Introducció als Computadors

Tema 12: Computadors Harvard unicicle i multicicle

http://personals.ac.upc.edu/enricm/Docencia/IC/IC12.pdf

Enric Morancho
(enricm@ac.upc.edu)

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

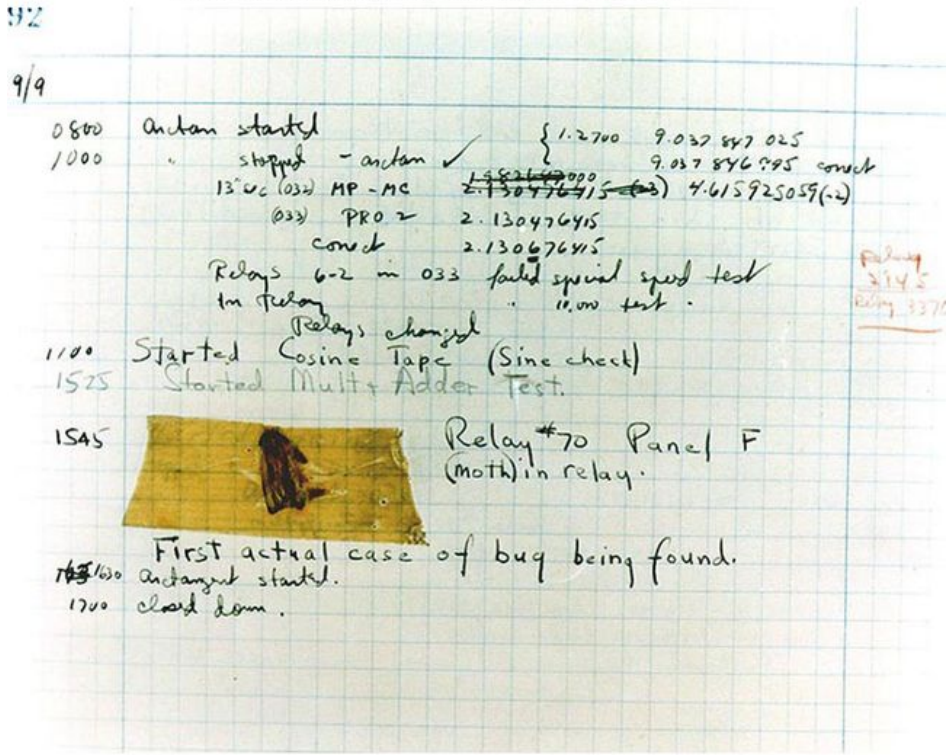
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

2020-21, 1^{er} quad.

The devil is in the details

- Hem de determinar el T_c del nostre computador model Harvard
  - El Harvard Mark I (1944) feia 3 sumes o restes per segon [1]
    - $T_c = 0,\widehat{3}s$
    - $Freq = 1/T_c = 3Hz$
- Compte amb els *bugs*!
  - Sep 9, 1947: World's First Computer Bug at Harvard Mark II

Photo # NH 96566-KN (Color) First Computer "Bug", 1947



[2]

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*	-	-	-	✓	✓	✓	✓	✓

Objectius del tema

- Completar el disseny del computador Harvard unicicle
  - Determinar temps de cicle
  - Garantir correcta temporalitat a les comunicacions asíncrones
    - Amb memòria RAM i el sistema d'entrada/sortida
- Fer una versió multicicle del computador Harvard
  - Constatar que hi ha instruccions LM SISA "lentes" i "ràpides"
  - Redefinir el temps de cicle per no malbaratar el temps a l'executar instruccions "ràpides"
- Avaluació del rendiment

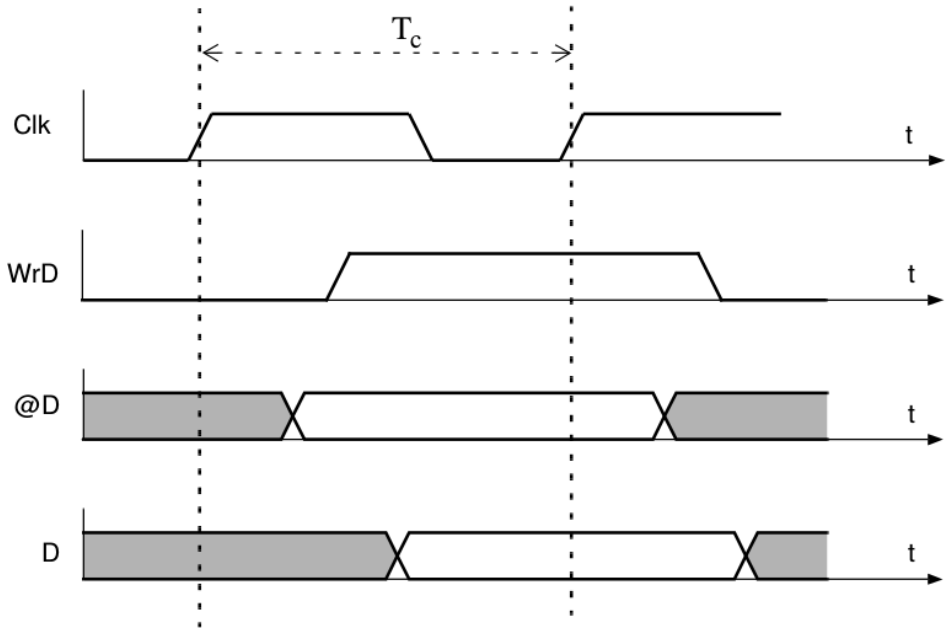
Índex

Completant el disseny del processador

- Senyals de la paraula de control
  - Des de que es produeix un flanc ascendent de rellotge, els senyals triguen un cert temps en estabilitzar-se
    - Tot i que abans d'estabilitzar-se es poden produir *glitches*
    - Cada senyal pot trigar un temps diferent en estabilitzar-se
  - Es mantindran estables, com a mínim, fins al següent flanc de rellotge
- El T_c ha de ser més gran que el temps de propagació del camí crític
  - Des de la sortida Q d'un biestable fins a l'entrada D d'un biestable
    - El mateix o un altre biestable
- Estudiarem les restriccions addicionals que puguin imposar els senyals que comporten la modificació de l'estat del computador
  - WrD, Wr-Mem, Rd-In, Wr-Out

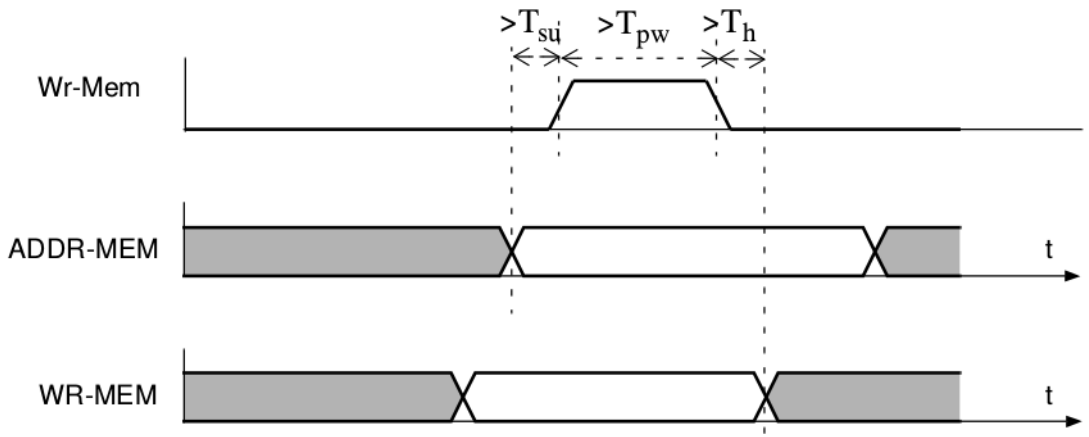
Senyal WrD

- Determina si al final d'aquest cicle es farà una escriptura al REGFILE
  - Escriptura síncrona (per flanc ascendent de rellotge)
- Cal garantir que els senyals WrD, @D i D siguin estables quan es produeix el flanc ascendent
  - L'ordre en el que s'estabilitzen és irrellevant



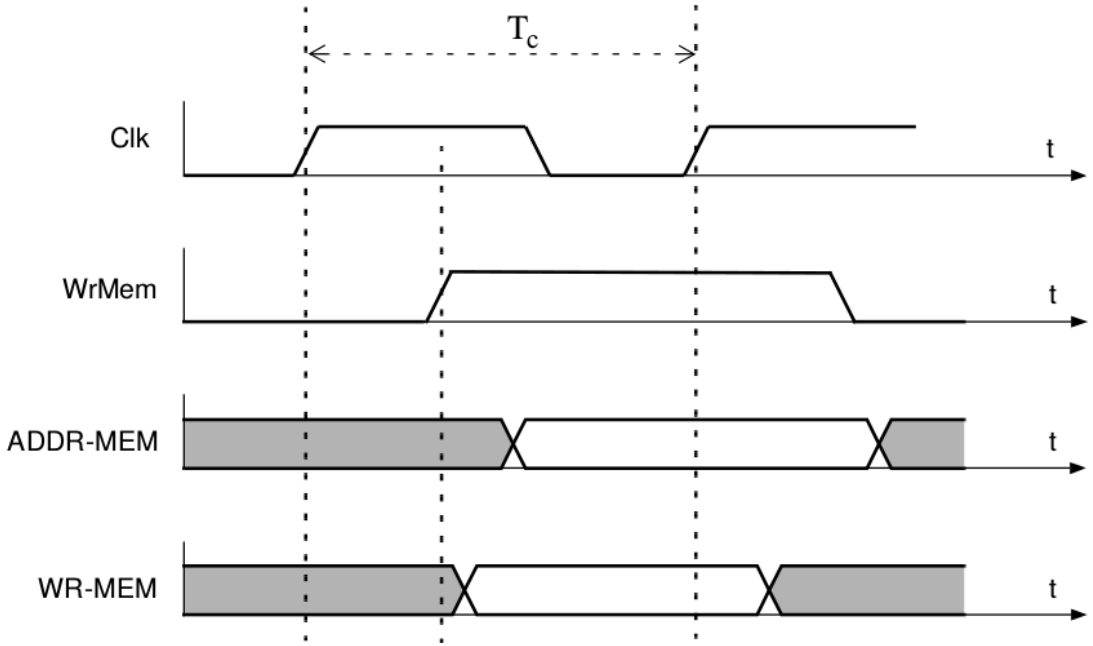
Senyal Wr-Mem

- Determina si aquest cicle es farà una escriptura a la RAM
  - Escriptura asíncrona (per nivell del senyal Wr-Mem)
- **Cal garantir que quan Wr-Mem passa a "1", ADDR-MEM i WR-MEM ja siguin estables**
- A més, el mòdul de memòria imposa que els senyals siguin estables un cert temps abans que Wr-Mem passi a "1", mentre val "1" i després que passi a "0"
  - T_{su} (*setup*), T_{pw} (*pulse width*), T_h (*hold*)
- Cronograma d'una escriptura **correcta** a memòria:



