Advanced Operating System and Virtualization

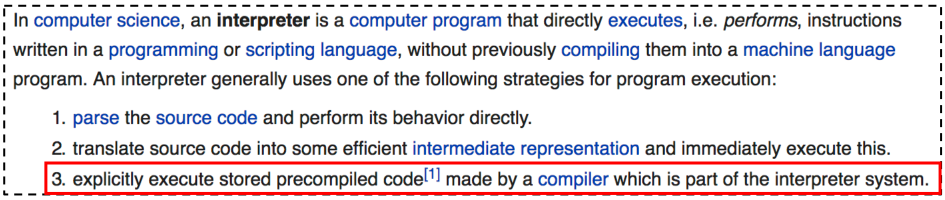
Introduction of Interpreter

Hiroaki Fukuda

Contents

- What is interpreter
- Architecture of 8086

What is interpreter



In computer science, an **interpreter** is a computer program that directly executes, i.e. *performs*, instructions written in a programming or scripting language, without previously compiling them into a machine language program. An interpreter generally uses one of the following strategies for program execution:

1. parse the source code and perform its behavior directly.
2. translate source code into some efficient intermediate representation and immediately execute this.
3. explicitly execute stored precompiled code[1] made by a compiler which is part of the interpreter system.

Defined by Wikipedia

Our interpreter executes minix binary

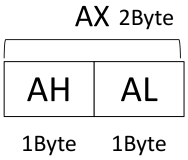
- Need to know 8086 architecture
- Need to know minix operating system

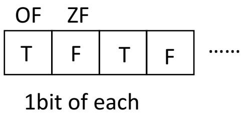
Von Neumann architecture

- Computer consists of following components
  - CPU
  - Memory
  - Register
- Execution Cycle
  - Fetch
  - Decode
  - Execute
  - Store

8086 architecture

- CPU
  - 16bits
- Memory
  - Little Endian
- Register
  - General
    - AX, CX, DX, BX, SP, BP, SI, DI (16bits)
    - AL, CL, DL, BL, AH, CH, DH, BH (8bits)
  - Flag
    - OF, DF, IF, TF, SF, ZF, NF, AF, PF, CF

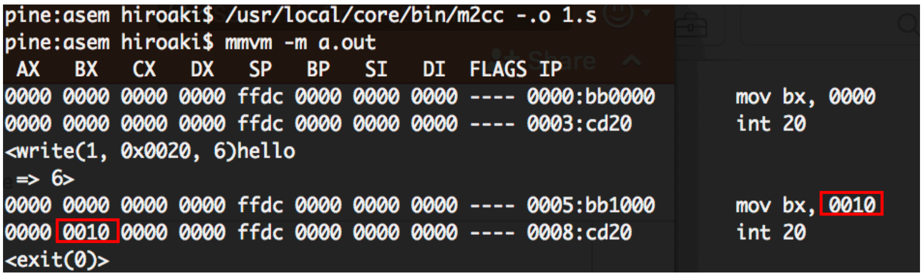




Execute program

1. Extract text and data
2. Copy text and data to the memory
   1. Text and data are stored separately
3. Set registers to initial value (0)
4. Fetch/decode/execute/store

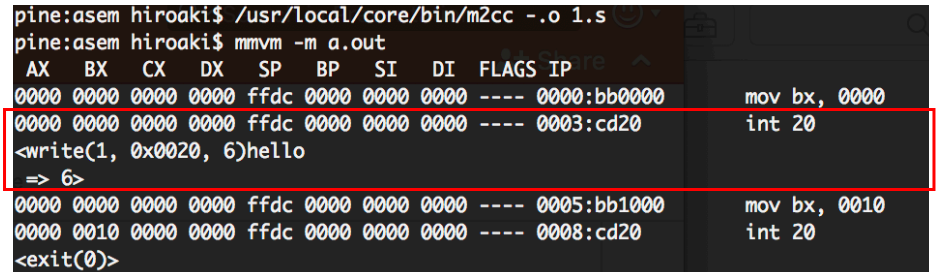
Execute 1.s Why it happens?



```
pine:asem hiroaki$ /usr/local/core/bin/m2cc -.o 1.s
pine:asem hiroaki$ mmvm -m a.out
 AX   BX   CX   DX   SP   BP   SI   DI  FLAGS IP
0000 0000 0000 0000 ffdc 0000 0000 0000 ---- 0000:bb0000      mov bx, 0000
0000 0000 0000 0000 ffdc 0000 0000 0000 ---- 0003:cd20        int 20
<write(1, 0x0020, 6)hello
 => 6>
0000 0000 0000 0000 ffdc 0000 0000 0000 ---- 0005:bb1000      mov bx, 0010
0000 0010 0000 0000 ffdc 0000 0000 0000 ---- 0008:cd20        int 20
<exit(0)>
```

