Digital Electronics

Course Contents

- Introduction.
- Number Systems.
- Digital Electronics.
  - Basics, Gates , Flip Flops, Minimisation
- Computers.
- Communications.
- Micro Processors and Controllers.
- Device interfacing.

Introduction

Introduction.

- Number Systems.
- Number Conversion.
- Digital Logic.
- Flip Flops.
- Logic Minimisation.

Number Systems

Number Systems.

- You have a simple Machine / Computer.
- You wish to process numbers in the range +999,999 to -1,000,000 to an accuracy of say 10 decimal places.
- The Machine only understands Integer values in the range 0 to 255 or +127 to -128.
- The Machine has no Multiply or Divide capability and you may require these for your calculations. How do you proceed ?

Number Systems.

- (Back to Basics) Simple Counting :-
- One, Two, Three, Four, Five, Six, Seven, Eight, Nine .. Remember we started at Zero.
- Gives us :- 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 our basic counting digits.
- We could extend :-
- Ten, Eleven, Twelve, Thirteen, Fourteen, Fifteen, Sixteen, Seventeen, Eighteen, Nineteen ...

Number Systems.

- We could extend the sequence:-
- Ten, Eleven, Twelve, Thirteen, Fourteen, Fifteen, Sixteen, Seventeen, Eighteen, Nineteen ...
- 10=T, 11=E, 12=W, 13=H, 14=F, 15=I, 16=S, 17=V, 18=G, 19=N
- the sequence so far :-
- 0,1,2,3,4,5,6,7,8,9,T,E,W,H,F,I,S,V,G,N
- !!!! Not very memorable !!!!

Number Systems.

- An Alternative could be to use the Alphabet sequence so as before :-
- 10=A, 11=B, 12=C, 13=D, 14=E, 15=F, 16=G, 17=H, 18=I, 19=J
- the sequence so far :-
- 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F,G,H,I,J
- !!!! At least it is memorable !!!!
- So where next ?
- Simple Addition using counting.

Number Systems.

Simple Addition using counting

7 + 5 =

1	2	3	4	5	6	7	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	0	1	2	3	4	5	.	.	.	.
1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Gives us C and from previous table translates to Twelve.

Number Systems.

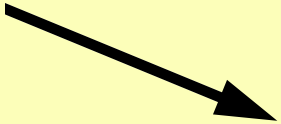
- Simple Addition exercise.
- 569 + 123 =
- Solution is 692
-
- 326 + 549 =
- Solution is 875
- ? How did we do the last calculation ?

Number Systems.

Vertical Group Addition (Remember only 1 digit per column)

.	.	.	.	3	2	6
.	.	.	.	5	4	9
.	.	.	.	.	.	.

Count the values in the columns



.	.	.	.	3	2	6
.	.	.	.	5	4	9
.	.	.	.	8	6	F

10=A, 11=B, 12=C, 13=D, 14=E, 15=F, 16=G, 17=H, 18=I, 19=J

Number Systems.

Vertical Group Addition (Remember only 1 digit per column)

.	.	.	.	3	2	6
.	.	.	.	5	4	9
.	.	.	.	.	.	.

Count the values in the columns

.	.	.	.	3	2	6
.	.	.	.	5	4	9
.	.	.	.	8	6	F

NOW Repeat next phase as often as required

10=A, 11=B, 12=C, 13=D, 14=E, 15=F, 16=G, 17=H, 18=I, 19=J

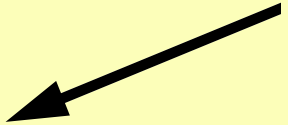
Number Systems.

Vertical Group Addition (Remember only 1 digit per column)

Convert columns with the Excess values ie. “F” translates to base column = 5 and next column on the left = 1

Excess values are any value that need more that one column.

.	.	.	.	3	2	6
.	.	.	.	5	4	9
.	.	.	.	8	6	F



.	.	.	.	8	6	0
.	.	.	.	0	1	5
.	.	.	.	8	7	5

Count the values in the columns to give the final result

10=A, 11=B, 12=C, 13=D, 14=E, 15=F, 16=G, 17=H, 18=I, 19=J

Number Systems.

- Simple Subtraction exercise.
- 569 - 123 =
- Solution is 446
-
- 329 - 546 =
- Solution is -217
- ? How did we perform the last calculation ?

Number Systems.

Simple Subtraction by counting backwards

9 - 6 =

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
.	.	.	6	5	4	3	2	1	0	.	.	.	.	.	.
.	.	→	**3**	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Gives us **3**

Number Systems.

Vertical Group Addition (Remember only 1 digit per column)

.	.	.	.	3	2	9
.	.	.	.	5	4	6
.	.	.	.	.	.	.

Count backwards the values in the columns

.	.	.	.	3	2	9
.	.	.	.	5	4	6
.	.	.	.	?	?	3

The first column OK , however the next column gives a **MAJOR** problem...

10=A, 11=B, 12=C, 13=D, 14=E, 15=F, 16=G, 17=H, 18=I, 19=J

Number Systems.

Simple Subtraction by counting backwards

2 - 4 =

-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8
.	.	.	.	.	4	3	2	1	0	.	.	.	.	.	.
-7	-6	-5	-4	-3	**-2**	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Gives us **-2**

Number Systems.

- The problem **Number one** is each column can only contain a single character.
- Problem **Number two** is having columns that contain **Positive** values and columns that contain **Negative** values is really confusing.
- Lets consider the difficulty from another direction :-

Number Systems.

- Solution Number 1.
- If largest number first --- Subtract values.
- If largest number second --- Reverse order of values --- Subtract values --- Prefix answer with minus sign.
- **Question** --- If this is a Machine doing the calculation how will it know which is the largest number ?
- **Answer** --- Subtract the values ????

Number Systems.

A little bit more of Number revision.

Number Systems.

What do we mean by the number 321.45 ?

- First we have 3 Hundreds
- Then we have 2 Tens
- and we have 1 Unit
- and we have 4 Tenths
- and we have 5 Hundredths

Number Systems.

What do we mean by the number 321.45 ?

- Alternatively
- First we have 3 x 10 x 10 (or 3 x 100)
- Then we have 2 x 10
- and we have 1 x 10/10 (or 1 x 1)
- and we have 4 x 1/10
- and we have 5 x 1/10 x 1/10
  (or 5 x 1/100)

Number Systems.

What do we mean by the number 321.45 ?

- Alternatively
- First we have 3 x 10 x 10 (or 3 x 100)
- Then we have 2 x 10
- and we have 1 x 10/10 (or 1 x 1)
- and we have 4 x 1/10
- and we have 5 x 1/10 x 1/10
  (or 5 x 1/100)

Note

Each Time Digit Base Value Divides by Ten.

Number Systems.

Powers of 10

- 10 x 10 is the same as 10^2 or 10^2
- 10 x 10 x 10 is the same as 10^3 or 10^3
- 1 / 10 is the same as 10^{-1} or 10^-1
- Any number raised to power 0 is equal to 1
- 1 is the same as 10^0 or 10^0

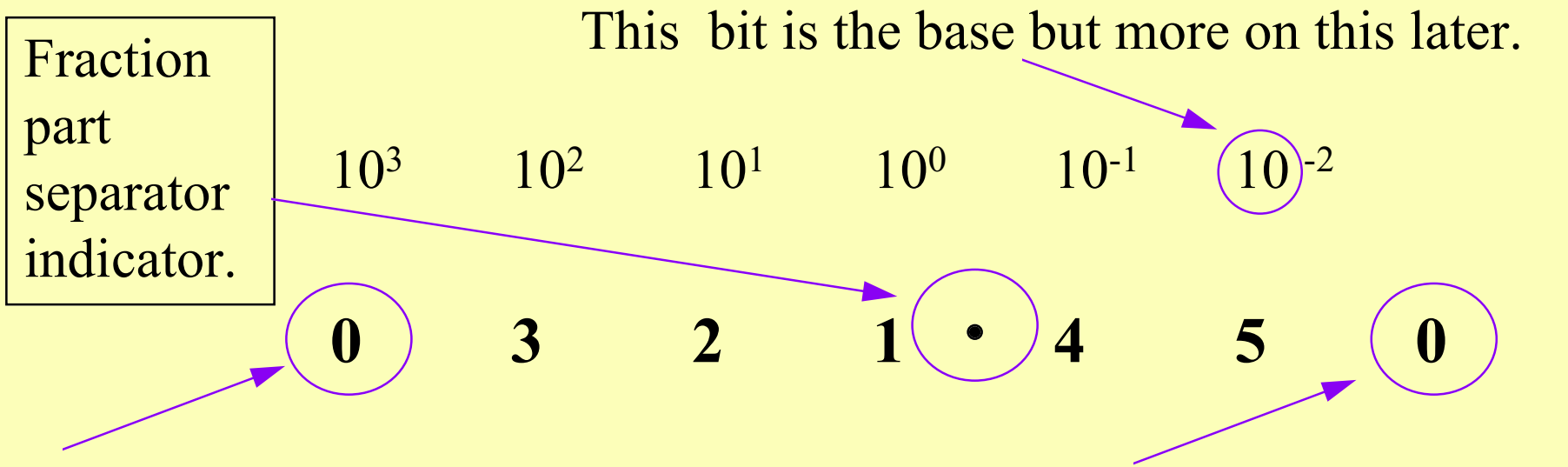
Number Systems.

What do we mean by the number 321.45 ?

- Alternatively in mathematical terms
- First we have 3 x 10^2 or 3×10^2
- Then we have 2 x 10 ^1 or 2×10^1
- and we have 1 x 10^0 or 1×10^0
- and we have 4 x 10^-1 or 4×10^{-1}
- and we have 5 x 10^-2 or 5×10^{-2}

Number Systems.

What do we mean by the number 321.45 ?



The implicit number base representation format.

Number Systems.

- Terminology:
- The count of the different digits that can be put in any single column is called the Number **Base**.
- Our normal counting system has the following digits 0,1,2,3,4,5,6,7,8,9 therefore the **Base** of our counting system is **10**.
- If the counting digit of another system was 0,1,2,3,4,5 then its **Base** would be **6**.

Number Systems.

- Lets us consider an alternative way of representation numbers to distinguish if a number is Positive or Negative.
- If all the numbers we were working with are of a fixed width ... this could give us

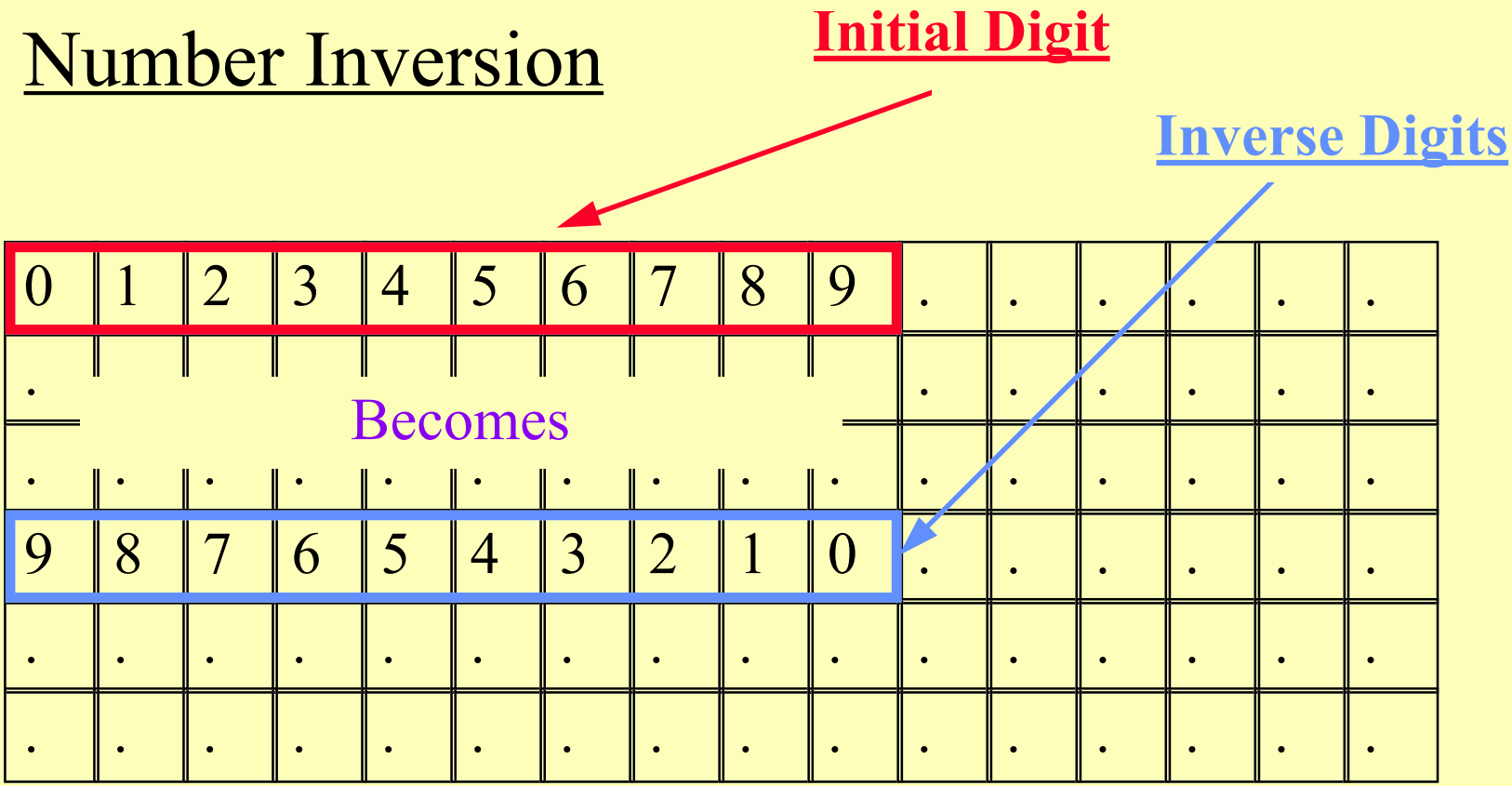
Two important Questions ...

(1) Could Number Inversion be of any help ?

(2) What is Number Inversion ?

Number Systems.

Number Inversion



How did we calculate the Inversion digits.

We subtracted all the Initial digits from (Base -1).

or count backwards from largest value.

Number Systems.

Number Inversion

0123 represents +123 ← **Initial Number**

Becomes

9876 represents -123 ← **Inverse Number**

0000 represents +0 ← **Initial Number**

However

9999 represents -0 ← **Inverse Number**

Minor problem here because we have two version of Zero ie. +0 and -0 and this may cause problems in calculation.

Number Systems.

- The Zero problem could be overcome by adding one to the inversion value each time the process is used.

0000 represents +0 ← **Initial Number**

Becomes

9999 represents -0 ← **Inverse Number**

Add One and it Becomes

0000 represents 0 ← **Complement Number**

This way **ZERO** only has one numeric representation form.

Number Systems.

- Lets check method with other values.

0123 represents +123 ← **Initial Number**

Becomes

9876 represents -123 ← **Inverse Number**

Add One and it Becomes

9877 represents -123 ← **Complement Number**

The Reverse Process gives ...

0122 represents +123 ← **Inverse Number**

Add One and it Becomes

0123 represents +123 ← **Initial Number**

Number Systems.

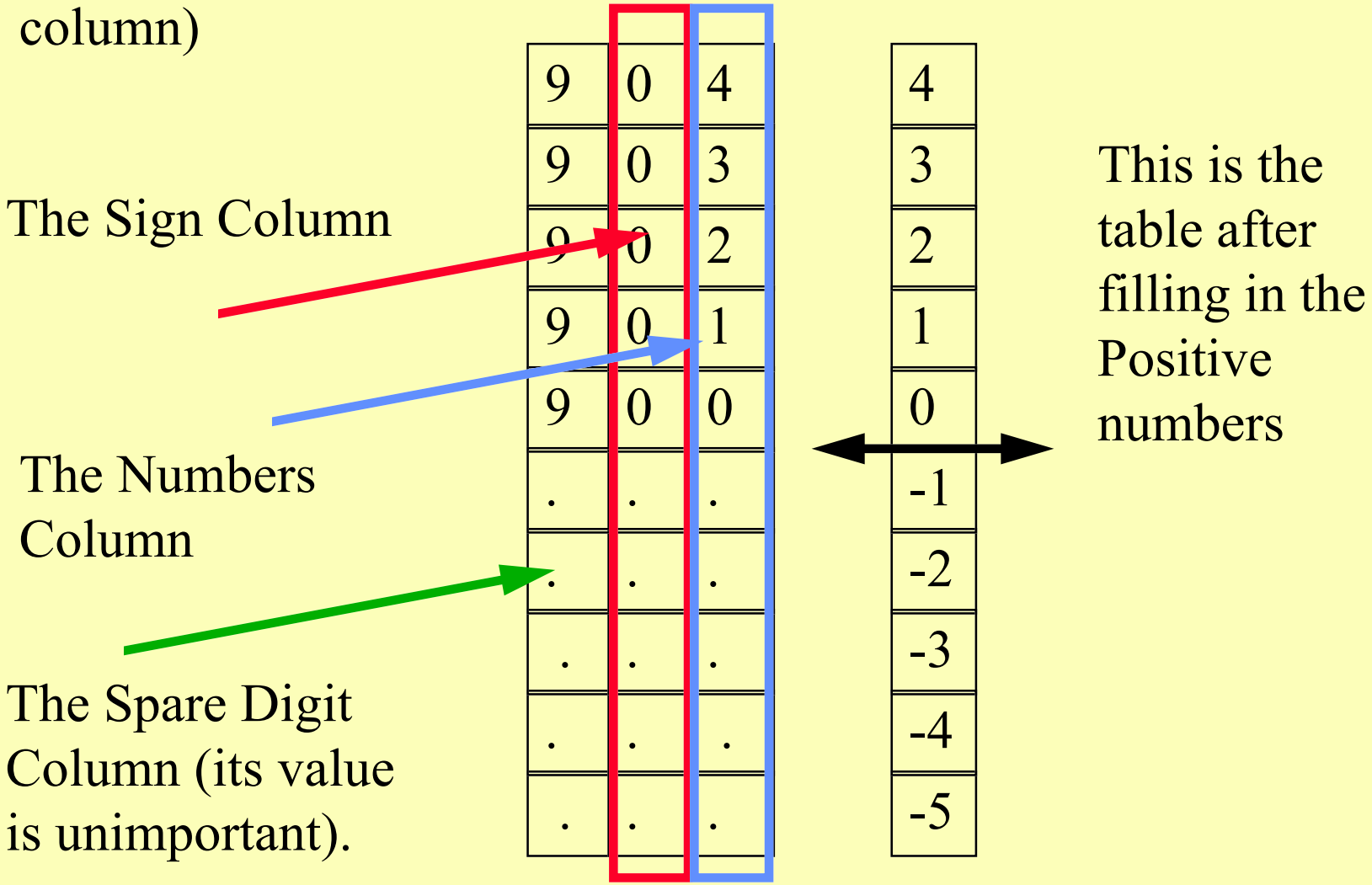
- Summary:
- The **Complement** system is **reversible**.
- Now all we need to do is see if it works with mathematics the way we expect the system to do.

Number Systems.

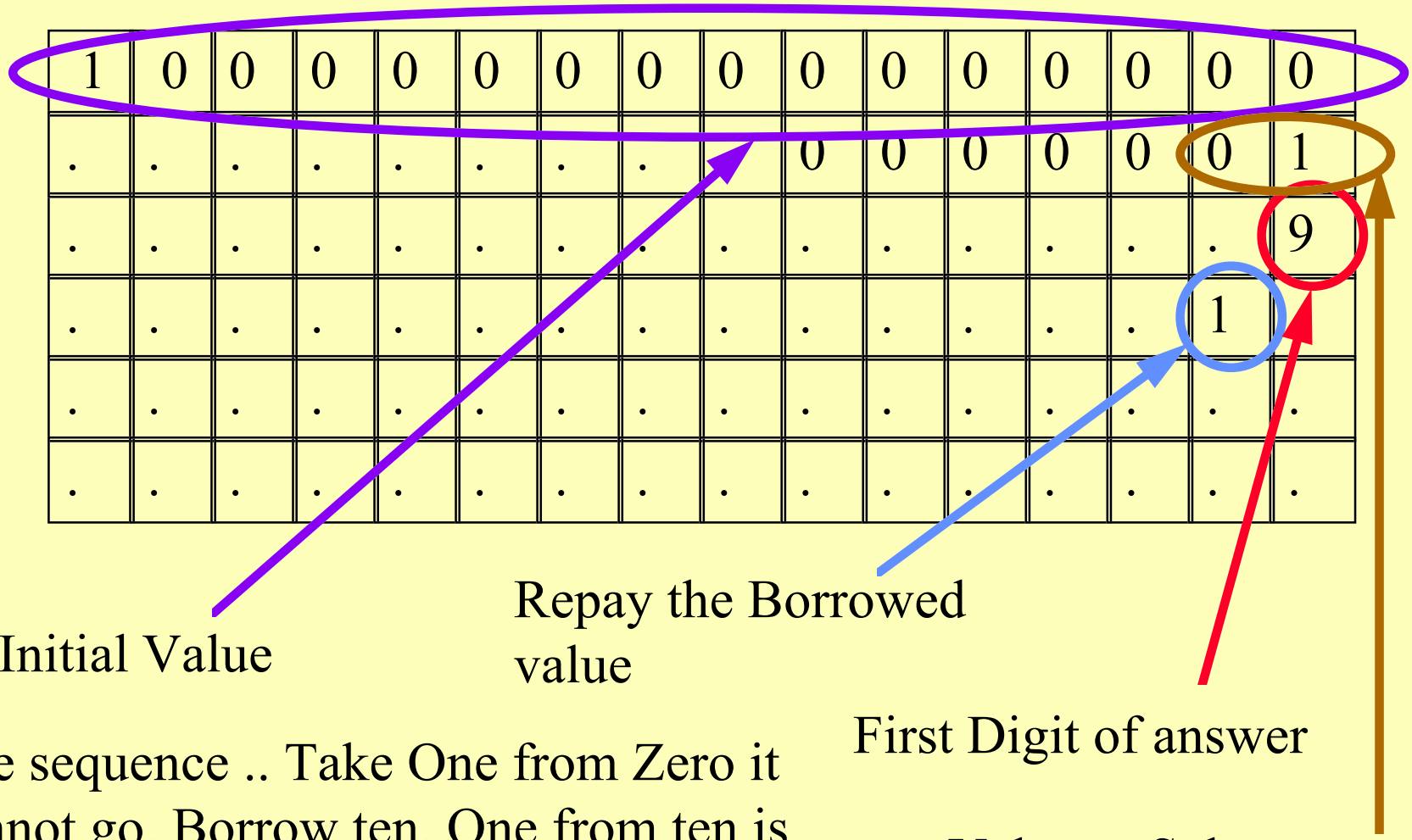
- Lets us consider the situation where we always work with a fixed number of columns for our calculations.
- (1) To practice we will assume that we only have one column numeric sums.
- (2) However we will always allocate one additional column so that we can use it to remember whether the overall number is positive or negative. (The Sign Digit)

Number Systems.

Single column mathematics (Remember only 1 digit per column)



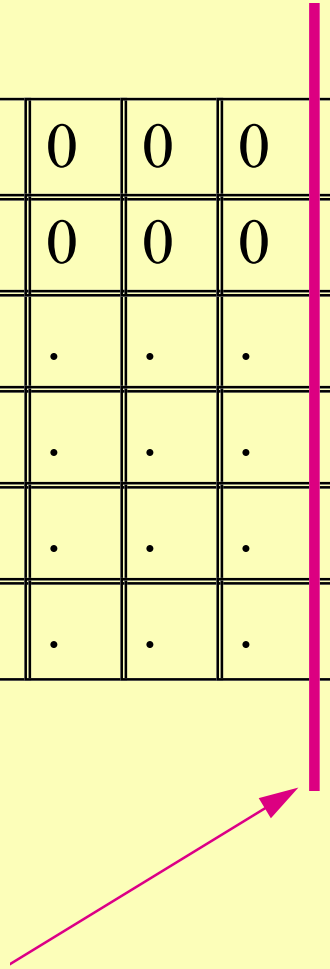
Number Systems.



The sequence .. Take One from Zero it cannot go. Borrow ten. One from ten is nine. Pay back the ten in next column..

Number Systems.

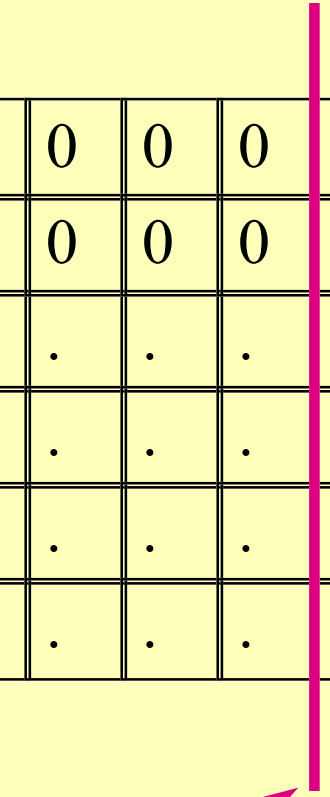
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
.	.	.	.	.	.	.	.	.	0	0	0	0	0	0	1
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	9
.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.



Continue the subtractions sequence to this line.

Number Systems.

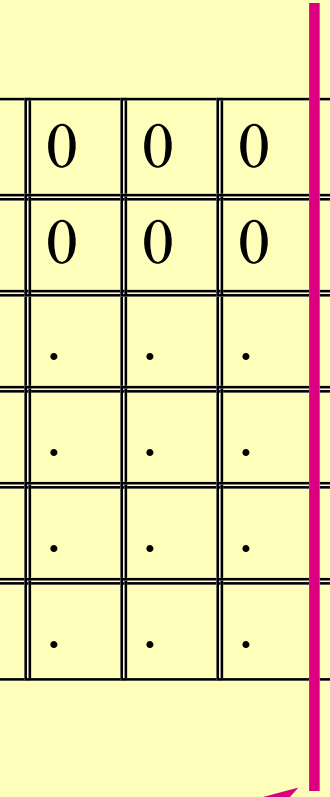
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
.	.	.	.	.	.	.	.	.	0	0	0	0	0	0	1
.	.	.	.	.	.	.	.	.	.	.	.	.	.	9	9
.	.	.	.	.	.	.	.	.	.	.	.	.	1	1	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.



Continue the subtractions sequence to this line.

Number Systems.

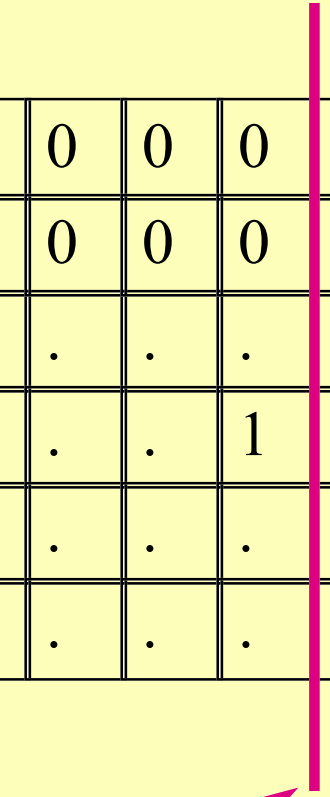
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
.	.	.	.	.	.	.	.	.	0	0	0	0	0	0	1
.	.	.	.	.	.	.	.	.	.	.	.	.	9	9	9
.	.	.	.	.	.	.	.	.	.	.	.	1	1	1	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.



Continue the subtractions sequence to this line.

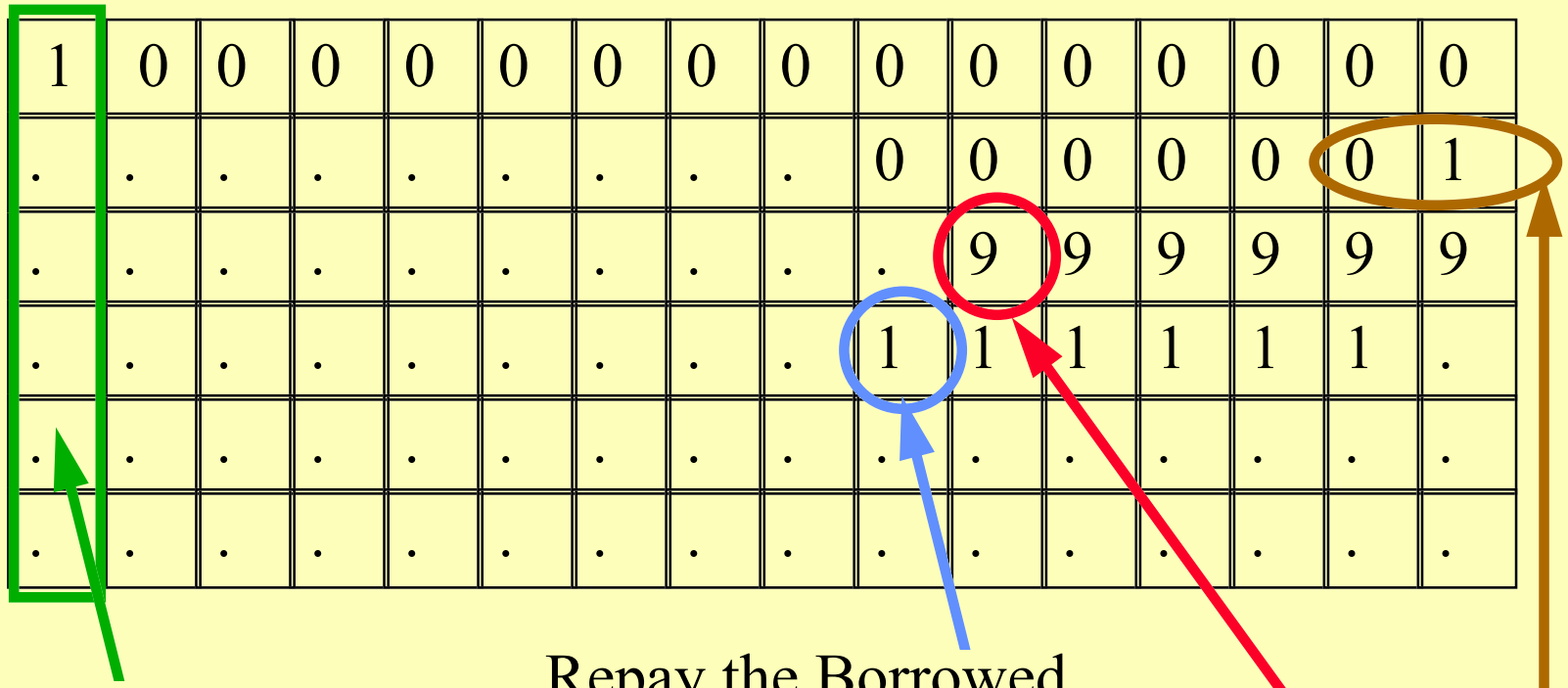
Number Systems.

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
.	.	.	.	.	.	.	.	.	0	0	0	0	0	0	1
.	.	.	.	.	.	.	.	.	.	.	.	9	9	9	9
.	.	.	.	.	.	.	.	.	.	.	1	1	1	1	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.



Continue the subtractions sequence to this line.

Number Systems.



1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
.	.	.	.	.	.	.	.	.	0	0	0	0	0	0	1
.	.	.	.	.	.	.	.	.	.	9	9	9	9	9	9
.	.	.	.	.	.	.	.	.	1	1	1	1	1	1	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Spare Digit Column

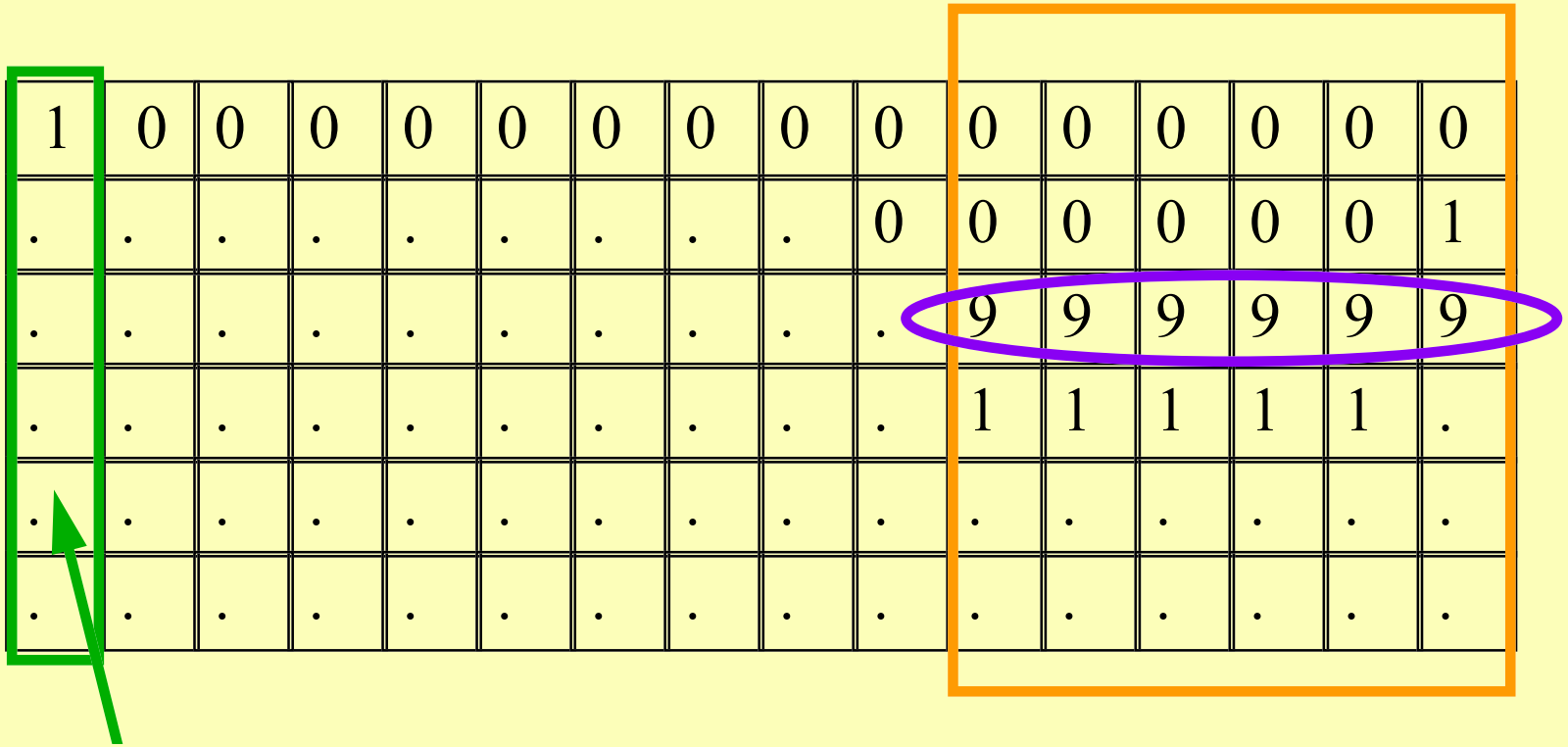
Repay the Borrowed value

Next Digit of answer

Value to Subtract

The sequence .. By now you should see a pattern building up. What if we keep all our calculations in first six columns.

Number Systems.



1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
.	.	.	.	.	.	.	.	.	0	0	0	0	0	0	1	
.	.	.	.	.	.	.	.	.	.	9	9	9	9	9	9	
.	.	.	.	.	.	.	.	.	.	1	1	1	1	1	.	
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	

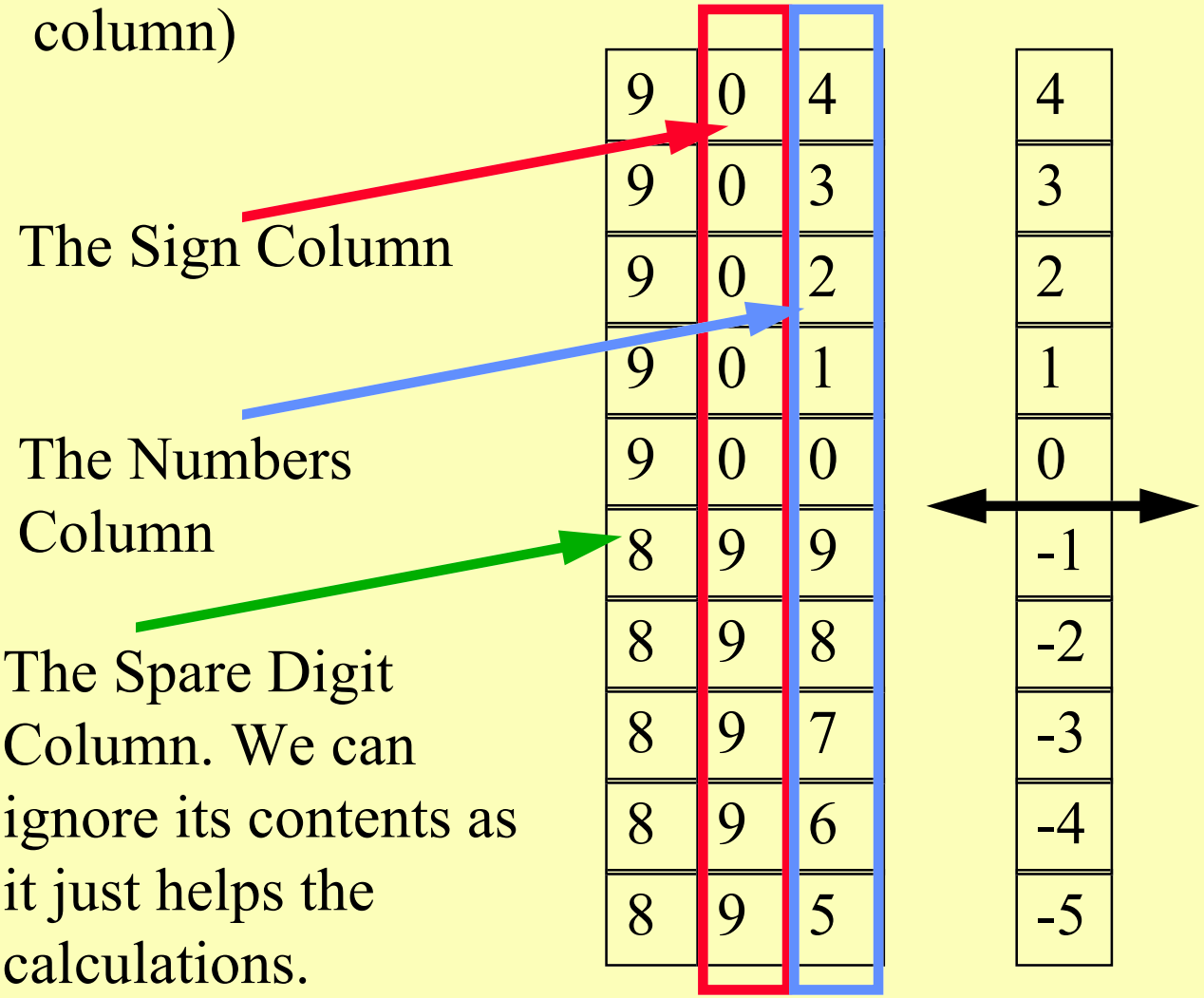
Spare Digit Column is just used to help the calculation process.

Summary

When we keep to say just six columns then
000000 - 000001 = 999999 (or -1)

Number Systems.

Single column mathematics (Remember only 1 digit per column)



Number Systems.

- Mathematical tricks:
- Consider the calculation 423 - 124.
- Would this be any different to:-
- 423 + (-124)
- Try it on your calculator.
- So provided we can find a method to make numbers **NEGATIVE** we only need to know how to perform the mathematical ADDITION process. (Any ideas ?????)

Number Systems.

- So let try real calculations BUT First.
- We need to set some ground rules:-

1 We need to define how many columns we are going to use. (We will use 6 columns)

2 If a number is negative we need to convert it to its complementary form.

3 If we want to Subtract we complement the number and then we Add.

Number Systems.

- So let try real calculations 367 + 234 = ?

367 =	000367
234 =	000234
	======
	. . . . .

Number Systems.

- So let try real calculations 367 + 234 = ?

367 =	000367
234 =	000234
	======
	000601

Number Systems.

- So let try real calculations 367 - 234 = ?
- or 367 + (-234)

234 =	000234
Inversion =	999765
Complement =	999766
367 =	000367
-234 =	999766
	======
	000133 = +133

Number Systems.

- So let try real calculations 234 - 376 = ?
- or 234 + (-376)

376 =	000376
Inversion =	999623
Complement =	999624
234 =	000234
-376 =	999624
	======
	. . . . . = ?

Number Systems.

- So let try real calculations 234 - 376 = ?
- or 234 + (-376)

376 =	000376
Inversion =	999623
Complement =	999624
234 =	000234
-376 =	999624
	======
	999858 = ?

Number Systems.

- So let try real calculations 234 - 376 = ?
- or 234 + (-376)

376 =	000376
Inversion =	999623
Complement =	999624
234 =	000234
-376 =	999624
	======
	999858 = ?

As this digit is the (Base value -1) then the number must be negative.

So we need to do a bit more processing to get a recognised value.

Number Systems.

- So let try real calculations 234 - 376 = ?
- or 234 + (-376)

234 =	000234	
-376 =	999624	
	======	
	999858	= negative value
Inversion =	000141	
Complement =	000142	
		= -142

Number Systems.

- More complex 234 - 376 + 576 - 195 = ?
- or 234 + (-376) + 576 + (-195)

376 =	000376	
Inversion =	999623	
Complement =	999624	= -376

195 =	000195	
Inversion =	999804	
Complement =	999805	= -195

Number Systems.

- More complex 234 - 376 + 576 - 195 = ?
- or 234 + (-376) + 576 + (-195)

234 =	000234
-376 =	999624
576 =	000576
-195 =	999805
	======
	000239 = +239

With normal mathematics this process would have taken three separate calculations sums.

Number Systems.

- If we want to Subtract we complement the number and then we Add.
- How about the Multiply process ?
- Easy we just count a number of Additions.
- Example 3 * 25 = 25 + 25 + 25
- How about the Division process ?
- We just count the number of complement Additions until the value reduces to zero.
- Example 8 /2 = 8-2(1) →6-2(2) →4-2(3) →2-2(4) →0 therefore 8/2 = 4

Number Systems.

- Summary:
- To complete Addition , Subtraction , Multiplication and Division the only skill we need is to be able to count in a Positive or Negative direction.
- We need to ensure we always to a fixed number of columns.
- We need to remember the Complementary number conversion process.
- Note this process works for all number Bases.

Number Conversion.

Number Conversion.

- How do we convert numbers from one Base to another ?
- Example:
- Convert the Decimal number 37 (in Base 10) to its equivalent (in Base 2).
- General Reminder :-
- Base 10 Digits are 0,1,2,3,4,5,6,7,8,9 and weighted *100, *10, *1
- Base 2 Digits are 0,1 and weighted *32, *16, *8, *4, *2, *1

Example. **Number Conversion.**

- Convert the Decimal number 37 (in Base 10) to its equivalent (in Base 2).
- The "Process" used is division by Base value and record the division remainder as so:-
- 37 / 2 = 18 r 1
- 18/ 2 = 9 r 0
- 9 / 2 = 4 r 1
- 4/ 2 = 2 r 0
- 2 / 2 = 1 r 0
- 1 / 2 = 0 r 1

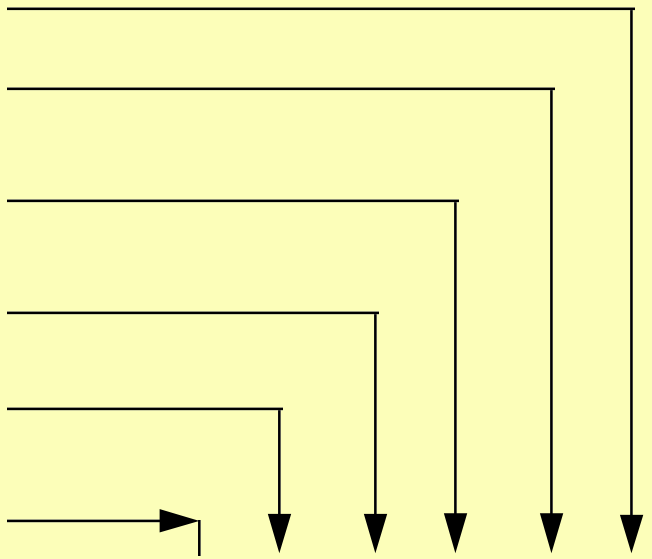
Example. **Number Conversion.**

- Convert the Decimal number 37 (in Base 10) to its equivalent (in Base 2).
- The Process used is division by Base value and record the division remainder as so:-
- 37 / 2 = 18 r 1
- 18/ 2 = 9 r 0
- 9 / 2 = 4 r 1
- 4/ 2 = 2 r 0
- 2 / 2 = 1 r 0
- 1 / 2 = 0 r 1 (No more stages **So stop**)

Example.

Number Conversion.

- Convert the Decimal number 37 (in Base 10) to its equivalent (in Base 2).
- Now copy the digits out in correct order.
- 37 / 2 = 18 r 1
- 18 / 2 = 9 r 0
- 9 / 2 = 4 r 1
- 4 / 2 = 2 r 0
- 2 / 2 = 1 r 0
- 1 / 2 = 0 r 1



- Binary or Base 2 value is **1 0 0 1 0 1**

Example. # Number Conversion.

- Converting back the other way to Base 10
- Binary or Base 2 value is **1 0 0 1 0 1**
- Gives us starting from RHS:-
- 1 * 1 = 1
- 0 * 2 = 0
- 1 * 4 = 4
- 0 * 8 = 0
- 0 * 16 = 0
- 1 * 32 = 32

Example.

Number Conversion.

- Converting back the other way
- Binary or Base 2 value is **1 0 0 1 0 1**
- Gives us starting from RHS:-
- 1 * 1 = 1
- 0 * 2 = 0
- 1 * 4 = 4
- 0 * 8 = 0
- 0 * 16 = 0
- 1 * 32 = 32

Add all these values together to revert value back to a Decimal number that we can recognise.

Example.

Number Conversion.

- Converting back the other way
- Binary or Base 2 value is **1 0 0 1 0 1**
- Gives us starting from RHS:-

• 1 * 1	= 1
• 0 * 2	= 0
• 1 * 4	= 4
• 0 * 8	= 0
• 0 * 16	= 0
• 1 * 32	= 32
	37

Add all these values together to revert value back to a Decimal number that we can recognise.

Common Bases.

Number Conversion.

- General Reminder :-
- Base 2 Digits are 0, 1 and weighted *128, *64, *32, *16, *8, *4, *2, *1.
- Base 8 Digits are 0, 1, 2, 3, 4, 5, 6, 7 and weighted *4096, *512, *64, *8, *1.
- Base 10 Digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and weighted *10000, *1000, *100, *10, *1.
- Base 16 Digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F and weighted *65536, *4096, *256, *16, *1.

Number Conversion.

- Hexadecimal (HEX) or Base 16 numbers.
- This is a special case where it is often simpler to convert the number to a Base 2 value.
- Then sub-divide the Base 2 value into blocks of FOUR digits
- Finally translate each block to its equivalent HEX value.
- Example: 1001111101010111
- Split 1001111101010111
- becomes 9 F 5 7

Number Conversion.

- Octal (OCT) or Base 8 numbers.
- Again a special case where it is often simpler to convert the number to a Base 2 value.
- Then sub-divide the Base 2 value into blocks of THREE digits
- Finally translate each block to its equivalent Octal value.
- Example: 1001111101010111
- Split 1001111101010111
- becomes 1 1 7 5 2 7

Digital Electronics

Section Contents

- Digital Logic
- Logic Functions.
  - Logic Circuit analysis
  - Karnaugh Maps
  - De-Morgan Theorem
- Communications.
- Device Interfacing.
- Computers.

Digital Logic

Digital Logic.

- The sections will cover :-
- The Binary, Octal, Hexadecimal numbers systems and Truth tables.
- Logic functions :-
  - AND , OR , INV or NOT , XOR
  - NAND , NOR
- DeMorgan theorem
- Karnaugh Maps
- Combinational Logic.

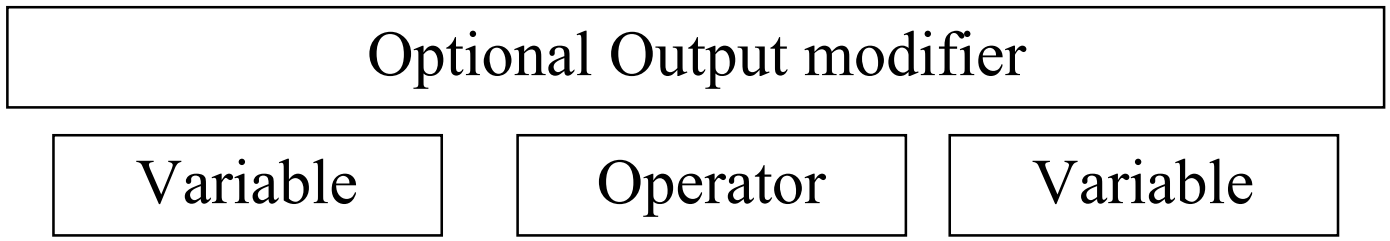
Logic Functions

Logic Functions.

- The sections will cover :-
- The Binary, Octal, Hexadecimal numbers systems and Truth tables.
- Logic functions :-
  - AND , OR , INV or NOT , XOR
  - NAND , NOR
- DeMorgan theorem
- Karnaugh Maps
- Combinational Logic.

Logic Functions.

- The Binary function consists of three parts.
- 1. The Input variables
- 2. The Output.
- 3. The Function operator.
- The function is written as follows :-



Where :-

The Operator is either {&, . , (Logic AND)} or {! , + , (Logic OR)}

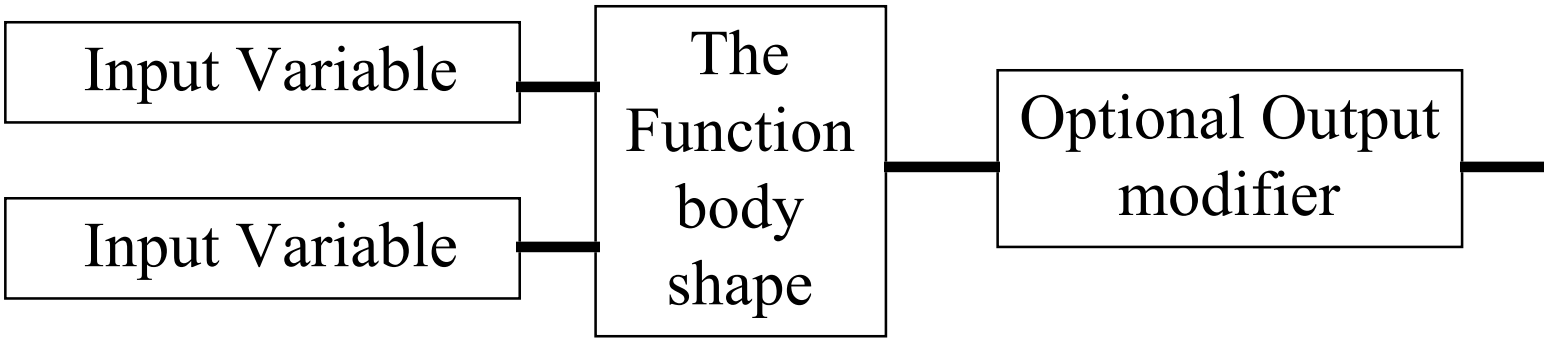
The Output modifier is a BAR when the function is Inverted (NOT).

Logic Functions.

- The Binary function examples.
- A {AND} B = A&B or AB or A.B
- A {OR} B = A!B or A+B
- A {OR} B inverted = $\overline{A!B}$ or $\overline{A+B}$
- NOT A {OR} B inverted = $\overline{\overline{A}!B}$ or $\overline{\overline{A}+B}$
- You can also bracket functions
- $\overline{(A\&B) + (C\&D+E)}$ translates to :-
- (A and B) or (C and D or E) with whole function being inverted
- Remember Always Process the bracket bit first.

Logic Functions.

- The Logic Symbols can also be considered to consists of three parts.
- 1. The Input variables
- 2. The Output.
- 3. The Function operator.
- The symbols are drawn as follows :-

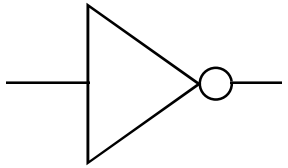


Logic Functions.

- The Truth Table is a formal method that defines how Binary Inputs relate to Binary Outputs.
- Normally the Inputs are defined of the left hand side of the table.
- The resultant or required Outputs are recorded on the right hand side of the table.

Binary Input Descriptions		Binary Output Descriptions	
The Binary Input patterns eg.	0000 0001 0010	00110001 00101001 01010101	The Binary Output patterns corresponding to their particular Inputs.

Logic Functions.



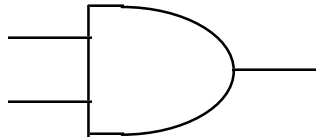
The NOT Gate

- The NOT or INV function.

. A	$\overline{A}$ or NOT A
. 0	1
. 1	0
. .	.
. .	.

Starting with the simplest logic function the NOT gate

Logic Functions.



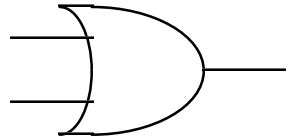
The AND Gate

- The AND function.

A	B	A&B , AB , A.B
0	0	0
0	1	0
1	0	0
1	1	1

Note: How the AND function is similar to the Arithmetic multiply function and is quite often written that way.

Logic Functions.



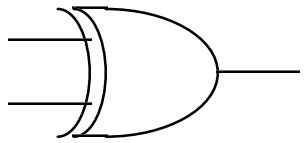
The OR Gate

- The OR function.

A	B	A ! B , A+B
0	0	0
0	1	1
1	0	1
1	1	1

Note: How the OR function is similar to the Arithmetic Addition function and is quite often written that way.

Logic Functions.



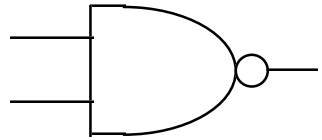
The XOR Gate

- The XOR or Exclusive OR function.

A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

Note: The Exclusive OR excludes the AND part of the OR function.

Logic Functions.



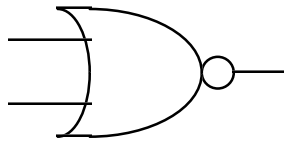
The NAND Gate

- The NAND function.

A	B	$\overline{A \& B}$
0	0	1
0	1	1
1	0	1
1	1	0

Note: NAND is just the NOT AND function.

Logic Functions.



The NOR Gate

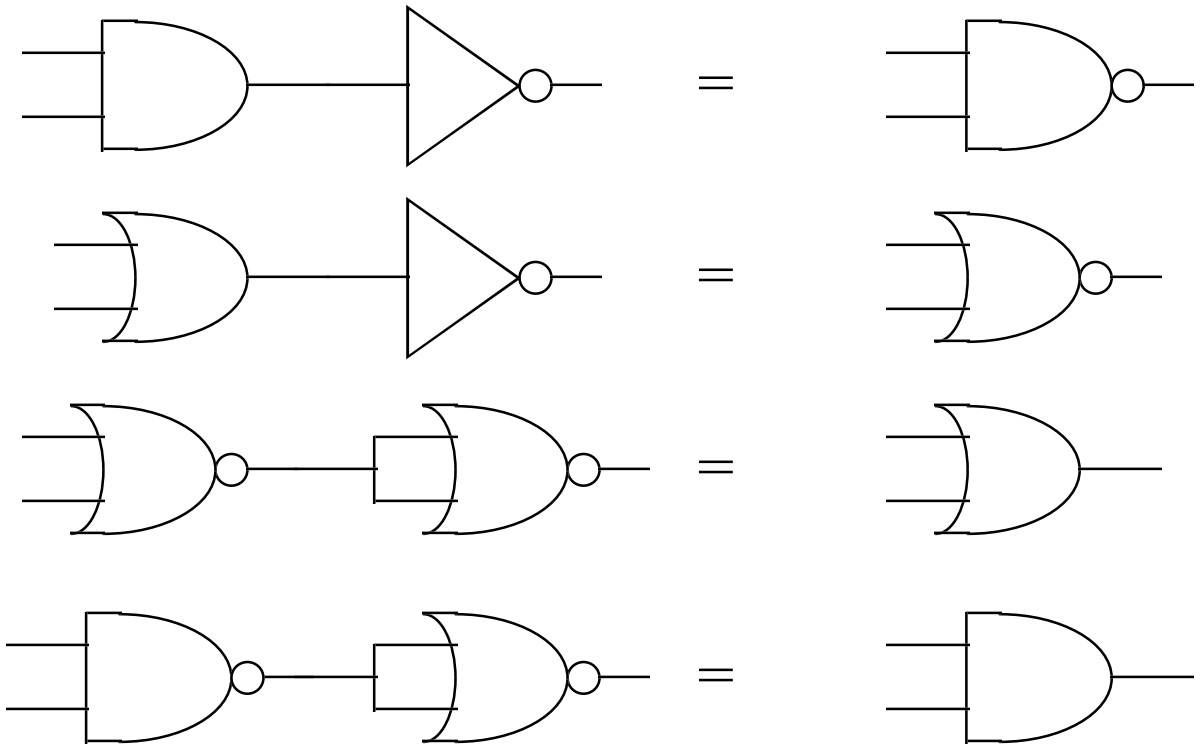
- The NOR function.

A	B	$\overline{A \,!\, B}$
0	0	1
0	1	0
1	0	0
1	1	0

Note: NOR is just the NOT OR function.

Logic Functions.

Using NOT , NAND and NOR Gates.



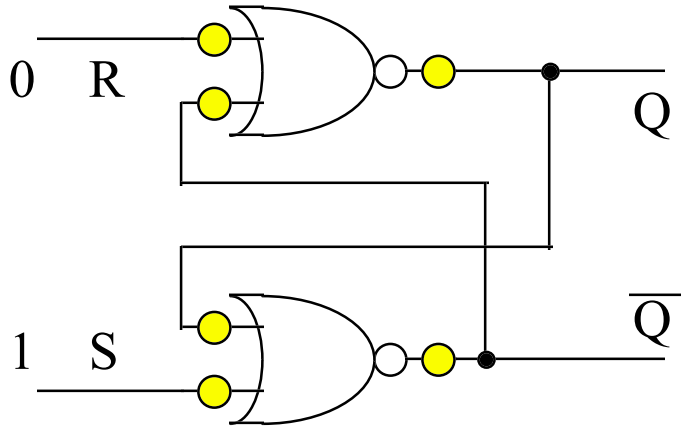
Basic Flip-Flops

● = 0
● = 1

Logic Functions.

NOR Gate

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0



Practice Page

Convert the Yellow Dots to the correct Logic Level

R = Reset (Reset to Logic Zero)

S = Set (Set to Logic One)

Q = General Code for an Output

$\overline{Q}$ = Bar Q = Inverse of Q

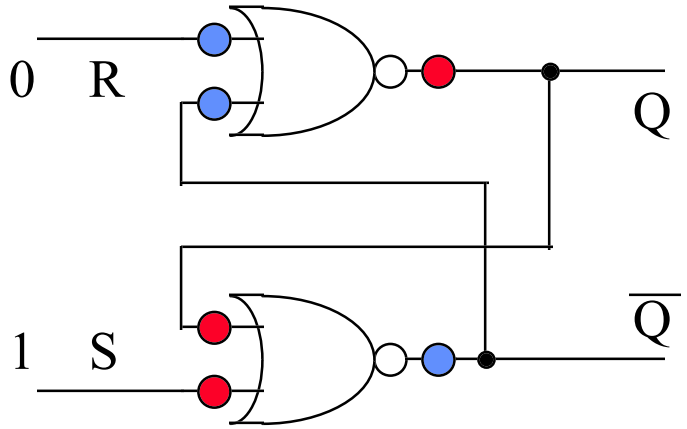
Cross coupled NOR gates to produce a Latch.

● = 0
● = 1

Logic Functions.

NOR Gate

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0



R = Reset (Reset to Logic Zero)

S = Set (Set to Logic One)

Q = General Code for an Output

$\overline{Q}$ = Bar Q = Inverse of Q

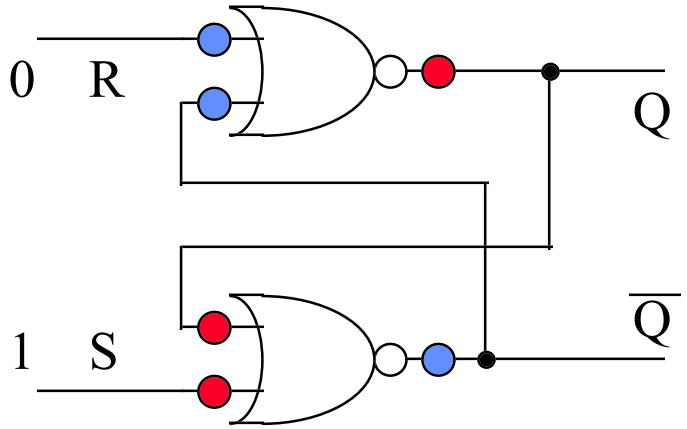
Cross coupled NOR gates to produce a Latch.

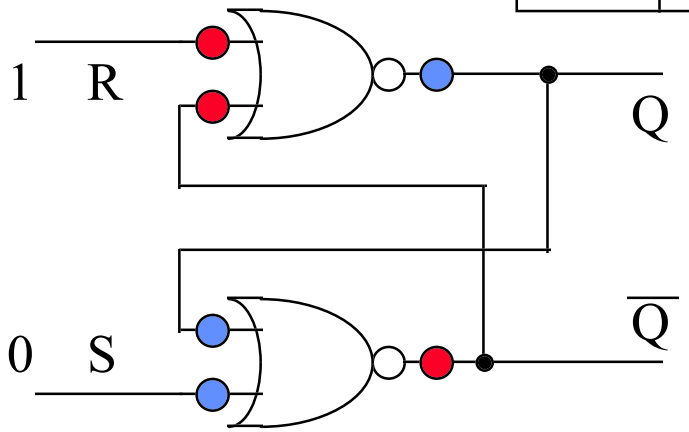
● = 0
● = 1

Logic Functions.

