386 SOFTWARE TOOLS

PL/M 386 Software Package

- **Systems Programming Language for the Protected Virtual Address Mode 386**
- **Upward Compatible with PL/M 286, PL/M 86, and PL/M 80 Assuring Software Portability**

386 Relocation, Linkage and Library Tools

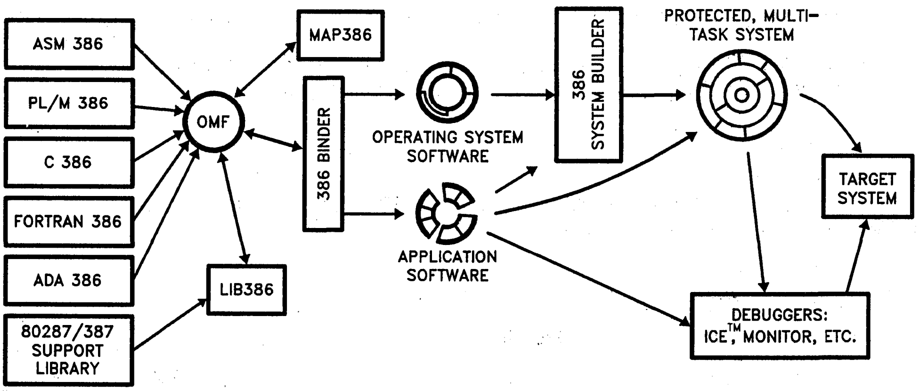
- **Provides System Development Capability for High-Performance 386 Applications**
- **Allows Creation of Multi-User Virtual Memory, and Memory-Protected Systems**

C 386

- **Implements Full C Language and New Extensions**
- **Produces High Density Code Rivaling Assembler**
- **Supports Intel Object Module Format (OMF)**

ASM 386

- **Instruction Set and Assembler Mnemonics Are Upward Compatible with ASM 286 and ASM 86**
- **Type-Checking at Assembly Time Helps Reduce Errors at Run-Time**



261637-1

Figure 1. Development Environment Tools for the 386

386 Software tools are available on industry standard hosts, including VAX/VMS, PC-DOS, and XENIX*

November 1986
Order Number: 231637-002

ASM 386

- **Instruction Set and Assembler Mnemonics Are Upward Compatible with ASM 286 and ASM 86**
- **Powerful and Flexible Text Macro Facility**
- **Type-Checking at Assembly Time Helps Reduce Errors at Run-Time**
- **Structures and Records Provide Powerful Data Representation**
- **"High-Level" Assembler Mnemonics Simplify the Language**
- **Supports Full Instruction Set of the 386, Including Memory Protection and Numerics**
- **Supports 286 Addressing Modes**

ASM 386 is the "high-level" macro assembler for the 386 assembly language. ASM 386 translates symbolic assembly language mnemonics into relocatable object code. The assembler mnemonics are a superset of ASM 286/86/88 mnemonics; new ones have also been added to support the new 386 instructions. The segmentation directives have been greatly simplified.

The 386 assembly language includes approximately 275 instruction mnemonics. From these few mnemonics the assembler can generate over 40,000 distinct machine instructions. Therefore, the software development task is simplified, as the programmer need know only 275 mnemonics to generate all possible machine instructions. ASM 386 will generate the shortest machine instruction possible (given explicit information as to the characteristics of any forward referenced symbols).

The powerful macro facility in ASM 386 saves development and maintenance time by coding common program sequences only once. A macro substitution is made each time the sequence is to be used. This facility also allows for conditional assembly of certain program sequences.

ASM 386 offers many features normally found only in high-level languages. The assembly language is strongly typed, which means it performs extensive checks on the usage of variables and labels. This means that many programming errors will be detected when the program is assembled, long before it is being debugged.

ASM 386 object modules conform to a thorough, well-defined format used by all 386 high-level languages and utilities. This means it is easy to call (and be called from) HLL object modules.

SUPPORT

Hotline Telephone Support, Software Performance Report (SPR), Software Update, Technical Reports, and Monthly Technical Newsletters are available.

