INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of "sectioning" the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.

University Microfilms International
300 N. Zeeb Road
Ann Arbor, MI 48106

1327829

Chen, Jawji

DESIRE-P IMPLEMENTATION IN THE INTEL 286/310 MICROCOMPUTER

The University of Arizona M.S. 1986

University
Microfilms
International 300 N. Zeeb Road, Ann Arbor, MI 48106

DESIRE-P IMPLEMENTATION IN THE INTEL 286/310 MICROCOMPUTER

by

Jawji Chen

A Thesis Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

MASTER OF SCIENCE
WITH A MAJOR IN ELECTRICAL ENGINEERING

In the Graduate College

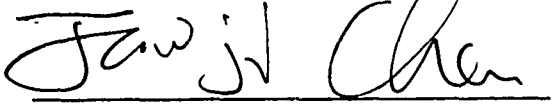
THE UNIVERSITY OF ARIZONA

1 9 8 6

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major deparment of by the Dean of the Graduate College when in his/her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: 

APPROVAL BY THE THESIS DIRECTOR

This thesis has been approved on the date shown below:

RALPH MARTINEZ
Associate Professor of Electrical and Computer Engineering

18 April 86
Date

ACKNOWLEDGMENTS

The author wishes to express his appreciation to his advisor and committee chairman, Dr. Ralph Martinez, for his guidance and professional assistance throughout this thesis. The author would also like to thank the other committee members, Dr. Frederick J. Hill and Dr. Francois E. Cellier, for their suggestions.

The author would also like to thank his wife, his father and mother, for their support and encouragement. This work is dedicated to them.

TABLE OF CONTENTS

TABLE OF CONTENTS--Continued

LIST OF ILLUSTRATIONS

LIST OF TABLES

ABSTRACT

DESIRE-P, Direct Execution Simulation in Real Time, is currently running in a VAX-11/750/VMS system. The objective of the thesis is to convert DESIRE-P to execute in the INTEL 286/310 microcomputer system under the iRMX-286 operating system. To accomplish the objective, it was necessary to convert the incompatible areas of DESIRE-P from one operating system to another. These conversions included input/output operations, assembly code routines, and code generation processes and run under the iRMX-286 operating system. Because iRMX-286 is a user configurable operating system, it was necessary to write additonal programs under iRMX-286 in supporting the objectives. These conversion efforts, nevertheless, were quite successful and the DESIRE-P now executes under the iRMX-286 operating system. Sample programs were written and then execution time was measured on the INTEL 286/310 system. As expected, the execution speed was slower than that of the VAX-11/750. However, the complex DESIRE-P system now can be further researched on microcomputer systems.

CHAPTER 1

INTRODUCTION

The paper submitted to 1979 summer simulation conference (Martinez and Korn, 1979) contains a survey the history of direct execution computer architecture. A brief review of the samples are gathered here.

The first computer system built with the concept of direct execution architecture was Burroughs 85500 in 1961. The 85500 expressed the executable statements in reverse polish notation and stored to the hardware stack. An ALGOL 60 design (Anderson, 1961) used three stack memories and pointers acted as an extention of 85500. The stacks were used to process the control states, operators, and operands. The SNOBOL computer (Rice and Smith, 1971) used several dedicated processors to execute the SNOBOL language. The modules are hardware which made the implementation and extension difficult. Several APL implementations had also appeared (Zak, 1979; Nissen, 1973). Other direct execution computer emerged are SNOBOL 4, HYDRA, PL/1, and the IPL processors.

The University of Arizona Computer Enginering Research Laboratory (CERL) in the Electrical and Computer

Engineering Department started the research of direct execution languages for continuous system simulation during the late 1970's. However, DARE I (Goltz, 1970) was the first continuous system simulation language developed in CERL and was written in PDP-9 assembly language. During the 1970's there were several generations of the continuous system simulation systems, such as DARE II (Liebert, 1970), and DARE/ELEVEN (Martinez, 1976). The later versions of DARE systems replaced their ancestors with improved performance and better structured contract statement like REPEAT, WHILE, FOR and the screen editor. However, these simulation systems were written in PDP-11 assembly languages and Fortran and were confined to running in DEC PDP-11 systems. Moreover, the code generation process for these systems was slow. It often took two to ten minutes for these systems to translate from the high level problem description to executable code. The user would have to wait until the translation, Fortran compilation, and linking processes were finished before he could run the simulation problem. These problems led to the development of Direct Execution Simulation in REal time (DESIRE) language system such as MICRODARE (Korn, 1979) and EARLY DESIRE (Korn, 1982). The latest version is DESIRE-P (Vakilzadian, 1985), which is currently running in a DEC VAX-11/750. DESIRE-P requires no translation, Fortran compilation and

linking processes. Subsequently, the time was saved due to the lack of these processes. DESIRE-P was written in Pascal so that it could be converted to other computers. This made it possible for DESIRE-P to implement to another computer (i.e., it is portable).

1.1. Objective

The objectives for this thesis were:

1. Convert DESIRE-P to execute in the INTEL 286/310 microcomputer and the iRMX-286 operating system.
2. Compare performance of DESIRE-P under iRMX-286 and VAX-11/750 in VMS operating system.

It is interesting to test the performance and portability by using 286/310 microcomputer system in the CERL to implement DESIRE-P. INTEL 286/310 microcomputer and iRMX-286 was the only new available resource in the CERL when the project started.

1.2. Approach

This project explored the implementation of DESIRE-P in INTEL 286/310 microcomputer with the following approaches:

1. Convert the source code to iRMX-286 environment and make it execute in the INTEL 286/310 microcomputer.

2. Modify the code generation process so that the code generated would be able to run directly by INTEL 80286 and 80287 processors.
3. Find the ways to recover from the errors without losing the control of the user program.
4. Implement the necessary operating system commands to help the user in developing DESIRE-P program.
5. Test the DESIRE-P system on the INTEL 286/310 microcomputer by timing execution speed with established DESIRE-P benchmark problems and compare the times to previously run VAX-11/750 times.
6. Determine the necessary system modifications to improve the execution speed of DESIRE-P on the 286/310 system.

CHAPTER 2

DIRECT EXECUTION SIMULATION SYSTEMS

Conventionally, simulation programs written with high level language would have to first translate into compiler or assembler-based languages, relocatable object code and linked with the simulation run library to produce a executable module. The direct execution simulation system (Korn and Martinez, 1979) accepts the high level statements and execute the statements without explicit calls to compiler, or linking/loader. The users sitting at the terminal would get the response at the flick of the key board switch.

A direct execution simulation system permits the user to enter or modify programs/data as well as to run the program interactively. The user can then modify simulation problem parameters or models based on the results. The direct execution simulation systems plays an important role in helping the user analyze and design the continuous system and modeling. This chapter discusses requirements for direct execution simulation, and the simulation with DESIRE-P.

2.1. Direct Execution Simulation Requirements

A user friendly simulation tool should satisfy the following requirements.

1. Easy Editing: The model and parameters need to be modified frequently due to the unsatisfied results.
2. Interactive Commands: Interactive systems provide several commands for user to manipulation the files, observe the data, plot data, and store results on disk.
3. Error Detection and Recovery: Error reporting is important in debugging and error recovery is required to prevent the user program from being lost.
4. Fast Execution Speed: Continuous system simulation using numerical integration techniques require repeated evaluation of differential equations which must be executed efficiently by the CPU.

2.2. Simulation with DESIRE-P

The DESIRE-P simulation system not only solves differential equations but permits convenient programming to obtain insight into experiments. The sample DESIRE-P program shown in Figure 2.1 and Figure 2.2 are Van Der Pol's equation and the PHYsiological Simulation Benchmark Experiment (Mcleod, 1966).

```
20      *** VAN DER POL"S DIFFERENTIAL EQUATION
30      *** ----------------------------------
40      DSPLY=700
50      T=0
60      DT=0.008
70      TMAX=11
80      IRULE 2
90      X=0.5
100     XDOT=0
110     FOR A=0.8 TO 1.8 STEP 0.2   DRUNR   NEXT A
115     END
120     DYNAMIC
258     *** ------------------------------------
268     D / DT X=XDOT
278     D / DT XDOT=-X+A*(1-X*X)*XDOT
288     XX=0.25*X \ YY=0.25*XDOT
298     TYPE   YY,XX
```

TIME	YY	XX
0.00000	0.125000	0.000000
0.08000	0.125000	-0.010000
0.16000	0.124200	-0.020480
0.24000	0.122562	-0.031403
0.32000	0.120049	-0.042735
0.40000	0.116636	-0.054443
0.48000	0.112275	-0.066500
0.56000	0.106955	-0.078879
0.64000	0.100645	-0.091560
0.72000	0.093320	-0.104552
0.80000	0.084958	-0.117745
0.88000	0.075539	-0.131270
0.96000	0.065042	-0.144880
1.04000	0.053452	-0.158782
1.12000	0.040753	-0.172699
1.20000	0.026938	-0.186718
1.28000	0.012000	-0.200684
1.36000	-0.004055	-0.214458
1.44000	-0.021211	-0.227856
1.52000	-0.039440	-0.240637
1.6000	-0.058691	-0.252499

.
.
.

