CS 240 Summer 2017 (Park)

C CODE AND MACHINE CODE

1. A modern computer

As discussed in class, our simplified view of a modern computer is comprised of two main components: CPU and memory. The details of this architecture, invented by Alan Turing in 1936, are discussed in CS 251. Although Turing came up with the idea and its design, it was built in the 1940s and 1950s by several groups including an effort lead by John von Neumann, one of the giants of computer science. Some people refer to the computer discussed in class as "von Neumann architecture." Turing first coined the term "computer" and in his honor we call today's computers, which follow essentially the same design proposed in 1936 (with nifty improvements), Turing machines.

As noted in class, today's computers have lots of interfaces (USB, WiFi, Bluetooth, Ethernet, GPS, cellular, NFC, etc.) but in CS 240 the focus will be on terminal input/output and file I/O. The former means reading input from keyboard and writing output to a terminal display (or screen). The latter means reading/writing from files in an operating system, in our case, Linux.

2. Translating C code into machine code

We said that a CPU does not understand C programs. Nor does it know what to do with Java programs and bytecode. The only code that a CPU understands is machine code. Code written in any other language must be translated to machine code for it to run on a computer.

The example code given in lab1/v1

```
main()
{
int x, y, z;

  x = 2;
  y = 3;
  z = x + y;

}
```

was translated with the help of a compiler (gcc) into a hypothetical machine code

```
MOV #2 100
MOV #3 101
ADD 100 101 102
```

The first instruction, MOV, has two arguments, also called operands. The first denotes a constant 2, the second the memory location or address 100 in RAM. Each memory location contains bits which are its content. In a 32-bit computer architecture, each memory location contains 32 bits. Each memory location is identified by its address which is also 32 bits long. As we noted, in 64-bit architectures, memory contents are 64-bits wide, but addresses typically use less than 64 bits since 2^64 is way too large even by today's standards.

The second instruction puts the constant 3 into memory location 101, and the third instruction ADD adds the content of addresses 100 and 101, then stores the result 6 into address 102.

3. Apples and oranges

As emphasized in class, it is very important to distinguish the two features of memory: address and content. In the above example, 102 is an address. Its content, after addition, is 5. This is fundamental to understanding the notion of pointers in C and the myriad programming techniques that depend on it. Even the simple act of reading an integer with the help of the standard I/O library function scanf() exposes us to pointers.

4. What a CPU does

The following is an additional note that is useful but not necessary to following the discussion in class.

A CPU, conceptually, is a simple device that repeats the following three steps forever (i.e., in an infinite loop):

```
fetch the next instruction
decode the fetched instruction
execute the fetched instruction
```

To fetch means to read the next instruction from memory (i.e., RAM). The first instruction of our C code is the first instruction of main(). Of course, after it's been translated into machine code. In the above example, the first instruction is

```
MOV #2 100
```

The next, MOV #3 101. There are no jumps unless the programmer uses a jump instruction to tell the CPU to do so. In high level languages these are known as "goto" instructions. It is strongly recommended against using goto's as they tend to lead to unwieldy "spaghetti code." Up until the 1980s, most app code was written using "goto" instructions. A computer scientist, Dijkstra, went on a mission to discourage programmers from using goto's in their programs. He also visited Purdue. Obviously, he succeeded since we do not use goto statements which exist in many programming languages, including C.

In machine languge, however, the only way to make the CPU not execute the next instruction is through goto (i.e., jump) instructions. The jump can be unconditional (e.g., jump to address 1000 which contains the instruction to be executed next) or conditional (e.g., if the content of address 50 is 0 then jump to address 1000).

Decode means to determine what the instruction is (MOV, ADD, JMP, etc.). Execute means to do it.