Computer Science & Engineering Department

Calculatoare Numerice

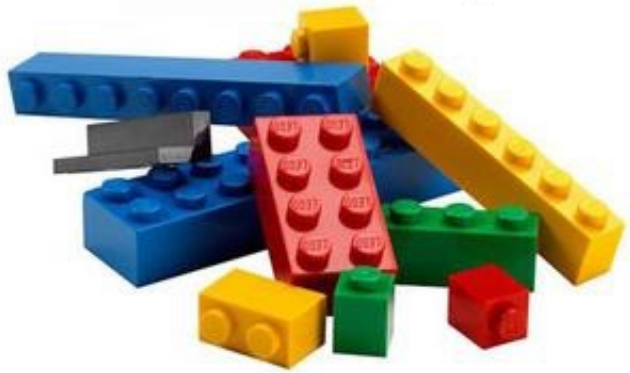
– Cursul 4 –

Calculatorul SAP-1

Facultatea de Automatică și Calculatoare
Universitatea Politehnica București

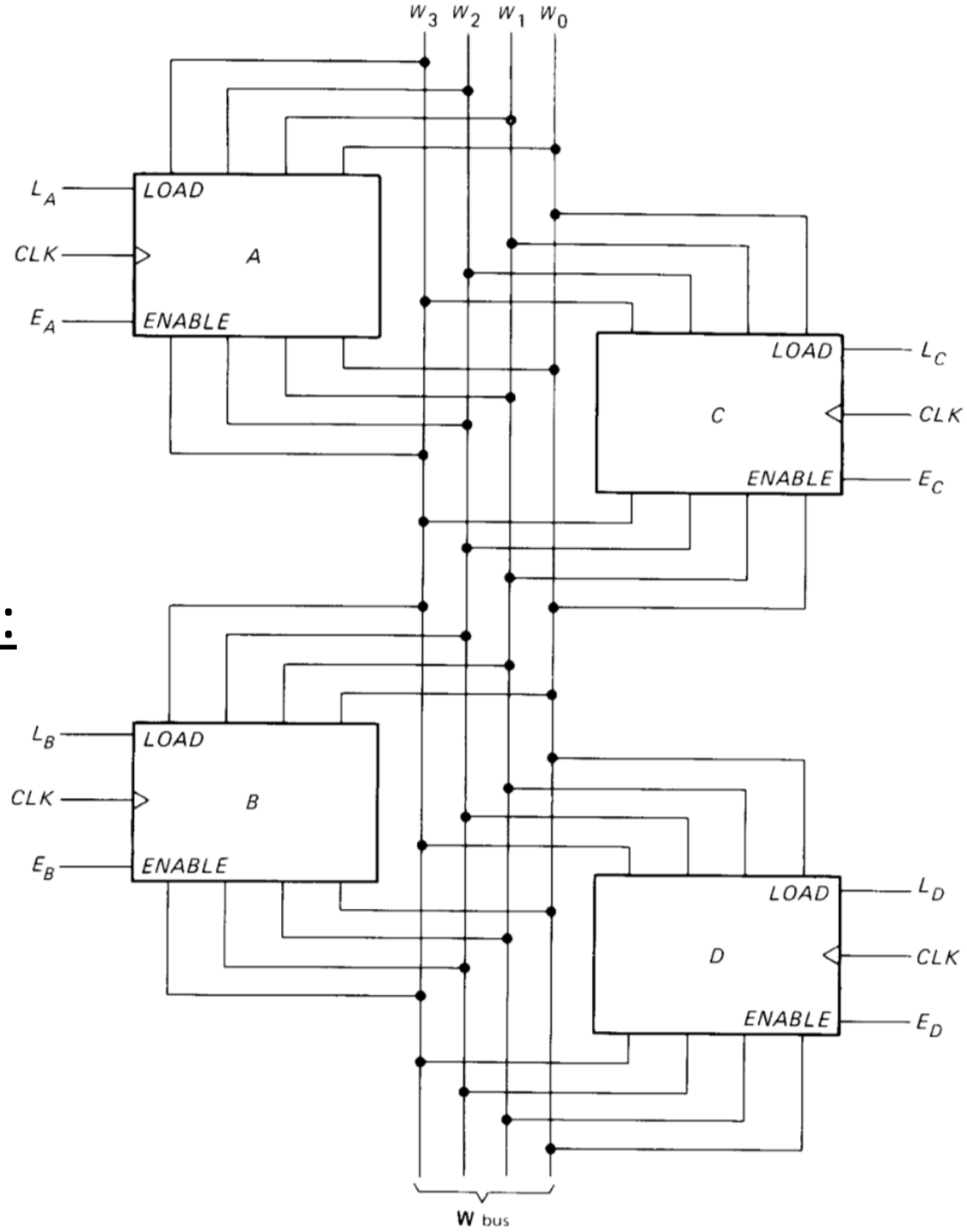
What people think building a computer is like

What it’s actually like



Bus-Organized Computer

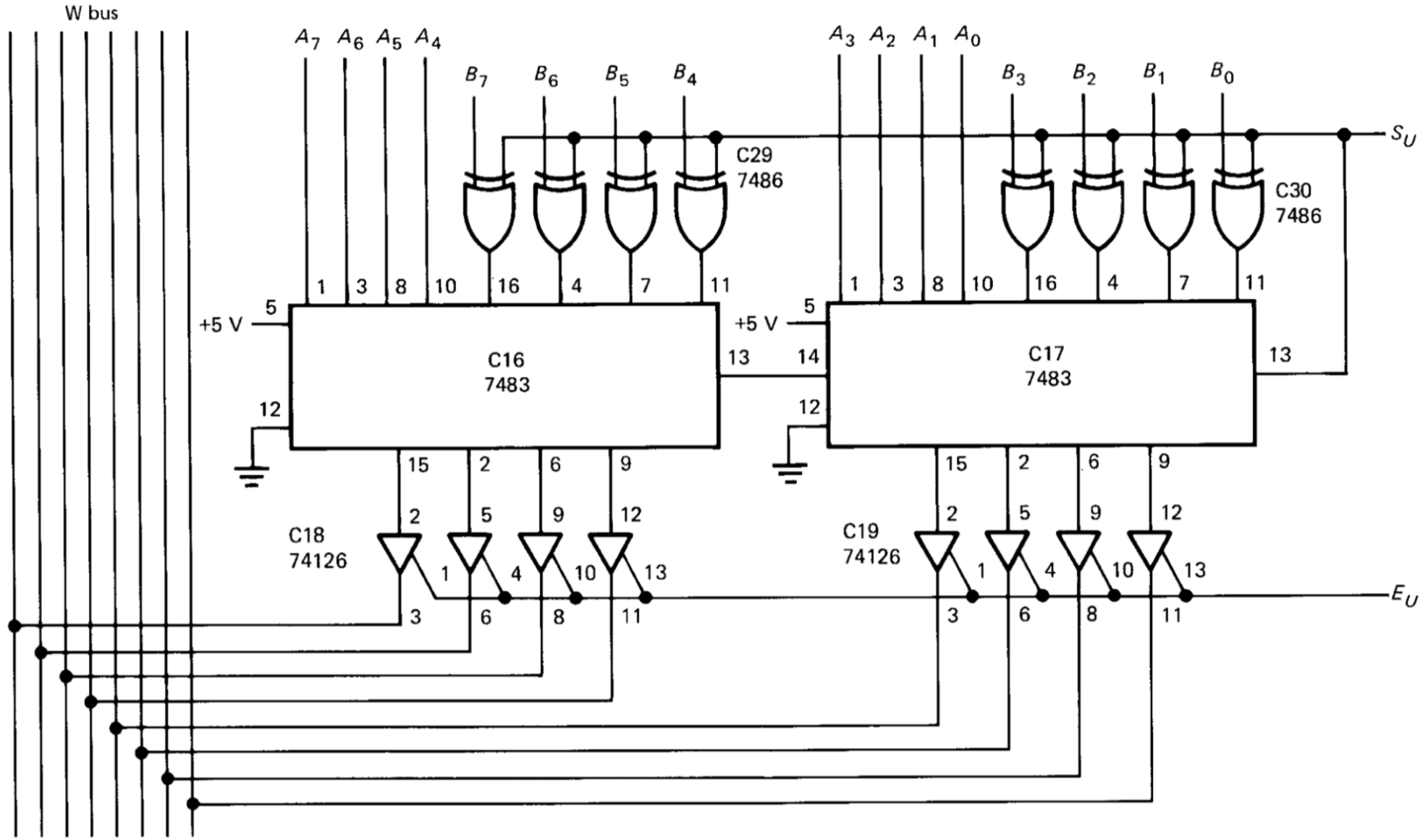
Patru registre cu încărcare paralelă de 4 biți conectate la aceeași magistrală W.
Ieșirile fiecărui registru sunt three-state și controlate prin liniile ENABLE.



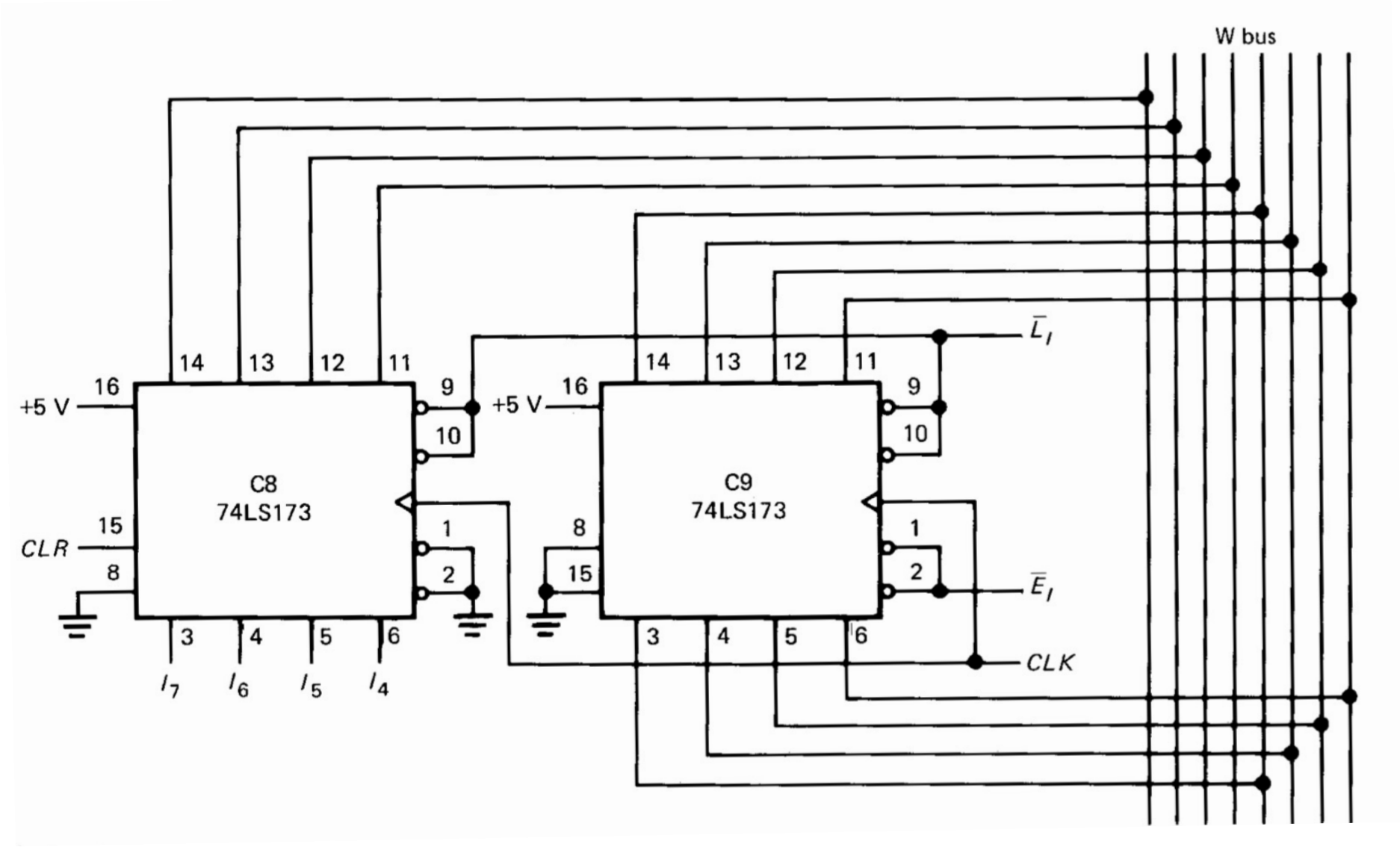
Transfer de date din Registrul A în Registrul D:

- inițial toate semnalele (ENABLE, LOAD și CLK) sunt inactive (= 0 logic)
- $E_A = 1$, $E_B = 0$, $E_C = 0$, $E_D = 0$
- $L_D = 1$, $L_A = 0$, $L_B = 0$, $L_C = 0$
- CLK ↑
- toate semnalele revin la starea inițială

