

Organización de las Computadoras

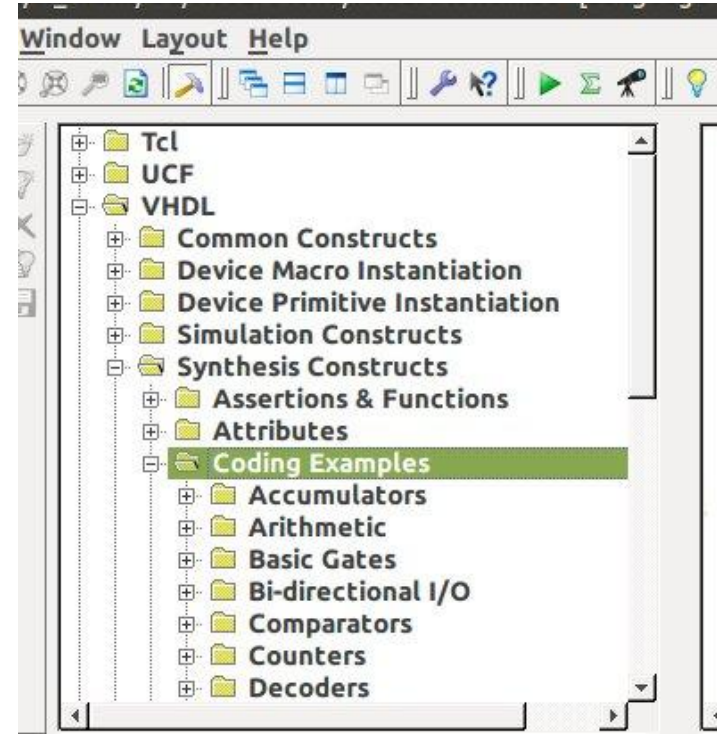
2021

Temario

- 1- Diseño de las arquitecturas de los componentes
- 2- Diseño de la Unidad Aritmético Lógico (ALU)
- 3- Diseño de la Unidad de Control
- 4- Nota sobre el mapa memoria
- 5- Tarea (toda la guia..)

1- Diseño de las arquitecturas de los componentes

Si bien en la guía de práctica les doy algunos códigos de ejemplo, pueden encontrar más en los Language Templates del ISE Design de Xilinx.



2- Unidad Aritmético Lógico

Si recordamos las operaciones

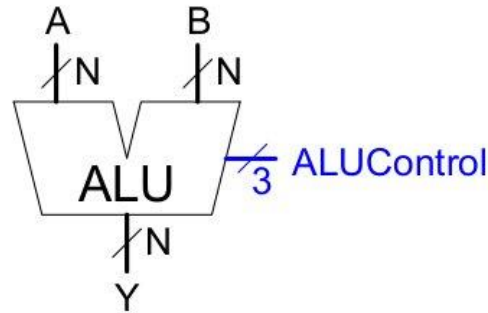
encontramos:	TR	TI	Ap.
Aritméticas	Suma	*	*
	Resta		*
Lógicas	And	*	*
	Or	*	*
	Xor	*	
Comparar	SLT	*	*
	SLTU		

Desplazar shift left logic
 shift right logic
 shift right arithmetic

2- Unidad Aritmético Lógico

Si recordamos las operaciones encontramos:

	TR	TI	Ap.
Aritméticas	Suma	*	*
	Resta		*
Lógicas	And	*	*
	Or	*	*
	Xor	*	
Comparar	SLT	*	*
	SLTU		



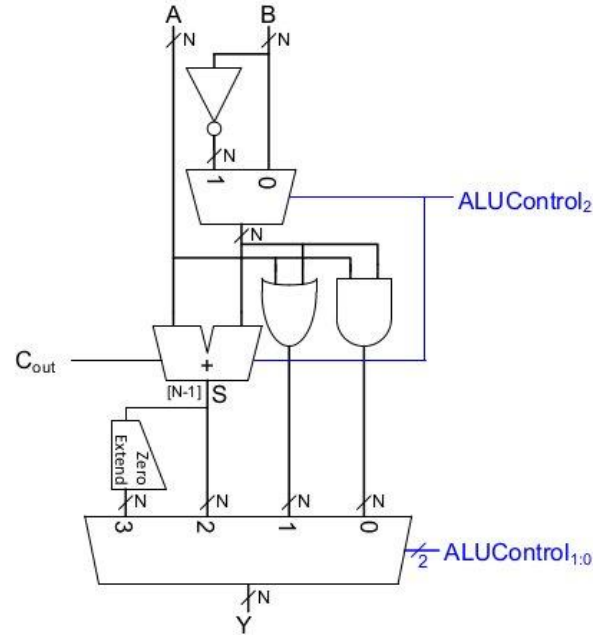
ALUControl _{2:0}	Function
000	A & B
001	A B
010	A + B
110	A - B
111	SLT

