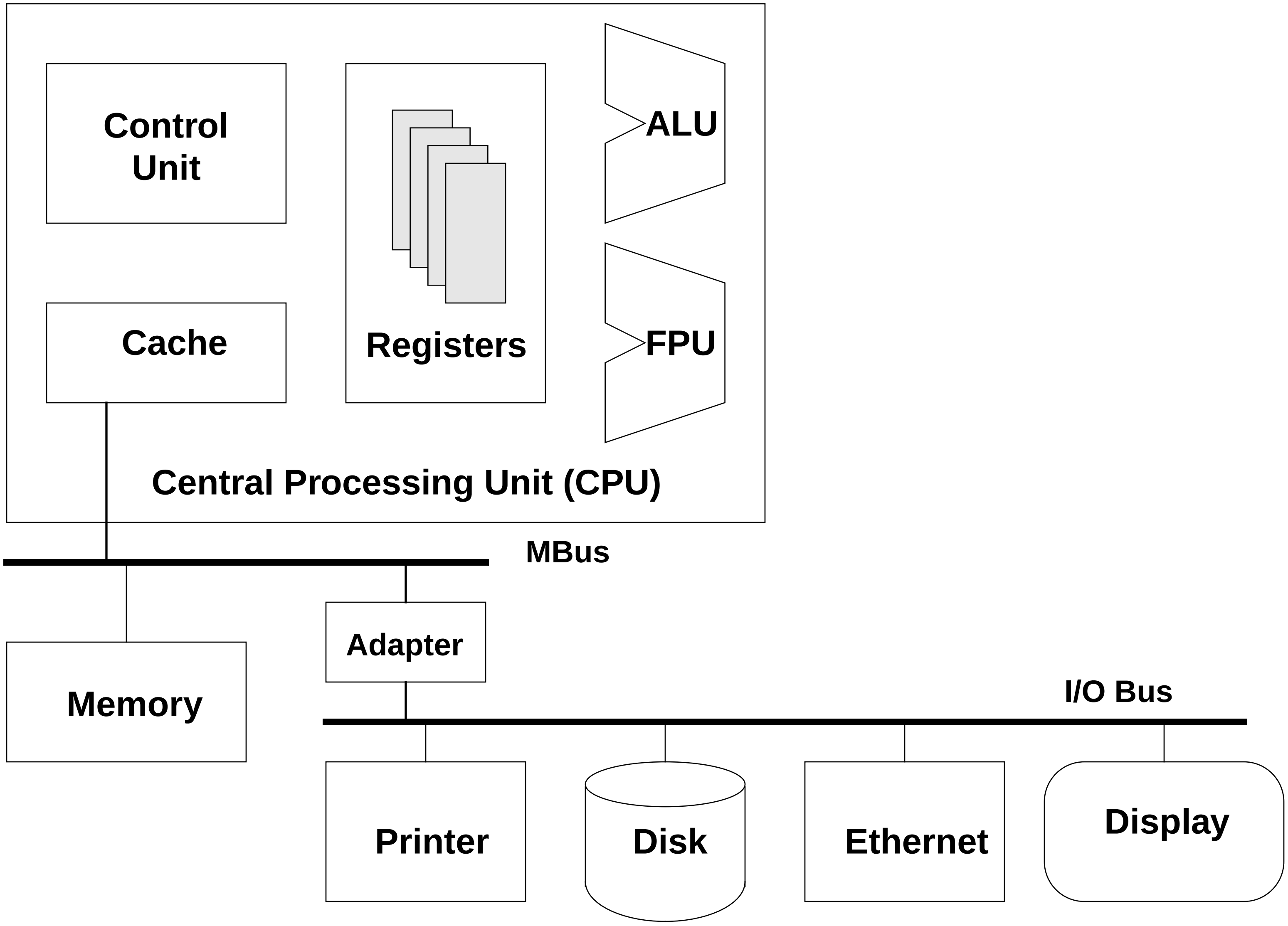
Computer Organizations



Storage Hierarchy

- Registers

  fastest storage (as fast as CPU cycle time), but often very few (<128)

- **Caches**

  "small" but faster than main memory with 1 to 3 levels (1K-4Mbytes)

- Memory

  fairly fast (200ns) and quite large (1-1000Mbytes)

  an array of cells made of dynamic random-access memory (DRAM)

  each cell is usually a byte and has an ***address***

  most machines operate most efficiently on one data type called a ***word***

  words are typically composed of several cells, e.g., 4 bytes in 1 word

  Address size may be unrelated to the amount of allowable memory

- **Disk**

  long latency (10ms to find a block), but large (200M-10Gbytes)

- **Tape**

  Very long latency (seconds to find a block), very low-cost and large (Gbytes)

Compilation to Machine Code

- Compiler:

Source code

```
x = a + b;
```

Assembly language code

```
ld a, %r1
ld b, %r2
add %r1, %r2, %r3
st %r3, x
```

- Assembler

converts each assembly lang. instruction into a bit pattern that hardware understands

these bit patterns constitute machine code

Machine Language

- ***Machine language*** is the bit patterns that specify CPU instructions

- Understanding machine languages helps

  build intuition about the cost of high-level functionality

  learn about low-level operating system support;

  understand how operating systems implement security

  understand what compilers do and how to implement code generators

  understand procedure call mechanisms

  learn how to write ***very fast*** code, when — and only when — it’s necessary

  design a better instruction set and faster processor

Instruction Formats

- ***Instructions*** are composed of

  opcode — specifies function to be performed

  operands — data that is operated on

- Most machines have only a ***few*** formats

- Typical 0, 1, 2, 3-operand instruction format:

  opcode
  opcode dst
  opcode src dst
  opcode src1 src2 dst

Instruction Execution

- **CPU's algorithm for executing a program:**

```
PC <- memory location of the 1st instruction
while ( PC != lastInstructionLocation ) {
    execute ( MEM[ PC ];
    };
```

- Each machine instruction has several phases

  Fetch -- Instruction fetch, increment PC

  Decode -- Instruction decode

  Operand Fetch -- Fetch registers

  Execute --Instruction execution

  Store -- Store results