Modula-2 Implementation Overview

Pieter H. Hartel and Dolf Starreveld

Vakgroep Informatica, Universiteit van Amsterdam, Nieuwe Achtergracht 166, 1018 WV Amsterdam, The Netherlands

An account is given of the efforts which have led to an implementation of the programming language Modula-2. The implementations are classified according to various criteria. Some compilers are "back-ended" by different code generators for widely used micros and minis and are structured such that code generators may be added or adapted relatively easily. Other compilers produce binary code for one or more target machines directly and are more difficult to adapt to special requirements.

With a number of Modula-2 implementations, many interesting program development tools for special and general purpose applications are provided.

Measurements have been performed to obtain an indication of the efficiency of the code as generated by various Modula-2 compilers.

INTRODUCTION

Modula-2 has been described as a solution to Pascal's problems [1]. Although many of Pascal's problems are solved indeed with Modula-2, some problems remain [2]. Modula-2 is the successor to Pascal and as such has inherited many of Pascal's characteristics [3]. The major difference between the two languages is perhaps the design philosophy. Pascal was intended for educational purposes, whereas Modula-2 is a true systems programming language. The important module concept has given Modula-2 its name. In Modula-2, modules may be compiled separately. The compiler performs cross module type checking. Other languages that have influenced the design of Modula-2 are Mesa [4] and Modula [5]. Of the members of the Pascal family, ADA* is the most advanced, complete (yet complex) language. With respect to the latter, Modula-2 is a much more compact language [6,7] for which currently more implementations exist. Not many ADA compilers have been released so far, and those which have been mostly implement a subset of the ADA language.

REFERENCE DOCUMENT

The Modula-2 programming language is described in the book "Programming in Modula-2" by Prof. N. Wirth [8]. The book contains an introduction to the language and a report. Unfortunately, some aspects of the semantics of the language have been left unspecified in the book [2]. Other books have recently been published such as Gleaves, *Modula-2 for Pascal Programmers,* Springer-Verlag: New York, 1985, and Ford, Wiener, *Modula-2: A Software Development Approach,* Wiley: New York, 1985.

AVAILABILITY

Modula-2 implementations have been realized by several universities and commercial firms. Within the academic world, the compiler and associated software are usually distributed in source form. Commercial implementations of the language are distributed in binary form in almost all cases.

The distribution of Modula-2 compilers is sub-

*ADA is a registered trademark of the U.S. Department of Defense.

Journal of Pascal, Ada, & Modula-2, Vol. 4, No. 4, pp. (9–23) (1985)

CCC 0735-1232/85/040009-15$04.00

ject to a license agreement. The licensee may use the compiler for any of its requirements, but may not charge its customers for "software cost." The licensee may distribute the compiler to third parties under the same conditions under which he obtained it.

The licensee is not allowed to alter the Modula-2 compiler in such a way that the language accepted by the compiler is changed. Extensions to the language are not considered changes. It is, however, strongly recommended not to make any extensions to the language.

IMPLEMENTATIONS

The principal component of any Modula-2 implementation is the compiler, which translates a Modula-2 source program into intermediate or executable code. On some implementations, code for the target machine is generated directly (either in binary or in assembly source language). Other compilers generate a form of intermediate code, such as M-code or P-code. The M-code compilers are derived from the major Modula-2 compiler at the ETH, which was developed for the Personal work-station Lilith [9,10]. The Lilith architecture is that of a stack machine, with a micro-programmed M-code interpreter. The M-code generating compilers are back-ended by a target machine-specific code generator, with the exception of the system used on Lilith itself. In all cases, object modules are bound by linkage editor to form executable images.†

Most Modula-2 implementations provide their own linkage editor. The major reason is that the standard linker of most operating systems cannot verify the coherence of a system of separately compiled modules.

With some of the Modula-2 implementations, a symbolic debugger is provided, which may be used to trace, breakpoint, and debug a Modula-2 program.

As the definition of I/O has been left out of the Modula-2 language proper, all implementations provide library modules to perform input and output.

Institut für Informatik ETH (IFI)

The first Modula-2 compiler was developed at the ETH Zürich for the RT-11 operating system on a PDP-11.† It was used successfully to develop the Lilith operating system Medos, even before the Lilith machine was actually built. The compiler is based on a recursive descent parser. The address space limitation of the PDP-11 forced the designers into a five-pass compiler. This compiler, written in Modula-2, is still available (M2RT11).

The second compiler (M2M) developed at the Institute für Informatik generates M-code for the Lilith machine. This compiler is also written entirely in Modula-2. The compiler is split into four different main passes which execute as subsequent programs.

With the Medos operating system, a comprehensive set of library modules is provided for mathematical functions, input/output, and window management for the Lilith bitmap display.

The Modula-2 debugger on the Lilith machine makes extensive use of these window management capabilities.

Institut für Informatik
ETH Zürich
Clausiusstrasse 55
CH-8092 Zürich
Switzerland

The Lilith machine is marketed by:

R. Ohran
Modula Computer Systems
940 N University Avenue
Provo, Utah 84604
U.S.A.

Both the RT11 and M-code compilers are available from the above-mentioned address.

Rechenzentrum ETC (RZETH)

At RZETH in Zürich the original Modula-2 compiler from IFI for the PDP-11, called M2RT11, was transliterated into Pascal 3 for CDC NOS/BE systems. The result of this translation was a compiler, which cross-compiles Modula-2 programs into binary code for the PDP-11.

While extending the syntactical part of the compiler, code generators were added to the cross-compiler to produce binary code for the Motorola MC6809 and the Motorola MC 68000 micro-processor family (MC 68000, MC 68010, . . .).

