Allen Jiajuin Wei

Author

DUES

(Display Utilities and Environments for Simulation)

Title

RESEARCH PROJECT

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, to partial satisfaction of the requirements for the degree of Master of Sciences, Plan II.

Approval for the Report and Comprehensive Examination:

Committee: [illegible], Research Adviser

14 May 1988 Date

[illegible], Second Reader

18 May 1988, Date

Acknowledgement

I would like to thank Professor Yale Patt for his valuable guidance and encouragement. His time and patience throughout this project are greatly appreciated. I would also like to thank Professor Alvin Despain for his valuable comments and reading the paper.

I also gratefully acknowledge my debt to Bruce Holmer, Steve Melvin, Mike Shebanow, and Wen-Mei Hwu who influence the overall design; to Mike Carlton, Chien Chen, Jeff Gee, Tam Nguyen, Ashok Singhal, Jerric Tam, Jim Testa, Jim Wilson, and others who reviewed the work and gave positive feedback; and to Kinson Ho, and Jeff Gee, who read the draft of this report, and gave me valuable comments.

To the friends who have encouraged me and given me moral support throughout my years at Berkeley, I thank Joe Wu, Pehong Chen, Shin-Yuan Tzou and Belle Wei.

Finally, to my late father Wen-Ping Wei and my mother Li-miin Men for their constant loving support and patience for all my endeavor.

DUES

(Display Utilities and Environments for Simulation)

Allen Jiajuin Wei

Master's Report Plan II
Computer Science Division
Department of Electrical Engineering and Computer Science
University of California, Berkeley, California 94720

ABSTRACT

DUES (Display Utilities and Environments for Simulation) is an X Window based interactive tool for running the PLM simulator. It invokes the simulator, and communicates with it through UNIX[†] *sockets*, while providing a better user interface with objects such as menus, buttons, display windows, scroll bars, etc.. It is designed to allow future extension with minimum change to allow interface to other simulators. In this report, the internal structure of DUES is explored in detail. Issues regarding data structures, process communication, and the functionality of each object are examined. Directions for future work are also recommended.

May 16, 1988

† UNIX is a registered trademark of AT&T Bell Laboratories in the USA and other countries

Table of Contents

Chapter 1

Introduction

DUES (Design Utilities and Environments for Simulation) is a front-end user interface for the Prolog Machine (PLM) simulator [1], which is part of the Berkeley Aquarius Project [2,3]. The PLM simulator runs programs of compiled Prolog code [4,5] and provides facilities to allow observation of internal machine states and control of running programs. Two levels of simulation are provided which differ only in the granularity of operations simulated. The Level 1 simulator performs simulation at the macro-instruction level while the Level 2 simulator runs one microinstruction at a time. DUES is interfaced to the level two simulator.

The goal of the DUES project is to provide a user friendly interface [6] to the PLM simulator. DUES has been designed in a clean way in order to interface to other simulators with minimum efforts. The PLM simulator has a primitive user interface. Users usually enter commands to the simulator by typing commands on a character terminal. Frequently, it is found inconvenient to memorize various command names. Besides, because the size of the screen on a terminal is too small, it is hard to keep all information provided by the simulator before it is scrolled off the screen.

From the need to haye a nice user interface, DUES implements an environment consisting of a command menu bar, dialogue boxes, pull-down menus, scroll bars, display windows for internal states, program windows, and other objects. In addition to supporting all the commands in the PLM simulator, it contains a few enhancements, and provides room for future extension. All of these features have become viable due to the arrival of workstations with bitmap high resolution displays, mouse devices, and window systems. DUES is now running on microVAX stations and Sun workstations, and is able to be used with any other workstation with X window systems under 4.3BSD UNIX or its variants.

This report is divided into six chapters and two appendices. A brief introduction to the X Window system is given in chapter 2. Chapter 3 presents an overview of DUES. Chapter 4 discusses major design issues. Chapter 5 describes how to interface DUES to other simulators. Chapter 6 offers some concluding remarks. Appendix 1 contains the DUES user's manual. Appendix 2 lists indices of all DUES procedures and the files they are contained.

Chapter 2

X Window System

X [7] is a network transparent window system that supports overlapping windows under 4.3BSD UNIX, ULTRIX-32, VAX/VMS and many other operating systems [8]. A window is an area on the display screen associated with an application. In X, windows are organized in a strict hierarchy. At the top of the hierarchy is the "root" window, which covers the entire display screen. All windows have parents except the root window. Parent windows are partially or completely covered by child windows. A child window may be larger than its parent; part or all of the child window may extend beyond the boundaries of the parent. However, all output to a window is clipped by the boundaries of its parent.

X can be described in three parts: the X display server, Xlib, and the X Protocol. The X display server runs on computers (usually workstations) with either monochrome or color bitmap displays. The server distributes user input from the keyboard or the mouse device to and accepts output requests from various client programs located either on the same machine or elsewhere on the network. Xlib is a C subroutine library that application programs use to interface to the X server by means of a stream connection. Although an application program usually runs on the same machine as the X server it is talking to, this need not be the case. The communication between application programs and the X server is defined by a network protocol. Currently, DUES is implemented in protocol version 10 release 4 (X10R4).

The C routines in Xlib are a low-level interface to the X Window System protocol in the sense that they allow drawing of lines, polygons, text strings, etc.. Applications often find it more convenient to work on higher level abstractions like buttons, pull-down menus, pop-up menus, scroll bars, labels, text windows, dialogue boxes, etc.. There are toolkits available that contain libraries of packages layered on top of the X Window System. The implementation of DUES does not use any toolkits. There are two reasons for this decision. First, because the X Window Protocol was undergoing a major change from version 10 to version 11 when DUES was being developed, none of the toolkits were stable enough to build applications on. Second, although toolkits offer more convenience and higher abstraction, flexibility suffers since applications can only create and use objects defined in the toolkits. To better fit the specific requirements of the PLM simulator, DUES builds every object from Xlib.

Chapter 3

Overview of DUES

3.1. Introduction

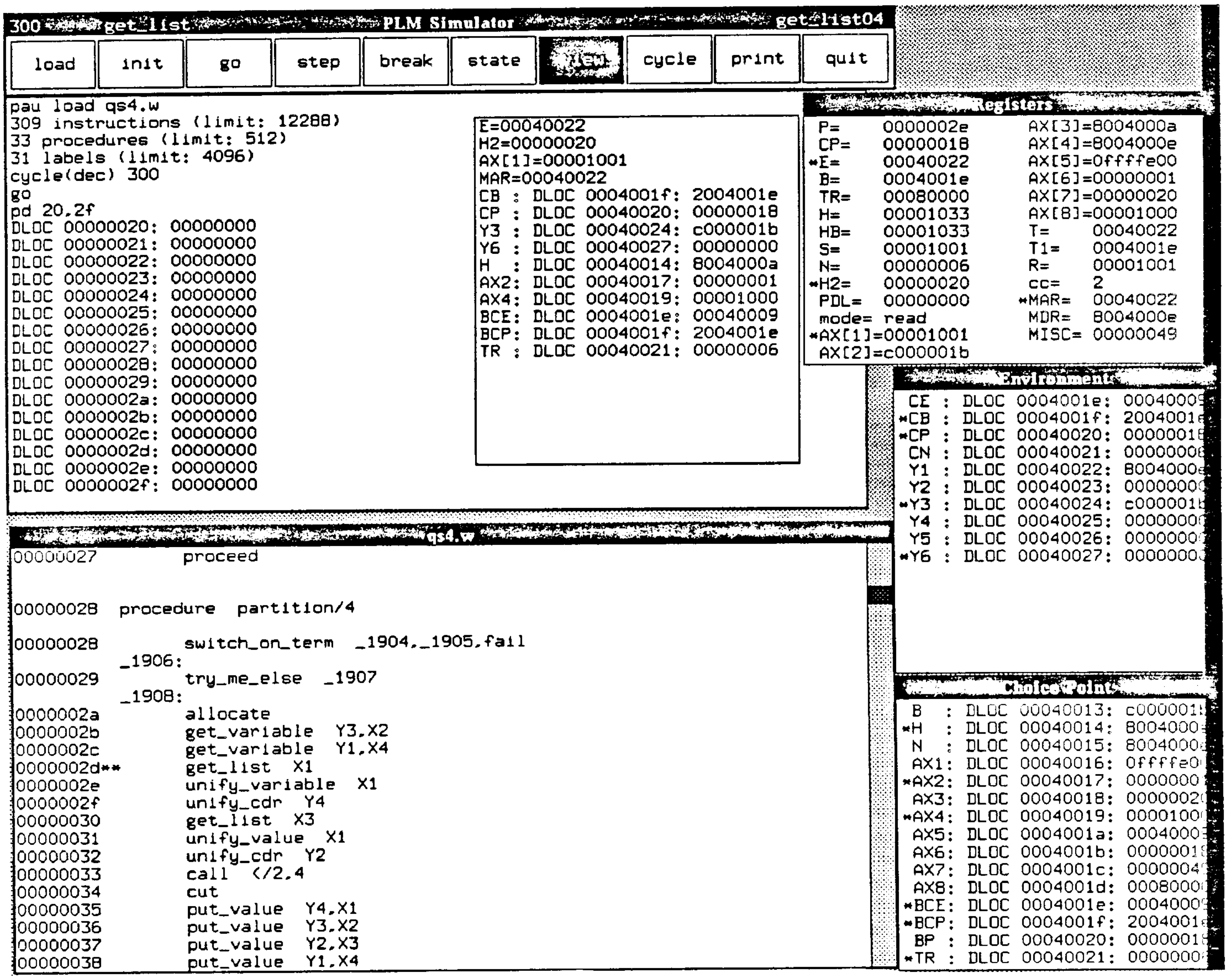
DUES manages seven windows in an attempt to achieve user-friendliness. Each window has its own functionality. All are child windows of *RootWindow*; therefore, they can be manipulated by any window manager. The windows are *Command Menu Bar*, three *State Display Windows*, *Dialog Window*, *View Window*, and *Copy Window*. The following sections will describe all the windows in DUES. Fig. 3.1 shows a typical configuration of the windows.

3.2. Command Menu Bar

The *Command Menu Bar* consists of two parts, the title bar and the command menu. Although it is usually located at the top of the display monitor, it can be put anywhere according to *MenuBarGeometry* specification in the *.Xdefaults* file. Fig. 3.2 shows the *Command Menu Bar*.

The title bar contains several pieces of information. From left to right, there is an integer number indicating the absolute current cycle number counting from 0, the name of the instruction being executed, the title of "PLM Simulator," and the name of the microinstruction being executed. All these except the title will be updated after each cycle is executed.

The command menu has ten commands available, each of which is represented by a



```
pau load qs4.w
309 instructions (limit: 12288)
33 procedures (limit: 512)
31 labels (limit: 4096)
cycle(dec) 300
go
pd 20,2f
DLOC 00000020: 00000000
DLOC 00000021: 00000000
DLOC 00000022: 00000000
DLOC 00000023: 00000000
DLOC 00000024: 00000000
DLOC 00000025: 00000000
DLOC 00000026: 00000000
DLOC 00000027: 00000000
DLOC 00000028: 00000000
DLOC 00000029: 00000000
DLOC 0000002a: 00000000
DLOC 0000002b: 00000000
DLOC 0000002c: 00000000
DLOC 0000002d: 00000000
DLOC 0000002e: 00000000
DLOC 0000002f: 00000000
```

```
E=00040022
H2=00000020
AX[1]=00001001
MAR=00040022
CB : DLOC 0004001f: 2004001e
CP : DLOC 00040020: 00000018
Y3 : DLOC 00040024: c000001b
Y6 : DLOC 00040027: 00000000
H  : DLOC 00040014: 8004000a
AX2: DLOC 00040017: 00000001
AX4: DLOC 00040019: 00001000
BCE: DLOC 0004001e: 00040009
BCP: DLOC 0004001f: 2004001e
TR : DLOC 00040021: 00000006
```

```
 P=     0000002e      AX[3]=8004000a
 CP=    00000018      AX[4]=8004000e
*E=     00040022      AX[5]=0ffffe00
 B=     0004001e      AX[6]=00000001
 TR=    00080000      AX[7]=00000020
 H=     00001033      AX[8]=00001000
 HB=    00001033      T=     00040022
 S=     00001001      T1=    0004001e
 N=     00000006      R=     00001001
*H2=    00000020      cc=    2
 PDL=   00000000     *MAR=   00040022
 mode= read           MDR=   8004000e
*AX[1]=00001001       MISC=  00000049
 AX[2]=c000001b
```

```
 CE : DLOC 0004001e: 0004000
*CB : DLOC 0004001f: 2004001
*CP : DLOC 00040020: 0000001
 CN : DLOC 00040021: 0000000
 Y1 : DLOC 00040022: 8004000
 Y2 : DLOC 00040023: 0000000
*Y3 : DLOC 00040024: c000001
 Y4 : DLOC 00040025: 0000000
 Y5 : DLOC 00040026: 0000000
*Y6 : DLOC 00040027: 0000000
```

```
00000027         proceed


00000028   procedure  partition/4

00000028         switch_on_term  _1904,_1905,fail
           _1906:
00000029         try_me_else  _1907
           _1908:
0000002a         allocate
0000002b         get_variable  Y3,X2
0000002c         get_variable  Y1,X4
0000002d**       get_list  X1
0000002e         unify_variable  X1
0000002f         unify_cdr  Y4
00000030         get_list  X3
00000031         unify_value  X1
00000032         unify_cdr  Y2
00000033         call  </2,4
00000034         cut
00000035         put_value  Y4,X1
00000036         put_value  Y3,X2
00000037         put_value  Y2,X3
00000038         put_value  Y1,X4
```

```
 B  : DLOC 00040013: c000001
*H  : DLOC 00040014: 8004000
 N  : DLOC 00040015: 8004000
 AX1: DLOC 00040016: 0ffffe0
*AX2: DLOC 00040017: 0000000
 AX3: DLOC 00040018: 0000002
*AX4: DLOC 00040019: 0000100
 AX5: DLOC 0004001a: 0004000
 AX6: DLOC 0004001b: 0000001
 AX7: DLOC 0004001c: 0000004
 AX8: DLOC 0004001d: 0008000
*BCE: DLOC 0004001e: 0004000
*BCP: DLOC 0004001f: 2004001
 BP : DLOC 00040020: 0000001
*TR : DLOC 00040021: 0000000
```

Figure 3.1: DUES -- An Overview

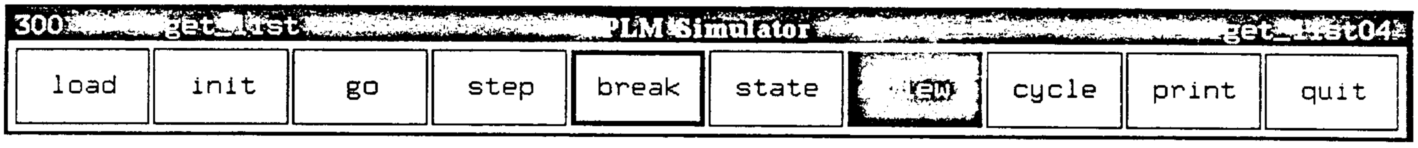


Figure 3.2: The Command Menu Bar

command button. When the mouse is moved into the region of the screen defined by one of the command buttons, the button will be highlighted. Clicking a mouse button while a command button is highlighted will select the corresponding command. Depending on which device is used, there may be more than one button on the mouse. For example, the mouse on a microVAX or Sun workstation has three buttons. In most cases, DUES does not differentiate which button is clicked. most cases. Once a command is selected, the corresponding button is redrawn in reverse video. Then, objects such as pull-down menus or dialogue boxes can be invoked to either provide further selections or prompt for more input parameters. DUES, in turn, sends appropriate simulator commands to the PLM simulator and waits for the PLM simulator to respond. When this sequence of steps complete, the button returns to normal video. DUES can then accept the next command. The commands are *step, cycle, break, go, state, print, view, load, init,* and *quit.* Refer to the DUES User's Manual in Appendix A for details of each command.

