Organización de computadoras

2024

Temario

1- Introducción

2- Simulador

3- Instrucciones

4- Práctica 1

1- Introducción

1.0 Bibliografía

1.1 Computadora

1.2 Memoria

1.3 Núcleo

1.4 Registros

1.5 Camino de datos

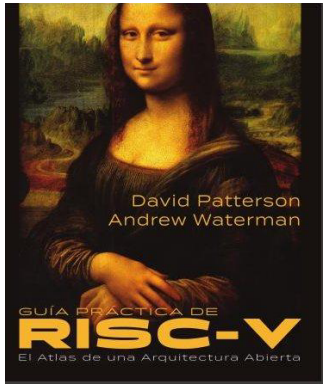
1.6 Direccionamiento a memoria

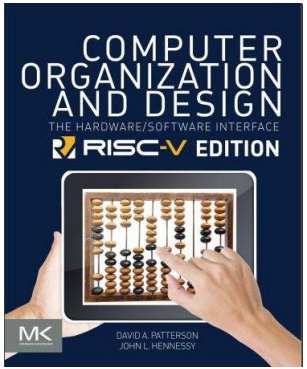
1.7 Formato de palabras

1.0- Bibliografía

Patterson, D.; Watterman, A. - *Guia Practica de RISC-V*

Hennessy, J.; Patterson D. - *Computer Organization and Design - The hardware software interface [RISC-V Ed]*

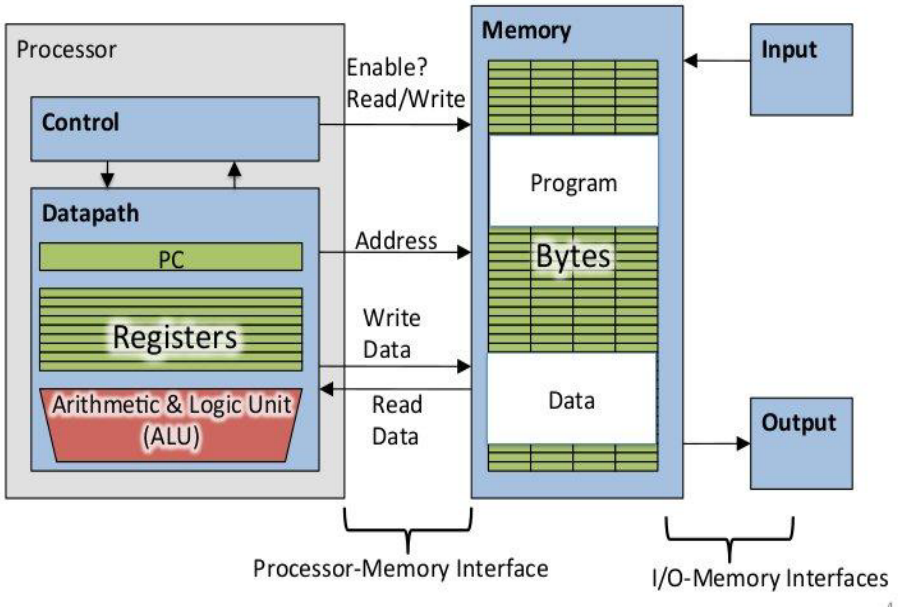




1.1- Computadora

Desde un punto de vista electrónico, la computadora es una **máquina secuencial sincrónica** cuya tarea consiste básicamente en **“leer, interpretar, ejecutar”**.

¿Qué cosa? Programas.

