UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

ORGANIZACIÓN Y ARQUITECTURA DE COMPUTADORAS

PROYECTO FINAL

ROMERO ANDRADE VICENTE

29 DE ENERO DE 2021

Índice

Introducción

Durante este curso se vieron temas de arquitectura de computadoras, estos tienen la finalidad de poder diseñar hardware como lo son un procesador que sirva para un propósito general y específico según sea el caso.

Para este curso se tomó como base el microprocesador M68HC11 el cual forma parte de la familia de microcontroladores de Motorola derivados de los famosos 6800. Este microcontrolador en su diseño original tiene una arquitectura von neumann, además de ser del tipo CISC con instrucciones de longitud variable compatibles con los derivados 6800.

Para este proyecto se hará uso de 2 versiones clónicas de este microcontrolador, el primero en su versión CISC y el segundo en su versión RISC. Para esto se harán uso de diferentes conceptos vistos a lo largo de curso y se intentará implementar las instrucciones para el manejo de interrupciones que se encuentran el libro de texto de la materia, además de que para el caso de RISC se desarrollarán las unidades de detección y anticipación para resolver problemas de accesos múltiples a memoria.

Se hará uso del programa Quartus en su versión 20.x tanto para el desarrollo del código como para las simulaciones.

Descripción del programa

El programa a desarrollar es el siguiente:
Programa que recorre un vector de números en 8 bits, suma los impares y resta los pares guardando el resultado en el índice X.
El algoritmo es el siguiente:

- Definir inicio vector contenido en la memoria
- Definir fin del vector contenido en la memoria
- Asignar localidad de memoria que guardara resultado de la operación del programa
- Asignar acumulador A para guardar suma de los impares
- Asignar acumulador B para guardar suma de los pares
- Recorrer vector en memoria
    - ¿La Localidad en memoria es par?
        - Restar contenido y guardar en acumulador B
    - ¿La Localidad en memoria es impar?
        - Sumar contenido y guardar en acumulador A
    - ¿El apuntador es igual al final del vector?
        - Volver al inicio de recorrer
- Sumar acumulador A y B y guardar contenido en índice X.

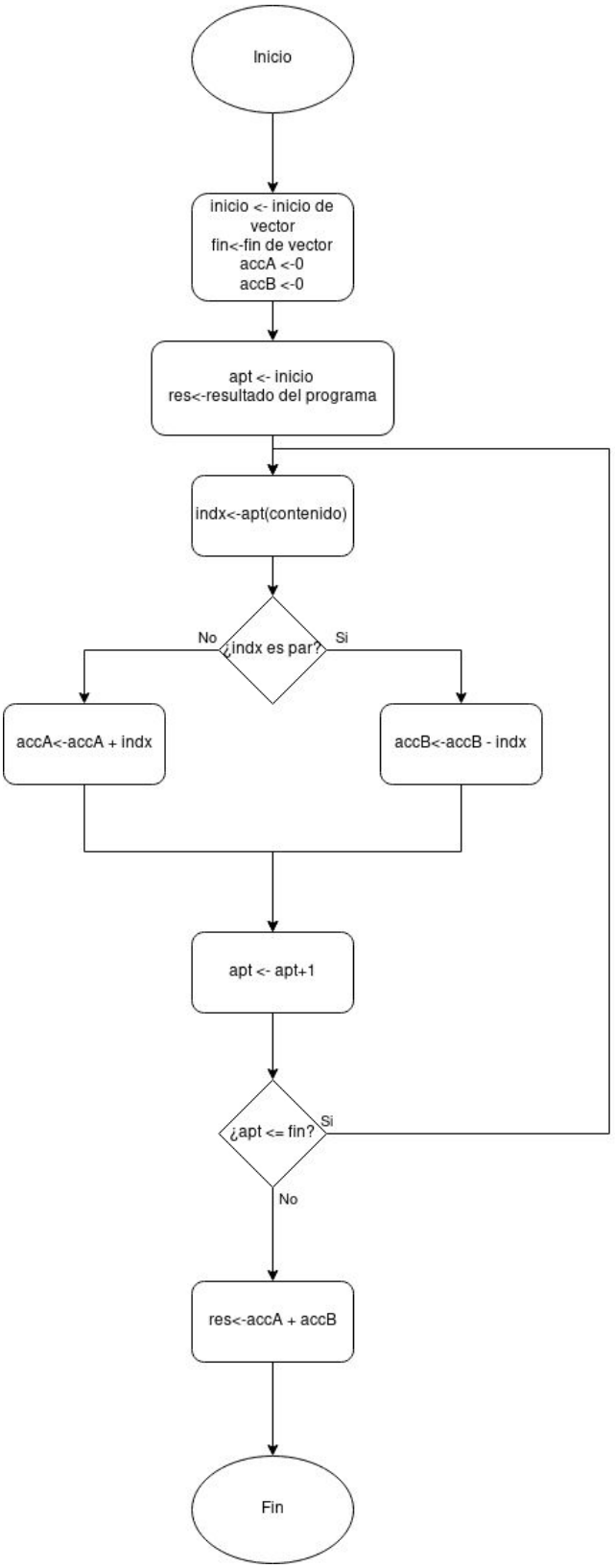


Diagrama de flujo del programa propuesto

Desarrollo

Para el desarrollo de esta práctica se realizará primero el programa en lenguaje ensamblador, este servirá para el análisis de las instrucciones que se utilizaran y se van a implementar en la arquitectura.

Programa ASM

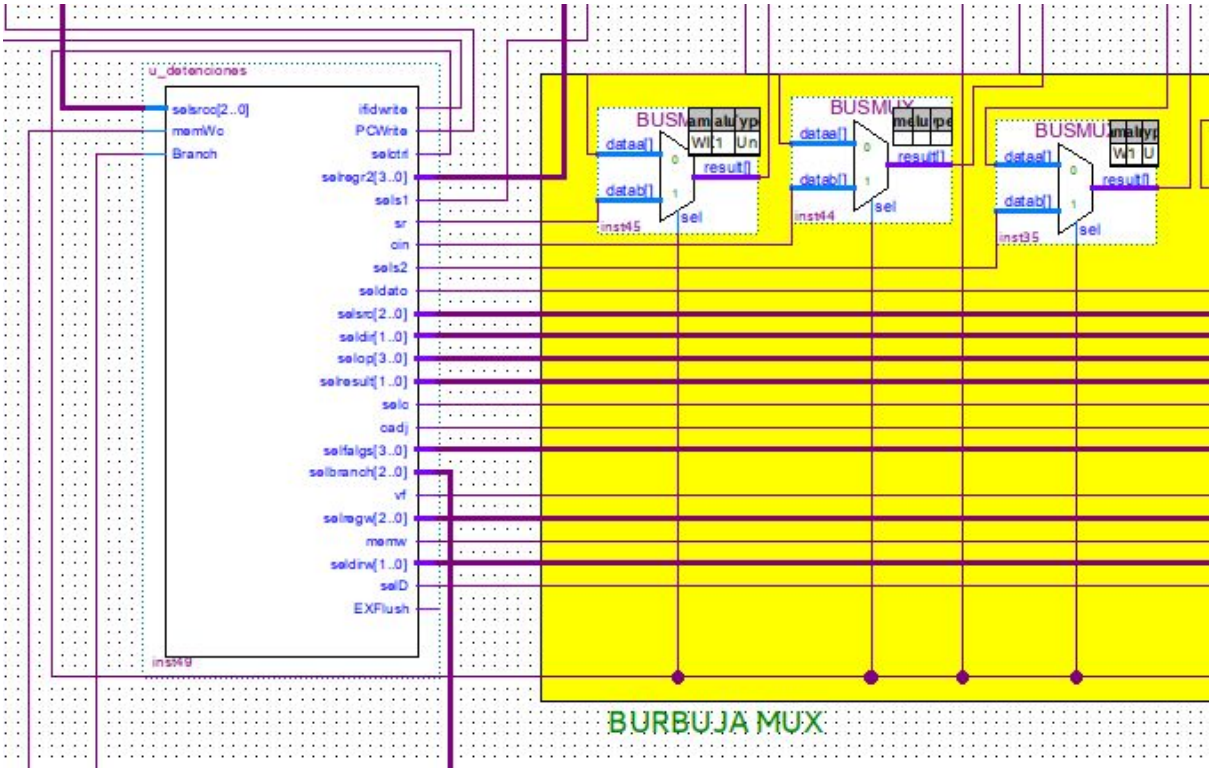
```
vector  db  02, 04, 05, 01, 02, 06, 07, 09  ;La suma de este vector
debe ser 9
fin     db  1;

.inicio
      LDAA  0x00   ;Asignar cero al acumulador A
      LDAB  0xFF   ;Asignar FF al acumulador B
      LDX   #(vector)  ;Este sera el apuntador y se le metera la
direccion inicial del vector
.rutina
      XPAR        ;Compara el contenido que apunta X, si c=1 es impar
si c = 0 es par
      BLO   3
      JMP   .par     ;Este ira a la rutina
      ADDA  0,X     ;Suma de numero impar y guardo en ACCA
      INX           ;Incrementa el apuntador
      CPX   #(fin)  ;fin del vector
      BEQ   3       ;Si IX == #(fin) entonces .suma
      JMP   .rutina
      JMP   .suma   ; Va a suma
.par
      SUBB  0,X   ;Resta de numero par y guardo en ACCB
      INX   ;Incrementa el apuntador
      CPX   #(fin) ;Comparacion con fin de vector
      BEQ   3      ;Si IX == #(fin) entonces .suma
      JMP .rutina ;Si el apuntador no es igual al fin de la rutina z!=0
entonces regreso a .rutina
.suma
      XABA   ; Suma de ACCA + ACCB y se guarda en X
      JMP .inicio  ;Loop infinito
```

Unidad de detenciones, registros internos y anticipaciones

Esta unidad tiene la finalidad de poder controlar los accesos múltiples a memoria, riesgos generados por saltos y señales de salida generadas por la unidad de detenciones.
Su tabla de verdad es la siguiente:

Condiciones de entrada				Señales de salida				
SelSrcs	EX/WB MemW	EX/WB Branch	EX/WB isBranch	PCWrite	IF/IDWrite	SelD	SelCtrl	EXFlush
2,4,5	1	0	0	0	0	1	1	0
No Importa	0	1	1	1	1	0	1	1
Combinación no presente en la tabla				1	1	0	0	0

