



SYMBOL—A major departure from classic software dominated von Neumann computing systems

by REX RICE and WILLIAM R. SMITH

Fairchild Camera and Instrument Corporation
Palo Alto, California

INTRODUCTION

The prime goal of the SYMBOL research project was to demonstrate, with a full-scale working system, that a very high-level, general-purpose, procedural, "state of the art" language and a large portion of a time-sharing operating system could be implemented directly in hardware and achieve a significant increase in overall computational rates. A further objective was to create hardware design and construction techniques which could be easily applied by a small number of people to implement such a system in a reasonable time and at a relatively low cost. Although this was a research project, there was a high dedication to developing automation, hardware, test equipment and documentation to support the project. The name SYMBOL was chosen to signify direct hardware symbolic addressing.

Another goal of the project was to develop hardware/software algorithms which directly aid a casual as well as a professional user working with non-numeric data. Particular attention was given to the manipulation of data structures for file maintenance coupled with powerful field and character manipulating instructions.

The general-purpose SYMBOL language¹ was developed after studying a large number of modern languages including ALGOL, FORTRAN, PL/1 LISP and EULER. It was decided early that the new language had to perform all useful operations on application problems without being cluttered with machine-dependent operators; also, since it was a research project there was no strong necessity to be compatible with available programs written in other languages. On the hardware side it was decided that no appreciable restriction to the language would be permitted and that hardware would have to be invented to match the language. As the project progressed it became desirable to include conversational-mode multiproces-

sing and multiprogramming as well as source language text editing.

The architectural philosophy of the SYMBOL system led to hardware implementation of a variety of features which have been and are software functions of current systems. Some of the interesting features directly implemented in hardware in SYMBOL are:

- Dynamic Memory Allocation
- Dynamic Memory Reclamation
- Dynamically Variable Field Lengths
- Dynamically Variable Structures
- Automatic Virtual Memory Management
- Automatic Data Type Conversion
- Automatic Time-Sharing Supervision
- Direct Symbolic Addressing
- Precision-Controlled Arithmetic Processing
- Direct Hardware Compilation
- Alphanumeric Field Manipulation
- Direct Text Editing

PROJECT HISTORY

From the beginning, this research project was committed to producing a real and functioning system rather than a simple simulation. A brief outline of the project history appears below. It should be noted that considerable emphasis was placed on hardware design and implementation techniques. In fact, the complete project was treated as a "closed system" where no item (such as a user with his application) was considered separately from the language, operating system, or hardware.

1963-64: Hardware Technology Development
Fairchild CT μ L Family for Circuits
Power Distribution Techniques

Large Two-Layer Printed Circuit Boards
Dual In-Line Package Invented

1964-65: Data Flow Model² Process Unit Construction
Two-Layer Printed Circuit Board System Interconnections
Cam-Operated, Zero Insertion Force Contacts
High-to-Low Order Variable Length Arithmetic Processing
Use of CT μ L

1964-67: Language Development for SYMBOL
Dynamically Variable Field Length Data
Structured Data Literally Represented
Complete Syntax and Semantics
Source Text and Program Editing
Definition of Operating System Hardware/
Software

1965-69: Computer-Aided Engineering Design Package Development
Equation Expansion and Checking
Timing, Loading and Offset Checking
System Logic Function Factoring
System Interconnections
Placement
Wire Routing

1964-70: Hardware Development for SYMBOL
December, 1967—Logic Flow Charts Complete
December, 1968—Partial Construction and Testing Started
December, 1969—Fabrication Complete and Debugging Started
June, 1970—SYMBOL Hardware Operational
January, 1971—SYMBOL Delivered to Iowa State University

SYMBOL FROM A USER'S VIEWPOINT

Much of the written or printed communication in our society is conducted on what may be described as "Typewriter English." We communicate with upper and lower case alphabet, decimal numbers and a group of commonly accepted special symbols. These communications are generated in all sorts of forms and may vary from handwritten, to typewriter produced or to automatic computation output, etc. Several features become evident on examination of the preferred communication forms.

Variable field length

People use a free flowing written communication style which requires a complete variability in the num-

ber of characters in a word (English that is). The information contained in groupings of characters or words (i.e., fields) is also of variable length and can change as manipulation upon that field occurs. Although past computing practice has developed abbreviated mnemonics and codes to fit fixed word machines and to reduce writers cramp, it may be noted that deciphering one's program or data base a year later is often laborious and is especially difficult for a third party. To ease this problem in SYMBOL, techniques were developed which allow complete and dynamic variability in the length of a string of characters used as a name, a word or a field. Each user is under no software/hardware constraint and may use any word or field size he wishes. He never needs to predetermine field size by declarations. In both the source program and in the data base this variability is provided.

A new character called a Field Mark was introduced to define the start and end of a variable length field in the data base and for non-numeric literals in the program. The Field Mark is a long vertical line and was chosen so as to be easy to see and not to conflict with commonly used characters. The field mark is entered from keys, or automatically generated by the system, as the data base is developed.

