SLAC-PUB-2302
April 1979
(M)

SIMULATION OF LSI-11/PDP-11 SERIES MINICOMPUTERS

J. R. Myers, R. L. A. Cottrell, B. M. Bricaud
Stanford Linear Accelerator Center
P. O. Box 4349
Stanford, California 94305

ABSTRACT

A functional simulation of the PDP-11 series minicomputers has been implemented to run either interactively or as a batch job on an IBM 370 computer. The simulator operates in two modes, the supervisor mode and the run mode. In the supervisor mode, the simulator implements a command language, which allows users to examine and change the contents of memory, or other addressable registers in the simulated machine. Optionally an instruction trace may also be turned on or off. In the run mode, the simulation of the intruction set has been tested, by successfully running DEC's MAINDEC basic instruction test on the simulated machine.

The interrupt structure is modeled. The simulation is open ended in the sense that users may define new peripheral devices, by including their own FORTRAN callable subroutines for each simulated device. Currently the following devices are supported: floppy disks, a console terminal, a card reader, a card punch, a line printer and a communications multiplexer (DH11). With these devices DEC's RT-11 versions 2C and 3B have been successfully run on the simulator. At SLAC this simulator is proving useful in debugging software for one of a kind hardware configurations, such as communications front end processors, that are not readily accessible for stand alone testing.

INTRODUCTION

At the Stanford Linear Accelerator Center (SLAC), over the last 4 years, many LSI-11/PDP-11 stand alone systems have been developed to provide such diverse functions as:

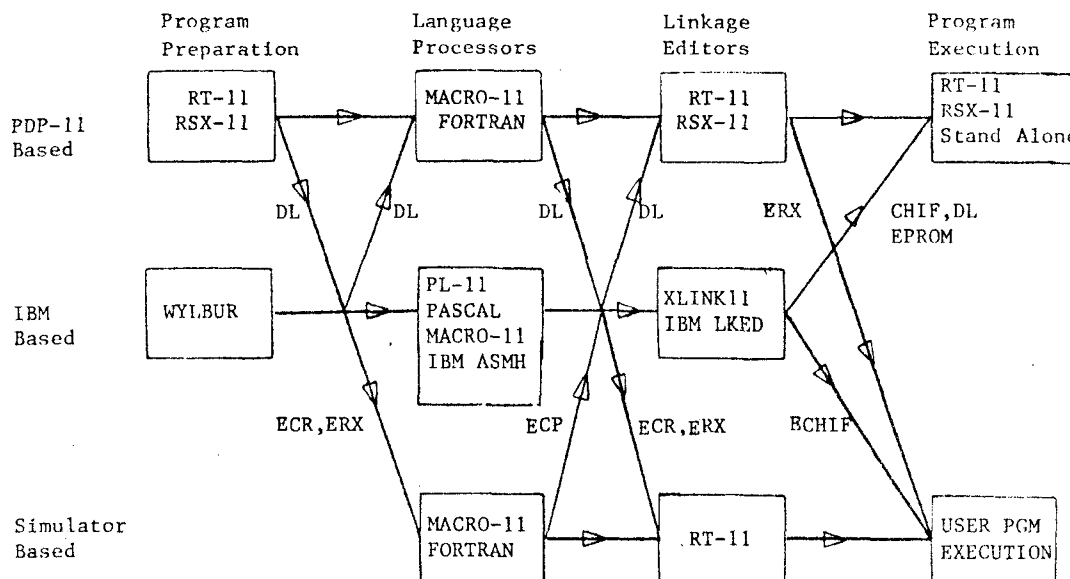
- testing a hardware random bit generator;
- closed loop feedback position and energy steering of a 20GeV electron beam;
- terminal communication multiplexing to a large IBM 360/370 computer complex (the TRIPLEX);
- testing of detectors for high energy physics experiments.

To support the above systems we have developed a System 370 software package (Ref. 1) to support LSI-11/PDP-11 program development. This package includes a PL-11 cross-compiler (Ref. 2), a PASCAL cross-compiler (REF. 3), a MACRO-11-like cross-assembler, a set of macros to enable the IBM ASMH assembler to generate PDP-11 object code, a cross-linker (XLINK11), various utilities to aid in file formatting and examination, downline loading, and file transfers between DEC's RT-11 operation system and the TRIPLEX. Recently we have implemented an LSI-11/PDP-11 simulator, which enables us to extend the program development functions that can be performed on the TRIPLEX, to include execution and hence, debugging, of the binary program. The interrelations between the various processors is shown in Fig. 1. Also shown is the "device" via which the modules (source, object, load) are transferred between the processors.

