

MEMOS-1

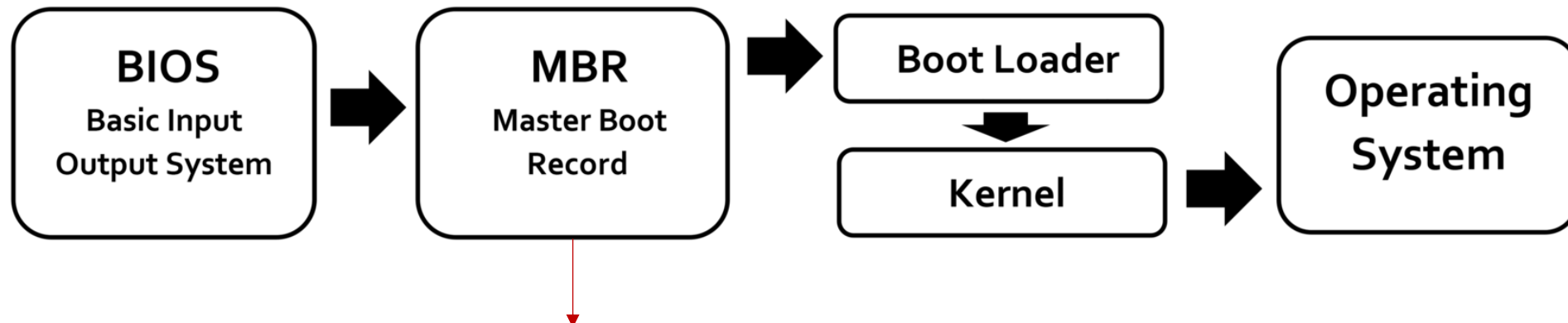
Timothy Borunov (Most slides by Shriram Raja)
Ph.D. Student, Computer Science

BIOS

- Basic Input/Output System
- Part of firmware (comes with the processor)
- Initializes the system during bootup and provides runtime services
- Usually, performs power on self test (POST) and then loads the bootloader which then loads the kernel

BIOS

Legacy Boot



Tells how the disc's sectors (aka "blocks") are divided into partitions, each partition notionally containing a file system and contains the bootloader

BIOS Functions

- Several functions are available [\[4\]](#):
 - Video display access
 - Mass storage access
 - Keyboard functions
- Usually accessed by setting register A (AH, AX, or EAX) and calling an interrupt

```
# set video mode  
    movw $0x0003, %ax  
    int $0x10
```

Real Mode

- First x86 mode design: operates on 16-bits
- Can access 32-bit registers by including the "Operand Size Override Prefix" (0x66) to the instruction [done by the assembler]
- Still the first mode for compatibility
- Has several cons:
 - No hardware memory protection (GDT)
 - Restricted addressing and operand length - only 1 MB is accessible

Protected Mode

- Main operating mode of x86 processors
- Before switching to Protected Mode:
 - Disable interrupts, including NMI (as suggested by Intel Developers Manual)
 - Enable the A20 Line
 - Load the Global Descriptor Table with segment descriptors suitable for code, data, and stack
- Cannot access BIOS functions

MEMOS-1

- No disk.img needed for this part
- An assembly file that basically runs on “bare metal” in QEMU - vga16.s in assignment page
 - Modern compilers only generate protected mode executables from C; that is why we need assembly
- This file:
 - first, detects memory: using BIOS functions (INT 0x15 - there’s a whole page on osdev for this, link in assignment)
 - then, prints the detected memory regions (INT 0x10, [Wikipedia](#) gives a list of commands, also vga16.s)

Real Mode Programming

- Control flow statements (if-else, while) are different: conditions are baked into the instruction, or the control variable is a predetermined register (JNE, LOOP)
- Segment registers: CS, DS, ES, FS, GS, and SS
- Physical address = (Segment * 16) + Offset; e.g.: %es:(%di)
- Stack must be initialized (convention for various reasons to first write to an intermediate register, here AX)

```
_start:
    movw $0x9000, %ax
    movw %ax, %ss
    xorw %sp, %sp
```

Executable and Linkable Format

- Standard format for executables in Unix-like systems
- Defines different sections of the executable such as *.text* for code, *.data* for global tables and variables
- These segment definitions help the loader place them in the appropriate regions of memory
- Linker script collects object files and compiles into exec/linkable files

```
OUTPUT_FORMAT("elf32-i386")
OUTPUT_ARCH(i386)
ENTRY(_start)
SECTIONS
{
    .foo : { *(.*) }
}
```

Debugging in Qemu

- You can start Qemu with it set to wait for you to attach a debugger

```
$qemu-system-i386 -S -s -had memos-1 &
```

- The above command uses -S to stop QEMU from running until you remote control it from within GDB. The -s option is equivalent to -gdb tcp::1234. You can then start GDB:

```
$gdb memos-1  
(gdb) target remote localhost:1234  
(gdb) ...enter debugging commands here, or type 'continue' to resume execution  
of your debugged system...
```

- If you compile source file with -g, you will be able to list source code and set breakpoints by symbols (This is possible for memos-2, but for memos-1, you will need a workaround or just set breakpoints via addresses)

References

1. [Real Mode](#)
2. [Master Boot Record](#)
3. BIOS - [Wiki](#), [osdev](#)
4. [Ralf Brown's Interrupt List](#)
5. [ELF](#)
6. [Felix Cloutier x86 Reference](#)