Examples:

```
| 123 |
| This is a field |
| A field may be as long as desired |
| A field may be short |
| A field may be short and then expanded |
| or contracted |
```

Decimal arithmetic

People have been raised with the decimal system. Even though they adapt to computers and the use of the hex/binary system, it is unnatural. The SYMBOL system accepts either fixed or floating point decimal numbers with positive or negative mantissas varying from one to ninety nine digits with or without a decimal point. Exponents, if any, may contain a plus or minus sign and may have up to two decimal digits. The operands for arithmetic may be both fixed, both floating or mixed. The system hardware automatically converts them to an internal floating point form and computes a left justified floating point decimal result. Since arithmetic computations on variable field length numbers can produce even longer results a LIMIT register is provided to truncate the resulting computations at a desired number of significant digits. The LIMIT value may be dynamically changed by the user's program as

String Fields:

```
| Joe Doaks |
| Flight No. 306, SFO to JFK |
| Cape Code House, 2 Bedroom, 1 Bath, Living Room, Dining Room, Fireplace, Basement, 2 Car Attached
| Garage |
```

Numeric Fields:

```
| 374.1279536844879310-72 |
| -1234.5 |
| $*****1,576,265.46 |
```

Dynamically Varying Structures (Time Sequence)

```
<123 | 456 | 789>          ..INITIAL Structure
<123 <456 | ABC>789>       ..Modified Once
<123 <456 | ABC <DEF | GHI | JKL>>789> ..Modified Twice
```

The last sturcture above can be visualized as:

```
<123                      789>
  <456 | ABC              >
    <DEF | GHI | JKL>
```

Few limits are placed on data structures. Fields may grow to the size of main memory. No restriction is placed on the depth of nesting in a structure.

Operating system complexity

SYMBOL directly implements almost all of a time-sharing and multiprogramming system directly in hardware. Further, the internal machine language is the source language. This direct implementation of source language significantly reduces the layers of software normally found between source and object codes. This in turn reduces the "hidden rules" or system dependent constraints which plague the casual or professional user of the system. These features combine to make the system exceptionally easy to use for problems where data base manipulation on alphanumeric data is of prime consideration. Since the system provides powerful arithmetic operations on variable field decimal data it is also excellent for most engineering and scientific uses.

A valid point can be made that if all language is "hard wired" then error correction, extensions, new language elements, etc., are difficult to achieve. This

need was recognized in SYMBOL and features are provided in the hardware/software interface to allow expansion or extension. First, interrupts are provided for traps at hardware compile and/or execution time. These interrupts allow a break-out from the high-level language and may call a "system program" to perform some desired task. Second, privileged memory operations are provided which allow a privileged program to initiate directly any memory operation available to the hardware. A combination of regular and privilege operations may be used to create new language elements and/or new macro instructions.

Facilities for "file management," for example, are supported with software that uses a combination of ordinary language and privileged instructions. These algorithms could have been directly implemented using SYMBOL techniques, but they were not sufficiently clear to be stabilized and did not need the higher performance of direct implementation. Further research evaluating this type of tradeoff would prove most interesting.

The objective in SYMBOL was to support the high duty cycle and basic features of the operating system in hardware. Many of the algorithms are hardware executed with software established parameters so as to

obtain higher performance without losing the needed flexibility. This support provides a significant simplification in the overall operating system.

SYMBOL FROM THE ARCHITECTURAL VIEWPOINT

Studies of modern large computer systems have shown that a large portion of the logic in the main frame hardware is idle most of the time. Some of the largest systems have achieved more parallelism using a main CPU and several auxiliary smaller computers to handle input and output tasks, etc. (i.e., CDC 6600 and 7600). The GAMMA 60 developed earlier by the Bull Company in France departed from tradition by exploring the running of several problem segments in small units each containing sufficient registers and logic to operate autonomously for short computation sequences. The matrix-type systems such as the ILLIAC IV allow many identical, or nearly identical, programs to operate simultaneously on the same type of large problem.

The SYMBOL system architecture shares some similarities to and has some differences from these systems. SYMBOL achieves parallelism and execution efficiency by using time-sharing with multiprogramming and multiprocessing done through seven simultaneously operating autonomous processing units sharing a common virtual memory. The hardware contains a large main memory, used as a virtual memory buffer, and the "Autonomous Functional Units" (AFU). In the tradition of GAMMA 60 the autonomous units have sufficient logic registers, control, etc., to perform sequences of operations without being under control of a conventional CPU. SYMBOL departs from previous systems at this point. Each AFU is dedicated to a portion of the computing process and its logic (i.e., instruction sequences) is hard wired so that source language is essentially machine language.^{1,4} The gross block diagram in Figure 1 shows the common communication bus structure of the system. Each AFU is a special purpose processor.

The architecture is designed so that the time-sharing supervision is managed by the System Supervisor (SS). Provided the job load permits, each AFU can be performing its tasks on a different users job while simultaneously sharing virtual memory with other AFUs. The SS maintains queues of jobs ready for each AFU and schedules the system tasks. Communication is conducted along the main bus and by several local buses.

A unique feature of the system is automatic and dynamic hardware memory management. The assign-

ment, and access of memory is done by the Memory Controller (MC). With memory control as a service function the logic for generating and manipulating data is distributed to the various other AFUs.

Figure 2 presents a summary of the functions hard wired into each of the AFUs. The Interface Processor (IP) provides source text editing without use of the Central Processor (CP). The Translator (TR) converts the source language into a reverse Polish string form ready for processing by the CP. Below each box a number gives the count of large printed circuit cards, each containing 160 to 200 integrated circuits, used for the AFU. This gives the reader a rough feel for the relative sizes of the units.

