Bài giảng Kỹ thuật Vi xử lý

Ngành Điện tử-Viễn thông
Đại học Bách khoa Đà Nẵng
của Hồ Viết Việt, Khoa CNTT-ĐTVT

Tài liệu tham khảo

[1] Kỹ thuật vi xử lý, Văn Thế Minh, NXB Giáo dục, 1997

[2] Kỹ thuật vi xử lý và Lập trình Assembly cho hệ vi xử lý, Đỗ Xuân Tiến, NXB Khoa học & kỹ thuật, 2001

Chương 1
Các hệ thống số, mã hoá, linh kiện số cơ bản

1.1 Các hệ thống số

- Hệ thập phân
- Hệ nhị phân
- Hệ thập lục phân

1.2 Các hệ thống mã hoá

- ASCII
- BCD

1.3 Các linh kiện điện tử số cơ bản

- Các cổng logic: AND, OR, XOR,NOT
- Các bộ giải mã, Các IC chốt, đệm

1.1 Các hệ thống số

- Hệ đếm thập phân (Decimal)
- Còn gọi là hệ đếm cơ số mười
  (Vì có quá ít người có chín ngón tay hoặc mười một ngón chân?)
- Dùng mười ký hiệu:
  1,2,3,4,5,6,7,8,9,0
- Ví dụ:1.1:
  Ba nghìn Chín trăm Bảy mươi Tám
  $3978 = 3x10^3 + 9x10^2 + 7x10^1 + 8x10^0$
  $= 3000 + 900 + 70 + 8$

1.1 Các hệ thống số

- Hệ đếm nhị phân (Binary)
- Còn gọi là Hệ đếm cơ số hai
- Sử dụng hai ký hiệu (bit): 0 và 1
  (Các hệ thống điện tử số chỉ sử dụng hai mức điện áp?)
- Kích cỡ, LSB, MSB của số nhị phân
- Số nhị phân không dấu (Unsigned)
- Số nhị phân có dấu (Số bù hai)

Số nhị phân

- Mỗi ký hiệu 0 hoặc 1 được gọi là 1 Bit (Binary Digit- Chữ số nhị phân)
- Kích cỡ của một số nhị phân là số bit của nó
- MSB (Most Significant Bit): Bit sát trái
- LSB (Least Significant Bit): Bit sát phải
- Ví dụ 1.1: 1010101010101010

là một số nhị phân 16-bit

Số nhị phân không dấu

- Chỉ biểu diễn được các giá trị không âm (>= 0)
- Với n-bit có thể biểu diễn các giá trị từ 0 đến $2^n - 1$
- Ví dụ 1.3: Giá trị V của số nhị phân không dấu 1101 được tính:

$$V(1101) = 1x2^3 + 1x2^2 + 0x2^1 + 1x2^0$$

$$= 8 + 4 + 0 + 1 = 13$$

Số nhị phân không dấu

- Tổng quát: Nếu số nhị phân N n-bit:

$$N = b_{(n-1)}\, b_{(n-2)} \ldots b_1\, b_0$$

thì giá trị V của nó là:

$$V = b_{(n-1)} \times 2^{(n-1)} + b_{(n-2)} \times 2^{(n-2)} + \ldots + b_1 \times 2^1 + b_0 \times 2^0$$

Các số nhị phân không dấu 4-bit biểu diễn được các giá trị từ ? đến ?

16 giá trị từ 0 đến 15

Nhị phân không dấu	Giá trị thập phân
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

Số nhị phân không dấu

- Dải giá tri của các số không dấu 8-bit là [0,255] *(unsigned char trong C)*
- Dải giá tri của các số không dấu 16-bit là [0,65535] *(unsigned int trong C)*

Chuyển đổi thập phân sang nhị phân

- Ví dụ 1.4

  Chuyển 25 sang nhị phân không dấu. Dùng phương pháp chia 2 liên tiếp

Chia 2		Thương số	Dư số	
25/2	=	12	1	LSB
12/2	=	6	0	
6/2	=	3	0	
3/2	=	1	1	
1/2	=	0	1	MSB

  Kết quả là: 11001

Số nhị phân có dấu

- Biểu diễn được cả các giá trị âm
- Còn gọi là Số bù hai
- Với n-bit có thể biểu diễn các giá trị từ $-2^{(n-1)}$ đến $2^{(n-1)} - 1$
- Ví dụ 1.3: Giá trị V của số nhị phân có dấu 1101 được tính:

$$V(1101) = -1x2^3 + 1x2^2 + 0x2^1 + 1x2^0$$

$$= -8 + 4 + 0 + 1 = -3$$

Số nhị phân có dấu

- Tổng quát: Nếu số nhị phân N n-bit:

$$N = b_{(n-1)}\, b_{(n-2)} \ldots b_1\, b_0$$

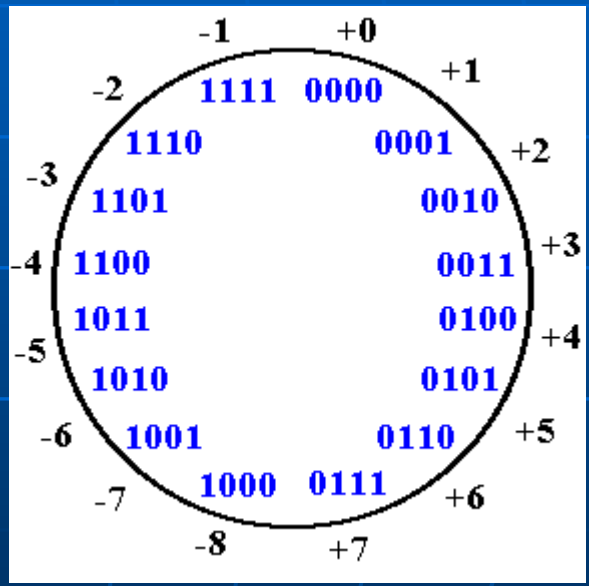
thì giá trị V của nó là:

$$V = -b_{(n-1)} \times 2^{(n-1)} + b_{(n-2)} \times 2^{(n-2)} + \ldots + b_1 \times 2^1 + b_0 \times 2^0$$

Các số nhị phân có dấu 4-bit biểu diễn được các giá trị từ ? đến ?

16 giá trị từ - 8 đến 7

Nhị phân có dấu	Giá trị thập phân
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	- 8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1



Số nhị phân có dấu

- Dải giá tri của các số có dấu 8-bit là [-128,+127] *(char trong C)*
- Dải giá tri của các số có dấu 16-bit là [-32768,+32767] *(int trong C)*

Tìm đối số (Lấy bù 2)

- Tổng của một số với đối số của nó bằng 0
- Ví dụ 1.5

Đối số của số nhị phân có dấu 10011101?

```
          10011101    Số có dấu (-99)
          01100010    Lấy bù 1
      +          1    Cộng 1
        ------------
          01100011    Kết quả   (+99)
```

Chuyển số thập phân sang nhị phân có dấu

- Với số dương:Giống như chuyển thập phân sang nhị phân không dấu rồi thêm bit 0 vào sát bên trái
- Ví dụ: Chuyển 25 sang nhị phân có dấu:
  Kết quả: 011011
- Với số âm: Chuyển đối số sang nhị phân có dấu rồi lấy bù 2

Chuyển số thập phân sang nhị phân có dấu

Ví dụ 1.6 Chuyển – 26 sang nhị phân

1. chuyển đổi số: +26 = 11010
2. Đưa 0 vào sát trái: 011010
3. Bù 1: 100101
4. Cộng 1: + 1

-26 = 100110

Số thập lục phân

- Quen gọi là số Hexa (Hexadecimal)
- Còn gọi là hệ đếm cơ số mười sáu
- Sử dụng 16 ký hiệu để biểu diễn: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
- Mỗi ký hiệu tương ứng với 4-bit
- Mục đích: Biểu diễn số nhị phân ở dạng ngắn gọn

11110000	=	F0
10101010	=	AA
01010101	=	55
Nhị phân		Thập lục phân

Mỗi ký hiệu tương ứng với 4-bit

Hexa	Binary	Hexa	Binary
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

Chuyển đổi Hexa & nhị phân

- Ví dụ 1.7

Chuyển số hexa 2F8 và ABBA sang nhị phân

Thay thế mỗi ký hiệu hexa bằng 4-bit tương ứng với nó

2	F	8
0010	1111	1000

A	B	B	A
1010	1011	1011	1010

- Kết quả 2F8h = 001011111000b

ABBAh = 1010101110111010b

Chuyển đổi Hexa & nhị phân

- Ví dụ 1.8

  Chuyển số nhị phân 1100101011111110 sang hexa

  - Trước hết theo hướng từ LSB về MSB chia số nhị phân đó thành các nhóm 4-bit
  - Sau đó thay thế mỗi nhóm 4-bit bằng ký hiệu hexa tương ứng với nó

1100	1010	1111	1110
C	A	F	E

- Kết quả: 1100101011111110b = CAFEh

1.2 Các hệ thống mã hoá

- ASCII: American Standard Code for Information Interchange.
- Dùng để biểu diễn các ký tự (characters):
  Gồm ký tự hiển thị được và ký tự điều khiển
- Mỗi ký tự được biểu diễn bằng 8-bit gọi là mã ASCII của ký tự đó
  - Các chữ cái in và thường: A..Z và a..z
  - Các chữ số thập phân: 0,1,…,9
  - Các dấu chấm câu: ; , . : *vân vân*
  - Các ký tự đặc biệt: **$ & @ / {** *vân vân*
  - Các ký tự điều khiển: carriage return (CR) , line feed (LF), beep, *vân vân*

Mã ASCII

- Với bảng mã được sắp xếp theo trật tự tăng dần của mã ASCII:
  - Các chữ số thập phân: 0,1,…,9 nằm liên tiếp nhau, chữ số 0 có mã ASCII là 30h
  - Các chữ cái in:A..Z nằm liên tiếp nhau, chữ A có mã ASCII là 41h
  - *Các chữ cái thường: a..z nằm liên tiếp nhau, chữ a có mã ASCII là 61h*
  - *Mã ASCII của chữ in và chữ thường tương ứng chỉ khác nhau ở bit 5*
    A: 01**0**00001 B: 01**0**00010 Z: 01**0**11010
    a: 01**1**00001 b: 01**1**00010 z: 01**1**11010
  - 32 ký tự điều khiển được xếp đầu bảng mã (00h đến 1Fh)

Bảng mã ASCII

Decimal	Hex	ASCII	Decimal	Hex	ASCII	Decimal	Hex	ASCII	Decimal	Hex	ASCII
0	0	NUL	32	20		64	40	@	96	60	`
1	1	SOH	33	21	!	65	41	A	97	61	a
2	2	STX	34	22	"	66	42	B	98	62	b
3	3	ETX	35	23	#	67	43	C	99	63	c
4	4	EOT	36	24	$	68	44	D	100	64	d
5	5	ENQ	37	25	%	69	45	E	101	65	e
6	6	ACK	38	26	&	70	46	F	102	66	f
7	7	BEL	39	27	'	71	47	G	103	67	g
8	8	BS	40	28	(	72	48	H	104	68	h
9	9	HT	41	29	)	73	49	I	105	69	i
10	A	LF	42	2A	*	74	4A	J	106	6A	j
11	B	VT	43	2B	+	75	4B	K	107	6B	k
12	C	FF	44	2C	,	76	4C	L	108	6C	l
13	D	CR	45	2D	-	77	4D	M	109	6D	m
14	E	SOH	46	2E	.	78	4E	N	110	6E	n
15	F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	□

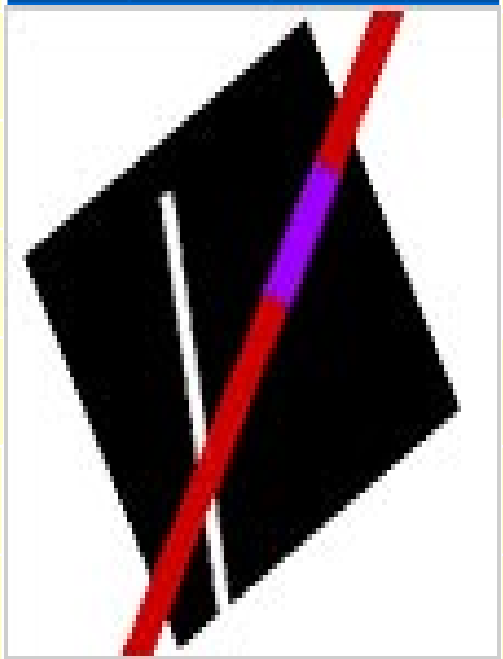
Bảng mã ASCII

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	ø
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB		251	FB	√
156	9C	£	188	BC	╝	220	DC		252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD		253	FD	²
158	9E	₧	190	BE	╛	222	DE		254	FE	■
159	9F	ƒ	191	BF	┐	223	DF		255	FF	

Mã BCD

- BCD (Binary Coded Decimal)
- Quen gọi là số BCD
- Dùng để mã hoá các số thập phân bằng các ký hiệu nhị phân
- Mỗi chữ số thập phân được biểu diễn bằng một tổ hợp 4-bit
- Các tổ hợp 4-bit không sử dụng gọi là các tổ hợp cấm
- Nhiều linh kiện điện tử sử dụng mã này (Bộ giải mã BCD-LED bảy đoạn 7447)

Bảng mã BCD

Thập phân	BCD	Thập phân	BCD
0	0000	8	1000
1	0001	9	1001
2	0010		1010
3	0011		1011
4	0100		1100
5	0101		1101
6	0110		1110
7	0111		1111

Mã BCD

- Đừng nhầm mã hoá BCD với việc chuyển đổi thập phân sang nhị phân:

  Ví dụ 1.9: Cho số thập phân 15

  Mã BCD của nó là: 00010101

  Số nhị phân không dấu
  8-bit tương ứng là: 00001111

Bit, Nibble, Byte, Word

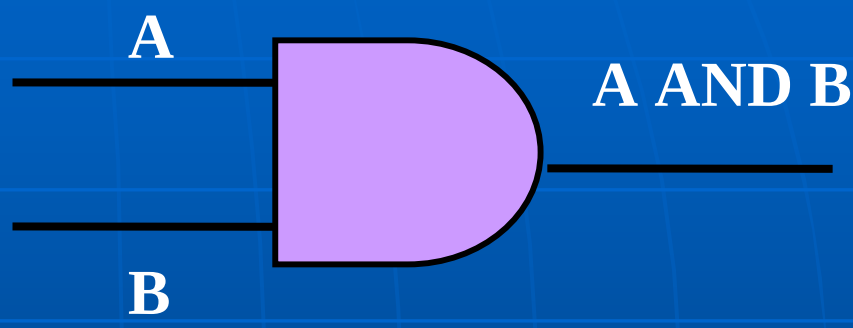
- Bit: Một chữ số nhị phân 0 hoặc 1
- Nibble: 4-bit (nửa byte)
- Byte: 8-bit (Còn gọi là Octet)
- Word (Từ): 16-bit
- Double Word (Từ kép): 32-bit
- K = 2^{10} = 1024
  Kb (kilôbit) = 1024 bit = 128 byte
  KB (kilôbyte) = 1024 byte
  Kbps (Kilobit per second): Kilôbit trên giây
- M = 2^{20} = 1024 K = 1048576
  Mb (Mêgabit) = 1024 Kb = 1048576 bit
  MB (Mêgabyte) = 1024 KB = 1048576 byte
- G = 2^{30} = 1024 M = 1048576 K
  Gb (Gigabit) = 1024 Mb = 1048576 Kb
  GB (Gigabyte) = 1024 MB = 1048576 KB
- T = ?

1.3 Các linh kiện điện tử số cơ bản

- Phân chia linh liện số theo mật độ tích hợp: SSI, MSI, LSI, VLSI
  SSI (Small Scale Integration): Vi mạch tích hợp cỡ nhỏ
  MSI (Medium Scale Integration): Vi mạch tích hợp cỡ trung
  LSI (Large Scale Integration): Vi mạch tích hợp cỡ lớn
  VLSI (Very Large Scale Integration):Vi mạch tích hợp cỡ cực lớn
- SSI: Các cổng logic and, or, xor, not
- MSI: Các bộ giải mã, Các chốt, đệm
- LSI,VLSI: Các bộ vi xử lý, vi điều khiển, DSPs

Cổng logic AND

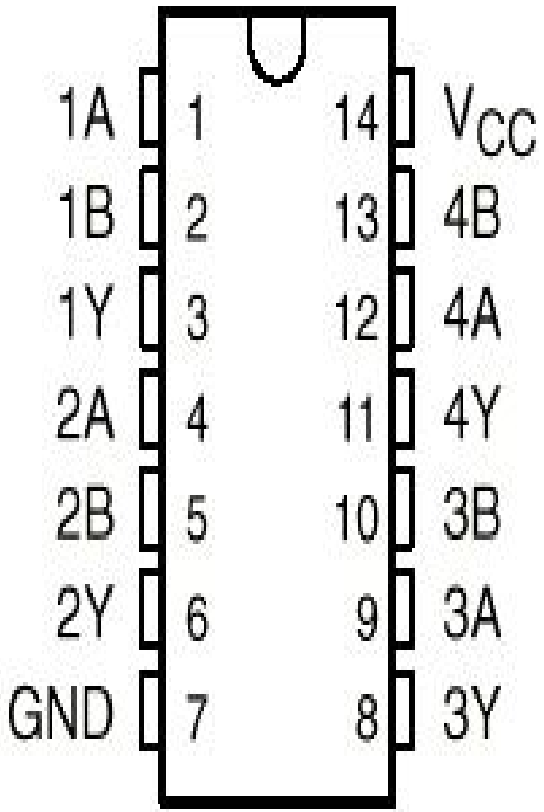
A	B	A AND B
1	1	1
1	0	0
0	1	0
0	0	0



Cổng AND có thể có nhiều hơn 2 đầu vào

Trên một chip có thể có nhiều cổng AND

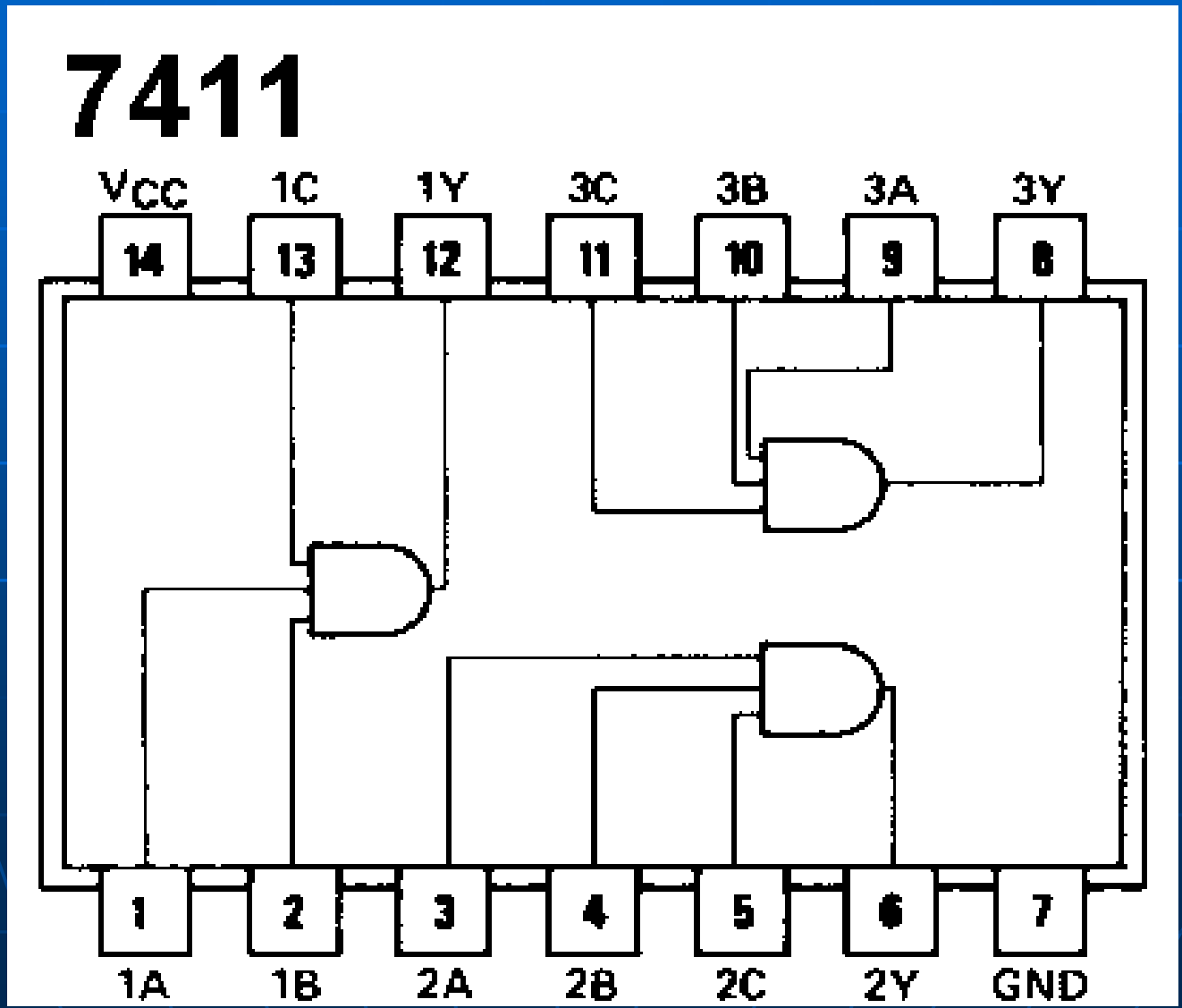
Cổng logic AND: IC 7408



FUNCTION TABLE
(each gate)

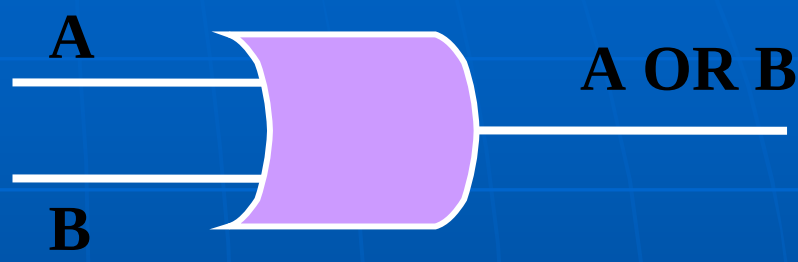
INPUTS		OUTPUT
A	B	Y
H	H	H
L	X	L
X	L	L

Cổng logic AND: IC 7411



Cổng logic OR

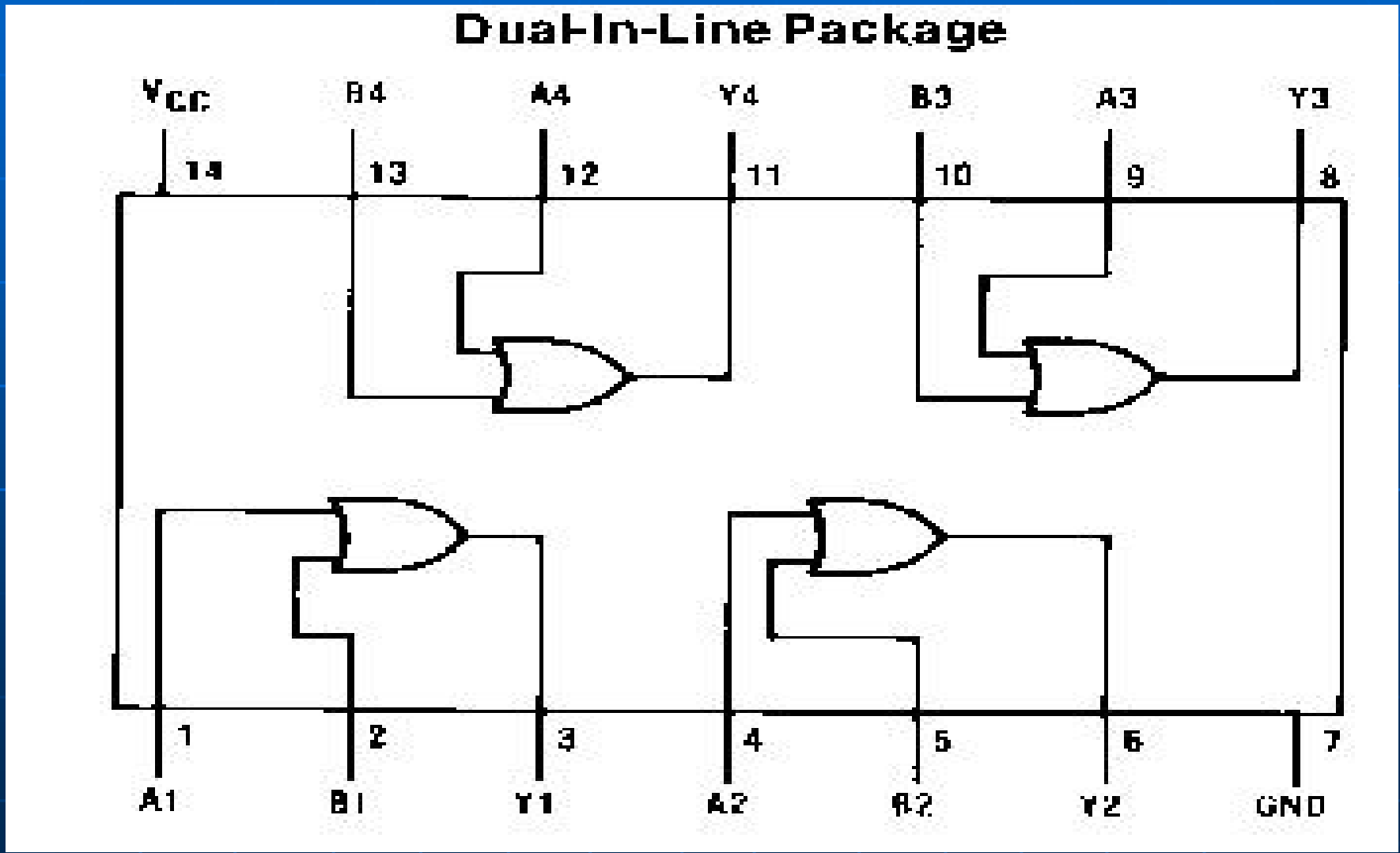
A	B	A OR B
1	1	1
1	0	1
0	1	1
0	0	0



Cổng OR có thể có nhiều hơn 2 đầu vào

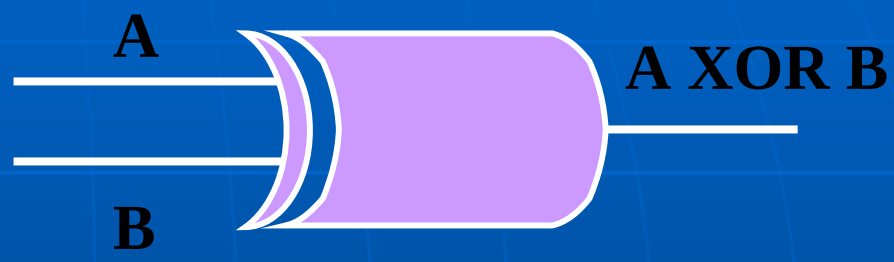
Trên một chip có thể có nhiều cổng OR

Cổng logic OR: IC 7432



Cổng logic XOR

A	B	A XOR B
1	1	0
1	0	1
0	1	1
0	0	0

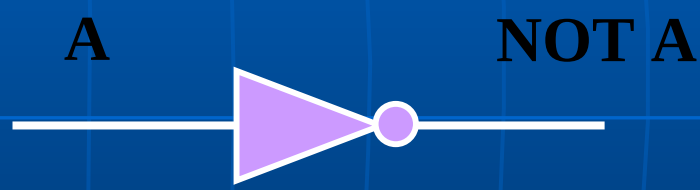


Cổng XOR có thể có nhiều hơn 2 đầu vào

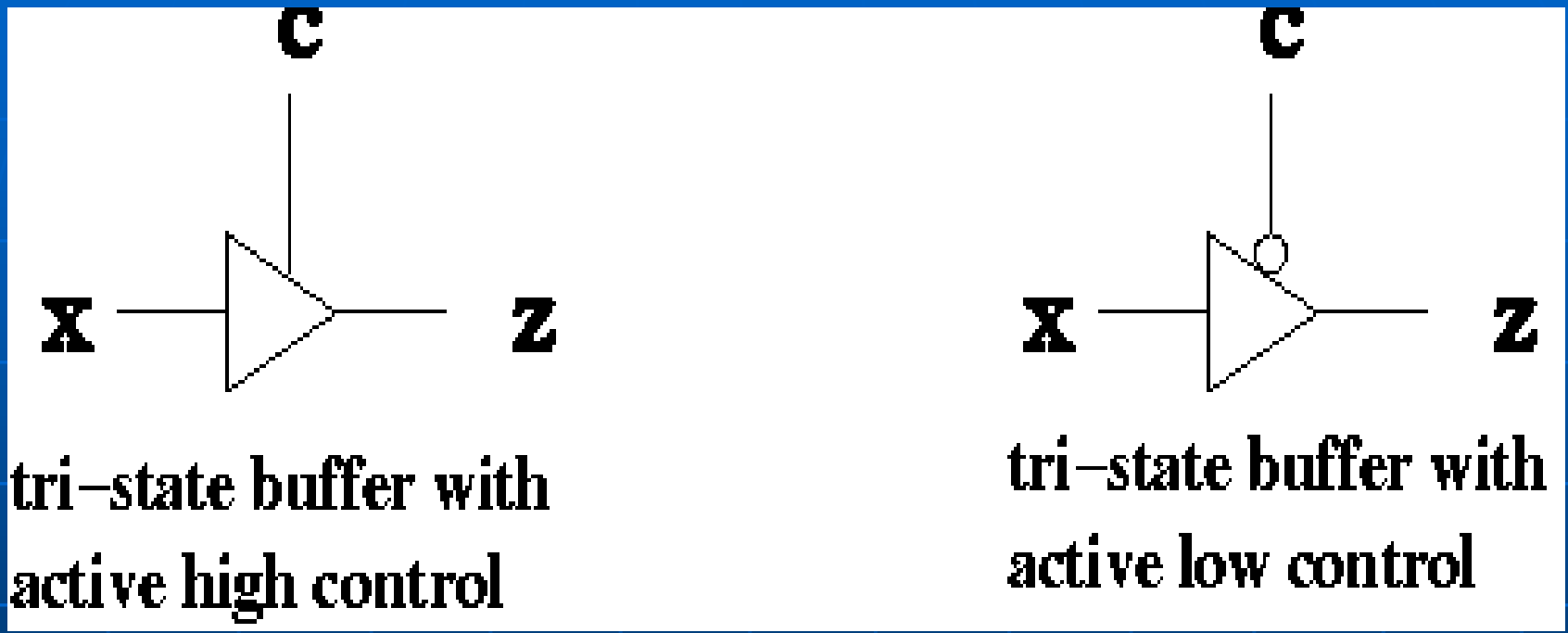
Trên một chip có thể có nhiều cổng XOR

Cổng logic NOT

A	NOT A
1	0
1	0
0	1
0	1



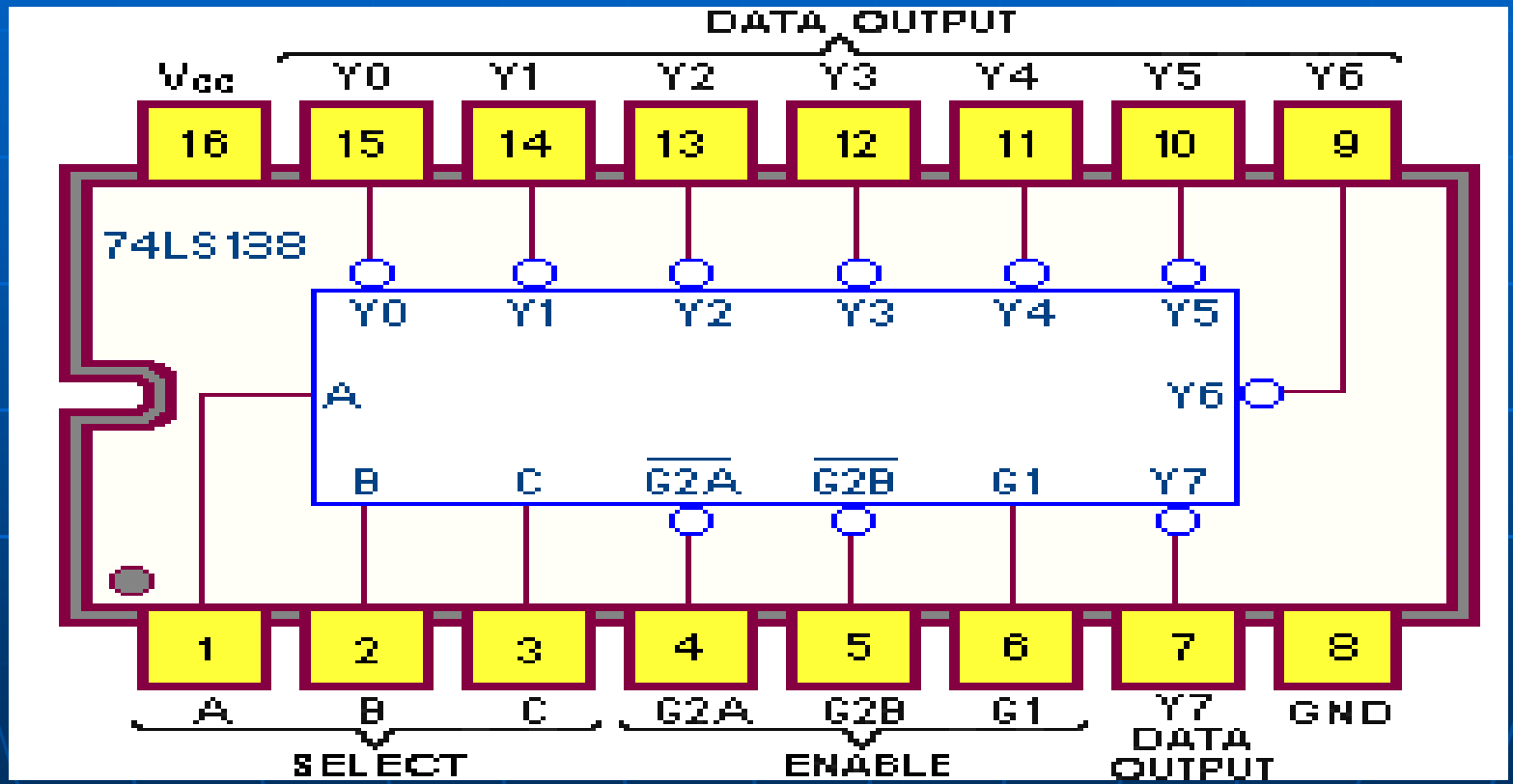
Đệm 3 trạng thái



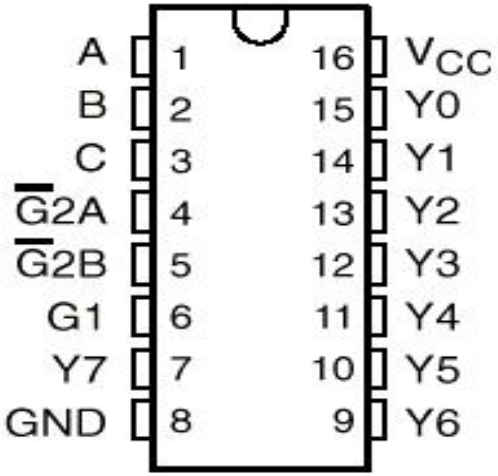
c	z
0	HiZ
1	x

c	z
1	HiZ
0	x

Chip giải mã 74138



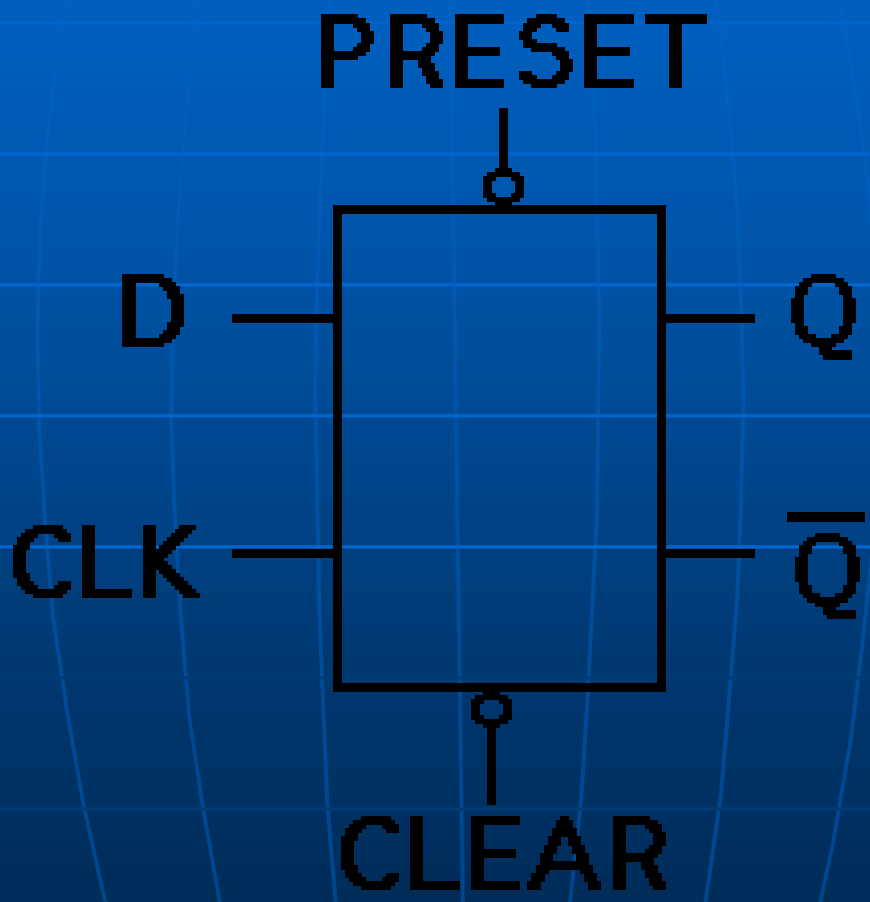
Chip giải mã 74138



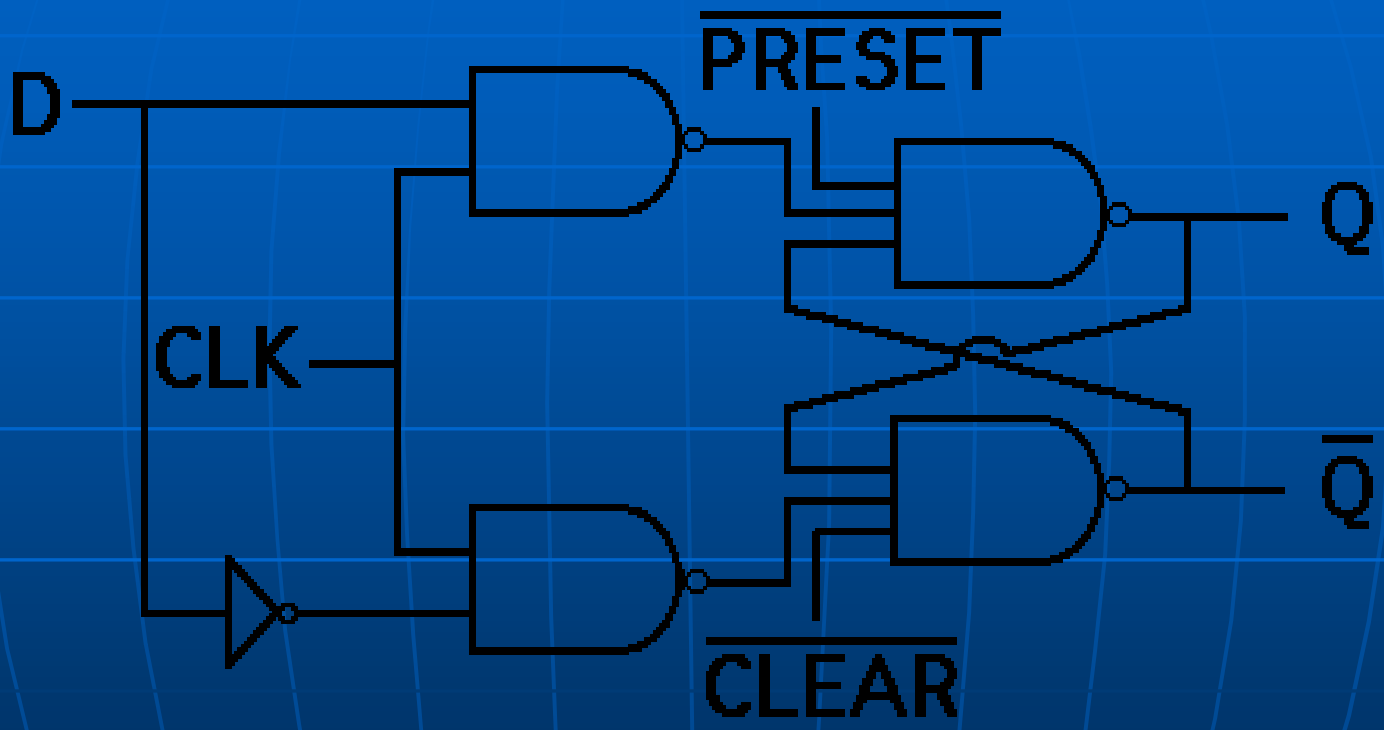
FUNCTION TABLE

ENABLE INPUTS			SELECT INPUTS			OUTPUTS							
G1	$\overline{G2A}$	$\overline{G2B}$	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	X	H	H	H	H	H	H	H	H
X	X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	L	H	H	H	H	H	L	H	H	H	H
H	L	L	H	L	L	H	H	H	H	L	H	H	H
H	L	L	H	L	H	H	H	H	H	H	L	H	H
H	L	L	H	H	L	H	H	H	H	H	H	L	H
H	L	L	H	H	H	H	H	H	H	H	H	H	L

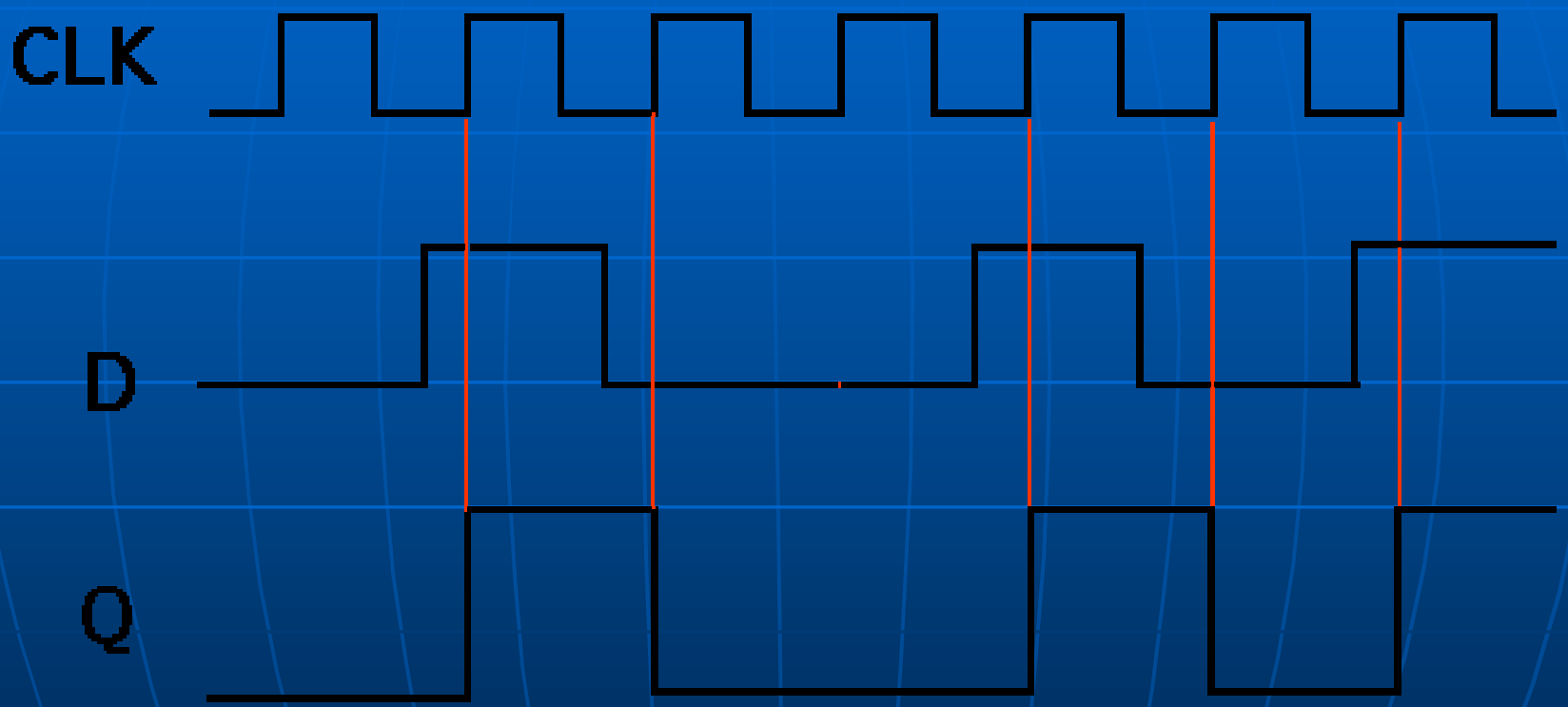
Flip Flop kiểu D



Flip Flop kiểu D



Flip Flop kiểu D

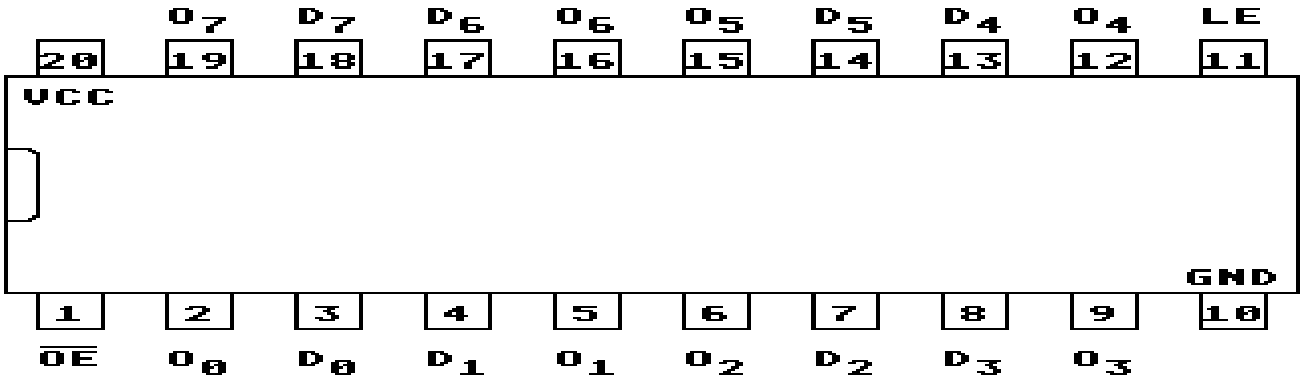


Flip Flop kiểu D

Clear	Preset	Clock	Q	$\bar{Q}$
0	0	x	undefind	
0	1	x	0	1
1	0	x	1	0
1	1	↑	D	$\bar{D}$

Chốt 8-bit 74373

74373 Octal Transparent Latch

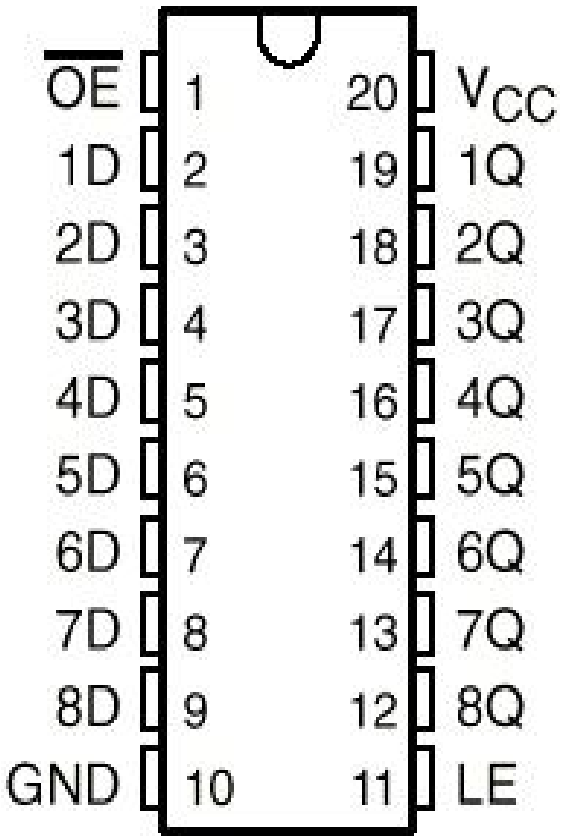


3-State Outputs

Inputs			Outputs
$\overline{OE}$	LE	D	O_n
L	H	H	H
L	H	L	L
L	L	X	O_0
H	X	X	Z

Z : High Impedance
X : Immaterial
O_0: Previous O_0 before L-to-H Transition of Clock

Chốt 8-bit 74573

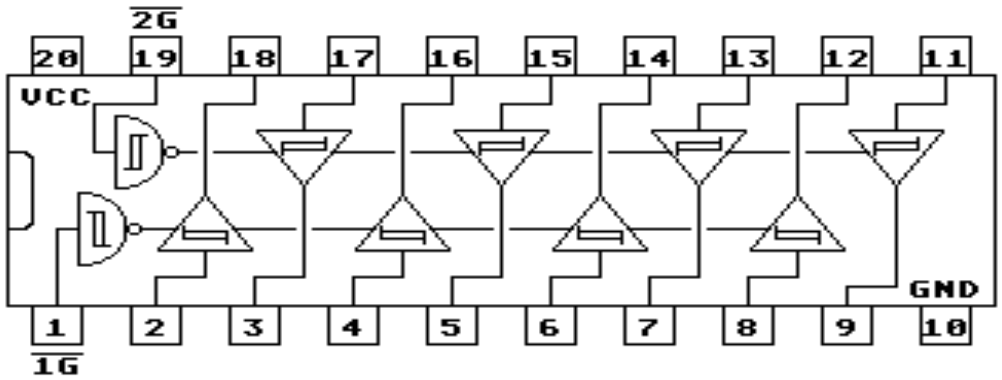


FUNCTION TABLE
(each latch)

INPUTS			OUTPUT
$\overline{OE}$	LE	D	Q
L	H	H	H
L	H	L	L
L	L	X	Q_0
H	X	X	Z

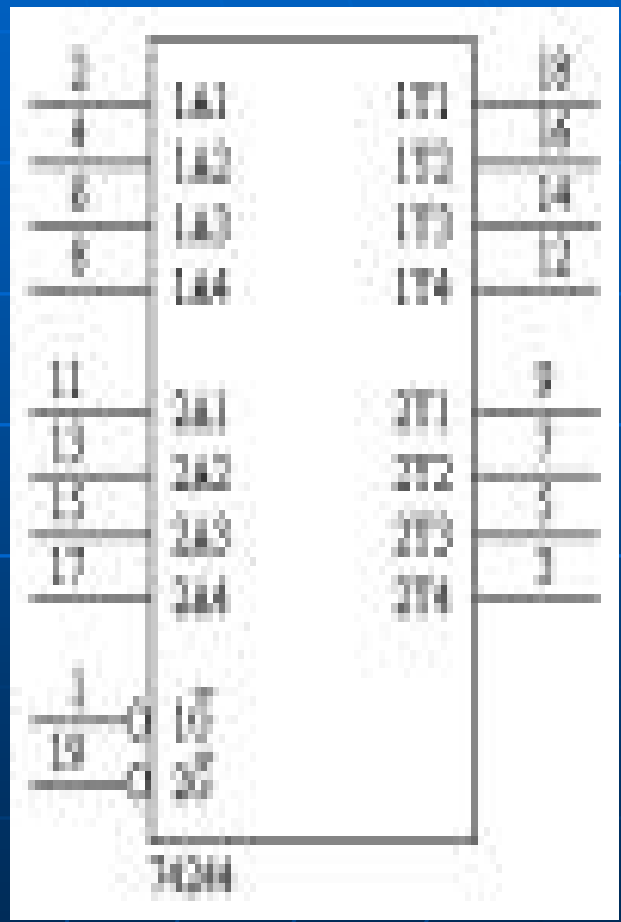
IC 74244

74244 Octal Buffer/Line Driver



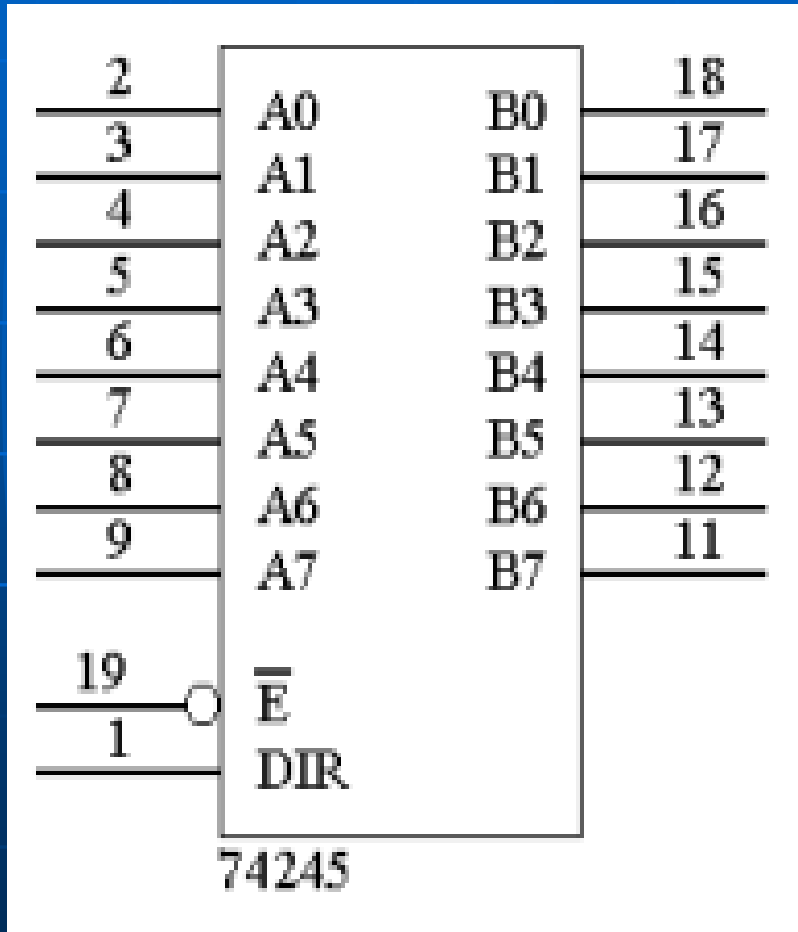
3-State Outputs

Truth Table		
$\overline{1G}$, $\overline{2G}$	D	Output
L	L	L
L	H	H
H	X	High Impedance



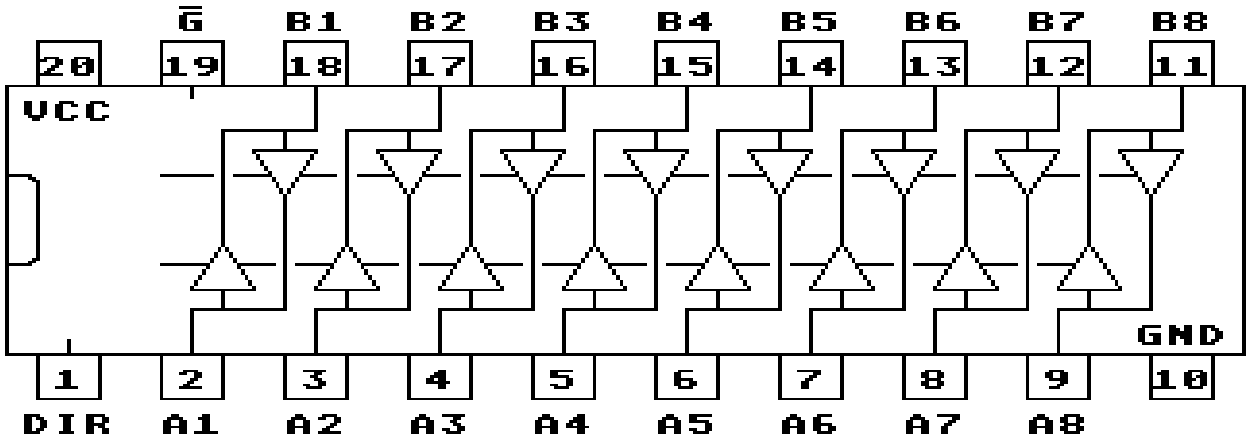
IC 74244

Đệm 2 chiều 74245



Đệm 2 chiều 74245

74245 Octal Bus Transceiver



Truth Table	
$\overline{G}$ DIR	Output
L L	Bus B Data to Bus A
L H	Bus A Data to Bus B
H X	Isolation

Bài giảng Kỹ thuật Vi xử lý

Ngành Điện tử-Viễn thông
Đại học Bách khoa Đà Nẵng
của Hồ Viết Việt, Khoa CNTT-ĐTVT

Tài liệu tham khảo

[1] Kỹ thuật vi xử lý, Văn Thế Minh, NXB Giáo dục, 1997

[2] Kỹ thuật vi xử lý và Lập trình Assembly cho hệ vi xử lý, Đỗ Xuân Tiến, NXB Khoa học & kỹ thuật, 2001

Chương 2

Vi xử lý và Hệ thống vi xử lý

2.1 Bộ vi xử lý

- Bộ vi xử lý (Microprocessor) là gì?
- Các thành phần của bộ vi xử lý
- Ứng dụng của bộ vi xử lý

2.2 Các họ vi xử lý

- Họ x86 của Intel- Luật Moore
- Họ 68x của Motorola

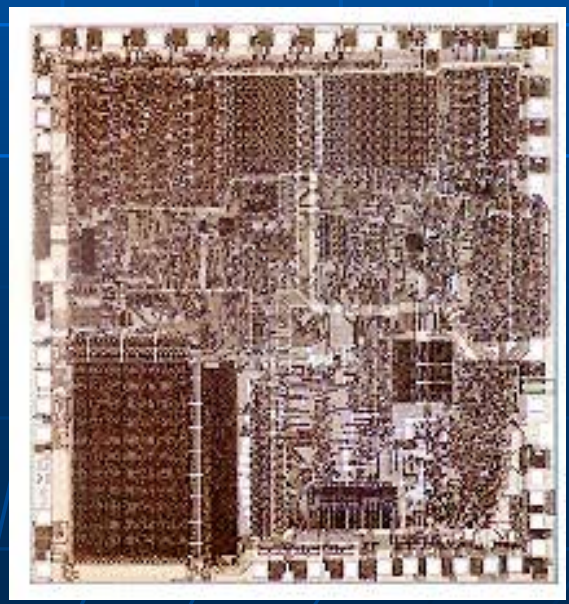
2.3 Hệ thống vi xử lý

- Bộ nhớ
- Các cổng I/O
- Bus hệ thống: D-Bus, A-Bus, C-Bus
- Thiết kế hệ thống vi xử lý?

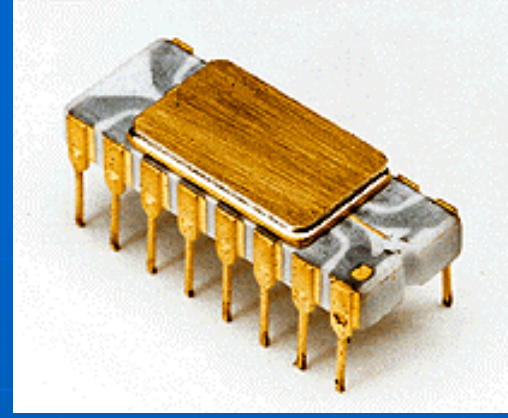
2.1 Bộ vi xử lý

- Một bộ vi xử lý là một mạch tích hợp chứa hàng ngàn, thậm chí hàng triệu transistor (LSI, VLSI) được kết nối với nhau
- Các transistor ấy cùng nhau làm việc để lưu trữ và xử lý dữ liệu cho phép bộ vi xử lý có thể thực hiện rất nhiều chức năng hữu ích
- Chức năng cụ thể của một bộ vi xử lý được xác định bằng phần mềm (có thể lập trình được)

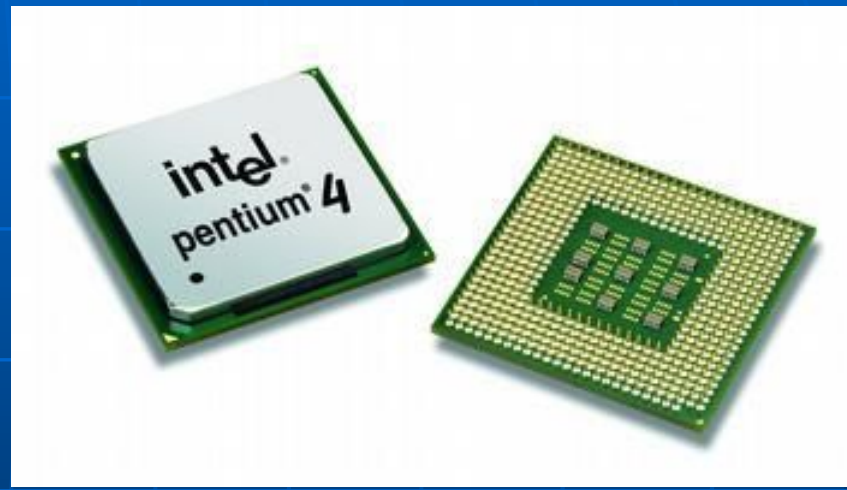
Intel® 8088 Chip



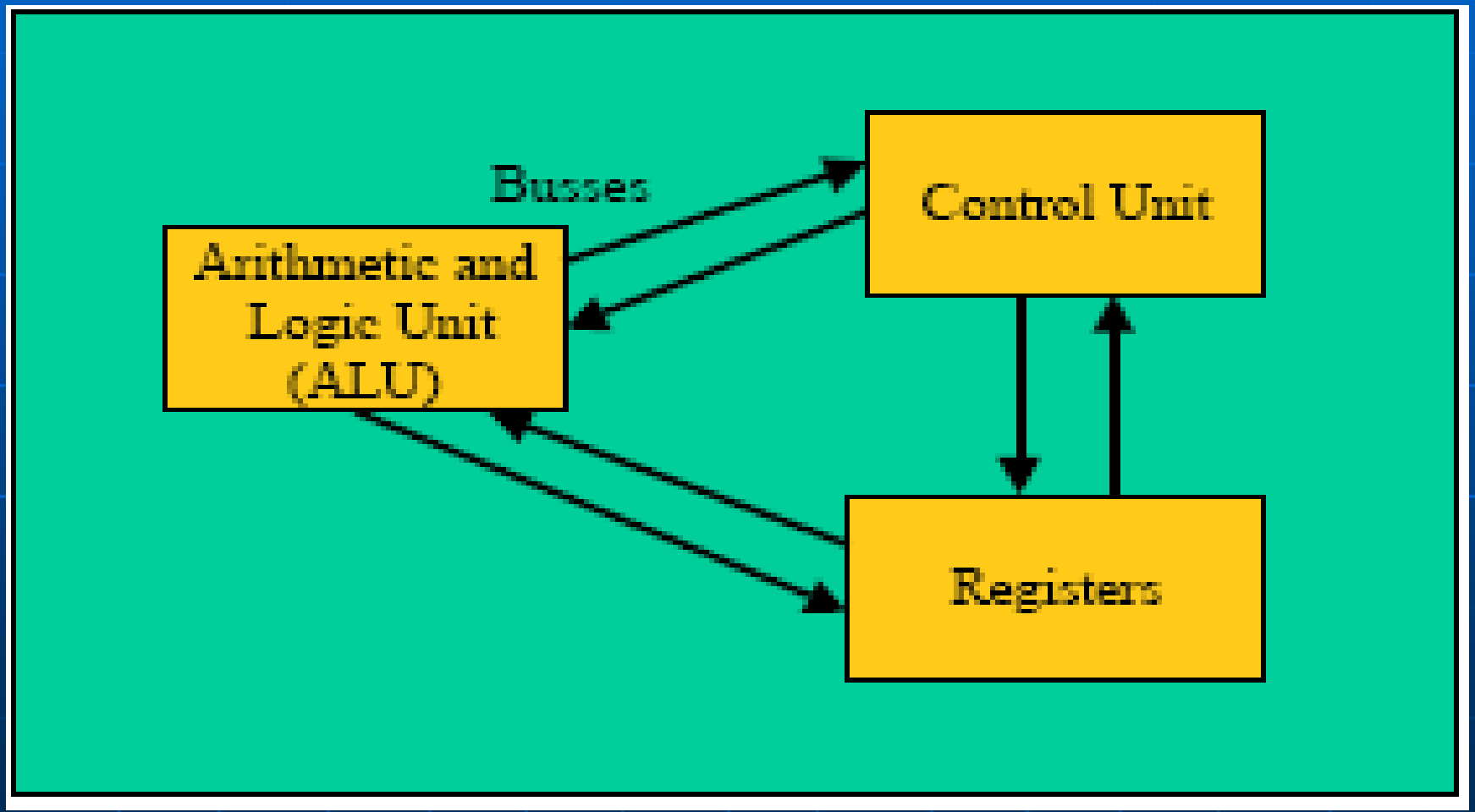
Bộ vi xử lý



- Bộ vi xử lý đầu tiên của Intel,4004, được giới thiệu vào năm 1971.
- 4004 chứa 2300 transistor.
- Bộ vi xử lý Pentium 4 hiện nay chứa 55 triệu transistor.
- Bộ vi xử lý thường được sử dụng trong các máy vi tính (microcomputer) với vai trò là CPU. Ngoài ra, chúng còn có mặt ở nhiều thiết bị khác.



Các thành phần của bộ vi xử lý



ALU và Control Unit

ALU

- Thực hiện các phép toán logic (AND, OR, XOR, NOT) và các phép toán số học (cộng, trừ, nhân, chia)
- Thực hiện việc chuyển dữ liệu
- Việc thực hiện lệnh thực sự diễn ra ở ALU

Control Unit

- Có trách nhiệm liên quan đến việc tìm và thực hiện các lệnh bằng cách cung cấp các tín hiệu điều khiển và định thời cho ALU và các mạch khác biết phải làm gì và làm khi nào.

