Introducció als Computadors

Tema 13: Computador SISC von Neumann

http://personals.ac.upc.edu/enricm/Docencia/IC/IC13a.pdf

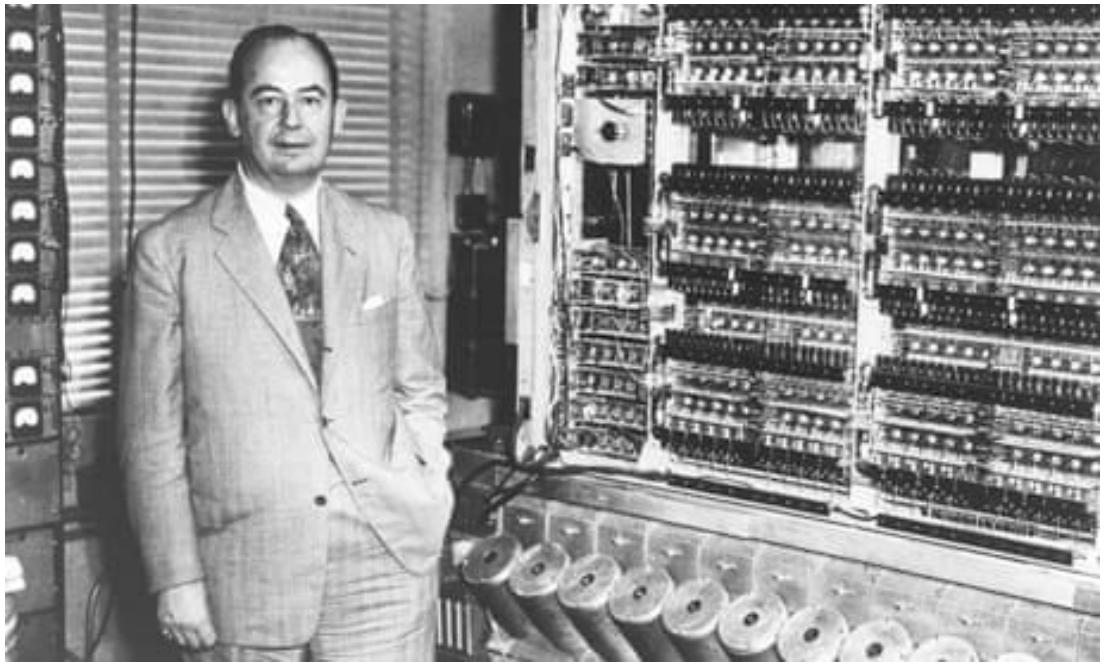
Enric Morancho
(enricm@ac.upc.edu)

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.

John von Neumann (1903-1957)



[1]

- Hongarès / Nord-americà
- Contribucions destacades a diversos camps
  - Física, matemàtiques, informàtica, economia, ...el darrer polímata?
- Idea clau per a nosaltres: la mateixa memòria conté dades i codi
  - IAS machine (Institute for Advanced Study (IAS) in Princeton) [2]
    - 40-bit word. 20-bit instructions. 5.1KB memory (1.024 words). Ca2.
    - Development: 1945-51. Operational: 1951-58
  - El codi es podrà manipular com a una dada més

Índex

Full de ruta

	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	UPG	UPG	UPG	UPG	UPG	UPG
Entrada/Sortida	-	-	IO	IO	IO	IO	IO	IO
Memòria RAM	-	-	-	-	MEM	MEM	MEM	MEM
Harvard unicicle	-	-	-	-	-	✓	-	-
Harvard multicicle	-	-	-	-	-	✓	-	-
Von Neumann	-	-	-	-	-	-	✓	✓
Lleng. *assembler*	-	-	-	✓	✓	✓	✓	✓

