# The Programming Language Modula-2 and its Environment


**Günter Dotzel and Klaus Moritzen**
**ModulaWare.com**

## On standard library definitions based upon different environments

*Abstract:*

Modula-2 is not just a language. It comes with a complete environment supporting software module interface definition, version control, compatibility check and source code level debugging. This paper serves as an introduction to the Modula-2 environment and gives some background information about the different releases and new developments.

*KEYWORDS:* **Modula-2, Programming Environment, Module Library, Native Code Implementations: PDP-11, Professional 350, RT-11, Modula-2/XM, Share-11, RSX-11M, VAX-11/VMS, M68000, i8086/88, MS-DOS, CP/M-86.**

*Introduction:*

Modula-2 was developed at the ETH-Zürich, Switzerland under the direction of Niklaus Wirth (Institut für Informatik) [WIRT83]. Modula-2 is the powerful heir of Pascal and extends this language by the following concepts:

"1. The *module* concept, and in particular the facility to split a module into a *definition part* and an *implementation part*.

2. A more systematic syntax which facilitates the learning process. In particular, every structure starting with a keyword ends with a keyword, i.e. properly bracketed.

3. The concept of the *process* as the key to multiprogramming facilities.

4. So-called *low-level-facilities* which make it possible to breach the rigid type consistency rules and allow to map data with Modula-2 structure onto a store without inherent structure.

5. The *procedure type* which allows procedures to be dynamically assigned to variables." [WIRT83]

If someone compares Modula-2 and Pascal, he will recognize, that there are fewer language constructs and concepts in Modula-2. Everything that could be expressed and efficiently implemented in Modula-2 itself, was omitted. This includes terminal input-output, operating system and file system interface, mathematical library functions, heap management and so on. Standard library modules written in Modula-2 provide these features. This makes Modula-2 easy to learn and allows small and efficient implementations for microcomputers and personal workstations.

*Releases:*

The **Modula-2** system was released in April 1981 for DEC PDP-11 computers. Some corrections concerning the run time system (emulator of not existing machine instructions) and some improvements were made by ETH-Zürich and released in September 1981. In April 1983 the structure debugger was released. Until now Modula-2 compilers for the following computers, û-processors and operating systems are available: DEC PDP-11 (RT-11, Share-11, Star-11, RSX-11M, Unix), DEC VAX/VMS, Lilith (Modula Machine, bit slice computer executing M-Code), M68000, M6809, i8086/88 (MS-DOS, CP/M86) ... implemented by different persons and institutes all over the world.

The Modula-2 & LIDAS (see below) license agreements from the Institute für Informatik of the ETH-Zürich were updated in summer 1983 and do now allow sublicensing (license agreement and handling charge).

**The Modula-2 implementations show some slight differences.** Lets have a closer look at the Modula-2 Environment of the DEC® PDP-11/RT-11 implementation which was the first one. The so-called M2RT11 system is assumed to be the most wide spread implementation, since it is operational on all PDP-11 machines ranging from -11/04 to -11/73 and now also on DEC's *personal computer Professional 350* (-11/23 microcomputer chip set) under the RT-11 operating system.

Here let's study the debugger, the compiler and its options, library modules, an extended implementation for very large programs, file types, main storage and mass storage requirements, typical run time errors and the recommended user reactions, a little more in detail. The environment

description is intended to be an introduction, but it also gives some detailed information and hints for Modula-2 users.

*Modula-2 Native Code Implementation on PDP-11 under RT-11*:

The Modula-2 distribution kit for PDP-11/RT-11 includes compiler, linker, loader, debugger, utilities like intermediate file decoders, cross reference generator, conversions for simple types (except for REAL's), process scheduler, all programmed in Modula-2. The kit includes more than 25,000 lines of Modula-2 source code. The run time system (RTS) is very small (about 1.5KByte) and is programmed in Assembler (MACRO). The RTS provides trap decoding for stack test, copy routines for fixed and variably sized data, synchronous and asynchronous coroutine switching, saving and restoring operating system vectors, emulation of non-existing hardware and interpretation of illegal or reserved instructions.

The resident monitor consists of the modules SystemTypes, Exceptions, TTIO, Files, Loader and ResidentMonitor and is about 5KByte. The command interpreter is not resident.