A cross-linker, again written in Pascal 3, links separately compiled modules with the module containing the runtime support procedures. The linker generates LDA format binary for the PDP-11 or S-

†The Lilith architecture is such, that compiled object modules may be executed on the system without prior linkage editing. However, a linkage editor is available under Medos, which may be used to bind a number of modules before loading. This will speed up program loading considerably.

†DEC, PDP, VAX, and VMS are registered trademarks of Digital Equipment Corporation.

records format text for the Motorola micro-processors.

The cross-system, called SMILER-2, includes cross-compilers, a cross linker, dis-assemblers, a (minimal) runtime system for each of the target processors, and some very basic library modules.

SMILER-2, as a genuine cross-system, is intended for stand-alone applications and for bootstrapping compilers written in Modula-2 itself.

The package is distributed by RZETH in source code only.

Mr. H. Seiler
Rechenzentrum ETH
ETH-Zentrum
CH-8092 Zürich
Switzerland

Institut für Elektronik ETH (IFE)

At the Institut für Elektronik of the ETH in Zürich a Modula-2 compiler is developed for MC 68000-based systems (MODULA-2/68K). Part of the compiler was derived from the M2RT11 compiler, whereas the code generator was transliterated from Pascal into Modula-2 using the code generator of the RZETH cross-compiler SMILER-2. The compiler and support system run under control of an RT-11-like operating system written for the MC 68000. Code may be generated both for applications which run under control of the operating system or for stand-alone applications [11,12].

The system has been set up such that it can be transported to other systems relatively easily. For a successful port a cross-system, such as the RZETH SMILER-2, is required. The interface from the MODULA-2/68K system to the underlying operating system is specified by only 5 definition modules. In order to bootstrap the system, the implementation modules of these must be written.

Dr. H. Burkhart
Institut für Elektronik
ETH-Zürich
Gloriastrasse 35
CH-8092 Zürich
Switzerland

University of New South Wales (NSW)

The Modula-2 compiler for seventh edition UNIX* on the PDP-11 was implemented at the University of South Wales in Australia. The compiler, library, and support programs are largely based on the RT-11 implementation from IFI. The compiler generates object modules in a format which is incompatible with the standard UNIX object module format. The Modula-2 linker serves to bind object modules. A special loader program will convert a set of bound object modules into an executable image (a.out). The Modula-2 debugger can be used to analyze a post-mortem dump. The latter is written in a format which is incompatible with the standard UNIX dump file format (core). The set of library modules provide an (almost complete) interface to the UNIX system.

Dr. J. Tobias
Department of Computer Science
University of New South Wales
P.O. Box 1
Kensington N.S.W. 2033
Australia

DEC Western Research Laboratory

An experimental Modula-2 compiler is available from the Western Research Laboratory of Digital Equipment Corporation for use on the VAX under Berkeley Unix 4.x [13]. The compiler is licensed to universities in the United States for internal, noncommercial use only.

The compiler allows convenient access to the Unix environment, permits linking of Berkeley Pascal and C programs with Modula-2, and has a simple optimizer that produces code comparable to the best compilers for Modula-2 and other languages on the VAX. It supports language extensions that allow the programmer to control the size and alignment of data types, and defines a simple I/O facility similar to the C "printf" and "scanf" routines.

Michael L. Powell
Digital Equipment Corporation
Western Research Laboratory
4410 El Camino Real
Los Altos, CA 94022
U.S.A.

*UNIX is a registered trademark of Bell Laboratories.

University of Cambridge

A very elegant implementation of the Modula-2 programming language was developed at the University of Cambridge in England for use on the VAX under UNIX 4.1 BSD. The compiler is based on the M-code compiler for the Lilith machine. The M-code generated by the compiler is converted by a separate code generator program into VAX/UNIX assembly language. The assembly language modules may be assembled by the standard assembler (as). Program linkage is done ultimately by the standard UNIX linker (ld). However, a special Modula-2 linker must be invoked to discover the complete set of modules which constitute the program being built. From this information, the Modula-2 linker constructs the global frame table for the program being linked. The global frame table is used for address computations during external references (i.e., from one module to another) [9].

Included in the implementation is a comprehensive set of library modules, a GKS (version 7.0) [14] implementation with drivers for various devices, a set of program development tools (such as a "make-file" generator for Modula-2 programs) and an interactive symbolic debugger for Modula-2 programs.

The Modula-2 implementation as developed at the Computer Laboratory for use under seventh edition UNIX for the MC 68000 is very similar to that for the VAX. The major difference lies in the code generator, which produces assembly language code for the MC 68000. Of the 131 modules, which together constitute the (M-code) compiler (M2M), only 2 are different from those on the VAX.

During the last two years, three new compilers have been developed which share a common front end. The M-code interface to the code-generators has been replaced by a tree-based representation of the programs being compiled. Code generators are available for the VAX (using the procedure calling standard), the ICL Perq using C-code under PNX and the GEC 4090. The code generated by the new compilers is much more efficient, both in execution time and in memory requirements. For debugging purposes, the global frame table is still maintained. It is no longer used for external access and procedure calls.

Similar compilers for the IBM 3081 under Phoenix/MVS and for the National Semiconductor 16032 under Unix are being developed, but are not yet available.

Dr. P. Robinson
University of Cambridge Computer Laboratory
Corn Exchangestreet
Cambridge CB2 3QG England

Acorn Research Inc.

Acorn is using Modula-2 as a systems programming language and has a compiler for the National Semiconductor 16032 under development, using the same (tree-based) code generation strategy as described in the previous section. This compiler will be made available in due course.

