# A LIMITATION OF HIGHER-ORDER LANGUAGES

by

David Feign

University of California, Irvine

Technical Report #82

# A LIMITATION OF HIGHER-ORDER LANGUAGES

by

David Feign

University of California
Irvine
December 30, 1975

ACKNOWLEDGEMENTS

The work on which this paper is based was originally done as part of a computer subsystem study for a vehicle avionics system at Rockwell International. I would like to thank Mr. Jaak Jurison of Rockwell International for the opportunity to do this study and for his guidance during the study. I would also like to thank Prof. Tim Standish of the University of California at Irvine for his comments in reviewing this report.

## ABSTRACT

The use of Higher-Order Languages as a programming tool tends to reduce the development cost of Software. However, there is a penalty in memory space and execution time that must be paid for the use of HOLs. In spite of the falling cost of computer hardware, there are now, and will continue to be, a significant set of computer applications where the penalty in hardware cost exceeds the Software cost savings associated with the use of HOLs. For such applications there is an economic justification for coding in Assembly language.

The tradeoff between HOL and Assembly-level coding depends primarily on the number of systems being developed for the particular application. The breakeven point is dependent on the hardware technology, the compiler available, and the size and speed requirements for the application. For any combination of parameters, there is always some breakeven point beyond which Assembly-level coding gives minimum cost.

## INTRODUCTION

Almost everyone agrees that computer programs should be written in some Higher-Order Language (HOL). It's quicker, easier, cheaper, more readable, etc. than writing the same program in an Assembly Language (AL). Yet, a survey of commercial and industrial users of computers (reference 1) revealed that nearly 1/3 of all programs were being written in Assembly Language. Why is this so? Can it be due to the clinging to old habits, or ignorance? Or can there be some good reason for using Assemblers in this day and age?

The answer is that Assemblers are still cost effective in many applications. I recently completed a design study to select a computer configuration for an Avionics system. The costs for different arrangements was derived, and the effects of software on the costs were also considered. For this system, the software was estimated to cost approximately $1,500,000 if coded in Assembly Language and somewhat less than $1,200,000 if a Higher-Order Language was used.

However, there was a penalty involved in using the Higher-Order Language: the computer would have to be faster and have a larger memory to meet the requirements. This increased the unit computer cost from about $32,000 to about $37,000. While this additional $5,000 was certainly less than the $300,000 difference in the software cost, this system was to be installed in 200 to 400 vehicles. At this rate the $300,000 savings in software cost would be more than offset by the $1 to $2 million additional for the hardware. In fact, for any number of systems beyond 60, using the Assembler turned out to less expensive overall.

This paper describes some conditions under which the use of Assembly Language rather than some Higher-Order Language is justified economically.

## BACKGROUND

While the analysis presented here was originally part of a study to select a computer configuration for a military aircraft. The analysis applies in general to any computer application which will be used in many copies. Thus, it applies to most applications of microprocessors and microcomputers.

In aerospace systems, computer hardware is configured carefully not merely from cost considerations, but also because of restrictions on allowable weight, size and power. Thus, reducing the amount of computer hardware is very important in such systems. But, as microminiaturization is continuing to reduce the physical restrictions, cost is emerging as the major factor in minimizing computer size. Therefore, cost is the item to be minimized in this analysis, and other factors, such as scheduling, and availability of trained personnel are ignored.

The costs considered in this paper are the procurement cost for the computer hardware and the development cost for the software.

Hardware-Software Tradeoffs. In the past, systems have been built by specifying the hardware first and then designing the software to tie the system together. Until complete functional specifications are available, the software cannot be designed, or even reasonably well estimated. If the computer system design is frozen before good software estimates are available, the computer sizing may be wrong. If too much capacity is designed into the system, the cost will increase; approximately linearly with memory and with the square-root of the speed (Grosch's law). If too little capacity is designed, the software cost will go up with the effort to "squeeze" the requirements into the available capacity, as shown in Figure 1, taken from reference 2.

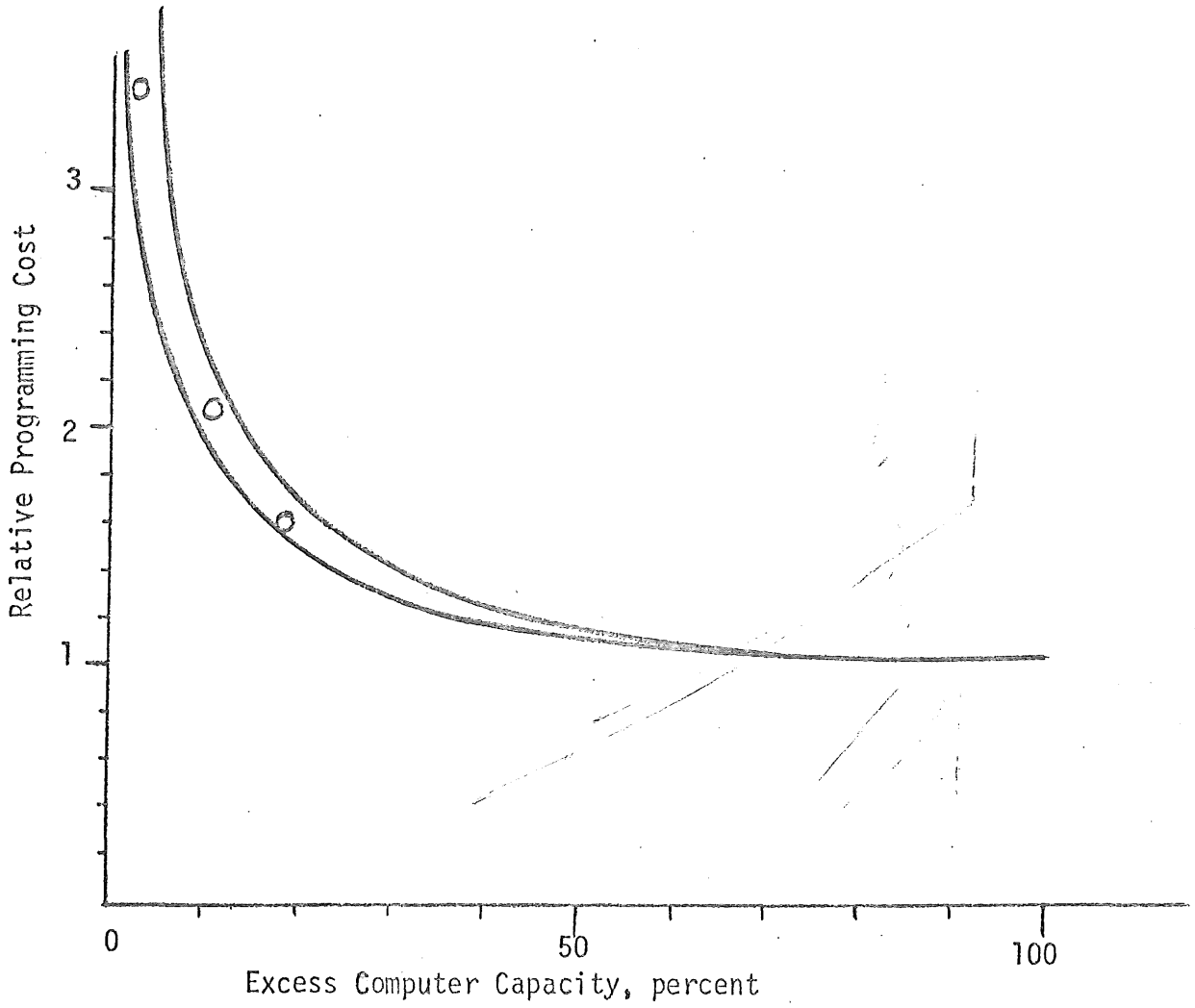



Figure 1. Effect of Computer Capacity on Programming Costs.

