A UNIFIED APPROACH TO THE DESIGN AND USE OF RESTRUCTURABLE COMPUTER SYSTEMS: THE META MACROMODULE MACHINE

Technical Report No. 7

June, 1968

Computer Systems Laboratory
Washington University
St. Louis, Mo.

A UNIFIED APPROACH TO THE DESIGN AND USE OF RESTRUCTURABLE COMPUTER SYSTEMS: THE META MACROMODULE MACHINE

Robert A. Ellis

TECHNICAL REPORT NO. 7

June, 1968

Computer Systems Laboratory
Washington University
St. Louis, Missouri

This work has been supported by the Advanced Research Projects Agency of the Department of Defense under contract SD-302 and by the Division of Research Facilities and Resources of the National Institutes of Health under Grant FR-00218.

ABSTRACT

A restructurable computer system offers the user an evolutionary approach to the design and use of computer systems. To support this, a unified approach is proposed in this report. A meta machine and its environment are described which provide the ability to treat a macromodular description of a system as a program to be executed or as a set of specifications from which the system may be directly implemented in macromodules.

TABLE OF CONTENTS

No. Page

TABLE OF CONTENTS

continued

LIST OF FIGURES

BLANK PAGE

A UNIFIED APPROACH TO THE DESIGN AND USE OF RESTRUCTURABLE COMPUTER SYSTEMS: THE META MACROMODULE MACHINE

1. INTRODUCTION

The macromodular project at Washington University is a particular implementation of a class of computer systems which are restructurable.[1,2] A restructurable computer system is capable of a flexibility in hardware that has long been possible only by programming. In addition, the macromodular concept proposes to make this restructurability available to the user without requiring him to be concerned with logically irrelevant engineering details. The user will have functional units of the nature of registers, adders, subtracters, etc.,whose electrical and timing details have been solved for him by the designers of the macromodules.

Although several computer and systems designs have been investigated in the course of the of the research, no one has attempted to investigate in general the operating environment of a macromodular system. In particular, the support necessary to achieve the smooth, evolutionary approach to computer design that macromodules promise has not yet been investigated. This report identifies the problems associated with this evolutionary approach and proposes specific measures to support it. The unified approach of the title refers to the ability to treat a macromodular description of a system as a program to be executed or as a set of specifications which will allow the user to directly implement the system in macromodules. Central to the approach is the concept that the user-designer may choose to implement whatever sections of the design he wishes and leave the rest to be run as a program. The paper includes an investigation, in detail, of that which is necessary to implement the unified approach.

2. THE PROBLEM AND A PROPOSED SOLUTION

2.1 RESTRUCTURABLE COMPUTER SYSTEMS

The advantages of restructurable computer systems have been well documented in the literature.[1,2] The ability to easily construct special and general purpose computers whose design is tailored to a single class of problems is a most desirable goal. Indeed the *economy of specialization*[3] may far outweigh the *economy of scale* which has been so completely taken as an absolute truth in the computer field.

2.2 MACROMODULAR SYSTEMS

The macromodular project at Washington University is an investigation of a concept of restructurable computer systems.[2] For the present, macromodules consist of a set of relatively simple, easily interconnected modules from which working systems can be easily assembled. The modules are functionally large enough to reduce logical detail by a significant amount and are relatively easy to understand and assemble. The modules are directly combined to form larger structures by straightforward mechanical assembly and easily connected cables. They have been designed so that the assembling of these units into working systems presents no logically irrelevant details such as those related to circuit loading, waveform deterioration, signal propagation delay, and power supply interactions regardless of the size and complexity of the system.

2.3 AN OPERATING ENVIRONMENT FOR MACROMODULAR SYSTEMS

While several computer systems have been designed utilizing macromodules,[4,5,6,7] very little investigation has been done concerning the operating environment and the problems that the great generality of restructurable computer systems pose. The reason is understandable: the lack of precedent in the field tends to overshadow all else. A very reasonable approach is to adopt a *wait and see* attitude with regard to any conventions or aids relating to the use of macromodules.

It is interesting to note that most of the systems which have been designed have relied on a fixed, programmable computer for support of the macromodular machine. In some cases[5,6] the support required is merely a replacement for those macromodules which have not yet been designed *i.e. Input-Output* while in other cases[4] the supporting machine is used to perform some of the operations. The comment has generally been made concerning macromodules that it is difficult to see a situation in which some fixed computer would not be required to give operating and programming support to the collection of macromodules.

2.4 PROBLEMS INTRODUCED BY MACROMODULAR SYSTEMS

There are several problems, other than the obvious one of programming support, which are inherent in the macromodular concept. For example, one may not wish to actually implement all of a design in macromodular hardware at any one time. There is a general reason for this in that the evolutionary approach to the design of a suitable configuration of macromodules is certainly desirable. This is often cited as an advantage of restructurable computer systems, but it does not automatically follow. There may be other reasons for not actually constructing the complete system. For example, there might be a problem with inventory if several users are sharing a common inventory of macromodules.

Finally, there appears to be no real reason, except for absolute necessity, for actually building special configurations. The necessity almost always concerns time in both real-time and straight computation situations.

Another problem is that if one wants to make a change in an already working system, the wires have to be changed. It is probably not feasible to make a copy of the frame wiring so it will be difficult to back up from changes which have been made but do not yet work. Analog computers have removable patch panels for interconnection of processing elements. However, these panels restrict the number and types of connections which can be made. Macromodules require the possible interconnection of any module to any other module; a feat not possible with analog computer patch panels. To further complicate matters, the physical arrangement may have changed due to the addition of new modules. It will also be very difficult to make a quick change to see what effect it has on the operation of the system. This is a very important problem and one which has been somewhat overlooked until now.

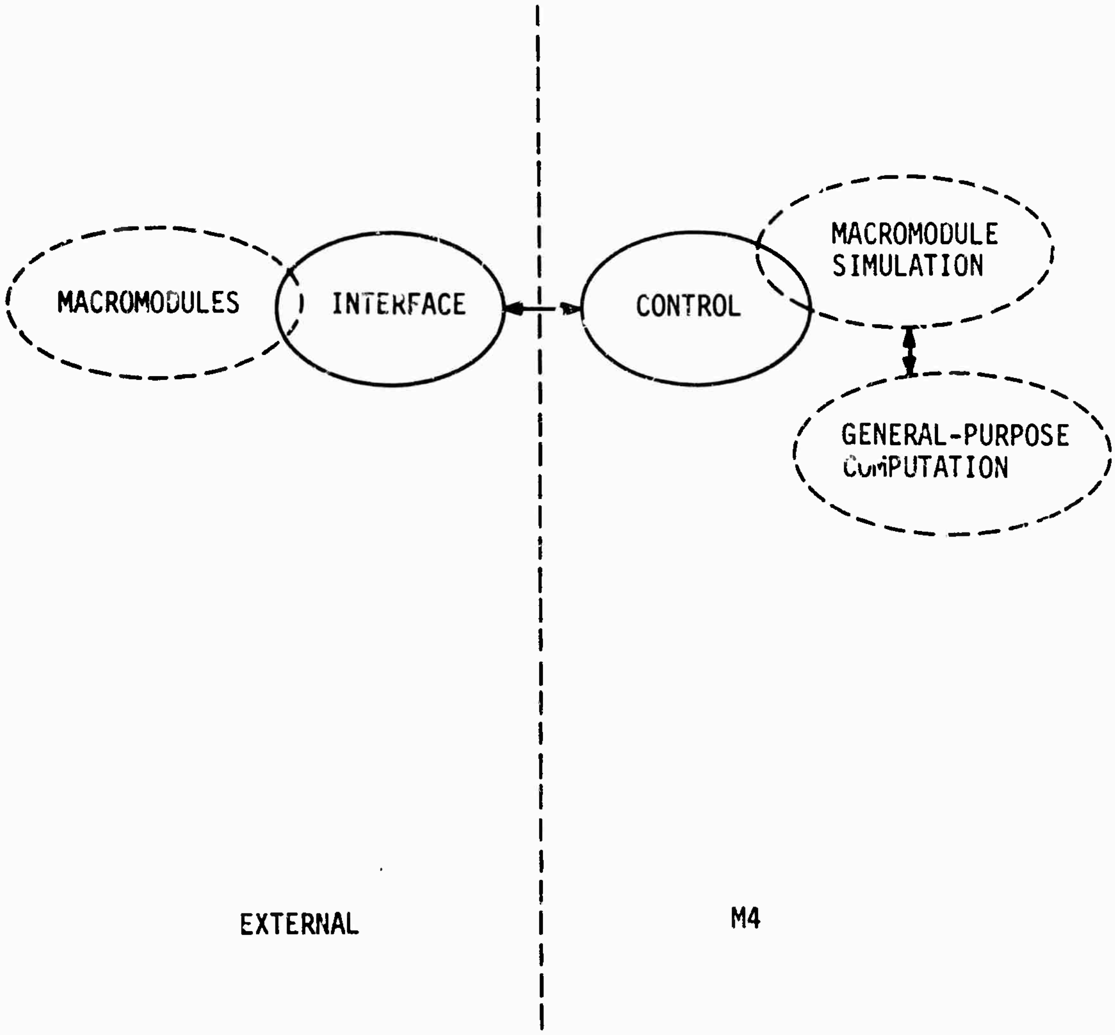
Finally there is the problem of simulation. A natural tool to use in an area of hardware development is functional simulation. Simulation allows one to largely correct a design before building it and to try out proposed changes. With simulation comes a very large overhead. It seems difficult to improve the approximately 1000 to 1 ratio in time with any of the simulation efforts to date.[8,9,10] The 1000 to 1 figure is actually quite optimistic because it is quite easy to increase the ratio several times by inefficient code or increased power and flexibility of the simulation effort.

2.5 A UNIFIED APPROACH

The unified approach in this report refers to the capability of treating a description of a macromodular system as a program which may be executed or as a set of specifications which will allow the user to directly implement the system. It is proposed in support of the unified approach, that an essentially fixed machine be constructed whose primary function is to route control and data transfer operations as specified by a description of a macromodular system so as to mediate between actually implemented functions and those which must be simulated. This function defines a class of machines; however, enough of the details of the external appearance required of these machines will be defined that this class is reduced to a specific machine for the purposes of this paper. Although an analogy between this description and a program has been introduced here, the reader should be cautioned not to expect this meta macromodule machine *M4* to have the typical order code organization of conventional stored program machines. Obviously, the M4 must have suitable features to enable it to interact with the external macromodular structures. A common, machine-readable description of a macromodular system can now function as a program for the meta machine or as a specification for actually constructing all or parts of the system. Functions which must be simulated are identified as internally implemented functions while those actually constructed from macromodules are called externally implemented functions. With proper design it is possible to switch actively between internally and externally implemented functions at the lowest possible level of macromodular primitive operations.

Figure 1 shows the several components of this unified approach. The macromodular component consists of the user-designed functions which are implemented by macromodules. This component is not discussed in this paper. The interface, which partly overlaps the macromodular component, permits communication between externally and internally implemented macromodular functions. The overlap area consists of the conventions which must be observed in implementing the macromodular component in order for it to work with the interface. The entire interface is specified in detail in Section 4.

The M4 control permits internally implemented macromodular functions. The macromodule simulation component, which is only a part of the M4, provides the capability to perform internally implemented



THE COMPONENTS OF THE UNIFIED APPROACH

Figure 1

macromodular functions. The M4 control plus those areas of the macromodule simulation which directly interact with it are discussed in Section 3.3. The details of simulation of macromodules by hardware or software is not of great interest in this report, but some considerations are presented in Section 3.4 and one particular scheme is discussed in Appendix 6.4.

Finally, the details of the general purpose computation component are beyond the scope of this report; however general comments regarding this component appear throughout the report.

2.6 ADVANTAGES OF THE UNIFIED APPROACH

The lack of irrelevant engineering detail required in the design of macromodular systems puts a description of such a system on the level of assembly language programming. The description can ignore such details as actual physical arrangement of modules and wire placement and leave them for the construction phase. This means that the description and program for the M4 can be truly identical in all respects. The proposed meta machine approach requires that only those critical portions of the design need to be implemented in hardware. Also, proposed changes may be made to the programmed structure, checked out, and only then built with actual macromodules.

Initial studies indicate that the meta machine can typically simulate non-memory operations at a ratio of 100 to 1 in time and that 50 to 1 is perfectly feasible. See Appendix 6.2 for supporting calculations. Memory operation times approach to a 1 to 1 ratio between externally and internally implemented functions. The meta machine can also be used to provide programming support for the macromodular configuration and in any of the supporting roles which require a stored-program computer. Finally, it can be seen that the meta machine can itself be constructed from macromodules. We call this the macromodular meta macromodule machine M6. Of course in the final sense, the meta machine should be constructed in a fixed form except for certain sections.

Because the M4 is essentially fixed, a compiler for higher level languages can be written for it. Then, these languages could be used for macromodular descriptions although because of efficiency of equipment considerations their results would probably only be executed internally to the M4. If this higher level language work were done with the unified approach in mind, it would be possible to describe the solution algorithm in these higher level languages in a compatible manner, particularly if operating efficiency could be sacrificed. See Appendix 6.3 for an example.

2.7 PROBLEMS ASSOCIATED WITH THE UNIFIED APPROACH

The use of a meta macromodule machine is not without problems. The distinction between hardware and program-implemented structures must be at the lowest possible level. This means that it is possible to implement one register, or one add operation, or even a single control branch as hardware in the operating environment envisaged here.

Another problem occurs with the basic unit of storage, the register. It is impossible for a register to exist both as actual hardware and simultaneously as a storage location within the memory of the meta machine because of the problems of maintaining both images. Once a register exists in macromodular form, all references to it must refer to the actual register.

A problem also exists in the control area. Naturally a means must be provided for control to cross the boundary between hardware and program in both directions. This is reasonably straightforward until we consider concurrent processes. Concurrent processes must be executed in an equivalent sequential order within the meta machine and any implemented in macromodules must be executed in parallel if at all possible.

Naturally the meta machine must be capable of representing macromodular systems subject to the constraint of indefinite extension. In general this must be accomplished without resort to changing the meta machine.

3. DISCUSSION OF THE META MACROMODULE MACHINE

3.1 INTRODUCTION TO THE DISCUSSION OF THE META MACROMODULE MACHINE

In this section the organization, operation, and implementation of the meta macromodule machine are discussed. The basic requirement of the meta machine is that it be able to route control and data transfer operations and communicate with external macromodular structures as specified by a description of a macromodular system. Questions of the operations of the M4 are required to support the unified approach are discussed in terms of macromodular designs.

3.2 ORGANIZATION OF THE META MACROMODULE MACHINE

The most obvious organization of the M4 is as a basic computer whose order code consists of the primitive operations available with macromodules plus control and data options which permit communication with external macromodules. A suitable programming language could then be developed to accompany this machine. For example, any of the functional macromodular simulation languages could provide a model for this development,[8,9,10] or see Appendix 6.4 for a specific example.

This approach is less than satisfactory, however, because a given macromodule can often be used in several different ways. For example, a data gate macromodule may be used to transfer data into a macromodular register or to indicate data to be stored into a macromodular memory. Indeed, it is possible for a single, multiple-length data gate to perform both of these operations simultaneously. This is in contradiction to the simulator languages which treat these as two distinctly different operations because of the lack of context dependence in such languages. Also, should new slightly different macromodules be designed, it is just possible that the functional language descriptions could not handle such new modules.

The requirements imposed by command oriented languages are awkward when compared to the usual macromodular design process. A macromodular system is completely described by a picture of the connectivity of data processing operations and a flowchart of the control operations which are performed. This notation is described as a uniform and consistent scheme of notation in Appendix 6.1. A far more direct approach would be to find some suitable way to enter the diagrams and flowcharts into the meta machine and have the execution directly oriented to the actual internal operations of the macromodules. Because of the above, all of the material in this report is presented independent of any actual machine language code for the M4.

3.3 OPERATIONS OF THE META MACROMODULE MACHINE

Now the internal operations of the meta machine are described which are required to support externally implemented macromodular functions. The operations which must be supported are the transfer of data in both directions across the M4-external interface and the ability of the M4 to generate and accept macromodular control signals. The external aspect of these operations is discussed in Section 4. For the present, we are not concerned with the rest of the M4.

It is assumed that registers and control points are properly marked in the description so that the M4 may make the decision between external or internal implementation. For an example of a suggested symbolic marking technique, see Appendix 6.1. A more detailed possibility for the marking is presented in Appendix 6.4. For external implementation, a number must be associated with each unique register and control point to serve as a reference for the purpose of crossing the interface between the M4 and externally implemented functions. It is possible that extra memory will be required in the M4 to provide space for this marking data, but this must be accepted in an approach with as much generality as is being proposed here. This marking function must be a centıal part of the M4 design so that all internal operations may make use of it.

3.3.1 DATA FETCH FROM EXTERNAL REGISTER

Figure 2 shows the operations necessary to fetch the contents of an externally implemented macromodular register. The number of the external register, which is available as a consequence of the operation of the M4, is first transferred into the External Register Select Register. Control then exits to the external environment and fetches these data as shown in Section 4.2. After the data have been made available to the M4, control returns to the M4 for processing of the data. Note that only one such operation is executed at one time because as far as the M4 is concerned all data are transferred sequentially in 12-bit segments.

3.3.2 DATA STORE INTO EXTERNAL REGISTER

Figure 3 shows the internal operations required to store data into an externally implemented macromodular register. The operations are analogous to those of Section 3.3 1 except that control exits and returns at a different set of control points.

