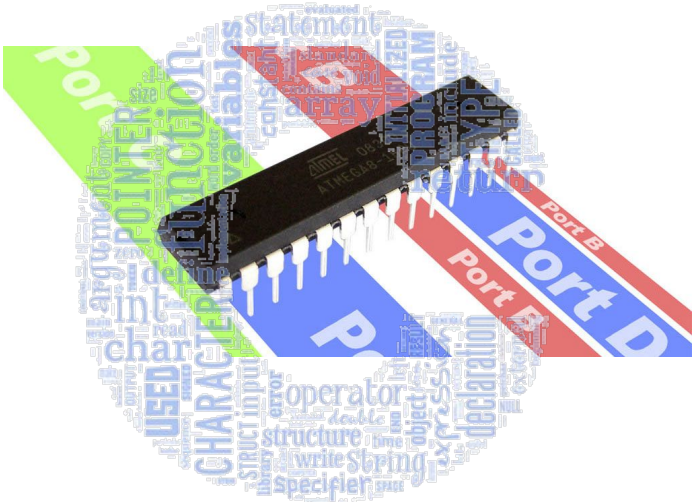
C Diline Giris



Suhap SAHIN

Veri Tipleri

Veri Tipi	Byte(8 bit)
char	8
byte	8
short	16
int	32
long	64
float	32
double	64
boolean	1

Veri Tipleri

Veri Tipi	Byte(8 bit)
char	8
byte	8
short	16
int	32
long	64
float	32
double	64
boolean	1

Veri Tipleri

Veri Tipi	Byte(8 bit)
char	8
byte	8
short	16
int	32
long	64
float	32
double	64
boolean	1

Veri Tipleri

Veri Tipi	Byte(8 bit)
char	8
byte	8
short	16
int	32
long	64
float	32
double	64
boolean	1

Veri Tipleri

Veri Tipi	Byte(8 bit)
char	8
byte	8
short	16
int	32
long	64
float	32
double	64
boolean	1

Veri Tipleri

Veri Tipi	Byte(8 bit)
char	8
byte	8
short	16
int	32
long	64
float	32
double	64
boolean	1

Veri Tipleri

Veri Tipi	Byte(8 bit)
char	8
byte	8
short	16
int	32
long	64
float	32
double	64
boolean	1

Veri Tipleri

Veri Tipi	Byte(8 bit)
char	8
byte	8
short	16
int	32
long	64
float	32
double	64
boolean	1

Bit Operatörleri

Operator	islem
~	1'tümleyen
>>	Saga Kaydırma
<<	Sola Kaydırma
&	Bit düzeyinde AND
|	Bit düzeyinde OR
^	Bit düzeyinde XOR

Bit Operatörleri

Operator	islem
~	1'tümleyen
>>	Saga Kaydırma
<<	Sola Kaydırma
&	Bit düzeyinde AND
|	Bit düzeyinde OR
^	Bit düzeyinde XOR

Bit Operatörleri

Operator	islem
~	1'tümleyen
>>	Saga Kaydırma
<<	Sola Kaydırma
&	Bit düzeyinde AND
|	Bit düzeyinde OR
^	Bit düzeyinde XOR

Bit Operatörleri

Operator	islem
~	1'tümleyen
>>	Saga Kaydırma
<<	Sola Kaydırma
&	Bit düzeyinde AND
|	Bit düzeyinde OR
^	Bit düzeyinde XOR

Bit Operatörleri

Operator	islem
~	1'tümleyen
>>	Saga Kaydırma
<<	Sola Kaydırma
&	Bit düzeyinde AND
|	Bit düzeyinde OR
^	Bit düzeyinde XOR

Bit Operatörleri

Operator	islem
~	1'tümleyen
>>	Saga Kaydırma
<<	Sola Kaydırma
&	Bit düzeyinde AND
|	Bit düzeyinde OR
^	Bit düzeyinde XOR

Bit Operatörleri

Giris Bitler		AND	OR	XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Bit Operatörleri

Giris Bitler		AND	OR	XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Bit Operatörleri

Giris Bitler		AND	OR	XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Bit Operatörleri

Giris Bitler		AND	OR	XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Bit Operatörleri

Operatör	Kullanımı
~	Bir biti terslemek
&	Saklayıcının belirli kısmını maskelemek
<<	Bitleri sola kaydırmak
>>	Bitleri sağa kaydırmak

Bit Operatörleri

Operatör	Kullanımı
~	Bir biti terslemek
&	Saklayıcının belirli kısmını maskelemek
<<	Bitleri sola kaydırmak
>>	Bitleri sağa kaydırmak

Bit Operatörleri

Operatör	Kullanımı
~	Bir biti terslemek
&	Saklayıcının belirli kısmını maskelemek
<<	Bitleri sola kaydırmak
>>	Bitleri sağa kaydırmak

Bit Operatörleri

Operatör	Kullanımı
~	Bir biti terslemek
&	Saklayıcının belirli kısmını maskelemek
<<	Bitleri sola kaydırmak
>>	Bitleri saga kaydırmak

AND

```c
#include<stdio.h>
int main()
{
        int a=12;
        int b=10;



        return(0);
}
```

$a = 12 = (0000\ 0000\ 0000\ 1100)_2$

$b = 10 = (0000\ 0000\ 0000\ 1010)_2$

AND

```c
#include<stdio.h>
int main()
{
        int a=12;
        int b=10;
        printf("\nNumber1 AND Number2 : %d",a & b);


        return(0);
}
```

