UNIVERSITY COMPUTER CENTER

NOTES AND COMMENTS

October, 1968
Volume 3, Number 2

UNIVERSITY OF MINNESOTA
MINNEAPOLIS, MINNESOTA

SCOPE 3.1 SYSTEM MODIFICATIONS

CDC MODIFICATIONS

The latest officially available modification and correction set from CDC has been added to the SCOPE 3.1 system. This set was released about three months ago under the correction summary PSR120. This system will be released following completion of Extended Core Storage installation.

1. Corrective code has been added in the following programs:
   COMPASS
   RUN
   FUN (see special section on FUN)
   ALGOL
   COBOL
2. An almost completely revised SIMSCRIPT compiler (version 2.0 has replaced version 1.0, and the corresponding updates have been made to its object time routines.
3. The IDA SNOBOL4 interpreter has been updated, and descriptions of its use are available in the UCC library. (This does not imply that the updates have changed the description of its use in any way.)
4. The release version of FORTRAN EXTENDED (FTN) has been added to the system library. A special section on FTN is included in this newsletter.

LOCAL MODIFICATIONS

In addition to modifications mentioned in the September newsletter, the following will also become effective following completion of ECS modification work.

1. The (possible) 4th branch of an IF(UNIT) statement (parity error) is now operable when used in a program compiled by RUN or FUN.

For example

```
BUFFER IN(1,0)(DATA(1),DATA(2))
IF(UNIT,1)1,2,3,4
```

transfers to statement 4 upon detection of a parity error. On BUFFER IN and OUT statements, a parity error message is still put into the dayfile, but the user may now act upon this error return.

Routines affected: BUFFEO, BUFFEI, IOCHEK

2. The EXIT card is again operable. It will be listed on the dayfile as a control card, and will have the following form:

   EXIT.C

3. The time limit on the job card will be interpreted as decimal. In addition, only priorities of 1, 2, and 3 are recognized. (See July and September newsletters.)

The FTN Compiler

The primary object of FTN is to produce the most efficient object code possible. A technique of actually timing the object code is used in order to find the best possible arrangement of instructions. The 6600 is capable of executing a variety of instructions simultaneously (provided that they do not conflict, e.g., trying to use the result of an operation as the input to another operation before the original operation has been completed), and FTN attempts to fully exploit this capability by critically examining the sequence of generated instructions.

FTN also attempts to allow the user many liberties and extensions in writing his programs. These include:

a. Any valid (non-double, non-complex) arithmetic expression may be used as a subscript (including subscripted variables)

b. Multiple returns from a function or subroutine

c. Arithmetic 2-branch IF statements

d. A tabulation (T) specification in format statements

e. ECS and random (disk) input-output (not yet applicable)

f. Hand-optimizable code (this is due to the fact that the actual output of FTN is really COMPASS assembly language--FTN calls in COMPASS automatically if no compilation errors occur).

g. Revised object routines to speed up execution (e.g., SQRT, TAN, etc.)

h. Improved diagnostics (e.g., undefined variables)

One must be aware of the following limitations and restrictions also:

a. Compilation speed may be very slow, the rate is dependent on length and complexity of source code.

b. The method of passing argument locations from program or subprogram to subprogram is different from (and incompatible with) that of RUN and FUN. (The UCC is in the process of ensuring that its routines may be accessed by FTN compiled programs.)

Generally speaking, FTN is recommended for static programs which are being used repeatedly. In this case execution of binary decks produced by FTN makes the additional compile time necessary, profitable.

The FUN Compiler

The FUN compiler is really a SCOPE-style (capable of producing relocatable binary records to be loaded by the SCOPE linking loader) compiler based on a COS (Chippewa Operating System) compiler called RUNDEC (RUN, December version). Approximately a third of the compiler is from RUN 2.3 and two-thirds is relatively new.

The new sections of the FUN compiler are primarily optimization routines. These routines mainly affect the object code compiled for DO loops and references to indexed variables. In both cases attempts are made to produce very efficient object code.