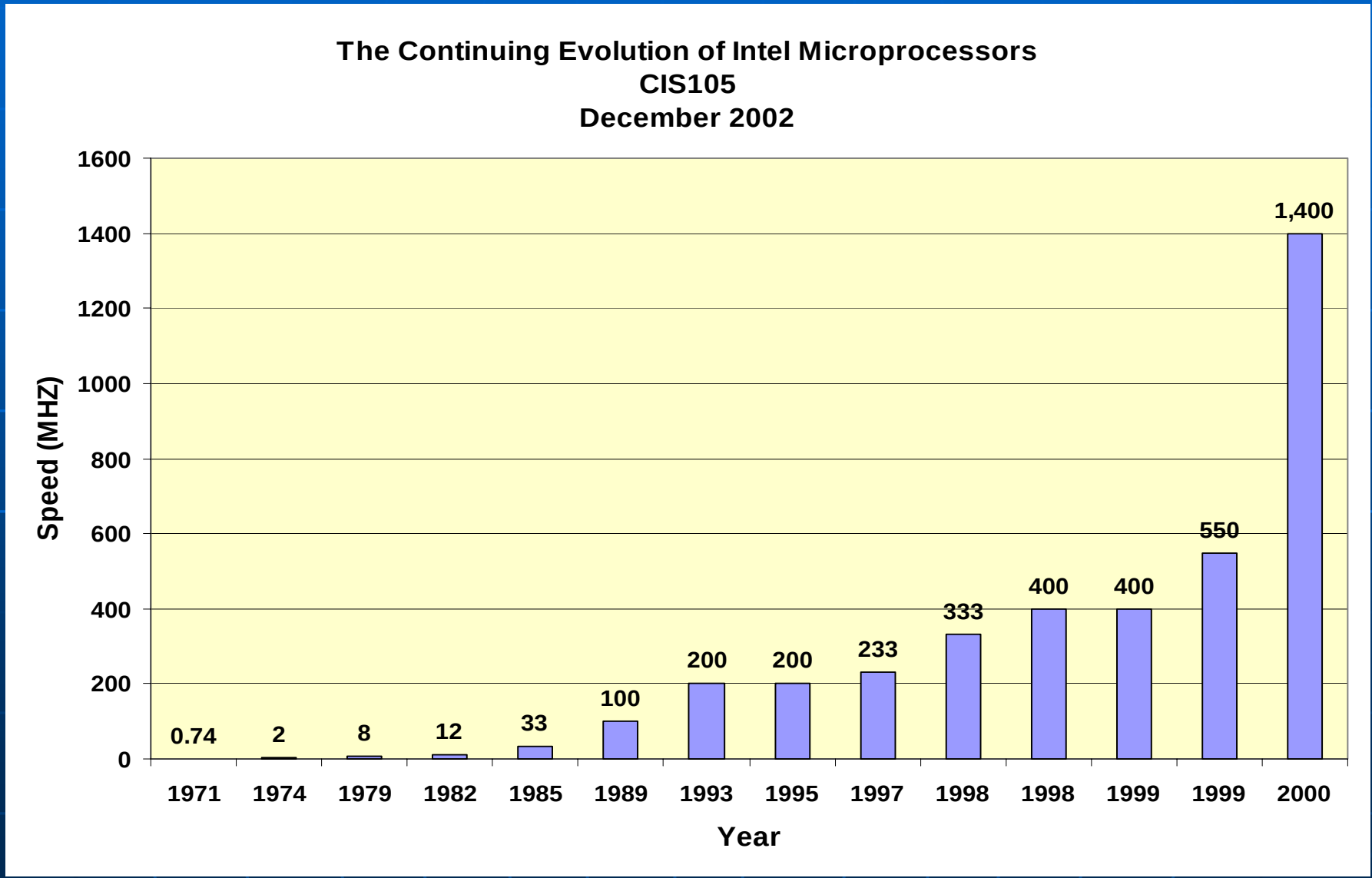
Các thanh ghi (Registers)

- Thanh ghi là nơi mà bộ vi xử lý có thể lưu trữ được một số nhị phân (Kích cỡ của thanh ghi tính bằng bit)
- Bộ vi xử lý dùng các thanh ghi để lưu trữ dữ liệu tạm thời trong quá trình thực hiện chương trình
- Các thanh ghi có thể được truy cập bằng các câu lệnh ngôn ngữ máy thường được gọi là các thanh ghi người sử dụng có thể nhìn thấy được (có thể truy cập được)
- Các thanh ghi điều khiển và các thanh ghi trạng thái được CU dùng để điều khiển việc thực hiện chương trình. Đa số các thanh ghi này người sử dụng không thể nhìn thấy được

2.2 Các họ vi xử lý

- Hiện nay, có rất nhiều nhà sản xuất ra các chip vi xử lý:Intel, AMD, Motorola, Cyrix …
- Thông thường, một họ vi xử lý là các chip vi xử lý được sản xuất bởi một nhà sản xuất nào đó.
- Trong phạm vi một họ vi xử lý, theo thời gian và theo công nghệ chế tạo có các đời (thế hệ) vi xử lý khác nhau phân biệt theo Độ dài Từ của chúng (bit) và tốc độ (Hz).
- Độ dài Từ (Word Length) của một chip vi xử lý là kích cỡ tối đa của các toán hạng nhị phân mà nó có thể thực hiện các phép toán trên đó.

Tốc độ của họ vi xử lý x86 của Intel



Họ vi xử lý x86 của Intel

Model	Năm sản xuất	Số lượng Transistor
4004	1971	2,300
8008	1972	2,500
8080	1974	5,000
8086	1978	29,000
80286	1982	120,000
80386™ processor	1985	275,000
80486™ DX processor	1989	1,180,000
Pentium® processor	1993	3,100,000
Pentium II processor	1997	7,500,000
Pentium III processor	1999	24,000,000
Pentium 4 processor	2000	55,000,000

Họ vi xử lý x86 của Intel 70's

	4004	8008	8080	8086
Introduced	11/15/71	4/1/72	4/1/74	6/8/78
Clock Speeds	108KHz	200KHz	2MHz	5MHz, 8MHz, 10MHz
Bus Width	4 bits	8 bits	8 bits	16 bits
Number of Transistors	2,300 (10 microns)	3,500 (10 microns)	6,000 (6 microns)	29,000 (3 microns)
Addressable Memory	640 bytes	16 KBytes	64 KBytes	1 MB
Virtual Memory	--	--	--	--
Brief Description	First microcomputer chip, Arithmetic manipulation	Data/character manipulation	10X the performance of the 8008	10X the performance of the 8080

Họ vi xử lý x86 của Intel 80's

	80286	Intel386™ DX Microprocessor	Intel386™ SX Microprocessor	Intel486™ DX CPU Microprocessor
Introduced	2/1/82	10/17/85	6/16/88	4/10/89
Clock Speeds	6MHz, 8MHz, 10MHz, 12.5MHz	16MHz, 20MHz, 25MHz, 33MHz	16MHz, 20MHz, 25MHz, 33MHz	25MHz, 33MHz, 50MHz
Bus Width	16 bits	32 bits	16 bits	32 bits
Number of Transistors	134,000 (1.5 microns)	275,000 (1 micron)	275,000 (1 micron)	1.2 million (1 micron) (.8 micron with 50MHz)
Addressable Memory	16 megabytes	4 gigabytes	16 megabytes	4 gigabytes
Virtual Memory	1 gigabyte	64 terabytes	64 terabytes	64 terabytes
Brief Description	3-6X the performance of the 8086	First X86 chip to handle 32-bit data sets	16-bit address bus enabled low-cost 32-bit processing	Level 1 cache on chip

Họ vi xử lý x86 của Intel 90's

	Intel486™ SX Microprocessor	Pentium® Processor	Pentium® Pro Processor	Pentium® II Processor
Introduced	4/22/91	3/22/93	11/01/95	5/07/97
Clock Speeds	16MHz, 20MHz, 25MHz, 33MHz	60MHz,66MHz	150MHz, 166MHz, 180MHz, 200MHz	200MHz, 233MHz, 266MHz, 300MHz
Bus Width	32 bits	64 bits	64 bits	64 bits
Number of Transistors	1.185 million (1 micron)	3.1 million (.8 micron)	5.5 million (0.35 micron)	7.5 million (0.35 micron)
Addressable Memory	4 gigabytes	4 gigabytes	64 gigabytes	64 gigabytes
Virtual Memory	64 terabytes	64 terabytes	64 terabytes	64 terabytes
Brief Description	Identical in design to Intel486™ DX but without math coprocessor	Superscalar architecture brought 5X the performance of the 33-MHz Intel486™ DX processor	Dynamic execution architecture drives high-performing processor	Dual independent bus, dynamic execution, Intel MMX™ technology

Evolution of Intel Microprocessors: 1971 to 2003

Family	Trade Name (Code Name for Future Chips)	Clock Frequency in MegaHertz***	Millions of Instructions per Second	Date of Introduction	Number of Transistors	Design Rule (Pixel Size)	Address Bus Bits
80786	(Northwood)	3,000.0 MHz	TBA	2003	TBA	0.13 micron	64 bit
80786	(Madison)	TBA	TBA	2003	TBA	0.13 micron	64 bit
80786	(Deerfield)**	TBA	TBA	2002Q2	TBA	0.13 micron	64 bit
80786	(McKinley)	1,000.0 MHz	TBA	2001Q4	TBA	0.18 micron	64 bit
80786	Itanium (Merced)	800.0 MHz	TBA	2000H2	TBA	0.18 micron	64 bit
80686	(Willamette)	1,100.0 MHz	*1,100.00 MIPS	2000Q1	TBA	0.18 micron	32 bit
80686	P III Xeon	733.0 MHz	*733.00 MIPS	October 25, 1999	28.0 million	0.18 micron	32 bit
80686	Mobile P II	400.0 MHz	*400.00 MIPS	June 14, 1999	27.4 million	0.18 micron	32 bit
80686	P III Xeon	550.0 MHz	*550.00 MIPS	March 17, 1999	9.5 million	0.25 micron	32 bit
80686	Pentium III	500.0 MHz	*500.00 MIPS	February 26, 1999	9.5 million	0.25 micron	32 bit
80686	P II Xeon	400.0 MHz	*400.00 MIPS	June 29, 1998	7.5 million	0.25 micron	32 bit
80686	Pentium II	333.0 MHz	*333.00 MIPS	January 26, 1998	7.5 million	0.25 micron	32 bit
80686	Pentium II	300.0 MHz	*300.00 MIPS	May 7, 1997	7.5 million	0.35 micron	32 bit
80586	Pentium Pro	200.0 MHz	*200.00 MIPS	November 1, 1995	5.5 million	0.35 micron	32 bit
90586	Pentium	133.0 MHz	*133.00 MIPS	June 1995	3.3 million	0.35 micron	32 bit
80586	Pentium	90.0 MHz	*90.00 MIPS	March 7, 1994	3.2 million	0.60 micron	32 bit
80586	Pentium	60.0 MHz	*60.00 MIPS	March 22, 1993	3.1 million	0.80 micron	32 bit
80486	80486 DX2	50.0 MHz	*50.00 MIPS	March 3, 1992	1.2 million	0.80 micron	32 bit
80486	486 DX	25.0 MHz	20.00 MIPS	April 10, 1989	1.2 million	1.00 micron	32 bit
80386	386 DX	16.0 MHz	5.00 MIPS	October 17, 1985	275,000	1.50 micron	16 bit
80286	80286	6.0 MHz	0.90 MIPS	February 1982	134,000	1.50 micron	16 bit
8086	8086	5.0 MHz	0.33 MIPS	June 8, 1978	29,000	3.00 micron	16 bit
8080	8080	2.0 MHz	0.64 MIPS	April 1974	6,000	6.00 micron	8 bit
8008	8008	.2 MHz	0.06 MIPS	April 1972	3,500	10.00 micron	8 bit
4004	4004	.1 MHz	0.06 MIPS	November 15, 1971	2,300	10.00 micron	4 bit

* Approximately one instruction per processor clock cycle

** Itanium, formerly codenamed Merced, may be replaced by McKinley if further delayed. Deerfield is a low cost version of Madison.

*** 1 KHz (KiloHertz) = 1 thousand cycles per second; 1 MegaHertz = 1 thousand KiloHertz; 100 KHz = .1 MHz,
1 GHz (GigaHertz) = 1 billion cycles per second; 1 GigaHertz = 1 thousand MegaHertz

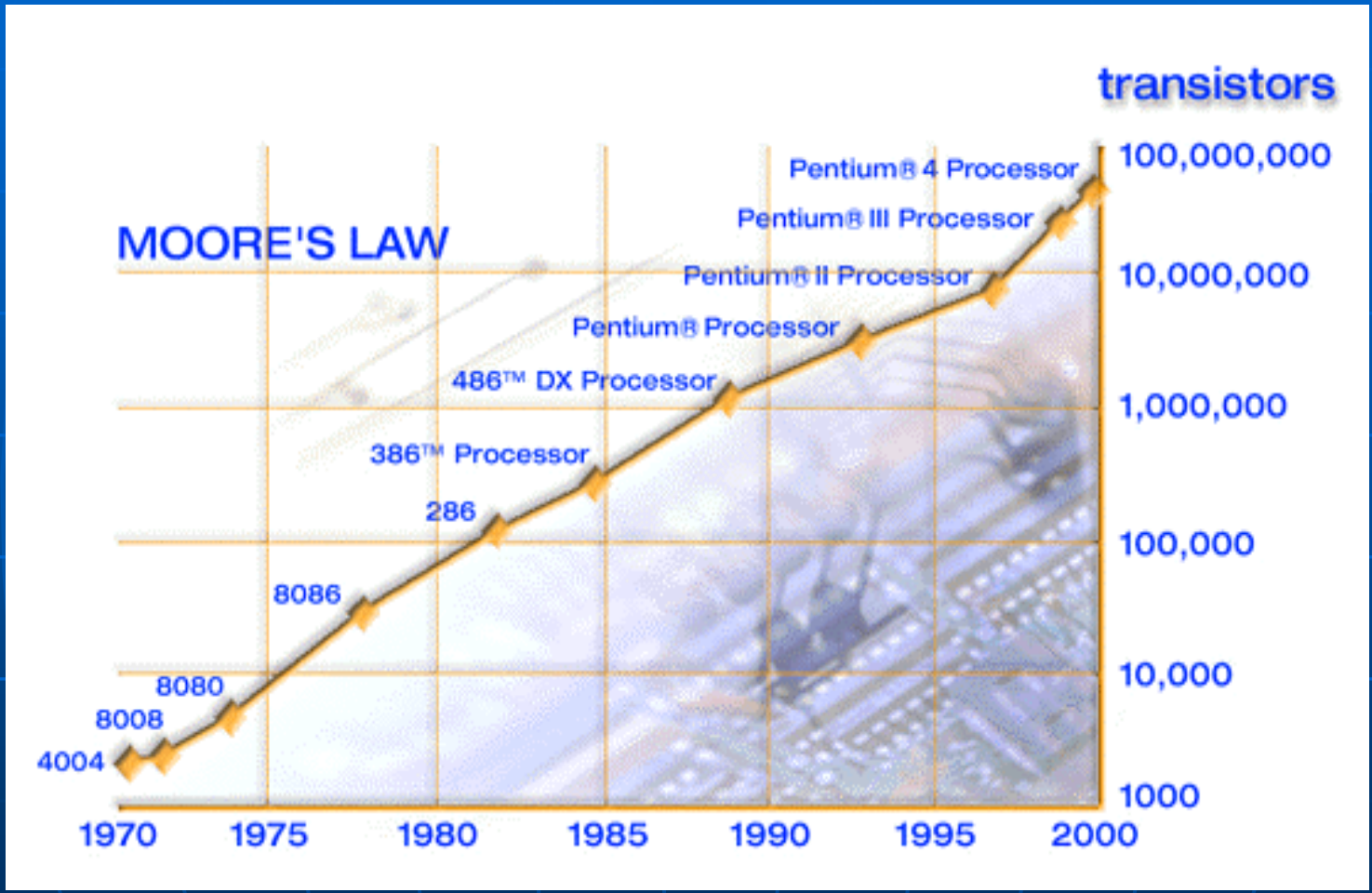
TBA To be announced

http://www.csc-ca.com/processors/intel/future.htm (one source of data for future microprocessors)
http://www.Intel.com/pressroom/kits/processors/quickref.htm (source of data for released microprocessors)

Evolution of the Intel Microprocessors

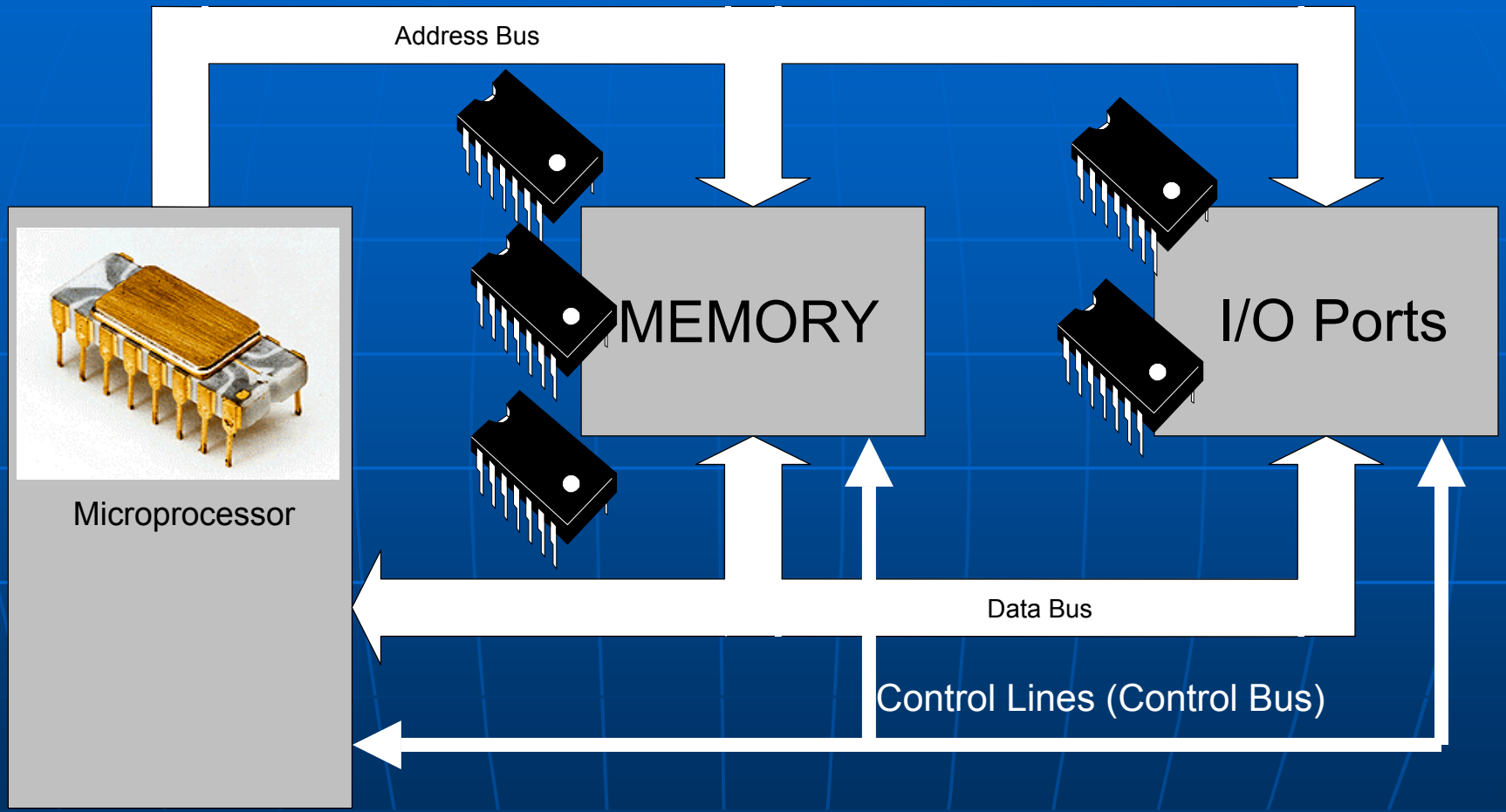
	8080	8085	8086	8088	80286	80386	80486	Pentium	P Pro	P II	P III
Year Introduced	1974	1976	1978	1979	1982	1985	1989	1992	1995	1997	
Clock rate (Hz)	2-3	3-8	5–10	5-8	6-16	16-33	25-50	60-166	150-200	200-300	450-1.13G
# Transistors	4.5k	6.5k	29k	29k	130k	275k	1.2M	3.1M	5.5M	7.5M	8.2M
Physical Memory	64k	64k	1M	1M	16M	4G	4G	4G	64G	64G	64G
Internal Data Bus	8	8	16	16	16	32	32	32	32	32	32
External Data Bus	8	8	16	8	16	32	32	64	64	64	64
Address Bus	16	16	20	20	24	32	32	32	36	36	64
Data type (bits)	8	8	8,16	8,16	8,16	8,16,32	8,16,32	8,16,32	8,16,32	8,16,32	8,16,32

Luật Moore



Dr. Gordon E. Moore, Chairman Emeritus of Intel Corporation, dự đoán rằng
Cứ một năm rưỡi thì số lượng transistor được tích hợp trên chip vi xử lý tăng gấp đôi

2.3 Hệ thống vi xử lý



Sơ đồ khối chức năng của một hệ thống vi xử lý

Hệ thống vi xử lý

- Gồm 3 khối chức năng: Vi xử lý, Bộ nhớ, Các cổng I/O
- Bộ nhớ được thực hiện bằng các chip nhớ bán dẫn ROM hoặc RWM, là nơi lưu trữ chương trình và dữ liệu. Đối với vi xử lý, bộ nhớ là một tập hợp các ô nhớ phân biệt theo địa chỉ của chúng.
- Các cổng I/O được thực hiện bằng các chip MSI hoặc LSI, là phần mạch giao tiếp giữa vi xử lý với các thiết bị I/O. Bộ vi xử lý cũng phân biệt các cổng I/O theo địa chỉ của chúng.

Hệ thống vi xử lý

- 3 khối chức năng: Vi xử lý, Bộ nhớ, Các cổng I/O của một hệ thống vi xử lý trao đổi tín hiệu với nhau thông qua Bus hệ thống.
- Bus hệ thống là một tập hợp các đường truyền dẫn dùng chung, bao gồm: Bus địa chỉ (A-Bus), Bus dữ liệu (D-Bus) và Bus điều khiển (C-Bus)
- Các tín hiệu địa chỉ di chuyển trên A-Bus theo hướng từ vi xử lý đến Bộ nhớ và các cổng I/O. Số lượng đường truyền dẫn của A-Bus (gọi là Độ rộng của A-Bus) tính bằng bit, phản ánh khả năng quản lý bộ nhớ của chip vi xử lý.

Hệ thống vi xử lý

- Các tín hiệu dữ liệu di chuyển trên D-Bus theo cả 2 hướng từ vi xử lý đến Bộ nhớ và các cổng I/O và ngược lại (mỗi lúc một hướng). Số lượng đường truyền dẫn của D-Bus (gọi là Độ rộng của D-Bus) tính bằng bit, phản ánh một phần tốc độ trao đổi dữ liệu của chip vi xử lý vớI các khối chức năng khác.
- Đa số các tín hiệu trên C-Bus là các tín hiệu điều khiển riêng lẽ, có tín hiệu xuất phát từ vi xử lý, có tín hiệu đi vào vi xử lý. Vi xử lý sử dụng các tín hiệu này để điều khiển hoạt động và nhận biết trạng thái của các khối chức năng khác.

Thiết kế phần cứng của hệ thống vi xử lý

- Thiết kế bộ nhớ cho hệ thống vi xử lý: Ghép nối các chip nhớ bán dẫn sẵn có với bus hệ thống sao cho khi bộ vi xử lý truy cập bộ nhớ thì không xảy ra xung đột giữa các chip nhớ với nhau và không xung đột với các chip dùng làm cổng I/O
- Tương tự, Thiết kế các cổng I/O cho hệ thống vi xử lý: Ghép nối các chip MSI hoặc LSI thường dùng làm cổng I/O với bus hệ thống sao cho khi bộ vi xử lý truy cập các thiết bị I/O thì không xảy ra xung đột giữa các chip đó với nhau và không xung đột với các chip dùng làm bộ nhớ

Thiết kế phần mềm của hệ thống vi xử lý

- Viết chương trình điều khiển hoạt động của hệ thống phần cứng theo chức năng mong muốn (thường dùng ngôn ngữ Assembly của chip vi xử lý dùng trong hệ thống)
- Dịch chương trình đã viết sang ngôn ngữ máy sử dụng các chương trình dịch thích hợp
- Nạp chương trình ngôn ngữ máy vào bộ nhớ của hệ thống vi xử lý
- Kiểm tra hoạt động của hệ thống và thực hiện các hiệu chỉnh nếu cần thiết
- Có thể nhờ sự trợ giúp của các chương trình mô phỏng trên máy tính

Bay giảng Kỹ thuật Vi xử lý

Ngành Điện tử-Viễn thông
Đại học Bách khoa Đà Nẵng
của Hồ Viết Việt, Khoa CNTT-ĐTVT

Tài liệu tham khảo

[1] Kỹ thuật vi xử lý, Văn Thế Minh, NXB Giáo dục, 1997

[2] Kỹ thuật vi xử lý và Lập trình Assembly cho hệ vi xử lý, Đỗ Xuân Tiến, NXB Khoa học & kỹ thuật, 2001

Chương 3

Vi xử lý 8088-Intel

3.1 Kiến trúc và hoạt động của 8088
- Nguyên lý hoạt động
- Sơ đồ khối chức năng

3.2 Cấu trúc thanh ghi của 8088

3.3 Phương pháp quản lý bộ nhớ

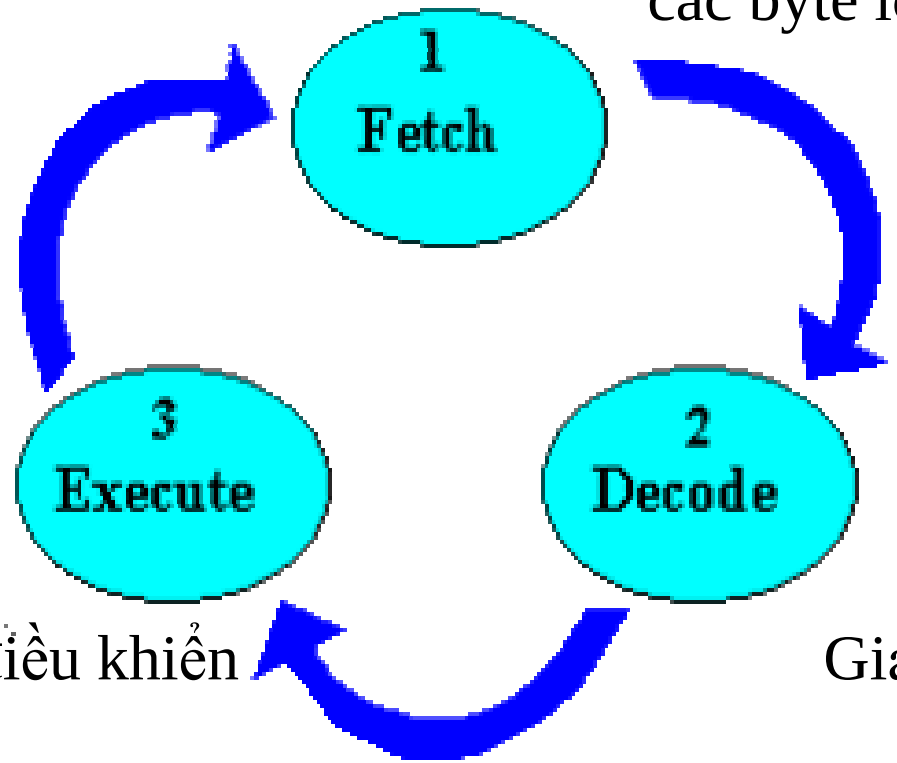
3.4 Mô tả tập lệnh Assembly

Intel® 8088 Chip

Nguyên lý hoạt động của một bộ vi xử lý

Lấy - Giải mã - Thực hiện lệnh

Tìm và copy
các byte lệnh từ bộ nhớ



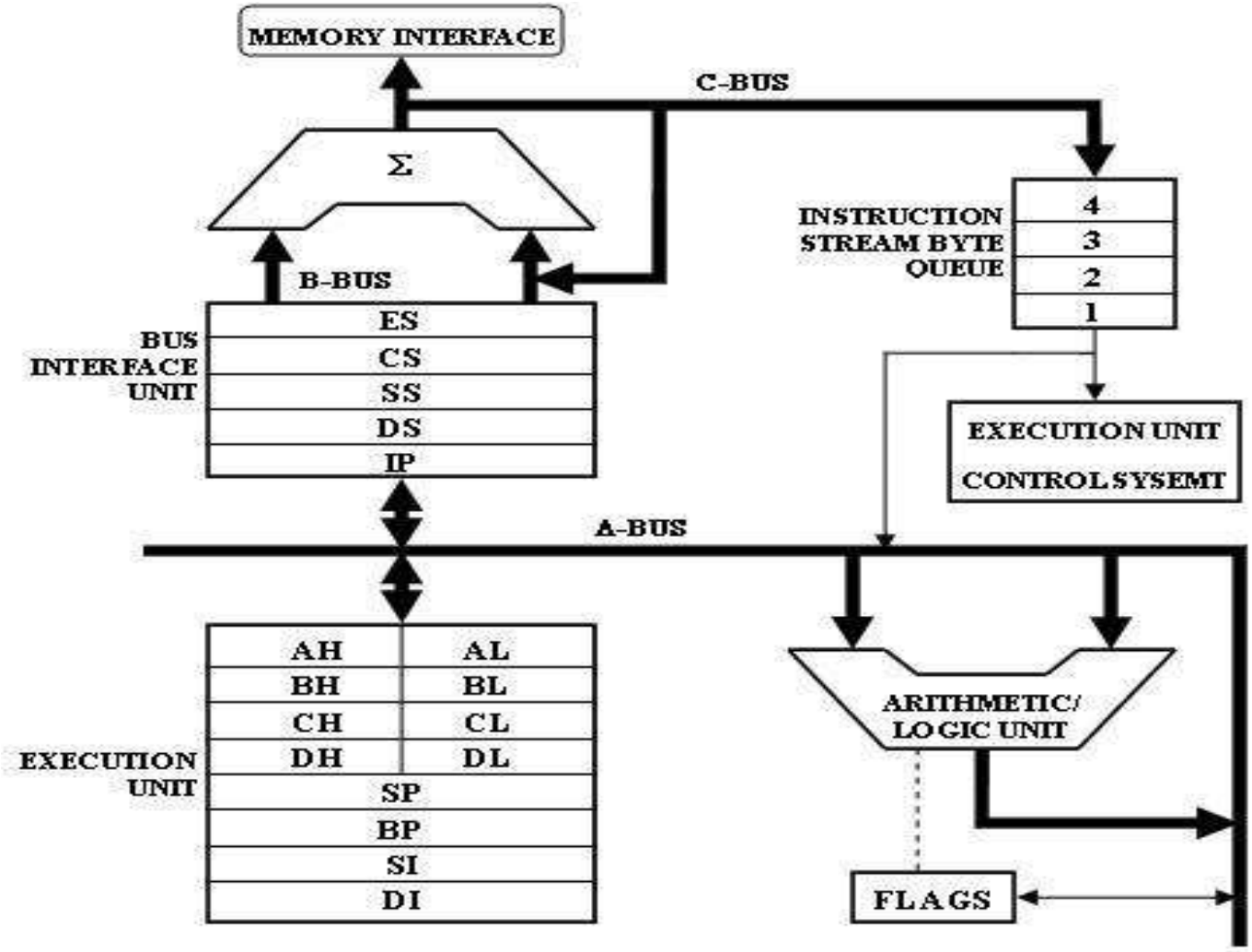
Giải mã lệnh

Tạo ra các tín hiệu điều khiển
để thực hiện lệnh

Chu kỳ lệnh và Chu kỳ máy

- Chu kỳ lệnh: Tổng thời gian tìm lệnh, giải mã lệnh và thực hiện 1 lệnh
- Nói chung, Chu kỳ lệnh của các lệnh khác nhau là khác nhau
- Chu kỳ lệnh bao giờ cũng bằng một số nguyên lần chu kỳ máy
- Chu kỳ máy bằng nghịch đảo của tần số hoạt động (tốc độ đồng hồ) của bộ vi xử lý

3.1 Kiến trúc và Hoạt động của 8088



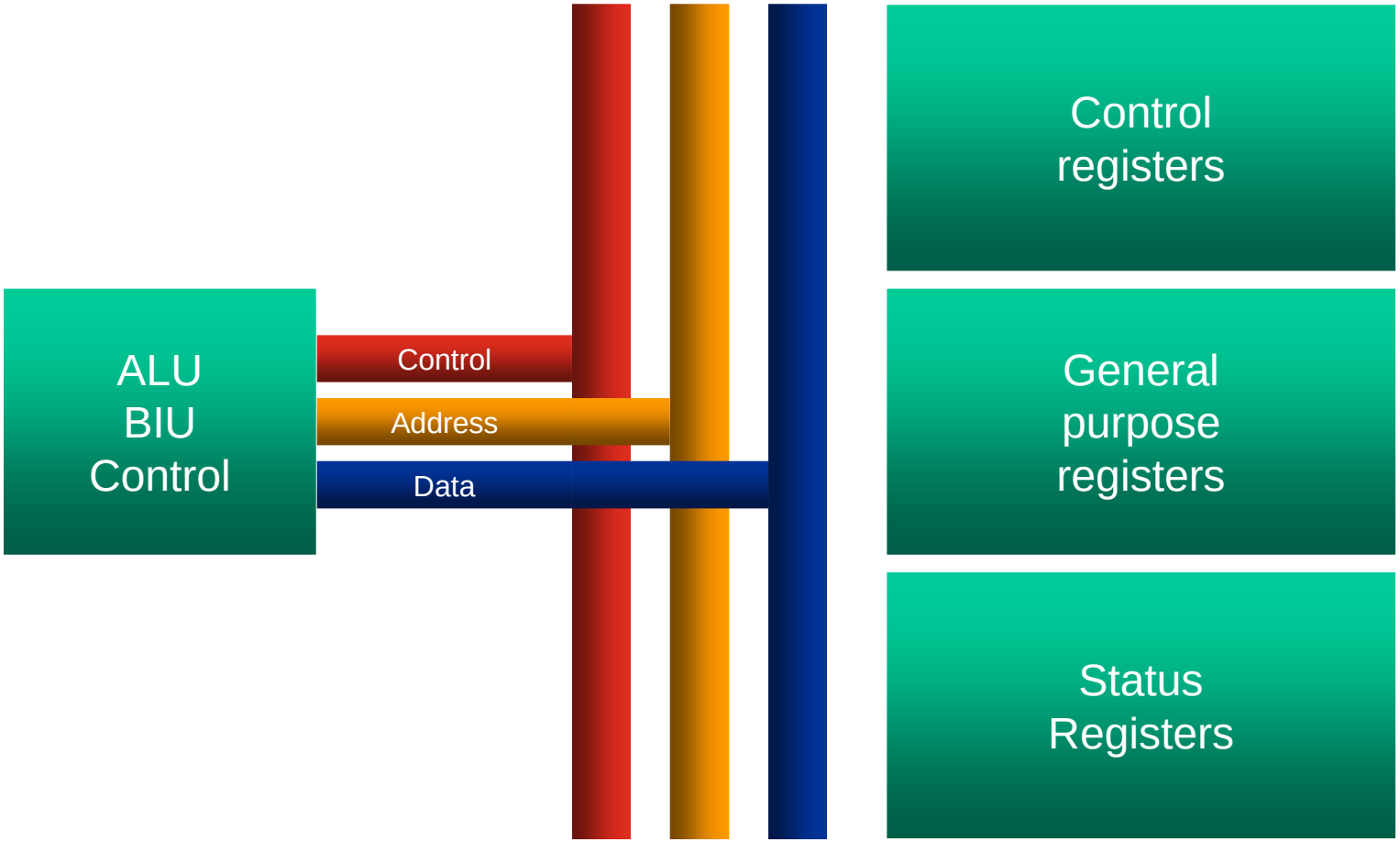
Đơn vị giao tiếp Bus - BIU

- Phát các tín hiệu địa chỉ đến bộ nhớ và các cổng I/O thông qua A-Bus
- Đọc mã lệnh từ bộ nhớ thông qua D-Bus
- Đọc dữ liệu từ bộ nhớ thông qua D-Bus
- Ghi dữ liệu vào bộ nhớ thông qua D-Bus
- Đọc dữ liệu từ các cổng I thông qua D-Bus
- Ghi dữ liệu ra các cổng O thông qua D-Bus

Đơn vị thực hiện - EU

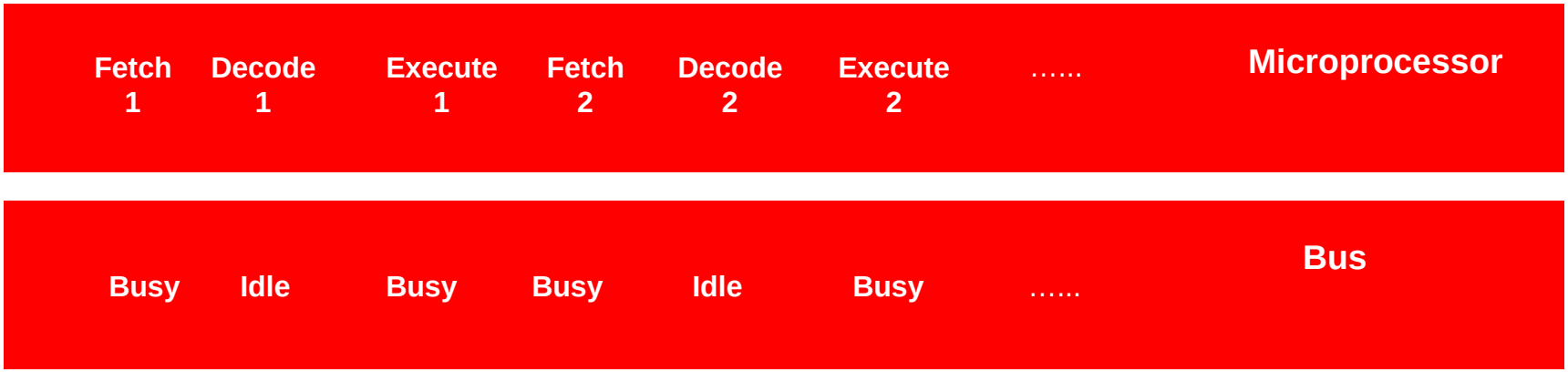
- Bao gồm CU và ALU
- CU : Giải mã lệnh để tạo ra các tín hiệu điều khiển nhằm thực hiện lệnh đã được giải mã
- ALU: thực hiện các thao tác khác nhau đối với các toán hạng của lệnh

Tổ chức của microprocessor

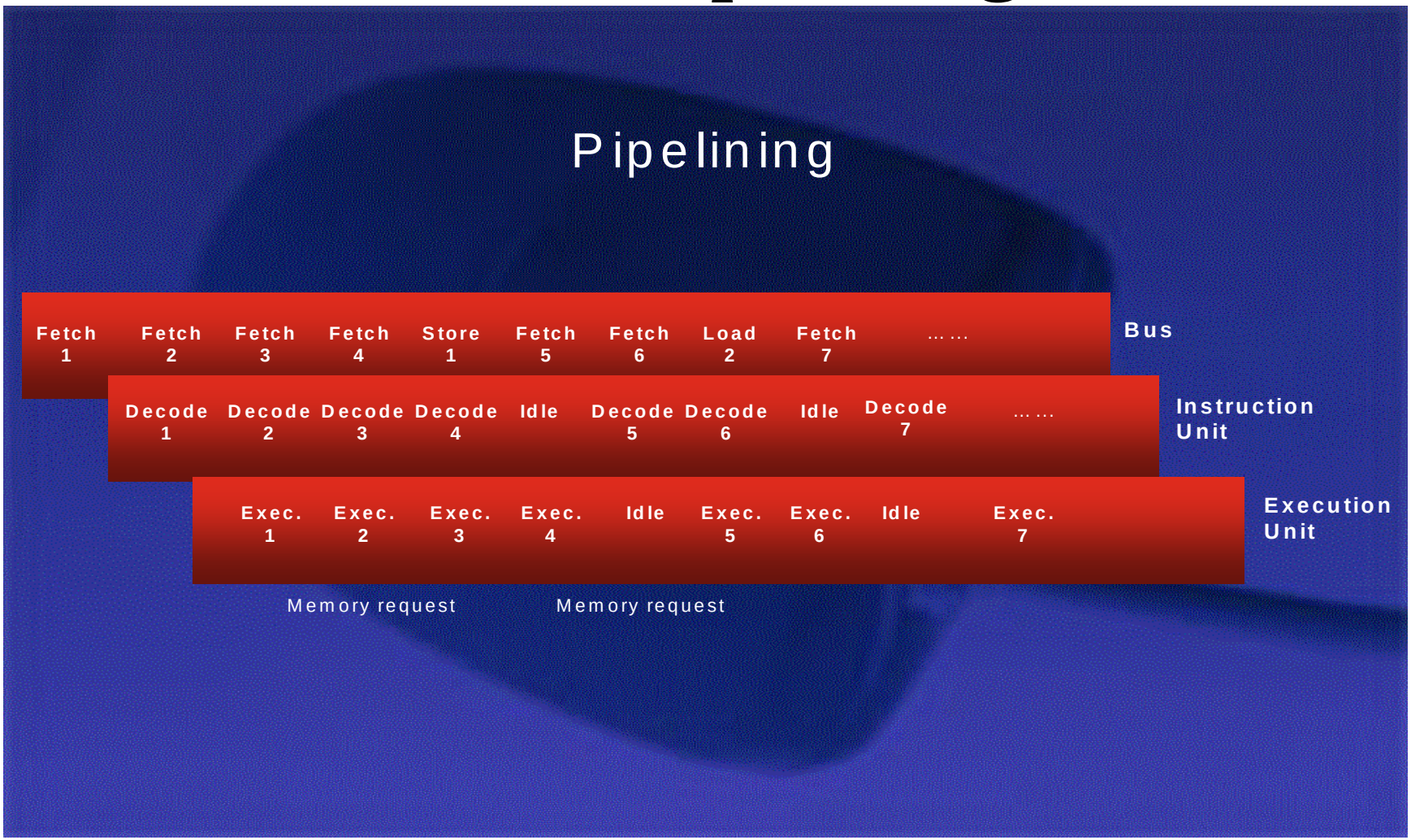


Xử lý lệnh của các vi xử lý trước 8086/8088

- Một thủ tục đơn giản gồm 3 bước:
  - Lấy lệnh từ bộ nhớ
  - Giải mã lệnh
  - Thực hiện lệnh
    - Lấy các toán hạng từ bộ nhớ (nếu có)
    - Lưu trữ kết quả

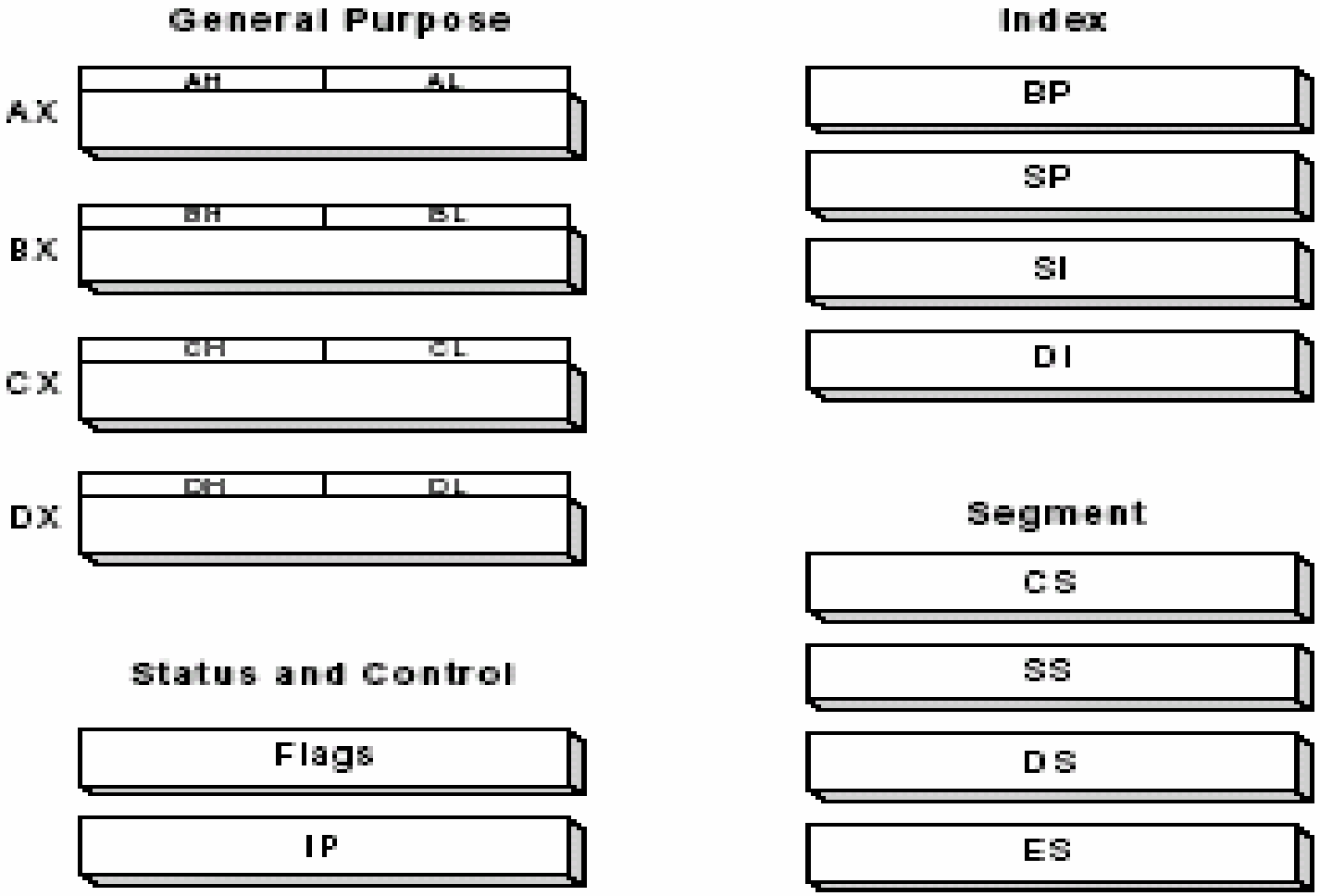


Cơ chế Pipelining

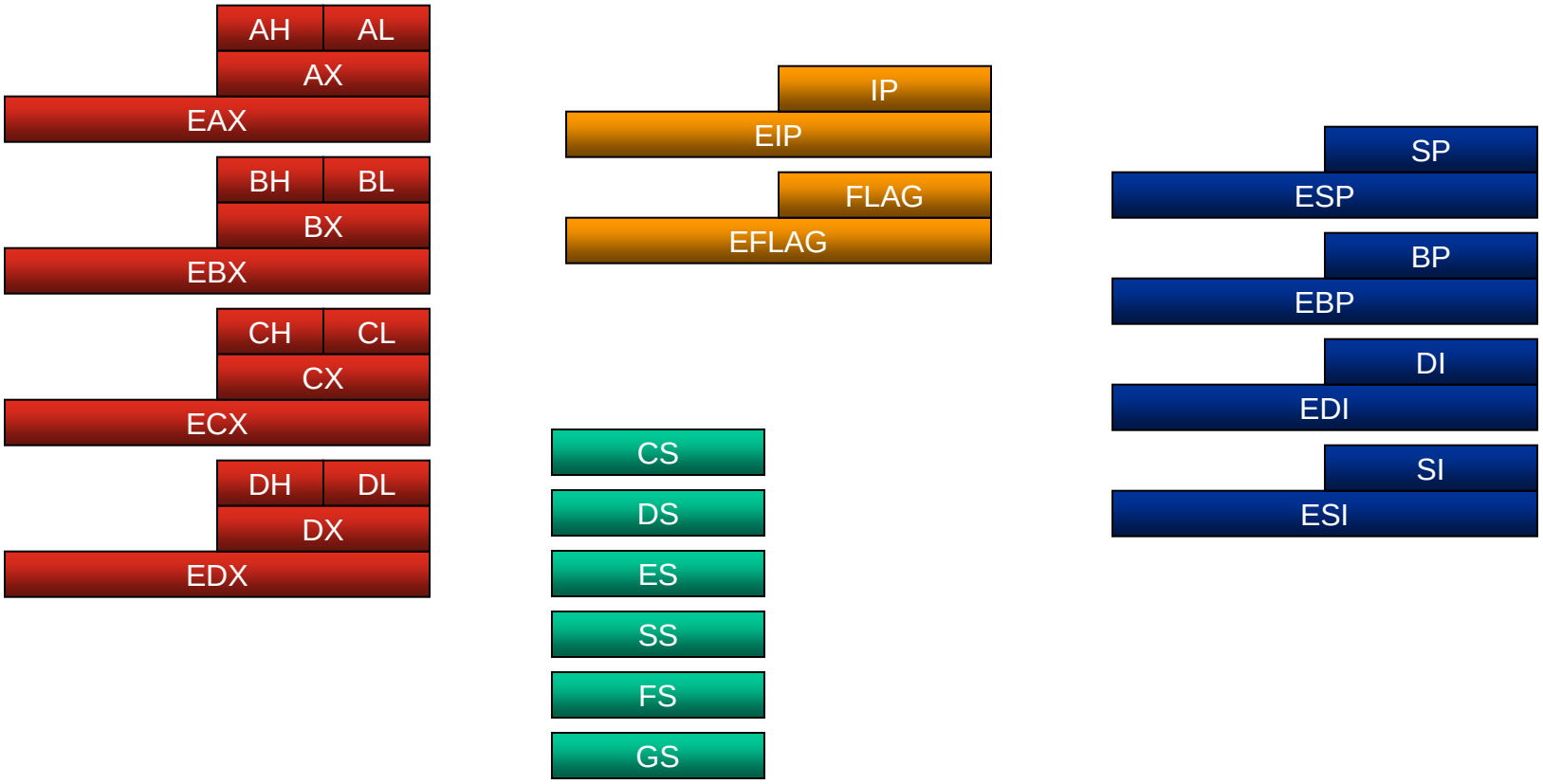


3.2 Cấu trúc thanh ghi của 8088

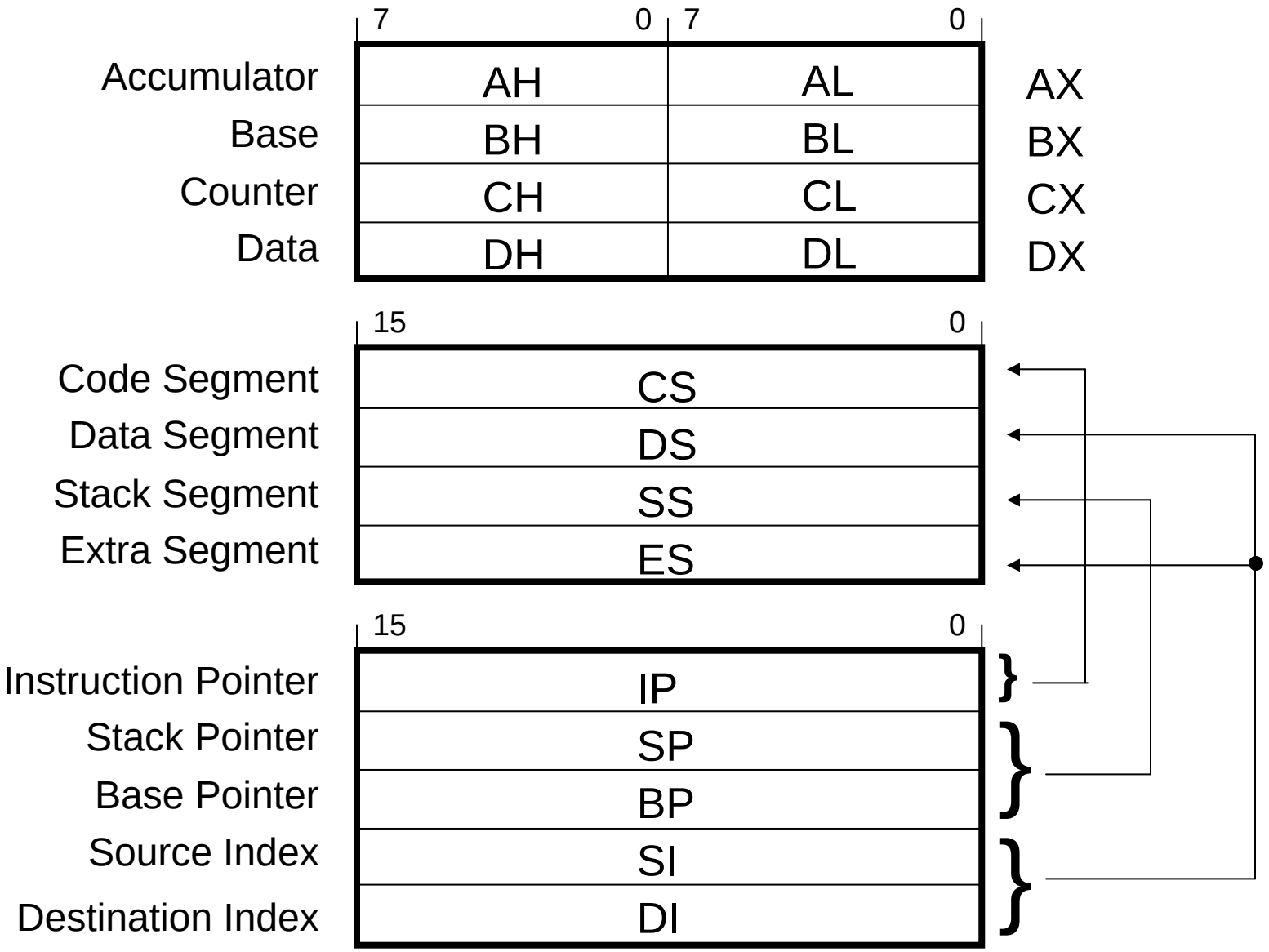
8088 có 14 thanh ghi 16-bit



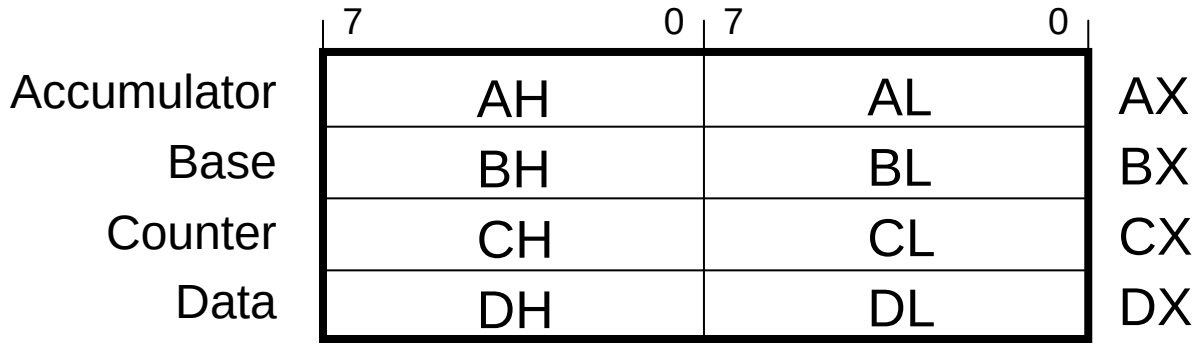
Cấu trúc thanh ghi của họ x86



Cấu trúc thanh ghi 8086/8088

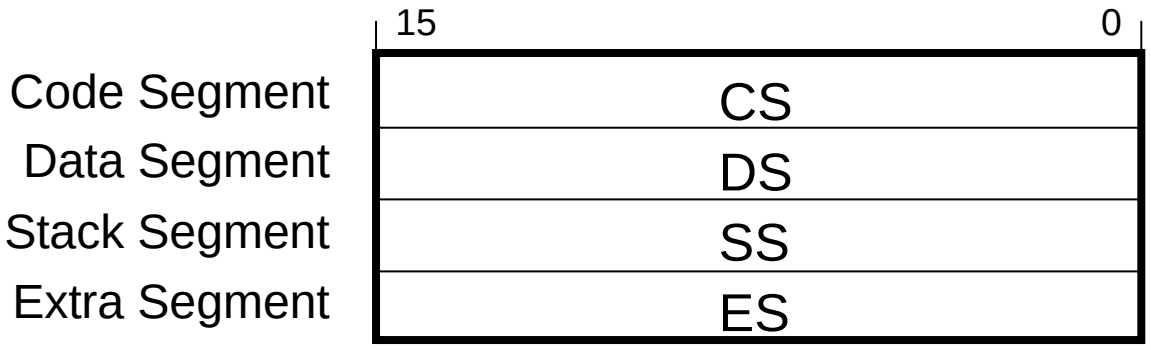


Các thanh ghi đa năng



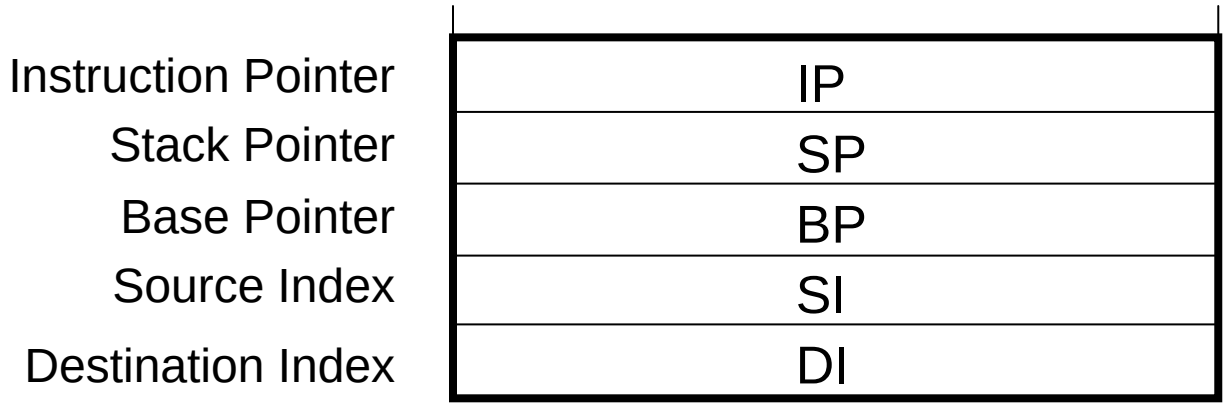
- Có thể truy cập như các thanh ghi 8-bit
- Lưu trữ tạm thời dữ liệu để truy cập nhanh hơn và tránh khỏi phải truy cập bộ nhớ
- Có công dụng đặc biệt đối với một số câu lệnh

Các thanh ghi segment



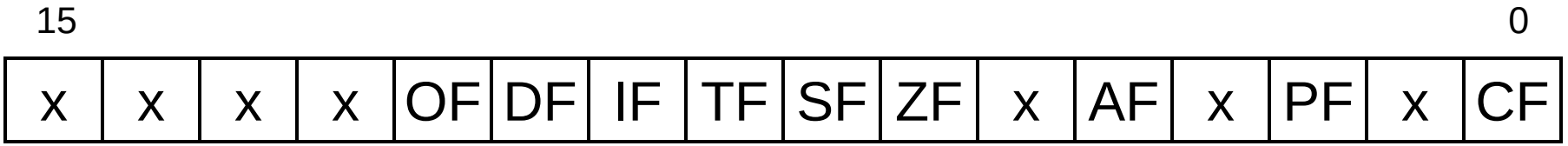
- Lưu trữ địa chỉ segment của một ô nhớ cần truy cập
- Kết hợp với các thanh ghi offset nhất định

Các thanh ghi offset



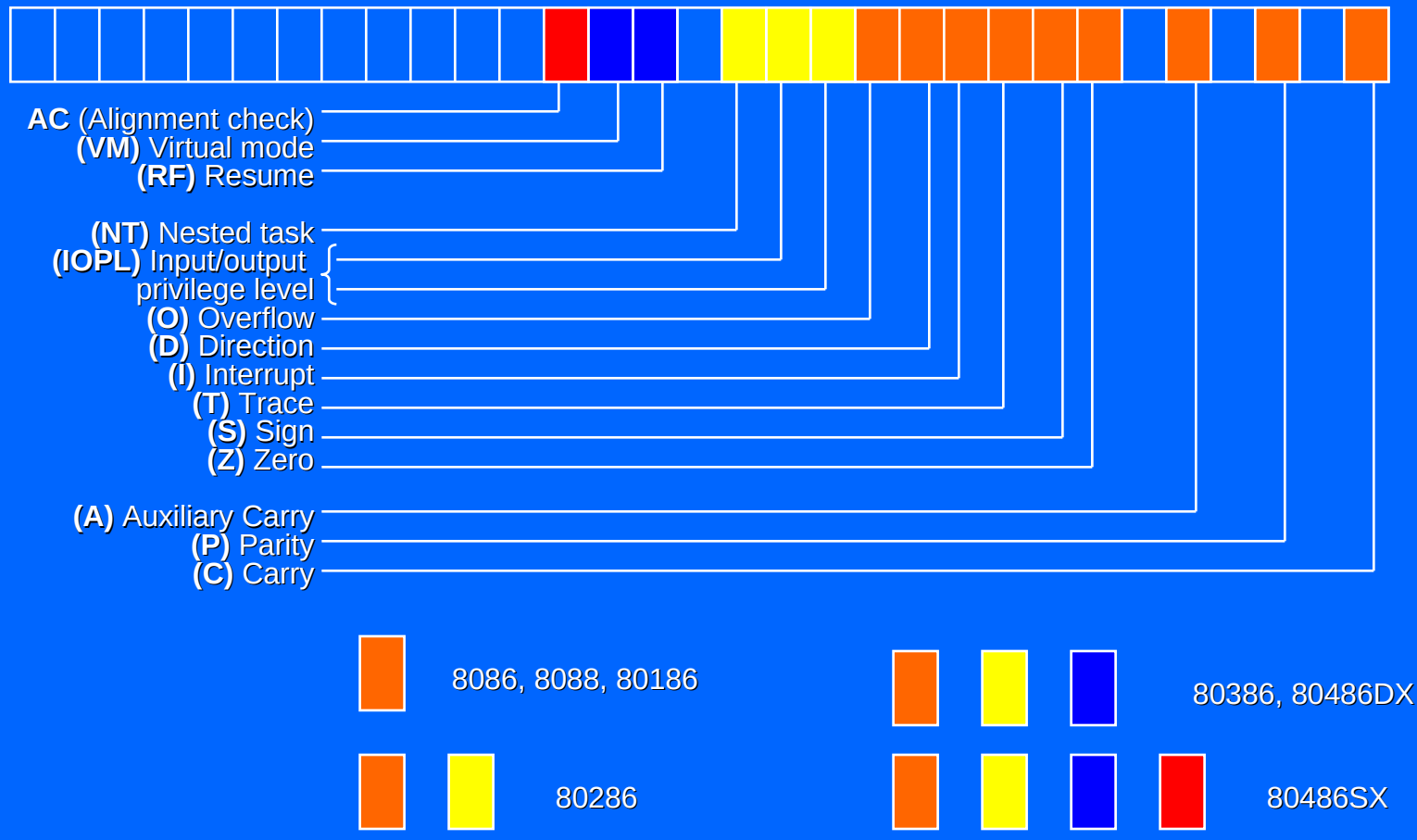
- Lưu trữ địa chỉ offset của một ô nhớ cần truy cập
- Kết hợp với các thanh ghi segment nhất định

Thanh ghi cờ



- Không phải tất cả các bit đều được sử dụng
- Mỗi bit được sử dụng được gọi là một cờ
- Các cờ đều có tên và có thể được Lập/Xoá riêng lẽ
- Bao gồm các cờ trạng thái và các cờ điều khiển

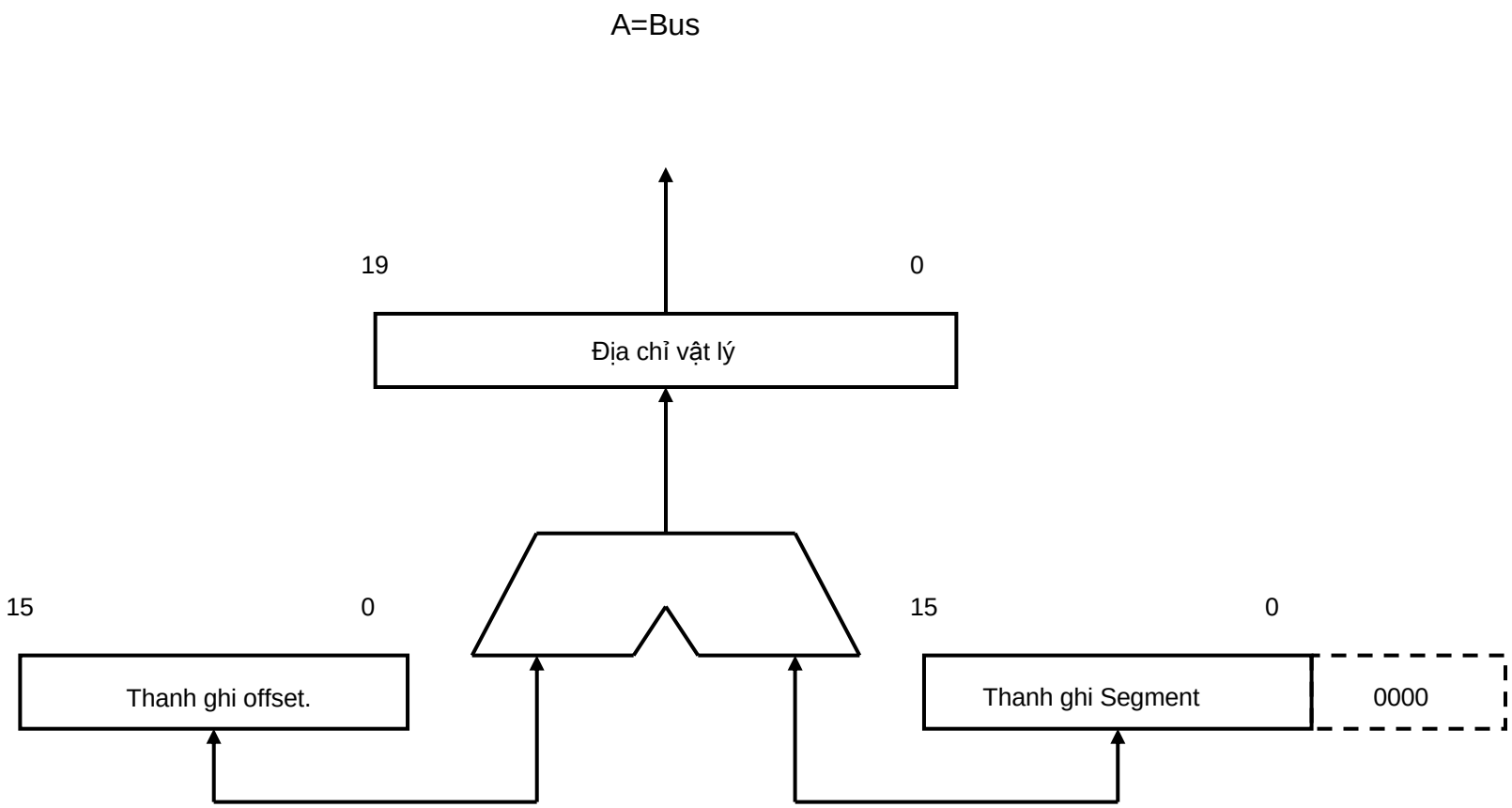
Flags register



3.3 Phương pháp quản lý bộ nhớ

- Bộ nhớ được xem là một tập hợp các ô nhớ
- Mỗi ô nhớ được nhận dạng bằng một Địa chỉ vật lý duy nhất 20-bit
- Trong hoạt động truy cập một ô nhớ, Địa chỉ vật lý của nó được tạo ra từ hai giá trị 16-bit: Địa chỉ segment và Địa chỉ Offset
- Địa chỉ logic = Địa chỉ segment:Địa chỉ offset