Desplazar shift left logic
 shift right logic
 shift right arithmetic

2- Unidad Aritmético Lógico

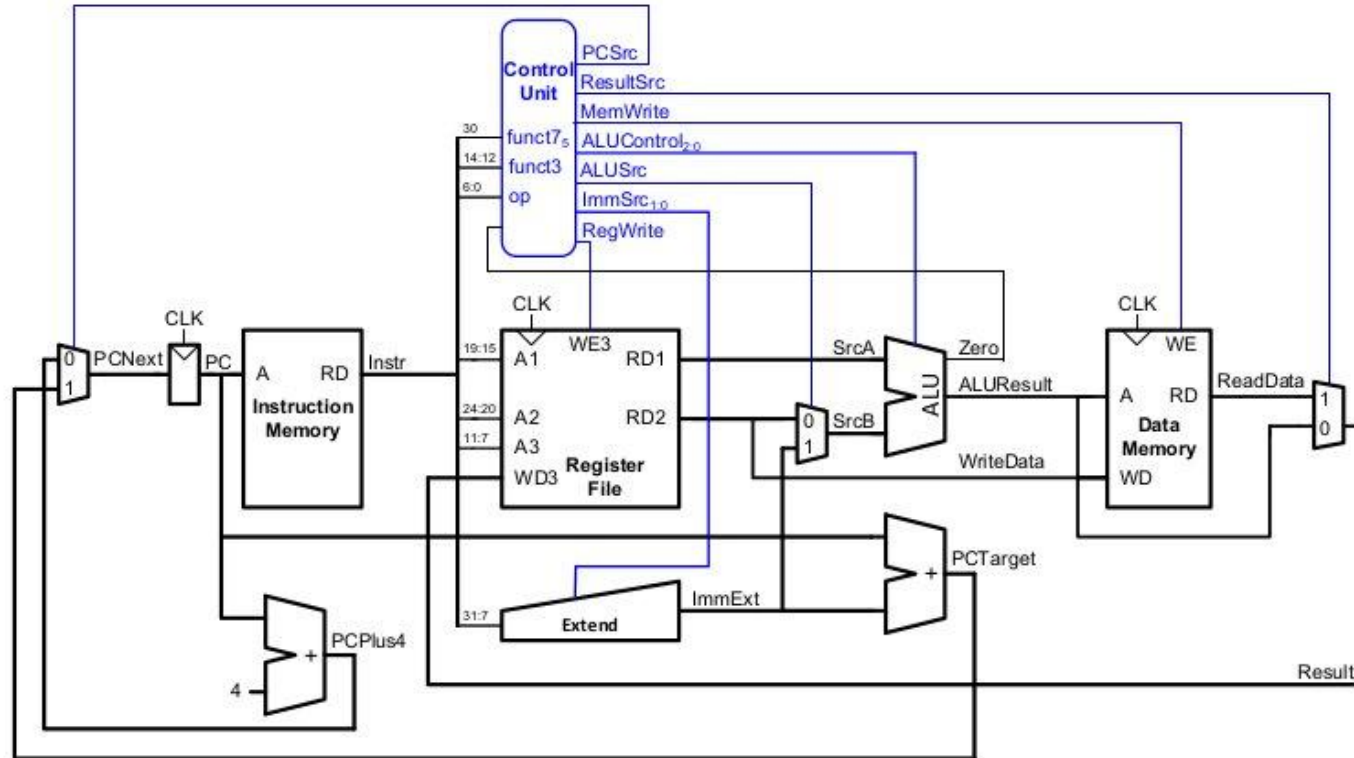
Si recordamos las operaciones encontramos: TR TI **Ap.**

