MULTIPROCESSOR SYSTEM DESIGN - TUTOR

(EXPERT SYSTEM APPROACH)

BY

RAKESH KAMDAR

Thesis submitted to the Faculty of the Graduate School of the New Jersey Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering
/1989

1990

APPROVAL SHEET

Title of Thesis: Multiprocessor System Design - Tutor (Expert System Approach)

Name of Candidate: Rakesh Kamdar

Master of Science in Electrical Engineering

19~~89~~[illegible]

Thesis and abstract approved:

Dr. John Carpinelli Department of Electrical and Computer Engineering	Date
Dr. Stanley Reisman Department of Electrical and Computer Engineering	Date
Dr. Solomon Rosenstark Department of Electrical and Computer Engineering	Date

VITA

Name: Rakesh Kamdar

Degree and date to be confirmed: M. S. (E. E.), 19~~89~~[illegible]

Secondary Education:

College	**Date**	**Degree**	**Date of Degree**
L.D.College of Engg.	1981-85	B.S. (E.C.)	1985
NJIT	1986-89	M.S. (E.E.)	1989

Major: Electrical Engineering

Position Held: Systems Engineer

Lobb Systems Inc.

20 Corporate Place 128

Wakefield, MA 01880

ABSTRACT

Title of Thesis: Multiprocessing System Design - Tutor

Rakesh I. Kamdar, Master of Science in Electrical Engineering, 19~~89~~90

Thesis Directed by: Dr. John Carpinelli, Assistant Professor
Dept. of Electrical and Computer Engineering.

To increase computational bandwidth and system resilience, integration of several microprocessors in a single system becomes necessary. The overall throughput and efficiency of such a system is directly dependent on the hardware and software interconnection supported by the basic microprocessor chip. Sometimes it becomes difficult to put together all the information for design criteria and all the design related formulas.

The approach made here is to continuously update the hardware and software information in the database related to a given microprocessor. This information can be accessed at any time for efficient design solution. Intel 80386 and Motorola 68020 microprocessors are reviewed in detail and all the information is stored in a database.

The above approach has been implemented in the Multiprocessor System Design - Tutor (MSDT) using the Informix relational database management system. MSDT is a menu driven system implemented to help the system design engineers. MSDT

stores and maintains information related to multiprocessor system design, which includes multiprocessor system requirements, microprocessor characteristics, the role of microprocessor in multiprocessor system design and interconnection network configurations and their performance factors. This information is presented to the user via the screen building utility of Informix-4GL; the user can also get a hard copy of all the information within the database by running the report generation utility. MSDT also has security password protection. The system has a good help facility available for the design process. At any given time the user can update the data in the table using this menu driven system.

The system is intended to grow into a complete evaluation system based on the Informix-4GL. It is developed on the basis of Fourth Generation Language which has a screen building utility, a menu building utility, a report writer and a window manager.

This system will suggest the candidate microprocessor and suitable support chips and interconnection techniques for different applications.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION TO MULTIPROCESSING

High performance computers are in demand in the areas of structural analysis, weather forecasting, petroleum exploration, fusion energy research, medical diagnosis, aerodynamics, and simulations. They are also needed in artificial intelligence, expert systems, industrial automation, remote sensing, military defense, and genetic engineering, among many other scientific and engineering applications. Without superior computers, many of these challenges to advance human civilization cannot be made within a reasonable time period. It has become necessary for human beings to develop advanced computer architectures and various VLSI technologies. The new era of multiprocessing involves the efforts of the hardware designer for efficient utilization of available resources at a low cost [1].

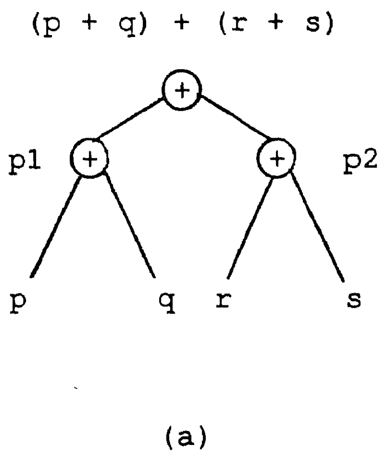
This thesis is devoted to study some features of multiprocessors and the capabilities of presently available microprocessors for multiprocessing. This chapter of the thesis covers the following two features of multiprocessing protocols:

* Context switching
* Operating systems

A multiprocessor is defined as having these characteristics:

* A multiprocessor contains two or more processors of approximately comparable capabilities.
* All processors share access to common storage.
* All processor share access to input/output channels, control units, and input/output devices.
* The entire system is controlled by one operating system providing interaction between processors and their programs at the job, task, step, data set, and data element levels.

One advantage of the multiprocessor is that it reduces the height of the formula tree by performing independent calculations simultaneously. The following is an example of the simple operation (p + q + r + s). Here, two processes, (p + q) and (r + s), can be divided as shown in Figure 1.1.a. If the same operation is performed by the single processor, the operation tree is as shown in Figure 1.1.b.



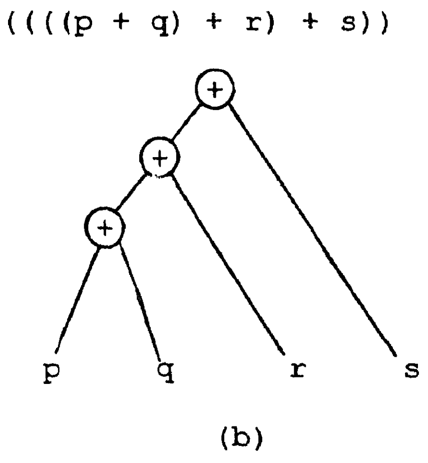


Figure 1.1 Multiprocessor tree structure

Thus it is very advantageous to use the multiprocessor configuration with the least cost increase [1].

1.2 TRENDS TOWARD MULTIPROCESSING

Multiprocessors are computer systems encompassing more than one general-purpose processor, each capable of executing a separate instruction stream, all of them sharing a global memory. One of the most important components in a multiprocessor is the processor memory interconnection network [4].

The first multiprocessor is S-1, developed by Widdoes and Correll[4]. The S-1, as many of the multiprocessors built in the past, consists of a few processors and a few memory modules interconnected through a crossbar switch. Other examples in this class include the Burroughs 5000 series, the Burroughs D825 command and control computer, and Carnegie-Mellon's C.mmp. [4]

The second multiprocessor is Cm* from Carnegie-Mellon University. The interconnection used in Cm* consists of asynchronous buses; this allows the inclusion of processors and memory modules at a cost roughly linear in their number.

The third multiprocessor is Burroughs' FMP. Even though it has not been built, a careful study of its feasibility and performance has been done. The interconnection network chosen for this machine was the baseline network[4], whose cost is proportional to Nlog N where N is the number of processors. This seems to be a good compromise between the costly crossbar switch and slow asynchronous buses.

