CANADIAN THESES ON MICROFICHE

THÈSES CANADIENNES SUR MICROFICHE

National Library of Canada
Collections Development Branch

Canadian Theses on
Microfiche Service

Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Direction du développement des collections

Service des thèses canadiennes
sur microfiche

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

Canada

NL 339 (r. 85/09)

THE INTEGRATION OF CONTROL STRUCTURES IN MACHINE ARCHITECTURES

by

Luc A. Lepine

A thesis presented to the School of Graduate Studies and Research
of the University of Ottawa
in partial fulfillment of the requirements for the degree
of Master in Computer Science
in the Department of Computer Science

© Luc A. Lepine, Ottawa, Canada, 1984.

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

ACKNOWLEDGEMENTS

I wish to thank Peter Hickey for the technical assistance he provided, Dr. Robert Probert, Dr. Stan Matwin and Dr. Jacques Raymond for the useful suggestions they provided when reviewing this paper.

ABSTRACT

This thesis attempts to describe the development and installation of machine instructions which implement the following control structures:

1. If-Then-Else
2. While-Do
3. Repeat-Until

The decription includes the syntax, function, restrictions, implementation and use of the new instructions in their chosen enviroments. The enviroments are an old but still commonly used machine architecture and a more recently developed pseudo machine architecture for a popular programming language.

CONTENTS

Chapter I

INTRODUCTION

Control structures have had many forms. The earliest form of control structures used the GOTO or simple branch instruction. Although it has many forms (i.e. in FORTRAN the computed, assigned and unconditional GOTO statements) it has been utilised to give structure to the execution of programs. These instructions are used explicitly, and are generated by pre-processors and compilers to implement more sophisticated control structures, such as If-Then-Else and Case. A major problem with this implementation of a control structure is that programs relying heavily on this type of instruction tend to be cryptic and unreadable[5], requiring a lot of effort, even on the author's part, to determine what had actually been done.

The development of the If-Then-Else, While-Do and Repeat-Until control structures followed a trend towards the use of more general, understandable and expressive control structures. Their use in Algol-60[8], PL360[14], Pascal[13] and more recently developed languages such as Modula-2[12] and their use in various assembler languages in the form of macros, as in Hibal[11] and Macros[10], show the versatility and general acceptance of these particular control structures.

Other control structures have been designed and are being designed. The trend is towards allowing a simple implementation of some basic programming mechanisms. The result has been the development of such control structures as the CASE statement and the If-Then-ElsIF ... Else End, although the latter is more of a syntactic solution to the problem of heavily nested IF statements. The Case statement has been implemented as an instruction in certain machine enviroments, such as P-code. However, the basic structures have not yet been implemented in many architectures. The implementation of the more generally recognised control structures using branch instructions still leaves some aspects of control structures relatively untouched by machine architecture. This thesis attempts to describe the implementation of machine instructions for the three most prominent control structures, If-Then-Else, While-Do and Repeat-Until.

It has been found by Mills[21] that these control structures have limitations as to the class of problems they can be used in. This has led to the development of other, more general control structures. The paper by Parnas[16] has shown one such control structure. Among its benefits is the apparent removal of the need for temporary variables used in the performance of the control structure and the ability to express non-deterministic algorithms.

The intent of this research has been to:

1. Move the control structures If-Then-Else, While-Do and Repeat-Until into two existing machine architectures. The I.B.M. System/370 and the PERQ Systems PERQ machine which runs an operating system using Pascal, which uses a P-Code based architecture.
2. To determine if they add any legibility to program code.
3. To document the limitations and restrictions in the implementation of these control structures and explain why they occur.

The paper first contains a brief history of control structures and the basic design of each control structure. Then, it proceeds to describe for both machine architectures, which instructions were added, how they were implemented, their syntax, function and use. Any alterations to the operating systems involved is also documented.

The appendixes at the end of the paper contain a list of the definition of each of the Q-Code op-codes found in the PERQ microcode, the listing of the interpreter micro-code for the PERQ and various source files and corresponding disassembled listings. They also include the routine which performs the instruction handling for the AMDAHL V7A, a System/370 compatible machine, two tests which verify the proper functioning of the new instructions on the AMDAHL, and a disassembled listing of one of the tests.

Chapter II

HISTORY AND DESIGN

There are very few prominent control structures. Some have been added since the first implementation of the three most prominent control structures:

1. If-Then-Else
2. While-Do, and
3. Repeat-Until

Other control structures include the Select or Case statement, the For or Do statement and the If-Then-Elsif-...-Else-End type statement.

Some of these other control structures have been implemented on certain machine architectures, to a point. For example, the For and Do statements are a commonly recognised structure found in Pascal, Fortran and PL/1. On I.B.M[17] machines, they have been implemented using the branch on index high [BXH] and branch on index less than or equal [BXLE] instructions so as to limit the number of instructions required to perform a loop with a positive or negative counter.

What we wish to do is to demonstrate how to implement the more commonly used control structures on any available machine, to determine what use they could be, and any restrictions that would apply.

2.1 A DEFINITION OF A CONTROL STRUCTURE

The implementation of a control structure depends on what properties it is given. We need to define a control structure as an entity which is used to control the flow of execution. The discussion is limited to machines which operate in a sequential manner. Because it is defined as a structure, it is a building block which can be used to map the logic of a program. To be of any use it must be available in any valid programming context.

To permit any control structure, the definition must also allow for any method of implementation. It is an abstract concept which permits the user to build a program in a logical manner. This in effect enforces the use of structured programming techniques.

2.2 A BRIEF HISTORY OF CONTROL STRUCTURES

Control structures have been around since the beginning of programming. It is simply a method by which the logical structure of a program is implemented. The structure usually resides in the reader's mind because the implementation is usually done using branch instructions. However, it has been generally recognised that the use of control structures in a programming enviroment is more suitable than implementing a program than using explicit branch instructions[5].

As previously indicated, the simple branch instruction was, and still is, the way most control structures are im-

plemented on most machines. The reason for this is that it is usually the only way to alter the program counter and change the flow of execution. Some minor changes have been made towards allowing different uses for branch instructions, for example the Branch And Link (BAL) instruction in IBM/370 architecture also places the current value of the program counter in a register before branching, but they still have restrictions on their use in structured programming (i.e. the BAL instruction is unconditional).

The most familiar control structures today are the If-Then-Else and While-Do structures. These were apparently not the first control structures. The Do statement precedes these structures by a couple of years[1]. Since then other control structures, some mentioned above, have been implemented. The purpose of these control structures was to make the task of implementing programs as easy as possible.

The easiest way to make something simple is to have it short and recognisable. This allows the concept involved in the decision to be extracted and analysed quickly. The concepts of If-Then-Else and While-Do are very easy to understand and remove many branch instructions. Removal of temporary variables also makes it easy to understand high level languages, which is why they are so attractive for general purpose programming.

---

[1] We assume Fortran-II was 'created' circa 1958 and Algol-60 was 'created' circa 1960.

In more recent years, not only has work been done in high level programming languages but in certain low level programming languages, and even machine code. On I.B.M. machines, the assemblers all have very powerfull macro processing facilities. These facilities allow the user to code a 'macro' which has the ability to test the 'type' of a parameter, and generate the appropriate code to the implement the most prominent control structures, i.e. If-Then-Else While-Do[11,10]. On certain micro-code driven machines, such as the PERQ[2], these control structures are available at the micro-code level, and are used in the micro-operating system to interpret the Q-code, a P-code dialect, which runs the Pascal based operating system.

2.3 PROPERTIES AND TRAITS

The three control structures which are of interest all have similar basic properties. Some other properties are traits of their specific function. All the control structures are performed under control of a boolean expression. They all perform some form of branching operation and they all return control to the instruction syntactically following the structure when their task has been completed.

---

[2] The PERQ is manufactured by PERQ Systems Inc., formerly the Three Rivers Computer Corporation, of Pittsburgh Pensylvania.

What follows is a simple description of the three control structures we are studying. The block A represents the boolean expression which sets the branch condition. The blocks B and C represent the object of the control structures.

2.3.1 The If-Then-Else structure

The If-Then-Else structure has two unique properties. It does not itself perform a loop and it may optionally have a null Else clause.

The common implementation of this control structure on a machine architecture is as follows:

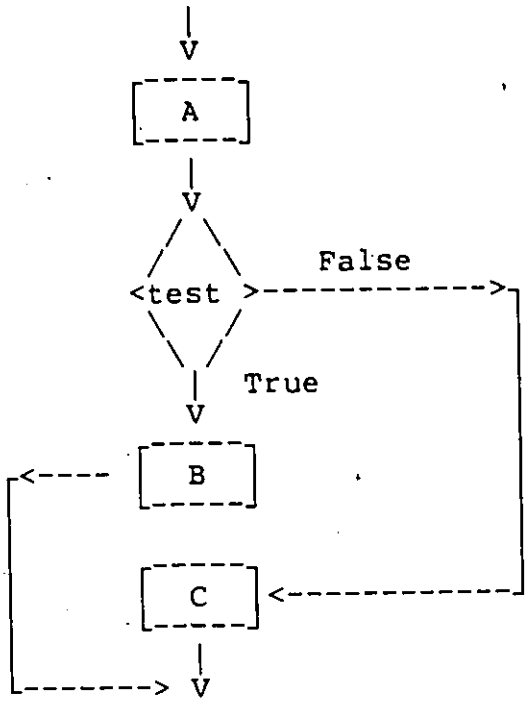


Figure 1: Layout of If A Then B (Else C) structure

2.3.2 The While-Do structure

The While-Do instruction performs a loop while the tested condition is true. If the condition is originally false, no statement inside the structure will be executed.

The common implementation of this control structure is:

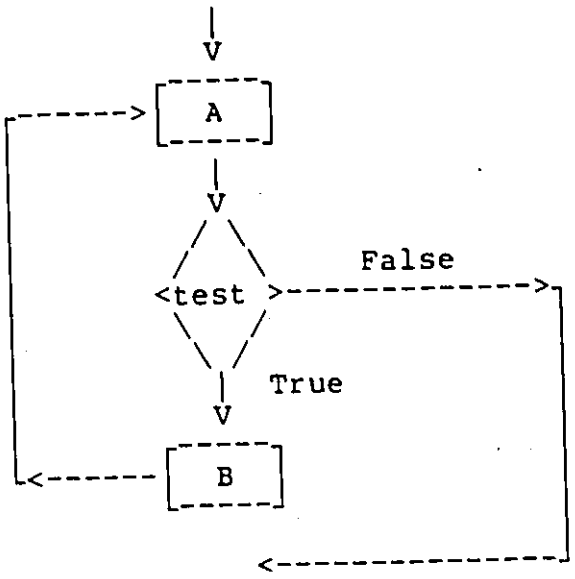


Figure 2: Layout of While A Do B structure

2.3.3 The Repeat-Until structure

The Repeat-Until structure is basically a While-Do structure with the statements internal to the While-Do structure being repeated once in front of the structure[21]. This means that the statements are executed at least once, then the condition is applied. Note that the condition used is the logical complement of that used in the If-Then-Else and While-Do structures.

The common implementation of this structure is as follows:

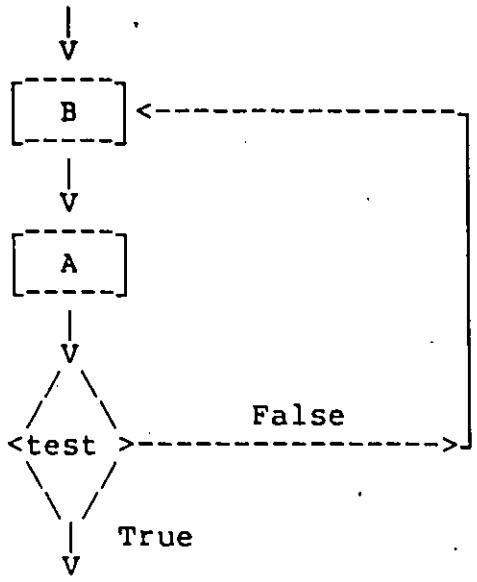


Figure 3: Layout of Repeat B Until A structure

2.4 DESIGN

To implement these control structures on a machine architecture, they should each be designed as a single instruction. They may include extra instructions to set up the correct enviroment but they must be self-contained.

To make each control structure an instruction, its basic properties must be encoded in the instruction. The basic properties of each instruction are:

1. They rely on the evaluation of a boolean expression.
2. They all contain at least one conditional branch instruction. They may also contain unconditional branch instructions.

3. When the condition allows the instruction to fall through, i.e. the statements inside the control structure are no longer to be executed, the next instruction executed is usually the next textual statement. This is not always the case in certain programming languages. For example, in some dialects of Pascal, using P-Code, there is an Exit-GoTo statement or instruction. In Fortran-IV there is the indexed RETURN statement for exiting SUBROUTINEs. These operations allow the ability to return to locations other than the one syntactically following the calling statement.
4. They contain one or two blocks of statements to be executed. Each block may contain zero or more instructions.

We should leave the basic machine architectute unchanged to allow all previously running programs to continue to run successfully with the new control structures. To this end we must break down each structure into its individual components. This will allow us to implement these structures as single instructions. What we obtain is a general design for each of the control structures.

1.

If-Then-Else

Boolean Expression	<- not part of instruction
Op-Code	<- start of instruction
Address 1	<- Address of routine

holding Then clause

Address 2 <- Address of routine
holding Else clause

If-Then

Boolean Expression <- not part of instruction

Op-Code <- start of instruction

Address <- Address of routine
holding Then clause.

2.

While-Do

Marker <- possible instruction
marking position of
boolean expression

Boolean Expression <- not part of instruction

Op-Code <- start of instruction

Address <- address of routine
holding statements
to be executed.

3.

Repeat-Until

Marker <- possible instruction
marking position of
boolean expression

Routine <- statement(s) performing
required actions inside

```
                                    loop
Boolean Expression          <- as indicated
Op-Code                     <- instruction
```

This design requires that we make some basic assumptions on how these control structures function.

1. There must exist sufficient information such that when the control structure is exited, the syntactically next sequential instruction can be executed.
2. The boolean expression to be evaluated cannot be included in the machine instruction. These expressions can be complex and can contain functions and other non-simple expressions.
3. In keeping with the design of the control structures, the boolean expressions will be calculated using facilities provided by the current architecture. The result of the boolean expression will be either the value true or false. The new instructions will test this result, and will conditionally invoke the appropriate code block or procedure. If we require that this expression be re-evaluated we must
4. Maintain a pointer to the expression so that it can be re-evaluated.
5. The instructions implementing these structures cannot perform two or more functions. For example, the evaluation of the boolean expression is a complex task which may require the evaluation of functions, array

references etc.... This would be very difficult to implement in most architectures, i.e. I.B.M. System/370, and therefore not practical.

This is not always the case. There has been work done[7] where the concepts of Top Down Structured Programming have been implemented at the machine level. The technique used three simple structures, DO, WHILE and IF-THEN-ELSE to control the flow of execution of a program. The boolean expression in this case was evaluated by processing the list provided to the instruction. The entire evaluation was performed by the instruction. This is very difficult to do in this case because the addresses used to point to the parameters are constants. We wish to allow any parameter to be used, however it is passed to the architecture, and therefore should allow for variable addressing.

6. Because the instructions cannot perform two or more functions, they will not be interruptable i.e. they will not be cut off when an interrupt becomes pending. Interrupts will be handled as specified by the principles of operation of the machines involved. The invoked procedures are not part of the instructions but of the entire control structure and inherit the masks present when the structure is invoked.

7. The information passed to an instruction (its parameters) must be passed using the normal conventions followed in the architecture. This usually means that specific locations in storage or arguments available as part of the instruction itself can be used to pass information to the instruction. The return address is maintained through the procedure call process.
8. Because the block of statements to be executed are separate from the actual decision made to use them, the block is located outside the instruction itself and is invoked as a procedure. The end of the block uses the normal procedure termination to exit to the appropriate return point.
9. The final implementation is machine dependent.

The one or more blocks which contain the statements to be executed are logically separated using the definitions of the programming languages, although recognising the bounds of a block is not always a trivial problem. This feature allows to implement each block as a procedure which is called by the instruction and has been featured in the definition of the SPLM [20 p. 138] architecture, which has not actually been implemented.

Chapter III

IMPLEMENTATIONS

The redesigned control structures were implemented on two machines. The first is the PERQ. It is a micro-programmable machine which has a host Q-code architecture and Pascal based operating system. With the micro operating system source it was possible to add and alter certain instructions to implement these new control structures.

The second machine was an AMDAHL V7A. This is an I.B.M. System/370 compatible CPU. It is compatible in that it basically handles the same machine instructions, except where machine diagnostics are involved. None of this directly affects a normal user. This machine is not micro-programmable. The implementation is done using what is termed macro-code. We use certain properties in the machine architecture to obtain information for the instruction, then perform that instruction using a special program inside the operating system itself. This approach is primarily used to test the design and implementation of an instruction before actually installing it in a machine, either by altering the hardware or the micro-program running the machine.

3.1 IMPLEMENTATION ON THE PERQ

All three control structures were added to the PERQ's micro-operating system. This was done by adding one instruction for each of the control structures and an instruction to mark the beginning of the boolean expression for the While-Do and Repeat-Until instructions. Note that the boolean expression is not part of the control instruction but it must immediately precede the instruction. Various other routines internal to the micro-operating system were altered to minimise the amount to code that had to be added to install the new instructions. One control block was altered to contain the pointer to the beginning of the boolean expression, when required.

Various resources were required by the new routines which are not immediately available to a normal user. Since the routines were implemented in micro-code, three resources were required.

1. Sufficient writeable control storage (WCS) to install the new instruction were required. This type of memory is for all intents and purposes read only.
2. Two work areas, one to hold a flag, the other to hold a counter which would be used as an offset from the code base. The code base is the location in real storage of the start of the segment which contains the procedure currently being executed.

3. The actual micro-operating system itself. Various routines were altered slightly to permit their use as subroutines, or to alter their behaviour such that they would perform the required extra tasks to initialise the execution of various instructions in the correct manner. For example, the routine which performs the instruction Load Variable Routine Descriptor (LVRD)[3] was altered so that a flag would be set indicating that the termination of the instruction was to result in the return to the calling routine, and not execution of the next instruction.

The work areas were taken from the pool of 256 20[4] bit registers available in the micro-engine. The name of the register which contains the flag is CR0. The name of the register which contains the loop pointer is NOTE1.

This specific implementation is very dependent on the PERQ's hardware characteristics. The use of the work areas and storage addressability are unique to the PERQ. The expansion of the ACB and changing the behaviour of a procedure call could be extended to Q-Code and P-Code.

---

[3] The LVRD instruction is used to place the generic address of a procedure on top of the PERQ expression stack. This address can be used as a parameter in procedures or for system 'hacking', as in this case. Q-code has three other forms of addresses used to refer to procedures.

[4] The PERQ's registers are 20 bits wide, so they can be used to address and calculate addresses for 1 megabyte of real storage using single precision arithmetic.

3.1.1 The New Instructions

The implementation of the new instructions requires the addition of some new features to the QCode architecture. At initialisation of the micro-operating system, the work areas are initialised to zero, to indicate that all is to proceed as it would have done in the unmodified system.

The two work areas play critical roles in the performance of each machine instruction. CR0, the flag register, is used to signal the modified routines that they are to take a slightly different approach to their tasks. When CR0 is zero, all functions as it would in the unmodified system. When it is one, each routine will behave in a slightly different way. The two routines which use this flag do so in the following manner:

1. The LVRD instruction is called as a subroutine by the new instructions. When the control flag is one, it will use the RETURN micro-instruction instead of proceeding to decode the next instruction.
2. The routine which builds the activation control block (ACB) for procedure calls will use the value in the NOTE1 register instead of the offset of the next instruction as the default return address.

In this way the amount of duplicated code is reduced.

The NOTE1 register is used to indicate a loop boundary. It is set during initialisation to zero. It is stored when a procedure is invoked in the ACB, then set to zero for the

new routine. It is restored when a procedure terminates, and the value is obtained from the ACB. The register is explicitly set by a NOTE instruction. The value stored is the offset from the current code base to the instruction immediately following the NOTE instruction.

Errors can occur in the invocation of the new instructions. There are three classes of errors.

1. An invalid boolean expression. This occurs when the location which is supposed to contain the value true or false does not. The Pascal compiler always places the result of a boolean expression on the top of a 16 element expression stack and this is where these instructions look for the result of the expression.
2. An invalid loop address. For the While-Do and Repeat-Until control structures, the loop boundary must be set to indicate where to go to when a loop is required. This value is found in NOTE1. The value in NOTE1 is undefined when it contains zero. This is because the NOTE instruction cannot set the value in NOTE1 to zero.
3. An invalid procedure address. In the If-Then-Else and While-Do instructions, the address of a procedure to be invoked is required. Because of compiler restrictions, this was supplied by placing the appropriate NOOP or LVRD instructions after the op-code. If these instructions are not found after the op-code, an error is detected.

Because it is possible to detect errors, the built in micro-code error routine is called with the appropriate parameters to signal the host operating system that an error has occured. Because of the restricted number of error indicators, those deemed most appropriate were selected to indicate what type of error had occured.

3.1.1.1 Opcodes

B'01010100'

Instruction

Note

Op-Code X'54'

Operands None.

Function To update the Note register. The value placed in the Note register is the offset from the code base, the start of the segment containing this routine, of the next sequential instruction in this segment. The value zero can never be placed into the Note register by this instruction.

Errors None.

B'01010001'	LVRD or NOOP	LVRD or NOOP

Instruction

If-Then-Else

Op-Code X'51'

Operands Two. Both Procedure addresses. The first is the address of the Then clause, the second is the address of the Else clause. The address is created by using either a LVRD instruction (Load Variable Routine Descriptor) or a NOOP instruction. If it is a NOOP instruction, no branch is taken if that clause is invoked.

Function This instruction takes the result of a boolean expression and conditionally performs a two way procedure call. When control is returned to the caller throught the return instruction, the next sequential instruction is executed. The result of the boolean expression must be on the top of the expression stack (TOS).

Errors

1. Expression out of range
2. Invalid routine address

B'01010010'	LVRD or NOOP

Instruction

While-Do

Op-Code X'52'

Operands The address of the routine to be invoked if the expression on top of the expression stack contains the value true. This address takes the form of either a LVRD instruction, or a NOOP instruction. If the address is a NOOP instruction, no branch is taken. The return address of the invoked routine will be that of the instruction following the last Note instruction executed in this routine.

Errors

1. Invalid expression
2. Invalid loop address
3. Invalid routine address

B'01010011'

Instruction .

Repeat-Until

Op-Code X'53'

Operands None.

Function If the expression on top of the expression stack is false, the next instruction executed will be that following the last Note instruction executed in this procedure.

Errors
1. Invalid expression
2. Invalid loop address

3.1.1.2 Compilation and Usage

Because the basic language on the PERQ is Pascal, there exists no assembler for the Q-code. Instead, the designers of the Pascal compiler allowed for the explicit inclusion of various instructions. They are either generated by the compiler to satisfy a user directive, as in the case of the MakeVRD directive which generates a complete LVRD instruction, or inserted completely by the user. Any combination of instructions is valid. The Pascal compiler does not optimise the code. See appendix C for examples of how this is done.

3.1.1.3 Decompilation

To look at the generated code, the routine QDIS was used to disassemble the generated object code. The object code generated by the Pascal compiler on the PERQ is reentrant and contains no data other than that contained in the machine instructions themselves.

QDIS is a table driven program, the table being stored as a data file somewhere on a system file structured device. The table was edited to insert the definition of the four new instructions. the results can be seen in appendix C, after the source of each program. You should note that the location of the control structures is relatively easy to spot.

3.1.2 Altered Routines

Various routines were altered to either provide their function as subroutines, or slightly alter their function so that they would not have to be copied, then modified to obtain the required function.

The PERQ's micro-code initialisation code was altered so that both the control register and the Note register would be initialised to zero.

The routine which performs the LVRD instruction was modified so that if the control register is not zero, the routine would return control to the caller using a 'return' micro-instruction, instead of executing the next instruction.

The routine which creates the ACB was altered so that the value in the Note register would be used as the default return address, not the address of the next instruction, when the control register is not zero. In all cases, this routine also stores the value of the Note register into the ACB, then sets the Note register to zero for the new procedure. This routine resets the register to zero and does not return control to the calling micro-code routine.

The routine which handles the exit of all functions and procedures was altered to restore the value of Notel. The value in Notel is always restored when a procedure returns to that lexical level in the program.

3.1.3 Altered Control Block(s)

One system control was altered. The Activation Control Block (ACB) was expanded by 1 word. This area contains the current value of the Note register when that procedure is invoked. This does not affect the addressability of any variable in the invoked procedure since the new top of the memory stack is set to the address immediately after the ACB, resulting in no control problems.

3.1.4 Interrupts

The PERQS operating system has a paging facility. The unit of space paged is a segment. There are two types of segments:

1. The data segment. This segment can be up to 64k words long.
2. The code segment. This segment can only be up to 32k words long.

All addressing is primarily based on segments.

In the implementation of the new instructions, only procedure addresses are used. These addresses are interpreted by the micro-code with the help of the operating system into real storage addresses. If the procedure is not in core, the micro-operating system will generate a page fault. The result is that there is no special requirements for system interrupts in the implementation.

The completion of I/O does not interrupt the execution of the micro-code. A flag is set indicating a pending I/O operation. This flag can be tested in the micro-code using the 'IntrPend' flag. With this type of control, handling I/O interrupts is simply done by adding the qualifier 'If IntrPend Call(Vector)'. This causes the micro-operating system to go to a routine which vectors to the correct I/O handling routine. The placement of this qualifier need only be tempered by two restrictions:

1. It uses the address field to perform a call operation, so no other operation using the address field can be used in the same micro-instruction.
2. The condition flags set by the current arithmetic operation in the ALU will be lost if the branch occurs because the next sequential instruction will not be seeing the result of the previous instruction when it tests the condition flags.

As dictated by the design, the instructions will not be interrupted by I/O interruptions or page faults. The mask which dictates when interrupts can be handled still applies and if an interrupt is to be signalled to the operating system, it will be done using the normal paths and at the appropriate times.

Because there are error conditions which can be detected by the new instructions, the routine used to generate exceptions is used to signal to the operating system that an er-

ror has occured. Because there are only a certain number of predefined error codes used by the micro-code routines to indicate various programming errors, the codes which most closely resembled the meaning of the new errors were used to indicate them.

3.1.5 Examples and Testing

Appendix C contains a number of test programs which were run to verify the proper functioning of the new instructions. To insert the new instructions, intrinsics or compiler directives were used to directly place the required instructions into the object code. The object code is also included in some smaller examples to show how they were placed.

The testing on the PERQ is supposed to exercise all paths that could be taken by any of the new instructions. These paths are those generated by conditions which require execution of the statements associated with true and false looping conditions and all the possible exceptions that might be generated by the implementation of the new instructions. No problems were found during testing.

3.2 IMPLEMENTATION ON THE AMDAHL

Only two of the three control structures were implemented as machine instructions on the AMDAHL V7A. The If-Then-Else and While-Do control structures were implemented but the Repeat-Until was not. The reason for this is that the latter control structure, when designed for implementation using IBM/370 architecture, would be implemented as a simple conditional branch instruction, which already exists in IBM/370 architecture.

The actual implementation was done using macro-code. This term refers to the fact that the instructions are not directly implemented in the machine's architecture, but are executed using other machine instructions. The new instruction causes a program check which indicates that the operation is invalid. A specially written routine determines that it is a new instruction, and performs the required changes to the user's registers and program counter to carry out the execution of these new instructions.

This method is used by IBM and those who manufacture IBM/370 compatible CPUs. It allows the design and testing of machine instuctions without having to alter the hardware or implement the instructions in micro-code. The mechnism allows the use of the same logic required in the hardware. But using hardware controls, it is possible to easily trace the execution of the pseudo instruction without interfering with the actual function of the machine.

3.2.1 The New Instructions

The specially written routine takes the form of a program written in assembler, and executed on the user's operating system, the Conversational Monitoring System(CMS) in this case. This technique has been used on other operating systems, and is extensively used in another form in VM/370[5], the host operating system for CMS.

The routine remains permanently in storage while the new instruction set is supposed to work. Another special routine loads this program into storage in such a way that it will not be removed from storage under any normal or abnormal circumstances.

To effect the proper execution of the program we use some common facts about the System/370 machine architecture in creating a short routine to determine the length of an undefined machine instruction. This architecture uses the first 2 bits of the op-code to indicate the actual length of the machine instruction. The binary patterns 00, 01, 10, and 11 correspond to 2, 4, 4, and 6 byte op-codes, even for undefined machine instructions. The result is that it is very simple to pick up all the required information from the Pro-

---

[5] VM/370 is an operating system that runs on machines that support the I.B.M. System/370 architecture. This operating system presents the user with an image of a real machine, with some extensions to allow it to communicate with VM/370. This image allows the user to execute all the functions that a real machine can but allowing for the sharing of various limited resources, i.e. CPU time, disk space, unit record devices (printers, punches etc...), tapes etc....

gram Status Word (PSW), storage and registers, and perform the instructions.

Since this architecture is consistent, any addresses used in an instruction takes the form. of a base displacement address. This relocatable address requires a general purpose register and an offset to create the absolute address for use by an instruction. This type of address is called an 'scon'. The address is used to determine the location of any subroutine to be executed. Subroutine linkage conventions in this architecture are also consistent. The register which is to contain the return address is also pointed to by the instruction.

The Branch Condition is determined in the same way it is for Branch on Condition instructions in this architecture. The branch mask is used to test the Condition Code (CC) bits found in the PSW. The mask allows the program to select and test any combination of conditions, and perform a branch if the test if successfull. This same technique is used to determine the branch condition in the new instructions.

3.2.1.1 Opcodes

B'11110100'	m1	r1	d1(b1)	d2(b2)

Instruction

If-Then-Else

Mnemonic IFTE

Op-Code X'F4'

Operands Two relocatable address constants. The first is the address of the routine to be evoked if the branch condition is true[D1(B1)]. The second is the address of the routine to be evoked if the branch condition is false[D2(B2)].

A mask[M1] is used to determine the validity of the branch condition. For a full explanation see [17] for a description of the Branch on Condition instructions.

A link register[R1], which indicates which general purpose register is to hold the address of the area immediately after this instruction.

Function This instruction performs the function of the If-Then-Else structure at the machine level. Using the mask, the instruction determines whether the branch condition is true or false. Using this in-

formation, it takes either the first or second address and uses it to calculate the new value of the program counter. In all cases, the link register will contain the address of the field immediately after this instruction.

Errors None.

B'10100001'	M1	R1	D1(B1)

Instruction

If-Then

Mnemonic IFT

Op-Code X'A1'

Operands One relocatable address[D1(B1)], a link register indicator[R1] and a mask[M1] to determine the branch condition.

Function The branch mask is used to determine the branch condition. If the condition is true, the address field is used to determine the address of the next instruction to be executed, and the program counter is updated. If the branch condition is true, the indicated register is updated with the address of the area immediately following this instruction.

Errors None.

B'10100000'	M1	R1	D1(B1)

Instruction

While-Do

Mnemonic While

Op-Code X'A0'

Operands One relocatable address[D1(B1)], a link register indicator[R1] and a mask[M1] to determine the branch condition.

Function The branch mask is used to determine the branch condition. If the condition is true, the address field is used to determine the address of the next instruction to be executed, and the program counter is updated. If the branch condition is true, the indicated register is updated with the address of this instruction.

Errors None.

3.2.1.2 Compilation and Usage

To use these new instructions, they had to be explicitly included in the assembly of some program. Since there are many assemblers for I.B.M. System/370 architecture machines, we used one of them to generate the object code.

The nature of this architecture is that data and instructions are intermingled, there is no definite pattern which identifies one from the other. To use the new instructions, they merely have to be coded directly into an instruction stream. This was done using macros, one for each of the instructions, using the mnemonics given above.

The assembler is capable of generating any form of address constant the machine can use, be it absolute or relocatable. Using this ability, the instructions were written such that they are compatible with other machine instructions. The result is that the IFT and WHILE instructions are RS type instructions, the IFTE instruction is an SS type instruction.

3.2.1.3 Decompilation

As in the PERQ, there exists a routine which can decompose the object code generated by a compiler into a series of instructions. However, since System/370 architecture allows the user to imbed data into the object code, the information provided is not entirely consistent with the original source program. The program (HMASPZAP or 'super zap') will

display an op-code for each half-word of data in the object code if the value corresponds to a valid op-code. Altering the tables allows us to insert our new op-codes. The results can be seen in appendix F. The result is that locating the control structure is now easier, but the results must be verified with the actual program being decoded.

3.2.2 Altered Control Block(s)

To install the routine which handles the new instructions, an alteration to a field in the system nucleus was required. The field, the program check new psw, was replaced with information which points the hardware to a new interrupt handler for program checks. This handler is the second part of the program which performs the new instructions.

The first part of the program actually swaps the program check new psw with another value, and remembers its contents for use by the second part. It also restores the value when called to do so. This part is evoked by the CMS command 'NEWINST'.

The only other field used in the system nucleus was the general purpose register logout area (GPRLOG). This field is used when the 'system store' operation is performed by the hardware. Since this operation is only performed when a system dump will be performed, it is not hazardous to use this field to temporarily store information. Some operating system routines use this field for the same purpose.

3.2.3 Altered Routines

No system routines were altered to implement these new instructions. The new instructions are implemented by altering a field in the operating system which points to the routine which gains control when a programming error occurs. The hardware uses this field to pass control to the operating system when a programming error occurs. By replacing this field with one which points to the new routine which performs the new machine instructions, it is possible to suppress the error and perform the require function.

The new routine determines whether the interrupt is due to one of the new instructions. When this is not the case, the host operating system, obtains control to perform the normal recovery procedure. The steps which determine this are:

1. The program check old psw is examined to determine which type of program check caused the interrupt. To be handled by the new routine it must be an undefined machine instruction.
2. Determine the supposed length of the instruction that was to be executed. This information can be determined from the instruction length field in the old PSW.
3. The address of the instruction is then determined.
4. The op-code is checked. If it is one of the new instructions, a branch to the appropriate routine is made to emulate the instruction.

This method, sometimes refered to as 'macro-coding' the instructions, is usually used when new instruction logic and performance are to be checked . It is easier to install and verify the performance of the instructions in this case than to alter to hardware to execute the instructions.

3.2.4 Interrupts

Because of the method of implementation, no hardware interrupts will be fielded by the code handling the instructions. Interrupts handled by the software fall into two categories:

1. Those generated by the hardware, and
2. Those generated by the software.

Those interrupts generated by the hardware, I/O, external, and machine check interrupts are disabled during the execution of this code. Interrupts in the second category are not generated. Supervisor calls are explicitly generated by the 'SVC' instruction, which is not used in the emulating routine. Program checks, which are generated by program errors, are not generated if the emulating program is correct.

Errors can be generated by the new instructions, but the hardware will perform the checks far more quickly than the emulating program. These errors, having to do with storage addressing and protection, will be checked by the hardware when the instruction emulation has been completed. The appropriate errors will be generated by the hardware, so the tests have not been included in the emulating code.

3.2.5 Examples and Testing

Appendix E contains two sample programs which make use of the If-Then-Else, If-Then and While-Do instructions. The tests for these new instructions were designed to exercise all the conditions to which the new instructions could be exposed. Since there were no exceptions that could be generated that would not be detected by the hardware, only simple programs which exercised the consistency of the instructions were written. These tests found one problem. This problem was associated with the incorrect determination of the condition code. It was easily found and resolved using the instruction tracing mechanism available in the hardware coupled to some software in the host operating system.

3.3 OBSERVATIONS

There are some technical and aesthetic aspects to these new instructions on the AMDAHL which differ from the ones implemented on the PERQ.

1. The IF-THEN and IF-THEN-ELSE instructions are different on the AMDAHL. This occurs because there is no simple method to indictate to the hardware that the address is a no-op. The method which is used in some System/370 instructions to indicate a null address cannot be used for these new instructions. The resulting address is calculated as zero, which in this case is valid, and cannot be ignored.

2. The branch instruction is not so nearly restricted as it is in the PERQ. Because addressing is done using actual storage addresses and not offsets, as it is on the PERQ, a jump could be calculated to jump from one procedure to another. On the PERQ, this would be difficult, if not impossible in some circumstances, to do.
3. The control block addresses could be passed as parameters. For the PERQ the address was hand coded into the instruction, but using the relocatable address format in the System/370 does allow a parameter to be used. The PERQs instruction format could be altered to allow this type of function.
4. The nesting of procedures on the PERQ and AMDAHL are limited by the amount of storage available. For the PERQ, the total size of stack cannot exceed 64K. For the AMDAHL it is limited to amount of storage that can be addresses, 16384K (or 16 megabytes). In either case, the amount of nesting could get very unwieldy, as it is for heavily nested IF statements.
5. The evaluation of some boolean expressions may result in the generation of code which is far more complex than the control structure being invoked. If a compiler or pre-processor attempts to optimise the evaluation of a boolean expression, many branch instructions may be generated in an attempt to quickly

evaluate an expression. This would cause severe problems in decoding the resulting object code.