3.3.3 SOME INTERNAL CONTROL OPERATIONS

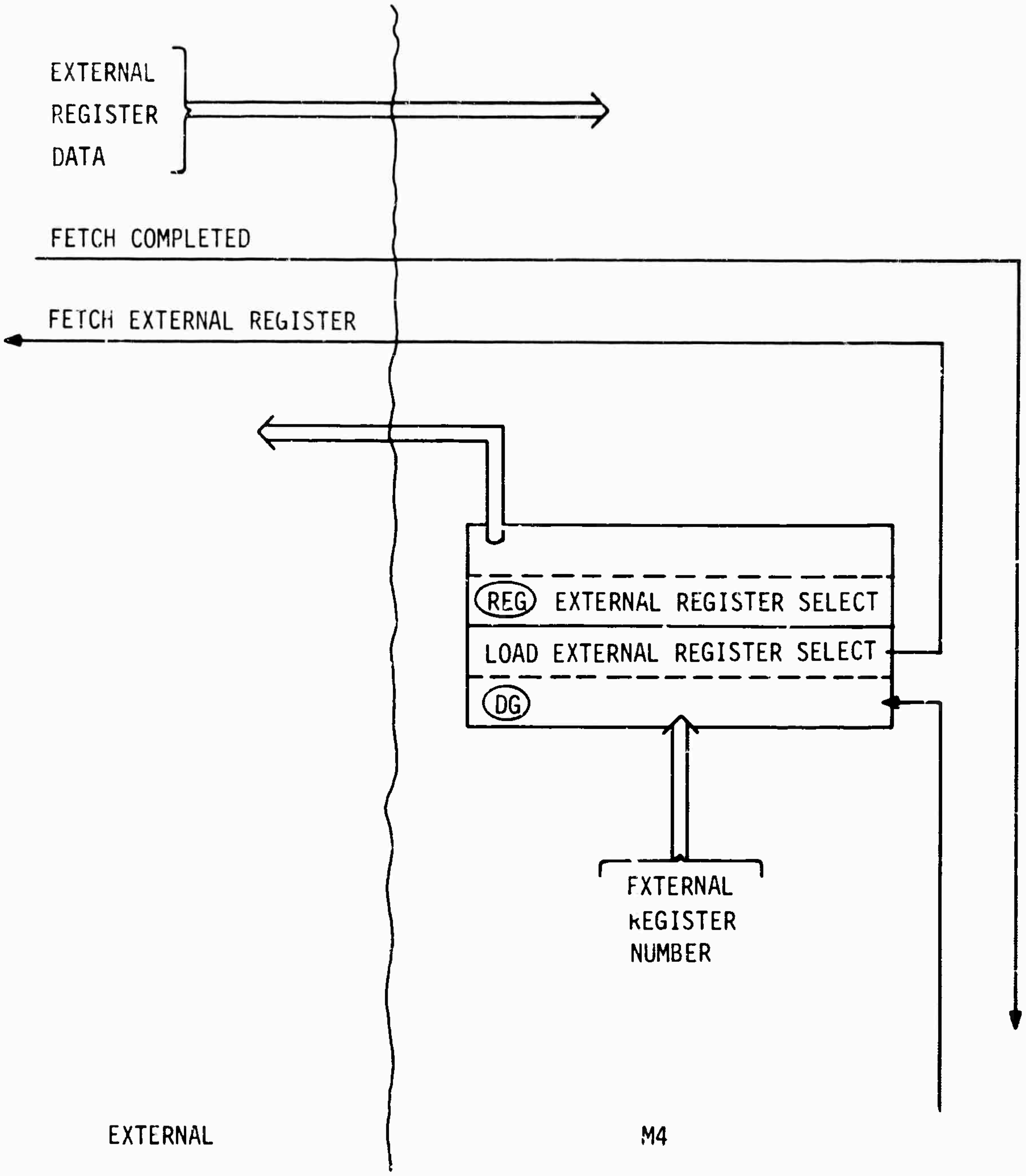
Before discussing control signal generation and return, it is necessary to discuss, in general terms, the internal M4 operations required to start the next internally specified macromodular function. Figure 4 shows the operation when strictly sequentia. functions are specified.

If the next function is internally implemented, it is decoded and executed in a form similar to the typical stored-program computer. If the function is externally implemented, an external control signal is generated as explained in Section 3.3.4 and then control enters a wait status. Central to the control structure of the M4 is a stack, which can be either first in-first out or last in-first out, which is used to remember control operations which cannot immediately be executed. Returning external control, as will be shown in Section 3.3.5, forces a pointer to the next internal function to be performed into the control stack. The wait status therefore simply causes a pause in operation until a pointer becomes available at the top of the stack.

Concurrent control requires that at least two functions be performed at the same time. This two-way splitting or branching must be capable of operation for all combinations of external and internal functions specified as the branch operations. Also, it is required that all concurrent sequences which are externally implemented be truly concurrent with each other and with internally implemented functions. Finally, it is required that internally implemented functions are executed in correct sequential order even if concurrent processing is indicated. Figure 5 shows the design for initiation of concurrent control.

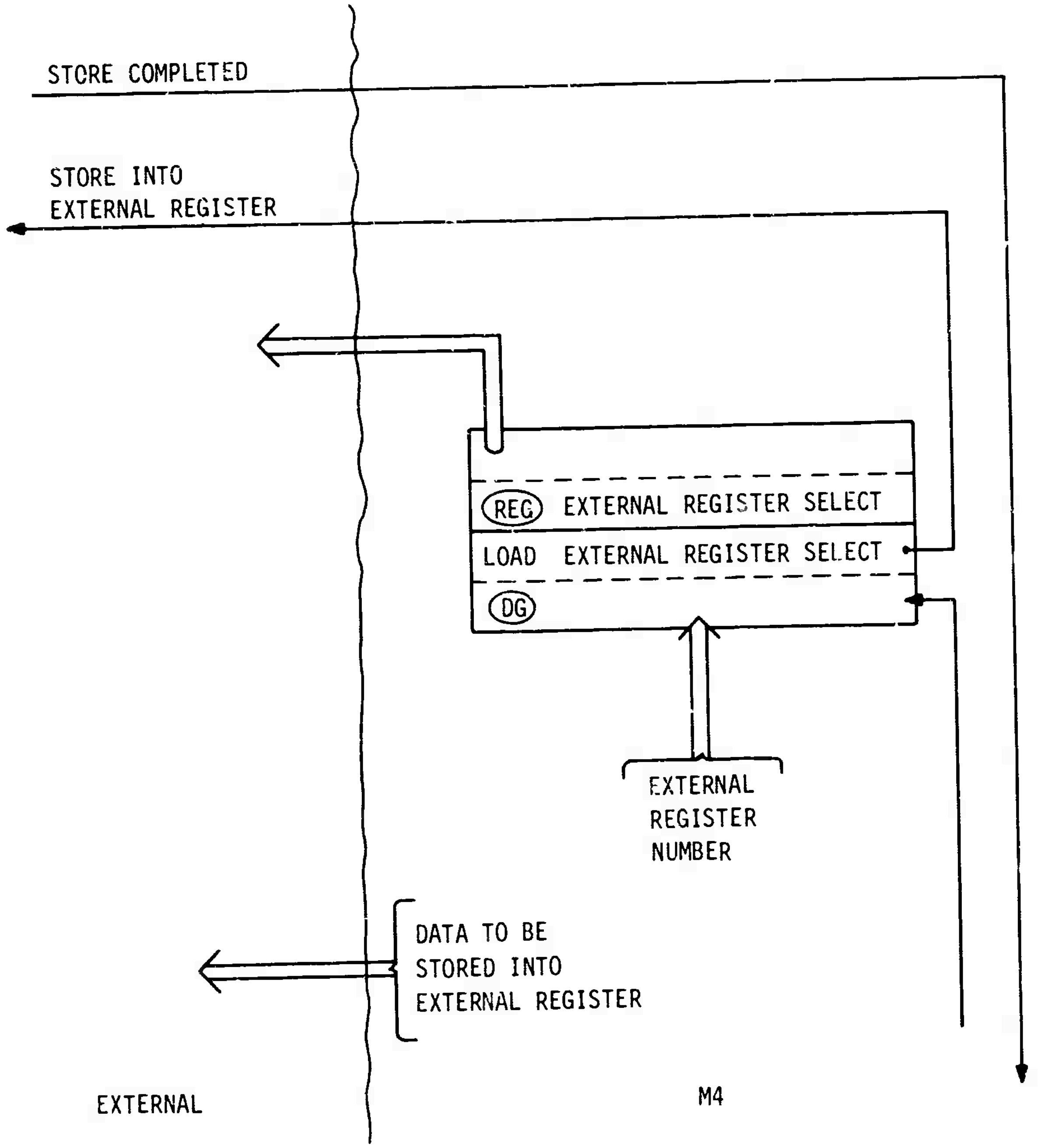
3.3.4 EXTERNAL CONTROL OUTPUT

Figure 6 shows the design for generation of external control. Concurrent operations are possible; therefore the rendezvous is used to protect the operation of storing a new control sequence number.



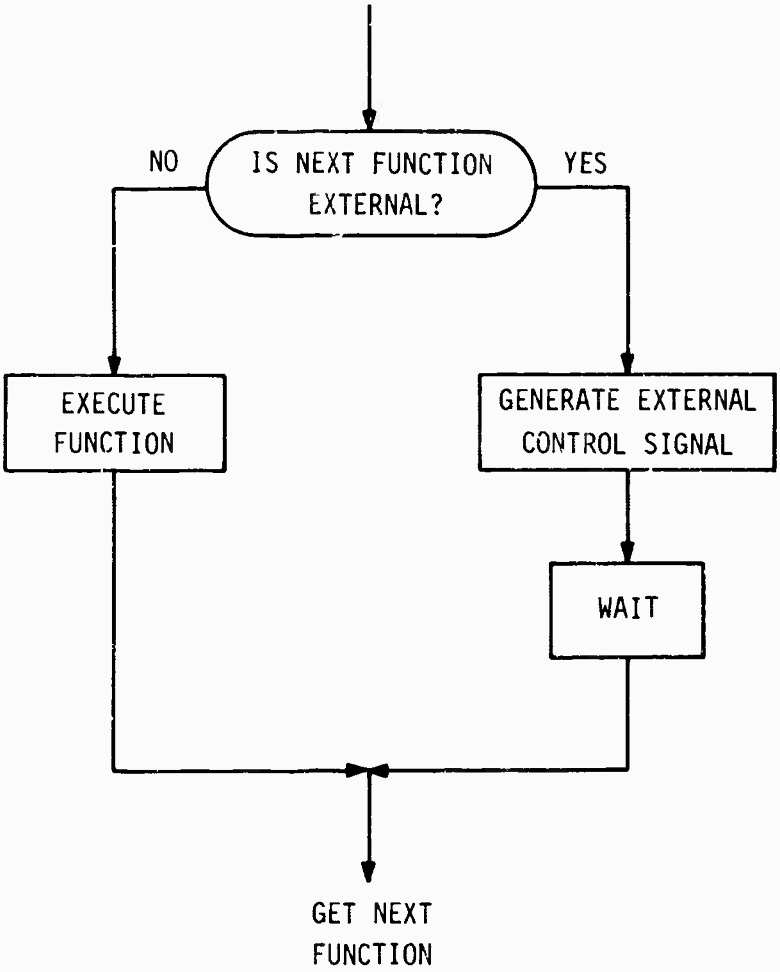
DATA FETCH FROM EXTERNAL REGISTER

Figure 2


STORE COMPLETED
STORE INTO EXTERNAL REGISTER
REG EXTERNAL REGISTER SELECT
LOAD EXTERNAL REGISTER SELECT
DG
EXTERNAL REGISTER NUMBER
DATA TO BE STORED INTO EXTERNAL REGISTER
EXTERNAL
M4

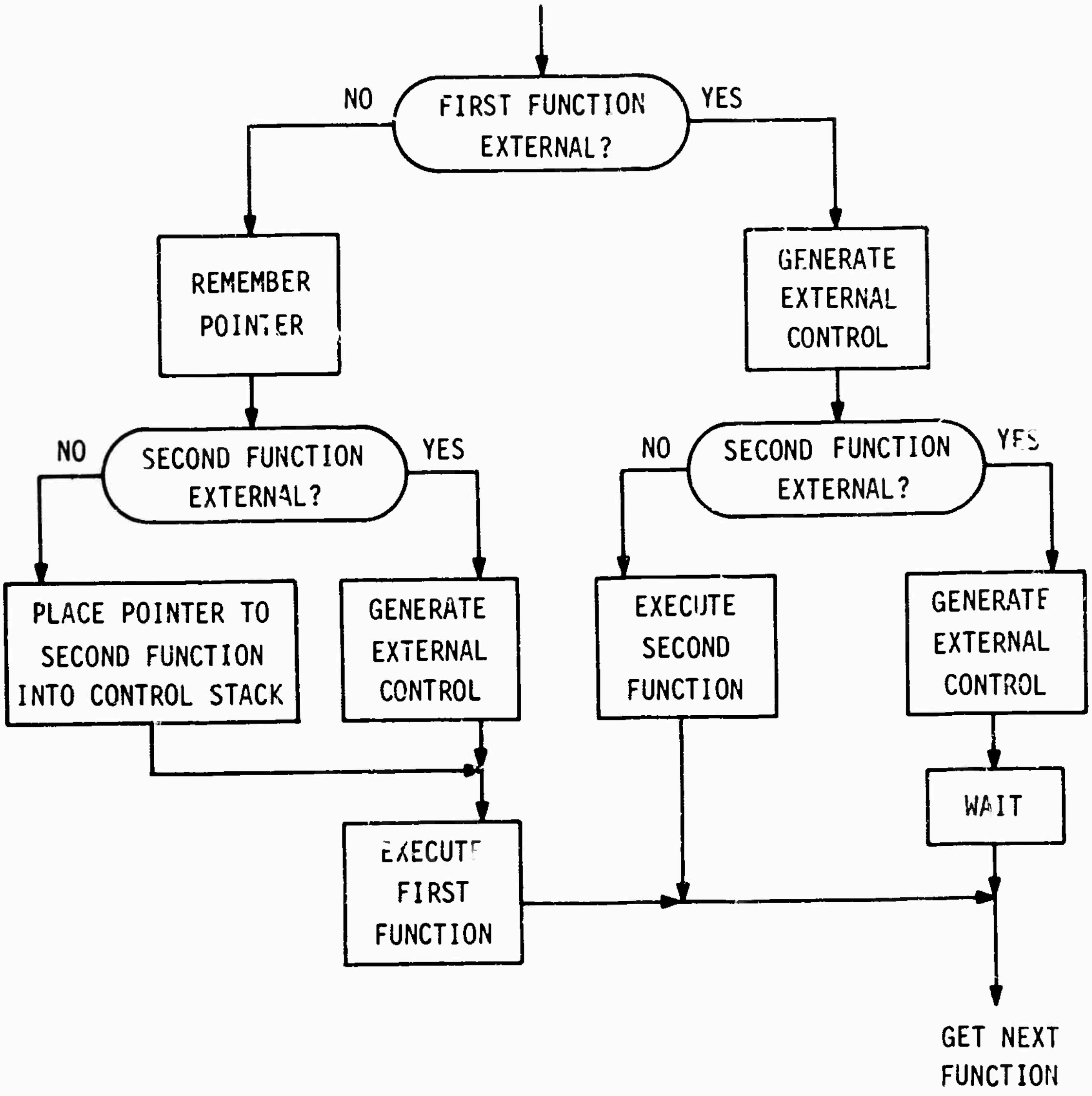
STORE DATA INTO EXTERNAL REGISTER

Figure 3



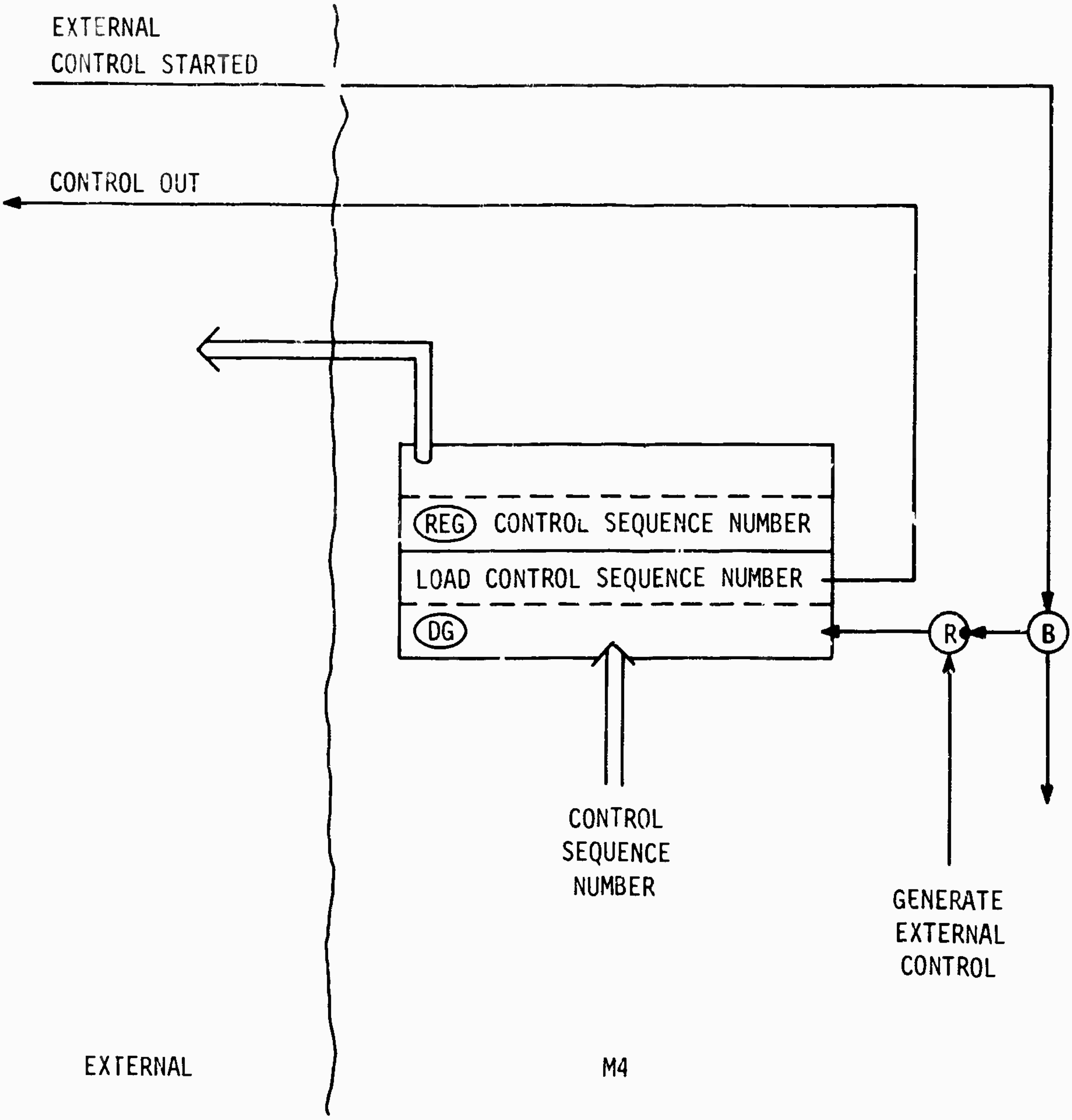
SEQUENTIAL INTERNAL CONTROL

Figure 4


FIRST FUNCTION EXTERNAL?
NO
YES
REMEMBER POINTER
GENERATE EXTERNAL CONTROL
NO
SECOND FUNCTION EXTERNAL?
YES
NO
SECOND FUNCTION EXTERNAL?
YES
PLACE POINTER TO SECOND FUNCTION INTO CONTROL STACK
GENERATE EXTERNAL CONTROL
EXECUTE SECOND FUNCTION
GENERATE EXTERNAL CONTROL
WAIT
EXECUTE FIRST FUNCTION
GET NEXT FUNCTION