The original figure in reference 2 was qualitative. Three points were derived from cost data for software development for the Minuteman missile, where speed and memory capacity were restricted. These data verify the trend mentioned in reference 2.

Since many aerospace applications, and most applications of microprocessors involve hundreds or thousands, or even millions (as in calculators or automobile controls) of duplicates of the hardware, a small saving in unit cost will be worth some additional expenditure in software development cost, as long as the software cost does not climb too high up the curve in Figure 1.

Software Costs. For many years, the growing costs of developing software has been of concern to many people. Several technical symposia and workshops have been devoted to considerations of software development cost:

1. How to anticipate and estimate them,
2. How to reduce them.

A summary of the problems and considerations was presented in reference 3.

Where software and hardware are developed together, the total costs of software and hardware have been of the same order of magnitude. However, if an off-the-shelf production computer has been purchased for some application, the software costs can exceed the hardware cost by factors of tens or hundreds. The cost of producing hardware has been dropping at the rate of about 20% per year (reference 2), which means that in the future, software development costs may greatly exceed hardware purchase costs, for single computer systems, perhaps by factors of thousands.

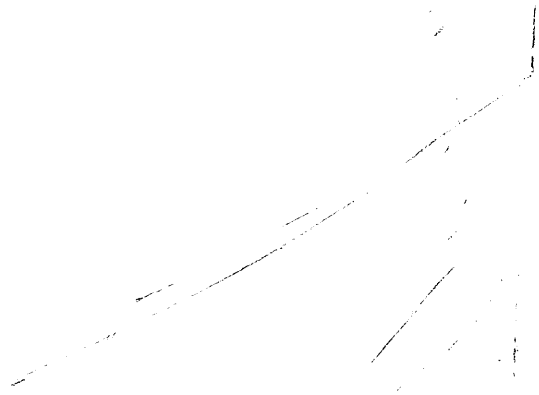
Many suggestions have been made to reduce the cost of software. Software Engineering has become a recognized discipline that is replacing the Art of Programming. Among the tools that have been developed to reduce the cost of software are Higher-Order Languages in which procedures and data organization may be expressed, and compilers for translating such procedures and data into machine code. Compilers are sometimes significant software development efforts themselves, but are usually used sufficiently frequently to justify the cost of their development. Assembly Languages are only slightly removed from machine code and require more time and effort on the part of programmers to express procedures and data organizations. They also require a detailed knowledge of the operations and idiosyncracies of the particular computer for which the code is being generated. However, this detailed knowledge permits the user to take advantage of these idiosyncracies to make more efficient use of the memory capacity and speed of the computer.

## PROGRAMMING LANGUAGE CONSIDERATIONS

In her book on programming languages (reference 4), Jean Sammet doesn't even consider that Assembly Languages are worthy of the label "programming language". Nevertheless, since assemblers are still being used for writing programs, I will refer to Assembly Language in this paper as an alternative to Higher-Order Language. The purpose of this paper is to compare Higher-Order Languages (HOLs) with Assembly Languages (ALs) from the point of view of the economics of designing computer systems for multiple-copy applications. For this purpose, we are interested mainly in one of the disadvantages of the HOLs relative to the ALs.

Advantages of HOLs. Experience over about 20 years has shown a significant advantage of HOLs over ALs in increasing programmer productivity. For the same effort, 2 to 5 times as much code can be generated with HOLs as with ALs. In addition, debugging is simpler: the compiler catches a great many errors that cannot be caught at the AL level, the code is much easier to read, and the debugging is done at a less detailed level. Also, the documentation is simplified, programmer training is reduced, and the programs, being less machine-dependent, can be transferred more easily from one machine to another.

In general, experience has shown that HOLs can reduce a part of the software development process to between 20% and 60% of the cost of using ALs. A rule of thumb that is generally used is that a programmer can produce so many lines of code in a given time. Since a single line of HOL code will be compiled into several machine instructions, while a single line of AL code generally produces only one machine instruction, the programmer is obviously more productive with HOLs.



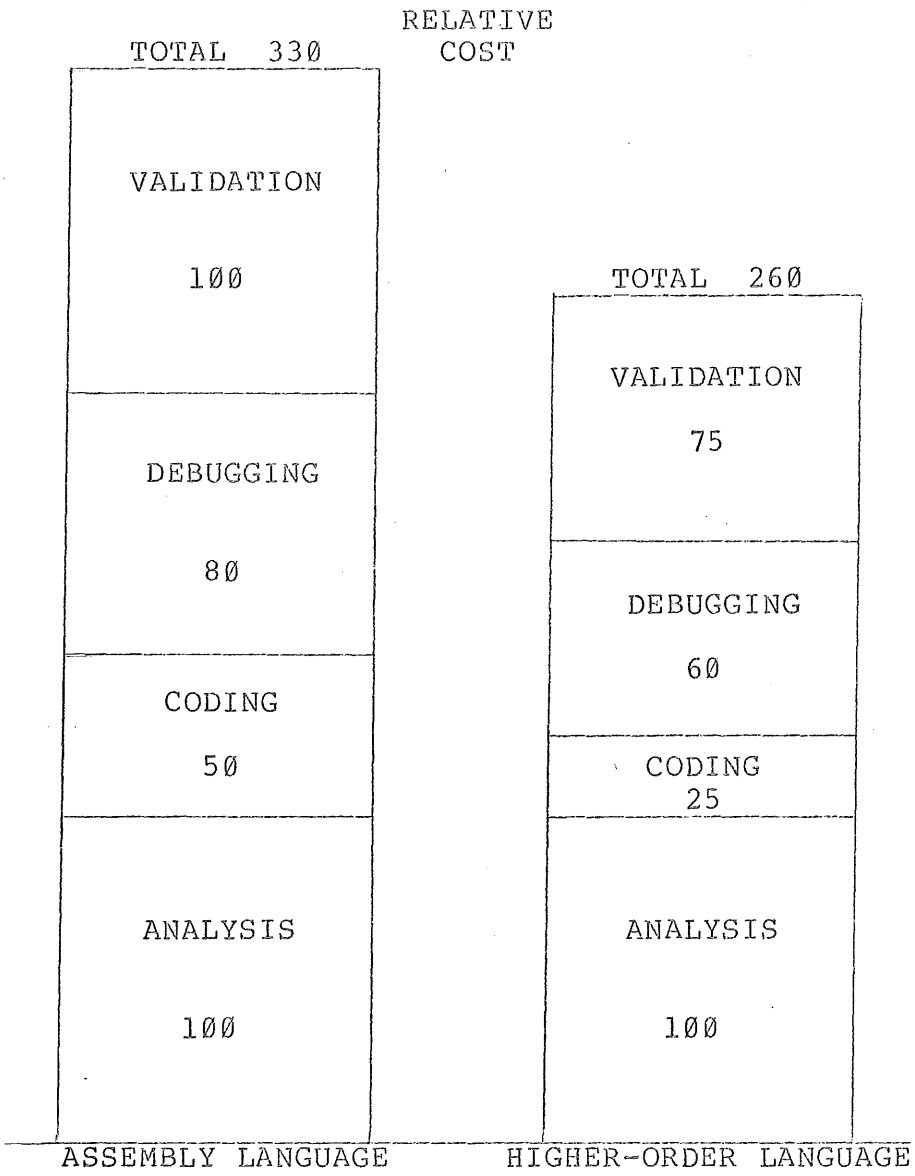



Figure 2. Relative Software Development Costs.

