I/O


mov rax, rbx
add rax, rcx
HELLO IT'S ME THE KEYBOARD
I HAVE AN IMPORTANT MESSAGE
E
FALSEKNEES.COM 2016

Besides the cache/memory management unit, what sorts of things does the processor need to talk to?

Abstraction of I/O

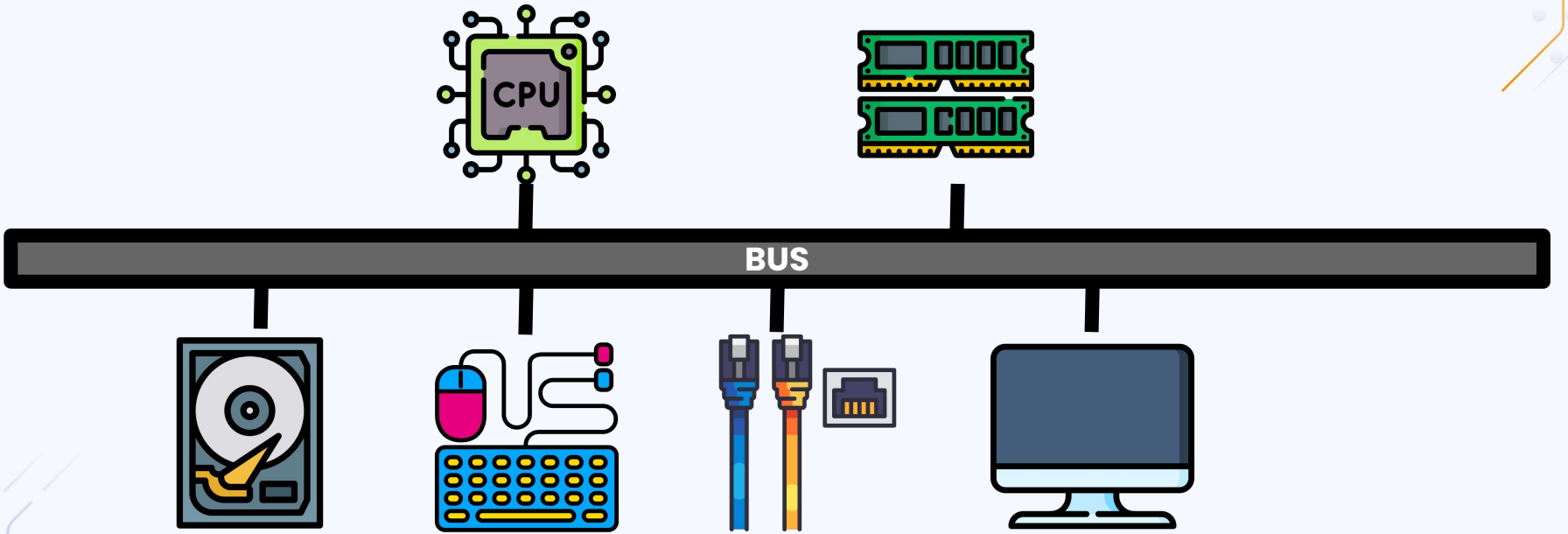
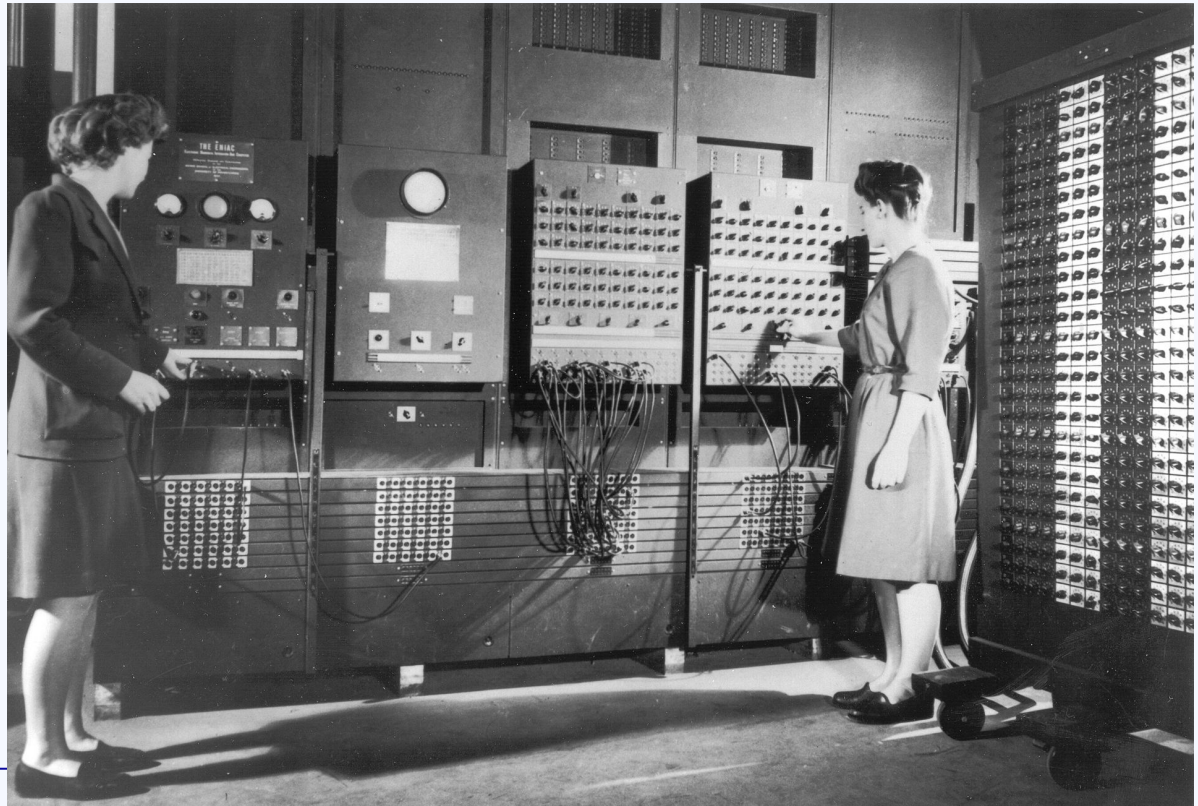