NOR Gate

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0





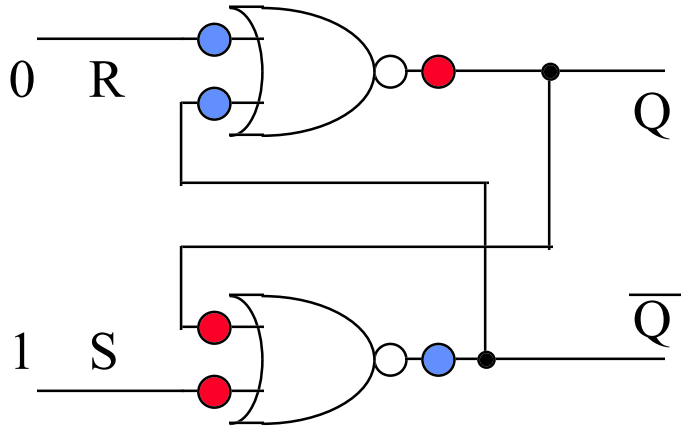
Cross coupled NOR gates to produce a Latch.

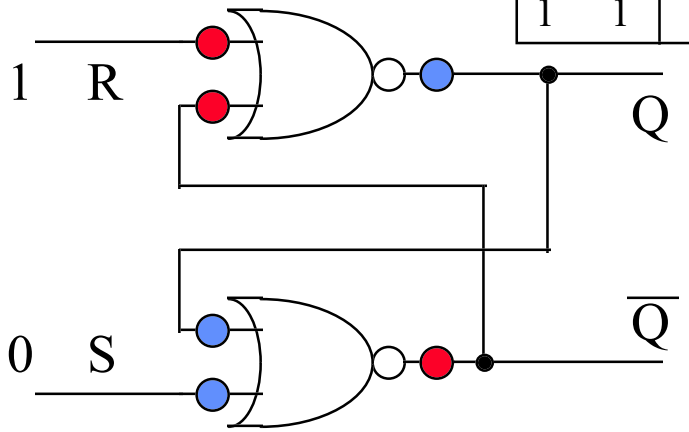
● = 0
● = 1

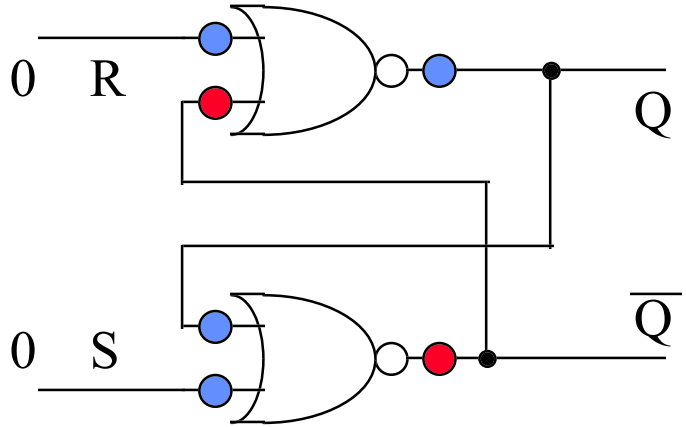
Logic Functions.

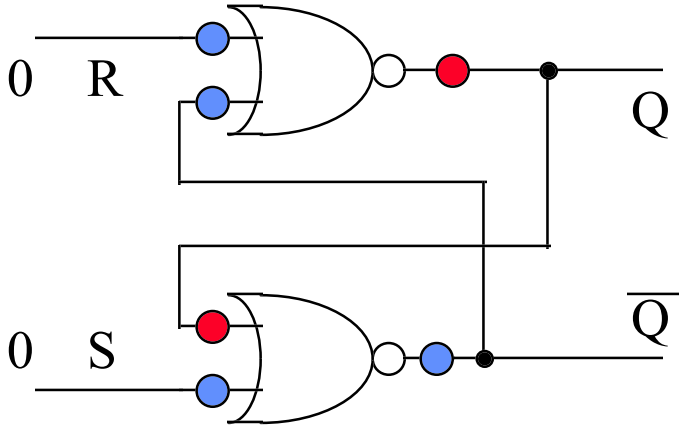
NOR Gate

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0









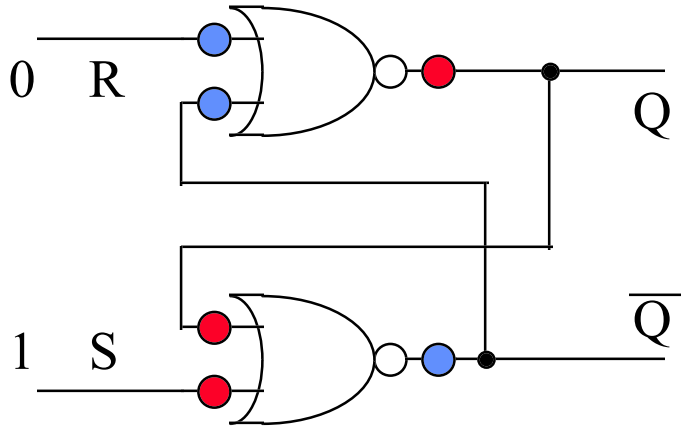
Note a “0” on both R & S results in NO output change.

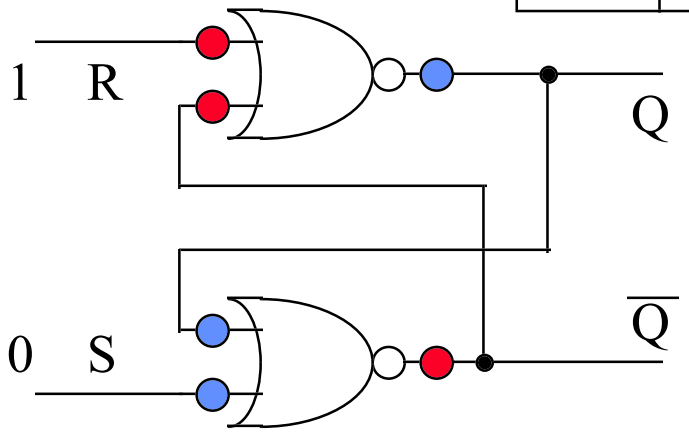
● = 0
● = 1

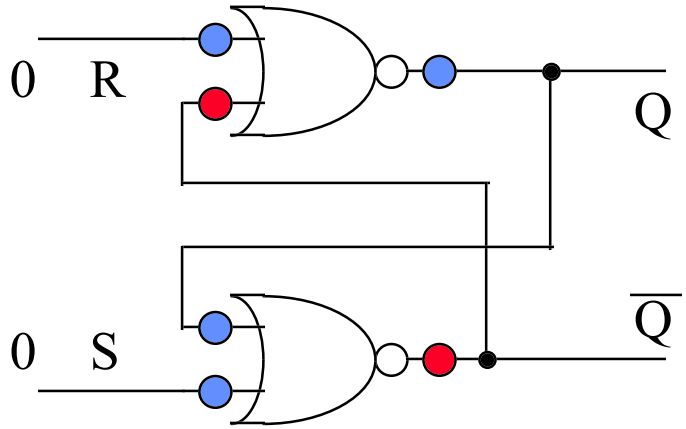
Logic Functions.

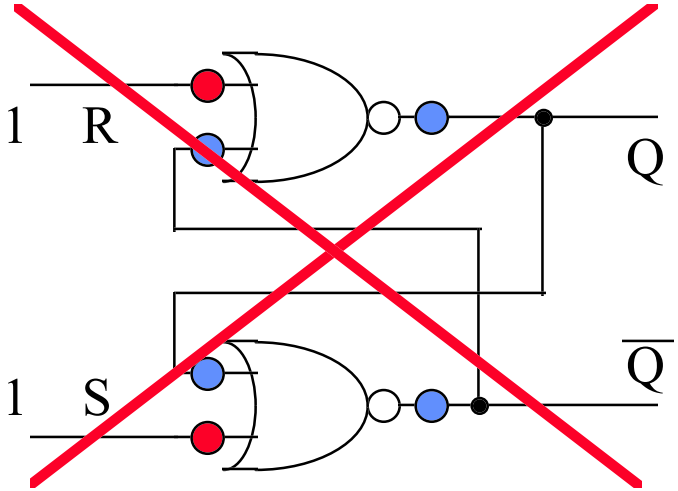
NOR Gate

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0









Logic Functions.

Summary.

- NOR Logic RS Latch Truth Table.

R	S	Q	$\overline{Q}$
0	0	Q_0	$\overline{Q_0}$
0	1	1	0
1	0	0	1
1	1	X	X

X indicates this condition is **NOT** allowed

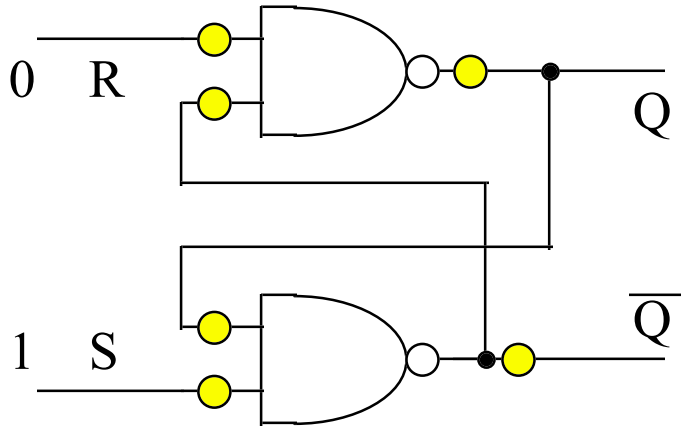
Q_0 indicates the state Q was in before Change

● = 0
● = 1

Logic Functions.

NAND Gate

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0



Practice Page

Convert the Yellow Dots to the correct Logic Level

R = Reset (Reset to Logic Zero)

S = Set (Set to Logic One)

Q = General Code for an Output

$\overline{Q}$ = Bar Q = Inverse of Q

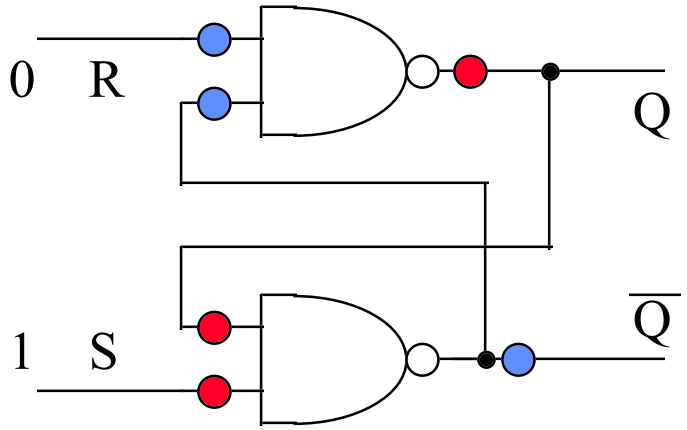
Cross coupled NAND gates to produce a Latch.

● = 0
● = 1

Logic Functions.

NAND Gate

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0



R = Reset (Reset to Logic Zero)

S = Set (Set to Logic One)

Q = General Code for an Output

$\overline{Q}$ = Bar Q = Inverse of Q

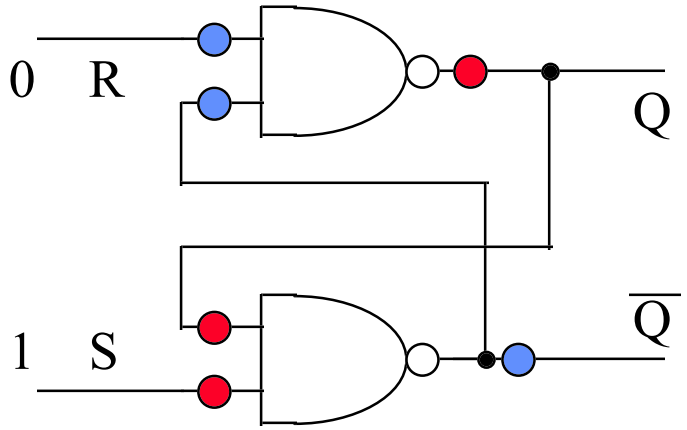
Cross coupled NAND gates to produce a Latch.

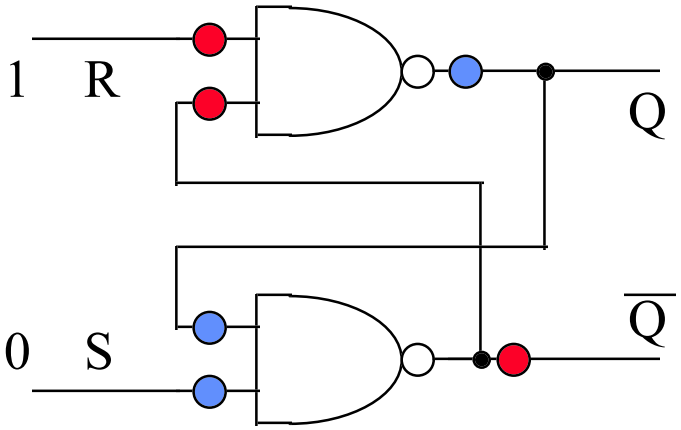
● = 0
● = 1

Logic Functions.

NAND Gate

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0





Cross coupled NAND gates to produce a Latch.

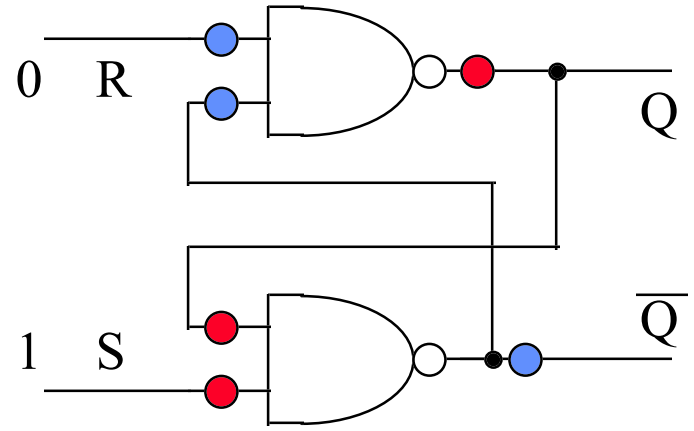
● = 0

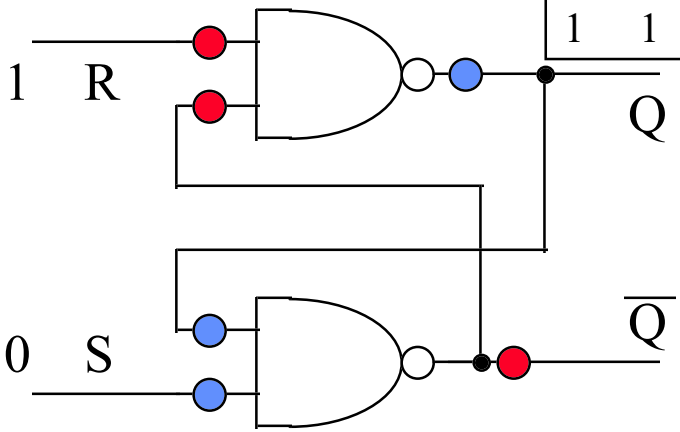
● = 1

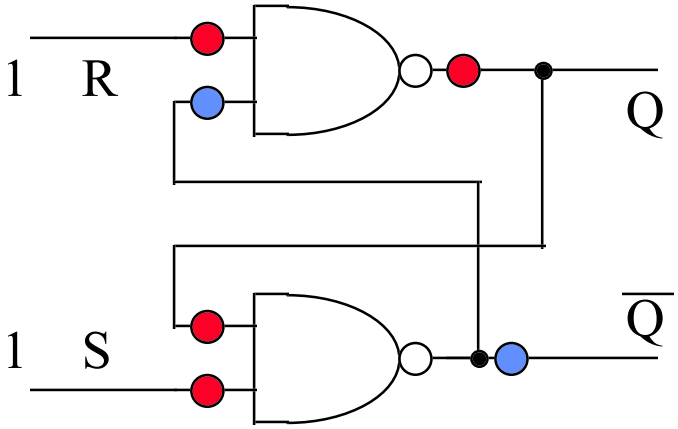
Logic Functions.

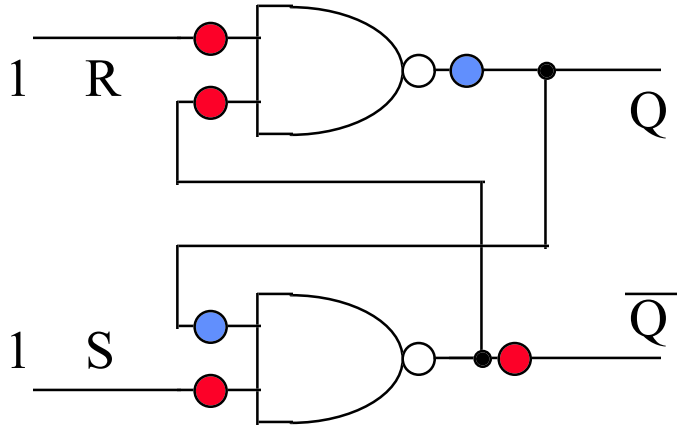
NAND Gate

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0









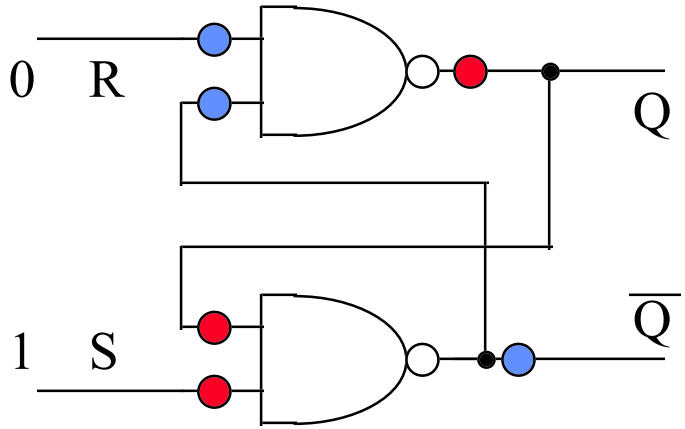
Note a “1” on both R & S results in NO output change.

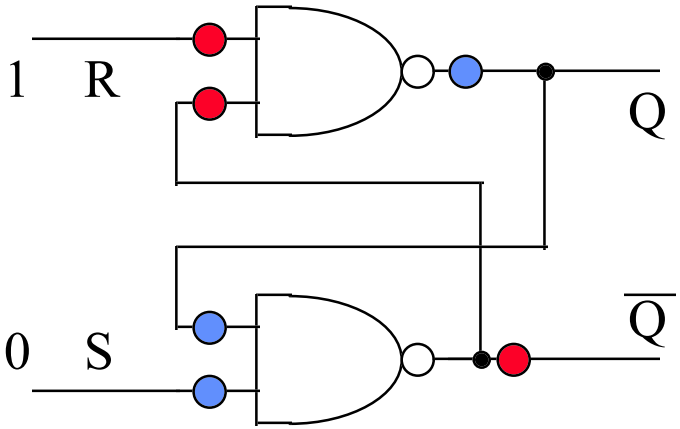
● = 0
● = 1

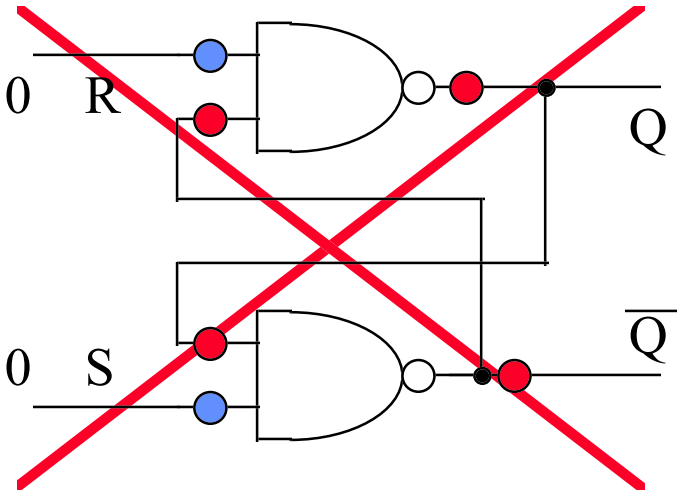
Logic Functions.

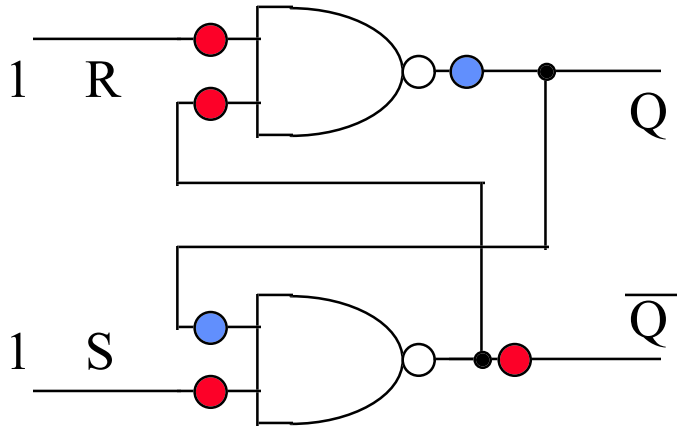
NAND Gate

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0









Logic Functions.

Summary.

- NAND Logic RS Latch Truth Table.

R	S	Q	$\overline{Q}$
0	0	X	X
0	1	1	0
1	0	0	1
1	1	Q_0	$\overline{Q_0}$

X indicates this condition is **NOT** allowed

Q_0 indicates the state Q was in before Change

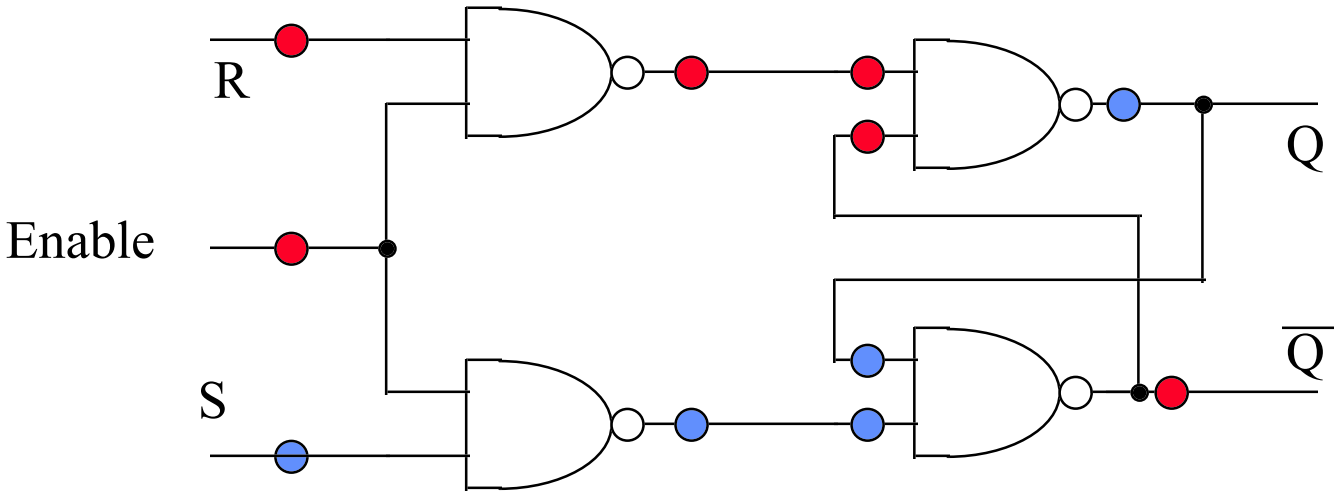
● = 0

● = 1

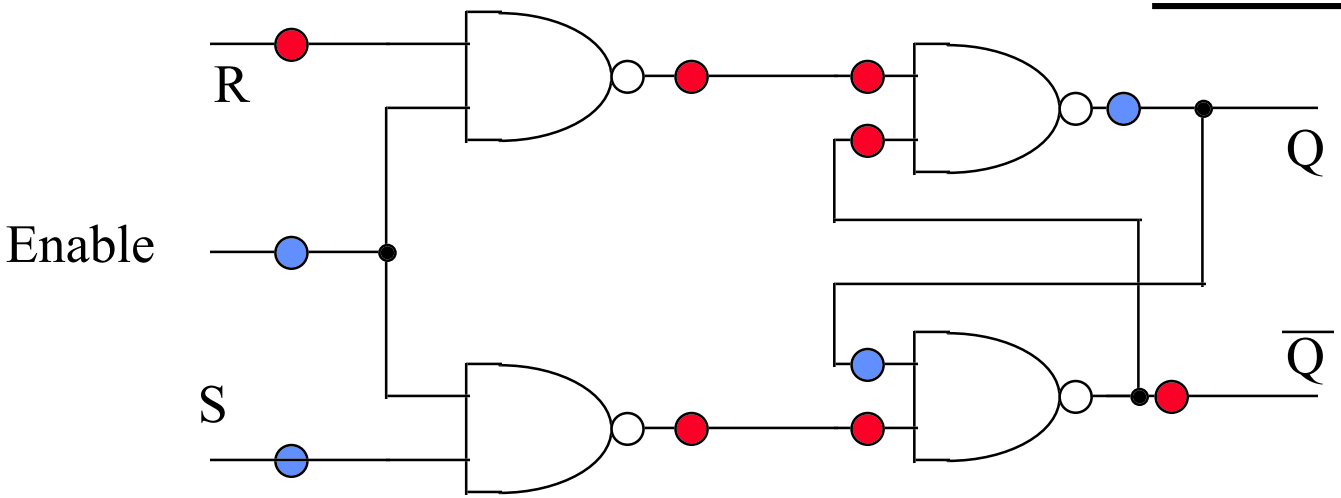
Logic Functions.

NAND Gate

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0



Gated RS Latch



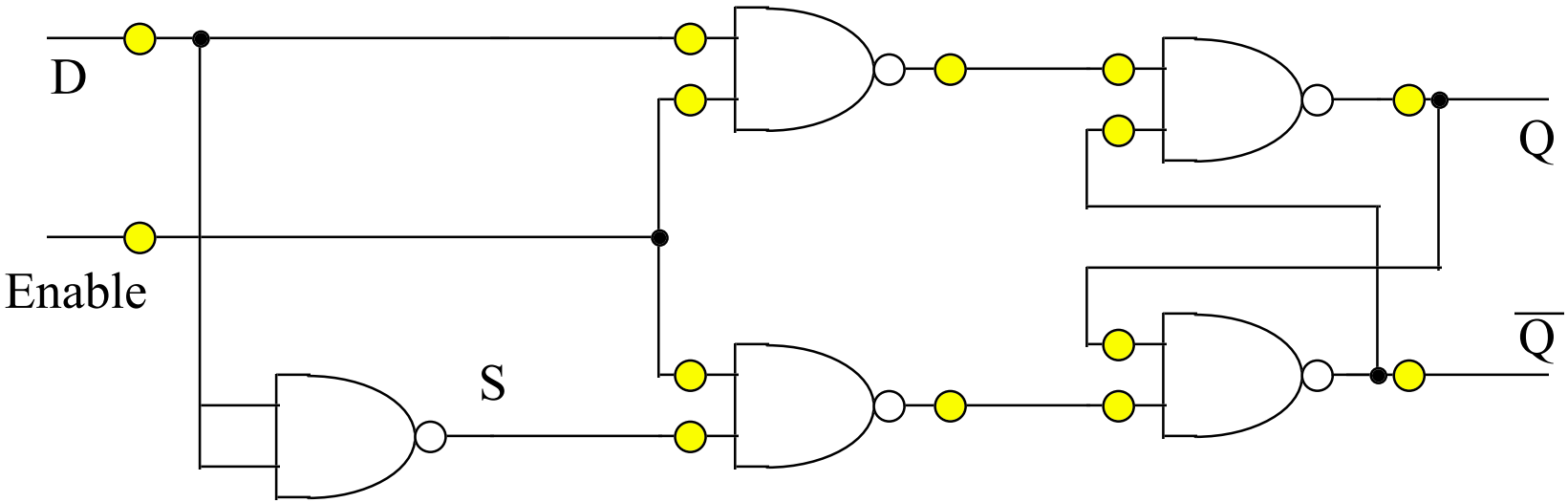
● = 0

● = 1

Logic Functions.

NAND Gate

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0



Practice Page Convert Yellow Dots Perm D and Enable.

D Type Latch

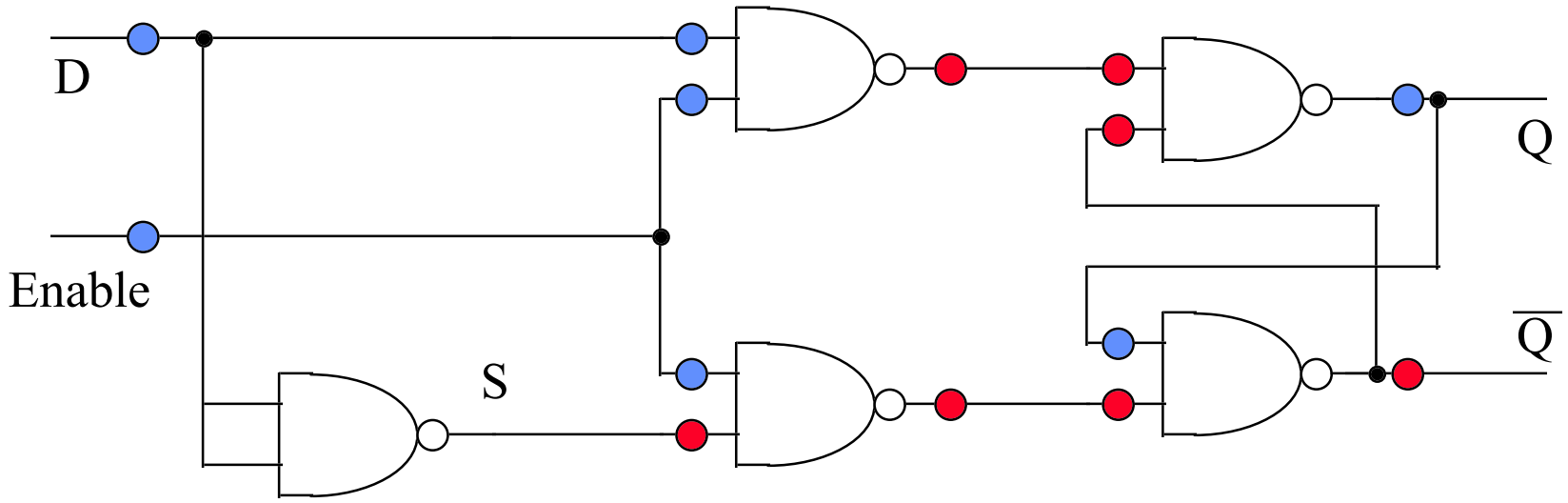
● = 0

● = 1

Logic Functions.

NAND Gate

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0



Enable = 0 , Output Does Not Change

D Type Latch

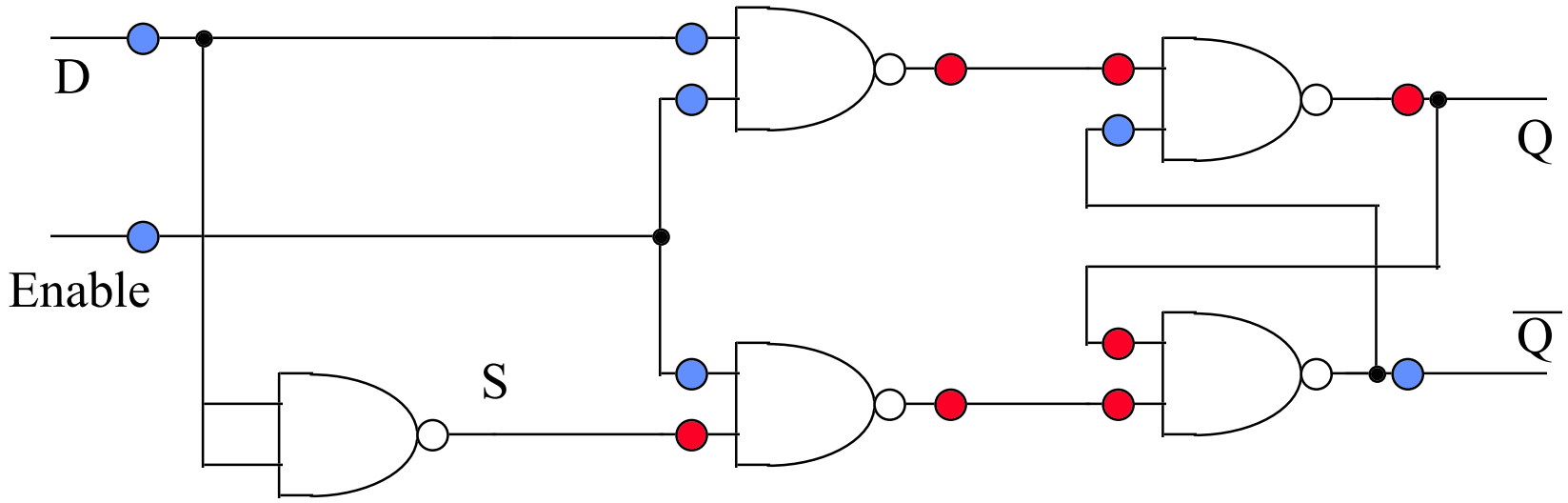
● = 0

● = 1

Logic Functions.

NAND Gate

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0



Enable = 0 , Output Does Not Change

D Type Latch

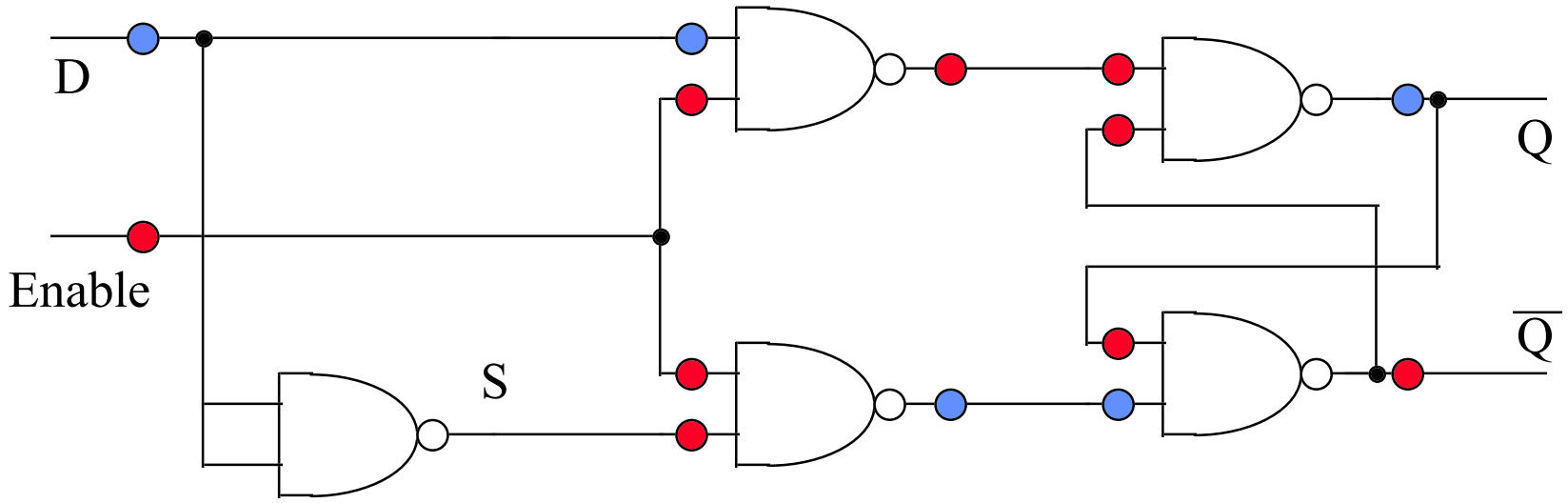
● = 0

● = 1

Logic Functions.

NAND Gate

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0



Enable = 1 , D = 1 Then Q = 1

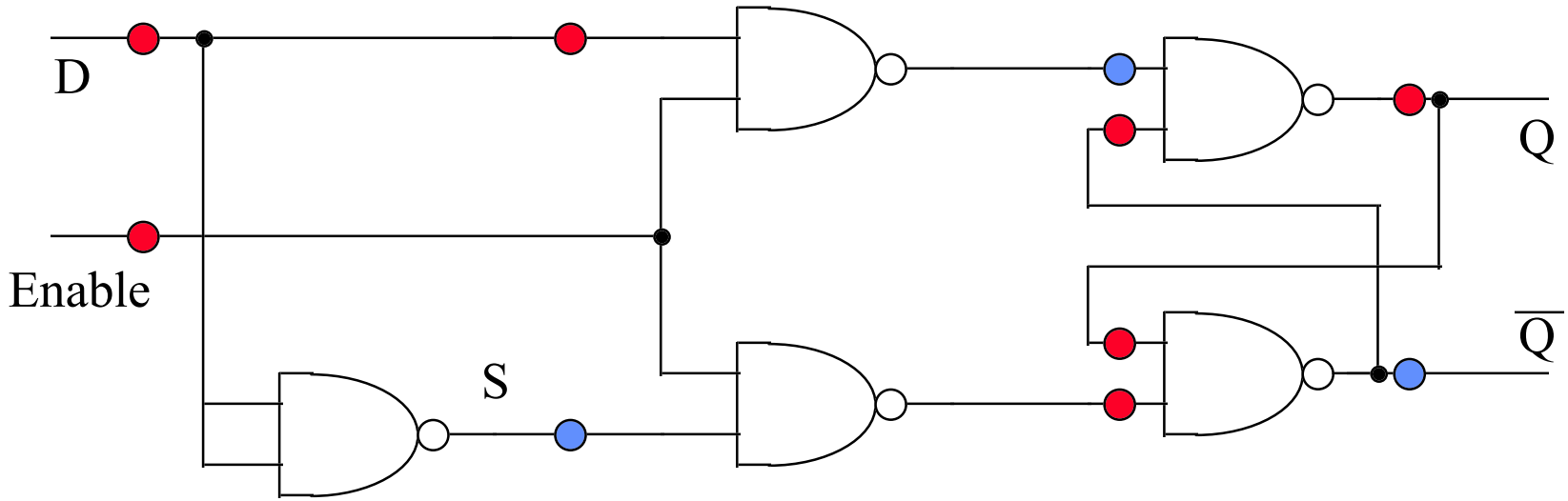
D Type Latch

● = 0
● = 1

Logic Functions.

NAND Gate

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0



Summary

D Type Latch

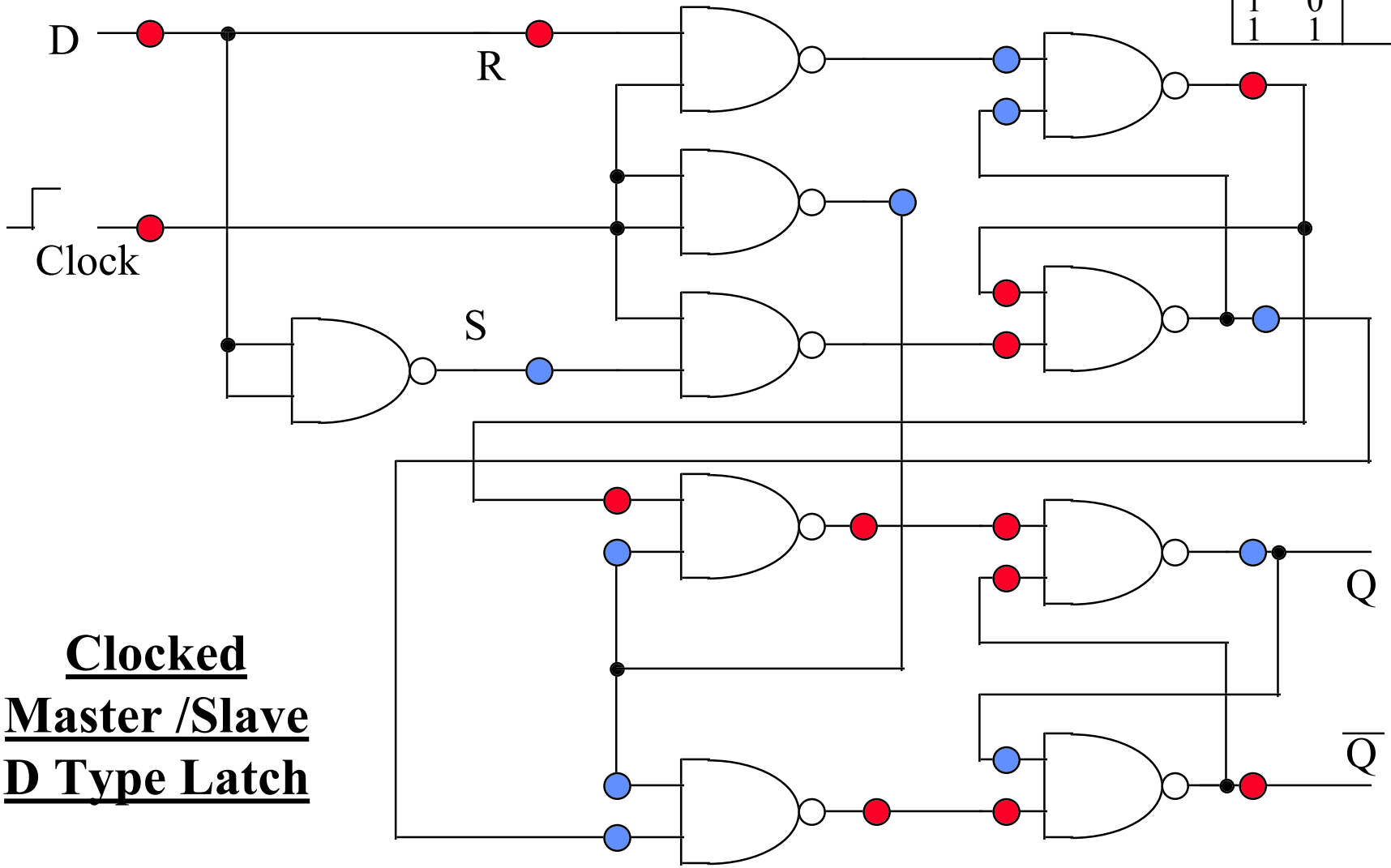
D	Enable	Q
Don't Care	0	No Change
0	1	0
1	1	1

Logic Functions.

● = 0
● = 1

NAND Gate

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0



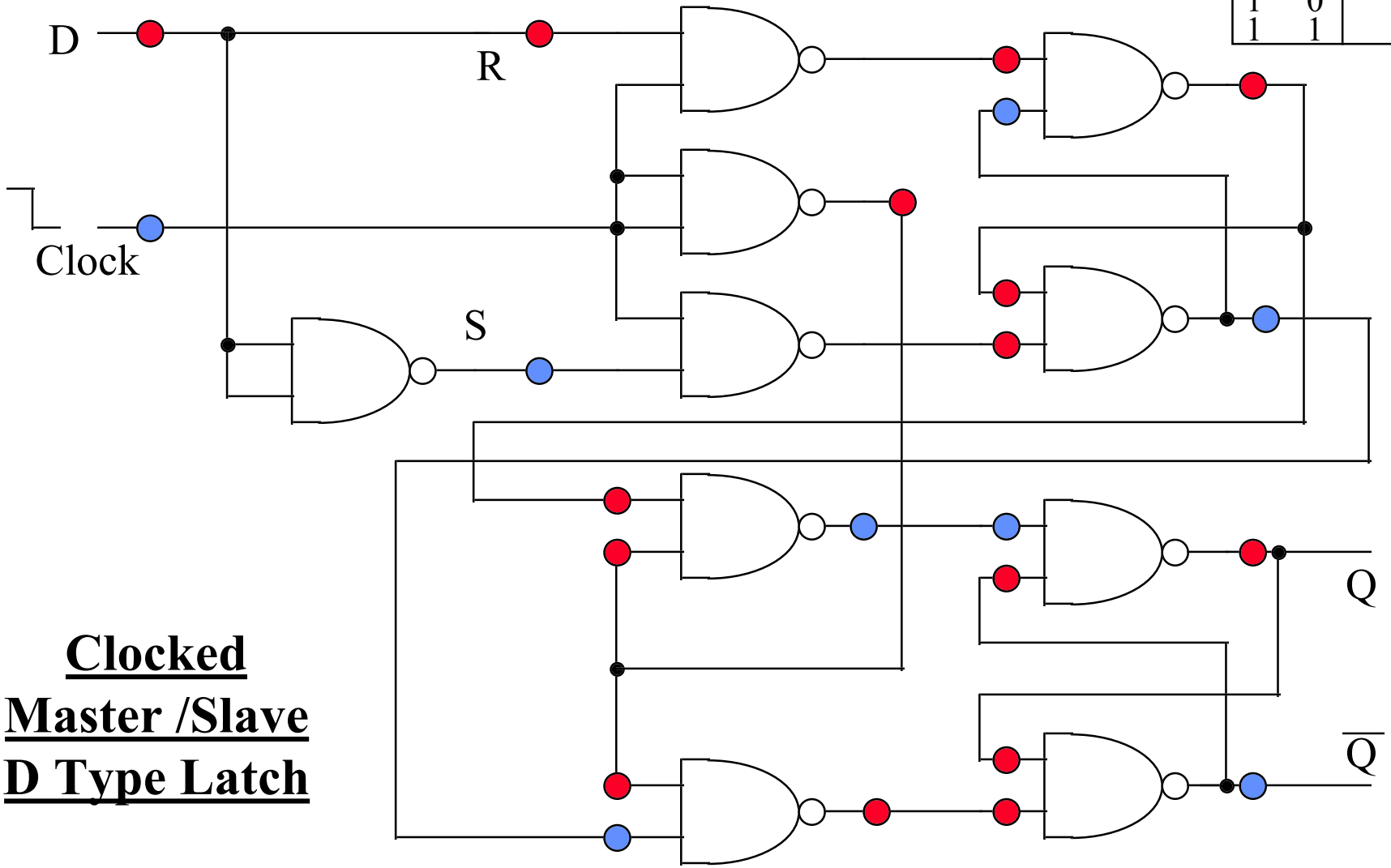
Clocked Master /Slave D Type Latch

Logic Functions.

● = 0
● = 1

NAND Gate

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0



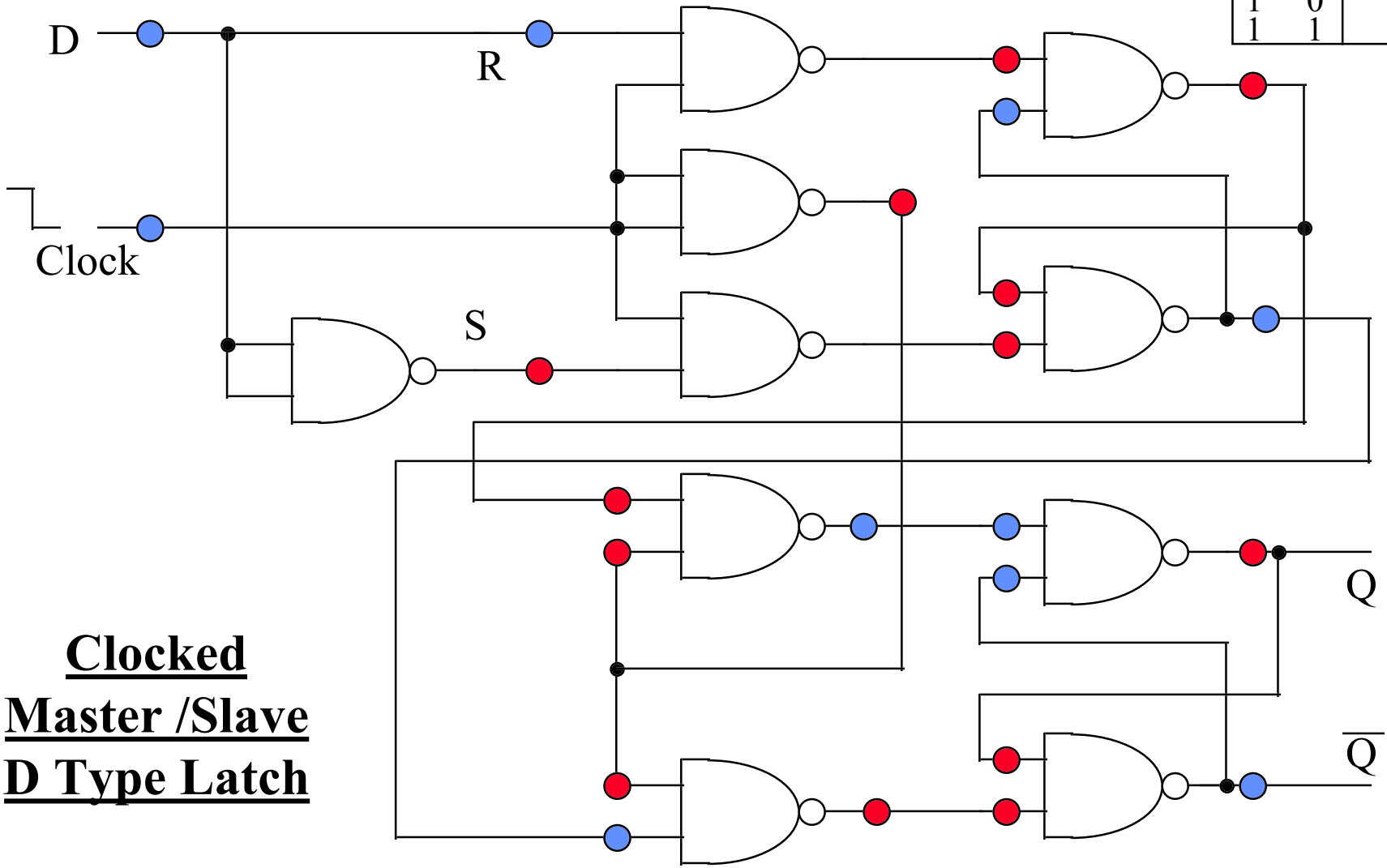
Clocked Master /Slave D Type Latch

Logic Functions.

● = 0
● = 1

NAND Gate

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0



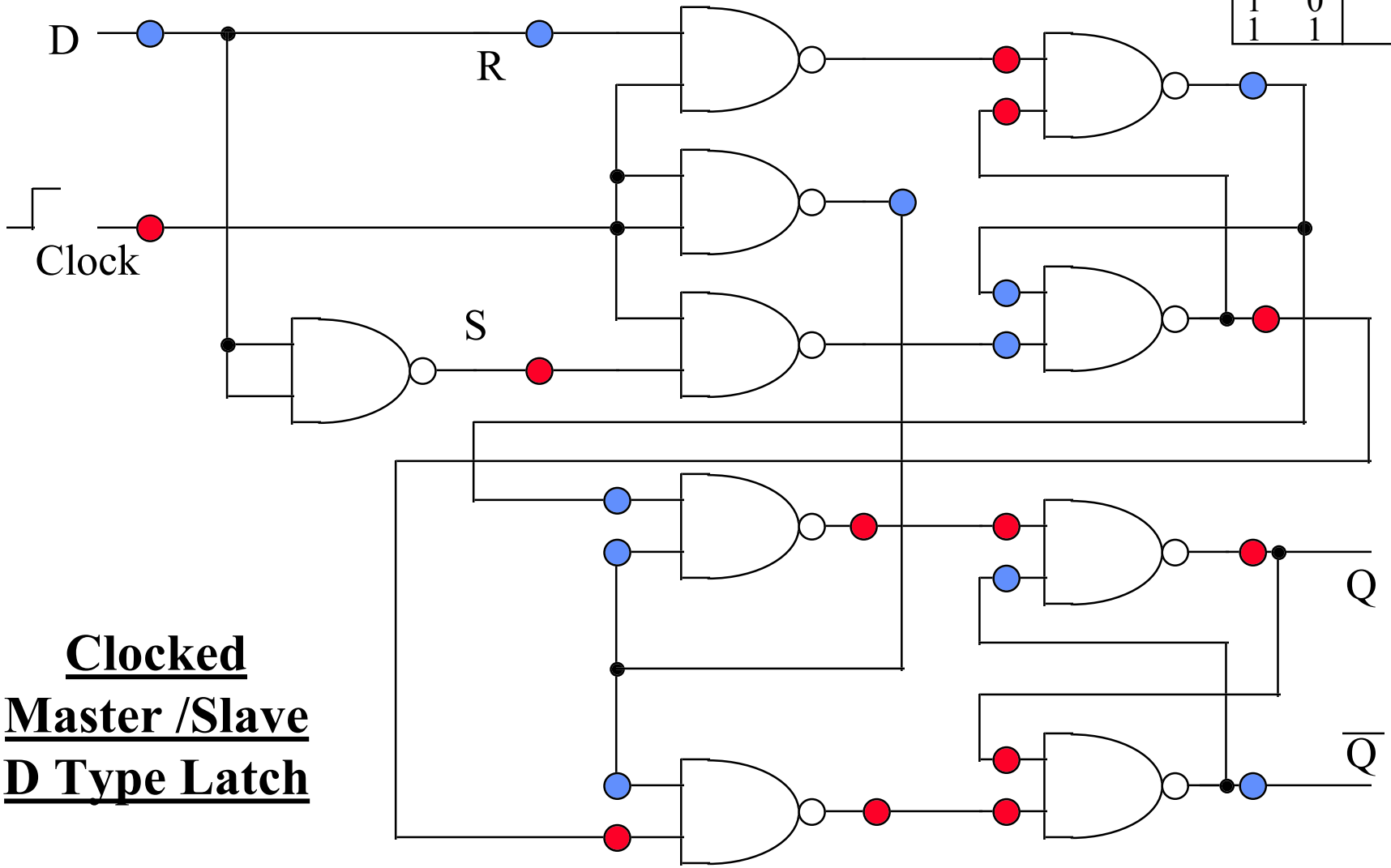
Clocked Master /Slave D Type Latch

Logic Functions.

● = 0
● = 1

NAND Gate

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0



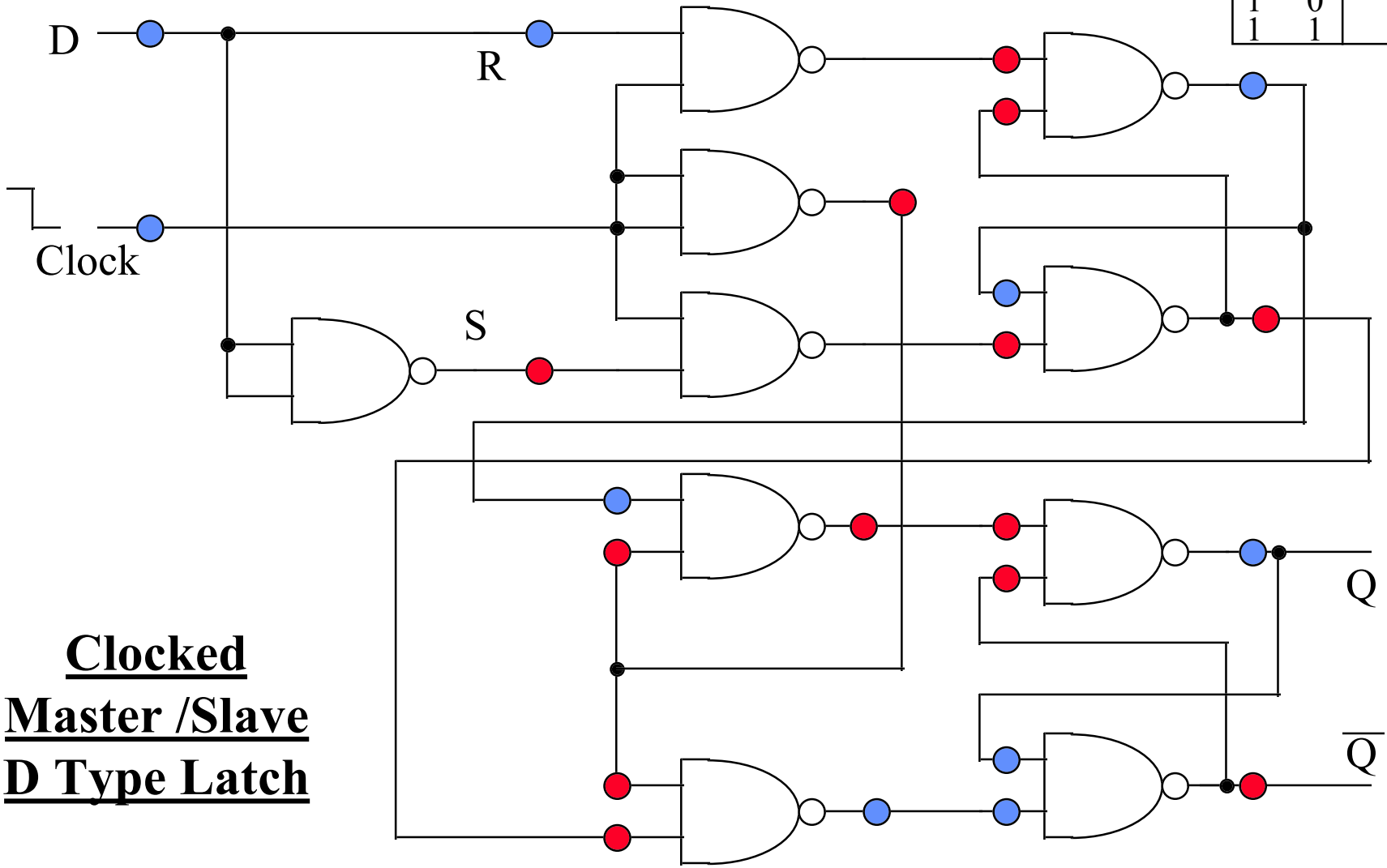
Clocked Master /Slave D Type Latch

Logic Functions.

● = 0
● = 1

NAND Gate

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0



Clocked Master /Slave D Type Latch

Logic Functions.

- The D (Data)Type Latch Truth Table.

D	Clk	Q
0	⌐	0
0	¬	Q_0
1	⌐	1
1	¬	Q_0

Summary.

Positive Edge Triggered D Type

Q_0 indicates the state Q was in before Clock Pulse

Logic Functions.

"D" Type Variant Configurations.

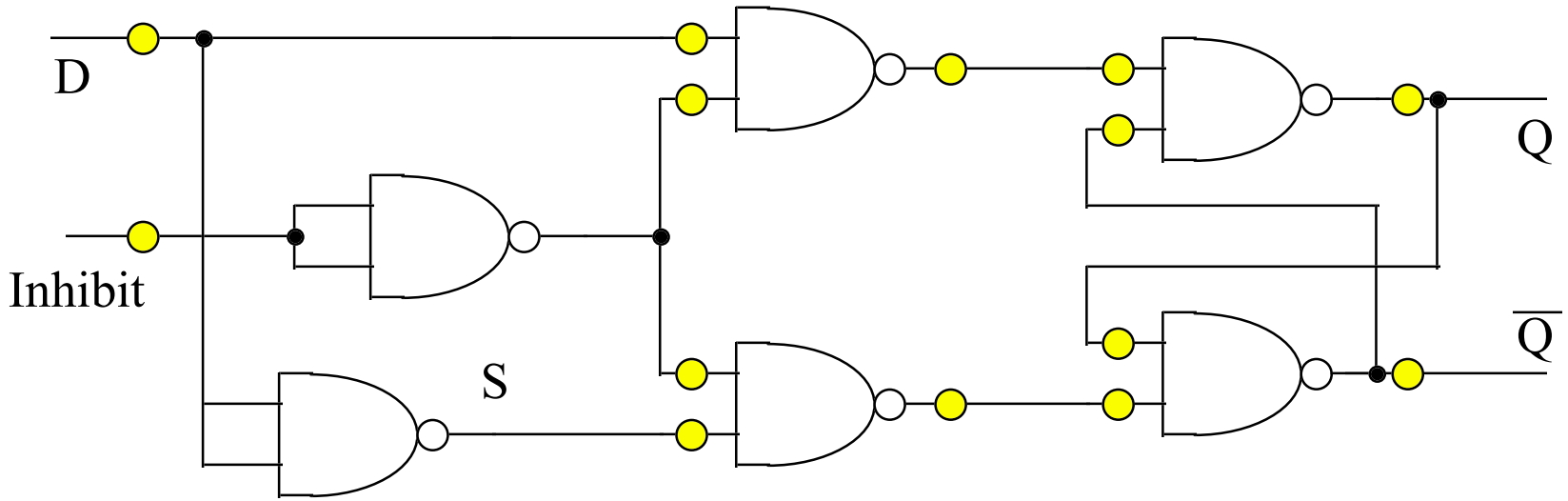
● = 0

● = 1

Logic Functions.

NAND Gate

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0



Practice Page Convert Yellow Dots Perm D and Inhibit.

