NIXDORF
COMPUTER

TARGON® /35

UNIX Operating System Release Description 1.0

Ausgabe 1. 1. 1986 Bestell-Nr. 10128.00.2.93

Separate and insert in
pocket on spine of book.

Systems Literature

TARGON® /35

In order to ensure your inclusion in the distribution list for your revision service, please send us this card.
Your Nixdorf branch is responsible for sending modifications.

Please send any modifications supplements to this document to the following address.

The Nixdorf branch responsible
(must be stated):

Name: ____________________

In company: ____________________

Or NCAG dept.: ____________________

Reference number:
10128.00.2.93

Date of issue of the last revision as per
modification sheet (must be stated):

____

Postcard

Nixdorf Computer AG
Abt. ZSI
Fürstenallee 7

D-4790 Paderborn
West-Germany

UNIX Operating System Overview 1

Procedure Call Mechanism 2

Language Support 3

UNIX Operating System Commands and Facilities 4

Appendix Index A

Modifications Sheet

Modifications sheet

This sheet lists all modifications made to this module since the appearance of the first edition. It should be replaced by the new sheet provided whenever further modifications are announced.

First edition	01.01.86	Rel. 1

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Errors/Suggestions for Improvement

Errors/suggestions for improvement

If you have noticed any errors while using this part of the systems literature or if you have any suggestions for the improvement of the module, please send your written comments to the following address:

Nixdorf Computer AG
Abt. ZSI
Fürstenallee 7

D-4790 Paderborn

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Contents

1	**UNIX Operating System Overview**	**1-1**
1.1	Features	1-1
1.2	Implementation of UNIX System V and 4.2BSD	1-3
1.3	Effect on Machine-Dependent Programs	1-4
1.4	Commands and Facilities	1-4
2	**Procedure Call Mechanism**	**2-1**
2.1	Parameter-Passing Mechanism	2-1
2.2	Assigning Parameters and Return Values to Registers	2-4
2.3	Floating Point Mechanism	2-6
2.3.1	Using Variables and Assignment Statements	2-6
3	**Language Support**	**3-1**
3.1	C Support	3-1
3.2	Implementation of Language Support Features	3-1
3.2.1	Format of a.out and ar files	3-2
3.3	Effects of TARGON /35 Architecture on Application Programs	3-3
3.3.1	Performing Shift Operations in C	3-3
3.3.2	Passing Parameters	3-3
3.3.3	Implementing Varargs Macros in the Varargs.h Header	3-7
3.3.4	Addressing Bytes in a Specific Order	3-8
3.3.5	Evaluating Parameters in a Specific Order	3-9
3.3.6	Setting Local Auto Variables	3-13
3.3.7	Storing Data Structures	3-13
4	**UNIX Operating System Commands and Facilities**	**4-1**
Appendix	Index	A-1

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Preface

This manual discusses the features and facilities included in the Nixdorf UNIX™ (UNIX is a trademark of Bell Laboratories) Operating System, release 1.0. Also included is a discussion of the TARGON /35 parameter-passing mechanism and its effect on the execution of programs written in C.

This document is written for an application or systems programmer experienced in using the UNIX Operating System and in understanding UNIX Operating System concepts.

This document contains four chapters:

- **Chapter 1 – UNIX Operating System Overview.** This chapter lists the major features of the UNIX Operating System.
- **Chapter 2 – Procedure Call Mechanism.** This chapter discusses three aspects of the TARGON /35 System architecture: the parameter-passing mechanism, the registers to which the parameters and return values are assigned, and the floating point mechanism.
- **Chapter 3 – Language Support.** This chapter describes specific ways in which the C language compiler of the UNIX Operating System differs from that of UNIX System V or 4.2BSD.
- **Chapter 4 – UNIX Operating System Commands and Facilities.** This chapter briefly describes the commands and facilities that are available with the UNIX Operating System.

1 UNIX Operating System Overview

The UNIX Operating System on the TARGON /35 System is a general purpose, multi-user system supporting large applications and distributed processing. The Operating System combines AT&T Bell Laboratories UNIX System V with the University of California, Berkeley 4.2BSD enhancements to provide the user with a comprehensive system based on the UNIX Operating System.