Tabla de verdad de detenciones

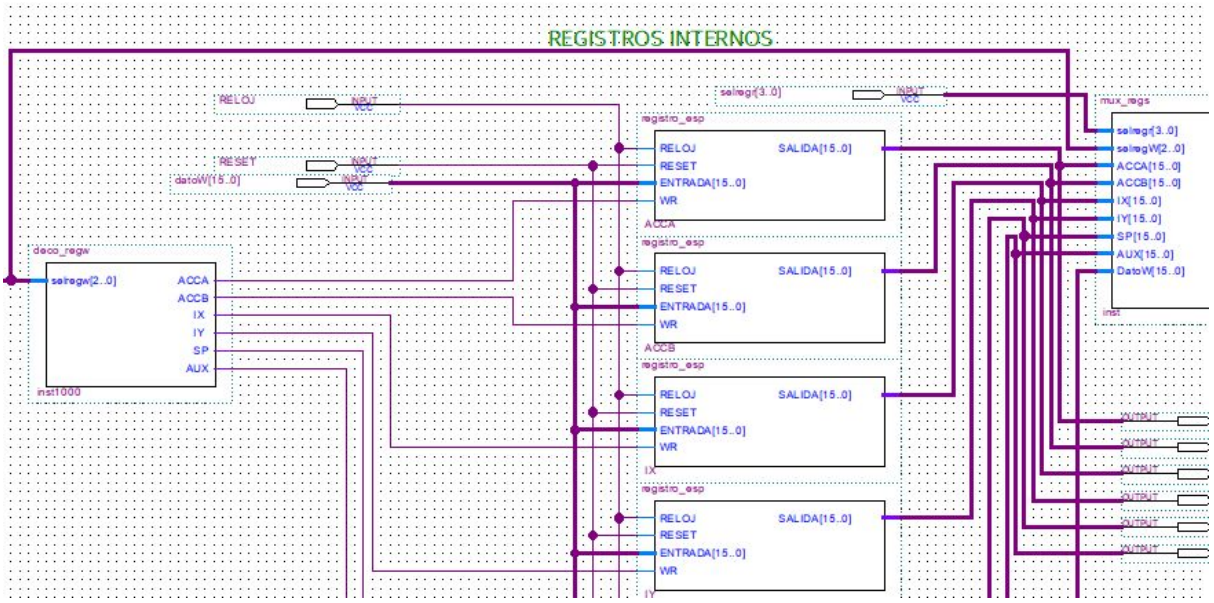


Unidad de detenciones

Condiciones de entrada		Señales de salida		Condiciones de entrada		Señales de salida	
SelRegR	EX/WB SelRegW	D1	D2	SelRegR	EX/WB SelRegW	D1	D2

1	1	DatoW	ACCB	9	2	0	DatoW
1	4	ACCA	DatoW	A	3	0	DatoW
2	4	DatoW	IX	B	6	0	DatoW
2	2	ACCB	DatoW	C	1	DatoW	SP
3	4	DatoW	IY	C	6	ACCA	DatoW
3	3	ACCB	DatoW	D	4	DatoW	SP
4	1	DatoW	0	D	6	ACCB	DatoW
5	4	DatoW	0	E	2	DatoW	SP
6	1	DatoW	IX	E	6	IX	DatoW
6	2	ACCA	DatoW	F	3	DatoW	SP
7	1	DatoW	IY	F	6	IY	DatoW
7	3	ACCA	DatoW				
8	5	DatoW	0				

Tabla de verdad Registros internos

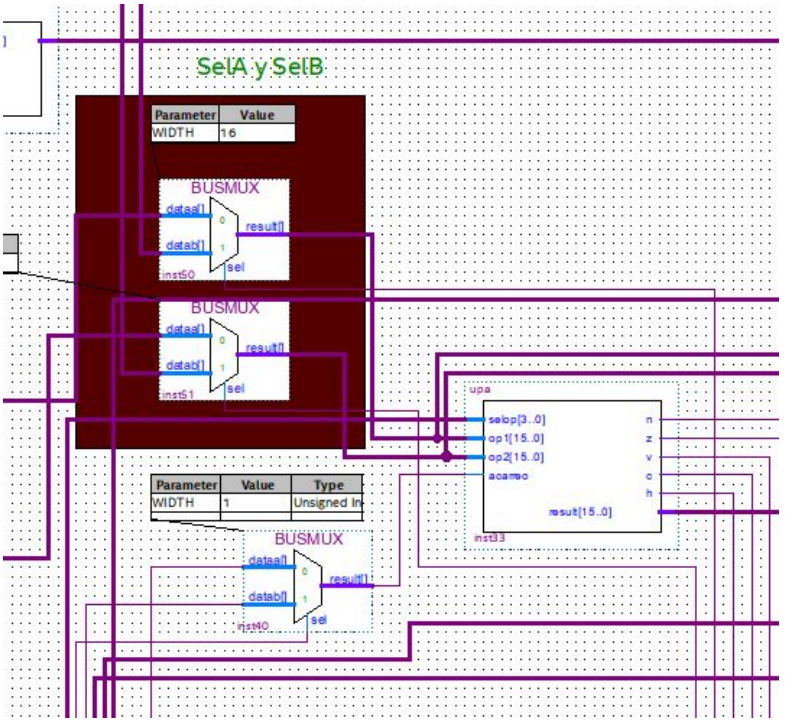


Logica de registros internos

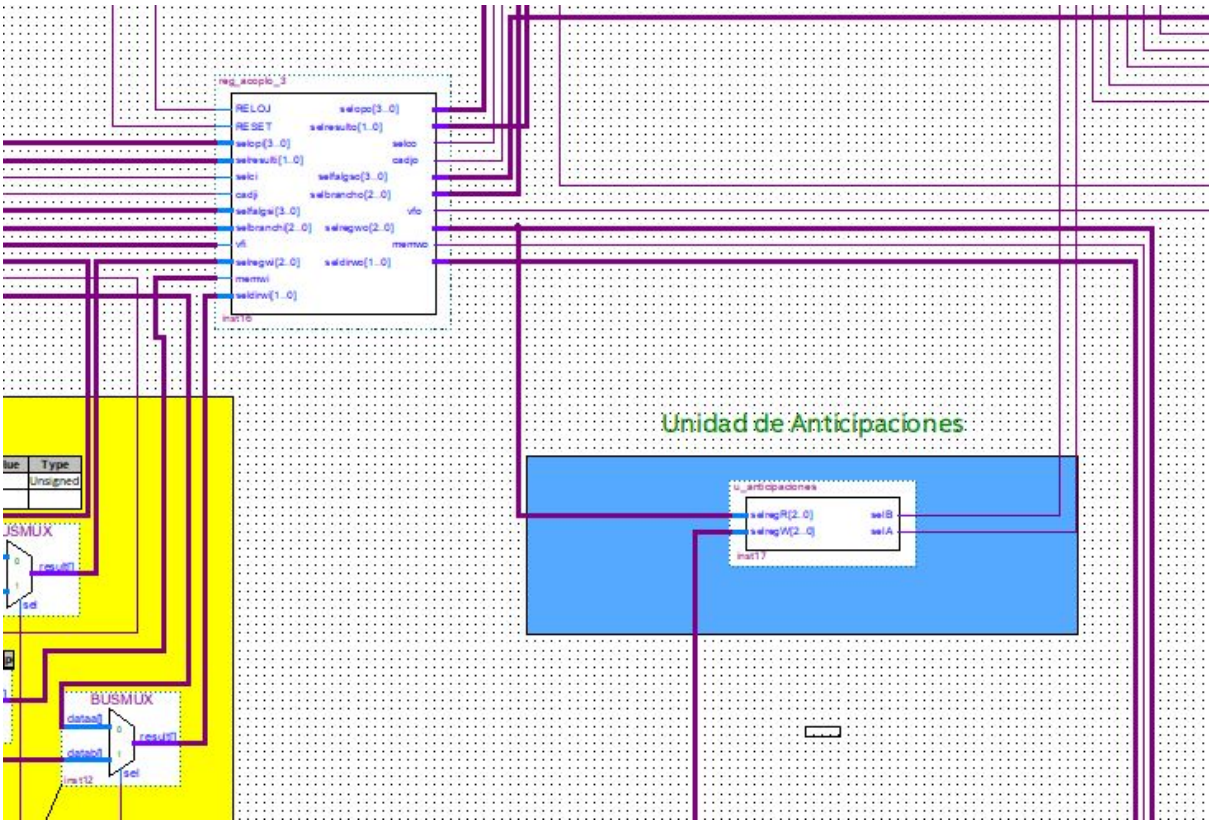
Condiciones de entrada		Señales de salida		Condiciones de entrada		Señales de salida	
SelRegR	EX/WB SelRegW	D1	D2	SelRegR	EX/WB SelRegW	D1	D2

1	1	1	0	9	2	0	1
1	4	0	1	A	3	0	1
2	4	1	0	B	6	0	1
2	2	0	1	C	1	1	0
3	4	1	0	C	6	0	1
3	3	0	1	D	4	1	0
4	1	1	0	D	6	0	1
5	4	1	0	E	2	1	0
6	1	1	0	E	6	0	1
6	2	0	1	F	3	1	0
7	1	1	0	F	6	0	1
7	3	0	1			0	0
8	5	1	0			0	0

Tabla de verdad unidad de anticipaciones



SelA y SelB antes de la UPA



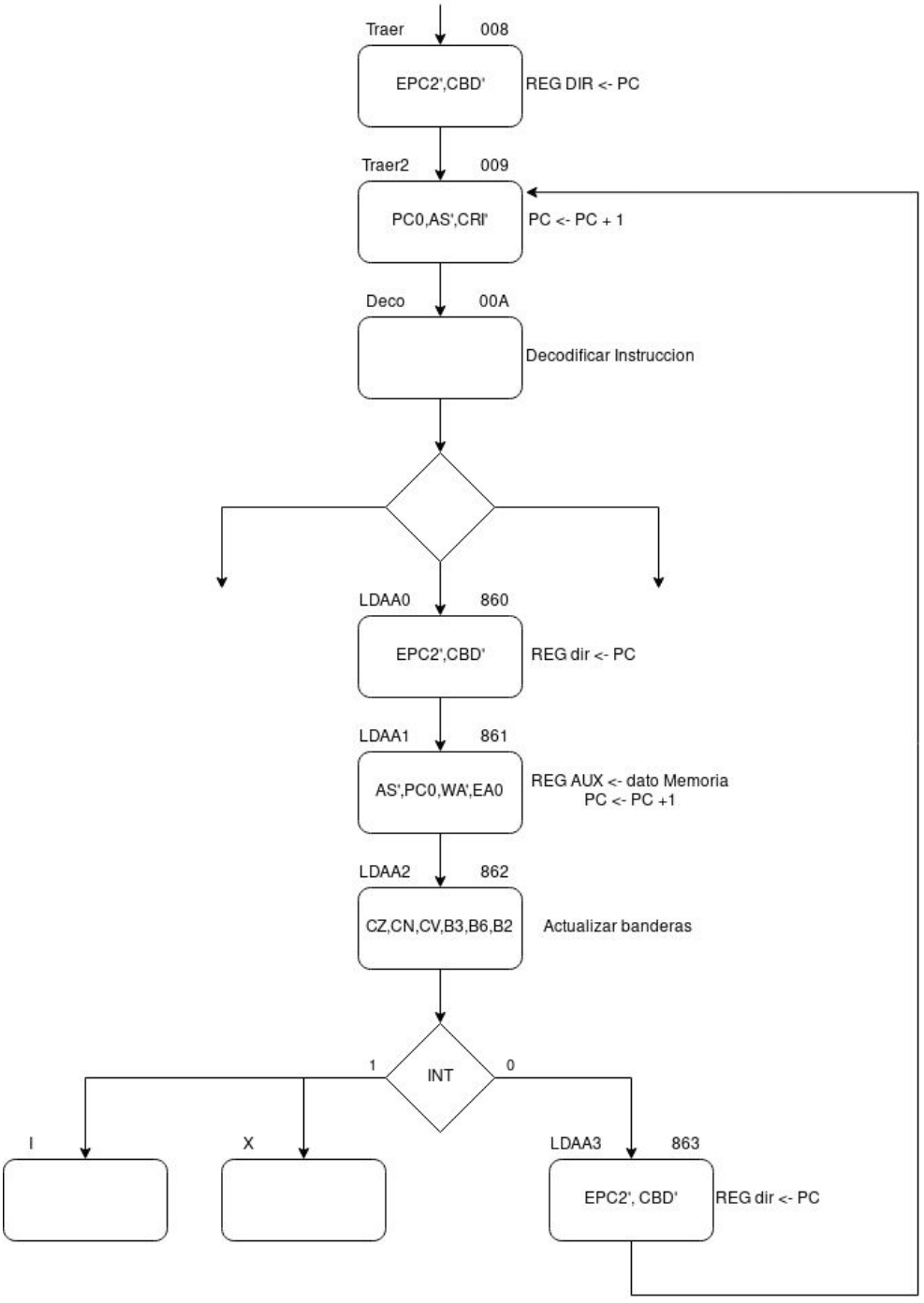
Unidad de Anticipaciones

Instrucciones

Para la construcción de las instrucciones de la arquitectura CISC, se usó un programa hecho en python que ayuda en la generación del código VHDL correspondiente usando solo las banderas y demás parámetros especiales.

