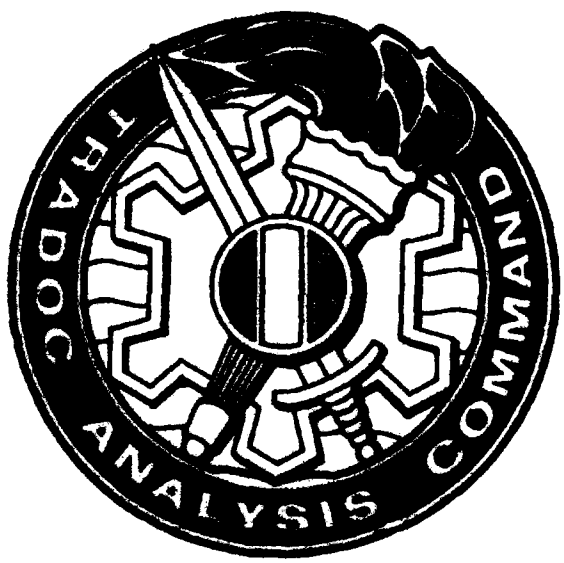
(2)

SEP 1988

TRAC-F-TM-1088

ACN 16306

VIC 1.2 MODEL CONVERSION FROM THE VAX TO THE SUN 4



Fort Leavenworth

US ARMY
TRADOC ANALYSIS COMMAND - FORT LEAVENWORTH
(TRAC - FLVN)
OPERATIONS DIRECTORATE
FORT LEAVENWORTH, KANSAS 66027

Technical Memorandum TRAC-F-TM-1088
September 1988

TRADOC Analysis Command-Fort Leavenworth (TRAC-FLVN)
Model Support Directorate
Fort Leavenworth, Kansas 66027-5200

VIC 1.2 MODEL CONVERSION FROM THE VAX TO THE SUN 4

by

Mike Hannon

ACN 16306

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) TRAC-F-TM-1088	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION TRAC-FLVN 6b. OFFICE SYMBOL (If applicable) ATRC-FM	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Director TRAC-FLVN (ATTN: ATRC-FM) Ft Leavenworth, KS 66027-5200	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11 TITLE (Include Security Classification)
VIC 1.2 Model Conversion from the VAX to the SUN 4

12. PERSONAL AUTHOR(S)
MIKE HANNON

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final Report	FROM 6-88 TO 8-88	1988 September	21

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This document describes work completed at the TRADOC Analysis Command-Fort Leavenworth in converting the VIC 1.2 reference model to the SUN 4 computer. VIC is a corps/division level model used for combat developments. The VIC model was developed as an analysis tool used on the Digital Equipment Corporation family of computers. Conversion of the model to the SUN 4 provides the analysis community with an alternative computer resource for work requiring the use of VIC.

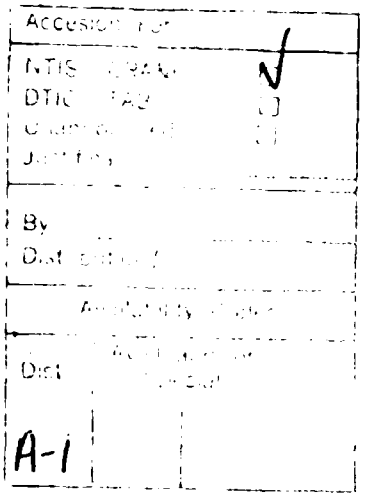
20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL MIKE HANNON	22b. TELEPHONE (Include Area Code) (913) 684-5418 22c. OFFICE SYMBOL ATRC-FM

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE

ACKNOWLEDGEMENTS

While Vector-in-Commander (VIC) was being converted to the SUN 4 a number of individuals were working on converting VIC to other computer systems. Jim Blaise from TRADOC Analysis Commands-White Sands Missle Range (TRAC-WSMR) was working with VIC on the Hewlett Packard, Rich Sandemeyer from the Army Materials Systems Analysis Agency (AMSAA) was converting VIC to run on the CRAY II, and from RAND, Jim Gillogly along with Walt Hobbs were converting VIC to a SUN 3. These individuals were very helpful in passing on suggestions about errors encountered in their work and possible corrections.