$a = 12 = (0000\ 0000\ 0000\ 1100)_2$

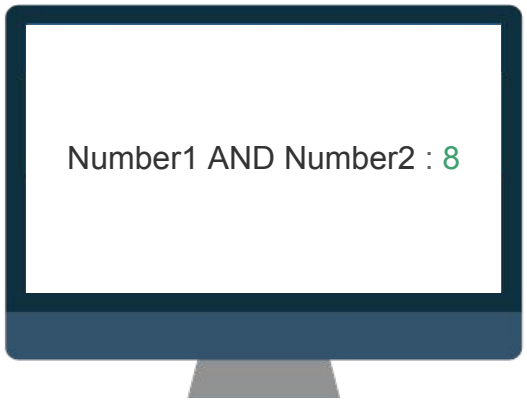
$b = 10 = (0000\ 0000\ 0000\ 1010)_2$

a = 0000 0000 0000 1100

b = 0000 0000 0000 1010

AND

```c
#include<stdio.h>
int main()
{
        int a=12;
        int b=10;
        printf("\nNumber1 AND Number2 : %d",a & b);


        return(0);
}
```

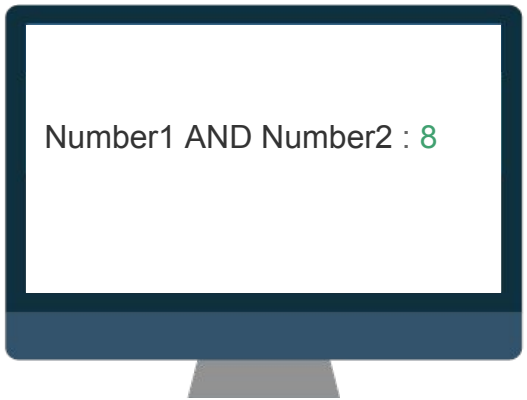


$a = 12 = (0000\ 0000\ 0000\ 1100)_2$
$b = 10 = (0000\ 0000\ 0000\ 1010)_2$

```
a =        0000 0000 0000 1100
b =        0000 0000 0000 1010
      ---------------------------------
a & b      0000 0000 0000 1000
```

OR

```c
#include<stdio.h>
int main()
{
        int a=12;
        int b=10;
        printf("\nNumber1 AND Number2 : %d",a & b);
        printf("\nNumber1  OR  Number2 : %d",a | b);

        return(0);
}
```



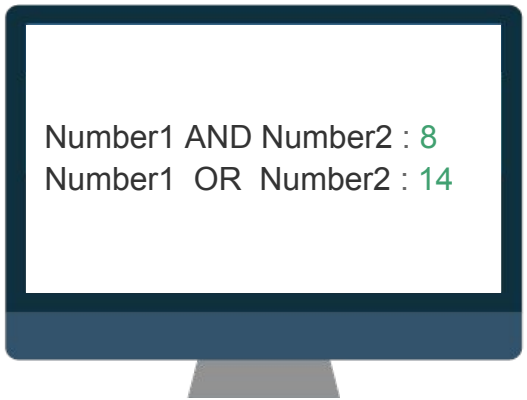
$a = 12 = (0000\ 0000\ 0000\ 1100)_2$

$b = 10 = (0000\ 0000\ 0000\ 1010)_2$

```
a =        0000 0000 0000 1100
b =        0000 0000 0000 1010
      ---------------------------------
a | b      0000 0000 0000 1110
```

OR

```c
#include<stdio.h>
int main()
{
        int a=12;
        int b=10;
        printf("\nNumber1 AND Number2 : %d",a & b);
        printf("\nNumber1  OR  Number2 : %d",a | b);

        return(0);
}
```

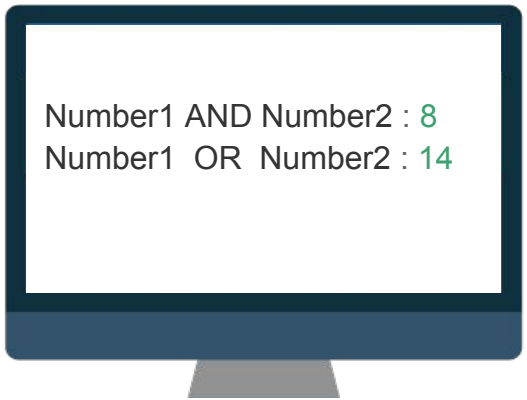


$a = 12 = (0000\ 0000\ 0000\ 1100)_2$
$b = 10 = (0000\ 0000\ 0000\ 1010)_2$

```
a =        0000 0000 0000 1100
b =        0000 0000 0000 1010
      ---------------------------------
a | b      0000 0000 0000 1110
```

XOR

```c
#include<stdio.h>
int main()
{
        int a=12;
        int b=10;
        printf("\nNumber1 AND Number2 : %d",a & b);
        printf("\nNumber1  OR  Number2 : %d",a | b);
        printf("\nNumber  XOR Number2 : %d",a ^ b);
        return(0);
}
```



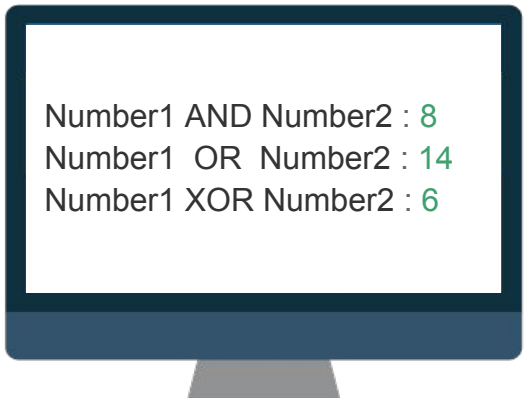
$a = 12 = (0000\ 0000\ 0000\ 1100)_2$

$b = 10 = (0000\ 0000\ 0000\ 1010)_2$

```
a =        0000 0000 0000 1100
b =        0000 0000 0000 1010
      ---------------------------------
a ^ b      0000 0000 0000 0110
```