Figure 2.1 VAN DER POL'S EQUATION

```
30    *** PHYSBE (BLOOD CIRCULATION SIMULATION)
45    IRULE 2
50    DT=0.01
60    TMAX=1
70    DSPLY=79
80    *** -------INITIAL VALUE
90    VRV=91
100   VAP=220
110   VVP=613
120   VLV=373
258   VAO=69
268   VSC=2785
278   VVC=450
288   *** -------- FUNCTION TABLES
298   DIM FAA[24],FBB[24]
308   DATA 0,.04,.08,.12,.16,.2,.24,.28,.32,.36,.4,1
318   DATA 0.0066,0.6,1,1.25,1.4,1.5,1.6,1.6,1.5,1
328   DATA 0.0066,0.0066,0,.04,.08,.12,.16,.2,.24
338   DATA .28,.32,.36,.4,1,.0033,.05,.1,.15,.17,.24
340   DATA .3,.36,.4,.3,.0033,.0033
348   READ FAA[],FBB[]
358   *** ----------------------------------------
378   PRINT "GO"
516   FOR I=1 TO 10 \ DRUNR \ NEXT I
526   *** ----------------------------------------
531   END
536   DYNAMIC
546   *** -------- MODEL EQUATION
556   FUNC SRV=FAA(T) \ FUNC SLV=FBB(T)
566   *** ----------------------------------------
576   PRV=VRV*SRV \ PAP=0.133*VAP \ PPV=0.033*VVP
586   PLV=VLV*SLV \ PAO=0.8*VAO \ PSC=0.0153*VSC
596   PVC=0.004*VVC
606   *** ----------------------------------------
616   FTV=78*LIM(PVC-PRV) \ FPV=90*LIM(PRV-PAP)
626   FPS=7*(PAP-PPV) \ FMV=17*LIM(PPV-PLV)
636   FAV=80*LIM(PLV-PAO) \ FAS=1.63*(PAO-PSC)
646   FVS=1.65*(PSC-PVC)
794   D/DT VRV=FTV=FPV \ D/DT VAP=FPV-FPS
804   D/DT VVP=FPS-FMV \ D/DT VLV=FMV-FAV
814   D/DT VAO=FAV-FAS \ D/DT VSC=FAS-FVS
824   D/DT VVC=FVS-FTV
844   OUT
854   P1=0.01*PAO-0.99
864   P2=0.01*PLV-0.99
874   DISPT   P1,P2
```

Figure 2.2 PHYSBE Benchmark Program Listing

A DESIRE-P program consists of two parts, job-control section and fast-task section. The job-control section provides the environment needed for the simulation like initialize or reset the state variables, change parameters, and repeatedly call DRUN/DRUNR for multi-run study. The DESIRE-P fast-task section which followed by the DYNAMIC statement allows user define the model by the first order differential equation and provide the output statement.

CHAPTER 3

DESIRE-P STRUCTURE

Before the source code translation can be finished, we need to know how to divide the simulation system into machine dependent and independent parts. The translation of the machine independent part (portable part) is to modify the format of VAX/VMS Pascal to the format of INTEL Pascal-86. During the process of modification, the unportable part is heavily dependent on the machine used. The distinctions between portable and unportable parts are not so obvious and need to be looked very carefully. In this section the structure of DESIRE-P system is described and the features of INTEL PASCAL-86 are presented.

3.1. Overall Program Flow

In Figure 3.1 (Korn, 1985) the overall program structure of DESIRE-P is shown.

Upon execution of the DESIRE-P program, the INITIALIZE routine is called to reset all the system parameters, define the keywords, set the addresses of the compiler-runtime libraries, and reset the user terminal channels. Then a check is made for the existence of the file :BB:PDARE.EDT to determine whether last log-out

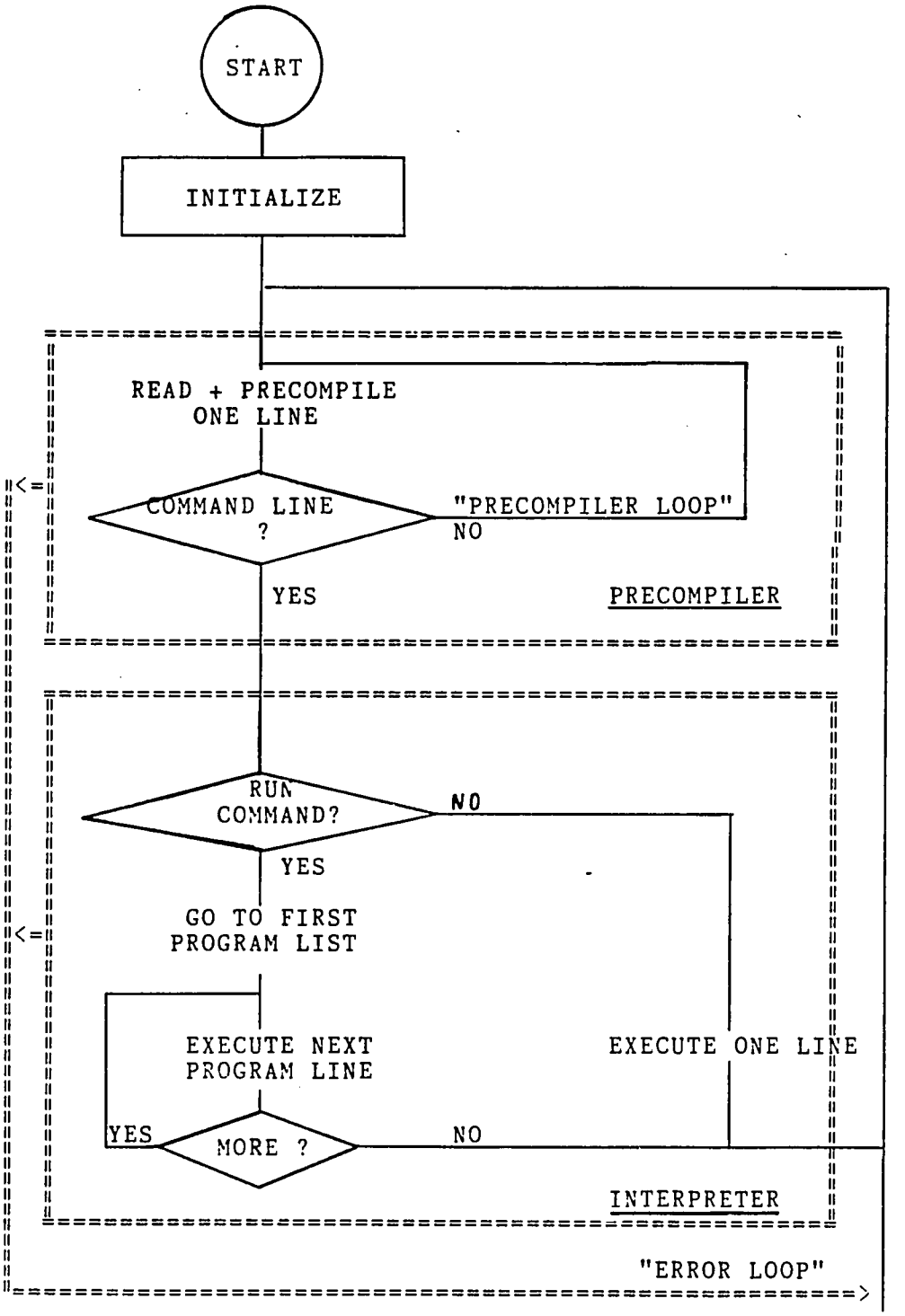


Figure 3.1 DESIRE-P Implementation Flow Chart

preserved user's program. If the file exists, it is opened and the user terminal's input channel is assigned to it. Otherwise a banner "Portable Desire INTEL Version 1, Rev. 1" is displayed followed by the current date and time, as well as the request for the terminal type. The DESIRE-P supports VT-100 and VT-52 terminals in the current version. After the terminal type is determined, the control is passed to the precompiler.

During the precompiler phase, the input can be either command or program statement. The command is the input with no line number and the program statement is the one with line number. The precompiler passes the control to interpreter if the input is command. Otherwise, the precompiler needs to parse the DESIRE-P program statements, extract state variables, and create an intermediate code which facilitates the program execution.

All commands except RUN can be executed immediately by the interpreter. The RUN command causes the intermediate code interpretation until a DRUN/DRUNR statement is encountered. These statements cause a temporary interruption of the interpreter and pass the execution control to the compiler.