3.3. State Display Windows

The *Registers, Environment, and Choice Point* are three parts of the PLM machine states of most interest during simulation. DUES provides separate windows for each of them and keeps their contents up to date after each execution cycle.

The *Registers Display Window* shows the special purpose registers in the PLM engine:

P	program counter
CP	continuation program counter
E	last environment pointer
B	last choice point pointer
TR	pointer to top of the trail
H	pointer to top of the heap
HB	heap backtrack pointer
S	structure pointer
N	size of last environment

H2	pointer to the global heap
PDL	top of the PDL
mode	read/write unification mode
AX1..8	argument and temporary registers
T	input register for ALU
T1	input register for ALU
R	output register for ALU
cc	micro condition code
MAR	memory address register
MDR	memory data register
MISC	scratch register used in microcode

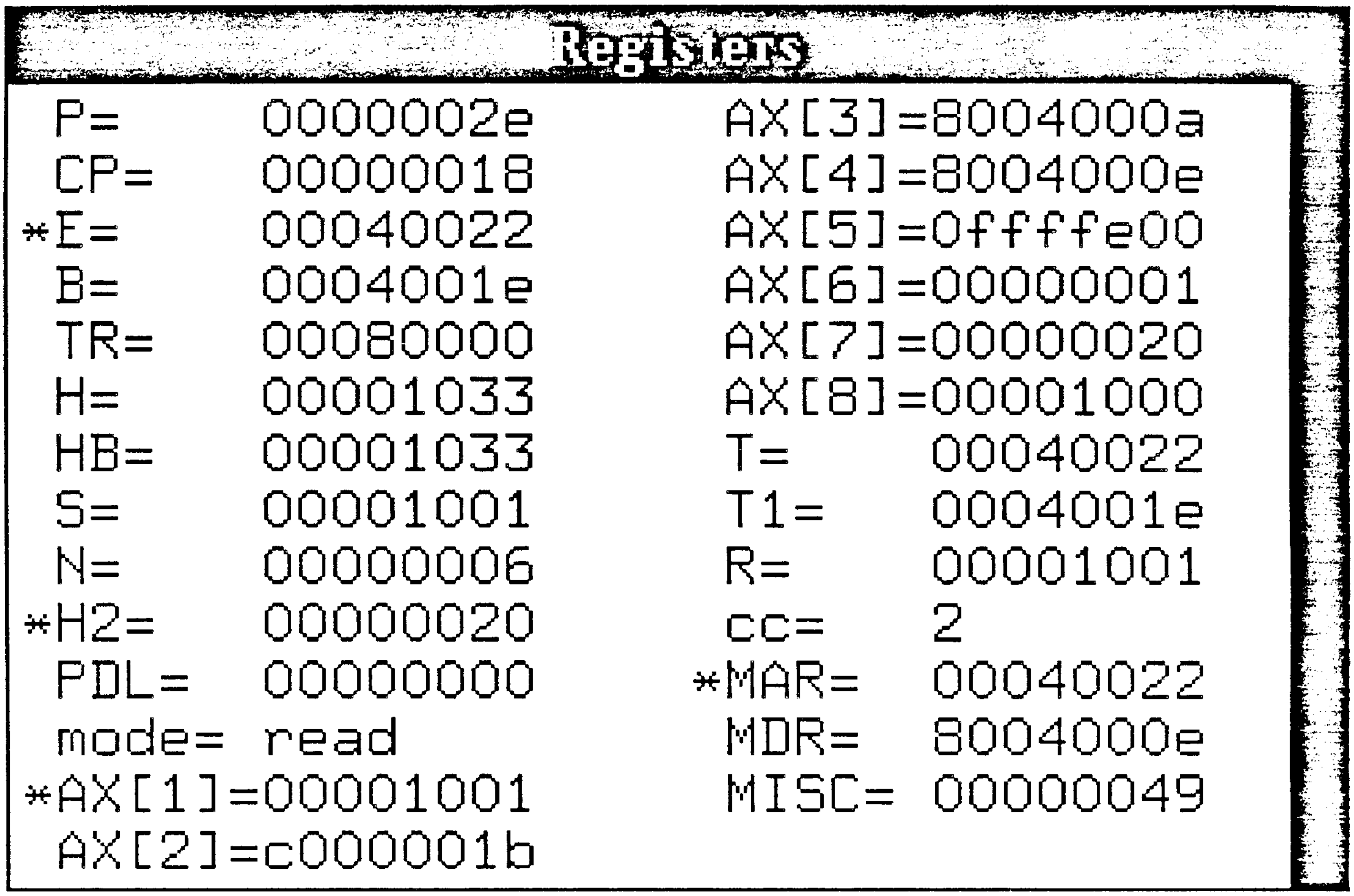
Fig. 3.3 shows the *Registers Display Window*.

The *Environment Display Window* contains the current environment as defined by the *E* and *N* registers. An environment corresponds to the saved state of a Prolog clause. It contains pertinent register values and permanent variables. The window displays the contents of these registers and variables and their addresses in the memory. The entries include:

CE	pointer to last environment
CB	pointer to last choice point
CP	continuation program pointer
CN	size of last environment
Y1..16	permanent variables

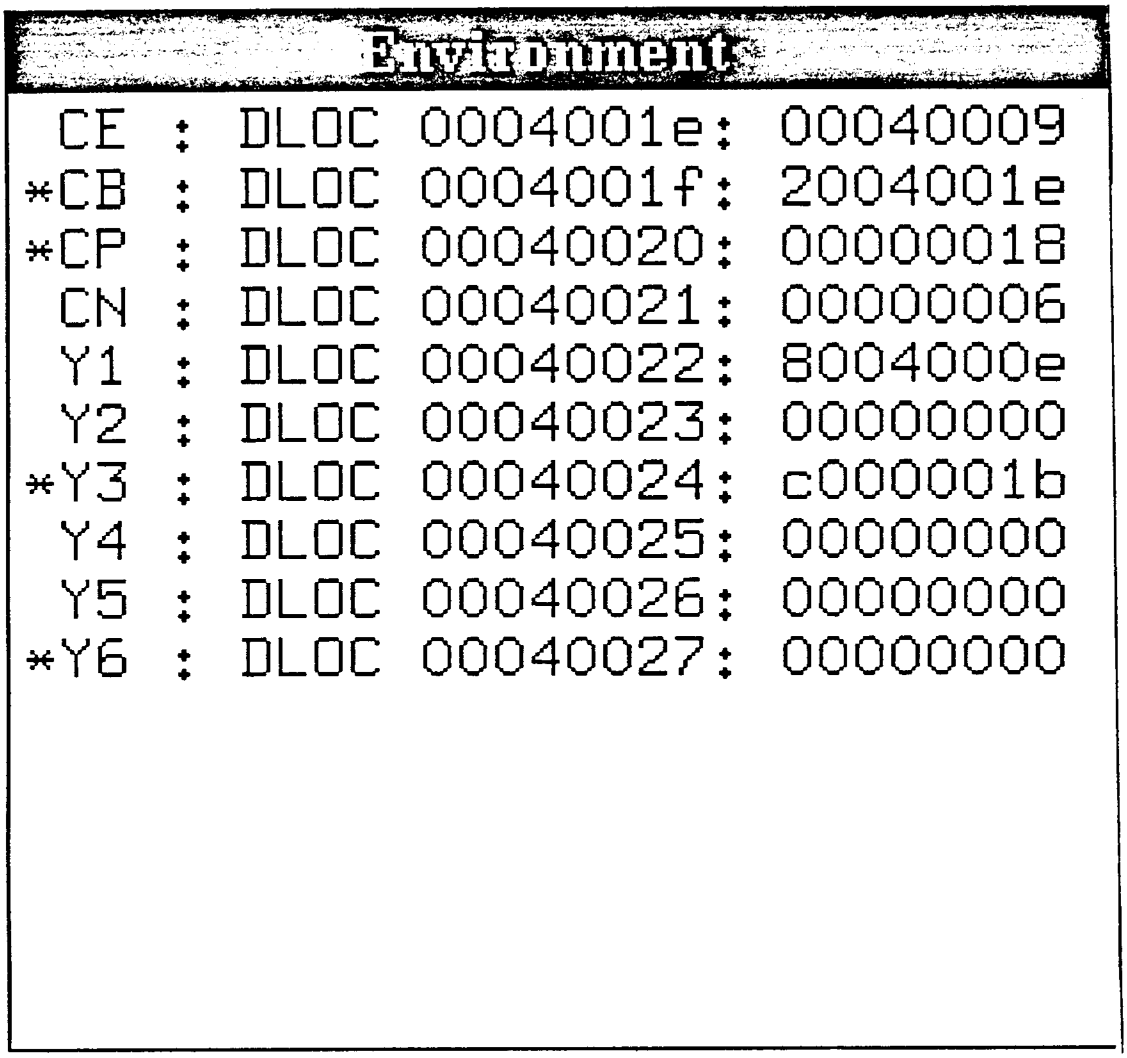
Fig. 3.4 shows the *Environment Display Window*.

The *Choice Point Display Window* displays the active choice point defined by the *B* register. A choice point contains sufficient information to restore the state of a computation if a goal fails, and to indicate the next procedure to try. The window shows the following register values that are saved in the choice point and their memory addresses.



```
                          Registers
 P=      0000002e        AX[3]=8004000a
 CP=     00000018        AX[4]=8004000e
*E=      00040022        AX[5]=0ffffe00
 B=      0004001e        AX[6]=00000001
 TR=     00080000        AX[7]=00000020
 H=      00001033        AX[8]=00001000
 HB=     00001033        T=     00040022
 S=      00001001        T1=    0004001e
 N=      00000006        R=     00001001
*H2=     00000020        cc=    2
 PDL=    00000000       *MAR=   00040022
 mode=   read            MDR=   8004000e
*AX[1]=00001001          MISC=  00000049
 AX[2]=c000001b
```

Figure 3.3: The Registers Display Window


Environment

```
 CE  :  DLOC 0004001e:  00040009
*CB  :  DLOC 0004001f:  2004001e
*CP  :  DLOC 00040020:  00000018
 CN  :  DLOC 00040021:  00000006
 Y1  :  DLOC 00040022:  8004000e
 Y2  :  DLOC 00040023:  00000000
*Y3  :  DLOC 00040024:  c000001b
 Y4  :  DLOC 00040025:  00000000
 Y5  :  DLOC 00040026:  00000000
*Y6  :  DLOC 00040027:  00000000
```

Figure 3.4: The Environment Display Window

B	pointer to last choice point
H	pointer to the top of the heap when choice point is built
N	the number of permanent variables in the current environment
AX1..8	argument registers
BCE	pointer to last environment
BCP	address of next clause should this one succeed
BP	address of next clause should this one fail
TR	pointer to top of trail when choice point is built

Fig. 3.5 shows the *Choice Point Display Window.*

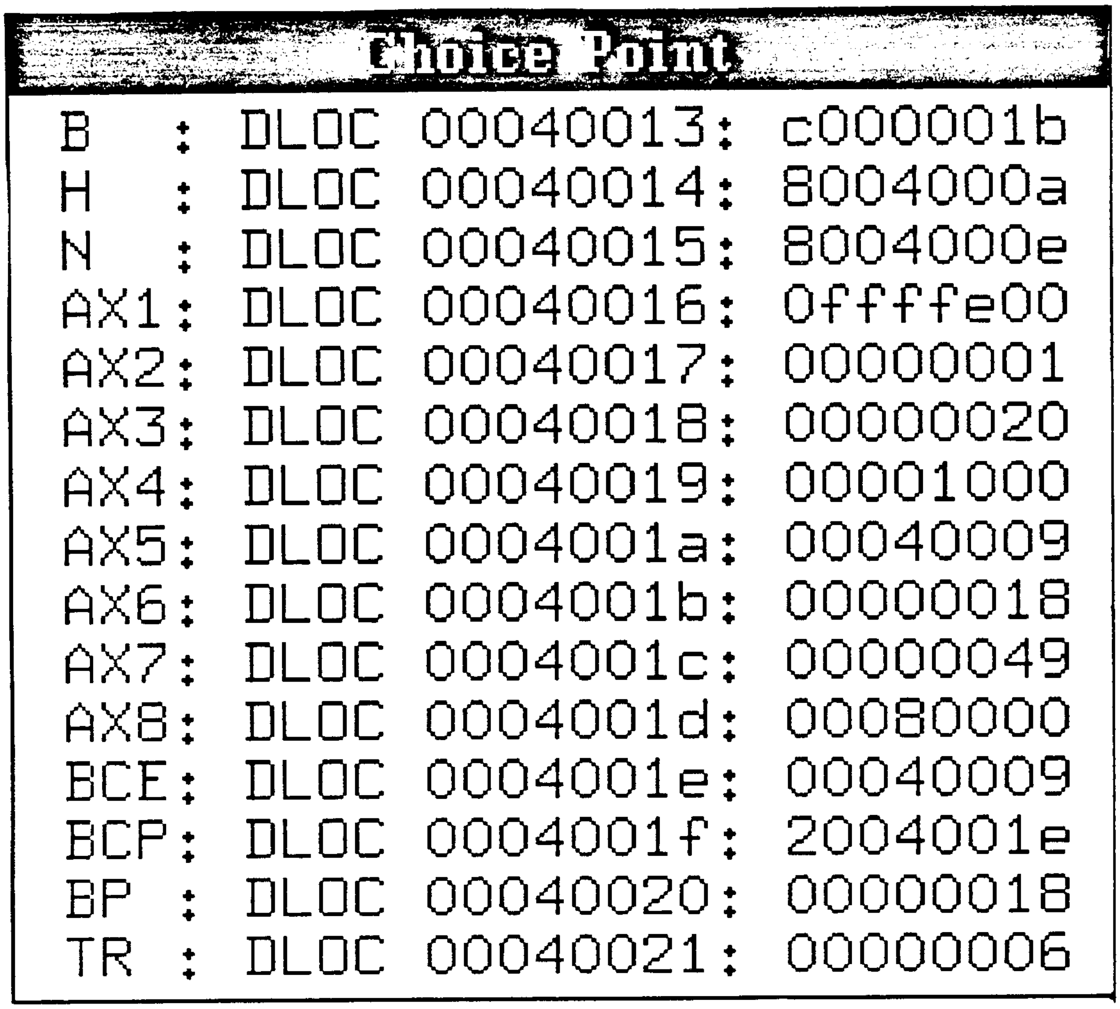
3.4. Copy Window

The role of the *Copy Window* is a scratch pad for part or all information in the *State Display Windows*. Although machine states are updated after every cycle of execution, the *Copy Window* makes previous machine states still available. It becomes most useful when one wants to compare contents of machine states between cycles.