Mối liên hệ giữa ĐCVL và ĐCLG



3.4 Mô tả tập lệnh Assembly của 8086/8088

- Khuôn dạng: Mnemonics Các toán hạng
- Nhóm lệnh chuyển số liệu
- Nhóm lệnh số học
- Nhóm lệnh logic
- Nhóm lệnh Rẽ nhánh
- Nhóm lệnh thao tác string
- Nhóm lệnh hỗn hợp

Nhóm lệnh chuyển số liệu
Data Transfer Instructions

-Chuyển số liệu (sao chép số liệu) từ vị trí này sang vị trí khác
- Nguồn số liệu không thay đổi
- Đich sẽ có giá trị như giá trị của Nguồn
- Các lệnh chuyển số liệu không ảnh hưởng đến các cờ trạng thái trên thanh ghi cờ
- Một số lệnh tiêu biểu: MOV, XCHG

Data Transfer Instructions - MOV

Khuôn dạng: MOV Đích,Nguồn

- Tác dụng: (Đích) ← (Nguồn)
- Đích: có thể là:

1. Một thanh ghi 8 hoặc 16 bit của VXL

2. Một vị trí nhớ (1 hoặc 2 ô nhớ liên tiếp nhau)

- Nguồn: có thể là:

1. Một thanh ghi 8 hoặc 16 bit của VXL

2. Một vị trí nhớ (1 hoặc 2 ô nhớ liên tiếp nhau)

3. Một giá trị cụ thể

Một số lưu ý đối với MOV

- Đích và Nguồn phải có cùng kích cỡ
- Đích và Nguồn không thể đồng thời thuộc bộ nhớ
- Nếu Đích là một thanh ghi segment của VXL thì Nguồn không thể là một giá trị cụ thể (nói cách khác, không thể nạp giá trị trực tiếp cho một thanh ghi segment bằng lệnh MOV)

Data Transfer Instructions - XCHG

Khuôn dạng: XCHG T/h1,T/h2

- Tác dụng: (T/h1) ← (T/h2)
- T/h1: có thể là:

1. Một thanh ghi 8 hoặc 16 bit của VXL

2. Một vị trí nhớ (1 hoặc 2 ô nhớ liên tiếp nhau)

- T/h2: có thể là:

1. *Một thanh ghi 8 hoặc 16 bit của VXL*
2. *Một vị trí nhớ (1 hoặc 2 ô nhớ liên tiếp nhau)*

Một số lưu ý đối với XCHG

- T/h1 và T/h2 phải có cùng kích cỡ
- T/h1 và T/h2 không thể đồng thời thuộc bộ nhớ
- T/h1 và T/h2 không thể là các thanh ghi segment

Các mode địa chỉ

- Khi thực hiện lệnh, VXL sẽ thực hiện những thao tác nhất định trên số liệu, các số liệu này được gọi chung là các toán hạng.
- Các toán hạng trong một câu lệnh có thể là một phần của câu lệnh (ở dạng mã máy), có thể nằm ở một thanh ghi của VXL hoặc ở Bộ nhớ
-Cách xác định toán hạng trong các câu lệnh được gọi là các mode (định) địa chỉ

Các mode địa chỉ

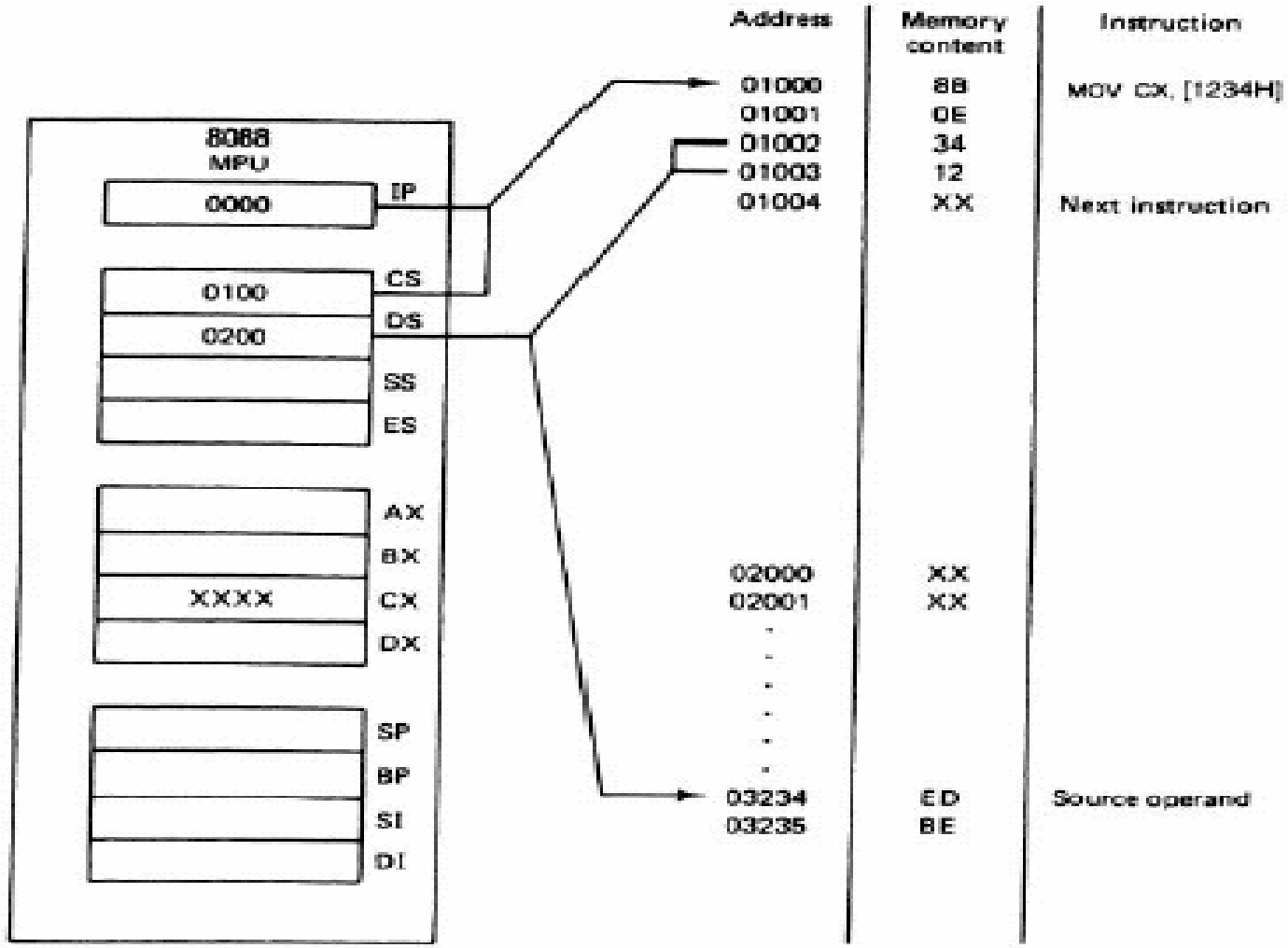
- Mode địa chỉ thanh ghi: MOV AX,BX
- Mode địa chỉ tức thì: MOV AL,55h
- Các mode địa chỉ bộ nhớ: Các cách thức xác định địa chỉ vật lý của toán hạng nằm trong bộ nhớ:

    Mode địa chỉ trực tiếp

    Các mode địa chỉ gián tiếp …

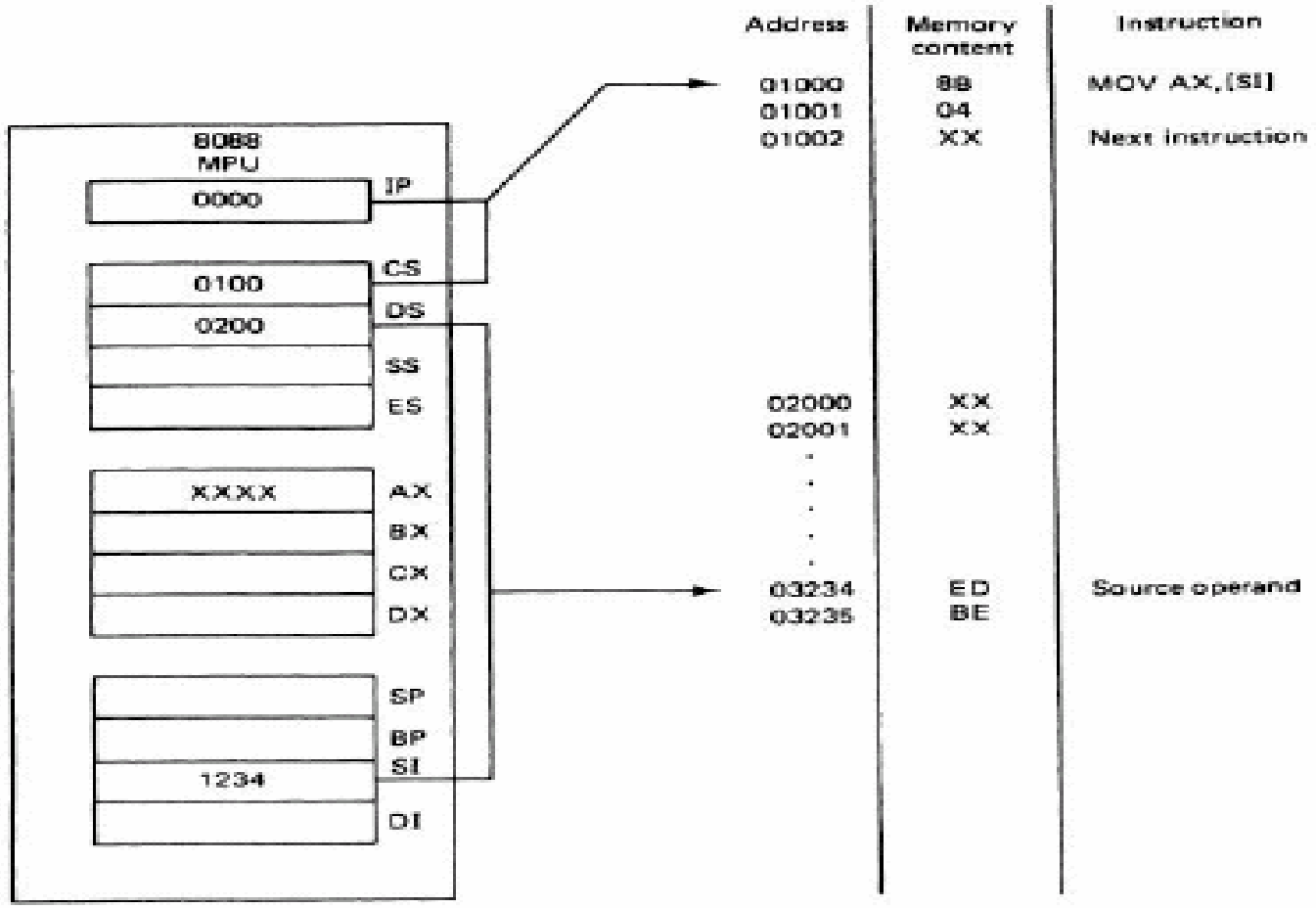
Mode địa chỉ trực tiếp
(**Direct Addressing Mode)**

MOV CX, [1234h]



Mode địa chỉ gián tiếp thanh ghi (**Register Indirect Addressing Mode**)

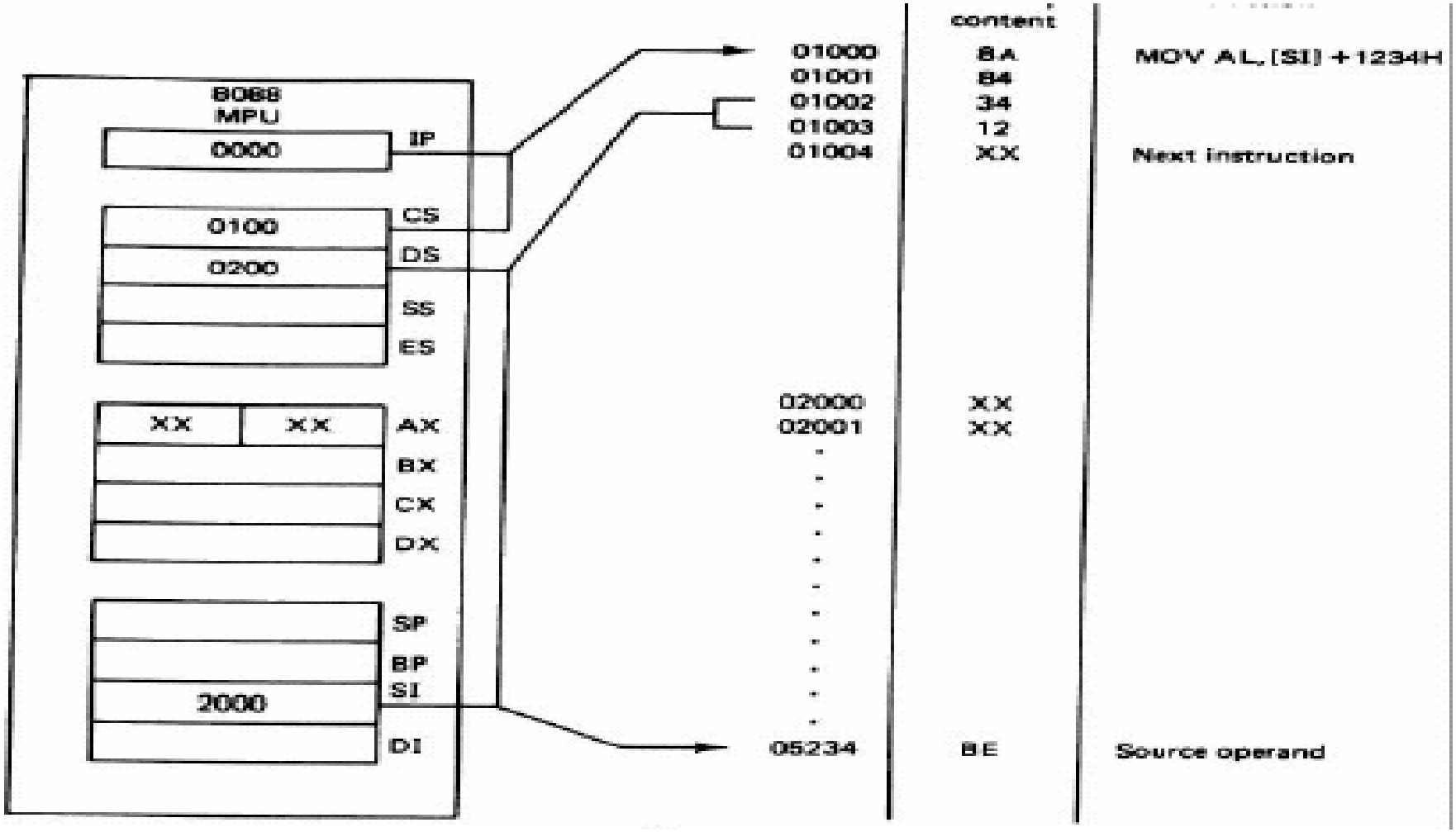
MOV AX, [SI]



Mode địa chỉ cơ sở-chỉ số

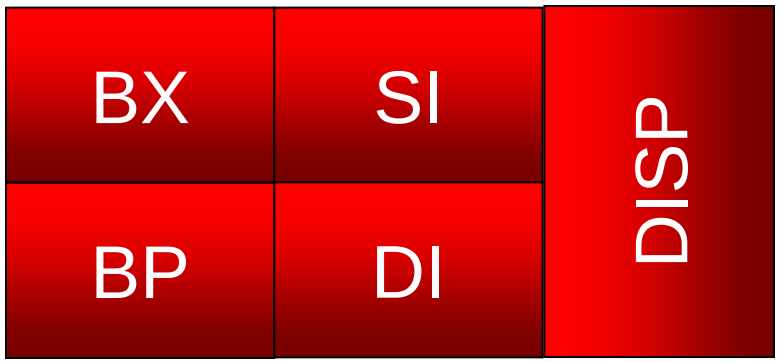
(Based-Indexed Addressing Mode)

MOV AH, [BX] [SI] + 1234h



Nhớ các mode địa chỉ bộ nhớ như thế nào?

- Tất cả bắt đầu trong bảng sau đây:



- Lấy ra 0 hoặc 1 phần tử từ mỗi cột
- (Không lấy 2 phần tử từ một cột)
- Phải lấy ít nhất 1 phần tử từ bảng

Các ví dụ

Instruction	Comment	Addressing Mode	Memory Contents
MOV AX, BX	Move to AX the 16-bit value in BX	Register	89 D8 OP MODE
MOV AX, DI	Move to AX the 16-bit value in DI	Register	89 F8 OP MODE
MOV AH, AL	Move to AL the 8-bit value in AX	Register	88 C4 OP MODE
MOV AH, 12h	Move to AH the 8-bit value 12H	Immediate	B4 12 OP DATA8
MOV AX, 1234h	Move to AX the value 1234h	Immediate	B8 34 OP DATA16
MOV AX, CONST	Move to AX the constant defined as CONST	Immediate	B8 lsb msb OP DATA16
MOV AX, X	Move to AX the address or offset of the variable X	Immediate	B8 lsb msb OP DATA16
MOV AX, [1234h]	Move to AX the value at memory location 1234h	Direct	A1 34 12 OP DISP16
MOV AX, [X]	Move to AX the value in memory location DS:X	Direct	A1 lsb msb OP DISP16

Các ví dụ

Instruction	Comment	Addressing Mode	Memory Contents
MOV [X], AX	Move to the memory location pointed to by DS:X the value in AX	Direct	A3 lsb msb OP DATA16
MOV AX, [DI]	Move to AX the 16-bit value pointed to by DS:DI	Indexed	8B 05 OP MODE
MOV [DI], AX	Move to address DS:DI the 16-bit value in AX	Indexed	89 05 OP MODE
MOV AX, [BX]	Move to AX the 16-bit value pointed to by DS:BX	Register Indirect	8B 07 OP MODE
MOV [BX], AX	Move to the memory address DS:BX the 16-bit value stored in AX	Register Indirect	89 07 OP MODE
MOV [BP], AX	Move to memory address **SS:BP** the 16-bit value in AX	Register Indirect	89 46 OP MODE
MOV AX, TAB[BX]	Move to AX the value in memory at DS:BX + TAB	Register Relative	8B 87 lsb msb OP MODE DISP16
MOV TAB[BX], AX	Move value in AX to memory address DS:BX + TAB	Register Relative	89 87 lsb msb OP MODE DISP16
MOV AX, [BX + DI]	Move to AX the value in memory at DS:BX + DI	Base Plus Index	8B 01 OP MODE

Các ví dụ

Instruction	Comment	Addressing Mode	Memory Contents
MOV [BX + DI], AX	Move to the memory location pointed to by DS:X the value in AX	Base Plus Index	89 01 OP MODE
MOV AX, [BX + DI + 1234h]	Move word in memory location DS:BX + DI + 1234h to AX register	Base Rel Plus Index	8B 81 34 12 OP MODE DISP16
MOV word [BX + DI + 1234h], 5678h	Move immediate value 5678h to memory location BX + DI + 1234h	Base Rel Plus Index	C7 81 34 12 78 56

Mã máy

Một lệnh có thể dài từ1 đến 6 byte

• Byte 1 gồm:

– Opcode (6 bit) xác định phép toán cần thực hiện

– Bit D xác định toán hạng ở REG của Byte 2 là nguồn hay đích:

1: Đích

0: Nguồn

– Bit W xác định kích cỡ của toán hạng là 8 bit hay 16 bit

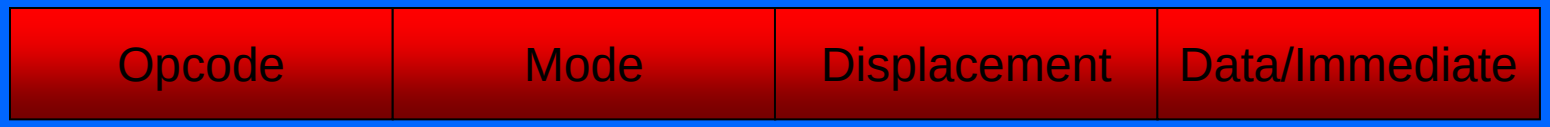
0: 8 bit

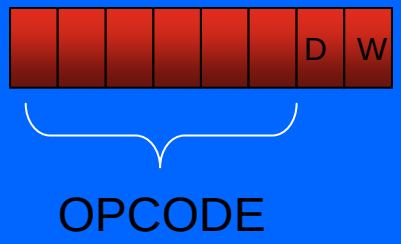
1: 16 bit

• Byte 2 gồm:Mode field (MOD), Register field (REG)

Register/memory field (R/M field)

Anatomy of an instruction

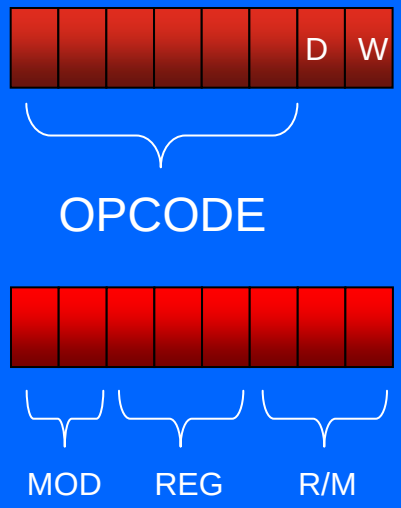




- Opcode contains the type of instruction we execute plus two special bits, D and W
- The mode byte is used only in instructions that use register addressing modes and encodes the source and destination for instructions with two operands
- D stands for direction and defines the data flow of the instruction
  - D=0, data flows from REG to R/M
  - D=1, data flows from R/M to REG
- W stands for the size of data
  - W=0, byte-sized data
  - W=1, word (in real mode) or double-word sized (in protected mode)

Anatomy of an instruction

Opcode	Mode	Displacement	Data/Immediate



- MOD field specifies the addressing mode
- 00 – no displacement
- 01 – 8-bit displacement, sign extended
- 10 – 16-bit displacement
- 11 – R/M is a register, register addressing mode
- If MOD is 00,01, or 10, the R/M field selects one of the memory addressing modes

Registers in the REG and R/M fields

Code	W=0 (Byte)	W=1 (Word)	W=1 (DWord)
000	AL	AX	EAX
001	CL	CX	ECX
010	DL	DX	EDX
011	BL	BX	EBX
100	AH	SP	ESP
101	CH	BP	EBP
110	DH	SI	ESI
111	BH	DI	EDI

Example

- Consider the instruction 8BECh
- 1000 1011 1110 1100 binary
- Opcode 100010 -> MOV
- D=1 data goes from R/M to REG
- W=1 data is word-sized
- MOD=11, register addressing
- REG=101 destination, R/M=100 source
- MOV BP, SP

Code	W=0	W=1	W=1
000	AL	AX	EAX
001	CL	CX	ECX
010	DL	DX	EDX
011	BL	BX	EBX
100	AH	SP	ESP
101	CH	BP	EBP
110	DH	SI	ESI
111	BH	DI	EDI

Displacement addressing

- If MOD is 00, 01, or 10 R/M has an entirely different meaning

MOD	FUNCTION
00	No displacement
01	8-bit sign-extended displacement
10	16-bit displacement
11	R/M is a register (register addressing mode)

Examples:

If MOD=00 and R/M=101 mode is [DI]

If MOD=01 and R/M=101 mode is [DI+33h]

If MODE=10 and R/M=101 modes is [DI+2233h]

R/M Code	Function
000	DS:BX+SI
001	DS:BX+DI
010	SS:BP+SI
011	SS:BP+DI
100	DS:SI
101	DS:DI
110	SS:BP
111	DS:BX

Example

- Instruction 8A15h
- 1000 1010 0001 0101
- Opcode 100010 -> MOV
- D=1, data flows from R/M to REG
- W=0, 8-bit argument
- MOD=00 (no displacement)
- REG=010 (DL)
- REG=101 ([DI] addressing mode)
- MOV DL, [DI]

Code	W=0	W=1	W=1
000	AL	AX	EAX
001	CL	CX	ECX
010	DL	DX	EDX
011	BL	BX	EBX
100	AH	SP	ESP
101	CH	BP	EBP
110	DH	SI	ESI
111	BH	DI	EDI

R/M Code	Function
000	DS:BX+SI
001	DS:BX+DI
010	SS:BP+SI
011	SS:BP+DI
100	DS:SI
101	DS:DI
110	SS:BP
111	DS:BX

Direct Addressing Mode

- MOD is always 00
- R/M is always 110
- REG encodes the register to/from we take data as usual
- Third byte contains the lower-order bytes of the displacement, fourth byte contains the high order byte of the displacement

Direct Addressing

- Example: 8816 00 10
- 1000 1000 0001 0110 0000 0000 0001 0000
- Opcode 100010 -> MOV
- W=0 (byte-sized data)
- D=0 data flows from REG
- MOD 00, REG=010 (DL), R/M=110
- Low-order byte of displacement 00
- High-order byte of displacement 10
- MOV [1000h], DL

Code	W=0	W=1	W=1
000	AL	AX	EAX
001	CL	CX	ECX
010	DL	DX	EDX
011	BL	BX	EBX
100	AH	SP	ESP
101	CH	BP	EBP
110	DH	SI	ESI
111	BH	DI	EDI

Segment MOV instructions

- Different opcode 100011
- Segments are selected by setting the REG field

REG Code	Segment reg.
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS

Example MOV BX, CS

Opcode 10001100

MOD=11 (register addressing)

REG=001 (CS)

R/M=011 (BX)

8CCB

Mã máy

REG xác định thanh ghi cho toán hạng thứ nhất

REG	W = 0	W = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Mã máy

MOD và R/M cùng nhau xác định toán hạng thứ hai

CODE	EXPLANATION
00	Memory Mode, no displacement follows*
01	Memory Mode, 8-bit displacement follows
10	Memory Mode, 16-bit displacement follows
11	Register Mode (no displacement)

*Except when R/M = 110, then 16-bit displacement follows

(a)

Mã máy

MOD và R/M cùng nhau xác định toán hạng thứ hai

MOD = 11			EFFECTIVE ADDRESS CALCULATION			
R/M	W = 0	W = 1	R/M	MOD = 00	MOD = 01	MOD = 10
000	AL	AX	000	(BX) + (SI)	(BX) + (SI) + D8	(BX) + (SI) + D16
001	CL	CX	001	(BX) + (DI)	(BX) + (DI) + D8	(BX) + (DI) + D16
010	DL	DX	010	(BP) + (SI)	(BP) + (SI) + D8	(BP) + (SI) + D16
011	BL	BX	011	(BP) + (DI)	(BP) + (DI) + D8	(BP) + (DI) + D16
100	AH	SP	100	(SI)	(SI) + D8	(SI) + D16
101	CH	BP	101	(DI)	(DI) + D8	(DI) + D16
110	DH	SI	110	DIRECT ADDRESS	(BP) + D8	(BP) + D16
111	BH	DI	111	(BX)	(BX) + D8	(BX) + D16

(b)

Ví dụ

Mã hoá lệnh MOV BL,AL

- Opcode đối với MOV là 100010
- Ta mã hoá AL sao cho AL là toán hạng nguồn:
  - D = 0 (AL là toán hạng nguồn)
- W bit = 0 (8-bit)
- MOD = 11 (register mode)
- REG = 000 (mã của AL)
- R/M = 011 (mã của BL)

Kết quả:: 10001000 11000011 = 88 C3

Nhóm lệnh Số học

- Bên cạnh tác dụng, cần chú ý đến ảnh hưởng của lệnh đối với các cờ trạng thái
- Các lệnh số học th/thường: ADD, SUB, …
- Các lệnh số học khác: CMP. NEG, INC, DEC, …
- Ảnh hưởng đến các cờ trạng thái
  - CF
  - OF Phụ thuộc vào quá trình thực hiện phép toán
  - AF

  - ZF = 1 nếu Kết quả bằng 0
  - SF = 1 nếu MSB của Kết quả = 1
  - PF = 1 nếu byte thấp của kết quả có Parity chẵn

Arithmetic Instructions - ADD

Khuôn dạng: ADD Đích,Nguồn

- Tác dụng: (Đích) ← (Đích)+(Nguồn)
- Đích: có thể là:

1. Một thanh ghi 8 hoặc 16 bit của VXL

2. Một vị trí nhớ (1 hoặc 2 ô nhớ liên tiếp nhau)

- Nguồn: có thể là:

1. Một thanh ghi 8 hoặc 16 bit của VXL

2. Một vị trí nhớ (1 hoặc 2 ô nhớ liên tiếp nhau)

3. Một giá trị cụ thể

Ảnh hưởng của ADD

- ZF = 1 nếu Kết quả bằng 0
- SF = 1 nếu MSB của Kết quả = 1
- PF = 1 nếu byte thấp của kết quả có Parity chẵn

- CF được lập nếu tràn không dấu (có nhớ từ MSB)
- OF được lập nếu tràn có dấu:
  - Có nhớ từ MSB, Không có nhớ vào MSB
  - Có nhớ vào MSB, Không có nhớ từ MSB
- AF được lập nếu có nhớ từ nibble thấp vào nibble cao (từ bit 3 vào bit 4)

Các cờ trên thanh ghi cờ

- Các bit nhất định trên thanh ghi cờ điều khiển hoạt động hoặc phản ánh trạng thái của vi xử lý
  - Các cờ điều khiển (TF, IF, DF)
    - Quyết định cách đáp ứng của vi xử lý trong các tình huống nhất định
  - Các cở trạng thái (CF, PF, AF, ZF, SF, OF)
    - Bị ảnh hưởng bởi các phép toán nhất định
    - Phục vụ cho các lệnh có điều kiện

Các cờ điều khiển

- DF - Direction flag (Cờ hướng)
  - DF = 1: hướng xuống
  - DF = 0: hướng lên
- IF – Interrupt flag (Cờ ngắt)
  - IF = 1: cho phép ngắt ngoài
  - IF = 0: cấm ngắt ngoài (đối với ngắt che được)
- TF - Trace flag
  - TF = 1: vi xử lý thực hiện từng lệnh một

Các cờ trạng thái

- Carry
  - carry or borrow at MSB in add or subtract
  - last bit shifted out
- Parity
  - low byte of result has even parity
- Auxiliary
  - carry or borrow at bit 3
- Zero
  - result is 0
- Sign
  - result is negative
- Overflow
  - signed overflow occurred during add or subtract

(Signed) Overflow

- Can only occur when adding numbers of the same sign (subtracting with different signs)
- Detected when carry into MSB is not equal to carry out of MSB
  - Easily detected because this implies the result has a different sign than the sign of the operands
- Programs can ignore the Flags!

Signed Overflow Example

```
  10010110
+ 10100011
  00111001
```

Carry in = 0, Carry out = 1

Neg+Neg=Pos

Signed overflow occurred

OF = 1 (set)

```
  00110110
+ 01100011
  10011001
```

Carry in = 1, Carry out = 0

Pos+Pos=Neg

Signed overflow occurred

OF = 1 (set)

Examples of No Signed Overflow

```
  10010110
+ 01100011
  11111001
```

Carry in = 0, Carry out = 0

Neg+Pos=Neg

No Signed overflow occurred

OF = 0 (clear)

```
  10010110
+ 11110011
  10001001
```

Carry in = 1, Carry out = 1

Neg+Neg=Neg

No Signed overflow occurred

OF = 0 (clear)

Unsigned Overflow

- The carry flag is used to indicate if an unsigned operation overflowed
- The processor only adds or subtracts - it does not care if the data is signed or unsigned!

```
   10010110
+  11110011
   10001001
```

Carry out = 1

Unsigned overflow occurred

CF = 1 (set)

DEBUG's Register Display

```
-R
…000 SP=0010 BP=0000 SI=0000 DI=0000
…00F IP=004F NV UP DI PL NZ NA PO NC
```

- The state of the Flags are shown in line 2
- **OV/NV: (no)oVerflow DN/UP: direction**
- **EI/DI: En(Dis)abled Interrupts**
- **NG/PL: sign ZR/NZ: (not)Zero**
- **AC/NA: (no)Auxiliary PE/PO: Even/Odd**
- **CY/NC: (no)Carry (*set/clear*)**

Arithmetic Instructions - SUB

Khuôn dạng: SUB Đích,Nguồn

- Tác dụng: (Đích) ← (Đích)-(Nguồn)
- Đích: có thể là:

1. *Một thanh ghi 8 hoặc 16 bit của VXL*
2. *Một vị trí nhớ (1 hoặc 2 ô nhớ liên tiếp nhau)*

- Nguồn: có thể là:

1. *Một thanh ghi 8 hoặc 16 bit của VXL*
2. *Một vị trí nhớ (1 hoặc 2 ô nhớ liên tiếp nhau)*
3. *Một giá trị cụ thể*

Ảnh hưởng của SUB

- ZF = 1 nếu Kết quả bằng 0
- SF = 1 nếu MSB của Kết quả = 1
- PF = 1 nếu byte thấp của kết quả có Parity chẵn

- CF được lập nếu tràn không dấu (có mượn vào MSB)
- OF được lập nếu tràn có dấu:
  - Có mượn từ MSB, Không có mượn từ MSB
  - Có mượn từ MSB, Không có mượn vào MSB
- AF được lập nếu có mượn từ nibble cao vào nibble thấp (từ bit 4 vào bit 3)

Arithmetic Instructions - CMP

Khuôn dạng: CMP Đích,Nguồn

- Tác dụng: (Đích)-(Nguồn)
- Đích: có thể là:

1. Một thanh ghi 8 hoặc 16 bit của VXL

2. Một vị trí nhớ (1 hoặc 2 ô nhớ liên tiếp nhau)

- Nguồn: có thể là:

1. Một thanh ghi 8 hoặc 16 bit của VXL

2. Một vị trí nhớ (1 hoặc 2 ô nhớ liên tiếp nhau)

3. Một giá trị cụ thể

Arithmetic Instructions – INC, DEC, NEG

- INC T/h
- Trong đó: T/h có thể là các thanh ghi hoặc vị trí nhớ
- Tác dụng: (T/h) ← (T/h)+1

- DEC T/h
- Trong đó: T/h có thể là các thanh ghi hoặc vị trí nhớ
- Tác dụng: (T/h) ← (T/h)-1
- Lưu ý: Các lệnh INC và DEC không ảnh hưởng đến cờ CF
- Lệnh NEG T/h: Đảo dấu của T/h (Lấy bù 2)
- *Lệnh NEG sẽ lập cờ OF nếu giá trị của T/h là giá trị âm nhất trong dải giá trị của các số có dấu tương ứng*

Nhóm lệnh Logic

- Cần chú ý đến ảnh hưởng của lệnh đối với các cờ trạng thái
- Các lệnh logic th/thường: NOT, AND, OR, XOR

  NOT A: ~A

AND A,B: A &= B

OR A,B : A |= B

XOR A,B: A ^= B

- NOT không ảnh hưởng đến các cờ trạng thái.
- Các lệnh khác:
  - CF = 0
  - OF = 0
  - ZF = 1 nếu Kết quả bằng 0
  - SF = 1 nếu MSB của Kết quả = 1
  - PF = 1 nếu byte thấp của kết quả có Parity chẵn
  - AF không xác định

Một số ví dụ

AL	1100 1010
NOT AL	
AL	0011 0101

AL	0011 0101
BL	0110 1101
AND AL, BL	
AL	0010 0101

AL	0011 0101
BL	0000 1111
AND AL, BL	
AL	0000 0101

AL	0011 0101
BL	0110 1101
OR AL, BL	
AL	0111 1101

AL	0011 0101
BL	0000 1111
OR AL, BL	
AL	0011 1111

AL	0011 0101
BL	0110 1101
XOR AL, BL	
AL	0101 1000

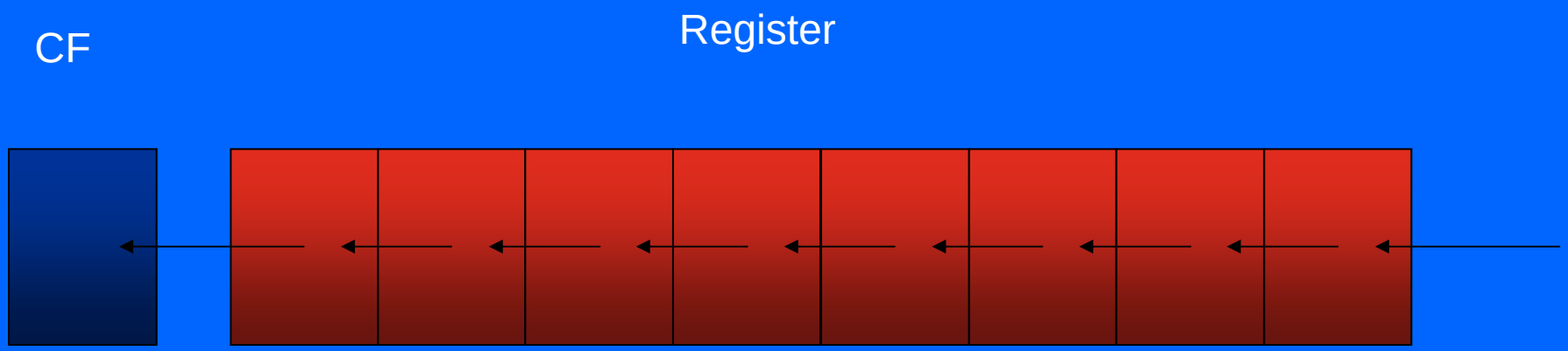
Một số ứng dụng

- Bài toán Xoá bit: Xoá một bit nào đó của một toán hạng mà không làm ảnh hưởng đến các bit còn lại của toán hạng đó
- Bài toán Kiểm tra bit: Xác định một bit nào đó của một toán hạng là bằng 0 hay 1 (thông qua giá trị của một cờ trạng thái)
- Bài toán Lập bit: Lập một bit nào đó của một toán hạng mà không làm ảnh hưởng đến các bit còn lại của toán hạng đó

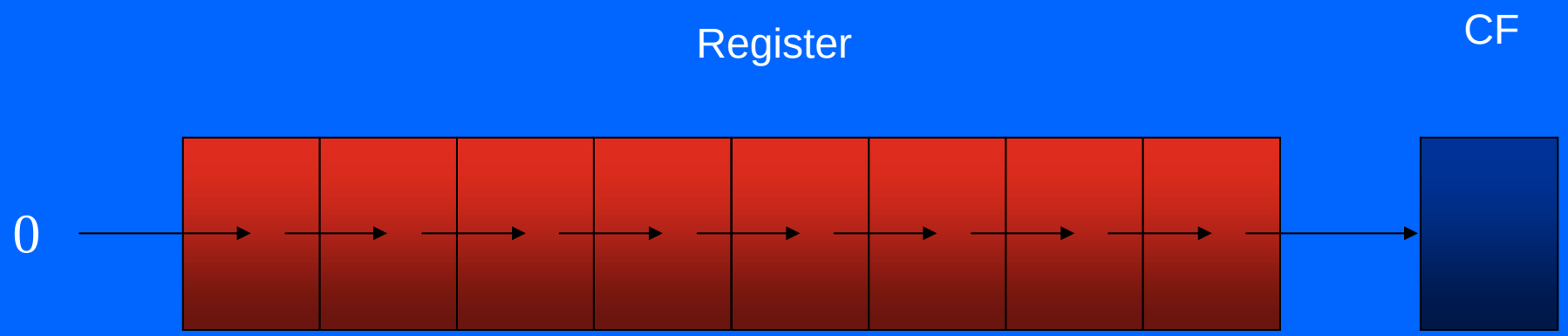
Nhóm lệnh logic

- Các lệnh logic khác: Lệnh TEST, Các lệnh dịch (Shift) và Các lệnh quay (Rotate)
- Lệnh TEST chỉ khác lệnh AND là không giữ lại kết quả của phép toán
- Các lệnh dịch và Các lệnh quay đều có hai khuôn dạng:
  Khuôn dạng 1: Mnemonic Toán hạng,1
  Khuôn dạng 2: Mnưmonic Toán hạng,CL
- Tác dụng của một câu lệnh theo khuôn dang 2 giống như tác dụng liên tiếp của N câu lệnh tương ứng theo khuôn dạng 1, với N là giá trị của thanh ghi CL

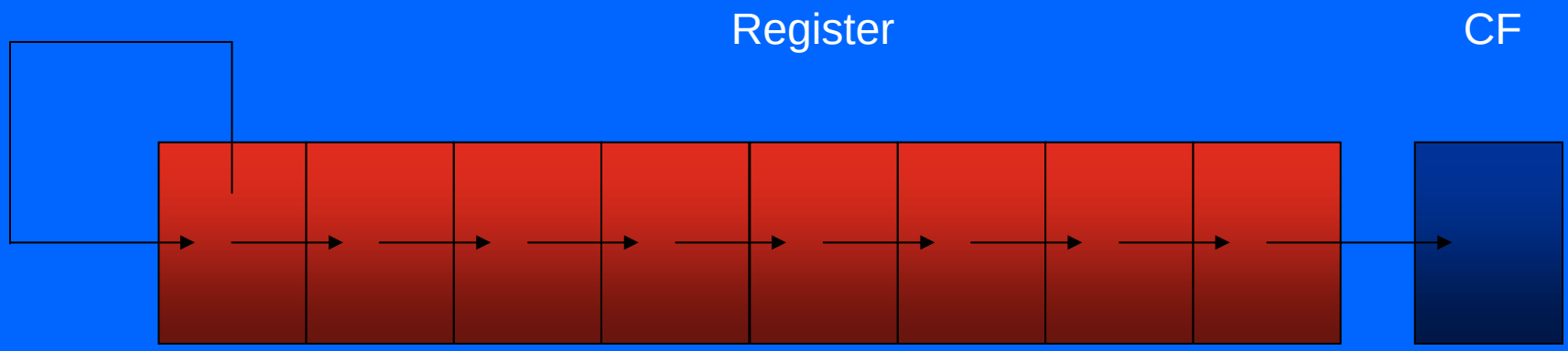
Các lệnh Dịch trái: SHL, SAL



Shift right SHR

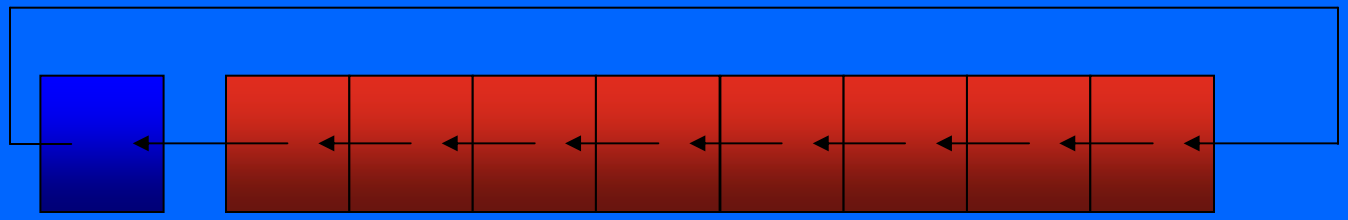


Shift right SAR

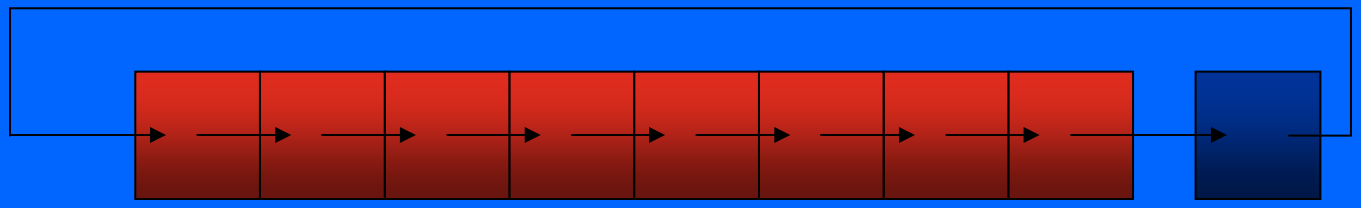


Rotate through Carry L/R
(Quay trái/phải thông qua carry)

RCL

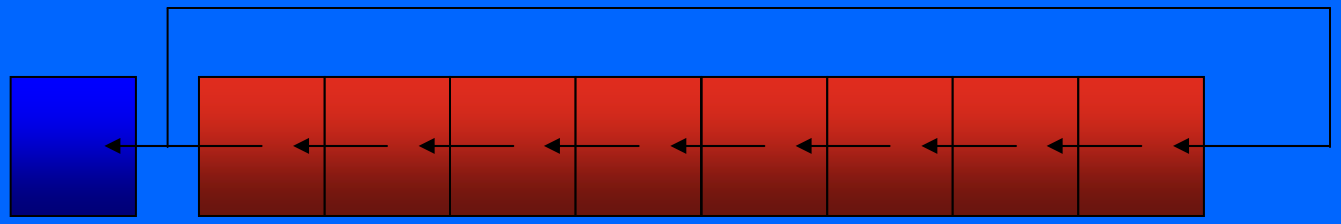


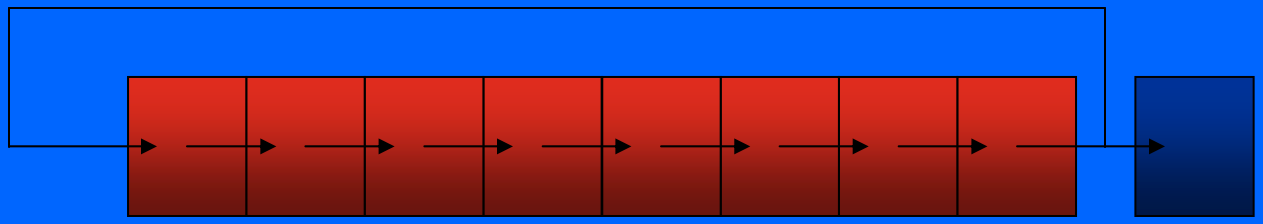
RCR



Rotate left/right
(Quay trái/phải không qua carry)

ROL





ROR

Nhóm lệnh rẽ nhánh

- Làm thay đổi trật tự thực hiện lệnh bình thường của vi xử lý
- *Lệnh nhảy không điều kiện: JMP*
- *Các lệnh nhảy có điều kiện: Jxxx*
- *Lệnh lặp: LOOP và các biến thể của nó*
- *Các lệnh có liên quan đến Chương trình con:*
  - CALL (gọi chương trình con)
  - RET (trở về chương trình gọi)
- *Các lệnh có liên quan đến Chương trình con phục vụ ngắt*
  - INT (gọi chương trình con phục vụ ngắt - Gọi ngắt)
  - IRET (quay về chương trình gọi ngắt)

Lệnh nhảy không điều kiện

- JMP *nhãn*
  - Nhảy gần: E9 xx xx (3 byte)
  - Nhảy ngắn: EB xx (2 byte)
  - Nhảy xa: EA xx xx xx xx (5 byte)
- *Nhãn: tên do ngườI lập trình tự đặt ra theo qui tắc đặt tên của Assembler và có thể đặt vào trước một câu lệnh bất kỳ trong chương trình cùng với dấu :*

  nhãn: Câu lệnh cần thực hiện

- *Nhãn sẽ được dịch thành địa chỉ*
- *Khoảng cách nhảy: Khoảng cách đại số (có dấu) từ lệnh nhảy đến lệnh cần thực hiện*

Cơ chế thực hiện lệnh nhảy

- Các lệnh nhảy ngắn và gần chỉ làm thay đổi giá trị của thanh ghi IP
  - Lệnh nhảy ngắn cộng khoảng cách nhảy 8-bit có dấu vào giá trị hiện thời của IP
  - Lệnh nhảy gần cộng khoảng cách nhảy 16-bit có dấu vào giá trị hiện thời của IP
- Lệnh nhảy xa làm thay đổi cả CS và IP
  - Gán cho CS và IP các giá trị mới

Mã máy của lệnh nhảy

1106:0100 EB2A JMP 012C

- **012C-0102=002A**

1106:0102 EBFC JMP 0100

- **0100-0104=FFFC**

1106:0104 E97F00 JMP 0186

- **0186-0106=0080 (too far for short!)**
- **0186-0107=007F**

1106:0107 E9F5FE JMP FFFF

- **FFFF-010A=FEF5**

Các lệnh nhảy có điều kiện

- Jxxx *nhãn*
  - Có gần 40 menmonic khác nhau
- Các lệnh nhảy điều kiện đơn: phụ thuộc vào giá trị của 1 cờ.
- JNZ/JNE - Nhảy nếu cờ ZF = 0, nghĩa là kết quả của phép toán trước đó khác không
- JC - Nhảy nếu CF = 1, nghĩa là câu lệnh trước đó lập cờ carry
- JZ/JE
- JNC

Các lệnh nhảy có điều kiện

- Tất cả các lệnh nhảy có điều kiện phải là nhảy ngắn
  - khoảng cách nhảy: -128 to +127 bytes
- Tổ hợp với lệnh nhảy không điều kiện để có thể vượt qua giới hạn này.
- Các lệnh nhảy điều kiện kép: phụ thuộc vào giá trị của nhiều cờ
- JB/JNAE
- JNL/JGE

ứng dụng của các lệnh nhảy có điều kiện

- Kết hợp với JMP để xây dựng các cấu trúc lập trình cơ bản:

  - Cấu trúc điều kiện
  - Cấu trúc lặp

- Các lệnh nhảy thường theo sau các lệnh làm thay đổi giá trị của các cờ trạng thái:
  - CMP
  - TEST …

Cấu trúc điều kiện

```
mov  ax,n
cmp ax,7
jz nhan1
lệnh 1
jmp nhan2
nhan1:lệnh 2
nhan2:lệnh 3
```

Cấu trúc lặp

```
    mov ax,n

  nhan1: cmp ax,0
    jz nhan2
    lệnhi
    sub ax,2
    jmp nhan1

nhan2: lệnhk
```

Cấu trúc điều kiện - AND

```
char n; int w,x;
if (n>='A' && w==x)
    whatever();
```

```
;if(n>='A'&&w==x)
   mov ah,n
   cmp ah,'A'
   jl nogo
   mov ax,w
   cmp ax,x
   jne no_go
;then-part
   call whatever
nogo:
```

Cấu trúc điều kiện - OR

```
char n,k; unsigned int w;
if (n<>k || w<=10)
   whatever();
```

```
;if(n<>k||w<=10)
   mov ah,n
   cmp ah,k
   jne then_
   cmp w,10
   ja end_if
then_:
   call whatever
end_if:
```

Lệnh LOOP

- LOOP *nhan*
  - Giảm CX đi 1
  - Nếu (CX) <> 0 thì JMP nhan. Nếu không thì tiếp tục thực hiện lệnh theo trật tự bình thường

```
        mov cx,9
nhan:   lệnh 1
        lệnh 2
        lệnh 3
        loop nhan
```

LOOPZ/E và LOOPNZ/E

- Các biến thể của LOOP
- Giá trị của cờ ZF có thể làm kết thúc sớm vòng lặp
- Loop while ZF/equal && CX!=0
- Loop while (NZ/ not equal) && CX!=0
- Lưu ý: LOOP giảm CX nhưng không ảnh hưởng đến các cờ
- LOOPZ == LOOPE
- LOOPNZ==LOOPNE
- Các lệnh trong vòng lặp có thể tác động đến cờ ZF (CMP ?)

Chương trình con

- Chương trình con trong ngôn ngữ Assembly được gọi là Thủ tục (Procedure)
- Một thủ tục có thể được thực hiện nhiều lần
- Có liên quan đến stack:
  - lưu giữ Địa chỉ quay về
  - lưu giữ giá trị của các thanh ghi của vi xử lý

Stack ?

- Cấu trúc dữ liệu LIFO ở RWM
  - PUSH : ghi dữ liệu vào stack,
  - POP: đọc dữ liệu từ stack
- (SS:SP) trỏ đến đỉnh của stack
- (SS:BP) truy cập stack ngẫu nhiên (không theo LIFO)

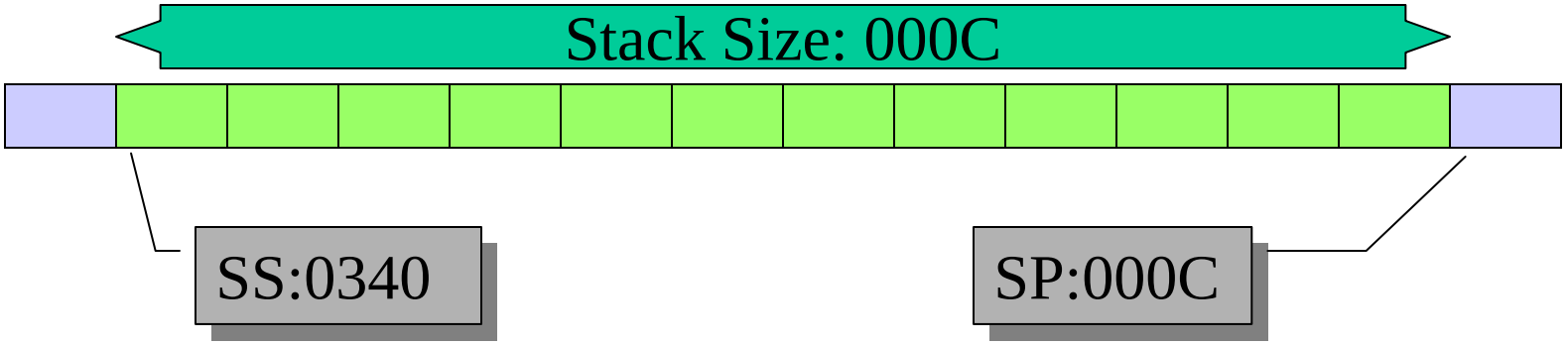
Stack Initialization

- The .stack directive hides an array allocation statement that looks like this
  - **`The_Stack  DB  Stack_Size dup (?)`**
- On program load…
  - SS is set to a segment address containing this array (usually **`The_Stack`** starts at offset 0)
  - SP is set to the offset of **`The_Stack+Stack_Size`** which is one byte past the end of the stack array
    - This is the condition for an empty stack

Initial Stack Configuration

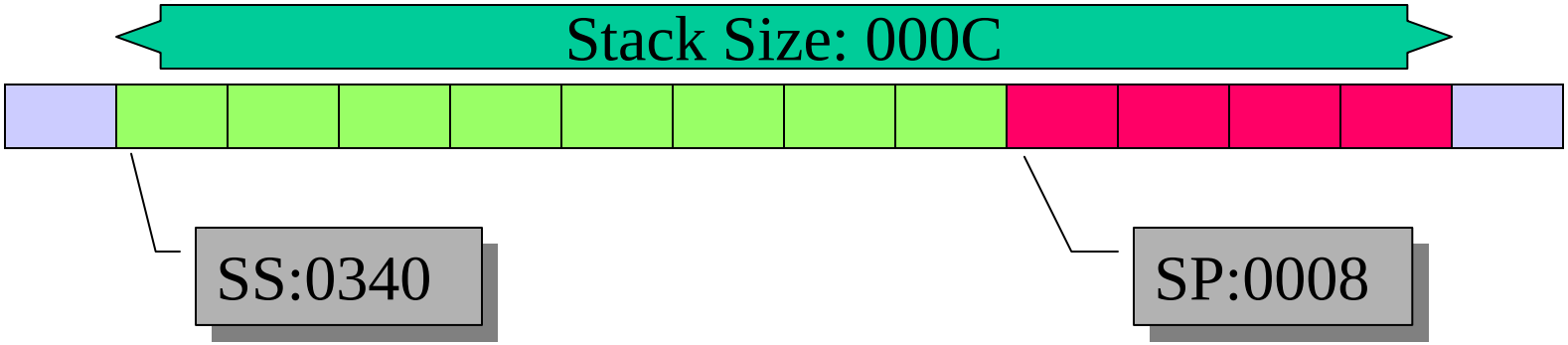
.stack 12 ;Reserve space for the stack

- Loader determines actual segment address for the start of the stack
  - This is an empty stack



How Does The Stack Work?

- The stack grows backwards through memory towards the start of the stack segment

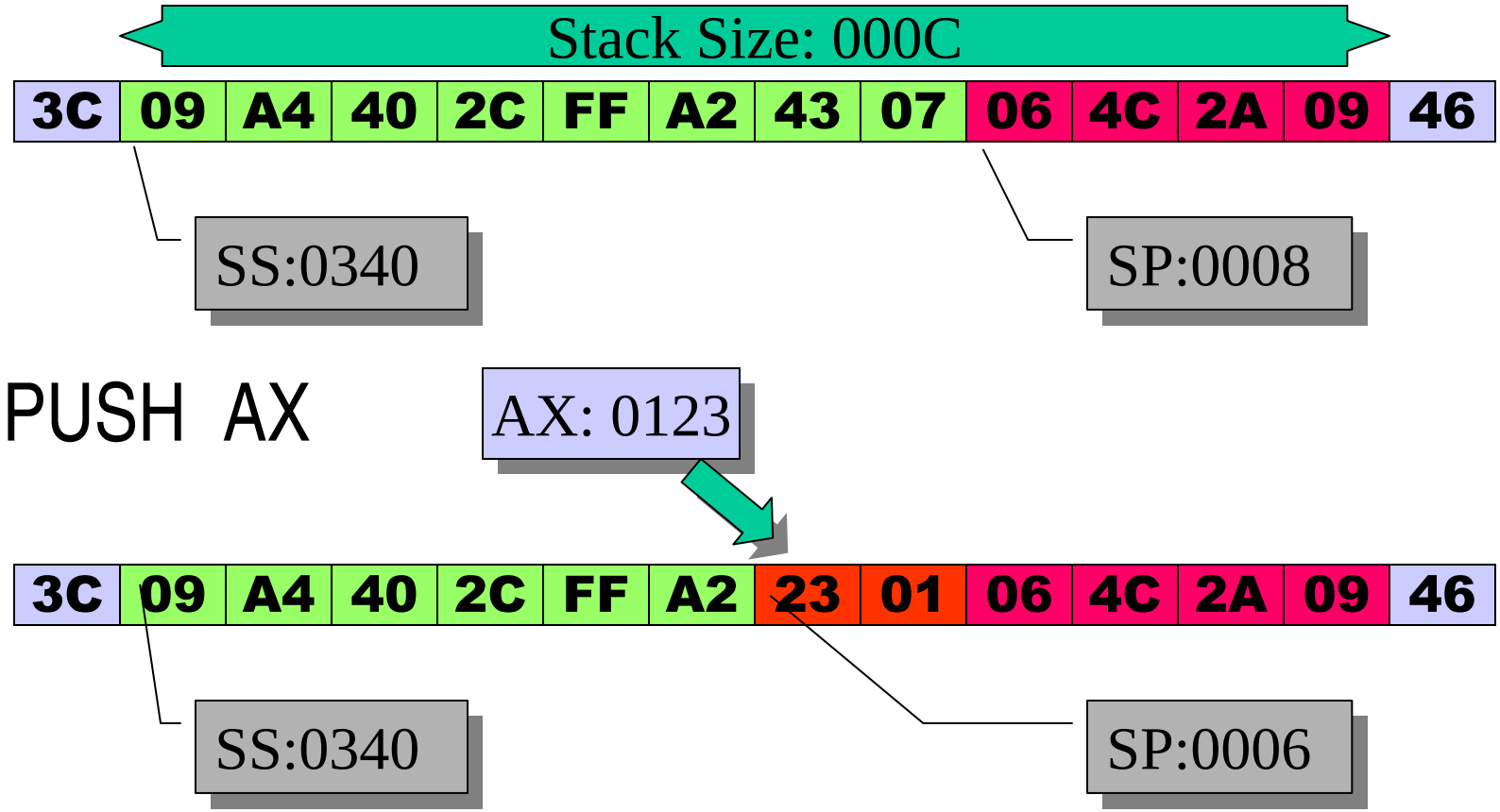


- Push decrements stack pointer
  Pop increments stack pointer

PUSH

- PUSH *nguồn*
  - Push nguồn vào stack
- PUSHF
  - Push thanh ghi cờ vào stack
- Lệnh PUSH trước hết sẽ giảm SP đi 2 rồi lưu giá trị của nguồn vào vị trị nhớ được trỏ bởi (SS:SP)

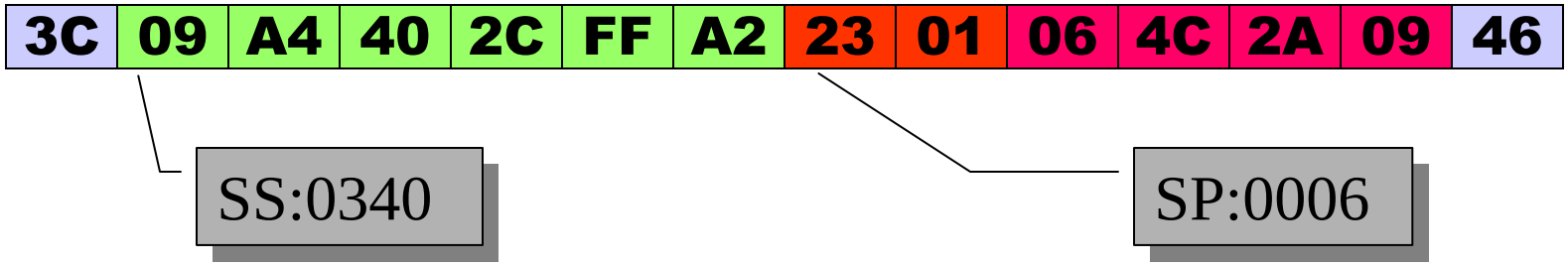
Ví dụ PUSH


Stack Size: 000C
3C 09 A4 40 2C FF A2 43 07 06 4C 2A 09 46
SS:0340
SP:0008
PUSH AX
AX: 0123
3C 09 A4 40 2C FF A2 23 01 06 4C 2A 09 46
SS:0340
SP:0006

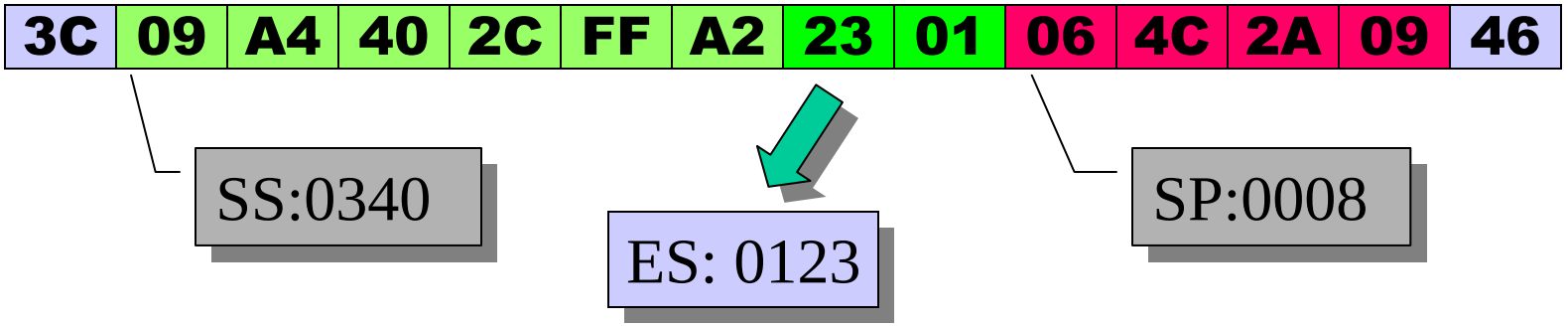
POP

- POP *đích*
  - Pop dữ liệu từ đỉnh stack vào đich
- POPF
  - Pop dữ liệu từ đỉnh stack vào thanh ghi cờ
- Lệnh POP trước hết copy dữ liệu được trỏ bởi (SS:SP) đến đích rồi tăng SP lên 2

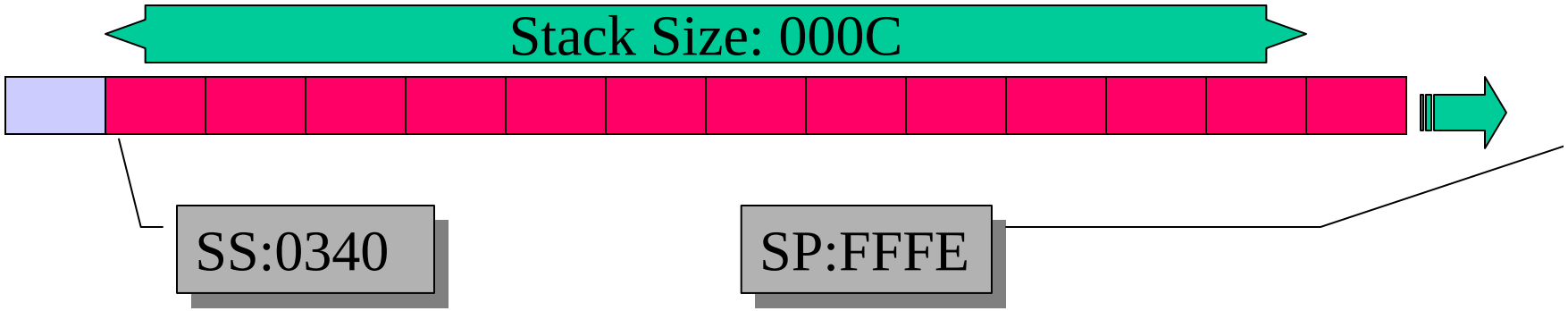
Ví dụ POP



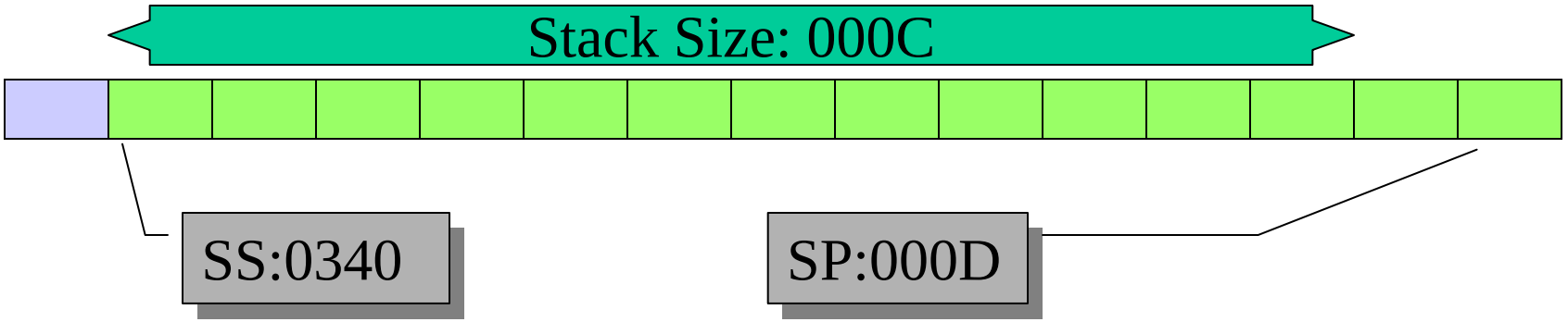
POP ES



Tràn stack!



