

Introducció als Computadors

Tema 10: Unitat de Control General

<http://personals.ac.upc.edu/enricm/Docencia/IC/IC10a.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

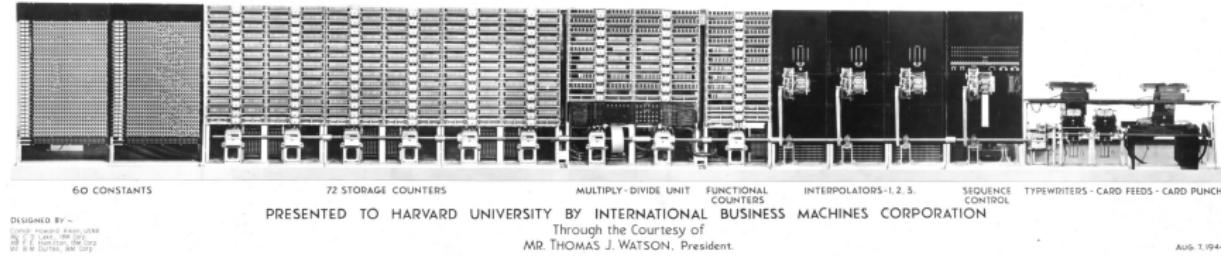
2020-21, 1^{er} quad.

Presentació publicada sota llicència Creative Commons 4.0



Harvard Mark-I (1944-1959)

IBM AUTOMATIC SEQUENCE CONTROLLED CALCULATOR



[1]

Howard H. Aiken
(1900-1973)
Dissenyador



[2]

Grace M. Hopper
(1906-1992)
Programadora



[3]

Programa (amb patches)

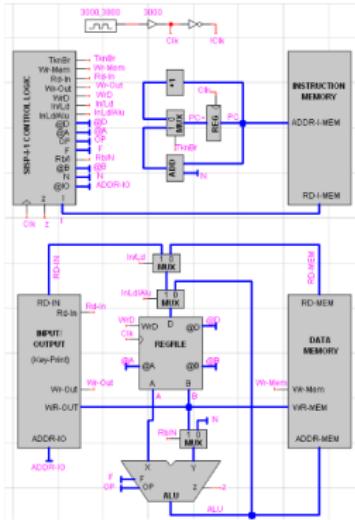


[4]

- Mides: 15 m de llargada, 5 t de pes

● Introducció

- Estructura de la Unitat de Control General
- Creant el llenguatge màquina i l'ensamblador SISA
- Del graf d'estats de la UC al LM SISA
- Exercicis
- Conclusions



Microarquitectura del processador que dissenyarem a IC

Software

Aplicacions

Algoritme

Llenguatge d'alt nivell

Llenguatge ensamblador

Sistema Operatiu

Codi màquina

Instruction Set Architecture (ISA)

Hardware

Microarquitectura

Disseny lògic

Dispositius

Nivell físic



arm

RISC-V

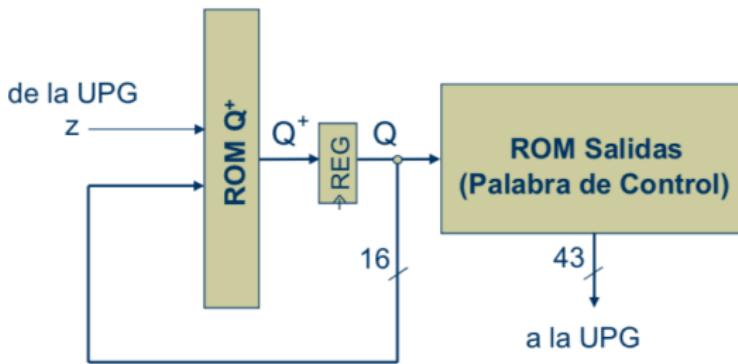
MIPS

ISA: conjunt d'instruccions que accepta el computador

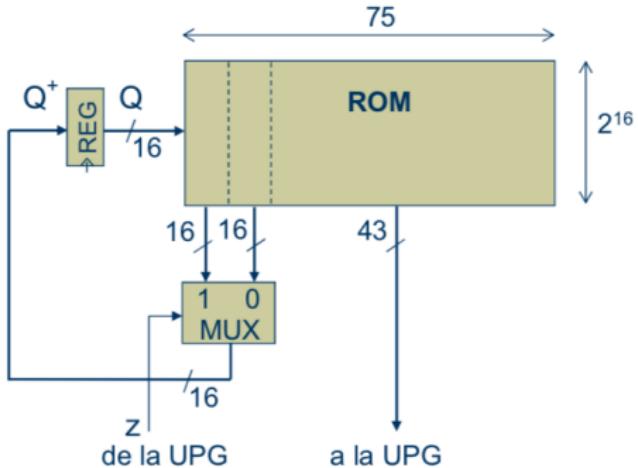
	Tema							
	7	8	9	10	11	12	13	14
Unitat de Control	UCE	UCE	UCE	UCG	UCG	UCG	UCG	UCG
Unitat de Procés	UPE	UPG						
Entrada/Sortida	-	-	IO	IO	IO	IO	IO	IO
Memòria RAM	-	-	-	-	MEM	MEM	MEM	MEM
Harvard unicicle	-	-	-	-	-	✓	-	-
Harvard multicicle	-	-	-	-	-	✓	-	-
Von Neumann	-	-	-	-	-	-	✓	✓
Lleng. assembler	-	-	-	✓	✓	✓	✓	✓

- La UC només té un senyal d'entrada
 - Bit de condició z
 - Cada estat té dos estats futurs possibles
 - Un per a $z=0$ i un altre per a $z = 1$
- Si la UC té k bits d'estat
 - Fins a 2^k estats
 - Considerarem $k = 16$
 - $2^{16} = 65.536$
- Dues opcions de síntesi de la UC
 - Síntesi amb dues ROMs
 - ROM Q^+ (de l'estat següent)
 - ROM de sortides
 - Síntesi amb una ROM combinada

