# MEMOS-1

CS552 Operating Systems

04/10/2023

Anton Njavro

# Agenda for today

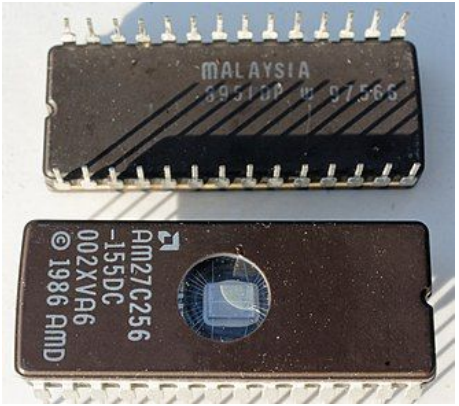
- Understand BIOS and booting process on BIOS firmware.
- Overview of MEMOS-1 assignment.
- X86 Real-mode programming

# *Ab ovo...*

- What is the point of booting anyways?
- We must bring machine to some known state, allowing us to run our OS.
- Hardware checks.
- Operating system must find its way from storage device to RAM/CPU.

# Basic Input Output System (BIOS)

- First appeared in 1975.
- Exists as **firmware** on the edge between HW and SW.
- Its purpose is to serve as a layer between HW/SW, allowing HW to diversify without hassling SW.
- Shortcomings might be some performance bottlenecks (e.g. video) where direct HW access might be optimal.
- CPU usually preprogrammed to go to FFFF0h
- That location just contains jump direction to actual BIOS boot program.




AMD BIOS Chips

# BIOS Interrupt Service Routines

- Used in Real-Mode by loading AH (AX/EAX) registers with specific value, and calling an interrupt.
- Due to lack of standardization rigor in the industry, number of function varies.
- Each BIOS function has specific set of “result” registers, other registers should stay unaffected.
- BIOS functions should never crash, so make sure to check their error returns manually.
- Once in protected mode it’s hard to access BIOS functions. We have to use Virtual 8086 mode or switch back to Real Mode (both are cumbersome).

- INT 0x10 = Video display functions (including VESA/VBE)
- INT 0x13 = mass storage (disk, floppy) access
- INT 0x15 = memory size functions
- INT 0x16 = keyboard functions

# How does BIOS boot?

- **P**ower **O**n **S**elf **T**est: First thing is to check HW and make sure it is working properly.
- Iterate through boot devices looking for magic signature, once found load MBR at 0x7C00.
- Transfer execution to MBR code.
- We are currently in **Real Mode**.
- HDD has only 446 bytes left to load the kernel. Floppy has all 510.
- GCC generates protected code executable **only.**

# Our current environment? x86 Real Mode

- Relic from the past, left for compatibility reasons.
- Only 1MB of memory is “addressable”, with no HW protection.
- Default CPU operand length is 16 bits.
- Accessing more than 64K requires cumbersome usage of segment registers.
- Faster memory access, and access to BIOS functions.

# Real Mode memory segmentation

- Memory access is done via **Segmentation** by using **segment:offset**.
- Size of a segment can range from 1 up to 65,536 bytes (using 16 bits for offset).
- Segment registers contain **16-bit segment selector**, which are most significant 16-bits of 20-bit **segment address**.
- Segment address is added to 16-bit **offset** in instruction to create **linear address** (same as physical here).

| | | | |
|---|---|---|---|
| | 0000 0110 1110 1111 0000 | **Segment** | 16 bits, shifted 4 bits left (or multiplied by 0x10) |
| + | 0001 0010 0011 0100 | **Offset** | 16 bits |
| | | | |
| | 0000 1000 0001 0010 0100 | **Address** | 20 bits |

Wikiepdia

# ELF Format

- **E**xecutable and **L**inkable **F**ormat
- Organizes code and data into different sections:
  - **.text:** machine instructions
  - **.data:** global tables and variables
  - **.rodata:** constant data (e.g strings…)
  - **.bss:** uninitialized data
- Each section can be loaded at a different location in memory.
- Specifies entry point.
- **Flat binary contains no metadata**: just mixed bag of code and data
  - Order: By the order of your code and as specified by **linker script**.
  - BIOS requires it.

# Linker Scripts

- Linker collects compiled object files, resolves references to symbols to memory addresses and builds binary executable/linkable file.
- Linker script **guides** linker to:
  - Desired **OUTPUT_FORMAT** (in our case flat binary).
  - Target system architecture: **OUTPUT_ARCH.**
  - Entry point to executable (**ENTRY**).
  - Sections of output file (**SECTIONS**):
    - Where to put sections in main memory.
    - Where to get sections from (which section of which object file).