Trabajo Práctico - Organización de las computadoras

Simulación del RISC-V

Vamos a desglosar el trabajo en 5 etapas

Etapa 1:
Creación de los módulos

Etapa 2:
Conexión de los módulos

Etapa 3:
Modificaciones tipo I - tipo J

Etapa 4:
Diseño del rv32i

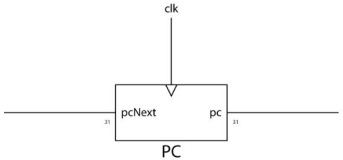
Etapa 5:
Chequeo del procesador

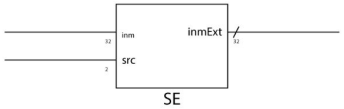
Etapa 1

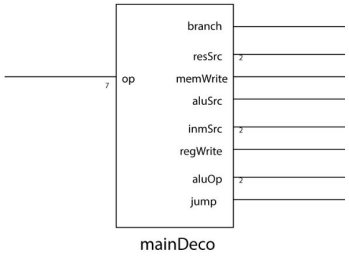
CREAMOS LOS SIGUIENTES CIRCUITOS CORRESPONDIENTES AL RISC-V:

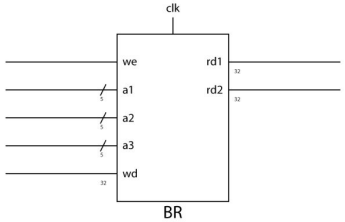
- **PC** (Registro contador de programa)
- **IM** (Memoria de instrucciones)
- **BR** (Banco de registros)
- **DM** (Memoria de datos)
- **SE** (Extensión de signo)
- **ALU** (Unidad aritmético lógica)
- **Adder** (Unidad aritmetico logica suma)
- **Mux2x1** (Multiplexor 2x1)
- **UC** (Unidad de control)
- **mainDeco** (Decodificador Principal)
- **aluDeco** (Decodificador de ALU)

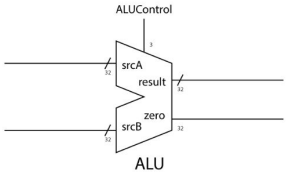
Estructura de los circuitos

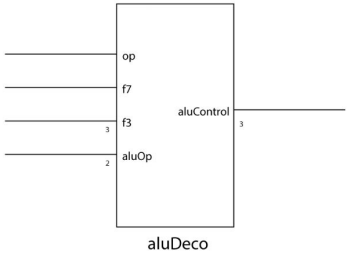


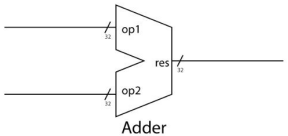


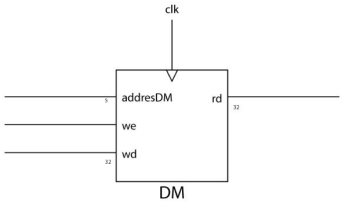


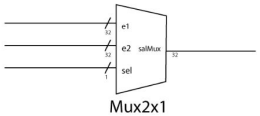


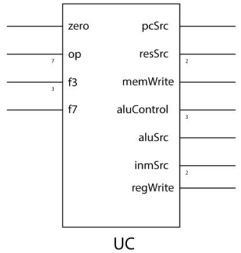


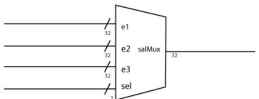






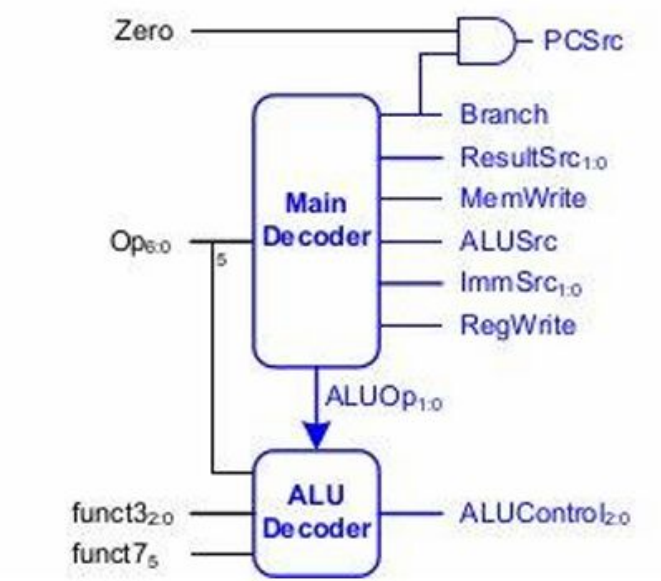




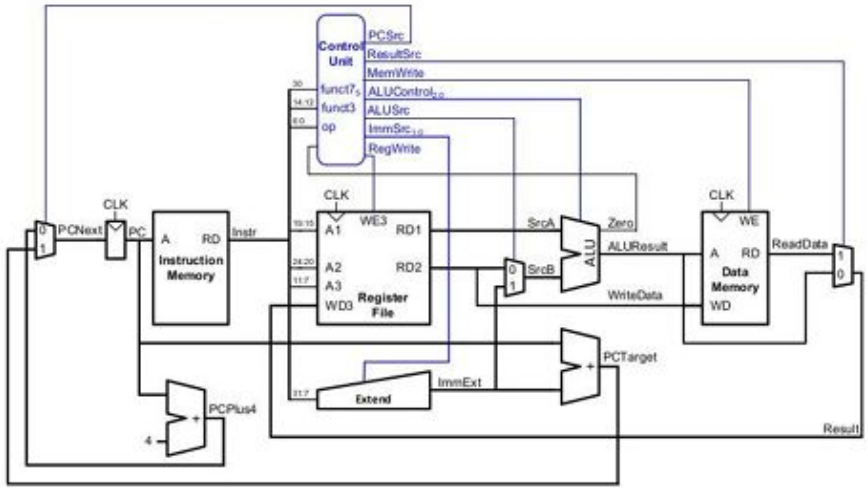


Etapa 2 - Conexión de los módulos

Unidad de control



RV32I



Etapa 3 - Modificaciones para las instrucciones tipo I y tipo J

En mainDeco modificamos la tabla

Antes:

op	Instruct.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
3	lw	1	00	1	0	01	0	00
35	sw	0	01	1	1	XX	0	00
51	R-type	1	XX	0	0	00	0	10
99	beq	0	10	0	0	XX	1	01