Comparativa models de computador

	Harvard (uni, multicicle)	von Neumann (multicicle)
Eficiència temporal	(uni) $t_{mig/inst} = 3.000 u.t.$ $2.250 \leq t_{mig/inst} \leq 3.000$	$4.200 \leq t_{mig/inst} \leq 5.600$
Eficiència espacial	No poden reutilitzar UF's	Pot reutilitzar UF's
Paraula control	Única per instrucció	**Vàries per instrucció**
Versatilitat	No (requereix afegir HW)	Sí

Objectius del tema

- Dissenyar el computador von Neumann
  - Modificacions a la UCG i a la UPG
- Completar el llenguatge màquina SISA
  - Afegirem instrucció `JALR` (*Jump and Link Register*)
  - Imprescindible per a expressar en LM crides/retorns a subrutines
- Estudiar temps de cicle i senyals de modificació de l'estat

Índex

Visió global

- Reutilitza recursos *hardware*
  - Només tindrà un mòdul de memòria
    - Emmagatzemarà codi i dades
    - Per executar `LD`, `LDB`, `ST`, `STB` caldrà fer dos accessos a memòria
  - L'ALU serà l'únic bloc aritmètic del computador
    - També s'encarregarà d'actualitzar el PC (PC+2 i PC + 2·N)
- Implicacions de la reutilització de recursos...
  - Processador multicicle
    - Per poder utilitzar un recurs per a vàries tasques, cada tasca s'ha de fer en un cicle diferent
    - Necessitarem afegir registres per a mantenir estables els resultats calculats a cada cicle
  - La paraula de control variarà cada cicle d'execució d'una instrucció
    - Seran generades per la unitat de control, que serà un CLS

