Version 2.11

# UCSD CSE 30 Section B

## Computer Organization and Systems Programming

Lecture 1

April 2, 2024

Keith Muller

DEC PDP 11/45 - 1973

# CSE30 Section B Spring 2024

Instructor: Keith Muller

- **I highly encourage feedback**
  - Please bring any issues to my attention, I will promptly address them
- How to contact me directly:
  - kmuller@ucsd.edu
  - Please do not use canvas messages

- In Person Office Hours: CSE 2109
  - Tue, Thu: 2:00 PM to 3:00 PM
  - These office hours are group meetings
  - Ask questions, review material, or just come to listen
  - Students who attend office hours tend to do better in the course

- Zoom Office Hours

  https://ucsd.zoom.us/j/94331007124
  - Friday: 4:00 PM to 4:45 PM
  - These office hours can be indivual of for a group if you like
  - Additional office times By Appointment
  - Send me email to schedule



x

# CSE 30 Spring 2024 – Staff Covers Both Sections A & B

Section A (Cao) and B (Muller) share the same pool of TA's and Tutors

**TA's**

- Nitya Agarwal
- Mihir Kekkar
- Yuchen Jing
- Liam Fernandez

**Tutors**

Ali Alabiad

Bryan Cho

Charlotte Dong

Vivian Liu

Kate Romero

Kevin Shen

Charvi Sukla

Fong Vachirathanusorn

Joseph Edmonston

Thanh-Nhan Lam

**Tutors**

Christian Lee

Jessie Ouyang

Brandon Reponte

Adrian Rosing

Luffy Saito

Leica Shen

Shijie Wang

Alex Simonyan

Reese Whitlock



x

# Overview of Grading - See Syllabus (Canvas) for More Details

70 pts – Attending Lecture in person
- 5 points per section B lecture

120 pts total – Canvas Quizzes

240 pts total – Programming Assignments

190 pts - Midterm – In Person

380 pts - Final – In Person

1000 pts total for graded assignments

- **Special grading circumstances** (e.g., extended absence, illness, other issues, etc.)
  - PROMPTLY Contact me directly (kmuller@ucsd.edu)



x

# Lecture 1 QR Code

- Class attendance points: To encourage you to attend lecture
  - Over the years we have found that students that attend lectures in CSE30 get better grades
- Section B has 20 lectures, attend 14 to get the 70 points
  - Attending more than 14 gets you up to 30 more points
- Attendance is taken at the start of class using google forms that is accessed with a lecture QR code in the slides
  - For the first lecture only, the form will be open until 9 PM
  - bring a device that can use QR codes to access goggle forms and allows you to sign into UCSD SSO
  - You will be required to supply a code word announced in class
  - You will eventually get an email acknowledgement from google that your attendance was recorded
- **ONLY If you cannot access the google form,** send me email (kmuller@ucsd.edu) with the code word in the subject line
  - The email must be timestamped within the first 20 minutes of lecture

If you have issues with this method, please contact me



x

# CSE30 Spring 2024 Section B Specific

- There are two sections: Section A (Cao) and **Section B (Muller)**
- **What is the same in the two sections**
  - **study topics** (roughly in-sync by the end of each week)
  - **quizzes**
  - **Programming Assignments**
- **What is different between the two sections**
  - **lecture materials**
  - **midterm questions (from Sect B lecture)**
  - **final questions (from Sect B lecture)**
- **In-person lecture attendance** is **strongly encouraged** (attendance points)
  - Lectures are podcast recorded
- **Discussion section attendance** is **optional but strongly encouraged**
  - You may attend either discussion section and still be enrolled in Sect B
  - Section B sections are podcast recorded
- **See the syllabus for grading details**



x

# CSE30 Class Resources

- **Section B Lecture Slides:** https://github.com/cse30-sp24/Muller-Slides
  - Located on class github in both pptx and pdf format
  - Slides **are updated constantly** to correct errors and to improve content
    - Version is at the upper left on the title slide
    - Always check you have the current version the morning before lecture
- **Class github:** https://github.com/cse30-sp24
- **Piazza:** https://piazza.com/ucsd/spring2024/cse30_sp24_a0/home
  - **First Place to go to** for **Q/A** and **important announcements**
  - **Public piazza posts are for:** general questions on PA's and lectures
    - **Do not post publicly** any parts of an assignment, quiz or exam solution
  - **Private posts are for:** specific situation relating to just you or you are not sure
- **Tutor Lab hour schedule:** https://autograder.ucsd.edu
  - For getting help from the tutors
- **Canvas**: https://canvas.ucsd.edu/courses/54650
  - Links to quizzes, textbooks, programming assignments, exams
- **Gradescope:** https://www.gradescope.com
  - Quizzes and Submitting programming assignments



x

# Surviving Section B Lectures (In-person)

- **Make sure you bring your copy of lecture slides to class, it helps**
- **How to get my attention in class**
  - I **never intentionally ignore questions**; I **just may not see you**
  - Raise your hand, or just **call out** if I appear to **ignore you by accident**
- **You must SLOW ME DOWN**: Otherwise, **I tend to speed up**
  - Please do not be shy, **speak up** and **remind me to slow down**
- If you have questions, or I went too fast, or the material is not clear, etc.
  - Please ask me to go over it again (do this right away, not 5 slides later)
    - Just don't sit there and waste your time
  - **my responsibility:** help you learn the material
  - **your responsibility**: ask questions (I love questions, they also slow me down!)