LDAA(IMM)

CISC



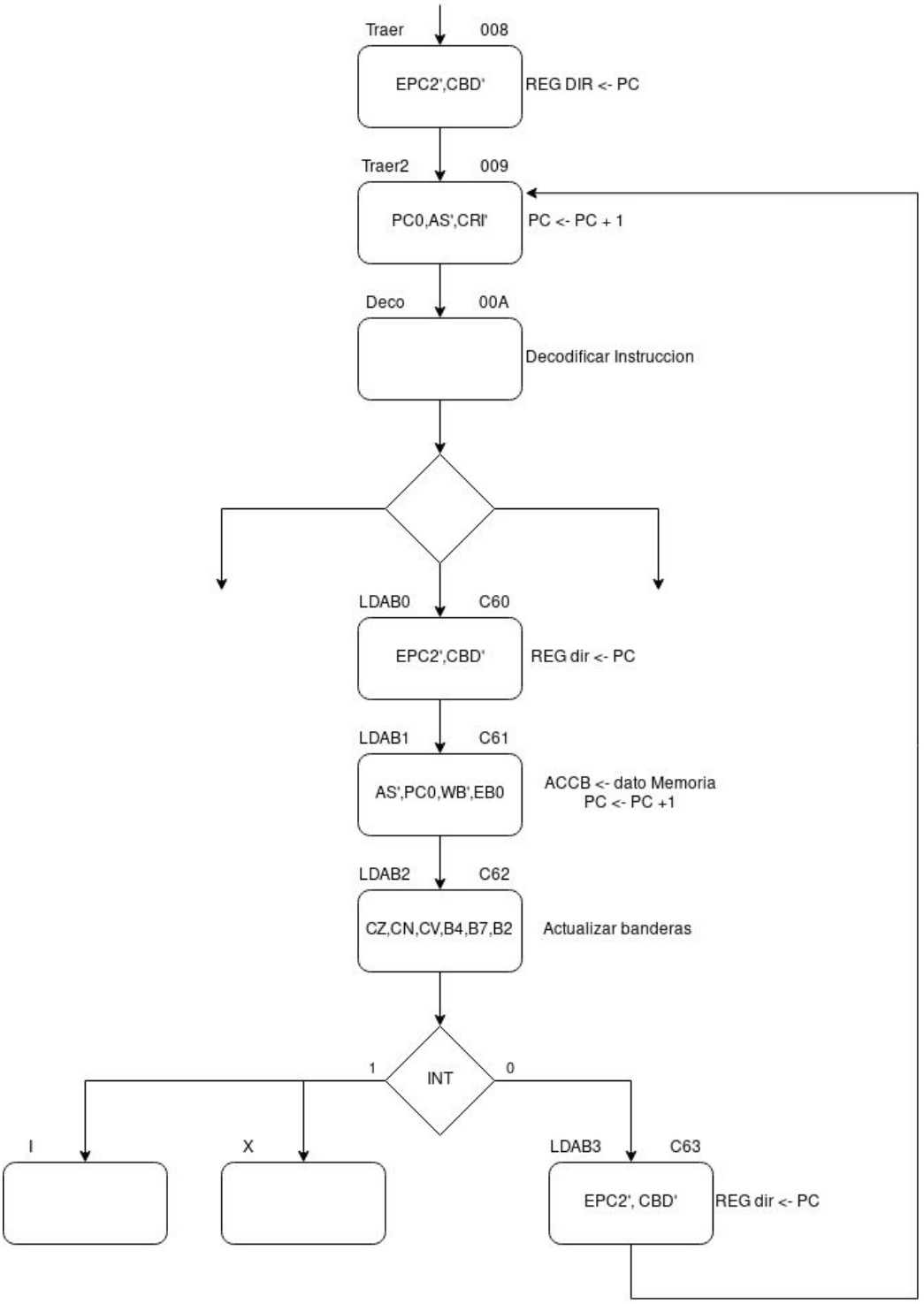
RISC

Etapa	Señal de control	Valor
Etapa 2	SelRegR	0
	SelS1	0
	Sr	1
	Cin	0
	SelS2	0
	SelDato	1
	SelScrs	3
	SelDir	0
Etapa 3	SelOp	4
	SelResult	1
	SelC	1
	Cadj	0
	SelFlags	1
	SelBranch	0
	VF	1
Etapa 4	SelRegW	1
	MemW	0
	SelDirW	0

En esta instrucción se hace uso del operador D1 y D5 para los cálculos en la UPA, esto es conveniente ya que D1 vale 0 (SelRegR=0) y D5 es el contenido inmediato contenido en la instrucción, se realiza un OR en la tercera etapa de la ejecución y este devuelve las banderas que se van a actualizar N, Z y V=0. Al final en la etapa 4 se guarda el resultado en ACCA

LDAB(IMM)

CISC

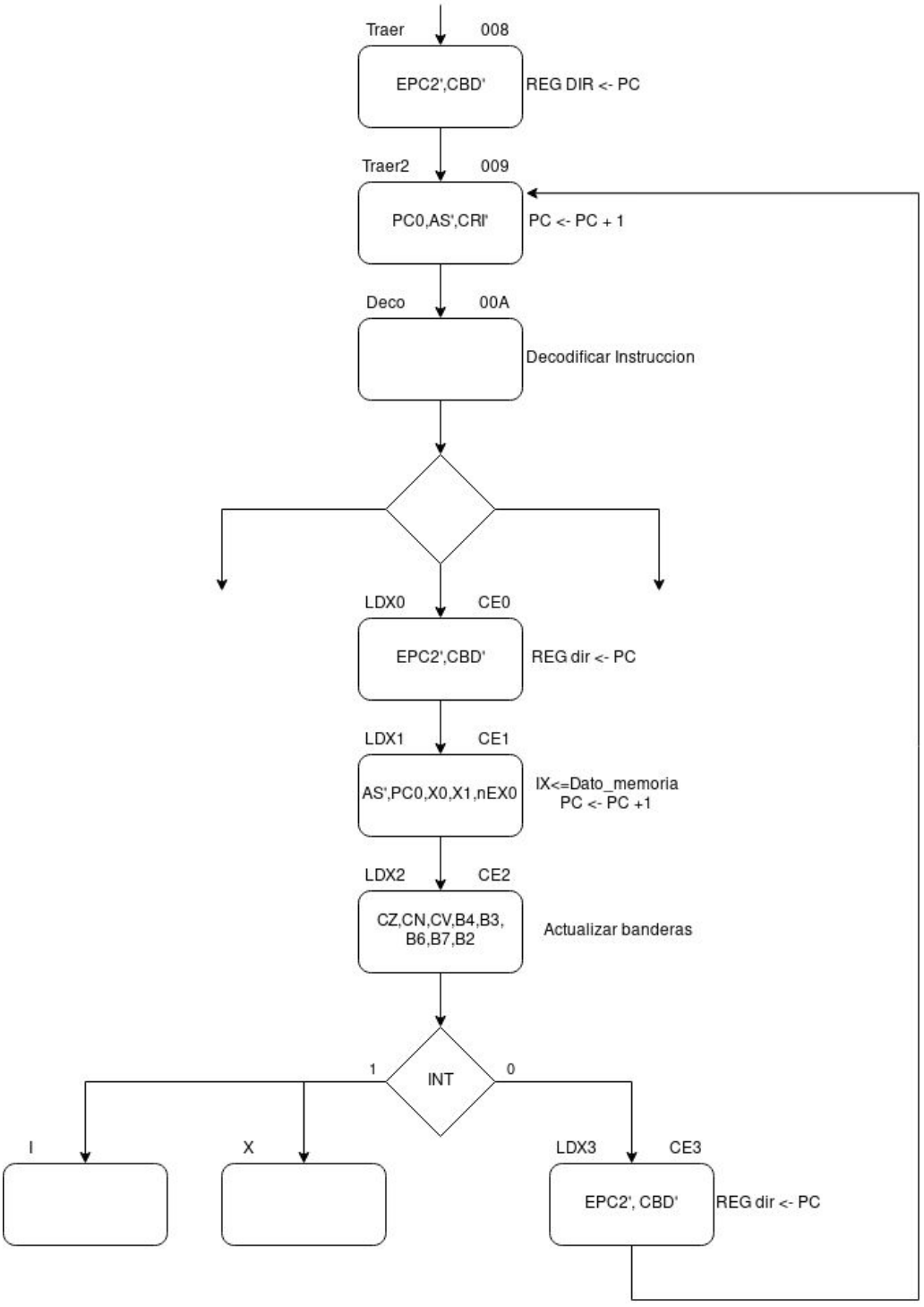


Etapa	Señal de control	Valor
Etapa 2	SelRegR	0
	SelS1	0
	Sr	1
	Cin	0
	SelS2	0
	SelDato	1
	SelScrs	3
	SelDir	0
Etapa 3	SelOp	4
	SelResult	1
	SelC	1
	Cadj	0
	SelFlags	1
	SelBranch	0
	VF	1
Etapa 4	SelRegW	4
	MemW	0
	SelDirW	0

Esta instrucción tiene el mismo funcionamiento que la del ACCA(IMM) solo que SelRegW apunta a ACCB.