Within TRAC-FLVN, Terry Gach was a valuable resource in providing information on SUN/UNIX commands and possible reasons for model problems. Gary VanKuiken and John Lance were instrumental in providing the system support required with the SUN 4 computers.



CONTENTS

ABSTRACT

This document describes work completed at the TRADOC Analysis Command-Fort Leavenworth in converting the VIC 1.2 reference model to the SUN 4 computer. VIC is a corps/division level model used for combat developments. The VIC model was developed as an analysis tool used on the Digital Equipment Corporation family of computers. Conversion of the model to the SUN 4 provides the analysis community with an alternative computer resource for work requiring the use of VIC.

VIC 1.2 MODEL CONVERSION FROM THE VAX TO THE SUN 4

1. Background. The Vector-in-Commander (VIC) model was developed to represent combat in a combined arms environment at the Army's Corps-Division level. VIC is currently used as an analysis tool for combat development studies and scenario generation.

a. VIC was developed on the Digital Equipment Corporation family of computers using the VMS operating system. Most of the model is written in the SIMSCRIPT II.5 programming language developed for the VAX. A much smaller portion of the model consisting of utility routines and graphics interface routines, is written in FORTRAN. New releases of the model are distributed to the user community after significant enhancements are made. The latest release of VIC (version number 1.2) consists of over 1,830 routines.

b. Due to the large number of study requirements for VIC and the lack of adequate VAX computer resources available within the TRADOC Analysis Command-Fort Leavenworth (TRAC-FLVN), the Model Support Directorate began an effort to convert VIC to the SUN 4. The approach taken was to convert the model without worrying about converting those routines which allowed for graphics display during game execution. VIC graphics are currently designed to display on RAMTEK hardware.

2. Computer Characteristics.

a. The SUN 4 computer that was used in the conversion effort for VIC had the following characteristics:

- SUN 4/260 using RISC technology with the Scalable Processor Architecture (SPARC).
- C compiler.
- networking software (terminal communication protocol/ interface protocol (TCP/IP), network file server (NFS)).
- SunOS Release 4.0 UNIX operating system (a necessary requirement since earlier levels of the operating system did not allow for the large number of open files actively accessed by VIC).

- 260 megabytes of disk storage. For any large study effort with VIC, this alone will not be enough disk storage. Larger resident disks or use of a network file server scheme will be required.

- 32 megabytes of real memory.

- 66 megabytes of swap memory allowed (changed from 32 megabytes) to handle a large run of VIC and allow other processes to run. Only 41 megabytes for swap memory need be allocated if only VIC is being run.

- Cartridge tape drive.

a. The following additional software packages were used:

- SIMSCRIPT II.5 release 1.0 provided by C.A.C.I. for the SUN 4.

- FORTRAN compiler, libraries and linker used with the SunOS 4.0 operating system.

- Evaluation copy of network software (TCP/IP and NFS) for the VAX, obtained through the Wollongong company, allowing for --

  -- Remote logins (telnet) either from the SUN 4 to the VAX or from the VAX to the SUN 4.

  -- File transfer protocol (ftp) for getting files from the VAX to the SUN 4 or vice versa.

  -- The capability to mount a VAX disk to the SUN, enabling the VAX to act as a network file server (NFS) to the SUN 4.

3. Model changes.

a. Code Changes. The SUN 4 SIMSCRIPT compiler appeared to provide better code and syntax checking than the VAX SIMSCRIPT compiler. In compiling on the SUN 4 a number of warnings appeared which did not occur during the VAX compile. Model code related to these warnings were corrected as appropriate. Warnings and examples of corrections are as follows:

- warning 1002 missing ')'

i.e.: WRITE TB.FEATURE.NUMBER(TB.OBSTACLE.NUMBER(.NUMBER)
was corrected to
WRITE TB.FEATURE.NUMBER(TB.OBSTACLE.NUMBER(.NUMBER))

- warning 1007 number of subscripts different from definition or previous use

i.e.: RESERVE FI.PROB.DETECTION.TABLE(*,*) AS .NR.TABLES
was corrected to
RESERVE FI.PROB.DETECTION.TABLE(*,*) AS .NR.TABLES BY *

(1) Pagefeed characters (^L) in the code were not handled by the SUN 4 SIMSCRIPT compiler. A large number of these had to be removed from the decision table module (dt) code.

