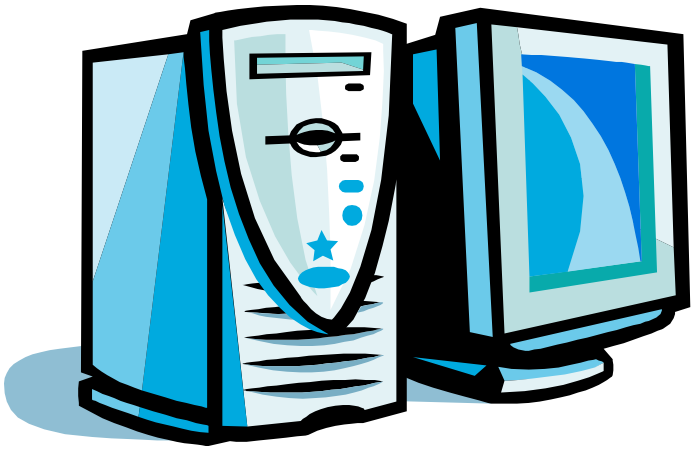
CS232

Instruction sets, RISC vs. CISC, Compilers, Assemblers, Linkers, Loaders, Memory images, and who cares about assembly.

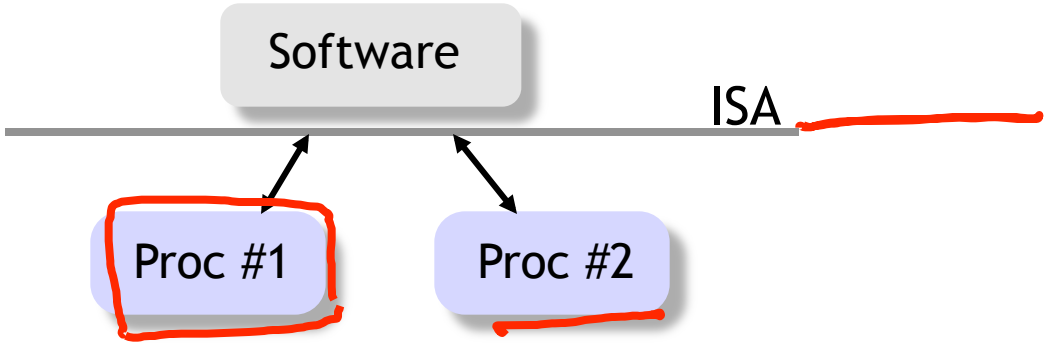


Pick up HANDOUT...

Instruction Set Architecture (ISA)

- The ISA is the interface between hardware and software.

- The ISA serves as an abstraction layer between the HW and SW
  - Software doesn't need to know how the processor is implemented
  - Any processor that implements the ISA appears equivalent



- An ISA enables processor innovation without changing software
  - This is how Intel has made billions of dollars.
- Before ISAs, software was re-written/re-compiled for each new machine.

A little ISA history

- 1964: IBM System/360, the first computer family
  - IBM wanted to sell a range of machines that ran the same software
- 1960's, 1970's: Complex Instruction Set Computer (CISC) era
  - Much assembly programming, compiler technology immature
  - Simple machine implementations
  - Complex instructions simplified programming, little impact on design
- 1980's: Reduced Instruction Set Computer (RISC) era
  - Most programming in high-level languages, mature compilers
  - Aggressive machine implementations ← fit whole computer on a chip
  - Simpler, cleaner ISA's facilitated pipelining, high clock frequencies
- 1990's: Post-RISC era
  - ISA complexity largely relegated to non-issue
  - CISC and RISC chips use same techniques (pipelining, superscalar, ..)
  - ISA compatibility outweighs any RISC advantage in general purpose
  - Embedded processors prefer RISC for lower power, cost
- 2000's: Multi-core and Multithreading

