SHARE PROGRAM LIBRARY AGENCY

PROGRAM NUMBER

114004

University of Miami

1365 MEMORIAL DRIVE - CORAL GABLES, FLORIDA

(305) - 284-6257

SHARE PROGRAM LIBRARY SUBMITTAL FORM

SHARE PROGRAM LIBRARY AGENCY
Triangle Universities Computation Center
Post Office Box 12076
Research Triangle Park, North Carolina USA 27709

SPLA CONTROL NUMBER:

This form should be completed and submitted with the program package to the SHARE Program Library Agency at the address shown above. Standards and instructions for submitting programs are in the SHARE Reference Manual, Section 6.

(1) Program Number (to be filled by SPLA) 370D-11.4.004

(2) Title of Program . . . . . SIMCOMP

(3) System Type(s) (Machine) . . . . . IBM 370/148

(4) Search Key(s) . . . . . FILE 1 PROGRAM SOURCE DECK
SPRG. IN COLS. 1 - 72
368 - 80 BYTE RECORDS (BLOCKED AT 3120)
TAPE MARK EOF

(5) Programming Systems/Languages . . . . IBM OS/PL1(F) or PL1 OPTIMIZER

(6) Primary Subject Code . . . . . TRAINING

(7) Minimum System Requirements OS PL1, PL1 TRANSIENT LIBRARY, 256K STORAGE

(8) New (N) or Revision (R) (if revision, show prior Program Number in Item 1) NEW

(9) Date of Submittal . . . . . February 27, 1981

(10) Documentation (number of original pages submitted) 18

(11) Author's Name and Address . . . . . PAUL K. DEAN
INGERSOLL MILLING MACHINE COMPANY
707 FULTON AVENUE
ROCKFORD, IL 61101

(12) Direct Technical Inquiries to Name & Address (if different than Author)

DISCLAIMER

Triangle Universities Computation Center (TUCC) serves solely as the distribution agent for contributed programs and does not test or maintain them. They are distributed essentially in the original form submitted by the author. Neither TUCC nor SHARE, INC., makes any warranty, expressed or implied, as to the documentation, function, or performance of the contributed programs.

(13) Submitter's Installation Membership Code . . . . . ING

(14) Abstract (should contain sufficient information for a reader to determine the value of the program). Listed on the reverse side of this form are subjects which may serve as a guide for a descriptive abstract.

SHARE PROGRAM LIBRARY SUBMITTAL FORM

Subject Guide:

a. Purpose
b. Programming Language used
c. Version and modification level or release number
d. Field of application
e. Type of routine (main program, subroutine, etc.)
f. Specific description of machine requirements

SIMCOMP is a computer training aid that helps novice programmers develop a "feel" for how a computer works. The program simulates a small computer that duplicates, on a simple scale, the basic operations of the larger computer. A student using SIMCOMP enters a program in "object code" and the program is executed according to the code entered. In this system, errors are diagnosed rather than "bombing" the system as can happen when mistakes are made in programming a microprocessor.

SIMCOMP is written in OS/PL1. It can be compiled using either the OS/PL1(F) compiler or the OS/PL1 Optimizing compiler. Simcomp is a main program routine, using about 256K of memory. This is version 1.1 of SIMCOMP.

DISCLAIMER

Triangle Universities Computation Center (TUCC) serves solely as the distribution agent for contributed programs and does not test or maintain them. They are distributed essentially in the original form submitted by the author. Neither TUCC nor SHARE, INC., makes any warranty, expressed or implied, as to the documentation, function, or performance of the contributed programs.

(Please attach additional pages if necessary) . . . . . Total pages attached _____

An "Acknowledgement of Assistance" statement must be attached to this Submittal Form.

Permission to Publish

"I hereby give the SHARE Program Library Agency permission to reprint, reproduce, and distribute this program"

(15) Signature of Submitter and Date Paul K. Gleason 2/27/81

(15) Signature of Installation Addressee George J. Hess 2/27/81

SPLA 1001 (R) Revised 11-76

TAPE KEY

The source file may be loaded using the IBM utility IEBCOPY. The tape file description is: DSN=SOURCE,LABEL=(1,NL),DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120).

