[org 0x7C00]; bios maps us here

The ‘org’ directive in assembly language is used to set the starting address, for the subsequent address. It tells the assembler to assume that the address following the ‘org’ directive starts in memory.

BIOS of any firmware will load the bootloader in the specific location in memory 0x7C00 it is the standard location process for x86 architecture where the BIOS loads the bootloader in memory since the last stage of BIOS is to Loading the Boot Sector at the address, we can say that different computer architectures have their own boot processes, other architectures have their own standard addresses for the initial boot code. Here's a look at typical boot addresses for ARM, RISC-V, SPARC, and other architectures:

* **ARM Architecture:**

I can’t say there a particularly initial boot code for ARM systems as the initial boot address can vary depending on the specific SoC and its configuration however in common:

1. ‘0x00000000’: Some ARM processors start executing from address.
2. ‘0xFFFF0000’: Others start at the top of the memory space.
3. On-Chip Boot ROM: Many ARMs include an on-chip boot ROM that starts execution at a ‘specific address’ determined by the chip design. This boot ROM initializes the system and jumps to a configurable address to continue the boot process (often to load an FSBL from external storage).

* **RISC-V Architecture:**

The initial boot address is also flexible and can depend on the specific implementation. Common initial boot addresses:

1. ‘0x00001000: A common boot address for RISC-V implementations, especially for systems with a ROM at this address.
2. ‘0x80000000: Another common address used in many RISC-V systems, particularly for boot code in external memory.

* **SPARC Architecture:**

SPARC systems often use OpenBoot PROM (OBP) firmware (‘0xFFF00000’: start executing firmware from this address), which starts the boot process (The exact starting address can vary based on the system design).

Finally, each architecture has its own conventions for boot addresses, which are chosen based on the system design and typical use cases.

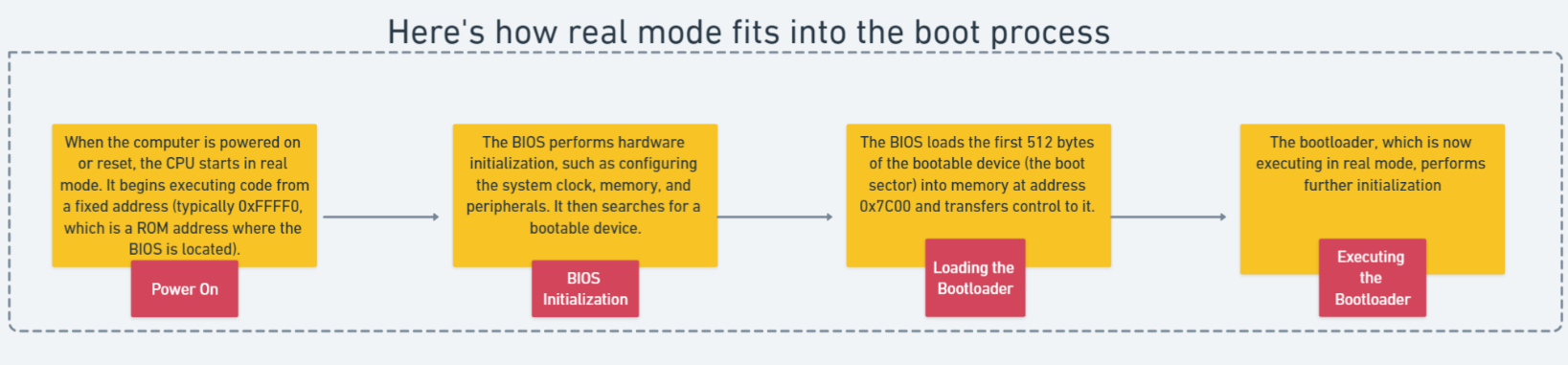
[bits 16]; we start in real mode

**What’s meant by the real mode and why in particular 16 bit?**

In brief, real mode is the initial operating mode of x86-compatible CPUs, which represent the behavior of the Intel 8086 processor. This is the mode the CPU starts in immediately after a reset (or power on).

**16-bit:** Real mode operates in 16-bit mode, meaning the CPU uses 16-bit registers for its operations. This limits the size of the data it can process in one operation to 16 bits, make sense right?

**Why Use Real Mode?**

The BIOS (Basic Input/Output System) operates in real mode. When the computer is powered on, the BIOS initializes the hardware and loads the first sector of the bootable device (like a hard disk or USB drive) into memory at address 0x7C00. The CPU starts executing this code in **real mode**.

