Introducció als Computadors

Tema 13: Computador SISC von Neumann

http://personals.ac.upc.edu/enricm/Docencia/IC/IC13c.pdf

Enric Morancho
(enricm@ac.upc.edu)

Departament d'Arquitectura de Computadors
Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

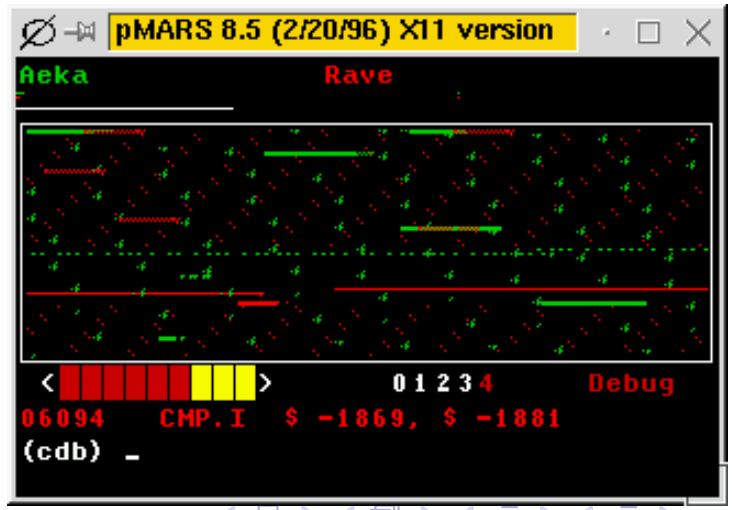
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

2020-21, 1^{er} quad.

Self-modifying code (codi automodificable)

- A una màquina von Neumann el codi és automodificable
  - A IC **no** escriurem codi automodificable
  - Antigament, s'utilitzava per optimitzar programes [1]
  - Ara, virus i altre *malware* ho utilitzen per a dificultar la seva detecció
  - A processadors "von Neumann" actuals, es tracta com a cas excepcional
    - Realment implementen *Modified Harvard architecture* (*Split cache*) [2]
    - Arquitectura dels vostres PC's, portàtils, mòbils, ...
- *Core War*
  - Joc de programació on dos programes en execució a una màquina virtual competeixen per sobre-escriure's i provocar un error en l'altre

[1]



[3]

Índex

Introducció

- Implementarem nova instrucció `JALR`
- Determinarem el temps de cicle mínim del computador Von Neumann
- Estudiarem el comportament dels senyals de modificació d'estat
  - Wr-Mem, Rd-In i Wr-Out
- Compararem rendiments del tres computadors
  - Harvard unicicle, Harvard multicicle i Von Neumann

Índex

Instrucció JALR

- Afegirem la darrera instrucció de LM al repetori SISA
- *Jump Address and Link Register*
  - Imprescindible per a expressar en LM crides/retorns a rutines
  - Guarda el valor actual del PC a un registre i assigna al PC un nou valor
- Sintaxi assemblador:
  - `JALR Rd, Ra`
    - Format 2-R
- Semàntica:
  - `PC = PC + 2; tmp = Ra&(~1); Rd = PC; PC = tmp;`
    - Amb `Ra&(~1)` força que el nou valor del PC sigui parell
- Codificació en llenguatge màquina:
  - `0111 aaa ddd xxxxxx`
    - Els 6 bits baixos de la codificació són irrellevants

Repertori complet LM SISA

16-bit Instruction																Mnemonic	Format
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0	0	0	0	a	a	a	b	b	b	d	d	d	f	f	f	AND, OR, XOR, NOT, ADD, SUB, SHA, SHL	3R
0	0	0	1	a	a	a	b	b	b	d	d	d	f	f	f	CMPLT, CMPLE, -,CMPEQ, CMPLTU,CMPLEU, -, -	
0	0	1	0	a	a	a	d	d	d	n	n	n	n	n	n	ADDI	2R
0	0	1	1	a	a	a	d	d	d	n	n	n	n	n	n	LD	
0	1	0	0	a	a	a	b	b	b	n	n	n	n	n	n	ST	
0	1	0	1	a	a	a	d	d	d	n	n	n	n	n	n	LDB	
0	1	1	0	a	a	a	b	b	b	n	n	n	n	n	n	STB	
0	1	1	1	a	a	a	d	d	d	x	x	x	x	x	x	JALR	
1	0	0	0	a	a	a	0	n	n	n	n	n	n	n	n	BZ	1R
							1									BNZ	
1	0	0	1	d	d	d	0	n	n	n	n	n	n	n	n	MOVI	
							1									MOVHI	
1	0	1	0	d	d	d	0	n	n	n	n	n	n	n	n	IN	
				a	a	a	1									OUT	

Implementació JALR

- El càlcul de `Ra&(~1)` el farem amb una nova funcionalitat de la ALU
  - Tal i com està implementat el mòdul de memòria, els accessos a *word* ja es fan a l'adreça ADDR_MEM&(~1)
  - No passaria res si el PC tingués un valor senar
  - Ho implementem per complir l'especificació de la instrucció

F			OP			
b_2	b_1	b_0	1 1	1 0	0 1	0 0
0	0	0	---	X	CMPLT (X, Y)	AND (X, Y)
0	0	1	---	Y	CMPLE (X, Y)	OR (X, Y)
0	1	0	---	MOVHI(X, Y)	---	XOR(X, Y)
0	1	1	---	X&(~1)	CMPEQ (X, Y)	NOT (X)
1	0	0	---	---	CMPLTU (X, Y)	ADD (X, Y)
1	0	1	---	---	CMPLEU (X, Y)	SUB (X, Y)
1	1	0	---	---	---	SHA(X, Y)
1	1	1	---	---	---	SHL(X, Y)

- Els detalls de d'aquesta nova funcionalitat de la ALU els teniu a la documentació de l'assignatura

Implementació JALR v1.0

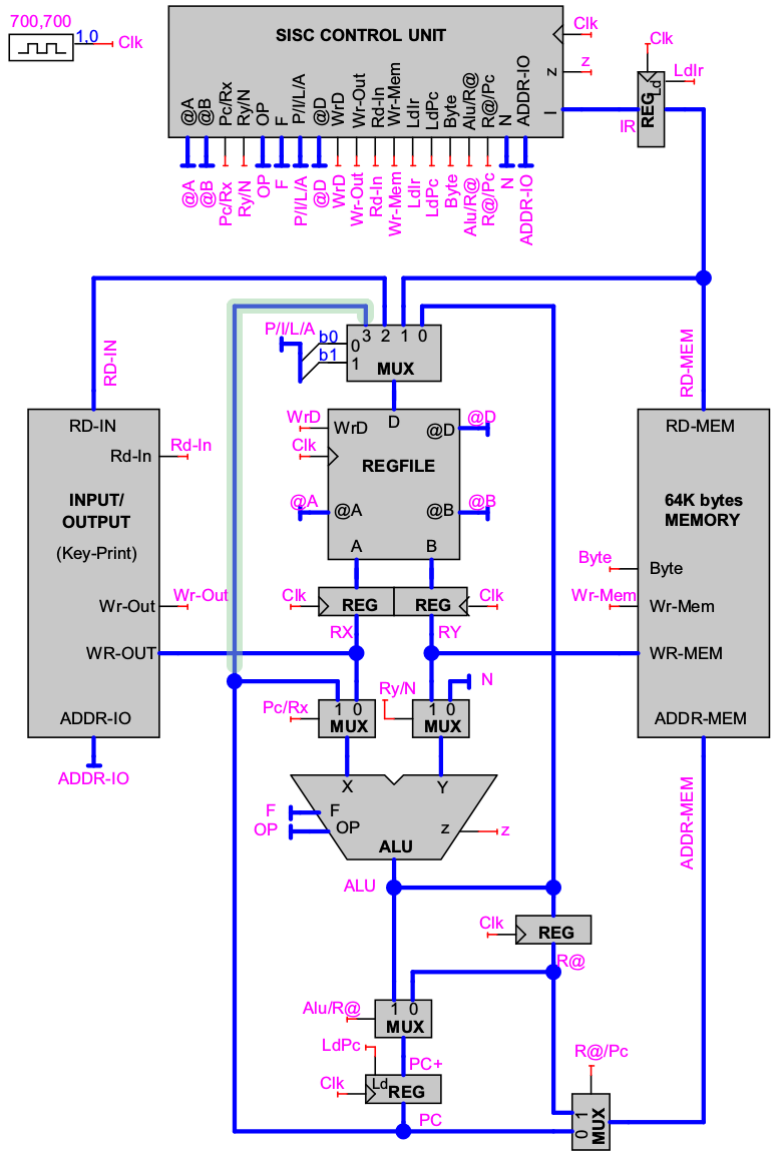
- Utilitza 4 nodes: *Fetch*, *Decode* i dos de càlcul
  - Jalr1: per carregar a R@ el nou valor del PC
  - Jalr2: per carregar a PC el valor de R@ i a Rd el valor de PC

Estado	Acciones	Palabra de control compactada
F	IR ← Memw[PC] // PC ← PC + 2	R@/Pc=0, Byte=0, LdIr=1, Pc/Rx=1, N=0x0002, Ry/N=0, OP=00, F=100, Alu/R@=1, LdPc=1.
D	R@ ← PC + SE(N8)*2 // (RX ← Ra) // (RY ← Rb)	N=SE(IR<7..0>)*2, Pc/Rx=1, Ry/N=0, OP=00, F=100.
Jalr1	R@ ← RX & (~1)	Pc/Rx=0, OP=10, F=011.
Jalr2	PC ← R@ // Rd ← PC	Alu/R@=0, LdPc=1, Pc/Rx=1, OP=10, F=000, P/I/L/A=00, WrD=1, @D=IR<8..6>.