image source: flaticon.com

History of I/O: Plugboard computers (ENIAC, <1946)



History of I/O: UNIVAC 1 (1951)

image source

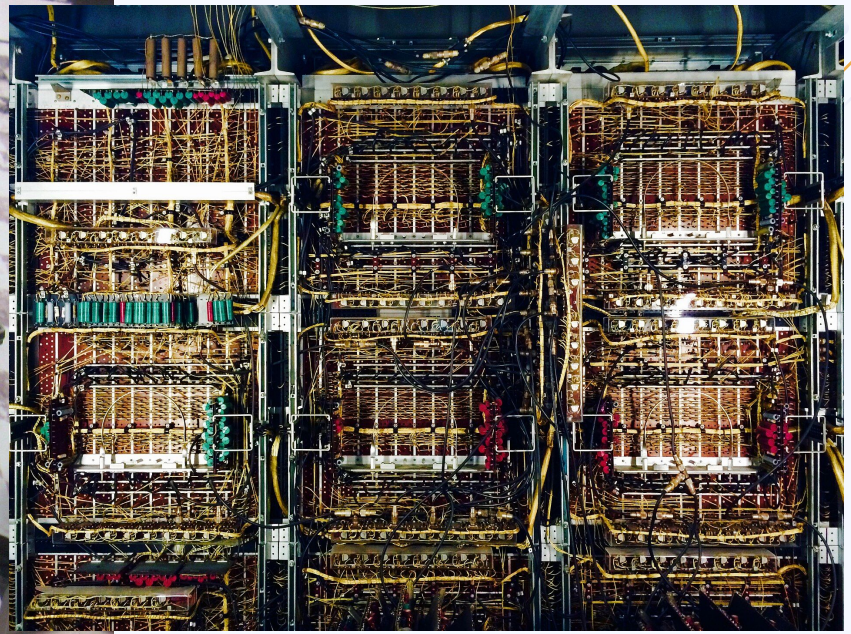


image source

History of I/O: IBM System/360 (>=1964)

image source

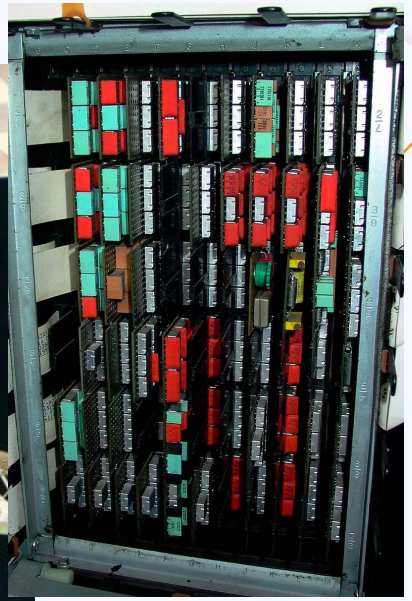


image source

History of I/O: PDP-8 (>=1965)

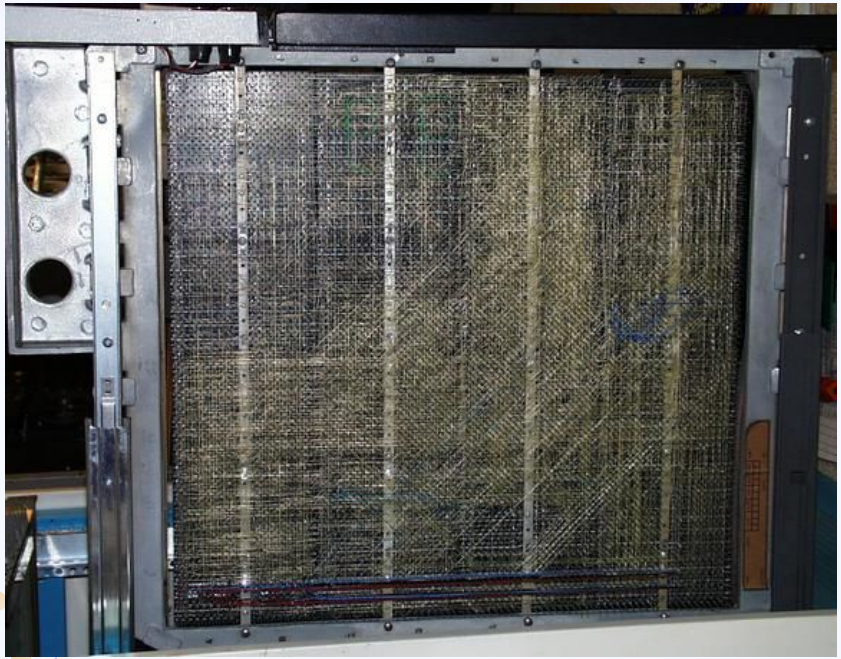


image source

image source

VAX11: Successor to the PDP (1977)

5.2 LA36 TERMINAL

The LA36 DECwriter II is a medium-sized, low-cost, interactive data communications terminal (Figure 5-1). It is designed as an I/O device that is used in the VAX-11/780 system console subsystem and may be used as a remote communications terminal or a user terminal.