The latest FUN compiler, level 31 (which refers to a correction level) will be available on the system tape. Many new features have been added over the level 19 FUN compiler currently available and the most significant are discussed below:

1. In general, compilation field lengths for FUN average 2000B to 4000B CM cells above corresponding field length requirements for RUN.
2. The octal listing options (L,M) now will produce COMPASS mnemonics in addition to the octal instructions.
3. Compilation rate is on the order of 4000 to 10000 cards per CPU minute (cf. for RUN, 5000 to 11000+ cards per CPU minute). The rate is program-type and program-size dependent.
4. Improvements to internal buffering routines have speeded up real-time throughput.
5. The addition of an option to compile rounded floating-point arithmetic instructions has been implemented.
6. The addition of a form-free parameter-calling-sequence similar to FTN and COMPASS. The parameters for the fixed position sequence are:

FUN,MODE,OFL,BL,I,Ø,B,NL,A,C,R

where		Default Value
MODE	= G,L,M,P,S	G
OFL	= object time field length	(current FL may be reduced by loader)
BL	= object time buffer lengths	2022B
I	= source input file	INPUT
Ø	= listable output file	OUTPUT
B	= binary output file	LGO

NL = line count 10000B

(note that this may be changed at object-time by LC option on file load (e.g., LGO.) card, see UCC newsletter, Sept., 1968)

A = ASA switch (no ASA format interaction)

C = cross-reference switch No cross-reference

R = rounded FP arithmetic option Non-rounded arithmetic

A, C, and R are normally OFF, and they must be set non-zero or non-blank to be set ON.

The form-free card is

FUN,P1,P2,P3,P4,P5,...,P10.

where the PN are of the form (in any order) PN or PN=LFN, or PN=octal number.

a. For compile mode and options parameter

G,L,M,P,S ACR A, C, and R as above in fixed-field sequence. Any combination of the above selects the appropriate option.

b. For line limit

NLXXXXXX, XXXXXX an octal number (upper limit 77777)

c. For source I=LFN, desired source input file=LFN

d. For output file Ø=LFN, desired listable output file=LFN

e. For binary output B=LFN, desired binary output file=LFN

f. OFL and BL options are not available on the free-form calling sequence.

g. default values are:

G,A,R,C
I = INPUT
Ø = OUTPUT
B = LGO
NL = 10000 (octal)

Quick Compilation

The SCOPE 3.1 operating system is capable of handling the original Chippewa Operating System (COS) compiler and the COS versions of SCOPE object-time routines (e.g., INPUTC, OUTPUTC, etc.). A very useful advantage to the latest COS compiler RUNDEC (from which the current FUN compiler was derived) is that it has an option to compile directly to core, and an option to run in batch mode. Both of these concepts have been utilized in the UOFM 1604 operating system compiler FORTRAN-60. (This compile-to-core concept is also being employed in the student compiler being developed at the UCC.) The basic advantage of this option is that no disc

references are made for the binary output. This can be most useful for short production jobs, student runs, and debugging runs. The UCC has investigated the possibility of adding this compiler and object routines to the SCOPE 3.1 system, and it should be implemented in the near future.

FATAL ERRORS

Analysis of jobs run during September has shown that the most frequent "FATAL ERRORS" are 65, 78, and 84. To aid in program debugging a brief description of the cause of each will be given.

Fatal Error 65 "BCD ENDFILE INPUT"

This error occurs when the program tries to read more data cards than are available. As an example, consider the data deck which contains only the following four cards:

```
12.00    13.00
14.00    15.00
16.00    17.00
18.00    19.00
```

If the read statement in the program is

```
      READ 100,A,B
  100 FORMAT(2F10.2)
```

the result of the first execution of the read statement is that A and B assume the values 12.00 and 13.00 respectively. The second and third times the read statement is executed the next two cards are read. The fourth execution of the read statement assigns a value of 18.00 to A and a value of 19.00 to B. If the read statement is executed a fifth time, since there are no more data cards, an "END OF FILE" will be read. No diagnostic will occur on this fifth read so as to allow the programmer to check for the END OF FILE with an "IF" statement if he desires. The values read for A and B on this fifth read are indeterminate and any computations using A and B will give random results. If the program executes the read statement a sixth time the program will terminate with the message

```
BCD "ENDFILE" INPUT FORMAT NO. 100
```

Fatal Error 78 "ILLEGAL DATA IN FIELD"

Suppose that the data deck of the program contains a card which has "12.4" in columns 1-4 and that the program has the read statement

```
      READ 50,JACK
   50 FORMAT(I4)
```

when this read statement is executed an attempt is made to read an integer number (no decimal point) from the data card. Since the data card contains a decimal point "FATAL ERROR 78" will stop the program. An indication of the error together with the card image causing the error will appear on the output file. This allows the programmer to easily find and correct the "bad" card.

Fatal Error 84 "LINE LIMIT EXCEEDED"

The methods of setting or altering the line limit for the file output were treated in detail in the September newsletter.

The Fortran NAMELIST Feature

Users developing new Fortran programs (and those familiar with UOFM Fortran-60's versatile and very useful OUTPUT statement on the 1604) may wish to become familiar with the Fortran IV NAMELIST statement available under the RUN, FUN, and FTN compilers.

