Renacuajo:
A simple SISA processor

Raúl Gilabert Gámez i Pol Saumell Hill

Projecte d'Enginyeria de Computadors 2024

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Índex

1

Etapa 1: Processador base

1.1 Introducció

Aquesta primera etapa del processador serà la més senzilla de totes. Comptarà només amb tres instruccions: *MOVI*, *MOVHI* i *HALT*.

Per poder implementar awuestes instruccions ens cal una unitat de control amb la etapa de *fetch* que decodifiqui la instrucció que tingui a memòria.

En aquesta etapa només tindrem memòria de instruccions, no tindrem de dades.

Comptem amb 4 blocs principals:

- Banc de registres: 8 registres de propòsit general.
- ALU: podem executar les instruccions de *MOVI* i *MOVHI*.
- Control: ens permet gestionar totes les senyals del nostre processador.
- Memòria: en aquesta etapa no l'hem hagut d'implementar.

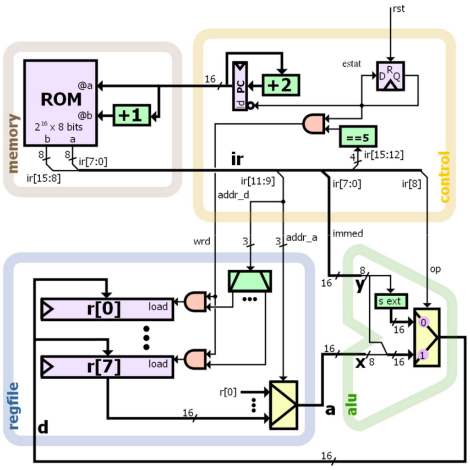

Figura 1.1: Esquema del processador en la primera etapa.

1.2 Instruccions

1.2.1 Instruccions amb immediat

MOVI

Escriu l'immediat codificat en 8 bits i l'escriu en el registre destí de 16 bits fent-li extensió de signe.

15 14 13 12	11 10 9	8	7 6 5 4 3 2 1 0		
0 1 0 1	Rd	0	n n n n n n n n	Moure immediat	MOVI

Taula 1.1: Codificació de *MOVI* en lleguatge màquina.

MOVHI

Escriu els 8 bits de l'immediat a la part alta del registre destí de 16 bits. Amb aquesta instrucció i *MOVI* es poden escriure registres de 16 bits.

15 14 13 12	11 10 9	8	7 6 5 4 3 2 1 0		
0 1 0 1	Rd Ra	1	n n n n n n n n	Moure immediat	MOVHI

Taula 1.2: Codificació de *MOVHI* en lleguatge màquina.

1.2.2 Instrucció especial

HALT

Instrucció especial que ens permet aturar el PC del processador.

15 14 13 12	11 10 9	8 7 6	5	4 3 2 1 0		
1 1 1 1	1 1 1	1 1 1	1	f f f f f	Instrucció especial	HALT

Taula 1.3: Codificació de *HALT* en lleguatge màquina.

1.3 Banc de registres

El banc de registres d'aquesta etapa és una versió molt simple. Només té un port de lectura i un d'escriptura.

Compte amb:

- 8 registres de 16 bits. Adreçats amb 3 bits.
- Un port de lectura que es selcciona amb *addr_a*.
- Un port d'escriputra que es selecciona amb *addr_d*.
- Un permís d'escriptura amb *wr_d*.

1.4 ALU

En aquesta versió l'ALU és capaç de executar les instruccions de *MOVI* i *MOVHI*. Farà de canal d'arribada fins el banc de registres i combinar els bits de menys pes del registre *Ra* amb els de més pes amb la crida *MOVHI*.

Compte amb:

- Dues entrades de 16 bits, x i y.
- Un selector d'operació *op* d'un bit per escollir entre les dues possibles.
- Una sortida de 16 bits que va directament al banc de registres.

1.5 Unitat de control

En aquesta etapa el processador és unicicle ja que no tenim ca risc estructural.