The *distribution kit* from ETH-Zürich does not provide a possibility to use existing system libraries (e.g. SYSLIB.OBJ or FORLIB.OBJ) nor does it offer library modules for trigonometric functions, real to string conversions, double precision floating point arithmetic, 32 Bit integers, software for driving the video screen VT-52 or VT-100 (e.g. cursor addressing, terminal independent input line editing), channel oriented input-output for simple types other than CHAR, time and date I/O conversion, support for 60 Hz (crystal) clock time interpretation.

The programmer should use only software written in Modula-2, to gain the full advantages of the system-wide consistency control and debugging feature. Modula-2 is a real-time language. Real-time languages must be reentrant (keywords: same code, multiple data, instantiation) and the current process pointer must not be destroyed. In fact, there is no guarantee, that library modules written in any other language, obey these strong conventions. It is better to avoid this problem than to solve it. If you think that you can't live without some routines of the SYSLIB, PASLIB or FORLIB, look for the routines you need in the chapter "Modula-2 User Library Modules" below.

*Symbolic Structure Debugger:*

The first version of the symbolic debugger DEBUG did not decode REAL's, ARRAY's or RECORD's. The decoding of REAL's was implemented in summer 1981 by the author.

The new symbolic structure debugger DBUG by Georg Maier (ETH-Zürich) was released in April 1983 and replaced DEBUG. DBUG is a very useful post-mortem dump analyser. It decodes all structures, simple types, structured types and process variables, i.e. displaying the variable names used in the Modula-2 source modules and their values according to their types and in the case of a process type, it allows to switch to another process context. DBUG allows walking around through the user data structures (linked lists, trees).

DBUG knows the module names of your program. This information is stored (with about 15% mass storage overhead) at the end of an executable load file. This part is not read by the loader.

DBUG knows the values of your data: if a run-time error occurs, the resident system writes the contents of the main storage to a special environment file: DUMP.COR.

DBUG knows the symbolic names of your structures: The reference file (.REF), produced by pass 2 of the compiler contains the necessary informations. The syntax of the reference file was extended to support the structure decoding (new module M2CREF.MOD). A reference file decoder completes the distributed version.

*Modula-2 under the Share-11 Operating System*:

Share-11 is the fastest multi-user extension to RT-11 for PDP/LSI-11s. It is the most complete RT-11 compatible system and allows realtime applications also. Share-11 is easy to install and requires no system generation. It uses any combination of standard mapped and unmapped RT-11 handlers and allows you to mix 18 and 22 bit controllers with impunity. Handlers are loaded into kernel space and not into user space.

The Modula-2 environment used under Share-11, is actually identical to the RT-11 environment. The proper assignments must be performed in the system's startup files of each user (SHAREx.COM): since several programmers or jobs can operate the Modula-2 compiler at the same time now, each job

has to use its own work files for the compilation process and its own DUMP.COR file for the debugging process (logical device VS). The file lookup strategy for different devices (DK and SY in the standard environment) is further supported by Share-11 in an optimal way through the .ASSIGN/PATH command: libraries can now be grouped on logical disks (LD or VD). The .SET TT COMMAND feature allows command files with program parameters also.

The standard Modula-2 run time system intercepts software traps with its own trap-handler for the emulation purposes and run time error handling, but it does not tell the operating system anything about that. This would cause serious problems in a multi-user environment. The trap-handler was modified by the author: The programmed request .TRPSET is used for the interception of illegal memory reference, emulation of direct PSW-access, time-out, illegal or reserved instruction and interpretation of FIS-instructions with FPU (vectors 4 and 10) and .SFPA for the interception of floating point unit trap (244). Thus core-dump and debug features will now operate under Share-11 too.
The IOTRANSFER feature of Modula-2 together with direct I/O-page access can not be used under Share-11 currently. This is the only limitation.