SYMBOL FROM THE PHYSICAL VIEWPOINT

At the start of the SYMBOL project it was decided to use a functionally-factored, bus-oriented system. Preliminary studies showed that large printed circuit boards (i.e., 12"×17") with about 200 integrated circuit packages (providing the equivalent of about 800 two-input gates) would be sufficient to minimize system interconnections (Figure 3).⁵

It was also obvious that two-layer printed circuit boards were much less expensive than multi-layer boards. A substantial and successful effort was mounted to develop placement and wire routing algorithms and to obtain a computer-aided engineering package which enabled us to effectively use two-layer boards.

The choice of a circuit family drastically affected the type and number of interconnections required. The Complementary Transistor Micro Logic (CT μ L) family was chosen because of its useful "wired OR" capability which has proven to reduce interconnections between 20 and 40 percent compared to other circuit families.⁶

The system bus implementation was also given much consideration. After preliminary studies it was decided to use a set of 200 Interconnections running as parallel lines for the length of the main frame. Figure 4 is a view of one of the system bases which is a simple two-sided printed circuit board. One hundred and eleven lines run the full length of the system and are used as the main bus. The main base is partitioned as follows:

- 64 bidirectional data lines
- 24 bidirectional address lines
- 10 bidirectional priority lines
- 6 operation code lines
- 5 terminal number lines
- 1 system clock
- 1 system clear

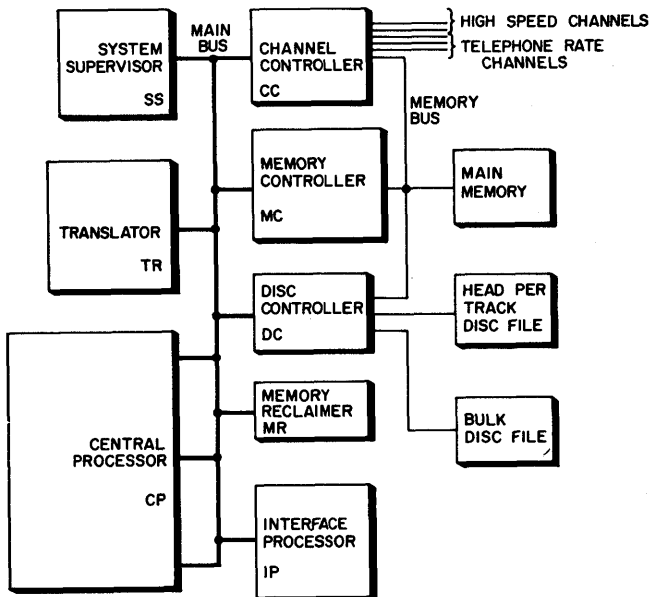


Figure 1—Gross block diagram of the SYMBOL system

The balance of the 200 lines are used for local interconnections within AFUs of two to seventeen boards in size. As the design progressed it was necessary to add an additional 200 "bypass" lines on the bases. The

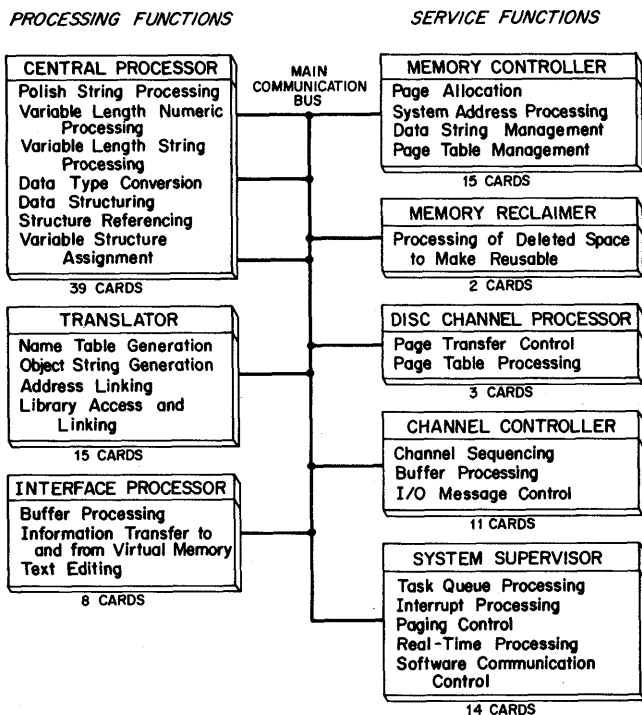


Figure 2—Functions performed in the SYMBOL main frame



Figure 3—Basic 12"×17" two sided printed circuit board with up to 220 dual in-line components

final design allows each large board to contact up to 200 bus lines and have 200 lines bypass it. Each board contact can be connected to the same contact number on the next card or alternatively can be connected via a bypass line to a board several slots distant. Using these techniques the whole system was implemented with a maximum of 600 parallel lines, with cuts, on two-layer printed circuit bases.