For information on how to install the Nixdorf UNIX Operating System, see the Nixdorf manual "System Administrator's Guide".

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

1.1 Features

The UNIX Operating System kernel incorporates all of the performance enhancements of 4.2BSD, yet remains fully compatible with System V. In this way, the UNIX Operating System offers the 4.2BSD features of virtual memory and disk I/O performance needed to run the UNIX Operating System effectively on large machines, while at the same time providing the System V tools needed to run the UNIX Operating System within commercial environments. The UNIX Operating System also provides a Common Language Environment (CLE) which allows users to develop and execute programs written in C, FORTRAN, and Pascal with maximum efficiency on the TARGON /35. Nixdorf enhancements to the UNIX Operating System, specifically designed to improve performance on the TARGON /35, are included.

The major features of the Nixdorf UNIX Operating System are:

- **Virtual Memory**. Supports demand paging providing up to 4 Gbytes of directly addressable space per process.
- **C Language.** Supports multple data types and program utilities.
- **Shell Command Language.** Provides user interface to the UNIX Operating System.
- **C Shell Command Language**. Provides additional shell capabilities of job control and historical references.

UNIX Operating System Overview

- **Programmer's Workbench.** Provides Source Code Control System (SCCS), compiler development tools, "desk calculator" utilities, high-level language macro-processors, and text pattern manipulation routines.
- **Document Preparation.** A line-oriented editor (ed), text formatting and typesetting utilities (nroff and troff), and memorandum macros (MM).
- **Full-screen Interactive Editor.** Supports a sophisticated, easy-to-use full-screen editor (vi).
- **System Activity Package.** Gathers system activity data and generates reports.
- **I/O Subsystem Auto-configuration.** Automatically configures I/O channels, subchannels, disk drives, and ITPs.
- **Termcap.** Data base and routines allowing almost any asynchronous terminal to be used with the TARGON /35.

The UNIX Operating System provides various facilities for the development and execution of C. These facilities currently provide symbolic debugging, code generation, and runtime execution.

Another feature included in the UNIX Operating System is the uucp utility, which permits communication between two or more UNIX systems. To use the uucp utility successfully, see the Nixdorf manual "UUCP – UNIX to UNIX copy".

1.2 Implementation of UNIX System V and 4.2BSD

The TARGON /35 supports both AT&T Bell Laboratories UNIX System V and the University of California, Berkeley 4.2BSD enhancements to the UNIX Operating System. This unique, dual-port system allows a user to use the features of either System V or 4.2BSD, or a combination of both. To provide this capability to the user, Nixdorf has implemented an interface within the UNIX Operating System that allows users to execute commands and develop programs in more than one environment, or "universe."

Currently, the UNIX Operating System supports two universes – System V (att) and 4.2BSD (ucb). These two universes coexist within the same file structure and share a common 4.2BSD kernel. At any time, the user can operate in either the **att** or the **ucb** universe. The **universe** command allows the user to set or change the UNIX Operating System environment in which he or she operates. Specific information on how to establish a universe is contained in the Nixdorf manual "System Administrator's Guide" and the procedures for using these universes are described in the UNIX Operating System online manual pages.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

1.3 Effect on Machine-Dependent Programs

Nixdorf's implementation of C, combined with the unique TARGON /35 architecture, may require programmers to modify their machine-dependent programs in specific ways to ensure proper program execution. The specific aspects of the UNIX Operating System that may affect machine-dependent programs are:

- file formats
- block size
- performance of shift operations
- passing of arguments
- implementation of varargs macros in the varargs.h header
- performing of I/O to or from local variables or parameters
- order in which bytes are addressed
- order of parameter evaluation
- setting of local auto variables

1.4 Commands and Facilities

The UNIX Operating System provides complete sets of both System V and 4.2BSD utilities, system calls, libraries, and files. Some of these facilities have been enhanced and new facilities have been created to enable users to develop and execute programs quickly and efficiently on the TARGON /35 System. These commands and facilities are discussed in Chapter "UNIX Operating System Commands and Facilities" and available as online manual pages.

