ESCOLA TÈCNICA SUPERIOR
D'ENGINYERIA
Universitat Rovira i Virgili

Práctica 2

Alumnos: Alain Martínez
Antoni Llop
Carrera: Ingeniería Informática
Profesor: Carles Aliagas
Fecha: 2025

FASE 1

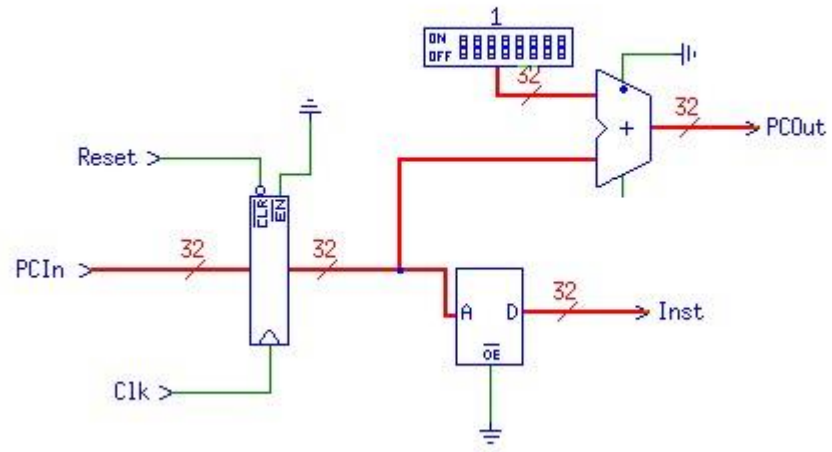
Tarea 1.1

Realizad la parte del circuito del procesador que se encarga del fetch de las instrucciones y que se muestra en la siguiente figura.

Diseño

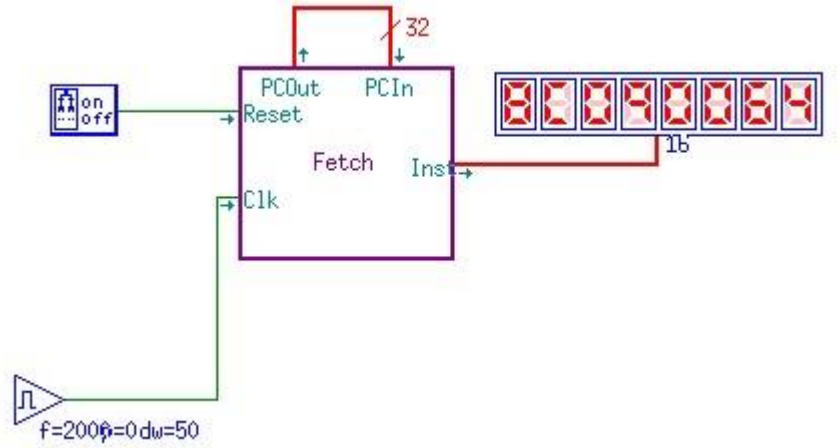
El fetch es el proceso de obtener, actualizar y devolver una instrucción de la memoria de instrucciones utilizando el Program Counter (PC). Se conecta el PC a la memoria de instrucciones (ROM) para obtener la instrucción deseada. Después de obtenerla, se incrementa el PC en un byte para apuntar a la siguiente instrucción

Implementación



Juego de pruebas

Para comprobar que nos funciona hemos puesto el archivo mult.mem y hemos ido viendo las instrucciones que devolvía.