x

# How to do well in CSE30 - 1

- Go to lecture
  - Before lecture go over the class slides
  - Lecture slides are posted the day before class (last minute updates that morning)
  - Keep your lecture slides up to date (I update them to fix errors and address questions)
- Go to Discussion Sessions
  - ask the TA's and Tutors for help
- Studying for exams
  - All the exam question topics are found in my slides and the PA writeups
  - Try to write the exam yourself, with practice you will be able to guess the questions
- Post to piazza when you have questions
- Do the readings on time
- Review the material: watch the podcasts and occasional special topic videos



x

# How to do well in CSE30 - 2

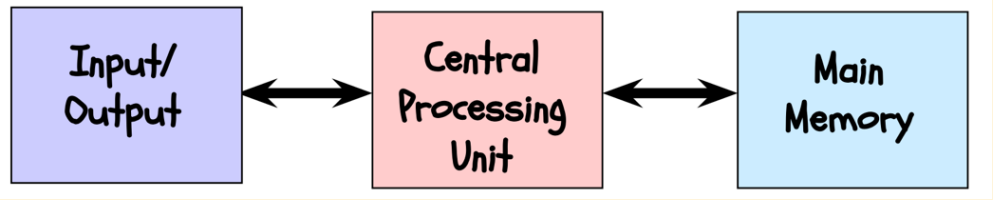
- **Most important:** Keep up, **do not procrastinate as it is hard to catch up**
  - The class material starts easy and gets much harder over the quarter
  - Do not expect you can do later programming assignments in less than 5 days
  - Do not expect to learn the material by binge watching podcasts, this never ends well
- **Please be careful when using web resources** for this class
  - a lot of the material you will find is either not correct or does not apply to our programming environment
  - this is especially true with assembly language programming topics
- **Are you struggling?**
  - **Do not wait**, **ask for help as soon as possible** – do not fall behind
  - **Best advice: Come to my office hours** (or schedule a zoom meeting)
    - Give me a chance to help you
    - I will spend as much time as necessary to help you understand the material



x

# A General-Purpose Computer – Von Neuman Architecture

- Since the middle of the 20th century, many architectural approaches to the **general-purpose computer** have been tried
- The **architecture** which **nearly all modern computers** are based was proposed by John Von Neuman in the late 1940's
- The **major components** are:




- Central Processing Unit (CPU): a device which fetches, interprets (decodes), and executes a specified set of operations called instructions
- Memory: Storage of N words of W bits, where W is a fixed architectural parameter, and N can can be expanded to meet **workload** (the programs running on the CPU) and **cost requirements**
- I/O: Devices for communication with the outside world (including external persistent storage)
  - External connections (from CPU to memory and I/O) typically use industry **"standards"**
  - **Standards** enable technologies from **different companies to interoperate**



x

# What is Computer Architecture?

## Instruction Set Architecture (ISA)

- Functional behavior of a computer system as viewed by a programmer
  - describes how the CPU is controlled by software programs
  - specifies both what the processor can do as well as how it gets it done
- Architectural Characteristics (partial list):
  - supported data types (how data is encoded)
  - CPU registers (number, size, use, etc.)
  - how the hardware manages main memory
  - instructions a microprocessor can execute
    - Operations they perform
    - Instruction "format" (bit patterns) in memory
  - input/output model

## Machine Organization

- Physical (design) realization of what is specified by the instruction set architecture
- It deals with how the hardware components are linked together to meet the requirements specified by instruction set architecture
  - The ISA allows variability in the physical design implementation to address different workload needs (cost, scalability, etc.)
- Machine organizational characteristics (partial)
  - Hardware component choices to achieve:
    - Expandability
    - Configurability
    - etc.
  - Physical layout
  - Number and type of peripherals (I/O devices)



