INTERFACING

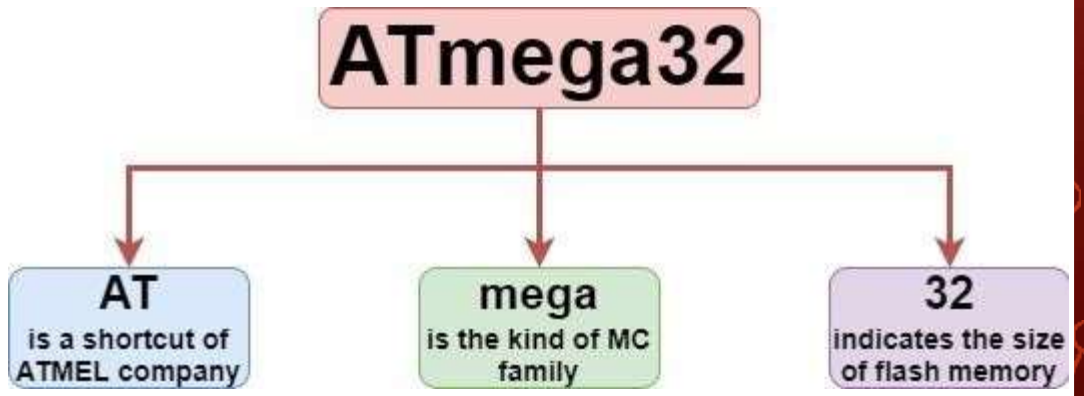
DIO MODULE & INTERFACES

AMIT

Digital Input Output:

Specs of Our Microcontroller:

- As we mentioned before, any computing system must have three main components: Processor, Memory and Input/Output Peripherals.
  We declared Processor and Memory in the last Session, so we will explain the first peripheral of our Input/Output Peripherals "**DIO**".
- Before in deep in DIO, we will recognize on our microcontroller on which we will work, it is an AVR microcontroller "ATmega32".
- The name of microcontroller consists of three syllables:
  - AT: indicates to the producer, ATMEL company.
  -mega: indicate to the type of family, whether there are three families "**tiny, mega, Xmega**".
  -32: indicates how many k bytes are the size of flash?
- Let's speak about the Specs of ATmega32.



AMIT

Digital Input Output:

Specs of Our Microcontroller:

- High-performance, Low-power AVR® 8-bit Microcontroller:
  it means that its **data bus** size is eight bits.
- Advanced RISC Architecture, so it means that:
  – 131 Powerful Instructions – Most Single-clock Cycle Execution.
  – 32 x 8 General Purpose Working Registers.
  – Fully Static Operation.
  – Up to 16 MIPS Throughput at 16 MHz.
  – On-chip 2-cycle Multiplier.
- Nonvolatile Program and Data Memories:
  – 32K Bytes of In-System Self-Programmable Flash: Endurance: 10,000 Write/Erase Cycles.
  – Optional Boot Code Section with Independent Lock Bits:
  In-System Programming by On-chip Boot Program and True Read-While-Write Operation.
  – 1024 Bytes EEPROM: Endurance: 100,000 Write/Erase Cycles.
  – 2K Byte Internal SRAM – Programming Lock for Software Security.

AMIT

Digital Input Output:

DIO Peripheral:

- A Digital Input Output is a peripheral that only deals with digital signals, it can generate output digital signal "**OUTPUT Mode**", or it can also read or detect input digital signals by using "**INPUT Mode**".

- Any peripheral consists of Hardware Circuit, but unfortunately the processor deals with memory only, so any peripheral must have a piece of memory to act as an intermediary between the processor and the hardware circuit.

- According to the last point, there is a new type of memory will be appeared, this is "Input/Output Memory".

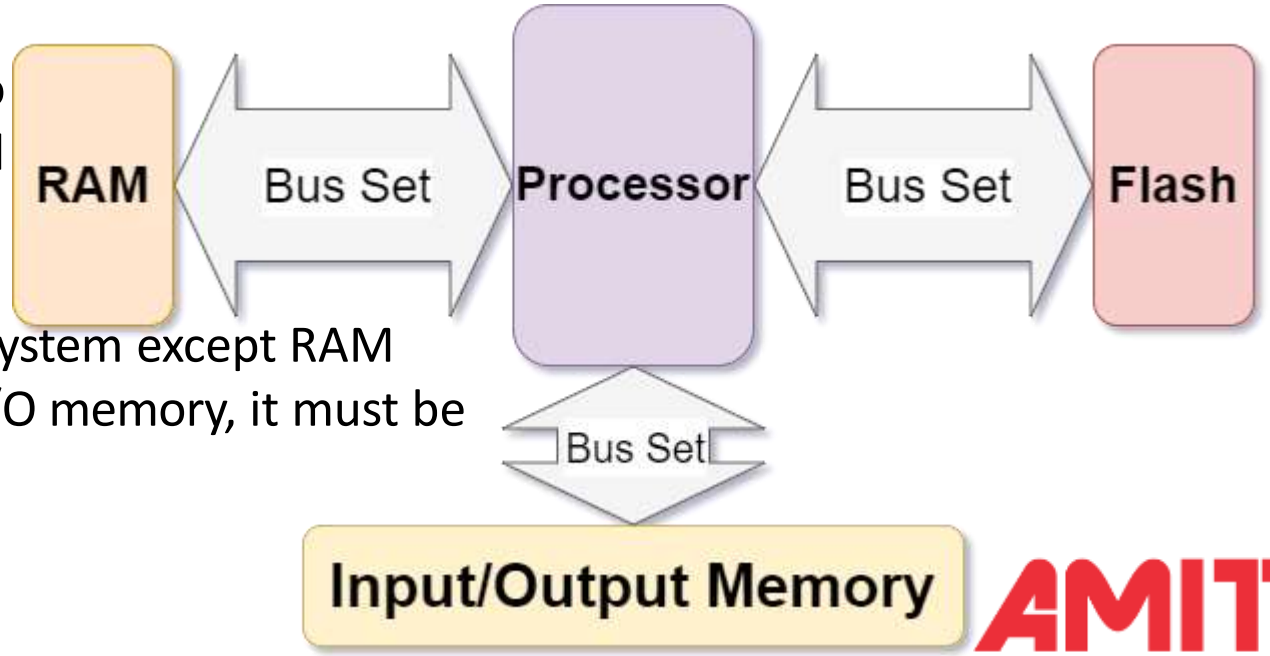
- The type of architecture of our microcontroller is "Harvard Architecture".

AMIT

Digital Input Output:

Accessing to I/O memory:

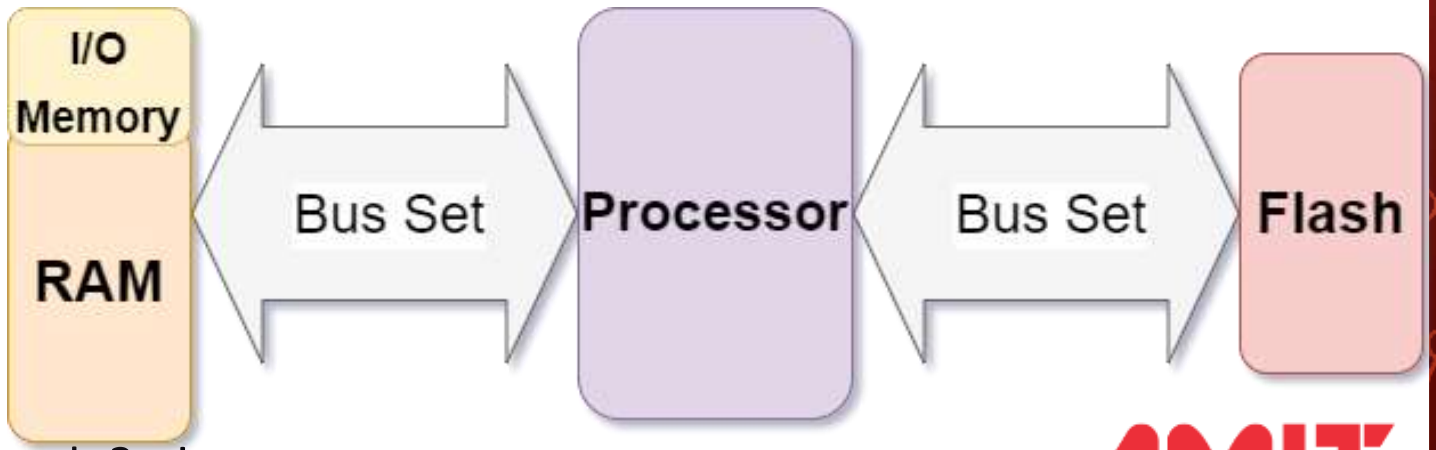
- As we mentioned before that ATmega32 architecture is a Harvard Architecture, so every memory system will be connected to the processor by a separated “Bus Set” so if the I/O memory will be connected by two ways:
  - Port Mapping:
    As the following figure, every memory system is connected to the processor using a separated Bus Set, so according to Harvard architecture, there is no way to access any memory system except RAM using the C level, so to access I/O memory, it must be accessed using Assembly level.



Digital Input Output:

Accessing to I/O memory:

- As we mentioned before that ATmega32 architecture is a Harvard Architecture, so every memory system will be connected to the processor by a separated “Bus Set” so if the I/O memory will be connected by two ways:
  - Memory Mapping:
    As the following figure, I/O memory is connected on the same Bus Set of RAM, so I/O and RAM will share the same bus as a Von-Neumann Architecture, so the I/O now can be accessed using C level, but unfortunately there is no way to pipeline between I/O and RAM memory, So if the user needs the fast access not the easy way access, what should exactly the user do?...!



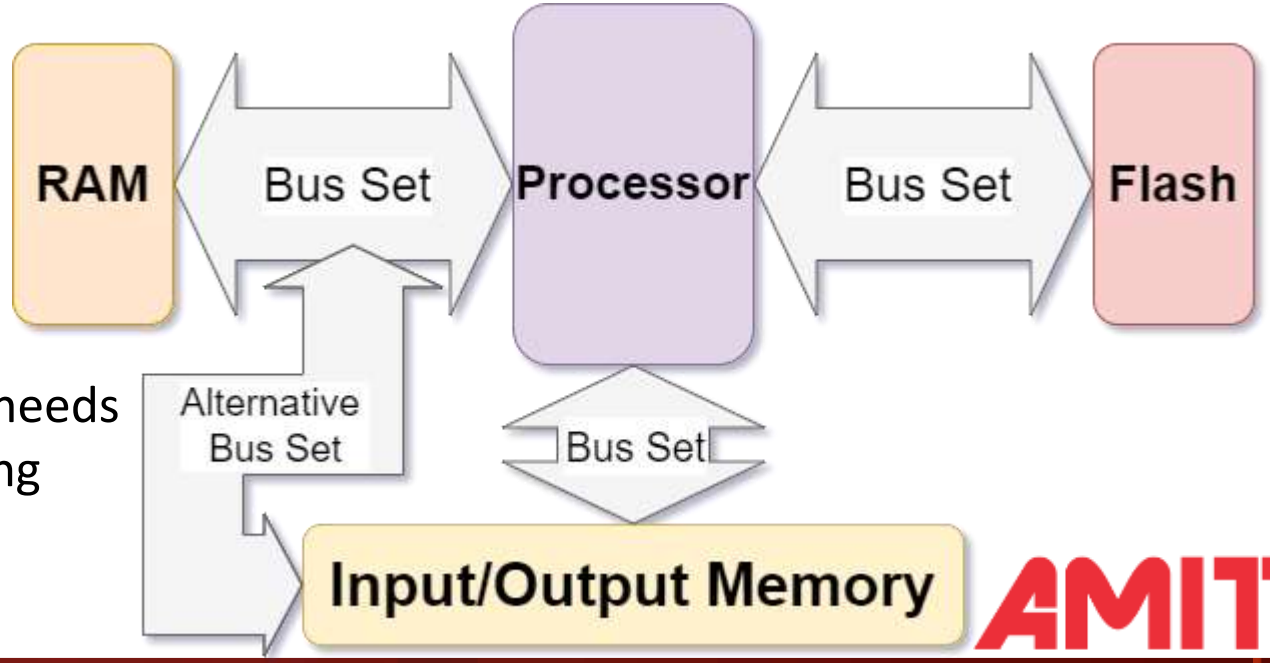
AMIT

Digital Input Output:

Accessing to I/O memory:

- Our Microcontroller will merge the two techniques in its Architecture, so the I/O memory will be connected by a separated Bus Set “so its addresses will be started with $00 address, but these addresses are only accessed by Assembly Level”, also it is connected to with the Bus Set of RAM memory “So the addresses of I/O registers will be started from $20, because the previous 32 locations are the remapping addresses of GPRs, and it can be accesses by C Level”.

- So, if the user needs easy access, he can use the Memory Mapping addresses into C Level, and if the user needs fast access, he can use the Port Mapping addresses.



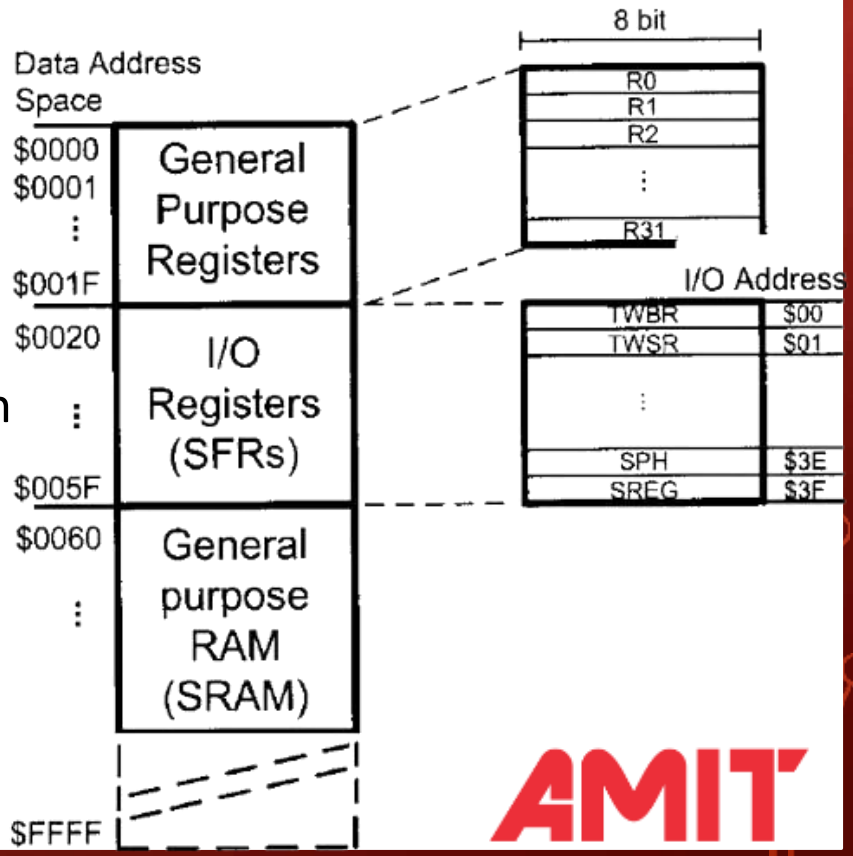
AMIT

Digital Input Output:

Accessing to I/O memory:

- To Access I/O by using Port Mapping addresses, we must use Assembly Level by using for example: “IN & OUT” instruction.

- The IN instruction tells the CPU to load one byte from an I/O register to the GPR. After this instruction is executed, the GPR will have the same value as the I/O register. For example, the “in 20, 0x16” instruction will copy the contents of location 16 (in hex) of the I/O memory into R20. Each location in I/O memory has two addresses: I/O address and data memory address. Each location in the data memory has a unique address called the data memory address.
  Each I/O register has a relative address in comparison to the beginning of the I/O memory; this address is called the I/O address.

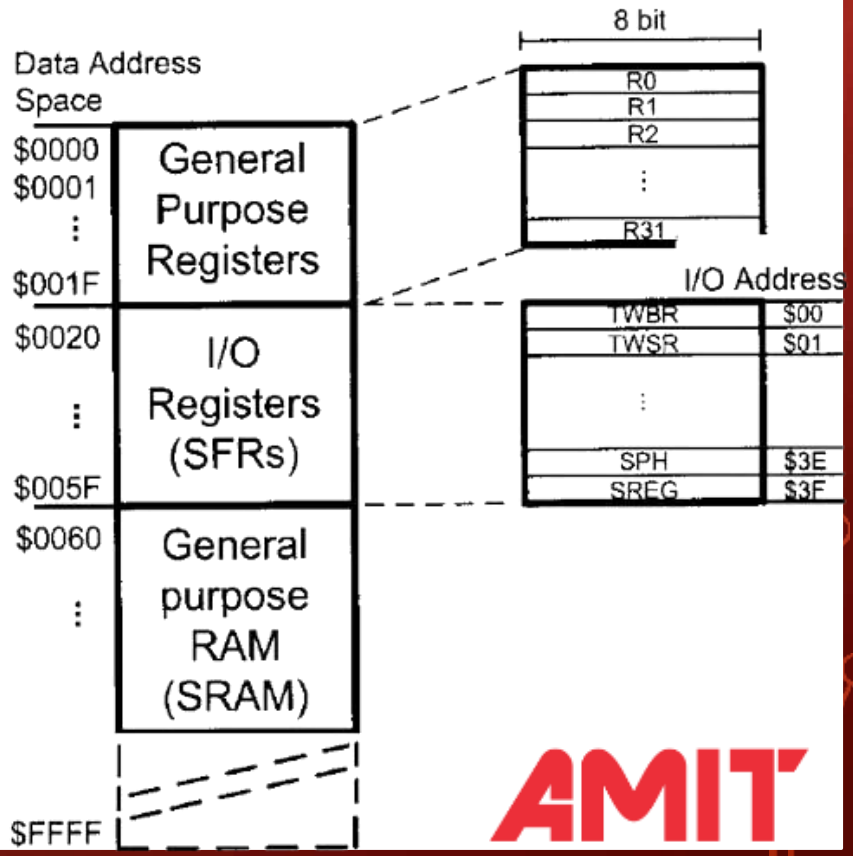


Digital Input Output:

Accessing to I/O memory:

- To Access I/O by using Port Mapping addresses, we must use Assembly Level by using for example: “IN & OUT” instruction.

- The OUT instruction tells the CPU to store the GPR to the I/O register. After the instruction is executed, the I/O register will have the same value as the GPR. For example, the “out PORTD, R10” instruction will copy the contents of R10 into PORTD (location 12 of the I/O memory).
  Notice that: in the OUT instruction, the I/O registers are referred to by their I/O addresses (like the IN instruction).



Digital Input Output:

Difference between LDS & IN:

- LDS instruction to copy the contents of a memory location to a GPR.
  This means that we can load an I/O register into a GPR, using the LDS instruction.
  So, what is the advantage of using the IN instruction for reading the contents of I/O registers over using the LDS instruction? The IN instruction has the following advantages:
  - The CPU executes the IN instruction faster than LDS, IN instruction lasts 1-machine cycle, where as LDS lasts 2-machine cycles.
  - The IN is a 2-byte instruction, whereas LDS is a 4-byte instruction.
    This means that the IN instruction occupies less code memory.
  - When we use the IN instruction, we can use the names of the I/O registers instead of their addresses.
  - The IN instruction is available in all of the AVRs, where as LDS is not implemented in some of the AVRs.
- Notice that in using the IN instruction we can access only the standard I/O memory, while we can access all parts of the data memory using the LDS instruction,

AMIT

Digital Input Output:

Difference between LDS & IN:

- LDS instruction to copy the contents of a memory location to a GPR.
  This means that we can load an I/O register into a GPR, using the LDS instruction.
  So, what is the advantage of using the IN instruction for reading the contents of I/O registers over using the LDS instruction? The IN instruction has the following advantages:
  - The CPU executes the IN instruction faster than LDS, IN instruction lasts 1-machine cycle, where as LDS lasts 2-machine cycles.
  - The IN is a 2-byte instruction, whereas LDS is a 4-byte instruction.
    This means that the IN instruction occupies less code memory.
  - When we use the IN instruction, we can use the names of the I/O registers instead of their addresses.
  - The IN instruction is available in all of the AVRs, where as LDS is not implemented in some of the AVRs.
- Notice that in using the IN instruction we can access only the standard I/O memory, while we can access all parts of the data memory using the LDS instruction,

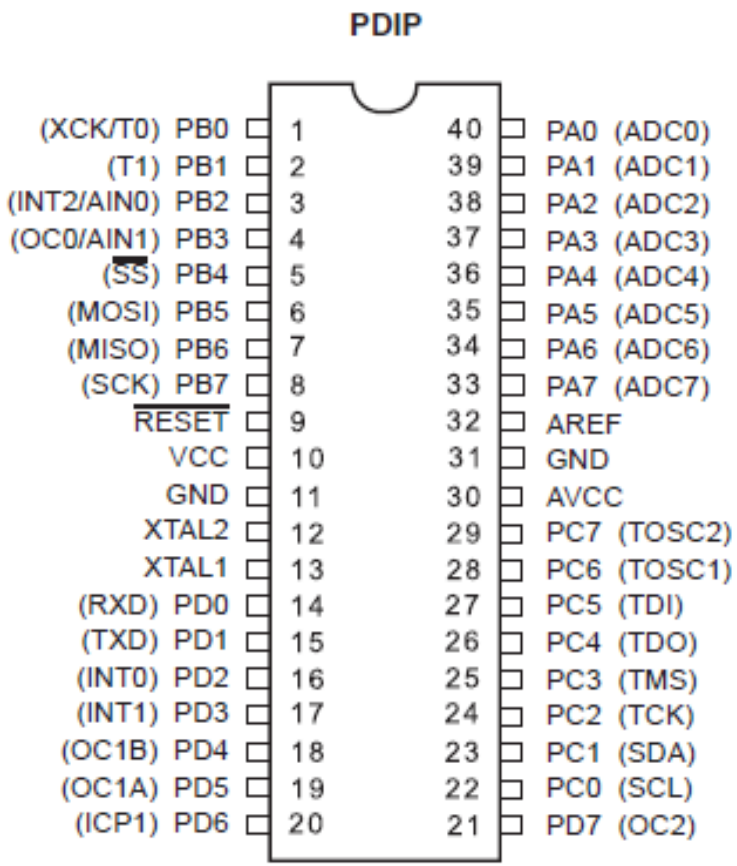
AMIT

Digital Input Output:

Pin Configuration of ATmega32:

- ATmega32 has 40 pins into DIP manufacturing, 44 pins into TQFP manufacturing, there is no difference between of them in the features, but the extra 4 pins only extra pins for ground and Vcc.

- There are 32 General Purpose Input/Output pins, and only 8 Specific Function pins, Lets start with specific function pins:
  - **Vcc**: This pin must be connected to “5V”.
  - **GND**: This pin must be connected to “0V”, there are two pins at each side for PCB usage.
  - **RESET**: This pin is Active Low pin, when it is connected to GND, the reset handler will be run, so the code will be started from the beginning.

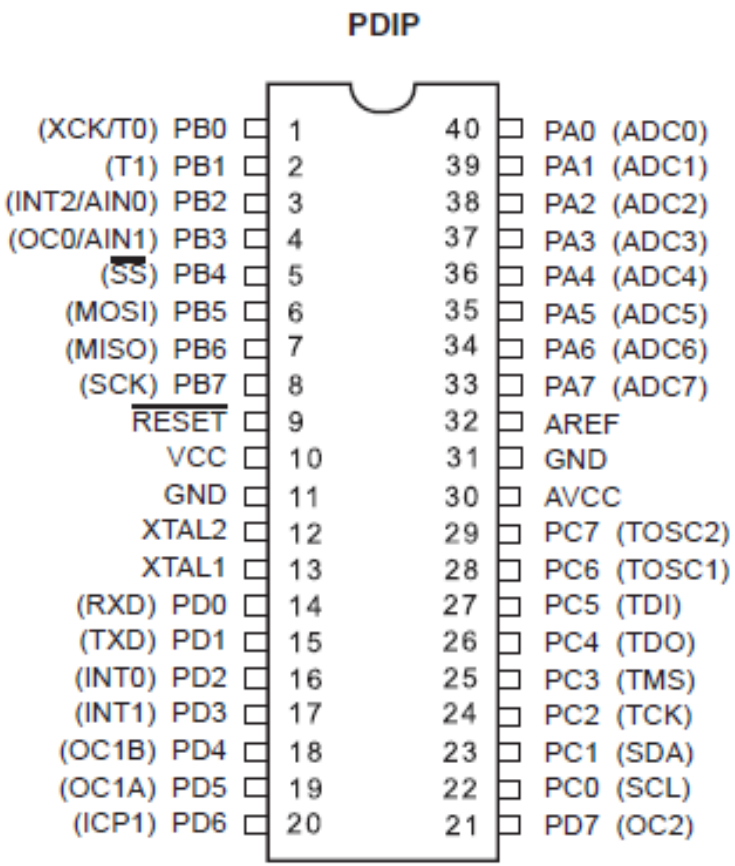


AMIT

Digital Input Output:

Pin Configuration of ATmega32:

- There are 32 General Purpose Input/Output pins, and only 8 Specific Function pins, Lets start with specific function pins:
  - **AREF**: This pin is connected to a volt “max is 5V”, which is used as reference voltage for ADC peripheral, it will be declared later in ADC session.
  - **AVCC**: This pin is connected to a volt “5V”, which is used as the power of ADC peripheral, because ADC is only peripheral that takes its power from external pin, not fro m the power of microcontroller like the remaining peripherals.