(2) The SIMSCRIPT compiler 1.0 release for the SUN 4 does not support the checkpoint/restart feature found with the VAX SIMSCRIPT compiler. Consequently, OPEN statements, which indicated files could be repositioned in the instance of a restart, required modification as follows:

```
OPEN SS.EXTERNAL.EVENT.FILE, INPUT, RECORDSIZE=132, REPO
```

The repositionable feature was commented out, thus:

```
OPEN SS.EXTERNAL.EVENT.FILE, INPUT, RECORDSIZE=132  '',
REPO
```

The RECORD and RESTORE statements associated with a checkpoint/restart were also commented out. However, C.A.C.I. indicated that the next release of the SUN 4 SIMSCRIPT compiler should support the checkpoint/restart feature.

(3) A number of PREAMBLE changes were required as a result of work that Los Alamos Laboratory conducted during conversion of VIC to a UNIX system. Los Alamos had modified certain utility routines which called VAX system runtime routines to instead call UNIX-based routines when the variable ..VMS was set appropriately. The variable ..VMS was defined to mean 0 in the PREAMBLE to indicate the model was operating under a UNIX-based system. Normally with the VIC 1.2 release ..VMS is defined to mean 1, indicating a VMS system is being used. Another change suggested by Los Alamos was to define PROMPT.V as a text variable in the PREAMBLE (this system variable is not currently supported under the SUN 4 compiler) and to comment out the inword placement of global variables which hold the values indicating placement of events in the EV.S set. For example,

```
THE SYSTEM
HAS A AT.FIRING.PERMISSION IN WORD 3151
HAS A I.AT.FIRING.PERMISSION IN WORD 3152
HAS A AD.TARGET.CHECK IN WORD 3153
HAS A I.AD.TARGET.CHECK IN WORD 3154
                    .
                    .
                  <etc>
```

became:

```
THE SYSTEM
HAS A AT.FIRING.PERMISSION IN WORD 3151
''HAS A I.AT.FIRING.PERMISSION IN WORD 3152
HAS A AD.TARGET.CHECK IN WORD 3153
''HAS A I.AD.TARGET.CHECK IN WORD 3154
                    .
                    .
                  <etc>
```

The modifications associated with the VIC PREAMBLE for ..VMS and PROMPT.V were also incorporated into the UTILITY (UT) SIMSCRIPT PREAMBLE.

(4) The calls to FORTRAN routines were somewhat tricky in that FORTRAN expected real arguments to be passed down from SIMSCRIPT as single precision. This only became a problem when SIMSCRIPT did not pass a local real variable directly to FORTRAN. Real variables passed between SIMSCRIPT routines are passed as double precision. Thus, passing a real variable from one SIMSCRIPT routine to another SIMSCRIPT routine, then on to a FORTRAN routine caused the variable to enter into the FORTRAN routine as double precision but was used by FORTRAN as single precision. The appropriate correction was to ensure that all real variables being passed to FORTRAN routines from SIMSCRIPT were first set to local real variables in the SIMSCRIPT routines. The SIMSCRIPT routines changed were UT.COMPUTE.FRAC.OF.CIRCLE.INSIDE, UT.IS.LOCATION.INSIDE.POLYGON and UT.IS.POINT.INSIDE.POLYGON. An example of the change, using code from routine UT.IS.POINT.INSIDE.- POLYGON follows:

```
DEFINE .X.ARG, .Y.ARG AS REAL VARIABLES
''previously .x and .y were sent as arguments
''put .x and .y values in local real variables and send
''local real variables to fortran routine contour instead
LET .X.ARG = .X
LET .Y.ARG = .Y
```

```
CALL CONTOUR(.NR.VERTICES,.VERTICES(1,*),.VERTICES(2,*),
             X.ARG, .Y.ARG, .ANSWER)
```

(5) As previously stated graphics routines were ignored for the immediate work. The following FORTRAN routines relating to graphics or VAX system calls were stubbed out.

