PORTLAND STATE UNIVERSITY
CS201: COMPUTER SYSTEMS PROGRAMMING
SPRING 2016

HOMEWORK 3

DUE: May 4, 2:00 PM

1. Goals and Overview

1) Develop practical knowledge of x64 Assembly Language including Branches, Arithmetic and Logic Operations
2) Interface with C programs using the System V x64 calling convention
3) Optimize Assembly code using Conditional Moves to replace branches
4) Access unidimensional arrays using Assembly

2. Development Setup

For this programming assignment you must work individually. You will work in the CS Linux Lab (linuxlab.cs.pdx.edu) located in FAB 88-09 and 88-10. If you don't already have an account, go to http://www.cat.pdx.edu/students.html for instructions.

For this Homework you must use x64 Assembly (AT&T Syntax) and Make. Please refrain from using C code, any language extensions or 3rd party libraries. We will use the GNU Assembler (AS), the GNU C Compiler (gcc) and Make already installed in the development machines in the lab.

Grading will be done in this setup so please make sure that your code works under this conditions.

3. Introduction

In this Homework we will implement a Cyclic Redundancy Check algorithm (CRC) in Assembly. CRC is an Error Detecting Algorithm based on cyclic codes. CRC is widely used in many systems including USB, Bluetooth, the Dallas 1-wire bus, Zlib and GSM and CDMA cellphone networks. It is very fast and efficient to implement in hardware and software.

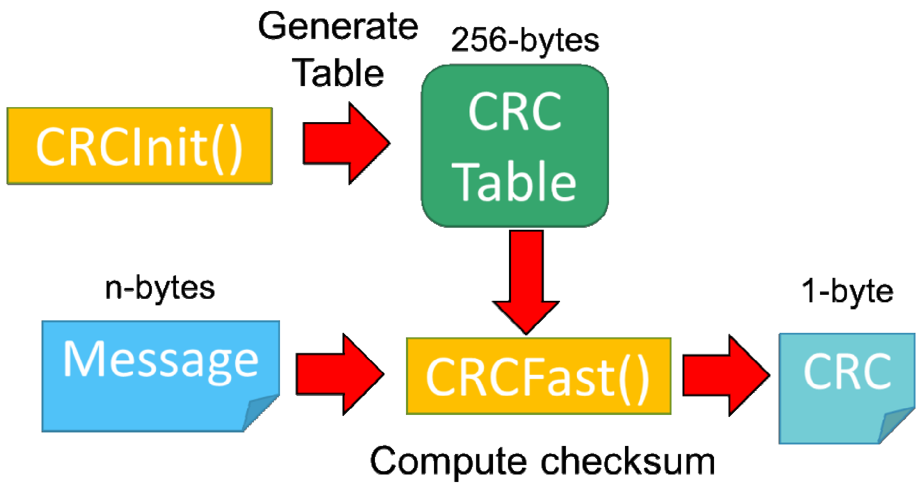
CRC is computed as the polynomial division of the Message M divided by the CRC Polynomial C. The checksum takes the form of the reminder R resulting from dividing M/C. There are many variants based on different polynomials. The length of the resulting checksum depends on the length of the CRC Polynomial.

Our implementation will use CRC-8. This particular implementation of CRC output an 8-bit signature and it uses 0x5D as the CRC Polynomial.

4. Problem Description

4.1 Fast CRC Implementation

Our CRC-8 implementation will be based on the fact that the number of residues for a given CRC polynomial is fixed. So we can precompute every possible residue and use that table to compute the CRC for a given message. For the case of CRC-8 we have 256 possible residues, so the size of the precomputed CRC table is 256 bytes.



4.2 Assembly Implementation

You are provided with a complete solution of the CRC-8 algorithm in C containing the Fast CRC-8 algorithm in crc.c and a main function that takes as parameter a string in the command line with the message and calls CRCInit() and CRCFast().

