AN APPROACH TO REAL-TIME MICROPROCESSOR PROGRAMMING

J. SARBÓ and G. RÁCZ

Department of Instrumentation and Metrology, Technical University H-1521 Budapest

Received January 13, 1983
Presented by Prof. Dr. L. SCHNELL

Summary

Actually, programs for microprocessor based devices are mostly written in assembly language. Application of CDL, a high-level programming language and a software technology system developed at the Department of Instrumentation and Metrology, Technical University, Budapest, will be presented. A system generating a real-time monitor (CAL) and its application to CDL programs will be described.

The intelligent measuring instruments are classed into three groups according to the produced series numbers:

1. Medium and great series	100 pieces and over
2. Small series	10 to 100 pieces
3. Unique	1 to 10 pieces

At present, the lengths of programs necessary to their control used to be:
1. 100 to 1000 lines,
2. 1000 to 5000 lines,
3. 5000 to ... lines, respectively.

In Hungary, great many second category devices are being developed. Their hardware and software exclusively are developed by engineers. The great majority of device programs are prepared in assembly language, and are run in a blank computer or under some kind of individually developed monitor. It is characteristic of these programs that the part describing the realtime structure can be separated definitely from the sequential algorithms. Also the sequential part can be divided into two kinds of algorithms: the seldom run data processing and control part making up most of the program; and the programs serving interruption of short execution time. The cross development technology which is usually based on some small computer (PDP-11, TPA 1140, etc.) or possibly on a development system and "cross-assembler", is widely used for developing sequential algorithms.

Devices in category 2 actually consist of a 8-bit microprocessor (I8080, Z80, M6800, ...) as a rule. In this category the device programs are about

1000...5000 lines long, involving the preparation of an about 2.5...12.5 Kbyte program beside e.g. the 2.5-byte average instruction length for the Z80, and assembly programming supposed. According to our experience the preparation of programs of this size and adaptation to real-time environment generally mean a serious problem. It may be ascribed to the qualitatively increased complexity of bigger programs, to the assembly language practically unfit for drafting, and the missing technology.

What are the most important problems of assembly programming for us an answer expected from the application of high-level language?

1. Modifiability:

simple error recognition and error correction (maintenance), program modification (new functions).

2. Understandability:

comprehension, possibly alteration of the program on the basis of the source-test and documentation (if exists) long after having finished the program development.

3. Safety of programming:

close estimation of programming time, with special regard to debugging of the program.

Obviously, the problems above are in close relation to the power of expression, the self-documenting ability and level of the applied programming language.

At the same time application of a high-level language raises problems already solved in assembly programming. These are mainly questions connected with the program's efficiency:

1. Program size
2. Run time (execution speed).

With a microprocessor device both can be problematic. For a device produced in small series (10 to 100 pieces) or in high numbers, it is not the same, how much of memory has to be accommodated. On the other hand, the devices work in real-time environment, there may be time-critical parts which require programs of suitable speed.

Because of the real-time environment, eventual parallel processes may require a monitor for synchronization and mutual exclusion. The monitors are generally of unique development, although the same monitor — more or less altered — may be used in different devices within the same developing environment. This fact, however, little affects uniqueness of the monitors which can be tested only to a degree by the relatively narrow range of users (formal verification would be too complicated to ponder even its possibility).

Microprocessor software-technology system

The technology system applies cross-method in every phase of the program development, i.e. not only for translating the programs but also for their algorithmic verification. It is an essential condition that in the technological system debugging of the program can be done on the high level of the sourse-language (rather than on the level of computer code).

The system is based on the CDL (Compiler Description Language) programming language. The CDL permits modular programming, supports structured programming, step by step refinement. At a difference from the traditional programming languages, the CDL is an open-ended language, that is, it contains little elementary algorithms. To prepare a program, in every case a "language" suitable for describing the given problem has to created, contrary to the traditional languages where the problem has to be expressed in the given language. Thus, learning the way of thinking required by the CDL may last long.

The CDL program contains computer-independent and computer-dependent parts. The computer-dependent algorithms (CDL macros) are composed in assembly language. Adaptation of the program means to transcribe the computer-dependent parts into the syntax of the target processor. One part of the macros may be common between certain programs, transferable from libraries or else. According to our experiences, the ratio of macros — which can be adopted from the library — may be 70 to 90% for system programs. The use of library macros does not limit the open-endedness of the language, the missing macros have to be uniquely written in every case.

From the system's aspect portability of programs written in CDL is the most important characteristic of the CDL, i.e. they can be ported from one type of processor to the other one without efficiency loss. This characteristic of the language permits to do the time-consuming debugging phase of the program development in a PDP-11 computer. The memory size, speed and great number of peripheral units of PDP-11 are more efficient supports of development than is a microprocessor development system.

The process of microprocessor program development is shown in Fig. 1.

The main phases of development are:

1. Preparation of program in the host (PDP-11) computer.
2. Debugging of program in the PDP-11 computer.
3. Preparation of the algorithmically correct program for Z80 (transcription of computer-dependent parts).
4. The program's test in Z80.

As the target processor (Z80) is less suitable for software development, it is much simpler to test the transcribed version of an algorithmically correct program, and to find errors put it in the adaptation phase than to discover errors of a still unknown (perhaps wrong) program.