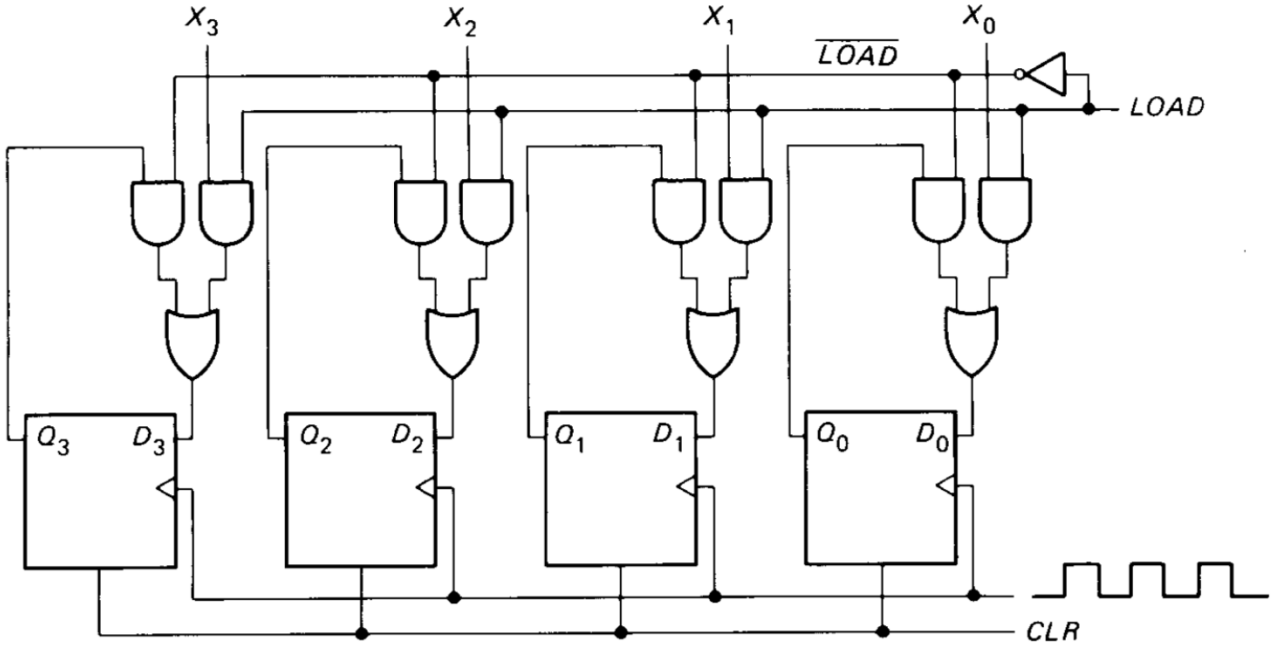
Exemplu: UAL conectat la o magistrală de 8 biți



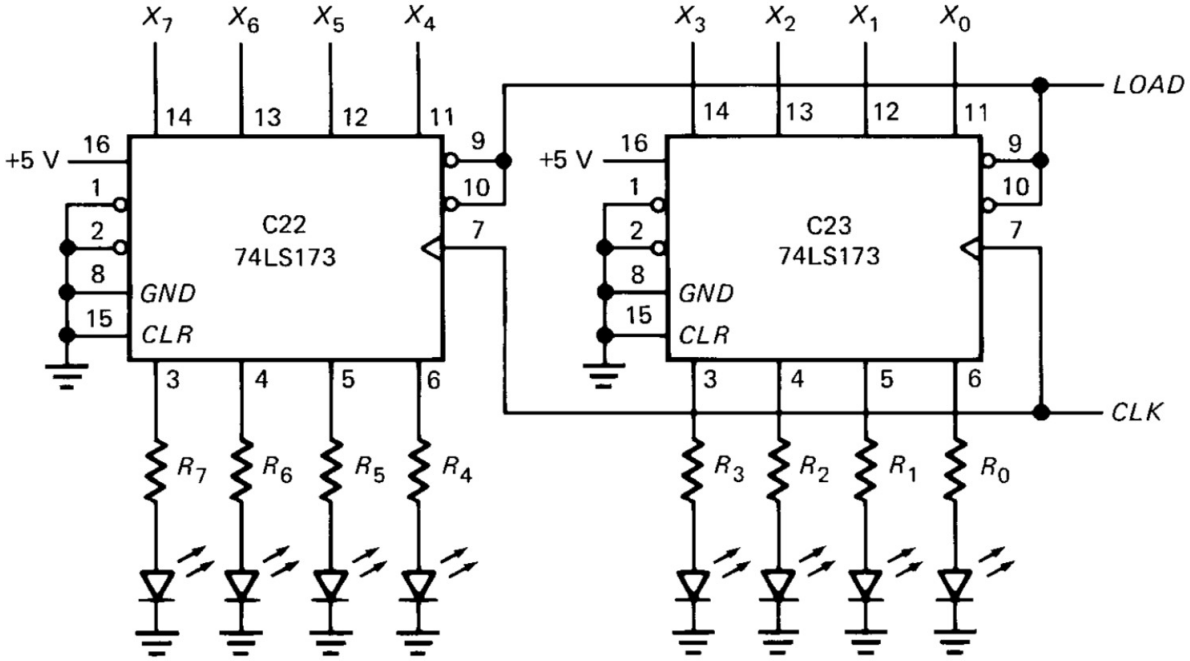
Alt exemplu: Registru 8 biți conectat la aceeași magistrală



Output registers



Registru de 4 biți cu încărcare paralelă



Registru de ieșire pe 8 biți cu încărcare paralelă și vizualizarea stării binare stocate pe LED-uri.
Fiecare circuit 74LS173 este un registru de 4 biți cu încărcare paralelă a datelor.

Ring Counters

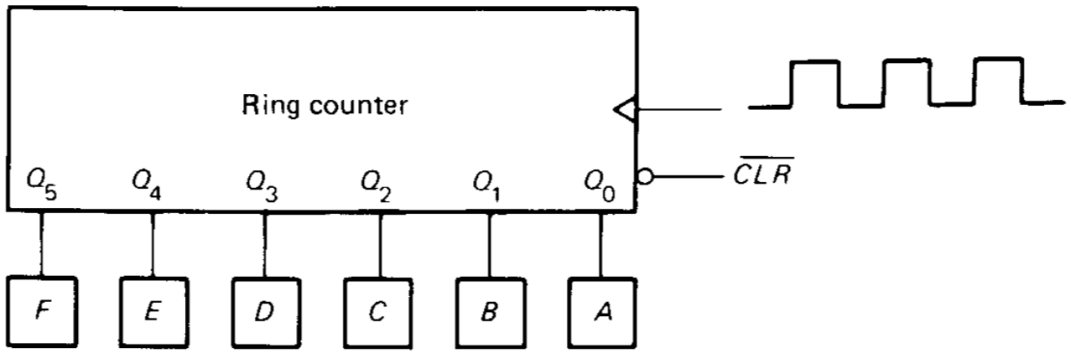
La fiecare impuls de ceas doar o singură ieșire a numărătorului este activă. Aceste circuite sunt folosite în procesoare pentru secvențierea operațiilor.