The compiler compiles the fast-task segment and put the generated code in the array CAREA. The DIFEQ subroutine then executes the code in CAREA.

The control flow is passed back to precompiler at the end of interpreter.

Figure 3.2 shows the command control implemented in the iRMX-286 operating system and the INTEL 286/310 microcomputer. The original DESIRE-P structure is still retained as an individual running process. The command control program starts by creating the DESIRE-P process and DESIRE-P runs immediately. The command control process (root process) then stops the execution. The root process will never continue execution except when the DESIRE-P process detects an error or the user decides that he wants to exit from the DESIRE-P (by the command BYE), whichever should stop the execution and enable the root process for execution. The root process then delete the DESIRE-P process and distinguish the reasons why DESIRE-P process stopped execution and either re-activate (create) DESIRE-P or exit the execution.

3.2. Portable Structure of the DESIRE-P

In this section, we define the portable parts of the DESIRE-P system to be set of subroutines that meet the Standard Pascal, as well as the extensions that were identical between VAX Pascal and INTEL Pascal-86. Most of the publications refer to the Pascal of Jensen and Wirth's Pascal User Manual and Report as "Standard Pascal."

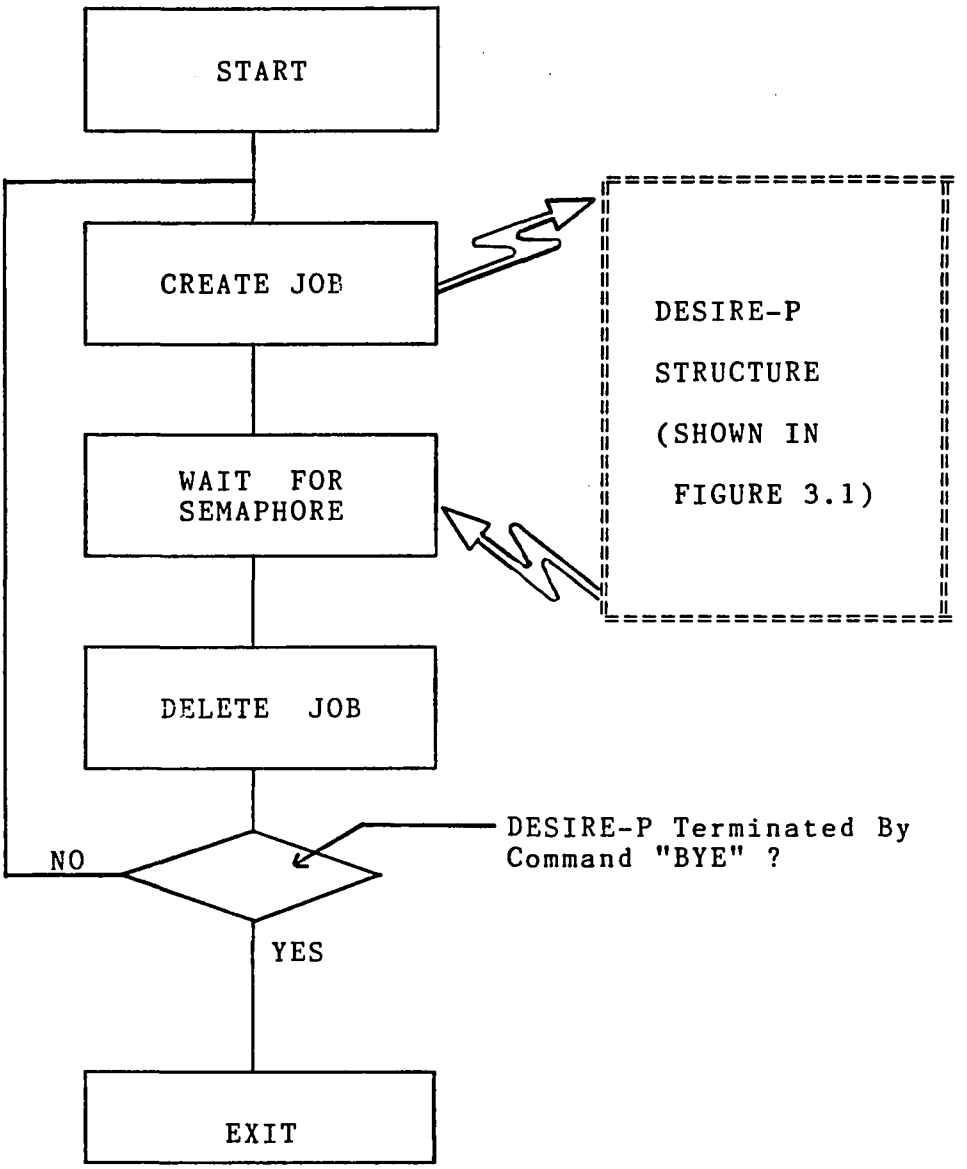


Figure 3.2 Command Control For DESIRE-P Implementation in iRMX-286

Fortunately, most of the DESIRE-P code was written in Pascal and Fortran. This was an objective of DESIRE-P and it makes the language translation process faster. However, there are still a lot of differences between the two PASCAL languages in the VAX and INTEL 286/310. The following sections described the portions of DESIRE-P that need to be converted from the VAX to the INTEL 286/310.

3.2.1. Language Symbols and Elements

The symbols that makes up the building blocks or "words" of the program such as, digits, blanks, keywords, identifiers, and special punctuation symbols are almost the same for VAX/VMS Pascal and INTEL Pascal-86 with some exceptions. The following paragraphs shows the language elements that are different and that are used to implement the DESIRE-P:

3.2.1.1. Key Words. The definitions of PUBLIC and PRIVATE in INTEL Pascal-86 replace the GLOBAL and EXTERNAL usage in VAX/VMS Pascal. Other key words used in DESIRE-P are the same.

3.2.1.2. Numbers. The Integer in VAX/VMS Pascal are represented by 32 bits whose value ranges from -2**31 to (2**31)-1. In INTEL Pascal-86, the Integer is a 16-bit

representation whose value ranges from -2**15 to (2**15)-1. The single precision real in VAX/VMS are represented by 32 bits with the precision datum approximately 7 decimal digits (normalized data) and magnitude from 0.29*(10**-38) to 1.7*(10**38). In INTEL Pascal-86, 32 bits are used for single precision real with precision datum 24 bits (not necessarily normalized) and values are in the intervals (-2**128), (0,0), and (2**-127, 2**128). The double precision real in VAX/VMS are represented by 64 bits with the precision datum approximately 16 decimal digits (normalized data). In INTEL Pascal-86, 80 bits are used for representation with values are in the intervals (-2**16384, -2**-16384), (0,0), and (2**-16384, 2**16384) to 64 bits of precision.

3.2.1.3. ASCII Character Set. A lot of ASCII characters are not supported by the INTEL high level languages, such as NULL, ESC, etc. The ASCII character that INTEL does not support are shown in Table 3.1.

3.2.2. Predefined Functions and Procedures

VAX/VMS Pascal provides more predefined functions and procedures than INTEL Pascal-86. These include:

.OPEN and CLOSE procedures used in file input and output.

Table 3-1 ASCII Character Set

Columns / Rows	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BELL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	!
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

NOTE: The Characters with underline is not supported by Intel Pascal-86

.CLOCK function for recording the central processor time in milliseconds.

3.2.3. Language Translations

The major work of the DESIRE-P program translations included the input/output operations used in the Pascal program. The file open and close for the two systems are different.

An example of the OPEN procedure used in VAX/VMS Pascal is as follows:

```
OPEN(fd[channo],fname,HISTORY=OLD,ERROR=CONTINUED) ;
```

Where

fd is an array[1..maxchan] of Text, and

fname is a packed array[1..maxnam] of Character.

The modification of this operation is to use the RESET predefined procedure in INTEL Pascal-86 to perform the file OPEN operation. The result is:

```
RESET(fd[channo],fname) ;
```

The fact that the "language elements" of iRMX-286 does not support the null character makes these two operations incompatible. We use the same file variable "fd[channo]" to connect to a different file name in "fname". Once the length of the packed array "fname" was fixed, the

length of the file name in "fname" was also fixed. Since INTEL languages do not support the null character, we would not be able to take the advantage by using the null character to fill the array "fname". The solution is to define a set of different length character packed arrays to store the different length of file name. If we need to open a file for reading then we need to store the file name based on its length into the appropriate array. Then we could use it to do the OPEN operation. We had defined the maximum file name length in INTEL DESIRE-P to 14 characters (including the pathname). Shown below is an example of an OPEN operation for a file whose name length is 6 characters.

```
RESET(fd[channo],fname6) ;
```

The same method was also used to OPEN the file for writing.

```
REWRITE(fd[channo],fname6) ;
```

The file CLOSE operation of the system in the INTEL Pascal-86 does not provide a way to close a file without destroying it. So, file closing is implemented by using PLM/86 to call a specific system function (INTEL, iRMX 86 UDI Reference Manual). The example of the CLOSE operation is:

```
file$token=D@$ATTACH(@fname6,@exception) ;
CALL D@$CLOSE(file$token,@exception) ;
```

Where fname is BYTE (6).

file$token and except is word.