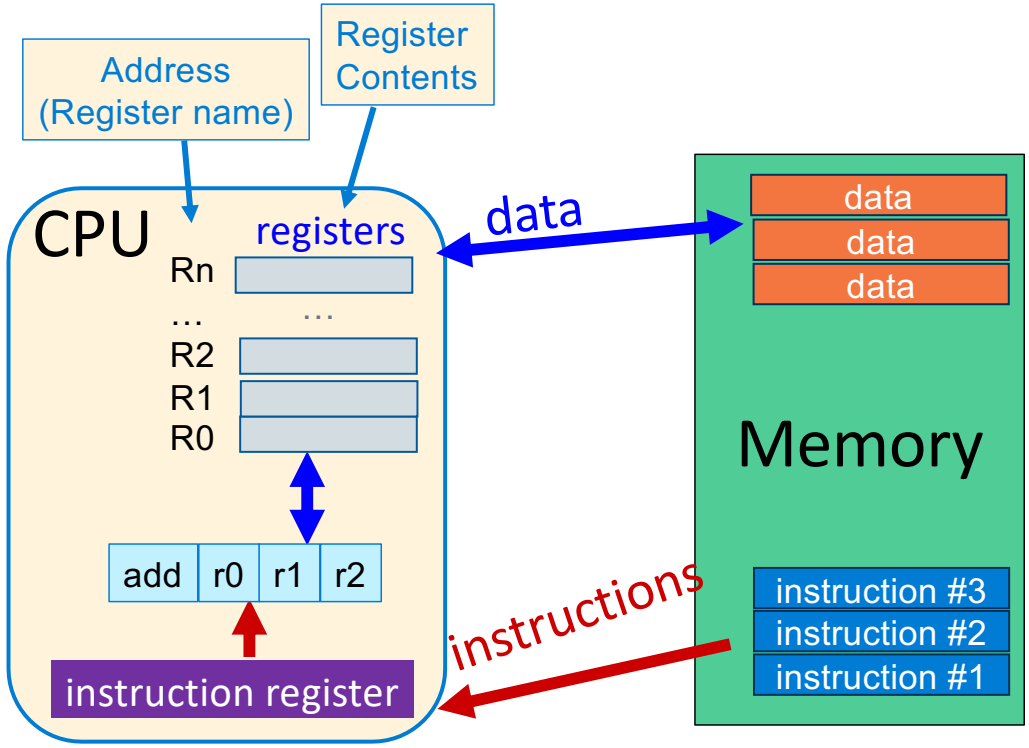
x

# Von Neuman Architecture

- **Distinguishing feature**: Memory contains **both** program CPU instructions and data
- **CPU Instructions** are encoded in memory with patterns of ones and zeros (similar to binary numbers)
  - **Encoded CPU instructions** are called **machine code (or machine language)**
- **Example**: three 32-bit instructions (shown in hexadecimal format below)

```
81 fe 89 32
81 54 22 af
81 22 10 9A
```

- **Instructions** operate on **data** that is stored in a small capacity volatile (fast) memory in the CPU
  - This memory is called registers
- CPU **reads/writes data** from **memory** from these **data registers** to **operate on the data**






x

# C, Assembly and Executable Programs

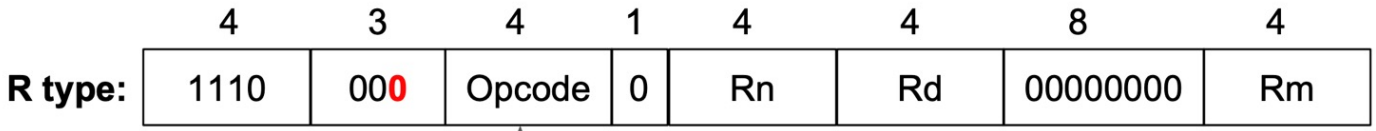
- **Assembly language** is a symbolic version of the **machine code (language)**
  - **Instructions** describe operations the hardware can perform (e.g., =, +, -, *)
  - **Unique to a specific ISA**: e.g., ARM-32 versus IA-64
  - May be stored in a human readable text file
  - You can write in assembly language just like C or Java
    - Assembly is much easier to program than machine code
- A **high-level language** (like C) is **compiled** into an **assembly language equivalent**
  - A statement in C is represented by a sequence of one or more assembly language instructions (why a do you think it is a sequence?)
- **Assembly language program**
  - assembly language program is **translated (assembled)** into **machine code**
- An **executable program** contains
  - **series of instructions in machine code** (the program)
  - (maybe some) **data** to operate on



x

# Assembly & Machine Code Example: ARM-32 (32-bits)

Consider an addition statement
R0 = R1 + R3;




Simple R-type instructions follow the following template:

OP Rd, Rn, Rm

0000 - AND
0001 - EOR
0010 - SUB
0011 - RSB
0100 - ADD
0101 - ADC
0110 - SBC
0111 - RSC
1000 - TST
1001 - TEQ
1010 - CMP
1011 - CMN
1100 - ORR
1101 - MOV
1110 - BIC
1111 - MVN

Assembly Language (human readable)