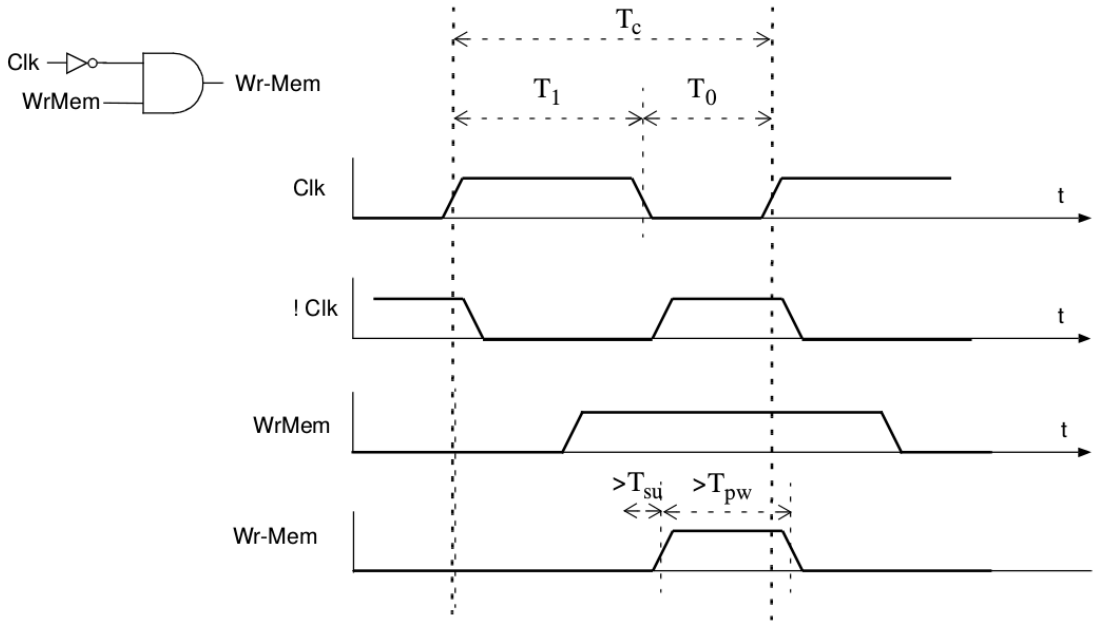
Senyal Wr-Mem: situació actual

- La implementació actual **no garanteix** escriptures correctes a memòria
- Si fem l'estudi del temps de propagació veuríem:
  - WrMem/Wr-Mem triga 960 u.t. en estabilitzar-se
    - A més, continua estable un cert temps després del flanc de rellotge!
  - ADDR-MEM triga 1.950 u.t. en estabilitzar-se
  - WR-MEM triga 1.030 u.t. en estabilitzar-se
- Cronograma d'escriptura **incorrecta** a memòria:

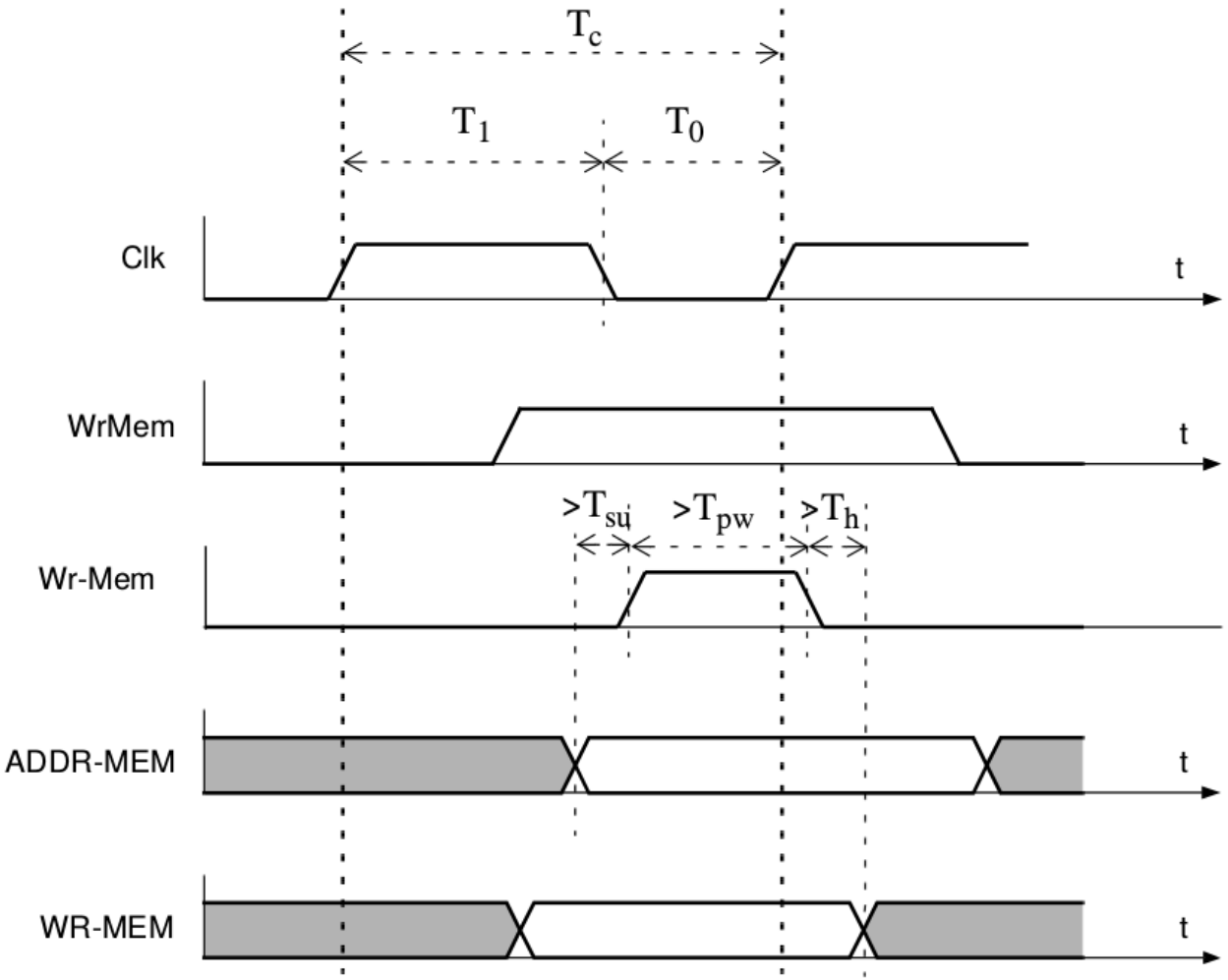


Senyal Wr-Mem: solució

- Diferenciarem WrMem i Wr-Mem
  - WrMem: sortida de la ROM_CTRL_LOGIC
  - Wr-Mem: WrMem amb restriccions temporals
- Per implementar Wr-Mem modificarem la forma del senyal de rellotge
  - Ja no serà simètrica
    - Fins ara, el flanc descendent de rellotge ha estat irrellevant
  - Determinarem T_0 de forma que ja sigui segur posar Wr-Mem a "1"



