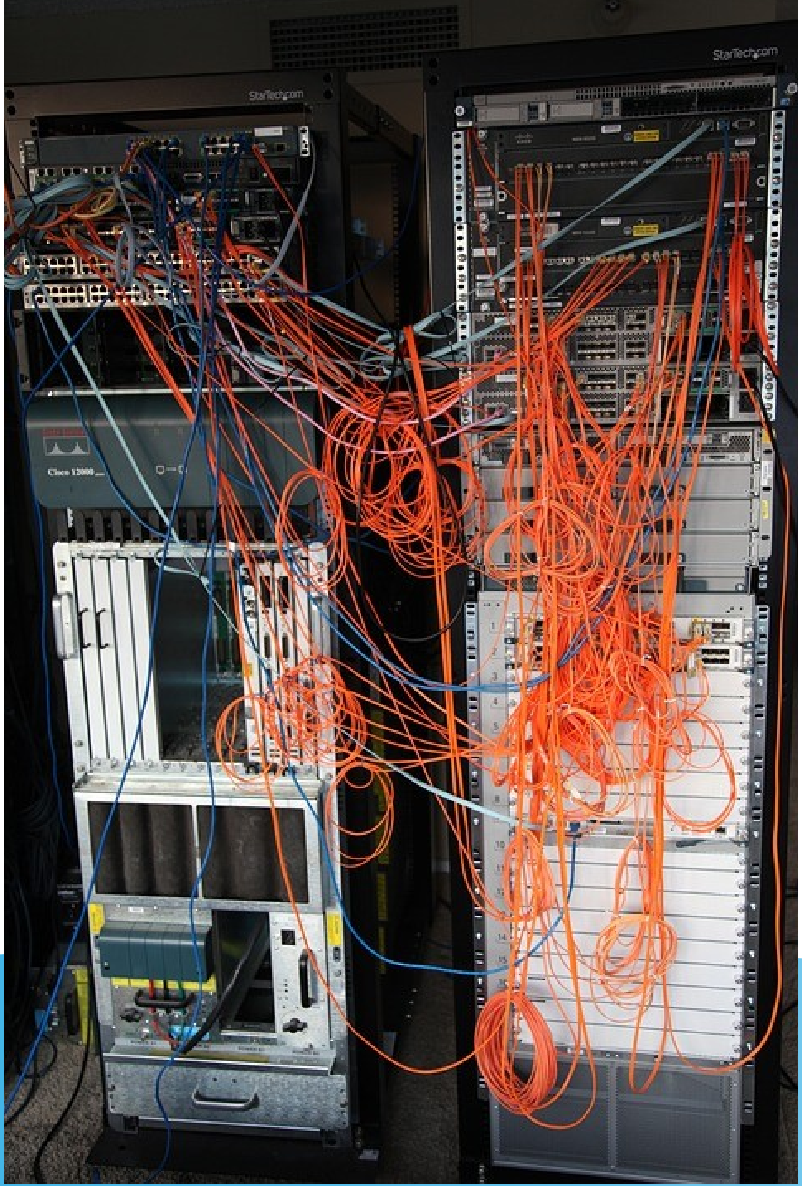
The Bare Metal



https://pixabay.com/de/photos/server-rack-server-elektronik-kabel-441494/

Motivation

- Why even study the „bare metal“ of a computer?
- Your answers here :) …
- Some observations:
  - CPU: dumb but super fast (ns per instruction)
  - Storage: dumb but super large (ms for disks, µs SSD, TB)
  - Periphery: very dumb (think mouse), slow (Mbits/s, 8µs/byte)
  - Network: complicated, slowest but can’t do without (Mbit/s, but long latency)
- How to do something useful with such a machine???

How to make a computer work for us

- We can program something to calculate on the CPU.
- For this, we might need data or input from
  - Storage
  - Periphery
  - Network
- The program will often need to WAIT AGES for the data to arrive.
- Meanwhile, we heat the house with the CPU-fan…
- Can we do better? How?