6. The actual implementation of these control structures results in the creation of conditional procedure calls, which are not available in either of the architectures studied.

Chapter IV

CONCLUSIONS

The result of attempting to implement various control structures in different architectures pointed out problems in some currently used machine architectures which restrict the implementation of new instructions. This can be observed in the context of machine architecture limitations and dependencies which usually fall into the following categories:

1. Address space size.
2. Real storage size. This does not have to be the same as address space size. For example, the IBM System/38 [20 pp 96-98] has a 48 bit addressing capability but only 31 bit real storage addressing. The user only sees the 48 bit architecture.
3. Hardware / software interface, i.e. interrupt vectors and I/O ports.
4. Interrupt mechanisms
5. Addressability constraints. These include (a) how references to storage are made, and (b) What storage is available at any one time.
6. Non-Uniform instruction sets, for example an instruction available for one data type but not available for another similar data type.

7. Limited instruction sets. i.e. no conditional procedure call instructions. The PERQ micro-code has a conditional procedure call feature but the micro-operating system (Q-code) did not.

Of all these dependencies, only the last three have any bearing on how difficult it is to implement control structures in a given machine architecture.

These dependencies manifest themselves in various ways:

1. How the boolean expression is evaluated.
2. How procedure addresses are passed to the instruction
3. The lack of suitable instructions to implement these control structures without explicit new instructions, i.e. a conditional procedure call.

For these reasons it was decided to break down the control structures. We split up the implementation of the control structures into a test, a conditional subroutine call, and a subroutine which contains the block of statements to be executed. Using procedure coding conventions in the appropriate architecture, it is possible to implement the control structures with very little effort. However, because users can program any way they wish, it is by no means certain that the control structures will be used. It is still up to the user to make a conscious effort to use these available control instructions.

In some machine architectures, these dependencies result in the inability to effectively implement some of these con-

trol structures. For example, on the IBM/370 machine, the Repeat-Until structure is best implemented using a conditional branch instruction. Even if we break the block of statements inside the structure into a procedure call, the loop is still formed by a simple conditional branch instruction.

The advantages of properly implemented control structures outweigh the disadvantages. The major disadvantage in using this method of implementing control structures is that it severely restricts the use of the simple branch instruction. For example, in PASCAL the following program segment would be valid:

```
If A Then Begin
        1 : B
    End
Else If C Then Begin
        D
    End
Else GoTo 1
```

Although this is generally recognised as very poor programming style, it is still possible to code a program in this manner.

This expression would not be possible using the new instructions because the result would cross procedure boundaries. A GoTo statement can be implemented (as it is in PL/1) so that it can be allowed to cross procedure boundaries, both forward and backward, but the overhead is significant.

Another minor drawback is the overhead, compared to current implementations using branch instructions, involved in entering and exiting a block of code. This overhead is due to the procedure call and exit requirements. In some cases this is not severe because the actual events of entering and exiting a procedure are each contained in one instruction, i.e. in Q-code on the PERQ. The actual severity of this drawback is machine and implementation dependent.

The major advantage is the simplification of looking at the flow of the program at the object code level. Anybody who has tried to debug a high level language program at the machine level will appreciate the ability to associate machine instructions directly with program. This does not however remove all the problems involved with finding the statement in the program associated with the machine code.

Another advantage falls into the domain of assembler language programmers. The introduction of conditional procedure call instructions will allow the user to write modular programs using single machine intructions instead of using convoluted code to accomplish the same thing. For example, the normal way to execute a conditional subroutine call is:

```
        Test
        Branch to End If False (or true)
        Subroutine Call
End:
```

With the new instruction it could be done in the following manner:

Test

Subroutine Call If True (or false)

Although this work does help in looking at some problems in using high level programming concepts at the machine level, some more work could be done in:

1. Conditional procedure calls. This type of instruction would be of great help to programmers who use assemblers.
2. Coupling Source and Object programs in the machine architecture. Although work has been done in many versions of Pascal[22, 23, 24] to provide debugging information when program problems occur, this is still not done with sufficient consistency to help debug user programs. For example Pascal 8000[22] only provides for trace back and variable dumps. This does not allow for break points and interactive debugging, where Pascal/VS[23] and Waterloo Pascal[24] do.
3. More uniform instruction sets would of great use in trying to debug programs i.e. one ADD instruction instead of many, each working on one data type. From the disassembly listings provided in the appendixes, it still is a great chore to follow the logic of a program. With these new instructions, it would be easier to locate control structures.

Appendix A

QCODE DEFINITIONS

The following is a list of the Qcode* op-codes taken from the operating system definition. Included is a comparison of the p-code op-code list. Please note that there is not a one to one correspondence. Some functions available in p-code (i.e. sine, cosine etc... and various addressing operations) are not available in Q-Code and conversely some functions in Q-Code (i.e. the raster-op function, store writeable control storage as well as various addressing operations) are not available in P-Code. The four new instructions are also included in this list and have the values 80 through 83. The op-code values are in decimal.

---

* These definitions are reprinted with the permission of PERQ systems.

Q-Code	Op-Code	P-Code	Op-Code
LDC0	0	SLDC	0
LDC1	1	SLDC	1
LDC2	2	SLDC	2
LDC3	3	SLDC	3
LDC4	4	SLDC	4
LDC5	5	SLDC	5
LDC6	6	SLDC	6
LDC7	7	SLDC	7
LDC8	8	SLDC	8
LDC9	9	SLDC	9
LDC10	10	SLDC	10
LDC11	11	SLDC	11
LDC12	12	SLDC	12
LDC13	13	SLDC	13
LDC14	14	SLDC	14
LDC15	15	SLDC	15
LDCMO	16		
LDCB	17		
LDCW	18	LDCI	199
LSA	19	LCA	166
ROTSHI	20		
STIND	21	STO	154
LDCN	22	LDCN	159
LDB	23	LDB	190
STB	24	STB	191
LDCH	25		
LDP	26	LDP	186
STPF	27	STP	187
STCH	28		
EXGO	29		
QAND	30	LAND	132
QOR	31	LOR	141
QNOT	32	LNOT	147
EQUBool	33	EQUBOOL	175
NEQBool	34	NEQBOOL	183
LEQBool	35	LEQBOOL	180
LESBool	36	LESBOOL	181
GEQBool	37	GEQBOOL	176
GTRBool	38	GTRBOOL	177
EQUI	39	EQUI	195
NEQI	40	NEQI	203
LEQI	41	LEQI	200
LESI	42	LESI	201
GEQI	43	GEQI	196
GTRI	44	GTRI	197
UNDF45	45		
UNDF46	46		
UNDF47	47		
UNDF48	48		
UNDF49	49		
UNDF50	50		
EQUStr	51	EQUSTR	175 4

NEQStr	52	NEQSTR	183	4
LEQStr	53	LEQSTR	180	4
LESStr	54	LESSTR	181	4
GEQStr	55	GEQSTR	176	4
GTRStr	56	GTRSTR	177	4
EQUByt	57	EQUBYT	175	10
NEQByt	58	NEQBYT	183	10
LEQByt	59	LEQBYT	180	10
LESByt	60	LESBYT	181	10
GEQByt	61	GEQBYT	176	10
GTRByt	62	GTRBYT	177	10
EQUPowr	63	EQUPOWR	175	8
NEQPowr	64	NEQPOWR	183	8
LEQPowr	65	LEQPOWR	180	8
SGS	66	SGS	151	
GEQPowr	67	GEQPOWR	176	8
SRS	68	SRS	148	
EQUWord	69	EQUWORD	175	12
NEQWord	70	NEQWORD	183	12
ABI	71	ABI	128	
ADI	72	ADI	130	
NGI	73	NGI	145	
SBI	74	SBI	149	
MPI	75	MPI	143	
DVI	76	DVI	134	
MODI	77	MODI	142	
CHK	78	CHK	136	
UNDF79	79			
AIF	80			
AWHILE	81			
AREPEAT	82			
ANOTE	83			
UNDF84	84			
UNDF85	85			
UNDF86	86			
UNDF87	87			
INN	88	INN	139	
UNI	89	UNI	156	
QINT	90	INT	140	
DIF	91	DIF	133	
EXITT	92	EXIT	158	4
NOOP	93	NOP	215	
REPL	94			
REPL2	95			
MMS	96			
MES	97			
LVRD	98			
LSSN	99			
XJP	100	XJP	172	
PSW	101			
RASTOP	102			
STRTIO	103			
PBLK	104			
INTOFF	105			

INTON	106		
LDLB	107	LDL	202
LDLW	108		
LDL0	109		
LDL1	110	SLDL0	216
LDL2	111	SLDL1	217
LDL3	112	SLDL2	218
LDL4	113	SLDL3	219
LDL5	114	SLDL4	220
LDL6	115	SLDL5	221
LDL7	116	SLDL6	222
LDL8	117	SLDL7	223
LDL9	118	SLDL8	224
LDL10	119	SLDL9	225
LDL11	120	SLDL10	226
LDL12	121	SLDL11	227
LDL13	122	SLDL12	228
LDL14	123	SLDL13	229
LDL15	124	SLDL14	230
LLAB	125	LLA	198
LLAW	126		
STLB	127	STL	204
STLW	128		
STL0	129		
STL1	130		
STL2	131		
STL3	132		
STL4	133		
STL5	134		
STL6	135		
STL7	136		
LDOB	137		
LDOW	138	LDO	167
LDO0	139		
LDO1	140	SLDO1	232
LDO2	141	SLDO2	233
LDO3	142	SLDO3	234
LDO4	143	SLDO4	235
LDO5	144	SLDO5	236
LDO6	145	SLDO6	237
LDO7	146	SLDO7	238
LDO8	147	SLDO8	239
LDO9	148	SLDO9	240
LDO10	149	SLDO10	241
LDO11	150	SLDO11	242
LDO12	151	SLDO12	243
LDO13	152	SLDO13	244
LDO14	153	SLDO14	245
LDO15	154	SLDO15	246
LOAB	155		
LOAW	156		
STOB	157		
STOW	158	SRO	171
STO0	159		

STO1	160		
STO2	161		
STO3	162		
STO4	163		
STO5	164		
STO6	165		
STO7	166		
MVBB	167	MVB	169
MVBW	168		
MOVB	169	MOV	168
MOVW	170		
INDB	171		
INDW	172	IND	163
LDIND	173	SINDO	248
IND0	173	SIND0	248
IND1	174	SIND1	249
IND2	175	SIND2	250
IND3	176	SIND3	251
IND4	177	SIND4	252
IND5	178	SIND5	253
IND6	179	SIND6	254
IND7	180	SIND7	255
LGAWW	181		
STMW	182	STM	189
STDW	183		
SAS	184	SAS	170
ADJ	185	ADJ	160
CALLL	186	CLP	206
CALLV	187		
ATPB	188		
ATPW	189		
WCS	190		
JCS	191		
LDGB	192		
LDGW	193		
LGAB	194		
LGAW	195	LAO	165
STGB	196		
STGW	197		
UNDF198	198		
UNDF199	199		
RET	200	RNP	173
MMS2	201		
MES2	202		
LDTP	203		
JMPB	204		
JMPW	205	UJP	185
JFB	206		
JFW	207	FJP	161
JTB	208		
JTW	209		
JEQB	210		
JEQW	211	NFJ	212
JNEB	212		

```
JNEW      213       EFJ      211
IXP       214       IXP      192
LDIB      215       LOD      182
LDIW      216
LIAB      217       LDA      178
LIAW      218
STIB      219       STR      184
STIW      220
IXAB      221
IXAW      222       IXA      164
IXA1      223
IXA2      224
IXA3      225
IXA4      226
TLATE0    227
TLATE1    228
TLATE2    229
EXCH      230
EXCH2     231
INCB      232
INCW      233       INC      162
CALLXB    234
CALLXW    235       CXP      205
LDMC      236       LDC      179
LDDC      237
LDMW      238       LDM      188
LDDW      239
STLATE    240
LINE      241
ENABLE    242
QRAISE    243
LDAP      244
UNDF245   245
UNDF246   246
UNDF247   247
UNDF248   248
UNDF249   249
ROPS      250            { See below for 2nd byte }
INCDDS    251
LOPS      252            { See below for 2nd byte }
KOPS      253            { See below for 2nd byte }
BREAK     254
REFILLOP  255
```

```
{-------------------------------------------------------------
{
{ Long Operations - Second byte of LOPS opcode
{
{-------------------------------------------------------------}
```

```
CVTLI     0
CVTIL     1
ADL       2
NGL       3
```

```
SBL         4
MPL         5
DVL         6
MODL        7
ABL         8
EQULong     9
NEQLong     10
LEQLong     11
LESLong     12
GEQLong     13
GTRLong     14
LUNUSED     15
```

```
{-------------------------------------------------------------
{
{ Real Operations - Second byte of ROPS opcode
{
{-------------------------------------------------------------}
```

```
TNC         0           TNC       158 22
FLT         1           FLT       138
ADR         2           ADR       131
NGR         3           NGR       146
SBR         4           SBR       150
MPR         5           MPR       144
DVR         6           DVR       135
RND         7           RND       158 23
ABR         8           ABR       129
EQUReal     9           EQUREAL   175 2
NEQReal     10          NEQREAL   183 2
LEQReal     11          LEQREAL   180 2
LESReal     12          LESREAL   181 2
GEQReal     13          GEQREAL   176 2
GTRReal     14          GTRREAL   177 2
RUNUSED     15
```

```
{-------------------------------------------------------------
{
{ Spice Kernel Operations - Second byte of KOPS opcode
{
{-------------------------------------------------------------}
```

```
KBLOCK              15
KUNBLOCK            14
KSLEEP              13
KWAKEUP             12
KREMOVEFROMQUEUE    11
KADDTOQUEUE         10
KRESUMEMICROSTATE   9
KCLOCKTICK          8
KINITQUEUES         7
KINTRSRV            6
KINTROFF            5
KINTRON             4
```

KSETSOFT	3
KCLEARSOFT	2
KCURPROCESS	1
KUNUSED	0

Appendix B

PERQ MICROCODE SOURCE

The implementation of the four new machine instructions required one new routine which is comprised of 54 micro-instructions and 20 micro-instructions added or altered in various old routines. This comprises about 2.5% of 2880 micro-instructions in the micro-operating system. If some parts of the code had simply been duplicated, about 31 more micro-instructions would have been required, bringing the total to 105 micro-instructions added or modified, or 3.6% of the micro operating system. It is not possible to implement all the new instructions without changing some parts of the micro-operating systems, specifically the common CALL and RETURN routines and the initialisation of the micro operating system.

The routines that were modified comprise the basic CALL and RETURN mechanisms, the LVRD instruction which was modified to behave as a subroutine instead of a simple machine instruction and the initialisation of the micro operating system.

Certain components of the PERQs interpreter micro-code' are contained in this appendix. Only those portions which were heavily modified to provide support for the new instructions are included. Those portions which were modified are marked by a '|' to the left of the text. Those portions which were added are marked by a '*' to the left of the text. Certain alterations are not marked because the only change was to add a label to those instructions.

The only changed code not included is the removal of the invalid instruction routine(s) for the new instructions. There is a lot of micro-code, both interpreter and other material, which is not included to keep the appendix short. You will find at the end of this appendix, a list of pages which contain the routines which were either added or modified to implement the new instructions.

N.B. The object code is in octal.

' The listing in this appendix is reprinted with the permission of PERQ systems.

```
$Title Perq.Micro - Perq Q-Code interpreter microcode.
! Perq Microcode.
!-----------------------------------------------------------------------
!
!       Perq.Micro - Perq Q-code interpreter microcode.
!       Horst Mauersberg, Brian Rosen, Miles Barel      ca. 1 Jan 80.
!       J. P. Strait                    , rewritten     21 Nov 80.
!       Copyright (c) Three Rivers Computer Corporation, 1980.
!
! Abstract:
!
!-----------------------------------------------------------------------
```

Perq.Micro - Perq Q-Code interpreter microcode
VA PA X Y A B W H AL F SF Z CN JP VT


```
! 13 Jan 82   V2.5   WJHansen
! change , to ; in StkOv (in Perq.Routine.1) to help PrqPlace

! 31 Dec 81   V2.4   M. Kristofic
! Added floating point.

!  9 Sep 81   V2.3   J. Strait
! Fix bugs in double precision--see change history in double precision

! 14 May 81   V2.2   G. Robertson.
! 1. Moved IO up to 4400, expanded space for Perq to 2.25K.
! 2. Added double precision arithmetic operations.
! 3. Added Spice kernal operations.
! 4. Added RO and Line as part of interpreter.

! 14 Mar 81   V2.1   J. Strait.
! 1. Begin installing exception handling microcode.
! 2. Make sure that the SL and RA from the main program are zero, and
!    SL of procedures inside the main program are also zero.  This is
!    for stack searches for exceptions.
! 3. Minor bug corrections to stack overflow processing.
! 4. Bug correction to external calls.

! 21 Nov 80   V2.0   J. Strait.
! Start file.
```

Perq.Micro - Perq Q-Code interpreter microcode
VA PA X Y A B W H AL F SF Z CN JP VT


```
$NoList
 xxxWord - Multiple word comparisons.
```

Perq.Micro - Perq Q-Code interpreter microcode
VA PA X Y A B W H AL F SF Z CN JP VT

```
Perq.Micro - Perq Q-Code interpreter microcode
VA    PA   X   Y A B W H AL F SF   Z CN JP   VT
```

```
$List
 xxxWord - Multiple word comparisons.
File: Perq.QCode.4   Perq Microcode  Page        5

$Include Perq.QCode.4
$Title Jxxx, XJP - Jumps.
```

```
Perq.Micro - Perq Q-Code interpreter microcode
VA    PA   X   Y A B W H AL F SF   Z CN JP   VT
```

```
 Jxxx, XJP - Jumps.
File: Perq.QCode.4   Perq Microcode  Page        6
```

```
! Opcode JMPB.
!-------------------------------------------------------------------------
!
! Abstract:
!       JMPB is a two byte unconditional jump instruction.
!
! Instruction:
!       JMPB    Offset
!
! Environment:
!       old PC = Byte address + 1 of the Offset operand.  PC = UPC * 2
!
! Result:
!       new PC = old PC + Offset.
!
! Calls:
!       AdjustPC.
!
!-------------------------------------------------------------------------

! Opcode JFB, JTB.
!-------------------------------------------------------------------------
!
! Abstract:
!       JFB and JTB are two byte conditional jump instructions which j
!       if the value on the top of the expression stack is false or tr
!       respectively.
!
! Instruction:
!       JxB     Offset
!
! Environment:
!       old PC = Byte address + 1 of the Offset operand.  PC = UPC * 2
!       (Tos) = Boolean value.
!
! Result:
!       Stack popped.
!       If condition met then new PC = old PC + Offset.
!       If condition not met then new PC = old PC.
!
! Calls:
!       AdjustPC.
```

```
!-----------------------------------------------------------------------
! Opcode JEQB, JNEB.
!-----------------------------------------------------------------------
!
! Abstract:
!       JEQB, JNEB are two byte conditional jump instructions which ju
!       if the two values on the top of the expression stack are equal
!       not equal respectively.
!
! Instruction:
!       JxB     Offset
!
! Environment:
!       old PC = byte address + 1 of the Offset operand.  PC = UPC * 2
!       (Tos) = Value0.
!       (Tos-1) = Value1.
!
! Result:
!       Stack popped twice.
!       If condition met then new PC = old PC + Offset.
!       If condition not met then new PC = old PC.
!
! Calls:
!       AdjustPC.
!
!-----------------------------------------------------------------------
```

```
JMPB1:
Opcode(JMPB), JmpOffset := NextOp;              ! byte offset
        If ByteSign Goto(JMPB2);                ! if backward jump
        Goto(AdjustPC);

JMPB2:  JmpOffset := JmpOffset or(not 377,.Goto(AdjustPC);  ! sign ext

Opcode(JFB), Tos and 1, Pop;
JFB1:   If Eql Goto(JMPB1);
PuntByte: NextOp;
        NextInst(0);

Opcode(JTB), Tos and 1, Pop;
JTB1:   If Neq Goto(JMPB1);
        NextOp;
        NextInst(0);

Opcode(JEQB), tmp := Tos, Pop;
        Tos xor tmp, Pop, Goto(JFB1);

Opcode(JNEB), tmp := Tos, Pop;
        Tos xor tmp, Pop, Goto(JTB1);
```

```
1150   314   52   0 1 0 1 0   0 3 17 164   4  3 3513
1151   315    0   0 6 0 0 0   0 3 15 320   7  3 1153
1152   316    0   0 6 0 0 0   0 3 10 272   0  3 3503

1153  1057   52 377 6 1 1 0  10 3 10 272   0  3 3503

1154   304    0   1 7 1 0 0   4 0  5 326   0  3    -
1155    51    0   0 6 0 0 0   0 3 17  63  15  3 1150
1156    52    0   0 1 0 0 0   0 3 17 164   4  3 3513
1157    53    0   0 6 0 0 0   0 0  0 377   0  2    -

1160   274    0   1 7 1 0 0   4 0  5 322   0  3    -
1161    55    0   0 6 0 0 0   0 3 17  63  10  3 1150
1162    56    0   0 1 0 0 0   0 3 17 164   4  3 3513
1163    57    0   0 6 0 0 0   0 0  0 377   0  2    -

1164   264   30   0 7 0 1 0   0 0  5 354   0  3    -
1165    23    0  30 7 0 0 0  12 0  5 326   0  3 1155

1166   254   30   0 7 0 1 0   0 0  5 350   0  3    -
1167    27    0  30 7 0 0 0  12 0  5 322   0  3 1161
```

```
! Opcode JMPW.
!-------------------------------------------------------------------------
!
! Abstract:
!       JMPW is a three byte unconditional jump instruction.
!
! Instruction:
!       JMPW    LowByteOffset HighByteOffset
!
! Environment:
!       old PC = Byte address + 1 of the HighByteOffset operand.
!                PC = UPC * 2 + BPC.
!
! Result:
!       new PC = old PC + Offset.
!
! Calls:
!       AdjustPC.
!
!-------------------------------------------------------------------------

! Opcode JFW, JTW.
!-------------------------------------------------------------------------
!
! Abstract:
!       JFW and JTW are three byte conditional jump instructions which
!       if the value on the top of the expression stack is false or tr
!       respectively.
!
! Instruction:
!       JxW     LowByteOffset HighByteOffset
!
! Environment:
!       old PC = Byte address + 1 of the HighByteOffset operand.
!                PC = UPC * 2 + BPC.
!       (Tos) = Boolean value.
!
! Result:
!       Stack popped.
!       If condition met then new PC = old PC + Offset.
!       If condition not met then new PC = old PC.
!
! Calls:
!       AdjustPC, PuntByte.
!
!-------------------------------------------------------------------------

! Opcode JEQW, JNEW.
!-------------------------------------------------------------------------
```

```
Perq.Micro - Perq Q-Code interpreter microcode   Jxxx, XJP - Jumps.
 VA    PA   X   Y A B W H AL F SF   Z CN JP   VT  File: Perq.QCode.4   Perq Microcode  Page        9

1170   310  52   0 1 0 1 0  0 3 17 164  4  3 3513
1171   311  31   0 1 0 1 0  0 3 17 164  4  3 3513
1172   312  31   0 6 0 0 0  0 2  0 160  0 16    -
1173   313  52  52 0 0 1 0  7 3 10 272  0  3 3503

1174   300   0   1 7 1 0 0  4 0  5 306  0  3    -
1175    71   0   0 6 0 0 0  0 3 17  67 15  3 1170
1176    72   0   0 1 0 0 0  0 3 17 164  4  3 3513
1177    73   0   0 6 0 0 0  0 3 17 325  0  3 1156

1200   270   0   1 7 1 0 0  4 0  5 276  0  3    -
1201   101   0   0 6 0 0 0  0 3 17  67 10  3 1170
1202   102   0   0 1 0 0 0  0 3 17 164  4  3 3513
1203   103   0   0 6 0 0 0  0 3 17 325  0  3 1156

1204   260  30   0 7 0 1 0  0 0  5 330  0  3    -
1205    47   0  30 7 0 0 0 12 0  5 306  0  3 1175

1206   250  30   0 7 0 1 0  0 0  5 315  0  3    -
1207    62   0  30 7 0 0 0 12 0  5 276  0  3 1201
```



```
!
! Abstract:
!       JEQW, JNEW are three byte conditional jump instructions which
!       if the two values on the top of the expression stack are equal
!       not equal respectively.
!
! Instruction:
!       JxW     LowByteOffset HighByteOffset
!
! Environment:
!       old PC = Byte address + 1 of the HighByteOffset operand.
!                PC = UPC * 2 + BPC.
!       (Tos) = Value0.
!       (Tos-1) = Value1.
!
! Result:
!       Stack popped twice.
!       If condition met then new PC = old PC + Offset.
!       If condition not met then new PC = old PC.
!
! Calls:
!       AdjustPC, PuntByte.
!
!-----------------------------------------------------------------------

JMPW1:
Opcode(JMPW), JmpOffset := NextOp;              ! why not use WordParm?
        tmp1 := NextOp;
        tmp1, LeftShift(10);
        JmpOffset := Shift or JmpOffset, Goto(AdjustPC);  ! jump offse

Opcode(JFW), Tos and 1, Pop;
JFW1:   If Eql Goto(JMPW1);
        NextOp;
        Goto(PuntByte);

Opcode(JTW), Tos and 1, Pop;
JTW1:   If Neq Goto(JMPW1);
        NextOp;
        Goto(PuntByte);

Opcode(JEQW), tmp := Tos, Pop;
        Tos xor tmp, Pop, Goto(JFW1);

Opcode(JNEW), tmp := Tos, Pop;
        Tos xor tmp, Pop, Goto(JTW1);
```