XOR

```c
#include<stdio.h>
int main()
{
        int a=12;
        int b=10;
        printf("\nNumber1 AND Number2 : %d",a & b);
        printf("\nNumber1  OR  Number2 : %d",a | b);
        printf("\nNumber  XOR Number2 : %d",a ^ b);
        return(0);
}
```



$a = 12 = (0000\ 0000\ 0000\ 1100)_2$
$b = 10 = (0000\ 0000\ 0000\ 1010)_2$

a = 0000 0000 0000 1100
b = 0000 0000 0000 1010
---
a ^ b 0000 0000 0000 0110

Saga Bit Kaydırma (>>)

Verinin hareket yönü

Sag ========>>>>>

Saga Bit Kaydırma (>>)

Verinin hareket yönü

Sag ========>>>>>

Orjinal Sayı A

0000 0000 0011 1100

Saga Bit Kaydırma (>>)

Verinin hareket yönü	Sag ========>>>>>
Orjinal Sayı A	0000 0000 0011 1100
1 bit kaydırma	0000 0000 0011 110

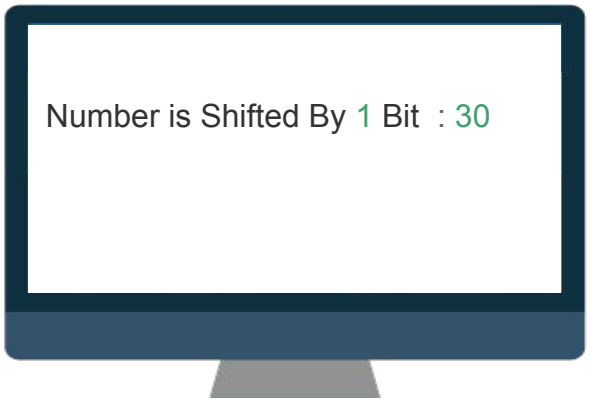
Saga Bit Kaydırma (>>)

```
#include<stdio.h>
int main()
{
        int a = 60;



        return(0);
}
```

a 0000 0000 0011 1100

Saga Bit Kaydırma (>>)

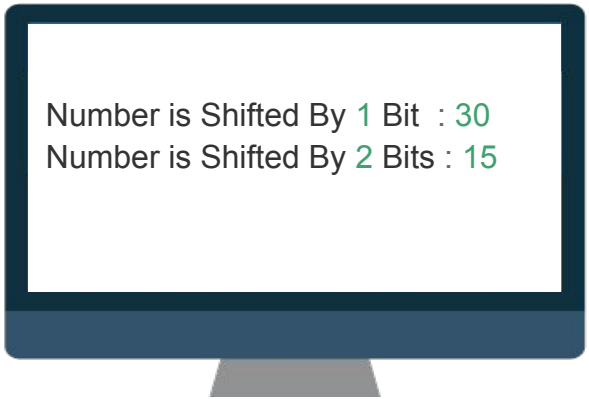
```c
#include<stdio.h>
int main()
{
        int a = 60;
        printf("\nNumber is Shifted By 1 Bit  : %d",a >> 1);


        return(0);
}
```



a	0000 0000 0011 1100
a >> 1	0000 0000 0001 1110

Saga Bit Kaydırma (>>)

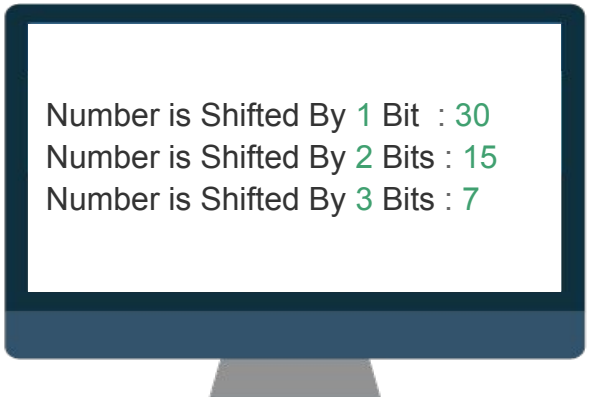
```c
#include<stdio.h>
int main()
{
        int a = 60;
        printf("\nNumber is Shifted By 1 Bit  : %d",a >> 1);
        printf("\nNumber is Shifted By 2 Bits : %d",a >> 2);


        return(0);
}
```



a	0000 0000 0011 1100
a >> 1	0000 0000 0001 1110
a >> 2	0000 0000 0000 1111

Saga Bit Kaydırma (>>)

```c
#include<stdio.h>
int main()
{
	int a = 60;
	printf("\nNumber is Shifted By 1 Bit  : %d",a >> 1);
	printf("\nNumber is Shifted By 2 Bits : %d",a >> 2);
	printf("\nNumber is Shifted By 3 Bits : %d",a >> 3);

	return(0);
}
```



a	0000 0000 0011 1100
a >> 1	0000 0000 0001 1110
a >> 2	0000 0000 0000 1111
a >> 3	0000 0000 0000 0111

Sola Bit Kaydırma (<<)

