BINARY CODED DECIMAL: B.C.D.

- ANOTHER METHOD TO REPRESENT DECIMAL NUMBERS
- USEFUL BECAUSE MANY DIGITAL DEVICES PROCESS + DISPLAY NUMBERS IN TENS

IN **BCD EACH** NUMBER IS DEFINED BY A BINARY CODE OF **4 BITS**.

*** **8 – 4 – 2 – 1** MOST COMMON CODE

8 – 4 – 2 – 1 CODE INDICATES THE **WEIGHT** OF EACH BIT $\mathbf{2^3 – 2^2 – 2^1 – 2^0}$

E.G. 934 = 1001 0011 0100

1001	0011	0100
9	**3**	**4**

FOR **EACH** DIGIT A **BINARY** [NORMAL] **CODE** IS ALLOCATED.

OHER REPRESENTATION FORMS ARE **2-4-2-1** AND **EXCESS-3**

BINARY	**8-4-2-1**	**2-4-2-1**	**EXCESS-3**
0000	**0**	**0**	**NOT USED**
0001	**1**	**1**	**NOT USED**
0010	**2**	**2**	**NOT USED**
0011	**3**	**3**	**0**
0100	**4**	**4**	**1**
0101	**5**	**NOT USED**	**2**
0110	**6**	**NOT USED**	**3**
0111	**7**	**NOT USED**	**4**
1000	**8**	**NOT USED**	**5**
1001	**9**	**NOT USED**	**6**
1010	**NOT USED**	**NOT USED**	**7**
1011	**NOT USED**	**5**	**8**
1100	**NOT USED**	**6**	**9**
1101	**NOT USED**	**7**	**NOT USED**
1110	**NOT USED**	**8**	**NOT USED**
1111	**NOT USED**	**9**	**NOT USED**

- WE WILL USE 8-4-2-1 BCD
- DECIMAL NUMBERS > **9** MAY BE OBTAINED WHEN ADDING **TWO** DECIMAL DIGITS (**RANGE**: 0-18) **I.E.** 0 + 0 ÷ 9 + 9. **ONLY 0→9** HAVE THE CORRECT BCD CODE.
- WE NEED TO **CORRECT** THE OTHERS

DECIMAL	**UNCORECTED** BCD SUM $C'_3\ S'_3\ S'_2\ S'_1\ S'_0$	**CORRECTED** BCD SUM $C_N\ S_3\ S_2\ S_1\ S_0$
0	0 0 0 0	0 0 0 0
⋮	⋮	⋮
9	1 0 0 1	1 0 0 1
10	1 0 1 0	**1 0 0 0 0**
11	1 0 1 1	**1 0 0 0 1**
12	1 1 0 0	**1 0 0 1 0**
13	1 1 0 1	**1 0 0 1 1**
14	1 1 1 0	**1 0 1 0 0**
15	1 1 1 1	**1 0 1 0 1**
16	1 0 0 0 0	**1 0 1 1 0**
17	1 0 0 0 1	**1 0 1 1 1**
18	1 0 0 1 0	**1 1 0 0 0**
19	1 0 0 1 1	**1 1 0 0 1**

- **0→9** ONLY LEGAL CODES

  E.G. 19 = **1 9** = **0001 1001** = **11001**

THUS, FOR SUMS BETWEEN **10 → 18** MUST **SUBTRACT 10** AND **PRODUCE A CARRY**

SUBTRACT 10 = 1010_2 >> **ADD 2's COMPLEMENT = 0110**

4-BIT BCD ADDER

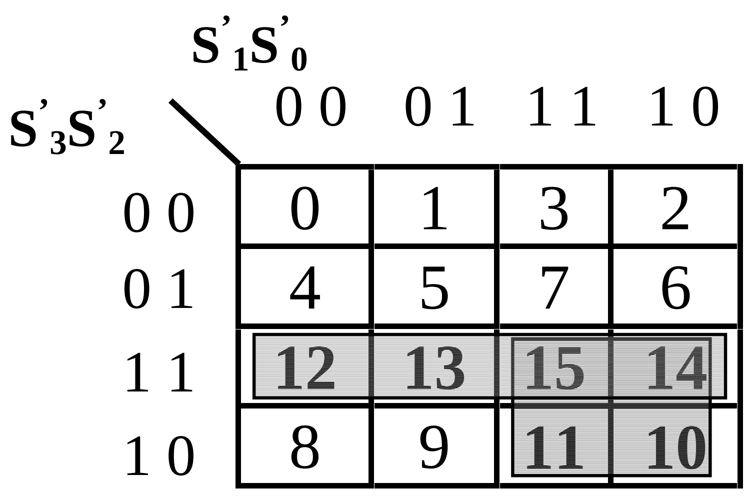
TO ADD TWO DIGITS

FOR SUMS >**9** WE NEED TO ADD 2's COMPLEMENT of 10_{10} TO THE UNCORRECTED RESULT (**$S'_3\ S'_2\ S'_1\ S'_0$**)

CORRECTION IS **ALSO** NEEDED WHEN A **CARRY OUT (C'_3)** IS GENERATED [NUMBERS **16** → **18**]

>>>> **A DECODER IS REQUIRED TO DETECT WHEN CARRY OUT (C_N) TO THE NEXT STAGE IS NEEDED**

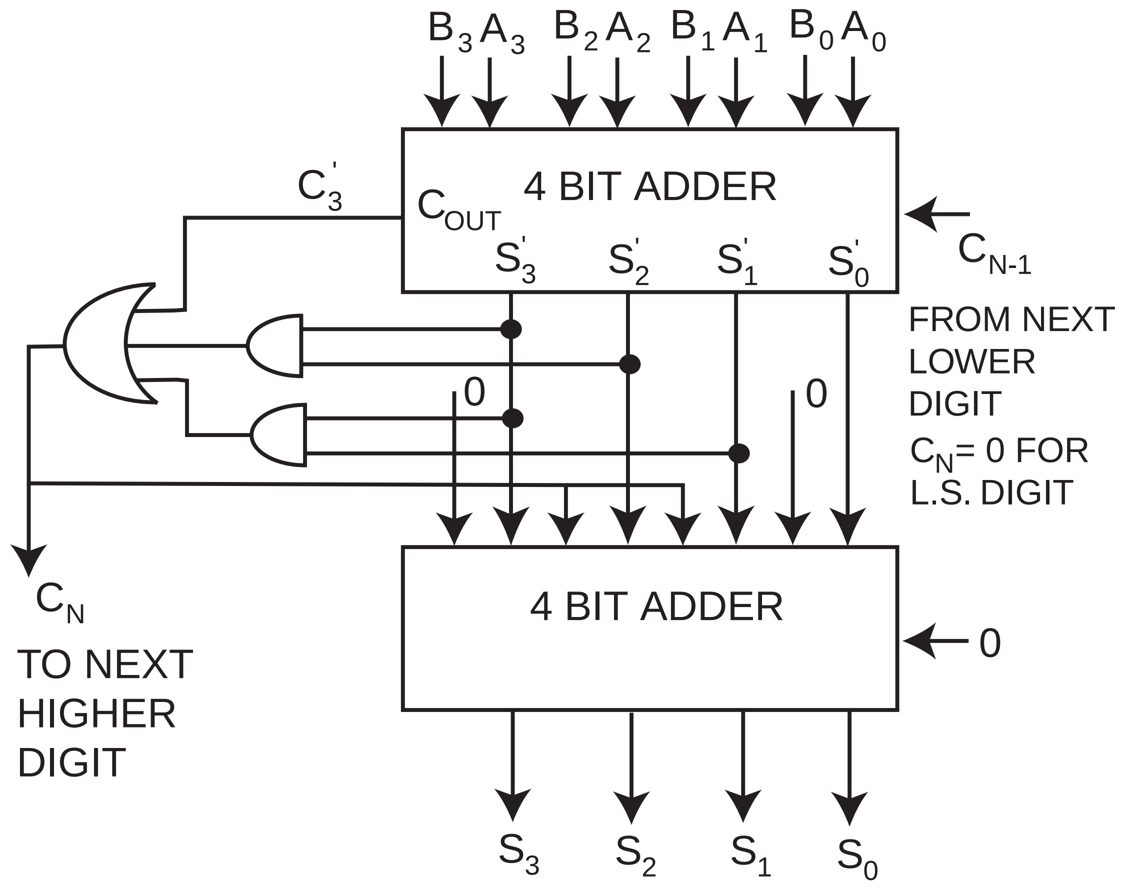
K-MAP FOR **C_N**



>>> **C_N** $= C'_3 + S'_3S'_2 + S'_3S'_1$

TO IMPLEMENT A 4_BIT BCD ADDER WE NEDD **TWO 4-BIT FULL ADDERS, ONE TO ADD TWO 4-BIT BCD NUMBERS** AND THE **OTHER FULL ADDER TO ADD 2's COMPLEMENT OF 10_{10}** TO THE RESULT **IF $C_N = 1$**