Disadvantages of HOLs. The code generation, debugging and testing are only part of the software development cost. Figure 2 taken from reference 5 shows a summary of Air Force experience with HOLs indicating a saving of only about 20%, although more recent experience has shown greater savings; sometimes as much as 50%. But these savings are not completely free. The development of the HOL and its associated compilers is also a significant software development that costs money. In addition, the code generated by the compiler from the HOL statements takes more memory and execution time than the code for the same function generated from AL statements by the assembler. This latter is reflected in higher hardware costs for additional memory and increased speed. The HOL and compiler development cost may be ignored if an off-the-shelf computer with an available compiler is used. The compiler cost will be buried in and amortized with the computer development cost (and therefore show up as a slightly higher computer cost, which cannot be

avoided once the compiler has been developed).

But the additional hardware penalty cannot be ignored if some large number of computers is to be purchased. Simple compilers may cause penalties of 100% or more in memory and speed, but such inefficient compilers should not even be considered for the cases discussed in this paper. Reference 5 indicates that a penalty of some 60% in space and 70% in time was experienced in some aerospace programs when HOLs were used. More recent experience, however, with a HOL designed for aerospace use, shows better performance. In reference 6, Intermetrics indicates what can be done with a compiler for HAL/S, a language designed for real-time aerospace applications. Their experience indicates that for equal effort spent with HOL and AL programming, the HOL imposes a 10% speed and 12% memory penalty. Note that these results were achieved with no saving of programming or debugging effort, since the same amount of time was spent in both cases. It is reasonable to assume, therefore, that a 20 to 30% hardware penalty is imposed by the HOL, with 10% perhaps as the limiting case.

For only a few hardware systems, this is a small penalty to pay when the software costs may be reduced 20 to 50% and the software costs may be several hundred times the hardware cost. When many systems are involved, the software, which is a development item with negligible reproduction cost, becomes small relative to the hardware. There is, therefore, some breakeven point for the number of systems involved where the cost advantage of the HOL relative to the AL is exactly offset by the hardware penalty. Beyond this point, the AL proves to be the more cost effective tool, if schedule considerations are not involved.

Again, careful consideration must be given to computer capacity if the HOL advantage is not to disappear entirely. The effect of the HOL hardware penalty on software cost under hardware restrictions is indicated in figure 3. This figure was constructed from figure 1 by superimposing the cost for HOL programming, assuming that the basic HOL savings were 50% and the HOL hardware penalty ranged from 10% to 30%.

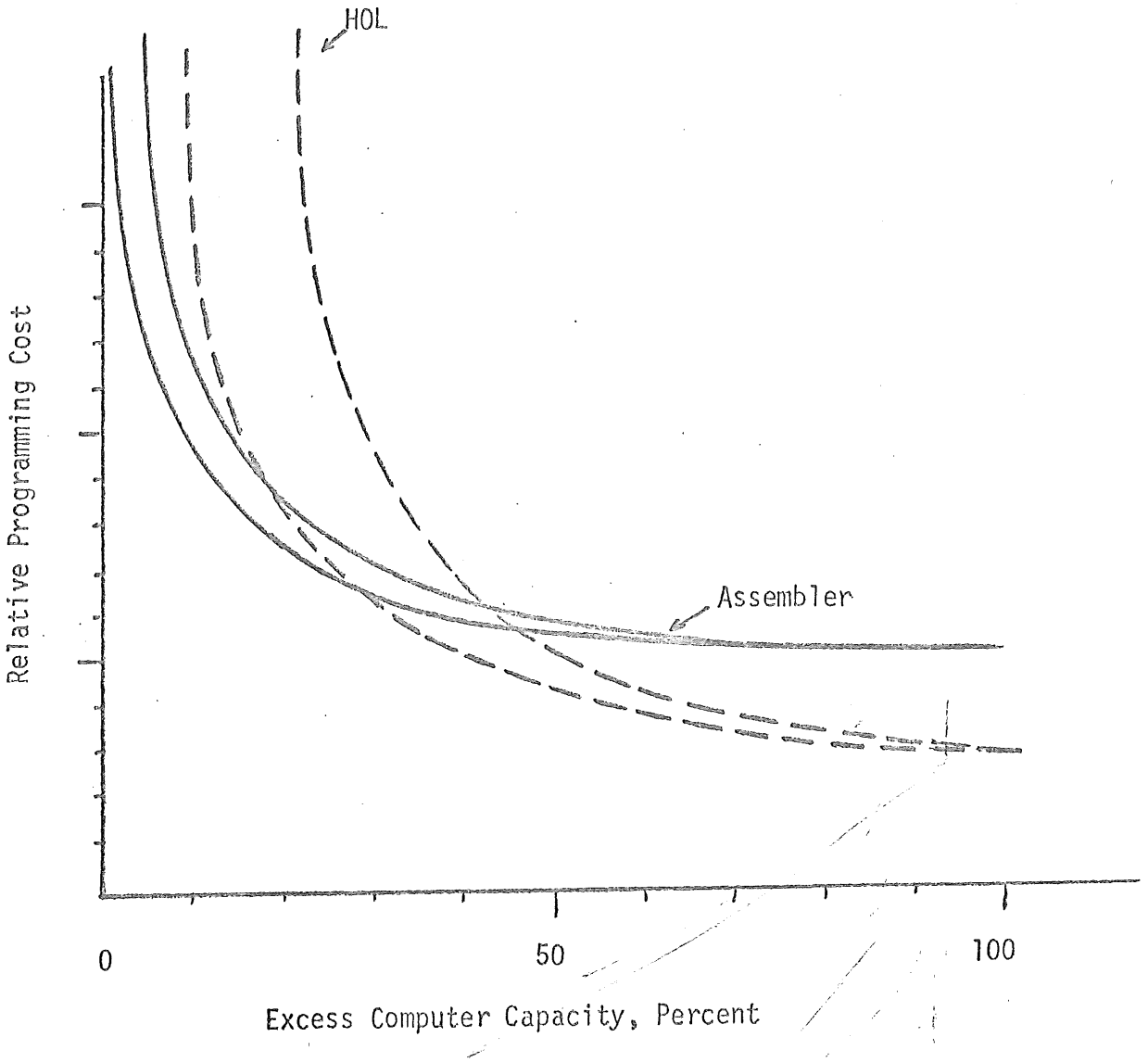



Figure 3. Effect of Using Higher-Order Languages on the Programming Cost for a Limited Capacity Computer