Verinin hareket yönü

Sol <<<<<=========

Sola Bit Kaydırma (<<)

Verinin hareket yönü

Sol <<<<<=========

Orjinal Sayı A

0000 0000 0011 1100

Sola Bit Kaydırma (<<)

Verinin hareket yönü	Sol <<<<<==========
Orjinal Sayı A	0000 0000 0011 1100
1 bit kaydırma	0000 0000 0111 1000

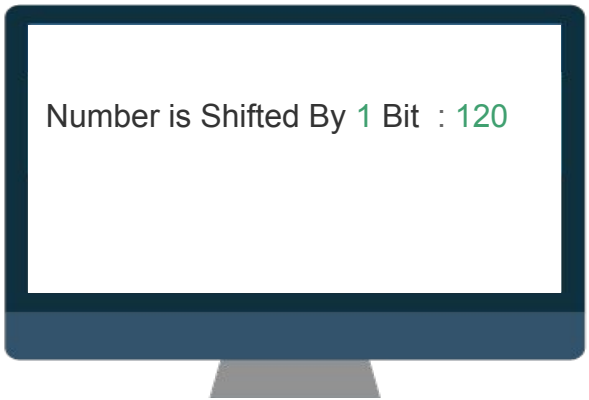
Sola Bit Kaydırma (<<)

```c
#include<stdio.h>
int main()
{
        int a = 60;



        return(0);
}
```

a 0000 0000 0011 1100

Sola Bit Kaydırma (<<)

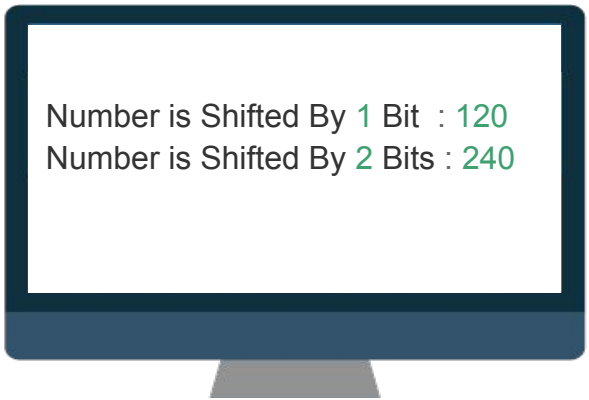
```c
#include<stdio.h>
int main()
{
        int a = 60;
        printf("\nNumber is Shifted By 1 Bit  : %d",a << 1);


        return(0);
}
```



a	0000 0000 0011 1100
a << 1	0000 0000 0111 1000

Sola Bit Kaydırma (<<)

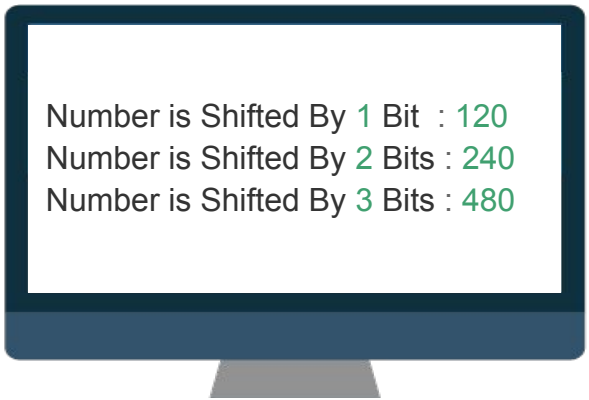
```c
#include<stdio.h>
int main()
{
        int a = 60;
        printf("\nNumber is Shifted By 1 Bit  : %d",a << 1);
        printf("\nNumber is Shifted By 2 Bits : %d",a << 2);


        return(0);
}
```



a	0000 0000 0011 1100
a << 1	0000 0000 0111 1000
a << 2	0000 0000 1111 0000

Sola Bit Kaydırma (<<)

```c
#include<stdio.h>
int main()
{
        int a = 60;
        printf("\nNumber is Shifted By 1 Bit  : %d",a << 1);
        printf("\nNumber is Shifted By 2 Bits : %d",a << 2);
        printf("\nNumber is Shifted By 3 Bits : %d",a << 3);

        return(0);
}
```



a	0000 0000 0011 1100
a << 1	0000 0000 0111 1000
a << 2	0000 0000 1111 0000
a << 3	0000 0001 1110 0000

Bit Maskeleme

Gerekli verilerin tutulup ve kalanının maskelenmesi (bloke edildigi)

En Sık Kullanılan Operatör: AND (&)

Bit Maskeleme

Gerekli verilerin tutulup ve kalanının maskelenmesi (bloke edildigi)

En Sık Kullanılan Operatör: AND (&)

0000 0000 100 1 01 01 (veri)

Bit Maskeleme

Gerekli verilerin tutulup ve kalanının maskelenmesi (bloke edildigi)

En Sık Kullanılan Operatör: AND (&)

0000 0000 100|1 01|01 (veri)

Bit Maskeleme

Gerekli verilerin tutulup ve kalanının maskelenmesi (bloke edildigi)

En Sık Kullanılan Operatör: AND (&)

	0000 0000 100|1 01|01	(veri)
AND	0000 0000 000 111 00	(maske)

Bit Maskeleme

Gerekli verilerin tutulup ve kalanının maskelenmesi (bloke edildigi)

En Sık Kullanılan Operatör: AND (&)

```
      0000 0000 100|1 01|01     (veri)
AND   0000 0000 000|1 11|00     (maske)
      ---------------------
```

Bit Maskeleme

Gerekli verilerin tutulup ve kalanının maskelenmesi (bloke edildigi)