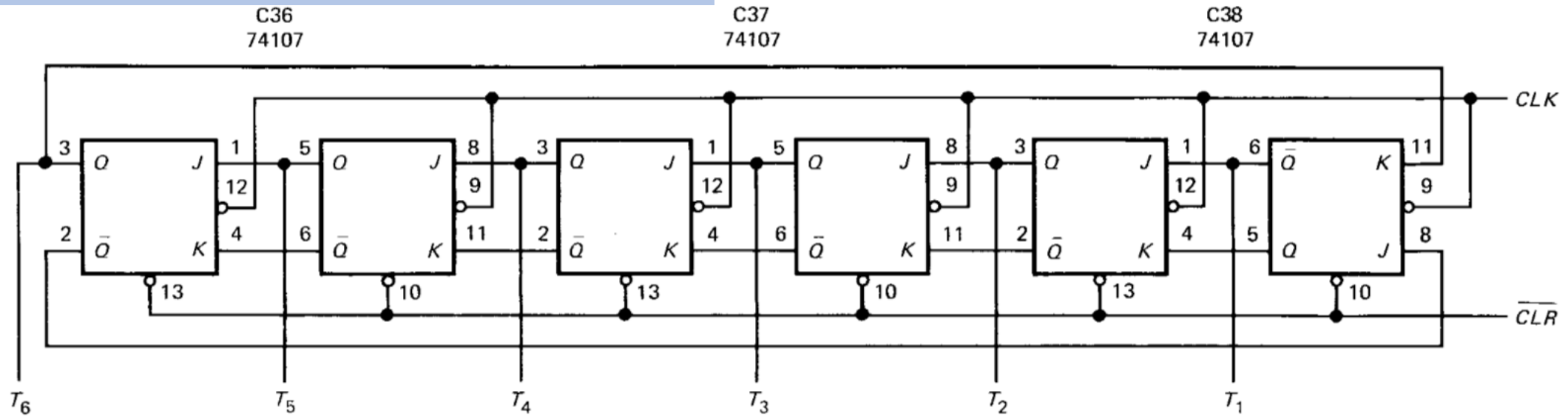
CLK1 – $T_1T_2T_3T_4T_5T_6$=100000

CLK2 – $T_1T_2T_3T_4T_5T_6$=010000

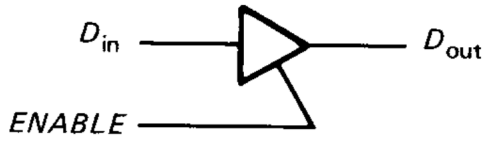
CLK3 – $T_1T_2T_3T_4T_5T_6$=001000

……

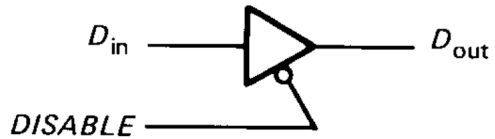




Three-state Buffers

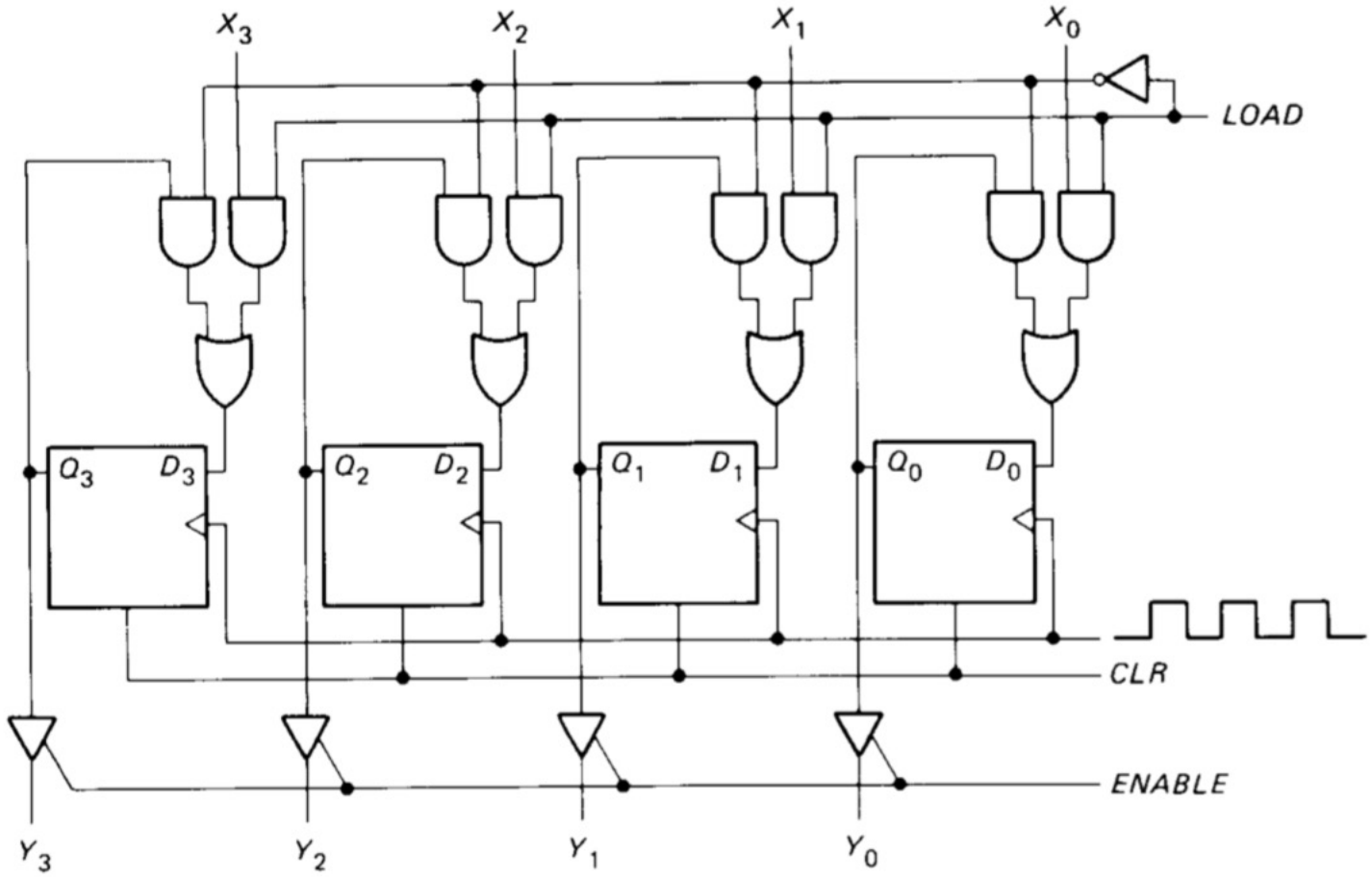


Switch three-state normal deschis



Switch three-state normal închis

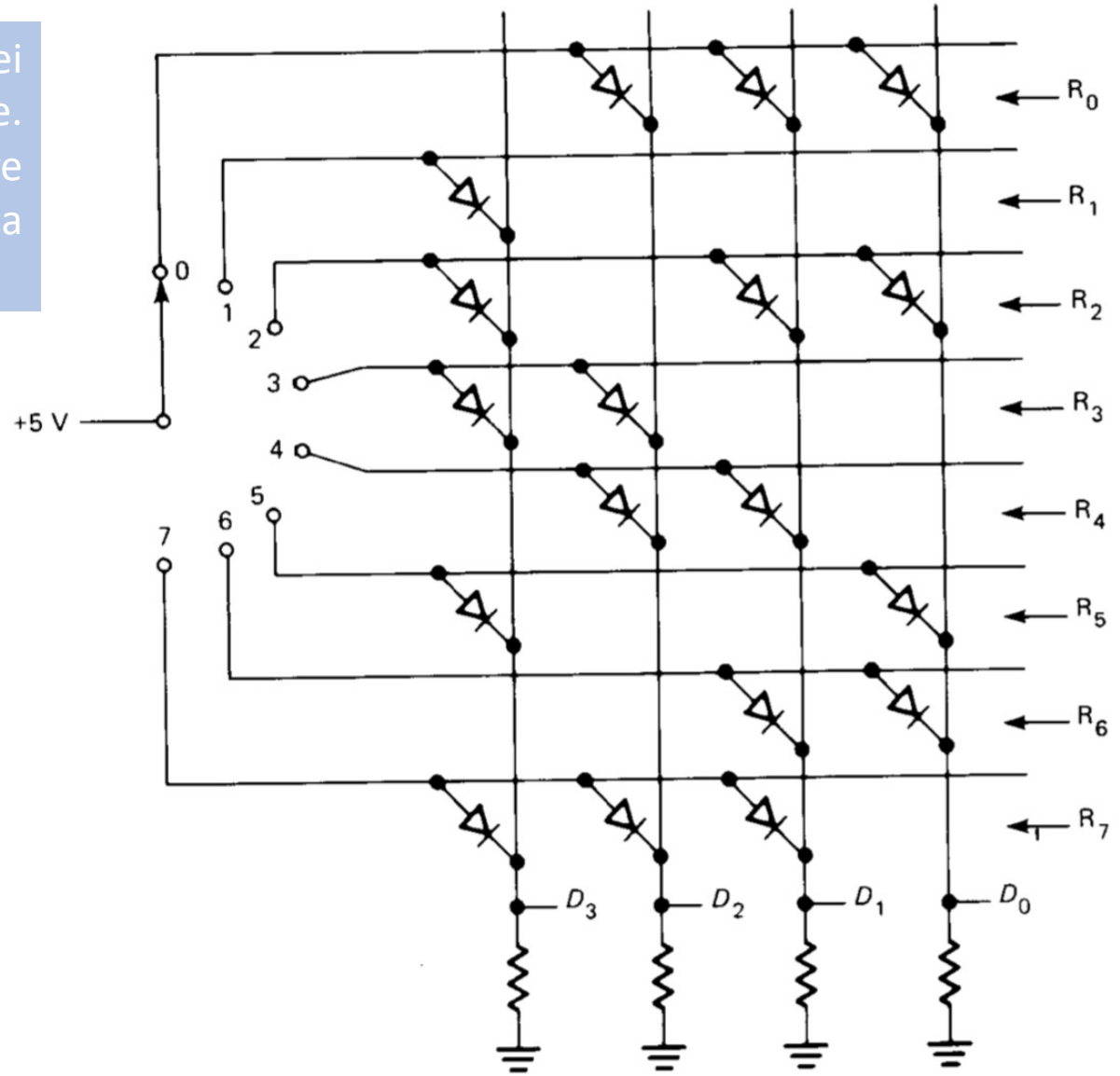
Registru de 4 biți cu încărcare paralelă (X) și ieșire three-state (Y)



ROM – Read-Only Memory

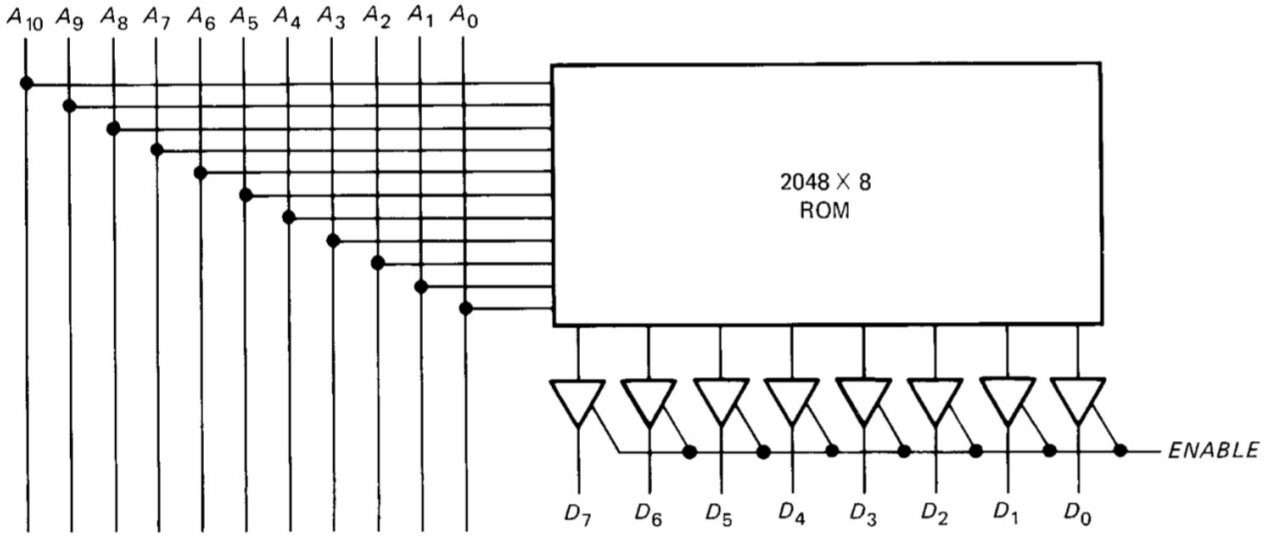
Posibil cea mai simplă (și ”antică”) implementare a unei memorii ROM: matrice de conexiuni folosind diode. Memoria codifică un 1 logic prin prezența unei diode între linia de adresă și cea de date. Lipsa unei diode este citită ca 0 logic. Memoria din figură are 8 registre de 4 biți.

Register	Address	Word
R_0	0	0111
R_1	1	1000
R_2	2	1011
R_3	3	1100
R_4	4	0110
R_5	5	1001
R_6	6	0011
R_7	7	1110

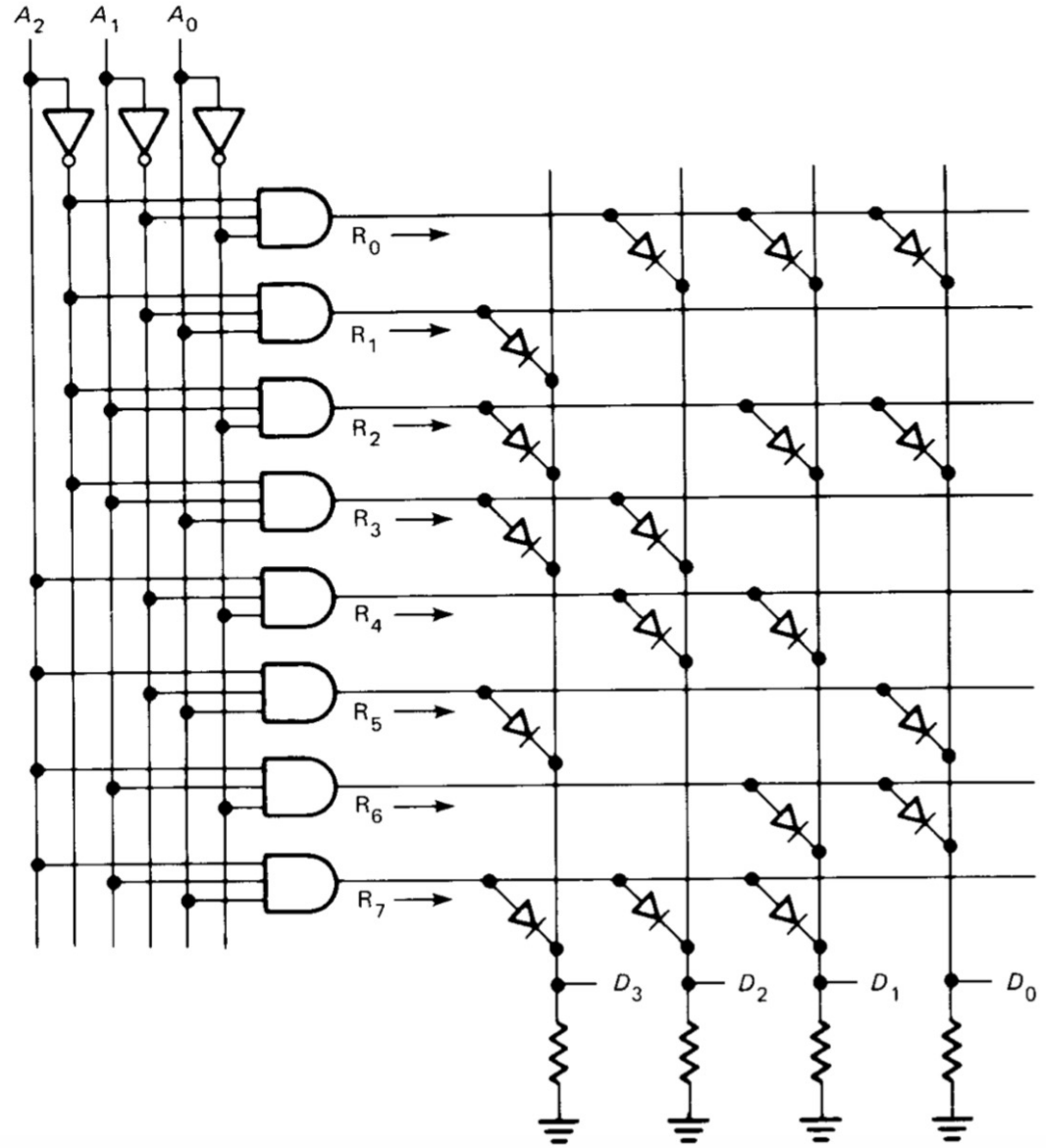


ROM ver. 2.0

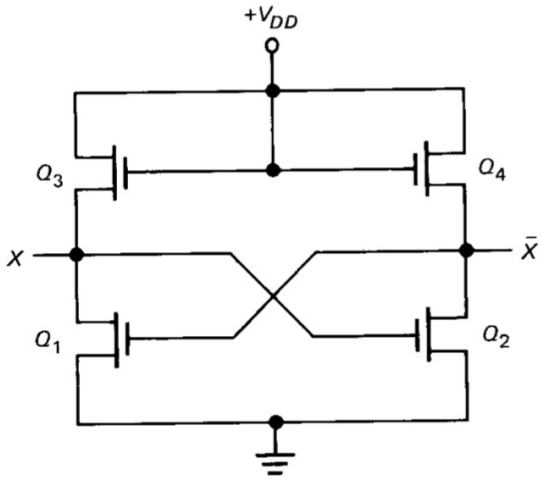
Îmbunătățire majoră:
- Decodificator pentru adresarea individuală a registrelor
- Liniile de adresă selectează cuvântul de 4 biți conrespunzător din memorie



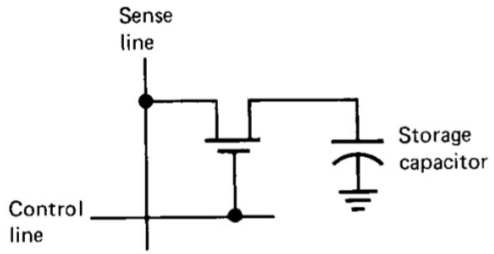
ROM 2K X 8 (2048 cuvinte de 8 biți) cu ieșire three-state