INIT_FONTS
RAMTIME
HOWDRAW
DRAW_SYMBOL
MAP
MAKE_HARD_COPY
KKCOLOR
KRMINIT
KNBATCH
KOERASE
KCIRCLE
KLINTEX
VSHCUR
IMG_EXIT
IMG_INIT
DOROT
VSHGCUR
HMOVE
HTEXT
KRECT
K_TOGGLE_FILE_WR
GRAPHICS_INIT
INITFT
KERASE
KBATCH
TIME

(6) As with any large model there are always some errors uncovered as the model is reviewed. One such error was that the global array CO.TRAFFIC.LOAD was not defined as a 3-dimensional real array in the PREAMBLE. This was corrected and should be part of the next release of the model. Los Alamos also noticed that the variable AT.MUN.ROUNDS.FIRED.FROM.ALL.UNITS should be declared as a real variable in the PREAMBLE.

(7) The code changes for the conversion have been forwarded to the VIC model proponent, TRAC-White Sands Missle Range (TRAC-WSMR).

b. Data file changes. The data files used in testing the conversion of VIC to the SUN 4 were the unclassified division data base files provided with the reference version 1.2 release. VIC has two types of data files. The first type has data interspersed with comments, to help in understanding the data content and format. The second is the same as the first except all comments are stripped out using a preprocessing program. The preprocessed files are used as input to the model. All documented data files were preprocessed before testing began to ensure the data matched between the two types of files. The only data file change required was to alter the control file (i.e.: division.ctl). The control file is used to notify the model as to which directory given module data can be found. The data for each module was defined in this file. Module locations were specified in the control file, thus:

```
FL   /home/vicdev/data/division.fl
FI   /home/vicdev/data/division.fi
EW   /home/vicdev/data/division.ew
                .
                .
              <etc>
```

Note that file location is given in UNIX format.

4. Building the running model.

a. Compilation. The VIC model was compiled two ways for testing -- with subscript checking on and with subscript checking off.

(1) With subscript checking on, the following commands were used:

```
simc -cCg PREAMBLE.sim

simc -vcCglx PREAMBLE.sim $module.sim >$module.lis
```

Simc is the SIMSCRIPT compiler command. Module is a particular module being compiled (i.e.: FL, DT, etc). Options listed with this command are:

v - Do not generate object file for PREAMBLE.
c - Do not link any object files after compilation.
C - Perform subscript (runtime) checking.
g - Include code to provide detailed traceback information.

l - Display a routine-by-routine listing.
x - Display a local cross-reference listing.

(2) With subscript checking off, the following commands were used:

```
simc -cO PREAMBLE.sim

simc -vcO PREAMBLE.sim $module.sim
```

where the O option invokes the C compiler optimizer.

(3) Note that the output from model runs on the SUN 4 were compared using these two methods to verify that the compilation option used created no differences. This was done to double check the SIMSCRIPT compiler which is still undergoing a maturing process, for use on the SUN 4.

(4) After compilation the object files were placed in an archive library thus:

```
ar rvl  vicon.a   $module.o
```

where vicon.a (vicoff.a if subscript checking off used) is the archive library. The options listed with the archive (ar) command are as follows:

r - Replace file in library.
v - Verbose (show what is happening to archive library).
l - Place temporary files generated in this directory.

(a) The SIMSCRIPT utility routines were compiled just as above but were placed in a seperate archive library called util.a (utoff.a if compiled with subscript checking off).

(b) FORTRAN routines were compiled, thus:

```
f77 -c -C  routines.f    -o routines.o
```

with options being:

c - Do not link objects created
C - Do bounds checking during runtime
o - Name of object created follows

The FORTRAN objects were placed in the utility archive library with the SIMSCRIPT utility routines.

(5) When all libraries were completed, a table of symbol definitions (to ease linking) were placed in the libraries using the following command:

```
ranlib vicon.a
```

Anytime an object module needed to be replaced (due to code changes) the table of symbol definitions was rebuilt using the ranlib command.