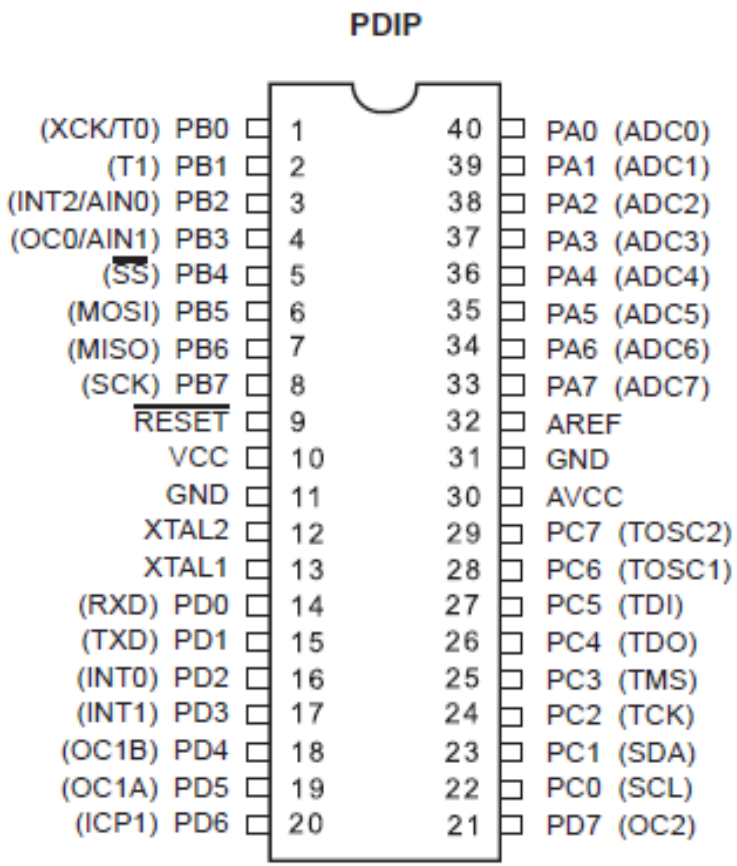


AMIT

Digital Input Output:

Pin Configuration of ATmega32:

- There are 32 General Purpose Input/Output pins, and only 8 Specific Function pins, Lets start with specific function pins:
  - **XTAL1 & XTAL2**:
    These pin are connected to the external crystal clock which XTAL1 pin is an output from microcontroller to drive the crystal clock “Mechanical Clock” and XTAL2 is an input pin to get through the output clock from the crystal clock to microcontroller.
    If the type of clock is electrical, XTAL2 pin is only pin will be connected to the output of Clock circuit to this pin because the circuit takes its power from an external power source, so this connection is called as “By Pass Connection”.

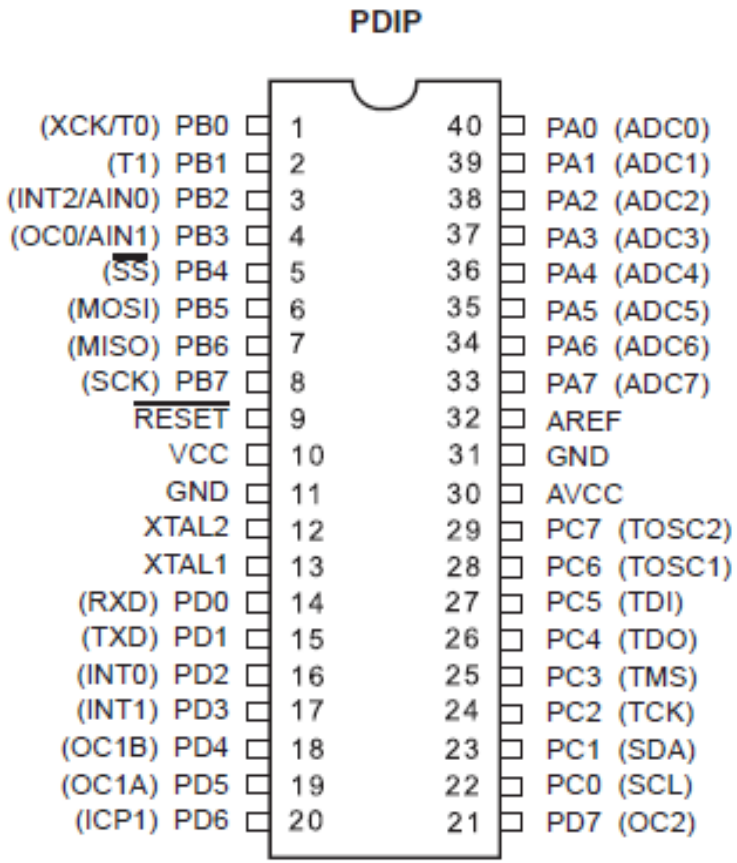


AMIT

Digital Input Output:

Pin Configuration of ATmega32:

- There are 32 General Purpose Input/Output pins, now we will discuss the General Purpose Input Output Pins.
- As our microcontroller is an 8-bit Data bus, also every pin can be controlled by a single bit, the pins will be divided into Four Groups “A, B, C and D”.
- Every Group will contain eight pins, so every group has a register at least to control the pins of group.
- In fact, every group has three registers to control its pins:
  - Data Direction Register “DDRx”:
    it controls the direction of pins “OUTPUT or INPUT”.
  - Output Register “PORTx”:
    it controls the level of pins “HIGH or LOW”.
  - Input Register “PINx”:
    it stores the level that is connected to pins “HIGH or LOW”.



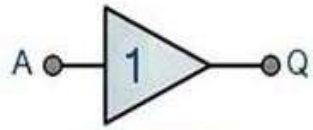
AMIT

Digital Input Output:

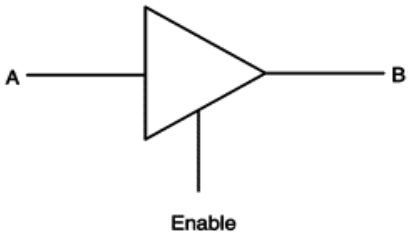
Hardware Circuit of DIO:

- As we mentioned before, any peripheral must have main two components: Hardware Circuit and memory to manage the processor to order or request a command from the hardware circuit, and the main hardware component of our DIO is Tri-State Buffer.

- Normal Buffer Component as the following figure works as high impedance buffer that can allow the voltage which is applied on “A” gets through to “Q” after reducing voltage power, it only reduces the current.

Symbol	Truth Table	
 A Digital Buffer	A	Q
	0	0
	1	1
Boolean Expression **Q = A**	Read as: **A** gives **Q**	

- Tri-State Buffer looks like the normal buffer, but it has an extra pin, this pin controls the voltage flow between “A & B”, so if enable pin is powered to “0”, there is no voltage will get through, if it is ? powered to “1”, the voltage will get through from “A → B”.



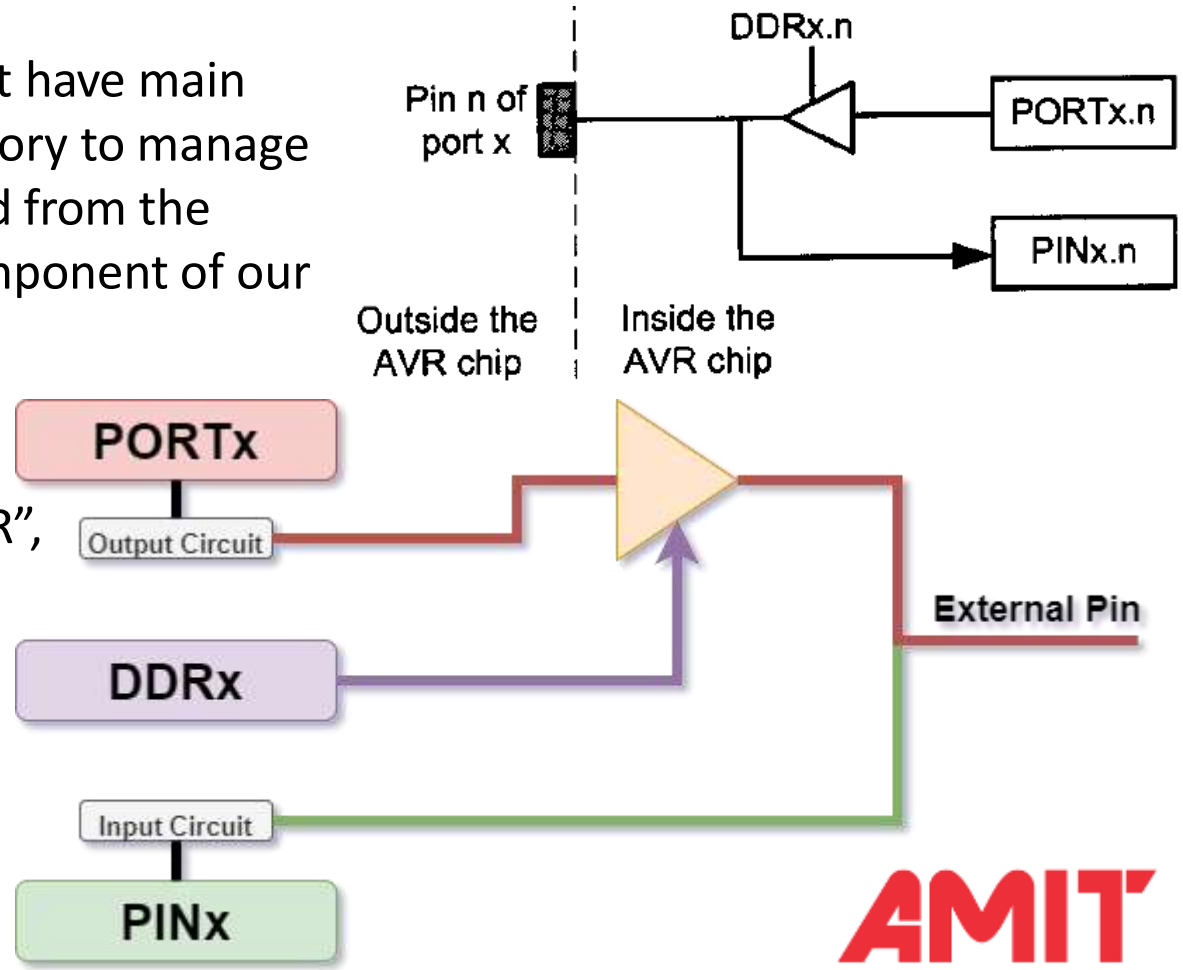
Enable	A	B
0	0	Z
0	1	Z
1	0	0
1	1	1

Digital Input Output:

Hardware Circuit of DIO:

- As we mentioned before, any peripheral must have main two components: Hardware Circuit and memory to manage the processor to order or request a command from the hardware circuit, and the main hardware component of our DIO is Tri-State Buffer.

- There are main three registers into our DIO, first is the Direction Data Register called “DDR”, which controls the direction of pin “input =0, output=1” because it is triggered the Tri-State Buffer to enable “1” or Disabled “0” the output circuit.



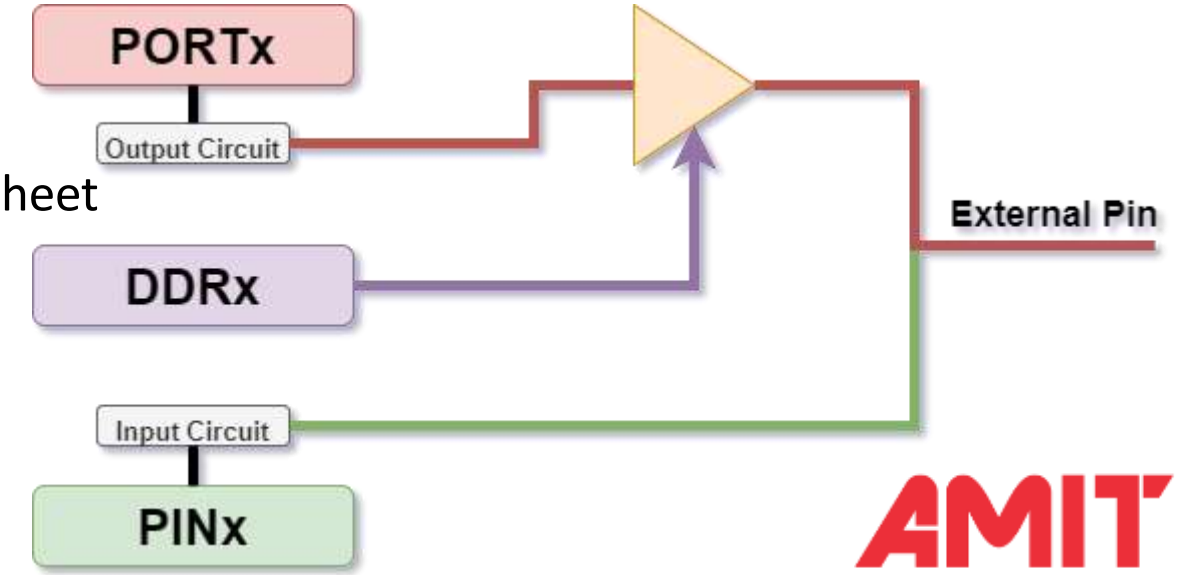
Digital Input Output:

Hardware Circuit of DIO:

- If the Direction of pin is configured as an output, the DIO can gets through High or Low level, by configuring the output register called “PORT” by “High=1 , Low=0”.

- If we need to turn on a LED, we have two ways to access the DIO register “actually, we have two addresses to access any I/O register: memory and port mapping, Lets start with memory mapping addresses.

- Assume that the LED is connected to Pin 3, Group B, now we must open the datasheet to get the addresses of registers we need.



AMIT

Digital Input Output:

Register Summery of I/O registers:

- We will find that every I/O register has two addresses, the memory mapping exists into practices (), Port mapping will be outside, because it starts from $00 location.

Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C	8
$3E ($5E)	SPH	–	–	–	–	SP11	SP10	SP9	SP8	10
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	10
$3C ($5C)	OCR0	Timer/Counter0 Output Compare Register								80
$3B ($5B)	GICR	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	45, 65
$3A ($5A)	GIFR	INTF1	INTF0	INTF2	–	–	–	–	–	66
$39 ($59)	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	80, 110, 128
$38 ($58)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	81, 111, 128
$37 ($57)	SPMCR	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	246
$36 ($56)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	175
$35 ($55)	MCUCR	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	30, 64
$34 ($54)	MCUCSR	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	38, 65, 226
$33 ($53)	TCCR0	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	78
$32 ($52)	TCNT0	Timer/Counter0 (8 Bits)								80
...	...	...	...	...	...	...	...	...	...	...
$01 ($21)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	176
$00 ($20)	TWBR	Two-wire Serial Interface Bit Rate Register								175

Digital Input Output:

Turning on a LED:

- After checking the register summery, the addresses of DIO registers start from "0x30 → 0x3B" as memory mapping addresses, and also they start from "0x10 → 0x1B" as port mapping addresses.

- As we mentioned before, Port Mapping addresses are used only in the assembly level.

- All IO registers can be accessed by "**IN** / **OUT**" instructions, but there are Two instructions that can access IO registers from "0x00 → 0x1F", which are "**SBI**: Set Bit Instruction, **CBI**: Clear Bit Instruction" and they can Set/Clear a specific bit into an IO register from "0x00 → 0x1F" as one instruction that takes two clock cycles, and it can not be interrupted because it is a single instruction not like Set/Clear in C level, because it takes minimum there instructions "Read – Modify – Write".

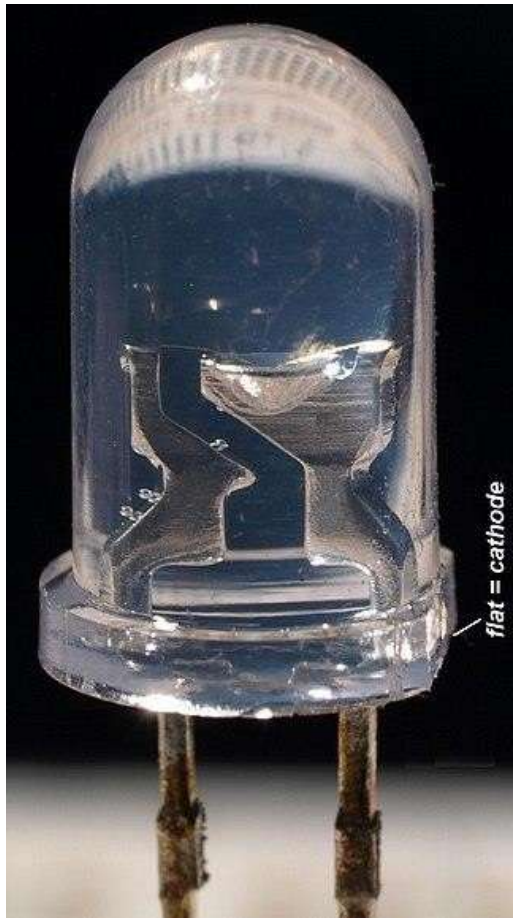
$1B ($3B)	PORTA
$1A ($3A)	DDRA
$19 ($39)	PINA
$18 ($38)	PORTB
$17 ($37)	DDRB
$16 ($36)	PINB
$15 ($35)	PORTC
$14 ($34)	DDRC
$13 ($33)	PINC
$12 ($32)	PORTD
$11 ($31)	DDRD
$10 ($30)	PIND

AMIT

Digital Input Output:

Turning on a LED:

- Light Emitting Diode:
  A light-emitting diode (LED) is a semiconductor light source that emits light when current flows through it. Electrons in the semiconductor recombine with electron holes, releasing energy in the form of photons.
  The color of the light (corresponding to the energy of the photons) is determined by the energy required for electrons to cross the band gap of the semiconductor.
  White light is obtained by using multiple semiconductors or a layer of light-emitting phosphor on the semiconductor device.
- Most of LEDs run at “3 volts” maximum, and the maximum current that it consumes is about “20 mA” maximum, so if it is connected to “5 volts”, it must be connected in series with a resistor that its value can be calculated by: $\mathbf{Resistor\ value} = \frac{5-3}{20 \times 10^{-3}} = 100\ \Omega$, but most common LEDs can be connected to $330\ \Omega$.

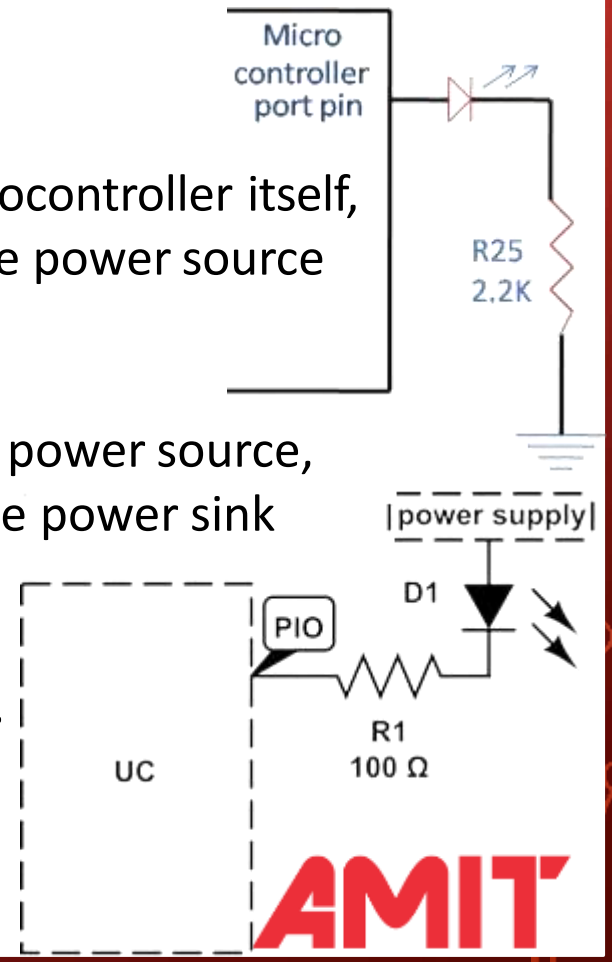


AMIT

Digital Input Output:

Turning on a LED:

- Types of Connection:
  - Source Connection:
    It means that the LED will obtain the power source from the microcontroller itself, like the next figure, which the microcontroller is considered as the power source of LED, so its activation state is to power the Pin to High.
  - Source Connection:
    It means that the LED will obtain the power source from external power source, like the next figure, which the microcontroller is considered as the power sink of LED, so its activation state is to power the Pin to Low.
  - Current sinking and sourcing:
    Each I/O pin can sink or source a recommended current of 20mA.
    Each I/O pin can sink or source a maximum current of 40mA.
    It must be ensured that the current sourced or sunk from all the ports combined, should not exceed 200mA.



Digital Input Output:

Turning on a LED:

- To control a LED, first we must set the pin that will be connected to it as an output, then the output register must be set/cleared according to the type of connection, so we have two methods to control the LED by accessing “memory or port mapping addresses”.

- Assume that the LED will be connected to “pin 3” in group “ C ”, so the accessing method will be memory mapping “**DDRC**: 0x34 , **PORTC**: 0x35”.
  We have the address of register, but it must be assigned into pointer or casted to be accessible, so casting will be chosen to reduce memory consumption.

- For example, “**0x34**” will be casted to pointer to the register size itself “1 Byte”, so it will be casted to pointer to **u8** “**unsigned char**”, as the following: (**u8** ***)0x34, now it becomes a pointer, but we need to dereference this pointer, so the final result will be:
  ***((u8 *)**0x34 **) =** any_value_you_need;

AMIT

Digital Input Output:

Turning on a LED:

- **Example Code to turn on the led "using Memory Mapping":**
  This code will turn on the LED that will be connected to pin 3, in group C, now we use the <util/delay.h> library to use its APIs:
  "_delay_ms(time_in_milli_second); "
  "_delay_us(time_in_micro_second);"
  to make an application that blinks this LED with one second interval.

```
typedef unsigned char u8;

int
main(void)
{
    //set pin 3 in group C as output
    *((u8*)0x34) |= (1<<3);
    //set pin 3 in group C as high
    *((u8*)0x35) |= (1<<3);
}
```

AMIT

Digital Input Output:

Turning on a LED:

- **Example Code to Blink the led "using Memory Mapping":**
  The code besides is written to blink the LED, but in fact it will be turned on once, then turned off once, because there now thing that orders the processor to repeat the code, normally the execution of main function will end and never to enter again, so, we need to use a keyword in C that can order the processor to run this code forever.
  Guess, what should we use?....!!

```
int
main(void)
{
    //set pin 3 in group C as output
    *((u8*)0x34) |= (1<<3);
    //set pin 3 in group C as high
    *((u8*)0x35) |= (1<<3);
    _delay_ms(1000);
    //set pin 3 in group C as low
    *((u8*)0x35) &=~(1<<3);
    _delay_ms(1000);
}
```

AMIT

Digital Input Output:

Turning on a LED:

- **Super Loop:**

  The super loop is used for two reasons:
  - For running the program for ever, and making the main function never exists.
  -To protect the program memory, because if there is no super loop, PC register will be increased forever, until it is overflowed, so the Program Counter will start from zero, and the processor will access all flash sections that are not accessed in normal cases, but if you try to do it, you will not find any result that because the startup code protect it be finalize function that only has a super loop, and it is only executed if the main function is an exited function.

```
int
main(void)
{
    //set pin 3 in group C as output
    *((u8*)0x34) |= (1<<3);
    //set pin 3 in group C as high
    *((u8*)0x35) |= (1<<3);
    _delay_ms(1000);
    //set pin 3 in group C as low
    *((u8*)0x35) &=~(1<<3);
    _delay_ms(1000);
}
```

AMIT

Digital Input Output:

Turning on a LED:

- **Super Loop**:
  This is our code after using super loop system, so, the LED will be blinked forever.
- Now, we want to access the pins using "Port Mapping" addresses, so we must use assembly level "**IN/OUT** 0x00 – 0x3F" or "**SBI/CBI** 0x00 – 0x1F" to access, but if we need to access it within a C code, is it available?.....!!
- If an assembly instruction is written with a C code, compiler will generate an error because the assembly instructions are not defined in C syntax.
- So, if an assembly instructions should be written within a C code, we must use inline assembly.

```
int
main(void)
{
    //set pin 3 in group C as output
    *((u8*)0x34) |= (1<<3);
    while (1)
    {
        //set pin 3 in group C as High
        *((u8*)0x35) |= (1<<3);
        _delay_ms(1000);
        //set pin 3 in group C as Low
        *((u8*)0x35) &=~(1<<3);
        _delay_ms(1000);
    }
}
```

AMIT

Digital Input Output:

Turning on a LED:

- **Inline Assembly:**
  To use this feature, we will use the internal function "**asm**", so we can write the assembly instruction can be written as a string, and be passed as an argument for "**asm**".

- So if pin 3 in Group C will be configured as an output pin using "**SBI/CBI**" instructions, it will be written '**SBI 0x14, 3**', so
  it will be:

  __asm__ ("SBI 0x14 , 3");
  // set bit 3 in DDRC register by 1

```
int
main(void)
{
    //set pin 3 in group C as output
    asm ("SBI  0x14,  3");
    while (1)
    {
        //set pin 3 in group C as High
        asm ("SBI  0x15,  3");
        _delay_ms(1000);
        //set pin 3 in group C as Low
        asm ("CBI  0x15,  3");
        _delay_ms(1000);
    }
}
```

AMIT

Digital Input Output:

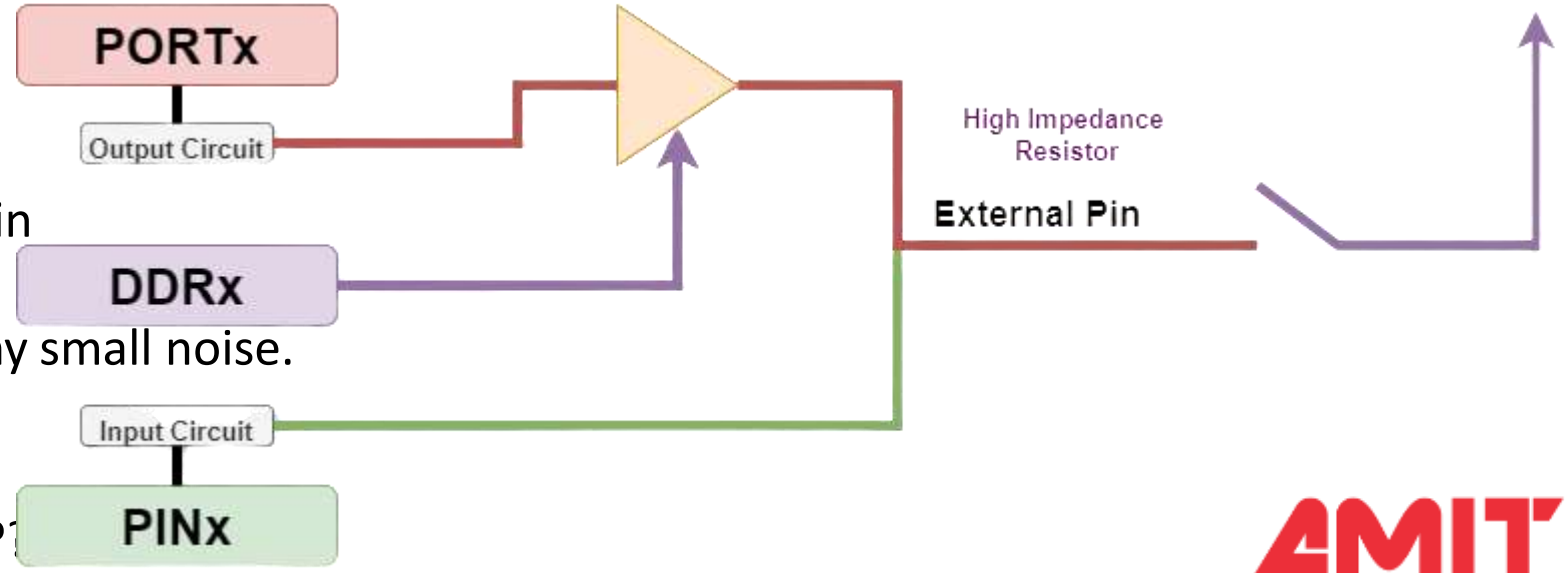
Input Mode:

- **Connection of Switch:**

  - Connect Switch to Power Source:

  In this case, the pin should be configured as an Input, so the Tri-State Buffer will be disabled, Now if the switch is closed, PINx register will be triggered and “Logic ONE” will be written.

