TAXICRINIC PROCESSOR OF ILLIAC III:
SIMULATION MANUAL

by

Mary Ann Kreger

June, 1970

MASTER

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

TAXICRINIC PROCESSOR OF ILLIAC III:
SIMULATION MANUAL*

Volume 1

by

Mary Ann Kreger

June 1970

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

—LEGAL NOTICE—

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Atomic Energy Commission, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

*Supported by Contract AT(11-1)-1018 with the
U.S. Atomic Energy Commission

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

ACKNOWLEDGMENT

The conceptual design of the simulation of the Taxicrinic Processor of the Illiac III is largely the work of Bernard J. Nordmann, Jr., and the author.

Familiarity with the Illiac III Computer System Manual: Taxicrinic Processor Volume I, edited by Bernard J. Nordmann, Jr., 1969, is assumed. Familiarity with Volume II of the manual mentioned above, not yet published, will be necessary to Section 5 of this manual.

Bernard J. Nordmann, Jr. also assisted in the task of editing this volume of the Simulation Manual. Mrs. Betty J. Gunsalus labored over the many revisions and rewrites to type the document.

TABLE OF CONTENTS

1. Introduction
2. Control Point Simulation-Routines.
 - 2.1 The MAIN Program
 - 2.1.1 Example MAIN Program Listing
 - 2.2 CP, CCP Procedure
 - 2.2.1 CP Description
 - 2.2.2 CCP Description
 - 2.2.3 SIMCALL Procedure
 - 2.2.4 CP Listing
 - 2.2.5 CP Variable List
 - 2.3 PRNTSIGS Procedure
 - 2.3.1 PRNTSIGS Description
 - 2.3.2 PRNTSIGS Listing
 - 2.3.3 PRNTSIGS Variable List
 - 2.4 EXTRACT Procedure
 - 2.4.1 EXTRACT Description
 - 2.4.2 EXTRACT Listing
 - 2.4.3 EXTRACT Variable List
 - 2.5 DCLGEN Procedure
 - 2.5.1 DCLGEN Description
 - 2.5.2 DCLGEN Listing
3. Utility Routines
 - 3.1 CLEAR Procedure
 - 3.1.1 CLEAR Description
 - 3.1.2 CLEAR Listing
 - 3.2 PRINT1 Procedure
 - 3.2.1 PRINT1 Description
 - 3.2.2 PRINT1 Listing
 - 3.2.3 PRINT1 Variable List
 - 3.2.4 PRINT1 Sample Output

3.3 REGSØUT Procedure

3.3.1 REGSØUT Description

3.3.2 REGSØUT Listing

3.3.3 REGSØUT Variable List

3.3.4 REGSØUT Sample Output

3.4 PTRSØUT Procedure

3.4.1 PTRSØUT Description

3.4.2 PTRSØUT Listing

3.4.3 PTRSØUT Variable List

3.4.4 PTRSØUT Sample Output

3.5 BRSØUT Procedure

3.5.1 BRSØUT Description

3.5.2 BRSØUT Listing

3.5.3 BRSØUT Variable List

3.5.4 BRSØUT Sample Output

3.6 ØSØUT Procedure

3.6.1 ØSØUT Description

3.6.2 ØSØUT Listing

3.6.3 ØSØUT Variable List

3.6.4 ØSØUT Sample Output

3.7 MEMØUT Procedure

3.7.1 MEMØUT Description

3.7.2 MEMØUT Listing

3.7.3 MEMØUT Variable List

4. Simulating Routines

4.1 FFSET Procedure

4.1.1 FFSET Description

4.1.2 FFSET Listing

4.1.3 FFSET Variable List

- 4.2 CSGEN Procedure
 - 4.2.1 CSGEN Description
 - 4.2.2 CSGEN Listing
 - 4.2.3 CSGEN Variable List
- 4.3 CØNSGEN Procedure
 - 4.3.1 CØNSGEN Description
 - 4.3.2 CØNSGEN Listing
 - 4.3.3 CØNSGEN Variable List
- 4.4 ADDER5 Procedure
 - 4.4.1 ADDER5 Description
 - 4.4.2 ADDER5 Listing
 - 4.4.3 ADDER5 Variable List
- 4.5 ØPSTKIN Procedure
 - 4.5.1 ØPSTKIN Description
 - 4.5.2 ØPSTKIN Listing
 - 4.5.3 ØPSTKIN Variable List
- 4.6 BREGIN Procedure
 - 4.6.1 BREGIN Description
 - 4.6.2 BREGIN Listing
 - 4.6.3 BREGIN Variable List
- 4.7 PREGIN Procedure
 - 4.7.1 PREGIN Description
 - 4.7.2 PREGIN Listing
 - 4.7.3 PREGIN Variable List
 - 4.7.4 ADD1 Description
 - 4.7.5 ADD1 Listing
 - 4.7.6 ADD1 Variable List
- 4.8 RWBGEN Procedure
 - 4.8.1 RWBGEN Description
 - 4.8.2 RWBGEN Listing
 - 4.8.3 RWBGEN Variable List
- 4.9 BØØLSHFT Procedure
 - 4.9.1 BØØLSHFT Description
 - 4.9.2 BØØLSHFT Listing
 - 4.9.3 BØØLSHFT Variable List

- 4.10 PER-IN Procedure
 - 4.10.1 PER-IN Description
 - 4.10.2 PER-IN Listing
 - 4.10.3 PER-IN Variable List
- 4.11 ADD32 Procedure
 - 4.11.1 ADD32 Description
 - 4.11.2 ADD32 Listing
 - 4.11.3 ADD32 Variable List
- 4.12 PER_OUT Procedure
 - 4.12.1 PER_OUT Description
 - 4.12.2 PER_OUT Listing
 - 4.12.3 PER_OUT Variable List
- 4.13 ADDOVF Procedure
 - 4.13.1 ADDOVF Description
 - 4.13.2 ADDOVF Listing
 - 4.13.3 ADDOVF Variable List
- 4.14 COMPARE Procedure
 - 4.14.1 COMPARE Description
 - 4.14.2 COMPARE Listing
 - 4.14.3 COMPARE Variable List
- 5. Control Sequences
 - 5.1 Memory Sequences
 - 5.1.1 MEMDIR, MEMMOD, IMMAC Procedure
 - 5.1.2 PMAC Procedure
 - 5.1.3 READAC Procedure
 - 5.1.4 WRAC Procedure
 - 5.1.5 QUPDATE Procedure
 - 5.2 Operand Stack Sequences
 - 5.2.1 OSENTRY, OVENTRY Procedure
 - 5.2.2 SCRMOD Procedure
 - 5.2.3 OSREAD Procedure
 - 5.2.4 OSWRITE Procedure

6. Environmental Procedures

6.1 MEMLØG Procedure

6.1.1 MEMLØG Description

6.1.2 MEMLØG Listing

6.1.3 MEMLØG Variable List

6.2 XNACC Procedure

6.2.1 XNACC Description

6.2.2 XNACC Listing

6.2.3 XNACC Variable List

6.3 TII1, TII2 Procedure

6.3.1 TII1, TII2 Description

6.3.2 TII1, TII2 Listing

6.3.3 TII1, TII2 Variable List

6.4 MDAT1, MDAT2 Procedure

6.4.1 MDAT1, MDAT2 Description

6.4.2 MDAT1, MDAT2 Listing

6.4.3 MDAT1, MDAT2 Variable List

1. Introduction

The TP logic level simulator, written entirely in PL/I, simulates the Taxicrinic Processor of the ILLIAC III at the signal name level. When completed, the simulator can be used to check the logical consistency of the TP design and eventually provide a maintenance aid in troubleshooting the main frame.

The main program of the simulator is essentially an initiating procedure. It is used currently to initiate a control sequence. The control sequence itself then calls the other necessary control sequences.

The procedures of the simulator can be divided into four classes.

The first class consists of 4 routines; CP (with CCP as another entry), EXTRACT, PRNTSIGS, and DCLGEN. These routines simulate the actions of a control point and of a calling control point. DCLGEN is used to generate a list of signal declarations which will be discussed later.

The second class of procedures simulates the logical units of the TP. Among them are:

FFSET	(Simulates setting and resetting of various flip-flops)
CSGEN	(Cell Size Generator)
CØNSGEN	(Constant Generator)
ADDER5	(5 bit adder)
ØPSTKIN	(operand stack - input to permuter)
BEGIN	(Base Registers - input to permuter)
PBEGIN	(Pointer Stack - input to permuter)
RWBGEN	(Read/Write Byte Generator)
BOOLSHFT	(Boolean Shift operations)
PER-IN	(Permuter input)
ADD32	(32 bit adder)
PER_OUT	(Permuter - output)
ADDØVF	(32 bit adder overflow)
COMPARE	(Greater than, less than, equal comparison operations)

These procedures are explained in Section 4.

The third class of procedures simulate the control sequences (SCRMOD, OSWRITE, OSREAD, OSEENTRY...). This part of the simulation is done on a control point by control point basis, with decision logic between the control points. These procedures are described in Section 5.

The fourth class of procedures are the utility routines which are used to print out contents of registers for diagnostic purposes. These routines are:

OSOUT	(prints out operand stack contents)
BRSOUT	(prints out base register contents)
PTRSOUT	(prints out pointer register contents)
REGSOUT	(prints out certain commonly used registers)
PRINT1	(prints out any specified 36 bit register)
MEMOUT	(prints out contents of memory stage)

Another utility procedure, CLEAR, is used to zero out the contents of all the registers. These are described in Section 3.

The fifth and final class of procedures are the environmental procedures, described in Section 6. They are needed in order to test the control sequences but are not themselves a part of the TP.

The following is a brief description of how the simulator operates: The main program generally begins by calling CLEAR, which sets all the registers to zero. It then calls a control sequence to be tested. The control sequence consists of a number of control points. At each control point there is a list of control signals to be turned on. The procedure CP is called. This turns on the specified signals, calls PRNTSIGS to print them out, and then calls all the logical units of the TP. This is exactly what would happen in the machine: a task signal is initiated, and this signal propagates throughout the machine. After the logical simulation procedures have been called, CP prints out all the signals which were turned on as a result of the initial control signals being on. After this, it may call one of the utility routines (this depends on the individual CP call) and prints out the contents of registers if desired. CP then turns off the original signals and returns to the control point in the sequence. In a given control sequence, there may

also be "calling control points." A calling control point calls CCP (another entry to CP) which performs all the functions of CP, but also initiates and finishes another control sequence before returning to the original one.

When the simulator is completed, an ILLIAC III instruction will be decoded and will initiate the calls to the control sequences. Also, the CLEAR routine will not be needed, since a "bootstrapping" routine will do necessary initializations.

2. Control Point Simulation Routines

The procedures in this section, with the exception of MAIN, are used to simulate the actions of a control point. The MAIN program is used to test the control sequences--initialize register contents and call procedures to be tested.

The procedure, CP, with its entry point CCP, contains calls to all the procedures of the simulator. The total effect of a call to CP is a simulation of a control point, and a print out of diagnostics.

2.1 The Main Program

The Main program of the simulator is used to activate the simulator itself, in order to test one or more control sequences. There are actually many MAIN program possibilities--the instructions in MAIN depend only on which sequences are to be tested.

This is also the place to initialize registers if desired. However, if one wishes to initialize a register, it must first be declared in MAIN in the exact same way it is declared elsewhere.

Declarations which must be included in MAIN are

```
DCL CP ENTRY (CHAR(50), CHAR(180));
```

```
DCL CCP ENTRY (CHAR(50), CHAR(180), CHAR(20), BIT(1));
```

```
DCL ON (1000) BIT(1) EXTERNAL;
```

```
DCL NAME (1000) CHAR(10) EXTERNAL;
```

Before initializing registers, each MAIN program should include a call to CLEAR, which sets all external registers to zero, and then initializes the associative registers, the name registers, and the queue counters. Every MAIN program also includes one other initial step: setting the ON array to zero. A thorough explanation of the use of the ON array is given in Section 2.2, the CP procedure.

The rest of the MAIN procedure is arbitrary. It usually consists of calls to CP and/or CCP. CP is a procedure which simulates the actions of a control point by turning on control signals, letting them propagate throughout the TP and then turning them off. CCP is another entry point to CP. It is used to simulate the action of a "calling control point" by turning on control signals, letting them propagate throughout the TP, calling a control sequence, letting the control sequence complete itself, and then turning off the signals.

A listing of a sample MAIN procedure is given below. Many of the statements will be unclear at this time, but will be explained in the following chapters.

2.1.1 Example MAIN Program Listing

```
// EXEC PL1LKGO,PARM.LKED='LIST,MAP,NCAL'
//PL1.SYSIN DD *
MAIN: PROC OPTIONS(MAIN);
      DCL CLEAR ENTRY EXTERNAL;
      DCL ON(1000) BIT(1) EXTERNAL;
      DCL RRET BIT(1);
      DCL CP ENTRY(CHAR(50),CHAR(180));
      DCL CCP ENTRY(CHAR(50),CHAR(180),CHAR(20),BIT(1));
      DCL (AR,DR,DB,LR,IR) (36) BIT(1) EXTERNAL;
      DCL TGR(1:4) BIT(1) EXTERNAL;
      DCL BR(0:7,36) BIT(1) EXTERNAL;
      DCL MEMSTOR(0:128,0:1,1:36) BIT(1) EXTERNAL;
      DCL PR(0:14,1:36) BIT(1) EXTERNAL;
      DCL OK(0:7) BIT(1) EXTERNAL;
      DCL SCR(1:5) BIT(1) EXTERNAL;
      DCL OSTR(1:2) BIT(1) EXTERNAL;
      ON='0'B;
      CALL CLEAR;
      DO I=1 TO 36;
        MEMSTOR(0,0,I)='0'B;
      END;
      MEMSTOR(0,0,9),MEMSTOR(0,0,18)='1'B;
      MEMSTOR(0,0,17)='1'B;
      OK='1'B;
      IR(2),IR(8)='1'B;
      DO I=0 TO 7;
        BR(I,9),BR(I,18),BR(I,27),BR(I,36)='1'B;
      END;
      AR(36),IR(35),LR(34),DR(33),DR(32)='1'B;
      PR(1,33),PR(2,32),PR(3,32),PR(3,33),PR(4,31),PR(5,31),
      PR(5,33),PR(6,31),PR(6,32),PR(7,31),PR(7,32),PR(7,33),
      PR(8,30),PR(9,33),PR(9,30),PR(10,30),PR(10,32),PR(11,30),
      PR(11,32),PR(11,33),PR(12,30),PR(12,31),PR(13,30),PR(13,31),
      PR(13,33),PR(14,30),PR(14,31),PR(14,32)='1'B;
      CALL CP('MAINSETUP','SCSEF,CSBFE. ');
      CALL CP('MAIN1A','SIMMS,IRTGG(0). ');
      CALL CCP('MAIN1AA','.', 'MEMDIR',RRET);
      SCR='1'B; SCR(1),SCR(2)='0'B;
      OSTR(1)='0'B; OSTR(2)='1'B;
      CALL CCP('MAIN5','RPOPS,CS2C.', 'OSENTRY',RRET);
      CALL BRROUT;
      END MAIN;
/*
```


2.2 CP, CCP Procedures

CP and CCP are separate entry points to the same procedure. They simulate the actions of a control point and calling control point respectively.

2.2.1 CP Description

CP simulates a control point by setting to 1 control signals which are passed to it as parameters. It propagates these signals throughout the TP by calling all the simulating routines, and then prints out all the signals which are on at the end of this control point. Finally, it sets the original signals to 0. CP also prints out information necessary to determine flow of control.

The parameters are passed to CP in the form of two character strings. The first is a character string containing the control point task name and, possibly, sub-task names for the control point. The second string is a list of signal names to be set to 1 by the control point. The sub-task names of a control point indicate extra signals to be set to 1 due to certain conditions. The signals to be turned on are grouped so that they are associated with the control task name or sub-task name. These groups are separated by a vertical bar "|". In addition to signals which come from the control point, extra "utility" signals, which activate one or more of the utility print routines, may be added to this list. They are also separated from the others by a vertical bar and come last in the list. The list is always ended with a period.

The following is an example of a call to CP:

```
CALL CP ('MT4, MT4X', 'ARPG, BDRG|IBC(0)|WRREGS.');
```

CP will call a routine (PRNTSIGS) which will print the following on the output:

```
MT4:      ARPG, BDRG
MT4X:     IBC(0)
UTILITY:  WRREGS
```

PRNTSIGS will then call EXTRACT which sets these signals to one. Then CP calls SIMCALL, an internal procedure, which calls all the simulating routines of the TP, and also the utility routine REGSOUT, which is

triggered by the signal WRREGS. SIMCALL then prints out all the signals which are now equal to one, and returns to CP. CP then calls EXTRACT to set the original signals to zero.

In order to facilitate the task of setting the signals to one and resetting them to zero, the signals were defined as follows: First, an array, $\emptyset N$, was declared. $\emptyset N$ has 1000 1-bit positions. Each signal name is defined as a position of $\emptyset N$, i.e., each 1-bit signal takes the same position in storage as does an element of the $\emptyset N$ array. The DEFINED declaration attribute for PL/1 is similar to the EQUIV attribute in FORTRAN. A corresponding 1000 element array, NAME, is declared so that each of its elements is a character string of length 8. These character strings are initialized to be the signal names themselves, in alphabetical order. As an example, NAME(1) = 'ACTE'. The signal, ACTE, is defined as $\emptyset N$ position (1), that is, $\emptyset N(1)$ and ACTE refer to the same bit in storage. If $\emptyset N(1) = '1'B$ then ACTE = '1'B; if $\emptyset N(1) = '0'B$ then ACTE = '0'B.

This process of doubly defining the signals is not aesthetically pleasing--at least at first. However it is entirely necessary in order to allow the simulator to monitor what is happening. Basically we need to refer to the signals in two ways: 1) as individual logic signals and 2) as a special class of indicators which help to determine the hardware status of the TP. The first use is easily satisfied by making each control signal a PL/1 variable. The second use is more complicated.

One example of a CP monitoring function which must use the array structure is the listing of active control signals. If these signals were only defined as individual PL/1 variables, it would be necessary to have separate IF statements and PUT statements for each control signal in the simulator. By using the $\emptyset N$ array structure however, along with the array of 8-character names in NAME, it is possible to check the control signals with a single DO loop containing a single IF statement and a PUT statement. If $\emptyset N(I) = '1'B$, then the signal whose name is located in NAME(I) is equal to '1'B.

A second monitoring function which uses the array structure occurs when the signals are turned on and off. In this situation the control

logic routines send the names of the signals to be activated to CP in the form of a character string instead of turning them on themselves. CP then uses the EXTRACT and FIND routines to break up the string and set the proper signals to the desired state. This allows the simulator to have a generalized capability to check the validity of the signal names and also to indicate exactly where in the simulation the signal was turned on. This would not be possible if the control logic routines simply used assignment statements to turn on the logic signals.

As might be expected the construction of such a complicated variable structure could be fraught with human error. As a result it was decided to write a separate program, DCLGEN, which given an input of signal names, would generate an output of punched cards in the format of PL/1 DECLARE statements. These statements define every signal name as a particular signal name and at the same time initialize the corresponding entry in NAME to the correct 8 character string.

Another thing to notice about the $\emptyset N$ array, is that the signals are divided into three sections. The first section, from position 1 to $ENDP\emptyset S$, contains the independent control signals in alphabetical order. These signals are set equal to '1'B by a call to CP and are independent of all operations in the simulating routines. These signals are in alphabetical order in order to allow the CP routines to use binary chop searching algorithms. The signals which are defined as $\emptyset N$ positions $ENDP\emptyset S + 1$ to $ENDP\emptyset S2$ are the dependent control signals which are also arranged in alphabetical order. These signals are set according to boolean equations involving other signals in the simulating routines. Hence the name "dependent" - the state of each of these signals is dependent upon the states of other signals.

The third section of signals, from $ENDP\emptyset S2 + 1$ to $ENDP\emptyset S3$ are signals which were added later. By having this third section of signals, it is not necessary to run DCLGEN every time a new signal is to be added. The signal is simply put into this last section temporarily.

The independent and dependent control signals were separated for two reasons: first, since the control sequences will only set the independent signals to '1'B, it is wasteful to search the whole list for the correct signal - when only the independent ones need be checked.

Secondly, if a control sequence sets a dependent control signal to '1'B, then this is a logical error. Since only the independent signals will be searched, the dependent signal will not be found, and a message will be printed out--thus uncovering an error.

2.2.2 CCP-Description

CCP is another entry point to CP. It simulates the action of a calling control point by doing many of the same things that CP does, but in addition, transfers control to a control sequence before returning to set the original signals to zero.

CCP has four parameters. The first two are exactly the same as the two parameters of CP. The third is a character string representing the name of the control sequence to be called, and the fourth is a 1-bit variable which represents the return code of the control sequence. (The sequence returns '1'B if it finished with regular return, '0'B if it finished with interrupt return.)

CCP first acts exactly like CP -- it calls PRNTSIGS to print out the control point names and corresponding signals, calls EXTRACT to turn these signals on, and calls SIMCALL to let these signals propagate through the machine. Then it compares the control sequence name parameter to a table (TABLE) which contains these names. If the control sequence name is the same as TABLE (I), it then transfers control to L(1) which is a label to a call to that control sequence. After this, EXTRACT is again called to set the original signals to zero.

All possible control sequence calls are included in CCP, since they each need certain 1-bit control signal parameters which are declared in CP. These signals cannot be declared as external since they are "defined" as having the same core location as another variable. PL/1 will not allow a variable to have both attributes "EXTERNAL" and "DEFINED".

2.2.3 SIMCALL Procedure

SIMCALL is an internal procedure to CP. SIMCALL has three main functions: it calls all the procedures of the simulator, conditionally calls the utility print routines, and finally, prints out all the signals which are equal to 1 at the end of the control point.

The calls to the simulating routines are very long, each having numerous parameters. It was for this reason that all the calls were grouped in CP, where the parameter definitions occur.

2.2.4 CP Listing

```
// EXEC PL1
//PL1.SYSIN DD *
CP: PROC(CONTPTS,INSTR) RECURSIVE;
    DCL CONTPTS CHAR(50);
    DCL INSTR CHAR(180),
        EXTRACT ENTRY(CHAR(180),BIT(1)) EXTERNAL,
        SIMCALL ENTRY;
    DCL PRNTSIGS ENTRY(CHAR(50),CHAR(180));
    DCL FLAG BIT(1);
    DCL NAME(1000) CHAR(8) EXTERNAL INITIAL(
'ACTE ',      'ADDE ',      'ADDN(0)',    'ADDN(1)',
'ADDN(2)',    'ADDN(3)',    'ADDOUT ',   'ADD5E ',
'ADRG ',      'ALCKE ',     'AMCKE ',    'ALGNER ',
'ANBG ',      'ANDE ',      'AOVSG ',    'ARARFG ',
'ARF33 ',     'ARF34 ',     'ARF35 ',    'ARG(0)',
'ARG(1)',     'ARG(2)',     'ARG(3)',    'ARGG(0)',
'ARGG(1)',    'ARGG(2)',    'ARGG(3)',   'ARL(0)',
'ARL(1)',     'ARL(2)',     'ARL(3)',    'ARPG ',
'ASAO ',      'ASBG ',      'ASE ',      'ASRWE ',
'AST ',       'AUCG ',      'BARG ',     'BDCKE ',
'BDRG ',      'BIRG ',      'BIRN(0)',   'BIRN(1)',
'BIRN(2)',    'BIRN(3)',    'BLRG ',     'BNRG ',
'BOV ',       'BOVI ',     'BRGS ',     'BRS(0)',
'BRS(1)',     'BRS(2)',     'BRS(3)',    'BRS(4)',
'BRS(5)',     'BRS(6)',     'BRS(7)',    'BRSPG ',
'BRSPG ',     'BRSSG ',     'BSR1G ',    'BSR2G ',
'CCSBE ',     'CCSDE ',     'CCSHE ',    'CCSWE ',
'CCTE ',      'CDRE ',      'CINJ ',     'CKOFF ',
'CNBG ',      'CONS ',      'CPAE ',     'CPHE ',
'CPLE ',      'CKRLG ',     'CSBFE ',    'CSDFE ',
'CSFG ',      'CSHFE ',     'CSSE ',     'CSWFE ',
'CS2C ',      'CTD ',       'CTDE ',     'CTMHZ ',
'CTMLZ ',     'CTPG ',      'CTSBG ',    'CTUE ',
'CYCTE ',     'DBBRSG ',    'DBCTG ',    'DBDG(0)',
'DBDG(1)',    'DBDG(2)',    'DBDG(3)',   'DBOSG ',
'DBPRG ',     'DBRJ ',      'DBSTG ',    'DLRG ',
'DLRN(0)',    'DLRN(1)',    'DLRN(2)',   'DLRN(3)',
'DREDY ',     'DRL(0)',     'DRL(1)',    'DRL(2)',
'DRL(3)',     'DRPG ',      'DUP ',      'ENMCBE ',
'EQUE ',      'EQVE ',      'FLOSN ',    'FLPRN ',
'FPT ',       'HOL ',       'IB(0)',     'IB(1)',
'IB(2)',      'IB(3)',      'IBC(0)',    'IBC(1)',
'IBC(2)',     'IBC(3)',     'IBE(0)',    'IBE(1)',
'IBE(2)',     'IBE(3)',     'IBRS ',     'ICTPG ',
'ILAC ',      'INB1G ',     'INB2G ',    'IRG(Q)',
'IRG(1)',     'IRG(2)',     'IRG(3)',    'IRHMG ',
'IRL(0)',     'IRL(1)',     'IRL(2)',    'IRL(3)',
'IRLMG ',     'IRPG ',      'IRTGG(0)',  'IRTGG(1)',
'IRTGG(2)',   'KBG ',       'LDREGE ',   'LDRG ',
'LFX ',       'LJIE ',      'LJOPE ',    'LKCTCE ',
'LNK ',       'LOC ',       'LRDG(0)',   'LRDG(1)',
'LRDG(2)',    'LRDG(3)',    'LRG(0)',    'LRG(1)',
'LRG(2)',     'LRG(3)',     'LRL(0)',    'LRL(1)',
'LRL(2)',     'LRL(3)',     'LRMCTG ',   'LRPG ',
'LRSE1 ',     'LSE ',       'LSPE ',     'MCTE ',
'MZER ',      'M1E ',       'M16E ',     'M2E ',
'M32E ',      'M4E ',       'M8E ',      'NCSE ',
```


'NFCKE '	'NFRG '	'NPR13S '	'NPR14S '
'NOS '	'OPJE '	'ORE '	'OSBER '
'OSCGE '	'OSCG1E '	'OSCG16E '	'OSCG2E '
'OSCG4E '	'OSCG8E '	'OSPGF '	'OSPS '
'OSPTG '	'OSRLG '	'OSRRG '	'OSTPG '
'OTAG '	'OVFL '	'PART '	'PCHKE '
'PERR '	'PL(0) '	'PL(1) '	'PL(2) '
'PL(3) '	'PLCO '	'PLC1 '	'PLC2 '
'PLC3 '	'PMAREP '	'PRPG '	'PSNPRBG '
'PSNRR15 '	'PSNRW15 '	'P1E '	'P16E '
'P2E '	'P32E '	'P4E '	'P8E '
'QCNTS '	'QSETUP '	'RACT '	'RASES '
'RASTRS '	'RBOVIS '	'RCOMFF '	'RCSAS '
'RCT '	'RCYCTS '	'RDBRE '	'RDCPRS '
'REG '	'REQE '	'REQS '	'RGLE '
'RIMMS '	'RJIE '	'RMCT '	'RMSRS '
'RNEGS '	'RNFR '	'RNWTOSS '	'RO '
'ROKS '	'ROPANS '	'ROSBES '	'ROSCS '
'ROSF '	'RPBRE '	'RPERS '	'RPOPS '
'RPROMS '	'RQOVFS '	'RREPS '	'RRSCNS '
'RSDE '	'RSE '	'RSES '	'RSNR '
'RSPE '	'RSUFS '	'RWAITS '	'RWBGE '
'RWRPS '	'SASES '	'SASTRS '	'SBRs '
'SBSRG '	'SCAG '	'SCRPG '	'SCSAS '
'SCSCS '	'SCSFS '	'SCSSS '	'SCT2G '
'SDCPRS '	'SE '	'SFX '	'SGBOV '
'SGBOVs '	'SGDNT '	'SGMNT '	'SGMNTS '
'SGTRP '	'SGTRPS '	'SGTS '	'SILACS '
'SIMMS '	'SLTS '	'SMSRS '	'SN '
'SNAMT '	'SNCSS '	'SNEGS '	'SNNRG '
'SNRDE '	'SNSG '	'SNWTE '	'SNWTOSS '
'SOPANS '	'SOSBES '	'SOSCS '	'SPERS '
'SPOPS '	'SPROMS '	'SREPS '	'SRSCNS '
'SSUFS '	'STBOV '	'STBOVS '	'STONT '
'STDNTS '	'STMNTS '	'STOR '	'STTRPS '
'SUF '	'SWAITS '	'SWRPS '	'TIIE '
'TNBG '	'TSBG '	'TZE '	'T1R2G '
'T1SCG '	'T1STG '	'T2AG '	'T2BG '
'T2SCG '	'T2STG '	'VAL '	'WAIT '
'WRBRE '	'WRBRS '	'WROPSTK '	'WRPRE '
'WRPRLF '	'WRPRVE '	'WRPTRS '	'WRREGS '
'WRSBE '	'WSBLE '	'WSBVE '	'XCHSTRT '
'XNNRP '	'XNREL '	'XNRET '	'XORE '
'X2 '	'ADV '	'A5OV '	'CEQA '
'CSA '	'CSB '	'CSBH '	'CSD '
'CSH '	'CSW '	'CYCOV '	'CYC6 '
'C5INJ '	'DA '	'DECPR '	'EQ '
'GT '	'IBPL13 '	'IBPL21 '	'IMM '
'LRSE '	'LT '	'MSTOR '	'NEG '
'NFOHE '	'NRSETG '	'NT '	'NWTOS '
'NYET '	'N13S '	'OPANDF '	'OSC '
'OSF '	'OSPG '	'OSSS '	'OV1 '
'OV2 '	'PER '	'POP '	'PROMOD '
'RSCN '	'RSCTRS '	'RSYS '	'SQTOV '
'TGIM '	'TRAP '	'UF1 '	'UF2 '
'WRAP '	'XQCNTS '	'XRVE '	'ZSCR '
'ZZZZZZZZ', 'IAC', 'XQCNTS', 'RSCTRS', 'WRMEM',			

```

      'SGPNTS', 'PGMPAC');
DCL ON(1000) BIT(1) EXTERNAL;
DCL
ACTE      BIT(1) DEFINED ON POSITION( 1),
ADDE      BIT(1) DEFINED ON POSITION( 2),
ADDN(0:3) BIT(1) DEFINED ON POSITION( 3),
ADDOUT    BIT(1) DEFINED ON POSITION( 7),
ADD5E     BIT(1) DEFINED ON POSITION( 8),
ADRG      BIT(1) DEFINED ON POSITION( 9),
ALCKE     BIT(1) DEFINED ON POSITION(10),
AMCKE     BIT(1) DEFINED ON POSITION(11),
ALGNER    BIT(1) DEFINED ON POSITION(12),
ANBG      BIT(1) DEFINED ON POSITION(13),
ANDE      BIT(1) DEFINED ON POSITION(14),
AOVSG     BIT(1) DEFINED ON POSITION(15),
ARARFG    BIT(1) DEFINED ON POSITION(16),
ARF33     BIT(1) DEFINED ON POSITION(17),
ARF34     BIT(1) DEFINED ON POSITION(18),
ARF35     BIT(1) DEFINED ON POSITION(19),
ARG(0:3)  BIT(1) DEFINED ON POSITION(20),
ARGG(0:3) BIT(1) DEFINED ON POSITION(24),
ARL(0:3)  BIT(1) DEFINED ON POSITION(28),
ARPG      BIT(1) DEFINED ON POSITION(32),
ASAO      BIT(1) DEFINED ON POSITION(33),
ASBG      BIT(1) DEFINED ON POSITION(34),
ASE       BIT(1) DEFINED ON POSITION(35),
ASRWE     BIT(1) DEFINED ON POSITION(36),
AST       BIT(1) DEFINED ON POSITION(37),
AUCG      BIT(1) DEFINED ON POSITION(38),
BARG      BIT(1) DEFINED ON POSITION(39),
BDCKE     BIT(1) DEFINED ON POSITION(40),
BDRG      BIT(1) DEFINED ON POSITION(41),
BIRG      BIT(1) DEFINED ON POSITION(42),
BIRN(0:3) BIT(1) DEFINED ON POSITION(43),
BLRG      BIT(1) DEFINED ON POSITION(47),
BNRG      BIT(1) DEFINED ON POSITION(48),
BOV       BIT(1) DEFINED ON POSITION(49),
BOVI      BIT(1) DEFINED ON POSITION(50),
BRGS      BIT(1) DEFINED ON POSITION(51),
BR5(0:7)  BIT(1) DEFINED ON POSITION(52),
BR5BPG    BIT(1) DEFINED ON POSITION(60),
BRSPG     BIT(1) DEFINED ON POSITION(61),
BRSSG     BIT(1) DEFINED ON POSITION(62),
BSK1G     BIT(1) DEFINED ON POSITION(63),
BSR2G     BIT(1) DEFINED ON POSITION(64),
CCSBE     BIT(1) DEFINED ON POSITION(65),
CCSDE     BIT(1) DEFINED ON POSITION(66),
CCSHE     BIT(1) DEFINED ON POSITION(67),
CCSWE     BIT(1) DEFINED ON POSITION(68),
CCTE      BIT(1) DEFINED ON POSITION(69),
CDRE      BIT(1) DEFINED ON POSITION(70),
CINJ      BIT(1) DEFINED ON POSITION(71),
CKOFE     BIT(1) DEFINED ON POSITION(72),
CNBG      BIT(1) DEFINED ON POSITION(73),
CONS      BIT(1) DEFINED ON POSITION(74),
CPAE      BIT(1) DEFINED ON POSITION(75),
CPHE      BIT(1) DEFINED ON POSITION(76),

```

CPL	BIT(1)	DEFINED ON POSITION(77),
CRRLG	BIT(1)	DEFINED ON POSITION(78),
CSBFE	BIT(1)	DEFINED ON POSITION(79),
CSDFE	BIT(1)	DEFINED ON POSITION(80),
CSFG	BIT(1)	DEFINED ON POSITION(81),
CSHFE	BIT(1)	DEFINED ON POSITION(82),
CSSE	BIT(1)	DEFINED ON POSITION(83),
CSWFE	BIT(1)	DEFINED ON POSITION(84),
CS2C	BIT(1)	DEFINED ON POSITION(85),
CTD	BIT(1)	DEFINED ON POSITION(86),
CTDE	BIT(1)	DEFINED ON POSITION(87),
CTMHZ	BIT(1)	DEFINED ON POSITION(88),
CTMLZ	BIT(1)	DEFINED ON POSITION(89),
CTPG	BIT(1)	DEFINED ON POSITION(90),
CTSBG	BIT(1)	DEFINED ON POSITION(91),
CTUE	BIT(1)	DEFINED ON POSITION(92),
CYCTE	BIT(1)	DEFINED ON POSITION(93),
DBBRSG	BIT(1)	DEFINED ON POSITION(94),
DBCTG	BIT(1)	DEFINED ON POSITION(95),
DBDG(0:3)	BIT(1)	DEFINED ON POSITION(96),
DBOSG	BIT(1)	DEFINED ON POSITION(100),
DBPRG	BIT(1)	DEFINED ON POSITION(101),
DBRJ	BIT(1)	DEFINED ON POSITION(102),
DBSTG	BIT(1)	DEFINED ON POSITION(103),
DLRG	BIT(1)	DEFINED ON POSITION(104),
DLRN(0:3)	BIT(1)	DEFINED ON POSITION(105),
DREDY	BIT(1)	DEFINED ON POSITION(109),
DRL(0:3)	BIT(1)	DEFINED ON POSITION(110),
DRPG	BIT(1)	DEFINED ON POSITION(114),
DUP	BIT(1)	DEFINED ON POSITION(115),
ENMCBE	BIT(1)	DEFINED ON POSITION(116),
EQUE	BIT(1)	DEFINED ON POSITION(117),
EQVE	BIT(1)	DEFINED ON POSITION(118),
FLOSN	BIT(1)	DEFINED ON POSITION(119),
FLPRN	BIT(1)	DEFINED ON POSITION(120),
FPT	BIT(1)	DEFINED ON POSITION(121),
HOL	BIT(1)	DEFINED ON POSITION(122),
IB(0:3)	BIT(1)	DEFINED ON POSITION(123),
IBC(0:3)	BIT(1)	DEFINED ON POSITION(127),
IBE(0:3)	BIT(1)	DEFINED ON POSITION(131),
IBRS	BIT(1)	DEFINED ON POSITION(135),
ICTPG	BIT(1)	DEFINED ON POSITION(136),
ILAC	BIT(1)	DEFINED ON POSITION(137),
INB1G	BIT(1)	DEFINED ON POSITION(138),
INB2G	BIT(1)	DEFINED ON POSITION(139),
IRG(0:3)	BIT(1)	DEFINED ON POSITION(140),
IRHMG	BIT(1)	DEFINED ON POSITION(144),
IRL(0:3)	BIT(1)	DEFINED ON POSITION(145),
IRLMG	BIT(1)	DEFINED ON POSITION(149),
IRPG	BIT(1)	DEFINED ON POSITION(150),
IRTGG(0:2)	BIT(1)	DEFINED ON POSITION(151),
KBG	BIT(1)	DEFINED ON POSITION(154),
LDREGE	BIT(1)	DEFINED ON POSITION(155),
LDRG	BIT(1)	DEFINED ON POSITION(156),
LFX	BIT(1)	DEFINED ON POSITION(157),
LJIE	BIT(1)	DEFINED ON POSITION(158),
LJOPE	BIT(1)	DEFINED ON POSITION(159),

