北京理工大學
Beijing Institute of Technology

本科生课程设计

课程名称：计算机原理与应用

设计名称：利用汇编语言计算字符串 CRC-32 校验和

任课教师：	李海	学生姓名：	胡森康
日期：	2020 年 11 月	选　题　：	A 题
类　　型：	□ 原 理 验 证　　■ 综 合 设 计　　□ 自 主 创 新		
班　　级：	05961808	学　　号：	1120183150
学　　院：	信息与电子学院	专　　业：	电子信息工程
组　　号：		同组同学：	
成　　绩：			

信息与电子学院
SCHOOL OF INFORMATION AND ELECTRONICS

利用汇编语言计算字符串 CRC-32 校验和

1120183150 胡森康

2020 年 11 月 12 日

目　录

1 实验任务

- 在程序时间段定义一个字符串用于测试 CRC-32 校验和的计算；
- 编写子程序计算 CRC-32 校验和，生成多项式为：$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$
- 主程序调用子程序计算校验和后，将结果输出到屏幕。

2 实验环境

- VSCODE
- MASM 插件（16 位环境）

3 理论原理

循环冗余校验（英语：Cyclic redundancy check，通称“CRC”）是一种根据网络数据包或电脑文件等数据产生简短固定位数校验码的一种散列函数，主要用来检测或校验数据传输或者保存后可能出现的错误。生成的数字在传输或者存储之前计算出来并且附加到数据后面，然后接收方进行检验确定数据是否发生变化。由于本函数易于用二进制的电脑硬件使用、容易进行数学分析并且尤其善于检测传输通道干扰引起的错误，因此获得广泛应用。

采用 CRC 校验时，发送方和接受方用同一个生成多项式 $g(x)$，$g(x)$ 是一个 $GF(2)$ 多项式，并且 $g(x)$ 的首位和最后一位的系数必须为 1 。

CRC 的处理方法：发送方用发送数据的二进制多项式 $t(x)$ 除以 $g(x)$，得到余数 $y(x)$ 作为 CRC 校验码。校验时，以计算的矫正结果是否为 0 为据，判断数据帧是否出错。设生成多项式是 r 阶的，（最高位为 x^r）具体步骤如下：

1. 在发送的 m 位数据的二进制多项式 $t(x)$ 后添加 r 个 0 ，扩张到 $m+r$ 位，以容纳 r 位的校验码，追加 0 后的二进制多项式为 $T(x)$
2. 用生成多项式 $g(x)$ 除以 $T(x)$，得到 r 位余数 $y(x)$，它就是 CRC 校验码。
3. 把 $y(x)$ 追加到 $t(x)$ 后面，此时的数据 $s(x)$ 就是包含了 CRC 校验码的待发送字符串；由于 $s(x)=t(x)y(x)$ ，因此 $s(x)$ 肯定能被 $g(x)$ 除尽。

4 算法思路

CRC-32 码由四个字节构成，在开始时 CRC 寄存器的每一位都预置为 1，然后把 CRC 寄存器与 8bit 的数据进行异或，之后对 CRC 寄存器从高位到低位进行移位，在最高位的位置（MSB）补 0，而最低位（LSB，移位后已经被移出 CRC 寄存器）如果为 1，则把寄存器与预定义的多项式码进行异或，若 LSB 为 0，则无需进行异或。重复上述的由高至低的移位 8 次，第一个 8bit 数据处理完毕，用此时 CRC 寄存器的值与下一个 8bit 数据异或并进行如前一个数据似的 8 次移位。所有字符处理完成后 CRC 寄存器内的值即为最终的 CRC 值。

具体过程如下：

1. 设置 CRC 寄存器，并给其赋值 FFFFH
2. 将数据的第一个 8bit 字符与 32 位 CRC 寄存器的低 8 位进行异或，并把结果存入 CRC 寄存器。
3. CRC 寄存器向右移移位，MSB 补 0，移出并检查 LSB。
4. 重复第 3 与第 4 步直到 8 次移位全部完成，此时一个 8bit 数据处理完毕。
5. 重复 2 至 5 步直到所有数据全部处理完毕。
6. 最终 CRC 寄存器的内容即为 CRC 值。

5 程序介绍

本程序的代码段由 4 个程序组成，分别是主程序 *main* ，CRC 校验和计算子程序 *calculate_CRC*，二进制结果输出子程序 *output_BIN* 和十六进制结果输出子程序 *output_HEX* 组成。