ALSO WE NEED **2 AND GATES** AND **ONE OR** GATE TO GENERATE **C_N**



- ADD **0110** WHEN **C_N=1**
- ADD **0000** WHEN **C_N=0**

BCD SUBTRACTION

9’s COMPLEMENT

THE **9’s COMPLEMENT** OF A **DECIMAL NUMBER** IS FOUND BY **SUBTRACTING EACH DIGIT** IN THE NUMBER FROM **9**

DECIMAL DIGIT	**9’s COMPLEMENT**
0	9
1	8
2	7
⋮	⋮
9	0

E.G. 9’s COMPLEMENT of 28 = **99** –28
= 71

9’s COMPLEMENT of 562 = **999** –562
= 437

SUBTRACTION OF A **SMALLER DECIMAL** NUMBER FROM A **LARGER** ONE CAN BE DONE BY **ADDING** THE **9’s COMPLEMENT** OF THE **SMALLER** NUMBER TO THE **LARGER** NUMBER AND THEN **ADDING** THE **CARRY** TO THE RESULT (**END AROUND CARRY**).

WHEN **SUBTRACTING** A **LARGER** NUMBER FROM A **SMALLER** ONE THERE IS **NO CARRY** AND THE **RESULT** IS IN **9's COMPLEMENT FORM** AND **NEGTIVE**.

EXAMPLES:

$$\begin{array}{llcrl}
\textbf{(a)} & +8 & & +8 & \\
 & \underline{-3} & & \underline{+6} & \leftarrow \textbf{9's COMP}.\text{ OF 3} \\
 & \ \ 5 & (1) & 4 & \\
 & & \hookrightarrow & +\mathbf{1} & \text{END AROUND CARRY} \\
 & & & \overline{\ \ 5} &
\end{array}$$

$$\begin{array}{llcrl}
\textbf{(b)} & 54 & & 54 & \\
 & \underline{-21} & & \underline{78} & \leftarrow \textbf{9's COMP}.\text{ OF 3} \\
 & 33 & (1) & 32 & \\
 & & \hookrightarrow & \underline{+\mathbf{1}} & \text{END AROUND CARRY} \\
 & & & 33 &
\end{array}$$

$$\begin{array}{llrl}
\textbf{(c)} & 15 & 15 & \\
 & \underline{-28} & \underline{+71} & \leftarrow \textbf{9's COMP}.\text{ OF 3} \\
 & -13 & 86 & \rightarrow \mathbf{-13}
\end{array}$$

NO CARRY >>> NEGATIVE RESULT

86 – **99** = – **13**

BCD SUBTRACTION

RECALL FOR DECIMAL SUBTRACTION:

A – B = **A + [9's COMPLEMENT** OF **B**]

- **SIMILARLY FOR BCD**

RULES:

(a) ADD 9's COMP. OF **B** TO **A**

(b) IF RESULT > 9, **CORRECT** BY ADDING **0110**

(c) IF MOST SIGNIFICANT CARRY **IS PRODUCED** [i.e. =1] THEN THE **RESULT IS POSITIVE** AND THE **END ARROUND CARRY** MUST BE **ADDED**.

(d) IF MOST SIGNIFICANT CARRY IS **0** [i.e. NO CARRY] THEN THE RESULT IS **NEGATIVE** AND WE GET THE **9's COMP**. OF THE **RESULT**

E.G. 8 – 3 = 8 + [9's COMP. OF 3]
= 8 + 6

$$\begin{array}{lrl}
 & 1000 & \\
 & \underline{0110} & \\
 & 1110 & \leftarrow \textbf{INVALID}\ (>9) \\
 & \underline{0110} & \leftarrow \textbf{CORRECTION} \\
(\mathbf{1}) & 0100 & \\
\llcorner\!\!\longrightarrow & \underline{\mathbf{1}} & \leftarrow \textbf{END AROUND CARRY} \\
 & 0101 = 5 &
\end{array}$$

(b) 3 – 8 = -5

$$\begin{array}{r}
0011 \\
\underline{0001} \\
0100
\end{array}$$

NO CARRY >>> **NEGATIVE**
9's COMP. OF 0100 = **0101 = -5**

(c) 87 –39 >>> 87 + [9's COMP OF 39]

$$\begin{array}{rrr}
8\ 7 & 1000 & 0111 \\
6\ 0 & \underline{0110} & \underline{0000} \\
\textbf{INVALID} \nearrow & 1110 & 0111 \\
 & \underline{0110} & \\
(\mathbf{1}) & 0100 & 0111 \\
\llcorner\!\!\longrightarrow & \longrightarrow & \underline{\mathbf{1}} \\
 & 0100 & 1000 \\
= & \mathbf{4} & \mathbf{8}
\end{array}$$

(d) 18 –72 >>> 18 + [27]

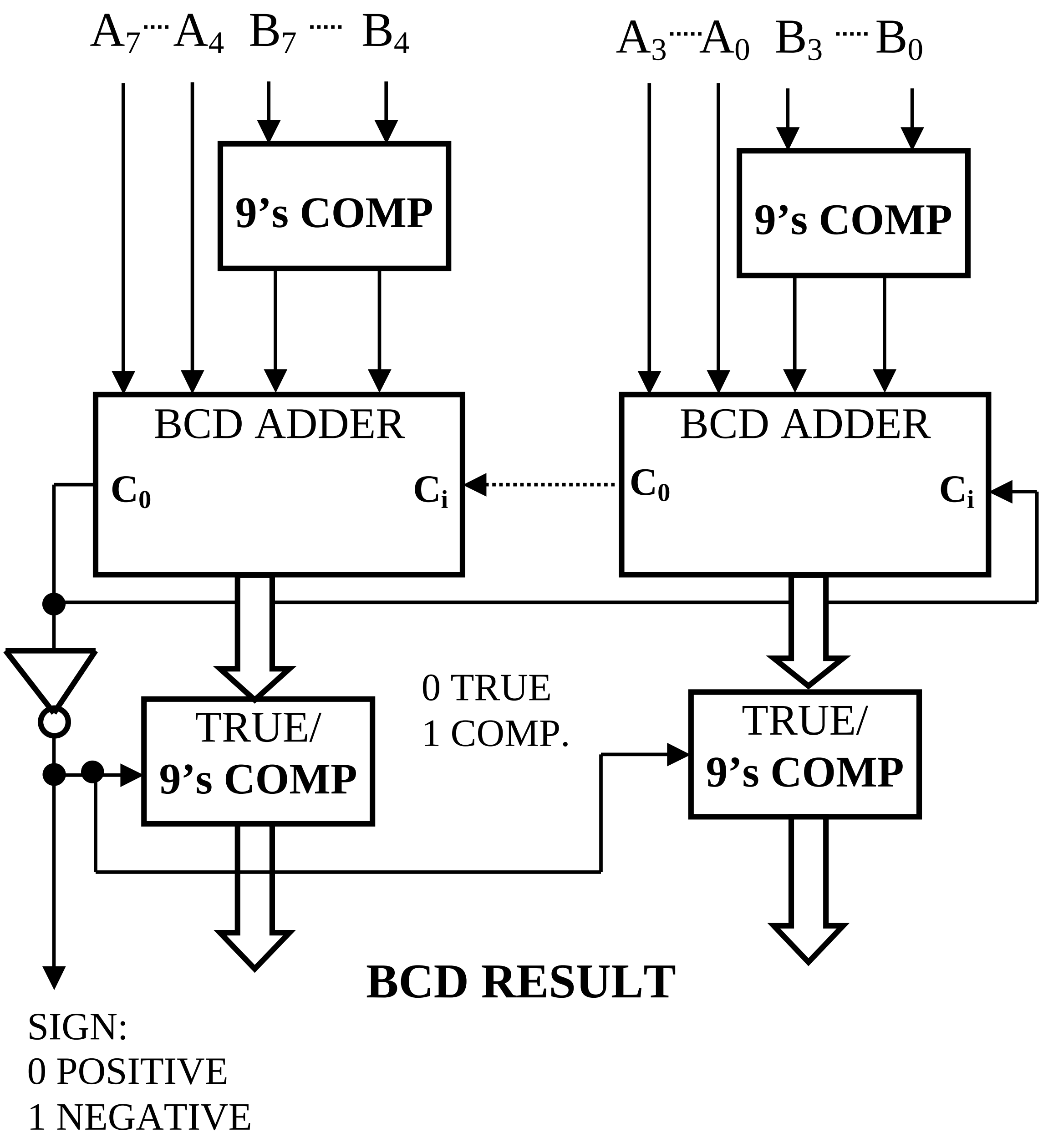
```
      0001          1000
      0010          0111
     ____________________
      0011          1111  → NO CARRY NEGATIVE
      0001 ←┐       0110
     ______ │      ______        CORRECTION
      0100 (1)      0101
        4             5   =  -54
```