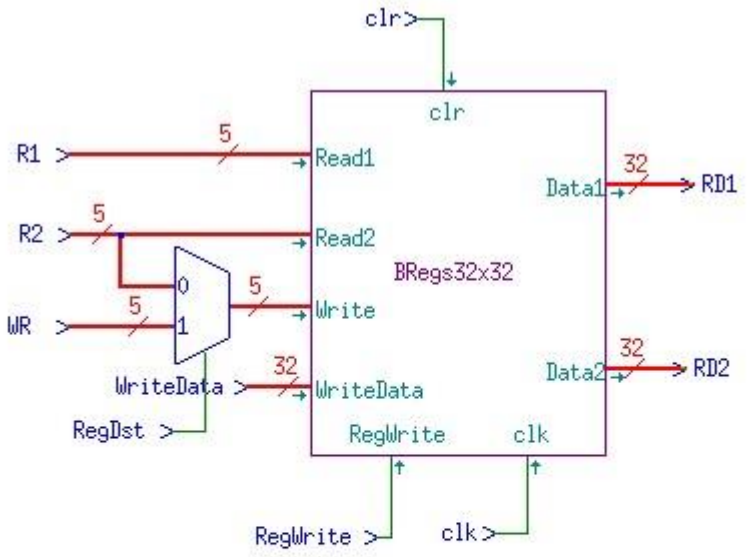


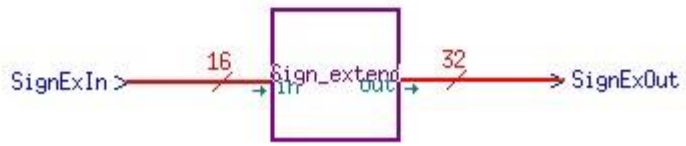
Tarea 1.2

Utilizando el banco de registros que se proporciona a través del Campus Virtual, probad el circuito que implementa la etapa de lectura de los registros y que se muestra en la siguiente figura. Realizad varios ciclos de escritura y lectura para comprobar el correcto funcionamiento del mismo. **Diseño**

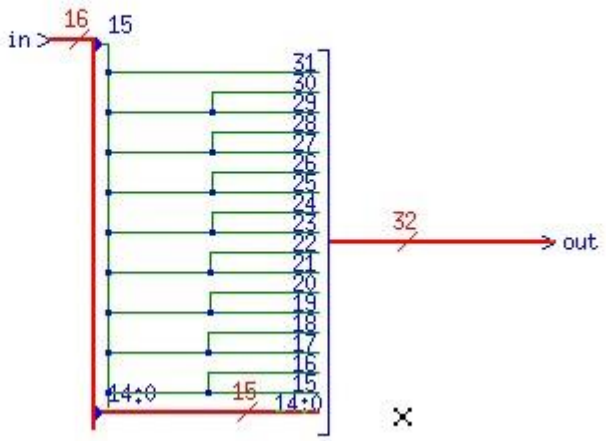
En esta tarea se ha implementado el Read, el cual se encarga de recibir los registros y realizar operaciones de lectura o escritura sobre ellos según se requiera. Además, incluye internamente un circuito de extensión de signo, cuya función es convertir un valor de 16 bits a 32 bits para que pueda ser utilizado en operaciones posteriores.

Implementación





El sign extend:

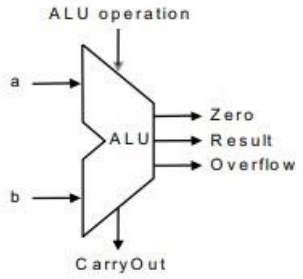


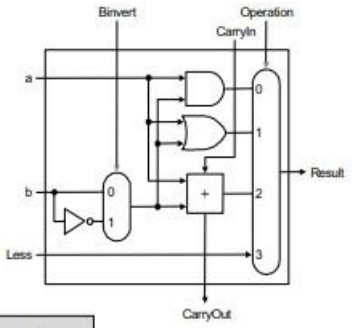
Tarea 1.3

Utilizando el banco de registros que se proporciona a través del Campus Virtual, realizad la parte del circuito del procesador que se encarga de la ejecución de las instrucciones de aritmético-lógicas (Formato R) y que se muestra en la siguiente figura

Diseño

- Operaciones básicas (solo usaremos 5 de las 8 posibles)