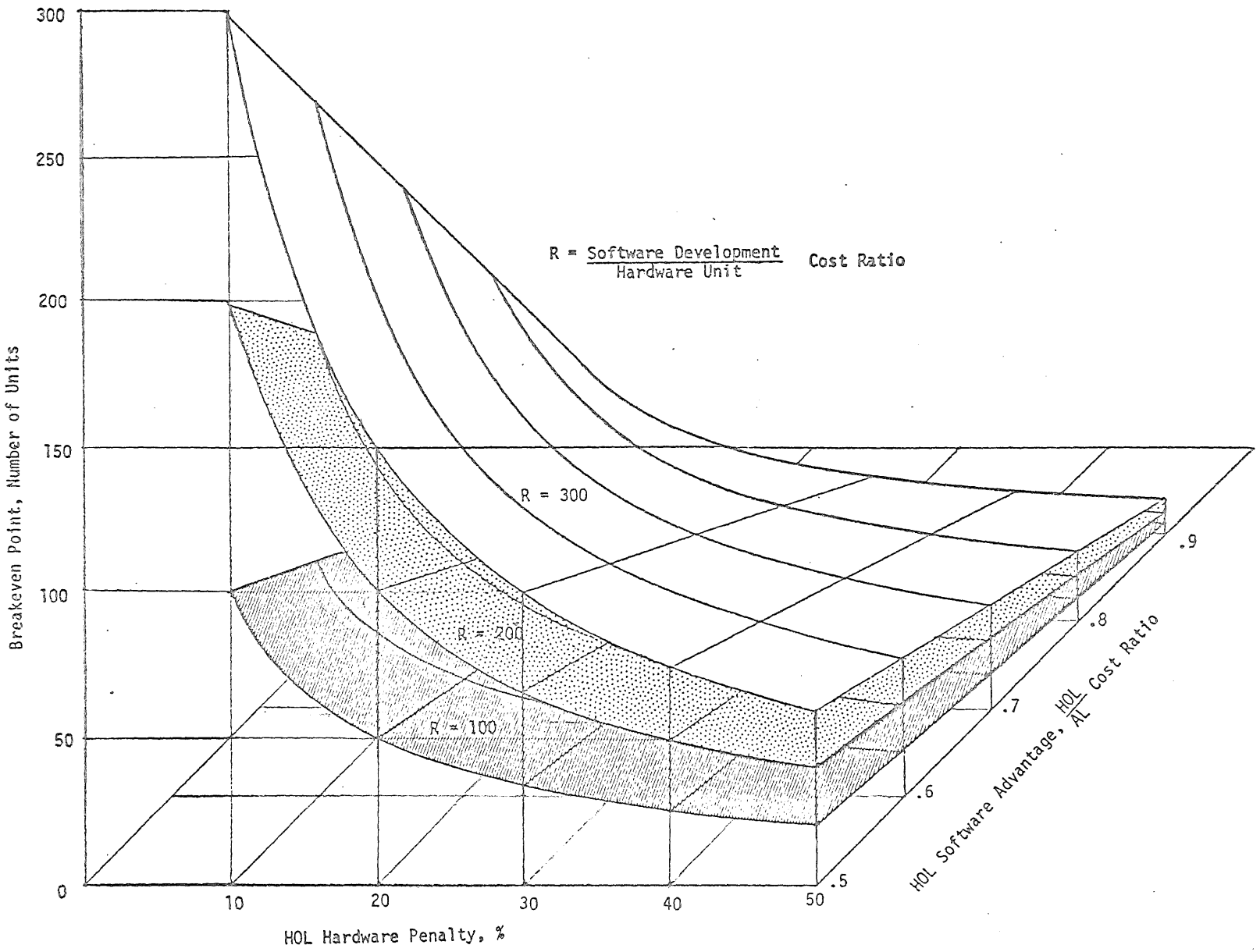



Figure 4. Breakeven Point Between Higher Order Language and Assembly Language Programming

## HOL VS. AL TRADEOFFS

In a paper presented to the ARINC Avionics Engineering Seminar on DAFCS (reference 7), E. S. Eccles showed some indications of the dividing line between HOLs and ALs in developing Avionics software. Figure 4 is a different presentation of the same ideas. It shows the breakeven point as a function of software:hardware cost ratio, the HOL cost saving, and the HOL hardware penalty. The software cost is a one-time development cost while the hardware cost is the unit cost for a single computer. The breakeven point may be simply calculated when the difference between the HOL and AL software costs equals the hardware penalty imposed by the HOL multiplied by the number of computers to be purchased.

$$BEP = (S(AL) - S(HOL))/(H(HOL) - H(AL))$$

$$= (S(AL) - S(HOL))/P(HOL) \cdot H(AL) \qquad (1)$$

where S(AL) is the software cost using AL
S(HOL) is the software cost using a HOL
H(AL) is the unit hardware cost when AL is used
H(HOL) is the unit hardware cost when a HOL is used
P(HOL) is the penalty associated with the HOL, as a fraction of the hardware cost

Expressing the above as ratios (which are more general and easier to guesstimate):

$$BEP = (1 - S(HOL)/S(AL))/(P(HOL) \cdot S(AL)/H(AL)) \qquad (2)$$

In figure 4, the vertical scale (number of units at the breakeven point) is a linear function of the software/hardware cost ratio, and so the values on this scale may be multiplied by the same factor as the cost ratio without changing the shapes of the curves (i. e. for cost ratios of 1,000, 2,000 and 3,000, simply multiply the breakeven point scale by 10). These curves, however, are quite general and indicate only trends. Specific values to be used depend on the actual costs of hardware and software. The costs involved are affected by such parameters as:

Program size (number of words in final code)

Programmer productivity (code produced, tested and documented per manmonth) which, in turn, is affected by experience and program complexity

Programmer cost (including cost of machines for producing and testing code)

Effect of language used and other software tools on productuvity

Scheduling restrictions on development

Cost of basic computer

Cost of adding memory

Cost of increasing speed.

While the information available for the numeric values to be used for the above parameters is scattered, inconsistent and measured with different tools and criteria, a simplified model that fits what observed data is available, can be developed. The costs used will be the latest available (1975) and extrapolations five years into the future (1980).

Hardware Cost Model.