2 Procedure Call Mechanism

The current parameter-passing mechanism on the TARGON /35 architecture differs from that of other machines by providing fast procedure calls and fast access to procedure parameters and local variables. For those programming in C, this difference causes machine-dependent programs to behave differently on the Nixdorf machine than on other machines, specifically because of the way that data is stored in parameter and local variable registers. To understand the effect of UNIX Operating System calling sequences on programs, it is necessary to discuss the way in which the TARGON /35 passes parameters and assigns them to registers.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

2.1 Parameter-Passing Mechanism

The architecture of the TARGON /35 includes separate stacks, known as control and data stacks. These stacks minimize memory references by providing temporary storage for procedure parameters, local variables, and local data structures. The processor uses one control stack and two data stacks for each process in the system. (One data stack is used in user mode and a second data stack in kernel mode.)

A control stack consists of a number of 32-word control stack frames, each containing parameter registers (16 words) and local variable registers (16 words).

The processor uses the control stack frame to:

- store procedure parameters, temporary values, and procedure linkage information
- store frequently-used local variables

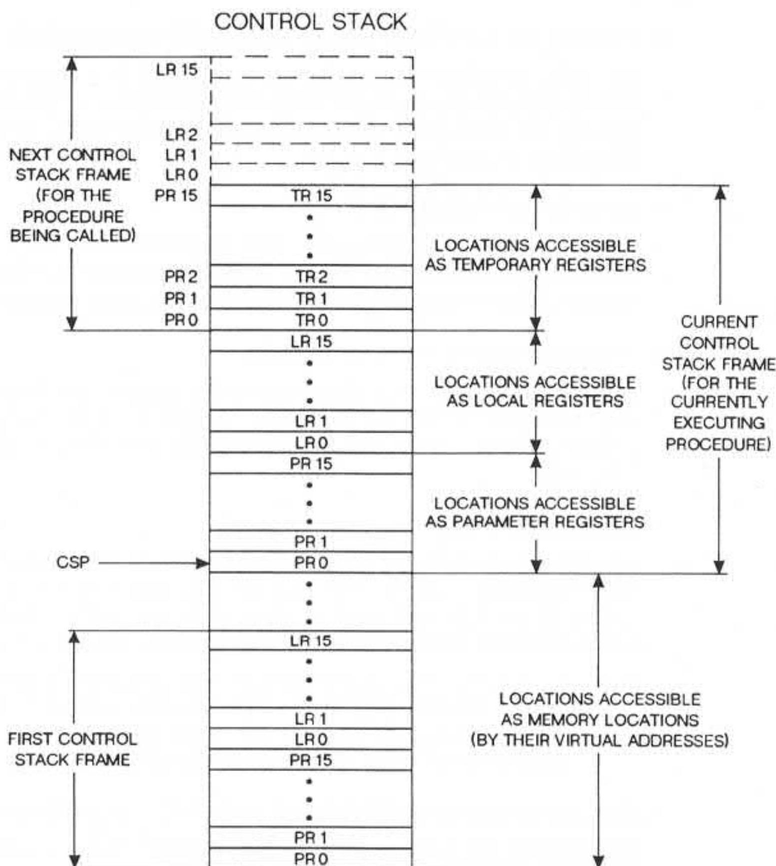
The current frame consists of 48 words, since it also includes the first 16 words of the next frame. These 16 words are accessed as temporary registers. A control stack frame is allocated for each procedure call and deallocated for each procedure return.

Procedure Call Mechanism

A data stack is used as an extension of the corresponding control stack. All parameters and local data which are not in the control stack (because of the fixed-size control stack frame) are stored in the data stack.

Procedure Call Mechanism

Figure 2-1: TARGON /35 Control Stack



© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

2.2 Assigning Parameters and Return Values to Registers

Parameters and return values are assigned, in ascending order, to registers tr0/pr0 through tr11/pr11 in the following ways:

- int, long, unsigned int, unsigned long, and float parameters

  Int, long, unsigned int, unsigned long, and float parameters are directly moved into available temporary/parameter registers. If registers are not available, these parameters are moved into word-aligned data stack locations.

- char and short parameters

  Char and short parameters are sign-extended into available temporary/parameter registers. If registers are not available, these parameters are moved into word-aligned data stack locations.