%define CENTRY 0x7E00

%define MEMMAP 0x9000

%define PMAPL4 0xA000

‘CENTRY’, ‘MEMMAP’, and ‘PMAPL4’ are memory addresses defined for use later in the code.

**BIOS Parameter Block (BPB)**

A data structure used in the FAT file system (file system format and similar file systems) that provides information about the layout and format of a disk (refers to the storage device from which the computer is attempting to boot: floppy disk) volume. It is located at the beginning of the boot sector of a volume and contains various fields that describe the disk geometry, file system structure, and other metadata. The BPB is essential for the BIOS and operating systems to properly access (This data is not executed as code).

**Field Breakdown**

* **OEMname**: "mkfs.fat" - Indicates the formatting tool used.
* **bytesPerSector**: 512 - Each sector is 512 bytes.
* **sectPerCluster**: 1 - Each cluster contains 1 sector.
* **reservedSectors**: 1 - The first sector is reserved for the boot sector.
* **numFAT**: 2 - There are two copies of the File Allocation Table (FAT).
* **numRootDirEntries**: 224 - Maximum number of entries in the root directory.
* **numSectors**: 2880 - Total number of sectors on the floppy disk.
* **mediaType**: 0xf0 - Media descriptor for a floppy disk.
* **numFATsectors**: 9 - Each FAT occupies 9 sectors.
* **sectorsPerTrack**: 18 - Number of sectors per track.
* **numHeads**: 2 - Number of heads (sides) on the disk.
* **numHiddenSectors**: 0 - No hidden sectors.
* **numSectorsHuge**: 0 - For large volumes, gives the total sector count.
* **driveNum**: 0 - Drive number (usually 0 for the first floppy or hard disk).
* **reserved**: 0 - Reserved byte.
* **signature**: 0x29 - Extended boot signature.
* **volumeID**: 0x2d7e5a1a - Volume serial number.
* **volumeLabel**: "NO NAME " - Volume label.
* **fileSysType**: "FAT12 " - File system type (Uses 12 bits per entry, suitable for smaller capacity disks (up to 16 MB))

boot:

jmp start TIMES 3-($-$$) DB 0x90

OEMname: db "mkfs.fat" bytesPerSector: dw 512

sectPerCluster: db 1 reservedSectors: dw 1

numFAT: db 2 numRootDirEntries: dw 224

numSectors: dw 2880 mediaType: db 0xf0

numFATsectors: dw 9 sectorsPerTrack: dw 18

numHeads: dw 2 numHiddenSectors: dd 0

numSectorsHuge: dd 0 driveNum: db 0

reserved: db 0 signature: db 0x29

volumeID: dd 0x2d7e5a1a volumeLabel: db "NO NAME "

fileSysType: db "FAT12 "

**Start of the Bootloader Code**

start:

; enable a20 line via BIOS

; TODO: more compatibility by trying different a20 enabling methods

mov ax, 0x2403

int 15h

jmp 0x0000:init ; far jump to reload cs

The instruction sequence mov ax, 0x2403 followed by int 15h is used to invoke BIOS interrupt services related to enabling the A20 line.

**Purpose: Enabling the A20 Line:** The A20 line is a signal used in x86-compatible computers to address more than 1 MiB of memory (1,048,576 bytes) during bootstrapping (refers to the process of starting up a computer system and getting it ready for operation), enabling this is necessary for accessing memory greater than the first 1 MB (Crucial for bootloader and operating system functionality to access sufficient memory for loading and running applications).

* **Function Code 0x2403**: This code tells the BIOS interrupt handler to perform the A20 enablement function.
* **BIOS Interrupt 0x15**: When the int 15h instruction is executed, control is transferred to the BIOS interrupt handler for interrupt 0x15.
* **A20 Enabling Routine**: The BIOS interrupt handler then executes its A20 enabling routine, which typically involves manipulating hardware registers to toggle the A20 line on.

**Jump**: jmp 0x0000:init is a jump to reload the code segment (CS) register of address 0x0000 and transfer control to the init label.

**why this is done**

* This jump often marks the transition from real mode to protected mode in x86 systems. Real mode is with limited features and direct access to hardware, suitable for the boot loader. Protected mode offers features like virtual memory, multitasking.
* By jumping to init after reloading CS, the boot loader can begin setting up the environment needed for the operating system kernel. This includes setting up memory paging, enabling multitasking, and initializing device drivers.