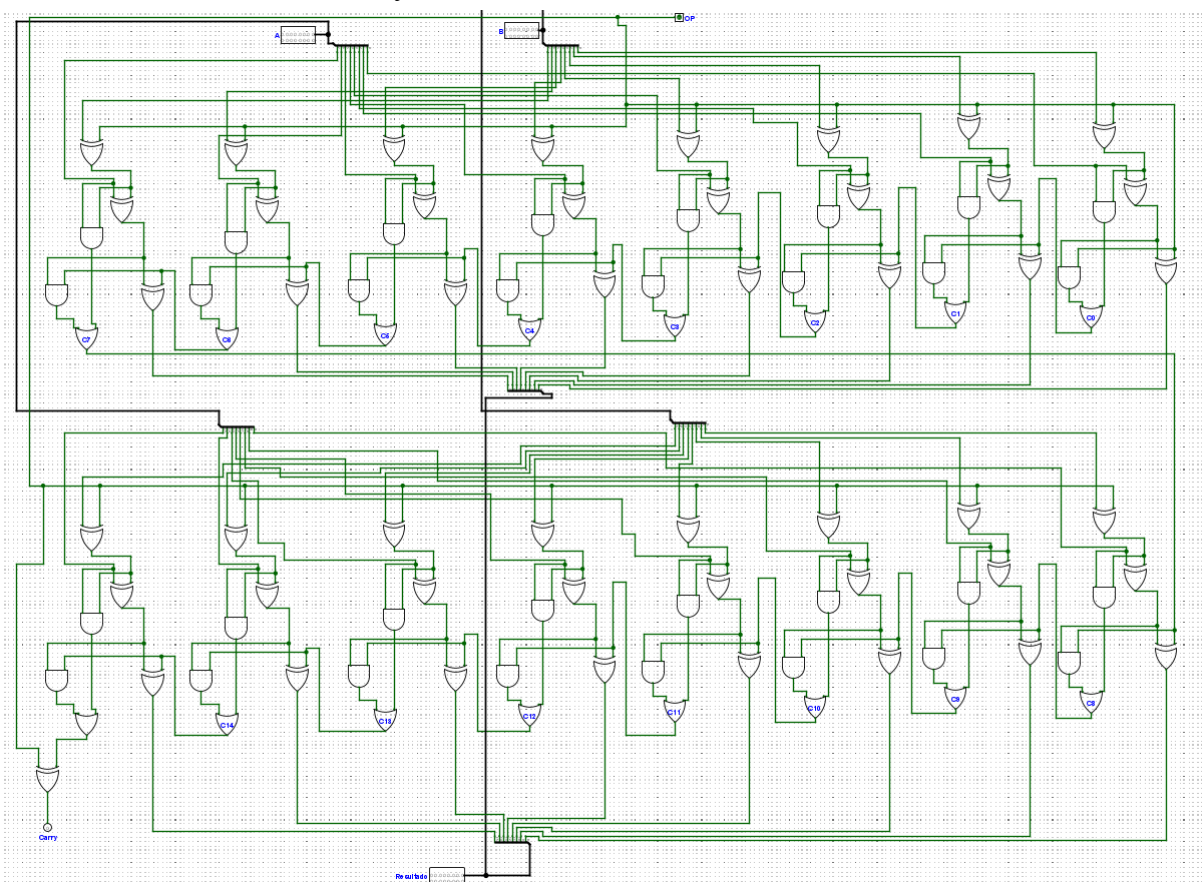
Lo mismo para la compuerta NOT.