How to keep our programs running

- Suppose we wrote a nice program as indicated on the prev. slide.
- And then, new hardware is available: faster, not so dumb
- But would our program run right out of the box?
- Why, why not?

The role of the OS

- The OS acts as an abstraction and resource manager of the HW.
- It manages
  - Which program may continue running on the CPU for how long
  - Fetching of data from various peripherals (RAM, disk, network)
- It presents the hardware as a consistent „computer“ to programs.
- Therfore,
  - **The OS is the ONLY program directly interacting with the HW!**
  - And it needs support from the HW to do it well!

Content

- What happens on a double click???
- A 10000ft view of a PC
- The x86 Architecture
- Assembler capabilities
- From C to hardware – a long journey (from Java, still longer :))

What happens on a double click?

- The program starts to run…
- What IS an Executable?
  - Human readable code?
  - Machine readable code!
- Where does it run from?
  - Persistent storage → Main memory (RAM)
  - Instructions execute on CPU
- What does a program need to run?
  - Memory, CPU time, peripherals

A 10000ft view of a PC

- CPU
  - Computing power
  - Hardware control
- MMU
  - Memory access
- Internal Peripherals Bus (PCIe)
  - Graphics, Extensions
- External Peripherals Bus
  - SATA, USB

https://pixabay.com/de/photos/motherboard-waschbecken-ventilator-197608/

CPU: The x86 Architecture

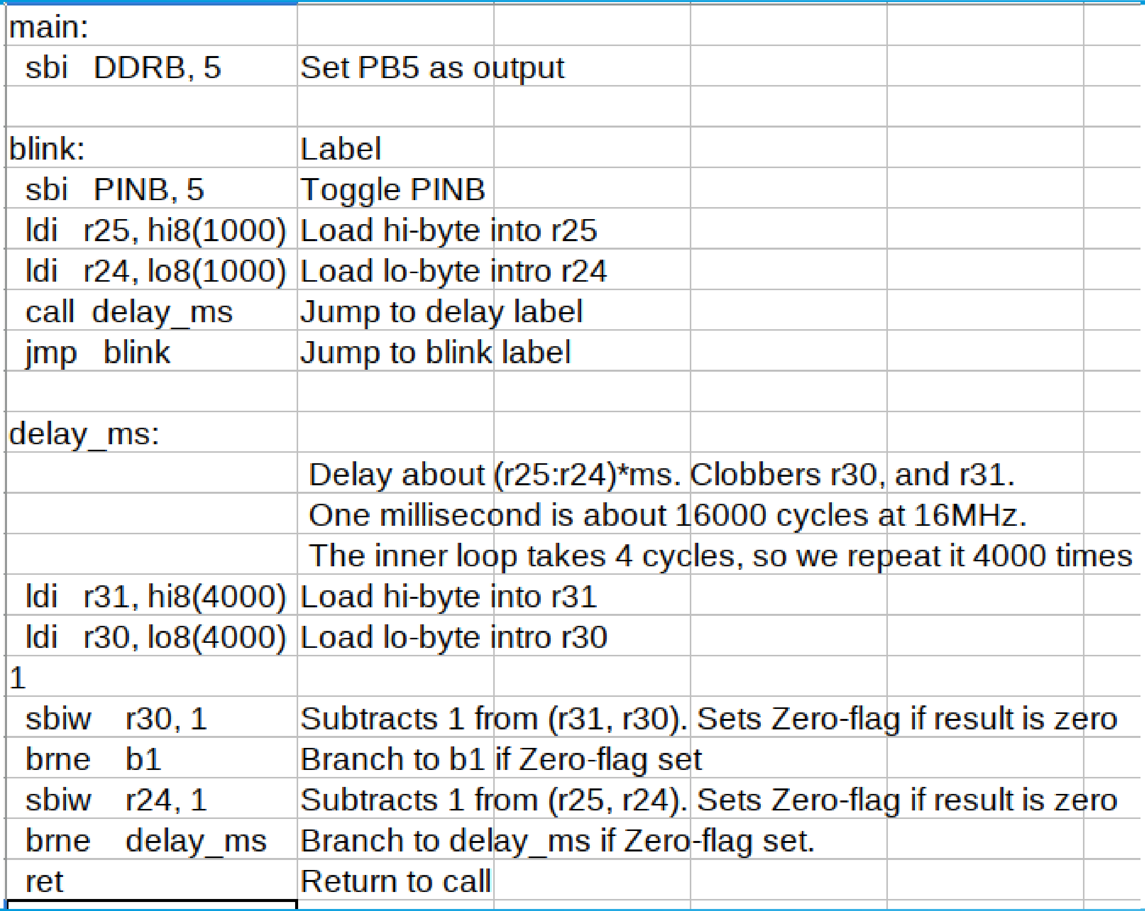
- Most common PC CPU Architecture
- A couple of onboard memory slots (~40) - **registers**
  - Bus width (64bit)
  - Fastest for calculations
- A built-in hardware based programming language
  - Instruction set (humans write this in assembler; **opcodes; CISC**)
- Provides a stack based programming model
- Provides modes which disables certain instructions (**user mode, kernel mode**)