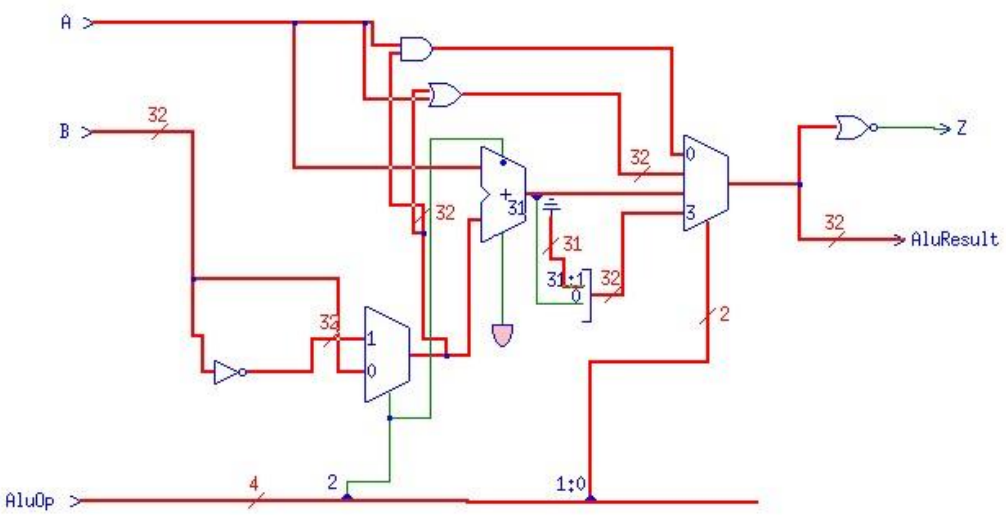


a.

Entradas de control (ALUctr)	FUNCIÓN
000	AND
001	OR
010	ADD
110	SUB
111	SLT

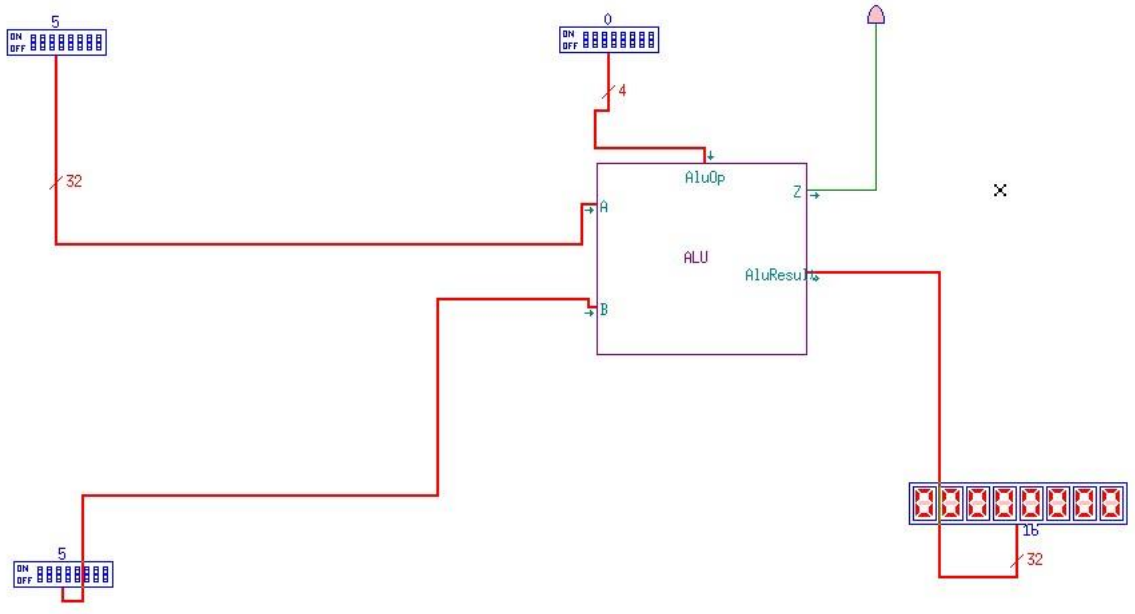
Para realizar nuestra ALU nos basaremos en esta imagen.

Implementación



Juego de pruebas

Para realizar un pequeño juego pruebas hemos hecho este circuito provisional:



ALU operation	ALU Function	Zero
0000	A **AND** B	✓
0001	A **OR** B	✓
0010	A + B	✓
0110	A - B	✓
0111	si (A < B) ALUresult = 1 sino ALUresult = 0	✓

ENTRADAS			RESULTADOS	
A (hex)	**B (hex)**	**ALU OP (bin)**	**ALU FUNCT**	**Zero**
0	0	0000	0	1
5	5	0000	5	0
0	0	0001	0	1
5	5	0001	5	0
2	3	0010	5	0
5	5	0010	A	0
0	0	0110	0	1
3	1	0110	2	0
0	0	0111	0	1
5	5	0111	0	1
1	3	0111	1	0