- Es podria fer amb un únic node de càcul?
  - No, perquè fem dos usos de la ALU
    - Per calcular el nou valor del PC
    - Per copiar el valor del PC a Rd del REGILE
- I si afegíssim un camí directe des del PC al REGFILE?
  - En paral·lel, podrà guardar el PC a Rd i fer un càlcul amb l'ALU

Unitat de procés (amb camí per a JALR)

- Nou camí PC → REGFILE
  - **Sense passar per la ALU**



Implementació JALR v2.0

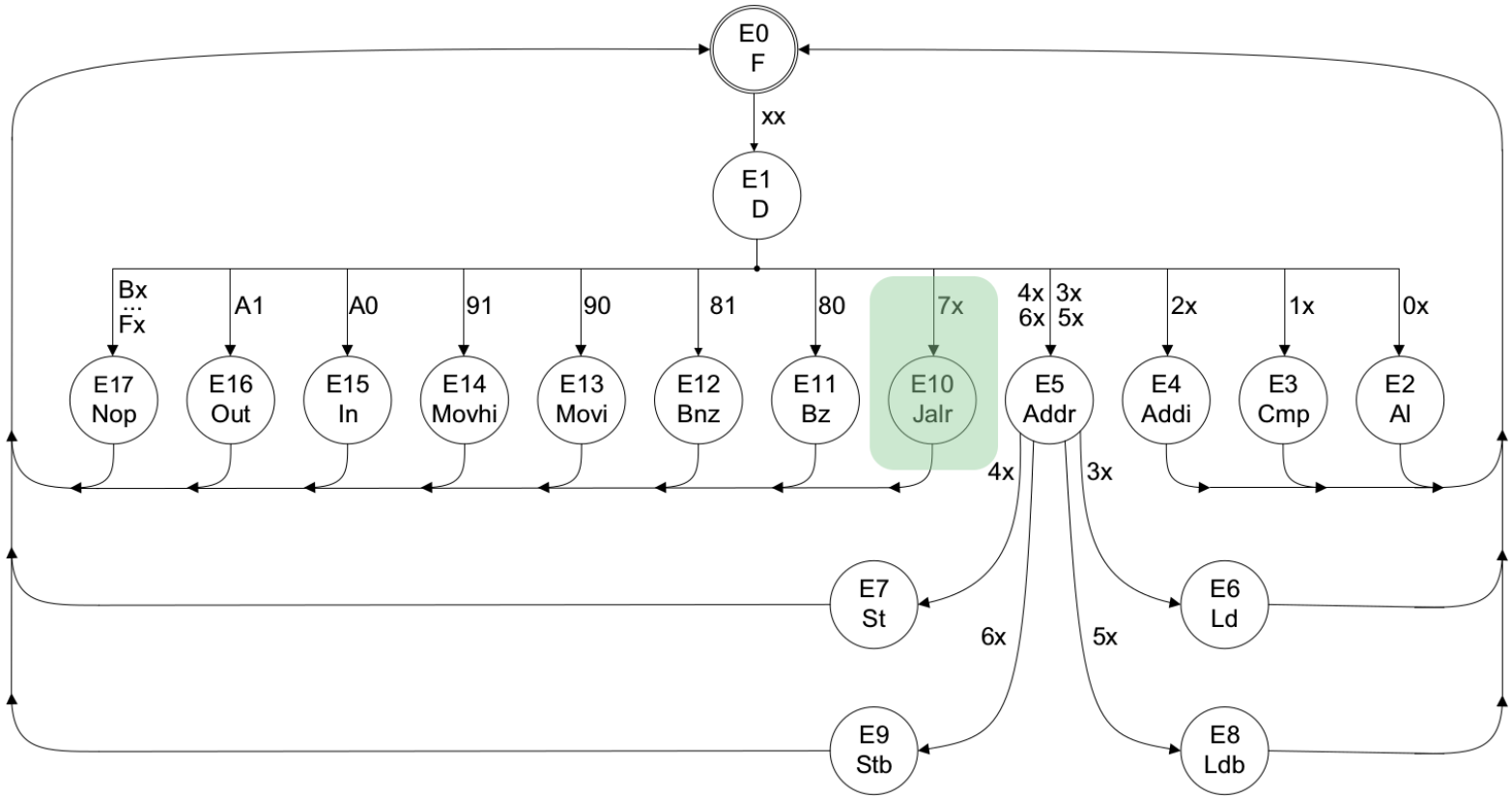
- Utilitzarà 3 nodes: *Fetch*, *Decode* i un de càlcul
  - Jalr: carrega al PC el valor de Ra&(~1) i a Rd el valor de PC

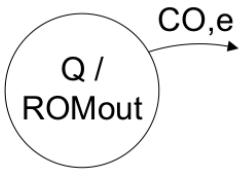
Estado	Acciones	Palabra de control compactada
F	IR ← Memw[PC] // PC ← PC + 2	R@/Pc=0, Byte=0, LdIr=1, Pc/Rx=1, N=0x0002, Ry/N=0, F=100, OP=00, Alu/R@=1, LdPc=1.
D	R@ ← PC + SE(N8)*2 // (RX ← Ra) // (RY ← Rb)	N=SE(IR<7..0>)*2, Pc/Rx=1, Ry/N=0, F=100, OP=00.
Jalr	PC ← RX & (~1) // Rd ← PC	Pc/Rx=0, OP=10, F=011, Alu/R@=1, LdPc=1, P/I/L/A=11, WrD=1, @D=IR<8..6>.

- **És la implementació que utilitzarem**
- Paraula de control per a l'estat Jarl de la instrucció `JARL R7, R0`
  - Senyals imprescindibles (@A, @B i ADDR-IO són irrellevants en aquest estat, tot i que podríem saber-ne el valor)

@A			@B			Pc/Rx	Ry/N	OP		F			P/I/L/A		@D			WrD	Wr-Out	Rd-In	Wr-Mem	Ldlr	LdPc	Byte	Alu/R@	R@/Pc	N (hexa)				ADDR-IO (hexa)	
x	x	x	x	x	x	0	x	1	0	0	1	1	1	1	1	1	1	1	0	0	0	x	1	x	1	x	X	X	X	X	X	X

Graf d'estats de la UC (amb JALR)





CO: Código de operación de la Instrucción en hexadecimal ($I_{15}I_{14}I_{13}I_{12}$)
e: Extensión del código de operación (I_8)
Q: Estado en hexadecimal
ROMout: Mnemotécnico que indica la salida de la ROM_OUT del CONTROL

Contingut ROM OUT per a l'estat Jalr

@ROM	Bnz	Bz	WrMem	RdIn	WrOut	WrD	Ldlr	Byte	R@/Pc	Alu/R@	Pc/Rx	Ry/N	P/I/L/A1	P/I/L/A0	OP1	OP0	MxN1	MxN0	MxF	F2	F1	F0	Mx@D1	Mx@D0
10	1	1	0	0	0	1	x	x	x	1	0	x	1	1	1	0	x	x	1	0	1	1	0	1

- Com sempre actualitza el PC, tant Bz com Bnz han de valer "1"
  - Anàleg a l'estat de *Fetch*
- El bus P/I/L/A ha de valer "11" perquè és l'entrada que encamina el valor del PC a l'entrada del REGFILE
- La funció de la ALU és OP=10 i F=011 (calcular `X&(~1)`)

Índex

Estudi del temps de cicle mínim

- Estudiarem el comportament dels estats potencialment més lents:
  - *Fetch*, *Decode*, Ldb i Cmp (en particular, el cas `CMPLE`)
- Els T_p dels blocs seran els mateixos que quan vàrem estudiar el computador Harvard
  - $T_p(\text{NOT}) = 10$ u.t.
  - $T_p(\text{And-2}) = T_p(\text{Or-2}) = 20$ u.t.
  - $T_p(\text{FF}) = 100$ u.t.
  - $T_{acc}(\text{32KB-RAM}) = 800$ u.t.
    - Temps d'accés (lectura) a un mòdul de memòria RAM
- Alguns casos:
  - $T_p(\text{ROM\_Q+})$=120 u.t i $T_p(\text{ROM\_OUT}) = 60$ u.t.
  - $T_p(\text{MUX-2-1-Sel}) = 50$ u.t. i $T_p(\text{MUX-2-1-Data})$= 40 u.t.
  - $T_p(ALU_{ADD}) = 860$ u.t.
  - $T_p(MEM_{Ldb}) = 880$ u.t. i $T_p(MEM_{Ld}) = 840$ u.t.