NOTE: The procedure P@CLOSE is a means to close a file in INTEL Pascal-86. However, files that have not been declared in the program heading are considered temporary files and will be deleted automatically when closed or at program termination (INTEL, Pascal-86 User's Manual).

Other modifications of the portable parts of DESIRE-P are the format conversions. Examples are given below that demonstrate the conversions. The module heading for VAX/VMS Pascal is:

```
[INHERIT('global.pen')] MODULE main; or
[environment('global.pen')] MODULE submodulename ;
```

The INTEL version would be:

```
MODULE modulename ;
@INCLUDE(global.pas)
```

The file "global.pen" is the module's environment or inherited file should be extracted and put into a separate file. Then for any other module needed it can simply include the file as an environment. Section 4.1. will explain the linking process for dealing with the global symbol or procedure.

The function/procedure defined to be "GLOBAL" or "EXTERNAL" in VAX/VMS would have to be redefined under the

PUBLIC directive in INTEL Pascal-86. For example: In VAX/VMS PASCAL, the statements:

```
    PROCEDURE extername(-------) ;  EXTERN ; or
    PROCEDURE globlname(-------) ;  GLOBAL ;
```

Would be declared in INTEL Pascal-86 as follows:

```
    PUBLIC exmodulename ;
    PROCEDURE extername(-------) ;
```

OR

```
    PUBLIC globalmodulename ;
    PROCEDURE globlname(-------) ;
```

Another very important point to note is that all the files should be compiled in large modules in order to call the external routines.

Here is a sample program which shows the format difference of these two languages (i.e., VAX/VMS Pascal and INTEL Pascal-86).

```
(* INTEL Pascal-86 *)  (* VAX/VMS Pascal *)
MODULE init;           [environment('global.pen')]MODULE init;
PUBLIC main;                        .
$INCLUDE(global.pas)                .
PUBLIC init;                        .
PROCEDURE initialize;               .
PUBLIC asmcode;                     .
PROCEDURE finloc;      PROCEDURE finloc ; EXTERN ;

PRIVATE init;                       .
PROCEDURE initialize;  [GLOBAL]PROCEDURE initialize ;
BEGIN                  BEGIN
.                                   .
.                                   .
.                                   .
END;                   END;
.                      END.
```

3.3. System Variable Declaration and Initialization

Some of the variables used in DESIRE-P are initialized in the "global.pas" procedure, but the INTEL Pascal-86 doesn't support the global variable initialization. Thus, the variables must be initialized using programming code. The procedure "initialize" performs all these jobs and sets the initial values for these variables. The list of variables initialized by "initialize" is given below:

```
fdused-----all false        endtxt----2
eoflg------all false        gii-------1
token------0                 gir-------1
gi---------1                 trace-----0
jclin------0                 savindx---0
gjj--------maxhvar           nsvar-----0
```

```
datagi-----1              lf--------chr(10)
tab--------chr(9)         eos-------chr(0)
esc--------chr(27)        termtyp---chr(0)
bye--------false          exit------false
old--------false          disply----false
runf-------false          dskstor---false
compile----false          dynamic---flase
drun-------false          drun------false
abort------false          lflag-----false
arraycheck-false
```

There is a declaration need to be modified to accommodate the INTEL Pascal-86's 16-bit representation of the integer. That is

```
caddress = RECORD
           CASE adflag :  BOOLEAN OF
           TRUE :  (intnol :  INTEGER) ;
           FALSE:  (intnos :  ARRAY[1..2]OF BYTES) ;
           END ;
```

This declaration is most often used to de-concatenate the integer representation into two bytes and which then can be stored directly into the code array "carea". If we let "address" to be the variable of type "caddress", the following example demonstrates the conversion:

```
(* The Example Is To Generated The Code "B8012C" *)
(* which is the instruction "MOV AX,300h"        *)
carea[ci] := B8h;  (* OP-CODE *)
WITH address DO
BEGIN
afflag := TRUE  ;
intnot := 300h  ;
adflag := FALSE ;
carea[ci+1] := intnos[1] ;
carea[ci+2] := intnos[2] ;
END ;
ci := ci+3 ;
```

3.4. Unportable Structure of DESIRE-P

The unportable parts of the DESIRE-P includes all the assembly routines and the Pascal routine "fcompile". The procedure "fcompile" in the Pascal program "compile.pas" generates the INTEL 80286/80287 machine codes and stores them in the array "carea". In addition, all the subroutines and functions which are used to support the run time environment during the simulation run are written in assembly language. These functions and subroutines include:

1. SIN(X) : Sine of the argument X in radians
2. COS(X) : Cosine of the argument X in radians
3. AIN(X) : Arc-tangent of the argument with result in radians
4. EXP(X) : Exponential function with the power of X
5. LOG(X) : Natural logarithm
6. ABS(X) : Absolute value of X

7. TRUNC(X) : Integer part of the real value X
8. SQR(X) : Square of the argument X
9. SQRT(X) : Square root of the argument X
10. RAN(X) : Return random number with the "seed" X
11. LIM(X) : Provides MAX(X,0)
12. SIGN(X) : Return sign of the argument X; 0 if zero, -1 if negative, +1 if positive
13. SWTCH(X) : Switching function; 0 if X <= 0, 1 if X > 0
14. SATAM(X/A) : Saturation transfer function; -1 if X < -A, X/A if (-a <= X <= A), 1 if X > A
15. DEADC(X/A) : Deadspace comparator function -1 if X < -A, 0 if (-a <= X <= A), 1 if X > A
16. DEADZ(X/A) : Deadzone function X/A+1 if X < -A, 0 if (-A <= X <= A), X/A-1 if X > A
17. COMP Statement
18. FUNC Statement
19. GET Statement
20. STORE Statement
21. OUT Statement
22. SAMPLE Statement
23. TRHHLD Statement
24. TERM Statement

The input parameter and the return value for these functions are always put on the top of 80287 stack (ST). But the block operator statements, (17) through (24) above, need more than one input or output parameters. All these parameters are in floating point format. The linkage for the 80287 functions are discussed in the next chapter.

CHAPTER 4

PROGRAMMING INTERFACE FOR DESIRE-P IN THE INTEL 286/310

Most of the DESIRE-P programs are written in Pascal; yet other languages such as Fortran, assembly language, and PLM/86 play important roles in the DESIRE-P system. Since all iRMX-286 supported high level languages (except C language) use the same kind of calling convention, they are able to call each other freely. This is an advantage of the iRMX-286 support tool environment. Each program under iRMX-286 will be divided into three segments after the code has been generated (i.e., code, data, and stack segment). The global varieties with the same name will be purged to form a single data segment at link time so the program will guarantee to reference the global symbol with the same address. In the following sections the features of the iRMX-286 operating system and language environment of DESIRE-P are described.

4.1. iRMX-286 Operating System Features

The following important features of iRMX-286 (INTEL, Introduction to the iRMX 86 Operating System) help aid the development and running of DESIRE-P.

a. Support for language translators and utilities, including a standard software interface that simplifies addition of software packages to the system.
b. The iRMX-286 file system, including file utilities and system calls to manipulate files.
c. Mechanisms to bootstrap load the operating system and to load and run programs.
d. Multi-tasking environment and corresponding system calls for concurrent processing.
e. Error and system recovery handling procedures.

4.1.1. Languages Translators and Utilities

To develop programs we need language translators and utilities that allow us to compile or assemble programs, link programs together, assign absolute address to programs, create libraries of programs, and convert absolute object modules to hexadecimal format. Software packages under iRMX-286 includes:

a. ASM86 : Translates assembly instructions into object code modules.
b. PLM86 : Translates PL/M instructions into object code modules.
c. LINK86 : Combines 8086 object modules and references between independently translated modules.

d. LOC86 : Changes a relocatable object module into an absolute object module.

e. LIB86 : Creates, modifies and examines library files.

f. PASCAL86 : Translates Pascal instructions into object code modules.

g. FORT86 : Translates Fortran instructions into object code modules.

h. TX : Screen editor utility.

These packages were used in the development of DESIRE-P under iRMX-286.

4.1.2. The iRMX-286 File System

The iRMX-286 system provides three distinct types of files to ensure efficient management of both program and data files, named files, physical files, and stream files. Each file type provides access to I/O devices through the standard device driver is used to access physical and named files. Stream files act as indirect channels, through system memory, from one task to another. These channels are very useful to concurrent processing programs, for example, wishing to preserve file and device independence allowing data send to printer one time, to a disk file another time, and to another program on a different occasion.