In order to copy machine states to the *Copy Window*, they should be selected by the *state* command first. Those states that are selected will have a star sign ("*") shown on the left of the their entries on the *State Display Windows*. Copying the selected states from the *State Display Windows* to the *Copy Window* is done by clicking on one of the *State Display Windows*. The contents of the *Copy Window* will scroll upward when more states get copied into the window. Fig. 3.6 shows the *Copy Window*.

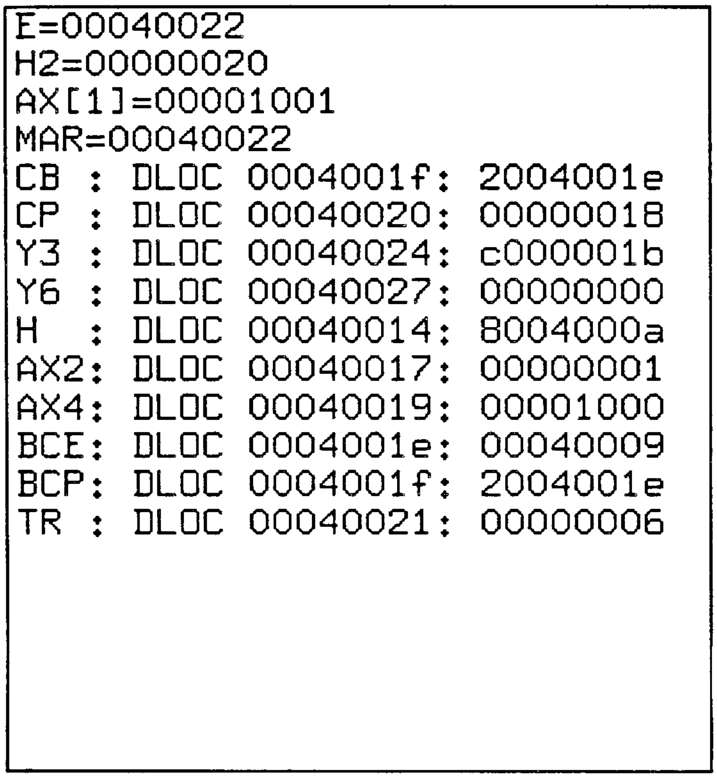
3.5. View Window

The *View Window* presents the compiled Prolog code executed by the simulator. DUES reads the file specified in the command line. The file may contain *insert* statements [1], which consist of a "!" sign in the first column followed by a file name. DUES manages file insertion properly. It also assigns a sequence number for each line of instructions, which greatly facilitates the setting of break points. The *View Window* also provides a



```
B   : DLOC 00040013: c000001b
H   : DLOC 00040014: 8004000a
N   : DLOC 00040015: 8004000e
AX1: DLOC 00040016: 0ffffe00
AX2: DLOC 00040017: 00000001
AX3: DLOC 00040018: 00000020
AX4: DLOC 00040019: 00001000
AX5: DLOC 0004001a: 00040009
AX6: DLOC 0004001b: 00000018
AX7: DLOC 0004001c: 00000049
AX8: DLOC 0004001d: 00080000
BCE: DLOC 0004001e: 00040009
BCP: DLOC 0004001f: 2004001e
BP  : DLOC 00040020: 00000018
TR  : DLOC 00040021: 00000006
```

Figure 3.5: The Choice Point Display Window



```
E=00040022
H2=00000020
AX[1]=00001001
MAR=00040022
CB : DLOC 0004001f: 2004001e
CP : DLOC 00040020: 00000018
Y3 : DLOC 00040024: c000001b
Y6 : DLOC 00040027: 00000000
H  : DLOC 00040014: 8004000a
AX2: DLOC 00040017: 00000001
AX4: DLOC 00040019: 00001000
BCE: DLOC 0004001e: 00040009
BCP: DLOC 0004001f: 2004001e
TR : DLOC 00040021: 00000006
```

Figure 3.6: The Copy Window

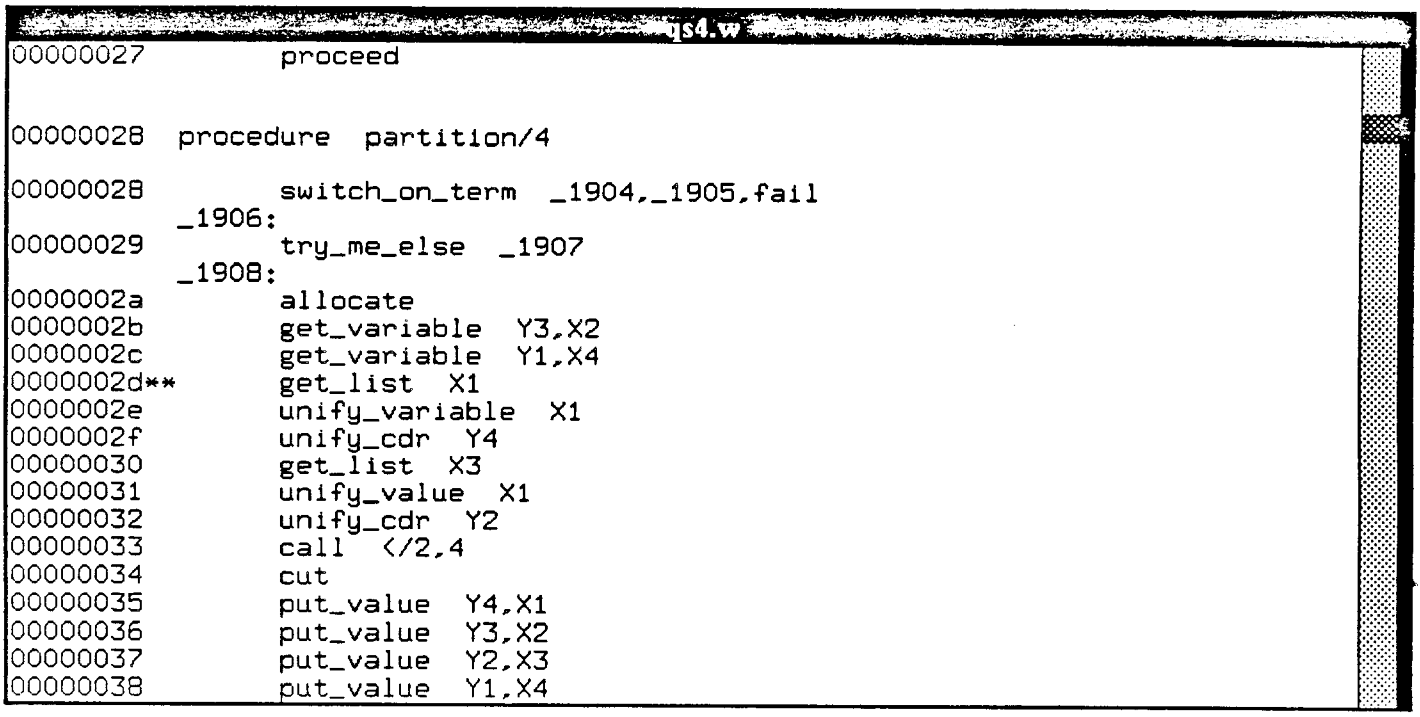
scroll bar on the right border, which allows users to be able to browse through all parts of the code.

More importantly, the *View Window* synchronizes with the execution of the PLM simulator. A pointer ("**") indicates which instruction is currently being executed as the simulator runs. If the current instruction is not in the range of the *View Window*, DUES will bring the appropriate page to the window automatically. Fig. 3.7 shows the *View Window*.

3.6. Dialogue Window

The *Dialogue Window* serves mainly as an echo area. During initialization, it displays messages such as the loading information, the number of instructions, the number of procedures, and the number of labels. During the execution, it echoes each command as one of the command buttons is selected. In addition, it shows the output from the *print-data* command.

The window was so named for historical reasons. Initially, this window also serves as a general dialogue box for several commands. Although it is still true for *load* and *init* commands, both of which are not very relevant to the current PLM simulator, most commands now use their own dialogue boxes. Fig. 3.8 shows the *Dialogue Window*.

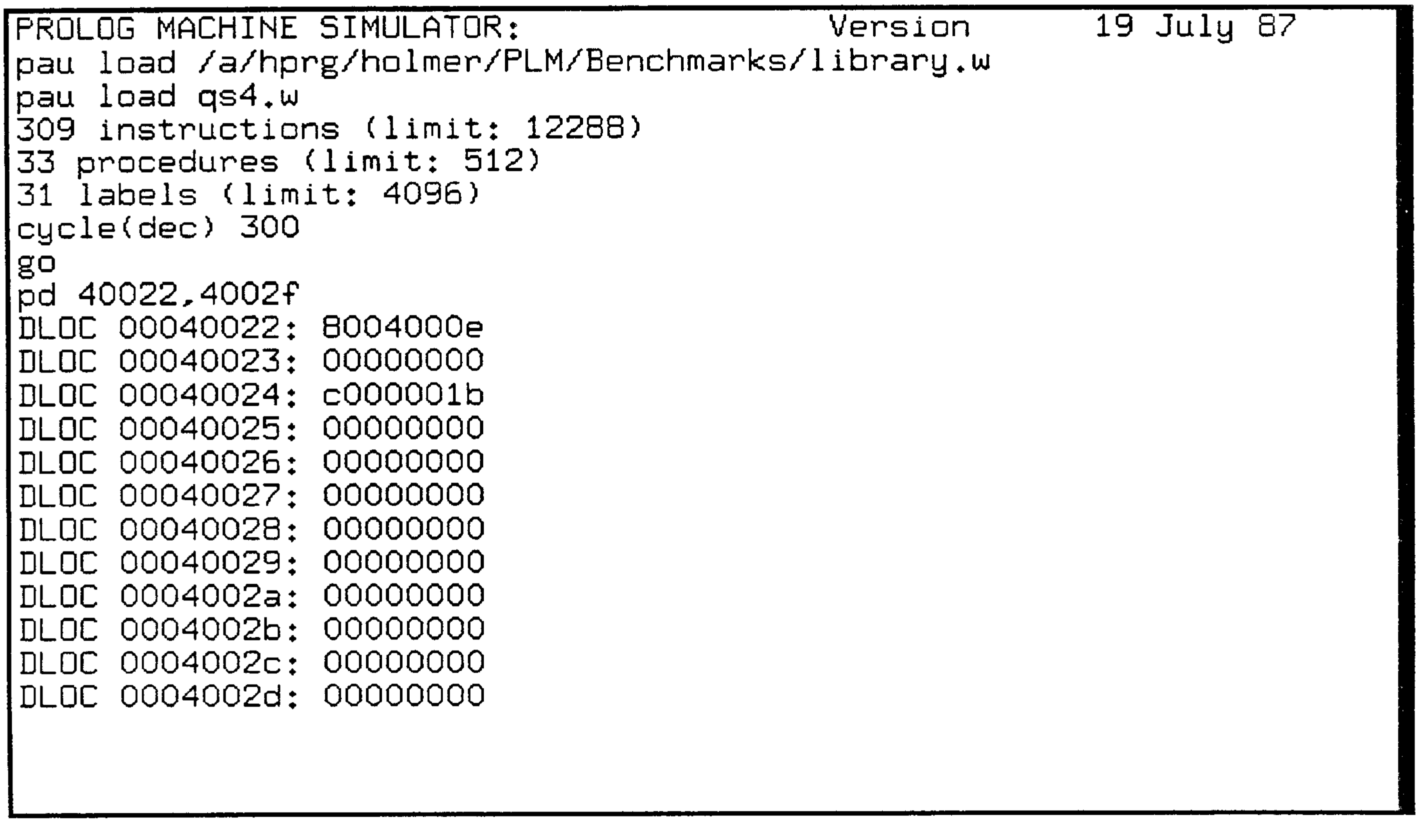


```
00000027          proceed


00000028  procedure  partition/4

00000028          switch_on_term  _1904,_1905,fail
          _1906:
00000029          try_me_else  _1907
          _1908:
0000002a          allocate
0000002b          get_variable  Y3,X2
0000002c          get_variable  Y1,X4
0000002d**        get_list  X1
0000002e          unify_variable  X1
0000002f          unify_cdr  Y4
00000030          get_list  X3
00000031          unify_value  X1
00000032          unify_cdr  Y2
00000033          call  </2,4
00000034          cut
00000035          put_value  Y4,X1
00000036          put_value  Y3,X2
00000037          put_value  Y2,X3
00000038          put_value  Y1,X4
```

Figure 3.7: The View Window



```
PROLOG MACHINE SIMULATOR:                 Version      19 July 87
pau load /a/hprg/holmer/PLM/Benchmarks/library.w
pau load qs4.w
309 instructions (limit: 12288)
33 procedures (limit: 512)
31 labels (limit: 4096)
cycle(dec) 300
go
pd 40022,4002f
DLOC 00040022: 8004000e
DLOC 00040023: 00000000
DLOC 00040024: c000001b
DLOC 00040025: 00000000
DLOC 00040026: 00000000
DLOC 00040027: 00000000
DLOC 00040028: 00000000
DLOC 00040029: 00000000
DLOC 0004002a: 00000000
DLOC 0004002b: 00000000
DLOC 0004002c: 00000000
DLOC 0004002d: 00000000
```

Figure 3.8: The Dialogue Window

Chapter 4

Design Issues

4.1. Introduction

In this chapter, three design issues are discussed. Section 2 presents one design concept, *the separation of policy and mechanism*. Section 3 explains how DUES and the PLM communicate with each other. Section 4 describes the standard output buffering and the relevant modification required for the PLM simulator.