ADD R0, R1, R3

machine code pattern in memory

1110 0000 1000 0001 0000 0000 0000 0011

**List of Different operations for this type of instruction**

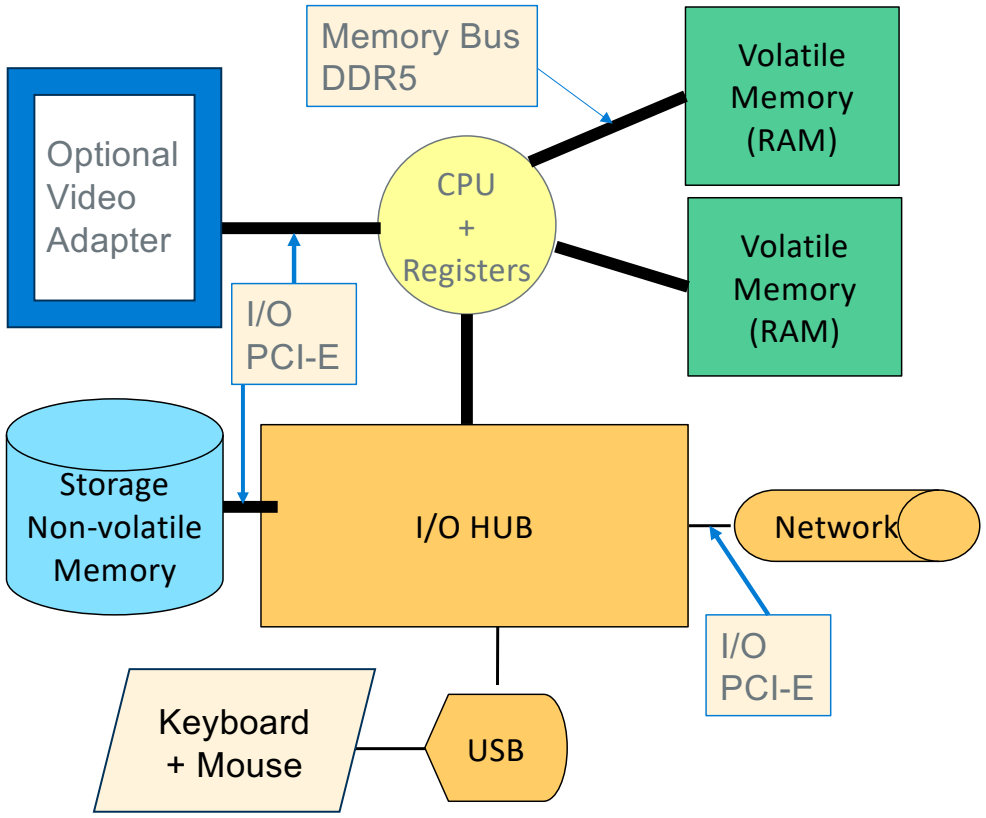


x

# Review: Machine Organization – Von Neuman

1. CPU executes a machine code program
   - Machine code is specific to a particular CPU Instruction set Architecture (ISA)
2. **Memory** contains **both data and programs**
3. I/O (input/Output): Connects the CPU and memory to the external world
   - An I/O operation is where data (including machine code) is copied between persistent storage (like an SSD) and ram memory

- Volatile (non-persistent) memory
  - contents lost when power is removed
  - Memory dimms (memory bus)
  - CPU registers (memory inside the CPU)
- Non-volatile (persistent) memory
  - contents preserved when power is removed
  - SSD (I/O bus attached)
  - NVDIMM (memory bus attached)