Carrega el PC en el cicle següent amb $PC + 2$. Això ho fa sempre menys quan hi ha una instrucció de *HALT* que aleshores no es carrega cap valor a PC.

L'unitat compte amb:

- Entrada del *ir* instrction register, on depenent de l'instrucció la lògica selecciona els senyals corresponents.
- Sortida de *addr_a*, ens indica en quin registre s'ha de llegir en el port a.
- Sortida de *addr_d*, ens indica en quin registre s'ha d'escriure en el port d.
- Sortida del permís d'escriptura *wrd*, habilita l'escriptura en el banc de reistres. Només es deshabilita en l'instrucció *HALT*.
- Sortida del codi d'operació *op*, difenrent en funció del contigut del *ir*.
- Sortida del immediat *immed*, extreu el immediat del *ir*.

2

Etapa 2.1: Processador multicicle

2.1 Introducció

En aquesta etapa s'implementen les instruccions de accès a memòria *LD*, *ST*, *LDB* i *STB*.

Com que la memòria de instruccions i la de dades és la mateixa ens apareix un risc estructural. La solució és fer el processador multicicle.

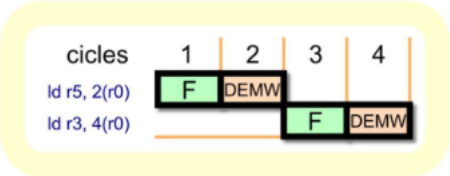

Figura 2.1: Diagrama temporal del cicles.

Aquest canvi complica la lògica de control es necessita un nou mòdul que controli l'estat del multicicle.

Comptem amb 4 blocs principals:

- Banc de registres: 8 registres de propòsit general.
- ALU: podem executar les instruccions de *MOVI*, *MOVHI* i ens calcula les adreces per els accessos a memòria.
- Control:
  - Control multicicle: Implementa el control d'estats del processador multicicle.
  - Lògica de control: Controla les senyals en funció de l'estat del processador.
- Memòria: S'implementen les instruccions d'accès a memòria. Es fa servir memòria entrellaçada.

2.2 Instruccions

S'afegeixen les seguents instrccions a les de l'etapa 1 (miri la secció 1.2):

2.2.1 Instruccions d'accés a memòria

LD

Carrega en el registre *Rd* el que la paraula guardada en l'adreça $Ra + Immed.$ de memòria.

15 14 13 12	11 10 9	8 7 6	5 4 3 2 1 0		
0 0 1 1	Rd	Ra	n n n n n n	Load	LD

Taula 2.1: Codificació de *LD* en lleguatge màquina.

ST

Guarda la paraula guardada en *Rb* a l'adreça $Ra + Immed.$ de memòria.

15 14 13 12	11 10 9	8 7 6	5 4 3 2 1 0		
0 1 0 0	Rb	Ra	n n n n n n	Store	ST

Taula 2.2: Codificació de *ST* en lleguatge màquina.

LDB

Carrega en el registre *Rd* el que la paraula guardada en l'adreça $Ra + Immed.$ de memòria.

15 14 13 12	11 10 9	8 7 6	5 4 3 2 1 0		
1 1 0 0	Rd	Ra	n n n n n n	Load Byte (8 bits)	LDB

Taula 2.3: Codificació de *LDB* en lleguatge màquina.

STB

Guarda el byte guardat en *Rb* a l'adreça $Ra + Immed.$ de memòria.

15 14 13 12	11 10 9	8 7 6	5 4 3 2 1 0		
1 1 0 1	Rb	Ra	n n n n n n	Store Byte (8 bits)	STB

Taula 2.4: Codificació de *STB* en lleguatge màquina.

2.3 Banc de registres

Aquesta versió del banc de registres té un port de lectura i dos d'escriptura. Aquesta crec que serà la versio final.