One of the problems affecting the performance of multiprocessors is memory interfaces. Independent processors provide the flexibility which makes multiprocessors ideal for the exploitation of parallelism in a range of applications where pipelined and array processors are not effective [4].

1.3 DESIGN AND IMPLEMENTATION OVERVIEW OF MSDT

MSDT (Multiprocessor System Design - Tutor) is a menu driven informative and intelligent system which keeps track of all the recent and important characteristics of microprocessors and interconnection networks.

The design of MSDT includes the different tables in the database and these tables are maintained through the screens developed using Informix-4GL (Fourth Generation Language) and SQL (Structured Query Language). This design is implemented on an IBM PC AT compatible computer and uses Microsoft C and PLINK (Phoenix Technology Group) to link all the larger Informix modules.

This system has three basic sections:

1. View function to look at the data of the multiprocessor and microprocessor.

2. Maintenance function to keep all the information up to date for future applications.
3. Report writing, which produces the report on the printer for system configuration, network or multiprocessor characteristics.

This system is very simple to use and all the instructions are provided when the user is executing MSDT application.

1.4 OUTLINE OF THE REST OF THE THESIS

The rest of this thesis is organized as follows. Chapter 2 introduces the need of automation for multiprocessing and basic information on the 68020 and 80386 microprocessors. Chapter 3 introduces the expert system approach to solve the multiprocessor system design and also describes the basic tools of Informix. Chapter 4 describes the basic introduction to interconnection networks and some mathematical formulas to evaluate the interconnection network. Chapter 5 describes a basic database design, modules, screen forms and reports generated. Chapter 6 is basically an operation guide for the software. Chapter 7 provides conclusions and suggestios for future enhancements. The Appendix shows the actual program code.

CHAPTER 2

MICROPROCESSOR CHARACTERISTICS AND SUPPORT

2.1 NEEDS OF AUTOMATION

The solution to multiprocessor design is a computerization of the data storage, so that one can try different ideas without implementing the actual hardware. It is important to integrate the system in such a way that all users can access and continuously update the information. This will facilitate the creation of an efficient system. Intel 80386 and Motorola 68020 microprocessors are reviewed in detail for the system implementation and other microprocessors can be added to the system.

2.2 INTRODUCTION TO THE 68020 MICROPROCESSOR

The MC68020 is the first full 32-bit implementation of the M68000 family of microprocessors from Motorola. Using VLSI technology, the MC68020 is implemented with 32-bit register and data paths, 32-bit addresses, a rich instruction set, and versatile addressing modes [5].

The MC68020 is object code compatible with the earlier members of the M68000 family and has the added features of new addressing modes in support of the high level languages, an on chip instruction cache, and a flexible co-processor interface with full IEEE floating point support. Also the internal operations of this microprocessor are designed to operate in parallel allowing multiple instructions to be executed concurrently.

The resources available for the MC68020 are:

* Virtual Memory/Machine support
* Sixteen 32-bit General purpose data and address registers
* Two 32-bit supervisor stack pointers
* 32-bit program counter
* Five special purpose control registers
* 4 Gigabyte Direct Addressing Range
* 18 Addressing modes
* Memory mapped I/O
* Coprocessor interface
* High performance on-chip instruction cache
* Operations on seven data types
* Complete Floating-point support via the MC68881 coprocessor

A block diagram of the MC68020 is shown in Figure 2.1. The major blocks depicted operate in a highly independent fashion that maximizes concurrency of operation while managing the essential synchronization of the instruction execution and the bus operation.

The bus controller loads instructions from the data bus into the decode unit and the on-chip cache. The sequencer and control unit provide overall chip control, managing the internal buses, registers, and the function of the execution unit.

2.2.1 EXCEPTION PROCESSING

The exception processing state is associated with interrupts, trap instructions, tracing, and other exceptional conditions. The exception may be internally generated by an instruction or by an unusual condition arising during the execution of the instruction. Exception processing can also be initiated by conditions external to the processor such as an interrupt, a bus error, a reset, or a coprocessor primitive command. Thus, exception processing is designed to provide an efficient context switch so that the processor may quickly and gracefully handle the unusual conditions.

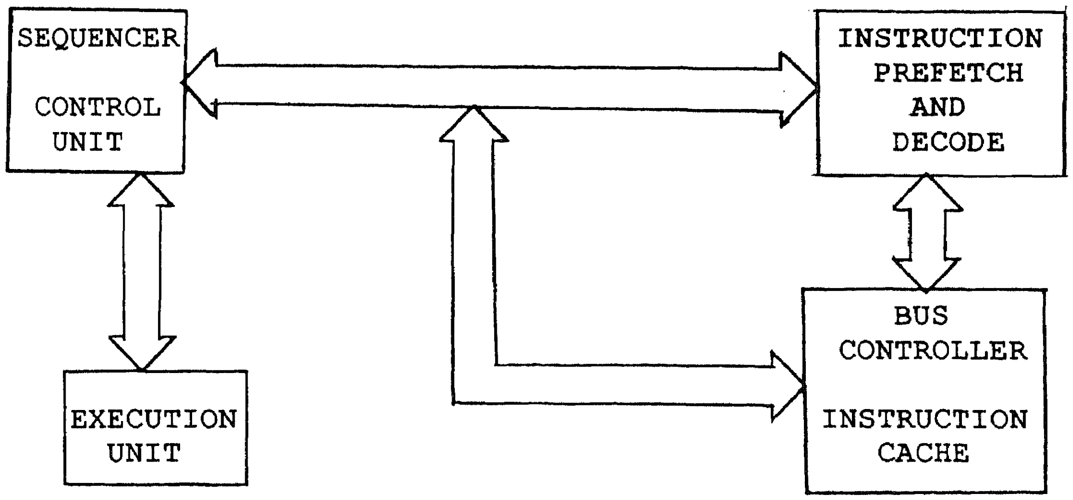


Figure 2.1 Block diagram of MC68020

2.2.2 EXCEPTION PROCESSING SEQUENCE

Exception processing occurs in four identifiable steps.

1. An internal copy is made of the status register. After the copy is made, the processor state bits in the status register are changed. The S bit is set, putting the processor into the supervisor privilege state. The T1 and T0 bits are cleared, which allows the exception handler to execute unhindered by tracing. For the reset and interrupt exceptions, the interrupt priority mask is also updated.

2. The vector number of the exception is determined. For interrupts, the vector number is obtained by the processor read from CPU address $F, which is defined as an interrupt acknowledge cycle.