CONCURRENT INTERNAL CONTROL

Figure 5



EXTERNAL CONTROL OUTPUT

Figure 6

The dot by one of the rendezvous inputs indicates that the rendezvous is preset with that input active. Note that the returning control signal *External Control Started* indicates only that the externally implemented sequence has been started and a new sequence number can be loaded, not that the external control sequence has actually been completed.

3.3.5 EXTERNAL CONTROL RETURN

Figure 7 shows the design for the M4 operations which process external control sequence returns. The interlock is required because control returns occur asynchronously with respect to other M4 operations. The M4 memory must contain a table which relates returning control sequence numbers to the place in the description where internally implemented functions are to be resumed. The pointers, when read from memory, are placed into the control stack where they are eventually read and used by the M4.

3.4 IMPLEMENTATION OF THE META MACROMODULE MACHINE

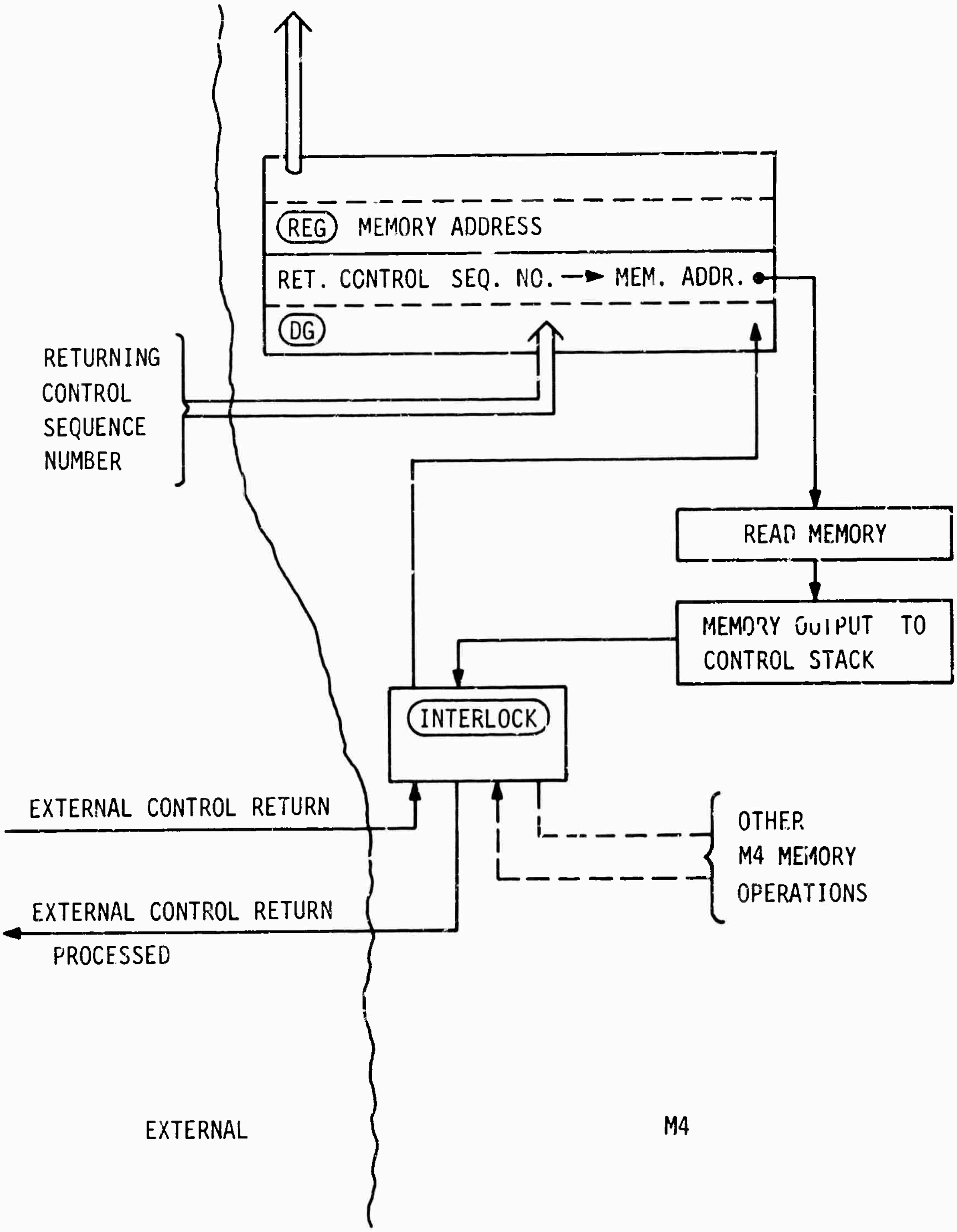
This report has intentionally remained general and has not specified any particular implementation of the M4. In this section some comments are made concerning several possible implementations. Each implementation has its own particular characteristics which may indicate which implementation would be better for a particular application.

3.4.1 MACROMODULAR META MACROMODULE MACHINE

The most obvious implementation, in macromodules, is described in detail elsewhere[11] and is summarized in Appendix 6.4. This implementation is flexible in that newly defined macromodules can be added to the meta machine at the cost of changing the internal operations. Of course, if the set of macromodules is closed, then a considerable overhead is inherent in the design using macromodules. Note that a class of instructions which are not macromodular operations must exist for this machine. This is necessary to control input-output devices which make the machine capable of general support of macromodules.

This implementation represents an extreme in the continuum of M4 implementations; it combines maximum hardware, ease of interface with external macromodules, maximum speed of simulation, and minimal space required for the specification of the macromodular system. The speed of simulation is essentially the 100 to 1 figure mentioned as an upper bound on speed in Section 2.6.1. However, maximum hardware is indicated by the count of over 100 data processing modules required, making this at least a medium-sized computer. This figure does not include any general-purpose computation capability or instructions which provide general programming support; however, these can be supplied by a very elegant method discussed in Section 3.5. A single 4096 word memory macromodule can store the specification for a system containing approximately 500 macromodular operations and 100 register macromodules, exclusive of any memory macromodules. Typically, a small computer, such as a PDP-5 contains approximately 40 macromodular operations and 6 register macromodules while a reasonably large machine, such as one designed for linear programming problems, might have approximately 300 macromodular operations and 40 register macromodules.

The use of the macromodular meta macromodule machine *M6* requires that the user have available a more than basic inventory of macromodules because so many are required for its construction. Its use may be indicated if the user does not have a suitable existing computer or if his existing computer does


REG
MEMORY ADDRESS
RET. CONTROL SEQ. NO. → MEM. ADDR.
DG
RETURNING CONTROL SEQUENCE NUMBER
READ MEMORY
MEMORY OUTPUT TO CONTROL STACK
INTERLOCK
EXTERNAL CONTROL RETURN
EXTERNAL CONTROL RETURN PROCESSED
OTHER M4 MEMORY OPERATIONS
EXTERNAL
M4

EXTERNAL CONTROL RETURN

Figure 7

not lend itself to interface with macromodules. Of course, if the user does not have the talent locally to interface his existing machine with macromodules, he may be forced into this implementation if he wishes to use the unified approach.

3.4.2 FIXED, MICRO--PROGRAMMED MACHINE

An implementation which would permit the addition of new macromodule types, if they did not vary drastically from the present general concepts, and yet not require the overhead associated with macromodules, consists of a fixed machine which could be microprogrammed by a fast read only memory. This would be a most attractive implementation, which represents an intermediate approach; however, it involves the designing of a completely new computer tailored toward simulation of macromodules and is not discussed further here.

3.4.3 EXISTING MACHINE

Another implementation could utilize a macromodule simulation program on an existing machine with some interface to macromodules. Such an effort is underway in the laboratory using a LINC (Laboratory INstrument Computer) and a LINC-macromodule interface designed by an associate of the author.[12] The availability of the LINC software which includes a macro facility and the ease of use and programming make this a natural choice. Macromodule simulation exists for the LINC,[8] but does not embody the concepts presented in this report. A useful aspect of this type of implementation is that the machine language of the existing computer can be used whenever such use is more appropriate than use of the macromodular functions.

The requirement that the user construct an interface between his existing machine and macromodules is one of the most important characteristics of this implementation. It does not seem reasonable that the designers of macromodules solve this problem except by providing an M4 macromodule. This would require that the unified approach become the standard method of using macromodules. This is a very real possibility, but it is too early in the development of macromodules to do more than speculate.

If the user chooses this implementation, he has provided himself with the opposite extreme to that discussed in Section 3.4.1. This implementation provides the slowest speed of simulation combined with a minimum amount of hardware. Most existing small computers can reasonably be used in this implementation. The machine should be capable of indefinite extended precision arithmetic, and it is convenient if it is a 12 bit computer Of course, the user also has all of the features of the computer to use in any way he desires, not an insignificant advantage.

Our experiences with the existing LINC macromodule simulation effort indicate the parameters associated with this implementation. The LINC is a 12 bit computer with an 8 microsecond cycle time and 2048 words of memory. The simulation consists of a set of macros and subroutines assembled to form a LINC program which simulates the operation of a macromodular system. The simulation soeed for *12 bit* macromodule operations is approximately 1300 to 1 as compared to real time macromodular operations. This ratio essentially increases linearly with increased length of simulated macromodular operations. Systems with a maximum of about 70 macromodular operations can be simulated. There is a direct trade between number of simulated registers and the amount of simulated memory. However, systems with only 70 operations almost never involve more than a dozen or so registers leaving 1024 words of 12 bit memory for simulated memory macromodules. If an interpretive simulation technique were used, the number of simulated operations could be increased with the penalty of decreasing the simulation speed by a factor of two or three. Of course, a 2048 word memory is extremely modest for even the typical small computers currently available.

The use of a large scale computer as an M4 has not been considered at all in this report. Neither have such things as several users of macromodules time sharing a large centrally located computer been discussed here. It is felt that the inclusion of these topics is not necessary in the present discussion.

3.5 CONCLUSIONS

This section has discussed the meta macromodule machine which supports the general concept presented in this report. The evaluations of alternate organizations and implementations have been left to future research. A few general concluding remarks seem to be in order at this time. Possibly the foremost concern in the specification of the meta machine must be to provide a machine which is straightforward and easy to use. In fact it should be possible to work with a specification of a macromodular system for execution on this machine directly at the lowest possible level and to by-pass any software support for some operations. This is a particular concern of the author, but the serious student of computer systems will see that a great amount of inefficiency may result. However, it must be remembered that this machine is intended to simulate hardware operations implemented in terms of macromodules. Thus a description of 1000 macromodules represents a large hardware system while a program of 1000 instructions is quite a small system.

The foregoing does point up one problem, however, that exists in some implementations and organizations of the M4. If the M4 is used to provide general software support for a collection of macromodules, then some programming operations are needed. The organization of the M4 may not be well suited to programming because, for example, there need not be any addressing or indexing flexibility provided. Two solutions have occurred to the author. One, which has been rejected for lack of elegance, would provide a set of instructions to be used in general programming of the M4. A more elegant approach would be to design a suitable machine in terms of macromodules and then use this description as an interpreter for a more convenient machine language. This is certainly elegant and is probably feasible from a pragmatic standpoint because any demanding computations would be performed by externally implemented macromodular operations. Compilers and arithmetic computation oriented machines can be designed for macromodules[5,6] which could then be efficiently interpreted by a macromodular description in the M4.

4. DISCUSSION OF EXTERNALLY IMPLEMENTED MACROMODULAR FUNCTIONS

4.1 INTRODUCTION

In Section 3.3 the use of externally implemented macromodular registers and control was discussed in connection with the meta macromodule machine. In this section the conventions are discussed which have to be observed in implementing external functions so that the interface between them and M4 can be standardized.

4.2 REGISTER GROUP

Figure 8 shows the conventions which must be followed to permit communication between external macromodular registers and M4. The dotted line indicates the physical and logical boundary of the M4. As can be seen, quite a large number of macromodular operations must be implemented by the user; however, they are quite regular and should pose no problems.

The example shows two registers implemented externally to the M4. One register is 12 bits and the other is 24 bits wide. Additional data processing modules are shown on the registers to indicate that the registers have externally implemented transfers, adders, etc., which involve them. The user must number the registers and provide the proper decoding of control signals. Because only 12 bits of data can be transferred at one time, each register module in a more than 12 bit register must be considered as a separate register for the purpose of these transfers.

It is the responsibility of the M4 to establish the proper data out signals and then generate a macromodular control signal on the proper line. The M4 processing is stopped until the proper return control signal is received. Because of this responsibility on the part of the M4, the user actually has very little to be concerned with when he implements the required operations.

There is a problem in that the overflow indication cannot be transferred from the M4 to the external registers. Therefore it is not possible to perform an internal operation on an external register and then have the correct overflow indication transferred to the register. The only difficulty is that an external test of overflow will always indicate no overflow after an internal operation. This is a rather small point but still one which the user must remember. A possible solution would be to build some macromodular registers which are capable of having overflow turned on by a control signal. However, from the pragmatic point of view it is felt that the user can live with the problem.