Después:

op	Instruct.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	Jump
3	lw	1	00	1	0	01	0	00	0
35	sw	0	01	1	1	XX	0	00	0
51	R-type	1	XX	0	0	00	0	10	0
99	beq	0	10	0	0	XX	1	01	0
19	I-type	1	00	1	0	00	0	10	0
111	jal	1	11	X	0	10	0	XX	1

Etapa 3 - Modificaciones para las instrucciones tipo I y tipo J

En la extensión de signo modificamos la tabla

Antes:

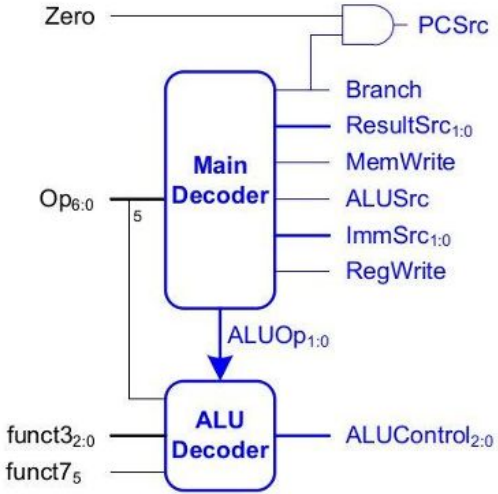
ImmSrc[1:0]	ImmExt	Instruction Type
00	{{20{instr[31]}}, instr[31:20]}	I-Type
01	{{20{instr[31]}}, instr[31:25], instr[11:7]}	S-Type
10	{{19{instr[31]}}, instr[31], instr[7], instr[30:25], instr[11:8], 1'b0}	B-Type

Después:

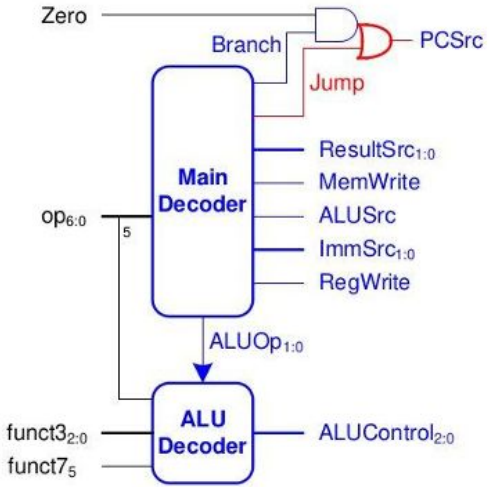
ImmSrc[1:0]	ImmExt	Instruction Type
00	{{20{instr[31]}}, instr[31:20]}	I-Type
01	{{20{instr[31]}}, instr[31:25], instr[11:7]}	S-Type
10	{{19{instr[31]}}, instr[31], instr[7], instr[30:25], instr[11:8], 1'b0}	B-Type
11	{{12{instr[31]}}, instr[19:12], instr[20], instr[30:21], 1'b0}	J-Type

En la unidad de control se modificó la forma de calcular PCSrc

Antes:



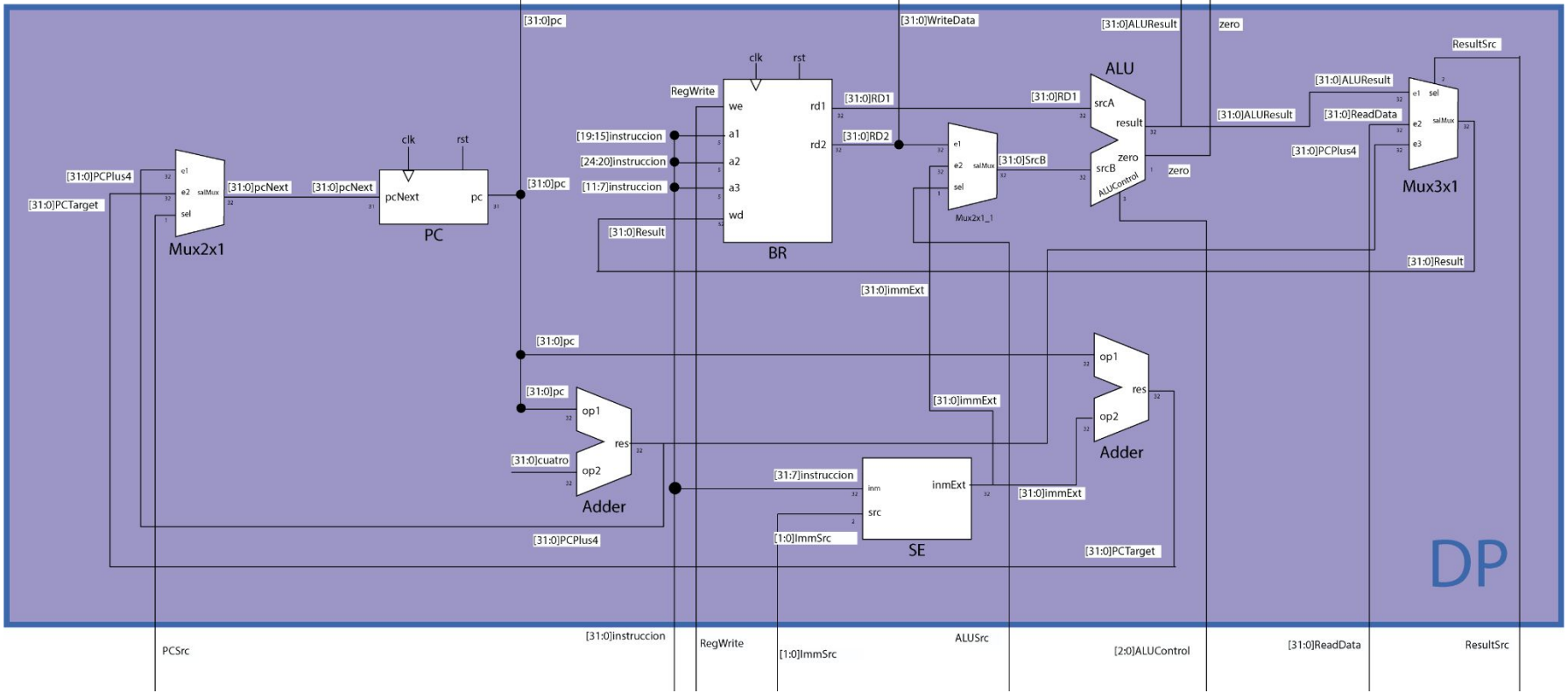
Después:



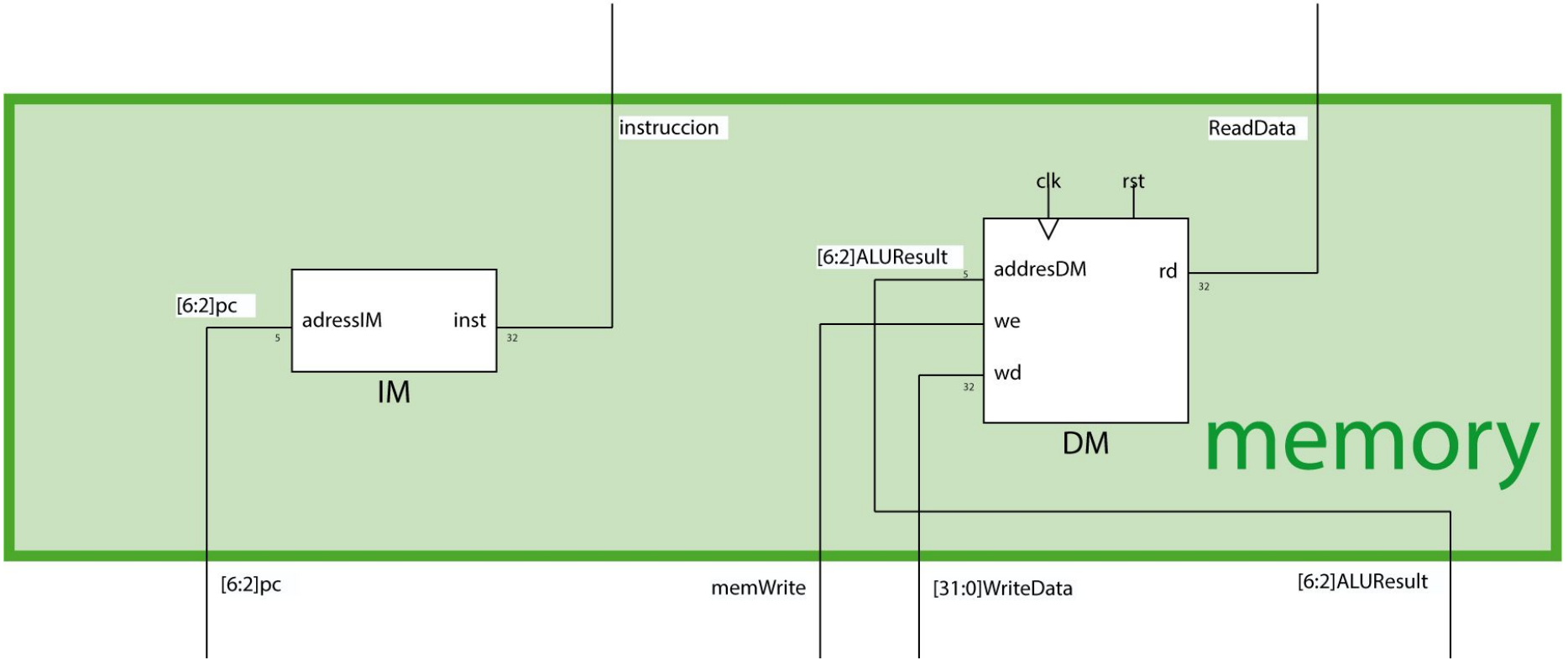
Etapa 4- Diseño del rv32i

A medida que se desarrollaron las distintas guías prácticas y se fueron siguiendo las presentaciones propuestas, se fue modelando un diseño de procesador propio con todas las conexiones.

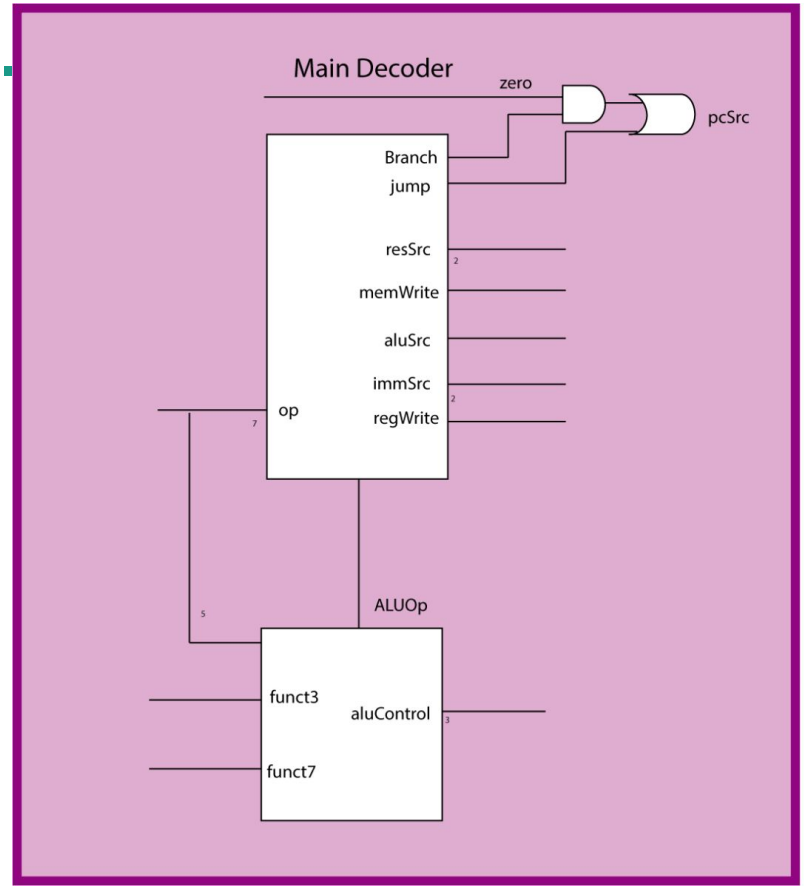
Datapath (Camino de datos)