RISC vs. CISC

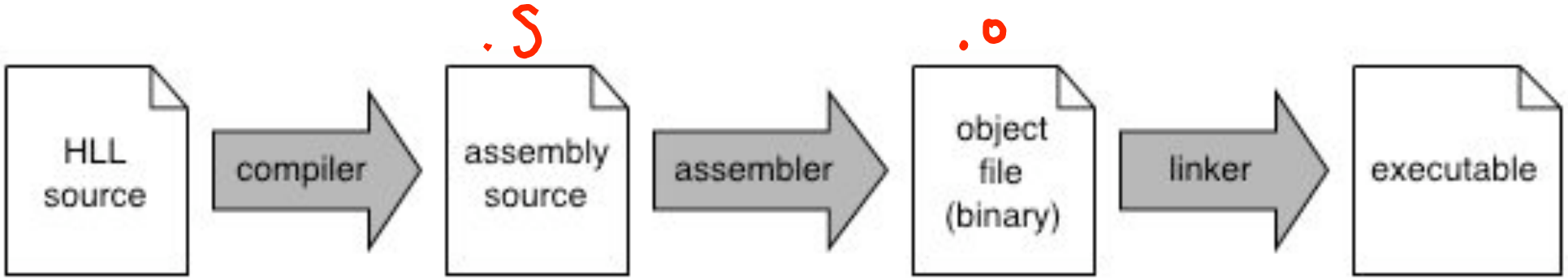
- MIPS was one of the first RISC architectures. It was started about 20 years ago by John Hennessy, one of the authors of our textbook.
- The architecture is similar to that of other RISC architectures, including Sun's SPARC, IBM and Motorola's PowerPC, and ARM-based processors.
- Older processors used complex instruction sets, or CISC architectures.
  - Many powerful instructions were supported, making the assembly language programmer's job much easier.
  - But this meant that the processor was more complex, which made the hardware designer's life harder.
- Many new processors use reduced instruction sets, or RISC architectures.
  - Only relatively simple instructions are available. But with high-level languages and compilers, the impact on programmers is minimal.
  - On the other hand, the hardware is much easier to design, optimize, and teach in classes.
- Even most current CISC processors, such as Intel 8086-based chips, are now implemented using a lot of RISC techniques.

Differences between ISA's

- Much more is similar between ISA's than different. Compare MIPS & x86:
  - Instructions:
    - same basic types
    - different names and variable-length encodings
    - x86 branches use condition codes
    - x86 supports (register + memory) -> (register) format
  - Registers:
    - Register-based architecture
    - different number and names, x86 allows partial reads/writes
  - Memory:
    - Byte addressable, 32-bit address space
    - x86 has additional addressing modes
    - x86 does not require addresses to be aligned
    - x86 has segmentation, but not used by most modern O/S's