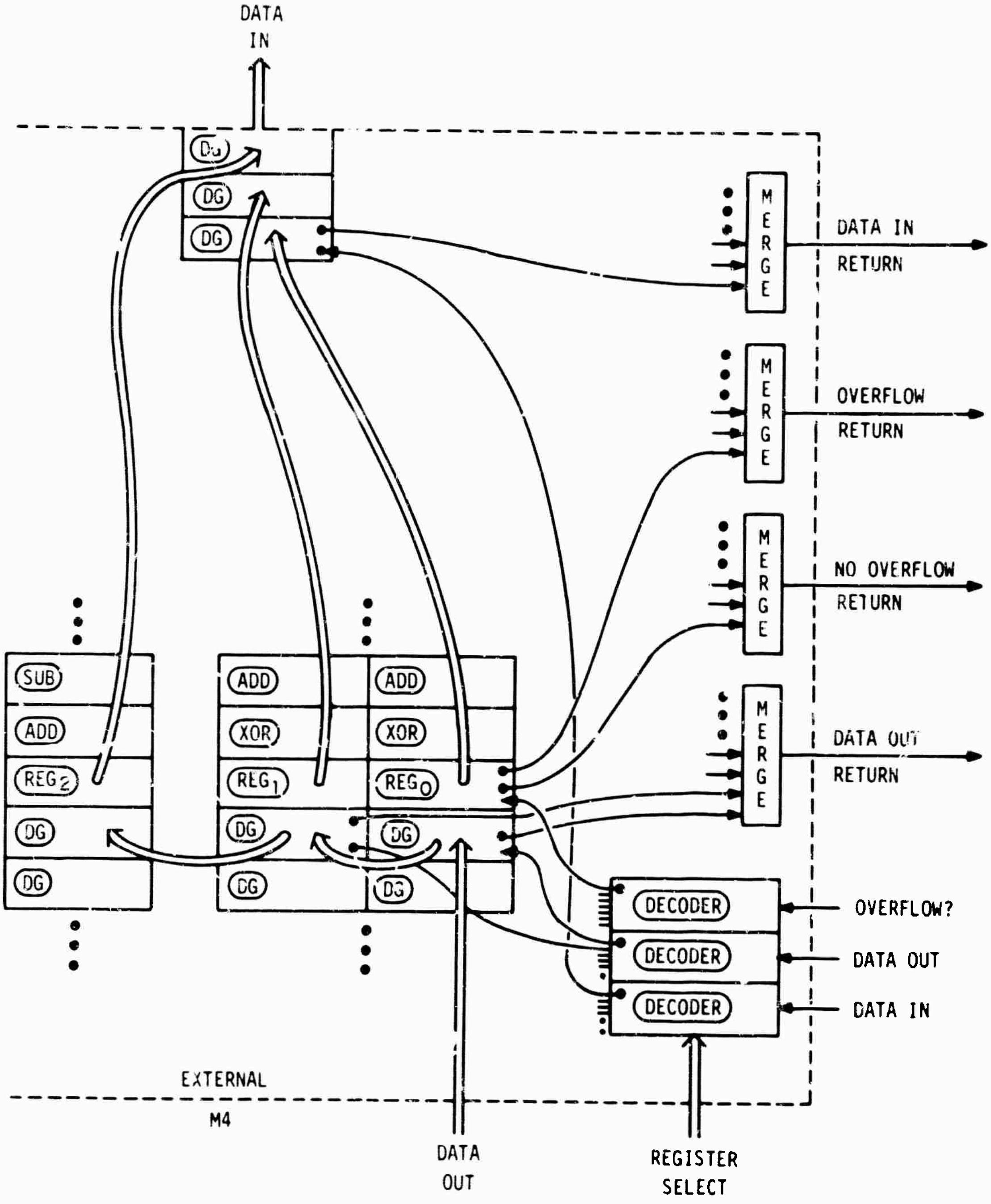
If there is a possibility of concurrent operation between M4 and externally implemented functions, interlocks must be placed at the appropriate places in the register operations. This is left entirely to the user and follows directly the normal macromodular use of the interlock.

4.3 CONTROL GROUP

This Section discusses the conventions required of externally implemented control functions which permit a standardized interface with the M4.

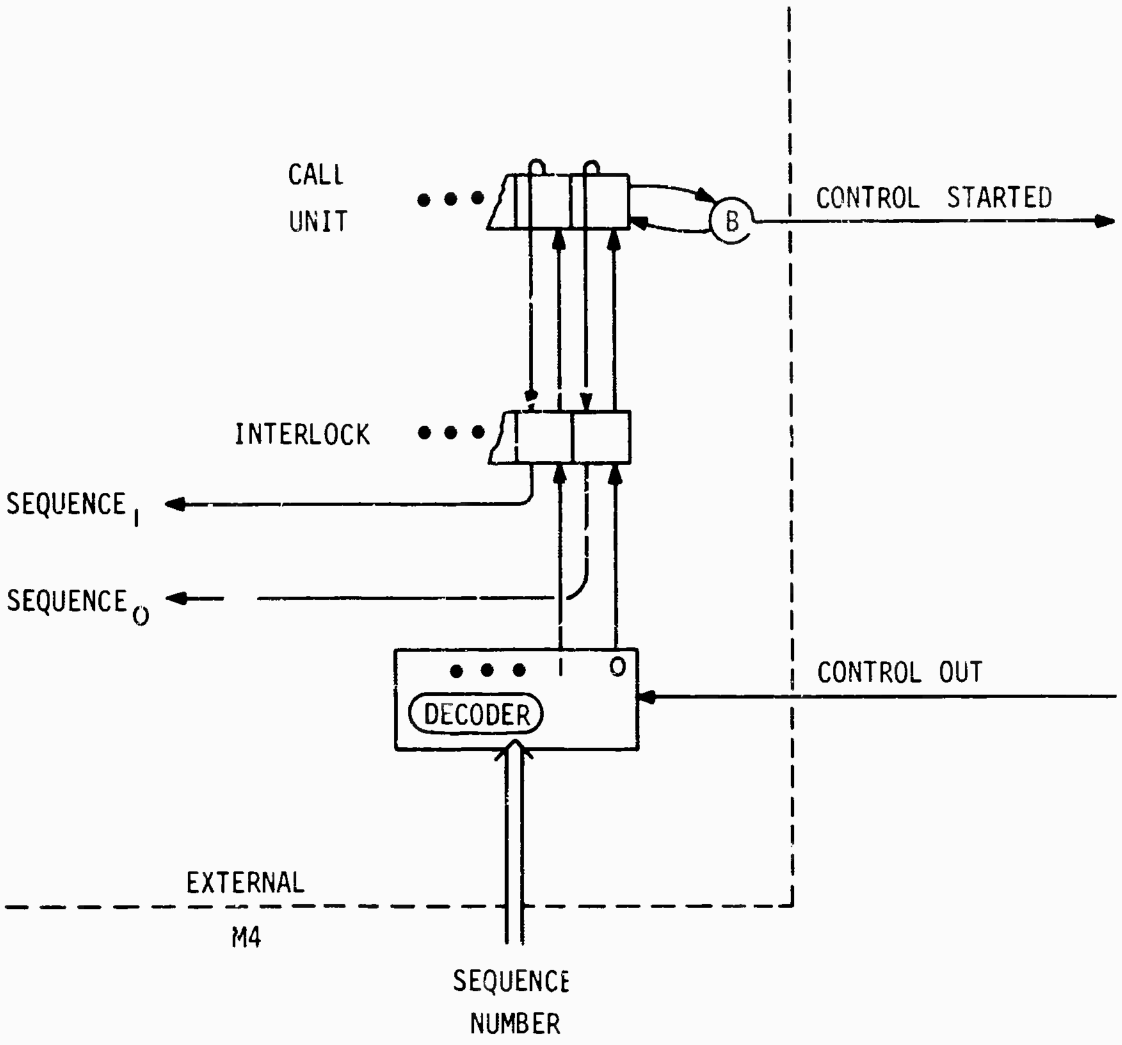
4.3.1 CONTROL OUTPUT FROM THE M4

The M4 presents a data word which represents a control sequence number which must be decoded by external decoders. Figure 9 indicates the necessary operations. A requirement of external control is that it be capable of concurrent operation with internal functions of the M4. Therefore after a control



EXTERNAL REGISTER GROUP

Figure 8



EXTERNAL CONTROL OUTPUT FROM M4

Figure 9

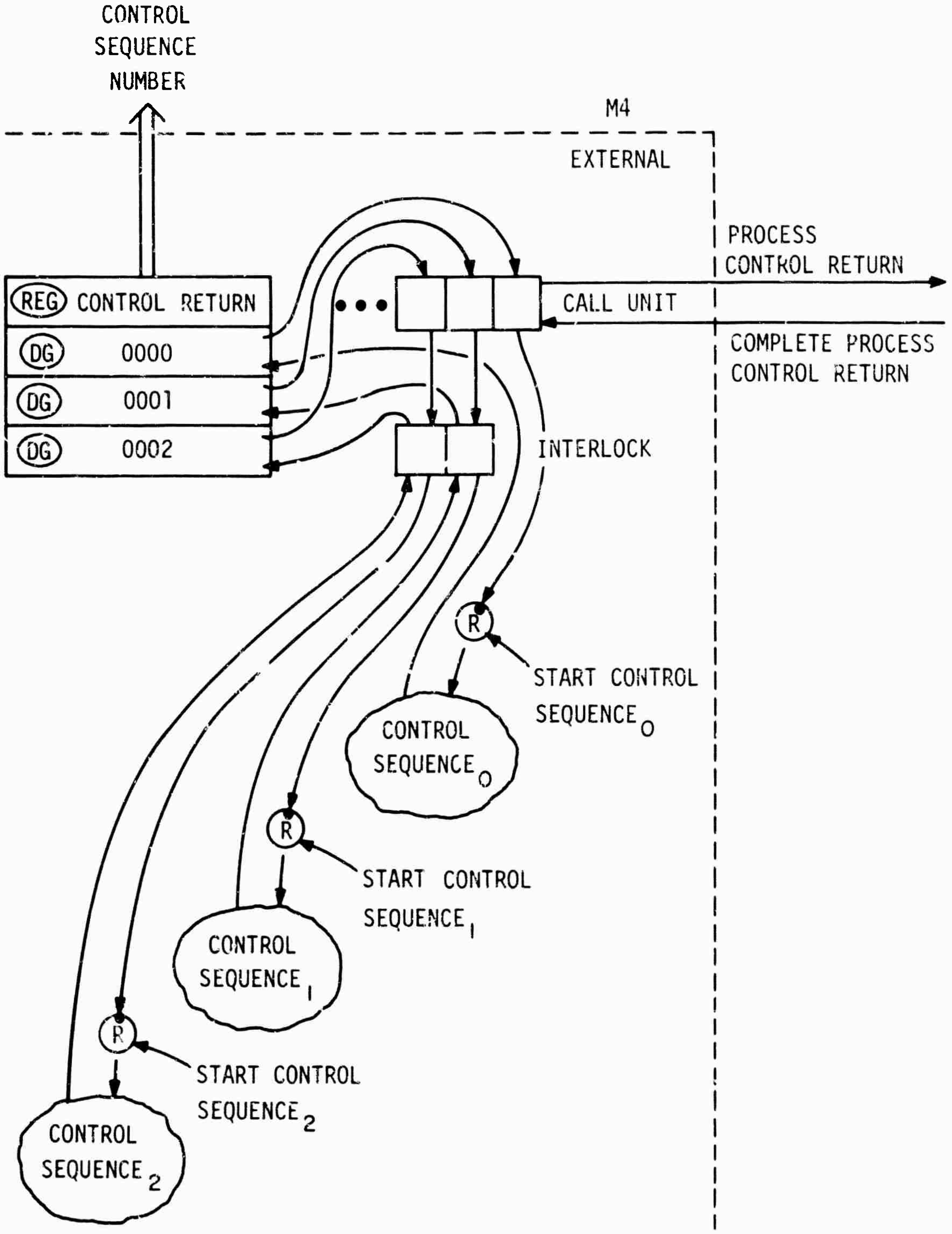
sequence has been decoded, the control signal must branch to indicate to the M4 that internal operations may be resumed. The interlock is required because there is a possibility that the M4 might be able to generate a control signal on another input to the call element before it has been cleared by the return signal. This possibility is very slight because the M4 must do a memory operation before another control signal could be generated. There is a considerable responsibility placed on the operation of the M4 in order to insure the proper operation of these concurrent sequences.

4.3.2 CONTROL RETURN TO THE M4

Figure 10 shows the use of control returns. Returning control must gate a number representing the control sequence into a register. Then control must pass to the M4 to process the return. The completion signal from the processing operation must finally rendezvous with the start of the particular control sequence because the M4 could immediately reactivate the same external control sequence. Failure to do this may mean that two control signals could be active in a single control sequence; an illegal operation in macromodules. The dot on an input to a rendezvous element is a standard notation to indicate the input is preset to an active state. An interlock is shown between control sequences 1 and 2 indicating that they may be concurrently active. Once again we see that he M4 is responsible for a considerable amount of control information.

4.4 IMPLEMENTATION CONSIDERATIONS

The implementation of external control and register operations has been shown as a purely macromodular implementation. This assumes a proper macromodular interface with the M4. In the non-macromodular implementations of the M4, it may be important to reduce the number of interface points between the M4 and externally implemented macromodular functions. A possible technique is to use a memory to centralize the actual interface connections.[12] The resulting changes in the designs shown here are not discussed but it is still possible to use these general ideas.



EXTERNAL CONTROL RETURN TO M4

FIGURE 10

5. CONCLUSION

This report has presented a unified approach to the design and use of macromodular computer systems. Central to the concept is a meta machine which is capable of routing control and data transfer operations as specified by a description of a macromodular system and also provides a standard interface for externally implemented macromodular functions. Several implementations of the meta macromodule machine *M4* have been discussed. The M4 is also capable of providing general programming support for a macromodular system.

It is felt that the unified approach is a very valuable concept which permits an evolutionary approach to computer design by allowing small incremental changes to be made and evaluated before they become a fixed part of the system. Even while these changes are being made, the user always has an easily recoverable working system. The unified approach appears to be a very useful means for providing a convenient input form for functional simulation of digital systems.

For the user who is not oriented toward computer hardware, the unified approach permits the use of the concept of designing a special machine which may, but does not have to, be built. If the user chooses not to build the machine, he still is able to do a very efficient interpretation of his machine using the meta machine. If he does decide to build the computer, he is not required to build all of it. The unified approach gives the software oriented user a fixed target for compiler development by providing the capability of having a system on which to run the compiler and its results.

Finally, and probably the most important, the unified approach permits the best hardware-software trade-offs to be made. Note that many parts of the solution can be expressed in a high level language if that is convenient. It is important to note that the hardware-software balance can always be changed as experience with a system grows.

These conclusions represent the contribution of the unified approach to the development and use of macromodules. Although only the relationship with macromodules has been discussed in this report, it seems that the unified approach can be applied to other restructurable computer systems. It is after all, primarily a way of thinking about a problem and as such has great generality.

APPENDIX 6.1

A UNIFORM AND CONSISTENT NOTATION FOR DESCRIBING A MACROMODULAR SYSTEM

6.1.1 INTRODUCTION

In this section a uniform and consistent notation is discussed which may be used to describe a macromodular system design. Besides being capable of completely describing the system, the notation is convenient to use. A macromodular system is completely described by considering two separate aspects. The exact definition of the meaning, or semantics, of operation is determined by the physical placement of the modules. This placement indicates what data may be added together, what data transfer operations may take place, and all the other considerations of meaning. Second, the order in which these operations are carried out, or syntax is definedby the routing of control cables which link together in specified sequences the various operations. Notice that neither one of these alone completely describes a system and both have distinct functions. The method of notation takes advantage of these considerations.

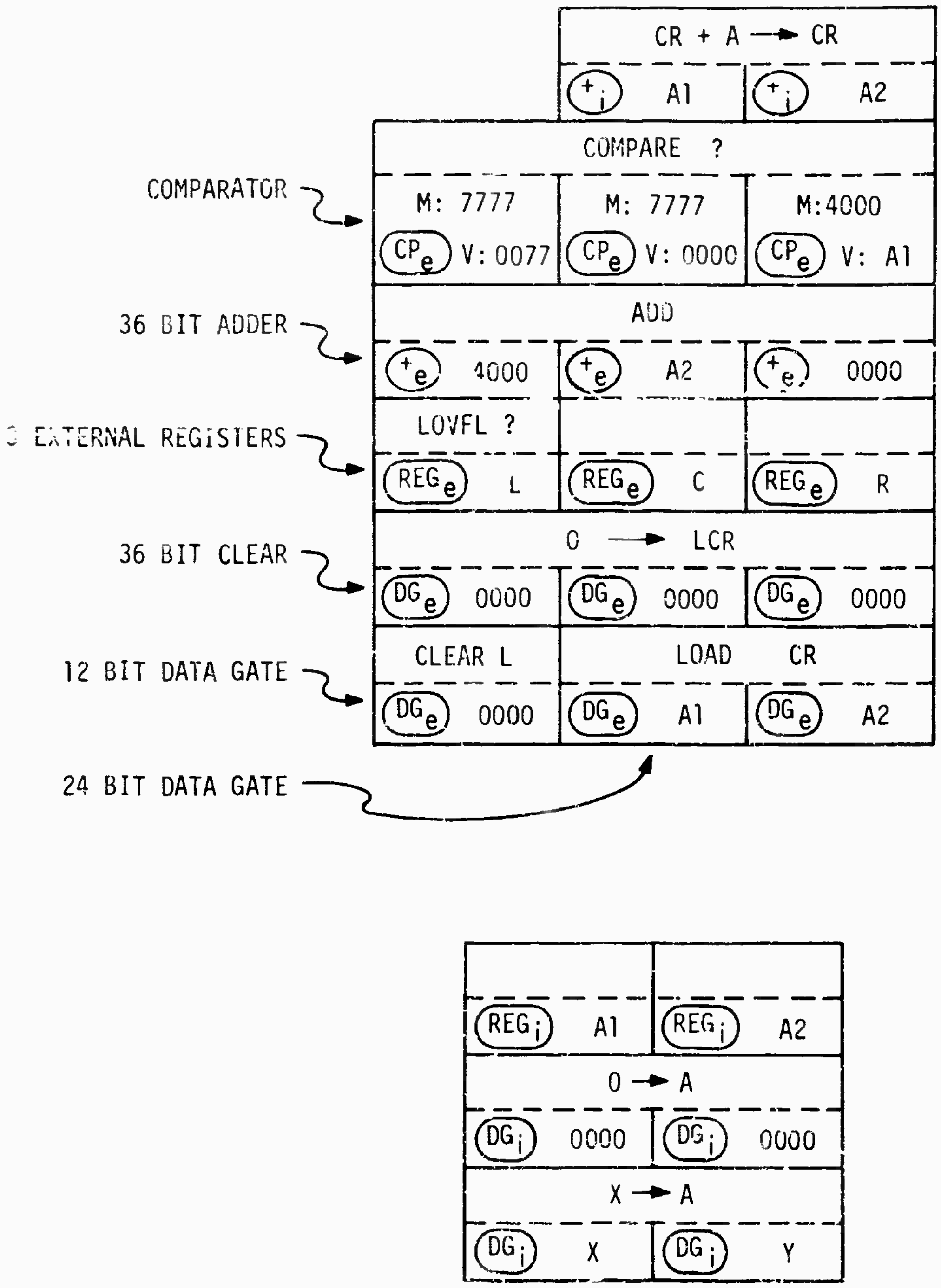
6.1.2 THE DATA PROCESSING DESCRIPTION

The data processing operations, or semantics, are described by a pictorial diagram which corresponds roughly to the actual physical placement of modules. Because the width, in terms of multiples of the basic macromodules, is quite flexible, the actual width of operations is shown by drawing short vertical lines between modules to indicate operations of more than one module length. The area above the dotted line is used to give a name to the operation which then appears in the control sequence descriptions. Additional names may be indicated as alternatives within this area of any single module. This arbitrary name extends for the full width of the defined operation. The module type appears in a circle in the lower left corner of each box. Module type information may have an e or i subscript to denote an externally or internally implemented function. These designations are subject to the rules on implementation. For example, it is not possible to have an internal register with an external adder on it. The area below the dotted line is used to describe data cable inputs, the name of a register, or other input information. Note that a line is used for each different cable input so some modules may take up more space in the picture than they actually do. Fixed parameters may be written in octal in place of cable input names. Figure 6.1.1 shows a typical description.

