1037 LOS ANGELES CALIFORNIA LASR1 C ** 213/629-1561

December 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number		1		
Average Delay		121.0		
Std Deviation		.0		
Minimum Delay		121		
Maximum Delay		121		

1043 ST LOUIS MISSOURI SL1 C 314/421-5110

June 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number	12	27	2	
Average Delay	800.4	766.9	309.0	
Std Deviation	211.1	212.4	39.0	
Minimum Delay	431	480	270	
Maximum Delay	1124	1347	348	

July 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number	16	83	11	
Average Delay	649.3	679.9	325.9	
Std Deviation	152.9	238.7	53.9	
Minimum Delay	435	243	244	
Maximum Delay	971	1550	420	

August 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number	8	27	1	
Average Delay	660.6	601.9	302.0	
Std Deviation	235.8	209.8	.0	
Minimum Delay	242	268	302	
Maximum Delay	942	1079	302	

September 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number	8	20	2	
Average Delay	569.4	538.7	369.0	
Std Deviation	221.0	228.4	95.0	
Minimum Delay	333	238	274	
Maximum Delay	988	939	464	

October 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number	11	26	2	
Average Delay	517.6	516.3	218.0	
Std Deviation	110.6	168.8	9.0	
Minimum Delay	380	237	209	
Maximum Delay	757	960	227	

November 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00

Number	2	9	1	1
Average Delay	500.5	532.1	258.0	225.0
Std Deviation	85.5	119.7	.0	.0
Minimum Delay	415	320	258	225
Maximum Delay	586	770	258	225

December 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number	4	9	1	
Average Delay	498.0	345.9	294.0	
Std Deviation	157.2	178.6	.0	
Minimum Delay	315	155	294	
Maximum Delay	749	807	294	

January 1976

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number	1	14		
Average Delay	374.0	399.6		
Std Deviation	.0	174.1		
Minimum Delay	374	177		
Maximum Delay	374	943		

February 1976

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number		11	3	
Average Delay		344.3	172.0	
Std Deviation		87.9	7.0	
Minimum Delay		153	163	
Maximum Delay		491	180	

March 1976

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number	5	12	4	1
Average Delay	849.6	432.7	381.3	160.0
Std Deviation	722.3	265.5	306.2	.0
Minimum Delay	210	238	160	160
Maximum Delay	1779	1200	909	160

April 1976

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number	4	10	1	
Average Delay	300.0	279.5	175.0	
Std Deviation	36.0	82.0	.0	
Minimum Delay	251	201	175	
Maximum Delay	347	431	175	

1051 PORTLAND OREGON POR1 C 503/224-0750

August 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number		1		
Average Delay		299.0		
Std Deviation		.0		
Minimum Delay		299		

Maximum Delay		299		

December 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number	3			3
Average Delay	666.0			229.7
Std Deviation	110.7			14.4
Minimum Delay	519			210
Maximum Delay	786			244

January 1976

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number			4	
Average Delay			458.3	
Std Deviation			154.5	
Minimum Delay			266	
Maximum Delay			614	

1054 SAN JOSE CALIFORNIA CRP2 C ** 408/446-4850

August 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number		1		
Average Delay		211.0		
Std Deviation		.0		
Minimum Delay		211		
Maximum Delay		211		

1060 MOUNTAIN VIEW CALIFORNIA AME1 E ** 415/965-8815

June 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number			3	
Average Delay			287.0	
Std Deviation			88.0	
Minimum Delay			171	
Maximum Delay			384	

July 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number		3		
Average Delay		318.0		
Std Deviation		124.7		
Minimum Delay		220		
Maximum Delay		494		

1063 PITTSBURGH PENNSYLVANIA PIT1 C 412/765-3511

June 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number		2		
Average Delay		471.5		
Std Deviation		45.5		
Minimum Delay		426		
Maximum Delay		517		

September 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number		3		
Average Delay		268.7		
Std Deviation		49.5		
Minimum Delay		200		
Maximum Delay		315		

November 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number	1			
Average Delay	283.0			
Std Deviation	.0			
Minimum Delay	283			
Maximum Delay	283			

December 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number	1			
Average Delay	267.0			
Std Deviation	.0			
Minimum Delay	267			
Maximum Delay	267			

February 1976

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number		1		
Average Delay		668.0		
Std Deviation		.0		
Minimum Delay		668		
Maximum Delay		668		

March 1976

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number	1			1
Average Delay	297.0			266.0
Std Deviation	.0			.0
Minimum Delay	297			266
Maximum Delay	297			266

1072 PALO ALTO CALIFORNIA PCOSR1 E ** 415/326-7015

August 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number		1		1
Average Delay		169.0		148.0
Std Deviation		.0		.0
Minimum Delay		169		148
Maximum Delay		169		148

1073 UNION NEW JERSEY UNISR1 E ** 201/964-3801

June 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number			2	
Average Delay			371.0	
Std Deviation			9.0	
Minimum Delay			362	
Maximum Delay			380	

August 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number	1	1		
Average Delay	484.0	692.0		
Std Deviation	.0	.0		
Minimum Delay	484	692		
Maximum Delay	484	692		

October 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number	2	1		
Average Delay	769.5	485.0		
Std Deviation	97.5	.0		
Minimum Delay	672	485		
Maximum Delay	867	485		

November 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number	7	10		
Average Delay	641.6	689.8		
Std Deviation	204.4	178.2		
Minimum Delay	419	476		
Maximum Delay	1106	1055		

January 1976

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number	1			
Average Delay	281.0			
Std Deviation	.0			
Minimum Delay	281			
Maximum Delay	281			

March 1976

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00

Number		2		
Average Delay		688.5		
Std Deviation		221.5		
Minimum Delay		467		
Maximum Delay		910		

April 1976

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number	1			
Average Delay	1125.0			
Std Deviation	.0			
Minimum Delay	1125			
Maximum Delay	1125			