Dr. M. J. Jordan
Acorn Research Inc.
Suite 910
5 Palo Alto Square
Palo Alto
CA 94306
U.S.A.

University of Nottingham

The Modula-2 compiler and support system as developed at the Psychology department of the University of Nottingham are particularly suitable for stand-alone real-time applications on the LSI-11. Extensive support for Modula-2 programming has been developed under RT-11 and seventh edition UNIX. The work was based on the Modula-2 compiler for RT-11 from IFI-ETH and the compiler for seventh edition UNIX from New South Wales.

Dr. R. B. Henry
Human-computer Interaction Group
Department of Psychology
Nottingham University
Nottingham NG7 2RD
England

Universität Karlsruhe

Work originally started at SIEMENS in Munich to port the RZETH cross-compiler SMILER-2 to a (MC 68000-based) UNIX version 7 system. This system was completed at the University of Karlsruhe.

The system provides full support for floating-point arithmetic. The linker, which has been developed at Karlsruhe, performs version control, linkage of modules written in C, partial linking, and transformation of the .lnk format to the UNIX a.out format.

Dr. D. Schwarz
Fakultät für Informatik
Universität Karlsruhe
Postfach 6380
D-7500 Karlsruhe 1
W-Germany

Universität Dortmund

At the University of Dortmund, the NSW compiler for seventh edition UNIX has been adapted to generate code for stand-alone applications. The runtime nucleus has been extended in several ways. Also, a runtime performance monitoring package has been developed using a coprocessor.

Dr. W. Kuhnhauser
Informatik III
Universität Dortmund
Postfach 500500
D-4600 Dortmund 50
W-Germany

Brown Boveri & Cie

BBC has developed a Modula-2 system on the basis of the M2RT11 compiler. The system runs under control of RSX-11M/S, as well as under control of RSX-11M-PLUS. The Modula-2 system is available in two variants:

- A basic Modula-2 system, which includes the compiler, linker, symbolic post mortem dump analyzer, some utilities, and a set of interface modules to RSX. Modula-2 programs may only be executed under the control of the resident Modula-2 monitor.
- The Modula-2 kit contains, in addition to the Basic Modula-2 system as described above, a facility to link assembly programs with Modula-2 programs, as well as a more comprehensive set of utility modules. Using the Kit, Modula-2 programs may be built as stand-alone applications, or for execution directly under control of RSX.

Both software packages are distributed in binary form only.

Dr. J. Muheim
Department ESL
BBC Brown Boveri & Company Limited
Werk Turgi
CH-5401 Baden
Switzerland

University of Virginia

At the Department of Biomedical Engineering, the M2RT11 compiler has been adapted for use under RSTS/E V8.0. The M2RT11 compiler itself is nearly unchanged. It generates native PDP-11 code, that will run in RSX emulation mode. The operating system support has been designed such that the compiler and runtime system will be easily portable to RSX11M-PLUS.

Modula-2 programs either may be executed directly under control of the operating system, or under the control of the Modula-2 resident monitor. The latter provides some basic services to running Modula-2 programs, such as the interpretation of exceptions and generation of dump files for use by the symbolic post-mortem dump analyzer, etc.

The Modula-2 package includes the compiler, linker, symbolic post-mortem dump analyzer, the resident monitor, the Modula-2 command interpreter, an RSX task file generator and a set of utility modules.

T. Breeden
Department of Biomedical Engineering
University of Virginia Medical Center
Box 377
Charlottesville, VA 22908
U.S.A.

Universität Linz

In 1982, work was started at the University of Linz to implement Modula-2 on 8086-based systems with INTEL's ISIS-2 operating system. A completely new compiler has been written both in Pascal-86 and Modula-2. The Pascal-86 version was needed for the purpose of bootstrapping. The Modula-2 compiler has been designed such that it may be run on small systems with 128 KB of memory.

A Modula-2 programming support environment is under development. The development system will include a Modula-2 structure-oriented editor, a dynamic symbolic debugger, project management tools, and document preparation tools.

Prof. Dr. P. Rechenberg
Institut für Informatik
Johannes Kepler
Universität Linz
Altenbergerstrasse 69
A-4040 Linz Auhof
Austria

Dr. G. Pomberger
Institut für Informatik
Johannes Kepler
Universität Linz
Altenbergerstrasse 69
A-4040 Linz Auhof
Austria

Logitech

A commercial Modula-2 development system for the 8086 is available from Logitech, a Swiss/American software house. The development system runs under control of MSDOS 2.0,* CP/M, or MP/M.† It is distributed for use on the IBM PC, Victor/Sirius machines, or any machine which supports 8″ single-density CP/M format floppy disks. A symbolic debugger is included in the distribution package as well as the source modules required to adapt the system to specific configurations [15].

The same product is also available as a cross-compiler, which executes under control of VAX/VMS. The compilers are fully compatible at all levels.

Mr. W. Steiger	Mr. A. Gorrengourt
Logitech SA	Logitech Inc.
	165 University Avenue
CH-1143 Apples	Palo Alta CA 94301
Switzerland	U.S.A.

Logitech's implementation may also be obtained from:

Springer Verlag Software
Heidelberg-New York-Tokyo

Volition Systems

From Volition Systems a Modula-2 development system, is available for use on machines which support Apple or UCSD* Pascal. The Modula-2 compiler generates P-code, which is interpreted by the Apple or version II UCSD Pascal systems [16].

With the Modula-2 distribution from Volition Systems, a set of software tools is available which emulates the Unix shell programming environment (P-shell).