Mov: move data to the specified register

Let’s see the execution log of 1.s



```
pine:asem hiroaki$ /usr/local/core/bin/m2cc -.o 1.s
pine:asem hiroaki$ mmvm -m a.out
 AX   BX   CX   DX   SP   BP   SI   DI  FLAGS IP
0000 0000 0000 0000 ffdc 0000 0000 0000 ---- 0000:bb0000      mov bx, 0000
0000 0000 0000 0000 ffdc 0000 0000 0000 ---- 0003:cd20        int 20
<write(1, 0x0020, 6)hello
 => 6>
0000 0000 0000 0000 ffdc 0000 0000 0000 ---- 0005:bb1000      mov bx, 0010
0000 0010 0000 0000 ffdc 0000 0000 0000 ---- 0008:cd20        int 20
<exit(0)>
```

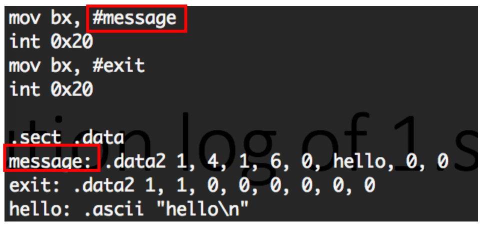
System call

write(fd, addr, num)

fd = 1
addr = 0x20
num = 6

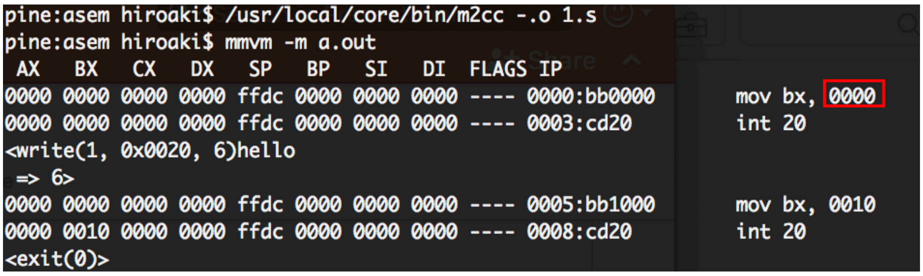
But Why??

1.S



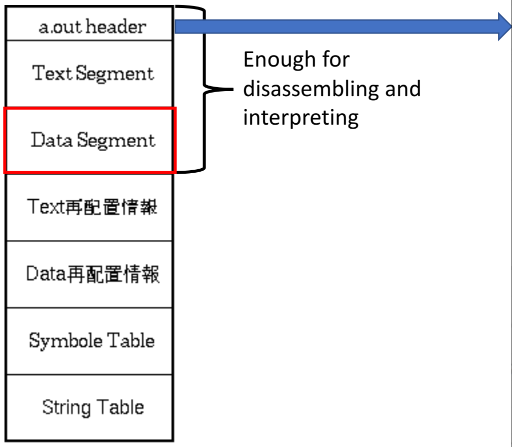
```
mov bx, #message
int 0x20
mov bx, #exit
int 0x20

.sect .data
message: .data2 1, 4, 1, 6, 0, hello, 0, 0
exit: .data2 1, 1, 0, 0, 0, 0, 0, 0
hello: .ascii "hello\n"
```

We need Data!



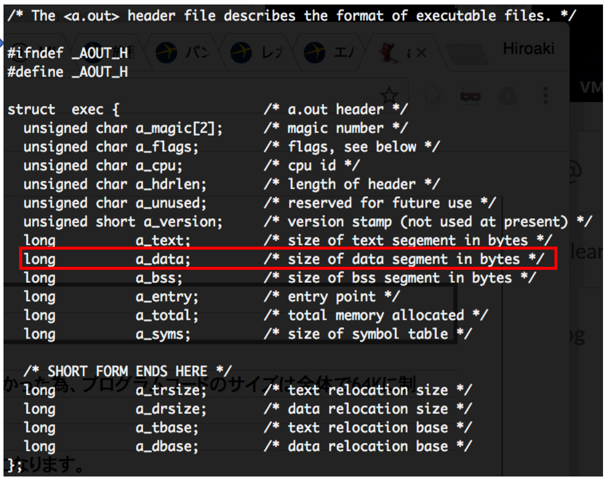
```
pine:asem hiroaki$ /usr/local/core/bin/m2cc -.o 1.s
pine:asem hiroaki$ mmvm -m a.out
 AX   BX   CX   DX   SP   BP   SI   DI  FLAGS IP
0000 0000 0000 0000 ffdc 0000 0000 0000 ---- 0000:bb0000       mov bx, 0000
0000 0000 0000 0000 ffdc 0000 0000 0000 ---- 0003:cd20         int 20
<write(1, 0x0020, 6)hello
 => 6>
0000 0000 0000 0000 ffdc 0000 0000 0000 ---- 0005:bb1000       mov bx, 0010
0000 0010 0000 0000 ffdc 0000 0000 0000 ---- 0008:cd20         int 20
<exit(0)>
```