ORDERING INFORMATION

Part Number	Description	Operating Environment
X286ASM386	386 Assembler	286/310 XENIX* System
D86ASM386	386 Assembler	PC-DOS 3.0 or greater

Documentation Package

ASM 386 Assembly Language Reference Manual
ASM 386 Macro Assembler Operating Instructions for XENIX* 286 Systems
ASM 386 Pocket Reference for XENIX 286 Systems

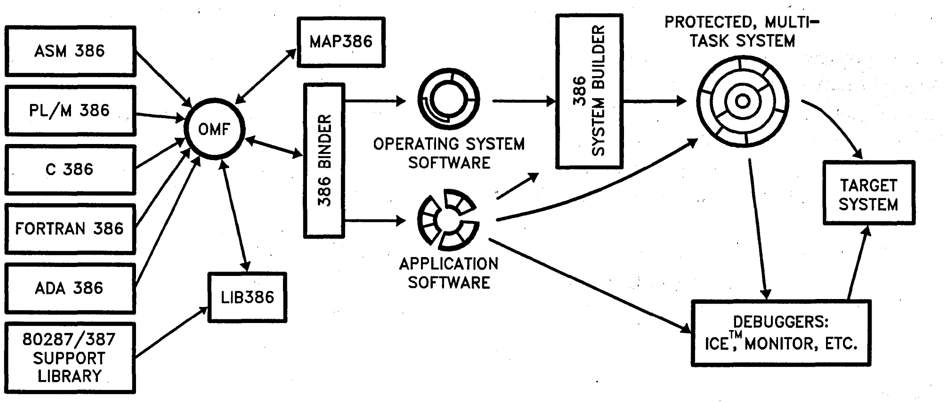
*XENIX™ is a trademark of Microsoft.

386 RELOCATION, LINKAGE AND LIBRARY TOOLS

- **System Development Capability for High-Performance 386 Applications**
- **Allows creation of Multi-User, Virtual Memory, and Memory-Protected Systems**
- **System Utilities for Program Linkage and System Building**
- **Package Supports Program Development with ASM 386, PL/M 386, C 386, Ada 386 and FORTRAN 386.**

The 80386 is a 32-bit microprocessor system with 32-bit addressing, integrated memory protection, and instruction pipelining for high performance. The 386 Relocation, Linkage, and Library Tools are a cohesive set of software design aids for programming the 386 microprocessor system. The package enables system programmers to design protected, multi-user and multi-tasking operating system software, and enables application programmers to develop tasks to run on a protected operating system.

The 386 Relocation, Linkage and Library tools include a program binder (for linking separately compiled modules together), a system builder (for configuring protected multiple-task systems), a cross reference mapper, a program librarian, and the 287/387 support library.



261637-2

Figure 1. Development Environment Tools for the 386

386 SYSTEM BUILDER

- **Supports Complete Creation of Protected, Multi-task Systems**
- **Resolves PUBLIC/EXTERNAL Definitions (between protection levels)**
- **Supports Memory Protection by Building System Tables, Initializing Tasks, and Assigning Protection Rights to Segments**
- **Creates a Memory Image of a 386 System for Cold-start Execution**
- **Target System may be Boot-loadable, Programmed into ROM, or loaded from Mass-store.**
- **Generates Print File with Command Listing and System Map**

BLD 386 is the utility that lets system programmers configure multi-tasking, protected systems from an operating system and discrete tasks. The Builder generates a cold-start execution module, suitable for ROM-based or disk-based systems.

The Builder accepts input modules from 386 translators or the 386 Binder. It also accepts a "Build File" containing definitions and initial values for the 386 protection mechanism - descriptor tables, gates, segments, and tasks. BLD 386 generates a Loadable or bootloadable output module, as well as a print file with a detailed map of the memory-protected system.

Using the Builder command Language, system programmers may perform the following functions:

— Assign physical addresses to segments; also set segment access rights and limits.

— Create Call, Trap, and Interrupt "Gates" (entry-points) for inter-level program transfers.

— Make gates available to tasks; this is an easier way to define program interfaces than using interface libraries.

— Support Page tables for boot files.

— Create Global (GDT), Interrupt (IDT), and any Local (LDT) Descriptor Tables.

— Create Task State Segments and Task Gates for multi-task applications.

— Resolve inter-module and inter-level references, and perform type-checking.

— Automatically select required modules from libraries.