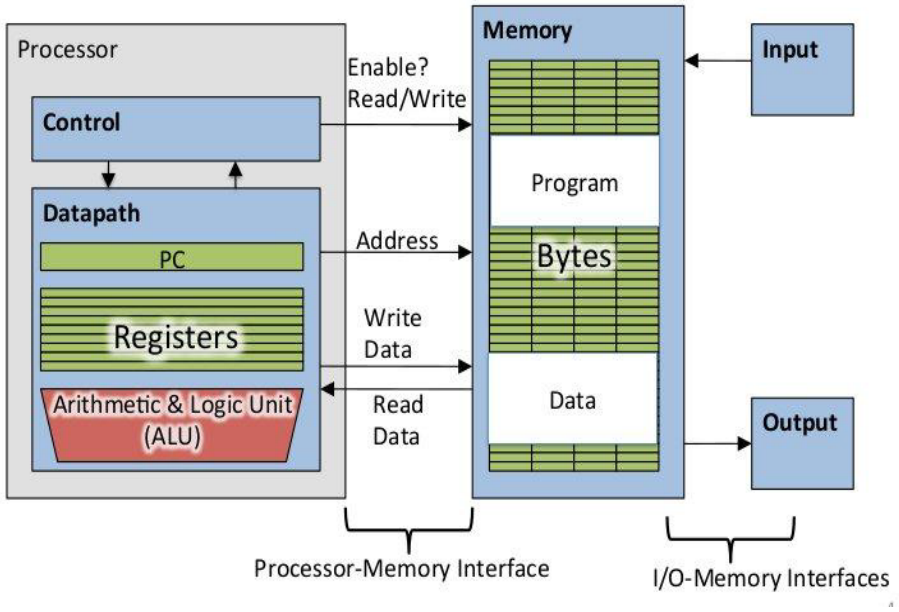


1.2- Memoria

El programa es cargado en memoria.

El procesador es encargado de recuperar “**leer**” esa información de la memoria.

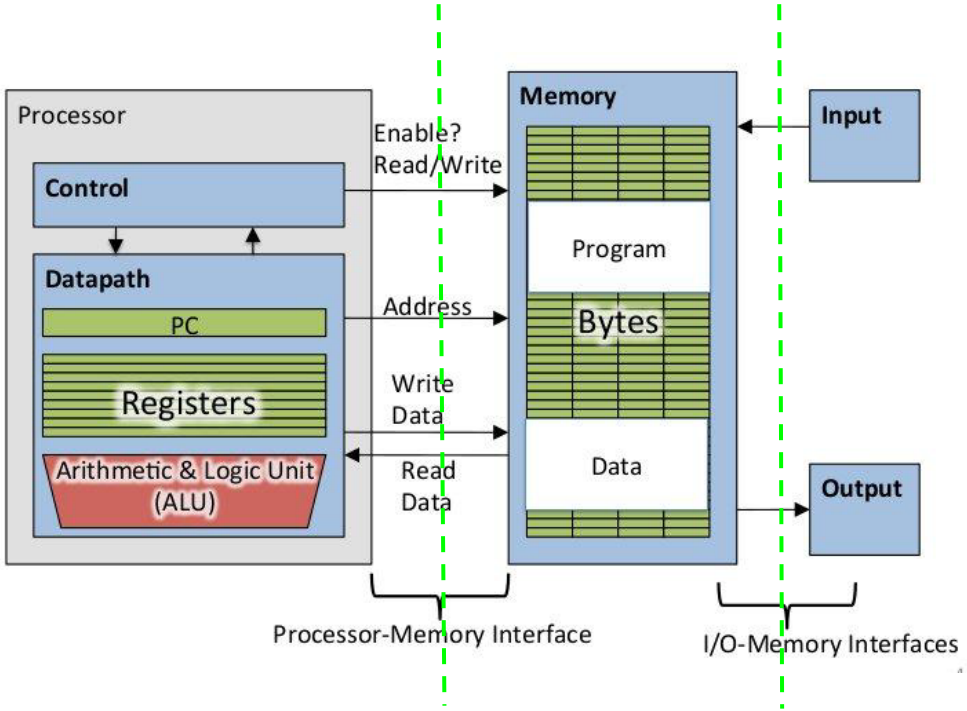
Los compiladores dividen la información en dos grupos: el código y los datos.



1.3- Núcleo

La decodificación es “**interpretar**” una instrucción del programa.

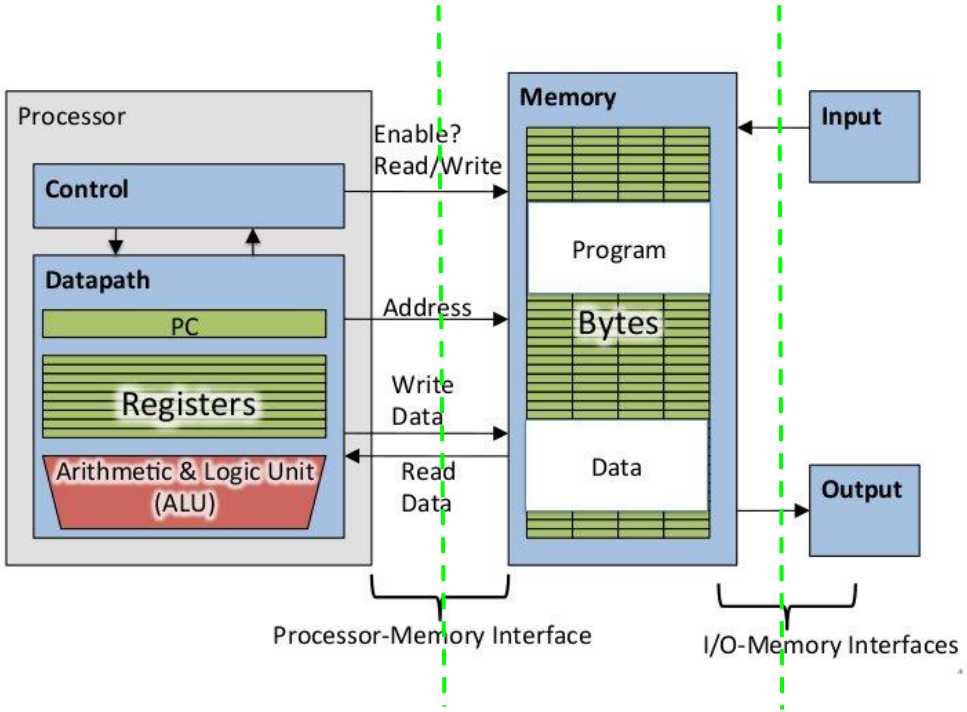
Esta tarea la realiza la **Unidad de Control** del procesador; disponiendo el **Camino de los datos** para tal fin.