FILE 1 PROGRAM SOURCE DECK
SPRG. IN COLS. 1 - 72
368 80-BYTE RECORDS (BLOCKED AT 3120)
TAPE MARK EOF

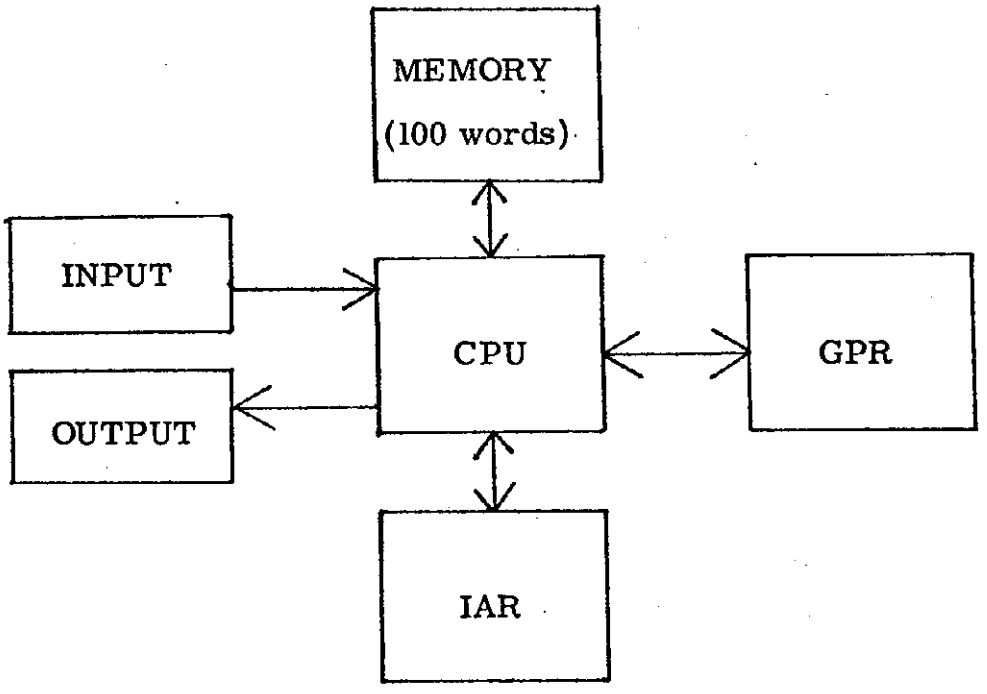
BASIC FACTS ABOUT SIMCOMP

SIMCOMP is a computer program that simulates the operation of an actual computer. It is not necessary that you understand the program itself, but in learning what it does, you will learn how a "real" computer works.

SIMCOMP operates on the stored program principle with direct addressing. This means a program is stored in the computer's memory along with any data used during the program's execution. The user states explicitly the memory location used by the computer.

The programs you will be writing for SIMCOMP will be written in machine code, which means you will be talking directly to the computer. Higher level computer languages, like BASIC and PL/1, are written in an English-like form which must be converted to machine code by the computer, but which the user finds much easier to work with.

The computer is composed of five basic parts, which are diagramed below.



The Central Processing Unit, or CPU, directs all operations of the computer. It interprets the instructions the computer receives, and coordinates all the processing that goes on. In short, the CPU acts like the "brains" of the computer.

The Instruction Address Register, or IAR, contains the location, or address, of the current instruction being acted upon. The IAR is always pointed to zero at the start of the program. It is automatically incremented by one after each instruction to bring up, or fetch, the next instruction. There is an exception to this. In a branch instruction, the number in the IAR is set directly by the instruction. You will learn more about branching and other instructions later on.

The Memory in SIMCOMP contains 100 locations, sometimes called words, in which to store the instructions and data of a program. Normally, the instructions are stored in the lower memory locations and the data is stored in the upper memory locations. The memory can be manipulated during a program by using a read or store instruction.

The General Purpose Register, or GPR, is the register where all arithmetic takes place. It is used to move data from one memory location to another. The GPR is the communicator between the CPU and data stored in the memory. SIMCOMP has one GPR; some larger computers may have as many as 16 general purpose registers.

RULES FOR USING SIMCOMP

What happens when SIMCOMP is given a program to execute? First, the computer must store the program in its memory, because SIMCOMP operates on the stored program principle. Input to the computer, which is keypunched on computer data cards, is written in an object code which is the "language" SIMCOMP uses. When the program is loaded in the computer (stored in its memory), the IAR is set to zero and the computer begins execution of the program. The program runs and, if there were no errors in it, you get your results.