With NAMELIST one may easily insert output statements in a program without the bother of synthesizing format statements. NAMELIST takes care of formating, and in addition, also prints out the names of the variables next to their values in the output. One restriction of NAMELIST is that (as with UOFM Fortran-60 OUTPUT) the variable may only be printed according to its TYPE (REAL, INTEGER, COMPLEX, DOUBLE or LOGICAL) and will not allow printing in octal or Hollerith (O, R, A) fields. (Users desiring octal or Hollerith printing should see descriptions of DUMP, PDUMP, in the RUN Fortran manual, pages 7-10, and the write-up of the MINN COMPASS dump routine CMPSDMP).

The only changes that a user needs to make to his program in order to use NAMELIST are:

1. Add an equivalenced file (to OUTPUT normally) on his program header card.
   EXAMPLE:
   ```
   PROGRAM NOTDONE(INPUT,OUTPUT)
   ```
   might be re-punched as
   ```
   PROGRAM NOTDONE(INPUT,OUTPUT,TAPE1=OUTPUT)
   ```

2. Declare the variables to be printed in a NAMELIST block (note the similarity to the labeled common declaration).
   EXAMPLE:
   ```
   NAMELIST/DEBUG/I,J,RESULT,VIX44,COUGH,MEDICIN
   ```
   (It must be noted that if an array is declared in the NAMELIST block, the entire contents of the array will be printed. Also every variable in a block is printed when a NAMELIST write statement is executed (see below)). The NAMELIST block name must be unique (i.e. not used anywhere else) and may appear only in READ or WRITE statements.

3. Insert NAMELIST write statements in his program at the appropriate places.
   EXAMPLE:
   ```
   WRITE(1,DEBUG)
   ```
   Note that the above statement is similar to the coded WRITE statements, but that there is no explicit list. The list of variables is implicit in the NAMELIST identifier DEBUG, and the NAMELIST name DEBUG is in the place of the format statement number.

The output from NAMELIST is distinctive and of the form:

Column 2

```
$DEBUG
I      = (integer value)
J      = (integer value)
RESULT = (real value)
.
.
ETC.
$END
```

The use of NAMELIST output statements will cause the system routine OUTPTN (length = about 600_8 cells) to be loaded with the program.

To more fully illustrate the use of NAMELIST for debugging, suppose that a program has a MODE1 arithmetic error (address out of range), and although we do not know the cause, we have isolated the statement in the program which caused the fault:

```
ARRAY(I*2-INDEX1+L2+M*N*INDEX2 = ARRAY(INDEX3)
```

We might suspect that somehow the index expression contains an incorrect (or at least an unexpected) value, but how might we easily find out which one?

Assuming we have equated TAPE1 (as per condition 1 above) to OUTPUT, we form the following NAMELIST declaration:

```
NAMELIST/HELPME/I,INDEX1,L2,M,N,INDEX2,INDEX3
```

and insert the following write statement

```
WRITE(1,HELPME)
ARRAY(I*2-INDEX1+L2+M*N*INDEX2) = ARRAY(INDEX3)
```

Now, whenever the statement in question is executed, we will have a record of all the variables which were combined to make up the final value of the index for ARRAY. In this way, we may check the flow of values and isolate (if not solve) the actual problem.

For a more detailed discussion of NAMELIST (especially the NAMELIST input technique) see the Fortran Extended Manual, pages 5.6 to 5.9.

NOTES & COMMENTS

WARNING!!!

The SCOPE 3.1 loader may transfer control to the wrong entry point if the file being loaded contains a subroutine or function (subprogram in general) of the same name as the file. If, for example, the file LGO contains a subprogram named LGO, control will be transferred to the subprogram LGO instead of the main program, which may cause the central processor to stop and the job to abort on a time limit. The same situation may be encountered in yet another way. If a binary deck including a subprogram named INPUT is loaded for execution from the file INPUT with the control card INPUT control will be transferred to subprogram INPUT instead of to the main program.

Available Memory

The addition of a second EXPORT/IMPORT station (West Bank) requires that 2000B addition of Central Memory (CM) be attached to the Export package for buffers. This, of course, implies 2000B fewer words of CM for user programs. As a first step in an attempt to regain CM for user programs as well as to speed up job throughput, a new READ/OUTPUT package is currently being checked out and will be pressed into service soon. The new package requires only one control point and from 300-700 octal words less of CM. A summary of available CM is given below with all numbers in octal.

Number of stations in operation	Old I/O	New I/O	
		best case	worst case*
No export stations	151000	151700	151300
1 export station	144600	145500	145100
2 export stations	142600	143500	143100

*The amount varies depending on the number of devices being driven. A total CM of 156400 is available if none of the I/O packages is being used, a very unlikely situation.