Perq.Micro - Perq Q-Code interpreter microcode
 VA   PA   X   Y A B W H AL F SF   Z CN JP   VT

```
! Opcode XJP.
!-----------------------------------------------------------------------
!
! Abstract:
!       XJP is a variable length instruction that implements the Pasca
!       case statement.  It is an n-way branch which chooses the targe
!       based on an integer value in some range Low..High.  Three word
!       follow the XJP in the code stream, and they must be word align
!       A noise byte is added when the XJP opcode is in a low order by
!       The three words are the minimum index (Low), the maximum index
!       (High), and the self-relative address of the code to be execut
!       when the case selector is outside the range Low..High.  A word
!       aligned jump table immediately follows the maximum index, and
!       each word in the table contains a self-relative address.
!
! Instruction:
!       XJP             Low
!                       High
!                       Address for out of range case
!       JumpTable:      Address for Low case
!                       Address for Low+1 case
!                           ...
!                       Address for High case
!
! Environment:
!       (Tos) = Index = Case selector.
!
! Result:
!       Stack popped.
!       If (Index < Low) or (Index > High) then
!                       new PC = JumpTable byte address + 2 * (High-Low).
!       Otherwise       new PC = JumpTable byte address + JumpTable(Index -
!
! Calls:
!       RefillJmp, VectSrv.
!
!-----------------------------------------------------------------------
```

```
1210  1154   0   0 5 0 0 0  0 2  0 374  0 16    -
1211  1155  30   1 0 1 1 0 14 2 10 341  0 16    -
1212  1156  16  16 0 0 1 0 14 1 16 350  0  3    -
1213  1027  16   3 6 1 1 0 14 3 15 266  0  3    -
1214  1111  31   0 3 0 1 0  0 3 17 236  2  1 3331
1215  1112  16   2 6 1 0 0 16 1 16 272  0  3    -
1216  1105  32  31 7 0 1 0 16 2  0 341  0 16    -
1217  1106  31   0 7 0 1 0  0 0  5 275 12  3 1225
1220  1107  32  60 6 0 1 0  4 3 15 314  0  3    -
1221  1063   0  31 3 0 0 0 16 3 15 306  0  3    -
1222  1071  16  32 6 0 0 0 14 1 16 276 12  3 1224
1223  1072  16  32 6 0 1 0 14 3 15 301  0  3 1227
```

```
Opcode(XJP), UState, Field(0,4);                    ! read BPC
        tmp := Shift + 1, RightShift(1);            ! round up to word bounda
        UPC := Shift + UPC, Fetch;                  ! fetch Low
        UPC := UPC + 3;                             ! JumpTable word address
        tmp1 := Md1, if IntrPend Call(VectSrv);     ! Low
        MA := UPC - 2, Fetch;                       ! fetch High
        tmp2 := Tos - tmp1, RightShift(1);          ! offset within JumpTable
        tmp1 := Tos, Pop, if Lss Goto(Case2);       ! if Index < Low
        tmp2 := tmp2 and AllOnes;                   ! ensure tmp2 is good off
        Md1 - tmp1;
        UPC + tmp2, Fetch,                          ! fetch JumpTable(Index -
                                if Lss Goto(Case1); ! if Index > High
!       Case selector is in range.

        UPC := UPC + tmp2, Goto(Case3);             ! word address of JumpTab
```

```
Perq Micro - Perq Q-Code interpreter microcode
 VA    PA    X    Y  A  B  W  H  AL  F  SF    Z  CN  JP   VT
1224  1101   0    0  3  0  0  0   0  3  17  236   2   1 3331
1225  1102  16    1  6  1  1  0  16  1  16  302   0   3    -
1226  1075   0    0  6  0  0  0   0  2   0  341   0  16    -
1227  1076  32    0  3  0  1  0   0  3  13  313   0   3    -
1230  2064  32    0  6  0  0  0   0  3  15  255  12   3 1237
1231  2065  16   16  0  0  1  0  14  2   0  340   0  16    -
1232  2066  31    7  0  1  1  0   4  3  17  236   2   1 3331
1233  2067  32    1  6  1  1  0   4  3  15  310   0   3    -
1234  1067  31   32  6  0  1  0  14  3  15  300   0   3    -

1235  1077  31    0  6  0  0  0   0  0  13  304   0   3    -
1236  1073  16    3  6  1  1  0   5  3  10  261   0   3 3525


1237  1122  30   61  0  0  1  0   7  2   0  340   0  16    -
1240  1123  16   30  6  0  1  0  14  3  13  311   0   3 1232
```

```
!       Case index is out of range, jump to default.
 Jxxx, XJP - Jumps.
File: Perq.QCode.4   Perq Microcode  Page       11
Case1:  Md1, if IntrPend Call(VectSrv);       ! allow mem to finish
Case2:  UPC := UPC - 1, Fetch;                ! fetch address for defau
        RightShift(1);
Case3:  tmp2 := Md1;                          ! byte offset
        tmp2, if Lss Goto(Case5);             ! if jumping backward
        UPC := Shift + UPC, LeftShift(1);     ! add word offset to UPC
Case4:  tmp1 := Shift and 7, if IntrPend Call(VectSrv);  ! word in qua
        tmp2 := tmp2 and 1;                   ! byte in word
        tmp1 := tmp1 + tmp2;                  ! byte in quad = word in
                                              !              + byte in
        BPC := tmp1;                          ! set BPC
        UPC := UPC and not 3, Goto(RefillJmp);  ! UPC is a quad addres

!       Byte offset is negative, sign extension is necessary.

Case5:  tmp := Shift or SignXtnd, LeftShift(1);
        UPC := UPC + tmp, Goto(Case4);
```

```
$Title CALLx, LVRD, RET, EXITT, EXGO - Calls and returns.
```

```
Perq Micro - Perq Q-Code interpreter microcode
 VA    PA    X    Y  A  B  W  H  AL  F  SF    Z  CN  JP   VT
1241   424  30    0  1  0  1  0   0  3  17  164   4   3 3513
```

```
 CALLx, LVRD, RET, EXITT, EXGO - Calls and returns.
File: Perq.QCode.4   Perq Microcode  Page       12

! Opcode CALLL.
!-----------------------------------------------------------------------
!
! Abstract:
!       CALLL is a two byte routine call instruction.  It is used to c
!       routines in the current code segment.
!
! Instruction:
!       CALLL  RoutineNumber
!
! Result:
!       New activation record built on memory stack.
!       Code state registers saved in new ACB.
!       Expression stack saved in new ACB.
!       Code state registers updated.
!
! Calls:
!       CllSub, RefillJmp, VectSrv.
!
!-----------------------------------------------------------------------

Opcode(CALLL), tmp := NextOp;                 ! new RN
```

```
1242   425   37    4 6 0 1 0   1 3 13 307   0  3     -
1243  2070   35   11 6 0 1 0   1 3 17 236   2  1  3331
1244  2071   41    2 6 1 1 0   1 3 16 335   0 14  3272

1245  2072   36    6 6 0 1 0   1 3 11 330   0  1  3064
1246  2073    0    0 6 0 0 0   0 3 10 261   0  3  3525
```

```
        tmp7 := GP;                                  ! new GP = old GP
        tmp5 := CS, If IntrPend Call(VectSrv);       ! new CS = old CS
        tmp11 := 2, LoadS(StkOv);                    ! instruction is two byte
        tmp6 := CB,                                  ! new CB = old CB
                  Call(CllSub);                      ! set up ACB etc.
        Goto(RefillJmp);                             ! enter the routine
```



```
! Opcode CALLXB.
!-----------------------------------------------------------------------

! Abstract:
!       CALLXB is a three byte routine call instruction.  It is used t
!       routines in an external code segment.  The external segment is
!       identified by an ISN (internal segment number) which is an ind
!       into the XST (external segment table).  The XST maps an ISN in
!       an XSN (external segment number) and an XGP (external global p

! Instruction:
!       CALLXB  ISN RoutineNumber

! Result:
!       New activation record built on memory stack.
!       Code state registers saved in new ACB.
!       Expression stack saved in new ACB.
!       Code state registers updated.

! Calls:
!       CllSub, XSTMap, ChkSeg, RefillJmp, SegFault, VectSrv.

!-----------------------------------------------------------------------

! Opcode CALLXW.
!-----------------------------------------------------------------------

! Abstract:
!       CALLXW is a four byte routine call instruction.  It is used to
!       routines in an external code segment.  The external segment is
!       identified by an ISN (internal segment number) which is an ind
!       into the XST (external segment table).  The XST maps an ISN in
!       an XSN (external segment number) and an XGP (external global p

! Instruction:
!       CALLXW  LowByteISN HighByteISN RoutineNumber

! Result:
!       New activation record built on memory stack.
!       Code state registers saved in new ACB.
!       Expression stack saved in new ACB.
!       Code state registers updated.
!
```

```
                                                ! Calls:
                                                !       CllSub, XSTMap, ChkSeg, WordParm, RefillJmp, SegFault, VectSrv
                                                !
                                                !-----------------------------------------------------------------------

1247   124   30   0 1 0 1 0  0 3 17 164  4  3 3513  Opcode(CALLXB), tmp := NextOp;                  ! ISN
1250   125   41   2 6 1 1 0  1 3 13 303  0  3    -          tmp11 := 2;                             ! backup 2 bytes if segfa
1251  2074    0   0 6 0 0 0  0 3 12 115  0  1 2726  CallExternal: Call(XSTMap);
1252  2075   35  30 6 0 1 0  1 3 16 335  0 14 3272          tmp5 := tmp, LoadS(StkOv);              ! XSN
1253  2076    0   0 6 0 0 0  0 3 12 147  0  1 2671          Call(ChkSeg);                           ! be sure XSN is resident
1254  2077   36  30 6 0 1 0  1 3 11 122  6  3 3256          tmp6 := tmp, If Odd Goto(SegFault1);    ! if not resident,
1255  2100   30   0 1 0 1 0  0 3 17 164  4  3 3513          tmp := NextOp;                          ! new RN
                                                            tmp11 := tmp11 + 1,                     ! back up one more if sta
1256  2101   41   1 6 1 1 0 14 3 11 330  0  1 3064                                Call(CllSub);      ! set up ACB etc.
1257  2102    0   0 6 0 0 0  0 3 10 261  0  3 3525          Goto(RefillJmp);                        ! enter the routine
   Perq.Micro - Perq Q-Code Interpreter microcode                CALLx, LVRD, RET, EXITT, EXGO - Calls and returns.
  VA     PA    X   Y A B W H AL F SF    Z CN JP   VT    File: Perq.QCode.4   Perq Microcode  Page       14

1260   120   41   3 6 1 1 0  1 3 10 255  0  1 3541  Opcode(CALLXW), tmp11 := 3, Call(WordParm);  ! backup 3 bytes if segfa
1261   121   30  30 0 0 1 0 14 3 13 303  0  3 1251          tmp := Shift + tmp, Goto(CallExternal);  ! ISN


   Perq.Micro - Perq Q-Code Interpreter microcode               CALLx, LVRD, RET, EXITT, EXGO - Calls and returns.
  VA     PA    X   Y A B W H AL F SF    Z CN JP   VT    File: Perq.QCode.4   Perq Microcode  Page       15

                                                ! Opcode LVRD.
                                                !-----------------------------------------------------------------------
                                                !
                                                ! Abstract:
                                                !       LVRD is a five byte instruction that builds a variable routine
                                                !       descriptor.  This descriptor may be used later in a CALLV (cal
                                                !       variable) instruction.  The external segment is identified by
                                                !       ISN (internal segment number).  An ISN of zero identifies the
                                                !       current segment.  The ISN is an index into the XST (external
                                                !       segment table).  The XST maps an ISN into an XSN (external
                                                !       segment number) and an XGP (external global pointer).
                                                !
                                                ! Instruction:
                                                !       LVRD    LowByteISN HighByteISNRoutineNumber LexicalLevel
                                                !
                                                ! Result:
                                                !       Stack pushed four times.
                                                !       (Tos..Tos-3) = Variable routine descriptor:
                                                !            (Tos) = XSN = External segment number.
                                                !            (Tos-1) = XGP = Global link as offset from stack base.
                                                !            (Tos-2) = new RN = Routine number.
                                                !            (Tos-3) = new SL = Static link as offset from stack base.
                                                !
                                                ! Calls:
                                                !       WordParm, XSTMap, GetLL, GetSL.
                                                !
                                                !-----------------------------------------------------------------------

1262  1164    0   0 6 0 0 0  0 3 10 255  0  1 3541  GoLvrd: Opcode(LVRD), Call(WordParm);
```

```
1263   1165   30   30 0 0 1 0 14 3 12 115  0  1 2726          tmp := Shift + tmp, Call(XSTMap);  ! ISN
1264   1166   32    0 1 0 1 0  0 3 17 164  4  3 3513          tmp2 := NextOp;                    ! new RN
1265   1167   40    0 6 1 1 0  1 3 17 126  0  3    -         tmp10 := 0;                        ! SL of top-level routine
1266    251   33    2 1 1 1 0 16 0 10 164  4  3 3513          tmp3 := NextOp - 2;                ! new LL - 2
1267    252   34    6 6 0 1 0  1 3 15 240 11  3 1304          tmp4 := CB, if Leq Goto(LVRD2);    ! if calling top-level routine
1270    253    0    0 6 0 0 0  0 3 15 251  0  3    -         Nop;                               ! allow placer to do page esca
1271   1126   40   10 6 0 1 0  1 3 12 133  0  1 2707          tmp10 := RN, Call(GetLL);          ! get current LL
1272   1127   33    2 6 1 1 0 14 3 15 274  0  3    -         tmp3 := tmp3 + 2;                  ! new LL
1273   1103   31   33 3 0 1 0 16 3 15 242  0  3    -         tmp1 := Mdi - tmp3;                ! current LL - new LL (typical
1274   1135   40    3 6 0 1 0  1 3 12 121 17  1 2722          tmp10 := AP, if Geq Call(GetSL);   ! if not calling deeper
                                                                                              ! if calling deeper, new SL =
1275   1136   40    7 6 0 0 0 16 0  4 244  0  3    -  LVRD1:  tmp10 - SB, Push;                 ! static link
1276   1133   32    0 6 0 0 0  0 0  4 245  0  3    -         tmp2, Push;                        ! routine number
1277   1132   37    7 6 0 0 0 16 0  4 260  0  3    -         tmp7 - SB, Push;                   ! global link
1300   1117   30    0 6 0 0 0  0 0  4 264  0  3    -         tmp, Push;                         ! system segment number
1301   1113  150    1 6 1 0 0  4 3 15 236  0  3    -         CR0 and 1;
1302   1141    0    0 6 0 0 0  0 0  0   0 10 12    -         If Neq Return;                     ! If CR0 is zero, normal instr
1303   1142    0    0 6 0 0 0  0 0  0 377  0  2    -         NextInst(0);                       ! Else it was a sub-routine ca

1304   1137   40    0 6 1 1 0  1 3 15 241  0  3 1275  LVRD2:  tmp10 := 0, Goto(LVRD1);
```



```
                                                ! Opcode CALLV.
                                                !-----------------------------------------------------------------
                                                !
                                                ! Abstract:
                                                !       CALLV is a one byte routine call instruction.  It is used to c
                                                !       routines described by variable routine descriptors.
                                                !
                                                ! Instruction:
                                                !       CALLV
                                                !
                                                ! Environment:
                                                !       (Tos..Tos-3) = Variable routine descriptor:
                                                !            (Tos) = XSN = External segment number.
                                                !            (Tos-1) = XGP = Global link as offset from stack base.
                                                !            (Tos-2) = new RN = Routine number.
                                                !            (Tos-3) = new SL = Static link as offset from stack base.
                                                !
                                                ! Result:
                                                !       New activation record built on memory stack.
                                                !       Code state registers saved in new ACB.
                                                !       Expression stack saved in new ACB after popping four words.
                                                !       Code state registers updated.
                                                !
                                                ! Calls:
                                                !       CllV, ChkSeg, RefillJmp, SegFault.
                                                !
                                                !-----------------------------------------------------------------

1305   420   30    0 7 0 1 0  0 3 12 147  0  1 2671  GoCallV:Opcode(CALLV), tmp := Tos, Call(ChkSeg);     ! XSN
```

```
1306   421   35    0 7 0 1 0   0 3  15 234   6  3 1323
1307   422   36   30 6 0 1 0   1 0   5 325   0 14 1320
1310   423    0    0 6 0 0 0   0 3  15 226   0  3    -
1311  1151   37    7 7 0 1 0  14 0   5 230   0  3    -
1312  1147   36    0 6 0 0 0   0 1  16 232   0  3    -
1313  1145   30    0 7 0 1 1   0 2   5 300   0 16    -
1314  1146   31   36 0 0 1 1  14 3  15 206   0  3    -
1315  1171    0   31 3 0 0 1  14 1  12 215   0  3    -

1316  1162   40    7 7 0 1 0  14 0   5 216   0  3 3076
1317  1163    0    0 6 0 0 0   0 3  10 261   0  3 3525


1320   452   30    0 6 0 0 0   0 0   4 330   0  3    -
1321   447    4    7 6 0 0 0  16 0   4 331   0  3    -
1322   446   11    0 6 0 0 0   0 0   4 335   0  3 3272


1323  1143   41    1 6 1 1 0   1 3  11 122   0  3 3256
```

```
        tmp5 := Tos,                                    ! new CS
                        if Odd Goto(CallV2);            ! if segment not resident
        tmp6 := tmp, Pop, LoadS(CallV1);                ! new CB
        nop;                                            ! *** separate "Pop" from
        tmp7 := Tos + SB, Pop;                          ! new GP
        tmp6, Fetch;                                    ! get routine dictionary
        Hold, tmp := Tos, Pop, LeftShift(3);            ! new RN
        Hold, tmp1 := Shift + tmp6;                     ! offset to dictionary en
        Hold, Mdi + tmp1, Fetch4;                       ! fetch dictionary entry
        tmp10 := Tos + SB, Pop,                         ! static link
                                        Call(CllV);     ! set up ACB etc.
        Goto(RefillJmp);                                ! enter the routine

!       Stack overflow.

CallV1: Tos := tmp, Push;                               ! restore expression stac
        Tos := GP - SB, Push;
        Tos := CS, Push, Goto(StkOv);

!       Segment fault.

CallV2: tmp11 := 1, Goto(SegFault1);                    ! instruction is one byte
```



```
! Opcode RET.
!-------------------------------------------------------------------------
!
! Abstract:
!       RET is a one byte instruction used to return from a routine.
!       the return address from the ACB is zero, the program counter i
!       set to the exit point of the routine that is being returned to
!       This is used by the EXITT and EXGO opcodes.
!
! Instruction:
!       RET
!
! Result:
!       Code state registers restored from old ACB.
!       Expression stack restored from old ACB.
!       Old activation record popped from memory stack.
!       Function result (if any) left on top of memory stack.
!
! Calls:
!       RetExit, RefillJmp, VectSrv, RestoreStack.
!
!-------------------------------------------------------------------------

(*****
Opcode(RET), AP + ACBRS, Fetch;
        Mdi - CS;
        tmp5 := Mdi, if Neq Goto(Return4);  ! if returning to another
```

```
Return1: Call(RestoreStack);                    ! restore expression stack
        AP + ACBGL, Fetch;
        GP := Mdi + SB, if IntrPend Call(VectSrv);  ! global pointer
        AP + ACBTL, Fetch;
        TP := Mdi + SB;                         ! top pointer
        AP + ACBRR, Fetch;
        RN := Mdi;                              ! routine number
        AP + ACBRA, Fetch;
        tmp := Mdi, RightShift(1);              ! return address
        UPC := Shift and not 3, if Eql Goto(Return2);  ! if return add
        UPC := UPC + CB, if IntrPend Call(VectSrv);

! ----> The following line is a good place to set a breakpoint.  RN, C
! ----> UPC have been restored (though BPC hasn't).

        BPC := tmp and 7, Goto(Return3);

!       In the middle of an exit sequence.

Return2: Call(RetExit);

Return3: AP + ACBDL, Fetch;
        AP := Mdi + SB;                         ! activation pointer
        AP + ACBLP,Fetch;
        LP := Mdi + SB,                         ! local pointer
                        Goto(RefillJmp);        ! enter routine

!       Cross segment return.
 CALLx, LVRD, RET, EXITT, EXGO - Calls and returns.
File: Perq.QCode.4    Perq Microcode  Page       18

Return4: tmp5 + tmp5, Fetch2;                   ! check residence
        tmp1 := CB;                             ! save old CB just in case
        CB := Mdi and not 376;
        CB := Mdx or CB, if Odd Goto(Return5);  ! if not resident
        CS := tmp5, Goto(Return1);              ! code segment

!       Segment fault.

Return5: CB := tmp1;                            ! restore CB
        tmp11 := 1, Goto(SegFault1);
*****)
Opcode(RET), AP + 4, Fetch4;                    !       t3
        Tos := Mdi, Push;                       ! TL    t1, 2
        tmp5 := Mdi;                            ! RS    t3
        tmp6 := Mdi;                            ! RA    t0
        tmp7 := Mdi;                            ! RR    t1
        tmp5 - CS;                              !       t2
        tmp10 := CB, if Neq Goto(Return6);      !       t3 if cross segment
Return1: TP := Tos + SB,                        !       t0
                        if IntrPend Call(VectSrv);
        RN := tmp7;                             !       t1
        AP + 2, Fetch2;                         !       t2, 3
        tmp4 := AP, Pop;                        !       t0 old AP
```

```
Perq.Micro - Perq Q-Code interpreter microcode
  VA     PA     X    Y A B W H AL F SF    Z CN JP   VT
```

```
1324    334    3    4 6 1 0 0 14 1 12 311  0  3    -
1325     66    0    0 3 0 0 0  0 0  4 314  0  3    -
1326     63   35    0 3 0 1 0  0 3 15 205  0  3    -
1327   1172   36    0 3 0 1 0  0 3 15 204  0  3    -
1330   1173   37    0 3 0 1 0  0 3 15 202  0  3    -
1331   1175   35   11 6 0 0 0 16 3 15 176  0  3    -
1332   1201   40    6 6 0 1 0  1 3 15 116 10  3 1366

1333   1202   17    7 7 0 1 0 14 3 17 236  2  1 3331
1334   1203   10   37 6 0 1 0  1 3 15 171  0  3    -
1335   1206    3    2 6 1 0 0 14 1 14 172  0  3    -
1336   1205   34    3 6 0 1 0  1 0  5 200  0  3    -
```

```
1337  1177  36   7 6 1 0 0  4 0 13 201  0  3    -
1340  1176   3   7 3 0 1 0 14 3 15 170  0  3    -
1341  1207   4   7 3 0 1 0 14 3 15 166  0  3    -
1342  1211  36   0 6 0 0 0  0 2  0 341  0 16    -

1343  1212  16   3 0 1 1 0  5 3 15 142 15  3 1345

1344  1213  16   6 6 0 1 0 14 3 15 141  0  3 1346


1345  1235   0   0 6 0 0 0  0 3 11 207  0  1 3164

1346  1236   3   1 6 1 0 0 14 1 16 160  0  3    -
1347  1217   5   7 3 0 1 0 14 3 15 140  0  3    -
1350  1237  34  11 6 1 0 0 14 1 16 144  0  3    -
1351  1233  34  12 6 1 1 0 14 3 15 136  0  3    -
1352  1241  33   0 3 0 1 0  0 3 15 135  0  3    -

1353  1242  34  33 6 0 1 0 14 3 15 121 15  3 1361
1354  1243 151  34 6 0 1 0  1 3 15 122  0  3    -

1355  1255  34   1 6 1 1 0 16 1 16 126  0  3    -

1356  1251  33   1 6 1 1 0 16 0  4 132  2  3 1364
1357  1252   0   0 3 0 0 0  0 0  3 122 16  3 1355

1360  1253  34 151 6 0 1 0  1 3 15 121  0  3    -
1361  1256  34   0 6 0 0 0  0 1 16 130  0  3    -
1362  1247 151   0 3 0 1 0  0 3 15 120  0  3    -

1363  1257   0   0 6 0 0 0  0 3 10 261  0  3 3525
```

```
                BPC := tmp6 and 7;                          !       t1
                AP := Md1 + SB;                             ! DL    t2
                GP := Md1 + SB;                             ! GL    t3
                tmp6, RightShift(1);                        !       t0
                UPC := Shift and not 3,                     !       t1
                                        if Eql Goto(Return2);
                UPC := UPC + CB,                            !       t2
                                        Goto(Return3);

        !       In the middle of an exit sequence.

Return2: Call(RetExit);

Return3: AP + ACBLP, Fetch;                                 !       t3
                LP := Md1 + SB;                             !       t0, 1, 2
                tmp4 + ACBStackSize, Fetch;                 !       t3
                tmp4 := tmp4 + ACBSaveStack;                !       t0
                tmp3 := Md1;                                !       t1 - t2  saved word

                tmp4 := tmp4 + tmp3,                        !       t3 address+1 of las
                                        if Eql Goto(ResetNote);  ! if stack empty
                Note1 := tmp4;                              !       else keep address f

Return4: tmp4 := tmp4 - 1, Fetch;                           !       t3
                tmp3 := tmp3 - 1, Push,                     !       t0
                                        if IntrPend Goto(Return5);
                Tos := Md1, if Gtr Goto(Return4);           !       t1

                tmp4 := Note1;                              !       copy address of spe
ResetNote:  tmp4, Fetch;                                    !       restore the special
                Note1 := MDI;                               !

                Goto(RefillJmp);                            !       And jump
```



```
1364  1245  34   1 6 1 1 0 14 3 17 236  0  1 3331
1365  1246  33   1 6 1 1 0 14 0  5 122  0  3 1355


1366  1261  30  35 6 0 1 0  1 3 12 147  0  1 2671
1367  1262   6  30 6 0 1 0  1 3 15 101  6  3 1371
1370  1263  11  35 6 0 1 0  1 3 15 175  0  3 1333
```

```
        !       Interrupt while restoring the stack.

Return5: tmp4 := tmp4 + 1, Call(VectSrv);
        Pop, tmp3 := tmp3 + 1, Goto(Return4);

        !       Cross segment return.

(*****
Return6: tmp5 + tmp5, Fetch2;                               ! check residence
                CB := Md1 and not 376;
                CB := Mdx or CB, if Odd Goto(Return7);  ! if not resident
                CS := tmp5, Goto(Return1);                  !code segment
*****)
Return6: tmp := tmp5, Call(ChkSeg);                        ! find new code base
                CB := tmp, if Odd Goto(Return7);            ! if not resident
                CS := tmp5, Goto(Return1);                  ! code segment

        !       Segment fault.
```

```
1371  1276   6   40 6 0 1 0   1 0  5 102  0  3    -  Return7: CB := tmp10, Pop;                    ! restore CB
1372  1275  41    1 6 1 1 0   1 3 11 122  0  3 3256          tmp11 := 1, Goto(SegFault1);
```



```
                                                ! Opcode EXITT.
                                                !-----------------------------------------------------------------
                                                !
                                                ! Abstract:
                                                !       EXITT is a four byte instruction used to exit from a routine.
                                                !       routine to exit from is identified in the same way as an exter
                                                !       procedure is identified in the CALLXW instruction.
                                                !
                                                !       The PC is set to the exit point of the current procedure.  The
                                                !       ACBs up to and including the one that returns to the target ro
                                                !       are modified so that the return address is zero.  Note that th
                                                !       RET opcode treats a return address of zero as an exit request.
                                                !
                                                ! Instruction:
                                                !       EXITT   LowByteISN HighByteISN RoutineNumber
                                                !
                                                ! Result:
                                                !       PC (UPC, BPC) = Exit point of current procedure.
                                                !       ACBs modified as described in abstract.
                                                !
                                                ! Calls:
                                                !       ExSub, RetExit, RefillJmp.
                                                !
                                                !-----------------------------------------------------------------
1373  1214   0    0 6 0 0 0   0 3 11 201  0  1 3174  Opcode(EXITT), Call(ExSub);          ! set return addresses to
1374  1215   0    0 6 0 0 0   0 3 11 207  0  1 3164         Call(RetExit);                 ! set PC to exit point
1375  1216   0    0 6 0 0 0   0 3 10 261  0  3 3525         Goto(RefillJmp);               ! continue at exit addres
```



```
                                                ! Opcode EXGO.
                                                !-----------------------------------------------------------------
                                                !
                                                ! Abstract:
                                                !       EXGO is a six byte instruction used to jump to a particular po
                                                !       in another routine.  The target routine is identified in the s
                                                !       way as an external procedure is identified in the CALLXW instr
                                                !       The target address is specified as an absolute byte address wi
                                                !       the target code segment--it is not self-relative.
                                                !
                                                !       The PC is set to the exit point of the current procedure.  The
                                                !       ACBs up to but not including the one that returns to the targe
                                                !       routine are modified so that the return address is zero.  Note
```

```
!       that the RET opcode treats a return address of zero as an exit
!       request.  The ACB that returns to the target routine has its
!       return address set to the target address.
!
! ----> The assumption is made that the segment and routine numbers do
!       specify the current procedure.
!
! Instruction:
!       EXGO    LowByteISN HighByteISN;
!               RoutineNumber
!               LowByteAddress HighByteAddress
!
! Result:
!       PC (UPC, BPC) = Exit point of current procedure.
!       ACBs modified as described in abstract.
!
! Calls:
!       ExSub, RetExit, RefillJmp, WordParm.
!-------------------------------------------------------------------------

Opcode(EXGO), Call(ExSub);                      ! set return addresses to
        Call(WordParm);
        tmp := Shift + tmp;                     ! target address
        tmp2 + ACBRA, Store;                    ! store in last ACB
        tmp, Call(RetExit);                     ! set PC to exit point
        Goto(RefillJmp);                        ! continue at exit addres

 CALLx, LVRD, RET, EXITT, EXGO - Calls and returns.
File: Perq.Micro   Perq Microcode  Page      22

$NoList
 Floating point arithmetic operators.
File: Perq.Micro   Perq Microcode  Page      23

$List
 Floating point arithmetic operators.
File: Perq.Routine.1   Perq Microcode  Page      24

$Include Perq.Routine.1

! 13 Jan 82  WJH       change , to ; in StkOv


$Title Addressing routines.


 Addressing routines.
File: Perq.Routine.1   Perq Microcode  Page      25

! Routine ChkSeg.
!-------------------------------------------------------------------------
!
! Abstract:
```

```
1376   1610   0    0 6 0 0 0   0 3 11 201   0   1 3174
1377   1611   0    0 6 0 0 0   0 3 10 255   0   1 3541
1400   1612  30   30 0 0 1 0  14 3 15  74   0   3    -
1401   1303  32    6 6 1 0 0  14 1 17  76   0   3    -
1402   1301  30    0 6 0 0 0   0 3 11 207   0   1 3164
1403   1302   0    0 6 0 0 0   0 3 10 261   0   3 3525

 Perq.Micro - Perq Q-Code interpreter microcode
VA    PA    X    Y A B W H AL F SF    Z CN JP    VT

 Perq.Micro - Perq Q-Code interpreter microcode
VA    PA    X    Y A B W H AL F SF    Z CN JP    VT

 Perq.Micro - Perq Q-Code interpreter microcode
VA    PA    X    Y A B W H AL F SF    Z CN JP    VT

 Perq.Micro - Perq Q-Code interpreter microcode
VA    PA    X    Y A B W H AL F SF    Z CN JP    VT
```

```
                                        !       ChkSeg checks a segment number for residency and returns its b
                                        !       address (if resident).
                                        !
                                        ! Environment:
                                        !       tmp = Segment number.
                                        !
                                        ! Result:
                                        !       tmp = Base address.
                                        !       tmp and ALU result are Odd if non-resident.
                                        !
                                        !-----------------------------------------------------------------

                                        (*****
                                        ChkSeg: tmp + tmp, Fetch2;                      ! fetch segment table ent
                                                tmp := Mdi and not 376;
                                                tmp := Mdx or tmp, Return;              ! base address and reside
                                        *****)
2671  2630  30   30 6 0 0 0 14 1 14 150  0 3    -   ChkSeg: tmp + tmp, Fetch2;                      ! fetch segment table ent
2672  2627  42   30 6 0 1 0  1 3 12 144  0 3    -           tmp12 := tmp;
2673  2633  30    0 3 0 1 0  0 0  4 146) 0 3    -           Tos := tmp := Mdi, Push;
                                                tmp := Mdx or tmp,                      ! base address and flags
2674  2631  30   30 4 0 1 0  7 3 12 143  6 3 2677                    if Odd Goto(ChkSeg1);  ! if not resident
2675  2632  30  377 6 1 1 0  5 3 12 142  0 3    -           tmp := tmp and not 377;         ! clear the flags
2676  2635  42   42 6 0 0 0 14 1 17 143  0 3    -           tmp12 + tmp12, Store;
2677  2634  0     4 7 1 0 0  7 0  5   0  0 12   -   ChkSeg1: Tos or 4, Pop, Return;          ! set RecentlyUsed in SAT

  Perq.Micro - Perq Q-Code interpreter microcode  Addressing routines.
VA     PA    X    Y A B W H AL F SF   Z CN JP   VT  File: Perq.Routine.1   Perq Microcode  Page       26

                                        ! Routine GetGP.
                                        !-----------------------------------------------------------------
                                        !
                                        ! Abstract:
                                        !       Get the global pointer for an external segment.  The input is
                                        !       internal segment number, and the output is a global pointer.
                                        !       An internal segment number of zero is used to mean the current
                                        !       segment.
                                        !
                                        ! Environment:
                                        !       tmp2 = Internal segment number.
                                        !
                                        ! Result:
                                        !       tmp2 = External global pointer.
                                        !
                                        ! Calls:
                                        !       VectSrv.
                                        !
                                        !-----------------------------------------------------------------

2700  2636   0    0 6 0 0 0  0 3 17 236  0  1 3331  GetGP2: Call(VectSrv);                      ! serve an interrupt
2701  2637   0    0 6 0 0 0  0 3 17 372  0  3   -           Nop;                            ! let placer make two gro
2702     5  32    0 6 0 0 0  0 3 12 141  2  3 2700  GetGP:  tmp2, if IntrPend Goto(GetGP2);
2703     6  32   32 6 0 1 0 14 3 12 136 10  3 2705          tmp2 := tmp2 + tmp2, if Neg Goto(GetGP1);
```

```
2704     7  32   4 6 0 1 0  1 0  0   0  0 12   -          tmp2 := GP, Return;                    ! same segment as current
                                                 !        Other segment.
2705  2641   4  32 6 0 0 0 16 1 16 137  0  3   -  GetGP1: GP - tmp2, Fetch;
2706  2640  32   7 3 0 1 0 14 0  0   0  0 12   -          tmp2 := Mdi + SB, Return;
```



```
                                                 ! Routine GetLL.
                                                 !-----------------------------------------------------------------------
                                                 !
                                                 ! Abstract:
                                                 !       GetLL gets the lexical level of a routine given its number and
                                                 !       base address.  The code segment must be resident.
                                                 !
                                                 ! Environment:
                                                 !       tmp4 = Code base address as a physical address.
                                                 !       tmp10 = Routine number.
                                                 !
                                                 ! Result:
                                                 !       tmp1 = Address of LL field.
                                                 !       LL word of the routine descriptor fetched--it may be read on M
                                                 !
                                                 ! Calls:
                                                 !       VectSrv.
                                                 !
                                                 !-----------------------------------------------------------------------
2707  2644  34   0 6 0 0 0  0 1 16 135  0  3   -  GetLL:  tmp4, Fetch;                          ! fetch dictionary pointe
2710  2642  40   0 6 0 0 0  0 2  0 300  0 16   -         tmp10, LeftShift(3);                   ! multiply routine number
2711  2643  31  34 0 0 1 0 14 3 12 132  0  3   -         tmp1 := Shift + tmp4;                  ! offset of dictionary en
2712  2645  31   5 6 1 1 0 14 3 12 131  0  3   -         tmp1 := tmp1 + RDLL;                   ! offset of LL field
2713  2646  31  31 3 0 1 0 14 3 17 236  2  1 3331       tmp1 := Mdi + tmp1, if IntrPend Call(VectSrv);  ! address of L
2714  2647  31   0 6 0 0 0  0 1 16   0  0 12   -         tmp1, Fetch, Return;                   ! fetch LL field
```



```
                                                 ! Routine GetLP.
                                                 !-----------------------------------------------------------------------
                                                 !
                                                 ! Abstract:
                                                 !       Get the local pointer for another activation record.  The inpu
                                                 !       is an offset in static nesting, and the output is a local poin
                                                 !
                                                 ! Environment:
                                                 !       tmp2 = AP.
                                                 !       tmp3 = offset in static nesting.
                                                 !
                                                 ! Result:
```

```
                                        !       The memory word containing the desired local pointer is Fetche
                                        !       in the instruction which returns.
                                        !
                                        ! Calls:
                                        !       VectSrv.
                                        !-------------------------------------------------------------------

2715  2653  33   1 6 1 1 0 14 3 17 236  0  1 3831  GetLP1: tmp3 := tmp3 + 1, Call(VectSrv);
2716  2654  32   0 6 0 0 0  0 1 16 127  0  3    -  GetLP:  tmp2, Fetch;                    ! fetch static link
2717  2650  33   1 6 1 1 0 16 3 12 124  2  3 2715          tmp3 := tmp3 - 1, if IntrPend Goto(GetLP1);
                                                          tmp2 := Mdi + SB,                ! AP for next activation
2720  2651  32   7 3 0 1 0 14 3 12 123 16  3 2716                          if Gtr Goto(GetLP);  ! if not there yet
2721  2652  32   1 6 1 0 0 14 1 16   0  0 12    -                 tmp2 + ACBLP, Fetch, Return;        ! fetch desired LP
```



```
                                        ! Routine GetSL.
                                        !-------------------------------------------------------------------
                                        !
                                        ! Abstract:
                                        !       GetSL gets the static link of a procedure that is n lexical le
                                        !       away.  Thus specifying zero gets the static link for a procedu
                                        !       that is at the same level as the starting one.
                                        !
                                        ! Environment:
                                        !       tmp10 = Starting AP.
                                        !
                                        ! Result:
                                        !       tmp10 = Desired SL.
                                        !
                                        ! Calls:
                                        !       VectSrv.
                                        !-------------------------------------------------------------------

2722  2656  40   0 6 0 0 0  0 1 16 122  0  3    -  GetSL:  tmp10, Fetch;                   ! get next static link
2723  2655  31   1 6 1 1 0 16 3 12 120  0  3    -         tmp1 := tmp1 - 1;
                                                          tmp10 := Mdi + SB,               ! static link
2724  2657  40   7 3 0 1 0 14 3 12 121 17  3 2722                          if Geq Goto(GetSL);  ! if not there yet
2725  2660   0   0 6 0 0 0  0 0  0   0  0 12    -                 Return;
```



```
                                        ! Routine XSTMap.
                                        !-------------------------------------------------------------------
                                        !
                                        ! Abstract:
                                        !       XstMap maps an internal segment number (ISN) into an external
                                        !       segment number (SSN) and an external global pointer (XGP).  An
```

```
                                        !       ISN of zero is taken to mean the current segment.
                                        !
                                        ! Environment:
                                        !       tmp = ISN.
                                        !
                                        ! Result:
                                        !       tmp = XSN.
                                        !       tmp7 = XGP.
                                        !
                                        ! Calls:
                                        !       VectSrv.
                                        !
                                        !-------------------------------------------------------------------------

2726  2662  30   0 6 0 0 0  0 2  0 340  0 16    -  XSTMap: tmp, LeftShift(1);
2727  2663  30   0 0 0 1 0  0 3 12 111 15  3 2733         tmp := Shift, if Eql Goto(XSTMap1);  ! if current segment desi
2730  2664   4  30 6 0 0 0 16 1 14 116  0  3    -         GP - tmp, Fetch2;                    ! fetch XST entry
2731  2661  37   7 3 0 1 0 14 3 12 112  0  3    -         tmp7 := Md1 + SB;                    ! XGP
2732  2665  30   0 3 0 1 0  0 0  0   0  0 12    -         tmp := Md1, Return;                  ! SSN

                                        !       ISN = 0 means XSN = CS, XGP = GP.

2733  2666  30  11 6 0 1 0  1 3 17 236  2  1 3331  XSTMap1:  tmp := CS, if IntrPend Call(VectSrv);
2734  2667  37   4 6 0 1 0  1 0  0   0  0 12    -         tmp7 := GP, Return;


                                        $Title Boolean routines.


  Perq.Micro - Perq Q-Code interpreter microcode   Boolean routines.
VA    PA    X    Y A B W H AL F SF    Z CN JP   VT  File: Perq.Routine.1   Perq Microcode  Page       31

                                        ! Routine SetFalse.
                                        !-------------------------------------------------------------------------
                                        !
                                        ! Abstract:
                                        !       SetFalse sets the top of the expression stack to false.  SetFa
                                        !       is not called, it is jumped to.  It exits via a NextInst.
                                        !
                                        ! Environment:
                                        !       (Tos) = Anything.
                                        !
                                        ! Result:
                                        !       (Tos) = False.
                                        !
                                        !-------------------------------------------------------------------------

2735  2670   0   0 6 1 0 0  1 0  3 377  0  2    -  SetFalse: Tos := 0, NextInst(0);

  Perq.Micro - Perq Q-Code interpreter microcode   Boolean routines.
VA    PA    X    Y A B W H AL F SF    Z CN JP   VT  File: Perq.Routine.1   Perq Microcode  Page -    32
```

```
                                        ! Routine SetTrue.
                                        !-----------------------------------------------------------------------
                                        !
                                        ! Abstract:
                                        !       SetTrue sets the top of the expression stack to false.  SetTru
                                        !       is not called, it is jumped to.  It exits via a NextInst.
                                        !
                                        ! Environment:
                                        !       (Tos) = Anything.
                                        !
                                        ! Result:
                                        !       (Tos) = True.
                                        !
                                        !-----------------------------------------------------------------------

2736  1516   0   1 6 1 0 0  1 0  3 377  0  2    -  SetTrue: Tos := 1, NextInst(0);


                                        $Title Byte array and string routines.



                                        ! Routine GetStringIndex.
                                        !-----------------------------------------------------------------------
                                        !
                                        ! Abstract:
                                        !       GetStringIndex gets an index into a string variable and checks
                                        !       against the dynamic length of the string.  If the index is out
                                        !       range, GetStringIndex causes an ErrInxCase error.  If the inde
                                        !       in range, GetStringIndex fetches the correct word and returns.
                                        !
                                        ! Environment:
                                        !       (Tos) = Byte offset.
                                        !       (Tos-1) = Word address as offset from stack base.
                                        !
                                        ! Result:
                                        !       Stack popped.
                                        !       tmp4 = Physical word address of the beginning of the string.
                                        !       tmp1 = Word offset within the string.
                                        !       Word containing character fetched and readable on Md1.
                                        !
                                        ! Calls:
                                        !       ChkOvr, VectSrv..
                                        !
                                        !-----------------------------------------------------------------------

2737  1475  30   60 7 0 1 0   4 2  5 341   0 16    -  GetStringIndex: tmp := Tos and AllOnes, Pop, RightShift(1);      ! by
2740  1476  31    0 0 0 1 0   0 3 17 236   2  1 3331        tmp1 := Shift, if IntrPend Call(VectSrv);! word offset
2741  1477   0    7 7 0 0 0  14 1 16 364   0  3    -         Tos + SB, Fetch;
```

```
2742  1413  34 377 3 1 1 0  4 3 12 106  0  3    -
2743  2671  34  30 6 0 0 0 16 3 12 105  0  3    -
2744  2672  34   7 7 0 1 0 14 3 11 157 12  3 3216
2745  2673  34  31 6 0 0 0 14 1 16   0  0 12    -
```

```
        tmp4 := Mdi and 377;
        tmp4 - tmp;
        tmp4 := Tos + SB, if Lss Goto(ChkOvr);
        tmp4 + tmp1, Fetch, Return;
```



```
! Routine GetSrcDst.
!-----------------------------------------------------------------------
!
! Abstract:
!       GetSrcDst gets a source and destination byte pointer from the
!       expression stack.  It also pre-fetches the first bytes referen
!       by the two byte pointers.  This is done to get the length byte
!       for string operations.  The result conditions of GetSrcDst are
!       precisely the evironment for GetSrc, GetDst, and PutDst.
!
! Environment:
!       (Tos) = Source byte offset.
!       (Tos-1) = Source address as offset from stack base.
!       (Tos-2) = Destination byte offset.
!       (Tos-3) = Destination address as offset from stack base.
!
! Result:
!       Stack popped four times.
!       Src = Source word physical address (Address + ByteOffset div 2
!       SrcLsb = Least significant bit of the source byte address.
!       SrcWord = First source word.
!       SrcByte = First source byte.
!       Dst = Destination word physical address (Address + ByteOffset
!       DstLsb = Least significant bit of the destination byte address
!       DstWord = First destination word.
!       DstByte = First destination byte.
!
! Calls:
!       VectSrv.
!
!-----------------------------------------------------------------------
```

```
2746  2677  33  60 7 0 1 0  4 2  5 341  0 16    -
2747  2700  54   7 0 0 1 0 14 3 17 236  2  1 3331
2750  2701  54  54 7 0 1 0 14 1 16 101  0  3    -
2751  2676  55  33 6 0 1 0  1 0  5 103  0  3    -
2752  2674   0   0 6 0 0 0  0 3 12  65  6  3 2764
2753  2675  57   0 3 0 1 0  0 3 12  75  0  3    -
2754  2702  56 377 3 1 1 0  4 3 12  71  0  3    -
2755  2706  33  60 7 0 1 0  4 2  5 341  0 16    -
2756  2707  64   7 0 0 1 0 14 3 17 236  2  1 3331
2757  2710  64  64 7 0 1 0 14 1 16  42  0  3    -
2760  2705  65  33 6 0 1 0  1 0  5  74  0  3    -
2761  2703   0   0 6 0 0 0  0 3 12  63  6  3 2766
2762  2704  67   0 3 0 1 0  0 3 12  66  0  3    -
```

```
GetSrcDst: tmp3 := Tos and AllOnes, Pop, RightShift(1);  ! get Src byt
        Src := Shift + SB, if IntrPend Call(VectSrv);
        Src := Tos + Src, Fetch;                    ! Src word address
        SrcLsb := tmp3, Pop;                        ! Src least significant b
        if Odd Goto(GetSD2);                        ! if first byte is upper
        SrcWord := Mdi;                             ! 1st Src word
        SrcByte := Mdi and 377;                     ! 1st Src byte
GetSD1: tmp3 := Tos and AllOnes, Pop, RightShift(1);    ! get Dst byt
        Dst := Shift + SB, if IntrPend Call(VectSrv);
        Dst := Tos + Dst, Fetch;                    ! Dst word address
        DstLsb := tmp3, Pop;                        ! Dst least significant b
        if Odd Goto(GetSD3);                        ! if first byte is upper
        DstWord := Mdi;                             ! 1st Dst word
```

```
2763  2711  66 377 3 1 1 0  4 0  0   0  0 12   -        - DstByte := Mdi and 377, Return;      ! 1st Dst byte

2764  2712  57   0 3 0 1 0  0 2  0 370  0 16   -  GetSD2: SrcWord := Mdi, Field(0,10);        ! 1st Src word
2765  2713  56   0 0 0 1 0  0 3 12  71  0  3 2755        SrcByte := Shift, Goto(GetSD1);     ! 1st Src byte

2766  2714  67   0 3 0 1 0  0 2  0 370  0 16   -  GetSD3: DstWord := Mdi, Field(0,10);        ! 1st Dst word
2767  2715  66   0 0 0 1 0  0 0  0   0  0 12   -        DstByte := Shift, Return;            ! 1st Dst byte
```



```
! Routine GetSrc.
!-------------------------------------------------------------------------
!
! Abstract:
!       GetSrc gets the next source byte from a byte array or a string
!
! Environment:
!       Src = Source word physical address.
!       SrcLsb = Least significant bit of the source byte address.
!       SrcWord = Current source word.
!
! Result:
!       SrcWord = Current source word.
!       SrcByte = Current source byte.
!       Source address (Src and SrcLsb) advanced to next byte.
!
! Calls:
!       VectSrv.
!
!-------------------------------------------------------------------------

2770  2716  55   0 6 0 0 0  0 3 12  56  0  3   -  GetSrc: SrcLsb;                                  ! if upper byte
2771  2721   0   0 6 0 0 0  0 3 12  53  6  3 2776        if Odd Goto(GetSrc2);                    ! get a new word
2772  2722  54   0 6 0 0 0  0 1 16  60  0  3   -  GetSrc1: Src, Fetch;
2773  2717  55   1 6 1 1 0  1 3 12  47  2  3 3002        SrcLsb := 1, if IntrPend Goto(GetSrc3);  ! next byte is upper
2774  2720  57   0 3 0 1 0  0 3 12  54  0  3   -        SrcWord := Mdi;                          ! current word
2775  2723  56 377 3 1 1 0  4 0  0   0  0 12   -        SrcByte := Mdi and 377, Return;          ! current byte

2776  2724  57   0 6 0 0 0  0 2  0 170  0 16   -  GetSrc2: SrcWord, RightShift(10);             ! get upper byte from cur
2777  2725  56 377 0 1 1 0  4 3 12  51  0  3   -        SrcByte := Shift and 377;                ! current byte
3000  2726  54   1 6 1 1 0 14 3 17 236  2  1 3331        Src := Src + 1, if IntrPend Call(VectSrv);  ! advance to next
3001  2727  55   0 6 1 1 0  1 0  0   0  0 12   -        SrcLsb := 0, Return;                     ! next byte is lower byte

                                                !       Serve an interrupt.

3002  2730   0   0 6 0 0 0  0 3 17 236  0  1 3331  GetSrc3: Call(VectSrv);
3003  2731   0   0 6 0 0 0  0 3 12  55  0  3 2772        Goto(GetSrc1);
```

```
                                                ! Routine GetDst.
                                                !-----------------------------------------------------------------
                                                !
                                                ! Abstract:
                                                !       GetDst gets the next destination byte from a byte array or a s
                                                !
                                                ! Environment:
                                                !       Dst = Destination word physical address.
                                                !       DstLsb = Least significant bit of the destination byte address
                                                !       DstWord = Current destination word.
                                                !
                                                ! Result:
                                                !       DstWord = Current destination word with current byte removed.
                                                !       DstByte = Current destination byte.
                                                !       Destination address (Dst and DstLsb) advanced to next byte.
                                                !
                                                ! Calls:
                                                !       VectSrv.
                                                !
                                                !-----------------------------------------------------------------

3004  2732   65    0 6 0 0 0 0  0 3 12  42  0  3     -   GetDst: DstLsb;
3005  2735    0    0 6 0 0 0 0  0 3 12  37  6  3  3012           if Odd Goto(GetDst2);                ! if upper byte
3006  2736   64    0 6 0 0 0 0  0 1 16  44  0  3     -   GetDst1: Dst, Fetch;                         ! get a new word
3007  2733   65    1 6 1 1 0 0  1 3 12  32  2  3  3017           DstLsb := 1, if IntrPend Goto(GetDst3);  ! next byte is upper
3010  2734   67  377 3 1 1 0 0  5 3 12  40  0  3     -           DstWord := Mdi and not 377;          ! current word
3011  2737   66  377 3 1 1 0 0  4 0 0    0  0 12     -           DstByte := Mdi and 377, Return;      ! current byte

3012  2740   67    0 6 0 0 0 0  0 2  0 170  0 16     -   GetDst2: DstWord, RightShift(10);            ! get upper byte from cur
3013  2741   66  377 0 1 1 0 0  4 3 12  35  0  3     -           DstByte := Shift and 377;            ! current byte
3014  2742   67  377 6 1 1 0 0  4 3 12  34  0  3     -           DstWord := DstWord and 377;          ! current word
3015  2743   64    1 6 1 1 0 0 14 3 17 236  2  1  3331           Dst := Dst + 1, if IntrPend Call(VectSrv);  ! advance to next
3016  2744   65    0 6 1 1 0 0  1 0 0    0  0 12     -           DstLsb := 0, Return;                 ! next byte is lower byte

                                                !       Serve an interrupt.

3017  2745    0    0 6 0 0 0 0  0 3 17 236  0  1  3331   GetDst3: Call(VectSrv);
3020  2746    0    0 6 0 0 0 0  0 3 12  41  0  3  3006           Goto(GetDst1);
```



```
                                                ! Routine PutDst.
                                                !-----------------------------------------------------------------
                                                !
                                                ! Abstract:
                                                !       PutDst puts the next destination byte into a byte array or a s
                                                !
                                                ! Environment:
                                                !       Dst = Destination word physical address.
                                                !       DstLsb = Least significant bit of the destination byte address
                                                !       DstWord = Current destination word with current byte removed.
                                                !       SrcByte = Current source byte to be put into the destination.
```

```
!
! Result:
!       DstWord = Current destination word with source byte added.
!
! Calls:
!       VectSrv.
!
!-----------------------------------------------------------------------

3021  2747  65    0 6 0 0 0  0 3 12  26  0  3     -  PutDst: DstLsb;
3022  2751   0    0 6 0 0 0  0 3 12  22  6  3  3027         if Odd Goto(PutDst1);                ! if lower byte
3023  2752  56    0 6 0 0 0  0 2  0 160  0 16     -         SrcByte, LeftShift(10);              ! move SrcByte to upper b
                                                            DstWord := Shift or DstWord,         ! put SrcByte in upper b
                                                                       if IntrPend Call(VectSrv);
3024  2753  67   67 0 0 1 0  7 3 17 236  2  1  3331
3025  2754  64    1 6 1 0 0 16 1 17  27  0  3     -         Dst - 1, Store;
3026  2750  67    0 6 0 0 0  0 0  0   0  0 12     -         DstWord, Return;                     ! store destination word

3027  2755  67   56 6 0 1 0  7 0  0   0  0 12     -  PutDst1:DstWord := DstWord or SrcByte, Return;  ! put SrcByte in lower
```



```
! Routine BytCmp.
!-----------------------------------------------------------------------
!
! Abstract:
!       BytCmp compares two byte arrays.  After the call to BytCmp, th
!       Neq, Leq, Lss, Geq, and Gtr condition codes can be used to che
!       the results of the comparison.
!
! Environment:
!       Next byte in the opcode/operand stream = Length.
!       If Length <> 0:
!            (Tos) = Byte offset for ByteArray0.
!            (Tos-1) = Word address of ByteArray0.
!            (Tos-2) = Byte offset for ByteArray1.
!            (Tos-3) = Word address of ByteArray1.
!       If Length = 0:
!            (Tos) = Length of byte arrays.
!            (Tos-1) = Byte offset for ByteArray0.
!            (Tos-2) = Word address of ByteArray0.
!            (Tos-3) = Byte offset for ByteArray1.
!            (Tos-4) = Word address of ByteArray1.
!
! Result:
!       Length removed from opcode/operand stream.
!       If Length <> 0:
!            Stack popped four times.
!       If Length = 0:
!            Stack popped five times.
!       ALU result := ByteArray0 compared to ByteArray1:
!
! Calls:
```

```
                                                !       GetSrcDst, GetSrc, GetDst, VectSrv.
                                                !
                                                !-------------------------------------------------------------------

3030  3061  32   0 1 0 1 0  0 3 17 164  4  3 3513  BytCmp: tmp2 := NextOp;                    ! get length
3031  3062   0   0 6 0 0 0  0 3 11 321 10  3 3033          if Neq Goto(BytCmp1);              ! if non-zero length byte
3032  3063  32   0 7 0 1 0  0 0  5 321  0  3    -          tmp2 := Tos, Pop;                  ! get length from express
3033  3056   0   0 6 0 0 0  0 3 12 100  0  1 2746  BytCmp1: Call(GetSrcDst);                ! get byte pointers
3034  3057   0   0 6 0 0 0  0 3 12  17  0  3 3037          Goto(BytCmp3);                     ! enter comparison loop

                                                BytCmp2: if Lss Goto(BytCmp4),             ! if done and all equal
3035  2756  66  56 6 0 0 0 16 3 12  14 12  3 3042                                dstbyte - srcbyte;  ! compare bytes
3036  2757   0   0 6 0 0 0  0 3 12  13 10  3 3043          if Neq Goto(BytCmp5);              ! if done and not equal
3037  2760   0   0 6 0 0 0  0 3 12  61  0  1 2770  BytCmp3:Call(GetSrc);                    ! get next byte
3040  2761   0   0 6 0 0 0  0 3 12  45  0  1 3004          Call(GetDst);                      ! get next byte
3041  2762  32   1 6 1 1 0 16 3 12  21  0  3 3035          tmp2 := tmp2 - 1, Goto(BytCmp2);   ! count byte

                                                !       Byte arrays are equal.

3042  2763   0   0 6 1 0 0  1 0  0   0  0 12    -  BytCmp4: 0, Return;                     ! return Eql

                                                !       Byte arrays are not equal, return with condition codes set for
                                                !       byte comparison.

3043  2764  66  56 6 0 0 0 16 0  0   0  0 12    -  BytCmp5: dstByte - srcByte, Return;


  Perq.Micro - Perq Q-Code interpreter microcode   Byte array and string routines.
   VA    PA   X   Y A B W H AL F SF   Z CN JP   VT  File: Perq.Routine.1   Perq Microcode  Page       39

                                                ! Routine StrCmp.
                                                !-------------------------------------------------------------------
                                                !
                                                ! Abstract:
                                                !       StrCmp compares two strings.  After the call to StrCmp, the Eq
                                                !       Neq, Leq, Lss, Geq, and Gtr condition codes can be used to che
                                                !       the results of the comparison.  The strings must be word align
                                                !
                                                ! Environment:
                                                !       (Tos) = Byte offset for String0.
                                                !       (Tos-1) = Word address of String0.
                                                !       (Tos-2) = Byte offset for String1.
                                                !       (Tos-3) = Word address of String1.
                                                !
                                                ! Result:
                                                !       Stack popped three times.
                                                !       ALU result = String0 compared with String1.
                                                !
                                                ! Calls:
                                                !       GetSrcDst, GetSrc, GetDst, VectSrv.
                                                !
                                                !-------------------------------------------------------------------

3044  2765   0   0 6 0 0 0  0 3 12 100  0  1 2746  StrCmp: Call(GetSrcDst);                  ! get string pointers
```

```
3045  2766  55   1 6 1 1 0 12 3 12   10  0  3     -        SrcLsb := SrcLsb xor 1;                    ! skip length bytes
3046  2767  65   1 6 1 1 0 12 3 11  310  0  3     -        DstLsb := DstLsb xor 1;
3047  3067  66  56 6 0 0 0 16 0  4  312  0  3     -        Tos := dstByte - srcByte, Push;            ! compare lengths, save d
3050  3065  30  56 6 0 1 0  1 3 12    7 17  3  3052        tmp := srcByte, if Geq Goto(StrCmp1);      ! if src is shorter
3051  3066  30  66 6 0 1 0  1 3 12    7  0  3     -        tmp := dstByte;

                                                     !       Compare the strings, tmp is length of shorter string.

3052  2770   0   0 6 0 0 0  0 3 12    1 15  3  3060 StrCmp1: if Eql Goto(StrCmp3);                    ! if strings equal up to
3053  2771   0   0 6 0 0 0  0 3 12   61  0  1  2770        Call(GetSrc);                              ! get next character
3054  2772   0   0 6 0 0 0  0 3 12   45  0  1  3004        Call(GetDst);                              ! get next character
3055  2773  66  56 6 0 0 0 16 3 11  306  2  3  3062        dstByte - srcByte, if IntrPend Goto(StrCmp5);