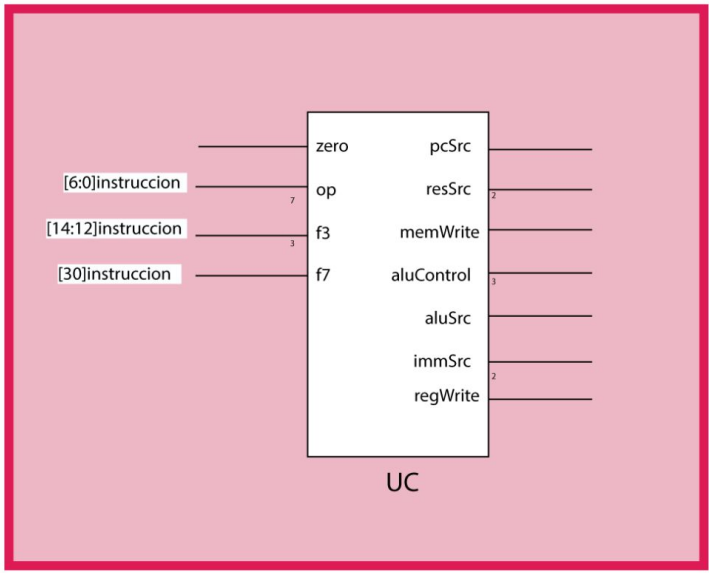


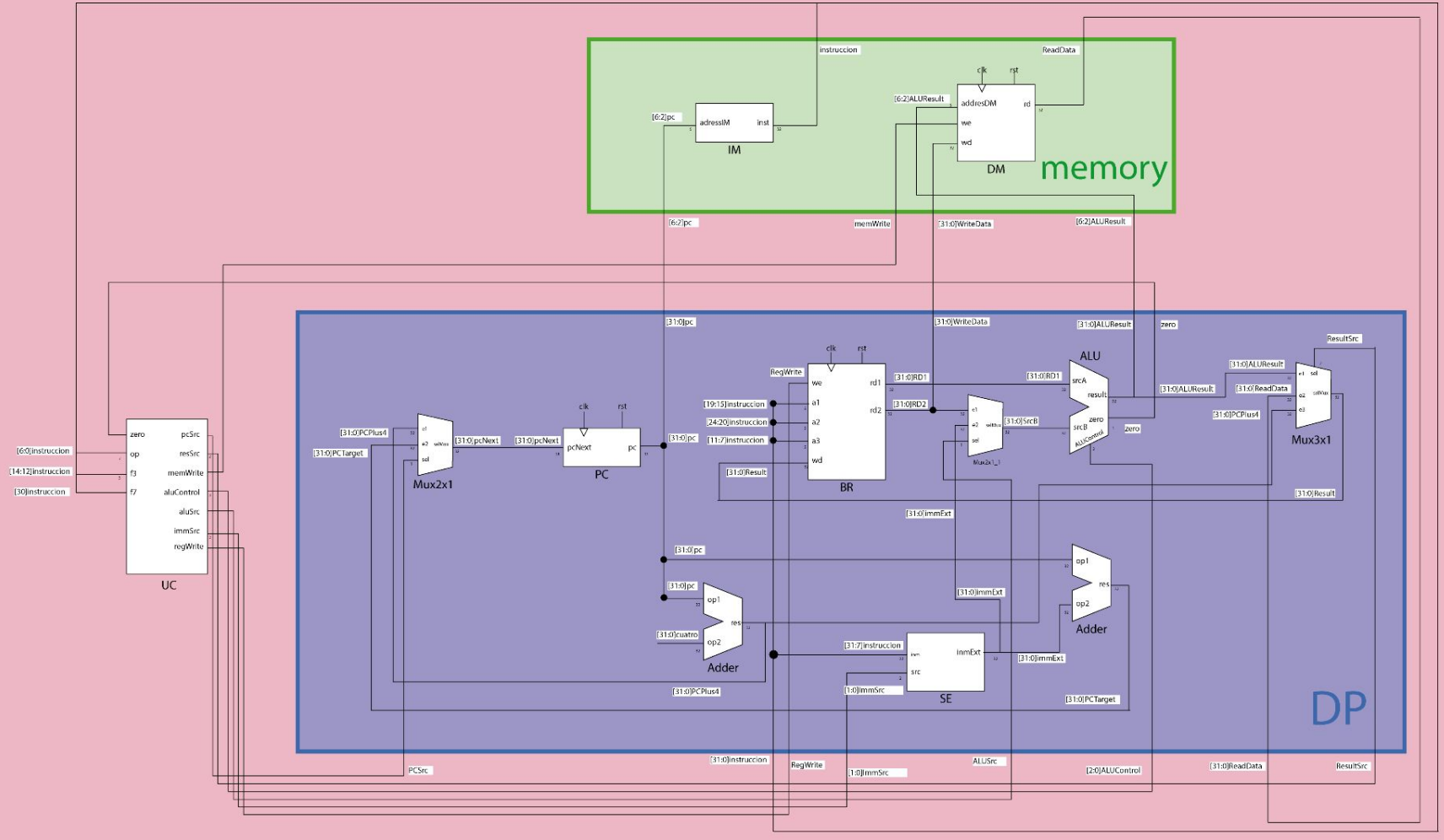
Memory


instruccion
ReadData
clk
rst
[6:2]ALUResult
addresDM
rd
we
wd
DM
[6:2]pc
adressIM
inst
IM
memory
[6:2]pc
memWrite
[31:0]WriteData
[6:2]ALUResult

