

# Talsystem

- Vi använder dagligen det decimala talsystemet
  - Bas 10
    - Dvs: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Men det finns andra talsystem
  - Binära: bas 2 (0, 1)
  - Oktal: bas 8 (0, 1, 2, 3, 4, 5, 6, 7)
  - Hexadecimala: bas 16 (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)
- Processorer använder det binära talsystemet
  - Det är enkelt, 1 / 0 mappar till på / av
    - Switch-teknik idag kan inte på ett pålitligt sätt hålla fler läge än på / av

# Decimala talssystemet

- Vad betyder 476 decimal egentligen?
  - $(4 * 100) + (7 * 10) + (6 * 1) = 476$

$10^5$	$10^4$	$10^3$	$10^2$	$10^1$	$10^0$
100.000	10.000	1.000	100	10	1
			<b>4</b>	<b>7</b>	<b>6</b>

# Binära talssystemet

- Så vad betyder 1001 binär?
  - $(1 * 8) + (0 * 4) + (0 * 2) + (1 * 1) = 9$

	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
32		16	8	4	2	1
			1	0	0	1

# Hexadecimala talsystemet

- Så vad betyder A2F hexadecimal?
  - $(10 * 256) + (2 * 16) + (15 * 1) = 2607$

$16^5$	$16^4$	$16^3$	$16^2$	$16^1$	$16^0$
1.048.576	65.536	4.096	256	16	1
			<b>A (10)</b>	<b>2</b>	<b>F (15)</b>

# Byte

- BIT = Binary Digit
- 1 bit = 0 eller 1
- 8 bitar = 1 byte
- Minsta värdet för en byte?
  - 00000000 = 0
- Största värdet för en byte?
  - 11111111 = 255

# Byte

- 256 möjliga olika värden
- Räcker till:
  - Alla små bokstäver (26st på engelska)
  - Alla stora bokstäver (26st)
  - Decimala siffror (9st)
  - Och andra symboler (!, " ", #, %, &, /, osv)
- En byte räcker till de flesta tänkbara tecknen - på engelska iaf
- En *char* = byte

# Short Int

- 1 byte räcker dock inte så långt för heltal
- short int = 2 byte (oftast)
- Största värdet för 2 bytes?
  - 1111111111111111 = 65.535
- Bättre!
- Men - vi vill även kunna lagra negativa heltal
  - Använd första biten till + eller -
  - Så kan vi uttrycka -32.768 till +32.767

# Int

- 2 byte räcker inte så långt för heltal heller
- int = 4 byte
- -2.147.483.648 till +2.147.483.647

```
#include <limits.h>
```

```
int main() {
```

```
    printf("The number of bits in a byte %d\n", CHAR_BIT);

    printf("The minimum value of SIGNED CHAR = %d\n", SCHAR_MIN);
    printf("The maximum value of SIGNED CHAR = %d\n", SCHAR_MAX);
    printf("The maximum value of UNSIGNED CHAR = %d\n", UCHAR_MAX);

    printf("The minimum value of SHORT INT = %d\n", SHRT_MIN);
    printf("The maximum value of SHORT INT = %d\n", SHRT_MAX);

    printf("The minimum value of INT = %d\n", INT_MIN);
    printf("The maximum value of INT = %d\n", INT_MAX);

    printf("The minimum value of CHAR = %d\n", CHAR_MIN);
    printf("The maximum value of CHAR = %d\n", CHAR_MAX);

    printf("The minimum value of LONG = %ld\n", LONG_MIN);
    printf("The maximum value of LONG = %ld\n", LONG_MAX);
```

Ok så det är ettor och nollor...vad är det bra för??

- Nu ska vi kolla på OPERATIONER för att sätta 0:or och 1:or
- Och läsa om en bit är en nolla eller en etta