- ROM de l'estat següent
 - 2^{k+1} paraules amb k bits/paraula
- ROM de sortides
 - 2^k paraules amb 43 bits/paraula
- Mida total = $(2^{k+1} \times k) + (2^k \times 43)$ bits
 - $k = 16 \implies$ mida ≈ 600 Kilobytes



- ROM combinada
 - 2^k paraules i $(2 \times k + 43)$ bits/paraula
- Mida = $2^k \times (2 \times k + 43)$ bits
 - $k = 16 \Rightarrow 75$ bits/paraula \Rightarrow mida ≈ 600 Kilobytes
 - Exactament igual que al cas anterior



- A partir d'ara només considerarem aquesta opció

- La memòria utilitzada a la UC pot ser una ROM o una RAM
 - ROM: *Read Only Memory*
 - RAM:
 - El seu contingut es pot modificar per a adaptar-se a un nou problema
- Sigui ROM o RAM, d'aquesta memòria en direm I-MEM
 - *Instruction Memory*
 - Cada paraula de la I-MEM és una **instrucció de llenguatge màquina**
 - A la ROM anterior, cada instrucció de LM ocupa 75 bits!
- Del conjunt d'instruccions que implementen el graf d'estats de la UC en direm **programa en llenguatge màquina**
- El registre d'estat de la UC ...
 - Conté un valor natural entre 0 i $2^k - 1$
 - Identifica la fila de la I-MEM on es troba la instrucció de llenguatge màquina que s'executa al cicle actual
 - **D'aquest registre en direm PC (*Program Counter*)**
 - També IP (*Instruction Pointer*) o IAR (*Instruction Address Register*)

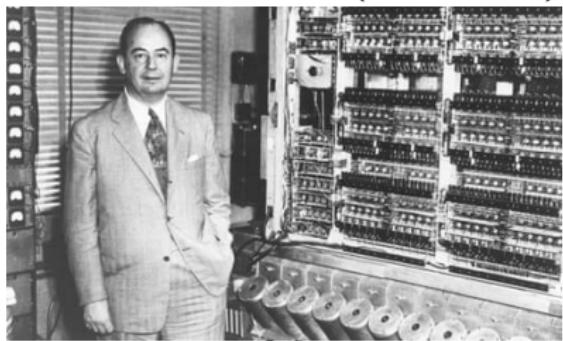
- En funció d'on s'emmagatzema el programa:
 - Model Harvard
 - A una memòria independent de la memòria de dades
 - Model von Neumann
 - A la mateixa memòria on s'emmagatzemen les dades

Harvard Mark I (1944-1959)



[5]

John von Neumann (1903-1957)



[6]

- En funció del propòsit del processador
 - Processadors de propòsit específic (empotrats, *embedded*)
 - Acostumen a seguir model Harvard
 - El programa està a una memòria ROM i sempre serà el mateix
 - Processadors de propòsit general
 - Segueixen model von Neumann
 - Una memòria RAM conté programa i dades
 - El programa es pot canviar quan ho considerem oportú
- A IC...
 - Als temes 10, 11 i 12 seguirem el model Harvard
 - La I-MEM serà una ROM
 - Al tema 11 introduirem la memòria de dades
 - Al tema 13 passarem al model von Neumann
 - Integrarem la I-MEM a la RAM

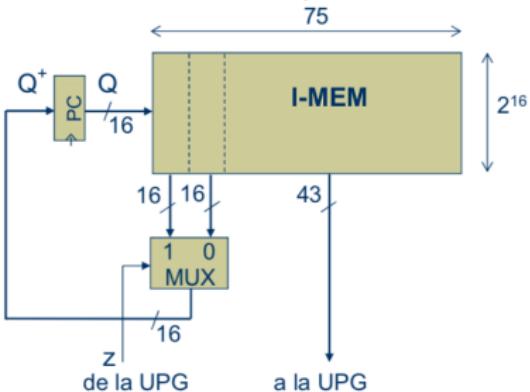
- Reduir la mida de les paraules de la I-MEM
 - Passarem de 75 bits/instrucció a 16 bits/instrucció
 - Crearem el llenguatge màquina SISA
 - *Simple Instruction Set Architecture*
 - Cada instrucció SISA ocuparà 16 bits
 - Per legibilitat, crearem llenguatge *assembler*
 - Convertirem entre *assembler* i llenguatge màquina i a l'inrevés
 - Al registre d'estat de la UC en direm PC (*Program Counter*)
 - Adreça de la I-MEM on es troba la instrucció a executar aquest cicle
- Quin preu pagarem?
 - Caldrà afegir *hardware*
 - **Lògica de control:** a partir dels 16 bits de la instrucció i del bit *z*, generarà els 43 bits de la paraula de control i actualitzarà el registre PC
 - Perdrem paral·lelisme
 - Algunes tasques que la UPG pot realitzar amb una única paraula de control, ara requeriran vàries instruccions LM SISA
 - Aquestes tasques passaran de trigar un cicle a trigar-ne varis (un cicle per a cada instrucció SISA)

- Introducció
- Estructura de la Unitat de Control General
 - Primera aproximació
 - Segona aproximació
- Creant el llenguatge màquina i l'ensamblador SISA
- Del graf d'estats de la UC al LM SISA
- Exercicis
- Conclusions

- Introducció
- Estructura de la Unitat de Control General
 - Primera aproximació
 - Segona aproximació
- Creant el llenguatge màquina i l'ensamblador SISA
- Del graf d'estats de la UC al LM SISA
- Exercicis
- Conclusions

- Seqüenciament:

- Després d'executar una instrucció, quina s'executará al cicle següent?