The technique of using a large computer center to develop programs for small computers has several advantages, namely:

- one has access to sophisticated I/O devices (microfilm, tapes, graphics, line printers, etc.);
- data set management, e.g., archival, backup, cataloguing, are automatically provided;
- makes the small computer configuration very simple since it does not have to support code development;

Contributed to the DECUS Spring Mini/Midi Symposium, New Orleans, Louisiana, April 17-20, 1979.



LEGEND: CHIF = IBM Channel to PDP UNIBUS Link
DL = Serial Line Link
ECP = Emulated Card Punch
ECR = Emulated Card Reader
ERX = Emulated Floppy Disk
ECHIF= Emulated CHIF

Fig. 1 Block diagram depicting the interrelation of PDP-11 program development tools at SLAC

> users only have to learn one editor (in our case WYLBUR) and operating system (in our case IBM's SVS);

> multiple users can develop code simultaneously;

> capital equipment does not sit idle when no users are developing code;

> the system can be accessed via telephone or hardwire line from any site, including the home.

In addition to these general advantages, a simulator provides several features, which are complementary to the features obtained by executing, and debugging the program on a real machine. Such features include the following abilities:

> A trace is provided that prints the contents of the registers, etc. after each instruction. Traces are especially useful when they can be scanned by a powerful text editor. They are also an excellent training tool.

> The code path lengths are simply determined.

> The execution can be precisely reproduced, in particular, the user can specify when and if interrupts are to occur.

> One of a kind special purpose hardware and production systems which may not be accessible for debugging may be emulated.

USER INTERFACE

To the user the simulator appears to have two modes, the supervisor mode and the run mode. The supervisor mode implements a command language (see Fig.2) that allows the user to perform LSI-11 ODT or console switch register functions such as: accessing and changing the contents of memory locations and registers, issuing RESET, and initiating program execution. Additional supervisor commands allow the user to set and clear breakpoints, set execution time limits, load and insert device emulators in the appropriate UNIBUS addresses, cause interrupts to occur at given interrupt vectors, select the level of error checking and to enable or disable instruction tracing. The instruction trace produces one line of output per instruction executed. This line includes the location of the instruction executed, the instruction in octal and as a 4 character mnemonic, the result of the instruction, the source and destination addresses (if relevant), the PS, condition codes, processor priority and the register contents after the instruction was executed. Instructions to the supervisor may come from the user's terminal or from a console file. The latter facility is used in batch jobs, and is also used in interactive jobs to provide a standard set of commands such as: "toggling in" a bootstrap loader or inserting standard device emulators.

In the run mode the user's terminal (in the case of interactive jobs), or terminal input and output files (in the case of batch jobs), becomes the console terminal for the

BEGIN	Enter the run mode, begin execution.
BOOT	Boot from DQØ and begin execution.
CMS	(Reserved.)
DEPOSIT	Change the contents of a memory cell.
DISPLAY	Show the contents of a memory cell.
DUMP	Dump memory to the line printer.
EXAMINE	(Same as DISPLAY.)
EXIT	Terminate the simulation.
INSERT	Install a device emulator.
INTERRUPT	Trap and begin execution.
MILTEN	Pass the next command to the SLAC monitor.
PROCEDE	(Same as BEGIN.)
RESET	Perform a UNIBUS reset.
SPECIAL	Transfer control to a user-supplied subroutine.
STORE	(Same as DEPOSIT.)
SET LIMIT	Specify the time limit.
SET PRINT	Enable the console log.
SET SPEED	Specify the level of error checking.
SET TRACE	Enable instruction tracing.

Figure 2. Commands available from the supervisor mode of the simulator.

simulator. The simulator in run mode continues execution of instructions until either: the time limit is reached, a HALT is executed, or an error occurs.