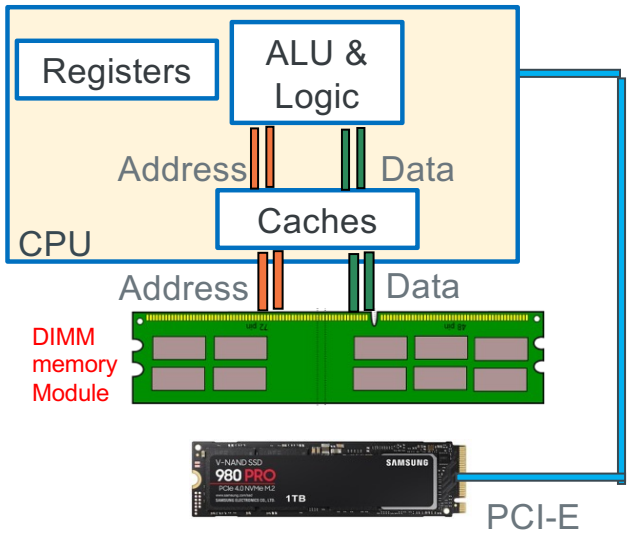


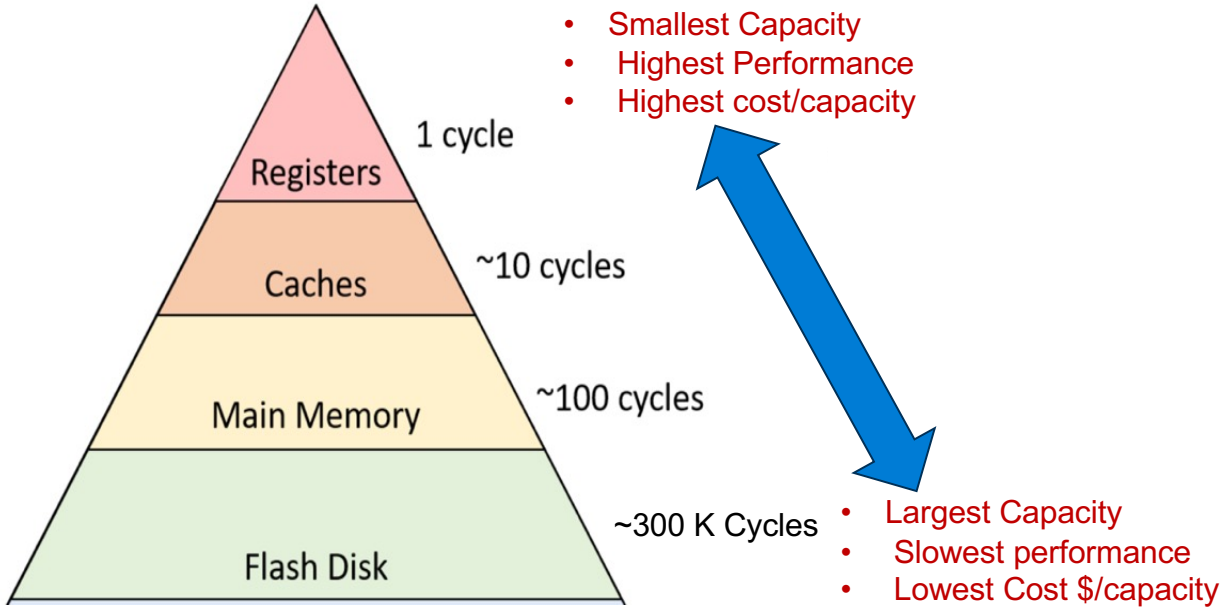

x

# Memory Triangle: Hardware Cost/Performance/Capacity Tiers

Assume each instruction takes 1 clock cycle

Clock cycle =~ time to access; larger is slower



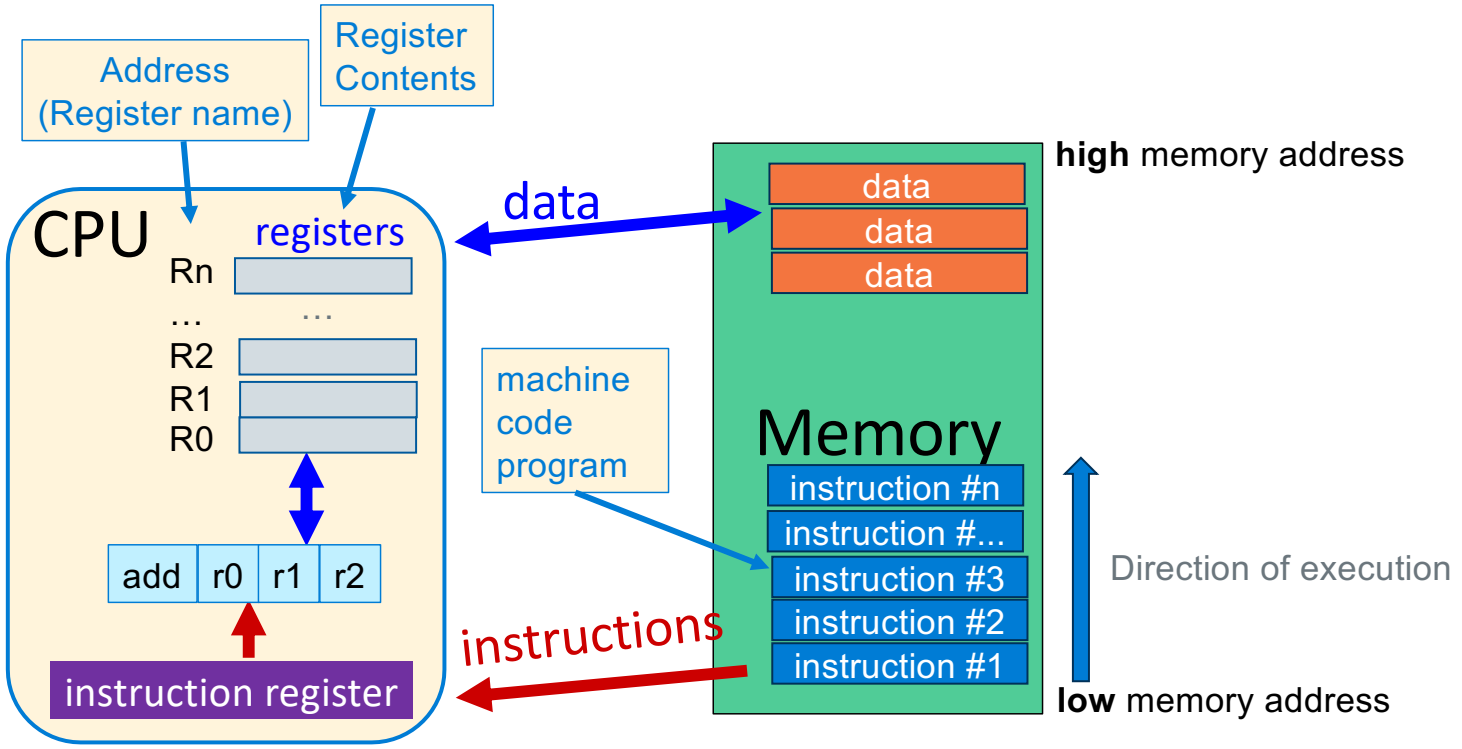




**Design Goal:** best performance at the lowest (or specific) cost
**Other goals:** performance/energy (operating cost), expandability, high margin (price/cost)