Fases d'execució de cada instrucció

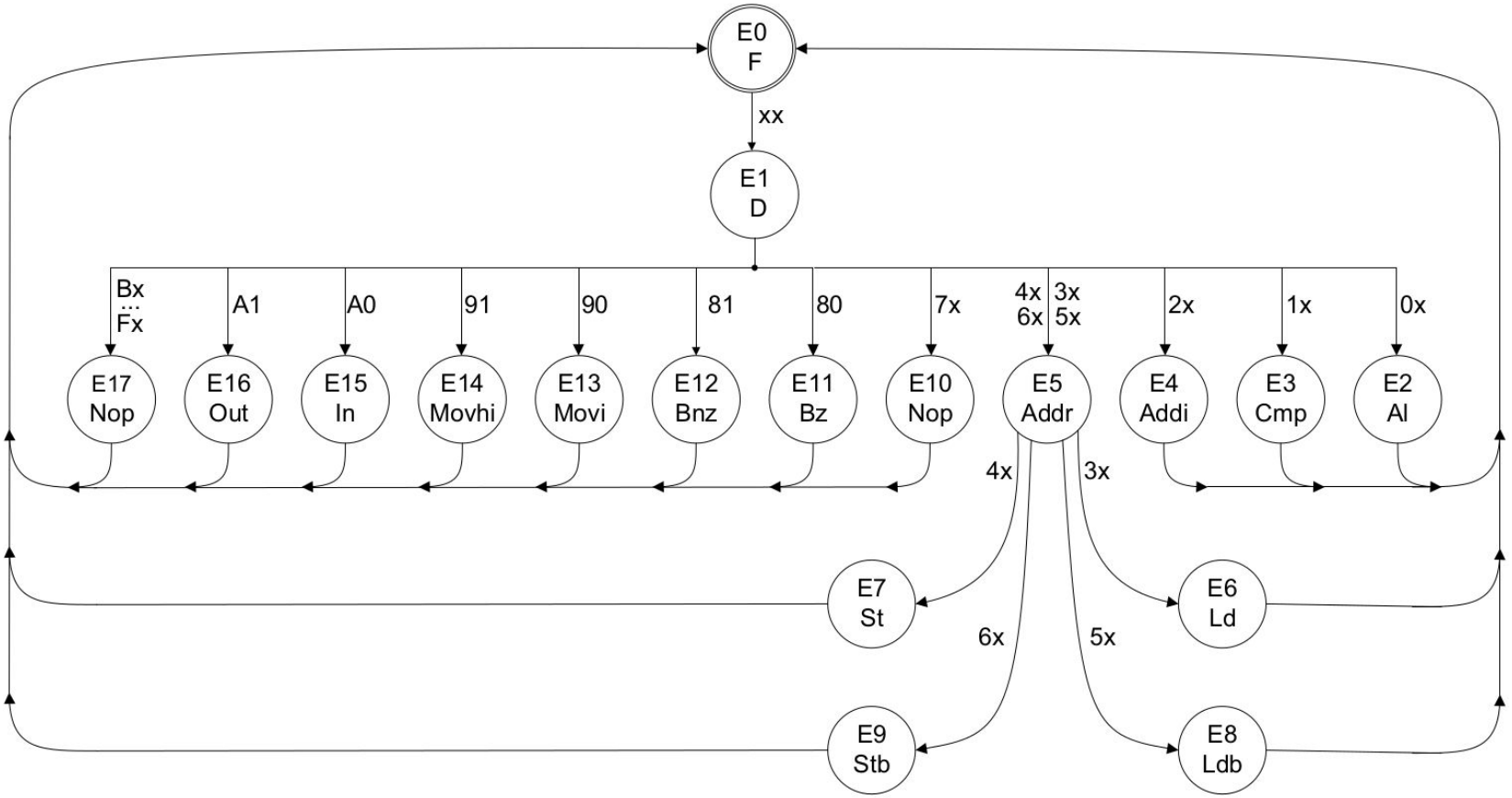
1. *Fetch* (Lectura)
   - Comú per a tots els codis d'operació
   - Llegeix una instrucció SISA de memòria i la guarda al registre IR
     - *Instruction Register*
   - Durarà un cicle
2. *Decode* (Descodificació)
   - Comú per a tots els codis d'operació
   - Analitzant el codi d'operació, decideix què és farà a la següent fase
   - Especulativament, avançarà algunes tasques
   - Durarà un cicle
3. *Execute and Update state* (Càlcul i actualització de l'estat)
   - Realitza el càlcul propi de la instrucció
   - Actualitza l'estat del computador
     - REGFILE, PC, memòria, *ports* d'entrada/sortida
   - Depenent del codi d'operació pot durar un o més cicles

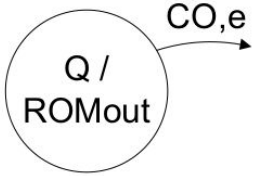
Informació d'estat a les UCs

- La UC del computador Harvard unicicle no tenia estat
  - Era un CLC
  - Al mateix cicle es llegia la instrucció de la I-MEM i es calculava la paraula de control corresponent
  - La paraula de control depenia únicament del codi d'operació
- La UC del computador Harvard multicicle tenia 2 bits d'estat
  - Permetia saber quan una instrucció arribava al darrer cicle d'execució
  - Llevat els bits de modificació d'estat, la paraula de control no canviava al llarg dels 3/4 cicles d'execució
- La UC de computador von Neumann tindrà 5 bits d'estat
  - Fins a $2^5 = 32$ estats possibles
  - Indica quin tipus d'instrucció està en execució i en quina fase es troba
  - **Cada estat de la UC tindrà associada una paraula de control**
  - La paraula de control dependrà del codi d'operació i de l'estat de la UC

Graf d'estats de la UC (sense JALR)

- CLS amb $n = 5$ $(I_{15}, I_{14}, I_{13}, I_{12}, I_8)$, $k = \lceil log_2 18 \rceil = 5$, $m =$?