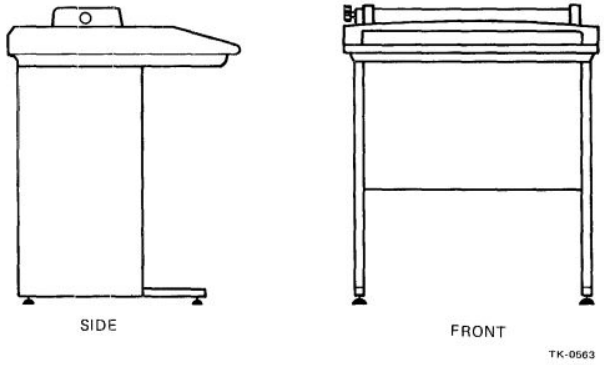
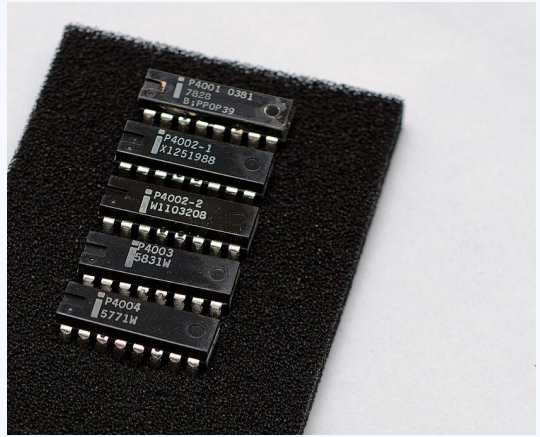


Figure 5-1 LA36 DECwriter II

source (VAX11 hardware user's guide)

image source

History of I/O: Intel 4004/MCS-4 (1971)



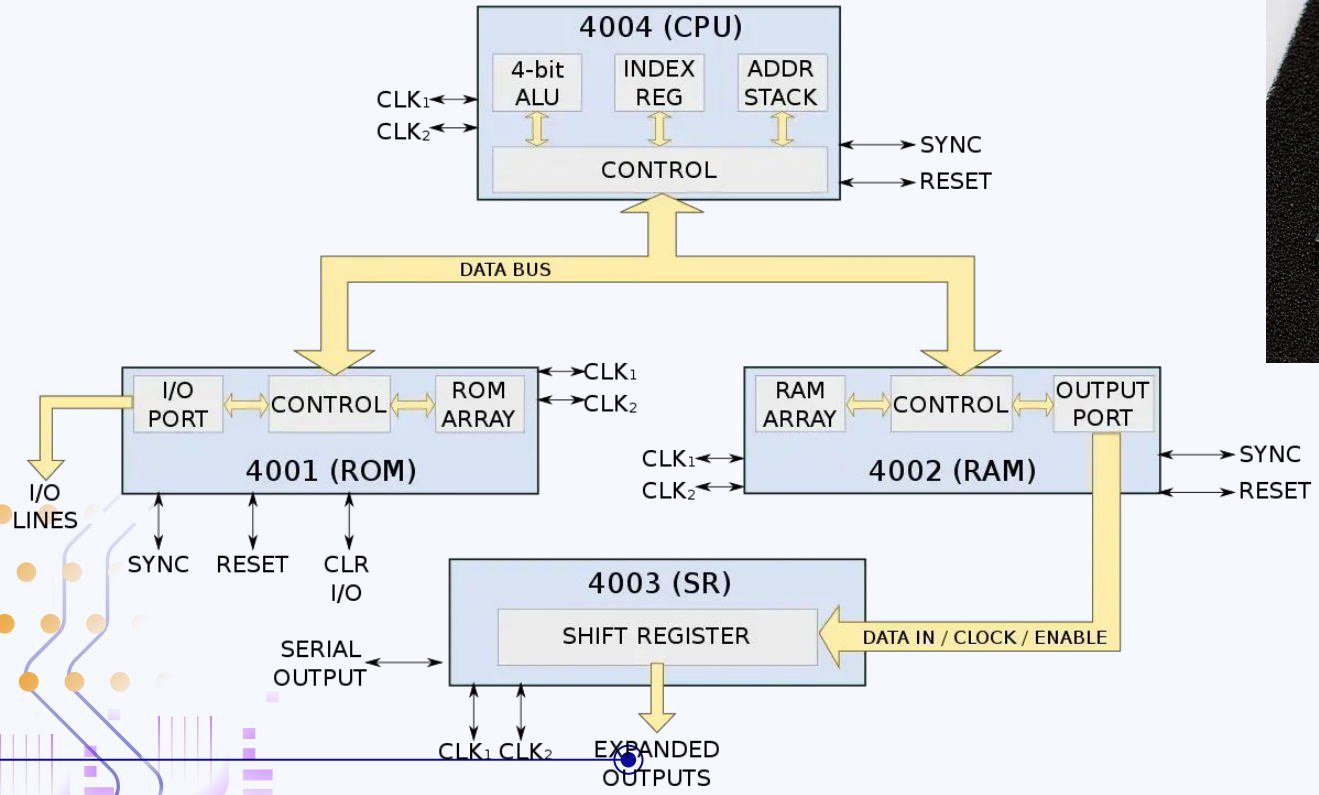
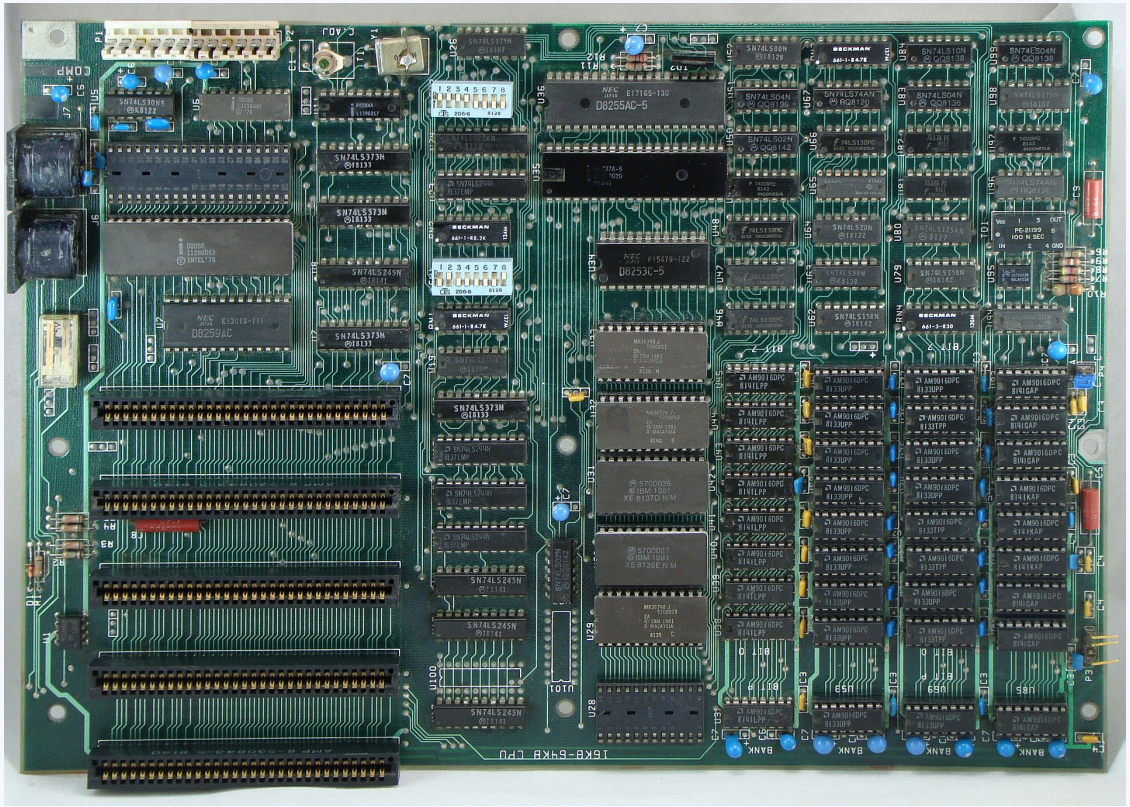