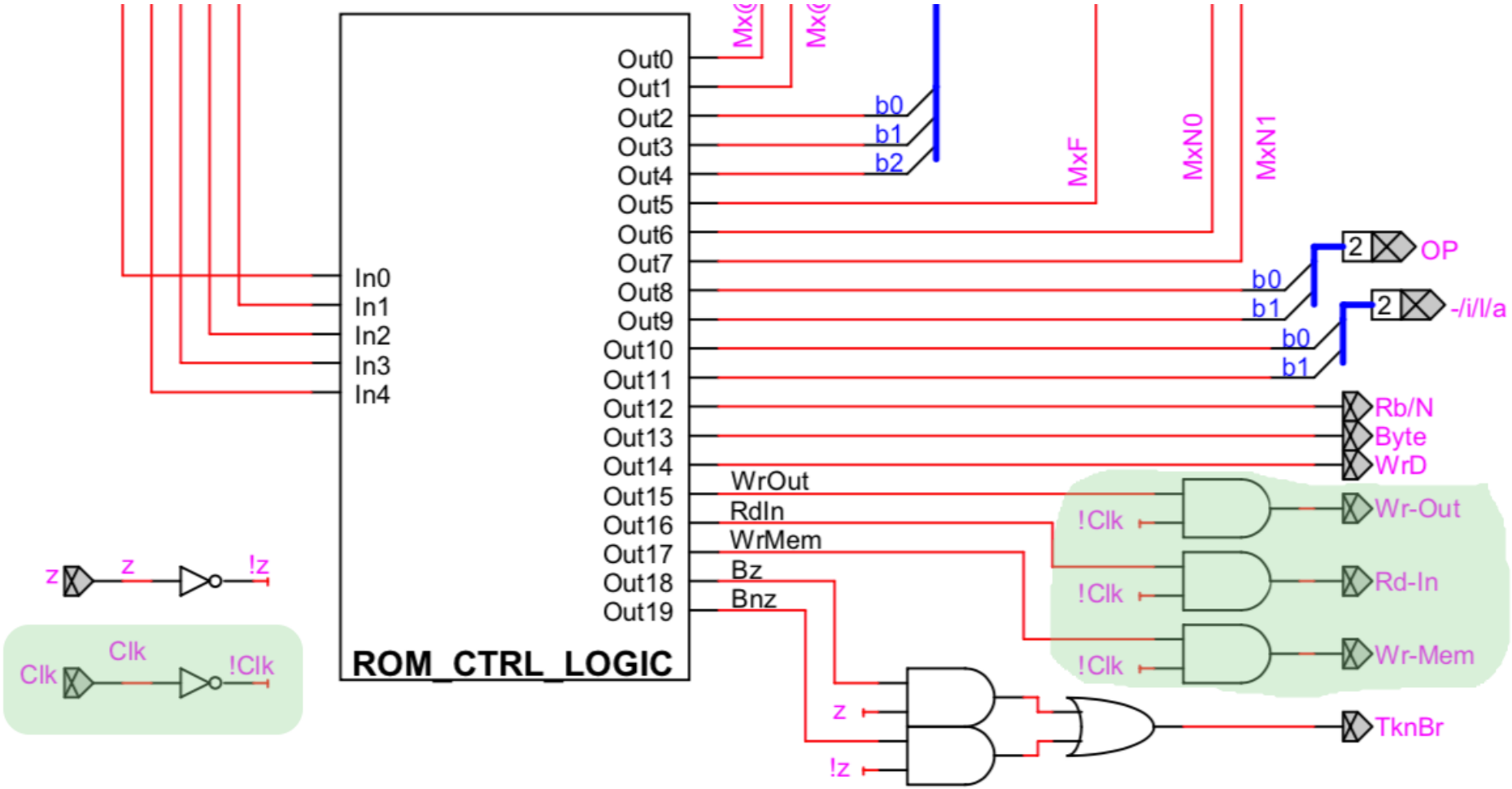
Senyal Wr-Mem: cronograma solució



Senyals Rd-In i Wr-Out

- Presenten el mateix problema que Wr-Mem
  - Wr-Out valida els valors dels senyals ADDR-IO i WR-OUT
  - Rd-In valida els valors dels senyals ADDR-IO
    - La lectura d'un *port* de dades té efecte lateral
- Assumim que les restriccions temporals dels ports d'E/S són menys restrictives que les de l'escriptura a memòria
- Generació del senyals com Wr-Mem:
  - Wr-Out = !Clk · WrOut
  - Rd-In = !Clk · RdIn

Lògica de control (fragment)

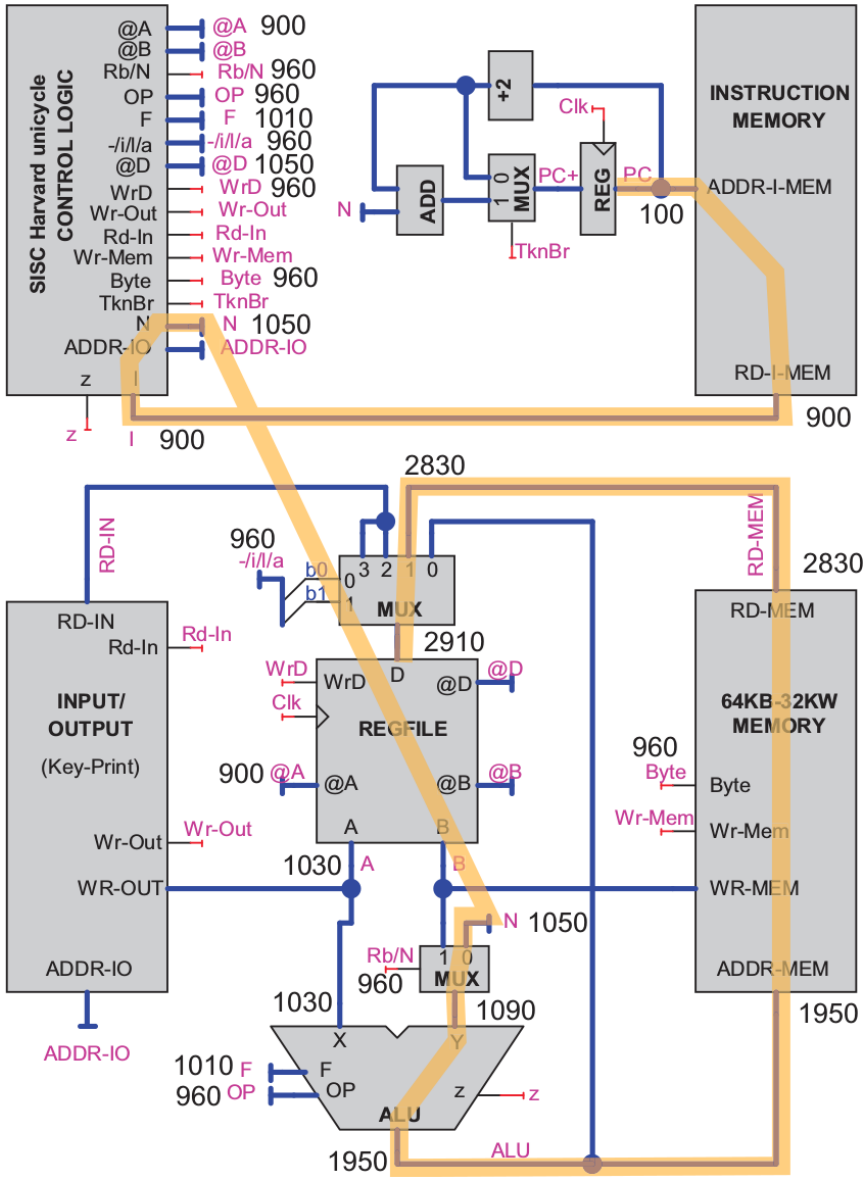


Índex

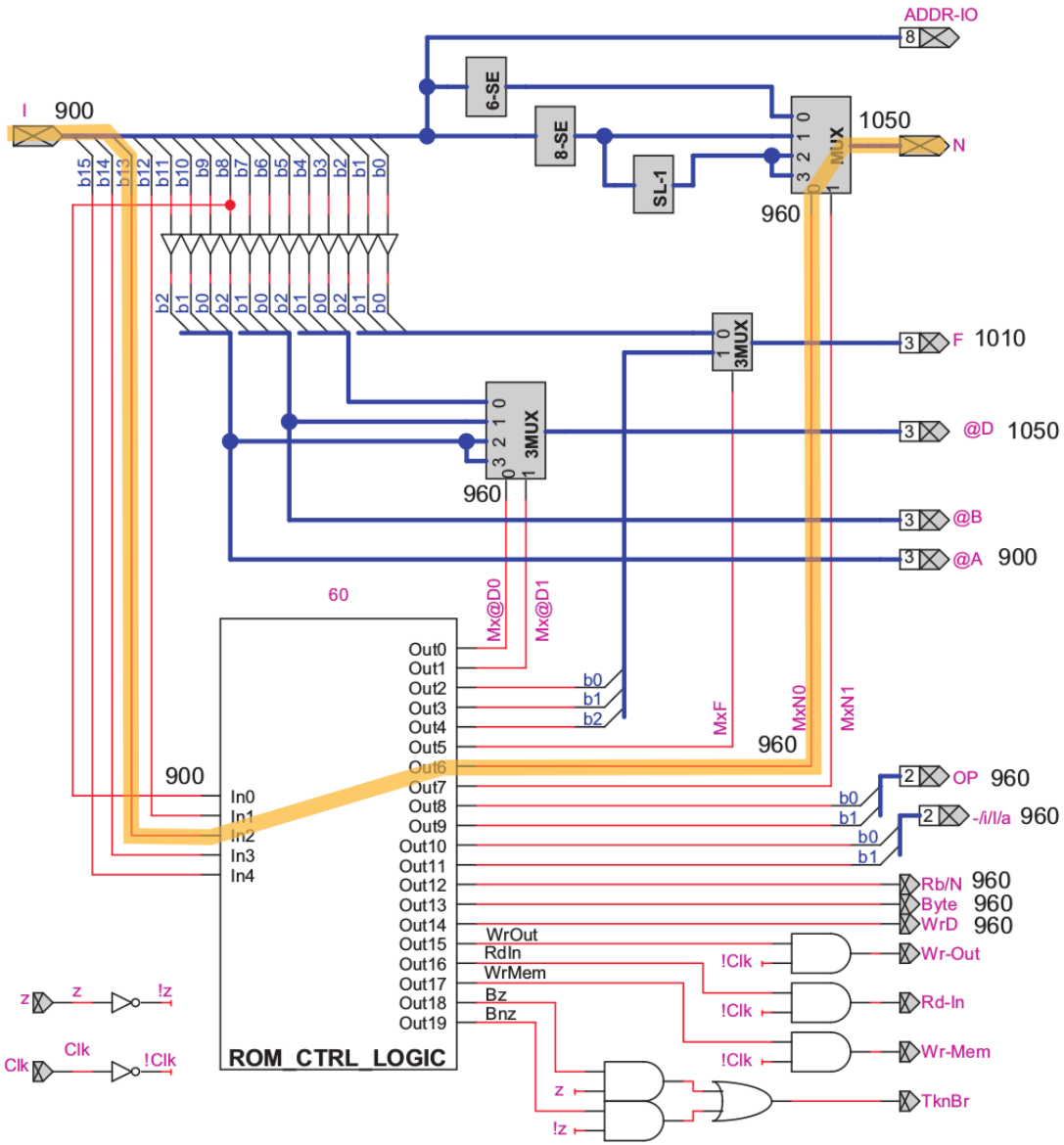
T_c del computador Harvard unicicle

- Identificarem el camí crític del computador Harvard i el seu T_p
- Necessitem saber el T_p dels components bàsics:
  - $T_p(\text{NOT}) = 10$ u.t.
  - $T_p(\text{And-2}) = T_p(\text{Or-2}) = 20$ u.t.
  - $T_p(\text{FF}) = 100$ u.t.
  - $T_p(\text{ROM\_CTRL\_LOGIC}) = 60$ u.t.
  - $T_{acc}(\text{I-MEM}) = 800$ u.t.
  - $T_{acc}(\text{32KB-RAM}) = 800$ u.t.
    - Temps d'accés (lectura) a un mòdul de memòria RAM
- De totes les instruccions SISA, el camí crític ve donat per `LDB`
  - El camí comença al registre `PC`
  - Finalitza al REGFILE (registre destí del `LDB`)