There have been many interesting debates within the project on the size of boards chosen and on the number of interconnecting lines needed on the bases. It is not clear that our choices are optimum; however, it is now clear that they were sufficient. The completed system used about 102 large boards inserted between the two system base structures. It is illustrated in Figure 5. It was interesting to find that all the main

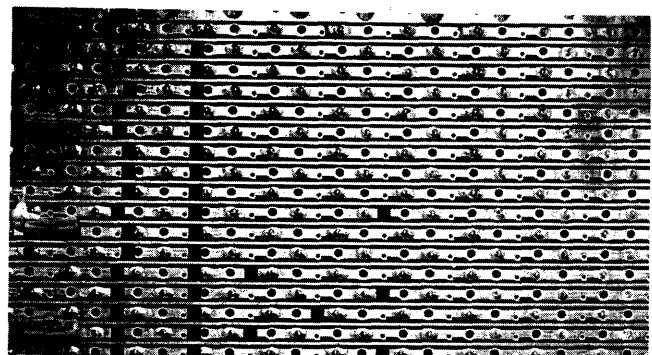


Figure 4—SYMBOL interconnection base

frame including I/O, memory, disk and channel interface logic took less than 20,000 CT μ L packages.

SYMBOL FROM A PERFORMANCE VIEWPOINT

The evaluation phase of SYMBOL IIR is just beginning with the hardware near completion. In order to obtain a preview of the performance a set of measurements has been made on the hardware.

Basic operation rates

The clock period on SYMBOL IIR now stands at 320 nsec and may be later reduced to about 200 nsec. All measurements were taken at the 320 nsec period. The basic clock period in SYMBOL IIR contains long logic chains allowing relatively complex tasks to be performed. Many of the key logic chains contain 20 to 25 levels of AND-OR logic. The system uses Fairchild CT μ L, type I throughout. The core memory is a 1964 model with a basic 2.5 μ sec cycle. Due to a semi-serial interface on the core memory it has an effective cycle of 4 μ sec,

An improved system (referred to as SYMBOL II) has been studied and has been partially specified. This system is based on the technology of the experimental system, SYMBOL IIR, but has been considerably optimized. SYMBOL II is also specified to use the latest cost orientated hardware technology. Conservative performance estimates of SYMBOL II will be made to give a comparison of how the SYMBOL algorithms would stand up in a contemporary hardware technology design. They will be based on a clock period of 100 nsec using a circuit family such as CT μ L, type II and an LSI memory with a 200 nsec period. One should keep in mind that the following comparisons are be-

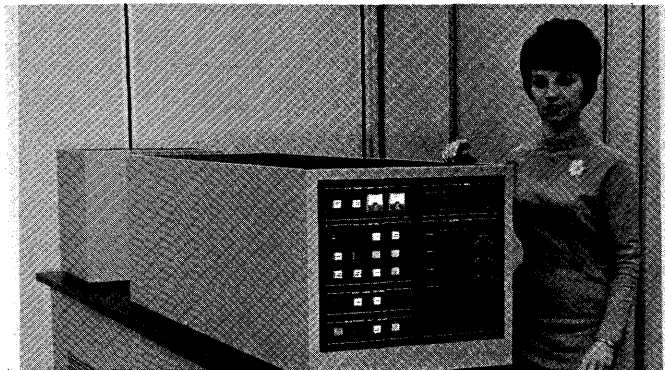


Figure 5a—The SYMBOL main frame

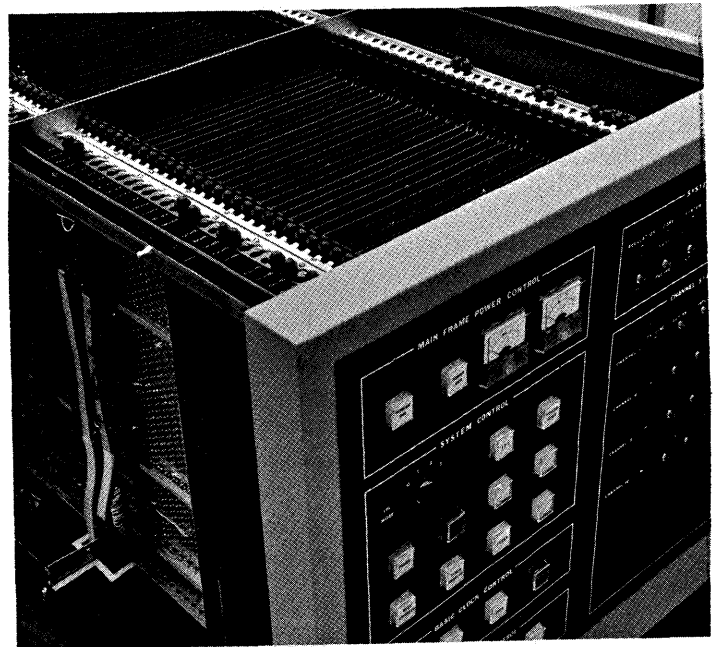


Figure 5b—Detail view of SYMBOL main frame

tween SYMBOL, which is a VFL machine running in a very dynamic execution time environment, and a more conventional fixed field machine running a language with the data boundaries determined at translate time. The former places more demands on the hardware while the latter shifts the burden of data management to the user.

For the purposes of comparison SYMBOL IIR will be referred to as SIIR and SYMBOL II as SII,

Field processing operations

SIIR performs all field operations in a VFL serial-by-character mode. It was always assumed that after system evaluation and bottle-neck analysis, if warranted, certain operators such as those shown below would be executed in a more parallel mode by using additional hardware. SII estimates are based on serial processing and known algorithm improvements that reduce or do not materially increase the hardware required.

The following table gives processing times measured on SIIR and estimated for SII. The execution time values are specified in microseconds and do not include the instruction fetch time or single word operand fetching and storing.