The typical control system which will use many copies of the same program, generally requires a computer with between 4K and 64K of 16-bit memory and speeds between 100 and 250 KOPS (thousands of operations per second). In this range, there would be a basic cost for the CPU, I/O, power supply, assembly and incidentals, plus a cost that varied linearly with memory size. In addition, the overall cost would vary with speed. While speed is very expensive when pushing the state of the art, for the range considered here, Grosch's Law - that cost varies with the square-root of the speed - applies quite well. The basic cost model for the hardware with 16-bit word memory would then be:

$$H = N \cdot (B + 16 \cdot C \cdot M) \cdot S^{1/2} \qquad (3)$$

where H is the unit hardware cost
N is some constant
B is the basic logic and power cost
C is the cost per bit of memory
M is the memory size in words
and S is the speed in KOPS

Although for any given application, memory and speed tradeoffs can be made, as programs grow more complex, both memory and speed requirements tend to increase. For military avionics systems, from which most of the data used here was gathered, observations indicate that the tradeoffs generally make the speed increase roughly as the square-root of the memory size. Therefore, our model will be modified as follows:

$$H = N \cdot (B + 16 \cdot C \cdot M) \cdot M^{1/4} \qquad (4)$$

The costs used in this study are for computers designed to military standards. A typical MIL/SPEC computer in the midrange (16K of memory and 200 KOPS) would cost today approximately $12,000, exclusive of memory. The memory is typically 5 cents per bit or $.80 per word, or about $13,000 for 16K of memory, for a total of $25,000. Substituting into equation 2, we find:

$$\$25,000 = N \cdot (12,000 + .8 \cdot 2^{14}) \cdot 2^{7/2} \quad \text{or} \quad N = .088$$

thus,

$$H = .088 \cdot (B + 16 \cdot C \cdot M) \cdot M^{1/4} \tag{5}$$

The use of HOLs, as stated before, imposed both a memory and a speed penalty on the hardware. Therefore, the hardware cost for the two approaches to software will be:

$$H(AL) = .088 \cdot (B + 16 \cdot C \cdot M) \cdot M^{1/4} \tag{6}$$

$$H(HOL) = .088 \cdot (1 + P_S)^{1/2} \cdot (B + 16 \cdot C \cdot M \cdot (1 + P_M)) \cdot M^{1/4} \tag{7}$$

and the difference between the two will be:

$$dH = .088 \cdot (B \cdot M^{1/4} \cdot ((1+P_S)^{1/2} - 1) + 16 \cdot C \cdot M^{5/4} \cdot ((1+P_M) \cdot (1+P_M)^{1/2} - 1)) \tag{8}$$

Where $P_S$ is the speed penalty

and $P_M$ is the memory penalty

for use of HOL

For illustrative purposes, the costs thus calculated for computers covering the range of requirements mentioned above are shown in Table 1. The basic size, speed and cost of 16K of memory and 200 KOPS and $25,000 are assumed as above, and the costs calculated from equations 6 and 7 for smaller and larger computers. At the present time, basic computer cost, B, is about $12,000 and memory cost, C, is between $.02 and $.05 per bit. Cost extrapolations show that memory costs are decreasing at the rate of about 35% per year, and the rest of the computer at the rate of about 20% per year. Thus, extrapolating to 1980, we get values for B of $4,000 and for C of $.002 to $.005 per bit. For the HOL, a penalty of 25% was assumed for both memory and speed.

Table 1 Unit Computer Costs

| Nominal Memory Size | | Memory Cost, per Bit | | | |
|---|---|---|---|---|---|
| | | 1975 | | 1980 | |
| | | $.05 | $.02 | $.005 | $.002 |
| 4K | AL | 10,800 | 9,400 | 3,100 | 2,900 |
| | HOL | 12,700 | 10,700 | 3,500 | 3,300 |
| 8K | AL | 15,500 | 12,200 | 3,900 | 3,600 |
| | HOL | 18,900 | 14,300 | 4,500 | 4,100 |
| 16K | AL | 25,000 | 17,200 | 5,300 | 4,500 |
| | HOL | 31,600 | 20,700 | 6,300 | 5,200 |
| 32K | AL | 45,200 | 26,600 | 7,800 | 6,000 |
| | HOL | 59,300 | 33,200 | 9,600 | 7,000 |
| 64K | AL | 90,700 | 46,400 | 13,000 | 8,600 |
| | HOL | 122,100 | 60,200 | 16,600 | 10,400 |

Software Cost Model.

Software costs are much harder to come by than hardware costs. Records that have been kept on software costs and programmer "productivity" are not directly comparable, since different organizations have different measuring and accounting procedures. Reference 3 shows a 20 to 1 spread in programmer productivity over nearly 170 programs. Reference 8 indicates differences in individual programmer productivity of factors up to 26 to 1. Reference 2 shows cost per instruction ranging from $2.50 to $15.00 which, in turn, is less than the commonly used 160 instructions per manmonth at a cost of $4,000 per manmonth (including overhead, computer usage and profit) or about $25 per instruction. Much of the variation in these numbers comes from measuring costs for many kinds and sizes of programs with different scheduling constraints and accounting procedures.

For this study, we will take a simple viewpoint. The complexity and size of the programs will vary over a small range and all other factors will be assumed to be fixed, except for the difference between the use of HOL or AL for writing the programs. The only measure of program size and complexity will be the amount of memory used. The cost per instruction increases with size and for the kind of program being considered here, our experience indicates that for AL programs, the cost will be approximately:

$$S(AL) = \$2 \cdot M^{5/4} \qquad (9)$$

The 5/4 power was chosen to simplify the final expression used for comparison of HOL and AL, but is as good an approximation to the available data in the range considered as any. The ratio S(HOL)/S(AL) can vary from .5 to .8, although the conclusions drawn apply even if this ratio goes to zero. The values of S(AL) corresponding to the memory sizes in table 1 are:

Table 2
Basic Software Cost

| M | S(AL) |
|---|---|
| 4K | $ 65,000 |
| 8K | 156,000 |
| 16K | 370,000 |
| 32K | 882,000 |
| 64K | 2,097,000 |

## UNIT COST COMPARISON

To illustrate the influence of the number of systems used on the HOL vs. AL cost tradeoff, figure 5 shows the per unit cost of both hardware and software as a function of the number of units. The basic software costs are taken from equation (8) or Table 2, and the savings for the HOL are taken as 25%. The basic computer costs are taken from equations 5 and 6 assuming a penalty for the HOL of 25% for both speed and memory. The values can be seen in Table 1.

The conclusion to be drawn from figure 5 is that Assemblers are tools that will be with us for some time to come. For multiple copies of small control systems, they provide cost savings over the use of HOLs.

For today's computers and the class of programs considered, AL coding is more cost effective beyond a relatively small number of computers --- in the dozens. Even if the HOLs reduced the software costs to zero (100% saving), the hardware penalties would override this saving at less than 100 computers in most cases. The small arcs in figure 5 show the breakeven points as a function of HOL cost saving.

Extrapolating to 1980, the values may change somewhat, but the conclusions, in general are the same. The breakeven point is now in the hundreds of computers, but the trends are to applications requiring many copies, especially with microprocessors, where the numbers in a given application may be in the millions (for automotive applications, for example). While figure 5 is illustrative only, the general trends are clear.