SOFTWARE DESIGN

The most convenient way to describe the design of the simulator software is to say that it consists of three types of processors and three sets of control blocks. The processors, which are described in more detail below, are:

> a command line interpreter/executor (the supervisor);

> an instruction executor;

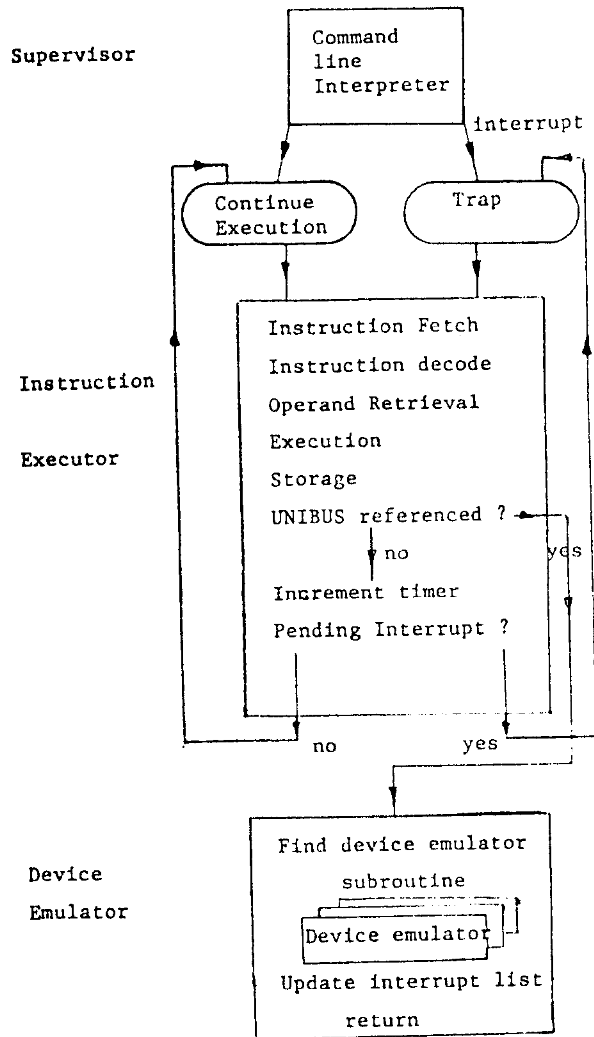


Figure 3. Flow chart of the Simulator.

> multiple input/output device emulators.

The interrelation of the processors is depicted in Fig. 3.

The three control blocks may be thought of as state tables which preserve the status of the simulated machine, the simulator itself, and the software linkages that are reestablished each time the simulator is loaded into the computer. Conceptually, it should be possible to checkpoint the machine state control block whenever there are no pending interrupts, however, the simulator does not have such a capability at the present time.

The supervisor is written primarily in PL/1, a language that has good string handling capabilities. A few small assembler language subroutines are used to access

operating system functions which are not available from the high-level PL/1 language. Particular instances of this are to dynamically load one of three possible instruction executors, or to insert or remove device emulators.

The instruction executor takes control when the simulator is operating in run mode. There are three possible instruction executors. They differ in the amount of error checking and instruction tracing included, and hence in their speed of execution. The user selects which executor is to be used from the supervisor level through such commands as "SET TRACE OFF" or "SET SPEED ON". The executor is written in IBM assembler language. The three versions are obtained from a single source copy by using conditional assembly techniques.

The device emulator subroutines are written specifically for each hardware device required. Most of these subroutines are written in MORTRAN, a structured FORTRAN preprocessor. An example of the calling sequence of a device emulator is seen in Fig. 4. Whenever the instruction executor references a register in the I/O page, the simulator determines whether the user has previously "inserted" a device on the UNIBUS that will respond to that address. If so, then the user-supplied subroutine is called. The user device emulator subroutine may alter the I/O page and memory locations, and, optionally, specify a delayed interrupt and trap vector location. References to nonexistent devices return control to the supervisor along with a diagnostic message. The user may optionally provide a FORTRAN callable subroutine to intercept such non existent memory traps.

We have implemented device emulators for devices such as disks, ASCII terminals, card readers, card punches, and line printers. In addition, primitive drivers requiring user interaction exist for a communications line multiplexer and an IBM channel interface. Provision is also made for unknown user devices (UNK1-UNK9).

TESTING THE SIMULATOR

After the simulator appeared to function correctly, we took DEC's MAINDEC basic instruction test and ran it through the simulator. This revealed several errors mainly having to do with the treatment of condition codes, and certain subtle differences in the way some instructions are executed on various machines in the PDP-11 series. After these errors were fixed we were able to successfully run RT-11 versions 2C and 3B on the simulator. One further error was later fixed in the emulation of the EIS ASH instruction, which is not tested by the basic instruction test diagnostic.