— Configure the memory image into partitions in the address space.

— Selectively generate an object file and various sections of the print file.

386 BINDER

- **Links Separately Compiled Program Modules Into an Executable Task**
- **Makes the 386 Protection Mechanism Invisible to Application Programmers**
- **Works with PL/M 386, C 386, FORTRAN 386 and ASM 386 Object Modules**
- **Performs Incremental Linking with Output of Binder and Builder**
- **Resolves PUBLIC/EXTERNAL Code and Data References, and Performs Intermodule Type-Checking**
- **Provides Print File Showing Segment Map, Errors and Warnings**
- **Assigns Virtual Addresses to Tasks in the 2^{32} Address Space**
- **Generates Linkable or Loadable Module for Debugging**

The Binder is the only utility an application programmer needs to develop and debug an individual task. Users of the Binder need not be concerned with the architecture of the target machine, making application program development for the 386 very simple.

BND 386 combines 386 object modules into executable tasks. In creating a task, the Binder resolves Public and External symbol references, combines segments, and performs address fix-ups on symbolic code and data.

The Binder takes object modules written in ASM 386, PL/M 386, C 386 and FORTRAN 386 and generates a loadable module (for execution or debugging), or a linkable module (to be re-input to the Binder later; this is called incremental binding). The binder accepts library modules as well, linking only those modules required to resolve external references. BND 386 generates a print file displaying a segment map and error messages.

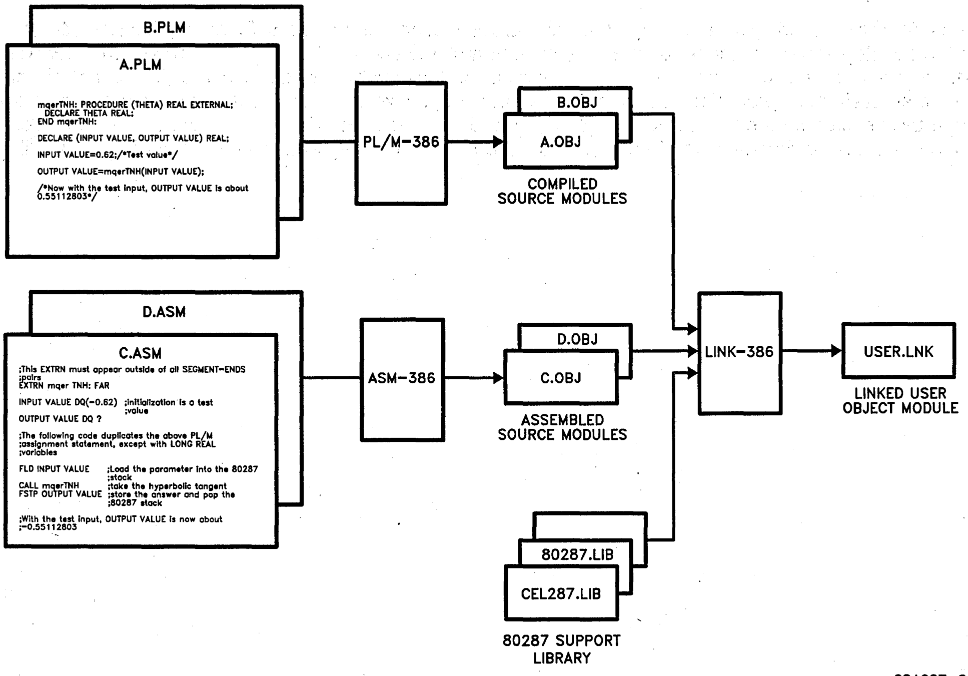
The Binder will be used by system programmers and application programmers. Since application programmers need to develop software independent of any system architecture, the 386 memory protection mechanism is "hidden" from users of the Binder. This allows application tasks to be fully debugged before becoming part of a protected system. (A protected system may be debugged, as well.) System protection features are specified later in the development cycle, using the 386 System Builder. It is possible to link operating system services required by a task using either the Binder or the Builder. This flexibility adds to the ease of the 386 utilities.