OUTPUT IS A **NEGATIVE** NUMBER >> THE RESULT IS IN **9’s COMP.** FORM

(e) 65 – 12 >>> 65 + [87]

```
        0110            0101
        1000            0111
       _____________________
        1110 ──┐        1100 ──→  INVALID
               └────────────────→
        0110            0110      CORRECTION
       ______          ______
    (1) 0100        (1) 0010
     │       1 ←──────┘     1 ←┐  END AROUND
     │  ______         ______  │     CARRY
     │  0101            0011   │
     └─────────────────────────┘

          5               3
```



A_7 ···· A_4 B_7 ····· B_4
A_3 ····· A_0 B_3 ····· B_0
9's COMP
9's COMP
BCD ADDER
C_0
C_i
BCD ADDER
C_0
C_i
TRUE/ 9's COMP
0 TRUE
1 COMP.
TRUE/ 9's COMP
BCD RESULT
SIGN:
0 POSITIVE
1 NEGATIVE

9's COMPLEMENT

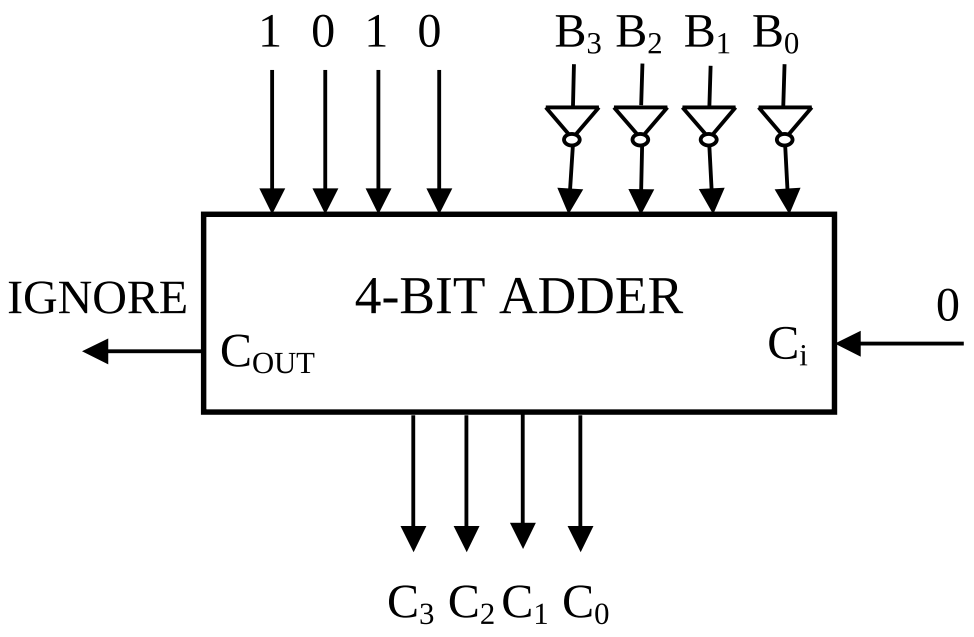
9's COMPLEMENT OF A NUMBER
= 9 – NUMBER

BUT SUBTRACTORS ARE NOT WIDELY AVAILABLE >>> **WE GENERATE** THE **9's COMPLEMENT** BY **ADDING 1010** TO THE **INVERTED** **NUMBER**

BCD DIGIT	$\overline{\textbf{DIGIT}}$	$\overline{\textbf{DIGIT}}$ **+ 1010 = 9's COMP** $C_3\ C_2\ C_1\ C_0$
0 0 0 0	1 1 1 1	1 0 0 1
0 0 0 1	1 1 1 0	1 0 0 0
0 0 1 0	1 1 0 1	0 1 1 1
0 0 1 1	1 1 0 0	0 1 1 0
0 1 0 0	1 0 1 1	0 1 0 1
0 1 0 1	1 0 1 0	0 1 0 0
0 1 1 0	1 0 0 1	0 0 1 1
0 1 1 1	1 0 0 0	0 0 1 0
1 0 0 0	0 1 1 1	0 0 0 1
1 0 0 1	0 1 1 0	0 0 0 0

WE IGNORE THE CARRY OUT

EXERCISE: DESIGN A **9's COMPLEMENT GENERATOR** FOR A BCD DIGIT:



OCTAL & HEXADECIMAL NUMBERS

TWO MORE NUMBER SYSTEMS

OCTAL >>> **BASE 8** **E.G.** 3_8

HEXADECIMAL >>> **BASE 16** **E.G.** 5_{16}

THE **OCTAL** NUMBER SYSTEM IS COMPOSED OF **8 DIGITS**:

0, 1, 2, 3, 4, 5, 6, 7

TO COUNT ABOVE 7, WE BEGIN ANOTHER **COLUMN** AND **START OVER**:

10, 11, 12, 13, 14, 15, 16, 17, 20, 21,…

>>> COUNTING IN **OCTAL** IS SAME AS COUNTING IN **DECIMAL** EXCEPT ANY NUMBERS WITH **8 OR 9** WHICH ARE **OMITTED**.

THE **HEXADECIMAL** SYSTEM IS COMPOSED OF **16 DIGITS** AND **CHARACTERS. EACH HEXADECIMAL CHARACTER** REPRESENTS a **4-BIT BINARY NUMBER**

DECIMAL	**OCTAL**	**HEXADECIMAL**
0	0	0
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	10	8
9	11	9
10	12	**A**
11	13	**B**
12	14	**C**
13	15	**D**
14	16	**E**
15	17	**F**
16	20	10

17	21	11
18	22	12
19	23	13
20	24	14
⋮	⋮	⋮
24	30	18
⋮	⋮	⋮
32	40	20

DECIMAL >>> OCTAL

DIVIDE BY 8, GET **REMAINDERS** AND **READ UP**.

$$\begin{array}{lll} 123_{10} : 8 & & \\ 15 : 8 & 3 & \Big\uparrow \\ 1 : 8 & 7 & \text{READ UP} \\ 0 & 1 & \end{array}$$

>>> $\mathbf{123_{10} = 173_8}$

(b)
$$\begin{array}{rcl} 100_{10} : 8 & & \\ 12 : 8 & 4 & \uparrow \\ 1 : 8 & 4 & \Big| \text{ READ UP} \\ 0 & 1 & \Big| \end{array}$$

>>> $\mathbf{100_{10} = 144_8}$

OCTAL >>> DECIMAL

MULTIPLY BY SUCCESIVE **POWERS OF 8**

$\mathbf{173_8} = 1 \times 8^2 + 7 \times 8^1 + 3 \times 8^0$
$= 64 + 56 + 3 = \mathbf{123_{10}}$

NOTE: WE BEGIN WITH **RIGHT-MOST POWER OF 8** AS 8^0

DECIMAL >>> HEXADECIMAL

DIVIDE BY 16, GET REMAINDERS AND READ UP

NOTE: REMAINDER = **13**
>>> USE CHARACTER **D**

E.G.