  If the switch is opened, the PINx register will be connected to open circuit, so the bit related to this pin will be floated, so it can be changed by any small noise. So, the pin must read “Logic ZERO” when the switch is open , But HOW?!



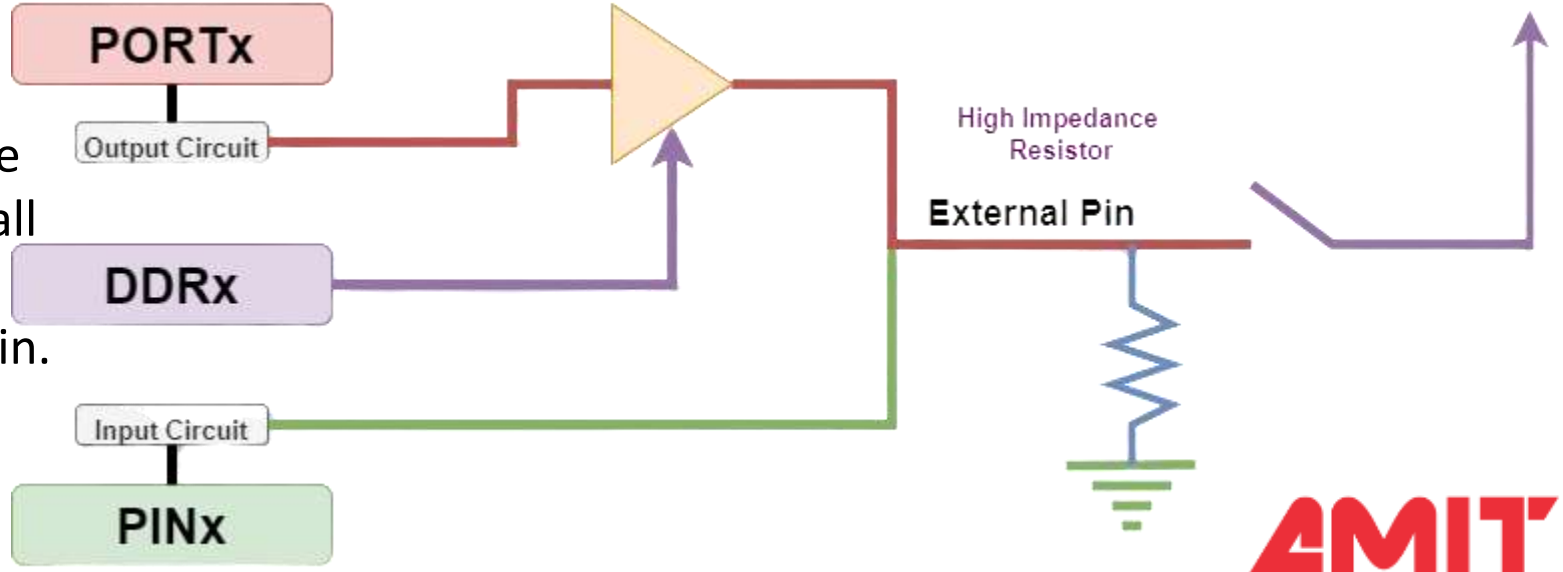
Digital Input Output:

Input Mode:

- **Connection of Switch:**

  - PULL-DOWN Resistor:

  In this case, the pull-down resistor is a mandatory to avoid the Open Circuit or Floating signal, and as we mentioned before that we need to trigger the PINx register by ZERO if it is opened, but a high impedance resistor must be connected to avoid the short circuit between power source and ground, also to make the power source avoid the high impedance line, and all power will be sink into the bit in PINx related to pin.



- **CLOSE state → 1**
  OPEN state → 0

AMIT

Digital Input Output:

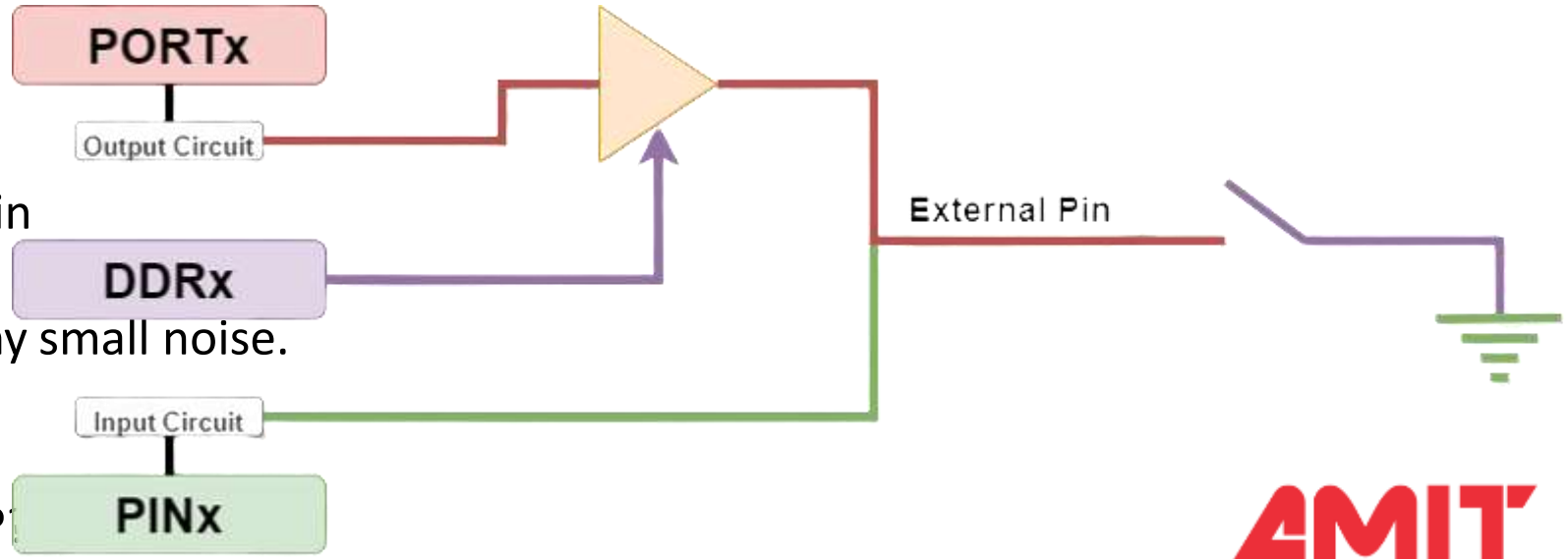
Input Mode:

- **Connection of Switch:**

  - Connect Switch to Ground:

  In this case, the pin should be configured as an Input, so the Tri-State Buffer will be disabled, Now if the switch is closed, PINx register will be triggered and “Logic ZERO” will be written.

  If the switch is opened, the PINx register will be connected to open circuit, so the bit related to this pin will be floated, so it can be changed by any small noise. So, the pin must read “Logic ONE” when the switch is open , But HOW?



AMIT

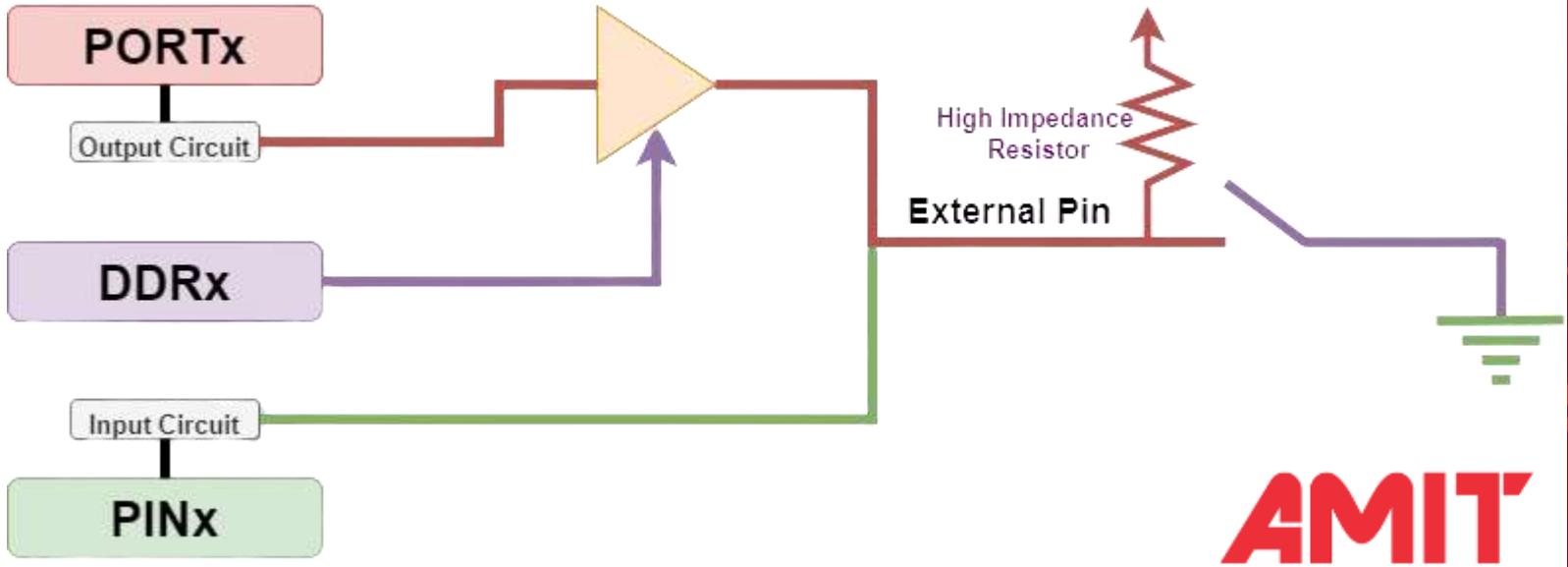
Digital Input Output:

Input Mode:

- **Connection of Switch:**

  - PULL-UP Resistor:

  In this case, the pull-up resistor is a mandatory to avoid the Open Circuit or Floating signal, and as we mentioned before that we need to trigger the PINx register by ONE if it is opened, but a high impedance resistor must be connected to avoid the short circuit between power source and ground, also for power exhausting in the resistor when it is closed, and the ground will be the dominant signal that will affect on PINx register.



- **CLOSE state → 0**
  OPEN state → 1

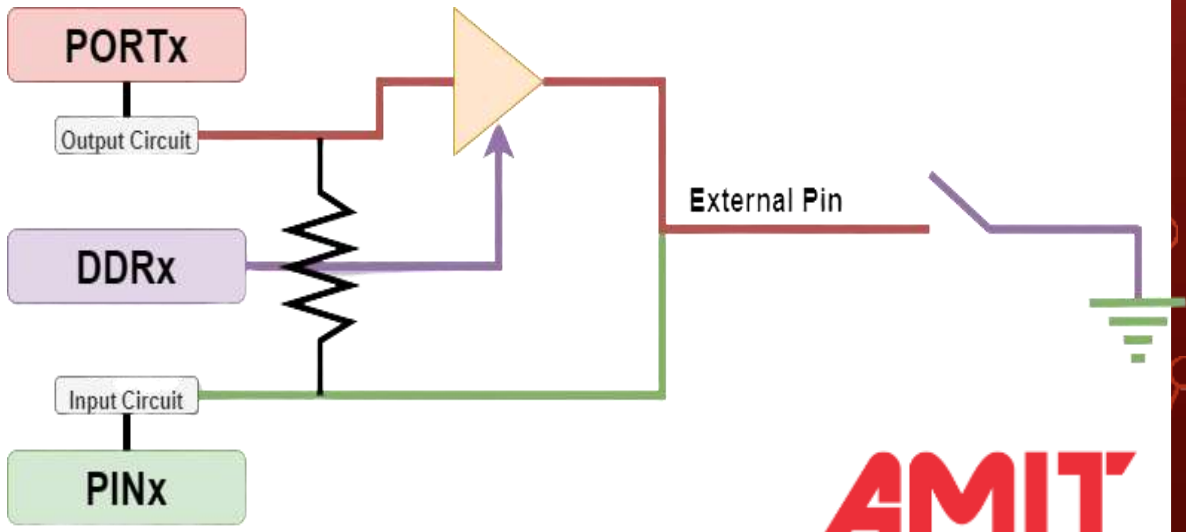
Digital Input Output:

Internal Pull-up Resistor:

- **Connection of Switch:**

  - PULL-UP Resistor:

  In this case, the pull-up resistor exists internally into our microcontroller, this resistor has two poles, when the external pull up resistor is connected to pin, one of their poles is connected to pin and the other is connected to power source, so when it is connected internally, the pole which is connected to pin is still connected to pin, but the second must be connected to Vcc, so the register that can generate 5v is PORTx, so to enable the internal pull up resistor of an input pin, PORTx must also be set in the same order of pin.

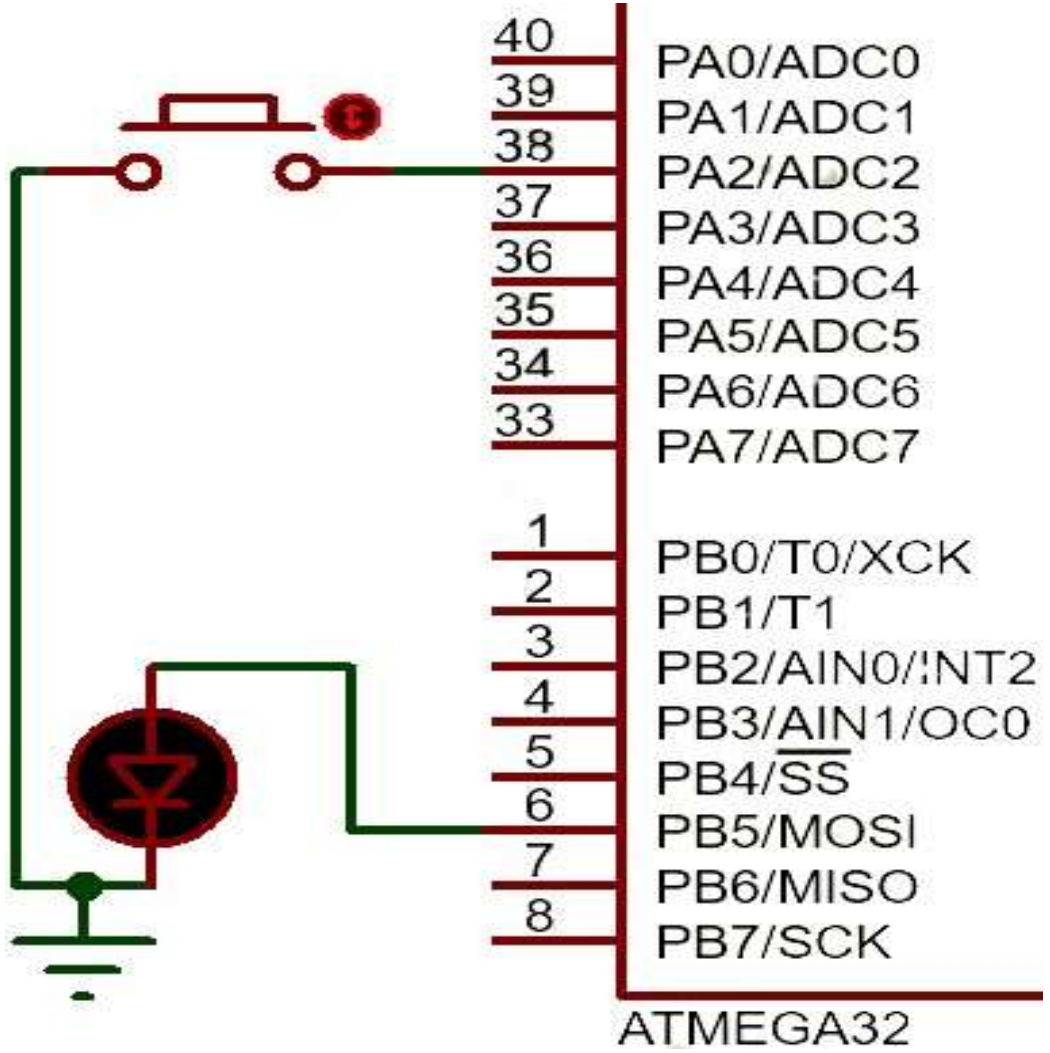


AMIT

Digital Input Output:

Example on Input Signals:

- **Toggle the LED, if the switch is pressed:**
  Switch is connected to PA2,
  LED is connected to PB5:
  - First, set the pin connected as output.
  -Then, set the pin connected an pulled up input.
  -After that, check if the bit related to the pin connected to switch if it has ZERO "because the close state in this type of connection is ZERO, and ONE if it is opened", the LED will be toggled, if not, there is nothing will be happened.



AMIT

Digital Input Output:

Example on Input Signals:

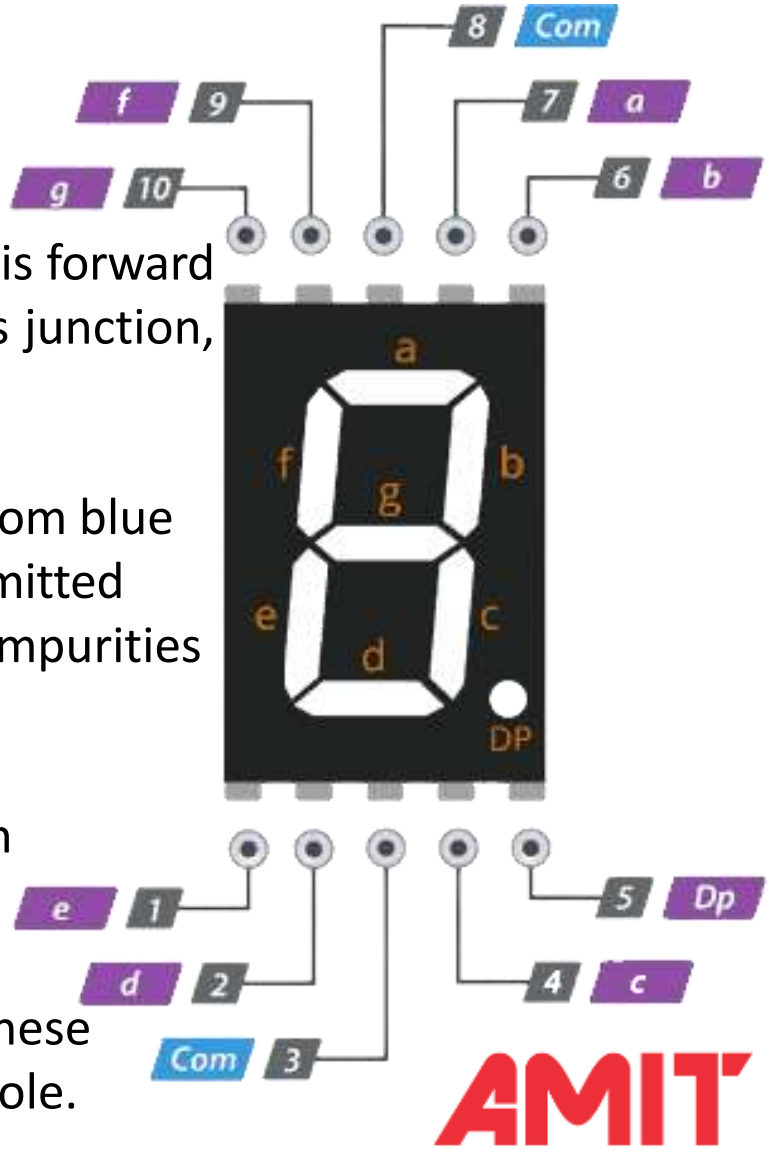
- **Toggle the LED, if the switch is pressed**:
  Switch is connected to PA2,
  LED is connected to PB5:
  - First, set the pin connected as output.
  -Then, set the pin connected an pulled up input.
  -After that, check if the bit related to the pin connected to switch if it has ZERO "because the close state in this type of connection is ZERO, and ONE if it is opened", the LED will be toggled, if not, there is nothing will be happened.

```
int
main(void)
{
    //set pin 5 in group B as output
    *((u8*)0x37) |= (1<<5);//DDRB
    //set pin 2 in group A as input pull up
    *((u8*)0x3A) &=~(1<<2);//DDRA
    *((u8*)0x3B) |= (1<<2);//PORTA
    while (1)
    {
        //Read bit 2 in PINA
        if ( (( *((u8*)0x39) >>2) &1) == 0 )
        {
            // Toggling LED
            *((u8*)0x38) ^= (1<<5);//PORTB
        }
    }
}
```

AMIT

Digital Input Output:

Seven Segment Display:



- The emission of these photons occurs when the diode junction is forward biased by an external voltage allowing current to flow across its junction, and in Electronics we call this process electroluminescence.

- The actual color of the visible light emitted by an LED, ranging from blue to red to orange, is decided by the spectral wavelength of the emitted light which itself is dependent upon the mixture of the various impurities added to the semiconductor materials used to produce it.

- The seven segment simply contains seven LEDs arranged to form numbers from 0 → F, it has 10 pins, from **a** to **g**, every pin is connected to every LED, and one is connected to the DOT LED, and the remaining two pins are connected to common pole of these LEDs whether they are connected to ANODE pole or CATHODE pole.