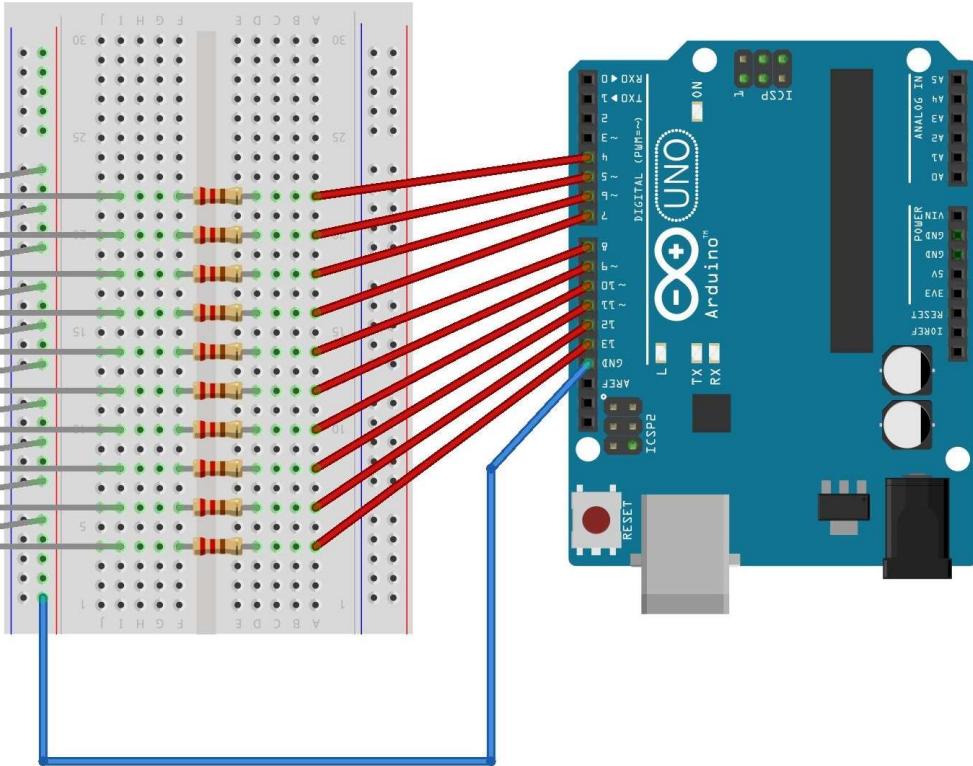
Ok men varför?

# VIKTIGT - IOT!!!

Varje led borde kontrolleras av  
egen 1 eller 0 eller hur?

```
int led4 = 1;  
int led5 = 1;  
int led6 = 1;  
int led7 = 0;
```

TOTAL WASTE!!!!

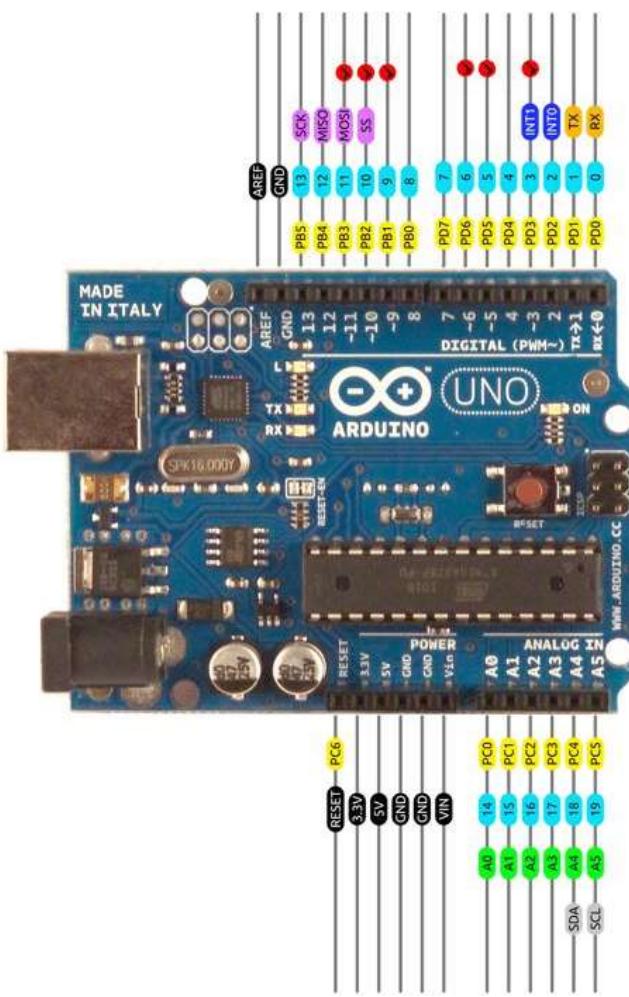


# EN CHAR (BYTE) kan kontrollera 8 LEDS

Bit fields (flags). På en device är det begränsat med utrymme!  
Kan vi spara state i en BIT (tillsammans med andra states i en single char)  
Så vinner vi mycket!