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

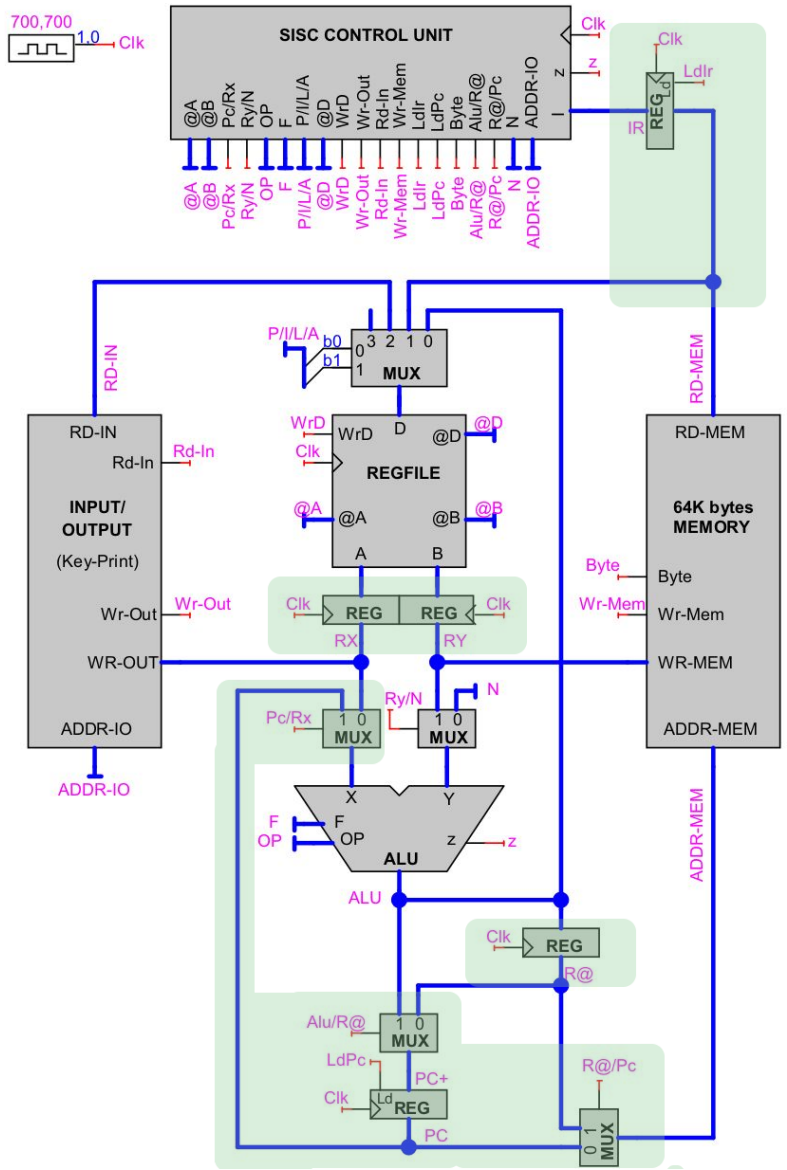
Correspondència instruccions SISA i nodes

- Totes les instruccions passen pels nodes *Fetch* i *Decode*
- Nodes de càlcul i actualització de l'estat

Instrucció SISA	Node(s) al graf
AND, OR, XOR, NOT, ADD, SUB, SHA, SHL	Al
CMPLT, CMPLE, CMPEQ, CMPLTU, CMPLEU	Cmp
ADDI	Addi
BZ	Bz
BNZ	Bnz
MOVI	Movi
MOVHI	Movhi
IN	In
OUT	Out
LD	Addr i Ld
LDB	Addr i Ldb
ST	Addr i St
STB	Addr i Stb