4.1.3. Bootstrap and Application Loader

Two utilities are supplied with the system to load programs and data into system memory from secondary storage devices. The iRMX-286 bootstrap loader is typically used to load the initial system from the system disk into memory, and begin its execution. The application loader is used by application programs already running in the system to load additional program and data from and secondary storage device. The application loader is capable of loading both relocatable and absolute code, as well as program overlays.

4.1.4. Error Processing

The iRMX-286 operating system contains a default exception handler that will terminate a program if an exception occurs. The default exception handler will identify the problem by displaying on the console terminal of the exception code. If one wants to provide his own exception handler, rather than using the default exception handler, the operating system provides a mechanism for transferring control to his own exception handler.

4.2. Mechanisms of Linkage and Relocation

The language translators in INTEL such as INTEL Pascal-86, Fortran-87, and ASM-86 usually produce 80286 relocatable object modules with unresolved external

references. A relocatable object module contains no references that require it to be placed at any particular place in 80286 memory. The 80286 code contains no references to physical addresses. The translator's object output serves as input to LINK86 (INTEL., iAPX 86,88 Family Utilities User's Guide).

Link86 combines a list of 80286 object modules into a single object, and attempts to match all external symbol definitions with their public symbol definitions. LOC86 converts relocatable object module can not be executed or accessed until it resides at its assigned address. An address field that refers to a location in a different module is called an external reference. An external reference differs from a relative address because the translator that generates the module knows nothing about the location. You must declare these references as external when coding a program. This tells the translator, and subsequently the relocation and linkage commands, that the target of the reference is in a different module. A module that contains external reference is called an unsatisfied module. To satisfy the module, a module with a public symbol that matches the external symbol must be found. Associated with a public symbol in a module is an address that allows other modules, with the appropriate external reference, to

reference the module with the public symbol. You must declare these symbols as public when coding the program. This tells the source translator and linking/loader commands that other module can reference the symbol.

4.2.1. Overview of 80286 CPU

To fully understand how the languages can be integrated we must have an understanding of how 80286 programs are constructed. To do this, the main architectural features of the 286 CPU are described here.

4.2.1.1. Addressing Techniques. The system 286/310 uses the iSBC-286/10 CPU board. The 80286 operates in two modes: iAPX86 real address mode and protected virtual address mode. In iAPX86 real address mode, programs use real address space. The real address mode is the address mode used for iRMX-286 in 286/310 system. The 80286 addresses memory with a 20-bit address that is constructed from a segment address and a 16-bit offset from that segment, 64K bytes of memory is directly addressable by changing only the offset.

A hardware segment address is a 20-bit address. But the segment is constructed such that the segment is placed on the boundary that is a multiple of 16 (10 Hex). Because the low four bits of the 20-bit segment address are

always zero, the segment address can be represented with only 16 bits. The segment address is keep in one of four 16-bit segment registers (i.e., CS, DS, SS, and ES). Because there are four segments, the 80286 CPU can, at any moment, access 256K (4*64K) bytes of memory. The full one megabyte of memory is accessible by changing the value in the segment registers.

4.2.1.2. Overlaps. Sometimes the program is too large to fit into the memory available on the system. Overlaps or overlays permit programs to be larger than the available memory. Typically, an overlay is composed of code and data that is executed in one phase of a program's execution, but not used at any other time. Once executed the memory used by this code can be overwritten with code and data used in an other phase. Sections of code that occupy the same part of memory at different times during execution are called overlays. Part of an overlayed program is always resident in memory. It usually is comprised of the main program module, frequently used routines, and the overlay loader. This part of the program is called the root.

4.2.1.3. Stack. Each 80286 stack position holds one 16-bit word. Arguments passed by reference normally

take two words, the segment addresa and offset. Arguments passed by value takes one, two, four, or five bytes, depending on the data length. One byte arguments have an undefined high-order byte. The large control is used for all the language compilation. The stack sections from all modules are combined and allocated space in one logical segment.

4.2.2. Overview of 80287 Coprocessor

The programmer accessible features of the 80287 Numeric Processor architecture consist of the eight floating-point stack elements. The seven words which constitute the 80287 environment (status word, control word, tag word, 2-word instruction address, and 2-word data address) and the seven data types accessible by the 80287 (The 8086 Family User's Manual, Numerics Supplement).

4.2.2.1. Floating Point Stack. The 80287 stack consists of eight elements divided into the fields shown in Figure 4.1. The format of the fields corresponds with the temporary real data format used in all stack calculations.

4.2.2.2. Environment. The 80287 environment consists of the seven words shown in Figure 4.2.

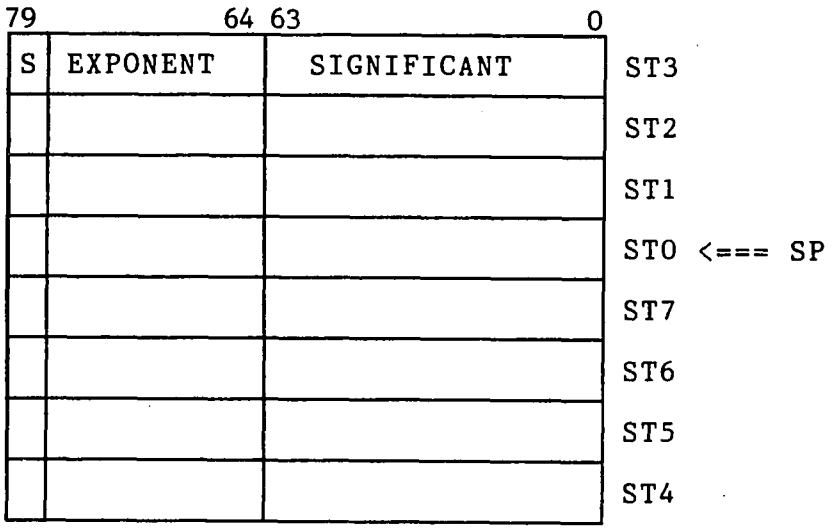


Figure 4.1 The 80287 Stack Fields

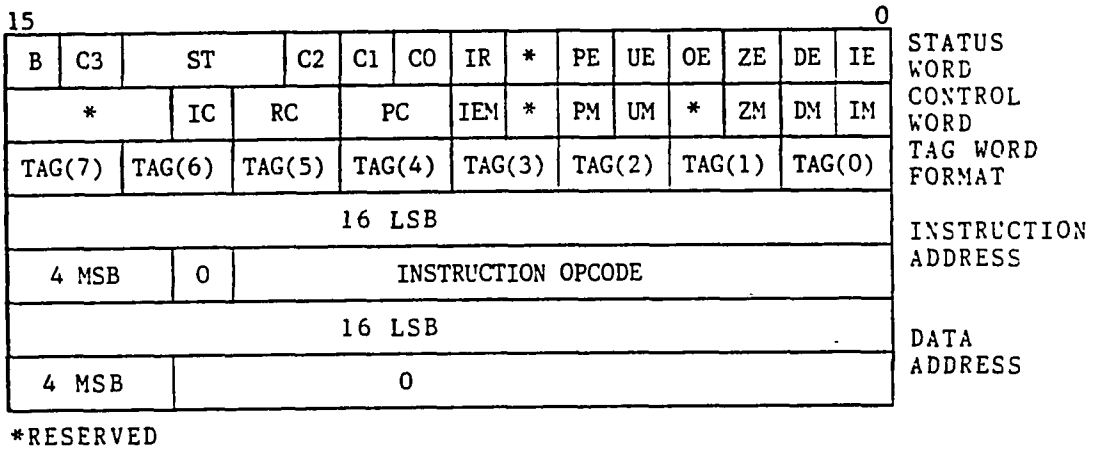


Figure 4.2 80287 Environment

4.2.2.3. Status Word. The status word reflects the overall condition of the 80287. It may be examined by storing it into memory with an 80287 instruction and then inspecting it with 80286/80287 CPU code. The status word is divided into the execution flag and status bit fields. The busy field (bit 15) indicates whether the 80287 is executing an instruction (B=1) or is idle (B=0). Bits 13-11 of the status word points to the 80287 stack element that the current stack top (ST).
Bit 7 (IR) is the interrupt request field. The 80287 latches this bit to record a pending interrupt to the 80286/80287 CPU.
Bit 5-0 (PE, UE, OE, EE, DE, and IE) are set to indicate that the 80287 has detected an exception while executing an instruction.

4.2.2.4. Control Word. The control word consists of the exception masks, an interrupt enable mask, and control bits as shown in Figure 3.2. During the execution of most instructions, the 80287 checks for six classes of except conditions (i.e., invalid operations, Overflow, Underflow, Zerodivide, Denormalized, and Precision). When one of the six conditions occurs, the corresponding flag

in the status word is set to 1. The 80287 checks the appropriate mask in the Control Word to determine if it should process the exception with a default handling procedure on chip (mask=1) or invoke a user written exception handler (mask=0).

