Bits all the way down

Data representation so far

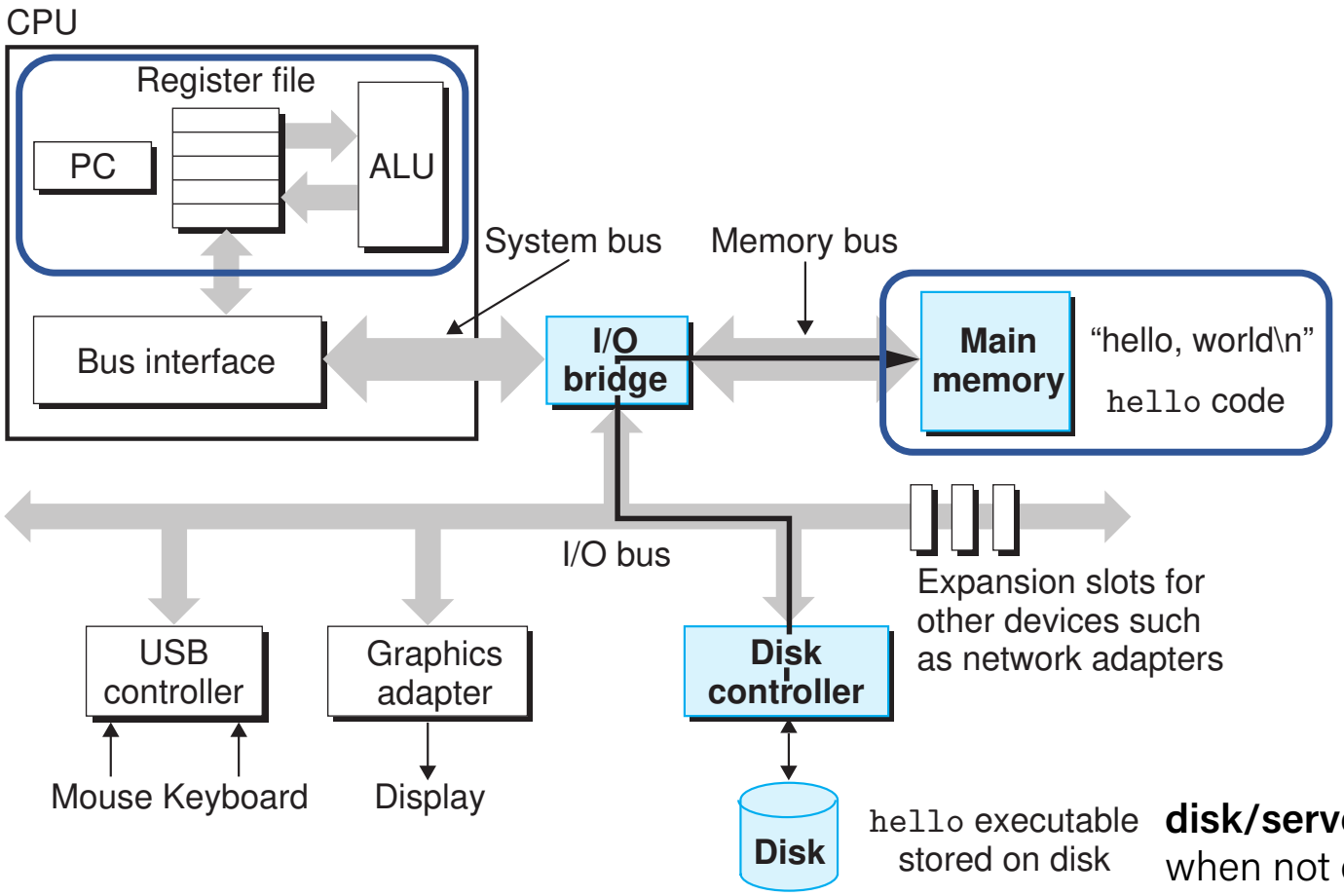
- Integer (`unsigned int`, 2's complement signed `int`)
- Floating Points (IEEE single (`float`) and double (`double`) precision
- `char` (ASCII)
- Address (unsigned long)
- Aggregates (arrays, `structs`)

The code itself is binary too!