3056  2774   0   0 6 0 0 0  0 3 12    0 10  3  3061 StrCmp2:if Neq Goto(StrCmp4);                    ! if characters are not e
3057  2775  30   1 6 1 1 0 16 3 12    7  0  3  3052        tmp := tmp - 1, Goto(StrCmp1);             ! count the character

                                                     !       Strings are equal up to length of shorter string, return with
                                                     !       condition codes set for length comparison.

3060  2776   0   0 7 0 0 0  0 0  0    0  0 12     -  StrCmp3: Tos, Return;

                                                     !       Strings are not equal, return with condition codes set for cha
                                                     !       comparison.

3061  2777  66  56 6 0 0 0 16 0  0    0  0 12     -  StrCmp4: dstByte - srcByte, Return;

                                                     !       Serve an interrupt.

3062  3071   0   0 6 0 0 0  0 3 17  236  0  1  3331 StrCmp5: Call(VectSrv);
3063  3072  66  56 6 0 0 0 16 3 12    3  0  3  3056        dstbyte - srcbyte, Goto(StrCmp2);
```



```
$Title Call and return routines.
```



```
! Routine CllSub.
!-------------------------------------------------------------------------
!
! Abstract:
!       CllSub is a common routine used by the call opcodes to build t
!       activation record and set up the new pointers (GP, AP, TP, etc
!
! Environment:
!       tmp = New routine number.
!       tmp5 = New code segment number.
!       tmp6 = New code base.
!       tmp7 = New global pointer.
!       tmp11 = Length of call instruction (for PCBackup).
```

```
!       S = Routine to jump to on stack overflow.
!
! Result:
!       New activation record built.
!       Code state pointers saved in ACB.
!       Expression stack saved in ACB.
!       Code state pointers updated.
!
! Calls:
!       GetLL, GetSL, SvStk, VectSrv, S.
!
!-------------------------------------------------------------------------

! Routine CllV.
!-------------------------------------------------------------------------
!
! Abstract:
!       CllV is a common routine used to make a call to a routine desc
!       by a variable routine descriptor.  CllV builds the new activat
!       record and set up the new pointers (GP, AP, TP, etc.).
!
! Environment:
!       tmp = New routine number.
!       tmp5 = New code segment number.
!       tmp6 = New code base.
!       tmp7 = New global pointer.
!       tmp10 = New static link.
!       tmp11 = Length of call instruction (for PCBackup).
!       S = Routine to jump to on stack overflow.
!       First half of routine dictionary entry fetched and readable on
!
! Result:
!       New activation record built..
!       Code state pointers saved in ACB.
!       Expression stack saved in ACB.
!       Code state pointers updated.
!
! Calls:
!       SvStk, VectSrv, S.
!
!-------------------------------------------------------------------------

CllSub:
 Call and return routines.
File: Perq.Routine.1   Perq Microcode  Page      42

!       Get the new static link.

        tmp4 := tmp6;                           ! new code base (for GetL
        tmp10 := tmp, Call(GetLL);              ! get new lexical level
        tmp2 := Mdi - 2;
        tmp3 := tmp1,                           ! save address of LL in R
                        if Leq Goto(CllSub3);   ! if calling a top-level
        tmp4 := CB;                             ! current codebase
```

```
Perq.Micro - Perq Q-Code interpreter microcode
  VA     PA     X    Y A B W H AL F SF    Z CN JP    VT

3064   3047   34   36 6 0 1 0  1 3 11 302  0  3     -
3065   3075   40   30 6 0 1 0  1 3 12 133  0  1  2707
3066   3076   32    2 3 1 1 0 16 3 11 300  0  3     -

3067   3077   33   31 6 0 1 0  1 3 11 220 11  3  3155
3070   3100   34    6 6 0 1 0  1 3 11 276  0  3     -
```

```
3071  3101  40   10 6 0 1 0   1 3 12 133  0  1 2707        tmp10 := RN, Call(GetLL);                ! get current LL
3072  3102  32    2 6 1 1 0 14 3 11 324  0  3    -         tmp2 := tmp2 + 2;                        ! new lex level
3073  3053  31   32 3 0 1 0 16 3 15 225  0  3    -         tmp1 := Md1 - tmp2;                      ! current LL - new LL
                                                                                                  ! (typically positive)
3074  1152  40    3 6 0 1 0   1 3 12 121 17  1 2722        tmp10 := AP, if Geq Call(GetSL);         ! if not calling deeper
                                                                                                  ! if calling deeper, new
3075  1153  33    5 6 1 0 0 16 1 12 216  0  3    -  C11Sub1: tmp3 - RDLL, Fetch4;                 ! get 1st half of RD entr

                                                        !       Call variable routine entry point.

3076  1161  17    1 6 1 0 0 14 0  4 220  0  3    -  C11V:    Tos := TP + 1, Push;                 ! address of new locals a
3077  1157  31    7 3 0 1 0 14 3 11 304  0  3    -          tmp1 := Md1 + SB;                    ! PS = parameter size
3100  3073  32    0 3 0 1 0  0 3 11 274  0  3    -          tmp2 := Md1;                         ! RPS = result + paramete
3101  3103  33    0 3 0 1 0  0 3 11 273  0  3    -          tmp3 := Md1;                         ! LTS = locals + temporar
3102  3104  34    0 3 0 1 0  0 3 11 272  0  3    -          tmp4 := Md1;                         ! entry point

                                                        !       Check for stack overflow

3103  3105  42   17 6 0 1 0   1 3 11 271  0  3    -          tmp12 := TP;                         ! save TP in case of over
3104  3106  17   33 7 0 1 0 14 3 11 270  0  3    -          TP := Tos + tmp3;
3105  3107  17    3 6 1 1 0 14 3 11 267  0  3    -          TP := TP + 3;                        ! quad-align new ACB
3106  3110  17    3 6 1 1 0  5 3 17 236  2  1 3331          TP := TP and not 3, if IntrPend Call(VectSrv);  ! new AP
3107  3111  17   32 6 1 1 0 14 3 11 265  0  3    -          TP := TP + ACBReserve;
3110  3112  13   17 6 0 0 0 16 3 11 263  0  3    -          SL - TP;
3111  3114  17   32 6 1 1 0 16 3 11 217  5  3 3156          TP := TP - ACBReserve, if C19 Goto(C11Sub4);  ! if stack overf

                                                        !       Build new ACB.

3112  3115   5   32 7 0 1 0 16 0  5 264  0  3    -  C11Sub2: LP := Tos - tmp2, Pop;               ! new LP = local pointer
3113  3113  32   17 6 0 1 0  1 3 11 260  0  3    -          tmp2 := TP;                          ! new AP = activation poi
3114  3117  17    0 6 0 0 0  0 1 13 261  0  3    -          TP, Store4;
3115  3116  40    7 6 0 0 0 16 3 11 257  0  3    -          tmp10 - SB;                          ! new SL = static link
3116  3120   5    7 6 0 0 0 16 3 11 256  0  3    -          LP - SB;                             ! new LP = local pointer
3117  3121   3    7 6 0 0 0 16 3 11 255  0  3    -          AP - SB;                             ! new DL = dynamic link =
3120  3122   4    7 6 0 0 0 16 3 11 254  0  3    -          GP - SB;                             ! new GL = global link =

3121  3123 150    1 6 1 0 0 16 3 11 253  0  3    -          CRO - 1;                             ! Test control register
3122  3124 150    0 6 1 1 0  1 3 11 251 10  3 3124          CRO := 0, If neq GoTo (C11Sub2A);    ! If normal, go to usual
3123  3125  43  151 6 0 1 0  1 3 11 245  0  3 3127          tmp13 := Note1, GoTo(C11Sub2B);      ! Else reset return addre

3124  3126  43   17 5 1 1 0  4 3 11 250  0  3    -  C11Sub2A: tmp13 := UState and 17;             ! Calculate return addres
3125  3127  16    6 6 0 0 0 16 2  0 340  0 16    -          UPC - CB, LeftShift(1);
3126  3130  43   43 0 0 1 0 14 3 11 245  0  3    -          tmp13 := Shift + tmp13;
3127  3132  17    4 6 1 1 0 14 1 13 246  0  3    -  C11Sub2B: TP := TP + 4, Store4;               ! Store return address in

3130  3131  42   31 6 0 0 0 16 3 11 244  0  3    -          tmp12 - tmp1;                        ! new TL = top link = old
3131  3133  11    0 6 0 0 0  0 3 11 243  0  3    -          CS;                                  ! return CS = code segmen
3132  3134  43    0 6 0 0 0  0 3 11 242  0  3    -          tmp13;                               ! RA = return address
3133  3135  10    0 6 0 0 0  0 3 11 241  0  3    -          RN;                                  ! RR = return routine num
3134  3136  10   30 6 0 1 0  1 3 11 237  0  3    -          RN := tmp;                           ! new RN = routine number
3135  3140  11   35 6 0 1 0  1 3 17 236  2  1 3331          CS. := tmp5, if IntrPend Call(VectSrv);  ! new CS = code segmen
```

```
3136  3141   17    4 6 1 1 0 14 1 17 240  0  3     -        TP := TP + 4, Store;
3137  3137    0    0 6 1 0 0  1 3 11 235  0  3     -        0;                                  ! new EP = exception poin
3140  3142    6   36 6 0 1 0  1 3 11 234  0  3     -        CB := tmp6;                         ! new CB = code base
3141  3143    3   32 6 0 1 0  1 3 11 230  0  3     -        AP := tmp2;                         ! new AP = activation poi

                                                    !       Save the expression stack.

3142  3147   30    0 6 1 1 0  3 3 17 236  2  1  3331        tmp := not 0, if IntrPend Call(VectSrv); ! -1
3143  3150   17    1 6 1 1 0 14 3 11 214  0  1  3160        TP := TP + 1, Call(SaveStack);      ! start after length word
3144  3151   17   30 6 0 0 0 16 1 17 232  0  3     -        TP - tmp, Store;
3145  3145   30    0 6 0 0 0  0 3 17 236  2  1  3331        tmp, if IntrPend Call(VectSrv);     ! store E-stack length

3146  3146   17    1 6 1 1 0 14 1 17 233  0  3     -        TP := TP + 1, Store;                ! Save Note1 (special off
3147  3144  151    0 6 0 0 0  0 3 11 225  0  3     -        Note1;
3150  3152  151    0 6 1 1 0  1 3 11 224  0  3     -        Note1 := 0;                         ! Reset Note1 to undefine

3151  3153    4   37 6 0 1 0  1 3 11 221  0  3     -        GP := tmp7;                         ! new GP = global pointer

                                                    !       Set up new PC.

                                                    ! ----> The following instruction is a good place to set a breakpoint.
                                                    ! ----> RN, and UPC have their new values, although BPC doesn't yet.

3152  3156   34    7 6 1 0 0  4 0 13 223  0  3     -        BPC := tmp4 and 7;                  ! CB is on a 256 word bou
3153  3154   34    7 6 1 0 0  5 2 10 341  0 16     -        tmp4 and not 7, RightShift(1);
3154  3155   16    6 0 0 1 0 14 0  0   0  0 12     -        UPC := Shift + CB, Return;          ! quad program counter

                                                    !       Set SL for top-level routines: LL <= 2.

3155  3157   40    7 6 0 1 0  1 3 15 224  0  3  3075  CllSub3: tmp10 := SB, Goto(CllSub1);     ! static link is not used

                                                    !       Signal a stack overflow.  Restore Both stacks.

3156  3160  150    0 6 1 1 0  1 3 11 216  0  3     -  CllSub4: CRO := 0;                       ! Make sure abort goes ok
3157  3161   17   42 6 0 1 0  1 0  5   0  1  7     -        TP := tmp12, Pop, GotoS;
```



```
                                                    ! Routine SaveStack.
                                                    !-----------------------------------------------------------------------
                                                    !
                                                    ! Abstract:
                                                    !       SaveStack saves the expression stack in the activation control
                                                    !
                                                    ! Environment:
                                                    !       TP = First word address of the saved stack (length word + 1).
                                                    !       tmp = -1.
                                                    !
                                                    ! Result:
                                                    !       Expression stack pushed onto memory stack.
                                                    !       tmp = Number of saved words.
                                                    !
```

```
3160  3163   0   0 5 1 0 0  4 0  0   2  0 16   -
3161  3164  30   1 6 1 1 0 14 0  0   0 15 12   -
3162  3165  17   1 6 1 1 0 14 1 17 215  0  3   -
3163  3162   0   0 7 0 0 0  0 0  5 214  0  3 3160
```

```
 Perq.Micro - Perq Q-Code interpreter microcode   Call and return routines.
VA    PA   X   Y A B W H AL F SF   Z CN JP   VT  File: Perq.Routine.1   Perq Microcode  Page       45
```

```
!-------------------------------------------------------------------------
SaveStack:  UState and 1000;                    ! extract StackEmpty flel
        tmp := tmp + 1, if Eql Return;          ! bottom of stack reached
        TP := TP + 1, Store;
        Tos, Pop, Goto(SaveStack);              ! store a word
(*****
```

```
! Routine RestoreStack.
!-------------------------------------------------------------------------
!
! Abstract:
!       RestoreStack restores the expression stack from the ACB prior
!       returning from a routine.
!
! Environment:
!       Expression stack empty.
!
! Result:
!       Expression stack restored.
!
! Calls:
!       VectSrv.
!
!-------------------------------------------------------------------------

RestoreStack:AP + ACBStackSize, Fetch;
        tmp1 := AP;
        tmp1 := tmp1 + ACBSaveStack;
        tmp2 := Mdi;                            ! number of saved words o
        tmp1 := tmp1 + tmp2, if Eql Return;     ! if no words to restore
Restore1: tmp1 := tmp1 - 1, Fetch;
        tmp2 := tmp2 - 1, Push, if IntrPend Goto(Restore2);
        Tos := Mdi, if Gtr Goto(Restore1);      ! if more words to restor
        Return;

!       Serve an interrupt.

Restore2: tmp1 := tmp1 + 1, Call(VectSrv);
        Pop, tmp2 := tmp2 + 1, Goto(Restore1);
*****)
```

```
 Perq.Micro - Perq Q-Code interpreter microcode   Call and return routines.
VA    PA   X   Y A B W H AL F SF   Z CN JP   VT  File: Perq.Routine.1   Perq Microcode  Page       46
```

```
! Routine RetExit.
!-------------------------------------------------------------------------
!
! Abstract:
```

```
!       RetExit is used when a RETURN instruction finds that the retur
!       address is zero.  This means that the return address is the ex
!       point of the routine.
!
! Result:
!       PC set to exit point of current routine.
!
! Calls:
!       VectSrv.
!
!-------------------------------------------------------------------------

3164  3170   6   0 6 0 0 0  0 1 16 211  0  3    -  RetExit: CB, Fetch;                          ! fetch dictionary addres
3165  3166  10   0 6 0 0 1  0 2  0 300  0 16    -          Hold, RN, LeftShift(3);
3166  3167  31   6 0 0 1 1 14 3 11 206  0  3    -          Hold, tmp1 := Shift + CB;
3167  3171  31   4 6 1 1 1 14 3 11 203  0  3    -          Hold, tmp1 := tmp1 + RDEXIT;
3170  3174   0  31 3 0 0 1 14 1 16 205  0  3    -          Hold, Mdi + tmp1, Fetch;            ! fetch exit word in dict
3171  3172   0   7 3 1 0 0  4 2 13 341  0 16    -          BPC := Mdi and 7, RightShift(1);    ! set BPC and UPC
3172  3173   0   7 3 1 0 0  5 3 11 202  0  3    -          Mdi and not 7;
3173  3175  16   6 0 0 1 0 14 0  0   0  0 12    -          UPC := Shift + CB, Return;
```



```
! Routine ExSub.
!-------------------------------------------------------------------------
!
! Abstract:
!       ExSub is used by EXIT and EXGO to get the target segment numbe
!       routine number from the opcode/operand stream and set return a
!       to zero in the memory stack.
!
! Environment:
!       Segment and routine numbers are in the opcode/operand stream.
!       tmp2 = Target AP.
!
! Result:
!       Appropriate return addresses set to zero.
!
! Calls:
!       WordParm, XSTMap, VectSrv.
!
!-------------------------------------------------------------------------

3174  3176   0   0 6 0 0 0  0 3 10 255  0  1 3541  ExSub:  Call(WordParm);                      ! get ISN
3175  3177  30  30 0 0 1 0 14 3 12 175  0  1 2726          tmp := Shift + tmp, Call(XSTMap);   ! convert to XSN
3176  3200  31   0 1 0 1 0  0 3 17 164  4  3 3513          tmp1 := NextOp;                     ! get routine number
3177  3201  30  11 6 0 0 0 16 3 11 175  0  3    -          tmp - CS;
3200  3202  32   3 6 0 1 0  1 3 11 171 10  3 3203          tmp2 := AP, if Neq Goto(ExSub1);    ! if different code segme
3201  3203  31  10 6 0 0 0 16 3 11 172  0  3    -          tmp1 - RN;
3202  3205   0   0 6 0 0 0  0 0  0   0 15 12    -          if Eql Return;                      ! if current routine
3203  3206  32   6 6 1 0 0 14 1 17 173  0  3    -  ExSub1: tmp2 + ACBRA, Store;                ! set a return address to
3204  3204   0   0 6 1 0 0  1 3 11 167  0  3    -          0;
```

```
3205  3210   0   0 6 0 0 0  0 3 17 236  2  1 3331          if IntrPend Call(VectSrv);
3206  3211  32   5 6 1 0 0 14 1 16 170  0  3    -          tmp2 + ACBRS, Fetch;                    ! get return segment numb
3207  3207   0  30 3 0 0 0 16 3 11 164  0  3    -          Mdi - tmp;
3210  3213   0   0 6 0 0 0  0 3 11 160 10  3 3214          if Neq Goto(ExSub2);                    ! if segment numbers don'
3211  3214  32   7 6 1 0 0 14 1 16 165  0  3    -          tmp2 + ACBRR, Fetch;                    ! get return routine numb
3212  3212   0  31 3 0 0 0 16 3 11 161  0  3    -          Mdi - tmp1;
3213  3216   0   0 6 0 0 0  0 0  0   0 15 12    -          if Eql Return;                          ! if routine numbers matc
3214  3217  32   2 6 1 0 0 14 1 16 162  0  3    - ExSub2:  tmp2 + ACBDL, Fetch;                    ! get next ACB
3215  3215  32   7 3 0 1 0 14 3 11 171  0  3 3203          tmp2 := Mdi + SB, Goto(ExSub1);
```

```
$Title Error processing routines.
```

```
 Perq.Micro - Perq Q-Code interpreter microcode   Error processing routines.
VA    PA    X   Y A B W H AL F SF   Z CN JP   VT  File: Perq.Routine.1   Perq Microcode  Page      48
```

```
                                                 ! Routine ChkOvr.
                                                 !-----------------------------------------------------------------------
                                                 !
                                                 ! Abstract:
                                                 !       ChkOvr signals a ErrInxCase error.  ChkOvr is not called, rath
                                                 !       is jumped to.  It exits to RunError0.
                                                 !
                                                 ! Result:
                                                 !       tmp2 = ErrInxCase.
                                                 !
                                                 ! Calls:
                                                 !       RunError1.
                                                 !
                                                 !-----------------------------------------------------------------------
3216  3220  32  12 6 1 1 0  1 3 14  51  0  3 3253 ChkOvr: tmp2 := ErrInxCase, Goto(RunError0);
```

```
 Perq.Micro - Perq Q-Code interpreter microcode   Error processing routines.
VA    PA    X   Y A B W H AL F SF   Z CN JP   VT  File: Perq.Routine.1   Perq Microcode  Page      49
```

```
                                                 ! Routine ChkStk.
                                                 !-----------------------------------------------------------------------
                                                 !
                                                 ! Abstract:
                                                 !       ChkStk checks to be sure that N words can be pushed onto the m
                                                 !       stack without overflowing.  If the words will not fit, ChkStk
                                                 !       to the address in the 2910's S register.
                                                 !
                                                 ! Environment:
                                                 !       tmp10 = Number of words.
                                                 !       S, = Address of stack overflow handler:
                                                 !
                                                 ! Result:
                                                 !       tmp10 unchanged.
```

```
                                                    !
                                                    ! Calls:
                                                    !       VectSrv.
                                                    !
                                                    !-----------------------------------------------------------------

3217     13  40  17 6 0 1 0 14 3 11 155  0  3 3221  ChkStk: tmp10 := tmp10 + TP, Goto(ChkStk2);

3220  3221   0   0 6 0 0 0  0 3 17 236  0  1 3331  ChkStk1: Call(VectSrv);                    ! serve an interrupt
3221  3222  13  40 6 0 0 0 16 3 11 156  2  3 3220  ChkStk2: SL - tmp10, if IntrPend Goto(ChkStk1);
3222  3223  40  17 6 0 1 0 16 3 11 152  5  3 3224          tmp10 := tmp10 - TP, if C19 Goto(ChkStk3);  ! if no room
3223  3224   0   0 6 0 0 0  0 0  0   0  0 12   -          Return;                                 ! there's room, so return

                                                    !       No room, signal error by jumping to S.

3224  3225   0   0 6 0 0 0  0 0  0   0  1  7   -  ChkStk3:GotoS;


  Perq.Micro - Perq Q-Code interpreter microcode   Error processing routines.
 VA     PA   X    Y A B W H AL F SF    Z CN JP   VT  File: Perq.Routine.1   Perq Microcode  Page       50

                                                    ! Routine ErrCall.
                                                    !-----------------------------------------------------------------
                                                    !
                                                    ! Abstract:
                                                    !       ErrCall initiates the raising of an exception by calling routi
                                                    !       RNRaise in segment ExcCS (procedure Raise in module Except).
                                                    !       The caller pushes parameters to the exception (if any) onto th
                                                    !       memory stack before calling ErrCall.  ErrCall is jumped to and
                                                    !       exits to RefillJmp.  If the Except module has not been initial
                                                    !       is non-resident, or a stack overflow happens while calling Rai
                                                    !       transfer to Busted.
                                                    !
                                                    ! Environment:
                                                    !       tmp3 = ParameterSize = Number of words of parameters.
                                                    !       ExcCS = Segment number of the Except module.
                                                    !       ExcGP = Global pointer of the Except module.
                                                    !       tmp1 = Segment number of the exception.
                                                    !       tmp2 = Routine number of the exception.
                                                    !
                                                    ! Result:
                                                    !       Micro level call stack cleared.
                                                    !       Four words pushed on the memory stack.
                                                    !       Memory(TP - 0) = PStart = TP - 3 - ParameterSize.
                                                    !       Memory(TP - 1) = PEnd = TP - 3.
                                                    !       Memory(TP - 2) = ER = tmp2.
                                                    !       Memory(TP - 3) = ES = tmp1.
                                                    !       tmp10 unchanged.
                                                    !       ExcCS unchanged.
                                                    !       ExcGP unchanged.
                                                    !       tmp1 unchanged.
                                                    !
                                                    ! Calls:
                                                    !       ChkSeg, CllSub, RefillJmp, VectSrv.
```

```
                                                    !
                                                    ! Design:
                                                    !       If the Except module has not been initialized, ExcCS = 0.  Sin
                                                    !       segment 0 is guaranteed to be non-resident, we need not make a
                                                    !       separate check for ExcCS = 0.
                                                    !
                                                    !-----------------------------------------------------------------------

3225  3245  17    1 6 1 1 0 14 1 17 134  0  3    -   ErrCall: TP := TP + 1, Store;
3226  3243  31    0 6 0 0 0  0 3 17 377  0 17   -           tmp1, ThreeWayBranch(0);          ! ES, clear the call stac
3227  3244  17    7 6 0 0 0 16 0  4 135  0  3   -           Tos := TP - SB, Push;
3230  3242  17    1 6 1 1 0 14 1 17 137  0  3   -           TP := TP + 1, Store;
3231  3240  32    0 6 0 0 0  0 3 17 377  0 17   -           tmp2, ThreeWayBranch(0);          ! ER
3232  3241  17    1 6 1 1 0 14 1 17 141  0  3   -           TP := TP + 1, Store;
3233  3236   0   33 7 0 0 0 16 3 17 377  0 17   -           Tos - tmp3, ThreeWayBranch(0);    ! PStart
3234  3237  17    1 6 1 1 0 14 1 17 142  0  3   -           TP := TP + 1, Store;
3235  3235   0    0 7 0 0 0  0 0  5 151  0  3   -           Tos, Pop;                         ! PEnd
3236  3226  35   14 6 0 1 0  1 3 17 377  0 17   -           tmp5 := ExcCS, ThreeWayBranch(0); ! segment number of Raise
3237  3227  30   35 6 0 1 0  1 3 12 147  0  1 2671        tmp := tmp5, Call(ChkSeg);
                                                           tmp6 := tmp,                      ! code base of Raise
3240  3230  36   30 6 0 1 0  1 3 11  53  6  3 4042                  If Odd Goto(Busted);      ! if not resident
                                                           tmp := RNRaise,                   ! routine number of Raise
3241  3231  30    0 6 1 1 0  1 3 11  53  0 14 4042                  LoadS(Busted);            ! where to go on stack ov
3242  3232  37   15 6 0 1 0  1 3 17 377  0 17   -           tmp7 := ExcGP, ThreeWayBranch(0);
3243  3233  37    7 6 0 1 0 14 3 11 330  0  1 3064        tmp7 := tmp7 + SB, Call(CllSub);  ! global pointer of Raise
3244  3234   0    0 6 0 0 0  0 3 10 261  0  3 3525        Goto(RefillJmp);                  ! enter the routine
                                                    ! Error processing routines.
                                                    File: Perq.Routine.1   Perq Microcode  Page       51
```



```
                                                    ! Routine PCBackup
                                                    !-----------------------------------------------------------------------
                                                    !
                                                    ! Abstract:
                                                    !       PCBackup sets the Q-code program counter back N bytes.  This i
                                                    !       used when a recoverable error is detected (e.g. segment fault)
                                                    !       The PC is set back in order that the instruction can be re-exe
                                                    !       after the error condition is cleared.
                                                    !
                                                    ! Environment:
                                                    !       tmp11 = Number of bytes.
                                                    !
                                                    ! Result:
                                                    !       tmp = New program counter as byte offset from code base.
                                                    !       UPC set back.
                                                    !       BPC set back.
                                                    !
                                                    ! Calls:
                                                    !       VectSrv.
                                                    !
```

```
3245  3246  30  17 5 1 1 0  4 3 11 130  0  3    -  PCBackup: tmp := UState and 17;                  ! get byte program counte
3246  3247  16   0 6 0 0 0  0 2  0 340  0 16    -         UPC, LeftShift(1);                       ! word program counter ==
3247  3250  30  30 0 0 1 0 14 3 17 236  2  1 3331         tmp := Shift + tmp, if IntrPend Call(VectSrv);  ! full byte PC
3250  3251  30  41 6 0 1 0 16 2  0 341  0 16    -         tmp := tmp - tmp11, RightShift(1);       ! new byte program counte
3251  3252  16   3 0 1 1 0  5 3 11 124  0  3    -         UPC := Shift and not 3;                  ! set UPC back
3252  3253  30   7 6 1 0 0  4 0 13   0  0 12    -         BPC := tmp and 7, Return;                ! set BPC back
```



```
! Routine RunError, RunError0.
!-------------------------------------------------------------------------------
!
! Abstract:
!       RunError is called when the microcode wants to raise an except
!       The caller pushes parameters to the exception (if any) onto th
!       memory stack before calling RunError.  RunError is jumped to a
!       exits to ErrCall.  The variant of RunError that is called depe
!       on how many words of parameters were pushed on the memory stac
!       RunError0 is called when 0 words were pushed.  If parameters w
!       pushed onto the memory stack, RunError can be called with Para
!       in tmp3.
!
! Environment:
!       tmp3 = ParameterSize = Number of words of parameters (if calli
!                              RunError, not RunError0.
!       ExcCS = Segment number of the Except module.
!       ExcGP = Global pointer of the Except module.
!       tmp2 = Error number = Routine number of the exception.
!
! Result:
!       tmp3 unchanged.
!       ExcCS unchanged.
!       ExcGP unchanged.
!       tmp2 unchanged.
!       tmp1 = ExcCS.
!
! Calls:
!       ErrCall.
!
!-------------------------------------------------------------------------------
```

```
3253  1726  33   0 6 1 1 0  1 3 16 176  0  3    -  RunError0: tmp3 := 0;
3254   601  31  14 6 0 1 0  1 3 11 132  0  3 3225  Loc(RunError), tmp1 := ExcCS, Goto(ErrCall);
```



```
! Routine SASErr.
!-------------------------------------------------------------------------------
```

```
                                                    !
                                                    ! Abstract:
                                                    !       SASErr causes the ErrStrLong error.
                                                    !
                                                    ! Result:
                                                    !       tmp2 = ErrStrLong.
                                                    !
                                                    ! Calls:
                                                    !       RunError0.
                                                    !
                                                    !-------------------------------------------------------------------------

3255  3254  32  11 6 1 1 0  1 3 14  51  0  3 3253  SASErr: tmp2 := ErrStrLong, Goto(RunError0);

  Perq.Micro - Perq Q-Code interpreter microcode   Error processing routines.
 VA    PA   X   Y A B W H AL F SF   Z CN JP   VT  File: Perq.Routine.1   Perq Microcode  Page       55

                                                    ! Routine SegFault.
                                                    !-------------------------------------------------------------------------
                                                    !
                                                    ! Abstract:
                                                    !       SegFault signals a segment fault error.  If SegFault is called
                                                    !       STLATE, two segment numbers are passed.  Otherwise, tmp5 conta
                                                    !       segment number, and SegFault is entered at SegFault1.
                                                    !
                                                    ! Environment:
                                                    !       tmp5 = First segment number.
                                                    !       tmpstk0 = Second segment number.
                                                    !       tmp11 = Amount to back up the program counter.
                                                    !
                                                    ! Result:
                                                    !       Program counter backed up.
                                                    !       Four words pushed onto the memory stack.
                                                    !       Memory(TP - 0) = Code segment number.
                                                    !       Memory(TP - 1) = Stack segment number.
                                                    !       Memory(TP - 2) = Second segment number.
                                                    !       Memory(TP - 3) = First segment number.
                                                    !       Segment fault error signalled.
                                                    !
                                                    ! Calls:
                                                    !       PCBackup, RunError.
                                                    !
                                                    !-------------------------------------------------------------------------

3256  3255  70  35 6 0 1 0  1 3 11 120  0  3   -  SegFault1: tmpstk0 := tmp5;
3257  3257  17   1 6 1 1 0 14 1 17 121  0  3   -  SegFault: TP := TP + 1, Store;
3260  3256  35   0 6 0 0 0  0 3 11 116  0  3   -           tmp5;
3261  3261  17   1 6 1 1 0 14 1 17 117  0  3   -           TP := TP + 1, Store;
3262  3260  70   0 6 0 0 0  0 3 11 114  0  3   -           tmpstk0;
3263  3263  17   1 6 1 1 0 14 1 17 115  0  3   -           TP := TP + 1, Store;
3264  3262  12   0 6 0 0 0  0 3 11 112  0  3   -           SS;
3265  3265  17   1 6 1 1 0 14 1 17 113  0  3   -           TP := TP + 1, Store;
3266  3264  11   0 6 0 0 0  0 3 11 111  0  3   -           CS;
```

```
3267  3266  33   4 6 1 1 0  1 3 11 131  0  1 3245        tmp3 := 4, Call(PCBackup);          ! 4 words of parameters on
3270  3267  32   4 6 1 1 0  1 3 16 176  0  3    -        tmp2 := ErrSegmentFault, Goto(RunError);
```



```
                                        ! Routine StkOv:
                                        !-----------------------------------------------------------------------
                                        !
                                        ! Abstract:
                                        !       StkOv signals a stack overflow error.
                                        !
                                        ! Environment:
                                        !       tmp11 = Amount to back up the program counter.
                                        !
                                        ! Result:
                                        !       Program counter backed up.
                                        !       Stack overflow error signalled.
                                        !
                                        ! Calls:
                                        !       PCBackup, RunError0.
                                        !
                                        !-----------------------------------------------------------------------

                                        ! Routine StkOvPop.
                                        !-----------------------------------------------------------------------
                                        !
                                        ! Abstract:
                                        !       StkOvPop signals a stack overflow error after popping the expr
                                        !       stack.
                                        !
                                        ! Environment:
                                        !       tmp11 = Amount to back up the program counter.
                                        !
                                        ! Result:
                                        !       Stack popped.
                                        !       Program counter backed up.
                                        !       Stack overflow error signalled.
                                        !
                                        ! Calls:
                                        !       PCBackup, RunError0, SetQState.
                                        !
                                        !-----------------------------------------------------------------------

3271   441   0    0 6 0 0 0  0 0  5 335  0  3    -   StkOvPop: Pop;
3272   442   0    0 6 0 0 0  0 3 11 131  0  1 3245  StkOv:  Call(PCBackup);
3273   443  30  100 6 1 1 0  1 3 11 107  0  3    -          tmp := 100;                    !(must be separate line fo
3274  3270   0    0 6 0 0 0  0 3 12 177  0  1    -          Call(SetQState);               ! set stack limit (SL)
3275  3271  13  170 6 1 1 0 14 0  0   1  0 16    -          SL := SL + StackLimit;         ! add a little extra to w
3276  3272  32    5 6 1 1 0  1 3 14  51  0  3 3253          tmp2 := ErrStackOverflow, Goto(RunError0);
```

```
VA    PA   X   Y A B W H AL F SF   Z CN JP   VT  File: Perq.Routine.1   Perq Microcode  Page       57
                                                ! Routine UOP.
                                                !-----------------------------------------------------------------
                                                !
                                                ! Abstract:
                                                !       UOP signals an ErrUndfQcd error.
                                                !
                                                ! Result:
                                                !       tmp2 = ErrUndfQcd.
                                                !
                                                ! Calls:
                                                !       RunError0.
                                                !
                                                !-----------------------------------------------------------------
3277  3273  32  14 6 1 1 0  1 3 14  51  0  3 3253  UOP:    tmp2 := ErrUndfQcd, Goto(RunError0);


                                                $Title Interrupts: Microcode level and Pascal level.

 Perq.Micro - Perq Q-Code interpreter microcode   Interrupts: Microcode level and Pascal level.
VA    PA   X   Y A B W H AL F SF   Z CN JP   VT  File: Perq.Routine.1   Perq Microcode  Page       58
                                                ! Routine UserSrv.
                                                !-----------------------------------------------------------------
                                                !
                                                ! Abstract:
                                                !       UserSrv serves Pascal level interrupts by calling the appropri
                                                !       Pascal level interrupt service routine.
                                                !
                                                ! Environment.
                                                !       UserIntr bits 0..14 non-zero.  That is, UserIntr > 0.
                                                !
                                                ! Result:
                                                !       One interrupt served.
                                                !
                                                ! Calls:
                                                !       ChkSeg, ClIV, RefillJmp, VectSrv, StkOv.
                                                !
                                                ! Design:
                                                !       It is assumed that:
                                                !          1) The segment with the interrupt handler is always residen
                                                !          2) The table with variable routine descriptors (IntTab) is
                                                !             quadword aligned.
                                                !
                                                !-----------------------------------------------------------------
3300  3277  30  20 6 0 1 0  1 3 11  77  0  3   -  UserSrv: tmp := UserIntr;
3301  3300  30 377 6 1 0 0  4 3 11  76  0  3   -          tmp and 377;
3302  3301  64   0 6 1 1 0  1 3 11  75  0  3   -          dst := 0, Goto(UserSrv1);          ! initialize count
```

```
                                                    !*****          if Neq Goto(UserSrv1);
                                                    !*****  dst := 10, if IntrPend Call(VectSrv);
                                                    !*****  userIntr, RightShift(10);
                                                    !*****  tmp := Shift;                          ! swap bytes
3303  3302   0   0 6 0 0 0 0  0 3 17 236  2  1 3331 UserSrv1: if IntrPend Call(VectSrv);
3304  3303  30   0 6 0 0 0 0  0 2  0 341  0 16    -         tmp, RightShift(1);                    ! next bit position
3305  3304  30   0 0 0 1 0 0  0 3 11  66  6  3 3307         tmp := Shift, if Odd Goto(UserSrv2); ! if this bit is set.
3306  3305  64   1 6 1 1 0 14 3 11  75  0  3 3303         dst := dst + 1, Goto(UserSrv1);        ! update bit position cou

                                                    !       Bit found, index into IntTab by the bit position to get the ro
                                                    !       descriptor for the interrupt handler.

3307  3311  64   0 6 0 0 0 0  0 2  0 320  0 16    -  UserSrv2: dst, LeftShift(2);                   ! turn index into offset
3310  3312   0  21 0 0 0 0 14 1 12  71  0  3    -         Shift + IntPtr, Fetch4;                ! get variable routine de
3311  3306  20   0 6 1 1 0  7 0  0 200  0 16    -         UserIntr := UserIntr or 100000;        ! turn off further interr
3312  3307  41   0 6 1 1 0  1 3 16 335  0 14 3272         tmp11 := 0, LoadS(StkOv);              ! watch out for stack ove
3313  3310  35   0 3 0 1 0  0 3 11  64  0  3    -         tmp5 := Mdi;                           ! new CS = code segment n
3314  3313  37   7 3 0 1 0 14 3 11  63  0  3    -         tmp7 := Mdi + SB;                      ! new GP = global pointer
3315  3314  32   0 3 0 1 0  0 3 11  62  0  3    -         tmp2 := Mdi;                           ! new RN = routine number
3316  3315  40   7 3 0 1 0 14 3 11  52  0  3    -         tmp10 := Mdi + SB;                     ! new SL = static link
3317  3325  30  35 6 0 1 0  1 3 12 147  0  1 2671         tmp := tmp5, Call(ChkSeg);             ! get code base
                                                          tmp6 := tmp, Fetch,                    ! offset of routine dicti
                                                                                 if Odd Goto(Busted);  ! if not resident
3320  3326  36  30 6 0 1 0  1 1 16  53  6  3 4042         Hold, tmp := tmp2, LeftShift(3);
3321  3327  30  32 6 0 1 1  1 2  0 300  0 16    -         Hold, tmp1 := Shift + tmp6;            ! offset of dictionary en
3322  3330  31  36 0 0 1 1 14 3 11  42  0  3    -         Hold, Mdi + tmp1, Fetch4;              ! fetch dictionary entry.
3323  3335   0  31 3 0 0 1 14 1 12  45  0  3    -         Call(CllV);                            ! set up ACB
3324  3332   0   0 6 0 0 0 0  0 3 15 216  0  1 3076         dst, LeftShift(4);                     ! set up shift mask
3325  3333  64   0 6 0 0 0 0  0 2  0 260  0 16    -         Shift or 17, ShiftOnR;                 ! form LeftShift(dst)
3326  3334   0  17 0 1 0 0  7 0  1  46  0  3    -         1;                                     ! clear bit where interru
3327  3331   0   1 6 1 0 0  1 3 11  41  0  3    -         UserIntr := Shift xor UserIntr, Goto(RefillJmp);  ! enter rout
3330  3336  20  20 0 0 1 0 12 3 10 261  0  3 3525
```

```
  Perq.Micro - Perq Q-Code interpreter microcode
 VA     PA    X    Y A B W H AL F SF    Z CN JP    VT
```

```
 Interrupts: Microcode level and Pascal level.
File: Perq.Routine.1   Perq Microcode  Page       59

! Routine VectSrv.
!-------------------------------------------------------------------------
!
! Abstract:
!       VectSrv serves micro level interrupts by vectoring into the IO
!       microcode.  VectSrv should be called.
!
! Environment.
!       IntrPend true.
!
! Result:
!       Interrupt served.
!
! Calls:
!       IO microcode.
!
!-------------------------------------------------------------------------
```

```
3331   141   0   0 6 0 0 0   0 0   0 137   0   6     -
   Perq.Micro - Perq Q-Code interpreter microcode
  VA     PA    X    Y A B W H AL F SF    Z CN JP    VT

   Perq.Micro - Perq Q-Code interpreter microcode
  VA     PA    X    Y A B W H AL F SF    Z CN JP    VT

   Perq.Micro - Perq Q-Code interpreter microcode
  VA     PA    X    Y A B W H AL F SF    Z CN JP    VT
```

```
VectSrv: Vector(IntVec);
 Interrupts: Microcode level and Pascal level.
File: Perq.Micro   Perq Microcode  Page       60

$Nolist
 Miscellaneous.
File: Perq.Micro   Perq Microcode  Page       61

$List
 Miscellaneous.
File: Perq.Thesis  Perq Microcode  Page       62

$Include Perq.Thesis     ! Code used in Thesis
!
!       Luc A. Lepine (258009)                          August 14th, 198
!
!       This is implemented in partial fullfillment of my thesis.
!
!
!
! This implements the structured programming instructions using micro-
!
! Using previously undefined op-codes and registers, these routines an
! modified micro-code source implement the following instructions:
!
! If        <Lvrd1> <Lvrd2>
!
! While     <Lvrd>
!
! Repeat
!
! All these instructions require the evaluation of a boolean-expressio
! The result of this expression must be on the Top Of Stack when the
! instruction is executed. If the TOS is not 'true' or 'false' then a
! type error, exactly the same as the one for conditional branches, is
! raised.
!
! For simplicities sake, the following restrictions and conventions ap
!
! (1)   An LVRD instruction is used for the procedure addresses. This
!       us to check to make sure the correct type of information follo
!       op-code is a branch is to be taken. For the <If> instruction,
!       instruction may be replaced by a <No-op> instruction. This all
!       proper implementation of null 'else' and 'then' clauses.
!
! (2)   The instruction <Note> is used to delimit the range of <While>
!       <Repeat> instructions. Because a boolean-expression may contai
!       a function which changes variables, a <No-op> is allowed as th
!       <Lvrd> for a <While> instruction.
!
! (3)   The value of true is 1, of false is 0. This is defined in
!       set true/false in Perq.Routine.1
!
! (4)   Until the information created by a Note instruction can be pla
```

```
*                                                    !       In an ACB when a procedure is called, the <Repeat> instruction
*                                                    !       be nested. This does not matter for the <If> instruction and t
*                                                    !       <While> instruction places the information in the Acb as the
*                                                    !       return address.
*                                                    !
*                                                    ! (5)   An ErrInxCase exception will be raised if the instruction(s) f
*                                                    !       a While or If-Then-Else instruction is/are not the one(s) expe
*                                                    !
*                                                    ! (6)   An ErrStrIndx exception will be raised if the Note instruction
*                                                    !       not been executed prior to the execution of a While or Repeat
*                                                    !       instruction in a procedure.
*                                                    !

*    Perq.Micro - Perq Q-Code interpreter microcode   Miscellaneous.
*  VA    PA   X   Y A B W H AL F SF   Z CN JP   VT  File: Perq.Thesis   Perq Microcode  Page       63

*                                                    !
*                                                    !       This routine performs the IF operation. If TOS is true the 1st
*                                                    !       instruction is used to obtain the address of the procedure. If
*                                                    !       is false the 2nd instruction is used to obtain the address of
*                                                    !       procedure. A branch to CALLV is then made to perform the routi
*                                                    !       unless the instruction was a NOOP.
*                                                    !

* 3705  1274   0   0 6 0 0 0  0 3 10 113  0  3    -  OpCode(AIF), Nop;                        ! Required to allow NextInst bra
*                                                                                             ! the placer will complain other

* 3706  3664   0   0 6 0 0 0  0 3 10 103  0  1 3716       Call (ChkBool);                    ! Validate the TOS; Test value.
* 3707  3665   0   0 6 0 0 0  0 3 10 106 15  3 3713       If Eql GoTo (AifFalse);            ! If Tos = false, 2nd LVRD is us
* 3710  3666   0   0 6 0 0 0  0 3 10  63  0  1 3736       Call (DoLvrd);                     ! Load the 1st Lvrd
* 3711  3667   0   0 6 0 0 0  0 3 10  74  0  1 3725       Call (DoNull);                     ! The 2nd Lvrd is null
* 3712  3670   0   0 6 0 0 0  0 3 10  55  0  3 3744       GoTo (DoCallV);                    ! Now Do CallV

* 3713  3671   0   0 6 0 0 0  0 3 10  74  0  1 3725  AifFalse: Call (DoNull);              ! The 1st Lvrd is null
* 3714  3672   0   0 6 0 0 0  0 3 10  63  0  1 3736       Call (DoLvrd);                     ! Load the 2nd Lvrd
* 3715  3673   0   0 6 0 0 0  0 3 10  55  0  3 3744       GoTo (DoCallV);                    ! Now Do CallV

*                                                    !
*                                                    !       This routine determines whether the value on the top of stack
*                                                    !       be a boolean value. If not, a range exception is raised else t
*                                                    !       value of TOS is gated so that the instruction after the Call c
*                                                    !       if it is true (1) or false (0) ('neq' or 'eql').
*                                                    !

* 3716  3674 155    1 7 1 1 0  4 3 10 102  0  3    -  ChkBool: Work1 := Tos and 1;            ! Only valid values on Tos are 0
* 3717  3675   0 155 7 0 0 0 16 3 10 101  0  3    -       Tos - Work1;                       ! See if it checks
* 3720  3676   0   0 7 0 0 0  0 0  0   0 15 12    -       Tos, If Eql Return;                ! All went well, can test gate !
* 3721  3677  32  12 6 1 1 0  1 3 14  51  0  3 3253       tmp2 := ErrInxCase, GoTo (RunError0);  ! Else abort

*                                                    !
*                                                    !       This routine determines if the value in register Note1 has bee
*                                                    !       If not, an ErrStrIndx exception is raised to indicate that the
*                                                    !       is out of range (i.e. zero). If Note1 has been set, then a ret
```

```
                                                        !       made to the caller: Note1 is undefined if it contains zero.
*                                                       !
*
*  3722  3700 151   0 6 0 0 0  0 3 10  76  0  3    -   ChkNote: Note1;
*  3723  3701   0   0 6 0 0 0  0 0  0   0 10 12    -           If Neq Return;
*  3724  3702  32  10 6 1 1 0  1 3 14  51  0  3 3253           tmp2 := ErrStrIndx, GoTo (RunError0);
*                                                       !
*                                                       !       This routine looks at the next instruction ans skips by it.
*                                                       !       The intruction must be either an LVRD or a NOOP, else an excep
*                                                       !       is raised.
*                                                       !
*
*  3725  3703 155   0 1 0 1 0  0 3 17 164  4  3 3513   DoNull: Work1 := NextOp;
*  3726  3704 155 135 6 1 0 0 16 3 10  72  0  3    -           Work1 - Noop;                   !
*  3727  3705 155 142 6 1 0 0 16 0  0   0 15 12    -           Work1 - Lvrd, If Eql Return;  ! If <No-op>, return (1 byte op-
*  3730  3706  32  12 6 1 1 0  1 3 14  51 10  3 3253           tmp2 := ErrInxCase, If Neq GoTo(RunError0); ! Else must be <Lv
*  3731  3707 155   0 1 0 1 0  0 3 17 164  4  3 3513           Work1 := NextOp;
*  3732  3710 155   0 1 0 1 0  0 3 17 164  4  3 3513           Work1 := NextOp;
*  3733  3711 155   0 1 0 1 0  0 3 17 164  4  3 3513           Work1 := NextOp;
*  3734  3712 155   0 1 0 1 0  0 3 17 164  4  3 3513           Work1 := NextOp;
*  3735  3713   0   0 6 0 0 0  0 0  0   0  0 12    -           Return;                         ! <Lvrd> is a 5 byte op-code
*     Perq.Micro - Perq Q-Code interpreter microcode        Miscellaneous.
*      VA    PA   X   Y A B W H AL F SF   Z CN JP   VT    File: Perq.Thesis   Perq Microcode  Page       64
*                                                       !
*                                                       !       This routine looks at the next instruction. If it is a NOOP th
*                                                       !       it simply exits. If it is a LVRD then it is performed, else an
*                                                       !       exception is raised.
*                                                       !
*
*  3736  3714 157   0 1 0 1 0  0 3 17 164  4  3 3513   DoLvrd: Work3 := NextOp;
*  3737  3715 157 135 6 1 0 0 16 3 10  61  0  3    -           Work3 - Noop;
*  3740  3716 157 142 6 1 0 0 16 0  0   0 15 12    -           Work3 - Lvrd, If Eql Return;  ! No-op is simple
*  3741  3717  32  12 6 1 1 0  1 3 14  51 10  3 3253           tmp2 := ErrInxCase, If Neq GoTo (RunError0);
*  3742  3720 150   1 6 1 1 0  1 3 15 213  0  1 1262           CR0 := 1, Call(GoLvrd);       ! Into special mode, Load the VR
*  3743  3721 150   0 6 1 1 0  1 0  0   0 .0 12    -           CR0 := 0, Return;             ! Back to normal, return to call
*                                                       !
*                                                       !       This routine causes a CALLV instruction to be performed.
*                                                       !       it does not return to the caller
*                                                       !
*
*  3744  3722 157 135 6 1 0 0 16 3 10  54  0  3    -   DoCallV:Work3 - Noop;
*  3745  3723   0   0 6 0 0 0  0 3 16 357 10  3 1305           If Neq GoTo (GoCallV);
*  3746  3724   0   0 6 0 0 0  0 0  0 377 .0  2    -           NextInst(0);
*
*     Perq.Micro - Perq Q-Code interpreter microcode        Miscellaneous.
*      VA    PA   X   Y A B W H AL F SF   Z CN JP   VT    File: Perq.Thesis   Perq Microcode  Page       65
*                                                       !
*                                                       !       This instruction notes the offset into the code base of the in
*
```

```
*                                                    !       following it. This can be used to alter the default return
*                                                    !       address found in an ACB for a WHILE instruction, and as the ob
*                                                    !       of a Repeat loop.
*                                                    !
*                                                    !       N.B. Because of its implementation, Note1 can never be set to
*                                                    !            zero by this instruction. Therefore if Note1 is zero, it
*                                                    !            is undefined.
*                                                    !

*  3747  1260 151  17 5 1 1 0  4 3 10  52  0  3    -  OpCode(ANote),   Note1 := Ustate and 17;
*  3750  3725  16   6 6 0 0 0 16 2  0 340  0 16    -                  UPC - CB, LeftShift(1);
*                                                                     Note1 := Shift + Note1,
*  3751  3726 151 151 0 0 1 0 14 0  0 377  0  2    -                                      NextInst(0);


*     Perq.Micro - Perq Q-Code interpreter microcode   Miscellaneous.
*    VA     PA    X    Y A B W H AL F SF    Z CN JP   VT  File: Perq.Thesis   Perq Microcode  Page       66

*                                                    !
*                                                    !       This instruction causes a branch to the indicated procedure if
*                                                    !       expression in TOS is true. The return address is taken from th
*                                                    !       register set by the NOTE instruction. A special flag is set to
*                                                    !       indicate to the CALLV routine that the special register is to
*                                                    !

*  3752  3727   0   0 6 0 0 0  0 3 10  74  0  1 3725  FAWhile:         Call (DoNull);              ! Treat next instruction
*  3753  3730   0   0 6 0 0 0  0 0  0 377  0 .2    -                  NextInst(0);                ! That completes this ins

*  3754  1270   0   0 6 0 0 0  0 3 10  77  0  1 3722  OpCode(AWhile), Call (ChkNote);             ! Make sure Note1 has bee
*  3755  1271   0   0 6 0 0 0  0 3 10 103  0  1 3716                  Call (ChkBool);             ! Verify boolean value on
*  3756  1272   0   0 6 0 0 0  0 3 10  50 15  3 3752                  If Eql GoTo (FAWhile);      ! If false, treat LVRD as
*  3757  1273   0   0 6 0 0 0  0 3 10 046  0  3    -                  Nop;                        ! (Allows Placer to resol
*  3760  3731   0   0 6 0 0 0  0 3 10  63  0  1 3786                  Call (DoLvrd);              ! Else have to do LVRD
*  3761  3732 150   1 6 1 1 0  1 3 16 357  0  3 1305                  CRO := 1, GoTo (GoCallV);   ! Set flag, perform CALLV


*     Perq.Micro - Perq Q-Code interpreter microcode   Miscellaneous.
*    VA     PA    X    Y A B W H AL F SF    Z CN JP   VT  File: Perq.Thesis   Perq Microcode  Page       67

*                                                    !
*                                                    !       This routine causes a branch to the last NOTE instruction exec
*                                                    !       at this lexical level if the TOS is false.
*                                                    !

*                                                    DoRepeat:
*  3762  3733  52  17 5 1 1 0  4 3 10  43  0  3    -                  JmpOffSet := Ustate and 17;       ! determine current
*  3763  3734  16   6 6 0 0 0 16 2  0 340  0 16    -                  UPC - CB, LeftShift(1);           !
*  3764  3735  52  52 0 0 1 0 14 3 10  41  0  3    -                  JmpOffSet := Shift + JmpOffSet;   !
*  3765  3736  52 151 6 0 1 0 16 3 10  40  0  3    -                  JmpOffSet := JmpOffSet - Note1;   !
*  3766  3737  52   0 6 0 1 0  2 3 10  37  0  3    -                  JmpOffSet := not JmpOffSet;       ! Negate
*                                                                     JmpOffSet := JmpOffSet + 1,       !
*  3767  3740  52   1 6 1 1 0 14 3 10 272  0  3 3503                                GoTo (AdjustPC);    ! and Branch

*  3770  1264   0   0 6 0 0 0  0 3 10  77  0  1 3722  OpCode(ARepeat), Call (ChkNote);             ! Make sure Note1 has bee
```

```
*  3771  1265   0   0 6 0 0 0  0 3 10 103  0  1 3716          Call (ChkBool);              ! Verify boolean value on
*  3772  1266   0   0 6 0 0 0  0 3 10  44 15  3 3762          If Eql GoTo (DoRepeat);      ! If True, instruction co
*  3773  1267   0   0 6 0 0 0  0 0  0 377  0  2   -          NextInst(0);
     Perq.Micro - Perq Q-Code interpreter microcode  Miscellaneous.
      VA    PA   X   Y A B W H AL F SF   Z CN JP   VT  File: Perq.Init   Perq Microcode  Page      68
```

```
                                                      $Include Perq.Init


                                                      $Title Initialization.

     Perq.Micro - Perq Q-Code interpreter microcode  Initialization.
      VA    PA   X   Y A B W H AL F SF   Z CN JP   VT  File: Perq.Init   Perq Microcode  Page      69
                                                      ! Routine Init.
                                                      !-----------------------------------------------------------------
                                                      !
                                                      ! Abstract:
                                                      !       Init is the entrypoint and initialization of the Perq Q-code
                                                      !       interpreter microcode.
                                                      !
                                                      !-----------------------------------------------------------------

                                                      Loc(2400),

                                                      !       Initialize constant registers.

3774  2400  30   0 6 1 1 0  1 0  0   1  0 16   -      !       tmp := InitBlock;
                                                      !       Where := not 0;              !!!! use with CheckPC !!!
3775  2401  60 377 6 1 1 0  1 0  0 377  0 16   -      !       AllOnes := 177777;
3776  2402  53   0 6 1 1 0  1 0  0 200  0 16   -      !       SignBit := 100000;
3777  2403  61 377 6 1 1 0  3 0  0 177  0 16   -      !       SignXtnd := not 77777;       ! upper 5 bits set to 1
4000  2404 253 377 6 1 1 0  1 0  0   3  0 16   -      !       C1777 := 1777;
4001  2405 254   0 6 1 1 0  1 0  0   1  0 16   -      !       C400 := 400;

4002  2406 150   0 6 1 1 0  1 3 10  36  0  3   -      !       CR0 := 0;                    ! Control indicates all n
4003  3741 151   0 6 1 1 0  1 3 10  35  0  3   -      !       Note1 := 0;                  ! Note address is undefin


                                                      !       Initialize no interrupts and interrupts off.

4004  3742  20   0 6 1 1 0  1 0  0 200  0 16   -      !       UserIntr := 100000;          ! interrupts turned off

                                                      !       Z80 state registers for IO microcode.

4005  3743 227   0 6 1 1 0  1 0  0 200  0 16   -      !       Z80State := 100000;          ! use DpyTmp
4006  3744 241   0 6 1 1 0  1 3 10  32  0  3   -      !       Z80OState := 0;
4007  3745 233   0 6 1 1 0  1 3 10  31  0  3   -      !       Z80Status := 0;
4010  3746 243   0 6 1 1 0  1 3 10  26  0  3   -      !       Z80WantOutput := 0;
4011  3751 374   0 6 1 1 0  1 0  0  14  0 16   -      !       dpyTmp := 6000;              ! 128 lines of Off in Vis
4012  3752  51   0 6 1 1 0  1 0  2  27  0  3   -      !       zero := 0, StackReset;
```

```
4013  3750  30   0 6 0 0 0  0 1 14  30  0 3   -
4014  3747  35   0 3 0 1 0  0 3 10  24  0 3   -
4015  3753  12   0 3 0 1 0  0 3 10  23  0 3   -
4016  3754  14   0 6 1 1 0  1 3 10  22  0 3   -
4017  3755  15   0 6 1 1 0  1 3 10  21  0 3   -
4020  3756   3   7 6 0 1 0  1 3 10  20  0 3   -
4021  3757  10   0 6 1 1 0  1 3 10  17  0 3   -
4022  3760  11   0 6 1 1 0  1 3 10  16  0 3   -
4023  3761   6   0 6 1 1 0  1 3  7 351  0 3   -
4024  4026  16   0 6 1 1 0  1 0 13 353  0 3   -
4025  4024  30 160 6 1 1 0  1 3 12 177  0 1   -
4026  4025  40   7 6 0 1 0  1 3 10  14  0 3   -
4027  3763   7   0 6 1 0 0 14 1 16  15  0 3   -
4030  3762  17   7 3 0 1 0 14 3 11  54  0 3   -
4031  3323   7   1 6 1 0 0 14 1 16  61  0 3   -
4032  3316   4   7 3 0 1 0 14 3 11  53  0 14 4042
4033  3317  30  35 6 0 1 0  1 3 12 147  0 1 2671

4034  3320  36  30 6 0 1 0  1 1 16  53  6 3 4042
4035  3321  30   0 6 1 1 1  1 2 10 300  0 16  -
4036  3322  31  36 0 0 1 1 14 3  7 344  0 3   -
4037  4033   0  31 3 0 0 1 14 1 12 346  0 3   -
   Perq.Micro - Perq Q-Code interpreter microcode
  VA    PA   X   Y A B W H AL F SF   Z CN JP   VT

4040  4031  37   4 6 0 1 0  1 3 15 216  0 1 3076
4041  4032   0   0 6 0 0 0  0 3 10 261  0 3 3525

4042  3324   0   0 6 0 0 0  0 3 11  53  0 3 4042
   Perq.Micro - Perq Q-Code interpreter microcode
  VA    PA   X   Y A B W H AL F SF   Z CN JP   VT
```

```
        tmp, Fetch2;                                ! initial code segment nu
        tmp5 := Mdi;                                ! initial stack segment n
        SS := Mdi;                                  ! no exception module yet
        ExcCS := 0;
        ExcGP := 0;
        AP := SB;                                   ! main program dynamic li
        RN := 0;                                    ! main program return rou
        CS := 0;                                    ! main program return seg
        CB := 0;                                    ! main program return cod
        UPC := BPC := 0;                            ! main program return add
        tmp := 160, Call(SetQState);                ! set SB and SL registers
        tmp10 := SB;                                ! main program static lin
        SB + StkTP, Fetch;
        TP := Mdi + SB;                             ! initial top pointer
        SB + StkGP, Fetch;
        GP := Mdi + SB, LoadS(Busted);              ! initial global pointer
        tmp := tmp5, Call(ChkSeg);
        tmp6 := tmp, Fetch,                         ! new codebase
                                  if Odd Goto(Busted);  ! if not resident: B
        Hold, tmp := InitProc, LeftShift(3);
        Hold, tmp1 := Shift + tmp6;
        Hold, Mdi + tmp1, Fetch4;                   ! fetch dictionary entry:
 Initialization.
File: Perq.Init   Perq Microcode  Page        70

        tmp7 := GP, Call(CllV);                     ! set up ACB etc.
        Goto(RefillJmp);                            ! enter main program


Busted: Goto(Busted);
 Initialization.
File: Perq.Micro   Perq Microcode  Page        71

$NoList
```