Compte amb:

- 8 registres de 16 bits. Adreçats amb 3 bits.
- Port de lectura *a* que es selcciona amb *addr_a*.
- Port de lectura *b* que es selcciona amb *addr_b*.
- Un port d'escriptura que es selecciona amb *addr_d*.
- Un permís d'escriptura amb *wr_d*.

2.4 ALU

En aquesta versió l'ALU es manté la compatibilitat per les instrccions implementades en l'etapa 1 (Miri la secció 1.2) i s'afegeix suport per poder calcular l'adreça per a lectures o escriptures.

Compte amb:

- Dues entrades de 16 bits, x i y.
- Un selector d'operació *op* de dos bit per escollir entre sumar l'immediat amb *Ra* per a poder calcuar l'adreça o fer les operacions de *MOVI* i *MOVHI*.
- Una sortida de 16 bits que va directament al banc de registres.

2.5 Unitat de control

Cal implementar un processador multicicle per evitar riscos estructurals. S'implenenten dos cicles, un encarregat només del *fetch* i l'altre encarregat del *decode*, *execute*, accès a mèmoria i d'escriptura.

Igual que en l'etapa anterior s'actualitza el PC amb $PC + 2$ sempre i quan no s'executi una instrucció de *HALT*.

2.5.1 Control multicicle

Controla la lògica d'estats del processador i a partir de l'estat habilitat o deshabilita senyals.

Les senyals afectades per l'estat del processador són:

- *wrd* s'habilita en la segona etapa per evitar escriptures al banc de registres.
- *wr_m* s'habilita en la segona etapa per evitar escriptures a memòria.
- *word_bytes* s'habilita en la segona etapa per què no té sentit que ho estigui en la primera.
- *ldpc* s'habilita en la segona etapa per poder carregar el següent PC.

2.5.2 Lògica de control

En funció de l'instrucció que conté el *ir* s'activen certs senyals:

- *op* li inidica a la ALU quina operació he d'executar.

op	**Instrucció**	**op**	**Instrucció**
00	MOVI	10	LD, ST, LDB, STB
01	MOVHI	11	LD, ST, LDB, STB

- *ldpc* inidica si s'ha de carregar el nou PC en la següent etapa de *fetch*. S'activa en totes les instruccions menys en la de *HALT*.
- *wrd* habilita l'escriptura al banc de registres, s'activa en les instruccions de *MOVI*, *MOVHI*, *LD* i *LDB*.
- *addr_a* es carrega amb *ir(11 DOWNTO 9)* quan executem un *MOVI* o *MOVHI* i *ir(8 DOWNTO 6)* amb la resta d'instruccions.
- *addr_b* sempre pren el valor de *ir(11 DOWNTO 9)*.
- *addr_d* sempre pren el valor de *ir(11 DOWNTO 9)*.
- *immed* pren el valor del immediat de l'instrucció amb extensió de singne quan s'executa un *MOVI* o *MOVHI* i sense extensió de signe en la resta d'instruccions.
- *wr_m* s'activa quan cal escriure a memòria. En les instruccions de *ST* i *STB*.
- *in_d* habilita que es guardi el resultat en registre *Rd*.
- *immed_x2* en les instrccions de *ST* i *LD* s'activa aquest senyal per indicar que cal multiplicar el immeditat per 2.
- *word_byte* selecciona si cal escriure un byte o una paraula. S'activa amb les instruccions de *STB* i *LDB*.

3

Etapa 2.2: Controlador de memòria

3.1 Introducció

En aquesta nova etapa passem a tenir un processador a un SoC (*System on Chip*) que repsresenta tot el sistema que comforma un computador, principalment el processador i la memòria.

Per fer possible això hem d'afegir al nostre processador una memòria. Farem servir una memòria SRAM. Aquesta memòria, ja integrada en la placa de desenvolupament (model *IS61LV25616* de la marca *ISSI*), necessita el seu controlador de memòria per poder-hi accedir. El controlador serà la part més important d'aquesta etapa.