$$\begin{tikzcd}
123_{10} : 16 \arrow[dr] & \\
7 \ : \ 16 \arrow[dr] & (11) \leftarrow \mathbf{B} \\
0 & 7
\end{tikzcd}$$

>>> $\mathbf{123_{10} = 7B_{16}}$

HEXADECIMAL >>> DECIMAL

MULTIPLY BY SUCCESIVE POWERS OF 16

NOTE: **WE CONVERT** A CHARACTER TO ITS **DECIMAL VALUE** BEFORE WE **MULTIPLY**

E.G. $\mathbf{6A_{16}} = 6 \text{ x } 16^1 + A \text{ x } 16^0$

$= 6 \text{ x } 16^1 + 10 \text{ x } 16^0$

$= 96 + 10 = \mathbf{106_{10}}$

SEQUENTIAL LOGIC

UNTIL NOW WE HAVE DEALT WITH **COMBINATIONAL LOGIC** – WHERE THE **OUTPUT DEPENDED** ON THE PRESENT **COMBINATION OF INPUTS**

IN **SEQUENTIAL LOGIC** THE OUTPUT **DEPENDS ON** THE **PAST INPUTS** AS WELL AS **PRESENT INPUTS**

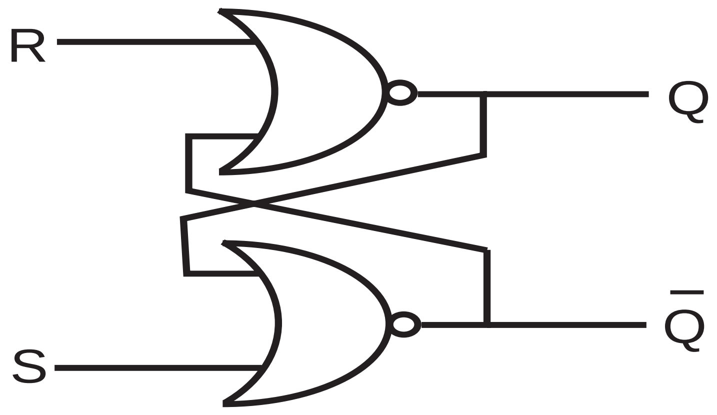
E.G. **MEMORY DEVICES** USED TO REMEMBER **PAST VALUES** OF INPUTS

FLIP-FLOPS

DIGITAL MEMORIES, COUNTERS AND SHIFT REGISTERS ARE BASED ON THE **FLIP-FLOP**. THIS IS A DEVICE THAT CAN **STORE 2 STATES 0 OR 1**

R-S FLIP-FLOP

THE BASIC RS [**R**ESET- **S**ET] FLIP-FLOP CONSISTS OF **2 NOR GATES** WITH THE OUTPUTS **CROSS-COUPLED** TO THE INPUTS



R	**S**	**Q**	$\overline{\mathbf{Q}}$	
0	0	**NO CHANGE**		
0	1	1	0	SET
1	0	0	1	RESET
1	1	**NOT ALLOWED**		

START WITH A KNOWN STATE

R = 0, S = 1 >>> **Q = 1** SET
R = 1, S = 0 >>> **Q = 0** RESET

WHEN **R=S=0**, **Q** AND **$\overline{Q}$** REMAIN AS THEY WHERE **BEFORE THE INPUT WAS CHANGED**

R=S=1 IS NOT ALLOWED AS IT CAUSES **Q=$\overline{Q}$=0** >>> THEY HAVE TO BE **OPOSITE** TO EACH OTHER

APPLICATION: MEMORY CELL

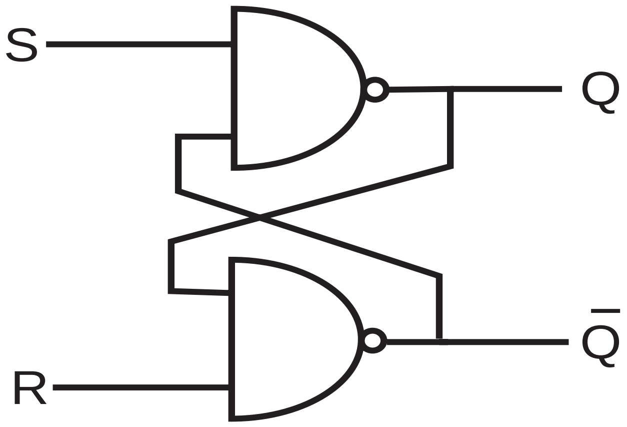
R	**S**	**Q_N**	**$\overline{Q}_N$**	**Q_{N+1}**	**$\overline{Q}_{N+1}$**
0	0	0	1	**0**	**1**
0	0	1	0	**1**	**0**

N = PREVIOUS STAGE (PAST)
N+1 = PRESENT STAGE

THE CIRCUIT IS SAID TO BE **BISTABLE** >>> IT CAN ASSUME **TWO STABLE** STATES.

RS FLIP-FLOP USING NAND GATES

** **TWO CROSS-COUPLED** NAND GATES

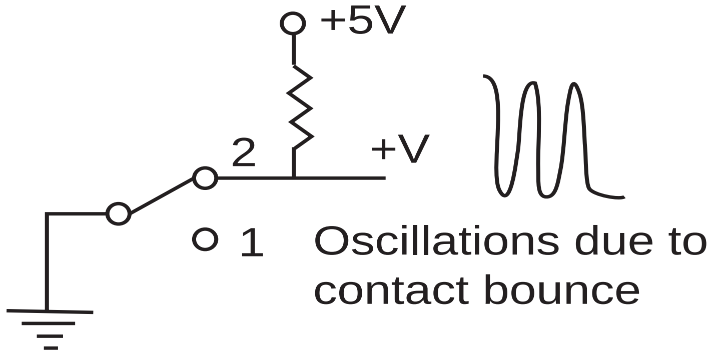


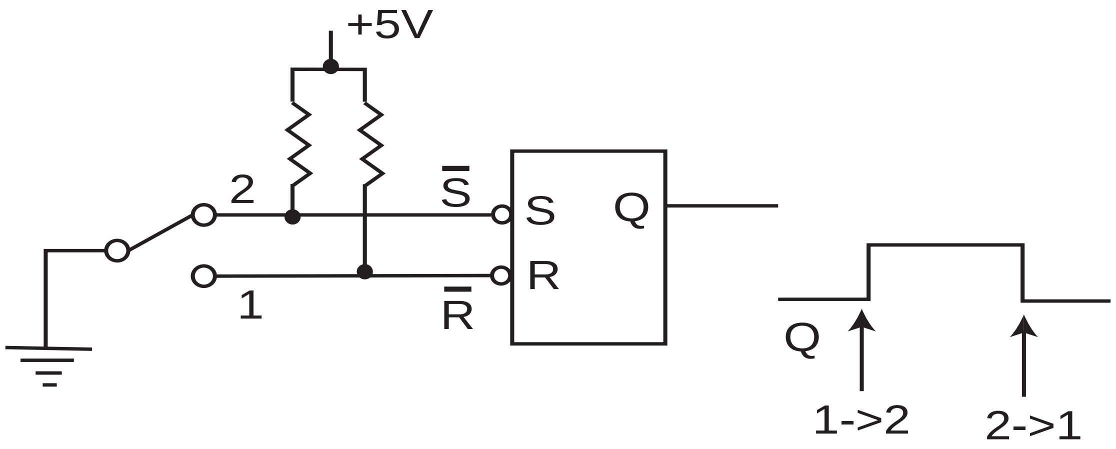
S	**R**	**Q**	**$\overline{Q}$**	
0	0	**NOT ALLOWED**		
0	1	1	0	SET
1	0	0	1	RESET
1	1	**NO CHANGE**		

ACTIVE LOW RS

THE **RS** FLIP-FLOP CAN BE USED TO ELLIMINATE “**CONTACT BOUNCES**” IN SWITCHES.

ELECTRICAL CONTACTS OFTEN PROVIDE **INPUT SIGNALS** TO LOGIC SYSTEMS >>> THESE CONTACTS MAY BE **A SOURCE OF NOISE** DUE TO **THEIR MECHANICAL PROPERTIES**.