LDX(IMM)

CISC

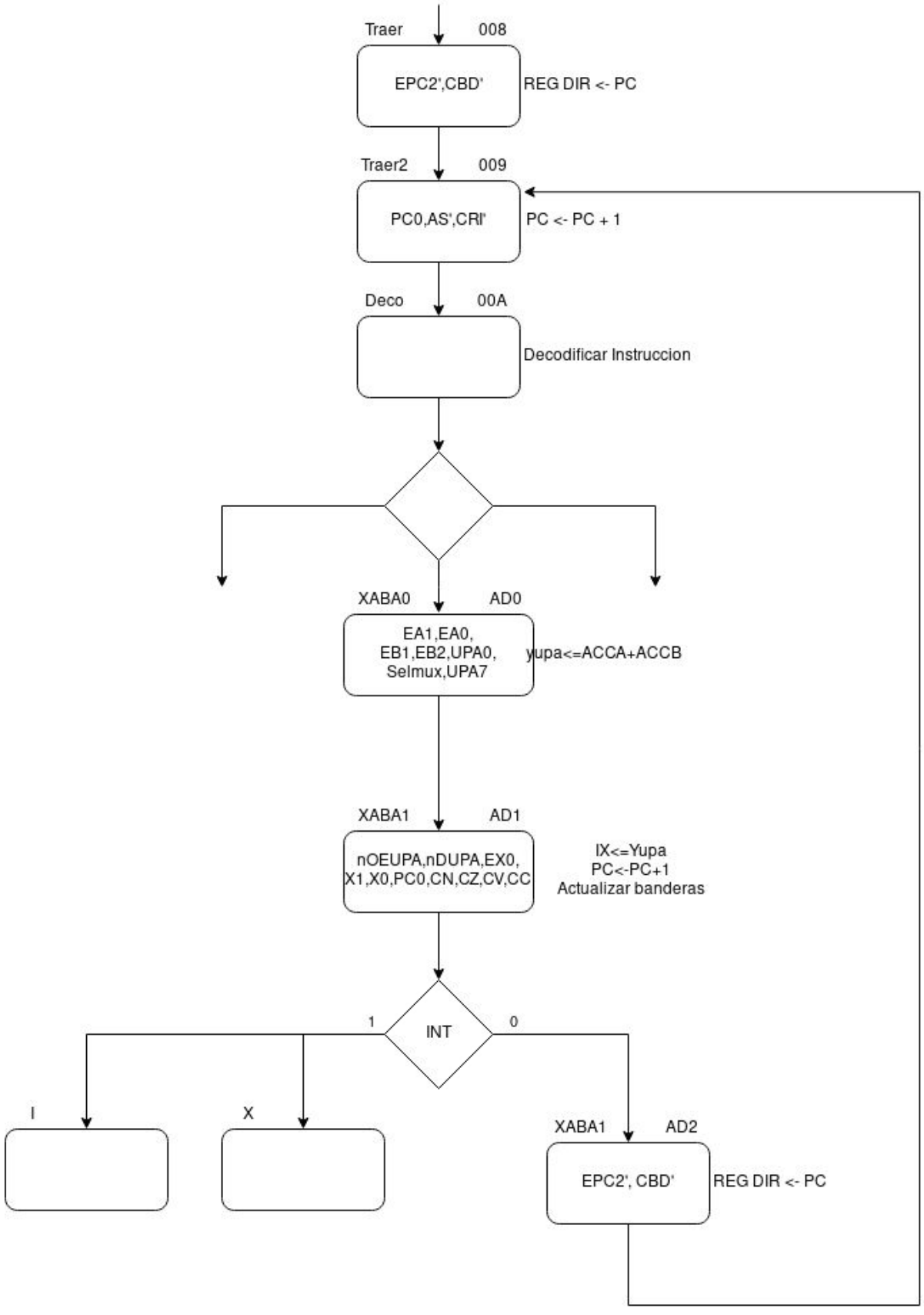

Traer
008
EPC2',CBD'
REG DIR <- PC
Traer2
009
PC0,AS',CRl'
PC <- PC + 1
Deco
00A
Decodificar Instruccion
LDX0
CE0
EPC2',CBD'
REG dir <- PC
LDX1
CE1
AS',PC0,X0,X1,nEX0
IX<=Dato_memoria
PC <- PC +1
LDX2
CE2
CZ,CN,CV,B4,B3,
B6,B7,B2
Actualizar banderas
1
INT
0
I
X
LDX3
CE3
EPC2', CBD'
REG dir <- PC

Etapa	Señal de control	Valor
Etapa 2	SelRegR	0
	SelS1	0
	Sr	1
	Cin	0
	SelS2	0
	SelDato	1
	SelScrs	3
	SelDir	0
Etapa 3	SelOp	4
	SelResult	1
	SelC	1
	Cadj	0
	SelFlags	1
	SelBranch	0
	VF	1
Etapa 4	SelRegW	2
	MemW	0
	SelDirW	0

Esta instrucción tiene el mismo funcionamiento que la del ACCA(IMM) solo que SelRegW apunta a IX.

XABA(INH)

Instrucción que suma lo que está en ACCA y ACCB y lo guarda en el índice X.
IX<- ACCA + ACCB


Traer
008
EPC2',CBD'
REG DIR <- PC
Traer2
009
PC0,AS',CRI'
PC <- PC + 1
Deco
00A
Decodificar Instruccion
XABA0
AD0
EA1,EA0,
EB1,EB2,UPA0,
Selmux,UPA7
yupa<=ACCA+ACCB
XABA1
AD1
nOEUPA,nDUPA,EX0,
X1,X0,PC0,CN,CZ,CV,CC
IX<=Yupa
PC<-PC+1
Actualizar banderas
1
INT
0
I
X
XABA1
AD2
EPC2', CBD'
REG DIR <- PC

RISC

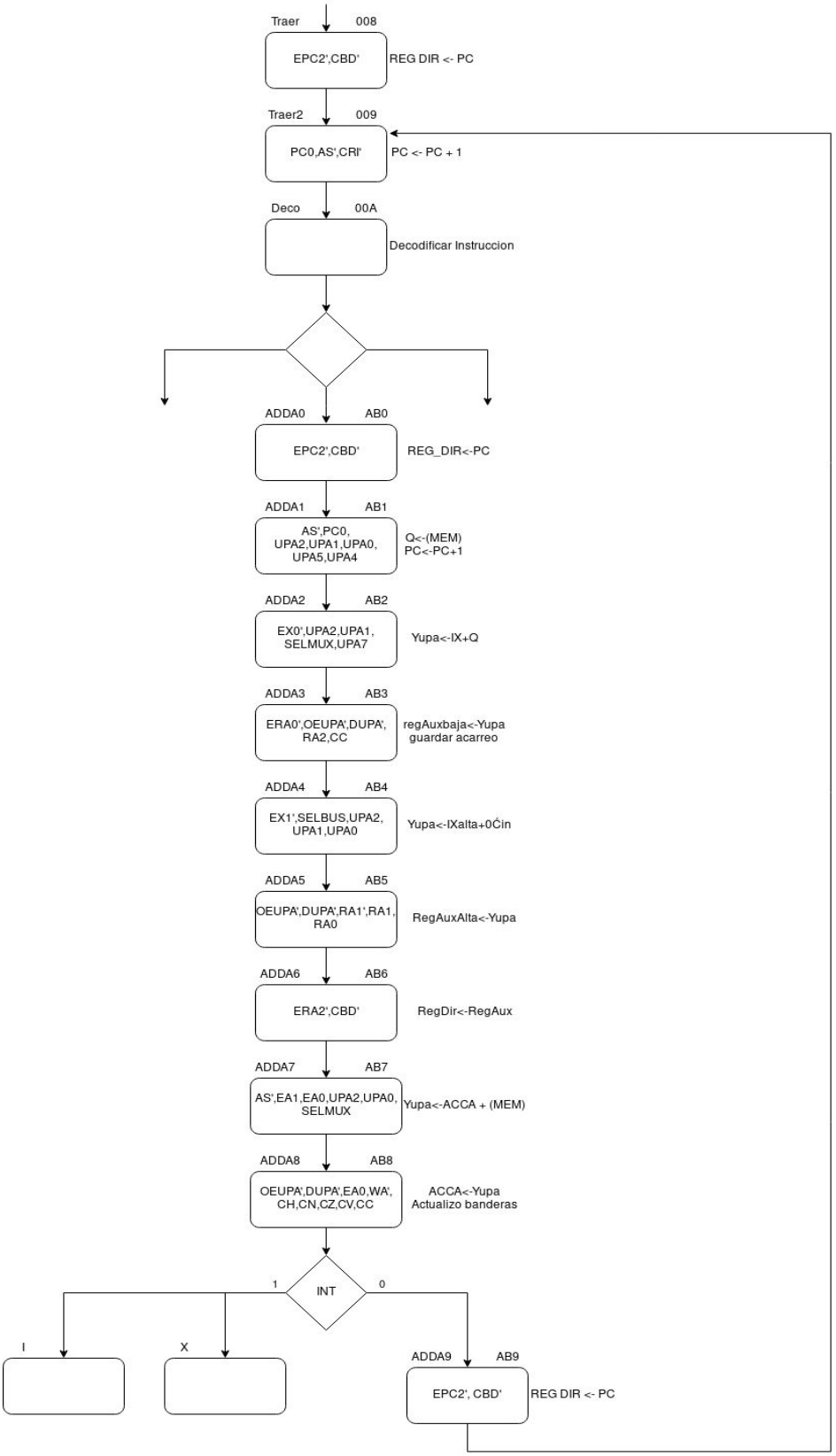
Etapa	Señal de control	Valor
Etapa 2	SelRegR	1
	SelS1	0
	Sr	1
	Cin	0
	SelS2	0
	SelDato	1
	SelScrs	1
	SelDir	0
Etapa 3	SelOp	1
	SelResult	1
	SelC	1
	Cadj	0
	SelFlags	2
	SelBranch	0
	VF	1
Etapa 4	SelRegW	2
	MemW	0
	SelDirW	0

Con SelRegR=1 se leen los registro ACCA y ACCB, por defecto D1 y D2 son los operandos seleccionados anteriormente, para esto se selecciona SelScrs=1 para que sean usados en la etapa de ejecución. En la etapa de ejecución los operandos son procesador por la UPA la cual se hace la operación Suma con el acarreo, pero como esta suma no debe tener variaciones no deseadas se fuerza a que el acarreo sea 0 y para esto se seleccion que el acarreo venga de Cadj=0 con SelC=1, se procesan las banderas correspondientes al CCR y se evita el branch. En la etapa de ejecución se guarda el resultado en IX.

ADDA(ind,x)

ACCA <= ACCA + (Memoria)