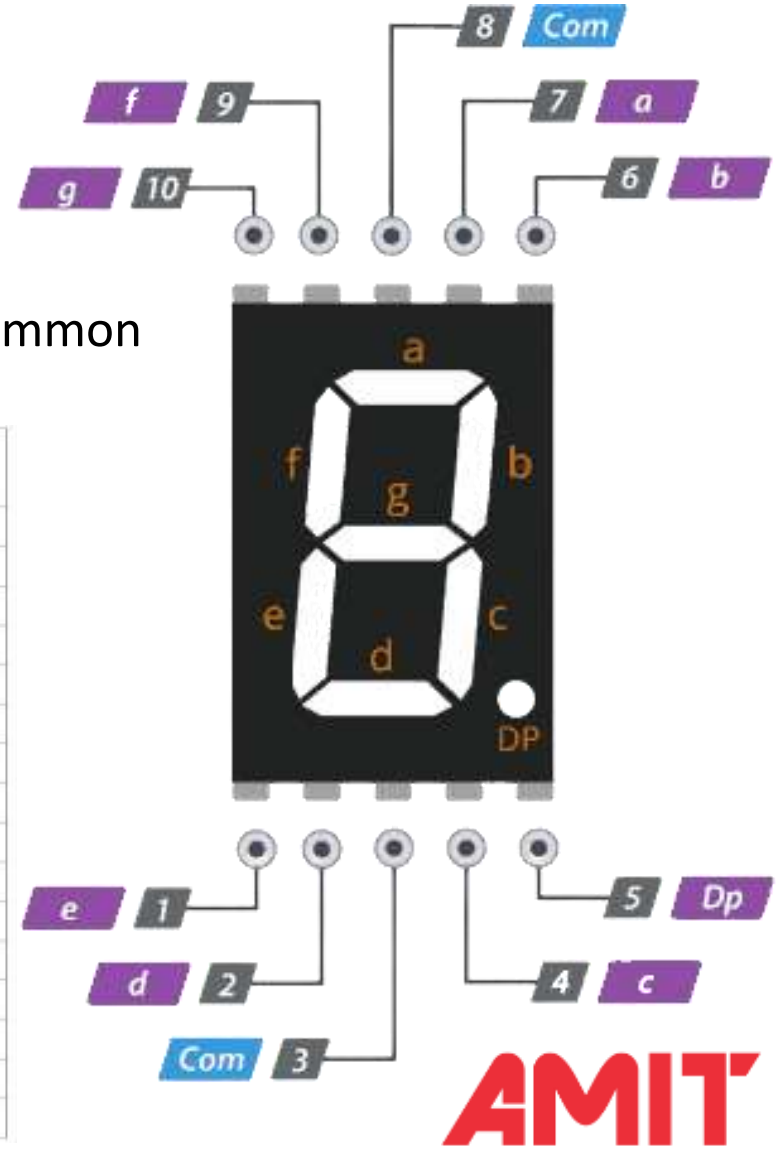
AMIT

Digital Input Output:

Seven Segment Display:

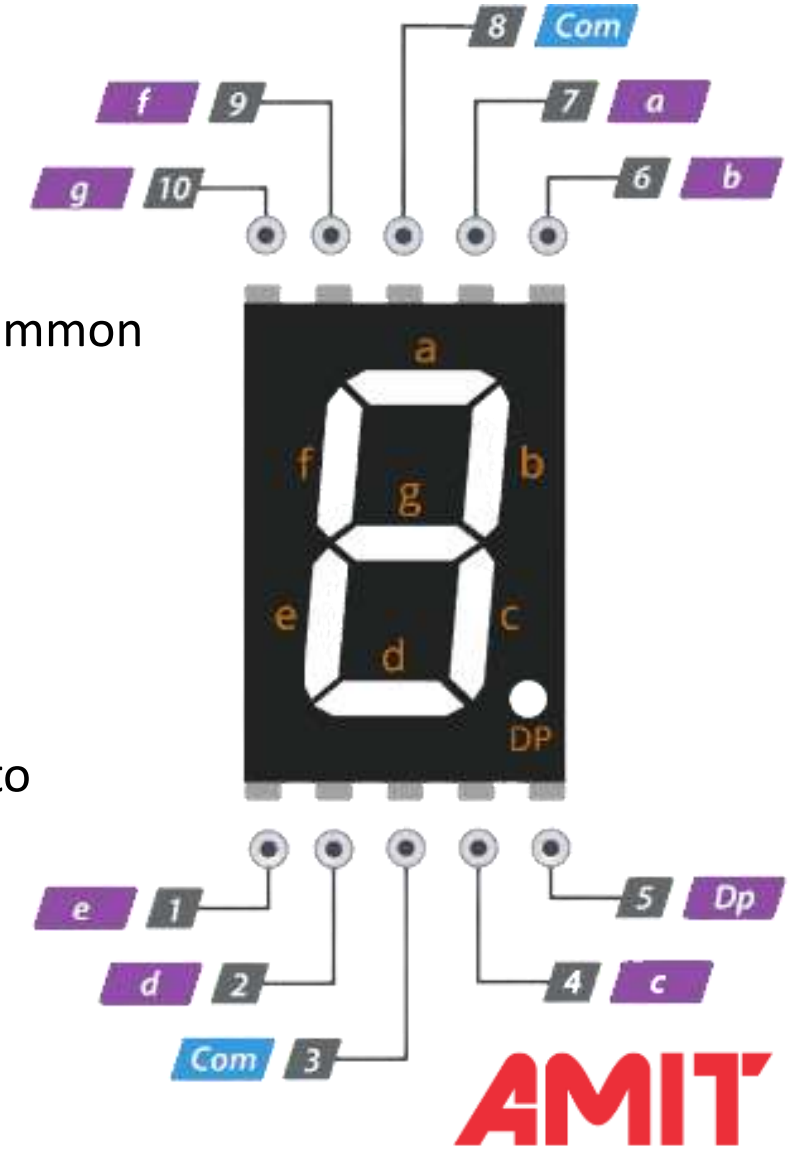
- **Common Cathode Seven Segment**:
  The activation of segment in this type is High Signal, and the Common Pin activation is Low Signal, So to Draw numbers from **0 → F**:

Digits	Display Segments							Hexadecimal
	g	f	e	d	c	b	a	
0	0	1	1	1	1	1	1	3F
1	0	0	0	0	1	1	0	06
2	1	0	1	1	0	1	1	5B
3	1	0	0	1	1	1	1	4F
4	1	1	0	0	1	1	0	66
5	1	1	0	1	1	0	1	6D
6	1	1	1	1	1	0	1	7D
7	0	0	0	0	1	1	1	07
8	1	1	1	1	1	1	1	7F
9	1	1	0	1	1	1	1	6F
A	1	1	1	0	1	1	1	77
B	1	1	1	1	1	0	0	7C
C	0	1	1	1	0	0	1	39
D	1	0	1	1	1	1	0	5E
E	1	1	1	1	0	0	1	79
F	1	1	1	0	0	0	1	71

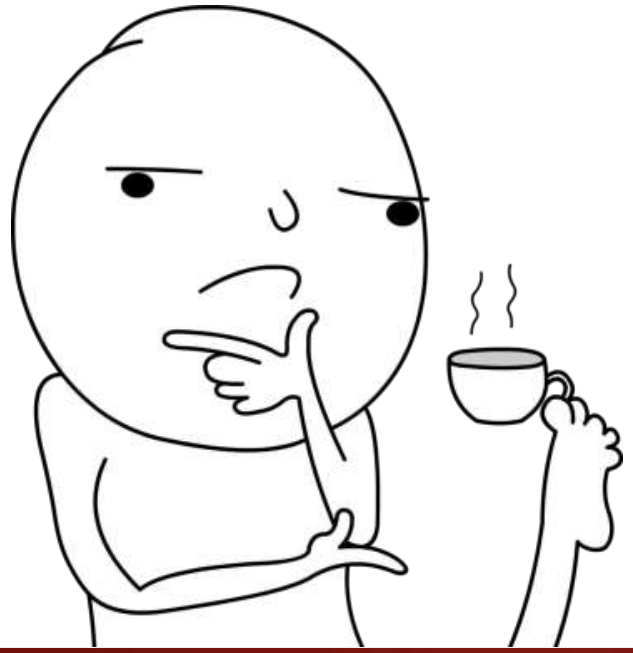


Digital Input Output:

Seven Segment Display:



- **Common Anode Seven Segment**:
  The activation of segment in this type is Low Signal, and the Common Pin activation is High Signal, So to Draw numbers from **0** → **F**:



just flip all activation level exists into the previous table or get the ONE's complement of bytes into the previous table.

AMIT

Labs On DIO :

GI:

- Make a system that has two switches, one of Them increases the value of Seven Segment, and the second decreases its value.

AMIT

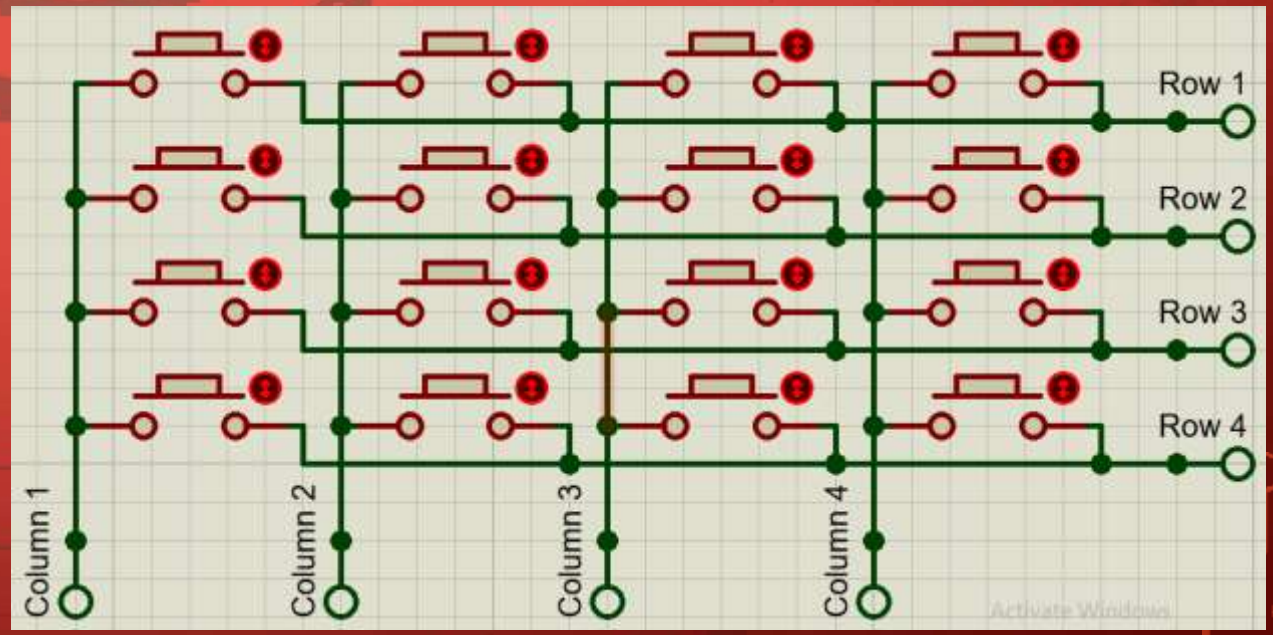
KEYPAD

KEYPAD CONCEPT:

- The keypad is a set of mechanical switches which are arranged into a matrix.

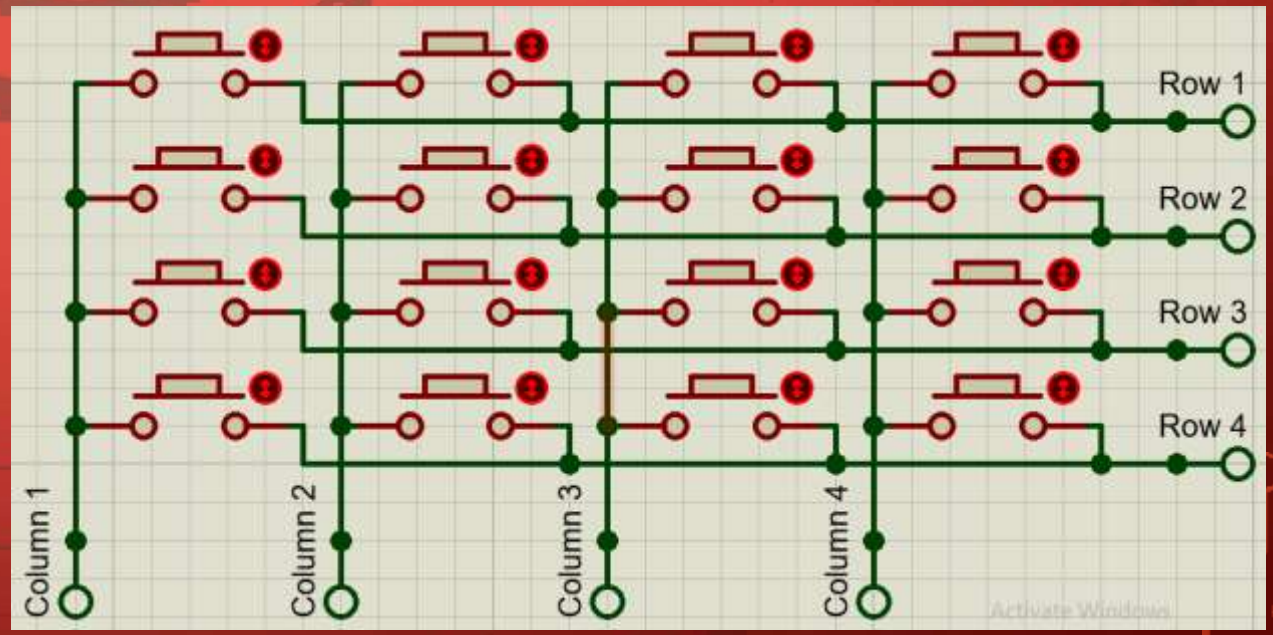
WHY KEYPAD?

- If we want to use about 16 switches into our project, we will need about 16 input pin from our microcontroller, about half of GPIO pins, but if we need about 49 switches, what shall we do?
- In this case, we need about 49 input pin for our switches, so if they are arranged into a matrix 7x7 we only need 14 pins only.



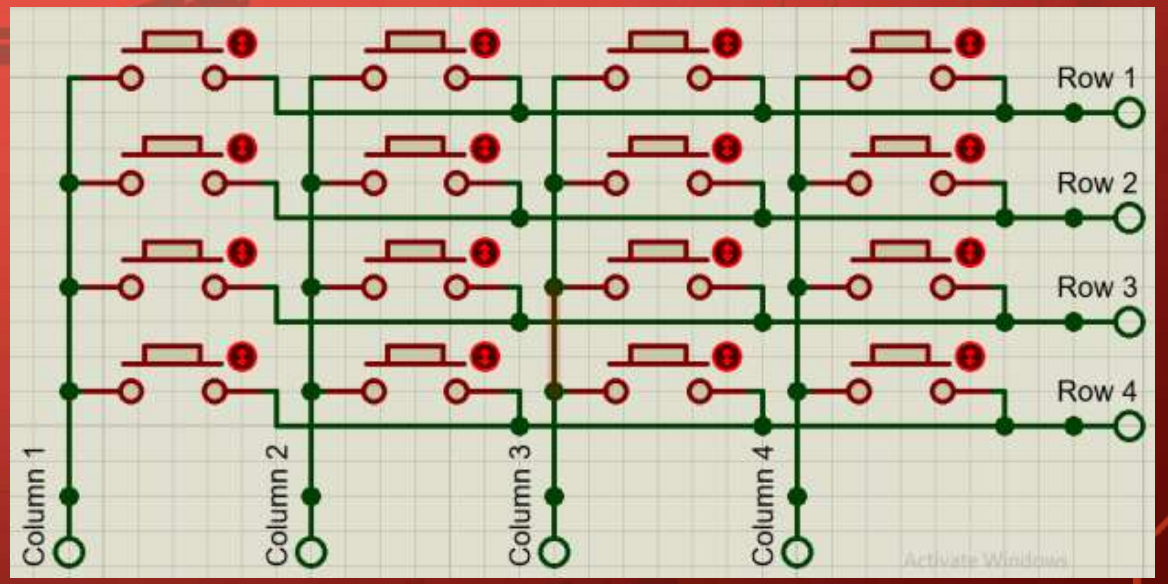
WHY KEYPAD?

- In the figure besides, keypad 4x4, so we only need about 8 pins to connect these switches into our system.
- **So**, the keypad concept is used to reduce the number of microcontroller pins.
- But in this case, how does the keypad work?



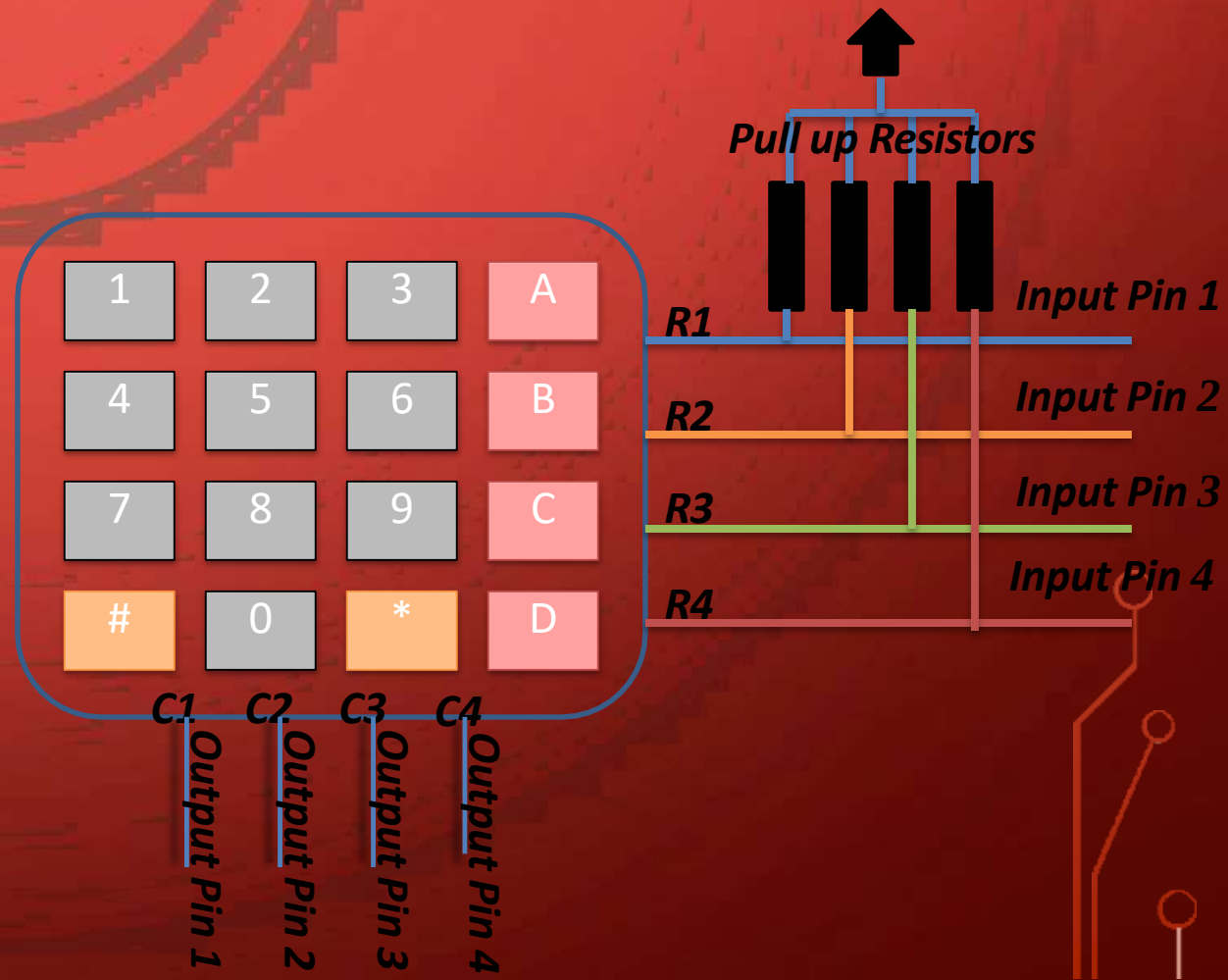
HOW KEYPAD WORKS?

- After arranging the switches into a matrix, we will have four columns and for rows, So, we can set any set of them as an output and the other as an input.
- Assume that the columns set is output, and the rows set is input, what is the kind of the signals must be written on these columns to easily indicate the pressed button?



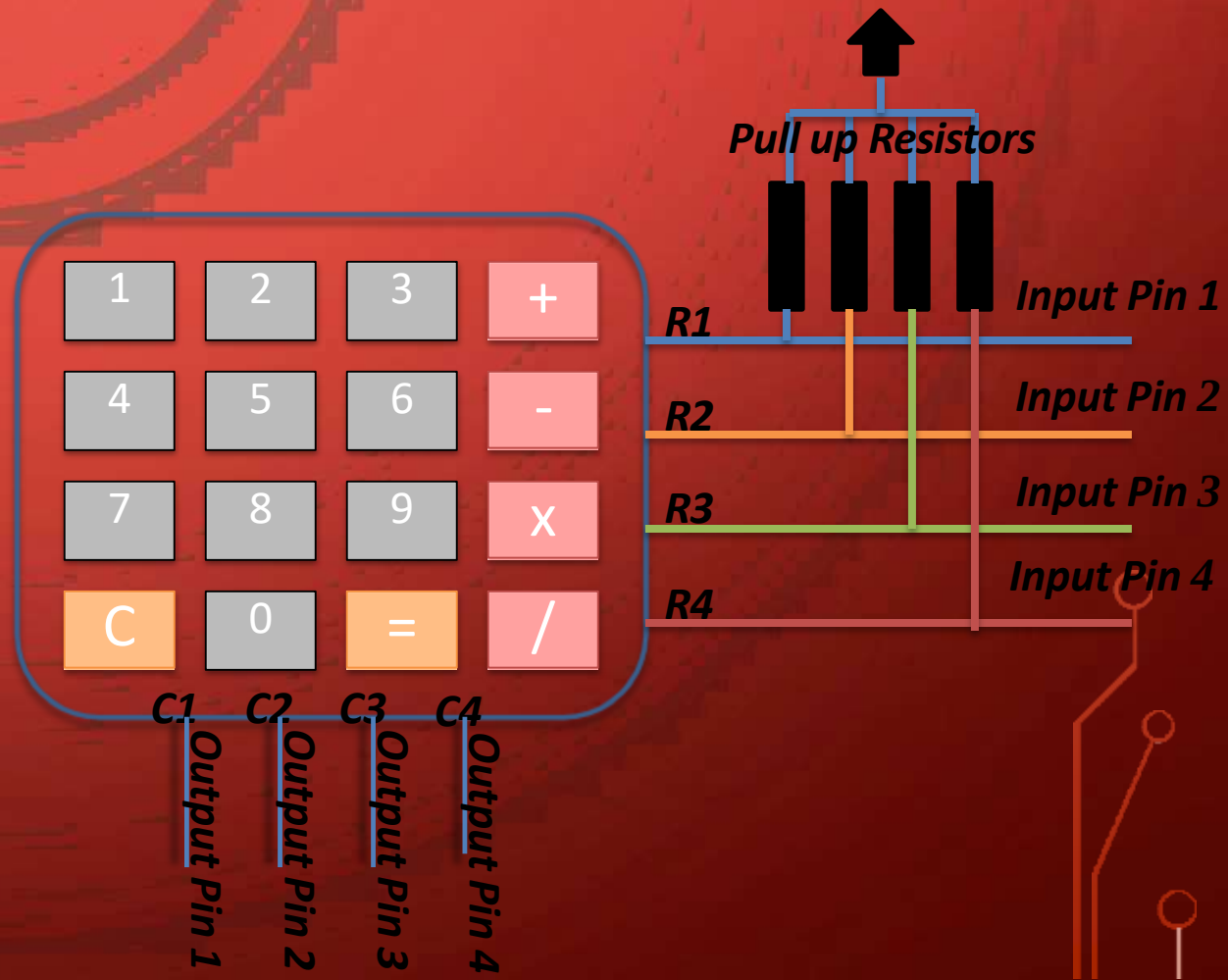
HOW KEYPAD WORKS?

- The best sequence is to set all the columns by HIGH at first.
- Then, change C1 signal to LOW then check if any row read Zero or not.
- Assume that R3 reads Zero signal, so the pressed button is C1R3.
- If not, turn C1 to HIGH again and set C2 to LOW and check if any Row reads Zero or not.
- Repeat this sequence until you find zero on any row then return the pressed key, or until columns are finished and return keypad not pressed.



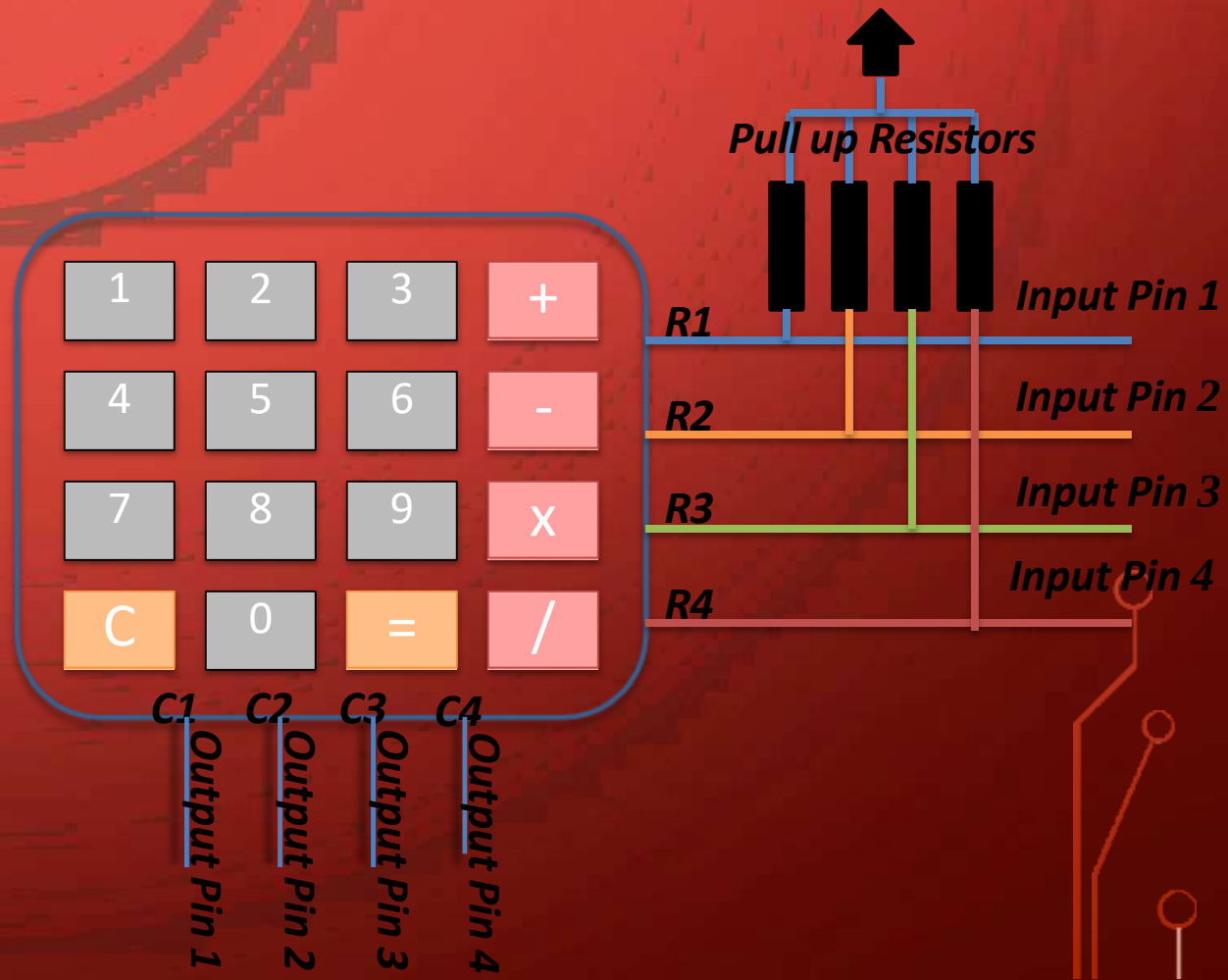
HOW KEYPAD WORKS?

- The best sequence is to set all the columns by HIGH at first.
- Then, change C1 signal to LOW then check if any row read Zero or not.
- Assume that R3 reads Zero signal, so the pressed button is C1R3.
- If not, turn C1 to HIGH again and set C2 to LOW and check if any Row reads Zero or not.
- Repeat this sequence until you find zero on any row then return the pressed key, or until columns are finished and return keypad not pressed.