Oh my...

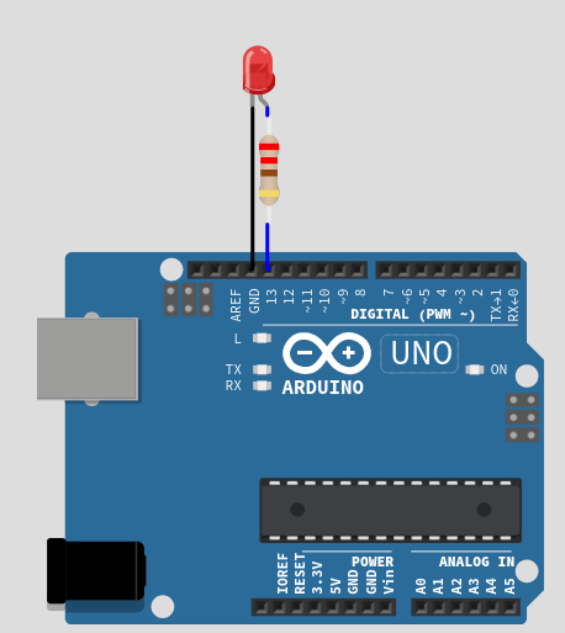
mov ax, 1	Write 1 to register ax	ax = (byte)1
cwd	Copy ax to dx and make it signed	dx = (int)ax
:Label1	Place a mark to jump to	:Label 1
add dx, ax	Add registers ax and dx, store in dx	dx = dx + ax
inc ax	Increment ax by 1	ax = ax +1
cmp ax, 10	Compare ax to 10. Sets status flag	sf = ax == 10 ? 1 : 0
jbe Label1	If status flag == 1, jump	if(sf) goto :Label1

How long does this programm need to count to 10 on a 1 GHz CPU?