image source

History of I/O: IBM PC (1981)



Motherboards

Motherboard: printed circuit board (PCB) that holds computer components

Chipset: (usually on mobo) circuit that manages connection of CPU w/ memory and peripherals

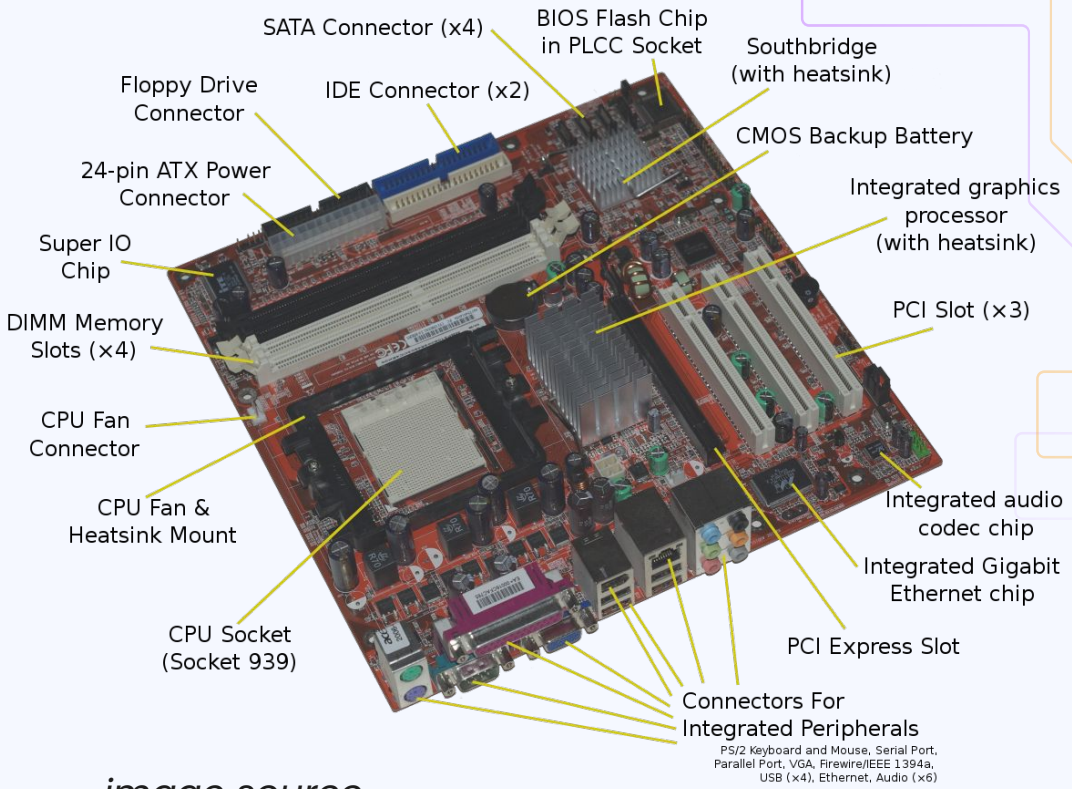


image source

Chipsets (1990s-2000s)