The input format for SIMCOMP programs takes the following form. Columns 2 & 3 of the data card hold a 2 digit number that gives the memory location for that instruction. Columns 4, 5 & 6 are left blank. Columns 7 thru 11 hold the object code. The first two digits of the object code are the operation code, or opcode, which detail the particular instruction to be executed. The last three digits of the object code hold either an address in memory or an immediate data integer. Columns 12 thru 19 are left blank. Columns 20 thru 49 are set aside for comments.

Comments are used to describe an instruction to make it more readable to humans. A comment must appear on each card. Without it, the program will run but SIMCOMP will issue a warning. Comments document each step of the program so the user knows what the steps are for.

If the proper numbers are not in the proper columns an error will occur. Keypunch your data cards carefully; most errors in SIMCOMP programs are the result of incorrectly typed data cards.

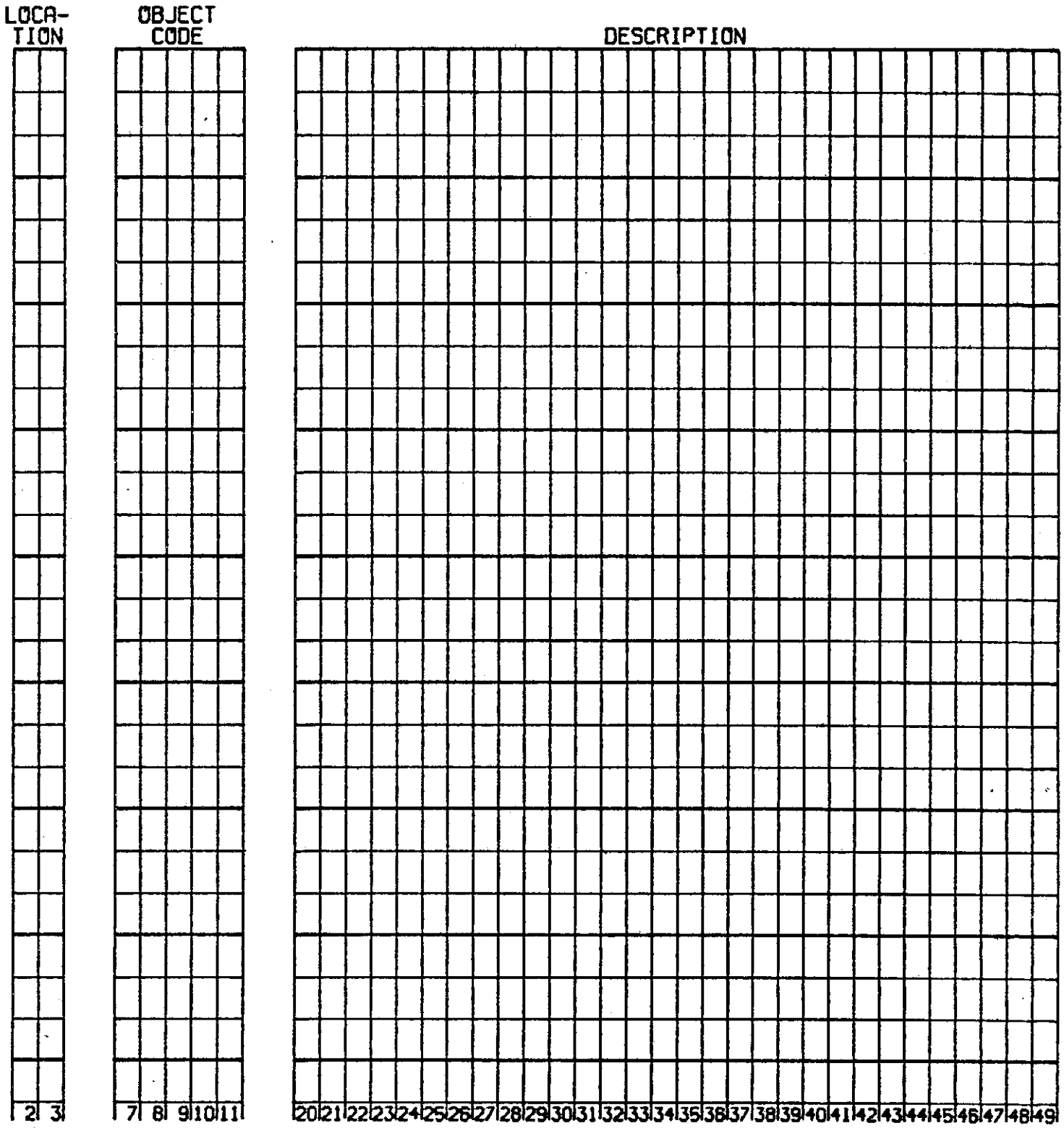
The last card of the program must be:

```
999  99999      END
```