Interacting with hardware

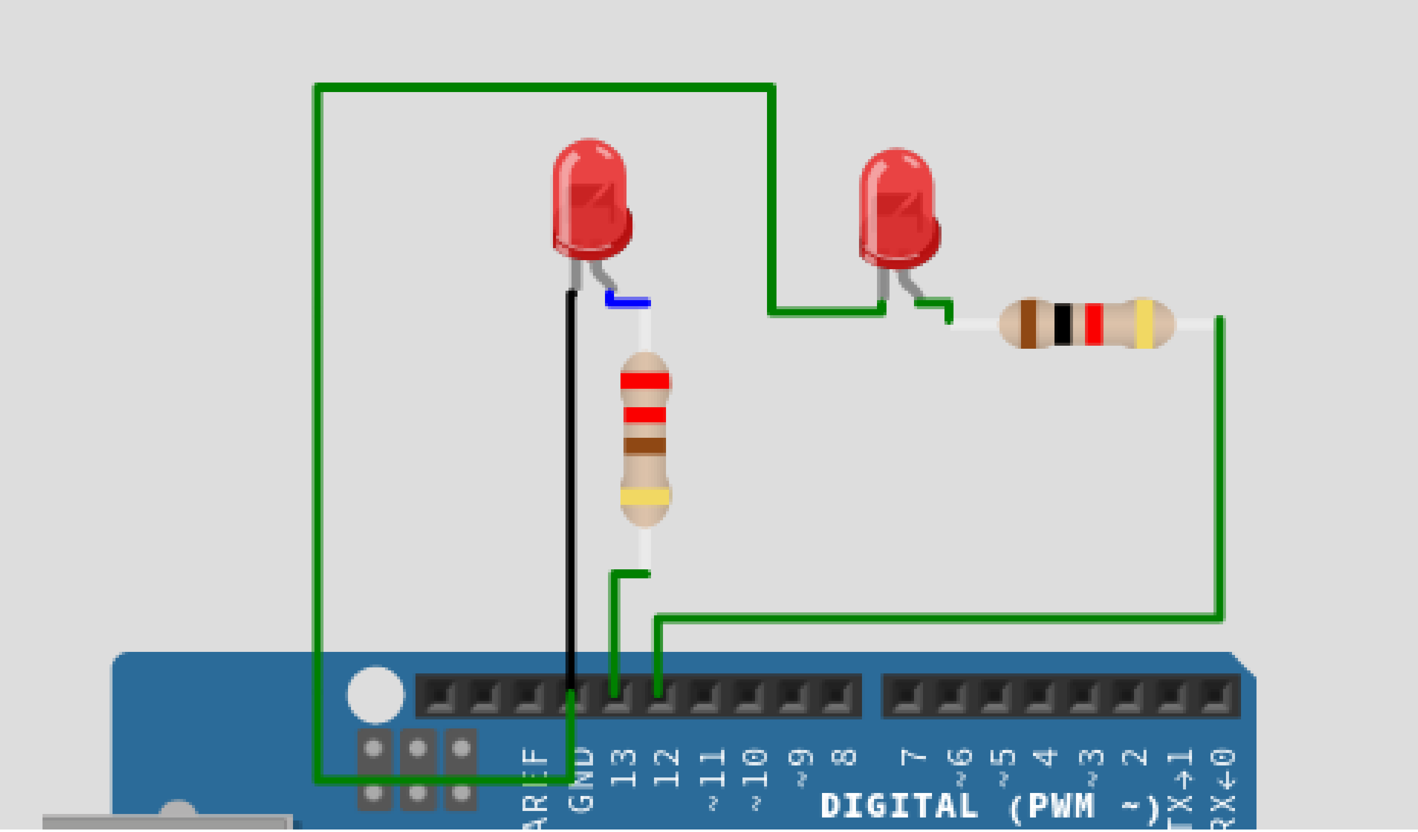


main:	
sbi DDRB, 5	Set PB5 as output
blink:	Label
sbi PINB, 5	Toggle PINB
ldi r25, hi8(1000)	Load hi-byte into r25
ldi r24, lo8(1000)	Load lo-byte intro r24
call delay_ms	Jump to delay label
jmp blink	Jump to blink label
delay_ms:	
	Delay about (r25:r24)*ms. Clobbers r30, and r31.
	One millisecond is about 16000 cycles at 16MHz.
	The inner loop takes 4 cycles, so we repeat it 4000 times
ldi r31, hi8(4000)	Load hi-byte into r31
ldi r30, lo8(4000)	Load lo-byte intro r30
1	
sbiw r30, 1	Subtracts 1 from (r31, r30). Sets Zero-flag if result is zero
brne b1	Branch to b1 if Zero-flag set
sbiw r24, 1	Subtracts 1 from (r25, r24). Sets Zero-flag if result is zero
brne delay_ms	Branch to delay_ms if Zero-flag set.
ret	Return to call