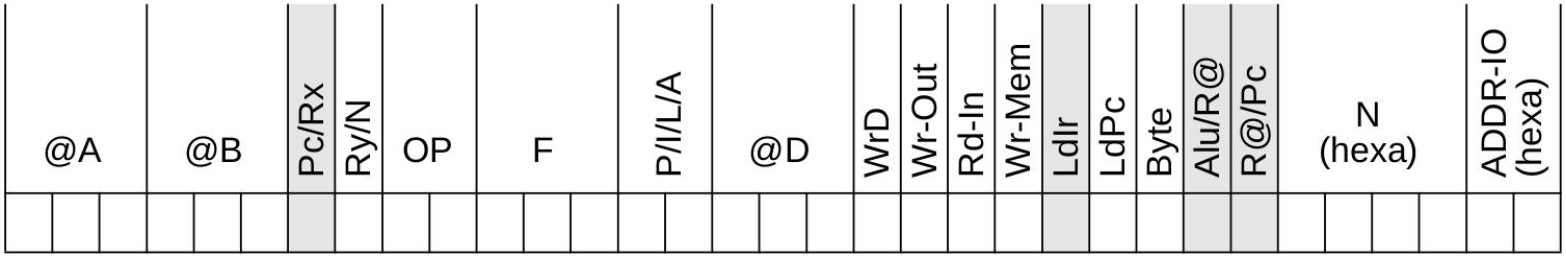
Unitat de procés (sense JALR)

- Nous registres
  - IR : *Instruction Register*
    - Amb senyal de càrrega (LdIr)
    - Guarda instrucció SISA
  - RX, RY
    - Operands llegits del REGFILE
  - R@
    - Adreça de memòria o nou PC
  - **Invisibles al programador**
- Nous multiplexors
  - Pc/Rx
    - L'ALU calcularà el nou PC
  - R@/PC
    - Memòria conté codi i dades
  - Alu/R@
    - Actualització del PC
- Elimina sumador i incrementador PC



Paraula de control

- 51 bits



  - LdIr: senyal de càrrega del registre IR
  - Pc/Rx, Alu/R@, R@/Pc: senyals de control dels nous multiplexors
  - LdPc: senyal de càrrega del registre PC
    - Eliminem el senyal TknBr

Índex

UC: paraules de control compactes

Estado		Acciones	Palabra de control compactada
0	F	Búsqueda de la Instr.: IR ← Mem_w[PC] // Incremento del 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.
1	D	Decodificación. Calculo @ salto tomado: R@ ← PC + SE(N8)*2 // Lectura de registros. (RX ← Ra) // (RY ← Rb)	La decodificación no requiere ninguna acción en la UPG por lo que se usa la UPG en este ciclo para adelantar trabajo que pueda ser útil. Este cálculo solo será util si la instrucción es BZ o BNZ. N=SE(IR<7..0>)*2, Pc/Rx=1, Ry/N=0, OP=00, F=100. Estas acciones se realizan sin tener que especificar nada en la palabra de control ya que RX y RY no tienen señal de permiso de escritura y @A y @B se generan directamente de los campos de bits del registro de instrucción IR<11..9> e IR<8..6>, respectivamente, en cada ciclo del grafo. Por ello las especificamos aquí entre paréntesis y ya no las especificaremos de ninguna forma en el resto de nodos.
2	Al	Rd ← RX Al RY	Pc/Rx=0, Ry/N=1, OP=00, F=IR<2..0>, P/I/L/A=00, WrD=1, @D=IR<5..3>.
3	Cmp	Rd ← RX Cmp RY	Pc/Rx=0, Ry/N=1, OP=01, F=IR<2..0>, P/I/L/A=00, WrD=1, @D=IR<5..3>.
4	Addi	Rd ← RX + SE(N6)	N=SE(IR<5..0>), Pc/Rx=0, Ry/N=0, OP=00, F=100, P/I/L/A=00, WrD=1, @D=IR<8..6>.
5	Addr	R@ ← RX + SE(N6)	N=SE(IR<5..0>), Pc/Rx=0, Ry/N=0, OP=00, F=100.
6	Ld	Rd ← Mem_w[R@]	R@/Pc=1, Byte=0, P/I/L/A=01, WrD=1, @D=IR<8..6>.
7	St	Mem_w[R@] ← RY	R@/Pc=1, Byte=0, Wr-Mem=1.
8	Ldb	Rd ← Mem_b[R@]	R@/Pc=1, Byte=1, P/I/L/A=01, WrD=1, @D=IR<8..6>.
9	Stb	Mem_b[R@] ← RY<7..0>	R@/Pc=1, Byte=1, Wr-Mem=1.
10	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>.
11	Bz	if (RX == 0) PC ← R@	Pc/Rx=0, OP=10, F=000, Alu/R@=0, LdPc=z.
12	Bnz	if (RX != 0) PC ← R@	Pc/Rx=0, OP=10, F=000, Alu/R@=0, LdPc=!z.
13	Movi	Rd ← SE(N8)	N=SE(IR<7..0>), Ry/N=0, OP=10, F=001, P/I/L/A=00, WrD=1, @D=IR<11..9>.
14	Movhi	Rd ← (N*(2^8)) | RX<7..0>	N=SE(IR<7..0>), Pc/Rx=0, Ry/N=0, OP=10, F=010, P/I/L/A=00, WrD=1, @D=IR<11..9>.
15	In	Rd ← Input[N8]	ADDR-IO=IR<7..0>, Rd-In=1, P/I/L/A=10, WrD=1, @D=IR<11..9>
16	Out	Output[N8] ← RX	ADDR-IO=IR<7..0>, Wr-Out=1.
17	Nop		
...	...		
31	Nop		