4.2.2.5. Tag Word. The tag word contains tags describing the contents of the corresponding stack element.

Tag values:

00 = Valid (Normal or Unnormal)

01 = Zero (True)

10 = Special (Not-A-Number, Infinite, or Denormal)

11 = Empty

4.2.2.6. Data Types. The 80287 addresses seven different data types using all of the 80286 addressing modes. These data types and their valid ranges are shown in Table 3.1.

4.3. Language Integration

The language integration is not just a straight-forward thing to do, especially when we are dealing with the assembly language. In this section, the explicit and implicit problems that we were face are described.

4.3.1. Assembly Language Interface in DESIRE-P

There are two kinds of assembly language routines in DESIRE-P. The first kind is used to provide the environment like run-time libraries needed for the second kind. The second kind is the code generated in the array "carea" which we called it in the "code program" and will be discussed more detail in Sec. 4.3.2.

The code program calls the assembly language routines such as sine, cosine, . . . by their global entry points. These routine called by code program will be referred to as "code routine".

The entry address for these routines are stored in "faddress" while executing the routine "finloc". The offsets of the code routine are stored in "faddress[1]" through "faddress[24]". The address faddress[25] stores the segment address of these code routines. The parameters passed to the code routine are by value or by reference. That is, the base address and offset of a parameter may be passed by reference or by its exact value. All the parameters and data used by the code routine are real data. To simplify the code generation, we require that the calling convention for code routine follow the rules.

1. For non-block operation code routine. The parameter and the return value are in the top of 80287 stack (ST).

2. All these code routines are called by a long address call (i.e., the segment address and offset).
3. For block operation code routine. Because all the real value are put in "relval", the base address for these real variables are the same. We need only pass the offset of the real variable to the code routine. The base address of the real variable should be put in the DESIRE-P system variable "rlvloc" in the assembly language routine "finloc".

4.3.2. Code Generation and Execution

The code generation process in the Pascal-86 routine "fcompile" stores the code in the array "carea". The type checking in the Pascal compiler is so strict that it does not allow the data stored in "carea" to be treated as executable code and run it directly. INTEL ASM86 also does the type checking so there is no way to jump or call the "carea" for execution. The trick to fool the INTEL ASM86 by first pushing the segment address and its offset into stack and then use the instruction "RET" to jump to the code program. The program to illustrate the interface has been shown in Figure 4.3.

4.3.3. PLM/86 Routines

The DESIRE-P in INTEL 286/310 uses the PLM/86 to drive the graphic output for VT-100 and VT-52, to access

```
NAME      PDMAC
EXTRN     INIT87:FAR
EXTRN     CAREA:WORD
EXTRN     RELVAL:DWORD

PARA_     SEGMENT       STACK
CONADD    DW            CON
          DW            100 DUP(?)
LASTWORD  LABEL         WORD
PARA_     ENDS

CODE_     SEGMENT       PUBLIC
          PUBLIC        DIFEQ
DIFEQ     PROC          FAR
          ASSUME        CS:CODE_,SS:PARA_
          MOV           BX,PARA_
          MOV           AX,SP
          MOV           CX,SS
          MOV           SS,BX
          MOV           SP,OFFSET LASTWORD
          PUSH          CX
          PUSH          AX
          PUSH          DS
          MOV           BX,SEG RELVAL
          MOV           DS,BX
          PUSH          CS
          LEA           BX,CONADD
          MOV           AX,SS:[BX]
          PUSH          AX
          MOV           AX,SEG CAREA
          PUSH          AX,DS:CAREA
          PUSH          AX
          CALL          INIT87
          RET
CON:      POP           AX
          MOV           DS,AX
          POP           AX
          POP           BX
          MOV           SS,BX
          MOV           SP,AX
          RET
DIFEQ     ENDP
            .
            .
CODE_     ENDS
          END
```

Figure 4.3 CAREA Code Execution

the operating system commands such as the screen editor TX, DIR, TIME, and DATE, etc. (INTEL., Run-Time Support Manual for iAPX 86,88 Application), and to overwrite the default CONTROL-C task. These features are described in the following sections.

4.3.3.1. Display Module. The PLM/86 display module communicates with DESIRE-P through the modular routine "drstrt", "drstry", "grafon", "grafof", "dispxy", and "scroll". Display modules in INTEL 286/310 produces 160x80 graphics on the VT-52 equivalent alphanumeric terminals and 100x80 graphics on the VT-100 equivalent terminals.

The most important part in the display module is to use the alternate graphic character set available on DEC-VT100, GT-100, or Z-29 terminals to display points. These terminal in their VT-52 mode, display the character "f" at column "nx" and line "ny" with the direct cursor addressing escape sequence.

ESC Y CHR(ny+31) CHR(nx+31) f

The major function of dispxt-dispxy becomes to convert a fixed-point variable to the above character format with correct values for display. If the terminal is in the VT-100 equivalent mode, the direct cursor addressing escape sequence would be:

ESC [nx ; ny H

The point display routine "dixpxt/dispxy" is called only once per communication interval rather than each integration step to speed up the simulation. The display module is so independent to rest of the program that we may add and implement a new graphic device within the present structure easily.

4.3.3.2. Operating System Command. The INTEL version of DESIRE-P uses some of the existing human interface command to help the users develop their own program. Using the method discussed here almost all of the system commands (including the user commands) can be accessed by DESIRE-P. The human interface provides three system calls to facilitate this process of programmatic command invocation (INTEL., iRMX 86 Human Interface Reference Manual):

A. C$CREATE$COMMAND$CONNECTION,

B. C$SEND$COMMAND.

C. C$DELETE$COMMAND$CONNECTION.

Invoking these commands involves the following operations:

A. Command Connection Creating. Before we send the command line to the operating system to be invoked, we

must create an object (called a command) connection to store the command line. The C$CREATE$COMMAND$CONNECTION system call creates this object and returns a token for the command connection. The token can be used in calls to C$SEND$COMMAND (to send command lines to the object) and in calls to C$DELETE$COMMAND$CONNECTION (to delete the object after using it).

B. Sending Command Lines to the Command Connection and Invoking the Command. The C$SEND$COMMAND system call sends command lines to a command connection and, when the command invocation is complete, invokes the command. The command can be any iRMX-286 Human Interface (as described in the iRMX-86 Operator's Manual) or any command that user provides. The C$SEND$COMMAND invokes the command by loading the command from secondary storage and starting it running. The C$SEND$COMMAND call that invokes the command does not return control until the invoked command finishes processing. Once the command finishes processing, the flow returns to the calling program.

C. Command Connection Deleting. After we have finished invoking commands programmatically, we must delete the command connection. The C$DELETE$COMMAND$CONNECTION system call performs this operation. This frees the memory used by the data structure of the command connection.

There are four system command calls implement in DESIRE-P now. They are DIR, TIME, DATE, and TX. We may add as many system command calls as we want by using the same kind of strategy used here.

4.3.3.3. Program Control. Normally, when the program is executing, the operator can not communicate with the program unless the program request the input from the terminal. This can present problems if the operator inadvertently enters the wrong command, or if the operator decides which the command is executing that the command is unnecessary. Under these circumstances, the operator can enter a CONTROL-C character (INTEL, iRMX 86 Human Interface Manual). In the default case, the CONTROL-C causes the Human Interface to abort the currently executing program. We certainly do not with to abort the program and destroy the user's program in DESIRE-P execution. We need to override the default CONTROL-C mechanism in certain running phase of DESIRE-P. The default CONTROL-C mechanism was overrided while the DESIRE-P is in the TX (screen editor) mode or in the SIMULATION RUN mode. To override the default CONTROL-C mechanism, we must change the semaphore to which the operating system sends the unit when the operator enters a

a CONTROL-C. By changing the semaphore to one that we create, we circumvent the CONTROL-C task of the Human Interface. The program flow for this would be:

1. Call S$CREARE$SEMAPHORE to create the CONTROL-C semaphore
2. Call S$CATALOG$OBJECT to catalog the token for the semaphore in an object directory
3. Call S$ATTACH$FILE to obtain a connection to the terminal. Using logical name :CI: as the pathname parameter
4. Call S$OPEN to open the connection to the terminal for reading only
5. Call S$CREATE$TASK to start the CONTROL-C task
6. Call S$SPECIAL to switch the CONTROL-C semaphore to the one just created. Use the token for the connection to the terminal as input
7. Continue to the calling program.

4.4. Error Recovery

Most of the high level programs are susceptible to crash. The program will be aborted and control returned to the operating system if an arithmetic error or I/O error occurs during the program execution. This may lead to lose the user program if the proper action are not taken to recover from these errors. We can implement three methods

for error recovery in DESIRE-P. These methods are described in this section.

