CmOS

2025

1 Introduction

In this file, the main concepts used to build CmOS (*C modeled OS*) will be detailed. The objective of this report is to complete the content of the comments that can be found in the code.
The goal of this project is to understand and apply the main principles of an OS without having to struggle with time demanding parts such as assembly code, bootloader, drivers and so on. Because it is a simulation of an OS done in C, it suits better to the definition of an hypervisor with a single virtual machine than to the one of an operating system.

2 Storage

2.1 Generalities

The storage of CmOS is a file located in `bin/disk` with a parametric size. To interface with it, `src/disk.c` gives functions to initialize, write and read it.

2.2 Organization of a file in CmOS

A file starts with 2 bytes that indicates where the first instruction is located. It is then followed by all the variables used in the program.

3 Programs

3.1 Generalities

For a classic OS, this step would not have been necessary because the programs are written in machine code and that is what is then run by the computer. However, the choice of building a dedicated simple low-level language has been made such that small programs can run on CmOS.
To begin with it, each instruction will consists of an instruction code followed by up to 3 arguments. The RAM size that was set at the beginning was 256 bytes, which leads to an 8-bit architecture so any pointer can be stored.

3.2 CPU

The CPU will have 16 bytes of registers. This way, a register can be accessed with a 4-bit identifiers. This allows any instruction to only take a maximum of 16 bits of arguments. These 16 bits can be up to 4 registers, 2 registers and an immediate byte value or an immediate 2 bytes value (disk address for example).

register name	address	size	use
FLAGS	0x0	8 bits	to be determined
R1	0x1	8 bits	general purpose register
R2	0x2	8 bits	general purpose register
R3	0x3	8 bits	general purpose register
R4	0x4	8 bits	general purpose register
R5	0x5	8 bits	general purpose register
R16	0x6	16 bits	16 bit register for arithmetic
RL	0x6	8 bits	general purpose register
RH	0x7	8 bits	general purpose register
RSI	0x8	16 bits	source pointer
RDI	0xA	16 bits	destination pointer
RI	0xC	16 bits	instruction pointer
RS	0xE	16 bits	stack pointer

Table 1: CPU registers

CF	ZF	SF	OF	⋆	⋆	⋆	⋆
Carry Flag	Zero Flag	Sign Flag	Overflow Flag				

Table 2: FLAGS register

3.3 Instruction set

3.3.1 8-bit configuration

In the instruction table, the following convention is followed:

- r* refers to a register (4 bits)
- imm* refers to an immediate value
- ⋆ indicates an unused field
- an *italic* field indicates an input
- a **bold** field indicates an output

Instruction	Op Code	byte1	byte2	16 bit reg compatibility	comment
AND	0x00	*r1* *r2*	⋆ **r3**		**r3** = *r1* AND *r2*
	0x01	*r1* **r2**	*imm8*		**r2** = *r1* AND *imm8*
OR	0x02	*r1* *r2*	⋆ **r3**		**r3** = *r1* OR *r2*
	0x03	*r1* **r2**	*imm8*		**r2** = *r1* OR *imm8*
NOT	0x04	*r1* **r2**	⋆		**r2** = NOT *r1*
SHL	0x05	*r1* *r2*	⋆ **r3**	●	**r3** = *r1* $\ll$ *r2*
	0x06	*r1* **r2**	*imm8*	●	**r2** = *r1* $\ll$ *imm8*
SHR	0x07	*r1* *r2*	⋆ **r3**	●	**r3** = *r1* $\gg$ *r2*
	0x08	*r1* **r2**	*imm8*	●	**r2** = *r1* $\gg$ *imm8*
ADD	0x10	*r1* *r2*	⋆ **r3**	●	**r3** = *r1* + *r2*
	0x11	*r1* **r2**	*imm8*	●	**r2** = *r1* + *imm8*
SUB	0x12	*r1* *r2*	⋆ **r3**	●	**r3** = *r1* - *r2*
	0x13	*r1* **r2**	*imm8*	●	**r2** = *r1* - *imm8*
	0x14	*imm8*	*r1* **r2**		**r2** = *imm8* - *r1*
MUL	0x15	*r1* *r2*	⋆		**R16** = *r1* * *r2*
	0x16	⋆ *r1*	*imm8*		**R16** = *r1* * *imm8*
IDIV	0x17	*r1* **r2**	⋆		**r2** = *R16* // *r1*
	0x18	*imm8*	⋆ **r1**		**r1** = *R16* // *imm8*
MOD	0x19	*r1* **r2**	⋆		**r2** = *R16* mod *r1*
	0x1A	*imm8*	⋆ **r1**		**r1** = *R16* mod *imm8*
MOV	0x20	*r1* **r2**	⋆	●	**r2** = *r1*
	0x21	⋆ **r1**	*imm8*		**r1** = *imm8*
	0x22	*imm16*		●	**R16** = *imm16*
LOAD	0x23	⋆ **r1**	⋆		**r1** = *(*RSI*)
	0x24	⋆	⋆		**R16** = *(*RSI*)
STORE	0x25	⋆ *r1*	⋆		*(**RDI**) = *r1*
	0x26	⋆	⋆		*(**RDI**) = *R16*
REGDUMP	0x27	⋆	⋆		stores all the registers in **RDI**
REGFILL	0x28	⋆	⋆		loads *RSI* in all the registers
CMP	0x30	*r1* *r2*	⋆	●	raises **ZF**, **SF** for (*r2* - *r1*)
TEST	0x31	*r1* *r2*	⋆	●	raises **ZF** for *r1* AND 0b1 $\ll$ *r2*
	0x32	⋆ *r1*	*imm8*	●	raises **ZF** for *r1* AND 0b1 $\ll$ *imm8*
SKIFZ	0x40	⋆	⋆		skip next instruction if *ZF* = 1
SKIFNZ	0x41	⋆	⋆		skip next instruction if *ZF* = 0
PRNT	0xF0	⋆ *r1*	⋆	●	prints *r1* characters from *RSI*
HLT	0xFF	⋆	⋆		stops the CPU

Table 3: 8-bit instruction set

3.4 Program syntax

To write code, you start by defining what will be stored in memory by declaring 1 byte at a time using `DB` and then you start writing your code using the instruction set table. There is no punctuation and no way to write comments.