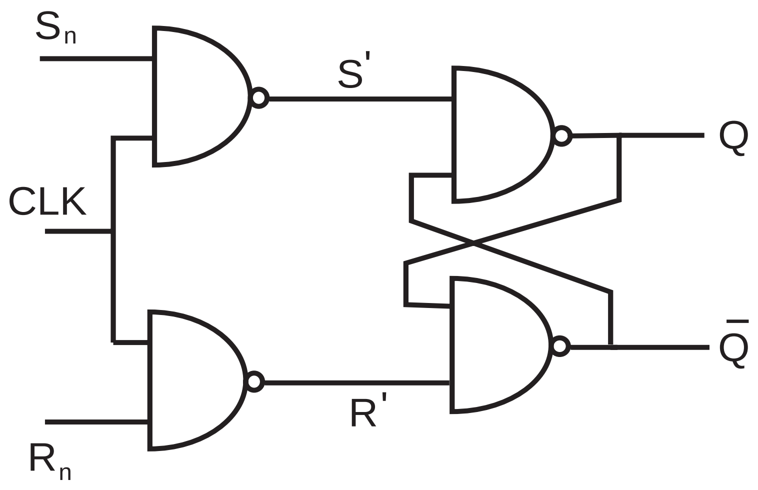


SWITCH MOVES FROM **1 TO 2** Q CHANGES FROM **0 TO 1** AS SOON AS THE FIRST CONTACT IS MADE (**EVEN** IF THE **SWITCH BOUNCES** Q STAYS AT **1**)

WHEN THE SWITCH MOVES FROM **2 TO 1** Q CHANGES FROM **1 TO 0** AS SOON AS THE FIRST CONTACT IS MADE.

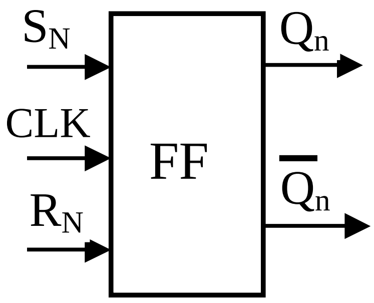
SYNCHRONOUS [CLOCKED] RS FLIP-FLOP

WE OTEN NEED TO **SET** OR **RESET** A **FLIP-FLOP** UNDER THE **CONTROL OF A CLOCK PULSE**



REMEMBER: **S' = R' = 1 NO OUTPUT CHANGE**

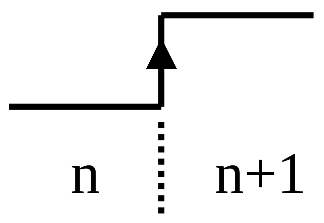
WHEN CLOCK APPLIED



S_N	R_N	Q_{n+1}	$\overline{Q_{n+1}}$
0	0	Q_n	$\overline{Q_n}$
0	1	0	1
1	0	1	0
1	1	**NOT ALLOWED**	

n = TIME **BEFORE** THE **CLOCK PULSE**
n+1 = TIME **AFTER** THE **CLOCK PULSE**

HERE THE FLIP-FLOP IS **SET** OR **RESET** UNDER THE **CONTROL OF THE CLOCK PULSE**

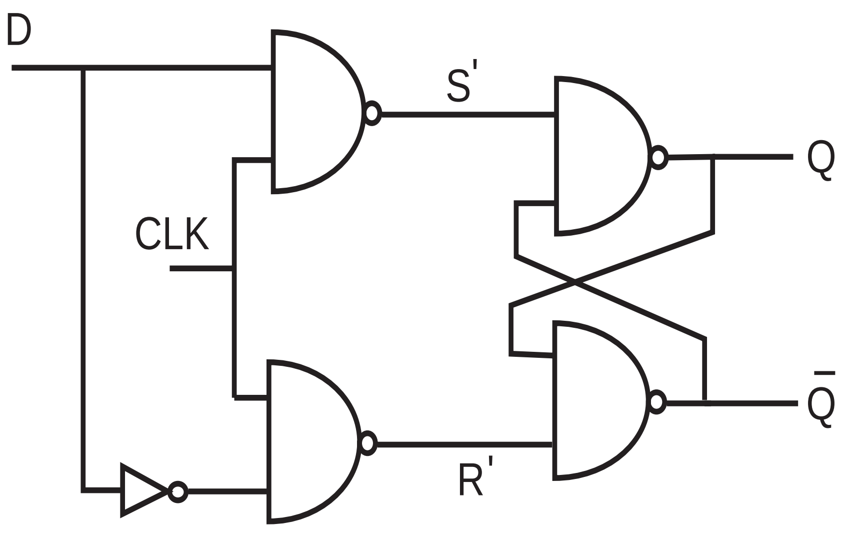


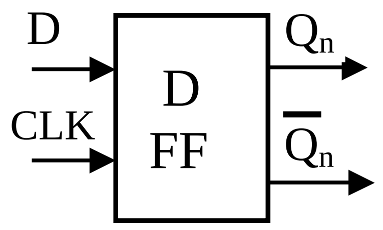
CLOCK PULSE

WHEN CLK IS AT “0” NO CHANGE IN OUTPUTS **IRRESPECTIVE** OF THE STATE OF **THE INPUTS** (S,R).

D FLIP-FLOP

D [**DELAY**] FLIP-FLOP IS A SIMPLE **EXTENSION** OF THE **CLOCKED RS** FLIP-FLOP WITH THE **D** INPUT CONNECTED TO THE “**SET**” INPUT AND $\overline{\mathbf{D}}$ CONNECTED TO THE “**RESET**” INPUT.





D	$\mathbf{Q_{n+1}}$	$\mathbf{\overline{Q}_{n+1}}$
0	0	1
1	1	0

D=1 >>> **SET**
D=0 >> **RESET**

ACTS AS **1-BIT DTORAGE ELEMENT**
INPUT TRANSFERRED TO **OUTPUT**
>>>> CALLED A **LATCH**.

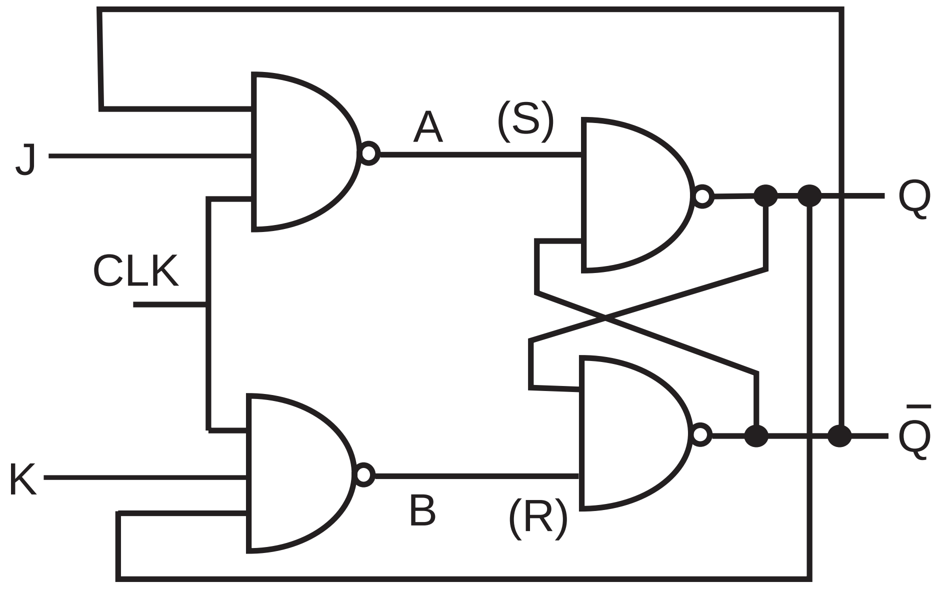
J-K FLIP-FLOP

J-K FLIP-FLOP ELIMINATES NON-DEFINED (NOT ALLOWED) STATE THAT OCCURRED IN THE RS FLIP-FLOP (R=S=1).

MOST VERSATILE AND WIDELY USED.