1112 NEW YORK NEW YORK NYCSR2 C ** 212/750-9433
NEW YORK NEW YORK NYCSR2 C ** 212/750-9445

June 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number	4	13		
Average Delay	668.5	308.1		
Std Deviation	207.6	51.3		
Minimum Delay	458	232		
Maximum Delay	960	439		

July 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number	5	7		
Average Delay	655.2	532.9		
Std Deviation	176.9	104.2		
Minimum Delay	401	356		
Maximum Delay	891	679		

August 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number		1		
Average Delay		600.0		
Std Deviation		.0		
Minimum Delay		600		
Maximum Delay		600		

December 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number	1			
Average Delay	894.0			
Std Deviation	.0			
Minimum Delay	894			
Maximum Delay	894			

1116 CHICAGO ILLINOIS CHISR1 C ** 312/368-4607

August 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number		1		
Average Delay		166.0		
Std Deviation		.0		
Minimum Delay		166		
Maximum Delay		166		

1173 VALLEYFORGE PENNSYLVANIA VFOSR1 E 215/666-9190

December 1975

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number	1	4		
Average Delay	311.0	392.8		
Std Deviation	.0	102.4		
Minimum Delay	311	266		
Maximum Delay	311	511		

January 1976

	05:00-09:00	09:00-17:00	17:00-22:00	22:00-05:00
Number		4		
Average Delay		457.5		
Std Deviation		28.2		
Minimum Delay		421		
Maximum Delay		496		

APPENDIX E

MAINSAIL DESIGN SUMMARY

A MACHINE-INDEPENDENT PROGRAMMING SYSTEM

Clark R. Wilcox
SUMEX Computer Project, Stanford University
Stanford, California

ABSTRACT

A general-purpose programming system is being developed for the support of portable software, and as a tool for research into machine-independent code generation. The issues involved in such a design project are discussed, and an overview is given of the approach taken for MAINSAIL.

INTRODUCTION

Much effort is now expended in the development of software whose conceptual framework, at least, is already well-understood and documented. A significant amount of time spent in such development is invariably attributable to the particular environment in which the program will execute, rather than the function of the program itself. An algorithm is easily overwhelmed by implementation details, and its intention obscured by the resulting program. The source language, the operating system, the size of the machine, the file system, the debugging facilities, the time schedule, the demands of efficiency: all seem to conspire against clarity and generality. The original purpose, and the means used to obtain a running program, can become inextricably enmeshed, the result having no application beyond its limited context. The program becomes tied to the machine, the operating system, a particular version of the operating system, and the various local enhancements, and certain terminals, with given keyboards and character sets; it continually becomes obsolete, never works quite right, and dies a certain death when the author departs. And yet essentially the same program is developed for other machines, and meets the same fate. There seems to be neither the time nor the tools to do it right once, and distribute it; indeed, everyone is busy writing his own version.

If a program is to find general use beyond the confines of a particular implementation, the multitude of machine-dependent traps must be defended against at every turn. Whether this necessarily entails a loss in efficiency (program size and execution time), and the inability to use local features which might otherwise enhance performance, is becoming less clear, and certainly less important as memory and processing rates increase. The programming task is being given increased scrutiny, with an eye to the elimination of duplication, obscurity and inflexibility solely for the purpose of execution-time efficiency. Software is viewed more as a product with general applicability than as a means to an end. The tremendous effort required for a quality software product is resulting in a less tolerant attitude towards programs which must be totally rewritten if "moved" to a new machine.

If programmers had access to programming systems which aided in the creation of portable software, then perhaps we would be surprised at the tasks now considered machine-dependent which could be cast in a more general mold, passed from one machine to another, with possibly minor changes isolated and well-documented. To gain acceptance, such a system must balance several conflicting requirements without adversely affecting its ease of use.

PORTABILITY

The programming system itself must be transportable among a wide variety of machines. Its design must incorporate the means to insure compatible versions among machines, and to allow a new machine to be implemented with a minimum of effort. A language standard, presumably enforced in all implementations, is not sufficient. There is little chance that every version will be totally compatible. A standard retards the introduction of improvements and new ideas, since every implementation requires concurrent upgrading to preserve compatibility. The orchestration of such updates across a broad class of computers is prohibitive. Thus the parallel development of the programming system on many machines is not sufficient, and is an example of the very redundancy which a machine-independent programming system can alleviate. Such is the case with many languages which are now used for program portability, for example FORTRAN, COBOL, BASIC and SIMULA.

If a single version of the system could be written and distributed to all sites, then an elegant solution would be provided to the problem of maintaining compatibility, and hence portability. There would be no need for a language standard, since each site would use the same compiler. Every version would without question be compatible, since there would be only one version. Any changes to the system would be immediately transmitted to all sites by merely sending copies of the updated software. Errors found by one site result in fixes for every site.

This type of distribution can take place if the programming system is written in its own language. All software comprising the MAINSAIL programming system is itself written in the MAINSAIL language. The compiler can compile itself, and its own runtime system. It is easily bootstrapped since it is written in a subset of MAINSAIL which can be compiled by an existing compiler for the language SAIL, from which MAINSAIL is derived. Furthermore, the creation of a MAINSAIL system for a new computer is largely automated by a compiler-generator program.

The programming system itself is one example of the portability of programs written for the system. As a corollary, user programs can be written which will execute correctly on any implementation. The consequences of being able to move programs freely among several computers and operating systems are far-reaching. Programs may be shared among all sites, regardless of what computers are involved. At a single site, the same language can be used on all computers, thus promoting program interchange, and removing the problems involved with using different languages on each computer. If one computer system becomes unavailable, programs may be moved to another. The introduction of new computers may take place without fear that existing programs will become obsolete: it is only necessary that the programming system be implemented on the new system.