FORTRAN Short Course

A short course in Fortran will be given by the UCC on the following days:

October 28 through November 1 (Monday-Friday)

The class will be from 4:15 to 5:15 PM in Room 18, Mechanical Engineering. There is no need to register for the class.

Supply Cost Changes

Beginning November 1, the cost of printer paper will be raised from .7 cents per page to 1 cent per page.

Programming Services

It has long been the policy of the University Computer Center not to provide programming services to users. Users either do their own programming or hire programmers in order to use the computers. To help the former type of user, the UCC periodically gives short courses on Fortran, the operating system, other languages and in other areas where we may provide information useful to the individual programmer. The classes on the operating system are usually given during the quarter breaks.

Helping the user who must hire programmers is another matter. At one time the UCC kept a file of available free-lance programmers but this was difficult to keep current. At present the UCC maintains a list of employers who need programmers on the bulletin board in 227 Experimental Engineering which works to some degree. We will now attempt to again maintain a file of programmers so that prospective employers can more quickly contact them. Programmers who are interested should contact Mrs. Levitan,

227 Experimental Engineering, extension 3-4360, and fill out a form. Familiarity with other languages and machine language will be noted on the form. As soon as there are some applicants, employers may call Mrs. Levitan and try out the system. (A sample copy of the form is attached to this newsletter.)

New Program

A general card read program, GENREAD, is available under the SCOPE 3.1 operating system. Many 1604 users were familiar with this program which allows reading a card as 960 separate bits of information or as 80 words of 12-bit numbers. Thus, the user may apply any interpretation he wishes to the punch patterns on a card. Certain "flag" cards force the in-between cards into binary files which are read by GENREAD. A more general program, GENREED, is also available.

A write-up is forthcoming but for immediate details, see Dennis Lienke, 212 Experimental Engineering, extension 3-5907.

NOTE: THESE JOBS MUST BE RUN AT THE LAUDERDALE SITE.

LIBRARY CHANGES

Additions

The following programs have been added to the Fortran library. Write-ups are available in room 223 Experimental Engineering.

QRCMPLX - eigenvalues of complex non-symmetric matrices
PERMUTE - random array permutations

Deletions

EIG7 has been removed from the library and replaced by QRCMPLX which is noticeably faster and more accurate.

Changes

1. The following routines have had error checks and messages inserted into the code:

AITKENF	GENREAD	PRNPLOT
AMEAN	GENSORT	QR
BESJ	IBIN	QRCMPLX
BETAI	IELIPFE	RANBIN
CBIN	IVLFREQ	RAN3F
CDFN	JACOBIN	RANDEV
CDFNI	LINEINT	RK
CELIPFE	LINT	RKGHN
CHSQ	MEANVAR	ROM1F
CHSQI	MERGE2	ROM2F
CINTEG	MERGE4	ROOT1
CMXLNEQ	MXCMBN	RVECT
CONVERT	MXEXTRM	SCLPLT
CVECT	MXMOV	SIMPSON
DESCRIB	MXMPLY	SORT1
DMXLNEQ	MXMPLY1	SORT2

EI	MXTRIDI	SYMINV
EIG3	MXTRP	SYMPACK
ERFN	ORTHON	SYMUPK
FINV	PERMUTE	SYMSOLV
FREQDSN	PLOTPAC	SYMSOLU
FVR	PLROOT1	TINV
GAMMAF	PLROOT2	XINT

2. The following routines have had changes made to their code to correct errors in certain versions or to improve results:

CBIN	MXTRID	
DESCRIB	ORTHON	RAN3F
ERPROC	PLROOT1	RVECT
FVR	PLROOT2	SCLPLT
GENSORT	PLROOT3	SORT2
IRAN	PRNPLOT	SYMSOLV
IBIN	RAN1F	SYMSOLU
JACOBIN	RAN2F	

3. The assembly language version of UMST531 has replaced the Fortran version.

PROGRAMMER'S QUESTIONAIRE

(Please type or print)

DATE: _____

NAME: Mr. / Mrs. / Miss _____

HOME ADDRESS: _____

CAMPUS ADDRESS: _____

HOME PHONE: _____ CAMPUS PHONE: _____

STUDENT? _____ YEAR: _____ DEGREES EARNED: _____

MAJOR: _____ MINOR: _____

COMPUTER RELATED COURSES TAKEN:

COMPUTERS YOU HAVE WRITTEN SUCCESSFUL PROGRAMS FOR:

COMPUTER LANGUAGES YOU HAVE USED:

RETURN TO: Mrs. Levitan
University Computer Center
227 Exp. Eng. Bldg.
373-4360