3. Save the current processor context. An exception stack frame is created and filled on the active supervisor stack. Other information may also be stacked, depending on which exception is being processed and the context of the processor prior to the exception. If the exception is an interrupt and the M bit is set, the M bit is cleared, and a second stack frame is created on the interrupt stack.

4. This step is the same for all the exceptions. The exception vector offset is determined by the multiplying the vector number by four. This offset is then added to the contents of the vector base register to determine the memory address of the exception vector. The program counter value is loaded with the value in the exception vector. The instruction at the address given in the exception vector is fetched, and instruction decoding and execution is resumed.

2.2.3 EXCEPTION STACK FRAME

Exception processing saves the most volatile portion of the current processor context on the top of the supervisor stack. The context is organized in a format called the exception stack frame. This information always includes the status register, the program counter, and the vector offset used to fetch the vector. The processor also marks the stack frame with a frame format. The format field allows the RTE instruction to identify what information is on the stack so that it may be properly restored and the stack space deallocated.

2.2.4 MULTIPLE EXCEPTION

The priority relationship between two exceptions determines which is processed first if both exceptions occur simultaneously. The term "process" in this context means the execution of the four steps defined previously. "Process" in this context does not include the execution of the routine pointed to by the fetched vector. As soon as the processor has completed processing for an exception, it is then ready to begin execution of the exception handler routine, or begin exception processing for other pending exceptions. Also, a higher priority exception can be processed before the completion of exception processing for the lower priority exceptions.

This priority scheme is very important in determining the order in which exception handlers are executed in multiple exception situations. As a general rule, the lower the priority of an exception, the more quickly the handler routine for that exception will be executed. An exception to this rule is the Reset exception, which is the highest priority and also the first exception handled, since all other exceptions are cleared by the reset condition [6].

2.2.5 RETURN FROM THE EXCEPTION

After exception stacking operations have been completed for all the pending exceptions, the processor resumes normal instruction execution at the address contained in the vector referenced by the last exception to be processed. Once the exception handlers have completed the execution, the processor must return to the system context prior to the exception. The mechanism used to accomplish this return for any exception is the RTE instruction.

When the RTE instruction is executed, the processor examines the stack frame on the top of the active supervisor stack to determine if it is a valid frame and what type of the context restoration should be performed.

2.2.6 MC68020 EXCEPTION STACK FRAMES

The MC68020 generates six different stack frames. Whenever the MC68020 writes or reads a stack frame, it will use long word operand transfers whenever possible. Thus, if the stack area resides in a 32-bit ported memory and the stack pointer is longword aligned, exception processing performance will be greatly enhanced. Also, the order of the bus cycles used by the processor to write or read a stack frame may not follow the order of the data in a frame [6].

The six different stack frames are:

1. Normal four word stack frame
2. Throwaway four word stack frame
3. Normal six word stack frame
4. Coprocessor Mid-instruction Exception stack frame
5. Short bus cycle fault stack frame
6. Long bus cycle fault stack frame

NORMAL FOUR WORD STACK FRAME

This frame is created by interrupts, format errors, TRAP#n instructions, illegal instructions, A-line and F-line emulator traps, privilege violation, and co-processor preinstruction exceptions. The program counter value is the address of the next instruction to be executed, or the instruction that caused the exception, depending upon the exception type.

THROWAWAY FOUR WORD STACK FRAME

This stack frame is the throwaway frame that is created on the interrupt stack during exception processing for an interrupt when a transition from the master state to the interrupt state occurs. The program counter value on the normal four word or coprocessor mid-instruction exception stack frame that was created on the master stack.

NORMAL SIX WORD STACK FRAME

This stack frame is created by instruction related exceptions which include coprocessor post-instruction exceptions, CHK, CHK2, TRAPcc, TRAPV trace, and zero divide. The instruction address value is the address of the next instruction that caused the exception. The program counter value is the address of the next instruction to be executed, and the address to which the RTE instruction will return.

COPROCESSOR MID-INSTRUCTION EXCEPTION STACK FRAME

This stack frame is created for three different exceptions, all related to coprocessor operations.

SHORT BUS CYCLE FAULT STACK FRAME

This stack frame is created whenever a bus cycle fault is detected, and the processor recognizes that it is at an instruction boundary and can use this reduced version of the bus fault stack frame. The program counter value is the address of the next instruction to be executed.

LONG BUS CYCLE FAULT STACK FRAME

This stack frame is created whenever the processor detects a bus cycle fault and recognizes that it is not an instruction boundary. The program counter value is the address of the instruction that was executing when the fault occurred.

Table 2.1 gives a summary of the stack frames.

FORMAT	FRAME TYPE
0000	SHORT FRAME
0001	THROWAWAY
0010	INSTRUCTION EXCEPTION
0011-0111	UNDEFINED, RESERVED
1000	MC68010 BUS FAULT
1001	COPROCESSOR MID INSTRUCTION
1010	MC68020 SHORT BUS FAULT
1011	MC68020 LONG BUS FAULT
1100-1111	UNDEFINED, RESERVED

Table 2.1 Stack Frames

The exception processing may occur from the following causes:

* Reset

* Bus error

* Instruction traps

* Breakpoints

* Format error

* Illegal instructions or unimplemented instruction

* Privilege violations

* Tracing

* Interrupts

* Return from exception

2.3 INTRODUCTION TO THE 80386 MICROPROCESSOR

The 80386 is an advanced 32-bit microprocessor for applications needing very high performance and optimized for multitasking operating systems. The 32-bit registers and data paths support 32-bit addresses and data types. The processor addresses up to four gigabytes of physical memory and 64 terabytes of virtual memory. The integrated memory management and protection architecture includes address transition registers, advanced multitasking hardware and a protection mechanism to support operating systems. In addition, the 80386 allows the simultaneous running of the operating systems [5].

Instruction pipelining, on-chip address translation, and high bus bandwidth ensure short average instruction execution times and high system throughput. The 80386 processor is capable of execution at sustained rates of between three and four million instructions per second.

The resources available for the 80386 are:

* Flexible 32-bit microprocessor
  - 8, 16, 32-bit data types
  - 8 general purpose 32-bit register
* Very large address space
  - 4 Gigabyte physical
  - 64 terabyte virtual
  - 4 gigabyte maximum segment size
* Integrated memory management unit
  - Virtual memory support
  - Optional on-chip paging
  - 4 levels of protection
  - Fully compatible with the 80286
* Object code compatible with all 8086 family microprocessors
* Virtual 8086 mode allows running of 8086 software in a protected and paged system
* Hardware debugging support
* Optimized for system performance
  - Pipelined instruction execution
  - On-chip Address translation caches
  - 12.5 and 16 MHz clock

- 32 megabytes/sec bus bandwidth

* High speed numeric support via 80287 and 80387 coprocessors
* Complete system development support
    - Software: C, PL/M, Assembler system generation tools
    - Debuggers: PSCOPE, ICE - 386
* High speed CHMOS III technology
* 132 pin grid array package

The 80386 offers new testability and debugging features. Testability features include a self test and direct access to the page translation cache. Four new breakpoints registers provide breakpoint traps on code execution or data accesses for powerful debugging of even ROM-based systems.