EFFICIENCY

In order to compete successfully with existing programming systems, a machine-independent system must offer advantages greater than the penalties derived from its lack of intimacy with the host machine. While this statement is nearly tautological, it nevertheless suggests the tradeoffs between efficiency and portability which must be dealt with in the design of such a system. Machine-independence is more a question of degree than possibility, since, in theory at least, even an extremely limited machine can be made to simulate the operations of the most powerful.

In order to obtain an acceptable level of efficiency, few assumptions concerning the target machines should be embodied in the programming system. It would be unacceptable to model all target machines as stack machines, if this model must be carried to the point of code generation. Similarly, register usage, linkage conventions, addressability, and storage allocation must not be given rigid characteristics if the system is to be truly portable. Interpreted code cannot be emitted in every case. Such considerations seem to rule out the effectiveness of a well-defined "abstract machine" for which code is generated. Instead, the code should be made to fit each target machine as well as most compilers now fit the machines for which they were designed. In many cases MAINSAIL is able to generate better code than existing compilers. For example MAINSAIL produces about 10 percent less code than the SAIL compiler, which was designed for a particular machine (PDP-10).

MACHINE-DEPENDENCIES

Somewhat paradoxically, a machine-independent programming system can benefit from features which support its use in machine-dependent applications. If the language attempts to ban any constructs which it considers machine-dependent, then programs which by their nature are heavily dependent on a particular machine configuration cannot be written. Programmers who would prefer use of the language must turn to another for such purposes; their preferences may be similarly turned.

At the very least, linkage should be allowed to external procedures written in other languages, so that a library of procedures of local interest can be constructed. If such a procedure is very short, say merely a call to the operating system, then the overhead for a procedure call may be unacceptable. In this case, the ability to insert assembly language directly into the program is most useful.

By its very design, MAINSAIL can benefit from machine-dependencies. Though most of the runtime system is written for portability in MAINSAIL, some system procedures are too machine-dependent to be written once for all computers. When writing these procedures for a particular implementation, it is desirable to use MAINSAIL if possible, because of the ease with which the machine-dependent portion can be interfaced with the machine-independent parts. Thus the entire runtime system may be written in MAINSAIL, which seems almost magical considering that everything else is also written in MAINSAIL.

There is of course a danger in explicitly allowing the introduction of machine-dependencies into the language. Programmers may begin using such constructs when not really necessary, so that the advantages of using a portable language are lost.

LANGUAGE DESIGN

In designing a general-purpose language for portability, one is immediately faced with the problem of data representation, for this is most closely dictated by the underlying machines. The selection of primitive data types must not be too narrow to prevent the full use of more powerful machines, nor too broad to require extensive simulation on smaller machines. Two basic approaches for data definition suggest themselves: offer standard definitions from which the programmer must choose; or give the programmer control over data characteristics such as range and precision. These approaches can be contrasted for the primitive data type integer.

The first would offer one or more standard ranges, for example INTEGER and LONG INTEGER, with ranges corresponding to, say, 16 bits, and greater than 16 bits (an upper bound would be of dubious value). These ranges would correspond to the minimal ranges expected for all computers to be implemented, and the programmer would understand that in a program written for portability, LONG INTEGER would preclude its use on computers with a small word size, unless this type were simulated. On larger computers, INTEGER might be represented with, say, 32 bits, and programs written specifically for such machines could make use of the full range.

The second approach would include, with each declaration, range information, for example the smallest and largest values. The compiler would use this information to allocate the integer, presumably choosing different representations for different ranges. The programmer need consider only the characteristics of his data, rather than the various machines which are to support his program. The inclusion of a range specification is also a useful form of program documentation, and aids the compiler in checking that the variable is properly used. Of course, the programmer must realize the consequences if his integer range is beyond that of a 16-bit word.

MAINSAIL presently offers the first approach with data types INTEGER, LONG (integer), REAL, and DOUBLE (real). LONG and DOUBLE are useful if the hardware provides these extended data types, or they are necessary for the intended applications, but must be supported by software. In the latter case they are expensive to use, and the single precision types should be employed where possible. In either case, machine-dependent considerations are involved in deciding to use these types, and thus they cannot appear in "portable" programs. This approach simplifies the compiler design, and perhaps results in more efficient code for smaller machines, where this is most crucial. The type BITS, for logical operations on bit vectors, is also offered, and defined as providing at least 16 bits. Thus the data types are optimized for ease of implementation, rather than optimal use of storage on machines with larger words. The compiler is never concerned with an attempt to "pack" a data type into the available words.

MAINSAIL says nothing about the bit patterns used to represent data. For example, integers can be represented as ones complement, twos complement, or even decimal. Bit operations are allowed only on the type BITS, with standard conversions among BITS and INTEGER. An INTEGER is converted to BITS by forming the binary representation of the integer (undefined if the integer is negative). Similarly, a BITS is converted to INTEGER by forming the non-negative integer whose binary representation is given by the bits. Thus it can be determined whether a positive integer is odd by converting to BITS and testing the low-order bit, no matter what representation is being used.

Another issue of data representation is the character codes. MAINSAIL offers the type STRING, which is a variable-length sequence of characters (the number of characters is automatically kept track of). There are two operations which are concerned with character codes: the first character of a string may be converted to its integer code; and an integer may be converted to a string of one character. The codes used to store characters within strings are of no consequence; there is only a need for a standard code during the two operations. MAINSAIL decrees that the ASCII codes are in effect whenever an integer is deemed to be a character code. Each implementation is responsible for any necessary conversions to and from the internal codes used in string storage.

In order to allow the runtime system to be largely written in MAINSAIL, some assumptions concerning memory and addressability are necessary. The amount of memory required by each data type is measured in "storage units." The physical interpretation of a storage unit is machine-dependent; for example, a storage unit may be a "byte" or a "word." The number of storage units required by n consecutive values of the same type, for example elements of an array, is n times the size of a single value. However, sizes of consecutive values of differing types cannot be added to obtain a total size, since machine-dependent "padding" may occur between the allocations for alignment purposes.