RAM – Random-Access Memory

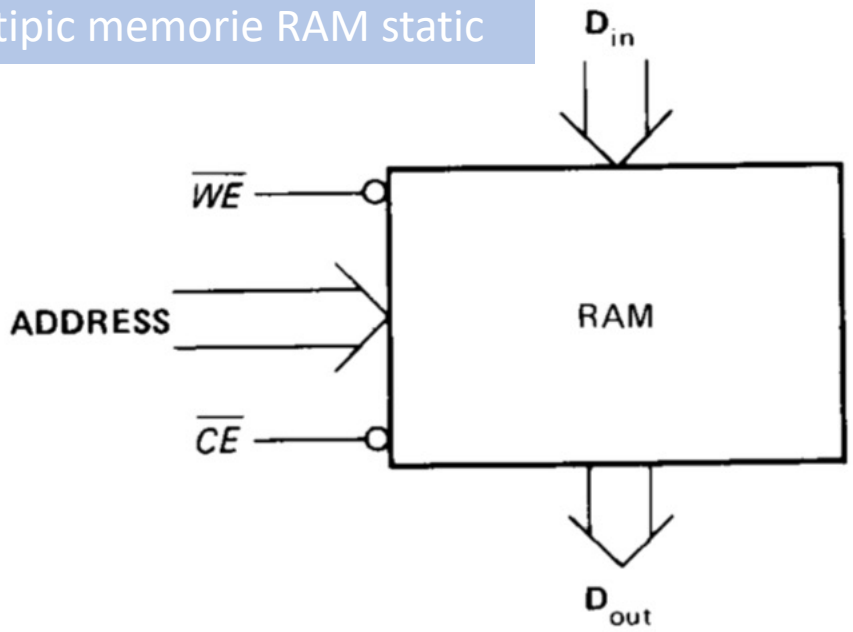


Celulă RAM static



Celulă RAM dinamic

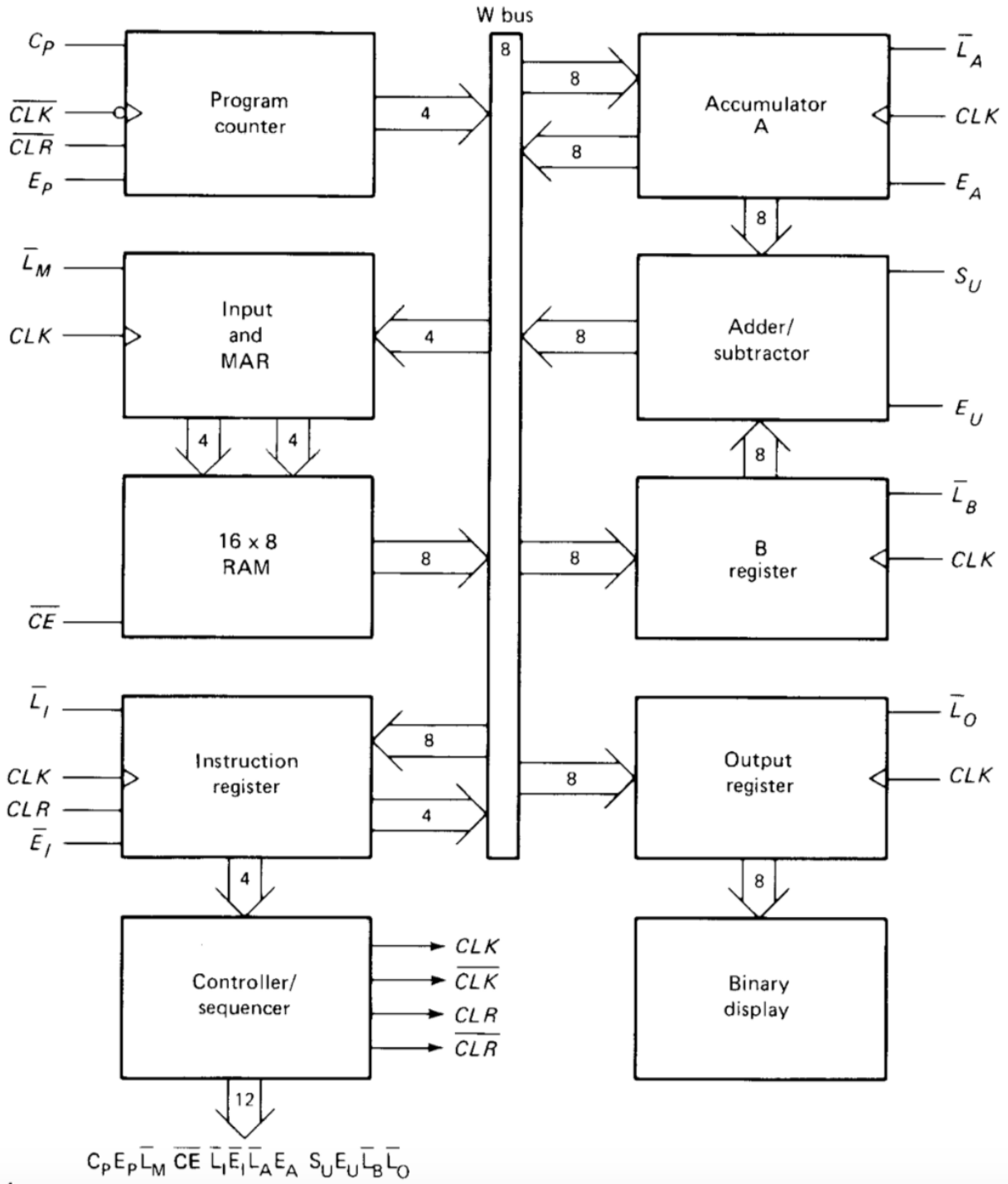
Circuit tipic memorie RAM static



$\overline{CE}$	$\overline{WE}$	Operation	Output
0	0	Write	Floating
0	1	Read	Connected
1	X	Hold	Floating

SAP-1: Simple-As-Possible Computer

- Un afișaj cu 8 LED-uri
- 16 octeți RAM
- 5 instrucțiuni (Load, Add, Substract, Out și Halt)
  - 3 cu 1 operand
  - 2 cu operanzi impliciți
- Arhitectură cu acumulator
  - Accumulator
  - Output Register
  - B Register
  - Memory Address Register (MAR)
  - Instruction Register (IR)



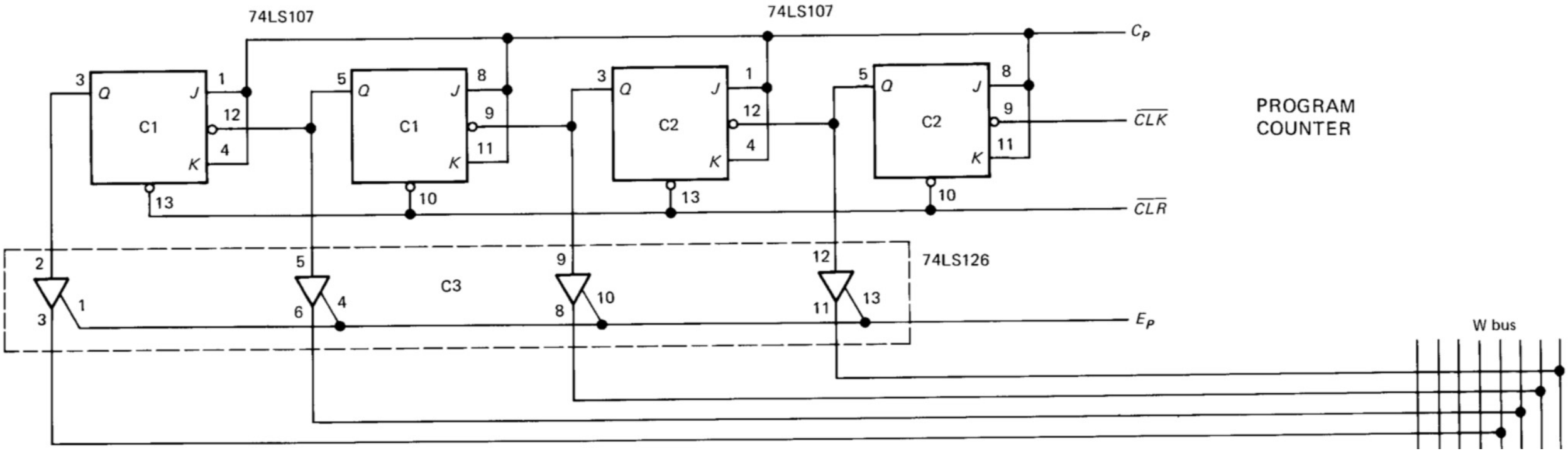
SAP-1 - Arhitectură

- 8-bit "W" bus.
- 4-bit Program Counter – numără de la 0 la 15, apoi se resetează
- 4-bit Memory Address Register (MAR)
- 16 Byte Memory
- 8-bit (1 Byte) Instruction Register (IR)
- 6-cycle controller with 12-bit microinstruction word
- 8-bit Accumulator
- 8-bit B Register
- 8-bit adder/subtractor
- 8-bit Output Register

Program Counter

- Instrucțiunile vor fi executate din memorie consecutiv, începând de la adresa 0000
- Primul pas pentru a executa instrucțiunea stocată în memorie este să generăm adresa 0000
- Adresa instrucțiunii curente este generată de Program Counter (PC), care este un numărător binar pe 4 biți - deci numără de la 0000 la 1111, apoi se resetează
  - Memoria de program are 16 octeți, deci PC va parcurge ciclic toate adresele din memorie
- Dacă PC = 0100, atunci instrucțiunea de la adresa 4 din memorie va fi executată
  - PC se comportă ca un registru pointer - marchează instrucțiunea din memorie ce va fi executată în continuare

Program Counter - implementare

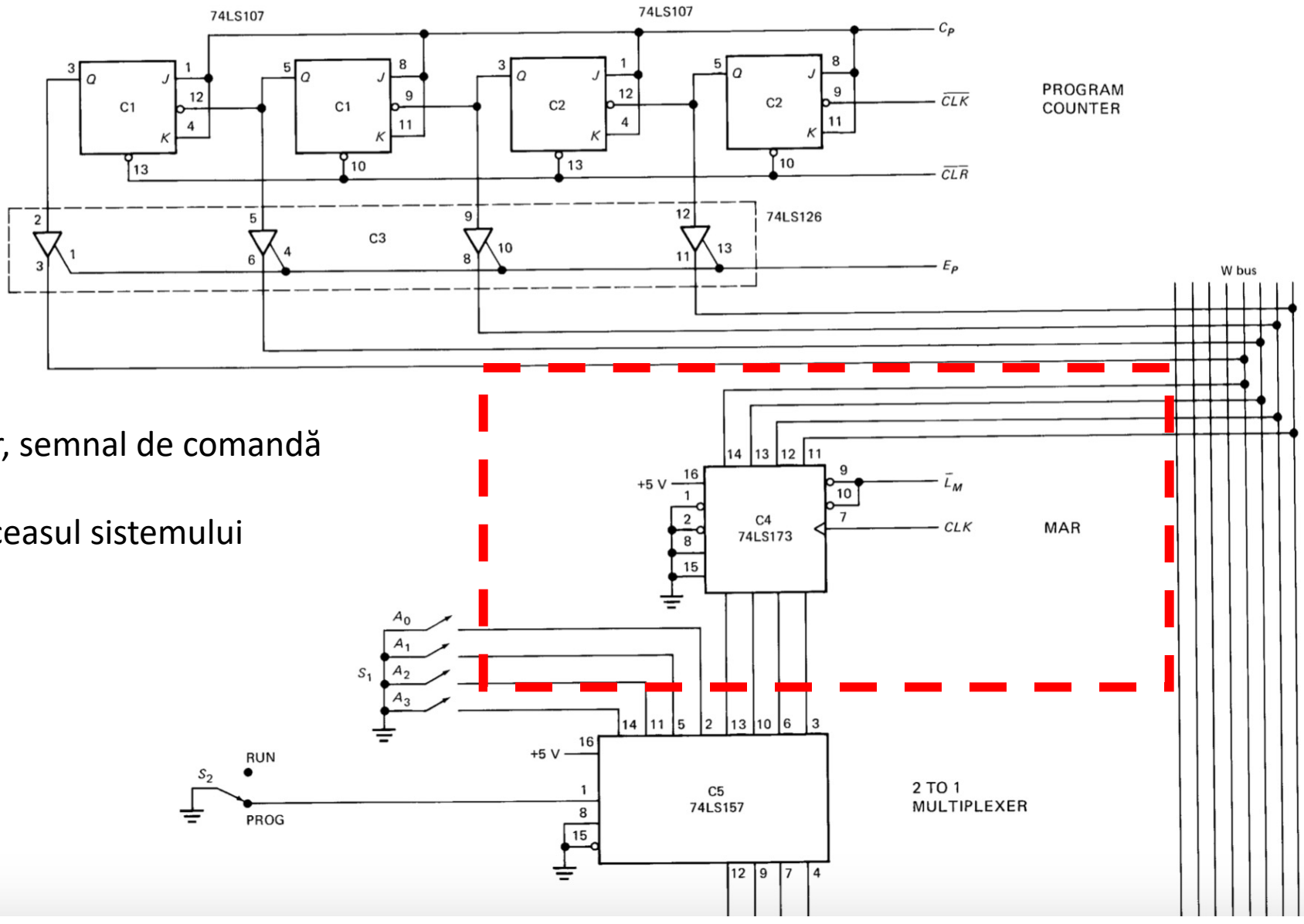


C_P – Count Pulse: semnal de control pentru incrementarea numărătorului
CLK – CLocK: numărătorul va fi incrementat dacă C_P=1 și CLK face o tranziție pe front negativ
CLR – CLeaR: resetează la 0000 numărătorul
E_P – Enable Program counter: comandă pentru switch-ul three-state, pune valoarea numărătorului pe magistrala W

Input and Memory Address Register (MAR)