En Sık Kullanılan Operatör: AND (&)

	0000 0000 100|1 01|01	(veri)
AND	0000 0000 000|1 11|00	(maske)
	0000 0000 000|1 01|00	(maskelenmis veri)

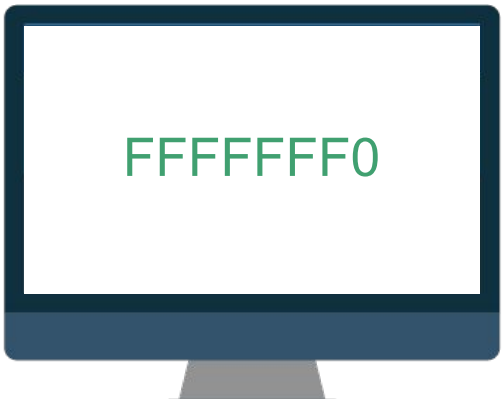
Bit Kaydırma (Negatif Sayılar)

```
#include<stdio.h>
void main()
{
        printf("%x",-1<<4);
}
```

-1	FFFFFFFF	1111 1111 1111 1111 1111 1111 1111 1111
-1 << 1	FFFFFFFE	1111 1111 1111 1111 1111 1111 1111 1110
-1 << 2	FFFFFFFD	1111 1111 1111 1111 1111 1111 1111 1100
-1 << 3	FFFFFFF8	1111 1111 1111 1111 1111 1111 1111 1000
-1 << 4	FFFFFFF0	1111 1111 1111 1111 1111 1111 1111 0000

Bit Kaydırma (Negatif Sayılar)

```
#include<stdio.h>
void main()
{
        printf("%x",-1<<4);
}
```



-1	FFFFFFFF	1111 1111 1111 1111 1111 1111 1111 1111
-1 << 1	FFFFFFFE	1111 1111 1111 1111 1111 1111 1111 1110
-1 << 2	FFFFFFFD	1111 1111 1111 1111 1111 1111 1111 1100
-1 << 3	FFFFFFF8	1111 1111 1111 1111 1111 1111 1111 1000
-1 << 4	FFFFFFF0	1111 1111 1111 1111 1111 1111 1111 0000

Bit Kaydırma (Negatif Sayılar)

```c
#include<stdio.h>
int main()
{
        int a = -60;

        printf("\nNegative Right Shift by 1 Bit : %",a >> 1);



        return(0);
}
```

Bit Kaydırma (Negatif Sayılar)

```c
#include<stdio.h>
int main()
{
        int a = -60;

        printf("\nNegative Right Shift by 1 Bit : %",a >> 1);
        printf("\nNegative Right Shift by 2 Bits : %d",a >> 2);
        printf("\nNegative Right Shift by 3 Bits : %d",a >> 3);

        return(0);
}
```

printf içinde degisken kaydırma

```
#include<stdio.h>
main()
{
        int i;
        for(i=0;i<5;i++)

}
```

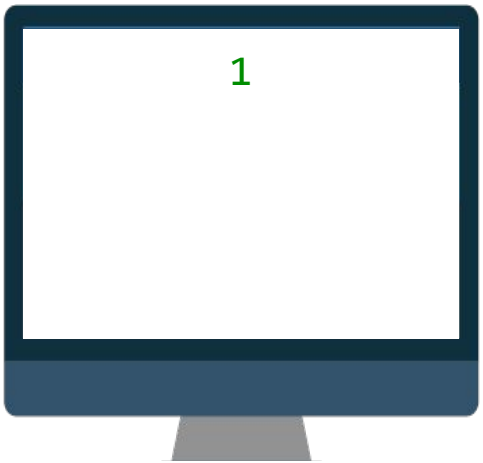
printf içinde degisken kaydırma

```c
#include<stdio.h>
main()
{
	int i;
	for(i=0;i<5;i++)
		printf("%d\n", 1 << i);
}
```

printf içinde degisken kaydırma

```c
#include<stdio.h>
main()
{
	int i;
	for(i=0;i<5;i++)
		printf("%d\n", 1 << i);
}
```

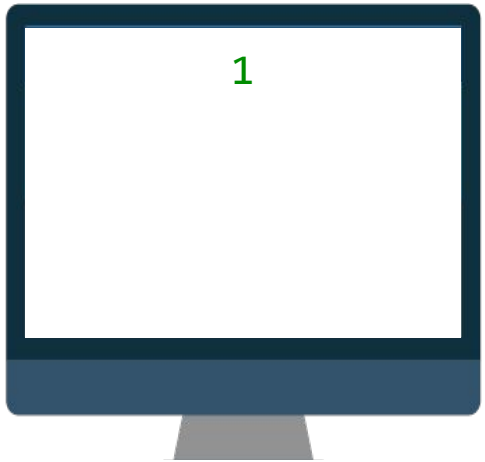
1 00000001 0000 0000 0000 0000 0000 0000 0000 0001



printf içinde degisken kaydırma

```c
#include<stdio.h>
main()
{
        int i;
        for(i=0;i<5;i++)
                printf("%d\n", 1 << i);
}
```