A	Negación
0	1
1	0

Parte interna de la ALU

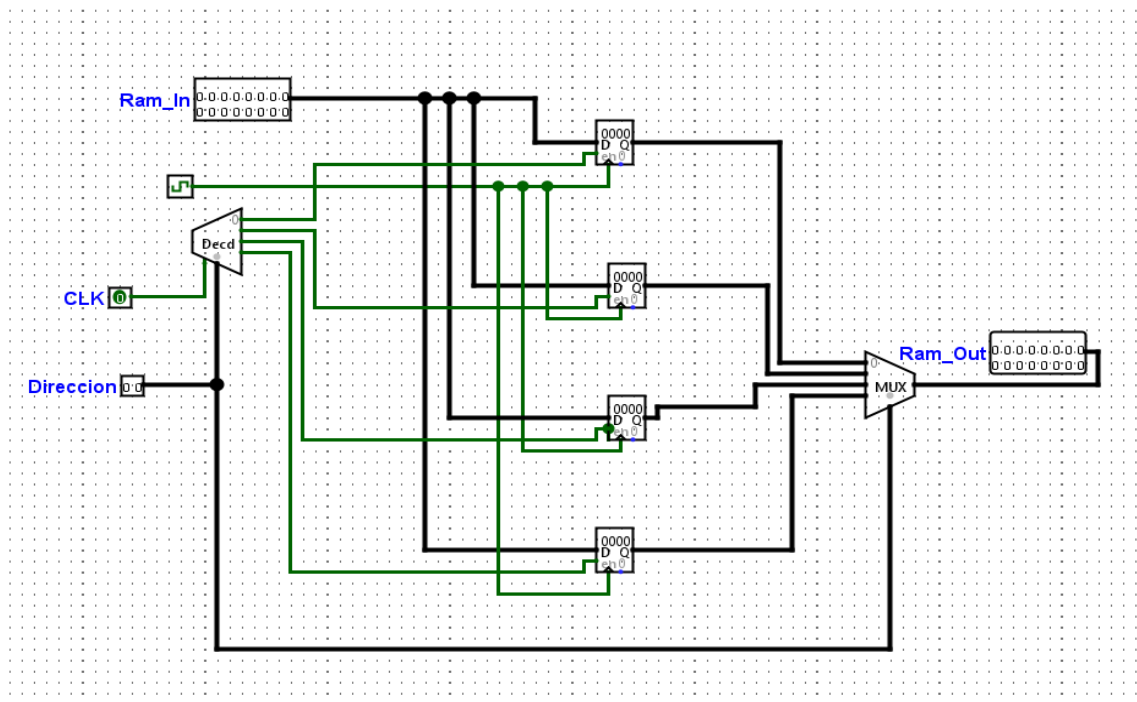


Parte interna del circuito “SumyRes”

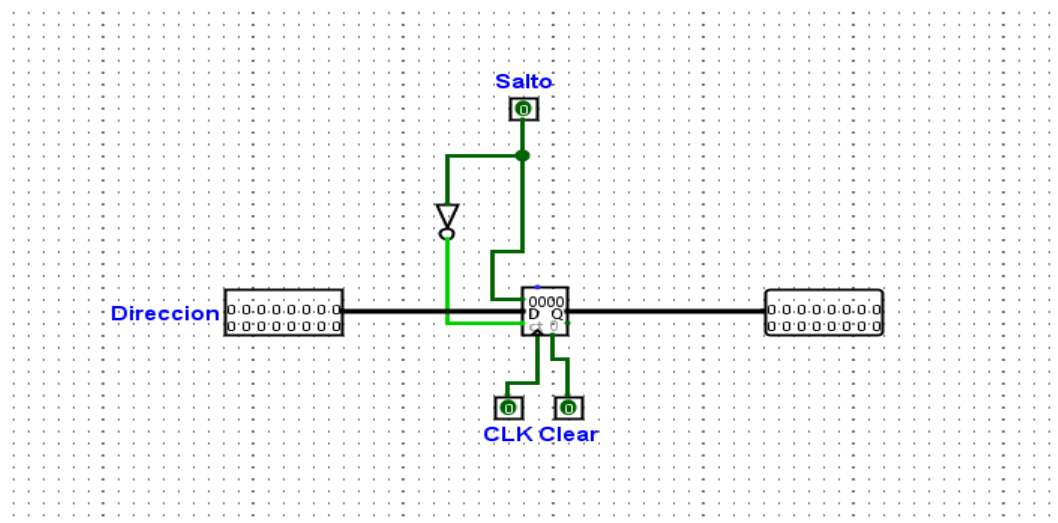


Siguiendo el criterio de evaluación se adaptó el sumador de 4 bits a uno de 16bits

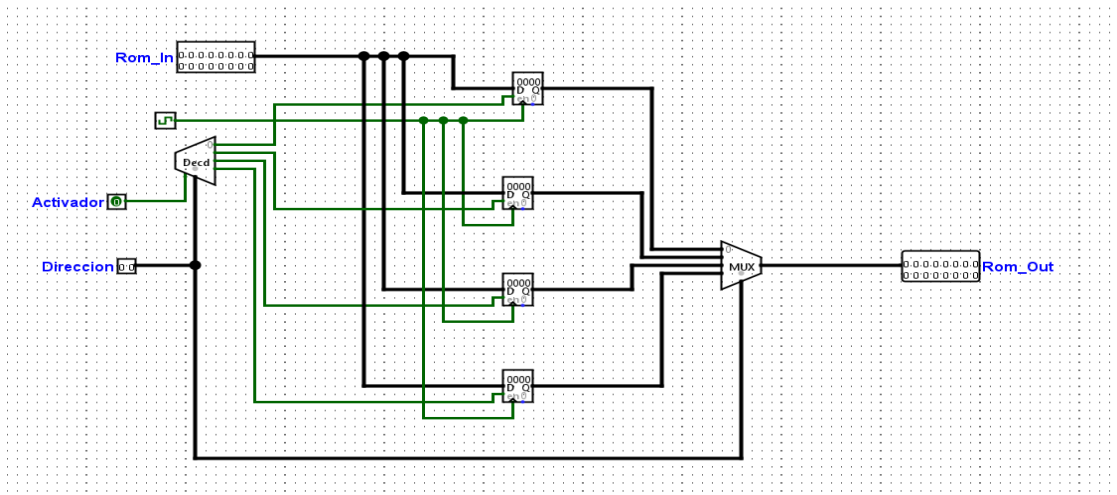
Random Access Memory (RAM)