*Modula-2/XM for the PDP-11/RT-11*:

A Modula-2/XM version for PDP-11/23 with hardware memory management unit (MMU) under RT11SJ (single job) operating system is available from Technische Universität München since December 1983. It is an upgrade to the Modula-2 M2RT11 version from ETH-Zürich. It is compatible with the normal implementation, i.e. no recompilation is necessary. Neither the language nor the Modula-2 compiler were modified. Modula/XM uses one page addressing register (PAR) of the MMU and allows 2MBytes of program code, up to 44KByte of user data in low memory and I/O-Page access. Linker, loader, debugger and the RTS were extended. Existing programs and modules must not be recompiled. Only relinking is required.
*Virtual Arrays:* Procedures are available for the handling of additional large virtual arrays in the 22Bit address space (4MBytes).
*Modula-2 is "**ROMable**":* The Modula/XM linker separates code and data and allows [E|P]ROMable code production. A "PROMer" module is available.

*Modula-2 User Library Modules:*

Many useful library modules have been added to the Modula-2 library by the author (and later ModulaWare™) since 1981, mainly to support **real to string** conversions and vice versa, **MathLib** and new I/O-concepts, namely UNIV, TIRF and CIO. *UNIV* incorporates parametric I/O-procedures for elementary data types and strings, *TIRF* (*t*erminal *i*ndependent *r*etyping *f*eature) serves for input line editing and *CIO* is a channel oriented ASCII-I/O-interface for sequential files (devices).

TIRF was integrated into the user library and into all commonly used programs. The TIRF concept stems from the environment of the programming language EDISON (developed for the LSI-11 under direction of Per Brinch Hansen, University of Southern California). The EDISON System implements a unique retyping technique for system monitor commands, data entry and the video-terminal editor. Only four control keys are used for line editing: back-space (BS), delete (DEL, rubout) tabulation (TAB, HT), and carriage return (CR, RET). BS moves left, TAB moves right, DEL deletes the character at the current cursor postion, and CR terminates input string editing. As a central feature in the Modula-2 environment, this feature is implemented by the module *ReadString*.

*Channel Oriented Input-Output, CIO*:

The module *CIO* offers a channel oriented user input-output interface. CIO supports reading and writing of characters, integers, reals, and strings from and to any device (channel or stream) in ASCII form. CIO defines a hidden type CHANNEL. The procedure Channel connects a device to a CHANNEL. The procedure DeChannel disconnects a CHANNEL. For CIO we show only two procedure definitions, *Channel* and *ReadR*:

```
TYPE CHANNEL; (* hidden type *)

PROCEDURE Channel (dfn: FileName; NewFileOrDevice: BOOLEAN;
  VAR c: CHANNEL; VAR ChannelOK: BOOLEAN);

  (* Connect an ASCII stream of dfn (device, filename, extension) to channel c; IF
  NewFileOrDevice THEN a new file dfn is created, ELSE an existing file is connected.
  ChannelOK reflects a good result (file found, or new file properly created). *)

PROCEDURE ReadR (c: CHANNEL; VAR r: REAL; VAR IsNumber: BOOLEAN; VAR
  NextChar: CHAR);

  (*Read a floating point number r from a previously "channeled" ASCII file or device c.
  The termination character is asssigned to NextChar. The variable IsNumber reflects
  whether a legal number was read or NOT. *)
```

*Example for CIO*:
This example shows the data definitions and a statement for reading a floating point number from a channel (file or device):

```
VAR x: CHANNEL;
r: REAL; good: BOOLEAN; ch: CHAR;

Channel ("DK Test DAT", FALSE, x, good); (* channel to DK:TEST.DAT *)
IF NOT good THEN (* error, file not found *) END;
ReadR (x, r, good, ch);
```

*Universal-Input/Output Concept, Univ*:
CIO is based upon the Universal-I/O-Concept: Every procedure that reads (writes) characters is parameterized by a procedure. This procedure reads (writes) one character at a time. The types

```
TYPE ReadProc = PROCEDURE ( VAR CHAR ); WriteProc = PROCEDURE ( CHAR );
```

are provided by the module UnivTypes. Further separate modules are available:

| | |
|---|---|
| UnivInput: | Reading integers |
| UnivOutput: | Writing integers, cardinals, strings and new lines |
| ASCII: | ASCII character set definition |
| UnivRInput, UnivROutput: | Reading and writing of floating point numbers |
| ReadString: | Reading strings with variable termination procedure |
| Retype: | String display and retyping |
| UnivDate: | Writing date and system-time |
| ReadFileName, ReadOptions: | Reading file names and program options |
| UnivLongIO: | Reading and writing of LONG (32 bit) integers or cardinals in decimal or octal notation |

In the case of *ReadFileName*, the default file name, typed fields (device, name, extension) and file name syntax conventions remain in effect.