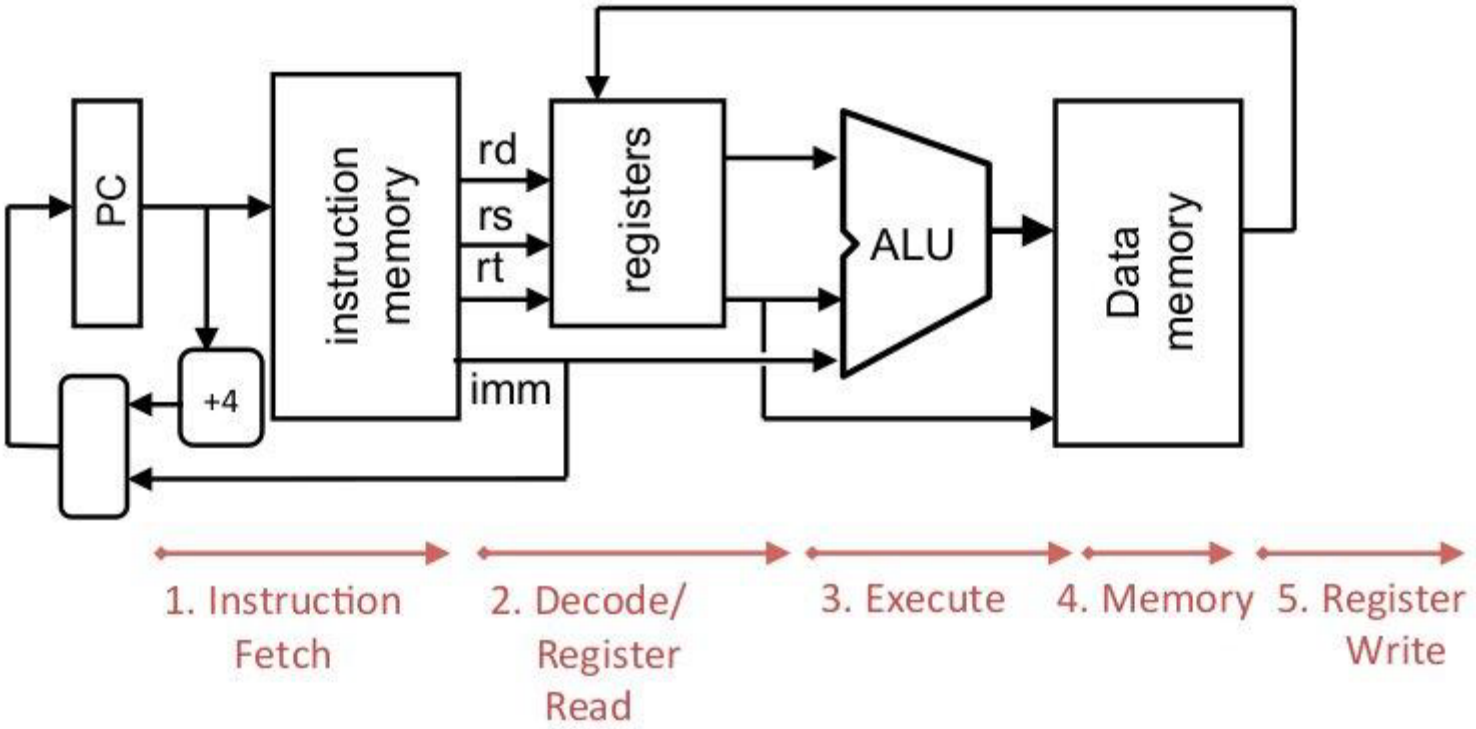
1.4- Registros

La información decodificada será almacenada en el **Banco de registros.**

Quedando allí disponibles para su "**ejecución**" en la **Unidad Aritmética Lógica**.