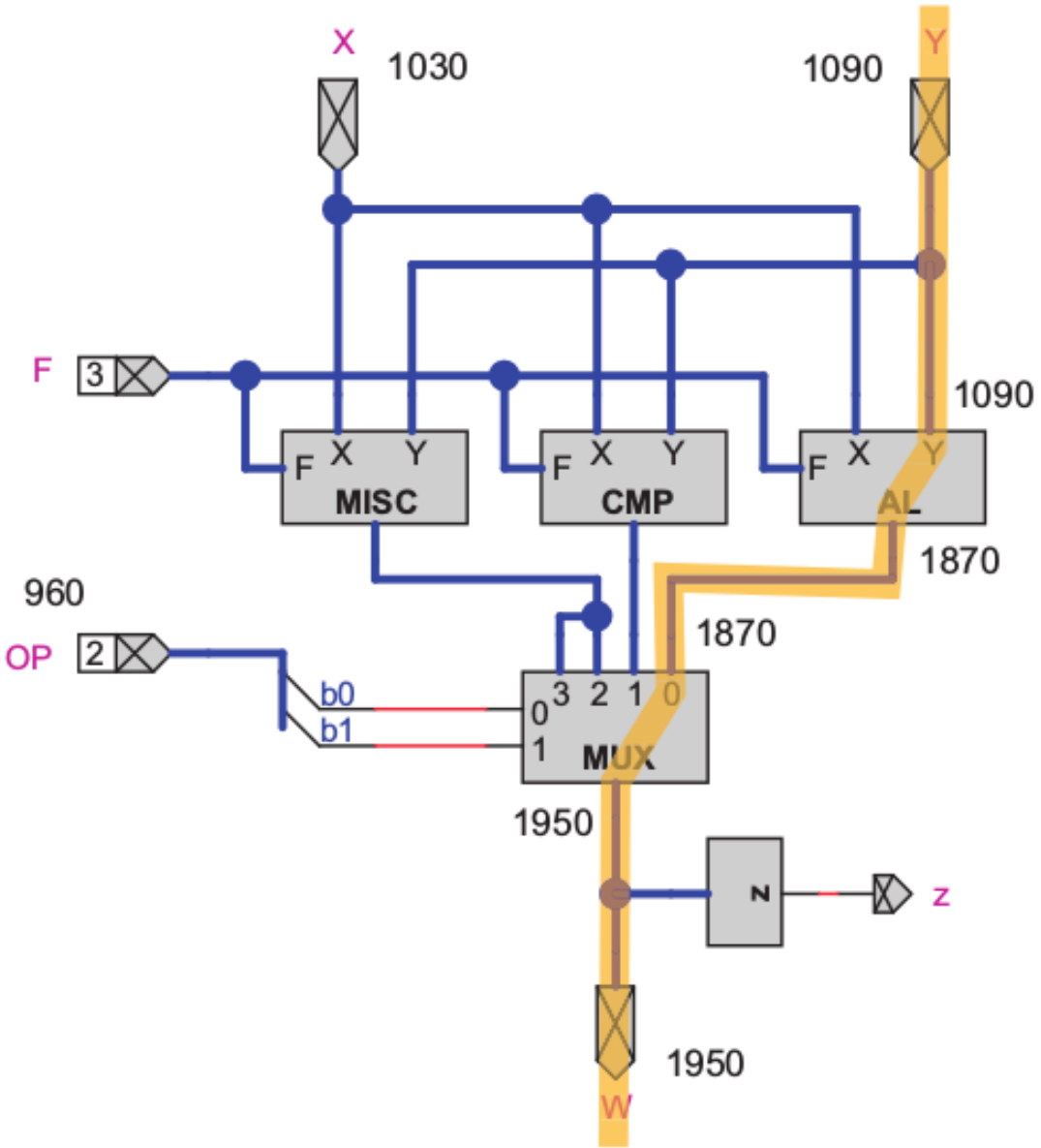
Execució LDB: global



Execució LDB: lògica de control

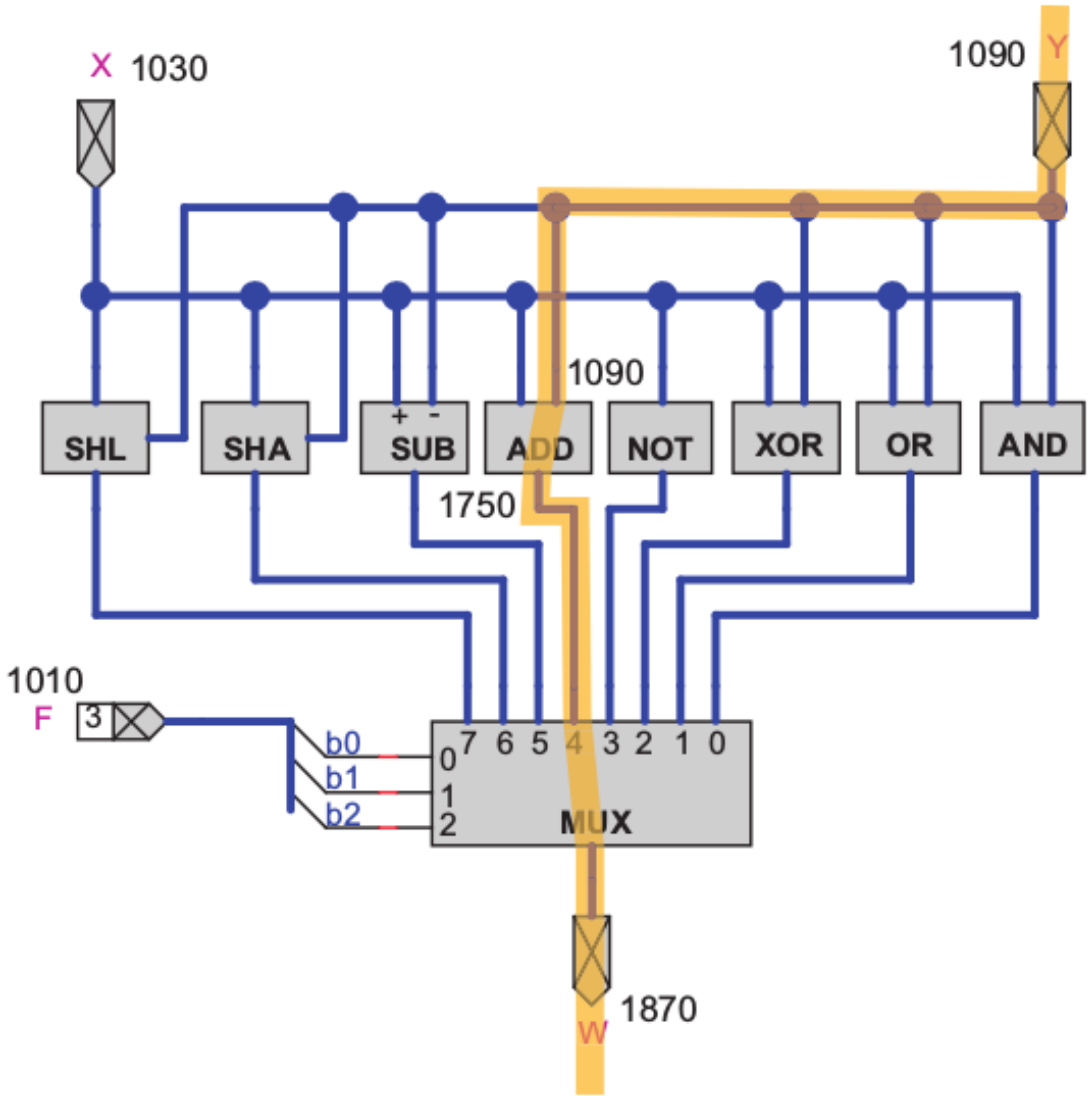


Execució LDB: ALU



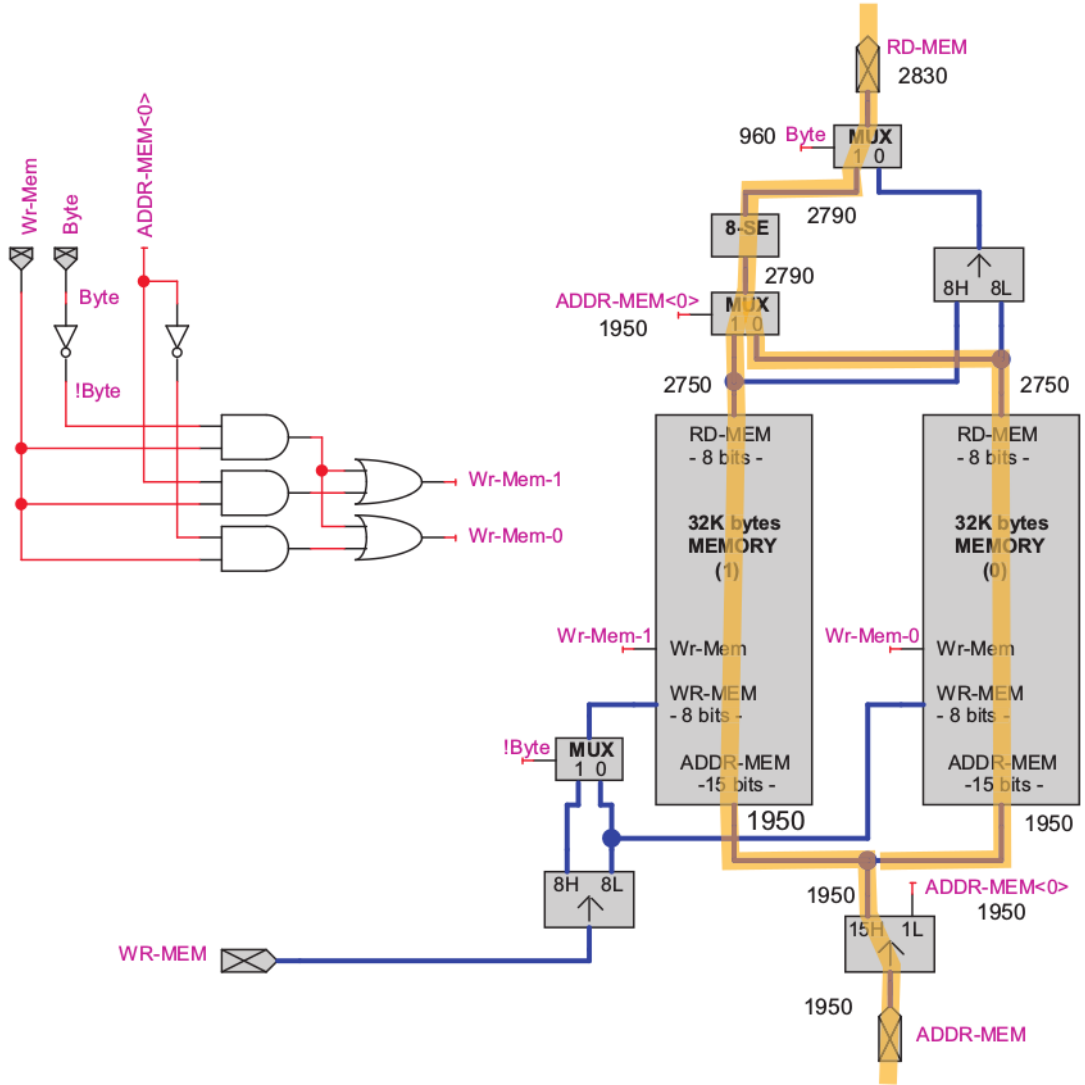
Execució LDB: AL

- $T_p(ADD) = 660u.t.$

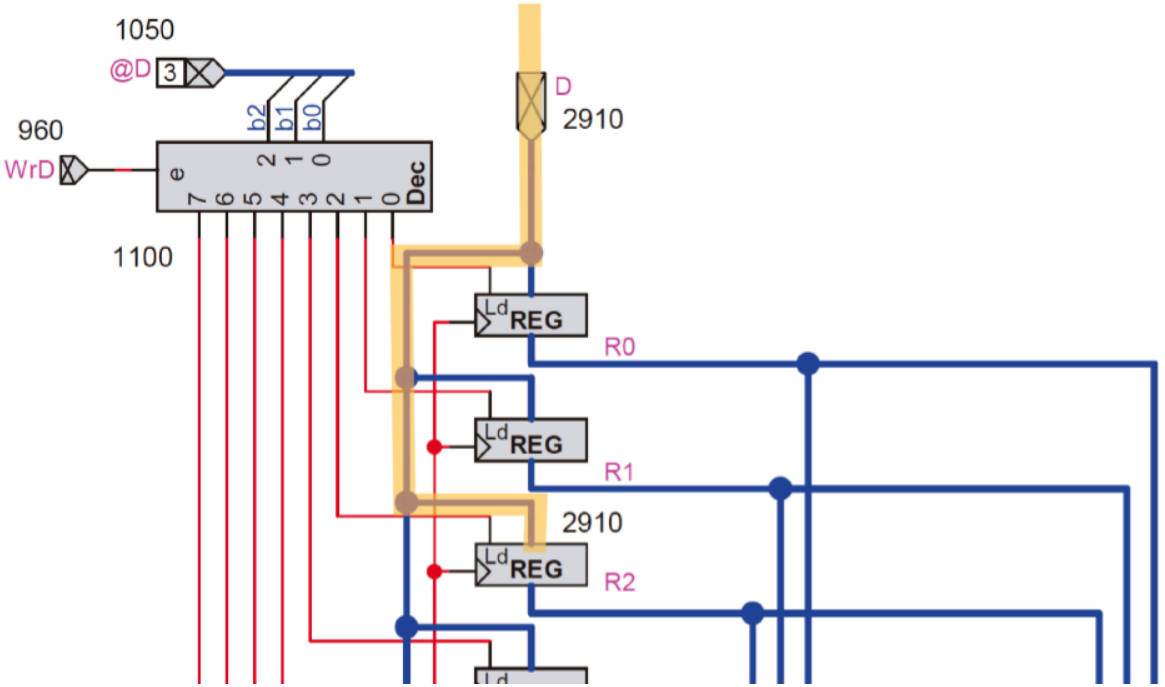


Execució LDB: RAM

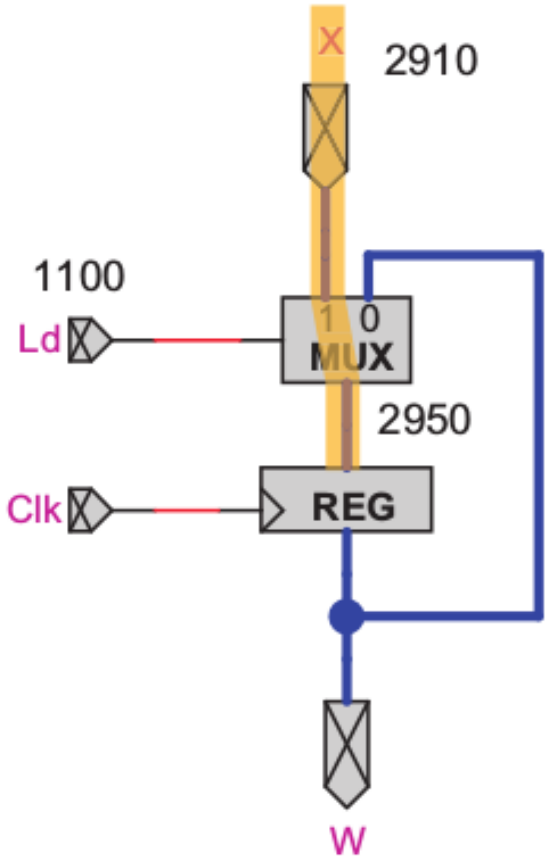
- $T_{acc}(32KB\ RAM) = 800u.t.$



Execució LDB: REGFILE



Execució LDB: càrrega registre



Temps de cicle Harvard unicicle

- Com a mínim, T_c ha de ser 2.950 u.t.
  - L'arrodonim a 3.000 u.t.
- El senyal de rellotge pot ser simètric?
  - $T_1 = T_0 = 3.000/2 = 1.500$ u.t. ?
  - **No**, perquè ADDR-MEM triga 1.950 en estabilitzar-se
  - El senyal Wr-Mem passaria a "1" massa d'hora
- Alternatives:
  - $T_1 = T_0 = 2.000$ u.t. $\Longrightarrow$ $T_c = 4.000$ u.t.
  - $T_1 = \mathbf{2000\ u.t.}$, $T_0 = \mathbf{1.000\ u.t.}$ $\Longrightarrow$ $T_c = \mathbf{3.000\ u.t.}$

$T_c = 2.000 + 1.000$ i la memòria RAM

- Especificació memòria RAM:
  - $T_{acc} = 800$ u.t., $T_{su} = 60$ u.t., $T_{pw} = 600$ u.t, $T_h = 40$ u.t.
- Temporització senyals:
  - Wr-Mem0 i Wr-Mem1 passen a "1" a 2.070 u.t.
  - ADDR-MEM és estable des de 1.950 u.t.
  - WR-MEM és estable des de 1.030 u.t.
- $T_c = 2.000 + 1.000$ compleix l'especificació
  - T_{su} podria fer fins a 2.070-1.950 = 120 u.t.
  - T_{pw} podria ser fins a 1.000 u.t.
  - Per analitzar T_h caldria determinar quant de temps ADDR-MEM i WR-MEM romanen estables després del pols ascendent del rellotge
    - Wr-Mem0 i Wr-Mem1 trigaran 70 u.t. en passar a "0"
    - Si ADDR-MEM i WR-MEM romanen estables k u.t., T_h podria arribar fins a $k - 70$ u.t.
    - Assumim $T_h = 40$ u.t. està dins del rang

Índex

Temps d'execució d'un programa

- A una màquina unicicle
  - $T_{execució} = N_{instruccions} \cdot T_c$
- Calcularem el temps d'execució per a quatre versions d'un programa
- Especificació del programa:
  - Copiar un vector de 1.000 *bytes* emmagatzemat a la RAM
  - Origen: a partir de l'adreça 0x5000, a posicions consecutives
  - Destí: a partir de l'adreça 0x8000, a posicions consecutives

v1: 1 *byte* per iteració


```
            MOVI    R1, 0x00
            MOVHI   R1, 0x50        ; R1 <- punter a A (0x5000)
            MOVI    R2, 0x00