- Stack Overflow
- Stack Underflow



Thủ tục

```
Tên_Thủ_tục  PROC     kiểu
   ;thân của thủ tục
   RET    ;quay về chương trình gọi
Tên_Thủ_tục  ENDP
```

- *kiểu* là NEAR hoặc FAR
  - ngầm định là NEAR
- Một thủ tục có thể có nhiều lệnh RET

Lệnh CALL và RET

- Gọi một thủ tục (NEAR)

  CALL *Tên_Thủ_tục*

  – push IP vào stack

  – copy địa chỉ của Tên_Thủ_tục vào IP
- Trở về từ một thủ tục (NEAR)

  RET

  – pop giá trị ở đỉnh stack vào IP

Thủ tục Far

- Gọi thủ tục (FAR)

  CALL *Tên_thủ_tục*

  – lần lượt push CS và IP vào stack

  – copy địa chỉ của Tên_thủ_tục vào CS và IP
- Trở về từ thủ tục (FAR)

  RET

  – pop giá trị từ đỉnh stack lần lượt vào IP và CS

Gọi ngắt

- Gọi ngắt là một lời gọi thủ tục đặc biêt
  - FAR
  - Thanh ghi cờ phải được bảo toàn
- INT Số ngắt
  - Thanh ghi cờ được push, TF và IF bị xoá
  - CS và rồI IP được push
  - Địa chỉ của một chương trình con phục vụ ngắt (Vector ngắt) tương ứng với Số ngắt được copy vào CS và IP

Trở về từ ngắt

- IRET
- Tác dụng của lênh:
  - Giá trị ở đỉnh của stack được pop vào IP
  - Giá trị ở đỉnh của stack được pop vào CS
  - Giá trị ở đỉnh của stack được pop vào thanh ghi cờ

- Chương trình bị ngắt tiếp tục thực hiện dường như không có chuyện gì xảy ra

Xuất ký tự ra màn hình PC

- Ngắt 21h
  - Ngắt này hỗ trợ rất nhiều dịch vụ trên PC
  - Nhận dạng dịch vụ bằng số dịch vụ (số hàm). Số dịch vụ cần được nạp voà thanh ghi AH
  - Tuỳ theo từng dịch vụ, có thể cần thêm một số đối số khác được nạp vào các thanh ghi xác định
- AH = 2, DL = Mã ASCII của ký tự cần xuất
  - Ký tự được hiển thị tại vị trí hiện thờI của con trỏ

Xuất xâu ký tự ra màn hình PC

- Dịch vụ 09h của ngắt 21h
  - DX = Địa chỉ Offset của xâu (trong đoạn dữ liệu)
  - DS = Địa chỉ segment của xâu
  - Xâu ký tự phải kết thúc bằng ký tự '$'
- Để nạp địa chỉ offset của xâu vào DX, có thể:
  - LEA DX, Tênxâu
  - MOV DX, OFFSET Tên xâu

Nhập 1 ký tự từ bàn phím PC

- Dịch vụ 01h của ngắt 21h
- Khi NSD gõ một ký tự từ bàn phím:
  - Ký tự sẽ hiện trên màn hình
  - AL sẽ chứa mã ASCII của ký tự đó
    - AL=0 nếu ký tự được nhập là ký tự điều khiển

Nhóm lệnh thao tác string

- Chúng ta hiểu: string là một mảng byte hoặc từ nằm trong bộ nhớ
- Các thao tác string:
  - Sao chép
  - Tìm kiếm
  - Lưu trữ
  - So sánh

Các đặc điểm

- Nguồn: (DS:SI), Đích: (ES:DI)
  - DS, ES chứa Địa chỉ Segment của string
  - SI, DI chứa Địa chỉ Offset của string
- Cờ hướng DF (0 = Up, 1 = Down)
  - DF = 0 - Tăng địa chỉ (trái qua phải)
  - DF = 1 - Giảm địa chỉ (phảI qua trái)

Chuyển (Sao chép)

- MOVSB, MOVSW
  - Chuyển 1 byte hoặc 1 word từ vị trí nhớ này sang vị trí nhớ khác
  - Tác dụng của lệnh:
    - Sao chép byte/word từ (DS:SI) đến (ES:DI)
    - Tăng/Giảm SI và DI 1 hoặc 2 giá trị
  - Nếu CX chứa một giá trị khác không:
    - REP MOVSB hoặc REP MOVSW sẽ tự động sao chép (CX) lần và CX sẽ về không

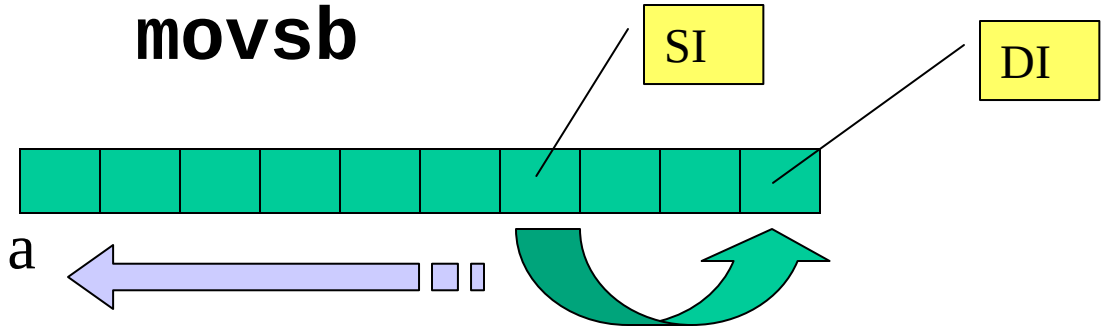
Ví dụ:Sao chép mảng

; Sao chép 10 byte từ mảng a sang mảng b, giả sử (DS) = (ES)

```
mov     cx, 10
mov     di, offset b
mov     si, offset a
cld     ;xoá cờ DF
rep     movsb
```

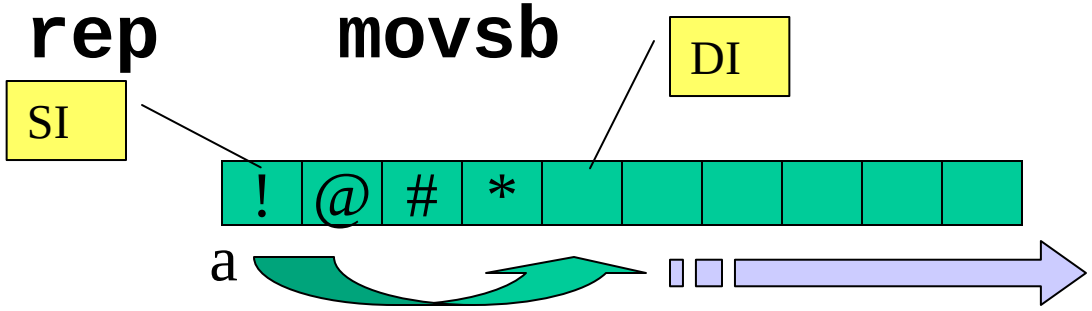
Ví dụ: Tịnh tiến các ô nhớ

```
mov     cx, 7
mov     di, offset a+9
mov     si, offset a+6
std     ;lập cờ DF
rep     movsb
```



Ví dụ

```
pattern     db   "!@#*"
            db   96 dup (?)
mov     cx,96
mov     si, offset pattern
mov     di, offset pattern+4
cld
rep     movsb
```



Lưu trữ string

STOSB, STOSW

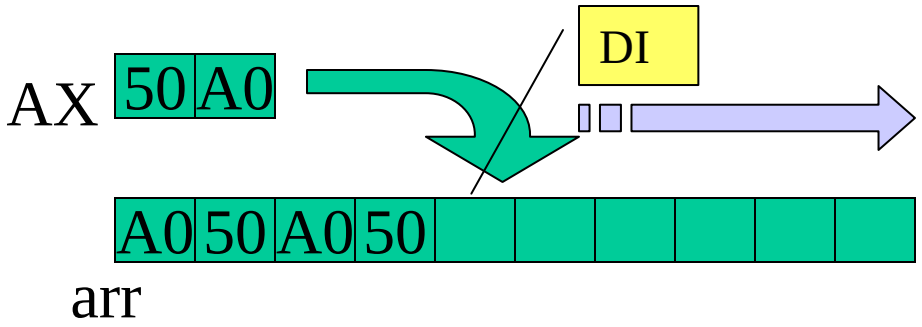
- Copy AL hoặc AX vào một mảng byte hoặc word
  - Đích (ES:DI)
- Tăng hoặc Giảm DI
  - phụ thuộc DF

- Thường được sử dụng có tiền tố REP và số lần lặp trong CX

Ví dụ:

```
arr  dw  200 dup (?)

 mov     ax,50A0h
 mov     di,offset arr
 mov     cx,200
 cld
  rep stosw
```



Nạp String

- LODSB, LODSW
  - Byte hoặc word tại (DS:SI) được copy vào AL hoặc AX
  - SI tăng hoặc giảm 1 hoặc 2 giá trị phụ thuộc DF
- Thường được dùng với STOSx trong một vòng lặp để xử lý từng phần tử trong một mảng

Ví dụ:

```
  mov di, offset b
  mov si, offset a
  mov cx,30
  cld
lp:
  lodsb
  and al,0DFh
  stosb
  loop    lp
```

Quét String

SCASB, SCASW

- So sánh AL hoặc AX vớI byte hoặc word tạI (ES:DI) và tự động tăng hoặc giảm DI
- Lệnh này ảnh hưởng đến các cờ trạng thái
  - Tuỳ theo kết quả so sánh
  - Dùng trong một vòng lặp REPs
    - REPZ, REPE, REPNZ, REPNE

Ví dụ

```
arr db 'abcdefghijklmnopqrstuvwxyz'
  mov         di, offset arr
  mov         cx,26
  cld
  mov         al,target
  repne  scasb

  jne         nomatch
```

So sánh String

CMPSB, CMPSW

- So sánh byte hoặc word tại (DS:SI) với byte hoặc word tạI (ES:DI), tác động đến các cờ và tăng hoặc giảm SI và DI
- Thường dùng để so sánh hai mảng với nhau

Ví dụ

```
  mov si, offset str1
  mov di, offset str2
  cld
  mov cx, 12
  repe    cmpsb
  jl      str1smaller
  jg      str2smaller
;the strings are equal - so far
;if sizes different, shorter string is
  less
```

Nhóm lệnh hỗn hợp

- Các lệnh Lập/Xoá trực tiếp các cờ:

STC, CLC
STD, CLD
STI, CLI

- Lệnh NOP (No Operation): Không làm gì!!!

- Lệnh NOP thường được dùng trong các vòng lặp tạo trễ (delay)bằng phần mềm

- Các lệnh Nhập/Xuất dữ liệu đối với các cổng I/O

IN
OUT

Lệnh IN

- Nếu Địa chỉ của cổng Nhỏ hơn hoặc bằng FFh:

IN Acc, Địa chỉ cổng

- Trong đó: Acc có thể là AL hoặc AX

- Nhập dữ liệu từ cổng vào Acc

- Nếu Địa chỉ của cổng Lớn hơn FFh:

MOV DX, Địa chỉ cổng
IN Acc, DX

- Trong đó: Acc có thể là AL hoặc AX

- Nhập dữ liệu từ cổng vào Acc

Lệnh OUT

- Nếu Địa chỉ của cổng Nhỏ hơn hoặc bằng FFh:

OUT Địa chỉ cổng, Acc

- Trong đó: Acc có thể là AL hoặc AX

- Xuất dữ liệu từ Acc ra cổng

- Nếu Địa chỉ của cổng Lớn hơn FFh:

MOV DX, Địa chỉ cổng
OUT DX, Acc

- Trong đó: Acc có thể là AL hoặc AX

- Xuất dữ liệu từ Acc ra cổng

Tóm tắt chương

- Tính tương thích về Cấu trúc thanh ghi của các vi xử lý họ x86

- Tính tương thích về Tập lệnh của các vi xử lý họ x86

Kỹ thuật Vi xử lý

Điện tử-Viễn thông

Đại học Bách khoa Đà Nẵng

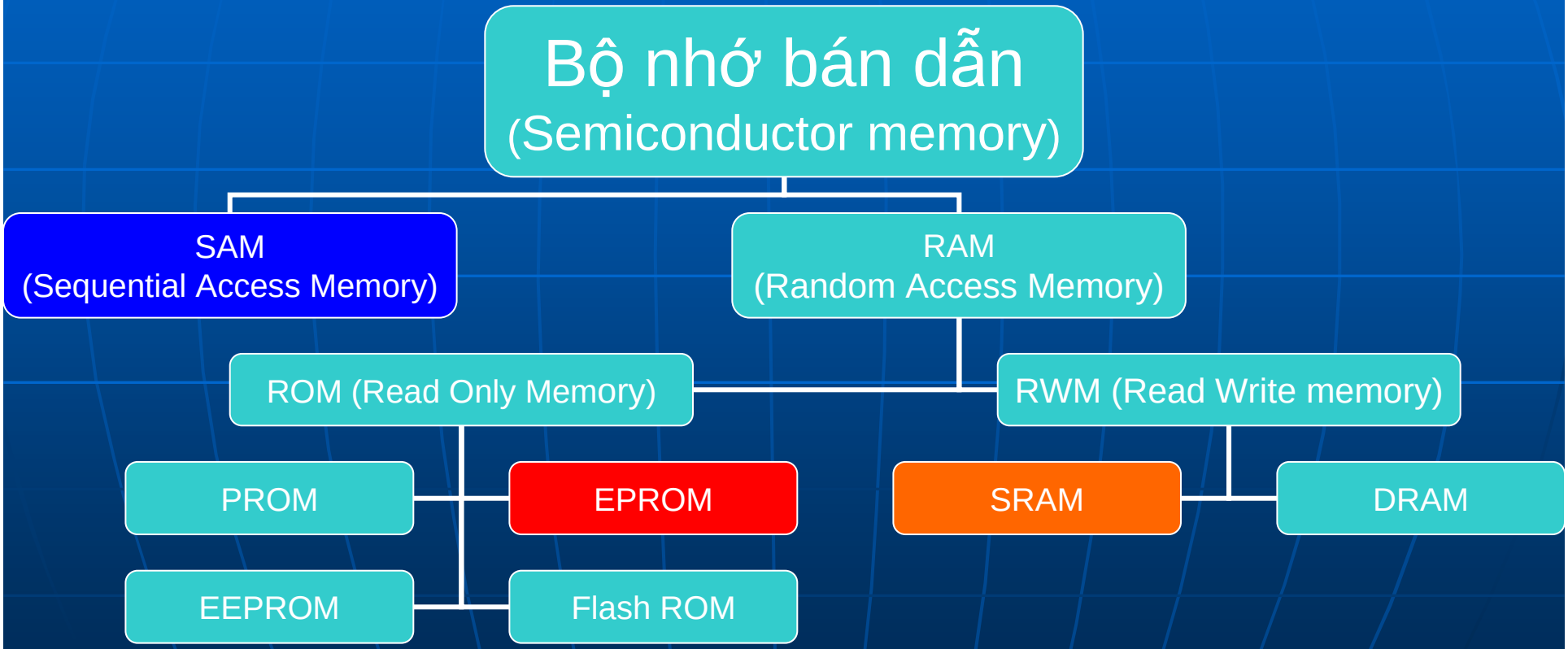
Chương 4

4.1 Phân loại bộ nhớ bán dẫn
4.2 Hoạt động của các chip EPROM
4.3 Hoạt động của các chip SRAM
4.4 Bus hệ thống của hệ vi xử lý 8088
4.5 Bài toán thiết kế bộ nhớ

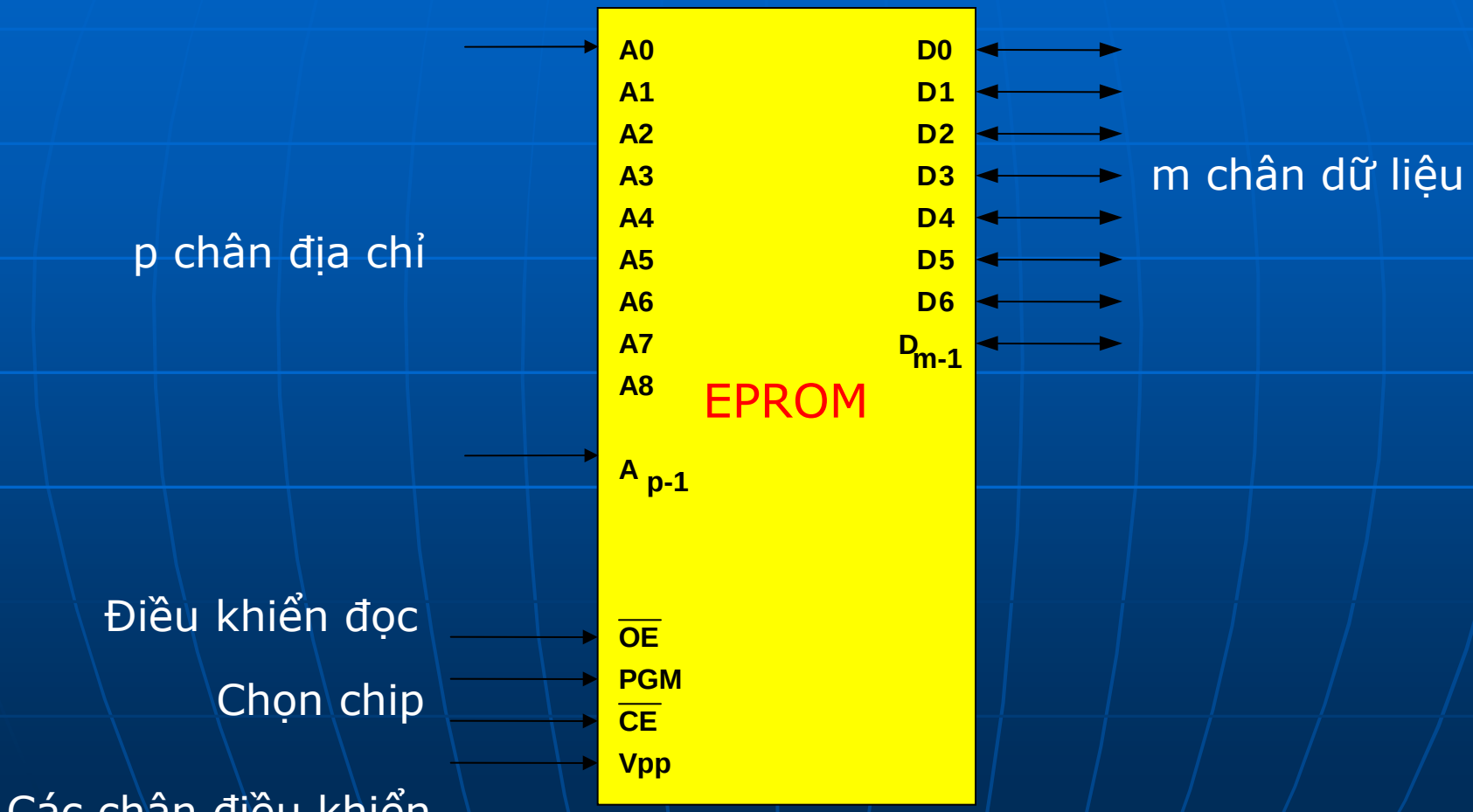
Mục tiêu và biện pháp thiết kế

- Ghép nối các chip nhớ EPROM và SRAM với Bus hệ thống sao cho không xảy ra xung đột:
  Các chip nhớ bị cấm khi vi xử lý truy cập các cổng I/O
  Chỉ có một chip nhớ hoạt động khi vi xử lý truy cập bộ nhớ
- Thực hiện một mạch giải mã địa chỉ bộ nhớ dùng các chip giải mã hoặc các cổng logic hoặc kết hợp cả hai

4.1 Phân loại bộ nhớ bán dẫn



4.2 Các chip EPROM



Dung lượng của 1 chip nhớ

- Một chip nhớ được xem như một mảng gồm n ô nhớ. Mỗi ô nhớ lưu trữ được m-bit dữ liệu
- Dung lượng của chip thường được biểu diễn: nxm
  Ví dụ: Một chip có dung lượng 2Kx8 nghĩa là chip đó có 2048 ô nhớ và mỗi ô nhớ có thể lưu trữ được 1 byte dữ liệu
- m chính là số chân dữ liệu của chip
- $\log_2(n) = p$ là số chân địa chỉ của chip

Hoạt động ghi dữ liệu vào EPROM

- Việc ghi dữ liệu vào EPROM được gọi là lập trình cho EPROM
- Được thực hiện bằng thiết bị chuyên dụng gọi là Bộ nạp EPROM
- Chân Vpp được cấp điện áp tương ứng với từng loại chip gọi là điện áp lập trình
- Dữ liệu tại các chân dữ liệu sẽ được ghi vào một ô nhớ xác định nhờ các tín hiệu đưa vào ở các chân địa chỉ và một xung (thường gọi là xung lập trình) đưa vào chân PGM

Hoạt động đọc dữ liệu từ một chip EPROM

Để đọc dữ liệu từ 1 ô nhớ nào đó của 1 chip EPROM nào đó, Bộ vi xử lý cần phải:

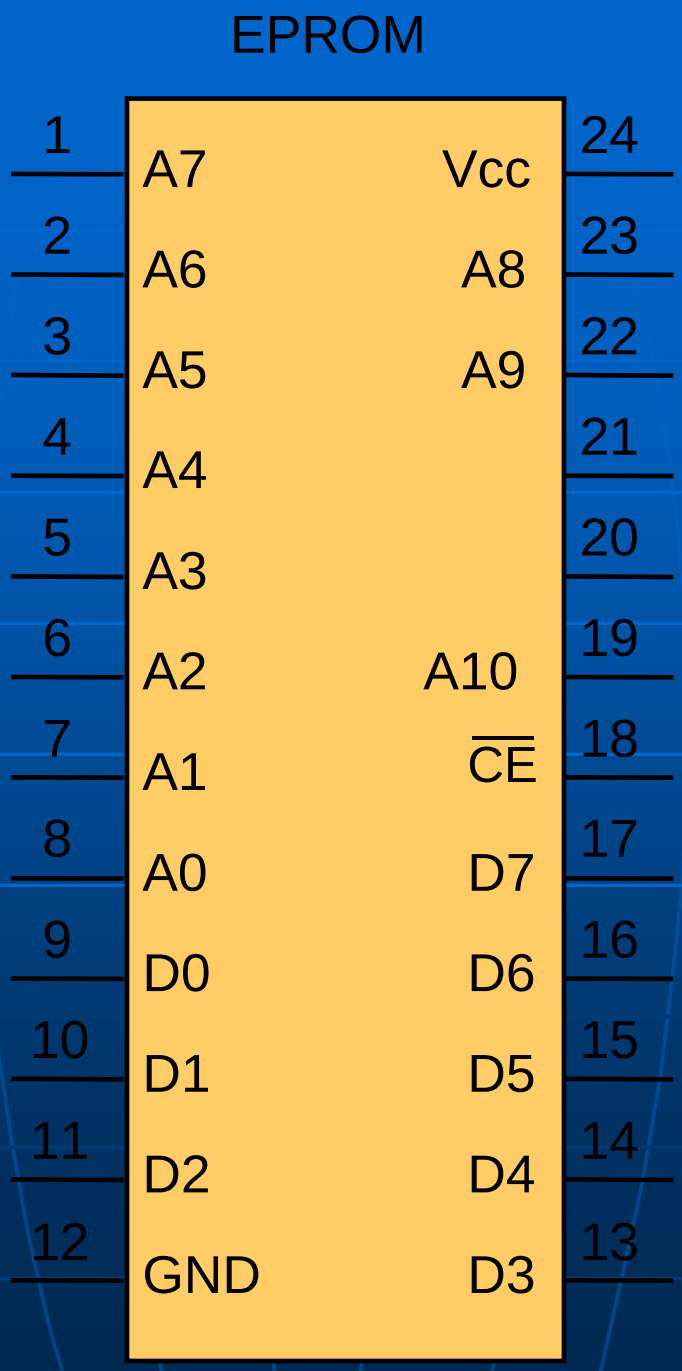
- Chọn chip đó: 0 -----> CE
- Áp các tín hiệu địa chỉ của ô nhớ cần đọc vào các chân địa chỉ $A_{p-1} – A_0$
- Đọc: 0 ------ > OE
- Kết quả là m bit dữ liệu cần đọc xuất hiện ở các chân dữ liệu $D_{m-1} – D_0$

Họ EPROM thông dụng 27x

Số hiệu của chip	Dung lượng
2716	2Kx8
2732	4Kx8
2764	8Kx8
27128	16Kx8
27256	32Kx8
27512	64Kx8

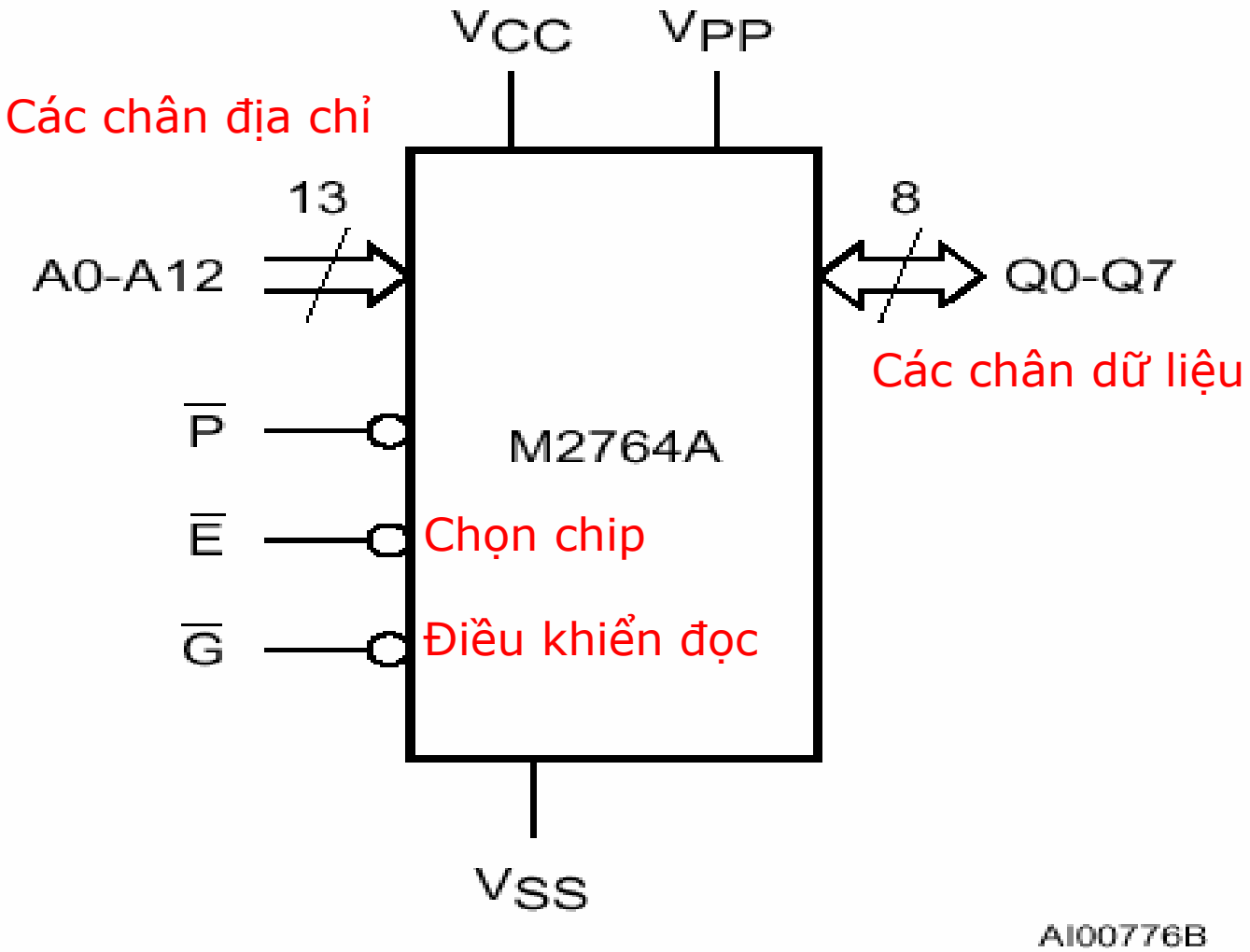
Bảng 4.1 Họ EPROM 27x

- Sơ đồ chân của 2716 và 2732



	2716	**2732**
21	Vpp	A11
20	$\overline{OE}$	$\overline{OE}$ / Vpp

EPROM 2764


VCC
VPP
Các chân địa chỉ
13
A0-A12
8
Q0-Q7
Các chân dữ liệu
P
M2764A
E
Chọn chip
G
Điều khiển đọc
VSS
AI00776B

EPROM 2764

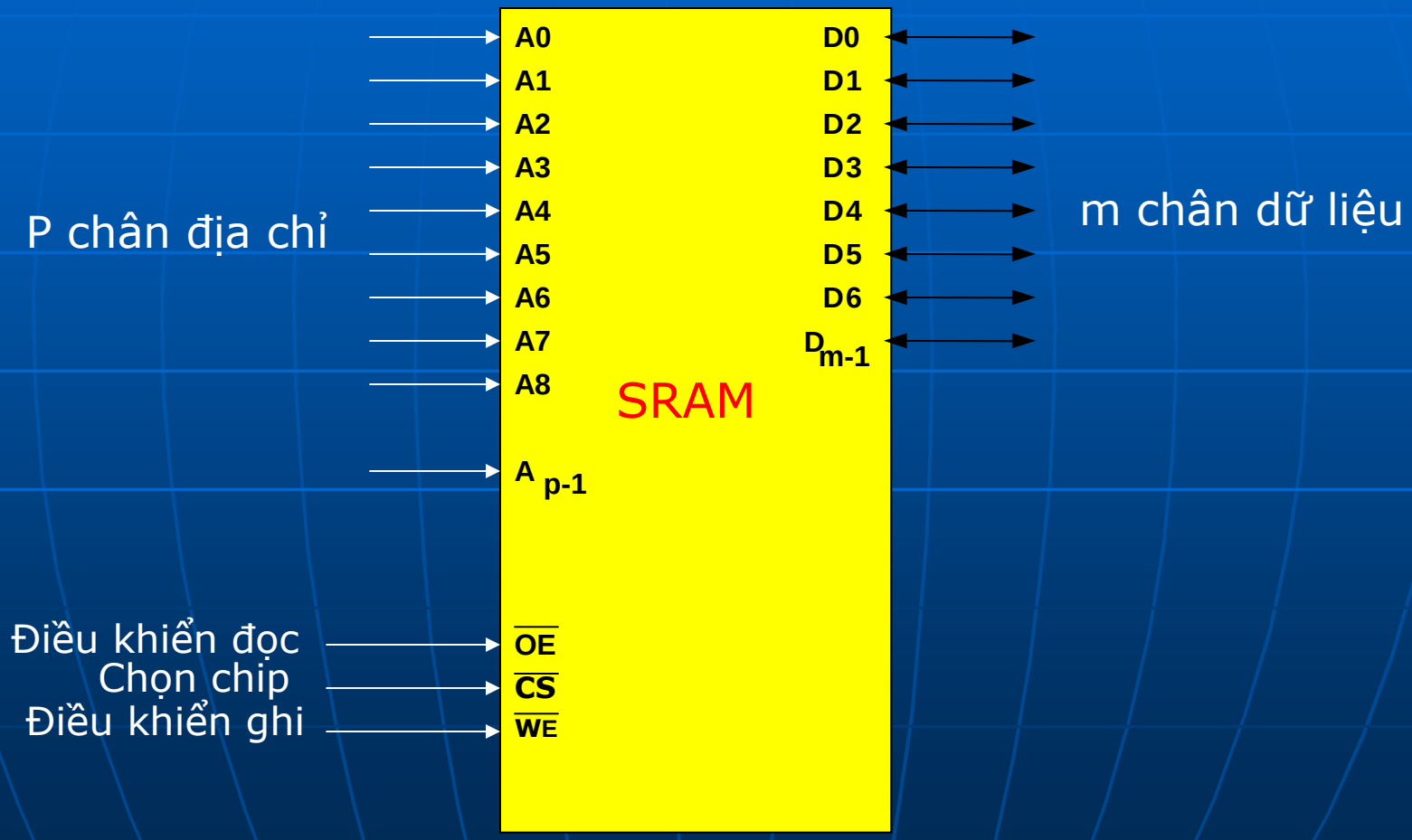
Mode	$\overline{E}$	$\overline{G}$	$\overline{P}$	A9	V_{PP}	Q0 - Q7
Read	V_{IL}	V_{IL}	V_{IH}	X	V_{CC}	Data Out
Output Disable	V_{IL}	V_{IH}	V_{IH}	X	V_{CC}	Hi-Z
Program	V_{IL}	V_{IH}	V_{IL} Pulse	X	V_{PP}	Data In
Verify	V_{IL}	V_{IL}	V_{IH}	X	V_{PP}	Data Out
Program Inhibit	V_{IH}	X	X	X	V_{PP}	Hi-Z
Standby	V_{IH}	X	X	X	V_{CC}	Hi-Z
Electronic Signature	V_{IL}	V_{IL}	V_{IH}	V_{ID}	V_{CC}	Codes Out

Note: X = V_{IH} or V_{IL}, V_{ID} = 12V ± 0.5%.

Lập trình cho 2764

- Trước hết cần phải xoá
  - Xoá một chip tức là làm cho tất cả các bit = 1
- Xoá một chip EPROM bằng tia cực tím
- Lập trình bằng cách:
  - VPP mắc ở mức 12.5V
  - E và P đều ở mức thấp TTL
- Các bit dữ liệu đưa vào các chân dữ liệu
- Các bit địa chỉ đưa vào các chân địa chỉ

4.3 Các chip SRAM



Đọc dữ liệu từ một chip SRAM

Để đọc dữ liệu từ 1 ô nhớ nào đó của 1 chip SRAM nào đó, vi xử lý cần phải:

- Chọn chip đó: 0 ----------> CS
- Áp các tín hiệu địa chỉ vào $A_{p-1} - A_0$
- Đọc: 0 ---------- > OE

Kết quả là m bit dữ liệu cần đọc xuất hiện ở các chân dữ liệu $D_{m-1} - D_0$

Ghi dữ liệu vào một chip SRAM

Để ghi m bit dữ liệu vào 1 ô nhớ nào đó của 1 chip SRAM nào đó, vi xử lý cần phải:

- Chọn chip đó: 0 --------> CS
- Áp các tín hiệu địa chỉ vào $A_{p-1} - A_0$
- Áp m bit dữ liệu cần ghi vào các chân dữ liệu $D_{m-1} - D_0$
- Ghi: 0 --------- > WE

Kết quả là các bit dữ liệu ở các chân dữ liệu sẽ được ghi vào ô nhớ đã chọn

4.4 Bus hệ thống của 8088

- Bus địa chỉ 20-bit: gồm các đường địa chỉ được ký hiệu từ A_{19} đến A_0
- Bus dữ liệu 8-bit: gồm các đường dữ liệu được ký hiệu từ D_7 đến D_0
- Bus điều khiển gồm các đường điều khiển riêng lẽ phục vụ cho hoạt động truy cập bộ nhớ và các cổng I/O, mỗi đường thường được ký hiệu bằng tên của tín hiệu điều khiển
- Bus hệ thống không nối trực tiếp với các chân của 8088: thông qua các mạch đệm, chốt.

80x86 Microprocessors

Product	***8008***	***8080***	***8085***	***8086***	***8088***	***80286***	***80386***	***80486***	***Pent.***	***Pent. Pro***
Year Introduced	1972	1974	1976	1978	1979	1982	1985	1989	1992	1995
Technology	PMOS	NMOS	NMOS	NMOS	NMOS	NMOS	CMOS	CMOS	BICMOS	BICMOS
Clock Rate	0.5-0.8	2-3	3-8	5-10	5-8	10-16?	16-40	66	60-66+	150
Number of Pins	18	40	40	40	40		132	168	273	387
Number of transistors	3000	4500	6500	29K	29K	130K	275K	1.2M	3M	5.5M
Number of instructions	66	111	113	133	133					
Physical Memory	16K	64K	64K	1M	1M	16M	16M4GB	4GB	4GB	64G
Virtual Memory	none	none	none	none	none	1G	64T	64T	64T	64T
Internal Data Bus	8	8	8	16	16	16	32	32	64	32
External Data Bus	8	8	8	16	8	16	16,32	32	64	64
Address Bus	8	16	16	20	20	24	24,32	32	32	36
Data Types	8	8	8	8,16	8,16	8,16	8,16,32	8,16,32	8,16,32	8,16,32

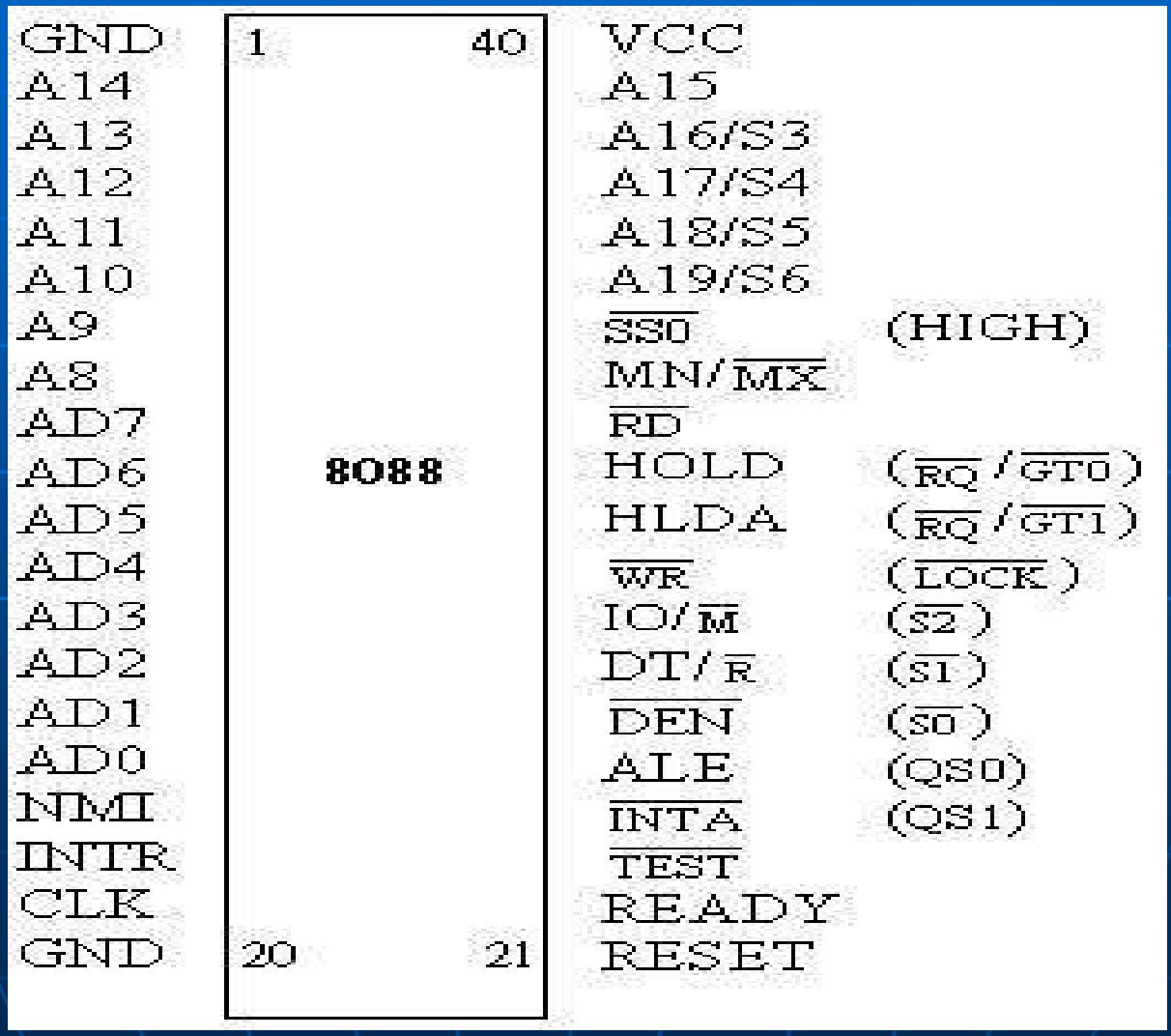
8088/8086 Microprocessor

- DIP 40 pin
- Data bus
  - Bus dữ liệu trong :16 bit
  - Bus dữ liệu ngoài của 8088: 8 bit dùng AD0-AD7
  - Bus dữ liệu ngoài của 8086:16 bit dùng AD0-AD15
  - ALE (Address Latch Enable)

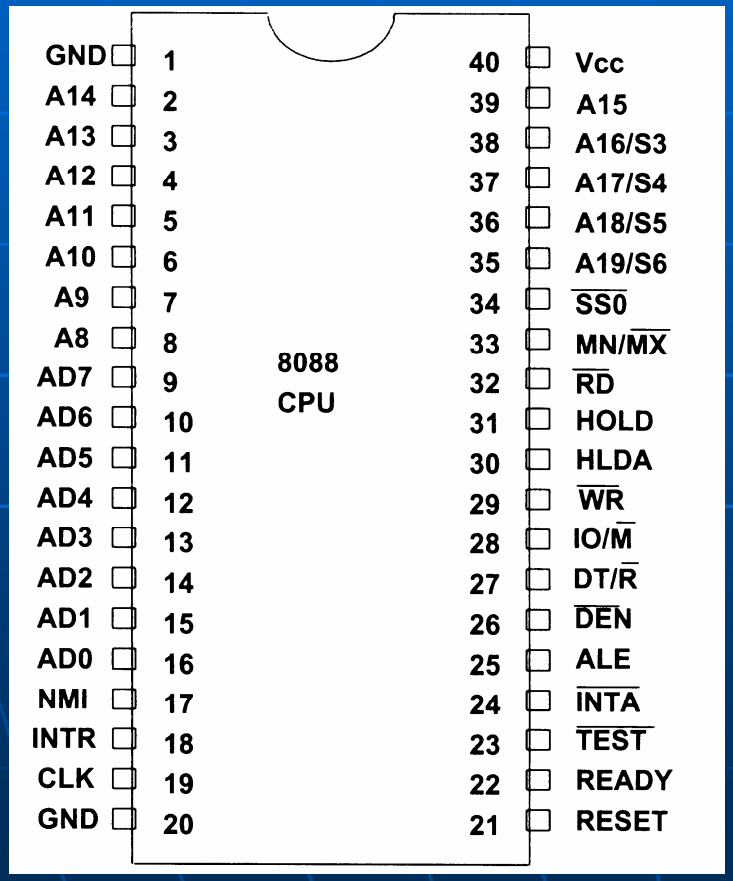
8088/8086 Microprocessor

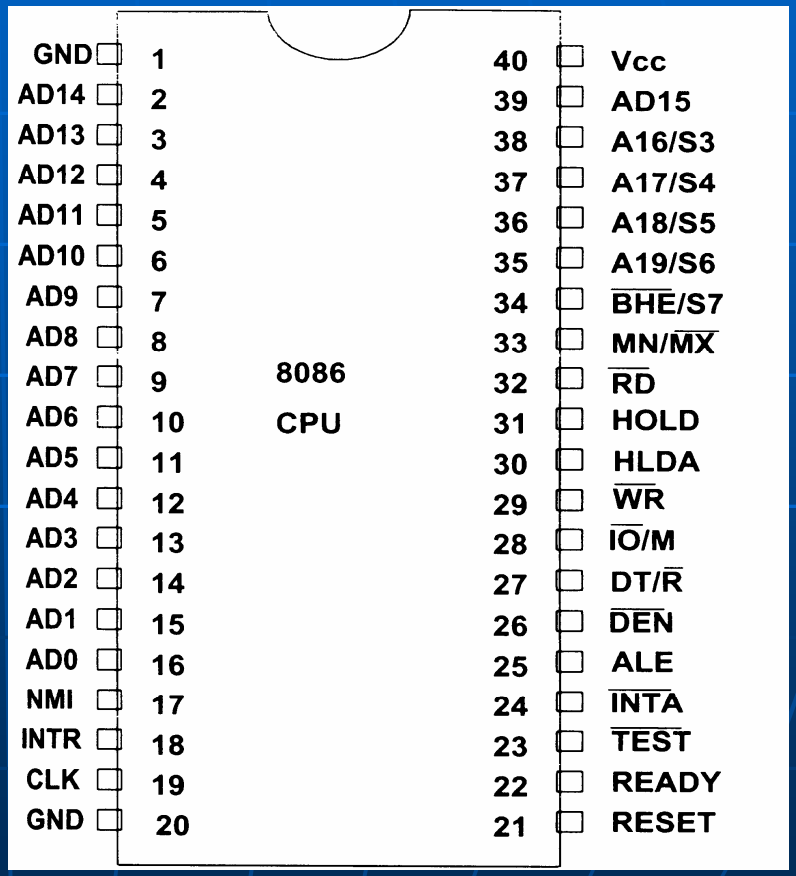
- Bus địa chỉ
  - ALE = 1
  - Sử dụng 74LS373 để tách và chốt địa chỉ
    - Đầu vào: AD0-AD7 (8088) hoặc AD0-AD15 (8086) và ALE
    - Đầu ra: A0-A7 (8088) hoặc A0-A15 (8086)

Sơ đồ chân của 8088



Sơ đồ chân 8088/8086 (Min Mode)

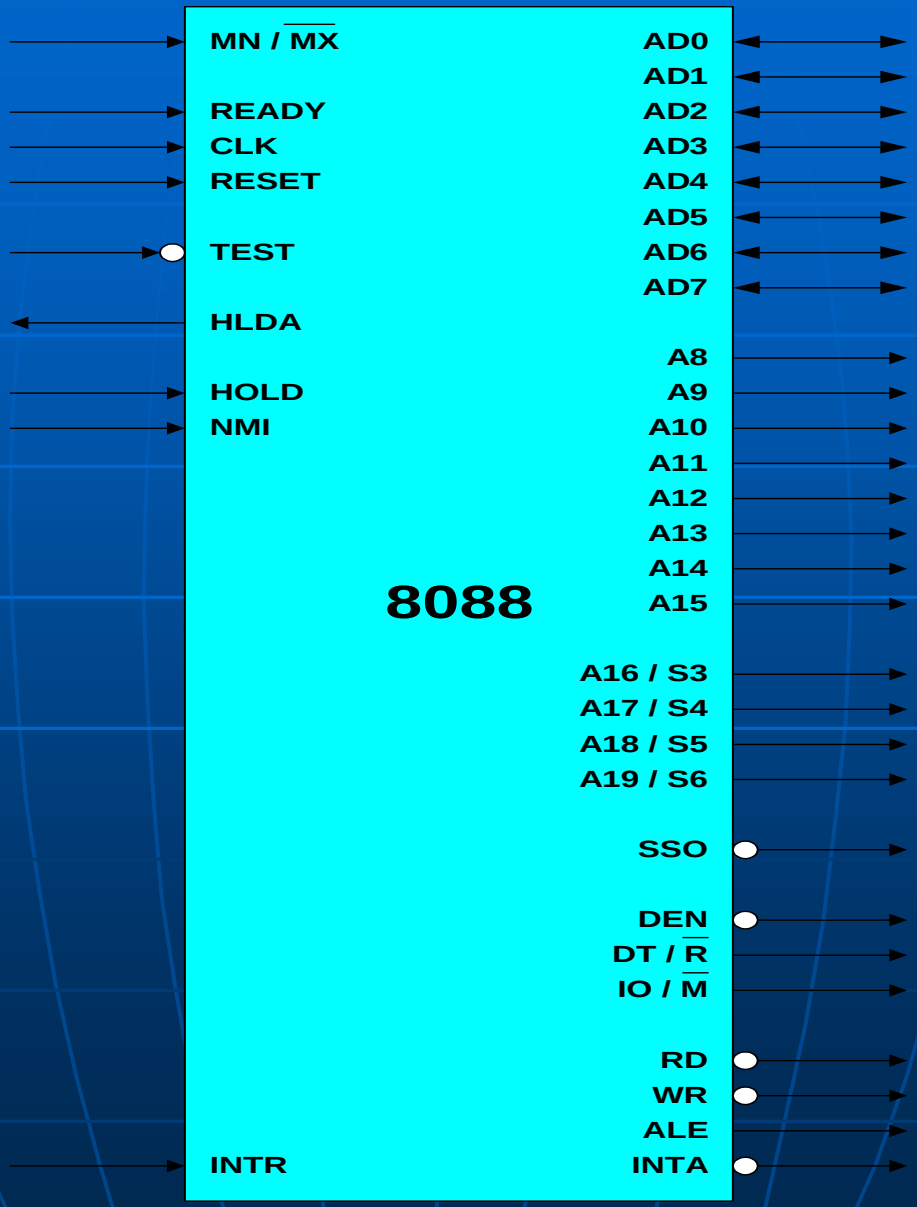




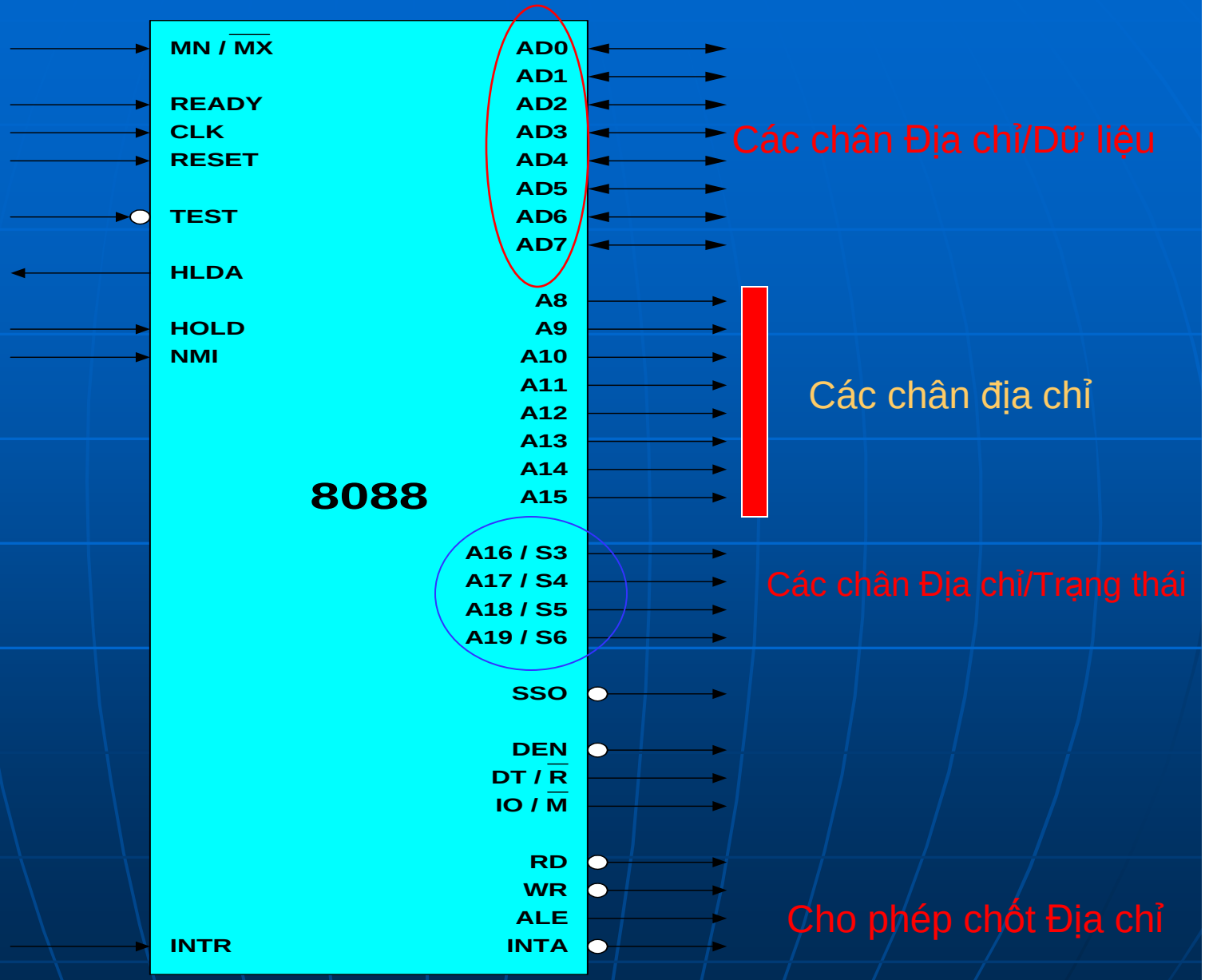
Minimum/Maximum Mode

- Ảnh hưởng đến các chân 24-31
- Minimum Mode
  - Các chân 24-31 là các tín hiệu điều khiển I/O và bộ nhớ
  - Các tín hiệu điều khiển đều từ 8088/8086
  - Tương tự với 8085A
- Maximum Mode
  - Một số tín hiệu điều khiển được tạo ra từ ngoài
  - Một số chân có thêm chức năng mới
  - Khi có dùng bộ đồng xử lý toán 8087

Sơ đồ chân của 8088



Tín hiệu ở các chân của 8088



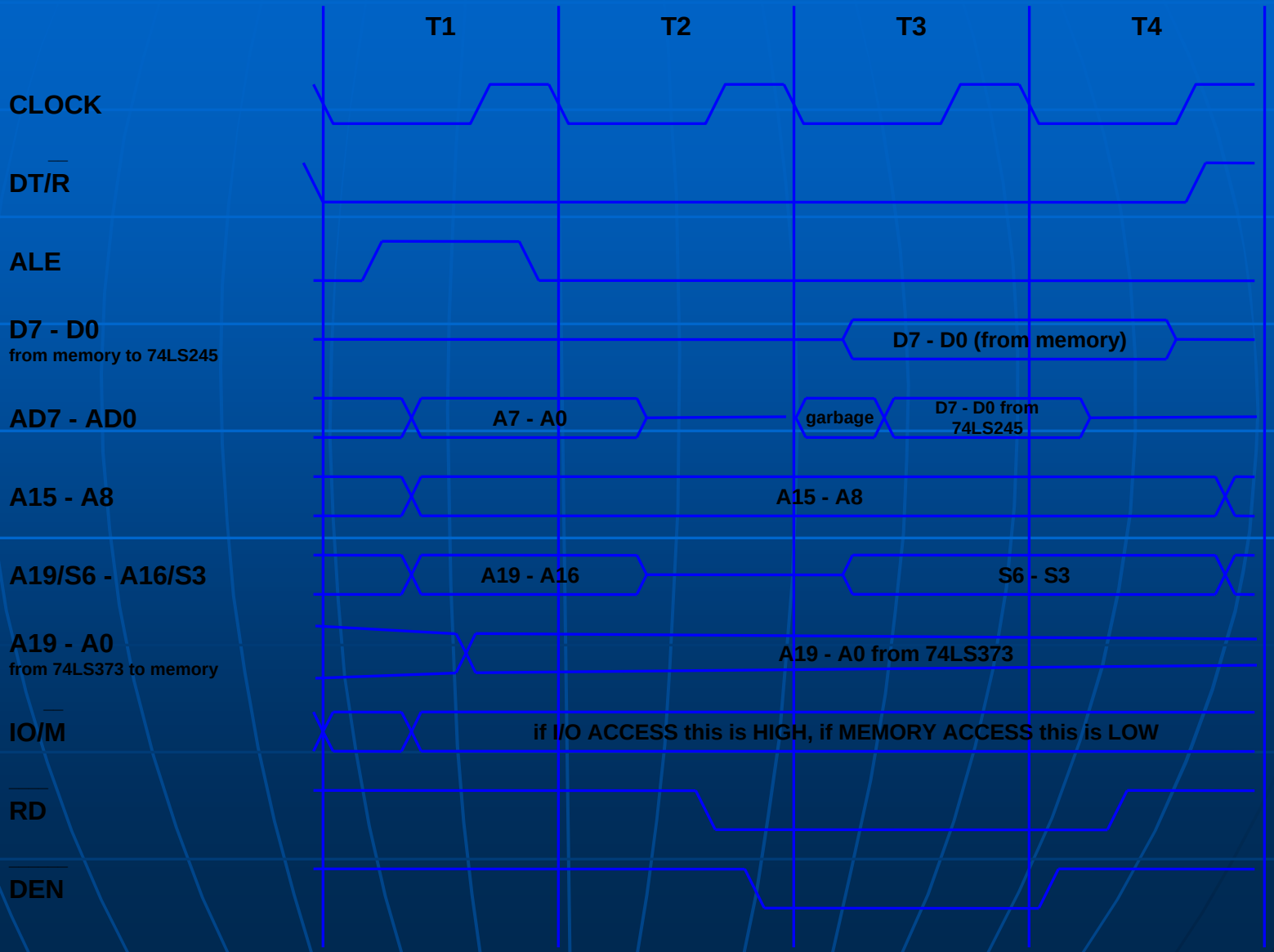
Các chân Địa chỉ/Dữ liệu

- Các chân AD_7 đến AD_0
- Kỹ thuật Multiplexing: Tín hiệu ở các chân này lúc này là tín hiệu địa chỉ, lúc khác là tín hiệu dữ liệu phụ thuộc vào tín hiệu điều khiển ALE (Address Latch Enable):
- ALE = 1: AD_7 đến AD_0 = A_7 đến A_0
- ALE = 0: AD_7 đến AD_0 = D_7 đến D_0

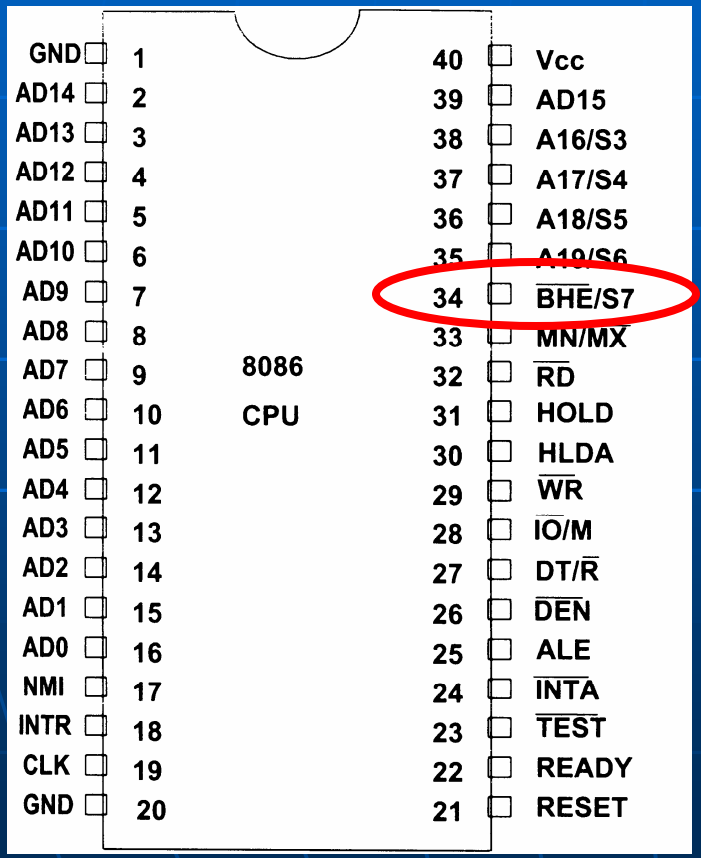
Các chân Địa chỉ và Các chân Địa chỉ/Trạng thái

- Các chân địa chỉ: A_{15} đến A_8
- Tín hiệu ở các chân này luôn là tín hiệu địa chỉ
- Các chân địa chỉ/trạng thái: A_{19}/S_6 đến A_{16}/S_3:
- ALE = 1: A_{19} đến A_{16}
- ALE = 0: S_6 đến S_3

Processor Timing Diagram of 8088 (Minimum Mode) for Memory or I/O Read (with 74245)

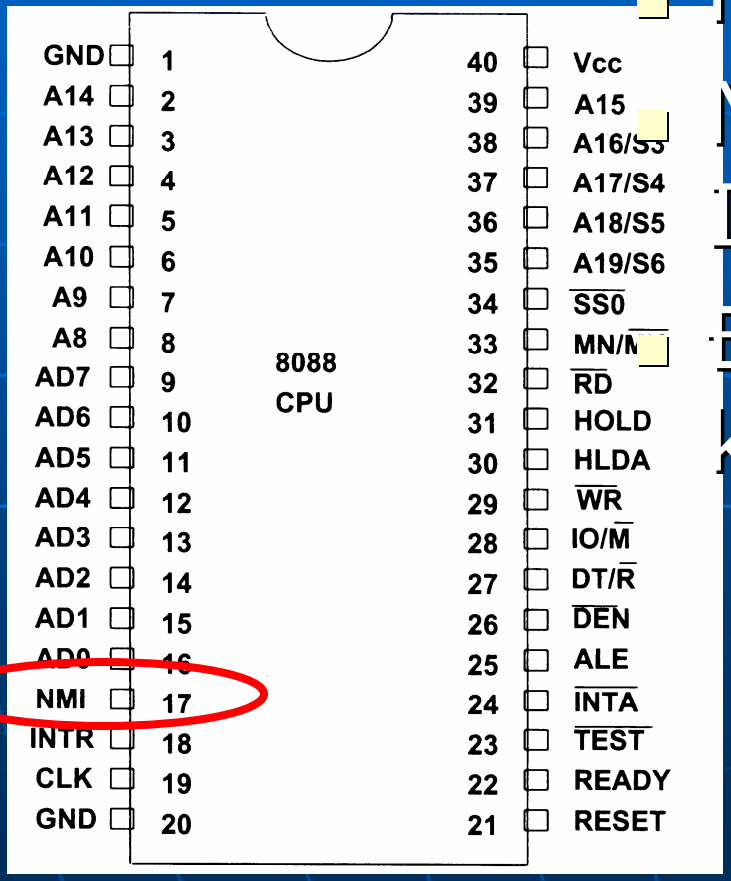


Mô tả chân



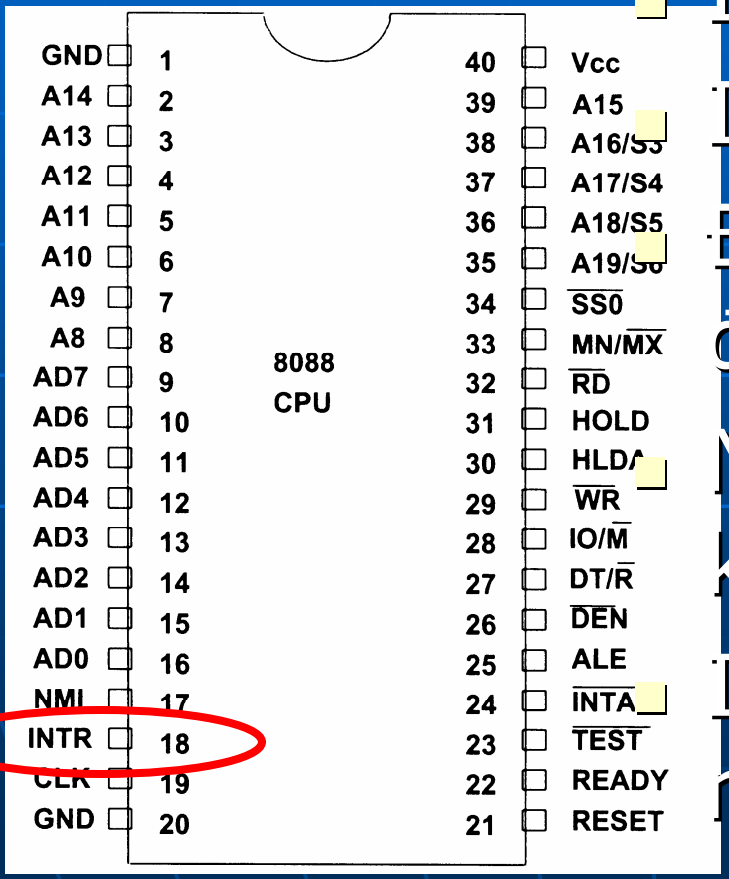
- BHE
  Bus High Enable
  Phân biệt byte thấp và byte cao của một từ (chỉ với 8086)

Mô tả chân



- NMI
- Non Maskable Interrupt
- Đầu vào ngắt không che được

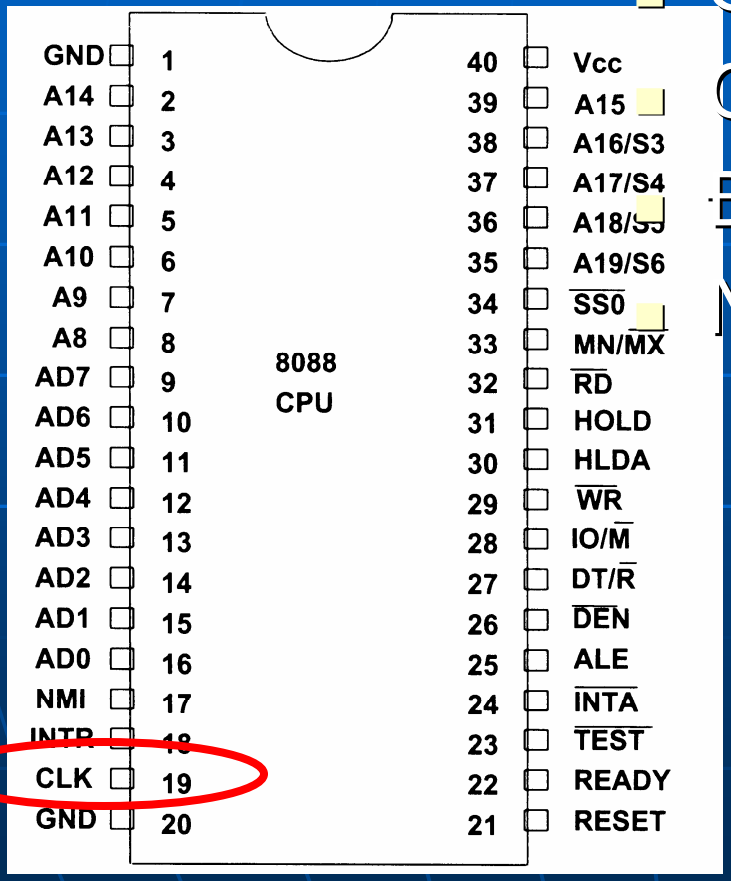
Mô tả chân



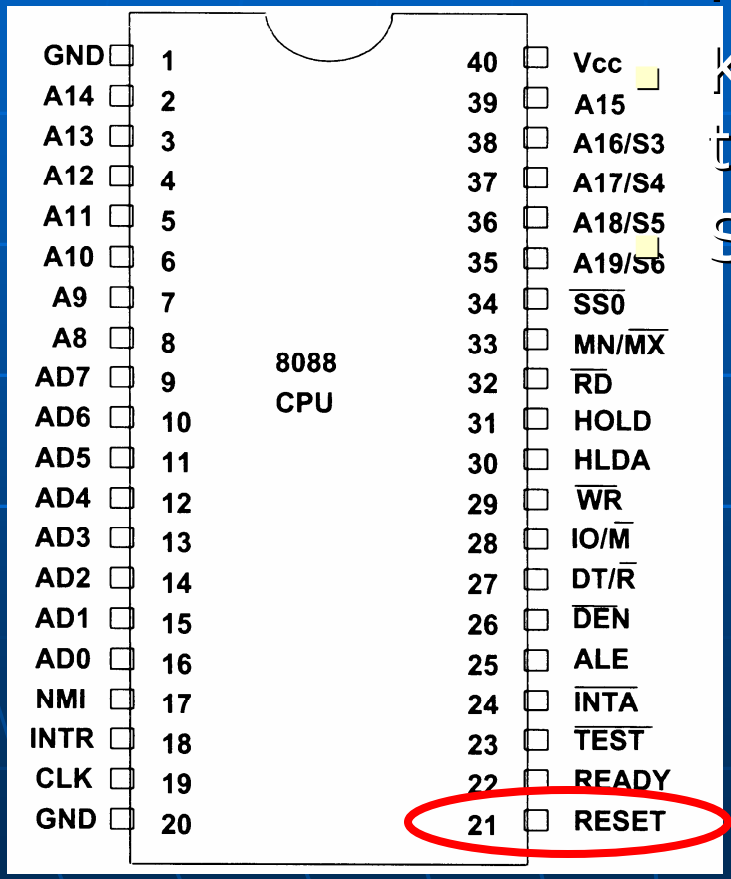
- INTR
  - Interrupt Request
  - Đầu vào ngắt che được
  - Nối với chip điều khiển ngắt 8259
- INTA: chấp nhận ngắt