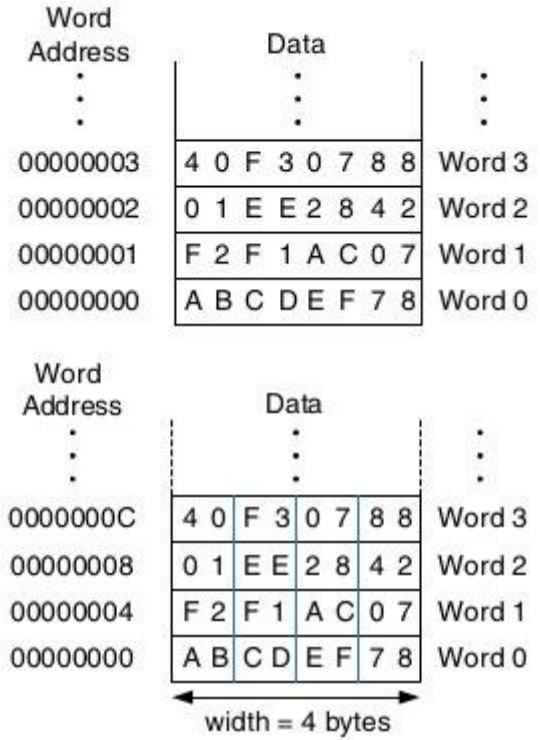
1.5- Camino de datos



1.6- Direccionamiento de Memoria

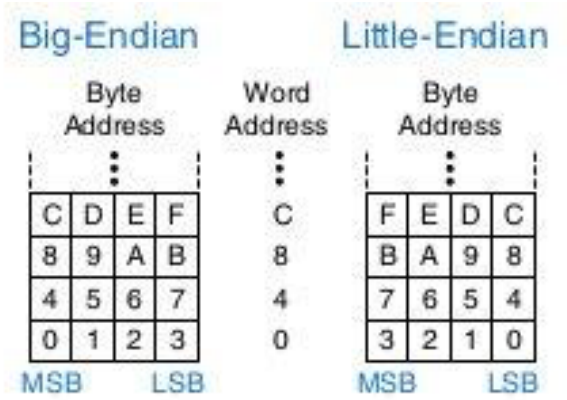
En el direccionamiento por palabra (32 bits) el PC va de 1 en 1.

En el direccionamiento por bytes el PC va de 4 en 4.



1.7- Formato de palabras

El formato de almacenamiento de instrucciones en memoria es Little Endian.



2- Simulador

Simulador del procesador risc-v RIPES

2.1- Obtención

2.2- Instalación

2.3- Uso

2.1- Obtención

2.2- Instalación

Instalación en **Ubuntu**

Instalar Java 10:

$ sudo apt-get install default-jdk

luego dar permisos de ejecución al archivo **RARS1_3.jar** y doble click.

o bien desde terminal:

$ java -jar RARS1_3.jar

Instalación en Windows

https://www.java.com/es/download/

...no tengo windows... 🤦‍♂️

2.3- Uso

El simulador tiene una interfaz simple y fácil, y funciones de depuración bastante intuitivas.

Al simulador ..!

3- Instrucciones

En el ISA (Instruction Set Architecture) RV32I, tenemos 47 instrucciones.

Cada instrucción es una palabra de 32 bits, almacenada como little-endian (31 downto 0), la cual está dividida en campos. Estos campos son:

- **Opcode**, correspondiente al código que identifica unívocamente a la instrucción.
- **Registros**, correspondientes a el o los operandos fuentes y destino de la ejecución.
- **Funciones**, para diferenciar entre instrucciones del mismo tipo.
- **Inmediatos**, corresponden constantes o direcciones de acceso a memoria.

3.1 Tipos de instrucciones

RISC-V es una arquitectura de carga y almacenamiento.

Las instrucciones aritméticas operan sólo entre los registros del banco o entre un registro y una constante.

Las instrucciones de carga y almacenamiento de datos operan como transferencia desde o hacia la memoria principal.