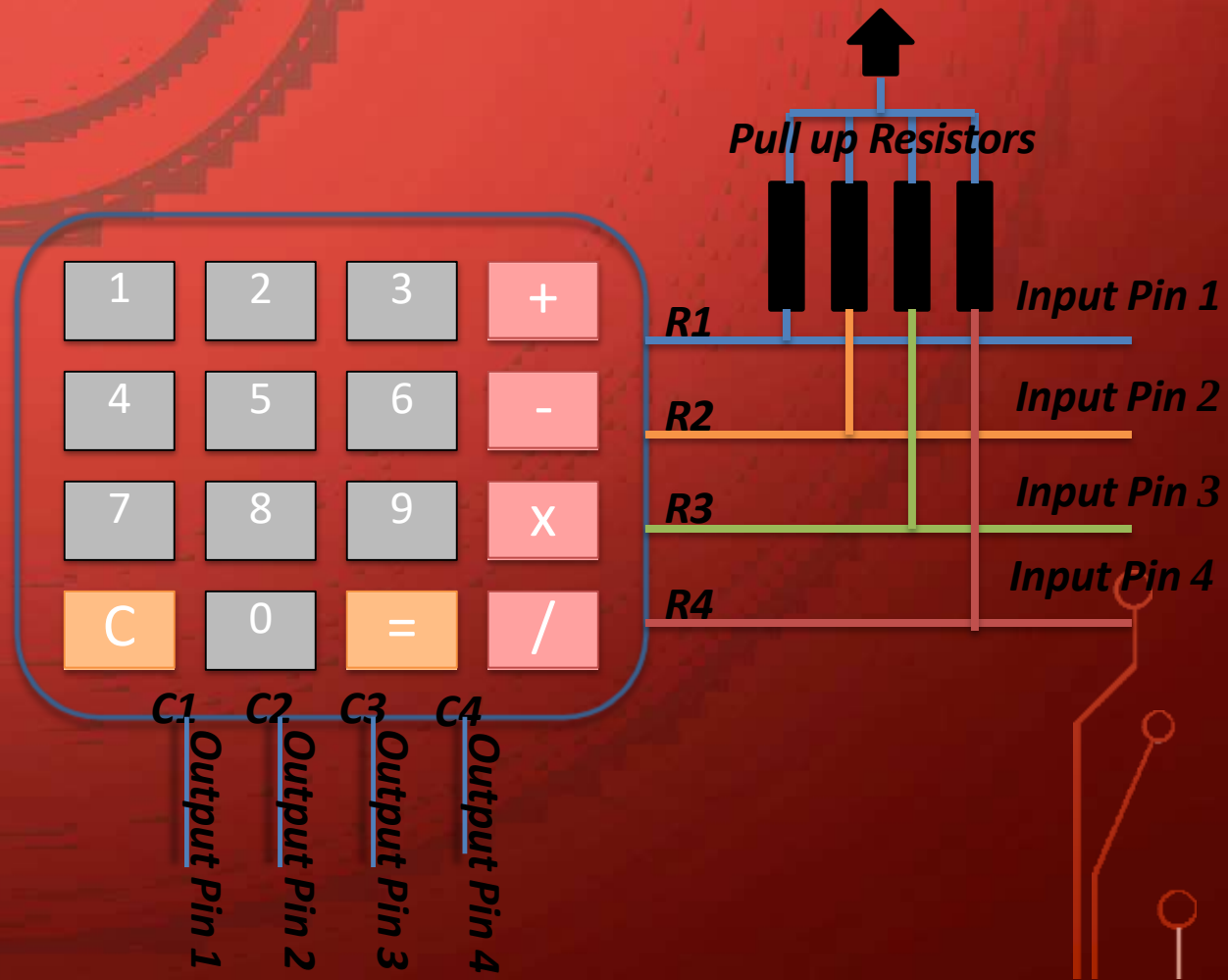
KEYPAD ALGORITHM:

- ***Write on C1 To C4 "0111E:***
  - This means that C1 is activated, and we are going to check the buttons in C1, If any row of the rows R1 to R4 is set to 0, then the corresponding button in C1 is pressed.
- ***Write on C1 To C4 "1011E:***
  - This means that C2 is activated, and we are going to check the buttons in C2, If any row of the rows R1 to R4 is set to 0, then the corresponding button in C1 is pressed.



KEYPAD ALGORITHM:

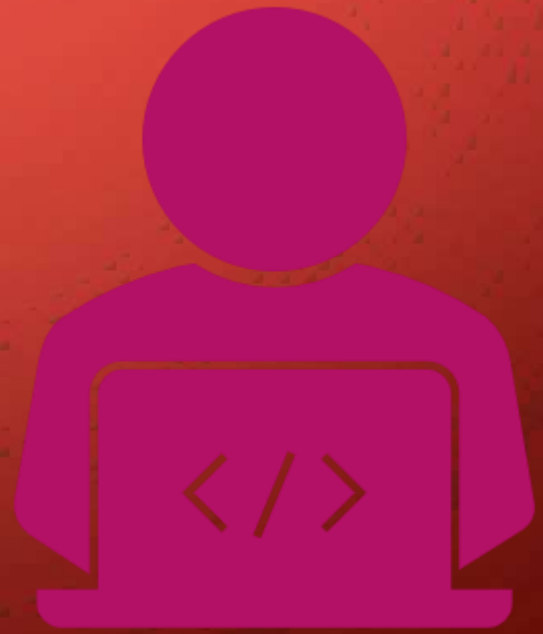
- ***Write on C1 To C4 "1101E:***
  - This means that C4 is activated, and we are going to check the buttons in C3, If any row of the rows R1 to R4 is set to 0, then the corresponding button in C1 is pressed.
- ***Write on C1 To C4 "1110E:***
  - This means that C4 is activated, and we are going to check the buttons in C4, If any row of the rows R1 to R4 is set to 0, then the corresponding button in C1 is pressed.



KEYPAD DRIVER:

- ***Now,*** *After understanding how to implement the Keypad matrix and the Algorithm of it,*

Try writing its software module



STEPPER MOTORS

STEPPER MOTORS:

- **What is the Stepper Motor?:**
  - A stepper motor, also known as step motor or stepping motor, is a brushless DC electric motor that divides a full rotation into several equal steps. The motor's position can be commanded to move and hold at one of these steps without any position sensor for feedback (an open-loop controller in contrast to Servo which is a closed Loop system), if the motor is correctly sized to the application in respect to torque and speed.

STEPPER MOTORS:

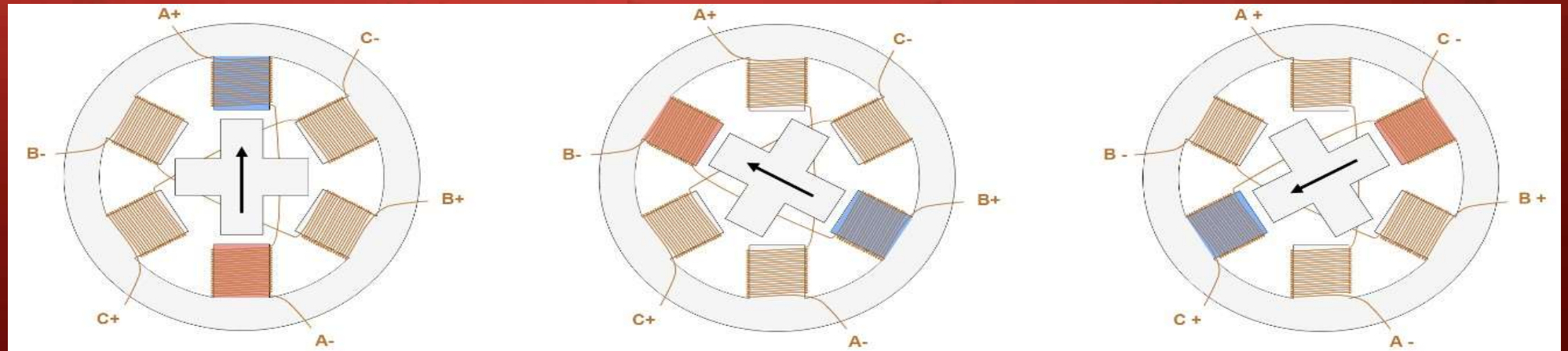
- **Why the Stepper Motor:**
  - Stepper motors are often an extremely important component in a motion control system.
  - Stepper motors can produce full, instantaneous torque - even from a standstill. This makes them very useful for motion control applications, where accuracy, repeatability, and power are paramount.

STEPPER MOTORS:

- **Types of Stepper Motors:**
  - Permanent magnet stepper
  - Variable reluctance stepper
  - Hybrid synchronous stepper
- **Types of Phases of Stepper Motors:**
  - **Unipolar.**
  - **Bipolar.**

THEORY OF STEPPER MOTORS:

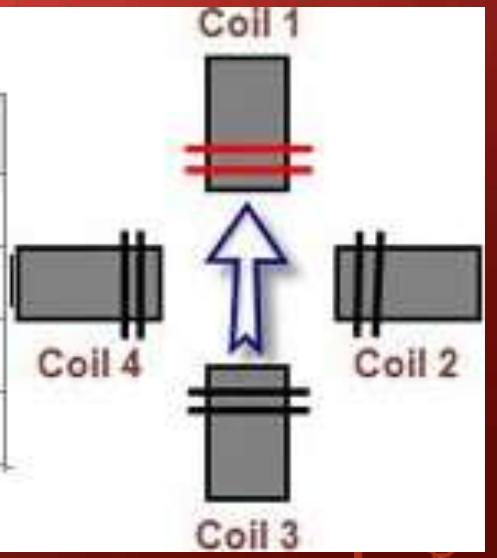
- **Types of Stepper Motors:**
  - The basic working principle of the stepper motor is the following: By energizing one or more of the stator phases, a magnetic field is generated by the current flowing in the coil and the rotor aligns with this field. By supplying different phases in sequence, the rotor can be rotated by a specific amount to reach the desired final position. Figure 2 shows a representation of the working principle. At the beginning, coil A is energized, and the rotor is aligned with the magnetic field it produces. When coil B is energized, the rotor rotates clockwise by 60° to align with the new magnetic field. The same happens when coil C is energized. In the pictures, the colors of the stator teeth indicate the direction of the magnetic field generated by the stator winding.



THEORY OF STEPPER MOTORS:

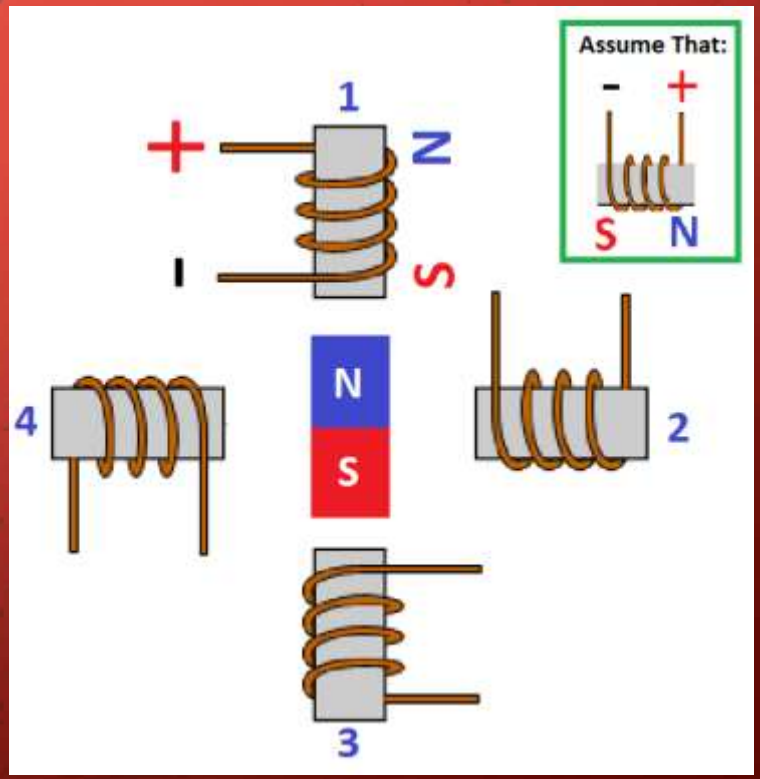
- **Four Coils Full Step Activation:**
  - In this type of stepper motors, the coils must be activated sequentially, as the following figure, the activated coils are 1 → 2 → 3 → 4 to generate Clockwise rotation.
  - To reverse the rotation, the sequence of activation will be 1→4→3→2.

Pulse	Coil 1	Coil 2	Coil 3	Coil 4
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

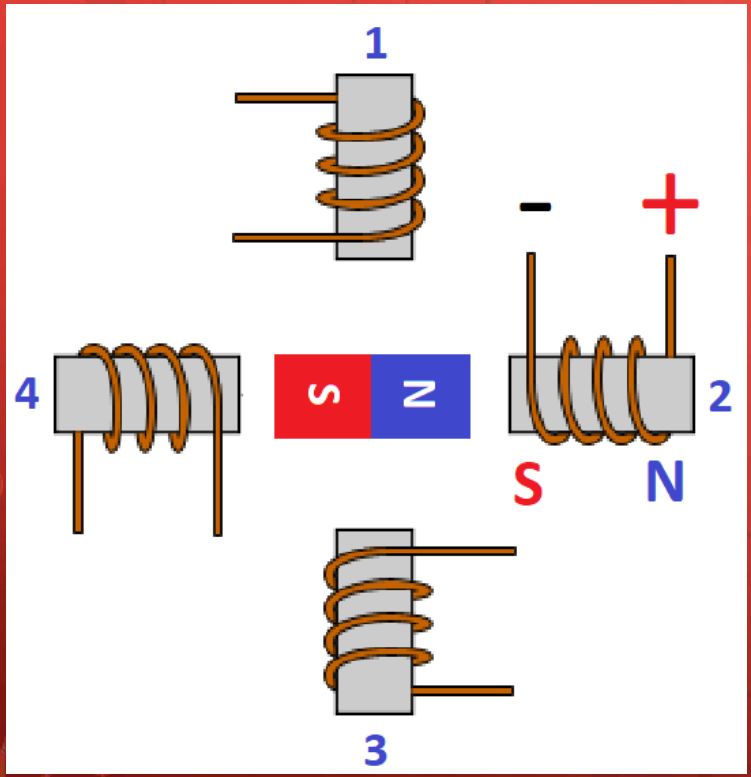


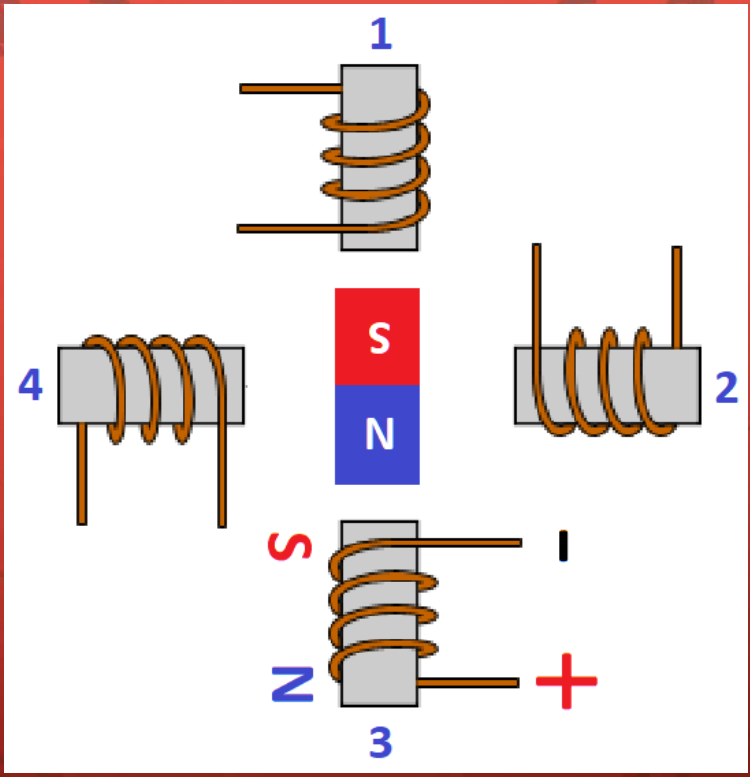
THEORY OF STEPPER MOTORS:

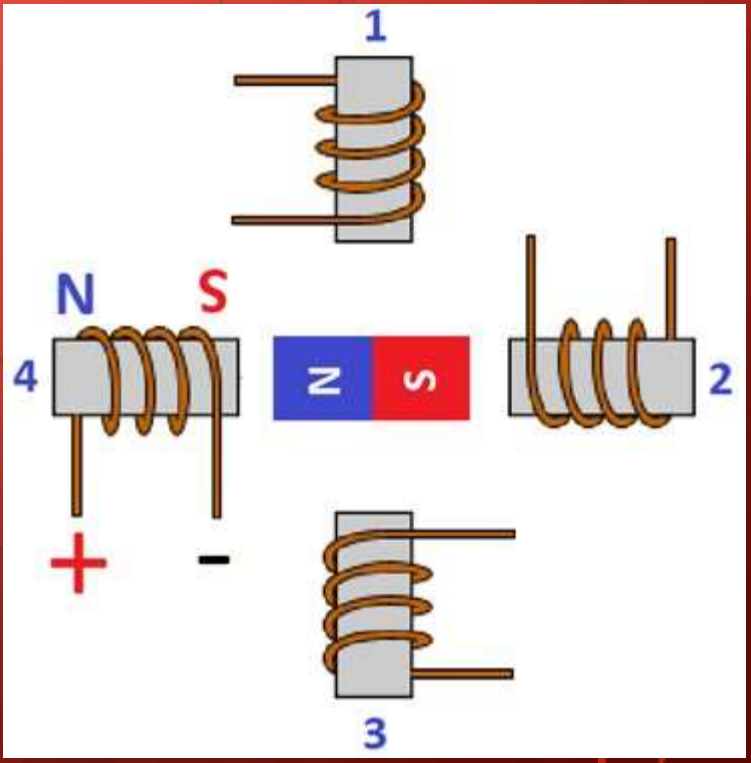
- **<u>Four Coils Full Step Activation:</u>**
  - Assume that if the current flows into the coil in clockwise direction, the North will be generated at left, South at right.
  - So, the south pole of electrical magnetic field will attract the North pole of permanent magnet.
  - After that, Deactivate Coil **1,** then activate the next coil.
  - Apply this sequence, Deactivate the current coil, then activate the next.
  - This type is called **Bipolar activation**.



THEORY OF STEPPER MOTORS:





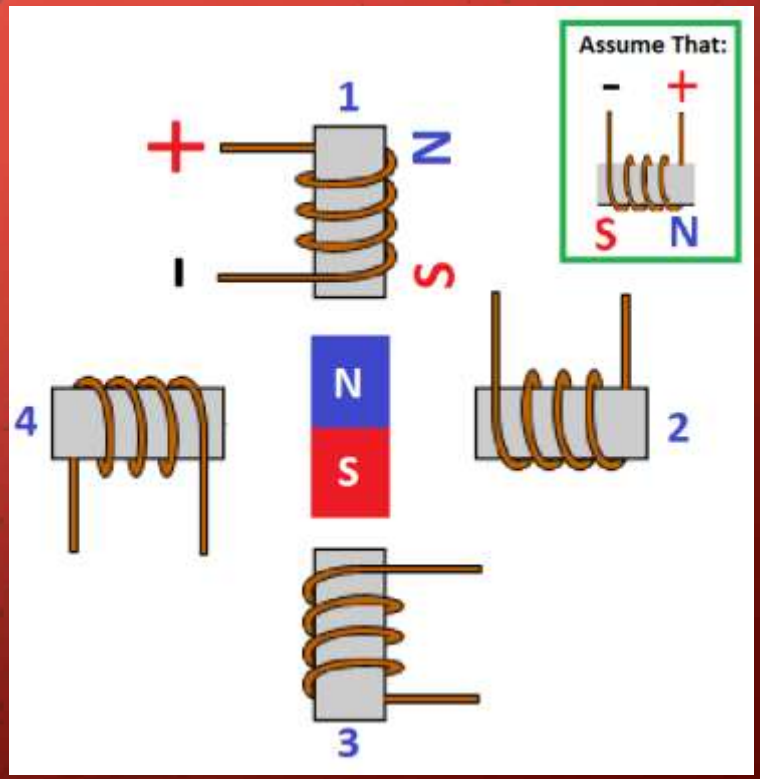


After applying this sequence, the rotor rotates at clockwise direction

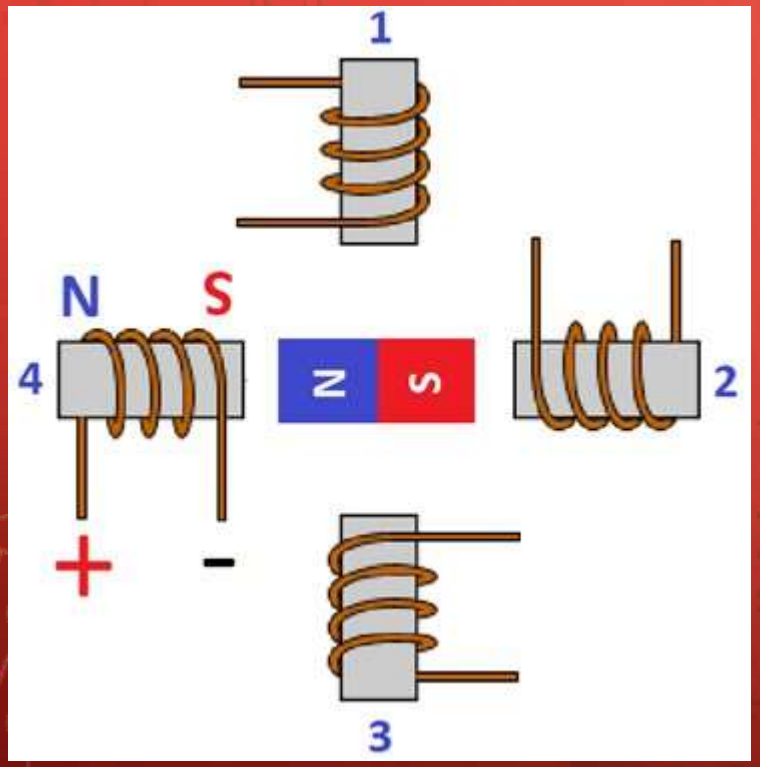
THEORY OF STEPPER MOTORS:

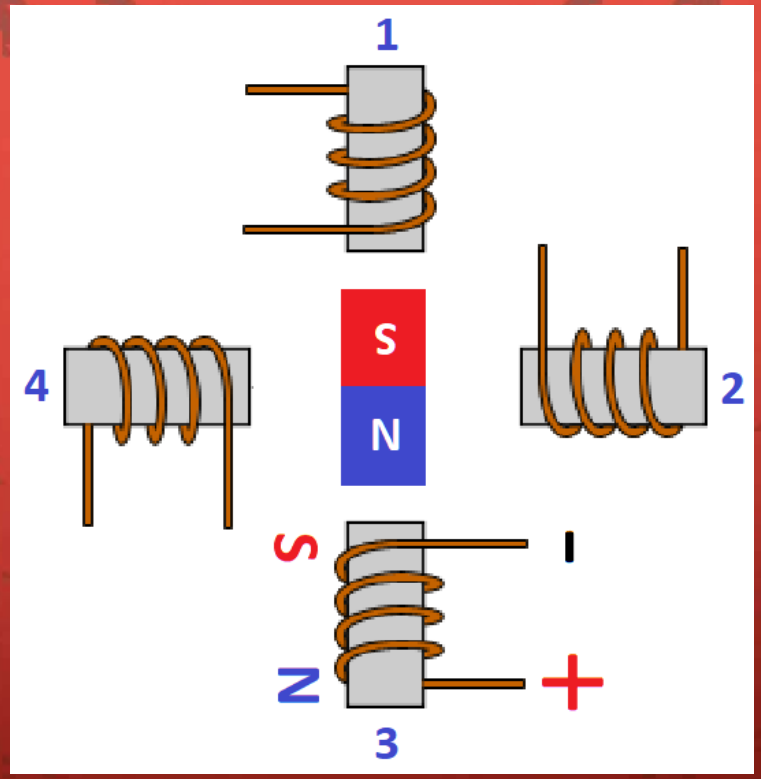
- **<u>Four Coils Full Step Activation:</u>**
  - To change the direction of rotation, there are two methods:
    - Changing the activation sequence:

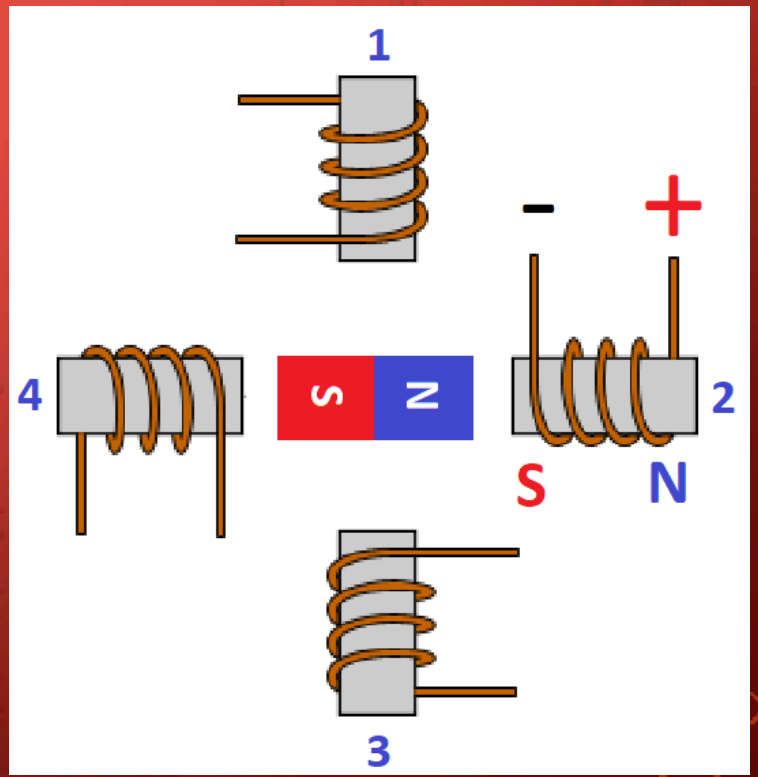
      1 → → 4 → 3 → 2



THEORY OF STEPPER MOTORS:



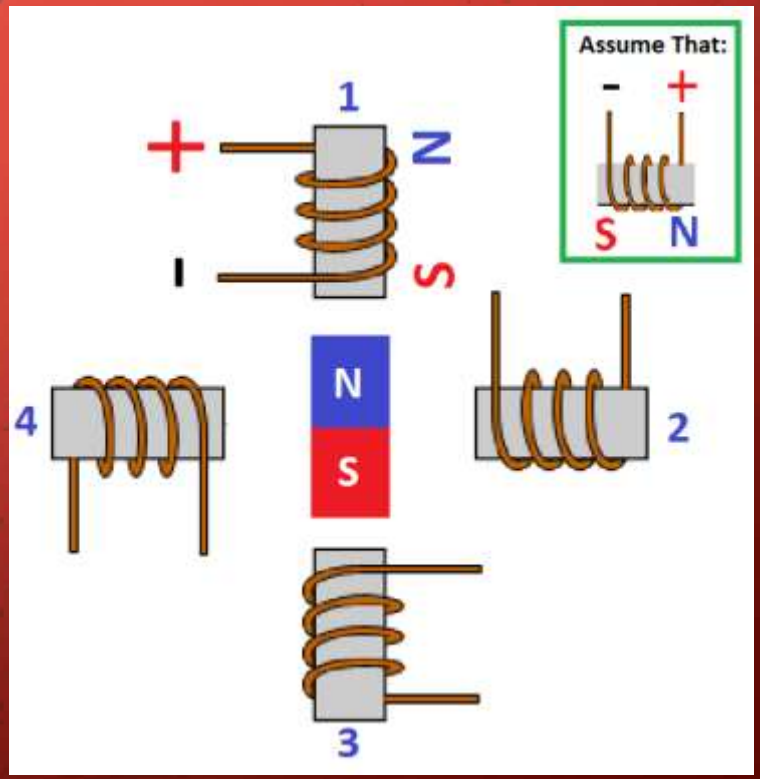




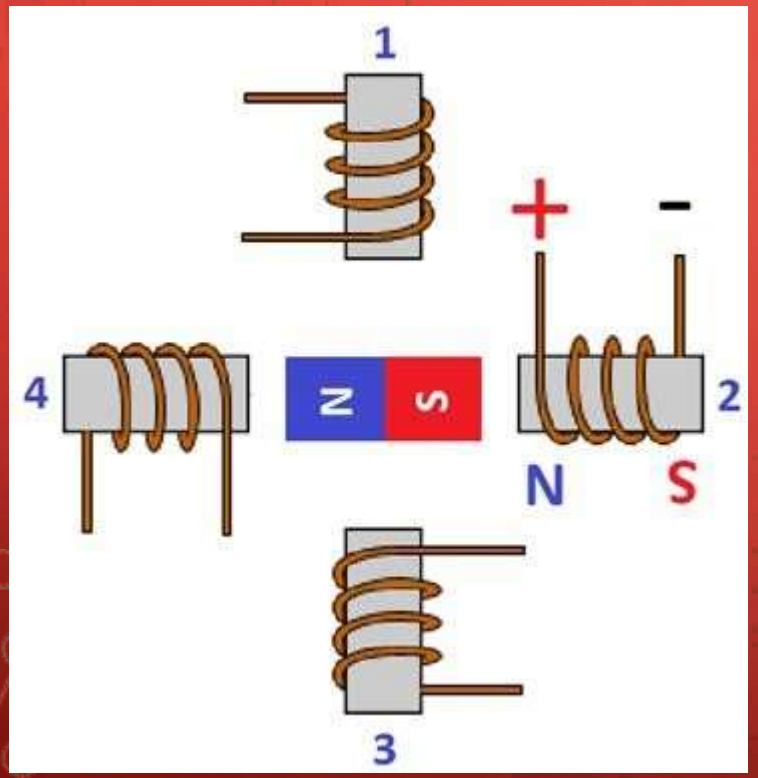
After applying this sequence, the rotor rotates at counter-clockwise direction

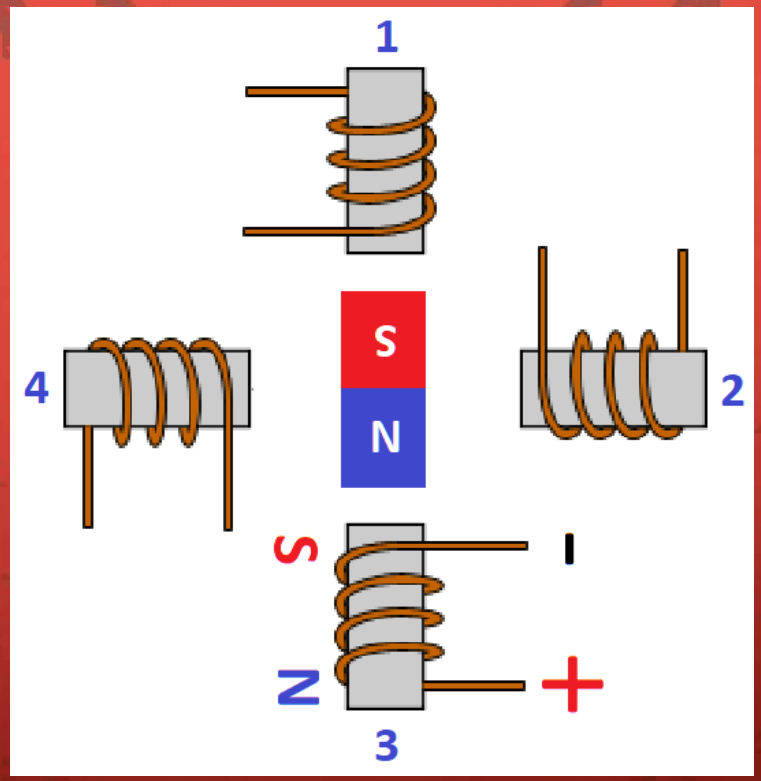
THEORY OF STEPPER MOTORS:

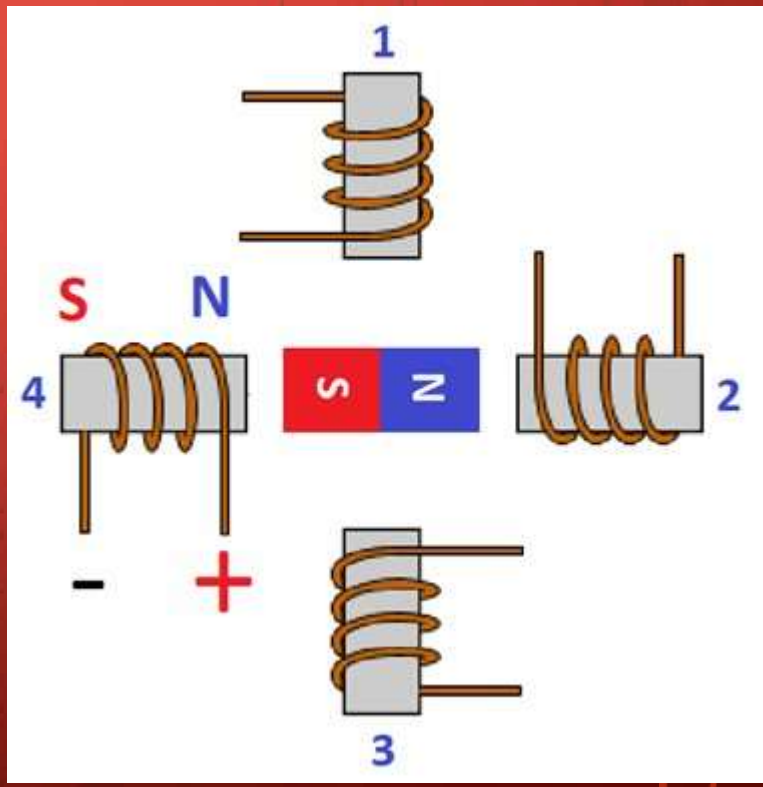
- **<u>Four Coils Full Step Activation:</u>**
  - To change the direction of rotation, there are two methods:
    - Changing the polarity of some coils.



THEORY OF STEPPER MOTORS:



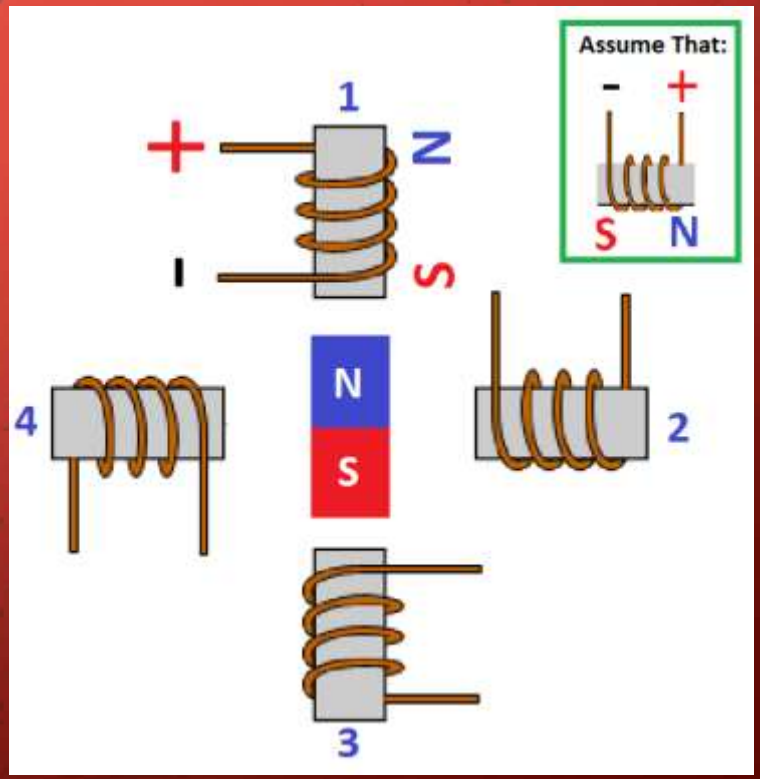




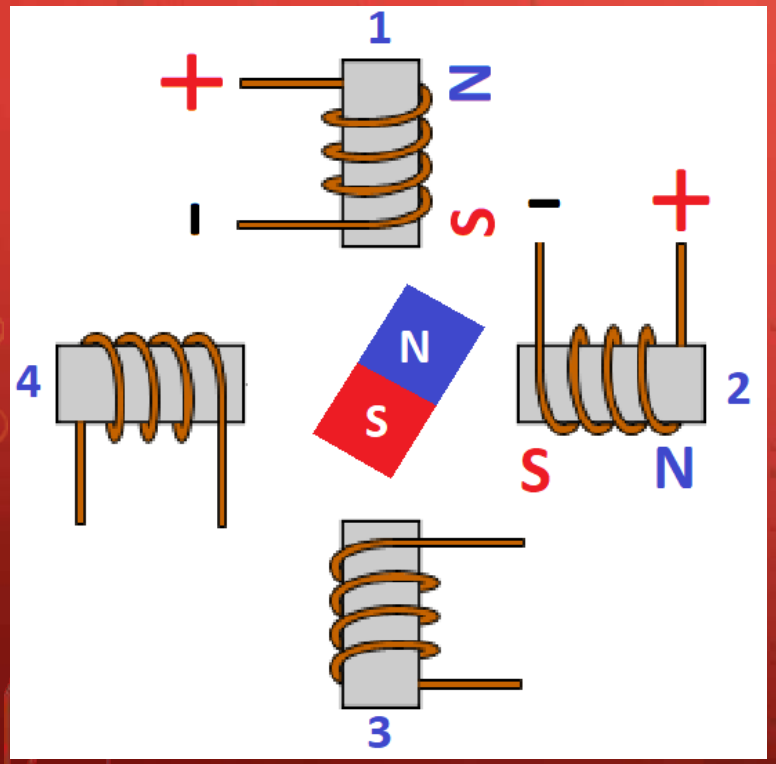
After applying this sequence, the rotor rotates at counter-clockwise direction

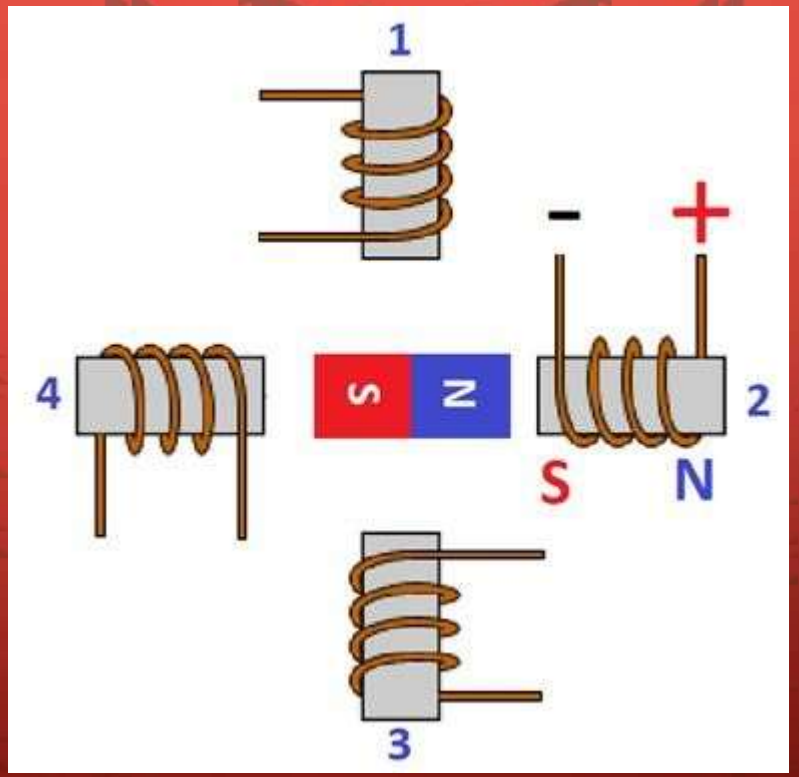
THEORY OF STEPPER MOTORS:

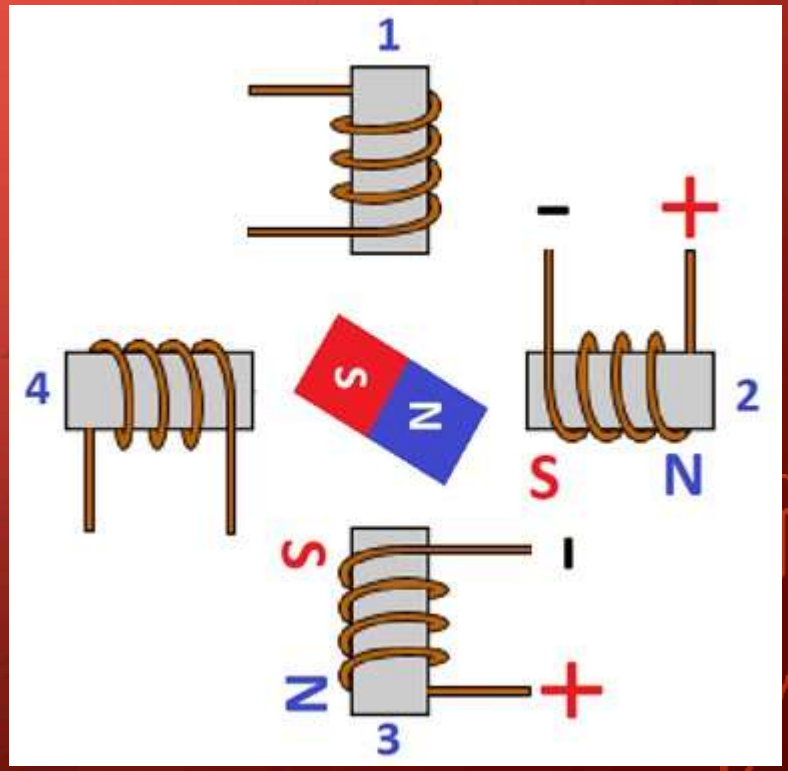
- **Four Coils Half Step Activation:**
  - Assume that if the current flows into the coil in clockwise direction, the North will be generated at left, South at right.
  - So, the south pole of electrical magnetic field will attract the North pole of permanent magnet.
  - After that, activate Coil **1** and the next coil to generate half step.
  - Apply this sequence.
  - This type is called **Bipolar activation**.



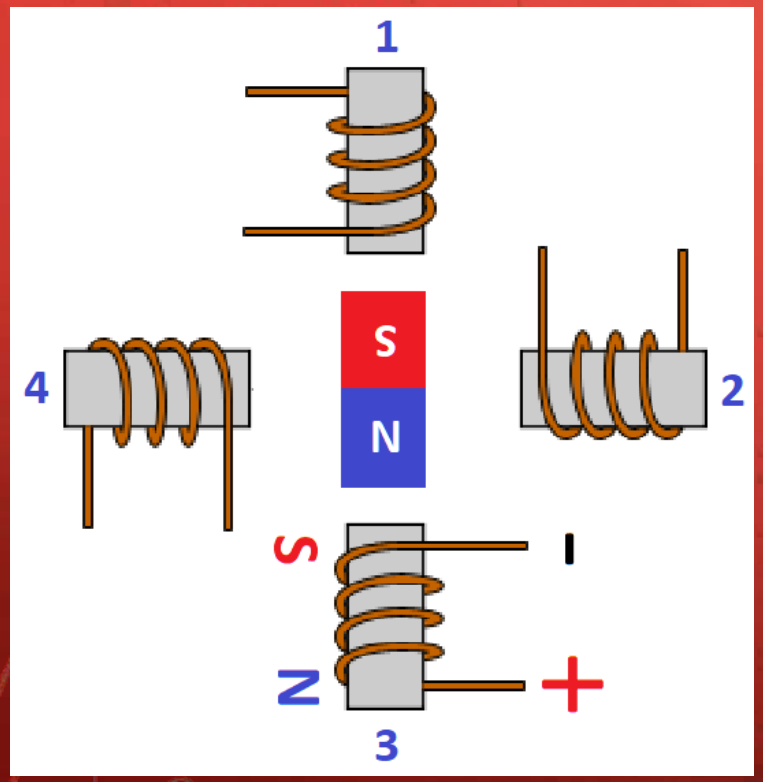
THEORY OF STEPPER MOTORS:

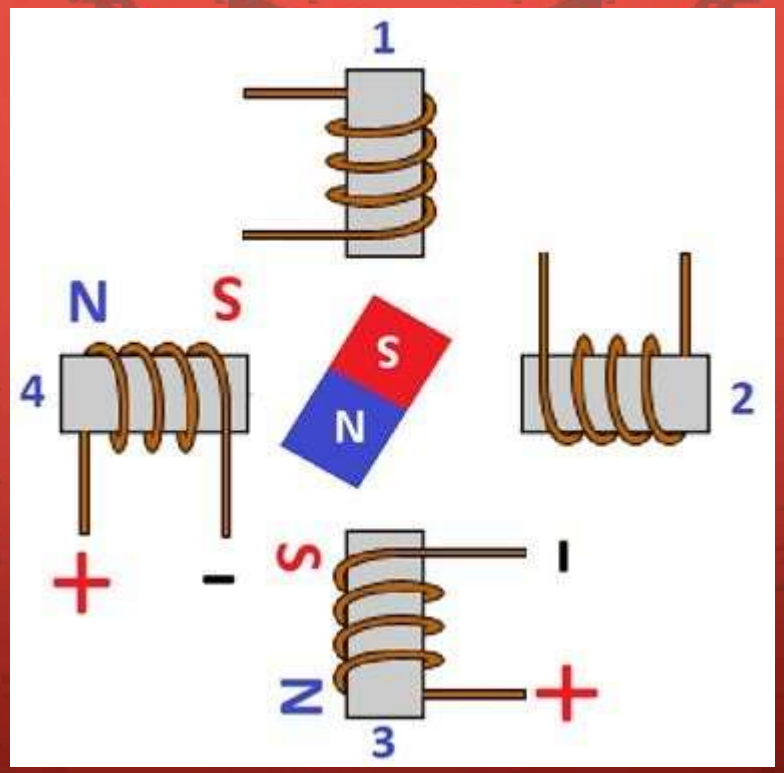


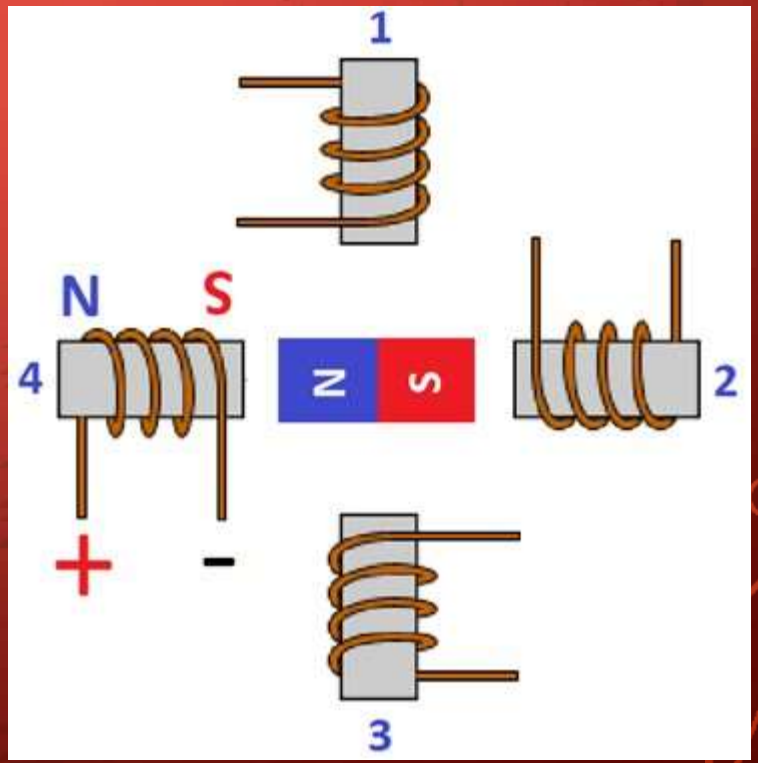




THEORY OF STEPPER MOTORS:

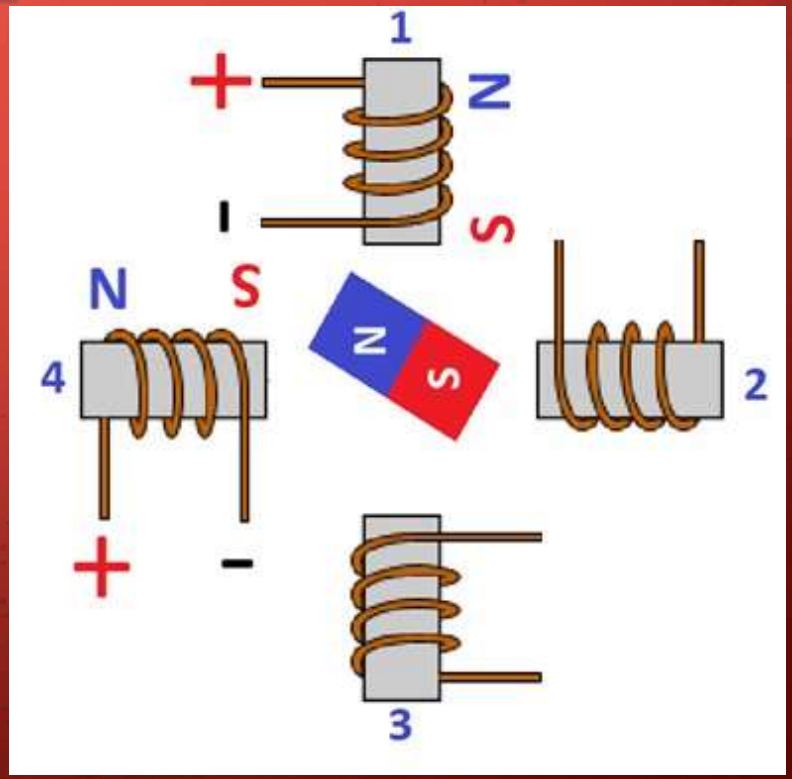






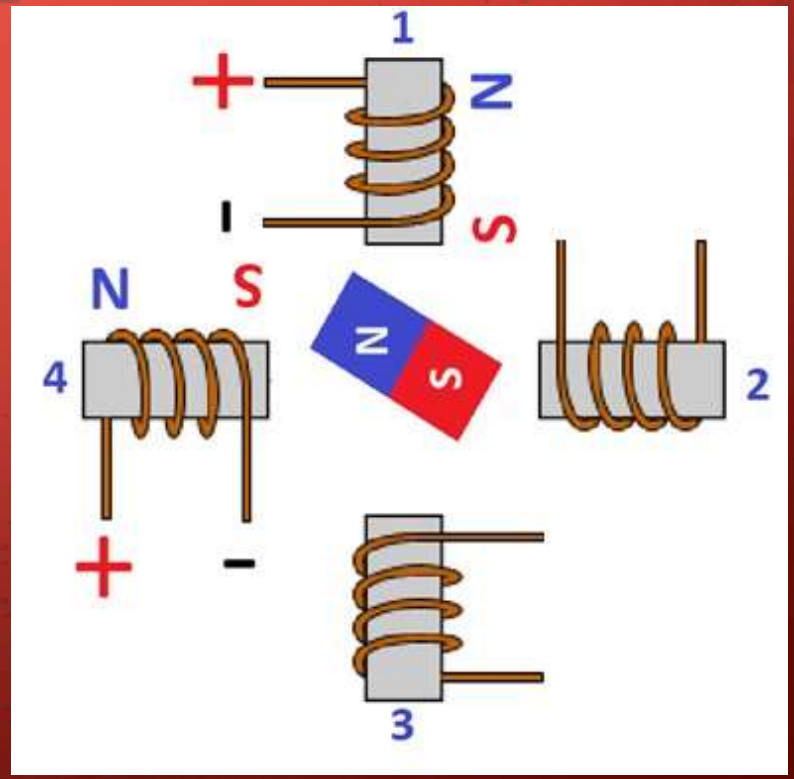
THEORY OF STEPPER MOTORS:

- **Four Coils Half Step Activation:**
  - It takes Eight steps for a completely cycle.
  - The stride angle become half of the full-step angle.
  - It can be reversed by the same way into full step by changing the activation sequence or by changing the polarity of some of coils.