SYMBOL IIR MEASURED EXECUTION TIMES IN μ SEC

OPERATION	SIIR	SII
1234+4321	5.6	1.2
12345678-87654321	10.0	1.6
50 digits + 50 digits	45.0	5.0
Convert to floating point 1234	5.2	1.2
Convert to floating point 12345678	12.5	1.8
Convert to floating point 50 digits	120.0	18.0
Compare 12345678,87654321	4.0	1.0
Compare 12345678,12345670	6.5	1.2
abc join def	4.5	1.2
12345678 join 12345678	60.0	12.0
1234 format ZZZ.DD	9.0	3.0
1234 format ZBZBV	8.0	2.6
12345.6789 format '\$*C***C***,DD	76.0	15.0

Compilation

Several programs were compiled on SIIR and the overall times and space usage measured. The SIIR results are tabulated below.

SYMBOL IIR MEASURED COMPILE TIMES IN μ SEC

	NO. STATEMENTS	BYTES OF SOURCE	BYTES OF OBJECT CODE	AVERAGE TIME PER STATEMENT
Program A	195	8330	7315	820
Program B	70	3528	5112	1280
Program C	157	7560	6025	760

This represents about 75,000 statements compiled per minute on SIIR.

A comparative table for SII assuming added flexibility on SII for handling various other languages in addition to the SYMBOL language is given below. The data is based on a sampled study of object code and projected execution times of several recently developed algorithms.

SYMBOL ESTIMATED COMPILE TIMES IN μ SEC

	STATEMENTS	BYTES OF SOURCE CODE	BYTES OF OBJECT CODE	AVERAGE TIME PER STATEMENT
Program A	195	8330	2350	185
Program B	70	3528	1735	220
Program C	157	7560	2110	185

This would give a compilation rate of 300,000 statements per minute.

Paging overhead

SYMBOL has very low overhead for paging. The algorithms are based on direct hardware execution using parameters set up by software. A count of worst

case paging overhead for SIIR in terms of memory cycles for a CP page out is given below.

SYMBOL IIR PAGING OVERHEAD IN MEMORY CYCLES

ITEM	WORST CASE	AVERAGE
CP Shut Down	7	7
SS Queuing and Push Selection	50	30
SS Disc Servicing	8	6
CP Start Up	6	6
TOTAL Memory Cycles	71	49

Assuming an average of 5 μ sec per memory cycle counting internal cycles this gives 355 μ sec worst case. In SII using an improved algorithm the overhead would be less than 20 μ sec.

Input/output

The overhead for I/O for a time-sharing system becomes an important factor in providing adequate terminal response time. To illustrate the effect of the hidden software overhead an operation trace of a IBM 360/44 during FORTRAN IV output was performed. A similar operation was performed on SIIR. The equivalent output statements in both languages are shown in the table below.

The trace of the FORTRAN statement indicated 1753 instructions being executed. Each instruction requires an average of two memory cycles. The trace program does not trace any of the supervisor or channel operations so that well over 3,000 and more likely near 4,500 memory cycles were used in executing the FORTRAN statement.

SYMBOL VS FORTRAN OUTPUT STATEMENT
TRACES IN MEMORY CYCLES

LANGUAGE	STATEMENT	TRACED	EST. OVERHEAD NOT TRACED
SYMBOL	OUTPUT 12345.56 FORMAT D.DDD,DD ;	130	0
FORTRAN	WRITE (6,10)X 10 FORMAT (1X,E9.3)	3466	1000

Task control overhead

In order to measure the overhead for compilation and execution a program consisting of one CONTINUE statement was executed on SIIR. This causes a null program to be entered, translated, and executed and thus places a large demand on any system resources required, isolating overhead from "useful" actions. All

memory cycles were traced with the following distribution:

PROCESSOR USED	MEMORY CYCLES
SS	41
TR	20
CP	18
	—
TOTAL	79

This could be compared with any contemporary system where the entire compiler would have been paged in and much of the supervisor would have been executed to establish many resources that would not have been needed.

Subscripting

It would seem that VFL data structures imply slow data referencing. However, the SYMBOL project demonstrated that efficient handling of dynamically varying data can be achieved with sophisticated list processing techniques. SYMBOL IIR established the foundation and the algorithms have now been developed to be competitive with conventional fixed field indexing while retaining the VFL features. A few references and their equivalent memory cycles for SIIR are given below. The subscript Fetch cycles are not counted.

REFERENCE	TYPICAL MEMORY CYCLES REQUIRED
A[4,9]	4-6
A[16,32,6]	8-10
A[3]	2-3
A[70]	9-12
A	2

A substantial improvement has been obtained for SYMBOL II promising to make it as fast or in some cases faster than conventional indexing.