- unsigned char and short parameters

  Unsigned char and short parameters are moved and zero-extended into available temporary/parameter registers. If registers are not available, these parameters are moved into word-aligned data stack locations.

- double parameters

  Double parameters are moved into adjacent parameter registers when available. (These registers are not required to be even/odd pairs.) If an adjacent register pair is not available, these double parameters are moved into word-aligned data stack locations.

  Note: If tr11/pr11 (a temporary/parameter register) is available but cannot be allocated to the next double parameter, the double parameter is pushed onto the data stack and the tr11/pr11 register is not used for passing parameters.

- structure and union parameters (struct/union parameters)

  Struct/union parameters are always pushed onto the data stack, regardless of whether or not they will fit into either a single register or a register pair.

Procedure Call Mechanism

- array parameters

  Array parameters are treated as pointers to the corresponding data type with a value corresponding to the array base.

- pointer parameters

  All pointer parameters are directly moved into available temporary/parameter registers. If registers are not available, these parameters are moved into word-aligned data stack locations.

- structure returns

  Structure returns pass a pointer to a static region containing the return structure. This pointer is in register tr0/pr0.

- scalar and float returns

  Scalar and float returns are assigned to register tr0/pr0. Double returns are assigned to register (tr0-tr1)/(pr0-pr1).

Besides the registers used for storing the parameters noted above, additional temporary registers are reserved for specific functions. These registers and their functions are:

TR 12 Reserved for use with the SSL instruction, that is, for static block structure references.

TR 13 Reserved for computing the address of the called subroutine, such as for calling a register variable.

TR 14 & 15 Cannot be used since they can be destroyed at any time by an interrupt.

Note: Subroutines that "know" that they have data stack parameters must perform an ADSF instruction upon entry.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

2.3 Floating Point Mechanism

The UNIX Operating System includes a floating point mechanism based on the IEEE floating point standard.

To ensure proper program execution on the TARGON /35, the user must use appropriate variables and explicit assignment statements. The required method for printing hexadecimal or binary versions of a floating point variable is explained below.

2.3.1 Using Variables and Assignment Statements

On some UNIX-based machines, a non-standard floating point representation is used. A feature of these non-standard representations is that for both single and double precision numbers, the same number of bits is used to represent the exponent. Thus, if the user has a double precision variable and wants to print the single precision value, the user must print out only the high-order word.

Because the IEEE floating point standard uses different representations for single and double precision values, the user must be careful to force conversions through appropriate variables and explicit assignment statements. For example, to extract a hexadecimal representation of a floating point variable, the following is incorrect:

```
printf("%x", a_float)
```

This statement is incorrect because the **float** argument to **printf** is first promoted to a double precision value.

Thus, the hexadecimal value, which would normally be printed, is the high-order portion of a double precision variable and not a floating point variable. Figure 2-2 shows code that will correctly print the binary representation of a **float** number.

Procedure Call Mechanism

Figure 2-2: Printing the Binary Representation of an Float Number

```
float x;
union {
      float a_float;
      unsigned an_int;
} overlay

overlay.a_float = x              /* avoid a type conversion */
printf("%x", overlay.an_int);    /* avoid a type conversion */
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

3 Language Support

Nixdorf's implementation of C allows users to convert existing application programs from their UNIX-based operating systems to the Nixdorf UNIX Operating System. In most cases, this conversion will be transparent to the user. This chapter describes the implementation of Nixdorf's language support and the effect of architecture on machine-dependent programs written in C.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

3.1 C Support

The UNIX Operating System provides a standardized environment for the development and execution of programs written in C. The UNIX Operating System currently provides facilities for source level (symbolic) debugging, code generation, and runtime execution supporting Nixdorf's implementation of C.

- **Symbolic debugging.** A common interface is used for symbolic debugging at the source level, providing a consistent set of debugging tools for the C language.
- **Code generation.** A common code generator ensures conformity with the Nixdorf procedure call standards and optimum use of the performance of the TARGON /35 register-intensive instruction set.
- **Runtime execution.** Uniform procedure calls allow calls to a common library of runtime routines.

3.2 Implementation of Language Support Features

The UNIX Operating System implements the a.out and ar files differently than do other systems that are based on the UNIX Operating System. The following paragraphs describe this implementation.

Language Support

3.2.1 Format of a.out and ar Files

Because of the way that the UNIX Operating System uses the a.out and ar files, the format of the a.out and ar files differs from that in UNIX System V. The format of these files in the UNIX Operating System is based on the format of the a.out and ar files in the 4.2BSD enhancements to the UNIX Operating System.

The format of the a.out file is:

```
long      a_magic;    /* magic number */
unsigned  a_text;     /* size of text segment */
unsigned  a_data;     /* size of data segment */
unsigned  a_bss;      /* size of bss segment */
unsigned  a_syms;     /* size of symbol table */
unsigned  a_entry;    /* entry point of program */
unsigned  a_trsize;   /* size of text relocation info */
unsigned  a_drsize;   /* size of data relocation */
```

The format of the ar file is:

```
char  ar_name[16];   /* archive member name */
char  ar_date[12];   /* archive date */
char  ar_uid[6];     /* archive user identification */
char  ar_gid[6];     /* archive group identification */
char  ar_mode[8];    /* archive member mode */
char  ar_size[10];   /* archive member size */
char  ar_fmag[2];    /*                     */
```

3.3 Effects of TARGON /35 Architecture on Application Programs

The TARGON /35 architecture may affect machine-dependent application programs in certain ways. The following paragraphs describe how programs may be affected.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

3.3.1 Performing Shift Operations in C

The TARGON /35 machine may perform shift operations differently than other machines. On the Nixdorf machine, the shift operators << and >> group from left to right. Each of these operators performs the usual arithmetic conversions on its operands; each operand must be integral. The right operand is then converted to **int** and the type of left operand indicates the type of the result. If the right operand is negative, or greater than or equal to the length of the object in bits, the result is undefined.

For example:

```
shift-expression:
      expression << expression
      expression >> expression
```

The value of E1<<E2 is E1, interpreted as a bit pattern, and is shifted E2 bits to the left. Vacated bits are 0-filled. The value of E1>>E2 is E1, shifted E2 bits to the right. If E1 is unsigned, the right shift will be logical, or 0-filled. Otherwise, the right shift will be arithmetic, or filled by a copy of the sign bit.

3.3.2 Passing Parameters

Some machine-dependent programs may require modification because of the way in which UNIX Operating System calling sequences affect programs written in C. The specific characteristics of the UNIX Operating System calling sequences that may necessitate program modification are described below.

- Parameters cannot be implicitly passed via register variables.

  If any compilers on non-Nixdorf machines have allowed parameters to be passed via register variables, this was due to accident and not design.

- Programs cannot contain a variable number of parameters.

  A C program cannot work through an argument list and determine the point at which it must leave the parameter registers and enter the data stack. However, to accommodate certain forms of access to "regular" variable arguments, the varargs macros in /usr/include/varargs.h may be used.

- Type conflicts may exist between calling and called subroutines.

  The command **lint** should be used whenever possible to check for these conflicts.

- Struct/union parameters are pushed onto the data stack, whether or not they will fit into a single register or a register pair.

- Figure 3-1 shows code that cannot be executed on the TARGON /35 machine because of the way the TARGON /35 treats struct/union parameters and because of the typing rules of the C language.

  Figure 3-2 shows an example of the same mechanics in a non-subroutine call environment. The typing rules of C make this statement explicitly illegal also.

  The recommended sequence is shown in Figure 3-3.

Figure 3-1: Type Conflicts in Parameters

```
union x {
        int a;
        char b[4];
}
main ()
{
        union x abc;

        subl (abc);          /* WRONG! */
        subl ((int) abc);    /* WRONG! */
}
subl (arg)
int arg;
{
}
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Figure 3-2: Type Conflicts in Assignments

```
union x {
        int a;
        char b[4];
}
main ()
{
        union x abc;
        int xyz;

        xyz = abc;          /* Illegal statement */
        xyz = (int) abc;    /* Illegal statement */
}
```

Figure 3-3: Fixing Type Conflicts in Parameters

```
union x {
        int a;
        char b[4];
}
main ()
{
        union x abc;

        {int i; i=abc.a subl (i);}
        subl (abc.a);
}
subl (arg)
int arg;
{
}
```

- Parameters are pushed on to the data stack from right to left.