- En aquest model, el seqüenciament és explícit absolut

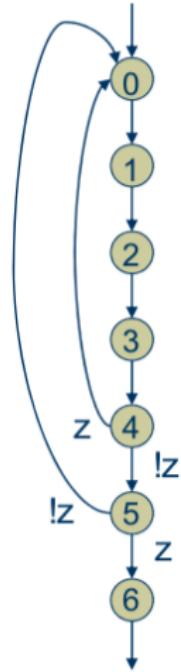
- Explícit:

- La I-MEM emmagatzema els identificadors de les dues possibles instruccions següents (per a $z=0$ i per a $z=1$)

- Absolut:

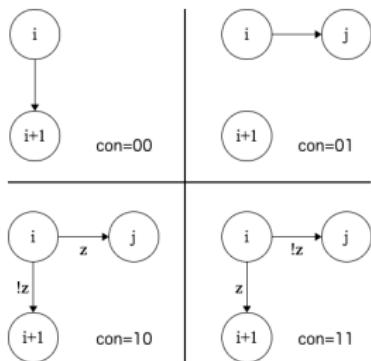
- Els identificadors de les instruccions següents poden corresponder a qualsevol fila de la I-MEM

- A alguns nodes, l'estat següent és independent de z
 - Els dos identificadors tindran el mateix valor a la I-MEM
- És habitual trobar seqüències de nodes que s'executen incondicionalment un darrera de l'altre
 - Si aquests nodes reben identificadors consecutius, l'estat següent es pot calcular sumant 1 a l'estat actual
 - **No caldria emmagatzemar-lo a la I-MEM**
- Típicament, els salts acostumen a implicar "pocs" nodes respecte al node actual
 - En comptes d'especificar l'estat següent de forma absoluta (16 bits) es podria especificar de forma relativa
 - L'estat següent es calcularia sumant un desplaçament (*offset*) de 8 bits a l'identificador d'estat actual
 - Entre -128 i +127 respecte a l'estat actual

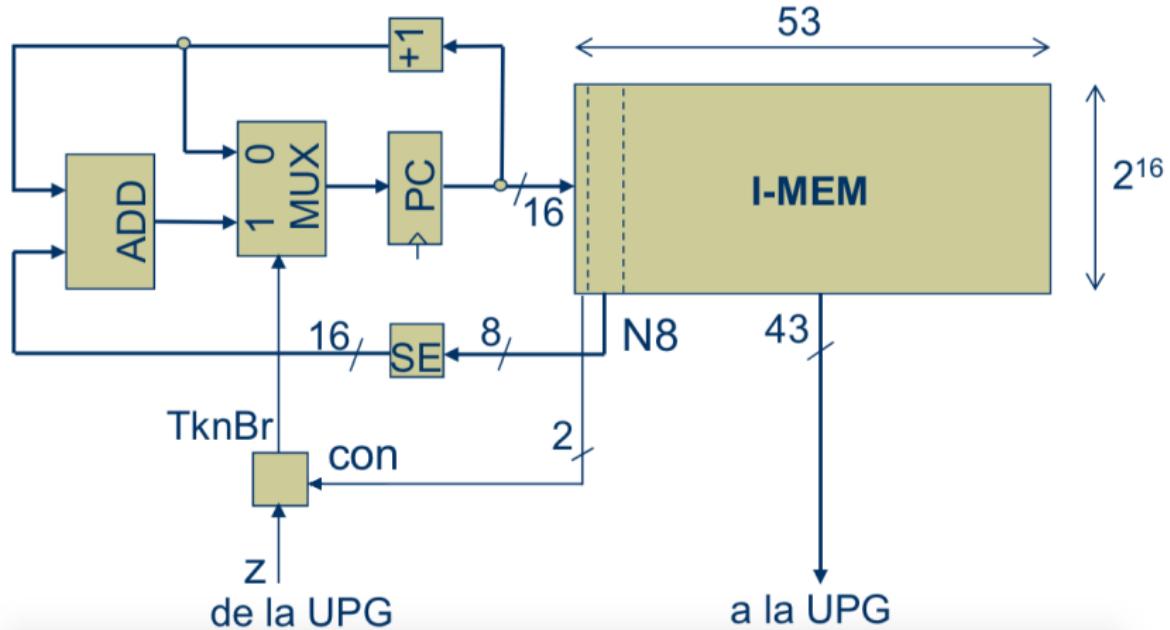


- Considerarem 4 tipus de nodes als grafs:

- $con=00$: Sempre segueixen en seqüència
- $con=01$: Sempre trenquen la seqüència
- $con=10$: Si $z=0$ segueix en seqüència, si $z=1$ trenca la seqüència
- $con=11$: Si $z=1$ segueix en seqüència, si $z=0$ trenca la seqüència



- Substituirem a la I-MEM els dos camps de 16 bits per:
 - con : camp de 2 bits que indica el tipus de node
 - $N8$: camp de 8 bits que indica el desplaçament respecte al PC actual
 - Per poder-lo sumar amb el PC (de 16 bits), li aplicarem una extensió de signe a 16 bits (SE(M8), *Sign Extension*)
- Afegirem hardware perquè a partir d'aquests dos camps, el bit z i el valor actual del registre PC, es calculi el nou valor del registre PC



- Per decidir com s'actualitza el registre PC, farem servir un CLC
 - Entrades: el camp de 2 bits *con* i el bit *z*
 - Sortida: 1 bit (*TakenBranch*)
 - 0: $PC \leftarrow PC + 1$
 - 1: $PC \leftarrow PC + 1 + SE(N8)$
- Taula de veritat:

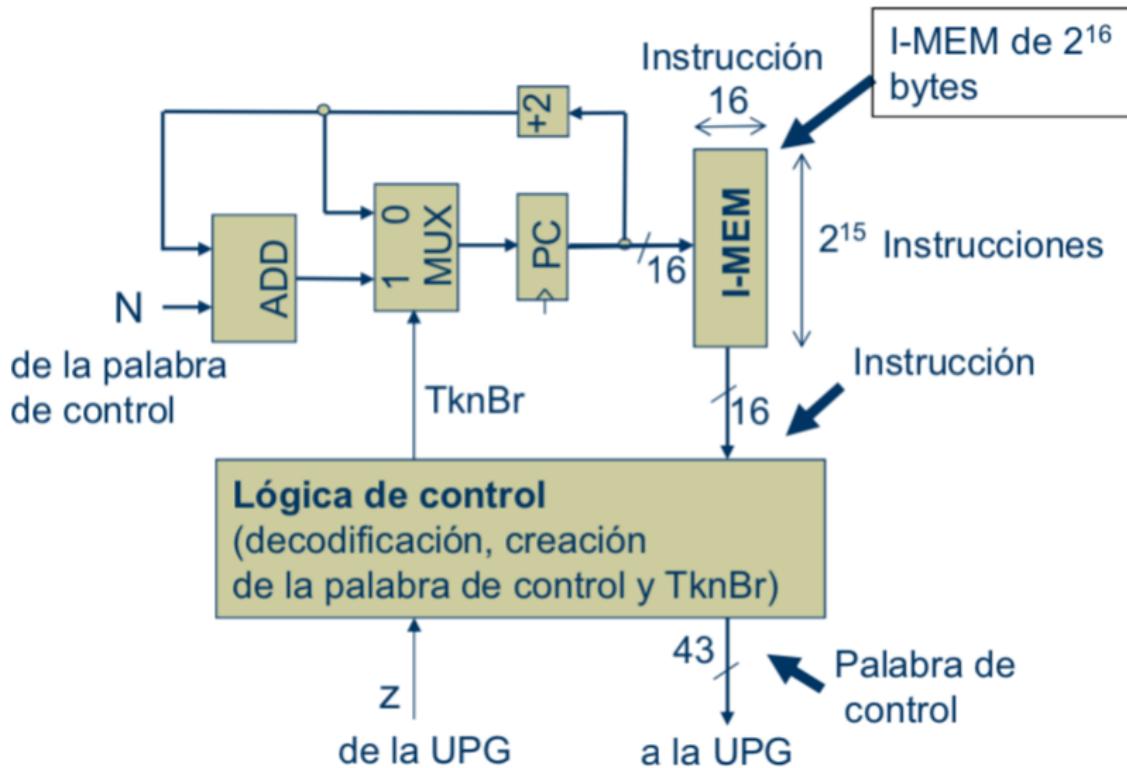
				con₁	con₀	<i>z</i>	TknBr
				0	0	0	0
				0	0	1	0
				0	1	0	1
				0	1	1	1
				1	0	0	0
				1	0	1	1
				1	1	0	1
				1	1	1	0

- Hem passat del seqüènciament **explícit absolut** al **semi-implícit relatiu**
 - A la documentació teniu aquest procés més detallat
 - També es deté en un model intermedi: **semi-implícit absolut**
- Hem reduït els 75 bits per instrucció a 53 bits per instrucció
 - Encara no és suficient, volem únicament 16 bits per instrucció
- Observeu que hi ha instruccions que no utilitzen el camp N8
 - Les que són de tipus $con=00$, seguir sempre en seqüència
- A la segona aproximació...
 - Representació dels bits de la paraula de control de forma més compacta
 - Algunes tasques que la UPG pot fer en un cicle, ara trigaran varis cicles

- Introducció
- Estructura de la Unitat de Control General
 - Primera aproximació
 - Segona aproximació
- Creant el llenguatge màquina i l'ensamblador SISA
- Del graf d'estats de la UC al LM SISA
- Exercicis
- Conclusions

- Cada paraula de la I-MEM contindrà 16 bits
 - Instruccions de LM SISA de 16 bits
 - No podran incorporar valors immediats de 16 bits
 - Ocuparien tots els bits de la instrucció!
 - Compactarem els bits significatius de les paraules de control
 - Moltes accions tenen x a diverses posicions de la paraula de control
- Modificarem per darrer cop el seqüenciament
 - Seqüenciament implícit

- Crearem un CLC **Lògica de control** tal que:
 - A partir de la codificació en 16 bits de la instrucció generi els 43 bits de la paraula de control
 - A partir del bit z, calculat per la UPG, i dels 16 bits de la instrucció generi el senyal $TknBr$
 - Actualització del registre PC
- En cas de ruptures de seqüènciaciament implícit, el desplaçament utilitzat per actualitzar el registre PC formarà part dels 16 bits de la instrucció
 - Eliminarem els camps *con* i *N8* del disseny anterior



- Hem passat de $PC + 1$ a $PC + 2$!!!
- Per compatibilitat amb la memòria de dades que veurem al tema 11, assumim que cada posició de la I-MEM conté un byte (8 bits)
 - Penseu que acabarem integrant la I-MEM a la memòria de dades i la memòria de dades estarà direccionalada a byte
 - Ens adaptem ja a aquesta restricció
- Per tant, el registre PC sempre contindrà un valor parell
- Com les instruccions són de 16 bytes...
 - La instrucció amb $PC=0$ ocupa els bytes 0 i 1 de la I-MEM
 - La instrucció amb $PC=2$ ocupa els bytes 2 i 3 de la I-MEM
 - ...
 - La instrucció amb $PC=2 \cdot j$ ocupa els bytes $2 \cdot j$ i $2 \cdot j + 1$ de la I-MEM
- Com la I-MEM conté 2^{16} bytes, només podrà emmagatzemar 2^{15} instruccions de llenguatge màquina

- Estudiarem com codificar les instruccions amb 16 bits
 - A partir d'aquests 16 bits caldrà generar els 43 bits de la paraula de control de la UPG
- Implementarem el CLC **Lògica de control** per fer aquesta tasca
 - La codificació escollida pot simplificar la implementació de la Lògica de Control