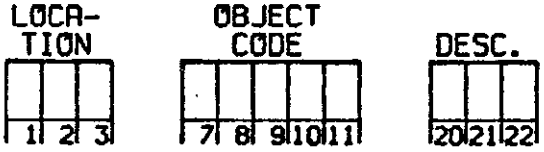
This signifies, "End of the program, data follows". If this card is left out, the program will not run and an error will be issued.

On the following page is a diagram showing the positions of each step of a SIMCOMP instruction. It will be useful as a coding sheet for your first program. Extra coding sheets are at the back of the book.

S I M C O M P



END STATEMENT



USING THE OPCODES AND INSTRUCTIONS

The READ and PRINT Instructions---Opcodes 50 & 60

In order for a computer to be useful for batch processing, there must be a method of getting data in and out of the computer. This is accomplished by using the READ and PRINT Opcodes (50 & 60).

Opcode 50 is used to read data from the input stream. Data for the program should be punched on cards following the 999 99999 END statement. Each time a READ opcode is encountered, the next card in the data stream is scanned for a number. This number is then placed into memory in the location specified by the instruction. The previous contents of that location is lost.

Here's a sample READ instruction:

```
00 50099     Read number into location 99
```

Once data is read in and processed, there needs to be a method of obtaining the results from the computer. The PRINT opcode (60) is used for this purpose. The PRINT instruction simply prints the contents of the specified memory location. A new line is printed for each PRINT instruction encountered. The content of the memory location is not changed by a PRINT instruction.

A diagnostic and debugging aid that may be used in developing and testing a program is the DUMP opcode (61). The DUMP instruction takes the form 61000. This instruction will cause the contents of all 100 memory locations to be printed.

SIMCOMP will print a maximum of 100 lines for each job. This is to help avoid enless PRINT loops. Each line printed is counted as one, and each dump is counted as ten in calculating the total lines. If a program attempts to print more than 100 lines, a diagnostic will be issued and the job will be terminated.

The following is a sample program using the READ, PRINT and DUMP instructions. Enterprising students wishing to view the output may want to keypunch and run the program.

SAMPLE PROGRAM #1

```
00    99000    SAMPLE PROGRAM USING READ,
01    99000    PRINT AND DUMP
02    50099    READ X
03    60099    PRINT X
04    50098    READ Y
05    60098    PRINT Y
06    50097    READ Z
07    60097    PRINT Z
08    61000    REQUEST A MEMORY DUMP
09    00000    STOP
999   99999    END
```

Put your data, numbers for X, Y, and Z, on three cards following the END statement.

The LOAD and STORE instructions---Opcodes 10 & 90

Using the 50 opcode, data is read into the memory from the input stream. Once the data is in memory, it must be manipulated in order to process the data according to our needs. The LOAD instruction is used to copy the data in memory into the GPR (General Purpose Register) through which all internal data manipulation takes place.

The LOAD instruction takes this form:

```
00  10099    Load GPR with contents of location 99
```

When a LOAD instruction is executed, the old contents of the GPR is lost, but the contents of the specified memory location remains unchanged. The data from the memory copied to the GPR is thus available for use.

Once a number is loaded into the GPR, it may be stored into a memory location, or it may be manipulated using arithmetic instructions, which we will discuss later on.

The STORE instruction, opcode 90, takes the form:

```
00  90099    Store GPR into location 99
```

A simple LOAD and STORE is an elementary operation. It is useful when you need 2 copies of the same number in memory. The STORE may also be used after any operation on the GPR in which the results need to be saved or printed. (Remember: PRINT works only from memory, not from the GPR.)