6.1.3 CONTROL DESCRIPTION

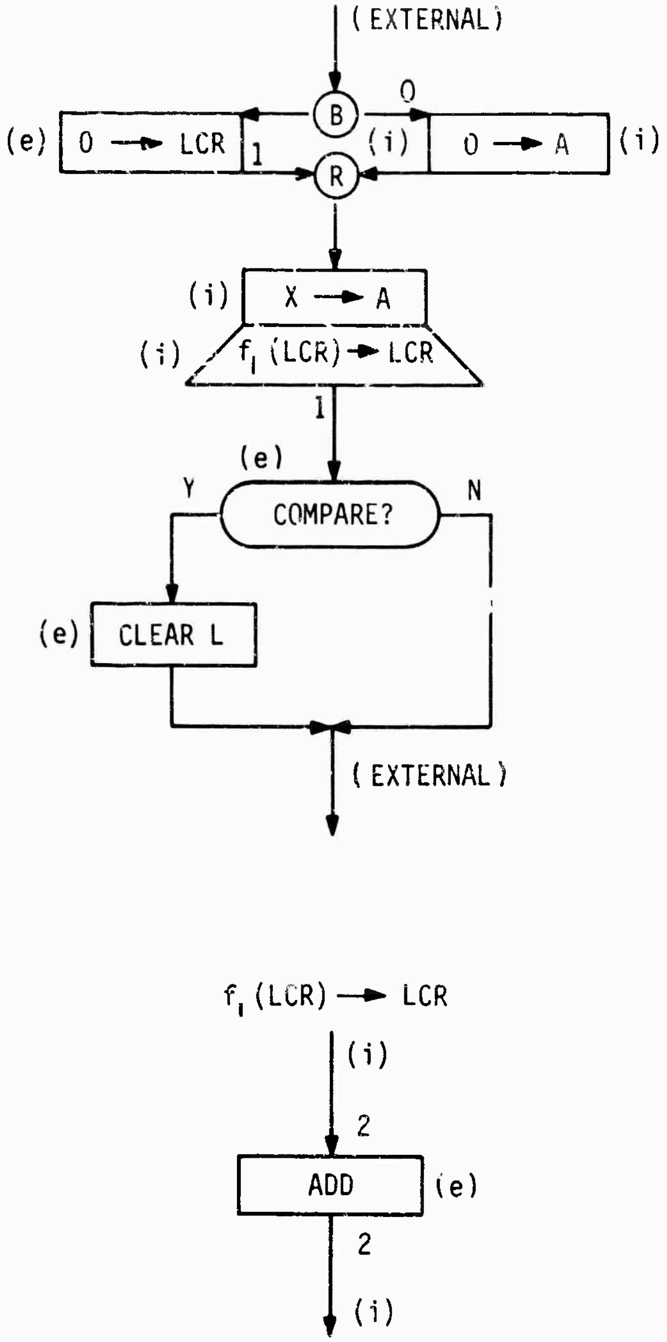
The description of the control sequences, or syntax, consists of flowcharts. Each entry consists of the name of an operation which has been defined in the data processing description. Control branch and rendezvous units are shown as small circles, operations as rectangular boxes, decisions as ovals, and subroutine calls as trapezoids. Figure 6.1.2 shows a typical flow chart for the operations defined in the previous section.

Notice that the flowchart contains annotation to describe external and internal functions. In some sense this is redundant information because these attributes are defined by the data processing module drawings. However, the small numbers on the connecting arcs are necessary and are used to assign numbers to control exit and return lines.



DATA PROCESSING DESCRIPTION

Figure 6.1.1



CONTROL DESCRIPTION

Figure 6.1.2

6.1.4 CONCLUSION

The above notation has been used by the author and has been found to be very convenient. Even though the notation is complete, it is not felt that the required detail is a drawback. Because the designer is committing actual expensive hardware, he must know exactly which operations are currently available. The number of flowcharts required is not large because a system with a few hundred operations in it represents a large system, while the same number of machine language instructions would represent only a very small program. One flowchart entry may represent several actual macromodules because of multiple width operations.

The ability to name the operation in the semantic description and then use it in the control flowcharts make possible flowcharts which are completely annotated yet are directly related to the definition of the operations. This is an extremely important aspect of the descriptive notation presented here. In the future it may be possible to name groups of operations and therefore introduce a macro facility into the notation.

After using this descriptive notation for a while, one is extremely unhapp to be required to render the description almost completely unintelligible in order to transform it to a machine readable torm. A concerted effort must be taken to devise a method of introducing the descriptive notation into a computer in a more direct form.

APPENDIX 6.2

CALCULATION OF THE RATIO OF SIMULATION TIME COMPARED TO ACTUAL OPERATION TIME

6.2.1 INTRODUCTION

In this appendix an approximate ratio is derived for the time required for simulation of macromodular operations compared to the time required for the actual macromodular operations. Because of the asynchronous operation of macromodules these calculations yield only an order of magnitude type value for this comparison. The calculations indicate that an order of magnitude of 100 to 1 is a reasonable value for a machine which is ideally suited for simulation of macromodules. Of course, for simulation on a typical digital computer the ratio is likely to be much higher.

6.2.2 THE CALCULATION

6.2.2.1 THE PROBLEM

As a typical macromodular operation, take a register to register transfer operation operating on 36-bit registers. This corresponds to three register macromodules connected to form one larger register. The active data gate on the destination register is the fourth one from the register in the stack of data gates. Finally, it is assumed that there are five data processing modules above the register. These details are important because operation times are dependent on these factors. Although formulas can be derived for timing operations, this is not an appropriate place for presenting this information.[13] The time for this operation in the actual macromodules is 250 nanoseconds.[14] Experienced intuition is the only guide for describing this as a typical operation.

6.2.2.2 THE SIMULATION MACHINE

The machine which will be assumed in this calculation is ideally suited for simulation of macromodules in that it follows the specification of the meta macromodule machine. This means that the machine can simulate the register transfer operation with only one instruction. However, because a basic 12 bit machine is postulated, several memory references are required to perform the operation. Our calculations therefore involve counting only memory references. One memory reference is required to access the operation code. A memory reference is required to get the addresses of the memory locations for the three source and three destination registers. Finally three words of data must be read from memory and stored into three other memory locations. Therefore, thirteen memory references are required to perform the operation described in section 6.2.2.1. If a memory cycle time of two microseconds is used, the total time for simulation of this operation is 26 microseconds. This is 100 times the 250 nanoseconds required for the actual operation. If a one microsecond memory is available, the ratio becomes 50 to 1.

6.2.3 CONCLUSION

It has been shown that a ratio of 100 to 1 for functional simulation time compared to actual operation time is realistic. The reader should be cautioned that this ratio holds only for the particular, but realistic, situations described here. If functional simulation is to be a realistic design aid ratios of this order are required.

Because this ratio can easily grow by decimal orders of magnitude, it is an extremely important factor for simulators to be aware of. No matter how convenient a particular simulation tool may be, if a user requires several hours of computer time to simulate a few seconds of real time, the tool is not very useful.

APPENDIX 6.3

AN EXAMPLE

6.3.1 INTRODUCTION

The example, which is part of an actual investigation, concerns the requirement for translating spherical to rectangular coordinates. The following equations define the translation:

$$x = \rho \sin \phi \cos \theta \tag{1}$$

$$y = \rho \sin \phi \sin \theta \tag{2}$$

$$z = \rho \cos \phi \tag{3}$$

The existence of a compiler for a FORTRAN-like language to the M4 operations is assumed. The example will show the evolution of the coordinate translation from a completely internal version to one implemented entirely in macromodules. The routine finds the input arguments in three variables called ρ, ϕ, and θ. The output of the routine is left in three variables called x, y, and z.

6.3.2 INITIAL FORM

Initially the coordinate translation is to be specified as a program in a higher level language.

```
phi = phi * 3.14/180;             }  translate degree arguments
theta = theta * 3.14/180;         }  into radians
x = rho * sin (phi) * cos (theta);  }
y = rho * sin (phi) * sin (theta);  }    compute x, y, z
z = rho * cos (phi);                }
```

The language is almost FORTRAN and the execution time for single precision,fixed point results is estimated to be approximately 500 microseconds on a machine with a two microsecond memory cycle time. The sine and cosine are computed by the standard, built-in trigonometric subroutines.

6.3.3 TABLE LOOK-UP

The next step is the realization that because the angles for this application are physically measured with values known to only ± ½ degree, the trigonometric functions can be performed by simple table look-up. An externally implemented table look-up routine is designed and called by simulated macromodular call elements. Naturally the subroutine could be marked and run in internal implementation for debugging purposes. Figure 6.3.1 shows the data processing module layout including those modules necessary for communication with the M4. Figure 6.3.2 is the control network flow chart. Note that internal functions are designed which operate on externally implemented registers.

6.3.4 IMPLEMENT TABLE LOOK UP CALL EXTERNALLY

The next step might be to implement the subroutine call externally. This is a reasonable step because the internal execution of the call and return takes about the same length of time as the entire

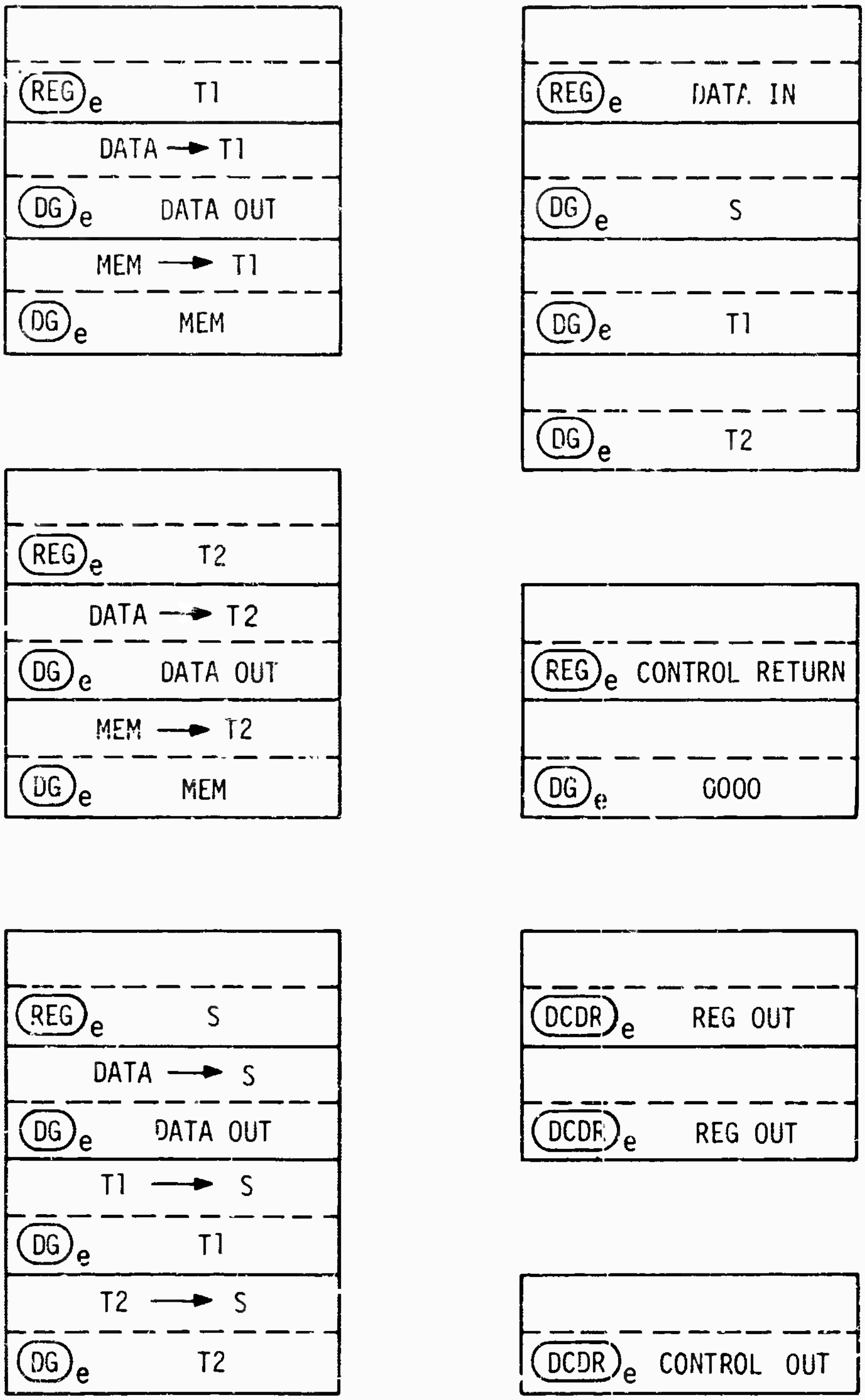
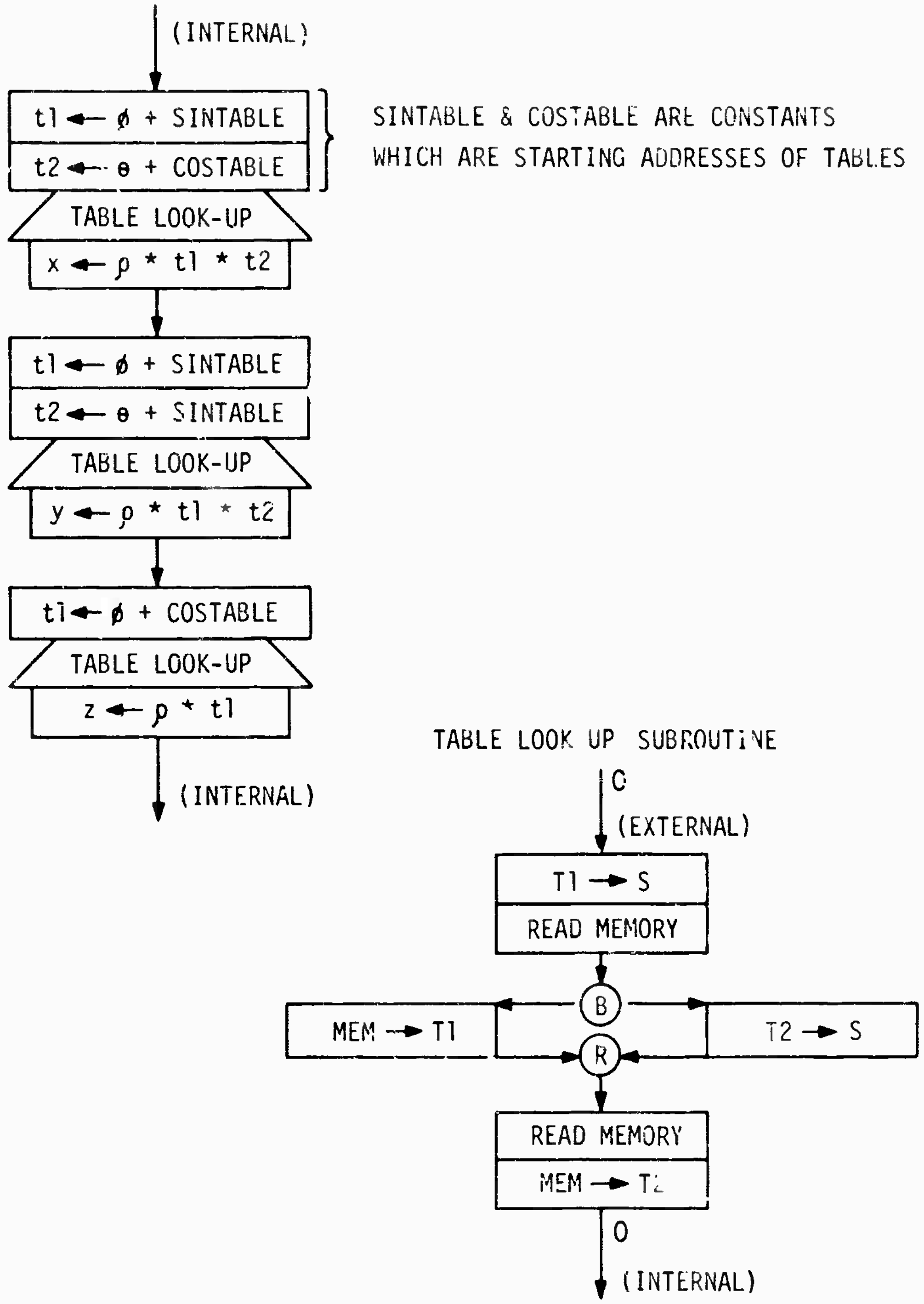


TABLE LOOK-UP - DATA PROCESSING MODULES

Figure 6.3.1



SINTABLE & COSTABLE ARE CONSTANTS WHICH ARE STARTING ADDRESSES OF TABLES

TABLE LOOK-UP - CONTROL

Figure 6.3.2

table look-up operation. Because the subroutine is called from three places, three control lines in each direction between the M4 and externally implemented functions are required. The data processing structure is not shown because it is almost identical to Figure 6.3.1. The control neiwork flow chart is shown in Figure 6.3.3.