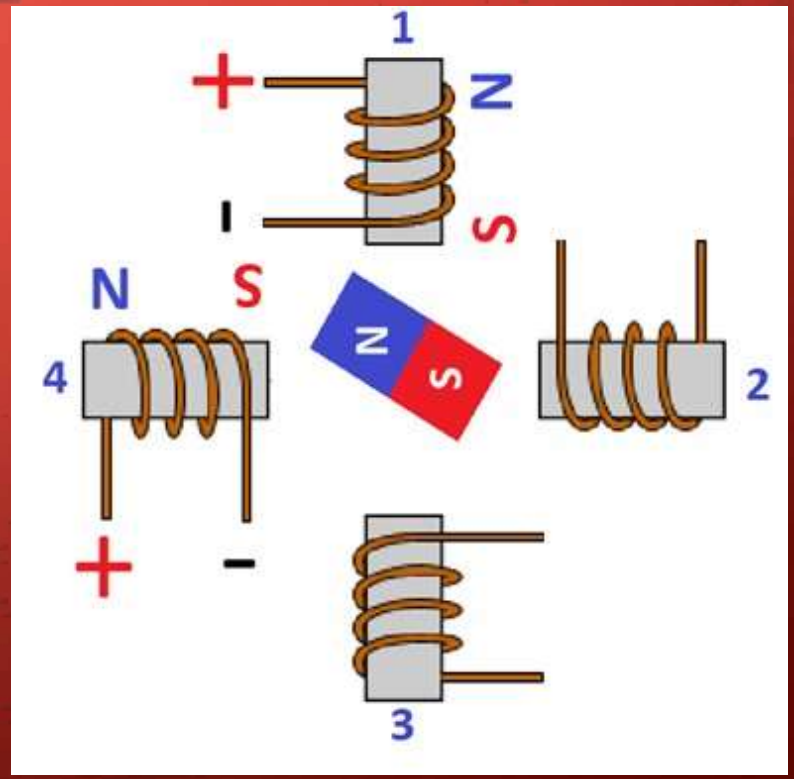
THEORY OF STEPPER MOTORS:

- **Four Coils Half Step Activation:**
  - Micro stepping:
    - It can be seen as a further enhancement of half-step mode, because it allows to reduce even further the step size and to have a constant torque output. This is achieved by controlling the intensity of the current flowing in each phase. Using this mode requires a more complex motor driver compared to the previous solutions.



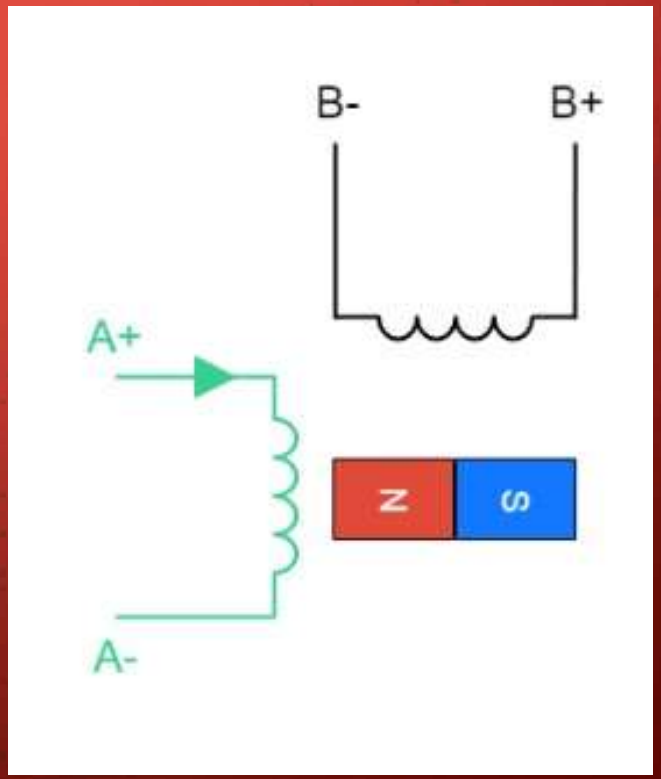
THEORY OF STEPPER MOTORS:

- **<u>Four Coils Half Step Activation:</u>**
  - Micro stepping:
    - Brief of micro stepping is to activate the different coils by different voltage, I meant that as example, at 1,2 activation, the voltage on coil 2 is reduced to generate several step angles.

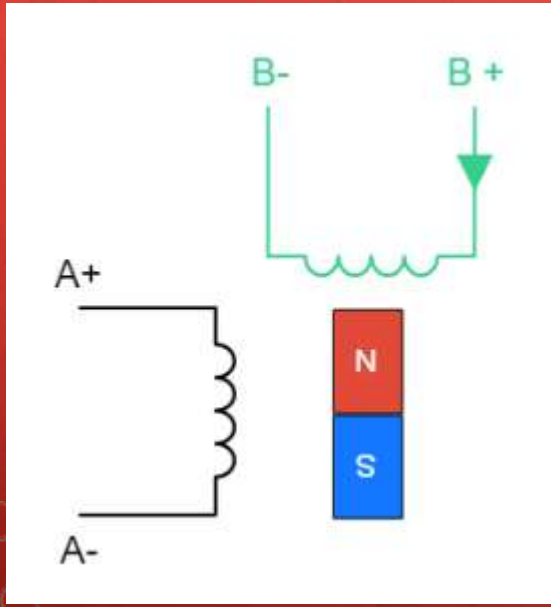


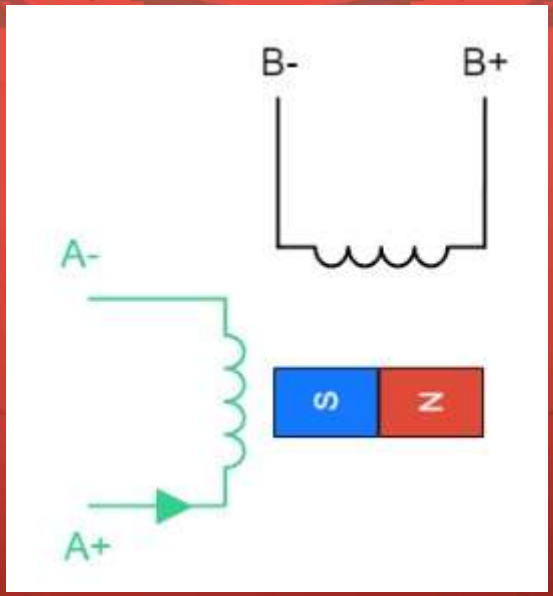
THEORY OF STEPPER MOTORS:

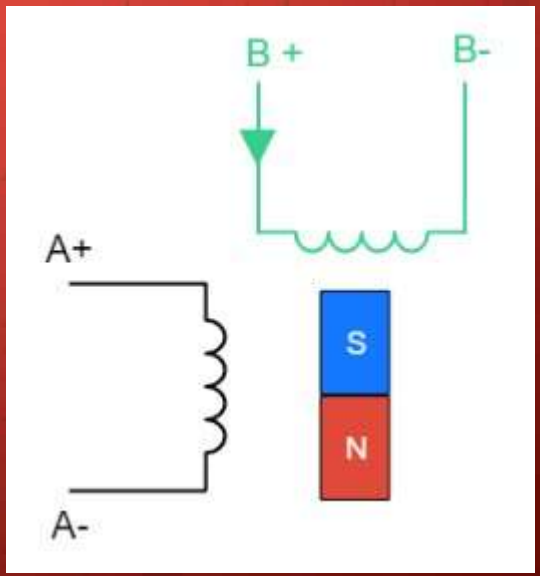
- **Two Coils Full Step Activation:**
  - Assume that if the current flows into the coil in clockwise direction, the North will be generated at left, South at right.
  - So, the south pole of electrical magnetic field will attract the North pole of permanent magnet.
  - After that, Deactivate Coil **1,** then activate the next coil.



THEORY OF STEPPER MOTORS:





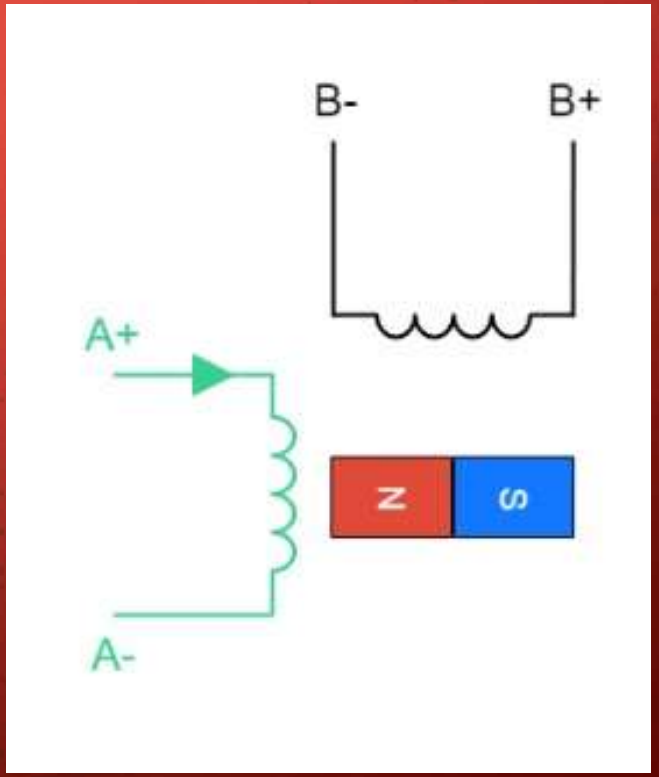


After deactivation of the coil 2, the first coil will be activated in inverse polarity.

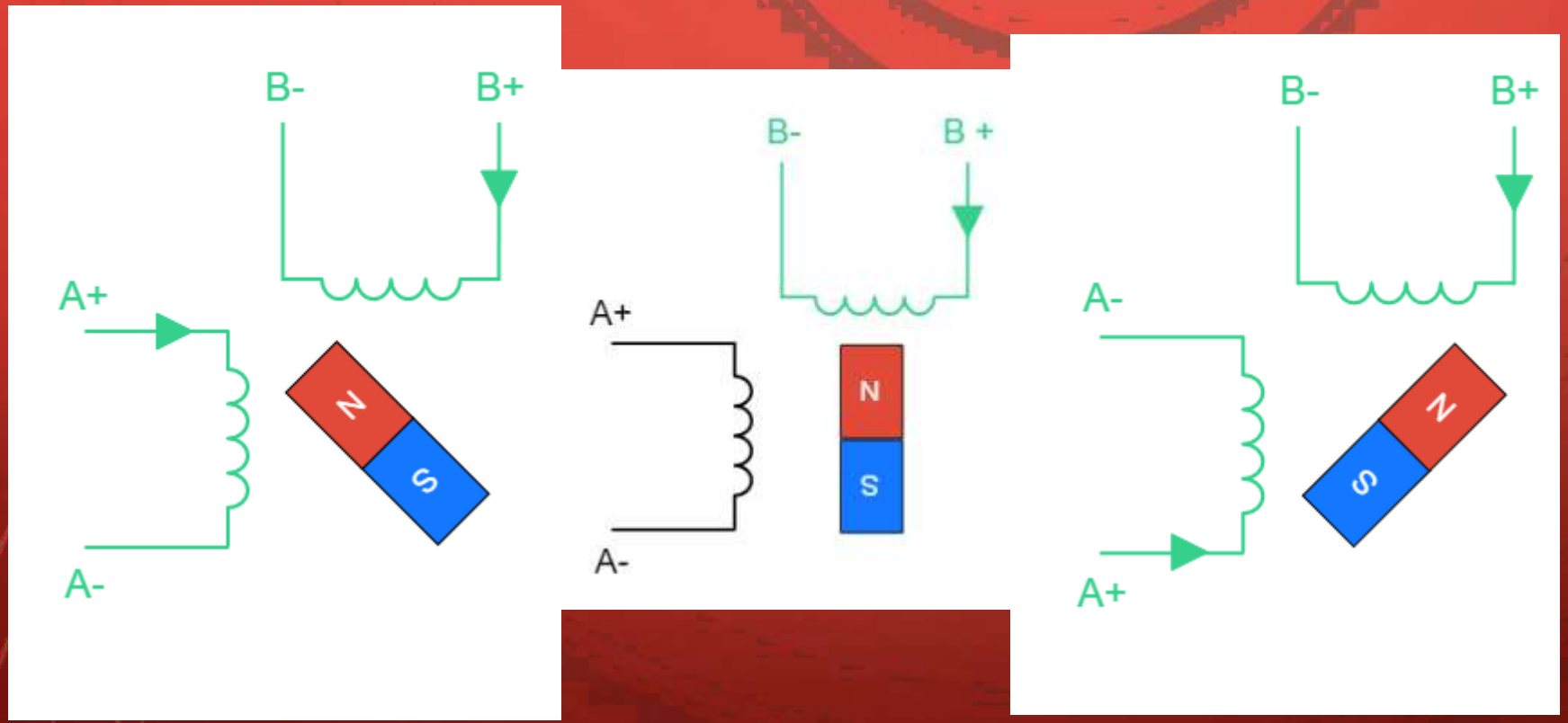
Final step, coil 2 also will be activated in inverse polarity.

THEORY OF STEPPER MOTORS:

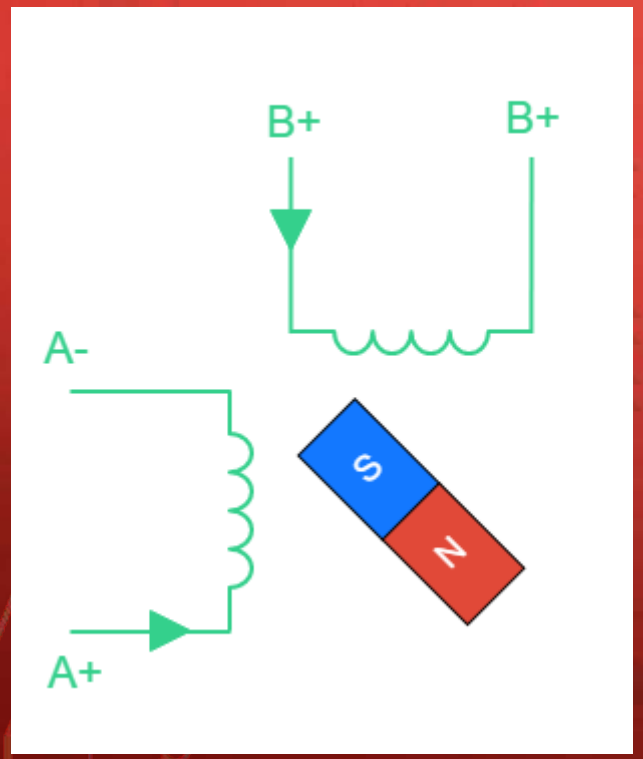
- **Two Coils Half Step Activation:**
  - Assume that if the current flows into the coil in clockwise direction, the North will be generated at left, South at right.
  - So, the south pole of electrical magnetic field will attract the North pole of permanent magnet.
  - After that, activate Coil 1 and activate the next coil.

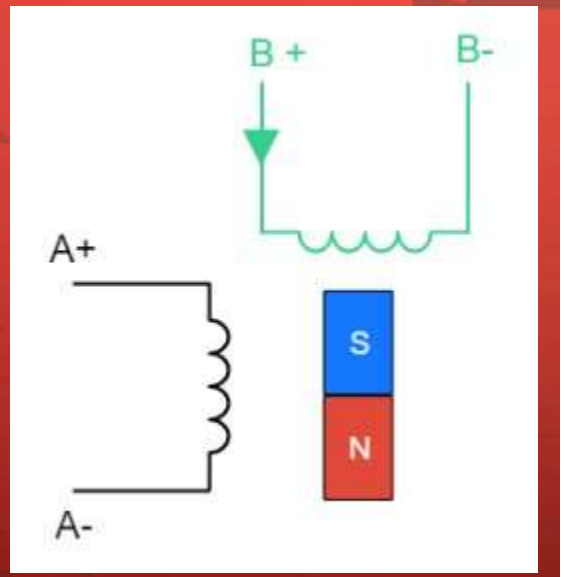


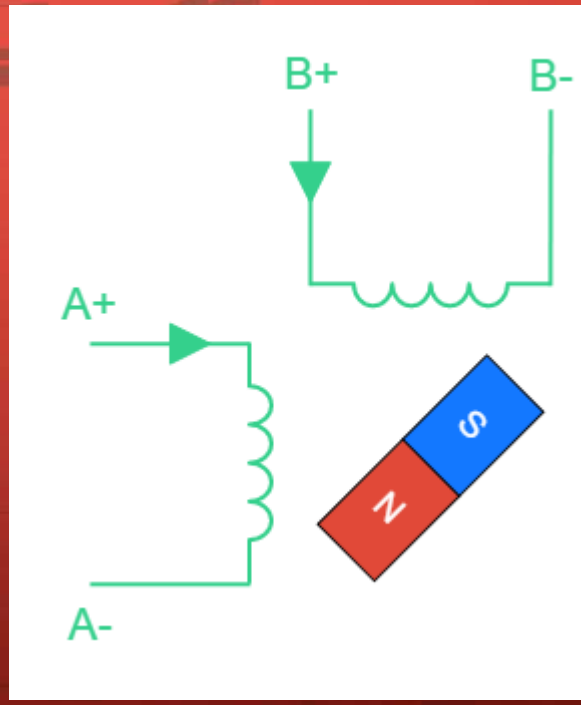
THEORY OF STEPPER MOTORS:

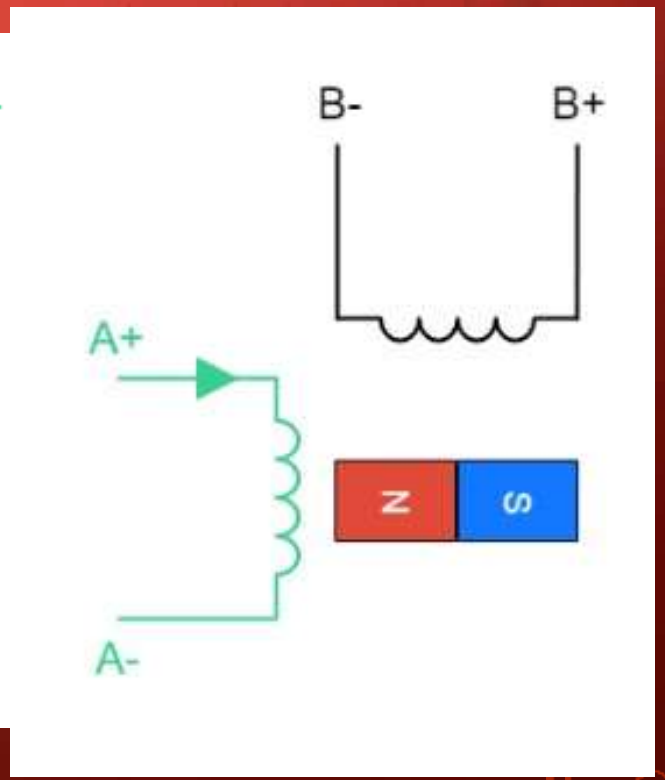


THEORY OF STEPPER MOTORS:



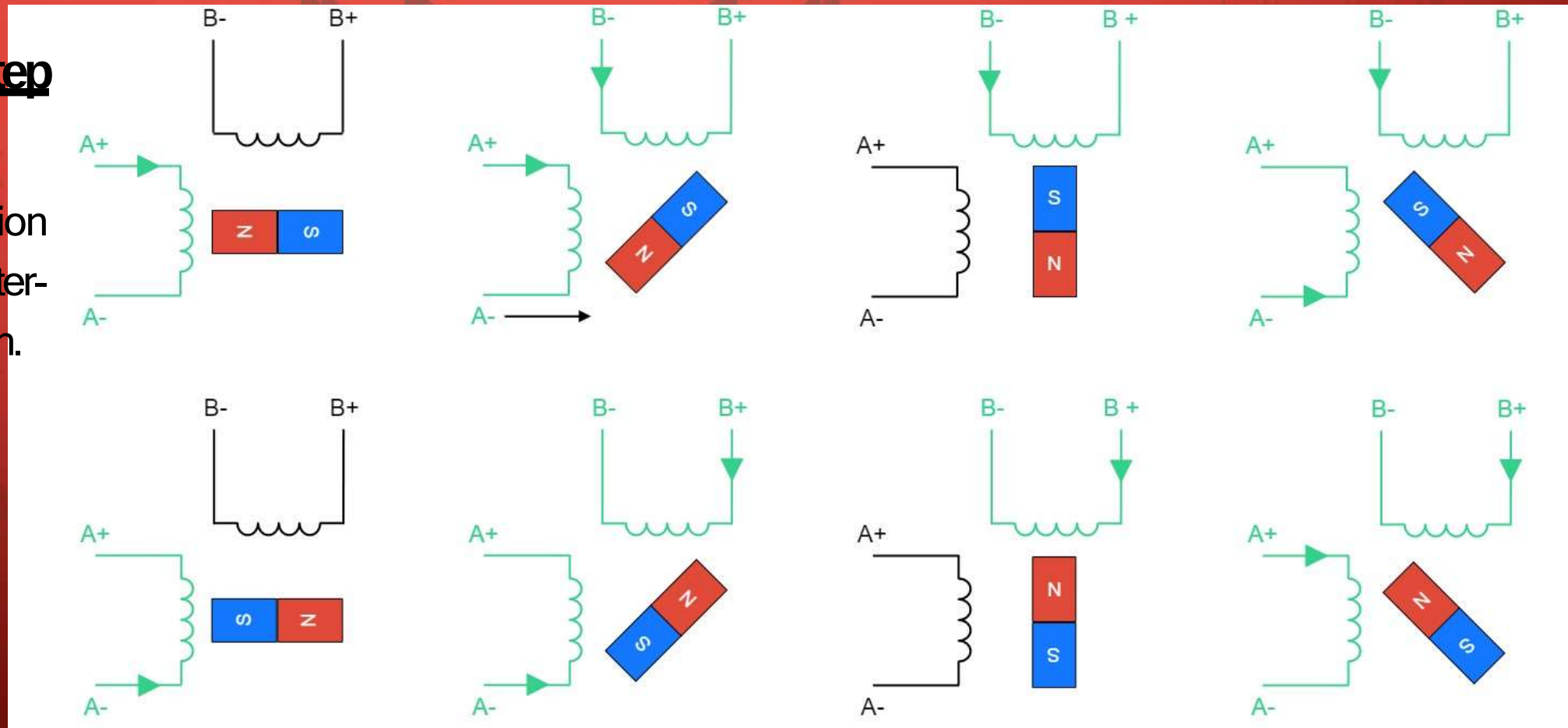






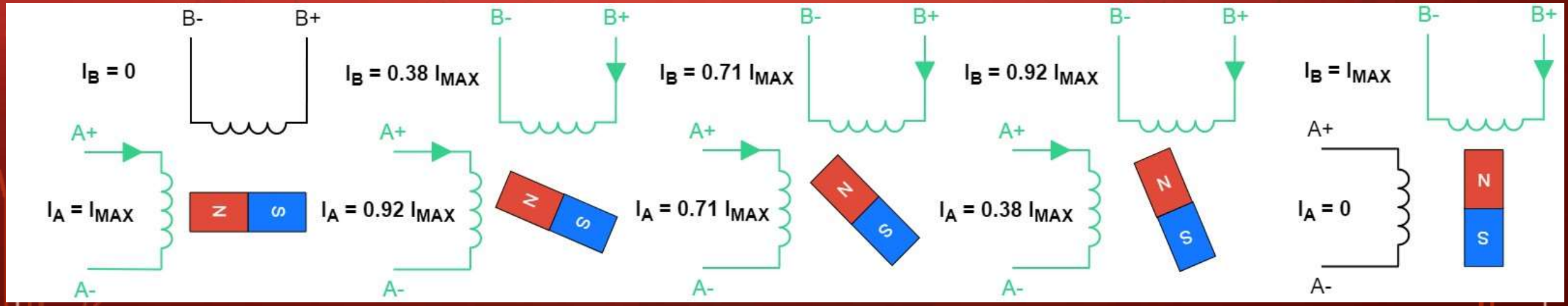
THEORY OF STEPPER MOTORS:

- **Two Coils Half Step Activation:**
  - This is the activation method for counter-clockwise rotation.



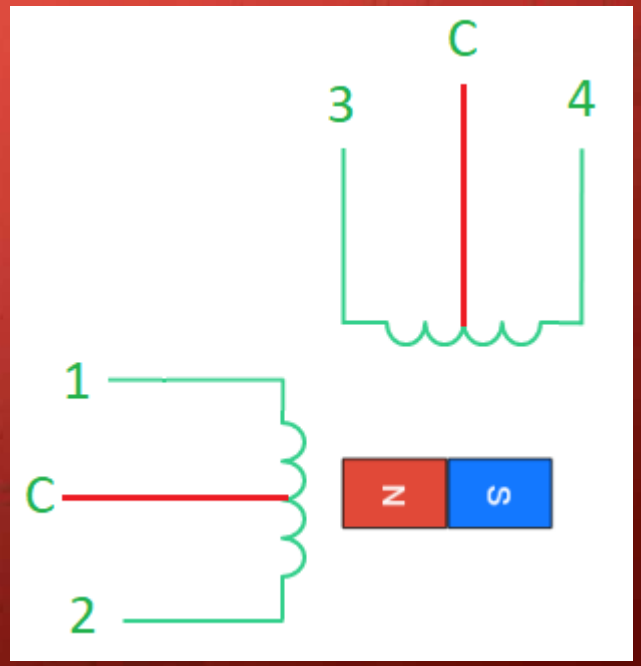
THEORY OF STEPPER MOTORS:

- **Two Coils, Micro Stepping:**
  - This is the activation method for counter-clockwise rotation.