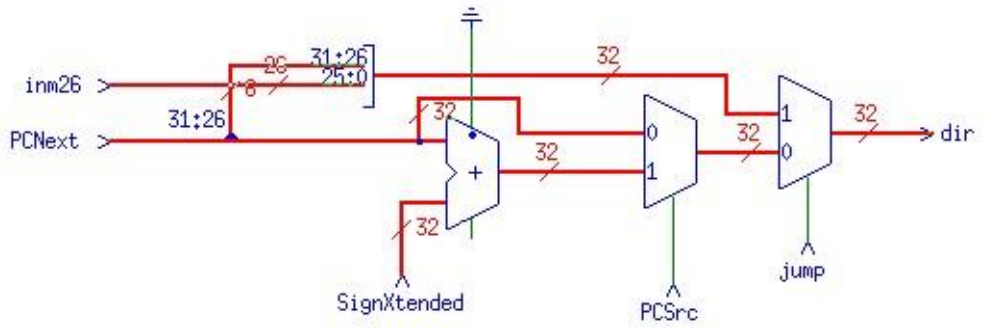
Tarea 1.4

Realizad la parte del circuito del procesador que se encarga de la ejecución de las instrucciones básicas de formato I y J (beq y j) y que se muestra en la siguiente figura.

Diseño

El circuito implementado en esta tarea es el Jump, la parte que se encarga de cambiar el contador del programa (PC) para saltar a otra dirección de memoria

Implementación



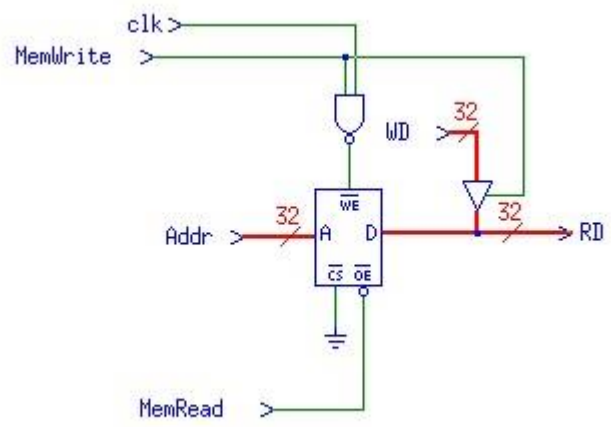
Tarea 1.5

A partir del diseño de la tarea anterior, añadir la parte del circuito del procesador que se encarga de la ejecución de las instrucciones de acceso a memoria, lw y sw (Formato I) y que se muestra en la siguiente figura

Diseño

La escritura en memoria se realiza únicamente en el flanco de subida del reloj, es decir, cuando la señal de reloj cambia de 0 a 1, y siempre que la señal de control MemWrite esté activada. Se utiliza una compuerta AND negada porque la señal WE (Write Enable) se activa con un valor bajo (0).
Para controlar si los datos provenientes de Write Data (los datos a escribir en memoria) llegan o no a la memoria, se emplea un Tri-State Buffer.

Implementación

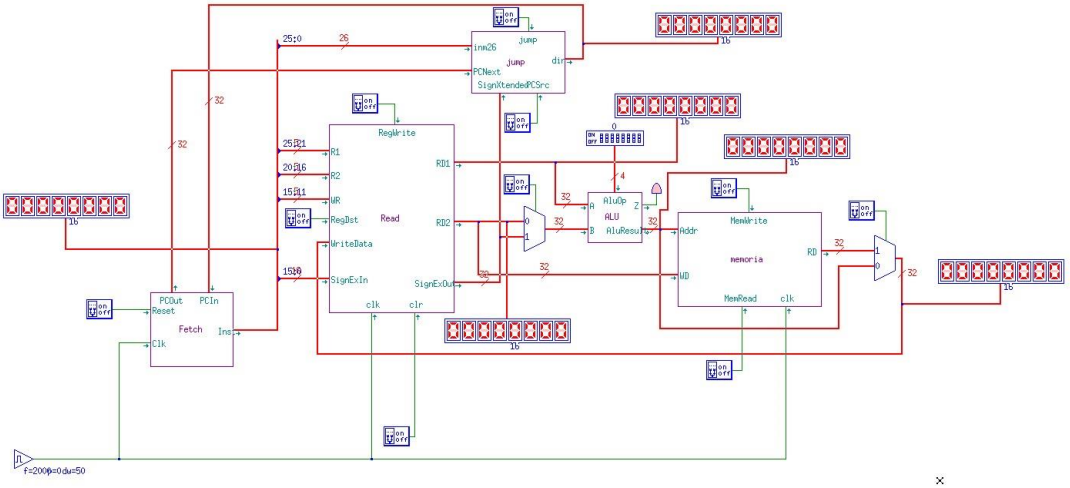


Tarea 1.6

Unid todos los componentes implementados hasta el momento para obtener la Unidad de Proceso completa del procesador MIPS monociclo. En la siguiente figura se muestra el esquema simplificado. **Diseño**

Básicamente hemos unido todos los componentes realizados anteriormente.