Miss T. Barrett
Volition Systems
P.O. Box 1236
Del Mar CA 92014
U.S.A.

Volition Systems' implementation may also be obtained from:

Springer Verlag Software
Heidelberg-New York-Tokio

*MSDOS is a registered trademark of Microsoft Inc.

†CP/M and MP/M are registered trademarks of Digital Research Inc.

*UCSD is a registered trademark of the Regents of the University of California.

Universität Frankfurt / Main

The Modula-2 compiler for the VAX/VMS operating system was originally developed by Dr. Schmidt's research group at the Fachbereich für Informatik, Universität Hamburg [17]. The compiler is maintained by the same research group, now located at the Fachbereich Informatik, Universität Frankfurt/M. The compiler generates VAX/VMS object modules, which may be bound (optionally with object modules generated by other VAX/VMS compiler) to form an executable image. The VAX/VMS linker provides full support for linking separately compiled modules. The VAX/VMS symbolic interactive debugger may be used with Modula-2 programs (the Modula-2 compiler generates symbolic information for the debugger in a format similar to that produced by the VAX/VMS Pascal compiler).

The prime distributor of the VAX/VMS implementation of Modula-2 is Logitech (see the section entitled "Logitech"). However institutions primarily engaged in education and research may also contact:

Dr. J. W. Schmidt
Fachbereich Informatik
Universität Frankfurt/Main
Dantestrasse 9
Frankfurt/Main
W-Germany

Rekencentrum Vrije Universiteit Brussel

At the Rekencentrum VUB, the RZETH cross-compiler SMILER-2 has been adapted to the NOS 1.4 operating system for the CDC Cyber. Various utilities, such as a Pascal–to–Modula-2 translator and a manager for separately compiled modules were developed at the computer center [18]. A complete set of floating-point and 32-bit integer arithmetic routines has been added to the runtime support for the MC 68000. Code generators were added to the cross-system for Z80, Z8002, and M-Code. An M-code interpreter (written in Pascal), which runs on the Cyber computer, is available.

Another cross-system has been developed for the MC 68000. This work was based on the NSW compiler, to which the MC 68000 code generator from the SMILER-2 cross-system was added. The complete compiler is being ported to an MC 68000-based Unix system.

Currently, work is being done to include the MC 68000 code generator pass into the VAX/VMS Modula-2 compiler, such that this system may also be used for the development of software for the MC 68000.

For the Cyber computer, a native code generator is under development.

Mr. F. Maene
V.U.B. Rekencentrum
Pleinlaan 2
B1050 Brussels
Belgium

CERN

At the DD division of CERN in Geneva, the RZETH cross-compiler SMILER-2 was ported to IBM/370 under MVS. In addition to the code generators for the MC 6809 and the MC 68000, a code generator is available for the TMS 9900. The system is back-ended by a comprehensive linkage and library management system, which operates on modules in CUFOM format (a universal object format).

At present, a new Modula-2 compiler is being developed by Dr. D. Foster. This is essentially a two-pass compiler. The second pass is shared with the (CERN version of) the Siemens Pascal compiler for the MC 68000. The first pass takes care of all Modula-2 specific syntax and semantics. The object code is produced in CUFOM format. As soon as the new compiler will become operational, the (CERN version of) the SMILER-2 compiler will no longer be supported.

From CERN, a number of other compilers are available, which also produce CUFOM modules.

Dr. J. D. Blake
DD Division
CERN
CH-1211 Geneve 23
Switzerland

Burroughs

A cross-compiler has been developed running on the Burroughs 6800. The compiler generates an intermediate code which may be interpreted on the host. Also, an alternative code generator pass of the compiler has been developed to generate native code for the MC 68010 or MC 68020. On the Motorola processors, programs may be run under control of a specially developed operating system or in stand-alone mode.

The system is currently not available outside the Burroughs company.

Mr. R. Jones
Software Engineering Department
Burroughs Machines Ltd.
Castle Cary Road
Cumbernauld
Scotland

BENCHMARK

A criterion for comparing the quality of language implementations may be obtained by measuring the time required to execute the code generated for certain language constructs. Another criterion may be the time required to compile and run entire programs [19,13]. The benchmark program of the appendix as proposed by N. Wirth allows for 15 different tests to be performed. Each test is run for exactly one minute (real time) and the number of "loops" performed during this period is taken as the measure of the test. Although this benchmark test is rather limited, the data obtained may indicate the performance of the generated code and the machine on which it is run. Even though the performance of the raw hardware in each case may be (very) different, comparisons between benchmarks on different machines are meaningful, since most applications will be written in high-level languages rather than machine language. The number of high-level language statements executed per unit of time is probably the best measure of the performance of a system.

Some of the Modula-2 implementations described earlier were benchmarked on various machines as shown in Figure 1. On time-sharing hosts, no other users, batch jobs, or background processes other than those supporting the operating system were present during the benchmark tests. On all systems, enough real memory was available to hold the entire benchmark program during execution.

As the benchmark program does not use any language features which are not available in Pascal, the program was transliterated into Pascal and timed on various machines. This provides some interesting data to compare the implementations of both languages. The systems which were benchmarked using Pascal are also shown in Figure 1.

The data obtained from the benchmarks are represented in Figure 2. Not all entries in the table are filled, as some target systems do not provide all facilities for which benchmark tests are included.