D Type Latch

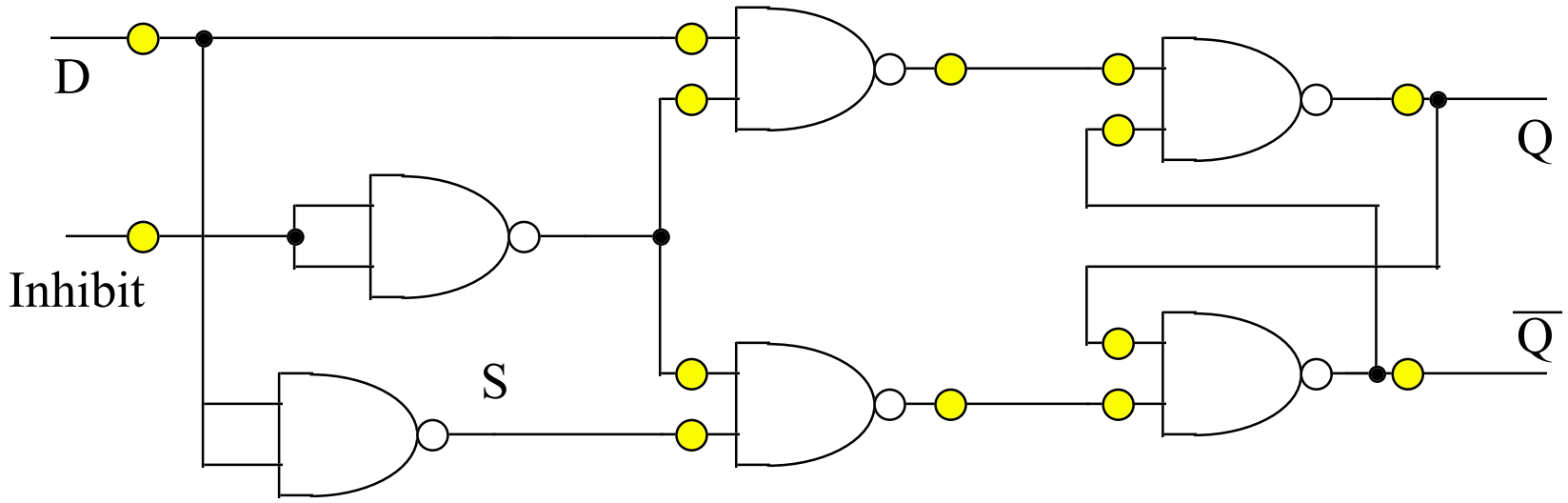
● = 0

● = 1

Logic Functions.

NAND Gate

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0



Summary

D	Inhibit	Q
Don't Care	1	No Change
0	0	0
1	0	1

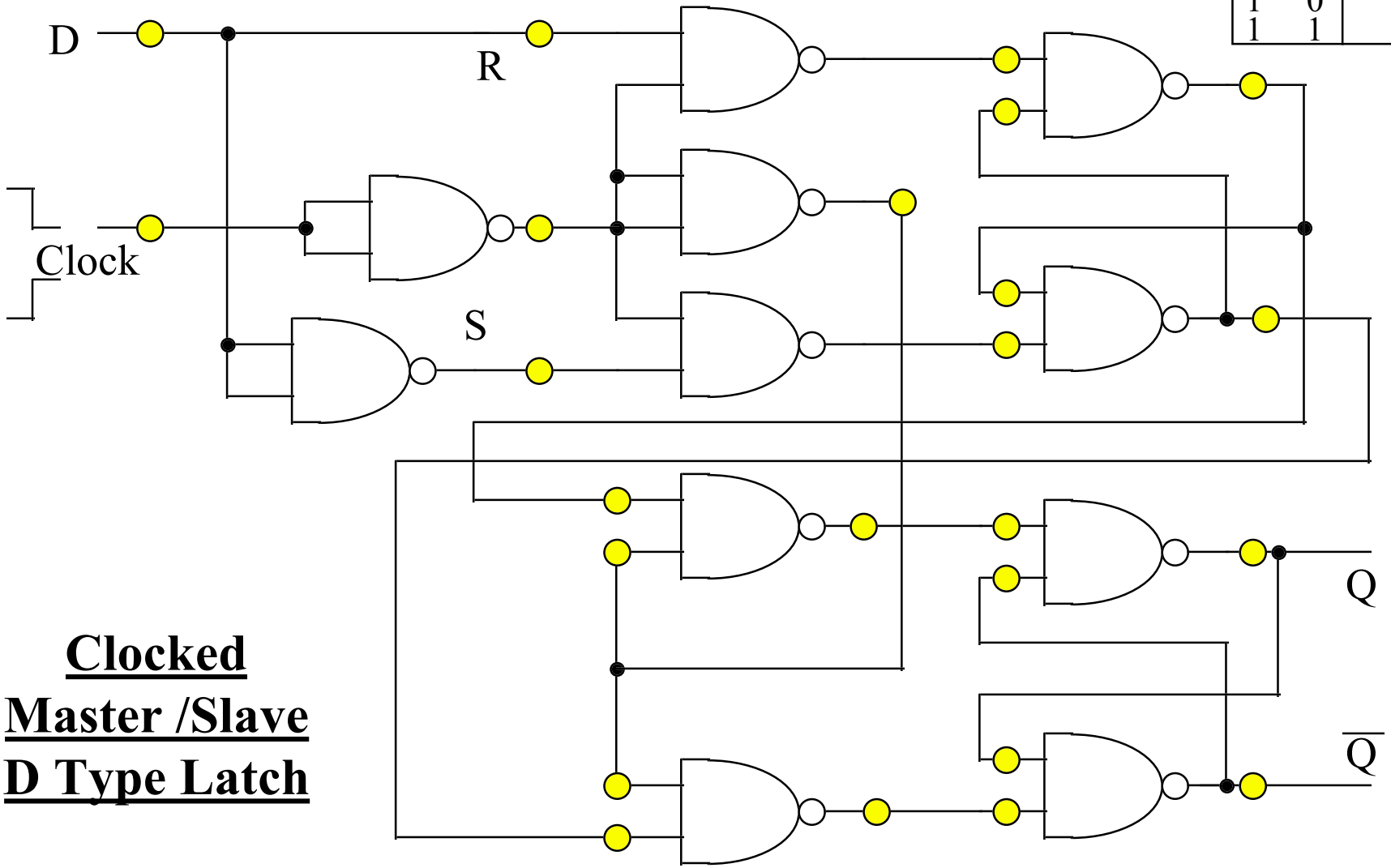
D Type Latch

Logic Functions.

● = 0
● = 1

NAND Gate

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0



Clocked Master /Slave D Type Latch

Practice Page Convert Yellow Dots Perm D and Clock.

Logic Functions.

- The D (Data)Type Latch Truth Table.

D	Clk	Q
0	↓	Q_0
0	↑	0
1	↓	Q_0
1	↑	1

Summary.

Negative Edge Triggered D Type

Q_0 indicates the state Q was in before Clock Pulse

Summary. Logic Functions.

Level Triggered D Type Latches.

D	Enable	Q
Don't Care	0	No Change
0	1	0
1	1	1

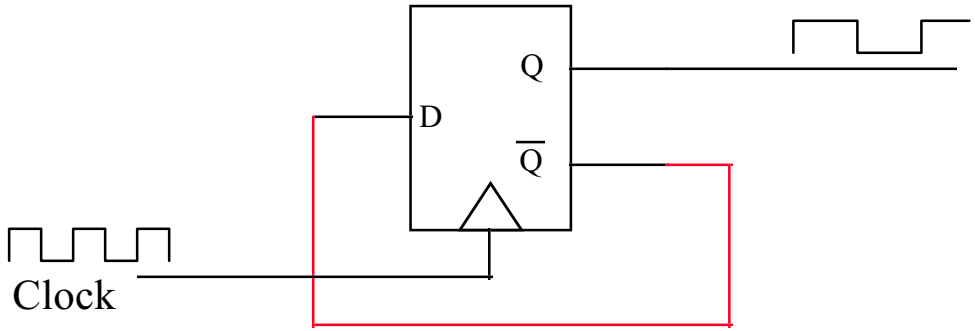
D	Inhibit	Q
Don't Care	1	No Change
0	0	0
1	0	1

Edge Triggered D Type Latches.

D	Positive Clock	Q
Don't Care	⌐.	No Change
0	_⌐.	0
1	_⌐.	1

D	Negative Clock	Q
Don't Care	_⌐.	No Change
0	⌐._	0
1	⌐._	1

Logic Functions.

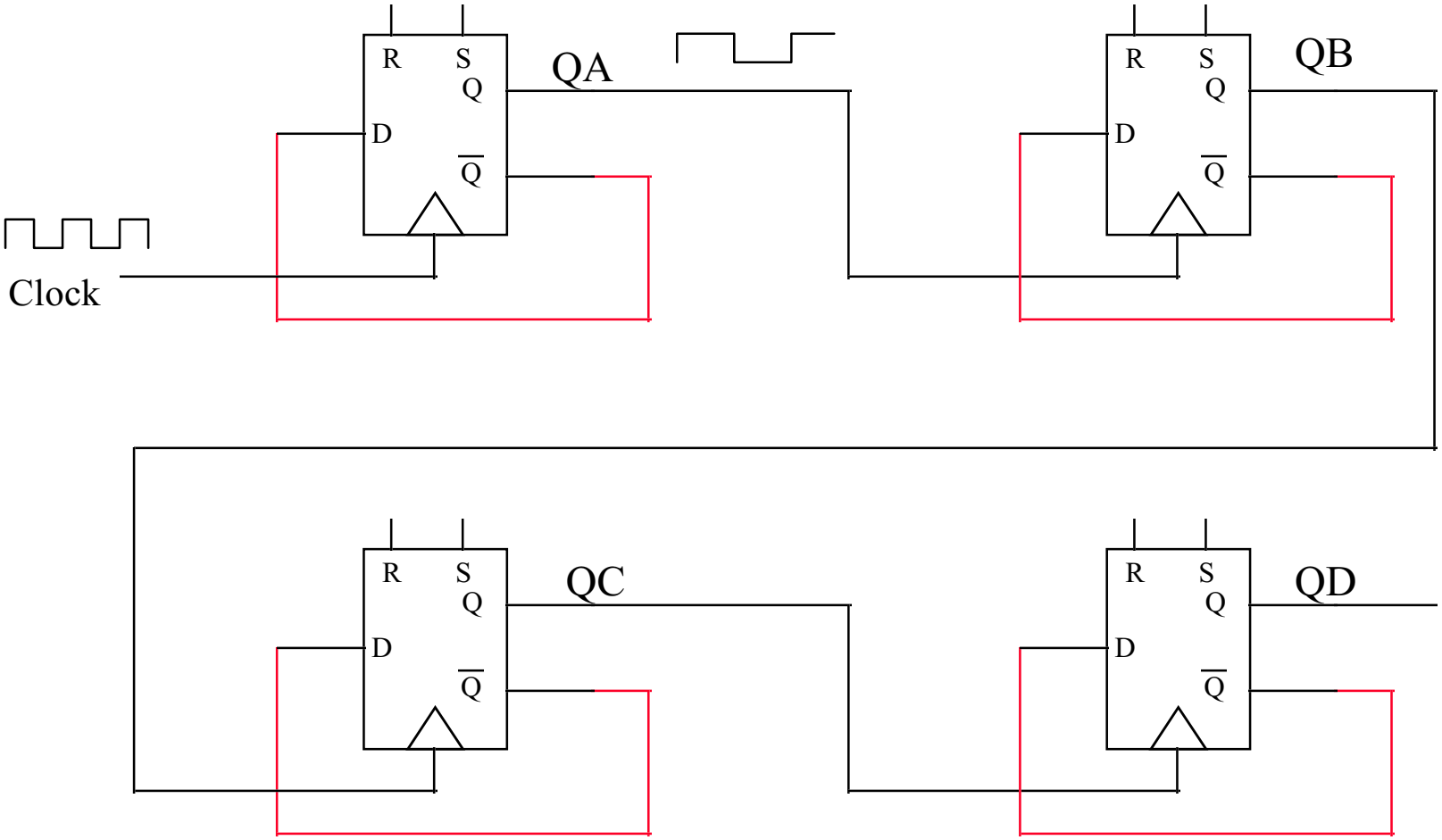


"D" Type configured as a "T" (Toggle) type Flip/Flop

Each Clock Pulse causes the Flip/Flop to change state.

This circuit acts a as divide by two circuit.

Logic Functions.



D Type Flip/Flops + RS as a Divide by 16 Divider.

Logic Functions.

? ? ? QUESTION ? ? ?

- Will the Output of a counter be effected if we use Positive or Negative edge clocked "D" Type Latches ?

Logic Functions.

First let us consider what will happen when we use a Positive edge clock

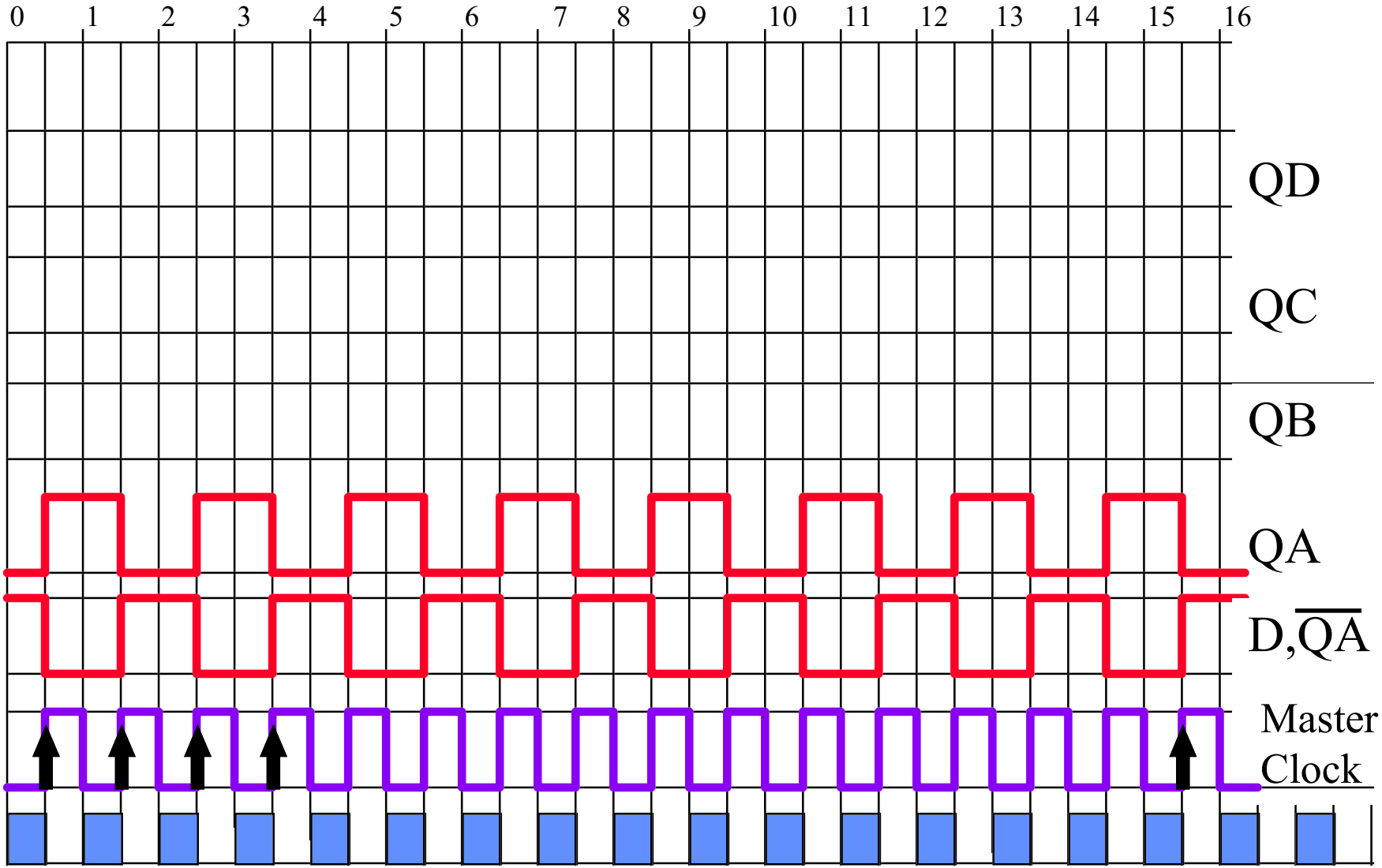
"D" Type device Counter.

CMOS Devices Positive Edge

Clock Edge

Logic Functions.

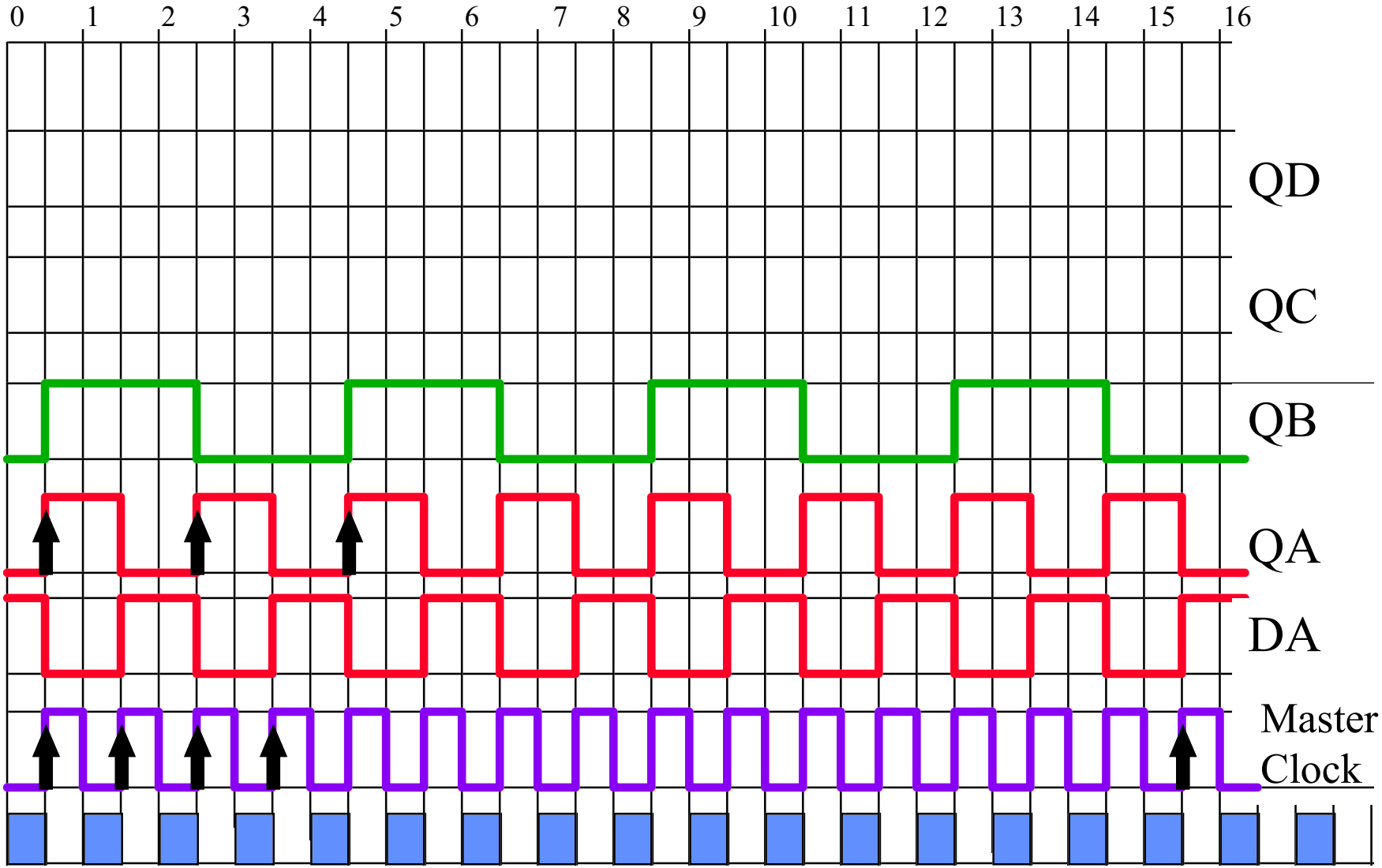
Using Positive Clock Edges



Clock Edge

Logic Functions.

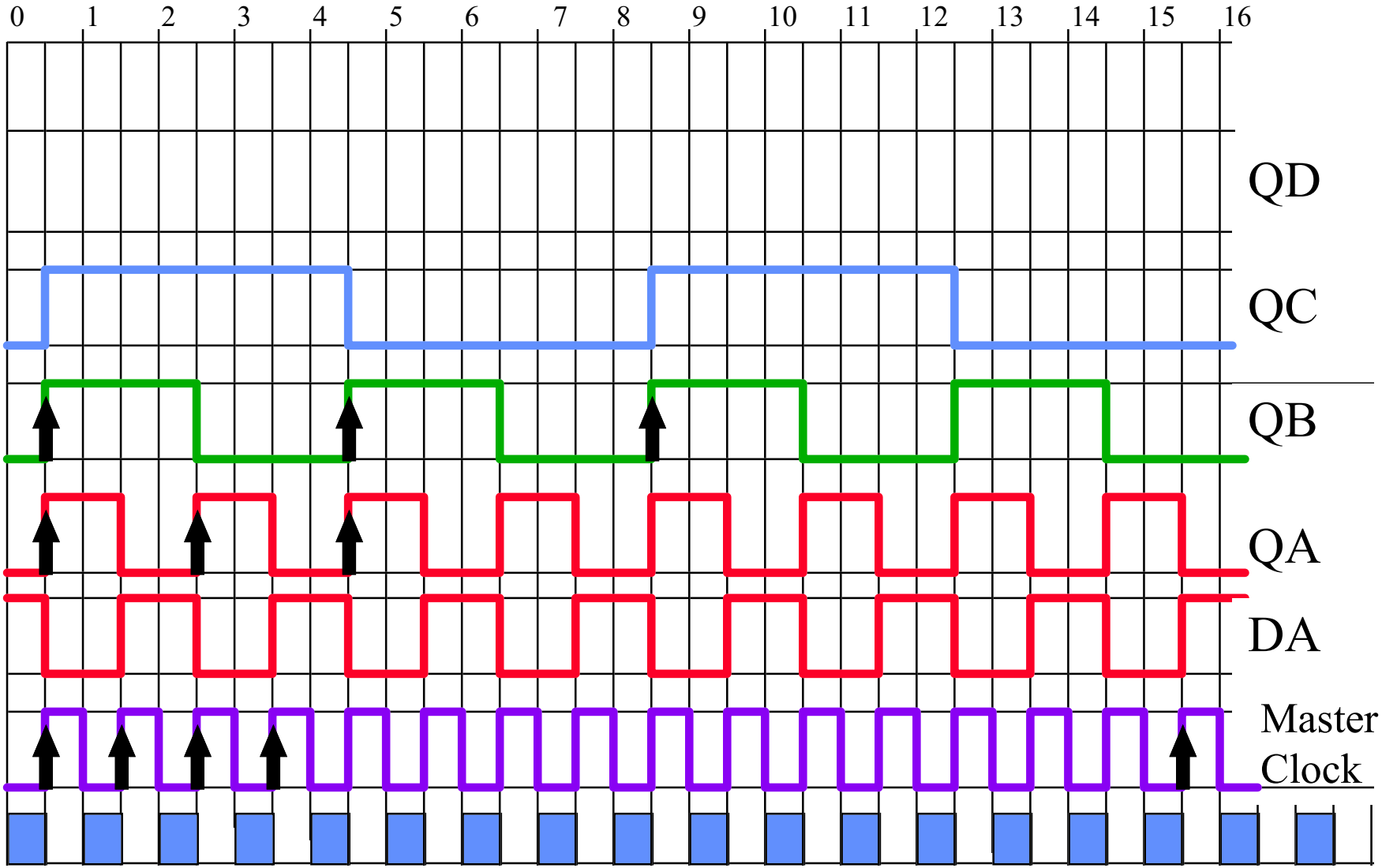
Using Positive Clock Edges



Clock Edge

Logic Functions.

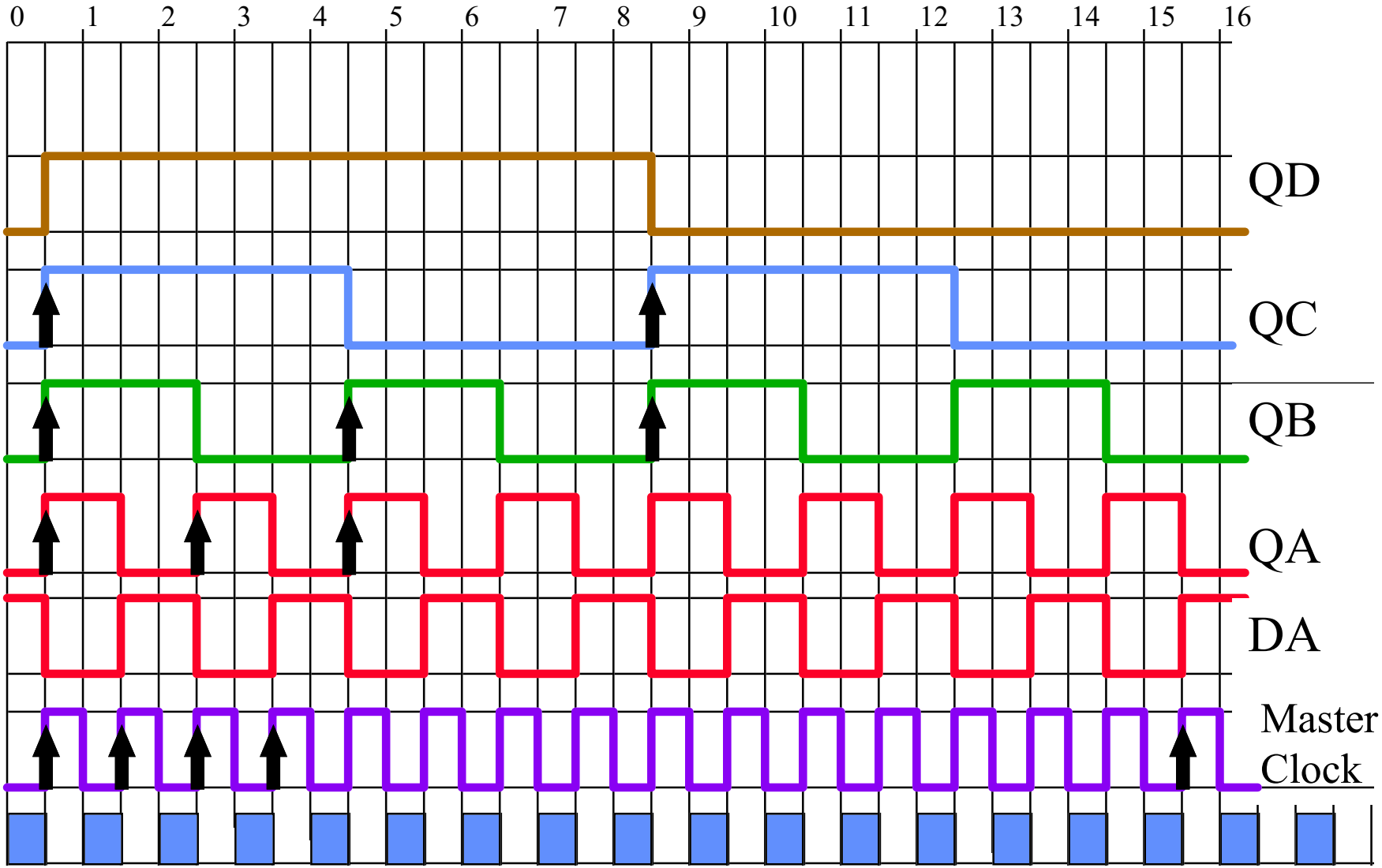
Using Positive Clock Edges



Clock Edge

Logic Functions.

Using Positive Clock Edges



Logic Functions.

Now let us consider what will happen when we use a Negative edge clock

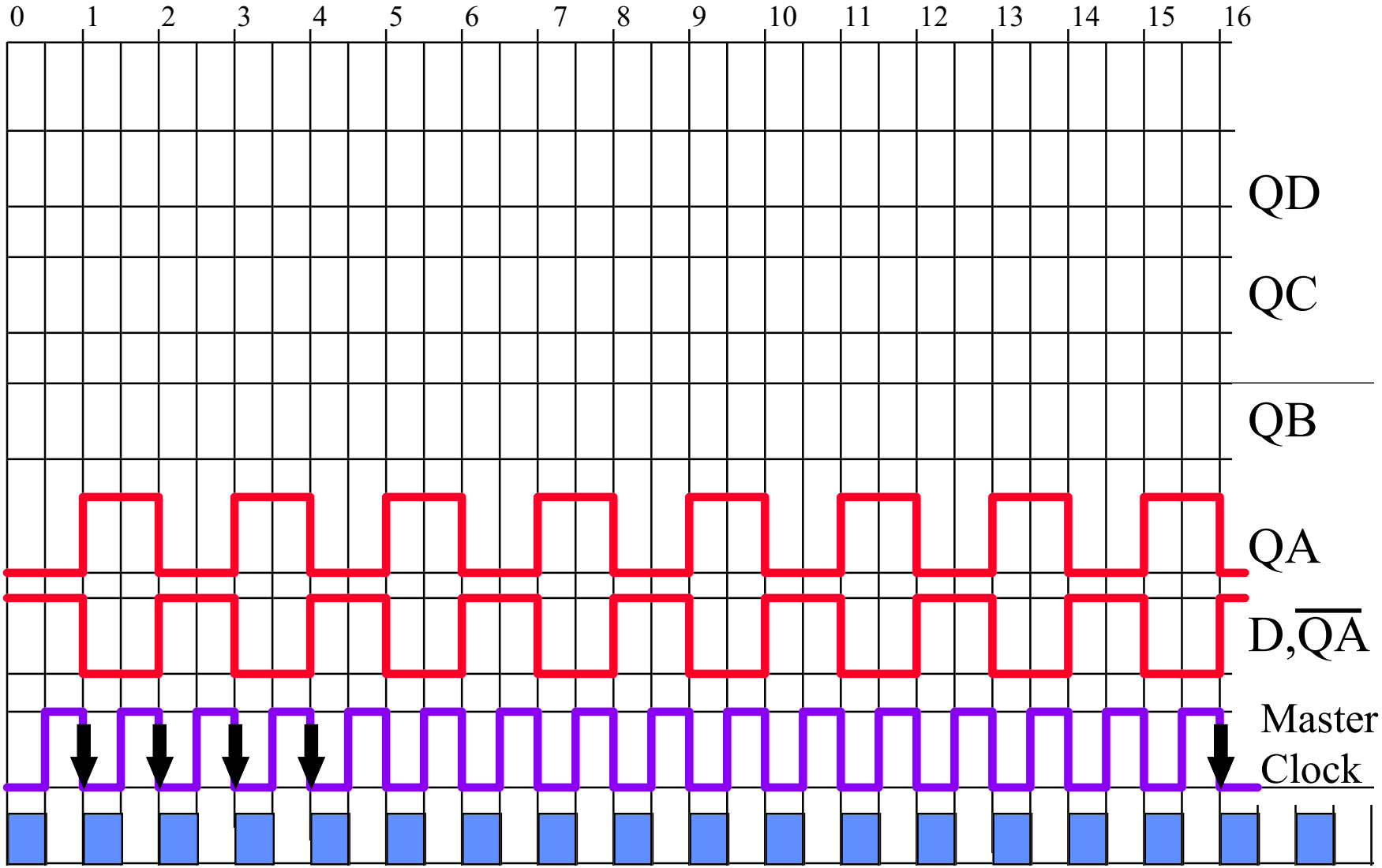
"D" Type device Counter.

TTL Devices Negative Edge

Clock Edge

Logic Functions.

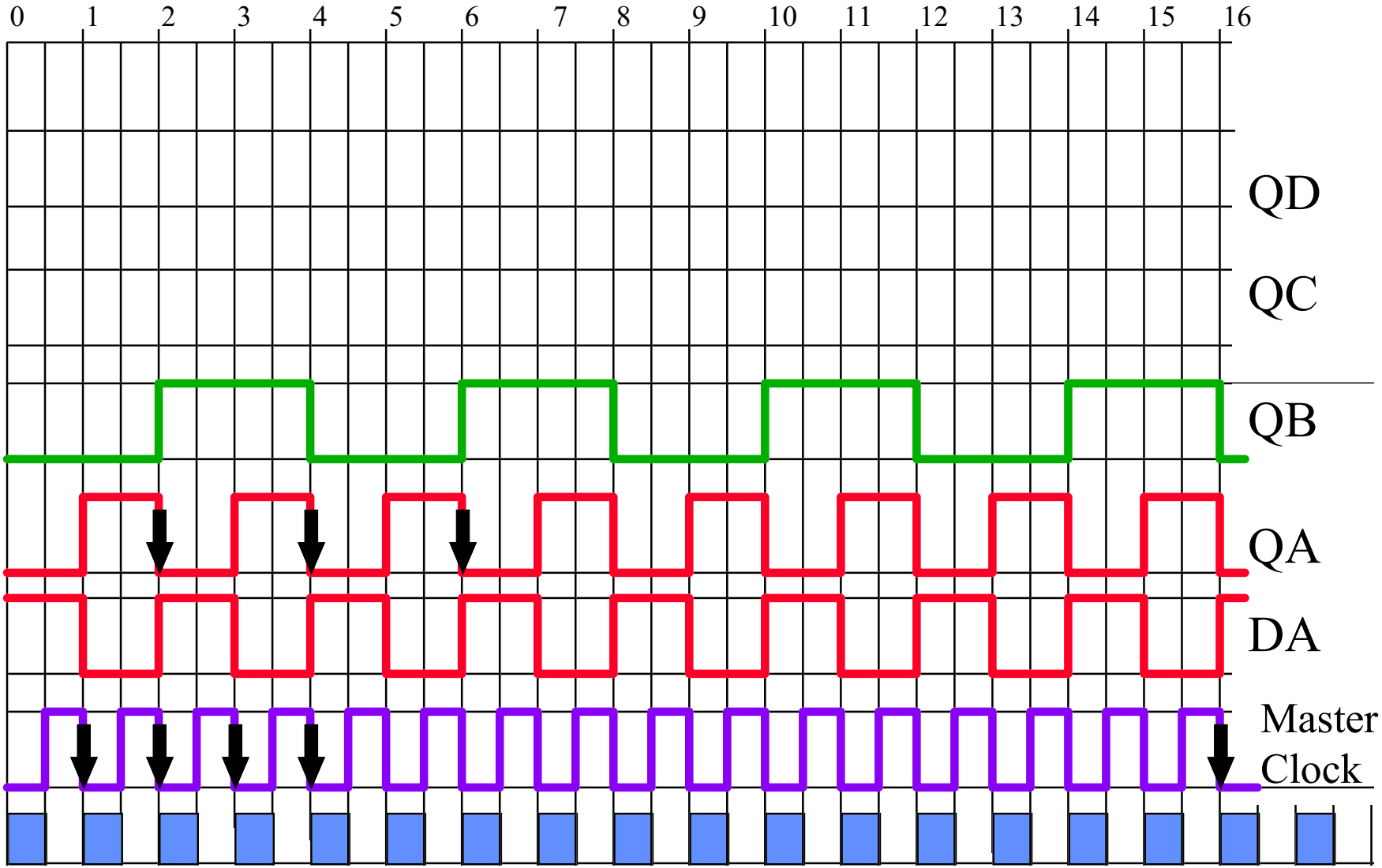
Using Negative Clock Edges



Clock Edge

Logic Functions.

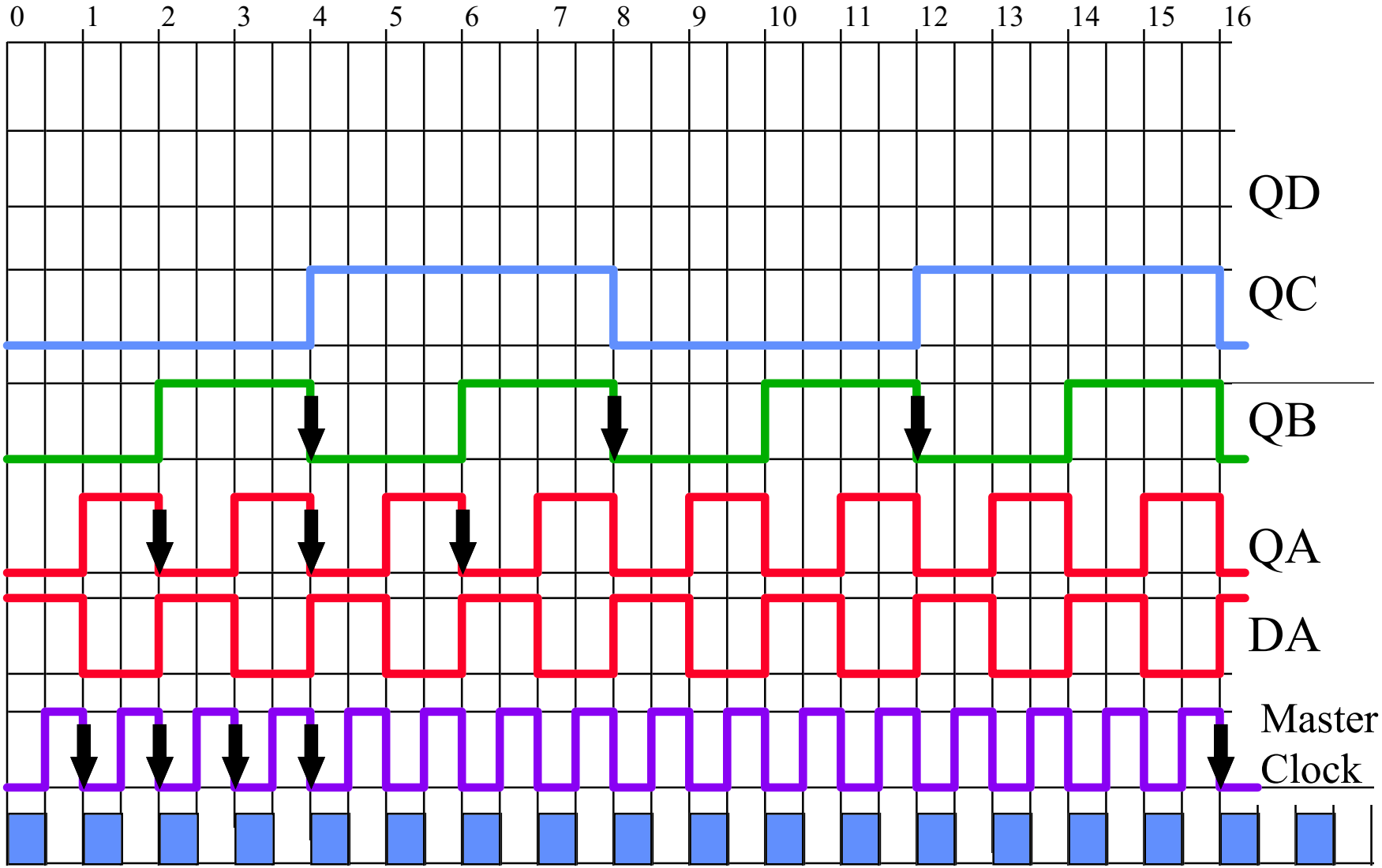
Using Negative Clock Edges



Clock Edge

Logic Functions.

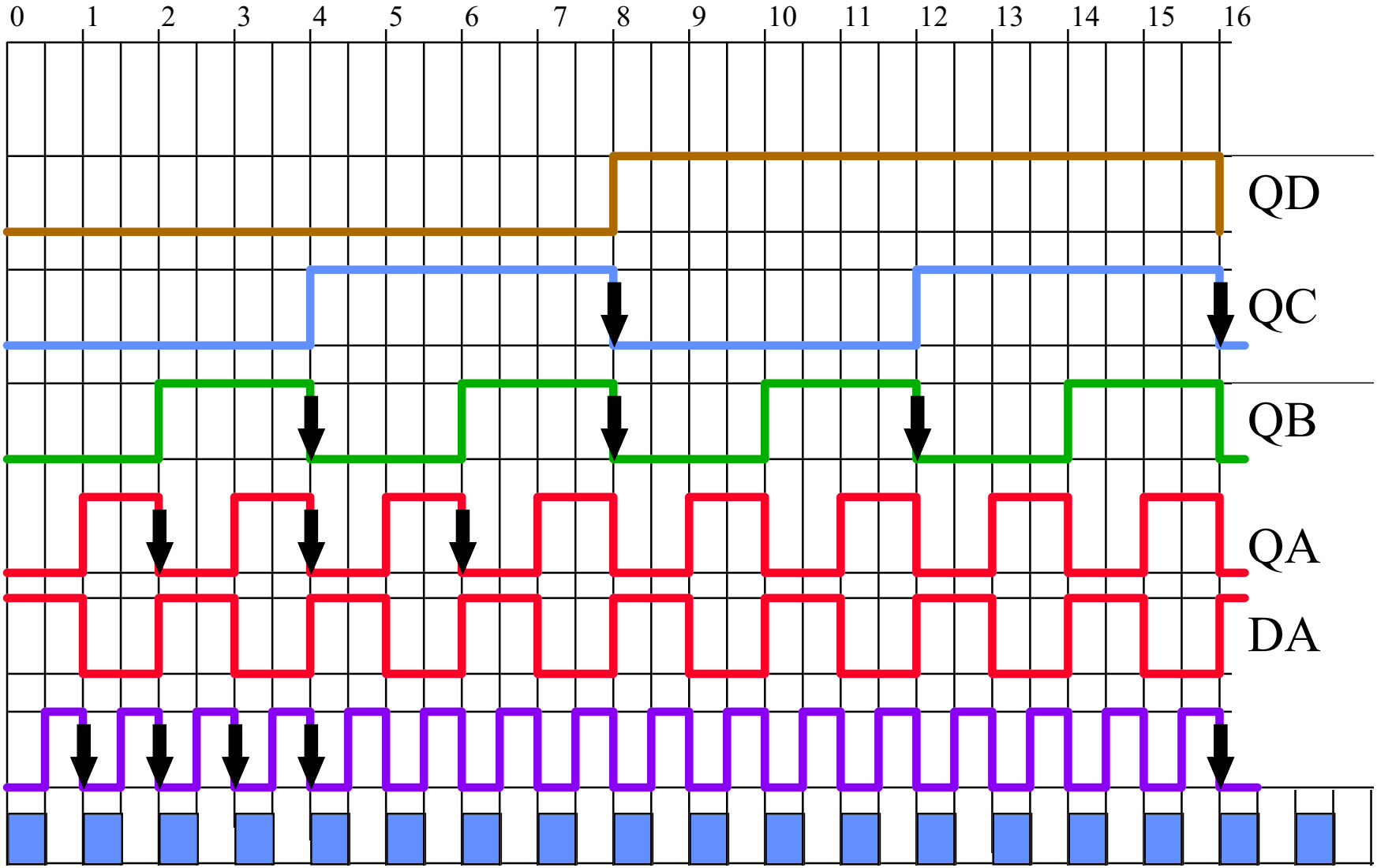
Using Negative Clock Edges



Clock Edge

Logic Functions.

Using Negative Clock Edges

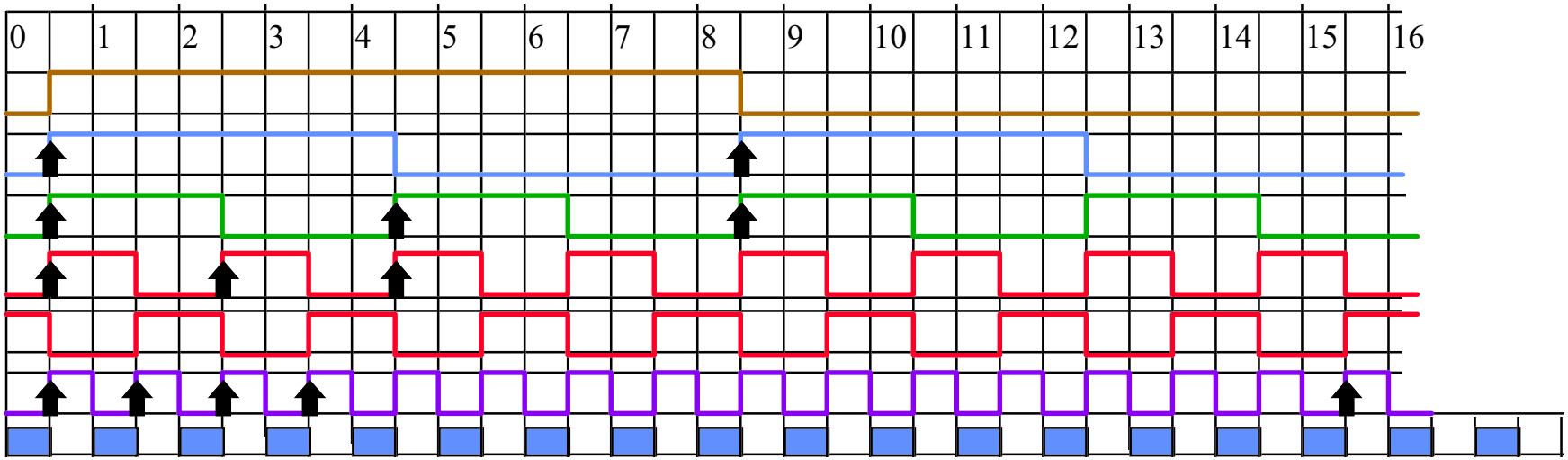


Logic Functions.

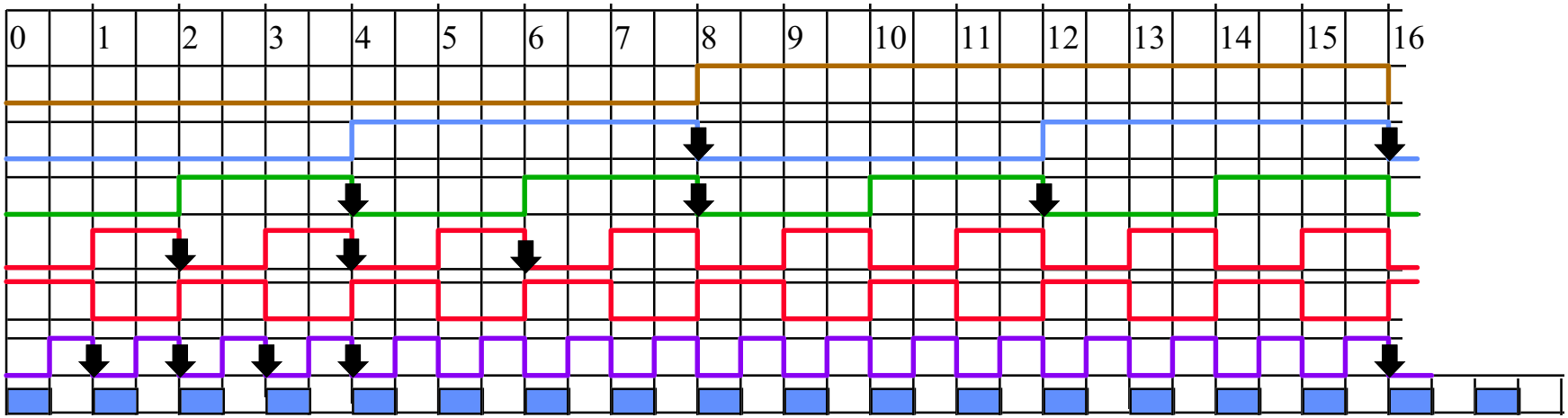
Now let us compare the outputs from a Positive and a Negative edge clocked

“D” Type device Counter.

Logic Functions.



Positive Clocked D Types.



Negative Clocked D Types.

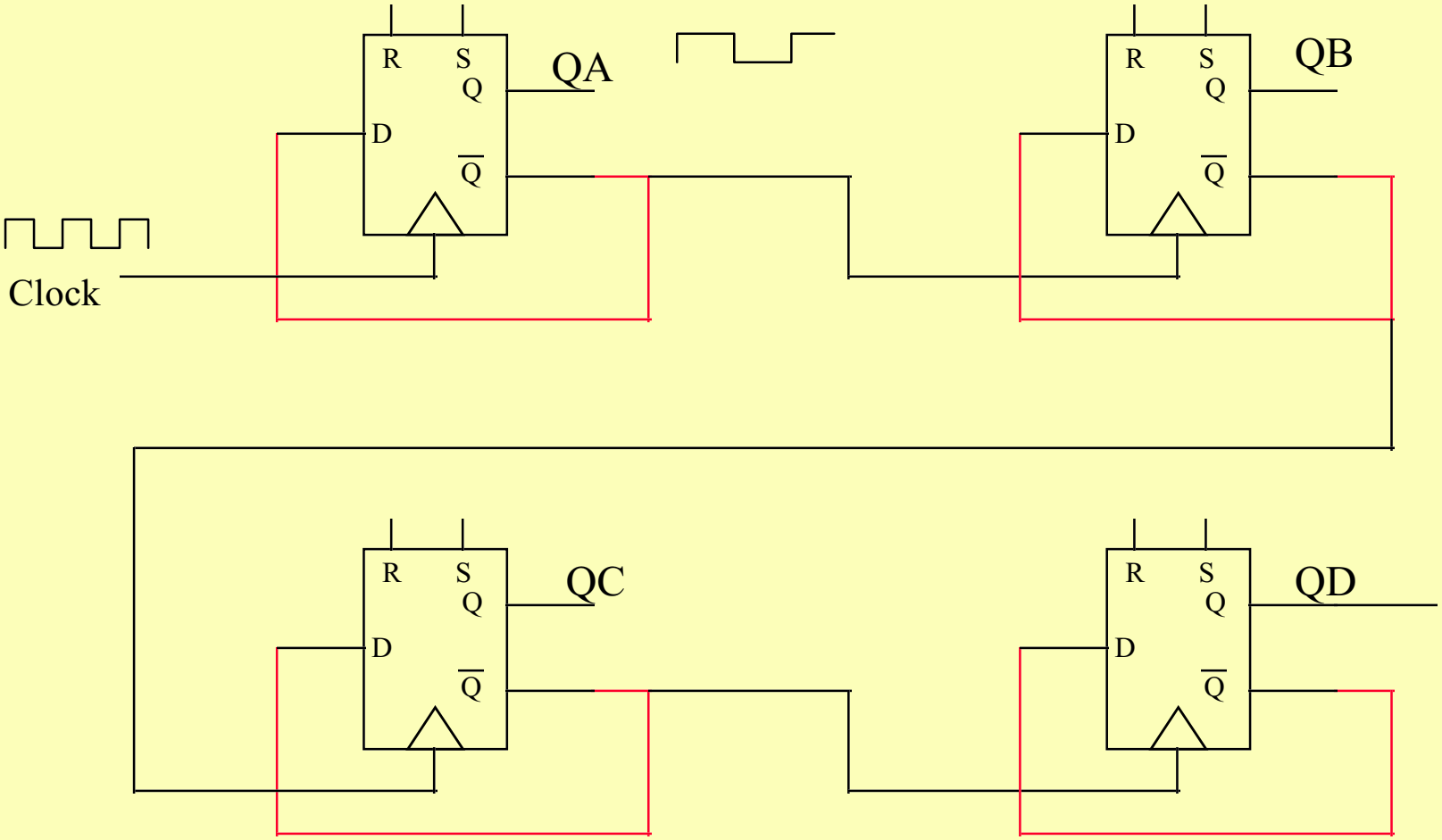
Logic Functions.

An Alternative Circuit using a Positive edge clock

"D" Type device Counter.

CMOS Devices Positive Edge

Logic Functions.



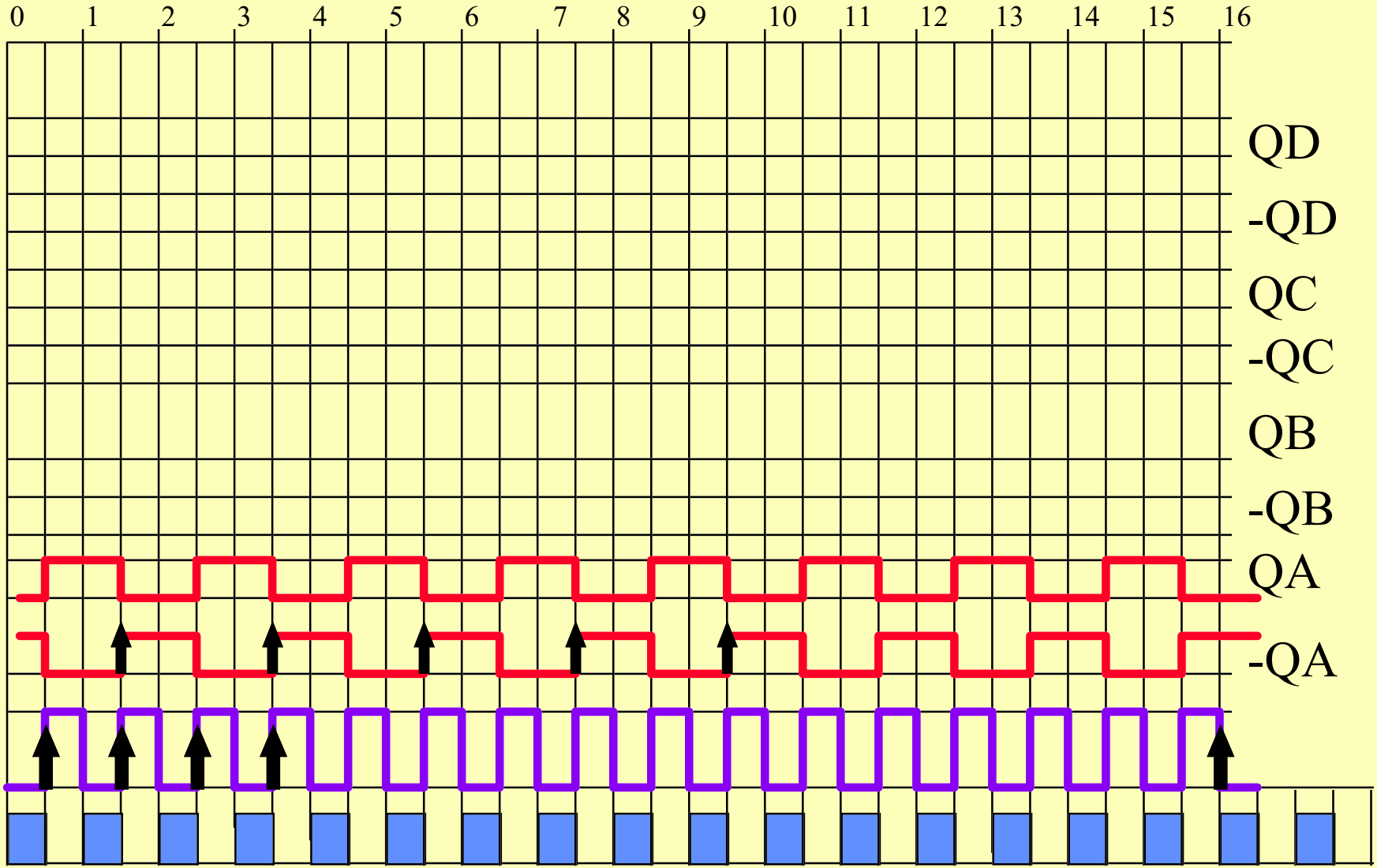
Note this time the clock derived from the NOT Q Output.

D Type Flip/Flops + RS as a Divide by 16 Divider.

Clock Edge

Logic Functions.

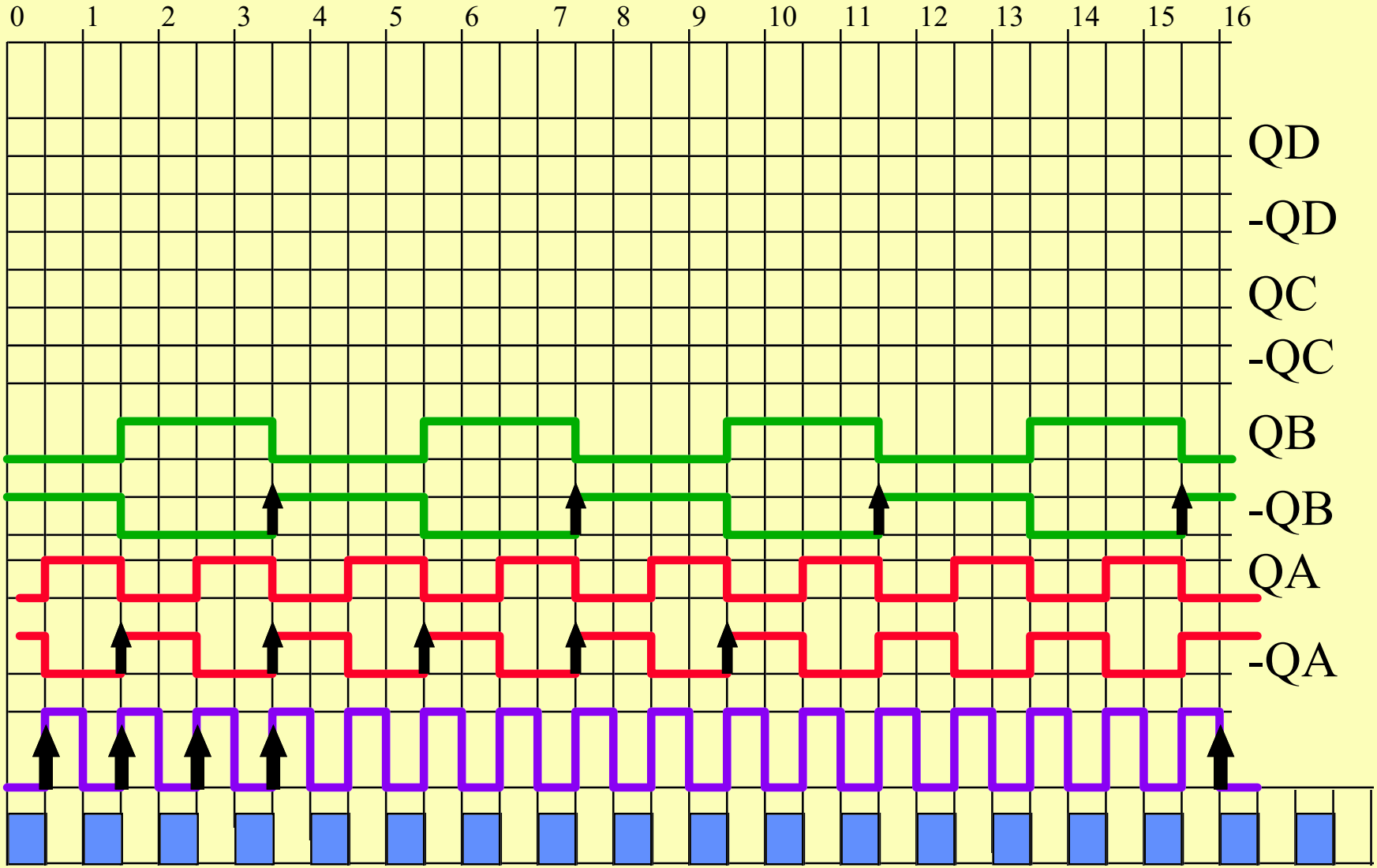
Using Positive Clock Edges



Clock Edge

Logic Functions.

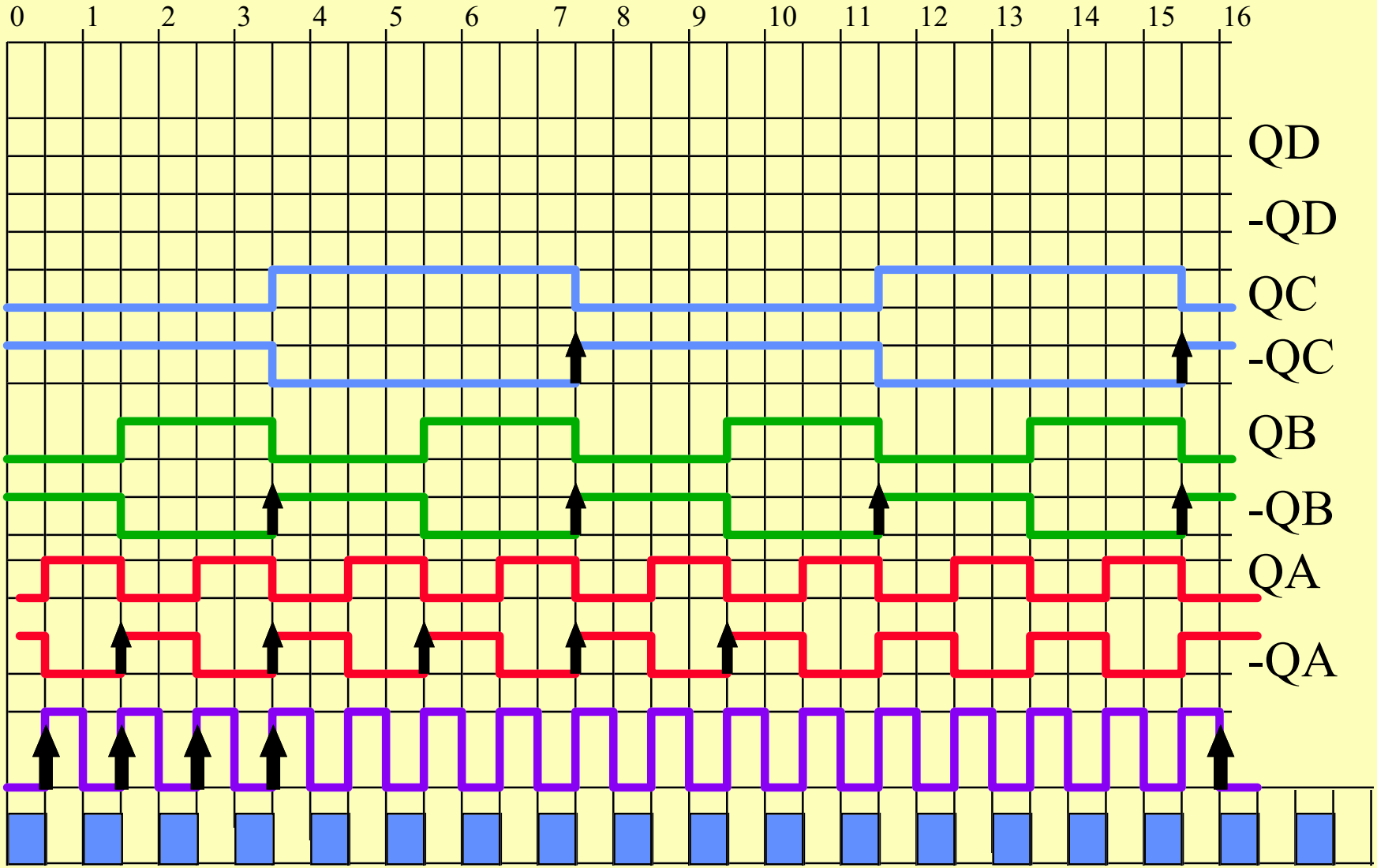
Using Positive Clock Edges



Clock Edge

Logic Functions.