Mô tả chân

- CLK
- Clock
- Đầu vào đồng hồ
- Nối với chip 8284



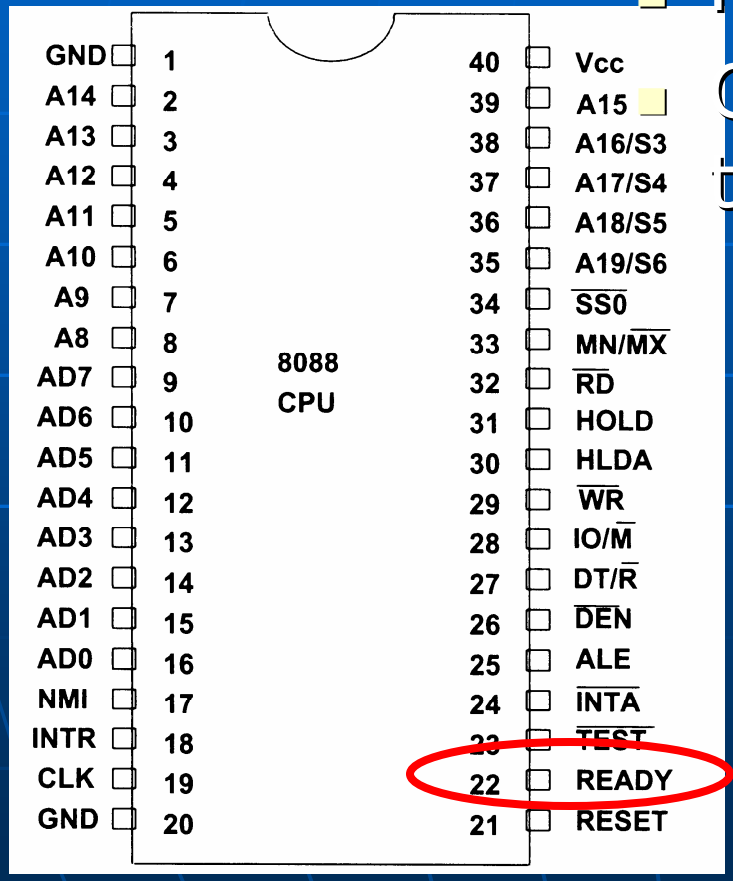
Mô tả chân



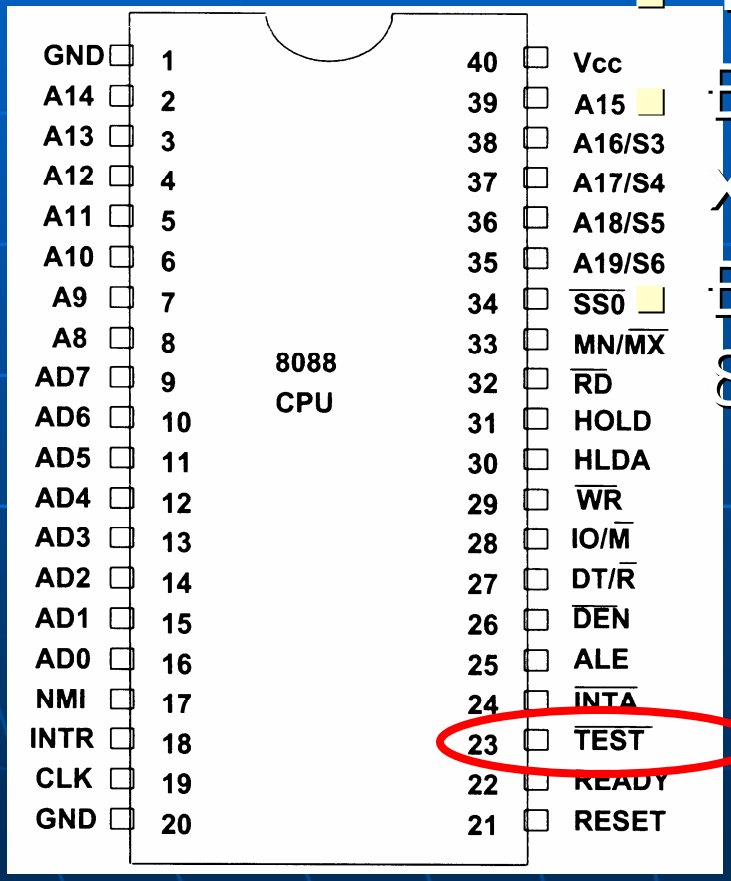
- RESET
- Kết thúc hoạt động hiện thời và huy bỏ mọi thứ
- Sau khi reset
  - CS=FFFFH
  - DS=0000H
  - SS=0000H
  - ES=0000H
  - IP=0000H
  - Các cờ bị xoá
  - Hàng đợI lệnh rỗng

Mô tả chân

- READY
  - Chèn thêm một trạng thái đợi (wait state)

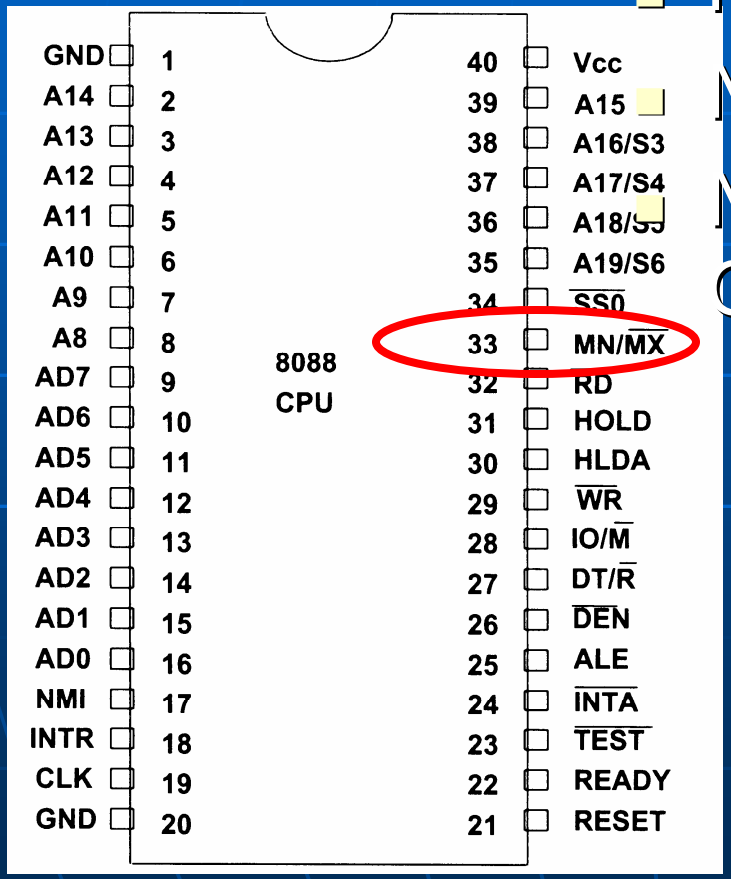


Mô tả chân



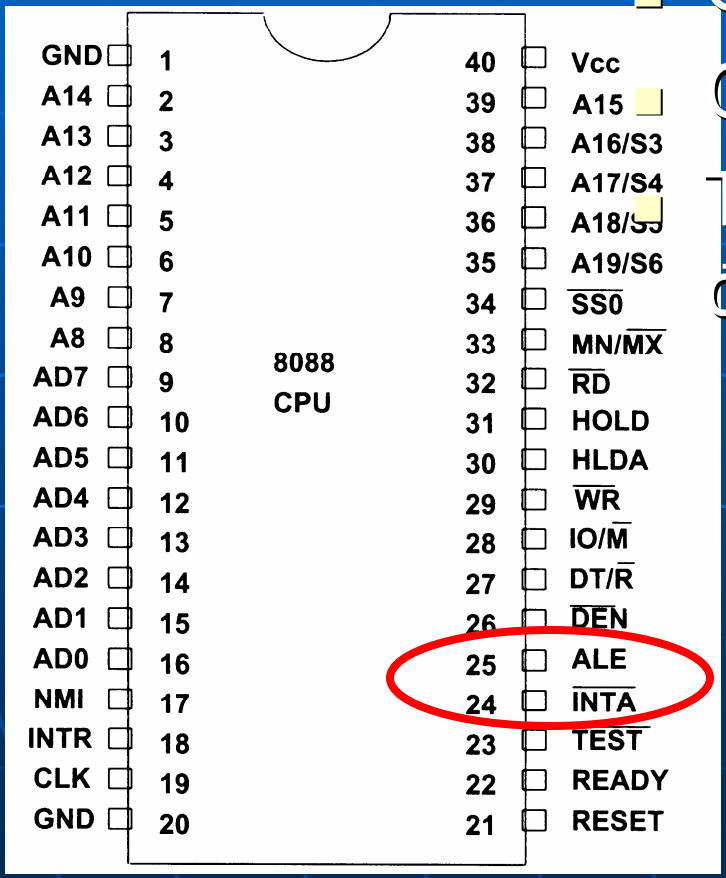
- TEST
  - Đến từ 8087 (Bộ đồng xử lý)
  - Đồng bộ 8088 và 8087

Mô tả chân



- MN/MX
  - Minimum mode = +5V
  - Maximum mode = Gnd

Mô tả chân – Max



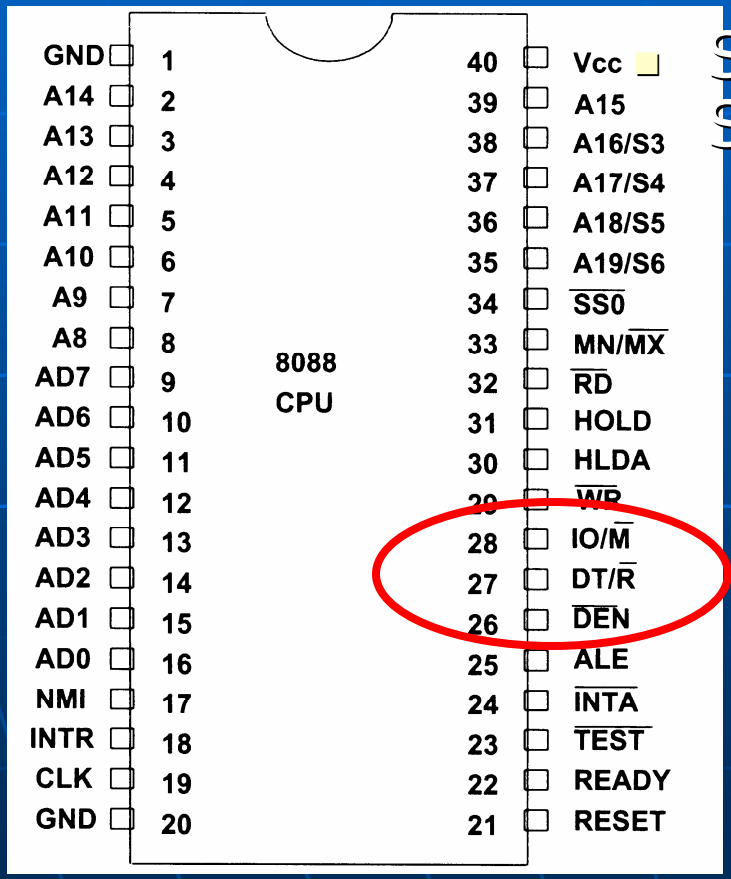
- QS0, QS1

Queue status

Trạng thái của hàng đợi lệnh:

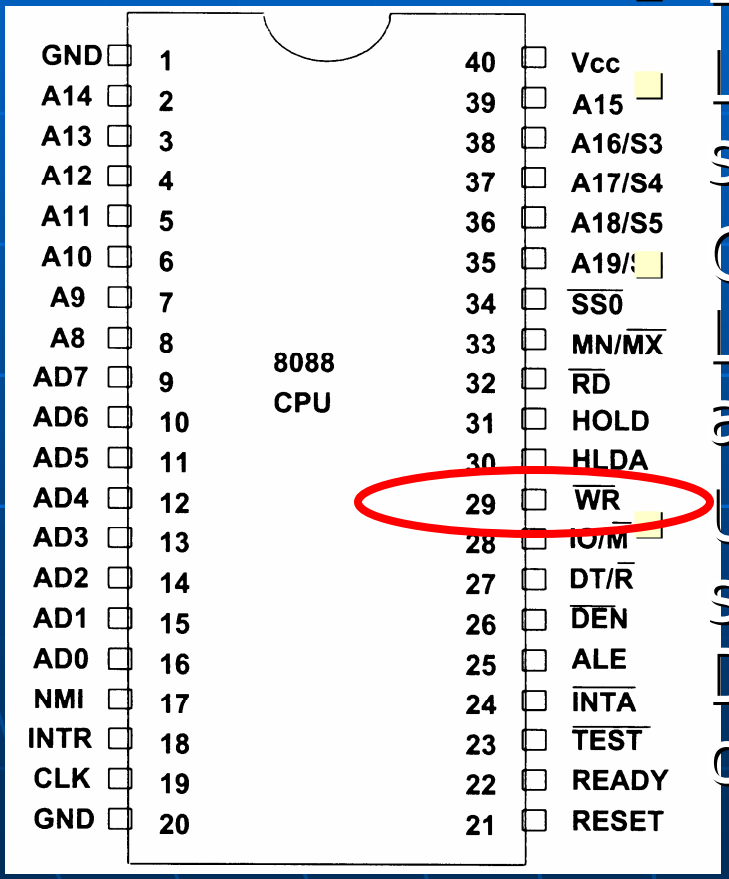
- 00 – No operation
- 01 – first byte of opcode from queue
- 10 – empty the queue
- 11 – subsequent byte from queue

Mô tả chân – Max



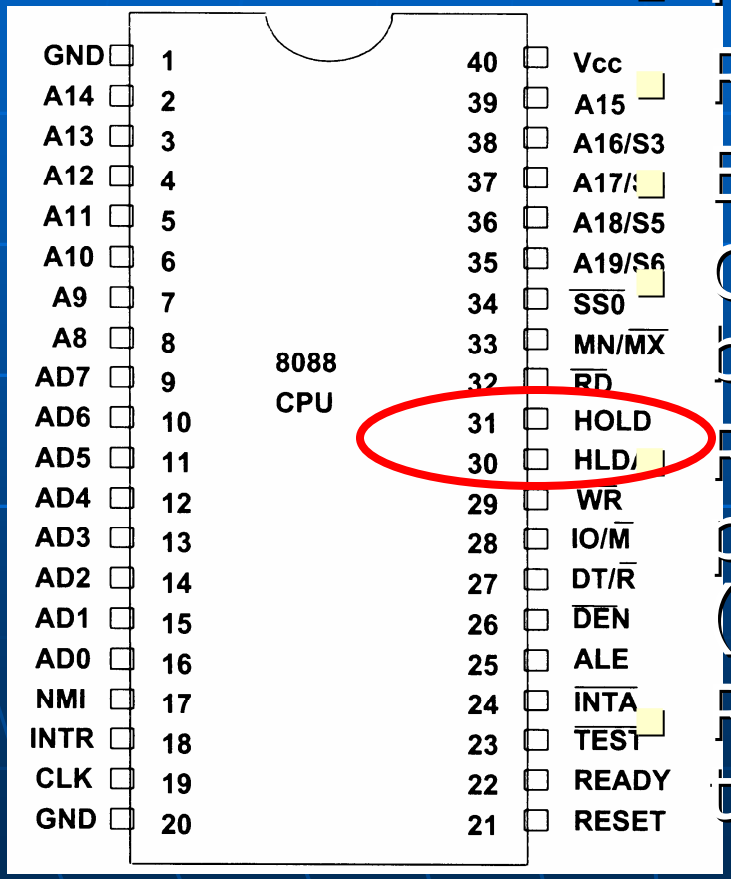
- S0, S1, S2 Status Signal Pins (S2-S0)
  - 000 – INTA – interrupt acknowledge
  - 001 – IORC – read I/O port
  - 010 – IOWC – write I/O port
  - 011 – none - halt
  - 100 – MRDC – code access
  - 101 – MRDC – read memory
  - 110 – MWTC – write memory
  - 111 – none - passive

Mô tả chân – Max



- LOCK
  - Locks processor to system bus
  - Gain the lock by using LOCK prefix on an assembly instruction
  - Used with status signals to prevent DMA from gaining control of the buses

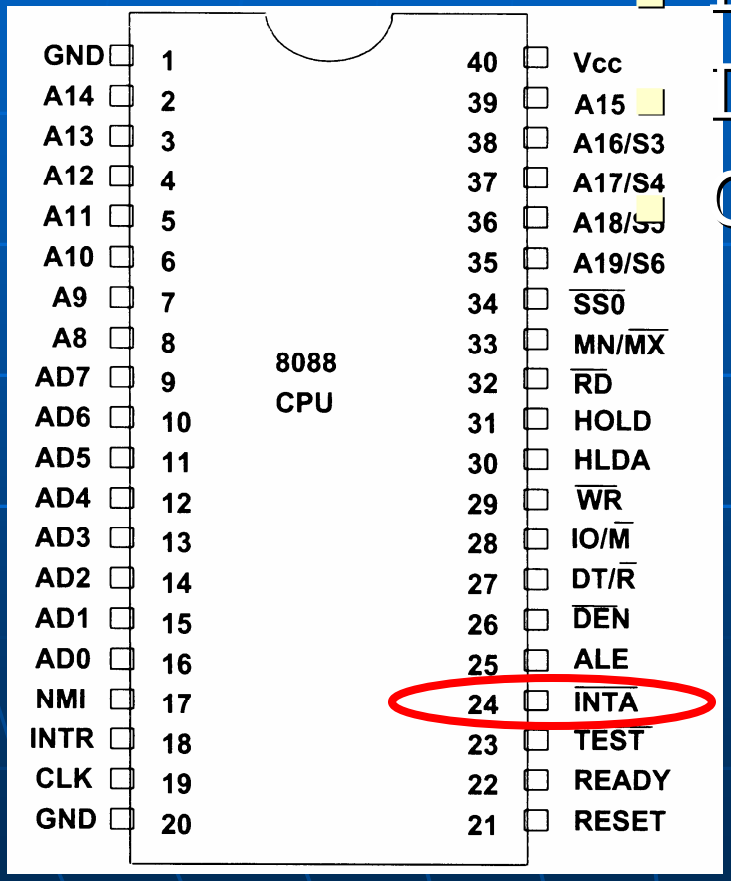
Mô tả chân – Max



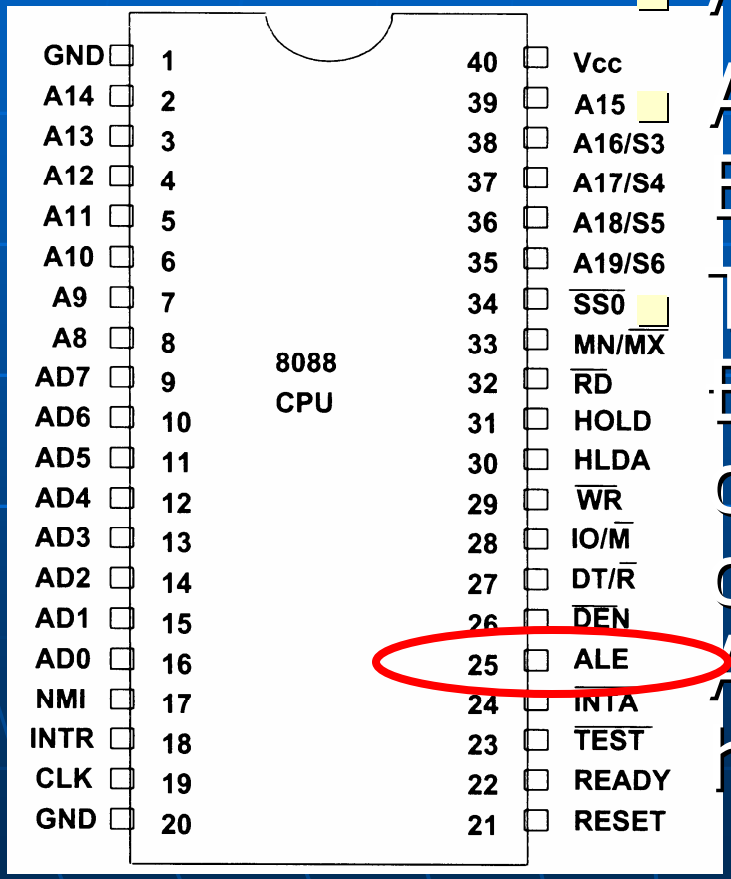
- RQ/GT0, RQ/GT1
  - Request/Grant
  - Bi-directional
  - Gain control of local bus
  - RQ/GT0 normally permanently high (disabled)
  - RQ/GT1 is connected to the 8087

Mô tả chân – Min

- $\overline{INTA}$
  - Interrupt acknowledge
  - Chấp nhận ngắt

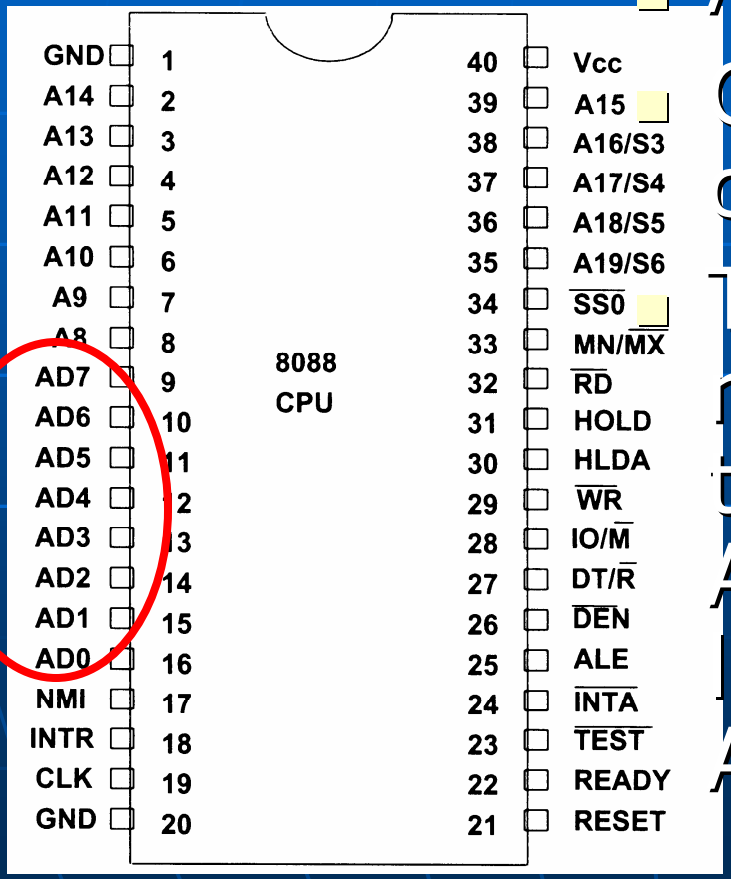


Mô tả chân – Min

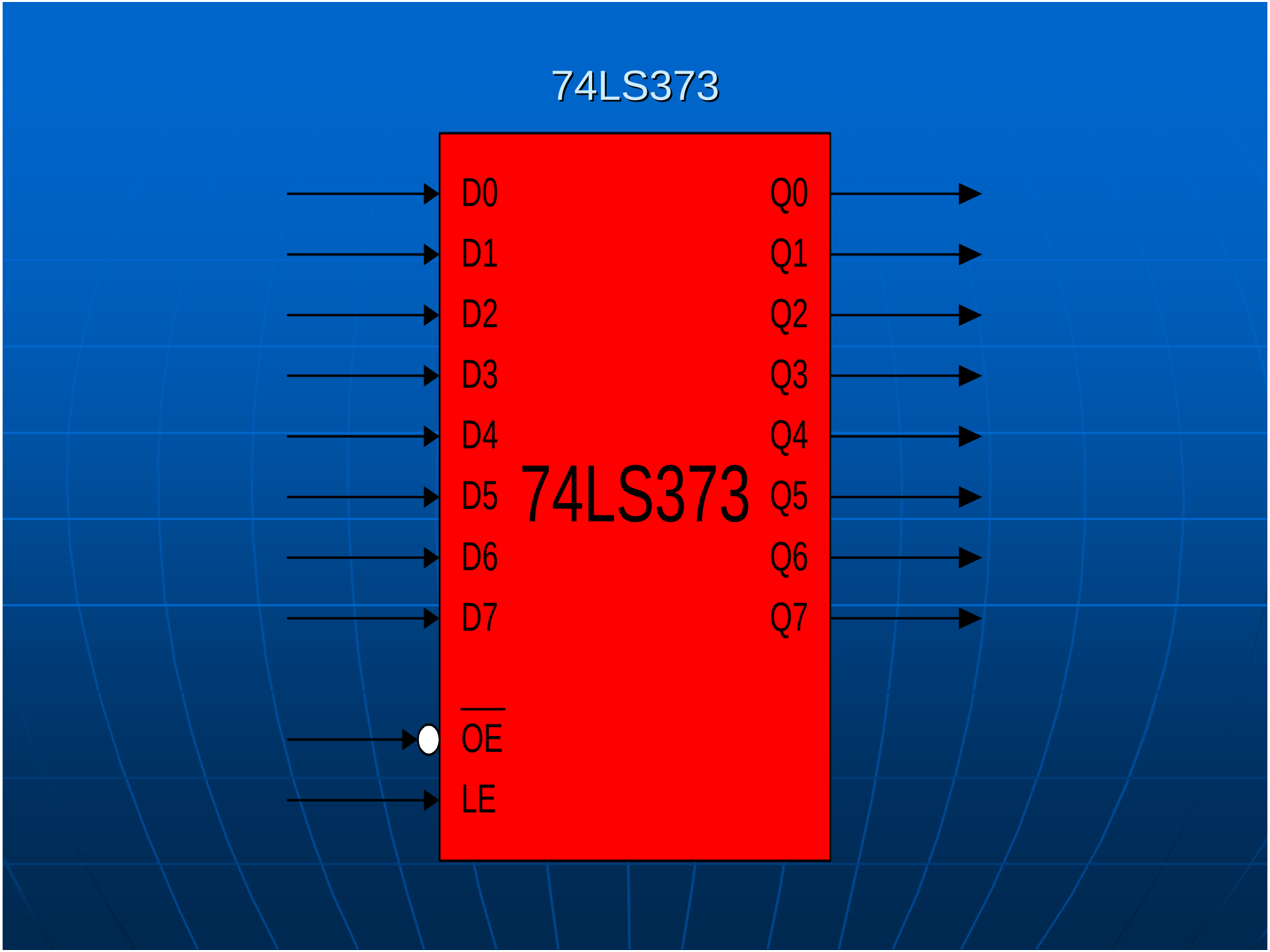


- ALE
- Address Latch Enable
- Tín hiệu ở các chân Địa chỉ/Dữ liệu và các chân Địa chỉ/Trạng thái lúc ALE = 1 là các tín hiệu địa chỉ

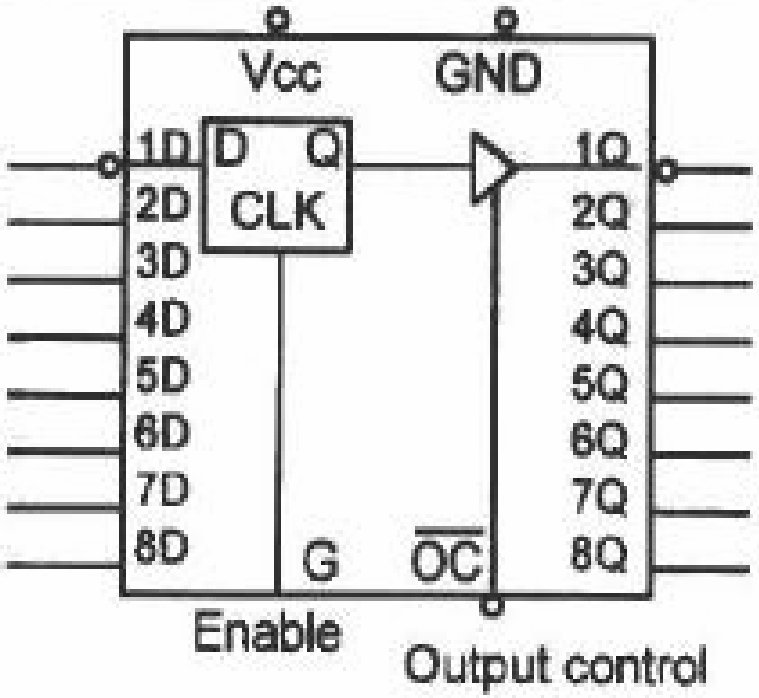
Mô tả chân



- AD0-AD7
  - Các chân Địa chỉ/Dữ liệu
  - Tín hiệu ở các chân này là 8 bit địa chỉ thấp A0 đến A7 khi ALE =1, là 8 bit dữ liệu D0 đến D7 khi ALE = 0


74LS373
D0
D1
D2
D3
D4
D5
D6
D7
74LS373
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
OE
LE

74LS373



Function Table

Output Control	Enable G	Enable D	Output
L	H	H	H
L	H	L	L
L	L	X	Q0
H	X	X	Z

Figure 11-1. 74LS373 D Latch

Dùng 74LS373 để tách và chốt địa chỉ

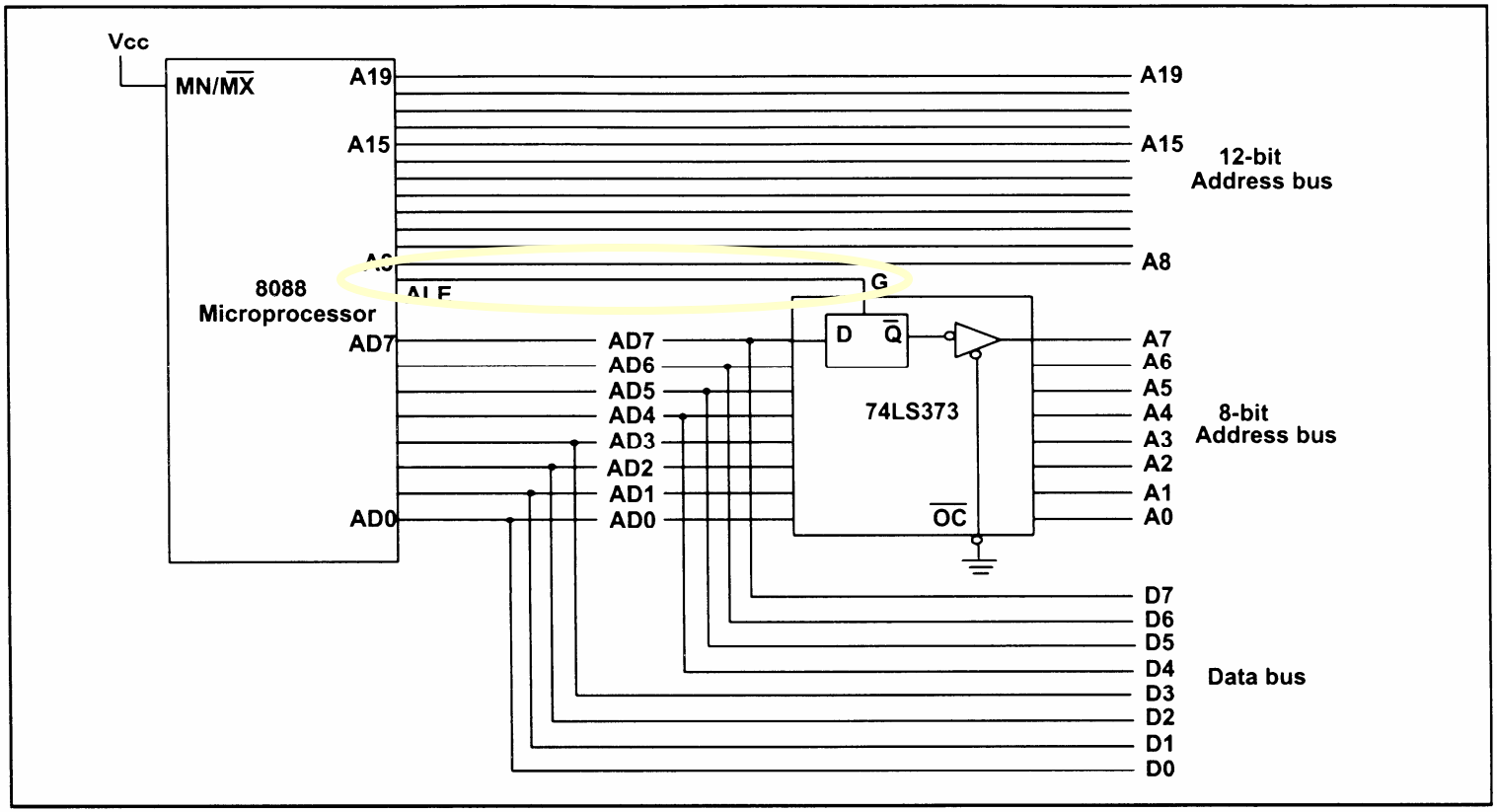
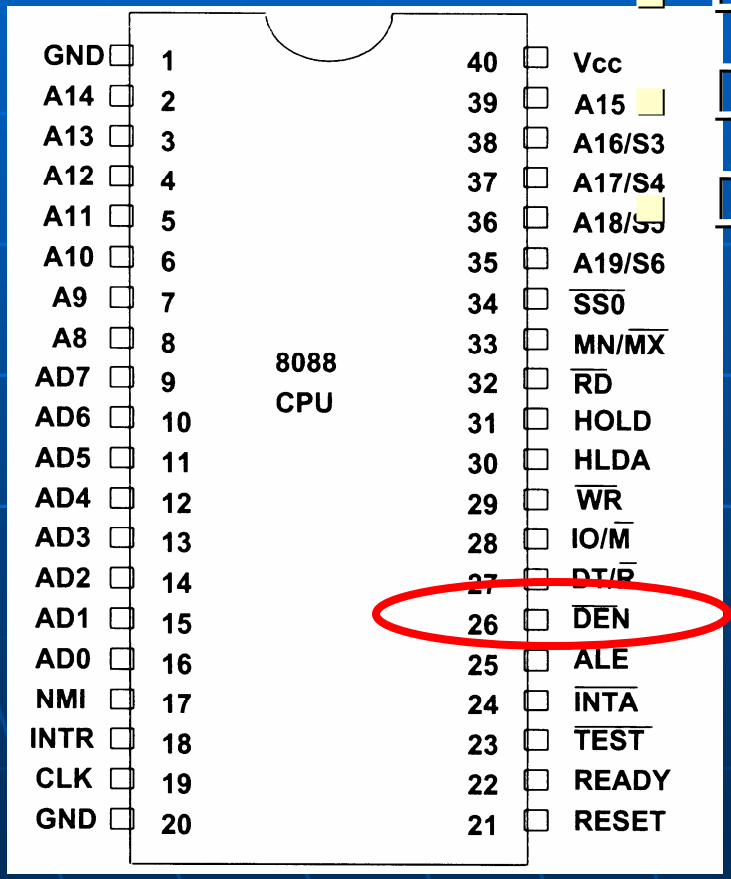


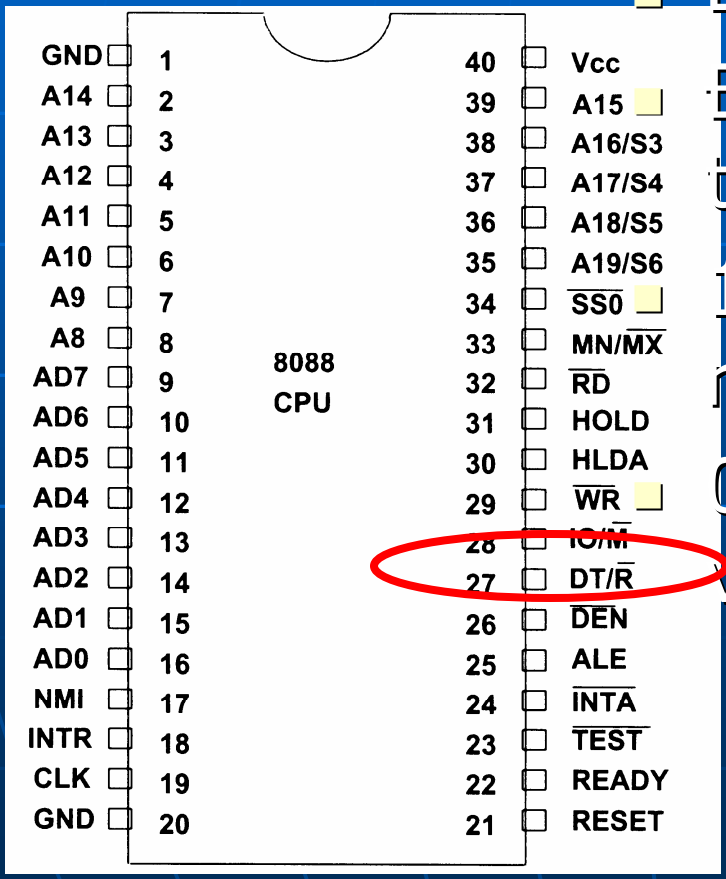
Figure 9-3. Role of ALE in Address/Data Demultiplexing

Mô tả chân – Min



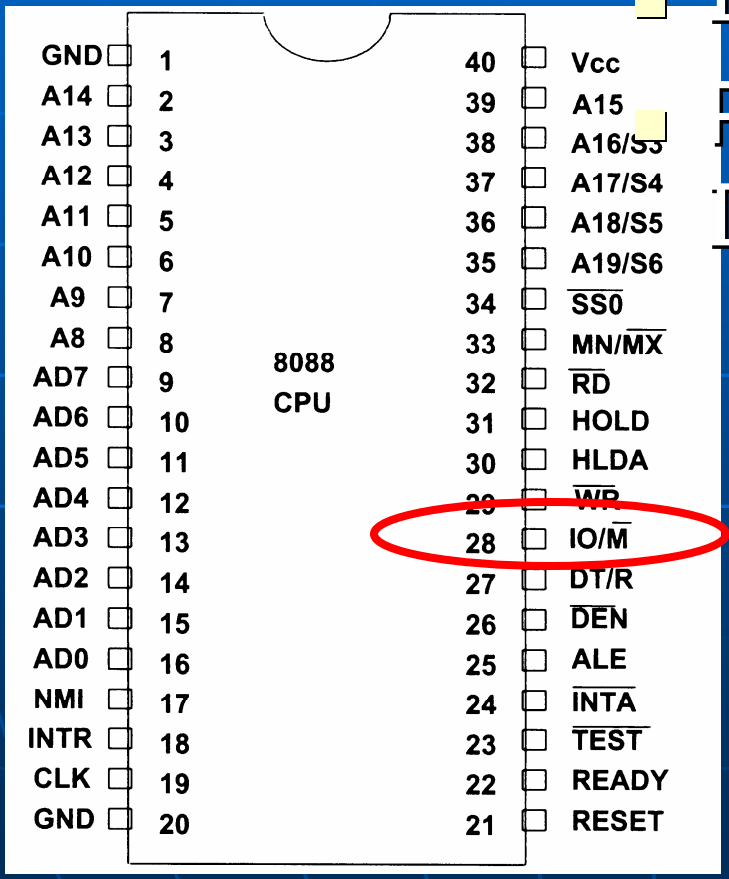
- DEN
  - Data Enable
  - Dữ liệu có nghĩa

Mô tả chân – Min



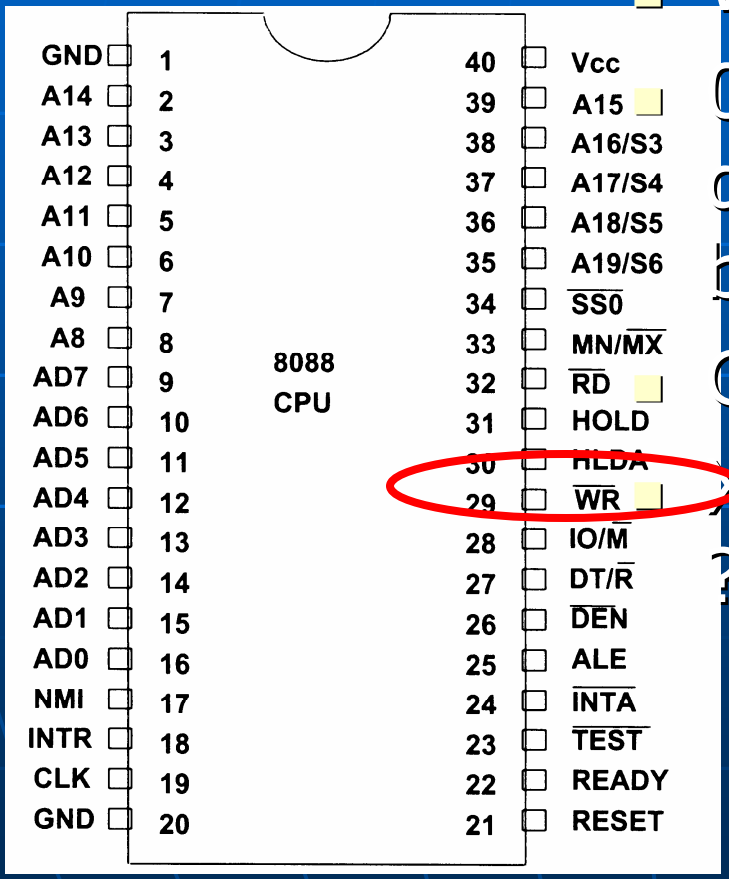
- DT/R _
  - Điều khiển hướng của tín hiệu dữ liệu:
  - 1: Tín hiệu dữ liệu đi ra từ 8088
  - 0: Tín hiệu dữ liệu đi vào 8088

Mô tả chân – Min



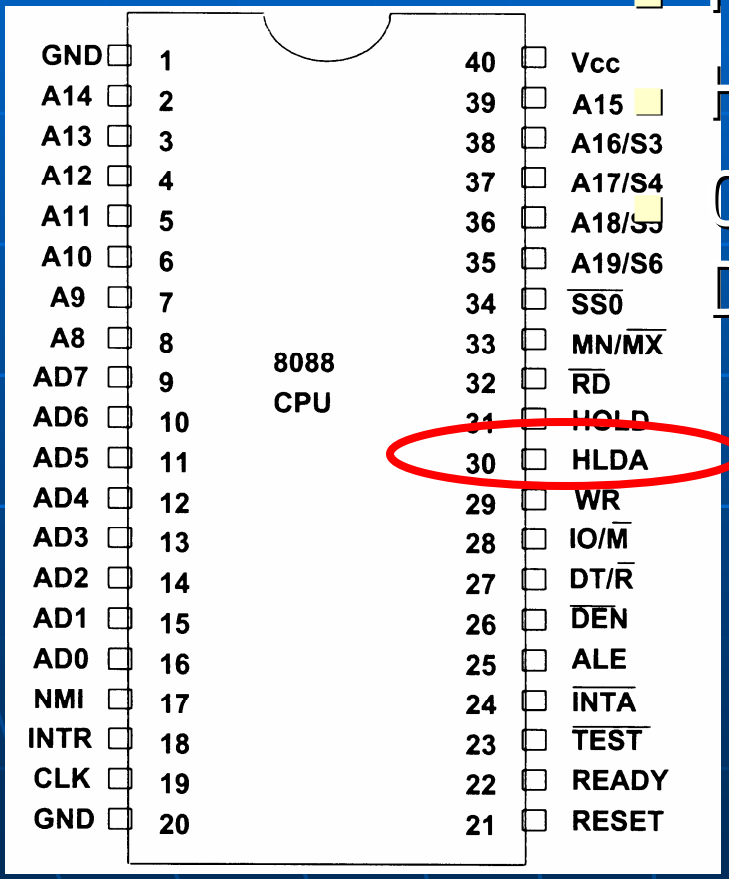
- IO/M _
- Phân biệt: truy cập I/O hay Bộ nhớ
  - 1: 8088 truy cập I/O
  - 0: 8088 truy cập bộ nhớ

Mô tả chân – Min



- WR__
  - 0: Tín hiệu trên bus dữ liệu được ghi vào bộ nhớ hoặc I/O
  - Ghi bộ nhớ: ?
  - Xuất dữ liệu ra cổng: ?

Mô tả chân – Min



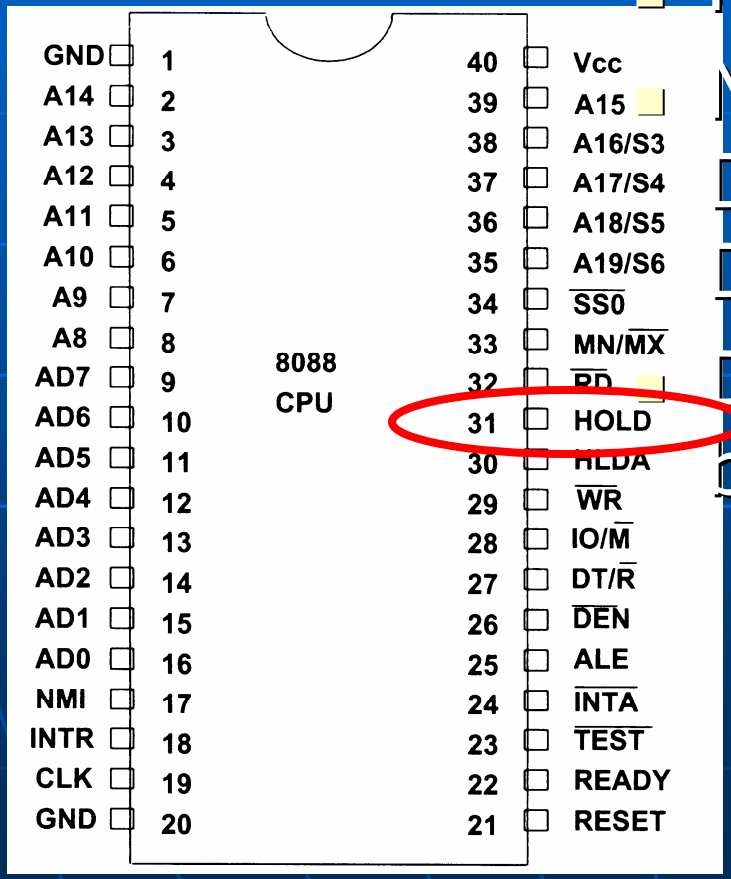
- HLDA

Hold Acknowledge

0: Chấp nhận yêu cầu DMA ở HOLD

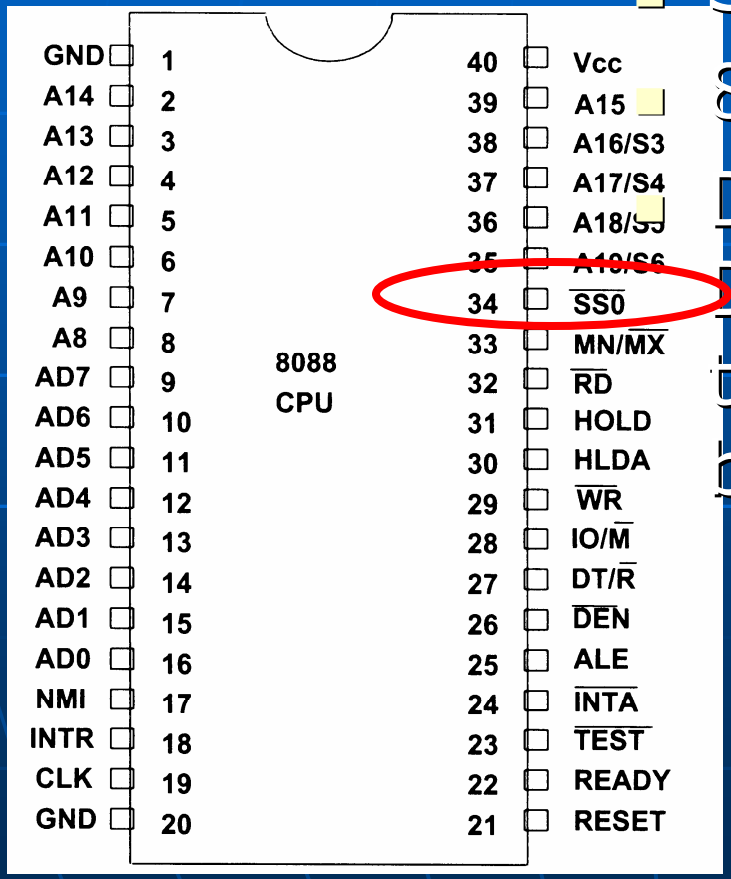
- Báo cho Bộ điều khiển DMA được phép sử dụng bus hệ thống

Mô tả chân – Min



- HOLD
  - Nhận tín hiệu yêu cầu DMA từ Bộ điều khiển DMA (DMAC)
  - DMAC muốn sử dụng bus hệ thống

Mô tả chân – Min



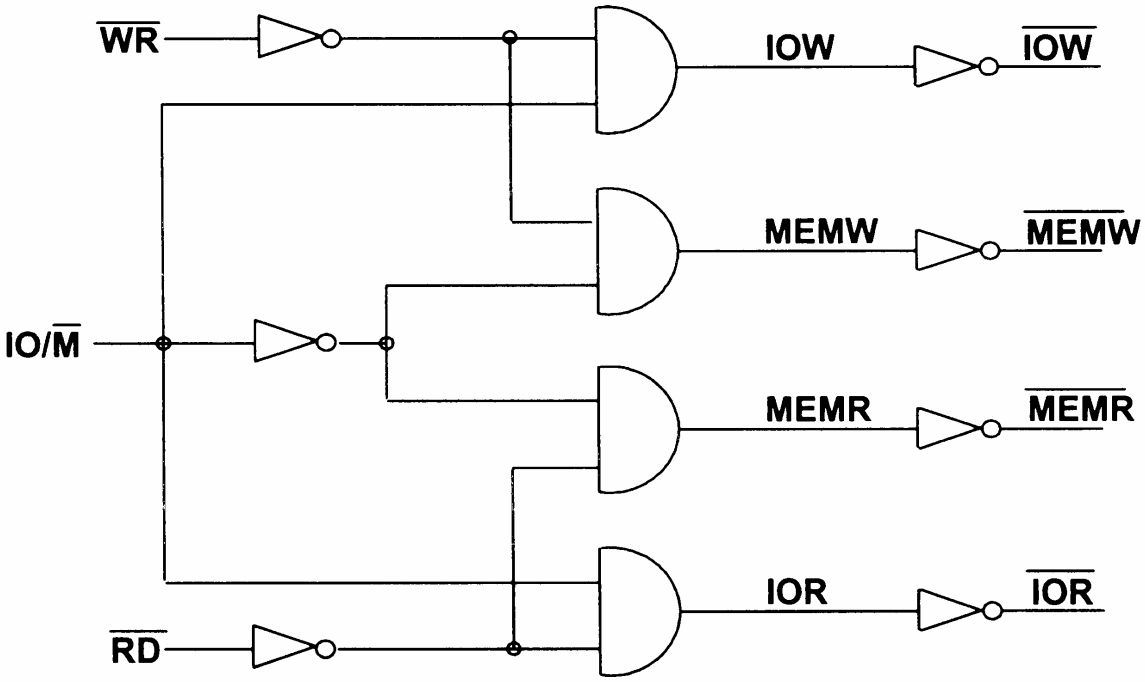
- SSO
- 8088
- Dùng với IO/M và DT/R để xác định trạng thái của chu kỳ bus hiện thời

Các tín hiệu điều khiển

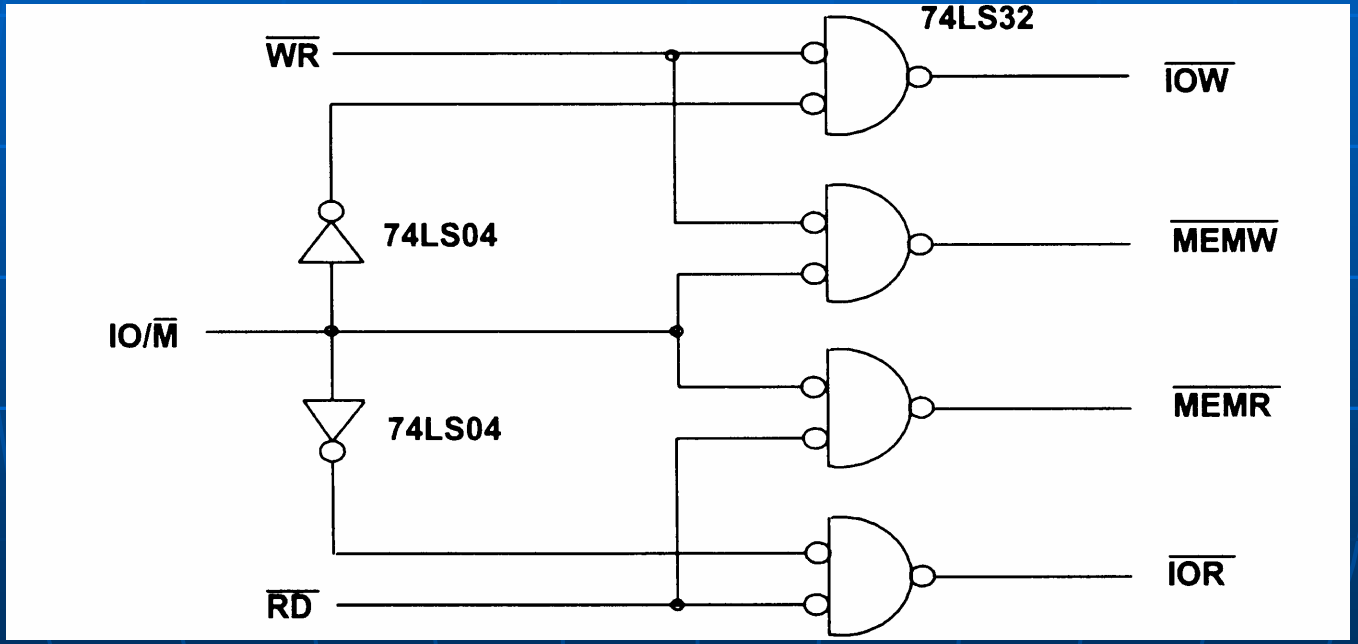
- Có thể sử dụng các cổng logic để tạo ra các tín hiệu điều khiển khác từ các tín hiệu điều khiển sẵn có
  - 3 Tín hiệu:
    - RD, WR and IO/M

$\overline{RD}$	$\overline{WR}$	IO/$\overline{M}$	Signal
0	1	0	$\overline{MEMR}$
1	0	0	$\overline{MEMW}$
0	1	1	$\overline{IOR}$
1	0	1	$\overline{IOW}$
0	0	X	Never happens

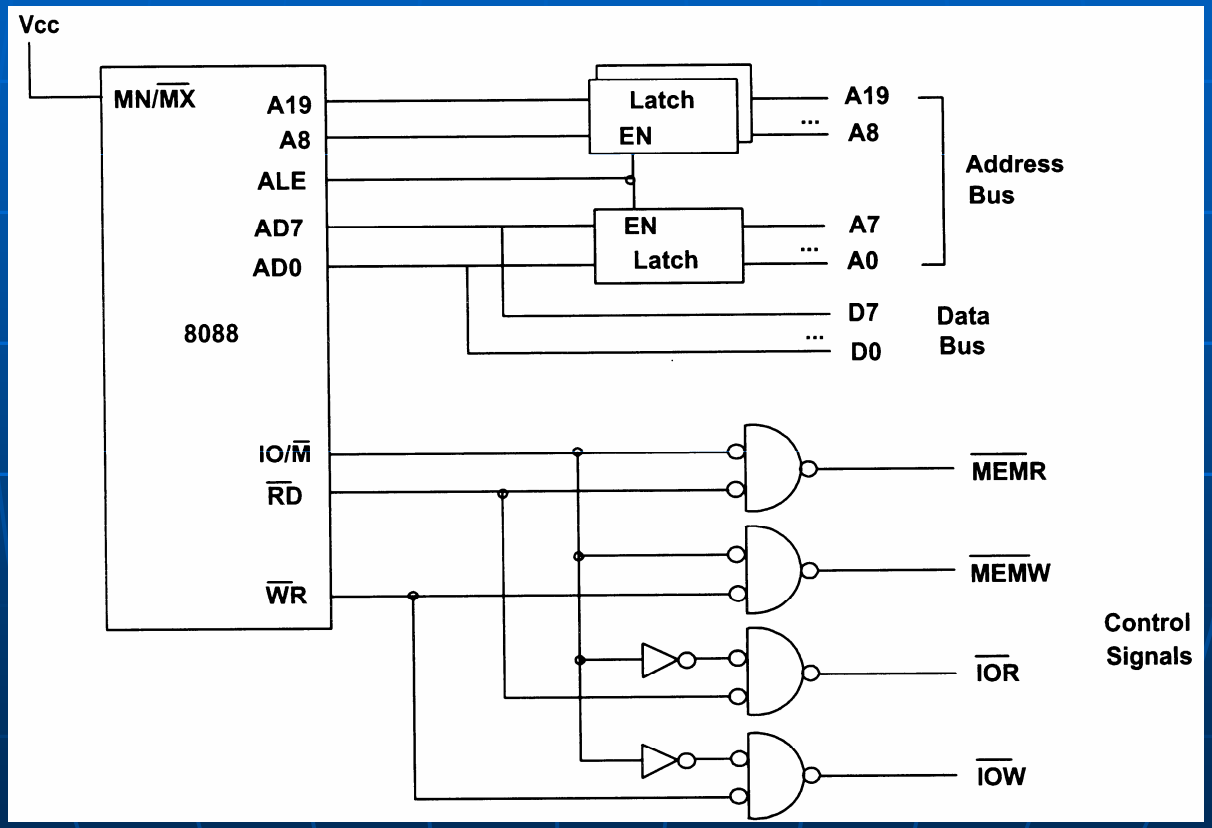
Tạo ra các tín hiệu điều khiển (Min Mode)



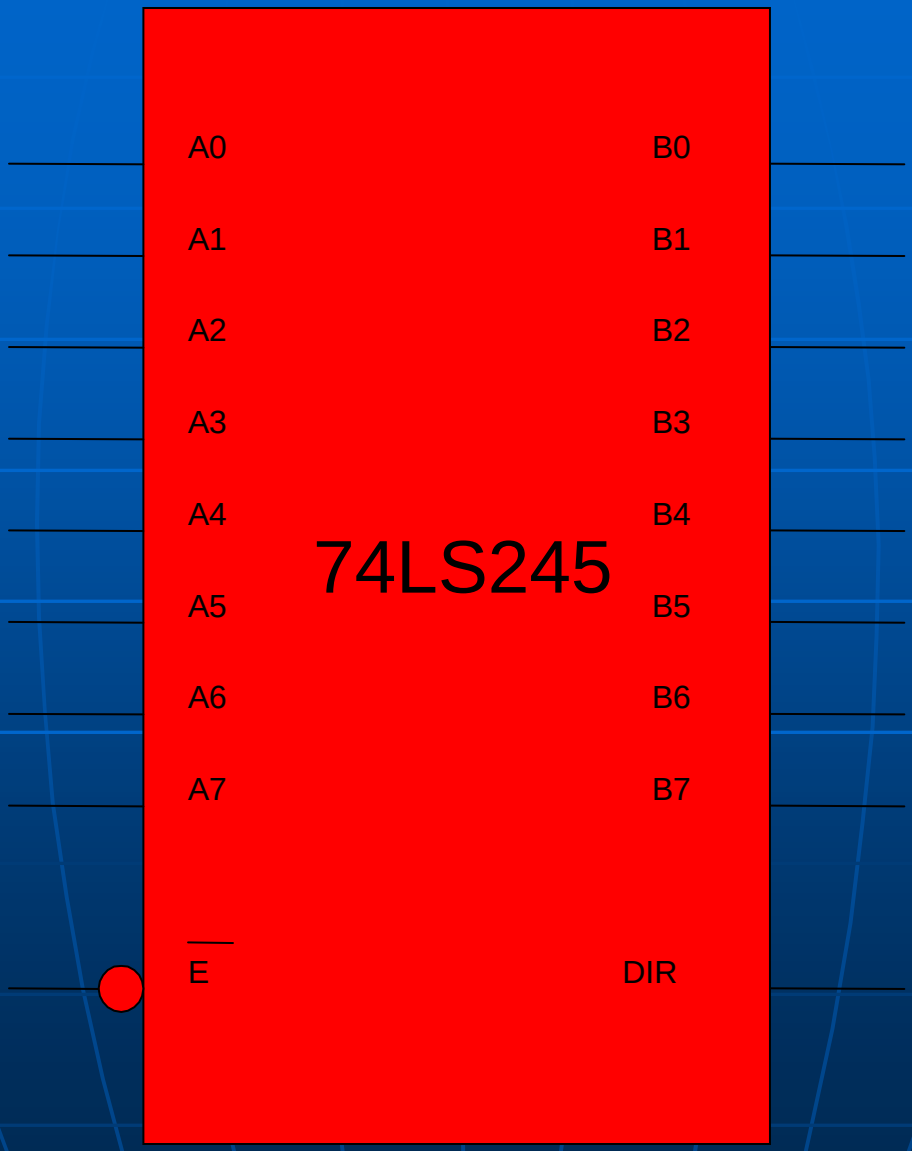
Tạo ra các tín hiệu điều khiển (Min Mode)



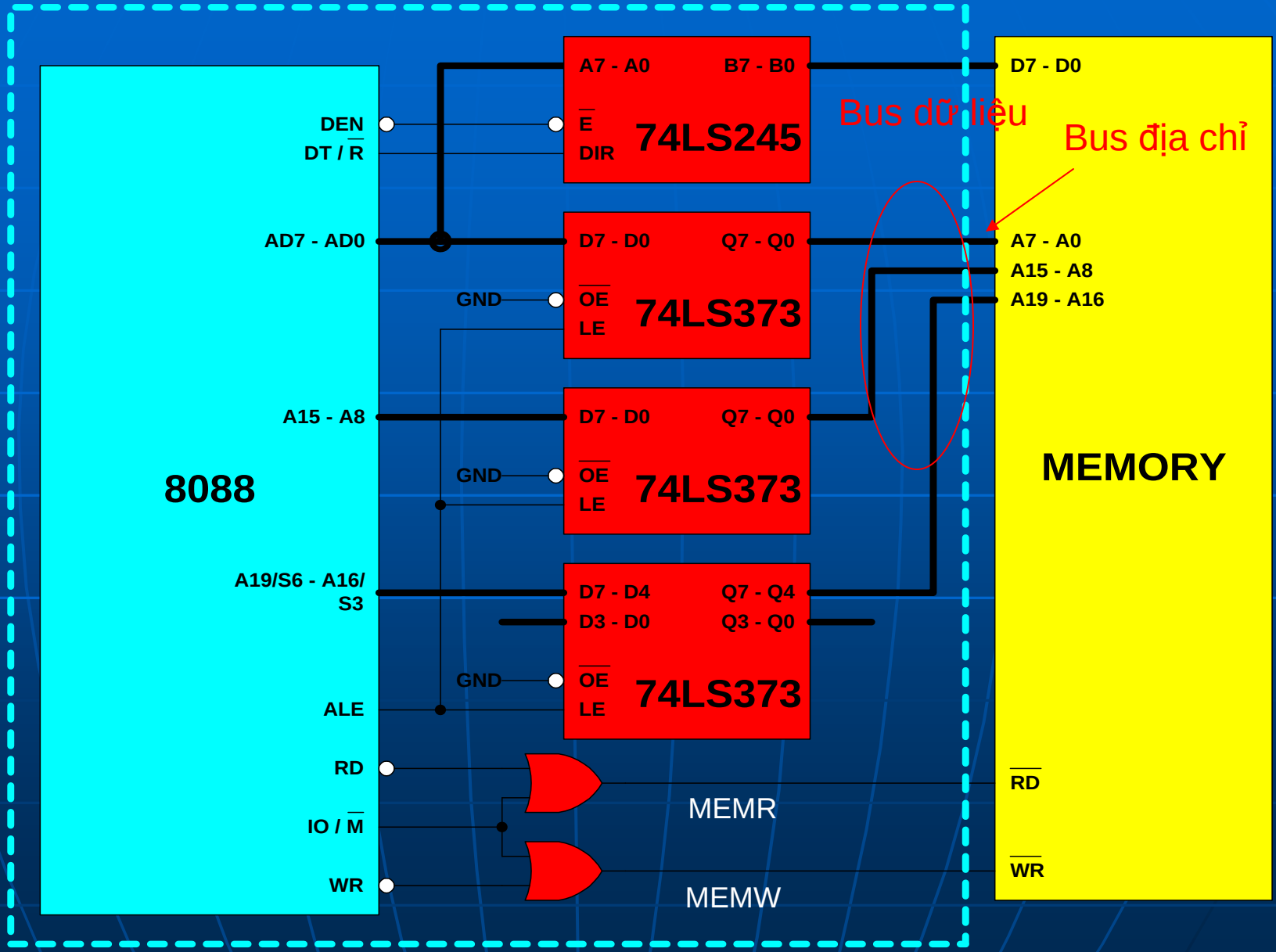
8088 Bus – Min Mode



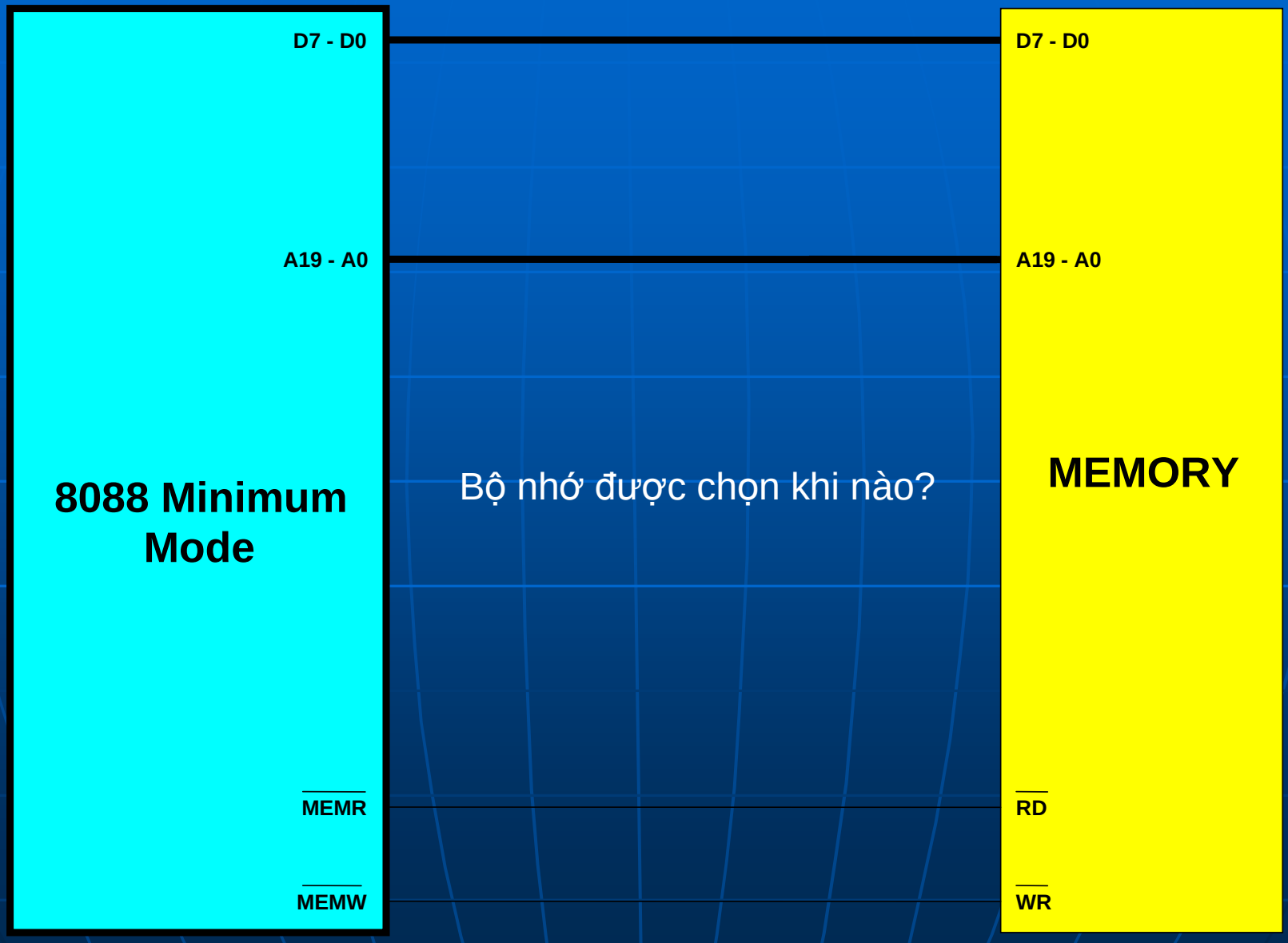
74LS245



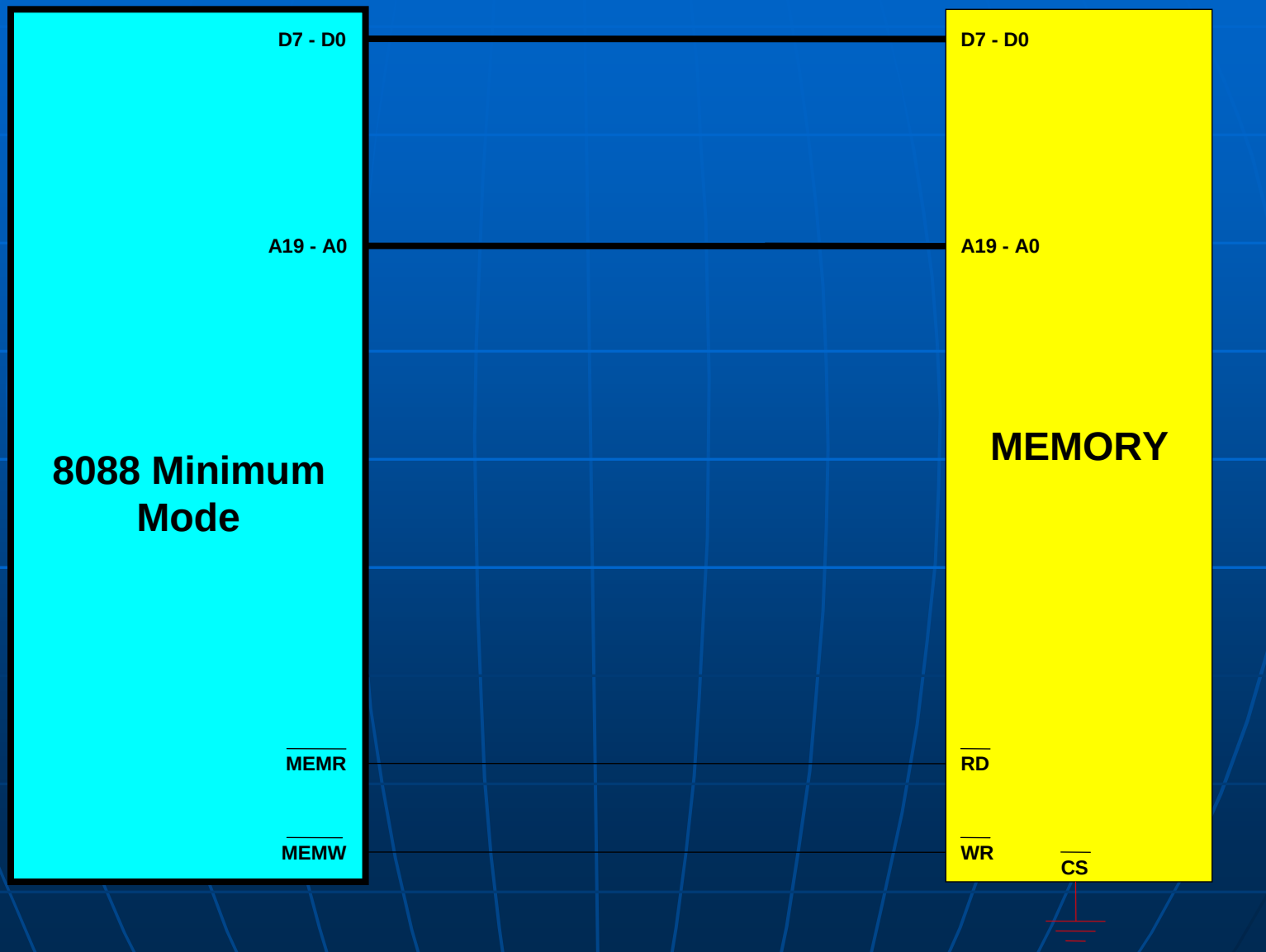
Bus hệ thống của hệ 8088 ở Mode Minimum



Minimum Mode 8088



Minimum Mode 8088



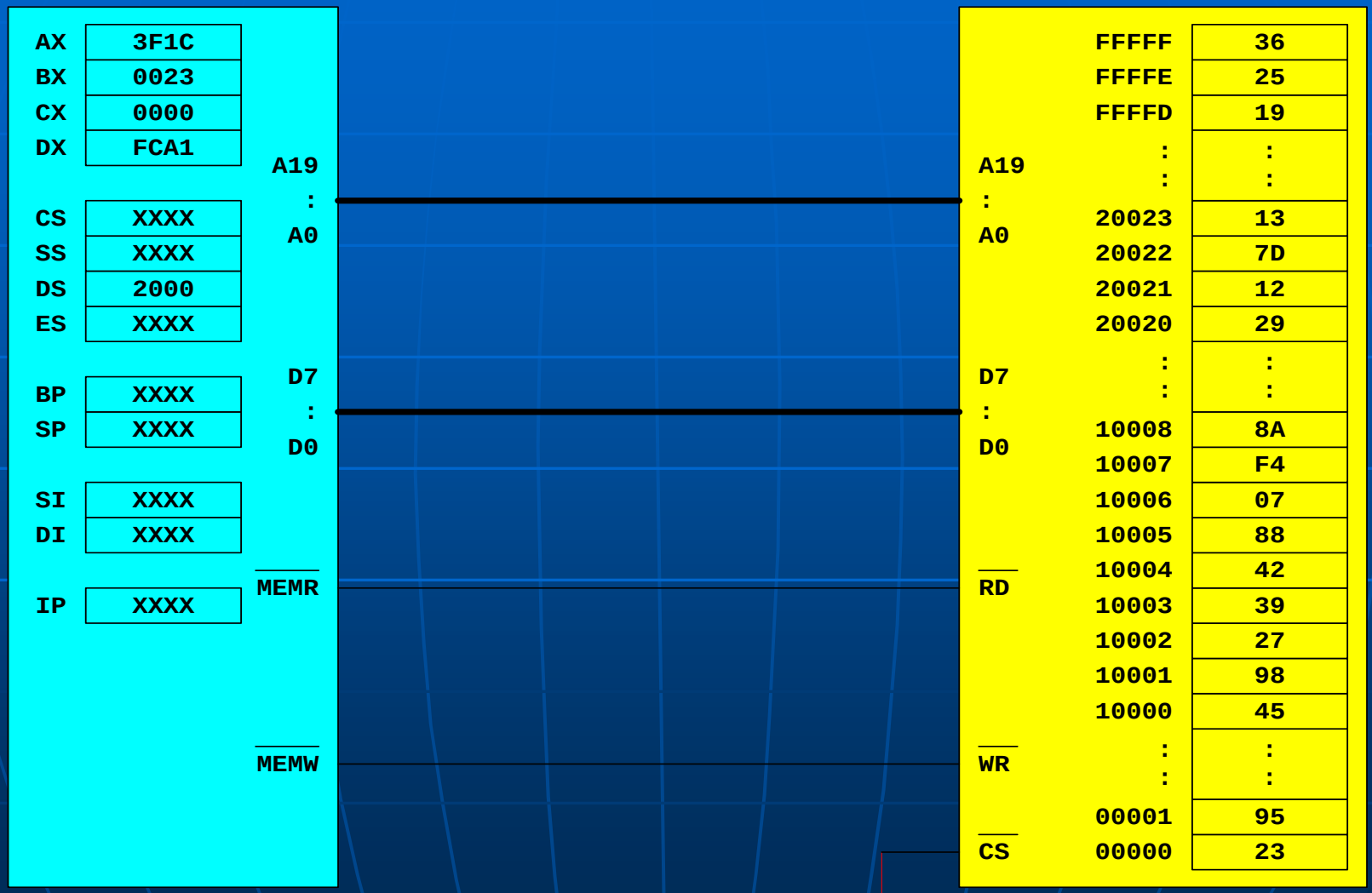
Không gian địa chỉ bộ nhớ 1M

A19 đến A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
00000	0000	0000	0000	0000	0000
FFFFF	1111	1111	1111	1111	1111

Ví dụ: Một địa chỉ bất kỳ 34FD0h

0011 0100 11111 1101 0000

Bộ nhớ đầy đủ 1MB



Nếu chỉ cần bộ nhớ có dung lượng nhỏ hơn 1MB thì giải quyết như thế nào?