The type ADDRESS is introduced for manipulating memory addresses. A memory model is adopted which specifies only those addressing characteristics necessary for the simplest memory accesses. For example, an address is not used to indicate a particular character of a string, since this is not possible on some machines without additional information concerning the location of the character within a word. Associated with each STRING is a "string descriptor" which contains the current length, and the location of the first character. A string descriptor is a primitive data type, since an integer-address pair may not be sufficient.

Addressability, and the associated issue of program linkage, is an area which requires special attention. MAINSAIL allows programs to be written as separate texts, called "segments." These segments are separately compiled, and linked together to form a program in some machine-dependent manner. Inter-segment communication is provided by global data and procedures. Each segment is given a name and characteristics such as MAIN and OVERLAY. A variable or procedure is declared "external" by preceding its declaration with the name of the segment which contains its "internal" occurrence. If a procedure is internal to an OVERLAY segment, then that segment must be brought into memory before the procedure can begin execution. MAINSAIL does not provide the facilities for such overlay handling, but does include the syntax for specifying which segments are overlays.

A machine must provide for an address composed of a static or dynamic base (possibly external), with a static or dynamic offset. Static means that the value does not change during program execution, i.e. it is known at compile-time (within relocation). Thus a computer which does not provide indexing will produce inefficient code. A single level of indirect addressing can also improve the code quality. For example, if an address variable is in memory, it is useful to be able to access, say, an integer pointed to by the address, without first loading the address into an index register.

The syntax of expressions and statements is more distant from the underlying machine, so that there are few difficulties in removing machine-dependencies. Perhaps the overall result is a clear and straightforward syntax, since the prejudices and peculiarities exhibited by more machine-dependent languages are missing. There are no exotic data operations, since every machine would have to support such operations. Probably no machine will have instructions corresponding to every operation, though some come rather close. For example, BITS can be shifted left or right by any amount. Some machines have instructions which do just this; others require several instructions, or even a procedure call. STRING operations are generally too complicated to be carried out in-line, and thus there is no requirement for byte addressability or compact byte-manipulation instructions.

COMPILER DESIGN

The primary consideration in the design of a machine-independent compiler is the interface between what is known about the language and assumed about all target machines, and what is left to be supplied for each implementation. If too much is assumed, then the class of machines is unduly restricted, and clumsy devices may be necessary to resolve a distorted model to reality, resulting in needless inefficiencies. If too little, then the generation of a new system could be a major undertaking, retarding the spread of the system to new machines.

In contrast to a compiler-compiler which has no knowledge of the source language, the MAINSAIL language and compiler evolved by an iterative process. Features which were felt necessary for an efficient compiler were simply put into the language. Similarly, the language was modified in those areas requiring an inordinate amount of time or space for compilation. With regard to optimizations, this intertwining of design may result in additional statements in the compiler, yet a smaller compiler when the optimized version compiles itself.

The compiler consists of two passes in order to cleanly separate the machine-independent and dependent phases. The first pass converts the source program to an intermediate language, and the second translates this intermediate language to the target assembly language (which must be assembled by some machine-dependent assembler not provided by MAINSAIL). The intermediate language consists of operators with a variable number of operands. The operators reflect either MAINSAIL operations, such as addition; program structure, such as procedure entry; or internal information, such as the handling of temporaries. In most cases an operand is a pointer into the symbol table.

This is quite different from an attempt to generate intermediate code for an abstract machine. For example, the intermediate code for "a := a + b" might be <push a>, <add b>, <pop a> if the abstract machine were stack-oriented, whereas MAINSAIL generates <add b a>. In the former case, a register-oriented machine could certainly simulate the pushes and pops, but the generated code would be of dubious quality. A machine with a memory-to-memory add would suffer even more. MAINSAIL, however, generates intermediate code which captures only what is in the source program, with no assumptions concerning the target machine. The <add b a> can involve registers, a stack, memory-to-memory, or even a procedure call.

The second pass consists of a machine-independent part, and a machine-dependent part which is translated from a code-generation language. The machine-independent part is responsible for creating a convenient interface to the machine-dependent part, consistent with the separation between the two. It fetches the intermediate instructions, and sets up the operator and operands for easy accessibility. It supplies answers to questions concerning the operands, or the current code generation environment which it is responsible for maintaining.

MAINSAIL employs a general notion of register which is useful in a number of contexts. An operand is always associated with a memory location, and may be temporarily marked as loaded in a register. The compiler provides several services related to registers, such as: mark an operand in a register, clear a register, or find the "best" free register. It will automatically load and store registers when necessary. A register may also be marked as containing the address of an operand.

The services provided for registers are never invoked unless the code generators either directly request a service, or indicate that registers are to be used in certain situations (for example, to pass procedure parameters). Thus code can be generated for machines with no registers, for example a stack machine (actually, the top of the stack can be modeled as a register). A code-generation environment is created and maintained which is flexible enough to be of use for a wide variety of computer architectures. Many checks insure the internal consistency of the environment, for example a register cannot be marked with two operands at the same time. By knowing the rules of this environment, code generators can be written for a new computer with minimal effort.

The code-generation language provides a powerful and convenient setting in which to specify code sequences. Declarations give semantic information concerning register usage, storage units, additional symbol table entries, and various parameters used within the compiler and runtime system. A code generator must be written for each intermediate instruction. A generator has available to it services such as those discussed above, and the operands of the intermediate instruction. In general a code-generator looks like the assembly language which it is to produce, except it contains keywords which are replaced during code generation with operand names, registers, or constants. The code-generation language is translated to MAINSAIL, and hence the full power of MAINSAIL is available. In practice, the constructs provided are sufficient for almost all situations which arise during code generation. A code generator usually takes the form of a series of conditions, each followed by pseudo assembly language which is to be processed if the condition is satisfied. The complexity of the conditions is determined by the degree to which the target machine conforms to the general framework provided for code generation, and the amount of optimization desired. Procedures can be used for commonly occurring code sequences.