Aritméticas	Suma	*	*
	Resta		*
Lógicas	And	*	*
	Or	*	*
Comparar	SLT	*	*



ALUControl _{2:0}	Function
000	A & B
001	A B
010	A + B
110	A - B
111	SLT

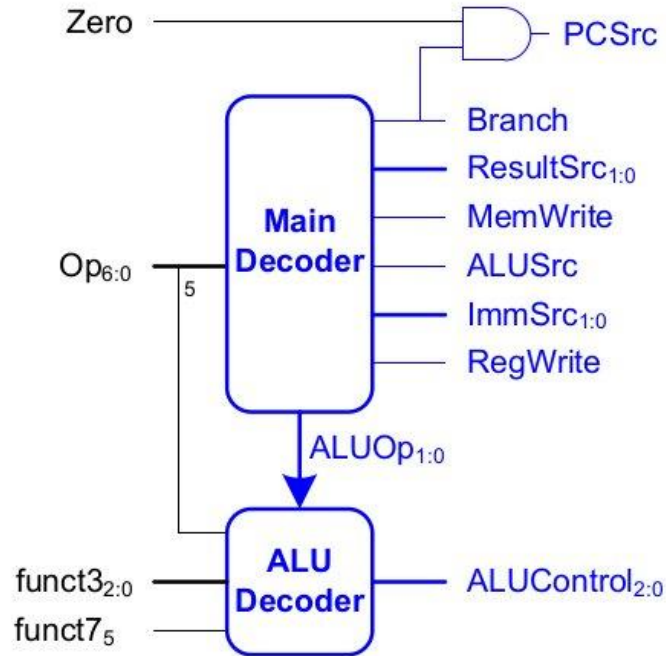
3- Control - La Unidad de Control



Algo tenía que controlar todo esto...

3- Control - Unidad de Control

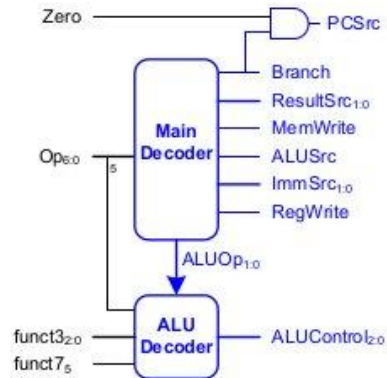
Un poco más en detalle.



3.1- Control - Unidad de Control Principal

¿Cómo se construye? Mediante tabla de verdad.

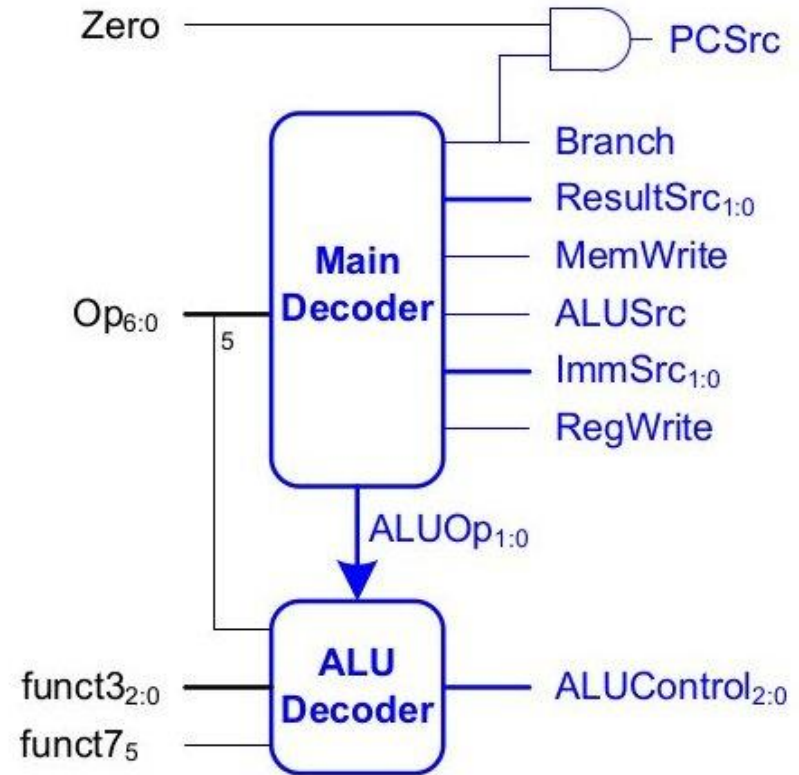
op	Instruct	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
3	lw	1	00	1	0	1	0	00
35	sw	0	01	1	1	X	0	00
51	R-type	1	XX	0	0	0	0	10
99	beq	0	10	0	0	X	1	01



3.2- ALU Decoder

Ya sabemos lo qué tiene que hacer.
Ahora le diremos en qué caso hacerlo.