4.2. Separation of Policy and Mechanism

One important concept behind the design of DUES is *the separation of policy and mechanism*. Policies should be mandated by the simulators while mechanisms are implemented by DUES. DUES never provides facilities to be implemented by the simulator. For example, there is no checkpoint saved with DUES because each simulator has its own different set of machine states; thus, it should be implemented by the simulator rather than DUES. The advantage of this concept is that DUES does not have any fixed data structure which simulators can assume they will have to interface to. In other words, DUES is intentionally kept flexible in order to be a front-end for a number of different simulators.

4.3. Interprocess Communication

When DUES is running, it is communicating with two other processes, i.e. the PLM simulator and the X server, via UNIX stream sockets. Fig. 4.1 shows the run time configuration of these processes. DUES and the PLM simulator must run on the same machine while the X server is usually running on another machine within the network. Input from the keyboard or the mouse is received by the X server, and is distributed asynchronously in the form of *events* to DUES. On the other hand, the communication between DUES and the PLM simulator is interlocked. When one of the command buttons is pressed, the appropriate simulator command is sent to the PLM simulator. DUES waits until the PLM simulator sends a message back before it begins processing the next event.

The pair of stream sockets between DUES and the PLM simulator are created by the *socketpair()* system call before DUES forks a new process to execute the PLM simulator. In DUES, two file descriptors, *fd[0]* and *fd[1]* are returned by the *socketpair()* system call. They serve as two end points of the communication link. DUES uses *fd[0]* while the PLM simulator uses *fd[1]*. Whatever is written to *fd[0]* will be received by *fd[1]*, and vice versa. Since all of *stdin, stdout, and stderr* of the PLM simulator are redirected to *fd[1]* before the *execlp()* call, the PLM simulator does not need to be aware of the existence of DUES; it receives commands as if the user were typing at a terminal. Similarly, the *stdin* and *stdout* are redirected to *fd[0]*, but not the *stderr*. The redirection is achieved by the *dup2()* system call.

The connection between DUES and the X server is established by the *XOpenDisplay()* call. The call takes a string of the display name from the command line as an argument. If the display name string is NULL, it uses the environment variable DISPLAY to determine which display to use. The display name string or the DISPLAY environment variable should be in the format "hostname:number", where "hostname" is the name of the machine, and "number" is the number of the display of that machine.

4.4. Standard Output Buffering

In general, the PLM simulator does not need to be modified to use DUES as a front-end. Some change is required for the simulator, however, if it uses the standard I/O routines such as *printf()*. This works fine when the *stdin*, *stdout*, and *stderr* are ttys, where the output buffer is flushed when it is full or when it detects a newline character ('\n').

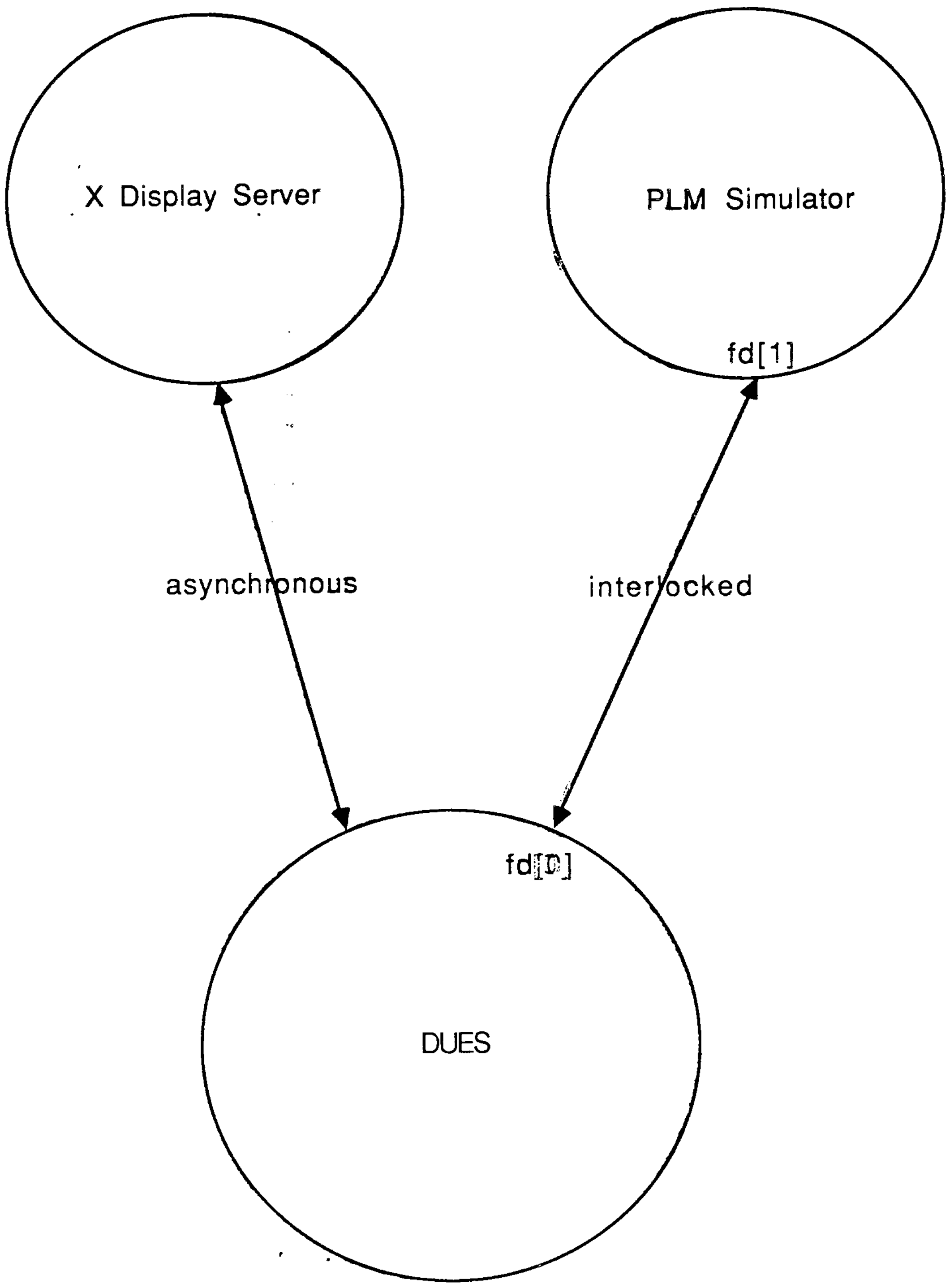


Figure 4.1: DUES interprocess communication mechanism

However, it does not flush the buffer at a newline character if the file descriptors are referencing sockets. To get around that, it is necessary to call *fflush()* in the PLM simulator to explicitly flush the buffer of *stdout* after each time a prompt is printed. For DUES, what is needed is simply set line buffering for the *stdout* by a *setlinebuf()* call, which notifies the standard I/O routines to flush the *stdout* buffer at each newline character.

Chapter 5

Interfacing DUES to Other Simulators

5.1. Introduction

Extensibility has been a major design goal; therefore, DUES is actually implemented as a framework for any simulator/debugger. Although a lot of optimization has been made specifically for the PLM simulator, DUES can be used for other simulators with minor modifications. In the following sections, we will describe how to interface DUES to other simulators with minimum effort even for people who do not know the X Window system extensively.

Interfacing DUES to other simulators involves modifying three parts of DUES. All of these parts are procedure modules in DUES. Therefore, following the systematic procedures described below will allow possible changes or upgrading from the simulator side to be made easily. First, the command menu bar needs to be extended to include new command buttons on it. Second, for each new command there should be a set of procedures to implement pull-down menus, dialogue boxes, etc.. Last, the part of DUES which handles the communication with the simulator needs rewriting.

5.2. Extending Commands

Currently, all commands are listed on the menu bar. The structure of each command is defined by a type *Command* in the file *gasix.h*.

```
typedef struct{
  char *name;
  int (*func)();
  int button; /* ON or OFF */
} Command;
```

Name is a pointer to the name of the command. *Func* points to a C procedure, which specifies the actions when the command button is clicked on. How to write specific C procedures to create pull-down menus, dialogue boxes, etc. is discussed in detail in the next section. The field *button* is a flag. If the command is being executed, it is *ON*; otherwise, it is *OFF*. Initially, flags of all commands are off. The array *cmd[MaxCmd]*, declared and initialized in the file *global.c*, holds all command structures.

```
extern int f_load();
extern int f_init();
extern int f_go();
extern int f_step();
extern int f_break();
extern int f_state();
extern int f_view();
extern int f_cycle();
extern int f_print();
extern int f_quit();

Command cmd[MaxCmd] = {
  "load", f_load, OFF,
  "init", f_init, OFF,
  "go", f_go, OFF,
  "step", f_step, OFF,
  "break", f_break, OFF,
  "select", f_state, OFF,
  "view", f_view, OFF,
  "cycle", f_cycle, OFF,
  "print", f_print, OFF,
  "quit", f_quit, OFF
};
```

Each command also has an ID, which is a constant defined in the file *gasix.h*. Note that the order of the command IDs are important and should correspond to that of the array *cmd[]*. The constant *MaxCmd* is the total number of commands defined. The command IDs for the PLM simulator are as follows:

```
#define LOAD          0
#define INIT          1
#define GO            2
#define STEP          3
#define BREAK         4
#define STATE         5
#define VIEW          6
#define CYCLE         7
#define PRINT         8
#define QUIT          9
#define MaxCmd        10 /* max number of commands */
```

In summary, to create new commands and to create their buttons on the menu, both the command IDs in the file *gasix.h* and the initialization of the array *cmd[]* in the file *global.c* must be updated appropriately.

5.3. Creating Objects

Although the DUES project is not intended to provide library routines like toolkits do to create objects such as pull-down menus, dialogue boxes, etc., implementing C procedures to create new objects for new commands is made easy by modifying existing DUES procedures, which serve as templates, to new ones. In general, there are three most often used objects: pull-down menus, dialogue boxes which prompt for one argument, and dialogue boxes prompting for two arguments. They will be described in the following sections.

5.3.1. Pull-down Menu

One command that invokes a pull-down menu is the *select* command. The pull-down menu provides options to select machine states among *Registers*, *Environment*, or *Choice Point*. As the mouse is moved over an option, the option will be highlighted. The C procedures implementing these features are in the file *state.c*.

The options on the pull-down menu are declared in an array *stMenuItem[]*. Each element in the array is of the type *struct StMenuItemStr*.

```
struct StMenuItemStr{
  char *name;
  int (*f)();
};
```

Name is a pointer to a character string, which is the name for the option on the pull-down menu. *F* points to the procedure that will be executed when the option is selected. The array *stMenuItem[]* in DUES is initialized as follows:

```
int selectRegisters();
int selectEnvironments();
int selectChoicePoints();

static struct StMenuItemStr stMenuItem[] = {
  "Registers", selectRegisters,
  "Environment", selectEnvironments,
  "ChoicePoint",selectChoicePoints
};
```

To create a new pull-down menu, use the file *state.c* as a template and fill in the array *stMenuItem* with proper values.

5.3.2. Dialogue Box for One Argument

The *cycle* command is an example which creates a dialogue box prompting for one argument. The dialogue box pops up when the command is selected. A prompt string is printed in the box. At that point, DUES, waits until the input is entered. Internally, a buffer is allocated for the input string. All control characters are processed at the input buffer. For example, *Control-H* or *delete* key is the *erase* character; *control-U* or *control-X* is the *kill-line* character.

The procedures associated with the *cycle* command are in the file *cycle.c*. The prompt string is defined as a constant *ThePrompt*. In DUES, the definition is

```
#define ThePrompt "cycle number(decimal)"
```

The input in the buffer is converted to an integer and is assigned to an integer variable *arg*.

```
static int arg = 0;
static char inputbuf[7+1] = "";
```

To create a similar dialogue box, use the file *cycle.c* as a template and define the constant *ThePrompt* as the appropriate prompt string. The input entered can be found in the array *inputbuf[]*; its integer value is in *arg*.

5.3.3. Dialogue Box for Two Arguments

The *print* command activates a dialogue box prompting for two arguments, namely, the starting and ending addresses in the Data Space. The dialogue box contains two buttons (*OK* and *Cancel*), a title, and two input areas each labeled with a prompt string. At any time, only one of the input areas is active to accept the key input, and can be identified by where the cursor is located. The control characters are also handled similarly as above.

The file *pd.c* contains all procedures for the *print* command. The title and prompt strings are defined as three constants: *Title*, *Prompt1*, and *Prompt2*.

```
#define Title "Print Data"
#define Prompt1 "From:"
#define Prompt2 "To:"
```

Each input area is represented as a structure of the type *InputBox*.

```
typedef struct _InputBox{
  int x_start, y_start;
  int x_cursor, y_cursor;
  char buf[9];
  int tail;
} InputBox;
```

X_start and *y_start* are the coordinates of the upper left corner of the input area. *X_cursor*, and *y_cursor* specify where the cursor is when the input area becomes active. *Buf[]* is the buffer holding input characters. The buffer works as a queue, and, *tail* serves as an index to the end of the queue. For the *print* command, the two input areas are declared as two variables, *from* and *to*.

```
static InputBox from;
static InputBox to;
```

To create a dialogue box for two input arguments, use the file *pd.c* as a template. Then, define *Title, Prompt1*, and *Prompt2* as the title string and the prompt strings respectively. The two arguments entered by the user are stored in two character strings, *from.buf[]* and *to.buf[]*.

5.4. Communicating with the Simulator

Different simulators will have different sets of commands and different reply messages for each command. The communication between DUES and the simulator contains two parts. DUES sends the commands to the simulator; the simulator responds with messages back to DUES. The first part is easier to handle because all that is needed is to assemble the appropriate simulator command and possibly the arguments to a string, and to write the string on the socket *fd[0]*. The code for this part is scattered in each individual file corresponding to each command. Below is an example of the *print* command.

```
char cmdbuf[32];

strcpy(cmdbuf,"pd ");
strcat(cmdbuf,from.buf);
strcat(cmdbuf,",");
strcat(cmdbuf,to.buf);
strcat(cmdbuf,"0);
write(1,cmdbuf,strlen(cmdbuf));
```

Cmdbuf is an array of character that keeps the command string to be sent. *Strcat()* calls append the first argument (*from.buf*), a comma, the second argument (*to.buf*), and a newline character to the command "pd." The *write()* call writes the string in *cmdbuf[]* to file descriptor 1, which references the socket *fd[0]* (see section 4.3).