- Introducció
- Estructura de la Unitat de Control General
- Creant el llenguatge màquina i l'ensamblador SISA**
- Del graf d'estats de la UC al LM SISA
- Exercicis
- Conclusions

- Crearem llenguatge màquina SISA
 - *Simple Instruction Set Architecture*
 - La primera versió tindrà 20 instruccions
- Instruccions codificades amb 16 bits
 - Llenguatge màquina: 16 bits
 - Exemple: 0000101010111001 o 0x0AB9
 - Llenguatge ensamblador (*assembler*): mnemotècnics de les instruccions i identificadors de registres
 - Exemple: OR R7, R5, R2
 - Guanyem legibilitat
 - Caldrà saber convertir entre els dos formats
 - Ensamblar: de llenguatge ensamblador a llenguatge màquina
 - Desensamblar: de llenguatge màquina a llenguatge ensamblador
- Passos a partir d'ara:
 - Redefinirem seqüènciament
 - Definirem repertori d'instruccions
 - Codificarem les instruccions

- Fins ara, qualsevol instrucció podia trencar el seqüenciament implícit
 - Però codificar amb 16 bits una operació aritmètica i com trencar el seqüenciament implícit no serà possible
 - Com a mínim, cada registre involucrat necessita 3 bits + especificar l'operació d'ALU necessita 5 bits + el desplaçament del salt són 8 bits ..., com a mínim 19 bits
- Passarem a tenir seqüenciament implícit
 - Totes les instruccions convencionals seguiran el seqüenciament implícit
 - SISA introduirà dues instruccions per trencar-lo
 - Instruccions de salt BZ i BNZ
 - En funció de si un registre val 0 (BZ) o diferent de 0 (BNZ)
 - El desplaçament estarà codificat dins dels 16 bits de la instrucció
- Una acció UPG que pugui trencar el seqüenciament implícit es convertirà en dues instruccions SISA

- Aritmètic-Lògiques (AL):
 - AND, OR, XOR, NOT, ADD, SUB, SHA, SHL
 - Llevat NOT, operen amb dos registres font i actualitzen el registre destí
- Comparació (CMP):
 - CMPLT, CMPLE, CMPEQ, CMPLTU, CMPLEU
 - Anàleg a les Aritmètic-Lògiques
- De salt:
 - BZ, BNZ
- Entrada/Sortida:
 - IN, OUT
 - Lectura/Escriptura d'un *port* d'entrada/sortida
- Càrrega de valors immediats:
 - MOVI, MOVHI
 - Càrrega part baixa (amb extensió de signe)/part alta
- Suma amb valor immediat de fins a 6 bits:
 - ADDI
 - Útil per actualitzar comptadors sense haver de carregar l'increment a un registre

- Els 4 bits de més pes de la instrucció indiquen el codi d'operació
 - En funció del valor del codi d'operació, s'interpretaran la resta de bits
 - En alguns casos representaran identificadors de registres, en altres valors immediats, en altres identificador de *ports*, ...
- SISA defineix tres formats de codificació en funció del nombre de registres involucrats a la instrucció
 - El codi d'operació determina quin format correspon a cada instrucció

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Formato 3-R	c	c	c	c	a	a	a	b	b	b	d	d	d	f	f	f
Formato 2-R	c	c	c	c	a	a	a	d	d	d	n	n	n	n	n	n
Formato 1-R	c	c	c	c	a	a	a	e	n	n	n	n	n	n	n	n

- Format 3-R (2 registres font, Ra i Rb, 1 registre destí, Rd)

Formato 3-Reg	Código de operación				Ra		Rb			Rd			F			
	c	c	c	c	a	a	a	b	b	b	d	d	d	f	f	f
	15		12	11	9	8		6	5		3	2		0		

- Codi d'operació:

- 0000: instruccions aritmèticò-lògiques (AL)
- 0001: instruccions de comparació (CMP)

- El camp F (3 bits) indica quina funció de la ALU cal utilitzar

	000	001	010	011	100	101	110	111
OP=00	AND	OR	XOR	NOT	ADD	SUB	SHA	SHL
OP=01	CMPLT	CMPLE	-	CMPEQ	CMPLTU	CMPLEU	-	-

- Assembler/Semàntica: OPERACIÓ Rd, Ra, Rb

- Rd \leftarrow Ra OPERACIÓ Rb; PC \leftarrow PC + 2;

- Exemples conversió:

- OR R7, R5, R2 \rightsquigarrow 0000 101 010 111 001 \rightsquigarrow 0xAAB9
- CMPLTU R5, R6, R7 \rightsquigarrow 0001 110 111 101 100 \rightsquigarrow 0x1DEC

- Format 1-R (1 registre font, Ra, i un immediat de 8 bits, N8)

Código de operación	Ra	e	N8
Formato 1-R	c c c c a a a e n n n n n n n n		
	15 12 11 9 8 7 0		

- Codi d'operació: 1000 més el bit e (extensió del codi d'operació)

- e=0 → BZ (*Branch on Zero*)
- e=1 → BNZ (*Branch on No Zero*)

- Assembler/Semàntica:

- BZ Ra, N8
 - if (Ra==0) PC ← PC + 2 + 2·SE(N8); else PC ← PC + 2;
- BNZ Ra, N8
 - if (Ra!=0) PC ← PC + 2 + 2·SE(N8); else PC ← PC + 2;

- Exemples conversió:

- BZ R5, -2 ↳ 1000 101 0 1111 1110 ↳ 0x8AFE
- BNZ R3, 12 ↳ 1000 011 1 0000 1100 ↳ 0x870C

- Format 1-R (1 registre, Ra o Rd, i un immediat de 8 bits, N8)

Código de operación	Ra/Rd	e	N8
Formato 1-R	c c c c a/d a/d a/d e n n n n n n n n n n		
	15 12 11 9 8 7		0