Breakeven Point. The points of interest in the abovementioned figure are the crossover points where the total costs using HOLs and ALs are equal. Above this number of units, it does not pay to use HOLs for programming. These breakeven points can be calculated from equation (1) using equations 5, 6, and 8:

$$BEP = \frac{\$2 \cdot M^{5/4} \cdot (1 - S(HOL)/S(AL))}{.088 \cdot (1+P_S)^{1/2} \cdot (B+16 \cdot C \cdot M \cdot (1+P_M)) \cdot M^{1/4} - .088 \cdot (B+16 \cdot C \cdot M) \cdot M^{1/4}}$$

$$= \frac{22.7 \cdot M \cdot (1 - S(HOL)/S(AL))}{(1+P_S)^{1/2} \cdot (B+16 \cdot C \cdot M \cdot (1+P_M)) - (B+16 \cdot C \cdot M)}$$

$$= \frac{22.7 \cdot M \cdot (1 - S(HOL)/S(AL))}{B \cdot ((1+P_S)^{1/2} - 1) + 16 \cdot C \cdot M \cdot ((1+P_M) \cdot (1+P_S)^{1/2} - 1)} \qquad (10)$$

which can be evaluated for various assumptions of program size, HOL cost advantage or hardware penalty, and basic hardware costs. This model assumes that the basic software costs will not change. However, the

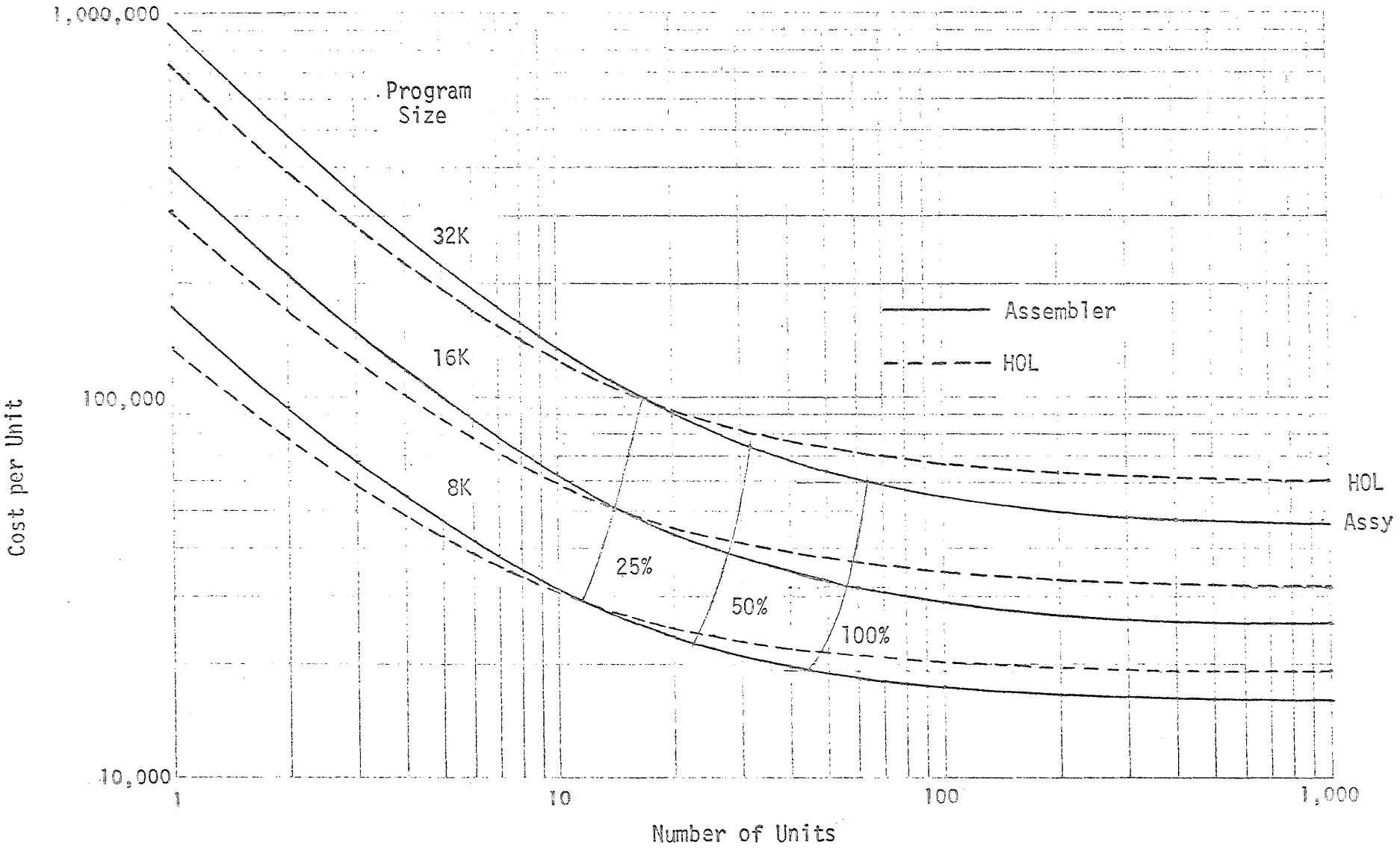



Figure 5. Variation of hardware plus software cost per unit computer as a function of the number of computers purchased