Unidad de control



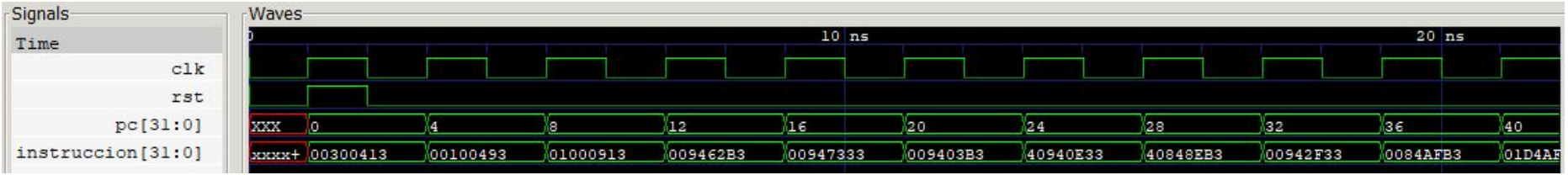



memory
IM
DM
adressIM
inst
addresDM
rd
we
wd
clk
rst
instruccion
ReadData
[6:2]pc
[6:2]ALUResult
[6:2]pc
memWrite
[31:0]WriteData
[6:2]ALUResult
DP
UC
zero
op
f3
f7
pcSrc
resSrc
memWrite
aluControl
aluSrc
immSrc
regWrite
[6:0]instruccion
[14:12]instruccion
[30]instruccion
Mux2x1
[31:0]PCPlus4
[31:0]PCTarget
[31:0]pcNext
PC
pcNext
pc
[31:0]pc
BR
RegWrite
we
a1
a2
a3
wd
rd1
rd2
[19:15]instruccion
[24:20]instruccion
[11:7]instruccion
[31:0]Result
[31:0]RD1
[31:0]RD2
ALU
srcA
srcB
result
zero
ALUControl
[31:0]SrcB
[31:0]ALUResult
[31:0]ReadData
[31:0]PCPlus4
Mux3x1
ResultSrc
[31:0]Result
[31:0]immExt
[31:0]pc
[31:0]cuatro
Adder
op1
op2
res
[31:7]instruccion
SE
inm
src
InmExt
[31:0]immExt
[1:0]ImmSrc
[31:0]PCTarget
[31:0]PCPlus4
PCSrc
[31:0]instruccion
RegWrite
[1:0]ImmSrc
ALUSrc
[2:0]ALUControl
[31:0]ReadData
ResultSrc

rv32i

Etapa 5 - Prueba del procesador

Primero vemos que el clk, el rst y las instrucciones se van ejecutando y cambiando correctamente según el archivo .hex generado



El pc representa el contador de programa, este avanza de 4 en 4 o se determina por las instrucciones de salto.

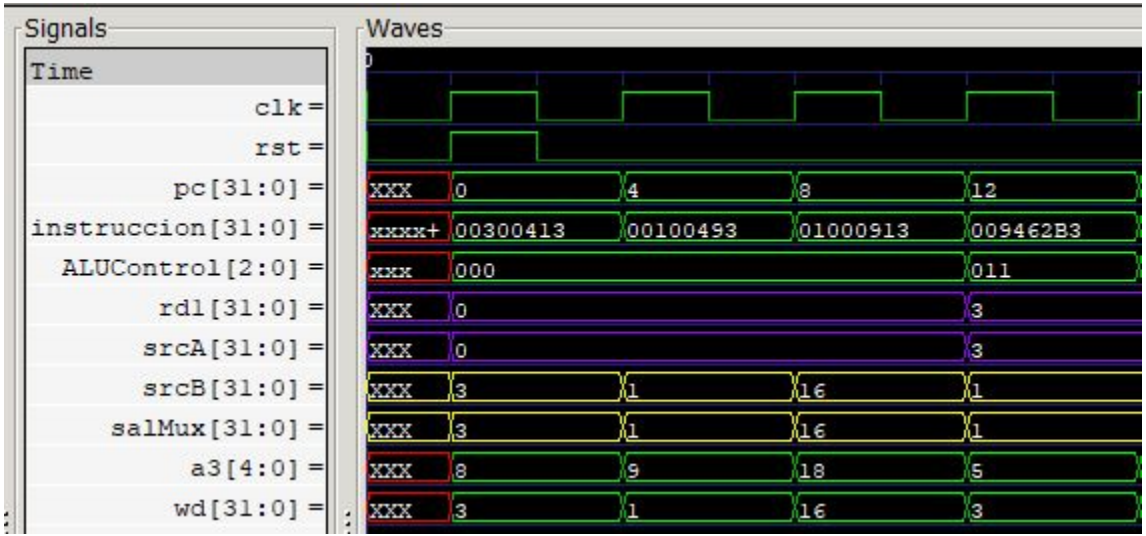
Valores de los registros

Para comprender el correcto funcionamiento del risc-v es necesario comprender los valores por los cuales están representados los registros, en el cual se observa por ejemplo, t0=5 t1=6 t2=7...

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

```
# Inicializo tres variables
    addi s0, zero, 3          # a = 3
    addi s1, zero, 1          # b = 1
    addi s2, zero, 16         # cte = 16
```