CISC

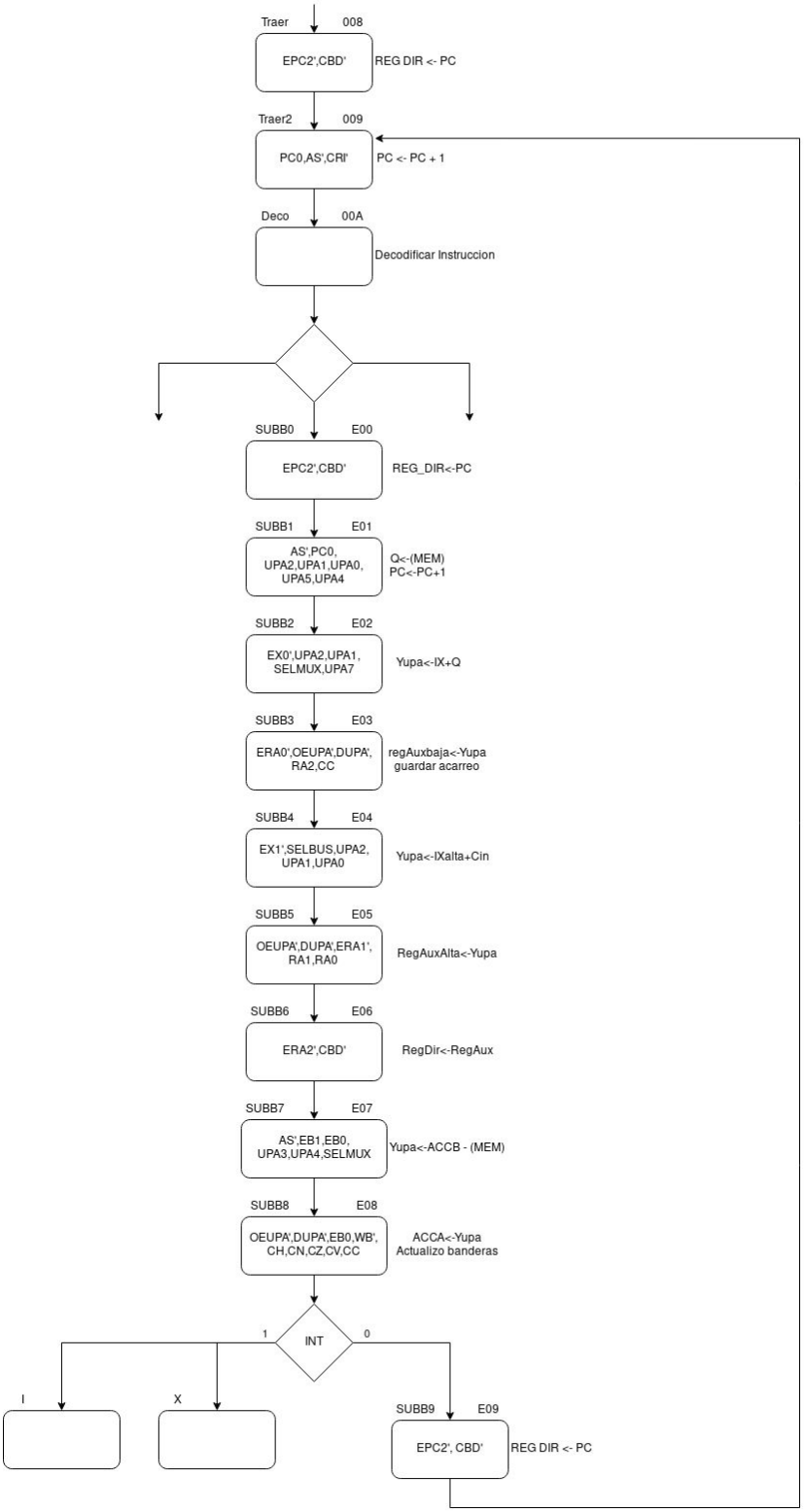

Traer
008
EPC2',CBD'
REG DIR <- PC
Traer2
009
PC0,AS',CRl'
PC <- PC + 1
Deco
00A
Decodificar Instruccion
ADDA0
AB0
EPC2',CBD'
REG_DIR<-PC
ADDA1
AB1
AS',PC0,
UPA2,UPA1,UPA0,
UPA5,UPA4
Q<-(MEM)
PC<-PC+1
ADDA2
AB2
EX0',UPA2,UPA1,
SELMUX,UPA7
Yupa<-IX+Q
ADDA3
AB3
ERA0',OEUPA',DUPA',
RA2,CC
regAuxbaja<-Yupa
guardar acarreo
ADDA4
AB4
EX1',SELBUS,UPA2,
UPA1,UPA0
Yupa<-IXalta+0Cin
ADDA5
AB5
OEUPA',DUPA',RA1',RA1,
RA0
RegAuxAlta<-Yupa
ADDA6
AB6
ERA2',CBD'
RegDir<-RegAux
ADDA7
AB7
AS',EA1,EA0,UPA2,UPA0,
SELMUX
Yupa<-ACCA + (MEM)
ADDA8
AB8
OEUPA',DUPA',EA0,WA',
CH,CN,CZ,CV,CC
ACCA<-Yupa
Actualizo banderas
INT
1
0
I
X
ADDA9
AB9
EPC2', CBD'
REG DIR <- PC

Etapa	Señal de control	Valor
Etapa 2	SelRegR	6
	SelS1	1
	Sr	1
	Cin	0
	SelS2	0
	SelDato	1
	SelScrs	2
	SelDir	0
Etapa 3	SelOp	1
	SelResult	1
	SelC	1
	Cadj	0
	SelFlags	2
	SelBranch	0
	VF	1
Etapa 4	SelRegW	1
	MemW	0
	SelDirW	0

Como esta instrucción usa direccionamiento indexado, es necesario que se procese la suma del contenido de IX(SelRegR=6) con el desplazamiento y se obtenga el valor del ACCA, esta instrucción a diferencia de las anteriores hace uso del resultado del sumador/restador para calcular la dirección efectiva del dato que se va a leer en memoria, por esto se selecciona S1 para que pase la dirección de memoria con el contenido de D2 (IX), para que sea suma se pone en el S/R = 1 y selDir =0 para procesar exitosamente el la dirección de memoria. Las etapas que siguen son el mismo proceso que se siguió en la instrucción de acceso inmediato. Para la etapa de ejecución se guarda el resultado en ACCA.

SUBB(ind,x)

CISC



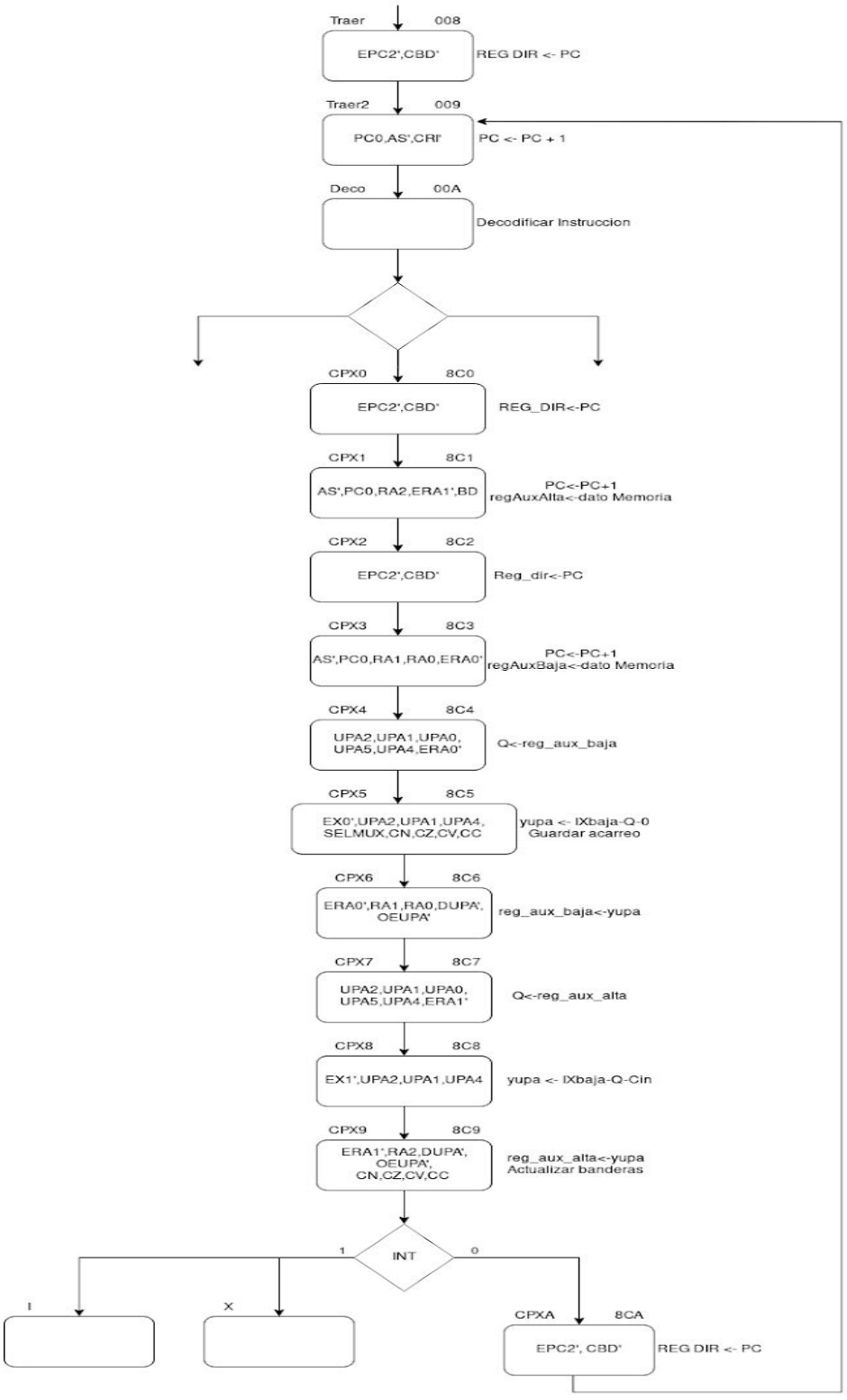
RISC

Etapa	Señal de control	Valor
Etapa 2	SelRegR	2
	SelS1	1
	Sr	1
	Cin	0
	SelS2	0
	SelDato	1
	SelScrs	2
	SelDir	0
Etapa 3	SelOp	2
	SelResult	1
	SelC	1
	Cadj	0
	SelFlags	2
	SelBranch	0
	VF	1
Etapa 4	SelRegW	4
	MemW	0
	SelDirW	0

CPX(EXT)

(IX)-(M:M+1)

CISC


Traer
008
EPC2',CBD'
REG DIR <- PC
Traer2
009
PC0,AS',CRI'
PC <- PC + 1
Deco
00A
Decodificar Instruccion
CPX0
8C0
EPC2',CBD'
REG_DIR<-PC
CPX1
8C1
AS',PC0,RA2,ERA1',BD
PC<-PC+1
regAuxAlta<-dato Memoria
CPX2
8C2
EPC2',CBD'
Reg_dir<-PC
CPX3
8C3
AS',PC0,RA1,RA0,ERA0'
PC<-PC+1
regAuxBaja<-dato Memoria
CPX4
8C4
UPA2,UPA1,UPA0,
UPA5,UPA4,ERA0'
Q<-reg_aux_baja
CPX5
8C5
EX0',UPA2,UPA1,UPA4,
SELMUX,CN,CZ,CV,CC
yupa <- IXbaja-Q-0
Guardar acarreo
CPX6
8C6
ERA0',RA1,RA0,DUPA',
OEUPA'
reg_aux_baja<-yupa
CPX7
8C7
UPA2,UPA1,UPA0,
UPA5,UPA4,ERA1'
Q<-reg_aux_alta
CPX8
8C8
EX1',UPA2,UPA1,UPA4
yupa <- IXbaja-Q-Cin
CPX9
8C9
ERA1',RA2,DUPA',
OEUPA',
CN,CZ,CV,CC
reg_aux_alta<-yupa
Actualizar banderas
1
INT
0
I
X
CPXA
8CA
EPC2', CBD'
REG DIR <- PC