T_p a l'estat de *Fetch*

- Lectura instrucció:
  - $REG_Q \rightarrow R@/PC \rightarrow ADDR\_MEM \rightarrow RDMEM \rightarrow IR$
  - T_p = 100 (REG_Q) + 60 (ROM OUT) + 50 (mux R@/PC sel) + 840 (Memòria Ld) + 40 (dada registre càrrega IR) = 1.090 u.t.
- PC = PC +2
  - $REG_Q \rightarrow MxN \rightarrow N \rightarrow Ry/N \rightarrow ALU \rightarrow Alu/R@ \rightarrow PC$
  - T_p = 100 (REG_Q) + 60 (ROM OUT) + 90 (MUX_{4-1} MxN selecció) + 40 (Mux Ry/N Dada)+ 860 (ALU ADD) + 40 (MUX_{2-1} Alu/R@ dada) + 40 (dada registre càrrega PC) = 1.230 u.t.
- Conclusió, l'etapa de *Fetch* imposa que $T_c \geq 1.230 u.t.$

T_p a l'estat de *Decode*

- Càlcul següent estat UC:
  - $REG_Q \rightarrow Q^+ \rightarrow REG_Q$
  - T_p = 100 (REG Q) + 120 (ROM Q+) = 220 u.t.
- Lectura de registres
  - $IR \rightarrow REGFILE \rightarrow RX/RY$
  - T_p = 100 (IR) + 130 (REGFILE MUX_{8-1} selecció) = 230 u.t.
- Càlcul adreça destí del salt
  - $REG_Q \rightarrow MxN \rightarrow N \rightarrow Ry/N \rightarrow ALU \rightarrow R@$
  - T_p = 100 (REG_Q) + 60 (ROMOUT) + 90 (MUX_{4-1} MxN selecció) + 40 (Mux Ry/N Dada)+ 860 (ALU ADD) = 1.150 u.t.
- Conclusions:
  - L'estat *Decode* imposa que $T_c \geq 1.150 u.t.$
  - És menys restrictiu que *Fetch*

T_p als estats Ldb i Cmp

- Ldb: Accés a memòria i escriptura a REGFILE:
  - $REG_Q \rightarrow R@/PC \rightarrow ADDR\_MEM \rightarrow RD\_MEM \rightarrow P/I/L/A \rightarrow REGFILE$
  - T_p = 100 (REG_Q) + 60 (ROMOUT) + 50 (MUX_{2-1} R@/PC selecció) + 880 (RAM Ldb) + 80 (MUX_{4-1} P/I/L/A) + 40 (dada registre càrrega Rd) = 1.210 u.t.
- Cmp (`CMPLE`)
  - $REG_Q \rightarrow Pc/Rx, Ry/N \rightarrow X, Y \rightarrow ALU \rightarrow P/I/L/A \rightarrow REGFILE$
  - T_p = 100 (REG_Q) + 60 (ROMOUT) + 50 (PC/Rx, Ry/N MUX_{2-1} selecció) + 1.020 (ALU `CMPLE`) + 80 (MUX_{4-1} P/I/L/A dada) + 40 (dada registre càrrega Rd) = 1.350 u.t.
- Conclusions:
  - L'estat més restrictiu és Cmp (cas `CMPLE`)
  - Cmp imposa $T_c \geq 1.350 u.t.$
  - Arrodonim i determinem T_c = 1.400 u.t.

Índex

Escriptura a memòria

- Assumim els mateixos paràmetres que als temes anteriors
  - $T_{acc} = 800\ u.t.$, $T_{su} = 60\ u.t.$, $T_{pw} = 600\ u.t.$, $T_h = 40\ u.t.$
- Assumim $T_c = 1.400\ u.t$
- Com al Harvard unicicle, el computador Von Neumann generarà Wr-Mem a la fase T0 del senyal de rellotge
- Com podem descompondre $T_c = T1 + T0$?
  - T1 mínim ha de ser 210 u.t.
    - Justificat a la documentació de l'assignatura
  - $T_{pw} = 600 \leq T0 \leq Tc - T1 = 1.400 - 210 = 1.190\ u.t.$
    $\Longrightarrow\ 600 \leq T0 \leq 1.190$
- Conclusions:
  - Un senyal de rellotge simètric amb $T1 = T0 = 700\ u.t.$, $T_c = T1 + T0 = 1.400\ u.t.$ garanteix escriptures correctes
  - Els senyals Rd-In i Wr-Out són menys restrictius

Índex

Temps mig per instrucció

- Sigui r el ratio d'instruccions lentes a l'execució d'un programa
  - $r = \frac{N_{instruccions\ \ lentes}}{N_{instruccions\ \ lentes} + N_{instruccions\ \ ràpides}} \qquad 0 \leq r \leq 1$

Computador	Temps mig per instrucció
Hardvard unicicle	3.000 u.t./inst.
Harvard multicicle	$(3 + r) \cdot 750$ u.t./inst.
Von Neumann	$(3 + r) \cdot 1.400$ u.t./inst.

- Observacions:
  - Von Neumann és el més lent
  - Harvard multicicle és el més ràpid
    - Empatat amb el Harvard unicicle en el cas extrem r=1

Comparació de rendiments: temps

- Quant **trigarà més** (en %) el Von Neumann que el Harvard multicicle
  - Regla de 3 amb el temps mig per instrucció

    750·(3+r) u.t./inst. - 100
    1.400·(3+r) u.t./inst. - x
  - x = 100 · (1.400·(3+r)) / (750·(3+r)) = 100·1.400/750 = 186,67
    $\Longrightarrow$ triga un 86,67% més
- Quant **trigarà més** (en %) el Von Neumann que Harvard unicicle?
  - Regla de 3 amb el temps mig per instrucció

    3.000 u.t./inst. - 100
    1.440·(3+r) u.t./inst. - x
  - x = 100 · (1.400·(3+r)) / 3.000 = 140·(3+r)/3
    - Si r=0 $\Longrightarrow$ x=140 $\Longrightarrow$ Triga un 40% més
    - Si r=0,2 $\Longrightarrow$ x=149,33 $\Longrightarrow$ Triga un 49,33% més
    - Si r=1 $\Longrightarrow$ x=186,67 $\Longrightarrow$ Triga un 86,67% més

Índex

Exercici: afegir una nova instrucció al LM

- Escenaris:
  - Sense afegir *hardware*
  - Afegint *hardware* (multiplexors, registres, busos,...)
    - A la UC $\Longrightarrow$ afegir sortides a la ROM OUT
    - A la UP $\Longrightarrow$ afegir senyals a la paraula de control $\Longrightarrow$ modificar la UC
- Cal decidir quants estats de càlcul necessitarà la nova instrucció:
  - Actualitzar graf d'estats de la UC
  - Modificar ROM Q+ per reflectir noves transicions
- Cal determinar la paraula de control a cada nou estat
  - Cal determinar sortides de la ROM OUT a cada nou estat
- La resta d'instruccions del repertori SISA han de continuar executant-se com fins ara

Exemples

- Afegirem noves instruccions en diferents escenaris
  - Sense modificar *hardware*: instrucció `DECTEST1`
  - Modificant *hardware*:
    - A la UC: instrucció `ACCUMV`
    - A la UP: instruccions `mnemoI16`
- Alguns cops l'enunciat ens dirà en quin cas estem, però en d'altres ho haurem de deduir