LKCTCE	BIT(1)	DEFINED ON POSITION(160),
LNK	BIT(1)	DEFINED ON POSITION(161),
LOC	BIT(1)	DEFINED ON POSITION(162),
LRDG(0:3)	BIT(1)	DEFINED ON POSITION(163),
LRG(0:3)	BIT(1)	DEFINED ON POSITION(167),
LRL(0:3)	BIT(1)	DEFINED ON POSITION(171),
LRMCTG	BIT(1)	DEFINED ON POSITION(175),
LRPG	BIT(1)	DEFINED ON POSITION(176),
LRSE1	BIT(1)	DEFINED ON POSITION(177),
LSE	BIT(1)	DEFINED ON POSITION(178),
LSPE	BIT(1)	DEFINED ON POSITION(179),
MCTE	BIT(1)	DEFINED ON POSITION(180),
MZER	BIT(1)	DEFINED ON POSITION(181),
M1E	BIT(1)	DEFINED ON POSITION(182),
M16E	BIT(1)	DEFINED ON POSITION(183),
M2E	BIT(1)	DEFINED ON POSITION(184),
M32E	BIT(1)	DEFINED ON POSITION(185),
M4E	BIT(1)	DEFINED ON POSITION(186),
M8E	BIT(1)	DEFINED ON POSITION(187),
NCSE	BIT(1)	DEFINED ON POSITION(188),
NFCKE	BIT(1)	DEFINED ON POSITION(189),
NFRG	BIT(1)	DEFINED ON POSITION(190),
NPR13S	BIT(1)	DEFINED ON POSITION(191),
NPR14S	BIT(1)	DEFINED ON POSITION(192),
NOS	BIT(1)	DEFINED ON POSITION(193),
OPJE	BIT(1)	DEFINED ON POSITION(194),
ORE	BIT(1)	DEFINED ON POSITION(195),
OSBER	BIT(1)	DEFINED ON POSITION(196),
OSCGE	BIT(1)	DEFINED ON POSITION(197),
OSCG1E	BIT(1)	DEFINED ON POSITION(198),
OSCG16E	BIT(1)	DEFINED ON POSITION(199),
OSCG2E	BIT(1)	DEFINED ON POSITION(200),
OSCG4E	BIT(1)	DEFINED ON POSITION(201),
OSCG8E	BIT(1)	DEFINED ON POSITION(202),
OSPFG	BIT(1)	DEFINED ON POSITION(203),
OSPS	BIT(1)	DEFINED ON POSITION(204),
OSPTG	BIT(1)	DEFINED ON POSITION(205),
OSRLG	BIT(1)	DEFINED ON POSITION(206),
OSRRG	BIT(1)	DEFINED ON POSITION(207),
OSTPG	BIT(1)	DEFINED ON POSITION(208),
OTAG	BIT(1)	DEFINED ON POSITION(209),
OVFL	BIT(1)	DEFINED ON POSITION(210),
PART	BIT(1)	DEFINED ON POSITION(211),
PCHKE	BIT(1)	DEFINED ON POSITION(212),
PERR	BIT(1)	DEFINED ON POSITION(213),
PL(0:3)	BIT(1)	DEFINED ON POSITION(214),
PLC0	BIT(1)	DEFINED ON POSITION(218),
PLC1	BIT(1)	DEFINED ON POSITION(219),
PLC2	BIT(1)	DEFINED ON POSITION(220),
PLC3	BIT(1)	DEFINED ON POSITION(221),
PMAREP	BIT(1)	DEFINED ON POSITION(222),
PRPG	BIT(1)	DEFINED ON POSITION(223),
PSNPRBG	BIT(1)	DEFINED ON POSITION(224),
PSNRR15	BIT(1)	DEFINED ON POSITION(225),
PSNRW15	BIT(1)	DEFINED ON POSITION(226),
P1E	BIT(1)	DEFINED ON POSITION(227),
P16E	BIT(1)	DEFINED ON POSITION(228),

P2E	BIT(1)	DEFINED	ON	POSITION(229),
P32E	BIT(1)	DEFINED	ON	POSITION(230),
P4E	BIT(1)	DEFINED	ON	POSITION(231),
P8E	BIT(1)	DEFINED	ON	POSITION(232),
QCNT	BIT(1)	DEFINED	ON	POSITION(233),
QSETUP	BIT(1)	DEFINED	ON	POSITION(234),
RACT	BIT(1)	DEFINED	ON	POSITION(235),
RASES	BIT(1)	DEFINED	ON	POSITION(236),
RASTRS	BIT(1)	DEFINED	ON	POSITION(237),
RBOVIS	BIT(1)	DEFINED	ON	POSITION(238),
RCOMFF	BIT(1)	DEFINED	ON	POSITION(239),
RCSAS	BIT(1)	DEFINED	ON	POSITION(240),
RCT	BIT(1)	DEFINED	ON	POSITION(241),
RCYCTS	BIT(1)	DEFINED	ON	POSITION(242),
RDBRE	BIT(1)	DEFINED	ON	POSITION(243),
RDCPRS	BIT(1)	DEFINED	ON	POSITION(244),
REG	BIT(1)	DEFINED	ON	POSITION(245),
REQE	BIT(1)	DEFINED	ON	POSITION(246),
REQS	BIT(1)	DEFINED	ON	POSITION(247),
RGLE	BIT(1)	DEFINED	ON	POSITION(248),
RIMMS	BIT(1)	DEFINED	ON	POSITION(249),
RJIE	BIT(1)	DEFINED	ON	POSITION(250),
RMCT	BIT(1)	DEFINED	ON	POSITION(251),
RMSRS	BIT(1)	DEFINED	ON	POSITION(252),
RNEGS	BIT(1)	DEFINED	ON	POSITION(253),
RNFR	BIT(1)	DEFINED	ON	POSITION(254),
RNWTSS	BIT(1)	DEFINED	ON	POSITION(255),
RO	BIT(1)	DEFINED	ON	POSITION(256),
ROKS	BIT(1)	DEFINED	ON	POSITION(257),
ROPANS	BIT(1)	DEFINED	ON	POSITION(258),
ROSBES	BIT(1)	DEFINED	ON	POSITION(259),
ROSCS	BIT(1)	DEFINED	ON	POSITION(260),
ROSF	BIT(1)	DEFINED	ON	POSITION(261),
RPBRE	BIT(1)	DEFINED	ON	POSITION(262),
RPERS	BIT(1)	DEFINED	ON	POSITION(263),
RPOPS	BIT(1)	DEFINED	ON	POSITION(264),
RPRMS	BIT(1)	DEFINED	ON	POSITION(265),
RQOVFS	BIT(1)	DEFINED	ON	POSITION(266),
RREPS	BIT(1)	DEFINED	ON	POSITION(267),
RRSCNS	BIT(1)	DEFINED	ON	POSITION(268),
RSDE	BIT(1)	DEFINED	ON	POSITION(269),
RSE	BIT(1)	DEFINED	ON	POSITION(270),
RSES	BIT(1)	DEFINED	ON	POSITION(271),
RSNR	BIT(1)	DEFINED	ON	POSITION(272),
RSPE	BIT(1)	DEFINED	ON	POSITION(273),
RSUFS	BIT(1)	DEFINED	ON	POSITION(274),
RWAITS	BIT(1)	DEFINED	ON	POSITION(275),
RWBGE	BIT(1)	DEFINED	ON	POSITION(276),
RWRPS	BIT(1)	DEFINED	ON	POSITION(277),
SASES	BIT(1)	DEFINED	ON	POSITION(278),
SASTRS	BIT(1)	DEFINED	ON	POSITION(279),
SBR	BIT(1)	DEFINED	ON	POSITION(280),
SBSRG	BIT(1)	DEFINED	ON	POSITION(281),
SCAG	BIT(1)	DEFINED	ON	POSITION(282),
SCRPG	BIT(1)	DEFINED	ON	POSITION(283),
SCSAS	BIT(1)	DEFINED	ON	POSITION(284),
SCSCS	BIT(1)	DEFINED	ON	POSITION(285),

SCSFS	BIT(1)	DEFINED	ON	POSITION(286),
SCSSS	BIT(1)	DEFINED	ON	POSITION(287),
SCT2G	BIT(1)	DEFINED	ON	POSITION(288),
SDCPRS	BIT(1)	DEFINED	ON	POSITION(289),
SE	BIT(1)	DEFINED	ON	POSITION(290),
SFX	BIT(1)	DEFINED	ON	POSITION(291),
SGBOV	BIT(1)	DEFINED	ON	POSITION(292),
SGBOVS	BIT(1)	DEFINED	ON	POSITION(293),
SGDNT	BIT(1)	DEFINED	ON	POSITION(294),
SGMNT	BIT(1)	DEFINED	ON	POSITION(295),
SGMNTS	BIT(1)	DEFINED	ON	POSITION(296),
SGTRP	BIT(1)	DEFINED	ON	POSITION(297),
SGTRPS	BIT(1)	DEFINED	ON	POSITION(298),
SGTS	BIT(1)	DEFINED	ON	POSITION(299),
SILACS	BIT(1)	DEFINED	ON	POSITION(300),
SIMMS	BIT(1)	DEFINED	ON	POSITION(301),
SLTS	BIT(1)	DEFINED	ON	POSITION(302),
SMSRS	BIT(1)	DEFINED	ON	POSITION(303),
SN	BIT(1)	DEFINED	ON	POSITION(304),
SNAMT	BIT(1)	DEFINED	ON	POSITION(305),
SNCSS	BIT(1)	DEFINED	ON	POSITION(306),
SNEGS	BIT(1)	DEFINED	ON	POSITION(307),
SNNRG	BIT(1)	DEFINED	ON	POSITION(308),
SNRDE	BIT(1)	DEFINED	ON	POSITION(309),
SNSG	BIT(1)	DEFINED	ON	POSITION(310),
SNWTE	BIT(1)	DEFINED	ON	POSITION(311),
SNWTOSS	BIT(1)	DEFINED	ON	POSITION(312),
SOPANS	BIT(1)	DEFINED	ON	POSITION(313),
SOSBES	BIT(1)	DEFINED	ON	POSITION(314),
SOSCS	BIT(1)	DEFINED	ON	POSITION(315),
SPERS	BIT(1)	DEFINED	ON	POSITION(316),
SPOPS	BIT(1)	DEFINED	ON	POSITION(317),
SPROMS	BIT(1)	DEFINED	ON	POSITION(318),
SREPS	BIT(1)	DEFINED	ON	POSITION(319),
SRSCNS	BIT(1)	DEFINED	ON	POSITION(320),
SSUFS	BIT(1)	DEFINED	ON	POSITION(321),
STBOV	BIT(1)	DEFINED	ON	POSITION(322),
STBOVS	BIT(1)	DEFINED	ON	POSITION(323),
STDNT	BIT(1)	DEFINED	ON	POSITION(324),
STDNTS	BIT(1)	DEFINED	ON	POSITION(325),
STMNTS	BIT(1)	DEFINED	ON	POSITION(326),
STOR	BIT(1)	DEFINED	ON	POSITION(327),
STTRPS	BIT(1)	DEFINED	ON	POSITION(328),
SUF	BIT(1)	DEFINED	ON	POSITION(329),
SWAITS	BIT(1)	DEFINED	ON	POSITION(330),
SWRPS	BIT(1)	DEFINED	ON	POSITION(331),
TIE	BIT(1)	DEFINED	ON	POSITION(332),
TNBG	BIT(1)	DEFINED	ON	POSITION(333),
TSBG	BIT(1)	DEFINED	ON	POSITION(334),
TZE	BIT(1)	DEFINED	ON	POSITION(335),
T1R2G	BIT(1)	DEFINED	ON	POSITION(336),
T1SCG	BIT(1)	DEFINED	ON	POSITION(337),
T1STG	BIT(1)	DEFINED	ON	POSITION(338),
T2AG	BIT(1)	DEFINED	ON	POSITION(339),
T2BG	BIT(1)	DEFINED	ON	POSITION(340),
T2SCG	BIT(1)	DEFINED	ON	POSITION(341),
T2STG	BIT(1)	DEFINED	ON	POSITION(342),

VAL	.BIT(1) DEFINED ON POSITION(343),
WAIT	BIT(1) DEFINED ON POSITION(344),
WRBRE	BIT(1) DEFINED ON POSITION(345),
WRBRS	BIT(1) DEFINED ON POSITION(346),
WROPSTK	BIT(1) DEFINED ON POSITION(347),
WRPRE	BIT(1) DEFINED ON POSITION(348),
WRPRLE	BIT(1) DEFINED ON POSITION(349),
WRPRVE	BIT(1) DEFINED ON POSITION(350),
WRPTRS	BIT(1) DEFINED ON POSITION(351),
WRREGS	BIT(1) DEFINED ON POSITION(352),
WRSBE	BIT(1) DEFINED ON POSITION(353),
WSBLE	BIT(1) DEFINED ON POSITION(354),
WSBVE	BIT(1) DEFINED ON POSITION(355),
XCHSTRT	BIT(1) DEFINED ON POSITION(356),
XNNRP	BIT(1) DEFINED ON POSITION(357),
XNREL	BIT(1) DEFINED ON POSITION(358),
XNRET	BIT(1) DEFINED ON POSITION(359),
XORE	BIT(1) DEFINED ON POSITION(360),
X2	BIT(1) DEFINED ON POSITION(361),
ADV	BIT(1) DEFINED ON POSITION(362),
A5OV	BIT(1) DEFINED ON POSITION(363),
CEQA	BIT(1) DEFINED ON POSITION(364),
CSA	BIT(1) DEFINED ON POSITION(365),
CSB	BIT(1) DEFINED ON POSITION(366),
CSBH	BIT(1) DEFINED ON POSITION(367),
CSD	BIT(1) DEFINED ON POSITION(368),
CSH	BIT(1) DEFINED ON POSITION(369),
CSW	BIT(1) DEFINED ON POSITION(370),
CYCOV	BIT(1) DEFINED ON POSITION(371),
CYC6	BIT(1) DEFINED ON POSITION(372),
C5INJ	BIT(1) DEFINED ON POSITION(373),
DA	BIT(1) DEFINED ON POSITION(374),
DECPR	BIT(1) DEFINED ON POSITION(375),
EQ	BIT(1) DEFINED ON POSITION(376),
GT	BIT(1) DEFINED ON POSITION(377),
IBPL13	BIT(1) DEFINED ON POSITION(378),
IBPL21	BIT(1) DEFINED ON POSITION(379),
IMM	BIT(1) DEFINED ON POSITION(380),
LRSE	BIT(1) DEFINED ON POSITION(381),
LT	BIT(1) DEFINED ON POSITION(382),
MSTOR	BIT(1) DEFINED ON POSITION(383),
NEG	BIT(1) DEFINED ON POSITION(384),
NFONE	BIT(1) DEFINED ON POSITION(385),
NRSETG	BIT(1) DEFINED ON POSITION(386),
NT	BIT(1) DEFINED ON POSITION(387),
NWTQS	BIT(1) DEFINED ON POSITION(388),
NYET	BIT(1) DEFINED ON POSITION(389),
N13S	BIT(1) DEFINED ON POSITION(390),
OPANDF	BIT(1) DEFINED ON POSITION(391),
OSC	BIT(1) DEFINED ON POSITION(392),
OSF	BIT(1) DEFINED ON POSITION(393),
OSPG	BIT(1) DEFINED ON POSITION(394),
OSSS	BIT(1) DEFINED ON POSITION(395),
OV1	BIT(1) DEFINED ON POSITION(396),
OV2	BIT(1) DEFINED ON POSITION(397),
PER	BIT(1) DEFINED ON POSITION(398),
POP	BIT(1) DEFINED ON POSITION(399),

```

PROMOD      BIT(1) DEFINED ON POSITION(400),
RSCN        BIT(1) DEFINED ON POSITION(401),
RSYS        BIT(1) DEFINED ON POSITION(403),
SQTOV       BIT(1) DEFINED ON POSITION(404),
TGIM        BIT(1) DEFINED ON POSITION(405),
TRAP        BIT(1) DEFINED ON POSITION(406),
UF1         BIT(1) DEFINED ON POSITION(407),
UF2         BIT(1) DEFINED ON POSITION(408),
WRAP        BIT(1) DEFINED ON POSITION(409),
XRVE        BIT(1) DEFINED ON POSITION(411),
ZSCR        BIT(1) DEFINED ON POSITION(412),
ZZZZZZZZ   BIT(1) DEFINED ON POSITION(413);
DCL IAC BIT(1) DEFINED ON POSITION(414);
DCL XQCNT BIT(1) DEFINED ON POSITION(415);
DCL RSCTRS BIT(1) DEFINED ON POSITION(416);
DCL WRMEM BIT(1) DEFINED ON POSITION(417);
DCL PGMPAC BIT(1) DEFINED ON POSITION(419);
DCL ENDPOS FIXED BINARY EXTERNAL INITIAL(361);
DCL ENDPOS2 FIXED BIN EXTERNAL INITIAL(414);
DCL ENDPOS3 FIXED BIN EXTERNAL INITIAL(419);
DCL PRS(0:14) BIT(1) EXTERNAL;
DCL RRET BIT(1);
CALL PRNTSIGS(CONTPTS,INSTR);
CALL SIMCALL;
GO TO FINISH;
CCP: ENTRY(CONTPTS,INSTR,SEQNAME,RRET) RECURSIVE;
DCL L(20) LABEL;
DCL SEQNAME CHAR(20);
DCL TABLE(20) CHAR(20) INITIAL
('SCRMOD','OSWRITE','OSREAD','OSENTRY','READAC','QUPDATE',
'WRAC','PMAC','IMMACC','.', 'MEMDIR','MEMMOD',
'OVENTRY');
CALL PRNTSIGS(CONTPTS,INSTR);
PUT EDIT('CALL:',SEQNAME) (SKIP,A,X(15),A);
CALL SIMCALL;
DO I=1 TO 20;
    IF SEQNAME=TABLE(I) THEN GO TO L(I);
END;
PUT SKIP LIST(SEQNAME,'IS AN ILLEGAL NAME');
GO TO FINISH;
/*SCR-MODIFY SEQUENCE */
L(1): CALL SCRMOD(SE,A5OV,WAIT,AOV,RRET);
    PUT SKIP LIST('SCRMOD FINISHED WITH REGULAR RETURN');
GO TO FINISH;
/*OSWRITE SEQUENCE */
L(2): CALL OSWRITE(CSD,RRET);
GO TO FINISH;
/* OSREAD SEQUENCE */
L(3): CALL OSREAD(STOR,NEG,NWTOS,CSD,RRET);
GO TO FINISH;
/* OPERAND STACK SEQUENCE ENTRY */
L(4): CALL OSENTRY(WRAP,AOV,A5OV,CS2C,OV1,OV2,DUP,POP,UF1,
    UF2,RRET);
GO TO FINISH;
/* READ ACCESS TO MEMORY SEQUENCE */
L(5): CALL READAC(CSD,ARF33,RRET);
GO TO FINISH;

```



```

/*QUEUE COUNTER UPDATE SEQUENCE */
L(6): CALL QUPDATE(SQTOV,CYCOV,RRET);
      GO TO FINISH;
/*WRITE ACCESS TO MEMORY SEQUENCE */
L(7): CALL WRAC(DREDY,CSD,ARF33,RRET);
      GO TO FINISH;
/*PARTITIONED MODE ACCESS TO MEMORY SEQUENCE */
L(8): CALL PMAC(MSTOR,SNAMT,ARF33,RRET);
      GO TO FINISH;
/* IMMEDIATE ACCESS TO MEMORY SEQUENCE */
L(9): GO TO FINISH;
/* MEMORY */
L(10): GO TO FINISH; /* NULL CALLING CONTROL POINT */
L(11): CALL MEMDIR(TGIM,ALGNER,DA,NYET,NT,PART,ILAC,BOVI,LOC,MSTOR,
      ARF33,TRAP,SQTOV,CYCOV,ARARFG,LNK,VAL,REG,RRET);
      GO TO FINISH;
L(12): CALL MEMMOD(TGIM,ALGNER,DA,NYET,NT,PART,ILAC,BOVI,
      LOC,MSTOR,ARF33,TRAP,SQTOV,CYCOV,ARARFG,
      LNK,VAL,REG,RRET);
      GO TO FINISH;
L(13): CALL OVENTRY(WRAP,AOV,A5OV,CS2C,OV1,OV2,DUP,POP,UF1,
      UF2,RRET);
FINISH: FLAG='O'B;
        CALL EXTRACT(INSTR,FLAG);

```

```

SIMCALL: PROC;
DCL PTRSOUT ENTRY(FIXED BIN, FIXED BIN);
PUT SKIP(3) LIST('PROCEDURES CALLED:');
CALL FFSET
(CSA,      DECPR,      IMM,      MSTOR,      NEG,      NWTOS,
OPANDF,    OSC,        PER,      POP,        PROMOD,   RCSAS,
RDCPRS,    RIMMS,      RMSRS,    RNEGS,      RNWTOSS,   ROPANS,
RQSCS,     RPPERS,     RPOPS,    RPPROMS,   RRSCNS,    RSCN,
RWRPS,     SCSAS,      SDCPRS,   SIMMS,     SMSRS,     SNEGS,
SNWTOSS,   SOPANS,     SOSCS,    SPERS,     SPOPS,     SPROMS,
SRSCNS,    SWRPS,      WRAP);
CALL CSGEN(CCSBE,      CCSDE,      CCSHE,      CCSWE,      CSB,
           CSBFE,      CSBH,       CSD,        CSDFE,      CSFG,
           CSH,        CSHFE,      CSSE,        CSW,        CSWFE,
           FPT,        HOL,        IMM,        LFX,        NCSE,
           SCSCS,      SCDFS,      SCSSS,      SFX,        SNCSS);
CALL RWBGEN(CSB,CSH,CSW,CSD,RWBGE,ARF33,ARF34,ARF35,
           MSTOR,PGMPAC,ARPG);
CALL CONSGEN(C5INJ, CSB,      CSD,      CSH,      CSW,
           OSCG16E, OSCG1E,  OSCG2E,  OSCG4E,  OSCG8E,
           OSCGE,   SE,      X2);
CALL ADDER5(A5OV, ADD5E, C5INJ, KBG,      OTAG,
           RSYS,   SCAG,   SCT2G,  T1R2G,  T1SCG,
           T1STG,  T2AG,   T2BG,   T2SCG);
CALL OPSTKIN(CKOFE, CSB,      CSD,      CSH,      CSW,
           DBOSG,   OSF,     OSPFG,   OSPTG,   OSSS,   OV1,
           OV2,     ROSF,    UF1,      UF2,     ZSCR);
CALL BREGIN(BRGS, BRSSG, CYC6, CYCOV, CYCTE,
           DA,     NYET,   PSNRR15, PSNRW15, QCNTE,
           QSETUP, RCYCTS, RDBRE, RPBRE, ROOVFS,

```

```

        SNRDE,      SNSG,      SNWTE,      SOTOV,      WRBRE,
        XQCNT, BRS, PSNPRBG);
CALL PREGIN(ACTE, ANBG, ASBG, CCTE, CEQA,
CNBG, CTSBG, IMM, INB1G, INB2G,
N13S, NFCKE, Nfone, NFRG, NPR13S,
NPR14S, RACT, RCT, RNFR, RSNR,
SBRG, SBSRG, SNNRG, TGIM, TNBG,
TSBG, IRTGG, PSNPRBG);
/* CALL IBREGIN; */
CALL BOOLSHFT(ANDE, CTDE, CTMHZ, CTMLZ, CTUE,
DLRG, IBPL13, IBPL21, LRMCTG, LSE,
LSPE, MCTE, ORE, RMCT, RSE,
RSPE, IB, PL);
CALL PER_IN (ARPG, BNRG, BRGS, BRSBPG, CRRLG,
CSB, CSBH, CSH, CTPG, DBRJ,
DRPG, IBPL13, IBPL21, IBRS, ICTPG,
IRPG, LJIE, LJOPE, LRP, LRSE,
M16E, M1E, M2E, M32E, M4E,
M8E, NRSETG, OPJE, OSPG, OSTPG,
P16E, P1E, P2E, P32E, P4E,
P8E, PLC1, PLC2, PLC3, PRPG,
RJIE, RSDE, SCRPG, LRSE1, IB,
IBC, PL, USSS, OSPTG, OSPFG);
IF EQVE|ADDE|XORE THEN
    CALL ADD32(ADDE, CINJ, EQVE, XORE, ADDOUT);
CALL MEMLOG(AMCKE, PART, TRAP, NT, IAC, MSTORE);
CALL PER_OUT(ADDE, ADRG, ASRWE, BARG, BDRG,
BIRG, BLRG, BRGS, BSR1G, BSR2G,
DBPRG, DBSTG, DLRG, EQVE, FLPRN,
LDRG, LSE, RSE, RWBGE, SBKS,
WRBRE, WRPRLE, WRPRVE, WSBLE, WSBVE,
XORE, XRV, ANDE,
ADDN, ARG, ARGG, ARL, RIRN,
DBG, DLRN, DRL, IRG, IRL,
LRDG, LRG, LRL, DBOSG, CORE);
IF ADDE THEN
    CALL ADDOV(ADDE, ADV, AUCG, CSB, CSH,
CSW, RSY);
CALL COMPARE(AUCG, CPAE, CPHE, CPLE, CSB,
CSH, CSW, EQ, GT, LT,
RCOMFF, REQS, RGLE, SGTS, SLTS,
TZE);
IF ENMCBE THEN CALL XNACC;
IF WROPSTK THEN CALL OSOUT( OV1, OV2,
UF1, UF2);
IF WRBRS THEN CALL BRSOUT;
IF WRPTRS THEN CALL PTRSOUT(0,14);
IF WRREGS THEN CALL REGSOUT;
IF WRMEM THEN CALL MEMOUT;
PUT SKIP (2);
PUT EDIT('SIGNALS ON AT COMPLETION OF CONTROL POINT:')(SKIP,A);
PUT SKIP;
DO I=1 TO ENDPOS3;
    IF ON(I) THEN PUT EDIT(NAME(I))(X(5),A);
END;
PUT EDIT('PRS=',PRS) (SKIP,A,15 B(1));
END SIMCALL;

```

CPDONE: END CP;

/*

2.2.5 CP, CCP Variable List

ENDPØS	- Fixed binary external variable-- holds position of end of independent signals
ENDPØS2	- Fixed binary external variable--holds position of end of dependent signals
ENDPØS3	- Fixed binary external variable--used to hold position of last signal in ØN array
INSTR	- Parameter passed to CP and CCP. A character string holding signals to be acti- vated.
NAME	- 1000 position array of character strings initialized to signal names.
ØN	- 1000 position array of bits--the signals
RRET	- 1 bit return signal for control sequences
SEQNAME	- Formal parameter to CCP. A character string holding name of control sequence to be called.
TABLE	- 20 position array of character strings, initialized to the control sequence names.

NOTE: The control signals are defined in the DCLGEN listing.
Other variables involved are procedure names.

2.3 PRNTSIGS Procedure

2.3.1 PRNTSIGS Description

PRNTSIGS is a very complicated looking procedure which does a very simple thing. It prints out the control task name and sub-task names for the control point being activated and the signals which are to be activated in those tasks.

PRNTSIGS, which was originally a part of CP, has the same parameters as does CP. PRNTSIGS takes the first parameter, A, which is a character string of the task names separated by commas, and breaks these names apart, storing them as elements of an array. It then takes the second parameter, B, the character string containing the signals to be activated, and separates the groups. It prints out the task name, and its group of signals, the sub-task names with their signals, if any, followed by

"UTILITY": together with the utility signals if any.

Finally, PRNTSIGS calls EXTRACT to activate the signals, passing it the original string with the bars which had separated the groups now replaced with commas.

2.3.2 PRNTSIGS Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
PRNTSIGS: PROC(A,B);
    DCL B1 CHAR(180),BAR FIXED BIN, C CHAR (180) VARYING;
    DCL A CHAR(50),B CHAR(180),LABEL(10) CHAR(10),
        I FIXED BIN,A1 CHAR(50),START FIXED BIN,STOP FIXED BIN,
    END FIXED BIN,FLAG BIT(1);
    DCL EXTRACT ENTRY(CHAR(180),BIT(1));
    A1=A;
    I=0;
    START=1;
ASAVE: I=I+1;
    END=INDEX(A1,',');
    IF END=0 THEN DO;
        END=INDEX(A1,' ');
        LABEL(I)=SUBSTR(A1,START,END-1)||':';
        GO TO BPRINT;
    END;
    ELSE LABEL(I)=SUBSTR(A1,START,END-1)||':';
    A1=SUBSTR(A1,END+1,50);
    GO TO ASAVE;
BPRINT: B1=B;
    PUT SKIP(5);
    LABEL(I+1)='UTILITY: ';
    START=1;
    STOP=INDEX(B1,',');
    DO I=1 TO 10;
        BAR=INDEX(B1,'|');
        IF BAR=0 THEN DO;
            C=SUBSTR(B1,START,STOP-START);
            PUT EDIT(LABEL(I),C)(SKIP,A(20),A);
            GO TO FINISH;
        END;
        ELSE C=SUBSTR(B1,START,BAR-START);
        PUT EDIT(LABEL(I),C)(SKIP,A(20),A);
        SUBSTR(B1,BAR,1)='|';
        START=BAR+1;
    END;
FINISH: B=B1;
    FLAG='1'B;
    CALL EXTRACT(B,FLAG);
    END PRNTSIGS;
/*
```

2.3.3 PRNTSIGS Variable List

Local Variables:

<u>Simulation Name</u>	<u>Description</u>
A	Parameter passed to PRNTSIGS. Same as CØNTPTS in CP - holds character string of task names.
A1	Used to hold remainder of string A when each name is separated.
B	Parameter passed to PRNTSIGS - same as INSTR in CP - holds string of signals.
B1	Used to hold contents of B with bars changed to commas.
BAR	Used to determine positions of bars in B - where groups are separated.
C	Used to hold a single group of signals.
END	Used to determine separations between labels (task names).
FLAG	The parameter passed to EXTRACT
LABEL	Used to hold individual task names (an array).

2.4 EXTRACT Procedure

2.4.1 EXTRACT Description

The EXTRACT procedure, originally a part of CP, is used to activate or deactivate signals by setting them equal to one or zero respectively.

EXTRACT has two parameters. The first, INSTR, is a character string containing the signals, separated by commas, and ending with a period. The second, FLAG, is a bit which is 1 if the signals are to be set to 1, or 0 if they are to be set to zero.

EXTRACT is so named, since it must "extract" the individual signals from the list. Once it has extracted a signal, it passes it to the internal procedure FIND.

FIND does a binary chop on the elements of the first section of the NAME array, trying to match the given signal with one of them. The external variable ENDPØS gives the position of the last independent signal in NAME, that is, where the first section of the list ends. Since this first section is in alphanumeric collating sequence, these character strings are in increasing numerical value when converted to numbers. Therefore, it is possible to do greater than and less than comparisons between the elements of NAME and the signal being searched. For this reason, the binary chop method of searching could be utilized-- which is desirable since it is faster than a sequential search.

Note that FIND does not search for the signal SIG in the dependent signals section of FIND, since these should not ever be activated via a control point.

If FIND does not succeed in matching the signal to one in the independent list, it searches the third section (sequentially) since these are extra signals.

When a match is found, the corresponding ØN position is set equal to the value of FLAG. If a match is not found, an appropriate message is printed out declaring the signal to be invalid.

2.4.2 EXTRACT Listing

```

// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
  EXTRACT:  PROC(INSTR,FLAG);

    DCL INSTR CHAR(180), FLAG BIT(1);
    DCL (END,IND) FIXED BIN,
    SIG CHAR(8),
    TSTR CHAR(181),
    FIND ENTRY(CHAR(8));
    IF INSTR='.' THEN GO TO EXTDONE;
    END=INDEX(INSTR,'. ');
    IF END=0 THEN DO;
      PUT EDIT('NO PERIOD AT END OF GIVEN STRING(',INSTR,')THIS
SET HAS BEEN SKIPPED.')(SKIP,A,A,A);
      GO TO EXTDONE;
    END;
    TSTR=SUBSTR(INSTR,1,END-1) || ' ';
    DO WHILE (END>0);
      IND=INDEX(TSTR,' ');
      END=END-IND;
      SIG=SUBSTR(TSTR,1,IND-1);
      TSTR=SUBSTR(TSTR,IND+1,END);
      CALL FIND(SIG);
    END;

  FIND:  PROC(SIG);
    DCL SIG CHAR(8),I FIXED BIN;
    DCL NAME(1000) CHAR(8) EXTERNAL, ON(1000) BIT(1) EXTERNAL;
    DCL TWO FIXED BIN;
    DCL (UB,LB) FIXED BIN;
    DCL (ENDPOS,ENDPOS2,ENDPOS3) FIXED BIN EXTERNAL;
    TWO=2;
    UB=ENDPOS+1;
    LB=0;
    START:  IF (UB=LB+1) THEN GO TO ERROR;
    I=(LB+UB)/TWO;
    IF SIG<NAME(I) THEN UB=I;
    ELSE IF SIG>NAME(I) THEN LB=I;
    ELSE GO TO SET;
    GO TO START;
  SET:  ON(I)=FLAG;
    GO TO DONE;
  ERROR:  DO I= ENDPOS2 TO ENDPOS3;
    IF SIG=NAME(I) THEN GO TO SET;
  END;
    PUT EDIT('SIGNAL GIVEN,',SIG,', IS NOT VALID.')(
      (SKIP,A,A,A));
  DONE:  END FIND;
  EXTDONE:  END EXTRACT;

```

/*

2.4.3. EXTRACT Variable List

External variables:

<u>Simulation Name</u>	<u>Description</u>
ENDPØS	Marks end of independent signal in NAME array
ENDPØS2	Marks end of dependent signals in NAME array
ENDPØS3	Marks end of all signals in NAME array.
NAME	1000 element array of signal names
ØN	1000 element bit array of signals

Local Variables:

<u>Simulation Name</u>	<u>Description</u>
END	Set equal to the position of the period at the end of INSTR - used to determine end of the string.
FLAG	Passed as a parameter to EXTRACT, 1 bit - holds value to which signals are to be set
INSTR	Parameter to EXTRACT - list of signals to be activated or deactivated
SIG	Signal extracted from INSTR and then passed as parameter to FIND
TSTR	Remainder of string as each signal is extracted

2.5 DCLGEN Procedure

2.5.1 DCLGEN Description

The DCLGEN procedure is run whenever an up-to-date list of the signals used in the simulator is desired. The procedure generates the declarations of the signals as positions of the $\emptyset N$ array and the corresponding initializations of the NAME array.

The signal names are simply data cards for the procedure. On each data card is a signal name and its description.

The procedure gets a data card (a signal name) and punches a card which is the declaration of the signal defining it to be position (i) of the $\emptyset N$ array. Then it sets NAME(i) equal to the signal name (a character string). Each time the program reads a card, it increments the value of i.

When the last card has been read, the contents of the NAME array are punched out in a form useful for initialization of the array.

The data cards are in alphabetical order in two sections--the independent and dependent signals. The program does not check for any mistakes in the order of the cards, since only a few are added at one time.

2.5.2 DCLGEN Listing

```

/*FORMAT PR,DDNAME=SYMSG,TRAIN=PL1
// EXEC PL1LKGO
//PL1.SYSIN DD *
  MAIN:  PROC OPTIONS(MAIN);

    DCL SIG CHAR(10) VARYING;
    DCL M FIXED BIN, (J,LB,UB) FIXED DEC(1,0);
    DCL JCHAR CHAR(4),I FIXED DEC(3,0), NAME(1000) CHAR(11) VARYING;
    I=0;
    ON ENDFILE(SYSIN) GO TO FINAL;
  BEGIN:  GET EDIT (SIG) (SKIP,COLUMN(10),A(10));
    PUT FILE(SYSPNCH) EDIT (SIG, 'BIT(1) DEFINED ON POSITION(',
      I+1,')',') (SKIP,X(5),A,A,F(3),A);

    M=INDEX(SIG,':');
    IF M=0 THEN DO;
      LB=SUBSTR(SIG,M-1,1);
      UB=SUBSTR(SIG,M+1,1);
      DO J=LB TO UB;
        I=I+1;
        JCHAR=J;
        NAME(I)=''||SUBSTR(SIG,1,M-2)||SUBSTR(JCHAR,4,1)||'','','';
      END;
    ELSE DO;
      I=I+1;
      J=INDEX(SIG,':');
      NAME(I)=''||SUBSTR(SIG,1,J)||'','','';
    END;