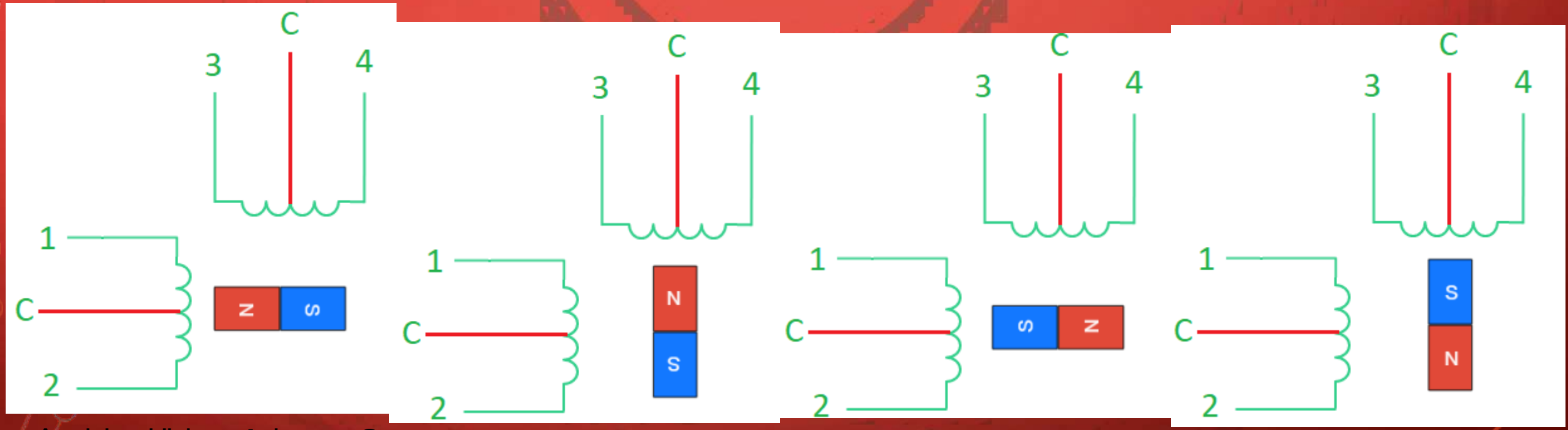


THEORY OF STEPPER MOTORS:

- **<u>Two Coils, Common Polar, Full Step Activation:</u>**
  - There are only two coil in this type of stepper motor, but each coil is divided from its middle point.
  - Because of division, Common pin polar exists, so the coil activation needs only one polar activation.
  - Now, we will apply clockwise rotation sequence on the coils as the following:



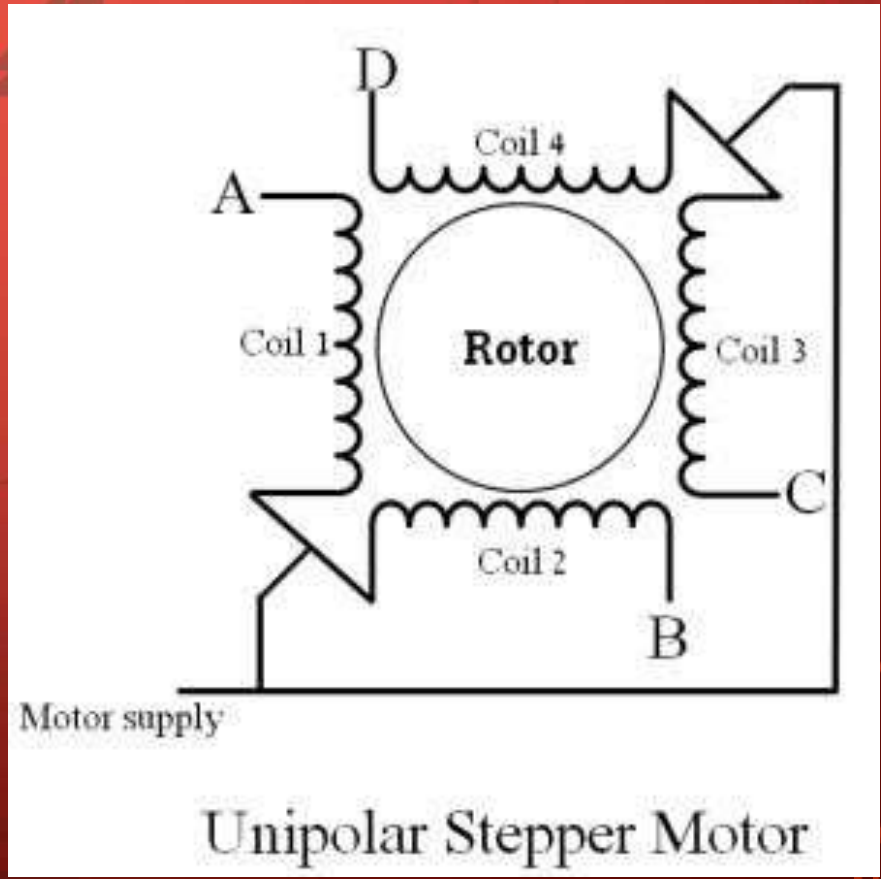
THEORY OF STEPPER MOTORS:



Applying High on **1,** Low on **Common.**
Applying High on **3,** Low on **Common.**
Applying High on **2,** Low on **Common.**
Applying High on **4,** Low on **Common.**

THEORY OF STEPPER MOTORS:

- **Two Coils, Common Polar, Full Step Activation:**
  - According the last activation method, you can notice that the common pin was activated every time by the same signal, So, this type of this motor is called **Unipolar activation.**
  - This motor type we will work on.



Unipolar Stepper Motor

THEORY OF STEPPER MOTORS:

- **Different between Unipolar and Bipolar:**
  - Assume that we has two identical motors with two coils, but one of them is unipolar and the other is bipolar

Criteria	Unipolar	Bipolar
Coil implementation	There are two coils with common pole	There are two coils without additional pole
Num of Control transistors	4	8
Torque	Weak	Double of unipolar
Configurability	Can be run as a Bipolar, also	Only runs as Bipolar

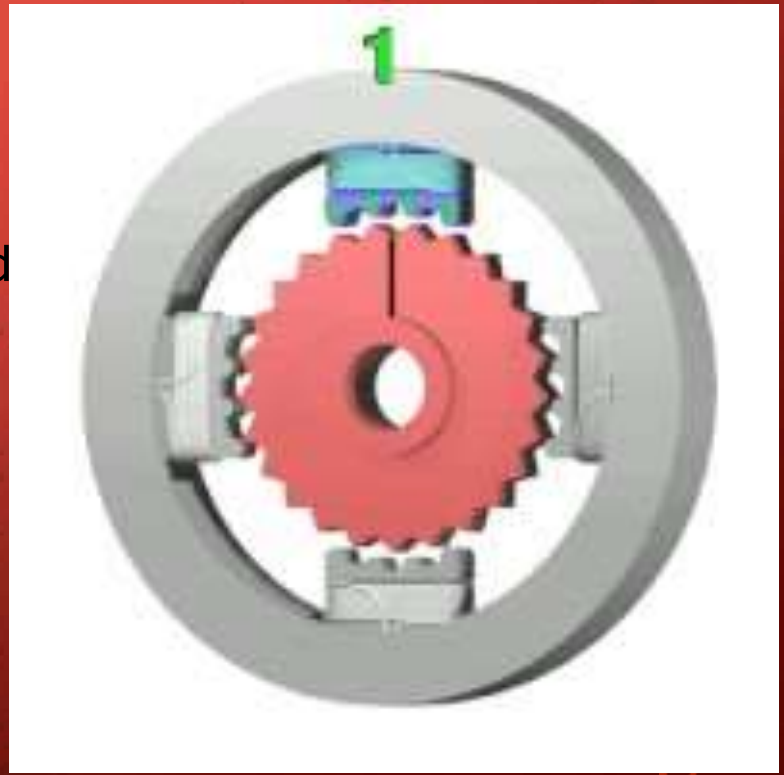
THEORY OF STEPPER MOTORS:

- **Internal Modification:**
  - As we mentioned before, we can decrease the angle of step by activation double coils or

    micro stepping, but now we will decrease the step by increasing the number of internal coils.
  - Unfortunately, increasing the number of coils will increase the number of wires outside the motor.
  - It will be difficult to control this motor because of huge quantity of wires.

THEORY OF STEPPER MOTORS:

- **Internal Modification:**
  - The last problem has been solved by that: the coils has 90 degrees between of them will have common wires outside the motor, like the following figure, all blue coils have the same wire outside the motor, and pink coil also and yellow and orange and so on.
  - So, after this modification, there are only four wires are outside of the motor.
  - If the blue coil is activated, all coils will generate electrical-magnetic field, and the nearest coil to the permanent magnet will attract the permanent magnet.

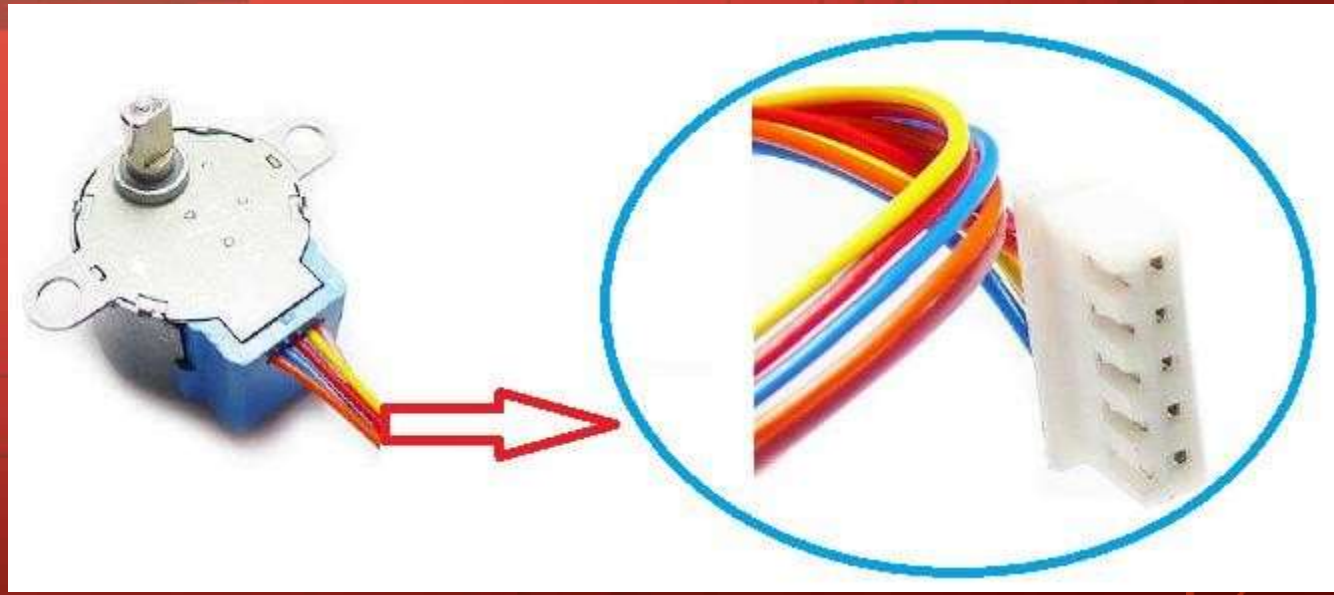


THEORY OF STEPPER MOTORS:

- **Internal Modification:**
  - The last problem has been solved by that: the coils has 90 degrees between of them will have common wires outside the motor, like the following figure, all blue coils have the same wire outside the motor, and pink coil also and yellow and orange and so on.
  - So, after this modification, there are only four wires are outside of the motor.
  - If the blue coil is activated, all coils will generate electrical-magnetic field, and the nearest coil to the permanent magnet will attract the permanent magnet.

OUR STEPPER MOTOR:

- We will use a small stepper motor that works at 5-Volt power, it is Two-coil-common-wire motor.
- This Stepper can be run as:
  - Unipolar.
  - Bipolar.
- Its name is 28BYJ-48.
- It has five wires outside the motor:
  - Blue.
  - Pink.
  - Yellow.
  - Orange.
  - Red.



OUR STEPPER MOTOR:

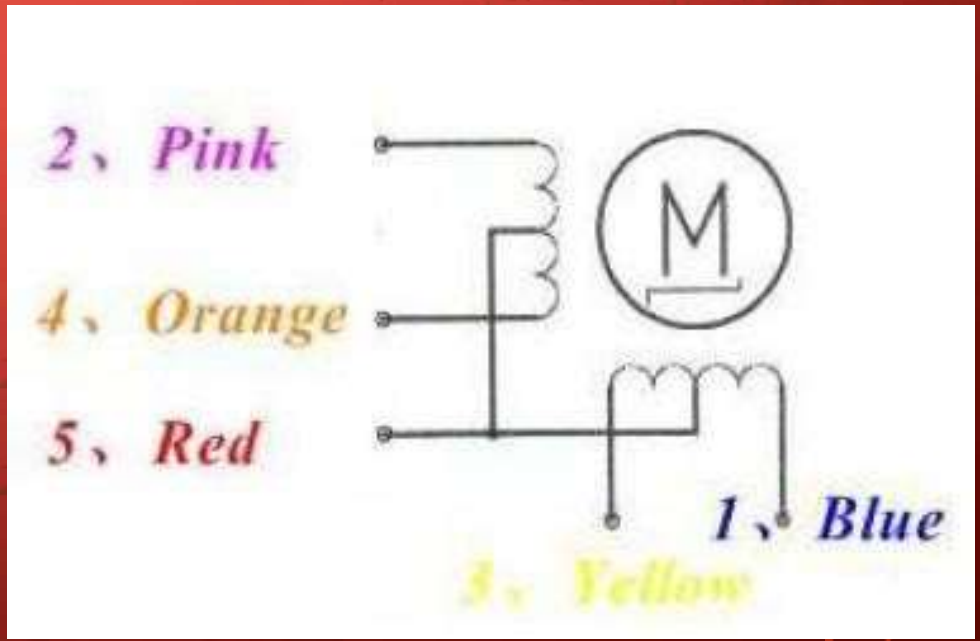
- Into the following figure, there is the datasheet of 28BYJ-48.
- Stride angle means the minimum angle per step(half step).
- Variation ratio means that there is a gearbox that has been implemented on the rotor shaft of motor, it is used to increase the torque and decrease the rotational speed.
- We can get the full step by multiplying the stride angle by Two (11.25/64).

Rated voltage :	5VDC
Number of Phase	4
Speed Variation Ratio	1/64
Stride Angle	5.625°/64
Frequency	100Hz
DC resistance	50Ω±7%(25℃)
Idle In-traction Frequency	> 600Hz
Idle Out-traction Frequency	> 1000Hz
In-traction Torque	>34.3mN.m(120Hz)
Self-positioning Torque	>34.3mN.m
Friction torque	600-1200 gf.cm
Pull in torque	300 gf.cm
Insulated resistance	>10MΩ(500V)
Insulated electricity power	600VAC/1mA/1s
Insulation grade	A
Rise in Temperature	<40K(120Hz)
Noise	<35dB(120Hz,No load,10cm)
Model	28BYJ-48 – 5V

OUR STEPPER MOTOR:

- It can be used as a bipolar (blue, yellow), (pink, orange).
- It can be used as a unipolar by this sequence activation.

(blue), (pink), (yellow), (orange).



FULL STEP SEQUENCE:

- These are the activation signals if the common wire is connected to ground(0 v).
- To rotate counterclockwise, start from last to beginning.

Full Step Mode Clockwise			
4 Orange	3 Yellow	2 Pink	1 Blue
0	0	0	1
0	0	1	0
0	1	0	0
1	0	0	0

HALF STEP SEQUENCE:

- These are the activation signals if the common wire is connected to ground(0 v).
- To rotate counterclockwise, start from last to beginning.

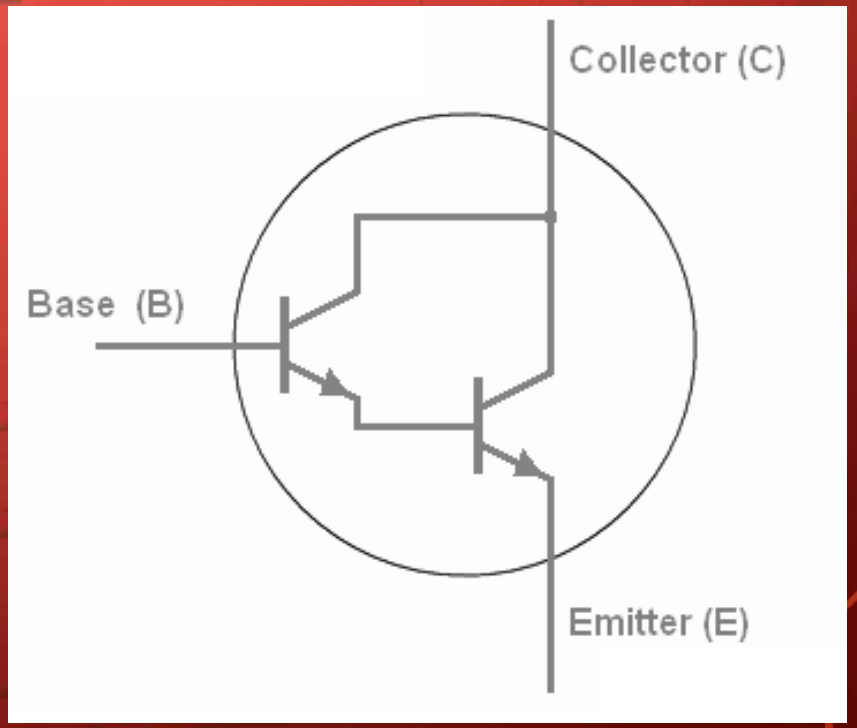
Half Step Mode Clockwise			
4 Orange	3 Yellow	2 Pink	1 Blue
1	0	0	1
0	0	0	1
0	0	1	1
0	0	1	0
0	1	1	0
0	1	0	0
1	1	0	0
1	0	0	0

ELECTRICAL SWITCH THAT WILL BE USED:

- Because the stepper is a coil-based component, so it will consume a huge power, that our micro can not be able to provide, and all coils generate a high Back EMF.
- Those problems can damage the micro controller, so we must use any electrical switch, the most common and widely used is transistors.
- The main component we will use is a Darlington Pair.

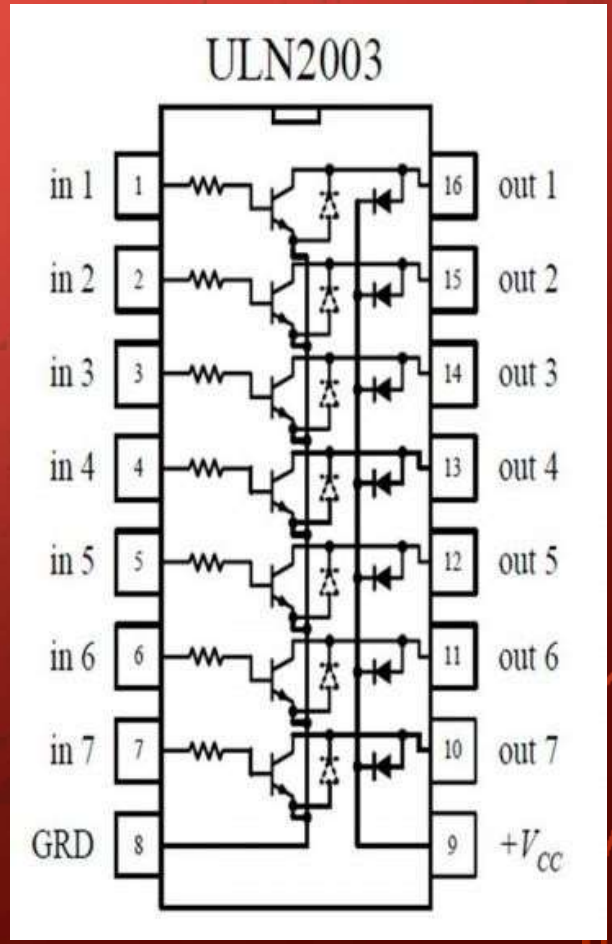
DARLINGTON PAIR:

- It is a multi-transistor configuration called the Darlington configuration (commonly called a Darlington pair) is a circuit consisting of two bipolar transistors with the emitter of one transistor connected to the base of the other, such that the current amplified by the first transistor is amplified further by the second one.[1] The collectors of both transistors are together connected. This configuration has a much higher current gain than each transistor taken separately. It acts like and is often packaged as a single transistor. It was invented in 1953 by Sidney Darlington.



ULN2003 IC WHICH IS BASED ON DARLINGTON PAIR:

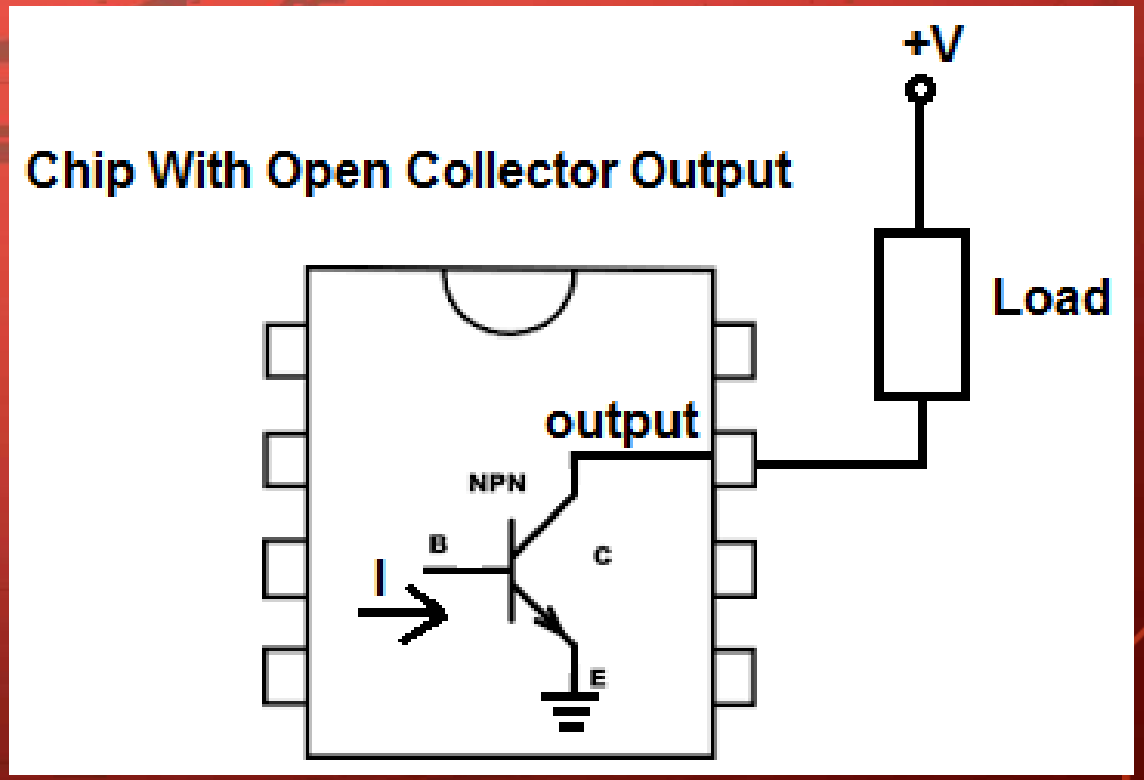
- It is an IC that has seven Darlington pair, every base pin of them is connected to in pins from In1 to in7.
- Every collector of Darlington pair is connected to out pins from out1 to out7.
- +Vcc pin16 is called free-wheeling diode which is connected to Vcc from sinking the current generated from Back EMF.
- GND pin8 is called common Emitter pin, all Emitter pins of Darlington Pair are assembled and connected to this pin.
- The last pin must be connected to GND because of the connection of this circuit is Open-Collector Circuit.



OPEN COLLECTOR CIRCUIT:

- The following figure shows that the load will be connected to the collector and the other side must be connected to Vcc.
- The common pin of our motor must be connected to Vcc.
- The

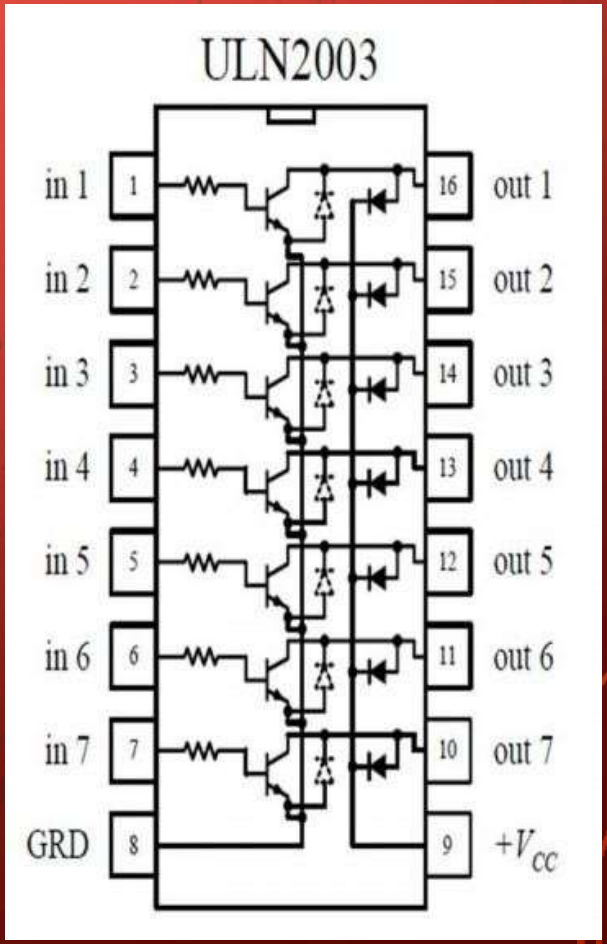
B	C
0	float
1	0



ULN2003 IC WHICH IS BASED ON DARLINGTON PAIR:

- To be the signals more logic, the ULN2003 added a NOT-Gate before the base of Darlington Pair, So the new truth table will be:

B	C
0	0
1	float



OUR STEPPER MOTOR:

- In the following figure, there are a PCB circuit which has been prepared for control our stepper motor.
- Now, the kind of activation signal is LOW because of UNL2003 Circuit as we mentioned before.
- Th

Blue	Pink	Yellow	Orange
0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	0

OUR STEPPER MOTOR:

- As we mentioned before, the full step of our motor is (**11.25/64**), so we will calculate how many steps our motor must move to complete one cycle?
  - Steps in a completed Cycle $= \frac{full\ rotation\ angle}{one\ step\ angle}$

$$= \frac{360}{\left(\frac{11.25}{64}\right)} = 2048\ steps.$$

  - Steps in a specific angle $= \frac{Angle\ X\ steps\ into\ a\ completed\ cycle}{full\ rotation\ angle}$

$$= \frac{Angle\ X\ 2048}{360}$$

THANK YOU!

AMIT