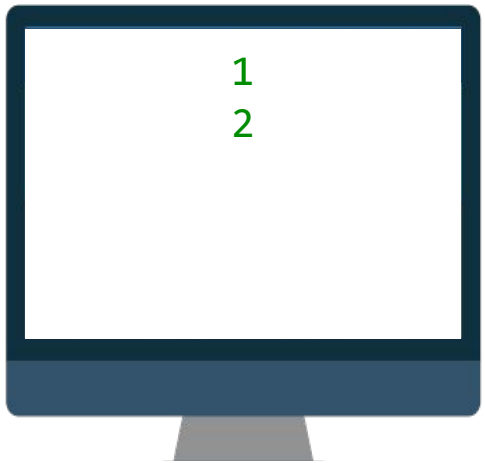
1	00000001	0000 0000 0000 0000 0000 0000 0000 0001
1 << 0	00000001	0000 0000 0000 0000 0000 0000 0000 0001



printf içinde degisken kaydırma

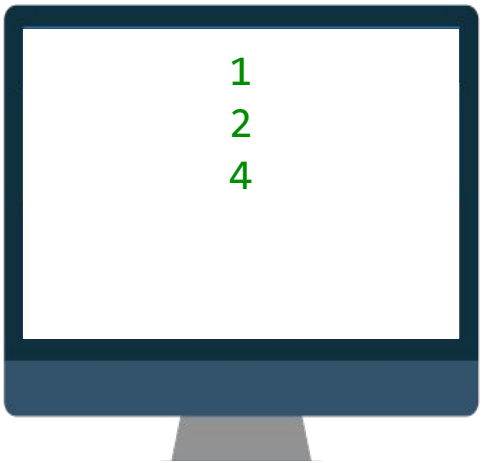
```c
#include<stdio.h>
main()
{
	int i;
	for(i=0;i<5;i++)
		printf("%d\n", 1 << i);
}
```

1	00000001	0000 0000 0000 0000 0000 0000 0000 0001
1 << 0	00000001	0000 0000 0000 0000 0000 0000 0000 0001
1 << 1	00000002	0000 0000 0000 0000 0000 0000 0000 0010



printf içinde degisken kaydırma

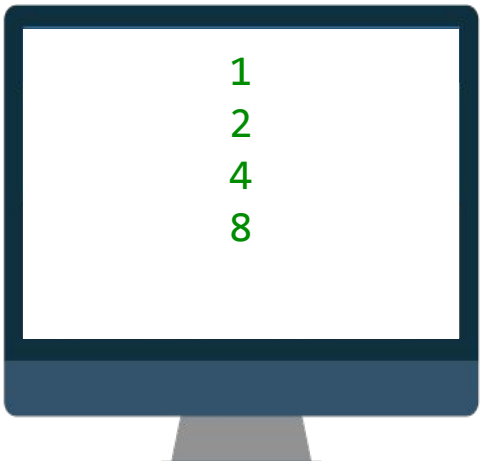
```c
#include<stdio.h>
main()
{
        int i;
        for(i=0;i<5;i++)
                printf("%d\n", 1 << i);
}
```



1	00000001	0000 0000 0000 0000 0000 0000 0000 0001
1 << 0	00000001	0000 0000 0000 0000 0000 0000 0000 0001
1 << 1	00000002	0000 0000 0000 0000 0000 0000 0000 0010
1 << 2	00000004	0000 0000 0000 0000 0000 0000 0000 0100

printf içinde degisken kaydırma

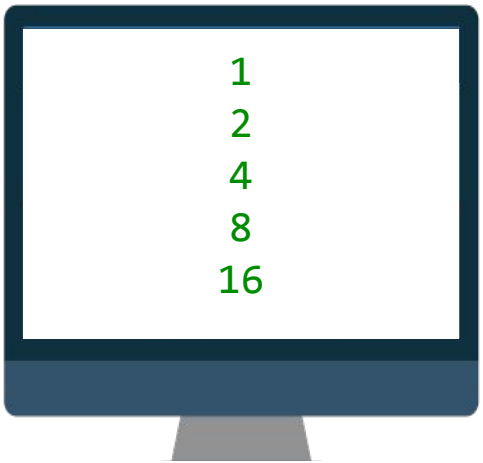
```c
#include<stdio.h>
main()
{
        int i;
        for(i=0;i<5;i++)
                printf("%d\n", 1 << i);
}
```



1	00000001	0000 0000 0000 0000 0000 0000 0000 0001
1 << 0	00000001	0000 0000 0000 0000 0000 0000 0000 0001
1 << 1	00000002	0000 0000 0000 0000 0000 0000 0000 0010
1 << 2	00000004	0000 0000 0000 0000 0000 0000 0000 0100
1 << 3	00000008	0000 0000 0000 0000 0000 0000 0000 1000

printf içinde degisken kaydırma

```c
#include<stdio.h>
main()
{
        int i;
        for(i=0;i<5;i++)
                printf("%d\n", 1 << i);
}
```



1	00000001	0000 0000 0000 0000 0000 0000 0000 0001
1 << 0	00000001	0000 0000 0000 0000 0000 0000 0000 0001
1 << 1	00000002	0000 0000 0000 0000 0000 0000 0000 0010
1 << 2	00000004	0000 0000 0000 0000 0000 0000 0000 0100
1 << 3	00000008	0000 0000 0000 0000 0000 0000 0000 1000
1 << 4	00000010	0000 0000 0000 0000 0000 0000 0001 0000