Paraula de control node F (*Fetch*)

- **Fase comuna per a totes les instruccions**
- Llegeix instrucció SISA de memòria i la carrega al registre IR
- La ALU actualitza registre PC amb PC+2
  - Avancem feina perquè la ALU no faria res aquesta fase
  - Si la instrucció és un `BZ/BNZ` que salta, tornarem a actualitzar el PC

Estado		Acciones	Palabra de control compactada
0	F	Búsqueda de la Instr.: IR ← Mem_w[PC] // Incremento del 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.

- Com es generarà la constant 0x0002 per poder fer la suma?

Paraula de control node D (*Decode*)

- **Fase comuna per a totes les instruccions**
- De forma especulativa...
  - La ALU calcula quin serà el valor del PC en cas que aquesta instrucció resulti ser `BZ` o `BNZ` i es compleixi la condició de salt
    - Es guarda al registre R@
  - Llegeix del REGFILE els valors dels que haurien de ser els registres font Ra i Rb i es carreguen als registres RX i RY

Estado		Acciones	Palabra de control compactada
1	D	Decodificación.	La decodificación no requiere ninguna acción en la UPG por lo que se usa la UPG en este ciclo para adelantar trabajo que pueda ser útil.
		Calculo @ salto tomado: R@ ← PC + SE(N8)*2 //	Este cálculo solo será util si la instrucción es BZ o BNZ. N=SE(IR<7..0>)*2, Pc/Rx=1, Ry/N=0, OP=00, F=100.
		Lectura de registros. (RX ← Ra) // (RY ← Rb)	Estas acciones se realizan sin tener que especificar nada en la palabra de control ya que RX y RY no tienen señal de permiso de escritura y @A y @B se generan directamente de los campos de bits del registro de instrucción IR<11..9> e IR<8..6>, respectivamente, en cada ciclo del grafo. Por ello las especificamos aquí entre paréntesis y ya no las especificaremos de ninguna forma en el resto de nodos.

Paraula de control node AI (i Cmp)

- Fase de càlcul comuna per a totes les instruccions aritmètico-lògiques
- La ALU fa el càlcul corresponent a la instrucció
- El resultat s'escriu al REGFILE

Estado		Acciones	Palabra de control compactada
2	AI	Rd ← RX AI RY	Pc/Rx=0, Ry/N=1, OP=00, F=IR<2..0>, P/I/L/A=00, WrD=1, @D=IR<5..3>.