%eax %al

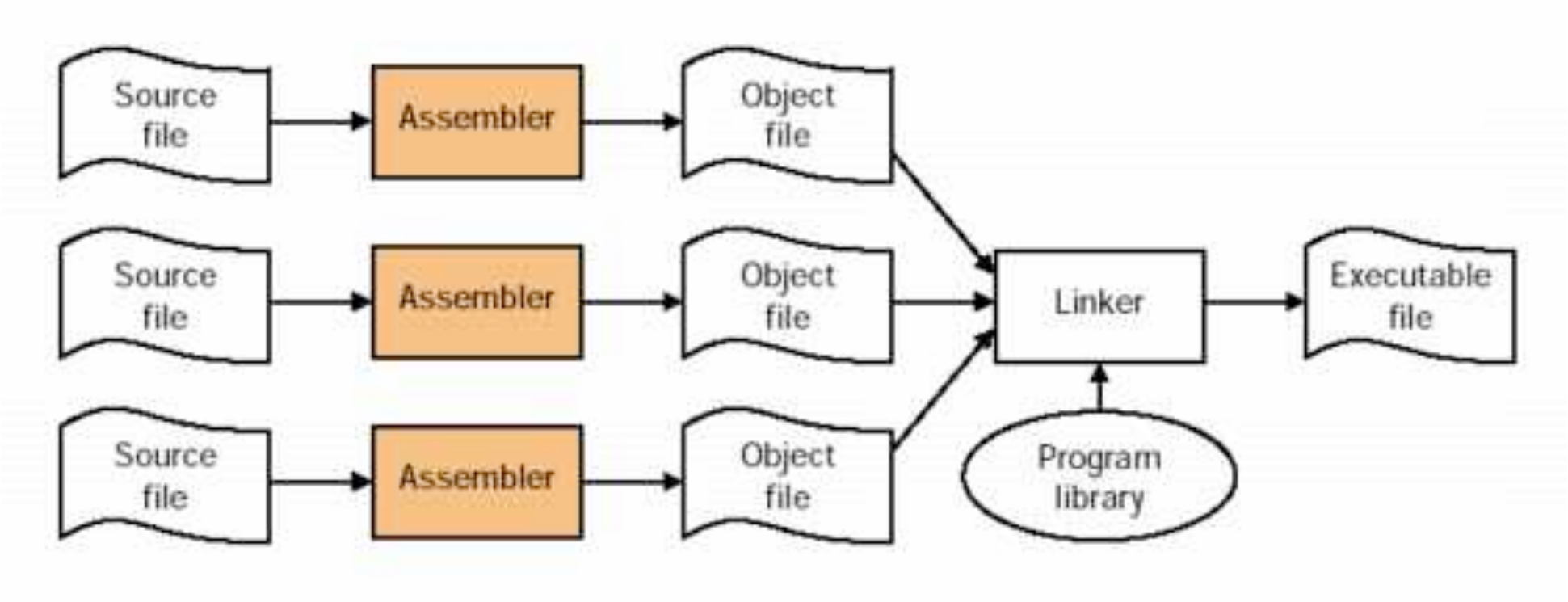
The compilation process



g++ sudoku.c

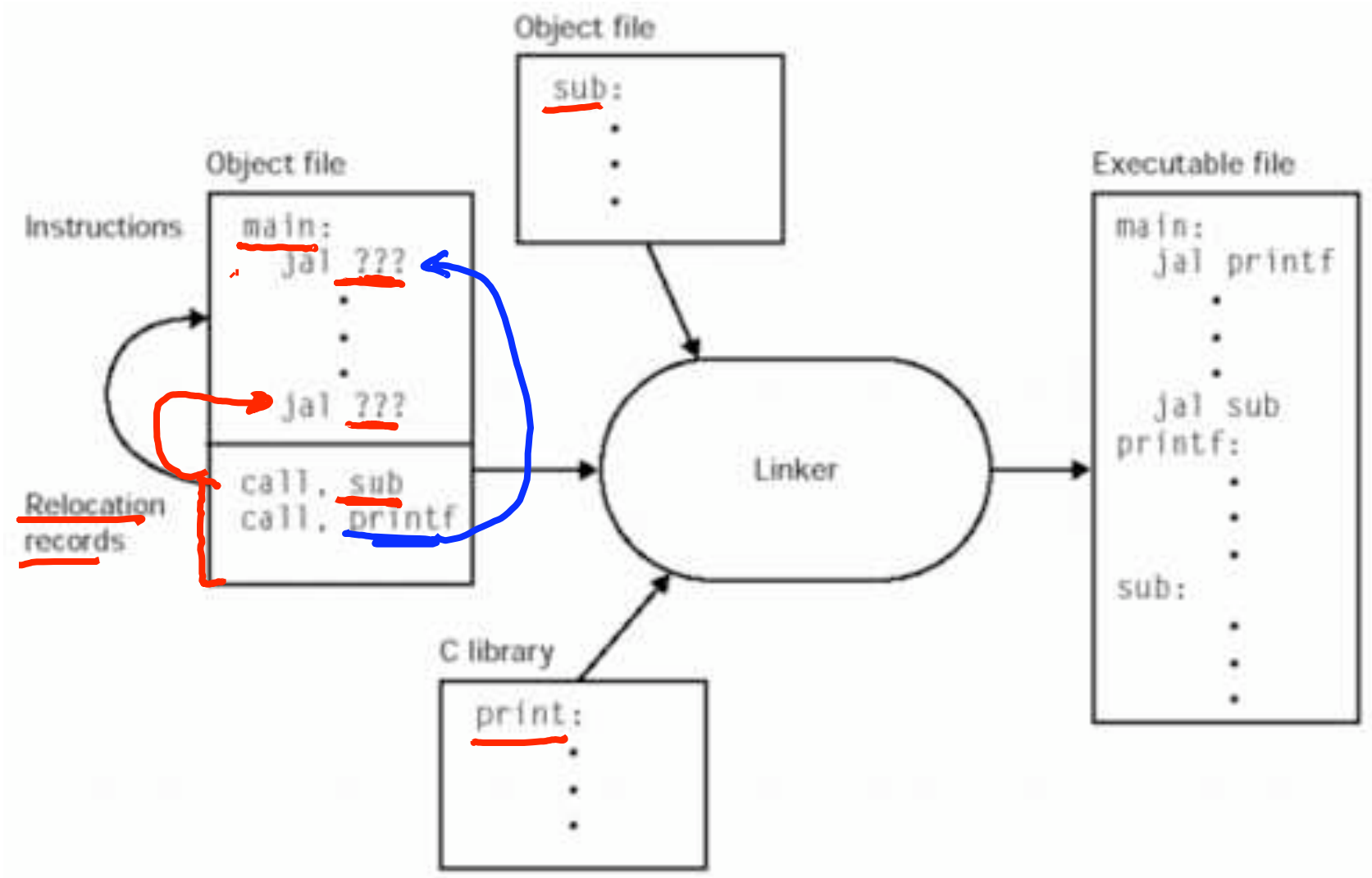
a.out

The purpose of a linker

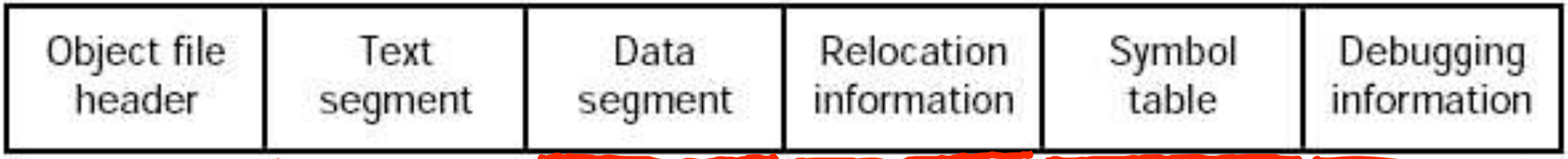