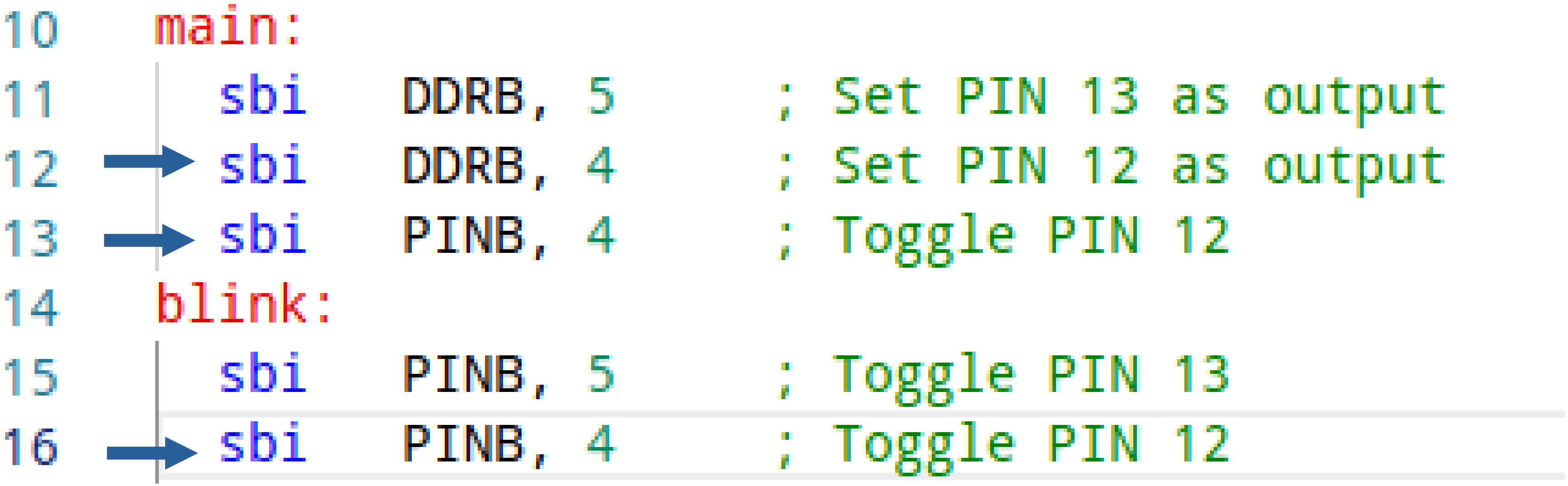


https://wokwi.com/arduino/projects/290348681199092237

Alternating LED Blinker: HW

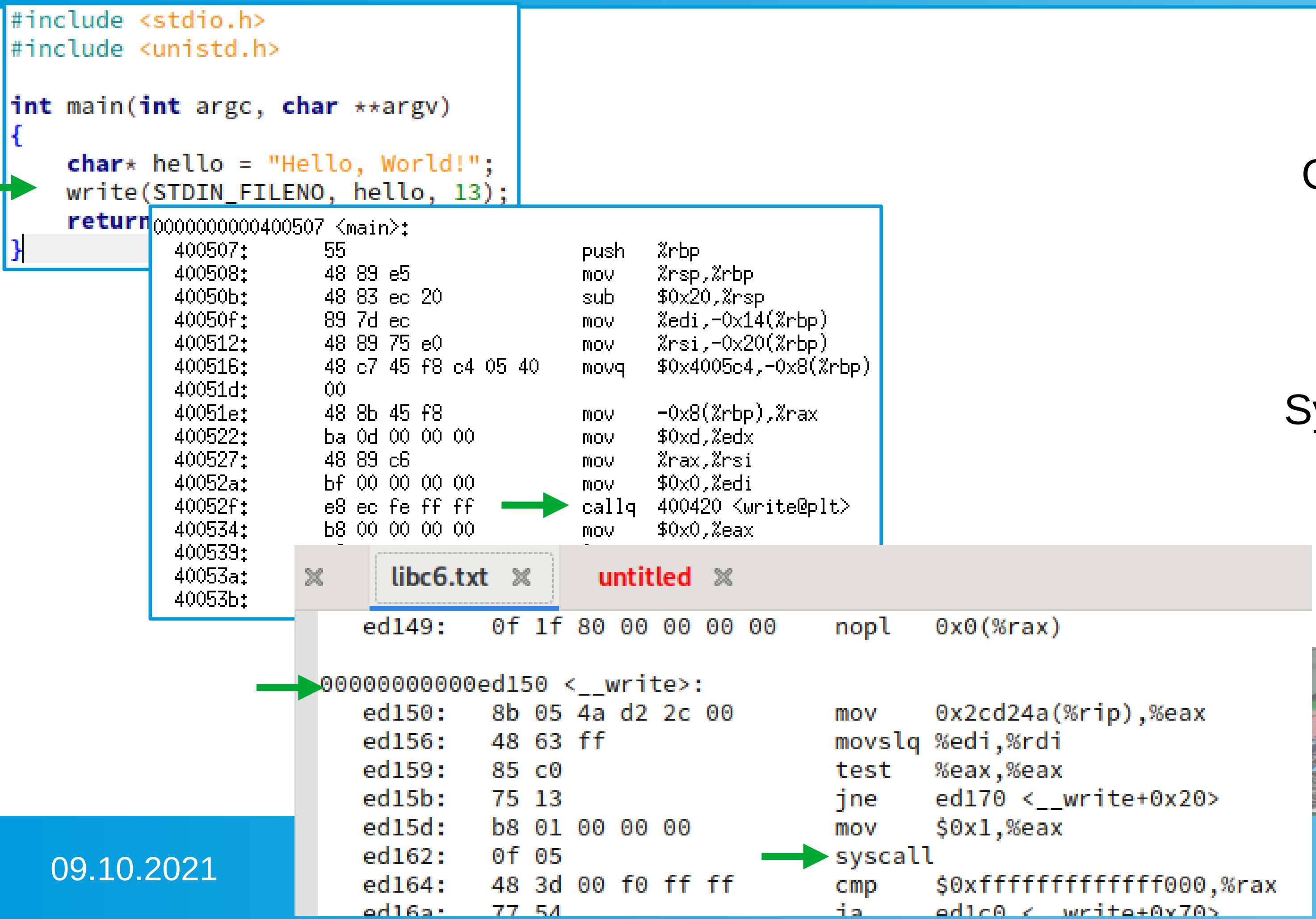


Alternating LED Blinker: SW



```
main:
    sbi    DDRB, 5         ; Set PIN 13 as output
    sbi    DDRB, 4         ; Set PIN 12 as output
    sbi    PINB, 4         ; Toggle PIN 12
blink:
    sbi    PINB, 5         ; Toggle PIN 13
    sbi    PINB, 4         ; Toggle PIN 12
```