Implementación



Tarea 1.7

Realizad la parte del circuito que se encarga de la Unidad de Control del procesador MIPS y que se muestra junto a la Unidad de Proceso en la siguiente figura. **Diseño**

Para realizar nuestra Unidad de Control nos basaremos en la tabla que nos dan en el enunciado.

Op	RegDst	Branch	Jump	MemRead	MemToReg	ALUCtrl	ALUOp	MemWrite	ALUSrc	RegWrite
add	1	0	0	0	0	0010	10	0	0	1
sub	1	0	0	0	0	0110	10	0	0	1
and	1	0	0	0	0	0000	10	0	0	1
or	1	0	0	0	0	0001	10	0	0	1
slt	1	0	0	0	0	0111	10	0	0	1
lw	0	0	0	1	1	0010	00	0	1	1
sw	X	0	0	0	X	0010	00	1	1	0
beq	X	1	0	0	X	0110	01	0	0	0
j	X	0	1	0	X	XXXX	XX	0	0	0

Para llevar a cabo la implementación de este circuito, lo hemos dividido en dos secciones. La primera se encarga de identificar si la instrucción corresponde a una operación aritmética, lw, sw, beq o jump, utilizando los 6 bits más significativos de la instrucción. En caso de que se trate de una operación aritmética, se analizan los 4 bits menos significativos

para determinar qué valores deben enviarse a la ALU y así indicar qué operación debe realizar.

En cada caso, las entradas que deberían ser 0 se niegan, y luego todas las condiciones se combinan mediante una compuerta AND. De este modo, la salida será 1 únicamente cuando la instrucción coincida exactamente con la que se desea detectar. Esta salida se conecta directamente a las señales de control correspondientes, o bien a una compuerta OR si dicha señal puede ser activada por más de una instrucción.

Además, hemos observado que todas las instrucciones aritméticas generan las mismas señales de control, salvo la señal ALUCtrl. Por lo tanto, se pueden tratar de forma unificada en cuanto a la lógica del circuito.

De la tabla de arriba deducimos esto:

op	funct	ALUop	ALUctr
100011 (lw)	XXXXXX	00	010
101011 (sw)		00	010
000100 (beq)		01	110
000000 (tipo-R)	100000 (add)	10	010
	100010 (sub)	10	110
	100100 (and)	10	000
	100101 (or)	10	001
	101010 (slt)	10	111

Y jump que sería = 000010.

Ahora desglosamos la parte de las operaciones aritméticas de la tabla que tenemos aquí para que se vea más claro.

Instrucción	Funct	AluCtr
add	10 0000	0010
sub	10 0010	0110
and	10 0100	0000
or	10 0101	0001
slt	10 1010	0111

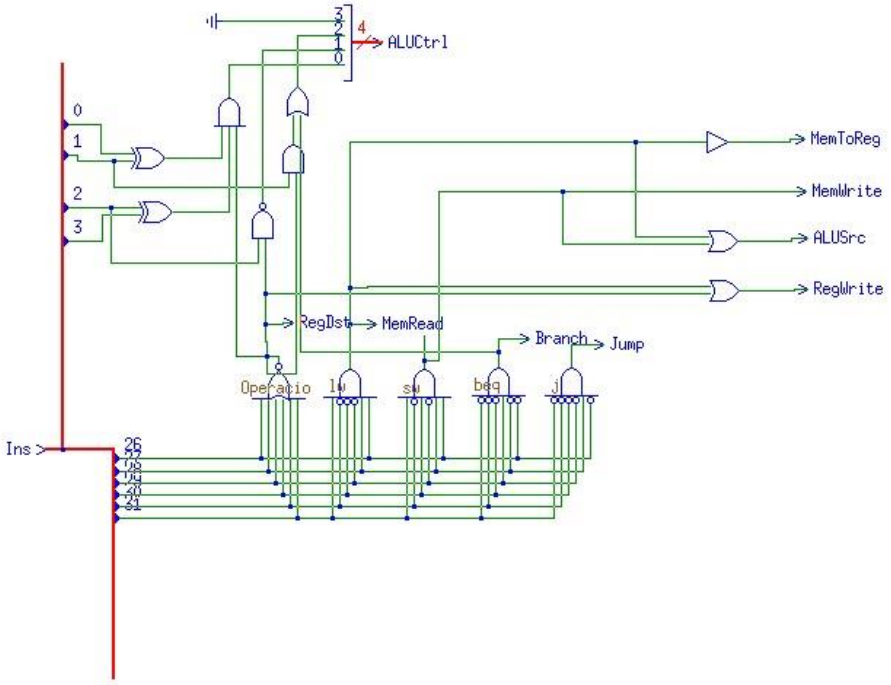
Para las señales externas:

Op5-0 -> (bits 31-26)	00 0000 0_D	10 0011 35_D	10 1011 43_D	00 0100 4_D
	R-Format	lw	sw	beq
RegDst	1	0	x	x
ALUSrc	0	1	1	0
MemToReg	0	1	x	x
RegWrite	1	1	0	0
MemRead	0	1	0	0
MemWrite	0	0	1	0
Branch	0	0	0	1
ALUOp1	1	0	0	0
ALUOp0	0	0	0	1

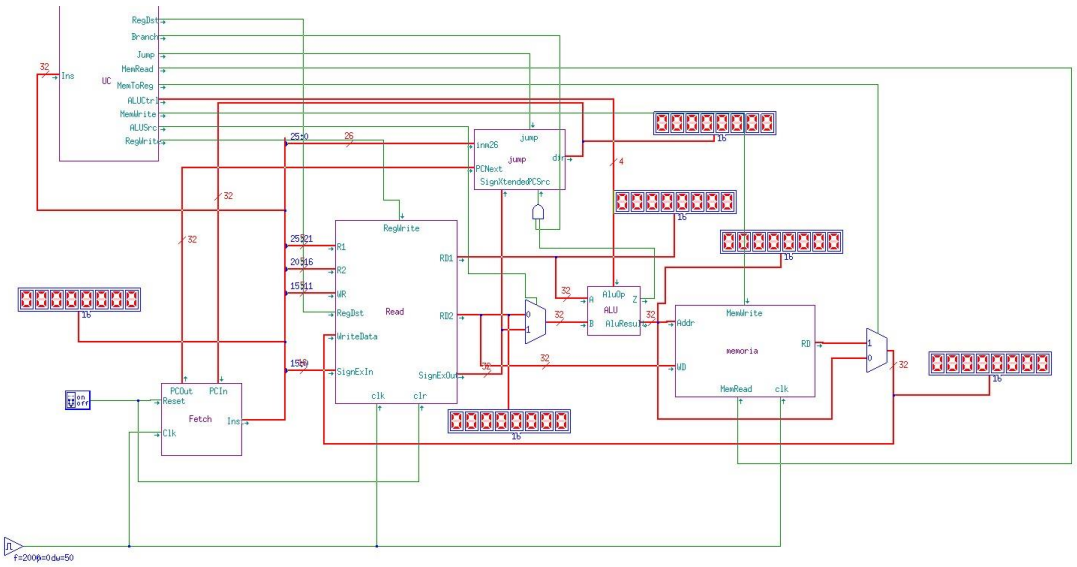
Y en la tabla esta falta el jump que sería 1 en J.

Implementación

Implementación de la UC:

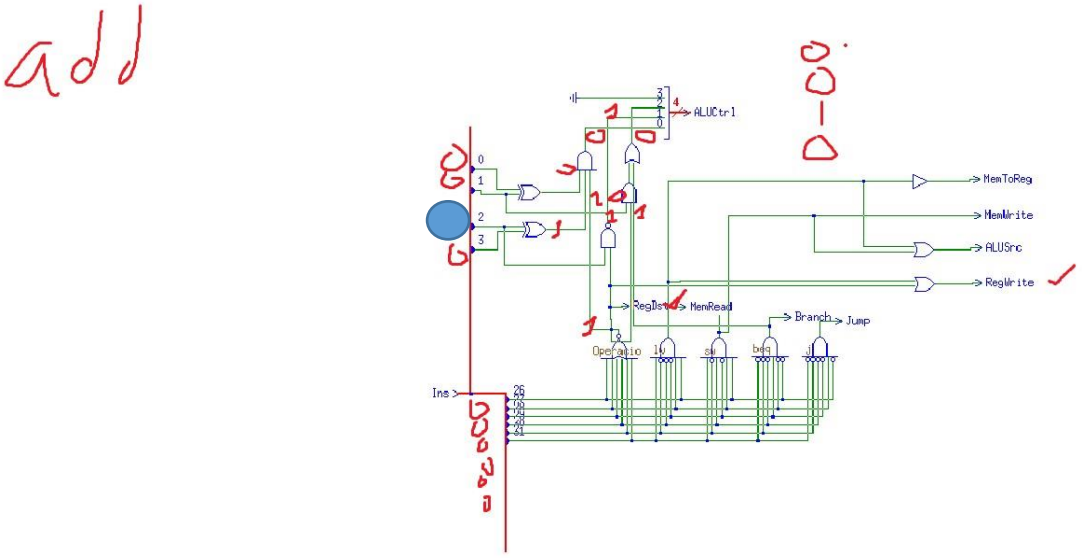


Implementación de todo el procesador:



Juego de pruebas

Aquí probaremos una instrucción de la UC.



Podemos observar que las señales externas que tenemos son el RegWrite y el RegDst. Y el ALUCtrl nos da 0010 qué es el que hemos deducido anteriormente.

Tarea 1.8

Realizad una prueba conjunta de funcionamiento de las instrucciones del repertorio ISA básico que se ha asumido en esta práctica, a través de la ejecución del programa de prueba que se proporciona.

Juego de pruebas

Ejecutamos el programa de prueba mult.mem y vemos que se detiene correctamente en el bucle en la dirección 0x0B y nos da el resultado correcto:

#main:

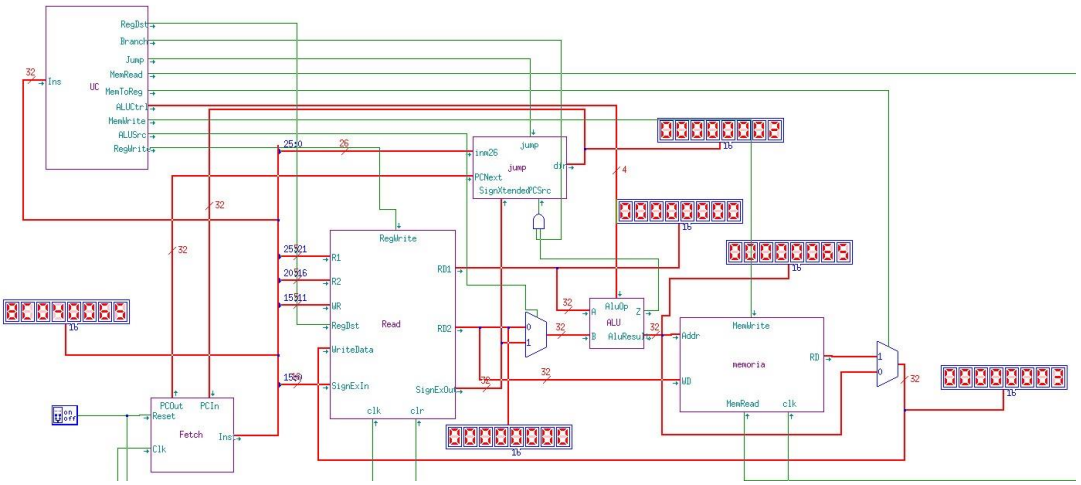
00/ 8c090064 **8c040065** 8c050066 00008024 00008824

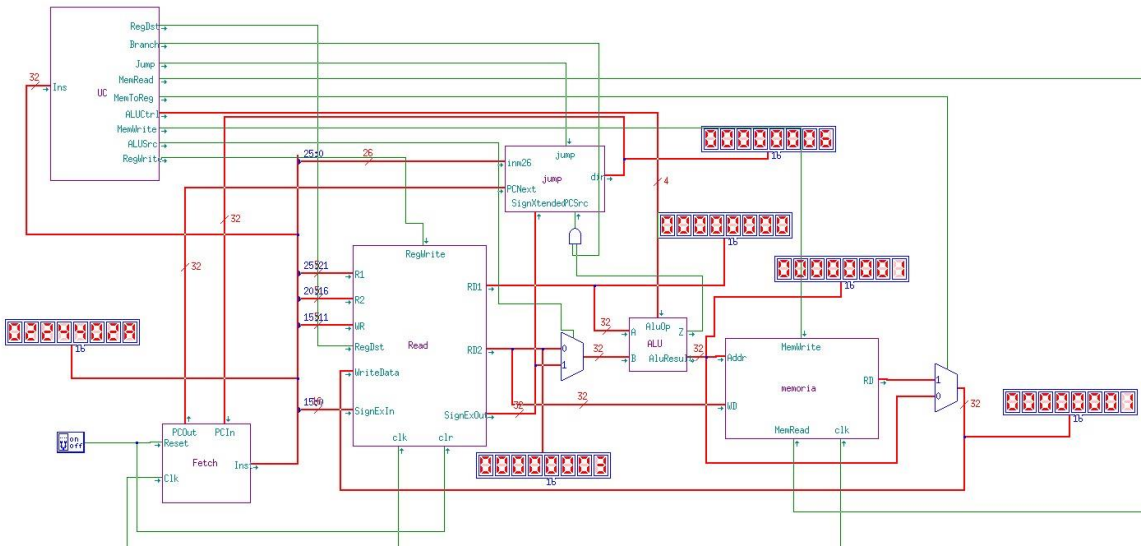
per:

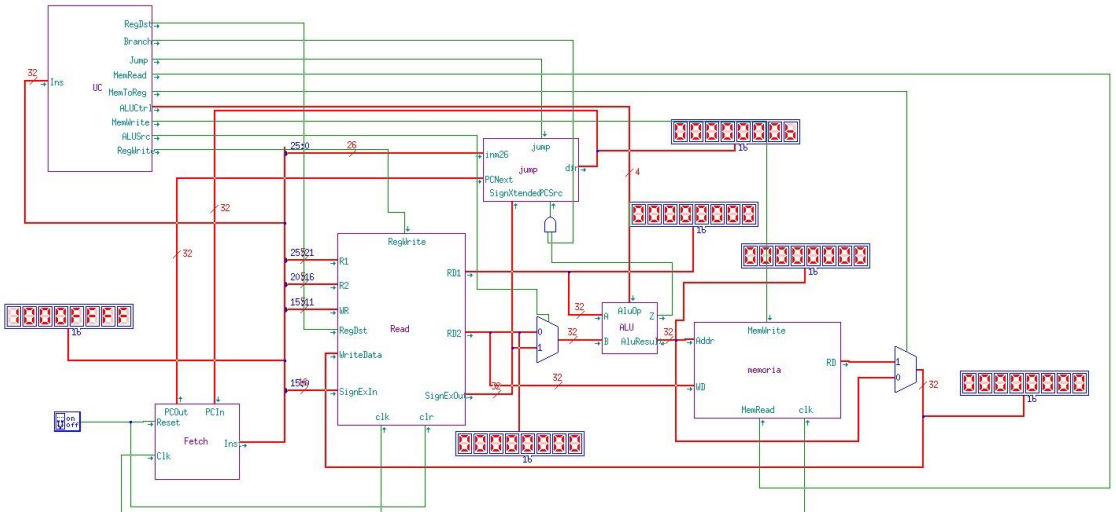
05/ **0224402a** 11000003 02058020 02298820 1000fffb
fiper: 0a/
ac100067
fi:
0b/ **1000ffff**

DATA IN MEMORY
00000064/ 00000001 00000003 00000005 00000000

Adjuntamos algunas fotos para que se vea que funciona correctamente (hemos hecho captura de las que están en negrita):







Tarea 1.9

Realizad una prueba conjunta de funcionamiento de las instrucciones del repertorio ISA básico que se ha asumido en esta práctica, a través de la ejecución de un programa de prueba propio.

Diseño

El programa que nos dan en esta práctica sólo utiliza las instrucciones lw, sw, and, slt y beq. Así que nuestra idea para el programa de prueba es hacer uno que use las instrucciones que faltan: sub, or, j.

Este es nuestro programa:

```
.text
.globl main

main:
    lw $a0, 100($zero)   # Cargar A en $a0
lw $a1, 101($zero)   # Cargar B en $a1

    or  $s0, $a0, $a1    # $s0 = $a0 | $a1
sub $s1, $a0, $a1    # $s1 = $a0 - $a1

    sw $s0, 102($zero)   # Guardar resultado OR en mem[102]
sw $s1, 103($zero)   # Guardar resultado SUB en mem[103]

loop:
    j loop              # Bucle infinito

.data        a:
.word 240 b:
.word  15
```

Juego de pruebas

Podemos comprobar que acaba en el bucle infinito.
main:
00/ 8C040064 8C050065 00858025 00858822 AC100066 AC110067
loop:
06/ 08000006