The second part is related to interpreting messages returned from the simulator and distributing information to appropriate windows. This part is contained in the file *reply.c*. The interpretation of the reply message from each simulator command is done in two steps. First, all reply messages are read in at once. Normally, the reply message ends with a simulator prompt "dbg> ." Because the PLM simulator flushes the *stdout* buffer, it is guaranteed that all messages will be received in one chunk. The second step involves interpreting the reply messages returned from various simulator commands, and distributing data to appropriate windows. A *switch* statement in the file *reply.c* branches to the relative part of program for each command according to the value in an integer variable *code*. The value of *code* can be a constant defined in the file *reply.h*. Each constant corresponds to one simulator command.

```
#define RE_LOAD       0
#define RE_STEP       1
#define RE_ENV        2
#define RE_CP         3
#define RE_CYCLE      4
#define RE_GO         5
#define RE_PD         6
#define RE_BP         7
#define RE_WS         8
```

In summary, the communication between DUES and the simulator consists of two parts, sending commands and receiving messages. For the first part, DUES needs to send the right simulator command

with arguments if appropriate over the socket to the simulator. For the second part, both the files *reply.c* and *reply.h* should be updated to interpret the messages returned by the simulator, and, possibly, to distribute information to appropriate windows.

Chapter 6

Conclusions

6.1. Report Summary

This report describes DUES, an implementation of a front-end user interface for the PLM simulator. The initial implementation of DUES has met its design goals to provide a user friendly environment for people to run the PLM simulator in the following ways.

(1) A *Command Menu Bar* provides a list of commands; Further selection is made possible by pull-down menus or dialogue boxes. Users no longer need to memorize or type different commands.

(2) Three important parts of the PLM machine states, *Registers*, *Environment*, and *Choice Point* are displayed in separate windows and are updated after each execution cycle. No extra commands are needed to examine them.

(3) Part or all of the machine states can be selected and later be copied to a scratch pad area for comparisons of data between cycles.

(4) The compiled Prolog code is listed in a window along with sequence numbers. A vertical scroll bar is provided to allow users to browse through the code. In addition, the current instruction being executed is indicated.

(5) Mechanisms are implemented, wherever appropriate, to give feedback to users through highlighting, reverse-videoing, or beeping.

(6) Extensibility is achieved as DUES provides procedure templates for implementing

procedures of similar functionalities. DUES is actually implemented as a framework for interfacing to other simulators.

6.2. Future Enhancements

Although, nearly all the desired features have been implemented, a few of them still leave room for improvements. Listed below are suggestions for future work.

(1) Currently, DUES uses only two colors for either a monochrome or a color display. With a little more work, it should be able to use more colors on a color display.

(2) A log file that records all commands entered to DUES would be useful especially for debugging purpose.

(3) The output of the *print-data* command is now directed to the *Dialogue Window*. This information will be lost when the *Dialogue Window* scrolls in order to display other things such as command echoes. A separate window for its output is more appropriate.

(4) The interface to the command to set break points with instruction sequence numbers can be improved. The current interface requires users to enter the number through a dialogue box. A more ideal interface would be simply clicking on the instruction in the *View Window*.

(5) A feature to print Prolog structures would be useful, and should be implemented in a future release.

References

[1] T.P. Dorby, *PLM Simulator Reference Manual,* Computer Science Division, U. C. Berkeley, Sep. 1984.

[2] T.P. Dorby, A.M. Despain, and Y.N. Patt, *Performance Studies of a Prolog Machine Architecture,* Proceedings of the 12th Intl. Symposium on Computer Architecture, 1985, pp.180-190.

[3] T.P. Dorby, *A High Performance Architecture for Prolog,* Ph.D. Dissertation, University of California, Berkeley, California, 1987.

[4] B. Fagin, and T.P. Dorby, *The Berkeley PLM Instruction Set: An Instruction Set for Prolog,* UCB Research Report 86/257, CS Division, University of California, Berkeley, California, 1985.

[5] D.H.D. Warren, *An Abstract Prolog Instruction Set,* Technical Report, SRI International, Menlo Park, California, 1983.

[6] B. Shneiderman, *Design the User Interface, Strategy for Effective Human-Computer Interface,* Addison-Wesley Publishing Company, Reading, MA, 1987.

[7] R.W. Scheifler, J. Gettys, *The X Window System,* Transactions on Graphics #63, Special Issue on User Interface Software, Association for Computing Machinery, 1986.

[8] J. Gettys, R. Newman, and T. D. Fera, *Xlib - C Language X Interface (Protocol Version 10),* Massachusetts Institute of Technology, November 1986.

Appendix 1

DUES User's Manual

A.1. Introduction

Access to DUES is provided by a shell script on ji.berkeley.edu as:

```
~awei/rs/plm/dues [-rv] [hostname:display] filename.w
```

where *filename.w* is the name of the file containing the compiled Prolog code and the loader instructions. The optional argument *hostname:display* specifies on which machine the display hardware is used. *Display*, the display number, is usually 0 for single display machines. Another argument *-rv*, if given, reverse the definition of foreground and background color. DUES sets up the default foreground color as white and the default background color as black.

Shortly after DUES is invoked, six windows are displayed on the screen. These include a *Command Menu Bar*, a *Dialogue Window*, a *Copy Window* and three *Display Windows* that show the contents of the *Registers*, *Environment*, and *Choice Point* respectively. A seventh window that displays the compiled Prolog code can be opened by selecting the *view* button on the command menu bar. These windows are child windows of the root window. Therefore, they may be manipulated (raised, moved, resized, iconified, etc.) by window managers such as *uwm*. The structures of these windows and the facilities they provide will be discussed in the following sections.

DUES reads defaults from a customization file in the user's home directory, *.Xde-*

faults. The format of each line in the file is "DUES.<keyword>:<string>." Refer to the UNIX manual *X(1)* for more details. Keywords recognized by DUES are listed below.

SimulatorName

specifies the pathname of the simulator DUES is going to invoke.

MenuBarGeometry

specifies the geometry of the *Command Menu Bar.* The string takes the form of "=<width>x<height>{+-}<xoff>{+-}<yoff>," which is a standard in X to indicate the window size and placement acceptable by *XParseGeometry().* Here, the *width* and *height* are not significant because the size of the menu bar is fixed. *Xoff* and *yoff* are in pixels, each specifying the distance from a corner of the screen to the corresponding corner of the window. Their signs are meaningful, too. They determine which corner should be used as the origin of the coordinate system:

+xoff+yoff	upper left corner
-xoff+yoff	upper right corner
+xoff-yoff	lower left corner
-xoff-yoff	lower right corner

DialogWindowGeometry

specifies the geometry of the *Dialogue Window.* The string takes the same form as **MenuBarGeometry**.

CopyWindowGeometry

specifies the geometry of the *Copy Window.* The string takes the same form as **MenuBarGeometry**.

RegDpyGeometry

specifies the geometry of the *Register Display Window.* The string takes the same form as **MenuBarGeometry**

EnvDpyGeometry

specifies the geometry of the *Environment Display Window*. The string takes the same form as **MenuBarGeometry**.

CpDpyGeometry

specifies the geometry of the *Choice Point Display Window*. The string takes the same form as **MenuBarGeometry**.

So far, we have seen two ways to specify options: one on the command line, the other in the *.Xdefaults* file. If neither of the above is specified, DUES will use predefined constants for the options. DUES always looks for options on the command line first, then, the *.Xdefaults* file, and at last uses its predefined constants. This way, users can have their own customized environments. At the same time, they could also be unaware of any options and still be able to run DUES in a standard way.

DUES has ten commands available, each of which is represented by a command button. When the mouse is moved into any of the command buttons, the button will be highlighted. To select one of the commands, clicking any mouse button will do. Depending on which device is used, there may be more than one button on the mouse. For example, the mouse on a micro VAX or Sun workstation has three buttons. DUES does not differentiate which button is clicked for most cases. Once a command is selected, the corresponding button is redrawn in reverse video. Then, some objects such as pull-down menus or dialogue boxes will be invoked to either provide further selections or prompt for more input parameters. When the command is completed, the button returns to normal video. At that point, DUES is ready to accept the next command. The commands are *step, cycle, break, go, state, print, view, load, init,* and *quit*. They will be described in the following sections.

A.2. The Step Command

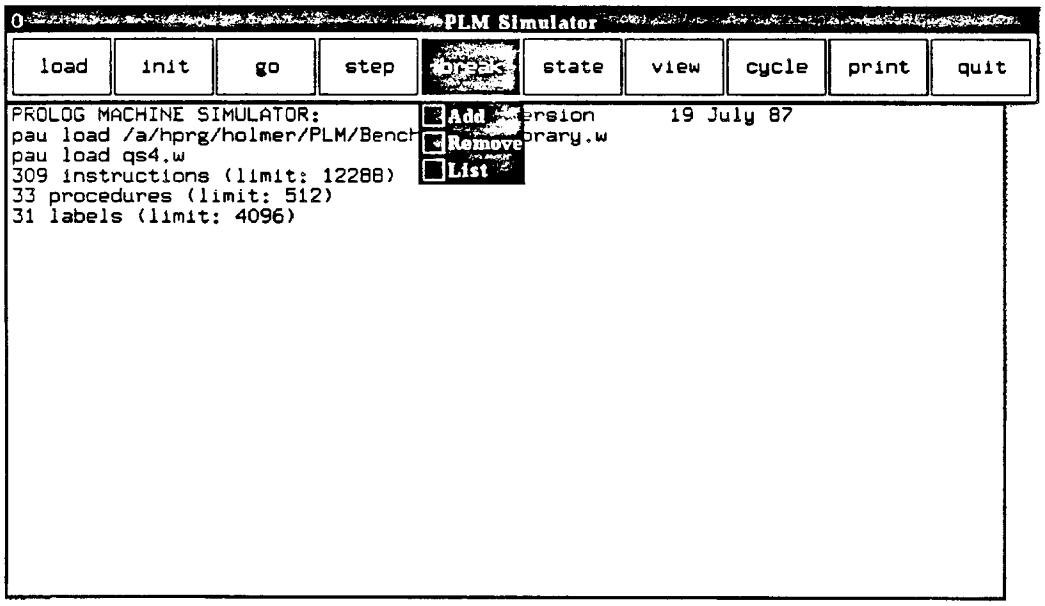
The *step* command notifies the PLM simulator to execute one microinstruction. DUES actually sends three PLM simulator commands: *single-step* (s), *print-environment* (pe), and *print-choice-point* (pb). Upon completing a single step, the PLM simulator reports to DUES the cycle number, the instruction being executed and its location, the microinstruction being executed, the contents of the PLM registers, the current environment as defined by the *B* and *N* registers, and the active choice point defined by the *B* register. DUES uses all the above information to update the title bar, the *Registers, Environment,* and *Choice Point Display Windows*, and the *View Window.*

A.3. The Go Command

The *go* command notifies the PLM simulator to execute until the first breakpoint is encountered. Breakpoints can be entered through two commands, *break* and *cycle.* DUES actually sends three PLM simulator commands: *go* (g), *print-environment* (pe), and *print-choice-point* (pb). When the PLM simulator stops at a breakpoint, it reports to DUES the same information as it does with the *step* command. DUES uses that information to update the title bar, the *Registers, Environment,* and *Choice Point Display Windows*, and the *View Window.*

A.4. The Break Command

The *break* command allows setting breakpoints in the form of PLM instruction sequence numbers. The instruction sequence numbers can be found in the *View Window*, where both the instruction sequence number and the corresponding PLM instruction are displayed. The PLM simulator will check if a breakpoint has been reached and will stop just before that instruction. The *break* command contains two levels of menus. When the *break* button is selected, a top level sub-menu is pulled down, where more options are available. At this point, the user can either *add, remove,* or *list* breakpoints by selecting the corresponding button on the left of the item. If no further selection is intended, the user can leave the *break* command by simply moving the mouse out of the pull-down menu. Fig. A.1 shows the the pull-down menu for the break command.



```
PROLOG MACHINE SIMULATOR:          ersion      19 July 87
pau load /a/hprg/holmer/PLM/Bench  brary.w
pau load qs4.w
309 instructions (limit: 12288)
33 procedures (limit: 512)
31 labels (limit: 4096)
```

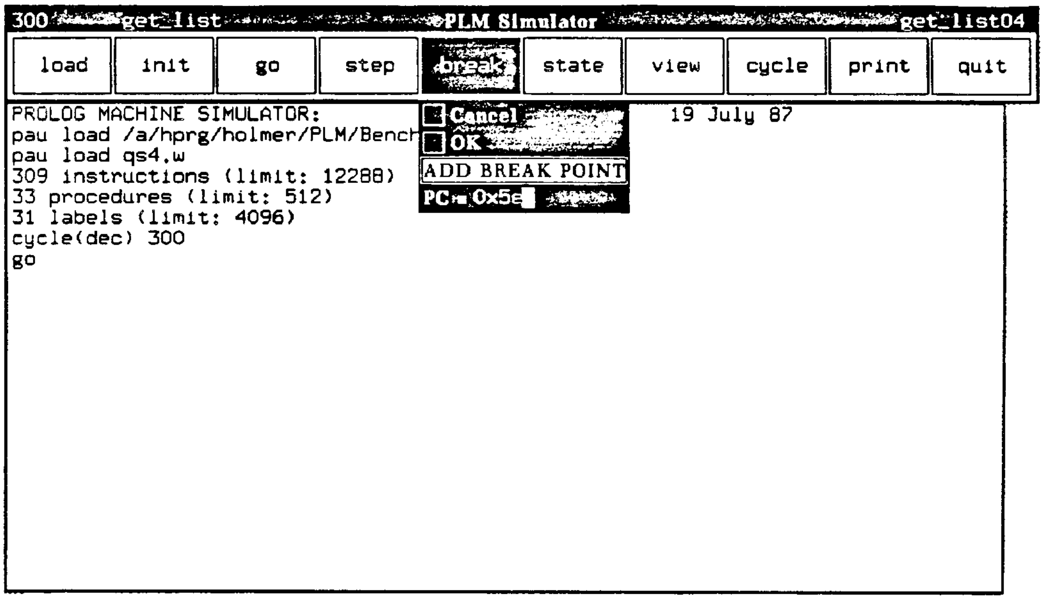
Figure A.1: The Pull-down Menu for The Break Command

A.4.1. Add Breakpoint

Selecting the *Add* button at the top level menu will invoke a second level dialogue box, which will prompt users to enter breakpoints. Fig. A.2 shows the dialogue box for adding a breakpoint. The add-breakpoint dialogue box has two command buttons (*OK* and *Cancel*), a title, and an input area. Breakpoints are entered at the input area, which can easily be identified as a cursor is shown next to a prompt string "PC = ." Users can type in the instruction sequence number in either hexadecimal or decimal format. Hexadecimal numbers should be prefixed by "0x;" numbers start with any decimal digit will be treated as decimal numbers.