SYMBOL FROM A COST VIEWPOINT

A study of a modern computer installation and its users as a total "system" reveals where and how the computing dollar is divided. Consultants from Iowa State University made available all the necessary data for such a study early in the program.⁷ Figure 6 illustrates the I.S.U. IBM 360/50 installation in 1966 at

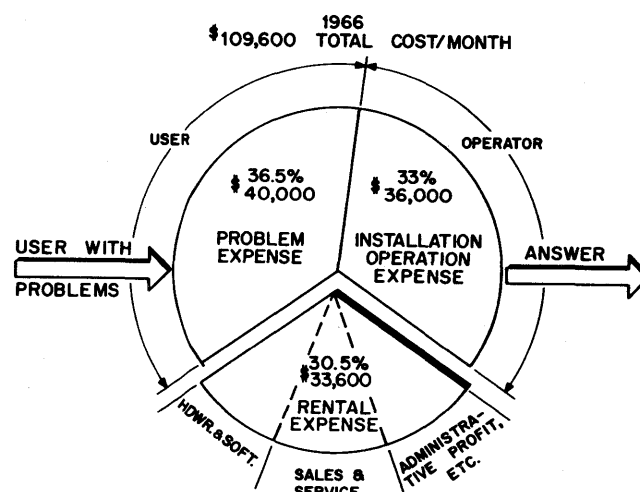


Figure 6—The computing pie illustrated for Iowa State University 360/50 installation

the time the study was made. This "pie" has since been compared with many other business and scientific installations of varying sizes with different computer systems. There is general agreement that the minor variations in the size of the slices for different installations do not materially affect the picture. This applies to most modern "classic software-dominated systems."

The objective of data processing is to solve problems where the "user with a PROBLEM" is the input and the "ANSWER" is the output. It is assumed that the user has his problem well defined and has the data available but the data is not yet programmed. The conversion of his problem to a computable language and the debugging necessary for correct execution is included in the total cost of operating an installation.

I.S.U. calculated the total system operation on this basis as approximately \$109,600 per month. The rate and labor costs were adjusted to normal commercial standards for the calculations. Both commercial and scientific problems were run in the problem mix. The following sections discuss the breakdown of the overall cost.

About 37 percent or \$40,000 is used by the problem originator and/or the professional programmer to convert the problem to a debugged, high-level language and to obtain answers.

Thirty three percent or \$36,000 is required for operating personnel, key-punch operators, file clerks, systems programmers, administration, space, power, etc.

Thirty percent of the total pie or \$33,000 goes for

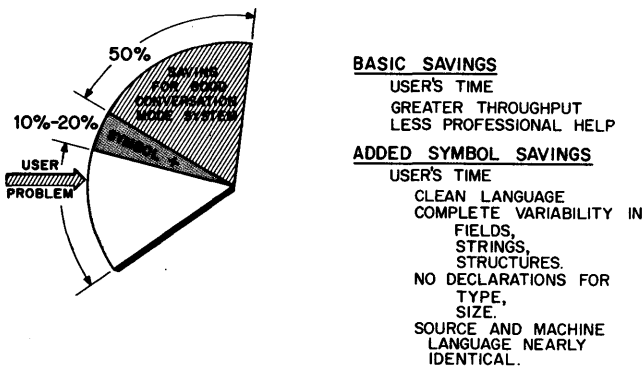


Figure 7—Savings in problem expense

machine rental. It is estimated that about one third of the rental expense goes for direct development of hardware and system software (perhaps half and half), one third for sales, service, and application support, and one third for administrative costs, overhead, and profit.

The choice of a hardware configuration and its machine language is the tail wagging the dog. Inexpensive hardware and a good, easy-to-use programming system can reduce the size (i.e., total cost) of the pie but in conventional systems will not materially alter the relative size of the slices.

In the following text the computing pie is used to illustrate SYMBOL concepts from a cost point of view. Each major slice will be further subdivided into its own percentage parts (i.e., each major slice will be 100 percent of the portion under consideration and will be divided into its constituent parts).

Figure 7 shows the potential problem expense saving to be obtained from any good conversation-mode, high-level language, time-sharing system. It has been estimated that approximately 50 percent of the problem

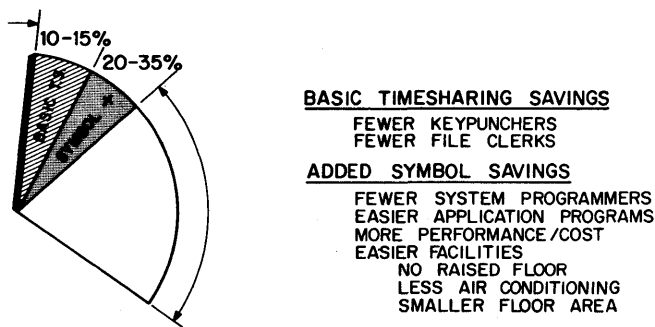


Figure 8—Savings in installation operation expense

expense slice can be saved in reduced user learning time, increased throughput, less professional programming support required, etc. We estimate the SYMBOL system will further reduce these costs with its "clean" and "concise" directly implemented high-level language and simplified operating system.¹

The savings in the operation of an installation comes from four sources. This is illustrated in Figure 8. First: A good time-sharing system will reduce the administrative help such as file clerks, keypunch operators, etc. It is estimated that this saving can be ten to fifteen percent of the installation operating expense exclusive of system rental. The SYMBOL system with conversation-mode multiprocessing and multiprogramming will also share in this saving. Second: The "system software" support required in a conventional installation is a very significant portion of the expense. Here SYMBOL shows a definite added saving. What system software remains can be written in the high-level, general-purpose language and will be easier to write, debug and understand later. This will reduce the number of professional personnel required. Third: The SYMBOL language is directly implemented in hardware and thus uses less main memory for "system software." For example, a resident compiler is not required. In addition, much less program swapping occurs and thus less virtual memory transfer time is needed. Hardware execution of algorithms is also faster and results in enhanced instruction execution speed. These features will require less programming attention and also provide more throughput per installation dollar spent. Fourth: The SYMBOL hardware is designed with modern integrated circuits and large two-layer printed circuit boards. The total system hardware package is compact and does not