    GO TO BEGIN;
  FINAL: PUT FILE(SYSPNCH) EDIT
    ('DCL NAME(1000) CHAR(8) EXTERNAL INITIAL(')
    (SKIP,X(5),A);
    PUT FILE(SYSPNCH) EDIT
    ((NAME(L) DO L=1 TO I))
    (SKIP,COLUMN(5),A,COLUMN(20),A,COLUMN(35),A,COLUMN(50),A);
  END MAIN;
/*
//GO.SYSIN DD *

```

ACTE	ENABLE ACT=ACT+1
ADDE	ENABLE 32 BIT ADDER
ADDN(0:3)	INHIB BYTE 1 OF ADDER OUT TO AR
ADDOUT	PRINT RESULTS FROM ADDER
ADD5E	ENABLE 5 BIT ADDER
ADRG	ARITH REG TO DIST REG
ALCKE	ALIGNMENT CHECK ENABLE
AMCKE	ENABLE ACCESS MODE CHECK CIRCUIT
ALGNER	ALIGNMENT
ANBG	GATE ACT TO NB
ANDE	DR .AND. DB TO LR
AOVSG	GATE AOV TO SAOV
ARARFG	GATE AR BITS TO ARF SIGNALS
ARF33	STORAGE FOR BIT 33 OF ADDER
ARF34	STORAGE FOR 0JTH BIT OF ADDER
ARF35	STORAGE FOR ADDER BIT 35
ARG(0:3)	GATE BYTE(1) OF ARN TO AR

ARGG(0:3)	GATE BYTE I OF AR TO DR
ARL(0:3)	AR LATCH SIGNAL - BYTE I
ARPG	GATE AR TO PERMUTER
ASAO	AVAIL SPACE SEQ ADV OUT SIGS
ASBG	GATE ACT TO SNB
ASE	AVAILABLE SPACE EMPTY FF
ASRWE	ENABLE WRITE INTO ASSOC REG
AST	AVAILABLE SPACE SEQ. TASK SIG.
AUCG	AU FLAGS, TO LT,GT,EQ FROM LR,DR
BARG	GATE DB TO AR
BDCKE	ENABLE MEMORY BOUNDS CHECK
BDRG	GATE DB TO DR
BIRG	GATE DBP TO IR
BIRN(0:3)	INHIBIT DB TO IR GATE BYTE I
BLRG	GATE DB TO LR
BNRG	SET PER CONT TO GATE BR LEFT JUSTIFIED
BOV	MEMORY BOUNDS OVERFLOW
BOVI	BOUNDS OVERFLOW INDICATOR FLIP-FLOP
BRGS	SELECT BR PORTION OF BR-IBR STORAGE
BRS(0:7)	BASE REGISTER I SELECT
BRSBPG	BR STOR BLK(BR OR IBR) TO PER
BRSPG	GATE BROS TO PERMUTER
BRSSG	GATE BRN/S/ SIGS INTO BR SELECT
BSR1G	GATE DB TO SNR 8 - 14 INTRPT CYC1
BSR2G	GATE DB TO SNR0-7 INTRPT CYC2
CCSBE	ENABLE BYTE CS SIG FROM CONTROL
CCSDE	ENABLE D WORD CS SIG FROM CONTROL
CCSHE	ENABLE HWRD CS SIG FROM CONTROL
CCSWE	ENABLE WORD CS SIG FROM CONTROL
CCTE	ENABLE CCT=CCT+1
CDRE	GATE CABLE DRIVERS OF EXCH
CINJ	INJECT CARRY IN 32 BIT ADDER
CKOFE	ENABLE OS OVER-UNDERFLOW CHECK
CNBG	GATE CCT TO NB
CONS	IF 0 CELL IS FROM FREE LIST, IF 1 - CONT. STOR
CPAE	ENABLE ALGEBRAIC COMPARE
CPHE	ENABLE HOLLERITH COMPARE
CPL	ENABLE LOGICAL COMPARE
CRRLG	GATE CORE TO REG BY PER L JUST
CSBFE	ENABLE BYTE CS FF
CSDFE	ENABLE DOUBLE WORD CS FF
CSFG	GATE CS FF TO CELL SIZE SIGS
CSHFE	ENABLE HALFWORD CS FF
CSSE	ENABLE CS DECODE FORM IR
CSWFE	ENABLE WORC CS FF
CS2C	CONT SIG OS ENTRY - NEED 2 CELLS
CTD	COUNT DOWN ON MCT
CTDE	SET DOWN-SELECT SIG FOR MCT
CTMHZ	HIGH 2 BITS OF MCT=0
CTMLZ	LOW 3 BITS OF MCT=0
CTPG	GATE CABLE TERM TO PERMUTER
CTSBG	GATE CCT TO SNB
CTUE	SET UP-SELECT SIG FOR MCT
CYCTE	ENABLE COUNT ON CYCLE COUNTER
DBBRSG	GATE DBP TO BR-IBR STORAGE BLOCK
DBCTG	GATE DB TO ICT
DBDG(0:3)	GATE DB TO DR BYTE I

DBOSG	GATE DBP TO OS STORAGE
DBPRG	GATE DB TO PR OR SBR
DBRJ	RIGHT JUSTIFY PERMUTER OUTPUT
DBSTG	GATE DB TO OSTR AND SCR
DLRG	GATE DR TO LR DIRECT
DLRN(0:3)	INHIB DR TO LR BYTE I
DREDY	DATA READY FOR MEM SEQ WRITE
DRL(0:3)	DR LATCH SIGNAL BYTE I
DRPG	GATE DR TO PERMUTER INPUT
DUP	OBSOLETE
ENMCBE	ENABLE EXCHANGE NET MEMORY CONT
EQUE	ENABLE BOOLEAN EQUIVALENCE
EQVE	ENABLE BOOLEAN EQUIVALNCE
FLOSN	INHIB GATING OF FLAGS O OS
FLPRN	INHIB GATING OF FLAGS TO PR
FPT	FLOATING POINT
HOL	HOLERITH
IB(0:3)	INHIBIT BYTE I IN PERMUTER
IBC(0:3)	INHIBIT PER, BYTE I FROM CONT
IBE(0:3)	INHIBIT BYTE I IN PERMUTER
IBRS	SELECT IBR PART OF BR-IBR STORAGE
ICTPG	ENABLE ICT MASK INTO PER.
ILAC	ILLEGAL ACCESS MODE FF
INB1G	GATE NR TO DB INTRPT CYC1
INB2G	GATE NR TO DB INTRPT CYC2
IRG(0:3)	INHIBIT PER. BYTE I FROM CONTROL
IRHMG	
IRL(0:3)	IR LATCH SIGNAL BYTE I
IRLMG	OBSOLETE
IRPG	GATE IR TO PERMUTER INPUT
IRTGG(0:2)	GATE IR BYTE I TO TGR
KBG	GATE CONST TO B INPUT 5 BIT ADDER
LDREGE	ENABLE LOADING REGS FOR WRITE CYCLE
LDRG	GATE LR TO DR DIRECT
LFX	LONG FIXED PT. NUM.
LJIE	INHIB BYTES FOR LEFT JUST CELL
LJOPE	ENABLE LEFT JUSTIFY ON 2ND DATA BYTE
LKCTCE	ENABLE AVAIL SPACE LINK COUNT COMPARE
LNK	
LOC	
LRDG(0:3)	GATE LR TO DR BYTE I
LRG(0:3)	GATE DB TO LR BYTE I
LRL(0:3)	LR LATCH SIGNAL BYTE I
LRMCTG	GATE LOW 5 LR BITS TO MCT
LRPG	GATE LR TO PERMUTER INPUT
LRSE1	ENABLE LFT OR RT PER. SHIFT
LSE	ENABLE LEFT SHIFT, DR TO LR
LSPE	ENABLE LEFT SHIFT USING PERMUTER
MCTE	ENABLE M COUNTER TO COUNT
MZER	M COUNTER = ZERO
M1E	GEN NEG CONST = -1 IN PERMUTER
M16E	GEN NEG CONST = -16 IN PER.
M2E	GEN. NEG. CONST = -2 IN PERMUTER
M32E	GEN NEG CONST = -32 IN PER.
M4E	GEN NEG CONST = -4 IN PERMUTER
M8E	GEN NEG CONST = -8 IN PER.
NCSE	ENABLE DECODING OF FIELD DESIGNATION

NFCKE	ENABLE NAME FLAG CHECK
NFRG	SETS NFE 1 ON IF NCE 1 IS ON
NPR13S	FORCE SELECTION OF PR 13
NPR14S	FORCES ELECTION OF PR14
NOS	TGR = 0. INST. POINTER SELETED
OPJE	ENABLE OPCODE JUSTIFY ON BYTE 1
ORE	DB,OR. DR TO LR
OSBER	OPERAND STACK ERROR BOUNDS FF
OSCGE	ENABLE OS CONST GEN FROM CONTROL
OSCG1E	GENERATE 1 ON OS CONSTANT GEN
OSCG16E	GENERATE 16 ON OS CONSTANT GEN
OSCG2E	GENERATE 2 OS OS CONSTANT GEN
OSCG4E	GENERATE 4 OS OS CONSTANT GEN
OSCG8E	GENERATE 8 ON OS CONSTANT GEN
OSPG	GATE FALSE OUT OF OS TO PER
OSPS	
OSPTG	GATE TRUE OUT OF OS TO PER
OSRLG	
OSRRG	
OSTPG	GATE OSTR TO PER VIA OSR
OTAG	GATE OSTR TO A INPUT 5 BIT ADDER
OVFL	OVERFLOW ENT USED IN OS ENTRY
PART	PARTITIONED MODE FLIP-FLOP
PCHKE	ENABLE PARITY CHECK
PERR	PARITY ERROR FF INDICATOR
PL(0:3)	PERM LEFT I BYTES FROM SHIFT LOGIC
PLC0	PERMUTE LEFT 0 BYTES - CONTROL
PLC1	PERMUTE LEFT 1 BYTE- CONTROL
PLC2	PERMUTE LEFT 2 BYTES - CONTROL
PLC3	PERMUTE LEFT 3 BYTES - CONTROL
PMAREP	
PRPG	GATE PR BUS TO PERMUTER
PSNPRBG	GATE PRSNB TO PRB BUS INTO PERMUTER
PSNRR15	PR SEGMENT NAME REG 15 - READ
PSNRW15	PR SEGMENT NAME REG 15 - WRITE
P1E	GEN POS CONST=+1 ON PER
P16E	GEN. POS CONST=+16 IN PER
P2E	GEN POS CONST=+2
P32E	GEN POS CONST=+32 IN PER
P4E	GEN POS CONST=+4 IN PER
P8E	GEN POS CONST=+8 IN PER
QCNT	ENABLE COUNT ON ALL Q COUNTERS
QSETUP	
RACT	RESET ACT TO 00000
RASES	RESET ASE
RASTRS	RESET ASTRS
RBOVIS	RESET BOVI
RCOMFF	RESET COMPARE FFS - EQ,LT,GT
RCSAS	RESET CSA
RCT	RESET CCT TO 00000
RCYCTS	RESET CYCLE COUNTER
RDBRE	
RDCPRS	RESET DECPR
REG	
REQE	
REQS	RESET EQ FLIPFLOP
RGLE	RESET GT AND LT FLIPFLOPS

RIMMS	RESET IMM
RJIE	INHIB BYTES FOR RIGHT JUST CELL
RMCT	RESET M-COUNTER TO 00000
RMSRS	RESET MSTOR CON FF IN MEMORY
RNEGS	RESET NEG CON FF IN OS READ
RNFR	RESET NAME FLAG REG TO ZERO
RNWTOS	RESET NWTOS FF
RO	
ROKS	RESET OK'S 1 THRU 7 TO 0 STATE
ROPANS	RESET OPANDF
ROSBES	RESET OSBER
ROSCS	RESET OCS
ROSF	RESET OF OSF FLIPFLOP
RPBRE	REPLACE BR SELECTED BY Q COUNTERS
RPERS	RESET PER
RPOPS	RESET POP FF
RPROMS	RESET PROMOD
RQOVFS	RESET QUEUE CONTR OVFLOW BITS
RREPS	RESET PMAREP
RRSCNS	RESET RSCN
RSCTRS	RESET Q COUNTER SELECTED BY BRQS SI
RSDE	ENAB SORS, DEST BYTE ADD REVERSE
RSE	ENABLE RIGHT SHIFT, DR TO LR
RSES	
RSNR	RESET SHADOW NAME REGISTER
RSPE	ENABLE RIGHT SHIFT USING PER
RSUFS	RESET SUF FF
RWAITS	RESET WAIT CON FF IN OS READ
RWBGE	ENABLE READ/WRITE BYTE GENERATOR
RWRPS	RESET WRAP CON FF IN OS
SASES	SEY ASE
SASTRS	SET ASTRS FF
SBR	SELECT SPARE BUFFER REGISTER
SBSRG	GATE SNB TO SNR
SCAG	GATE SCR TO A INPUT 5 BIT ADDER
SCRPG	GATE SCR TO PER VIA OSR
SCSAS	SET CSA
SCSCS	USE CONTROL SIGS IN CS DECODER
SCSFS	SET CSFG OF CELL SIZE DECODER
SCSSS	SET CSSE OF CELL SIZE DECODER
SCT2G	GATE SCR TO T2R
SDCPRS	SET DECPR
SE	SUBTRACT ENABLE
SFX	SHORT FIXED
SGBOV	SEGMENT BOUNDS OVERFLOW
SGBOVS	SET SGBOV
SGDNT	SEGMENT DATA NOT THERE
SGMNT	SEGMENT MAP NOT THERE
SGMNTS	SET SGMNT
SGTRP	SEGMENT TRAP
SGTRPS	SET SGTRP
SGTS	SELECT GREATER THAN COMPARE
SILACS	SET ILAC
SIMMS	SET IMM
SLTS	SELECT LESS THAN COMPARE
SMSRS	SET MSTOR CON FF IN MEMORY
SN	

SNAMT	INDICATES PARTITIONED MODE ACCESS IS TO SNR
SNCSS	SET NCSE OF CELL SIZE DECODER
SNEGS	SET NEG CON FF IN OS READ
SNNRG	GATE SNR TO NR
SNRDE	ENABLE PR SEG NAME REG. READ
SNSG	GATE PR SELECT SIGS INTO SELECT
SNWTE	ENABLE PR SEG NAME REG. WRITE
SNWTOSS	SET NWTOS
SOPANS	SET OPANDF
SOSBES	SET OSBE FLIP-FLOP
SOSCS	SET OSC
SPERS	SET PER
SPOPS	SET POP FF
SPROMS	SET PROMOD
SREPS	SET PMAREP
SRSCNS	SET RSCN
SSUFS	SET STACK UNDERFLOW FLIP-FLOP
STBOV	SEGMENT NAME TABLE BOUNDS OVERFLOW INTERRUPT IND
STBOVS	SET STBOV INTERRUPT INDICATOR
STDNT	SEGMENT NAME TABLE DATA NOT THERE INT IND.
STDNTS	SET STDNT INTERRUPT INDICATOR
STMNTS	SEGMENT NAME TABLE MAP NOT THERE
STOR	CONTROL FF FOR OS READ
STTRPS	SET STTRP INTERRUPT INDICATOR
SUF	STACK UNDERFLOW FF
SWAITS	SET WAIT CON FF IN OS READ
SWRPS	SET WRAP CON FF IN OS
TIIE	ENABLE GENERATION OF TII ON EXCH NFI CONT BYTE
TNBG	GATE TGR TO NB
TSBG	GATE TGR TO SNB
TZE	ENABLE TEST ZERO
T1R2G	GATE T1R TO T2R
T1SCG	GATE T1R TO SCR
T1STG	GATE BITS 1,2 OF T1R TO OSTR
T2AG	GATE T2R TO A INPUT 5 BIT ADDER
T2BG	GATE T2R TO B INPUT K BIT ADDER
T2SCG	GATE T2R TO SCR
T2STG	T2R TO OSTR GATE
VAL	
WAIT	SCR MOD CON SIG - DON'T MOD PR13
WRBRE	READ/WRITE BR. WRITE WHEN 1
WRBRS	WRITE OUT BASE REGISTERS
WRUPSTK	WRITE OUT CONTENTS OF OPSTK
WRPRE	
WRPRLE	READ/WRITE PR LINK. WRITE WHEN 1
WRPRVE	READ/WRITE PR VALU. WRITE WHEN 1
WRPTRS	WRITE OUT POINTER REGISTERS
WRREGS	WRITE OUT AR,DR,LR,IR,DB,PRB,AND BROS
WRSBE	
WSBLE	READ/WRITE SBR LINK-VWRITE WHEN 1
WSBVE	READ/WRTIE SBR VALU-WRITE WHEN 1
XCHSTRT	START XCH SEQUENCE
XNNRP	EXCHANGE NET NO REPLY TO TP
XNREL	RELEASE EXCHANGE NET
XNRET	EXCHANGE NET RETURN SIGNAL
XORE	DB DR TO LR
XQCNT	COUNT ONCE ON ALL QCNTS NOT OV

X2	IF 0 MULT OS CONST GEN BY 2
AOV	32 BIT ADDER OVERFLOW BIT
A5OV	5 BIT ADDER OVERFLOW BIT
CEQA	CCT AND ACT ARE EQUAL
CSA	CON FF. COND SUB WAS ATTEMPTED
CSB	CELL SIZE IS BYTE
CSBH	CELL SIZE IS BYTE OR HALF WORD
CSD	CELL SIZE IS DOUBLE WORD
CSH	CELL SIZE IS HALF WORD
CSW	CELL SIZE IS WORD
CYCOV	CYCLE COUNTER OVERFLOW
CYC6	CYCLE COUNTER=6
C5INJ	CARRY INJECT INTO 5 BIT ADDER
DA	A BR WAS SELECTED BY ASSOC LOGIC
DECPR	DECREMENT PR13 FF
EQ	EQUAL FLIP-FLOP INDICATOR
GT	GREATER THAN FF INDICATOR
IBPL13	INHIB BYTE 1, PERMUTE LEFT 3
IBPL21	INHIB BYTE2, PERMUTE LEFT 1
IMM	CON FF INDICATES INSTR USES IMM OPTION
LRSE	ENABLE LFT OR RT PER. SHIFT
LT	LESS THAN FF INDICATOR
MSTOR	MEMORY STORE/READ CONTROL FF
NEG	CONTROL FF FOR OS READ
NFONE	NAME FLAG EQUALS ONE
NRSETG	INITIALIZE NAME REGS VIA ACT,CCT AND PERM.L
NT	
NWTOS	CONTROL FF FOR OS READ
NYET	NO BR WAS SELECTED BY ASSOC LIGIC
N13S	TGR=13, OS PR WAS SELECTED
OPANDF	CON FF. OPERAND COND SUB FAIL
OSC	CON FF. ON IF OS IS CLEARED
OSF	OPERAND STACK FF FULL FF
OSPG	GATE OPERAND STACK ONTO PER.
OSSS	SELECT SOME OSS(I)
OV1	OS OVERFLOW- NO ROOM FOR 1 CELL
OV2	OS OVERFLOW- NO ROOM FOR 2 CELLS
PER	CON FF PERMUTE NAMES IN IMP INST
POP	CONTROL FF FOR OS ENTRY
PROMOD	FF=1 IF PRO IS MODIFIED
RSCN	CON FF IMP IN RE-SCAN CYCLE
RSYS	RESET SYSTEM
SQTOV	QCNTER SELECTED BY BRQS HAS OVERFLOWED
TGIM	TAG REGISTER IMM BIT
TRAP	
UF1	OS UNDRF-LESS THAN 1 CELL LEFT
UF2	OS UNDRF - LESS THAN 2 CELLS LEFT
WRAP	WRAP FLIP-FLOP
XRVE	ON IF XORE EQVE=1
ZSCR	LOW ORDER 3 BITS OF SCR=0
ZZZZZZZZ	END MARKER FOR SIGNAL NAMES

3. Utility Routines

The utility routines consist of the following procedures, CLEAR, PRINT1, REGSOUT, PTRSOUT, BRSOUT, OSOUT, and MEMOUT. The first is used as an initialization routine while the others allow for the display of various registers, the operand stack, and the contents of Memory Storage for diagnostic purposes. Except for CLEAR these routines are usually called from a control point in a control sequence, by way of special "control signals". These signals are placed at the end of the regular signals to be turned on for a control point. For example, the instruction, CP1: CALL CP('CP1', 'ADDE, ..., TLSCG|WRPSTK. '); will cause OSOUT to be called at the completion of the control point, which will print out the contents of the operand stack.

3.1 CLEAR Procedure

3.1.1 CLEAR Description

The CLEAR procedure sets all the externally declared variables and arrays which are used in the simulator to zero. All similarly declared variables (arrays with the same dimensions) are set to zero in a multiple assignment statement.

CLEAR also provides a listing and definitions of all the external variables used in the simulator with the exceptions of those which are contained in the "ON" and "NAME" arrays.

If CLEAR is compiled before any of the other procedures, there is no need to call it at the beginning of a main program, since CLEAR initializes all the variables to zero when space is allocated for them by the loader.

Some of the variables in CLEAR are not zeroed but are initialized to some value. The OS bytes are loaded with the numbers 0 to 31. The Queue counters are loaded with 0 to 6. The Name Registers are loaded with 0 to 14, and the associative registers are loaded with 0 to 6.

CLEAR may be called more than once in a main test program if the program is doing more than one independent operation, and clearing of all the registers is desired.

The calling sequence is simply:

CALL CLEAR;

This procedure has no parameters since all of the variables are declared external.

3.1.2 CLEAR Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
  CLEAR: PROC;
    DCL(

      /* 36 BIT REGISTERS & COUNTERS */

      (AR,      /* ARITHMETIC REGISTER          */
      ARN,      /* AR INPUT BUS                */
      BROS,     /* BR, OS, IBR OUTPUT          */
      CD,       /* CABLE DRIVERS               */
      CT,       /* CABLE TERM TO EXCH          */
      DB,       /* DISTRIBUTION BUS            */
      DBB,      /* CONT SECT DB DRIVER BUS     */
      DBP,      /* PERMUTER OUTPUT             */
      DR,       /* DISTRIBUTION REGISTER       */
      IR,       /* INSTRUCTION REGISTER        */
      LR,       /* LOGIC REGISTER              */
      LRN,      /* LR INPUT BUS,               */
      PRB,      /* POINTER REG BIS             */
      PRSNB,    /* POINTER REGISTER SEGMENT NAME BIS */
      SBR,      /* SPARE BUFFER REGISTER       */

      T,
      ARB,
      LRB,
      DRB)

      (1:36) INITIAL ((36) (1) '0'B),

      XNCB      /* EXCHANGE NET CONTROL BYTE   */
      (1:10) INITIAL ((10) (1) '0'B),

      /*      3 BIT REGISTERS AND COUNTERS

      (BNR,     /* BASE NAME REGISTER          */
      ICT)      /* INSTRUCTION BYTE COUNTER    */
      (1:3) INITIAL ( (3) (1) '0'B),

      /* 5 BIT REGISTERS

      (T2R,     /* OS TEMP REGISTER 2         */
      K,        /* CONSTANT GENERATOR OUTPUT  */
      SCR)      /* OP STACK CONTROL REGISTER   */
      (1:5) INITIAL ( (5) (1) '0'B),

      /* OS TEMP REGISTER 1

      T1R      (0:5) INITIAL ( (6) (1) '0'B),

      /* OPERAND STACK TOP REGISTER

      OSTR      (1:2) INITIAL ( (2) (1) '0'B),

      /* 4 BIT REGISTERS & COUNTERS

      (ACT,     /* AUXILIARY COUNTER          */
      CCT,     /* 4 BIT COUNTER              */
      CYCTR,    /* CYCLE COUNTER              */

```

```

TGR)      /* TAG REGISTER                                */
          (1:4) INITIAL ( (4) (1) '0'B),

          /* NAME REGISTERS                                */

(NR,      /* NAME REGISTER                                */
SNR)      /* SHADOW NAME REGISTER                                */
          (0:14,4) INITIAL ( (60) (1) '0'B),

          /* NAME FLAG REGISTER                                */

NFR      (0:14) INITIAL ((15) (1) '0'B),
/*POINTER REGISTER SELECT SIGNALS */
PRS      (0:14) INITIAL ((15) (1) '0'B),

          /* POINTER REGISTER                                */

PR      (0:14,36) INITIAL ( (540) (1) '0'B),

          /* 16 BIT SIGNALS                                */

(PSNRS1, /*PR SEG. NAME REG READ SELECT 1 */
PSNRS2, /* PR SEG. NAME REG READ SELECT 2 */
PSNWS1, /* PR SEG. NAME REG WRITE SELECT 1 */
PSNWS2, /* PR SEG. NAME REG WRITE SELECT 2 */
SNSRG)  (0:15) INITIAL ( (16) (1) '0'B),

(CMA,    /* 1ST QUARTER OF BR(I) MATCHES PRSNB */
CMB,    /* 2ST QUARTER OF BR(I) MATCHES PRSNB */
CMC,    /* 3RD QUARTER OF BR(I) MATCHES PRSNB */
CMD,    /* 4TH QUARTER OF BR(I) MATCHES PRSNB */
OK )    /* DATA IN ASSOC REG FOR BR(I) IS VALID */
          (0:7) INITIAL ((8) (1) '0'B),

          /* 7 BIT REGISTERS */

(ASRWE1, /* ASSOCIATIVE REG(I), WRITE ENABLE1 */
ASRWE2, /* ASSOCIATIVE REG(I), WRITE ENABLE 2 */
BRQS,   /* BR(I) SELECT FOR QUEUE COUNTERS */
RBRIS)  /* BR(I) SELECT FOR REPLACEMENT */
          (1:7) INITIAL ( (7) (1) '0'B),

          /* 8 BIT REGISTERS */

(BRRS)  /* BR(I) READ OUT SELECT BY ASSOC LOGIC */
          (0:7) INITIAL ((8) (1) '0'B),

          /* BASE REGISTERS */

(BY,
OSS)    (0:31) INITIAL ((32) (1) '0'B),
OS      (0:31,9) INITIAL ((288) (1) '0'B),

```

```

        QCOUNT      (1:7,4) INITIAL ((28) (1) '0'B),
        ASRG (7,18) INITIAL ((126) (1) '0'B),
        PRSNR        (0:15,18) INITIAL ( (286) (1) '0'B),
        BR           (0:7,36) INITIAL ( (288) (1) '0'B)
                ) BIT(1) EXTERNAL;
/*CLEAR REGISTERS DECLARED (1:36) */
AR,      ARN,      BROS,      DB,      DBB,      DBP,
CD,      CT,      DR,      IR,      LR,      LRN,
PRB,      SBR,      T,      PRSNB,
        ARB,      LRB,      DRB      = '0'B;
PRSNR='0'B;
/* CLEAR REGISTERS DECLARED(0:31) */
BY,      OSS      = '0'B;
/*LOAD OS BYTES WITH CONSTANTS */
ACT='0'B;
DO I = 0 TO 31;
DO J= 1 TO 4;
OS(I,J+4)=ACT(J);
END;
CALL ADD1(ACT);
END;
/* CLEAR REGISTERS DECLARED (0:15) */
PSNRS1, PSNRS2, PSNWS1, PSNWS2, SNSRG      = '0'B;
/*CLEAR REGISTERS DECLARED (0:14) */
NFR,      PRS      = '0'B;
/*CLEAR REGISTERS DECLARED (0:14,4) */
SNR      = '0'B;
/*CLEAR REGISTERS DECLARED (0:14,36) */
PR      = '0'B;
/*CLEAR REGISTERS DECLARED (0:7) */
BRRS,      CMA,      CMB,      CMC,      CMD,      = '0'B;
OK      = '0'B;
/*CLEAR REGISTERS DECLARED(1:7) */
ASRWE1, ASRWE2, BRQS,      RBR      = '0'B;
/*CLEAR REGISTERS DECLARED (0:5) */
T1R      = '0'B;
/* CLEAR REGISTERS DECLARED (1:5) */
K,      SCR,      T2R      = '0'B;
/*CLEAR REGISTERS DECLARED (1:4) */
ACT,      CCT,      CYCTR,      TGR      = '0'B;
/*CLEAR REGISTERS DECLARED (1:3) */
BNR,      ICT      = '0'B;
/*CLEAR REGISTERS DECLARED (1:2) */
OSTR      = '0'B;
/* GENERATE NUMBERS 0 TO 6 ON QCOUNTS 1 TO 7 RESPECTIVELY */
QCOUNT='0'B;
QCOUNT(2,4),QCOUNT(3,3),QCOUNT(4,3),QCOUNT(4,4),
QCOUNT(5,2),QCOUNT(6,2),QCOUNT(6,4),
QCOUNT(7,2),QCOUNT(7,3) = '1'B;
/* INSTEAD OF ZEROING THE NAME REGISTERS, GENERATE 0 TO 14
ON NR(0) TO NR(14) RESPECTIVELY. */
DO I=0 TO 14;
NR(I,*)=ACT(*);
CALL ADD1(ACT);
END;
ASRG='0'B;
ASRG(2,17)='1'B;

```



```
ASRG(3,16)='1'B;  
ASRG(4,16),ASRG(4,17)='1'B;  
ASRG(5,15)='1'B;  
ASRG(6,15),ASRG(6,17)='1'B;  
ASRG(7,15),ASRG(7,16)='1'B;  
/*CLEAR BASE REGISTERS */      BR='0'B;  
/*CLEAR EXCHANGE NET CONTROL BYTE */      XNCB='0'B;  
END CLEAR;
```

/*

3.2 PRINT1 Procedure

3.2.1 PRINT1 Description

The PRINT1 procedure consecutively prints out the bits of any 36 bit register. The flag bit (9th bit) of each register byte is printed out following the eighth bit of the byte, but one line above the preceding eight bits.

The procedure has as formal parameters REGISTER and NAME. REGISTER is the 36 bit register to be printed (i.e., AR, DR, etc.) and NAME is a four character string constant which will be printed as the name of the register (i.e., 'AR ', 'DR ', etc.). The procedure breaks the non-flag bits of REGISTER into 4 arrays of length 8 (BYTE1, BYTE2, BYTE3, BYTE4). The flag bits of REGISTER are put into 4 simple variables (BIT1, BIT2, BIT3, BIT4). First the flags are printed out on one line by printing out BIT1...BIT4. Then on the next line the non-flag bits are printed out by printing out BYTE1...BYTE4. Preceding these bits, a character string (length 4) and an equal sign are printed.

This procedure is called by the procedures REGSOUT and PTRSOUT which print out several of the most often used registers. However, PRINT1 can be called from anywhere in the simulator or main program. The calling sequence is:

```
CALL PRINT1 (REGISTER, 'RNAME');
```

where RNAME is the name of the register stored in REGISTER. This is the name which will be printed with the output and will be 4 characters long.

3.2.2 PRINT1 Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
    PRINT1:  PROC(REGISTER,NAME);
/*PRINT1 PRINTS OUT ONE REGISTER.*/
    DCL REGISTER(36) BIT(1);
    DCL NAME CHAR(4),(BYTE1,BYTE2,BYTE3,BYTE4)(1:8) BIT(1),
        I FIXED BIN, (BIT1,BIT2,BIT3,BIT4) BIT(1);
    DO I=1 TO 8;
        BYTE1(I)= REGISTER(I);
        BYTE2(I)= REGISTER(I+9);
        BYTE3(I)= REGISTER(I+18);
        BYTE4(I)= REGISTER(I+27);
    END;
    BIT1=REGISTER(9);
    BIT2=REGISTER(18);
    BIT3=REGISTER(27);
    BIT4=REGISTER(36);
    PUT EDIT(BIT1,BIT2,BIT3,BIT4)(SKIP,X(15),B,3(X(8),B));
    PUT EDIT(NAME,' = ',BYTE1,BYTE2,BYTE3,BYTE4)
        (SKIP,A(4),A, 4(8 B(1),X(1)));
END PRINT1;
/*
```

3.2.3 PRINT1 Variable List

<u>Simulation Name</u>	<u>Variable Description</u>
BIT 1	First flag bit of REGISTER
BIT 2	Second flag bit of REGISTER
BIT 3	Third flag bit of REGISTER
BIT 4	Fourth flag bit of REGISTER
BYTE 1(i)	Data bits of the first byte of REGISTER
BYTE 2(i)	Data bits of the second byte of REGISTER
BYTE 3(i)	Data bits of the third byte of REGISTER
BYTE 4(i)	Data bits of the fourth byte of REGISTER
NAME	Formal parameter. Character string of length 4 which is printed out as name of register printed.
REGISTER	36 bit global variable which must be loaded with contents of register to be printed before call is made.

3.2.4 PRINT1 Sample Output

		0		0		C		0
IR	=	01000001	00000000	00000000	00000001			

Produced by:

```
CALL PRINT1 (IR, 'IR  ');
```

3.3 REGSØUT Procedure

3.3.1 REGSØUT Description

The REGSØUT procedure is used to print out several 36 bit registers. The procedure consists of repeated calls to PRINT1, using as parameters the register itself, and a 4 character string which represents the name of that register. Presently, this is done for the main registers of the permuter (AR, IR, LR, DR, DB, BROS, PRB, CT, CD, and PRSNB). Any register can be easily added or subtracted from the registers to be printed out.

The control signal which is used to indicate desired writing of the registers is "WRREGS."

The calling sequence in the procedure SIMCALL is:

IF WRREGS THEN CALL REGSØUT;

3.3.2 REGSOUT Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
  REGSOUT: PROC;
/*REGSOUT PRINTS OUT THE MAIN REGISTERS USED IN
  THE PERMUTER.*/
  DCL (AR,DB,DR,IR,LR,BROS,PRB)(36) BIT(1) EXTERNAL;
  DCL (CT,CD)(36) BIT(1) EXTERNAL;
  DCL PRSNB(36) BIT(1) EXTERNAL;
  CALL PRINT1(AR,'AR  ');
  CALL PRINT1(IR,'IR  ');
  CALL PRINT1(LR,'LR  ');
  CALL PRINT1(DR,'DR  ');
  CALL PRINT1(DB,'DB  ');
  CALL PRINT1(BROS,'BROS');
  CALL PRINT1(PRB,'PRB ');
  CALL PRINT1(CT,'CT  ');
  CALL PRINT1(CD,'CD  ');
  CALL PRINT1(PRSNB,'PRSN');
  END REGSOUT;
```

/*

3.3.3 REGSOUT Variable List

Registers and buses (external variables):

<u>Simulation Name</u>	<u>Description</u>
AR	Arithmetic Register
BROS	Base Register - Operand Stack Bus
CD	Cable Drivers
CT	Cable Terminators
DB	Distribution Bus
DR	Distribution Register
IR	Instruction Register
LR	Logic Register
PRB	Output Bus from Pointer Register
PRSNB	PR Segment Name Register Output Bus

3.3.4 REGSOUT Sample Output

		0	0	C	1
AR	=	00000000	00000000	00000000	00000000
		0	C	C	0
IR	=	01000001	00000000	00000000	00000001
		C	C	C	0
LR	=	00000000	00000000	00000000	00000000
		0	0	C	0
DR	=	00000000	00000000	00000000	00000100
		0	0	C	0
DB	=	00000000	00000000	00000000	00000000
		0	0	C	0
BROS	=	00000000	00000000	00000000	00000000
		0	C	C	0
PRB	=	00000000	00000000	00000000	00010000
		0	C	C	C
CT	=	00000000	00000000	00000000	00000000
		0	C	C	0
CD	=	00000000	00000000	00000000	00000000
		0	0	C	0
PRSN	=	00000000	00000000	00000000	00000000

produced by:

CALL REGSOUT;

3.4 PTRSØUT Procedure

3.4.1 PTRSØUT Description

The PTRSØUT procedure is used to print out the contents of the pointer register (PR(0) - PR(14)) the Spare Buffer Register, (SBR) and the Pointer Register segment Name Registers (PRSNR(0) - PRSNR(15)).

The procedure prints out a consecutive series of pointer registers designated by the Calling Sequence. The printing is done using the PRINT1 procedure after first inspecting the name registers to see which PR was the name of the PR to be printed. The name which is printed out along with the data and which is used as the PRINT1 parameter is obtained from a table called TABLE which contains 15 entries containing the 4 character strings 'PRO' through 'PR14'.

After each pointer register is printed, the corresponding PRSNR is printed next to it, omitting the flags (which really do not exist in the hardware). After the last PR is printed out, the SBR is printed, and next to it the last PRSNR.

The calling sequence is:

CALL PTRSØUT (P1, P2); where P1 is a decimal number indicating the number of the first pointer register to be printed and P2 is the decimal number indicating the last pointer register to be printed.