x

# Machine code execution order

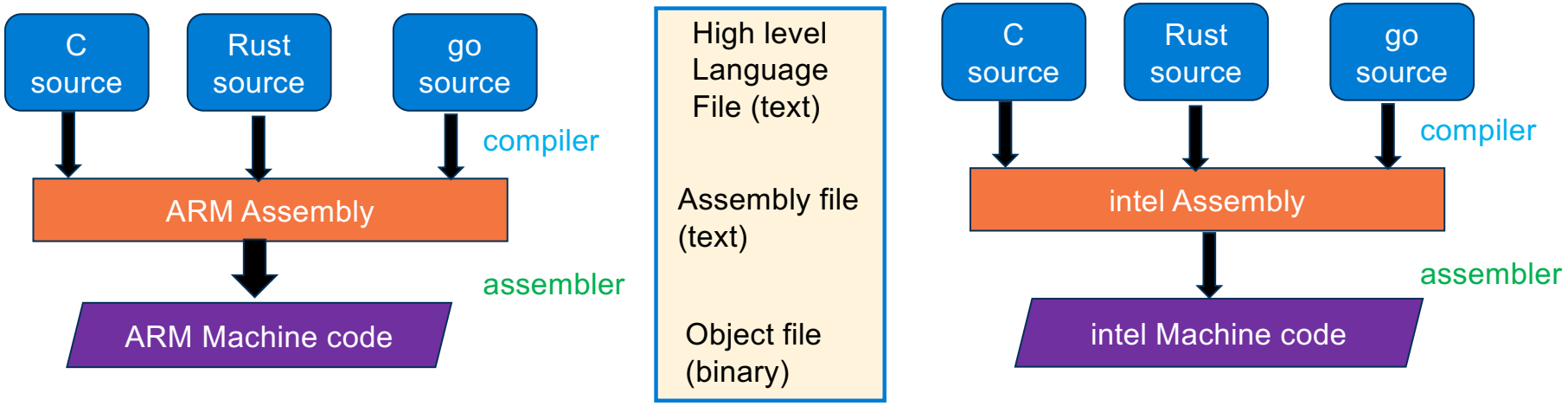



- **Execution order**: Programs execute from instructions located in **low address** memory to **high address** memory stepping **one machine instruction at a time** (called **execution order**) **unless there is a branch** (example: loop, if statement etc.)



x

# From Source to Machine code

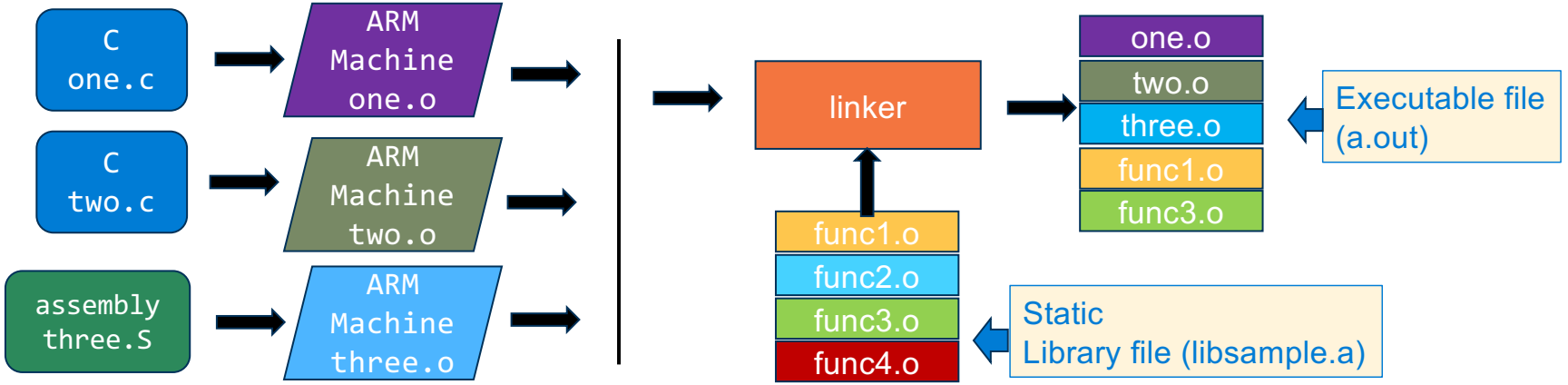



- The **granularity** of **compilation and assembly** is a **single text file** (called a **translation unit**)
  - **.c** file is a C **source** file (file.c)
  - **.S** file (upper case S) is a **human** written **assembly source** file (file.S)
  - **.s** file (lower case s) is a **compiler** generated **assemble source** file (file.s)
  - **.o** file is a **machine code binary (object)** file (file.o)
- Multiple **.o** files are **combined** (linked) into an **executable file**