Object-code compatibility with 80XX family members means that the 80386 offers immediate access to the world's largest microprocessor base [5].

2.3.1 TERMINOLOGY

The following is the terminology used throughout the discussion of the descriptor tables.

PL:	Privilege Level
RPL:	Requester Privilege Level
DPL:	Descriptor Privilege Level
CPL:	Current Privilege Level
EPL:	Effective Privilege Level

TASK: One instance of the execution of a program. Tasks are also referred to as processes.

2.3.2 PROTECTION

The 80386 has four levels of protection which are optimized to support the needs for a multi-tasking operating system to isolate and protect user programs from each other and the operating system. The privilege levels control the use of the privileged instructions, I/O instructions, and access to segments and segment descriptors. Unlike traditional microprocessor based systems, where this protection is achieved only through the use of complex external hardware and software, the 80386 provides the protection as part of its integrated Memory Management Unit. The 80386 offers an additional type of protection on a page basis, when paging is enabled.

It is an extension of the user/supervisor privilege mode commonly used by minicomputers and in fact, the user/supervisor mode is fully supported by the 80386 paging mechanism. The privilege level 0 is the most privileged or trusted level. The privilege levels are numbered 0 through 3.

2.3.3 RULES OF PRIVILEGE

The 80386 controls the access to both data and procedures between levels of a task according to the following rules:

* Data stored in a segment with privilege level **p** can be accessed by the code executing at a privilege level at least as privileged as **p**.

* A code segment/procedure with privilege level **p** can only be called by a task executing at the same or a lesser privilege level than **p**.

2.3.4 DESCRIPTOR TABLES

The descriptor tables define all of the segments which are used in an 80386 system. There are three types of tables in the 80386 that hold descriptors:

GDT: Global Descriptor Table

LDT: Local Descriptor Table

IDT: Interrupt Descriptor Table

All of the tables are variable length memory arrays. They can range in size between 8 bytes and 64K bytes. The upper 13 bits of the selector are used as an index into the descriptor table.

Each of the tables has a register associated with it. The GDTR, LDTR, IDTR. The LGDT, LLDT, and LIDT instructions load the base and limit of each descriptor, while the SGDT, SLDT, and SIDT instructions store the base and limit of each descriptor. These tables are manipulated by the operating system only. Therefore these are the privilege instructions.

GDT

The Global Descriptor Table contains descriptors that are possibly available to all of the tasks in the system. The GDT can contain any type of segment descriptor except for the descriptors which are used for servicing interrupts. Every 386 system contains code and data segments used by the operating systems and task state segments, and descriptor for the LDTs in a system.

LDT

LDTs contain descriptors which are associated with a given task. Generally, operating systems are designed so that each task has a separate LDT. The LDT may contain only code, data, stack, task gate, and call gate descriptors. LDTs provide a mechanism for isolating a given task's code and data segments from the rest of the operating system, while the GDT contains descriptors for segments which are common to all tasks if its segment descriptor does not exist in either the current LDT or the GDT. This provides both isolation and the protection for a task's segments, while still allowing global data to be shared among the tasks.

IDT

The third table needed for the 80386 system is the interrupt descriptor table. The IDT contains the descriptors which point to the location of up to 256 interrupt service routines. The IDT may contain only task gates, interrupt gates, and trap gates. The

IDT should be at least 256 bytes in size in order to hold the descriptors for the 32 Intel reserved interrupts. Every interrupt used by a system must have an entry in the IDT. IDT entries are referenced via INT instructions, external interrupt vectors, and exceptions.

2.3.5 PRIVILEGE LEVELS

Task Privilege

At any point in time, a task on the 80386 is always executed at one of the four privilege levels. The Current Privilege Level specifies the task's privilege level. A task's CPL may only be changed by control transfer through gate descriptor to a code segment with a different privilege level. Thus, an application program running at PL = 3 may call an operating system routine at PL = 1 which would cause the task's CPL to be set to 1 until the operating system routine was finished.

Selector Privilege

The privilege level of a selector is specified by the RPL field. The RPL is the two least significant bits of the selector. The selector's RPL is only used to establish a less trusted privilege level than the current privilege level for the use of a segment. This level is called the task's effective privilege level for the use of a segment. The EPL is defined as being the least privileged level of a CPL and a selector's RPL. Thus, if the selector's RPL = 0 then the CPL always specifies the privilege level for marking an access using the selector. On the other hand if RPL = 3 then a selector can only access segment level 3 regardless of the task's CPL. The RPL is most commonly used to verify that pointers passed to an operating system procedure do not access data that is of higher level privilege than the procedure that originated the pointer. Since the originator of a selector can specify an RPL value, the Adjust RPL instruction is provided to force the RPL bits to the originator's CPL.

2.3.6 PRIVILEGE VALIDATION

The 80386 provides several instructions to speed pointer testing and help maintain system integrity by verifying that the selector value refers to an appropriate segment.

This pointer verification prevents the common problem of an application at PL = 3 calling a operating system routine at PL = 0 and passing the operating system routine a "bad" pointer which corrupts a data structure that belongs to the operating system. If the operating system routine uses the ARPL instruction to ensure that the RPL of the selector has no greater privilege than that of the caller, then this problem can be avoided.

DESCRIPTOR ACCESS

There are basically two types of segment accesses: those involving code segments such as control transfers, and those involving data accesses. Determining the ability of a task to access a segment involves the type of descriptor used and CPL, RPL, and DPL as described above.

Any time an instruction loads the data segment register the 80386 makes protection validation checks. Selectors loaded in the DS, ES, FS, and GS registers must refer only to data segments or readable code segments. The data access rules are specified in advance. The only exception to those rules are readable conforming code segments which can be accessed at any time.

The rules regarding the stack segment are slightly different than those involving data segments. Instructions that load the selector in to the Stack Segment must refer to data segment descriptors for writable data segments. The DPL and RPL must equal the CPL. All other descriptor types or a privilege level violation will cause exception 13. A stack not present fault causes exception 12. Note that an exception 11 is used for a not-present code or data segment.

2.3.7 PRIVILEGE LEVEL TRANSFER

Inter-segment control transfers occur when a selector is loaded in the CS register. For a typical system most of these transfers are simply the result of a call or a jump to another routine. There are five types of control transfers which are described. Many of these transfers result in a privilege level transfer. Changing privilege levels is done only via control transfers, by using gates, task switches, and interrupt or trap gates.