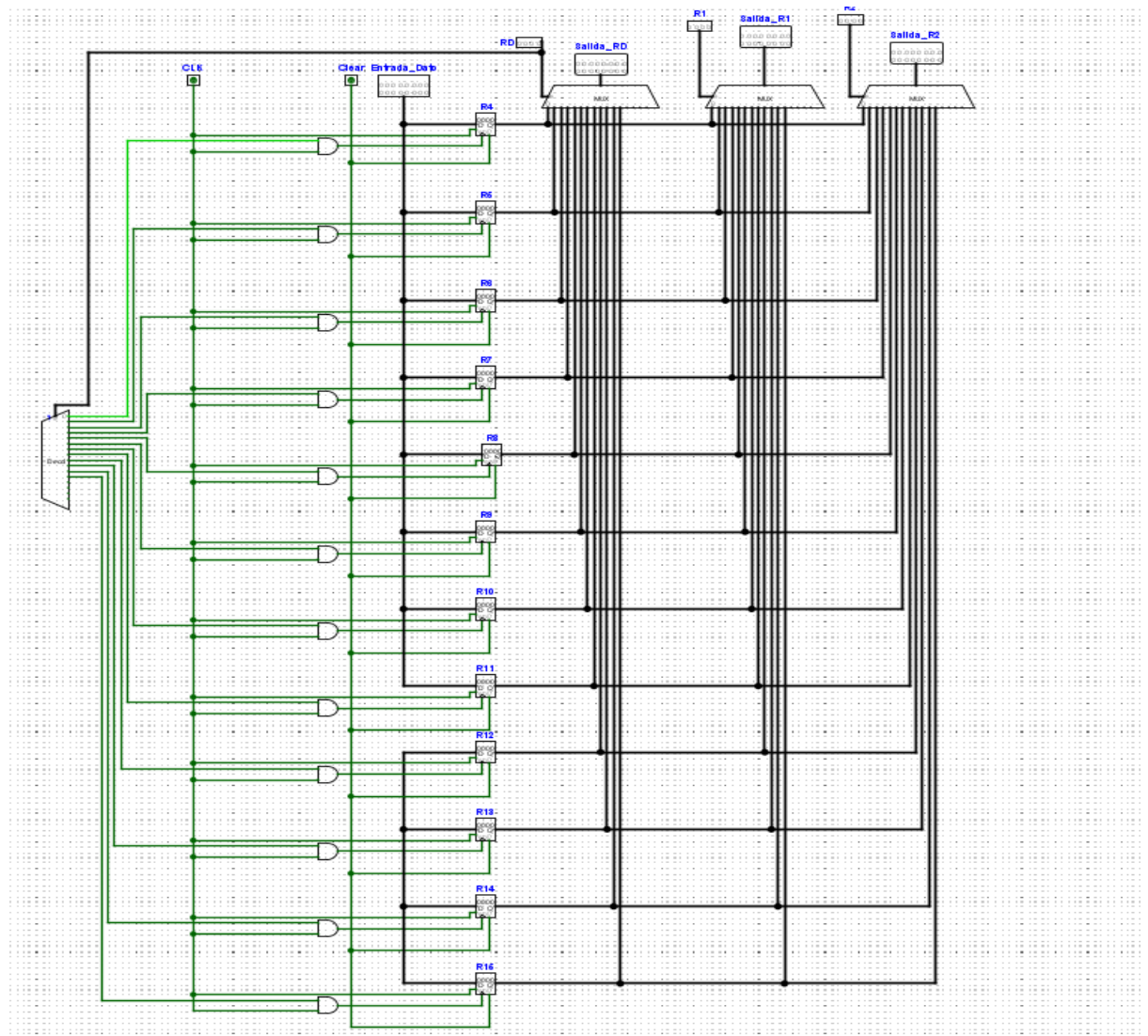
Counter Program (PC)



Read Only Memory (ROM)



Banco de Registros



Parte 3: Consigna

- **Especificaciones del programa:** El array está almacenado en memoria RAM
 - Tamaño del array: 10 elementos
 - Cada elemento es de 16 bits (word)
 - La dirección base del array está en la posición de memoria 0
 - El programa debe buscar el valor 100 dentro del array
- **Funcionamiento:** Recorrer el array elemento por elemento desde el índice 0
 - Comparar cada elemento con el valor buscado (100)
 - Si encuentra el valor: Saltar a la etiqueta encontrado
 - El índice donde se encontró quedará almacenado en R4
 - Si recorre todo el array sin encontrarlo: Continuar a la etiqueta fin
- **Condiciones de terminación:** El programa termina en un loop infinito (etiqueta fin), ya sea que encuentre o no el valor
 - Registros utilizados:
 - R4: Índice actual del array (0 a 9)
 - R5: Valor a buscar (100) R6: Valor leído del array en la posición actual