The following pages contain the routines which were added or modified to insert the new instructions:

70, 73, 73, 88, 89, 100 through 104 and 104

Appendix C

INSTRUCTION TEST SAMPLE

To properly evaluate the performance of the new instructions, a series of tests were written to attempt to exercise all of the required features. These tests were to exercise both correctly and incorrectly coded instructions. The tests are split into two groups:

1. One test to exercise all the valid sequences of instructions.
2. Eight other tests to exercise invalid instruction coding of parameters for these new instructions.

Please note that some apparent procedures are not procedures but instrinsics. Procedures like statements which cause the compiler to generate code other than procedure calls. The instrinsics used in the test are the following:

1. MakeVRD: This causes the compiler to generate a LVRD instruction and place into its object code at that particular point. The only parameter is the name of a procedure or function.
2. LoadExpr: This causes the compiler to evaluate any expression and leave the result where it normally would be found before the result is assigned to a variable, or used as a parameter. LoadExpr in this

case is used to load the result of a boolean expression. Boolean expression results are always placed on the top of the expression stack.

3. InLineByte: This allows the compiler to insert a byte into the object code at that particular point. The byte to be inserted must be a constant whose value lies between 0 and 255.

Also included is the corresponding object code as disassmbled by the QDIS program on the PERQ. The mnemonic table used by this program was altered to include the definition of the four new op-codes. The op-codes should be simple to locate because they all begin with the character '<'.

Note that the section of the listing for the program 'TEST.PAS', which occurs first in this appendix, has had the section containing the PERQ op-codes removed. An edited version is contained in appendix A.