当用户输入一串字符时，计算其 CRC-32 校验和并分别将其输出为二进制数和十六进制数，当用户输入字符“#”时，程序结束。

5.1 CRC 校验和计算子程序

CRC 校验和计算子程序如下图（5-1）所示

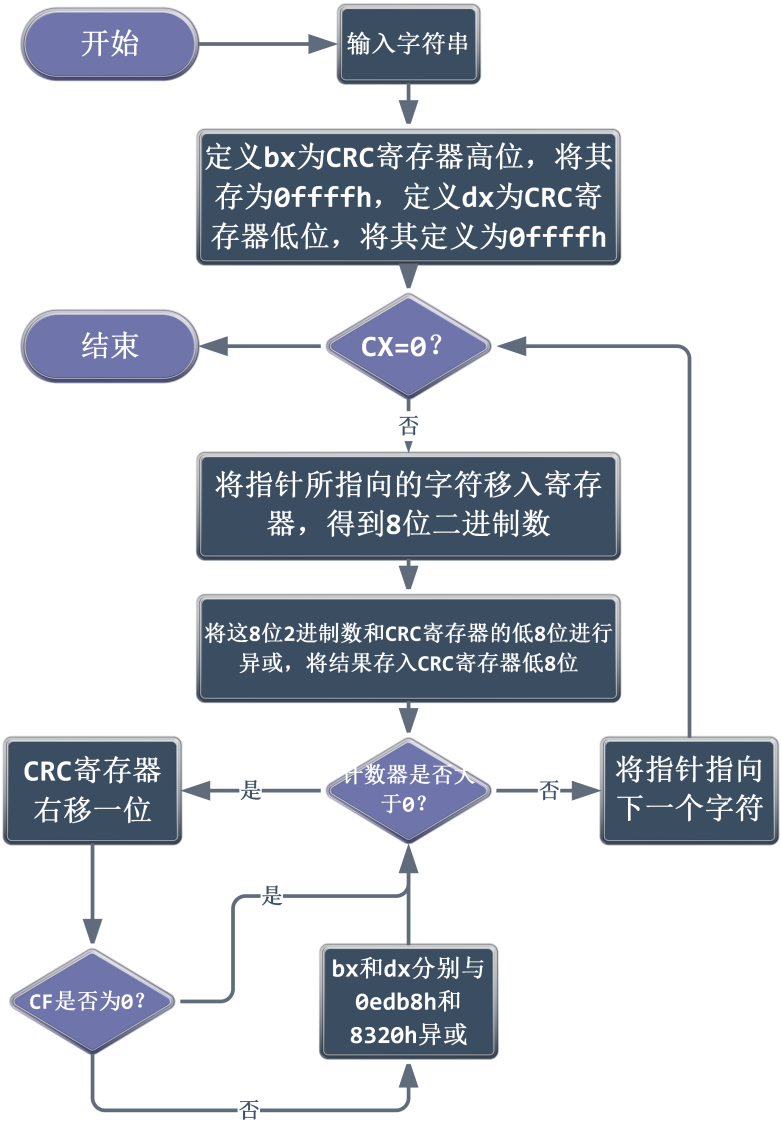


图 5-1: CRC 校验和计算子程序流程图

5.2 二进制结果输出子程序

二进制输出的主要思路为：对寄存器中的数进行左移，使最高位溢出到标志位 *CF* 中，再利用有符号加法 *adc* 将 *CF* 中的值取出，并输出即可。

流程图如下图（5-2）所示:

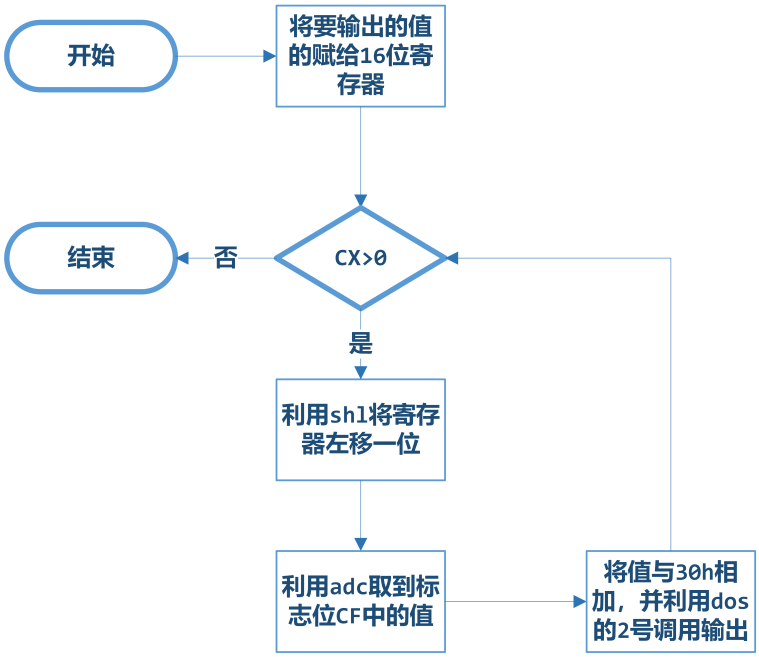


图 5-2: 二进制结果输出子程序流程图

5.3 十六进制结果输出子程序

十六进制输出的主要思路为：先考虑高八位，使八位数据逻辑右移四位，再加上 30h，比较是否是对应字符的 ASCII 码，即是否小于 39h，小于则可以直接输出，因为此时对应的就是该数字的 ASCII 码；用高八位与 0fh 做与运算，使高四位为 0，低四位为原来的数值，再与 30h 相加，比较是否小于 39h，小于则直接输出，若大于 39h，则表示要输出的是 A-F，将其自加 7h，得到正确的 ASCII 码，然后输出。

流程图如下图（5-3）所示:

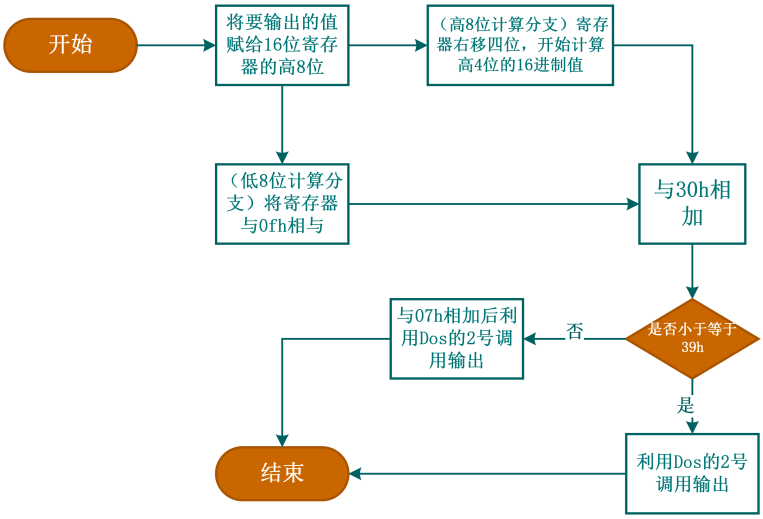


图 5-3: 十六进制结果输出子程序流程图

6 程序结果

程序运行结果如下图（6-4）所示：

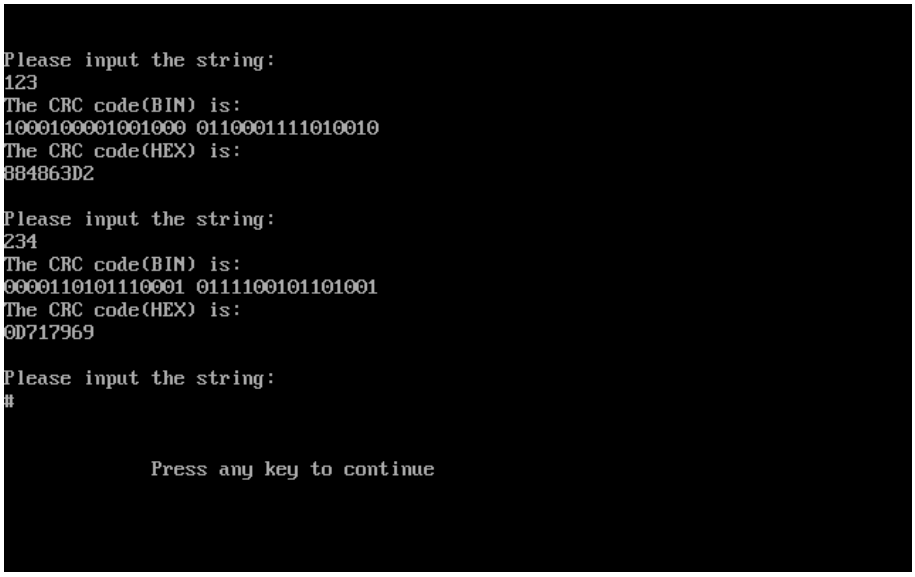


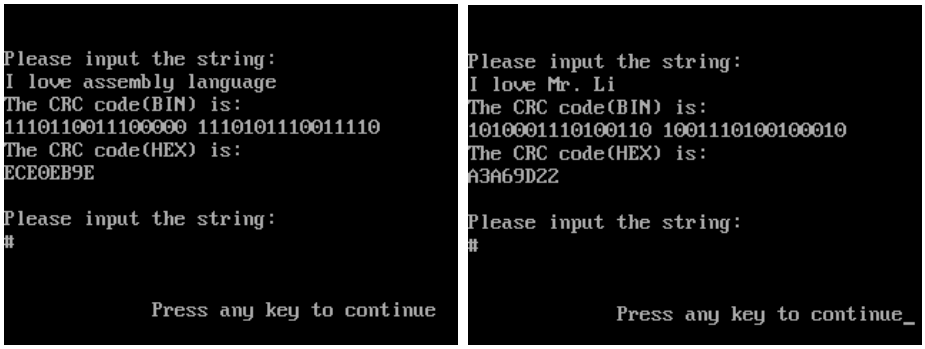
图 6-4: 程序运行结果

可见，输入字符串为 123 时，程序输出的 CRC-32 二进制校验码为 1000 1000 0100 1000 0110 0011 1101 0010，十六进制校验码为 884863d3h，经检验是正确的。

此时，程序并没有退出，而是继续等待用户输入字符串。用户输入字符串为 234 时，程序输出的 CRC-32 二进制校验码为 0000 1101 0111 0001 0111 1001 0110 1001，十六进制校验码为 0d717969h，经检验是正确的。

当用户继续输入一个 # 时，程序结束。

更多的结果如下：



7 思考题

7.1 题目描述

是否有 2 个字符串的 CRC-32 校验和是相同的？

7.2 解答

可能有两个字符串的 CRC-32 校验和是相同的。

对于 32 位校验，若输入超过 32 位，因此不能有超过 2^{32} 个不同的输入来生成不同的 CRC。且 CRC 多项式是线性结构，可以很容易地故意改变数据而维持 CRC 不变。

就比如字符串“b5a7b602ab754d7ab30fb42c4fb28d82”和字符串“d19f2e9e82d14b96be4fa12b8a27ee9f”的 CRC-32 就是相同的，均为 997d37bah 如下图（7-5）所示：

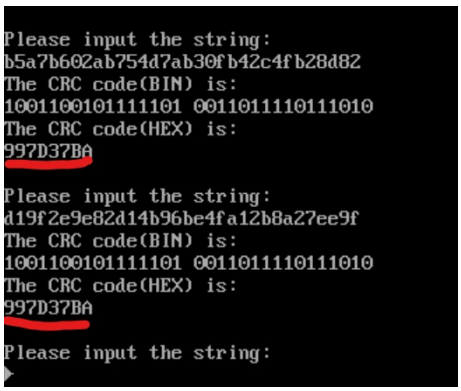


图 7-5: CRC-32 相同的两个字符串

8 收获和感想

在本次大作业中我收获了很多，首先是自学的能力，其次是编程的能力。

在进行大作业的过程中，我看了一些文献和博客，将 CRC 校验的原理搞清，并思考实现的算法。在这个过程中增强了我的自主学习能力，也收获了很多新的知识，比如模 2 除的计算方法。搞清楚了 CRC 校验的原理，接下来就是如何计算 CRC 码，这就涉及到了算法的问题了，在思考算法的过程中，我意识到模 2 除的本质是异或。因此很快便明确了算法的方向，就是不断的移位和异或。

因为是 32 位校验，因此在编程的过程中我先利用了 vs 进行编程，算法很简洁，也是正确的，但编出来总是结果不正确，这个问题让我头疼了好几天，最终还是在李老师的点播下才成功。在 32 位环境中是比较好实现，但不知道什么问题我的 vs 总是导不进 masm 的库，这就导致了我不能使用 stdout 等函数进行输入输出，我看了很多技术帖子，但还是没解决我的 vs 谜之一样的 bug，遂放弃用 vs 编，改用 vscode 的 masm 插件（16 位）。虽然利用 16 位环境算法稍稍比 32 位环境复杂点，但还是很快就将 32 位代码移植到 16 位环境中了。

在保证程序能得到正确的结果后，便开始处理输入和输出，主要是输出。要将结果输出为 2 进制或 16 进制也是需要一些代码量的，在处理输入输出的过程中虽然也遇到了很多意想不到的 bug，但还是被成功解决了。

在此次大作业中，我意识到要熟练掌握编程，核心在于多动手去编。其次是，数学原理和算法很多时候差别是很大的，比如计算 CRC，数学原理是对进行模 2 除，最后得到余数即为结果，但在算法中是进行移位和异或，这在表面上是不同的，而且此种情况在低级语言中由显得突出，因为这和芯片的运作方式是分不开的。