The control signal which is used to indicate desired printing of all the pointer registers is "WRPTRS". The calling sequence in SIMCALL is:

IF WRPTRS THEN CALL PTRSØUT (0,14);

3.4.2 PTRSOUT Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
PTRSOUT: PROC (START, END);
/*PTRSOUT PRINTS OUT DESIGNATED POINTER REGISTERS.*/
  DCL (NR(0:14,4), PR(0:14,36)) BIT(1) EXTERNAL;
  DCL REGISTER(36) BIT(1);
  DCL PRSNR(0:15,18) BIT(1) EXTERNAL;
  DCL (PART1,PART2) (8) BIT(1);
  DCL SBR(36) BIT(1) EXTERNAL;
  DCL PRINT1 ENTRY EXTERNAL;
  DCL (I,K) FIXED BIN (4,0);
  DCL J FIXED BIN;
  DCL (START,END) FIXED BIN, REGNAME CHAR(4), TABLE(0:14)
    CHAR(4) INITIAL ('PRO ', 'PR1 ', 'PR2 ', 'PR3 ', 'PR4 ',
    'PR5 ', 'PR6 ', 'PR7 ', 'PR8 ', 'PR9 ', 'PR10 ', 'PR11 ',
    'PR12 ', 'PR13 ', 'PR14 '), BITS BIT(4), BITA(4) BIT(1);
  DO I= START TO END;
    BITS = I;
    DO K=1 TO 4;
      BITA(K) = SUBSTR(BITS,K,1);
    END;
    J = 0;
    DO WHILE(J<15);
      IF BITA(1)=NR(J,1) & BITA(2)=NR(J,2) &
        BITA(3)=NR(J,3) & BITA(4)=NR(J,4) THEN GO TO PNT;
      J = J + 1;
    END;
    PNT: REGISTER=PR( J,*);
    REGNAME=TABLE(J);
    CALL PRINT1(REGISTER,REGNAME);
    NEXT: DO L=1 TO 8;
      PART1(L)=PRSNR(J,L);
      PART2(L)=PRSNR(J,L+9);
    END;
    PUT EDIT('PRSNR(',J,')= ',PART1,PART2)
      (X(10),A,F(2),A,8 B(1),X(1),8 B(1));
    END;
    CALL PRINT1(SBR,'SBR ');
    DO L=1 TO 8;
      PART1(L)=PRSNR(15,L);
      PART2(L)=PRSNR(15,L+9);
    END;
    PUT EDIT('PRSNR(15)= ',PART1,PART2)
      (X(10),A,8 B(1), X(1), 8 B(1));
    END PTRSOUT;
/*
```

3.4.3 PTRSOUT Variable List

Local Variables:

<u>Simulation Name</u>	<u>Description</u>
END	Second formal parameter. Decimal number indicating last PR to be printed.
REGISTER	36 bit local variable used as temporary storage for the PR passed as an actual parameter in calling PRINT1.
REGNAME	Used as temporary storage for the name of the register passed as an actual parameter to PRINT1.
START	First formal parameter. Decimal number indicating first PR to be printed.
TABLE	Array of 4 character strings. Used to hold the names of the 15 PR's.

External Variables:

<u>Simulation Name</u>	<u>Description</u>
NR(i,j)	Name Register i - bit j 15 registers of 4 bits each
PR(i,j)	Pointer Register i - bit j 15 registers of 36 bits each
PRSNR(i,j)	Pointer Register i - bit j 16 registers of 18 bits each
SBR(j)	Spare buffer register - bit j 36 bits long.

3.4.4 PTRSOUT Sample Output

PR0 = 00000000 00000000 00000000 00000000	PRSNR(0) = 00000000 00000000
PR1 = 00000000 00000000 00000000 00000000	PRSNR(1) = 00000000 00000000
PR2 = 00000000 00000000 00000000 00000000	PRSNR(2) = 00000000 00000000
PR3 = 00000000 00000000 00000000 00000000	PRSNR(3) = 00000000 00000000
PR4 = 00000000 00000000 00000000 00000000	PRSNR(4) = 00000000 00000000
PR5 = 00000000 00000000 00000000 00000000	PRSNR(5) = 00000000 00000000
PR6 = 00000000 00000000 00000000 00000000	PRSNR(6) = 00000000 00000000
PR7 = 00000000 00000000 00000000 00000000	PRSNR(7) = 00000000 00000000
PR8 = 00000000 00000000 00000000 00000000	PRSNR(8) = 00000000 00000000
PR9 = 00000000 00000000 00000000 00000000	PRSNR(9) = 00000000 00000000
PR10 = 00000000 00000000 00000000 00000000	PRSNR(10) = 00000000 00000000
PR11 = 00000000 00000000 00000000 00000000	PRSNR(11) = 00000000 00000000
PR12 = 00000000 00000000 00000000 00000000	PRSNR(12) = 00000000 00000000
PR13 = 00000000 00000000 00000000 00000000	PRSNR(13) = 00000000 00000000
PR14 = 00000000 00000000 00000000 00000000	PRSNR(14) = 00000000 00000000
SAR = 00000000 00000000 00000000 00000000	PRSNR(15) = 00000000 00000000

Produced by:

CALL PTRSOUT (0, 14);

3.5 BRSOUT Procedure

3.5.1 BRSOUT Description

This procedure prints out the contents of the Base Registers (BR), the Associative Registers (ASRG), and the Q-counters (QCOUNT).

The BR's are declared as four bytes each, but of these 4 bytes, only 3 of them are actually a part of each base register. To complicate matters even more, the zeroth, first, and second bytes of each Base Register are in different positions of the four BR bytes. For example bytes 0, 1, and 2 of Base Register 0 correspond to bytes 0, 1, 2 of BR(0). But bytes 0, 1, and 2 of Base Register 1 correspond to bytes 4, 1, and 2 of BR(1) respectively.

For this reason, some selection process was needed to determine which bytes, of each BR were to be printed, and in what order. The three bytes desired are put into the BYTES array for future printing. The flags of these selected bytes are put into the FLAGS array. Once the BYTES array and FLAGS array are filled, the printing is done. First FLAGS(0) (the flag bits of BR(0)) are printed out. Underneath them BYTES(0,*,*) (the three data bytes of BR(0)) are printed. Next the flags of BR(1) are printed. On the next line ASRG(1), QCOUNT(1) and the data bytes of BR(1) are printed. This process is continued till all eight of the base registers are printed, as well as all seven of the Associative Registers and Queue Counters.

The "utility control signal" used in a call to CP to activate this procedure is "WRBRS".

The calling sequence in SIMCALL is:

IF WRBRS THEN CALL BRSOUT;

BRSOUT may also be called from anywhere in the simulator by the calling sequence: CALL BRSOUT;

3.5.2 BRSOUT Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
  BRSOUT: PROC;
    DCL ASRG(1:7,1:18) BIT(1) EXTERNAL;
    DCL QCOUNT(1:7,1:4) BIT(1) EXTERNAL;
    DCL BR(0:7,1:36) BIT(1) EXTERNAL;
    DCL BYTES(0:7,0:2,1:8) BIT(1);
    DCL FLAGS(0:7,0:2) BIT(1);
    /* GATE THE CORRECT BYTE OF EACH BR(I) INTO BYTE(I) */
    L=0;
    DO I=0 TO 7;
      DO J=0 TO 2;
        /* GATE BYTE L OF BR(I) INTO BYTE(J) OF OF BYTES(I) */
        DO K=1 TO 8;
          BYTES(I,J,K)=BR(I,L*9+K);
        END;
        /*SAVE THE FLAG OF BYTE L OF BR(I) */
        FLAGS(I,J)=BR(I,L*9+9);
        L=L+1;
        IF (L=5) THEN L=0;
      END;
    END;

    DO I=0 TO 7;
      PUT EDIT(FLAGS(I,0),FLAGS(I,1),FLAGS(I,2))
        (SKIP,COL(65),B(1),COL(74),B(1),COL(83),B(1));
      PUT SKIP;
      IF I=0 THEN GO TO NEXT;
      PUT EDIT('ASRG(',I,')= ',(ASRG(I,J) DO J= 1 TO 8),
        (ASRG(I,J) DO J= 10 TO 17))
        (COL(1),A(5),F(1),A(3),8 B(1),X(1),8 B(1));
      PUT EDIT('QCOUNT(',I,')= ',(QCOUNT(I,J) DO J=1 TO 4))
        (COL(30),A(7),F(1),A(3),4 B(1));
    NEXT: PUT EDIT('BR(',I,')= ',((BYTES(I,J,K) DO K=1 TO 8)
      DO J=0 TO 2))
      (COL(50),A(3),F(1),A(3),8 B(1),X(1),8 B(1),
        X(1),8 B(1),X(1));

    END;
  END BRSOUT;
/*
```


3.5.3 BRSØUT Variable List

External Variables:

<u>Simulation Name</u>	<u>Description</u>
ASRG(i,j)	Associative Register i, bit j
BR(i,j)	Base Register i, bit j
QCØUNT(i,j)	Queue Counter i - bit j

Local Variables:

<u>Simulation Name</u>	<u>Description</u>
BYTES(i,j,k)	Used to hold byte j, bit k of BRi
FLAGS(i,j)	Used to hold flag j of BRi

3.5.4 BRSØUT Sample Output

					1	1	1
				PR(0)=	00000000	00000000	00000000
					1	1	1
ASRG(1)=	00000000	00000000	CCOUNT(1)=	0000	PR(1)=	00000000	00000000
					1	1	1
ASRG(2)=	00000000	00000001	CCOUNT(2)=	0001	PR(2)=	00000000	00000000
					1	1	1
ASRG(3)=	00000000	00000010	CCOUNT(3)=	0010	PR(3)=	00000000	00000000
					1	1	1
ASRG(4)=	00000000	00000011	CCOUNT(4)=	0011	PR(4)=	00000000	00000000
					1	1	1
ASRG(5)=	00000000	00000100	CCOUNT(5)=	0100	PR(5)=	00000000	00000000
					1	1	1
ASRG(6)=	00000000	00000101	CCOUNT(6)=	0101	PR(6)=	00000000	00000000
					1	1	1
ASRG(7)=	00000000	00000110	CCOUNT(7)=	0110	PR(7)=	00000000	00000000

Produced by:

CALL BRSØUT:

3.6 ØSØUT Procedure

3.6.1 ØSØUT Description

ØSØUT prints out the contents of the 32 bytes of the operand stack, ØS, values of the SCR and the ØSTR and states of the signals ØV1, ØU2, UF1, and UF2. These latter four signals are parameters to ØSØUT.

After printing a heading, "Operand Stack", the procedure prints out each of the ØS bytes by first printing out the byte number (0 to 31) followed on the next line by the flag bits (the 9th bit of each byte), which are then followed on the next succeeding line by the eight data bits. Only 8 bytes are printed across the page so that group 1 contains bytes 0 through 7, group 2 contains bytes 8 through 15, etc.

Then the contents of the SCR and the ØSTR are decoded into their decimal equivalence. For example, if the SCR = 00111, then its decimal equivalence is 7. This is printed out. The two bits of the ØSTR are decoded as if they were the leading two bits of a 5 bit number. This is done because, while the ØSTR actually marks the double word boundary, (i.e. 0,1,2 or 3) it is convenient to note which byte this is (0, 8, 16, or 24). Then the decoded value of the ØSTR is printed out. Following this are the values of ØV1, ØU2, UF1, and UF2.

In the control sequences, if it is desired to write the operand stack contents after the operation of a control point, this can be indicated in the call to CP by the signal "WRØPSTK". At the end of the SIMCALL procedure, if WRØPSTK = 1, then ØSØUT is called.

The calling sequence in SIMCALL is:

```
IF WROPSTK THEN CALL ØSØUT(ØU1, ØV2, UF1, UF2);
```

3.6.2 OSOUT Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
OSOUT: PROC(OV1, OV2, UF1, UF2);
        DCL(OV1, OV2,          UF1,          UF2) BIT(1);
        DCL ( OS(0:31,9), OSTR(1:2), SCR(1:5)) BIT(1) EXTERNAL;

        /* THIS PROCEDURE WILL PRINT OUT THE CONTENTS OF THE 32 BYTES
        OF THE OPERAND STACK, AND THE VALUES OF:
        OV1 (1 IF THERE IS NO ROOM IN OS FOR CELL OF CURRENT SIZE
        OV2 (1 IF THERE IS NO ROOM IN OS FOR 2 CELLS OF CURRENT SIZE
        UF1 (1 IF OS DOES NOT CONTAIN 1 CELL THE SIZE OF CS)
        UF2 (1 IF OS DOES NOT CONTAIN 2 CELLS THE SIZE OF CS)
        SCR, (STACK CONTROL REG - GIVES CURRENT ENTRY POINT TO OS
        OSTR ( OPERAND STACK TOP REGISTER - VALUE IS THE BYTE
                ADDRESS OF THE DOUBLE WORD WHICH IS AT THE BOTTOM
                OF THE OS IN THE TP'S HARDWARE STORAGE (0,8,16,24)
                */

        DCL (X,I,J,L) FIXED BIN;
        PUT EDIT ( 'OPERAND STACK' ) (SKIP(4),X(45),A);
        DO L= 0 TO 3;
            PUT EDIT ( ( I DO I= (L*8) TO (L*8+7)))
                (SKIP(2),X(3),8 (F(2), X(13)))

                ((OS(I,9) DO I=(L*8) TO (L*8+7)))
                (SKIP, X(8), 8 (B(1), X(14)))

                (((OS(I,J) DO J=1 TO 8) DO I=(L*8) TO (L*8+7)))
                ( SKIP, 8 ( 8 B(1), X(7))));

        END;
NEXT: X=0;
        DO I=1 TO 5;
            IF SCR(I) THEN X=X+2**(5-I);
        END;
        PUT EDIT ('SCR = ',X) (SKIP(2),A,F(3));
        X=0;
        IF OSTR(1) THEN X=X+16;
        IF OSTR(2) THEN X=X+8;
        PUT EDIT ('OSTR= ',X) (SKIP,A,F(3));
        PUT EDIT ('OV1 =',OV1,
                'OV2 =',OV2,
                'UF1 =',UF1,
                'UF2 =',UF2) (SKIP,A,X(3),B(1));

END OSOUT;
/*
```

3.6.3 ØSØUT Variable List

Parameters (1 bit each);

<u>Simulation Name</u>	<u>Description</u>
ØU1	There is no room in ØS for 1 cell of current size
ØU2	There is no room in ØS for 2 cells of current size
UF1	ØS does not contain 1 cell of current size
UF2	ØS does not contain 2 cells of current size

External Variables:

<u>Simulation Name</u>	<u>Description</u>
$\emptyset S(i,j)$	Operand Stack byte i - bit j
$\emptyset STR(i)$	Operand Stack Top Register - bit i
$SCR(i)$	Stack Control Register - bit i

3.6.4 ØSØUT Sample Output

OPERAND STACK							
0	1	2	3	4	5	6	7
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
8	9	10	11	12	13	14	15
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
16	17	18	19	20	21	22	23
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
24	25	26	27	28	29	30	31
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
SCR = 7							
OSTR = C							
OV1 = C							
OV2 = C							
UF1 = C							
UF2 = C							

Produced by:

CALL ØSØUT (ØU1, ØU2, UF1, UF2);

3.7 MEMOUT Procedure

3.7.1 MEMOUT Description

The MEMOUT procedure is a utility routine which prints out the contents of the two pages of memory which are now simulated.

MEMOUT prints out one doubleword of the memory (MEMSTOR) on each line, with the two single words separated and the flags raised, for ease of reading.

The procedure operates in much the same way as does PRINT1.

MEMOUT is called from SIMCALL if the utility signal WRMEM is on. This signal can be set in a call to CP in the same way as the other utility signals are set.

The sample Printout of the routine has been omitted since it is so large.

3.7.2 MEMOUT Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
MEMOUT: PROC;
  DCL MEMSTOR(0:128,0:1,1:36) BIT(1) EXTERNAL;
  DCL (BIT1,BIT2,BIT3,BIT4) BIT(1);
  DCL (BYTE1,BYTE2,BYTE3,BYTE4) (1:8) BIT(1);
  PUT EDIT('MEMORY STORAGE CONTENTS')(SKIP(2),X(20),A);
  DO K=0 TO 128;
    PUT EDIT(' ')(SKIP,COLUMN(11),A);
    DO J=0 TO 1;
      BIT1=MEMSTOR(K,J,9);
      BIT2=MEMSTOR(K,J,18);
      BIT3=MEMSTOR(K,J,27);
      BIT4=MEMSTOR(K,J,36);
      PUT EDIT(BIT1,BIT2,BIT3,BIT4
        (X(14),B,3 (X(8),B)));
    END;
    PUT EDIT('DWORD ',K,' =')(SKIP,A,F(3),A);
    DO J=0 TO 1;
      DO I=1 TO 8;
        BYTE1(I)=MEMSTOR(K,J,I);
        BYTE2(I)=MEMSTOR(K,J,I+9);
        BYTE3(I)=MEMSTOR(K,J,I+18);
        BYTE4(I)=MEMSTOR(K,J,I+27);
      END;
      PUT EDIT(BYTE1,BYTE2,BYTE3,BYTE4)(X(7),4 (8 B(1),X(1)));
    END;
  END;
END MEMOUT;
/*
```

3.7.3 MEMOUT Variable List

Registers (external variables)

<u>Simulation Name</u>	<u>Description</u>
MEMSTOR	Memory storage - 128 double words.

Local variables:

Simulation Name

Description

BIT1	Storage for flag 1
BIT2	Storage for flag 2
BIT3	Storage for flag 3
BIT4	Storage for flag 4
BYTE1	Storage for byte 1
BYTE2	Storage for byte 2
BYTE3	Storage for byte 3
BYTE4	Storage for byte 4

4. Simulating Routines

The procedures in this section simulate the low level logic of the TP.

Each procedure is a logical or conceptual section of the hardware. Complete descriptions of the hardware logic of these routines are found in the TP Manual (1).

4.1 FFSET Procedure

4.1.1 FFSET Description

The FASET procedure is a catchall for many of the flip-flops of the Taxicrinic Processor. Where these are to be set in the simulator is a problem in many cases. Therefore it was decided that setting or resetting flip-flops was best done before most of the other procedures of the TP simulator, since these procedures test the flip-flops. Hence, many are grouped in this procedure, which is one of the first to be called from SIMCALL.

The procedure is completely straightforward. If the set signal is on for the flip-flop, the flip-flop is set to 1. If the re-set signal is on, the flip-flop is set to 0.

4.1.2 FFSET Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
  FFSET: PROCEDURE
    (CSA,      DECPR,    IMM,      MSTORE,    NEG,      NWTOS,
     OPANDF,    OSC,      PER,      POP,      PROMOD,   RCSAS,
     RDCPRS,    RIMMS,    RMSRS,    RNEGS,    RNWTOSS,  ROPANS,
     ROSCS,     RPERS,    RPOPS,    RPROMS,   RRSCNS,   RSCN,
     RWRPS,     SCSAS,    SDCPRS,    SIMMS,    SMSRS,    SNEGS,
     SNWTOSS,   SOPANS,   SOSCS,    SPERS,    SPOPS,    SPROMS,
     SRSCNS,    SWRPS,    WRAP);
  DECLARE
    (CSA,      DECPR,    IMM,      MSTORE,    NEG,      NWTOS,
     OPANDF,    OSC,      PER,      POP,      PROMOD,   RCSAS,
     RDCPRS,    RIMMS,    RMSRS,    RNEGS,    RNWTOSS,  ROPANS,
     ROSCS,     RPERS,    RPOPS,    RPROMS,   RRSCNS,   RSCN,
     RWRPS,     SCSAS,    SDCPRS,    SIMMS,    SMSRS,    SNEGS,
     SNWTOSS,   SOPANS,   SOSCS,    SPERS,    SPOPS,    SPROMS,
     SRSCNS,    SWRPS,    WRAP) BIT(1);
  IF SCSAS THEN CSA='1'B;
  ELSE IF RCSAS THEN CSA='0'B;
  IF SDCPRS THEN DECPR='1'B;
  ELSE IF RDCPRS THEN DECPR='0'B;
  IF SIMMS THEN IMM='1'B;
  ELSE IF RIMMS THEN IMM='0'B;
  IF SMSRS THEN MSTORE='1'B;
  ELSE IF RMSRS THEN MSTORE='0'B;
  IF SNEGS THEN NEG='1'B;
  ELSE IF RNEGS THEN NEG='0'B;
  IF SNWTOSS THEN NWTOS='1'B;
  ELSE IF RNWTOSS THEN NWTOS='0'B;
  IF SOPANS THEN OPANDF='1'B;
  ELSE IF ROPANS THEN OPANDF='0'B;
  IF SOSCS THEN OSC='1'B;
  ELSE IF ROSCS THEN OSC='0'B;
  IF SPERS THEN PER='1'B;
  ELSE IF RPERS THEN PER='0'B;
  IF SPOPS THEN POP='1'B;
  ELSE IF RPOPS THEN POP='0'B;
  IF SPROMS THEN PROMOD='1'B;
  ELSE IF RPROMS THEN PROMOD='0'B;
  IF SRSCNS THEN RSCN='1'B;
  ELSE IF RRSCNS THEN RSCN='0'B;
  END. FFSET;
```

/*

4.1.3 FFSET Variable List

1 bit control signals (parameters):

<u>Simulation Name</u>	<u>Description</u>
CSA	Control FF - Conditional subtract was attempted
DECPR	Control FF - Decrement Pointers
IMM	Control FF - Instruction uses immediate option
MSTOR	Control FF - Memory Stores/Read
NEG	Control FF - for ØS Read
NWTØS	Control FF - for ØS Read
ØPANDF	Control FF - Operand stack conditional subtract failure
ØSBER	Control FF - Operand Stack Bounds Error
ØSC	Control FF - Permute names in IMP instr.
PØP	Control FF - for ØS entry sequence
PRØMØD	Control FF - equals 1 if PRO is modified
RSCSAS	Reset CSA FF
RDCPRS	Reset DECPR FF
RIMMS	Reset IMM FF
RMSRS	Reset MSTØR FF
RNEGS	Reset NEG FF
RNWTØSS	Reset NWTØS FF
RØPANS	Reset ØPANDF FF
RØSBES	Reset ØSBER FF
RØSCS	Reset ØSC FF
RPERS	Reset PER FF
RPØPS	Reset PØP FF
RPRØMS	Reset PRØMØD FF
RRSCNS	Reset RSCN FF
RSCN	Control FF - IMP instruction rescan cycle
RWRFS	Reset WRAP FF
SCSAS	Set CSA FF
SDCPRS	Set DECPR FF
SIMMS	Set IMM FF
SMSRS	Set MSTØR FF
SNEGS	Set NEG FF
SNWTØSS	Set NWTØS FF
SØPANS	Set ØPANDF FF

<u>Simulation Name</u>	<u>Description</u>
SØSBES	Set ØSBER FF
SØSCS	Set ØSC FF
SPERS	Set PER FF
SPØPS	Set PØP FF
SPRØMS	Set PRØMØD FF
SRSCNS	Set RSCN FF
SWRPS	Set WRAP FF
WRAP	Control FF for operand stack

4.2 CSGEN Procedure

4.2.1 CSGEN Description

The CSGEN procedure simulates the hardware logic used to determine the cell size signal for the various logic and control groups. This operation is explained in detail in Section 2.10 of the TP Manual.

Initially, in the section labelled "SET-SELECTOR", the cell size gate selector flip-flop, (GSFF) is set to one of four possible states depending upon whether the cell size will be determined by the field designator bits when interpreted as a cell size or immediate address field selector (SCSS = 1), by the field designator bits when interpreted as a number type (SNCSE = 1), by the contents of the control cell size flip-flops (SCSFG = 1), or by the four individual cell size control signals (SCSCS = 1). If none of the four possible selector signals are on, GSFF will maintain its previous state.

Then, in the section labelled "DECODE-SELECTOR", the state of the cell size gate selector is used to determine which dependent control signals are to be turned on (CSSE, CSFG, NCSE, or none.)

The control signals CSBFE, CSHFE, CSWFE, and CSDFE are used to set the cell size control flip-flop CSCFF in the section labelled "SET-CSCFF".

Finally, in the section labelled "SET-CELL-SIZES", the dependent control signals and flip-flops are used to determine whether the cell size is to be a byte (CSB), halfword (CSH), a full word (CSW), or a double word (CSD). If CSB or CSH is on, the CSBH signal (byte or halfword) is also turned on.

If the field designator bits are to be decoded as a number type, then one of the signals FPT, H/L, SFX, or LFX (floating point, hollerith, short fixed, or long fixed), are turned on.

These signals are then used throughout the simulator by other procedures.

4.2.2 CSGEN Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
  CSGEN: PROC(  CCSBE,    CCSDE,    CCSHE,    CCSWE,    CSB,
                CSBFE,    CSBH,     CSD,      CSDFE,    CSFG,
                CSH,     CSHFE,    CSSE,     CSW,      CSWFE,
                FPT,     HOL,      IMM,      LFX,      NCSE,
                SCSCS,    SCSFS,    SCSSS,    SFX,      SNCSS);
  DCL(CCSBE,CCSDE,    CCSHE,    CCSWE,    CSB,      CSBFE,
       CSBH,     CSD,    CSDFE,    CSFG,    CSH,
       CSHFE,    CSSE,    CSW,     CSWFE,    FPT,
       HOL,      IMM,     LFX,     NCSE,    SCSCS,
       SCSFS,    SCSSS,    SFX,     SNCSS) BIT(1);
  DCL  IR(36) BIT(1) EXTERNAL;
  DCL  (GSFF,      /* GATE SELECTOR FLIP-FLOP OUTPUT */
       CSCFF)     /* CELL SIZE CONTROL FLIP-FLOP OUTPUT */
        (1:2) BIT(1) STATIC,
        (IMCSSE,  /* CELL SIZE DETERMINED BY IMM OPTION */
       NICSSE)    /* CELL SIZE DETERMINED WITH IMM OPTION OFF */
        BIT(1);

  /*RESET CELL SIZE SIGNALS TO ZERO*/
  CSB,CSBH,CSH,CSW,CSD = '0'B;
SET_SELECTOR:
  /* SET CELL SIZE GATE SELECTOR FLIP-FLOP STATES */

  /* SET GATE SELECTOR FOR CSSE */

  IF SCSSS THEN DO;
    GSFF(1)='0'B;
    GSFF(2)='0'B;
  END;

  /* SET GATE SELECTOR FOR NCSE */

  IF SNCSS THEN DO;
    GSFF(1)='0'B;
    GSFF(2)='1'B;
  END;

  /* SET GATE SELECTOR FOR CSFG */

  IF SCSFS THEN DO;
    GSFF(1)='1'B;
    GSFF(2)='0'B;
  END;

  /* SET GATE SELECTOR OFF FOR CONTROL SIGNAL SELECT */

  IF SCSCS THEN DO;
    GSFF(1)='1'B;
    GSFF(2)='1'B;
  END;
```

DECODE_SELECTOR:

/* CHECK IF FIELD DESIGNATOR BITS ARE TO BE DECODED AS A CELL
SIZE INDEPENDENT OF IMMEDIATE OPTION */

CSSE= $\neg$ GSFF(1)& $\neg$ GSFF(2);

NICSSE=CSSE& $\neg$ IMM;

IMCSSE=CSSE&IMM;

/* CHECK IF CONTROL FLIP-FLOPS ARE TO BE GATED TO CELL
GENERATOR */

CSFG=GSFF(1)& $\neg$ GSFF(2);

/*CHECK IF FIELD DESIGNATOR BITS ARE TO BE DECODED AS A
NUMBER TYPE */

NCSE= $\neg$ GSFF(1)&GSFF(2);

SET_CSCFF:

/*SET CELL SIZE CONTROL FIIP-FLOP STATES */

/* ENABLE BYTE OUTPUT LINE */

IF CSBFE THEN DO;

CSCFF(1)='0'B;

CSCFF(2)='0'B;

END;

/* ENABLE HALFWORD OUTPUT LINE */

IF CSHFE THEN DO;

CSCFF(1)='0'B;

CSCFF(2)='1'B;

END;

/* ENABLE WORD OUTPUT LINE */

IF CSWFE THEN DO;

CSCFF(1)='1'B;

CSCFF(2)='0'B;

END;

/* ENABLE DOUBLE WORD OUTPUT LINE */

IF CSDFE THEN DO;

CSCFF(1)='1'B;

CSCFF(2)='1'B;

END;

SET_CELL_SIZES:

/* SET CELL SIZES */

/* CHECK IF CELL SIZE IS TO BE DETERMINED INDEPENDENTLY
OF THE IMMEDIATE OPTION */

IF CSSE THEN DO;
 CSH= $\neg$ IR(7)&IR(8);
 CSBH= $\neg$ IR(7);
 CSW=IR(7)& $\neg$ IR(8);
END;

/* CHECK IF CELL SIZE TO BE DETERMINED BY THE IMMEDIATE
OPTION ON */

IF IMCSSE THEN DO;
 CSH= $\neg$ IR(7)& $\neg$ IR(8)|CSH;
 CSW= IR(7)& IR(8)|CSW;
END;

/* CHECK IF CELL SIZE IS DETERMINED BY IMMEDIATE OPTION OFF*/

IF NJCSSE THEN DO;
 CSB= $\neg$ IR(7)& $\neg$ IR(8)|CSB;
 CSD= IR(7)& IR(8)|CSD;
END;

/* CHECK IF CELL SIZE IS TO BE DETERMINED BY CELL SIZE CONTROL
FLIP-FLOPS */

IF CSFG THEN DO;
 CSB= $\neg$ CSCFF(1)& $\neg$ CSCFF(2)|CSB;
 CSH= $\neg$ CSCFF(1)& CSCFF(2)|CSH;
 CSBH= CSCFF(1)|CSBH;
 CSW= CSCFF(1)& $\neg$ CSCFF(2)|CSW;
 CSD= CSCFF(1)& CSCFF(2)|CSD;
END;

/* CHECK IF FIELD DESIGNATOR BITS ARE TO BE DECODED AS A
NUMBER TYPE */

IF NCSE THEN DO ;
 CSH= $\neg$ IR(7)&IR(8)|CSH;
 CSBH= $\neg$ IR(7)& $\neg$ IR(8)|CSBH;
 CSD= IR(7)|CSD;

 FPT= IR(7)& $\neg$ IR(8);
 HOL= IR(7)& IR(8);
 SFX= $\neg$ IR(7)& $\neg$ IR(8);
 LFX= $\neg$ IR(7)& IR(8);

END;
ELSE FPT,HOL,SFX,LFX='0'B;

/* CHECK IF CELL SIZE IS DETERMINED BY CONTROL */

CSB=CSB|CCSBE;
CSH=CSH|CCSHE;
CSBH=CSBH|CCSHE|CCSBE;
CSW=CSW|CCSWE;
CSD=CSD|CCSDE;

END CSGEN;

/*

4.2.3. CSGEN Variables List

Control Signals:

<u>Simulation Name</u>	<u>Description</u>
CCSBE	Enable byte output line
CCSDE	Enable double word output line
CCSHE	Enable halfword output line
CCSWE	Enable word output line
CSB	Cell size is byte
CSBFE	Set cell size control flip-flop to byte
CSBH	Cell size is byte or halfword
CSD	Cell size is double word
CSDFE	Set cell size control flip-flop to double word
CSFG	Gate control flip-flops to CS generator
CSH	Cell size is halfword
CSHFE	Set cell size control flip-flop to halfword
CSSE	Enable decoding of field designator bits as a cell size independent of immediate option.. Cell size is word.
CSWFE	Set cell size control flip-flop to word
FPT	Floating point
HØL	Hollerith
IMM	Control flip-flop indicates instruction uses immediate option
LFX	Long fixed point
NCSE	Enable decoding of field designator bits as a number type
SCSCS	Set CS gate selector off for control signal select.
SCSFS	Set CS gate selector for CSFG.
SCSSS	Set CS gate selector for CSSE
SFX	Short fixed point
SNCSS	Set CS gate selector for NCSE

Internal Variables:

<u>Simulation Name</u>	<u>Description</u>
GSFF	Gate selector flip-flop output, 2 bits.
CSCFF	Cell size control flip-flop, 2 bits
IMCSSE	Cell size determined by immediate option
NICSSE	Cell size determined with immediate option off

Registers:

<u>Simulation Name</u>	<u>Description</u>
IR (i)	Instruction-Register (36 bits) - bit i

4.3 CØNSGEN Procedure

4.3.1 CØNSGEN Description

The CØNSGEN procedure simulates the hardware logic of the TP's constant generator which generates a specific constant used as an input to the 5 bit adder for the Operand Stack logic.

The output from the constant generator is a 5 bit number K. This number is related to the number of bytes in the cell size which has been currently chosen and is determined by several control signals: the cell size signals (CSB, CSH, CSW, CSD); the multiply by 2 signal (X2), the subtract signal (SE), and the control signals for generating specific constants (ØSCG1E, ØSCG2E, ØSCG4E, ØSCG8E, and ØSCG16E). The magnitude of the desired output is first placed on an intermediate bus CGB. Then depending upon whether SE is on or off, this value, or the complemented value is gated to K. Note that in the hardware, CGB is actually the complement of what is calculated here.

If the number is to be negative, C5INJ is set to 1 to indicate subtraction in the 5 bit adder.

4.3.2 CONSGEN Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
  CONSGEN: PROC (C5INJ, CSB, CSD, CSH, CSW,
                OSCG16E, OSCG1E, OSCG2E, OSCG4E, OSCG8E,
                OSCGE, SE, X2);
    DCL(C5INJ,CSB, CSD, CSH, CSW, OSCG16E,
        OSCG1E, OSCG2E, OSCG4E, OSCG8E, OSCGE,
        SE, X2) BIT(1);
    DCL K(5) BIT(1) EXTERNAL;

    /* THE CONSTANT GENERATOR OUTPUT BUS (K) IS A 5 BIT NUMBER
    WHICH BECOMES ONE OF THE 8 INPUTS TO THE 5 BIT ADDER.
    INPUTS TO THE GENERATOR ARE THE CELL SIZE SIGNALS(CS),
    THE ADD-SUBTRACT SIGNAL (SE), THE MULTIPLY BY 2 SIGNAL (X2),
    THE CONTROL LINES FOR GENERATING SPECIFIC CONSTANTS (OSCG).
    CGB IS A 5 BIT INTERMEDIATE BUS WHICH IS GATED EITHER
    POSITIVELY OR NEGATIVELY TO K.
    T1 AND T2 ARE DEFINED FOR CONVENIENCE TO SIMPLIFY THE
    EQUATIONS. */

    DCL CGB(5) BIT (1), (T1,T2) BIT (1);

    T2=-OSCGE&X2;
    T1=-X2;

    CGB(1)=T2&CSD|OSCG16E;
    CGB(2)=T1&CSD|T2&CSW|OSCG8E;
    CGB(3)=T1&CSW|T2&CSH|OSCG4E;
    CGB(4)=T1&CSH|T2&CSB|OSCG2E;
    CGB(5)=T1&CSB|OSCG1E;
    /* IF NUMBER IS TO BE NEGATIVE, GATE THE CGB'S TO THE K(I)'S
    DIRECTLY, OTHERWISE, FOR POSITIVE NUMBER, GATE -CGB(I)*/
    IF -SE THEN DO;
      K=CGB;
      C5INJ='0'B;
    END;
    ELSE DO;
      K=-CGB;
      C5INJ='1'B;
    END;

  END CONSGEN;

/*
```

4.3.3 CØNSGEN: Variable List

1 bit control signals (Parameters):

<u>Simulation Name</u>	<u>Description</u>
C5INJ	Carry inject to the 5 bit adder
CSB	Cell size is byte
CSD	Cell size is double word
CSH	Cell size is halfword
CSW	Cell size is word
ØSCG16E	Generate 16 on constant bus
ØSCG1E	Generate 1 on constant bus
ØSCG2E	Generate 2 on constant bus
ØSCG4E	Generate 4 on constant bus
ØSCGE	Enable generation of a constant from control
SE	Subtract enable - if = 1, generate a negative constant
X2	If 1, multiply output of constant generator by 2.

Registers and local variables:

<u>Simulation Name</u>	<u>Description</u>
CGB (i)	Constant generator intermediate bus - bit i (local variable, 5 bits)
K(i)	Output from the constant generator - bit i Used as an input to the 5-bit adder. (External variable, 5 bits)
T1	Temporary variable ($=\bar{X}2$) used to simplify equations. (Local variable, 1 bit)
T2	Temporary variable ($= \overline{\emptyset SCGE} \& X2$) used to simplify equations. (1 bit)

4.4 ADDER5 Procedure

4.4.1 ADDER5 Description

The ADDER5 procedure is not a real simulation of the 5 bit adder of the TP at this time, but should be written as such in the future. The internal subprocedure ADD5SUB performs the actual addition by simply decoding the bits of the A and B inputs of the adder into their numerical values and adding them. If C5INJ = 1, the sum is incremented by 1. Then the sum is converted back into a bit string, and the bits are placed into the array T1R.

The other parts of ADDER5 merely perform gating functions. If the SCR and T2R are to be gated to each other it is done here, before the addition, since it was decided that this operation is usually desired before the addition. The A and B inputs to the adder are determined by checking the gating signals. Possible A inputs are ØSTR, SCR, and T2R.

If ADD5E is on, ADD5SUB is called and A and B are added together and the result is left in the T1R. Afterwards, any gating from the T1R to other registers is done.

The ADDER5 procedure is called from SIMCALL.

4.4.2 ADDER5 Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
  ADDER5: PROC( A5OV,      ADD5E,      C5INJ,      KBG,      OTAG,
                RSYS,      SCAG,      SCT2G,      T1R2G,      T1SCG,
                T1STG,      T2AG,      T2BG,      T2SCG);
  DCL(          A5OV,      ADD5E,      C5INJ,      KBG,      OTAG,
            RSYS,      SCAG,      SCT2G,      T1R2G,      T1SCG,
            T1STG,      T2AG,      T2BG,      T2SCG) BIT(1);
  DCL (OSTR(2), (K,SCR,T2R)(5), T1R(0:5)) BIT(1) EXTERNAL;
  DCL (A,B)(0:5) BIT(1);
  DCL I FIXED BIN;
  IF T2SCG THEN DO I=1 TO 5;
    SCR(I)=T2R(I);
  END;
  IF SCT2G THEN DO I=1 TO 5;
    T2R(I)=SCR(I);
  END;
  /* LOAD A AND B INPUTS */
  A,B='0'B;
  IF OTAG THEN BEGIN;
    /* GATE OSTR TO A */
    A(1)=OSTR(1);
    A(2)=OSTR(2);
    A(3),A(4),A(5)='0'B;
  END;
  IF SCAG THEN DO I = 1 TO 5;
    A(I)=SCR(I);
  END;

  IF T2AG THEN DO I=1 TO 5;
    A(I)=T2R(I);
  END;
  IF T2BG THEN DO I=1 TO 5;
    B(I)=T2R(I);
  END;
  IF KBG THEN DO I=1 TO 5;
    B(I)=K(I);
  END;
  /* PERFORM ADDITION */
  IF ADD5E THEN
    CALL ADD5SUB;
  /* CHECK FOR OVERFLOW */

  A5OV=ADD5E&(-C5INJ&T1R(0)|C5INJ&-T1R(0));
  /* CHECK FOR TRANSFERS INVOLVING T1R,T2R,SCR, OR THE OSTR */

  IF T1STG THEN DO I=1 TO 2;
    OSTR(I)=T1R(I);
  END;

  IF T1SCG THEN DO I =1 TO 5;
    SCR(I)=T1R(I);
  END;

  IF T1R2G THEN DO I=1 TO 5;
    T2R(I)=T1R(I);
```

```

        END;
    /* RESET SYSTEM */
    IF RSYS THEN
        A50V='0'B;

ADD5SUB: PROC;
    DCL (I,ONE) FIXED BIN, C BIT (6), X FIXED BIN(6,0);
    ONE=1;
    X=0;
    DO I = 0 TO 5;
        IF A(I) THEN X=X+2**(5-I);
        IF B(I) THEN X=X+2**(5-I);
    END;
    IF C5INJ THEN X=X+1;
    C=BIT(X,6);
    DO I =1 TO 6;
        T1R(I-1)=SUBSTR(C,I,ONE);
    END;
END ADD5SUB;
END ADDER5;

```

/*

4.4.3 ADDER5 Variable List

Counters and Registers (external variables):

<u>Simulation Name</u>	<u>Description</u>
K (t)	Output constant generator (5 bits) - bit i
ØSTR(i)	Operand Stack Top Register (2 bits) - bit i
SCR(i)	Stack Control Register (5 bits) - bit i
T1R(i)	Temporary Register One (6 bits) - bit i (Output of 5 bit adder)
T2R(i)	Temporary Register Two (5 bits) - bit i

Control Signals (parameters):

<u>Simulation Name</u>	<u>Description</u>
A5OV	5 bit adder overflow
ADD5E	Enable addition
C5INJ	Inject carry into 5 bit adder
KBG	Gate Constant Generator output to B input of 5 bit Adder
ØTAG	Gate ØSTR to A input of 5 bit adder
RSYS	Reset system
SCAG	Gate SCR to A input of 5 bit adder
SCT2G	Gate SCR to T2R
T1R2G	Gate T1R to T2R
T1SCG	Gate T1R to SCR
T1STG	Gate T1R to ØSTR
T2AG	Gate T2R to A input of 5 bit adder
T2BG	Gate T2R to B input of 5 bit adder
T2SCG	Gate T2R to SCR

Local Variables:

<u>Simulation Name</u>	<u>Description</u>
A (i)	A input to 5 bit adder (6 bits) - bit i
B (i)	B input to 5 bit adder (6 bits) - bit i
C	Bit String of length 6 used to convert sum into bit string
X	Used to hold subtotal in the addition

4.5 ØPSTKIN Procedure

4.5.1 ØPSTKIN Description

The ØPSTKIN procedure simulates the hardware logic of the Operand Stack's functions which provide input to the Permuter. Functions which provide output from the Permuter to the Operand Stack are simulated in the PER-OUT procedure.