DECTEST1: especificació

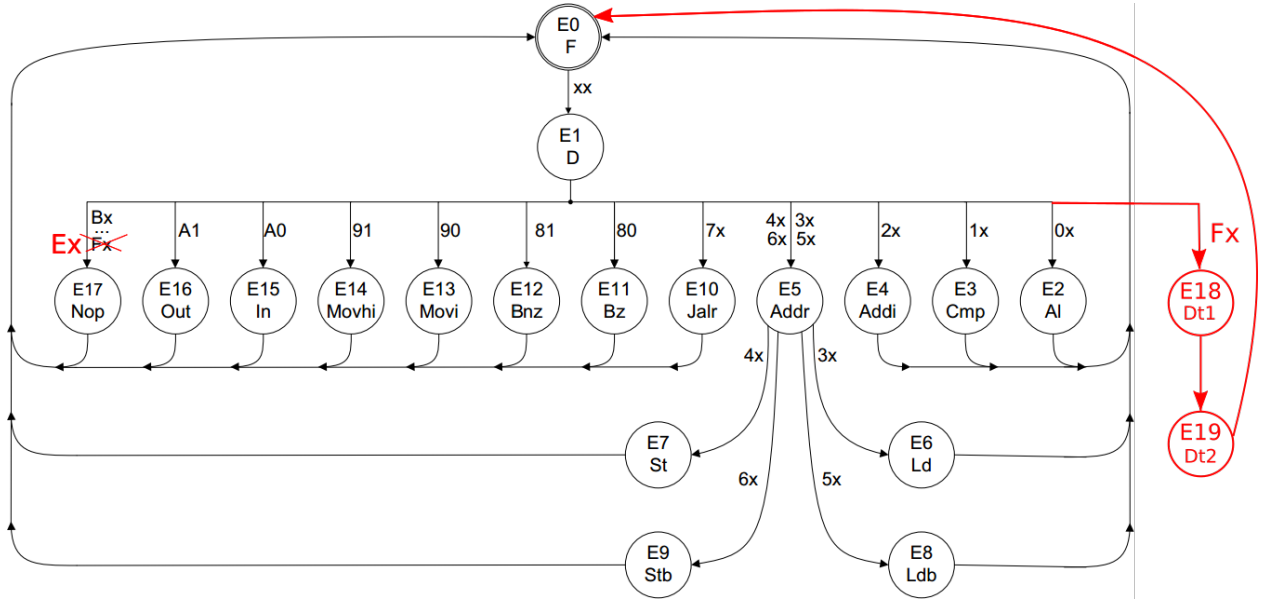
- *Decrement 1 and test*: permetrà implementar una espera
  - Sintaxi: `DECTEST1 Ra`
  - Codificació: `1111 aaa x 11111111`
  - Semàntica: `PC = PC+2; Ra = Ra-1; if (Ra!=0) PC = PC-2;`
- La instrucció decrementa Ra. Si el resultat no és 0, modifica el PC de forma que la següent instrucció a executar torni a ser el `DECTEST1`
  - Permet implementar bucles dins d'una instrucció de LM
- D'on traurem el -1?
  - Dels 8 bits baixos de la codificació de la instrucció :-)

DECTEST1: estratègia

- Estratègia d'implementació: Imitarem `BZ`
  - A un cicle guardarem a R@ la direcció de salt PC-2
  - Al cicle següent guardem (RX-1) a Rd; com el bit z reflecteix si (RX-1)!=0, si z=0 actualitzarem PC amb R@
- Calen dos estats per a completar l'execució:
  - Dt1: `R@` $\leftarrow$ `PC - 2`
    - També necessitem `RX` $\leftarrow$ `Ra` però ja es fa a tots els cicles
  - Dt2: `Ra` $\leftarrow$ `RX - 1; if (!z) PC` $\leftarrow$ `R@`
- Modificacions a les ROM's:
  - ROM Q+: reflectir les noves transicions
  - ROM OUT: per generar les paraules de control

DECTEST1: ROM Q+ i ROM OUT

- Graf d'estats UC:



- Modificacions a la ROM Q+
  - ROM_Q+[00001 1111 x] = 0x12, ROM_Q+[10010 1111x] = 0x13 i ROM_Q+[10011 xxxxx] = 0x00
- Modificacions a la ROM OUT

@ROM	Bnz	Bz	WrMem	RdIn	WrOut	WrD	LdIr	Byte	R@/Pc	Alu/R@	Pc/Rx	Ry/N	P/I/L/A1	P/I/L/A0	OP1	OP0	MxN1	MxN0	MxF	F2	F1	F0	Mx@D1	Mx@D0
18	0	0	0	0	0	0	0	x	x	x	1	0	x	x	0	0	1	1	1	1	0	1	x	x
19	1	0	0	0	0	1	x	x	x	0	0	0	0	0	0	0	0	1	1	1	0	0	1	0

Acciones asociadas al estado (en lenguaje de transferencia de registros)

R@ ← PC - 2
Ra ← RX-1; if ((RX-1)!=0) PC ← R@

DECTESTN: generalització de DECTEST1

- *Decrement N and test*
  - Sintaxi: `DECTESTN Ra, N8`
  - Codificació: `1111 aaa x nnnnnnnn`
  - Semàntica: `PC = PC+2; Ra = Ra-N8; if (Ra!=0) PC = PC-2;`
- La solució de `DECTEST1` seria vàlida per a `DECTESTN` perquè el valor de N s'extreu de la codificació de la instrucció.

DECTEST1: possible optimització

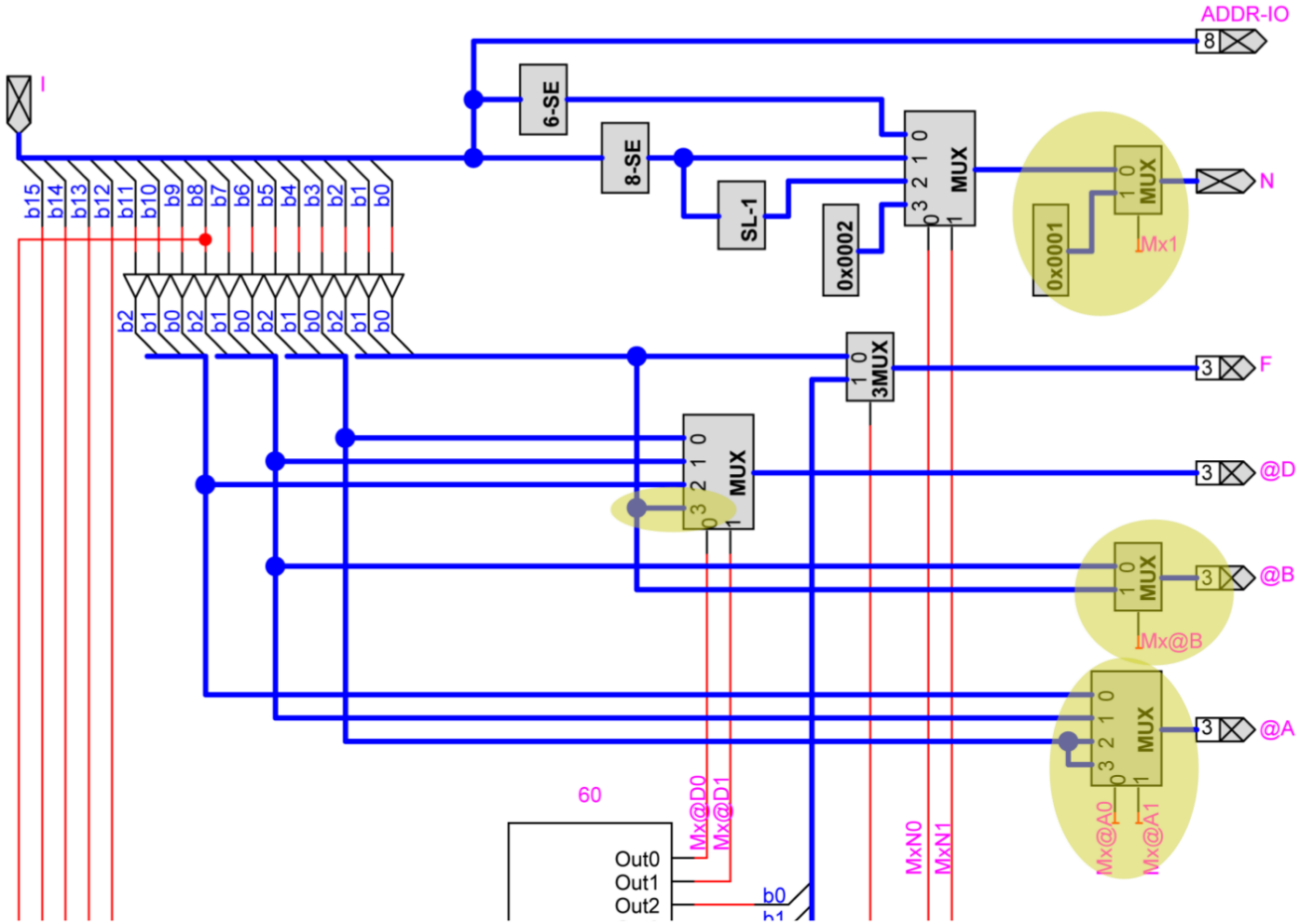
- Sense la generalització, les accions de Dt1 ja es fan a Decode
  - A Decode, l'execució de `DECTEST1` fa `R@` $\leftarrow$ `PC + SE(N8)` $\cdot 2$, que seria `R@` $\leftarrow$ `PC + (-1)` $\cdot 2$
    - Aprofitem el "-1" de la codificació de la instrucció per decrementar tant el registre Ra com el PC
  - A Decode també es carrega Ra a Rx
- Podríem eliminar Dt1 i passar de Decode a Dt2
- `DECTEST1` trigaria 3 cicles