x

# Linker: Combines object files to create an executable file

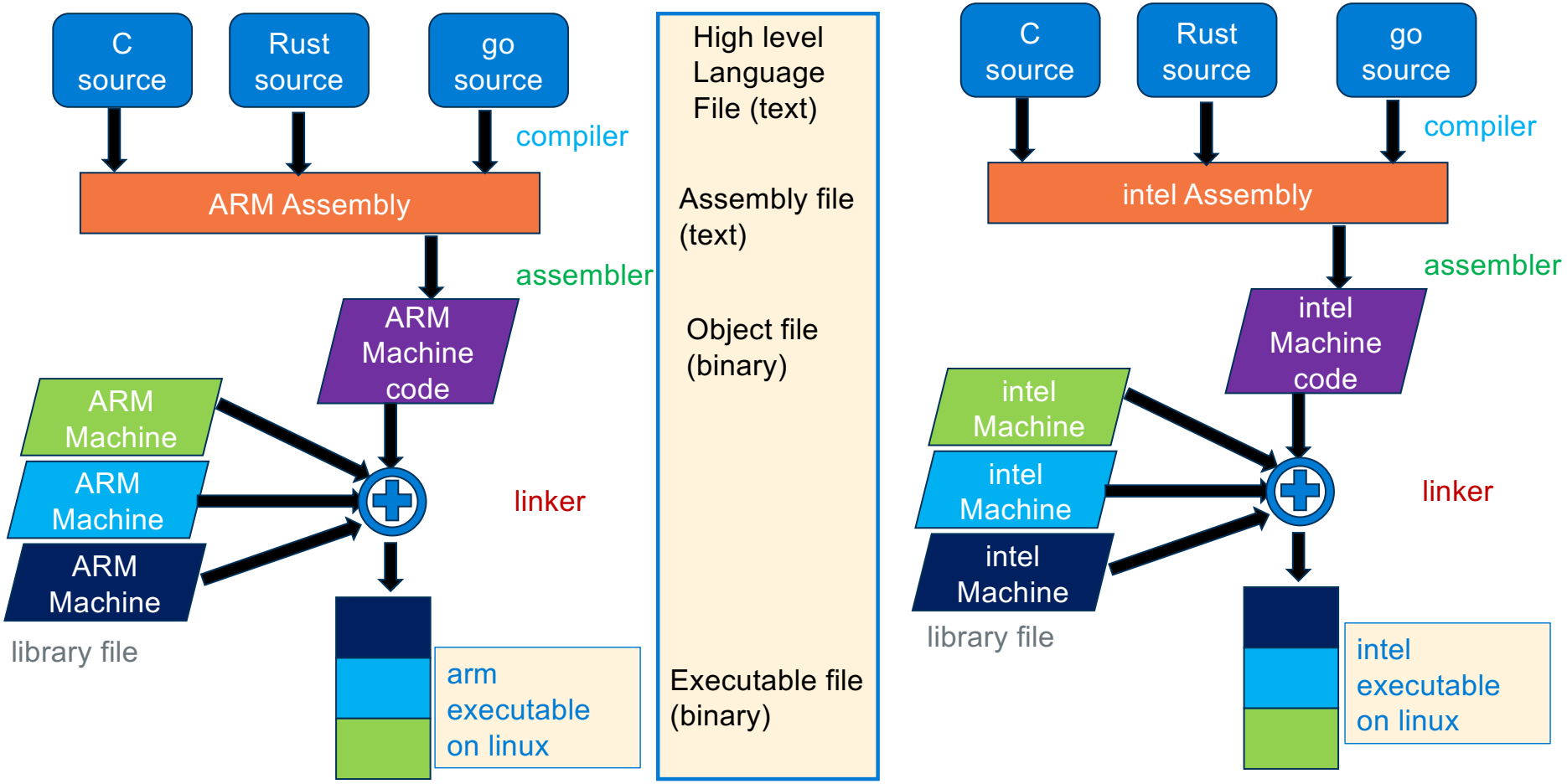



- Each source file (**Translation unit)** is compiled (or assembled) independently to an object file
  - When we modify a single file in a multi-source file program, we want to only **recompile** the **file** that changed and combine it with the other already compiled object files
- **Library file** (libXX.a – where XX is the library name) is an **aggregation** of **distinct object** (.o) files
- Linker combines all the listed object files together plus **just those object files in libraries** whose **contents are referenced**
  - **Example**: one.c and two.c call functions contained in func1.o and func3.o (no calls to func2.o or func4.o)
- Important: **Object files created from C and assembly source can be linked** (call each other) into a working executable when certain rules are followed (we will be doing a lot of this later this quarter)