Separate compilation

What the linker does



Object File Formats



gcc -g m

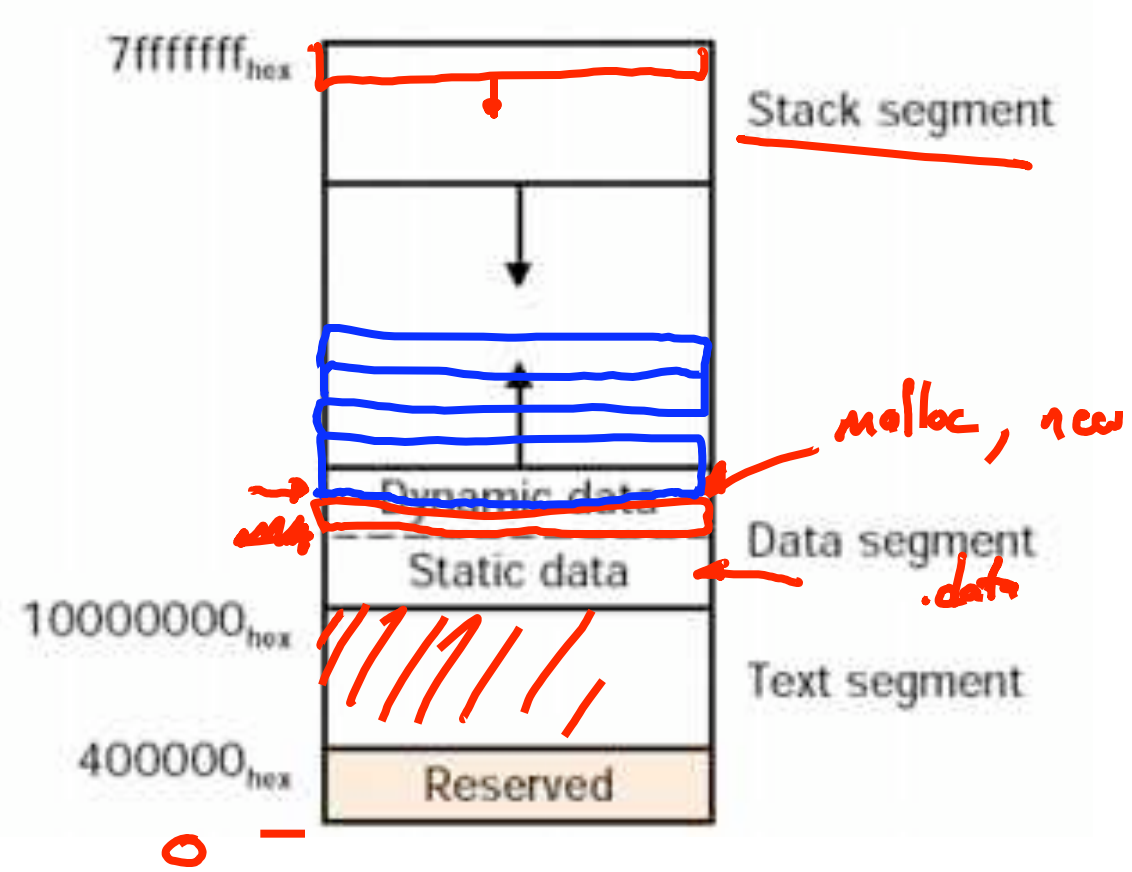
Loader

- Before we can start executing a program, the O/S must **load** it:

- Loading involves 5 steps:
  1. Allocates memory for the program's execution.
  2. Copies the text and data segments from the executable into memory.
  3. Copies program arguments (*e.g.*, command line arguments) onto the stack.
  4. Initializes registers: sets $sp to point to top of stack, clears the rest.
  5. Jumps to start routine, which: 1) copies `main`'s arguments off of the stack, and 2) jumps to `main`.

gcc -S sudoku.s

MIPS memory image



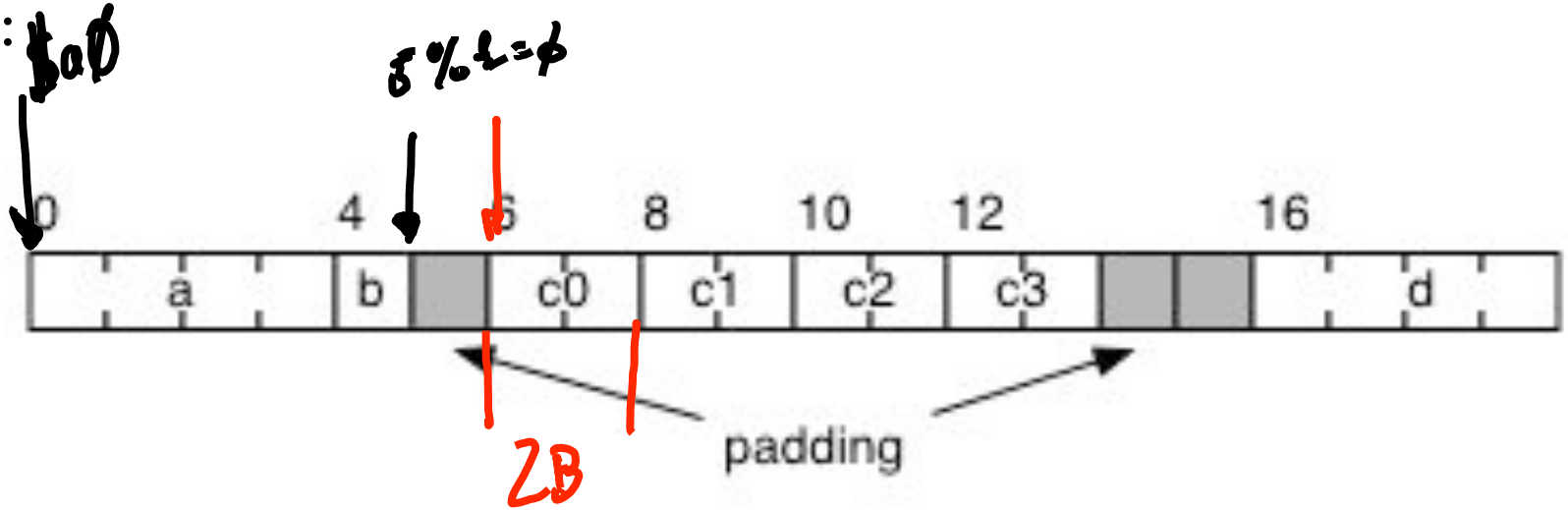
Structs

- Structs are like arrays, but the elements can be different types.
  - Same with objects

- Compiler/assembler inserts padding to "naturally align" data
  - Sometimes you can reorganize fields to eliminate padding.

- Consider:

```
struct {
  int a;
  char b;
  half c[4];
  int d;
}
```



Whither Assembly Language

ALL: productivity ←
readability
portability

90/10

ASM: fast, efficient
size
power
expressibility
} control

Inline assembly Example

```
intadd(int a, int b) { /* return a + b */
   int ret_val;
   __asm("add %2, %0, %1", a, b, ret_val);
   return(ret_val);
}
```