Control transfers can only occur if the operation which loaded the selector references the correct descriptor types. Any violation of these descriptor usage rules will cause an exception 13.

In order to provide further system security, all control transfers are also subject to the privilege rules.

The privilege rules require that:

- Privilege level transitions can occur only via gates.
- Jumps can be made to a non-confirming code segment with the same privilege or to a conforming code segment with greater or equal privilege.
- CALLs can be made to a non-conforming code segment with the same privilege or via a gate to a more privilege level.
- Interrupts handled within the task obey the same privilege rules as CALLS.
- Conforming code segments are accessible by privilege levels which are the same or less privileged than the conforming code

segment's DPL.

- Both the requested privilege level (RPL) in the selector pointing to the gate and the task's CPL must be of equal or greater privilege than the gate's DPL.

-The code segment selected in the gate must be the same or more privileged than the task's CPL.

- Return instructions that do not switch tasks can only return control to a code segment with the same or less privilege.

- Task switches can be performed by the CALL, JMP, or INT instructions, which reference either a task gate or a task state segment whose DPL is less privileged or the same privilege as the old task's CPL.

Any control transfer that changes the CPL within a task causes a change of stacks as a result of the privilege level change. The initial value of stack segment for privilege levels 0, 1, and 2 is retained in the task state segment. During a CALL or JUMP control transfer, the new stack pointer is loaded into the SS and ESP registers and the previous stack pointer is pushed onto a new stack.

2.3.8 TASK SWITCHING

A very important attribute of any multi-tasking/multiuser operating systems is its ability to rapidly switch between tasks or processes. The 80386 directly supports this operation by providing a task switch instruction in hardware. The 80386 task switch operation saves the entire state of the machine, loads the new execution state, performs protection checks, and commences execution in the new task in about 17 microseconds. Like transfer of control via gates, the task switch operation is invoked by executing an inter-segment JMP or CALL instruction, which refers to a Task State Segment (TSS), or a task gate descriptor in the GDT or LDT. An INT n instruction, exception, trap, or external interrupt may also invoke the task switch operation if there is a task gate descriptor in the associated IDT descriptor slot.

The TSS descriptor points to a segment containing a TSS selector. The 80386 supports both 286 and 386 style TSSs. The limit of the 386 TSS must be greater than 0064H, and can be as large as 4 Gigabytes. In the additional TSS space the operating system is free to store additional information such as the reason the task is inactive, the time the task has spent running, and open files belonging to the task.

Each task must have a TSS associated with it. The current TSS is defined by the special register in the 80386 called the Task State Segment Register (TR). This register contains a selector referring to the task state segment descriptor that defines the current TSS. A hidden base and limit register associated with TR is loaded whenever TR is loaded with a new selector. Returning from a task is accomplished by the IRET instruction. When IRET is executed, control is returned to the task which was interrupted. The current executing task's state is saved in the TSS and the old task state is restored from its TSS.

Several bits in the flag register and machine status word give information about the state of a task which are useful to the operating system. The Nested Task (NT) bit controls the function of the IRET instruction. If NT = 0, the IRET instruction performs the regular return; when NT = 1, IRET performs a task switch operation back to the previous task. The NT bit is set or reset in the following fashion:

When a call or INT instruction initiates the task switch, the new TSS will be marked busy and the back link field of the new task is set to the old TSS or INT initiated task switches. An interrupt that does not cause a task switch will clear NT. NT may also be set or cleared by the POPF or IRET instructions.

The 386 task state segment is marked busy by changing the descriptor type field from TYPE 9H to TYPE BH. A 286 TSS is marked busy by the changing the descriptor type field from TYPE 1 to TYPE 3. Use of a selector that references a busy task state segment causes an exception 13.

The Virtual Mode (VM) bit 17 is used to indicate if a task is a virtual 8086 task. If VM = 1 than the tasks will use the real mode addressing scheme. The virtual 8086 environment is only entered and exited via a task switch.

The coprocessor state is not automatically saved when a task switch occurs, because the incoming task may not use the coprocessor. The task switched (TS) bit helps deal with the coprocessor's state in a multi-tasking environment. Whenever the 80386 switches tasks, it sets the TS bit. The 80386 detects the first use of a processor extension instruction after a task switch and causes the processor extension not available exception 7. The exception handler for exception 7 may then decide whether to save the state of the coprocessor. A processor extension not present exception will occur when attempting to execute a WAIT or ESC instruction if the Task Switched and Monitor coprocessor extension bits are both set.

The T bit in the 386 TSS indicates that the processor should generate a debug exception when switching to a task. If T = 1 then upon entry to a new task, a debug exception 1 will be generated.

CHAPTER 3

EXPERT SYSTEM AND DATABASE

3.1 EXPERT SYSTEM APPROACH

3.1.1 DEFINITION OF AN EXPERT SYSTEM

There are many areas where traditional computing methods cannot be applied. Here, experts are needed to gather and interpret data and select a strategy for solving a problem. Such problems are typically poorly specified, difficult to define and heavily dependent upon rules of thumb. A decision making system design by domain expert with the help of an expert is called an expert system.

3.1.2 BUILDING EXPERT SYSTEM

Over the years, expert systems have emerged as a major practical application of artificial intelligence. "Expert system" is the name given to software systems which augment the decision making process of human experts. These systems are designed to support and extend human problem solving.

At least two people are needed to create an expert system: a knowledge engineer and a domain expert. The domain expert is someone who is intimately familiar with the target problem.

During the knowledge acquisition phase of a project, the knowledge engineer acquires, by trial and error, a working knowledge of the domain expert's understanding. The model that results is not a static one. Gradually, as new aspects of the problem are introduced, the existing program is modified, and the complete system is then tested. This process is repeated cyclically. The prototype is thus expanded and refined in ever increasing degrees of detail and sophistication until the expert concludes that the system meets the standard of excellence in finding correct solutions to the problems at hand.

Throughout this process, knowledge about the problem is encoded in such a way that it can be interpreted by the expert system's inference mechanism (the part of the software that draws conclusions from the given set of facts and conditions). AI researchers refer to such codification as knowledge representation and to the sum of such representation as a knowledge base.

The reasons to use an expert system are the following:

a. Experts' time is valuable and in short supply.

b. Expert systems can become perfect as time goes by, thus ultimately eliminating the need for an expert. However, such a system should be able to diagnose as perfectly and as fast as the human expert.

c. Once the expertise is secured in an expert system, it can be copied, distributed, and used in far-flung locations, a difficult feat to achieve with the human expert.