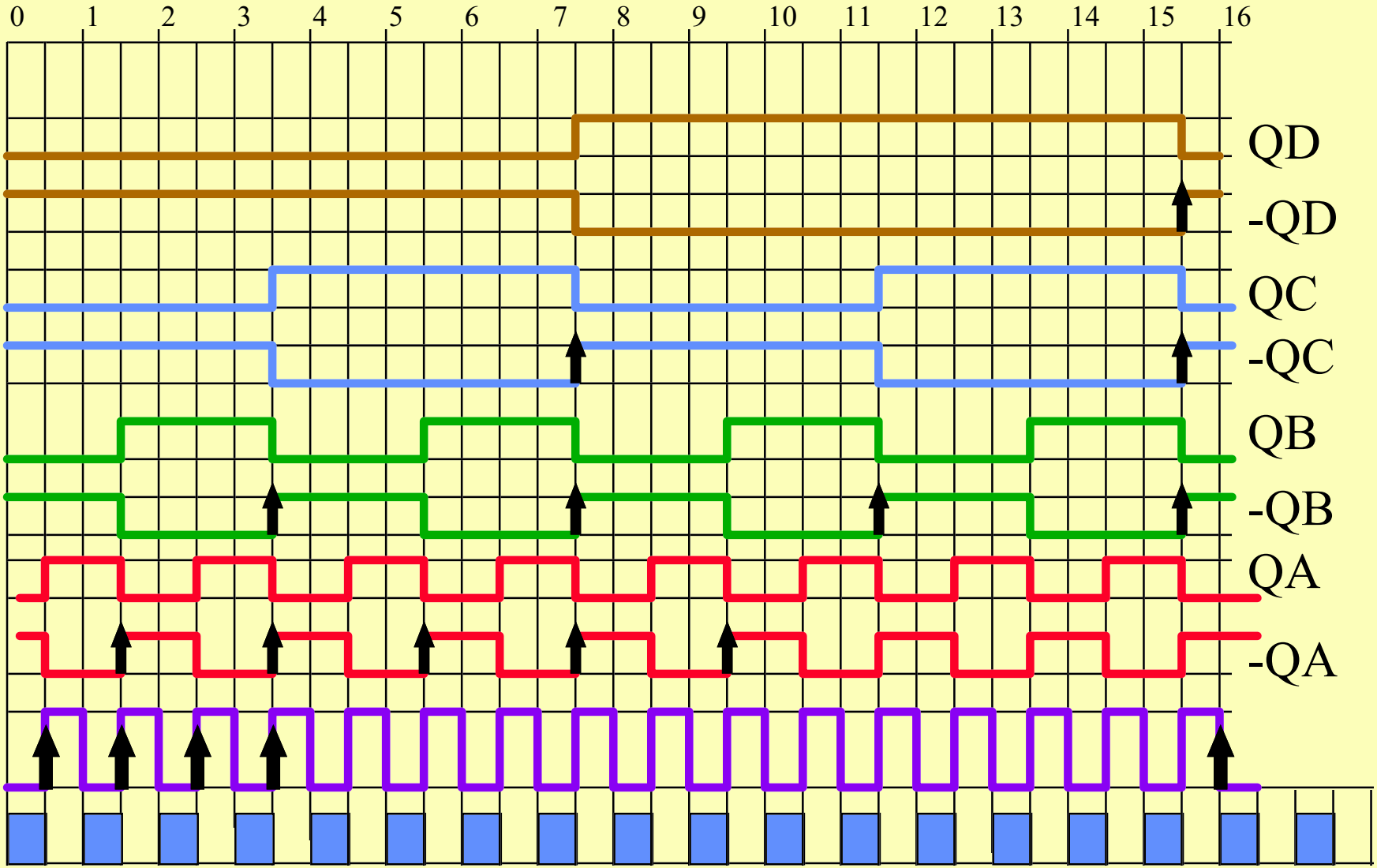
Using Positive Clock Edges



Clock Edge

Logic Functions.

Using Positive Clock Edges



Logic Functions.

Let us convert the levels to Binary Numeric form with their representative decimal value equivalents.

Logic Functions.

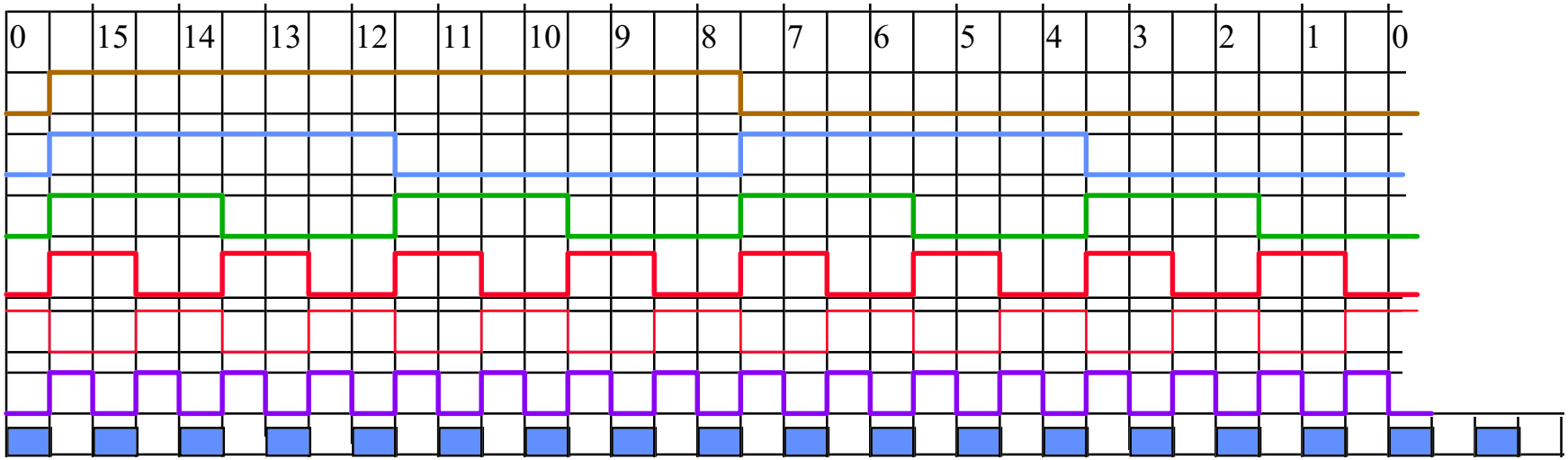
Negative Edge Counting

QD	QC	QB	QA	Value
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

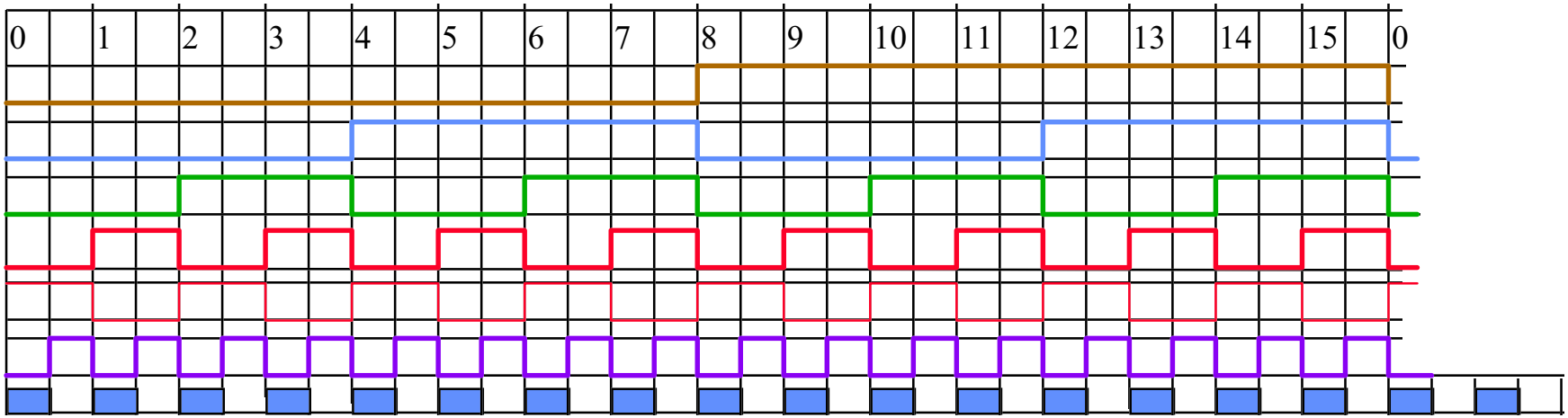
Positive Edge Counting

QD	QC	QB	QA	Value
1	1	1	1	15
1	1	1	0	14
1	1	0	1	13
1	1	0	0	12
1	0	1	1	11
1	0	1	0	10
1	0	0	1	9
1	0	0	0	8
0	1	1	1	7
0	1	1	0	6
0	1	0	1	5
0	1	0	0	4
0	0	1	1	3
0	0	1	0	2
0	0	0	1	1
0	0	0	0	0

Logic Functions.



Positive Clocked D Types.



Negative Clocked D Types.

Logic Functions.

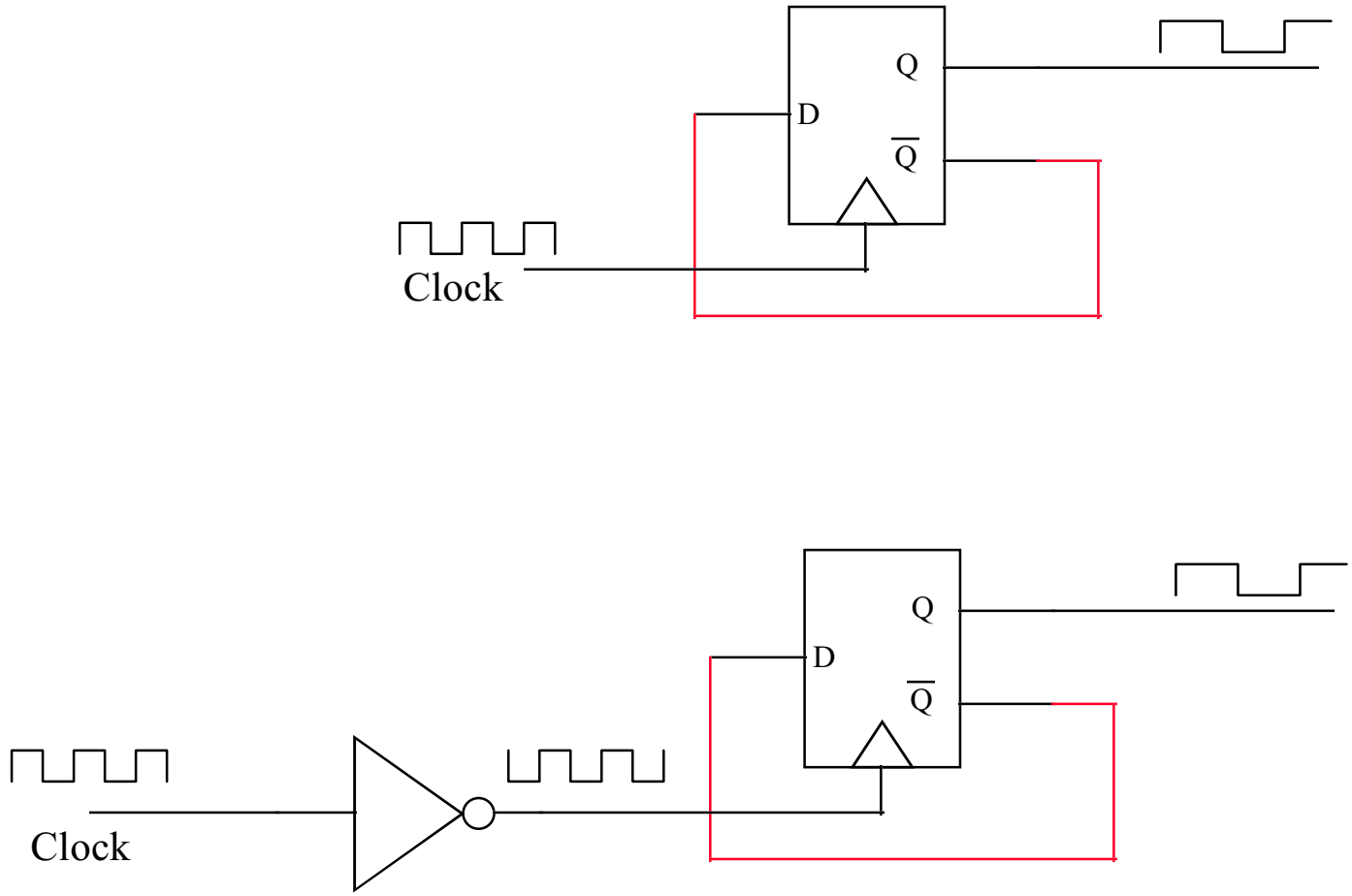
- Positive and Negative Edge Trigger Summary.
- Observation 1. The time reference where the Flip/Flops change is half a cycle different.
- Observation 2. Output Waveforms are the inverse of each other if you disregard the half cycle clock difference.
- Observation 3. If the Flip/Flops are connected the same way (Using Q → D) then Negative Edge triggered Chips **Count UP** and Positive Edge triggered Chips **Count Down**.

Logic Functions.

? ? ? QUESTION ? ? ?

- Can we convert a Positive Edge Trigger Flip/Flop to a Negative Edge Trigger Flip/Flop ?

Logic Functions.



Yes by simply inverting the Clock Signal

Logic Functions.

? ? ? QUESTION ? ? ?

- How do we produce dividers that are not a power of two ?

Logic Functions.

MSB QD	QC	QB	LSB QA	Decimal
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11

The Binary and Decimal Counting sequence of a counter

Is there anything unique we could use to produce a count sequence of TEN.

Logic Functions.

MSB QD	QC	QB	LSB QA	Decimal
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11

When we detect this pattern here.

Logic Functions.

MSB QD	QC	QB	LSB QA	Decimal
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11

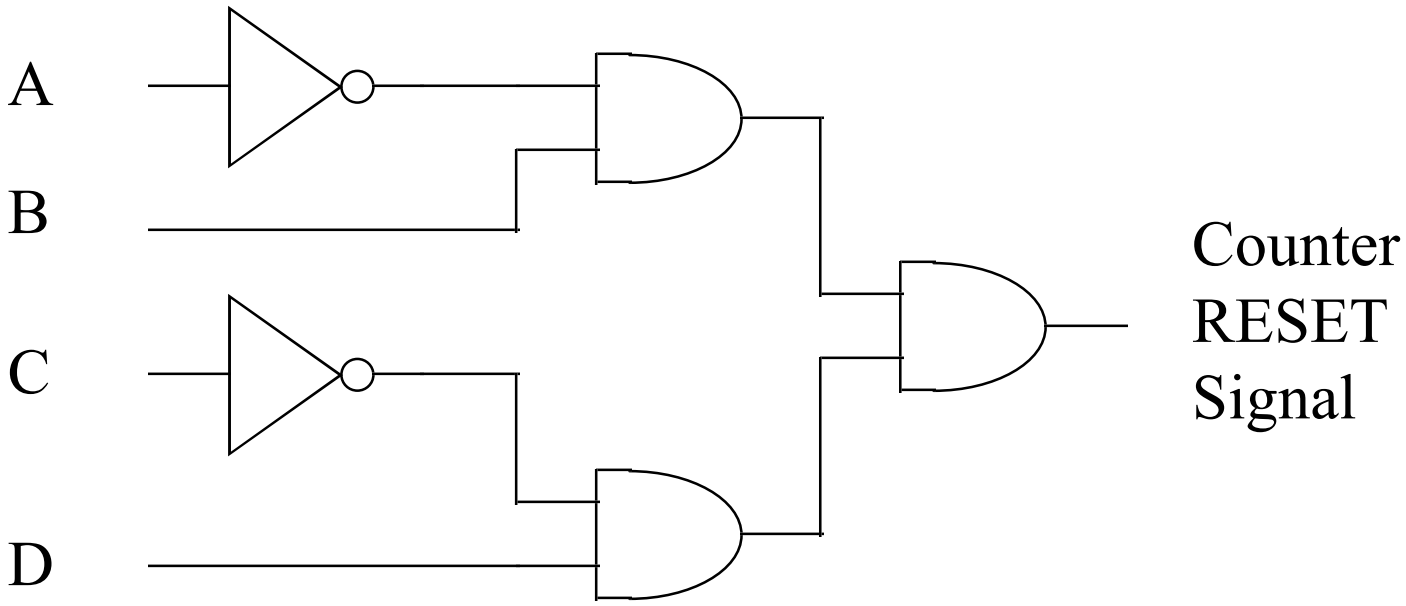
We RESET the counter back to here.

Logic Functions.

? ? ? QUESTION ? ? ?

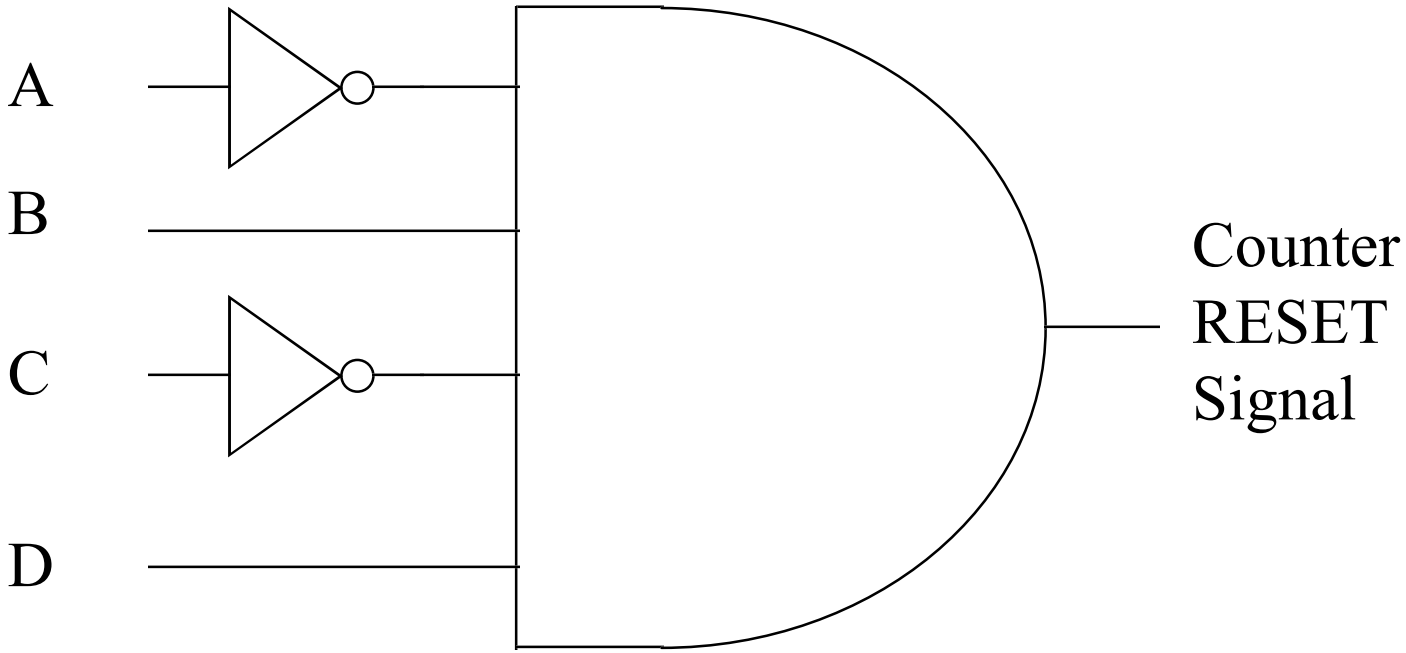
- How do we detect the RESET Pattern ?

Logic Functions.



Option One Use Three Two Input AND Gates.

Logic Functions.



Option Two Use One Quad Input AND Gate.

Logic Functions.

? ? ? QUESTION ? ? ?

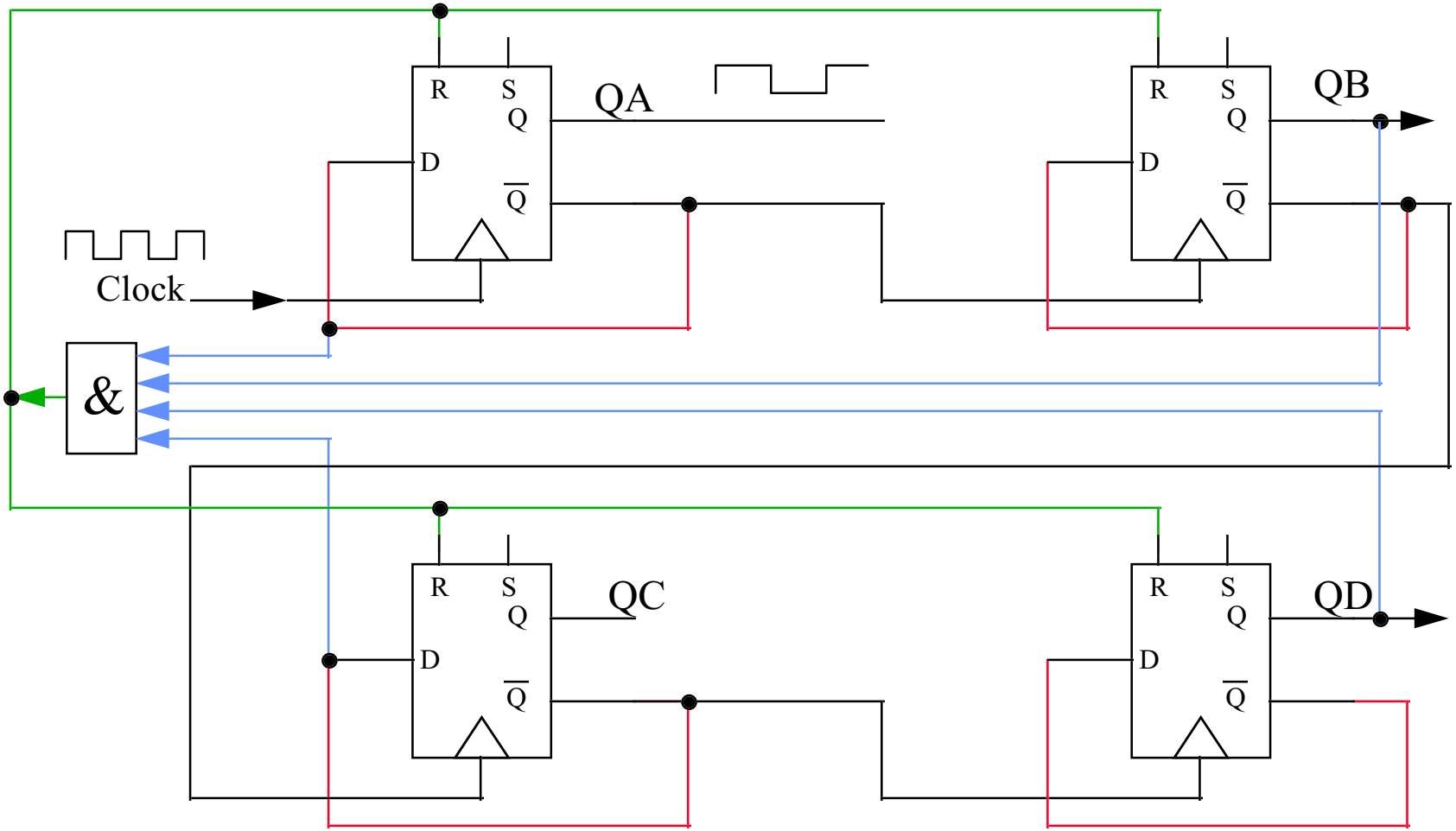
- What will the counting sequence look like ?

Logic Functions.

MSB QD	QC	QB	LSB QA	Decimal
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	*0*	*1*	*0*	*Transient*

The Binary and Decimal Counting sequence of a Divide by Ten counter

Logic Functions.



Negative Edge D Type Flip/Flops+RS as a Divide by 10 Divider.

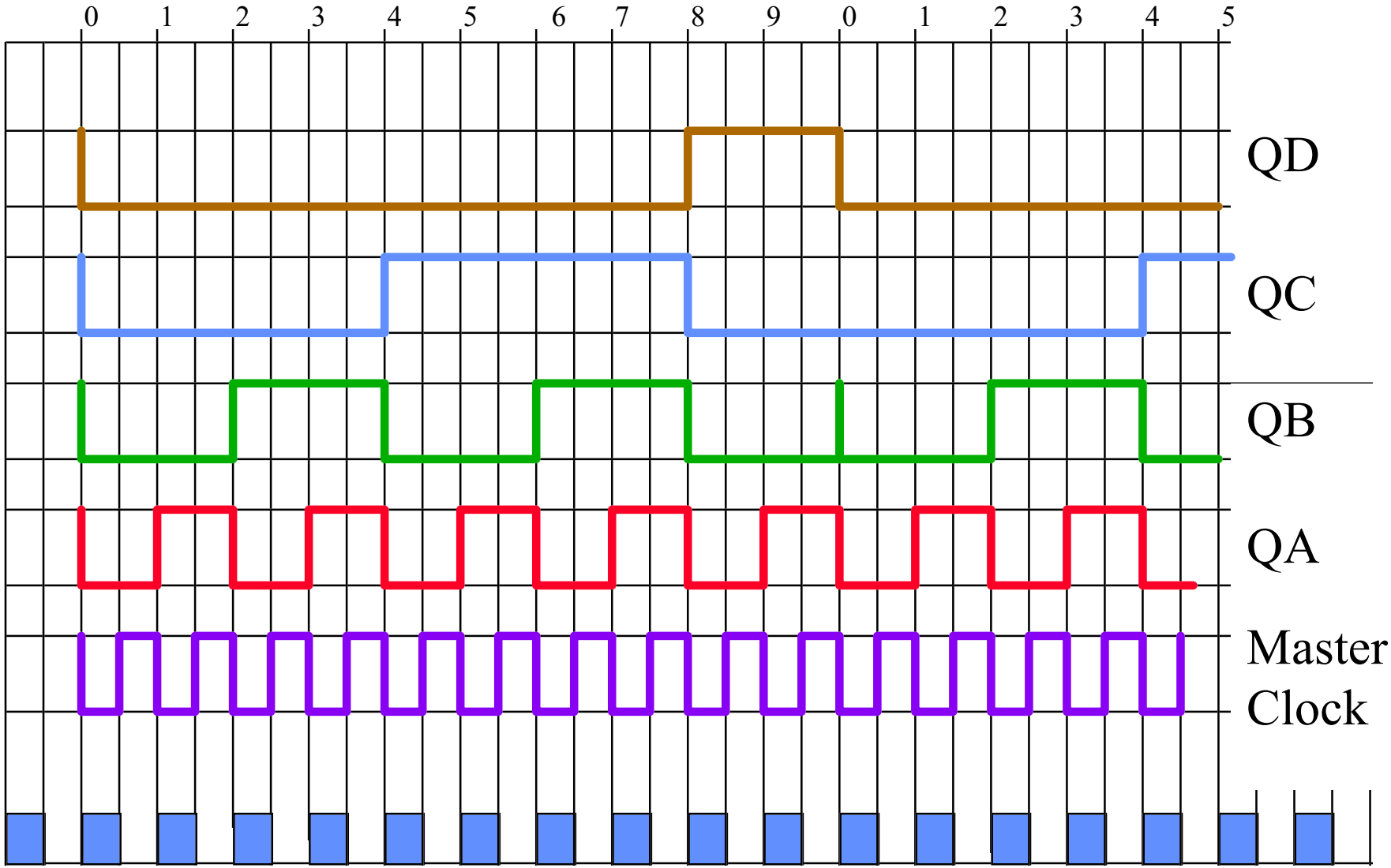
Note the Clock comes from inverted Output therefore Negative Edge triggered.

Logic Functions.

? ? ? QUESTION ? ? ?

- What will the Waveforms look like ?

Logic Functions.



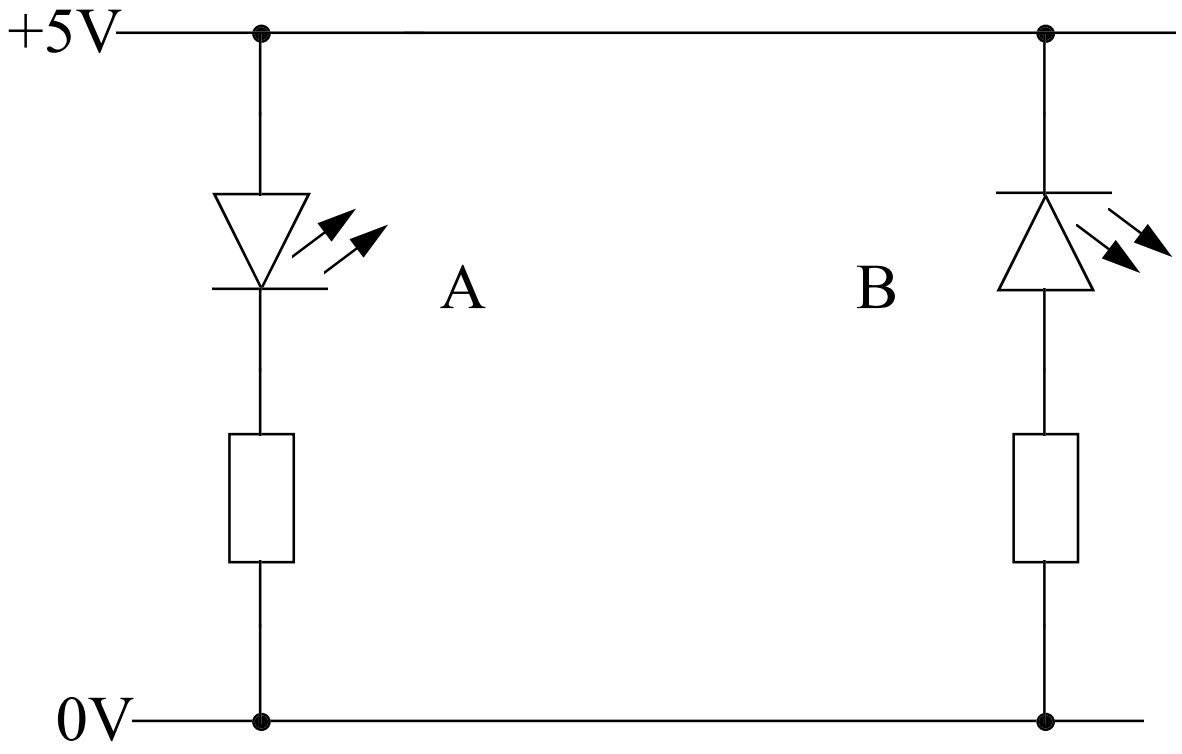
Logic Functions.

? ? ? QUESTION ? ? ?

- How can we Display Numeric Data?

Logic Functions.

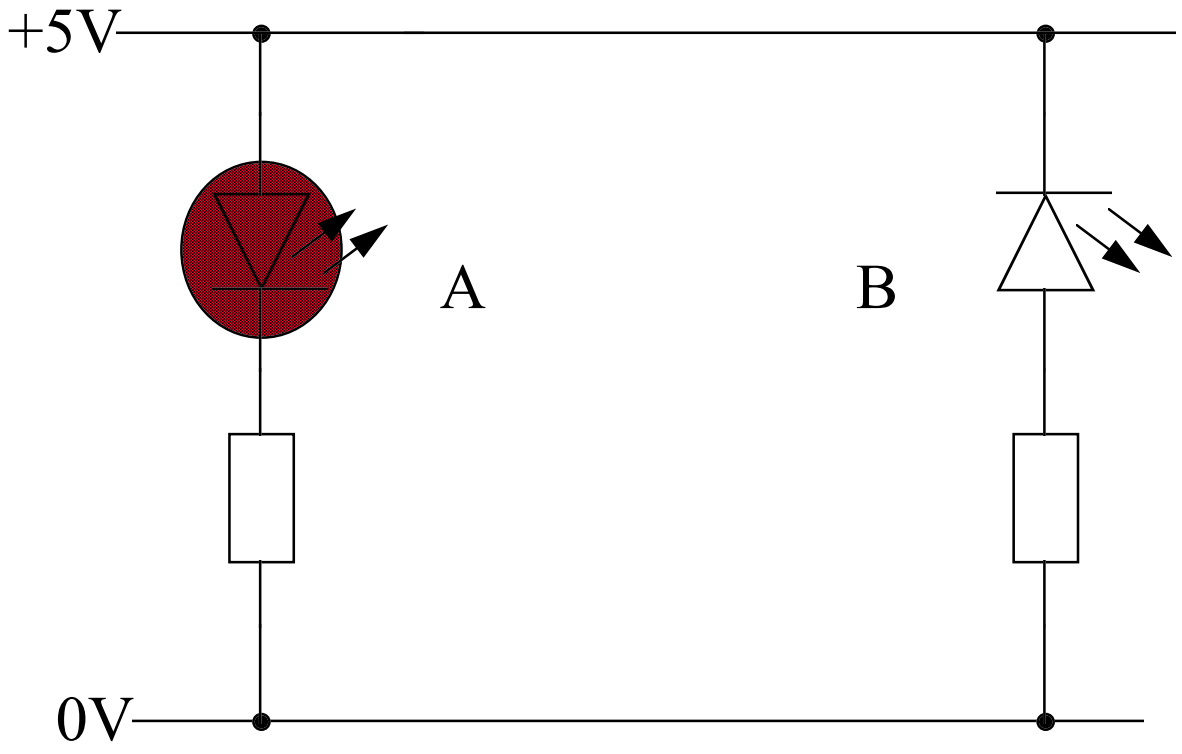
Resistor Diode Logic.



Question which LED will Glow?

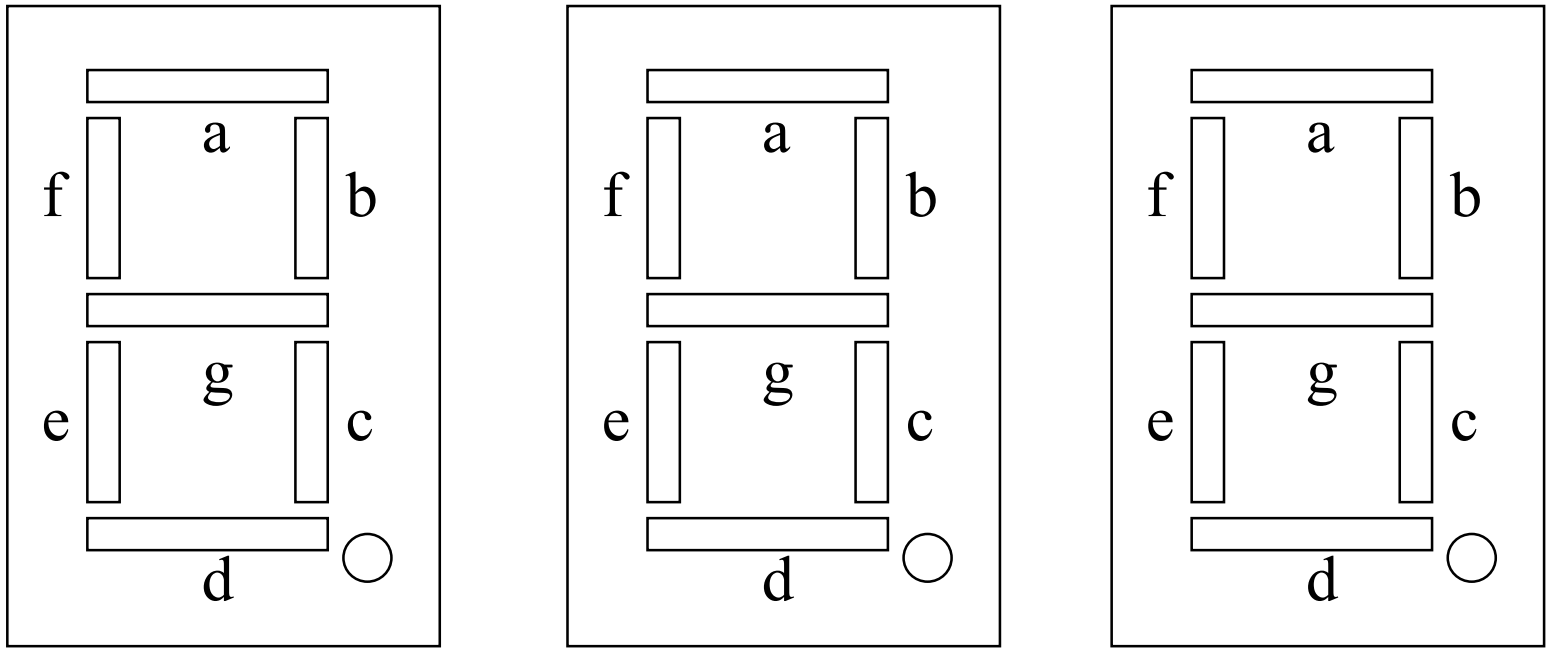
Logic Functions.

Resistor Diode Logic.



Did you get the correct answer?

Logic Functions.

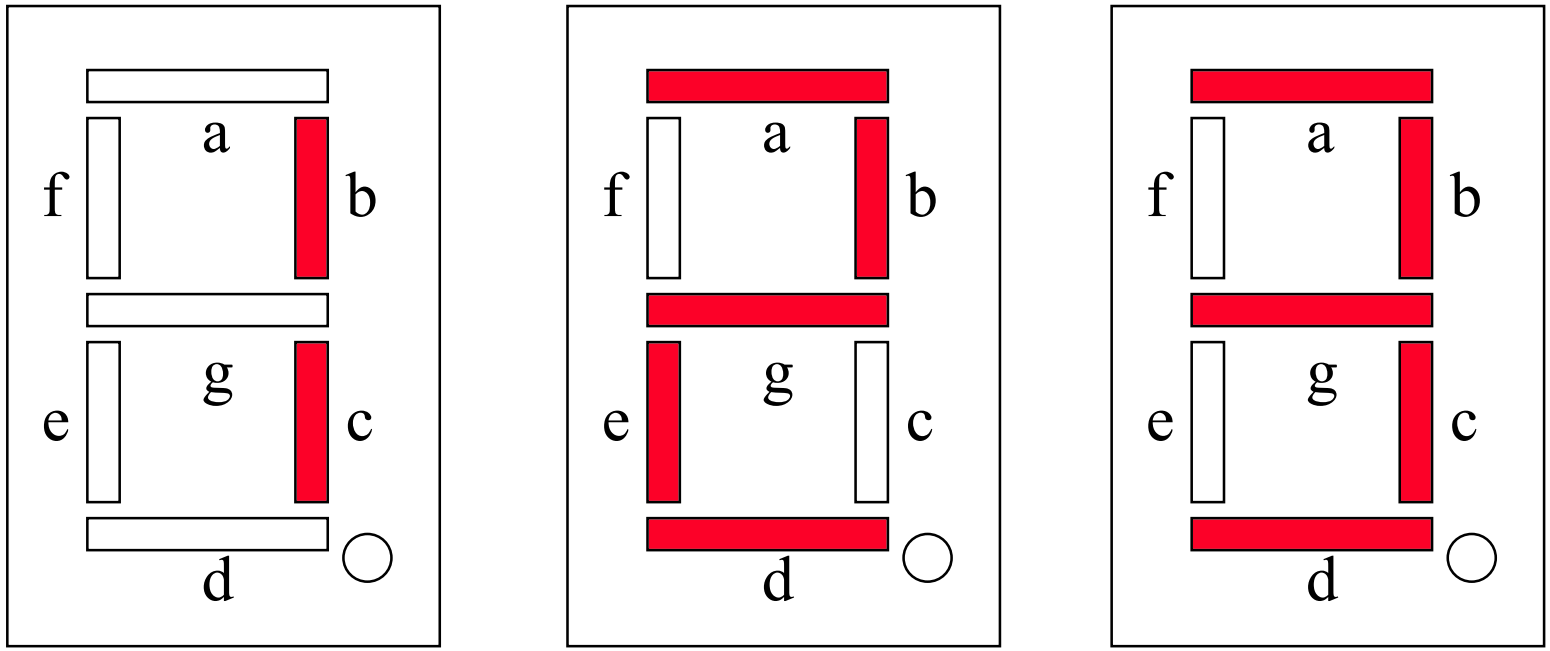


How we create the Digit Displays

Which Segments are lit up to give the display 123 ?

(The Common Cathode) Seven Segment Display.

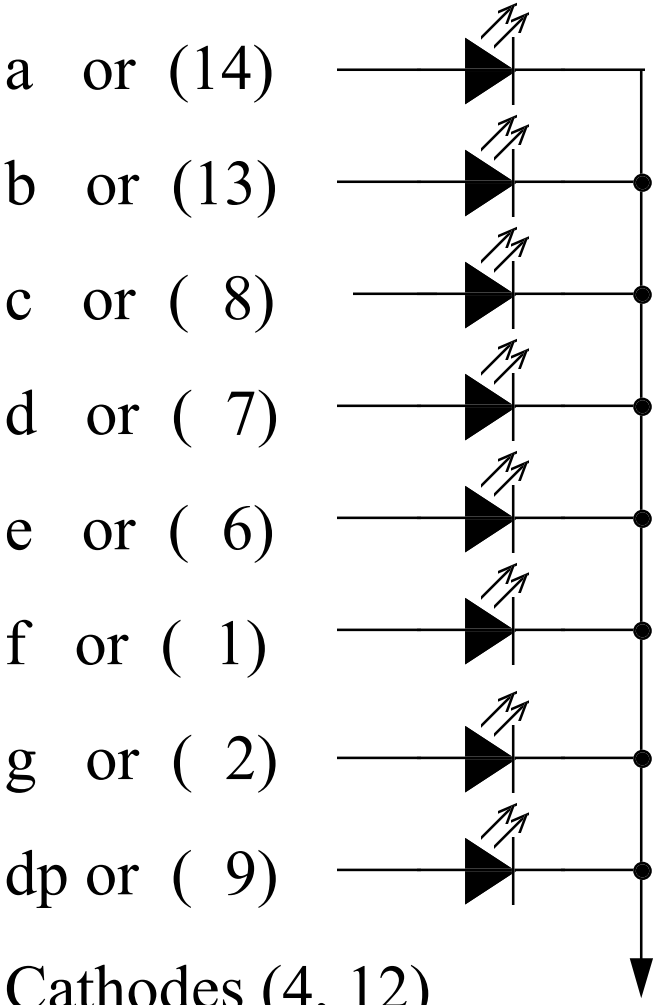
Logic Functions.

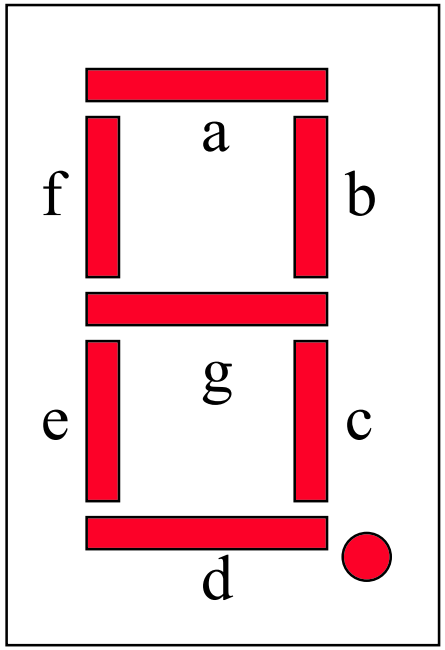


How we create the Digit Displays 123.

(The Common Cathode) Seven Segment Display.

Logic Functions.

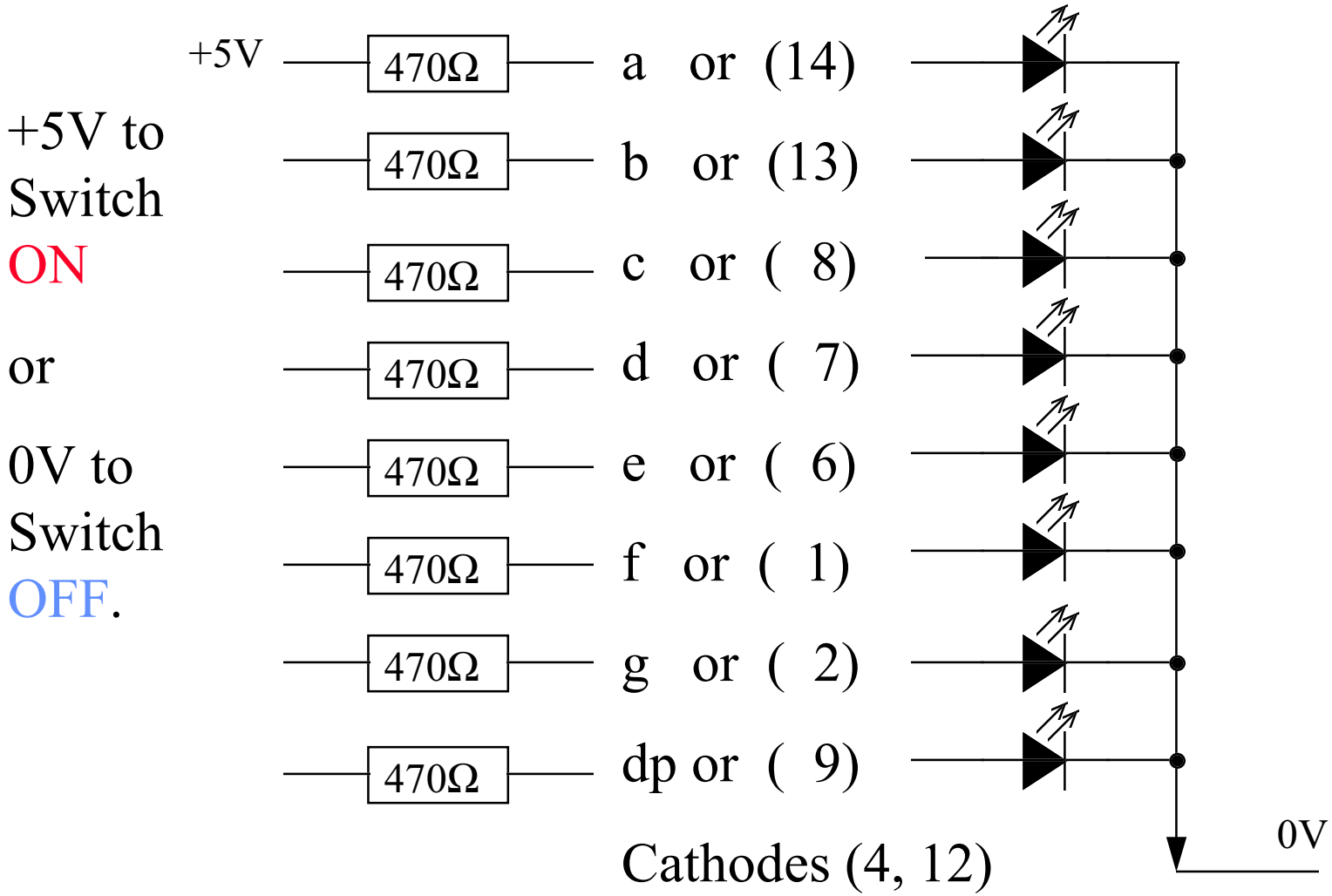




N/c Pins are (3,5,10 and 11)

(The Common Cathode) Seven Segment Display.

Logic Functions.



+5V to Switch ON

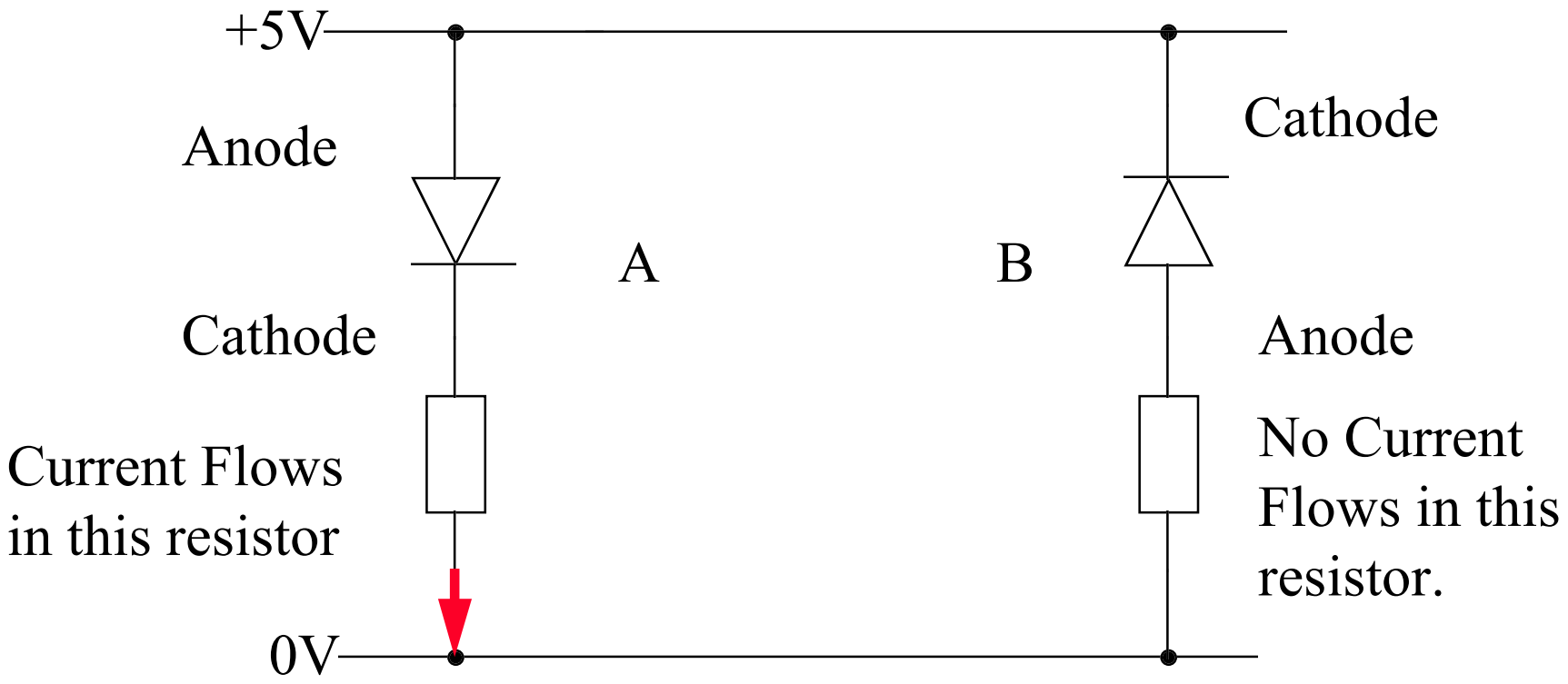
or

0V to Switch OFF.

(The Common Cathode) Seven Segment Display.

Logic Functions.

Resistor Diode Logic.

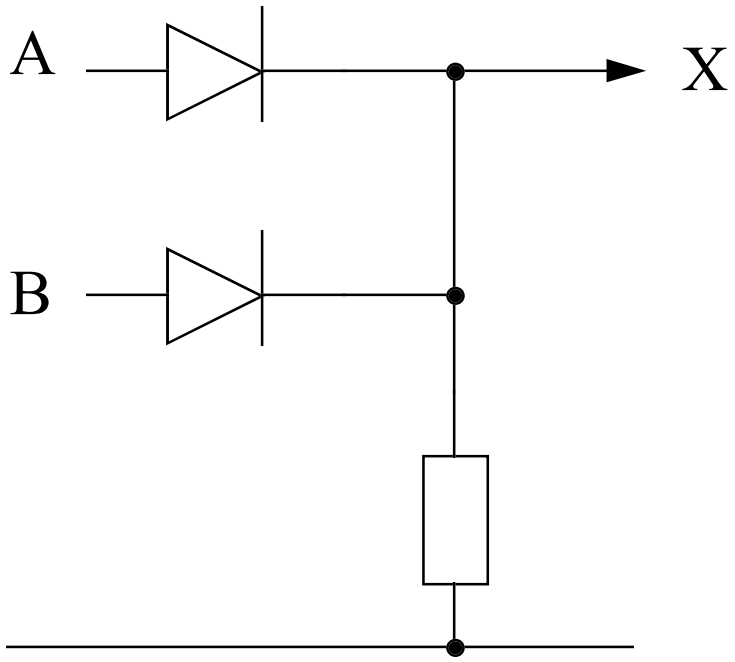


Using a standard Diode.

Logic Functions.

● = 0 = 0V

● = 1 = +5V



A	B	X	
0	0	.	.
0	1	.	.
1	0	.	.
1	1	.	.

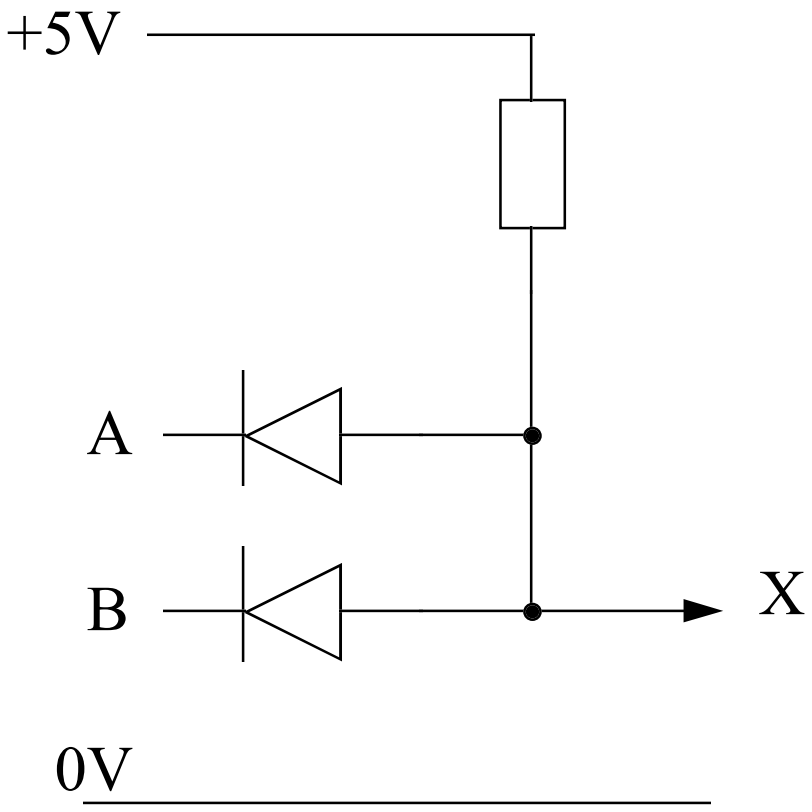
Resistor Diode Logic.

Logic Functions.

● = 0 = 0V

● = 1 = +5V

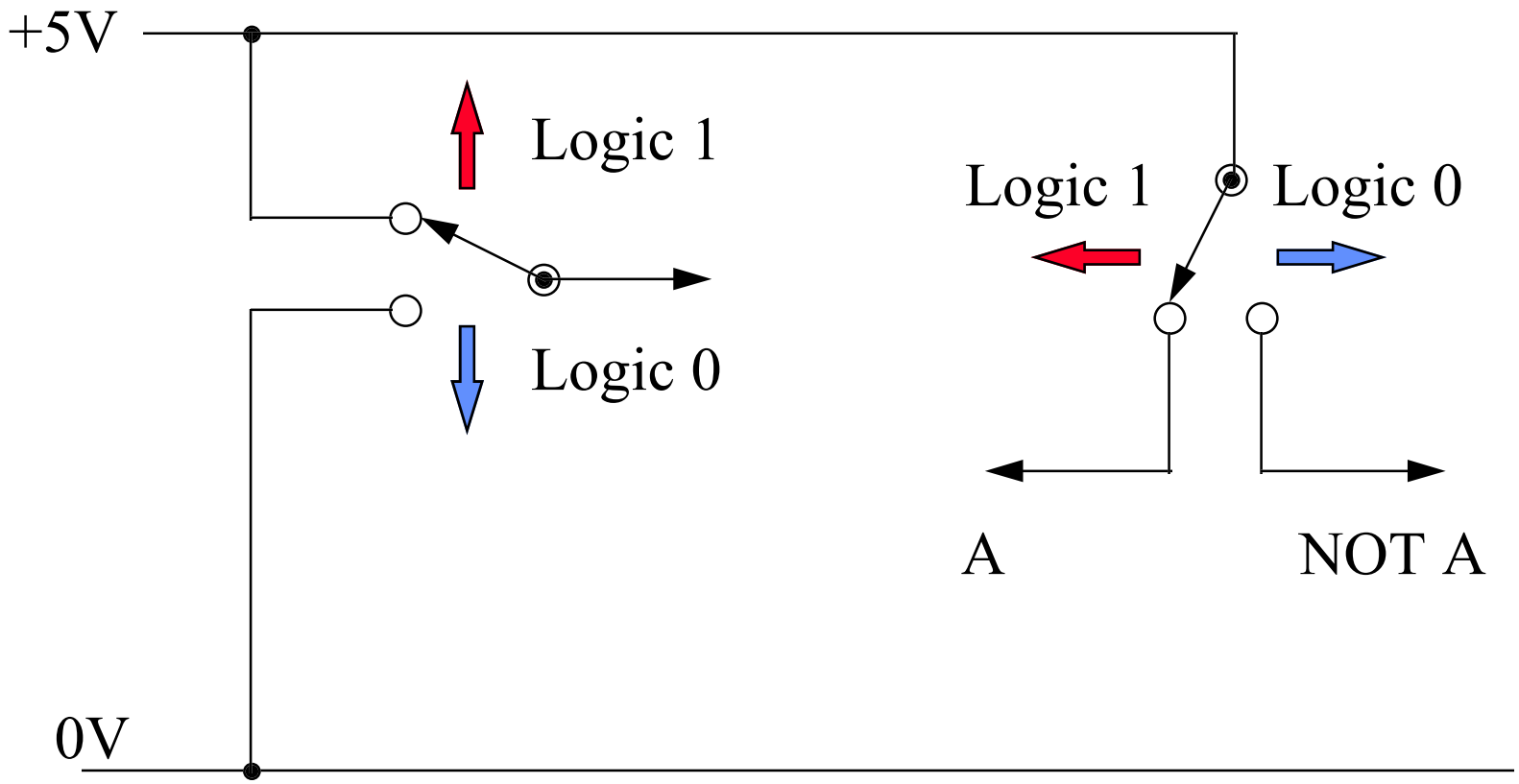
Resistor Diode Logic.



A	B	X
0	0	. .
0	1	. .
1	0	. .
1	1	. .

Resistor Diode Logic.

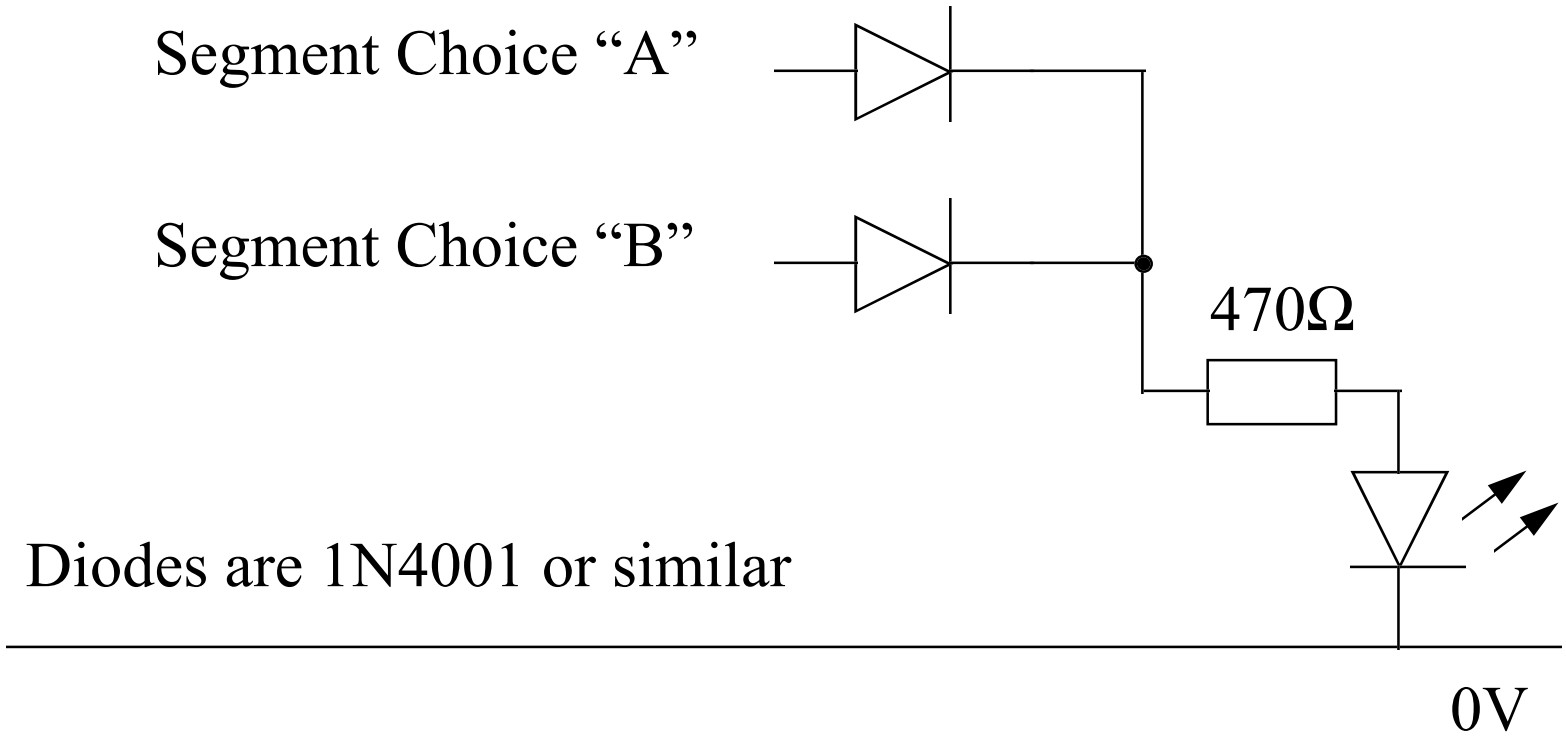
Logic Functions.



Switch Logic Options.

Logic Functions.