            MOVHI   R2, 0x80        ; R2 <- punter a B (0x8000)
            MOVI    R3, 0xE8
            MOVHI   R3, 0x03        ; R3 <- 1000 (0x03E8)
            ADD     R3, R3, R1      ; R3 <- Darrera adreça A
Bucle1b:    LDB     R4, 0(R1)       ; R4 <- Llegim un byte de A
            STB     0(R2), R4       ; Escrivim byte a B
            ADDI    R1, R1, 1       ; Incrementem punter A
            ADDI    R2, R2, 1       ; Incrementem punter B
            CMPLTU  R5, R1, R3      ; Hem acabat?
            BNZ     R5, -6          ; Si no, tornem a iterar
```

- A cada iteració del bucle es copia un *byte*
- Nombre instruccions executades
  - 7 instruccions abans de bucle (pròleg)
  - 1000 iteracions de 6 instruccions/iteració = 6.000 instruccions

v2: 4 *bytes* per iteració (*loop unrolling*)


```
...                                 ; 7 instruccions proleg
Bucle4b: LDB     R4, 0(R1)          ;
         STB     0(R2), R4          ; Primer element bucle
         LDB     R4, 1(R1)          ;
         STB     1(R2), R4          ; Segon element bucle
         LDB     R4, 2(R1)          ;
         STB     2(R2), R4          ; Tercer element bucle
         LDB     R4, 3(R1)          ;
         STB     3(R2), R4          ; Quart element bucle
         ADDI    R1, R1, 4          ; Incrementem punter A
         ADDI    R2, R2, 4          ; Incrementem punter B
         CMPLTU  R5, R1, R3         ; Hem acabat?
         BNZ     R5, -12            ; Si no, tornem a iterar
```

- A cada iteració del bucle es copien quatre *bytes*
- Nombre instruccions executades
  - 7 instruccions abans de bucle (pròleg)
  - 250 iteracions de 12 instruccions/iteració = 3.000 instruccions

v3: 2 *words* per iteració


```
...                                     ; 7 instruccions proleg
Bucle2w: LD      R4, 0(R1)              ;
         ST      0(R2), R4              ; Primer element bucle
         LD      R4, 2(R1)              ;
         ST      2(R2), R4              ; Segon element bucle
         ADDI    R1, R1, 4              ; Incrementem punter A
         ADDI    R2, R2, 4              ; Incrementem punter B
         CMPLTU  R5, R1, R3             ; Hem acabat?
         BNZ     R5, -8                 ; Si no, tornem a iterar
```