A breakpoint is read in after a hexadecimal or a decimal number is entered. Although only one breakpoint is allowed in the current PLM simulator, DUES uses a linked list to store breakpoints as they are added and removed. This provides a mechanism for maintaining multiple breakpoints by future simulators. DUES first checks if the input is in the right format, then checks if it is repeated or not. Once this is done, DUES confirms by flashing the input in bold-face font. After the input is accepted, the input area is cleared, and DUES is repositioned for new input. Finally, when all breakpoints are entered, a clicking on the *OK* button will bring DUES to the first level pull-down menu. At the same time, the simulator command *set-breakpoint* (b) is sent to the PLM simulator for each newly added breakpoint, and no reply is expected.

During the course of input, several control keys are significant. Following the convention, *control-H* or *delete* key is the *erase* character; *control-U* or *control-X* is the *kill-line* character. Clicking on the *Cancel* button during the input, has the same effect as the *kill-line* character. If the *Cancel* button is clicked on before any new breakpoint is entered, DUES will abort the dialogue box and return to the first level pull-down menu. In addition, all breakpoints entered in the same add-breakpoint session will be removed. Another feature worth mentioning is that since instruction sequence numbers are represented as hexadecimal numbers in the *View Window*, DUES always furnishes the input area with the



```
PROLOG MACHINE SIMULATOR:
pau load /a/hprg/holmer/PLM/Bench
pau load qs4.w
309 instructions (limit: 12288)
33 procedures (limit: 512)
31 labels (limit: 4096)
cycle(dec) 300
go
```

Figure A.2: The Dialogue Box for Adding a Breakpoint

prefix "0x," which is erasable if it is not desired.

A.4.2. Remove Breakpoint

Selecting the *Remove* button at the top level menu will invoke another second level menu, which provides a list of the breakpoints which have been previously added before. Fig. A.3 shows the menu for removing breakpoints. To remove, first select the breakpoints to be removed by clicking on their buttons. Once all breakpoints to be removed are selected, click on the *OK* button. DUES will remove the selected breakpoints from the list. Since the current PLM simulator does not have the capability to remove any breakpoint, no command is sent by DUES. The user will be notified by a message on the screen that this feature is not available with the PLM simulator yet. Any time during the *Remove* session, if the user clicks on the *Cancel* button, all previously selected breakpoints will be ignored. Clicking on both *OK* and *Cancel* button will cause DUES to return to the first level pull-down menu.

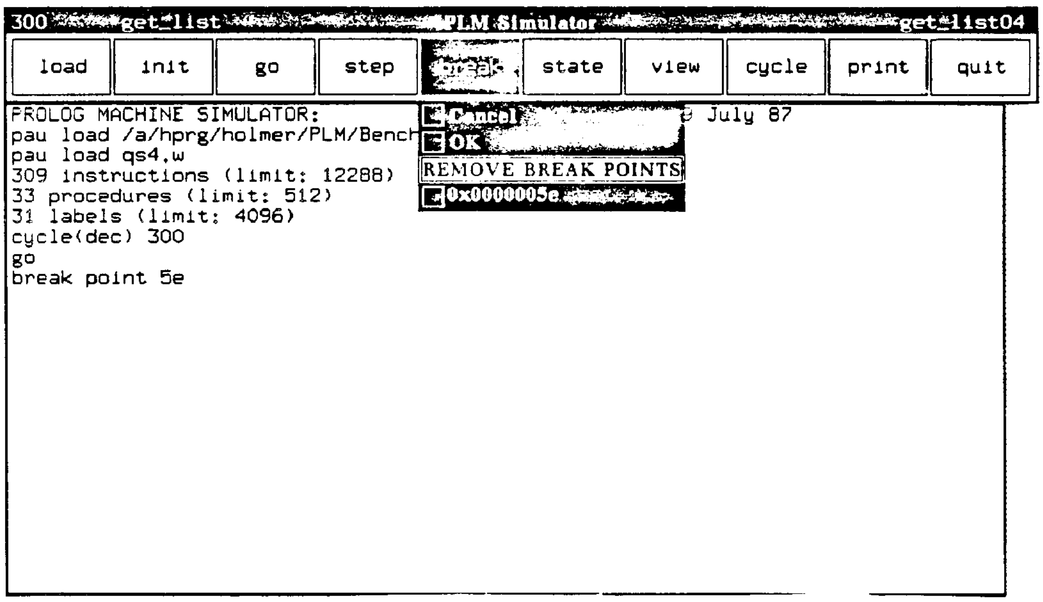
A.4.3. List Breakpoint

Selecting the *List* button at the top level will invoke a third second level menu, which lists all current breakpoints. Fig A.4 shows the menu for listing breakpoints. Clicking on the *OK* button will return DUES to the first level menu.

A.5. The Cycle Command

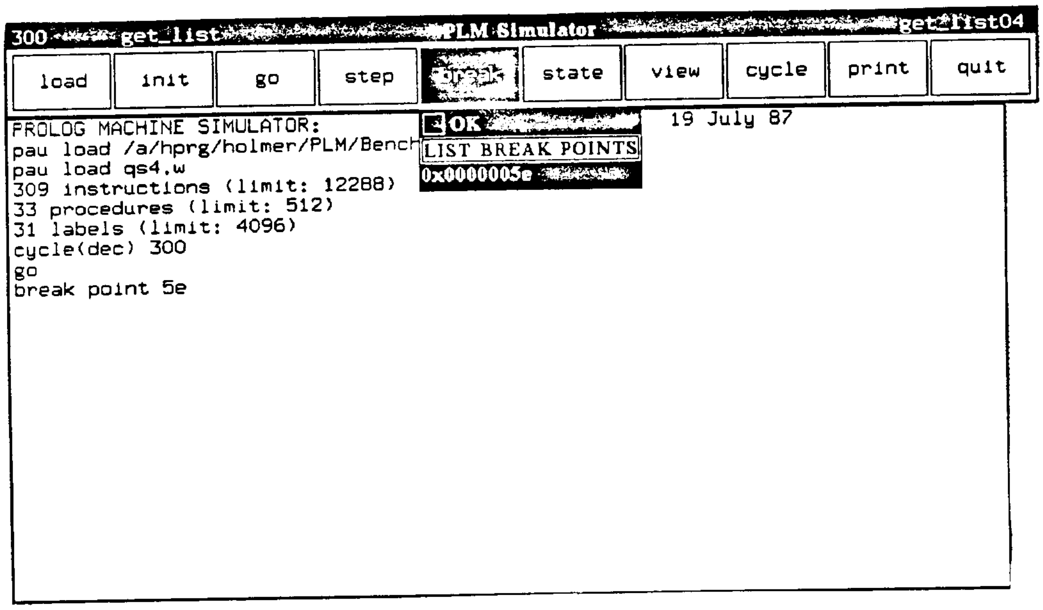
The *cycle* command allows users to set breakpoints in the form of a cycle count. The PLM simulator keeps track of the count of machine cycles from the start of simulation. Each microinstruction takes one cycle to execute. The number entered through the *cycle* command will be compared with the cycle count at the end of each machine cycle. The PLM simulator will stop when there is a match.

Clicking on the *cycle* command button will invoke a dialogue box, which prompts for a decimal number. Fig. A.5 shows the dialogue box for the cycle command. DUES accepts



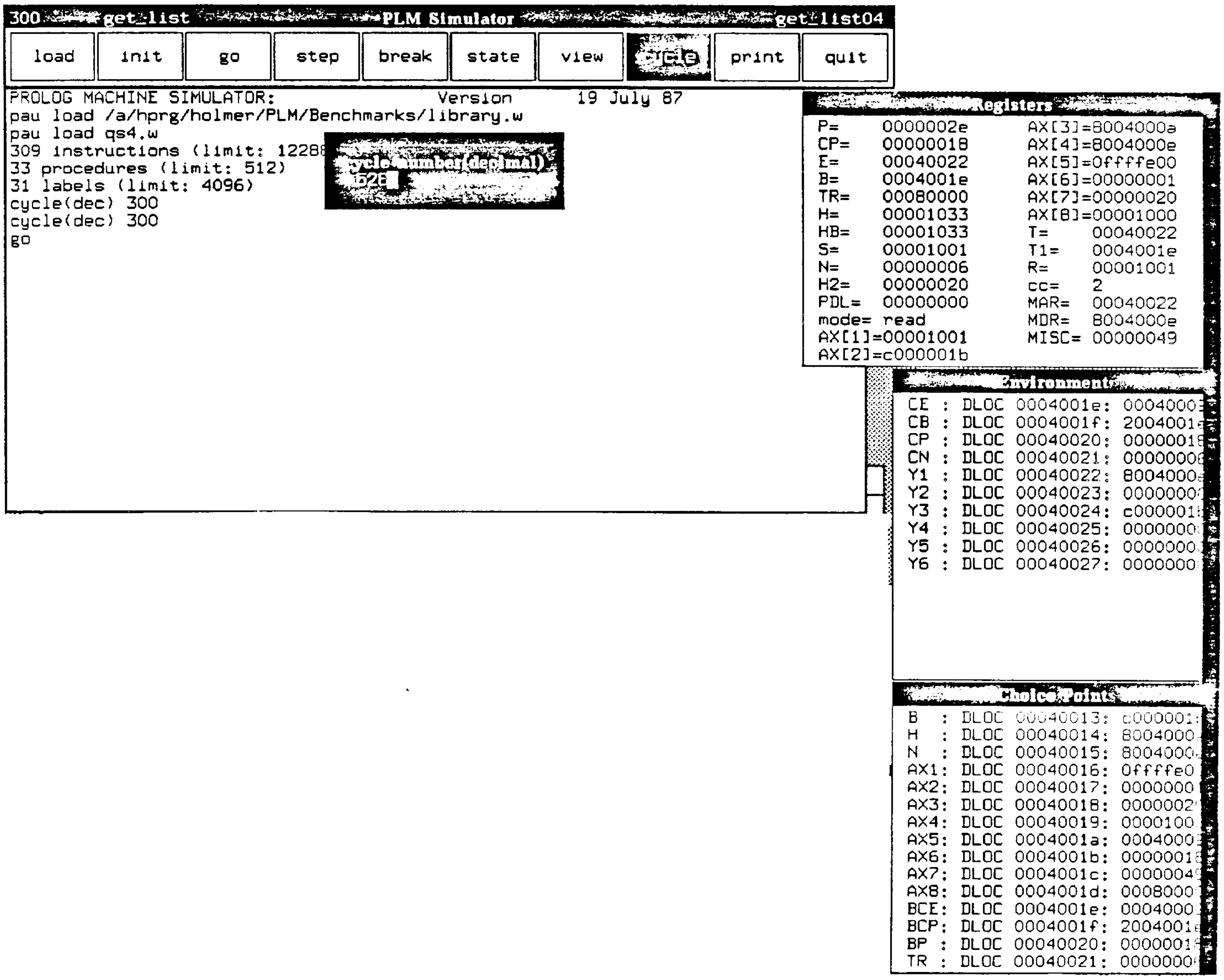
```
PROLOG MACHINE SIMULATOR:                              July 87
pau load /a/hprg/holmer/PLM/Bench
pau load qs4.w
309 instructions (limit: 12288)
33 procedures (limit: 512)
31 labels (limit: 4096)
cycle(dec) 300
go
break point 5e
```

Figure A.3: The Menu for Removing Breakpoints



```
PROLOG MACHINE SIMULATOR:
pau load /a/hprg/holmer/PLM/Bench
pau load qs4.w
309 instructions (limit: 12288)
33 procedures (limit: 512)
31 labels (limit: 4096)
cycle(dec) 300
go
break point 5e
```

Figure A.4: The Menu for Listing Breakpoints



```
PROLOG MACHINE SIMULATOR:                Version      19 July 87
pau load /a/hprg/holmer/PLM/Benchmarks/library.w
pau load qs4.w
309 instructions (limit: 12288
33 procedures (limit: 512)
31 labels (limit: 4096)
cycle(dec) 300
cycle(dec) 300
go
```

```
P=      0000002e        AX[3]=8004000a
CP=     00000018        AX[4]=8004000e
E=      00040022        AX[5]=0ffffe00
B=      0004001e        AX[6]=00000001
TR=     00080000        AX[7]=00000020
H=      00001033        AX[8]=00001000
HB=     00001033        T=      00040022
S=      00001001        T1=     0004001e
N=      00000006        R=      00001001
H2=     00000020        cc=     2
PDL=    00000000        MAR=    00040022
mode= read              MDR=    8004000e
AX[1]=00001001          MISC=   00000049
AX[2]=c000001b
```

```
CE  : DLOC 0004001e: 0004000
CB  : DLOC 0004001f: 2004001
CP  : DLOC 00040020: 0000001
CN  : DLOC 00040021: 0000000
Y1  : DLOC 00040022: 8004000
Y2  : DLOC 00040023: 0000000
Y3  : DLOC 00040024: c000001
Y4  : DLOC 00040025: 0000000
Y5  : DLOC 00040026: 0000000
Y6  : DLOC 00040027: 0000000
```

```
B   : DLOC 00040013: c000001
H   : DLOC 00040014: 8004000
N   : DLOC 00040015: 8004000
AX1: DLOC 00040016: 0ffffe0
AX2: DLOC 00040017: 0000000
AX3: DLOC 00040018: 0000002
AX4: DLOC 00040019: 0000100
AX5: DLOC 0004001a: 0004000
AX6: DLOC 0004001b: 0000001
AX7: DLOC 0004001c: 0000004
AX8: DLOC 0004001d: 0008000
BCE: DLOC 0004001e: 0004000
BCP: DLOC 0004001f: 2004001
BP  : DLOC 00040020: 0000001
TR  : DLOC 00040021: 0000000
```