- *Phụ thuộc vào các chip nhớ sẵn có*
- *Phụ thuộc yêu cầu phân bố địa chỉ cho các loại bộ nhớ vật lý khác nhau*
- *…*

512K đầu tiên của không gian địa chỉ bộ nhớ (Các địa chỉ có bit cao nhất A19 = 0)

A18 đến A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
00000	0000	0000	0000	0000	0000
7FFFF	0111	1111	1111	1111	1111

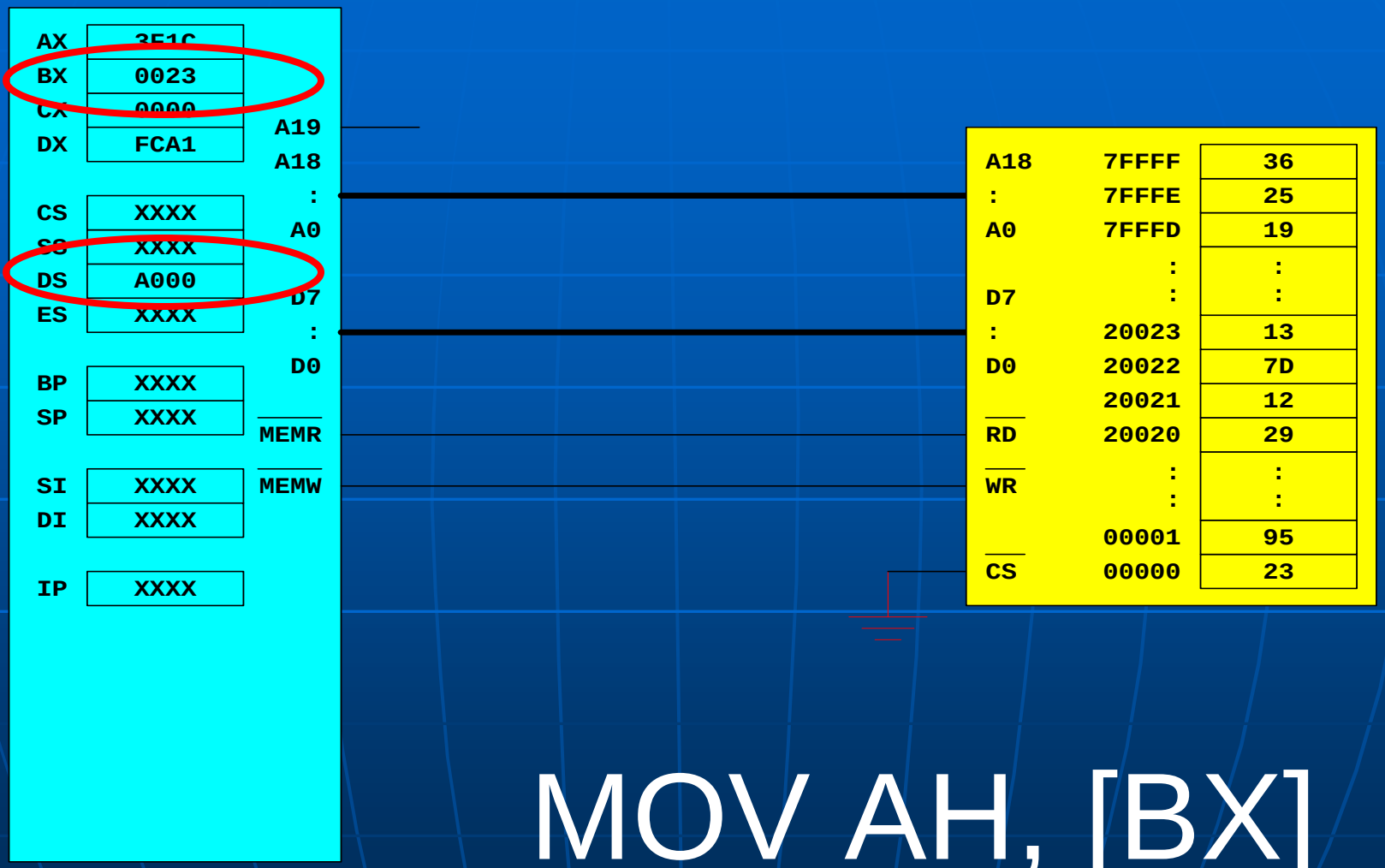
512K tiếp theo của không gian địa chỉ bộ nhớ (Các địa chỉ có bit cao nhất A19 = 1)

A18 đến A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
80000	1000	0000	0000	0000	0000
FFFFF	1111	1111	1111	1111	1111

Bộ nhớ 512KB


AX 3F1C
BX 0023
CX 0000
DX FCA1
CS XXXX
SS XXXX
DS 2000
ES XXXX
BP XXXX
SP XXXX
SI XXXX
DI XXXX
IP XXXX
A19
A18
:
A0
D7
:
D0
MEMR
MEMW
Làm gì với A19?
A18
:
A0
D7
:
RD
WR
CS
7FFFF 36
7FFFE 25
7FFFD 19
20023 13
20022 7D
20021 12
20020 29
00001 95
00000 23

Điều gì xảy ra nếu 8088 đọc ô nhớ A0023h?



MOV AH, [BX]

Điều gì xảy ra nếu 8088 đọc ô nhớ A0023h?

A19 đến A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
A0023	1010	0000	0000	0010	0011

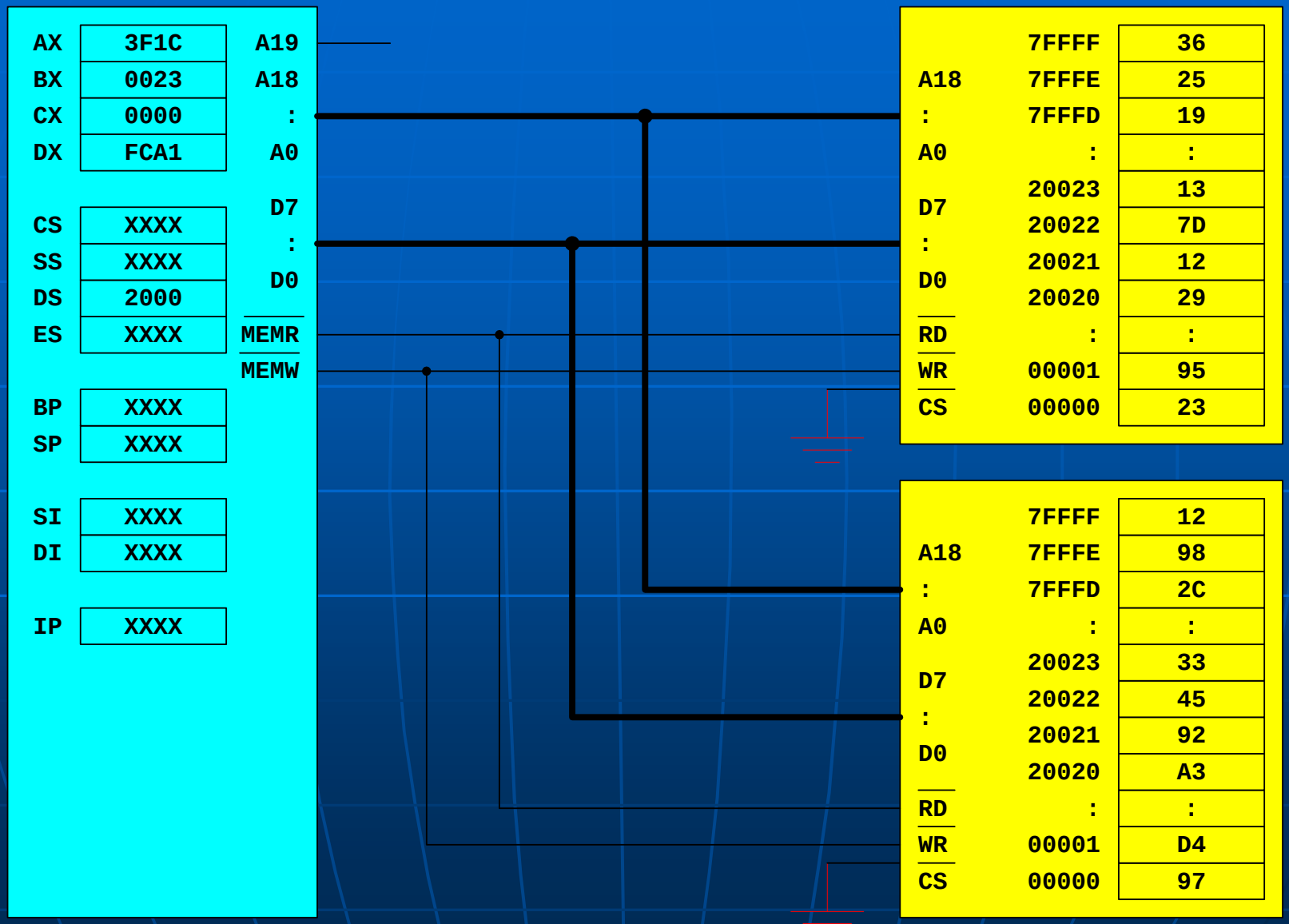
A19 không được nối đến bộ nhớ nên nếu 8088 phát logic "1" trên A19 thì bộ nhớ cũng không nhận biết được.

Điều gì xảy ra nếu 8088 đọc ô nhớ 20023h?

A18 đến A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
20023	0010	0000	0000	0010	0011

Với bộ nhớ tình hình không có gì khác!

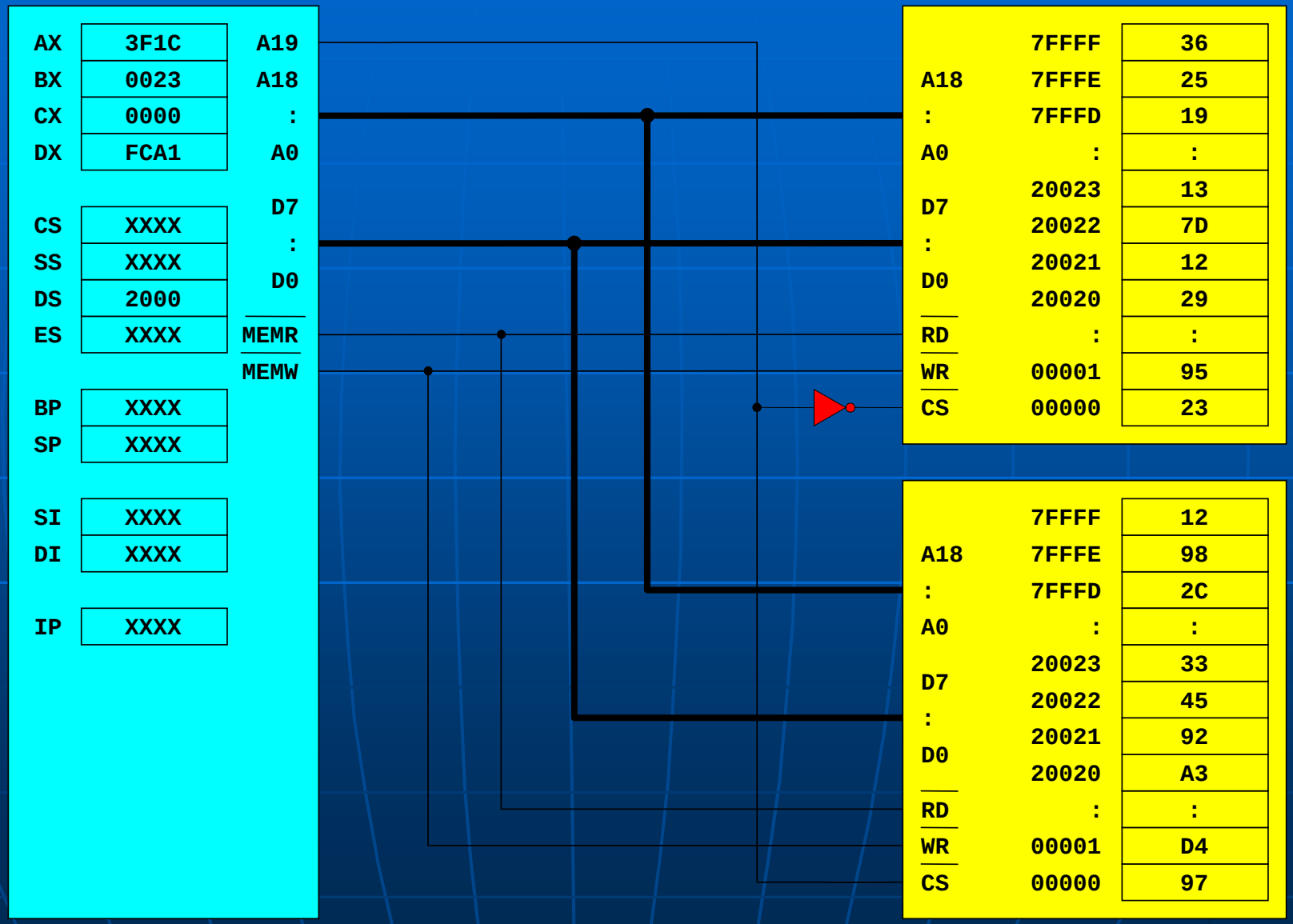
Nếu Bộ nhớ gồm 2 khối 512KB như thế này?



Có vấn đề !!!

- Vấn đề là: Xung đột Bus. Hai khối nhớ sẽ cung cấp dữ liệu cùng một lúc khi 8088 đọc bộ nhớ
- Giải pháp: Dùng A19 làm "người phân xử" để giải quyết xung đột trên bus. Nếu A19 ở mức logic "1" thì khối nhớ trên hoạt động (khối nhớ dưới bị cấm) và ngược lại

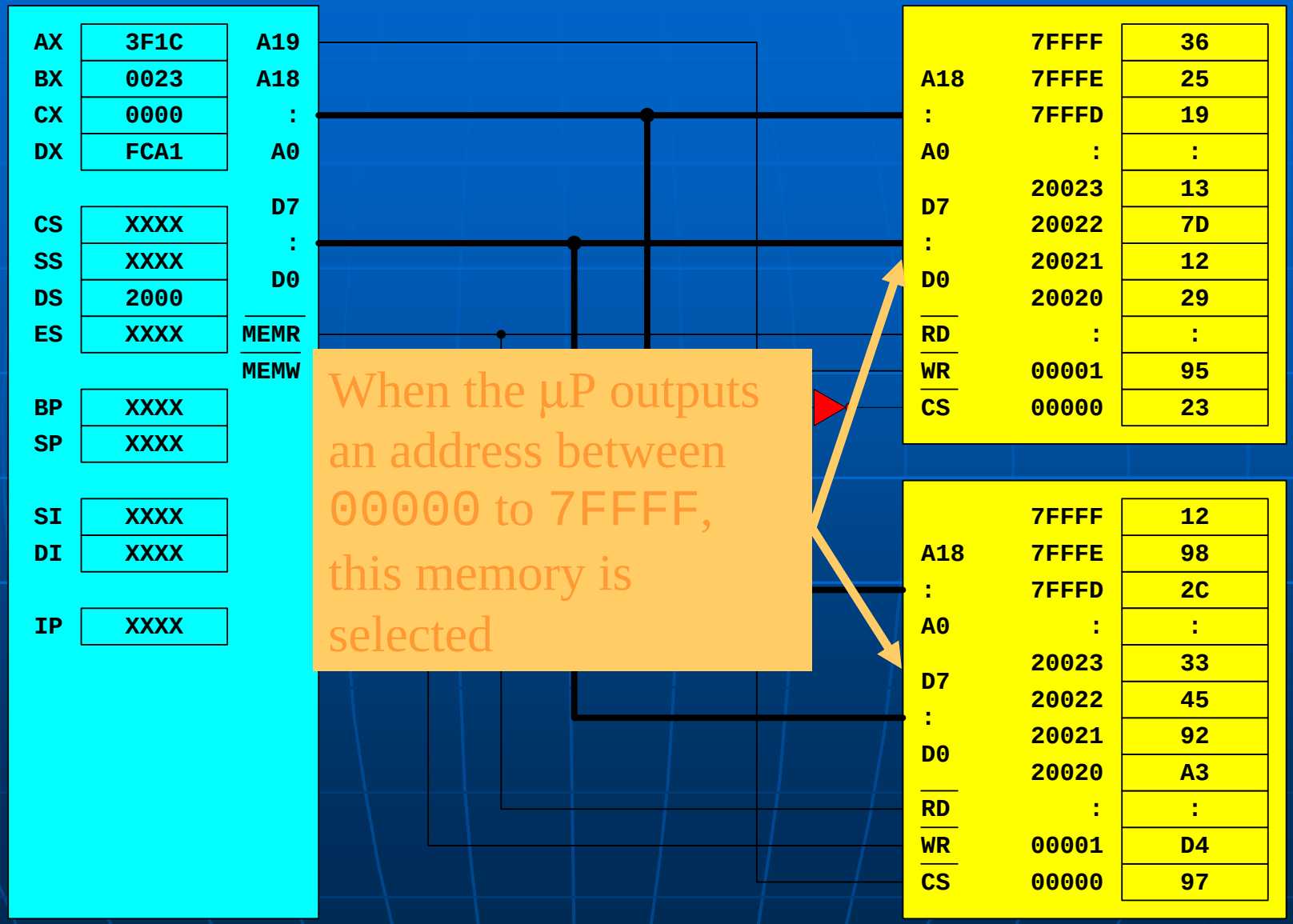
Bộ nhớ gồm hai khối nhớ 512KB



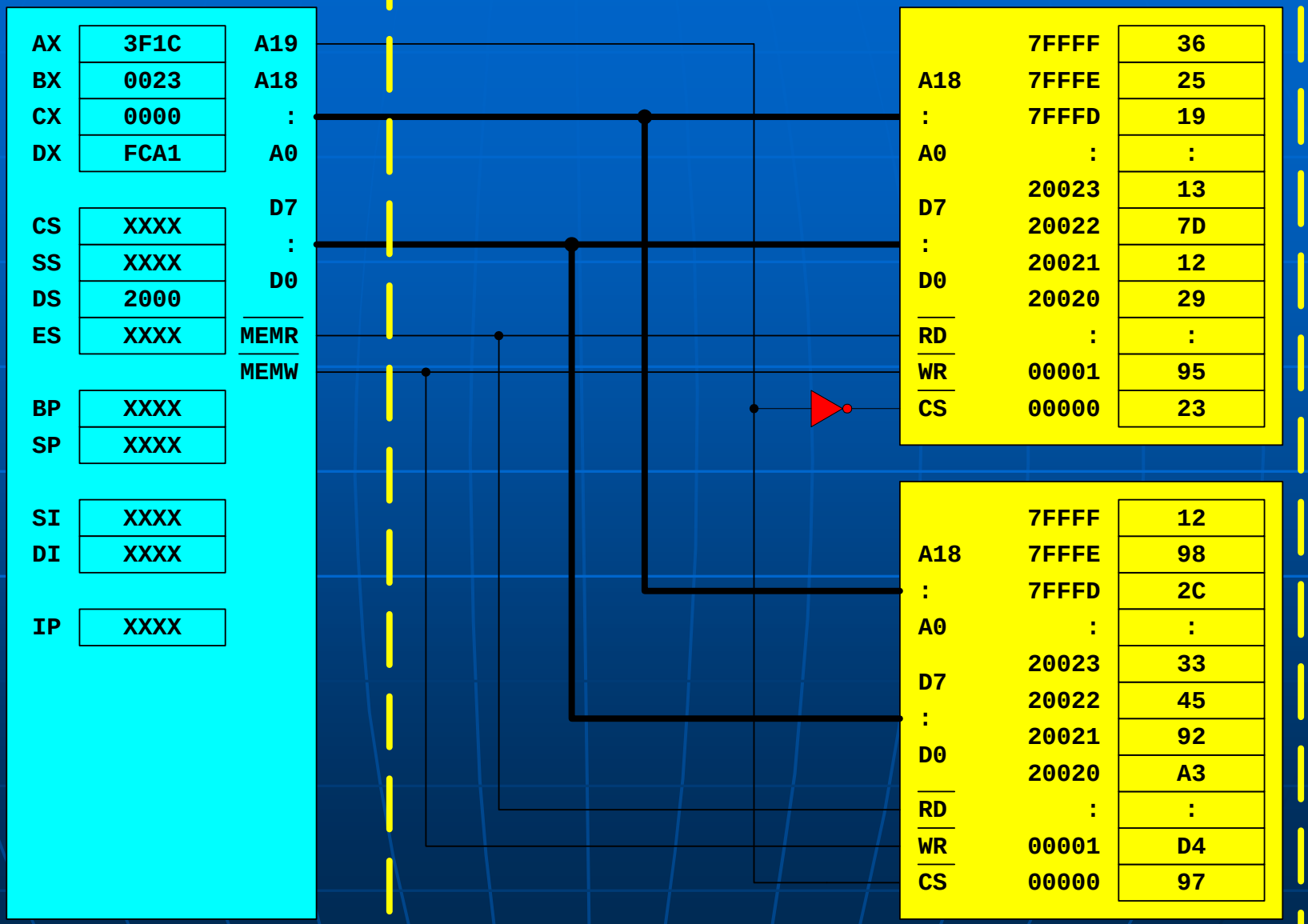
Không gian địa chỉ bộ nhớ 1M

A19 đến A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
00000	0000	0000	0000	0000	0000
7FFFF	0111	1111	1111	1111	1111
80000	1000	0000	0000	0000	0000
FFFFF	1111	1111	1111	1111	1111

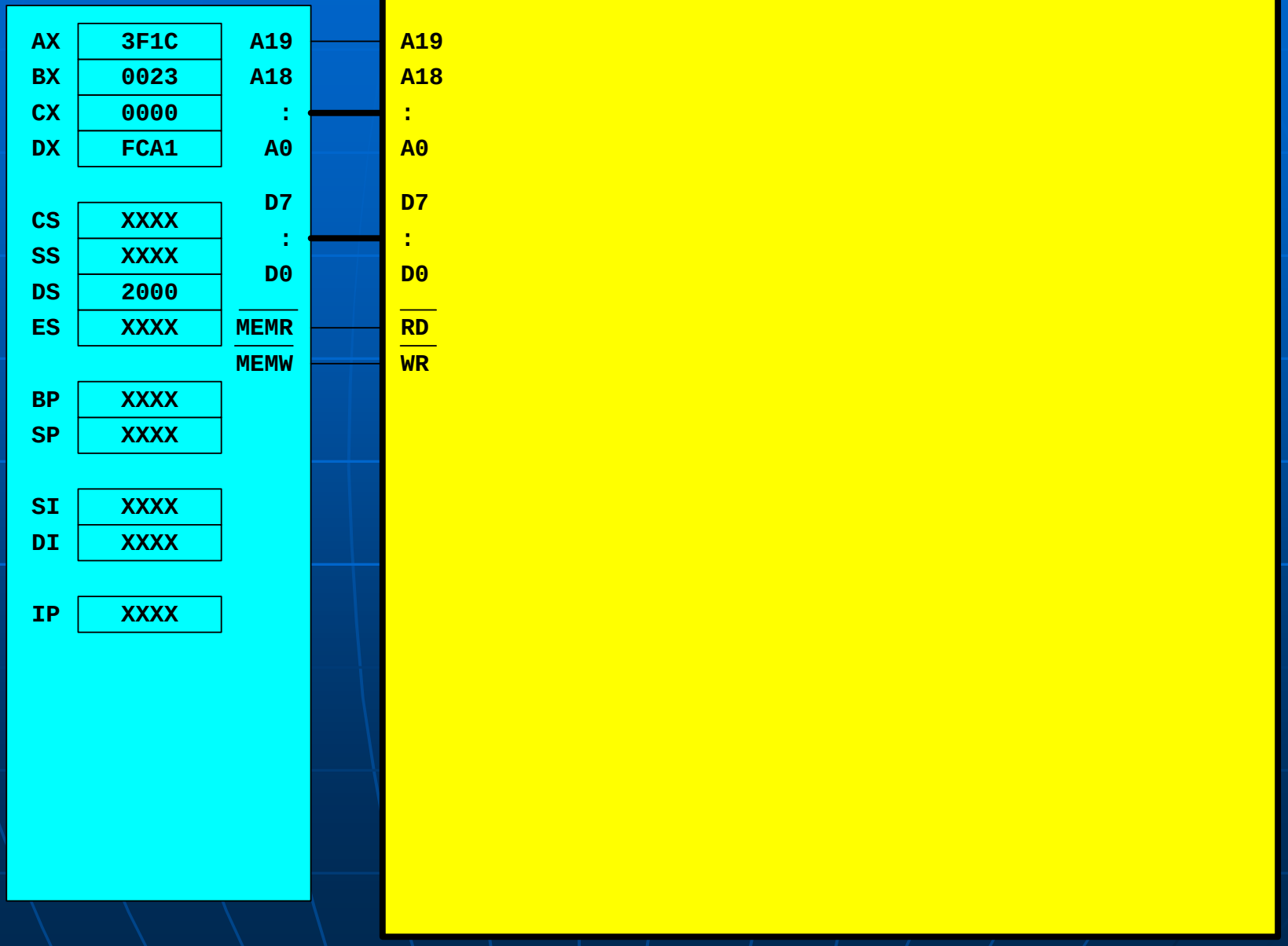
Interfacing **two** 512KB Memory to the 8088 Microprocessor



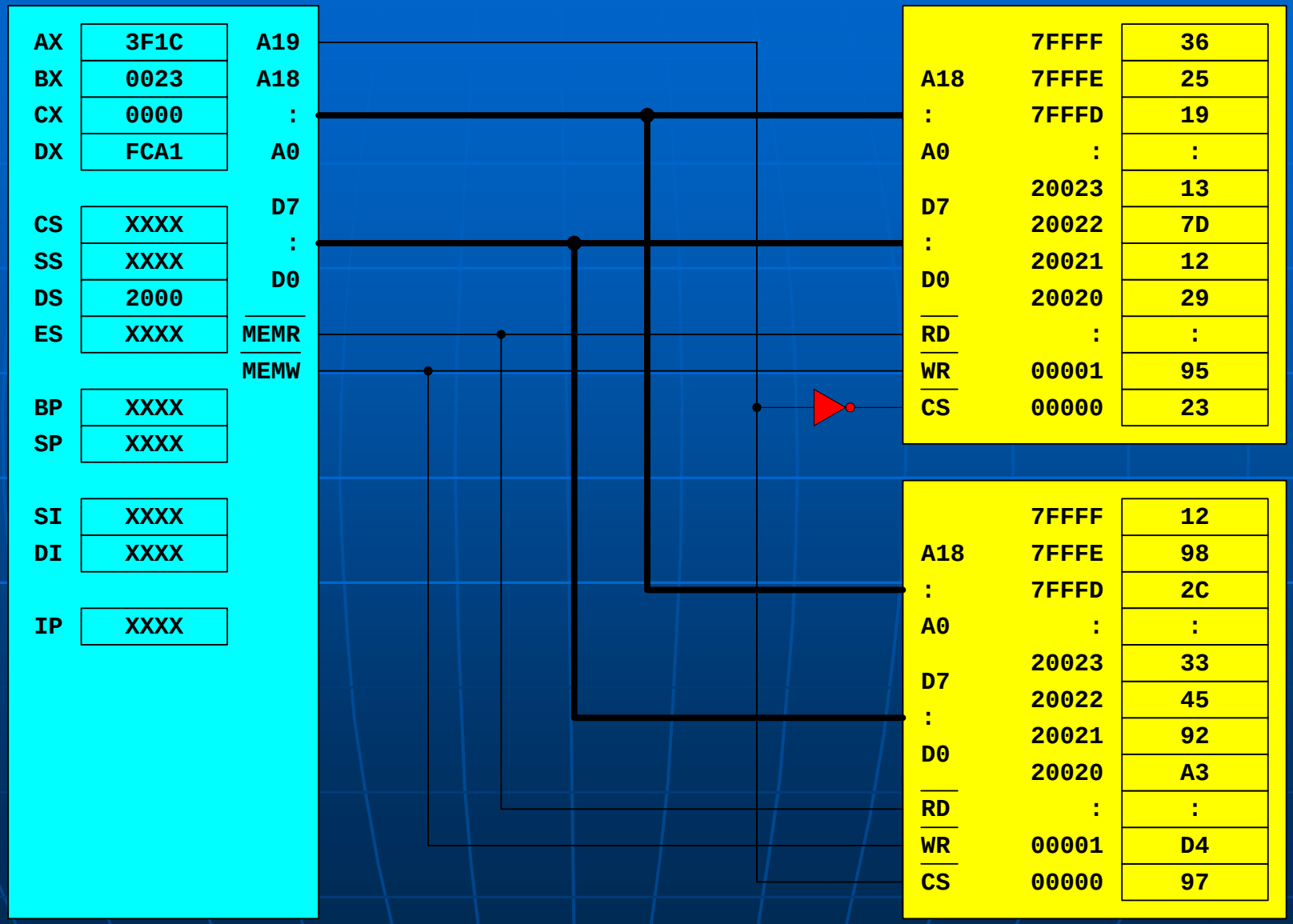
Interfacing **two** 512KB Memory to the 8088 Microprocessor



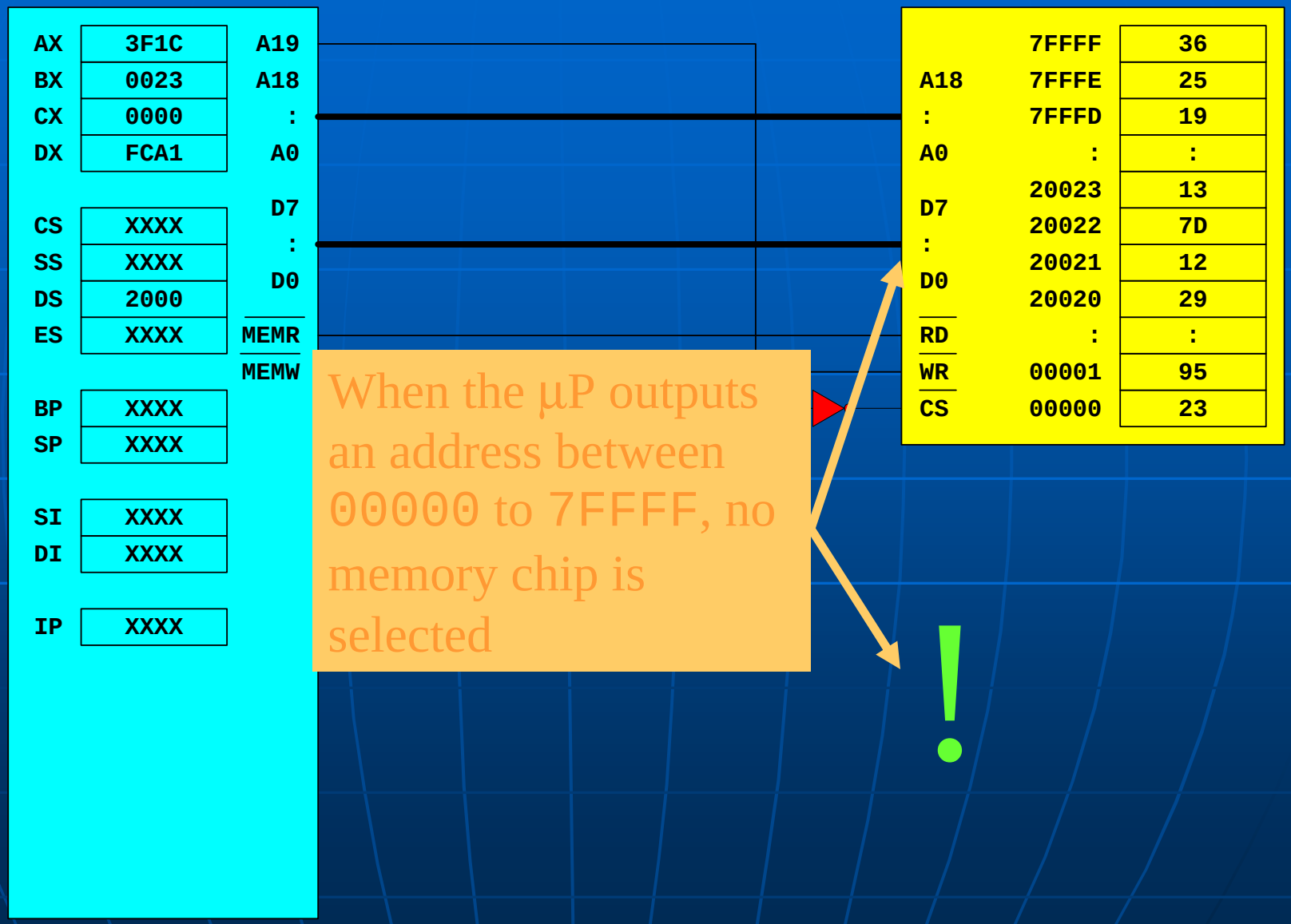
Interfacing **two** 512KB Memory to the 8088 Microprocessor



What if we remove the lower memory?



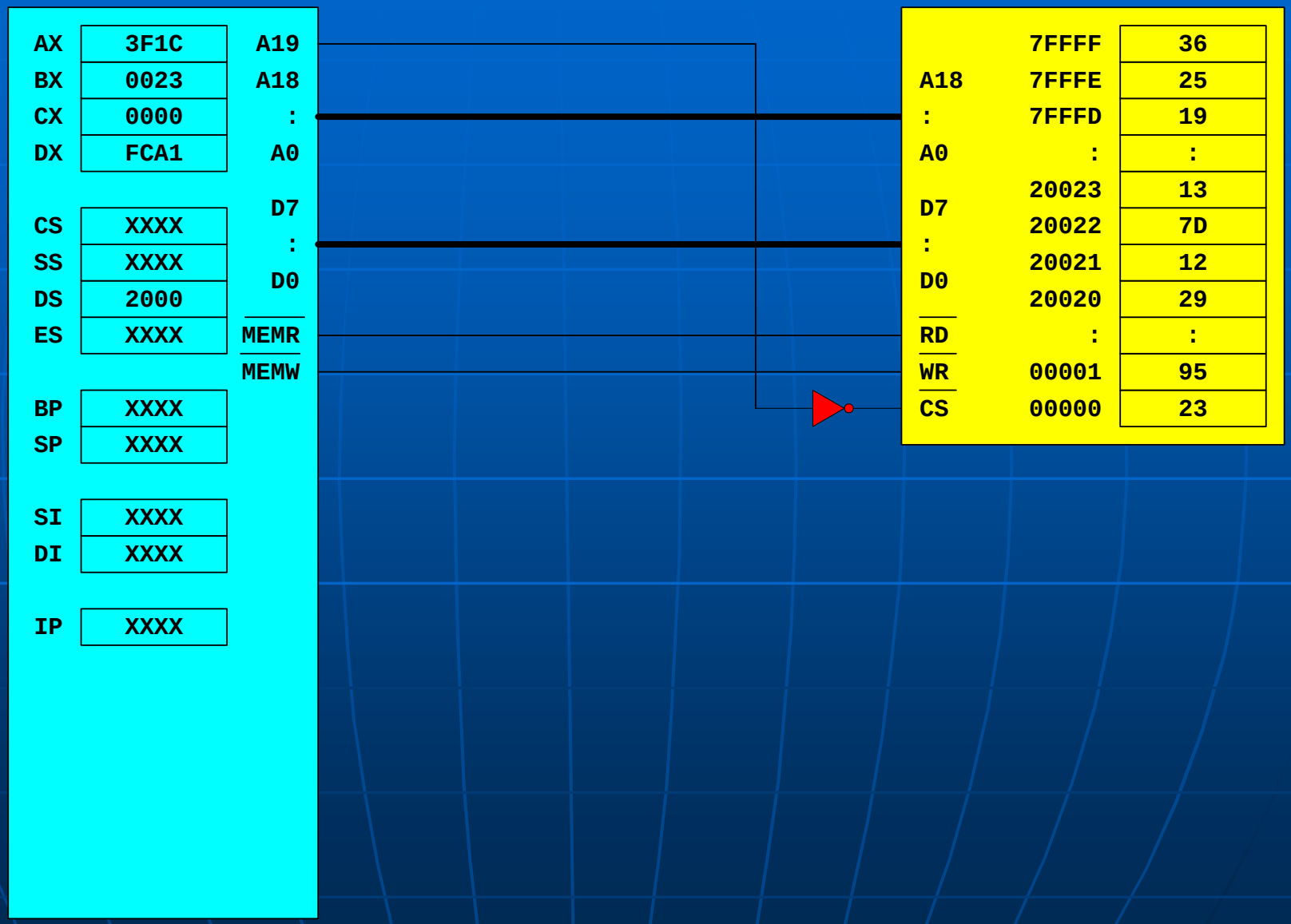
What if we remove the lower memory?



Full and Partial Decoding

- Full Decoding
  - When all of the "useful" address lines are connected the memory/device to perform selection
- Partial Decoding
  - When some of the "useful" address lines are connected the memory/device to perform selection
  - Using this type of decoding results into roll-over addresses

Full Decoding



Full Decoding

A19 to A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
80000	1000	0000	0000	0000	0000
FFFFF	1111	1111	1111	1111	1111

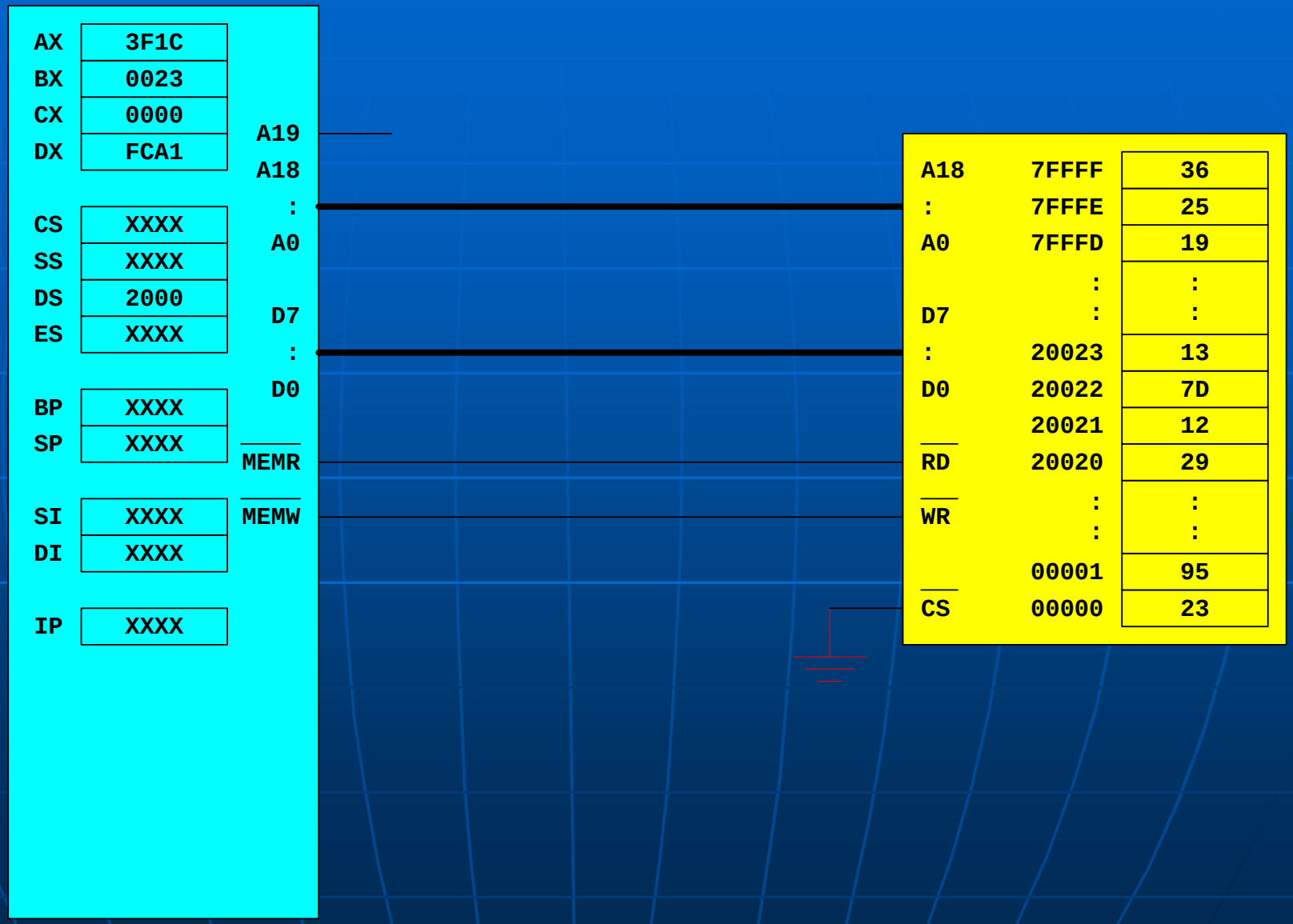
A19 should be a logic “1” for the memory chip to be enabled

Full Decoding

A19 to A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
00000	0000	0000	0000	0000	0000
7FFFF	0111	1111	1111	1111	1111

Therefore if the microprocessor outputs an address between 00000 to 7FFFF, whose A19 is a logic “0”, the memory chip will not be selected

Partial Decoding



Partial Decoding

A19 to A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
00000	0000	0000	0000	0000	0000
7FFFF	0111	1111	1111	1111	1111
80000	1000	0000	0000	0000	0000
FFFFF	1111	1111	1111	1111	1111

The value of A19 is INSIGNIFICANT to the memory chip, therefore A19 has no bearing whether the memory chip will be enabled or not

Partial Decoding

A19 to A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
00000	0000	0000	0000	0000	0000
7FFFF	0111	1111	1111	1111	1111
80000	1000	0000	0000	0000	0000
FFFFF	1111	1111	1111	1111	1111

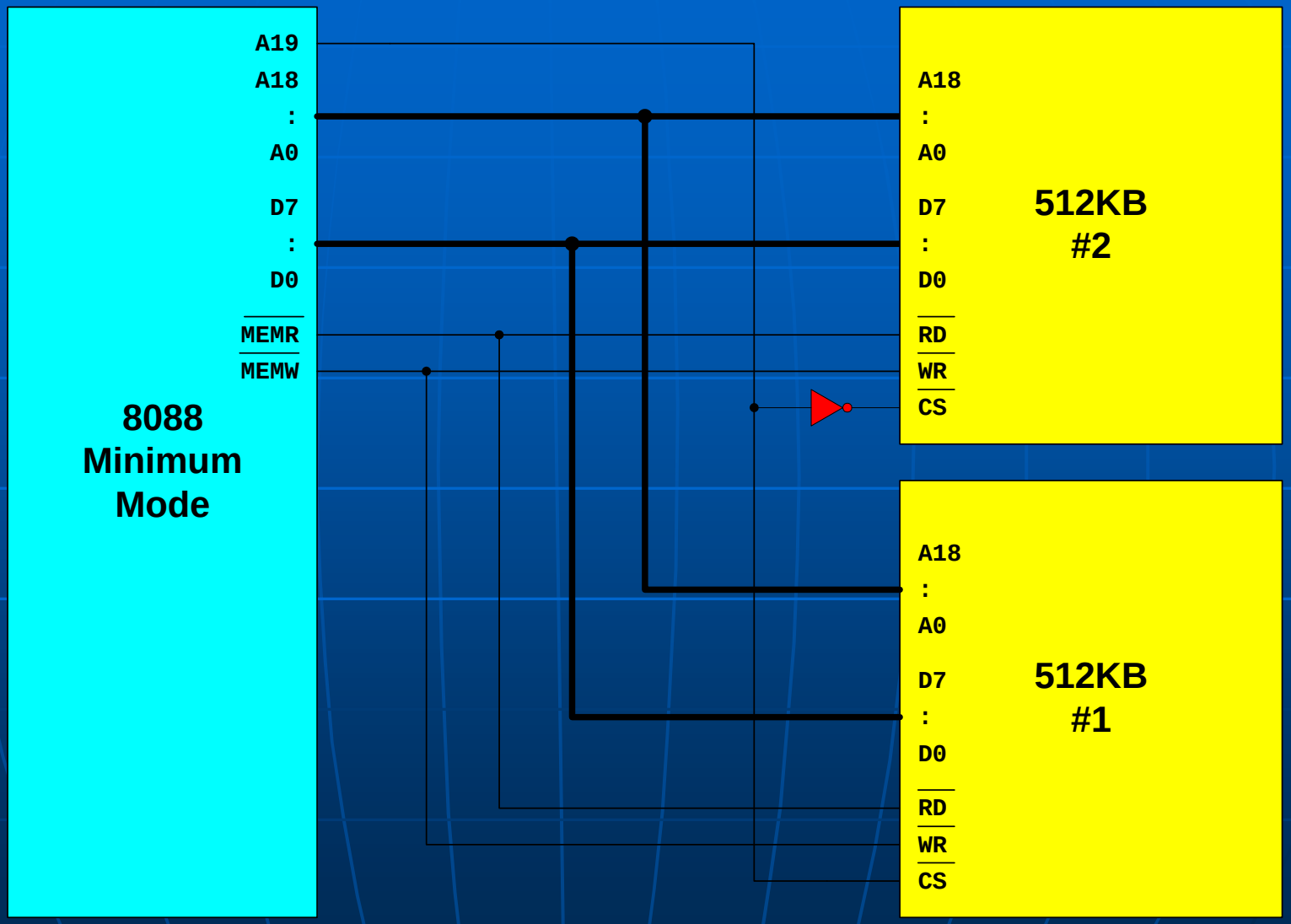
ACTUAL ADDRESS

Partial Decoding

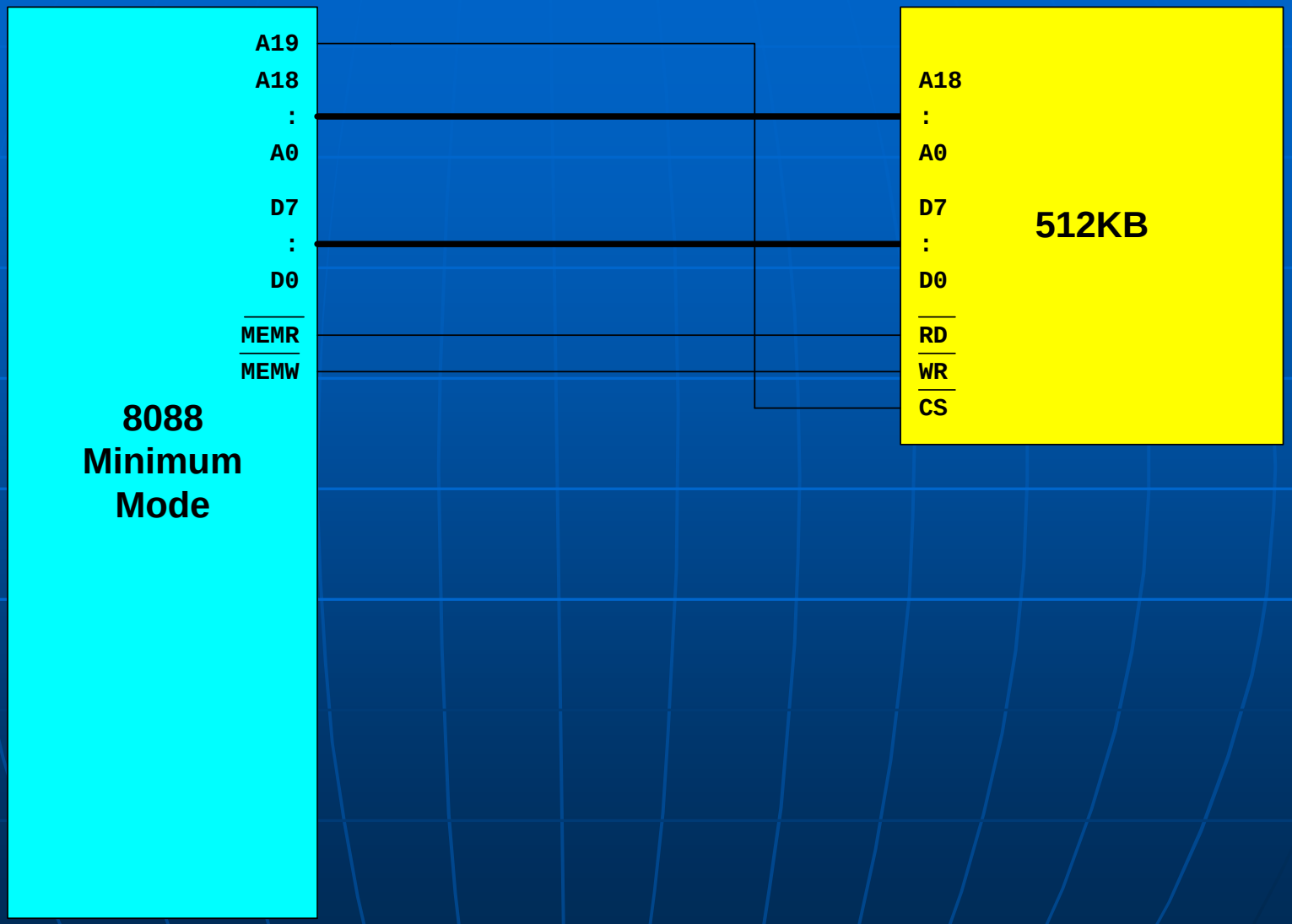
A19 to A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
00000	0000	0000	0000	0000	0000
7FFFF	0111	1111	1111	1111	1111
80000	1000	0000	0000	0000	0000
FFFFF	1111	1111	1111	1111	1111

ACTUAL ADDRESS

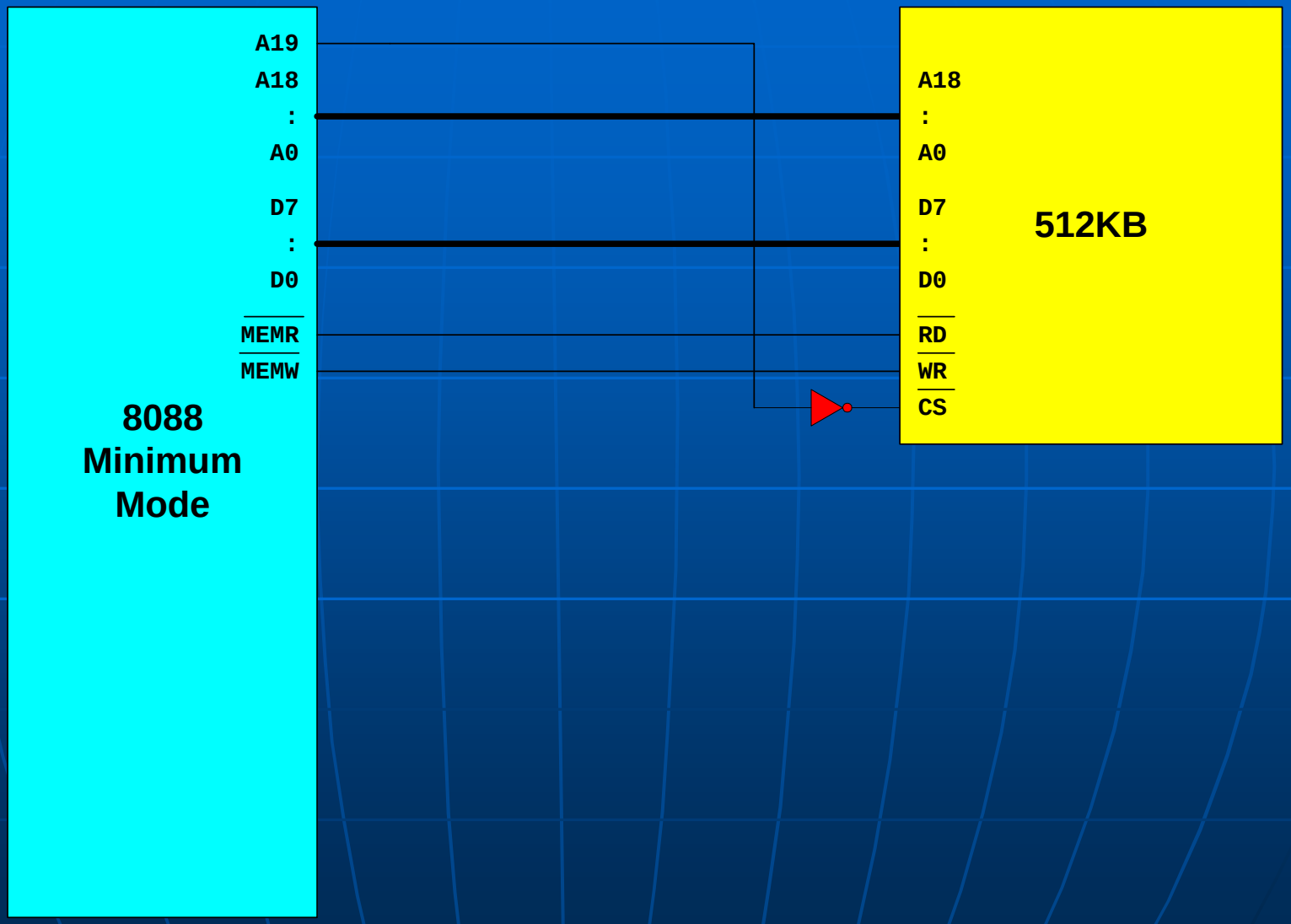
Interfacing two 512K Memory Chips to the 8088 Microprocessor



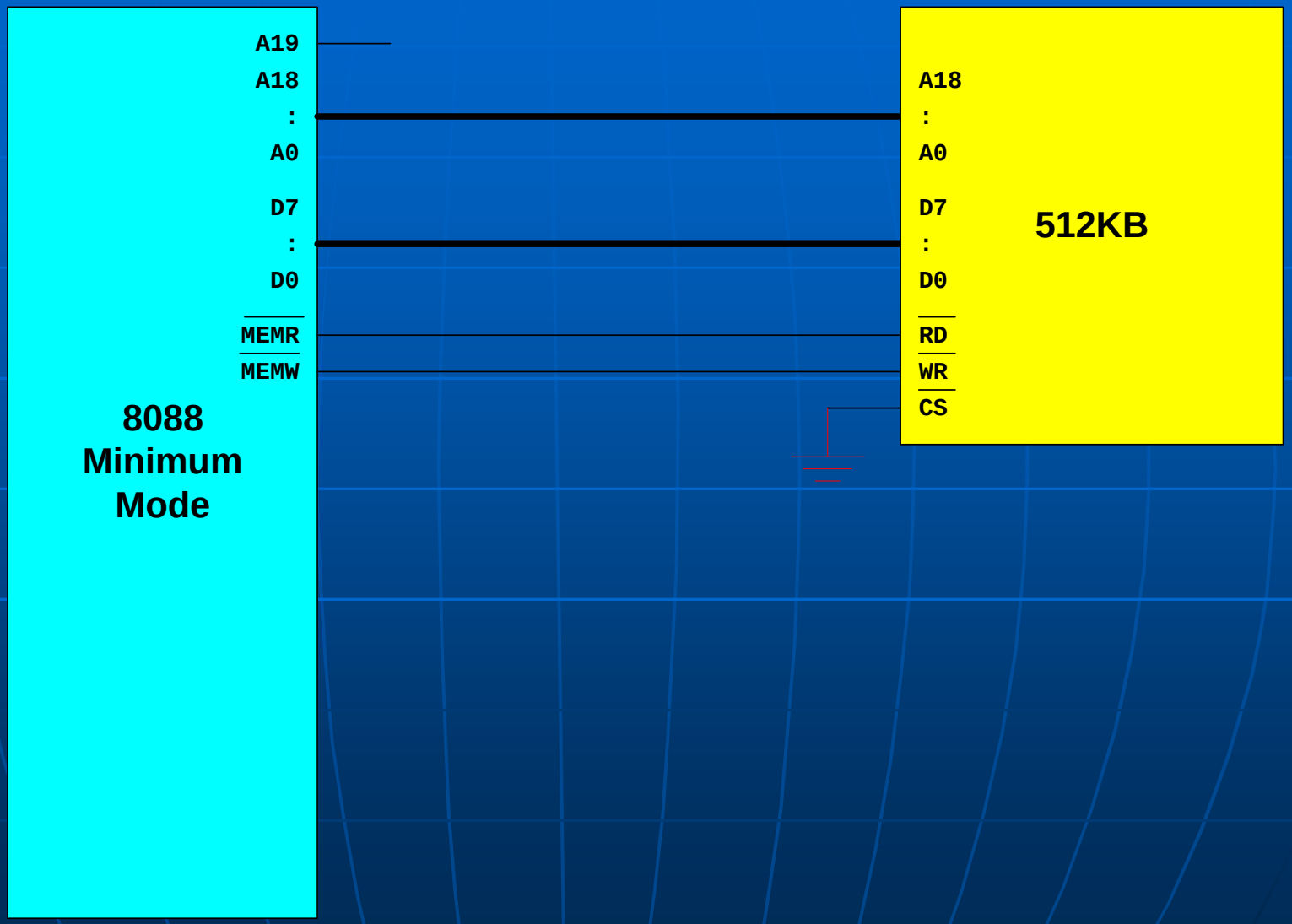
Interfacing one 512K Memory Chips to the 8088 Microprocessor