Desarrollo

A continuación el segmento de código en lenguaje ensamblador Risc-V que cumple con la consigna asignada:

```
1  .data
2  # Array de 10 elementos de 16 bits.
3  array: .half 10, 25, 5, 88, 100, 42, 12, 1, 99, 0
4
5  .text
6  .globl main
7
8  main:
9      li      t0, 0          # R4 = 0 (Índice inicial)
10     li      t1, 100        # R5 = 100 (Valor a buscar)
11     li      t3, 10         # Límite array
12     li      s0, 0
13
14  loop:
15     #Compara el índice con el limite del array
16     beq      t0, t3, fin    # Si índice == 10 salta a fin
17
18     # 3. Calcular dirección logica
19     slli     t4, t0, 1
20     # t5 = Base+offset
21     add      t5, s0, t4
22
23     # 4. Lee memoria
24     la       t6, array      # Carga dirección física
25     add      t6, t6, t5     # Sumamos nuestra dirección lógica a la física.
26     lh       t2, 0(t6)     # Carga el dato sin error.
27
28     # 5. Comparar
29     beq      t2, t1, encontrado # Si R6 == 100, salta a la etiqueta encontrado.
30
31     # 6. Suma el índice
32     addi     t0, t0, 1      # R4 = R4 + 1
33
34     # 7. Repetir
35     j        loop
36
37  encontrado:
38     j        fin
39
40  fin:
41     j        fin           # Loop infinito
42
```

Text Segment						
Bkpt	Address	Code	Basic	Source		
<input type="checkbox"/>	0x00400000	0x00000293	addi x5,x0,0	9:	li	t0, 0 # R4 = 0 (Índice inicial)
<input type="checkbox"/>	0x00400004	0x06400313	addi x6,x0,100	10:	li	t1, 100 # R5 = 100 (Valor a buscar)
<input type="checkbox"/>	0x00400008	0x00a00e13	addi x28,x0,10	11:	li	t3, 10 # Límite array
<input type="checkbox"/>	0x0040000c	0x00000413	addi x8,x0,0	12:	li	s0, 0 # Direccion base de memoria = 0
<input type="checkbox"/>	0x00400010	0x03c28663	beq x5,x28,44	16:	beq	t0, t3, fin # Si indice == 10 salta a fin
<input type="checkbox"/>	0x00400014	0x00129e93	slli x29,x5,1	19:	slli	t4, t0, 1
<input type="checkbox"/>	0x00400018	0x01d40f33	add x30,x8,x29	21:	add	t5, s0, t4
<input type="checkbox"/>	0x0040001c	0x0fc10f97	auipc x31,64528	24:	la	t6, array # Carga dirección física
<input type="checkbox"/>	0x00400020	0xfe4f8f93	addi x31,x31,-28			
<input type="checkbox"/>	0x00400024	0x01ef8fb3	add x31,x31,x30	25:	add	t6, t6, t5 # Sumamos nuestra dirección lógica a la física.
<input type="checkbox"/>	0x00400028	0x000f9383	lh x7,0(x31)	26:	lh	t2, 0(t6) # Carga el dato sin error.
<input type="checkbox"/>	0x0040002c	0x00638663	beq x7,x6,12	29:	beq	t2, t1, encontrado # Si R6 == 100, salta a la etiqueta encontrado.
<input type="checkbox"/>	0x00400030	0x00128293	addi x5,x5,1	32:	addi	t0, t0, 1 # R4 = R4 + 1
<input type="checkbox"/>	0x00400034	0xfddf06f	jal x0,-36	35:	j	loop
<input type="checkbox"/>	0x00400038	0x0040006f	jal x0,4	38:	j	fin
<input type="checkbox"/>	0x0040003c	0x0000006f	jal x0,0	41:	j	fin # Loop infinito

Name	Number	Value
zero	0	0
ra	1	0
sp	2	2147479548
gp	3	268468224
tp	4	0
t0	5	4
t1	6	100
t2	7	100
s0	8	0
s1	9	0
a0	10	0
a1	11	0
a2	12	0
a3	13	0
a4	14	0
a5	15	0
a6	16	0
a7	17	0
s2	18	0
s3	19	0
s4	20	0
s5	21	0
s6	22	0
s7	23	0
s8	24	0
s9	25	0
s10	26	0
s11	27	0
t3	28	10
t4	29	8
t5	30	8
t6	31	268501000
pc		4194364