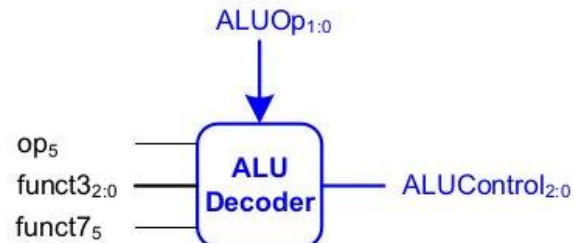
*Ver Guia practica RISC-V Fig 2.3



3.2- ALU Decoder

Mediante Tabla de verdad

ALUOp	op ₅	funct3	funct7 ₅	Instruction	ALUControl _{2:0}
00	X	X	X	lw, sw	010 (add)
01	X	X	X	beq	110 (subtract)
10	X	000	0	add	010 (add)
	1	000	1	sub	110 (subtract)
	X	010	0	slt	111 (set less than)
	X	110	0	or	001 (or)
	X	111	0	and	000 (and)



4- Nota sobre el mapa memoria

En el RV32I, con un PC de 32bits; se accede a un espacio de memoria de 4 Gby. Como el acceso se realiza de 4 en 4 el rango de memoria se reduce de 0x00000000 a 0xFFFFF000.

Por otra parte el espacio de direcciones está dividido en segmentos: El segmento de texto o código, el segmento de datos globales, el segmento de datos dinámicos, segmentos reservados para el kernel, segmento de dispositivos mapeados en memoria.

4- Nota sobre el mapa memoria

Si bien la construcción física del RV32I puede tomar cualquier forma.

La distribución lógica de los segmentos que propone el RARS es la siguiente.

<div>Configuration</div> <div><input checked="" type="radio"/> Default</div> <div><input type="radio"/> Compact, Data at Address 0</div> <div><input type="radio"/> Compact, Text at Address 0</div>	0xffffffff	memory map limit address
	0xffffffff	kernel space high address
	0xffff0000	MMIO base address
	0x80000000	kernel space base address
	0x7fffffff	user space high address
	0x7fffffff	data segment limit address
	0x7ffffffc	stack base address
	0x7fffeffc	stack pointer (sp)
	0x10040000	stack limit address
	0x10040000	heap base address
	0x10010000	.data base address
	0x10008000	global pointer (gp)
	0x10000000	data segment base address
	0x10000000	.extern base address
	0x0ffffffc	text limit address
	0x00400000	.text base address

4- Nota sobre el mapa memoria

Esta vista compacta nos ofrece un RV con 16 bits de PC.

Y con esta trabajaremos.!

Configuration <input type="radio"/> Default <input type="radio"/> Compact, Data at Address 0 <input checked="" type="radio"/> Compact, Text at Address 0	0x00007fff	memory map limit address
	0x00007fff	kernel space high address
	0x00007f00	MMIO base address
	0x00004000	kernel space base address
	0x00003fff	user space high address
	0x00003fff	data segment limit address
	0x00003ffc	stack pointer (sp)
	0x00003ffc	stack base address
	0x00003000	stack limit address
	0x00003000	heap base address
	0x00002000	.data base address
	0x00001800	global pointer (gp)
	0x00001000	data segment base address
	0x00001000	.extern base address
	0x00000ffc	text limit address
	0x00000000	.text base address

4- Nota sobre el mapa memoria

Dadas las particularidades de nuestro RV didáctico, nos basamos en el modelo compacto con solo tres segmentos.

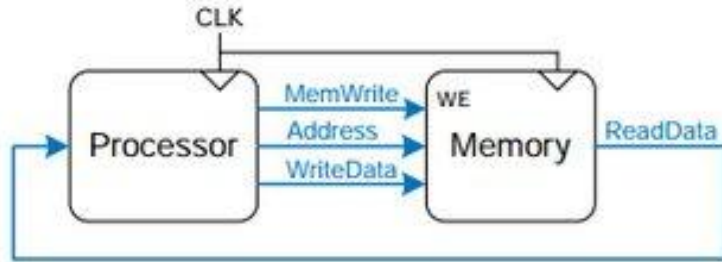
El segmento de código coincide con la memoria de instrucciones, comenzando en la dirección 0x0000 extendiéndose M palabras de 32 bits.