The type LONG is provided by the module **LongTypes** and the associated operations for calculation and comparison are provided by the module **LongOperations**.

It is now possible, for example, to read a floating point number or even a file name from system console, from a channel or stream, or from a remote device and so on. The module *UnivRInput* serves as an example for a procedure definition in the Universal-I/O-Concept:

```
DEFINITION MODULE UnivRInput ;
  FROM UnivTypes IMPORT ReadProc, WriteProc;

  EXPORT QUALIFIED ReadReal;

  PROCEDURE ReadReal (Read: ReadProc; Write: WriteProc;
    VAR r: REAL; VAR isnum: BOOLEAN; VAR TermChar: CHAR);
  (* Read a floating point number r in free format, using Write for echoing. If the system
    console is used (e.g. TTIO.Read, TTIO.Write are substituted for the parameters Read and
    Write), the input string can be edited with TIRF.
    The variable isnum reflects whether a legal number was read or NOT.
    The termination character is assigned to TermChar; the standard terminators are <CR>,
    <LF> or <ESC>. *)

END UnivRInput.
```

*Example for Univ*:

The following statement reads a floating point number with the procedure UnivRInput.ReadReal from system console using the module TTIO:

```
VAR r: REAL; good: BOOLEAN; ch: CHAR;

ReadReal (TTIO.Read, TTIO.Write, r, good, ch);
```

*Mathematical Functions, MathLib:*

A MathLib written in Modula-2 offers the set of trigonometric functions offered in Pascal implementations. An alternate implementation is written in PDP-11 assembler (FP-11 code), provides the same performance as the FORTRAN IV (FPP) library version (e.g. the function *sin* needs 1.7ms on PDP-11/23 with FPU-microcode chip KEF-11AA and 0.34ms with hot floating point processor FPF11) with the following definition:

```
TYPE RealFunction = PROCEDURE (REAL): REAL;
```

and procedure variables at fixed locations in the RTS:

```
VAR sqrt, exp, ln, sin, cos, arctan: RealFunction;

VAR Random: PROCEDURE (VAR INTEGER, VAR INTEGER): REAL;
```

If a function is called with an illegal parameter value (e.g. s:=sqrt(r), if r<0.0), a floating point arithmetic error is generated and a post-mortem dump is produced to allow debugging. The function Random (i,j) gives a value between 0 and 1, like the FORTRAN function RAN(i,j).

These library modules and the screen-editor *MEd* (see below) are the main components of the so-called *ModulaWare-2 Environment*.

*Compiler Options:*

The Modula-2 compiler has options for /[*N*O]LIST, /*V*ERSION, /03, /40, /RT11, /UNIX. In May 1983 the compiler options were extended: /FIS (default) or /FPP for floating point code generation (FP-11 hardware support). This option determines the pass 4 file used for p4: M2CP4.M2C (FIS) or M2CP4R.M2C (FPU) (this method was implemented by the author).

The improvements from the ETH-Zürich in September 1981 were delivered in hardcopy source form, showing the differences between the old and the new files:

Improvements for the FP-11 Hardware: the code generation now uses one to four floating point accumulators (AC0 .. AC3) for code generation (default AC0, AC1). There was a little bug in module M2CR4F.MOD (procedure PushRealValue: TSTF generated on ":= 0.0" assignment). This error was corrected immediately by the author. The new efficient symbol file interface needs further simple corrections in the module M2CP2.MOD (procedure ExportList and procedure ImportList). The error occurs only at recompilation of M2CP1.DEF and M2CP2. To avoid this error use the old symbol file interface or work out the appropriate modifications in M2CP2.MOD.

*File Types:*

The Modula-2 environment creates a lot of file types (extensions).