ACCUMV (E4 Q2 1314)

- *Accumulate memory vector*: Acumula a Rd la suma dels elements d'un vector, on l'adreça inicial del vector és Ra i el vector té Rb elements.
  - Sintaxi: `ACCUMV Rd, Ra, Rb, Rc`
  - Codificació: `1011 aaa bbb ddd ccc`
  - Semàntica:
    `PC=PC+2; Rc=`Mem_w`[Ra]; Rd=Rd+Rc; Ra=Ra+2; Rb=Rb-1;`
    `if (Rb!=0) PC=PC-2;`
- Observacions:
  - La instrucció utilitza Rc com a registre temporal
  - La instrucció també modifica Ra i Rb.
- Cal afegir *hardware*?
  - Sí, perquè hem de poder llegir els registres identificats per IR_{543} i IR_{210}
  - També hem de poder escriure el registre identificat per IR_{210}
  - També caldrà poder generar la constant "1"
  - Caldrà afegir multiplexors per a generar @A i @B
    - Tindran senyals de control que haurà de generar la ROM OUT
    - Haurem de determinar el seu valor per als estats ja existents

ACCUMV: control

- L'enunciat ens indica com queda la UC
  - No modifica UP
  - Quatre nous senyals a la ROM OUT: Mx1, Mx@B, Mx@A0, Mx@A1



ACCUMV: accions a cada estat

- L'enunciat ens diu que caldran 6 estats de càlcul
- Cal omplir els forats amb una acció per forat

Nodo/Estado		Acciones
Número	Mnem.	
E0	F	IR ← MEMw[PC] // PC ← PC+2
E1	D	[ ] // [ ] // RY ← Rb
E17	Acc1	R@ ← [ ]
E18	Acc2	Rc ← [ ] // [ ]
E19	Acc3	Ra ← [ ] // [ ] // RY ← Rc
E20	Acc4	Rd ← [ ]
E21	Acc5	R@ ← [ ] // [ ]
E22	Acc6	[ ] // if([ ]) [ ]

ACCUMV: accions a cada estat

- L'enunciat ens diu que caldran 6 estats de càlcul
- Cal omplir els forats amb una acció per forat

Nodo/Estado		Acciones
Número	Mnem.	
E0	F	IR ← MEMw[PC] // PC ← PC+2
E1	D	**R@ ← PC+SE(N8)*2** // **RX ← Ra** // RY ← Rb
E17	Acc1	R@ ← **RX**
E18	Acc2	Rc ← **MEMw[R@]** // **RX ← Ra**
E19	Acc3	Ra ← **RX + 0x0002** // **RX ← Rd** // RY ← Rc
E20	Acc4	**Rd ← RX + RY**
E21	Acc5	**R@ ← PC − 2** // **RX ← Rb**
E22	Acc6	**Rb ← RX − 1** // if(**RX-1 != 0x0000**) **PC ← R@**

ACCUMV: ROM OUT

- Cal omplir les files i columnes indicades

@ROM	Mx@A1	Mx@A0	Mx@B	Mx1	Bnz	Bz	WrMem	RdIn	WrOut	WrD	Ldlr	Byte	R@/Pc	Alu/R@	Pc/Rx	Ry/N	P/I/L/A1	P/I/L/A0	OP1	OP0	MxN1	MxN0	MxF	F2	F1	F0	Mx@D1	Mx@D0	
0																													F
1																													D
2																													AI
3																													Cmp
4																													Addi
5																													Addr
6																													Ld
7																													St
8																													Ldb
9																													Stb
10																													Jalr
11																													Bz
12																													Bnz
13																													Movi
14																													Movhi
15																													In
16																													Out
17																													Acc1
18																													Acc2
19																													Acc3
20																													Acc4
21																													Acc5
22																													Acc6
23..31																													Nop

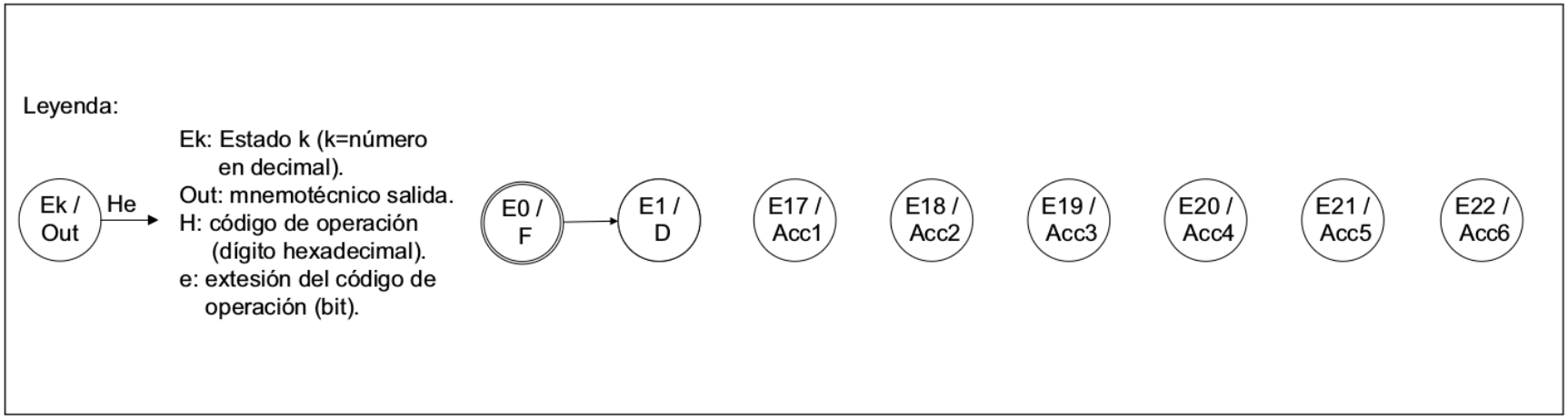
ACCUMV: ROM OUT

- Cal omplir les files i columnes indicades

@ROM	Mx@A1	Mx@A0	Mx@B	Mx1	Bnz	Bz	WrMem	RdIn	WrOut	WrD	LdIr	Byte	R@/Pc	Alu/R@	Pc/Rx	Ry/N	P/I/L/A1	P/I/L/A0	OP1	OP0	MxN1	MxN0	MxF	F2	F1	F0	Mx@D1	Mx@D0	
0			x	0																									F
1			0	0																									D
2			x	x																									AI
3			x	x																									Cmp
4			x	0																									Addi
5			0	0																									Addr
6			x	x																									Ld
7			x	x																									St
8			x	x																									Ldb
9			x	x																									Stb
10			x	x																									Jalr
11			x	x																									Bz
12			x	x																									Bnz
13			x	0																									Movi
14			x	0																									Movhi
15			x	x																									In
16			x	x																									Out
17			x	x																									Acc1
18	0	0	x	x	0	0	0	0	0	1	0	0	1	x	x	x	0	1	x	x	x	x	x	x	x	x	1	1	Acc2
19	1	x	1	0	0	0	0	0	0	1	0	x	x	x	0	0	0	0	0	0	1	1	1	1	0	0	1	0	Acc3
20			x	x																									Acc4
21	0	1	x	0	0	0	0	0	0	0	0	x	x	x	1	0	x	x	0	0	1	1	1	1	0	1	x	x	Acc5
22	x	x	x	1	1	0	0	0	0	1	x	x	x	0	0	0	0	0	0	0	x	x	1	1	0	1	0	1	Acc6
23..31			x	x																									Nop