3.2- Formato de Instrucciones (en código de máquina)

R

I

S

SB

U

UJ

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	funct7							rs2					rs1					funct3			rd					OpCode							
N° Bits	7							5					5					3			5					7							

3.2.1 Instrucciones de cálculo (entre registros)

Instrucciones tipo R (operaciones entre registros) (10)

aritméticas	suma	add rd, rs1, rs2	rd <- rs1 + rs2
	resta	sub rd, rs1, rs2	rd <- rs1 - rs2
de bits	and	and rd, rs1, rs2	rd <- rs1 $\wedge$ rs2
	or	or rd, rs1, rs2	rd <- rs1 $\vee$ rs2
	xor	xor rd, rs1, rs2	rd <- rs1 $\veebar$ rs2
comparaciones	set if less than	slt rd, rs1, rs2	rd <- rs1 < rs2 1:0
	set if less than unsigned	sltur d, rs1, rs2	rd <- rs1 < rs2 1:0
desplazamientos	shift left logic	sll rd, rs1, rs2	rd <- rs1 << rs2
	shift right logic	srl rd, rs1, rs2	rd <- rs1 >> rs2
	shift right arithmetic	sra rd, rs1, rs2	rd <- rs1 >> rs2

3.1- Formato de Instrucciones (en código de máquina)

R

I

S

SB

U

UJ

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	funct7							rs2					rs1					funct3			rd					OpCode						
N° Bits	7							5					5					3			5					7						

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	imm [11:0]												rs1					funct3			rd					OpCode						
N° Bits	12												5					3			5					7						

3.2.2 Instrucciones de cálculo (con constantes)

Instrucciones tipo I (operaciones con constantes) (8)

aritméticas	suma	addi rd, rs1, Inm	rd <- rs1 + Inm
de bits	and	andi rd, rs1, Inm	rd <- rs1 $\wedge$ Inm
	or	ori rd, rs1, Inm	rd <- rs1 $\vee$ Inm
	xor	xori rd, rs1, Inm	rd <- rs1 $\veebar$ Inm
comparaciones	set if less than	slti rd, rs1, Inm	rd <- rs1 < Inm 1:0
desplazamientos	shift left logic	slli rd, rs1, Inm	rd <- rs1 << Inm
	shift right logic	srli rd, rs1, Inm	rd <- rs1 >> Inm
	shift right arithmetic	srai rd, rs1, Inm	rd <- rs1 >> Inm

3.1- Formato de Instrucciones (en código de máquina)

R

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	funct7							rs2					rs1					funct3			rd					OpCode							
N° Bits	7							5					5					3			5					7						

I

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	imm [11:0]												rs1					funct3			rd					OpCode							
N° Bits	12												5					3			5					7						

S

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	imm [11:5]							rs2					rs1					funct3			imm [4:0]					OpCode							
N° Bits	7							5					5					3			5					7						

SB

U

UJ

3.2.3- Instrucciones de lectura y almacenamiento

Tipo I (5)			
lectura	leer byte	lb **rd**, offset(**rs1**)	rd <- [rs1 + signExt(offset)]
	leer media palabra	lh rd, offset(rs1)	rd <- [rs1 + signExt(offset)]
	leer palabra	lw rd, offset(rs1)	rd <- [rs1 + signExt(offset)]
	leer byte unsig.	lbu rd, offset(rs1)	rd <- [rs1 + signExt(offset)]
	leer med. pal.unsig.	lhu rd, offset(rs1)	rd <- [rs1 + signExt(offset)]
Tipo S (3)			*
almac.	guardar byte	sb **rs2**, offset(**rs1**)	Mem[rs1+offset] <-rs2(7,0)
	guardar med. pal.	sh rs2, offset(rs1)	Mem[rs1+offset] <-rs2(16,0)
	guardar palabra	sw rs2, offset(rs1)	Mem[rs1+offset] <-rs2

3.1- Formato de Instrucciones (en código de máquina)