Your job in this assignment is to write a replacement of CRCInit() and CRCFast() in assembly (crca1.S). **The main C function provided in hw3.c should be able to link to the new object file assembled (crca1.o). The resulting linked executable file should be named hw3a1.**

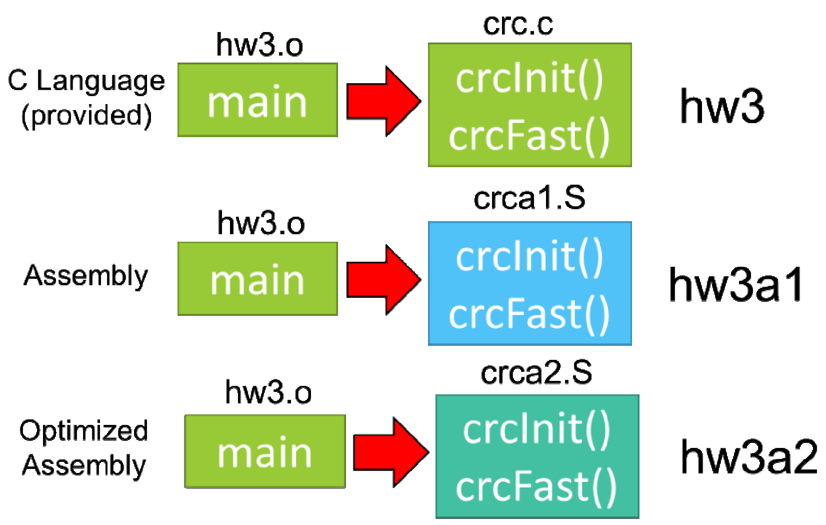
You must not make any changes to the structure or implementation of hw3.c and you must follow all the calling conventions necessary to interface with the provide hw3.c file.

4.3 Optimized Assembly Implementation

As mentioned in class If/Else blocks in which the outcome if difficult to predict are very impactful in the performance of modern microarchitectures. To avoid this problem most modern ISA provide support of Conditional Moves in which the If/Else block can be replaced by a Conditional Move that involves no change in the Program Counter (RIP). As part of this Homework, you must also provide an optimized assembly version based on crca1.S that removes the if condition in the CRCInit() function. You can identify this condition in line 30 of file crc.c. You must replace your original assembly code for this If condition with a Conditional Move and submit it as crca2.S. The main C function provided in hw3.c should be able to link to the new object file assembled (crca2.o). The resulting linked executable file should be named hw3a2.

4.3 Makefile

As part of your program your solution you should augment the provided Makefile so that it automatically compiles all the 3 variants of the code: Executable using C Version of CRC named hw3 (provided), Executable using Assembly Version of CRC with If/Else branches named hw3a1 and Executable using Optimized Assembly version with Conditional Move instructions named hw3a2. Your Makefile must also provide a cleanup entry (make clean) to delete all the files generated by the compilation process (i.e. object files, executables, libraries, etc.)



5. Implementation Overview

These are a few key concepts you must address as part of your implementation:

1) You must allocate an array in the data segment to be able to hold the CRC Table
2) You must declare the names of both CRCInit and CRCFast as global symbols in your Assembly file so that the linker can find them and properly interface to them
3) Your functions must respect the System V calling conventions by preserving the values of the callee-saved registers.
4) Your function must follow the System V calling conventions for parameters passing and function return so that your code can interface with the function caller (hw3.c).
5) Both of your assembly implementations must return the exact same values as the provided implementation in C.

6. Hand-In

For submission, you should provide only source code (*.S, *.c, *.h) and a Makefile script as outlined in Section 4.3. More specifically you must provide the original unchanged crc.c file, the non-optimized assembly CRC version in crc1a.S, the optimized assembly CRC version in crc2a.S and the Makefile.

Please pack your files into a TAR file with the following filename structure CS201_odinId_HW3.tar. Please replace “odinid” with your ODIN id.

We will use the D2L system for submission (https://d2l.pdx.edu/). **The deadline for submission is May 4th at 2:00 pm.** Please remember that there is no late policy.

7. Rubric

Grading will be done according to the following Rubric with partial credit given for features partially implemented.

Feature	Possible Points
CRC Table is properly allocated in Data segment	10
CRC Table is properly initialized by CRCInit() and assembly function interfaces with the C caller function main	35
CRC checksum is properly computer by CRCFast() and assembly function interfaces with the C caller function main	20
If condition is replaced by Conditional in CRCInit()	20
The code has no segmentation faults	5
All callee-saved registers are preserved across the function calls	10
Total	***100***

8. References

Assembler Manual - http://web.mit.edu/gnu/doc/html/as_toc.html

GNU Assembler Examples - http://cs.lmu.edu/~ray/notes/gasexamples/

GNU Make Manual (Look at Section 2.2 for a simple Makefile example) - http://www.gnu.org/software/make/manual/make.html