Low level communication (sockets etc)

# VIKTIGT - IOT!!



```
int led = 7;  
void setup() {  
  pinMode(led, OUTPUT);  
}  
  
void loop() {  
  digitalWrite(led, HIGH);  
  delay(1000);  
  digitalWrite(led, LOW);  
  delay(1000);  
}
```

AVR   DIGITAL   ANALOG   POWER   SERIAL   SPI   I2C   PWM   INTERRUPT

# Kod utan Arduinolib == måste vara bitmanipulerig

```
DDRD |= (1 << PB7);

while (1)
{
    PORTD |= (1 << 7);
    _delay_ms(1000);
    PORTD &= ~(1 << 7);
    _delay_ms(1000);
    printf("Hello\n");
}
```

# Använd macros!

```
BIT_SET(DDRD, PB7);

while (1)
{
    BIT_SET(PORTD, 7);
    _delay_ms(1000);
    BIT_CLEAR(PORTD, 7);
    _delay_ms(1000);
    printf("Hello\n");
}
```

# Vaddå PORTB etc?

## Port Registers

Port registers allow for lower-level and faster manipulation of the i/o pins of the microcontroller on an Arduino board.

The chips used on the Arduino board (the ATmega8 and ATmega168) have three ports:

- B (digital pin 8 to 13)
- C (analog input pins)
- D (digital pins 0 to 7)

Each port is controlled by three registers, which are also defined variables in the arduino language. The DDR register, determines whether the pin is an INPUT or OUTPUT. The PORT register controls whether the pin is HIGH or LOW, and the PIN register reads the state of INPUT pins set to input with pinMode(). The maps of the [ATmega8](#) and [ATmega168](#) chips show the ports. The newer Atmega328p chip follows the pinout of the Atmega168 exactly.

DDR and PORT registers may be both written to, and read. PIN registers correspond to the state of inputs and may only be read.

DDRD - The Port D Data Direction Register - read/write  
PORTD - The Port D Data Register - read/write  
PIND - The Port D Input Pins Register - read only

PORTB maps to Arduino digital pins 8 to 13. The two high bits (6 & 7) map to the crystal pins and are not usable

DDRB - The Port B Data Direction Register - read/write  
PORTB - The Port B Data Register - read/write  
PINB - The Port B Input Pins Register - read only

PORTC maps to Arduino analog pins 0 to 5. Pins 6 & 7 are only accessible on the Arduino Mini

DDRC - The Port C Data Direction Register - read/write  
PORTC - The Port C Data Register - read/write  
PINC - The Port C Input Pins Register - read only

Each bit of these registers corresponds to a single pin; e.g. the low bit of DDRB, PORTB, and PINB refers to pin PBO (digital pin 8). For a complete mapping of Arduino pin numbers to ports and bits, see the diagram for your chip:  
**ATmega8, ATmega168.** (Note that some bits of a port may be used for things other than i/o; be careful not to change the values of the register bits corresponding to them.)

# Bitwise operators: AND

- A & B

	1	1	0	0	1	1	0	0	=	204
AND	1	0	1	0	1	0	1	0	=	170
=	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>	=	<b>136</b>

Usecase: släck en bit (AND med 0)

# Bitwise operators: OR

- $A | B$

	1	1	0	0	1	1	0	0	=	204
OR	1	0	1	0	1	0	1	0	=	170
=	<b>1</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>0</b>	=	<b>238</b>

Usecase: tänd en bit (OR med 1)

# Bitwise operators:

## XOR

- $A \wedge B$  (exclusive or)

	1	1	0	0	1	1	0	0	=	204
XOR	1	0	1	0	1	0	1	0	=	170
=	0	1	1	0	0	1	1	0	=	102

Usecase: flippa en bit ( $\wedge$  med 1)

# Bitwise operators: right shift

- $A >> 1$

	1	1	0	0	1	1	0	0	=
>>									
=	0	1	1	0	0	1	1	0	=

Flytta alla bitar ett steg åt höger!

# Bitwise operators: left shift

- $A \ll 2$

	1	1	0	0	1	1	0	0	=
>>									204
=	0	0	1	1	0	0	0	0	= 48

Flytta alla bitar två steg åt  
vänster!

```
int i1 = 204 << 2;  
char i2 = 204 << 2;
```

TEST: Blir värdet verkligan

Use case: sätta en bit MED  $<1 <$   
bitno

# Vi kollar Stefans C-kod

char PORTB;

set bit?

clear bit?

flip bit?

check bit?

- OCH MED MACROS ISTÄLLET

# BIT LABB

<http://www.rjhcoding.com/avrc-bit-manip.php>

Utgå från kod:

<https://gist.github.com/aspcodenet/1b09ea951a22cd7cf874a8f30bb59fc>

Det du ska göra är:

- Ändra så din led är inkopplad på pin 4 istället
- lägg in printBits från  
<https://github.com/aspcodenet/yacbitar/blob/main/main.c> så du skriver och kan se
- prova med BIT\_SET macros – vad gillar du?