ACCUMV: Graf d'estats i ROM Q+

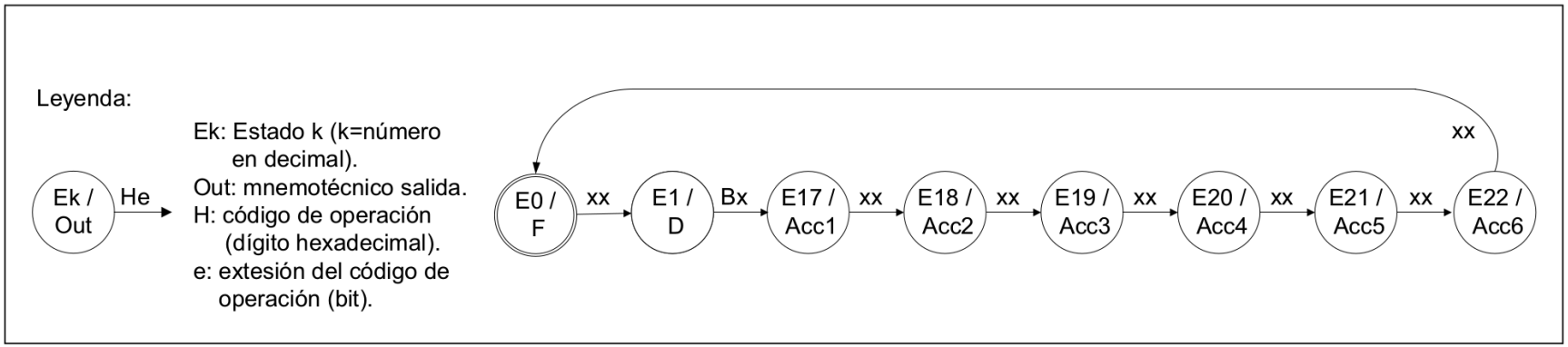
- Cal indicar transicions entre els nous estats



- Indiqueu el contingut de l'adreça(ces) de la ROM Q+ que implementa(en) la transició de E1 a E17 ?

ACCUMV: Graf d'estats i ROM Q+

- Cal indicar transicions entre els nous estats



- Indiqueu el contingut de l'adreça(ces) de la ROM Q+ que implementa(en) la transició de E1 a E17 ?
  - L'adreça 0x036 ($00001\ 1011\ 0_2, 54_{10}$) contindrà 0x11 ($10001_2, 17_{10}$)
  - L'adreça 0x037 ($00001\ 1011\ 1_2, 55_{10}$) contindrà 0x11 ($10001_2, 17_{10}$)

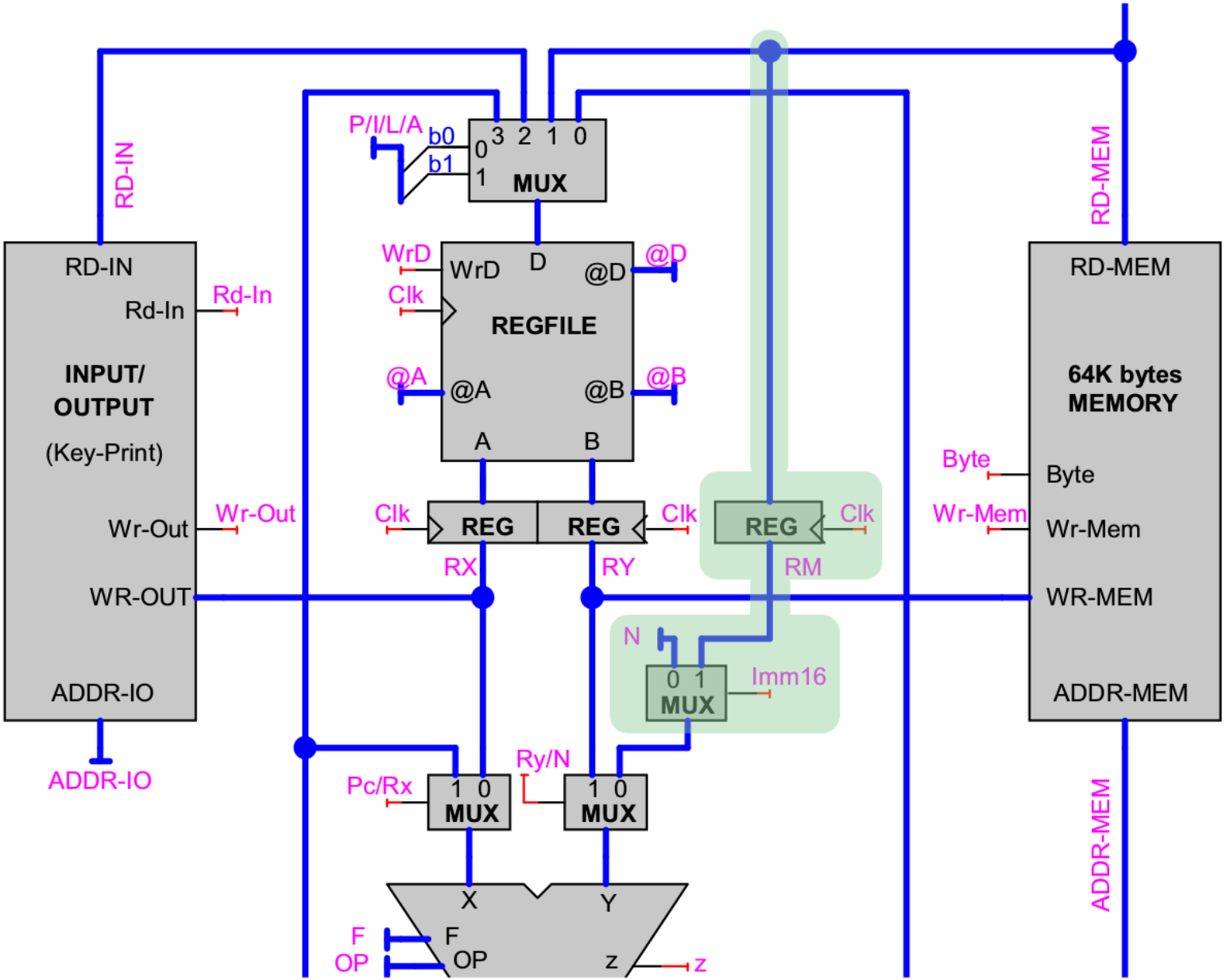
- Completad el diseño del SISC Von Neumann para que pueda ejecutar, además de las 25 instrucciones originales SISA, las nuevas 7 instrucciones Aritmético-Lógicas y las 5 de Comparación en las que el segundo operando fuente es un inmediato de 16 bits.
  - La instrucción NOT no tiene segundo operando por lo que no existe la NOT con inmediato de 16 bits.
- Cada nueva instrucción ocupa dos palabras consecutivas en memoria, la de dirección @, donde se encuentran los 16 bits de más peso de la instrucción y la de dirección @+2 donde se encuentran los 16 bits del inmediato (N16 = nnnnnnnnnnnnnnn).
- Una vez ejecutada la instrucción el PC debe quedar incrementado en 4 para apuntar a la siguiente instrucción en secuencia.

mnemoI16: codificació/sintaxi/semàntica

- Codificación: `1011 aaa e xx ddd fff nnnnnnnnnnnnnnn`
  con e = 0 para las operaciones AL y con e = 1 para las CMP.
  El campo `fff` codifica la operación a realizar de la misma manera que se codifica para las instrucciones AL y CMP originales
- Sintaxis: `mnemoI16 Rd, Ra, N16`
  siendo `mnemo` cualquiera de los mnemotécnicos de las operaciones AL o CMP originales.
  - Ejemplo de instrucción AL: `SUBI16 Rd, Ra, N16`.
  - Ejemplo de instrucción CMP: `CMPLEUI16 Rd, Ra, N16`.
- Semántica: `Rd = Ra op N16`
  siendo op la operación AL o CMP que corresponde al `mnemo` (de la instrucción en ensamblador) o al campo `fff` (de la instrucción en lenguaje máquina).