- Codi d'operació: 1010 més el bit e (extensió del codi d'operació)

- e=0 → IN
- e=1 → OUT

- Assembler/Semàntica:

- IN Rd, N8
 - $Rd \leftarrow \text{InputPort}[N8]; \quad PC \leftarrow PC + 2;$
- OUT N8, Ra
 - $\text{OutputPort}[N8] \leftarrow Ra; \quad PC \leftarrow PC + 2;$

- Exemples conversió:

- IN R3, 15 $\rightsquigarrow 1010\ 011\ 0\ 0000\ 1111 \rightsquigarrow 0xA60F$
- OUT 0x24, R1 $\rightsquigarrow 1010\ 001\ 1\ 0010\ 0100 \rightsquigarrow 0xA324$

- Format 1-R (1 registre destí, Rd, i un immediat de 8 bits, N8)

Código de operación	Rd	e	N8
Formato 1-R	c c c c d d d e n n n n n n n n		
	15 12 11 9 8 7 0		

- Codi d'operació: 1001 més el bit e (extensió del codi d'operació)

- e=0 → MOVI (*Move Immediate*)
- e=1 → MOVHI (*Move High Immediate*)

- Assembler/Semàntica:

- MOVI Rd, N8
 - Rd ← SE(N8); PC ← PC + 2;
- MOVHI Rd, N8
 - Rd<15..8> ← N8; PC ← PC + 2;

- Exemples conversió:

- MOVI R2, 0xAF ↼ 1001 010 0 1010 1111 ↼ 0x94AF
- MOVHI R1, -2 ↼ 1001 001 1 1111 1110 ↼ 0x93FE

- Format 2-R (registres: font, Ra, destí, Rd, i immediat de 6 bits, N6)

Formato 2-R	Código de operación	Ra	Rd	N6
	c c c c a a a d d d n n n n n n	15 12 11 9 8 6 5 0		

- Codi d'operació: 0010
- Assembler/Semàntica: ADDI Rd, Ra, N6
 - $Rd \leftarrow Ra + SE(N6); PC \leftarrow PC + 2;$
- Exemples conversió:
 - ADDI R3, R5, 15 \rightsquigarrow 0010 101 011 001111 \rightsquigarrow 0x2ACF
 - ADDI R3, R5, -1 \rightsquigarrow 0010 101 011 111111 \rightsquigarrow 0x2AFF

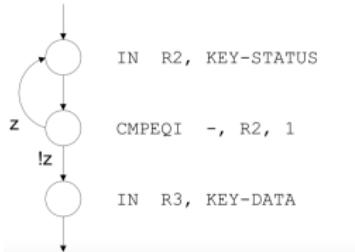
- Hem creat el llenguatge màquina SISA
 - Codifica les instruccions amb 16 bits
 - El llenguatge assembler facilita la legibilitat del llenguatge màquina
- La Lògica de control generarà la paraula de control de 43 bits a partir d'aquests 16 bits
- Restriccions que imposa SISA
 - No permet operar i trencar el seqüenciament en un únic cicle
 - No pot inhibir l'escriptura al registre destí
 - No opera amb immediats de 16 bits
 - Les instruccions MOVI, MOVHI ho fan amb valors immediats de 8 bits
 - La instrucció ADDI ho fa amb valors immediats de 6 bits

- Als exàmens la tindreu a un xuletari, no cal memoritzar-la

15 14 13 12 11 10	φ φ φ φ φ φ	0 0 1 1 0 0	0 0 1 1 0 1	0 0 d d d d	0 0 d d d d	0 0 n n n n	0 0 n n n n	Name	Mnemonic	Format
0 0 0 0	a a a b b b	0 0 0 1	d d d f f f					Logic and Aritmetic Operations	AND, OR, XOR, NOT, ADD, SUB, SHA, SHL	
0 0 0 1	a a a b b b	0 0 1 0	d d d f f f					Compare Signed and Unsigned	CMPLT, CMPLE, -, CMPEQ, CMPLTU, CMPLEU, -, -	3R
0 0 1 0	a a a d d d	0 0 1 1	n n n n n n					Add Immediate	ADDI	
0 0 1 1		0 1 0 0								
0 1 0 0		0 1 0 1								
0 1 0 1		0 1 1 0								
0 1 1 0		0 1 1 1								
0 1 1 1								Branch future extension		
								BZ		
								BNZ		
1 0 0 0	a a a 0	1 0 0 1	n n n n n n n n	d d d 0	a a a 1	n n n n n n n n	d d d	Branch on Zero	MOVI	
								Branch on Not Zero	MOVHI	
								Move Immediate		
								Move Immediate High		
1 0 1 0	d d d 0	1 0 1 1	x x x x x x x x x x	a a a 1				Input	IN	
								Output	OUT	
1 0 1 1	x x x x x x x x x x							Future extensions		
1 1 x x										

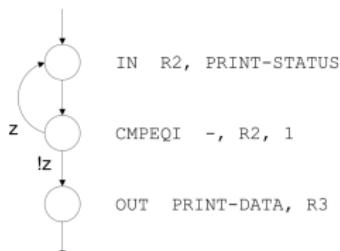
- Al tema 9 calien 3 nodes al graf d'estats.
- SISA també necessita 3 instruccions
 - Aprofitem que registre d'status val 0 o 1
 - Mentre valgui 0, cal fer l'espera activa
 - El desplaçament de salt, -2, es calcula relatiu a la instrucció següent
- Lectura

```
IN  R2 , KEY-STATUS
BZ  R2 , -2
IN  R3 , KEY-DATA
```



- Escripció

```
IN  R2 , PRINT-STATUS
BZ  R2 , -2
OUT PRINT-DATA , R3
```



- Introducció
- Estructura de la Unitat de Control General
- Creant el llenguatge màquina i l'ensamblador SISA
- Del graf d'estats de la UC al LM SISA**
- Exercicis
- Conclusions