The Operand Stack Constant Generator is simulated separately in the procedure "CØNSGEN". Also, the 5 bit Adder used by the Operand Stack is simulated separately in the procedure "ADDER5".

The Operand Stack Select signals, ØSS(I), and the first byte select from OS signals, BY(I), are first initialized to zero. If ØSSS is not on, then no byte is selected from the operand stack, so the entire selection process may be skipped.

The T2R (Temporary Register 2) is a five bit register which holds the address of the first byte to be selected. The contents of T2R are decoded to give a value (T2RVAL) from 0 to 31 which will be this address. BY (T2RVAL) is then set to a 1 to indicate the first byte to be selected. If this byte address is less than or equal to 28, then that byte and the next 3 are chosen for selection by setting the corresponding OSS(I) signals to 1. If the byte address is greater than 28, the bytes are selected from the top of the Stack, and then starting back at the bottom to select four bytes in all, turning on the corresponding OSS(I) bits.

Since information will be overwritten or destroyed if anything is written into OS storage beyond the top of the Operand Stack, the Operand Stack Register, ØSTR, is checked for the current double word address of the top of the OS in the TP's Storage. The value of the ØSTR is decoded (ØSTRVAL) and this byte address plus the 3 succeeding byte addresses are prevented from being written into by setting the corresponding OSS(I) signals to zero.

Next the Base Register-Operand Stack bus BRØS is loaded from the selected bytes of the Operand Stack. Since each byte in the ØS is hardwired to one byte in the BRØS, (to save wiring each byte of the ØS to all four bytes of the BRØS) some selection process is necessary

to choose which byte of the BRØS is to be loaded. The MØD function was used to determiné this, since this is how the logic actually performs. If byte (I) of the Operand Stack is to be loaded into the BRØS, it will go into byte (MØD(I,4)) of the BRØS. Since four consecutive bytes are always chosen, this causes no overwriting. The bits are transferred and then complemented if ØSPFG = 1. Next the EØ1, EØ2, EU1, and EU2 signals, which indicate the position of the SCR to the ØSTR are calculated. X(I) and Y(I) are set equal to SCR(I) and ØSTR(I) simply to make the equations more readable.

The overflow and underflow signals ØV1, ØV2, UF1, and UF2 are calculated using the above E signals, cell sizes, and SCR contents.

Finally, if CKØFE = 1 (check OS overflow enable), then the state of the ØSF (Operand Stack full flip-flop) is determined.

4.5.2 OPSTKIN Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
  OPSTKIN: PROC(CKOFE,      CSB,      CSD,      CSH,      CSW,
                DBOSG,      OSF,      OSPFG,      OSPTG,      OSSS,
                OV1,        OV2,      ROSF,      UF1,        UF2,
                ZSCR);
  DCL(CKOFE,CSB,      CSD,      CSH,      CSW,      DBOSG,
       OSF,      OSPFG,      OSPTG,      OSSS,      OV1,      OV2,
       ROSF,      UF1,      UF2,      ZSCR) BIT(1);
  DCL(  BROS(36),  BY(0:31),  OS(0:31,9),  OSS(0:31),  OSTR(2),
       SCR(5),  T1R(0:5),T2R(5)) BIT(1) EXTERNAL;

  DCL
    (X(5),
     Y(2)
    ) BIT(1),
    (I,J,L) FIXED BIN,
    (E01,
     E02,
     EU1,
     EU2
    ) BIT(1);
  DCL T2RVAL FIXED BIN;
  DCL OSTRVAL FIXED BIN;

  /* USE BYTE DECODER*/
  /* USE THE CONTENTS OF T2R TO SELECT THE CORRESPONDING
  BY(I) SIGNAL */

  OSS='0'B;
  BY='0'B;
  OSSS=DBOSG|OSPTG|OSPGF;
  IF -OSSS THEN GO TO CKSCRZ;

  T2RVAL=0;
  DO I=1 TO 5;
    IF T2R(I) THEN
      T2RVAL=T2RVAL+2*(5-I);
  END;
  BY(T2RVAL)='1'B;

  /* NEXT 3 BYTE SELECT.
  USE THE BY(I) SIGNALS TO SELECT AN OSS(I) SIGNAL AND UP TO
  3 SUCCESSIVE OSS(I) SIGNALS */

  IF T2RVAL<=28 THEN
    DO I=T2RVAL TO (T2RVAL+3);
      OSS(I)='1'B;
    END;

  ELSE DO;
    DO I=T2RVAL TO 31;
      OSS(I)='1'B;
    END;
```

```

        DO I=0 TO (T2RVAL-28);
            OSS(I)='1'B;
        END;
    END;

    OSTRVAL=0;
    IF OSTR(1) THEN OSTRVAL=OSTRVAL +16;
    IF OSTR(2) THEN OSTRVAL=OSTRVAL +8;
    IF OSTRVAL=0 THEN DO;
        IF BY(29)|BY(30)|BY(31) THEN OSS(0),OSS(1),OSS(2)='0'B;
    END;
    ELSE DO;
        I=OSTRVAL;
        IF BY(I-1)|BY(I-2)|BY(I-3)
            THEN OSS(I),OSS(I+1),OSS(I+2)='0'B;
    END;

    /* LOAD BROS FROM OS */

    IF (OSPTG|OSPFG)&~DBOSG THEN DO;

        DO I=0 TO 31;
            IF OSS(I) THEN DO;
                L=MOD(I,4);
                /* L IS THE BYTE POSITION OF THE BROS TO WHICH THE
                   SELECTED BYTE OF THE OS IS GATED */
                DO J=1 TO 9;
                    BROS(L*9+J)=OS(I,J);
                END;
            END;
        END;

        IF OSPFG THEN BROS =~BROS;
    END;

    /*CHECK FOR SCR ON DOUBLE WORD BOUNDARY, I.E. LOW ORDER 3
       BITS =0 */

    CKSCRZ:  ZSCR=~SCR(3)&~SCR(4)&~SCR(5);

    /* CHECK FOR OVERFLOW*/

    IF ROSF THEN OSF ='0'B;

    DO I=1 TO 5;
        X(I)=SCR(I);
    END;

    DO I=1 TO 2;
        Y(I)=OSTR(I);
    END;

    EO1= ~X(1)&~X(2)&~Y(1)&Y(2)|~X(1)&X(2)&Y(1)&~Y(2)
        |X(1)&~X(2)&Y(1)&Y(2)|X(1)&X(2)&~Y(1)&~Y(2);

```

```

EO2= X(1)&¬X(2)&¬Y(1)&¬Y(2)| X(1)& X(2)&¬Y(1)& Y(2)
      |¬X(1)&¬X(2)& Y(1)&¬Y(2)|¬X(1)& X(2)& Y(1)& Y(2);

EU1= ¬X(1)&¬X(2)&¬Y(1)&¬Y(2)|¬X(1)& X(2)&¬Y(1)& Y(2)
      | X(1)&¬X(2)& Y(1)&¬Y(2)| X(1)& X(2)& Y(1)& Y(2);

EU2= ¬X(1)&¬X(2)& Y(1)& Y(2)|¬X(1)& X(2)&¬Y(1)&¬Y(2)
      | X(1)&¬X(2)&¬Y(1)& Y(2)| X(1)& X(2)& Y(1)&¬Y(2);

OV1= OSF |EO1&CSD&(X(3)|X(4)|X(5))
      |EO1&CSW&X(3)&(X(4)|X(5))
      |EO1&CSH&X(3)& X(4)& X(5);

OV2= OV1 |EO2&CSD&(X(3)|X(4)|X(5))
      |EO1&CSD&¬X(3)&¬X(4)&¬X(5)
      |EO1&CSW&(X(3)|X(4)|X(5))
      |EO1&CSH&X(3)&(X(4)|X(5))
      |EO1&CSB&X(3)&X(4)&X(5);

UF1= EU1&¬OSF&CSD
      |EU1&¬OSF&CSW&¬X(3)
      |EU1&¬OSF&CSH&¬X(3)&¬X(4)
      |EU1&¬OSF&CSB&¬X(3)&¬X(4)&¬X(5);

UF2= UF1
      |EU2&CSD
      |EU1&¬OSF&CSW
      |EU1&¬OSF&CSH&¬X(3)
      |EU1&¬OSF&CSB&¬X(3)&¬X(4);

IF CKOFE THEN
  IF OV2 THEN
    IF OSTR(1)=T1R(1) THEN
      IF OSTR(2)=T1R(2) THEN DO;
        OSF='1'B;
        OV1='1'B;
      END;
    END;
  END OPSTKIN;

```

/*

4.5.3 OPSTKIN Variable List

1 Bit Control Signals:

<u>Simulation Name</u>	<u>Description</u>
CKØFE	Enable ØS over-underflow check
CSB	Cell size is byte
CSD	Cell size is double word
CSH	Cell size is halfword
CSW	Cell size is word
DDØØG	Gate DDB to ØØ Storage
ØSF	Operand Stack full flip-flop
ØSPFG	Gate false out of ØS to Per.
ØSPTG	Gate true out of ØS to Per.
ØSSS	Operand Stack Storage Select
ØV1	ØS overflow-no room for 1 cell
ØV2	ØS overflow-no room for 2 cells
RØSF	Reset ØSF flip-flop
UF1	ØS underflow-less than 1 cell left
UF2	ØS underflow-less than 2 cells left
ZSCR	Low Order 3 bits of SCR = 0

Registers & Busses:

<u>Simulation Name</u>	<u>Description</u>
BRØS(i)	(36 bits) BR, ØS, IBR output bus
BY(i)	(32 bits) Decoder output for T2R. (if the Value of the T2R = I, BY(I) is set to 1)
ØS(i,j)	(32 bytes, 9 bits each) Operand Stack byte (i) bit (j)
ØSS(i)	(32 bits) ØS select; if on indicates that ith byte of ØS has been selected
ØSTR	(2 bits) Operand Stack top register indicates which double word is at top of stack
SCR	(5 bits) Stack Control register-gives current entry point to ØS
T1R	(6 bits) Temporary Register #1 - output from 5-bit Adder.
T2R	(5 bits) Temporary Register #2 - output from T1R or SCR - bit i

Local Variables:

<u>Simulation Name</u>	<u>Description</u>
EØ1	= 1 if top of OS is in the doubleword immediately preceding ØSTR. (1 bit)
EØ2	= 1 if the top of the ØS is in the second doubleword immediately preceding the ØSTR (1 bit)
EU1	= 1 if top of ØS is in the same double word as ØSTR boundary.
EU2	= 1 if top of ØS is in the second double word above the ØSTR boundary.
ØSTRVAL	Fixed binary variable - set equal to the decoded value of the ØSTR.
T2RVAL	Fixed binary variable - set equal to the decoded value of the T2R.
X(i)	5 bits - set equal to the bits of the SCR to make ØV/UF equations more readable
Y(i)	2 bits - set equal to the ØSTR to make the ØV/UF equations more readable.

4.6 BEGIN Procedure

4.6.1 BEGIN Description

The BEGIN procedure simulates the hardware logic of the actual base register operations in the TP.

Initially, the associative logic dependent control signals are set for either reading or replacing a base register.

If SNSG is on, then the contents of the PRS bus is gated into the Segment Name Select register SNSRG. Then if SNRDE or SNWTE is on the appropriate selection signals corresponding to the PR Segment Name Register indicated by the SNSRG are turned on. The 15th bits of these signals are controlled by the corresponding control signals PSNRR15 and PSNW15.

Next, the contents of the DRB is written into the Pointer Register Segment Name Register (PRSNR) selected by the PSNWS1 and S2 signals.

Now the pointer Register Segment Name Bus (PRSNB) is loaded. The PRSNB is set equal to the selected Pointer Register Segment Name Register.

As described in Section 2.5.1.2 of the TP Manual, the queue counters are used to keep track of which base registers have not been used for the longest length of time, by giving them numbers from 0 to 6 indicating the most and least recently used base registers respectively.

BRSREG is the BR queue counter select register. BRSSG gates the contents of BRRS into BRSREG.

A cycle counter is used to select each queue counter in turn to be reset at setup time. After the queue counters are set at values

from 0 to 6, their selection is made by a process using the value in the BRSREG (BRQS = BRSREG). CYCTE causes the cycle counter to be incremented, and RCYCTS sets it to zero.

QCNTNTR (i,j) is the ith queue counter, jth bit. RQOVFS causes all the queue counter overflow bits to be reset. Each queue counter is incremented if either QCNTNTE is on, or if XQCNTNTE is on and the counter's overflow bit is off. If the ith queue counter = 6, then RBRS(I) (Select BR(I) for reloading) is set to 1 since this means that BR(I) was least recently used.

To load the associative registers the data bits of the DBB are gated into the selected associative register if ASRWE is on. Also the corresponding write enable signals ASRWE1(I) and ASRWE2(I) are turned on.

In a memory access, a pointer register will be selected, and one of the segment name registers will be gated out to the PRSNB bus. This is then associatively compared with all the associative registers.

CMA(I) is an internal signal which is set to 1 if the first quarter of the ASRG(I) (associative register 1) matches the first quarter of the PRSNB. Similarly CMB, CMC and CMD are set in the same way after comparing the remaining quarters.

Now these CM signals are used to select the desired base register if a read is requested (RDBRE = 1). If for a given value of I (1 to 7), OK(I), CMA(I), CMB(I), and CMD(I) are all on, (indicating that the associative register (I) contains valid data and matches the PRSNB) then BR(I,*) is selected. (BRRS(I) is set to 1).

In this same section of code, the contents of the DBP can be written into a base register if signals BRGS and WRBRE are on. The base register is selected by the RBRS logic above, if some RBRS(I) signal is on. Otherwise, the BR is selected by the BRRS(I) signals. The contents of the DBP is then written into selected base register.

Finally, the contents of the selected base register is written onto the BRØS bus provided BRGS is on and WRBRE is off.

4.6.2 BEGIN Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
  BEGIN: PROC(BRGS,      BRSSG,      CYC6,      CYCOV,      CYCTE,
              DA,        NYET,        PSNRR15,   PSNRW15,   QCNTE,
              QSETUP,    RCYCTS,      RDBRE,      RPBRE,      RQOVFS,
              SNRDE,     SNSG,        SNWTE,      SOTOV,      WRBRE,
              XQCNT,     BRS,         PSNPRBG);
  DCL(BRGS, BRSSG, CYC6, CYCOV, CYCTE, DA,
       NYET, PSNRR15, PSNRW15, QCNTE, QSETUP,
       RCYCTS, RDBRE, RPBRE, RQOVFS, SNRDE,
       SNSG, SNWTE, SOTOV, WRBRE, XQCNT)
       BIT(1);
  DCL PSNPRBG BIT(1);
  DCL ADD1 ENTRY( (4) BIT(1)) EXTERNAL;
  DCL PRB(1:36) BIT(1) EXTERNAL;
  DCL( ASRG(7,18), BR(0:7,36), BROS(36), CYCTR(4),
       QCOUNT(7,4), PRSNB(1:36), PRSNR(0:15,18),
       (PRS, PSNRS1, PSNRS2, PSNWS1, PSNWS2, SNSRG)(0:15),
       (BRCS, BROCS, BRKS, BRWS, CMA, CMB, CMC, CMD, OK, RBR)
       (0:7)) BIT(1) EXTERNAL;
  DCL BRS(0:7) BIT(1);
  DCL (PSNA, PSNB)(1:17) BIT(1),
       COUNTER(4) BIT(1),
       (X,I,J) FIXED BIN;
  DCL BRSREG(0:7) BIT(1) EXTERNAL;

  /* TURN ON ASSOCIATIVE LOGIC CONTROL SIGNALS */

  RPBRE=BRGS&WRBRE;
  RDBRE=BRGS&~WRBRE;
  BRSSG=RDBRE;

  /* GATE PR SELECT SIGNALS INTO SELECT REGISTER SNSRG */

  IF SNSG THEN SNSRG=PRS;
    IF SNRDE THEN DO I=0 TO 14;
      PSNRS1(I), PSNRS2(I)=SNSRG(I);
    END;
    IF SNWTE THEN DO I=0 TO 14;
      PSNWS1(I), PSNWS2(I)=SNSRG(I);
    END;
  PSNRS1(15), PSNRS2(15)=PSNRR15;
  PSNWS1(15), PSNWS2(15)=PSNRW15;

  PSNA, PSNB='0'B;

  DO J=1 TO 8;
    DO I=0 TO 7;
      PSNA(J)=PSNA(J)|(PSNRS1(I)&PRSNR(I,J));
    END;
    DO I=8 TO 15;
      PSNB(J)=PSNB(J)|(PSNRS1(I)&PRSNR(I,J));
    END;
  END;
END;
```

```

DO J=10 TO 17;
  DO I=0 TO 7;
    PSNA(J)=PSNA(J)|(PSNRS2(I)&PRSNR(I,J));
  END;
  DO I=8 TO 15;
    PSNB(J)=PSNB(J)|(PSNRS2(I)&PRSNR(I,J));
  END;
END;

/* PR SEGMENT NAME REG OUTPUT BUS */

DO I=1 TO 17;
  PRSNB(I)=PSNA(I)|PSNB(I);
END;
IF PSNPRBG THEN PRB=PRSNB;

/* COMPARE THE 4 QUARTERS OF EACH ASSOCIATIVE REGISTER WITH
THE RESPECTIVE QUARTERS OF PRSNB. IF THE QUARTERS MATCH, SET
THE RESPECTIVE CM SIGNALS TO 1 */

CMA,CMB,CMC,CMD='1'B;
DO I=1 TO 7;
  DO J=1 TO 4;
    IF PRSNB(J)=ASRG(I,J) THEN CMA(I)='0'B;
  END;
  DO J=5 TO 8;
    IF PRSNB(J)=ASRG(I,J) THEN CMB(I)='0'B;
  END;
  DO J=10 TO 13;
    IF PRSNB(J)=ASRG(I,J) THEN CMC(I)='0'B;
  END;
  DO J=14 TO 17;
    IF PRSNB(J)=ASRG(I,J) THEN CMD(I)='0'B;
  END;
END;

/* BASE REGISTER SELECT SIGNAL DRIVERS */
BRRS='0'B;
BRWS='0'B;
IF RPBRE THEN DO I = 1 TO 7;
  BRWS(I)=RBRIS(I);
END;
IF RDBRE THEN DO I=1 TO 7;
  BRRS(I)=OK(I)&CMA(I)&CMB(I)&CMC(I)&CMD(I);
END;
/*CALCULATE BRCS WHEN IMPLEMENTED */
BRS='0'B;
IF BRGS THEN DO I=1 TO 7;
  BRS(I)=BRRS(I)|BRWS(I);
END;
/* BRSREG IS THE REGISTER ON DRAWING 23-3 */

IF BRSSG THEN BRSREG=BRS;

```

/* CYCTR IS THE 4 BIT COUNTER ON DRAWING 23-3 */

IF CYCTR THEN CALL ADD1(CYCTR);
IF RCYCTS THEN CYCTR='0'B;
CYCOV='0'B;
IF CYCTR(1) THEN CYCOV='1'B;

/* DECODE VALUE IN CYCTR */

X=0;
IF CYCTR(2) THEN X=X+4;
IF CYCTR(3) THEN X=X+2;
IF CYCTR(4) THEN X=X+1;

CYC6='0'B;
IF X=6 THEN CYC6='1'B;

IF QSETUP THEN BRQS(X+1)='1'B;
ELSE DO I=1 TO 7;
 BRQS(I)=BRSREG(I);
END;

/* QCOUNT(I,J) IS THE ITH COUNTER JTH BIT OF THE INTERNAL
COUNTERS ON DRAWINGS 23-1,2 */

IF RQOVFS THEN DO I=1 TO 7;
 QCOUNT(I,1)=~QCOUNT(I,1);
END;

/* INCREMENT COUNTERS AND CALCULATE RBRIS(I) SIGNALS */

DO I=1 TO 7;
 IF (QCNTE|(XQCNTE&~QCOUNT(I,1))) THEN DO;
 COUNTER=QCOUNT(I,*);
 CALL ADD1(COUNTER);
 QCOUNT(I,*)=COUNTER;
 END;
 RBRIS(I)=~QCOUNT(I,4)&QCOUNT(I,3)&QCOUNT(I,2);
END;

/* CHECK IF COUNTER SELECTED BY BRS(I) HAS OVERFLOWED */

SQTOV='0'B;
DO I=1 TO 7;
 SQTOV=SQTOV|(BRQS(I) & QCOUNT(I,1));
END;

DA='0'B;
DO I=1 TO 7;
 DA=DA|BRRIS(I);
END;
NYET=~DA;
IF BRGS&~WRBRE THEN DO;


```
BROS='0'B;  
DO I=0 TO 7;  
    IF BRS(I) THEN BROS(*)=BROS(*)|BR(I,*);
```

```
END;  
END;
```

```
END BREGIN;
```

```
/*
```

4.6.3 BEGIN Variable List

Parameters (1 or more bit signals):

<u>Simulation Name</u>	<u>Description</u>
BRGS	Select BR portion of BR-IBR storage
BRS(i)	Select BR(i)
BRSSG	Gate BRNS sigs into BR select storage
CYC6	Cycle counter = 6
CYCØV	Cycle counter overflow
CYCTE	Enable count on cycle counter
DA	A BR was selected by associative logic
NYET	No BR was selected by associative logic
PSNPRBG	Gate PRSNB to PRB bus into permuter
PSNRR15	PR sig Name Reg 15 read
PSNRW15	PR sig Name Reg 15 write
QCNTTE	Enable count on all Queue Counters
QSETUP	Queue counters are being initialized
RCYCTS	Reset cycle counter
RDBRE	Read BR selected by assoc. logic
RFBRE	Replace BR selected by assoc. logic
SNRDE	Enable PR segment Name Register-Read
SNSG	Gate PR select signals into select register
SNWTE	Enable PR segment Name Write
SQTØV	Q counter selected by BRQS has overflowed
WRBRE	Read/write BR - write when 1
XQCNTTE	Count once on all queue counters not overflowed

External Variables (Registers & Busses):

<u>Simulation Name</u>	<u>Description</u>
ASRG(i,j)	Associative Register i - bit j
BR(i,j)	Base Register i - bit j
BRØS(j)	BR, ØS, IBR output bus - bit j
BRQS(i)	BR(i) select for queue counters
BRRS(i)	BR(i) Readout select by assoc. logic
BRS(i)	Select BR(i)
BRSREG(i)	Base Register Select Register
BRWS(i)	BR(i) Write Select
CMA(i)	First quarter of BR(i) matches PRSNB
CMB(i)	Second quarter of BR(i) matches PRSNB
CMC(i)	Third quarter of BR(i) matches PRSNB
CMD(i)	Fourth quarter of BR(i) matches PRSNB
CYCTR(j)	Cycle counter - bit j
ØK(i)	Data in Assoc. Reg. for BRi is valid
PRB(j)	Pointer Reg. bus to permuter - bit j
PRS(i)	Select PRi
PRSNB(j)	PR seg. name output bus - bit j
PRSNR(i,j)	Pointer Register Segment Name Register (i, bit j)
PSNRS1(i)	PR seg name reg i - read select 1
PSNRS2(i)	PR seg name reg i - read select 1
PSNWS1(i)	PR seg name reg i - write select 1
PSNWS2(i)	PR seg name reg i - write select 2
QCØUNT(i,j)	Queue Counter i - bit j
RBRs(i)	Select BRi for replacement
SNSRG(i)	Segment name select register - bit i

4.7 PREGIN Procedure

The PREGIN procedure simulates the input functions of the pointer stack (PREGIN is mnemonic for "pointer registers input"). The ADD1 procedure is used to increment the ACT and CT counters. Since ADD1 is also needed by main control, it is not written inside the PREGIN procedure but instead is written as an external procedure. However, it is still basically a part of the PREGIN control since the main functions of ACT and CT have to do with renaming the pointer registers.

4.7.1 PREGIN Description

The PREGIN procedure controls the operations of the pointer registers. Internal to this procedure is the name bus (NB) and shadow name bus (SNB) which are 4 bits in length.

Initially, two counters (ACT, CCT) are set up. They can be zeroed or incremented by one. If the counters are equal, CEQA is turned on.

After the NB and SNB are initially cleared, the TGR (tag register) is loaded with the first four bits of byte i of the IR if IRTGG(i) = '1'. TGIM (tag register uses immediate option) is set to 1 or 0 using the IMM signal and the flag of the IR byte selected by the IRTGGli) signal.

The NB and SNB are loaded with the contents of the TGR, ACT, or CCT depending on the states of the gating signals.

If NPRI3S or NPRI4S is on the NB is forced to 13 or 14, respectively. In effect, this will force selection of PR13 or PR14.

If the TGR=13, N13S is set to '1'.

Next, the NB is compared with each name register (NR(i)) and if a match occurs between name register i and the NB, then an internal signal (NCE(i)) representing that NR is flagged. If the SBR select signal (SBRs) is off the NCE signals are gated to the PR select signals (PRS); otherwise the PRS signals are zeroed.

In the following section, the NFR (name flag register) may be either zeroed or loaded from the NCE signals. NFONE (name flag = 1) is turned on, and then turned off if NFCKE is on and NCE and NFR are both flagged on the same bit.

The next section of code sets up the group select signals. The "groups" refer to the pointer registers. The first group consists of pointer registers 0 to 7; the second group consists of pointers 8 to 14 and the SBR. PR8GS is set on if the select signal of any register of the second group (PRS(i), SBRS) is flagged; otherwise PR7GS is turned on.

The simulator now loads PRB (the pointer register output bus) with the PR selected or the SBR. The BRØS (BR, ØS, IBR Output Bus) is then loaded. If INBIG is on, name registers 0 to 7 are consecutively loaded in the BRØS; name register 0 occupies the first four bits of the BRØS. If INB2G is on, name registers 8 to 14 are consecutively loaded in the BRØS starting with name register 8 and ending in zeroes in bits 33-36.

Finally, the SNR's can be zeroed or the SBR can be gated into the SNR selected by the NCE signals. The SNR indicated by the NFR can be gated into the NR also indicated by the NFR.

4.7.2 PREGIN Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
  PREGIN: PROC(
    ACTE,      ANBG,      ASBG,      CCTE,      CEQA,      CNBG,
    CTSBG,     IMM,       INB1G,     INB2G,     N13S,      NFCKE,
    NFOE,      NFRG,      NPR13S,    NPR14S,    RACT,      RCT,
    RNFR,      RSNR,      SBRG,      SBSRG,     SNNRG,     TGIM,
    TNBG,      TSBG,      IRTGG,     PSNPRBG);
    DCL(      ACTE,      ANBG,      ASBG,      CCTE,      CEQA,
    CNBG,      CTSBG,     IMM,       INB1G,     INB2G,     N13S,
    NFCKE,     NFOE,      NFRG,      NPR13S,    NPR14S,    RACT,
    RCT,       RNFR,      RSNR,      SBRG,      SBSRG,     SNNRG,
    TGIM,      TNBG,      TSBG) BIT(1), IRTGG(0:2) BIT(1);
    DCL PSNPRBG BIT(1);
    DCL ADD1 ENTRY((4)BIT(1)) EXTERNAL;
    DCL( (ACT,CCT,TGR)(4), NFR(0:14), (BROS,IR,PRB,SBR)(36),
        PRS(0:14), (NR,SNR)(0:14,4), PR(0:14,36)) BIT(1)
        EXTERNAL;
    DCL NCE(0:14) BIT(1),
        (NB,
         SNB
          )(4) BIT(1),
        (PR7GS,
         PR8GS
          ) BIT (1),
        (I,J)      FIXED BIN;

/* SET UP COUNTERS */
  IF RCT THEN CCT='0'B;
  IF RACT THEN ACT='0'B;
  IF ACTE THEN CALL ADD1(ACT);
  IF CCTE THEN CALL ADD1(CCT);
  DO I=1 TO 4;
    IF ACT(I)~=CCT(I) THEN GO TO SKIP1;
  END;
  CEQA='1'B;

/* LOAD NAME BUS AND/OR SHADOW NAME BUS */
  SKIP1:NB,SNB='0'B;
  IF IRTGG(0) THEN
    DO I=1 TO 4;
      TGR(I)=IR(I);
      TGIM=IR(8)&IMM;
    END;
  IF IRTGG(1) THEN
    DO I= 1 TO 4;
      TGR(I)=IR(I+9);
      TGIM=IR(17)&IMM;
    END;
  IF IRTGG(2) THEN
    DO I= 1 TO 4;
      TGR(I)=IR(I+18);
      TGIM=IR(26)&IMM;
    END;
  IF TNBG THEN NB=TGR;
  IF ANBG THEN NB=ACT;
```

```

IF CNBG THEN NB=CCT;
IF TSBG THEN SNB=TGR;
IF ASBG THEN SNB=ACT;
IF CTSBG THEN SNB=CCT;
IF NPR13S THEN DO;
    NB='1'B;
    NB(3)='0'B;
    END;
IF NPR14S THEN DO;
    DO I=1 TO 3;
        NB(I)='1'B;
    END;
    NB(4)='0'B;
END;
IF TGR(1)&TGR(2)&¬TGR(3)&TGR(4) THEN N13S='1'B;
    ELSE N13S='0'B;

/* COMPARE NAME BUS WITH EACH REGISTER */
DO I=0 TO 14;
    NCF(I)='0'B;
    DO J=1 TO 4;
        IF NR(I,J)¬=NB(J) THEN GO TO NOTEQ;
    END;
    NCE(I)='1'B;
NOTEQ:END;

/* SET UP POINTER SELECT SIGNALS */
IF SBRS|PSNPRBG THEN PRS='0'B;
    ELSE PRS=NCE;

/* CHECK FOR NAME FLAG RESETS */
IF RNFR THEN NFR='0'B;

/* CHECK FOR GATING IN NAME FLAG REGISTER */
IF NFRG THEN DO I=0 TO 14;
    IF NCE(I) THEN NFR(I)='1'B;
END;

/* CHECK NAME FLAG */
NFOE='1'B;
IF NFCKE THEN DO I=0 TO 14;
    IF NCE(I)&NFR(I) THEN NFOE='0'B;
END;

/* SET UP GROUP SELECT SIGNALS */
PR8GS='0'B;
DO I=8 TO 14;
    IF NCE(I) THEN PR8GS='1'B;
END;
IF SBRS THEN PR8GS='1'B;
PR7GS=¬(PR8GS);

/* GATE POINTER TO POINTER BUS AS OUTPUT */
IF PR7GS THEN DO I=0 TO 7;
    IF PRS(I) THEN PRB(*)=PR(I,*);
END;
IF PR8GS THEN DO;
    DO I=8 TO 14;

```



```

        IF PRS(I) THEN PRB(*)=PR(I,*);
        END;
        IF SBR5 THEN PRB=SBR;
        END;

/* INTERRUPT BUS */
    IF INB1G THEN DO I=0 TO 3;
        DO J=1 TO 4;
            BROS(I*9+J)=NR(2*I,J);
            BROS(I*9+J+4)=NR(2*I+1,J);
        END;
    END;
    IF INB2G THEN DO I=0 TO 2;
        DO J=1 TO 4;
            BROS(I*9+J)=NR(2*I+8,J);
            BROS(I*9+J+4)=NR(2*I+9,J);
        END;
        DO J=1 TO 4;
            BROS(27+J)=NR(14,J);
            BROS(31+J)='0'B;
        END;
    END;

/* CHECK FOR RESET OF SNR */
    IF RSNR THEN SNR='0'B;

/* CHECK FOR GATING(SNB TO SNR) */
    IF SBSRG THEN DO I=0 TO 14;
        IF NCE(I) THEN SNR(I,*)=SNB;
    END;

/* CHECK FOR SNR TO NR */
    IF SNNRG THEN DO I=0 TO 14;
        IF NFR(I) THEN NR(I,*)=SNR(I,*);
    END;
END PREGIN;
/*

```

4.7.3 PREGIN Variable List

Registers and counters (external variables):

<u>Simulation Name</u>	<u>Description</u>
ACT(i)	Auxiliary 4 bit counter, - bit i
BRØS(i)	Base Register - Operand Stack Bus - bit i
CCT(i)	4 bit counter - bit i
IR(i)	Instruction Register - bit i
NFR(i)	Name Flag Register - bit i
NR(i,j)	Name Register i - bit j
PR(i,j)	Pointer Register i - bit j
PRB(i)	Pointer Register Bus to Permuter - bit i
PRS(i)	Select Pointer Register i signal
SBR(i)	Spare Buffer Register - bit i
SNR(i,j)	Shadow Name Register i - bit j
TGR(i)	Tag Register - bit i - holds tag from IR

Control Signals (1 or more bit parameters):

<u>Simulation Name</u>	<u>Description</u>
ACTE	Enable count on ACT (ACT = ACT + 1)
ANBG	Gate Signal ACT to NB
ASBG	Gate Signal ACT to SNB
CCTE	Enable Count on CCT (CCT = CCT + 1)
CEQA	Output Value of CCT = value of ACT
CNBG	Gate signal, CCT to NB
CTSBG	Gate signal CCT to SNB
IMM	Control FF indicates instruction uses immediate option
INB1G	Interrupt Save - Store lower SNR's onto BRØS
INB2G	Interrupt Save - Store higher SNR's onto BRØS
IRTGG(i)	Gate signal - IR (byte i-bits 1 to 4) to TGR
N13S	Output - Set to 1 if TGR = 13
NFCKE	Name Flag check enable
NFØNE	Output - name flag of SNR indicated by NB = '1'
NFRG	Gate Name Flag Register
NPR13S	Load NB with +13 - Force selection of PR13
NPR14S	Force selection of PR14
RACT	Reset ACT to zero
RCT	Reset CCT to zero
RNFR	Reset all name flag registers to zero
RSNR	Reset all shadow name registers to zero
SBRS	Spare buffer register select
SBSRG	Gate signal, SNB to SNR indicated by NCE signals
SNNRG	Gate signal, SNR(i) to NR(i), i indicated by NFR.
TG1M	Tag register immediate bit
TNBG	Gate signal, TGR to NB
TSBG	Gate signal, TGR to SNB

Local variables:

<u>Simulation Name</u>	<u>Description</u>
NB(i)	Name bus - bit i
NCE(i)	Name compare equal for i th register
PR7GS	Select PR group 0-7
PR8GS	Select PR group 8-15 or SBR
SNB(i)	Shadow name bus, bit i

4.7.4 ADD1 Description

The ADD1 procedure increments one of the two 4-bit counters (ACT, CCT) by one. The only parameter specifies which counter is to be incremented.

The new value of each bit of the counter is the 'exclusive or' of the old value of that bit and the carry of the previous bit. The new carry of each bit is the 'and' of the carry of the previous bit and the new value of the previous bit. The carry of the rightmost bit is a one.

It should be noted that the bits of the ACT and CCT are numbered left to right, which accounts for the need to iterate "backwards" for the carries.

4.7.5 ADD1 Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
  ADD1: PROC(COUNTER);
/* ADD1 ADDS ONE TO ACT OR CCT.*/
  DCL CARRY(4) BIT(1), COUNTER(4) BIT(1), I FIXED BIN;
  CARRY(4)='1'B;
  DO I=4 TO 2 BY -1;
    CARRY(I-1)=CARRY(I)&COUNTER(I);
    COUNTER(I)= ~COUNTER(I)&CARRY(I) |~CARRY(I)&COUNTER(I);
  END;
  COUNTER(1)= ~COUNTER(1)&CARRY(1) |~CARRY(1)&COUNTER(1);
  END ADD1;
/*
```

4.7.6 ADD1 Variable List

<u>Simulation Name</u>	<u>Description</u>
CARRY	Array of 4, 1 bit strings. Used to hold the carry from the Ith stage of the counter to the I-1 stage.
COUNTER	Formal parameter. 4 bit counter represented by an array of 4, 1 bit strings.
I	Iteration variable over 4 stages.

4.8 RWBGEN Procedure

The RWBGEN procedure simulates the read/write byte generator of the TP. The read/write byte itself is the first byte of the AR. When an access to the memory is made, the read/write byte is checked. If it is all zeroes, a read is done. If it is not all zeroes, the positions of the 1's in the read/write byte indicate the bytes to be written into of the double word accessed from the memory. A complete description of the read/write byte generator, and its logic equations, is found in the TP manual (2), Section 4.2.2.6.

4.8.1 RWBGEN Description

Since this is the first place in which it is used, the ARN is set to zero here. Then the dependent signals, ARF33, ARF34, ARF35 are set equal to the corresponding bits of the AR, if the control signal ARARFG is on.

If the read/write byte is to be generated, these signals are decoded into their numerical equivalence for use in the equations determining the bits of the read/write byte. This is done simply to make the equations more readable. The equations for the read/write byte are derived directly from the logic.

Finally, if the read/write byte has been generated, the first byte of the ARN is gated to the AR.