80287 SUPPORT LIBRARY

- **Library to support floating point arithmetic in C 386, PL/M 386, ADA 386, ASM 386, and FORTRAN 386**
- **Decimal conversion module supports binary-decimal conversions**
- **Supports proposed IEEE Floating Point Standard for high accuracy and software portability**
- **Common elementary function library provides trigonometric, logarithmic and other useful functions**
- **Error-handler module simplifies floating point error recovery**

The 80287 Support Library provides C 386, PL/M 386, ADA 386, ASM 386 and FORTRAN 386 users with numeric data processing capability. With the Library, it is easy for programs to do floating point arithmetic. Programs can bind in library modules to do trigonometric, logarithmic and other numeric functions, and the user is guaranteed accurate, reliable results for all appropriate inputs. Figure 1 below illustrates how the 80287 Support Library can be bound with PL/M 386 and ASM 386 user code to do this. The 80287 Support Library supports the proposed IEEE Floating Point Standard. Consequently, by using this Library, the user not only saves software development time, but is guaranteed that the numeric software meets industry standards and is portable—the software investment is maintained.

The 80287 Support Library consists of the common elementary function library (CEL287.LIB), the decimal conversion library (DC287.LIB), the error handler module (EH287.LIB) and interface libraries (80287.LIB), (NUL287.LIB).



```
mqerTNH: PROCEDURE (THETA) REAL EXTERNAL;
  DECLARE THETA REAL;
END mqerTNH:

DECLARE (INPUT VALUE, OUTPUT VALUE) REAL;

INPUT VALUE=0.62;/*Test value*/

OUTPUT VALUE=mqerTNH(INPUT VALUE);

/*Now with the test input, OUTPUT VALUE is about
0.55112803*/
```

```
;This EXTRN must appear outside of all SEGMENT-ENDS
;pairs
EXTRN mqer TNH: FAR

INPUT VALUE DQ(-0.62)  ;initialization is a test
                       ;value
OUTPUT VALUE DQ ?

;The following code duplicates the above PL/M
;assignment statement, except with LONG REAL
;variables

FLD INPUT VALUE      ;Load the parameter into the 80287
                     ;stack
CALL mqerTNH         ;take the hyperbolic tangent
FSTP OUTPUT VALUE    ;store the answer and pop the
                     ;80287 stack

;With the test input, OUTPUT VALUE is now about
;-0.55112803
```

231637-3

Figure 2. Use of 80287 Support Library with PL/M 386 and ASM 386.

386 MAPPER

- **Flexible Utility to Display Object File Information**
- **MAP 386 Selectively Purges Symbols from a Load Module**
- **Provides Inter-Module Cross-Referencing for Modules Written in All Languages**
- **Supports OS Information**
- **Mapper Allows Users to Display:**

Protection Information	**Debug Information**
SEGMENT TABLES	**MODULE NAMES**
GATE TABLES	**PROGRAM SYMBOLS**
PUBLIC ADDRESSES	**LINE NUMBERS**

The cross-reference map shows references between modules, simplifying debugging. The map also lists and controls all symbolic information in one easy-to-read place.

386 LIBRARIAN

- **Fast, Easy Management of 386 Object Module Libraries**
- **Only Required Modules Are Linked, When Using the Binder or Builder**
- **Librarian Allows Users to: Create Libraries, Add Modules, Replace Modules, Delete Modules, Copy Modules from Another Library, Save Library Module to Object File, Create Backup, Display Module Information (creation date, publics, segments)**

Program libraries improve management of program modules and reduce software administrative overhead. (386 Librarian provides efficient use of program libraries.)

SUPPORT:

Hotline Telephone Support, Software Performance Report (SPR), Software Updates, Technical Reports, and Monthly Technical Newsletters are available.

ORDERING INFORMATION:

Part Number	Description	Operating Environment
X286RLL386	386 Relocation, Linkage and Library Tools	286/310 XENIX* System
VVSRLL386	386 Relocation, Linkage, and Library Tools	VAX/VMS 4.3 and Later
D86RLL386	386 Relocation, Linkage, and Library Tools	PC-DOS 3.0 or Greater

Documentation Package

386 Utilities User's Guide for Xenix* 286 System
386 System Builder User's Guide for Xenix* 286 System
80287 Support Library Reference Manual

*XENIX is a trademark of Microsoft.