- MAR stochează adresa de 4 biți a datelor sau instrucțiunilor din memorie
- Când SAP-1 este în Running Mode, această adresă e generată de Program Counter și stocată în MAR prin magistrala W
- În ciclii următori, MAR aplică această adresă memoriei RAM, pentru a citi datele sau instrucțiunile stocate acolo

Input and Memory Address Register (MAR)



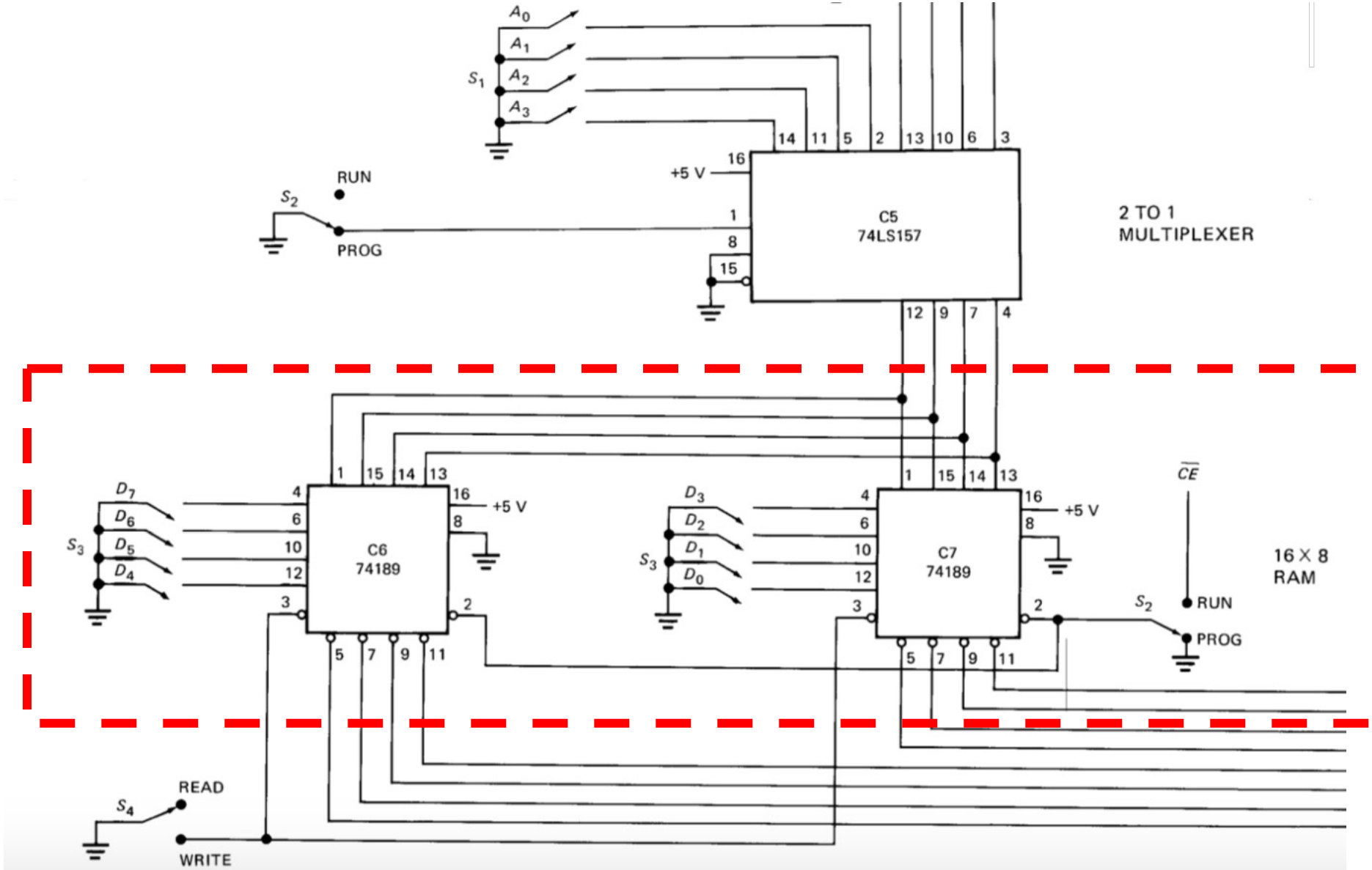
L_M – Load Mar, semnal de comandă pentru scriere

CLK – CLocK, ceasul sistemului

Memoria RAM

- Memoria RAM este implementată prin un circuit 16 x 8 biți SRAM (construit cu 2 integrate 74LS189 de 16 x 4 biți SRAM)
- Sunt 16 adrese de memorie, fiecare conține 8 biți de date sau instrucțiuni
- RAM-ul poate fi programat de utilizator prin intermediul unor butoane ce selectează adresele și datele. Acestea permit utilizatorului stocarea unui program înainte de rulare.
- În timpul rulării, memoria RAM primește adrese de 4 biți de la MAR și face operații de citire de la aceste adrese
- În acest fel, instrucțiunile sau datele stocate în RAM sunt plasate pe magistrala W

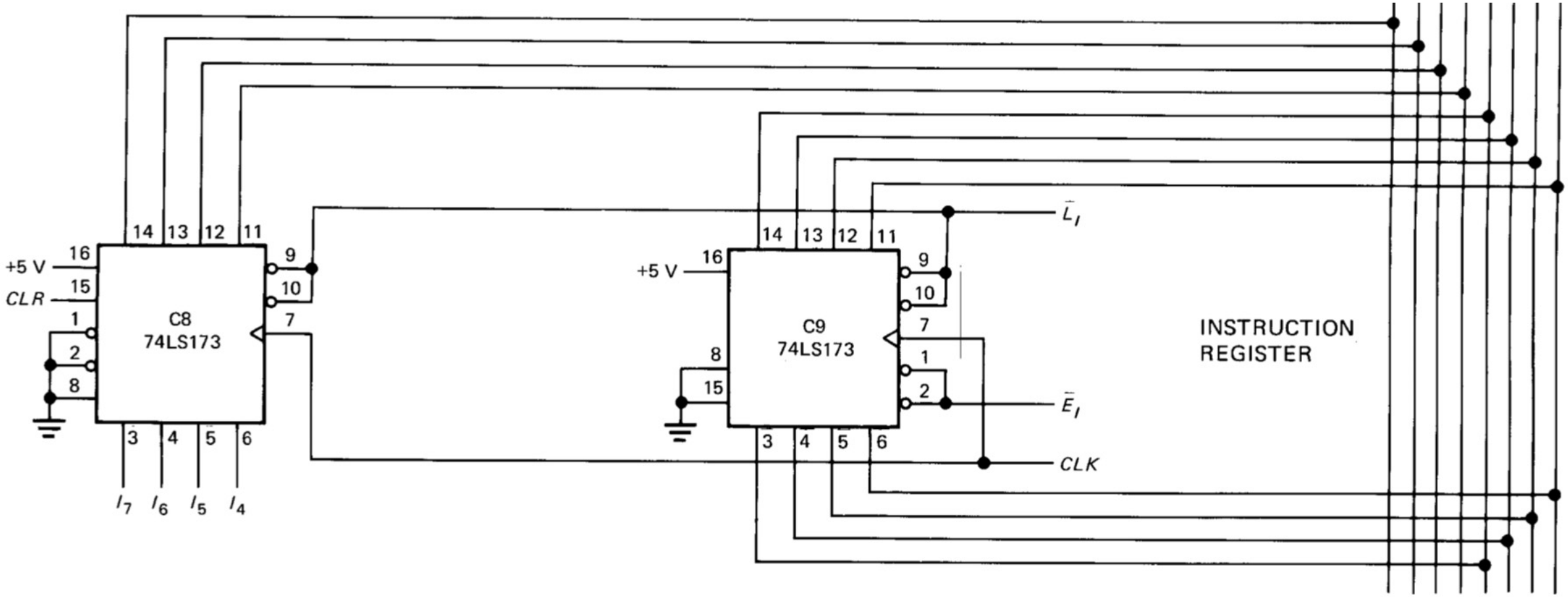
Memoria RAM


A0
A1
A2
A3
S1
14 11 5 2 13 10 6 3
16
+5 V
RUN
S2
PROG
1
8
15
C5
74LS157
2 TO 1
MULTIPLEXER
12 9 7 4
1 15 14 13
D7
D6
D5
D4
S3
4
6
10
12
3
C6
74189
16
+5 V
8
2
5 7 9 11
D3
D2
D1
D0
S3
C7
74189
CE
S2
RUN
PROG
16 X 8
RAM
READ
S4
WRITE

Instruction Register

- Când o instrucțiune este prezentă pe magistrala W din memorie, Instruction Register o va stoca pe următorul front pozitiv al ceasului
- Conținutul registrului instrucțiune este împărțit în două părți de 4 biți (nibbles).
- Upper nibble conține codul de instrucțiune și este legat direct la Unitatea de Control
- Lower nibble este o ieșire three-state care este citită de pe magistrala W atunci când este nevoie

Instruction Register



L_I – Load Instruction, semnal de încărcare a octetului instrucțiune de pe magistrala W

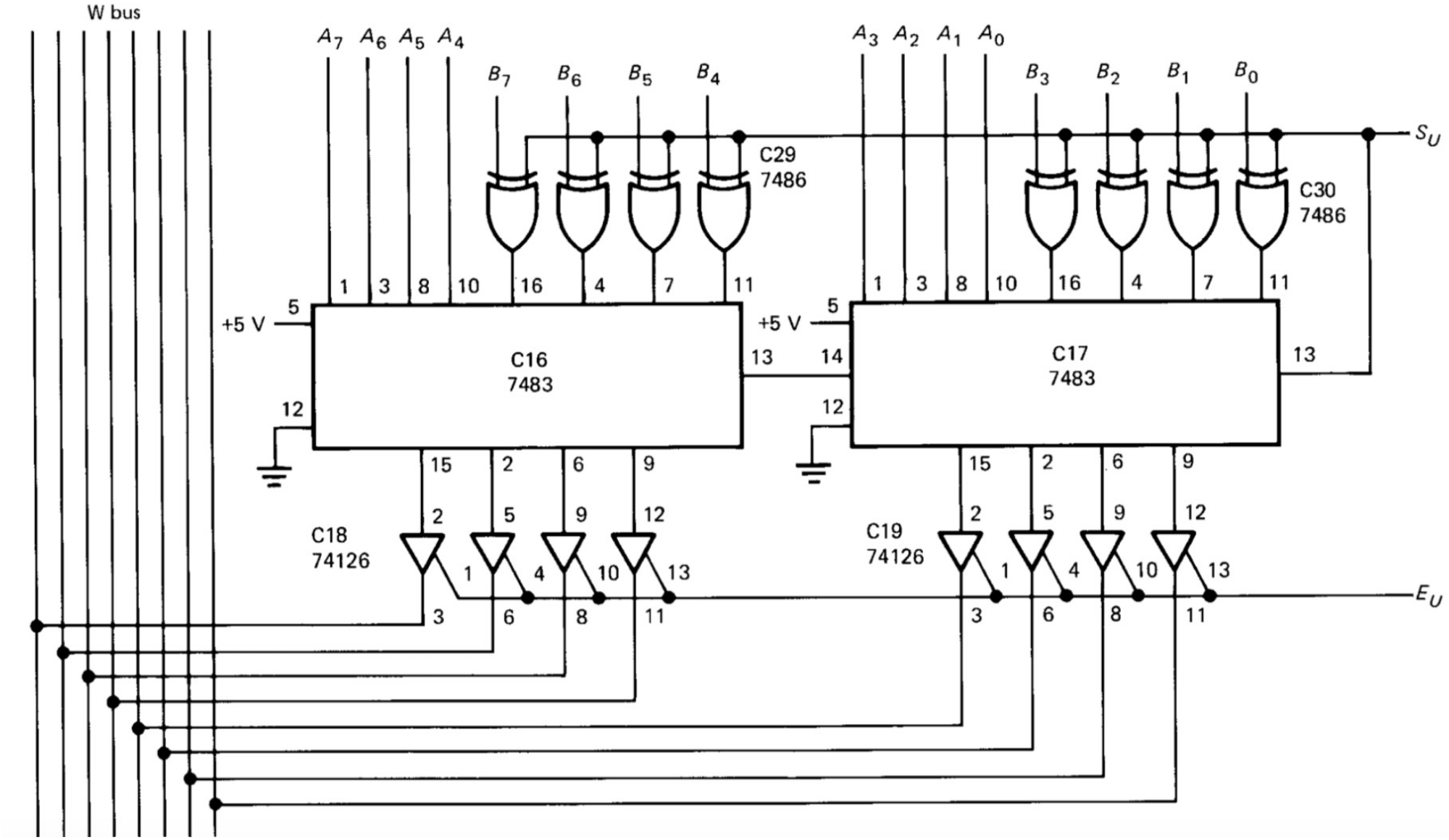
E_I – Enable Instruction, semnal de scriere a nibble-ului low pe magistrala W

I_4, I_5, I_6, I_7 – nibble-ul high al octetului de instrucțiune, conectat la Unitatea de Comandă