6.3.5 IMPLEMENT PARALLELISM WITH MOST CALCULATIONS INTERNAL

The next step is a redesign based on the fac' 'hat it is possible to use some concurrent processing. It is possible to compute a sine or cosine function at the same time that a multiplication is being done. Also, there is a common subexpression in the equations for x and y which only needs to be computed once. Finally, at this time ve do not wish to perform the multiplication in macromodules but do, however want to shift the focus so that the coordinate translation is essentially controlled by the externally implemented functions. Again we note that any of the macromodular functions may be internally implemented by the M4. Figure 6.3.4 shows the data processing modules required, excluding most of those used for communication with the M4. Figure 6.3.5 gives the control network flowchart.

6.3.6 ALL OPERATIONS IMPLEMENTED EXTERNALLY WITH MACROMODULES

Finally, the entire coordinate translation is implemented externally with macromodules. The execution time for this implementation is approximately 20 microseconds, a 25 to 1 improvement in speed over the original implementation. For this description a multiply macromodule is postulated even though it is not part of the initial set of modules. Obviously, the multiply function can be implemente in terms of the original set of modules.

The completely external implementation is shown in Figures 6.3.6 and 6.3.7.

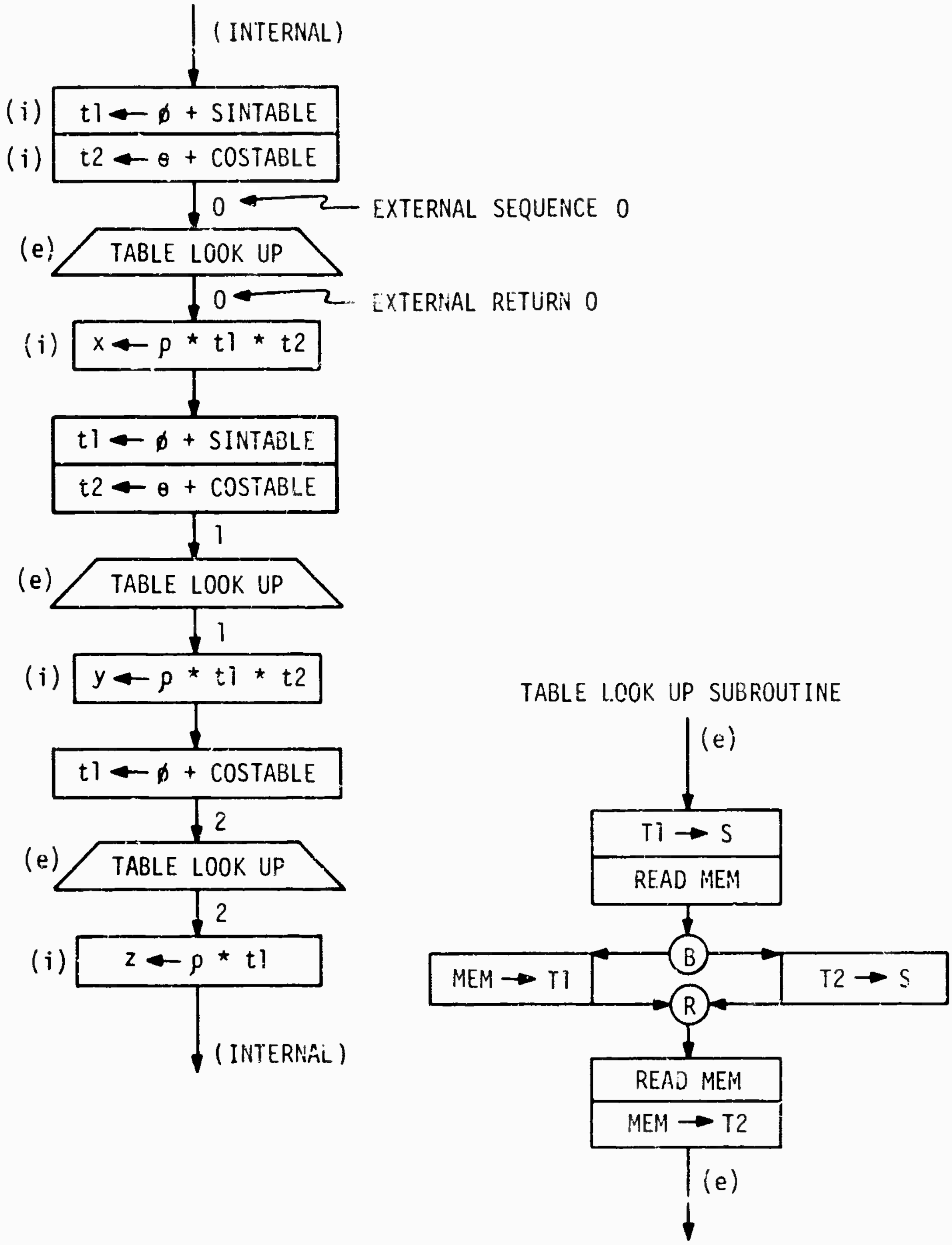
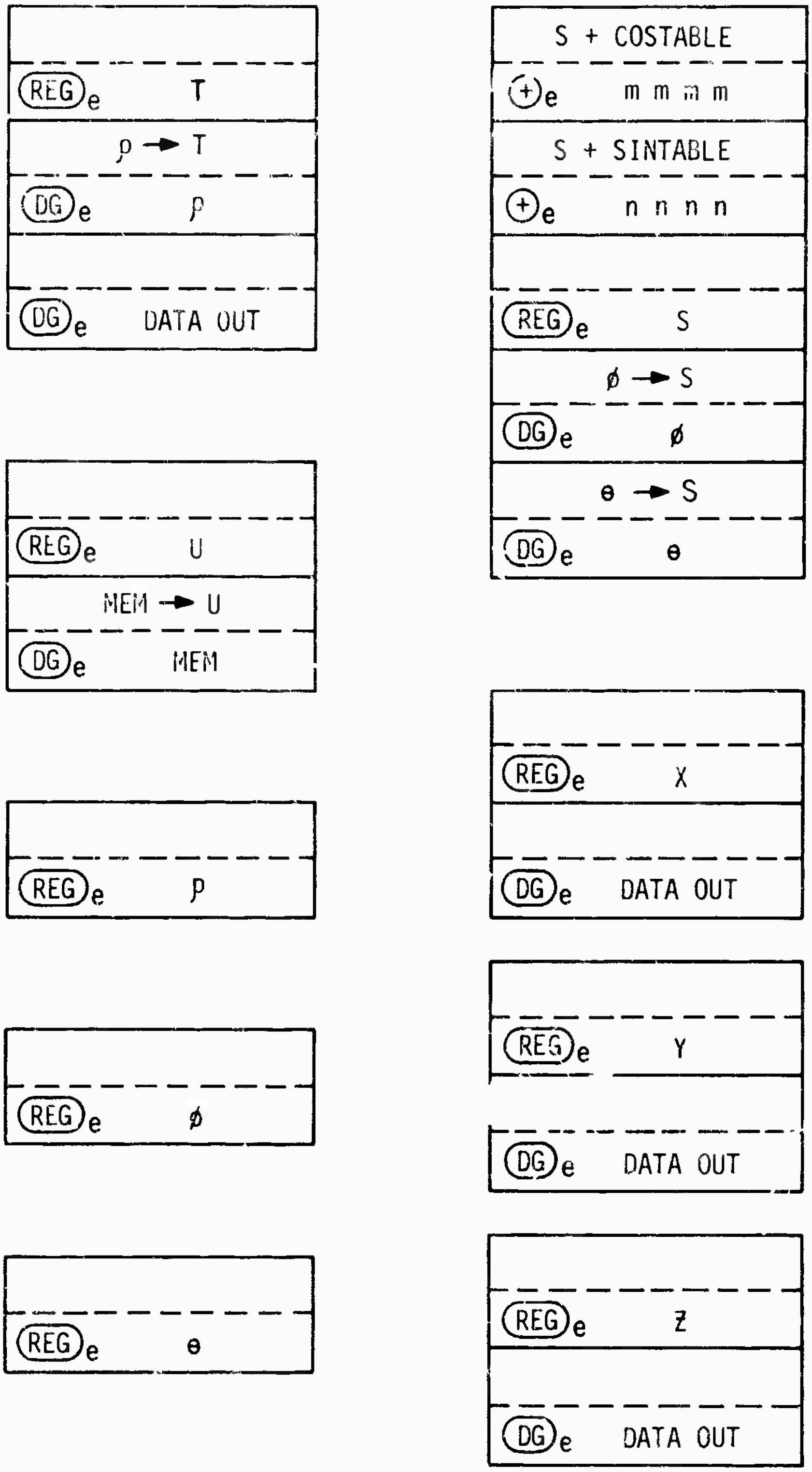