PL/M 386 SOFTWARE PACKAGE

- **Systems programming language for the protected virtual address mode 386**
- **Upward compatible with PL/M 286, PL/M 86 assuring software portability**
- **Enchanced to support design of protected, multi-user, multi-tasking, virtual memory operating system software**
- **Produces relocatable object code which is linkable to object modules generated by all other 386 language translators**
- **Advanced, structured system implementation language for algorithm development**
- **Supports Intel Object Module Format (OMF)**

PL/M 386 is a powerful, structured, high-level system implementation language for the development of system software for the protected virtual address mode 386. PL/M 386 has been enhanced to utilize 386 features—memory management and protection—for the implementation of multi-user, multi-tasking virtual memory operating systems.

PL/M 386 is upward compatible with PL/M 286, PL/M 86 and PL/M 80. Existing systems software can be re-compiled with PL/M 386 to execute in protected virtual address mode on the 80386.

PL/M 386 is the high-level alternative to assembly language programming on the 80386. For the majority of 386 system programs, PL/M 386 provides the features needed to access and to control efficiently the underlying 386 hardware and consequently it is the cost-effective approach to develop reliable, maintainable system software.

The PL/M 386 compiler has been designed to efficiently support all phases of software development. Features such as a built-in syntax checker, multiple levels of optimization, virtual symbol table and four models of program size and memory usage for efficient code generation provide the total program development support needed.

FEATURES

Major features of the Intel PL/M 386 compiler and programming language include:

Structured Programming

PL/M source code is developed in a series of modules, procedures, and blocks. Encouraging program modularity in this manner makes programs more readable, and easier to maintain and debug. The language becomes more flexible by clearly defining the scope of user variables (local to a private procedure, for example).

The use of modules and procedures to break down a large problems leads to productive software development. The PL/M 386 implementation of block structure allows the use of REENTRANT procedures, which are especially useful in system design.

Language Compatibility

PL/M 386 object modules are compatible with object modules generated by all other 386 translators. This means that PL/M programs may be linked to programs written in any other 386 languages.

Object modules are compatible with In-Circuit Emulators; DEBUG compiler control provides the In-Circuit Emulators with full symbolic debugging capabilities.

PL/M 386 language is upward compatible with PL/M 286, PL/M 86 and PL/M 80 so that application programs may be easily ported to run on the protected mode 80386.

Supports Fourteen Data Types

PL/M makes use of fourteen data types for various applications. These data types range from one to eight bytes and facilitate various arithmetic, logic, and addressing functions:

—BIT(n):	1 to 32 bit unsigned number
—BYTE:	8 bits unsigned number
—HWORD:	16 bits unsigned number
—WORD:	32 bits unsigned number
—DWORD:	64 bits unsigned number
—OFFSET:	32 bits memory address
—CHARINT:	8 bits signed number
—SHORTINT:	16 bits signed number
—INTEGER:	32 bits signed number
—LONGINT:	64 bits signed number
—REAL:	32 bits floating-point number
—SELECTOR:	16 bits segment name
—POINTER:	48 bits selector, offset
—LONGREAL:	64 bits floating-point number

Another powerful facility allows the use of BASED variables which permit run-time mapping of variables to memory locations. This is especially useful for passing parameters, relative and absolute addressing, and dynamic memory allocation.

Data Type Compatibility

PL/M 286 programs may be recompiled and retargetted to the 386 by use of the WORD16 control. With this control, PL/M 386 provides transparent access to the seven data types provided by PL/M 286.

Two Data Structuring Facilities

In addition to the 14 data types and based variables, PL/M supports two powerful data structuring facilities. These help the user organize data into logical groups.

- Array: Indexed list of same type data elements
- Structure: Named collection of same or different type data elements
- Combinations of both: Arrays of structures or structures of arrays and structures within structures.

Numerics Support

PL/M programs that use 32-bit REAL data are executed using the 80287 Numeric Data Processor for high performance. All floating-point operations supported by PL/M are executed on the 80287 according to the IEEE floating-point standard. PL/M 386 programs can use built-in functions and predefined procedures—INIT$REAL$MATH$UNIT, SET$REAL $MODE, GET$REAL$ERROR, SAVE$REAL$ STATUS, RESTORE$REAL$STATUS—to control the operation of the 80287 within the scope of the language.