What follows is a list of the programs and the results they generated.

```
Source File Name:  Sys:Lepine>TestSys>Test.PAS

Segment File Name: Sys:Lepine>TestSys>Test.SEG

Compiled: 29 Sep 83 11:00:59



Line#    Seg#   Proc#

   1       0       0:D     0 Program Tests (Input, Output):
   2       0       0:D    32
   3       0       0:D    32 {
   4       0       0:D    32       Luc Lepine (258009)
   5       0       0:D    32
   6       0       0:D    32       This program written to partially fullfill requirements for CSI7999
   7       0       0:D    32
   8       0       0:D    32       This program test the various capabilities of the Structured
   9       0       0:D    32       instructions added in the Perq micro-code.
  10       0       0:D    32
  11       0       0:D    32       All forms of tests are included and documented. The basic
  12       0       0:D    32       structure of the program is straight line, except for the
  13       0       0:D    32       procedures required for testing purposes.
  14       0       0:D    32
  15       0       0:D    32 }
  16       0       0:D    32   Const

      The Qcode definitions that are part of the listing
      have been removed. An abbreviated form of this part of
      the listing has been placed in Appendix A.

  18       0       0:D    32
  19       0       0:D    32   Var
  20       0       0:D    32     I : Integer;
  21       0       0:D    32
  22       0       1:D     0   Procedure Test(Number:Integer);
  23       0       1:D     1
  24       0       1:D     1 {     This procedure displays the current test                }
  25       0       1:D     1
  26       0       1:D     1     Begin
  27       0       1:5     0       Writeln;
  28       0       1:5     0       Write('Test No. ', Number:3, ' - ')
  29       0       1:5     7     End;
  30       0       1:5    54
  31       0       0:5    56   Procedure SayTrue;
  32       0       0:5    56
  33       0       2:D     0 {     This procedure is called to write 'True !' on the terminal  }
  34       0       2:D     0
  35       0       2:D     0     Begin
  36       0       2:5     0       Write ('True !')
  37       0       2:5     0     End;
```

```
38      0       2:5     19
39      0       0:5     20   Procedure SayFalse;
40      0       3:D      0
41      0       3:D      0 {     This procedure is called to write 'False !' on the terminal }
42      0       3:D      0
43      0       3:D      0     Begin
44      0       3:5      0       Write('False !')
45      0       3:5      0     End;
46      0       3:5     20
47      0       0:5     22   Procedure NextI;
48      0       4:D      0
49      0       4:D      0 {     This procedure increments I, writes it out, then exits      }
50      0       4:D      0
51      0       4:D      0     Begin
52      0       4:5      0       I := I + 1;
53      0       4:5      0       Write(I:3)
54      0       4:5      6     End;
55      0       4:5     18
56      0       4:5     20 {     These procedures are used to demonstrate that the address
57      0       4:5     20       capability is not affected by new instruction calls        }
58      0       4:5     20
59      0       4:5     20
60      0       5:D      0   Procedure Show (A: Integer);
61      0       5:D      1     Begin
62      0       5:5      0       Write(A:3);
63      0       5:5      0     End;
64      0       5:5     11
65      0       5:5     12
66      0       0:5     12   Procedure CheckA;
67      0       6:D      0     Begin
68      0       6:5      0       Show(I);
69      0       6:5      0     End;
70      0       6:5      5
71      0       0:5      6   Procedure CheckB;
72      0       7:D      0     Begin
73      0       7:5      0       Show(I-4);
74      0       7:5      0     End;
75      0       7:5      7
76      0       0:5      8   Procedure CheckC;
77      0       8:D      0     Begin
78      0       8:5      0       Show(I);
79      0       8:5      0       I := I - 1;
80      0       8:5      5     End;
81      0       8:5     11
82      0       0:5     12   Procedure LastI;
83      0       9:D      0
84      0       9:D      0 {     This procedure is used to test nesting of the while loop,
85      0       9:D      0       to make sure the information generated by a NOTE is kept
86      0       9:D      0       through procedure calls and restored to the correct value
87      0       9:D      0       when exiting a routine.                                    }
88      0       9:D      0
89      0       9:D      0     Var
90      0       9:D      0       J : Integer;
91      0       9:D      0
```

```
 92    0      9:D    1    Begin
 93    0      9:5    0      J := I;
 94    0      9:5    0      InLineByte(ANOTE);
 95    0      9:5    3      LoadExpr(I < 10);                    { While I < 10 Do }
 96    0      9:5    8      InLineByte(AWHILE);
 97    0      9:5    8      MakeVrd(NextI);                      { NextI }
 98    0      9:5   14      Writeln;
 99    0      9:5   21      I := J - 1;
100    0      9:5   21    End;
101    0      9:5   26
102    0      0:5   28  Procedure TestExit;
103    0     10:D    0
104    0     10:D    0 {     This procedure tests the EXIT instruction and verifies that
105    0     10:D    0       an exit at a location other than the end of a procedure
106    0     10:D    0       will still result in the NOTE information being restored    }
107    0     10:D    0
108    0     10:D    0  Begin
109    0     10:5    0    Write('Testing EXIT routine. ');
110    0     10:5    0    InLineByte(ANOTE);
111    0     10:5   35    Exit(TestExit)
112    0     10:5   36  End;
113    0     10:5   40
114    0     10:5   42 {     The main program !
115    0     10:5   42
116    0     10:5   42  Begin
117    0      0:5    0
118    0      0:5    0    {Test the IF structure}
119    0      0:5    0
120    0      0:5    0    Test(1); LoadExpr(True);
121    0      0:5   41    InLineByte(AIF); MakeVRD(SayTrue); MakeVRD(SayFalse);
122    0      0:5   48
123    0      0:5   53    Test(2); LoadExpr(False);
124    0      0:5   57    InLineByte(AIF); MakeVRD(SayTrue); MakeVRD(SayFalse);
125    0      0:5   64
126    0      0:5   69    Test(3); LoadExpr(True);
127    0      0:5   73    InLineByte(AIF); InLineByte(NOOP); Makevrd(SayFalse);
128    0      0:5   76
129    0      0:5   81    Test(4); LoadExpr(True);
130    0      0:5   85    InLineByte(AIF); MakeVRD(SayTrue); InLineByte(NOOP);
131    0      0:5   92
132    0      0:5   93    Test(5); LoadExpr(True);
133    0      0:5   97    InLineByte(AIF); InLineByte(NOOP); InLineByte(NOOP);
134    0      0:5  100
135    0      0:5  101    Test(6); LoadExpr(False);
136    0      0:5  105    InLineByte(AIF); InLineByte(NOOP); InLineByte(NOOP);
137    0      0:5  108
138    0      0:5  109    { Test the Repeat structure }
139    0      0:5  109    Test(7);
140    0      0:5  113
141    0      0:5  113    I := 0;
142    0      0:5  113    InLineByte(ANOTE);
143    0      0:5  116    NextI;                    { Start of repeat loop }
144    0      0:5  119    LoadExpr (I >= 10);
145    0      0:5  119    InLineByte(AREPEAT);
```

```
146     0       0:5     123
147     0       0:5     124     { Test the While structure }
148     0       0:5     124     Test(8);
149     0       0:5     124
150     0       0:5     128     I := 0;
151     0       0:5     131     InLineByte(ANOTE);                  { Start of While loop }
152     0       0:5     132     LoadExpr (I < 7);                   { While I < 7 Do }
153     0       0:5     136     InLineByte(AWHILE);
154     0       0:5     137     MakeVrd(NextI);                     { Call NextI }
155     0       0:5     142
156     0       0:5     142     { Test nesting of instructions to check the NOTE information in the ACB }
157     0       0:5     142     Test(9);
158     0       0:5     142
159     0       0:5     146     Writeln;
160     0       0:5     146     I := 9;
161     0       0:5     153     InLineByte(ANOTE);                  { Start of While Loop }
162     0       0:5     157     LoadExpr (I >= 4);                  { While I > 4 Do }
163     0       0:5     161     InLineByte(AWHILE);
164     0       0:5     161     MakeVrd(LastI);                     { Call LastI }
165     0       0:5     167
166     0       0:5     167     { Test exit by other than simple termination }
167     0       0:5     167     Test(10);
168     0       0:5     167
169     0       0:5     171     I := 2;
170     0       0:5     171     InLineByte(ANOTE);
171     0       0:5     174     TestExit;
172     0       0:5     175     I := I + 1;
173     0       0:5     177     LoadExpr(I = 4);
174     0       0:5     183     InLineByte(AREPEAT);
175     0       0:5     187
176     0       0:5     188     Writeln ('Test Successfull !');
177     0       0:5     188
178     0       0:5     226     { This test verifies that addressing variables in other code segments
179     0       0:5     226       is still effective after a procedure envocation by a While/If-Then-Else
180     0       0:5     226       type instruction(s) }
181     0       0:5     226
182     0       0:5     226     Test(11); I := 27;                  { Test data }
183     0       0:5     234
184     0       0:5     234     LoadExpr(True);
185     0       0:5     234     InLineByte(AIF); MakeVRD(CheckA); MakeVRD(CheckB);
186     0       0:5     241
187     0       0:5     246     Test(12); I := 39;                  { Test data }
188     0       0:5     254
189     0       0:5     254     LoadExpr(False);
190     0       0:5     254     InLineByte(AIF); MakeVRD(CheckA); MakeVRD(CheckB);
191     0       0:5     261
192     0       0:5     266     Test(13); I := 47;                  { Test data }
193     0       0:5     274
194     0       0:5     274     InLineByte(ANOTE);
195     0       0:5     275     LoadExpr(I = 47);
196     0       0:5     275     InLineByte(AWHILE); MakeVRD(CheckC);
197     0       0:5     286
198     0       0:5     286     Writeln;
199     0       0:5     286     Writeln('Finis !')
```

```
 200        0        0:5    293
 200        0        0:5    293    End.


Disassembling the program: TESTS
     which was generated by compiling: Sys:Lepine>TestSys>Test.PAS

QCode Version Number:              3
Size of Global Data Block:        33
Length of Identifiers:             8
Number of Imported Segments:       2
Block containing import list:      3

File: Sys:Lepine>TestSys>test.SEG      Routine: TESTEXIT (10)

Lex Lev     PS       RPS       LTS     Enter     Exit
    2        0         0         0       188      228

  0: LSSN
  1: LOAB          16
  3: MMS2
  4: LSA       'Testing EXIT routine.'
 29: MMS2
 30: LDCO
 31: MMS
 32: CALLXB        2        5
 35: <Note>
 36: EXIT          0       10
 40: RETURN

File: Sys:Lepine>TestSys>test.SEG      Routine: LASTI     (9)

Lex Lev     PS       RPS       LTS     Enter     Exit
    2        0         0         1       160      186

  0: LDOB          32
  2: STLO
  3: <Note>
  4: LDOB          32
  6: LDC10
  7: LESI
  8: <While>
  9: LVRD           0        4        2
 14: LSSN
 15: LOAB          16
 17: MMS2
 18: CALLXB         1        8
 21: LDLO
 22: LDC1
 23: SBI
 24: STOB          32
 26: RETURN

File: Sys:Lepine>TestSys>test.SEG      Routine: CHECKC    (8)
```

```
Lex Lev     PS      RPS     LTS     Enter     Exit
    2        0        0       0       148      159

  0: LDOB          32
  2: MMS
  3: CALL           5
  5: LDOB          32
  7: LDC1
  8: SBI
  9: STOB          32
 11: RETURN
```

File: Sys:Lepine>TestSys>test.SEG Routine: CHECKB (7)

```
Lex Lev     PS      RPS     LTS     Enter     Exit
    2        0        0       0       140      147

  0: LDOB          32
  2: LDC4
  3: SBI
  4: MMS
  5: CALL           5
  7: RETURN
```

File: Sys:Lepine>TestSys>test.SEG Routine: CHECKA (6)

```
Lex Lev     PS      RPS     LTS     Enter     Exit
    2        0        0       0       134      139

  0: LDOB          32
  2: MMS
  3: CALL           5
  5: RETURN
```

File: Sys':Lepine>TestSys>test.SEG Routine: SHOW (5)

```
Lex Lev     PS      RPS     LTS     Enter     Exit
    2        1        1       0       122      133

  0: LSSN
  1: LOAB          16
  3: MMS2
  4: LDLO
  5: MMS
  6: LDC3
  7: MMS
  8: CALLXB         2       4
 11: RETURN
```

File: Sys:Lepine>TestSys>test.SEG Routine: NEXTI (4)

```
Lex Lev     PS      RPS     LTS     Enter     Exit
    2        0        0       0       102      120
```

```
  0: LDOB          32
  2: LDC1
  3: ADI
  4: STOB          32
  6: LSSN
  7: LOAB          16
  9: MMS2
 10: LDOB          32
 12: MMS
 13: LDC3
 14: MMS
 15: CALLXB        2       4
 18: RETURN

File: Sys:Lepine>TestSys>test.SEG     Routine: SAYFALSE (3)

Lex Lev    PS      RPS      LTS     Enter     Exit
    2       0        0        0        80      100

  0: LSSN
  1: LOAB          16
  3: MMS2
  4: LSA       'False !'
 14: MMS2
 15: LDCO
 16: MMS
 17: CALLXB        2       5
 20: RETURN

File: Sys:Lepine>TestSys>test.SEG     Routine: SAYTRUE (2)

Lex Lev    PS      RPS      LTS     Enter     Exit
    2       0        0        0        60       79

  0: LSSN
  1: LOAB          16
  3: MMS2
  4: LSA       'True !'
 13: MMS2
 14: LDCO
 15: MMS
 16: CALLXB        2       5
 19: RETURN

File: Sys:Lepine>TestSys>test.SEG     Routine: TEST     (1)

Lex Lev    PS      RPS      LTS     Enter     Exit
    2       1        1        0         4       58

  0: LSSN
  1: LOAB          16
  3: MMS2
  4: CALLXB         1       8
```

```
 7: LSSN
 8: LOAB          16
10: MMS2
11: LSA       'Test No. '
22: MMS2
23: LDCO
24: MMS
25: CALLXB        2        5
28: LSSN
29: LOAB          16
31: MMS2
32: LDLO
33: MMS
34: LDC3
35: MMS
36: CALLXB        2        4
39: LSSN
40: LOAB          16
42: MMS2
43: LSA
48: MMS2
49: LDCO
50: MMS
51: CALLXB        2        5
54: RETURN

File: Sys:Lepine>TestSys>test.SEG     Routine: TESTS     (0)

Lex Lev     PS       RPS      LTS     Enter     Exit
    1        0         0        0       230      549

 0: LSSN
 1: LOAB          0
 3: MMS2
 4: LSA
 7: MMS2
 8: LDCO
 9: MMS
10: LDC8
11: MMS
12: LDC1
13: MMS
14: LDCO
15: MMS
16: CALLXB        1        1
19: LSSN
20: LOAB          16
22: MMS2
23: LSA
25: MMS2
26: LDCO
27: MMS
28: LDC8
29: MMS
```

```
30: LDC1
31: MMS
32: LDC1
33: MMS
34: CALLXB      1       1
37: LDC1
38: MMS
39: CALL        1
41: LDC1
42: <If>
43: LVRD        0       2       2
48: LVRD        0       3       2
53: LDC2
54: MMS
55: CALL        1
57: LDC0
58: <If>
59: LVRD        0       2       2
64: LVRD        0       3       2
69: LDC3
70: MMS
71: CALL        1
73: LDC1
74: <If>
75: NOOP
76: LVRD        0       3       2
81: LDC4
82: MMS
83: CALL        1
85: LDC1
86: <If>
87: LVRD        0       2       2
92: NOOP
93: LDC5
94: MMS
95: CALL        1
97: LDC1
98: <If>
99: NOOP
100: NOOP
101: LDC6
102: MMS
103: CALL       1
105: LDC0
106: <If>
107: NOOP
108: NOOP
109: LDC7
110: MMS
111: CALL       1
113: LDC0
114: STOB       32
116: <Note>
117: CALL       4
```

```
119: LDOB       32
121: LDC10
122: GEQI
123: <Repeat
124: LDC8
125: MMS
126: CALL        1
128: LDC0
129: STOB       32
131: <Note>
132: LDOB       32
134: LDC7
135: LESI
136: <While>
137: LVRD        0       4       2
142: LDC9
143: MMS
144: CALL        1
146: LSSN
147: LOAB       16
149: MMS2
150: CALLXB      1       8
153: LDC9
154: STOB       32
156: <Note>
157: LDOB       32
159: LDC4
160: GEQI
161: <While>
162: LVRD        0       9       2
167: LDC10
168: MMS
169: CALL        1
171: LDC2
172: STOB       32
174: <Note>
175: CALL       10
177: LDOB       32
179: LDC1
180: ADI
181: STOB       32
183: LDOB       32
185: LDC4
186: EQUI
187: <Repeat
188: LSSN
189: LOAB       16
191: MMS2
192: LSA      'Test Successfull !'
213: MMS2
214: LDC0
215: MMS
216: CALLXB      2       5
219: LSSN
```

```
220: LOAB          16
222: MMS2
223: CALLXB         1        8
226: LDC11
227: MMS
228: CALL           1
230: LDCB          27
232: STOB          32
234: LDC1
235: <If>
236: LVRD           0        6        2
241: LVRD           0        7        2
246: LDC12
247: MMS
248: CALL           1
250: LDCB          39
252: STOB          32
254: LDCO
255: <If>
256: LVRD           0        6        2
261: LVRD           0        7        2
266: LDC13
267: MMS
268: CALL           1
270: LDCB          47
272: STOB          32
274: <Note>
275: LOOB          32
277: LDCB          47
279: EQUI
280: <While>
281: LVRD           0        8        2
286: LSSN
287: LOAB          16
289: MMS2
290: CALLXB         1        8
293: LSSN
294: LOAB          16
296: MMS2
297: LSA       'Finis !'
306: MMS2
307: LDCO
308: MMS
309: CALLXB         2        5
312: LSSN
313: LOAB          16
315: MMS2
316: CALLXB         1        8
319: RETURN

Test No.    1 - True !
Test No.    2 - False !
Test No.    3 -
Test No.    4 - True !
```

```
Test No.     5 -
Test No.     6 -
Test No.     7 -    1  2  3  4  5  6  7  8  9 10
Test No.     8 -    1  2  3  4  5  6  7
Test No.     9 -
 10
  9 10
  8  9 10
  7  8  9 10
  6  7  8  9 10
  5  6  7  8  9 10

Test No.    10 - Testing EXIT routine. Testing EXIT routine. Test Successfull !

Test No.    11 -  27
Test No.    12 -  35
Test No.    13 -  47
```

For brevity, only the program source is included in the rest of this appendix.

```
Program Test2;

{ This program will test what happens when a NOTE instructions is
  not executed before a While instruction }

Const
  {$Include Perq.Qcodes.DFS}

Begin
  Writeln('Testing While clause.');
  LoadExpr(False);
  InLineByte(AWHILE)
End.
```

```
     Disassembling the program: TEST2
          which was generated by compiling: Sys:Lepine>TestSys>Test2.PAS

     QCode Version Number:       3
     Size of Global Data Block:  32
     Length of Identifiers:      8
     Number of Imported Segments: 2
     Block containing import list: 2

     File: Sys:Lepine>TestSys>Test2.SEG     Routine: TEST2     (0)

     Lex Lev     PS       RPS      LTS     Enter     Exit
         1        0        0        0         4       83

       0: LSSN
       1: LOAB          0
       3: MMS2
       4: LSA
       7: MMS2
       8: LDC0
       9: MMS
      10: LDC8
      11: MMS
      12: LDC1
      13: MMS
      14: LDC0
      15: MMS
      16: CALLXB        1        1
      19: LSSN
      20: LOAB         16
      22: MMS2
      23: LSA
      25: MMS2
      26: LDC0
```

```
      27: MMS
      28: LDC8
      29: MMS
      30: LDC1
      31: MMS
      32: LDC1
      33: MMS
      34: CALLXB          1         1
      37: LSSN
      38: LOAB           16
      40: MMS2
      41: LSA       'Testing While clause.'
      64: MMS2
      65: LDCO
      66: MMS
      67: CALLXB          2         5
      70: LSSN
      71: LOAB           16
      73: MMS2
      74: CALLXB          1         8
      77: LDCO
      78: <While>
      79: RETURN
Testing While clause.

String index out of range
Aborted at     79 in routine TEST2 (0) in TEST2.
Called from   169 in routine  0 in LOADER.
Called from   226 in routine  1 in SYSTEM.
Called from   568 in routine  0 in SYSTEM.
```

```
Program Test3;

{ This program will test what happens when a NOTE instructions is
  not executed before a Repeat instruction }

Const
  {$Include Perq.Qcodes.DFS}

Begin
  Writeln('Testing Repeat clause.');
  LoadExpr(False);
  InLineByte(AREPEAT)
End.
```

```
Disassembling the program: TEST3
     which was generated by compiling: Sys:Lepine>TestSys>test3.PAS

QCode Version Number:          3
Size of Global Data Block:     32
Length of Identifiers:         8
Number of Imported Segments:   2
Block containing import list: 2

File: Sys:Lepine>TestSys>Test3.SEG     Routine: TEST3     (0)

Lex Lev     PS       RPS      LTS     Enter     Exit
    1        0         0        0         4       84

  0: LSSN
  1: LOAB           0
  3: MMS2
  4: LSA
  7: MMS2
  8: LDC0
  9: MMS
 10: LDC8
 11: MMS
 12: LDC1
 13: MMS
 14: LDC0
 15: MMS
 16: CALLXB         1        1
 19: LSSN
 20: LOAB          16
 22: MMS2
 23: LSA
 25: MMS2
 26: LDC0
 27: MMS
 28: LDC8
 29: MMS
 30: LDC1
 31: MMS
 32: LDC1
```

```
      33: MMS
      34: CALLXB         1        1
      37: LSSN
      38: LOAB          16
      40: MMS2
      41: LSA       'Testing Repeat clause.'
      65: MMS2
      66: LDCO
      67: MMS
      68: CALLXB         2        5
      71: LSSN
      72: LOAB          16
      74: MMS2
      75: CALLXB         1        8
      78: LDCO
      79: <Repeat
      80: RETURN

Testing Repeat clause.

String index out of range
Aborted at     80 in routine TEST3 (0) in TEST3.
Called from   169 in routine  0 in LOADER.
Called from   226 in routine  1 in SYSTEM.
Called from   568 in routine  0 in SYSTEM.
```

```
Program Test4;

{ This test verifies that a Note instruction must be executed in the
  same procedure to be valid, the value will not migrate forwards through
  procedure calls }

Const
  {$Include Perq.Qcodes.DFS}

  Procedure Test4A;
    Begin
      Writeln('Procedure Test4A entered.');
      LoadExpr(True);
      InLineByte(AWHILE); MakeVRD(Test4A)
    End;

Begin
  Writeln('Testing nested control structures !');
  InLineByte(ANOTE);
  Test4A;
End.
```

```
    Disassembling the program: TEST4
         which was generated by compiling: Sys:Lepine>TestSys>test4.PAS

    QCode Version Number:          3
    Size of Global Data Block:     32
    Length of Identifiers:         8
    Number of Imported Segments:   2
    Block containing import list:  2

    File: Sys:Lepine>TestSys>Test4.SEG     Routine: TEST4A    (1)

    Lex Lev     PS       RPS      LTS     Enter     Exit
        2        0         0        0         4       56

      0: LSSN
      1: LOAB          16
      3: MMS2
      4: LSA        'Procedure Test4A entered.'
     32: MMS2
     33: LDCO
     34: MMS
     35: CALLXB         1        5
     38: LSSN
     39: LOAB          16
     41: MMS2
     42: CALLXB         2        8
     45: LDC1
     46: <While>
     47: LVRD           0        1        2
     52: RETURN
```

```
File: Sys:Lepine>TestSys>Test4.SEG    Routine: TEST4    (0)

Lex Lev     PS       RPS      LTS     Enter     Exit
    1        0         0        0        58      152

  0: LSSN
  1: LOAB           0
  3: MMS2
  4: LSA
  7: MMS2
  8: LDC0
  9: MMS
 10: LDC8
 11: MMS
 12: LDC1
 13: MMS
 14: LDC0
 15: MMS
 16: CALLXB         2        1
 19: LSSN
 20: LOAB          16
 22: MMS2
 23: LSA
 25: MMS2
 26: LDC0
 27: MMS
 28: LDC8
 29: MMS
 30: LDC1
 31: MMS
 32: LDC1
 33: MMS
 34: CALLXB         2        1
 37: LSSN
 38: LOAB          16
 40: MMS2
 41: LSA       'Testing nested control structures !'
 78: MMS2
 79: LDC0
 80: MMS
 81: CALLXB         1        5
 84: LSSN
 85: LOAB          16
 87: MMS2
 88: CALLXB         2        8
 91: <Note>
 92: CALL           1
 94: RETURN
Testing nested control structures !
Procedure Test4A entered.

String index out of range
Aborted at     47 in routine TEST4A (1) in TEST4.
Called from    94 in routine TEST4 (0) in TEST4.
```

```
Called from    169 in routine  0 in LOADER.
Called from    226 in routine  1 in SYSTEM.
Called from    568 in routine  0 in SYSTEM.
```

```
Program Test5;

{ This test verifies that the object of a While instruction must be
  either a NOOP or a LVRD instruction.

  The instruction following the While instruction will be a RETURN }

Const
  {$Include Perq.Qcodes.DFS}

Begin
  Writeln('Testing validation of instruction after While instruction !');
  InLineByte(ANOTE);
  InLineByte(AWHILE);
End.
```

```
Disassembling the program: TEST5
     which was generated by compiling: Sys:Lepine>TestSys>test5.PAS.

QCode Version Number:          3
Size of Global Data Block:     32
Length of Identifiers:         8
Number of Imported Segments:   2
Block containing import list:  2

File: Sys:Lepine>TestSys>test5.SEG     Routine: TEST5     (0)

Lex Lev     PS       RPS      LTC      Enter     Exit
    1        0         0        0          4      121

  0: LSSN
  1: LOAB          0
  3: MMS2
  4: LSA
  7: MMS2
  8: LDCO
  9: MMS
 10: LDC8
 11: MMS
 12: LDC1
 13: MMS
 14: LDCO
 15: MMS
 16: CALLXB        1        1
 19: LSSN
 20: LOAB         16
 22: MMS2
 23: LSA
 25: MMS2
 26: LDCO
 27: MMS
 28: LDC8
 29: MMS
```

```
    30: LDC1
    31: MMS
    32: LDC1
    33: MMS
    34: CALLXB          1       1
    37: LSSN
    38: LOAB           16
    40: MMS2
    41: LSA       'Testing validation of instruction after While instruction !'
   102: MMS2
   103: LDCO
   104: MMS
   105: CALLXB          2       5
   108: LSSN
   109: LOAB           16
   111: MMS2
   112: CALLXB          1       8
   115: <Note>
   116: <While>
   117: RETURN
Testing validation of instruction after While instruction !

Expression out of range
Aborted at     118 in routine TEST5 (0) in TEST5.
Called from    169 in routine  0 in LOADER.
Called from    226 in routine  1 in SYSTEM.
Called from    568 in routine  0 in SYSTEM.
```

```
Program Test6;

{ This test verifies that the first instruction after an IF instruction
  must be a NOOP or a LVRD.                                              }

Const
  {$Include Perq.Qcodes.DFS}

Begin
  Writeln('Test 6 !');
  LoadExpr(True);
  InLineByte(AIF)
End.
```

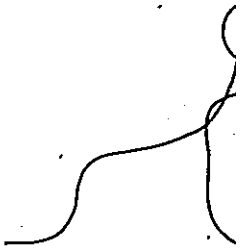
```
     Disassembling the program: TEST6
          which was generated by compiling: Sys:Lepine>TestSys>test6.PAS

     QCode Version Number:          3
     Size of Global Data Block:     32
     Length of Identifiers:         8
     Number of Imported Segments:   2
     Block containing import list:  2

     File: Sys:Lepine>TestSys>test6.SEG     Routine: TEST6     (0)

     Lex Lev    PS      RPS      LTS     Enter     Exit
         1       0        0        0         4       70

       0: LSSN
       1: LOAB          0
       3: MMS2
       4: LSA
       7: MMS2
       8: LDC0
       9: MMS
      10: LDC8
      11: MMS
      12: LDC1
      13: MMS
      14: LDC0
      15: MMS
      16: CALLXB        1        1
      19: LSSN
      20: LOAB         16
      22: MMS2
      23: LSA
      25: MMS2
      26: LDC0
      27: MMS
      28: LDC8
      29: MMS
      30: LDC1
      31: MMS
      32: LDC1
```

```
    33: MMS
    34: CALLXB          1         1
    37: LSSN
    38: LOAB           16
    40: MMS2
    41: LSA       'Test 6 !'
    51: MMS2
    52: LDCO
    53: MMS
    54: CALLXB          2         5
    57: LSSN
    58: LOAB           16
    60: MMS2
    61: CALLXB          1         8
    64: LDC1
    65: <If>
    66: RETURN


Expression out of range
Aborted at     67 in routine TEST6 (0) in TEST6.
Called from   169 in routine  0 in LOADER.
Called from   226 in routine  1 in SYSTEM.
Called from   568 in routine  0 in SYSTEM.
```

```
Program Test7;

{ This test verifies that the second instruction after an IF instruction
  must be a NOOP or a LVRD.                                              }

Const
  {$Include Perq.Qcodes.DFS}

Begin
  Writeln('Test 7 !');
  LoadExpr(True);
  InLineByte(AIF);
  MakeVRD(Test7);
End.
```

```
Disassembling the program: TEST7
     which was generated by compiling: Sys:Lepine>TestSys>test7.PAS

QCode Version Number:          3
Size of Global Data Block:     32
Length of Identifiers:         8
Number of Imported Segments:   2
Block containing import list: 2

File: Sys:Lepine>TestSys>test7.SEG     Routine: TEST7     (0)

Lex Lev     PS       RPS      LTS     Enter     Exit
    1        0         0        0         4       75

  0: LSSN
  1: LOAB          0
  3: MMS2
  4: LSA
  7: MMS2
  8: LDC0
  9: MMS
 10: LDC8
 11: MMS
 12: LDC1
 13: MMS
 14: LDC0
 15: MMS
 16: CALLXB        1        1
 19: LSSN
 20: LOAB         16
 22: MMS2
 23: LSA
 25: MMS2
 26: LDC0
 27: MMS
 28: LDC8
 29: MMS
 30: LDC1
```

```
        31: MMS
        32: LDC1
        33: MMS
        34: CALLXB          1         1
        37: LSSN
        38: LOAB           16
        40: MMS2
        41: LSA        'Test 7 !'
        51: MMS2
        52: LDCO
        53: MMS
        54: CALLXB          2         5
        57: LSSN
        58: LOAB           16
        60: MMS2
        61: CALLXB          1         8
        64: LDC1
        65: <If>
        66: LVRD            0         0         2
        71: RETURN

Test 7 !
Expression out of range
Aborted at      72 in routine TEST7 (0) in TEST7.
Called from    169 in routine  0 in LOADER.
Called from    226 in routine  1 in SYSTEM.
Called from    568 in routine  0 in SYSTEM.
```



```
Program Test8;

{ This test verifies that the second instruction after an IF instruction
  must be a NOOP or a LVRD.                                              }

Const
  {$Include Perq.Qcodes.DFS}

Begin
  Writeln('Test 8 !');
  LoadExpr(False);
  InLineByte(AIF);
  MakeVRD(Test8);
  InLineByte(NOOP)
End.
```

```
Disassembling the program: TEST8
     which was generated by compiling: Sys:Lepine>TestSys>test8.PAS

QCode Version Number:          3
Size of Global Data Block:     32
Length of Identifiers:         8
Number of Imported Segments:   2
Block containing import list:  2

File: Sys:Lepine>TestSys>test8.SEG     Routine: TEST8     (0)

Lex Lev     PS      RPS      LTS     Enter      Exit
    1        0        0        0         4        76

  0: LSSN
  1: LOAB          0
  3: MMS2
  4: LSA
  7: MMS2
  8: LDCO
  9: MMS
 10: LDC8
 11: MMS
 12: LDC1
 13: MMS
 14: LDCO
 15: MMS
 16: CALLXB        1        1
 19: LSSN
 20: LOAB         16
 22: MMS2
 23: LSA
 25: MMS2
 26: LDCO
 27: MMS
 28: LDC8
 29: MMS
 30: LDC1
```

```
31: MMS
32: LDC1
33: MMS
34: CALLXB        1        1
37: LSSN
38: LOAB         16
40: MMS2
41: LSA       'Test 8 !'
51: MMS2
52: LDCO
53: MMS
54: CALLXB        2        5
57: LSSN
58: LOAB         16
60: MMS2
61: CALLXB        1        8
64: LDCO
65: <If>
66: LVRD          0        0        2
71: NOOP
72: RETURN

Test 8 !
```

```
Program Test9;

{ This test verifies that the value on the top of the expression stack
  must be boolean, i.e. 0 or 1}

Const
  {$Include Perq.Qcodes.DFS}

Begin
  Writeln('Test 9 !');
  LoadExpr(7);
  InLineByte(AIF);
  MakeVRD(Test9);
  InLineByte(NOOP)
End.
```

```
Disassembling the program: TEST9
     which was generated by compiling: Sys:Lepine>TestSys>test9.PAS

QCode Version Number:          3
Size of Global Data Block:     32
Length of Identifiers:         8
Number of Imported Segments:   2
Block containing import list:  2

File: Sys:Lepine>TestSys>test9.SEG     Routine: TEST9     (0)

Lex Lev     PS       RPS      LTS     Enter     Exit
    1        0         0        0         4       76

  0: LSSN
  1: LOAB          0
  3: MMS2
  4: LSA
  7: MMS2
  8: LDC0
  9: MMS
 10: LDC8
 11: MMS
 12: LDC1
 13: MMS
 14: LDC0
 15: MMS
 16: CALLXB        1        1
 19: LSSN
 20: LOAB         16
 22: MMS2
 23: LSA
 25: MMS2
 26: LDC0
 27: MMS
 28: LDC8
 29: MMS
 30: LDC1
```

```
31: MMS
32: LDC1
33: MMS
34: CALLXB          1         1
37: LSSN
38: LOAB           16
40: MMS2
41: LSA       'Test 9 !'
51: MMS2
52: LDC0
53: MMS
54: CALLXB          2         5
57: LSSN
58: LOAB           16
60: MMS2
61: CALLXB          1         8
64: LDC7
65: <If>
66: LVRD            0         0         2
71: NOOP
72: RETURN

Test 9 !
Expression out of range
Aborted at      66 in routine TEST9 (0) in TEST9.
Called from    169 in routine  0 in LOADER.
Called from    226 in routine  1 in SYSTEM.
Called from    568 in routine  0 in SYSTEM.
```

Appendix D

AMDAHL INSTRUCTION EMULATING ROUTINE

This is the listing of the routine which performs the emulation of the three new instructions on the Amdahl V7A. The routine is written in standard IBM assembler, and was compiled using the H level assembler from IBM. This version of the assembler contains modifications obtained from the Stanford Linear Accelerating Centre (SLAC), but this has no effect on the object code produced by the compiler, or the performance of the routine.

The implementation of this routine requires no modifications to the CMS operating system. It is implemented by loading the routine in this appendix into permanent system storage, using a CMS command. The program is then called, specifying the 'START' parameter. This causes the program check interupt vector to be altered by the program so that it obtains control when a program check occurs. Whenever any of the new instructions are executed, a program check specifying an invalid machine instruction occurs. The program obtains control, verifies it is a new instruction, performs it, then passes control back to the user program. If the interupt is not caused by one of the new instructions, the program will pass control to CMS, so that it can handle the error.

The program occupies 630 bytes of storage, which is comprised of 81 instructions for the emulating program and a few constants and more instructions for the CMS command interface.