Nr.	target	OS	Compiler	Bits*	Remarks
1.	PDP-11/40	RT-11	ETH-IFI	16	
2.	PDP-11/45	Unix V7	NSW	16	no FIS EIS
3.	PDP-11/70	Unix V7	NSW	16	no FIS EIS
4.	PDP-11/70	RSTS/E V8.0	Virginia	16	with FPP
5.	PDP-11/70	RSTS/E V8.0	Swedish Pascal	16	with FPP†
6.	Lilith	Medos	ETH-IFI M-code	16	
7.	Z80A	stand-alone	VUB		3 MHz
8.	6502	UCSD-P	Volition	16	Apple IIe
9.	MC 6809	stand-alone	RZETH	16	1 MHz
10.	MC 68B09	stand-alone	RZETH	16	2 MHZ
11.	MC 68000	stand-alone	RZETH	16/32	8 MHz
12.	MC 68010	stand-alone	RZETH	16/32	8 MHz
13.	MC 68000	stand-alone	VUB	32	10 MHz
14.	MC 68000	Unix V7	Cambridge	32	Microproject
15.	MC 68000	HP 9121	HP Pascal	16	Hewlett Packard
16.	MC 68000	Munix	Siemens Pascal	32	PCS Cadmus 9000
17.	MC 68000	Munix	Karlsruhe	32	PCS Cadmus 9000
18.	MC 68000	Workshop	Lisa Pascal	16	6 Mhz
19.	MC 68000	Macintosh	MacPascal	16	Interpreter
20.	VAX-11/750	4.1 BSD	Cambridge	32	M-code Emulation
21.	VAX-11/750	VMS 3.2	DEC-Pascal 1.3	32	
22.	VAX-11/750	VMS 3.2	Frankfurt	32	
23.	VAX-11/750	4.1 BSD	Berkeley-Pascal	32	
24.	VAX-11/750	4.1 BSD	ACK Pascal	32	
25.	VAX-11/780	4.x BSD	Cambridge	32	New compiler
26.	VAX-11/780	VMS 3.4	Frankfurt	32	
27.	VAX-11/780	4.2 BSD	Berkeley-Pascal	32	
28.	VAX-11/780	VMS 3.5	DEC-Pascal 1.3	32	
29.	Perq I	PNX	ICL-Pascal	32	
30.	CDC Cyber-730	NOS/BE 1.5	Pascal 3	60/18	
31.	CDC Cyber-875	NOS/BE 1.5	Pascal 3	60/18	
32.	Future FX30	CPM-68	Modula-2/86 Logitech	16/32	8MHz 8088
33.	Future FX30	CPM-68	Pascal MT+/86	16/32	8MHz 8088
34.	Superbrain	CPM	Turbo Pascal	16/16	4MHz 280A
35.	8085A	CPM	Pascal MT+	16/16	3MHz 808SA

*n/m n = number of bits for INTEGER arithmetic.
m = number of bits for ADDRESS arithmetic.
†Swedish Pascal is an implementation of Pascal for the PDP-11, which is available from DECUS.

Figure 1. The systems which were benchmarked using Modula-2 / Pascal.

A number of minor problems were encountered when the benchmark program was ported to various machines.

1) The procedure "BusyRead" cannot be implemented easily on some machines. On those machines, the program was stopped all together after one minute and restarted for the next test. On the Lilith the overhead in calling the procedure "BusyRead" is negligible, so removing the call from the program does not make the results incomparable.
2) On some machines, output is buffered until a complete line has been assembled. In those cases, the tests were run long enough, but at least for one minute, to allow an integral number of lines to be output. Then the results were scaled back to one minute.
3) Some systems are too slow to perform a reasonable number of "loops" during one minute. The number of loops performed during a longer period of time was measured and scaled back to one minute.

BENCHMARK TEST

nr.	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	
1.	184	185	230	84	—	—	54	11	93	21	37	29	11	66	—	
2.	226	227	271	119	—	—	68	18	131	35	49	38	13	78	—	
3.	404	448	483	190	—	—	138	31	232	51	89	69	18	154	—	
4.	571	576	677	221	368	148	174	42	285	81	148	101	27	89	—	
5.	264	247	591	121	155	109	56	34	140	84	142	102	56	—	—	
6.	321	334	422	187	130	87	109	89	197	164	144	94	63	125	207	(harddisk)
7.	30	29	30	—	—	—	7	—	5	4	22	13	13	—	—	
8.	5.8	4.9	6.9	2.0	1.6	0.9	1.5	1.0	2.0	1.6	2.0	1.5	4.4	2.4	—	
9.	91	91	91	2	—	—	19	13	40	28	48	25	4	33	—	
10.	182	182	182	4	—	—	37	26	79	55	95	51	8	65	—	
11.	276	276	320	108	36	44	75	61	130	109	128	79	53	71	—	
12.	281	281	327	116	38	46	75	61	134	113	128	79	76	73	—	
13.	492	492	571	160	57	69	122	100	206	170	241	142	99	123	—	
14.	202	184	215	36	—	—	46	30	43	35	84	44	21	60	—	
15.	348	291	325	45	15	9	60	34	59	49	63	48	43	107	—	
16.	451	370	380	24	48	59	97	56	29	27	115	70	38	127	—	
17.	397	398	463	143	54	19	100	82	176	146	97	74	75	99	—	
18.	701	509	585	102	9	1	95	53	142	110	116	91	43	182	—	
19.	1.0	1.1	6.6	0.7	1.2	1.6	3.7	—	10	—	7.0	4.3	17	—	—	
20.	254	247	284	95	192	—	71	45	80	62	68	50	97	60	19†	
21.	480	382	494	160	261	148	173	36	176	78	66	45	79	149	—	
22.	382	408	481	133	188	150	75	39	135	81	106	60	92	173	—	
23.	401	362	282	109	74	46	50	—	102	—	57	51	75	177	—	
24.	301	286	405	179	66	24	106	—	145	—	39	37	35	158	—	
25.	701	503	612	130	232	—	170	33	105	65	64	54	56	168	—	
26.	773	785	790	178	405	259	135	78	184	109	159	116	96	317	—	
27.	650	595	722	252	478	122	135	—	283	—	144	115	112	339	—	
28.	599	500	710	383	415	254	528	179	233	138	202	72	100	246	—	
29.	295	217	196	97	35	13	44	9	77	21	109	72	52	96	—	
30.	577	489	549	418	845	543	277	147	373	230	88	70	299	222	—	
31.	5k4	4k7	7k8	2k8	7k2	6k4	2k2	1k1	4k6	2k4	511	—	—	—	—	
32.	213	207	260	69	—	—	63	44	73	60	102	58	48	17	25	(harddisk)
33.	221	205	224	68	0.7	0.3	36	7	59	13	80	55	35	16	13	(harddisk)
34.	44	36	104	7	2.0	1.3	16	11	29	21	40	20	12	19	8	(floppy)
35.	35	30	71	5	2.4	1.2	12	1.7	10	3	29	13	4	12	2	(floppy)