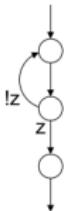
- Moltes accions de la UPG equivalen a 1 instrucció de LM SISA
- Però d'altres poden requerir vàries instruccions SISA
 - Operar amb immediats immediats
 - Cal carregar el valor immediat a un registre (llevat ADDI)
 - Carregar valors immediats que requereixen més de 8 bits en Ca2
 - Calen dues instruccions (una per a la part alta i una altra per a la baixa)
 - Salts condicionals
 - Cal una per guardar el resultat de la comparació i una altra per al salt
 - Salts incondicionals
 - Cal una instrucció de salt
 - No es poden fer accions en paral·lel
 - Cada acció s'haurà de fer amb una (o més) instruccions SISA

Grafo de estados para la UPG		Ensamblador SISA
a)	 ADD R3, R4, R5	ADD R3, R4, R5
b)	 SHAI R4, R3, -3	MOVI R7, -3 SHA R4, R3, R7
c)	 IN R0, 1 // OUT 1, R6	IN R0, 1 OUT 1, R6
d)	 MOVEI R5, 0x3AF6	MOVI R5, 0xF6 MOVHI R5, 0x3A
e)	 MOVEI R3, -16	MOVI R3, 0xF0

- a) L'acció es pot representar amb una instrucció de LM
- b) L'acció es representa amb dues instruccions de LM
 - Una per carregar -3 a un registre
 - Una altra per fer el desplaçament
- c) Les accions en paral·lel es representen amb dues instruccions de LM
 - Una per a cada acció
- d) L'acció es representa amb dues instruccions de LM
 - Una per carregar la part baixa del valor
 - Una altra per carregar la part alta
- e) L'acció es representa amb una instrucció de LM
 - Perquè la constant en Ca2 requereix 8 bits o menys

Grafo de estados para la UPG

f)

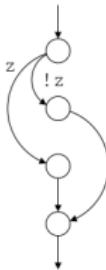


ADD R0, R0, R1
SUBI R2, R2, 1
MOVE R6, R0

Ensamblador SISA

ADD	R0, R0, R1
ADDI	R2, R2, -1
BNZ	R2, -3
ADDI	R6, R0, 0

g)



CMPLEUI -, R1, 378
SUBI R3, R3, 1
ADD R3, R3, R4

MOVI	R7, 0x7A
MOVHI	R7, 0x01
CMPLEU	R7, R1, R7
BZ	R7, 2
ADDI	R3, R3, -1
BNZ	R7, 1
ADD	R3, R3, R4

- ① El graf de tres estats necessita quatre instruccions de LM
 - Cal una instrucció de LM per a implementar el salt
 - El MOVE l'implementem amb un ADDI R6, R0, 0
 - També es podia haver fet amb una OR R6, R0, R0
- ② El graf de tres estats necessita set instruccions de LM
 - Dues instruccions per carregar la constant a un registre
 - Una instrucció per avaluar la condició
 - Una instrucció per realitzar el salt condicional
 - Una instrucció per al decrement
 - Una instrucció per al salt incondicional
 - Una instrucció per a la darrera suma

- Introducció
- Estructura de la Unitat de Control General
- Creant el llenguatge màquina i l'ensamblador SISA
- Del graf d'estats de la UC al LM SISA
- Exercicis
- Conclusions

- Ensamblar, desensamblar instruccions SISA

- Ensamblar:

- XOR R4, R2, R5
 - BZ R3, -2
 - IN R5, 0x12
 - MOVI R6, -3

- Desensamblar:

- 0x0564
 - 0x2345
 - 0x81F4
 - 0xB345

- Ensamblar, desensamblar instruccions SISA

- Ensamblar:

- XOR R4, R2, R5 \rightsquigarrow 0000 010 101 100 010 \rightsquigarrow 0x0562
 - BZ R3, -2
 - IN R5, 0x12
 - MOVI R6, -3

- Desensamblar:

- 0x0564 \rightsquigarrow ADD R4, R2, R5
 - 0x2345
 - 0x81F4
 - 0xB345

- Com modifica l'estat del computador l'execució d'una instrucció SISA
 - Quins registres, *ports* canvien de valor un cop executada
- Si l'estat inicial dels registres és R0=0x0000, R1=0x0001, ... R7=0x0007, PC=0x12AE i els ports d'entrada tenen el valor 0x1234, indiqueu quines modificacions es produiran a l'estat del computador després d'executar cadascuna de les següents instruccions SISA (els apartats són independents).
 - ADD R7, R2, R4
 - SHA R6, R2, R1
 - CMPEQ R2, R3, R4
 - BZ R1, +3
 - BNZ R1, -2
 - IN R4, 26
 - OUT 13, R2
 - MOVI R5, -4
 - MOVHI R5, -4
 - ADDI R4, R5, -7