- Instructions (machine encoding)

Computer architecture

registers accessed by name
ALU is main workhorse of CPU



memory needed for program execution (stack, heap, etc.) accessed by address

disk/server stores program when not executing

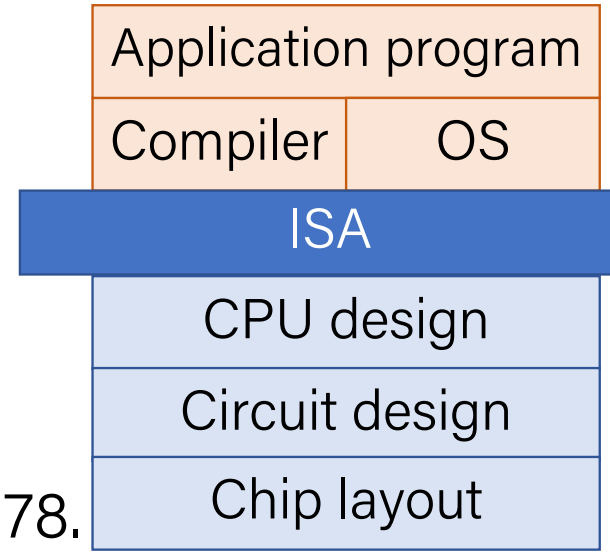
Assembly

- We are going to learn the **x86-64** instruction set architecture. This instruction set is used by Intel and AMD processors.
- There are many other instruction sets: ARM, MIPS, etc.

Instruction set architecture (ISA)

A contract between program/compiler and hardware:

- Defines operations that the processor (CPU) can execute
- Data read/write/transfer operations
- Control mechanisms



Intel originally designed their instruction set back in 1978.

- Legacy support is a huge issue for x86-64
- Originally 16-bit processor, then 32 bit, now 64 bit.
  These design choices dictated the register sizes
  (and even register/instruction names).

mov

The **mov** instruction copies bytes from one place to another;
it is similar to the assignment operator (=) in C.

```
mov     src,dst
```

The **src** and **dst** can each be one of:

- Immediate (constant value, like a number) (*only **src**)* **$0x104**
- Register **%rbx**
- Memory Location
  (*at most one of **src, dst**)* Direct address **0x6005c0**

Most General Operand Form

`Imm(`r_b`, `r_i`, s)` is equivalent to address **`Imm + R[`r_b`] + R[`r_i`]*s`**

Displacement: pos/neg constant (if missing, = 0)

Base: register (if missing, = 0)

Index: register (if missing, = 0)

Scale must be 1,2,4, or 8 (if missing, = 1)

Memory Location Syntax

Syntax	Meaning
`0x104`	Address `0x104` (no `$`)
`(%rax)`	What's in `%rax`
`4(%rax)`	What's in `%rax`, plus `4`
`(%rax, %rdx)`	Sum of what's in `%rax` and `%rdx`
`4(%rax, %rdx)`	Sum of values in `%rax` and `%rdx`, plus 4
`(, %rcx, 4)`	What's in `%rcx`, times 4 (multiplier can be 1, 2, 4, 8)
`(%rax, %rcx, 2)`	What's in `%rax`, plus 2 times what's in `%rcx`
`8(%rax, %rcx, 2)`	What's in `%rax`, plus 2 times what's in `%rcx`, plus 8

Operand Forms

Type	Form	Operand Value	Name
Immediate	$\$\mathrm{Imm}$	Imm	Immediate
Register	$\mathrm{r_a}$	$\mathrm{R[r_a]}$	Register
Memory	Imm	$\mathrm{M[Imm]}$	Absolute
Memory	$(\mathrm{r_a})$	$\mathrm{M[R[r_a]]}$	Indirect
Memory	$\mathrm{Imm(r_b)}$	$\mathrm{M[Imm + R[r_b]]}$	Base + displacement
Memory	$(\mathrm{r_b, r_i})$	$\mathrm{M[R[r_b] + R[r_i]]}$	Indexed
Memory	$\mathrm{Imm(r_b, r_i)}$	$\mathrm{M[Imm + R[r_b] + R[r_i]]}$	Indexed
Memory	$(, \mathrm{r_i}, \mathrm{s})$	$\mathrm{M[R[r_i] \cdot s]}$	Scaled indexed
Memory	$\mathrm{Imm}(, \mathrm{r_i}, \mathrm{s})$	$\mathrm{M[Imm + R[r_i] \cdot s]}$	Scaled indexed
Memory	$(\mathrm{r_b}, \mathrm{r_i}, \mathrm{s})$	$\mathrm{M[R[r_b] + R[r_i] \cdot s]}$	Scaled indexed
Memory	$\mathrm{Imm(r_b, r_i, s)}$	$\mathrm{M[Imm + R[r_b] + R[r_i] \cdot s]}$	Scaled indexed

Figure 3.3 from the book: "Operand forms. Operands can denote immediate (constant) values, register values, or values from memory. The scaling factor s must be either. 1, 2, 4, or 8."