3.2 DATABASE ENGINES

3.2.1 DATABASE CAPACITY

The limits of the database are important. The questions one should ask about capacity include limits on the number of databases, tables per database, and rows and columns per table. In addition, one needs to know the maximum size of a row and column, and how many fields can be indexed or how many indices can be stored for one table. The database management system must be able to accommodate the largest tables and databases that any business needs, now and in the future, so it is also important to understand whether the size of any of these individual items can exceed the physical size of a disk. As an application grows and requires additional disk storage, incorporating it into the database environment should be easy.

3.2.2 DATA TYPES

Data types describe the format of the data that is allowed for a column. In general, the most commonly used data types are integer, floating point, decimal, money, character and date. If the right data types are used, input displays, data storage, output format, and computations are much easier. Specific processes may dictate the need for special datatypes.

3.2.3 DATA INTEGRITY

Data integrity ensures that only valid types of data are stored in the database. With many different people updating a database, it is easy for users to attempt to store incorrect or invalid data. Integrity rules govern what can and cannot be stored. The assignment of a data type to a column creates the most basic type of integrity.

There is often a need to ensure that a value is entered into a field. If a database is able to store what is known as "a null value," then it can force the entry of a value for each update. Defining a field as "not null" forces the user to include important information.

Sometimes there is a need to enforce uniqueness within a field in a table. Defining a field as "unique" prevents duplicates, such as two employees with the same employee number. In addition, there are a number of other integrity constraints that a database or application can enforce. These constraints may involve relationships between fields in different tables.

3.2.4 DATA SECURITY

Sometimes there is confusion between security and integrity. In general, "Security means protecting the database against unauthorized users"[8]. Data security is an important factor in database selection. Because many corporations and people consider data as an asset of the corporation or a person, data must be protected against unauthorized individuals[8].

3.2.5 DATABASE RESTART AND RECOVERY

When data is being inserted and changed, a database can log or journal the database activity for use in recovery from failures. Usually there are archive logs stored on tape that contain previous "snapshots" of the database contents. In addition, there is an on-line log on disk that contains information about more recent transactions. In the event of a system failure, such as a power failure, the database should automatically recover without operator intervention. This can be done because DBMS (Database Management System) use disk logs to ensure that all committed transactions at the time of failure are in the database and all partial transactions are removed. The database is restored to a consistent state and is back on-line within minutes after a system restart.

3.3 DATABASE TOOLS: INFORMIX - SQL AND INFORMIX - 4GL

3.3.1 DATA DEFINITION

Informix provides both menu-driven and command driven interfaces to define databases. One can create and delete databases through a menu-driven interface or through structured query language commands included in a 4GL program. One can also define, rename, and delete tables, as well as update table definitions using either of the above-mentioned interfaces. The command oriented interface is also available from the QUERY LANGUAGE options. One can get this option from the main menu of SQL and 4GL. When one selects this option, it presents a blank screen. One can enter one or more RDSQL statements and execute them. A useful feature of the menu-oriented interface is that one can ask Informix to create an RDSQL command file based on the interactive input one provides. This can come in handy while one is still experimenting with database definitions. One can run the command file as is or can modify it through an editor and then run it. Another way to do this is with the DBSCHEMA utility, which will produce RDSQL statements required to replicate an entire database or a selected table.

3.3.2 FORM GENERATOR

A form generated by 4GL's form generator is used with a 4GL program, and a form generated by SQL's form generator is processed by PERFORM. With either form generator, one can create a default form and compile a form from a menu-oriented interface. If one wants to modify a form or create one from scratch, one has to learn specific syntax to modify and create from scratch. No menu-oriented capability exists to accomplish this.

A 4GL form specification contains five sections. The DATABASE section contains the name of the database. The SCREEN section contains a layout of the screen form. The screen layout cannot be more than 20 lines long, as 4GL reserves four lines of the screen for prompts, messages, comments, and error messages. The TABLES section lists the tables. The ATTRIBUTES section links field names (database or non-database) to field tags (contained in the screen section). One can assign more than one attribute to the field in this section. The INSTRUCTIONS section is used to define screen records or to change the default delimiters for display fields. Screen records can be used to group fields.

3.3.3 AD HOC QUERY, INSERT, UPDATE AND DELETION OF DATA FROM TABLE

Informix provides two methods for ad-hoc (without programming) query, insert, update, and deletion of data from tables. The first method is form-driven and involves generating a default form with SQL's form generator and processing it with PERFORM. When one executes the form using PERFORM it displays a horizontal menu with these choices: QUERY, NEXT, PREVIOUS, ADD, UPDATE, REMOVE, TABLE, SCREEN, CURRENT, MASTER, DETAIL AND OUTPUT. QUERY retrieves rows from a table based on the values entered in the form. Using the NEXT and PREVIOUS choices, one can go back and forward in the retrieved rows.

The second method is command oriented and it involves executing RDSQL statements from the query language option. This topic is covered in Section 3.3.1.

3.3.4 APPLICATION DEVELOPMENT

Informix supports three methods for building applications:

1. Generate forms using the form generator of SQL and run the forms under PERFORM. One can customize the form and can add some amount of control to it. This approach has limitations:

- One can run only one form at a time. It is not possible to present forms in a hierarchical fashion (one form displaying another form).
- The menus presented on the form cannot be customized.
- Complex computations and logic cannot be included.
- No database access statement can be included.

2. Generate forms with screen building utility of operating system and use forms in the routines written in C or COBOL programming language and embed SQL statements in these routines.

3. Generate forms with the form generator of the 4GL and use forms in the routines written in the 4GL programming language. A 4GL program consists of a set of routines. There are three types of routines: MAIN, FUNCTION and REPORTS. Before one runs the 4GL program, one has to preprocess the 4GL routine. Preprocessing converts the 4GL to C language and then it has to be compiled and linked. This project is implemented using this method.

3.3.5 REPORT WRITING

The 4GL product does not have a separate report generator; it is part of the 4GL programming language. The data is retrieved in a MAIN or FUNCTION routine, and the formatting and printing are controlled from a report routine. In the report routine, it is possible to include the other 4GL statements. This facility of the report routine makes it very flexible and powerful. One can include more complicated computations that require many lines of code. It is even possible to update the database in the middle of writing a report. One can combine the output of several SELECT statements into one report, and execute SELECT conditionally before one calls the REPORT routine.

3.3.6 VIEWS

One can create and drop views with RDSQL statements, and execute this statement from the query-language option or include them in a 4GL routine. Informix imposes some restrictions on updating tables through views. A view column may be updated only if it is derived directly from a column in a table of the database and not a result of an expression. Expression-derived columns are called virtual columns. One can insert rows through a view that contains virtual columns, although one may delete a row that contains a virtual columns. One cannot build indices on views.