Equivalencias entre nombres de registros:
s0 = 8
s1 = 9
s2 = 18

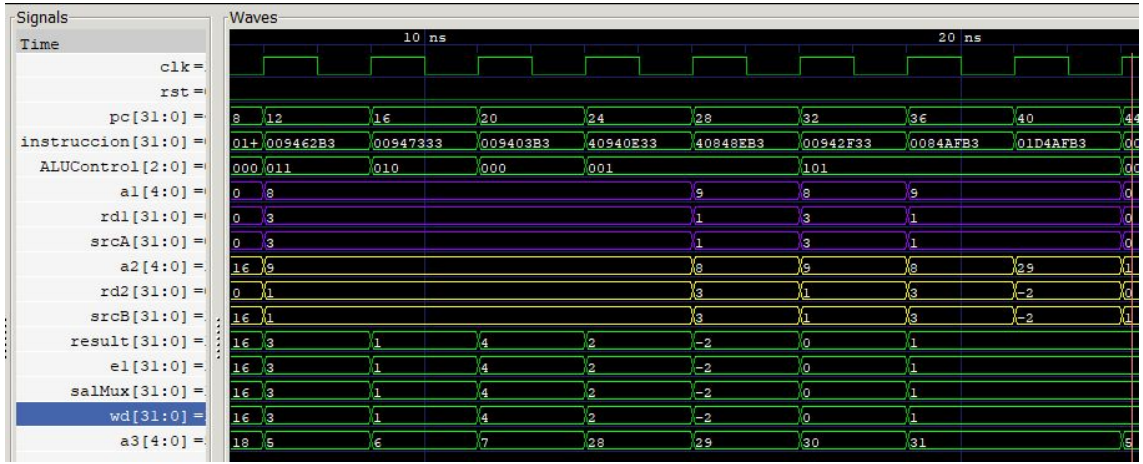


- **ALUControl** (aluDeco) =**ALUControl**(UC) = 000 indica la suma
- **srcA** (ALU) = Valor que toma para la suma (0 para los tres addi)
- **srcB** (ALU) = Valor que toma para la suma
- **a3** (BR) =Dirección de registro a escribir en este caso los registros 8,9 y 16

```
# Operaciones lógicas, aritméticas y slt
    or   t0, s0, s1          # c = 3
    and  t1, s0, s1          # d = 1
    add  t2, s0, s1          # e = 4
    sub  t3, s0, s1          # f = 2
    sub  t4, s1, s0          # g = 0xfffffffe = -2
    slt  t5, s0, s1          # h = 0
    slt  t6, s1, s0          # i = 1
    slt  t6, s1, t4          # j = 0
```

Equivalencias entre nombres de registros:

s0 = 8	t2 = 7
s1 = 9	t3 = 28
t0 = 5	t4 = 29
t1 = 6	t5 = 30
	t6 = 31



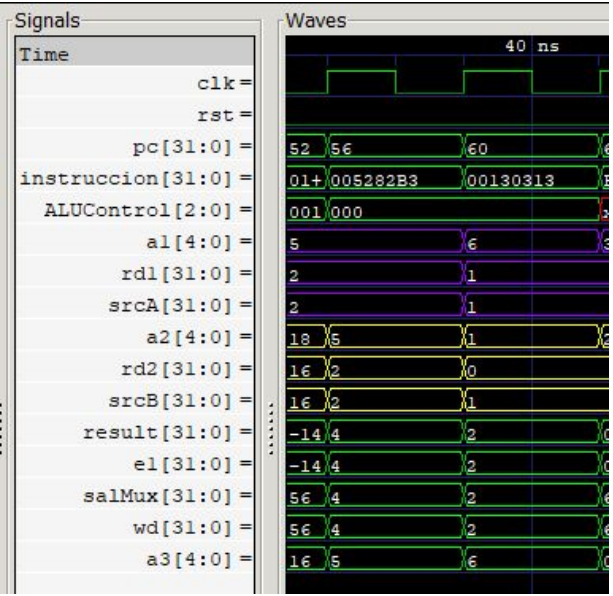
- **ALUControl** (aluDeco) = **ALUControl**(UC)= OR: 011 AND: 010 ADD: 000 SUB:001 SLT: 101
- **srcA** (ALU) = Valor que toma para las operaciones
- **srcB** (ALU) = Valor que toma para las operaciones
- **a1** = Dirección de registro a leer en **rd1**
- **a2** = Dirección de registro a leer en **rd2**
- **a3** = Dirección de registro donde se escribe **wd**
- **result** = **e1** = **salMux** = **wd** = resultado de la operación

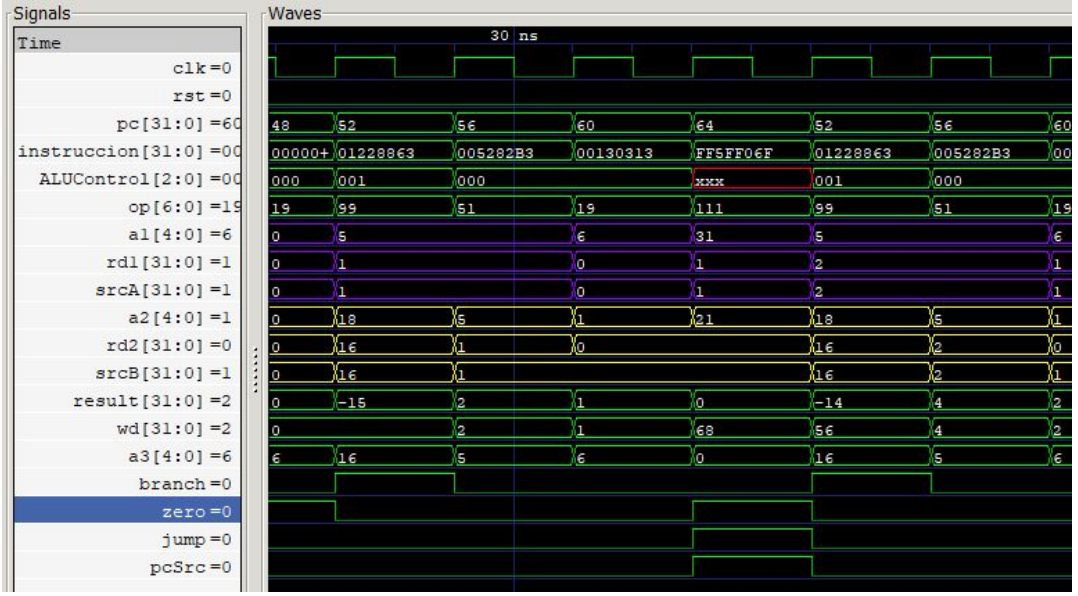
```
# Un ciclo while usando beq y j
# Inicializo 3 variables
# s2 = 0x10 cte para comparar (16)
    addi t0, zero, 1          # var = 1, variable de trabajo
    addi t1, zero, 0          # cuenta = 0, un contador

while:
    beq  t0, s2, sal1         # Si var == cte, sale del while
    add  t0, t0, t0           # var = var + var
    addi t1, t1, 1            # cuenta = cuenta + 1
    j    while
```

Equivalencias entre nombres de registros:

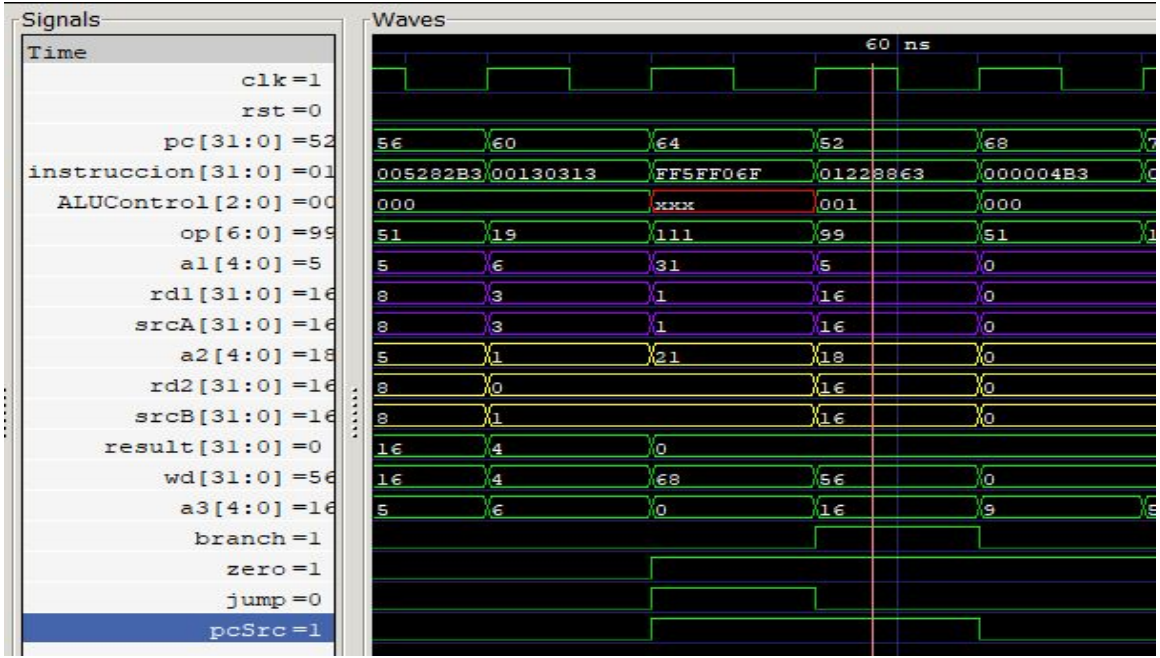
t0 = 5
t1 = 6
s2=18

- **ALUControl** (aluDeco) = **ALUControl**(UC)= OR: 011 AND: 010 ADD: 000 SUB:001 SLT: 101
- **srcA** (ALU) = Valor que toma para las operaciones
- **srcB** (ALU) = Valor que toma para las operaciones
- **a1** = Dirección de registro a leer en **rd1**
- **a2** = Dirección de registro a leer en **rd2**
- **a3** = Dirección de registro donde se escribe **wd**
- **result** = **e1** = **salMux** = **wd** = resultado de la operación
- **op** = indica que tipo de instrucción se va a realizar (destacamos la 99 branch y 111 jump)
- **branch** = bandera que se activa cuando se ejecuta una operaciòn beq
- **zero** = bandera que determina cuando el resultado de la ALU es 0
- **jump** =bandera que se activa cuando se ejecuta una operación de salto
- **pcSrc** = selector de siguiente valor de PC