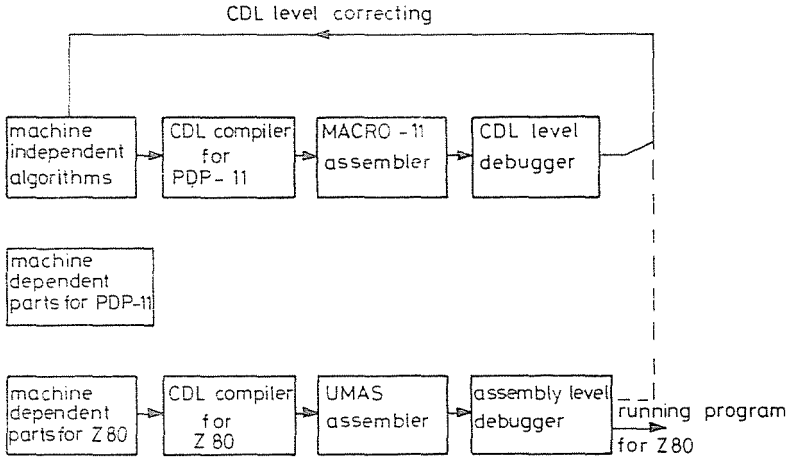

Fig. 1. Process of Microprocessor Program Development

Real-time programming system

One of the elements of software technology system developed at this Department is the CAL realtime programming language. The CAL is in fact a macro-library defined by UMAS (Universal Macro Assembler) offering high-level languages, up-to-date means for formulating real-time structure of the given problem, as well as for avoiding timedependent errors. Synchronization of parallel processes is done essentially by semaphores, a linguistic means is given for mutual exclusion, etc. within the frames of the language, there is a possibility to define programs processing interrupts, as well.

Segmentation of the microprocessor device program and giving real-time structure are made in CAL, but sequential algorithms including programs handling interrupts can be written in any of the programming languages. In this respect let us make two remarks:

1. CAL is a macro library with a limited checking power of type and structure, usual in high-level real-time programming languages (PASCAL, MODULA, ADA, etc.)

2. Owing to the macro-level of CAL assembly, sequential algorithms are the simplest given in assembly. But there seems no obstacle to write the algorithms in CDL or other high-level language, if adaptation of different parameter-transfer mechanisms and runtime systems at the connection points of the high-level language and CAL is provided for.

The CAL gives a possibility for modular programming. The efficiency of sequential programming may be improved by using available module-libraries.

Typical library moduli are the following:

— arithmetical module
— data-processing module
— peripheral-handling module

Fitting CDL and CAL parts

Fitting CDL-program and monitor has been realized at assembly level. The CAL supports it definitely, and in CDL the services of operating system or monitor have been accessible through macros. Also a possibility of CDL-level fitting emerged. But the two programs are correlated not only in control but in certain cases also in data activating a CDL-subroutine from the monitor would require simulation of the CDL parameter-transfer mechanism depending on the implementation. The data relation had also to be solved for assembly-level connection. To this aim, CDL macros writing the CAL data into CDL-globals (i.e. variables) have been defined. This solution does not depend on the given implementation, although it impairs the CDL program, more exactly its hiding-degree.

The most of problems are caused by the CDL in fitting CDL- and CAL-structures. Reacting on the events in the real-time environment the CAL monitor starts the right process activating thereby the (sequential) handler algorithm written in CDL. A solution can be imagined where each algorithm and the respective definition are written as independent CDL programs. This leads to a largesize program likely to comprise certain program parts several times. In this way the optimizing capability of the CDL compiler is missed.

The whole CDL program can only be optimized if all the algorithms are kept in one program. As the CDL is a high-level language, the compiler controls strictly the program closeness, hence the variable definition must be compiled together with the program. The optimum case would be to keep the variable definitions belonging to the processes (RAM) and the access algorithms in the monitor, mutually excluding the access to data of separate processes. But the CDL does not directly support such a separation of data and algorithms. A compromise solution of keeping the CDL-level data belonging to the real-time processes in the CDL program independently compiled by CAL monitor has been chosen. The assembly-level data of CAL monitor are mapped onto CDL globals by means of the presented macro mechanism. As these globals are not visible to the monitor, they are exempt from mutual exclusion, so the CDL program may impair them. (Fortunately, the number of these kinds of globals is low 0 to 2 in each process).

The CAL has only UMAS version at present, the language itself, however, is processor-independent. It has no equivalent PDP-11 version that is why the cross-development technology is irrelevant to the control of real-time characteristics of CDL programs.

The technology does not permit to control the real-time behaviour on the PDP-11, seeming to us to be anyhow needless, indeed. The real-time structure can be controlled in itself, helping to screen out real-time programming errors. If the real-time structure is wrongly chosen, in the majority of cases there is not help to rewrite the program.

Problems concerning execution time, memory size cannot be ulteriorly solved for programs written in assembly language. Programs, however, written in CDL can also be tuned from the aspect of time and memory size., for example, by formulating critical parts as macros or by applying a more suitable run-time system.

References

1. KOSTER, C.H.A.: A Compiler Compiler Matematisch Centrum Amsterdam, MR 127 (1971)
2. Concurrent Assembly Language (CAL) Programming Handbook, Technical University Budapest, Department of Instrumentation and Metrology
3. HANÁK, P.—RÁCZ, G.—SARBÓ, J.: Microprocessor Software Technology Conference on Programming Systems 81, Szeged (1981)

János SARBÓ
Gábor RÁCZ
} H-1521 Budapest