- A cada iteració del bucle es copien quatre *bytes*
  - **agrupats en dos** *words*
  - Aprofitem *memory bandwidth* del computador (1 *word*/cicle)
- Nombre instruccions executades
  - 7 instruccions abans de bucle (pròleg)
  - 250 iteracions de 8 instruccions/iteració = 2.000 instruccions

v4: 4 *words* per iteració


```
...                                     ; 7 instruccions proleg
Bucle4w: LD      R4, 0(R1)              ;
         ST      0(R2), R4              ; Primer element bucle
         LD      R4, 2(R1)              ;
         ST      2(R2), R4              ; Segon element bucle
         LD      R4, 4(R1)              ;
         ST      4(R2), R4              ; Tercer element bucle
         LD      R4, 6(R1)              ;
         ST      6(R2), R4              ; Quart element bucle
         ADDI    R1, R1, 8              ; Incrementem punter A
         ADDI    R2, R2, 8              ; Incrementem punter B
         CMPLTU  R5, R1, R3             ; Hem acabat?
         BNZ     R5, -12                ; Si no, tornem a iterar
```

- A cada iteració del bucle es copien vuit *bytes*
  - **agrupats en quatre** *words*
- Nombre instruccions executades
  - 7 instruccions abans de bucle (pròleg)
  - 125 iteracions de 12 instruccions/iteració = 1.500 instruccions

Comparativa entre les quatre versions

	$N_{instruccions}$	$T_{execució}$ $N_{instruccions} \times T_c$
v1	$7 + 6.000 \approx 6.000$	18×10^6 u.t.
v2	$7 + 3.000 \approx 3.000$	9×10^6 u.t.
v3	$7 + 2.000 \approx 2.000$	6×10^6 u.t.
v4	$7 + 1.500 \approx 1.500$	$4,5 \times 10^6$ u.t.

- Algunes comparacions:
  - v1 triga el doble de temps que v2
    - v1 triga un 100% més que v2
  - v2 triga la meitat de temps que v1
    - v2 triga un 50% menys que v1
  - v3 és el triple de ràpid que v1
    - v3 és un 200% més ràpid de v1
    - v3 triga un 66,7% menys que v1

Índex

Instruccions lentes vs. ràpides

- De l'estudi del camí crític constatem diferències significatives al T_p de les instruccions de LM SISA
  - Instruccions "lentes":
    - Instruccions d'accés a memòria: `LD`, `LDB`, `ST`, `STB`
    - Era d'esperar perquè totes elles, després d'utilitzar l'ALU per sumar Ra i N6, accedeixen a la RAM
    - El T_p de totes elles és proper a 3.000 u.t.
  - Instruccions "ràpides":
    - Totes les altres
    - La més lenta de les ràpides és `CMPLE` amb T_p=2.210 u.t.
    - Si T_c= 3.000 u.t., a l'executar cada instrucció ràpida al computador Harvard unicicle es malbarata un mínim de 790 u.t.
- Intentarem no malbaratar temps del processador quan executa instruccions "lentes"
  - Totes les instruccions no tindran el mateix temps d'execució

Alternatives al disseny unicicle

- Rellotge amb "taquicàrdia"
  - T_c variable en funció del tipus d'instrucció
- **Processador multicicle**
  - Introduir un T_c de durada inferior al temps necessari per executar qualsevol instrucció ("minicicle")
  - L'execució de cada instrucció requerirà varis "minicicles"
    - Les instruccions "lentes" trigaran més "minicicles" que les "ràpides"

Índex

Processador Harvard multicicle

- Introdueix els "minicicles", de durada inferior al temps necessari per a executar qualsevol instrucció
  - Les instruccions necessitaran varis "minicicles" per a executar-se
  - Al nostre cas, els "minicicles" seran de 750 u.t.
    - Les instruccions ràpides trigaran 3 "minicicles" ($3 \times 750 = 2.250$ u.t.)
    - Les instruccions lentes trigaran 4 "minicicles" ($4 \times 750 = 3.000$ u.t.)
- La paraula de control es mantindrà invariable al llarg de tots els "minicicles" d'execució de cada instrucció
  - Llevat els senyals de modificació d'estat: WrD, Wr-Mem, Rd-In Wr-Out
  - Els registres no s'actualitzaran al final de cada "minicicle" sinó al final del darrer "minicicle" d'execució de la instrucció
    - REGFILE, PC
  - Haurem de tornar a estudiar com generar els senyals per autoritzar la modificació de memòria RAM i *ports* d'E/S
- Per implementar el processador Harvard multicicle només farem modificacions mínimes a la Lògica de Control i a la UPG

Actualització de registres PC i REGFILE

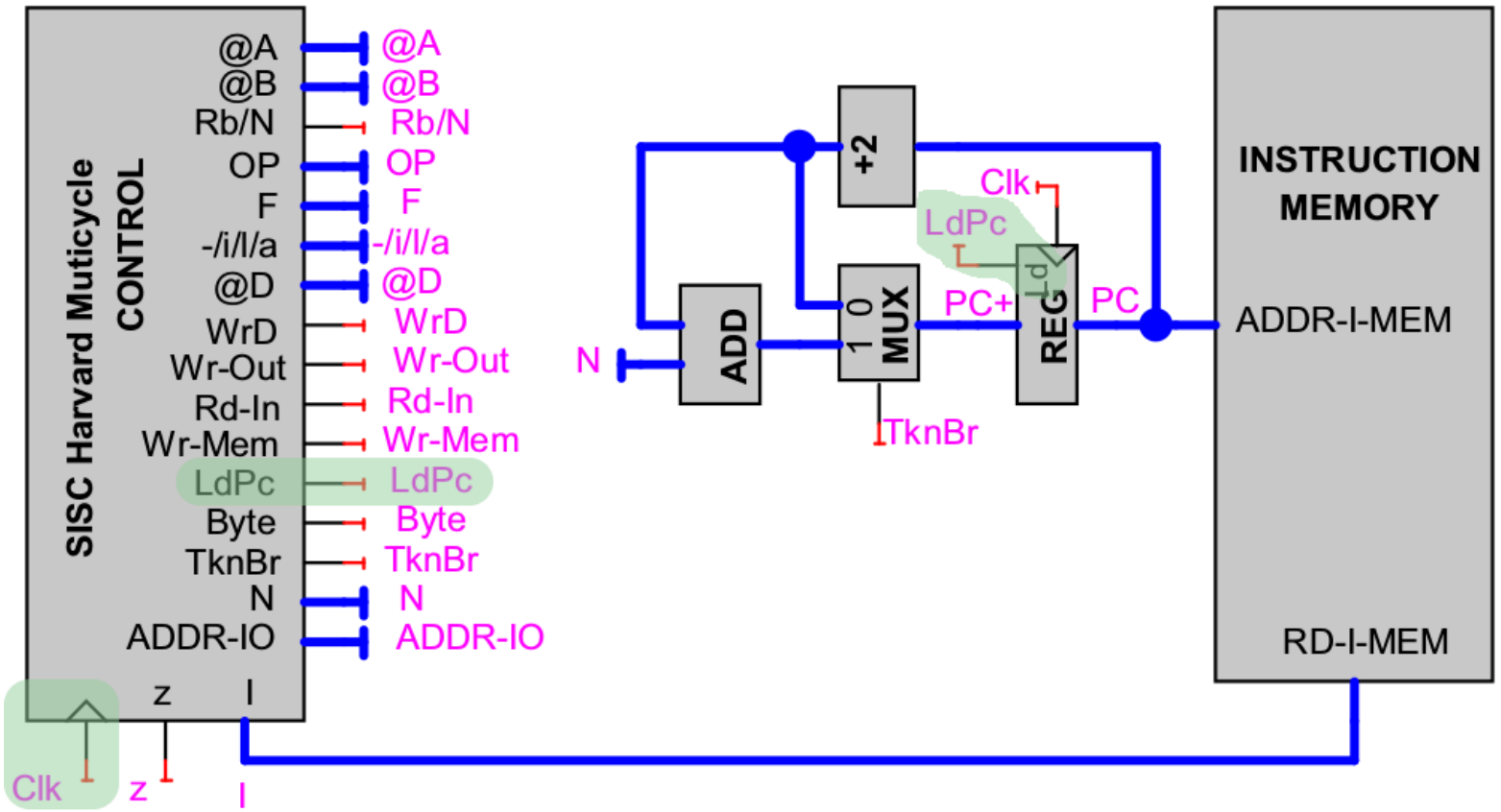
- Harvard unicicle:
  - Actualitza PC al final de cada cicle
  - Actualitza REGFILE al final dels cicles on WrD="1"
- Harvard multicicle:
  - **No pot actualitzar-los al final de cada "minicicle"**
    - Només al final del darrer "minicicle" d'execució de cada instrucció
  - Cal afegir senyal de càrrega al registre PC
    - LdPC: Nou senyal de sortida a la lògica de control
  - Els senyals WrD i LdPC només podran valer "1" el darrer minicicle d'execució de cada instrucció
    - La lògica de control haurà de de ser conscient de s'està executant una instrucció lenta o ràpida i de en quin minicicle d'execució es troba

Actualització de memòria i *ports*

- Harvard unicicle:
  - Utilitza un senyal de rellotge asimètric
  - La segona fase indica quan és segur modificar memòria i *ports*
  - Els senyals d'autorització de modificació de memòria i *ports* només poden valer "1" a aquesta segona fase del cicle
- Harvard multicicle:
  - Donades les latències del mòdul de memòria i dels dispositius d'E/S, els senyals Wr-Mem, Rd-In i Wr-Out només cal que estiguin a "1" un temps inferior a la durada d'un minicicle
  - Farem que aquests senyals només pugin valdre "1" el darrer minicicle d'execució de `ST`, `STB`, `IN`, `OUT`
  - Mateixa solució que a l'actualització de registres

Canvis a la UPG

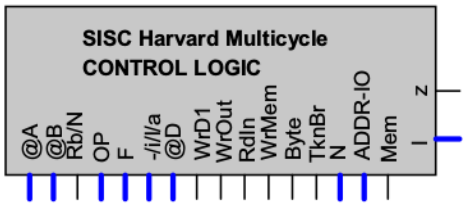
- Registre PC amb senyal de càrrega LdPC