Since code generators are associated with intermediate instructions, they provide only for local optimization. Because of the extreme ease with which the code generators can be altered, a compiler can be created from the current generators, and its output examined for errors and inefficiencies. Based on this, the generators can be altered, a new compiler created, and so forth. This process continues until the code appears correct, and is sufficiently efficient. Construction of a new compiler from a few changes in the generators can be done in a matter of minutes. Thus a single session spent tuning the generators can produce significant results.

The formal separation of target-machine semantics from the more general aspects of code generation has an exciting potential for research into the design of instruction sets. Since a wide variety of computers can be described with the code generators, experiments can be conducted to test features such as the number of registers, the utility of indirection, or various procedure linkages. Existing machines can be compared to determine which is best suited for a high-level language implementation. For example, an instruction set which allows complete addresses can be compared with one which offers a base with small displacement, to determine which requires the fewest memory accesses. A micro-coded instruction set based on the MAINSAIL intermediate instructions would produce optimized code sequences.

The facility with which code generators can be written makes MAINSAIL accessible to one-of-a-kind machines. For example, there is now under construction a three-address parallel processor with no registers which will use MAINSAIL as its high-level language. Programs can be written, and the code examined, before the machine is complete (even the assembler for the new machine can be written in MAINSAIL!). Providing such a machine with a high-level language would be a major undertaking if the compiler, runtime system and assembler had to be written in assembly language.

RUNTIME DESIGN

The runtime system provides support during program execution: program initialization, file manipulation, i/o, conversions among string and numeric-bits, string handling, mathematical routines, string and record collection, and dynamic memory allocation. If MAINSAIL is to be used as an implementation language, then it may be desired to limit the size of the runtime package. Since the system procedures are used only in response to implicit or explicit requests, programs may be written which require little, if any, support. For example, programs which involve only arithmetic, logical and address operations, with no i/o, string handling or dynamic storage allocation, may be compiled into assembly language programs which call only the system initialization procedure. By removing this call, a self-sufficient program is obtained which can be combined with hand-coded assembly-language modules. In this sense, MAINSAIL can be regarded as a convenient means of generating assembly language programs.

Mathematical routines for trigonometric functions, exponentiation, logarithm, square root, and random numbers have been written in MAINSAIL, accurate to at least 17 decimal digits in most cases. Since they are written in MAINSAIL, there are of course no assumptions regarding word size or representation. The obscurity of their assembly language counterparts is in stark contrast to the clarity with which the algorithms are expressed in a high-level language, and has probably contributed to the astounding number of times they have been written, over and over again, for different machines. The same can be said of the MAINSAIL routines for conversion between string and floating point numbers.

MAINSAIL has a well-developed i/o capability, including any number of sequential and random files, and terminal interaction. File names are represented as strings, and the format of these strings is transparent to MAINSAIL, since they are handled only by machine-dependent routines. There are two types of sequential files: text and data. Text files are meant for legible text, for example a program or document. Whenever numeric or bits data is written to a text file, an automatic conversion is made to a string representation; similarly, such reads from a text file automatically scan for the proper string representation.

A data file contains machine-readable data in some machine-dependent format. Any mixture of numeric and bits can reside on a data file, presumably stored in a compact form identical to the internal representation within the computer. Since no conversion is necessary, input and output is efficient.

A random file is composed of fixed-length blocks of data, called file-blocks. Reads and writes supply a file-block number, and the entire file-block is involved in the transfer. A file-block is read into, or written from, a memory area whose address is supplied to the read or write routine.

Files can be opened, closed, and deleted. Additional file-manipulation routines can be added for each site. Much of the i/o activity is handled in a machine-independent manner, so that only a few well-defined elementary procedures need be written for each machine.

CURRENT STATUS

MAINSAIL now runs on a PDP-10 with TENEX, and a PDP-11 with RT11. Development is under way for a PDP-10 with TOPS10, a PDP-11 with UNIX, and the IBM-370. Code has also been generated for an INTERDATA 7/16, VARIAN and NOVA. Many more machines were examined while developing MAINSAIL, and will be considered for implementation as sufficient resources are made available.

A number of projects across the country are interested in using MAINSAIL for the development of portable software. Among these are a robotics project, a mass spectrometry system, a program for chemical structure elucidation (now written in LISP), a computer-aided-instruction system for the teaching of logic, an automated cell classification laboratory, a machine-independent version of INTERLISP, and a display-oriented text editor.

APPENDIX F

SUBSYSTEMS AND DOCUMENTATION DIRECTORIES

Nancy Smith
December 1974
(updated April 1975)
(updated Sept. 1975)
(updated Oct. 1975)

The sources of available documentation for these programs will be abbreviated as follows:

TUG	Tenex User's Guide (1975 edition)
DUH	DEC Users Handbook
DAL	DEC Assembly Language Handbook
DML	DEC Mathematical Languages Handbook
HC	a hard-copy manual for the language
OL	on-line documentation which can be found by @DIR <DOC>programname.* . The following extensions are used on the <DOC> directory:

.MANUAL	complete usually fairly long manual
.HELP or .HLP	shorter summary, list of commands, etc.
.SUPPLEMENT	on-line supplement to hard-copy doc
.UPDATE	list of updates by date
.SAMPLE	sample program or output

See <DOC>A-LIST-OF-ALL-AVAILABLE-DOCUMENTS.INFO for complete details on these documents including where and how to order them.

Many of the major programs also have a <BULLETINS>programname.BBD file where messages about new developments, bugs, hints for using the program etc. are sent. These <BULLETINS> files can be read by any of the mail reading programs (READMAIL, RD, MSG, or BANANARD).