Northbridge: connects CPU, RAM, GPU

Southbridge: slower, connects I/O

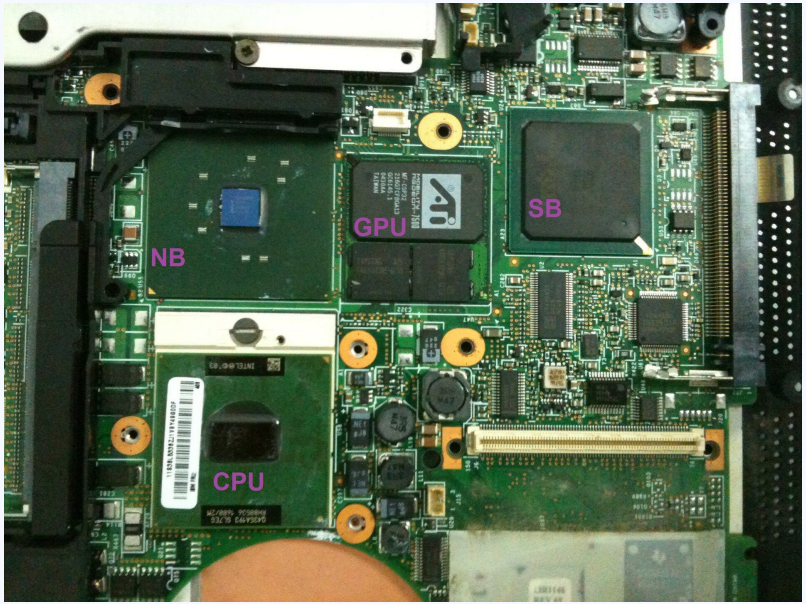
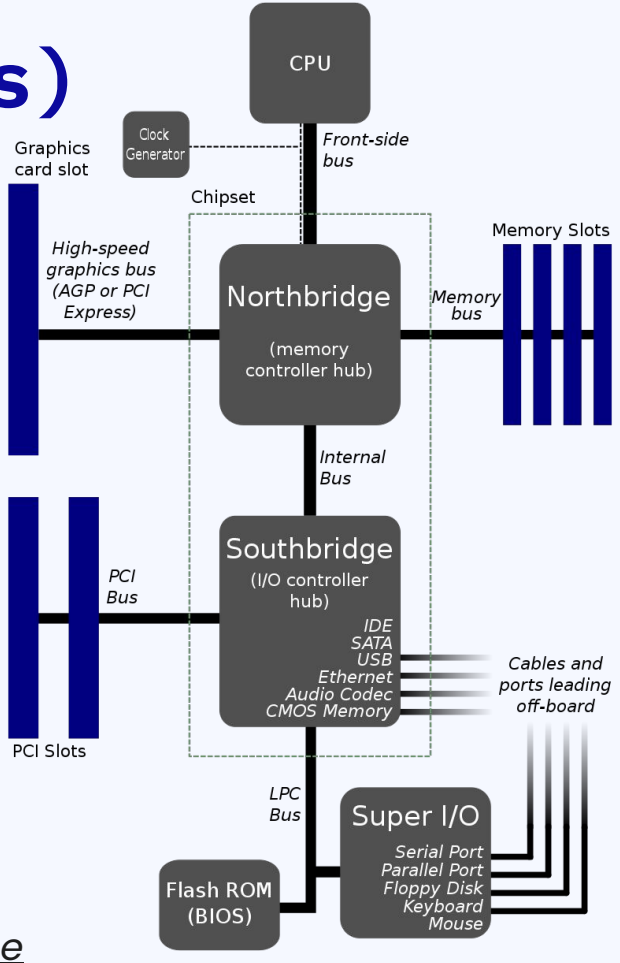
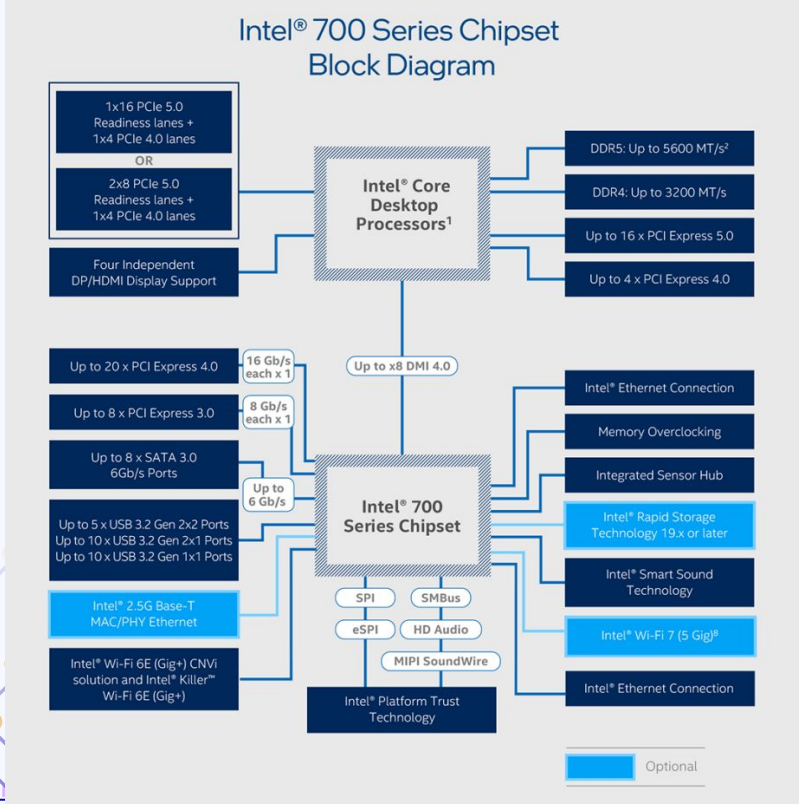