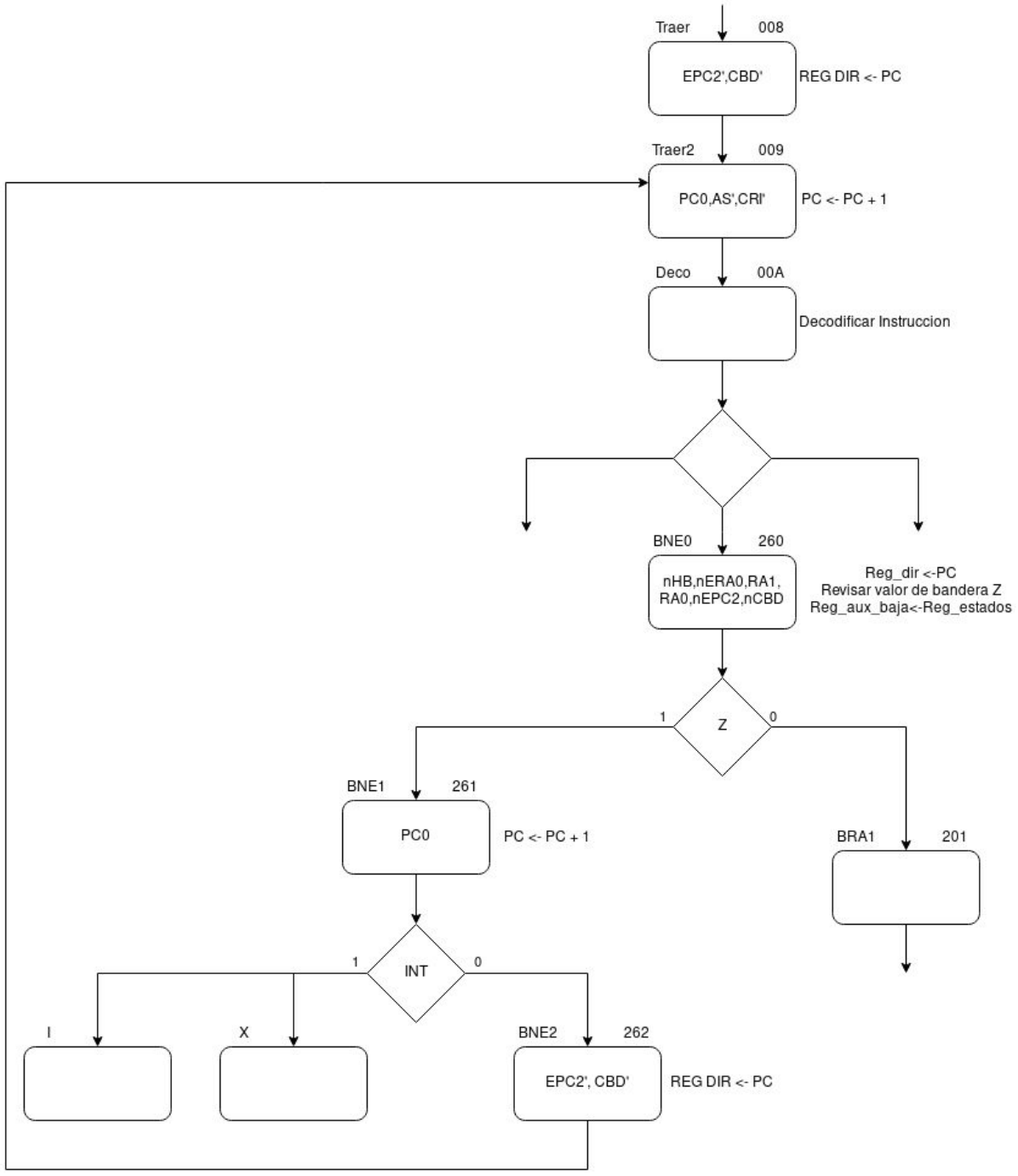
RISC

Etapa	Señal de control	Valor
Etapa 2	SelRegR	E
	SelS1	1
	Sr	0
	Cin	0
	SelS2	0
	SelDato	1
	SelScrs	3
	SelDir	1
Etapa 3	SelOp	2
	SelResult	0
	SelC	1
	Cadj	1
	SelFlags	3
	SelBranch	0
	VF	1
Etapa 4	SelRegW	0
	MemW	0
	SelDirW	0

BNE

PC<=PC+0x0002

CISC



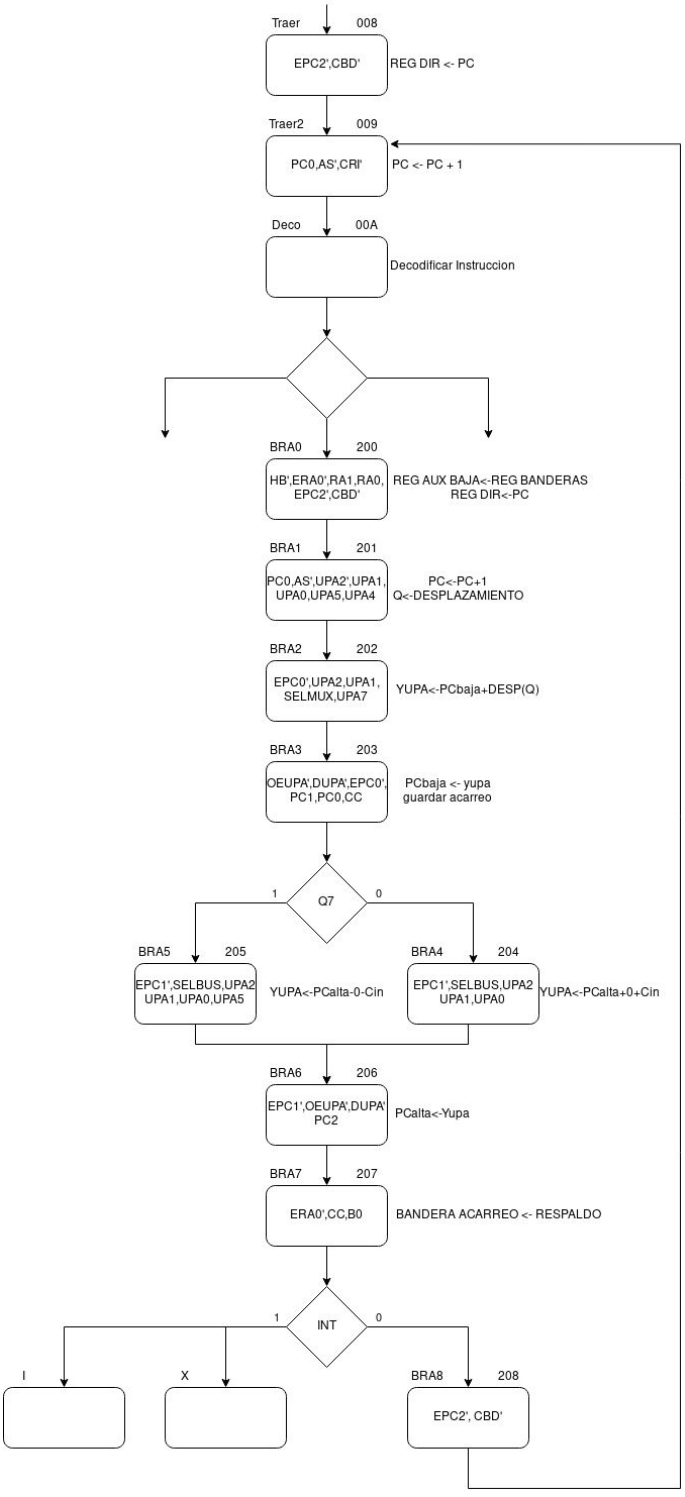
RISC

Etapa	Señal de control	Valor
Etapa 2	SelRegR	0
	SelS1	0

	Sr	1
	Cin	0
	SelS2	1
	SelDato	0
	SelScrs	5
	SelDir	0
Etapa 3	SelOp	1
	SelResult	1
	SelC	1
	Cadj	0
	SelFlags	0
	SelBranch	2
	VF	0
Etapa 4	SelRegW	0
	MemW	0
	SelDirW	0

BRA

CISC


Traer
008
EPC2',CBD'
REG DIR <- PC
Traer2
009
PC0,AS',CRI'
PC <- PC + 1
Deco
00A
Decodificar Instruccion
BRA0
200
HB',ERA0',RA1,RA0,
EPC2',CBD'
REG AUX BAJA<-REG BANDERAS
REG DIR<-PC
BRA1
201
PC0,AS',UPA2',UPA1,
UPA0,UPA5,UPA4
PC<-PC+1
Q<-DESPLAZAMIENTO
BRA2
202
EPC0',UPA2,UPA1,
SELMUX,UPA7
YUPA<-PCbaja+DESP(Q)
BRA3
203
OEUPA',DUPA',EPC0',
PC1,PC0,CC
PCbaja <- yupa
guardar acarreo
Q7
1
0
BRA5
205
EPC1',SELBUS,UPA2
UPA1,UPA0,UPA5
YUPA<-PCalta-0-Cin
BRA4
204
EPC1',SELBUS,UPA2
UPA1,UPA0
YUPA<-PCalta+0+Cin
BRA6
206
EPC1',OEUPA',DUPA'
PC2
PCalta<-Yupa
BRA7
207
ERA0',CC,B0
BANDERA ACARREO <- RESPALDO
INT
1
0
I
X
BRA8
208
EPC2', CBD'

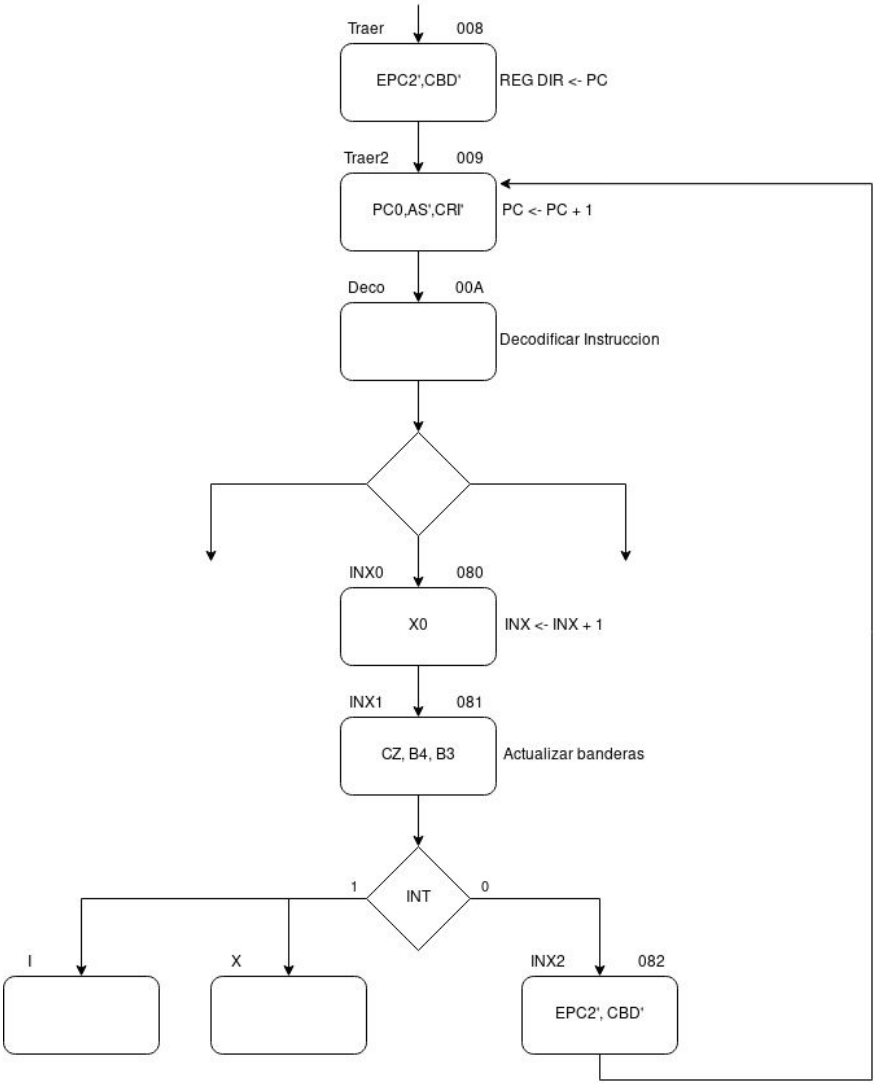
RISC

Etapa	Señal de control	Valor
Etapa 2	SelRegR	0
	SelS1	0
	Sr	1
	Cin	0
	SelS2	1
	SelDato	0
	SelScrs	5
	SelDir	0
Etapa 3	SelOp	1
	SelResult	1
	SelC	1
	Cadj	0
	SelFlags	0
	SelBranch	0
	VF	1
Etapa 4	SelRegW	0
	MemW	0
	SelDirW	0