New programs or new versions of old programs will be put on <NEWSYS> for a trial period. The file <NEWSYS>NEW-SYSTEMS.INFO which is a message file will have a message about each program available. These new programs will not be included in the list of programs given here.

The HELP program obtained by typing @HELP gives assistance in finding the appropriate on-line documents for the various programs.

SUBSYS	DESCRIPTION	DOC
2SIDES	makes files for multi-columns and/or 2-sided listing	OL
ACCESS	gives a list of subsys's currently available to GUESTs	
ADDMSG	appends a msg to a specified file	
AID	algebraic interpretive dialog conversational lang.	HC
AIFAIL	assembly lang. - early version of FAIL from SU-AI	OL,HC
ALIAS	allows a dummy name to be set up for a program	
BAIL	SAIL debugger (on <SAIL>)	OL
BACKUP	short term file loss protection	OL
BANANARD	msg reading program (many extra features)	OL
BASIC	conversational programming lang. (DEC version)	OL,DML,TUG
BCPL	compiler writing and systems programming lang.	HC
BINCOM	binary comparison of files (now replaced by FILCOM)	DAL
BLIS10	compiler for system implementation (DEC version)	OL,HC,TUG
BLIS11	BLISS for the PDP11	
BLISS	compiler for system implementation (TENEXized)	OL,HC(DEC)
BOOTGT	loader for the PDP11 (GT40)	
BUDGET	budget management program (especially proposals)	OL
BYE	@BYE same as @BREAK (LINKS)	
CALENDAR	calendar management and reminder system	OL,TUG
CAM	the compare and merge program of SOUP see <DOC>SOUP.MANUAL	
CCL	concise command language	OL,DUH
CLEAN	a file by file directory clean-up program	OL
COPYM	reading/writing DECtapes	OL,TUG
CREF	cross-reference assembly listing	OL,DAL
CRSREF	TENEX cross-referencing program (outfile_infile(s))	
CRYPT5	En/Decrypts textfiles to provide security	OL
DCHANGE	character set conversion for "foreign" tapes	OL
	see <DOC>DCHANGE.MANUAL and <DOC>DCHANG.HLP	
DCHECK	reads blocks of file into core & calls DDT to examine	OL
DDT	debugger (single-stepping added at IMSSS)	OL,TUG,DAL
DED	text-editor (designed for TENEX)	OL
DELOLD	deletes files by cutoff date of last access	OL
DELVER	deletes excess versions of files	TUG
DFTP	file transfers to and from the Datacomputer	OL
	(for certain special file storage needs)	
DIABLO	prints final copy of PUB-produced documents on DIABLO	OL
DIREXT	prints directory information for files sorted by	OL
	file extension rather than file name	
DO	creates or appends a line to a reminder file	OL
DOM	effects the assembly and loading of a single	
	MACRO program	OL
DONE	deletes a line from a reminder file	OL
DROP	similar to DELVER, deletes oldest and 2nd newest on *.*	
DSKACC	gives dsk allocation for all members of accounting groups	
DTACOP	DECtape to DECtape copy	
DUMPER	reads/writes magnetic tapes	
EOFIX	deletes any pages past end of file mark	OL
EXTR	"EXTRactor" processes MACRO/FAIL source files to	
	produce .FAI listing of labels defined	
F40	FORTRAN IV (see also <DOC>FORTRAN.HELP and	OL,TUG,DML
	<DOC>LISP-FORTRAN-INTERFACE.HELP)	
FAIL	assembly language (BBN version of FAIL)	OL,HC
	(see also JSYS manual & <DOC>SUMEX-JSYS'S.INFO)	

FED	the final edit program of SOUP see <DOC>SOUP.MANUAL	
FILCHK	checks SAIL programs for loader incompatibilities	OL
FILCOM	complete file comparison package	OL,DAL,TUG
FILDMP	dumps files in variety of formats	OL
FILES	multiple to multiple copies, renames, protections	
FILEX	for file transfers converts between DEC machine formats for dsk and DEC-tape.	OL,DUH
FORMAT	makes table of contents & index for SAIL sourcefiles	OL
FORTRA	FORTRAN10(version 4) (see also <DOC>FORTRAN.HELP)	OL,HC
FREQ	ranks words in text file according to frequency	
FRKCOM	compares an address space with address space of file	TUG
FTP	ARPANET file transfers	TUG
FUDGE2	updates/manipulates files containing rel programs	DAL,TUG
GETDMP	loads into core .dmp file from SU-AI (SAV only to 677777) type filename to * prompt	
GRIPE	sends comments or complaints about system to staff	TUG
HELP	helps locate on-line documentation	
HOSTAT	prints network site status information	TUG
IDDT	DDT for inferior forks	TUG,OL
IFAIL	assembly language (IMSSS version of FAIL)	OL,HC
ILISP	UC Irvine LISP (extension of LISP 1.6)	OL
IMSSS	direct link to IMSSS	
INSPEX	checks files for wasted space and pages past eof	OL
KILL	closes all jfns--useful when RESET can't get a file closed	
LAST	Gives date, time of last full dump, archive or daily dump	
LD	prints SYSTAT-like info	
LINK10	DEC loader	OL,DAL,TUG
LINK11	linker for PDP11 DOS operating system	
LINKSTAT	prints status of IMSSS link	
LISP	INTERLISP-see also <DOC>LISP-FORTRAN-INTERFACE.INFO	OL,HC
LOADER	(from IMSSS)-see <DOC>LINK10-LOADER-DIFFERENCES.HELP	TUG
LOADGT	GT40 standard format loader	
LOADVT	loader for PDP11 (GT40)	
LOWCASE	converts a text file to lowercase	
LPTSTS	gives the files on the lineprinter queue & their size	OL
MAC11	MACRO cross-compiler for the PDP11	
MACRO	assembly lang-JSYS manual & <DOC>SUMEX-JSYS'S.INFO	TUG,DAL
MAILBOX	to reroute mail (not fully implemented yet)	OL
MAILSTAT	info on queued mail	TUG
MANTIS	Fortran debugger	
MATHLAB	interactive symbolic algebraic system	OL
MLAB	mathematical modeling and graphics package	OL
MSGFIX	TECO routine to help fix the format of messages	
MTACPY	magtape program	TUG
MTCOPY	DEC magtape program	OL
MULTI	multiple-fork supervisor--switches between forks	
MY-ACCOUNTS	prints user's valid accountnames	
NDIR	gives compact list of files on connected directory	
NETSTAT	prints info on ARPANET status	TUG
NEWFILES	directory information for files written in last 24 hrs	OL
NEWINFO	gives all new files on public directories or for any file group (includes number of reads for each file)	OL
NODE	gives the geographical location of a TYMNET node	
NON	zero-compresses file, options to remove linenumbers, pagemarks, convert eol's, etc.	