(a) 1975 Memory Cost, 5¢/bit

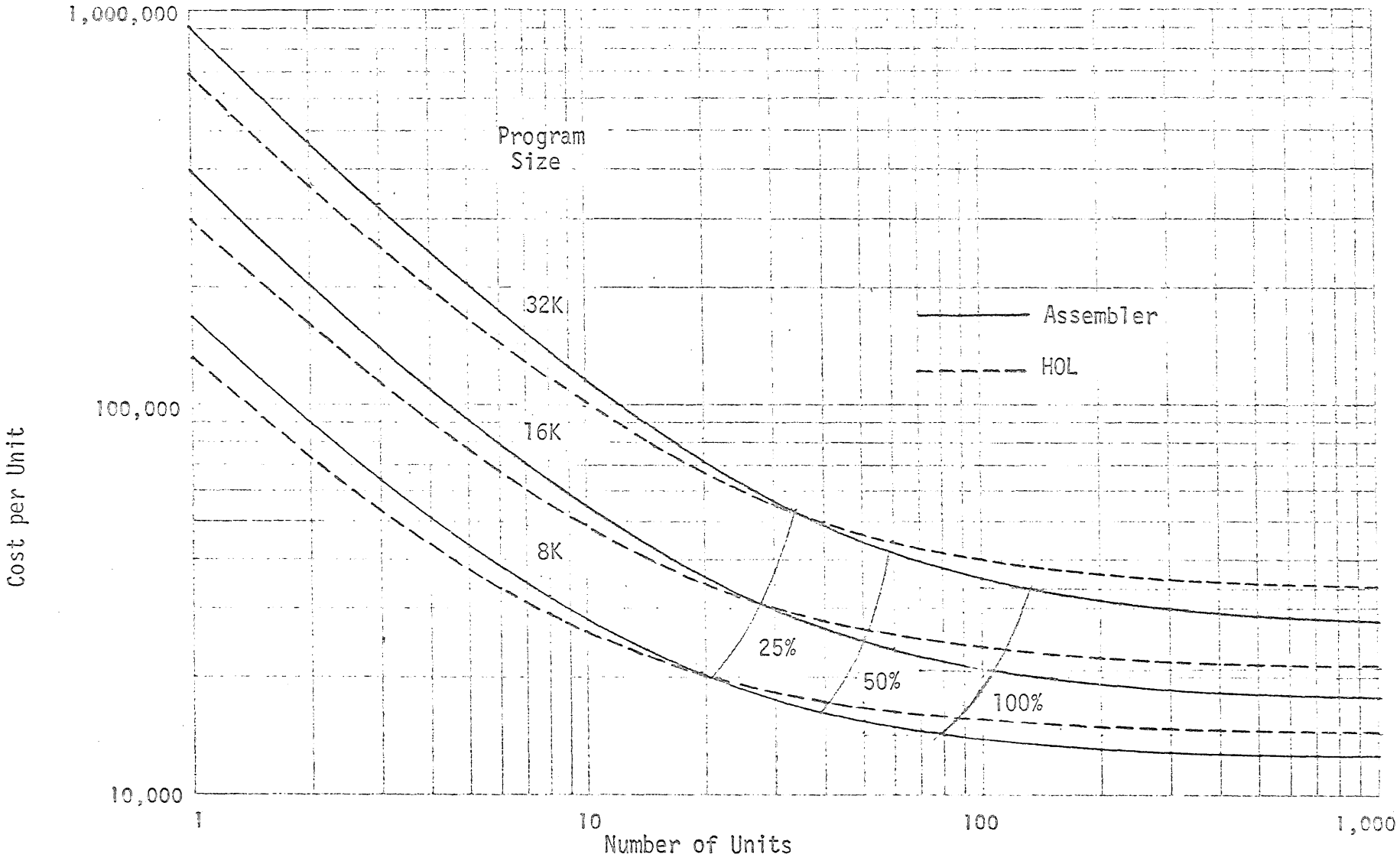



Figure 5.(Continued).

(b). 1975 Memory Cost, 2¢/bit

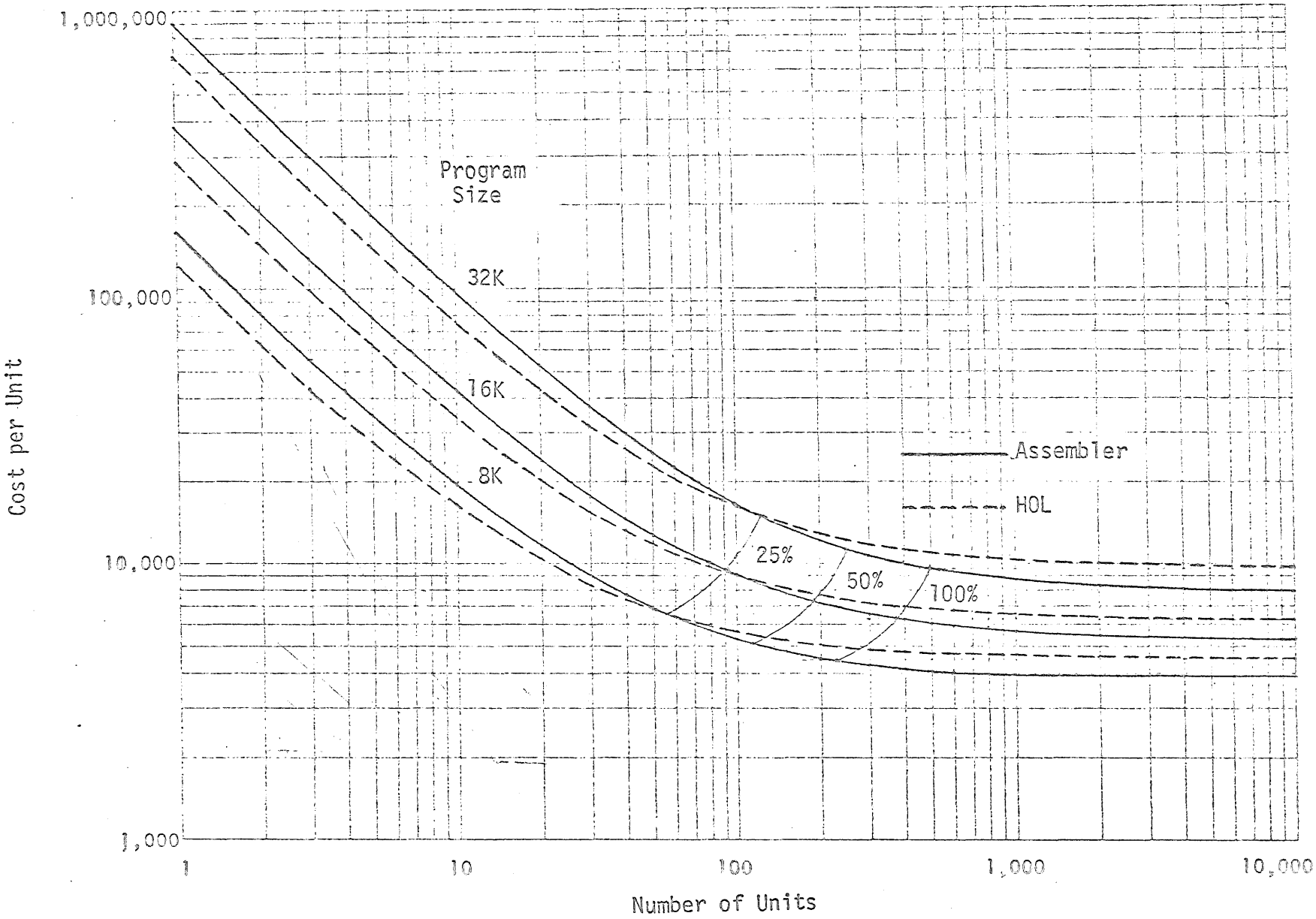



Figure 5. (Continued)