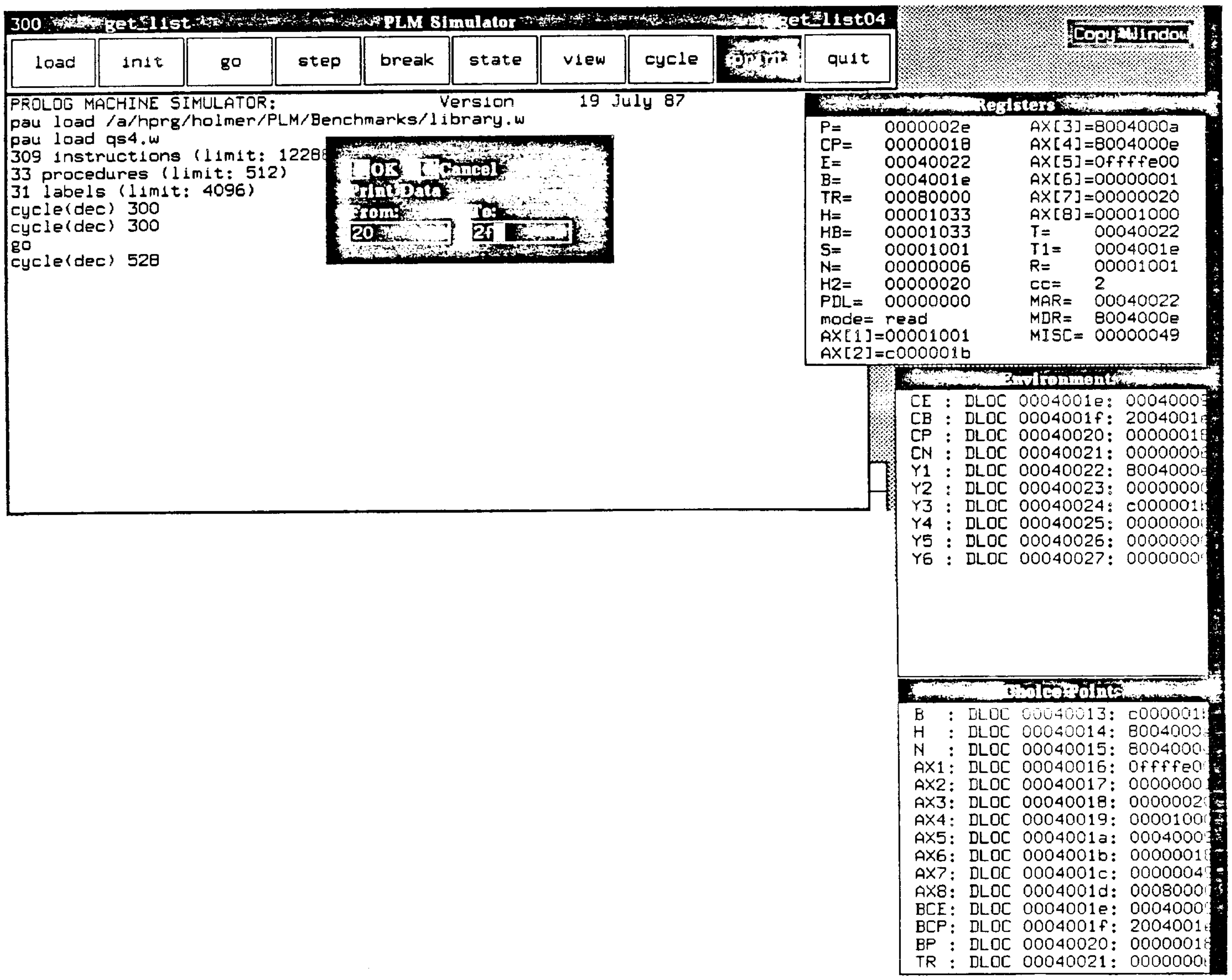
Figure A.5: The Dialogue Box for The Cycle Command

decimal digits only; no other characters are accepted. Control keys are recognized, however. *Control-H* or *delete* key is the *erase* character; *control-U* or *control-X* is the *kill-line* character. The mouse should be placed inside the dialogue box for DUES to accept any input for the *cycle* command. If the mouse is moved out of the dialogue box, the command will be aborted. As usual, DUES reads in the number upon seeing a *carriage-return*, and sends the PLM simulator the *cycle* command (c). No reply is expected.

A.6. The Print Command

The *print* command prints the contents of memory in the *Data Space*. Clicking on the command button will invoke a dialogue box. Fig. A.6 shows the dialogue box for the print command. Users enter the address range in the *Data Space* to be printed in two input areas labeled with *From* and *To*. A cursor is shown in one of these two input areas indicating which one is active, where the keyboard input should be sent to. Initially, the input area *From* is active. At a *carriage-return*, the other input area *To* becomes active until another *carriage-return* is received when the command is completed. The cursor can also be switched between input areas by moving the mouse around. As with other input areas, control keys are recognized. *Control-H* or *delete* key is the *erase* character; *control-U* or *control-X* is the *kill-line* character.

Two buttons, *Cancel* and *OK* are also available. Clicking on the *Cancel* button will abort the *print* command while clicking on the *OK* button will initiate the following procedure. DUES first checks if the *To* area is empty. If it is, the string received by the *From* area will be copied to the *To* area. The semantics for the PLM simulator maintains that if the *To* argument is not specified, it will print all data till the end of the *Data Space*. We make this change because most users expect only one memory location specified by the *From* argument to be printed. DUES sends the simulator command *print-from-data-space* (pd). The reply message which consists of data and their locations will be displayed in the *Dialogue Window*.



```
PROLOG MACHINE SIMULATOR:              Version      19 July 87
pau load /a/hprg/holmer/PLM/Benchmarks/library.w
pau load qs4.w
309 instructions (limit: 1228
33 procedures (limit: 512)
31 labels (limit: 4096)
cycle(dec) 300
cycle(dec) 300
go
cycle(dec) 528
```

Registers

```
P=      0000002e      AX[3]=8004000a
CP=     00000018      AX[4]=8004000e
E=      00040022      AX[5]=0ffffe00
B=      0004001e      AX[6]=00000001
TR=     00080000      AX[7]=00000020
H=      00001033      AX[8]=00001000
HB=     00001033      T=      00040022
S=      00001001      T1=     0004001e
N=      00000006      R=      00001001
H2=     00000020      cc=     2
PDL=    00000000      MAR=    00040022
mode= read            MDR=    8004000e
AX[1]=00001001        MISC=   00000049
AX[2]=c000001b
```

Environment

```
CE  : DLOC 0004001e: 0004000
CB  : DLOC 0004001f: 2004001
CP  : DLOC 00040020: 0000001
CN  : DLOC 00040021: 0000000
Y1  : DLOC 00040022: 8004000
Y2  : DLOC 00040023: 0000000
Y3  : DLOC 00040024: c000001
Y4  : DLOC 00040025: 0000000
Y5  : DLOC 00040026: 0000000
Y6  : DLOC 00040027: 0000000
```

ChoicePoint

```
B   : DLOC 00040013: c000001
H   : DLOC 00040014: 8004000
N   : DLOC 00040015: 8004000
AX1: DLOC 00040016: 0ffffe0
AX2: DLOC 00040017: 0000000
AX3: DLOC 00040018: 0000002
AX4: DLOC 00040019: 0001000
AX5: DLOC 0004001a: 0004000
AX6: DLOC 0004001b: 0000001
AX7: DLOC 0004001c: 0000004
AX8: DLOC 0004001d: 0008000
BCE: DLOC 0004001e: 0004000
BCP: DLOC 0004001f: 2004001
BP  : DLOC 00040020: 0000001
TR  : DLOC 00040021: 0000000
```

Figure A.6: The Dialogue Box for The Print Command

A.7. The State Command

The *state* command provides a way to select machine states, which are grouped into three sets *Registers, Environment,* and *Choice Point.* Each set of states has its contents shown in one of the *State Display Windows.* The selected states can be duplicated to the *Copy Window* later by clicking on their corresponding *State Display Window.*

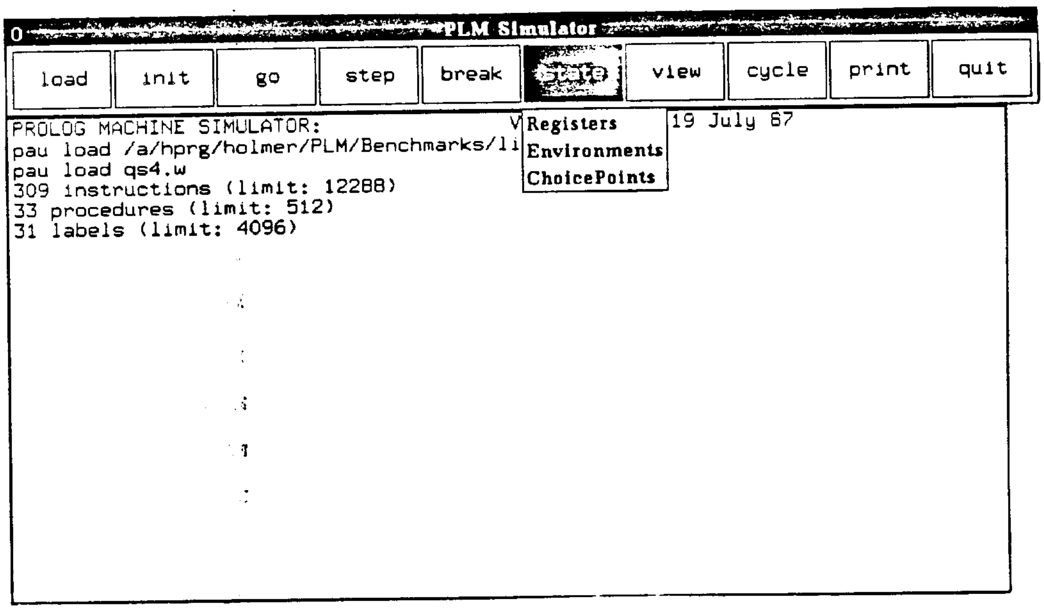
Clicking on the command button invokes a pull-down menu, which contains options to allow further selecting one of the machine states among *Register, Environment,* and *Choice Point.* As the mouse is moved over an option, the option will be highlighted. Fig. A.7 shows the pull-down menu for the state command.

The selection is made by clicking on the desired option. As a result, a sub-menu will appear under the command button, where the names of the machine states are listed. Users can select one machine state, whether it is a register, part of *Environment,* or part of *Choice Point,* by clicking on the name on the sub-menu. Consequently, the name will be redrawn in reverse video and a star sign ("*") will be labeled on the corresponding *State Display Window.* To unselect, simply click on the selected item again. It will be redrawn in normal video, and the corresponding star sign will be cleared, too. Two buttons, *Select All* and *Unselect All,* are provided to let users select everything or clear the selection in one click of mouse button. When the selection is done, clicking on the *OK* button will bring the command to an end. Fig. A.8 A.9 and A.10 show the menus for selecting *Registers, Environment* variables, and *Choice Point* variables, respectively.

The *state* command is an enhancement to the PLM simulator. There are no similar facilities implemented in the PLM simulator, thus, no command is sent by DUES. This feature, in conjunction with the *Copy Window,* works very well to alleviate users from the need for any paper and pencils.

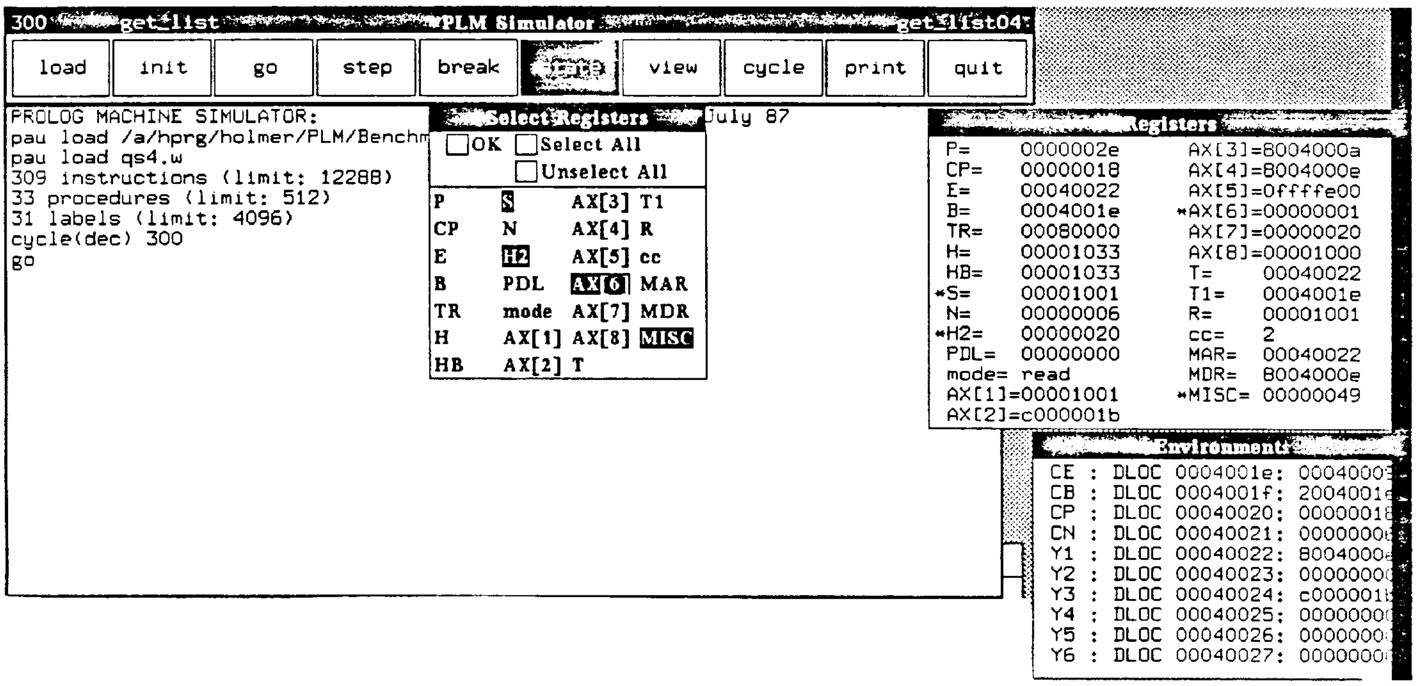
A.8. The View Command

The *view* command causes the *View Window* to pop up on the screen, which displays



```
PROLOG MACHINE SIMULATOR:                       V    19 July 87
pau load /a/hprg/holmer/PLM/Benchmarks/li
pau load qs4.w
309 instructions (limit: 12288)
33 procedures (limit: 512)
31 labels (limit: 4096)
```