image source

image source



Evolution of chipsets (Intel)



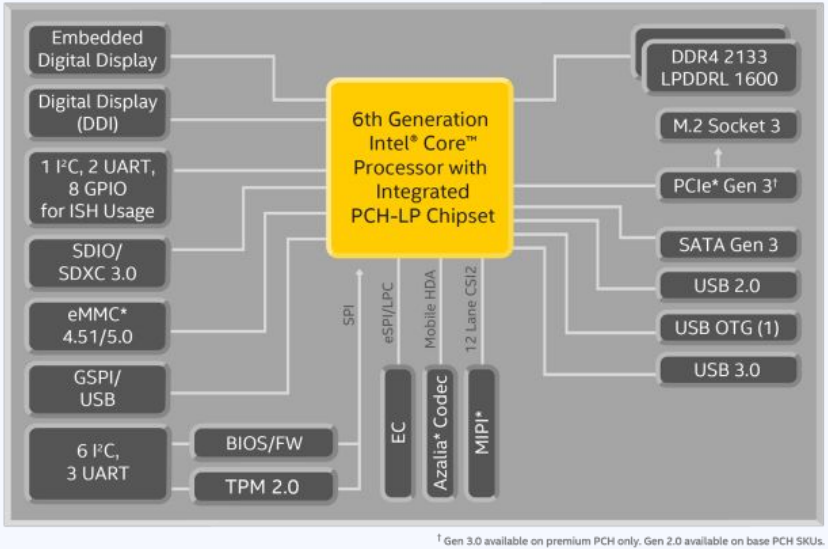


image source

image source

Buses

There are many ways to transfer data

Different bus standards are used for different applications

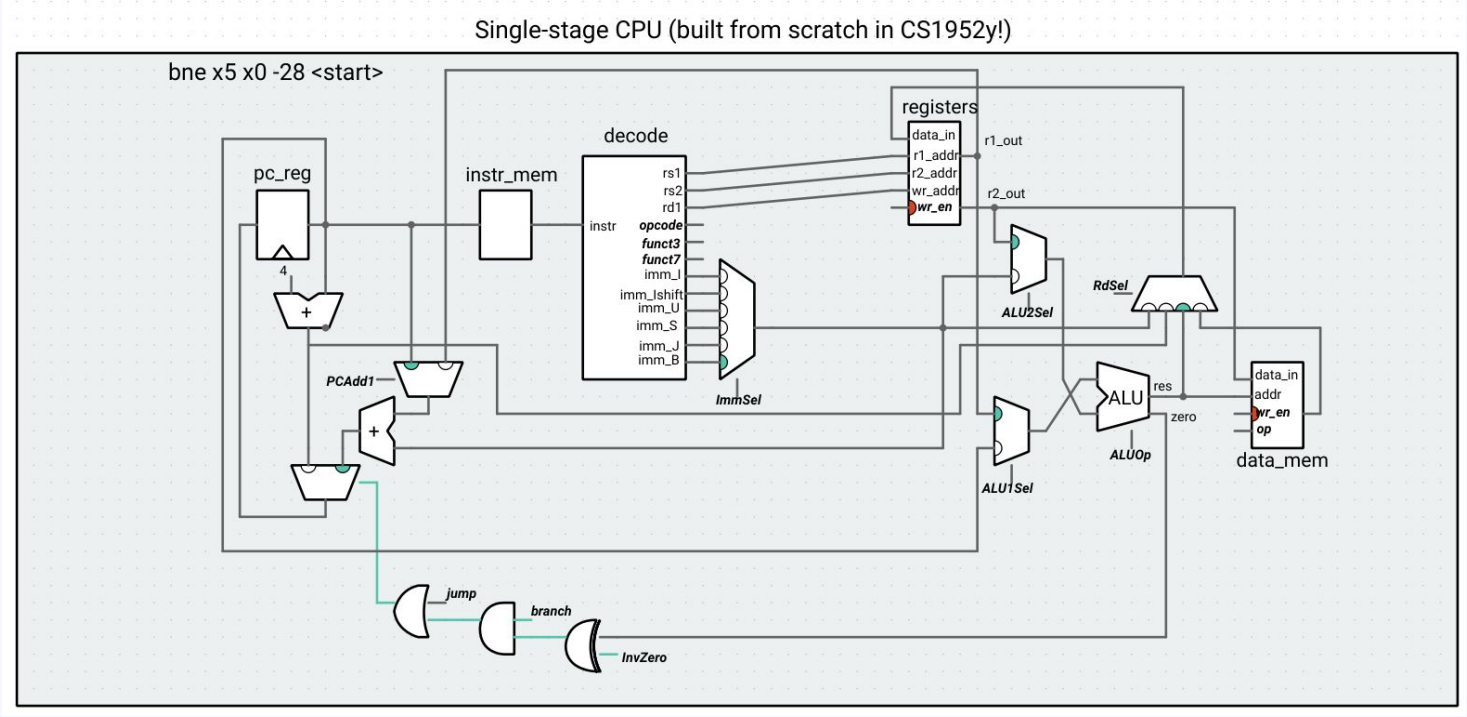
SATA is used for storage (also **NVME**)