3.3.7 JOIN

Informix supports two methods of handling joins. The first method is command-oriented. It involves using the SELECT statement with a where clause between at least one column from one table and at least one column from the other. One can execute a SELECT statement whenever one wants to join two tables or save the definition as a view. One can also join more than two tables having one-to-one or one-to-many relationships. The second method is form oriented. It involves using SQL's form generator and PERFORM. One can ask Informix to create an RDSQL command file based on the interactive input one provides. This can come in handy while one is still experimenting with database definitions. One can run the command file as is or can modify it through an editor and then run it. Another way to do this is with the DBSCHEMA utility, which will produce RDSQL statements required to replicate an entire database or a selected table.

CHAPTER 4

INTERCONNECTION NETWORKS

4.1 INTRODUCTION TO INTERCONNECTION NETWORKS

There is a limit to the maximum speed obtainable from a computer based on a single processor. The closer one approaches this limit, the more rapidly the cost of such a computer rises. The crucial decision that must be made in the design of such a multiprocessor system is the level of parallelism, or in other words, the size of the subtasks into which the original task is split. When several processors are required to work cooperatively on a single task, one expects frequent exchanges of data among the several subtasks that comprise the main task. The amount of data, the frequency with which they are transmitted, the speed of their transmission, and the route that they take are all significant in effecting this intercommunication. The speed of transmission is a function of the hardware used and is not the point of discussion here. There have been many approaches that try to address this problem--that is, given these n processors, how to connect them in the most cost-effective manner.

A variable interconnection topology must have a smaller number of channels, and relatively easy routing rules. There are also such other considerations as fault tolerance: how to route data and recover gracefully in case a processor fails. With the range of possible applications in mind, the designer must choose the most cost-effective one for his purposes. Any evaluation of the performance of these schemes must be, to a certain extent, qualitative. It is instructive to examine, at least qualitatively, some of the important characteristics of these interconnection schemes[2].

4.1.1 NETWORK CHARACTERISTICS

In all these networks, it should be emphasized that improving one parameter might adversely affect some other parameters: what is sought is an optimization of the network.

Average Distance

One of the more important evaluative measures of an interconnection network is the average distance. This is the distance a message must travel, on an average, in the network. It is advantageous to make this as short as possible. The average distance is defined as:

$$\text{AvgDist} = \frac{dN_d}{N - 1}$$

Where N_d is the number of computers at a distance d links away, d is the diameter (maximum of the minimum distance between any two pairs of nodes), and N is the total number of processors.

For the regular network, i.e. those in which each computer is connected to the same number of processors, the average distance is a constant. For irregular networks, the formula will yield different results, depending upon the node from which d is measured. A network that has a low average distance may require an unreasonable number of communication ports for each processor. In order to distinguish these cases, a normalized average distance is defined for link-based structures:

NormAvgDist (link) = AvgDist * Ports/Processor

In the case of bus structures, the distance d is the number of buses a message has to cross on the way to its destination. Also, the number of processors tied on a single bus may create bottlenecks due to bus contention. To account for this, define the normalized distance for bus structure as the average distance weighted by the number of processors that may have access to a single bus.

NormAvgDist (bus) = AvgDist * Ports/Bus

Communication links

The total number of communication links in a network of given size is another useful measure. Clearly, among two networks, the one that has fewer connecting links is the more desirable, assuming all else is equal.

Routing Algorithm

When a message is to be routed from one computer to another, the route it must take is obtained from the routing algorithm. It is desirable that the routing algorithm be simple and not require complete knowledge of the entire network. In particular, it would be convenient merely to have the destination address. It is possible to obtain the exact--and preferably the shortest--sequence of computers the message must traverse.

Fault Tolerance

If one of the processors along this route were to be faulty, then a breakdown in communication would result, and this could make any further computation pointless. To preclude such a possibility, networks must be fault tolerant. Fault-tolerant networks have at least one redundant path between any two processors; these redundant paths are used in the case of a fault in a connecting channel. Another fault that is potentially more dangerous is the failure of a processor. Should such a fault arise, it is desirable that the system bypass this faulty computer in all future computations and remain functional although possibly impaired. This "graceful degradation" feature is desirable in certain critical areas, such as space and military applications.

Expansion Capability

Any large system must be capable of expansion in such a way that requires a complete rebuilding, with fresh demands on the number of communication ports of individual processors. Every time extra computers are added, it is less preferable to one that can be extended in a natural way, without major upheaval of the entire system.

4.2 TYPES OF NETWORKS

4.2.1 THE RING NETWORK

The ring structure is one of the simplest networks. The routing is simple and the structure has been well analyzed, mainly because, along with the star and tree networks, it is among the most popular of the topologies used in local area networks (LANs). The topology has also been used in dataflow machine architectures [2]. It consists of a number of processors connected in the form of a ring, i.e. each connected to its two neighbors. Although most LAN topologies use a unidirectional ring (i. e. one in which data flows in one direction only around the ring), because of its obvious problem of poor fault tolerance, a bi-directional ring will be assumed.

Average Distance The normalized average distance is $(N + 1)/8$, as there are two ports on each processor [2]. This linear relationship between the total number of processors and average distance means that the average distance of the ring network increases as the total number of processors increases.

Communication Links The total number of communication links is N.

Routing Algorithm The routing algorithm is relatively straightforward because of the simplicity of the network. In a ring network a single processor is connected with two other processors, a message is to be routed from one processor to another, the route it must take is obtained from the routing algorithm. It is simplest for unidirectional rings and only slightly more involved for bi-directional rings.

Fault tolerance The fault tolerance of the ring structure is questionable. If any node in a unidirectional ring fails, it may render the entire system nonfunctional. In a bi-directional ring the failure of two nodes will cause the same result.

Expansion capability The ring network is obviously one of the simplest to expand.

4.2.2 THE CUBE CONNECTED NETWORK

This network connects 2^k computers (k is an integer) in such a way that groups of 2^r (r is the smallest integer such that r + 2 >= k) are interconnected so as to form a (k - r) dimension cube. Each processor has a k-bit address that is expressed as a pair (l, p) of integers, l having (k - r) bits, and p having r bits. There are three ports called F (Forward), B (Backward) and L (Lateral) provided on each processor.

Average distance The average distance for the Cube Connected Configuration is obtained as the product of the average distance of a subgroup of 2^r processors (which form a ring) and the main (k - r) cube network. The number of ports in each computer is three, so the normalized distance is simply the average distance times three.

Communication links The total number of communication links is at most (3/2)N, where N is the total number of nodes in the network.

Routing algorithm and fault tolerance When a node is faulty, an alternative path may be found with ease because of the simple routing algorithm.

Expansion capability Because of the cube structure, expansion must be in powers of two. The system must be restructured, i.e. as shown in Figure 4.1 a 4-cube network can be constructed from two 3-cube network by using $8 = 2^3$ extra edges between corresponding vertices at the corner positions [1].