A sample program using the LOAD and STORE instructions follows.

SAMPLE PROGRAM #2

```
00    99000    SAMPLE PROGRAM USING LOAD AND STORE
01    50099    READ X
02    60099    PRINT X
03    50098    READ Y
04    60098    PRINT Y
05    10099    LOAD X INTO GPR
06    90090    STORE X INTO LOCATION 90
07    10098    LOAD Y INTO GPR
08    90091    STORE Y INTO LOCATION 91
09    10099    LOAD X INTO GPR
10    90050    STORE X INTO LOCATION 50
11    61000    REQUEST A MEMORY DUMP
12    00000    STOP
999   99999    END
```

The student is advised to keypunch and run the program. Careful study of the output, especially the memory dump, will facilitate understanding of the LOAD and STORE processes.

The ARITHMETIC Opcodes---(20, 30, 40 & 70)

These opcodes are used to operate mathematically on the data in the GPR. The operations themselves are self-explanatory; however, there are some ideas to bear in mind regarding their implementation.

The first thing to remember is that SIMCOMP is an integer machine. This means that 5/2 = 2! This point must be kept in mind or the results of a program could be misleading or totally wrong.

The second idea to be brought out is the actual manner in which SIMCOMP executes arithmetic. The instructions take the form:

```
00    20099    Add contents of location 99 to GPR
```

SIMCOMP will add the contents of location 99 to that of the GPR. The results will be placed in the GPR. The previous contents of the GPR are lost and the contents of the memory location remain unchanged.

Similarly:

```
00    30099    Subtract contents of location 99 from GPR
00    40099    Multiply GPR by location 99
00    70099    Divide GPR by location 99
```

In each case, the results are placed in the GPR. Following is a sample program demonstrating the use of the ARITHMETIC opcodes.

SAMPLE PROGRAM #3

```
00    99000    SAMPLE PROGRAM USING ARITHMETIC
01    50099    READ X
02    60099    PRINT X
03    50098    READ Y
04    60098    PRINT Y
05    10099    LOAD X INTO GPR
06    20098    ADD X + Y TO YIELD Z
07    90097    STORE Z
08    60097    PRINT Z
09    10099    LOAD X INTO GPR
10    30098    SUBTRACT X - Y TO YIELD Z
11    90097    STORE Z
12    60097    PRINT Z
13    10099    LOAD X INTO GPR
14    40098    MULTIPLY X * Y TO YIELD Z
15    90097    STORE Z
16    60097    PRINT Z
17    10099    LOAD X INTO GPR
18    70098    DIVIDE X/Y TO YIELD Z
19    90097    STORE Z
20    60097    PRINT Z
21    00000    STOP
999   99999    END
```

In this program, use X=100 and Y=50. If you keypunch and run it correctly, your output will look like this:

```
  100
   50
  150
   50
 5000
    2
```

NO-OPeration---Opcode 99

The NO-OP opcode is used to aid the programmer. When it is encountered in a program, it does just as its name implies, nothing. The IAR is incremented by one and execution continues.

The instruction takes the form:

```
00    99000     Continue
```

A NO-OP is handy for a couple of reasons. Number one, an extra comment may be inserted in the program listing by using a NO-OP instruction. This may be useful in an exceptionally complex program where additional comments become necessary, or to break up logical sections of a program.

Secondly, a NO-OP can be used as a "place holder". While writing a program, it is handy to sprinkle in a NO-OP or two in case instructions need to be added later. The addition is more easily made by replacing a NO-OP with an instruction than it is by inserting the instruction and renumbering the entire program.

DATA INITIALIZATION

Data Initialization is the process of making a number, used as a data constant, an integral part of the program. Since SIMCOMP will accept and store in memory any number within its useable range, we may place a number into any memory location at the time a program is read in and have it available as a constant throughout program execution. Data Initialization is very useful when a number (constant) will be used several times in a program.

The code for Data Initialization looks like this:

```
99    00001    CONSTANT 1
```

This stores the number 1 in memory location 99. The constant will be in memory at the time the program begins. It will remain there throughout program execution unless the program reads or stores another number into that location, which will change the constant.

Use care when reading or storing data in your program so that you do not unintentionally replace your desired constant with another number and so cause errors in your program output.

A sample program using Data Initialization follows.