El segmento de datos coincide con la memoria de datos comenzando en la dirección 0x2000 extendiéndose N palabras de 32 bits. (Una opción es el caso de compartirlo desde atrás con el stack)

El segmento de dispositivos mapeados comenzando en la dirección 0x7FF0 extendiéndose 15 palabras de 32 bits.

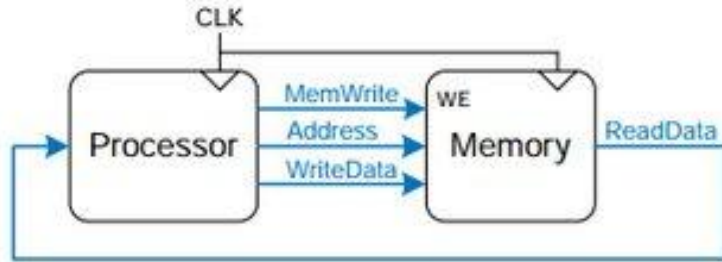
4- Unidad de gestión de memoria (MMU)

En este punto nuestro procesador es algo así:

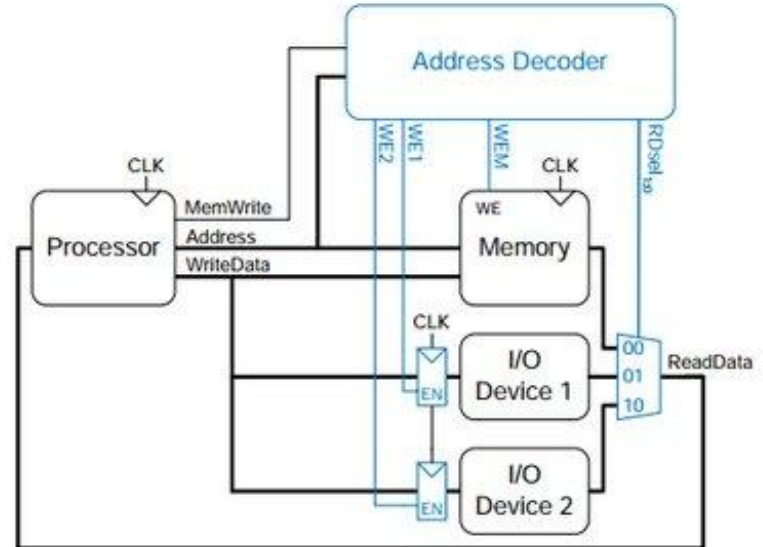


4- Unidad de gestión de memoria (MMU)

En este punto nuestro procesador es algo así:



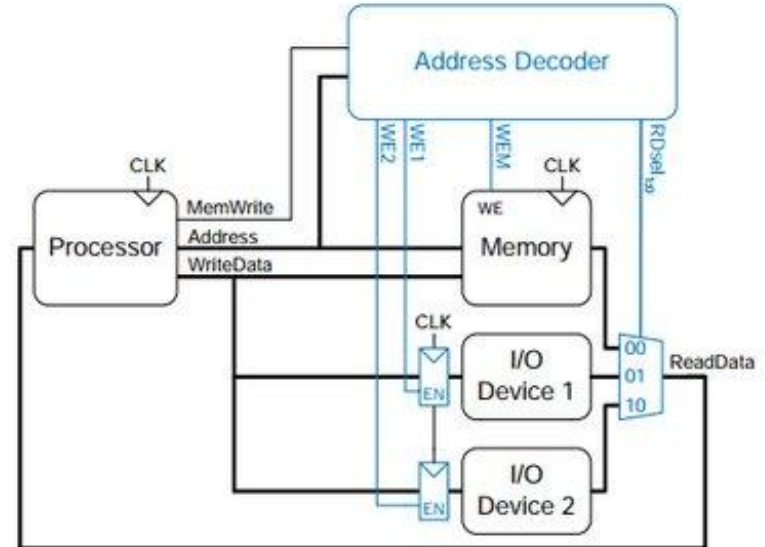
Nuestro objetivo es algo así:



4- Unidad de gestión de memoria (MMU)

El Address Decoder es nuestro MMU.

Quien se encarga de habilitar la Escritura o Lectura según la dirección accedida.



5- Tarea

Los animo a recrear el RV32I monociclo con los componentes y luego probarlo.

- hacer el camino para una instrucción *addi*.
- implementar la instrucción *jal*.

that's all !