UAL – Unitatea Aritmetică-Logică

- SAP-1 folosește un sumator/scăzător cod complement al lui 2.
- Când semnalul S_u este 0 logic suma este S = A + B
- Când semnalul S_u este 1 logic suma este S = A + B + 1
- Sumatorul-scazătorul este logică asincronă, deci rezultatul se schimbă la scurt timp după aplicarea celor doi operanzi
- Când E_U este 1 logic, rezultatul va fi scris pe magistrala W

UAL – Unitatea Aritmetică-Logică

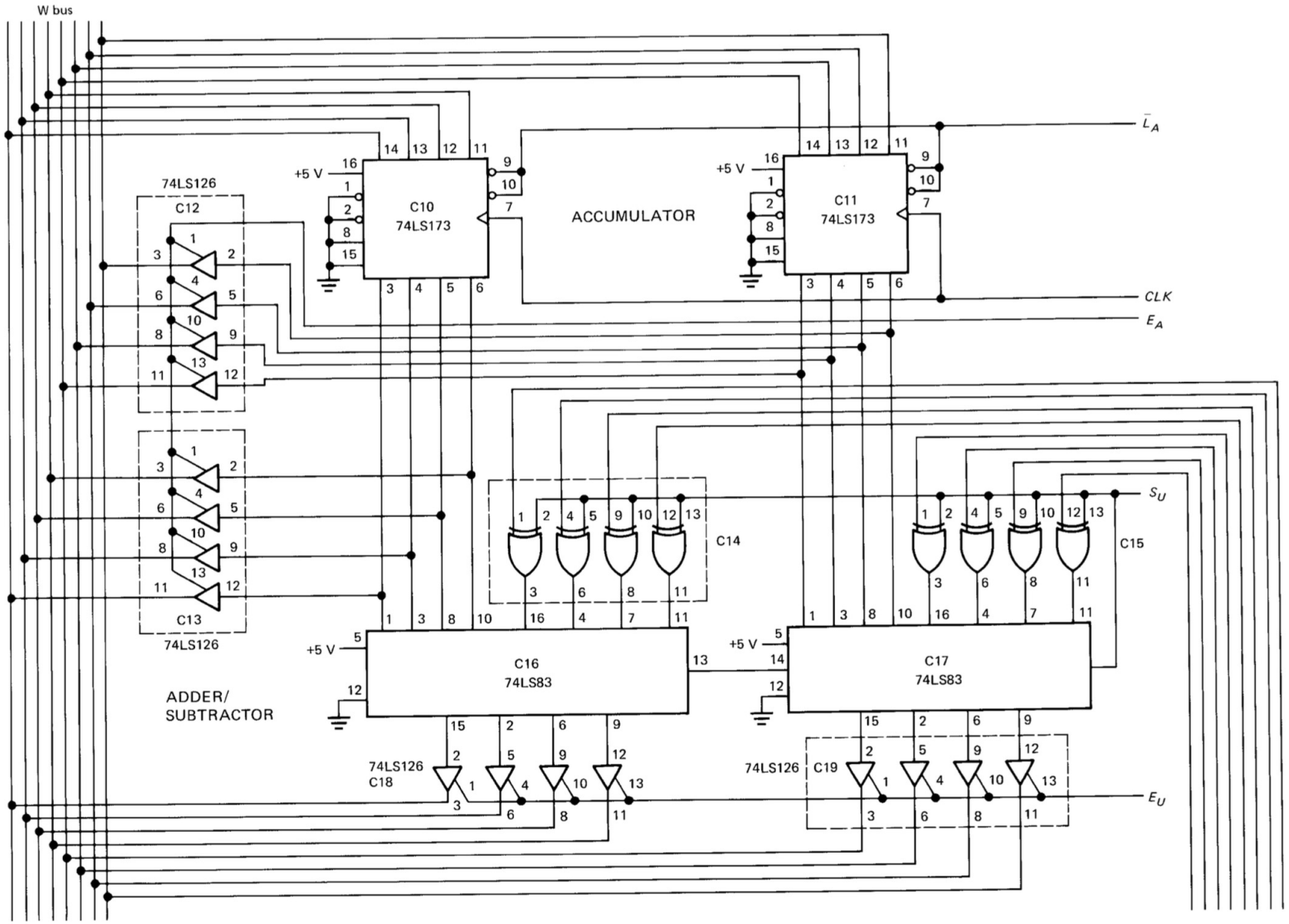


Registrul Acumulator

Pentru a aduna/scădea două numere A și B, acumulatorul stochează primul operand

- Are două ieșiri: una către UAL și cealaltă către magistrala W prin un buffer three-state
- Acumulatorul stochează rezultatul operației din UAL atunci când L_A este logic 0
- Valoarea stocată în Acumulator este disponibilă pe magistrala W atunci când E_A este logic 1

Complexul Acumulator - UAL

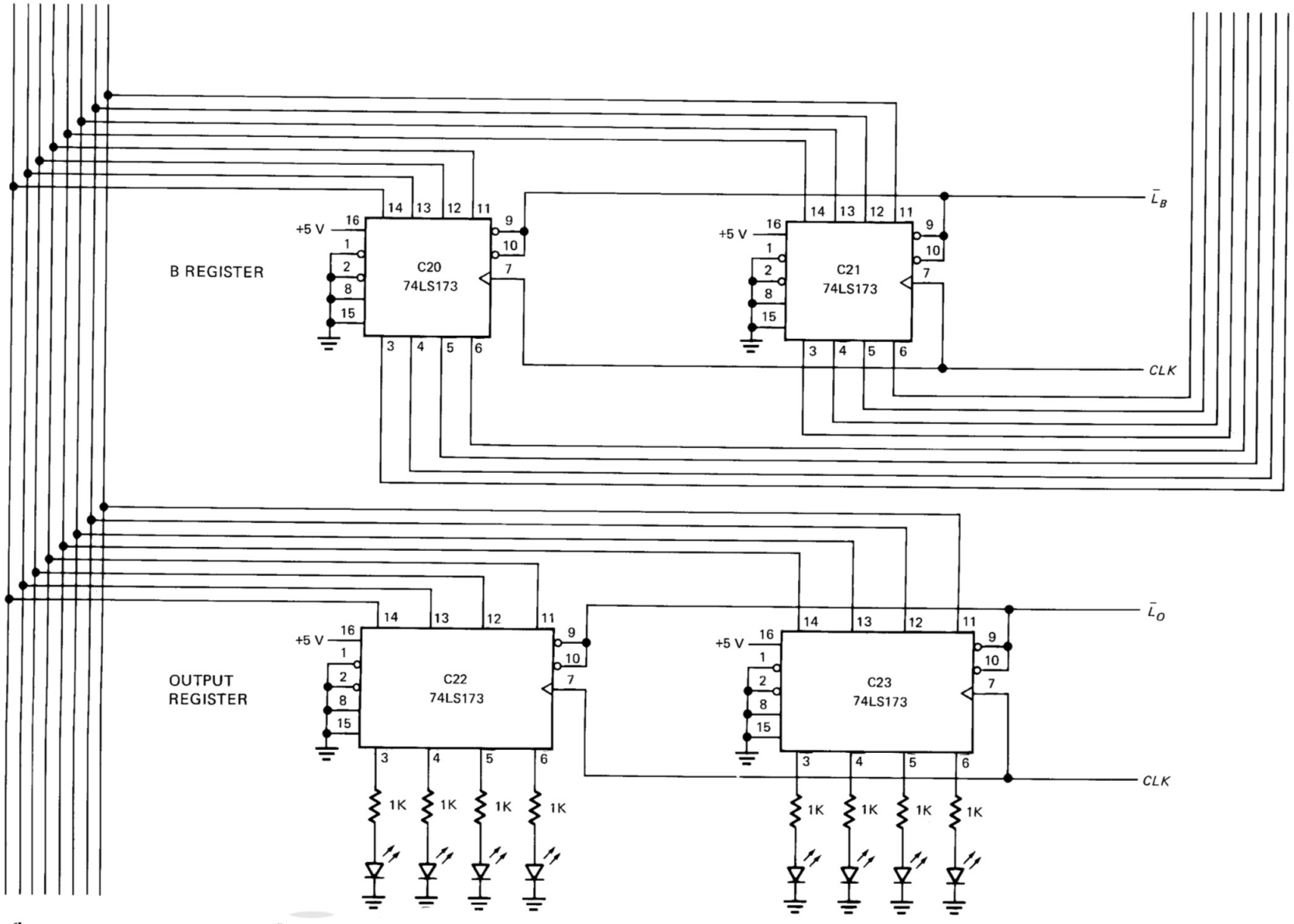


Registrul B și Registrul Output

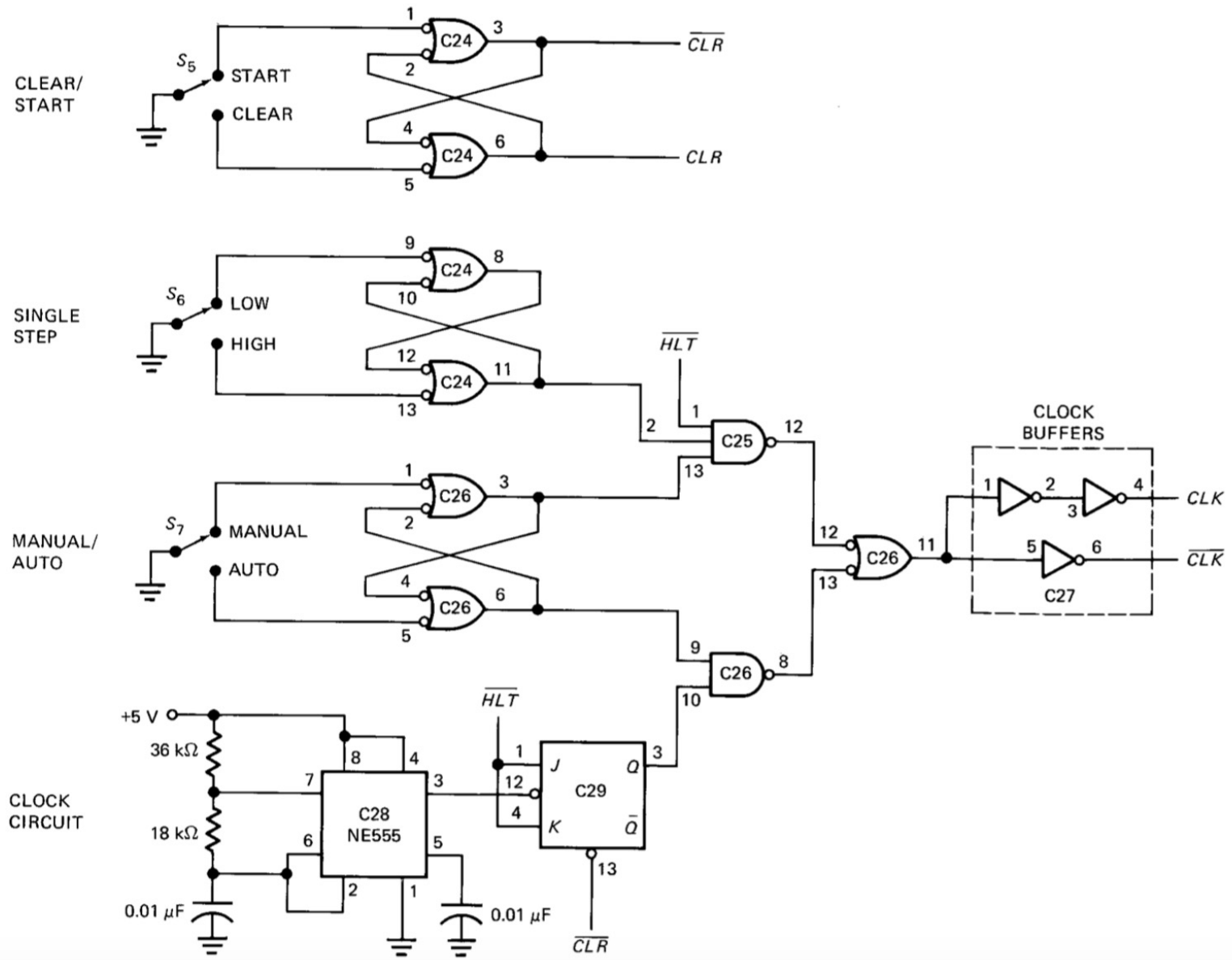
- Pentru a aduna/scădea două numere A și B, registrul B stochează al doilea operand
- Când datele sunt disponibile pe magistrala W și L_B este 0 logic, registrul B încarcă datele de pe magistrală

- La sfârșitul unei operații aritmetice, rezultatul este stocat în acumulator.
- Din acel registru, rezultatul trebuie încărcat în registrul Output pentru a fi vizualizat
- Acest lucru este făcut pe un posedge de ceas, când E_A este logic 1 și L_O este logic 0
- După acest pas, rezultatul este în registrul output și poate fi vizualizat pe cele 8 LED-uri conectate la ieșirea registrului
- Fiecare din cele 8 LED-uri se va aprinde sau stinge în funcție de valoarea binară stocată în registru (aprins pentru 1 logic și stins pentru 0 logic).