3.2 Controlador de memòria

La memòra que disposem en la placa de desenvolupament necessita d'un controlador per poder-hi accedit ja que disposa d'un protocol que hem de seguir. Aquest protocol està descrit en la documentació de la memòria SRAM: `https://www.issi.com/ww/pdf/61lv25616al.pdf`

3.2.1 Controlador de SRAM

Les senyaks independents del protocol i que depenen de l'arquitectura de la memòria són:

- *SRAM_ADDR* pren el valor de l'adreça que li passem de la unitat de control però desplacem un bit cap a l'esquerra.
- *SRAM_UB_N* aquest senyal val el bit més baix de l'adreça negat quan volem escriure un byte, o '0' quan no volem escriure.
- *SRAM_LB_N* aquest senyal val el bit més baix de l'adreça quan volem escriure un byte, o '0' quan no volem escriure.
- *SRAM_OE_N* sempre pren el valor de '0', ja que quan volem llegir ha d'estar a '0' i quan volem escriure ens es indiferent. Per tant, aquest senyal sempre està a '0'.
- *SRAM_CE_N* aquest senyal sempre està a '0'.
- *SRAM_DQ* està en alta impedància quan es vol llegir un valor, i quan es vols escriure se li fica la dada a escriure al bus.

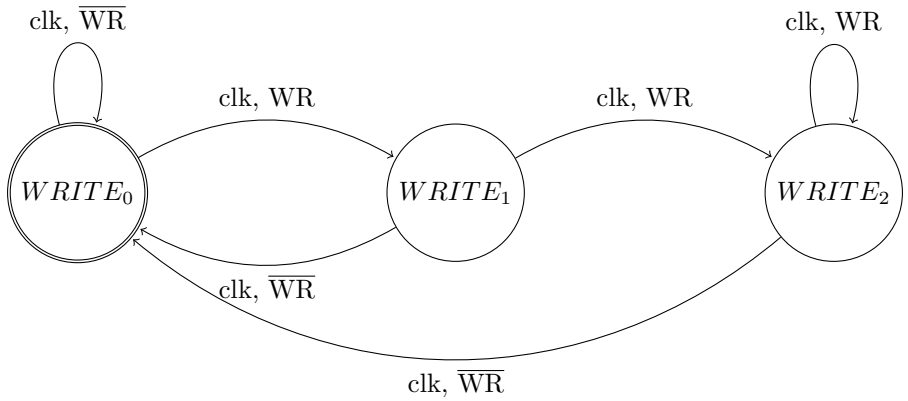

Figura 3.1: Automàta del controlador de SRAM.

Per poder llegir un valor de memòria només es passa pel primer estat, ja que el temps que requeireix per llegir és només d'un cicle.

El controlador de memòria triga 3 cicles en escriure un valor en memòria, passant pels tres estats de l'automàta. Aquest procés és necessàri per respectar els temps que requereix la memòria.

4

Etapa 3: Unitat aritmeticològica

4.1 Introducció

En aquesta etapa donarem suport a la ALU per les instruccions aritmeticològiques i comparacions.

Per l'implementació de la ALU hem optat per una versió similar a un processador RISC, per facilitat i per poder expendir el ISA si es vol.

Comptem amb 4 blocs principals:

- Banc de registres: Es manté el de l'etapa 2.1 (miri secció 2.3).
- ALU: es manté compatibilitat amb les instruccions anteriors i s'afegeix les unitats de càlcul corresponents per les noves instruccions.
- Control: es canvia la lògica de control per poder afegir les noves instruccions i mantenir les instruccions anteriors.
- Memòria: Es manté l'implementació de l'etapa 2.2 (miri secció 3.2.1).

4.2 Instruccions

En aquesta versió l'ALU es manté la compatibilitat per les instrccions implementades en l'etapa 2 (miri la secció 2.2) i s'afegeix suport per operacions aritmeticològiques i de comparació.

4.2.1 Operacions aritmetològiques

Dins de la següent codificació es selecciona l'operació concreta amb els tres bits de funció *f f f*.