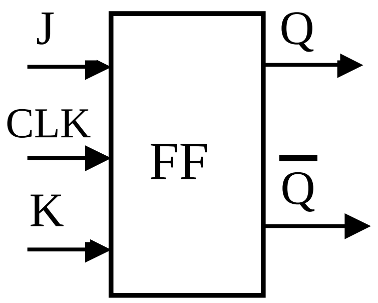
Q_n	**J**	**K**	Q_{n+1}	
0	**0**	**0**	0	NO CHANGE
0	0	1	0	RESET
0	1	0	1	SET
0	**1**	**1**	1	TOGGLE
1	**0**	**0**	1	NO CHANGE
1	0	1	0	RESET
1	1	0	1	SET
1	**1**	**1**	0	TOGGLE

WHEN CLOCK ACTIVE



DIFFERS TO **R-S** BY USING **ADDITIONAL FEEDBACK**

WHEN CLOCK ACTIVE



J	**K**	**Q_{n+1}**	
0	0	Q_n	NO CHANGE
0	1	0	RESET
1	0	1	SET
1	1	$\overline{Qn}$	TOGGLE

SAME AS **R-S FLIP-FLOP** EXCEPT WHEN **J=K=1**.

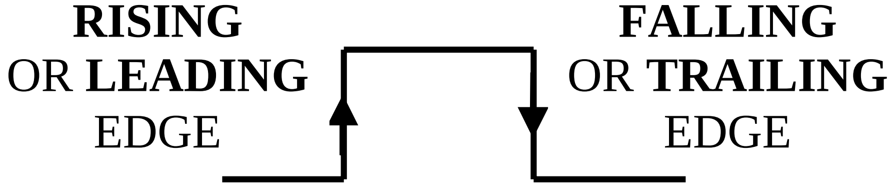
E.G. Q=0 [$\overline{Q}$=1] >>> RESET
THEN CLOCK GOES FROM **0 → 1**

>>> A→0 >>> **Q=1**
B→1 >>> **$\overline{Q}$=1**
>>> **SET**

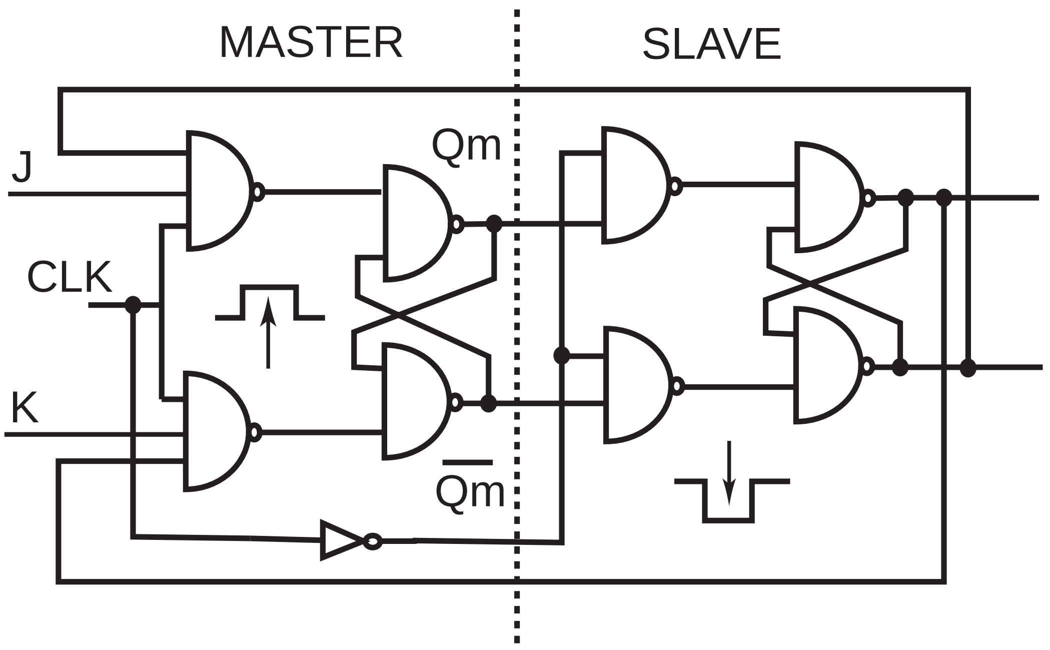
MASTER SLAVE J-K FLIP-FLOP

PROBLEM WITH J-K FLIP-FLOP >>> **OUTPUTS MAY CHANGE** IF THE **DATA INPUTS** [J,K,Q,$\overline{Q}$] **CHANGE WHILE THE CLOCK IS HIGH**

>>> WE NEED TO MAKE SURE THAT **Q** AND **$\overline{Q}$ DO NOT ASSUME** THEIR NEW VALUES UNTIL **AFTER THE TRAILING EDGE** ON THE CLOCK PULSE i.e. WHEN **PULSE → 0**



WE DO THIS BY USING A **MASTER-SLAVE** [M-S] FLIP-FLOP >>> **DIVIDE** THE FLIP-FLOP INTO **MASTER** + **SLAVE** FLIP-FLOP.



WHEN THE CLOCK PULSE IS **HIGH** THE J-K INPUTS ARE APPLIED TO THE **MASTER FLIP-FLOP** AND THE **SLAVE INPUT** IS NOT AFFECTED ($\overline{CLK}$=0)

WHEN THE CLOCK PULSE **FALLS TO ZERO**, THE **DATA FROM THE MASTER** IS APPLIED TO **THE SLAVE INPUTS** AND THE OUTPUTS Q AND $\overline{Q}$ GET THEIR NEW VALUES.

ANOTHER RESTRICTION: THE **INPUTS J AND K DO NOT CHANGE** WHILE **CLK=1**

A **R-S** MASTER SLAVE FLIP-FLOP MAY ALSO BE CONSTRUCTEDIN A SIMILAR FASHION, THE ONLY DIFFERENCE IS THAT WHEN **THE TWO INPUTS J=K=1** THE OUTPUT **Q** CHANGES IS STATE ON RECEIPT OF THE CLOCK PULSE i.e. THE FLIP-FLOP **TOGGLES**.

J	**K**	$\mathbf{Q_{n+1}}$	
0	0	$\mathbf{Q_n}$	UNCAHGED
0	1	0	RESET
1	0	1	SET
1	1	$\overline{\mathbf{Q_n}}$	TOGGLE

DIFFERENT APPROACH:

WE WILL NOW LOOK AT THE J-K FLIP FLOP IN A SLIGHTLY DIFFERENT WAY

IF $\mathbf{Q_n = 0}$ AND **J = 0** >>> $\mathbf{Q_{n+1} = 0}$
REGARDLESS OF **K**

IF $\mathbf{Q_n = 0}$ AND **J = 1** >>> $\mathbf{Q_{n+1} = 1}$
REGARDLESS OF **K**

IF $\mathbf{Q_n = 1}$ AND **K = 0** >>> $\mathbf{Q_{n+1} = 1}$
REGARDLESS OF **J**

IF $\mathbf{Q_n = 1}$ AND **K = 1** >>> $\mathbf{Q_{n+1} = 0}$
REGARDLESS OF **J**

J	**K**	$\mathbf{Q_n}$	$\mathbf{Q_{n+1}}$	$\overline{\mathbf{Q_{n+1}}}$
0	X	**0**	**0**	1
1	X	**0**	**1**	0
X	**1**	**1**	**0**	1
X	**0**	**1**	**1**	0