PCSAMP	measures the operation of other user programs	TUG
PDP6DT	DEC-tape program	
PIP	DEC utilities program	OL,DUH
PIP11	transfers PDP11 DOS DECtapes to/from TENEX files	OL
PNTMAK	converts underlines to suitable format for LPT:	OL
POET	text editor designed for TENEX use	OL
PPL	an interactive extensible programming lang.	TUG
PROFIL	gives freq of execution of SAIL statements	OL,HC
PUB	document preparation lang.	OL
PUB2	2nd pass of PUB -- used separately to change underlines	
RD	mail reading program (MSG is better)	TUG
READMAIL	mail reading program (MSG is better)	TUG
RECOG	when ordinary recognition is ambiguous RECOG gives the possible filename matches	OL
RECORD	for pseudo-ttys, typescript of job, detaching from running job	OL
REDUCE	symbolic algebraic language	OL
RPURGE	requires confirmation before purging (delete & expunge) & puts info on purged files in a file by date	OL
RSEXEC	restricted access only	TUG
RTTY	types out a file starting at the end (reverse)	OL
RUNFIL	uses file instead of tty for input commands	TUG
RUNOFF	document-preparation language (DEC not BBN version)	OL
SAIL	ALGOL-like lang.-see also <DOC>LEAP.MANUAL	OL,HC
SCAN	scans multi-directories for a variety of file info	OL
SEARCH	searches multi-text files for English words or SAIL identifiers, can be used with TV editor	OL
SEARCHDIR	substring search of directory information on files	OL
SEARCHP	substring search also allows random reading of file	OL
SEGSAV	reads .shr & .low files to produce TENEX .sav	OL
SITBOL	compiler version of SNOBOL	OL
SNDMSG	message sender	OL,TUG
SNOBOL	string-processing programming lang.	OL,HC
SORT	stand alone COBOL column-oriented text file sorter	OL,TUG
SOS	text editor	OL
SPELL	spelling checker/corrector for text files (not TENEX)	OL
SPSS	Statistical Package for the Social Sciences	OL
SRCCOM	compares text files	TUG
STP	Western Michigan University StaTistical Package	OL
SUBMIT	submission to batch (see <DOC>BATCH.HELP)	
SWITCH	switches the format of a reminder file	OL
SYSDPY	gives SYSTAT-like info constantly updated on display (CRT) terminal	OL
SYSIN	executes LISP SYSOUT's	OL
TABLE	creates conversion tables for DCHANGE	
TALK	used with LINK command to eliminate need for ;'s	
TAPCNV	reads card image file processed by MTACPY	TUG
TBASIC	TENEXized version of DARTMOUTH BASIC	OL
TCTALK	teleconferencing over ARPANET	OL
TECO	text editor (see TENEX TECO manual)	OL,TUG
TELNET	restricted access only	TUG
TIPCOPY	sends text files to a TIP port	TUG,OL
TMERGE	merges specified text pages from files into new file	OL
TODAY	lists the contents of today's reminder file	OL
TRITAP	processes magtapes from XEROX, IMSSS, BBN	OL

TTYTRB	used to report terminal line problems	TUG
TTYTST	prints test patterns for diagnosing terminal	TUG
TV	text editor for TEC and DATAMEDIA displays	OL
TVFIX	restores bad TV files (see <DOC>TV.MANUAL)	
TYMSTAT	(for TYMNET lines only) gives measure of current efficiency of TYMNET transmission	
TYPBIN	does an octal dump of a packed file	TUG
TYPEIN	appends type-in to file with some editing allowed	OL
TYPREL	analyzes contents of .REL files	TUG
UPCASE	converts an entire file to uppercase	
WATCH	continuous on-line monitoring of system activity	TUG
WATCH.IMS	IMSSS version of WATCH	
WHAT	lists the contents of a reminder file	OL
WHO	prints SYSTAT-like information	
WHOIS	looks up username & prints name/address info on user	OL
VIEW	examines a file word by word, several typeout modes	OL
XED	text-editor (used with BANANARD)	OL
XT	reformats and prints text file	OL
Z	logs jobs off including from inferior (lower) forks & prints a witty saying	

<DOC> DIRECTORY LISTING

The following is a listing of the <DOC> directory which contains most of the on-line formal documentation about the system and subsystems.