The first step for all three methods is the same. That is report the error by calling the routine "error". We specify the file "error.pas" as an interrupt procedure by using the interrupt control during the compilation. The interrupt number 16 is reserved for real arithmetic exception and we can assign the number to the interrupt either by using the interrupt control or the SETINTERRUPT building procedure. For example:

```
- :LANG:PASCAL86 error.pas LARGE INTERRUPT(error=16)
```

The interrupt number forms an interrupt vector, which is an absolutely located array of entries beginning at location 00000H. Each entry is a four-byte value containing a segment address and an offset (i.e., long pointer). In the case of real arithmetic exception exception, the 80286 processor will access the vector from the address 4x16 = 48 = 30H. The CPU uses the vector entry to make a long indirect call to activate the appropriate procedure. After the error message been reported, the program flow should direct to the main procedure (i.e., recover from the error).

We can implement three different kinds of the recovery in the implementation of DESIRE-P in INTEL 286/310. Each has its own advantages and disadvantages.

4.4.1. Direct Jump Recovery

The method calls the procedure "gomain" at the end of the interrupt routine "error". This would probably be the most straightforward and easy to implement error recovery method. But, it is a one way traffic to call "gomain". There are no returns from interrupt or subroutine in the flow. That is terribly waste of the stack space especially when the error occurred in the nested calls. If the program size is considerably small and the system has virtually large enough memory then this might not be a problem otherwise it might susceptible to the stack overflow system error and abort the program execution.

4.4.2. Smooth Recovery

The potential stack overflow error imposed in direct jump recovery is removed in the smooth recovery. We may implement the method by setting the error flag "err8087" upon returns from the interrupt handler. All the calling procedure then need to check whether the error had happened by looking into the error flag "err8087". If it had been set, the caller might need to skip the rest

of the program and return to a one level higher caller until the flow goes back to the main procedure. This method is also easy to implement but we need to do a lot of checking for the entire DESIRE-P program.

4.4.3. Multiprocessing Approach

Unlike the two methods discussed before, the multiprocessing approach requires a slight change of the structure of DESIRE-P control flow (see Chapter 3 for the structure). The DESIRE-P is treated as a separate running process (job) while the main process controls the flow for the entire execution. The error processing for DESIRE-P has three work (i.e., report the error, delete the DESIRE-P process, and re-activate the DESIRE-P process). Since the stack is initialized each time the process been created, the stack overflow problem is solved. If there is a system error in DESIRE-P, we need about 15 seconds for error processing and reporting.

4.5. iRMX-286 Memory Requirements

With iRMX-286 operating system, the static executable code size of DESIRE-P is 1,098,073 bytes. The dynamic memory (memory pool) size is virtually unlimited. The INTEL 286/310 in CERL owns 749K bytes of memory (RAM) which minus the space needed for OS and reserved codes has approximately 500K bytes free memory for application

program. The storage arrangement for the system looks like Figure 4.4 (INTEL., iRMX 86 System Configuration User's Guide). Figure 4.4 shows the current configuration of the iRMX-286. For different configurations of the iRMX system, the boundary addresses are not necessarily the same.

The locations 0:0H through 03:FH are used to store the interrupt vectors. The locations 40:0H through 7F:FH are reserved for monitors. The locations 80:0H through FF:FH are reserved for future use. Locations 100:0H through 103:FH are reserved for room for four 16 byte wake-up address area. The first of these is used by the default iSBC 215 and iSBC 220 configuration and maps to default base port address of 100H. Adhering to these recommendations for reserved address allows one to use the default address supported by the iRMX-286 BIOS. The operating system's code and data is located at the addresses 104:0H through 24D3:FH. Locations 24D4:0H through 2531:FH are for root job operations. Locations above 2532:0H are for application.

iRMX-286 SYSTEM MEMORY MAP

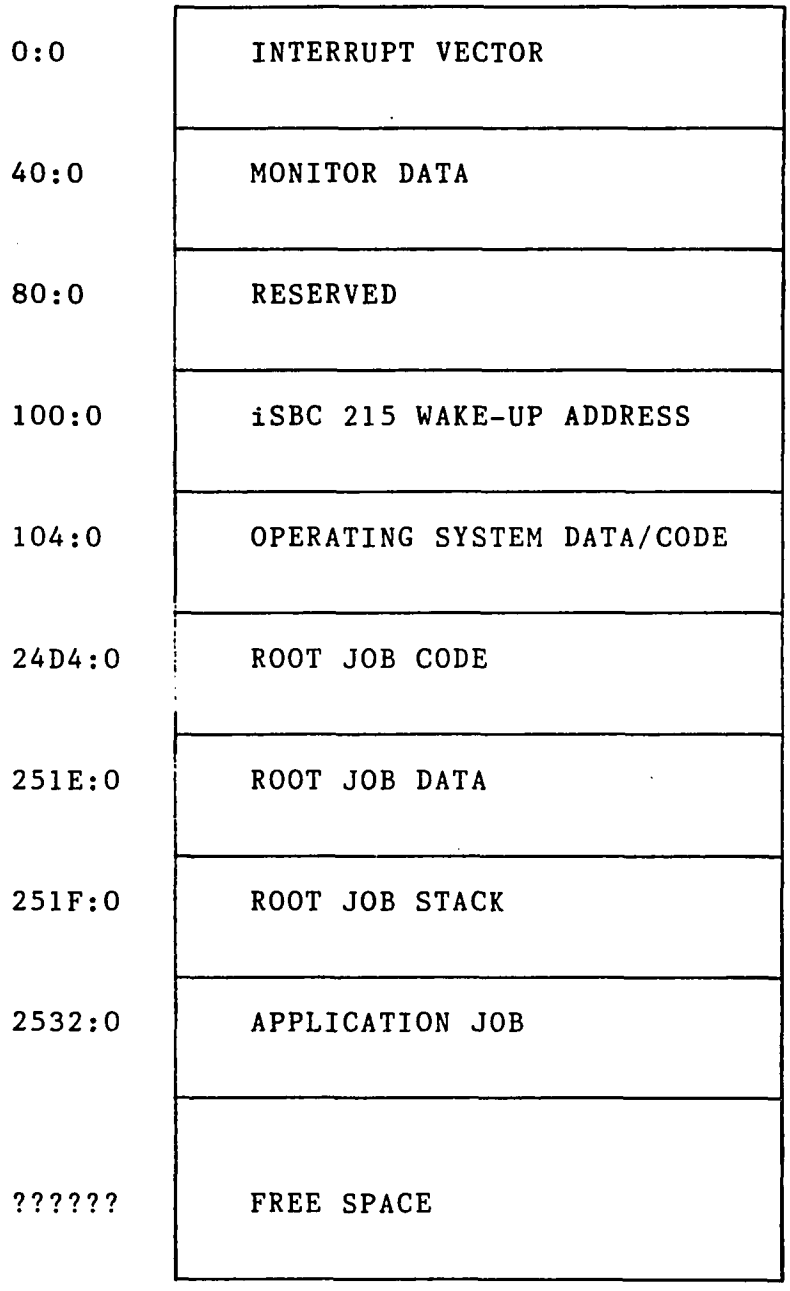


Figure 4.4 iRMX-286 System Memory Map

CHAPTER 5

CONCLUSION

The most significant part of the modification in the implementation of DESIRE-P was to rewrite the assembly code in DESIRE-P for error trapping and executing the code language (codes generated by routine "fcompile"). Some difficulties were faced but the result turned out to be good. The extensibility for the INTEL version DESIRE-P is good. We may add more functions we need to help developing the application program. This initial implementation of DESIRE-P on the 286/310 microcomputer allows additional research and development to take place which further enhances the system's capabilities.

5.1. Comparisons

Both the DESIRE-P code size and the execution speed are compared as listed in Figure 5.1 and Figure 5.2 respectively. As mentioned earlier the fast-task segment code generated in INTEL 286/310 would be 2.4 times more than that of VAX/VMS. The table also shows the execution speed for DESIRE-P in INTEL 286/310 is slower than that of in the VAX/VMS.

	VAX / VMS	iRMX 86
SOURCE CODE SIZE	403 BLOCKS	1,098,073 BYTES
EXECUTABLE CODE SIZE	171 BLOCKS	140,048 BYTES

Figure 5.1 DESIRE-P Code Size

	VAX / VMS		iRMX 86	
	CODE SIZE	TIME	CODE SIZE	TIME
PHYSBE	199 BYTES	0.18 SEC.	507 BYTES	0.79 SEC.
VAN DER POL'S	87 BYTES	0.08 SEC.	178 BYTES	0.21 SEC.

Figure 5.2 Speed and Fast-Task Segment Code Size

5.2. Difficulties

The implementation of DESIRE-P in INTEL 286/310 imposed some difficulties due to the lack of some system function support under the iRMX-286 operating system. These functions are described here.