PERFORMANCE

As a general rule, when tracing is disabled, it takes between 20 to 50 IBM machine language instructions to simulate the execution of one PDP-11 machine language instruction. Simulation of the operation of input/output devices may require considerably more instructions than this. However, the input/output subroutines are called only when the simulator accesses a memory location on the I/O page (150000(8) - 177777(8)). We suspect that the instruction count could be reduced by judicious recoding of the addressing mode calculations.

```
      SUBROUTINE CP(BUSADR,TIM,VECADR,PRI)
        INTEGER*4 BUSADR,VECADR,PRI
        REAL*4 TIM
C Provides a card punch device emulator
C
C Input:
C BUSADR=Address of device register. Can
C        contain 1 of 3 values:
C        172470(8)=address of register
C                  containing 2's comp-
C                  element of byte count
C                  to be written;
C        172472(8)=address of register
C                  containing the
C                  buffer address of
C                  the bytes to be
C                  written.
C        0        =this value is passed
C                  at simulator initial-
C                  ization time, CP then
C                  initializes certain
C                  variables.
C Output:
C TIM   =Time (in micro-seconds) after
C        which, on exit from CP, an
C        interrupt is to be fired. If TIM
C        is set to 0.0 then no interrupt
C        will be fired. If TIM is set to
C        -1.0 then the simulator will
C        HALT (ie go into supervisor mode).
C VECADR=Address to which the interrupt
C        will vector when it fires.
C PRI   =Priority at which the interrupt
C        will fire, 1<=PRI<=7.
C
      COMMON /MEM11A/FILL,MEMORY,UNIBUS
        INTEGER*2 FILL(156)
        LOGICAL*1 MEMORY(57344),
     1            UNIBUS(8192)
C FILL   Contains system dependent
C        variables that the user does
C        not usually access.
C MEMORY Contains the current contents of
C        memory of the simulated machine.
C UNIBUS Contains the current contents of
C        the I/O page of the simulated
C        machine.
```

Figure 4

Listing of part of the card punch device emulator to show the information that is accessible to the subroutine.

EXAMPLES OF USE

1) RT-11 FORTRAN

One use of the simulator has been to provide the equivalent of a cross- Fortran compiler. This was desirable in order to allow physicists to write LSI-11 numerical applications in a language they are familiar with. At the same time, however, they did not wish to have to learn a new editor and operating system. It was therefore decided to run the RT-11 FORTRAN compiler under the simulator, and then link the object modules produced by it with other MACRO-11 and PL-11 modules, using XLINK11 to produce a downline loadable module.

To install RT-11 under the simulator the following steps were performed:

- > We created on the TRIPLEX two FORTRAN direct access files DQØ and DQ1, which have the same record length as RT-11 blocks (512 bytes) and the same number of records (494) as there are blocks on an RXØ1 floppy diskette.
- > We wrote an RT-11 device handler to access the DQ files.
- > We modified the RT-11 bootstrap to access the DQ file and updated the RT-11 monitor device tables. Then we generated, on a real LSI-11, a new RT-11 monitor with this bootstrap and the DQ device handler.
- > We transferred the contents of the two floppies containing the new RT-11 monitor and other processors like FORTRAN, PIP, LINK, LIBR, ETC., from the LSI-11 to the DQØ and DQ1 files.
- > We wrote a simple device emulator to perform I/O for the DQ files.
- > Finally, we "toggled" in the initial bootstrap via the simulator "console".

It was purely for efficiency reasons that we chose to modify the RT-11 system to access the DQ devices, instead of making the device emulator fully emulate all the functions of an RXØ1 floppy drive. Also, for efficiency reasons we removed RT-11'S echoing of characters typed at the terminal, since the terminal echoes characters itself.