1'e tümleme

Orjinal Sayı A 0000 0000 0011 1100

1'e tümleme

Orjinal Sayı A	0000	0000	0011	1100
1'e Tümlenmis A	1111	1111	1100	0011

1'e tümleme

```
#include<stdio.h>
int main()
{
        int a=10;


}
```

a 0000 0000 0000 1010

1'e tümleme

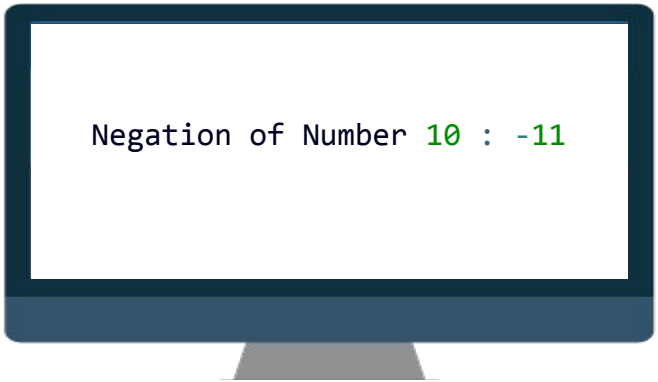
```c
#include<stdio.h>
int main()
{
        int a=10;
        printf("\nNegation of Number 10 : %d",~a);
        return(0);
}
```

a	0000 0000 0000 1010
~a	1111 1111 1111 0101

1'e tümleme

```c
#include<stdio.h>
int main()
{
        int a=10;
        printf("\nNegation of Number 10 : %d",~a);
        return(0);
}
```

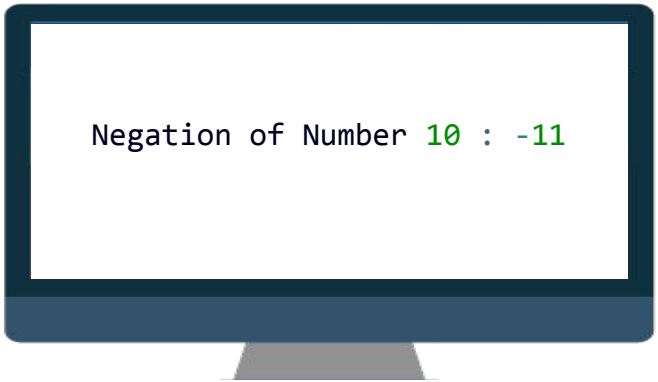
a	0000 0000 0000 1010
~a	1111 1111 1111 0101



1'e tümleme

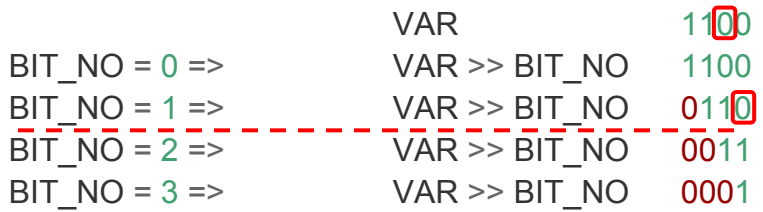
```c
#include<stdio.h>
int main()
{
        int a=10;
        printf("\nNegation of Number 10 : %d",~a);
        return(0);
}
```

a	0000 0000 0000 1010
~a	1111 1111 1111 0101



Örnek

```
#define BIT_GET(VAR, BIT_NO) ( VAR >> BIT_NO        )
```



Örnek

```
#define BIT_GET(VAR, BIT_NO) ((VAR >> BIT_NO) & 1)
```

```
      VAR   =    0110
&     1     =    0001
-----------------------------------
                 0000
```

```
                          VAR                1100
BIT_NO = 0 =>             VAR >> BIT_NO      1100
BIT_NO = 1 =>             VAR >> BIT_NO      0110
BIT_NO = 2 =>             VAR >> BIT_NO      0011
BIT_NO = 3 =>             VAR >> BIT_NO      0001
```

Örnek

```
#define BIT_GET(VAR, BIT_NO) ((VAR >> BIT_NO) & 1)
```

	VAR	1100
BIT_NO = 0 =>	VAR >> BIT_NO	1100
BIT_NO = 1 =>	VAR >> BIT_NO	0110
BIT_NO = 2 =>	VAR >> BIT_NO	0011
BIT_NO = 3 =>	VAR >> BIT_NO	0001

Örnek

```
#define BIT_GET(VAR, BIT_NO) ((VAR >> BIT_NO) & 1)
```

```
        VAR   =    0011
&       1     =    0001
-------------------------------------
                   0001
```

```
                                VAR                 1100
BIT_NO = 0 =>                   VAR >> BIT_NO       1100
BIT_NO = 1 =>                   VAR >> BIT_NO       0110
BIT_NO = 2 =>                   VAR >> BIT_NO       0011
BIT_NO = 3 =>                   VAR >> BIT_NO       0001
```

Örnek

```
#define BIT_SET(VAR, BIT_NO) do {
                    1<<BIT_NO;
} while (0)
```

	1	0001
BIT_NO = 0 =>	1 << BIT_NO	0001
BIT_NO = 1 =>	1 << BIT_NO	0010
BIT_NO = 2 =>	1 << BIT_NO	0100
BIT_NO = 3 =>	1 << BIT_NO	1000

Örnek

```
#define BIT_SET(VAR, BIT_NO) do {
        VAR |= 1<<BIT_NO;
} while (0)
```

```
        VAR             =       0101
|       1 << BIT_NO      =       0010
-------------------------------------------------
        VAR             =       0111
```

```
                                1               0001
BIT_NO = 0 =>                   1 << BIT_NO     0001
BIT_NO = 1 =>                   1 << BIT_NO     0010
BIT_NO = 2 =>                   1 << BIT_NO     0100
BIT_NO = 3 =>                   1 << BIT_NO     1000
```

Örnek

```
#define BIT_RESET(VAR, BIT_NO) do {
                        1<<BIT_NO ;
} while (0)
```