(6) It should be noted that the SUN 4 SIMSCRIPT compiler actually does a translation of SIMSCRIPT souce code to C language code, then the C compiler is invoked to complete the compilation to object level.

b. Link commands. Since the SIMSCRIPT source code is translated to C language source, the compiler/loader command shell that C.A.C.I. provided invoked the link loader after compilation using the SUN 4 C language compiler command. Several loader options are required when linking VIC using the C.A.C.I.-provided shell. To make linking VIC easier, a copy of the C.A.C.I. compiler/loader shell was modified for VIC use. This command shell for linking VIC has the following structure:

```
ar xv  vicon.a  main.o
cc -Bstatic                    \
   main.o                      \
   vicon.a  util.a             \
   -lsim -lF77 -lI77 -lm       \
-temp=.                        \
-o      vicon.out
rm main.o
```

The -l option indicates the named library should be found in what the system considers standard library locations. Thus either the SIMSCRIPT library (libsim.a) needs to be put in one of these locations or the location of the library needs to be explicitly specified with a path (i.e.: /home/hannon/libsim.a). The libraries required for FORTRAN are also placed in the above command stream. All named libraries in the above command shell, and by default C libraries are used to resolve routine references during the link load.

Dynamic linking did not work with the SIMSCRIPT 1.0 release. Therefore, static linking was used. The difference between these two options for linking is having shared executable code invoked at runtime (dynamic) or making all executable pieces part of the VIC executable (static).

The -temp=. option forced all temporary files to reside in the current directory. This assumes the current directory's top level has enough storage space available.

c. Command file changes for running VIC. The command file for running VIC on the VAX made a number of logical file assignments before starting execution of the model. These VAX assignments were replaced with UNIX links. An example of the UNIX shell for running VIC follows:

```
# shell script for running vic
#
# change directory to location where model output is desired
cd /home/output
#
# establish links
# remember ctl file modified to have location of input files
ln /home/input/division.ext    SIMU02
ln /home/input/division.ctl    SIMU50
#
# run model with input from shell for 12 hour test
# end of input found with !, standard output goes to logfile
/home/model/vicon.out  <<!   >logfile
N
EH GA AG AD AT AM AH AA AI DT FL RD DS
WT TB EN SM MF LO GI FI EW CO CH PT
PROCEED
Q
E
E
SI
SI
SI
SI
SI
SI
SI
SI
SI
SI
SI
SI
!
# echo ' -- MODEL RUN COMPLETE '
```

(1) The model output will show up as SIMUXX files (i.e.:SIMU13). These can be renamed using the UNIX mv command (i.e.:mv SIMU13 post.lis) or can be accessed with the correct

output file names using a shell which sets up logical connections using the UNIX ln command as shown above.

(2) If the model is accessing input and exporting output to a VAX disk using the WOLLONGONG network file server software, the link (ln) command does not work. In this case, a straight copy can be used in place of the ln command, thus:

```
cp /home/input/division.ext    SIMU02
cp /home/input/division.ctl    SIMU50
```

d. System changes. When running the model with a very large data base, it was found that the SUN 4 system allocation required for swap memory needed to exceed 32 meg. This was necessary, not only because of the swap memory used by the model, but also because windows such as the SUNVIEW texteditor, clock, etc., use swap memory. To get around this problem a little overkill was used, and the swap memory allocation was set up to 66 meg.

5. Comparison of results.

a. The model results were compared for a number of test cases. The first test consisted of comparing runs on the SUN 4 when the model was compiled with subscript checking on, versus a run where the model had subscript checking off and the C optimizer was used when compiling. No differences were found.

b. A second test was designed to verify that the network software was functioning as advertised. A SUN 4 VIC executable was run with input data files residing on the SUN and output from the model going to a SUN directory. Then, the same executable was run on the SUN 4 with the input data files residing on the VAX and output from the model going to the VAX. Again no differences were found.

