Why Convert from ASM to C?

White Paper

By Avi Nudelman MSCS
Chief Technology Officer
Micro-Processor Services, Inc.
http://www.mpsinc.com
avin@mpsinc.com

Keywords: ASM C Assembly Assembler ASM51 ASM86 ASM186 ASM286 ASM386 ASM486 MASM RASM TASM ASM360 ASM370 ASM390 ASM6800 ASM68K PowerPC Assembly Power Assembly ARM Assembly Translation Tools Concept Translation Simulation Translation

Introduction

In this white paper we describe the reasons for moving from ASM to C. We start with a description of ASM, including its benefits and disadvantages, and we finish with the benefits of C. Our definition of ASM includes any assembly language starting with ASM51, all the way through to IBM390 assembly.

ASM Description

ASM is a low-level programming language. ASM's structure is very close to the structure of computer hardware. Writing code in ASM requires a large amount of time and strict attention to detail. ASM is usually used in embedded programs where the programming structure is very close to the hardware details. This is especially true when fast turnaround in execution is required, i.e.: interrupt routines, high speed numerical calculations, and other tasks that require repeated execution and therefore use large amounts of computer time without optimization. Because ASM requires programming so close to the hardware details, it has a higher propensity for bugs when any of those details are missed. This makes ASM a very verbose language that requires many lines of code for even a simple task.

Benefits of ASM

Very close to the hardware

ASM is a programming language whose structure is very close to the hardware of the computer. Every detail of the computer hardware can be controlled using the programming language. For example, ASM360 has 360 different instructions (which is where the old IBM 360 computer got its name). Most micro-controllers today use fewer than 255 instructions. For example the 8051 microcontrollers use ASM51, which has approximately 117 instructions.

ASM is very fast

Because the programmer has a wide variety of instructions to choose from, he can select the instruction or instructions needed to minimize execution time. This

is good for the small, critical parts of a program, such as interrupt service routines, because it allows for the fast processing of similar objects, and the access of difficult hardware details.

Disadvantages of ASM

Low programming yield

It takes a long time to develop embedded systems using ASM because of the large amount of code required. With micro-controller code memory now approaching megabytes, the number of ASM statements you need to write can approach hundreds of thousands lines. You need to be very creative to build large systems with the small building blocks that ASM offers. Each ASM line can produce between 1-3 bytes for an average of 2 bytes in an 8 bit micro-controller and an average of 4 bytes in an 16-bit micro-controller. This means that in order to produce 1 megabyte of code, you need to program 500,000 lines for the 8 bit micro-controller and 250,000 lines for the 16 bit.

Slow to test and debug

This large volume of code necessitates a larger amount of time test. The complexity of checking all the possible usages of the different registers will add to the time needed to test and validate the program.

Prone to bugs

The large number of lines in the program guarantees the generation of a large number of bugs. The complexity involving the different registers will add its share of bugs because ASM does not check register usage.

Hard to maintain

ASM is hard to document because of the volume of code. It is difficult to edit without the possibility of introducing additional bugs. If changing a routine, the programmer needs to check not only for bugs but also for additional side effects for all users of the routine. Assembler does not provide any bug checking except for syntax.

No portability

Because ASM's structure is so close to the computer hardware, no two ASM programs are the same. Therefore, moving an ASM program from one computer to the other inherently requires a rewrite.

Large number of instructions

ASM requires significant time to master because of the large volume of instructions available.

Benefits of C

Higher level language

C has a small number of instructions to learn when compared to ASM. It has high level constructs for flow control, which allow the programmer to build basic concepts.

Higher yield

The higher level flow control instructions make management of the complexity of the program faster. In addition, there is easy use of subroutines and no need to worry about register allocation. C allows the programmer to focus on algorithm design.

Easier to debug

C is easier to debug because it uses a smaller number of lines, and allows the use of symbolic debuggers.

Easy to maintain

Maintenance is easier because the C compiler provides checking.

Portability

C source code is portable across many computers because programs that were written for one computer can be recompiled and used on another computer.

Types of translation

Concept translation

The concept type of translation converts an ASM statement (one line or few lines) to a C functional equivalent statement. The primary goals of this type of translation are to extract the business rules embedded in the program and to convey the concept and functionality of the ASM code, rather than precise implementation. Ultimate readability of the translated program is the main objective.

Simulation Translation

In the simulation translation, each ASM statement is translated to a precise C statement or a C function call that simulates precisely the behavior of the ASM statement. A run-time library in C is provided that can be compiled for any target. The goal in the simulation type of translation is to enable the programmer to run legacy programs on newer hardware without any modification. Readability of the resulting code is secondary.

Tools to use

Micro-Processor Services, Inc. has a number of software translator tools, which incorporate both concept and simulation translation, that enable you to move your ASM source code to C. Here are some of the Assembly to C porting tools and their functions:

- ASM51C – converts assembly for Intel 8051, 8052 and other to C
- MASM2C – converts Microsoft MASM assembly to C
- RASM2C – converts Motorola RASM01 RASM05 RASM09 RASM11 assembly to C
- ASM68C –converts ASM68xx ASM6800, ASM6801, ASM6805, ASM6809, and ASM6811 assembly to C
- ASM68K2C – converts Motorola ASM68K ASM68000, ASM 68010 ASM68020 and ASM68040 assembly to C
- ASM862C – converts all Intel ASMx86, ASM86, ASM186, ASM286, ASM386, ASM486 assembly to C
- TASM2C - converts Borland assembly to C
- ASM360C – converts IBM 360 assembly to C
- ASM370C – converts IBM 370 assembly to C
- ASM390C – converts IBM 390 assembly to C
- PPC2C - converts PowerPC assembly to C