R

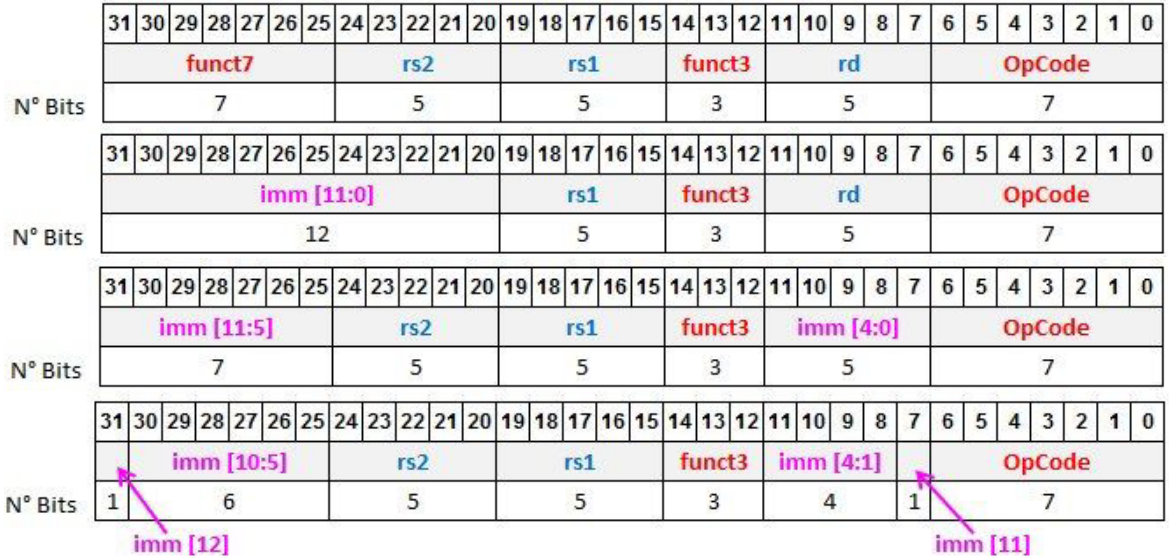
I

S

SB

U

UJ



3.2.4- Instrucciones de transferencia de control

Instrucciones tipo SB (operaciones control de flujo) (6)

saltar si igual	beq rs1 rs2 offset	si rs1=rs2 pc += sext(inm)
saltar si no igual	bnq rs1 rs2 offset	si rs1≠rs2 pc += sext(inm)
saltar si menor	blt rs1 rs2 offset	si rs1<rs2 pc += sext(inm)
saltar si mayor o igual	bge rs1 rs2 offset	si rs1≥rs2 pc += sext(inm)
saltar si menor unsig.	bltu rs1 rs2 offset	si rs1<rs2 pc += sext(inm)
saltar si mayor o igual uns.	bgeu rs1 rs2 offset	si rs1≥rs2 pc += sext(inm)
saltar al reg y enlazar	jalr rd offset(rs1)	rd <- pc + 4 pc <- rs1+sext(inm, 1b'0)

3.1- Formato de Instrucciones (en código de máquina)