Registrul B și Registrul Output



Ceasul sistemului



Unitatea de Control

- Cei 12 biți de la ieșirea Unității de Control formează cuvântul de control. Acestea sunt semnalele de control pentru toate unitățile din interiorul procesorului nostru. Înainte de fiecare punere în funcțiune, semnalul Clear (CLR) resetează toate semnalele.
- Cele 12 linii ce transmit semnalele de control se numesc Magistrala de Control. Cuvântul de control are următorul format:

$$CON = C_P\ E_P\ \overline{L_M}\ \overline{C_E} \qquad \overline{L_L}\ \overline{E_L}\ \overline{L_A}\ E_A \qquad S_U\ L_U\ \overline{E_B}\ \overline{L_O}$$

- Acest cuvânt determină cum se vor comporta registrele la fiecare front crescător al semnalului de ceas. De exemplu, un E_P=1 și un L_M=0 semnalizează că valoarea din Program Counter va fi transferată în MAR la următorul front pozitiv de ceas. Alt exemplu, un E_L=0 și L_A=0 semnifică că la următorul front pozitiv al ceasului valoarea octetului adresat din memoria RAM va fi transferată în registrul acumulator.

Ring Counter pentru Unitatea de Comandă

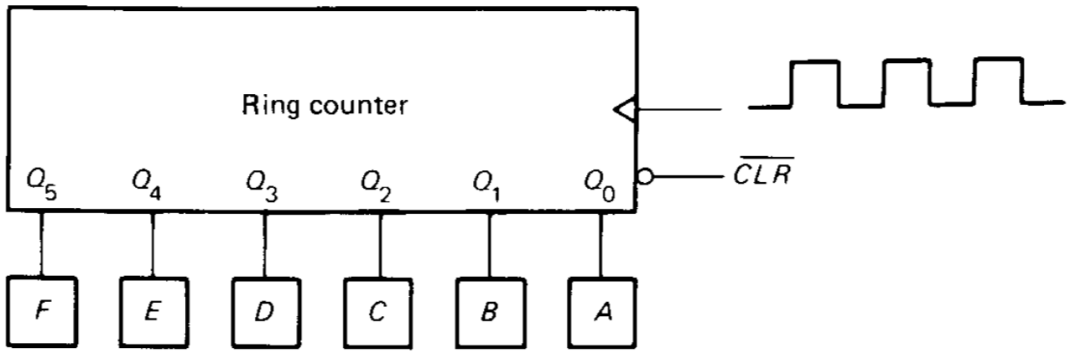
Fiecare operație SAP-1 se execută în șase cicli de ceas. Fiecare pas din acești 6 constituie o microinstrucțiune. Secvențierea acestor microinstrucțiuni este realizată cu ring counter-ul atașat unității de comandă.

CLK1 – $T_1T_2T_3T_4T_5T_6$=100000

CLK2 – $T_1T_2T_3T_4T_5T_6$=010000

CLK3 – $T_1T_2T_3T_4T_5T_6$=001000

……



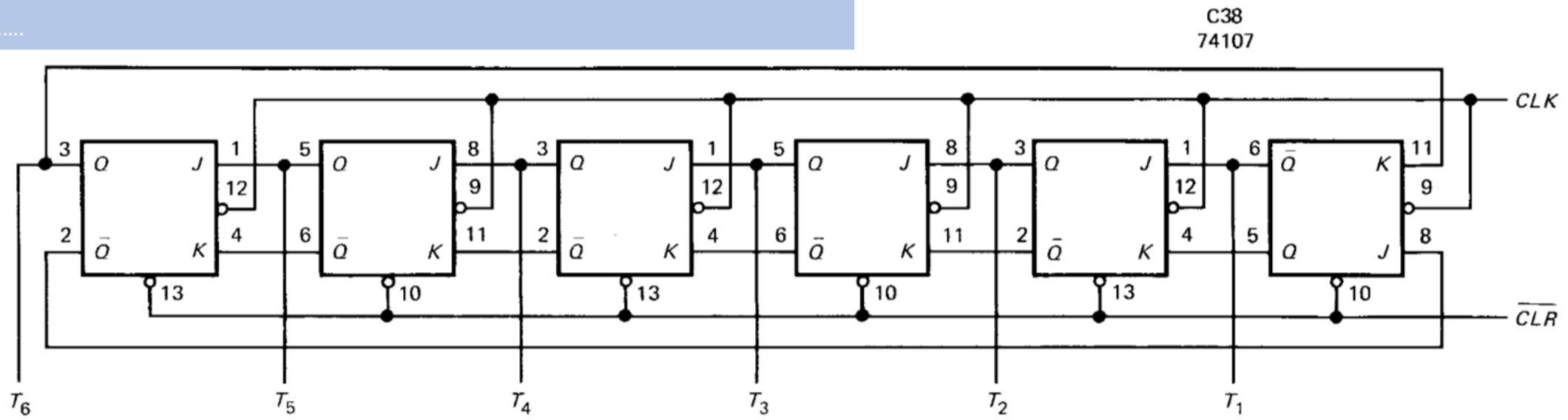
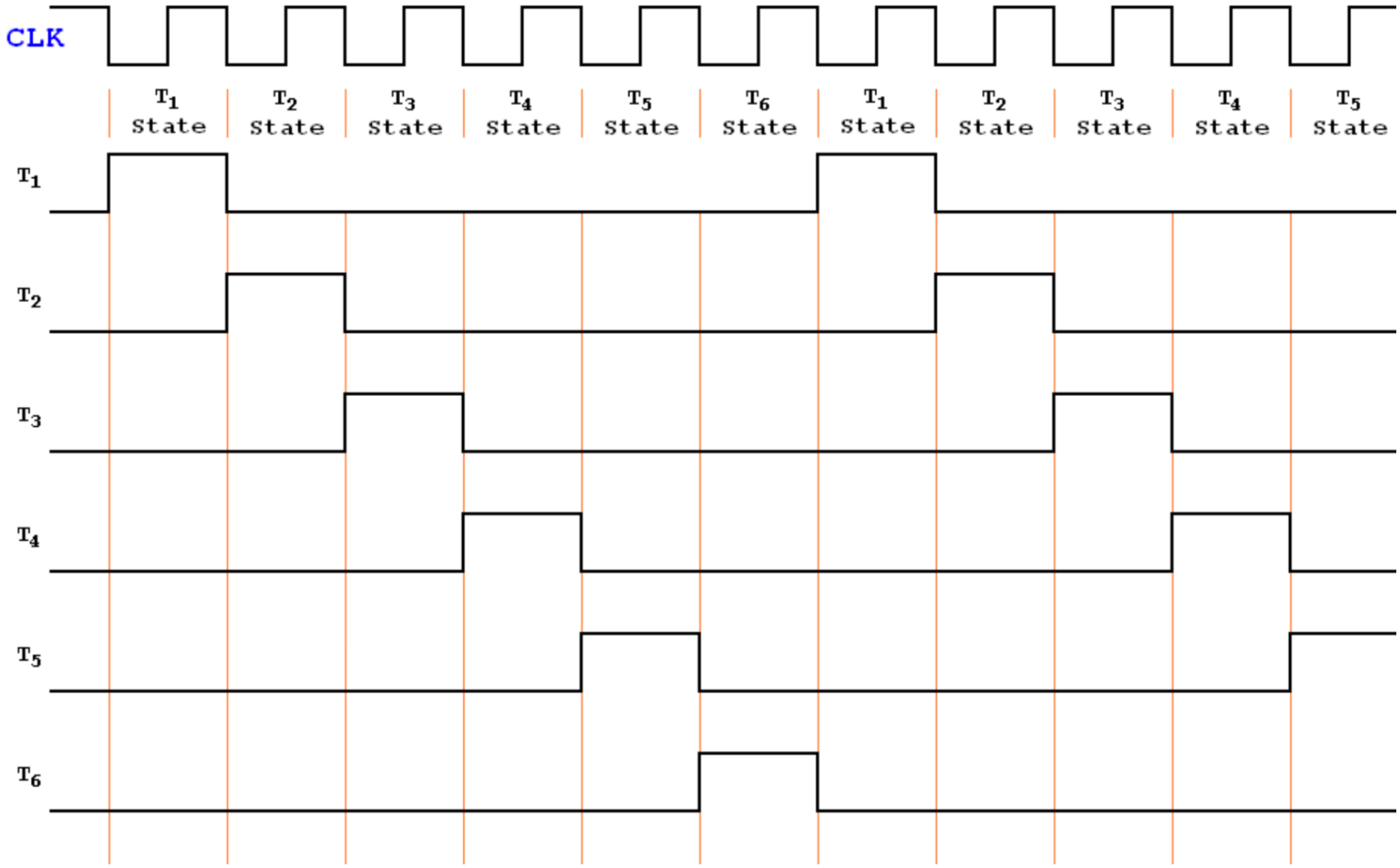
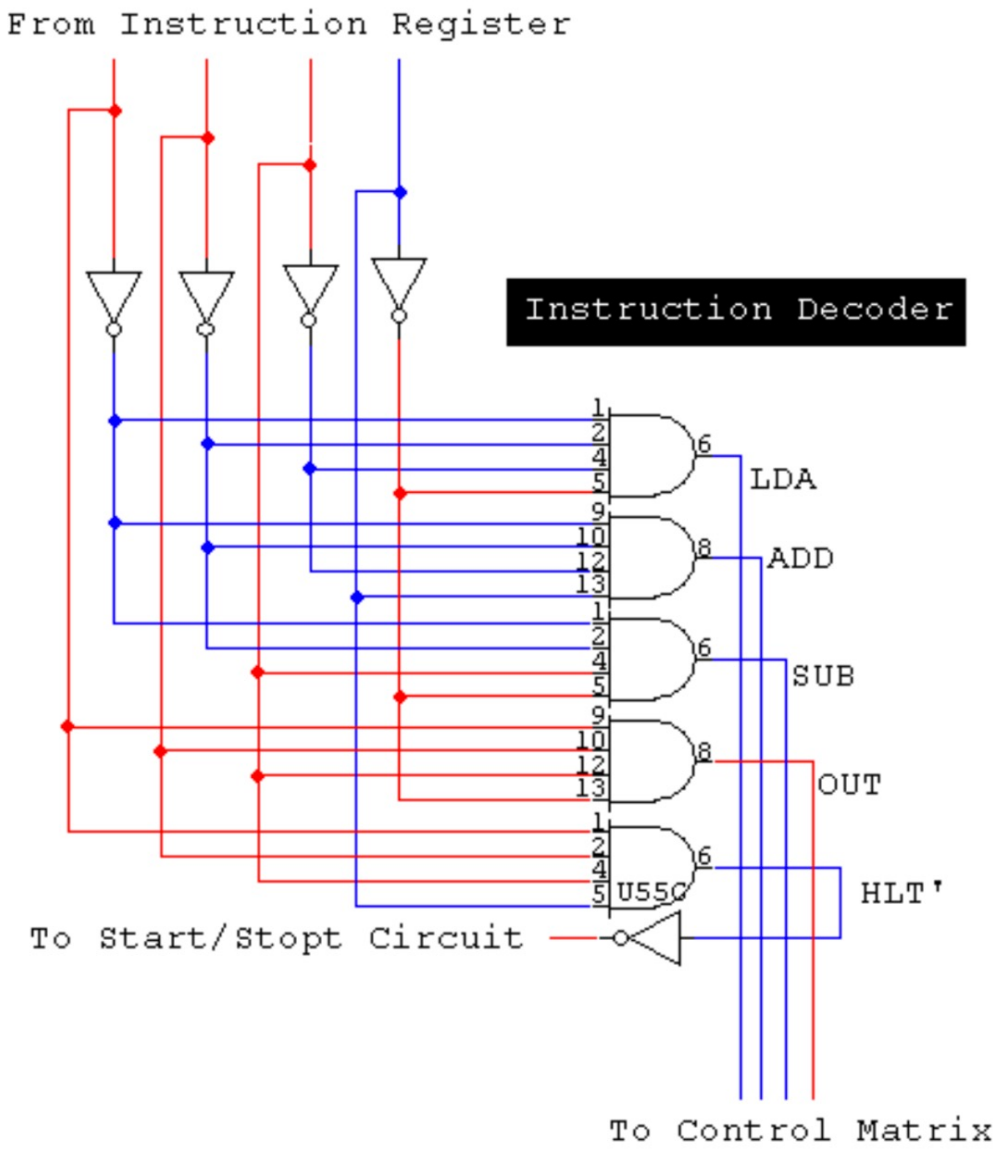