N.B. The object code is in hexadecimal.

```
NEWINST                                 EXTERNAL SYMBOL DICTIONARY                                  PAGE    1
SYMBOL   TYPE  ID  ADDR   LENGTH  LD ID FLAGS                          (SLAC V2.9)  ASM H V 02  10.40 10/25/83
NEWINST   SD 0001 000000 000276         00

                                MACRO AND COPY CODE EXTERNAL SOURCE CROSS REFERENCE

  DMSSP   ON  MNT390
     MEMBER(S) USED - NUCON

  CMSLIB  ON  MNT390
     MEMBER(S) USED - DMSLN    DMSLNC   DMSLND   DMSLNP   DMSLNU   DMSLNY   DMSLNZ   LINEDIT

  OSMACRO  ON  MNT390
     MEMBER(S) USED - RETURN   SAVE

                                        DSECT CROSS REFERENCE
  DSECT      STMT  LENGTH       DSECT      STMT  LENGTH       DSECT      STMT  LENGTH       DSECT      STMT  LENGTH
 @OPCODE      350  000000      @PARM        351  000000      NUCON        355  000DF8

NEWINST  NEW SYSTEM/370 INSTRUCTIONS FOR CONTROL STRUCTURES                                          PAGE    2
  ACTIVE USINGS - NONE
D-LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                          (SLAC V2.9)  ASM H V 02  10.40 10/25/83
                                     2 *                                                                        NEW00020
                                     3 * THIS ROUTINE IMPLEMENTS USING 'MACRO-CODE' THE 'IF' AND 'WHILE'        NEW00030
                                     4 * CONTROL STRUCTURES. THE IMPLEMENTATION REQUIRES THAT THIS ROUTINE      NEW00040
                                     5 * BE FIXED ANYWHERE IN THE CMS NUCLEUS AREA, AND THEN ENVOKED TO         NEW00050
                                     6 * TAKE CONTROL OF THE VIRTUAL MACHINE WHEN A PROGRAM CHECK OCCURS.       NEW00060
                                     7 *                                                                        NEW00070
                                     8 * THE FOLLOWING INSTRUCTIONS ARE IMPLEMENTED:                            NEW00080
                                     9 *                                                                        NEW00090
                                    10 * INSTRUCTION                 IF-THEN-ELSE                              NEW00100
                                    11 * OP-CODE                     X'F4'                                     NEW00110
                                    12 * INSTRUCTION LENGTH          6 BYTES                                   NEW00120
                                    13 * PARAMETERS: CC MASK         HIGH ORDER NIBBLE, SECOND BYTE            NEW00130
                                    14 *             LINK REG.       LOW ORDER NIBBLE, SECOND BYTE             NEW00140
                                    15 *             SCON 1          RELOCATABLE ADDRESS CONSTANT, TRUE CLAUSE NEW00150
                                    16 *             SCON 2          RELOCATABLE ADDRESS CONSTANT, FALSE CLAUSE NEW00160
                                    17 * FUNCTION                     THE CC MASK IS USED TO DETERMINE WHETHER  NEW00170
                                    18 *                             THE BRANCH CONDITION TO TRUE OR FALSE. IF NEW00180
                                    19 *                             IT IS TRUE, THE FIRST ADDRESS IS USED TO  NEW00190
                                    20 *                             DETERMINE THE LOCATION OF THE ROUTINE TO  NEW00200
                                    21 *                             BE EXECUTED. IF IT IS FALSE, THE SECOND   NEW00210
                                    22 *                             ADDRESS IS USED. THE PROGRAM COUNTER IS   NEW00220
                                    23 *                             UPDATED TO REFLECT THIS FACT. THE ADDRESS NEW00230
                                    24 *                             OF THE NEXT INSTRUCTION IS PLACED INTO    NEW00240
                                    25 *                             THE LINK REGISTER.                        NEW00250
                                    26 * ERROR(S)                    NONE.                                     NEW00260

                                    28 * INSTRUCTION                 WHILE.                                    NEW00280
                                    29 * OP-CODE                     X'A0'                                     NEW00290
                                    30 * INSTRUCTION LENGTH          4 BYTES                                   NEW00300
                                    31 * PARAMETERS: CC MASK         HIGH ORDER NIBBLE, SECOND BYTE            NEW00310
                                    32 *             LINK REG.       LOW ORDER NIBBLE, SECOND BYTE             NEW00320
                                    33 *             SCON            RELOCATABLE ADDRESS CONSTANT              NEW00330
```

```
 34 * FUNCTION               THE CC MASK IS USED TO DETERMINE WHETHER       NEW00340
 35 *                        BRANCH CONDITION IS TRUE. IF IT IS, THE        NEW00350
 36 *                        ADDRESS IS USED TO UPDATE THE PROGRAM          NEW00360
 37 *                        COUNTER TO EFFECT A BRANCH; THE LINK           NEW00370
 38 *                        REGISTER TO ALTERED SO THAT THE ADDRESS        NEW00380
 39 *                        OF THIS INSTRUCTION IS PLACED INTO IT.         NEW00390
 40 * ERROR(S)               NONE.                                          NEW00400

 42 * INSTRUCTION            IF-THEN                                        NEW00420
 43 * OP-CODE                X'A1'                                          NEW00430
 44 * INSTRUCTION LENGTH     4 BYTES                                        NEW00440
 45 * PARAMETERS: CC MASK    HIGH ORDER NIBBLE, SECOND BYTE                 NEW00450
 46 *             LINK REG.  LOW ORDER NIBBLE, SECOND BYTE                  NEW00460
 47 *             SCON       RELOCATABLE ADDRESS CONSTANT                   NEW00470
 48 * FUNCTION               THIS FUNCTION THE SAME WAY AS THE              NEW00480
 49 *                        IF-THEN-ELSE INSTRUCTION, EXCEPT THAT          NEW00490
 50 *                        THERE IS NO SECOND ADDRESS, AND NO ACTION      NEW00500
 51 *                        IS TAKEN IF THE BRANCH CONDITION IS FALSE,     NEW00510
 52 *                        THE NEXT SEQUENTIAL INSTRUCTION WILL BE        NEW00520
 53 *                        PROCESSED.                                     NEW00530
 54 * ERROR(S)               NONE.                                          NEW00540
                                                                           PAGE    3
NEWINST  NEW SYSTEM/370 INSTRUCTIONS FOR CONTROL STRUCTURES
  ACTIVE USINGS - NONE
D-LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT        (SLAC V2.9)  ASM H V 02 10.40 10/25/83
                                     56 * THE IMPLEMENTATION OF THE NEW INSTRUCTIONS IS AS FOLLOWS:        NEW00560
                                     57 *                                                                  NEW00570
                                     58 * (1)    THE NEW INSTRUCTION CAUSES A PROGRAM CHECK, CODE 1        NEW00580
                                     59 *        (UNDEFINED OP-CODE)                                       NEW00590
                                     60 * (2)    THIS ROUTINE OBTAINS CONTROL AND DETERMINES IF THIS IS AN NEW00600
                                     61 *        INSTRUCTION TO BE SIMULATED. IF NOT, THE CMS VERSION OF THE NEW00610
                                     62 *        PGMNPSW FIELD IS LOADED, AND CONTROL PASSES TO CMS.       NEW00620
                                     63 * (3)    THE CORRECT ROUTINE IS ENVOKED. ANY REGISTERS REQUIRED ETC. NEW00630
                                     64 *        ARE ALTERED. THE CORRECT ROUTINE IS CALLED BY REPLACING THE NEW00640
                                     65 *        ADDRESS IN THE PGMOPSW, THEN RELOADING THE REGISTERS, THEN NEW00650
                                     66 *        THE PGMOPSW TO TRANSFER CONTROL TO THE PROPER ROUTINE.    NEW00660

                                     68 * NOTES: ANY LINKAGE IS PERFORMED USING THE BRANCH AND SAVE CONVENTION, NEW00680
                                     69 *           ONLY THE ADDRESS IS PLACED INTO THE REGISTER.          NEW00690
                                     70 *        THIS ROUTINE SHOULD BE LOADED, USING THE PROGRAM 'RESLIB', NEW00700
                                     71 *           INTO CMS NUCLEUS STORAGE THEN ACTIVATED BY CALLING THE NEW00710
                                     72 *           PROGRAM 'NEWINST' WITH THE 'START' OPTION. TO DEACTIVATE NEW00720
                                     73 *           THE NEW INSRUCTIONS, CALL 'NEWINST' WITH THE 'STOP' OPTION. NEW00730
                                     74 *        THE CC MASK SERVES EXACTLY THE SAME FUNCTION AS IT DOES FOR NEW00740
                                     75 *           THE 'BRANCH ON CONDITION' INSTRUCTIONS.                NEW00750
                                     76 *        ONLY THE BC MODE PSW IS ASSUMED HERE. CMS DOES NOT UNDERSTAND NEW00760
                                     77 *           OR USE EC MODE PSWS, SO THIS IS A SAFE ASSUMPTION.     NEW00770
                                                                           PAGE    4
NEWINST  THE CMS PROGRAM
  ACTIVE USINGS - NONE
D-LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT        (SLAC V2.9)  ASM H V 02 10.40 10/25/83
                                     79 *                                                                  NEW00790
                                     80 * THIS ROUTINE MUST BE LOADED PERMANENTLY INTO STORAGE, OR THE CMS NEW00800
                                     81 * WILL FAIL ON THE FIRST PROGRAM CHECK.                            NEW00810
                                     82 *                                                                  NEW00820
000000                               83 NEWINST  START .                                                   NEW00830
```

```
                                        85          PRINT NOGEN          NO MACRO EXPANSIONS                                NEW00850
                                                                         USEFULL EQUATES                                    NEW00870
                                00000   87 RO       EQU   0                                                                 NEW00880
                                00001   88 R1       EQU   1                                                                 NEW00890
                                00002   89 PARMREG  EQU   2                                                                 NEW00900
                                00003   90 R3       EQU   3                                                                 NEW00910
                                00004   91 R4       EQU   4                                                                 NEW00920
                                00005   92 R5       EQU   5                                                                 NEW00930
                                00006   93 R6       EQU   6                                                                 NEW00940
                                00007   94 R7       EQU   7                                                                 NEW00950
                                00008   95 R8       EQU   8                                                                 NEW00960
                                00009   96 SCON2    EQU   9                                                                 NEW00970
                                0000A   97 SCON1    EQU   10                                                                NEW00980
                                0000B   98 OPCODE   EQU   11                                                                NEW00990
                                0000C   99 BASEREG  EQU   12                                                                NEW01000
                                0000D  100 R13      EQU   13                                                                NEW01010
                                0000E  101 R14      EQU   14                                                                NEW01020
                                0000F  102 R15      EQU   15                                                                NEW01040
                                000F4  104 IFTE     EQU   X'F4'          IF-THEN-ELSE                                       NEW01050
                                000A0  105 WHILE    EQU   X'A0'          WHILE-DO                                           NEW01060
                                000A1  106 IFT      EQU   X'A1'          IF-THEN                                            NEW01080
                                00000  108          USING NUCON,R0       ADDRESS NUCLEUS                                    NEW01090
                                00000  109          USING @PARM,PARMREG  ADDRESS PROGRAM PARAMETERS                         NEW01100
                                00000  110          USING @OPCODE,OPCODE                                                    NEW01120
                                       112          SAVE  (14,12),T,*    STOW THE REGISTERS                                 NEW01130
000010 18CF                            117          LR    BASEREG,R15       ESTABLISH ADDESSABILITY                      NEW01140
                                00000  118          USING NEWINST,BASEREG       TO PROGRAM                               NEW01160
000012 4120 1008                00008  120          LA    PARMREG,8(,R1)      VERIFY THE PARAMETER(S)                    NEW01170
000016 95FF 2000        00000          121          CLI   @PARM,X'FF'         MUST BE AT LEAST ONE                       NEW01180
00001A 4780 C072                00072  122          BE    TOOFEW                                                         NEW01190
00001E 95FF 2008        00008          123          CLI   @PARM+8,X'FF'                                                  NEW01200
000022 4770 C096                00096  124          BNE   TOOMANY                                                        NEW01220
000026 D507 C258 2000 00258 00000      126          CLC   =CL8'START',@PARM                                              NEW01230
00002C 4780 C0C2                000C2  127          BE    DOSTART                                                        NEW01240
000030 D507 C260 2000 00260 00000      128          CLC   =CL8'STOP',@PARM                                               NEW01250
000036 4780 C0DC                000DC  129          BE    DOSTOP                                                         NEW01260
                                       131          LINEDIT TEXT='........ - INVALID PARAMETER',DOT=NO,                  XNEW01270
                                                          SUB=(CHARA,(PARMREG))                                         NEW01280
                                       144 SETRC8   RETURN (14,12),T,RC=8     RETURN TO CALLER                           NEW01300
                                       151 TOOFEW   LINEDIT TEXT='NO PARAMETER SUPPLIED',DOT=NO                          NEW01320
                                                                                                                         NEW01330
000092 47F0 C064                00064  161          B     SETRC8                                                         NEW01350
                                       163 TOOMANY  LINEDIT TEXT='ONLY ONE PARAMETER IS ALLOWED',DOT=NO                  NEW01360
0000BE 47F0 C064                00064  173          B     SETRC8                                                         NEW01380
NEWINST  INITIALISE THE ENVIROMENT FOR THE NEW INSTRUCTIONS                                                             PAGE    5
  ACTIVE USINGS - (NUCON,R0) (@PARM,R2) (@OPCODE,R11) (NEWINST,R12)
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                    (SLAC V2.9)  ASM H V 02 10.40 10/25/83
                                       175 DOSTART  DS    0H                                                             NEW01390
0000C2                                 176          CLC   PGMNPSW,NEWPSW     ARE WE ALREADY RUNNING ENABLED ?           NEW01400
0000C2 D507 0068 C250 00068 00250      177          BE    SETRC0             YES, IGNORE THE REQUEST                    NEW01410
0000C8 4780 C0EC                00EC   178          MVC   OLDPSW,PGMNPSW     NO, KEEP A COPY OF THE CMS PGMNPSW         NEW01420
0000CC D207 C248 0068 00248 00068      179          MVC   PGMNPSW,NEWPSW        AND REPLACE WITH OUR OWN COPY           NEW01430
0000D2 D207 0068 C250 00068 00250      180          B     SETRC0             FINIS . . . .                              PAGE    6
0000D8 47F0 C0EC                00EC
NEWINST  TERMINATE THE ENVIROMENT FOR THE NEW INSTRUCTIONS
  ACTIVE USINGS - (NUCON,R0) (@PARM,R2) (@OPCODE,R11) (NEWINST,R12)
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                    (SLAC V2.9)  ASM H V 02 10.40 10/25/83
```

```
0000DC                                  182 DOSTOP   DS    0H                                                    NEW01450
0000DC D507 0068 C250 00068 00250       183          CLC   PGMNPSW,NEWPSW      ARE WE RUNNING ENABLED ?          NEW01460
0000E2 4770 C0EC              00EC      184          BNE   SETRCO              NO, IGNORE THE REQUEST            NEW01470
0000E6 D207 0068 C248 00068 00248       185          MVC   PGMNPSW,OLDPSW      YES, RE-INSERT THE CMS PGMNPSW    NEW01480

                                        187 SETRCO   LINEDIT TEXT='REQUEST COMPLETE',DOT=NO                       NEW01500
                                        197          RETURN (14,12),T,RC=0    RETURN TO SYSTEM                  NEW01510

                                        203          DROP  BASEREG            DROP ADDRESSABILITY               NEW01530
                                                                                                                PAGE    7
NEWINST  INSTRUCTION EMULATION ROUTINE
  ACTIVE USINGS - (NUCON,R0) (@PARM,R2) (@OPCODE,R11)
  LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT                    (SLAC V2.9)  ASM H V 02 10.40 10/25/83
                                        205 * THIS ROUTINE DETERMINES IF THE NEW INSTRUCTIONS SHOULD BE EMULATED.   NEW01550
                                        206 *                                                                       NEW01560
                                        207 * THE LOGIC IS AS FOLLOWS:                                              NEW01570
                                        208 *                                                                       NEW01580
                                        209 * (1)    THIS ROUTINE GAINS CONTROL ON A PROGRAM CHECK.                 NEW01590
                                        210 * (2)    IF NOT AN UNDEFINED INSTRUCTION CHECK, CONTROL PASSES TO CMS   NEW01600
                                        211 * (3)    IF NOT ANY OF THE NEW INSTRUCTIONS, CONTROL PASSES TO CMS      NEW01610
                                        212 * (4)    THE INSTRUCTION IS EMULATED, I.E. REGISTERS AND PSW ALTERED    NEW01620
                                        213 *        AS REQUIRED SO THAT NORMAL INSTRUCTION PROCESSING WILL         NEW01630
                                        214 *        CONTINUE AS IF THE HARDWARE HAD PERFORMED THE INSTRUCTION.     NEW01640
                                        215 * (5)    CONTROL IS RETURNED TO THE INSTRUCTION FOLLOWING THE ONE THAT   NEW01650
                                        216 *        CAUSED THE ORIGINAL PROGRAM CHECK. CONTROL IS NOT PASSED TO    NEW01660
                                        217 *        CMS.                                                           NEW01670

                                        219 PERFORM  DS    0H             ENTRY POINT FOR PROGRAM CHECK           NEW01690
000116                                  220          STM   R0,R15,GPRLOG  SAVE GENERAL REGISTERS                  NEW01700
000116 900F 0180              00180     221          BALR  BASEREG,0      ESTABLISH ADDRESSABILITY                NEW01710
00011A 05C0                             222          USING *,BASEREG                                              NEW01720
                              0011C     224          CLC   =Y(1),PGMOPSW+2     IS THIS DUE TO AN ILLEGAL OP-CODE ?  NEW01740
00011C D501 C158 002A 00274 0002A       225          BNE   TOCMS          NO, PASS CONTROL TO CMS                 NEW01750
000122 4770 C036              00152     226          L     OPCODE,PGMOPSW+4    GET THE ADDRESS OF THE NEXT OP-CODE NEW01760
000126 58B0 002C              0002C     227          SR    R14,R14        DETERMINE THE INSTRUCTION LENGTH         NEW01770
00012A 1BEE                             228          IC    R14,PGMOPSW+4                                        NEW01780
00012C 43E0 002C              0002C     229          SRL   R14,6                                                  NEW01790
000130 88E0 0006              00006     230          IC    R14,ILENGTH(R14)                                       NEW01800
000134 43EE C126              00242     231          SR    OPCODE,R14     AND CALCULATE ADDRESS OF THE OP-CODE    NEW01810
000138 1BBE                             232          CLI   @OPCODE,IFTE   DETERMINE IF IT IS A VALID INSTRUCTION  NEW01820
00013A 95F4 B000        00000           233          BE    DOIFTE                                                 NEW01830
00013E 4780 C064              00180     234          CLI   @OPCODE,IFT                                            NEW01840
000142 95A1 B000        00000           235          BE    DOIFT                                                  NEW01850
000146 4780 C044              00160     236          CLI   @OPCODE,WHILE                                          NEW01860
00014A 95A0 B000        00000           237          BE    DOWHILE                                                NEW01870
00014E 4780 C0A6              001C2     239 TOCMS    MVC   MCKOPSW,OLDPSW      PASS CONTROL TO CMS              NEW01890
000152 D207 0030 C12C 00030 00248       240          LM    R0,R15,GPRLOG       RESTORE GENERAL PURPOSE REGISTERS NEW01900
000158 980F 0180              00180     241          LPSW  MCKOPSW             AND LOAD CMS VERSION OF PGMNPSW  NEW01910
00015C 8200 0030        00030                                                                                   PAGE    8
NEWINST  IF-THEN INSTRUCTION
  ACTIVE USINGS - (NUCON,R0) (@PARM,R2) (@OPCODE,R11) (NEWINST+X'11C',R12)
  LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT                    (SLAC V2.9)  ASM H V 02 10.40 10/25/83
                                        243 DOIFT    DS    0H                                                    NEW01930
000160                                  244          LA    R1,@OPCODE+2  POINT TO SCON                          NEW01940
000160 4110 B002              00002     245          BAL   R14,SCON      OBTAIN ADDRESS                         NEW01950
000164 45E0 C0DA              001F6     246          LR    SCON1,R15     COPY RESULT                            NEW01960
000168 18AF
```

```
00016A 43E0 B001              00001   248          IC    R14,@OPCODE+1 OBTAIN BRANCH MASK                          NEW01980
00016E 54E0 C14C              00268   249          N     R14,=A(X'F0') AND ONLY BRANCH MASK                        NEW01990
000172 BFF8 002C              0002C   250          ICM   R15,8,PGMOPSW+4                                           NEW02000
000176 04F0                           251          SPM   R15           SET CC+ILC+PROGRAM MASK                     NEW02010
000178 44E0 C0A2              001BE   252          EX    R14,TESTIFTE  MERGE WITH IF-THEN-ELSE, BRANCH IFF TRUE   NEW02020
00017C 47F0 C11E              0023A   253          B     NOOP          RETURN CONTROL TO USER PROGRAM              NEW02030
                                                                                                                  PAGE    9
NEWINST  IF-THEN-ELSE INSTRUCTION
  ACTIVE USINGS - (NUCON,R0) (@PARM,R2) (@OPCODE,R11) (NEWINST+X'11C',R12)
  LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT                    (SLAC V2.9)  ASM H V 02 10.40 10/25/83
000180                                255 DOIFTE   DS    0H                                                      NEW02050
000180 4110 B002              00002   256          LA    R1,@OPCODE+2  POINT TO 1ST SCON                         NEW02060
000184 45E0 C0DA              001F6   257          BAL   R14,SCON      OBTAIN ADDRESS                            NEW02070
000188 18AF                           258          LR    SCON1,R15     COPY RESULT                               NEW02080
00018A 4110 B004              00004   260          LA    R1,@OPCODE+4  POINT TO 2ND SCON                         NEW02100
00018E 45E0 C0DA              001F6   261          BAL   R14,SCON      OBTAIN ADDRESS                            NEW02110
000192 189F                           262          LR    SCON2,R15     COPY RESULT                               NEW02120
000194 43E0 B001              00001   264          IC    R14,@OPCODE+1       COPY BRANCH MASK                    NEW02140
000198 54E0 C14C              00268   265          N     R14,=A(X'F0')       ONLY WANT BRANCH MASK               NEW02150
00019C BFF8 002C              0002C   266          ICM   R15,8,PGMOPSW+4     OBTAIN PROGRAM MASK                 NEW02160
0001A0 04F0                           267          SPM   R15                 SET CC+ILC+PROGRAM MASK             NEW02170
0001A2 44E0 C0A2              001BE   268          EX    R14,TESTIFTE        BRANCH IFF TRUE                     NEW02180
0001A6 45E0 C102              0021E   270          BAL   R14,SETLINK         FALSE, SET THE LINK REGISTER        NEW02200
0001AA BE97 002D              0002D   271          STCM  SCON2,7,PGMOPSW+5   UPDATE ADDRESS IN PGMOPSW           NEW02210
0001AE 47F0 C11E              0023A   272          B     NOOP                RETURN CONTROL TO USER PROGRAM      NEW02220
0001B2 45E0 C102              0021E   274 IFTRUE   BAL   R14,SETLINK       TRUE, SET THE LINK REGISTER         NEW02240
0001B6 BEA7 002D              0002D   275          STCM  SCON1,7,PGMOPSW+5   UPDATE ADDRESS IN PGMOPSW           NEW02250
0001BA 47F0 C11E              0023A   276          B     NOOP                RETURN CONTROL TO USER PROGRAM      NEW02260
0001BE 4700 C096              001B2   278 TESTIFTE BC    *-*,IFTRUE          *** EXECUTED INSTRUCTION ***        NEW02280
                                                                                                                  PAGE   10
NEWINST  WHILE INSTRUCTION
  ACTIVE USINGS - (NUCON,R0) (@PARM,R2) (@OPCODE,R11) (NEWINST+X'11C',R12)
  LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT                    (SLAC V2.9)  ASM H V 02 10.40 10/25/83
0001C2                                280 DOWHILE  DS    0H                                                      NEW02300
0001C2 4110 B002              00002   281          LA    R1,@OPCODE+2  POINT TO SCON                            NEW02310
0001C6 45E0 C0DA              001F6   282          BAL   R14,SCON      OBTAIN ADDRESS                            NEW02320
0001CA 18AF                           283          LR    SCON1,R15     COPY RESULT                               NEW02330
0001CC 43E0 B001              00001   285          IC    R14,@OPCODE+1 COPY BRANCH MASK                          NEW02350
0001D0 54E0 C14C              00268   286          N     R14,=A(X'F0') ONLY BRANCH MASK                          NEW02360
0001D4 BFF8 002C              0002C   287          ICM   R15,8,PGMOPSW+4     OBTAIN PROGRAM MASK                 NEW02370
0001D8 04F0                           288          SPM   R15           SET CC+ILC+PROGRAM MASK                     NEW02380
0001DA 44E0 C0D6              001F2   289          EX    R14,TESTWHIL  BRANCH IFF TRUE                           NEW02390
0001DE 47F0 C11E              0023A   290          B     NOOP          RETURN CONTROL TO USER PROGRAM              NEW02400
0001E2 BEB7 002D              0002D   292 WHILEDO  STCM  OPCODE,7,PGMOPSW+5  TRUE, RETURN TO SAME INSTRUCTION    NEW02420
0001E6 45E0 C102              0021E   293          BAL   R14,SETLINK        SET LINK REGISTER                   NEW02430
0001EA BEA7 002D              0002D   294          STCM  SCON1,7,PGMOPSW+5   UPDATE ADDRESS IN PGMOPSW           NEW02440
0001EE 47F0 C11E              0023A   295          B     NOOP                RETURN CONTROL TO USER PROGRAM      NEW02450
0001F2 4700 C0C6              001E2   297 TESTWHIL BC    *-*,WHILEDO         *** EXECUTED INSTRUCTION ***        NEW02470
                                                                                                                  PAGE   11
NEWINST  LOCAL SUBROUTINE(S)
  ACTIVE USINGS - (NUCON,R0) (@PARM,R2) (@OPCODE,R11) (NEWINST+X'11C',R12)
  LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT                    (SLAC V2.9)  ASM H V 02 10.40 10/25/83
0001F6                                299 SCON     DS    0H            RESOLVE SCON ADDRESS                      NEW02490
                                      300 *                            RETURN ADDRESS IS R14                     NEW02500
                                      301 *                            ADDRESS RETURNED IN R15                   NEW02510
                                      302 *                            SCON POINTED TO BY R1                     NEW02520
                                      303 *                            R0 IS A WORK REGISTER                     NEW02530
```

```
0001F6 1BFF                                305          SR    R15,R15          OBTAIN ADDRESS                          NEW02550
0001F8 43F0 1000          00000            306          IC    R15,0(,R1)       DETERMINE WHICH REGISTER                NEW02560
0001FC 88F0 0004          00004            307          SRL   R15,4                                                    NEW02570
000200 12FF                                308          LTR   R15,R15          IS IT ZERO ?                            NEW02580
000202 4780 C0F2          0020E            309          BZ    SCONZERO         YES, IBM SAYS RO = 0 FOR SCON           NEW02590
000206 89F0 0002          00002            310          SLL   R15,2            MULTIPLY BY 4                           NEW02600
00020A 58FF 0180          00180            311          L     R15,GPRLOG(R15)  R15 CONTAINS VALUE OF REGISTER          NEW02610
00020E                                     313 SCONZERO DS    0H                                                      NEW02630
00020E 4800 1000          00000            314          LH    R0,0(,R1)                                               NEW02640
000212 5400 C150          0026C            315          N     R0,=A(X'FFF')    ONLY USE OFFSET                         NEW02650
000216 1AF0                                316          AR    R15,R0           ADDRESS NOW IN R15                      NEW02660
000218 41F0 F000          00000            317          LA    R15,0(,R15)      STROBE OFF HIGH ORDER BYTE              NEW02670
00021C 07FE                                318          BR    R14              RETURN TO CALLER                        NEW02680
                                                                                                                      PAGE   12
NEWINST  LOCAL SUBROUTINE(S)                                                          (SLAC V2.9)  ASM H V 02 10.40 10/25/83
  ACTIVE USINGS - (NUCON,R0) (@PARM,R2),(@OPCODE,R11) (NEWINST+X'11C',R12)
  LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT
00021E                                     320 SETLINK  DS    0H               SET THE LINK REGISTER                   NEW02700
                                           321 *                               R14 - RETURN ADDRESS                    NEW02710
                                           322 *                               OPCODE - POINTS TO OP-CODE              NEW02720
                                           323 *                               R15 - WORK REGISTER                     NEW02730
00021E 43F0 B001          00001            325          IC    R15,@OPCODE+1 GET THE LINK REGISTER NUMBER              NEW02750
000222 54F0 C154          00270            326          N     R15,=A(15)                                              NEW02760
000226 89F0 0002          00002            327          SLL   R15,2            MULTIPLY BY 4                           NEW02770
00022A 41FF 0180          00180            328          LA    R15,GPRLOG(R15)     POINT TO REGISTER                  NEW02780
00022E D203 F000 002C 00000 0002C          329          MVC   0(4,R15),PGMOPSW+4  COPY CONTENTS OF PSW ADDRESS      NEW02790
000234 9200 F000          00000            330          MVI   0(R15),0         FOR BAS(R), NO ILC+CC+PGM MASK          NEW02800
000238 07FE                                331          BR    R14              RETURN TO CALLER                        NEW02810

00023A                                     333 NOOP     DS    0H               PASS CONTROL TO USER PROGRAM            NEW02830
00023A 980F 0180          00180            334          LM    R0,R15,GPRLOG RESTORE REGISTERS                        NEW02840
00023E 8200 0028          00028            335          LPSW  PGMOPSW          RETURN CONTROL TO USER PROGRAM          NEW02850
                                                                                                                      PAGE   13
NEWINST  CONSTANTS AND DSECTS
  ACTIVE USINGS - (NUCON,R0) (@PARM,R2) (@OPCODE,R11) (NEWINST+X'11C',R12)
  LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT                            (SLAC V2.9)  ASM H V 02 10.40 10/25/83
000242 02040406                            337 ILENGTH  DC    AL1(2,4,4,6)     ILC DEFINITION IN BYTES                 NEW02870
000248                                     339 OLDPSW   DS    D                AREA TO HOLD CMS VERSION OF PGMNPSW     NEW02890
000250 0000000000000116                    340 NEWPSW   DC    A(0,PERFORM)     OUR VERSION OF THE PGMNPSW              NEW02900
000258                                     342          LTORG ,                LITERAL POOL                            NEW02920
000258 E2E3C1D9E3404040                    343                =CL8'START'
000260 E2E3D6D740404040                    344                =CL8'STOP'
000268 000000F0                            345                =A(X'F0')
00026C 00000FFF                            346                =A(X'FFF')
000270 0000000F                            347                =A(15)
000274 0001                                348                =Y(1)
000000                                     350 @OPCODE  DSECT ,                THIS ALLOWS US TO USE @OPCODE           NEW02940
000000                                     351 @PARM    DSECT ,                THIS ALLOWS US TO USE @PARM             NEW02950
                                           353          NUCON ,                MAP OF CMS NUCLEUS                      NEW02970
000000                                     969          END   NEWINST          END OF ASSEMBLY                         NEW02990
                                                                                                                      PAGE   14
                                                CROSS REFERENCE                                (SLAC V2.9)  ASM H V 02 10.40 10/25/83
NEWINST
SYMBOL    LEN    VALUE    DEFN  REFERENCES
@OPCODE   00001 00000000 0350  0110 0232 0234 0236 0244 0248 0256 0260 0264 0281 0285 0325
@PARM     00001 00000000 0351  0109 0121 0123 0126 0128
BASEREG   00001 0000000C 0099  0117 0118 0203 0221 0222
```

```
CONSTACK 00320    000A60 0849  0622
CPULOG   00008    000080 0463  0464
DIAGTIME 00024    000280 0491  0492
DMPTITLE 00132    0003BC 0529  0527
DMSA0003 00002    00005E 0138  0132
DMSA0036 00002    00008E 0158  0153
DMSA0063 00002    0000BA 0170  0165
DMSA0090 00002    000104 0194  0189
DMSC0003 00001 000000C2 0139  0134
DMSL0003 00001 0000001C 0137  0135
DMSL0036 00001 00000015 0157  0155
DMSL0063 00001 0000001D 0169  0167
DMSL0090 00001 00000010 0193  0191
DMST0003 00001    000041 0135  0137
DMST0036 00001    000078 0155  0157
DMST0063 00001    00009C 0167  0169
DMST0090 00001    0000F2 0191  0193
DOIFT    00002    000160 0243  0235
DOIFTE   00002    000180 0255  0233
DOSTART  00002    0000C2 0175  0127
DOSTOP   00002    0000DC 0182  0129
DOWHILE  00002    0001C2 0280  0237
FPRLOG   00008    000160 0475  0526
GPRLOG   00004    000180 0476  0220 0240 0311 0328 0334 0522
IFT      00001 000000A1 0106  0234
IFTE     00001 000000F4 0104  0232
IFTRUE   00004    0001B2 0274  0278
ILENGTH  00001    000242 0337  0230
IPLPSW   00008    000000 0435  0438
LOWSAVE  00160    0000C0 0473  0523
MCKOPSW  00008    000030 0447  0239 0241
NEWINST  00001 00000000 0083  0118 0969
NEWPSW   00004    000250 0340  0176 0179 0183
NOOP     00002    00023A 0333  0253 0272 0276 0290 0295
NUCON    00001 00000000 0355  0108
OLDPSW   00008    000248 0339  0178 0185 0239
OPCODE   00001 0000000B 0098  0110 0226 0231 0292
PARMREG  00001 00000002 0089  0109 0120 0140
PERFORM  00002    000116 0219  0340
PGMNPSW  00008    000068 0459  0176 0178 0179 0183 0185
PGMOPSW  00008    000028 0446  0224 0226 0228 0250 0266 0271 0275 0287 0292 0294 0329 0335
R0       00001 00000000 0087  0108 0220 0240 0314 0315 0316 0334
R1       00001 00000001 0088  0120 0244 0256 0260 0281 0306 0314
R14      00001 0000000E 0101  0227 0227 0228 0229 0230 0230 0231 0245 0248 0249 0252 0257 0261 0264 0265 0268 0270 0274
                              0282 0285 0286 0289 0293 0318 0331
R15      00001 00000005 0102  0117 0220 0240 0246 0250 0251 0258 0262 0266 0267 0283 0287 0288 0305 0305 0306 0307 0308
                              0308 0310 0311 0311 0316 0317 0317 0325 0326 0327 0328 0328 0329 0330 0334
SCON     00002    0001F6 0299  0245 0257 0261 0282
SCONZERO 00002    00020E 0313  0309
SCON1    00001 0000000A 0097  0246 0258 0275 0283 0294
SCON2    00001 00000009 0096  0262 0271
SETLINK  00002    00021E 0320  0270 0274 0293
SETRC0   00002    0000EC 0188  0177 0180 0184
SETRC8   00002    000064 0145  0161 0173
```

```
TESTIFTE 00004    0001BE 0278  0252 0268
TESTWHIL 00004    0001F2 0297  0289
NEWINST                                                    CROSS REFERENCE
SYMBOL     LEN    VALUE  DEFN  REFERENCES
TOCMS      00006  000152 0239  0225
TOOFEW     00002  000072 0152  0122
TOOMANY    00002  000096 0164  0124
WHILE      00001 000000A0 0105  0236
WHILEDO    00004  0001E2 0292  0297
=A(X'FFF')
           00004  00026C 0346  0315
=A(X'FO')
           00004  000268 0345  0249 0265 0286
=A(15)     00004  000270 0347  0326
=CL8'START'
           00008  000258 0343  0126
=CL8'STOP'
           00008  000260 0344  0128
=Y(1)      00002  000274 0348  0224
```


NEWINST

```
    NO STATEMENTS FLAGGED IN THIS ASSEMBLY
  OVERRIDING PARAMETERS-  NONE
  OPTIONS FOR THIS ASSEMBLY
   OPCXA, NOLIBMAC, DXREF, LOOSE, NOFOLD, INTEGER, MSRC, UMAP, XL, NOOPCNTS, DISK, NOREL2, TERM, XREF(SHORT), NORLD,
   ESD, ALIGN, BATCH, NOTEST, NORENT, LIST, OBJECT, NODECK, LINECOUNT(59), FLAG(0), SYSPARM(), PEXIT()
  NO OVERRIDING DD NAMES
   474K ALLOCATED TO BUFFER POOL,  137K WOULD BE REQUIRED FOR THIS TO BE AN INCORE ASSEMBLY.
    299 CARDS FROM SYSIN     1277 CARDS FROM SYSLIB     0 SYSUT1 READS
    425 LINES OUTPUT           15 CARDS OUTPUT         30 SYSUT1 WRITES
```