x

# From Source to Execution: Different ISA




Pi-cluster system (all CSE30 PA's)

ieng6



x

# From Source code to Execution

```
$ cat test.c
#include <stdlib.h>
#include <stdio.h>
int main (void)
{
    printf("Hello!\n");
    return EXIT_SUCCESS;
}
$ gcc -Wall -Wextra -Werror -c -S test.c
$ ls -ls
total 8
4 -rw-r--r-- 1 kmuller kmuller 109 Mar 14 15:57 test.c
4 -rw-r--r-- 1 kmuller kmuller 725 Mar 14 15:58 test.s
$ gcc test.s
$ ls -ls
total 16
8 -rwxr-xr-x 1 kmuller kmuller 7708 Mar 14 15:58 a.out
4 -rw-r--r-- 1 kmuller kmuller  109 Mar 14 15:57 test.c
4 -rw-r--r-- 1 kmuller kmuller  725 Mar 14 15:58 test.s
$ ./a.out
Hello!
```

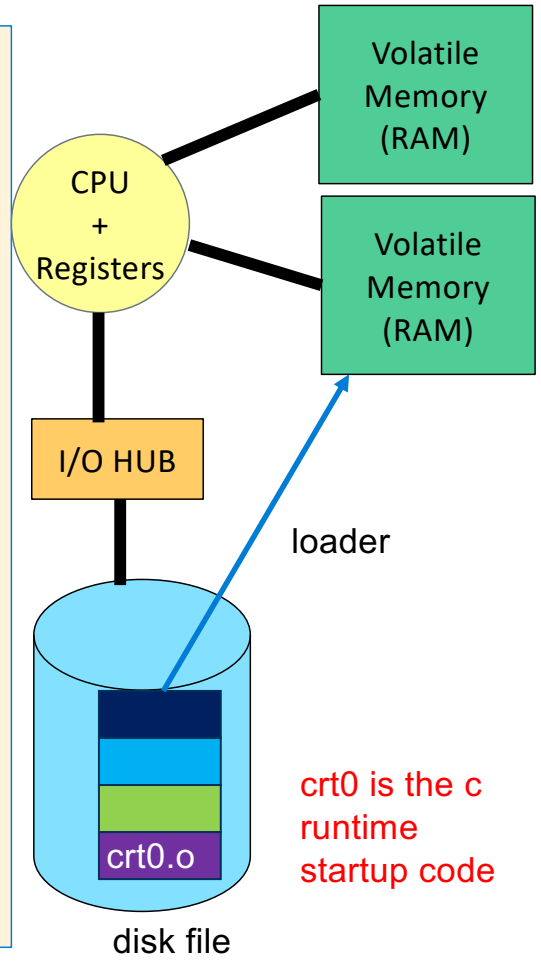
**Source to Execution Steps**

1. **Compile** (c source to assembler)
2. **Assemble** (assembler source to object)
3. **Link** (Combine object files to executable)
4. **Load** (Copy executable from into memory)
5. **Execute** (OS runs the code)

compile

assemble and link
gcc automatically calls the assembler with .S or .s files

load and then execute






x

# Equivalent Code: C -> Assembly -> Machine

C source

```
#include <stdlib.h>
#include <stdio.h>
int main(void)
{
    printf("Hello!\n");
    return EXIT_SUCCESS;
}
```

ARM-32 assembly

```
        .section .rodata
mesg:   .string "Hello!\n"
        .text
        .global main
        .type   main, %function
        .equ    FP_OFF, 4
        .equ    EXIT_SUCCESS,   0
main:   push    {fp, lr}
        add     fp, sp, FP_OFF
        ldr     r0, L1
        bl      printf
        mov     r0, EXIT_SUCCESS
        sub     sp, fp, FP_OFF
        pop     {fp, lr}
        bx      lr
L1:     .word mesg
```

address of mesg

Code aka TEXT

Data

memory address

high low bytes contents

corresponding assembly

```
00010408 <main>:
   10408: e92d4800       push {fp, lr}
   1040c: e28db004       add fp, sp, 4
   10410: e59f0010       ldr r0, [pc, 16]
//10428 <L1>
   10414: ebffffb3       bl 102e8 <printf@plt>
   10418: e3a00000       mov r0, 0
   1041c: e24bd004       sub sp, fp, 4
   10420: e8bd4800       pop {fp, lr}
   10424: e12fff1e       bx lr

00010428 <L1>:
   10428: 0001049c

0001049c <mesg>:
   1049c: 6c6c6548       // 'l', 'l', 'e', 'h'
   104a0: 000a216f       // '\0', '\n', '!', 'o'
```

Machine instructions

address of mesg



x