TABLE LOOK-UP - EXTERNAL CALL - CONTROL

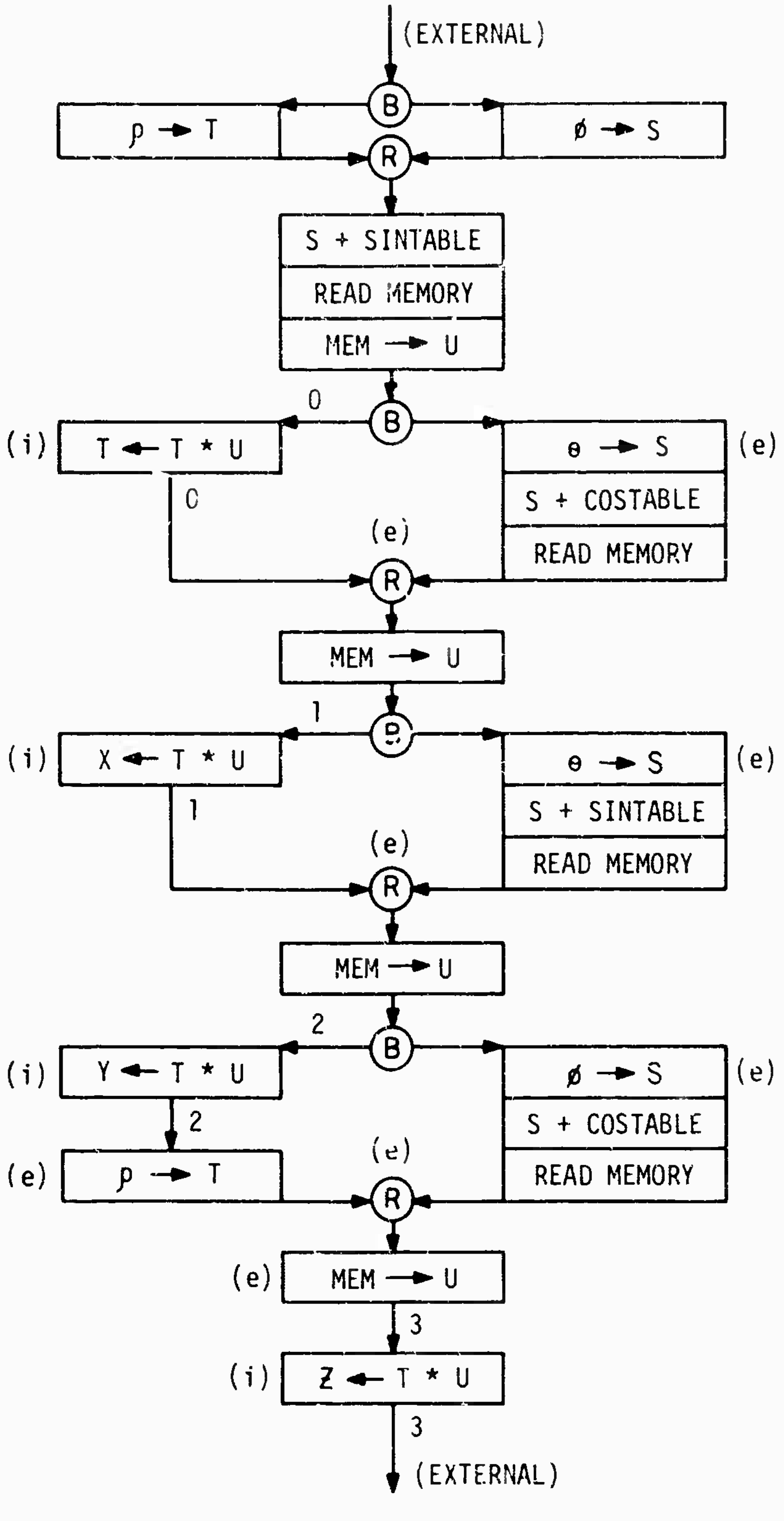
Figure 6.3.3



IMPLEMENT PARALLELISM MOST CALCULATIONS

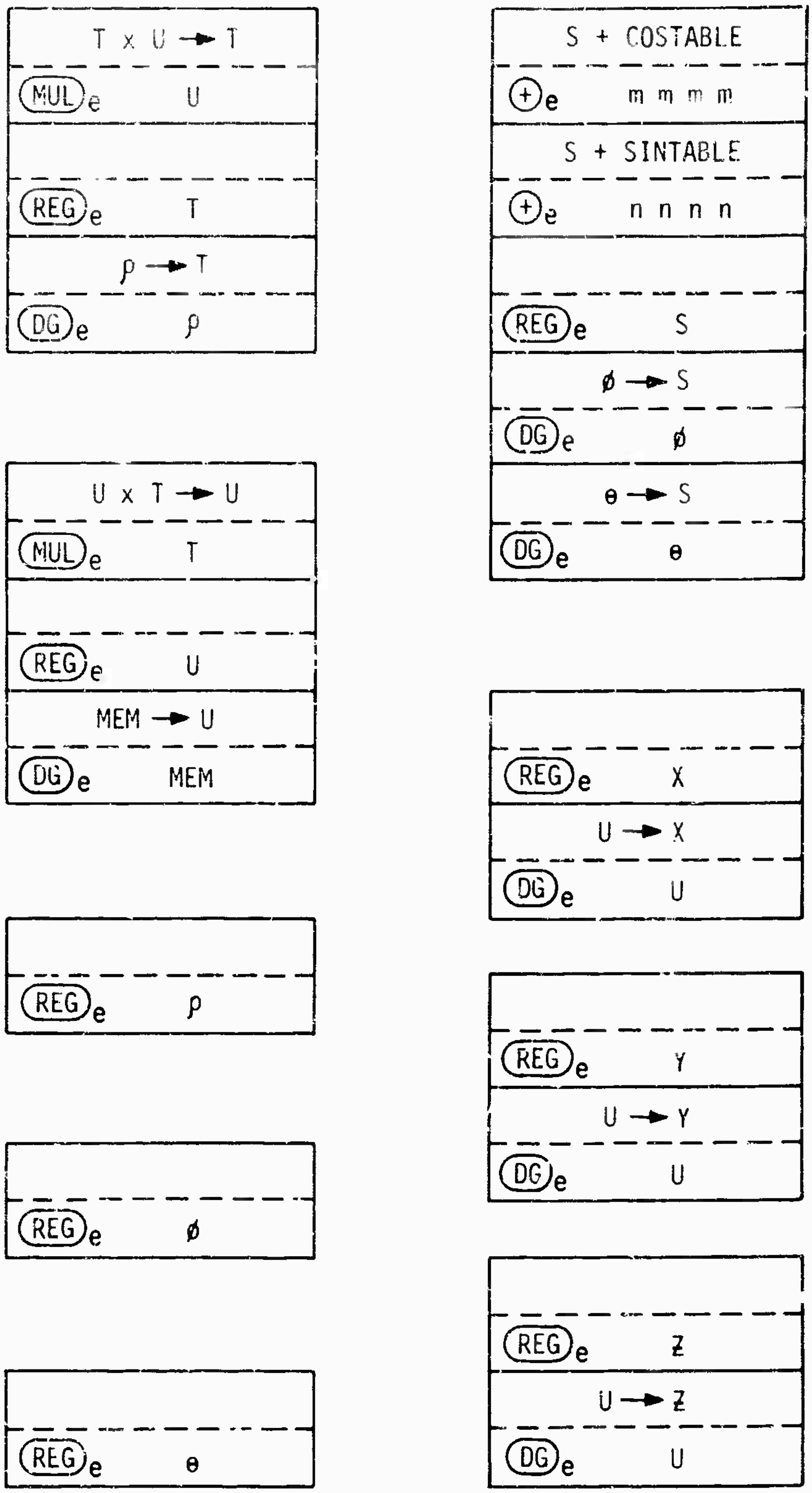
INTERNAL - DATA PROCESSING MODULES

Figure 6.3.4



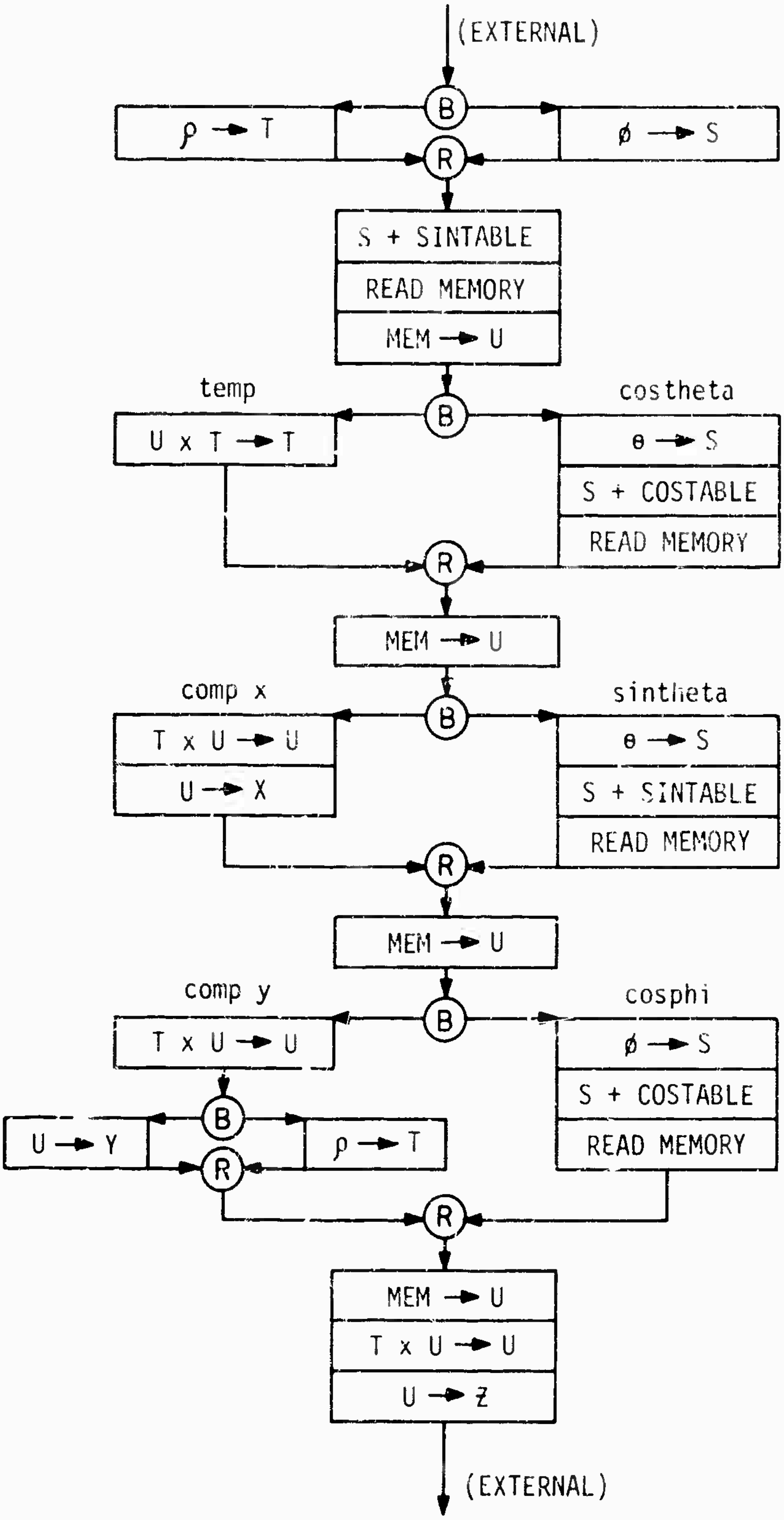
IMPLEMENT PARALLELISM MOST CALCULATIONS INTERNAL - CONTROL

Figure 6.3.5



COMPLETE EXTERNAL IMPLEMENTATION - DATA PROCESSING MODULES

Figure 6.3.6



COMPLETE EXTERNAL IMPLEMENTATION - CONTROL

Figure 6.3.7

APPENDIX 6.4

A MACROMODULAR META MACROMODULE MACHINE (M6)

6.4.1 INTRODUCTION

This appendix summarizes the design of an M6 which has been referred to in the body of the report. The important details of the M6 communication with external registers and control were presented in the body of the report and are not repeated here.

There is one primary register, which is 36 bits long, where all the macromodular data processing operations take place. Because of the great flexibility with which data cables may be connected, all of the simulated data cable inputs to a function must be read from actual or simulated registers before the operation starts. The M6 uses first-in first-out buffer stacks for the purpose of holding these operands.

6.4.2 PROGRAM AND INSTRUCTION FORMAT

The program and instruction format of M6 is analogous to a stored program digital computer. In general commands are stored in memory which describe macromodular operations and are executed sequentially except for decision-making operations. Along with each command is stored information regarding the simulated registers which are to be used in the command. The operation sequence can be broken by unconditional or conditional transfers to other sequences of operations. In this section we assume that the M6 has a 4096 word x 12 bit memory. Expansion of memory capacity can be handled by some type of paging scheme. The M6 really should also be capable of executing instructions to perform I/O and other support functions. These are not discussed further here.

The command is a two word sequence in which the first word is used to specify the macromodular operation and the second word is used to flag options on the command. The only flagged operation currently specified is the very important one of specifying whether or not control is to remain within the meta machine or is to exit to externally implemented macromodular functions. Only one bit is needed to specify this option; however, if control is to exit at this point, the remaining eleven bits are used to specify the external control sequence. This allows 2048 different external control sequences to be specified.

As long as no concurrent operations are implemented, external control generation and return is quite simple. However, if concurrent operations exist and some of them are external and some are internal, the situation becomes more complex. It is required that all externally implemented concurrent operations proceed in a truly concurrent form while internal operations proceed in the required sequential form. A first-in first-out stack is used to remember internal control sequences which have been encountered but not executed. Returning external control signals put the address of where internal operation is to resume into this stack. Thus, as internal rendezvous elements are encountered, the uncompleted sequences are executed but external control may be executed in true concurrent form.

Data references to real or simulated registers consist of a sequence of addresses of register specifications preceded by an integer which specifies the number of basic 12 bit register modules in the reference. There are as many consecutive sets of data references as are called for by a particular macromodular function. For example, data gates and registers require two data references while a comparator requires three. The following forms indicate the use of a data gate operation for both 12 and 24 bit data references:

12-BIT

word n: data gate
n + 1: flag
n + 2: 1
n + 3: source register specification address
n + 4: 1
n + 5: destination register specification address

24-BIT

word n: data gate
n + 1: flag
n + 2: 2
n + 3: source register specification address foi bits 0-11
n + 4: source register specification address for bits 12-23
n + 5: 2
n + 6: destination register specification address for bits 0-11
n + 7: destination register specification address for bits 12-23

Register specifications consist of either a single word or two consecutive words in the M6 memory. The first word contains one bit which specifies whether the register is simulated in memory or exists outside the meta machine implemented as an actual macromodular register. If this is the case, the remaining 11 bits are used to specify a register number. Thus it is possible to access a total of 2048 externally implemented registers. If the register is simulated within memory, the first specification word also contains a bit which is a true reflection, in the macromodular sense, of the overflow condition of the register. The remaining ten bits are available for other, future uses. Finally ,the second word holds the contents of the simulated register.

Control operations in M6 are analogous to those in stored program computers. For non-decision operations control passes to the next operation specified in contiguous memory locations. A location register is used as in a programmed computer. Decision operations interrupt the consecutive flow of control for all possible decision outputs. A transfer of control is specified by a 12 bit address which indicates the start of a new control sequence. This is directly analogous to branch or jump instructions in a stored program computer.

6.4.2 DATA OPERATIONS

The instruction formats of all macromodules which process or monitor data are presented here. Each word of the operation is not specified in detail. Specifically, the command and flag words are assumed to be part of the mnemonic for the operation and the details of the data references are subsumed in the notation: operand$_1$, operand$_2$, etc. Note also that the numerical code for each operation has not been specified. In general, the first operand specifies a cable input, and the second operand specifies the register on which the command operates. The parentheses are used to denote, *the contents of the register specified by*. Thus, (operand$_1$) means that operand$_1$ is a register specification address and the data described by that specification are used in the operation.

DATA GATE

dg, operand$_1$, operand$_2$

$(\text{operand}_1) \rightarrow (\text{operand}_2)$

ADDER

ad, operand$_1$, operand$_2$

$(\text{operand}_1) + (\text{operand}_2) \rightarrow (\text{operand}_2)$

SUBTRACT CABLE

subc, operand$_1$, operand$_2$

$(\text{operand}_2) - (\text{operand}_1) \rightarrow (\text{operand}_2)$

SUBTRACT REGISTER

subr, operand$_1$, operand$_2$

$(\text{operand}_1) - (\text{operand}_2) \rightarrow (\text{operand}_2)$

OR

or, operand$_1$, operand$_2$

$(\text{operand}_1) + (\text{operand}_2) \rightarrow (\text{operand}_2)$

EXCLUSIVE OR

Xor, operand$_1$, operand$_2$

$(\text{operand}_1) \oplus (\text{operand}_2) \rightarrow (\text{operand}_2)$

AND

and, operand$_1$, operand$_2$

$(\text{operand}_1) \cdot (\text{operand}_2) \rightarrow (\text{operand}_2)$

WRITE MEMORY

wrm, address of memory specification, operand$_1$, operand$_2$

The contents of the register specified by operand$_1$ are used as the address at which to store the contents of the register specified by operand$_2$. The memory specification consists of one or two consecutive words which give the address in the memory of the M4 which corresponds to address zero of the simulated memory. Consecutive 12 bit words are used to store the data. The operation does not permit writing into 12 bit fields of a wider memory module, an operation currently under feasibility study by the designers of macromodules.

READ MEMORY

rdm, addr of memory specification, operand$_1$, operand$_2$

The contents of the register specified by operand$_1$ are used as an address whose contents are moved into the register specified by operand$_2$. The other details of the operation are specified in the in the previous paragraph.

SHIFT LEFT

shl, operand$_1$, operand$_2$

(operand$_2$) is shifted left one bit position. The left most bit is lost and the vacated bit position is filled from the left most bit of (operand$_1$).

SHIFT RIGHT

shr, operand$_1$, operand$_2$

(operand$_2$) is shifted right one bit position. The right most bit is lost and the vacated bit position is filled from the right most bit of (operand$_1$).

6.4.4 CONTROL OPERATIONS

This section specifies the macromodular control operation commands.

COMPARATOR

com, address$_1$, address$_2$,. operand$_1$, operand$_2$, operand$_3$

The contents of the register specified by operand$_3$ are compared to the contents of the register specified by operand$_1$ using the contents of the register specified by operand$_2$ as a mask. If the comparison is false, the next command is taken from address$_1$. If the comparison is true, the next command is taken from address$_2$. No data are changed by this operation.

TEST OVERFLOW

tov, operand$_1$, address$_1$, address$_2$

If the overflow indicator of the register specified by operand$_1$ is on, the next command is taken from address$_1$, otherwise the next command is taken from address$_2$.

TRANSFER OF CONTROL

to, address$_1$

Control is unconditionally transferred to the command at address$_1$.

CALL ELEMENT

cle, address$_1$, address$_2$

Control is transferred to the first command of the macromodular subroutine which starts at $address_2$. The command at $address_1$ must be a transfer of control which is used to return control to the place from which the subroutine was called.

DECISION CALL ELEMENT

dce, $address_1$, $address_2$, $address_3$, $address_4$, $address_5$

Control is transferred to the two return subroutine beginning at $address_5$. $Address_1$ is the address of the command which follows the call for one return of the subroutine. $Address_2$ is the address of the transfer of control at the end of the subroutine for this same return. $Address_3$ and $address_4$ have the same functions as $address_1$ and $address_2$ for the other return from the subroutine.

BRANCH UNIT

bru, $address_1$, $address_2$

The branch unit initiates the two concurrent sequences which start at $address_1$ and $address_2$ Externally implemented sequences are initiated immediately while internal sequences are initiated in turn with more than one stored in the control stack.

RENDEZVOUS UNIT

rvu l, rvu r, mark

The rendezvous unit terminates concurrent sequences. There are two inputs, left and right, which must be specified in that order. The mark word is used to indicate whether or not the rendezvous has had both inputs activated. If control has been activated at both inputs, the command immediately following is executed. If only one input has been activated, control must pass to a sequence whose starting address has been stored in the control stack.

DECODER

dec, $operand_1$, $operand_2$, $address_0$, $address_1$,..., $address_7$

The contents of the register specified by $operand_1$ used as a mask which indicates three contiguous bits of the register specified by $operand_2$ to be decoded. Control passes to $address_0$, $address_1$,..., depending on the octal value of the three decoded bits.

INTERLOCK

Not simulated.

6.4.5 EXAMPLES OF THE M6 DESIGN

Although it is difficult to present some of the design of the M6 without presenting the detailed design in its entirety, this section will discuss, in general terms, the design of a few selected areas of the M6.

6.4.5.1 DATA FETCH

An important operation in the M6 is the fetching data which is to be used as a cable input to a simulated operation. The data may consist of any number of register segments up to the capacity of the buffer stacks mentioned in 6.4.1. There is also the requirement that any of the register segments may be implemented externally. The design is shown as a general flowchart in Figure 6.4.1.

6.4.5.2 AND OPERATION

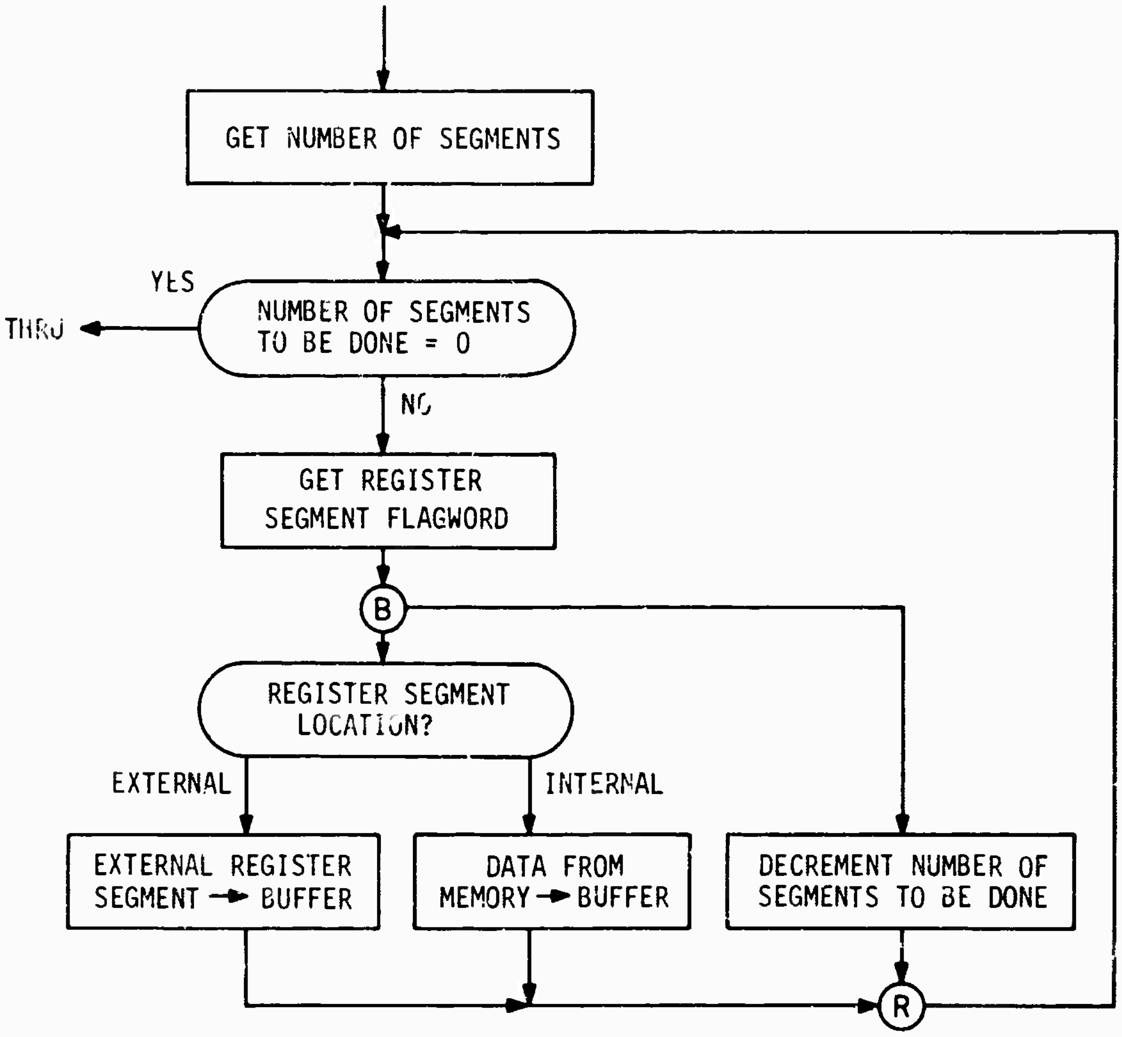
The AND operation represents that class of macromodular data processing functions which can be performed by simple iteration of segments. Each segment is complete by itself and there is no need to know information from any other segment. The design of the AND is presented in Figure 6.4.2. Notice that all segments for the cable input are fetched and stored into a buffer stack. Then each segment for the register operand is fetched, the next cable segment read from the stack, the operation performed, and the result restored to the proper segment. The operations concerned with fetching and restoring the register operand segments contain commands similar to those discussed in 6.4.5.1.

