PNL-SA-7604
CONF-790454--6

MASTER

A MICROPROCESSOR PROGRAM DEVELOPMENT SYSTEM*

Thomas J. Mathieu
Pacific Northwest Laboratory
Richland, Washington

*Prepared for United States Department of Energy
Under Contract EY-76-C-06-1830

DISCLAIMER

1.0 Introduction

Microprocessors and microcomputers are rapidly becoming more and more prevalent today. Applications for them are coming from the areas of instrumentation, control, communications, and entertainment systems. While the power and complexity of such devices has greatly increased, the cost has greatly decreased. As a result, a new, inexpensive, flexible approach to system design has arisen.

Unfortunately, program development systems for microprocessor applications have not kept pace with the technological advancements. The process of creating programs and getting them running on a microprocessor is often cumbersome and time-consuming. The facilities of a file system, text editor, assembler, compiler, linker, and loader which are normally available on larger computer systems are often deficient or non-existant on a microcomputer system.

One solution to this problem has been accomplished by implementing the microprocessor development system in a high level language and running it on a minicomputer. This enables the microprocessor development system to take advantage of the system facilities provided by the minicomputer. The implementation consists of a cross-assembler and linker written in Pascal for the INTEL 8086 microprocessor and runs on a DEC PDP-11/70.

The paper describes microprocessor development systems and their capabilites in general, the INTEL 8086 and its development system, in particular, and the benefits reaped from using a programming language such as Pascal for the implementation.

2.0 Microprocessor Development Systems

Program development for microprocessors does not, in reality, differ from program development for any computer. A programmer would like to be able to create his programs on a suitably flexible medium, to modify them with realtive ease, to translate them from symbolic form to object form, to link various modules together into an executable image, to load that image into the processor's memory, and to verify that it executes correctly. Normally, however, a microprocessor system does not run an operating system that would facilitate such processes or have access to I/O devices to support them. Fortunately, larger computer systems do have such facilites, and, assuming one is available, certain phases of the program development can be accomplished on one.

A minimum system would operate as follows. Program text creation and modification are accomplished through the host computer's file system and text editing utility. Program modules are assembled by a macro cross-assembler that generates object code for the target microprocessor. Separately assembled modules are linked together by a linker which creates an executable image for the microprocessor. This image can then be down-line loaded into the target microprocessor for testing. Such a minimal system is flexible and easy to use and provides the development capabilities necessary.

A maximal system would include the cross-assembler and linker from the minimal system. It also should provide a high level language compiler, such as Pascal, which generates code for the target microprocessor. Such a compiler would facilitate programming effort and provide greater portability of programs. The maximal system should also provide an online symbolic debugging facility. This would consist of a debugging emulator for the target microprocessor and enable the programmer to 'run' his programs on the host computer and detect and correct many errors before he down-line loads the program.

A final important criterion for microprocessor development systems is flexibility. Due to the variety of microprocessors available and their features, utilization of one microprocessor for all applications would be very inefficient: certain microprocessors are better suited to certain applications than others. Therefore, to utilize several microprocessors, several development systems should be available. And the ability to modify one development system to accomodate a new microprocessor should be inherent in the design of each module. It is for this reason that the programming language Pascal was chosen to implement the system for the INTEL 8086.

3.0 The INTEL 8086

The INTEL 8086 is a new microprocessor that has the capability to perform 16-bit operations. The basic motivation behind this is to enhance system performance by overcoming the limits of 8-bit instructions. The 8086 architecture consists of four general purpose 16-bit registers that can also be addressed as eight 8-bit registers, two 16-bit memory base pointers, two 16-bit index registers, and four 16-bit segment registers which enable programs to address up to 1-megabyte of memory.

The instruction set provides the capabiltiy to address operands in several ways: directly via an 8-bit or a 16-bit offset, or indirectly with base and/or index registers added to an optional 8-bit or 16-bit displacement. In addition memory references and operations can be done to 8-bit or 16-bit values. There are zero, one, and two address instructions and an automatic stack facilitiy. This wide variety of operand formats and operational modes makes the 8086 a very powerful, flexible processor.

4.0 The 8086 Development System

The current implementation of the 8086 program development system is a minimal system consisting of two programs: a cross-assembler and a linker. Both are written in Pascal and run on a DEC PDP-11/70.

The cross-assembler is a two pass macro relocating asembler allowing full expressions in operands, macros with basic parameter substitution, global references and declarations, the basic set of pseudo operations to enable control of the instruction address counter, constant declaration, and listing format, and the full set of 8086 machine instructions. The instruction and operand formats supported are those used by INTEL. In fact, the most significant complexity in the assembler is due to the complexity and wide variety of options in the operand specifications allowed. Fortunately, this complextity is easily and clearly dealt with in Pascal.

The main data structures in the assembler consist of a statements file which holds the statement images and intermediate data between the two passes, a dynamic linked list symbol table containing the symbol name, type, and value, and an array of operation codes containing the operation name, type, and numeric opcodes.

The structure of the program consists of several utility procedures, an initialization procedure, and the pass one and pass two procedures. The utility procedures provide the capabilities of constant conversion, expression evaluation, operand analysis, symbol table manipulation, and basic statement scanning. The intializtion sets up the various tables and values and enters default symbols into the symbol table. Pass one reads the input source file and is concerned with macro definition and expansion, statement parsing and classification, symbol definition, and storage allocation. Pass two produces the program listing and the object code file. There is also an intermendiate pass which, because position dependent displacements can cause inefficient code generation, scans the statements file, detects any statements which can be shortened, and adjusts the address of each symbol and statement accordingly.

The linker is also structured as a two pass program. It reads a command file which indicates which object files to link. The first pass scans each object file, builds a symbol table, and assigns storage to each module. The second pass builds the final program image, patches the global references into the text code, and produces a load map.

The final module in the 8086 program development system is the down-line loader. It loads the final program image into the 8086 via an I/O utility which uses a special device handler to load microprocessors as output devices.

5.0 Summary

The utilization of this microprocessor development system nas greatly increased tne programmer productivity of microprocessor system engineers. The 8086 assembler and linker nave been very easy to use, reliable, and efficient. Moreover, the ease of adding new features has proven to be a valuable asset of the system.

A few words must be said about tne value of the approach taken to this project. The structure of both progams was efficiently and clearly implemented with the constructs provided by Pascal. Errors were easy to detect and correct and improvements in the original design were easily accomplished. Finally, it should be noted that the original 8086 assembler has been modified to assemble code for two other microprocessors. The modifications necessary were straight-forward and involved primarily the opcode table, operand evaluation, and the code generation for machine instructions. These procedures were well isolated and documented. And the actual modifications for both microprocessors took less than a week and were accomplished by a programmer who nad no previous experience with Pascal. This is a very positive testimony to the value of modular programming and Pascal's ease of use.