.MOD Modula-2 source code for modules or implementation modules
.DEF Modula-2 source code for definition modules
.SYM compiler output of .DEF
.LNK compiler output of .MOD
.REF compiler output used by the debugger DBUG
.LST compiler output, list file
.LOD linker output, executable file
.MAP linker output, map file
.DEC decoder output of DECREF, DECLNK, DECLOD
.XRF XREF (cross reference generator) output

*The Modula-2 environment uses the following file types:*

.M2S linker base file, the command interpreter and run time system

.M2C compiler interpass (intermediate work) files and compiler overlays. The M2CP4R.M2C is not used on a machine without FP-11 (FPU). The interpass files must be on the logical device VS and each Modula-2 user (under Share-, Star-eleven) has to use another device or logical disk. The compiler overlays M2C*.M2C must be on the logical device VC. The M2CSYM.M2C must be on SY:

.COR Core dump file, used if a run time error occurs, read by DBUG

COMP.LOD the compiler base (TIRF)

LINK.LOD the linker (TIRF)

DBUG.LOD the debugger

LIST.LOD lists a file on system console (TIRF)

DIR.LOD lists directory on system console (TIRF)

XREF.LOD produces a numbered listing .LST and a cross reference .XRF

(The following files and file types belong to the screen editor *MEd (ModuleEditor)* for DEC's VT-52 or VT-100 terminal family. MEd is not part of a standard Modula-2 release kit, but it is used widely on Modula-2 installations. MEd provides all features of the well-known UCSD p-system editor and is available from ModulaWare as part of the Modulaware-2 Environment for a nominal handling charge.)

MED.LOD screen editor base (TIRF), sometimes renamed to ED.LOD

.M2E screen editor overlays must be on logical device VE

.MOB backup source file from .MOD, created by MEd

.DEB see .MOB, for .DEF files

.MO description file (one block) of .MOD, created by MEd for use by MEd

.DE see .MO, for .DEF files

*Mass Storage Considerations:*

The smallest useful system configuration is a PDP-11/03 microcomputer with RT-11SJ and RX02 floppy disks (2 * 500KBytes). On the system-disk (DY0:) are the following files: SWAP.SYS, RT11SJ.SYS, TT.SYS, DY.SYS, PIP, DUP, DIR, RESORC, *.M2C, *.M2S, Mxx.SAV, COMP.LOD, LINK.LOD, and an editor (all together about 950 blocks). The compilation and program loading speed can improved significantly through fine tuning of the Modula-2 program loader.

The USR should be set to NOSWAP, and a 30KWord RT-11 system is recommended.

On the user-disk DK: (DY1:) are the following files: .SYM and .LNK of the library (including CIO about 150 blocks), and the user files .DEF, .MOD, .LST, .REF, .SYM, .LNK and .LOD. The DBUG.LOD can be kept on system-, user- or on a separate user-disk.

The virtual memory handler VM.SYS (DECUS or RT-11V05) should be used if a PDP-11/23 processor and MMU (KT-11) are available. The compiler intermediate work files IL1, IL2 should be at least 40 blocks (50 blocks are enough for compilation of modules with more than thousand lines of source code), the compiler work file ASCII should have at least 10 blocks. The compiler overlays and intermediate files .M2C can be kept resident in VM if there are 256KByte of main storage available. If 768KByte (or even more) main storage is available, then the RT-11 system should be resident in VM (booted from VM). If even a hard disk is available then do not worry about mass storage and system speed.

*PDP-11/RT-11 Modula-2 Environment*: **ERROR MESSAGES and REACTIONS**

*COMP, MED: file not found*

The overlays .M2C (COMP) or the overlays for the editor .M2E (MEd) are not available on the logical device VC: (COMP) or VE: (MEd) or the approriate handler is not loaded. Make the assignments and load the handler(s).

*DBUG: illegal reference file syntax*

Use the new pass 2 overlay M2CP2.M2C generated after April 1983. Recompile your .MOD files needed for debugging.

*DBUG:*

After a program crash the main storage will be dumped to SY:DUMP.COR. If the debugger does not know the file name of the .LOD file (crashed program) after reading DUMP.COR, the information for the last loaded file is not available. The start address of the file name should be located at @504 in your resident monitor Mxx.SAV (M03, M04, M23, MS). At our installation 012372 (8) must be patched into this location once. If the address is not available to you, type the file name of the program producing the run time error (possibly an overlay!). Always use the same name for file name and module name (the first 6 characters are used).