INCX

IX <= IX+0x0001

CISC



RISC

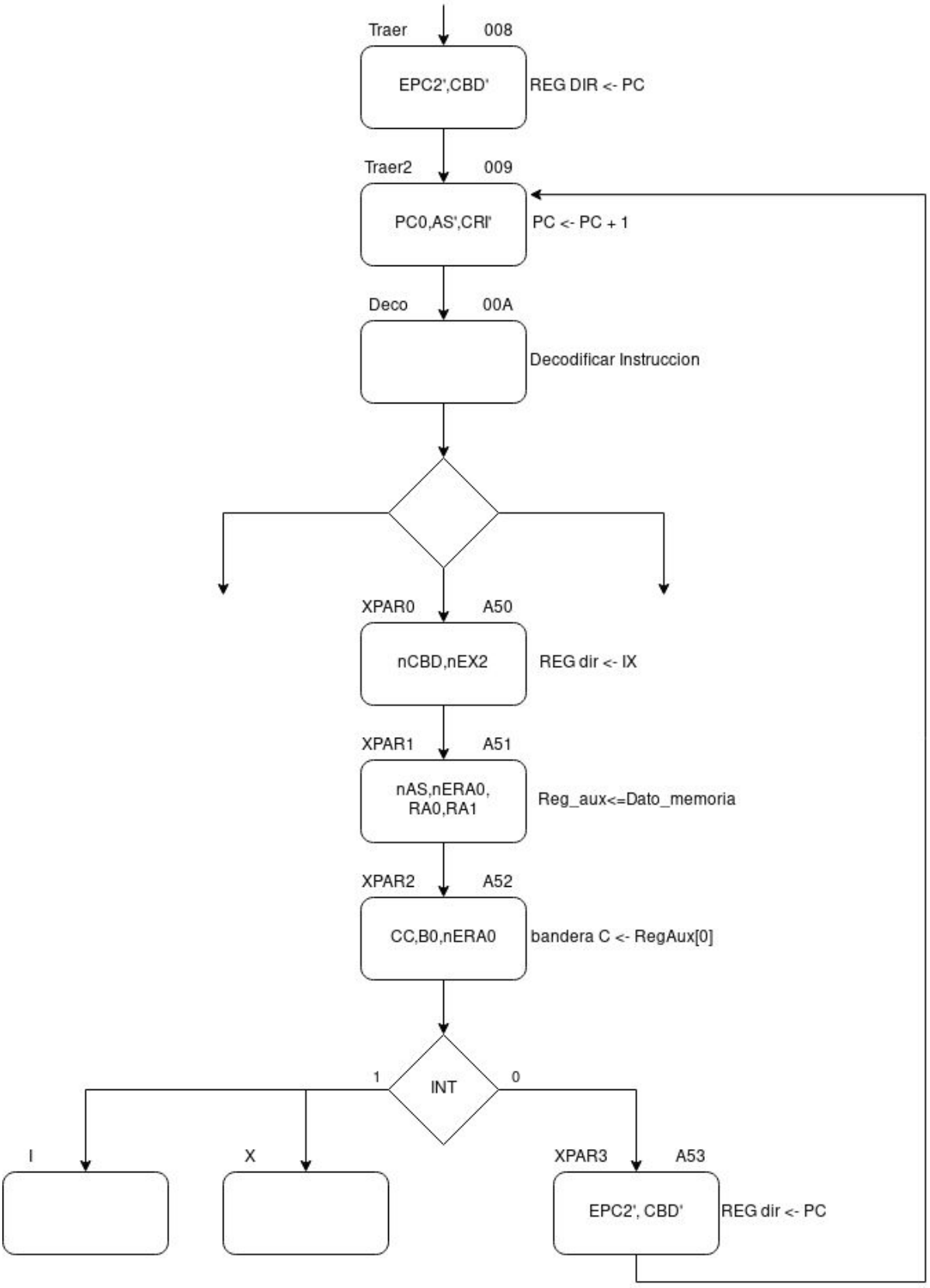
Etapa	Señal de control	Valor
Etapa 2	SelRegR	E
	SelS1	0
	Sr	1
	Cin	0
	SelS2	0
	SelDato	1
	SelScrs	3

	SelDir	0
Etapa 3	SelOp	1
	SelResult	1
	SelC	1
	Cadj	1
	SelFlags	C
	SelBranch	0
	VF	1
Etapa 4	SelRegW	2
	MemW	0
	SelDirW	0

Esta instrucción recibe el registro IX para su lectura y una dirección de memoria que siempre debe ser 0x0000, al hacer la suma se añade el acarreo Cadj=1 y SelC=1, al final se guarda el resultado en IX.

XPAR(ind,x)

Instrucción que defina si la localidad que apunta el índice X es par o no. Actualiza la bandera C el cual sirve para saber si es par o no por medio de otra instrucción.



Para esta instrucción se compara el valor del último bit del dato a comparar, el cual corresponde con la bandera C que está conectado al bus de datos interno.

RISC

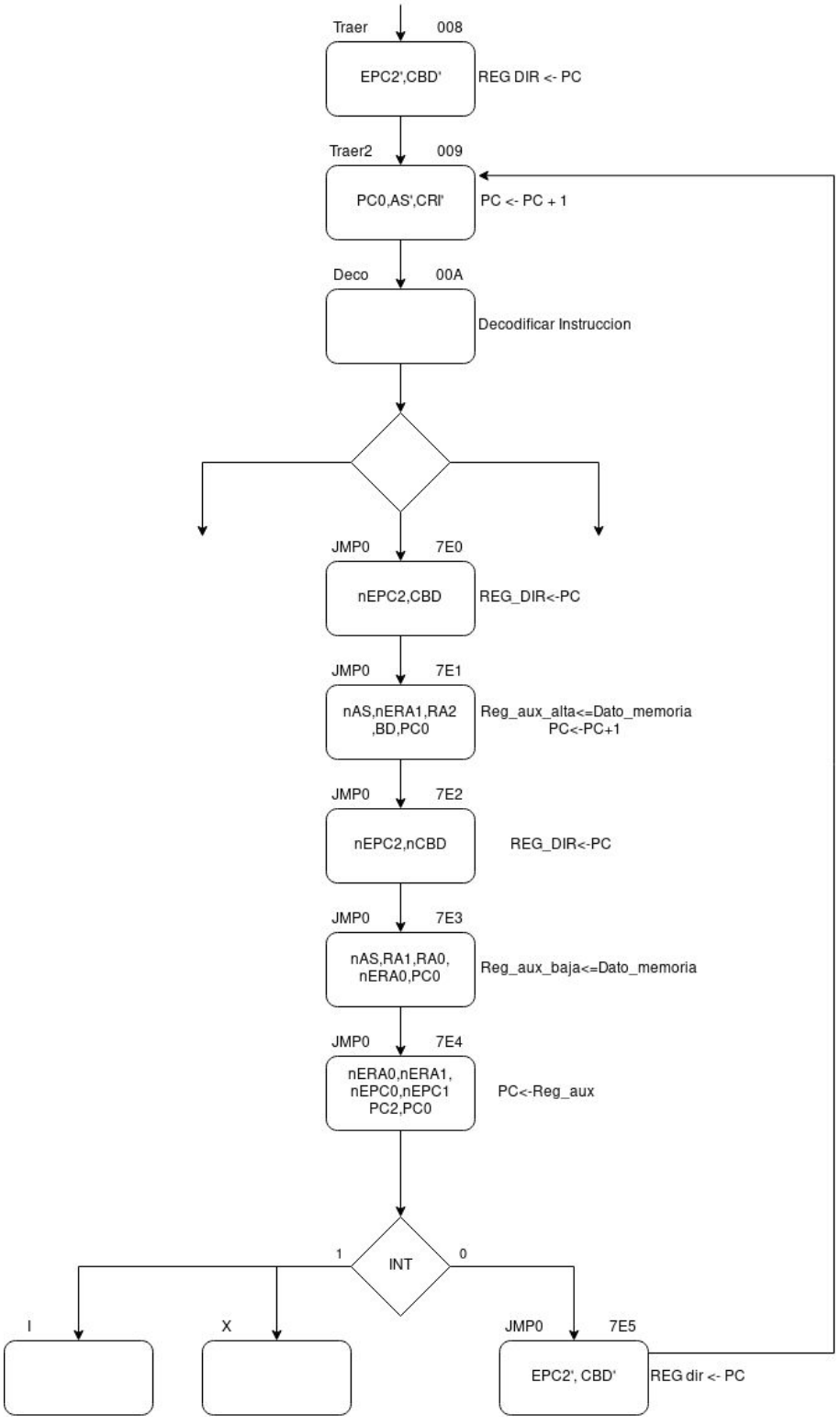
IX>>1
IX[0] = C

Etapa	Señal de control	Valor
Etapa 2	SelRegR	E
	SelS1	0
	Sr	1
	Cin	0
	SelS2	0
	SelDato	1
	SelScrs	4
	SelDir	0
Etapa 3	SelOp	9
	SelResult	0
	SelC	1
	Cadj	0
	SelFlags	4
	SelBranch	0
	VF	1
Etapa 4	SelRegW	0
	MemW	0
	SelDirW	0

Esta instrucción hace un corrimiento a la derecha quedando el bit que indica paridad en el valor de la bandera C, la cual es actualizada según el valor de este.

JMP(EXT)

CISC



RISC

Etapa	Señal de control	Valor
Etapa 2	SelRegR	0
	SelS1	0
	Sr	0
	Cin	0
	SelS2	0
	SelDato	0
	SelScrs	3
	SelDir	0
Etapa 3	SelOp	4
	SelResult	1
	SelC	0
	Cadj	0
	SelFlags	0
	SelBranch	0
	VF	1
Etapa 4	SelRegW	0
	MemW	0
	SelDirW	0

Contenido en memoria CISC

DIRECCIÓN (HEX)	CONTENIDO (HEX)	DESCRIPCIÓN
0x0000	7E	JMP
0x0001	00	Dirección de inicio de programa
0x0002	0D	Dirección de inicio de programa

0x0003	00	Variable res
0x0004	09	vector[0]
0x0005	04	vector[1]
0x0006	05	vector[2]
0x0007	01	vector[3]
0x0008	02	vector[4]
0x0009	06	vector[5]
0x000A	07	vector[6]
0x000B	09	vector[7]
0x000C	01	fin (Solo importa para reservar la dirección)
0x000D	86	instrucción LDAA
0x000E	00	Valor 0
0x000F	C6	instrucción LDAB
0x0010	FF	Valor FF
0x0011	CE	instrucción LDX
0x0012	04	(0x03) es la dirección inicial del vector
0x0013	A5	**.rutina** XPAR
0x0014	25	BLO
0x0015	03	0x00
0x0016	7E	JMP
0x0017	00	00
0x0018	26	.par
0x0019	AB	ADDA
0x001A	00	0,X
0x001B	08	INX
0x001C	8C	CPX
0x001D	0C	#(fin)

0x001E	27	BEQ
0x001F	03	0x01
0x0020	7E	JMP
0x0021	00	00
0x0022	13	.rutina
0x0023	7E	JMP
0x0024	00	00
0x0025	30	.suma
0x0026	E0	**.par** SUBB
0x0027	00	0,X
0x0028	08	INX
0x0029	8C	CPX
0x002A	0C	#(fin)
0x002B	27	BEQ
0x002C	03	0x03
0x002D	7E	JMP
0x002E	00	00
0x002F	13	.rutina
0x0030	AD	**.suma** XABA
0x0031	7E	JMP
0x0032	00	00
0x0033	0C	Dirección de la etiqueta .inicio