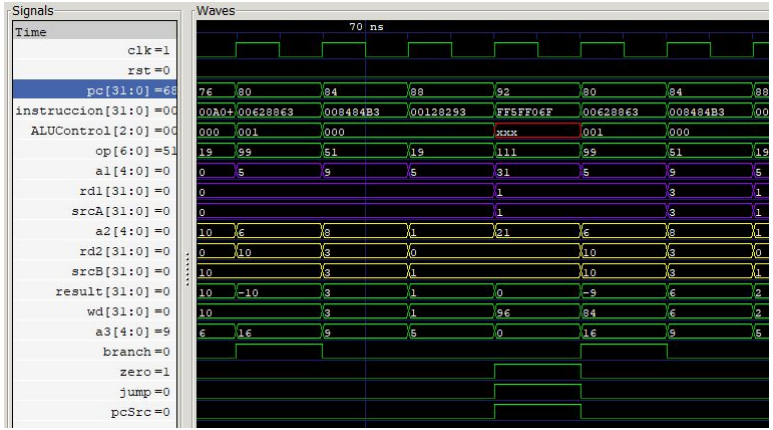


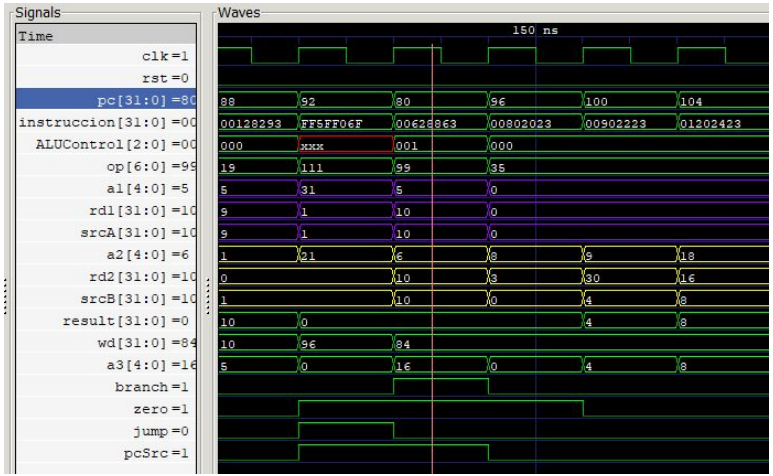


Captura que muestra el final del while



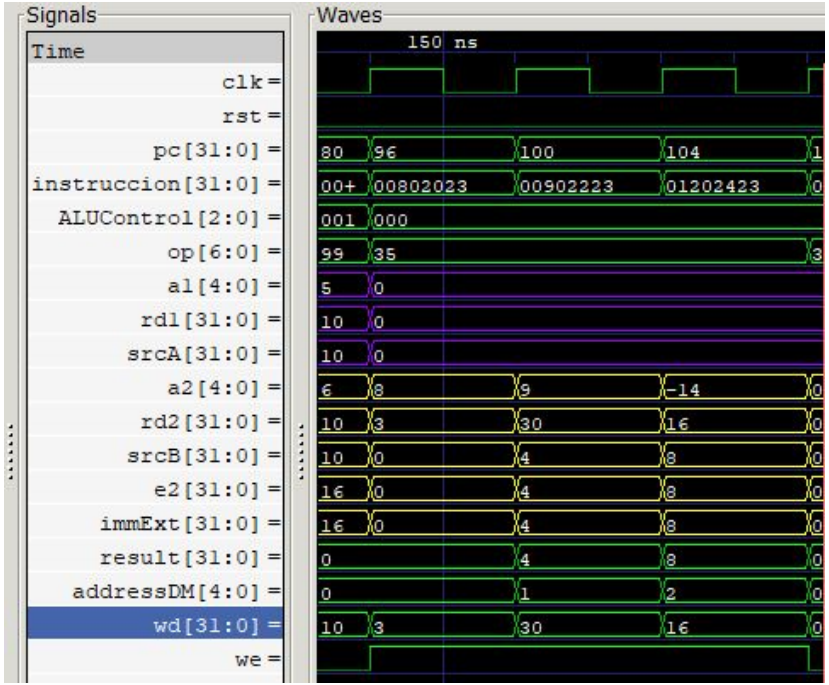
```
sal1:
# Al finalizar el ciclo, var debe quedar en 0x10 y cuenta en 4

# Ciclo for
# Inicializo 3 variables
# $t0 = i, $s1 = var

    add  s1, zero, zero      # var = 0, $s0
    #add $s0, $0, 3          # cte = 3, $s1
    addi t0, zero, 0         # indice = 0, $t0
    addi t1, zero, 10        # veces = 10, $t1

for:
    beq  t0, t1, sal2        # Si indice == veces, branch a sal2
    add  s1, s1, s0          # var = var + cte
    addi t0, t0, 1           # incremento indice
    j    for
```





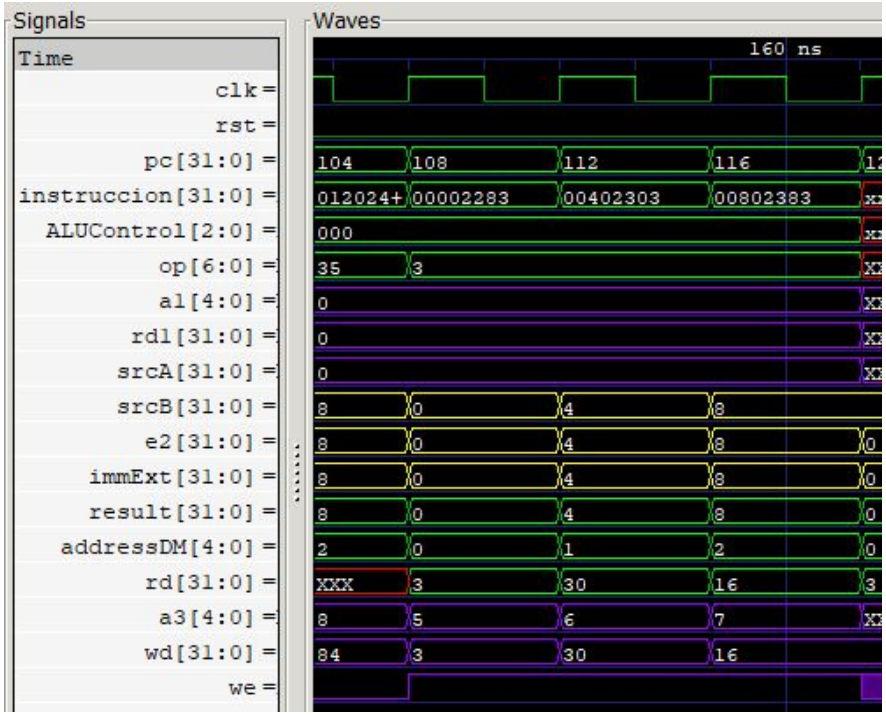
En las imágenes se puede ver el funcionamiento del bucle for:

```
sal2:
# Al finalizar el ciclo, var debe quedar en 30 (0x1e) y indice en 10 (0xa)

# Almacenamiento (escritura) con sw
    sw    s0, 0(zero)          # Guarda s0 en registro 0
    sw    s1, 4(zero)          # Guarda s1 en registro 4
    sw    s2, 8(zero)          # Guarda s2 en registro 8
```



- **ALUControl** (aluDeco) = **ALUControl**(UC)= 000 (suma para lw sw)
- **op** = indica que tipo de instrucción se va a realizar (35 store)
- **srcA** (ALU) = **rd1** = Valor que toma para las operaciones
- **srcB** (ALU) = **e2** = **immExt** = Valor que toma para las operaciones
- **a1** (BR) = Dirección de registro a leer en **rd1**
- **a2** (BR) = Dirección de registro a leer en **rd2**
- **e2** = **immExt** = valor a sumar a la dirección del registro
- **result** (ALU) = resultado de la suma entre registro e inmediato
- **addressDM**(DM) = dirección a guardar el dato en DM
- **wd** = indica que dato se escribirá en DM
- **we**(DM) = **memWrite**(UC) = bandera que se activa para habilitar escritura

```
# Carga (lectura) con lw
    lw    t0, 0(zero)          # Carga el valor de registro 0 en t0
    lw    t1, 4(zero)          # Carga el valor de registro 4 en t1
    lw    t2, 8(zero)          # Carga el valor de registro 8 en t2
```



- **ALUControl** (aluDeco) = **ALUControl**(UC)= 000 (suma para lw sw)
- **op** = indica que tipo de instrucción se va a realizar (3 load word)
- **srcA** (ALU) = **rd1** = Valor que toma para las operaciones
- **srcB** (ALU) = **e2** = **immExt** = Valor que toma para las operaciones
- **a1** (BR) = Dirección de registro a leer en **rd1**
- **e2** = **immExt** = valor a sumar a la dirección del registro
- **result** (ALU) = resultado de la suma entre registro e inmediato
- **addressDM**(DM) = dirección a leer el dato de DM
- **wd** (BR) = indica que dato se escribirá en BR
- **a3** (BR) = indica la dirección del BR donde se escribirá el dato
- **we**(BR) = **regWrite**(UC) = bandera que se activa para habilitar escritura
- **rd** (DM) = valor leído de DM