SAMPLE PROGRAM #4
DATA INITIALIZATION

```
00    50099    READ X INTO LOCATION 99
01    60099    PRINT X
02    20090    ADD CONSTANT TO X
03    90098    STORE X + C
04    60098    PRINT X + C
05    10099    LOAD X INTO GPR
06    40090    MULTIPLY X x C
07    90097    STORE X x C
08    60097    PRINT X x C
09    61000    DUMP MEMORY
10    00000    STOP
90    00005    CONSTANT = 5
999   99999    END
```

The BRANCH Opcodes---(80, 81, 82 & 83)

The BRANCH opcodes are used to cause a change in the sequence of program execution. Normally, the IAR is incremented by one after each instruction is executed. The BRANCH instruction, however, causes the IAR to point to a specified memory location.

The UNCONDITIONAL BRANCH, opcode 80, causes a branch to take place any time the instruction is executed. The instruction takes the form:

```
00   80036     Jump to location 36
```

When this instruction is encountered, it causes the IAR to be pointed to location 36 and execution of the program continues from there. It is possible to jump to any location using the BRANCH, so care must be taken in its use.

Three variations of the BRANCH opcode are opcodes 81, 82 and 83. These are BRANCH ON GPR POSITIVE, BRANCH ON GPR NEGATIVE, and BRANCH ON GPR ZERO respectively. The instructions take the same form as the UNCONDITIONAL BRANCH:

```
00   81054     Go to 54 if GPR > 0
00   82054     Go to 54 if GPR < 0
00   83054     Go to 54 if GPR = 0
```

When executed, a BRANCH is taken if the conditions are met. If they are not met, execution continues squentially. The IAR is incremented by 1 and execution of the program continues as normal.

Following is a sample program demonstrating the use of the BRANCH opcodes.

SAMPLE PROGRAM #5 - BRANCHING

```
00    50099    READ # OF NUMBERS TO AVERAGE
01    10099    LOAD THIS NUMBER INTO GPR
02    90090    STORE NUMBER IN MEMORY LOCATION 90
03    60099    PRINT THE NUMBER FOR A CHECK
04    82016    GO TO 16 IF NEGATIVE
05    83016    GO TO 16 IF ZERO
06    50098    READ NUMBER TO AVERAGE
07    10097    LOAD RUNNING TOTAL
08    20098    ADD NEW NUMBER
09    90097    STORE NEW RUNNING TOTAL
10    10099    LOAD COUNT INTO GPR
11    31001    DECRIMENT BY ONE
12    90099    STORE THE COUNT
13    60098    PRINT THE NUMBER JUST USED
14    80004    GET THE NEXT NUMBER
15    99000    CONTINUE
16    10097    LOAD RUNNING TOTAL
17    70090    DIVIDE BY NUMBER OF ENTRIES
18    90091    STORE AVERAGE
19    60091    PRINT AVERAGE
20    00000    STOP
97    00000    SET RUNNING TOTAL TO ZERO
999   99999    END
```

IMMEDIATE Instructions---Opcodes (11, 12, 21, 31, 41 & 71)

Immediate operations perform the same operations as their memory addressing counterparts. The difference is that the last 3 digits of the instruction are data rather than a memory address.

For example, LOAD IMMEDIATE opcode 11 has this form:

```
00    11075    Load the number 75 into the GPR
```

This loads the number 75 into the GPR... NOT the contents of memory location 75.

Another example, LOAD NEGATIVE IMMEDIATE opcode 12:

```
00    12767    Load -767 into GPR
```

loads the number -767 into the GPR.

Similarly:

00	21055	Add 55 to contents of GPR
00	31007	Subtract 7 from contents of GPR
00	41654	Multiply GPR by 654
00	71010	Divide GPR by 10

Remember that the previous contents of the GPR are lost following these operations.

WHAT AN ABEND DUMP & ASSOCIATED MESSAGES WILL TELL YOU

If a program abends (abnormally ends) for one reason or another, a message is generated describing the reason for program termination. The explanation of these messages will be found in Appendix B. The abend messages also show the IAR and GPR at the time the abend occurred. In the event an instruction itself caused the abend, the IAR will point to the offending instruction. Many times the contents of the GPR will assist you in determining what was happening at the time of the abend.