4.8.2 RWBGEN Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
RWBGEN: PROC(CSB,CSH,CSW,CSD,RWBGE,ARF33,ARF34,ARF35,MSTOR,
            PGMPAC,ARPG);
        DCL(CSB,CSH,CSW,CSD,RWBGE,ARF33,ARF34,ARF35,MSTOR,
            PGMPAC) BIT(1);
        DCL(AR,ARN)(1:36) BIT(1) EXTERNAL;
        DCL(ZERO,ONE,TWO,THREE,FOUR,FIVE,SIX,SEVEN) BIT(1);
        ARN='0'B;
        IF RWBGE&MSTOR&~PGMPAC THEN DO;
        /* DECODE LAST 3 BITS OF ADDRESS */
        ZERO= ~ARF33&~ARF34&~ARF35;
        ONE= ~ARF33&~ARF34& ARF35;
        TWO= ~ARF33& ARF34&~ARF35;
        THREE=~ARF33& ARF34& ARF35;
        FOUR= ARF33&~ARF34&~ARF35;
        FIVE= ARF33&~ARF34& ARF35;
        SIX= ARF33& ARF34&~ARF35;
        SEVEN= ARF33& ARF34& ARF35;
        /* GENERATE READ/WRITE BYTE */
        ARN(1)=ARN(1)|ZERO;
        ARN(2)=ARN(2)|CSB&ONE|(CSH|CSW|CSD)&ZERO;
        ARN(3)=ARN(3)|TWO&(CSB|CSH)|ZERO&(CSW|CSD);
        ARN(4)=ARN(4)|CSB&THREE|CSH&TWO|(CSW|CSD)&ZERO;
        ARN(5)=ARN(5)|FOUR&(CSB|CSH|CSW)|CSD&ZERO;
        ARN(6)=ARN(6)|CSB&FIVE|(CSH|CSW)&FOUR|CSD&ZERO;
        ARN(7)=ARN(7)|(CSB|CSH)&SIX|CSW&FOUR|CSD&ZERO;
        ARN(8)=ARN(8)|CSB&SEVEN|CSH&SIX|CSW&FOUR|CSD&ZERO;
        /* THIS IS A FUDGE. THIS OPERATION IS ESSENTIALLY
        THE SAME AS WILL BE PERFORMED LATER BY PER-OUT
        HOWEVER IT IS NECESSARY TO DO IT HERE ALSO SO
        THAT THE DATA WILL BE IN THE AR WHEN PER-IN IS
        CALLED. THIS CASE ARISES WHEN RWBGE AND ARPG ARE
        ON IN THE SAME CONTROL POINT */
        IF ARPG THEN DO J=1 TO 9;
            AR(J)=ARN(J);
        END;
        END;
END RWBGEN;
/*
```

4.8.3 RWBGEN Variable List

1 bit control signals (parameters):

<u>Simulation Name</u>	<u>Description</u>
ARARFG	AR bits to ARF signals gate
ARF33	Storage for low order adder-bit 33
ARF34	Storage for low order Adder-bit 34
ARF35	Storage for low order Adder-bit 35
CSB	Cell size is byte
CSD	Cell size is double-word
CSH	Cell size is half-word
CSW	Cell size is word
MSTØR	Control FF - Memory store/read
PGMPAC	This is a page map access
RWBGE	Read/write byte generator enable

Registers and busses (external variables):

<u>Simulation Name</u>	<u>Description</u>
AR	Arithmetic register
ARN	Arithmetic register input bus

Local variables:

<u>Simulation Name</u>	<u>Description</u>
FIVE	Decoded value of ARF bits = 5
FØUR	Decoded value of ARF bits = 4
ØNE	Decoded value of ARF bits = 1
SEVEN	Decoded value of ARF bits = 7
SIX	Decoded value of ARF bits = 6
THREE	Decoded value of ARF bits = 3
TWØ	Decoded value of ARF bits = 2
ZERØ	Decoded value of ARF bits = 0

4.9 B00LSHFT Procedure

The B00LSHFT procedure simulates some of the Boolean/Shift Logic performed by the TP. The right and left shifts on a bit basis are done here, as are the Boolean functions AND and OR. The exclusive or, XOR, and equivalence, EQV, Boolean functions are done in PER-OUT, since they should be performed after the DB has settled down to its final state, and after the 32 bit adder has been called (the 32 bit adder does XOR and EQV).

The B00LSHFT procedure also calculates the permutation and inhibit signals used by the permuter. These signals are calculated from equations using the right and left shift enable signals (RSPE and LSPE) and the M counter. The M counter (which gives the number of bytes to be shifted) is also simulated here as an internal procedure.

4.9.1 B00LSHFT Description

Initially in the procedure, LRN (the input bus to the LR) is zeroed. This is the first section in which the LRN is used, and later inputs will be "OR"ed together before being gated into the LRN.

Right and left shifts may be done here, shifting only the data bits of the DR and returning the result to the LRN. The flag bits are gated separately (directly) without being shifted. If a left shift is done, LRN(35) is set to a "0".

The boolean AND is done here, by simply ANDing all the bits of the DR and DB together, and gating the result to LRN.

Next the control signals CTMHZ (high 2 bits of Mcounter = 0) and CTMLZ (low 3 bits of Mcounter = 0) are calculated, using the current contents of the Mcounter.

The high order 5 bits of the LR may be gated to the Mcounter for a new calculation. The Mcounter is used by TP control to keep track of the number of bits yet to be shifted.

The Mcounter may be reset by setting MCT = '0'. The decision to count up or to count down is determined by control signals CTUE and CTDE. Then, if desired, MCTR is called.

The internal procedure MCTR, at the end of B00LSHFT, is not a precise simulation of the actual Mcounter of the TP, but it performs the necessary functions of adding or subtracting 1. This is done in the following way. The value of MCT is found, and placed in M, a fixed binary variable, then 1 is added or subtracted from M, corresponding to a count up or down. If the count reaches -1, MCT is set to all 1's. Otherwise, the value of M is converted to a 5 bit string, and the bits of this string are loaded back into the MCT array.

After MCTR is called, the PL (permutation) and IB (inhibit) signals are calculated from the value of the MCT and the states of control signals RSPE (enable right shift using permuter) and LSPE (enable left shift using permuter). These equations are explained in the TP Manual (1), Section 2.8.

4.9.2 BOOLSHFT Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
  BOOLSHFT: PROC(
    ANDE,      CTDE,      CTMHZ,      CTMLZ,      CTUE,      DLRG,
    IBPL13,    IBPL21,    LRMCTG,    LSE,        LSPE,      MCTE,
    ORE,       RMCT,      RSE,       RSPE,
    IB,        PL);
  DCL ( ANDE,CTDE,      CTMHZ,      CTMLZ,      CTUE,      DLRG,
        IBPL13,    IBPL21,    LRMCTG,    LSE,        LSPE,
        MCTE,      ORE,       RMCT,      RSE,        RSPE) BIT(1),
        (IB, PL)(0:3) BIT(1);
  DCL (DB,DR,LR,LRN)(36) BIT(1) EXTERNAL;
  DCL (CTU,CTD) STATIC BIT(1), (CTLZ,CTHZ,CTZ)BIT(1),
        MCT(1:5) BIT(1) STATIC;
  LRN='0'B;
        /* DRAWING 07-1*/
  IF RSE THEN DO;
    DO I=0 TO 3;
      DO J=I*9+1 TO I*9+8;
        LRN(J+1)=LRN(J+1)|DR(J);
      END;
    END;
  DO I=9 TO 36 BY 9;
    LRN(I)=DR(I);
  END;
  LRN(1)='0'B;
  END;
  IF LSE THEN DO;
    DO I=0 TO 3;
      DO J=I*9+1 TO I*9+8;
        LRN(J-1)=LRN(J-1)|DR(J);
      END;
    END;
  DO I=9 TO 36 BY 9;
    LRN(I)=DR(I);
  END;
  LRN(35)='0'B;
  CALL PRINT1(LRN,'LRN ');
  END;
  IF ANDE THEN LRN(*)=DR(*)&DB(*);
  /* CALCULATE CONTROL INFORMATION  DRAWING 17-1  */
  CTMHZ=~MCT(5)&~MCT(4);
  CTMLZ=~MCT(3)&~MCT(2)&~MCT(1);
  /* LOAD M-COUNTER FROM THE LR  DRAWING 17-1  */
  IF LRMCTG THEN DO I=1 TO 5;
    MCT(I)=LR(I+30);
  END;
  /*RESET COUNTER,SELECT COUNT UP OR DOWN, AND COUNT */
  /* DRAWING 17-2 */

  IF RMCT THEN MCT='0'B;
  IF CTUE THEN DO;
    CTU='1'B;
    CTD='0'B;
  END;
```

```

IF CTDE THEN DO;
    CTD='1'B;
    CTU='0'B;
END;
IF MCTE THEN CALL MCTR;

/* DETERMINE BYTE SHIFTING FOR PERMUTER */
/* DRAWING 17-1*/

IBPL21=RSPE&MCT(1)&MCT(2);
IBPL13=LSPE&MCT(1)&MCT(2);
PL(0)=~MCT(1)&~MCT(2)&(LSPE|RSPE);
PL(1)=LSPE&~MCT(1)&MCT(2);
PL(2)=(MCT(1)&~MCT(2))&(LSPE|RSPE);
PL(3)=RSPE&~MCT(1)&MCT(2);
IB(0)=RSPE&(MCT(1)|MCT(2));
IB(1)=RSPE&MCT(1);
IB(2)=LSPE&MCT(1);
IB(3)=LSPE&(MCT(1)|MCT(2));

DLRG=DLRG|ORE;
MCTR:PROC;
    DCL (M,I) FIXED BIN (5,0), MS BIT(5),ONE FIXED BIN;
    PUT LIST('MCTR ENTERED');
    ONE=1;
    DO I=1 TO 5;
        IF MCT(I) THEN M=M+2** (5-I);
    END;
    IF CTU THEN M=M+1;
    IF CTD THEN M=M-1;
    IF M<0 THEN MCT='1'B;
    ELSE DO;
        MS=M;
        DO I=1 TO 5;
            MCT(I)=SUBSTR(MS,I,ONE);
        END;
    END;
END MCTR;

END BOOLSHFT;

```

/*

4.9.3 B00LSHFT Variable List

Registers and Counters (External Variables):

<u>Simulation Name</u>	<u>Description</u>
DB (i)	Distribution Bus (36 bits) - bit i
DR (i)	Distribution Register (36 bits) - bit i
LR (i)	Logic Register (36 bits) - bit i
LRN	LR input bus (36 bits) - bit i

Control Signals (1 or more bit parameters):

<u>Simulation Name</u>	<u>Description</u>
ANDE	Enable "AND" of DR and DB, gate to LR
CTDE	Set down-select signal for MCT
CTMHZ	High 2 bits of MCT = 0
CTMLZ	Low 3 bits of MCT = 0
CTUE	Set up select signal for MCT
DLRG	Gate DR to LR direct
IB(i)	Inhibit byte i of permuter
IBPL13	Inhibit byte 1, permute left 3
IBPL21	Inhibit byte 2, permute left 1
LRMCTG	Gate low 5 bits of LR to MCT
LSE	Enable left shift, DR to LR
LSPE	Enable left shift using Permuter
MCTE	Enable Mcounter to count
ØRE	DB V DR to LR
PL(i)	Permute left i bytes (from shift logic)
RMCT	Reset M counter to 00000.
RSE	Enable right shift, DR to LR
RSPE	Enable right shift using Permuter

Local Variables:

<u>Simulation Name</u>	<u>Description</u>
CTD	Static flip flop - count down
CTU	Static flip-flop - count up
M	Fixed binary - used to hold subtotals for M counter
MCT	The Mcounter - 5 bit array
MS	5 bit string used to load value of M into MCT Array

4.10 PER-IN Procedure

4.10.1 PER-IN Description

The PER-IN procedure simulates the functions which provide input to the Permuter of the TP. The logic corresponding to this procedure is described in section 2.1 of the TP Manual (1).

$\emptyset$ SPG is first set so that if $\emptyset$ SPTG or $\emptyset$ SPFG are on, it too will be on.

If the LRSE signal is on the PLiE signals are to be determined only by the PL(i) signals, IBPL13, and IBPL21. Therefore the next section of code is skipped and the PLiE signals are initially zeroed. LRSE should be set to 1 if any of the signals in its equation are on.

Internally the permuter has four signals (PLOE ... PL3E) which are used to indicate the length of the permutation (number of bytes permuted to the left). The length of the permutation is determined by the source (S) and destination (D) positions of the word to be permuted. S and D are internal arrays (one dimensional, length 2) which represent these positions in binary form. Initially S and D are set to zero. S and D can be loaded from various registers or exchanged with one another. Then one of the PLiE signals is set depending on the states of S and D.

The PLiE signals can also be set by the PLCi, PL(i), IBPL13, and IBPL21 signals.

Next an internal register (FBI) of the permuter is initially zeroed and then various registers of the TP can be gated into the FBI. If the BR $\emptyset$ S is gated in, the signal BRSPG is set to 1.

The $\emptyset$ SR is a 5-bit bus internal to the permuter which is used to insert the operand stack top register ($\emptyset$ STR), stack control register (SCR), or bits of the instruction counter (ICT) onto the permuter output (DBP). This is done by use of the inhibit signals (IBE(i)). Initially the bits to be inserted on the DBP are set to one in the FBI (masked). Then the $\emptyset$ STR, SCR, or ICT can be gated to the $\emptyset$ SR with zeroes inserted if necessary. Any zero in the $\emptyset$ SR sets an inhibit bit (IBE3(i)) which will later zero that bit in the DBP. The $\emptyset$ SR is initially set to ones so that none of the DBP bits will be inhibited if nothing is gated into the $\emptyset$ SR.

If a byte or halfword is to be permuted and then right or left justified the proper inhibit signals are set so that the non-significant part of the word is zeroed. If the cell size is a byte then CSB and CSBH should be turned on, while if the cell size is a halfword then CSH and CSBH should be turned on.

The internal inhibit signals of the permuter ($IBE(i)$, $IBE3(i)$) can also be turned on by the independent control signals ($IB(i)$ and $IBC(i)$).

Negative constants can be formed by inhibiting the proper bits in the rightmost byte. To form a negative constant, LRSE must be on in order that no permutation takes place from the FBI to the DBP. Hence the MiE signals are in the equation for LRSE at the beginning of the procedure.

Positive constants are formed simply by setting the appropriate FBI bit to one.

The DBP is initially set to 1's, and later the proper bits of the DBP are inhibited (zeroed). If LRSE is on, it is possible that none of the $PLiE$ are on.

Finally the FBI is permuted into the DBP. The $PLiE$ signals indicate the length in bytes of the permutation.

The inhibiting of any first three bytes of the DBP is done if the corresponding inhibit signal ($IBE(i)$) is on. Any bit of the fourth byte of the DBP is inhibited if the corresponding inhibit signal ($IBE3(i)$) is equal to 1.

The DBP is now in its final state. Finally, the DB and the DBB are set equal to the DBP.

4.10.2 PER_IN Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
PER_IN: PROC(ARPG, BNRG,BRGS, BRSBPG, CRRLG, CSB,
CSBH, CSH, CTPG, DBRJ, DRPG, IBPL13,
IBPL21, IBRS, ICTPG, IRPG, LJIE, LJOPE,
LRPG, LRSE, M16E, M1E, M2E, M32E,
M4E, M8E, NRSETG, OPJE, OSPG, OSTPG,
P16E, P1E, P2E, P32E, P4E, P8E,
PLC1, PLC2, PLC3, PRPG, RJIE, RSDE,
SCRPG, LRSE1, IB, IBC, PL, OSSS,
OSPTG, OSPFG);
DCL (OSPTG,OSPG) BIT(1);
DCL LRSE1 BIT(1);
DCL OSSS BIT(1);
DCL ( ARPG,BNRG, BRGS, BRSBPG, CRRLG, CSB,
CSBH, CSH, CTPG, DBRJ, DRPG, IBPL13,
IBPL21, IBRS, ICTPG, IRPG, LJIE, LJOPE,
LRPG, LRSE, M16E, M1E, M2E, M32E,
M4E, M8E, NRSETG, OPJE, OSPG, OSTPG,
P16E, P1E, P2E, P32E, P4E, P8E,
PLC1, PLC2, PLC3, PRPG, RJIE, RSDE,
SCRPG) BIT(1), (IB, IBC, PL) (0:3) BIT(1);
DCL ( (AR,BROS,CT,DB,DBB,DBP,DR,IR,LR,PRB) (36), (BNR,ICT) (3),
(ACT,CCT) (4), (SCR,T2R) (5), OSTR(2), BRS(0:7))
BIT(1) EXTERNAL;
DCL (PLIOE,PLI1E,PLI2E,PLI3E) BIT(1);
DCL FBI(36) BIT(1),IBE(0:2)BIT(1),IBE3(1:9)BIT(1),
(S,D)(0:1)BIT(1),OSR(1:5) BIT(1),
(PLOE,PL1E,PL2E,PL3E,BRSPG) BIT(1),
(I,J) FIXED BIN;
OSPG=OSPTG|OSPG;
/* DETERMINE LENGTH OF SHIFT FROM SOURCE AND DESTI-
NATION POSITIONS, S AND D. IF LRSE IS ON INHIBIT
NORMAL CONTROL.*/

LRSE=M1E|M2E|M4E|M8E|M16E|M32E|LRSE1|PLC1|PLC2|PLC3;
IF LRSE THEN DO;
PLIOE,PLI1E,PLI2E,PLI3E='0'B;
GO TO SKIPSD;
END;

S(0),S(1),D(0),D(1)='0'B;

IF BRGS THEN DO;
S(0)=BRS(1)|BRS(2)|BRS(5)|BRS(6);
S(1)=BRS(1)|BRS(3)|BRS(5)|BRS(7);
END;

IF DBRJ THEN DO;
D(0)=CSH;
D(1)=CSB;
END;

IF OSSS THEN DO;
S(0)=T2R(4);
S(1)=T2R(5);
END;
```



```

IF BNRG THEN DO;
    S(1)=BNR(3);
    S(0)=BNR(2)&(-BNR(3))
        |BNR(3)&(-BNR(2));
END;

IF CRRLG THEN DO;
    S(0)=AR(34);
    S(1)=AR(35);
END;

IF IBRS THEN DO;
    S(0)=ICT(2);
    S(1)=ICT(3);
    D(1)=OPJE;
END;

IF LJOPE THEN D(1)='0'B;
IF RSDE THEN
    BEGIN;
        DCL TEMP(0:1) BIT(1);
        TEMP=S;
        S=D;
        D=TEMP;
    END;

```

```

DO;
    PL10E= ~S(0)&~S(1)&~D(0)&~D(1)
        |~S(0)&S(1)&~D(0)&D(1)
        |S(0)&~S(1)&D(0)&~D(1)
        |S(0)&S(1)&D(0)&D(1);
    PL11E= ~S(0)&S(1)&~D(0)&~D(1)
        |S(0)&~S(1)&~D(0)&D(1)
        |S(0)&S(1)&D(0)&~D(1)
        |~S(0)&~S(1)&D(0)&D(1);
    PL12E= S(0)&~S(1)&~D(0)&~D(1)
        |S(0)&S(1)&~D(0)&D(1)
        |~S(0)&~S(1)&D(0)&~D(1)
        |~S(0)&S(1)&D(0)&D(1);
    PL13E= S(0)&S(1)&~D(0)&~D(1)
        |~S(0)&~S(1)&~D(0)&D(1)
        |~S(0)&S(1)&D(0)&~D(1)
        |S(0)&~S(1)&D(0)&D(1);
END;

```

```

SKIPSD: PLOE=PL10E|PL(0);
        PL1E=PL11E|PL(1)|IBPL21|PLC1;
        PL2E=PL12E|PL(2)|PLC2;
        PL3E=PL13E|PL(3)|IBPL13|PLC3;

```

```

/* SELECT PROPER INPUT TO BE GATED*/

```

```

FBI='0'B;
BRSPG='0'B;
IF DRPG THEN FBI=DR;

```

```

IF IRPG THEN FBI=IR;
IF LRPG THEN FBI=LR;
IF ARPG THEN FBI=AR;
IF PRPG THEN FBI=PRB;
IF CTPG THEN FBI=CT;
IF OSPTG|OSPFG|BRSPG THEN DO;
    BRSPG='1'B; FBI=BROS; END;

/* CHECK FOR MASKING */
IF OSTPG | SCRPG THEN DO I=31 TO 35;
    FBI(I)='1'B;
    END;
IF ICTPG THEN DO I=33 TO 35;
    FBI(I)='1'B;
    END;
IF NRSEIG THEN DO I=1 TO 4;
    FBI(I+27)=ACT(I);
    FBI(I+31)=CCT(I);
    END;
/* DETERMINE INPUTS TO INHIBIT ENABLE NETWORK FROM
OPERAND STACK.*/

OSR='1'B;
IF OSTPG THEN DO;
    OSR(1)=OSTR(1);
    OSR(2)=OSTR(2);
    OSR(3),OSR(4),OSR(5)='0'B;
    END;

IF SCRPG THEN OSR=SCR;

/* DETERMINE INPUTS TO INHIBIT ENABLE NETWORK FROM
INSTRUCTION BYTE COUNTER.*/

IF ICTPG THEN DO;
    OSR(1),OSR(2)='0'B;
    DO I=1 TO 3;
        OSR(I+2)=ICT(I);
    END;
    END;

/* CALCULATE INHIBITS.*/

IBE='0'B;
DO I=1 TO 3; IBE3(I)='0'B; END;
DO I=1 TO 5; IBE3(I+3)=~OSR(I); END;

IF RJIE THEN DO;
    IF CSBH THEN IBE(0)='1'B;
    IF CSH THEN IBE(1)='1'B;
    IF CSB THEN IBE(1),IBE(2)='1'B;
    END;
IF LJIE THEN DO;
    IF CSBH THEN IBE3='1'B;
    IF CSH THEN IBE(2)='1'B;

```

```
IF CSB THEN IBE(1),IBE(2)='1'B;
END;
```

```
IF IBPL13 THEN IBE(1)='1'B;
IF IBPL21 THEN IBE(2)='1'B;
IF IB(0)||IBC(0) THEN IBE(0)='1'B;
IF IB(1)||IBC(1) THEN IBE(1)='1'B;
IF IB(2)||IBC(2) THEN IBE(2)='1'B;
IF IB(3)||IBC(3) THEN IBE3='1'B;
```

```
/* CHECK FOR INHIBITS FROM CONSTANT GENERATOR.*/
```

```
IF M1E THEN IBE3(8)='1'B;
IF M2E THEN IBE3(7)='1'B;
IF M4E THEN IBE3(6)='1'B;
IF M8E THEN IBE3(5)='1'B;
IF M16E THEN IBE3(4)='1'B;
IF M32E THEN IBE3(3)='1'B;
```

```
IF P1E THEN FBI(35)='1'B;
IF P2E THEN FBI(34)='1'B;
IF P4E THEN FBI(33)='1'B;
IF P8E THEN FBI(32)='1'B;
IF P16E THEN FBI(31)='1'B;
IF P32E THEN FBI(30)='1'B;
DBP='1'B;
```

```
/* SELECT AND PERFORM SHIFTS.*/
```

```
IF PLOE THEN DBP=FBI;
IF PL1E THEN DO;
    DO I=1 TO 27;
        DBP(I)=FBI(I+9);
    END;
    DO I=1 TO 9;
        DBP(I+27)=FBI(I);
    END;
END;
IF PL2E THEN DO I=1 TO 18;
    DBP(I)=FBI(I+18);
    DBP(I+18)=FBI(I);
END;
IF PL3E THEN DO;
    DO I=1 TO 9;
        DBP(I)=FBI(I+27);
    END;
    DO I=1 TO 27;
        DBP(I+9)=FBI(I);
    END;
END;
```

```
/* CHECK IB INHIBITS.*/
```

```
IF IBE(0) THEN DO I=1 TO 9;
    DBP(I)='0'B;
```

```

        END;
    IF IBE(1) THEN DO I=1 TO 9;
        DBP(I+9)='0'B;
    END;
    IF IBE(2) THEN DO I=1 TO 9;
        DBP(I+18)='0'B;
    END;
    DO I=1 TO 9;
        IF IBE3(I) THEN DBP(I+27)='0'B;
    END;
    /* MAKE DB OUTPUTS EQUAL TO DBP OUTPUTS.*/

    DB,DBB=DBP;
    END PER_IN;
/*

```

4.10.3 PER_IN Variable List

Registers and counters (external variables):

<u>Simulation Name</u>	<u>Description</u>
ACT(i)	Auxiliary 4 bit counter - bit i
AR(i)	Arithmetic Register - bit i
BNR(i)	Base Name Register (3 bits) - bit i
BRØS(i)	BR,ØS,IBR output bus - bit i
BRS(i)	Select base register (i)
CCT(i)	4 bit counter - bit i
CT(i)	Cable terminators to exchange net - bit i
DB(i)	Distribution bus - bit i
DBB(i)	Control Section DB driver - bit i
DBP(i)	Permuter output bus - bit i
DR(i)	Distribution register - bit i
ICT(i)	Instruction byte Counter (3 bits) - bit i
IR(i)	Instruction Register - bit i
LR(i)	Logic Register - bit i
ØSTR(i)	Operand Stack Top register - bit i
PRB(i)	Pointer Registers Bus to permuter - bit i
SCR(i)	Stack Control Register (5 bits) - bit i
T2R(i)	Temporary Register 2 in Operand Stack (5 bits) - bit i

Control Signals (1 or more bit parameters):

<u>Simulation Name</u>	<u>Description</u>
ARPG	AR is permuter input
BNRG	Base name register gate
BRGS	Select BR portion of BR-IBR storage
BRSBPG	BR Storage Block (BR or IBR) to permuter
CCRLG	Core register left justified
CSB	Cell Size is Byte
CSBH	Cell Size is Byte or Halfword
CSH	Cell Size is Halfword
CTPG	Cable terminators is permuter input
DBRJ	Right justify permuter output
DRPG	DR is permuter input
IB(i)	Enable permuter byte (i) inhibit
IBC(i)	Inhibit permuter byte (i) from control
IBPL13	Inhibit byte 1 - permute left 3
IBPL21	Inhibit byte 2 - permute left 1
IBRS	Select IBR portion of BR-IBR Storage
ICTPG	Low order bit masking for ICT
IRPG	IR is permuter input
LJIE	Left justify
LJØPE	Enable left justify on 2nd data byte
LRPG	LR is permuter input
LRSEL	Inhibit normal permutation control

M16E	Generate -16 on DBP
M1E	Generate -1 on DBP
M2E	Generate -2 on DBP
M32E	Generate -32 on DBP
M4E	Generate -4 on DBP
M8E	Generate -8 on DBP
NRSETG	Initialize name regs via ACT, CCT and Permuter
ØPJE	Justify OP code on byte 1 from IBR to IR
ØSPFG	Gate false out of OS to Permuter
ØSPG	Gate Operand Stack to Permuter
ØSPTG	Gate true out of ØS to Permuter
ØSSS	Operand Stack Storage Select
ØSTPG	Gate ØSTR to Permuter via ØSR
P16E	Generate +16 on DBP
P1E	Generate +1 on DBP
P2E	Generate +2 on DBP
P32E	Generate +32 on DBP
P4E	Generate +4 on DBP
P8E	Generate +8 on DBP
PLC1	Permute left 1 byte from Control
PLC2	Permute left 2 bytes from Control
PLC3	Permute left 3 bytes from Control
PRPG	PRB (pointer register output line) is permuter input
RJIE	Right justify
RSDE	Reverse source and destination bit
SCRPG	Low order bit masking for SCR

Local Variables:

<u>Simulation Name</u>	<u>Description</u>
BRSPG	Gate BRØS to permuter
D(i)	Permutation destination bit (i)
FBI(i)	Permuter input gate output - bit i
IBE(i)	Inhibit Permuter byte i
IBE 3(i)	Inhibit Permuter byte 3 - bit i
ØSR(1)	Bus from ØS Registers to Permuter - bit i
PLOE	Enable permute left 0 bytes
PL1E	Enable permute left 1 byte
PL2E	Enable permute left 2 bytes
PL3E	Enable permute left 3 bytes
PLIOE	Intermediate signal - permute left 0 bytes
PLI1E	Intermediate signal - permute left 1 byte
PLI2E	Intermediate signal - permute left 2 bytes
PLI3E	Intermediate signal - permute left 3 bytes
S(i)	Permutation source bit(i)
TEMP(i)	Temporary storage - bit (i) -- used to hold source bits in the event of a reverse of source and destination.

4.11 ADD32 Procedure

4.11.1 ADD32 Description

The ADD32 procedure simulates all the processes of the 32 bit adder of the TP. The procedure listing looks tremendously forboding because of the many similar equations involved, none of which could be fit into a nice DØ-group. The reasons behind this are varied. The flags are not added, so equations using successive bits of the inputs must jump over the flag bits. The adder uses 2's complement representation and employs two levels of carry lookahead to hasten carry propagation. A complete description of the logic making up the adder can be found in Section 2.7 of the TP Manual (1).

The T (carry Transmit) and G (carry Generate) signals are seen to be both arrays and single signals--the signal T_i is defined T position (i). This was done for the sake of efficiency in coding. When it is possible a DØ-group is always more efficient than a number of statements which do the same thing. However there is only one small beginning section of the adder where it is possible to use a DØ-group--that is where $T(i)$ is set equal to the exclusive or of $DR(i)$ and $DB(i)$, and $G(i)$ is set equal to the boolean "and" of $DB(i)$ and $DR(i)$. These two operations are done using the T and G arrays. From this point on, nothing can be done using a do-group; therefore instead of referencing the numerous array elements, which is very inefficient, the single signals were used.

The adder output consists of 8 groups of 4 bits each--2 groups for each byte. For each group there is a carry out signal ($COUT_j$), a propagate signal (P_j), a carry in signal (CIN_j), and the ripple carry into the i^{th} position of group j (CiG_j). The equations for these signals are derived straightforwardly from the logic. The adder output goes to the ARN.

At the end of the ADD32 procedure is a comprehensive print routine which will print the states of most of the internal signals used. This is activated by the simulator control signal ADDOUT. This signal has no counterpart in the logic; it is completely made up for the simulator, and is seldom used since the simulated adder seems to work properly.

4.11.2 ADD32 Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
ADD32: PROC(ADDE,CINJ, EQVE, XORE, ADDOUT);
DCL(ADDE, CINJ, EQVE,XORE) BIT(1);
DCL ADDOUT BIT(1);
DCL(ARN,DB,DR,T)(36) BIT(1) EXTERNAL;
DCL G(36) BIT(1) EXTERNAL,
G1 BIT(1) DEFINED G POSITION(1 ),
G2 BIT(1) DEFINED G POSITION(2 ),
G3 BIT(1) DEFINED G POSITION(3 ),
G4 BIT(1) DEFINED G POSITION(4 ),
G5 BIT(1) DEFINED G POSITION(5 ),
G6 BIT(1) DEFINED G POSITION(6 ),
G7 BIT(1) DEFINED G POSITION(7 ),
G8 BIT(1) DEFINED G POSITION(8 ),
G9 BIT(1) DEFINED G POSITION(9 ),
G10 BIT(1) DEFINED G POSITION(10),
G11 BIT(1) DEFINED G POSITION(11),
G12 BIT(1) DEFINED G POSITION(12),
G13 BIT(1) DEFINED G POSITION(13),
G14 BIT(1) DEFINED G POSITION(14),
G15 BIT(1) DEFINED G POSITION(15),
G16 BIT(1) DEFINED G POSITION(16),
G17 BIT(1) DEFINED G POSITION(17),
G19 BIT(1) DEFINED G POSITION(19),
G20 BIT(1) DEFINED G POSITION(20),
G21 BIT(1) DEFINED G POSITION(21),
G22 BIT(1) DEFINED G POSITION(22),
G23 BIT(1) DEFINED G POSITION(23),
G24 BIT(1) DEFINED G POSITION(24),
G25 BIT(1) DEFINED G POSITION(25),
G26 BIT(1) DEFINED G POSITION(26),
G28 BIT(1) DEFINED G POSITION(28),
G29 BIT(1) DEFINED G POSITION(29),
G30 BIT(1) DEFINED G POSITION(30),
G31 BIT(1) DEFINED G POSITION(31),
G32 BIT(1) DEFINED G POSITION(32),
G33 BIT(1) DEFINED G POSITION(33),
G34 BIT(1) DEFINED G POSITION(34),
G35 BIT(1) DEFINED G POSITION(35),
T1 BIT(1) DEFINED T POSITION(1 ),
T2 BIT(1) DEFINED T POSITION(2 ),
T3 BIT(1) DEFINED T POSITION(3 ),
T4 BIT(1) DEFINED T POSITION(4 ),
T5 BIT(1) DEFINED T POSITION(5 ),
T6 BIT(1) DEFINED T POSITION(6 ),
T7 BIT(1) DEFINED T POSITION(7 ),
T8 BIT(1) DEFINED T POSITION(8 ),
T10 BIT(1) DEFINED T POSITION(10),
T11 BIT(1) DEFINED T POSITION(11),
T12 BIT(1) DEFINED T POSITION(12),
T13 BIT(1) DEFINED T POSITION(13),
T14 BIT(1) DEFINED T POSITION(14),
T15 BIT(1) DEFINED T POSITION(15),
T16 BIT(1) DEFINED T POSITION(16),
T17 BIT(1) DEFINED T POSITION(17),
```

```

T19 BIT(1) DEFINED T POSITION(19),
T20 BIT(1) DEFINED T POSITION(20),
T21 BIT(1) DEFINED T POSITION(21),
T22 BIT(1) DEFINED T POSITION(22),
T23 BIT(1) DEFINED T POSITION(23),
T24 BIT(1) DEFINED T POSITION(24),
T25 BIT(1) DEFINED T POSITION(25),
T26 BIT(1) DEFINED T POSITION(26),
T28 BIT(1) DEFINED T POSITION(28),
T29 BIT(1) DEFINED T POSITION(29),
T30 BIT(1) DEFINED T POSITION(30),
T31 BIT(1) DEFINED T POSITION(31),
T32 BIT(1) DEFINED T POSITION(32),
T33 BIT(1) DEFINED T POSITION(33),
T34 BIT(1) DEFINED T POSITION(34),
T35 BIT(1) DEFINED T POSITION(35);
DCL ( /*INTERNAL VARIABLES,*/
/* CARRY PROPAGATE SIGNALS*/
P1,P2,P3,P4,P5,P6,P7,P8,
/* CARRY INTO ITH ADDER GROUP */
CIN1,CIN2,CIN3,CIN4,CIN5,CIN6,CIN7,CIN8,
/* CARRY GENERATED WITHIN ITH ADDR GROUP*/
COUT1,COUT2,COUT3,COUT4,COUT5,COUT6,COUT7,COUT8,
/* CARRY INTO FIRST POSITION OF ITH ADDER GROUP*/
C1G1,C1G2,C1G3,C1G4,C1G5,C1G6,C1G7,C1G8,
/*CARRY INTO SECOND POSITION OF ITH ADDER GROUP*/
C2G1,C2G2,C2G3,C2G4,C2G5,C2G6,C2G7,C2G8,
/*CARRY INTO THIRD POSITION OF ITH ADDER GROUP*/
C3G1,C3G2,C3G3,C3G4,C3G5,C3G6,C3G7,C3G8,
/*CARRY INTO FOURTH POSITION OF ITH ADDER GROUP*/
C4G1,C4G2,C4G3,C4G4,C4G5,C4G6,C4G7,C4G8
) BIT(1);
PUT LIST('ADD32 ENTERED');
/* CALCULATE THE CARRY TRANSMIT SIGNALS (T) AND CARRY GENERATE
SIGNALS (G) */
IF EQVE|ADDE|XORE THEN DO J=0 TO 3;
DO I=J*9+1 TO J*9+8;
T(I)=DB(I)&~DR(I)|~DB(I)&DR(I);
G(I)=DB(I)&DR(I);
END;
END;
COUT1=G1|T1&G2|T1&T2&G3|T1&T2&T3&G4;
COUT2=G5|T5&G6|T5&T6&G7|T5&T6&T7&G8;
COUT3=G10|T10&G11|T10&T11&G12|T10&T11&T12&G13;
COUT4=G14|T14&G15|T14&T15&G16|T14&T15&T16&G17;
COUT5=G19|T19&G20|T19&T20&G21|T19&T20&T21&G22;
COUT6=G23|T23&G24|T23&T24&G25|T23&T24&T25&G26;
COUT7=G28|T28&G29|T28&T29&G30|T28&T29&T30&G31;
COUT8=G32|T32&G33|T32&T33&G34|T32&T33&T34&G35;
/* THE PROPAGATEGENERATOR: THE P SIGNALS INDICATE WHETHER THE
4TH BIT GROUP WILL CONDUCT AN INPUT CARRY ALL THE WAY
THROUGH THE GROUP */
P1=T1&T2&T3&T4;
P2=T5&T6&T7&T8;
P3=T10&T11&T12&T13;
P4=T14&T15&T16&T17;
P5=T19&T20&T21&T22;

```

```

P6=T23&T24&T25&T26;
P7=T28&T29&T30&T31;
P8=T32&T33&T34&T35;
CIN3= COUT4
      |COUT5&P4
      |COUT6&P4&P5
      |COUT7&P4&P5&P6
      |COUT8&P4&P5&P6
      |CINJ &P4&P5&P6&P7&P8;
CIN4= COUT5
      |COUT6&P5
      |COUT7&P5&P6
      |COUT8&P5&P6&P7
      |CINJ &P5&P6&P7&P8;
CIN5= COUT6
      |COUT7&P6
      |COUT8&P6&P7
      |CINJ &P6&P7&P8;
CIN6= COUT7
      |COUT8&P7
      |CINJ &P7&P8;
CIN7= COUT8
      |CINJ&P8;
CIN8=CINJ;
/*CALCULATE RIPPLE CARRY */
CIN2= COUT3|CIN3&P3;
CIN1= COUT2|COUT3&P2|CIN3&P2&P3;
C1G1=G2|T2&G3|T2&T3&G4|T2&T3&T4&CIN1;
C2G1=G3|T3&G4|T3&T4&CIN1;
C3G1=G4|T4&CIN1;
C4G1=CIN1;
C1G2=G6|T6&G7|T6&T7&G8|T6&T7&T8&CIN2;
C2G2=G7|T7&G8|T7&T8&CIN2;
C3G2=G8|T8&CIN2;
C4G2=CIN2;
C1G3=G11|T11&G12|T11&T12&G13|T11&T12&T13&CIN3;
C2G3=G12|T12&G13|T12&T13&CIN3;
C3G3=G13|T13&CIN3;
C4G3=CIN3;
C1G4=G15|T15&G16|T15&T16&G17|T15&T16&T17&CIN4;
C2G4=G16|T16&G17|T16&T17&CIN4;
C3G4=G17|T17&CIN4;
C4G4=CIN4;
C1G5=G20|T20&G21|T20&T21&G22|T20&T21&T22&CIN5;
C2G5=G21|T21&G22|T21&T22&CIN5;
C3G5=G22|T22&CIN5;
C4G5=CIN5;
C1G6=G24|T24&G25|T24&T25&G26|T24&T25&T26&CIN6;
C2G6=G25|T25&G26|T25&T26&CIN6;
C3G6=G26|T26&CIN6;
C4G6=CIN6;
C1G7=G29|T29&G30|T29&T30&G31|T29&T30&T31&CIN7;
C2G7=G30|T30&G31|T30&T31&CIN7;
C3G7=G31|T31&CIN7;
C4G7=CIN7;
C1G8=G33|T33&G34|T33&T34&G35|T33&T34&T35&CIN8;
C2G8=G34|T34&G35|T35&CIN8;