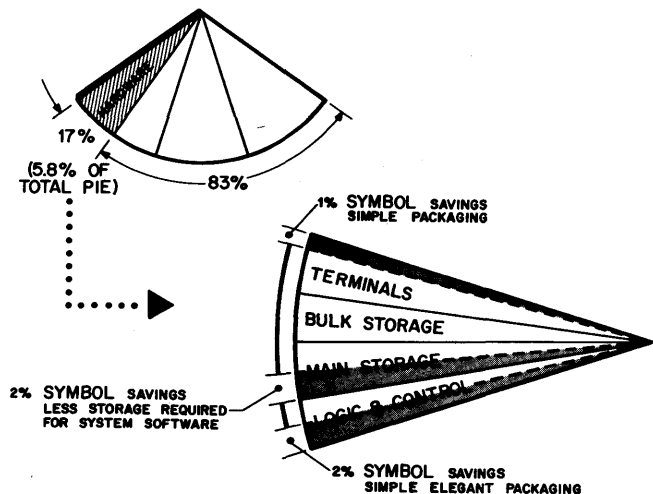


Figure 9—Manufacturer's direct hardware expense

need raised floors, special air conditioning, or vast amounts of floor space. It is estimated that these SYMBOL features will reduce installation operating expense by an additional 20-35 percent or a total of 30-50 percent.

The slice of the computing pie representing the computer manufacturer's hardware contribution is illustrated in Figure 9; approximately seventeen percent of this slice is attributable to hardware. For large systems the peripheral equipment and the bulk files can approximate about one half of the total cost. The main storage is another quarter and the CPU logic is another quarter. Naturally some variation in these amounts will occur from installation to installation and for different system types.

The SYMBOL approach saves costs in several ways: The first area of savings is in the use of large two-layer printed circuit boards and two-layer printed circuit bases with cam-operated contacts for *all* system interconnections.

Except for cables to peripherals and wires used for correction of design errors and for logical extensions no wire exists in the system. Figure 3 illustrates a logic board and Figure 5 illustrates the main frame of the SYMBOL system. This type of packaging lowers production costs for logic. It is estimated as much as a fifty percent saving will be achieved over small board, wire-wrap back panel, multi-cabinet conventional systems. This same technique reduces costs in terminal equipment but not to such a large degree. We estimate that three percent of the manufacturer's slice of the pie can be saved by this functionally-factored, bus-oriented, large printed circuit board design philosophy. The sec-

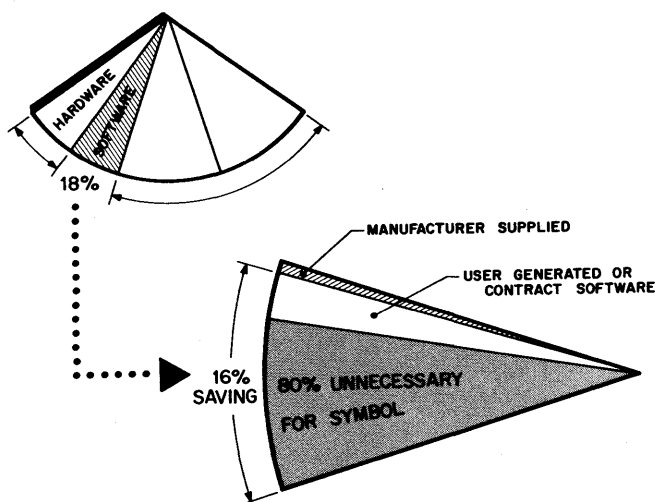


Figure 10—Manufacturer's system software expense

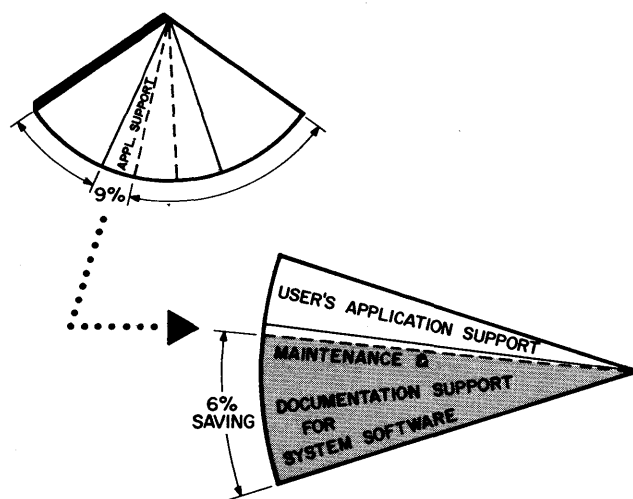


Figure 11—Manufacturer's software application expense

ond way savings are obtained is in the hardware efficiency gained by the SYMBOL system. Since most of the normal system software is hard wired, very little resident main memory is used, thus providing much larger percentages of main memory for application programs. The execution of system instructions is done at "clock speeds" in a "macro" rather than a "micro" manner. This provides much faster high-powered instruction execution. Finally, more of the system hardware is simultaneously operating due to the system organization which allows multiple jobs to be in the main frame for overlapped execution. We estimate that an additional two percent of the manufacturer's slice of the pie is saved here.