5.2.1. Speed Limitation

Because the assembly instruction and addressing mode are not as powerful as VAX-11 assembly instruction, the code generated was considerably longer (approximately 2.4 times longer) for iRMX-286. Also the numerical data processor 80287 and 80286 operate at different clock rate. So, it is required to generate the "WAIT" instruction at the beginning of every 80287 instruction and at the last 80287 instruction for handshaking. This also made the speed slow.

5.2.2. Machine Dependency

The code program generated by routine "fcompile" is executable only by the 80286 and 80287 processors, therefore we may not run DESIRE-P if the 8087 emulator is used instead of 80287/8087 processor.

5.2.3. Potential Danger of Stack Overflow

If the arithmetic or input/output error occurred within the nested called procedure and the stack frame for

each procedure call were not poped properly, the stack might overflow. We suggest that the error handler implemented here should not be altered even while we want to add more functions to the system.

We may delay the overflow by enlarging the stack size in the program link/locate process.

5.3. Future Expansion

The following features would enhance the versatility of DESIRE-P in the INTEL 286/310.

1. Add More Operating System Commands. The commands can be interfaced by spawning a new job from the current running job. Almost all the operating system commands can be added to the DESIRE-P. This would make the DESIRE-P more easy and convenient to use.
2. More accurate graphic display on other terminals like Tektronix 4105/7. The terminal would be user selectable.
3. Optimize the code generation for faster execution.
4. The overall code size of DESIRE-P can be virtually condensed by using the overlay technique in iRMX-286.

5. The display routines used in the INTEL DESIRE-P were written in PLM/86 which can be changed to ASM86 to speed up the display processing.
6. Use an array processor single board computer on the 286/310 Multibus to perform the direct execution of fast segment code.
7. Provide a communication interface via a local area network to DESIRE-P in the VAX-11/750, so large simulations can be uploaded to the VAX.

APPENDIX A

RUNNING INTEL VERSION DESIRE-P

A.1. Language Compilation

All the routines used in INTEL DESIRE-P should be compiled under the large control to facilitate the external call. The commands are shown here.

```
- :LANG:PASCAL86 MAIN.PAS LARGE
- :LANG:PASCAL86 INIT.PAS LARGE
- :LANG:PASCAL86 PROOO.PAS LARGE
- :LANG:PASCAL86 OPENF.PAS LARGE
- :LANG:PASCAL86 PRCOMP.PAS LARGE
- :LANG:PASCAL86 LIST.PAS LARGE
- :LANG:PASCAL86 EVALEXP.PAS LARGE
- :LANG:PASCAL86 PRINT.PAS LARGE
- :LANG:PASCAL86 FORLOP.PAS LARGE
- :LANG:PASCAL86 DEF.PAS LARGE
- :LANG:PASCAL86 CALL.PAS LARGE
- :LANG:PASCAL86 DUMP.PAS LARGE
- :LANG:PASCAL86 IFOO.PAS LARGE
- :LANG:PASCAL86 INPUT.PAS LARGE
- :LANG:PASCAL86 UTIL.PAS LARGE
- :LANG:PASCAL86 OLDOO.PAS LARGE
- :LANG:PASCAL86 RESETA.PAS LARGE
- :LANG:PASCAL86 COMPILE.PAS LARGE
- :LANG:PASCAL86 SIMULATE1.PAS LARGE
- :LANG:PASCAL86 SIMULATE2.PAS LARGE
- :LANG:PASCAL86 ERROR.PAS INTERRUPT(ERROR=10H) LARGE
- :LANG:FORT86 RULE1.FOR LARGE
- :LANG:FORT86 RULE2.FOR LARGE
- :LANG:FORT86 RULE3.FOR LARGE
- :LANG:PLM86 DIR.PLM LARGE
- :LANG:PLM86 DISPLY.PLM LARGE
- :LANG:PLM86 DISP.PLM LARGE
- :LANG:PLM86 CTRL.PLM LARGE
- :LANG:PLM86 DESIREP.PLM LARGE
- :LANG:ASM86 PDMAC.ASM
```

A.2. Linker Commands

In order to run the DESIRE-P in INTEL 286/310, we need to link all the object-code files. Besides, the system library should all be linked together to form the executable object. The commands are:

```
- :LANG:LINK86 MAIN.OBJ, INIT.OBJ, PROOO.OBJ, ERROR.OBJ, &
OPENF.OBJ, PRCOMP.OBJ, LIST.OBJ, EVALEXP.OBJ, PRINT.OBJ, &
FORLOP.OBJ, DEF.OBJ, CALL.OBJ, DUMP.OBJ, IFOO.OBJ, &
INPUT.OBJ, UTIL.OBJ, DIR.OBJ, OLDOO.OBJ, RESETA.OBJ, &
COMPILE.OBJ, SIMULATE1.OBJ, SIMULATE2.OBJ, PDMAC.OBJ, &
DISPLY.OBJ, RULE1.OBJ, RULE2.OBJ, RULE3.OBJ, DISP.OBJ, &
CTRL.OBJ, &
/LIB/PASC86/P86RNO.LIB, /LIB/PASC86/P86RN1.LIB, &
/LIB/PASC86/P86RN2.LIB, /LIB/PASC86/P86RN3.LIB, &
/LIB/PASC86/CEL87.LIB, /LIB/PASC86/EH87.LIB, &
/LIB/PASC86/8087.LIB, /RMX86/LIB/LARGE.LIB, &
/RMX86/LIB/EPIFL.LIB, /RMX86/LIB/IPIFL.LIB, &
/$MX86/LIB/HPIFL.LIB, /RMX86/LIB/RPIFL.LIB, &
/RMX86/LIB/LPIFL.LIB TO MAIN &
MP(+00000H,+2ffffH) BIND
- :LANG:LINK86 DESIREP.OBJ, /RMX86/LIB/LARGE.LIB, &
 /RMX86/LIB/EPIFL.LIB, /RMX86/LIB/IPIFL.LIB, &
 /RMX86/LIB/HPIFL.LIB, /RMX86/LIB/RPIFL.LIB, &
 /RMX86/LIB/LPIFL.LIB TO DESIREP BIND
```

A.3. Execution of the DESIRE-P

There are two ways to run the DESIRE-P in the INTEL version. We may type either the command MAIN or DESIREP. The difference between these two commands is the methods of handling the error recovery. The command MAIN uses the direct jump error recovery while the command DESIREP uses the multiprocessing error recovery. Both of

these methods are discussed in Sec. 4.4. The method for invoking the commands are:

- MAIN or - DESIREP

LIST OF REFERENCES

Anderson, J. P.
1961 A Computer for Direct Execution of Algorithmic Languages, Proceedings EFCC, pp. 31-62

Digital Equipment Corp.
VAX-11 Pascal Language Reference Manual
VAX-11 Macro Language Reference Manual
VAX-11 Fortran Language Reference Manual

Goltz, John R.
1970 DARE-I, An On-Line Digital Simulation System, Ph.D. Dissertation, University of Arizona

INTEL Corp.
Pascal-86 User's Guide
8087 Support Library Reference Manual
iAPX 86, 88 Family Utilities User's Guide
Getting Started with the iRMX 86 System
iRMX 86 Human Interface Reference Manual
Run-Time Support Manual for iAPX 86, 88 Application
iRMX 86 Nucleus Reference Manual
iRMX 86 Extended I/O System Reference Manual
iRMX 86 Programming Techniques
iRMX 86 UDI Reference Manual
iRMX 86 System Configuration User's Guide

Korn. G. A.
1979 Microdare: A Fast, Direct-Executing High-Level Language System for Small Computers, IEEE Transactions on Computer, October, pp. 61-71

1982 EARLY DESIRE, A Floating-Point Equation Language Simulation System, Simulation, May, pp. 151-159.

1982 EARLY DESIRE, Mathematics and Computers in Simulation, Vol. 24, pp. 30-36.

1985 DESCTOP Implementation Notes

Martinez, Ralph
1976 A Semi-Portable Simulation System Using Both Fixed and Floating-Point Derivation Blocks, Ph.D. Dissertation, University of Arizona

Martinez, Ralph
1979 Direct Execution Architecture in Continuous System Simulation, 1979 Summer Simulation Conference, 16-18 July, Toronto, Canada

Mcleod, John
1966 PHYSBE: A Physiological Simulation Benchmark Experiment, Simulation, Vol. 7, No. 6, December, p. 324

Nissen, S. M. and Wallach, S. J.
1973 An APL Microprogramming Structure, Proceedings Symposium on High-Level-Language Computer Architecture, University of Maryland, pp. 43-51

Rice, R. and Smith, W. R.
1971 SYMBOL--A Major Departure from Classic Software Dominated Von Neumann Computing Systems, Proceedings SJCC, pp. 575-587

Vakilzadian, Hamid
1985 Design of Portable Direct Executing Languages for Interactive Simulation, Ph.D. Dissertation, University of Arizona

Zaks, R.
1971 Microprogrammed APL, Proceedings IEE International Computing Society Conference, pp. 193-194