*Tests all run with the runtime tests enabled.

†This test was run in an emulated Medos environment as an extra layer on top of 4.1 BSD UNIX. This resulted in extra overhead for those parts of the program involving Input/Output, in particular test "o".

Figure 2. Benchmark test data Modula-2 / Pascal.

DISCLAIMER

This summary was prepared with great care. However no responsibility will be assumed for the correctness of the information. Of those implementations not described here, no information was available at the time of writing.

RECOMMENDATIONS

Modula-2 is particularly suitable for the design of operating systems and real-time software. The large number of implementations summarized above has been used successfully to support an even larger number of programming projects. For Modula-2 to be even more useful, the portability of programs must be improved. The two major sources of problems are:

- Assumptions are often made (maybe inadvertently) on the architecture of the target system. Many problems can be avoided if properly defined constants and types are used in architecture-dependent modules (e.g., during address calculations).
- Although, from the language designers point of view, the omission of input and output facilities makes the language proper more portable, few Modula-2 programs can be ported from one system to another without alteration. The vast majority of application programs written in Modula-2, including the compiler could be written using

a *standard* set of library modules, such as proposed for example in "Programming in Modula-2".

SUMMARY

In the figures below, a summary is given of the implementation efforts. In addition, a price indication* is given of binary and/or a source license. In some distributions, several code generators are included. In general, a single license fee will be charged for such a package.

The types of compilers are classified as follows:

1. Based on the original RT-11 compiler.
2. Based on the M-code compiler.
3. Based on the Pascal version of the RT-11 compiler (SMILER-2).
4. Not based on any of above.

*)n.a. not applicable.
h.c. handling charges.

CONCLUSIONS

Modula-2 implementations are available for many computer systems and more implementations are forthcoming, but the popularity of Modula-2 does not yet equal that of Pascal.

Modula-2 compilers do not necessarily produce less efficient code than Pascal compilers. This is what can be expected if the semantic properties of both languages are compared. From the first generation of compilers for any language, it can not reasonably be expected that the produced code is highly efficient. The first problem is to obtain a working compiler. Once this is done, more effort can be put into generating efficient code. Most compiler discussed above were first generation compilers. The benchmark results seem to confirm this statement.

There is much interest in Modula-2. Work is being done on Modula-2 at over a dozen Universities and an ever-growing number of commercial firms. Within the academic world alone, at least a few hundred Modula-2 compilers have been distributed and are being used for all kinds of purposes. The

Site	Host	OS	Type source	Binary
ETH-IFI	PDP-11	RT-11	1 Sfr 350	n.a.
Nottingham	PDP-11	RT-11	1 n.a.	n.a.
NSW	PDP-11	UNIX V7	1 Aus$ 150	n.a.
Nottingham	PDP-11	UNIX V7	1 n.a.	n.a.
Dortmund	PDP-11	UNIX V7	1 n.a.	n.a.
RZETH	Cyber	NOS/BE	3 Sfr 350	n.a.
VUB	Cyber	NOS	3 US$ 50	n.a.
Virginia	PDP-11	RSX-11	1 n.a.	n.a.
BBC	PDP-11	RSX-11 (Basic)	1 n.a.	Sfr 1000
BBC	PDP-11	RSX-11 (kit)	1 n.a.	Sfr 2500

Figure 3. Modula-2 implementations summary for PDP-11 and LSI-11.

Site	Host	OS	Type source	Binary
ETH-IFI	Lilith	Medos	2 Sfr 350	n.a.
VUB	Cyber	NOS 1.4	4 US$ 50	n.a.

Figure 4. Modula-2 implementations summary for M-code.

Site	Host	OS	Type source	Binary
RZETH	Cyber	NOS/BE	3 Sfr 350	n.a.
VUB	Cyber	NOS 1.4	3 US$ 50	n.a.
CERN	IBM/370	MVS	3 h.c.	n.a.

Figure 5. Modula-2 implementations summary for MC 6809.

Site	Host	OS	Type source	Binary
ETH-IFE	68000	special	2,3,4 Sfr 350	n.a.
Cambridge	Codata 68000	UNIX V7	2,4 UK 100	n.a.
Karlsruhe	PCS 68000	UNIX V7	3 DM 400	n.a.
Volition	Sage 68000	UCSD V2	4 n.a.	US$ 495
RZETH	Cyber	NOS/BE	3 Sfr 350	n.a.
VUB	Cyber	NOS 1.4	3 US$ 50	n.a.
CERN	VAX	4.2 BSD UNIX	4 h.c.	n.a.
CERN	IBM/370	MVS	4 h.c.	n.a.
Burroughs	B-6800	Burroughs	4 n.a.	n.a.