c. The benchmark test of the 1.2 reference version running on the SUN to that same model running on the VAX, consisted of comparing postprocessor output files, i.e., kills by weapon category, etc. To cut down on possible items which had the potential to effect the results seen, both the SUN 4 VIC 1.2 executable and the VAX VIC 1.2 executable ran from the same input files (residing on the VAX). Also, the same postprocessor executable (residing on the VAX) was used with output from both VIC runs. Even though the postprocessor at TRAC-FLVN has been converted for use on the SUN 4, for purity of test results, it was not used in the benchmarking procedure.

d. The VIC model provides the capability to execute only selected modules, so the test was conducted in an incremental

fashion. When initial modules were executed such as direct fire and artillery, results were identical. However as more modules were added results began to differ slightly. In exploring why this occurred it was found that the sequencing of model events were not always in the same order when comparing the VAX and SUN runs. The root of the problem was that VIC 1.2 has its' basic unit of simulation time defined in terms of days. When the VIC model scheduled various events/processes to occur at certain minute times (i.e.: artillery every 15 minutes, intelligence sensors on and off on some minute cycles) the model was forced to convert the scheduled time to an equivalent in days. The floating point precision between the SUN and the VAX differed on these scheduled times, thus causing certain events/processes to occcur in a different sequence. This caused differences in model results. To demonstrate how the event/process ordering differed between the two computer systems, the following test program was run:

```
PREAMBLE

EVENTS INCLUDE
EV.ONE,
EV.THREE

PROCESSES INCLUDE
PR.TWO,
PR.FOUR

PRIORIY ORDER IS
EV.ONE,
PR.TWO,
EV.THREE,
PR.FOUR
END.SIMULATION

END

MAIN

SCHEDULE A EV.ONE     IN 2 MINUTES
ACTIVATE A PR.TWO     IN 1 MINUTES
SCHEDULE A EV.THREE   IN 2 MINUTES
ACTIVATE A PR.FOUR    IN 1 MINUTES
SCHEDULE A END.SIMULATION IN 20 MINUTES

START SIMULATION

STOP
END
```

```
EVENT EV.ONE SAVING THE EVENT NOTICE
WRITE AS " ENTERED 1",/
RESCHEDULE THIS EV.ONE IN 2 MINUTES
RETURN
END

PROCESS PR.TWO
UNTIL 1=2, DO
  WORK 1 MINUTE
  WRITE AS " ENTERED 2",/
  WAIT 1 MINUTE
LOOP
RETURN
END

EVENT EV.THREE SAVING THE EVENT NOTICE
WRITE AS " ENTERED 3",/
RESCHEDULE THIS EV.THREE IN 2 MINUTES
RETURN
END

PROCESS PR.FOUR
UNTIL 1=2, DO
  WORK 1 MINUTE
  WRITE AS " ENTERED 4",/
  WAIT 1 MINUTE
LOOP
RETURN
END

EVENT END.SIMULATION
STOP
END
```

(1) The design of the above program was based loosely on artillery code and intelligence code found in VIC. These modules were the ones under investigation when the ordering problem was uncovered. Since in the above program events are on a 2 minute cycle and processes are waiting 1 minute then working 1 minute the priority ordering should force the write statements to output every 2 minutes of simulation time in the following order (always!):

```
ENTERED 1
ENTERED 2
ENTERED 3
ENTERED 4
```