```

```

C3G8=G35|T35&CIN8;
C4G8=CIN8;
/*CALCULATE THE OUTPUT SIGNALS*/
/*FIRST GROUP*/
ARN(1)=T1&-C1G1|~T1&C1G1;
ARN(2)=T2&-C2G1|~T2&C2G1;
ARN(3)=T3&-C3G1|~T3&C3G1;
ARN(4)=T4&-C4G1|~T4&C4G1;
/* GROUP # 2*/
ARN(5)=T5&-C1G2|~T5&C1G2;
ARN(6)=T6&-C2G2|~T6&C2G2;
ARN(7)=T7&-C3G2|~T7&C3G2;
ARN(8)=T8&-C4G2|~T8&C4G2;
ARN(10)=T10&-C1G3|~T10&C1G3;
ARN(11)=T11&-C2G3|~T11&C2G3;
ARN(12)=T12&-C3G3|~T12&C3G3;
ARN(13)=T13&-C4G3|~T13&C4G3;
/*GROUP#4*/
ARN(14)=T14&-C1G4|~T14&C1G4;
ARN(15)=T15&-C2G4|~T15&C2G4;
ARN(16)=T16&-C3G4|~T16&C3G4;
ARN(17)=T17&-C4G4|~T17&C4G4;
/*GROUP#5 */
ARN(19)=T19&-C1G5|~T19&C1G5;
ARN(20)=T20&-C2G5|~T20&C2G5;
ARN(21)=T21&-C3G5|~T21&C3G5;
ARN(22)=T22&-C4G5|~T22&C4G5;
/*GROUP #6 */
ARN(23)=T23&-C1G6|~T23&C1G6;
ARN(24)=T24&-C2G6|~T24&C2G6;
ARN(25)=T25&-C3G6|~T25&C3G6;
ARN(26)=T26&-C4G6|~T26&C4G6;
/*GROUP #7 */
ARN(28)=T28&-C1G7|~T28&C1G7;
ARN(29)=T29&-C2G7|~T29&C2G7;
ARN(30)=T30&-C3G7|~T30&C3G7;
ARN(31)=T31&-C4G7|~T31&C4G7;
/*GROUP #8*/
ARN(32)=T32&-C1G8|~T32&C1G8;
ARN(33)=T33&-C2G8|~T33&C2G8;
ARN(34)=T34&-C3G8|~T34&C3G8;
ARN(35)=T35&-C4G8|~T35&C4G8;
IF ADDOUT THEN DO;
PUT SKIP(5);
CALL PRINT1(DB,'DB ');
CALL PRINT1(DR,'DR ');
CALL PRINT1(G,'G ');
CALL PRINT1(T,'T ');
CALL PRINT1(ARN,'ARN ');
PUT EDIT('GROUP','1','2','3','4','5','6','7','8')(SKIP,A(10),
8 (X(2),A(1)));
PUT EDIT('COUT(I)=' ,COUT1,COUT2,COUT3,COUT4,COUT5,COUT6,
COUT7,COUT8)
(SKIP,A(10),8 (X(2),B(1)));
PUT EDIT('P(I)=' ,P1,P2,P3,P4,P5,P6,P7,P8)
(SKIP,A(10),8 (X(2),B(1)));
PUT EDIT('CIN(I)=' ,CIN1,CIN2,CIN3,CIN4,CIN5,CIN6,CIN7,CIN8)

```

```
      (SKIP,A(10),8 (X(2),B(1)));  
PUT EDIT('C1G(I)=',C1G1,C1G2,C1G3,C1G4,C1G5,C1G6,C1G7,C1G8)  
      (SKIP,A(10),8 (X(2),B(1)));  
PUT EDIT('C2G(I)=',C2G1,C2G2,C2G3,C2G4,C2G5,C2G6,C2G7,C2G8)  
      (SKIP,A(10),8 (X(2),B(1)));  
PUT EDIT('C3G(I)=',C3G1,C3G2,C3G3,C3G4,C3G5,C3G6,C3G7,C3G8)  
      (SKIP,A(10),8 (X(2),B(1)));  
PUT EDIT('C4G(I)=',C4G1,C4G2,C4G3,C4G4,C4G5,C4G6,C4G7,C4G8)  
      (SKIP,A(10),8 (X(2),B(1)));
```

```
END;
```

```
END ADD32;
```

```
/*
```

4.11.3 ADD32 Variable List

Registers

(External variables):

<u>Simulation Name</u>	<u>Description</u>
ARN(i)	Input bus to the AR - bit i
DB(i)	Distribution bus - bit i
DR(i)	Distribution register - bit i
G(i)	Holds boolean "AND" of DR and DB - bit i
T(i)	Output of 32 bit adder - bit i

Control Signals:

<u>Simulation Name</u>	<u>Description</u>
ADDE	Enable addition on 32 bit adder
ADDØUT	Triggers printout of all variables of adder for diagnostic purposes
CINJ	Inject carry into 32 bit adder
EQVE	Boolean "equivalence" enable
XØRE	Boolean "exclusive or" enable

Local variables:

<u>Simulation Name</u>	<u>Description</u>
CiGj	Carry into i^{th} position of j^{th} adder group
CINj	Carry into j^{th} adder group
COUTj	Carry generated within j the adder group
Gi	Generate carry in i^{th} bit position
Pj	Propagate carry through j^{th} adder group
Ti	Transmit carry through i^{th} bit position. Also equal to the exclusive or of DB and DR

4.12 PER-OUT Procedure

The PER-OUT procedure simulates that portion of the permuter which gates data from the DBP, DB, and DBP (which are all equivalent at this point) into other registers. It also does direct gating from register to register.

There are several sections of PER-OUT which appear to be misplaced. For instance gating into the pointer registers, base registers and operand stack is, conceptually, a part of those procedures. However, none of these operations can be done until the DB is in its final state--after PER-IN has been called. In one case, since the ARN is changed by the adder, and the adder has the DB as its input, the adder must be called after PER-IN but before PER-OUT, since it is not until PER-OUT that the ARN reaches its final state and is gated to the AR. There are other complications with the placement of the adder. The DR is an input to the adder, but the DR may be changed after PER-OUT. These are timing problems which are difficult to simulate. It has been determined that if the adder is to be called, and the contents of the DR is to be changed, both within the same control point, then it is usually the original contents of the DR which is desired as input to the adder. Hence the necessity of the two procedures PER-IN and PER-OUT.

4.12.1 PER-OUT Description

The PER-OUT procedure checks signals which indicate gating data from the DBP to any of the many other registers - the AR, IR, DR, LR, CD, SCR, ØSTR, ØS, PR, SNR, PRSNR, ASRG, and BR.

Often the signals which indicate this gating are independent control signals. Other times, they are dependent signals which are set in other procedures of the simulator.

Each of the registers AR, DR, LR, and IR have gate and latch signals for each byte. The gate and latch signals are the complements of each other. For instance, if the ARG(0) signal is on (AR byte 0 gate) then ARL(0) is off (AR byte 0 latch). If the gate is on for a particular byte, then information can be transferred into that byte. If the latch is on, no information can be put into it.

First signals which cause gating into the DR are checked. The DR can be gated into from the DBP, AR, or LR.

Next, gating into the LR is done via the LRN bus. All inputs to the LR are first "dot-ored" into the LRN. The final state of LRN is then gated into the LR. If the "exclusive-or" or "equivalence" functions are to be done, they are performed here by gating the contents of the T register (from the adder) into the LRN. The LRN flags are set to one if the corresponding DB and DR flags are equal (equivalent). Data can also be put into the LRN from the DB and DR. Then if LRG(i) is on, LRN byte (i) is gated into LR byte (i). A single byte can be inhibited if the corresponding DLRN signal is on.

The IR can be gated into from the DBP, depending on the state of the latch and inhibit signals.

The AR can also be gated into from the DB via the ARN. This operation is similar to the manner in which the LR receives information. The CD is loaded here from the DB.

The next section checks for initialization of the operand stack. The operand stack select signals (ØSS(i)) were determined in ØPSTKIN. The link and value fields of the selected pointer register (PR) can be gated into from the DBP with optional inclusion of the flags.

The link and value fields of the SBR can also be loaded here from the DBP.

Next the shadow name registers (SNR) are loaded from the DBP for interrupt recovery. Interrupt cycle 1 (indicated by BSR1G) requires gating the DBP into SNR8-14. Interrupt cycle 2 (indicated by BSR2G) requires gating the DBP into SNR0-7. The DBP bits are taken 4 at a time (excluding flag bits) and loaded consecutively into the SNR's. For example if BSR1G is on, SNR(8) = DBP bits 1 to 4
SNR(9) = DBP bits 5 to 8
SNR(10) = DBP bits 10 to 14

A pointer register segment name register (PRSNR) can now be gated into if one has been selected. The select signals are determined in the BREGIN procedure. The PRSNR's are two bytes long, and the first two bytes of the DBP are the ones to be gated.

The selected associative register (ASRG) can be written into from the DBP. Like the PRSNR's, the associative registers are two bytes long; as before the first two bytes of the DBP are the ones to be gated, excluding the flags. The selection process is done in the BREGIN procedure. The base register (BR) which was also selected in BREGIN is also loaded here from the DBP.

Finally the contents of the LR, AR, and DR are gated into their respective output buses LRB, ARB, and DRB.

4.12.2 PER_OUT Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
  PER_OUT: PROC (ADDE,      ADRG,      ASRWE,      BARG,      BDRG,
                BIRG,      BLRG,      BRGS,      BSR1G,      BSR2G,
                DBPRG,      DBSTG,      DLRG,      EQVE,      FLPRN,
                LDRG,      LSE,      RSE,      RWBGE,      SBRs,
                WRBRE,      WRPRLE,      WRPRVE,      WSBLE,      WSBVE,
                XORE,      XRVe,      ANDE,
                ADDN,      ARG,      ARGG,      ARL,      BIRN,
                DBDG,      DLRN,      DRL,      IRG,      IRL,
                LRDG,      LRG,      LRL,      DBOSG,      CDRE,
                ARF33,      ARF34,      ARF35,      ARARFG);
  DCL (ADDE, ADRG,      ANDE,      ASRWE,      BARG,      BDRG,
       BIRG,      BLRG,      BRGS,      BSR1G,      BSR2G,
       DBPRG,      DBSTG,      DLRG,      EQVE,      FLPRN,
       LDRG,      LSE,      RSE,      RWBGE,      SBRs,
       WRBRE,      WRPRLE,      WRPRVE,      WSBLE,      WSBVE,
       XORE,      XRVe,      DBOSG,      CDRE) BIT(1),
  (ADDN, ARG,      ARGG,      ARL,      BIRN,
   DBDG, DLRN,      DRL,      IRG,      IRL,
   LRDG, LRG,      LRL)(0:3) BIT(1) ;
  DCL (ARF33,ARF34,ARF35,ARARFG) BIT(1);
  DCL ( (AR,ARN,DB,DBB,DBP,DR,IR,LR,LRN,SBR,T)(36),OSTR(2),
        SCR(5), (ASRWE1,ASRWE2,RBRs)(7), BRS(0:7),
        PRS(1:14), (PSNWS1,PSNWS2)(0:15), OSS(0:31),
        BR(0:7,36), SNR(0:14,4), PR(0:14,36),
        PRSNR(0:15,18), OS(0:31,9), ASRG(7,18)) BIT(1)
  EXTERNAL;
  DCL (ARB,LRB,DRB) (36) BIT(1) EXTERNAL;
  DCL CD(36) BIT(1) EXTERNAL;
  IF CDRE THEN CD=DB;
  /* TRANSFER T TO LRN      DRAWING 07-1 */
  XRVe=XORE|EQVE;
  IF XRVe THEN DO;
    LRN=-T;
    DO I=9 TO 36 BY 9;
      IF DB(I)=DR(I) THEN LRN(I)='1'B;
    END;
  END;
END;

/* SEND OUTPUT TO PROPER REGISTERS.*/
DRL='1'B;
DBDG,ARGG,LRDG='0'B;

IF BDRG|ADRG|LDRG THEN DRL='0'B;

IF BDRG THEN DO;
  DBDG='1'B;
  DR=DBP;
END;
IF ADRG THEN DO;
  ARGG='1'B;
  DR=AR;
END;
IF LDRG THEN DO;
  LRDG='1'B;
```

```

        DR=LR;
        END;
IF BLRG THEN LRN(*)=DB(*)|LRN(*) ;
IF DLRG THEN LRN(*)=DR(*)|LRN(*) ;
/* CALCULATE LR GATE AND LATCH SIGNALS  DRAWING 04-4 */
DO I=0 TO 3;
LRG(I)=RSE|LSE|~DLRN(I)&DLRG|BLRG|ANDE|XRVE;
END;
LRL=~LRG;

DO I=0 TO 3;
    IF LRG(I) THEN DO J=I*9+1 TO I*9+9;
        LR(J)=LRN(J);
    END;
END;

IF BIRG THEN DO I=0 TO 3;
    IF ~BIRN(I) THEN DO;
        IRL(I)='0'B;
        IRG(I)='1'B;
        DO J=I*9+1 TO I*9+9;
            IR(J)=DBP(J);
        END;
    END;
END;

        END;
    IF BARG THEN ARN=ARN|DB;
DO I=0 TO 3;
    ARG(I)=BARG|ADDE&~ADDN(I);
END;
ARG(0)=ARG(0)|RWBGE;
ARL=~ARG;
DO I=0 TO 3;
    IF ARG(I) THEN DO J=I*9+1 TO I*9+9;
        AR(J)=ARN(J);
    END;
END;
IF ARAREG THEN DO;
    ARF33=AR(33);
    ARF34=AR(34);
    ARF35=AR(35);
END;
/* CHECK FOR INITIALIZATION OF STACK */
/* DBSTG WILL BE ON IF THE OSTR AND SCR ARE TO BE RELOADED
WITH A NEW OPERAND STACK POINTER ADDRESS */

IF DBSTG THEN DO;
    DO I=1 TO 2;
        OSTR(I)=DB(I+30);
    END;

    DO I=1 TO 5;
        SCR(I)=DB(I+30);
    END;
END;

```

```

/* LOAD OS FROM DB */

IF DBOSG THEN DO I=0 TO 31;
  IF OSS(I) THEN DO;
    L=MOD(I,4);
    DO J=1 TO 9;
      OS(I,J)=DB(L*9+J);
    END;
  END;
END;

```

```

/* WRITE ON SELECT POINTER OR SBR IF DESIRED */
IF DBPRG THEN DO;

```

```

  IF WRPRLE THEN DO I=0 TO 14;
    IF PRS(I) THEN DO;
      IF FLPRN THEN DO;
        DO J=1 TO 8;
          PR(I,J)=DBP(J);
        END;
        DO J=10 TO 17;
          PR(I,J)=DBP(J);
        END;
      END;
    ELSE DO J=1 TO 18;
      PR(I,J)=DBP(J);
    END;
  END;
END;

```

```

  IF WRPRVE THEN DO I=0 TO 14;
    IF PRS(I) THEN DO;
      IF FLPRN THEN DO;
        DO J=19 TO 26;
          PR(I,J)=DBP(J);
        END;
        DO J=28 TO 35;
          PR(I,J)=DBP(J);
        END;
      END;
    ELSE DO J=19 TO 36;
      PR(I,J)=DBP(J);
    END;
  END;
END;

```

```

  IF WSBLE&SBR5 THEN
    DO I=1 TO 18;
      SBR(I)=DBP(I);
    END;

```

```

  IF WSBVE & SBR5 THEN
    DO I=19 TO 36;
      SBR(I)=DBP(I);
    END;
END;

```

```

/* INTERRUPT RECOVERY FOR SHADOW NAME REGISTER */
IF BSR1G THEN DO I=0 TO 2;
  DO J=1 TO 4;
    SNR(2*I+8,J)=DB(I*9+J);
  END;
END;

```



```

        SNR(2*I+9,J)=DB(I*9+J+4);
    END;
    DO J=1 TO 4;
        SNR(14,J)=DB(27+J);
    END;
END;
IF BSR2G THEN DO I=0 TO 3;
    DO J=1 TO 4;
        SNR(2*I,J)=DB(I*9+J);
        SNR(2*I+1,J)=DB(I*9+J+4);
    END;
END;

/* CHECK FOR WRITING INTO THE POINTER REGISTER SEGMENT NAME
   REGISTER PRSNR */
/* DEFINE PRSNR(0:15,18) AS THE INTERNAL REGISTERS ON
   DRAWINGS 22-4 TO 22-19 IN THE LOGIC BOOK */
DO I=0 TO 15;
    IF PSNWS1(I) THEN DO J=1 TO 8;
        PRSNR(I,J)=DBB(J);
    END;
END;
DO I=0 TO 15;
    IF PSNWS2(I) THEN DO J=10 TO 17;
        PRSNR(I,J)=DBB(J);
    END;
END;

/* ASSOCIATIVE REGISTER WRITE SIGNAL DRIVERS */
IF ASRWE THEN DO I=1 TO 7;
    IF RBR5(I) THEN DO;
        ASRWE1(I),ASRWE2(I)='1'B;
        DO J=1 TO 17 WHILE (J<=9);
            ASRG(I,J)=DBB(J);
        END;
    END;
END;
ELSE ASRWE1,ASRWE2='0'B;

/* WRITE INTO BASE REGISTER */
IF BRGS&WRBRE THEN DO I=0 TO 7;
    IF BRS(I) THEN BR(I,*)=DBP(*);
END;
LRB=LR;
ARB=AR;
DRB=DR;
END PER_OUT;
/*

```

4.12.3 PER_OUT Variable List

Registers and Counters (external variables):

<u>Simulation Name</u>	<u>Description</u>
AR(i)	Arithmetic Register - bit i
ARN(i)	AR Input Bus - bit i
ASRG (i,j)	Associate Register i - bit j
ASRWE1(i)	Associate Register i - write enable Signal 1
ASRWE2(i)	Associate Register i - write enable Signal 2
BR(i,j)	Base register i - bit j
BRS(i)	Select base register i
DB(i)	Distribution register - bit i
DBB(i)	Control Section DB driver - bit i
DBP(i)	Perputer output - bit i
DR(i)	Distribution Register - bit i
IR(i)	Instruction register - bit i
LR(i)	Logic register - bit i
LRN(i)	LR input bus - bit i
ØS(i,j)	Operand Stack byte i - bit j
ØSS(i)	Operand Stack Select-byte i
ØSTR(i)	Operand Stack top register - bit i
PR(i,j)	Pointer register i - bit j
PRS(i)	Select pointer register i
PRSNR(i,j)	Pointer register segment name register i - bit j
PSNWS1(i)	Pointer register segment name register i-write select 1
PSNWS2(i)	Pointer register segment name register i-write select 2
RBRS(i)	Select BR(i) for replacement.

<u>Simulation Name</u>	<u>Description</u>
SBR(i)	Spare buffer register bit i
SCR	Stack control register - bit i
SNR(i,j)	Segment name register i - bit j
T(i)	DB(i) * EQV * DR(i) (32 bit adder signal)

Control Signals (1 or 4 bit parameters):

<u>Simulation Name</u>	<u>Description</u>
ADDE	Enable 32 bit adder
ADDN(i)	Inhibit gate
ADRG	Arithmetic register to distribution register
ANDE	DR AND DB to LR
ARG(i)	Gate byte i ARN to AR
ARGG(i)	Gate byte i AR to DR
ARL(i)	AR latch signal - byte i
ASRWE	Enable write into associative register selected by BRS signal
BARG	Gate DB to AR
BDRG	Gate DB to DR
BIRG	Gate DB to IR
BIRN(i)	Inhibit DB to IR, byte i
DLRG	Gate DB to LR
BRGS	Select BR portion of DR-IBR storage
BSR1G	Gate DB to shadow name registers 8-14 - Interrupt cycle 1
BSR2G	Gate DB to shadow name registers 0-7 - Interrupt cycle 2
DBDG(i)	Gate DB to DR byte i
DBØSG	Gate DBP to ØS storage
DBPRG	Gate DB to PR or SBR
DBSTG	Gate DB to OSTR and SCR
DLRG	Gate DR to LR direct

<u>Simulation Name</u>	<u>Description</u>
DLRN(i)	Inhibit DR to LR gate, byte i
DRL(i)	DR latch signal - byte i
EQVE	Enable Boolean equivalence
FLPRN	Inhibit writing of flag bits into PR storage
IRG(i)	Gate DB to IR byte i
IRL(i)	IR latch signal - byte i
LDRG	Gate LR to DR direct
LRDG(i)	Gate LR to DR - byte i
LRG(i)	Gate LRN to LR - byte i
LRL(i)	LR latch signal - byte i
LSE	Enable left shift, DR to LR
RSE	Enable right shift, DR to LR
RWBGE	Enable read/write byte generator
SBRS	Select spare buffer register
WRBRE	Read/write base register-write when 1
WRPRLE	Read/write PR link - write when 1
WRPRVE	Read/write PR value - write when 1
WSBLE	Read/write SBR link - write when 1
WSBVE	Read/write SBR values - write when 1
XØRE	DB.XØR.DR .TO. LR
XRVE	On if XØRE.ØR.EQVE=1

4.13 ADDØVF Procedure

4.13.1 ADDØVF Description

The ADDØVF procedure calculates the AØV (32 bit adder overflow) signal.

$$\text{The AØV signal} = \text{DR}(i) \cdot \text{DB}(I) \cdot \overline{\text{AR}(i)} \\ \vee \overline{\text{DR}(i)} \cdot \overline{\text{DB}(i)} \cdot \text{AR}(i)$$

This is by all rights a part of the adder itself. However, because of the order in which the procedures are called, the AR has not been calculated by the time the adder is called.

If RSYS = '1' (Reset System) AØV is set to 0.

4.13.2 ADDOVF Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
    ADDOVF: PROC (ADDE, AOV,      AUCG,      CSB,      CSH,
                  CSW,      RSYS);
    DCL (ADDE, AOV,      AUCG,      CSB,      CSH,      CSW,
          RSYS) BIT(1);
    DCL ( AR, DB, DR, LR)(36) BIT(1) EXTERNAL;
    PUT LIST('ADDOVF ENTERED');
    /* CHECK FOR ADDER OVERFLOW */
    AOV='0'B;
    IF LR(9)&AUCG THEN AOV='1'B;
    ELSE DO;
        IF CSW THEN AOV=ADDE&(DR(1)&DB(1)&~AR(1)
                               |DR(1)&~DB(1)&AR(1));
        IF CSH THEN AOV=ADDE&(DR(19)&DB(19)&~AR(19)
                               |~DR(19)&~DB(19)&AR(1));

        IF CSB THEN AOV=ADDE&(DR(28)&DB(28)&~AR(28)
                               |~DR(28)&~DB(28)&AR(28));
    END;
    IF RSYS THEN AOV='0'B;
END ADDOVF;
/*
```

4.13.3 ADDØVF Variable List

Registers and Counters (External Variables):

<u>Simulation Name</u>	<u>Description</u>
AR(i)	Arithmetic Register - bit i
DB(i)	Distribution Bus - bit i
DR(i)	Distribution Register - bit i
LR(i)	Logic Register - bit i

Control Signals (1 or more bit parameters):

<u>Simulation Name</u>	<u>Description</u>
ADDE	Enable 32 bit adder
AOV	32-bit adder overflow
AUCG	AU flags to LT, GT, EQ, from LR, DR
CSB	Cell size is byte
CSH	Cell size is halfword
CSW	Cell size is word
RSYS	Reset System

4.14 CØMPARE Procedure

The CØMPARE procedure simulates the actions of the Taxicrinic Processor which do algebraic and logical comparisons.

The comparisons are made by doing a subtraction and checking the result which is in the AR. Then, depending upon the cell size, a portion of the AR is tested.

4.14.1 COMPARE Description

The COMPARE procedure sets the greater than (GT), less than (LT), and equal (EQ) flip-flops to 1 or 0 depending upon control signals and the contents of the AR after a subtraction is done.

The LT and GT flip-flops are reset to 0 or 1 if necessary depending on the states of the various control signals involved. These are defined in the variable list.

Then the contents of the AR are examined for the sign bit - the position of which depends upon the cell size. If the sign is positive LT is set to 1. If the sign is negative, GT is set to 1.

To test for zero, all bits of the AR must be tested for zero if cell size is word, the last two bytes if cell size is halfword, and the last byte if cell size is byte. ZERO, an internal variable, goes to one if any of the tested bits of the AR is one. If ZERO remains equal to zero, then EQ is set to one.

4.14.2 COMPARE Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
    COMPARE:  PROC(      AUCG,      CPAE,      CPHE,      CPLE,
                    CSB,      CSH,      CSW,      EQ,      GT,
                    LT,      RCOMFF,  REQS,  RGLE,  SGTS,SLTS,
                    TZE);
    DCL(AUCG, CPAE,      CPHE,      CPLE,      CSB,      CSH,
        CSW,      EQ,      GT,      LT,      RCOMFF,
        REQS,RGLE,SGTS,SLTS,TZE) BIT(1);
    DCL( AR,DR,LR)(36) BIT(1) EXTERNAL;
    DCL (I,ZERO)FIXED BIN ;
    IF RCOMFF THEN LT,GT='0'B;
    IF EQ&RGLE THEN GT='1'B;
    IF LR(27)&AUCG|SLTS THEN LT='1'B;
    IF DR(9)&AUCG|SGTS THEN GT='1'B;
    IF (AR(28)&CSB&CPLE
        |AR(1)&CSW&(CPLE|CPAE)
        |AR(19)&CSH&(CPLE|CPAE)
        |AR(35)&CPHE)
        THEN GT='1'B;

    IF (¬AR(28)&CSB&CPLE
        |¬AR(1)&(CPLE|CPAE)&CSW
        |¬AR(19)&(CPLE|CPAE)&CSH
        |¬AR(35)&CPHE)
        THEN LT= '1'B;

    IF RCOMFF|REQS THEN EQ='1'B;
    IF ¬LR(36)&AUCG THEN EQ='1'B;
    IF (CPLE|CPAE|TZE) THEN DO;
        ZERO='0'B;
        IF (CSB|CSH|CSW) THEN DO I=28 TO 35;
            ZERO=ZERO|AR(I);
        END;
        IF (CSH|CSW) THEN DO I=19 TO 26;
            ZERO=ZERO|AR(I);
        END;
        IF CSW THEN DO I=1 TO 17 WHILE (I¬=9);
            ZERO=ZERO|AR(I);
        END;
        IF ¬ZERO THEN EQ='1'B;
    END;
END COMPARE;
```

/*

4.14.3 COMPARE Variable List

Registers and Counters (External Variables):

<u>Simulation Name</u>	<u>Description</u>
AR(i)	Arithmetic Register - bit i
DR (i)	Distribution Register - bit i
LR (i)	Logic Register - bit i

Control Signals (1 bit parameters):

<u>Simulation Name</u>	<u>Description</u>
AUCG	AU flags to LT, GT, EQ from LR, DR
CPAE	Enable algebraic compare
CPHE	Enable Hollerith compare
CPLE	Enable logical compare
CSB	Cell size is byte
CSH	Cell size is halfword
CSW	Cell size is word
EQ	Equal flip-flop indicator
GT	Greater than FF indicator
LT	Less than FF indicator
RCOMFF	Reset compare FF's - EQ, LT, GT
REQS	Reset EQ FF
RGLE	Reset GT and LT flip-flops
SGTS	Set GT
SLTS	Set LT
TZE	Enable test zero

5. Control Sequences

The procedures in this section are simulations of some of the control sequences of the TP--namely the Memory Sequences and the Operand Stack sequences.

A control sequence is called via a call to CCP. For example, to call the control sequence SCRMØD, as well as turn on some signals, one would use the instruction

```
CALL CCP('ØST1', 'SCSCS, CCSBE.', 'SCRMØD', RRET);
```

The first parameter is a label--the task name. The second parameter is a character string of the signals to be activated. This is followed by the control sequence name 'SCRMØD'. The last parameter is a bit which is set to 1 by SCRMØD if it makes a regular return, to 0 if SCRMØD makes an interrupt return.

The control sequences themselves consist almost exclusively of decision (tests of signals for 0 or 1) and calls to CP or CCP. The calls indicate which signals are to be activated, and in the case of a call to CCP, to which procedure control is to be transferred. CP and CCP do the rest.

With the exception of the sequence QUPDATE, the calls for these procedures are all contained within CCP.

The simulation of the control sequences is, for the most part, very straightforward. Debugging them is a difficult matter however, since a mistake may be due to a problem in any of the simulating routines.

5.1 Memory Sequences

The procedures in this section simulate the control sequences regulating accesses to the memory of the TP.

Complete descriptions of these sequences are in the TP Manual Section 4.2.3 (2).

5.1.1.1 MEMDIR, MEMMOD, IMMAC Procedure

MEMDIR is a procedure which has an extra entry point--MEMMOD.

MEMDIR and MEMMOD simulate the MEMORY DIRECT ACCESS and MEMORY MODIFIED ADDRESS sequences, respectively, which (together) form the INITIAL ADDRESS CONSTRUCTION control logic. A complete description of this sequence is found in the TP manual (2) Section 4.2.3.2.

The IMMAC label and what follows it, are simulating the IMMEDIATE ADDRESS sequence, which is indeed a part of the main memory sequence. It is described in Section 4.2.3.3 of the TP manual (2).

There are several things to notice about the simulation of these sequences. One point is the simple call to QUPDATE after MT4. A simple call was used, rather than a calling control point, because QUPDATE is logically a part of the memory INITIAL ADDRESS CONSTRUCTION sequence. It was implemented as a separate procedure only to make things clearer. This is the only place from which QUPDATE is called--so it can be viewed as a part of the sequence.

MT18 is a calling control point to PMAC which turns on no signals. After returning from PMAC, its return bit PMACRT is checked. PMACRT=0 if this was an interrupt return; in this case--a memory sequence interrupt is initiated. The same is true for MT8.

The check for memory failure between MT12 and MT13 which appears in the flow chart is not implemented in the simulator because a memory failure has not yet been simulated.

Since the Memory Interrupt Sequence has not been simulated, the sequence simply returns with the interrupt signal (in case of an interrupt) rather than calling the interrupt sequence.

```

// EXEC PL1.
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
MEMDIR:  PROC(TGIM,ALGNER,DA,NYET,NT,PART,ILAC,BOVI,LOC,MSTOR,ARF33,
  TRAP,SQTOV,CYCOV,ARARFG,LNK,VAL,REG,RRET);
  DCL(LNK,VAL,REG) BIT(1);
    DCL IR(36) BIT(1) EXTERNAL;
  DCL PRNTSIGS ENTRY(CHAR(180),BIT(1));
    DCL(TGIM,ALGNER,DA,TRAP,NT,PART,ILAC,BOVI,LOC,MSTOR,ARF33,
      ARARFG,RRET,PMACRT) BIT(1);
    DCL (SQTOV,CYCOV) BIT(1);
    DCL CP ENTRY(CHAR(50),CHAR(180));
    DCL CCP ENTRY(CHAR(50),CHAR(180),CHAR(20),BIT(1));
      IF TGIM THEN GO TO IMMACC;
      ELSE GO TO MT1;
MT1:  CALL CP('MT1','PRPG,BDRG,TNBG,SNSG|WRPTRS,WRREGS.');
```

```

MEMMOD: ENTRY(TGIM,ALGNER,DA,NYET,NT,PART,ILAC,BOVI,LOC,MSTOR,
  ARF33,TRAP,SQTOV,CYCOV,ARARFG,LNK,VAL,REG,RRET);
MT2:  CALL CP('MT2','DRPG,PSNRW15,ALCKE,PLC2|WRPTRS,WRREGS.');
```

```

  IF ALGNER THEN GO TO MSIRRET;
MT3:  CALL CP('MT3','SNRDE,BRGS|WRREGS.');
```

```

  IF DA THEN GO TO MT4;
  ELSE GO TO MT5;
MT4:  CALL CP('MT4','BRGS,SNRDE,BRSBPG,ADDE,ADDN(0),BDCKE,AMCKE,
  ||'ARARFG|WRREGS.');
```

```

  CALL QUPDATE(SQTOV,CYCOV,RRET);
  CALL BRROUT;
  IF TRAP THEN GO TO MT17W;
  IF NT & PART THEN GO TO MT17X;
  IF NI & -PART THEN GO TO MT17Y;
  IF ILAC THEN GO TO MT17V;
  IF -ILAC & BOVI THEN GO TO MT17Z;
  IF -BOVI & PART THEN GO TO MT18;
  IF -PART THEN GO TO CHKLOC;
MT17V: CALL CP('MT17V','SILACS.');
```

```

  GO TO MSIRRET;
MT17W: CALL CP('MT17W','SGTRPS.');
```

```

  GO TO MSIRRET;
MT17X: CALL CP('MT17X','SGMNTS.');
```

```

  GO TO MSIRRET;
MT17Y: CALL CP('MT17Y','SGDNTS.');
```

```

  GO TO MSIRRET;
MT17Z: CALL CP('MT17Z','SGBOVS.');
```

```

  GO TO MSIRRET;
MT18:  CALL CCP('MT18','.','PMAC',PMACRT);
  IF -PMACRT THEN GO TO MSIRRET;
CHKLOC: IF LOC|MSTOR THEN GO TO MEMRRT;
  ELSE CALL CCP('CHKLOC','.','READAC',RRET);
  IF -RRET THEN GO TO MSIRRET;
  ELSE GO TO MEMRRT;
MT5:  CALL CP('MT5','PSNPRBG,PRPG,BDRG,SNRDE,BRGS|WRPTRS,WRREGS.');
```

```

MT6:  CALL CP('MT6','BRGS,BRS(0),BRSBPG,ADDE,ADDN(0),
  ||'BDCKE,AMCKE,ARARFG.');
```

```

  IF TRAP THEN GO TO MT7X;
  IF NT & PART THEN GO TO MT7Y;
  IF NT & -PART THEN GO TO MT7Z;
  IF BOVI THEN GO TO MT7W;

```

```

        IF PART THEN GO TO MT8;
            ELSE GO TO MT10;
MT7W:   CALL CP('MT7W','STBOVS.');
```

GO TO MSIRRET;

```

MT7X:   CALL CP('MT7X','STTRPS.');
```

GO TO MSIRRET;

```

MT7Y:   CALL CP('MT7Y','STMNTS.');
```

GO TO MSIRRET;

```

MT7Z:   CALL CP('MT7Z','STDNTS.');
```

GO TO MSIRRET;

```

MT8:    CALL CCP('MT8','SNAMT.','PMAC',PMACRT);
```

IF ~PMACRT THEN GO TO MSIRRET;

```

MT10:   CALL CP('MT10','ARPG,CDRE,RWBGE|WRREGS.');
```

```

MT11:   CALL CP('MT11','ENMCBE.');
```

IF ARF33 THEN CALL MDAT2;

ELSE CALL MDAT1;

```

MT12:   CALL CP('MT12','CTPG,BDRG|WRREGS.');
```

```

MT13:   CALL CP('MT13','DRPG,BRGS,DBBRSG,WRBRE|WRREGS.');
```

```

MT14:   CALL CP('MT14','SNRDE,PSNPRBG,PRPG,BDRG|WRREGS,WRPTRS.');
```

```

MT15:   CALL CP('MT15','DRPG,ASRWE|WRREGS.');
```

```

MT16:   CALL CP('MT16','PSNRR15,PSNPRBG,PRPG,BDRG|WRREGS,WRPTRS.');
```

GO TO MT3;

```

MSIRRET: RRET='0'B;
```

PUT SKIP LIST ('MEMORY SEQUENCE FINISHED WITH INTERRUPT');

RETURN;

```

IMMACC: LNK=~IR(7)&~IR(8);
```

VAL=~IR(7)&IR(8);

REG=IR(7)&~IR(8);

SN=IR(7)&IR(8);

IF MSTORE THEN GO TO MT21;

```

MT20:   IF LNK THEN CALL CP ('MT20,MT20Y','PRPG,BLRG,TNBG|IBC(2),'
```

||'IBC(3)|WRREGS,WRPTRS.');

ELSE IF VAL THEN CALL CP('MT20,MT20X','PRPG,BLRG,TNBG|

||'IBC(2),IBC(3),PLC2|

||'WRREGS,WRPTRS.');

ELSE IF SN THEN CALL CP('MT20,MT20Z','PRPG,BLRG,'

||'TNBG|PSNPRBG,SNRG,SNRDE|

||'WRREGS,WRPTRS.');

ELSE IF REG THEN CALL CP('MT20','PRPG,BLRG,'

||'TNBG|WRREGS,'

||'WRPTRS.');

GO TO MEMRRT;

```

MT21:   CALL PRNTSIGS('LDREGE.','1'B);
```

```

MT22:   IF LNK THEN CALL CP ('MT22,MT22Z','LRPG,DBPRG,TNBG|FLPRN,'
```

||'WPRLE|WRREGS,WRPTRS.');

ELSE IF VAL THEN CALL CP('MT22,MT22Y','LRPG,DBPRG,TNBG|

||'FLPRN,WRPRVE,PLC2|WRREGS,'

||'WRPTRS.');

ELSE IF REG THEN CALL CP ('MT22,MT22X','LRPG,DBPRG,'

||'TNBG|WPRLE,WRPRVE|

||'WRREGS,WRPTRS.');

ELSE MT23: IF SN THEN CALL CP('MT23','TNBG,SNRG,SNWTE,'

||'LRPG|WRREGS,WRPTRS.');

```

MEMRRT: RRET='1'B;
```

PUT SKIP LIST('MEMORY SEQUENCE FINISHED WITH REGULAR RETURN');

END MEMDIR;

/*

5.1.2 PMAC Procedure

The PMAC procedure simulates the PARTITIONED MODE ADDRESS CONVERSION sequence of the memory, which is used to access the page map in a partitioned mode access and construct the actual physical core address bits connected with the page map entry to be sure the access is a valid one. A complete description of this sequence is in the TP manual (2), Section 4.2.3.5.

The PMAC procedure is quite similar to the flow chart of the sequence. There are two points to notice however. The first is the combining of MT37 and MT38. This was done as a simulation of the complex structure of these control points. In essence, MT37 is still on during MT38.

The second point to notice is the calling of MDAT1 or MDAT2 between MT39 and MT40. In the logic, there is a delay before MT40 while the data returns from memory. In the simulator, the call to MDAT1 or MDAT2 causes the data in the Memory Buffer (MEMBUF) to be gated into the cable terminators (CT).