- En la UP se ha añadido un camino que conecta el bus RD-MEM con la entrada 0 del MUX-2-1 con señal de selección Ry/N a través de un nuevo registro, llamado RM, y de un nuevo MUX-2-1, con señal de selección Imm16, como se muestra en la figura. Ahora, el bus N, que genera la UC, llega a la ALU a través de estos dos multiplexores (cuando Imm16=0 y Ry/N=0).
- En la UC solo se requiere modificar el contenido de algunas palabras de la ROM_Q+ (ya que el nuevo grafo de la UC tiene tres nuevos estados) y de la ROM_OUT (que hora tiene un bit más de salida, Imm16, que forma parte de la nueva palabra de control). No ha hecho falta añadir ninguna nueva lógica en la UC, por lo que no dibujamos su circuito interno.

mnemoI16: modificacions *hardware*



mnemoI16: accions a cada estat

- Completad el contenido de las cajas vacías de la siguiente tabla que indica, mediante una fila para cada nodo del grafo de estados de la unidad de control, la acción (o acciones en paralelo) que se realiza en el computador en cada uno de los nodos que se requieren para ejecutar las nuevas instrucciones (Fetch, Decode, y los 3 nodos nuevos): F, D, ImmA, ImmB e ImmC.
- Para especificar las acciones se usa el mismo lenguaje de transferencia de registros que en la documentación.

Nodo/Estado		Acciones
Número	Mnem.	
E0	F	`IR ← MEMw[PC] // ` [ ]
E1	D	[ ] `// RX ← Ra // RY ← Rb`
E17	ImmA	[ ] `//` [ ] `// RX ← Ra`
E18	ImmB	`Rd ←` [ ]
E19	ImmC	`Rd ←` [ ]

mnemoI16: accions a cada estat

- Completad el contenido de las cajas vacías de la siguiente tabla que indica, mediante una fila para cada nodo del grafo de estados de la unidad de control, la acción (o acciones en paralelo) que se realiza en el computador en cada uno de los nodos que se requieren para ejecutar las nuevas instrucciones (Fetch, Decode, y los 3 nodos nuevos): F, D, ImmA, ImmB e ImmC.
- Para especificar las acciones se usa el mismo lenguaje de transferencia de registros que en la documentación.

Nodo/Estado		Acciones
Número	Mnem.	
E0	F	`IR ← MEMw[PC]` // **`PC ← PC+2`**
E1	D	**`R@ ← PC+SE(N8)*2`** // `RX ← Ra` // `RY ← Rb`
E17	ImmA	**`RM ← MEMw[PC]`** // **`PC ← PC+2`** // `RX ← Ra`
E18	ImmB	`Rd ←` **`RX Al RM`**
E19	ImmC	`Rd ←` **`RX Cmp RM`**

mnemoI16: ROM OUT

- Completad (poniendo 0, 1 o x en cada bit) la columna Imm16 y las 3 filas sin sombrear de la tabla que especifica el contenido de la ROM_OUT para que se ejecuten correctamente todas las instrucciones, poniendo el máximo número de x posibles.
- La dirección 0 de la ROM corresponde al estado E0 (F), la 1 al E1 (D), la dirección 17 al estado E17 (ImmA), etc.

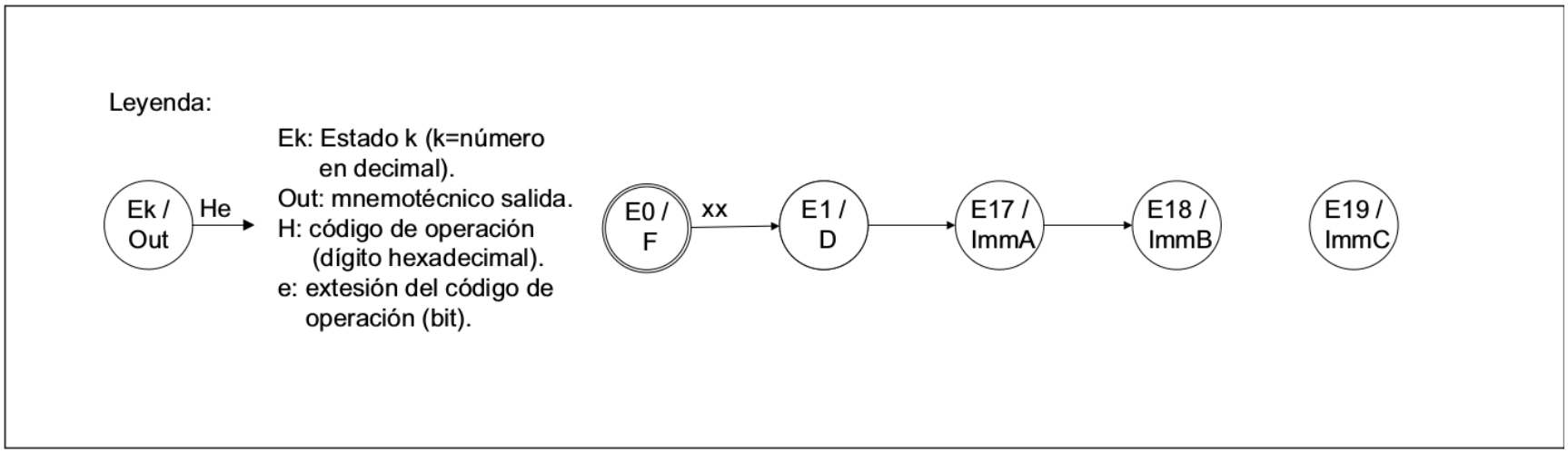
mnemoI16: ROM OUT

@ROM	Imm16	Bnz	Bz	WrMem	RdIn	WrOut	WrD	LdIr	Byte	R@/Pc	Alu/R@	Pc/Rx	Ry/N	P/I/L/A1	P/I/L/A0	OP1	OP0	MxN1	MxN0	MxF	F2	F1	F0	Mx@D1	Mx@D0	Node
0																										F
1																										D
2																										AI
3																										Cmp
4																										Addi
5																										Addr
6																										Ld
7																										St
8																										Ldb
9																										Stb
10																										Jalr
11																										Bz
12																										Bnz
13																										Movi
14																										Movhi
15																										In
16																										Out
17																										ImmA
18																										ImmB
19																										ImmC
20..31																										Nop