"**X**" → DEFINES A **DON'T CARE** TERM
→ USEFUL IN DESIGNING **LOGIC SYSTEMS WITH J-K FLIP FLOPS**

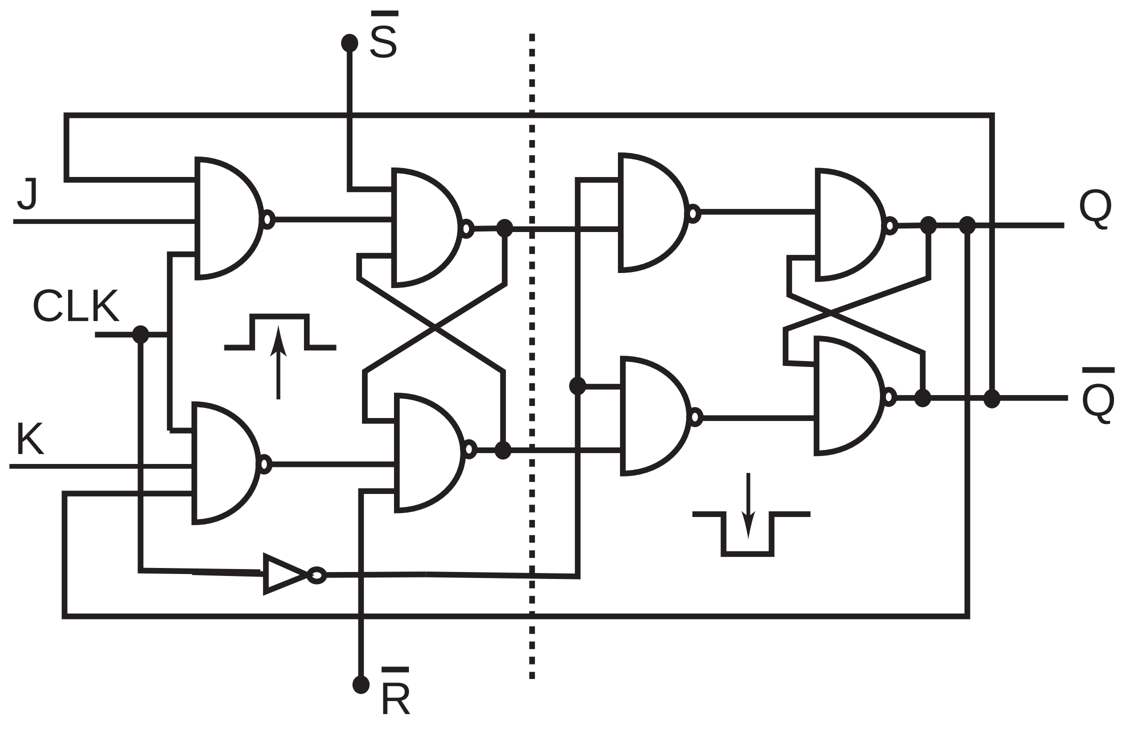
THE MASTER-SLAVE FLIP-FLOPS ARE **SYNCRONOUS DEVICES** >>> THE **OUTPUT CHANGES STATE** AT A **SPECIFIED POINT ON A CLOCK INPUT**.

J-K FLIP-FLOP WITH ASYNCHRONOUS SET AND RESET INPUTS

SEPARATE "SET" AND "RESET" INPUTS MAY BE ADDED TO THE J-K FLIP FLOP.

THESE SET & RESET INPUTS [$\overline{S}$,$\overline{R}$] OVER-RIDE THE J,K INPUTS AND ARE ASYNCHRONOUS [i.e. OVER-RIDE THE CLOCK ALSO].

NOTE: WHEN $\overline{S} = \overline{R} = 1$ >>> WE HAVE NORMAL FLIP-FLOP OPERATION



TRUTH TABLE:

$\overline{S}$	$\overline{R}$	**J**	**K**	Q_{n+1}	
0	0	X	X	NOT ALLOWED	
0	1	X	X	**1**	
1	**0**	X	X	**0**	
1	1	0	0	Q_n	
1	1	0	1	0	**NORMAL**
1	1	1	0	1	**OPERATION**
1	1	1	1	$\overline{Q}_n$	

FLIP-FLOP APPLICATIONS

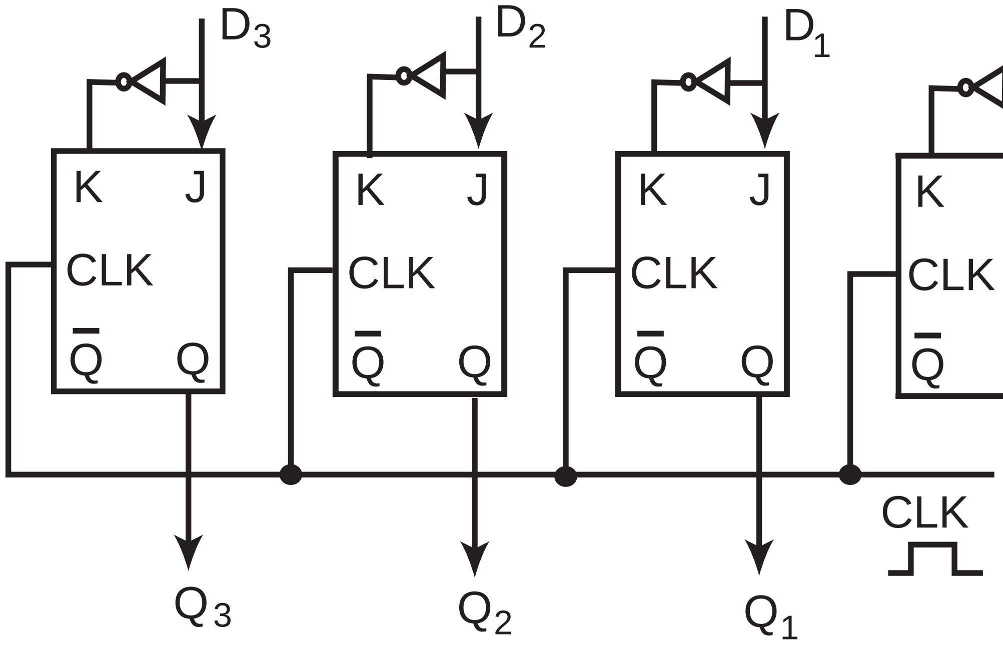
REGISTERS:

GROUPS OF FLIP-FLOPS ARRANGED TO PROVIDE **DATA STORAGE** FOR **SEVERAL DATA BITS**.

1. DATA REGISTER

A **J-K FLIP-FLOP** CAN BE USED AS A **1-BIT EMORY** [**D FLIP-FLOP**] BY APPLYING THE BIT TO BE STORED [**D**] TO **J** AND ITS INVERSE [$\overline{\mathbf{D}}$] TO **K**. AN “**n**” **BIT BINARY WORD** CAN BE STORED BY “**n**” SUCH FLIP-FLOPS >>>> **CALLED A n-BIT REGISTER**.

PARALLEL DATA STORAGE, i.e. **SEVERAL BITS ON PARALLEL LINES STORED SIMULTANEOUSLY**.



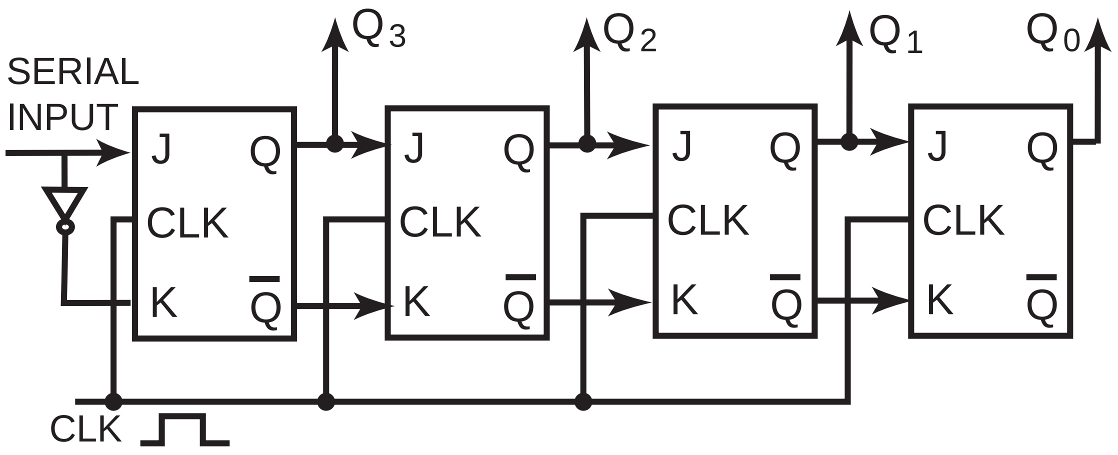
DATA BITS (D_3 – D_0) ARE STORED ON THE RECEIPT OF THE CLOCK PULSE.

2. SHIFT REGISTER

SERIAL DATA TRANSFER [i.e. **ONE BIT AT A TIME**], **A WORD OF N-BITS** READ INTO **N-FLIP-FLOP SHIFT REGISTER**

THE FIRST FLIP-FLOP HOLDS THE **WORD'S MSB** AND IS CONNECTED AS A **D FLIP-FLOP**.