```

// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
  PMAC:  PROC(MSTOR,SNAMT,ARF33,PMACRT);
         DCL DRB(36) BIT(1) EXTERNAL;
         DCL (MSTOR,SNAMT,ARF33,PMACRT) BIT(1);
         DCL CP ENTRY(CHAR(50),CHAR(180));
         /*THE PARTITIONED MODE ACCESS CONIERSION SEQUENCE IS USED TO ACCESS
           THE PAGE MAP IN A PARTITIONED MODE ACCESS AND CONSTRUCT THE
           ACTUAL PHYSICAL CORE ADDRESS WHICH WILL BE NEEDED TO ACCESS CORE
           AT THE SAME TIME IT CHECKS THE ACCESS BITS CONNECTED WITH THE
           PAGE MAP ENTRY TO BE SURE THE ACCESS IS A VALID ONE.*/
MT32:  CALL CP('MT32','LRPG,BARG|WRREGS.');//*LOAD AR FROM LR */
MT33:  CALL CP('MT33','LSE,RREPS|WRREGS.');//*RESET PMAREP FF */
/* M1E IS USED TO INHIBIT THE LOWEST ORDER DATA BIT OF THE DR */
MT34:  CALL CP('MT34','PLC3,M1E,IBC(0),IBC(1),LRPG,BDRG|WRREGS.');//
       /* M1E IS USED TO INHIBIT THE LOWEST ORDER BIT IN PERMUTER*/
MT35:  CALL CP('MT35','ARPG,BLRG|WRREGS.');//
MT36:  CALL CP('MT36','SNRDE,BRGS,RDBRE,BRSBPG,ADDE,ADDN(0)|WRREGS.');//
       /*CALL EXCHANGE NET. NOTE THAT RWBGE IS NOT TURNED
       ON BECAUSE ALL ACCESSES TO THE PAGE MAPS ARE READ ACCESSES.
       IN ADDITION IF IT WERE TURNED ON, THE WRONG RESULTS
       MIGHT BE OBTAINED IF MSTOR=1. */
MT37: MT38: CALL CP('MT37,MT38','ARPG,CDRE,RWBGE,PGMPAC|'
                   ||'ENMCBE|WRREGS.');//
MT39:  CALL CP('MT39','LRPG,BARG|WRREGS.');//
       /*WAIT FOR DATA TO RETURN*/
       IF ARF33 THEN CALL MDAT2;
       ELSE CALL MDAT1;
MT40:  CALL CP('MT40','LJOPE,CTPG,BDRG,IBC(2),IBC(3)|WRREGS.');//
       /* IF MEMORY FAILURE THEN GO TO INTERRUPT SEQ*/
MT41:  IF ~DRB(9)&~DRB(18) THEN DO;
       IF SNAMT THEN GO TO MT41V;
       ELSE GO TO MT41W;
       END;
       ELSE IF MSTOR&DRB(9)&~DRB(18) THEN GO TO MT41Z;
       ELSE IF DRB(18)&~DRB(9) THEN DO;
       IF SNAMT THEN GO TO MT41X;
       ELSE GO TO MT41Y;
       END;
       ELSE GO TO MT42;
MT41V: CALL CP('MT41V','STTRPS.');//
       GO TO IRRETURN;
MT41W: CALL CP('MT41W','SGTRPS.');//
       GO TO IRRETURN;
MT41X: CALL CP('MT41X','STPNTS.');//
       GO TO IRRETURN;
MT41Y: CALL CP('MT41Y','SGPNTS.');//
       GO TO IRRETURN;
MT41Z: CALL CP('MT41Z','SILACS.');//
       GO TO IRRETURN;
MT42:  CALL CP('MT42','PLC3,PSNRR15,PSNPRBG,PRPG,BLRG|WRREGS,'
                   ||'WRPTRS.');//
MT43:  CALL CP('MT43','DLRG,DLRN(2)|WRREGS.');//
MT44:  CALL CP('MT44','ARPG,BDRG|WRREGS.');//
MT45:  CALL CP('MT45','LRPG,BARG,PLC3|WRREGS.');//

```

```
MT46:  CALL CP('MT46','DLRG|WRREGS.');
```

RRETURN: PMACRT='1'B;
 PUT SKIP LIST('PMAC FINISHED WITH REGULAR RETURN');
 GO TO PEND;

IRRETURN: PMACRT='0'B;
 PUT SKIP LIST('PMAC FINISHED WITH INTERRUPT');

PEND: END PMAC;

/*

5.1.3 READAC Procedure

The READAC procedure simulates the memory READ ACCESS control sequence, which is of course used to read from the memory. A complete description of the READ ACCESS sequence is found in the TP manual (2) Section 4.2.3.6.

There are several places in the procedure which seem to differ from the flow chart representation. One is the combination of MT50 and MT51. This is done to simulate the action of these complex control points. What is desired is to have MT51 activated when MT50 is finished but not yet off. In the simulation, the control points must be combined to achieve this effect, and since the ENMCBE signal is not used until the end of SIMCALL, the combined control points work properly.

After MT51, if ARF33=1, MDAT2 is called to access the second word of the double word addressed. If ARF33=0, the first word is accessed.

After MT54, MDAT2 is called if CSD=1. CSD and ARF33 will never both be on. If CSD is not on, then MT55 is skipped, since it loads the data which would have been accessed into the DR.

Since the memory unit failure has not been simulated, the test for ACFAIL has been omitted from this procedure, and hence there is no interrupt return.

```

// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
READAC:  PROC(CSD,ARF33,RRET);
          DCL (CSD,ARF33,RRET) BIT(1);
          DCL CP ENTRY(CHAR(50),CHAR(180));
          DCL CCP ENTRY(CHAR(50),CHAR(180),CHAR(20),BIT(1));
MT50: MT51: CALL CP('MT50,MT51','ARPG,CDRE,RWBGE|'
                  ||'ENMCBE|WRREGS. ');
          IF ARF33 THEN CALL MDAT2;
          ELSE CALL MDAT1;
MT54:  CALL CP('MT54','CTPG,BLRG|WRREGS. ');
          IF -CSD THEN GO TO MT56;
          CALL MDAT2;
MT55:  CALL CP('MT55','CTPG,BDRG. ');
MT56:  CALL CP('MT56','XNREL. ');
/* IF THERE IS A MEMORY UNIT FAILURE AT THIS POINT, READAC WOULD
   TRANSFER TO THE MEMORY INTERRUPT SEQUENCE.  HOWEVER, THE HARDWARE
   FAILURE OF THE MEMORY UNIT HAS NOT BEEN SIMULATED AT THIS TIME*/
READRT: RRET='1'B;
          PUT SKIP LIST('READAC FINISHED WITH REGULAR RETURN');
DONE:   END READAC;

```


5.1.4 WRAC Procedure

The WRAC procedure simulates the WRITE ACCESS control sequence of the memory, which is used to write data into memory. A description of the WRITE ACCESS logic is found in the TP manual (2) Section 4.2.3.7.

Note that MT71-MT72, MT74-MT75, and MT76-MT77 are complex control points and hence are combined for the simulation.

After MT71-MT72, the flow chart shows that if ARF33 is on, "wait for 1st data time slot to pass". This is simulated by the NULL calling control point MT73 which also prints out the above message.

After MT74-75, TII1 or TII2 is called to write the contents into the first or the second word of the double word addressed--according to the status of ARF33. If CSD is on, then the second word is accessed by activating MT76-MT77 and calling TII2.

Note the use of the NULL control point MT79--used as a delay. This has no action in the simulator except to cause a printing of MT79 on the output--which is desirable for flow-of-control diagnostics.

Since a memory unit failure has not been simulated, the check for ACFAIL has been omitted.

```

// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
WRAC:  PROC(DREDY,CSD,ARF33,RRET);
        DCL CP ENTRY(CHAR(50),CHAR(180));
        DCL ARF33 BIT(1);
        DCL (DREDY,CSD,RRET,TII0,MURT,XNRET)BIT(1);
        DCL CCP ENTRY(CHAR(50),CHAR(180),CHAR(20),BIT(1));
MT71:  MT72:  CALL CP('MT71,MT72','ARPG,CDRE,RWBGF|ENMCBE|WRREGS.');
```

IF ARF33 THEN GO TO MT73;
ELSE GO TO MT74;

```

MT73:  CALL CCP('MT73-WAIT FOR FIRST TIME SLOT TO PASS',
               '','',RRET);
MT74:  MT75:  CALL CP('MT74,MT75','LRPG,CDRE|TII|WRREGS,WRPTRS.');
```

IF ARF33 THEN CALL TII2;
ELSE CALL TII1;
IF CSD THEN GO TO MT76;
ELSE GO TO MT78;

```

MT76:  MT77:  CALL CP('MT76,MT77','DRPG,CDRE|TII|');
        CALL TII2;
MT78:  CALL CP('MT78','XNREL|WRMEM.');
```

MT79: CALL CP('MT79','');

/* IF THERE IS A MEMORY UNIT FAILURE AT THIS POINT, WRAC WOULD
TRANSFER TO THE MEMORY INTERRUPT SEQUENCE. HOWEVER, THE HARDWARE FAILU
OF THE MEMORY INIT HAS NOT BEEN SIMULATED AT THIS TIME*/

```

        RRET='1'B;
        PUT SKIP LIST('WRAC FINISHED WITH REGULAR RETURN');
```

END WRAC;

/*

5.1.5 QUPDATE Procedure

The QUPDATE procedure simulates the QUEUE COUNTER UPDATE control sequence, which is used to update the status of the queue counters for the hardware base registers in the TP each time one of these registers is used. A complete description of this sequence is in the TP manual (2) Section 4.1.3.4.

The procedure itself follows exactly from the flowchart for the sequence. Note the implementation of the null control points MT26, MT29.

```

// EXEC PL1
//PL1.SYSPUNCH DD.SYSOUT=B
//PL1.SYSIN DD *
QUPDATE: PROC(SQTOV,CYCOV,RRET);
        DCL (SQTOV,CYCOV,RRET)BIT(1);
        DCL CP ENTRY(CHAR(50),CHAR(180));
        PUT EDIT('QUPDATE SEQUENCE ENTERED')(SKIP,X(20),A);
MT24:   CALL CP('MT24','RCYCTS|WRREGS. ');
CHKQOV: IF SQTOV THEN GO TO MT27;
MT25:   CALL CP('MT25','QCNTC,CYCTE. ');
MT26:   CALL CP('MT26','. ');
        GO TO CHKQOV;
MT27:   CALL CP('MT27','XQCNTC. ');
CHKCOV: IF CYCOV THEN GO TO MT30;
MT28:   CALL CP('MT28','QCNTC,CYCTE. ');
MT29:   CALL CP('MT29','. ');
        GO TO CHKCOV;
MT30:   CALL CP('MT30','RQOVFS. ');
MT31:   CALL CP('MT31','RSCTRS. ');
        RRET='1'B;
        PUT EDIT('QUPDATE SEQUENCE FINISHED')(SKIP,X(20),A);
END QUPDATE;
/*

```

5.2 Operand Stack Sequences

The procedures in this section simulate the basic ØS sequences: ØSENTRY, SCRMØD, ØSREAD, and ØSWRITE. These sequences directly control the manipulation of the ØS hardware. Complete descriptions of the sequences, with their flowcharts, are found in the TP Manual (2) Section 4.5.1.

5.2.1 ØSENTRY, ØVENTRY Procedure

This procedure has two entry points, ØSENTRY and ØVENTRY, which simulate the OS ENTRY sequence which is used to check for potential overflow or underflow in the Operand Stack. A complete description of the sequence is found in the TP Manual (2) Sections 4.5.1.1 and 4.5.1.2.

In the TP manual there are two flow charts which show the operations of the sequence in two cases - when a push is being done (in which case there is a possibility of overflow) and when a pop is being done (in which case there is a possibility of underflow.) These two flowcharts are implemented in the program by the two sections of code ØVLØØP and UNLØØP.

Note that UNLØØP heads a "BEGIN" group so that the same control point names can be used as labels. The identical control point names refer to the same control point, but different conditional subtasks. It is easier to think of the sequence as having two parts, rather than one part with a conditional branch at every control point.

UNLØØP, if entered, makes a read access to memory and decrements the ØSTR, while ØVLØØP makes a write access and increments ØSTR. In addition, if ØVENTRY was the original entry point, ØVLØØP checks to see if the SCR = ØSTR and if so returns.

At the end of the procedure, ØSØUT and MEMØUT are called to print out the contents of the operand stack and the memory.

```

// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
  OSENTRY: PROC(WRAP,AOV,A5OV,CS2C,OV1,OV2,DUP,POP,UF1,UF2,
               OSRRT) RECURSIVE;
    DCL(WRAP,AOV,A5OV,CS2C,OV1,OV2,DUP,POP,UF1,UF2,OSRRT,
         OVENSTRT,MEMRRT,UNSTRT) BIT(1);
    DCL (SCR(5),OSTR(2)) BIT(1) EXTERNAL;
    DCL CP ENTRY(CHAR(50),CHAR(180));
    DCL CCP ENTRY(CHAR(50),CHAR(180),CHAR(20),BIT(1));
/* OSENTRY INPUT DECISION LOGIC*/
  OVENSTRT='0'B;
  IF POP&UF2&CS2C
    |POP&UF1&~CS2C THEN GO TO UNLOOP;
  IF ~POP&OV2&CS2C
    |~POP&OV1&~CS2C
      | POP&~UF1&CS2C&DUP&OV2
      | POP&~UF1&~CS2C&DUP&OV1 THEN GO TO OVLOOP;
  IF POP&~UF2&CS2C&~DUP
    | POP&~UF1&~CS2C&~DUP
    | POP&~UF2&CS2C&DUP&~OV2
    | POP&~UF1&~CS2C&DUP&~OV1
    |~POP&~OV2&CS2C
    |~POP&~OV1&~CS2C THEN GO TO OSRRET;
  OVENTRY: ENTRY(WRAP,AOV,A5OV,CS2C,OV1,OV2,DUP,POP,UF1,UF2,OSRRT);
  OVENSTRT='1'B;
  OVLOOP:
OST1: CALL CP('OST1,OST1W','NPR13S,PRPG,BDRG,SNSG,OSTPG,SCSCS|'
             ||'SMSRS|WRREGS,WRPTRS.');
```

IF ~WRAP THEN GO TO OST7;

```

OST3: CALL CP('OST3','M32E,ADDE,CINJ,LRSE|WRREGS.');
```

IF AOV THEN

```

OST4: CALL CP('OST4,OST4X','ARPG,BDRG|SOSBES|WRREGS.');
```

ELSE CALL CP('OST4','ARPG,BDRG|WRREGS.');

```

OST7: CALL CCP('OST7','CCSDE.','MEMMOD',MEMRRT);
  IF MEMRRT='0'B THEN GO TO OSIRRET;
OST8: CALL CP('OST8','OTAG,ADD5E,T1R2G.');
```

```

OST10: CALL CP('OST10X,OST10','OSPTG|BLRG|WROPSTK,WRREGS.');
```

```

OST11: CALL CP('OST11,OST1Y','ADD5E,T1R2G|OTAG,KBG,X2,OSCGE,OSCG8E.');
```

```

OST13: CALL CP('OST10X,OST13','OSPTG|BDRG|WRREGS,WROPSTK.');
```

CALL CCP('OST13','.', 'WRAC',MEMRRT);

IF OVENSTRT THEN IF((SCR(1)=OSTR(1))&(SCR(2)=OSTR(2)))

THEN GO TO OSRRET;

```

OST14: CALL CP('OST1Y','X2,OSCGE,OSCG8E,OTAG,KBG,'
             ||'ADD5E.');
```

```

OST15: IF A5OV THEN CALL CP('OST15,OST15X','T1STG,ROSF|RWRPS.');
```

ELSE CALL CP('OST15','T1STG,ROSF.');

IF OVENSTRT THEN GO TO OVLOOP;

```

OST16: CALL CP('OST16','SCSSS.');
```

PUT SKIP LIST('WAIT FOR OV-UF LOGIC');

IF CS2C&OV1 THEN GO TO OVLOOP;

ELSE GO TO OSRRET;

```

UNLOOP: BEGIN;
  UNSTRT='1'B;
OST1: CALL CP('OST1,OST1X,OST1Y,OST1Z','NPR13S,PRPG,BDRG,SNSG,OSTPG,'
             ||'SCSCS|SE,CINJ|X2,OSCGE,OSCG8E,OTAG,KBG,ADD5E|'
             ||'RMSRS|WRPTRS,WRREGS,WROPSTK.');
```

```

OST2:  CALL CP('OST2','T1STG,RDSF,T1R2G.');
```

IF ~(A50V|WRAP) THEN GO TO OST7;

```

OST3:  CALL CP('OST3,OST3X','M32E,ADDE,CINJ,LRSE|SWRPS|WRREGS.');
```

```

OST4:  IF ADV THEN CALL CP('OST4,OST4X','ARPG,BDRG|SOSBES|WRREGS.');
```

ELSE CALL CP('OST4','ARPG,BDRG|WRREGS.');

```

OST7:  CALL CCP('OST7','CCSDE.','MEMMOD',MEMRRT);
```

IF MEMRRT='0'B THEN GO TO OST17;

```

OST9:  CALL CP('OST9,OST9X','LRPG|DBDSG,RSDE,DBRJ|WRREGS,'
```

||'WROPSTK.');

```

OST11: CALL CP('OST11,OST1Y','SE,T1R2G|X2,OSCGE,OSCG8E,OTAG,KBG,'
```

||'ADD5E.');

```

OST12: CALL CP('OST12,OST12X','DRPG|DBDSG,RSDE,DBRJ|WRREGS,'
```

||'WROPSTK.');

```

OST16: CALL CP('OST16','SCSSS.');
```

PUT SKIP LIST('WAIT FOR OV-UF LOGIC.');

IF CS2C&UF1 THEN GO TO UNLOOP;

ELSE GO TO OSRRET;

```

OST17: CALL CP('OST17,OST1Y','RSFS|X2,OSCGE,OTAG,KBG,ADD5E.');
```

```

OST18: IF A50V THEN CALL CP('OST18','RWRPS.');
```

GO TO OSIRRET;

END UNLOOP;

```

OSIRRET: OSRRT='0'B;
```

PUT SKIP LIST('OSENTRY FINISHED WITH INTERRUPT');

GO TO OSEND;

```

OSRRET: OSRRT='1'B;
```

PUT SKIP LIST('OSENTRY FINISHED WITH REGULAR RETURN');

```

OSEND: CALL OSOUT(OV1,OV2,UF1,UF2);
```

CALL MEMOUT;

END OSENTRY;

/*

5.2.2 SCRMØD Procedure

The SCRMØD procedure simulates the SCR MØD control sequence, which is used to change the value of the SCR and to provide corrections to PR#13 in the case of SCR overflow and underflow. A complete description of the SCR MØD sequences is found in the TP Manual (2) Section 4.5.1.3.

The simulation of the SCRMØD sequence is very straightforward. The procedure is in one-to-one correspondence with the SCRMØD flow chart found in the above reference.

SCRMØD is called through calling control points in ØSWRITE and ØSREAD as well as other sequences not yet simulated.

```

// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
SCRMOD: PROC(SE,A50V,WAIT,ADV,RRET);
        DCL (SE,A50V,WAIT,ADV,RRET) BIT(1);
        DCL CP ENTRY(CHAR(50),CHAR(180));

OST20:
        IF SE THEN CALL CP('OST20,OST20X','RDSF,KBG,SCAG,ADD5E|CINJ.');
```

ELSE CALL CP('OST20','RDSF,KBG,SCAG,ADD5E.');

```

OST21:
        IF A50V THEN DO;
                CALL CP('OST21','CKOFE,T1SCG|WROPSTK.');
```

GO TO SCRRETURN;

```

        END;
        IF WAIT THEN CALL CP('OST21,OST21Y',
                'CKOFE,T1SCG|SDCPRS|WROPSTK.');
```

ELSE DO;

```

                CALL CP('OST21,OST21X,OST21Z',
                'CKOFE,T1SCG|NPR13S,PRPG,BDRG|WROPSTK.');
```

```

OST22:
        IF SE THEN CALL CP('OST22,OST22X',
                'ADDE,ADDN(0),ADDN(1)|RWRPS,M32E,'
                || 'CINJ|WRREGS.');
```

ELSE CALL CP('OST22,OST22Y','ADDE,ADDN(0),ADDN(1)'

```

                || '|SWRPS,P32E|WRREGS.');
```

```

OST23:
        IF ADV THEN CALL CP('OST23,OST23X','FLPRN,WRPRVE,ARPG,'
                || 'DBRPG,NPR13S|SOSBES|WRPTRS,'
                || 'WRREGS.');
```

ELSE CALL CP('OST23','FLPRN,WRPRVE,ARPG,DBPRG,'

```

                || 'NPR13S|WRPTRS,WRREGS.');
```

```

        END;
        SCRRETURN: RRET='1'B;
        PUT SKIP LIST ('SCRMOD FINISHED WITH REGULAR RETURN');
        END SCRMOD;
/*
```

5.2.3 ØSREAD Procedure

The ØSREAD procedure simulates the ØSREAD sequence which is used to read data from the ØS hardware registers into the LR (and in the case of a double word cell into the DR.) A complete description of the ØSREAD sequence, with its flow-chart is found in the TP manual (2) Section 4.5.1.4.

One thing to note in ØSREAD is the operation of ØST32; if NWTØS is equal to one, the task must remain on until the ØSREAD sequence has returned to the calling logic. In order to simulate this case, ØST32 does not call CP, instead, it calls PRNTSIGS which prints out the task name and task signals, and calls EXTRACT to set the signals to one. It does not then reset them to zero as CP would have. PRINTSIGS is usually called by CP to do the above, but afterwards, CP calls EXTRACT to reset the signals to zero--which is exactly what is not wanted if NWTØS is on.

Note also that ØSREAD calls SCRMØD from the calling control points ØST31 and ØST34.

ØSREAD is called from the ØS instruction sequences which are not yet simulated.

```

// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
OSREAD: PROC(STOR,NEG,NWTOS,CSD,OSRDRT);
        DCL(STOR,NEG,NWTOS,CSD,OSRDRT,SCRRT)BIT(1);
        DCL CP ENTRY(CHAR(50),CHAR(180));
        DCL CCP ENTRY(CHAR(50),CHAR(180),CHAR(20),BIT(1));
        DCL PRNTSIGS ENTRY(CHAR(50),CHAR(180));
OST30:  CALL CP('OST30','RDCPRS.');
```

OST31: IF STOR THEN CALL CCP('OST31,OST31X','SE|WAIT.',
 'SCRMOD',SCRRT);

 ELSE CALL CCP('OST31','SE.','SCRMOD',SCRRT);

OST32: IF NEG&NWTOS THEN DO; /*TURN ON SIGNALS ONLY */
 CALL PRNTSIGS('OST32,OST32Y','SCT2G|'
 ||'OSPFG|WRREGS,WROPSTK.');

 GO TO READRET;
 END;

IF NEG&-NWTOS THEN CALL CP('OST32,OST32Y,OST32Z','SCT2G|'
 ||'OSPFG|BLRG|WRREGS,WROPSTK.');

IF -NEG&NWTOS THEN DO; /* TURN ON SIGNALS ONLY*/
 CALL PRNTSIGS('OST32,OST32X','SCT2G|'
 ||'OSPTG|WRREGS,WROPSTK.');

 GO TO READRET;
 END;

IF -NEG&-NWTOS THEN CALL CP('OST32,OST32X,OST32Z','SCT2G|'
 ||'OSPTG|BLRG|WRREGS,'
 ||'WROPSTK.');

OST33: IF -CSD THEN GO TO OST34;
IF NEG THEN CALL CP('OST33,OST33Y,OST41X','BDRG|OSPFG|'
 ||'SCAG,KBG,OSCGE,OSCG4E,T1R2G,X2,ADD5E|'
 ||'WRREGS,WROPSTK.');

 ELSE CALL CP('OST33,OST33X,OST41X','BDRG|OSPTG|'
 ||'SCAG,KBG,OSCGE,OSCG4E,T1R2G,X2,ADD5F|'
 ||'WRREGS,WROPSTK.');

OST34: IF STOR THEN CALL CCP('OST34','WAIT.','SCRMOD',SCRRT);

READRET :OSRDRT='1'B;
 PUT SKIP LIST('OSREAD HAS COMPLETED WITH REGULAR RETURN');

END OSREAD;

/*

5.2.4 ØSWRITE Procedure

The ØSWRITE procedure simulates the ØS WRITE control sequence, which is used to write data into the ØS hardware registers. A complete description of the ØS WRITE sequence is found in the TP Manual (2) Section 4.5.1.5.

Since the entire sequence consists of only three control points, the procedure is very straightforward and corresponds exactly with the ØSWRITE flow chart.

Note that in ØST42 a "null" calling control point is used. No new signals are to be turned on, but control is to be transferred to SCRMØD.

The ØSWRITE sequence is called from Operand Stack Instruction Sequences which are not yet simulated. However, the procedure has been tested to some extent.

```

// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
OSWRITE: PROC(CSD,OSWTRT);
        DCL (CSD,OSWTRT,SCRRT)BIT(1);
        DCL CP ENTRY(CHAR(50),CHAR(180));
        DCL CCP ENTRY(CHAR(50),CHAR(180),CHAR(20),BIT(1));
OST40:
        CALL CP('OST40','SCT2G,LRPG,OSSS,RSDE,DBDSG|WRREGS,WROPSTK.');
```

OST41:

```

        IF CSD THEN CALL CP('OST41,OST41X','RSES,OSSS,DRPG,DBDSG,RSDE|SCAG,'
        ||'KRG,OSCGE,OSCG4E,T1R2G,X2,ADD5E|WRREGS,WROPSTK.');
```

OST42:

```

        CALL CCP('OST42','.', 'SCRMOD',SCRRT);
        OSWTRT='1'B;
        PUT SKIP LIST('OSWRITE FINISHED WITH REGULAR RETURN.');
```

END OSWRITE;

/*

6. Environmental Procedures

The environmental procedures are so named because they are not part of the TP logic, nor are they part of the control sequences. They are necessary, however, to test out the control sequences.

The procedures are "fudged" short simulations of an access to the exchange net, and logic necessary to simulate the memory itself.

These procedures are in no way an exact simulation of the logic involved. They are merely very high level simulations of what results from much low level logic.

Some of these procedures are called from SIMCALL. Others are called directly from a control sequence.

6.1 MEMLØG Procedure

6.1.1 MEMLØG Description

The MEMLØG procedure is used to set flip-flops used by the memory sequences. These flip-flops are set according to their equations if an alignment check is to be made (AMCKE = 1).

MEMLØG is called from SIMCALL right before PER-ØUT since the equations involved use the flags of the DBB.

6.1.2 MEMLOG Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
MEMLOG: PROC(AMCKE,PART,TRAP,NT,IAC,MSTOR);
        DCL (AMCKE,PART,TRAP,NT,IAC,MSTOR) BIT(1);
        DCL DBB (36) BIT(1) EXTERNAL;
        IF AMCKE THEN DO;
                PART=DBB(9);
                TRAP=~DBB(27)&~DBB(18);
                NT= DBB(27) & ~DBB(18);
                IAC=MSTOR& DBB(18) & ~ DBB(27) ;
                END;
        END MEMLOG;
/*
```

6.1.3 MEMLOG Variable List

Control Signals (1 bit parameters):

<u>Simulation Name</u>	<u>Description</u>
AMCKE	Enable access mode check enable
IAC	Access mode is illegal
MSTØR	Memory store/read access
NT	Access mode is "not there"
PART	Access is partitioned mode
TRAP	Access mode is TRAP

Registers (External variables):

<u>Simulation name</u>	<u>Description</u>
DBB	DB output bus

6.2 XNACC Procedure

6.2.1 XNACC Description

The XNACC procedure simulates the interface between the TP and the exchange net. At the present time it is concerned mainly with memory accesses. Later, however, it will need to be able to handle accesses to the other units as well.

XNACC first calculates the byte address of the byte of memory to be accessed. This is done by initially setting the variable ADDRESS equal to zero. Then ADDRESS is incremented by a power of two for each data bit of the CD which is equal to one. In this way, the data bits of the CD are regarded as binary digits, and their numerical value is found. This is done using only the last three bytes of CD since these are the ones containing the address.

When the byte address is formed, it is printed out. Then the double-word address is found by dividing the byte address by eight. If the number is not evenly divisible by eight, the remainder is dropped since ADDRESS is defined to be an integer. Then the double word address is printed out. If the double word address is greater than 128 it is set to zero since we presently have only four (32 double-word) pages simulated.

The first byte of the CD contains the read/write byte. If the first byte is all zeroes then this is a read and the double word addressed from memory (MEMSTØR) is placed in the memory buffer (MEMBUF).

If the first byte of the CD is not all zeroes, this indicates that a write is to be done. The write is not done here, but the read/write byte (the first byte of the CD) is saved in the storage array RWBYTE.

XNACC is called near the end of SIMCALL if the control signal ENMCBE is equal to one.

6.2.2 XNACC Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
  XNACC:  PROC;
          DCL CD(36) BIT(1) EXTERNAL;
          DCL ADDRESS FIXED BIN(31,0) EXTERNAL;
          DCL MEMBUF(0:1,1:36) BIT(1) EXTERNAL;
          DCL RWBYTE(0:7) BIT(1) EXTERNAL;
          DCL MEMSTOR(0:128,0:1,1:36) BIT(1) EXTERNAL;
          PUT SKIP LIST('XNACC CALLED');
          /* CALCULATE ADDRESS */
          ADDRESS=0;
          DO I=10 TO 17;
              ADDRESS=ADDRESS+CD(I)*(2**((33-I)));
          END;
          DO I=19 TO 26;
              ADDRESS=ADDRESS+CD(I)*(2**((34-I)));
          END;
          DO I=28 TO 35;
              ADDRESS=ADDRESS+CD(I)*(2**((35-I)));
          END;
          PUT SKIP LIST('BYTE ADDRESS =',ADDRESS);
          ADDRESS=ADDRESS/8;
          PUT SKIP LIST('DOUBLE WORD ADDRESS=',ADDRESS);
          IF ADDRESS>128 THEN DO;
              ADDRESS=0;
              PUT LIST('ADDRESS HAS BEEN SET TO 0');
          END;
          /* CHECK READ-WRITE BYTE. IF ALL 0'S - THEN THIS IS A READ*/
          DO I=1 TO 8;
              IF CD(I) THEN GO TO WRITE;
          END;
  READ:   DO I=0 TO 1;
              MEMBUF(I,*)=MEMSTOR(ADDRESS,I,*);
          END;
          GO TO XNEND;
  WRITE:  DO I=0 TO 7;
              RWBYTE(I)=CD(I+1);
          END;
  XNEND:  PUT SKIP LIST('XNACC FINISHED');
          END XNACC;
  /*
```

6.2.3 XNACC Variable List

Registers and buses - external variables:

<u>Simulation Name</u>	<u>Description</u>
ADDRESS	Binary number - contains address of word of memory to be accessed
CD	Cable Drivers to exchange net
MEMBUF	Memory Buffer - contains double word from memory
MEMSTOR	Memory Storage 128 double words (4 pages)
RWBYTE	Read/write byte - storage for first byte of AR which contains information for reading and writing the memory

6.3 TII1, TII2 Procedure

6.3.1 TII2, TII2 Description

TII2 is a separate entry point to TII1. They are both used to write data into the memory storage (MEMSTØR). TII1 is called when it is necessary to write into the first word of the double-word addressed. TII2 is called when it is necessary to write into the second word of the double-word addressed.

The read/write byte has been saved in the RWBYTE array. A one in bit position I of the RWBYTE indicates that byte I of the double word is to be written into. So, if TII1 is called, bits 0 through 3 of RWBYTE are tested and a write is done from byte I of the CD (cable driver) to byte I of the double-word of MEMSTØR. If TII2 is called, bits 4 through 7 of the RWBYTE are tested for writing into bytes 4 through 7. These correspond to bytes 0 through 3 respectively of the second double word.

TII1 and TII2 are called from within the memory sequence.

6.3.2 TII1, TII2, Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
  TII1:  PROC;
        DCL CD(1:36) BIT(1) EXTERNAL;
        DCL MEMSTOR(0:128,0:1,1:36) BIT(1) EXTERNAL;
        \DCL ADDRESS FIXED BIN(31,0) EXTERNAL;
        DCL RWBYTE(0:7) BIT(1) EXTERNAL;
        PUT SKIP LIST('TII1 ENTERED');
        DO I=0 TO 3;
          IF RWBYTE(I) THEN DO J=I*9+1 TO I*9+9;
            MEMSTOR(ADDRESS,0,J)=CD(J);
          END;
        END;
        PUT SKIP LIST('TII1 FINISHED');
        RETURN;
  TII2:  ENTRY;
        PUT SKIP LIST('TII2 ENTERED');
        DO I=0 TO 3;
          IF RWBYTE(I+4) THEN DO J=I*9+1 TO I*9+9;
            MEMSTOR(ADDRESS,1,J)=CD(J);
          END;
        END;
        PUT SKIP LIST('TII2 FINISHED');
        END TII1;
/*
```


6.3.3 TII1, TII2 Variable List

External Variables:

<u>Simulation Name</u>	<u>Description</u>
ADDRESS	Contains double word address for memory access
CD	Cable drivers to exchange net
MEMSTØR	Memory Storage - 128 double words
RWBYTE	Holds read/write byte from AR

6.4 MDAT1, MDAT2 Procedures

6.4.1 MDAT1, MDAT2 Description

MDAT1, with its entry MDAT2, is a procedure to read from the memory buffer (MEMBUF), putting the data in the cable terminators (CT). MDAT1 is called when it is required to read the first word of the double-word addressed. MDAT2 is called when it is required to read the second word of the double-word addressed.

After the read is performed, the contents of CT is printed out in either case.

MDAT1 and MDAT2 are called from within the memory sequences.

6.4.2 MDAT1 Listing

```
// EXEC PL1
//PL1.SYSPUNCH DD SYSOUT=B
//PL1.SYSIN DD *
  MDAT1: PROC;
    DCL MEMBUF(0:1,1:36) BIT(1) EXTERNAL;
    DCL CT(1:36) BIT(1) EXTERNAL;
    PUT SKIP LIST('MDAT1 ENTERED');
    CT(*)=MEMBUF(0,*);
    CALL PRINT1(CT,'CT  ');
    RETURN;
  MDAT2: ENTRY;
    PUT SKIP LIST('MDAT2 ENTERED');
    CT(*)=MEMBUF(1,*);
    CALL PRINT1(CT,'CT  ');
  END MDAT1;
/*
```

6.4.3 MDAT1, MDAT2 Variable List

External variables:

<u>Simulation Name</u>	<u>Description</u>
CT	36 bit cable terminators
MEMBUF	Memory buffer - a double word

U. S. ATOMIC ENERGY COMMISSION
UNIVERSITY-TYPE CONTRACTOR'S RECOMMENDATION FOR
DISPOSITION OF SCIENTIFIC AND TECHNICAL DOCUMENT

(See Instructions on Reverse Side)

1. AEC REPORT NO.

C00-1018-1212

2. TITLE

TAXICRINIC PROCESSOR OF ILLIAC III: SIMULATION MANUAL

3. TYPE OF DOCUMENT (Check one):

- ☒ a. Scientific and technical report
☐ b. Conference paper not to be published in a journal:

Title of conference _____

Date of conference _____

Exact location of conference _____

Sponsoring organization _____

- ☐ c. Other (Specify) _____

4. RECOMMENDED ANNOUNCEMENT AND DISTRIBUTION (Check one):

- ☒ a. AEC's normal announcement and distribution procedures may be followed.
☐ b. Make available only within AEC and to AEC contractors and other U.S. Government agencies and their contractors.
☐ c. Make no announcement or distribution.

5. REASON FOR RECOMMENDED RESTRICTIONS:

6. SUBMITTED BY: NAME AND POSITION (Please print or type)

Mary Ann Kreger
Programmer

Organization Department of Computer Science
University of Illinois
Urbana, Illinois 61801

Signature

Mary Ann Kreger

Date

June 10, 1970

FOR AEC USE ONLY

7. AEC CONTRACT ADMINISTRATOR'S COMMENTS, IF ANY, ON ABOVE ANNOUNCEMENT AND DISTRIBUTION RECOMMENDATION:

8. PATENT CLEARANCE:

- ☐ a. AEC patent clearance has been granted by responsible AEC patent group.
☐ b. Report has been sent to responsible AEC patent group for clearance.
☐ c. Patent clearance not required.