The usual way in which the simulated FORTRAN compiler is run is via a prepackaged batch job (see Fig. 5), with the input to the compiler coming from the emulated card reader, printout going to a emulated line printer, and the object modules being written on the emulated card punch file. In a later step of the same job, the emulated card punch output is converted to XLINK11 input format, and saved in an object module library ready for later linking.

```
//RACRTVM  JOB   ,TIME=(2,Ø)
//VM11  EXEC VM11,GORGN=45ØK
//* PRINTER is for the supervisor print
//PRINTER DD SYSOUT=A
//* FTØ9 is the card punch emulator file
//FTØ9FØØ1 DD DSN=WYL.EA.RAC.OM,DISP=SHR
//FT11FØØ1 DD * (specify simulator mode)
 &OPTION BATCH=.TRUE.,&END
//FT12FØØ1 DD SYSOUT=A (printer emulator)
//* FT2Ø and FT21 are the floppy (RXØ1)
//* emulator files. They contain RT-11.
//FT2ØFØØ1 DD DSN=WYL.EA.RAC.DQØ,DISP=SHR
//FT21FØØ1 DD DSN=WYL.EA.RAC.DQ1,DISP=SHR
//PRINT DD SYSOUT=A (term emulator output)
//PROFILE DD *  (supervisor command input)
STORE 157744 Ø127Ø6

   ( Toggle in the bootstrap loader)

STORE 157774 ØØ5ØØ7
SET TRACE OFF
SET LIMIT 3ØØØØØØØ
STORE PC 157744
BEGIN
EXIT
//FTØ5FØØ1 DD * (Card reader emulator)
.TITLE CR:      CR DEVICE HANDLER FOR VM11
; RT-11 DEVICE HANDLER TO INPUT ASCII

   ( Source for a MACRO-11 assembly)

.END
^Z
//FTØ8FØØ1 DD * (terminal emulator input)

 following are RT-11 commands

DATE 19-AUG-78
INSTALL CR:
INSTALL CP:
COPY DK1:USAMPL.OBJ CP:
FORTRAN/LIST:LP:/OBJECT:CP: DK1:USAMPL.FOR
MACRO/LIST:LP: CR:
^C
D 1ØØØ=Ø  (force a HALT from the terminal)
START 1ØØØ
/*
```

Figure 5.

Simplified example of the the input for a simulator batch type job to make an RT-11 FORTRAN compilation followed by a MACRO-11 assembly.

2) Stand-alone terminal concentration code

One of the front end processors for the TRIPLEX is a PDP-11/34 that now supports about 7Ø interactive terminals. The operating system for this machine is a piece of stand alone code that was originally developed at Stanford University. This machine includes several I/O devices that are not available on any other PDP-11 at SLAC. Since the PDP-11/34 is in regular production service around the clock, it is difficult to provide access to it for stand alone program development or debugging. A particular

problem arises when programmers who are unfamiliar with the novel software of this front end processor attempt to add new features or otherwise alter the code.

The simulator has been particularly beneficial in performing initial checkouts of proposed changes to the front end code. Through the use of the simulator we have found two anomolies that might have gone undetected. We were surprised to learn that some initialization code does not run at the highest priority. Unexpected interrupts received during the initialization period could (and may have) caused previously unexplained start up problems. We also discovered that some code that has been added recently cannot be disabled from the console. Due to the inaccessibility of the terminal concentrator, these bugs have not yet been fixed. However we have changed our operating procedures to by pass the problems.

PROBLEM AREAS

No discussion of a software project is complete without some mention of the failures as well as the successes. One problem is the fundamental incompatibility between the half-duplex input/output hardware of the IBM 370 computer and the full-duplex terminal support which is provided by DEC. This incompatibility leads to excessive complexity in the terminal emulation software. It also forced us to abandon an attempt to implement ODT exactly as it is performed by the LSI-11's microcode. We did attempt to dodge this problem by doing unbuffered reads and writes, but the operating system overhead in the IBM software was enormous.

We uncovered a second problem when we found that XLINK11 incorrectly linked object modules produced by the RT-11 Version 2 FORTRAN compiler. This means that currently we can use only object modules produced by RT-11 FORTRAN Version 1 as input to XLINK11.

When running RT-11 in the interactive mode the interaction is most charitbly described as sluggish. This is due to the many processors involved in transferring characters to and from the terminal. These processors include: RT-11, the simulator executor, the device emulator, FORTRAN I/O and the terminal communication processor.

As already mentioned the simulator does not currently support the floating point instructions or memory mapping. This is due to a lack of need for these features in our current work and they could be included if such a need arose.

Although multiple outstanding interrupts are allowed in the simulator, we find that one gets diminishing returns on real time work in general. In particular, the trace of a multiply-threaded program is almost impossible to comprehend. The simulator is used to best advantage to: run in batch mode PDP-11 software (e.g. RT-11 FORTRAN, MACRO-11, or LINK11) on the TRIPLEX; or in the case of real time code, to test (usually interactively) certain sections of the code such as the initialization code, the interrupt handler, or any single logical thread of code.

REFERENCES

(1) R. L. A. Cottrell and C. A. Logg, An IBM 360/370 Software Package for Developing Stand Alone LSI-11 Systems, Proceedings of the Digital Equipment Users Society, Vol. 4, No. 4, pp 985/991 April 1978.

(2) B. L. Hitson, PASCAL/P-CODE Cross-Compiler for the LSI-11, SLAC-PUB-2246 (1979).

(3) R. Russel, PL-11: A Programming Language for the DEC PDP-11 Computer, Edited by T. C. Streater, CERN 74-24 (1974).