Built-In Port I/O

PL/M 386 directly supports input and output from the 386 ports for single BYTE, HWORD and WORD transfers. For BLOCK transfers, PL/M 386 programs can make calls to predefined procedures.

Interrupt Handling

PL/M 386 has the facility for generating and handling interrupts on the 386. A procedure may be defined as an interrupt handler through use of the INTERRUPT attribute. The compiler will then generate code to save and restore the processor status on each execution of the user-defined interrupt handler routine. The PL/M statement CAUSE$INTERRUPT allows the user to trigger a software interrupt from within the program.

Protection Model

PL/M 386 support the implementation of protected operating system software by providing built-in procedures and variables to access the protection mechanism of the 386. Predefined variables—TASK$REGISTER, LOCAL$TABLE, MACHINE$ STATUS, CONTROL$REGISTER, etc.—allow direct access and modification of the protection system. Untyped procedures and functions—SAVE$ GLOBAL$TABLE, RESTORE$GLOBAL$TABLE, SAVE$INTERRUPT$TABLE, RESTORE$INTERRUPT$TABLE, CLEAR$TASK$SWITCHED$FLAG, GET$ACCESS$RIGHTS, GET$SEGMENT$LIMIT, SEGMENT$READABLE, SEGMENT$WRITABLE, ADJUST$RPL—provide all the facilities needed to implement efficient operating system software.

Compiler Controls

The PL/M 386 compiler offers controls that facilitate such features as:

— Interface to other 386 languages
— Optimization
— Conditional compilation
— The inclusion of additional PL/M source files from disk
— Cross-reference of symbols
— Optional assembly language code in the listing file
— The setting of overflow conditions for run-time handling.
— WORD16/WORD32
— Interface to 286 languages

Addressing Control

The PL/M 386 compiler uses the SMALL and COMPACT controls to generate optimum addressing instructions for programs. Programs of any size can be easily modularized into "subsystems" to exploit the most efficient memory addressing schemes. This lowers total memory requirements and improves run-time execution of programs.

Code Optimization

The PL/M 386 compiler offers four levels of optimization for significantly reducing overall program size.

— Combination or "folding" of constant expressions; and short-circuit evaluation of Boolean expressions
— "Strength reductions": a shift left rather than multiply by 2; and elimination of common subexpressions within the same block
— Machine code optimizations; elimination of superfluous branches; removal of unreachable code
— Optimal local register allocation

Error Checking

The PL/M 386 compiler has a very powerful feature to speed up compilations. If a syntax or program error is detected, the compiler will skip the code generation and optimization passes. This usually yields a 2X performance increase for compilation of programs with errors.

A fully detailed and helpful set of programming and compilation error messages is provided by the compiler and user's guide.

Cost-Effective Alternative to Assembly Language

PL/M 386 programs are code efficient. PL/M 386 combines all of the benefits of a high-level language (ease of use, high productivity) with the ability to access the 386 architecture. Consequently, for the development of systems software, PL/M 386 is the cost-effective alternative to assembly language programming.

Support

Hotline Telephone Support, Software Performance Report (SPR), Software Updates, Technical Reports, and Monthly Technical Newsletters are available.

ORDERING INFORMATION

Part Number	Description	Operating Environment
X286PLM386	PL/M 386 Compiler	XENIX* 286/310
D86PLM386	PL/M 386 Compiler	PC-DOS 3.0 or Greater

Documentation Package
PL/M 386 User's Guide for Xenix* 286 System

*XENIX is a trademark of Microsoft.

C 386
C COMPILER FOR THE 386

- **Implements full C Language**
- **Produces High Density Code Rivaling Assembler**
- **Supports Intel Object Module Format (OMF)**
- **Written in C**
- **Supports IEEE Floating Point Math with 80287 Coprocessor**
- **Supports Bit Fields**
- **Supports Full Standard I/O Library (STDIO)**

Intel C 386 brings the full power of the C programming language to the 386 microprocessor system. Intel C386 supports the full C language as described in the Kernighan and Ritchie book, "The C Programming Language", (Prentice-Hall, 1978). Also included are the latest enhancements to the C language: structure assignments, functions taking structure arguments and returning structures, and the "void" and "enum" data types.