Figure 6. Modula-2 implementations summary for the MC 68000.

Site	Host	OS	Type source	Binary
CERN	IBM/370	MVS	3,4 h.c.	n.a.

Figure 7. Modula-2 implementations summary for TMS 9900.

Site	Host	OS	Type source	Binary
VUB	Cyber	NOS	3,4 US$ 50	n.a.

Figure 8. Modula-2 implementations summary for Z8002.

Site	Host	OS	Type source	Binary
Cambridge	VAX	4.1 BSD UNIX	2,4 UK 100	n.a.
Frankfurt	VAX	VMS	1,4 US$ 200	n.a.
Logitech	VAX	VMS	1,4 n.a.	?
Digital	VAX	4.x BSD UNIX	4 n.a.	US$ 100

Figure 9. Modula-2 implementations summary for the VAX.

Site	Host	OS	Type source	Binary
Volition	Z80,8080	UCSD V2	4 n.a.	US$ 595
Logitech	Z80,8080	UCSD,CP/M	4 n.a.	US$ 495
VUB	Cyber	NOS 1.4	3,4 US$ 50	n.a.

Figure 10. Modula-2 implementations summary for Z80 and 8080.

Site	Host	OS	Type source	Binary
Linz	8086	ISIS-2	2,4 n.a.	n.a.
Volition	IBM PC	UCSD V2	4 n.a.	US$ 395
Logitech	8086	CP/M	4 n.a.	US$ 495
Logitech	IBM PC	MSDOS 2.0	4 n.a.	US$ 495

Figure 11. Modula-2 implementations summary for 8086.

Site	Host	OS	Type source	Binary
Volition	Apple][	Pascal	4 n.a.	US$ 295
Volition	Apple / / /	SOS/Pascal	4 n.a.	US$ 495

Figure 12. Modula-2 implementations summary for 6502.

main area of application seems to lie in the development of stand-alone software for micro-computers.

ACKNOWLEDGEMENT

We would like to thank the implementors of Modula-2: W. Angioletti, J. D. Blake, T. Breeden, D. Budgen, H. Burkhart, H. von Eicken, D. Foster, L. Geissmann, A. Gorrengourt, R. B. Henry, Ch. Jacobi, R. Jones, M. J. Jordan, W. Kuhnhauser, S. E. Knudsen, J. McCormack, J. .Muheim, G. Pomberger, J. W. Schmidt, H. Seiler, W. Steiger, R. Sumner, J. Tobias, N. Wirth, and P. Zimmerman for giving permission to include information about their work.

APPENDIX—BENCHMARK PROGRAM

```
MODULE BenchMark;
   (*$T-
     a: empty REPEAT loop
     b: empty WHILE loop
     c: empty FOR loop
     d: CARDINAL arithmetic
     e: REAL arithmetic
     f: standard functions
     g: array of single dimension
     h: same as g but with index tests
     i: matrix access
     j: same as i but with index tests
     k: call of empty, parameterless procedure
     l: call of empty procedure with 4 parameters
     m: copying arrays (block moves)
     n: pointer chaining
     o: reading of file *)

     FROM  Storage IMPORT ALLOCATE;
     FROM  Terminal IMPORT Read, BusyRead,
           Write, WriteLn;
           InOut IMPORT WriteCard;
     FROM  FileSystem IMPORT
           File, Lookup, ReadWord, Reset,
           Response;
     FROM  MathlibO IMPORT sin, exp, ln, sqrt;

     TYPE  NodePtr = POINTER TO Node;
           Node = RECORD x, y: CARDINAL;
           next: NodePtr END;

     VAR   A,B,C: ARRAY [0 . . 255] OF
           CARDINAL;
           M: ARRAY [0 . . 99],[0 . . 99] OF
           CARDINAL
           m: CARDINAL;
           head: NodePtr;

PROCEDURE Test(ch: CHAR);
  VAR i, j, k: CARDINAL;
  r0, r1, r2: REAL;
  p: NodePtr;

  PROCEDURE P;
  BEGIN
  END P;

  PROCEDURE Q(x, y, z, w: CARDINAL);
  BEGIN
  END Q;

BEGIN
  CASE ch OF
       "a":       k := 20000;
                  REPEAT
                    k := k-1
                  UNTIL k = 0 |
       "b":       i := 20000;
                  WHILE i > 0 DO
                    i := i-1
                  END |
       "c":       FOR i := 1 TO 20000 DO
                  END |
       "d":       j := 0; k := 10000;
                  REPEAT
                    k := k-1; j := j+1;
                    i := (k*3) DIV (j*5)
                  UNTIL k = 0 |
       "e":       k := 5000; r1 := 7.28;
                  r2 := 34.8;
```

```
                REPEAT
                  k := k-1; r0 := (r1*r2) /
                  (r1+r2)
                UNTIL k = 0 |
"f":            k := 500;
                REPEAT
                  r0 := sin(0.7);
                  r1 := exp(2.0);
                  r0 := ln(10.0);
                  r1 := sqrt(18.0);
                  k := k-1
                UNTIL k = 0 |
"g":            k := 20000; i := 0; B[0] := 73;
                REPEAT
                  A[i] := B[i]
                  B[i] := A[i];
                  k := k-1
                UNTIL k = 0 |
"h":            (*$T+ *) k := 20000; i := 0;
                B[0] := 73;
                REPEAT
                  A[i] := B[i];
                  B[i] := A[i];
                  k := k-1
                UNTIL K = 0 (*$T - *) |
"i":            FOR i := 0 TO 99 DO
                FOR j := 0 TO 99 DO
                    M[i,j] := M[j,i]
                  END
                END |
"j":            (*$T + *)
                FOR i := 0 TO 99 DO
                  FOR j := 0 TO 99 DO
                    M[i,j] := M[j,i]
                  END
                END (*$T - *) |
"k":            k := 20000;
                REPEAT
                  P; k := k-1
                UNTIL k = 0 |
"l":            k := 20000;
                REPEAT
                  Q(i,j,k,m); k := k-1
                UNTIL k = 0 |
"m":            k := 500;
                REPEAT
                  k := k-1; A := B; B := C;
                  C := A
                UNTIL k = 0 |
"n":            k := 500;
                REPEAT
                  p := head;
                  REPEAT p := p↑.next
                  UNTIL p = NIL;
                  k := k-1
                UNTIL k = 0|
"o":            k := 5000;
                REPEAT
                  k := k-1; ReadWord(f,i)
                UNTIL k = 0;
                Reset (f)

    END (* CASE *)
END Test;

VAR ch,ch1: CHAR;
    n: CARDINAL;
    f: File;
    q: NodePtr;

BEGIN
  Lookup(f,"anyfile", FALSE);
  head := NIL; n := 100;
  REPEAT
    q := head; NEW(head); head↑ .next := q;
    n := n-1
  UNTIL n = 0;
  Write(">"); Read(ch);
  WHILE ("a" <= ch) & (ch <= "p") DO
    Write(ch); WriteLn; n := 0;
    REPEAT
      n := n+1; Test(ch);
      IF (n MOD 50) = 0 THEN WriteLn END;
      Write("."); BusyRead(ch1)
    UNTIL ch1 # 0C;
    WriteCard(n,6); WriteLn; Write(">"); Read(ch)
  END;
  Write(14C)
END BenchMark.
```