6.4.5.3 ADD OPERATION

The ADD operation represents a class of operations in which each segment is dependent on other segments. In this case, provision must be made to propagate the carry generated in each segment on to the next segment. This is done by having registers to the left and right of the register where the ADD takes place. The carry is generated into the leftmost register which is then moved to the most significant bit position of the rightmost register. The ADD operation then adds zero to the leftmost register, the cable segment to the center register, and a constant with a one in the most significant bit position to the rightmost register. The overflow indicator is cleared for all bit the most significant segment. The design is described by the flowchart in Figure 6.4.3.

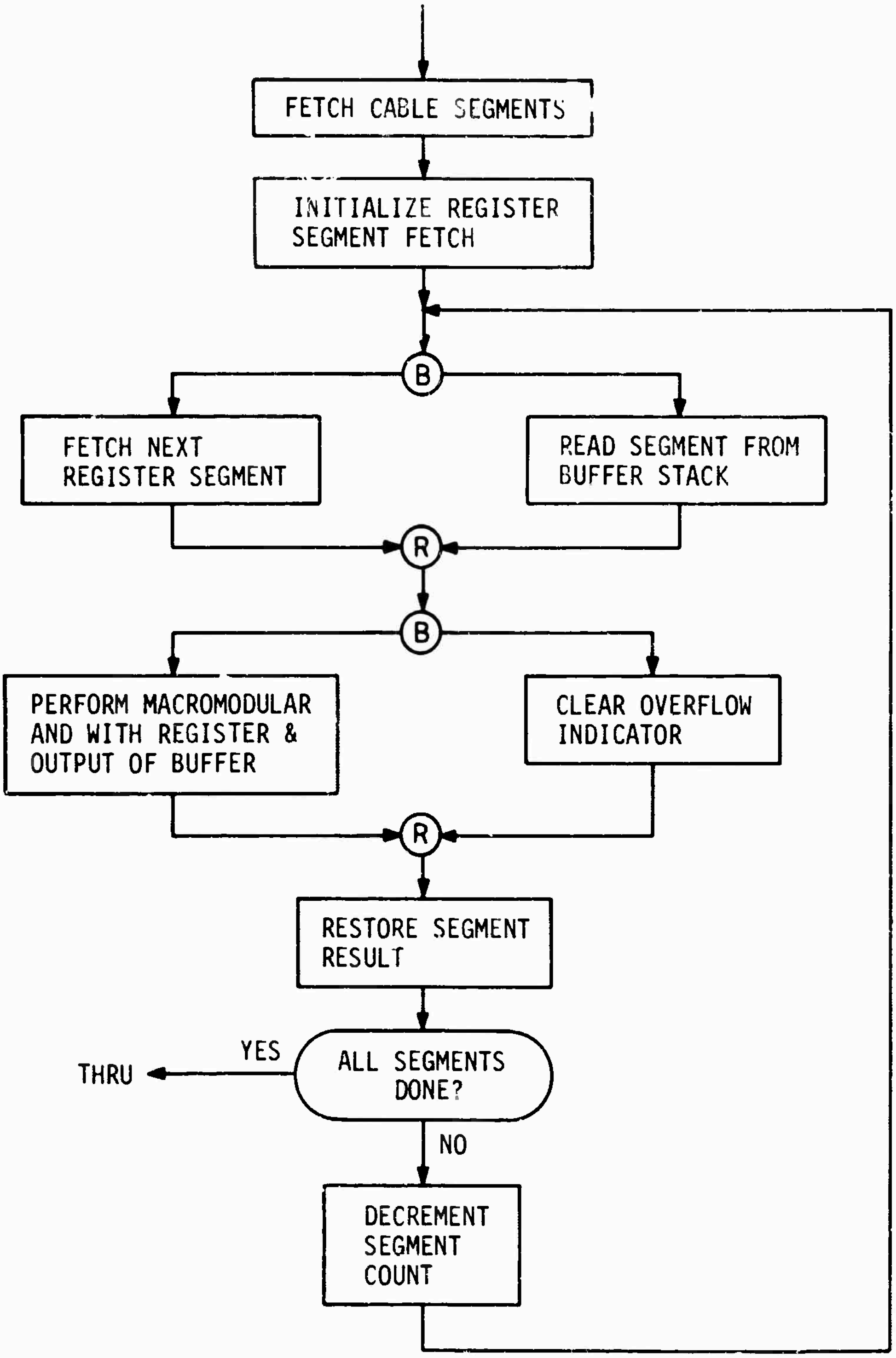
6.4.5 4 COMPARE OPERATION

The COMPARE operation represents those operations which cause a transfer of control. This is simply a matter of moving one of two possible addresses to the P register or location counter. The COMPARE operation requires two cable inputs. Because no data is changed, there is no need to restore any segments. Finally, if the comparison fails on any segments except the last, the remaining segments are not checked but there is some overhead required to clear the previously filled buffer stacks. The design of the COMPARE operation is shown in Figure 6.4.4.


GET NUMBER OF SEGMENTS
YES
THRU
NUMBER OF SEGMENTS TO BE DONE = 0
NO
GET REGISTER SEGMENT FLAGWORD
B
REGISTER SEGMENT LOCATION?
EXTERNAL
INTERNAL
EXTERNAL REGISTER SEGMENT → BUFFER
DATA FROM MEMORY → BUFFER
DECREMENT NUMBER OF SEGMENTS TO BE DONE
R

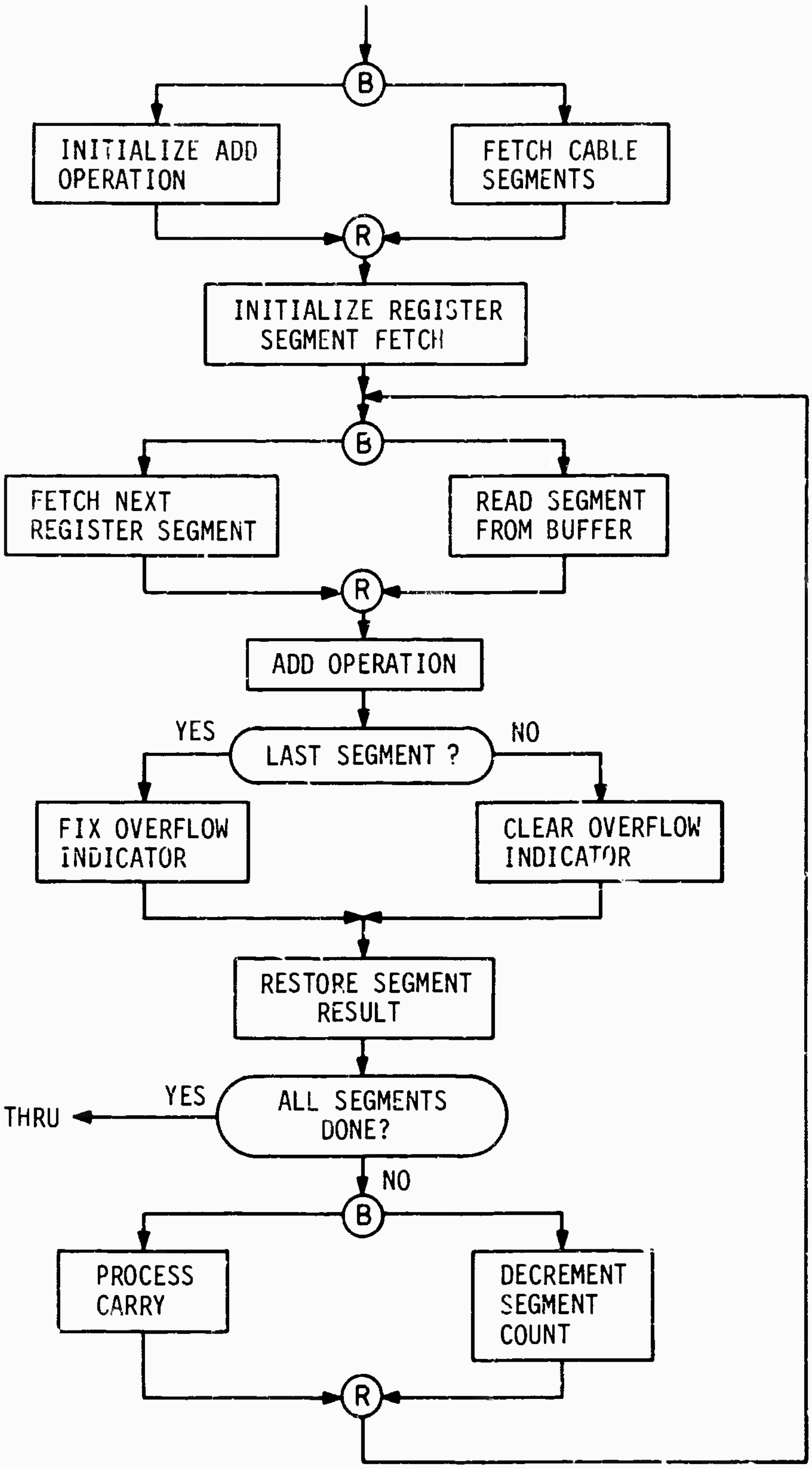
DATA FETCH FOR SIMULATED OPERATION

Figure 6.4.1


FETCH CABLE SEGMENTS
INITIALIZE REGISTER SEGMENT FETCH
B
FETCH NEXT REGISTER SEGMENT
READ SEGMENT FROM BUFFER STACK
R
B
PERFORM MACROMODULAR AND WITH REGISTER & OUTPUT OF BUFFER
CLEAR OVERFLOW INDICATOR
R
RESTORE SEGMENT RESULT
ALL SEGMENTS DONE?
YES
THRU
NO
DECREMENT SEGMENT COUNT

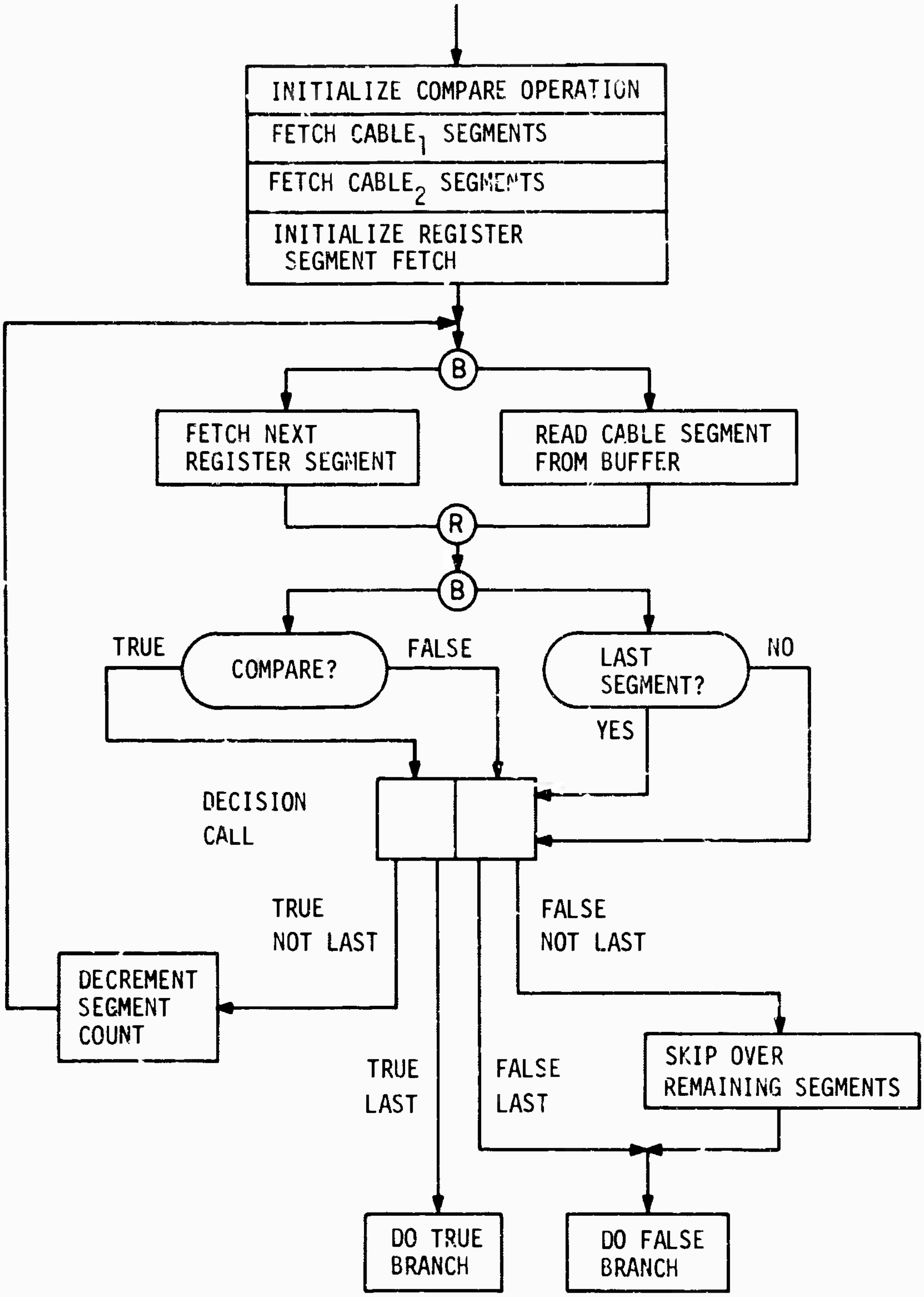
THE AND OPERATION

Figure 6.4.2



THE ADD OPERATION

Figure 6.4.3



THE COMPARE OPERATION

Figure 6.4.4

7. REFERENCES

1. Estrin, G., *Organization of Computer Systems: the Fixed Plus Variable Structure Computer,* Proceedings of the Western Joint Computer Conference, 1960, *33-40.*
2. Clark, W. A., et al, *Macromodular Computer Systems,* Proceedings of the Spring Joint Computer Conference, 1967,*335-401.Reprinted as Technical Report No.4,Computer Systems Laboratory, Washington University, St. Louis, Missouri, June 20,1967.*
3. Cox, J.R., *Economy of Scale and Economy of Specialization,* Submitted for publication.
4. Molnar, C.E., Ornstein, S.M., and Anné, A., *The Chasm: a Macromodular Computer for Analyzing Neuron Models,* Proceedings of the Spring Joint Computer Conference, 1967, *393-401.*
5. Ball, W.E., *A Macromodular Meta Machine,* Proceedings of the Spring Joint Computer Conference, 1967. *377-392.*
6. Rawizza, A.R., *Valgol II Machine,* M.Sc. Thesis, Washington University, St. Louis, Missouri, June, 1967.
7. Molnar, C.E., *A Macromodular Fourier Transform Computer,*Technical Memorandum No.38, Computer Systems Laboratory, Washington University, St. Louis, Missouri, August 8,1967.
8. Frankford,C., and Ellis,R.A., *Macromodular Simulation on the LINC,* Technical Memorandum No.1, Computer Systems Laboratory, Washington University, St. Louis, Missouri, August 24, 1966.
9. Dammkoehler, R.A . *A Macromodular Systems Simulator (MS2),* Proceedings of the Spring Joint Computer Conference, 1967. *371-376.*
10. Kitch, D., and Keller,R., *A Macromodular Programming Language (MACPL),* Technical Memorandum No. 43, Computer Systems Laboratory, Washington University, St. Louis, Missouri, October,1967.
11. Ellis, R.A., *A Macromodular Meta Macromodule Machine (M6),* Technical Memorandum No. 47, Computer Systems Laboratory, Washington University, St. Louis, Missouri, November, 1967.
12. Couranz, G.R., *A Proposed LINC-Macromodule Interface,* Technical Memorandum No.33, Computer Systems Laboratory, Washington University, St. Louis, Missouri, August 15, 1967.
13. Stucki, M.J., *Timing Study: Transfer Logic of Modules Containing the Up-Bus,* Technical Memorandum No. 44, Computer Systems Laboratory, St. Louis, Missouri, October 24, 1967.
14. Stucki, M.J., Personal Communication.

BLANK PAGE

Unclassified
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Computer Systems Laboratory Washington University St. Louis, Missouri	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE

A Unified Approach to the Design and Use of Restructurable Computer Systems: The Meta Macromodule Machine

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Interim

5. AUTHOR(S) *(First name, middle initial, last name)*

Robert A. Ellis

6. REPORT DATE January, 1968	7a. TOTAL NO. OF PAGES 49	7b. NO. OF REFS 14
8a. CONTRACT OR GRANT NO. (1) DOD(ARPA) Contr act SD-302 (2) NIH(DRFR) Grant No. 00218 b. PROJECT NO. (1) ARPA Project Code No. 5880 c. Order No. 655 d.	9a. ORIGINATOR'S REPORT NUMBER(S) Technical Report No. 7 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*	

10. DISTRIBUTION STATEMENT

Distribution of this document is unlimited

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY ARPA-Information Processing Techniques, Washington, D.C. N.I.H., Div. of Research

13. ABSTRACT

A restructurable computer system offers the user an evolutionary approach to the design and use of computer systems. To support this, a unified approach is proposed in this report. A meta machine and its environment are described which provide the ability to treat a macromodular description of a system as a program to be executed or as aaset of specifications from which the system may be directly implemented in macromodules.

Unclassified
Security Classification

Unclassified

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Restructurable Computer Systems						
Macromodular Computer Systems						
Functional Simulation						
Operating Environment						
Use of Restructurable Computer Systems						
Evolutionary Design						
Uniform Hardware–Software Notation						