  Arguments that are assigned to tr/pr registers, however, are assigned from left to right; that is, the left-most parameter is placed in register tr0/pr0.

- If the called subroutine issues an ADSF instruction, then (cfp), after executing ADSF, is the first word of the last parameter pushed on to the data stack.

- If the calling procedure places any data on the data stack to place parameters for a subroutine call, the stack must be popped by the procedure by an appropriate amount immediately after the call.

- The parameter to RETD must always be 0 since the caller is responsible for cleaning up the stack.

  If the called program performs an ADSF either to accommodate parameters that it expects on the data stack or to allocate temporaries, the program must perform an RETD instruction rather than an RET instruction.

- If there is a type mismatch between the calling and called programs where the actual parameter is struct/union and a single word quantity is expected, all subsequent parameters will be incorrect within the parameter registers. All previous and subsequent data stack parameters will also be incorrect.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

3.3.3 Implementing Varargs Macros in the Varargs.h Header

Another way that the TARGON /35 architecture affects machine-dependent programs is in its treatment of variable length argument lists. If the user abides by the conventions of varargs, variable length argument lists can always be accessed. Also, varargs.h is a standard system header in the UNIX Operating System, and all of its capabilities have been implemented in the UNIX Operating System.

There are several ways to handle functions that permit a variable number of arguments. If the maximum number of arguments is known, the user can simply define the function with that number of arguments.

For example:

```
debug("at main %d %x", argo, argv);
debug("read(%d, %d, %d) gives %d", f, buf, n, ret);
...

debug(fmt, a1, a2, a3, a4, a5)    /* 5 arguments max */
{
     printf(fmt, a1, a2, a3, a4, a5)
}
```

This solution to the variable number of arguments problem is simple, easy to understand, and reliable. However, in the case where the maximum number of arguments is not known or is inconveniently large, the user should use the standard UNIX Operating System varargs.h macros.

Language Support

Not all arguments to a function reside on the user data stack as an array of words. However, despite the problems posed by the control and data stack, the Nixdorf implementation of the varargs macros is functionally equivalent to that of other non-Nixdorf machines based on the UNIX Operating System.

3.3.4 Addressing Bytes in a Specific Order

The order in which bytes are addressed on the Nixdorf machine may differ from that of other machines. On the Nixdorf machine, bytes are numbered from left to right. Thus, byte 0 is the most significant byte. Because some non-Nixdorf architectures number their bytes differently from the TARGON /35 architecture, code sequences sometimes exist which take advantage of this difference in numbering. Upon examination, these code sequences are often incorrect, since they usually involve an illegal type conversion.

The following code sequence is incorrect:

```
{ int c;
  read(0, &c, 1);    /* read a single character into c */
  printf("%c\n", c);
}
```

A correct sequence is either:

```
{ char c;
  read(0, &c, 1);
  printf("%c\n", c);
}
```

Or:

```
{ int c;
  read(0, &c, 1);
  printf("%c\n", ((char *) &c) [0]);
}
```

3.3.5 Evaluating Parameters in a Specific Order

The TARGON /35 architecture causes parameters to be evaluated in a different order than on a non-Nixdorf machine based on the UNIX Operating System. The program in Figure 3-4 illustrates the order in which parameters are evaluated. Figure 3-5 illustrates the program's behavior on a non-Nixdorf machine, while Figure 3-6 illustrates the program's current behavior on the TARGON /35 System. Since the order of parameter evaluation is implementation dependent, the output of this program is subject to change with future revisions of the compiler.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Language Support

Figure 3-4: Sample Program Illustrating Parameter Evaluation

```
main ()
{
      int i;

      i = 0;
      abc(i++,i++,i++,i++,i++,i++,i++,i++,i++,i++,i++,i++,i++,i++,i++);
      printf("After abc i = %d0,i);
      exit(0);
}
abc(a,b,c,d,e,f,g,h,i,j,k,l,m,n,o)
int a,b,c,d,e,f,g,h,i,j,k,l,m,n,o;
{
#define xx(yy) printf("yy = %d0,yy);
      xx(a);
      xx(b);
      xx(c);
      xx(d);
      xx(e);
      xx(f);
      xx(g);
      xx(h);
      xx(i);
      xx(j);
      xx(k);
      xx(l);
      xx(m);
      xx(n);
      xx(o);
}
```