E.G. 4 –BIT SHIFT REGISTER



Q3 = MSB

REMEMBER:

J	**K**	**Q_{n+1}**
0	1	0
1	0	1

OUTPUT FOLLOWS **J**-INPUT WHEN **J** = $\overline{K}$

>>> SHIFT OCCURS WHEN CLOCK APPLIED.

SHIFT REGISTER OPERATION:

LSB OF SERIAL DATA IS APPLIED TO THE SERIAL INPUT [J_3]. THE CLOCK PULSE IS APPLIED AND LSB GOES INTO Q_3 [= J_2]. THE NEXT LSB IS THEN APPLIED TO J_3 AND AFTER A CLOCK PULSE IT IS STORED IN Q_3, WHILE AT THE SAME TIME THE LSB IS SHIFTED INTO Q_2.

AFTER 4 CLOCK PULSES THE SERIAL DATA IS STORED IN THE SHIFT REGISTER.

E.G. STORE THE WORD: **0 1 0 1**

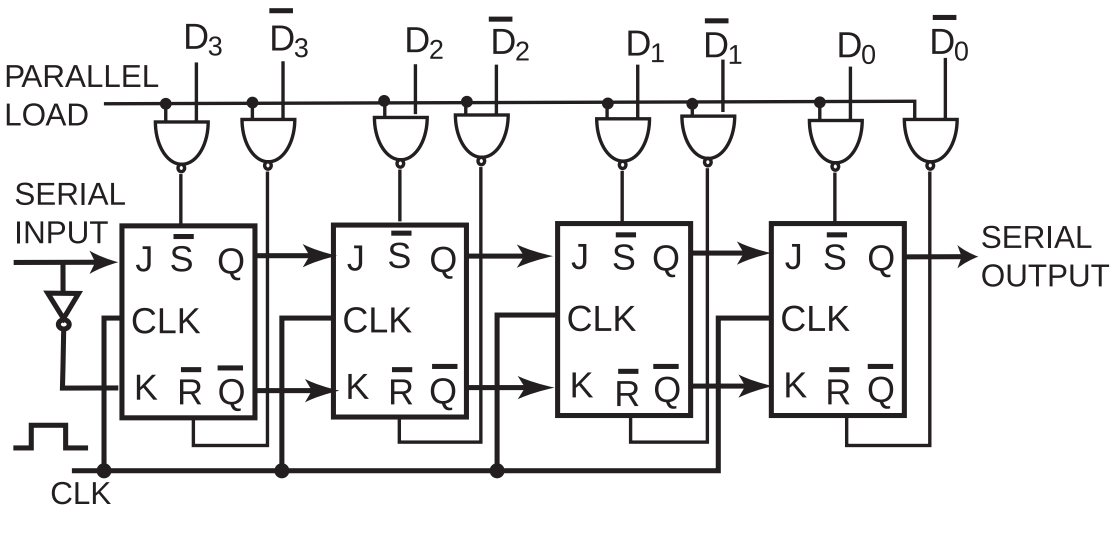
CLK	SERIAL INPUT	Q_3	Q_2	Q_1	Q_0
1	1	1	0	0	0
2	0	0	1	0	0
3	1	1	0	1	0
4	0	**0**	**1**	**0**	**1**

ASSUME THE STORED WORD IS INITIALLY **0 0 0 0**.

3. SHIFT REGISTERS WITH PARALLEL INPUTS [PARALLEL TO SERIAL CONVERTER]

- LOAD BITS IN PARALLEL FORM AND SFIFT THEM IN SERIAL FORM.

- DATA IS LOADED BY APPLYING SIGNALS TO ASYNCHRONOUS SET/RESET INPUTS OF THE J-K FLIP-FLOP.



PARALLEL DATA IS LOADED BY **APPLYING** A CLOCK PULSE TO "PARALLEL LOAD" SIGNAL.

WHEN "PARALLEL LOAD" IS **LOW** THE CIRCUIT OPERATES AS A **NORMAL SHIFT REGISTER** [$\overline{S} = \overline{R} = 1$]

WHEN "PARALLEL LOAD" IS **HIGH** THEN PARALLEL DATA IS **LOADED ASYNCRONOUSLY** [**INDEPENDENT** OF THE **CLOCK**] AND **SHIFTED OUT SYNCHRONOUSLY**.

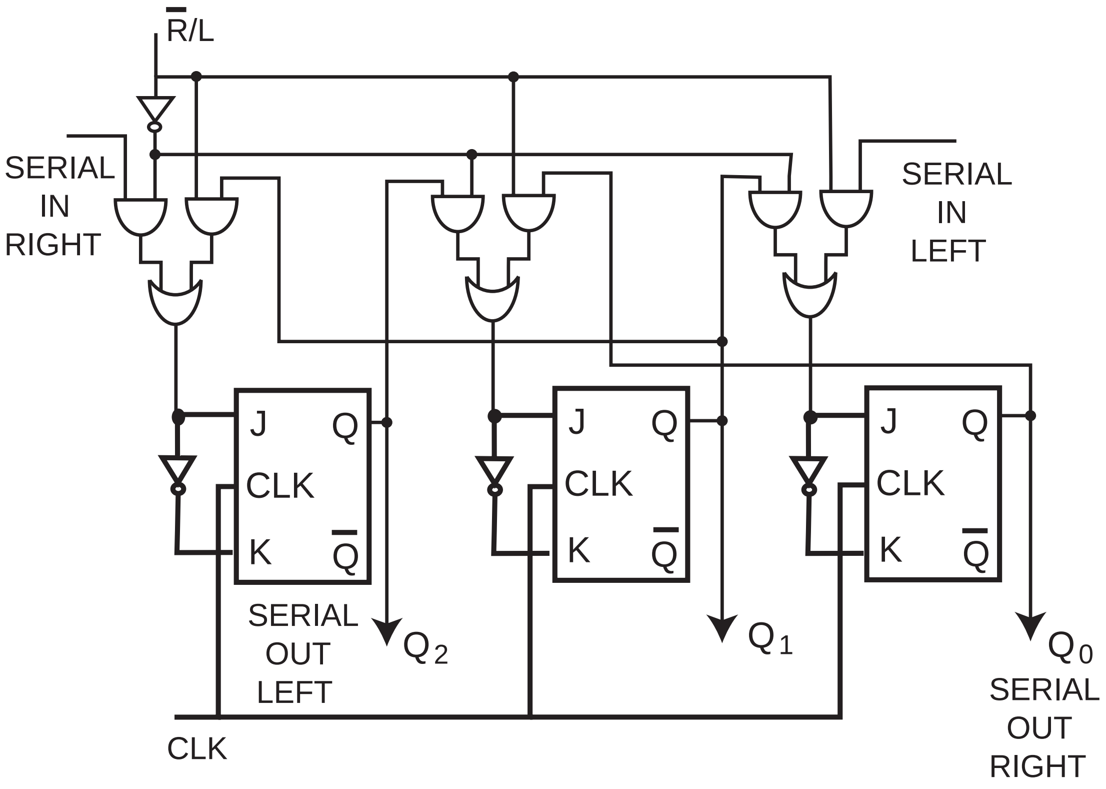
4. RIGHT-LEFT SHIFT REGISTER

SHIFT REGISTERS CAN ALSO BE USED TO TRANSFER DATA FROM RIGHT TO LEFT, **SFIFT LEFT**, BY CONNECTING THE **OUTPUT OF A FLIP-FLOP** BACK TO THE **INPUT OF THE FLIP-FLOP** ON ITS **LEFT**

A SHIFT FROM LEFT TO RIGHT, **SHIFT RIGHT**, CAN BE CARRIED OUT DURING **NORMAL OPERATION** OF THE SHIFT REGISTER.

A **SHIFT RIGHT AND SHIFT LEFT** REGISTER CAN BE COMBINED BY SUITABLE GATING AND CONTROL SIGNALS.

NOTE $\overline{R}/L$ >>> **$\overline{R}$/L = 0** >>> SHIFT RIGHT
$\overline{R}$/L = 1 >>> SHIFT LEFT


R̅/L
SERIAL IN RIGHT
SERIAL IN LEFT
J
Q
CLK
K
Q̅
SERIAL OUT LEFT
Q2
Q1
Q0
SERIAL OUT RIGHT
CLK