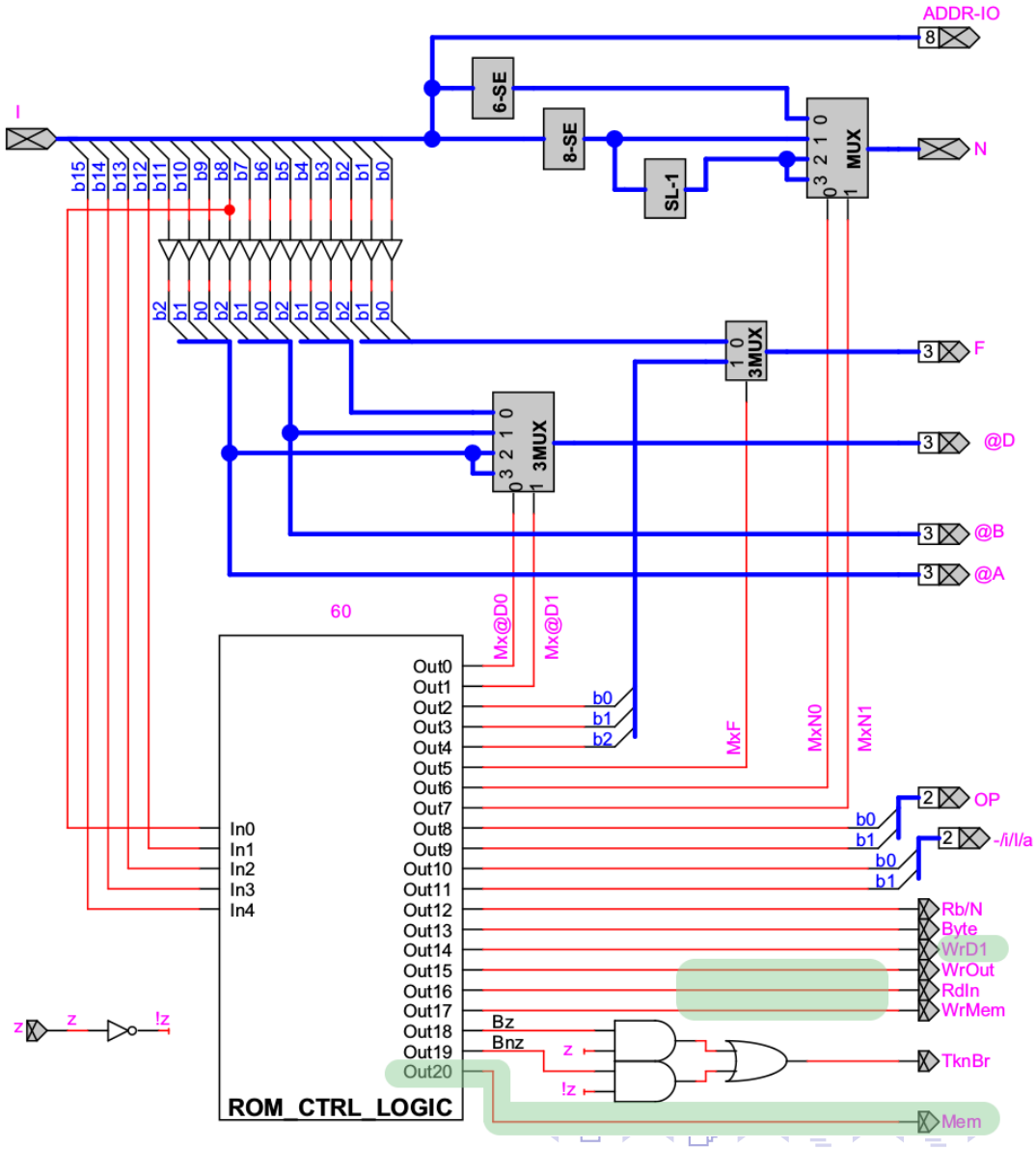


Canvis a la Lògica de control

- Tindrà una part seqüencial i una altra combinacional
- Combinacional: SISC Harvard Multicycle CONTROL LOGIC
  - La ROM generarà un nou senyal anomenat Mem
    - Valdrà "1" si la instrucció és lenta (`ST`, `STB`, `LD`, `LDB`); altrament "0"
  - El senyal WrD passa a dir-se WrD1
  - No farà la And-2 amb el senyal de rellotge negat
- Seqüencial: SISC Harvard Multicycle CONTROL
  - Calcularà la sortida LdPC que indica quan actualitzar el registre PC
    - Valdrà "1" el darrer cicle d'execució de cada instrucció
    - LdPC també s'utilitzarà per restringir quan poden valer "1" els senals Wr-Mem, Rd-In i Wr-Out

SISC Harvard Multicycle CONTROL LOGIC



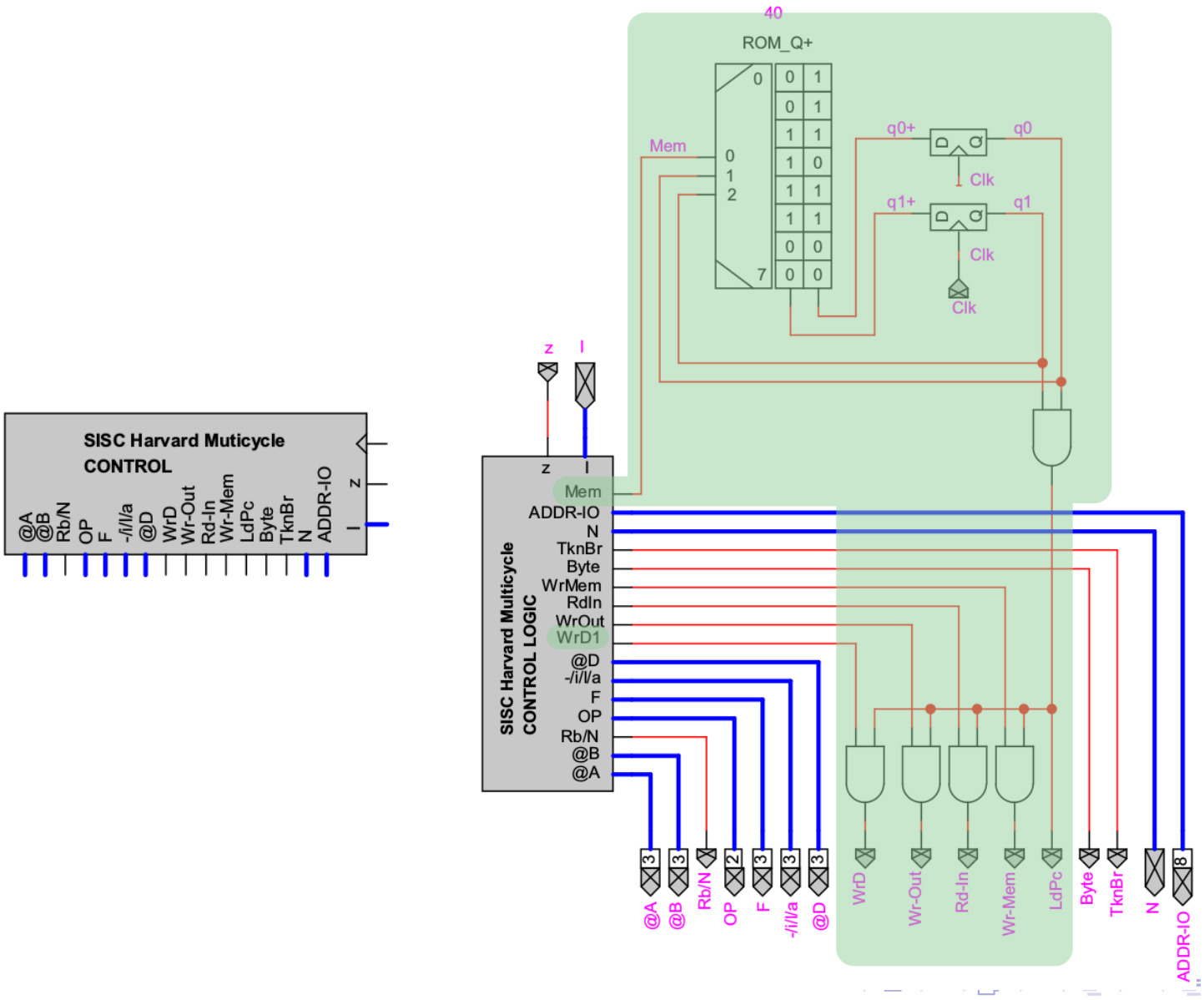


Contingut ROM-CTRL-LOGIC

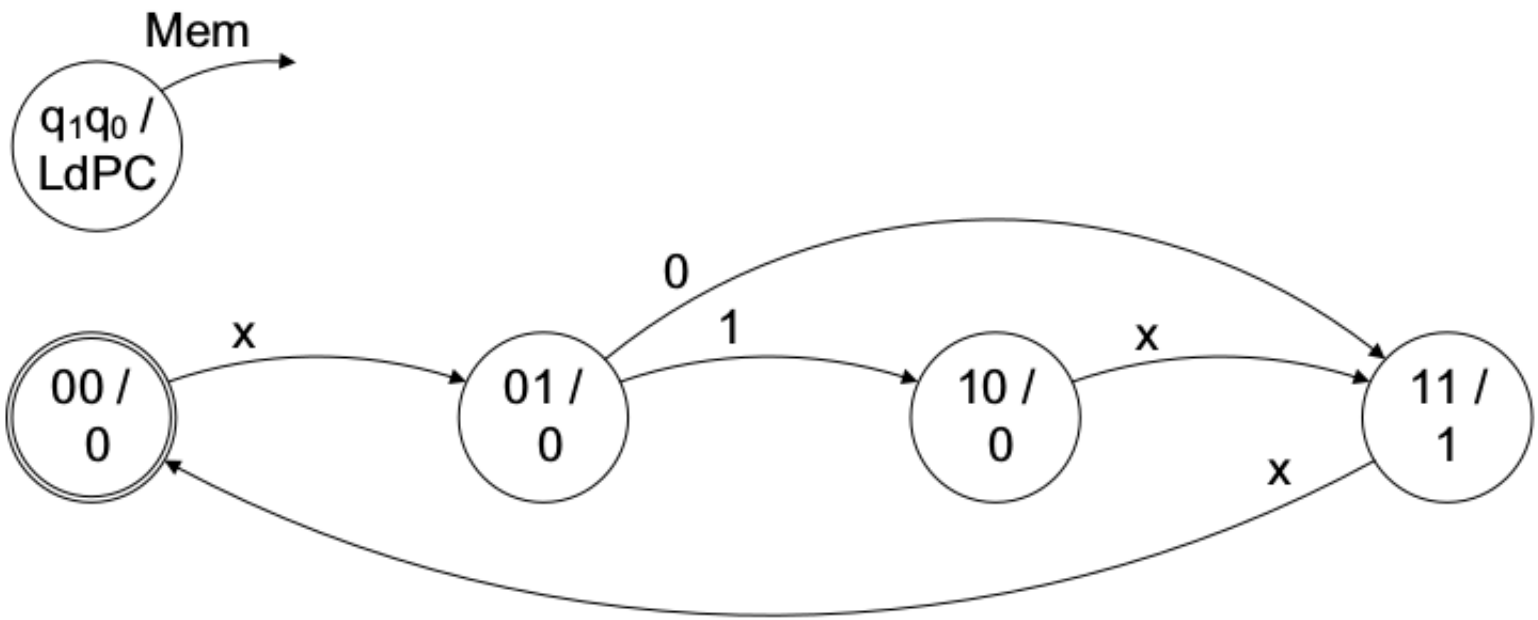
- Cal saber **deduir** el seu contingut a partir dels esquemes lògics