- El node Cmp és anàleg a aquest
  - A la paraula de control només varia el valor del bus OP (valdrà 01)

Paraula de control node Addi

- Fase de càlcul per a la instrucció `ADDI`
- La ALU fa la suma de RX amb SE(N6)
- El resultat s'escriu al REGFILE

Estado		Acciones	Palabra de control compactada
4	Addi	Rd ← RX + SE(N6)	N=SE(IR<5..0>), Pc/Rx=0, Ry/N=0, OP=00, F=100, P/I/L/A=00, WrD=1, @D=IR<8..6>.

Paraula de control node Addr

- Node comú per a les instruccions de memòria `LD`, `LDB`, `ST`, `STB`
- La ALU calcula la direcció de memòria a accedir RX + SE(N6)
  - El resultat es guarda al registre R@

Estado		Acciones	Palabra de control compactada
5	Addr	R@ ← RX + SE(N6)	N=SE(IR<5..0>), Pc/Rx=0, Ry/N=0, OP=00, F=100.

- Com a *Decode*, al final del cicle RX i RY es carregaran amb Ra i Rb

Paraules de control nodes Ld, Ldb, St, Stb

- Un node per a cada instrucció
- Es determinen senyals de control Wr-Mem Byte i WrD
- Es fa l'accés a memòria (a l'adreça emmagatzemada al registre R@)
  - `LD`, `LDB` guarden la dada llegida al REGFILE
  - `ST`, `STB` guarden RY a memòria

Estado		Acciones	Palabra de control compactada
6	Ld	Rd ← Mem_w[R@]	R@/Pc=1, Byte=0, P/I/L/A=01, WrD=1, @D=IR<8..6>.
7	St	Mem_w[R@] ← RY	R@/Pc=1, Byte=0, Wr-Mem=1.
8	Ldb	Rd ← Mem_b[R@]	R@/Pc=1, Byte=1, P/I/L/A=01, WrD=1, @D=IR<8..6>.
9	Stb	Mem_b[R@] ← RY<7..0>	R@/Pc=1, Byte=1, Wr-Mem=1.

Paraula de control node Bz (i Bnz)

- La ALU deixa passar RX i actualitza el bit z
- Si z=1, es trenca el seqüenciament implícit
  - Cal posar LdPc=1
    - R@ (calculat a *Decode*) es carrega al registre PC

Estado		Acciones	Palabra de control compactada
11	Bz	if (RX == 0) PC ← R@	Pc/Rx=0, OP=10, F=000, Alu/R@=0, LdPc=z.

- El node Bnz és anàleg a aquest
  - Només varia el valor del senyal LdPC (serà !z)

Paraules de control nodes Movi, Movhi

- Un node per a cada instrucció
- Varien en el senyals F i Pc/Rx
- Guarda el resultat al REGFILE

Estado		Acciones	Palabra de control compactada
13	Movi	Rd ← SE(N8)	N=SE(IR<7..0>), Ry/N=0, OP=10, F=001, P/I/L/A=00, WrD=1, @D=IR<11..9>.
14	Movhi	Rd ← (N*(2^8)) | RX<7..0>	N=SE(IR<7..0>), Pc/Rx=0, Ry/N=0, OP=10, F=010, P/I/L/A=00, WrD=1, @D=IR<11..9>.

Paraules de control nodes In, Out

- Un node per a cada instrucció
  - `IN` guarda el contingut d'un *port* al REGFILE
  - `OUT` escriu RX sobre un *port*

Estado		Acciones	Palabra de control compactada
15	In	Rd ← Input[N8]	ADDR-IO=IR<7..0>, Rd-In=1, P/I/L/A=10, WrD=1, @D=IR<11..9>
16	Out	Output[N8] ← RX	ADDR-IO=IR<7..0>, Wr-Out=1.