From C to hardware



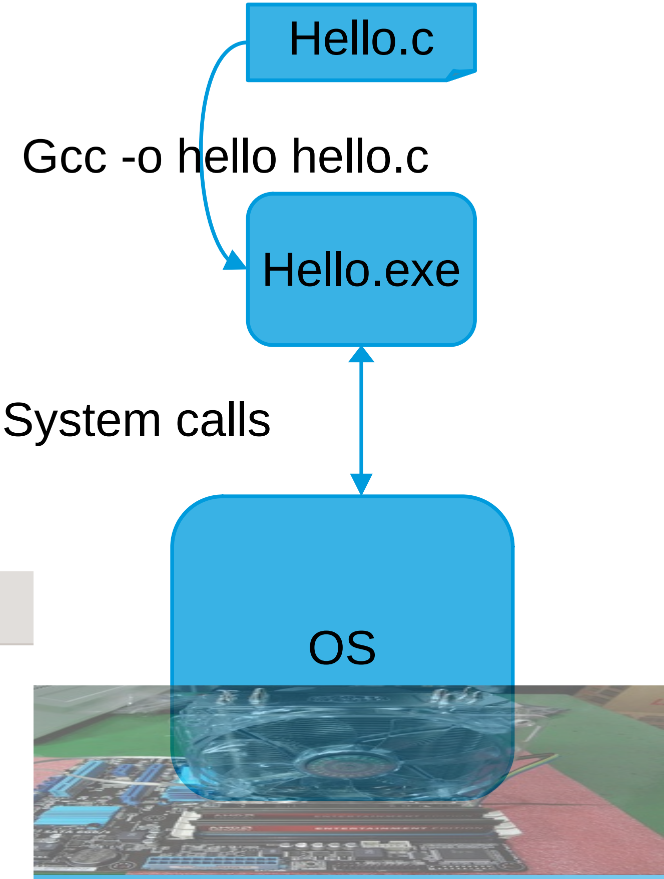
```
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    char* hello = "Hello, World!";
    write(STDIN_FILENO, hello, 13);
    return
}
```

```
0000000000400507 <main>:
  400507:       55                      push   %rbp
  400508:       48 89 e5                mov    %rsp,%rbp
  40050b:       48 83 ec 20             sub    $0x20,%rsp
  40050f:       89 7d ec                mov    %edi,-0x14(%rbp)
  400512:       48 89 75 e0             mov    %rsi,-0x20(%rbp)
  400516:       48 c7 45 f8 c4 05 40    movq   $0x4005c4,-0x8(%rbp)
  40051d:       00
  40051e:       48 8b 45 f8             mov    -0x8(%rbp),%rax
  400522:       ba 0d 00 00 00          mov    $0xd,%edx
  400527:       48 89 c6                mov    %rax,%rsi
  40052a:       bf 00 00 00 00          mov    $0x0,%edi
  40052f:       e8 ec fe ff ff          callq  400420 <write@plt>
  400534:       b8 00 00 00 00          mov    $0x0,%eax
  400539:
  40053a:
  40053b:
```

```
    ed149:   0f 1f 80 00 00 00 00    nopl   0x0(%rax)

00000000000ed150 <__write>:
    ed150:   8b 05 4a d2 2c 00       mov    0x2cd24a(%rip),%eax
    ed156:   48 63 ff                movslq %edi,%rdi
    ed159:   85 c0                   test   %eax,%eax
    ed15b:   75 13                   jne    ed170 <__write+0x20>
    ed15d:   b8 01 00 00 00          mov    $0x1,%eax
    ed162:   0f 05                   syscall
    ed164:   48 3d 00 f0 ff ff       cmp    $0xfffffffffffff000,%rax
    ed16a:   77 54                   ja     ed1c0 <__write+0x70>
```



Key points

- The **OS** is the only program interacting directly with **CPU/HW.**.
- It provides **System Calls** for programs to access HW.
  - Representation of different HW with same System Calls.
- Normal programs are **NOT ALLOWED** to interact directly with **HW**
  - CPU in **USER MODE**
- Only the OS interacts with HW
  - CPU in **KERNEL MODE**
- Happy hacking :)

The Bare Metal – Your notes

- Hardware eines modernen PC
- Rolle des Betriebssystem
- CPU: Was kann diese Komponente? Spezielle Modi?
- Assembler-Programmierung: Eigenschaften, Struktur