CRYPTOGRAPHY TOPIC I
AN UNDERVIEW OF C

PAUL L. BAILEY

We say that C is a third generation language.

(a) machine language: actual numbers that the CPU uses as commands
(b) assembler: converts mnemonic commands into machine language commands using a one to one correspondence.
(c) c-language: a handful of flow control keywords and block-determining punctuation, together with the ability to invoke functions compiled from assembler (or from C).

1. Machine Language

Powers of two are obviously important in programming; we list the first 16:

$$2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536.$$

We describe an simple 16 bit computer; a byte is eight bits, and for this computer, a word is 16 bits.

(a) The memory address space is 16 bits, so there are $2^{16} = 65536$ memory locations; the number of a memory location is called an *address*. The first address is 0 and the last address is 65535.
(b) Each memory location contains a 16 bit integer, so it is a number between 0 and 65535. The memory is divided into 256 segments, each containing 256 words. An address consists of a segment together with an 8 bit offset within that segment.
(c) The CPU contains four 8 bit registers:
- data segment
- data offset
- program segment
- program offset

Commands to the CPU are 16 bit integers, broken into two 8 bit parts:

(a) The first 8 bits are the command itself; thus there are 256 commands.
(b) The second 8 bits are the argument to the command.

The program pointer is a 16 bit integer which consists of the program segment in its high bits and the program offset in its low bits. Similarly, there is a data pointer.

The number stored at the address indicated by the program pointer is the next command for the CPU. After a command is executed, the program pointer is incremented, unless the command changed the program pointer.

Date: September 6, 2007.

There are commands to set the data and program segments, in which case the eight bits of the argument are copied into one of these registers. There are commands to jump to a memory location within the current segment, in which case the eight bits of the argument are copied into the program offset.

We may rotate the bits of a data address, in which case the 8 bits of the argument indicate the offset of the memory location to be rotated within the current data segment. Other unary operations also work in this way.

We may add two numbers in the same data segment. The offset of the location of one number is the argument to the command, and the offset of the location of the other number is the data offset. The result is placed in the location indicated by the data pointer. Other binary operations also work in this way.

The way in which 16 bit integers are converted into CPU behavior is referred to as the *machine language* for this particular computer.

2. Assembler

We continue discussing our theoretical 16 bit machine, and create an *assembly language* for it.

Each of the 256 commands are given a 3 letter mnemonic. Each mnemonic represents to a number between 0 and 255; this number is the corresponding machine language command. Addresses are given tags. We given an example, in which semicolons represent comments. Here, PP means program pointer, DP means data pointer, and *DP means the contents of the data pointer.

```
T1  ROR T2  ; Rotate memory at T2 right by 1
    DAT T4  ; Set DP to T4
    MOV T2  ; Move the contents of T2 to *DP
    ADD T3  ; Add the contents of T3 to *DP
    JMP T5  ; Copy the contents of T4 to PP
T2  32      ; some data
T3  61      ; some data
T4  0       ; some data
T5  ; continue with program
```

We use an *editor* to create a text file containing the text of this program, and the file is stored with a .ASM extension. This file contains *source code*.

We use a *compiler* to translate the mnemonics and tags into machine language; the output is stored in a file with a .OBJ extension. This file is called a *binary module*.

We may use a *library manager* to combine one or more modules into a *library*. Libraries are stored with .LIB extensions

We use a *linker* to combine one or more modules, obtained from .OBJ or .LIB files, into an executable program. This program is stored with a .EXE extension.

Each line of the program corresponds to exactly one machine language command; in this example, there is a bijective correspondence between words in a binary program and line in the assembly source code. Modern CPU are more complicated in this sense, but the underlying idea that machine code corresponds to lines of assembly language is the same.

3. C Language

Assembly allows the programmer to create the faster possible program, since it gives complete control over the CPU. Because assembly language is specific to the CPU, programs written using it are not portable; moreover, since the level of detail in so deep, it takes a lot of time to write the simplest programs.

The C language was developed to overcome these difficulties without sacrificing programmer control and run-time efficiency. C contains the ability to declare data of various types, an expression evaluator, and a handful of flow control keywords. All other functionality in a C program comes from commands referred to as *functions*, which are not specified as part of the C language per se. The flow control keywords of C do nothing more than determine the order in which functions are called. In this way, C is as close to assembler as any so-called third generation language gets.

The functions called by a C program may be written in assembler or in C and previously compiled (probably into a library but possibly into objects), or they may be written elsewhere in the program itself.

There is a *standard library* of pre-compiled functions which come with every C compiler; these perform io, memory management, string manipulation, and so forth.

Files important to the compilation of C programs (with Microsoft extensions) include:

(a) Source Files (.C for C or .CPP for C++): the actual text of the program
(b) Object Files (.OBJ): machine language modules
(c) Library Files (.LIB): sets of modules packaged together
(d) Executable Files (.EXE): actual code, executable from the command line

The process of using C to create an executable file is in four major steps.

(a) The *editor* allows the programmer to create the *source code*.
(b) The *precompiler* translates the *precompiler commands* in the source code. For example:
   - `#define` sets a compile time variable
   - `#include` loads the contents of another source file

   In C, the precompiler commands begin with `#`. The output of the precompiler is typically stored in memory or a temporary file; we never see it.
(c) The *compiler* translates the output of the precompiler into relocatable machine language. The output of a compiler is a *module*; in Microsoft, these are given .OBJ extensions.
(d) The *linker* combines one or more modules in .OBJ files, together with modules pulled from libraries stored with .LIB extensions, together with "startup code", to create an executable .EXE file.

4. C Language Keywords

The following is a complete list of the C language keywords.

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

We may group these keywords:

- Flow Control
  - Branching: if, else, switch/case/default
  - Looping
    - Outer Loop: while, do/while, for
    - Inner Loop: break, continue
  - Jumping: goto
  - Exiting: return
- Data Typing
  - Creating: typedef
  - Atomic Types: char, int, float, double, void
  - Type Modifiers: auto, const, extern, long, register, short, signed, static, unsigned, volatile
  - Composite Types: enum, struct, union
- Functions: sizeof

The following keywords are not part of C, but have been added to C++.

bool	catch	class	delete	friend	inline
new	namespace	operator	private	protected	public
tempate	this	throw	try	template	

Department of Mathematics and CSci, Southern Arkansas University
E-mail address: plbailey@saumag.edu