● = 0 = 0V
● = 1 = +5V

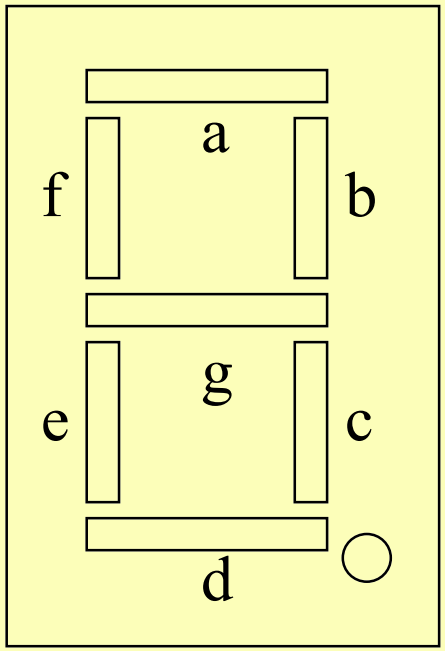


Resistor Diode Logic.

Logic Functions.

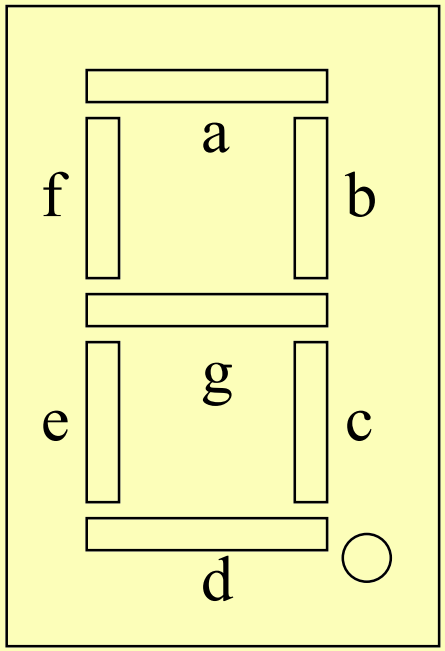
Bin	Dec	a	b	c	d	e	f	g
0 0 0 0	0							
0 0 0 1	1							
0 0 1 0	2							
0 0 1 1	3							
0 1 0 0	4							
0 1 0 1	5							
0 1 1 0	6							
0 1 1 1	7							
1 0 0 0	8							
1 0 0 1	9							
1 0 1 0	10							
1 0 1 1	11							
1 1 0 0	12							
1 1 0 1	13							
1 1 1 0	14							
1 1 1 1	15							

Practice Example.

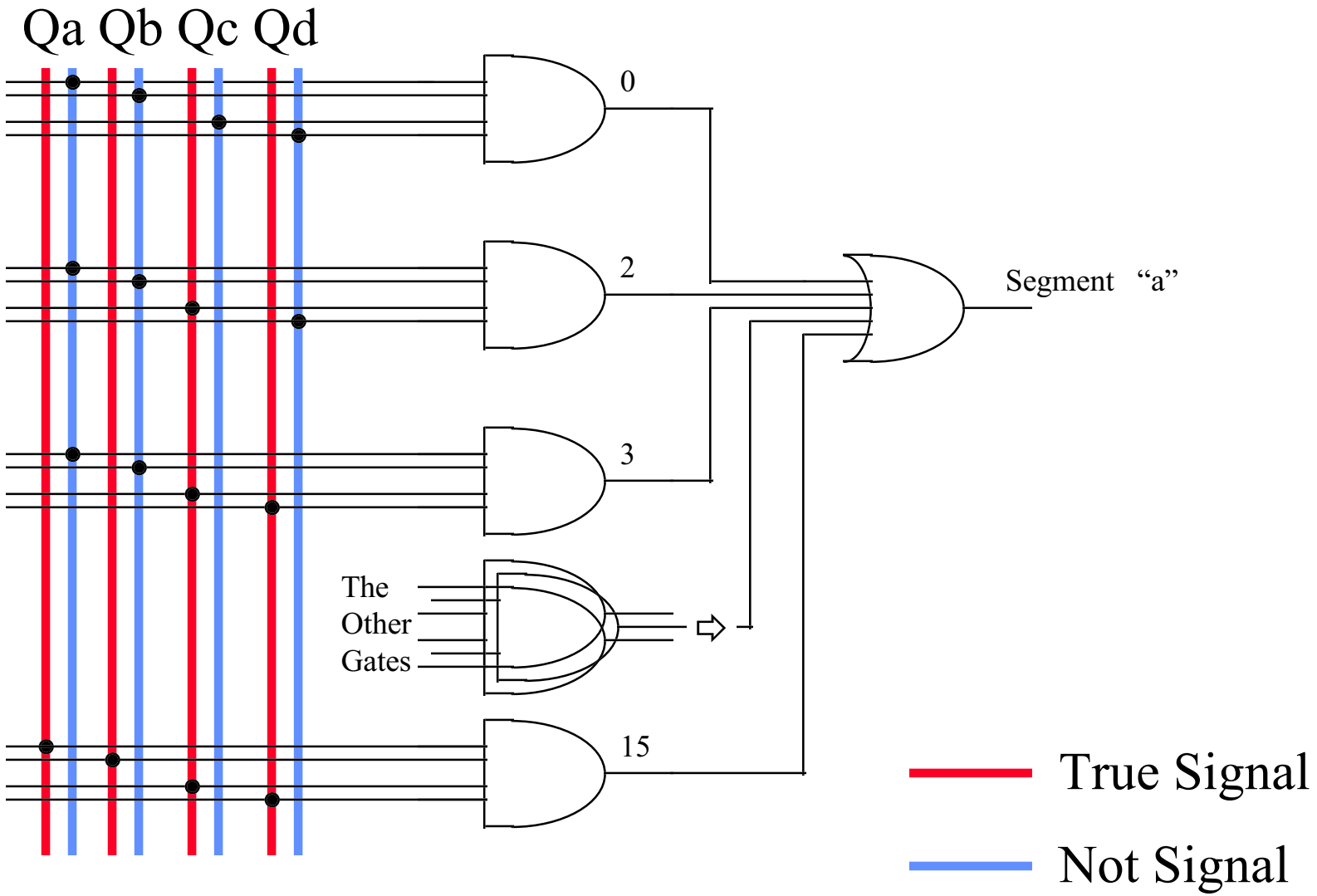


Logic Functions.

Bin	Dec	a	b	c	d	e	f	g
0 0 0 0	0	✓	✓	✓	✓	✓	✓	
0 0 0 1	1		✓	✓				
0 0 1 0	2	✓	✓		✓	✓		✓
0 0 1 1	3	✓	✓	✓	✓			✓
0 1 0 0	4		✓	✓			✓	✓
0 1 0 1	5	✓		✓	✓		✓	✓
0 1 1 0	6	✓		✓	✓	✓	✓	✓
0 1 1 1	7	✓	✓	✓				
1 0 0 0	8	✓	✓	✓	✓	✓	✓	✓
1 0 0 1	9	✓	✓	✓			✓	✓
1 0 1 0	10	✓	✓	✓		✓	✓	✓
1 0 1 1	11			✓	✓	✓	✓	✓
1 1 0 0	12	✓			✓	✓	✓	
1 1 0 1	13		✓	✓	✓	✓		✓
1 1 1 0	14	✓			✓	✓	✓	✓
1 1 1 1	15	✓				✓	✓	✓



Logic Functions.



Logic Functions.

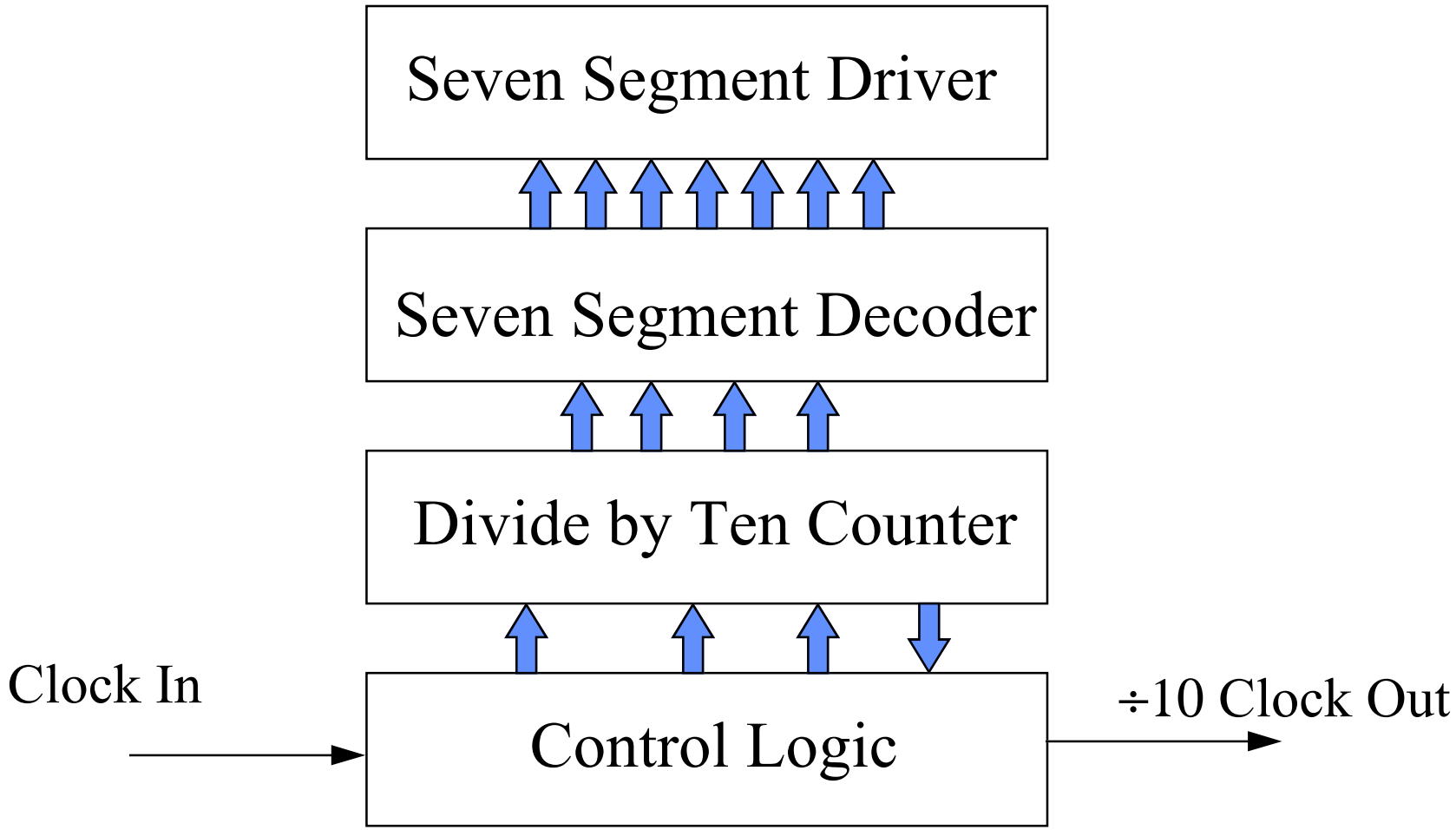
Making our life easy.

Using the 4026 Decade Counter Decoder.

or

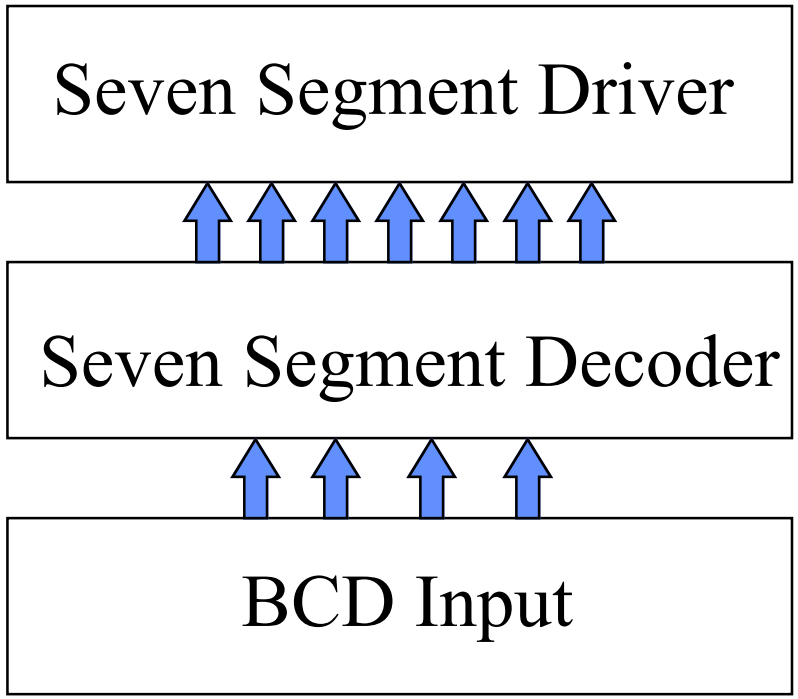
Using the 4511 Display Decoder.

Logic Functions.



The Basic Block diagram of a 4026 Chip.

Logic Functions.



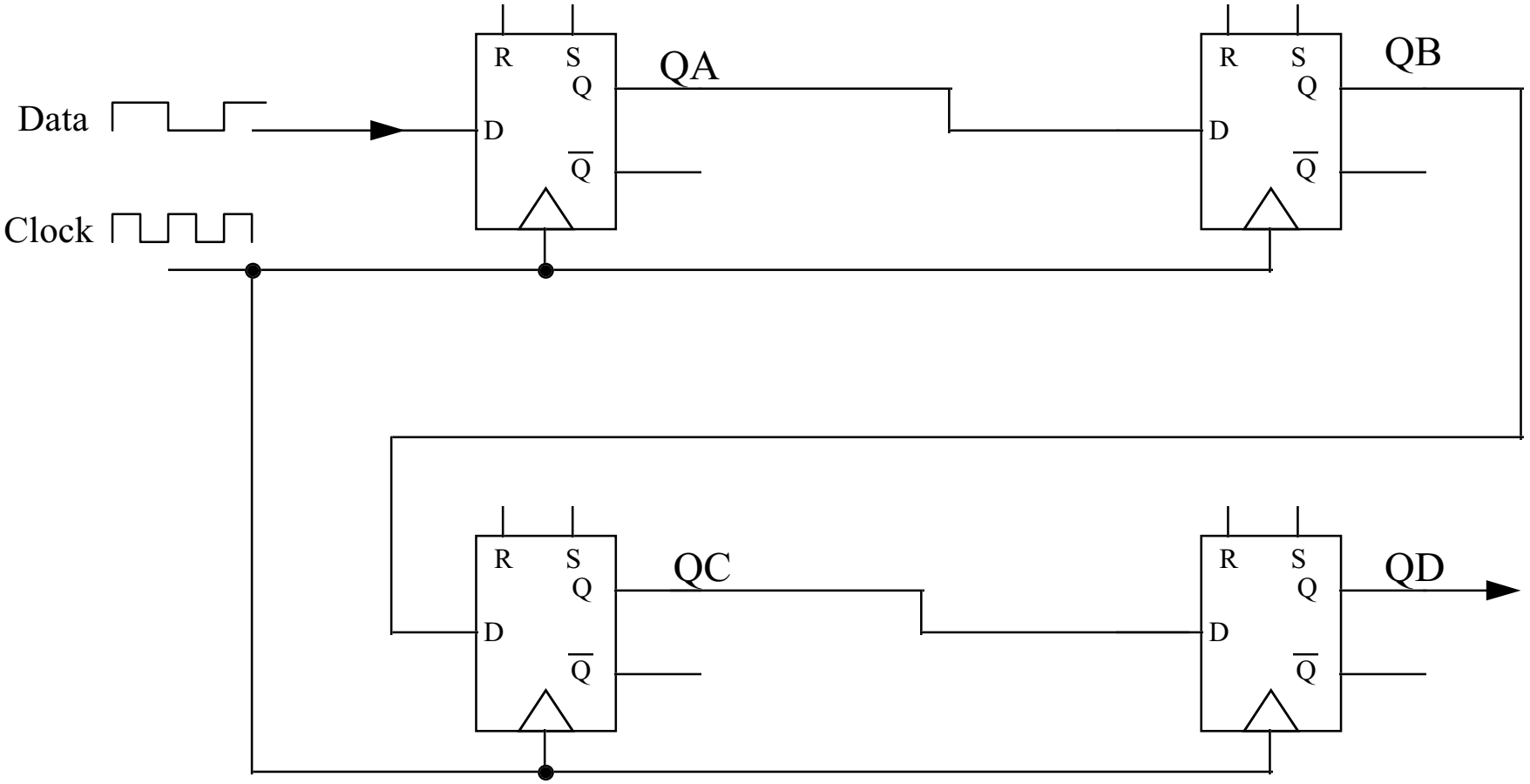
The Basic Block diagram of a 4511 Chip.

D Flip Flops Configured as Shift Registers

Logic Functions.

- What is a Shift Register
- A configuration that allows data to be :-
  - Converted from Serial to Parallel format.
  - Converted from Parallel to Serial format.
  - Delayed for a time period.
- It may be used as a memory system.
  - Typical applications are :-
    - Sound reverberation or echo effects.
    - Message or display storage

Logic Functions.



Positive Edge D Type Flip/Flops+RS as a shift Register.

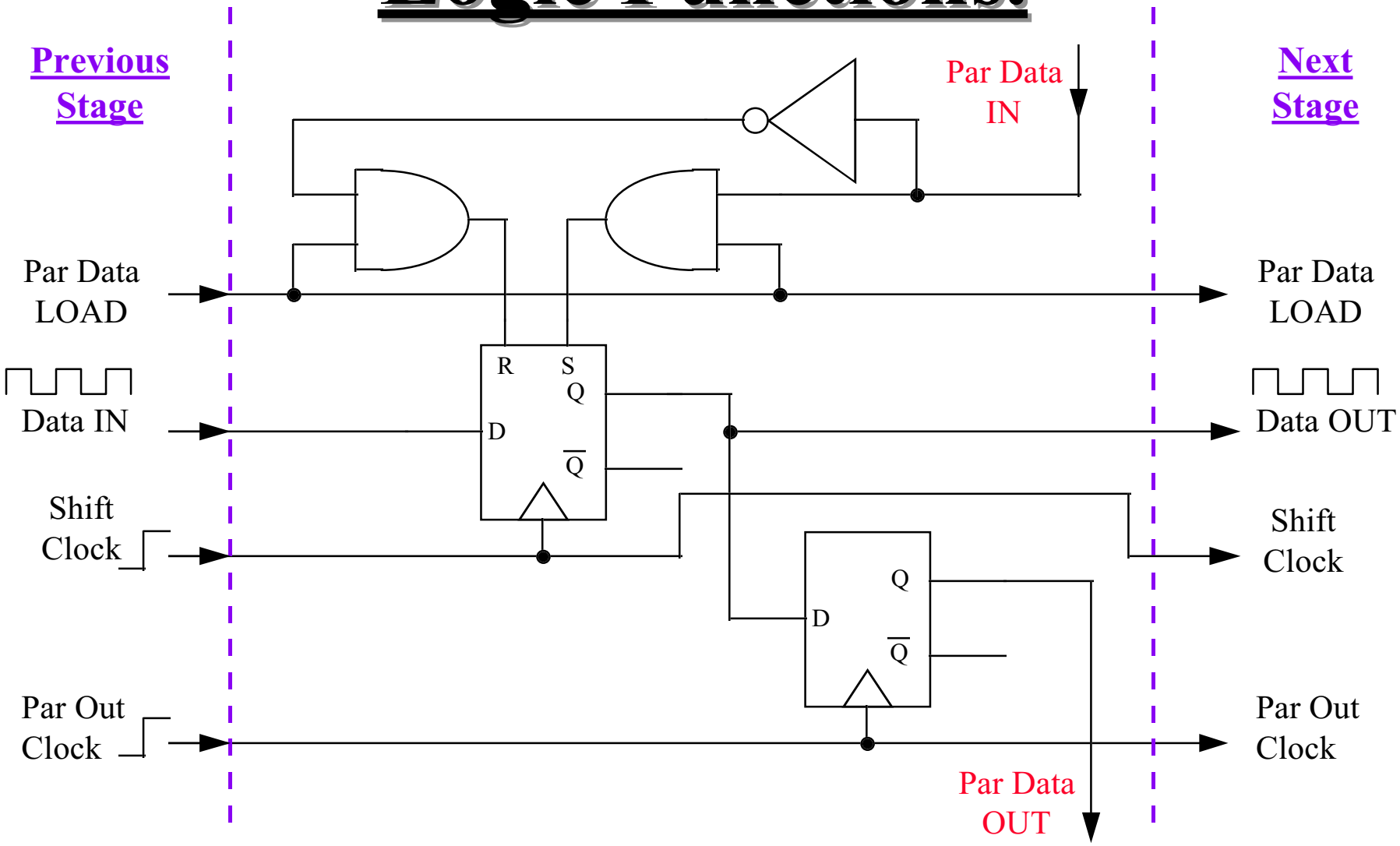
Logic Functions.

Out	MSB			LSB	In	Clock
D	QD	QC	QB	QA	D	
0	0	0	0	0	0	⌐
0	0	0	0	1	1	⌐
0	0	0	1	0	0	⌐
0	0	1	0	0	0	⌐
1	1	0	0	0	0	⌐
0	0	0	0	0	0	⌐
0	0	0	0	1	1	⌐
0	0	0	1	1	1	⌐
0	0	1	1	0	0	⌐
1	1	1	0	0	0	⌐
1	1	0	0	0	0	⌐
0	0	0	0	0	0	⌐

Notice how the Data ripples through the Truth Table on each Clock edge.

A parallel transfer operation could be implemented to Extract or Preset Data as required.

Logic Functions.



A Section of Serial / Parallel Converter

Logic Functions.

DeMorgan Theorem

Logic Functions.

- The DeMorgan theorem. (Why ?)
- To reduce the number of different chips one may need to use in a project it is often useful to be able to use only one type of device.
- The DeMorgan theorem enables you to convert "AND" logic to "OR" logic or "OR" logic to "AND" logic with the help of the "NOT" gate.
- The process required just three simple rules to be obeyed in sequence. NOTE: You must ensure that bracketed entities are treated as a single item.

Logic Functions.

- The DeMorgan theorem.
- Step 1. Invert the Input variables.
- Step 2. Change the function.
  - The “AND” function becomes “OR”.
  - The “OR” function becomes “AND”.
- Step 3. Invert the Output.
- Final activity is just to tidy the equation by removing any double negatives / inversions.

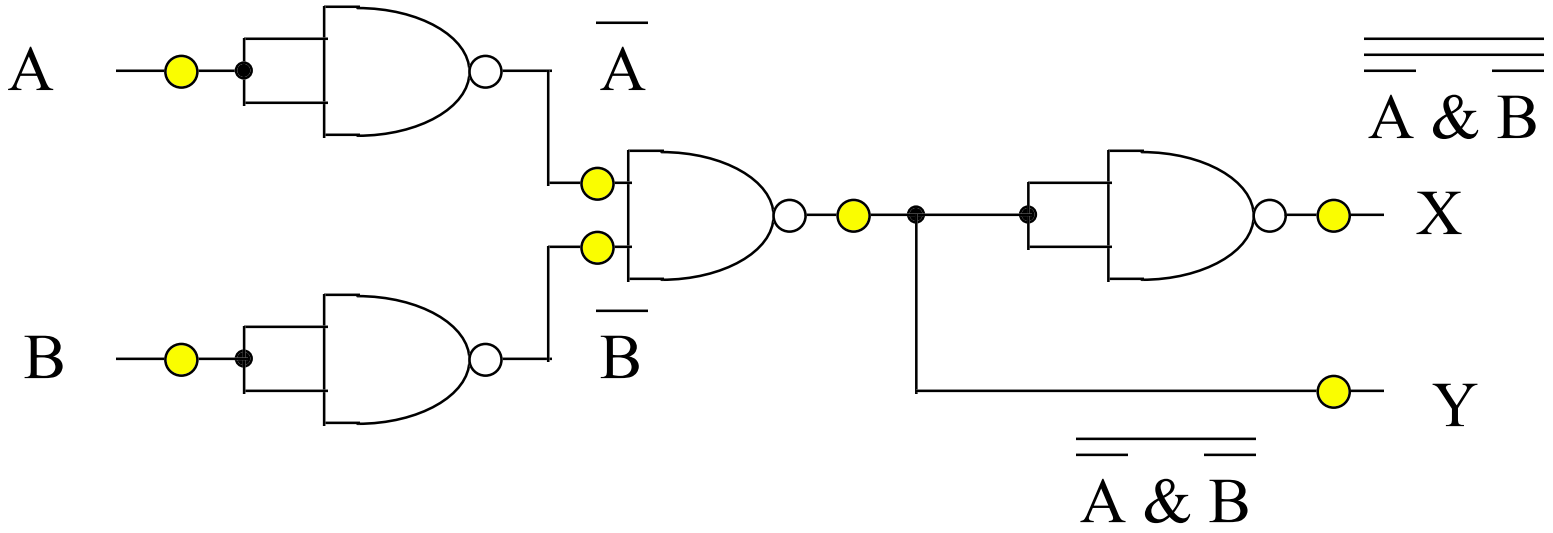
Logic Functions.

● = 0

● = 1

NAND Gate

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0



A	B	X	Y
0	0	.	.
0	1	.	.
1	0	.	.
1	1	.	.

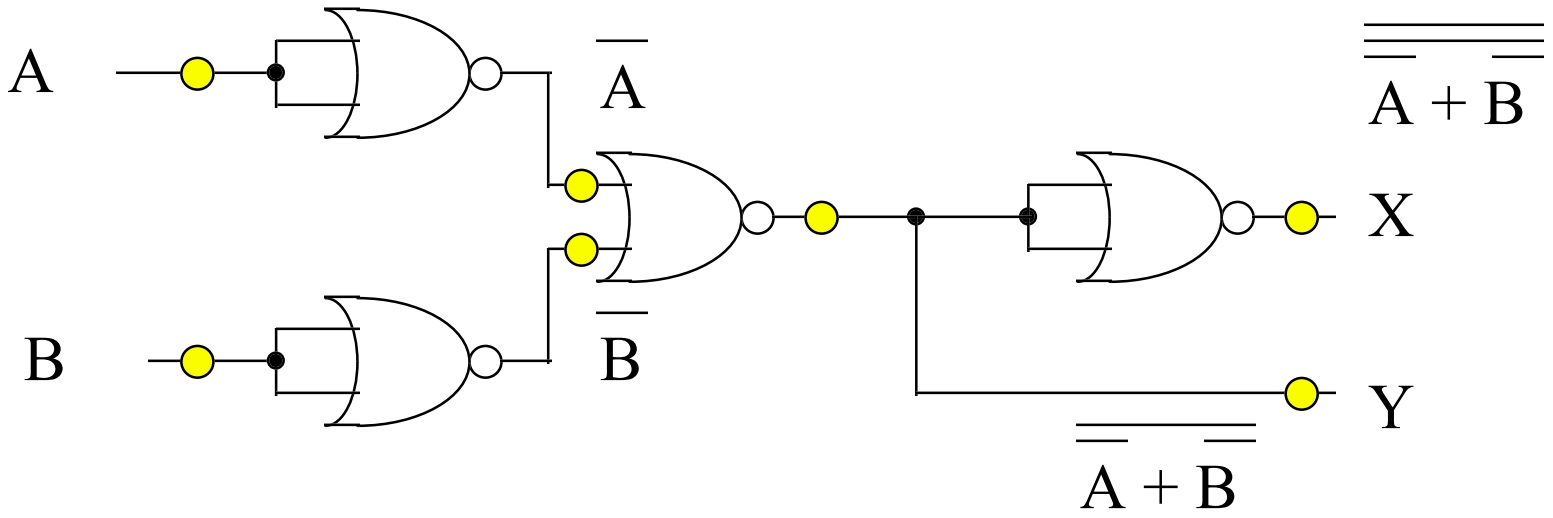
Logic Functions.

● = 0

● = 1

NOR Gate

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0



A	B	X	Y
0	0	.	.
0	1	.	.
1	0	.	.
1	1	.	.

Logic Functions.

Example

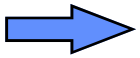
- The DeMorgan theorem.
- Start with a simple NAND function $\overline{A \,\&\, B}$
- Invert the Variables $\overline{\overline{A} \,\&\, \overline{B}}$
- Change the function $\overline{\overline{A} + \overline{B}}$
- Invert the Output. $\overline{\overline{\overline{A} + \overline{B}}}$
- Final activity is just to to tidy the equation by removing any double negatives / inversions.

Gives us ⇨ $\overline{A} + \overline{B}$

Logic Functions.

Example

- The DeMorgan theorem.
- Start with a simple NOR function $\overline{A + B}$
- Invert the Variables $\overline{\overline{A} + \overline{B}}$
- Change the function $\overline{\overline{A} \,\&\, \overline{B}}$
- Invert the Output. $\overline{\overline{\overline{A} \,\&\, \overline{B}}}$
- Final activity is just to to tidy the equation by removing any double negatives / inversions.

Gives us  $\overline{A} \,\&\, \overline{B}$

Logic Functions.

Karnaugh Maps

Logic Functions.

- The Karnaugh Map.
- The is a popular graphical method that allows information to be taken from a Truth table and simplified or Minimized diagrammatically.
- The Truth table is re-drawn so logic relationships can be easily identified.
- For example :-

Let us examine the circuit and equation

(B&A)+(B&-A)

The minus sign in this case indicates "Inversion"

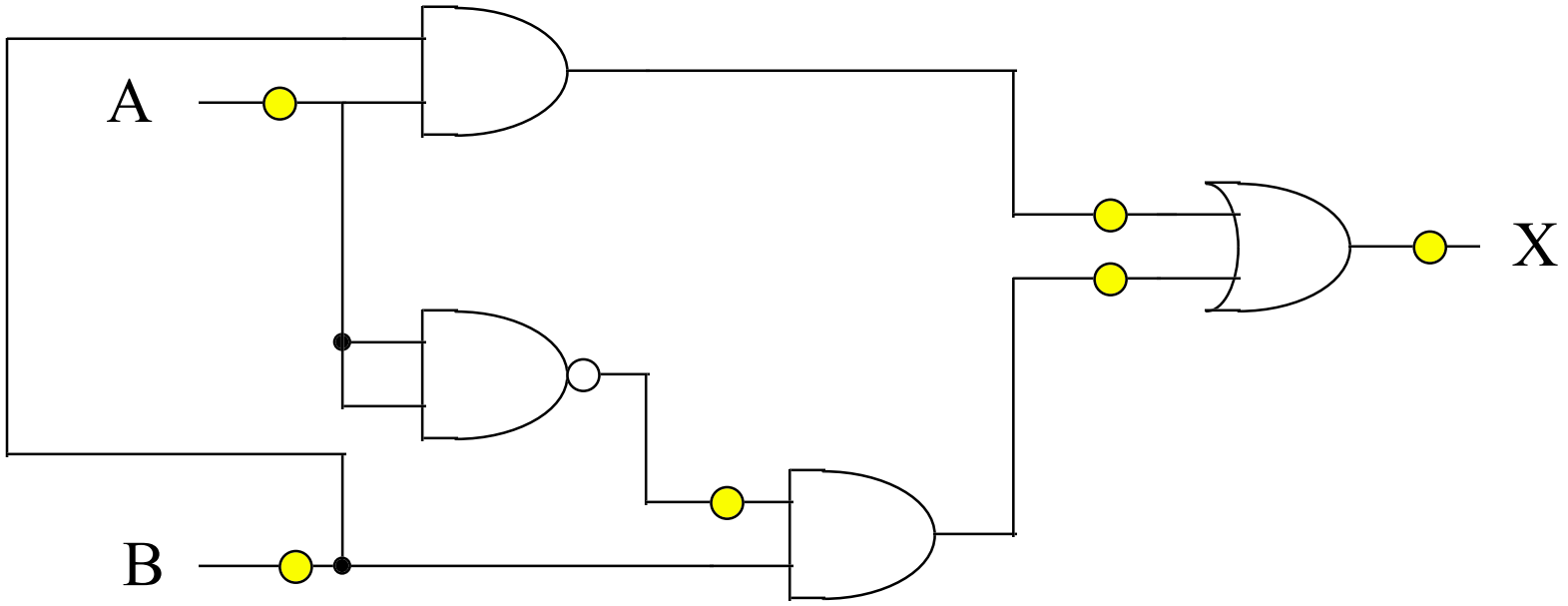
Logic Functions.

● = 0

● = 1

AND Gate

A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1



OR Gate

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

A	B	X	.
0	0	.	.
0	1	.	.
1	0	.	.
1	1	.	.

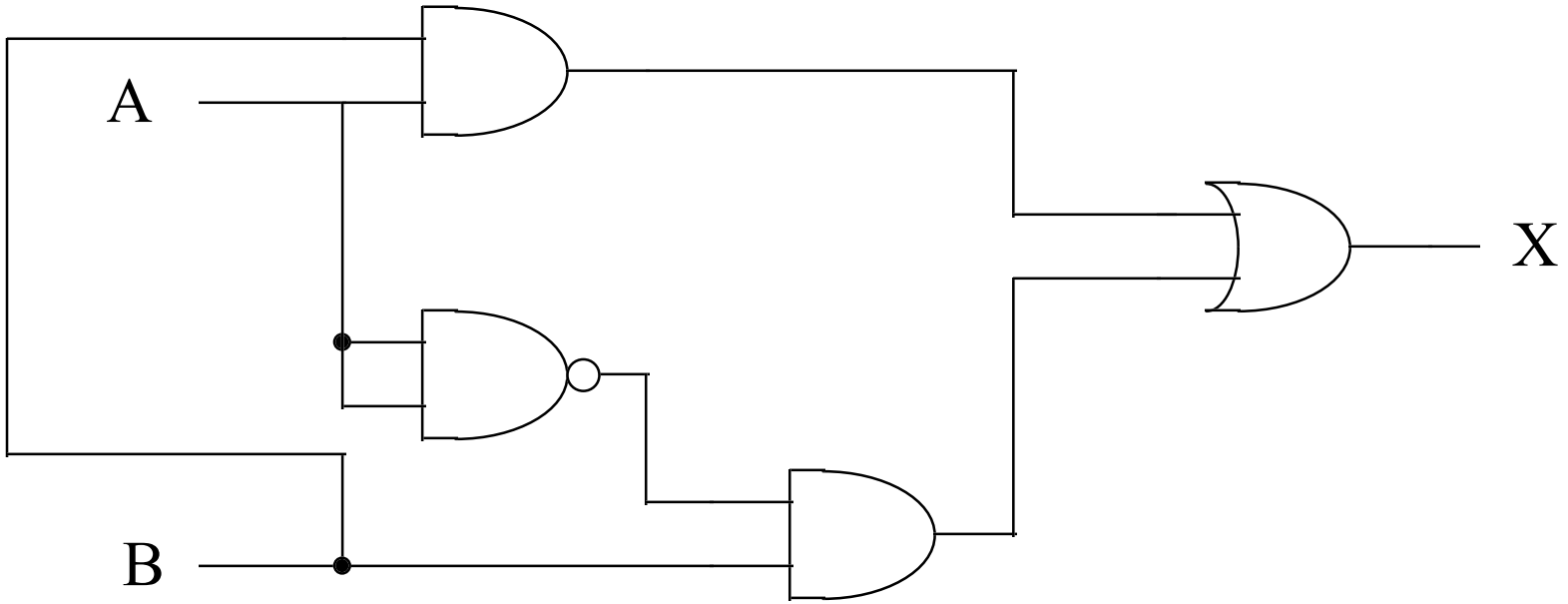
Logic Functions.

● = 0

● = 1

AND Gate

A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1



OR Gate

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

(B&A)+(B&-A)= **B**

Conclusion is

A is redundant in this case

A	B	X	.
0	0	0	.
0	1	1	.
1	0	0	.
1	1	1	.

Logic Functions.

- The Karnaugh Map.
- The is a popular graphical method that allows information to be taken from a Truth table and simplified or Minimized diagrammatically.
- The Truth table is re-drawn so logic relationships can be easily identified.
- In principle . . "IF" a logic function contains both the true and inverse of state of a signal then that signal can be ignored. for example :-

"A" + (NOT "A") = "1" also "B" & "1" = "B"

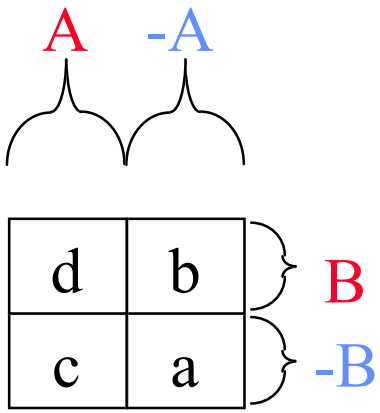
hence: (B&A)+(B&-A) = B&(A + -A) = B&1=B

Logic Functions.

- The Karnaugh Map.
- The is a popular graphical method that allows information to be taken from a Truth table and simplified or Minimized diagrammatically.
- The Truth table is re-drawn so logic relationships can be easily identified.
- The table ensures that adjacent squares only differ by ONE variable.
- All the table edges a wrap round to their opposite side.

Logic Functions.

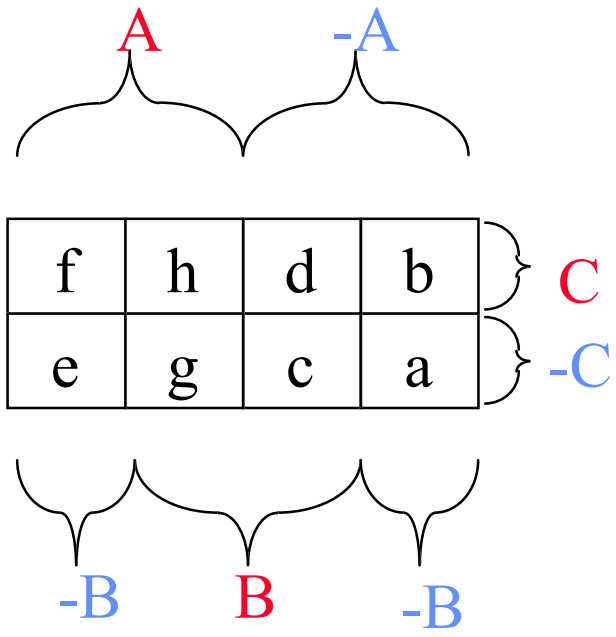
A	B	Output
0	0	a
0	1	b
1	0	c
1	1	d



How a Two variable table is presented.

Logic Functions.

A	B	C	Output
0	0	0	a
0	0	1	b
0	1	0	c
0	1	1	d
1	0	0	e
1	0	1	f
1	1	0	g
1	1	1	h



How a Three variable table is presented.

Logic Functions.

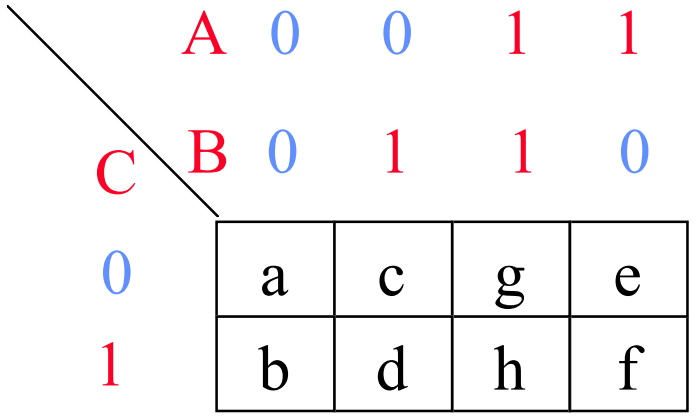
A	B	C	Output
0	0	0	a
0	0	1	b
0	1	0	c
0	1	1	d
1	0	0	e
1	0	1	f
1	1	0	g
1	1	1	h

	-A -B	-A B	A B	A -B
-C	a	c	g	e
C	b	d	h	f

Alternative Three variable presentation.

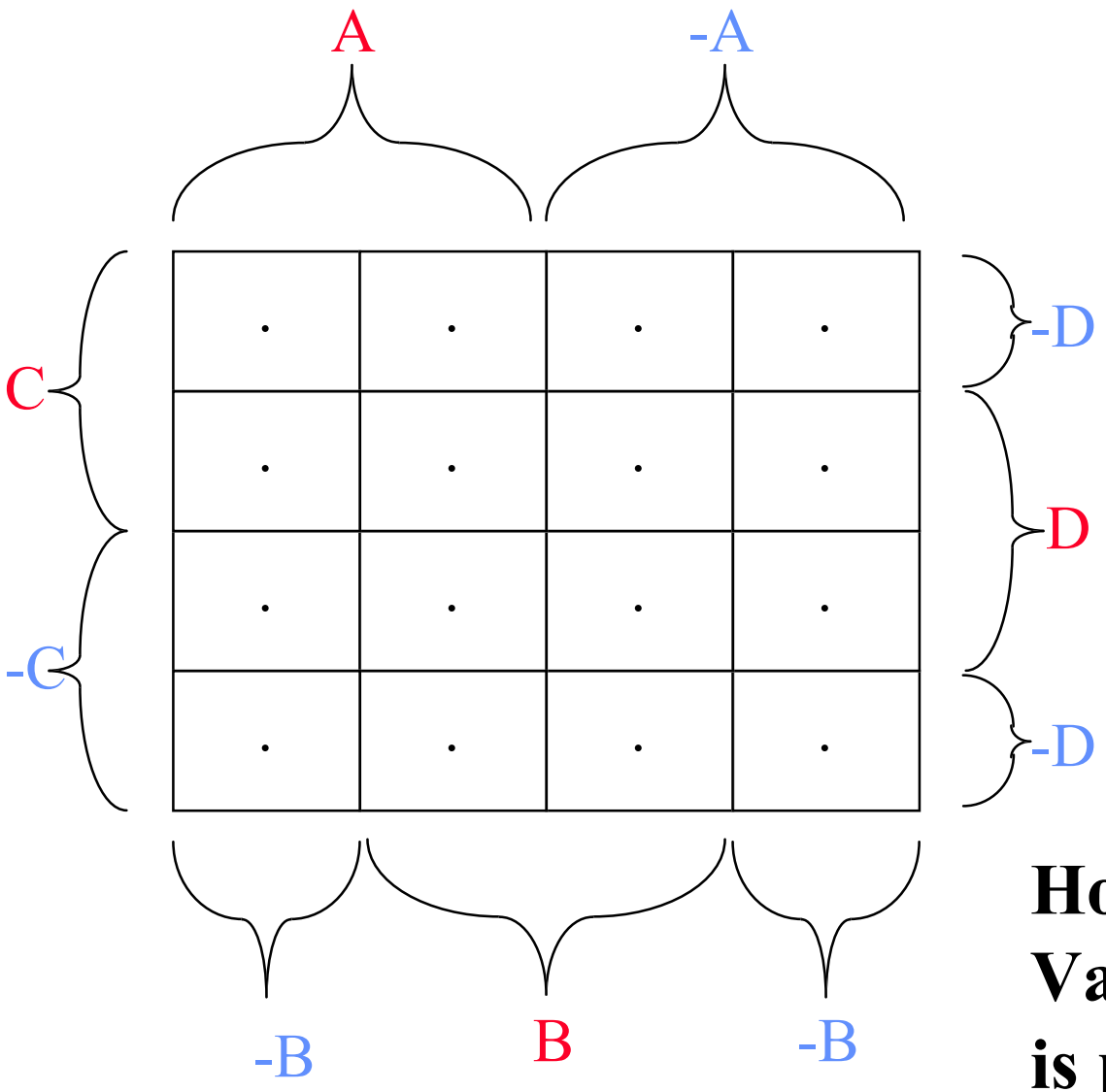
Logic Functions.

A	B	C	Output
0	0	0	a
0	0	1	b
0	1	0	c
0	1	1	d
1	0	0	e
1	0	1	f
1	1	0	g
1	1	1	h



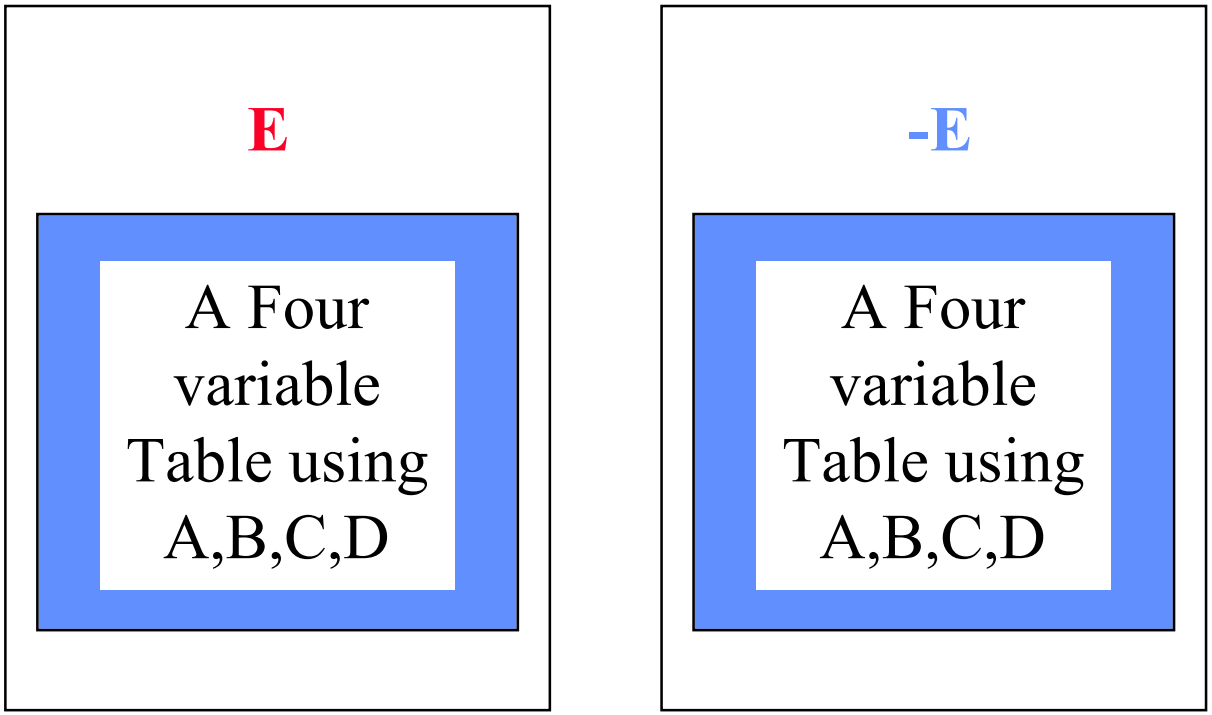
Alternative Three variable presentation.

Logic Functions.



How a Four Variable table is presented.

Logic Functions.



How a Five Variable table is presented.

Note: At this stage we are viewing a 3 dimensional structure and some edge relationships can become difficult to visualize. Five plus variables we use an alternative methods to resolve.

Logic Functions.

- The Karnaugh Map.
- How do we use the map.
- Step One: Taking the results directly from the truth table we place the 1's and 0's into the map.
- Step Two: We draw a ring around every (power of two) adjacent boxes that have a logic one in them.
- Step Three: We write down the equations of every ringed block plus any boxes that have not been included in the ringing process.
- Process complete (Circuit minimized)

Logic Functions.

A	B	C	Output
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

The equations for this truth table are:-

-A-B-C +

-A-BC +

-AB-C +

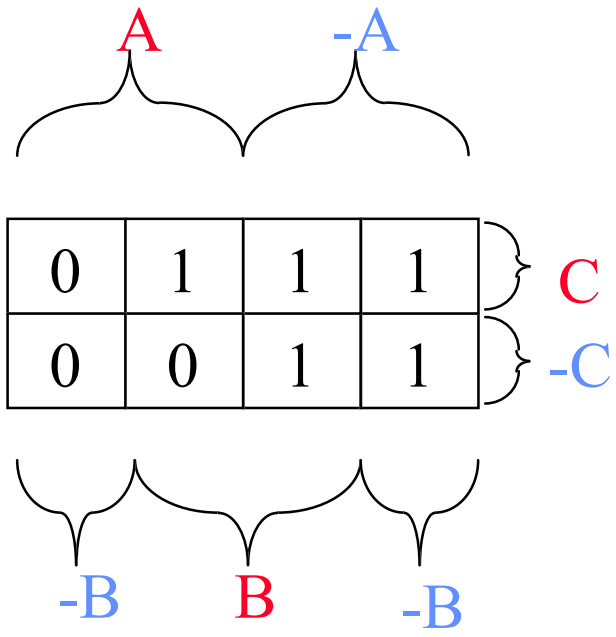
-ABC +

ABC

Truth table with equation description.

Logic Functions.

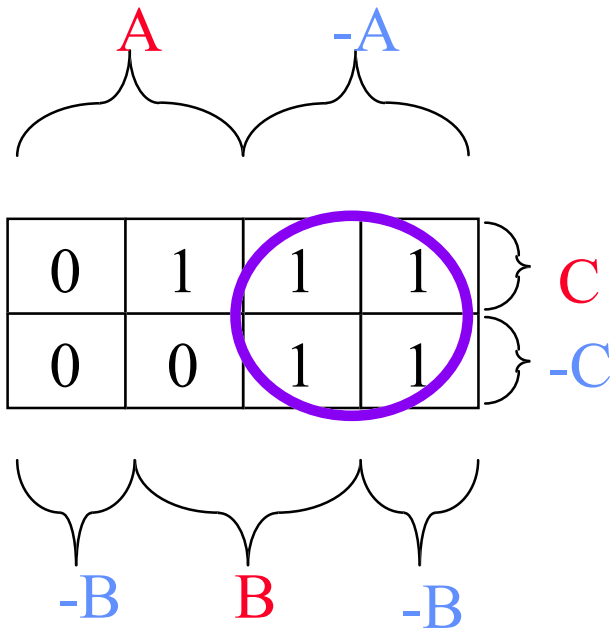
A	B	C	Output
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



Truth table to Map.

Logic Functions.

A	B	C	Output
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

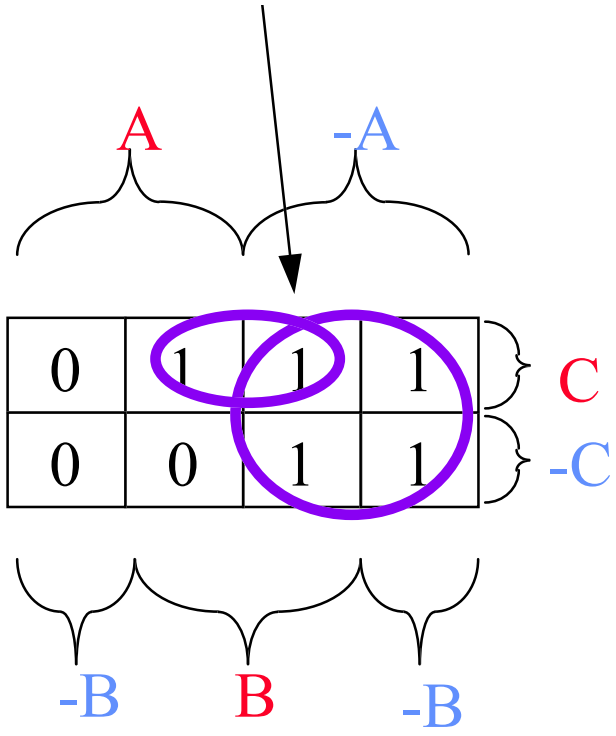


The ringing process.

Logic Functions.

A	B	C	Output
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Note it is acceptable to ring the same **1** more than once.



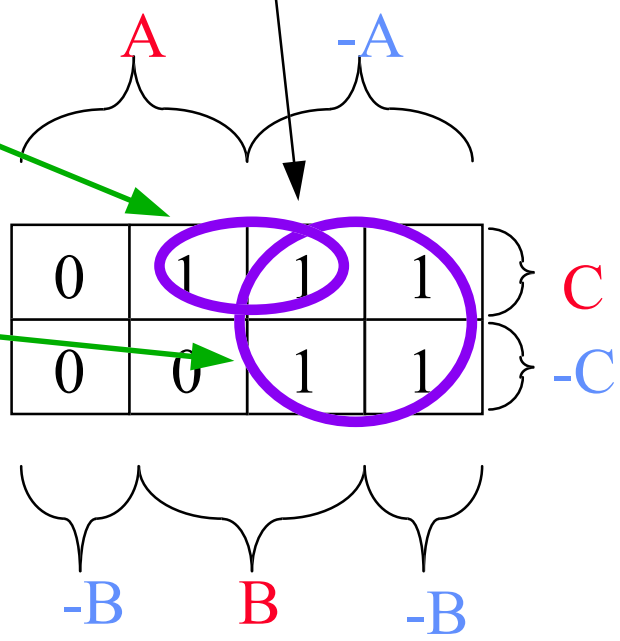
The ringing process.

Logic Functions.

This ring is uniquely in **C** and **B** .
So its equation is **BC**

This ring is uniquely in **-A** only.
So its equation is **-A**

Note it is acceptable to ring the same **1** more than once.



The overall equation for this truth table is:-

-A + **BC**.

The Equation process.

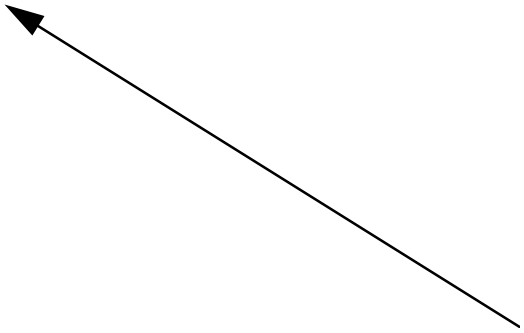
Logic Functions.

- Practical example using a Karnaugh Map.
- Design a circuit that will add two binary digits together.
- Stage 1 Produce Truth table of design

Logic Functions.

Basic example of an Adder Truth Table

Cin	A	B	Sum	Cout
	0	0		
	0	1		
	1	0		
	1	1		



Now fill in the Outputs

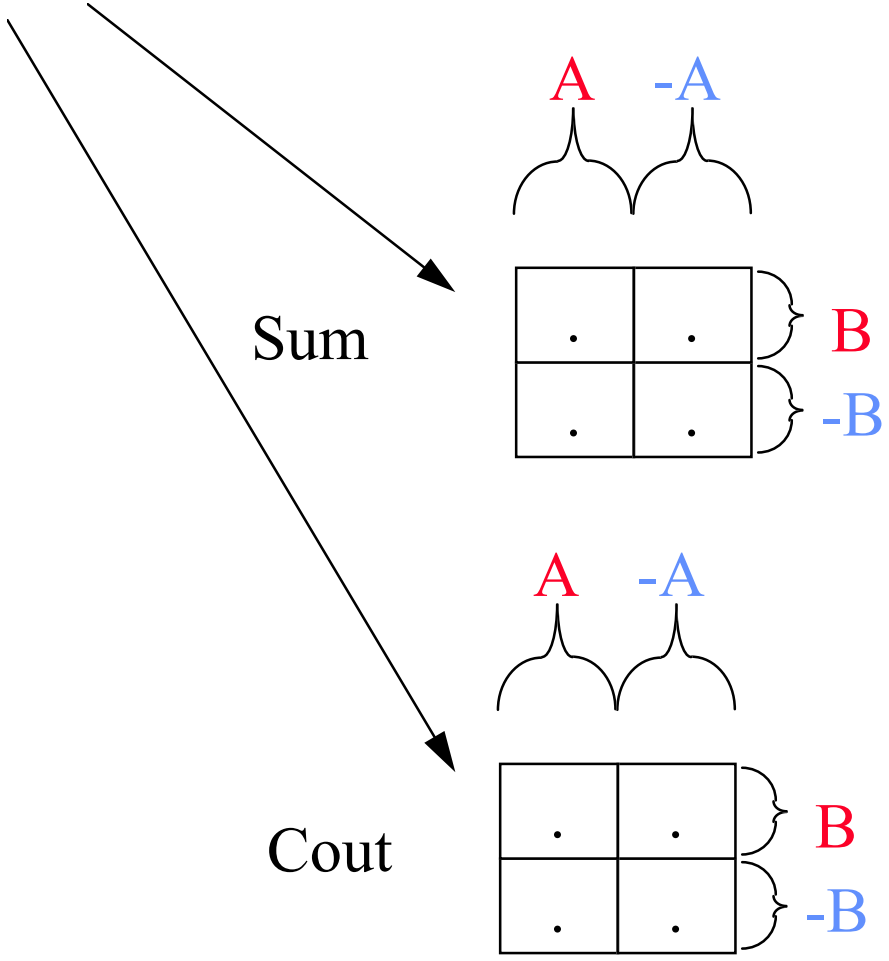
Logic Functions.

- Practical example using a Karnaugh Map.
- Design a circuit that will add two binary digits together (A Half Adder).
- Stage 1 Produce Truth table of design
- Stage 2 Produce Map of design

Logic Functions.

Now fill in the Truth Tables.

Cin	A	B	Sum	Cout
	0	0	0	0
	0	1	1	0
	1	0	1	0
	1	1	0	1

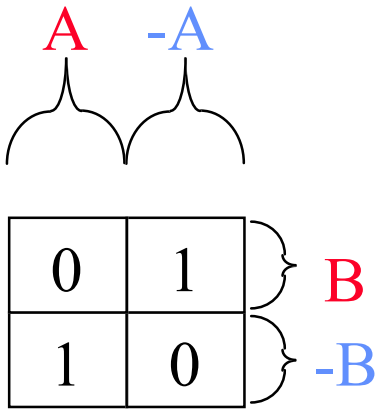


Logic Functions.

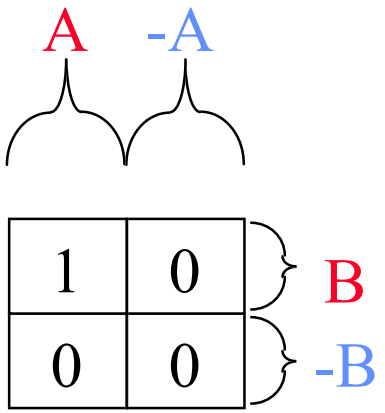
Ring the Equations.

Cin	A	B	Sum	Cout
	0	0	0	0
	0	1	1	0
	1	0	1	0
	1	1	0	1

Sum



Cout



Logic Functions.

- Practical example using a Karnaugh Map.
- Design a circuit that will add two binary digits together.
- Stage 1 Produce Truth table of design
- Stage 2 Produce Map of design
- Stage 3 Produce equations of design

Logic Functions.

Nothing Rings so :-

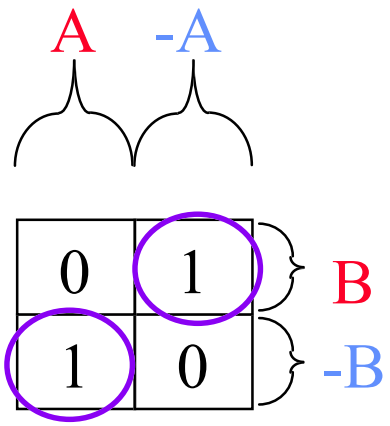
Sum = A-B + -AB

Count = AB

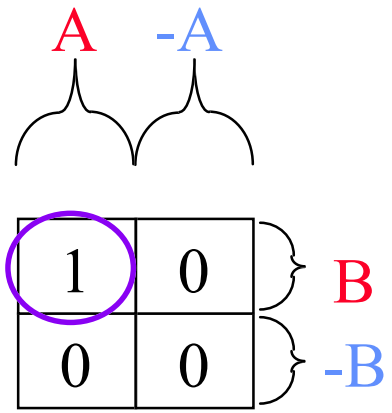
What we have designed here is called a Half Adder.

The Full Adder would need to include the **Carry In** bit in the calculation let us try the same procedure again.

Sum



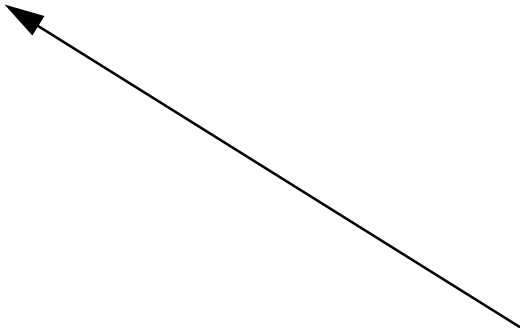
Cout



Logic Functions.

Basic example of an Full Adder Truth Table

Cin	A	B	Sum	Cout
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		



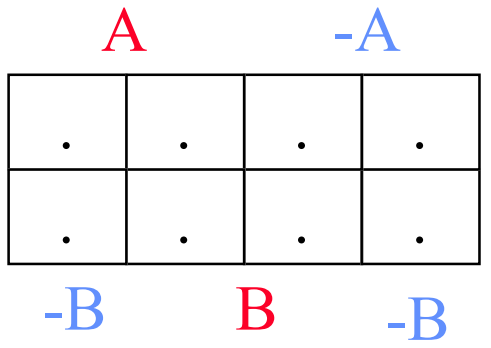
Now fill in the Outputs

Logic Functions.

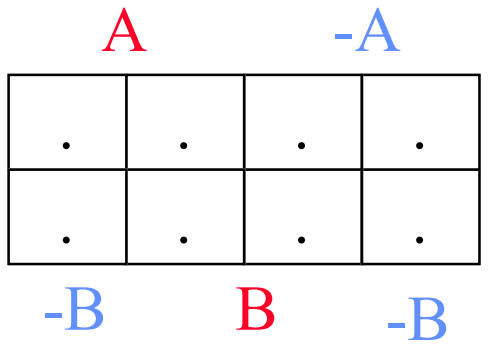
Now fill in the Truth Tables.

Cin	A	B	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Sum



Cout



Logic Functions.

Ring the Equations.