Appendix E

TEST PROGRAMS FOR AMDAHL V7A

There are two tests in this appendix which test out each of the three instructions implemented on the AMDAHL. These tests perform a bubble sort on an incore table of 20 elements, displaying the results on the user's console. The difference between the two tests is that the If-Then (IFT) and If-Then-Else (IFTE) instructions are interchanged. The IFT instruction executes the Else clause zero times while the IFTE instruction executes the Else clause more than zero times.

```
                                                                      PAGE    1
TEST                              EXTERNAL SYMBOL DICTIONARY
                                                  (SLAC V2.9)  ASM H V 02 16.54 10/27/83
SYMBOL   TYPE  ID  ADDR   LENGTH  LD ID FLAGS
TEST      SD 0001 000000 000110         00

                          MACRO AND COPY CODE EXTERNAL SOURCE CROSS REFERENCE

  MACRO DEFINITION FOUND WITHIN INPUT STREAM
     MEMBER(S) USED - IFT      IFTE     WHILE

  CMSLIB  ON  MNT390
     MEMBER(S) USED - DMSLN    DMSLNC   DMSLND   DMSLNP   DMSLNU   DMSLNY   DMSLNZ   LINEDIT  REGEQU

  OSMACRO  ON  MNT390
     MEMBER(S) USED - RETURN   SAVE

                                                                      PAGE    2
TEST     TEST NEW INSTRUCTIONS
  ACTIVE USINGS - NONE
  LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT          (SLAC V2.9)  ASM H V 02 16.54 10/27/83
                                      2 * THIS PROGRAM TESTS OUT THE 'IF' AND 'WHILE' INSTRUCTIONS          TES00020
                                      3 *                                                                   TES00030
                                      4 * THE PROGRAM IS A SIMPLE BUBBLE SORT ON 20 RANDOM ELEMENTS.        TES00040

                                      6          MACRO ,                                                    TES00060
                                      7 &LABEL   IFTE  &MASK,&LINK,&SCON1,&SCON2                            TES00070
                                      8 &LABEL   DC    X'F4',AL.4(&MASK,&LINK),S(&SCON1,&SCON2)             TES00080
                                      9          MEND  ,                                                    TES00090

                                     11          MACRO ,                                                    TES00110
                                     12 &LABEL   WHILE &MASK,&LINK,&SCON                                    TES00120
                                     13 &LABEL   DC    X'A0',AL.4(&MASK,&LINK),S(&SCON)                     TES00130
                                     14          MEND  ,                                                    TES00140

                                     16          MACRO ,                                                    TES00160
                                     17 &LABEL   IFT   &MASK,&LINK,&SCON                                    TES00170
                                     18 &LABEL   DC    X'A1',AL.4(&MASK,&LINK),S(&SCON)                     TES00180
                                     19          MEND  ,                                                    TES00190
                                                                                                           PAGE    3
TEST     BUBBLE SORT PROGRAM
  ACTIVE USINGS - NONE
  LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT          (SLAC V2.9)  ASM H V 02 16.54 10/27/83
000000                               21 TEST    START ,                                                    TES00210
                           00002     23 HIGH    EQU   X'02'                                                TES00230
                           00007     24 FALSE   EQU   X'07'                                                TES00240
                                     26         REGEQU ,                                                   TES00260
                                     27+        PUSH  PRINT                                                01-REGEQ
                                     28+*
                                     29+***     SYMBOLIC REGISTER EQUATES
                                     30+*
                                     32+* GENERAL PURPOSE REGISTERS                                        01-REGEQ
                           00000     34+R0      EQU   0                                                    01-REGEQ
                           00001     35+R1      EQU   1                                                    01-REGEQ
                           00002     36+R2      EQU   2                                                    01-REGEQ
                           00003     37+R3      EQU   3                                                    01-REGEQ
                           00004     38+R4      EQU   4                                                    01-REGEQ
                           00005     39+R5      EQU   5                                                    01-REGEQ
```

```
                        00006    40+R6       EQU   6                                                    01-REGEQ
                        00007    41+R7       EQU   7                                                    01-REGEQ
                        00008    42+R8       EQU   8                                                    01-REGEQ
                        00009    43+R9       EQU   9                                                    01-REGEQ
                        0000A    44+R10      EQU   10                                                   01-REGEQ
                        0000B    45+R11      EQU   11                                                   01-REGEQ
                        0000C    46+R12      EQU   12                                                   01-REGEQ
                        0000D    47+R13      EQU   13                                                   01-REGEQ
                        0000E    48+R14      EQU   14                                                   01-REGEQ
                        0000F    49+R15      EQU   15                                                   01-REGEQ
                                 51+* FLOATING POINT REGISTERS                                          01-REGEQ
                        00000    53+F0       EQU   0                                                    01-REGEQ
                        00002    54+F2       EQU   2                                                    01-REGEQ
                        00004    55+F4       EQU   4                                                    01-REGEQ
                        00006    56+F6       EQU   6                                                    01-REGEQ
                                 58+* EXTENDED CONTROL REGISTERS                                        01-REGEQ
                        00000    60+C0       EQU   0                                                    01-REGEQ
                        00001    61+C1       EQU   1                                                    01-REGEQ
                        00002    62+C2       EQU   2                                                    01-REGEQ
                        00003    63+C3       EQU   3                                                    01-REGEQ
                        00004    64+C4       EQU   4                                                    01-REGEQ
                        00005    65+C5       EQU   5                                                    01-REGEQ
                        00006    66+C6       EQU   6                                                    01-REGEQ
                        00007    67+C7       EQU   7                                                    01-REGEQ
                        00008    68+C8       EQU   8                                                    01-REGEQ
                        00009    69+C9       EQU   9                                                    01-REGEQ
                        0000A    70+C10      EQU   10                                                   01-REGEQ
                        0000B    71+C11      EQU   11                                                   01-REGEQ
                        0000C    72+C12      EQU   12                                                   01-REGEQ
                        0000D    73+C13      EQU   13                                                   01-REGEQ
                        0000E    74+C14      EQU   14                                                   01-REGEQ
                        0000F    75+C15      EQU   15                                                   01-REGEQ
TEST     BUBBLE SORT PROGRAM                                                                          PAGE    4
  ACTIVE USINGS - NONE                                                (SLAC V2.9)  ASM H V 02  16.54 10/27/83
  LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT                                           01-REGEQ
                                      77+         POP   PRINT                                          TES00280
                                      79          SAVE  (14,12),T,*      SAVE RESISTERS                01-SAVE
000000 47F0 F00A                0000A 80+         B     10(0,15)                                       01-SAVE
000004 04                             81+         DC    AL1(4)                                         01-SAVE
000005 E3C5E2E3                       82+         DC    CL4'TEST'              IDENTIFIER              01-SAVE
000009 00                                                                      SAVE REGISTERS          01-SAVE
00000A 90EC D00C                0000C 83+         STM   14,12,12(13)
                                      84          LR    R12,R15          ESTABLISH ADDRESSABILITY      TES00290
00000E 18CF                           85          USING TEST,R12                                       TES00300
                                00000 87          SR    R9,R9            SORT COUNT                   TES00320
000010 1B99                           88          BAL   R11,SORT         SORT THE NUMBERS             TES00330
000012 45B0 C07A                0007A 89          WHILE FALSE,R11,SORT                                 TES00340
                                      90+         DC    X'A0',AL.4(FALSE,R11),S(SORT)                  01-00013
000016 A07BC07A                       92          LM    R4,R5,=A(NUMBERS,COUNT)                        TES00360
00001A 9845 C100                00100 93 DISPLAY  LINEDIT TEXT='.....',DOT=NO,SUB=(DECA,(R4))        TES00370
                                      94+DISPLAY  DS    0H                                             02-DMSLN
00001E                                                                                                 02-DMSLN
00001E 4510 C02C                0002C 95+         BAL   1,DMSA0005                                     02-DMSLN
000022 1081                           96+         DC    AL1(16,129)              FLAG BYTES            03-DMSLN
000024 C1                             97+         DC    AL1(DMSC0005)
```

```
000025 05
000026 4B4B4B4B4B
                                00005
00002C
                                000C1
00002C 1804
00002E 0ACB
000030 FFFA
000032 4140 4004                00004
000036 4650 C01E                0001E


00003A 4510 C066                00066
00003E 1081
000040 81
000041 23
000042 C6C1D3E2C540C3D3
                                00023
000066
                                00081
000066 1809
000068 0ACB
00006A FFFA

00006C 98EC D00C                0000C
000070 92FF D00C        0000C
000074 41F0 0000                00000
000078 07FE
TEST     SORT ROUTINE
  ACTIVE USINGS - (TEST,R12)
  LOC  OBJECT CODE     ADDR1 ADDR2


00007A
00007A 1866
00007C 9845 C108                00108
000080 5820 4000                00000
000084 5920 4004                00004

000088 F42AC0A0C09A
00008E 4140 4004                00004
000092 4650 C080                00080
000096 1266
000098 07FB

00009A 4190 9001                00001
00009E 07FA
0000A0 4160 0001                00001
0000A4 D203 4000 4004 00000 00004
0000AA 5020 4004                00004
0000AE 07FA
TEST     WORKAREAS, CONSTANTS, ETC....
```

```
 98+DMST0005 DC    AL1(DMSL0005)          LENGTH OF MESSAGE TEXT                 02-DMSLN
 99+         DC    C'.....'                                                      02-DMSLN
100+DMSL0005 EQU   *-DMST0005-1           TEXT LENGTH                            02-DMSLN
101+DMSA0005 DS    0H                                                            02-DMSLN
102+DMSC0005 EQU   193                     SUBSTITUTION CODE                      04-DMSLN
103+         LR    0,R4                                                          05-DMSLN
104+         SVC   203                                                           03-DMSLN
105+         DC    H'-6'                                                         03-DMSLN
106          LA    R4,4(,R4)          DISPLAY ALL THE NUMBERS                    TES00380
107          BCT   R5,DISPLAY                                                    TES00390
108          LINEDIT TEXT='FALSE CLAUSE EXECUTED ..... TIMES',DOT=NO,           XTES00400
               SUB=(DEC,(R9))                                                    TES00410
109+         BAL   1,DMSA0037                                                    02-DMSLN
110+         DC    AL1(16,129)              FLAG BYTES                           02-DMSLN
111+         DC    AL1(DMSC0037)                                                 03-DMSLN
112+DMST0037 DC    AL1(DMSL0037)            LENGTH OF MESSAGE TEXT               02-DMSLN
113+         DC    C'FALSE CLAUSE EXECUTED ..... TIMES'                         02-DMSLN
114+DMSL0037 EQU   *-DMST0037-1             TEXT LENGTH                          02-DMSLN
115+DMSA0037 DS    0H                                                           02-DMSLN
116+DMSC0037 EQU   129                      SUBSTITUTION CODE                    04-DMSLN
117+         LR    0,R9                                                         05-DMSLN
118+         SVC   203                                                          03-DMSLN
119+         DC    H'-6'                                                        03-DMSLN
121          RETURN (14,12),T,RC=0                                              TES00430
122+         LM    14,12,12(13)                RESTORE THE REGISTERS            01-RETUR
123+         MVI   12(13),X'FF'                SET RETURN INDICATION            01-RETUR
124+         LA    15,0(0,0)                   LOAD RETURN CODE                 01-RETUR
125+         BR    14                          RETURN                           01-RETUR
                                                                               PAGE    5
```

```
                                                (SLAC V2.9)  ASM H V 02 16.54 10/27/83
127 * THIS ROUTINE PERFORMS ONE PASS THROUGH A LIST.                            TES00450
128 * IF THE LIST IS SORTED IT RETURNS TRUE.                                    TES00460
129 * IF THE LIST IS NOT SORTED, IT RETURNS FALSE AND SWAPS THE                 TES00470
130 *    OFFENDING ELEMENT.                                                     TES00480
132 SORT     DS    0H                                                          TES00500
133          SR    R6,R6                    ASSUME TRUE                        TES00510
134          LM    R4,R5,=A(NUMBERS,COUNT-1)    ADDRESS & COUNT                TES00520
135 NEXTNUM  L     R2,0(,R4)                GET THE FIRST NUMBER               TES00530
136          C     R2,4(,R4)                COMPARE                            TES00540
137          IFTE  HIGH,R10,SWAP,NOSWAP   IF HIGH SWAP, ELSE DON'T             TES00550
138+         DC    X'F4',AL.4(HIGH,R10),S(SWAP,NOSWAP)                         01-00008
139          LA    R4,4(,R4)                RUN THROUGH THE ENTIRE LOOP        TES00560
140          BCT   R5,NEXTNUM                                                  TES00570
142          LTR   R6,R6                    SET THE CONDITION CODE             TES00590
143          BR    R11                      RETURN TO CALLER                   TES00600

145 NOSWAP   LA    R9,1(,R9)                NUMBER OF TIMES TEST IS FALSE      TES00620
146          BR    R10                                                         TES00630
148 SWAP     LA    R6,1                     NOW FALSE                          TES00650
149          MVC   0(4,R4),4(R4)                                               TES00660
150          ST    R2,4(,R4)                SWAP THE NUMBERS                   TES00670
151          BR    R10                                                         TES00680
                                                                               PAGE    6
```

```
 ACTIVE USINGS - (TEST,R12)                                                    (SLAC V2.9)  ASM H V 02 16.54 10/27/83
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                                          TES00700
0000B0 0000000400000008            153 NUMBERS  DC    A(4,8,2,-25,21,-4,66,33,1,18,0,100,99,14,7,9)             TES00710
0000F0 0000001SFFFFFFEF            154          DC    A(21,-17,25,1984).                                        TES00720
                          00014    155 COUNT    EQU   (*-NUMBERS)/4                                             TES00740
                                   157          END   .
000100 000000B000000014            158                =A(NUMBERS,COUNT)
000108 000000B000000013            159                =A(NUMBERS,COUNT-1)
                                                                                                                  PAGE    7
TEST                                                CROSS REFERENCE                 (SLAC V2.9)  ASM H V 02 16.54 10/27/83
SYMBOL     LEN    VALUE  DEFN  REFERENCES
COUNT      00001 00000014 0155  0158 0159
DISPLAY    00002   00001E 0094  0107
DMSA0005   00002   00002C 0101  0095
DMSA0037   00002   000066 0115  0109
DMSC0005   00001 000000C1 0102  0097
DMSC0037   00001 00000081 0116  0111
DMSL0005   00001 00000005 0100  0098
DMSL0037   00001 00000023 0114  0112
DMST0005   00001   000025 0098  0100
DMST0037   00001   000041 0112  0114
FALSE      00001 00000007 0024  0090
HIGH       00001 00000002 0023  0138
NEXTNUM    00004   000080 0135  0140
NOSWAP     00004   00009A 0145  0138
NUMBERS    00004   0000B0 0153  0155 0158 0159
R10        00001 0000000A 0044  0138 0146 0151
R11        00001 0000000B 0045  0088 0090 0143
R12        00001 0000000C 0046  0084 0085
R15        00001 0000000F 0049  0084
R2         00001 00000002 0036  0135 0136 0150
R4         00001 00000004 0038  0092 0103 0106 0106 0134 0135 0136 0139 0139 0149 0149 0150
R5         00001 00000005 0039  0092 0107 0134 0140
R6         00001 00000006 0040  0133 0133 0142 0142 0148
R9         00001 00000009 0043  0087 0087 0117 0145 0145
SORT       00002   00007A 0132  0088 0090
SWAP       00004   0000A0 0148  0138
TEST       00001 00000000 0021  0085
=A(NUMBERS,COUNT)
           00004   000100 0158  0092
=A(NUMBERS,COUNT-1)
           00004   000108 0159  0134
                                                                                                                  PAGE    8
TEST                                                                            (SLAC V2.9)  ASM H V 02 16.54 10/27/83
                                                                                      BUFFER USAGE - 470K,  41K

    NO STATEMENTS FLAGGED IN THIS ASSEMBLY
   OVERRIDING PARAMETERS-  NONE
   OPTIONS FOR THIS ASSEMBLY
    OPCXA, NOLIBMAC, DXREF, LOOSE, NOFOLD, INTEGER, MSRC, UMAP, XL, NOOPCNTS, DISK, NOREL2, TERM, XREF(SHORT), NORLD,
    ESD, ALIGN, BATCH, NOTEST, NORENT, LIST, OBJECT, NODECK, LINECOUNT(59), FLAG(0), SYSPARM(), PEXIT()
   NO OVERRIDING DD NAMES
    470K ALLOCATED TO BUFFER POOL,   41K WOULD BE REQUIRED FOR THIS TO BE AN INCORE ASSEMBLY.
      74 CARDS FROM SYSIN     686 CARDS FROM SYSLIB      0 SYSUT1 READS
     224 LINES OUTPUT          10 CARDS OUTPUT           7 SYSUT1 WRITES
```

```
-25
-17
-4
0
1
2
4
7
8
9
14
18
21
21
25
33
66
99
100
1984
FALSE CLAUSE EXECUTED 250 TIMES
```

```
TEST                                  EXTERNAL SYMBOL DICTIONARY                                  PAGE    1
SYMBOL   TYPE  ID  ADDR   LENGTH  LD ID FLAGS                    (SLAC V2.9)  ASM H V 02 16.55 10/27/83
TEST      SD  0001 000000 000110          00

                                MACRO AND COPY CODE EXTERNAL SOURCE CROSS REFERENCE

  MACRO DEFINITION FOUND WITHIN INPUT STREAM
     MEMBER(S) USED - IFT       IFTE      WHILE

  CMSLIB  ON  MNT390
     MEMBER(S) USED - DMSLN     DMSLNC    DMSLND    DMSLNP    DMSLNU    DMSLNY    DMSLNZ    LINEDIT   REGEQU

  OSMACRO  ON  MNT390
     MEMBER(S) USED - RETURN    SAVE

                                                                                                  PAGE    2
TEST      TEST NEW INSTRUCTIONS
  ACTIVE USINGS - NONE
  LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT                (SLAC V2.9)  ASM H V 02 16.55 10/27/83
                                      2 * THIS PROGRAM TESTS OUT THE 'IF' AND 'WHILE' INSTRUCTIONS                TES00020
                                      3 *                                                                         TES00030
                                      4 * THE PROGRAM IS A SIMPLE BUBBLE SORT ON 20 RANDOM ELEMENTS.              TES00040
                                      6         MACRO ,                                                           TES00060
                                      7 &LABEL  IFTE  &MASK,&LINK,&SCON1,&SCON2                                   TES00070
                                      8 &LABEL  DC    X'F4',AL.4(&MASK,&LINK),S(&SCON1,&SCON2)                    TES00080
                                      9         MEND  .                                                           TES00090
                                     11         MACRO ,,                                                          TES00110
                                     12 &LABEL  WHILE &MASK,&LINK,&SCON                                          TES00120
                                     13 &LABEL  DC    X'A0',AL.4(&MASK,&LINK),S(&SCON)                           TES00130
                                     14         MEND  .                                                           TES00140
                                     16         MACRO ,,                                                          TES00160
                                     17 &LABEL  IFT   &MASK,&LINK,&SCON                                          TES00170
                                     18 &LABEL  DC    X'A1',AL.4(&MASK,&LINK),S(&SCON)                           TES00180
                                     19         MEND  .                                                           TES00190
                                                                                                               PAGE    3
TEST      BUBBLE SORT PROGRAM
  ACTIVE USINGS - NONE
  LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT                (SLAC V2.9)  ASM H V 02 16.55 10/27/83
                                     21 TEST   START .                                                          TES00210
000000                               23 HIGH   EQU   X'02'                                                      TES00230
                              00002  24 FALSE  EQU   X'07'                                                      TES00240
                              00007  26        REGEQU .                                                         TES00260
                                     27+       PUSH  PRINT                                                      01-REGEQ
                                     28+*
                                     29+***    SYMBOLIC REGISTER EQUATES
                                     30+*
                                     32+* GENERAL PURPOSE REGISTERS                                             01-REGEQ
                              00000  34+R0     EQU   0                                                          01-REGEQ
                              00001  35+R1     EQU   1                                                          01-REGEQ
                              00002  36+R2     EQU   2                                                          01-REGEQ
                              00003  37+R3     EQU   3                                                          01-REGEQ
                              00004  38+R4     EQU   4                                                          01-REGEQ
                              00005  39+R5     EQU   5                                                          01-REGEQ

                                  - 155 -
```

```
                        00006    40+R6        EQU   6                                                    01-REGEQ
                        00007    41+R7        EQU   7                                                    01-REGEQ
                        00008    42+R8        EQU   8                                                    01-REGEQ
                        00009    43+R9        EQU   9                                                    01-REGEQ
                        0000A    44+R10       EQU   10                                                   01-REGEQ
                        0000B    45+R11       EQU   11                                                   01-REGEQ
                        0000C    46+R12       EQU   12                                                   01-REGEQ
                        0000D    47+R13       EQU   13                                                   01-REGEQ
                        0000E    48+R14       EQU   14                                                   01-REGEQ
                        0000F    49+R15       EQU   15                                                   01-REGEQ
                                 51+* FLOATING POINT REGISTERS                                           01-REGEQ
                        00000    53+F0        EQU   0                                                    01-REGEQ
                        00002    54+F2        EQU   2                                                    01-REGEQ
                        00004    55+F4        EQU   4                                                    01-REGEQ
                        00006    56+F6        EQU   6                                                    01-REGEQ
                                 58+* EXTENDED CONTROL REGISTERS                                         01-REGEQ
                        00000    60+C0        EQU   0                                                    01-REGEQ
                        00001    61+C1        EQU   1                                                    01-REGEQ
                        00002    62+C2        EQU   2                                                    01-REGEQ
                        00003    63+C3        EQU   3                                                    01-REGEQ
                        00004    64+C4        EQU   4                                                    01-REGEQ
                        00005    65+C5        EQU   5                                                    01-REGEQ
                        00006    66+C6        EQU   6                                                    01-REGEQ
                        00007    67+C7        EQU   7                                                    01-REGEQ
                        00008    68+C8        EQU   8                                                    01-REGEQ
                        00009    69+C9        EQU   9                                                    01-REGEQ
                        0000A    70+C10       EQU   10                                                   01-REGEQ
                        0000B    71+C11       EQU   11                                                   01-REGEQ
                        0000C    72+C12       EQU   12                                                   01-REGEQ
                        0000D    73+C13       EQU   13                                                   01-REGEQ
                        0000E    74+C14       EQU   14                                                   01-REGEQ
                        0000F    75+C15       EQU   15                                                   01-REGEQ
TEST     BUBBLE SORT PROGRAM                                                                            PAGE    4
 ACTIVE USINGS - NONE                                  (SLAC V2.9)  ASM H V 02 16.55 10/27/83
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                               01-REGEQ
                                    77+         POP   PRINT                                              TES00280
                                    79          SAVE  (14,12),T,*      SAVE RESISTERS                    01-SAVE
000000 47F0 F00A                    80+         B     10(0,15)                 BRANCH AROUND ID           01-SAVE
000004 04                           81+         DC    AL1(4)                                              01-SAVE
000005 E3C5E2E3                     82+         DC    CL4'TEST'                IDENTIFIER                 01-SAVE
000009 00
00000A 90EC D00C            0000C   83+         STM   14,12,12(13)             SAVE REGISTERS             01-SAVE
00000E 18CF                         84          LR    R12,R15                                             TES00290
                            00000   85          USING TEST,R12          ESTABLISH ADDRESSABILITY          TES00300
                                    87          SR    R9,R9                                               TES00320
000010 1B99                                                               SORT COUNT                        TES00330
000012 45B0 C07A            0007A   88          BAL   R11,SORT            SORT THE NUMBERS                  TES00340
                                    89          WHILE FALSE,R11,SORT                                      01-00013
000016 A07BC07A                     90+         DC    X'A0',AL.4(FALSE,R11),S(SORT)                       TES00360
00001A 9845 C100            00100   92          LM    R4,R5,=A(NUMBERS,COUNT)                             TES00370
                                    93 DISPLAY  LINEDIT TEXT='.....',DOT=NO,SUB=(DECA,(R4))               02-DMSLN
00001E                              94+DISPLAY  DS    0H                                                  02-DMSLN
00001E 4510 C02C            0002C   95+         BAL   1,DMSA0005                                          02-DMSLN
000022 1081                         96+         DC    AL1(16,129)              FLAG BYTES                 03-DMSLN
000024 C1                           97+         DC    AL1(DMSC0005)
```

```
000025 05                                                98+DMST0005 DC    AL1(DMSL0005)          LENGTH OF MESSAGE TEXT          02-DMSLN
000026 4B4B4B4B4B                                        99+         DC    C'.....'                                               02-DMSLN
                                  00005                 100+DMSL0005 EQU    *-DMST0005-1           TEXT LENGTH                     02-DMSLN
00002C                                                  101+DMSA0005 DS     0H                                                     02-DMSLN
                                  000C1                 102+DMSC0005 EQU    193                    SUBSTITUTION CODE               04-DMSLN
00002C 1804                                             103+        LR    0,R4                                                   05-DMSLN
00002E 0ACB                                             104+        SVC   203                                                    03-DMSLN
000030 FFFA                                             105+        DC    H'-6'                                                  03-DMSLN
000032 4140 4004                  00004                 106         LA    R4,4(,R4)          DISPLAY ALL THE NUMBERS             TES00380
000036 4650 C01E                  0001E                 107         BCT   R5,DISPLAY                                           TES00390
                                                        108         LINEDIT TEXT='FALSE CLAUSE EXECUTED ..... TIMES',DOT=NO,   XTES00400
                                                                      SUB=(DEC,(R9))                                           TES00410
00003A 4510 C066                  00066                 109+        BAL   1,DMSA0037                                         02-DMSLN
00003E 1081                                             110+        DC    AL1(16,129)            FLAG BYTES                      02-DMSLN
000040 81                                               111+        DC    AL1(DMSC0037)                                          03-DMSLN
000041 23                                               112+DMST0037 DC     AL1(DMSL0037)          LENGTH OF MESSAGE TEXT          02-DMSLN
000042 C6C1D3E2C540C3D3                                 113+        DC    C'FALSE CLAUSE EXECUTED ..... TIMES'                   02-DMSLN
                                  00023                 114+DMSL0037 EQU    *-DMST0037-1           TEXT LENGTH                     02-DMSLN
000066                                                  115+DMSA0037 DS     0H                                                     02-DMSLN
                                  00081                 116+DMSC0037 EQU    129                    SUBSTITUTION CODE               04-DMSLN
000066 1809                                             117+        LR    0,R9                                                   05-DMSLN
000068 0ACB                                             118+        SVC   203                                                    03-DMSLN
00006A FFFA                                             119+        DC    H'-6'                                                  03-DMSLN
                                                        121         RETURN (14,12),T,RC=0                                      TES00430
00006C 98EC D00C                  0000C                 122+        LM    14,12,12(13)                   RESTORE THE REGISTERS   01-RETUR
000070 92FF D00C        0000C                           123+        MVI   12(13),X'FF'                   SET RETURN INDICATION   01-RETUR
000074 41F0 0000                  00000                 124+        LA    15,0(0,0)                      LOAD RETURN CODE        01-RETUR
000078 07FE                                             125+        BR    14                             RETURN                  01-RETUR
TEST     SORT ROUTINE                                                                                                        PAGE    5
  ACTIVE USINGS - (TEST,R12)
  LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT                                (SLAC V2.9)  ASM H V 02 16.55 10/27/83
                                    127 * THIS ROUTINE PERFORMS ONE PASS THROUGH A LIST.                                      TES00450
                                    128 * IF THE LIST IS SORTED IT RETURNS TRUE.                                              TES00460
                                    129 * IF THE LIST IS NOT SORTED, IT RETURNS FALSE AND SWAPS THE                           TES00470
                                    130 *    OFFENDING ELEMENT.                                                               TES00480
00007A                              132 SORT    DS    0H                                                                      TES00500
00007A 1B66                         133         SR    R6,R6                  ASSUME TRUE                                      TES00510
00007C 9845 C108          00108     134         LM    R4,R5,=A(NUMBERS,COUNT-1)    ADDRESS & COUNT                     TES00520
000080 5820 4000          00000     135 NEXTNUM L     R2,0(,R4)              GET THE FIRST NUMBER                             TES00530
000084 5920 4004          00004     136         C     R2,4(,R4)              COMPARE                                          TES00540
                                    137         IFJ   HIGH,R10,SWAP          IF HIGH SWAP, ELSE DON'T                         TES00550
000088 A12AC09E                     138+        DC    X'A1',AL.4(HIGH,R10),S(SWAP)                                           01-00018
00008C 4140 4004          00004     139         LA    R4,4(,R4)              RUN THROUGH THE ENTIRE LOOP                      TES00560
000090 4650 C080          00080     140         BCT   R5,NEXTNUM                                                              TES00570
000094 1266                         142         LTR   R6,R6                  SET THE CONDITION CODE                           TES00590
000096 07FB                         143         BR    R11                    RETURN TO CALLER                                 TES00600

000098 4190 9001          00001     145 NOSWAP  LA    R9,1(,R9)              NUMBER OF TIMES TEST IS FALSE                    TES00620
00009C 07FA                         146         BR    R10                                                                     TES00630
00009E 4160 0001          00001     148 SWAP    LA    R6,1                   NOW FALSE                                        TES00650
0000A2 D203 4000 4004 00000 00004   149         MVC   0(4,R4),4(R4)                                                           TES00660
0000A8 5020 4004          00004     150         ST    R2,4(;R4)              SWAP THE NUMBERS                                 TES00670
0000AC 07FA                         151         BR    R10                                                                     TES00680
TEST     WORKAREAS, CONSTANTS, ETC....                                                                                       PAGE    6
```

```
  ACTIVE USINGS - (TEST,R12)
   LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                    (SLAC V2.9)  ASM H V 02 16.55 10/27/83
0000AE 0000                                                                                                    TES00700
0000B0 0000000400000008                153 NUMBERS  DC    A(4,8,2,-25,21,-4,66,33.1,18,0,100,99,14,7,9)      TES00710
0000F0 00000015FFFFFFEF                154           DC    A(21,-17,25,1984)                                  TES00720
                              00014    155 COUNT     EQU   (*-NUMBERS)/4                                      TES00740
                                       157           END   .
000100 000000B000000014                158                 =A(NUMBERS,COUNT)
000108 000000B000000013                159                 =A(NUMBERS,COUNT-1)

                                                                                                      PAGE    7
TEST                                          CROSS REFERENCE
SYMBOL    LEN    VALUE  DEFN  REFERENCES                                  (SLAC V2.9)  ASM H V 02 16.55 10/27/83
COUNT     00001 00000014 0155  0158 0159
DISPLAY   00002   00001E 0094  0107
DMSA0005  00002   00002C 0101  0095
DMSA0037  00002   000066 0115  0109
DMSC0005  00001 000000C1 0102  0097
DMSC0037  00001 00000081 0116  0111
DMSL0005  00001 00000005 0100  0098
DMSL0037  00001 00000023 0114  0112
DMST0005  00001   000025 0098  0100
DMST0037  00001   000041 0112  0114
FALSE     00001 00000007 0024  0090
HIGH      00001 00000002 0023  0138
NEXTNUM   00004   000080 0135  0140
NUMBERS   00004   0000B0 0153  0155 0158 0159
R10       00001 0000000A 0044  0138 0146 0151
R11       00001 0000000B 0045  0088 0090 0143
R12       00001 0000000C 0046  0084 0085
R15       00001 0000000F 0049  0084
R2        00001 00000002 0036  0135 0136 0150
R4        00001 00000004 0038  0092 0103 0106 0106 0134 0135 0136 0139 0139 0149 0149 0150
R5        00001 00000005 0039  0092 0107 0134 0140
R6        00001 00000006 0040  0133 0133 0142 0142 0148
R9        00001 00000009 0043  0087 0087 0117 0145 0145
SORT      00002   00007A 0132  0088 0090
SWAP      00004   00009E 0148  0138
TEST      00001 00000000 0021  0085
=A(NUMBERS,COUNT)
          00004   000100 0158  0092
=A(NUMBERS,COUNT-1)
          00004   000108 0159  0134

                                                                                                      PAGE    8
TEST                                                                  (SLAC V2.9)  ASM H V 02 16.55 10/27/83
                                                                        BUFFER USAGE - 467K,  41K

    NO STATEMENTS FLAGGED IN THIS ASSEMBLY
   OVERRIDING PARAMETERS-  NONE
   OPTIONS FOR THIS ASSEMBLY
    OPCXA, NOLIBMAC, DXREF, LOOSE, NOFOLD, INTEGER, MSRC, UMAP, XL, NOOPCNTS, DISK, NOREL2, TERM, XREF(SHORT), NORLD,
    ESD, ALIGN, BATCH, NOTEST, NORENT, LIST, OBJECT, NODECK, LINECOUNT(59), FLAG(0), SYSPARM(), PEXIT()
   NO OVERRIDING DD NAMES
    467K ALLOCATED TO BUFFER POOL.   41K WOULD BE REQUIRED FOR THIS TO BE AN INCORE ASSEMBLY.
      74 CARDS FROM SYSIN     686 CARDS FROM SYSLIB      0 SYSUT1 READS
     224 LINES OUTPUT          10 CARDS OUTPUT           7 SYSUT1 WRITES
```

```
-25
-17
-4
0
1
2
4
7
8
9
14
18
21
21
25
33
66
99
100
1984
FALSE CLAUSE EXECUTED 0 TIMES
```

Appendix F

DISASSEMBLED I.B.M. OBJECT CODE

This appendix contains the disassembled version of the second test program found in the previous appendix. This information was generated using a modified version of the HMASPZAP utility available on the OS/VS1 operating system. It was modified to add the three op-codes IFT, IFTE and WHILE.

```
HMASPZAP  INSPECTS, MODIFIES, AND DUMPS CSECTS OR SPECIFIC DATA RECORDS ON DIRECT ACCESS STORAGE.
 DUMPT TEST TEST

**CCHHR- 005A001310   RECORD LENGTH- 0110          MEMBER NAME  TEST      CSECT NAME  TEST
 000000   47F0 F00A  04E3 C5E2  E300 90EC  D00C 18CF  1B99 45B0  C07A A07B  C07A 9845  C100 4510
          BC   SRP   SPM               STM       LR   SR   BAL        WHILE      LM         BAL
 000020   C02C 1081  C105 4B4B  4B4B 4B00  1804 0ACB  FFFA 4140  4004 4650  C01E 4510  C066 1081
               LPR        SH    SH   SH    LR   SVC        LA    STH  BCT        BAL        LPR
 000040   8123 C6C1  D3E2 C540  C3D3 C1E4  E2C5 40C5  E7C5 C3E4  E3C5 C440  4B4B 4B4B  4B4B 4B40
                    MVZ                         STH                        SH   SH    SH   SH
 000060   E3C9 D4C5  E200 1809  0ACB FFFA  98EC D00C  92FF D00C  41F0 0000  07FE 1B66  9845 C108
               NC         LR    SVC        LM         MVI        LA         BCR  SR    LM
 000080   5820 4000  5920 4004  A12A C09E  4140 4004  4650 C080  1266 07FB  4190 9001  07FA 4160
          L    STH   C    STH   IFT       LA   STH   BCT        LTR  BCR   LA   STM   BCR  LA
 0000A0   0001 D203  4000 4004  5020 4004  07FA 0000  0000 0004  0000 0008  0000 0002  FFFF FFE7
               MVC   STH  STH   ST   STH   BCR
 0000C0   0000 0015  FFFF FFFC  0000 0042  0000 0021  0000 0001  0000 0012  0000 0000  0000 0064

 0000E0   0000 0063  0000 000E  0000 0007  0000 0009  0000 0015  FFFF FFEF  0000 0019  0000 07C0
                                                                                       BCR
 000100   0000 00B0  0000 0014  0000 00B0  0000 0013

HMA113I COMPLETED DUMP REQUIREMENTS

HMA100I HMASPZAP PROCESSING COMPLETED
```

BIBLIOGRAPHY

1. OS/VS - DOS/VSE - VM/370 Assembler Language, 5th Edition, International Business Machines, Order No. GC33-4010, March 1979.

2. A Guide to using the University of Waterloo level G assembler for the IBM System/360 or System/370, 10th Edition, Department of Computer Services, Univerity of Waterloo, June 1976.

3. PDP-11 Macro-11 Language Reference Manual, Digital Equipment Corperation, Maynard Massachusetts, Order No. AA-5075A-TC, August 1977.

4. VAX-11 Macro Language Reference Manual, Digital Equipment Corperation, Maynard Massachusetts, Order No. AA-D032D-TE, May 1982.

5. GoTo Statement Considered Harmful Dijkstra, Edsger W.; Letter to the Editor, Communication of the Association for Computing Machinery (CACM), March 1968.

6. Flow Diagrams, Turing Machines, and Languages with only Two Formation Rules, Boehm, Corrado and Jacopini, Giuseppi. CACM, May 1966.

7. A Firmware Implementation of Block Structured Programming. Raymond, Jacques. Technical Report TR78-04, University of Ottawa, Department of Computer Science, 1978.

8. Revised Report on the Algorithmic Language Algol-60. Naur, Peter (Editor). CACM, January, 1963.

9. The Programming Language Landscape. Ledgard, Henry and Marcotty, Micheal. S.R.A Inc., 1981.

10. Macro Reference Manual, Roberts, G. Department of Computing Services, University of Waterloo, September 1980.

11. Hibal Structured Programming Macros, Stanford Linear Accelerating Center internal document, Stanford University, 1974.

12. Programming in Modula-2, 2nd Edition Wirth, Nicklaus. Springer-Verlag, 1983.

13. Pascal Users Manual and Report, Jansen, Karen and Wirth, Nicklaus. Springer-Verlag, 1978.

14. PL360, A programming language for the 360 Computers, Wirth, Nicklaus. Journal of the Association for Computing Machinery (JACM), 1968.

15. Toward a Theory of Test Data Selection, Gerhart, Susan L. and Goodenough, John B. IEEE Transactions on Software Engineering, Vol. SE-1, No. 2, June 1975.

16. A Generalised Control Structure and its Formal Definition, Parnas, David L. CACM, Vol. 26, No. 8, August 1983.

17. IBM System/370 Principles of Operation. International Business Machines. Order No. GA22-7000, October 1981.

18. PERQ Micro-Programmer's Guide. Rosen, Brian and Strait, John P. Three Rivers Computer Corporation, 8th Febuary 1982.

19. PERQ QCode Reference Manual. Barel, Miles A. and Strait, John P. Three Rivers Computer Corporation, 4th Febuary 1982.

20. Advances in Computer Architecture, 2nd edition. Myers, Glenford J. John Wiley & Sons publishers, 1982.

21. The New Math of Computer Programming. Mills, H.D. CACM Vol. 18 No. 1, Jan. 1975.

22. Pascal 8000 Reference Manual. Australian Atomic Energy Commission, Version 2.0A, 1980.

23. Pascal/VS Programmers Guide. International Business Machines, Order No. SH20-6162, April 1981.

24. Waterloo Pascal Users Guide & Language Description. Computer Science Group, University of Waterloo, August 1982.