*Trap to 10:*

An illegal or reserved instruction was executed. Use the appropriate run time system Mxx.SAV, to emulate FIS-EIS hardware. Recompile your modules using the module priority specification with the appropriate compiler switch: /03 or /40. This option determines the mode of PSW access via MTPS, MFPS or direct program status word (PSW) access. Some machines (11/04, 11/44) do not know the instructions MTPS, MFPS.

*Trap to 4:*

An illegal address was used: non existing hardware (status or command register address) or direct PSW access on 11/03. The PSW access on 11/03 is optionally patched at run time (M03.SAV) on first execution. This feature allows the generation of portable real-time Modula-2 programs.

*Input-Output error:*

The output file is full or end of input file occurred or the device is full or the device is scattered and must be squeezed or the directory segments are full or the handler is not loaded.

Initialize your diskettes with at least 8 segments (16 segments on double density, double sided floppy disks recommended) using the command .INIT DY1:/SEG:8. (.INIT DY1:/SEG:16). Do never operate the linker or xref on a device with directory overflow. One directory segment has up to 72 file entries (including deleted files) and up to 32 segments per device or logical disk can be created. One segment consists of 2 blocks (1KByte). The command .DIR/SUM can be used to examine the directory segments.

**In the next chapters we will outline differences of other Modula-2 implementations from the RT-11 implementation**:

*PDP-11/RSX-11M Native Code:*

The normal distribution kit consists of a small module library and task files for compiler and utilities [OLSE84]. It does neither contain the Modula-2 source code for compiler nor the source code for its environment. The new structure debugger DBUG is not yet supported. Since emulation trap-handling is not very efficient under the RSX multi-user operating system, a different technique

was used for the emulation of not-existing hardware, i.e. hardware floating point unit (FPU). The compiler was modified to import the module *M2Real*, if no FPU is available.

The module SYSTEM does not provide the features IOTRANSFER, SYSRESET, LISTEN, RSX-CALL. RSX operating system calls are provided by the module *rsx*. Since RSX uses not-contiguous files, the file system interface module *fsys* supports *sense* and *point* operations for random positioning in files and procedures for reading (writing) a record (ARRAY OF WORD) to (from) files. The module *NewIO* can be used to re-direct the current input-output stream. The automatic file scan feature was extended by a catalog file, containing up to ten sub-directory specifications. Remark: This feature should be provided on operating system level (see Share-11 feature ASSIGN/PATH).

*VAX/VMS Native Code:*

The successful system integration for the VAX implementation allows system library calls from Modula-2 (trigonometric functions, real to string conversion and vice versa, double precision floating point arithmetic and so on) [SCHM82]. This is a feature of the VAX architecture with generalized subroutine calls. The type ADDRESS is compatible with WORD (32 bit). The standard I/O library does not yet support simple data type I/O conversions for file channels (CIO). The stream concept however allows to read or write characters, bytes, words, or short words from or to a channel and random positioning to a byte address (module FileSystem, implemented by P. Putfarken).

The real time (system dependent) language feature IOTRANSFER is not supported on VAX Modula-2. The Modula-2 debugger is not implemented. The symbolic VAX/VMS debugger can be used. The different performance of these debuggers are not discussed here.

Recently the VAX family was extended by the MicroVAX, a Q-Bus machine. Of course the MicroVAX will be a target machine for a wide range of Modula-2 applications.

*Modula-2/68K Native Code:*

Modula-2 for the Motorola M68000 û-processor was delivered until now in source form only [BURK83]. You need another Modula-2 environment (e.g. SMILER2 cross compiler on CDC) for bootstrapping and you have to reprogram the Modula-2 interface for your operating system (e.g. File-System, Terminal). The type ADDRESS (32 bit) is not compatible with the type WORD (16 bit). The

structure debugger is in still in preparation. The run time system (SYSTEMX) is written in Modula-2 itself, using CODE procedures (machine code insertion facility). Since January 1984, a "code-release" kit is available, to simplify the bootstraping process. It allows the substitution of the target operating system specific parameters and avoids the need for bootstrapping the Modula-2 compiler on a cross compiler system.