Cin	A	B	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Sum

	A		-A	
Cin	0	1	0	1
-Cin	1	0	1	0
	-B	B	-B	

Cout

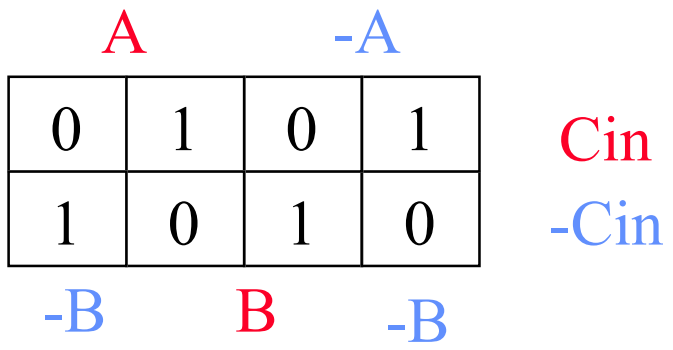
	A		-A	
Cin	1	1	1	0
-Cin	0	1	0	0
	-B	B	-B	

Logic Functions.

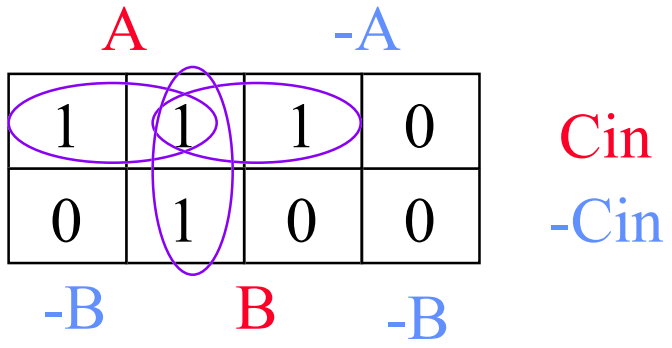
Describe the Equations.

Cin	A	B	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Sum



Cout



Logic Functions.

Describe the Equations.

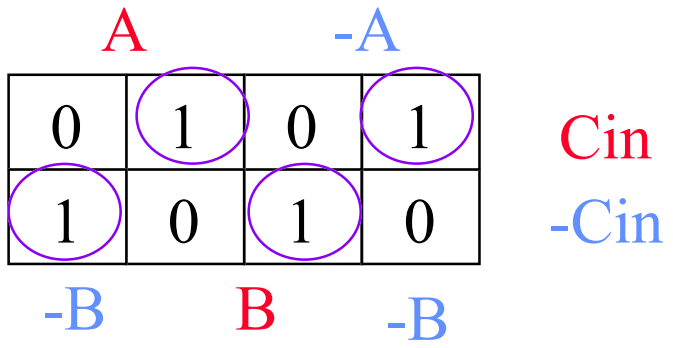
Nothing Rings so :-

Sum = A-B-Cin + ABCin +
-AB-Cin + -A-BCin

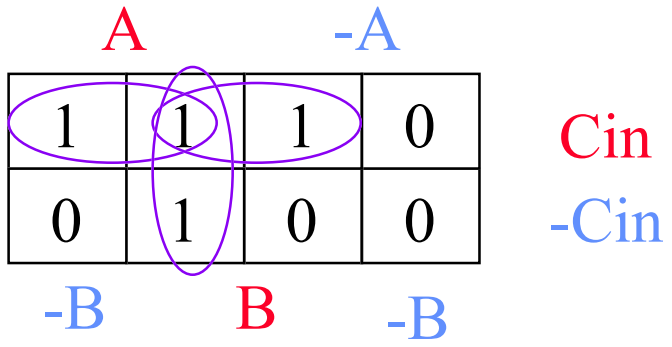
Count = AB + ACin + BCin

What we have now designed a Full Adder.

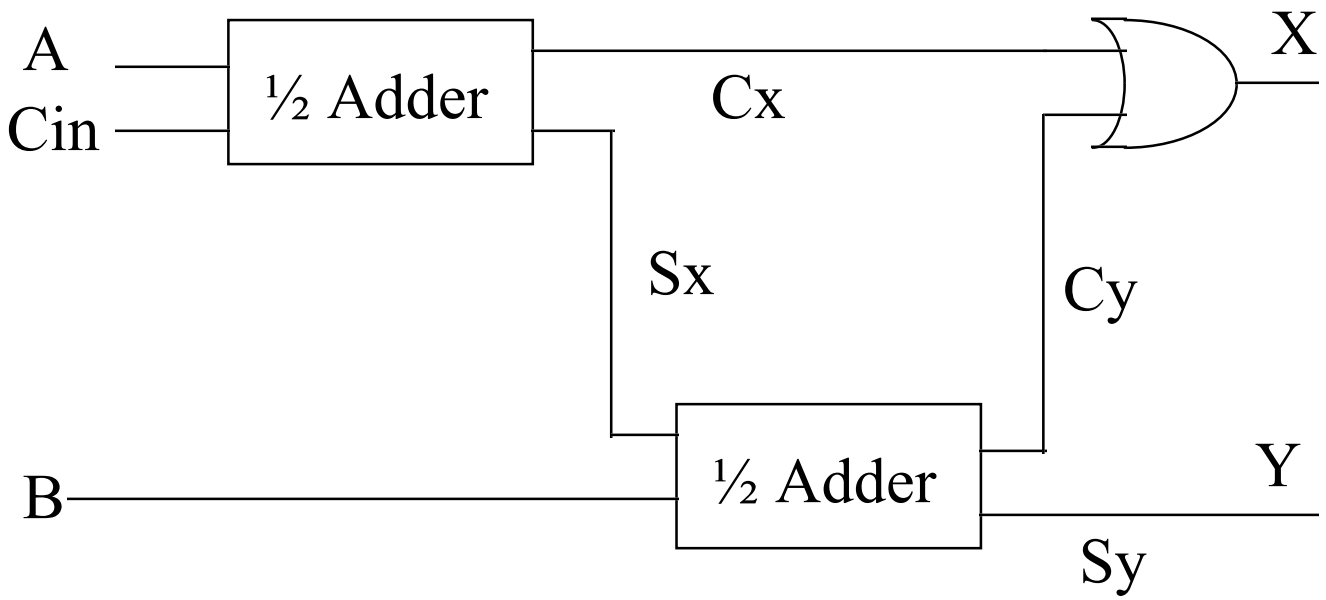
Sum



Cout



Logic Functions.



Question: What would we get if we connected our half adders up as shown above ?

Logic Functions.

Complete the Truth tables.

Cin	A	B	Sum	Cout
0	0	0	.	.
0	0	1	.	.
0	1	0	.	.
0	1	1	.	.
1	0	0	.	.
1	0	1	.	.
1	1	0	.	.
1	1	1	.	.

Cin+A		Sx+B		Cx+Cy
Sx	Cx	Sy	Cy	Cout
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.

Logic Functions.

Cin	A	B	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Cin+A		Sx+B		Cx+Cy
Sx	Cx	Sy	Cy	Cout
0	0	0	0	0
0	0	1	0	0
1	0	1	0	0
1	0	0	1	1
1	0	1	0	0
1	0	0	1	1
0	1	0	0	1
0	1	1	0	1

Logic Functions.

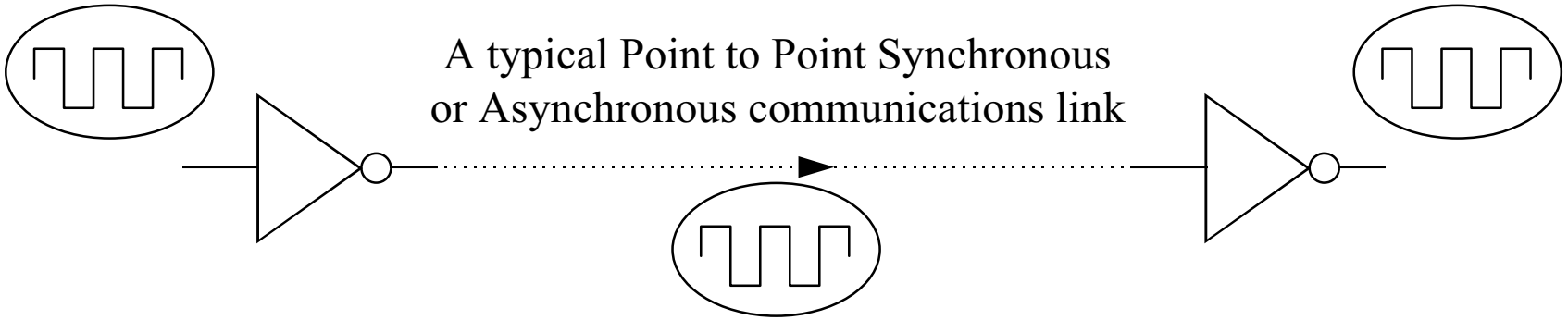
Cin	A	B	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Cin+A		Sx+B		Cx+Cy
Sx	Cx	Sy	Cy	Cout
0	0	0	0	0
0	0	1	0	0
1	0	1	0	0
1	0	0	1	1
1	0	1	0	0
1	0	0	1	1
0	1	0	0	1
0	1	1	0	1

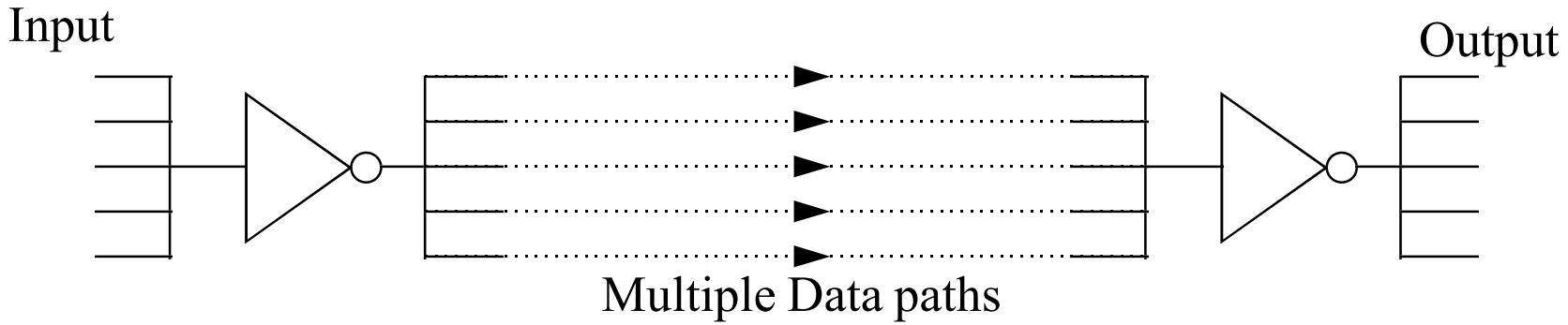
Conclusion: Two Half Adders plus one OR gate = a Full adder.

Communications

Communications.



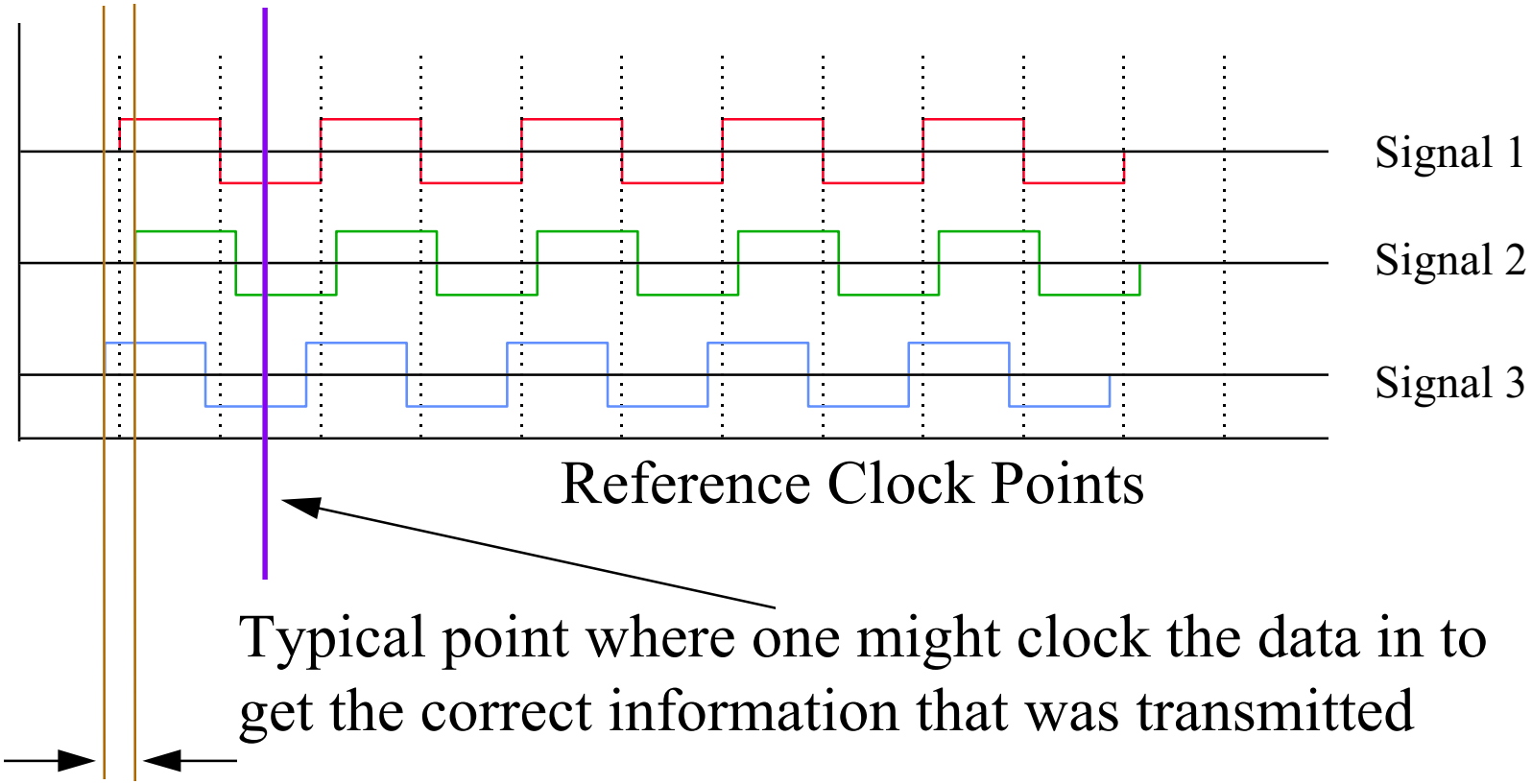
To transfer more data faster all we need to do in increase the number of data paths.



However if we use multiple data paths what happens if additional delays or lengths are added to some of the data paths.

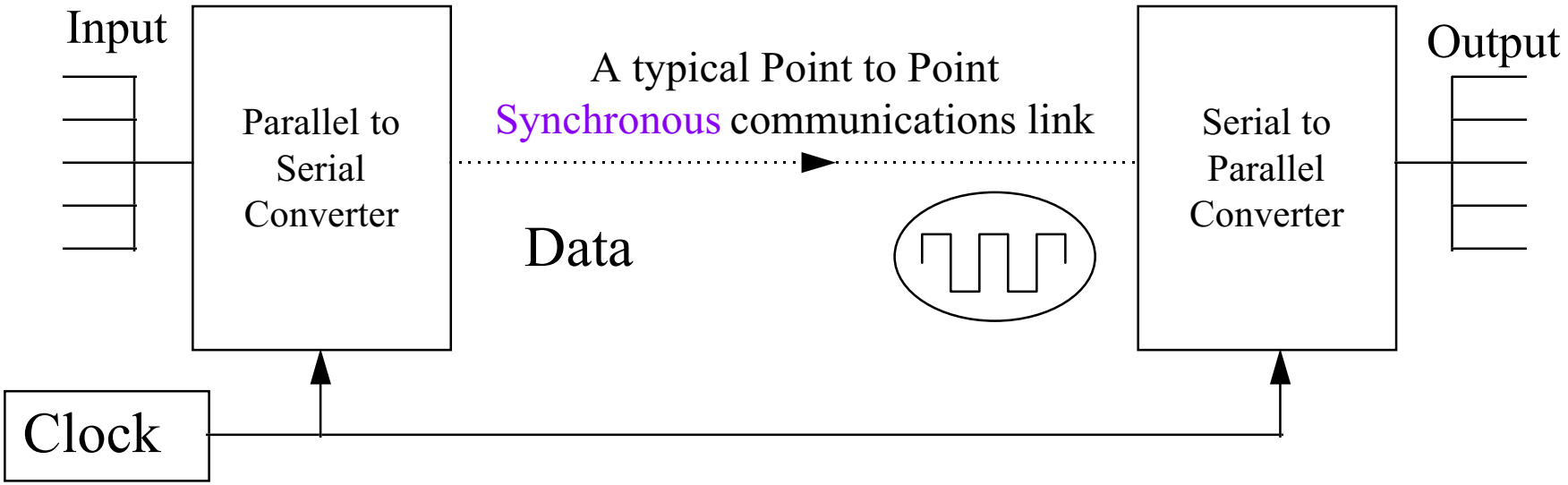
Remember that most logic systems need their data to be synchronous.

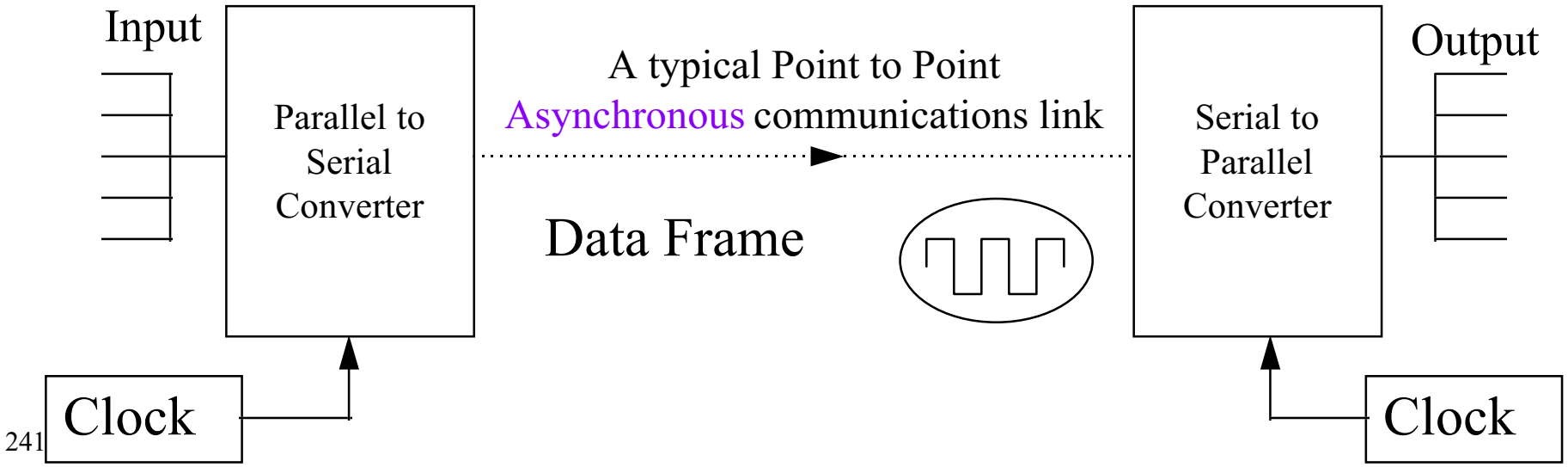
Communications.



Note as the frequency get higher (pulse width shorter) then resolving data skew problems becomes more significant.

Communications.





Communications.

- The (Parallel to Serial) and (Serial to Parallel) converters are often known as a **U**niversal **A**synchronous **R**eceiver **T**ransmitter **UART**.
- Chips that also permit Synchronous as well Asynchronous Communications are called a **U**niversal **S**ynchronous **A**synchronous **R**eceiver **T**ransmitter **USART**.

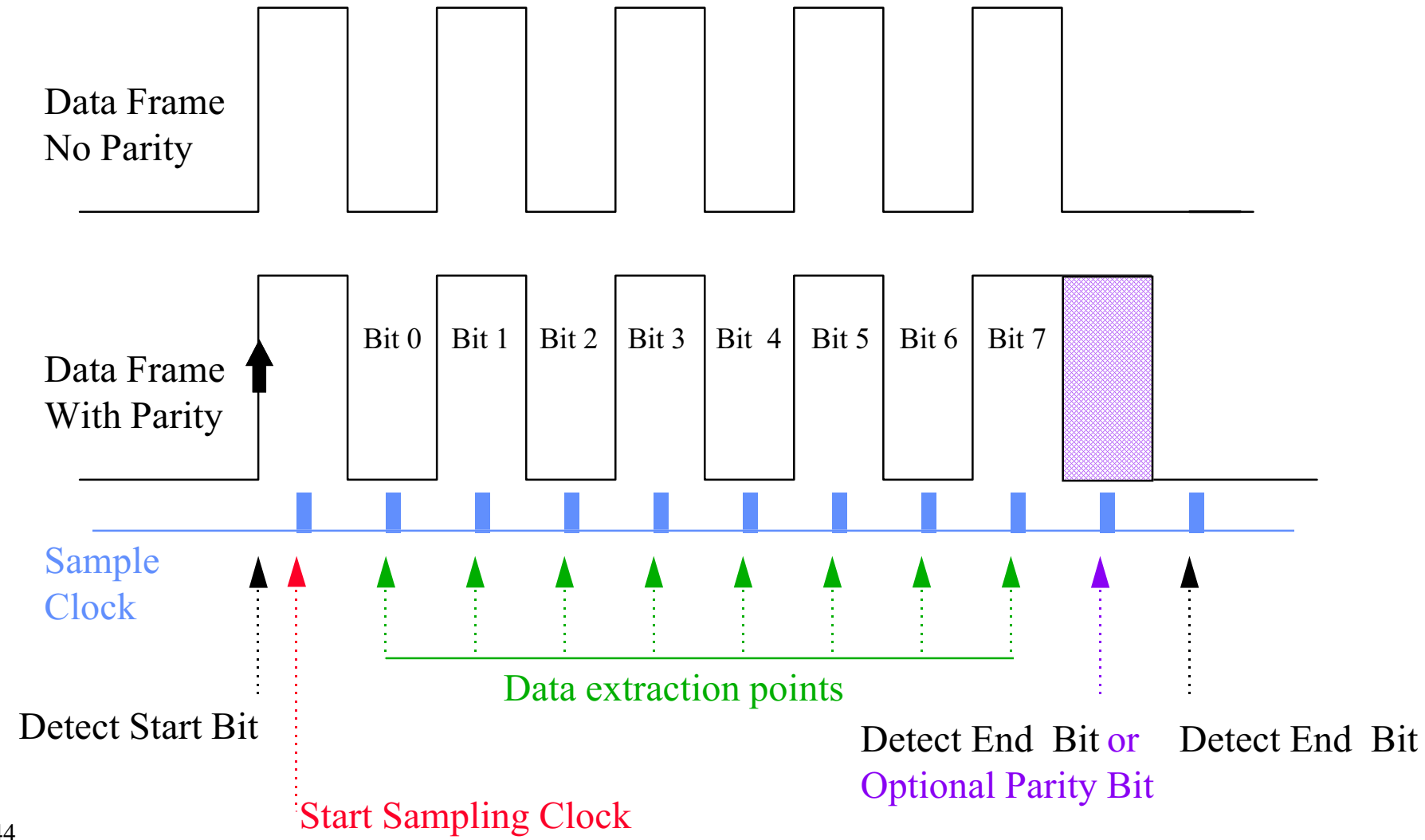
- However with the Asynchronous communications how do we know where the Data Frame starts.

Communications.

- Initially we need to ensure that the clocks are running at the same frequency.
- The Clock frequency is typically 8 or 16 times the Data transfer rate frequency.
- The UART will monitor the Data line looking for a valid start condition level.
- Once the start condition is located then at regular time intervals the Data bits are extracted.
- Finally the UART checks the end condition has arrived.
- The Status of the transfer is finally identified.

Communications.

Serial protocol.



Communications.

- With Serial communications we need to specify the following parameters :-
- 1) The The Baud Rate (Transitions per second)
  - Typically rates vary from 75 to 115000 baud.
  - Popular values are 1200, 2400, 4800, 9600 28K, 56K.
- 2) If parity required :-
  - Odd , Even, None.
- 3) Number of Stop Bits
  - One, One and a Half or Two.
- 4) Number of Data Bits
  - Typically Five, Seven or Eight bits.

Communications.

- Parity is a simple method for checking data.
- The system counts how many binary digits exist in the data section (the Bit Count) and :-
  - If Parity is ODD
    - and Bit Count is ODD then Parity is set to Zero.
    - and Bit Count is EVEN then Parity is set to One.
  - If Parity is EVEN
    - and Bit Count is ODD then Parity is set to One.
    - and Bit Count is EVEN then Parity is set to Zero.
- Note. This system only offers a limited checking capability.

Communications.

- Example :-
- Assume following Settings :-

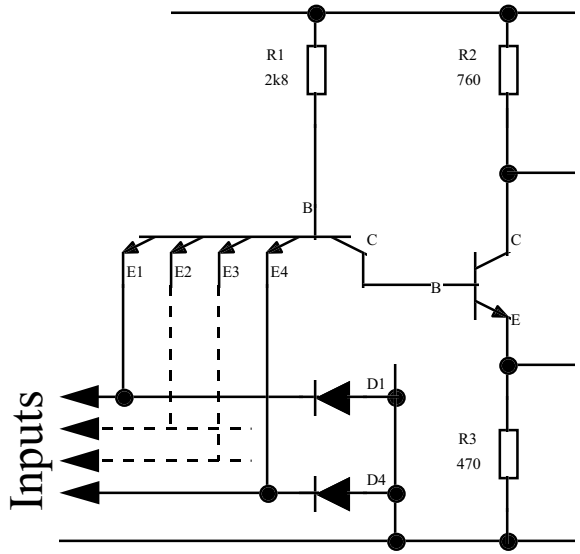
  9600 baud, 8 Bits, One Stop Bit, No Parity.
- The frame width will be :-
- One Start Bit + One Stop Bit + Eight Data Bits gives a total of Ten Bits wide.
- Number of frames per second = 9600/10 = 960 frames or 960 characters per second.
- **Note** BAUD and Bits per second may not always be the same because some transmission systems represent many bits as a single transition.

Device Interfacing

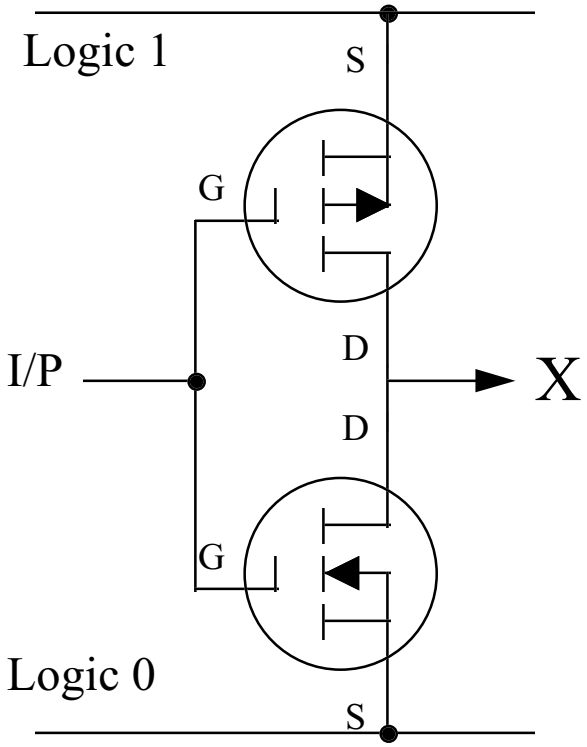
Logic Device Families

- DTL (Diode Transistor Logic) Package codes 8xx and 9xx.
- TTL (Transistor Transistor Logic) Package codes Commercial 74aaxx and Mil 54aaxx.
- CMOS (Complementary Metal Oxide Silicon) Package codes 4xxx , Low power, quite slow , Ultra ESD sensitive.
- ECL (Emitter coupled logic) High Speed but very high power requirements.

Logic Functions.



Typical TTL Gate
Front end of Circuit



Typical CMOS Gate
Inverter Circuit

TTL and CMOS Gates.

Logic Device Families

- DTL almost obsolete.
- TTL Less ESD sensitive, still quite popular.
- CMOS “B” series more reliable however being replaced with HC & HCT devices.
- ECL almost obsolete.
- FPLA , FPGA field programmable Logic and Gate arrays (design your own chip) this is the future direction for most development as most cost effective route.

Logic Device Families

- TTL family extension letter codes.
- S (Schottky) very fast devices.
- LS (Lower power Schottky)
- AS (Advanced Schottky)
- ALS (Advanced Lower power Schottky)
- H (High Speed)
- HC (High Speed CMOS)
- HCT (HC construction & TTL Compatible)
- Note: The numbers identify the logic function so that an LS00 is the same pinout and function as an HCT00 etc.

Logic Device Families

- DTL 0 to +6V operation for max. noise immunity, however normally runs at +5v.
- TTL 0 to +5V.
- CMOS “A” & “B” series 0 to +3 $\rightarrow$ +16V.
- FPLA , FPGA 0 to +5 or +3.3 or +1.x depends on type.
- Discrete logic, Resistor Diode logic voltages designer defined.

Logic Device Families

- DTL Pulses shorten as they propagate through the devices.
- TTL can be power hungry.
- CMOS ESD sensitivity & low speed.
- FPLA , FPGA may be complex to programme.
- Discrete logic, Resistor Diode logic specialist applications only.

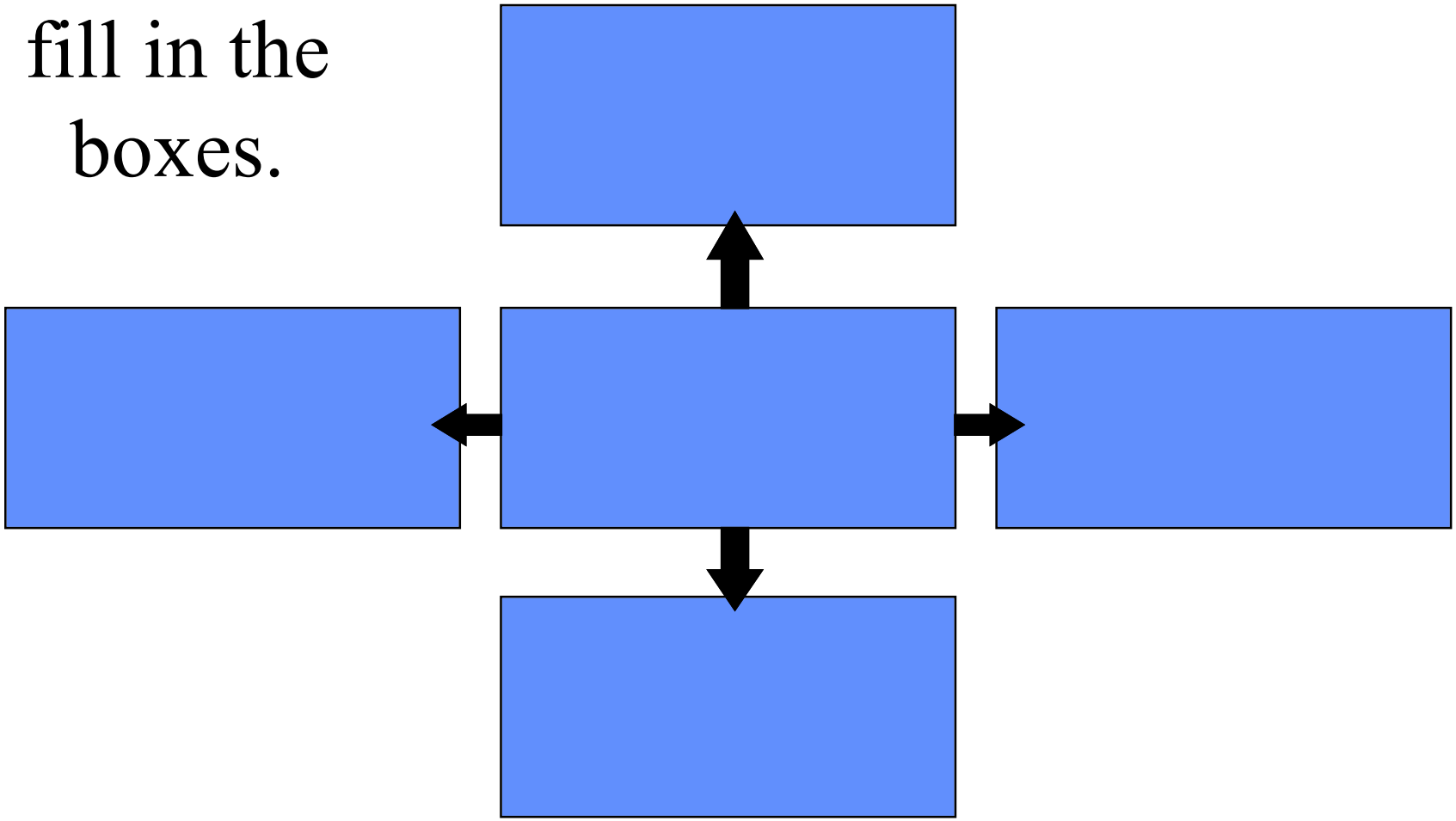
Computers

Computers.

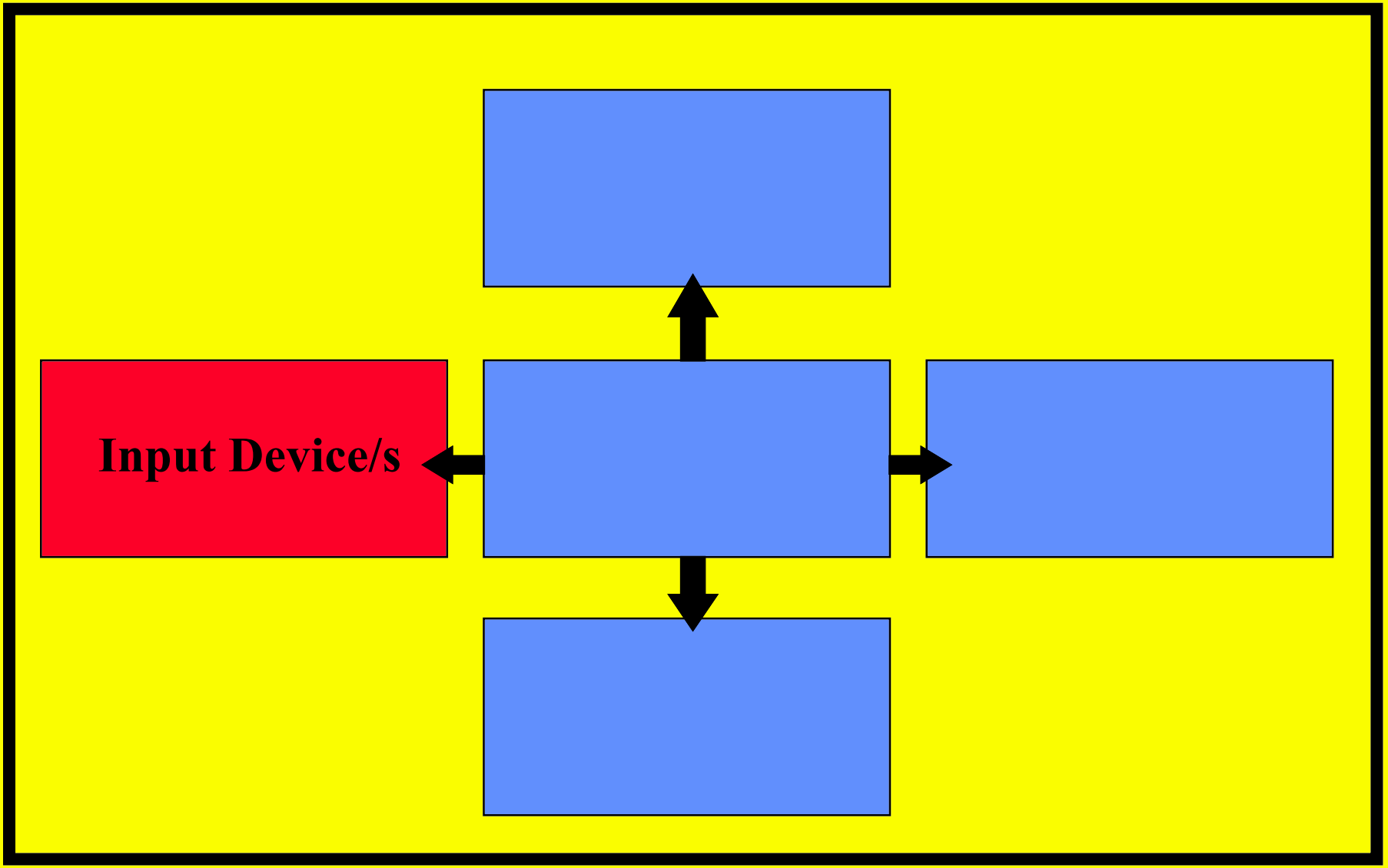
- What are Computers ?
- Historically it meant a person who performed calculations.
- Nowadays it means a machine that performs calculations.
- Let us consider what is the Generic Computer.
  Definition: *Generic = belonging to, characteristic of , class or genus "A **Hoover** is vacuum cleaner but not all vacuum cleaners are made by **Hoover** however we may still call it a **Hoover**"*

What is a Computer made from ?

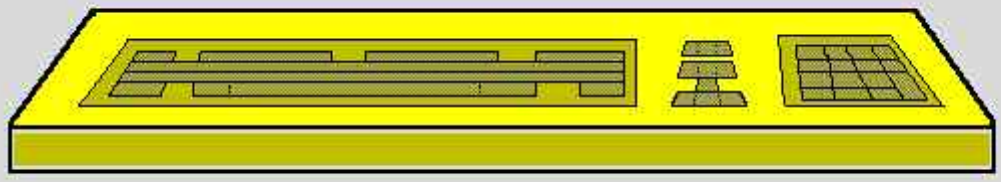
Let us try to fill in the boxes.



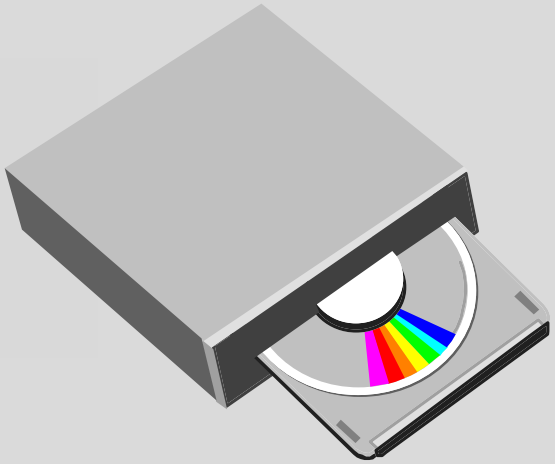
What is a Computer made from ?



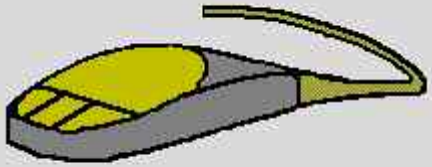
What are Input Devices ?



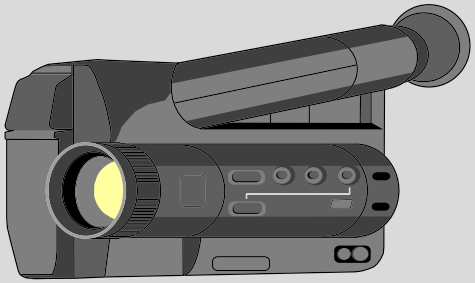
Keyboard



CD or DVD

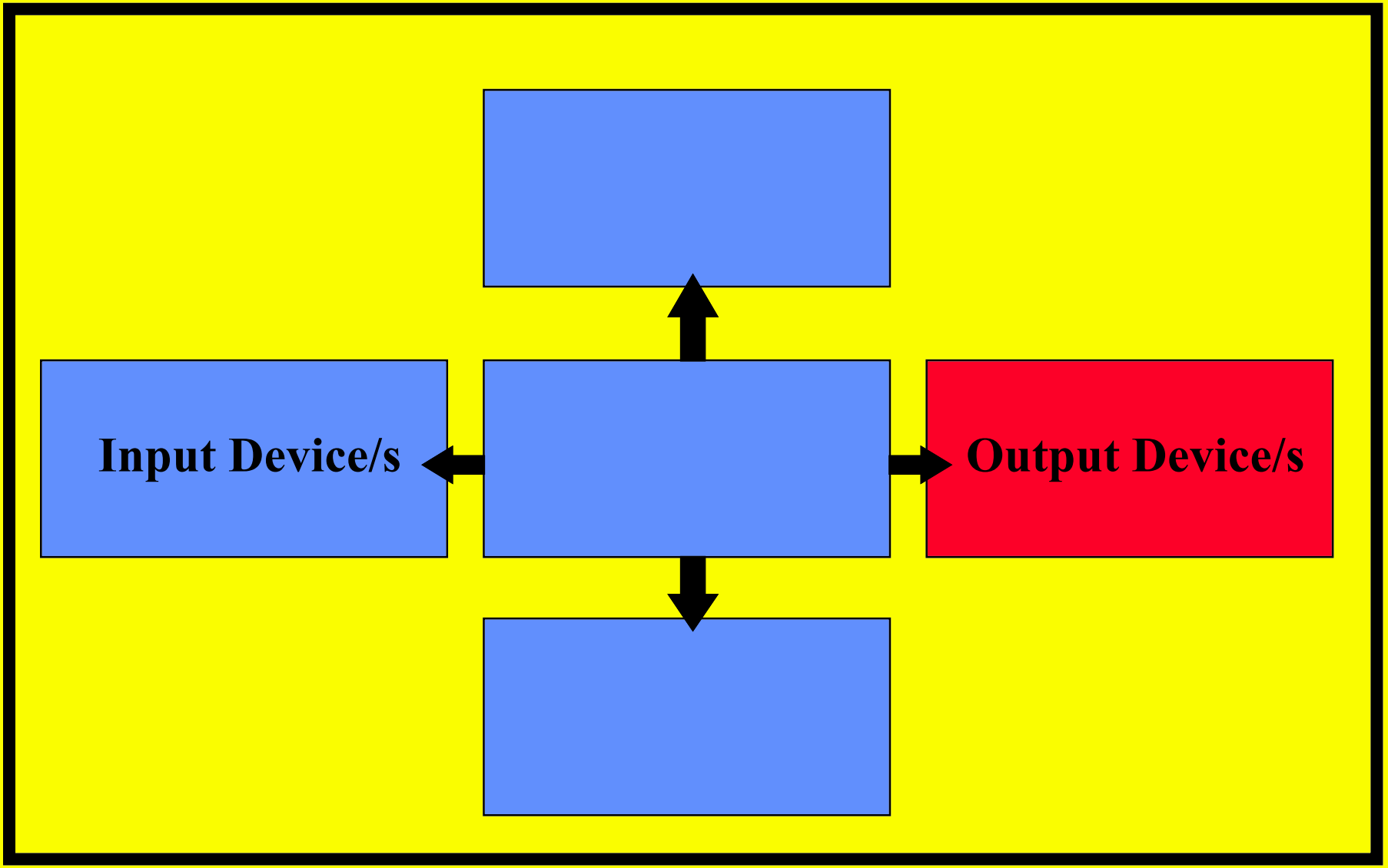


Mouse

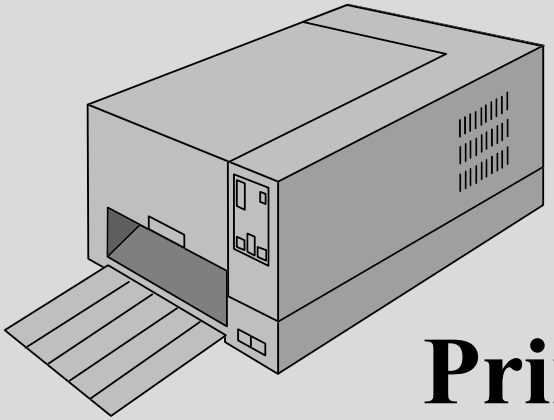


Camera

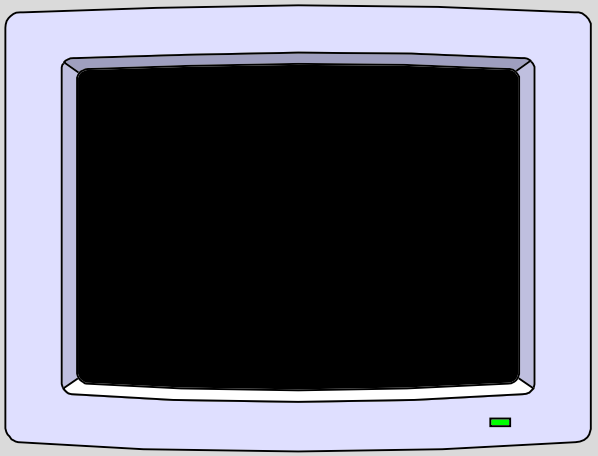
What is a Computer made from ?



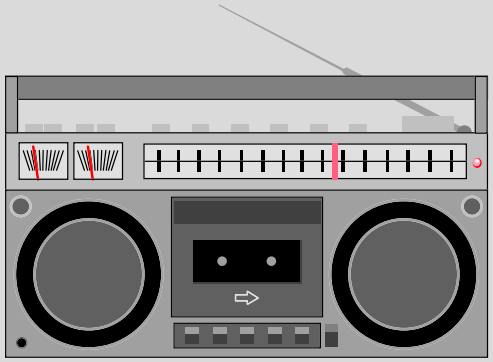
What are Output Devices ?



Printers

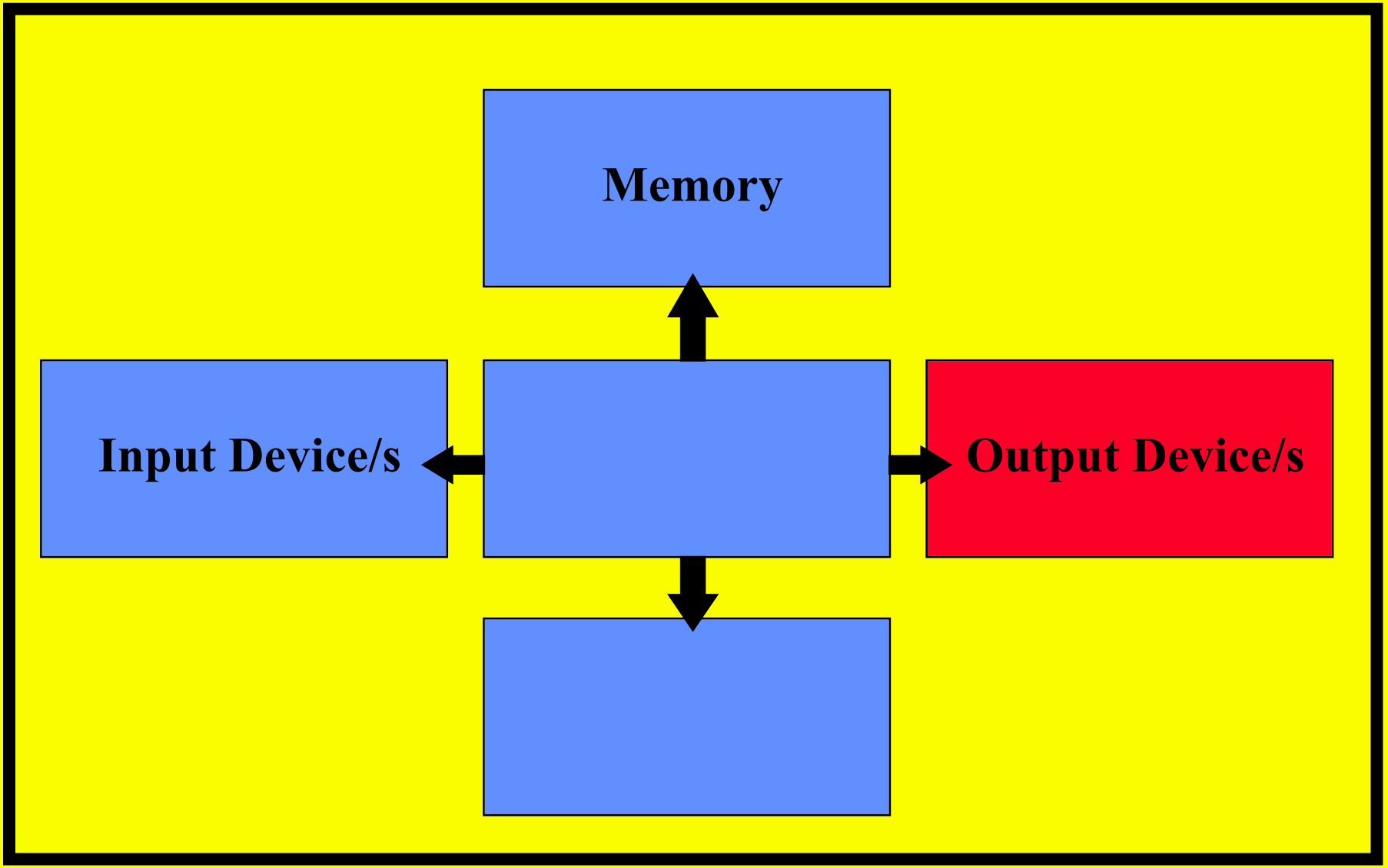


Monitors and Display Devices

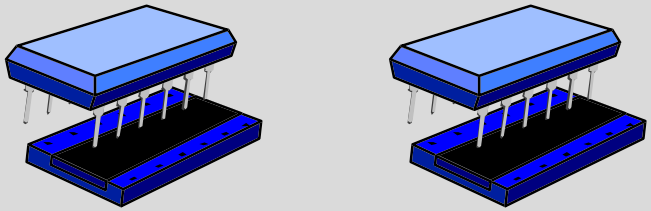


Speakers

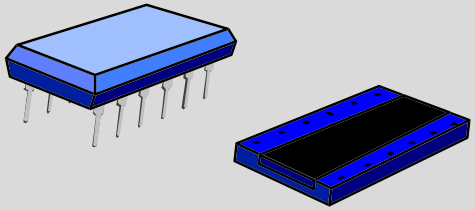
What is a Computer made from ?

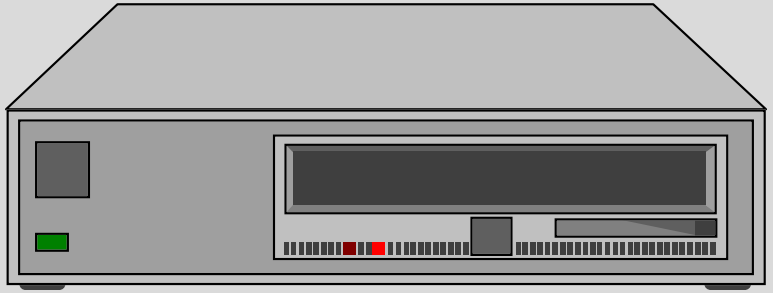


What are Memory Devices ?



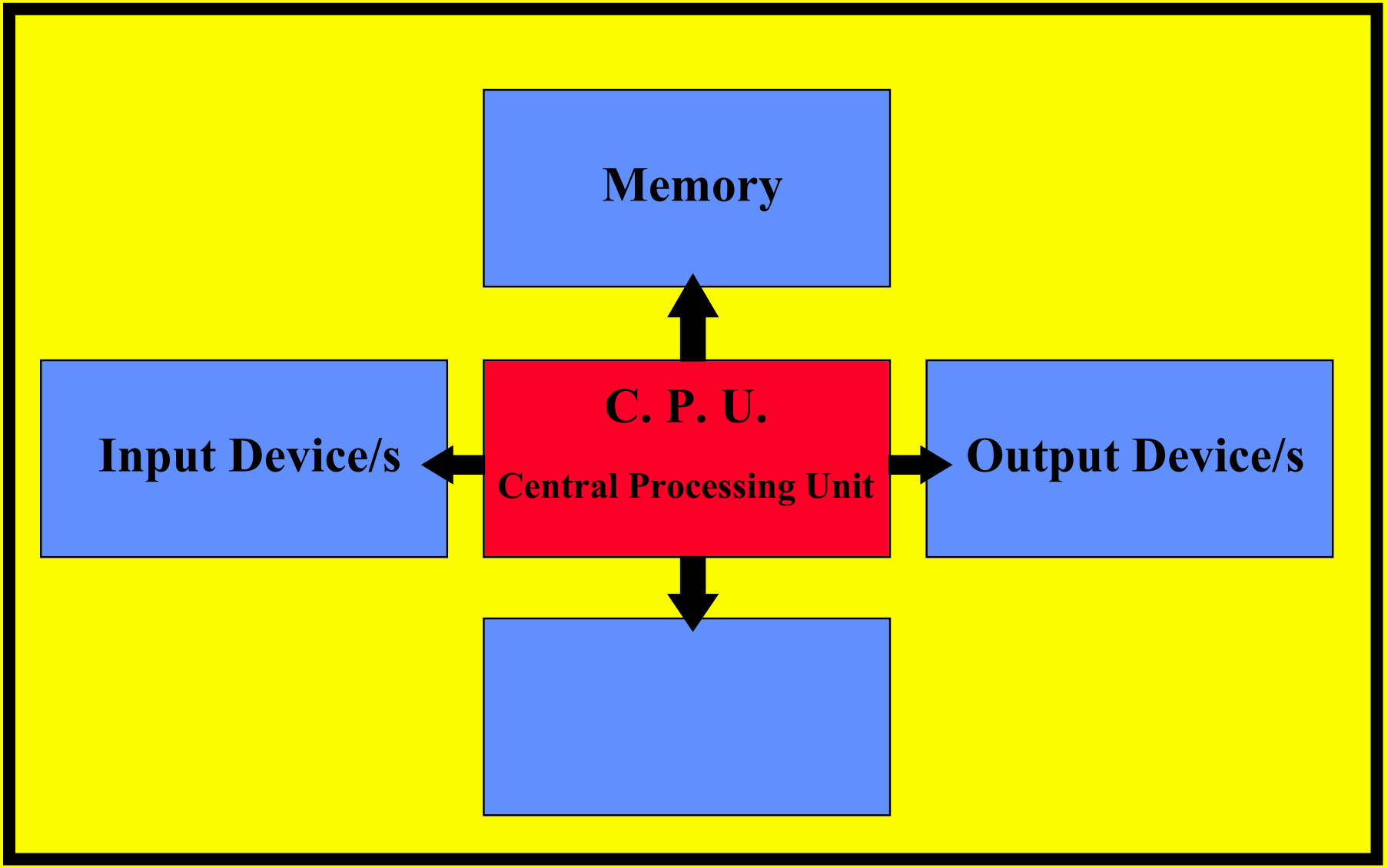
RAM , ROM , EPROM chips



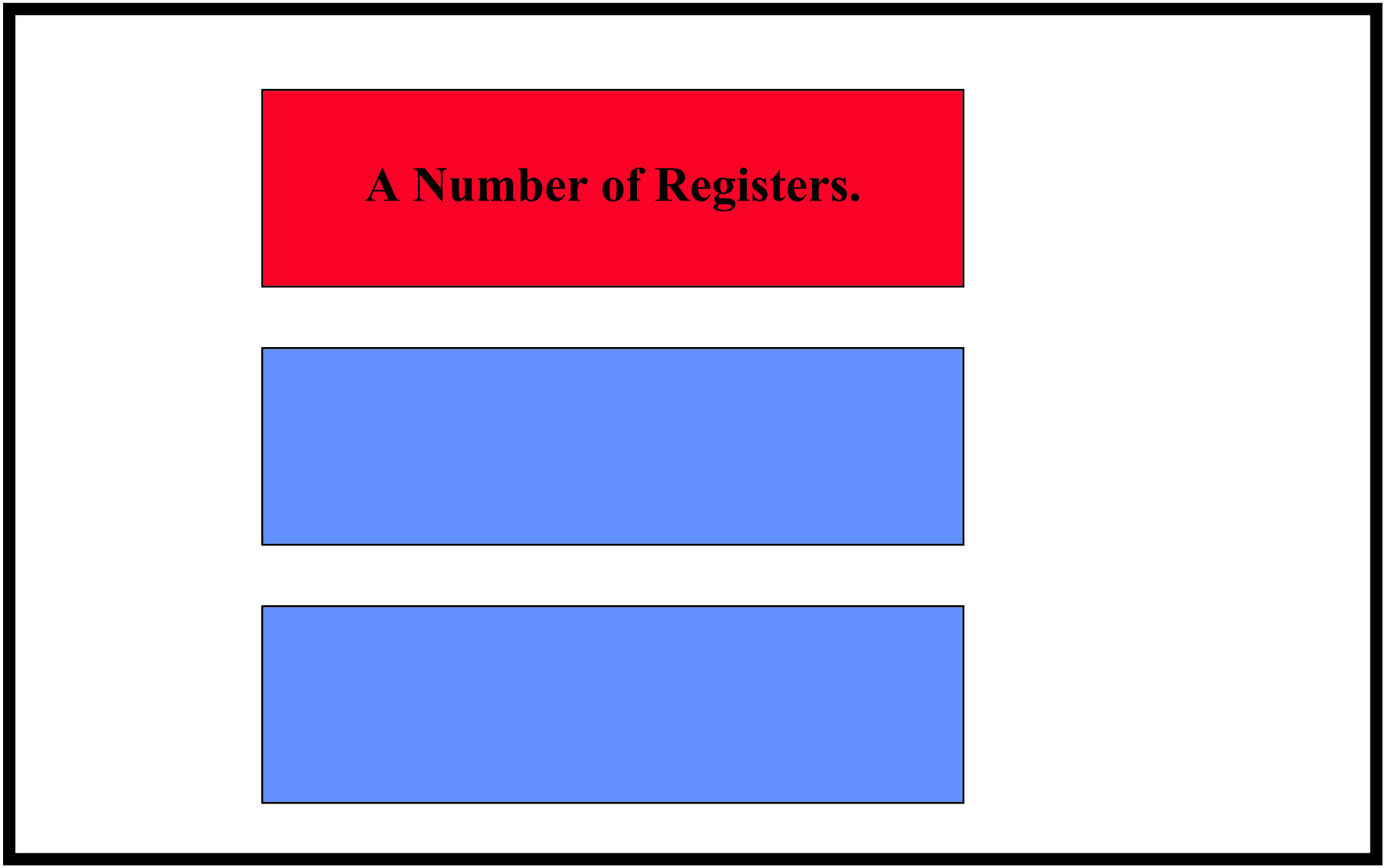


The Disk Drive

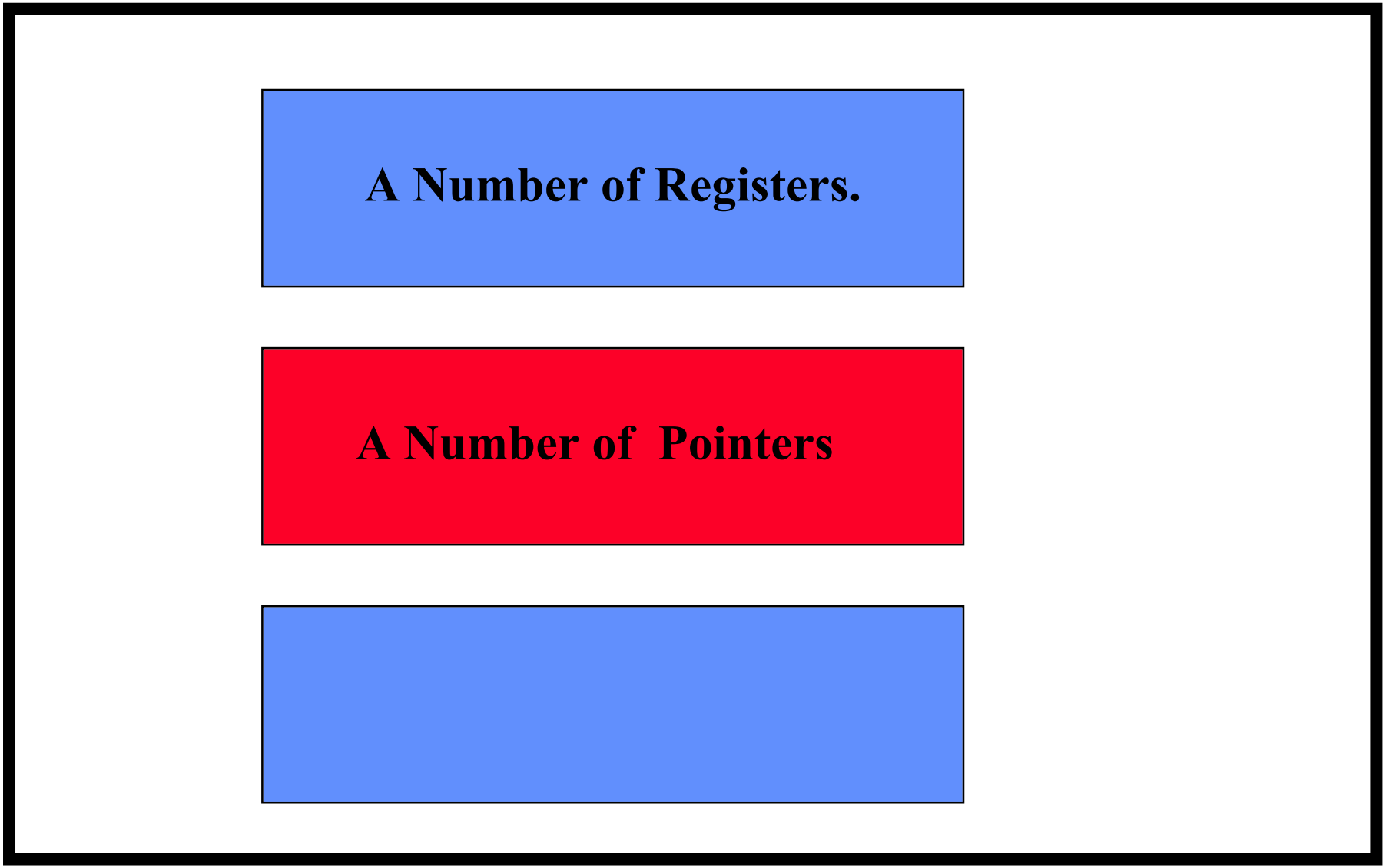
What is a Computer made from ?



What is in a C. P. U. ?



What is in a C. P. U. ?



What is in a C. P. U. ?

A Number of Registers.