Dirección					Contenido																					
In4	In3	In2	In1	In0	Out20	Out19	Out18	Out17	Out16	Out15	Out14	Out13	Out12	Out11	Out10	Out9	Out8	Out7	Out6	Out5	Out4	Out3	Out2	Out1	Out0	
I<15>	I<14>	I<13>	I<12>	I<8>	Mem	Bnz	Bz	Wr-Mem	Rdln	WrOut	WrD1	Byte	Rb/N	-/i/l/a1	-/i/l/a0	OP1	OP0	MxN1	MxN0	MxF	F2	F1	F0	Mx@D1	Mx@D0	
0	0	0	0	x	0	0	0	0	0	0	1	x	1	0	0	0	0	x	x	0	x	x	x	0	0	AL
0	0	0	1	x	0	0	0	0	0	0	1	x	1	0	0	0	1	x	x	0	x	x	x	0	0	CMP
0	0	1	0	x	0	0	0	0	0	0	1	x	0	0	0	0	0	0	0	1	1	0	0	0	1	ADDI
0	0	1	1	x	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	1	0	0	0	1	LD
0	1	0	0	x	1	0	0	1	0	0	0	0	0	x	x	0	0	0	0	1	1	0	0	x	x	ST
0	1	0	1	x	1	0	0	0	0	0	1	1	0	0	1	0	0	0	0	1	1	0	0	0	1	LDB
0	1	1	0	x	1	0	0	1	0	0	0	1	0	x	x	0	0	0	0	1	1	0	0	x	x	STB
0	1	1	1	x	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	(NOP)
1	0	0	0	0	0	0	1	0	0	0	0	x	x	x	x	1	0	1	0	1	0	0	0	x	x	BZ
1	0	0	0	1	0	1	0	0	0	0	0	x	x	x	x	1	0	1	0	1	0	0	0	x	x	BNZ
1	0	0	1	0	0	0	0	0	0	0	1	x	0	0	0	1	0	0	1	1	0	0	1	1	0	MOVI
1	0	0	1	1	0	0	0	0	0	0	1	x	0	0	0	1	0	0	1	1	0	1	0	1	0	MOVHI
1	0	1	0	0	0	0	0	0	1	0	1	x	x	1	0	x	x	x	x	x	x	x	x	1	0	IN
1	0	1	0	1	0	0	0	0	0	1	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	OUT
1	0	1	1	x	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	(NOP)
1	1	x	x	x	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	(NOP)

SISC Harvard Multicycle CONTROL



- Graf d'estats del CLS que calcula LdPC
  - Compta 3 o 4 minicicles en funció de si és una instrucció ràpida (Mem="0") o lenta (Mem="1")



Índex

Temps d'execució d'un programa

- Computador Harvard unicicle
  - Cada instrucció triga 3.000 u.t.
  - $T_{execució} = N_{instruccions} \times 3.000 u.t.$
- Computador Harvard multicicle
  - Les instruccions lentes triguen $4 \times 750 = 3.000$ u.t.
  - Les instruccions ràpides triguen $3 \times 750 = 2.250$ u.t.
  - $T_{execució} = N_{instruccions\ ràpides} \times 2.250 + N_{instruccions\ lentes} \times 3.000 u.t.$
- Llevat que el programa només contingui instruccions lentes, el computador Harvard multicicle l'executarà en temps temps que el computador Harvard unicicle

Temps mig per instrucció

- A qualsevol processador:
  - $T_{execució} = N_{instruccions} \times T_{mig\ \ per\ \ instrucció}$
- Computador Harvard unicicle
  - $T_{mig\ \ per\ \ instrucció} = 3.000$ u.t./instrucció
- Computador Harvard multicicle
  - Dependrà de la proporció d'instruccions "lentes" i "ràpides"
  - Cal fer una **mitjana ponderada**
  - Exemple:
    - Temps mig per instrucció d'un programa que executa un 30% instruccions lentes i un 70% de ràpides?
    - $T_{mig\ \ per\ \ instrucció} = (0,3 \times 4 + 0,7 \times 3) \times 750 = 2.475$ u.t/instrucció

Comparació de rendiments: temps

- Donats els següents temps migs per instrucció a un programa:
  - Unicicle: 3.000 u.t./instrucció
  - Multicicle: 2.475 u.t./instrucció
- Quant **trigarà més** (en %) l'unicicle que el multicicle?
  - Regla de 3 amb el temps mig per instrucció

    2.475 u.t./inst. - 100
    3.000 u.t./inst. - x
  - x = 100 · 3.000 / 2.475 = 121,2 $\Longrightarrow$ triga un 21,2% més
- Quant **trigarà menys** (en %) el multicicle que l'unicicle?
  - Regla de 3 amb el temps mig per instrucció

    2.475 u.t./inst. - x
    3.000 u.t./inst. - 100
  - x = 100 · 2.475 / 3.000 = 82,5 $\Longrightarrow$ triga un 17,5% menys

Comparació de rendiments: velocitat

- Donats els següents temps migs per instrucció a un programa:
  - Unicicle: 3.000 u.t./instrucció
  - Multicicle: 2.400 u.t./instrucció
- Quant **és més lent** (en %) l'unicicle que el multicicle?
  - Regla de 3 amb la velocitat (instruccions per u.t.)

    1/2.400 inst./u.t. - 100
    1/3.000 inst./u.t. - x
  - x = 100 · (1/3.000) / (1/2.400) = 80 $\Longrightarrow$ és un 20% més lent
- Quant **és més ràpid** (en %) el multicicle que l'unicicle?
  - Regla de 3 amb la "velocitat"

    1/2.400 inst./u.t. - x
    1/3.000 inst./u.t. - 100
  - x = 100 · (1/2.400) / (1/3.000) = 125 $\Longrightarrow$ és un 25% més ràpid

Índex

Exercici

- Calculeu el **temps d'execució** del següent programa als computadors SISC Harvard unicicle i multicicle, i el **percentatge d'augment de la velocitat d'execució** del multicicle respecte al unicicle

```
MOVI    R0, 0x00
MOVHI   R0, 0x10
LD      R1, 0(R0)          ; Assumiu que R1 <- 0x002B
MOVI    R3, 0
MOVI    R4, 0x01
MOVI    R6, -1
AND     R2, R1, R4
ADD     R3, R3, R2
SHL     R1, R1, R6
BNZ     R1, -4
OUT     3, R3
```

Exercici

- Analitzant el codi veiem ...
  - Executa 6 instruccions abans d'arribar al bucle (una de elles lenta)
  - El bucle executa 4 instruccions i itera 6 cops
  - Executa 1 instrucció després del bucle
- $N_{instruccions} = 6 + (4 \times 6) + 1 = 31$
  - 30 ràpides i 1 lenta
- Temps d'execució:
  - $T_{execució\ unicicle} = 31 \times 3.000 = 93.000 u.t.$
  - $T_{execució\ multicicle} = 30 \times 2.250 + 1 \times 3.000 = 70.500 u.t.$
- Increment de velocitat? (N=$N_{instruccions}$=31)
  - Regla de tres:

    N/93.000 inst./u.t. - 100
    N/70.500 inst./u.t. - x
  - x = 100 · (N/70.500) / (N/93.000) = 100 · 93.000 / 70.500 = 131,9
    $\Longrightarrow$ és un 31,9% més ràpid

Exercicis a entregar a Atenea

- Enunciat disponible a Atenea
  - `https://atenea.upc.edu/pluginfile.php/3603450/mod_assign/introattachment/0/Tema%2012%20-%20Exercicis%20en%20paper.pdf`
- Entrega a Atenea fins el dilluns 7/12
  - Format PDF

Índex

Conclusions

- Hem finalitzat el disseny del computador Harvard
- Dues versions:
  - Harvard unicicle:
    - Totes les instruccions triguen el mateix temps en executar-se
  - Harvard multicicle:
    - Diferencia entre instruccions "ràpides" i "lentes"
    - No malbarata temps quan executi instruccions "ràpides"
    - Cal redefinir el temps de cicle
    - Requereix petites modificacions a la lògica de control i a la UPG
- Al proper tema evolucionarem el disseny fins al model von Neumann
  - Multicicle
  - Instruccions i dades comparteixen memòria RAM
  - No replica recursos
    - La ALU també s'encarregarà d'actualitzar el PC
- No oblideu fer el qüestionari ET12 i els exercicis en paper (slide 54)

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] *Harvard Mark I*, [Online]. Available: https://en.wikipedia.org/wiki/Harvard_Mark_I.

[2] *Sep 9, 1947 CE: World's First Computer Bug*, [Online]. Available: https://www.nationalgeographic.org/thisday/sep9/worlds-first-computer-bug/#:~:text=On%5C%20September%5C%209%5C%2C%5C%201947%5C%2C%5C%20a,bug%5C%20identified%5C%20in%5C%20a%5C%20computer..

Introducció als Computadors

Tema 12: Computadors Harvard unicicle i multicicle

http://personals.ac.upc.edu/enricm/Docencia/IC/IC12.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.