	1	0001
BIT_NO = 0 =>	1 << BIT_NO	0001
BIT_NO = 1 =>	1 << BIT_NO	0010
BIT_NO = 2 =>	1 << BIT_NO	0100
BIT_NO = 3 =>	1 << BIT_NO	1000

Örnek

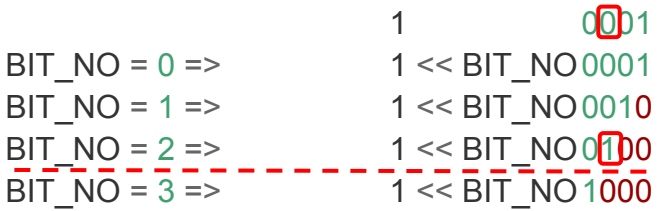
```
#define BIT_RESET(VAR, BIT_NO) do {
        VAR &= ~(1<<BIT_NO);
} while (0)
```

```
        VAR          =    0101
&   ~(1 << BIT_NO) =    1011
-------------------------------------------
        VAR          =    0001
```

```
                                    1              0001
BIT_NO = 0 =>                       1 << BIT_NO    0001
BIT_NO = 1 =>                       1 << BIT_NO    0010
BIT_NO = 2 =>                       1 << BIT_NO    0100
BIT_NO = 3 =>                       1 << BIT_NO    1000
```

Örnek

```
#define BIT_TOGGLE(VAR, BIT_NO) do {
                    1<<BIT_NO ;
} while (0)
```



Örnek

```
#define BIT_TOGGLE(VAR, BIT_NO) do {
        VAR ^= (1<<BIT_NO);
} while (0)
```

```
        VAR          =    0101
^   1 << BIT_NO      =    0100
-------------------------------------------
        VAR          =    0001
```

```
                                  1              0001
BIT_NO = 0 =>                     1 << BIT_NO    0001
BIT_NO = 1 =>                     1 << BIT_NO    0010
BIT_NO = 2 =>                     1 << BIT_NO    0100
BIT_NO = 3 =>                     1 << BIT_NO    1000
```

Örnek

```
#define BIT_SET(VAR, BIT_NO) do {
    VAR |= 1<<BIT_NO;
} while (0)
```

Örnek

```
#define BIT_SET(VAR, BIT_NO) do {
    VAR |= 1<<BIT_NO;
} while (0)

#define BIT_RESET(VAR, BIT_NO) do {
    VAR &= ~(1<<BIT_NO);
} while (0)
```

Örnek

```
#define BIT_SET(VAR, BIT_NO) do {
    VAR |= 1<<BIT_NO;
} while (0)

#define BIT_RESET(VAR, BIT_NO) do {
    VAR &= ~(1<<BIT_NO);
} while (0)

#define BIT_TOGGLE(VAR, BIT_NO) do {
    VAR ^= (1<<BIT_NO);
} while (0)
```

Örnek

```
#define BIT_SET(VAR, BIT_NO) do {
    VAR |= 1<<BIT_NO;
} while (0)

#define BIT_RESET(VAR, BIT_NO) do {
    VAR &= ~(1<<BIT_NO);
} while (0)

#define BIT_TOGGLE(VAR, BIT_NO) do {
    VAR ^= (1<<BIT_NO);
} while (0)

#define BIT_GET(VAR, BIT_NO) ((VAR >> BIT_NO) & 1)
```

Örnek

```
void print_bit(int var, int bit_count) {



    printf("\n");
}
```

Örnek

```
void print_bit(int var, int bit_count) {
    int i;
    for (i = bit_count-1 ; i >= 0 ; i--) {

    }
    printf("\n");
}
```

Örnek

```
void print_bit(int var, int bit_count) {
    int i;
    for (i = bit_count-1 ; i >= 0 ; i--) {
        printf("%d", BIT_GET(var, i));
    }
    printf("\n");
}
```

Örnek

```c
#include<stdio.h>
int main() {
    int a = 0b0011;
    print_bit(a, 4);

    return 0;
}
```

Örnek

```c
#include<stdio.h>
int main() {
    int a = 0b0011;
    print_bit(a, 4);
    BIT_SET(a, 0);
    print_bit(a, 4);

    return 0;
}
```

Örnek

```c
#include<stdio.h>
int main() {
    int a = 0b0011;
    print_bit(a, 4);
    BIT_SET(a, 0);
    print_bit(a, 4);
    BIT_RESET(a, 2);
    print_bit(a, 4);



    return 0;
}
```

Örnek

```c
#include<stdio.h>
int main() {
    int a = 0b0011;
    print_bit(a, 4);
    BIT_SET(a, 0);
    print_bit(a, 4);
    BIT_RESET(a, 2);
    print_bit(a, 4);
    BIT_TOGGLE(a, 3);
    print_bit(a, 4);



    return 0;
}
```

Örnek

```c
#include<stdio.h>
int main() {
    int a = 0b0011;
    print_bit(a, 4);
    BIT_SET(a, 0);
    print_bit(a, 4);
    BIT_RESET(a, 2);
    print_bit(a, 4);
    BIT_TOGGLE(a, 3);
    print_bit(a, 4);
    printf("a'nin 0. biti: %d\n", BIT_GET(a, 0));
    printf("a'nin 1. biti: %d\n", BIT_GET(a, 1));
    printf("a'nin 2. biti: %d\n", BIT_GET(a, 2));
    printf("a'nin 3. biti: %d\n", BIT_GET(a, 3));
    return 0;
}
```

Sorular