References

1. R. T. Sumner, and R. E. Gleaves, *"Modula-2—A solution to Pascal's Problems," ACM Sigplan Notices,* **17** (9), (Sept. 1982) pp. 28–33.
2. D. Spector, "Ambiguities and insecurities in Modula-2" *ACM Sigplan Notices,* **17** (8), (Aug. 1982) pp. 43–51.
3. P. H. Hartel, "Comparing Pascal and Modula-2 as Systems programming languages," in *Programming Languages and System Design* edited by J. Bormann, North Holland, Amsterdam. 1983, pp. 187–196.
4. J. G. Mitchell, W. Maybury, and R. E. Sweet, *Mesa Language Manual,* Technical Report CSL-79-3, XEROX Corporation, Palo Alto Research Center, 1979.
5. N. Wirth, Modula: A Language for Modular Multiprogramming *Software-Practice and Experience,* **7,** 3–35 (1977).

6. S. J. Young, *Real Time Languages—design and development* Ellis-Horwood Publishers, Chichester (1982).
7. P. H. Hartel, "Le language Modula-2 concepts et expérience" Contribué au: *Forum sur la Micro Informatique en physique nucléaire et physique des particules,* edited by G. Fontaine, Collège de France, Paris (Sept. 1983) pp. 323–331.
8. N. Wirth, *Programming in Modula-2,* Springer-Verlag, Berlin (1982).
9. N. Wirth, *The Personal Computer Lilith* Report No. 40, Institut für Informatik, ETH Zürich (1981).
10. L. Geissmann, J. Hoppe, Ch. Jacobi, S. E. Knudsen, W. Winiger, and N. Wirth, *Lilith Handbook—A Guide for Lilith Users and Programmers,* Institut für Informatik, ETH Zürich (1982).
11. E. Ballarin, H. Burkhart, R. Eigenmann, and H. Kindlimann, *Modula-2/68 Handbook,* Institut für Elektronik, ETH Zürich. (March 1984)
12. H. Burkhardt, M. Moser, C. Yen, "Merging High-level Language and Assembly Software: Principles and Case Study, *Proceedings of the conference on "Programmiersprachen und programmentwicklung",* Zürich, Springer-Verlag (1984).
13. M. L. Powell, "A Portable Optimizing Compiler for Modula-2," *Proceedings of the ACM SIGPLAN 1984 Symposium on Compiler Construction, SIGPLAN notices,* **19** (6), (June 1984) pp. 310–318.
14. "International Organisation for Standardisation," *Graphics Kernel System (GKS)—Functional Description,* ISO/TC 97 draft document (1983).
15. Logitech S. A., *Modula-2/86,* Apples—Switzerland (1983).
16. Volition Systems, *Modula-2 User's Manual,* Del Mar, California (1982).
17. J. Koch, M. Mall, and P. Putfarken, *Modula-2 für die VAX: Beschreibung einer System ubertragung* Contribution to: Langmaack, H.; Schendler, B.; Schmidt, J. W. *Implementierung PASCAL-Artiger Programmiersprachen,* Tagung II/1982 des German Chapter of the ACM, B.G. Teubner, Stuttgart.
18. W. Angioletti, and T. D'Hondt, "User experience with a Modula-2 cross-compiler on a Cyber" Contribution to *Proceedings of the 34 ECCODU conference,* Helsinki, Finland (April 1981).
19. T. L. Anderson, "Seven Modula compilers reviewed" Journal of Pascal, Ada & Modula-2, March–April 1984.