Figure 3-5: Output Program from a Non-Nixdorf Compiler

```
a  = 14
b  = 13
c  = 12
d  = 11
e  = 10
f  = 9
g  = 8
h  = 7
i  = 6
j  = 5
k  = 4
l  = 3
m  = 2
n  = 1
o  = 0
After abc i = 15
```

Figure 3-6: Output from the Current TARGON /35 Compiler

```
a  = 4
b  = 5
c  = 6
d  = 7
e  = 8
f  = 9
g  = 10
h  = 11
i  = 12
j  = 13
k  = 14
l  = 3
m  = 2
n  = 1
o  = 0
After abc i = 15
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

The order in which function parameters are evaluated is not specified. Thus, the following statement produces different results on different machines:

```
printf("%d %d\n", ++n, power(2, n));    /* WRONG */
```

The result depends on whether n is incremented before power is called. The following statement solves the problem:

```
++n;
printf("%d %d\n", n, power(2, n));
```

Function calls, nested arguments, and increment and decrement operators cause "side effects". Side effects result when some variable is changed as a byproduct of the evaluation of the expression. In an expression that involves side effects, there may be dependencies on the order in which variables that take part in the expression are stored. The following is an undesirable statement:

```
a[i] = i++;
```

The compiler can determine in various ways whether the subscript is the new or the old value of i and, depending on its interpretation, produce different answers. Because the best order strongly depends on machine architecture, the compiler determines the time of the assignment to actual variables.

Thus, code written in any language should not depend on the order in which parameters are evaluated. The C verifier **lint** will, however, detect most dependencies on the order of evaluation.

Language Support

3.3.6 Setting Local Auto Variables

The setting of local auto variables to any value should not be expected. (A non-Nixdorf machine may accidentally get a zero.)

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

3.3.7 Storing Data Structures

The storage layout of data structures differs from that of other machines and may change in future releases of the product.

UNIX Operating System Commands and Facilities

4 UNIX Operating System Commands and Facilities

The commands and facilities available with the UNIX Operating System fall into major categories with subsections as follows:

1. **Commands and Application Programs.** These programs are invoked directly by the user or by command language procedures. Classifications of these commands are:

   1. General-purpose commands
   1C. Communications commands
   1G. Graphics commands

2. **System Calls.** These programs function as entries into the UNIX Operating System kernel, which includes the C language interface.

3. **Subroutines.** The binary versions of these subroutines reside in various libraries within the system file structure. Classifications of subroutines are:

   3C. C and Assembler library routines
   3M. Mathematical library routines
   3S. Standard I/O library routines
   3X. Miscellaneous routines

4. **Special Files.** These files are used by the UNIX Operating System Administrator.

5. **File Formats.** This section describes the format of certain kinds of files.

6. **Miscellaneous Facilities.** These facilities consist of character sets and macro packages.

7. **System Maintenance Commands.** These Commands are used by the UNIX Operating System Administrator.

Note: Manual pages for all UNIX-supported commands and facilities are available online as follows:

$ **man** name

where name is the name of the desired command or facility.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Index

Appendix Index

a.out file 3.2.1
ar file 3.2.1
array parameters 2.2
att 1.2

C 3.1
char and short parameters 2.2
CLE 1.1
code generator 3.1
Common Language Environment 1.1
control stack 2.1
control stack frame 2.1

data stack 2.1
dual-port system 1.2

floating point mechanism 2.3

IEEE floating point standard 2.3, 2.3.1

pointer parameters 2.2

scalar and float returns 2.2
shift operations 3.3.1
struct/union parameters 2.2
structure returns 2.2
symbolic debugging 3.1

temporary/parameter registers 2.2

ucb 1.2
universe 1.2
UNIX Operating System kernel 1.1
unsigned char and short parameters 2.2
uucp 1.1

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.