PCI/e is used for many things (GPU, sound, ethernet…)

SPI is used for serial embedded communication

We’re not going to memorize these technologies – just know that there are different protocols for them

How does a CPU access an I/O bus? There’s no lio or sio instruction in RISCV…


Single-stage CPU (built from scratch in CS1952y!)
bne x5 x0 -28 <start>
pc_reg
4
+
instr_mem
decode
instr
rs1
rs2
rd1
opcode
funct3
funct7
imm_I
imm_Ishift
imm_U
imm_S
imm_J
imm_B
ImmSel
registers
data_in
r1_addr
r2_addr
wr_addr
wr_en
r1_out
r2_out
ALU2Sel
RdSel
PCAdd1
+
ALU
res
zero
ALUOp
ALU1Sel
data_in
addr
wr_en
op
data_mem
jump
branch
InvZero

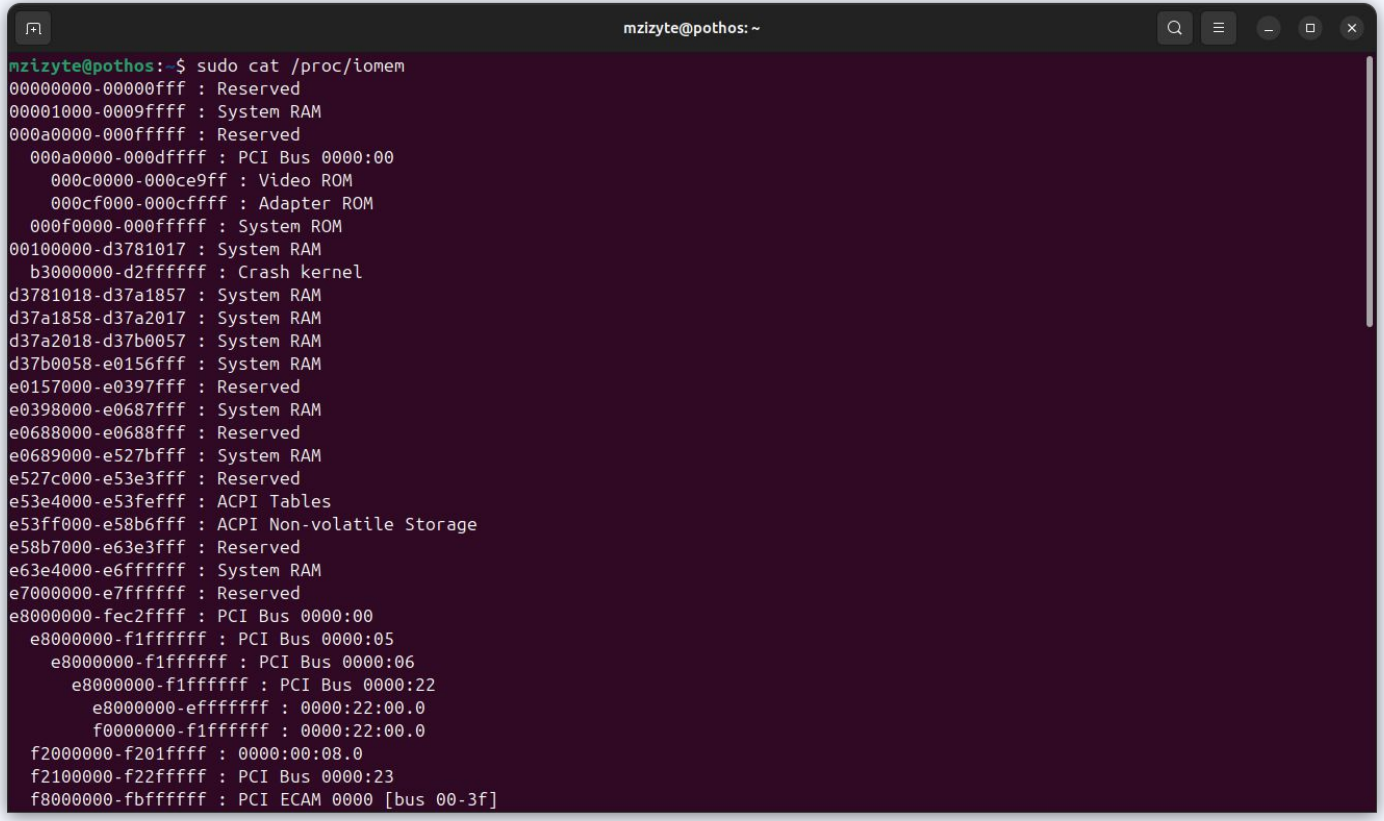
Memory-mapped I/O

Memory-mapped I/O: hardware translation of some memory addresses to status and control registers of I/O devices

- CPU writes/reads that address as usual, gets info about device
- Kernel-space (not user-space) addresses