A Number of Pointers

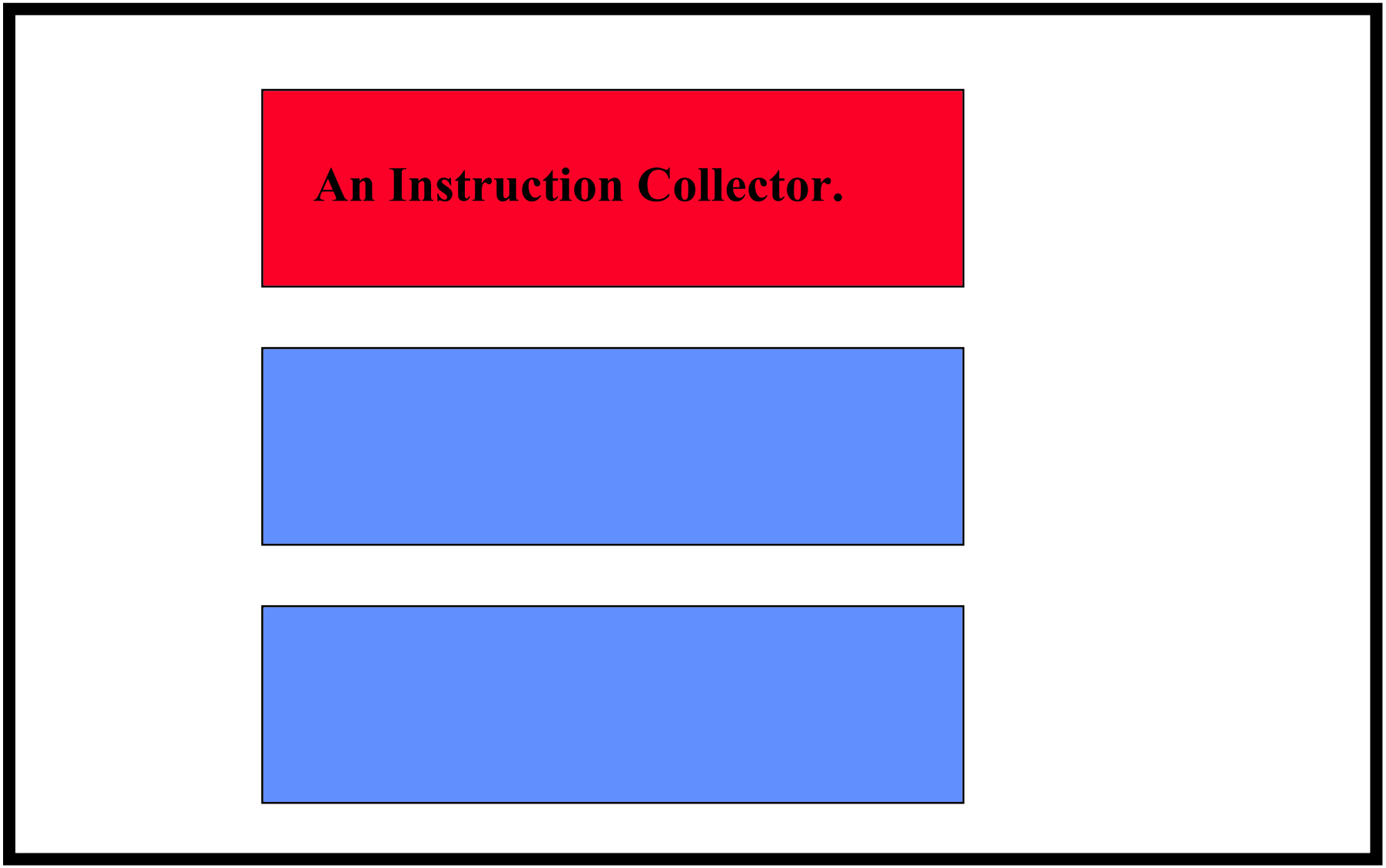
A Number of Information Markers

What is in a C. P. U. ?

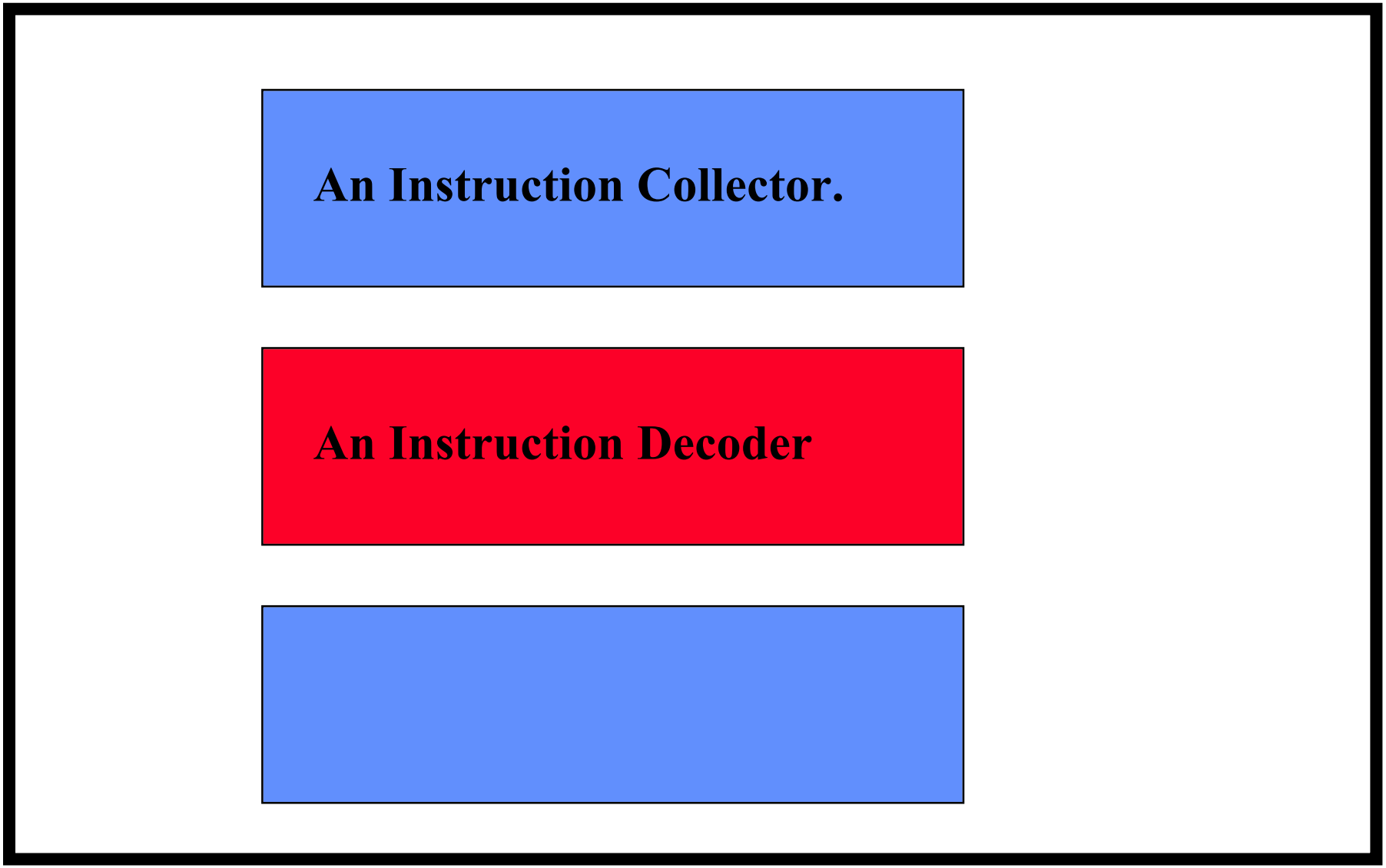
AND

ALSO

What is in a C. P. U. ?



What is in a C. P. U. ?



What is in a C. P. U. ?

An Instruction Collector.

An Instruction Decoder

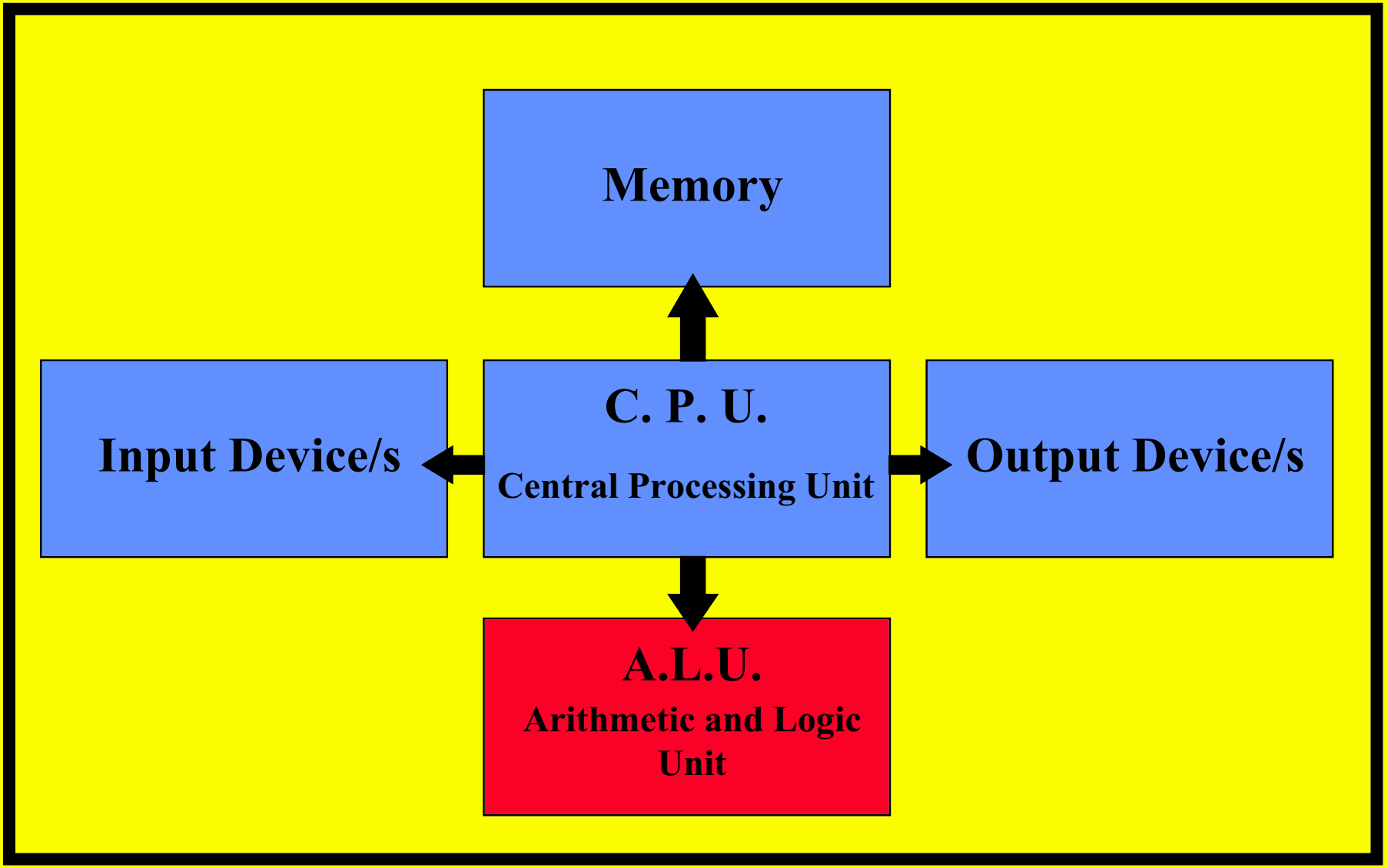
An Information Transfer Controller and Clock.

What is in a C. P. U. ?

Summary

- An Instruction Collector.
- A Number of Registers.
- An Instruction Decoder
- A Number of Pointers
- An Information Transfer Controller and Clock.
- A Number of Information Markers

What is a Computer made from ?



What Does the A. L. U. do ?

- It processes Numbers.
- However unlike Humans the number of columns that it can deal with is both limited and finite.
- The number of columns is used to describe the machine type i.e.
- 1 Bit, 4 Bit, 8 Bit, 16 Bit, 32 Bit or 64 Bit.

What Does the A. L. U. do ?

It Adds Numbers.

What Does the A. L. U. do ?

It Adds Numbers.

It Makes Numbers Negative.

What Does the A. L. U. do ?

- It Adds Numbers.
- It Makes Numbers Negative.
- It Shifts and Rotates Digits in Numbers.
-
-

What Does the A. L. U. do ?

- It Adds Numbers.
- It Makes Numbers Negative.
- It Shifts and Rotates Digits in Numbers.
- It Compares and Tests Numbers.
-

What Does the A. L. U. do ?

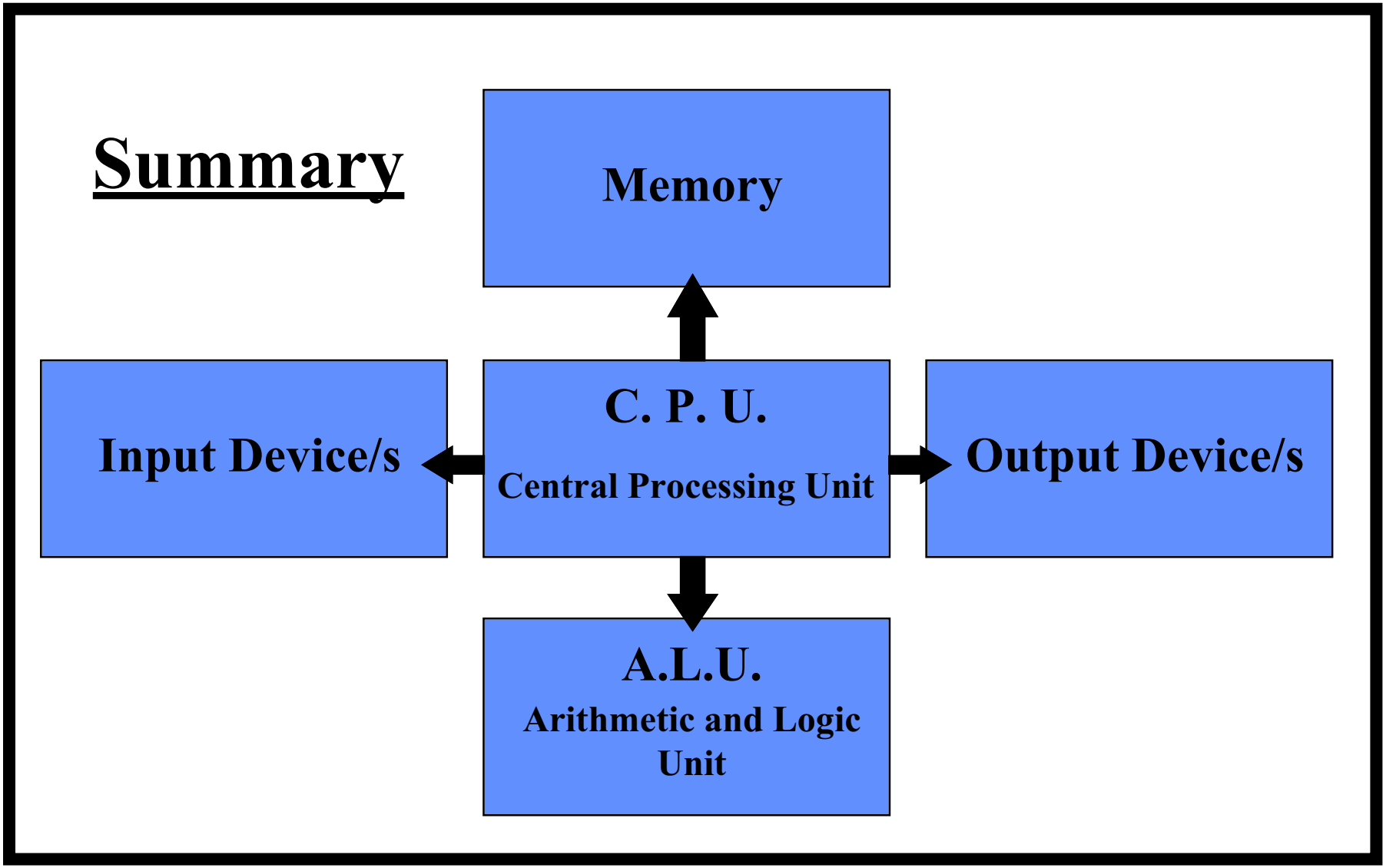
- It Adds Numbers.
- It Makes Numbers Negative.
- It Shifts and Rotates Digits in Numbers.
- It Compares and Tests Numbers.
- It performs Logic functions on Numbers.

What Does the A. L. U. do ?

Summary

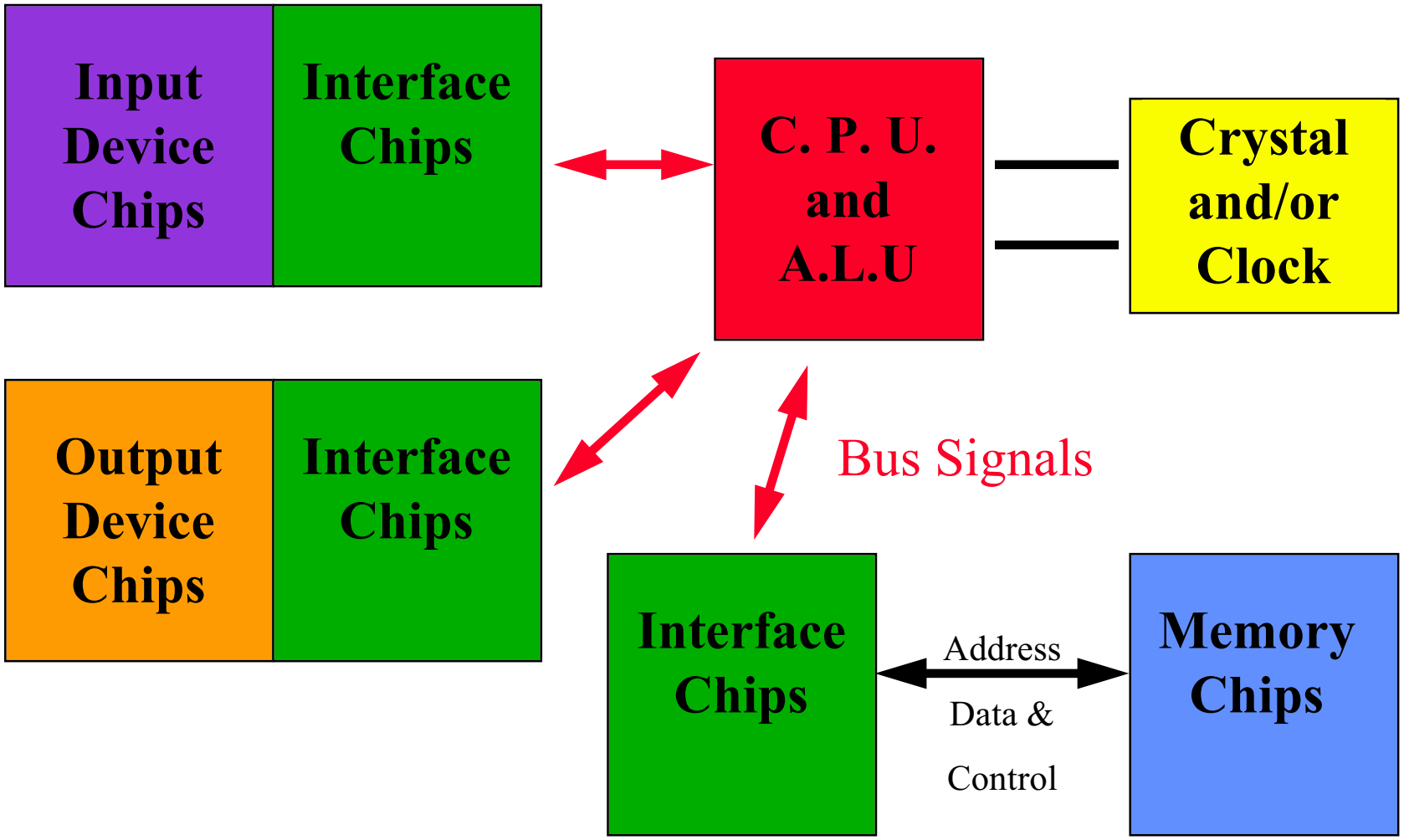
- It Adds Numbers.
- It Makes Numbers Negative.
- It Shifts and Rotates Digits in Numbers.
- It Compares and Tests Numbers.
- It performs Logic functions on Numbers.

What is a Computer made from ?



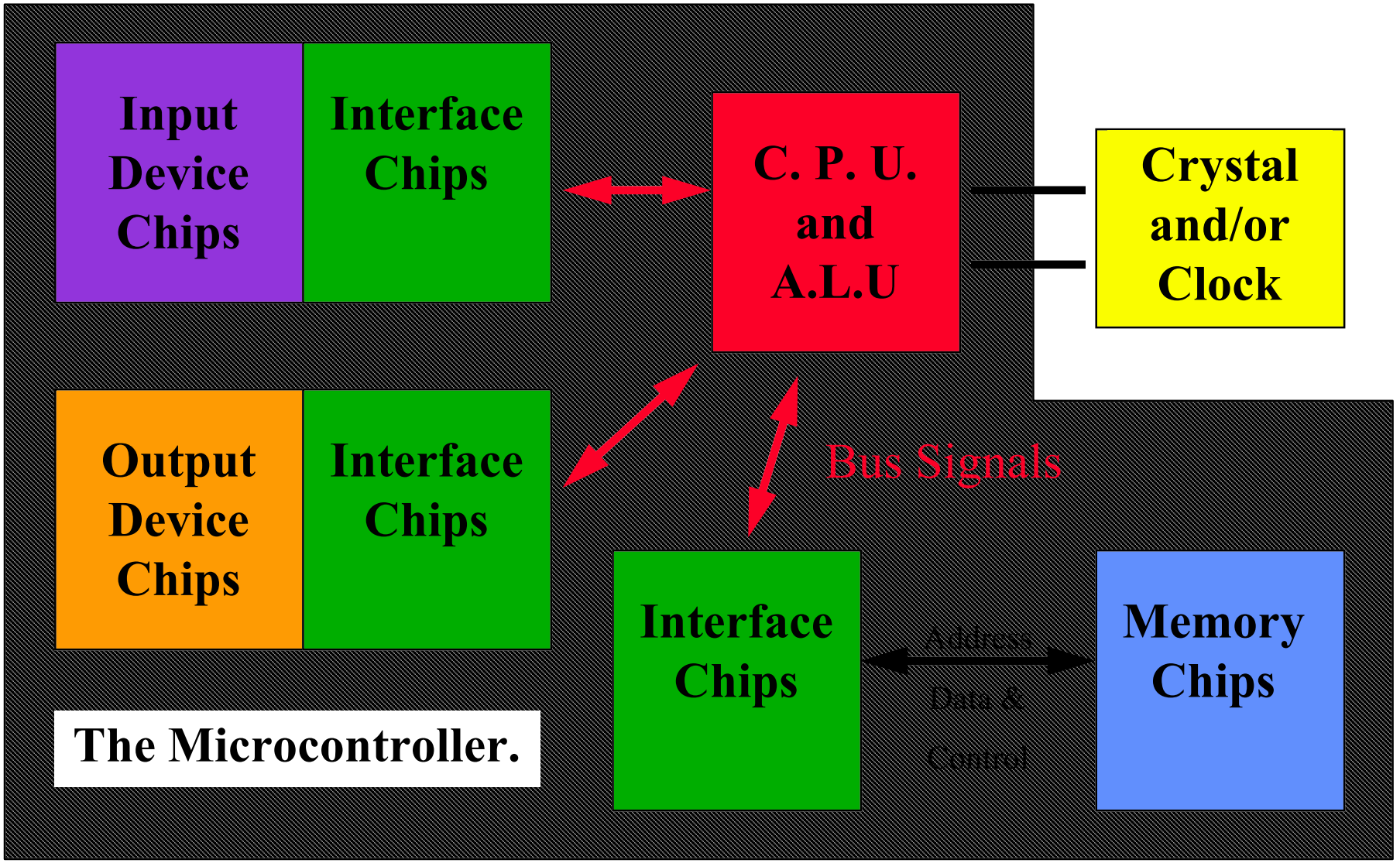
The Micro_Processor

Block diagram of a Microprocessor System.



The Micro_Controller

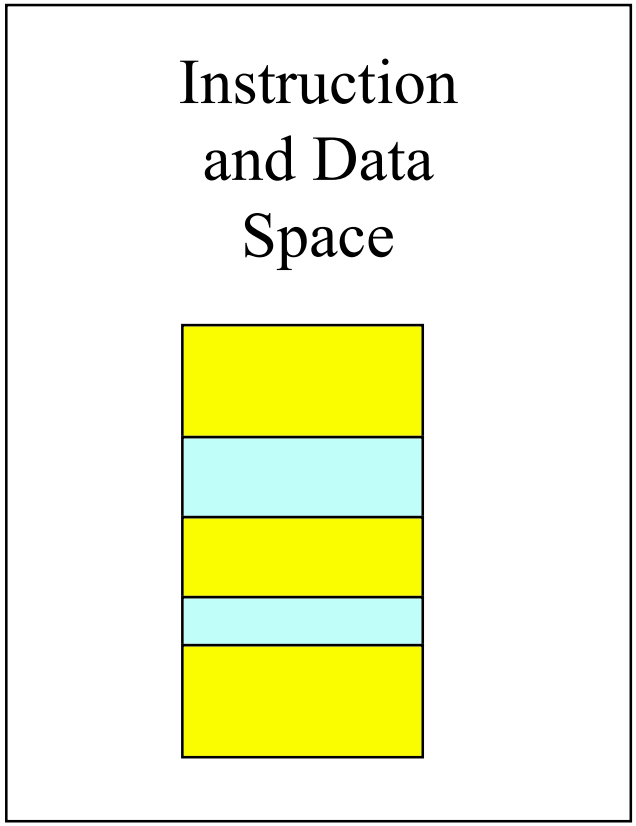
Block diagram of a Microcontroller.



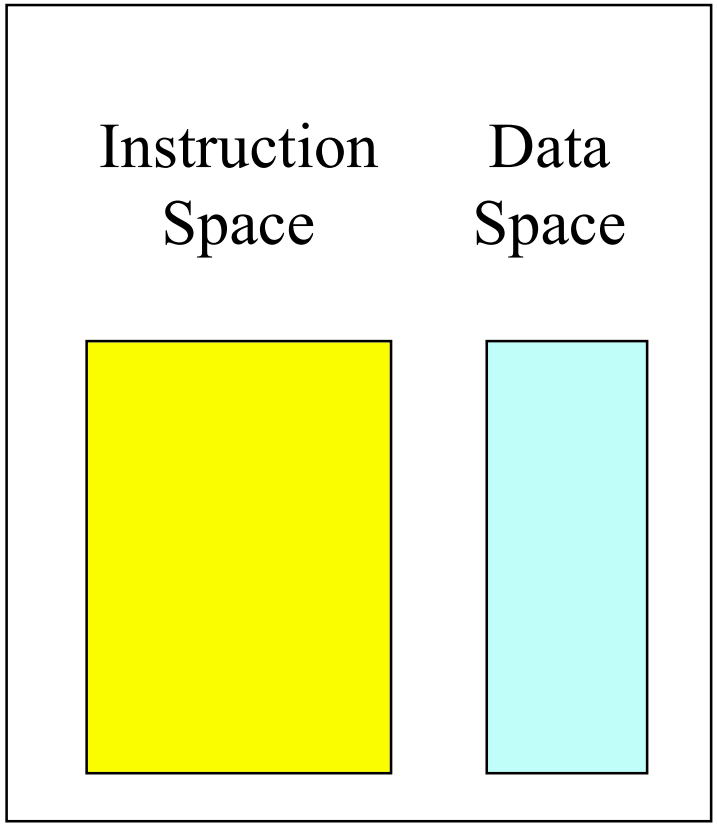
Architecture

Architecture.

Von Neumann Structure



Harvard Structure

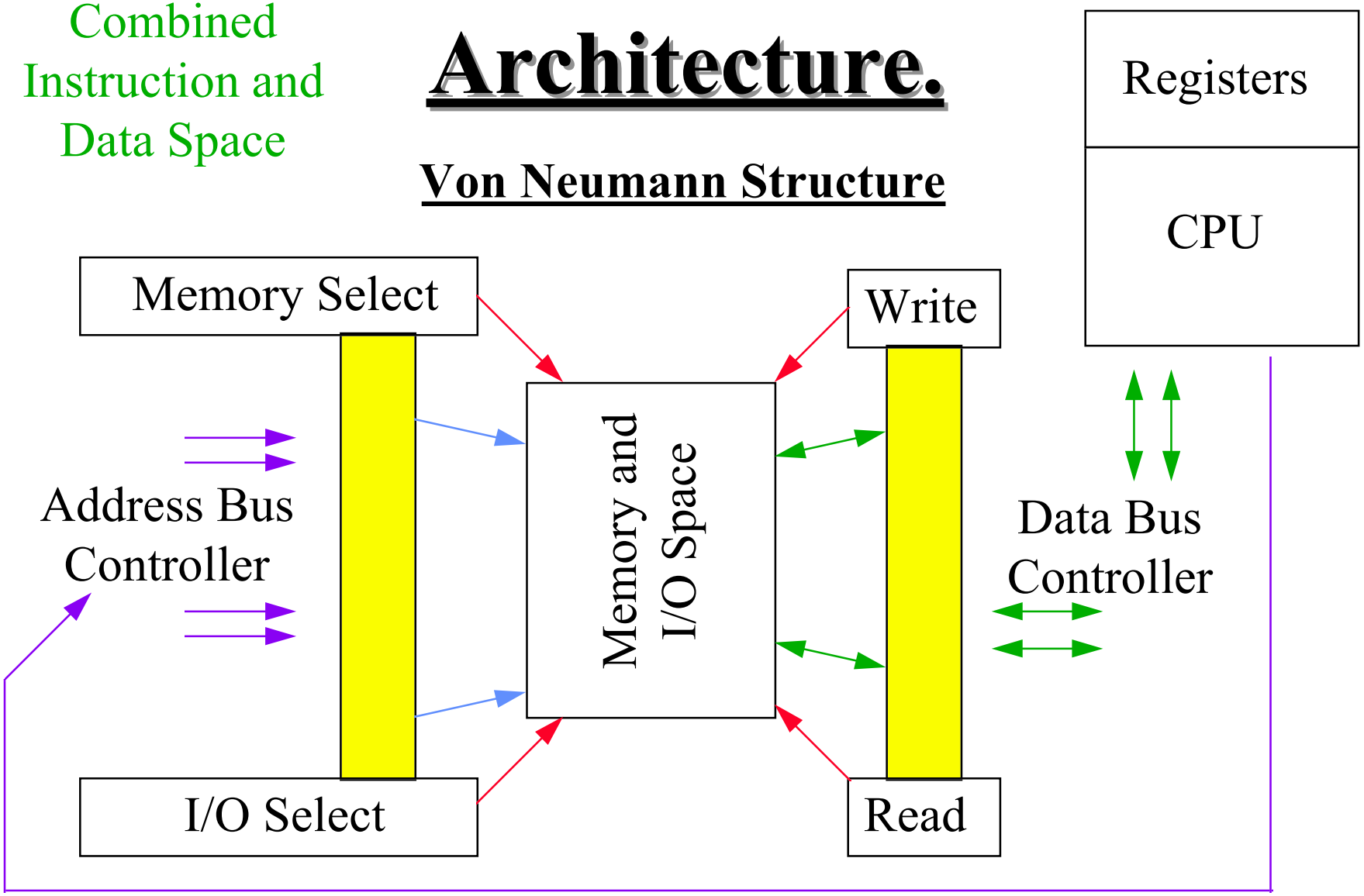


Memory Models

Combined Instruction and Data Space

Architecture.

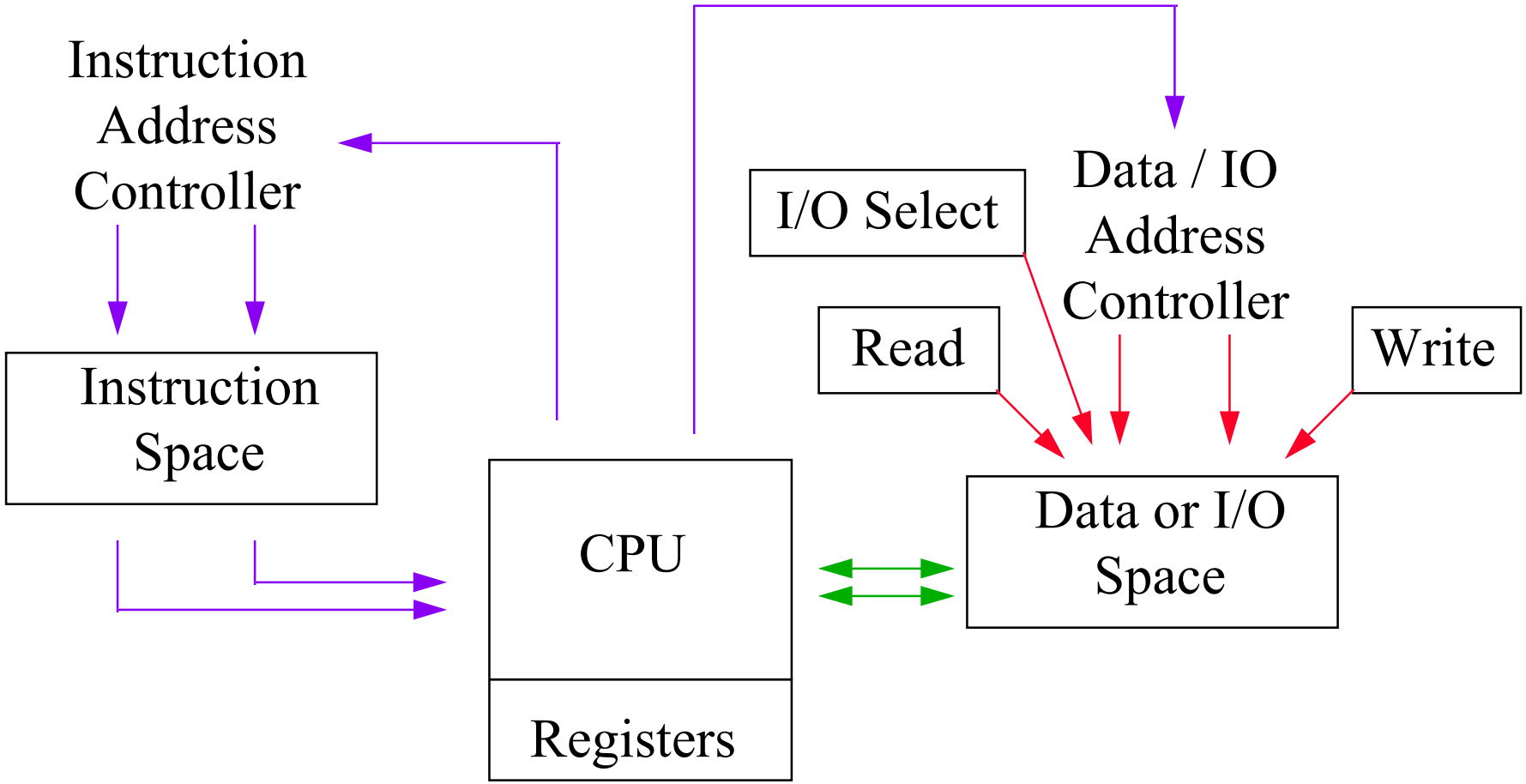
Von Neumann Structure



Memory Models : Von Neumann Structure

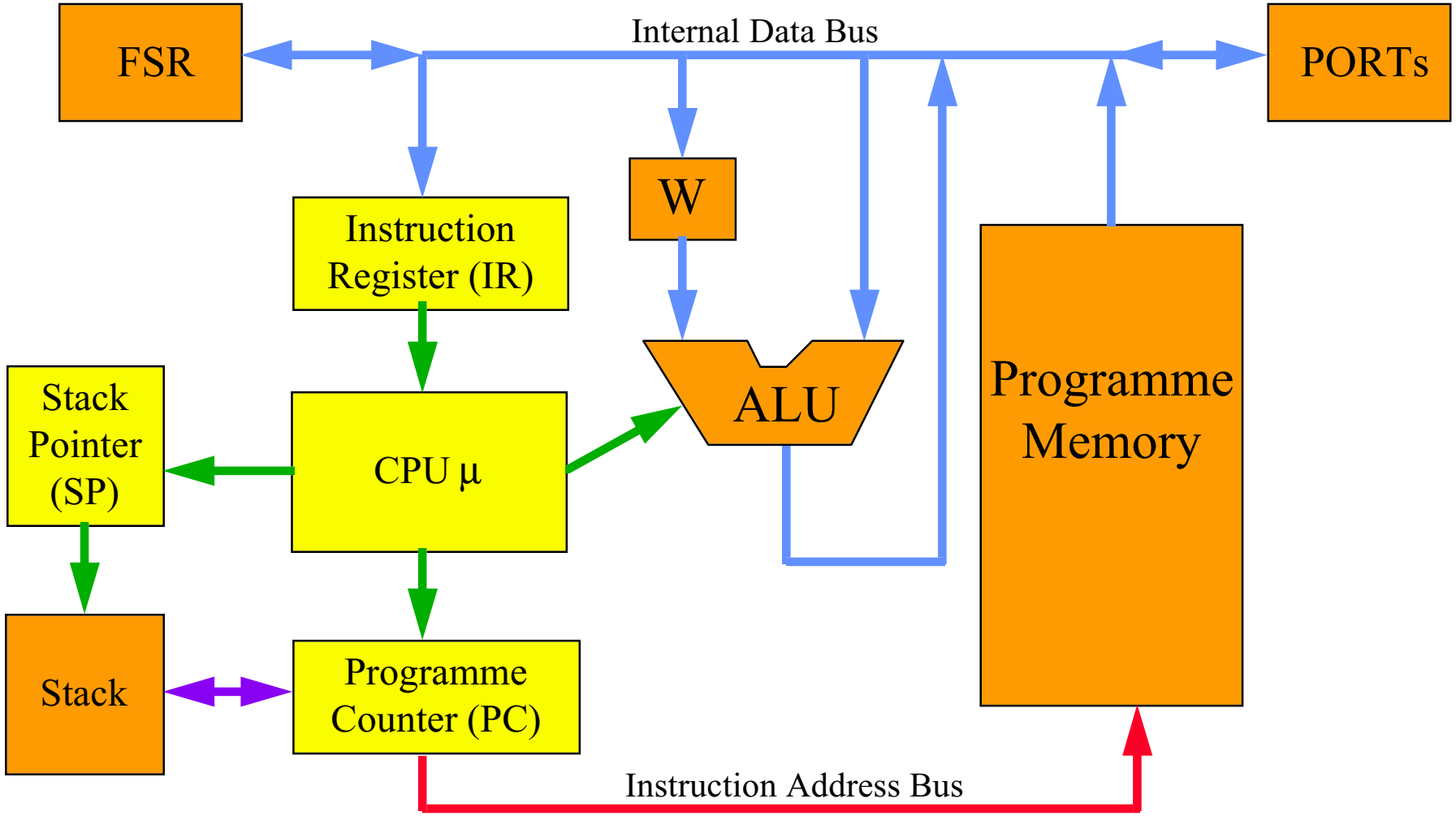
Separated Instruction and Data Space

Architecture.



Memory Models : Harvard Structure.

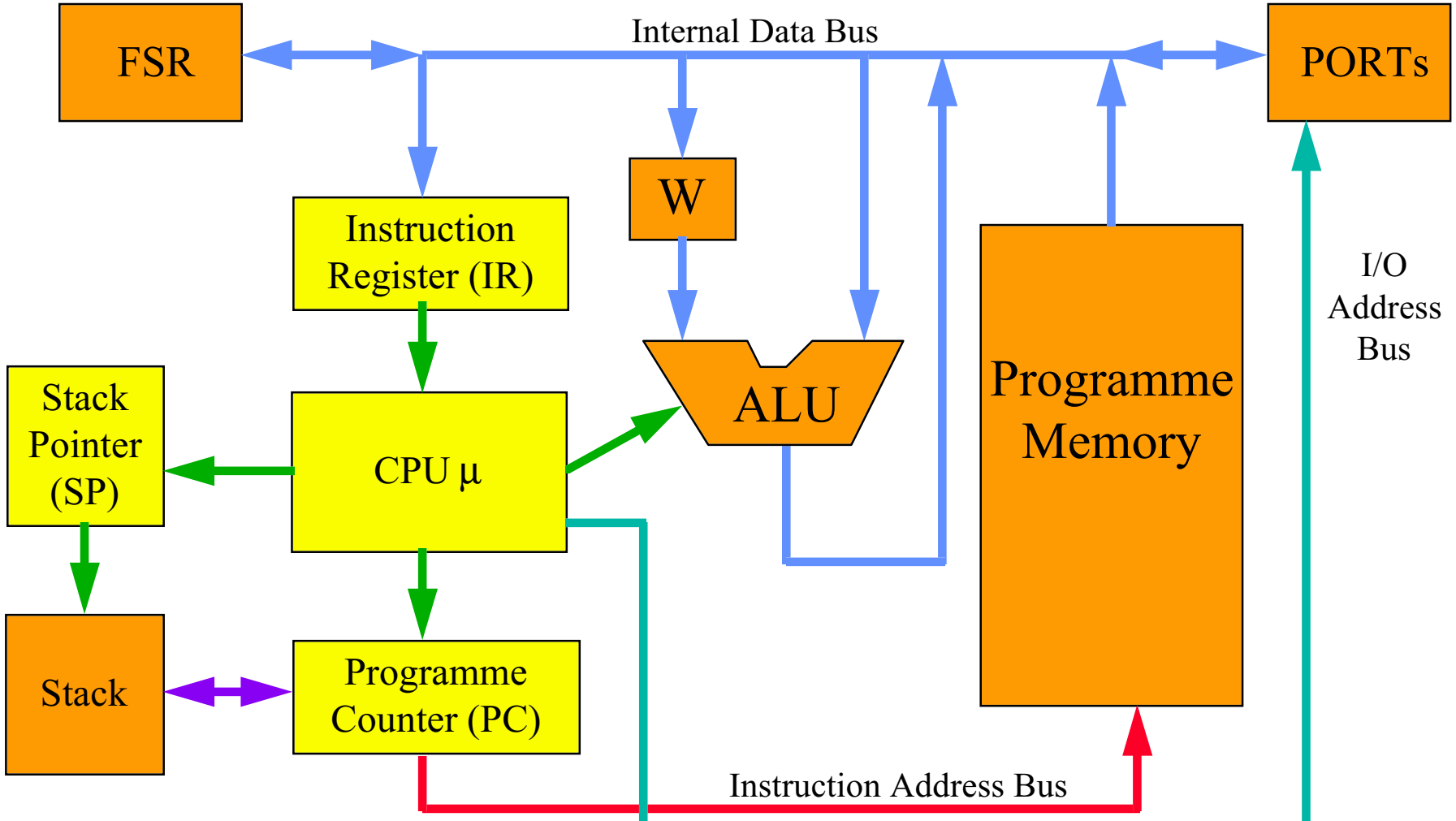
Processing Instructions.



ADDLW #9,W

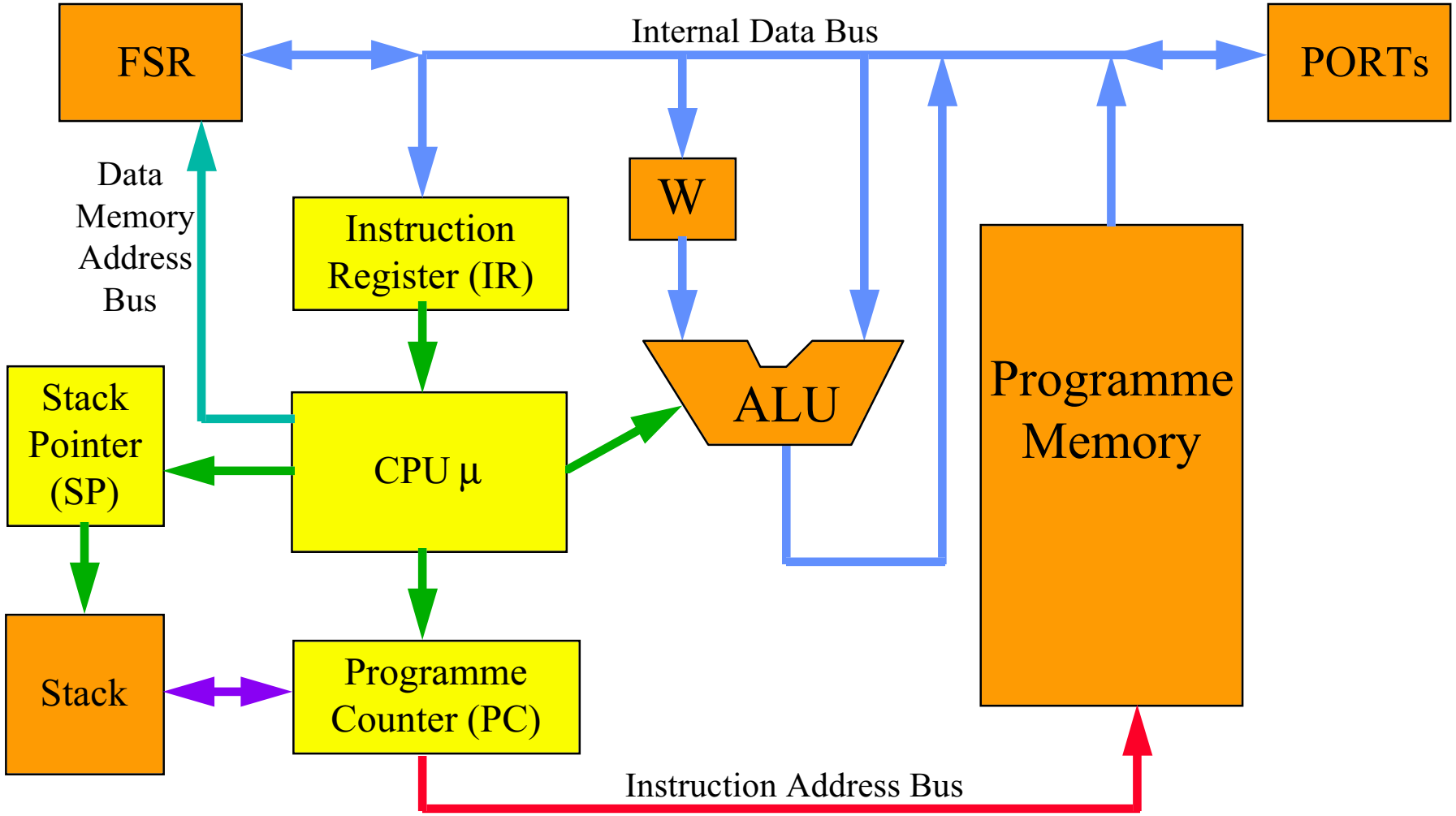
Typical Processor Harvard Hardware Structure.

Processing Instructions.



Typical Processor Memory Mapped I/O Access.

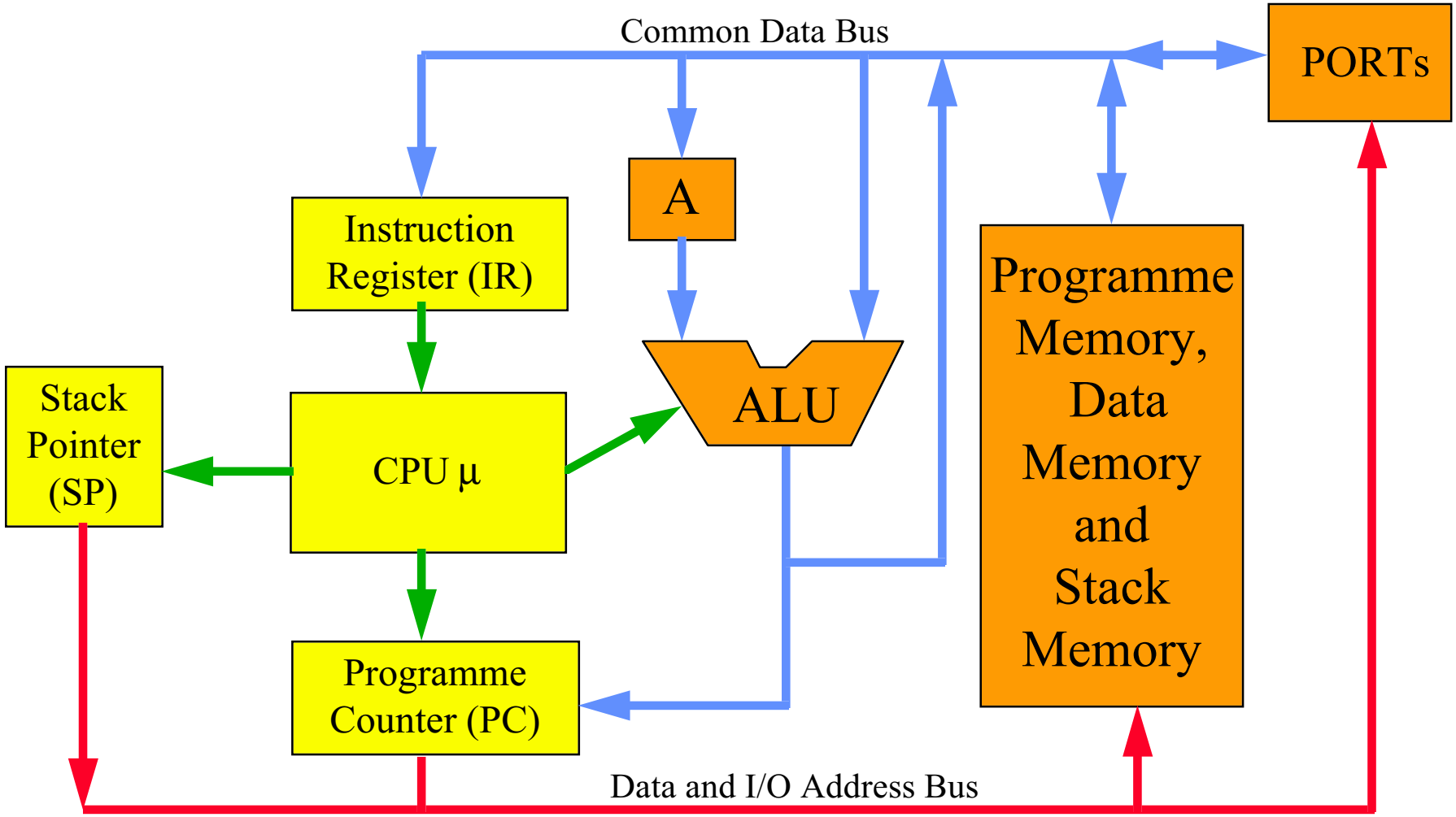
Processing Instructions.



MOVF PortA,W

Typical Processor Data Memory Access.

Processing Instructions.



Typical Processor Von Neumann Hardware.

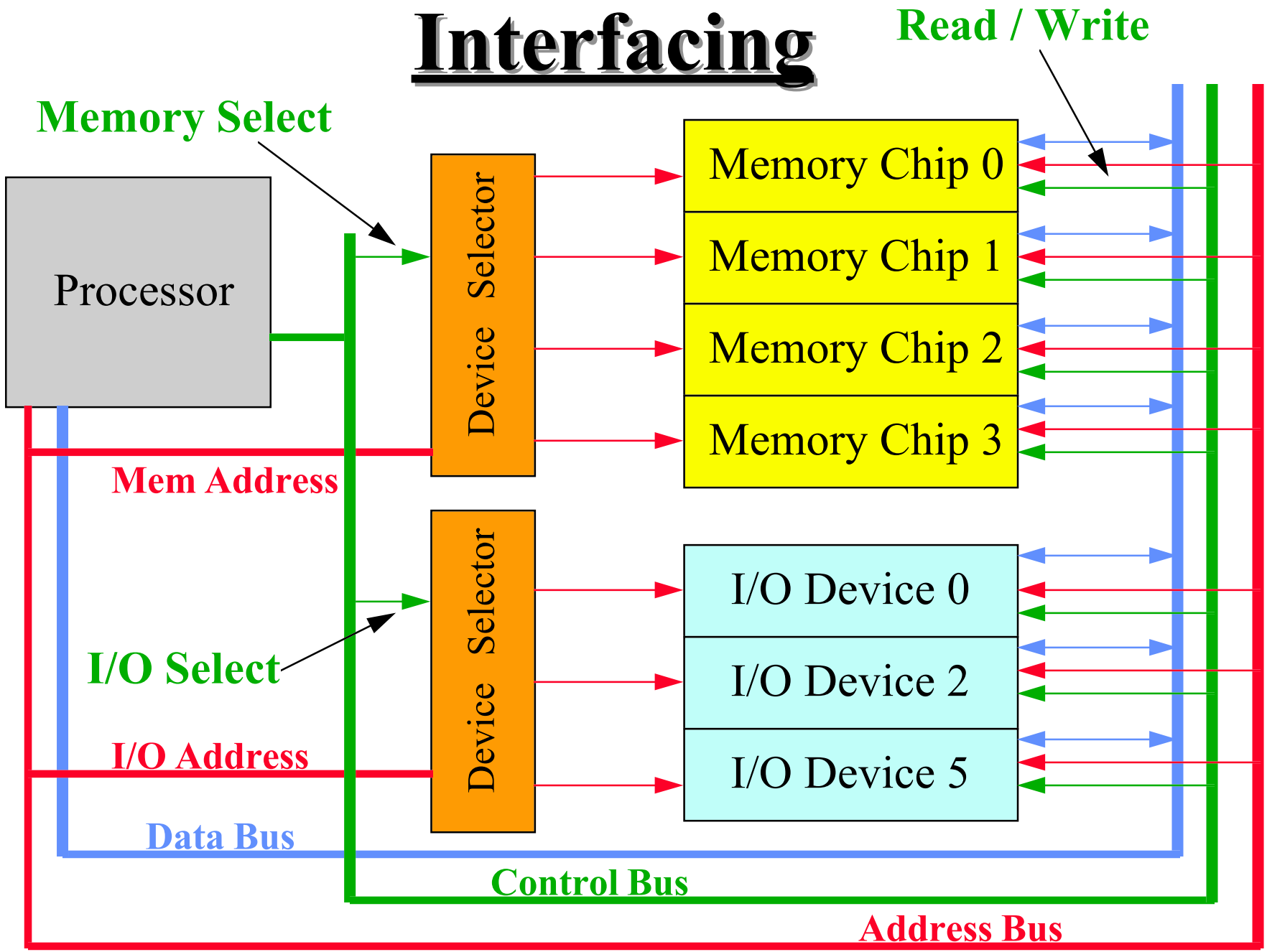
Interfacing

Interfacing.

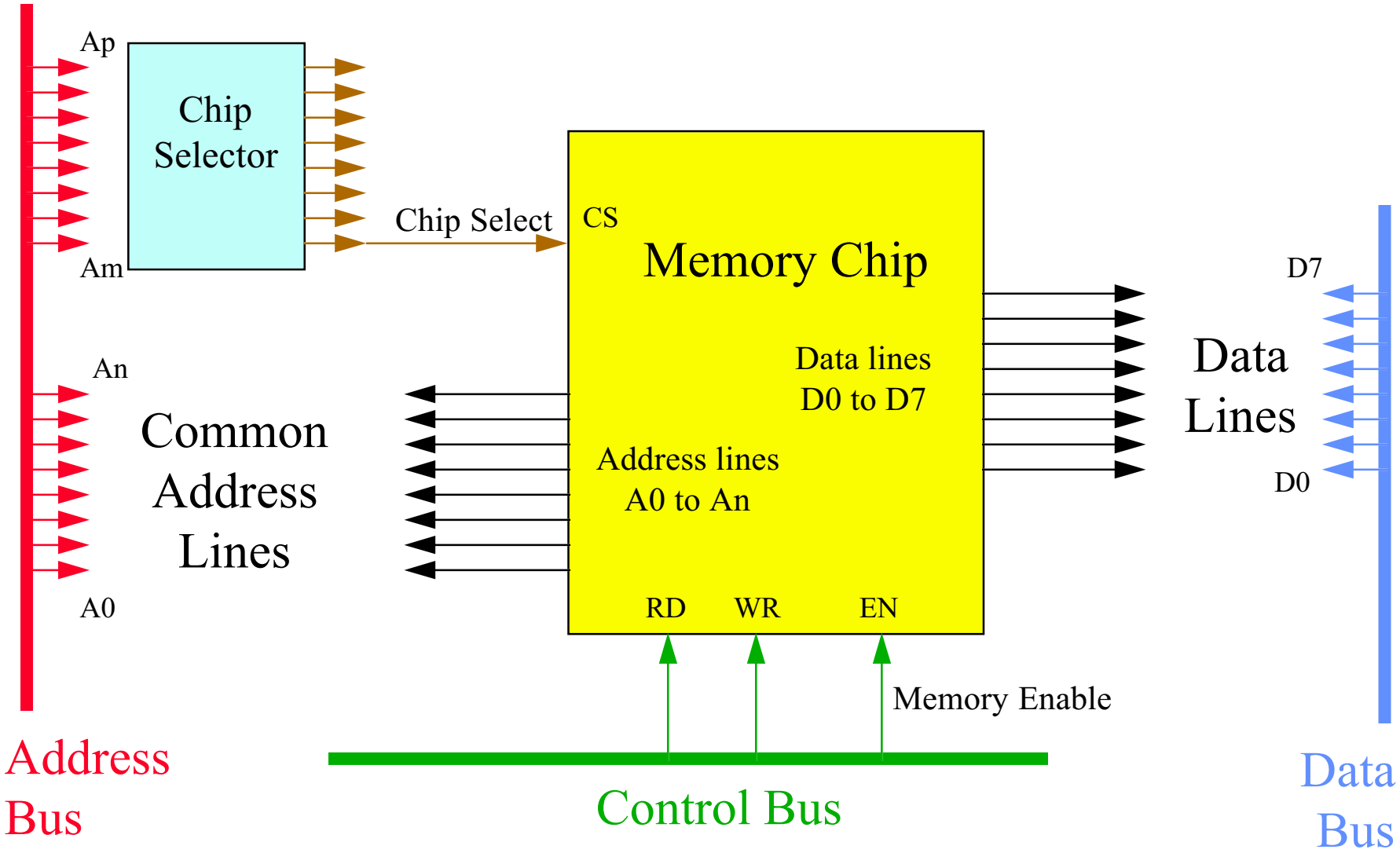
- A BUS is a collection of connections between two or more devices or systems.
- An Address Bus is used to identify Devices or Locations in a hardware system.
- A Data Bus is used to move information between Devices or Locations in a hardware system.
- A Control Bus is all the other common communication lines that are not Data or Address.

Common Terms.

Interfacing

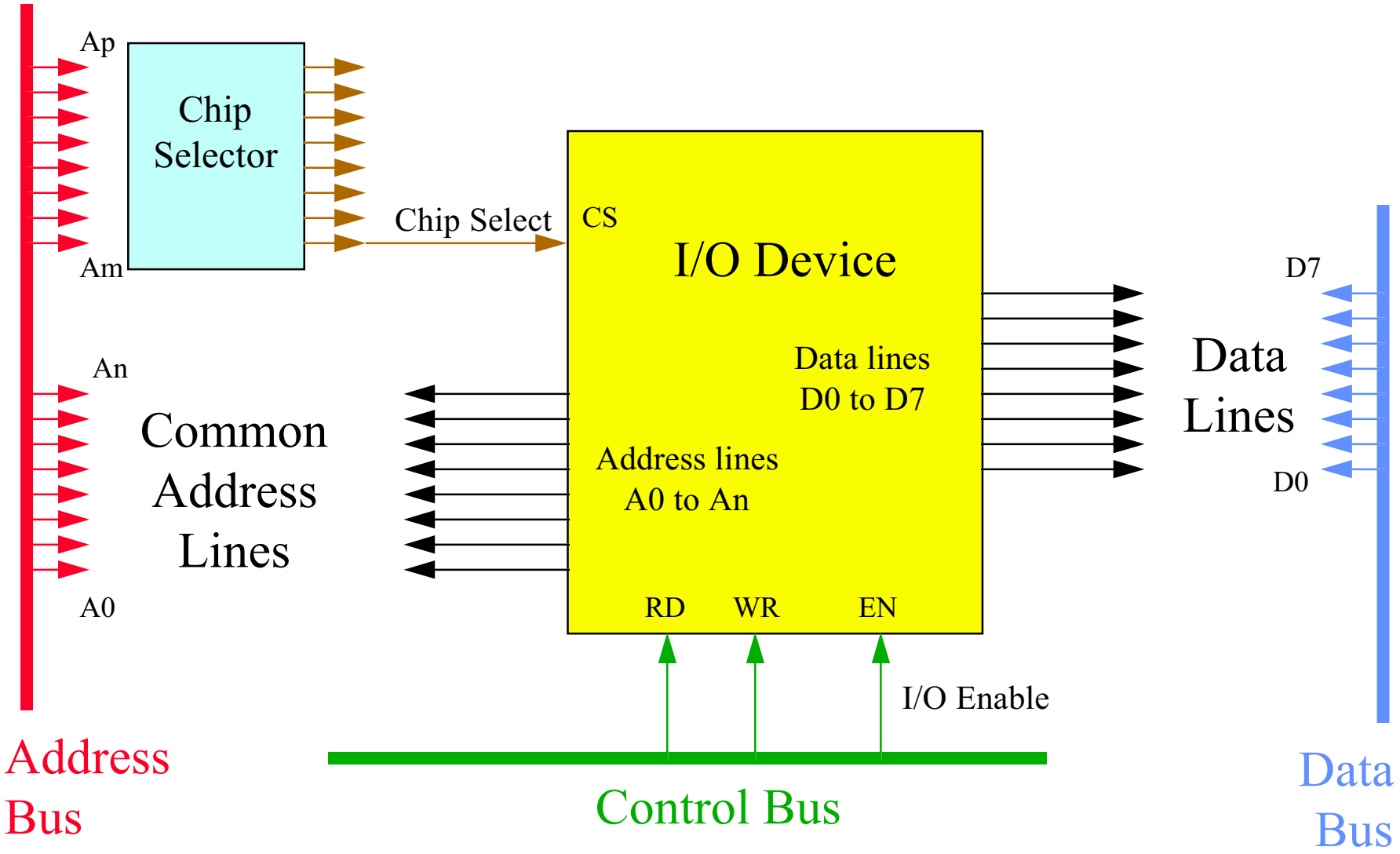


Interfacing



Interfacing Memory Devices in a Memory Mapped area.