Índex

Exercici: sortides UC

- Donat l'estat actual de la UC i el contingut del registre IR, indiqueu la paraula de control i l'estat següent de la UC (assumiu que abans d'executar f) el registre R4 val 0xFFFF)

Apartado	Nodo/Estado (Mnemo Salida)	Instrucción en el IR (en ensamblador)	Nodo/Estado Siguiente (Mnemo Salida)
a	F	`(no se sabe)`	
b	D	`BNZ R4, -2`	
c	Addr	`STB -3(R1), R7`	
d	AI	`SUB R3, R1, R2`	
e	Movhi	`MOVHI R1, 27`	
f	Bnz	`BNZ R4, -5`	

Apartado	@A	@B	Pc/Rx	Ry/N	OP	F	P/I/L/A	@D	WrD	Wr-Out	Rd-In	Wr-Mem	LdIr	LdPc	Byte	Alu/R@	R@/Pc	N (hexa)	ADDR-IO (hexa)
a																			
b																			
c																			
d																			
e																			
f																			

Exercici: sortides UC

- Donat l'estat actual de la UC i el contingut del registre IR, indiqueu la paraula de control i l'estat següent de la UC (assumiu que abans d'executar f) el registre R4 val 0xFFFF)

Apartado	Nodo/Estado (Mnemo Salida)	Instrucción en el IR (en ensamblador)	Nodo/Estado Siguiente (Mnemo Salida)
a	F	(no se sabe)	D
b	D	BNZ R4, -2	Bnz
c	Addr	STB -3(R1), R7	Stb
d	AI	SUB R3, R1, R2	F
e	Movhi	MOVHI R1, 27	F
f	Bnz	BNZ R4, -5	F

Apartado	@A	@B	Pc/Rx	Ry/N	OP	F	P/I/L/A	@D	WrD	Wr-Out	Rd-In	Wr-Mem	LdIr	LdPc	Byte	Alu/R@	R@/Pc	N (hexa)	ADDR-IO (hexa)
a	xxx	xxx	1	0	00	100	xx	xxx	0	0	0	0	1	1	0	1	0	0002	xx
b	100	xxx	1	0	00	100	xx	xxxx	0	0	0	0	0	0	x	x	x	FFFC	xx
c	xxx	111	0	0	00	100	xx	xxx	0	0	0	0	0	0	x	x	x	FFFD	xx
d	xxx	xxx	0	1	00	101	00	011	1	0	0	0	x	0	x	x	x	xxxx	xx
e	xxx	xxx	0	0	10	010	00	001	1	0	0	0	x	0	x	x	x	001B	xx
f	xxx	xxx	0	x	10	000	xx	xxx	0	0	0	0	x	1	x	0	x	xxxx	xx

- Els senyals en verd són coneguts (surten directament de l'IR) però en aquesta fase/instrucció són irrellevants

Índex

Conclusions

- El Computador von Neumann reutilitza els recursos *hardware*
  - La mateixa memòria conté codi i dades
  - La ALU també actualitza el PC
- Implicacions
  - És multicicle
  - La paraula de control variarà cada cicle
- L'execució de cada instrucció es descompon en *Fetch*, *Decode* i càlcul
  - *Fetch* i *Decode* duren un cicle cadascuna
  - El càlcul pot durar un o més cicles
- Modificacions respecte computador Harvard
  - La UC serà un CLS amb 18 estats
  - La UP té nous registres/camins respecte la UPG del Harvard
- Xuletari: `https://atenea.upc.edu/mod/resource/view.php?id=1673049`
- A la propera classe implementarem la UC
- Contesteu el qüestionari d'Atenea ET13a abans de la propera classe!

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] [Online]. Available: https://www.theguardian.com/technology/2012/feb/26/first-computers-john-von-neumann.

[2] *IAS machine*, [Online]. Available: https://en.wikipedia.org/wiki/IAS_machine.

Introducció als Computadors

Tema 13: Computador SISC von Neumann

http://personals.ac.upc.edu/enricm/Docencia/IC/IC13a.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.