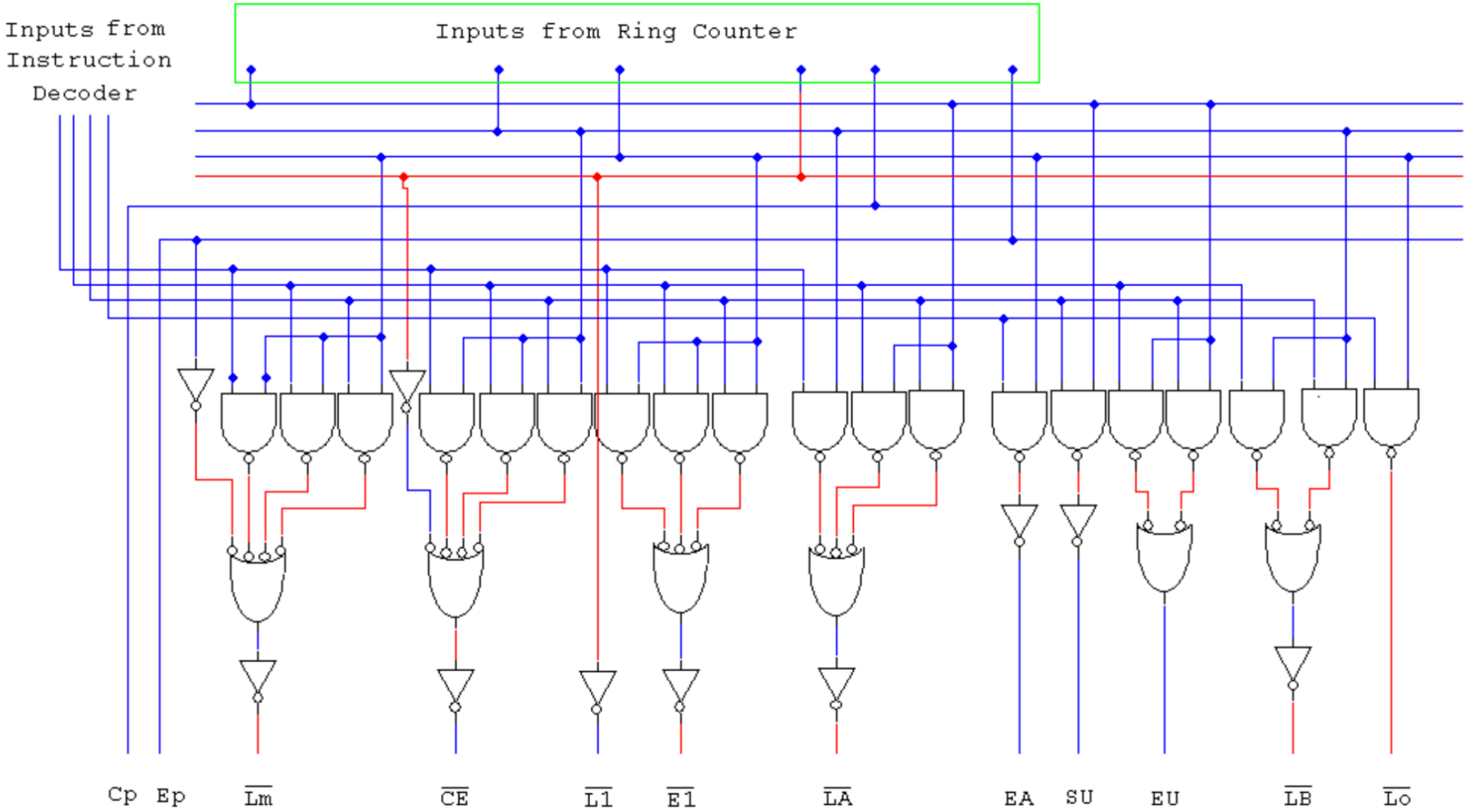
Diagrama de timing

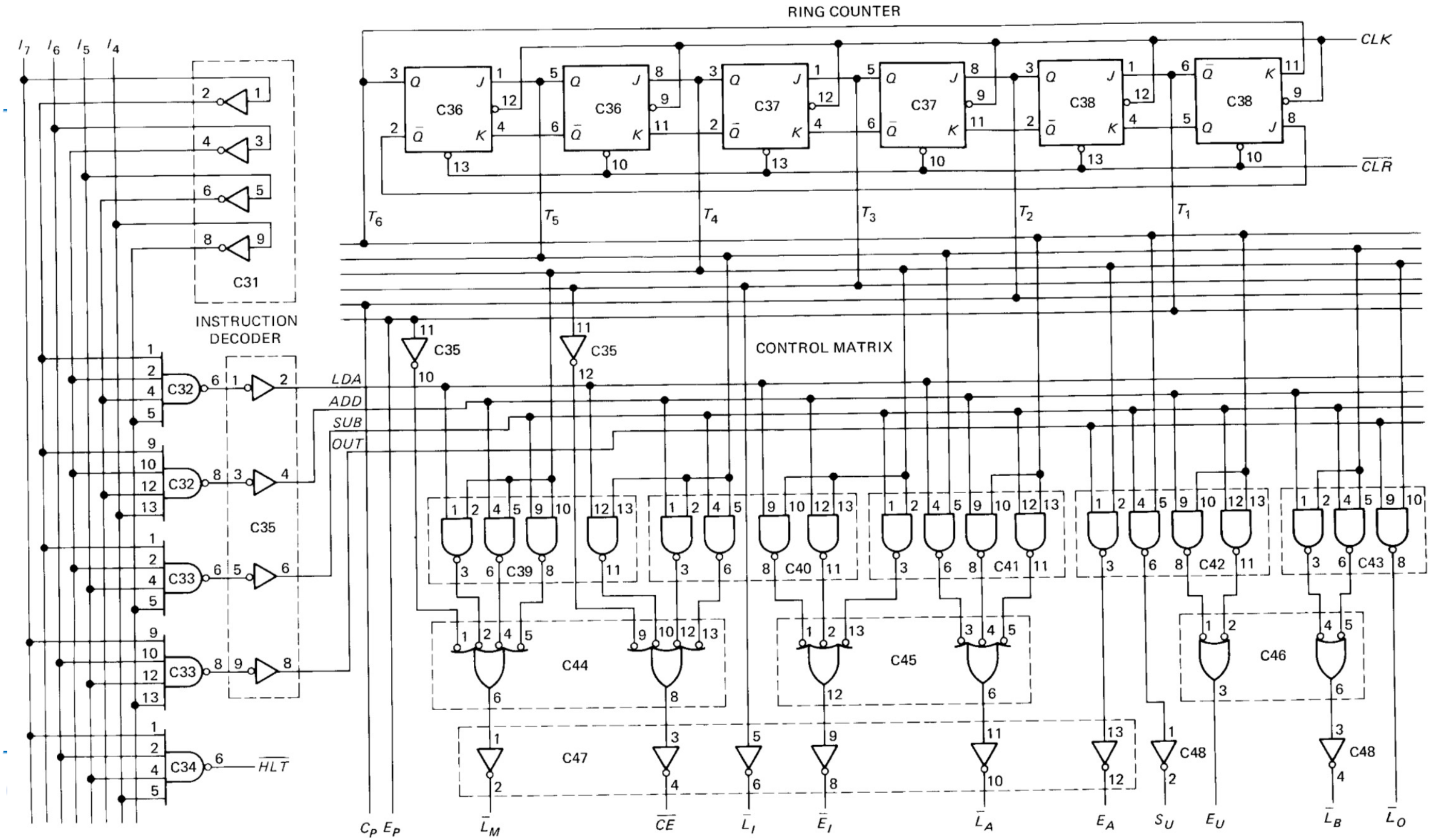


Instruction Decoder



Control Matrix




RING COUNTER
CLK
$\overline{CLR}$
C36
C37
C38
T_6
T_5
T_4
T_3
T_2
T_1
I_7 I_6 I_5 I_4
C31
INSTRUCTION DECODER
C32
C33
C34
C35
$\overline{HLT}$
LDA
ADD
SUB
OUT
CONTROL MATRIX
C39
C40
C41
C42
C43
C44
C45
C46
C47
C48
C_P E_P
$\overline{L}_M$
$\overline{CE}$
$\overline{L}_I$
$\overline{E}_I$
$\overline{L}_A$
E_A
S_U
E_U
$\overline{L}_B$
$\overline{L}_O$

Setul de Instrucțiuni

- Calculatorul este o bucată inutilă de hardware dacă nu este programat. Pentru SAP-1 (și nu numai), asta presupune încărcarea în memorie pas cu pas a tuturor instrucțiunilor înainte de rulare.
- Înainte de a începe acest pas, trebuie totuși să definim setul de instrucțiuni al procesorului nostru:

Mnemonica	Operație	Descriere
LDA	ACC <- RAM[MAR]	Încarcă în acumulator conținutul RAM la care pointează MAR
ADD	ACC <- ACC + B	Adună acumulatorul cu registrul B și scrie rezultatul în acumulator
SUB	ACC <- ACC - B	Scade registrul B din accumulator și scrie rezultatul în acumulator
OUT	OUT <- ACC	Scrie conținutul acumulatorului în registrul output (afișează rezultatul)
HLT	CLK <- 0	Oprire procesare

LDA – LoaD Accumulator

- O instrucțiune **LDA** completă include și adresa hexazecimală a locației din RAM de unde dorim să încărcăm datele
  - De exemplu **LDA 8H** înseamnă “încarcă acumulatorul cu octetul de la adresa 8H din memoria RAM”
  - Daca **RAM[8]** = 1111 0000
  - Execuția **LDA 8H** va rezulta în **ACC** = 1111 0000
  - Similar, **LDA FH** înseamnă “încarcă acumulatorul cu conținutul locației de memorie FH”

ADD

- **ADD 9H** înseamnă “adună datele din locația de memorie 9H cu datele din acumulator și salvează rezultatul în acumulator”
- De exemplu, presupunem că numărul 2 se află în acumulator și numărul 3 la adresa 9H în RAM. Atunci, **ACC** = 0000 0010 și **RAM[9]** = 0000 0011
- În timpul execuției **ADD** 9H:
  - Primul pas este ca datele din RAM de la adresa 9 să fie încărcate în registrul B, deci **B** = 0000 0011
  - Aproape instantaneu după încărcare, UAL formează operația A + B
  - Al doilea pas presupune încărcarea sumei în acumulator, pentru a obține **ACC** = 0000 0101
- Numerele negative sunt stocate în memorie folosind codul complement al lui 2.

SUB

- **SUB 9H** înseamnă “scade datele din locația de memorie 9H din datele din acumulator și salvează rezultatul în acumulator”
- De exemplu, presupunem că numărul 3 se află în acumulator și numărul 2 la adresa 9H în RAM. Atunci, **ACC** = 0000 0011 și **RAM[9]** = 0000 0010
- În timpul execuției **SUB** 9H:
  - Primul pas este ca datele din RAM de la adresa 9 să fie încărcate în registrul B, deci **B** = 0000 0010
  - Aproape instantaneu după încărcare, UAL formează operația A - B
  - Al doilea pas presupune încărcarea diferenței în acumulator, pentru a obține **ACC** = 0000 0001
- Numerele negative sunt stocate în memorie folosind codul complement al lui 2.

OUT

- Instrucțiunea **OUT** spune calculatorului SAP-1 să transfere acumulatorul în portul de ieșire.
- După ce **OUT** a fost executat, conținutul registrului poate fi vizualizat pe cele 8 LED-uri atașate
- **OUT** este completă in sine, nu are nevoie de niciun argument pentru că de fiecare dată va afișa doar conținutul acumulatorului.
- Dacă dorim să afișăm alte date, trebuie neapărat să le transferăm inițial în acumulator.

HLT - HaLT

- Instrucțiunea spune calculatorului să oprească procesarea prin dezactivarea semnalului de ceas.
- **HLT** marchează sfârșitul oricărui program SAP-1, așa cum punctul marchează sfârșitul oricărei propoziții
- Trebuie să folosiți instrucțiunea **HLT** la finalul oricărui program, altfel rezultatele rulării nu pot fi determinate
- **HLT** este completă și nu necesită niciun agument suplimentar

Coduri operație

- Pentru a încărca în memorie instrucțiunile SAP-1, trebuie să folosim un cod pe care calculatorul să îl poata interpreta ușor
- Sunt 5 operații, deci vom da fiecăreia un număr: 0000 pentru LDA, 0001 ADD, 0010 SUB, 1110 OUT și 1111 HLT
- Deoarece codul spune calculatorului ce operație să efectueze, le vom denumi **coduri operație** (operation codes, opcodes)

Mnemonică	Opcode
LDA	0000
ADD	0001
SUB	0010
OUT	1110
HLT	1111

- Limbajul de asamblare presupune lucrul cu mnemonici atunci când scriem un program
- Limbajul mașină presupune lucrul cu șiruri de 0 și 1

Program în Assembly		Program în Cod Mașină		
Adresă RAM	**Conținut**	**Adresă RAM**	**Conținut RAM (BIN)**	**Conținut RAM (HEX)**
0x0	LDA 9H	0000	0000 1001	0x09
0x1	ADD AH	0001	0001 1010	0x1A
0x2	ADD CH	0010	0001 1100	0x1C
0x3	SUB BH	0011	0010 1011	0x2B
0x4	OUT	0100	1110 1111	0xEF
0x5	HLT	0101	1111 1111	0xFF
0x6	0xFF	0110	1111 1111	0xFF
0x7	0xFF	0111	1111 1111	0xFF
0x8	0xFF	1000	1111 1111	0xFF
0x9	0x10	1001	0001 0000	0x10
0xA	0x18	1010	0001 1000	0x18
0xB	0x14	1011	0001 0100	0x14
0xC	0x20	1100	0010 0000	0x20
0xD	0xFF	1101	1111 1111	0xFF
0xE	0xFF	1110	1111 1111	0xFF
0xF	0xFF	1111	1111 1111	0xFF

Acknowledgements

- Aceste slide-uri conțin materiale aparținând:
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)
  - Behrooz Parhami (UCSB)

- MIT material derived from course 6.823
- UCB material derived from course CS252