Figure A.7: The Pull-down Menu for the State Command

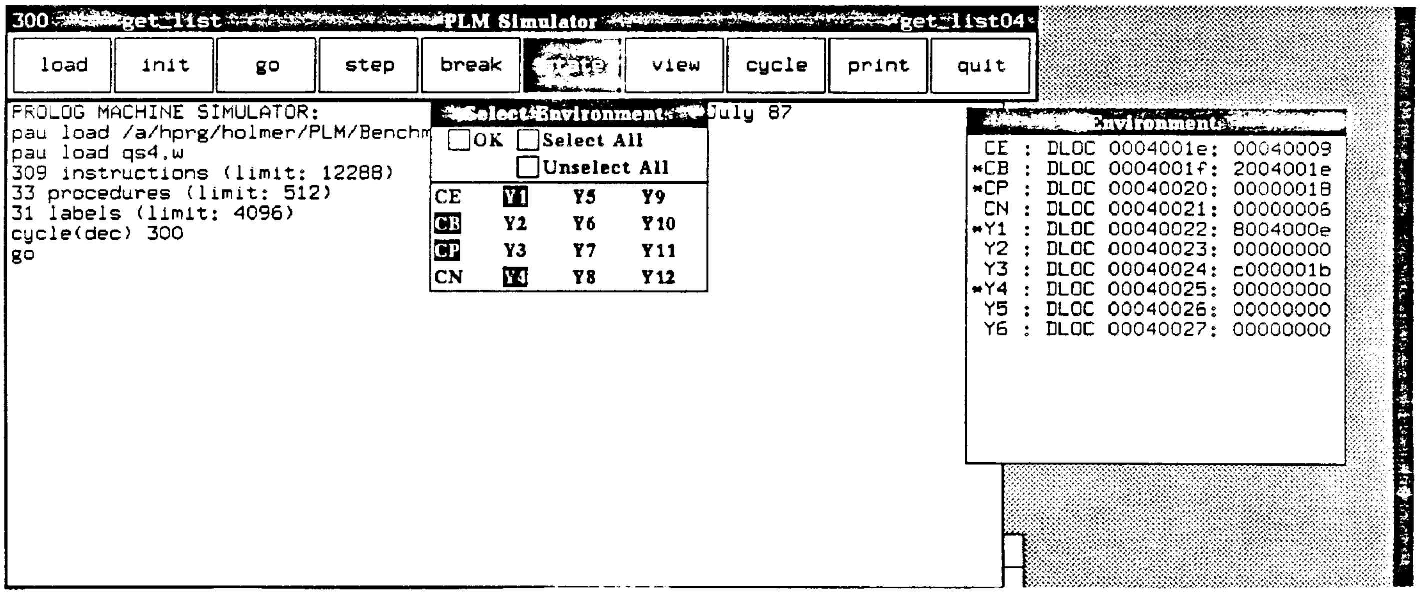


```
PROLOG MACHINE SIMULATOR:
pau load /a/hprg/holmer/PLM/Benchm
pau load qs4.w
309 instructions (limit: 12288)
33 procedures (limit: 512)
31 labels (limit: 4096)
cycle(dec) 300
go
```

```
 P=     0000002e       AX[3]=8004000a
 CP=    00000018       AX[4]=8004000e
 E=     00040022       AX[5]=0ffffe00
 B=     0004001e      *AX[6]=00000001
 TR=    00080000       AX[7]=00000020
 H=     00001033       AX[8]=00001000
 HB=    00001033       T=     00040022
*S=     00001001       T1=    0004001e
 N=     00000006       R=     00001001
*H2=    00000020       cc=    2
 PDL=   00000000       MAR=   00040022
 mode= read            MDR=   8004000e
 AX[1]=00001001       *MISC=  00000049
 AX[2]=c000001b
```

```
CE : DLOC 0004001e: 0004000
CB : DLOC 0004001f: 2004001
CP : DLOC 00040020: 0000001
CN : DLOC 00040021: 0000000
Y1 : DLOC 00040022: 8004000
Y2 : DLOC 00040023: 0000000
Y3 : DLOC 00040024: c000001
Y4 : DLOC 00040025: 0000000
Y5 : DLOC 00040026: 0000000
Y6 : DLOC 00040027: 0000000
```

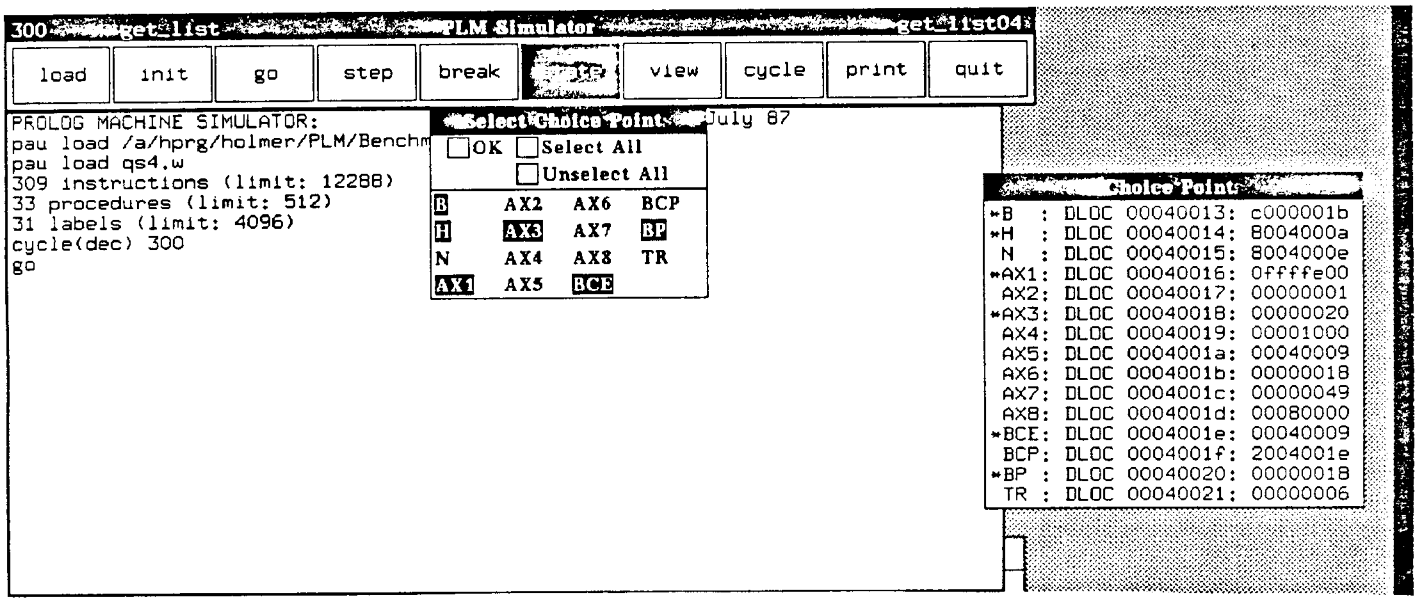
Figure A.8: The Menu for Selecting Registers



```
PROLOG MACHINE SIMULATOR:                          July 87
pau load /a/hprg/holmer/PLM/Benchm
pau load qs4.w
309 instructions (limit: 12288)
33 procedures (limit: 512)
31 labels (limit: 4096)
cycle(dec) 300
go
```

```
 CE  : DLOC 0004001e: 00040009
*CB  : DLOC 0004001f: 2004001e
*CP  : DLOC 00040020: 00000018
 CN  : DLOC 00040021: 00000006
*Y1  : DLOC 00040022: 8004000e
 Y2  : DLOC 00040023: 00000000
 Y3  : DLOC 00040024: c000001b
*Y4  : DLOC 00040025: 00000000
 Y5  : DLOC 00040026: 00000000
 Y6  : DLOC 00040027: 00000000
```

Figure A.9: The Menu for Selecting Environment Variables



```
PROLOG MACHINE SIMULATOR:                     July 87
pau load /a/hprg/holmer/PLM/Benchm
pau load qs4.w
309 instructions (limit: 12288)
33 procedures (limit: 512)
31 labels (limit: 4096)
cycle(dec) 300
go
```

```
*B   : DLOC 00040013: c000001b
*H   : DLOC 00040014: 8004000a
 N   : DLOC 00040015: 8004000e
*AX1: DLOC 00040016: 0ffffe00
 AX2: DLOC 00040017: 00000001
*AX3: DLOC 00040018: 00000020
 AX4: DLOC 00040019: 00001000
 AX5: DLOC 0004001a: 00040009
 AX6: DLOC 0004001b: 00000018
 AX7: DLOC 0004001c: 00000049
 AX8: DLOC 0004001d: 00080000
*BCE: DLOC 0004001e: 00040009
 BCP: DLOC 0004001f: 2004001e
*BP  : DLOC 00040020: 00000018
 TR  : DLOC 00040021: 00000006
```

Figure A.10: The Menu for Selecting Choice Point Variables

the compiled Prolog code specified from the command line. The *view* button remains reverse-videoed until a second click on the command button when the *View Window* will disappear.

The *View Window* also associates each line of code with a sequence number, which can be used to set breakpoints. Moreover, it also has a pointer ("**") indicating which instruction is under execution. Additionally, the scroll bar on the right border of the *View Window* is provided to allow users to browse through the code conveniently.

A.9. The Load Command

The *load* command was originally designed to notify the simulator to load a new program file; in the case of the PLM simulator, it is the compiled Prolog code. It should prompt for the file name, and send it to the simulator. Although, the current PLM simulator does not have this feature, this capability can still be simulated by terminating the old simulator process and starting a new one with appropriate program file. It is decided not to be implemented this way because of the idea of *separation of policy and mechanism*. DUES should not do any policy making; rather, it should just implement the mechanism--sending commands to the simulator instead of deciding what the simulator should behave. Finally, this command is kept for future upgrading.

A.10. The Init Command

The *init* command was originally designed to notify the simulator to initialize itself; for example, to reset the program counter. Although, the current PLM simulator does not have this feature, this capability can still be simulated by terminating the old simulator process and starting a new one with the same old arguments. Again, it is not implemented this way because of the idea of *separation of policy and mechanism*. Still, this command is still kept for future upgrading.

A.11. The Quit Command

The *quit* command kills the simulator process and exits. All windows created by DUES will be closed. This command brings the DUES session to an end.

Appendix B

Catalog of DUES Routines

Catalog of DUES Routines

PROCEDURE	*SOURCE FILE*
AddTextWindow	Xtextlib.c
assignCodeNum	view.c
Change_text_window_size	Xtextlib.c
Clear_lines	Xtextlib.c
clearOtherButtons	gasix.c
Count_lines	Xtextlib.c
createAddBPMenu	break.c
createBreakPointMenu	break.c
createListBPMenu	break.c
createMapChoicePtDpy	display.c
createMapEnvDpy	display.c
createMapRegDpy	display.c
createRemoveBPMenu	break.c
createSelectChoicePtMenu	state.c
createSelectEnvMenu	state.c
createSelectRegMenu	state.c
createSimulatorProcess	gasix.c
createStateMenu	state.c
createViewWindow	view.c
DelTextWindow	Xtextlib.c
dimCmdWin	gasix.c
dimStItem	state.c
do_addBreakPoint	break.c
do_cpDpyEvent	display.c
do_envDpyEvent	display.c
do_listBreakPoint	break.c

Catalog of DUES Routines

PROCEDURE	*SOURCE FILE*
do_regDpyEvent	display.c
do_removeBreakPoint	break.c
do_scrollBarEvent	view.c
Do_text_string	Xtextlib.c
do_viewIconWindowEvent	view.c
do_viewWindowEvent	view.c
drawAddBPMenu	break.c
drawBox	graphics.c
drawBreakPoint	break.c
drawBreakPointMenu	break.c
drawCmd	state.c
drawCmdBox	gasix.c
drawCmdMenu	gasix.c
drawCpDpy	display.c
drawCpMenu	state.c
drawCycleBox	cycle.c
drawEnvDpy	display.c
drawEnvMenu	state.c
drawListBPMenu	break.c
drawPDBox	pd.c
drawRegDpy	display.c
drawRegMenu	state.c
drawRemoveBPMenu	break.c
drawStateMenu	state.c
drawTitle	gasix.c
endOfReply	reply.c

Catalog of DUES Routines

PROCEDURE	*SOURCE FILE*
expand	view.c
f_break	break.c
f_Cancel	pd.c
f_cycle	cycle.c
f_go	load.c
findMenuCmd	state.c
f_init	init.c
firstToken	reply.c
f_load	load.c
f_modify	modify.c
f_OK	pd.c
f_print	pd.c
f_quit	load.c
f_state	state.c
f_step	step.c
f_view	view.c
getaddr	break.c
getCmdWinIndex	gasix.c
getCode	reply.c
getCpVal	reply.c
getCycle	reply.c
getCycleInput	cycle.c
getDialogStr	load.c
getEnvVal	reply.c
getInput	pd.c
getLine	reply.c

Catalog of DUES Routines

PROCEDURE	*SOURCE FILE*
getMicrocode	reply.c
getMsgs	gasix.c
getRegVal	reply.c
hextoi	break.c
highLightCmdWin	gasix.c
highlightStItem	state.c
initCopyWin	copy.c
initCycleBox	cycle.c
initPDBox	pd.c
initSelState	state.c
initStateDisplay	display.c
initViewWindow	view.c
isAddButton	break.c
isdec	break.c
ishex	break.c
isListButton	break.c
isRemoveButton	break.c
killChild	gasix.c
main	gasix.c
markAllCp	state.c
markAllEnv	state.c
markAllReg	state.c
markCurrentCode	view.c
min	break.c
MyXQueryWidth	misc.c
nextTabStop	Xtextlib2.c

Catalog of DUES Routines

PROCEDURE	*SOURCE FILE*
Normalize	Xtextlib.c
paintScrollBar	view.c
paintTitle	view.c
paintViewIcon	view.c
paintViewWindow	view.c
printCp	state.c
printCurrentInstruction	gasix.c
printCycle	gasix.c
printEnv	state.c
printReg	state.c
processEvent	gasix.c
processReply	reply.c
Redisplay_lines	Xtextlib.c
redrawMain	gasix.c
reverseVideo	gasix.c
Scroll_text_window	Xtextlib.c
selectChoicePoints	state.c
selectEnvironments	state.c
selectRegisters	state.c
Spin_lines	Xtextlib.c
standardVideo	gasix.c
storeWindowNames	names.c
TextClear	Xtextlib.c
TextCreate	Xtextlib.c
TextDestroy	Xtextlib.c
TextEvent	Xtextlib.c

Catalog of DUES Routines

PROCEDURE	*SOURCE FILE*
TextFlush	Xtextlib.c
TextPrintf	Xtextlib.c
TextPutChar	Xtextlib2.c
TextPutInputBuf	Xtextlib2.c
TextPutString	Xtextlib.c
TextRedisplay	Xtextlib.c
toggleCp	state.c
toggleEnv	state.c
toggleReg	state.c
UnmarkAllCp	state.c
UnmarkAllEnv	state.c
UnmarkAllReg	state.c