The largest and most important single saving for SYMBOL is in the "System Software." Figure 10 illustrates this point. Irrespective of whether the system manufacturer or someone else produces the software for a conventional computer this large expense is real. The SYMBOL features directly implemented in logic (i.e., hard wired) make unnecessary at least 80 percent of the conventional system software used in large time-sharing machines. This represents an estimated 16 percent saving in the system manufacturer's slice of the computing pie.

The field support of the system software is a major expense. The sheer volume of paper and record keeping to keep current with the latest changes is a major problem. In the design of the SYMBOL system this problem was given great attention. In studying the software delivered with large systems using a relatively static high-level language, we note that most (if not all) of the changes made were on the programmed imple-

mentation or were due to programming errors. Many levels of machine and assembly language programs and machine runs were between the hardware language and the programmers' source language. This quite naturally introduces confusion (and errors) either in original programming or in understanding the hidden rules when using the system.

It may also be noted that as more and more applications are programmed in a language it automatically becomes more rigid. We believe that the "clean," high-level, general-purpose SYMBOL language is excellent for most uses. Since direct hardware implementation requires little field support in the software sense, we estimate approximately a six percent saving in the manufacturer's support expense. This is illustrated in Figure 11.

Good service is a must in a large system. The SYMBOL hardware has been engineered for good reliability and at the same time easy maintenance. We do not anticipate any added expense for SYMBOL hardware maintenance over conventional systems with equivalent storage and logic circuit counts. Our experience on the SYMBOL model has verified this belief.

The previous material has split the computing dollar up in parts and has described how major savings can be realized with a "total systems" approach. The SYMBOL techniques described herein together with good time-sharing, conversation mode practice can reduce computing costs up to 50 percent. Referring to Figure 12, one may visualize how the savings in the whole computing pie add up.

CONCLUSION

The full scale running SYMBOL system has demonstrated the following:

1. A very high-level, general-purpose, procedural, "state of the art" language and a large portion of a time-sharing operating system can be effectively implemented directly in hardware.
2. Complete dynamic variability in data fields, data structures, processing length of strings and numbers and depth of structural nesting (subscripting) can be efficiently and directly implemented in hardware. Competitive execution speeds can be obtained as compared with more conventional fixed field and structure machines and in certain areas such as language translation extremely high performance rates can be demonstrated.
3. Design and construction techniques using only large two-layer printed circuit boards for all

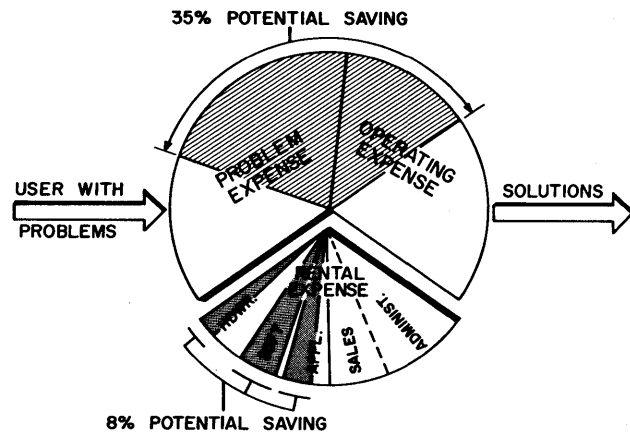


Figure 12—Potential savings with a good conversation mode hardware/software system

system interconnections and buses together with a functionally factored system results in an economical, serviceable and reliable system.

4. The direct hardware implementation of a general-purpose, high-level language, the use of the SYMBOL construction techniques and a good conversation mode system can save up to 50 percent of computing costs. This is contrasted to a good conventional system using a general-purpose, high-level batch oriented system.
5. The use of the SYMBOL system (software plus hardware) has shown that significantly fewer hidden rules exist to plague the casual or the professional user in debugging programs.

ACKNOWLEDGMENT

The authors wish to express their thanks to the Fairchild Camera and Instrument Corporation for the opportunity to do this research.

Special thanks are due to Professor Robert Stewart, Professor Arthur Pohm and Professor Roy Zingg of Iowa State University for their constructive help and support. We are most grateful to the National Science Foundation for the grant which made it possible for the SYMBOL IIR system to be delivered to Iowa State University for evaluation and continuing research on this technology.

Finally, we are deeply indebted to the past and present members of the Digital Systems Research Department at Fairchild. Their hard work and enthusiasm made this research project result in a real and operating system.

REFERENCES

- 1 G D CHESLEY W R SMITH
The hardware-implemented high-level machine language for SYMBOL
This volume
- 2 J R TENNANT G D CHESLEY
Design and layout of large integrated circuit boards
Second Annual Seminar on Integrated Circuits January 1965
- 3 A P MULLERY R F SCHAUER R RICE
ADAM-A problem-oriented SYMBOL processor
Proc SJCC Vol 23 1963 pp 367-380
- 4 W R SMITH et al
SYMBOL: A large experimental system exploring major hardware replacement of software
This volume
- 5 B E COWART R RICE S F LUNDSTROM
The physical attributes and testing aspects of the SYMBOL system
This volume
- 6 W R SMITH
System design based on LSI constraints: A case history
Digest of 1968 Computer Group Conference June 25-27
1968 International Hotel Los Angeles California IEEE
345 East 47th Street New York New York
- 7 R RICE
Impact of arrays on digital systems
IEEE Journ of Solid-State Circuits Vol SC-2 No 4
December 1967
An expanded set of references to cover work on the SYMBOL can be found in References 1, 4, and 5.