Contenido en memoria RISC

Memoria Instrucciones

Dirección	ID	Código	HH	LL	Descripción

0x0000	00	CE	00	03	LDX #vector (0x0000)
0x0001	00	86	00	20	LDAA 0x0000
0x0002	00	C6	00	FF	LDAB 0x00FF
0x0003	00	A5	00	00	**.rutina** XPAR
0x0004	00	25	00	03	BLO 0x00
0x0005	00	01	00	00	NOP
0x0006	00	01	00	00	NOP
0x0007	00	7E	00	15	JMP .par
0x0008	00	AB	00	00	ADDA 0,X
0x0009	00	80	00	00	INCX
0x000A	00	8C	00	08	CPX fin
0x000B	00	27	00	04	BEQ 0x00
0x000C	00	01	00	00	NOP
0x000E	00	01	00	00	NOP
0x000F	00	73	00	03	JMP .rutina
0x0010	00	01	00	00	NOP
0x0011	00	01	00	00	NOP
0x0012	00	73	00	1F	JMP .suma
0x0013	00	01	00	00	NOP
0x0014	00	01	00	00	NOP
0x0015	00	E0	00	00	**.par** SUBB 0,X
0x0016	00	80	00	00	INCX
0x0017	00	8C	00	08	CPX fin
0x0018	00	27	00	00	BEQ 0x00
0x0019	00	01	00	00	NOP
0x001A	00	01	00	00	NOP
0x001B	00	7E	00	03	JMP .rutina
0x001C	00	01	00	00	NOP

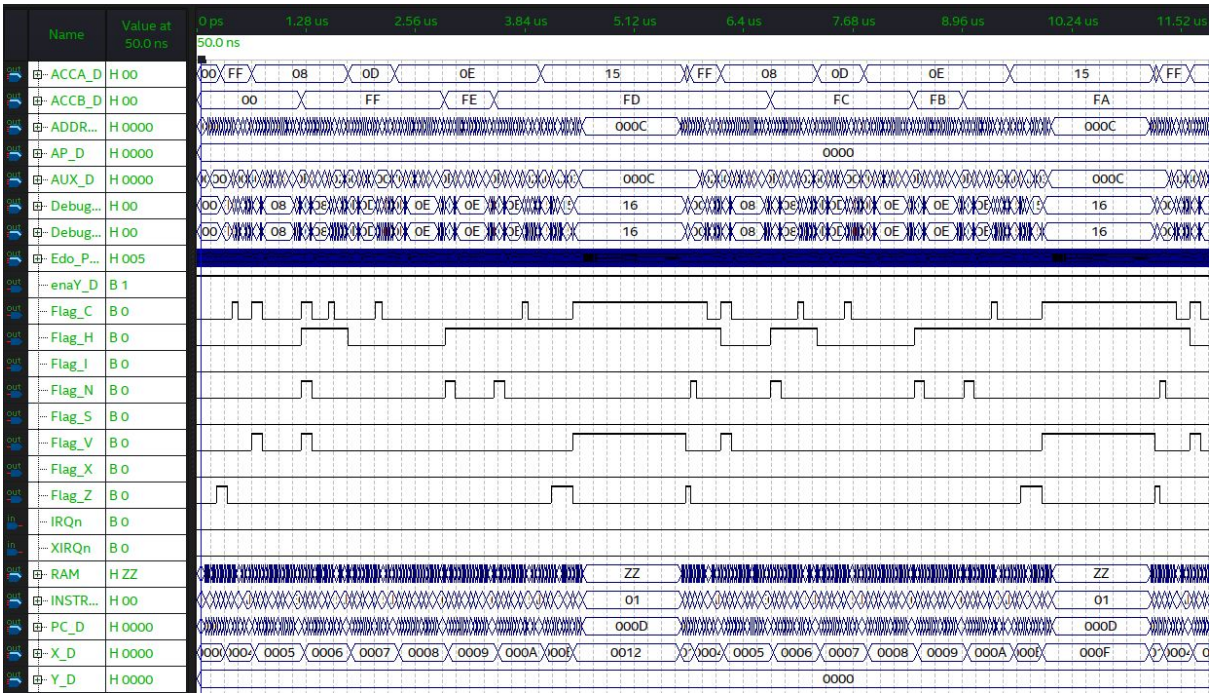
0x001D	00	01	00	00	NOP
0x001F	00	AD	00	00	**.suma** XABA
0x0020	00	7E	00	00	**.fin** JMP 0x0000
0x0021	00	01	00	00	NOP
0x0022	00	01	00	00	NOP

Memoria Datos

Dirección	HH	LL	Descripción
0x0000	0x00	0x02	vector[0]
0x0001	0x00	0x04	vector[1]
0x0002	0x00	0x05	vector[2]
0x0003	0x00	0x01	vector[3]
0x0004	0x00	0x02	vector[4]
0x0005	0x00	0x06	vector[5]
0x0006	0x00	0x07	vector[6]
0x0007	0x00	0x09	vector[7]
0x0008	0x00	0x01	fin

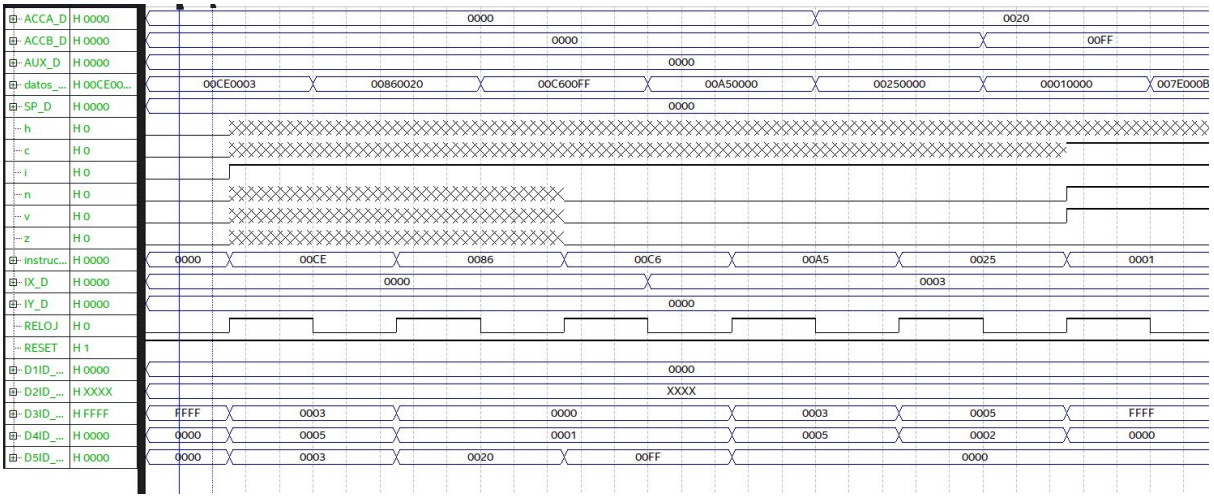
Resultados

En esta simulación se puede ver que la arquitectura CISC pudo correr de manera satisfactoria el programa, Se puede observar como en el us 1.24 está en retorno al inicio.
El resultado de la operación se guardó en IX y vale 0F.



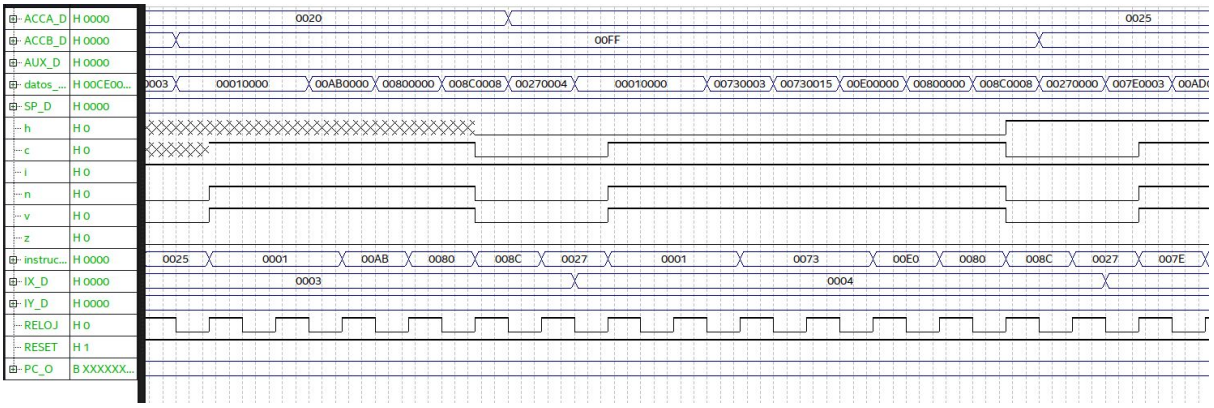
Simulación CISC completa.

En esta simulación se puede apreciar el correcto funcionamiento de la arquitectura RISC.



Simulación RISC 1

En esta arquitectura se ejecuta LDX(CE) el cual guarda un 03 en IX, inmediatamente se ejecuta LDAA(86) que guarda un 20 en ACCA, en el siguiente ciclo se ejecuta LDAB(C6) que guarda un FF en ACCB, se puede notar el resultado hasta el tercer ciclo después de la ejecución del programa. En el siguiente ciclo se ejecuta XPAR(A5) el cual compara el contenido del registro al que apunta a IX(que contiene el dato 0x0005), por lo que es impar así que se esperaría que levante la bandera C la cual se puede ver el resultado hasta después de la ejecución de BLO(25), se añade un NOP después de BLO para que este pueda leer la bandera en su etapa de ejecución (post-escritura para XPAR).



Simulación RISC 2

En esta simulación se puede observar como BLO(25) salta directamente a ADDA(AD) el cual toma el valor en memoria de la dirección que apunta IX(5) y da como resultado 25, inmediatamente después se ejecuta INCX(80) el cual aumenta en una unidad IX pasando de 03 a 04, se ejecuta la instrucción CPX para comparar el contenido de IX(04) con el valor de la dirección final del arreglo (08), como no es asi la bandera Z se queda en 0 y el branch no se ejecuta.

Conclusiones

En este proyecto se implementó lo visto en clase de teoría y laboratorio, para esto se implementó la arquitectura MC68HC11 en sus versiones CISC y RISC, por lo que se enfrentó a muchos problemas que fueron resueltos en el desarrollo del proyecto. Para la versión CISC fue el armar las cartas ASM de cada una de las instrucciones, estas requieren un conocimiento paso a paso de las microinstrucciones en cada ciclo de reloj, estas tienen una duración de ciclos variables pero con la ventaja de que es más flexible con las instrucciones que se pueden crear. La versión RISC consiste en diseñar las instrucciones según su etapa del ciclo fetch, esto permite que todas las instrucciones tenga una duración en ciclos de reloj constante además de poder aprovechar de una forma óptima el tiempo de ejecución de los componentes que que no se estén usando durante el ciclo fetch, una de sus desventajas principales es el que todas las instrucciones tienen un tamaño de palabra igual por lo que es un desperdicio de memoria de instrucción, pero la mayor desventaja son los problemas de múltiples accesos simultáneos a memoria de lectura y escritura además de los saltos los cuales pueden ser solucionado mediante adiciones a la arquitectura original.

Referencias

Savage Carmona, J., Vazquez Rodriguez, G., & Chavez Rodriguez, N. E. (2017). DISEÑO DE MICROPROCESADORES. Universidad Nacional Autónoma de México Facultad de Ingeniería.

Olivo, Á. G., & Martínez, J. P. (2007). *Diseño de procesadores con VHDL*. Universitat d' Alacant (Publicacions).