15 14 13 12	11 10 9	8 7 6	5 4 3	2 1 0		
0 0 0 0	Rd	Ra	f f f	Rb	Operacions lògiques i aritmètiques	AND, OR, XOR, NOT ADD, SUB, SHA, SHL

Taula 4.1: Codificació de les operacions aritmetològiques en lleguatge màquina.

AND

Codi de funció 000.

Operació *AND* bit a bit entre el registre *Ra* i el *Rb*, es guarda a *Rd*.
$Rd = Ra \wedge Rb$

OR

Codi de funció 001.

Operació *OR* bit a bit entre el registre *Ra* i el *Rb*, es guarda a *Rd*.
$Rd = Ra \vee Rb$

XOR

Codi de funció 010.

Operació *XOR* bit a bit entre el registre *Ra* i el *Rb*, es guarda a *Rd*.
$Rd = Ra \oplus Rb$

NOT

Codi de funció 011.

Operació *NOT* bit a bit del registre *Ra*, es guarda a *Rd*.
$Rd = \neg Rb$

ADD

Codi de funció 100.

Suma dels registres *Ra* i *Rb*, es guarda a *Rd*.
$Rd = Ra + Rb$

SUB

Codi de funció 101.

Resta entre els registres *Ra* i *Rb*, es guarda a *Rd*.
$Rd = Ra - Rb$

SHA

Codi de funció 110.

Desplaçament aritmètic del registre *Ra Rb* vegades, es guarda a *Rd*. Es fa extensió de signe.

SHL

Codi de funció 111.

Desplaçament lògic del registre *Ra Rb* vegades, es guarda a *Rd*.

4.2.2 Operacions de comparació

Dins de la següent codificació es selecciona l'operació concreta amb els tres bits de funció *f f f*.

15 14 13 12	11 10 9	8 7 6	5 4 3	2 1 0		
0 0 0 1	Rd	Ra	f f f	Rb	Comparació amb i sense signe	CMPLT, CMPLE, -, CMPEQ, CMPLTU, CMPLEU

Taula 4.2: Codificació de les operacions de comparació en lleguatge màquina.

CMPLT

Codi de funció 000.

Comparació d'enters entre *Ra* i *Rb*, es guarda el resultat a *Rd*.
$Rd = Ra < Rb$.

CMPLTU

Codi de funció 100.

Comparació de naturasl entre *Ra* i *Rb*, es guarda el resultat a *Rd*.
$Rd = RaRb$.

CMPLE

Codi de funció 001.

Comparació d'enters entre *Ra* i *Rb*, es guarda el resultat a *Rd*.
$Rd = Ra \leq Rb$.

CMPLEU

Codi de funció 101.

Comparació de naturasl entre *Ra* i *Rb*, es guarda el resultat a *Rd*.
$Rd = Ra \leq Rb$.

CMPEQ

Codi de funció 011.

Comparació d'enters entre *Ra* i *Rb*, es guarda el resultat a *Rd*.
$Rd = Ra == Rb$.

4.2.3 Suma amb immediat

15 14 13 12	11 10 9	8 7 6	5 4 3	2 1 0		
0 0 1 0	Rd	Ra	n n n n n n		Suma amb immediat	ADDI

Taula 4.3: Codificació de l'operació de suma amb immediat en lleguatge màquina.

ADDI

Suma del registre *Ra* amb immediat de 6 bit, el resultat es guarda a *Rd*.
$Rd = Ra + Immed$

4.2.4 Ampliació de les operacions aritmètiques

Dins de la següent codificació es selecciona l'operació concreta amb els tres bits de funció *f f f*.

15 14 13 12	11 10 9	8 7 6	5 4 3	2 1 0		
1 0 0 0	Rd	Ra	f f f	Rb	Extensió aritmètica	MUL, MULH, MULHU, -, DIV, DIVU

Taula 4.4: Codificació de les operacions d'ampliació aritmètica en lleguatge màquina.

MUL

Codi de funció 000.