(2) Because of floating point precision in converting minutes to days this does not happen. Results from running the above program on the various systems were as follows:

```
     VAX SYSTEM                    SUN SYSTEM

     ENTERED 1                     ENTERED 1
     ENTERED 2                     ENTERED 2
     ENTERED 3                     ENTERED 3
     ENTERED 4                     ENTERED 4

     ENTERED 1                     ENTERED 1
-->  ENTERED 3                     ENTERED 2    <--
-->  ENTERED 2                     ENTERED 3    <--
     ENTERED 4                     ENTERED 4

     ENTERED 1                     ENTERED 1
     ENTERED 2                     ENTERED 2
     ENTERED 3                     ENTERED 3
     ENTERED 4                     ENTERED 4

     ENTERED 1                     ENTERED 1
     ENTERED 2                     ENTERED 2
     ENTERED 3                     ENTERED 3
     ENTERED 4                     ENTERED 4

     ENTERED 1                     ENTERED 1
     ENTERED 2                     ENTERED 2
     ENTERED 3                     ENTERED 3
     ENTERED 4                     ENTERED 4

     ENTERED 1                     ENTERED 1
     ENTERED 2                     ENTERED 2
     ENTERED 3                     ENTERED 3
     ENTERED 4                     ENTERED 4

-->  ENTERED 2                     ENTERED 2    <--
     ENTERED 4                     ENTERED 4
     ENTERED 1                     ENTERED 1
     ENTERED 3                     ENTERED 3

     ENTERED 2                     ENTERED 2
     ENTERED 4                     ENTERED 4
     ENTERED 1                     ENTERED 1
     ENTERED 3                     ENTERED 3
```

ENTERED 2	ENTERED 2
ENTERED 4	ENTERED 4
ENTERED 1	ENTERED 1
ENTERED 3	ENTERED 3
ENTERED 2	ENTERED 2
ENTERED 4	ENTERED 4

Note that the VAX ordering becomes incorrect early on causing a difference between the ordering shown with the SUN 4. Later the SUN 4 ordering gets out of sequence but matches the VAX incorrect ordering at this time. As the simulation is run longer the ordering of events between the two systems will continue to differ at various times. Perhaps not so obvious from this test when looking at the results is that the end.simulation event occurred before events ev.one and ev.three. This should not have happened given the priority order set up in the preamble, but is again caused by the floating point precision problem when converting minutes to days. While other precision differences in executing the VIC code on the two systems will cause result differences, tracking these are very difficult given that game time events occur at different times between the systems.

e. A production version of VIC being used for the Close Combat Capabilities Analysis (C3A) study was converted to the SUN 4 to get a feel for not only how significantly the results differed, but also to find out how run speed compared between the VAX 8600 and the SUN 4. The model outputs were analyzed by the C3A study team who found that results matched in some areas and were 5 percent or less off in other areas. The study team concluded that the SUN 4 results were within an acceptable tolerance as differences tended to compensate in the aggregate. Given this the study team recommended use of VIC on the SUN 4.

6. Conclusion.

a. Under the current SIMSCRIPT 1.0 compiler release there is quite a penalty in run speed when running with subscript checking on. Mr. Paul Close from C.A.C.I., who generated the SUN 4 SIMSCRIPT compiler, stated the next compiler release should impose far less of a run speed penalty when running with subscript checking on. Additionally, features such as checkpoint/restart should be available.

b. When VIC was run on the SUN 4 with the C3A data base for a 48-hour simulation time, it completed in 8 hours and 56

minutes. The same run on the VAX 8600 took about 18 hours. Work going on at Los Alamos with VIC on the SUN 4 and further work here at TRAC-FLVN has the potential to increase the run speed even more.

c. Any questions regarding the VIC 1.2 conversion effort at TRAC-FLVN should be directed to Mr Mike Hannon, TRAC-FLVN, AV 552-5418/5419.

DISTRIBUTION LIST

	No. Copies
Defense Technical Information Center ATTN: DTIC, TCA Cameron Station Alexandria, VA 22314	2
US Army Library Army Study Documentation and Information Retrival System (ASDIRS) ANRAL-RS ATTN: ASDIRS Room 1A518, The Pentagon Washington, D.C. 20310	1
US Army TRADOC Analysis Command-WSMR ATTN: ATRC-WSL (Technical Library) White Sands Missile Range, NM 88002-5502	1
US ARMY TRADOC Analysis Command-FLVN ATTN: ATRC-FOA (Technical Info Center) Fort Leavenworth, KS 66027-5200	1
US Army Combined Arms Research Library (CARL) ATTN: ATZL-SWS-L Fort Leavenworth, KS 66027-5000	1
Commander US Army TRADOC Analysis Command ATTN: ATRC Fort Leavenworth, KS 66027-5200 Thru: Director, TRAC-Fort Leavenworth ATTN: ATRC-F Fort Leavenworth, KS 66027-5200	1