<DOC> 13-MAY-76 08:19:25

FILE NAME	SIZE (COMPUTER PGS)
.HELP;2	1
2SIDES.HELP;3	3
A-GENERAL.HELP;12	2
A-GUIDE-TO-TENEX-USER´S-GUIDE.INFO;2	5
A-LIST-OF-AVAILABLE-DOCUMENTATION.INFO;8	14
A-SURVEY-OF-THE-DEC-HANDBOOKS.INFO;10	5
ACCOUNT-NAME-USAGE.INFO;2	3
AID.HELP;4	2
.INFO;3	1
ALL-SUBSYS´S-AVAILABLE-AT-SUMEX.INFO;8	7
BACKUP.HELP;2	2
BAIL.HELP;5	1
.MANUAL;3	17
.UPDATE;1	3
BANANARD.HELP;1	1
BANK.MANUAL;2	46
BASIC.HLP;2	2
.UPDATE;2	12
BATCH.HELP;3	3
.UPDATE;2	4
BLIS10.HLP;4	2
.UPDATE;2	10
BLISS.HELP;2	2
BSYS.MANUAL;3	25
BUDGET.MANUAL;7	9
.UPDATE;2	1
.SMP;2	1
CALENDAR.MANUAL;2	6
CCL.HELP;2	2
CHECKDSK.HELP;3	4
CHESS.HELP;1	3
CLEAN.HELP;1	2
COPYM.HELP;2	5
CREF.HLP;1	1
.UPDATE;2	2
CRYPT5.HELP;1	2
DCHANG.HLP;2	2
DCHANGE.MANUAL;1	12
DCHECK.HELP;1	1
DDT.SUPPLEMENT;1	2
.HELP;1	1
.BRIEF;2	4
.SUMMARY;1	9

DEC-HANDBOOK-GLOSSARY-UPDATE.INFO;1	3
DEC/TENEX-COMMAND-EQUIVALENTS.INFO;4	11
DED.MANUAL;1	15
DELOLD.HELP;1	1
DESCRIPTION-OF-SUMEX-AIM-PROJECTS.INFO;3	4
DFTP.HELP;3	5
DIABLO.HELP;9	7
DIREXT.HELP;1	2
DOM.HELP;1	1
DUMP.INFO;1	1
EDIR.MANUAL;2	9
.HELP;1	1
.UPDATE;1	1
EDIT.INFO;1	2
EDITOR-PROGRAM-INTERFACE.INFO;2	2
EOFIX.HLP;2	1
FAIL.MANUAL;3	70
.HELP;5	3
FILCHK.HELP;1	1
FILCOM.HLP;4	1
FILDMP.HELP;2	2
FILEX.HLP;1	2
.UPDATE;1	4
FLECS.HLP;1	2
FORDDT.HLP;1	1
.UPDATE;1	2
FORMAT.HELP;1	3
FORTRA.HLP;1	1
FORTRAN.HELP;2	11
FTP.UPDATE;1	3
.ANONYMOUS-ACCESS;1	3
GLOB.HLP;1	1
.UPDATE;1	2
GRUMP.HELP;1	1
GT40-LIGHTPEN.HELP;1	3
GT40-LIGHTPEN-IMPL.DOC;1	8
GT40-OMNI-MONITOR-DIRECTIONS.HELP;1	1
GT40-OMNIGRAPH.INFO;1	2
GT40/OMNI-MONITOR.DOC;2	2
GUEST-ACCESS-SUMEX.INFO;1	1
GUEST-LOGIN.HELP;1	1
HOW-TO-UPDATE-DOC.INFO;3	3
IDDT.HELP;1	8
ILISP.MANUAL;1	116
.TENEX-MANUAL;1	49
.HELP;2	2
INSPEX.HLP;1	1
INTERROGATE.HELP;4	2
INTRO-TO-SUMEX-AIM-TENEX.INFO;5	6
ISAIL.HELP;1	1
JSYS-INDEX.INFO;1	5
LEAP.MANUAL;3	15
LINK10.HLP;1	2
.UPDATE;3	8
LINK10-LOADER-DIFFERENCES.HELP;1	6

LISP.HELP;3	2	
.UPDATE;5	4	
LISP-FORTRAN-INTERFACE.HELP;2		2
LIST.HELP;3	6	
LOADER.UPDATE;1	2	
LPTSTS.HELP;1	2	
MACRO.HLP;1	1	
.UPDATE;3	11	
MAILBOX.HELP;1	2	
MAKLIB.HLP;1	1	
MARK-MSGS.HELP;1	2	
MATHLAB.HELP;3	4	
MLAB.HELP;1	14	
MSG.MANUAL;4	17	
.UPDATE;3	5	
MTCOPY.HLP;2	1	
MULTI.HELP;1	1	
NEW-SOS-TO-SUMEX-SOS-COMPARISON.HELP;3		2
NEW-VERSION-SOS.INTRO;143		4
.MANUAL;1	31	
.SUPPLEMENT;144	20	
NEWFILES.HELP;2	2	
NEWINFO.HELP;1	2	
NOTE.HELP;1	2	
OLDFILES.HELP;1	1	
OMNIGRAPH-USER´S-GUIDE.INFO;1		94
OVERVIEW-OF-COMPUTER-SYSTEM.INFO;1		2
PAGESCAN.HELP;1	1	
PCAL.HELP;3	2	
PIP.HLP;3	1	
.UPDATE;2	10	
PIP11.HELP;1	2	
PLOTTER.INFO;1	2	
PNTMAK.HELP;1	1	
POET.HELP;1	3	
.MANUAL;1	13	
PROFIL.UPDATE;2	2	
PROJECTS-AND-ASSOCIATED-USERS.INFO;69		7
PSEARCH.HELP;1	2	
PUB.MANUAL;3	62	
.HELP;5	47	
.UPDATE;10	8	
RADIX.HELP;1	1	
RECOG.HELP;1	2	
RECORD.MANUAL;3	19	
REDUCE.MANUAL;1	44	
RPURGE.HELP;1	2	
RTTY.HELP;1	1	
RUNOFF.HLP;1	2	
.UPDATE;1	24	
.COMMANDS;1	3	
.HELP;1	1	
SAIL.HELP;2	1	
.SUPPLEMENT;4	34	
.TENEX-SUPPLEMENT;2	7	