Multiplació entre *Ra* i *Rb*, es guarda en *Rd* els 16 bits de menys pes del resultat.
$Rd = Ra \cdot Rb$

MULH

Codi de funció 001.

Multiplació d'enters entre *Ra* i *Rb*, es guarda en *Rd* els 16 bits de més pes del resultat.
$Rd = Ra \cdot Rb$

MULHU

Codi de funció 010.

Multiplació de naturals entre *Ra* i *Rb*, es guarda en *Rd* els 16 bits de més pes del resultat.
$Rd = Ra \cdot Rb$

DIV

Codi de funció 100.

Divisió d'enters entre *Ra* i *Rb*, el resultat es guarda en *Rd*.
$Rd = Ra \div Rb$

DIVU

Codi de funció 101.

Divisió de naturals entre *Ra* i *Rb*, el resultat es guarda en *Rd*.
$Rd = Ra \div Rb$

4.3 ALU

La nova ALU ens permet executar les noves instruccions. Compte amb varies unitats de càlcul, que calculen el resultat de totes les possibles operacions. En funció del codi d'operació que rep de la unitat de control surt un resultat o un altre.

Compte amb:

- Quatre unitats que calculen les operacions lògiques: AND, OR, XOR i NOT.
- Una unitat que suma, que ens val per varies operacions: ADD, $ADDI$, ST, STB, LD i LDB. Ja que totes aquestes instruccions requereixen d'una suma, ja sigui per calular una adreça o per la operació suma.
- Una unitat de resta, per la instrucció SUB.
- Dues unitats de desplaçament, una amb extesió de signe i una altra sense.
- Cinc unitats de compariació, que ens permeten fer les operacions de: $CMPLT$, $CMPLE$, $CMPLTU$ i $CMPLEU$.
- Dues unitats que mantenim per poder executar les instruccions de $MOVI$ i $MOVHI$.
- Una unitat de multiplicació de 32 bits, que ens permet executar les instruccions de: MUL, $MULH$ i $MULHU$.
- Una Unitat de divisió que ens permet executar les instruccions de: DIV i $DIVU$.

4.4 Unitat de control

L'unitat de control ha de donar suport a les noves instruccions. Per tal de poder-ho fer l'ALU té una codificació interna de 0 a $n-1$, on n és el número de instruccions que comptem. Aquesta manera ve inspirada dels processadors RISC i ens permet amb facilitat escollir quina instrucció ha d'executar l'ALU i posteriorment ens permetrà ampliar la ISA.

4.4.1 Lògica de control

Per poder implementar les noves instruccions, ens hem allunyat de l'implementaciò que fa un processador SISA i ens hem apropat més a l'implementació en un processador RISC.

Codi operació	**Instrucció**	**Codi operació**	**Instrucció**
00000	MOVI	01010	CMPLT
00001	MOVHI	01011	CMPLE
00010	AND	01100	CMPEQ
00011	OR	01101	CMPLTU
00100	XOR	01110	CMPLEU
00101	NOT	01111	MUL
00110	ADD, ADDI, LD, LDB, ST i STB	10000	MULH
00111	SUB	10001	MULHU
01000	SHA	10010	DIV
01001	SHL	10011	DIVU

Taula 4.5: Nous codis d'operació de les instruccions per la ALU.

També s'han fet canvis en algunes senyals:

- *op* ara és de 5 bits per poder codificar les instruccios per la ALU.
- S'afegeix el senyal *Rb_N* que ens permet selccionar si per l'entra y de la ALU entra el registre *Rb* o l'immediat. Aquest senyal serà 1 amb les instruccions: $ADDI$, LD, ST, $MOVI$, $MOVHI$, LDB i STB.
- La resta de senyals es manten igual que en l'etapa 2.1 (miri la seccio 2.5.2)

5

Etapa 4: Instruccions de salt

5.1 Introducció

En aquesta etapa donarem suport al es instrccions de salt: BZ, BNZ, JMP, JZ, JNZ i JAL. Que alteraran com calculàvem el PC fins ara.