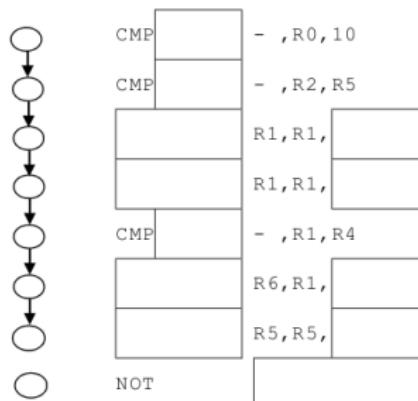
- Com modifica l'estat del computador l'execució d'una instrucció SISA
 - Quins registres, *ports* canvién de valor un cop executada
- Si l'estat inicial dels registres és R0=0x0000, R1=0x0001, ... R7=0x0007, PC=0x12AE i els ports d'entrada tenen el valor 0x1234, indiqueu quines modificacions es produiran a l'estat del computador després d'executar cadascuna de les següents instruccions SISA (els apartats són independents).
 - ADD R7, R2, R4 \rightsquigarrow R7 \leftarrow 0x0006; PC \leftarrow 0x12B0;
 - SHA R6, R2, R1 \rightsquigarrow R6 \leftarrow 0x0004; PC \leftarrow 0x12B0;
 - CMPEQ R2, R3, R4 \rightsquigarrow R2 \leftarrow 0x0000; PC \leftarrow 0x12B0;
 - BZ R1, +3 \rightsquigarrow PC \leftarrow 0x12B0;
 - BNZ R1, -2 \rightsquigarrow PC \leftarrow 0x12AC;
 - IN R4, 26 \rightsquigarrow R4 \leftarrow 0x1234; PC \leftarrow 0x12B0;
 - OUT 13, R2 \rightsquigarrow OutPort[13] = 0x0002; PC \leftarrow 0x12B0;
 - MOVI R5, -4 \rightsquigarrow R5 \leftarrow 0xFFFF; PC \leftarrow 0x12B0;
 - MOVHI R5, -4 \rightsquigarrow R5 \leftarrow 0xFC05; PC \leftarrow 0x12B0;
 - ADDI R4, R5, -7 \rightsquigarrow R4 \leftarrow 0xFFFF; PC \leftarrow 0x12B0;

- Donat un codi escrit en llenguatge C, expressar-lo com a accions de la UPG i després com a instruccions SISA
 - E3-Q1-1718 (assumiu que totes les dades són naturals)

```

while ((R0<10) || (R2!=R5)) {
    R1=R1/16+R5;
    if (R1>R4)
        R6=XOR(R1,0x0440);
    R5--;
}
R3=not (R6);

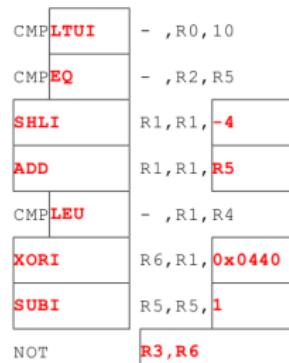
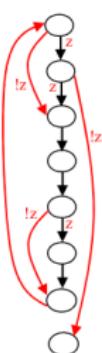
```



@I-Mem	
0x0000	_____ R7, 10
0x0002	CMP _____ R7, R0, R7
0x0004	B _____ R7, _____
0x0006	CMP _____ R7, R2, R5
0x0008	B _____ R7, _____
0x000A	_____ R7, _____
0x000C	_____ R1, R1, _____
0x000E	ADD R1, R1, _____
0x0010	CMP _____ R7, R1, R4
0x0012	B _____ R7, _____
0x0014	MOVI R7, _____
0x0016	S _____ R7, R7, R7
0x0018	_____ R6, R1, R7
0x001A	_____ R5, R5, _____
0x001C	B _____ _____, _____
0x001E	_____ R3, _____

- Donat un codi escrit en llenguatge C, expressar-lo com a accions de la UPG i després com a instruccions SISA
 - E3-Q1-1718 (assumiu que totes les dades són naturals)

```
while ((R0<10) || (R2!=R5)) {
    R1=R1/16+R5;
    if (R1>R4)
        R6=XOR(R1,0x0440);
    R5--;
}
R3=not (R6);
```



@I-Mem	
0x0000	MOVI R7, 10
0x0002	CMP LTU R7, R0, R7
0x0004	BNZ R7, 2
0x0006	CMP EQ R7, R2, R5
0x0008	BNZ R7, 10
0x000A	MOVI R7, -4
0x000C	SHL R1, R1, R7
0x000E	ADD R1, R1, R5
0x0010	CMP LEU R7, R1, R4
0x0012	BNZ R7, 3
0x0014	MOVI R7, 0x44
0x0016	SHL/A R7, R7, R7
0x0018	XOR R6, R1, R7
0x001A	ADDI R5, R5, -1
0x001C	BNZ R7, -15
0x001E	NOT R3, R6

- Introducció
- Estructura de la Unitat de Control General
- Creant el llenguatge màquina i l'ensamblador SISA
- Del graf d'estats de la UC al LM SISA
- Exercicis
- Conclusions

- Hem creat el llenguatge màquina SISA
 - Una acció de la UPG \approx una instrucció LM SISA
 - Tot i que algunes accions de la UPG poden correspondre a varíes instruccions SISA
 - Les instruccions en LM SISA es codifiquen amb 16 bits
 - El llenguatge *assembler* facilita la legibilitat del llenguatge màquina
 - Cal dominar la semàntica de les instruccions SISA
- Veurem que la lògica de control generarà els 43 bits de la paraula de control a partir dels 16 bits de la codificació SISA
 - Caldrà ampliar l'ALU per implementar la instrucció MOVHI
- No oblideu realitzar el qüestionari d'Atenea ET10a

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] [Online]. Available: <http://sites.harvard.edu/~chsi/markone>.
- [2] [Online]. Available: https://en.wikipedia.org/wiki/Howard_H._Aiken#/media/File:Aiken.jpeg.
- [3] [Online]. Available: <http://history-computer.com/ModernComputer/Software/FirstCompiler.html>.
- [4] ArnoldReinhold, 2014. [Online]. Available:
https://commons.wikimedia.org/wiki/File:Harvard_Mark_I_program_tape.agr.jpg.
- [5] [Online]. Available:
<https://www.britannica.com/technology/Harvard-Mark-I/images-videos/media/44895/19205>.
- [6] [Online]. Available:
<https://www.theguardian.com/technology/2012/feb/26/first-computers-john-von-neumann>.

Introducció als Computadors

Tema 10: Unitat de Control General

<http://personals.ac.upc.edu/enricm/Docencia/IC/IC10a.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.

Presentació publicada sota llicència Creative Commons 4.0