Interfacing



Interfacing I/O Devices in a Memory Mapped area.

Interfacing.

- An Address Bus is used to identify Devices or Locations **One Way Transfer**.
- A Control Bus is all the other common communication lines **One Way Transfer**.
- A Data Bus is used to move information between Devices or Locations **Two Way Transfer**.
- Question: How do we resolve the situation where two or more Devices / Chips try to communicate at the same time (Multiple Outputs connected to the same point).

Observations.

Problem Interfacing.

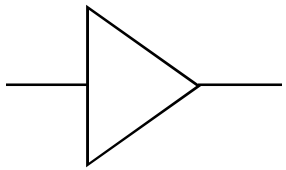
- If we connect two digital outputs together then:-
- If both Outputs are High the Output is High.
- If both Outputs are Low the Output is Low.
- If one Output is High and one Output is Low then the following may happen :-
  - the most powerful Output will overwhelm the weaker Output signal.
  - The Outputs will mutually destroy each other.
  - The Summation Output will be indeterminate (neither a Logic High or Logic Low).

Solution

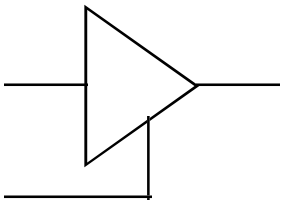
Interfacing.

- To avoid chip destruction special type of Output is introduced into the devices and is called :-
- TRISTATE (Three State Logic)
- Tristate puts the Output connection into a High Impedance (High Resistance) state so that it does not effect any other connected points.
- By ensuring all devices bar one is in Tristate then effective communications by a number of devices on a single wire is possible.
- Tristate devices therefore can be in one of three states :- Logic High, Logic Low or Tristate.

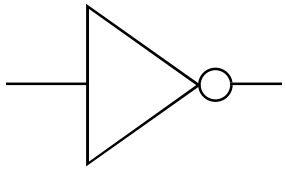
Interfacing.



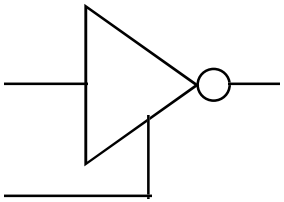
Buffer



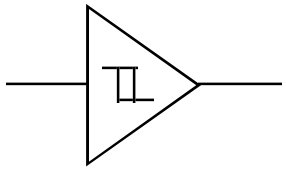
TriState Buffer



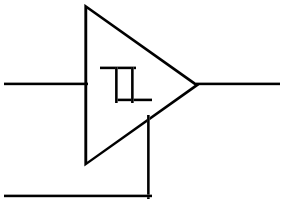
Inverter



TriState Inverter



Schmitt Trigger Buffer

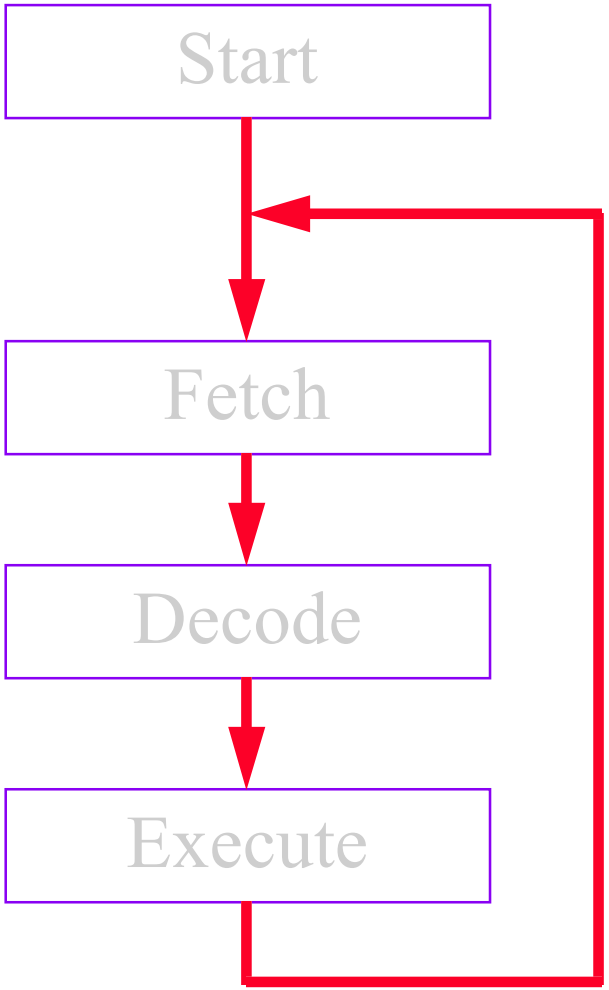


Schmitt Trigger Tristate Buffer

Tristate Device Symbols

Processing Instructions

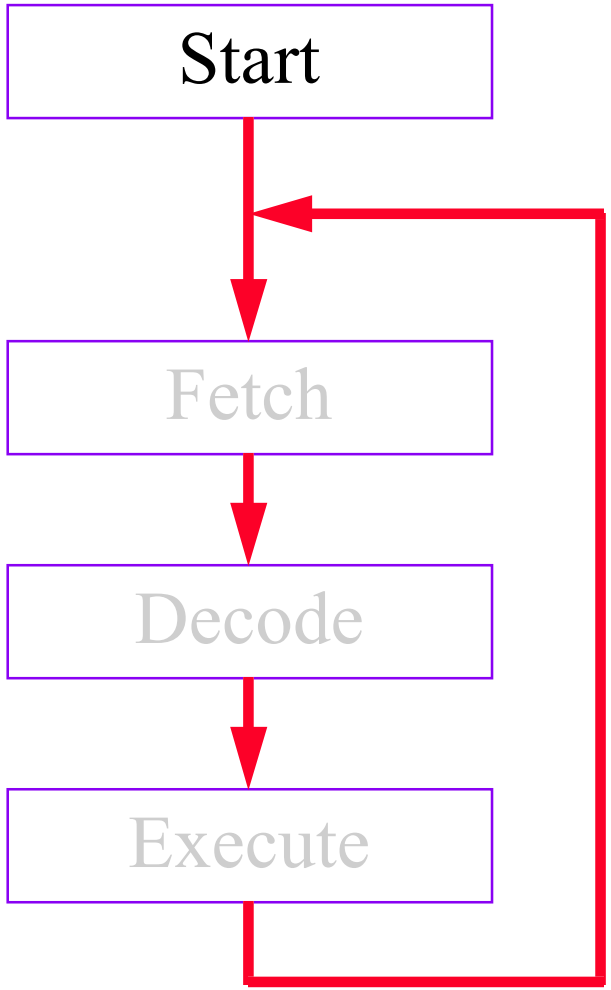
Processing Instructions.



The Fetch, Decode Execute Cycle

Processing Instructions.

Set the Programme code Pointer to the first instruction (start address).

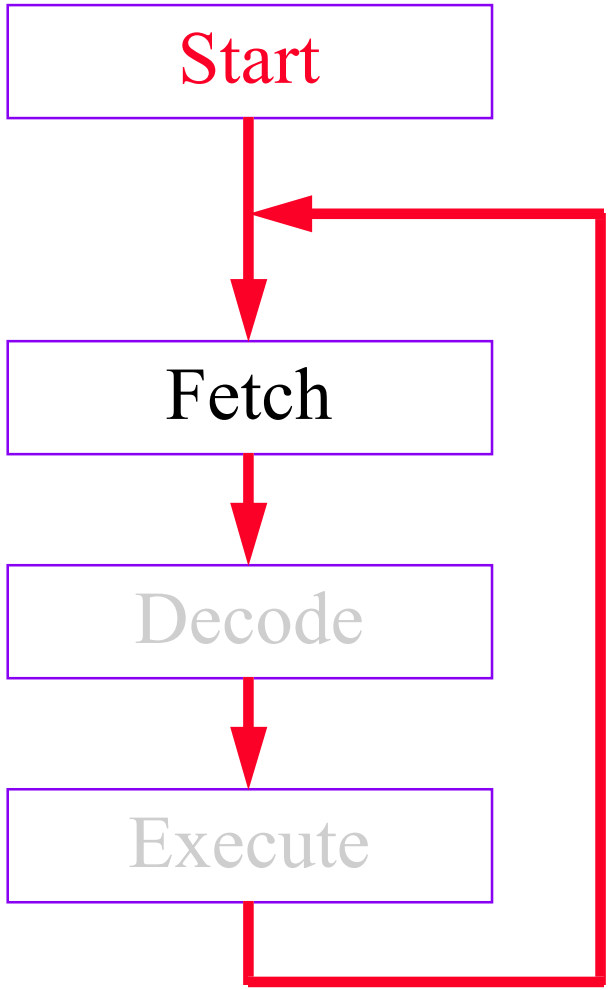


The Fetch, Decode Execute Cycle

Processing Instructions.

Set the Programme code Pointer to the first instruction (start address).

Get the current instruction(PC)→IR and then increment the Programme code Pointer. PC=PC+1



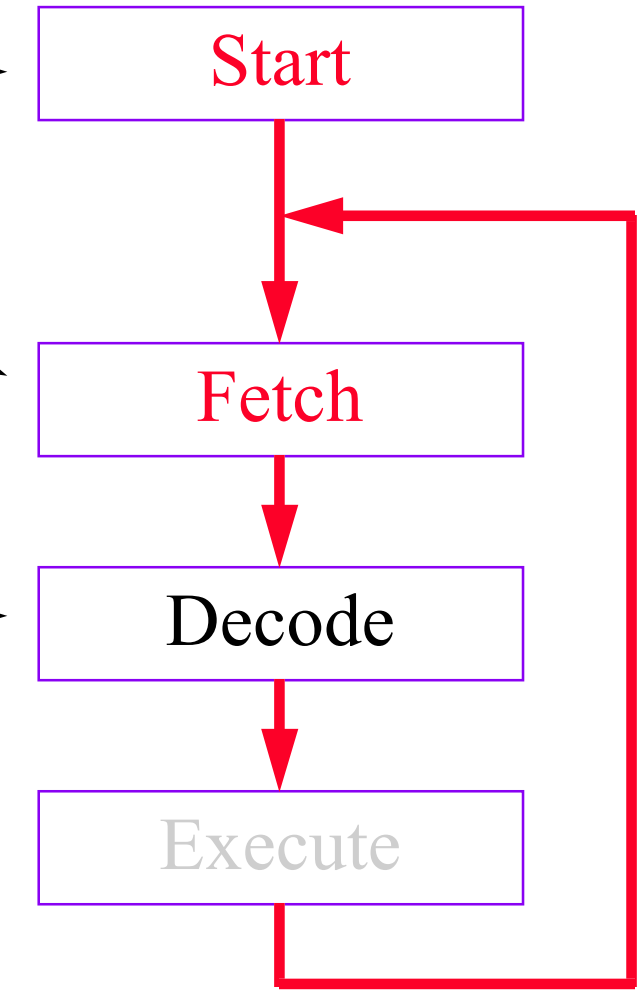
The Fetch, Decode Execute Cycle

Processing Instructions.

Set the Programme code Pointer to the first instruction (start address).

Get the current instruction(PC)→IR and then increment the Programme code Pointer. PC=PC+1

Plan the actions needed to be performed to obey the specification of the current instruction.



The Fetch, Decode Execute Cycle

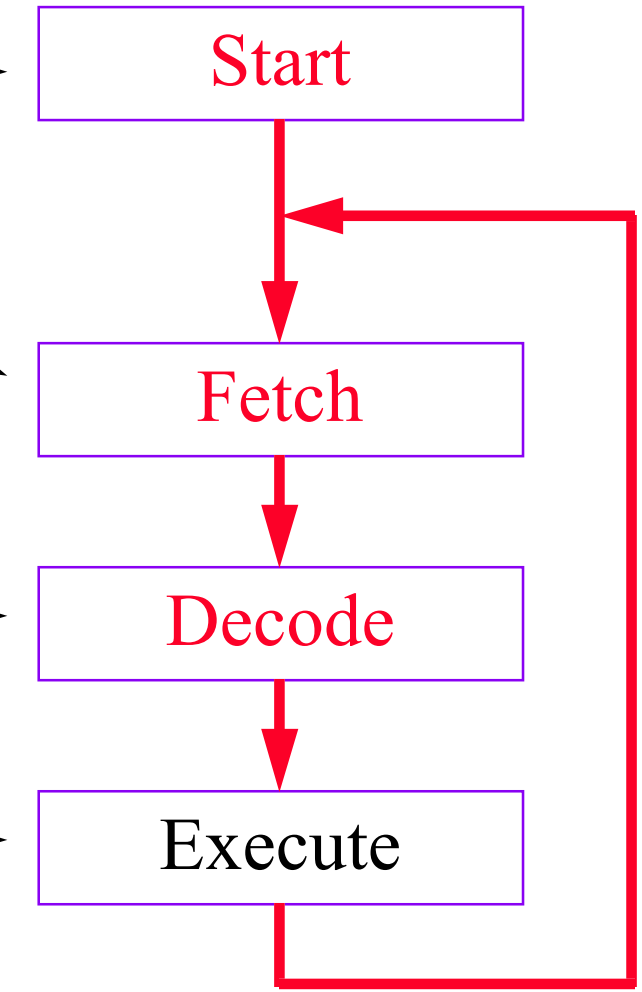
Processing Instructions.

Set the Programme code Pointer to the first instruction (start address).

Get the current instruction(PC)→IR and then increment the Programme code Pointer. PC=PC+1

Plan the actions needed to be performed to obey the specification of the current instruction.

Implement the operations planned for the current instruction.



The Fetch, Decode Execute Cycle

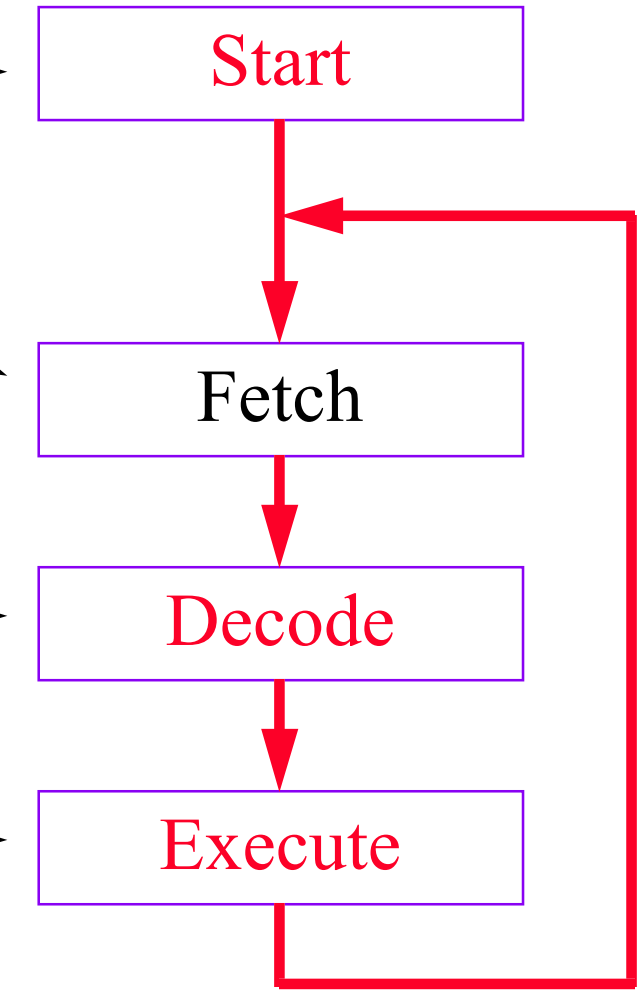
Processing Instructions.

Set the Programme code Pointer to the first instruction (start address).

Get the current instruction(PC)→IR and then increment the Programme code Pointer. PC=PC+1

Plan the actions needed to be performed to obey the specification of the current instruction.

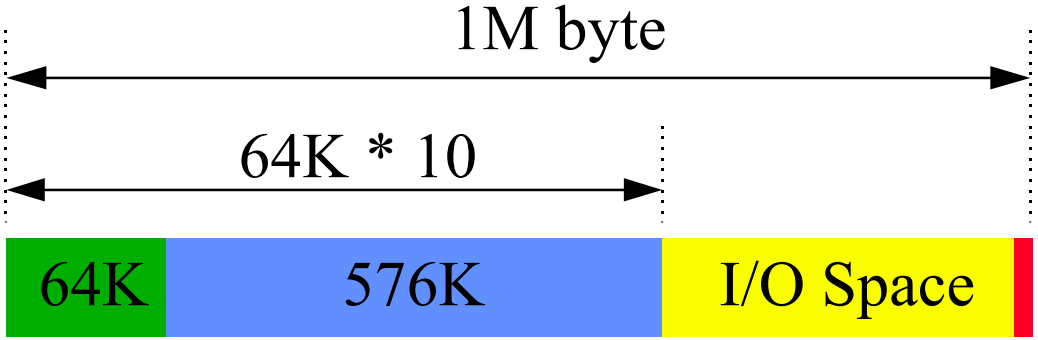
Implement the operations planned for the current instruction.



The Fetch, Decode Execute Cycle

Computers.

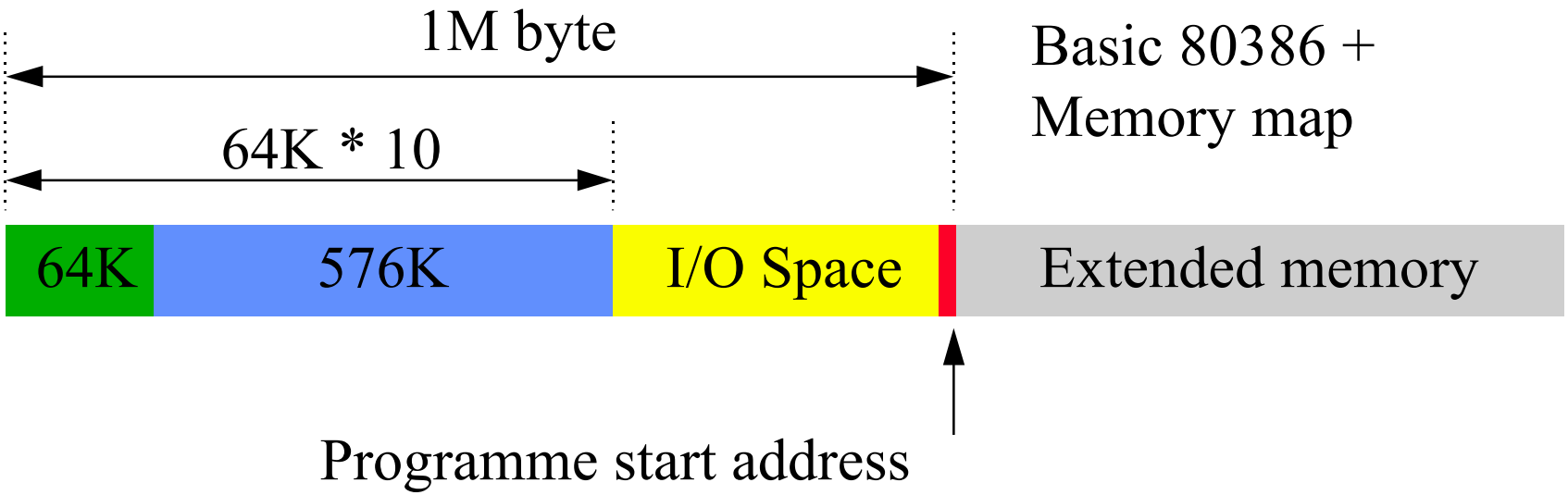
- What happens when you Switch on ?
- The instruction pointer is initialized (for example) :-
- Z80, 8080, PIC start point is beginning of memory.
  - Z80 1st OS CP/M (Computer Program & Monitor)
- 6800, 68000 start point is end of memory.
  - 68000 used for MAC OS (Operating System).
- 80x86 used for DOS (Disk Operating System)+WINDOWS.



Basic 8086
Memory map

Computers.

- Some processors like 80386 the start point is a specific address part way through memory.



The Start Up

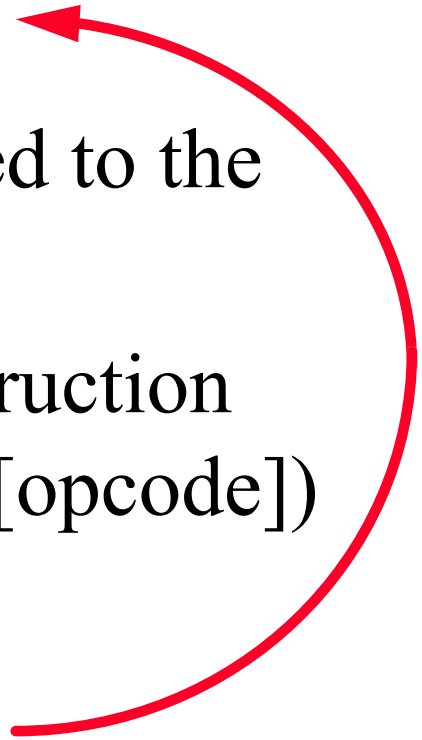
Computers.

- The System control clock starts.
- The C.P.U. resets all or some of its internal registers to some preset value or state.
- (*) The Repeat operations.
- The instruction pointer is incremented to the next instruction to be processed.
- The Instruction decoder gets the instruction from memory. (The Operation code [opcode])
- The C.P.U obeys the Instruction.

The Start Up

Computers.

- The System control clock starts.
- The C.P.U. resets all or some of its internal registers to some preset value or state.
- (*) The Repeat operations.
- The instruction pointer is incremented to the next instruction to be processed.
- The Instruction decoder gets the instruction from memory. (The Operation code [opcode])
- The C.P.U obeys the Instruction.
- Start again from the (*) marked line.



Computers.

- Implications of the start up sequence.
- The processor is at best only an **obedient moron**.
- If the information stored at the programme start address is not a valid programme then the actions of the processor are unpredictable.
- It is the programmers responsibility to ensure that the instructions the processor is to obey will implement the function that is required.
- Any coding error will usually manifest itself as a programme bug (Microsoft = a new feature).

ALU Instructions

Processing Instructions.

- Place the PC (Programme Counter) value on the address bus.
- Select Memory (Where the instruction should be)
- Read contents of the memory location
- Increment Programme Counter (PC = PC +1)

The Fetch Process. (Typical Operations)

Processing Instructions.

- Place the PC (Programme Counter) value on the address bus.
- Select Memory (Where the instruction should be)
- Read contents of the memory location
- Increment Programme Counter (PC = PC +1)

PC = 00 0111 1010 1100

The Default Clock Cycle

Instruction = FIDS AA BB CC = 7317 31 31 31

PC = 00 0111 1010 1101

The Fetch Process.

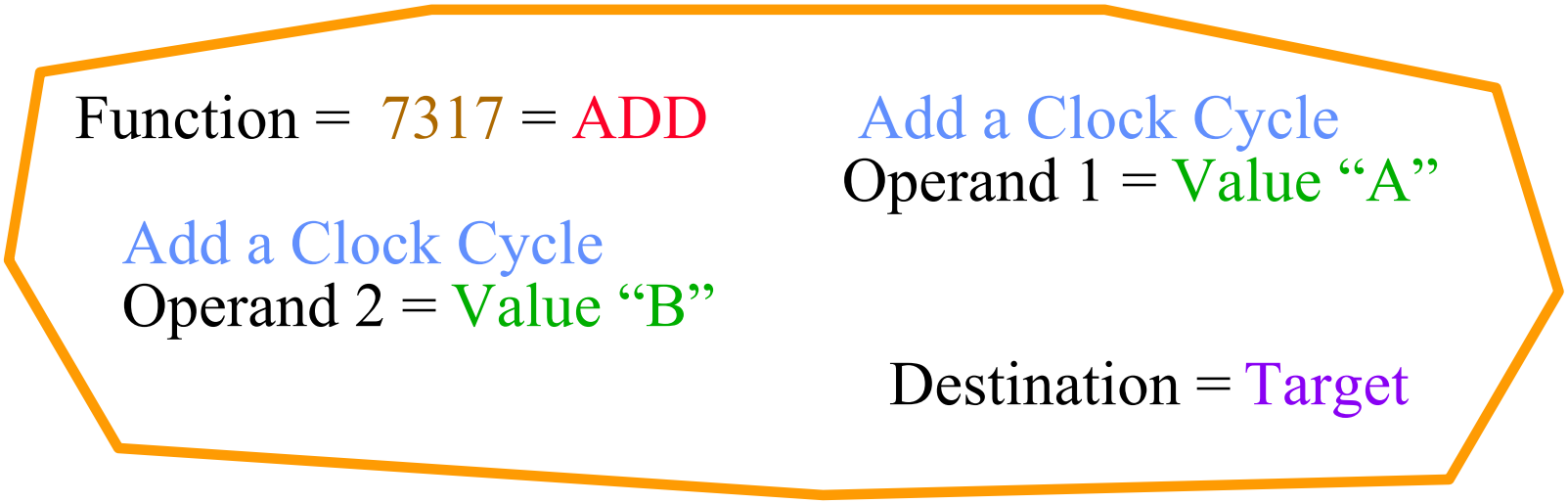
Processing Instructions.

- Identify Instruction Function.
- Locate operand/s (Access Address Bus maybe !)
- Get contents of the Operands (Inputs)
- Prepare for Destination (Output)

The Decode Process. (Typical Operations)

Processing Instructions.

- Identify Instruction Function.
- Locate operand/s (Access Address Bus maybe !)
- Get contents of the Operands (Inputs)
- Prepare for Destination (Output)



The Decode Process.

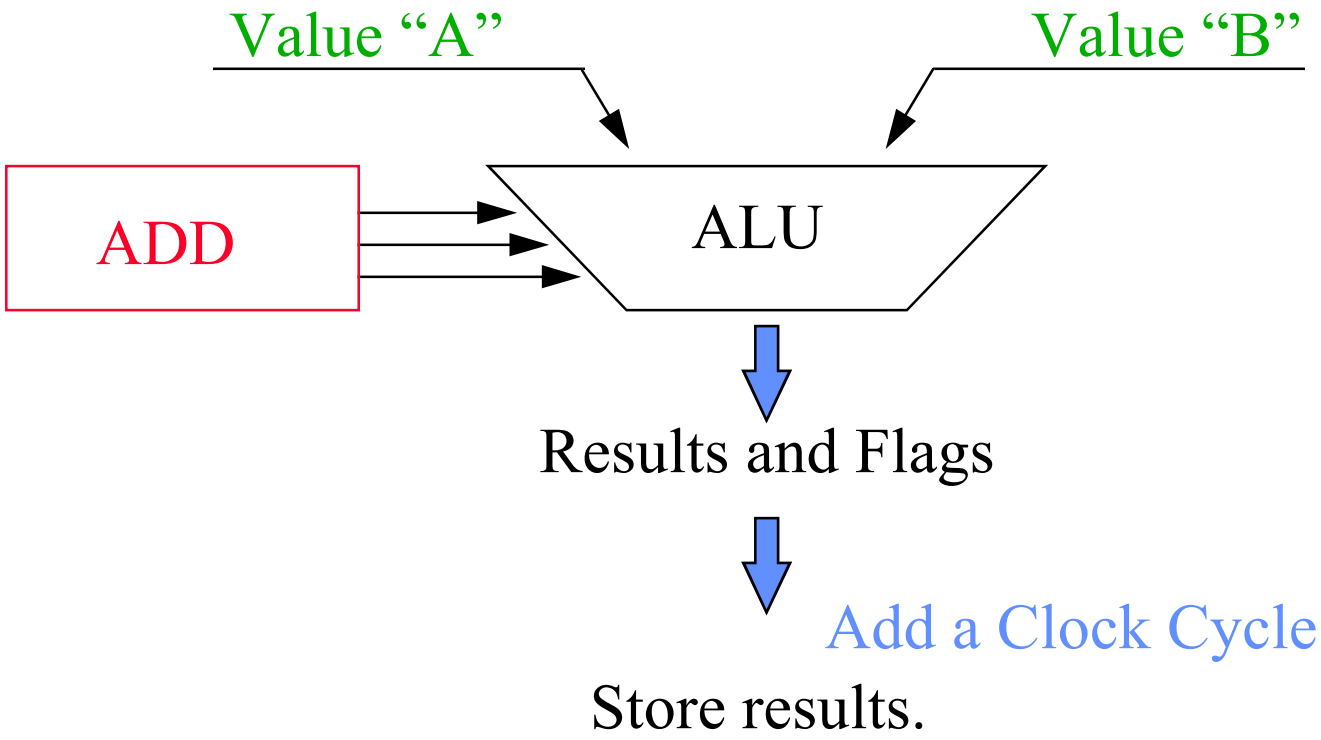
Processing Instructions.

- Implement the Decoded instruction.

The Execute Process. (Typical Operations)

Processing Instructions.

- Implement the Decoded instruction.



The Execute Process.

STACK Instructions

Processing Instructions.

- Place the PC (Programme Counter) value on the address bus.
- Select Memory (Where the instruction should be)
- Read contents of the memory location
- Increment Programme Counter (PC = PC +1)

The Fetch Process. (Stack Operations)

Processing Instructions.

- Place the PC (Programme Counter) value on the address bus.
- Select Memory (Where the instruction should be)
- Read contents of the memory location
- Increment Programme Counter (PC = PC +1)

PC = 00 0111 1010 1100

The Default Clock Cycle

Instruction = FIDS AA BB CC = 3123 21 21 21

PC = 00 0111 1010 1101

The Fetch Process. (Stack Operations)

Processing Instructions.

- Identify Instruction Stack Process.
- Instruction = CALL address
- or
- Instruction = RETURN

The Decode Process. (Stack Operations)

STACK "CALL" Instruction

Processing Instructions.

- Identify Instruction Stack Process.
- Instruction = CALL

Function = 3123 21 21 21 = CALL Address

Add a Clock Cycle

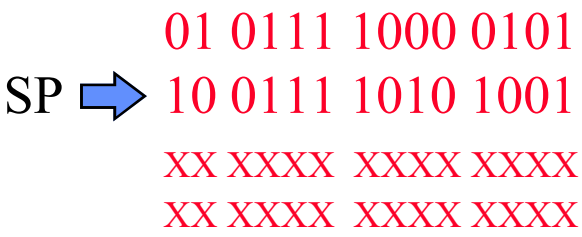
Extract New Programme Address = Address 212121

Get Current Stack Address = Address 10101000

The Decode Process. (Stack Operations)

Processing Instructions.

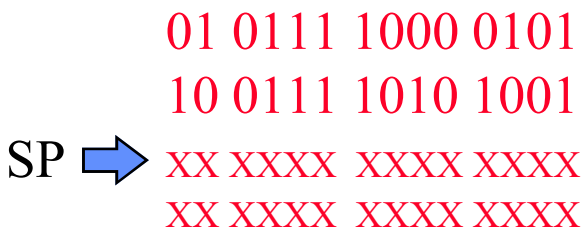
- Implement the Decoded instruction.



The Execute Process (Stack Operations).

Processing Instructions.

- Implement the Decoded instruction.
- Modify Stack Pointer (SP=SP-1)



The Execute Process (Stack Operations).

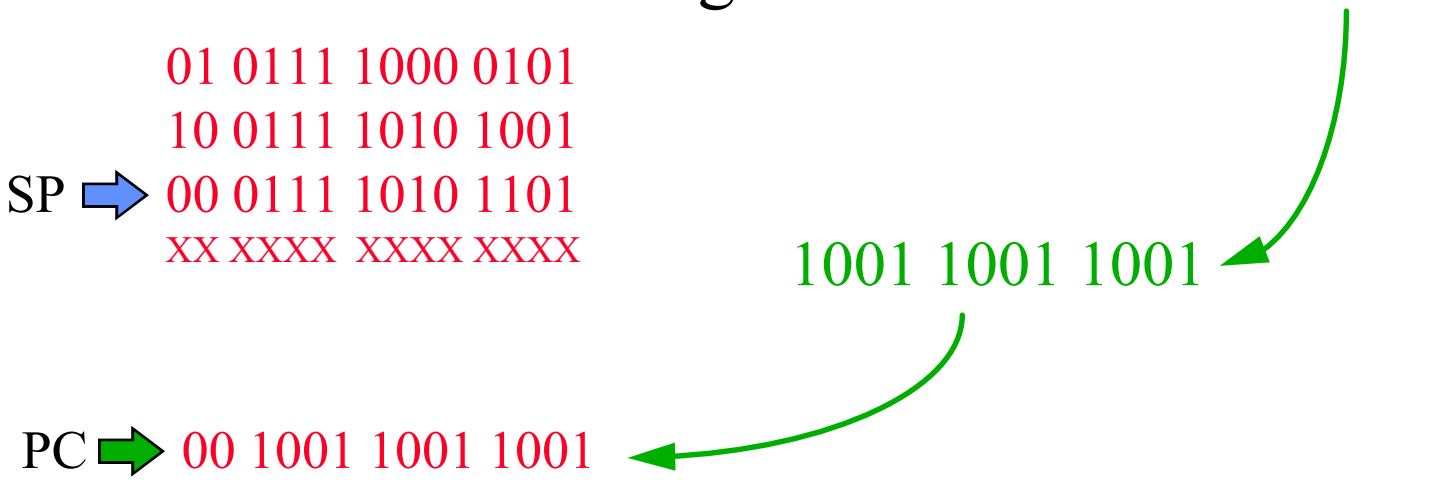
Processing Instructions.

- Implement the Decoded instruction.
- Modify Stack Pointer (SP=SP-1)
- Place PC address in Stack Memory Area

The Execute Process (Stack Operations).

Processing Instructions.

- Implement the Decoded instruction.
- Modify Stack Pointer (SP=SP-1)
- Place PC address in Stack Memory Area.
- PC set to New Programme Address 212121.



The Execute Process (Stack Operations).

STACK "RETURN" Instruction

Processing Instructions.

- Implement the Decoded instruction (RETURN).
- Get PC address from Stack Memory Area

PC ➡ 00 0101 1011 0000

The Execute Process (Stack Operations).

Processing Instructions.

- Implement the Decoded instruction (RETURN).
- Get PC address from Stack Memory Area

The Execute Process (Stack Operations).

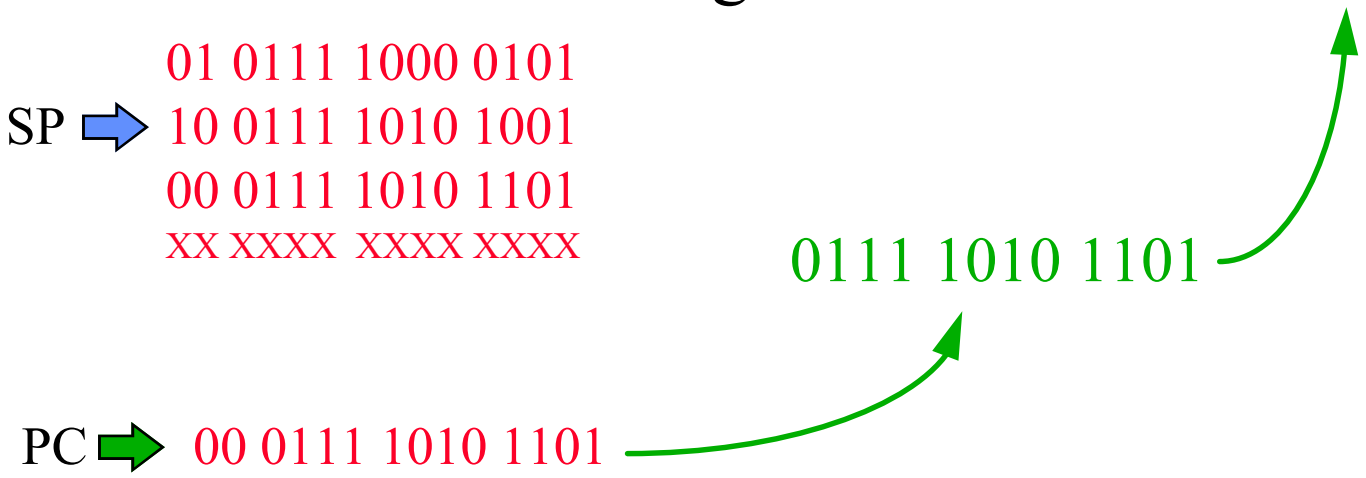
Processing Instructions.

- Implement the Decoded instruction (RETURN).
- Get PC address from Stack Memory Area
- Modify Stack Pointer (SP=SP+1)

The Execute Process (Stack Operations).

Processing Instructions.

- Implement the Decoded instruction (RETURN).
- Get PC address from Stack Memory Area
- Modify Stack Pointer (SP=SP+1)
- Restart at New Programme Address 132231.



The Execute Process (Stack Operations).

The PIC Processor

The Pic Processor.

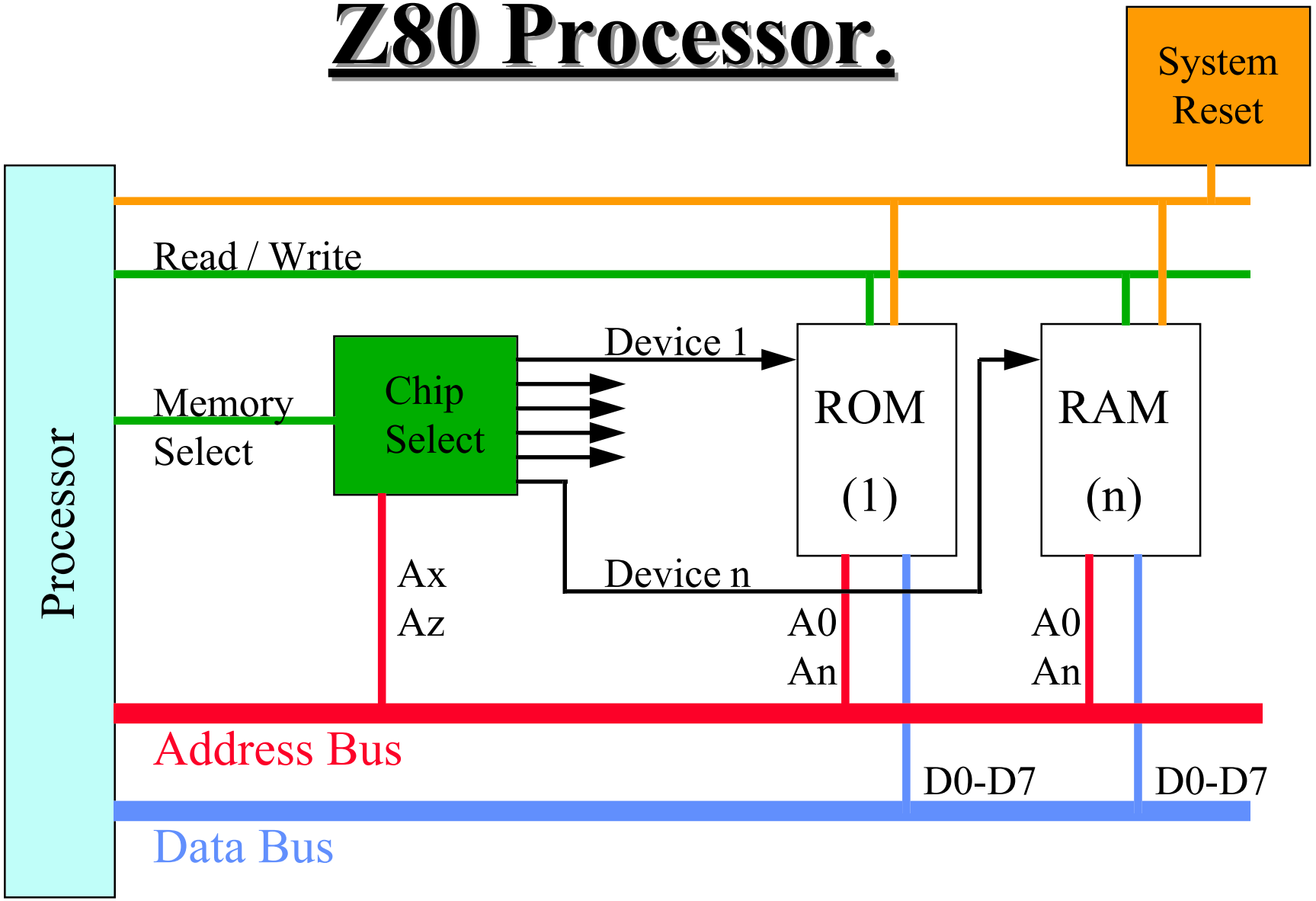
- What are our Pic Processor interfaces ?
- Eight default Output Port Lines.
- Eight default Input Port Lines.
- Four Analogue to Digital conversion Input channels.
- The Input and Output ports can be reconfigured if required.

Generic Processors.

- Zero Bit indicates the status of the last ALU calculation was ZERO.
- In the PIC (Status Register, Bit 2 = Zero Bit)
- Carry Bit indicates a Mathematical Overflow occurred during the last ALU calculation.
- If last operation was Addition it is a Digit Carry.
- If last operation was Subtraction or Compare then
  - IF Carry SET Value was Lower.
  - IF Carry CLEAR Value was Higher or Same.
- In the PIC (Status Register, Bit 0 = Carry Bit)

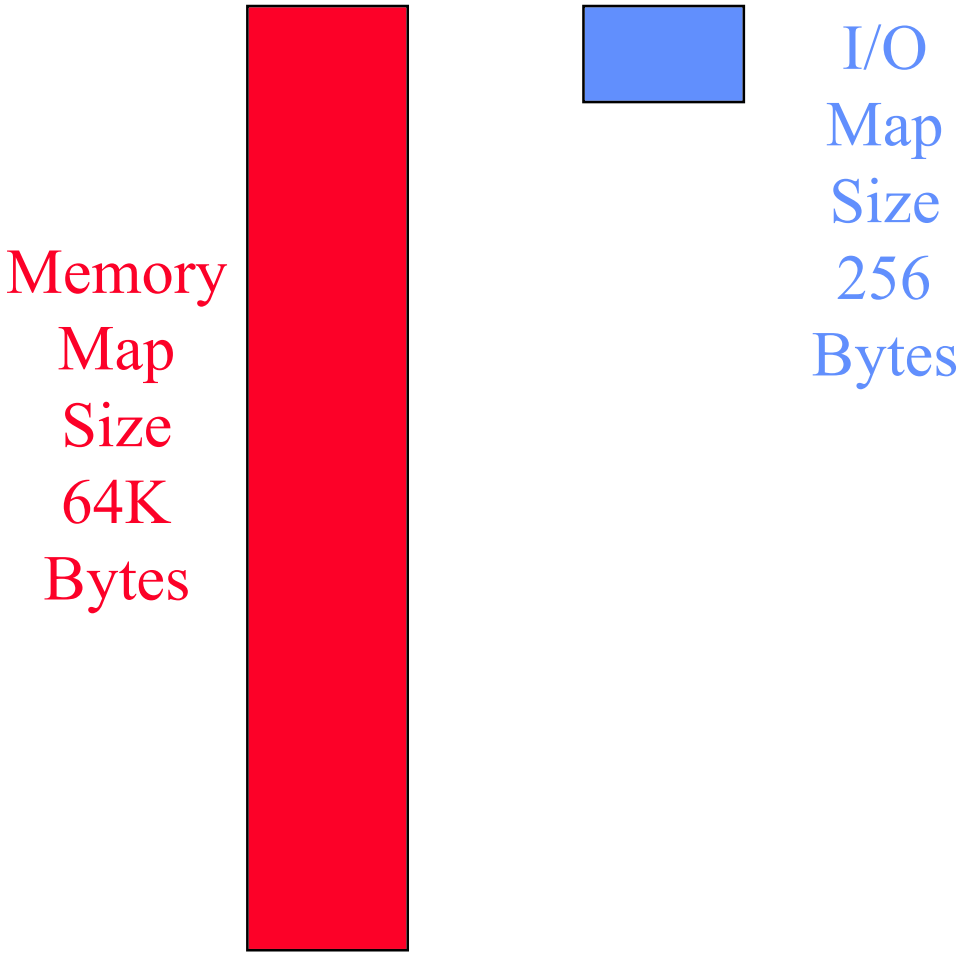
Z80 Processor

Z80 Processor.

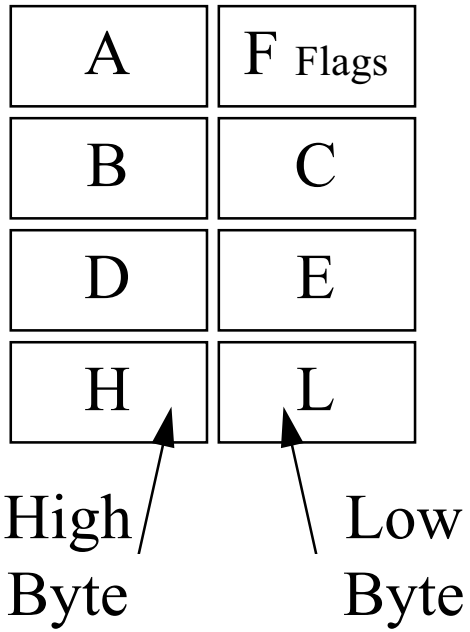


ROM could also be → PROM → EPROM → EEPROM

Z80 Processor.



The Basic Registers

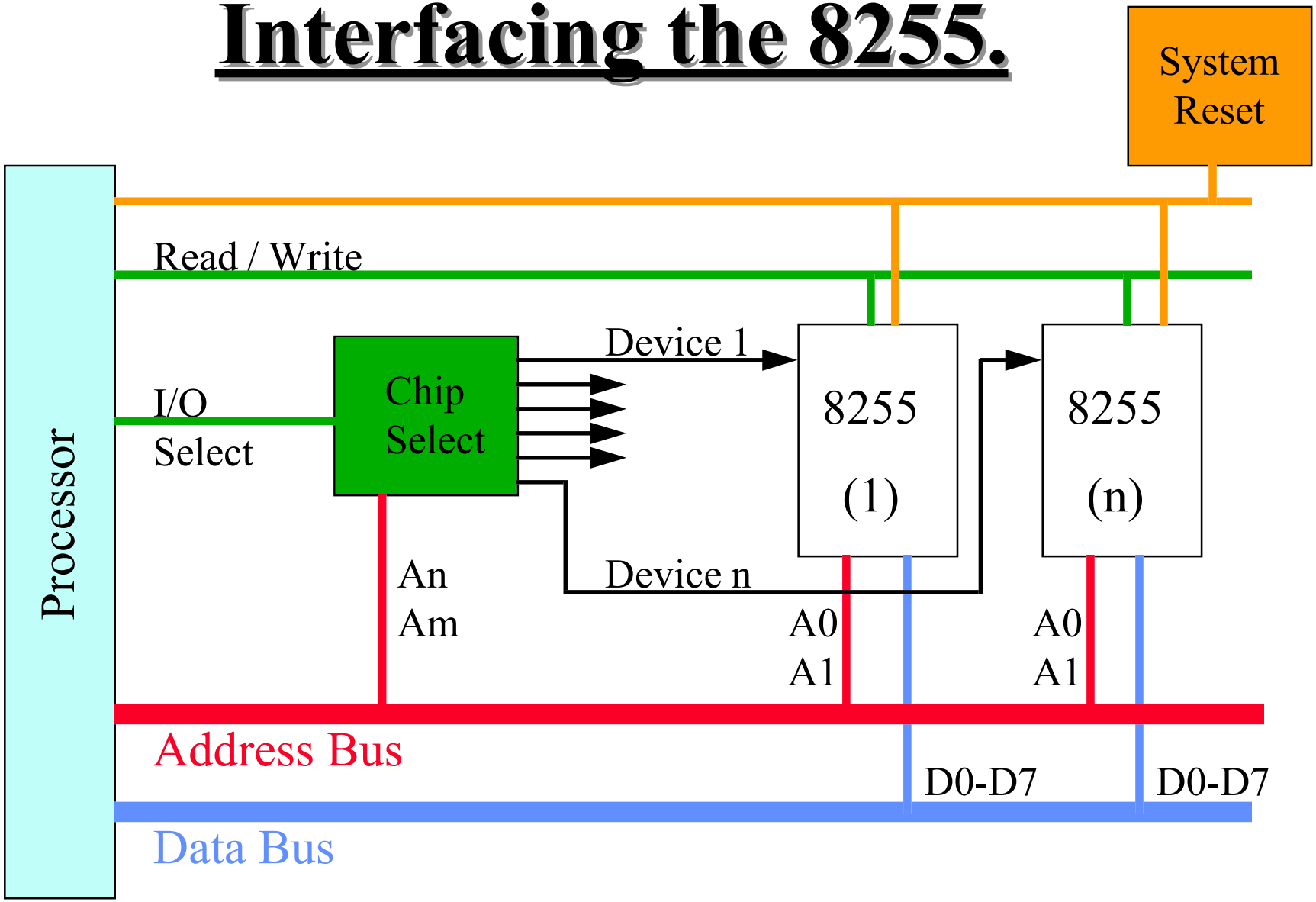


Actually two Sets

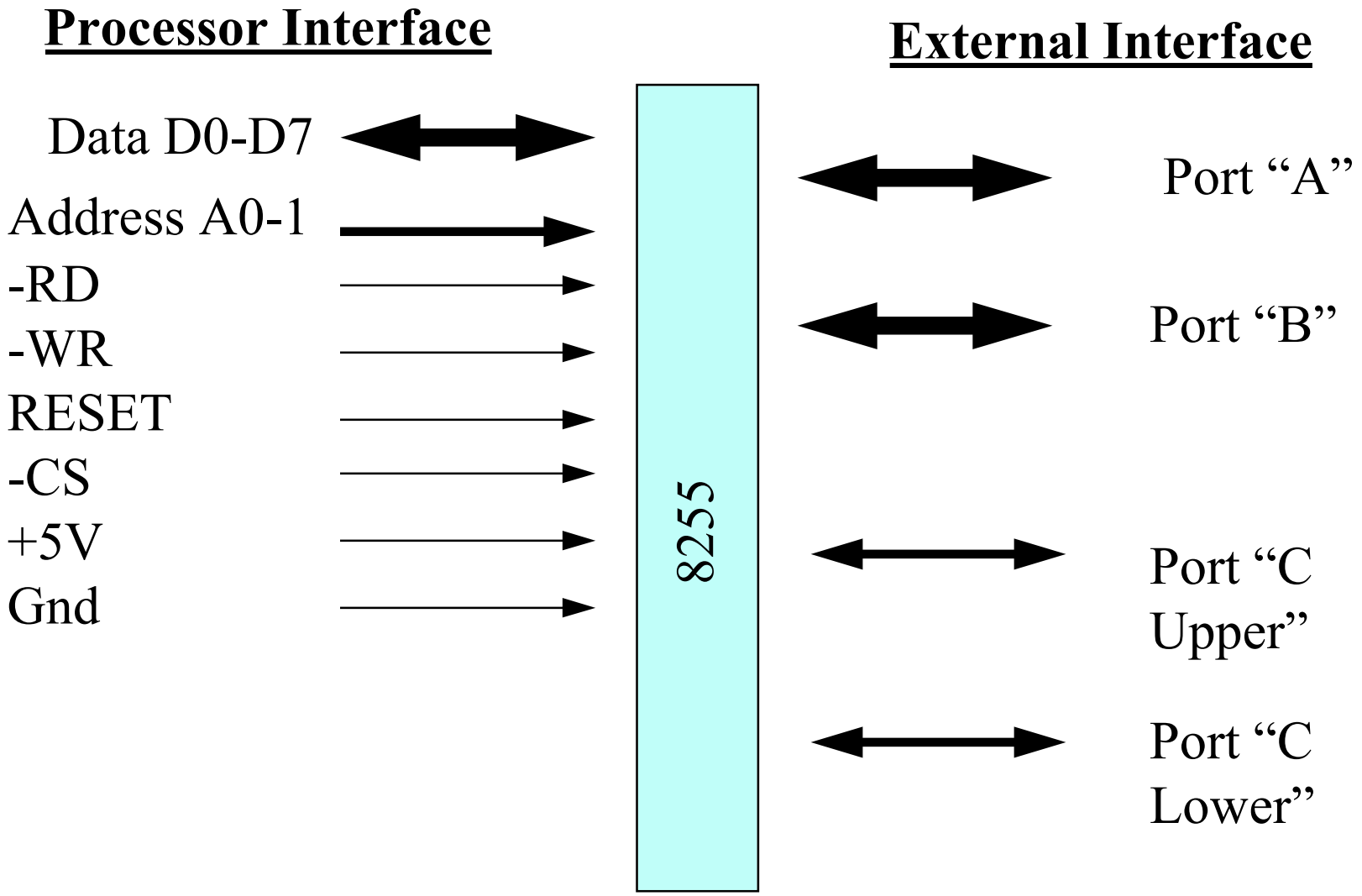
Address Space and Registers

Interfacing Using the 8255

Interfacing the 8255.



Interfacing the 8255.



Signals & Data and their direction.

ASS.

Assembler

Simulator

System

ASS and its IDE.

- ASS is a set of individual stand alone applications that can be used to develop software for virtually any micro-processor / computer system.
- ASS has an Integrated Development Environment started by the programme SYS or XASSYS.

ASS Configuration.

- ASS has most of its default configuration settings loaded into the file ASS.INI and the file is created by either the programme INIT.EXE or the IDE.
- The IDE can supports and manages up to ten sets of configuration environments through the SAVE and RESTORE options.
- The main purpose of the default settings enables rapid operation of the system.
- For example once a default filename has been set up you just press enter to accept the default.

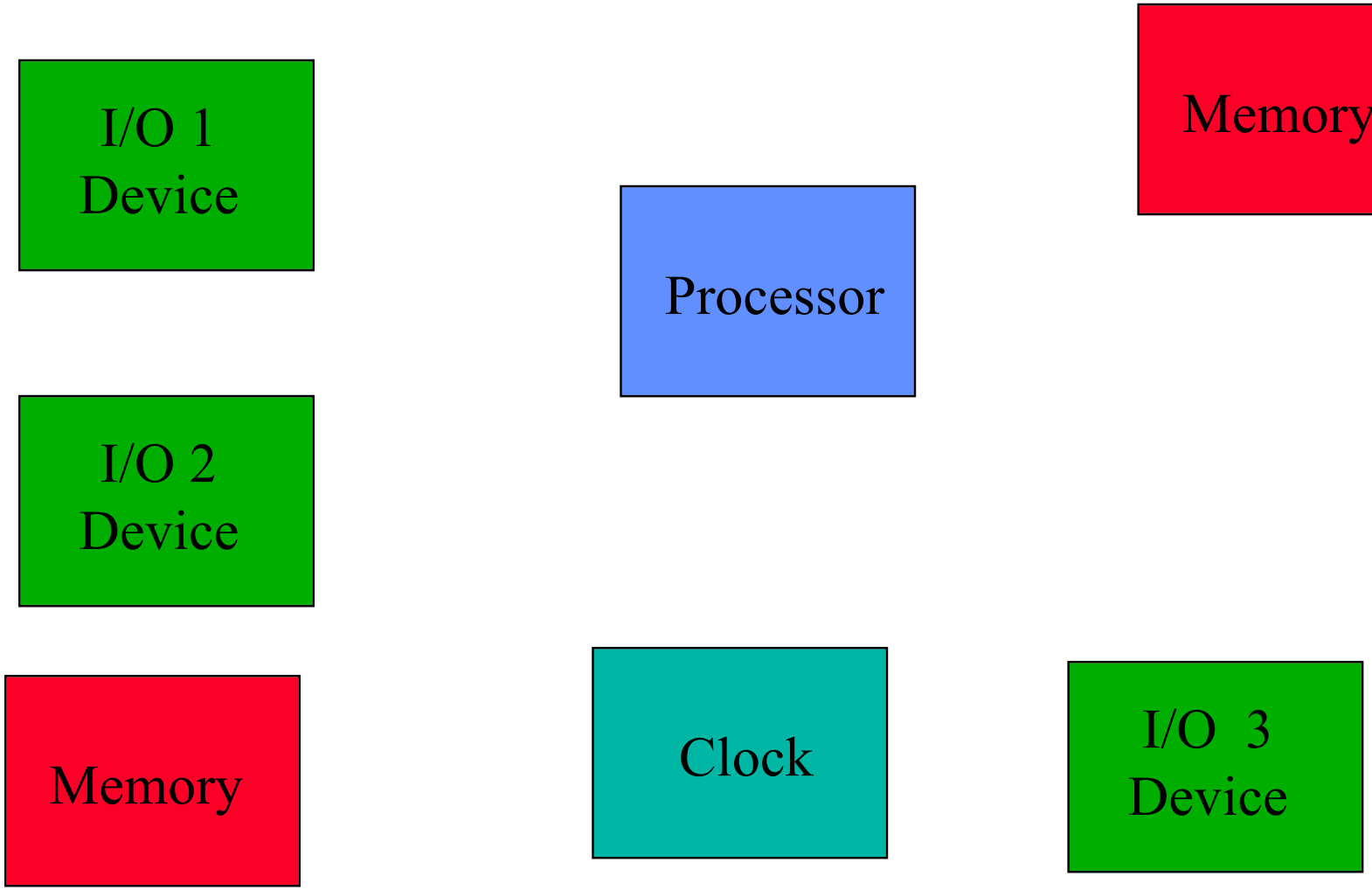
ASS Configuration.

- Example configuration options
  - Default names and file extensions.
  - Preference Text editor.
  - Workspace allocation.
  - Assembler Processor type.
  - Output file format.
  - Simulator Processor type.
  - Simulator Hardware configuration.
  - Dis-Assembler Processor type.

Simulator Configuration.

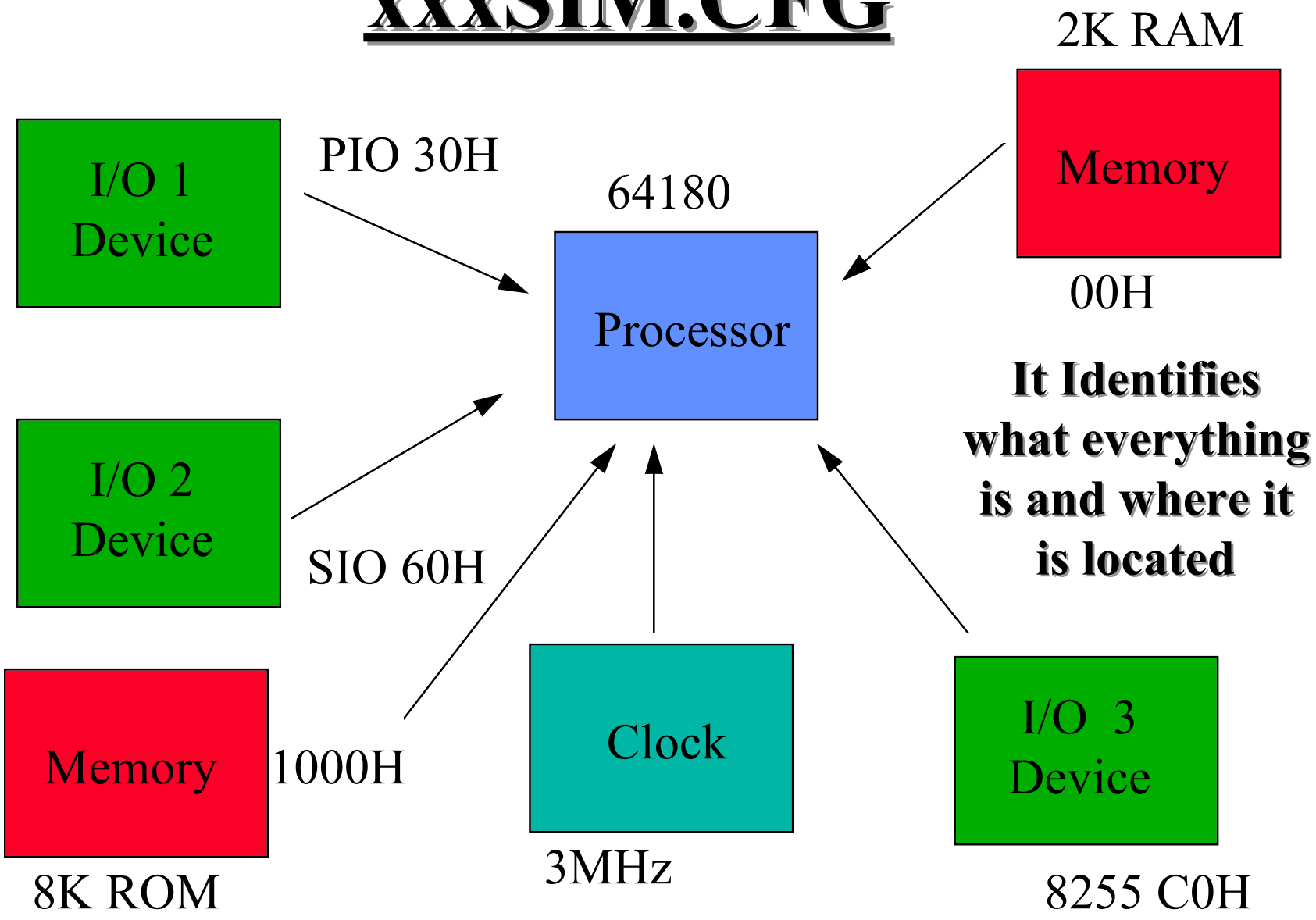
- Example configuration options
  - Processor type and variant options.
  - Processor Speed of operation.
  - Allocation of Memory and Type (RW, RO ...).
  - Physical mapping of Memory Locations.
  - Set Diagnostic marker on Memory Locations.
  - Selection of Peripheral Devices.
  - Connection of Peripheral Devices.
  - Mapping of Peripheral Devices Locations.
  - Real-time Logging Dis-Assembly options.

xxxSIM.CFG



What does xxxSIM.CFG do.

xxxSIM.CFG



What does xxxSIM.CFG do.

End Slide

Revision Page

Title	BASIC Digital Electronics
Author	R. J. Spriggs
Last Update	08/March/2007
Version	1.19
Edit	0055