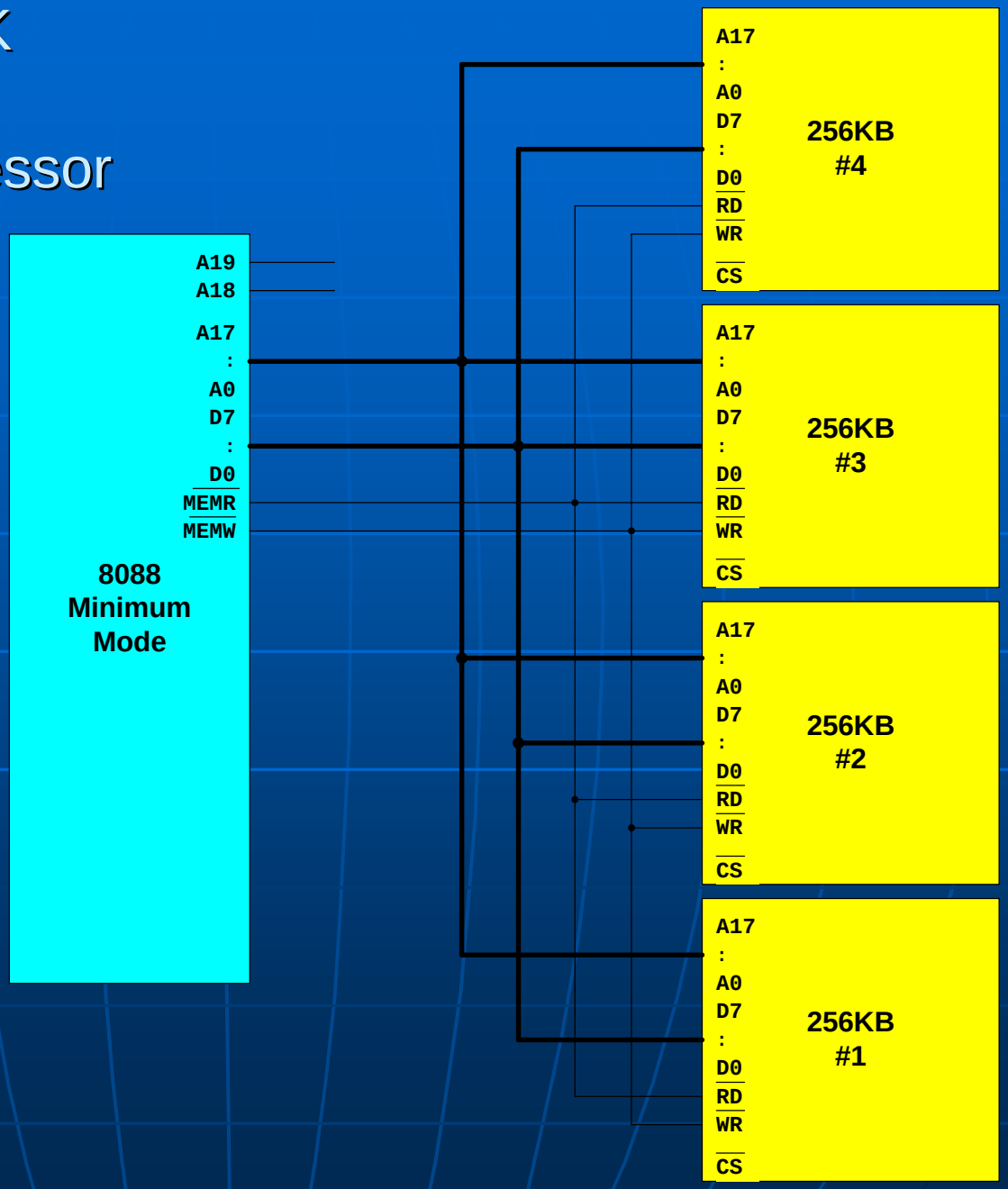
Interfacing one 512K Memory Chips to the 8088 Microprocessor (version 2)



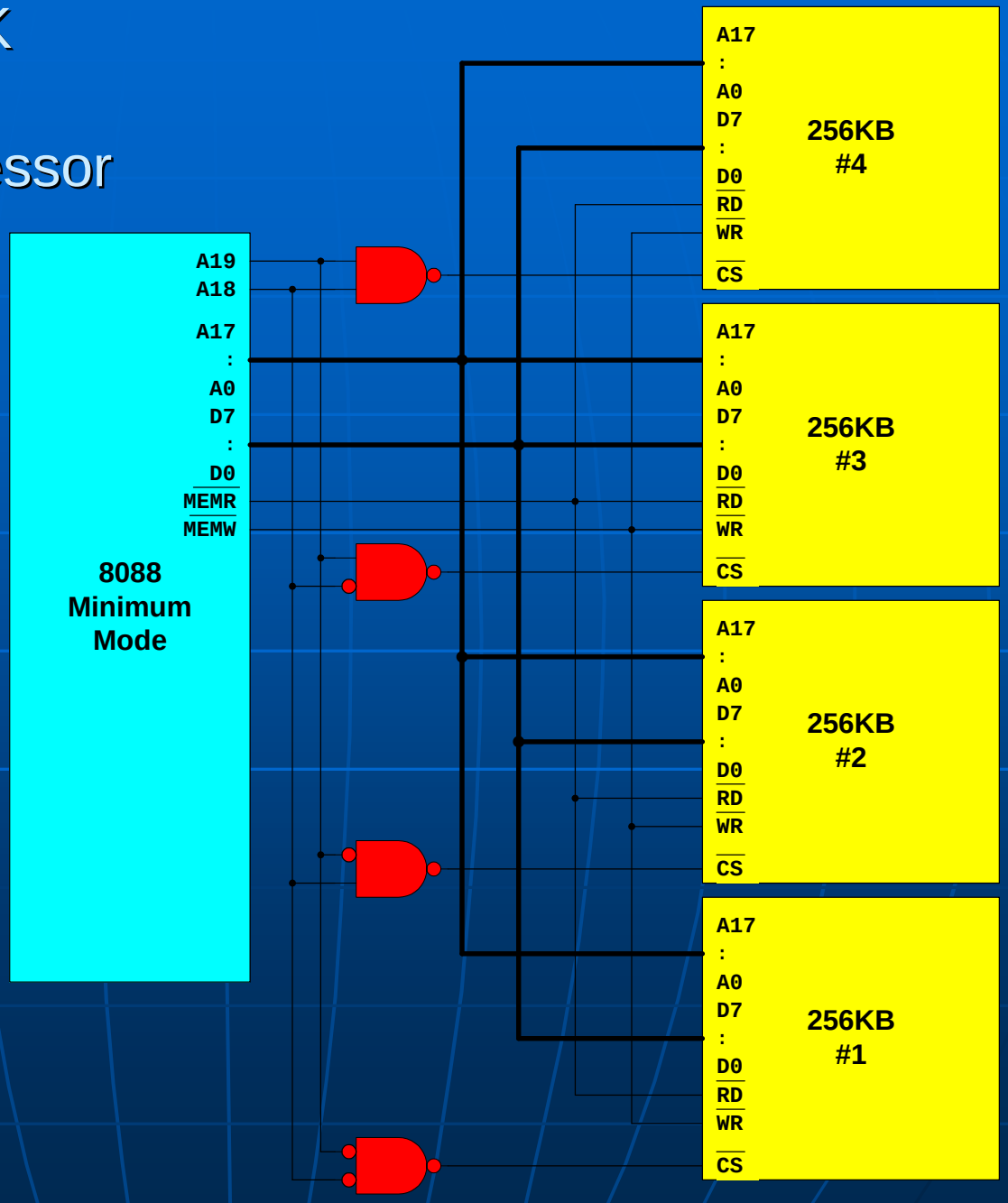
Interfacing one 512K Memory Chips to the 8088 Microprocessor (version 3)



Interfacing four 256K Memory Chips to the 8088 Microprocessor



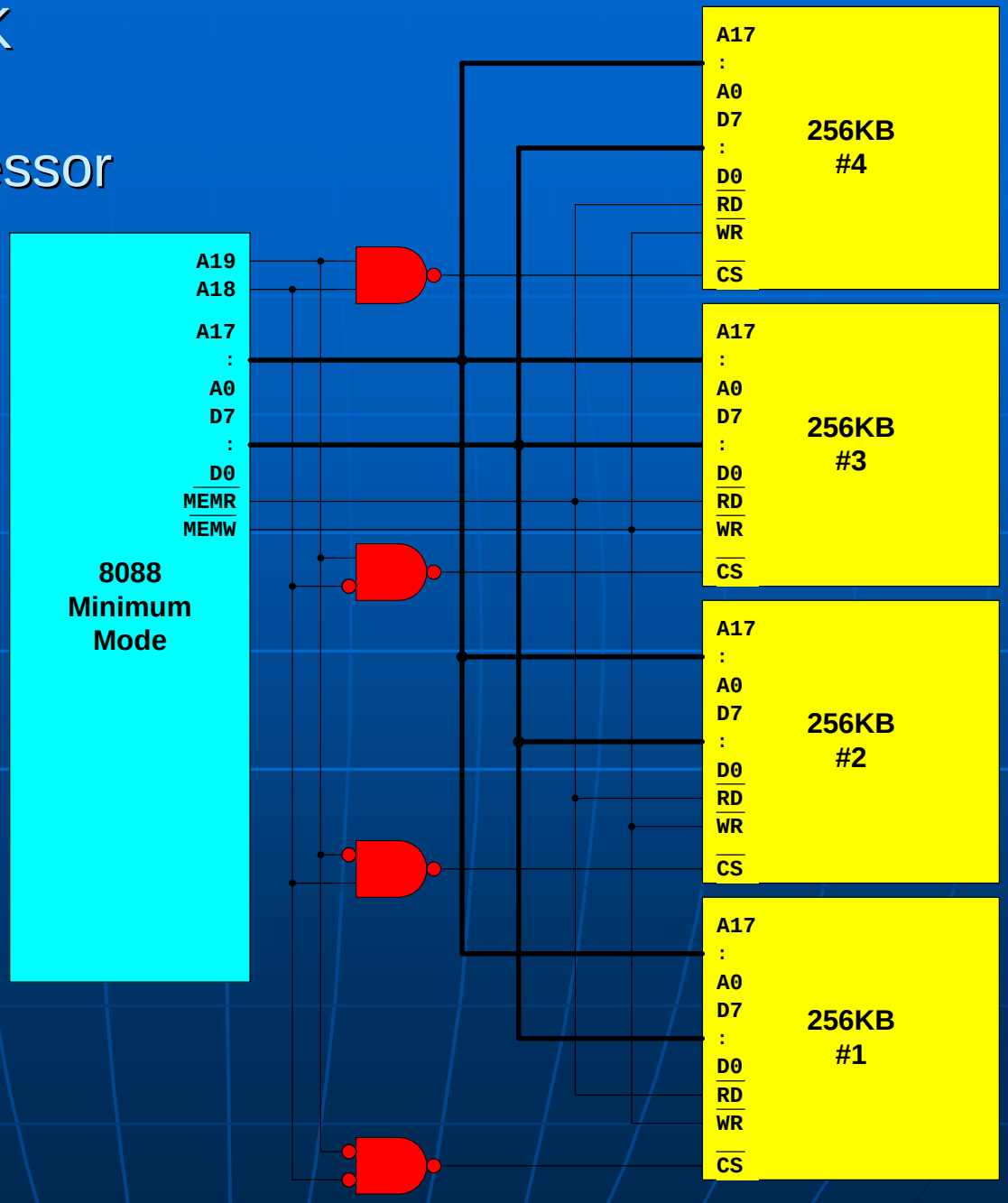
Interfacing four 256K Memory Chips to the 8088 Microprocessor



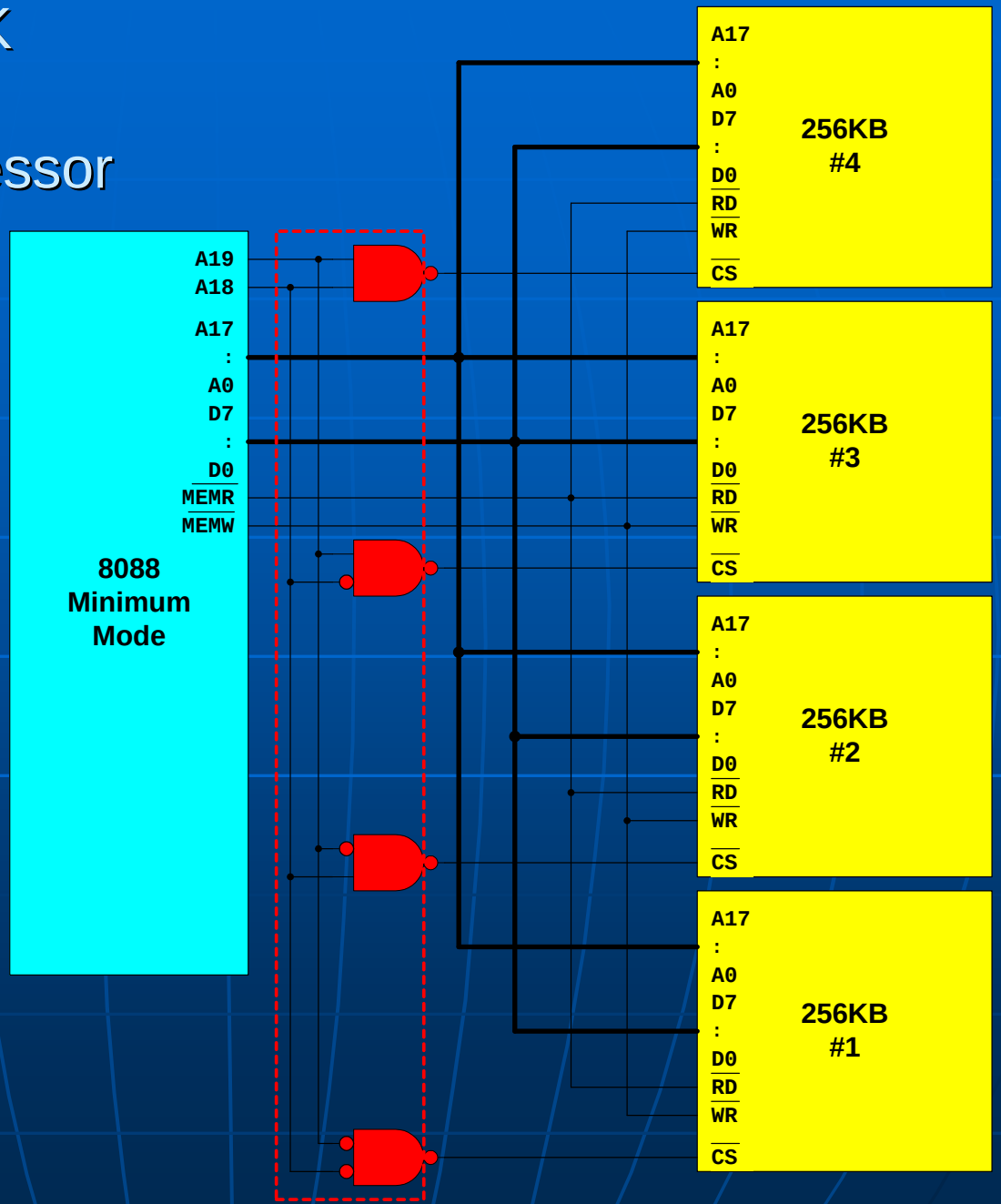
Memory chip#__ is mapped to:

A19 to A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
-----	----	----	----	----	----
-----	----	----	----	----	----

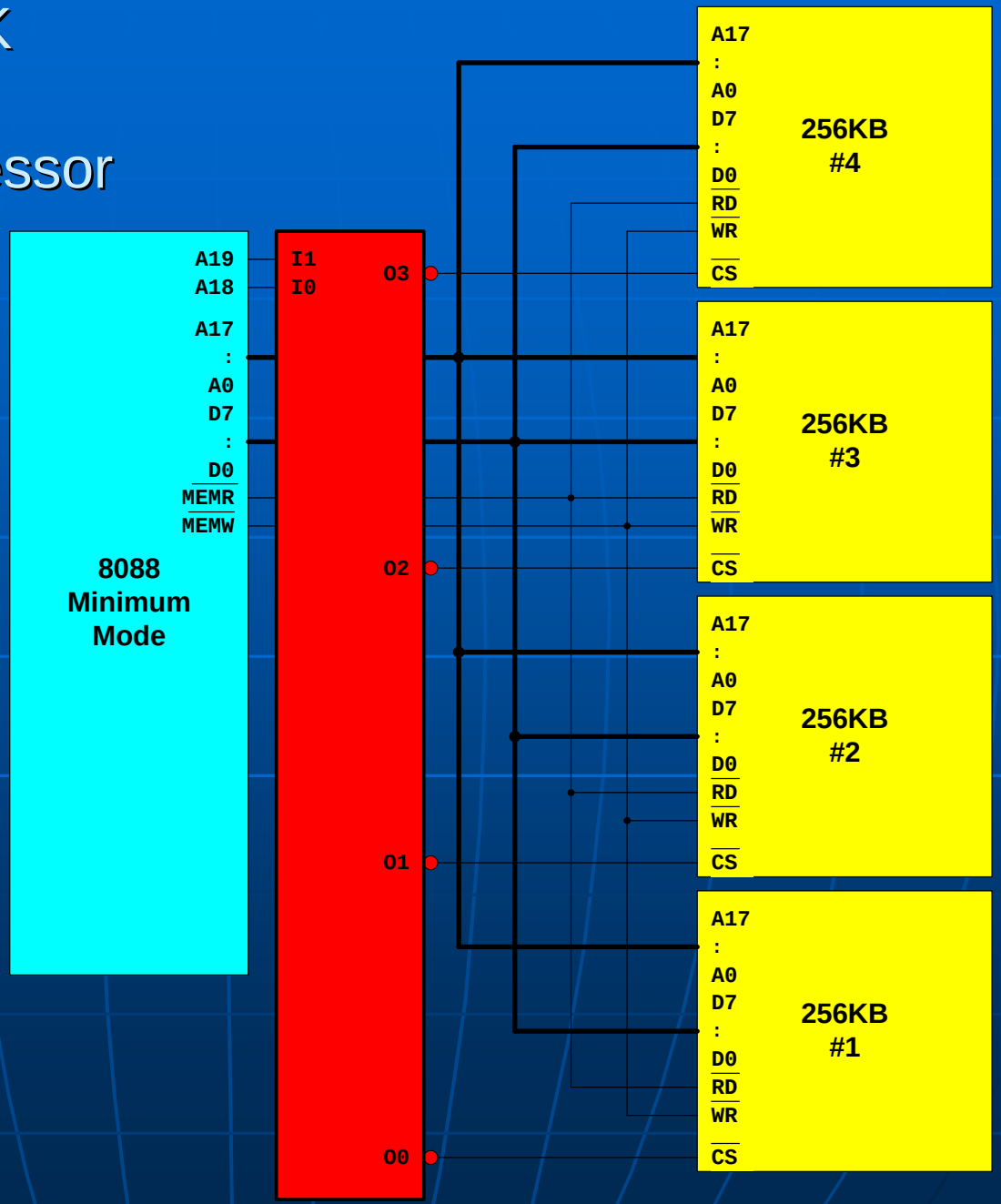
Interfacing four 256K Memory Chips to the 8088 Microprocessor



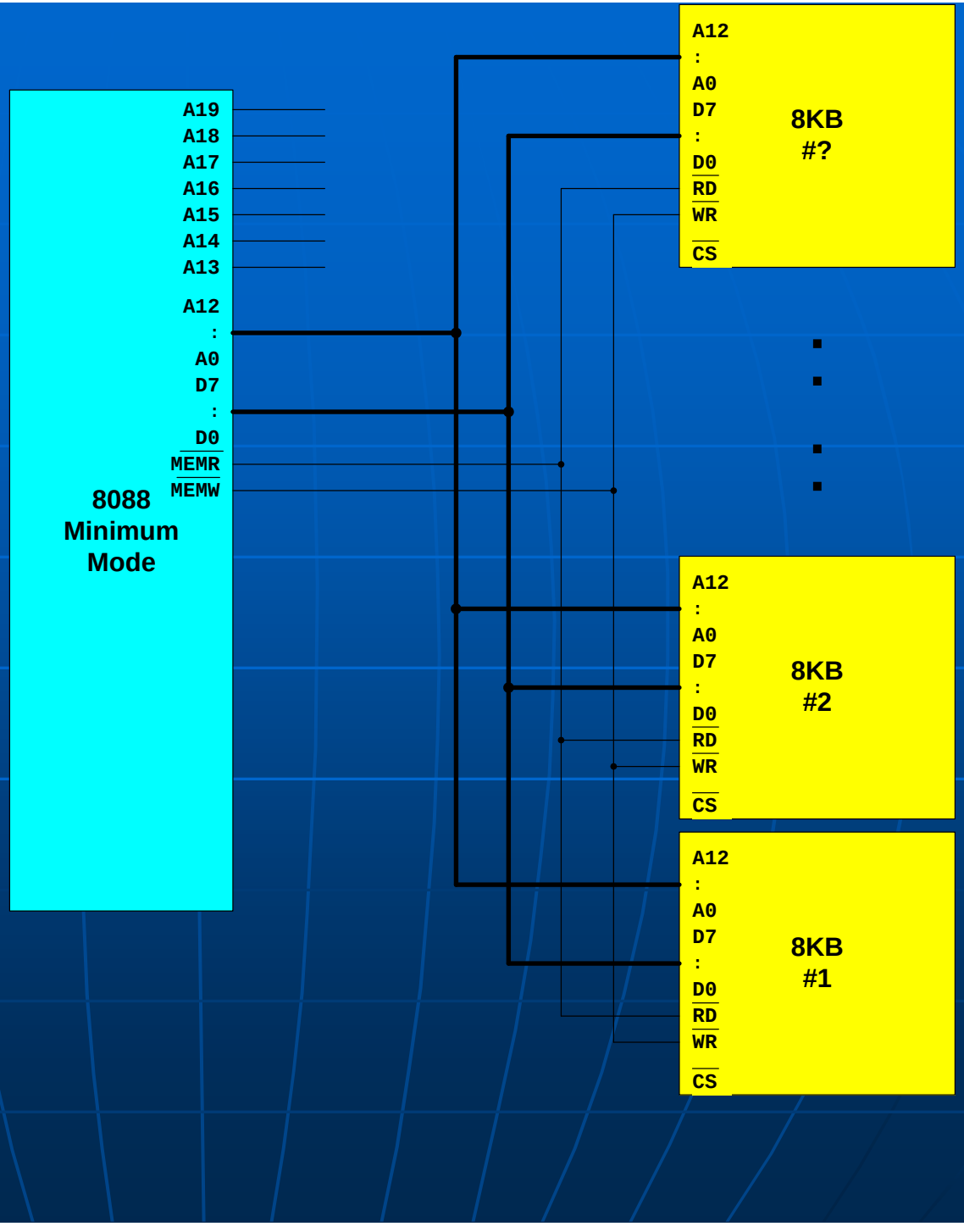
Interfacing four 256K Memory Chips to the 8088 Microprocessor



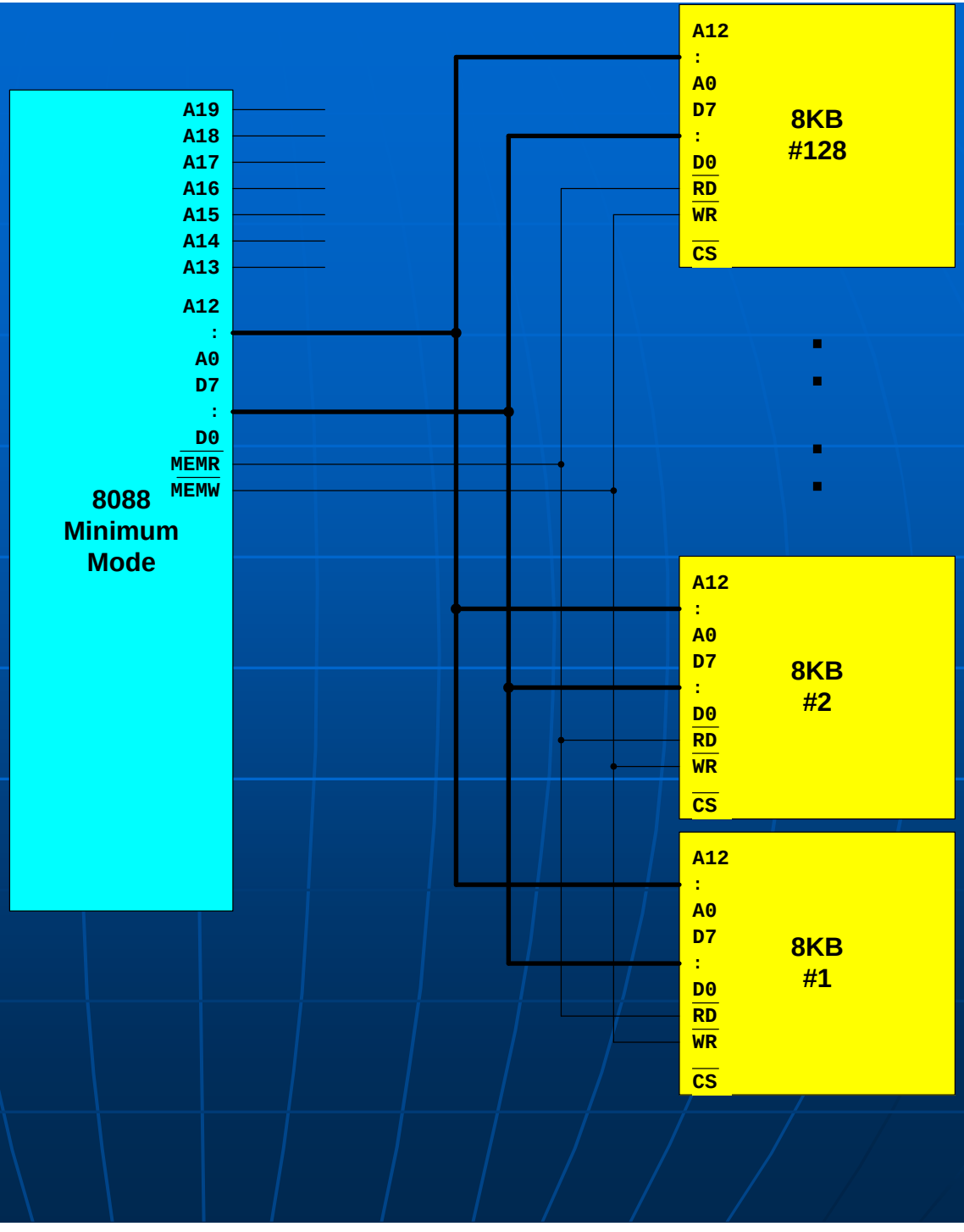
Interfacing four 256K Memory Chips to the 8088 Microprocessor



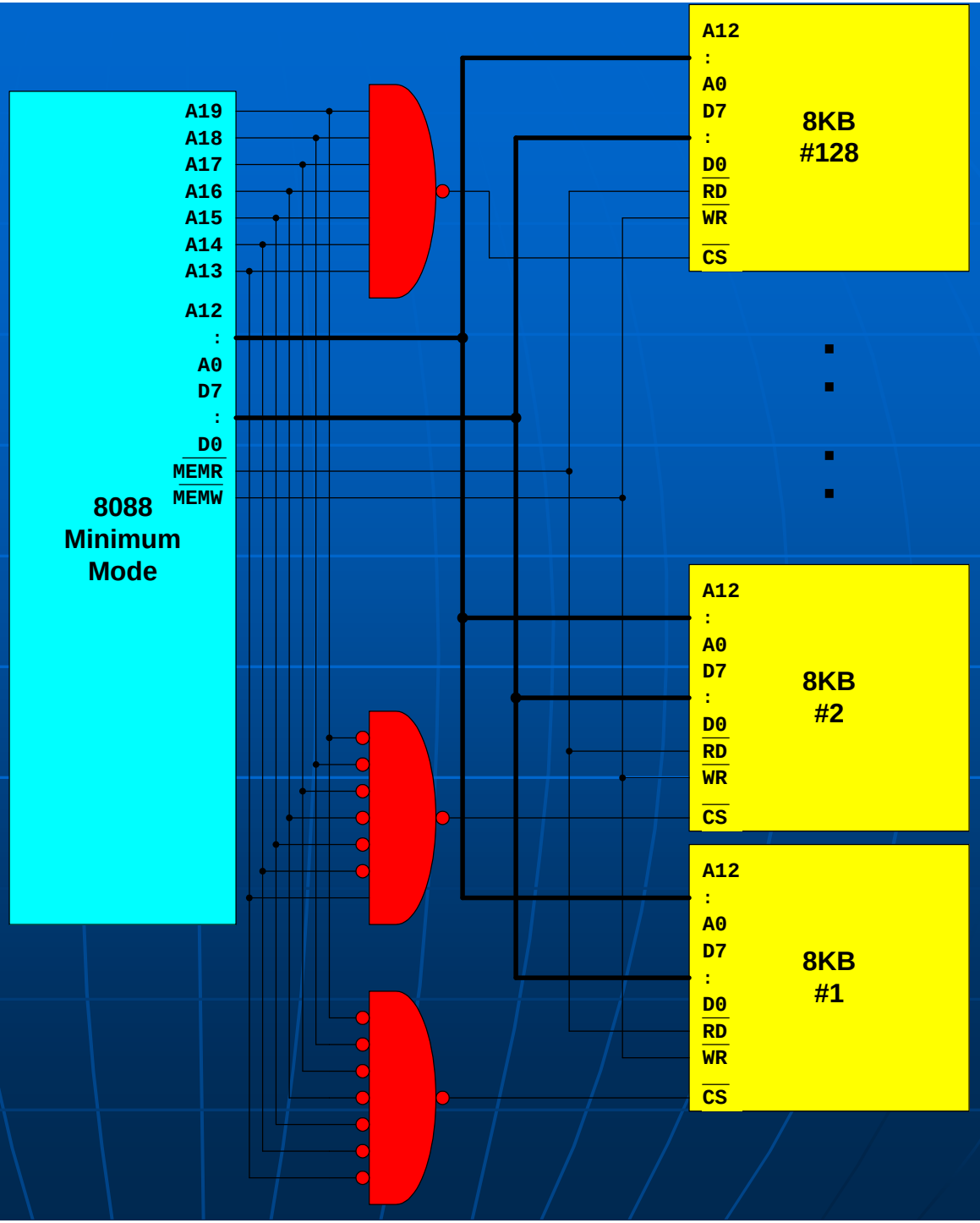
Interfacing several 8K Memory Chips to the 8088 µP



Interfacing 128 8K Memory Chips to the 8088 µP



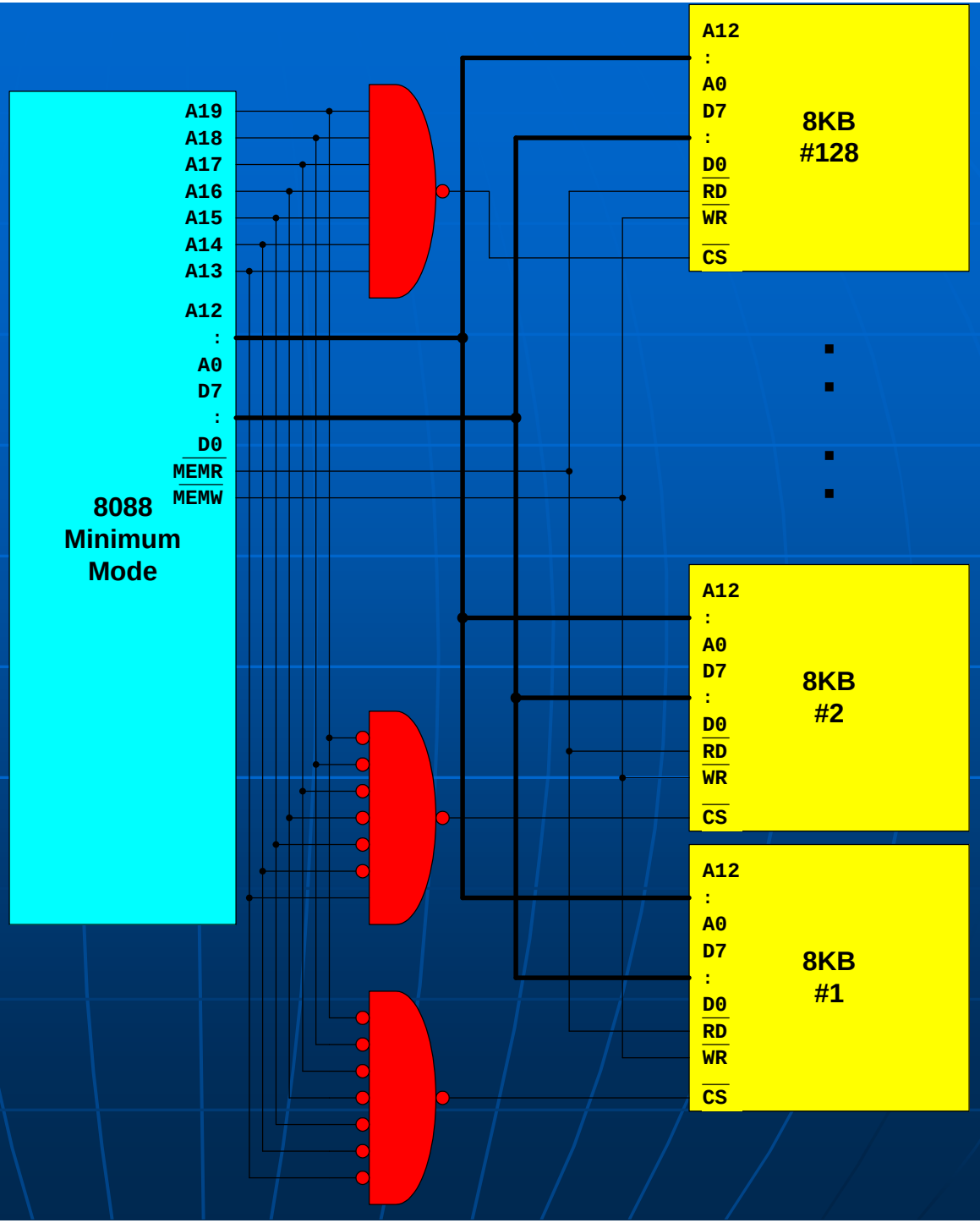
Interfacing 128 8K Memory Chips to the 8088 µP



Memory chip#__ is mapped to:

A19 to A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
-----	----	----	----	----	----
-----	----	----	----	----	----

Interfacing 128 8K Memory Chips to the 8088 µP



Bài giảng Kỹ thuật Vi xử lý

Ngành Điện tử-Viễn thông
Đại học Bách khoa Đà Nẵng
của Hồ Viết Việt, Khoa ĐTVT

Tài liệu tham khảo

[1] Kỹ thuật vi xử lý, Văn Thế Minh, NXB Giáo dục, 1997

[2] Kỹ thuật vi xử lý và Lập trình Assembly cho hệ vi xử lý, Đỗ Xuân Tiến, NXB Khoa học & kỹ thuật, 2001

Chương 5
Thiết kế các cổng I/O

5.1 I/O được phân vùng nhớ và I/O tách biệt

- I/O được phân vùng nhớ (Memory Mapped I/O)
- I/O tách biệt (Isolated I/O)

5.2 Các chip MSI dùng làm cổng I/O

- Cổng ra
- Cổng vào

5.3 Chip 8255

- Sơ đồ chân, Sơ đồ khối chức năng
- Các mode hoạt động
- Giải mã địa chỉ
- Lập trình cho 8255

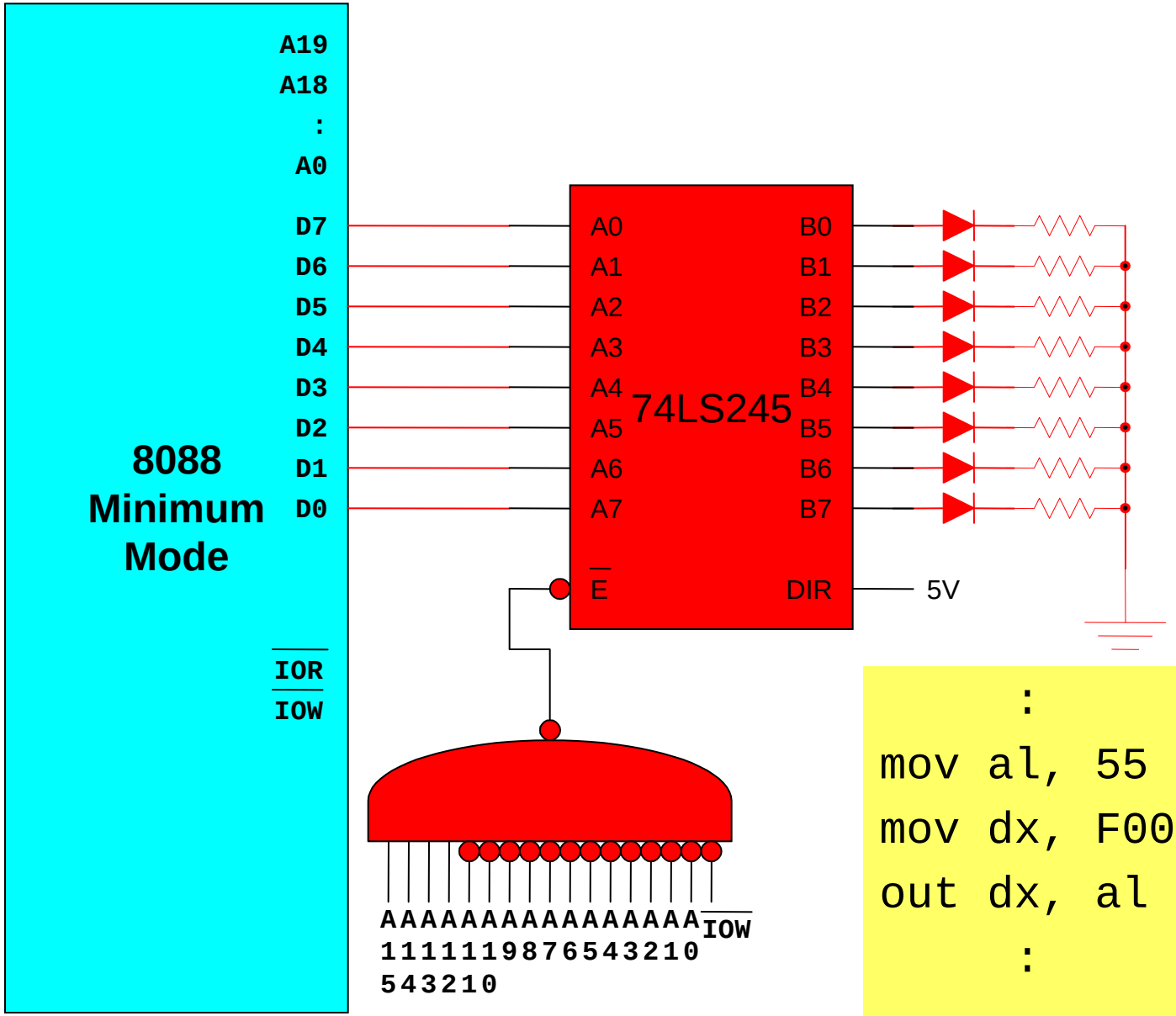
5.1 Cần phân biệt 2 kiểu thiết kế

- I/O được phân vùng nhớ (Memory mapped I/O):
  - 1 cổng được xem như một ô nhớ
  - 1 cổng có địa chỉ 20-bit
  - được truy cập khi IO/M = 0
  - không cần mạch giải mã địa chỉ riêng
- I/O tách biệt (isolated I/O)
  - 1 cổng được xem đúng là 1 cổng
  - 1 cổng có địa chỉ 16-bit, 12-bit, 8-bit
  - được truy cập khi IO/M = 1
  - cần mạch giải mã địa chỉ I/O riêng

5.2 Các chip MSI thường dùng làm cổng I/O

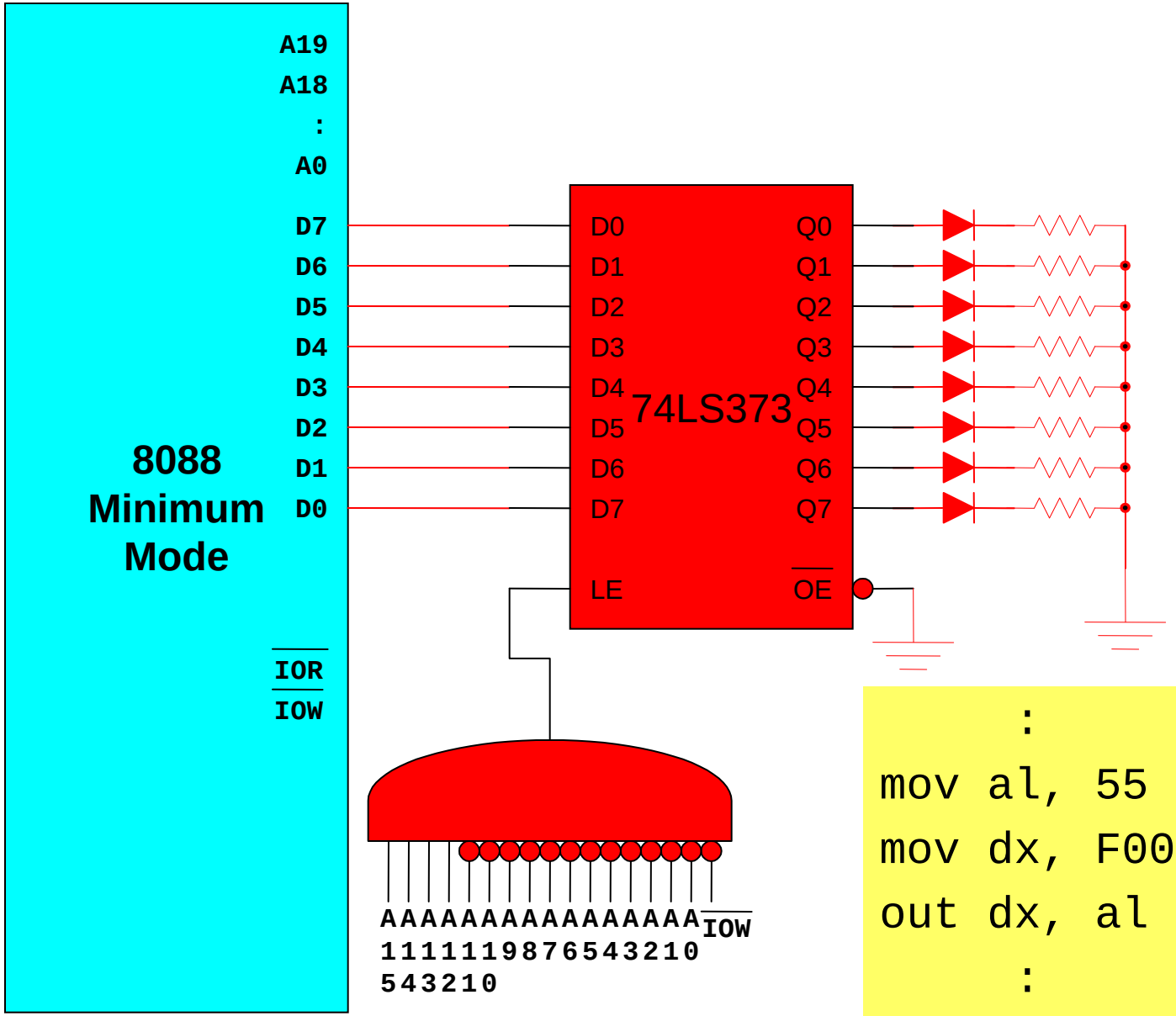
- 74LS373
- 74LS374
- 74LS244
- 74LS245
- Khi số lượng cổng ít và cố định
- Cách mắc mạch sẽ quyết định cho chip là cổng ra hay cổng vào và địa chỉ của nó

Sử dụng 74LS245 làm cổng ra



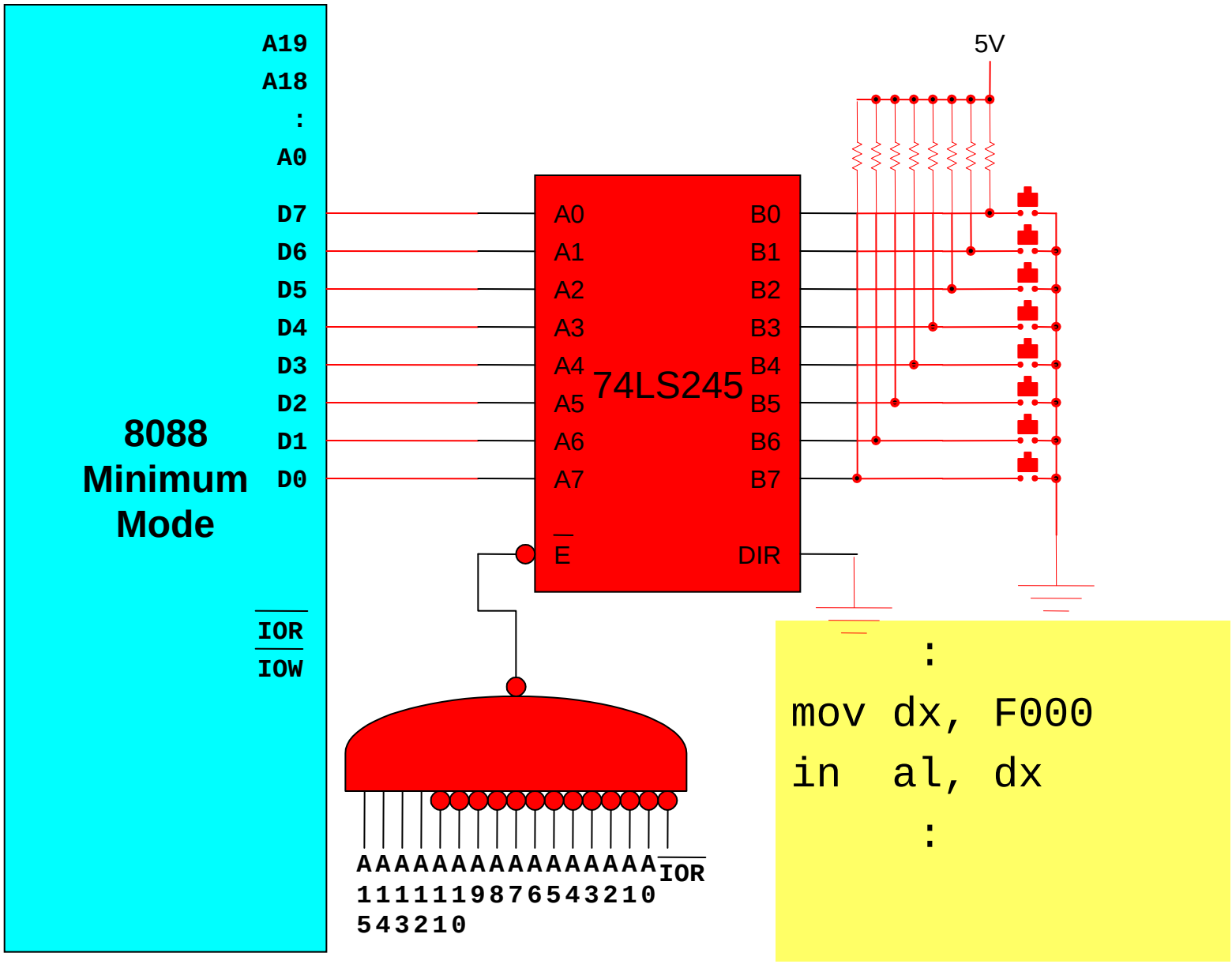

```
      :
mov al, 55
mov dx, F000
out dx, al
      :
```

Sử dụng 74LS373 làm cổng ra

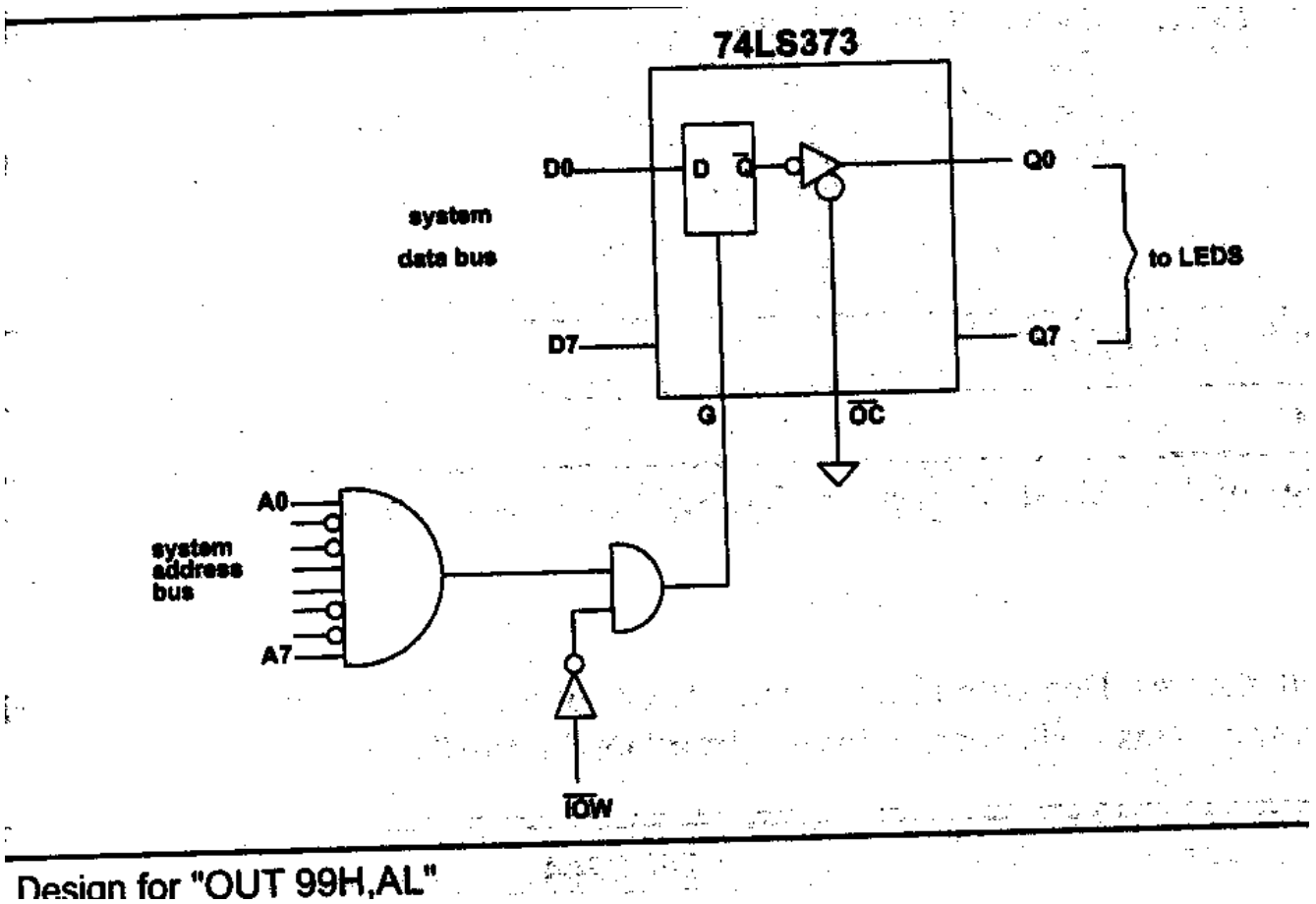



```
      :
mov al, 55
mov dx, F000
out dx, al
      :
```

Sử dụng 74LS245 làm cổng vào



Cổng ra



Design for "OUT 99H,AL"

Cổng vào

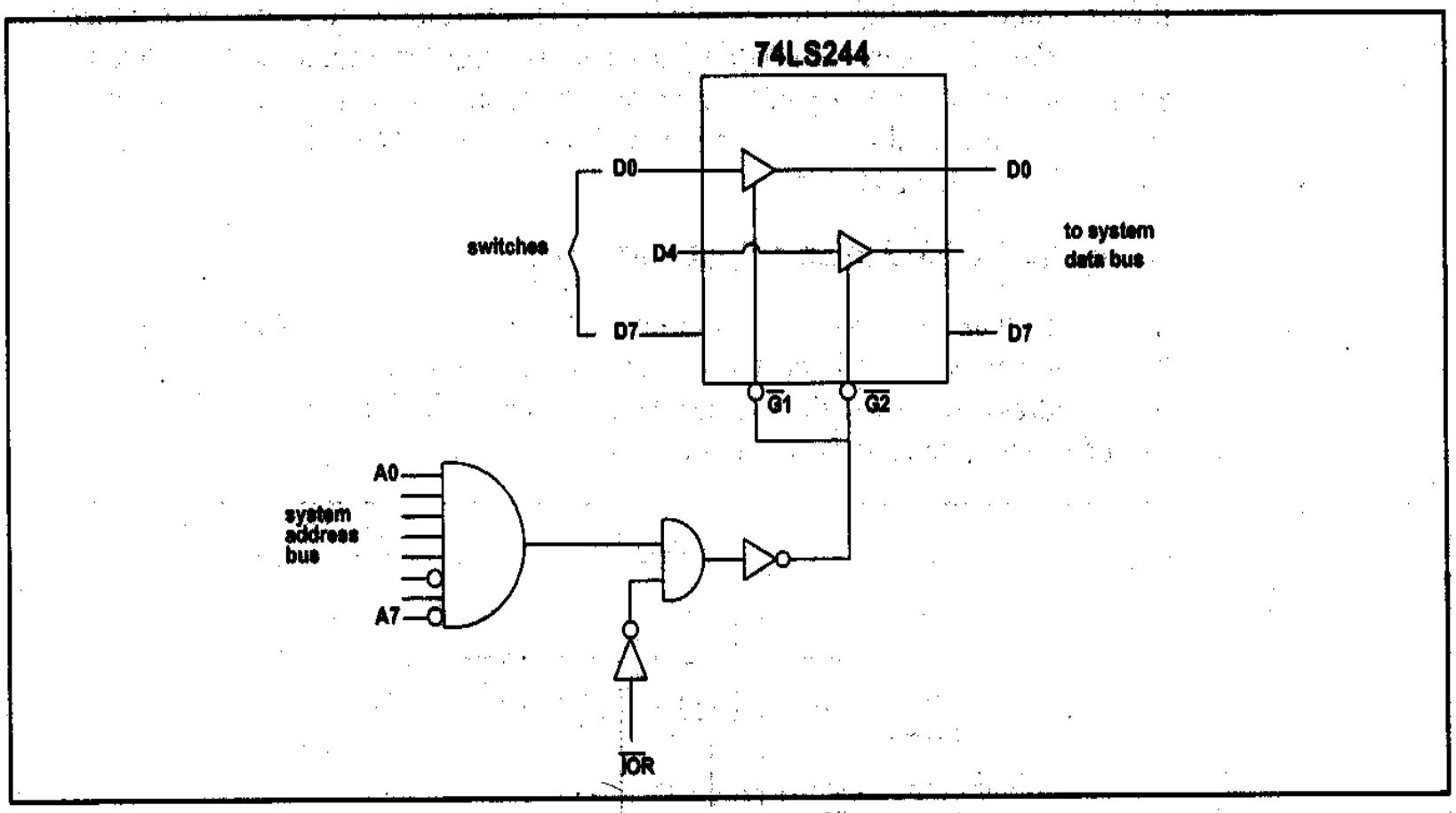


Figure 4-2. Design for "IN AL,5FH"

5.3 Chip LSI thường dùng làm cổng I/O

- PPI 8255
- Khi số lượng cổng I/O nhiều và không cố định
- Cách mắc mạch sẽ quyết định địa chỉ cho các cổng còn vai trò của cổng sẽ được quyết định bởi phần mềm

8255 PPI

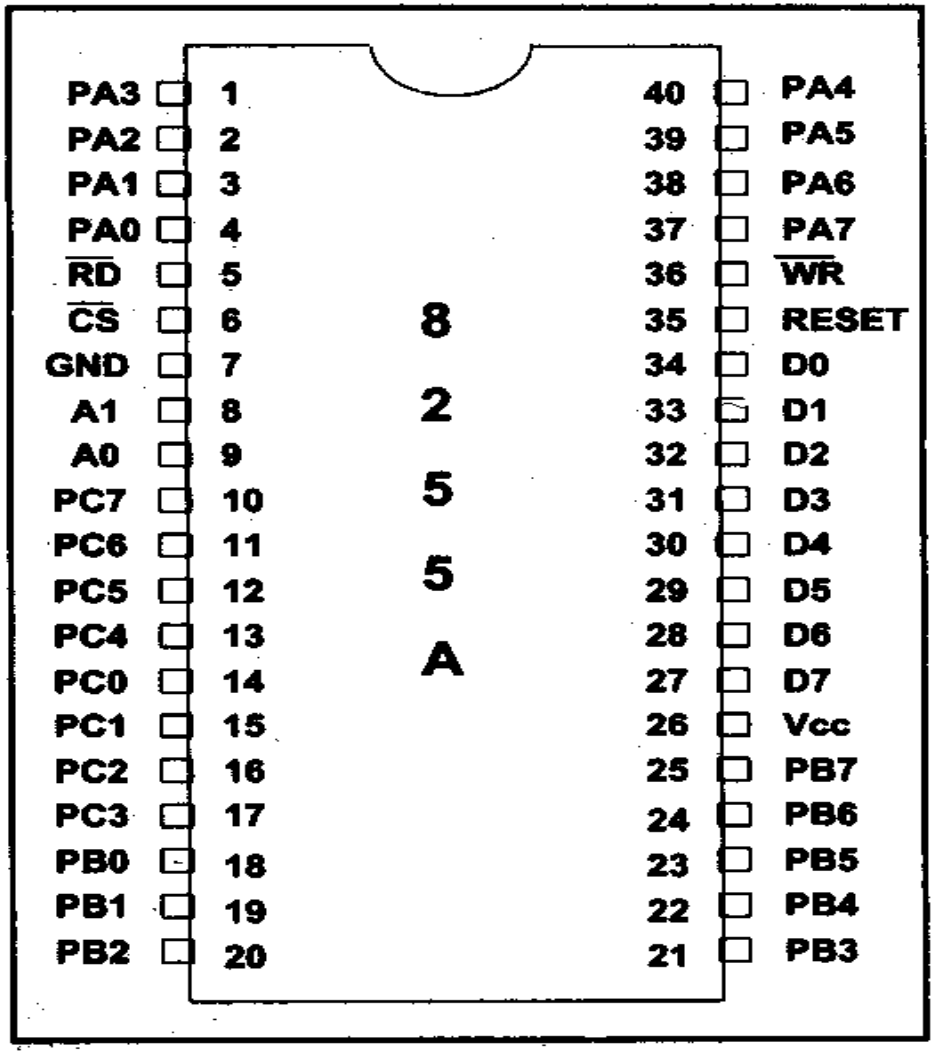


Figure 4-4. 8255 PPI Chip
(Reprinted by permission of Intel Corporation, Copyright Intel Corp. 1983)

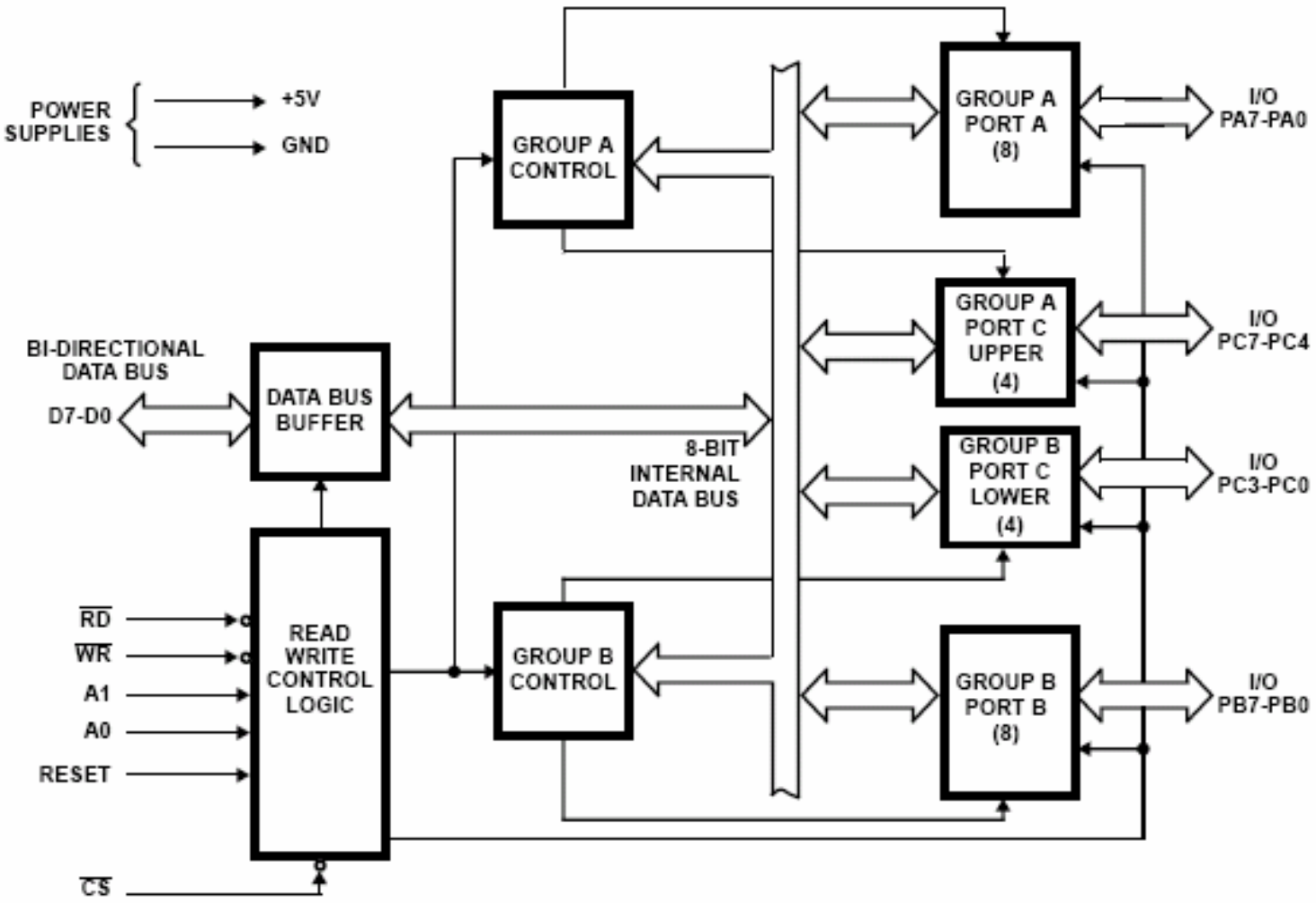
Table 4-1: 8255 Port Selection

CS*	A1	A0	Selects:
0	0	0	Port A
0	0	1	Port B
0	1	0	Port C
0	1	1	Control register
1	x	x	8255 is not selected

(Reprinted by permission of Intel Corporation, Copyright Intel Corp. 1983)

Sơ đồ khối chức năng của 8255

Functional Diagram



Các mode làm việc

- Mode 0
  - PA, PB, PCH (CU) và PCL (CL)
  - Có thể là Input hoặc Output
  - Việc Nhập hoặc Xuất dữ liệu là độc lập
- Mode 1
  - PA, PB
  - Có thể là Input hoặc Output
  - Việc Nhập hoặc Xuất dữ liệu là phụ thuộc vào một số bít của PC (các tín hiệu handshaking)
- Mode 2
  - PA
  - PA vừa là Input vừa là Output
  - Việc Nhập/Xuất dữ liệu với PA là phụ thuộc vào một số bít của PC (các tín hiệu handshaking)

Nhóm làm việc

- Nhóm A: PA và PCH
- Nhóm B: PB và PCL
- Định cấu hình làm việc cho 1 chip 8255: Gửi 1 Từ điều khiển định cấu hình đến thanh ghi điều khiển của chip đó
- Lập/xoá một bit của PC: Gửi 1 Từ điều khiển Lập/Xoá bit đến thanh ghi điều khiển của chip đó