Where is the data? - a.out format



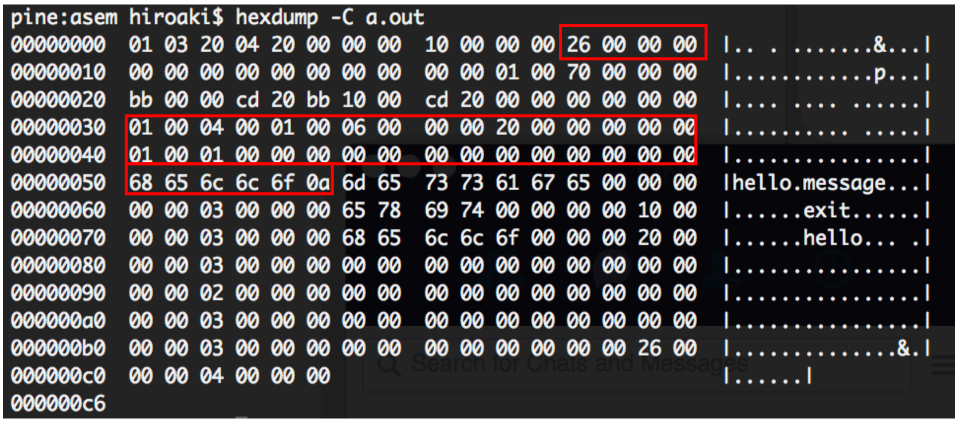
Enough for disassembling and interpreting

usr/include/a.out.h



```
/* The <a.out> header file describes the format of executable files. */

#ifndef _AOUT_H
#define _AOUT_H

struct  exec {                        /* a.out header */
  unsigned char a_magic[2];           /* magic number */
  unsigned char a_flags;              /* flags, see below */
  unsigned char a_cpu;                /* cpu id */
  unsigned char a_hdrlen;             /* length of header */
  unsigned char a_unused;             /* reserved for future use */
  unsigned short a_version;           /* version stamp (not used at present) */
  long          a_text;               /* size of text segement in bytes */
  long          a_data;               /* size of data segment in bytes */
  long          a_bss;                /* size of bss segment in bytes */
  long          a_entry;              /* entry point */
  long          a_total;              /* total memory allocated */
  long          a_syms;               /* size of symbol table */

  /* SHORT FORM ENDS HERE */
  long          a_trsize;             /* text relocation size */
  long          a_drsize;             /* data relocation size */
  long          a_tbase;              /* text relocation base */
  long          a_dbase;              /* data relocation base */
};
```

Binary Again



```
pine:asem hiroaki$ hexdump -C a.out
00000000  01 03 20 04 20 00 00 00  10 00 00 00 26 00 00 00  |.. . .......&...|
00000010  00 00 00 00 00 00 00 00  00 00 01 00 70 00 00 00  |............p...|
00000020  bb 00 00 cd 20 bb 10 00  cd 20 00 00 00 00 00 00  |.... .... ......|
00000030  01 00 04 00 01 00 06 00  00 00 20 00 00 00 00 00  |.......... .....|
00000040  01 00 01 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  68 65 6c 6c 6f 0a 6d 65  73 73 61 67 65 00 00 00  |hello.message...|
00000060  00 00 03 00 00 00 65 78  69 74 00 00 00 00 10 00  |......exit......|
00000070  00 00 03 00 00 00 68 65  6c 6c 6f 00 00 00 20 00  |......hello... .|
00000080  00 00 03 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000090  00 00 02 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000a0  00 00 03 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000b0  00 00 03 00 00 00 00 00  00 00 00 00 00 00 26 00  |..............&.|
000000c0  00 00 04 00 00 00                                 |......|
000000c6
```

Let’s implement program loader and execute

- Compile 1.s and load a.out to your own interpreter
  - How do we have to load text and data to memory?