*Modula-2/86 Native Code:*

The Modula-2 implementation for the i8086/88 comes with a very small set of library modules (CardinalIO, InOut, Processes, RS-232 serial line driver) [LOGI83]. The operating systems supported are CP/M-86 and MS-DOS. The minimum requirements for a (very slow) development system are two 384 kByte disk drives and at least 256 kBytes of memory. 2MBytes of mass storage are recommended. The run time support (written in assembly language) occupies 6 kByte (2 kByte without interface to CP/M or DOS). The resident monitor consists of the modules Display, Keyboard, Terminal, TerminalBase, Storage, CommandInterpreter, FileSystem, ProgramLoader, and System and is about 23KByte. An automatic library scan is not implemented, since the operating system does not return control to a user program if a file lookup operation failed (input-output error). For files which are not on the default disk, the user has to specify device and file name.

The linker allows separation of code and data to produce PROMable code and generates position independent (relocatable) load files. The loader was extended by this feature and this results in a slower load time. Due to this feature the heap manager (Storage) is part of the resident monitor. This has consequences in the heap management on overlays and processes. Note that every program is basically an overlay.

The type ADDRESS is not compatible with WORD (as is on PDP-11 or VAX-11). Address arithmetic is restricted to addition and subtraction. These operations are performed in the run time system (RTS). The priority specification in the module declaration is *ignored* by the compiler. The IOTRANSFER feature allows only a limited number (typically 8 or 16) of interrupt vectors to be installed. An IOTRANSFER operation requires 2.5ms per interrupt service routine pair, TRANSFER needs 1.2ms on an i8086 CPU running at 5MHz and using memory without wait states. A faster process switching mechanism by means of the operating system is not possible, because the RTS is not reentrant. The coroutine switching is slow, compared with the PDP-11 implementation, where IOTRANSFER consumes 140ûs and TRANSFER about 50ûs on a PDP-11/23 (3.3MHz CPU micro cycle).

Terminal I/O may be redirected through the use of procedure variables. The file system concept of the Lilith (Modula Machine) is used. New file structured devices are easily integrated due to the procedure variables concept. The type REAL (i8087 hardware) is not supported yet, but announced for March 1984. The structure debugger is supported.

*Modula/R and LIDAS:*

For informations regarding the Lilith Database System, the relational database concept and its language extension of Modula-2, so-called Modula/R, which was released in June 1983, we would like to refer to [REIM83], [ZEHN83], [DOTZ84].

*Acknowledgement:*

The author would like to thank the Institut für Informatik, ETH-Zürich for the proper preparation of the documentation and the release kit of Modula-2. Thanks are due to Hans-Georg Daun for proof-reading.

*REFERENCES*:

[BURK83]: Burkhardt: Modula-2/68K Release 2.0 Information, Institut für Elektronik und Informatik, ETH-Zentrum, Zürich, Switzerland, Dec-83.

[DOTZ84]: Dotzel, Günter; Moritzen, Klaus: The Relational Data System LIDAS - A database system for micro-computers in Modula-2, ModulaWare, Schwalbenweg 12, D-8520 Erlangen, F.R.G., Apr-1984.

[LOGI83]: Logitech S.A.: Modula-2/86 Users Guide (Preliminary), Apples, Switzerland, Rev. 0.3, Sep.-1983.

[OLSE84]: Olsen, Peter: User's Guide to the BBC Modula System for PDP-11 with RSX-11M, Release 3.0, Brown, Boveri & Co. LTD., Dept. ESL-22, Baden, Switzerland, Jan-1984.

[REIM83]: Reimer, Manuel: Modula/R Report -Lilith Version-, Research Project LIDAS, ETH-Zürich, Febr. 1983.

[SCHM82]: Schmidt, J.W.; Koch, J.; Mall, M.; Putfarken, P: System-dependent Facilities of Modula-2 on the VAX-11, Universität Hamburg, F.R.G, June-1982.

[WIRT83]: Wirth, Niklaus: Programming in Modula-2, Springer Verlag, 1982, 1983 (2nd. ed.).

[ZEHN83]: Zehnder, Carl August (Ed.): Database Techniques for Professional Workstations, Institut für Informatik, ETH-Zürich, Switzerland, ETH-Report Nr. 55, Sept. 1983.