Từ điều khiển định cấu hình làm việc cho một chip 8255

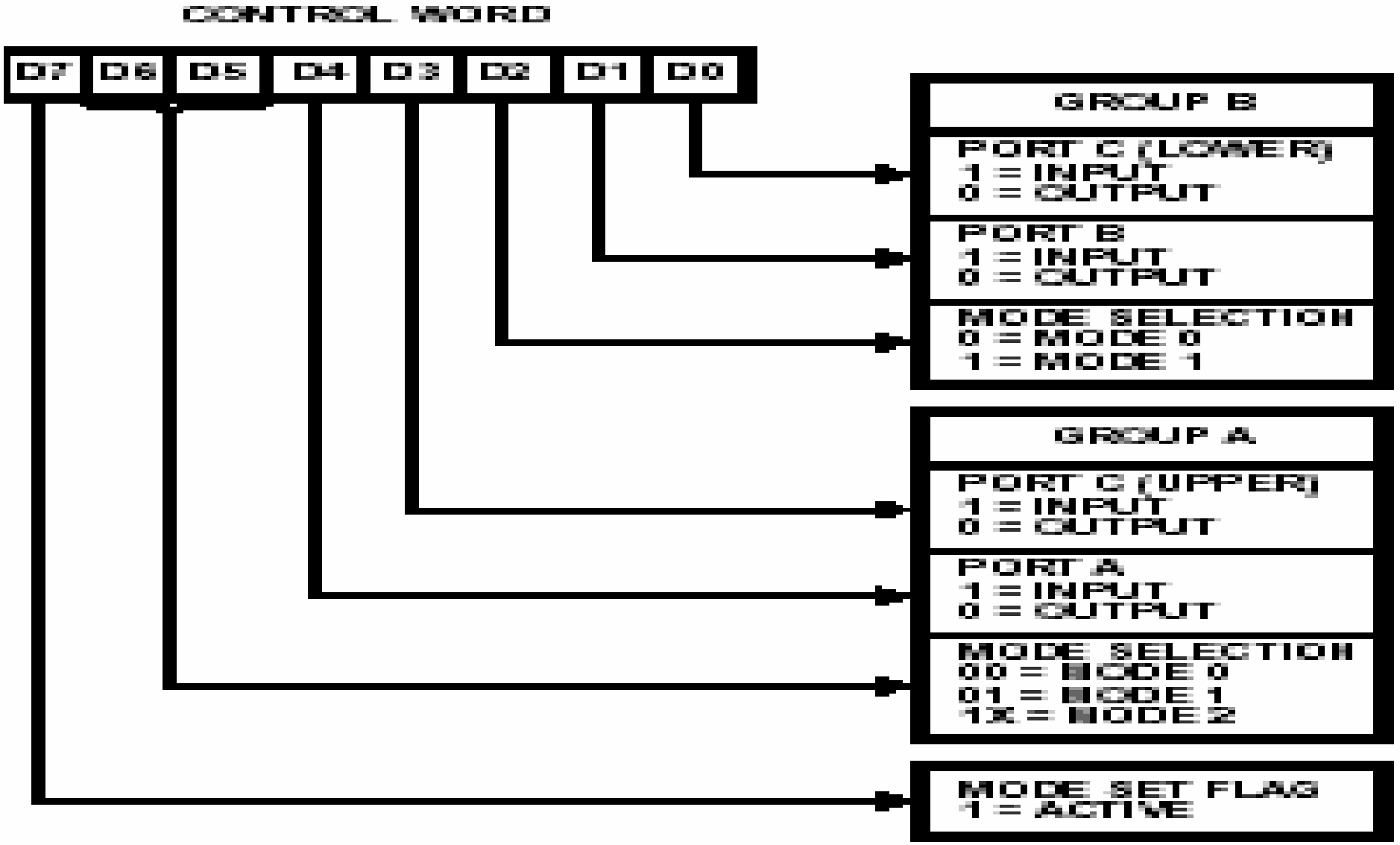


FIGURE 4. MODE DEFINITION FORMAT

Từ điều khiển lập/xoá bit cho một chip 8255

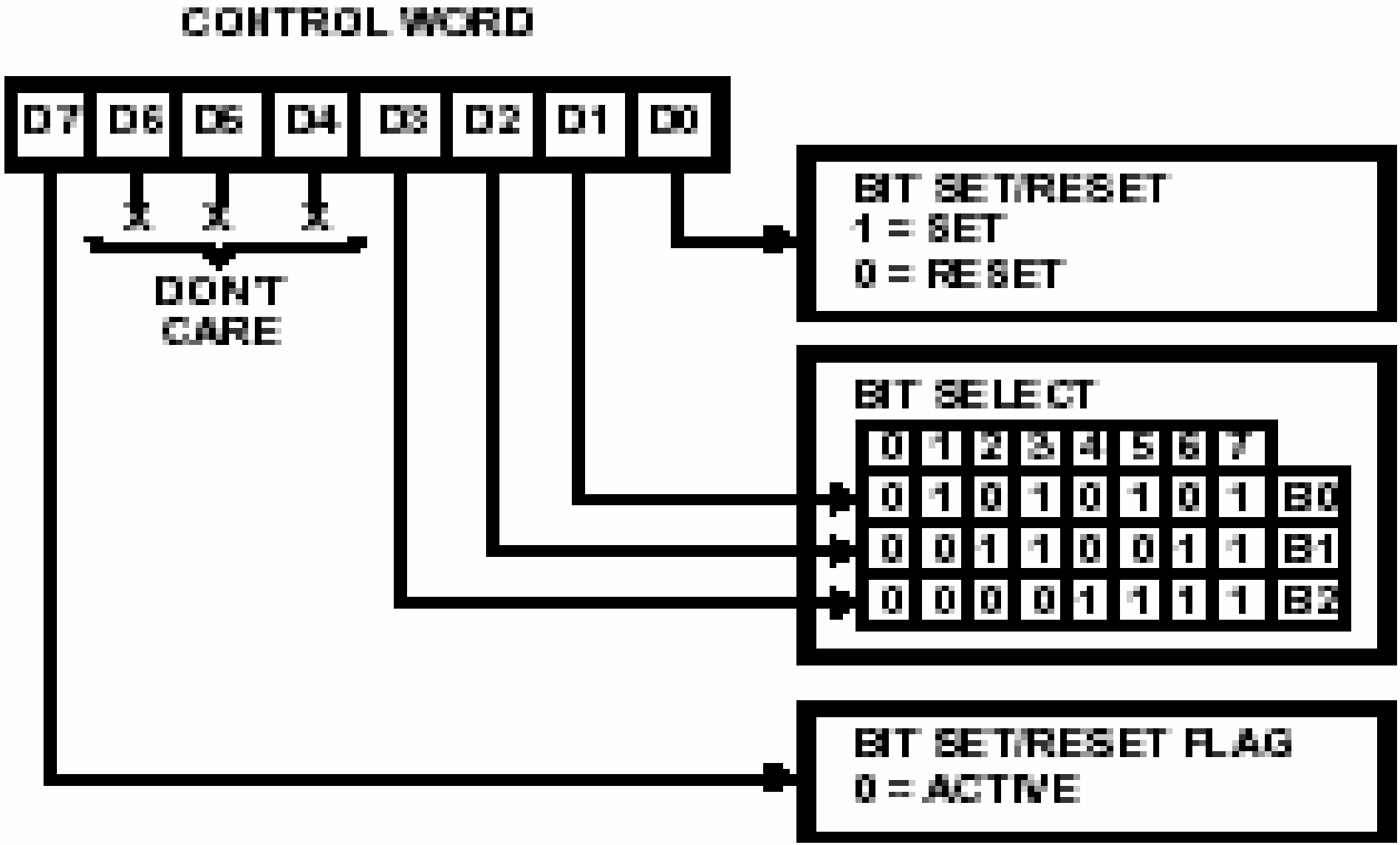
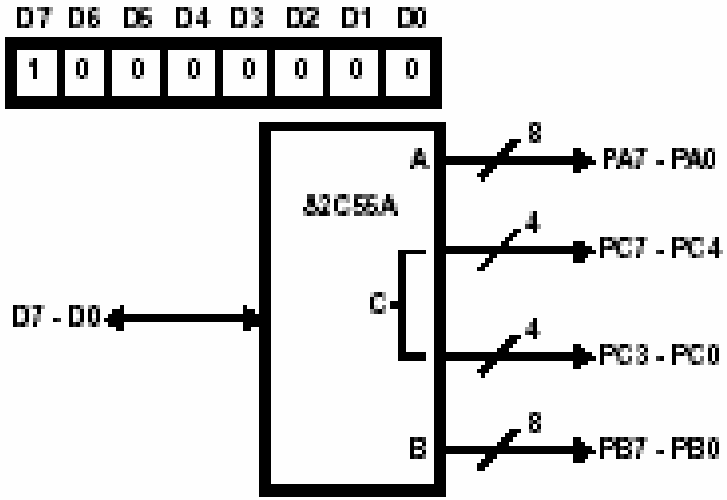


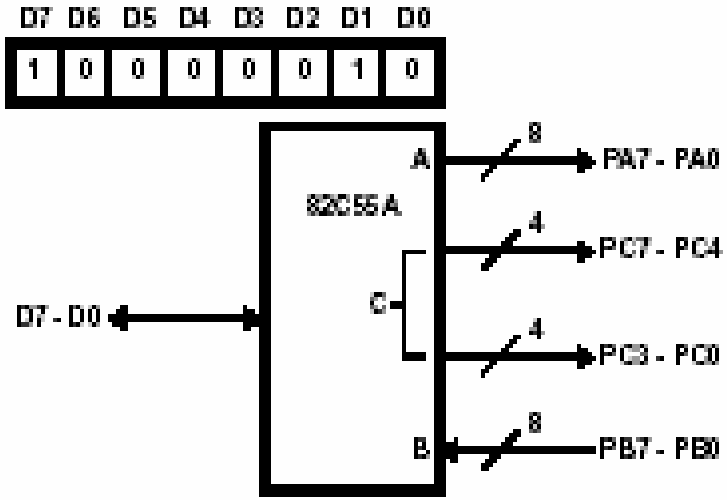
FIGURE 5. BIT SET/RESET FORMAT

Mode 0 Configurations

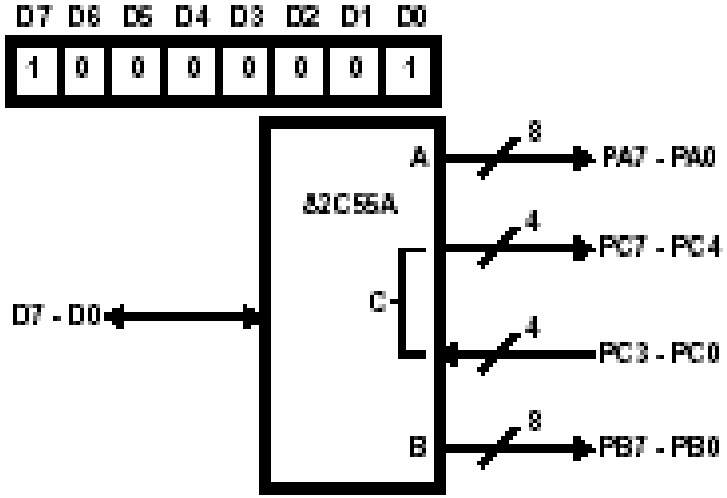
CONTROL WORD #0



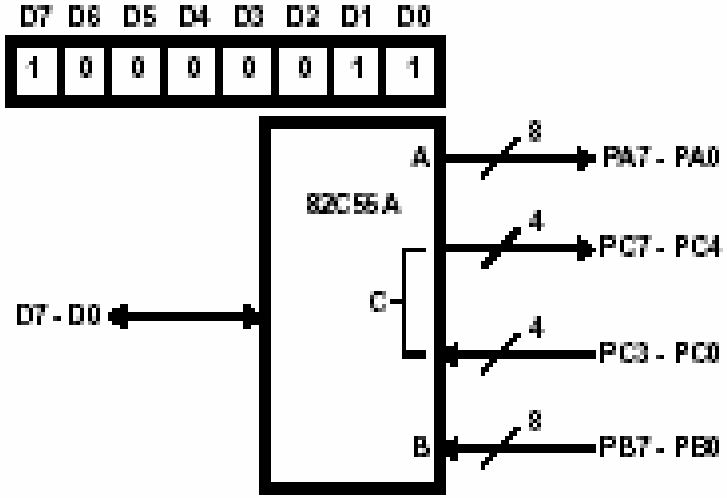
CONTROL WORD #2



CONTROL WORD #1



CONTROL WORD #3



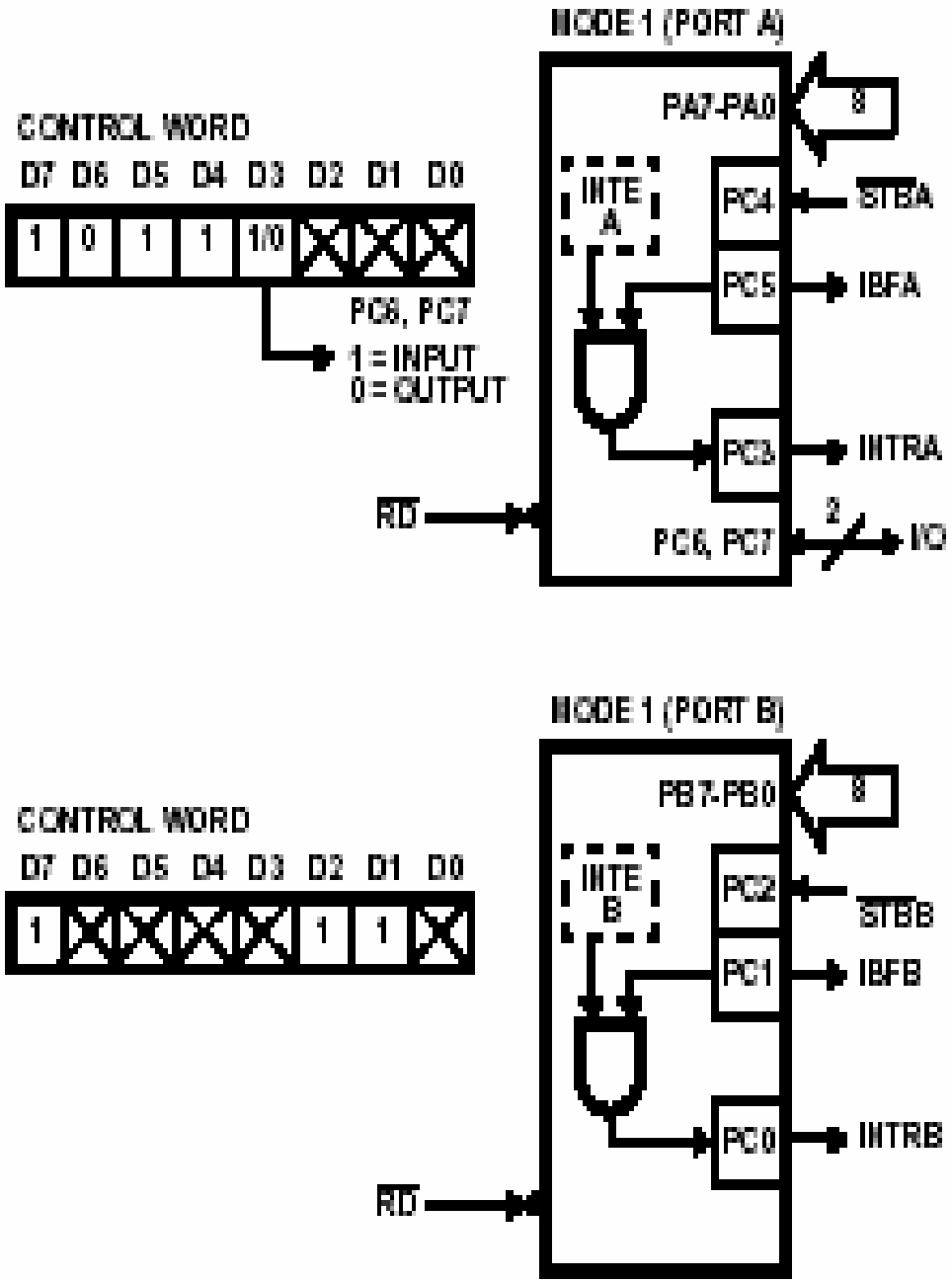


FIGURE 6. MODE 1 INPUT

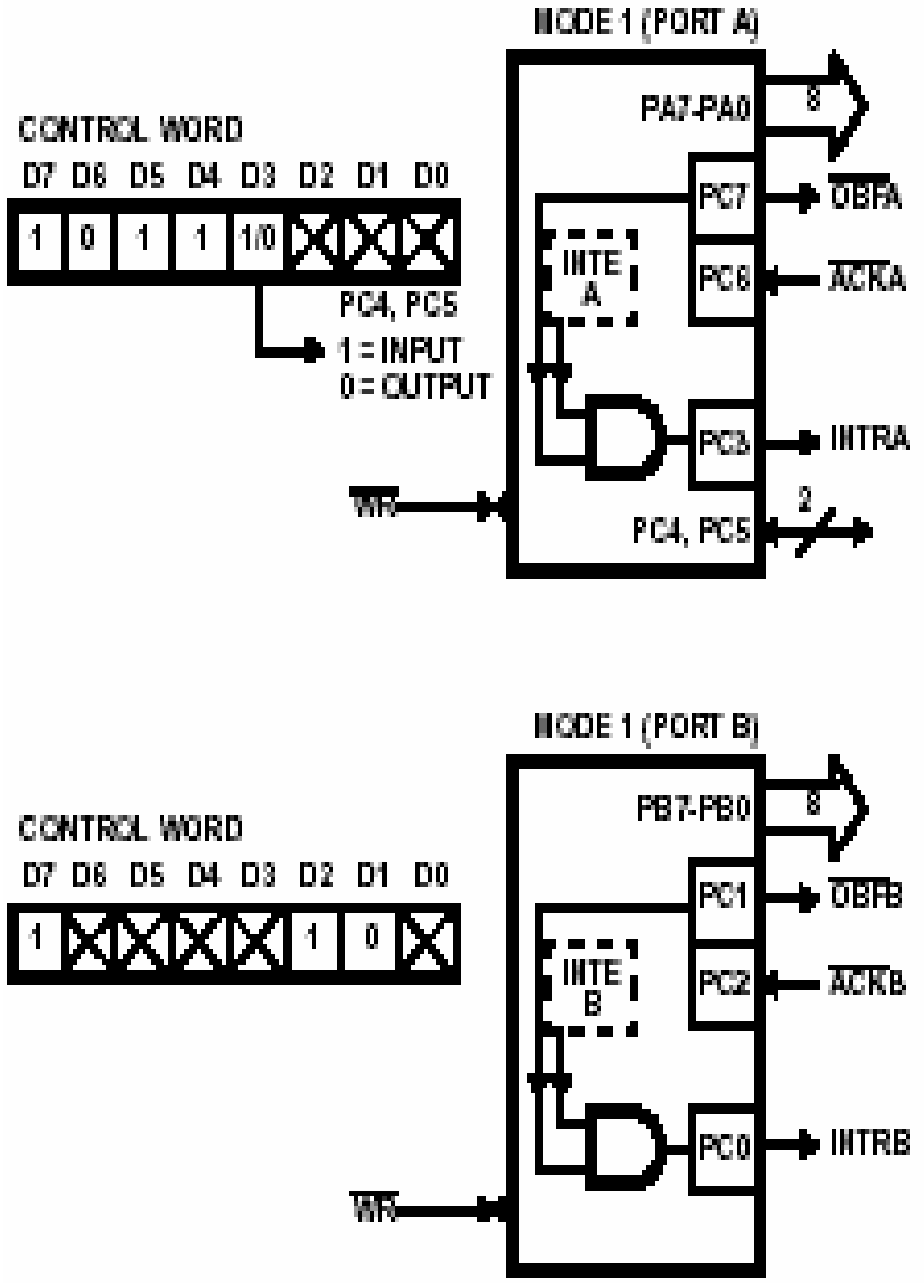
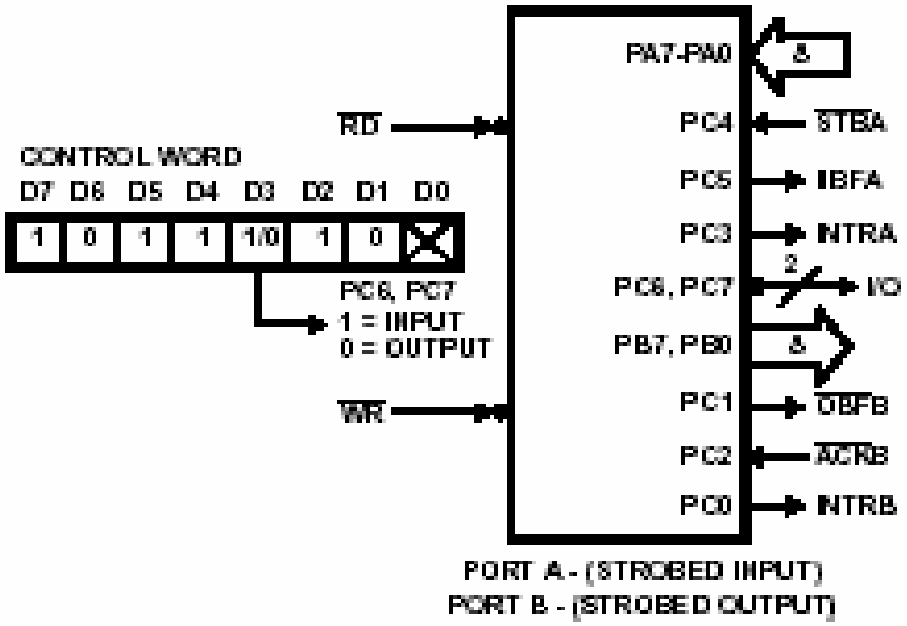
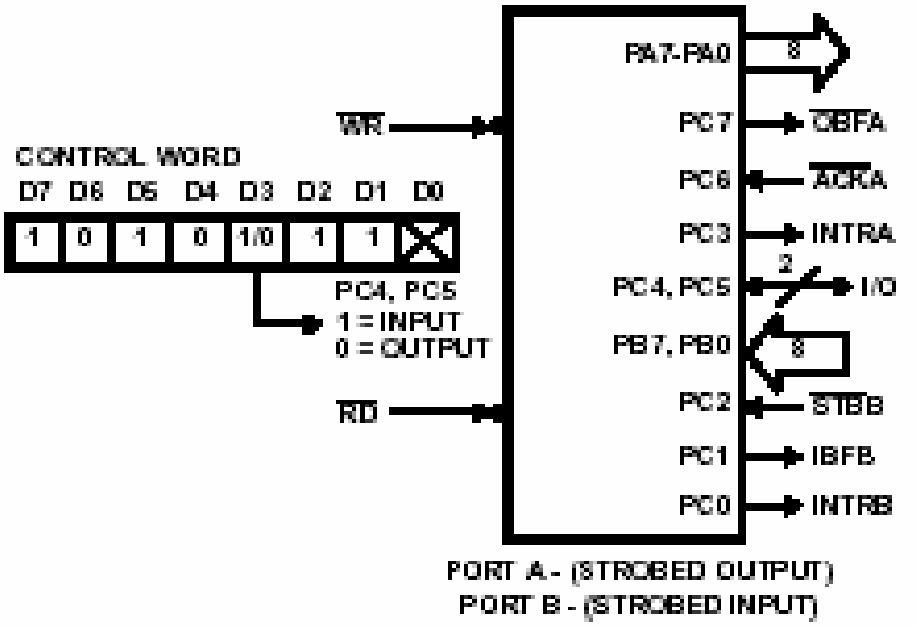


FIGURE 8. MODE 1 OUTPUT



PORT A - (STROBED INPUT)
PORT B - (STROBED OUTPUT)



PORT A - (STROBED OUTPUT)
PORT B - (STROBED INPUT)

Combinations of Mode 1: Port A and Port B can be individually defined as input or output in Mode 1 to support a wide variety of strobed I/O applications.

FIGURE 10. COMBINATIONS OF MODE 1

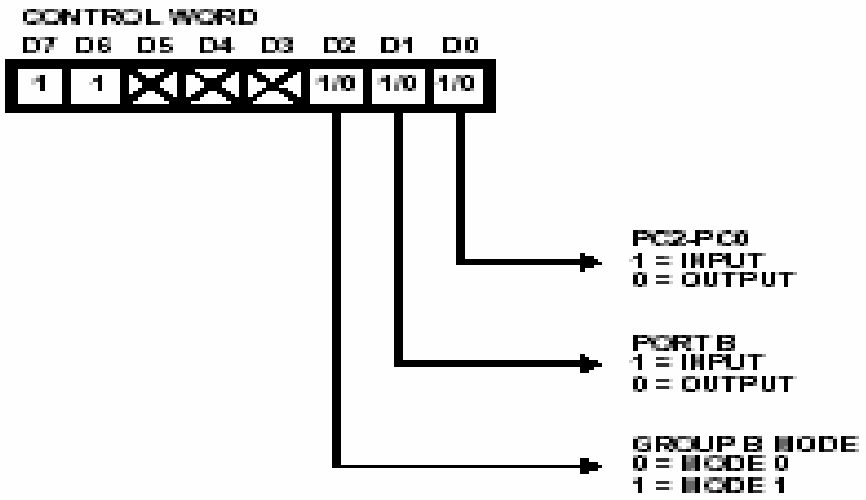


FIGURE 11. MODE CONTROL WORD

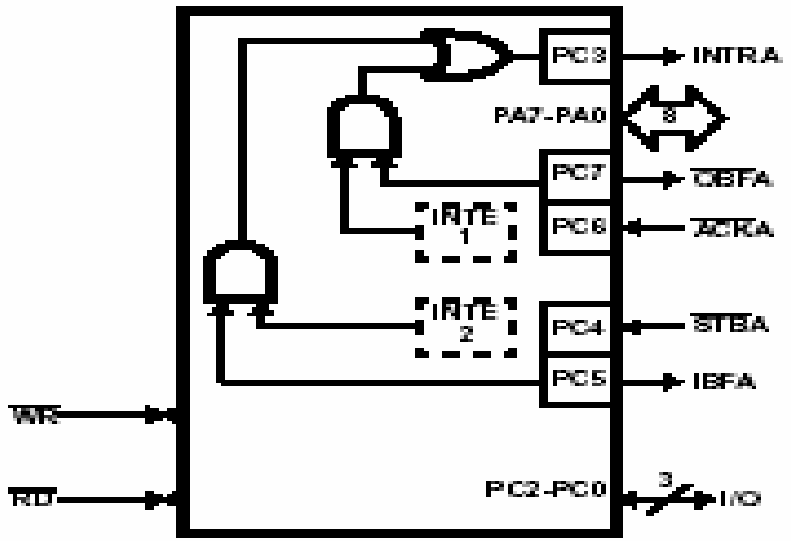


FIGURE 12. MODE 2

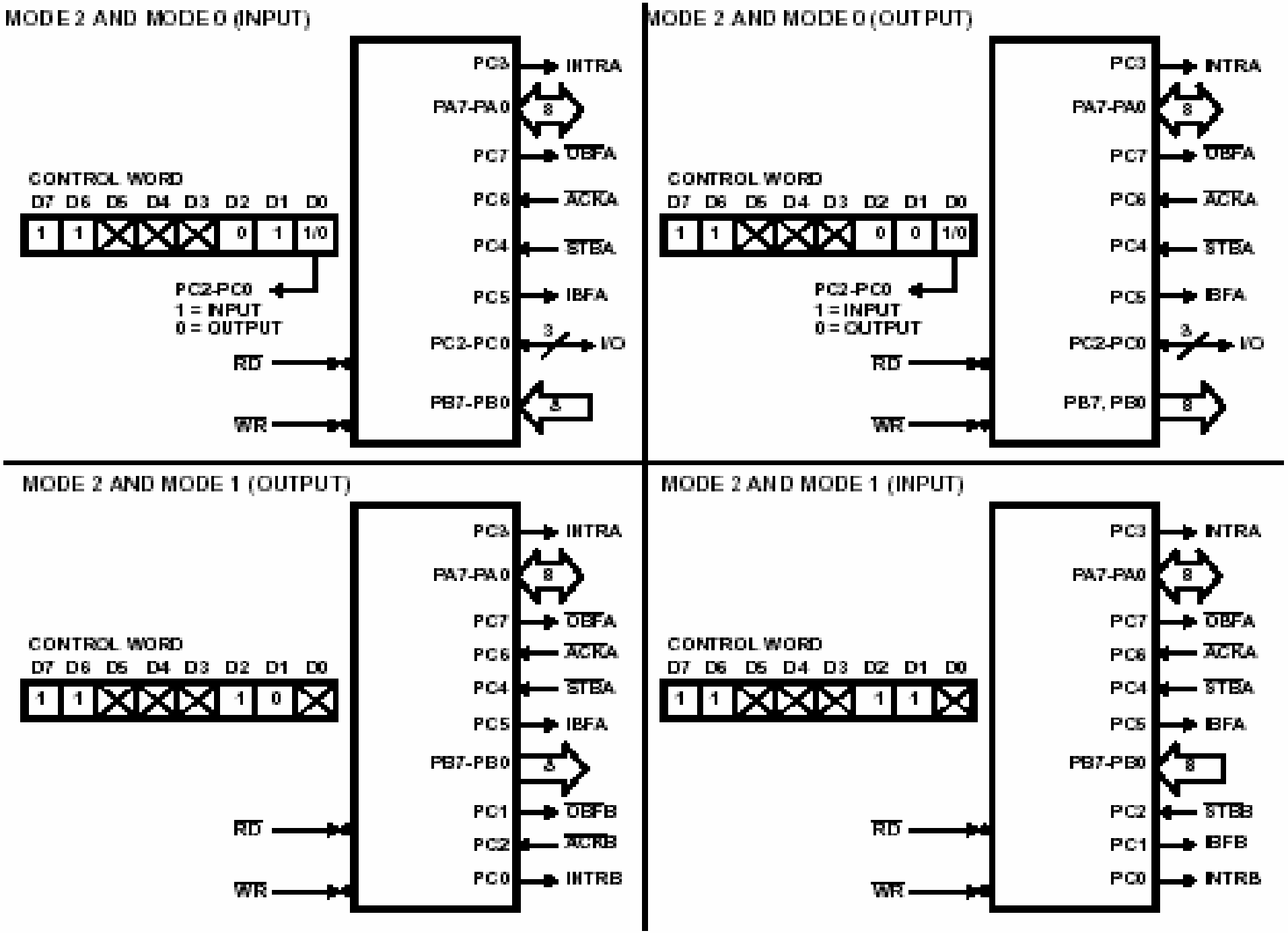


FIGURE 14. MODE 2 COMBINATIONS

MODE DEFINITION SUMMARY

	MODE 0		MODE 1		MODE 2	
	IN	OUT	IN	OUT	GROUP A ONLY	
PA0	In	Out	In	Out	← →	
PA1	In	Out	In	Out	← →	
PA2	In	Out	In	Out	← →	
PA3	In	Out	In	Out	← →	
PA4	In	Out	In	Out	← →	
PA5	In	Out	In	Out	← →	
PA6	In	Out	In	Out	← →	
PA7	In	Out	In	Out	← →	
PB0	In	Out	In	Out		Mode 0 or Mode 1 Only
PB1	In	Out	In	Out		
PB2	In	Out	In	Out		
PB3	In	Out	In	Out		
PB4	In	Out	In	Out		
PB5	In	Out	In	Out		
PB6	In	Out	In	Out		
PB7	In	Out	In	Out		
PC0	In	Out	INTRB	INTRB	I/O	
PC1	In	Out	IBFB	$\overline{OBF}$B	I/O	
PC2	In	Out	$\overline{STB}$B	$\overline{ACK}$B	I/O	
PC3	In	Out	INTRA	INTRA	INTRA	
PC4	In	Out	$\overline{STB}$A	I/O	$\overline{STB}$A	
PC5	In	Out	IBFA	I/O	IBFA	
PC6	In	Out	I/O	$\overline{ACK}$A	$\overline{ACK}$A	
PC7	In	Out	I/O	$\overline{OBF}$A	$\overline{OBF}$A	

82C55A BASIC OPERATION

A1	A0	RD	WR	CS	INPUT OPERATION (READ)
0	0	0	1	0	Port A → Data Bus
0	1	0	1	0	Port B → Data Bus
1	0	0	1	0	Port C → Data Bus
1	1	0	1	0	Control Word → Data Bus
					OUTPUT OPERATION (WRITE)
0	0	1	0	0	Data Bus → Port A
0	1	1	0	0	Data Bus → Port B
1	0	1	0	0	Data Bus → Port C
1	1	1	0	0	Data Bus → Control
					DISABLE FUNCTION
X	X	X	X	1	Data Bus → Three-State
X	X	1	1	0	Data Bus → Three-State

The 8255 Programmable Peripheral Interface

• Intel has developed several peripheral controller chips designed to support the 80x86 processor family. The intent is to provide a complete I/O interface in one chip.

• 8255 PPI provides **three 8 bit input ports** in one 40 pin package making it more economical than 74LS373 and 74LS244

• The chip interfaces directly to the data bus of the processor, allowing its functions to be programmed; that is in one application a port may appear as an output, but in another, by reprogramming it as an input. This is in contrast with the 74LS373 and 74LS244 which are hard wired and fixed.

8255 Pins

• PA0 - PA7: input, output, or bidirectional port

• PB0 - PB7: input or output

• PC0 - PC7: This 8 bit port can be all input or output. It can also be split into two parts, CU (PC4 - PC7) and CL (PC0 - PC3). Each can be used for input and output.

• RD or WR

– IOR and IOW of the system are connected to these two pins

• RESET

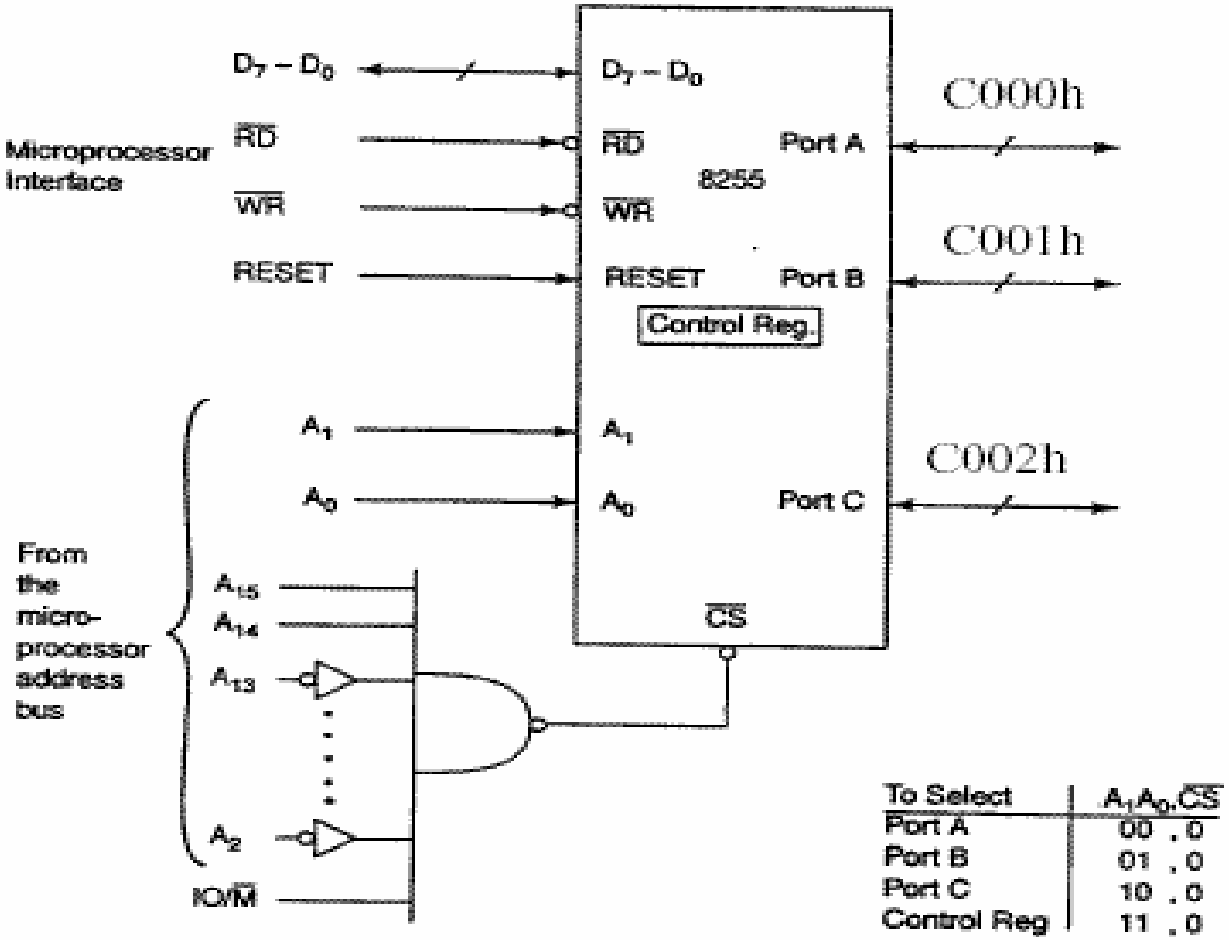
• A0, A1, and CS

– CS selects the entire chip whereas A0 and A1 select the specific port (A, B, or C) or Control Register.

CSBAR	A1	A0	SELECTS:
0	0	0	Port A
0	0	1	Port B
0	1	0	Port C
0	1	1	Control Register
1	x	x	8255 not selected

Giải mã địa chỉ cho 8255

Addressing an 8255



To Select	$A_1A_0.\overline{CS}$
Port A	00 . 0
Port B	01 . 0
Port C	10 . 0
Control Reg	11 . 0

Mode 0 - Simple input/output

• Simple I/O mode: any of the ports A, B, CL, and CU can be programmed as input or output.

• Example: Configure port A as input, B as output, and all the bits of port C as output assuming a base address of 50h

• Control word should be 1001 0000b = 90h

```
MOV AL, 90h
OUT 53h,AL
IN AL, 50h
OUT 51h, AL
OUT 52h, AL
```

Mode 1: I/O with Handshaking Capability

• Handshaking refers to the process of communicating back and forth between two intelligent devices

• Example. Process of communicating with a printer

– a byte of data is presented to the data bus of the printer

– the printer is informed of the presence of a byte of data to be printed by activating its strobe signal

– whenever the printer receives the data it informs the sender by activating an output signal called ACK

– the ACK signal initiates the process of providing another byte of data to the printer

• 8255 in mode 1 is equipped with resources to handle handshaking signals

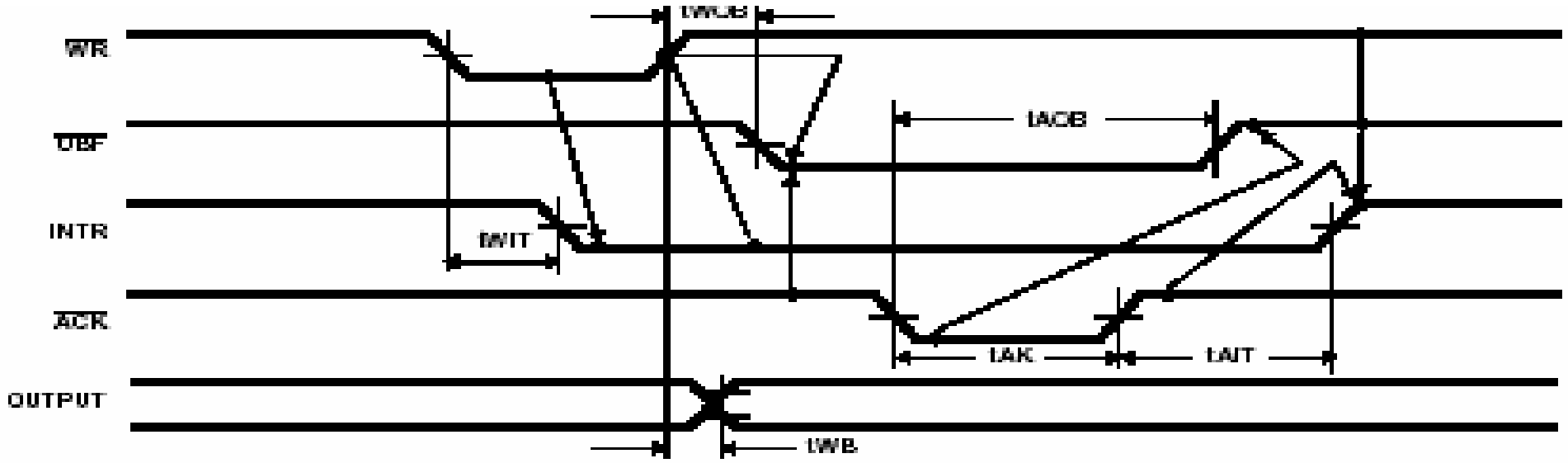


FIGURE 9. MODE 1 (STROBED OUTPUT)

Mode 1 Strobed Output Signals

• OBFa (output buffer full for port A)

– indicates that the CPU has written a byte of data into port A

– must be connected to the STROBE of the receiving equipment

• ACKa (acknowledge for port A)

– through ACK, 8255 knows that data at port A has been picked up by the receiving device

– 8255 then makes OBFa high to indicate that the data is old now. OBFa will not go low until the CPU writes a new byte of data to port A.

• INTRa (interrupt request for port A)

– it is the rising edge of ACK that activates INTRa by making it high. INTRa is used to get the attention of the microprocessor.

– it is important that INTRa is high only if INTEa, OBFa, ACKa are all high

– it is reset to zero when the CPU writes a byte to port A

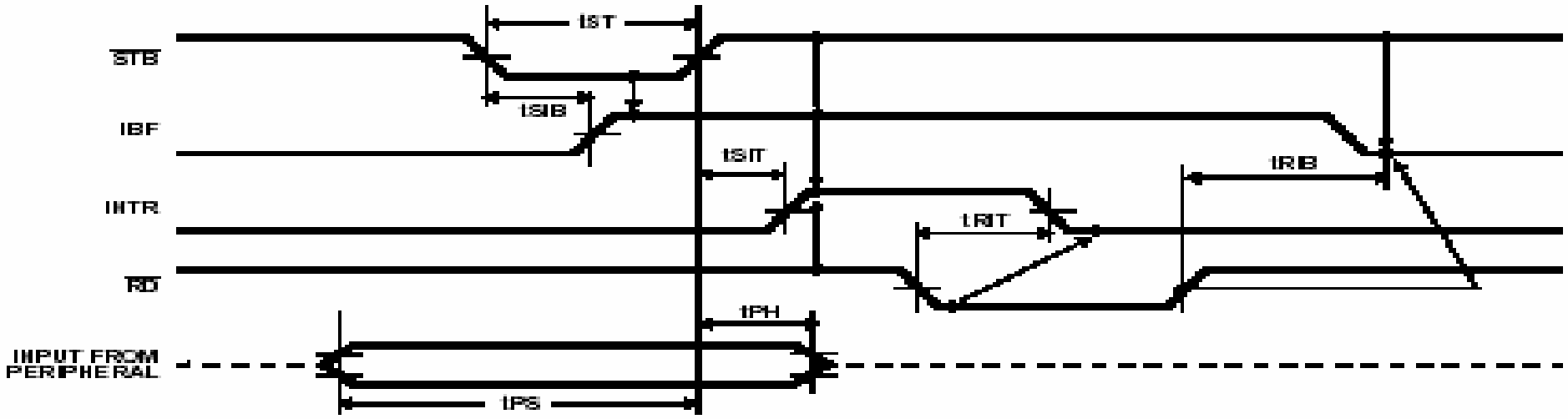


FIGURE 7. MODE 1 (STROBED INPUT)

Mode 1 Input Ports with Handshaking Signals

• STB

– When an external peripheral device provides a byte of data to an input port, it informs the 8255 through the STB pin. STB is of limited duration.

• IBF (Input Buffer Full) – In response to STB, the 8255 latches into its internal register the data present at PA0-PA7 or PB0-PB7.

– Through IBF it indicates that it has latched the data but it has not been read by the CPU yet.

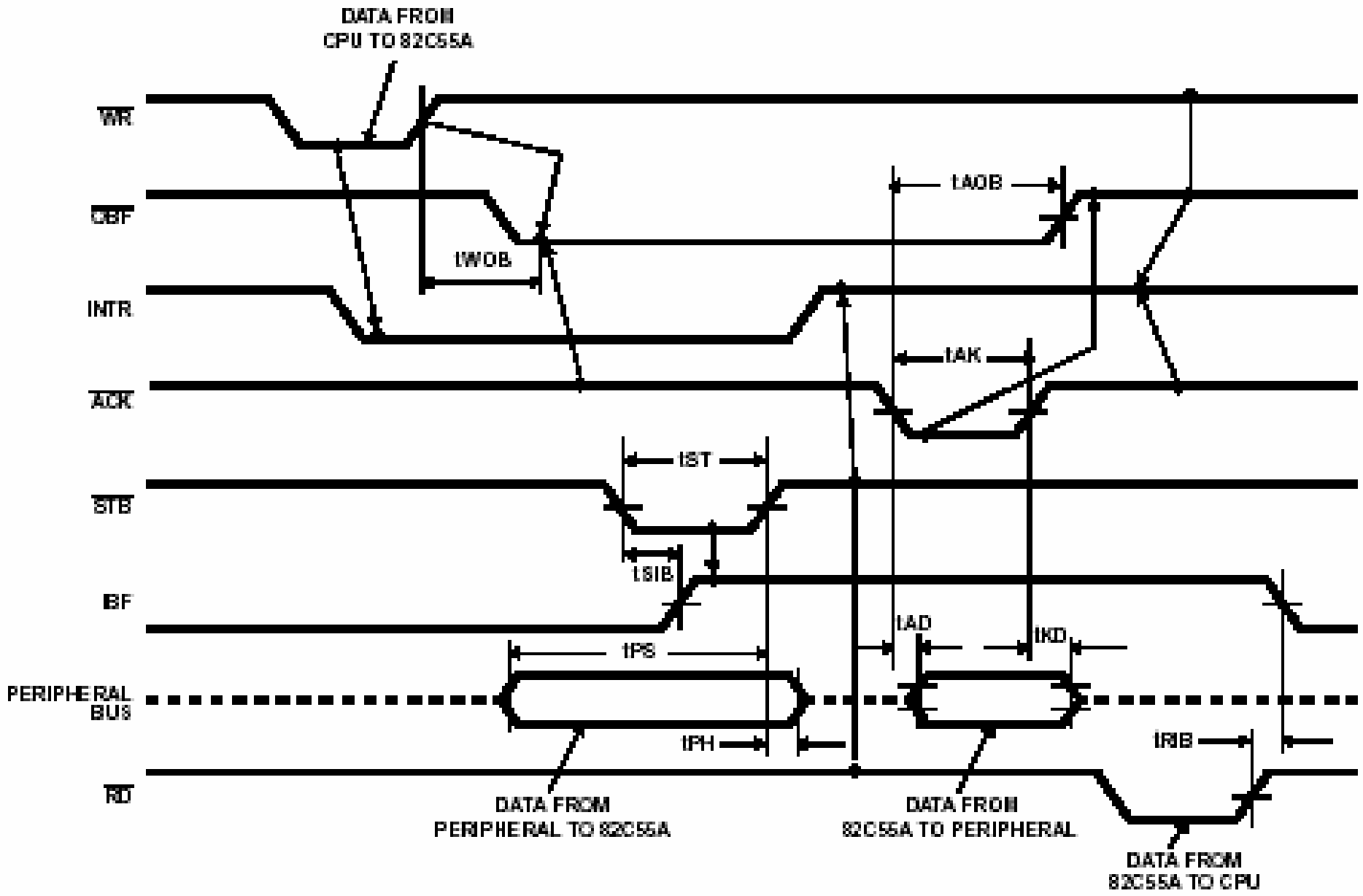
– To get the attention of the CPU, it IBF activates INTR

• INTR

– Falling edge of RD makes INTR low

– The RD signal from the CPU is of limited duration and when it goes high the 8255 in turn makes IBF inactive by setting it low.

– IBF in this way lets the peripheral know that the byte of data was latched by the 8255 and read into the CPU as well.



NOTE: Any sequence where $\overline{WR}$ occurs before $\overline{ACK}$ and $\overline{STB}$ occurs before RD is permissible. (INTR = IBF • MASK • $\overline{STB}$ • $\overline{RD}$ + $\overline{OBF}$ • MASK • $\overline{ACK}$ • $\overline{WR}$)

FIGURE 13. MODE 2 (BI-DIRECTIONAL)

Lập trình cho 8255

Example 4-5

The 8255 shown in Figure 4-6 is configured as follows: port A as input, B as output, and all the bits of port C as output.
(a) Find the port addresses assigned to A, B, C, and the control register.
(b) Find the control byte (word) for this configuration.
(c) Program the ports to input data from port A and send it to both ports B and C.

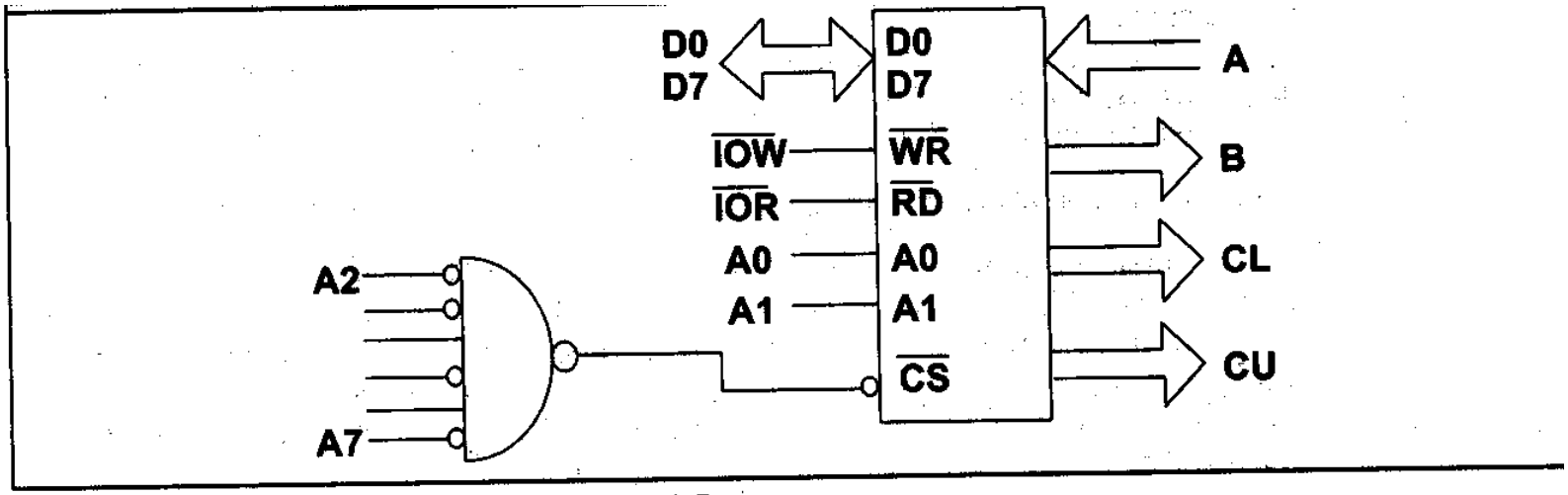


Figure 4-6. 8255 Configuration for Example 4-5

Lời giải

Solution:

(a) The port addresses are as follows:

CS*	A1	A0	Address	Port
0101 00	0	0	50H	Port A
0101 00	0	1	51H	Port B
0101 00	1	0	52H	Port C
0101 00	1	1	53H	Control register

(b) The control word is 90H, or 1001 0000.

(c) One version of the program is as follows:

```
MOV   AL,90H      ;control byte PA=in, PB=out, PC=out
OUT   53H,AL      ;send it to control register
IN    AL,50H      ;get the data from PA
OUT   51H,AL      ;send it to both PB
OUT   52H,AL      ;  and PC
```

Using the EQU directive one can rewrite the above program as follows:

```
PORTA     EQU   50H
PORTB     EQU   51H
PORTC     EQU   52H
CNTLREG   EQU   53H
          ...
          MOV   AL,90H        ;control byte PA=in, PB=out, PC=out
          OUT   CNTLREG,AL    ;send it to control register
          IN    AL,PORTA      ;get the data from PA
          OUT   PORTB,AL      ;send it to both PB
          OUT   PORTC,AL      ;  and PC
```

Lập trình cho 8255

Example 4-6

(a) Find the port address for Figure 4-7.
(b) Find the control word if PA =out, PB=in, PC0 - PC3 =in, and PC4 - PC7=out.
(c) Program the 8255 to get data from port B and send it to port A. In addition, data from PCL is send out to the PCU.

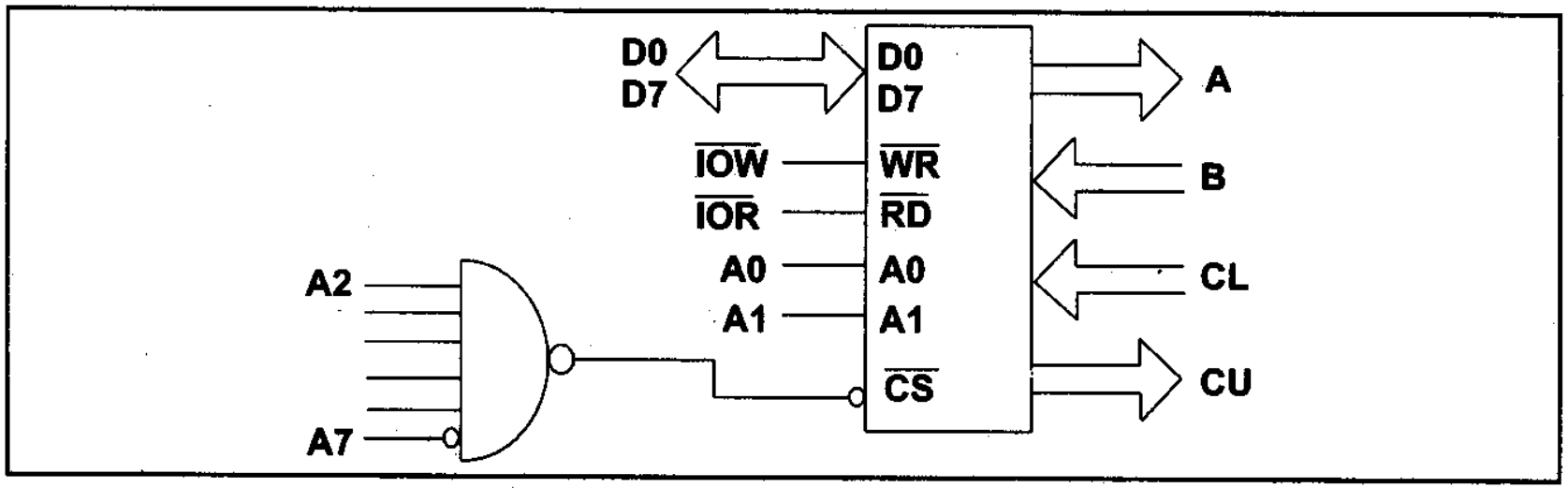


Figure 4-7. Configuration for Example 4-6

Lời giải

(a) The port addresses are as follows:

CS*	A1	A0	Address	Port
0111 11	0	0	7CH	Port A
0111 11	0	1	7DH	Port B
0111 11	1	0	7EH	Port C
0111 11	1	1	7FH	Control register

(b) The control word is 83H, or 1000 0011.

(c) The code is as follows.

```
MOV   AL,83H      ;control byte PA=out, PB=in, PCL=in, PCU=out
OUT   7FH,AL      ;send it to control register
IN    AL,7DH      ;get the data from PB
OUT   7CH,AL      ;send it to PA
IN    AL,7EH      ;get the bits from PCL
AND   AL,0FH      ;mask the upper bits
ROL   AL,1
ROL   AL,1        ;shift the bits
ROL   AL,1        ;to upper position
ROL   AL,1
OUT   7EH,AL      ;send it to PCU
```

Alternately, the four instructions above of "ROL AL,1" could be replaced with the following two instructions:

```
MOV   CL,4        ;count = 4
ROL   AL,CL       ;rotate 4 times
```

Tạo chuỗi xung bằng phần mềm

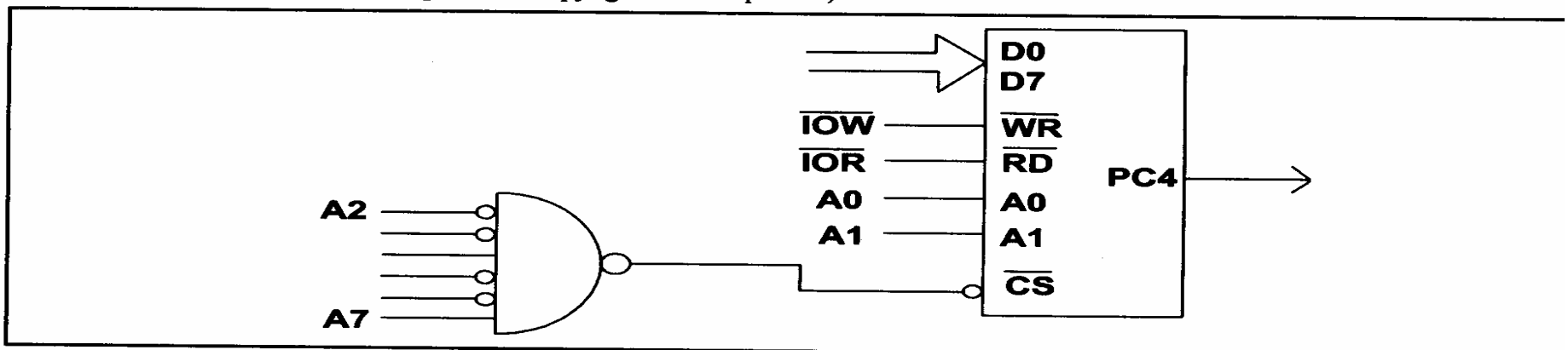


Figure 4-9. Configuration for Example 4-7

Example 4-7

Program PC4 of the 8255 in Figure 4-9 to generate a pulse of 50 ms with 50% duty cycle.

Solution:
To program the 8255 in BSR mode, bit D7 of the control word must be low. For PC4 to be high, we need a control word of "0xxx1001". Likewise, for low we would need "0xxx1000" as the control word. The x's are for "don't care" and generally are set to zero.

```
MOV   AL,00001001B      ;load the control byte (PC4=1)
OUT   93H,AL            ;set PC4 to high, sent to control reg
CALL  DELAY             ;time for the high part of pulse
MOV   AL,00001000B      ;load the control byte (PC4=0)
OUT   93,AL             ;set PC4 to low, sent to control reg
CALL  DELAY             ;time for the low part of pulse
```

In the above program, in the instruction "MOV AL,00001001B" the B stands for binary. There are various methods of writing a DELAY subroutine. Some are shown in Chapter 5.

Bài giảng Kỹ thuật Vi xử lý

Ngành Điện tử-Viễn thông
Đại học Bách khoa Đà Nẵng
của Hồ Viết Việt, Khoa ĐTVT

Tài liệu tham khảo

[1] Kỹ thuật vi xử lý, Văn Thế Minh, NXB Giáo dục, 1997

[2] Kỹ thuật vi xử lý và Lập trình Assembly cho hệ vi xử lý, Đỗ Xuân Tiến, NXB Khoa học & kỹ thuật, 2001

Chương 6
Các kiểu I/O

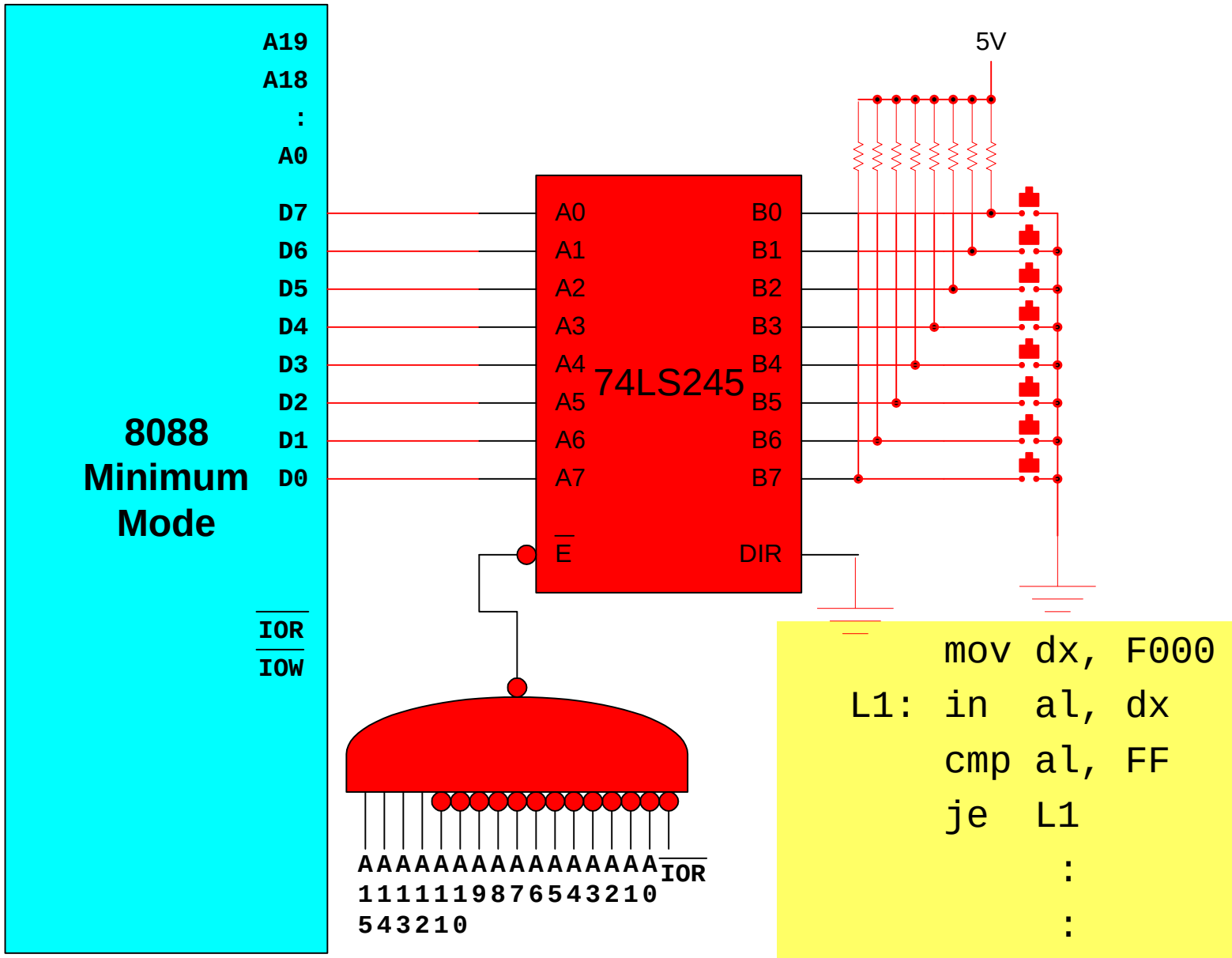
6.1 Thăm dò (Polling)

6.2 I/O điều khiển bằng ngắt (Interrupt)

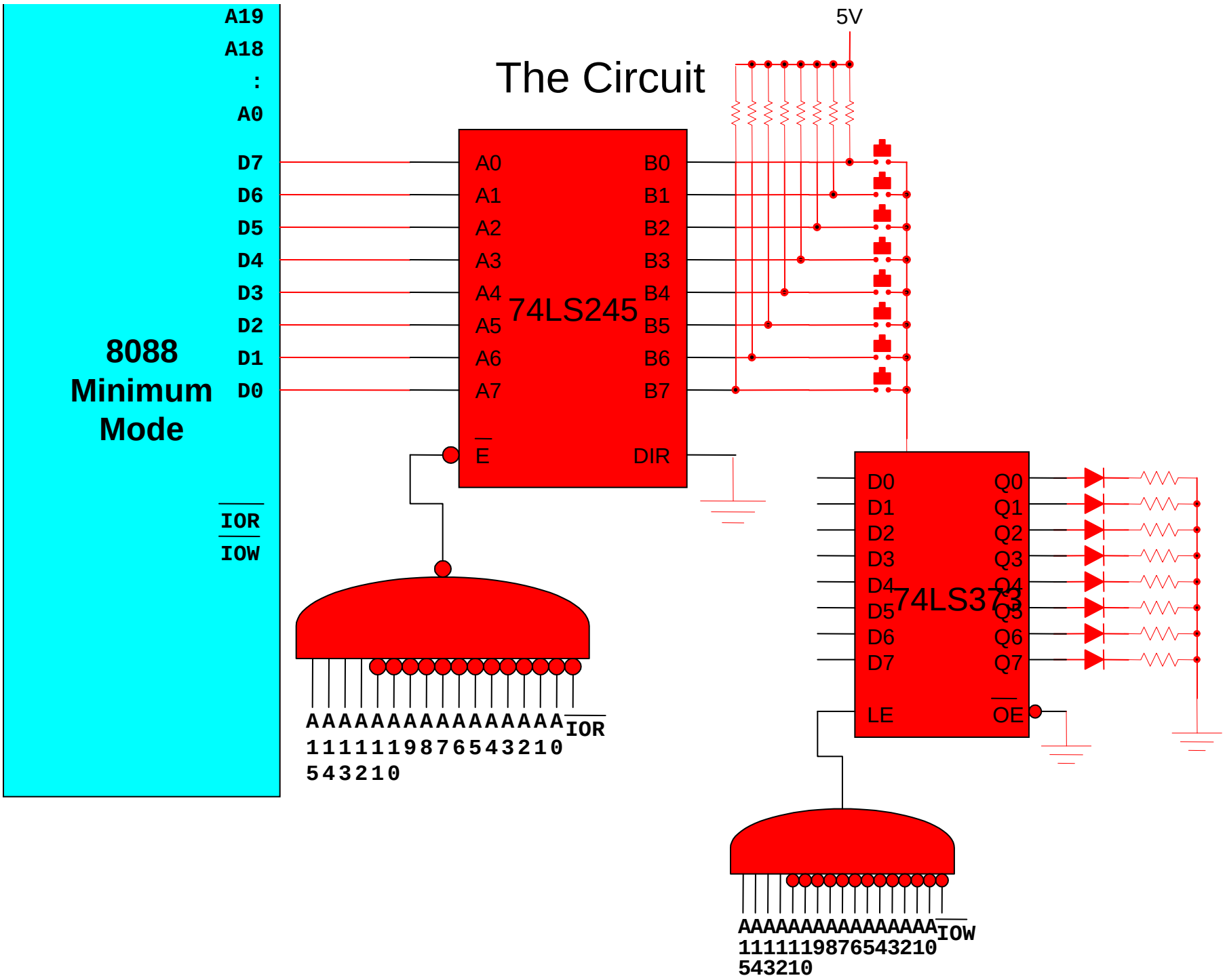
-

6.3 DMA (Direct Memory Access)

Thăm dò




```
      mov dx, F000
L1:   in  al, dx
      cmp al, FF
      je  L1
          :
          :
```


The Circuit
5V
8088
Minimum
Mode
A19
A18
:
A0
D7
D6
D5
D4
D3
D2
D1
D0
IOR
IOW
74LS245
A0
A1
A2
A3
A4
A5
A6
A7
B0
B1
B2
B3
B4
B5
B6
B7
E
DIR
74LS373
D0
D1
D2
D3
D4
D5
D6
D7
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
LE
OE
AAAAAAAAAAAAAAAA
1111119876543210
543210
IOR
AAAAAAAAAAAAAAAA
1111119876543210
543210
IOW

Bài toán

- Chương trình tạo ra hiệu ứng “LED chạy”:
  - Ban đầu LED chạy từ trên xuống
  - Khi nhấn phím thấp nhất thì LED thay đổi hướng chạy
  - Khi nhấn phím cao nhất thì chương trình kết thúc

Chương trình

```
        mov  dx, F000
        mov  ah, 00
        mov  al, 01
L1:     out  dx, al
        mov  cx, FFFF
L2:     dec  cx
        jnz  L2
        cmp  ah, 00
        jne  L3
        rol  al, 1
        cmp  al, 01
        jne  L1
        jmp  L4
L3:     ror  al, 1
        cmp  al, 80
        jne  L1
```

```
L4:     mov  bl, al
        in   al, dx
        cmp  al, FF
        je  L6
        test al, 01
        jnz  L5
        xor  ah, FF
        jmp  L6
L5:     test al, 80
        jz   L7
L6:     mov  al, bl
        jmp  L1
L7:
```

What's the problem with polling in the sample program?

- Running LED takes time
- User might remove his/her finger from the switch
- before the `in al, dx` instruction is executed
- the microprocessor will not know that the user has pressed the button

Problem with Polling

```
        mov  dx, F000
        mov  ah, 00
        mov  al, 01
L1:     out  dx, al
        mov  cx, FFFF
L2:     dec  cx
        jnz   L2
        cmp  ah, 00
        jne  L3
        rol  al, 1
        cmp  al, 01
        jne  L1
        jmp  L4
L3:     ror  al, 1
        cmp  al, 80
        jne  L1
```

```
L4:     mov  bl, al
        in   al, dx
        cmp  al, FF
        je  L6
        test al, 01
        jnz  L5
        xor  ah, FF
        jmp  L6
L5:     test al, 80
        jz   L7
L6:     mov  al, bl
        jmp  L1
L7:
```

Interrupt

- The microprocessor does not check if data is available.
- The peripheral will interrupt the processor when data is available

Polling vs. Interrupt

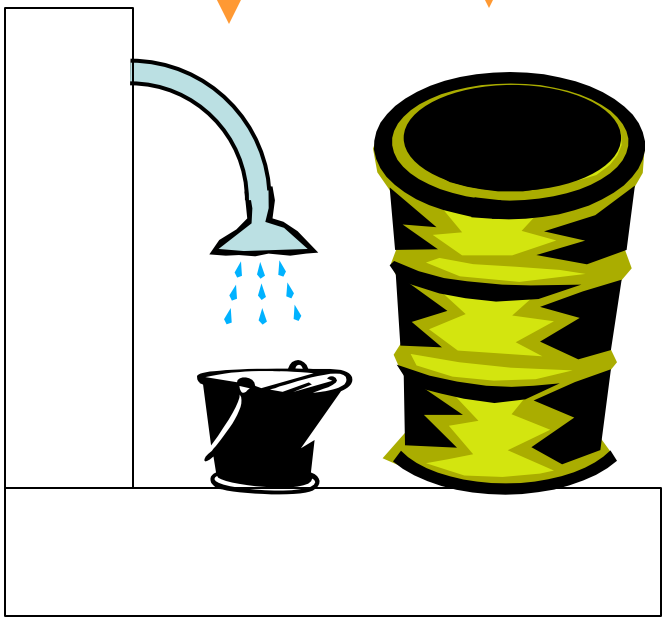
instruction

While studying, I'll check the bucket every 5 minutes to see if it is already full so that I can transfer the content of the bucket to the drum.

μP

Input Device

Memory



POLLING

Polling vs. Interrupt