Intel C 386 Compiler Description

The C 386 compiler operates in several phases: preprocessor, parser and code generator. The preprocessor phase interprets directives in C source code, including conditional compilations (# define). The parser phase converts the C program into an intermediate free form and does all syntactic and semantic error checking. The code generator phase converts the parser's output into an efficient intermediate binary code, performs constant folding, and features an extremely efficient register allocator, ensuring high quality code. The code generator outputs relocatable Intel Object Module Format (OMF) code, without creating an intermediate assembly file. The C386 compiler eliminates common code, eliminates redundant loads and stores, and resolves span dependencies (shortens branches) within a program.

The C 386 runtime library consists of a number of functions which the C programmer can call. The runtime system includes the standard I/O library (STDIO), conversion routines, routines for manipulating strings, and (where appropriate) routines for interfacing with the operating system.

C 386 uses Intel's Binder and Builder and generates debug records for symbols and lines on request, permitting access to Intel's PSCOPE Monitor/ICE™ emulator to aid in program testing.

FEATURES

Preprocessor Directives

#define—defines a macro

#include—includes code outside of the program source file

#if—conditionally includes or excludes code

Other preprocessor directives include #undef, #ifdef, #ifdef, #else, #endif, and #line.

Statements

The C language supports a variety of statements:

Conditionals: If, IF-ELSE

Loops: WHILE, DO-WHILE, FOR

Selection of cases: SWITCH, CASE, DEFAULT

Exit from a function: RETURN

Loop Control: CONTINUE, BREAK

Branching: GOTO

Expressions and Operators

The C language includes a rich set of expressions and operators.

Primary expression: invoke functions, select elements from arrays, and extract fields from structures or unions.

Arithmetic operators: add, subtract, multiply, divide, modulus

Relational operators: greater than, greater than or equal, less than, less than or equal, not equal

Unary operators: indirect through a pointer, compute an address, logical negation, ones complement, provide the size in bytes of an operand.

Logical operators: AND, OR

Bitwise operators: AND, exclusive OR, inclusive OR, bitwise complement

Data Types and Storage Classes

Data in C is described by its type and storage class. The type determines its representation and use, and the storage class determines its lifetime, scope, and storage allocation. The following data types are fully supported by C 386.

char: an 8 bit signed integer

int: a 32 bit signed integer

short: a 16 bit signed integer

long: a 32 bit signed integer

unsigned: a modifier for integer data types (char, int, short, and long) which doubles the positive range of values

float: a 32 bit floating point number which utilizes the 80287

double: a 64 bit floating point number

void: a special type that cannot be used as an operand in expressions; normally used for functions called only for effect (to prevent their use in contexts where a value is required).

enum: an enumerated data type

These fundamental data types may be used to create other data types including: arrays, functions, structures, pointers, and unions.

The storage classes available in C 386 include:

register: suggests that a variable be kept in a machine register, often enhancing code density and speed

extern: a variable defined outside of the function where it is declared; retaining its value throughout the entire program and accessible to other modules

auto: a local variable, created when a block of code is entered and discarded when the block is exited

static: a local variable that retains its value until the termination of the entire program

typedef: defines a new data type name from existing data types

BENEFITS

Faster Compilation

Intel C 386 compiles C programs substantially faster than standard C compilers because it produces Intel OMF code directly, eliminating the traditional intermediate process of generating an assembly file.

Portability of Code

Because Intel C 386 supports the STDIO and produces Intel OMF code, programs developed on a variety of machines can easily be transported to the 386.

Full Manipulation of the 386

Intel C 386 enables the programmer to utilize features of the C language to control bit fields, pointers, addresses and register allocation, taking full advantage of the fundamental concepts of the 386.

Support

Intel offers several levels of support for this product which are explained in detail in the price list. Please consult the price list for a description of the support options available.

ORDERING INFORMATION

Part Number	Description	Operating Environment
X286C386PP	C 386 Compiler	XENIX* 286/310 System
VVS386	C 386 Compiler	VAX/VMS 4.3 and later
D86C386	C 386 Compiler	PC-DOS 3.0 or greater

Documentation Package

C 386 User's Guide for Xenix* 286 System

*XENIX is a trademark of Microsoft.