SIMCOMP will tell you how many instructions were executed from the user program. It may be helpful in debugging the program to follow SIMCOMP's steps through the program to determine the error. In the case of an endless loop, 1000 instructions will show as being executed. A problem such as this may be uncovered by using the trace of the last ten instructions executed. Many times the pattern of the loop will be repeated, pointing to the area of the program in which the problem occurred. The trace is a useful debugging aid if used correctly. It will point not only the problem instruction(s) but also show the program's path up to the problem.

The memory dump lists the contents of all memory locations at the time of the abend. The user can then easily verify that the program which is in memory is what was intended. Data stored in memory may also be checked in order to determine the problem.

APPENDIX A

SIMCOMP INSTRUCTION SET

OPCODE	NAME	DESCRIPTION
00	STOP	Stop execution
10	LOAD	Load GPR with contents of addressed memory
11	LOAD IMMEDIATE	Load GPR with 3 digit immediate data
12	LOAD IMMEDIATE	Load GPR with 3 digit negative immediate data
20	ADD	Add contents of addressed memory to GPR; store results in GPR
21	ADD IMMEDIATE	Add 3 digit immediate data to GPR; store results in GPR
30	SUBTRACT	Subtract contents of addressed memory from GPR; store results in GPR
31	SUBTRACT IMMEDIATE	Subtract 3 digit immediate data from GPR; store results in GPR
40	MULTIPLY	Multiply GPR by contents of addressed memory; put results in GPR
41	MULTIPLY IMMEDIATE	Multiply GPR by 3 digit immediate data; store results in GPR
50	READ	Read data card from input stream (one number per card)
60	PRINT	Print addressed memory contents
61	STORAGE DUMP	Request a dump (listing) of all memory locations at current time
70	DIVIDE	Divide GPR by contents of addressed memory; store results in GPR

OPCODE	NAME	DESCRIPTION
71	DIVIDE IMMEDIATE	Divide GPR by 3 digit immediate data; store results in GPR
80	UNCONDITIONAL BRANCH	Branch to location in immediate data
81	BRANCH POSITIVE	Branch to address given if contents of GPR is positive; continue with next instruction if not
82	BRANCH NEGATIVE	Branch to address given if contents of GPR is negative; if not, continue with next instruction
83	BRANCH ZERO	Branch to address given if contents of GPR is zero; continue with next instruction if not
90	STORE GPR	Store contents of GPR into specified memory location
99	NO-OP	No-operation; used to aid program writing

APPENDIX B

SIMCOMP WARNINGS AND ERRORS

WARNINGS	REASONS
No comment on source record	User neglected commenting a line in the program. This is a gentle reminder to comment your program.
Character data in numerical field ... Statement ignored	Characters appeared in the location or object code fields of the user program. The program finished being read in but could not be executed as location and object code must be numeric.

ERRORS	REASONS
Invalid location value on input... Job not executed	A location beyond the bounds of available memory has been specified in the user program. The program cannot be run.
Invalid opcode encountered	During execution a meaningless opcode was encountered. Since the instruction could not be executed, the program terminated.
Addressing protection error	User program instructed the CPU to fetch or store data from or to a location outside of the memory range.
End of file on input stream	This may be caused by one of two problems. The error can occure while SIMCOMP is reading the user program if the end card has been omitted. It can also occur during program execution if a READ instruction is issured and no data appears for the READ.
Zero divide attempted	Self-explanatory... attempt to divide by zero!
Execution time limit exceeded	SIMCOMP will execute 1000 instructions. If more instructions are executed, it is assumed that the program has encountered an endless loop. The program execution is terminated.

ERRORS	REASONS
Integer overflow occurred... GPR cannot be stored	An operation has caused the value in the GPR to exceed a SIMCOMP word storage size (99999), thus the value cannot be stored.
Integer underflow occurred... GPR cannot be stored	An operation caused the value in the GPR to exceed a SIMCOMP word storage size in a negative direction (- 99999), so the value cannot be stored.
Print limit exceeded	SIMCOMP allows a maximum of 100 lines of printed output. (A storage dump is counted as 10 lines). If this limit is exceeded, an endless PRINT loop is assumed and execution is terminated.
Character data in numeric input data field	During execution of a READ instruction, characters appeared on an input card. Remember, SIMCOMP reads 1, 2 or 3 digit integers.