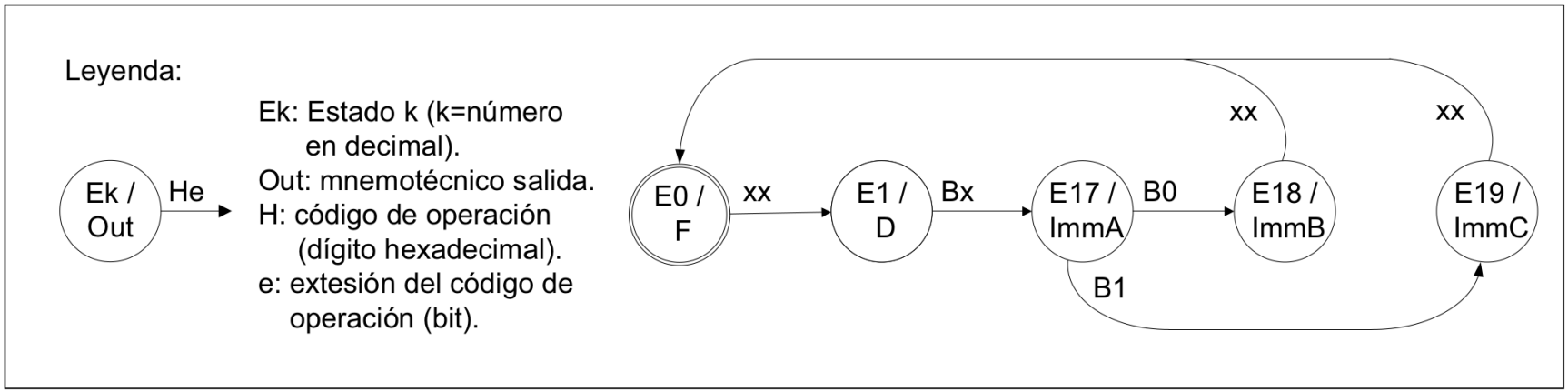
mnemoI16: ROM OUT

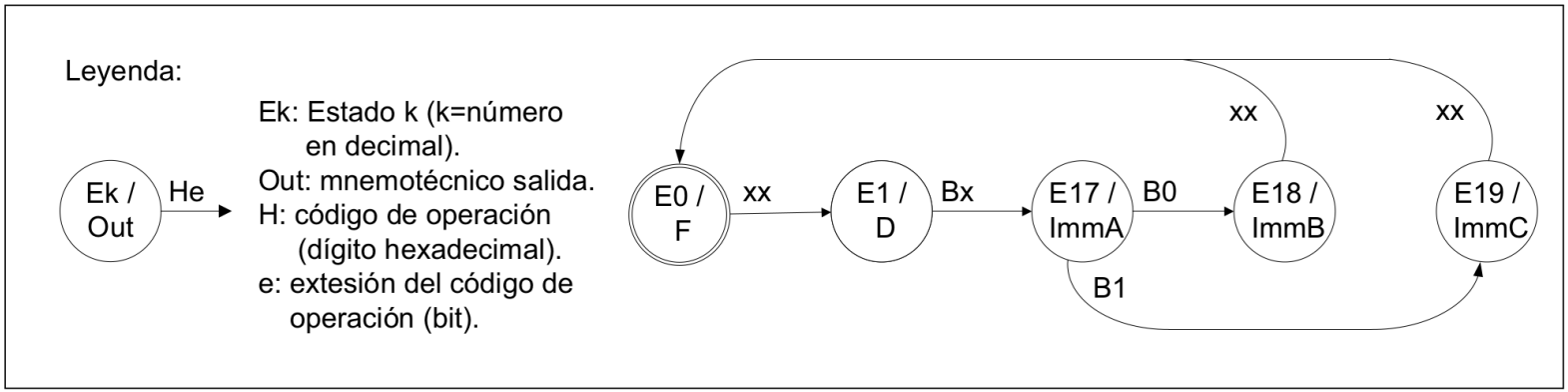
@ROM	Imm16	Bnz	Bz	WrMem	RdIn	WrOut	WrD	LdIr	Byte	R@/Pc	Alu/R@	Pc/Rx	Ry/N	P/I/L/A1	P/I/L/A0	OP1	OP0	MxN1	MxN0	MxF	F2	F1	F0	Mx@D1	Mx@D0	Node
0	0																									F
1	0																									D
2	x																									AI
3	x																									Cmp
4	0																									Addi
5	0																									Addr
6	x																									Ld
7	x																									St
8	x																									Ldb
9	x																									Stb
10	x																									Jalr
11	x																									Bz
12	x																									Bnz
13	0																									Movi
14	0																									Movhi
15	x																									In
16	x																									Out
17	0	1	1	0	0	0	0	0	0	0	1	1	0	x	x	0	0	1	1	1	1	0	0	x	x	ImmA
18	1	0	0	0	0	0	1	x	x	x	x	0	0	0	0	0	0	x	x	0	x	x	x	0	0	ImmB
19	1	0	0	0	0	0	1	x	x	x	x	0	0	0	0	0	1	x	x	0	x	x	x	0	0	ImmC
20..31	x																									Nop

- Completad el fragmento del grafo de estados del circuito secuencial de la unidad de control necesario para ejecutar completamente las nuevas instrucciones. Se da la leyenda del grafo y todos los nodos, pero faltan arcos etiquetas. Dibujad todos los arcos que faltan y todas las etiquetas. No os pedimos que dibujéis el nodo Nop.

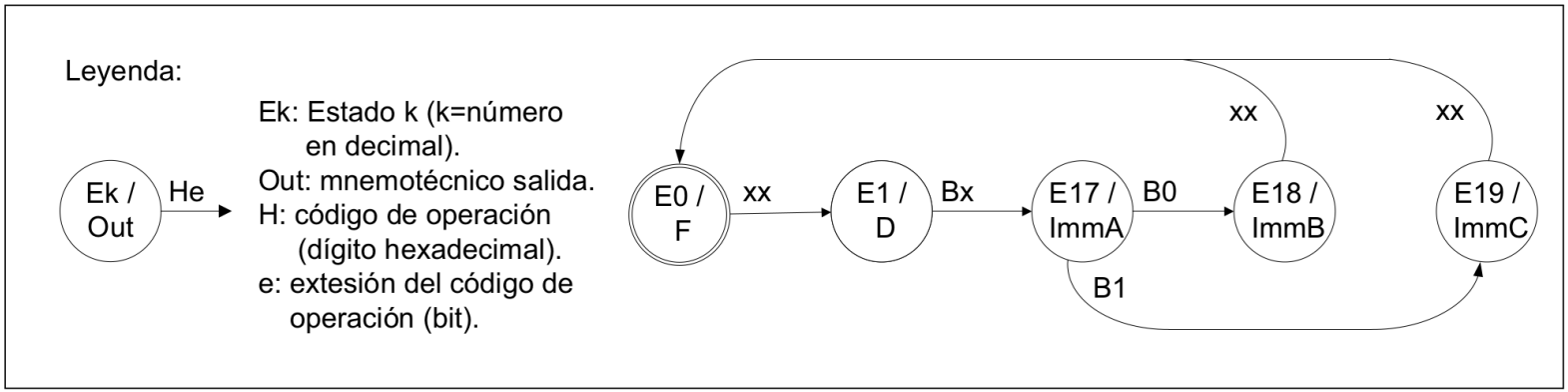


- Completad el fragmento del grafo de estados del circuito secuencial de la unidad de control necesario para ejecutar completamente las nuevas instrucciones. Se da la leyenda del grafo y todos los nodos, pero faltan arcos etiquetas. Dibujad todos los arcos que faltan y todas las etiquetas. No os pedimos que dibujéis el nodo Nop.





- Indicad la dirección o las direcciones (en binario con x cuando sea posible para referirnos a más de una dirección) de la memoria ROM_Q+ y su contenido (en hexadecimal) para implementar correctamente el paso del nodo/estado E1 (D) al E17 (ImmA) y del E17 ((ImmA) al E18 (ImmB).



- Indicad la dirección o las direcciones (en binario con x cuando sea posible para referirnos a más de una dirección) de la memoria ROM_Q+ y su contenido (en hexadecimal) para implementar correctamente el paso del nodo/estado E1 (D) al E17 (ImmA) y del E17 ((ImmA) al E18 (ImmB).
  - D a E17: A les adreces 00001 1011 x, el contingut ha de ser 0x11
  - E17 a E18: a l'adreça 10001 1011 0 el contingut ha de ser 0x12
  - Observació: també seria vàlida intercanviar el paper d'E18 i E19, amb el que existiria una altra solució vàlida a tots els apartats

Exercicis a entregar a Atenea

- Enunciat disponible a Atenea
  - `https://atenea.upc.edu/mod/quiz/view.php?id=2447262`
- Entrega a Atenea fins el dijous 17/12
  - Format PDF
  - Heu d'afegir una nova instrucció de llenguatge màquina SISA
    - Caldrà modificar *hardware* de la UCG/UPG? De quina forma (modificacions respecte a l'esquema lògic original, nous multiplexors, senyals de control, ...)?
    - Com caldrà modificar les ROM Q+ i OUT originals (files/columnes amb valors diferents)?

Índex

Conclusions

- Hem finalitzat el disseny del computador von Neumann
  - Darrera instrucció SISA: `JALR`
  - Determinació del temps de cicle $T_c = 1.400$ u.t.
  - Estudi dels senyals de modificació de l'estat
- Computador von Neumann és versàtil
  - Modificant ROM Q+ i ROM OUT podem incorporar noves instruccions al llenguatge màquina
  - Amb petites modificacions al *hardware* podem incorporar instruccions més potents
- Contesteu el qüestionari ET13c i l'exercici en paper (slide 46).

Referències I

Llevat que s'indiqui el contrari, les figures, esquemes, cronogrames i altre material gràfic o bé han estat extrets de la documentació de l'assignatura elaborada per Juanjo Navarro i Toni Juan, o corresponen a enunciats de problemes i exàmens de l'assignatura, o bé són d'elaboració pròpia.

[1] *Self-Modifying code and avoiding conditionals*, [Online]. Available: https://blog.kartones.net/post/self-modifying-code-and-avoiding-conditionals/.

[2] *Modified Harvard Architecture: Clarifying Confusion*, [Online]. Available: http://ithare.com/modified-harvard-architecture-clarifying-confusion/.

[3] *Core War*, [Online]. Available: https://en.wikipedia.org/wiki/Core_War.

Introducció als Computadors

Tema 13: Computador SISC von Neumann

http://personals.ac.upc.edu/enricm/Docencia/IC/IC13c.pdf

Enric Morancho
(enricm@ac.upc.edu)

Departament d'Arquitectura de Computadors
Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

2020-21, 1^{er} quad.