R

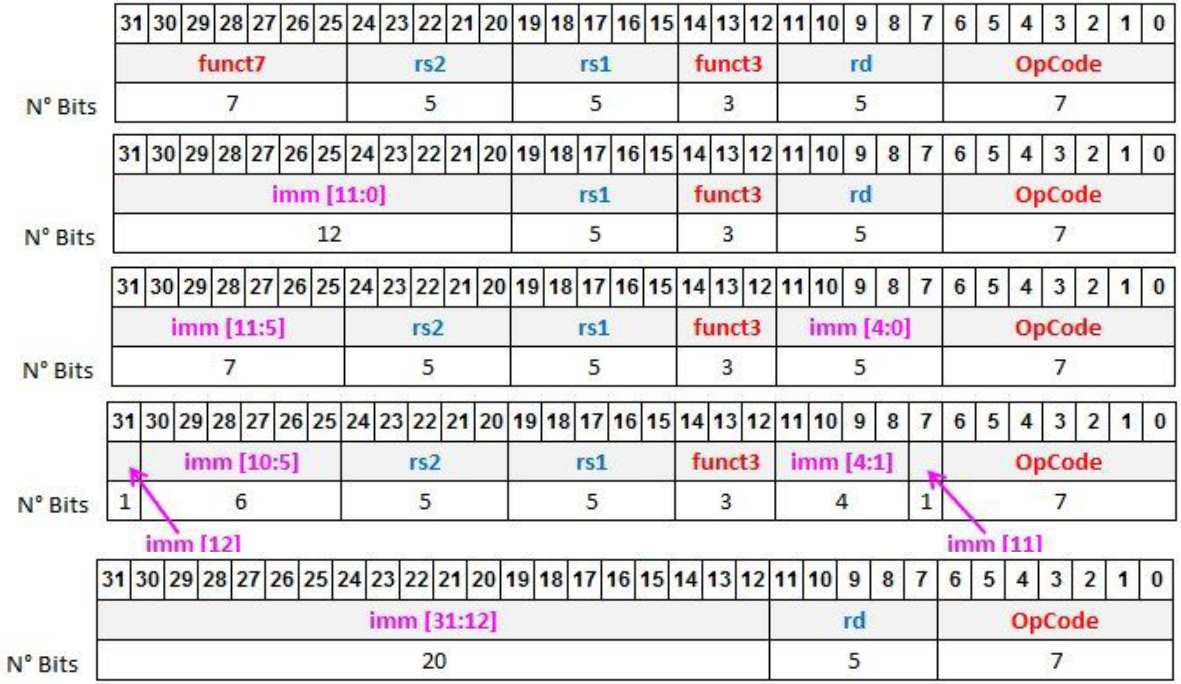
I

S

SB

U

UJ



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	funct7							rs2					rs1					funct3			rd					OpCode							
N° Bits	7							5					5					3			5					7							
I	imm [11:0]												rs1					funct3			rd					OpCode							
N° Bits	12												5					3			5					7							
S	imm [11:5]							rs2					rs1					funct3			imm [4:0]					OpCode							
N° Bits	7							5					5					3			5					7							
SB		imm [10:5]						rs2					rs1					funct3			imm [4:1]					OpCode							
N° Bits	1	6						5					5					3			4					7							
U	imm [31:12]																				rd					OpCode							
N° Bits	20																				5					7							

imm [12] imm [11]

3.1- Formato de Instrucciones (en código de máquina)

R

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
funct7							rs2					rs1					funct3			rd					OpCode							
N° Bits	7						5					5					3			5					7							

I

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
imm [11:0]												rs1					funct3			rd					OpCode							
N° Bits	12											5					3			5					7							

S

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
imm [11:5]							rs2					rs1					funct3			imm [4:0]					OpCode							
N° Bits	7						5					5					3			5					7							

SB

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	imm [10:5]						rs2					rs1					funct3			imm [4:1]					OpCode							
N° Bits	1	6					5					5					3			4				1	7							
	imm [12]																							imm [11]								

U

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
imm [31:12]																				rd					OpCode							
N° Bits	20																			5					7							

UJ

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	imm [10:1]											imm [19:12]								rd					OpCode							
N° Bits	1	10									1	8								5					7							
	imm [20]											imm [11]																				

3.2.5- Otras instrucciones especiales

Tipo U		
cargar inm arr.	lui rd offset	rd <- {inm_{20}, 12 b'0}
sumar inm arr. a pc	auipc rd offset	rd <- pc + ({inm_{20}, 12 b'0}
tipo UJ		
saltar y enlazar	jal rd offset	rd <- pc +4 pc <- pc + sext(inm_{20})

3.2.6- Otras instrucciones misceláneas

fence	Para modo privilegiado
fencei	
ecall	Llamadas al sistema
ebreak	Para debugger
crsrc	Registros de Control y Estado
crsrs	
crsrw	
crsrci	
crsrsi	
crsrwi	

3.2.7- Pseudo instrucciones

Son instrucciones que ayudan al proceso de ensamblado del programa, no traduciéndose a código máquina.
Las ofrece el simulador. Solo sirven como ayuda al programador.

4.1- Solución ejercicio 1

1. Escribir el siguiente código en el RARS.

```
.text
        lui t0, 0x12345
        lui t1, 201
        lui t2, 0xABCDE
```

La directiva .text indica que parte de lo escrito se cargará en el segmento de código (Text Segment). Grabar, ensamblar y responder lo siguiente:

a) ¿Qué diferencia se visualiza entre las instrucciones del código en Source y Basic?

4-1 Solución ejercicio 1

a) ¿Qué diferencia se visualiza entre las instrucciones del código en Source y Basic?

En Source los valores están tal cual los escribí en el editor.

En Disassembled los valores son traducidos (ABI) a registros de 32 bits en hexadecimal y binario.

4-1 Solución ejercicio 1

a) ¿Qué diferencia se visualiza entre las instrucciones del código en Source y Basic?

En Source los valores están tal cual los escribí en el editor.

En Basic los valores son traducidos (ABI) a registros de 32 bits en hexadecimal.

b) ¿Cuál es la dirección de comienzo del programa y que longitud tiene cada instrucción?

4-1 Solución ejercicio 1

a) ¿Qué diferencia se visualiza entre las instrucciones del código en Source y Basic?

En Source los valores están tal cual los escribí en el editor.

En Basic los valores son traducidos (ABI) a registros de 32 bits en hexadecimal.

b) ¿Cuál es la dirección de comienzo del programa y que longitud tiene cada instrucción?

La dirección de comienzo del programa es el valor del PC. Qué toma el valor de .text; en nuestro caso 0x00400000. (Ver Compacto .text)

La longitud de cada instrucción es de 4 bytes (32 bits).