Contrast with port-mapped I/O: special instructions to access I/O ports

- x86 `in` and `out` instructions (x86 supports both port- and memory-mapped I/O; see chapter 19 of the Intel 64 manual)


mzizyte@pothos: ~
mzizyte@pothos:~$ sudo cat /proc/iomem
00000000-00000fff : Reserved
00001000-0009ffff : System RAM
000a0000-000fffff : Reserved
  000a0000-000dffff : PCI Bus 0000:00
    000c0000-000ce9ff : Video ROM
    000cf000-000cffff : Adapter ROM
  000f0000-000fffff : System ROM
00100000-d3781017 : System RAM
  b3000000-d2ffffff : Crash kernel
d3781018-d37a1857 : System RAM
d37a1858-d37a2017 : System RAM
d37a2018-d37b0057 : System RAM
d37b0058-e0156fff : System RAM
e0157000-e0397fff : Reserved
e0398000-e0687fff : System RAM
e0688000-e0688fff : Reserved
e0689000-e527bfff : System RAM
e527c000-e53e3fff : Reserved
e53e4000-e53fefff : ACPI Tables
e53ff000-e58b6fff : ACPI Non-volatile Storage
e58b7000-e63e3fff : Reserved
e63e4000-e6ffffff : System RAM
e7000000-e7ffffff : Reserved
e8000000-fec2ffff : PCI Bus 0000:00
  e8000000-f1ffffff : PCI Bus 0000:05
    e8000000-f1ffffff : PCI Bus 0000:06
      e8000000-f1ffffff : PCI Bus 0000:22
        e8000000-efffffff : 0000:22:00.0
        f0000000-f1ffffff : 0000:22:00.0
  f2000000-f201ffff : 0000:00:08.0
  f2100000-f22fffff : PCI Bus 0000:23
  f8000000-fbffffff : PCI ECAM 0000 [bus 00-3f]

What does that look like?

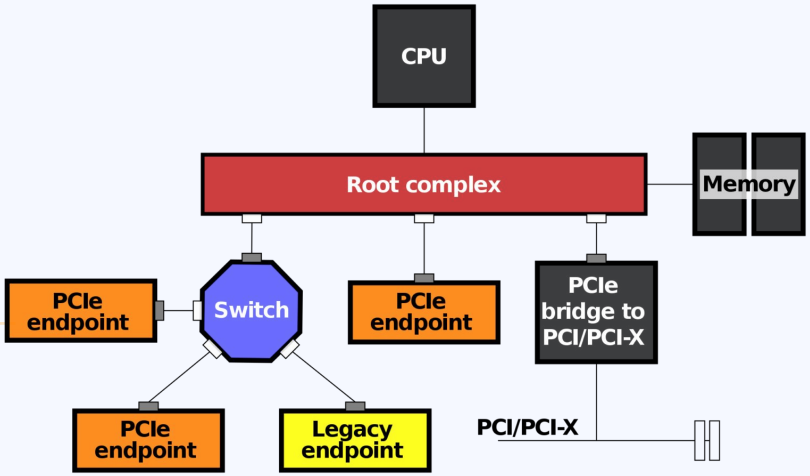
```
# Kernel process transfer 4kB page from disk to physical memory:
```

Bypassing CPU for memory access

Direct Memory Access (DMA) transfers data to and from memory without going through the CPU, using the controller of the I/O device

PCIe uses an optimized version of DMA that involves bus arbitration

image source



How should an I/O event (such as a keyboard key press) be detected and handled by the computer?

Polling

Manually check the status of I/O device periodically

Pros:

Check message app over and over for your crush to text you

Interrupts

Disrupt CPU execution when an I/O operation happens

Pros:

Get push notification that your crush has texted you

Exceptions vs Interrupts

Exception: unscheduled event that disrupts execution (P&H chapter 4)

- SW source: RISC-V ecall/ebreak instructions to invoke supervisor/debugger
- HW source: divide by 0 or page fault (**we covered this!**)

Interrupt: an exception that comes from outside the CPU

- DMA I/O interrupt

Umbrella term: **trap**

Warning: terminology changes depending on textbook/context

Review: handle traps

1) Figure out if we need to stop immediately or finish current instruction

   Hardware exceptions: immediately (can't proceed after dividing by 0)

   I/O interrupts: handle asynchronously (finish instruction)
2) Save processor state and provide exception info to supervisor
3) Switch control to supervisor
4) Go to PC of exception-handling routine
5) Handle exception
6) Switch back

Reminder: exception CSRs

SEPC holds address of exception-causing instr

SCAUSE holds the cause

STVAL holds info about exception (e.g. fault-causing virtual address)

STVEC holds address of supervisor exception-handler

SIP holds pending status of potential interrupts *(how I/O communicates to CPU about interrupt)*

Alternative approach: vectored interrupts (arm)

Interrupt	Exception Code	Description
1	0	*Reserved*
1	1	Supervisor software interrupt
1	2–4	*Reserved*
1	5	Supervisor timer interrupt
1	6–8	*Reserved*
1	9	Supervisor external interrupt
1	10–15	*Reserved*
1	≥16	*Designated for platform use*
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	10–11	*Reserved*
0	12	Instruction page fault
0	13	Load page fault
0	14	*Reserved*
0	15	Store/AMO page fault
0	16–23	*Reserved*
0	24–31	*Designated for custom use*
0	32–47	*Reserved*
0	48–63	*Designated for custom use*
0	≥64	*Reserved*

RISC-V spec v2 Table 4.2