Comptem amb 4 blocs principals:

- Banc de registres: Es manté el de l'etapa 2.1 (miri secció 2.3).
- ALU: es capaç de valorar si s'ha de fer el salt o no.
- Control: la unitat de ocntrol ha de ser capaç de calcular el nou PC i saltar-hi en cas d'una instrucció de salt.
- Memòria: Es manté l'implementació de l'etapa 2.2 (miri secció 3.2.1).

5.2 Instruccions

En aquesta versió l'ALU es manté la compatibilitat per les instrccions implementades en l'etapa 3 (miri la secció 4.2) i s'afegeix suport per les instruccions de salt.

5.2.1 Instruccions de salt condicional

15 14 13 12	11 10 9	8	7 6 5 4 3 2 1 0		
0 1 1 0	Rb	0	n n n n n n n n	Salt condicional relatiu al PC	BZ
		1			BNZ

Taula 5.1: Codificació de les operacions *BZ* i *BNZ* en lleguatge màquina.

BZ

Salt condicional. Es salta a $PC + immed$ si *Rb* és igual a zero.

BNZ

Salt condicional. Es salta a $PC + immed$ si *Rb* és diferent a zero.

5.2.2 Instruccions de ruptura de seqüènciament

Dins de la següent codificació es selecciona l'operació concreta amb els tres bits de funció *f f f*.

15 14 13 12	11 10 9	8 6 7	5 4 3 2 1 0		
0 1 1 0	Rb	Ra	0 0 0 f f f	Ruptura de seqüenciament	JZ, JNZ
	0 0 0	Ra			-, JMP
	Rd	Ra			JAL

Taula 5.2: Codificació de les operacions de ruptura de seqüènciament en lleguatge màquina.

JZ

Codi d'operació 000.

Salt condicional. Si *Rb* és igual a 0 es salta a l'adreça *Ra*.

JNZ

Codi d'operació 001.

Salt condicional. Si *Rb* és diferent a 0 es salta a l'adreça *Ra*.

JMP

Codi d'operació 011.

Salt incondicional a l'adreça *Ra*.

JAL

Codi d'operació 100.

Salt incondicional a l'adreça *Ra*, es guarda l'adreça de retorn a *Rd*.

5.3 ALU

Es manté la compatibilitat per les instruccions que haviem vist anteriorment (miri la secció 4.3). S'afegiex una senyal z que ens indica si el contingut del registre *Rb* és 0. D'aquesta manera avaluem si s'ha d'efectuar el salt o no.

La senyal z es calcula amb una operació XOR de tots el bits del registre *Rb*.

5.4 Unitat de control

La unitat de control ha de patir uns petits canvis ja que és l'encarregada de carregar el nou PC. On abans $PC = PC + 2$ ó $PC = PC$ depenent de la instrucció, ara el PC pot tenir uns altres valors.

Depenent del tipus de instrucció i de si la ALU calcula que s'ha de pendre el salt el PC tindrà uin valor o un altre. S'afegeix un senyal anomenat *tknbr* de dos bits.

El bit de més pes de *tknbr* ens indica si el salt s'ha de fer. El bit de menys pes ens indica quin PC s'ha de carregar.

tknbr	**PC nou**	**tknbr**	**PC nou**
00	$PC + 2$ ó PC	10	PC_{ALU}
01	$PC + 2$ ó PC	11	$PC + immed$

Taula 5.3: Nou PC en funció de la senyal *tknbr*.

5.4.1 Lògica de control

En el cas de voler executar un *JAL* s'ha d'habilitar el permís d'escriptura al banc de registres per tal de poder guardar el PC en el registre *Rd*.

S'ha afegit la codificació de les noves instruccions de salt per a la ALU:

op ALU	**Instrucció**	**op ALU**	**Instrucció**
10100	BZ	10111	JNZ
10101	BNZ	11000	JMP
10110	JZ	11001	JAL

Taula 5.4: Codificació de les instruccions per la ALU