4-1 Solución ejercicio 1

a) ¿Qué diferencia se visualiza entre las instrucciones del código en Source y Basic?

En Source los valores están tal cual los escribí en el editor.

En Basic los valores son traducidos (ABI) a registros de 32 bits en hexadecimal.

b) ¿Cuál es la dirección de comienzo del programa y que longitud tiene cada instrucción?

La dirección de comienzo del programa es el valor del PC. Qué toma el valor de .text; en nuestro caso 0x00400000. (Ver Compacto .text)

La longitud de cada instrucción es de 4 bytes (32 bits).

c) Escribir el código objeto (Code) de cada instrucción en binario

4-1 Solución ejercicio 1

a) ¿Qué diferencia se visualiza entre las instrucciones del código en Source y Basic?
En Source los valores están tal cual los escribí en el editor.
En Basic los valores son traducidos (ABI) a registros de 32 bits en hexadecimal.

b) ¿Cuál es la dirección de comienzo del programa y que longitud tiene cada instrucción?
La dirección de comienzo del programa es el valor del PC. Qué toma el valor de .text; en nuestro caso 0x00400000. (Ver Compacto .text)
La longitud de cada instrucción es de 4 bytes (32 bits).

c) Escribir el código objeto (Code) de cada instrucción en binario
Basta con seleccionar Binary en el lado derecho del editor.

Hexa	*Bin*
123452B7	*00010010001101000101001010110111*
000C9337	*00000000000011001001001100110111*
ABCDE3B7	*10101011110011011110001110110111*

4-1 Solución ejercicio 1 (cont)

d) ¿Qué valores inicialmente tienen los registros t0, t1, t2 y pc?

4-1 Solución ejercicio 1 (cont)

d) ¿Qué valores inicialmente tienen los registros t0, t1, t2 y pc?

t0 0x00000000
t1, 0x00000000
t2, 0x00000000
PC, 0x00400000

4-1 Solución ejercicio 1 (cont)

d) ¿Qué valores inicialmente tienen los registros t0, t1, t2 y pc?

t0 0x00000000
t1, 0x00000000
t2, 0x00000000
PC, 0x00400000

e) ¿En qué dirección comienza el segmento de datos?

4-1 Solución ejercicio 1 (cont)

d) ¿Qué valores inicialmente tienen los registros t0, t1, t2 y pc?

t0 0x00000000
t1, 0x00000000
t2, 0x00000000
PC, 0x00400000

e) ¿En qué dirección comienza el segmento de datos?

.data 0x10010000

4-1 Solución ejercicio 1 (cont)

d) ¿Qué valores inicialmente tienen los registros t0, t1, t2 y pc?

t0 0x00000000
t1, 0x00000000
t2, 0x00000000
PC, 0x00400000

e) ¿En qué dirección comienza el segmento de datos?

.data 0x10010000

f) Presionar F5 para ejecutar la primera instrucción. ¿Qué valor toma t0?

4-1 Solución ejercicio 1 (cont)

d) ¿Qué valores inicialmente tienen los registros t0, t1, t2 y pc?

t0 0x00000000
t1, 0x00000000
t2, 0x00000000
PC, 0x00400000

e) ¿En qué dirección comienza el segmento de datos?

.data 0x10010000

f) Presionar F5 para ejecutar la primera instrucción. ¿Qué valor toma t0?

t0, 0x12345000

4-1 Solución ejercicio 1 (cont)

d) ¿Qué valores inicialmente tienen los registros t0, t1, t2 y pc?

t0 0x00000000
t1, 0x00000000
t2, 0x00000000
PC, 0x00400000

e) ¿En qué dirección comienza el segmento de datos?

.data 0x10010000

f) Presionar F7 para ejecutar la primera instrucción. ¿Qué valor toma t0?

t0, 0x12345000

g) ¿En cuánto y por qué cambia el valor del registro PC? ¿Cuál es su función?

4-1 Solución ejercicio 1 (cont)

d) ¿Qué valores inicialmente tienen los registros t0, t1, t2 y pc?
t0 0x00000000
t1, 0x00000000
t2, 0x00000000
PC, 0x00400000

e) ¿En qué dirección comienza el segmento de datos?
.data 0x10010000

f) Presionar F7 para ejecutar la primera instrucción. ¿Qué valor toma t0?
t0, 0x12345000

g) ¿En cuánto y por qué cambia el valor del registro pc? ¿Cuál es su función?
cambia de 0x00400000 a 0x00400004
su función es indicar la dirección de la próxima instrucción.

h) Seguir ejecutando el programa (F7) y verificar los valores en t1, t2 y PC.

Preguntas