(c) 1980 Memory Cost, 1/2¢/bit

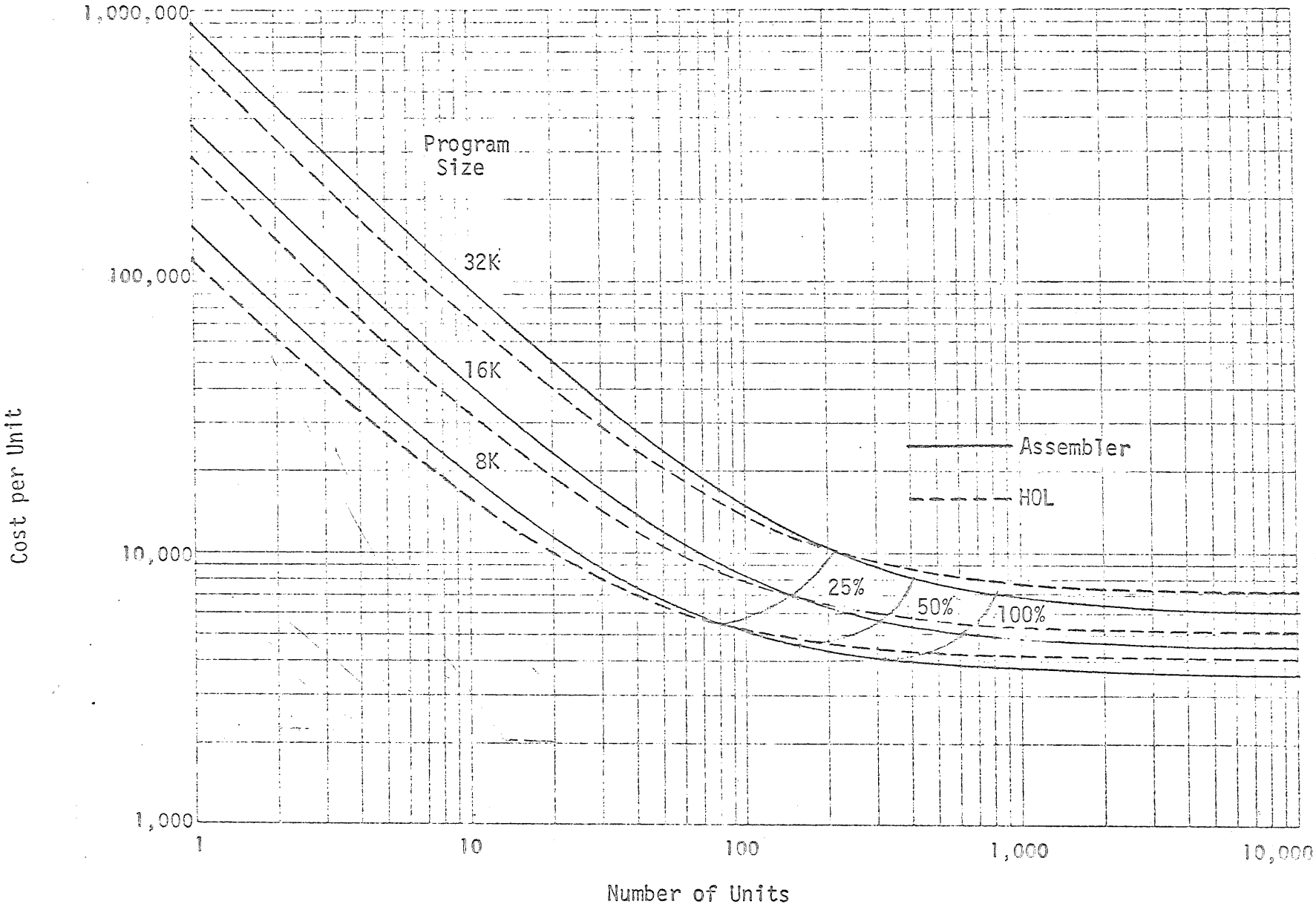



Figure 5. (Continued)

(d) 1980 Memory Cost, 1/5¢/bit

effect of a change is linear; the breakeven point increasing directly with overall software cost.

The BEP increases monotonically with program size, but is limited as M increases without bound, to:

$$1.42 \cdot (1 - S(HOL)/S(AL))/C \cdot ((P_M+1) \cdot (1+P_S)^{1/2} - 1) \qquad (11)$$

The same limit applies as the basic non-memory cost goes to zero (B = Ø).

The BEP decreases as the S(HOL)/S(AL) ratio increases, i. e., if there is little saving in HOL software, there is no point in using it. But even if the entire software cost disappears with the use of HOLs, the BEP is limited by the hardware penalty to:

$$22.7 \cdot M/(B \cdot ((1+P_S)^{1/2} - 1) + 16 \cdot C \cdot M \cdot ((1+P_M) \cdot (1+P_S)^{1/2} - 1)) \qquad (12)$$

If the memory cost becomes so low that it is negligible compared to other computer costs, the BEP depends mainly on the HOL speed penalty:

$$22.7 \cdot M/(B \cdot (1 + P_S)^{1/2} - 1) \qquad (13)$$

But, since, in many cases, speed and memory can be traded, the HOL compiler in such a case should be designed to take advantage of the low memory cost by sacrificing compactness to gain speed (e. g. by unwinding loops, replacing function subroutines by table look-up, etc.).

## REAL CONSIDERATIONS

The above analysis assumed continuous variations of cost with memory size and speed, and approximates very inaccurate data on software costs with the use of simple functions. In reality, memory comes in finite increments, usually of 4096 words, although in the future these increments will tend to be larger. For this reason, the cost penalty for HOL use may be zero if the memory penalty remains within one module, but will jump discontinuously if an additional module is required.

There is a lower limit on computer speeds that are actually available. Below this limit, there is no cost saving for slower computers. But there is also a maximum attainable speed for a single computer and, above some limit below this maximum speed, the cost of speed increases much faster than Grosch's Law indicates.

Software cost may include more than just initial development. Through the lifetime of some systems, requirements may change, or the software may be redesigned for other reasons. In this case, the software cost will not terminate. In addition, scheduling considerations may override the cost considerations. There is no doubt that HOLs permit significant reductions in elapsed time for the development of software.

## CONCLUSIONS

There is a definite hardware penalty paid for using HOLs, which means that when large quantities of computers are to be used for a given application, ALs provide a cost advantage. The breakeven point with today's technology is in the tens or hundreds of computers. With tomorrow's (1980) technology, this breakeven point may be in the hundreds or thousands. The main advantage then of using HOLs for software development lies in the schedule advantage of making modifications more quickly. A secondary advantage may be the ability to salvage a large percentage of previously developed software if it is decided to replace the computers with a different type during the lifetime of the system.

The computer system selected should be flexible and modular. Initial estimates of computer requirements may be wrong. Therefore, the system should be designed to allow incremental addition or removal of speed and/or memory capacity. This modularity should include memories, CPUs and special hardware (e. g. special functions to speed up operations, or additional registers). This will avoid the software penalty of trying to squeeze a program into available capacity (see Figures 1 and 3).

Finally, the tools used for software development should not be restricted. Assembly Language should be used in combination with a Higher-Order Language. The HOL can be used for initial design and development, with the AL used when speed or memory becomes critically close to some boundary. As shown in figure 3, the cost of using HOL exclusively will rise rapidly when an attempt is made to avoid going to the next increment of hardware module. At this point, it may pay to "tune" the code developed by the HOL compiler by using the AL, and thus follow the solid line in the figure and save the incremental hardware cost with a little additional software effort. In general, it is good practice to use, with judgment, whatever tools are available. But for any given small to medium size system, it probably does not pay to develop tools as sophisticated and expensive as a special-purpose Higher-Order Language and its associated optimizing compiler.

## REFERENCES


1. Programming Language Usage
   Andreas S. Philippakis
   Datamation, October 1973

2. Through the Central "Multiprocessor" Avionics Enters the Computer Era
   A. O. Williman and C. F O´Donnel
   Astronautics and Aeronautics, July 1970

3. Software and its Impact: A Quantitative Assessment
   Barry W. Boehm
   Datamation, May 1973

4. Programming Languages: History and Fundamentals
   J. E. Sammet
   Prentice-Hall, Englewood Cliffs, N. J. 1969

5. Aerospace Higher Order Language Processing
   Christine M. Anderson
   Technical Report AFAL-TR-73-151
   Air Force Avionics Laboratory, June 1973

6. HAL/S Configuration Inspection
   Intermetrics Corp. Presentation to NASA, July 1975

7. Software in Digital Systems - An Engineer ´s Approach
   E. S. Eccles
   ARINC Avionics Engineering Seminar on DAFCS, May 1973

8. Man-Computer Problem Solving
   H. Sackman
   Auerbach Publishers, Inc., 1970