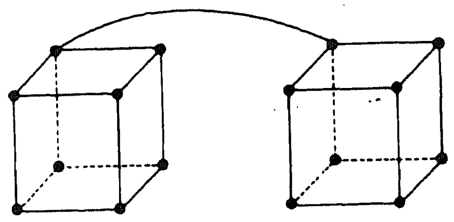


Figure 4.1 The construction of two 4-cube network

4.2.3 THE ALPHA NETWORK

This is a generalized hypercube structure.

Average distance The average distance is given by

$$\text{AvgDist (alpha)} = \frac{D(W - 1)W^{N-1}}{N - 1}$$

The number of ports on each processor is given by

$$\text{Ports} = D(W - 1)$$

where D is the distance between nodes of the alpha network and W is the total number of processors in each dimension.

Communication links The total number of communication links is $\text{Links(alpha)} = N * \text{Ports}/2$ and $N = W^D$

Routing algorithm and fault tolerance A simple routing algorithm is designed for a alpha connected network. Because of the several redundant paths that exist, this network is highly fault tolerant[2].

Expansion Capability Since this network is a generalized cube network, expansion is not easy as the number of ports is dependent upon network size. Unlike cube networks, however, any nonprime value of N can be accommodated.

In the same way one can derive the average distance and the analysis of communication link, routing algorithm, and fault tolerance for the rest of the network structures (i.e. Hyper tree network, multitree structure, and beta network).

CHAPTER 5

DESIGN AND FUNCTION DESCRIPTION

5.1 DATABASE DESIGN

Multiprocessing System Design - Tutor is designed with the Informix Relational database. The name of the database which stores all the information about the microprocessors is MULPROST. Tables and indices created in this database are explained below. A functional description is also given for each column in the database. All the 4GL modules are designed using the MULPROST database and its tables. The 4GL module is compiled using C4GL which converts the 4GL module into a C program; this C program is then compiled into executable form using the Microsoft C compiler and linker.

An Informix-4GL program consists of a series of English-like statements that obey a well-defined syntax. Informix-4GL deals with a number of different kinds of objects. These includes local and global program variables, constants, screen forms, functions and reports. A typical sequence in an Informix-4GL program consists of selecting a database, opening a form with or without window and allowing the user to select menu options to enter or edit data through the fields defined in the form. Multiple forms related to different actions are handled sequentially within the same program.

The following section describes the creation of tables and indices or indices for a table.

5.2 TABLES

The multiprocessor performance table, Table 5.1, holds information about the performance of a microprocessor. It contains performance factors such as the total number of basic instructions, direct addressing range, number of addressing modes, basic clock frequency, primitive data types, data structure and operating system support.

FIELD NAME	DESCRIPTION
micropro char(11),	Microprocessor name
bascinst char(25),	basic Instructions
dar char(20),	Direct Addressing Range
noofadmo char(40),	Number of Addressing Modes
bcf char(10),	Basic Clock Frequency
primdata char(20),	Primitive Data Types
datastrc char(30),	Data structure
primcont char(30),	Primitive Control
contstrc char(30),	Control Structure
ossupport char(20),	Operating System Support
gpr char(20)	General Purpose Register

TABLE 5.1 Multiprocessor performance

The grant command allows access of this table to all the users.

The multiprocessor characteristic table, Table 5.2, holds all the characteristics for a given multiprocessor configuration. It keeps track of characteristic names within the table. It has 12 description lines and two remark lines.

FIELD NAME	DESCRIPTION
charname char(50),	Charcteristic Name
chardesc1 char(54),	Line 1 of Description
chardesc2 char(54),	Line 2 of Description
chardesc3 char(54),	Line 3 of Description
chardesc4 char(54),	Line 4 of Description
chardesc5 char(54),	Line 5 of Description
chardesc6 char(54),	Line 6 of Description
chardesc7 char(54),	Line 7 of Description
chardesc8 char(54),	Line 8 of Description
chardesc9 char(54),	Line 9 of Description
chardesc10 char(54),	Line 10 of Description
chardesc11 char(54),	Line 11 of Description
chardesc12 char(54),	Line 12 of Description
remark1 char(35),	Line 1 for Remarks
remark2 char(35)	Line 2 for Remarks

TABLE 5.2 Multiprocessor characteristics

The grant command will allow access of this table to all the users.

The network characteristic table, Table 5.3, contains information about all the networks and their characteristics.

create table netwchar

FIELD NAME	DESCRIPTION
nettype char(20),	Type of Network
avgdist decimal(7,2),	Average Distance
diameter smallint,	Maximum of minimum distance between two nodes.
commlink smallint,	Communication Link
computers smallint,	Total Number of COmputers
routalgo char(25),	Routing Algorithm
algodes1 char(50),	Algorithm Description Line 1
algodes2 char(50),	Algorithm Description Line 2
faultolr smallint	Fault Tolerance

TABLE 5.3 Network charcteristic

The grant command will allow access of this table to all the users. It keeps track of network characteristics and information about routing algorithms for different networks. The user can add, update or delete information from this table for any type of network.

The design table, Table 5.4, stores all the created designs. This table holds the information about all the designs created during the design process.

FIELD NAME	DESCRIPTION
designno serial not null	Design Number
description char(40),	Description
micropro char(11),	Microprocessor
multtype char(20),	Multiprocessor Configuration
nettype char(20),	Network Type
desdesc1 char(60),	Design description Line 1
desdesc2 char(60),	Design description Line 2
desdesc3 char(60),	Design description Line 3
desdesc4 char(60),	Design description Line 4
appdesc1 char(60),	Application Description Line 1
appdesc2 char(60)	Application Description Line 2

TABLE 5.4 Multiprocessor design

This design number automatically becomes a unique index for this table because of the type serial. A unique index is created on this table on the designno field to increase the record search speed. The grant command will allow access of this table to all the users.

The configuration table, Table 5.5, stores all the different configurations of multiprocessor.

FIELD NAME	DESCRIPTION
conname char(30),	Configuration Name
confdes1 char(50),	Configuration Description Line 1
confdes2 char(50),	Configuration Description Line 2
confdes3 char(50),	Configuration Description Line 3
confdes4 char(50),	Configuration Description Line 4
confdes5 char(50),	Configuration Description Line 5
confdes6 char(50),	Configuration Description Line 6
confdes7 char(50),	Configuration Description Line 7
confdes8 char(50)	Configuration Description Line 8

TABLE 5.5 Multiprocessor configuration

A unique index is created for this table on conname field to increase the record search speed. The grant command will allow access of this table to all the users.

5.3 MODULES

The 4gl functions and related modules used in the system are described below.

add_date()

add_date.4gl

This function is used to display a date at the line no 3 of the form. It displays the date in the format dd mmm yy, e.g. 16 Sep

89.

get_config()

confhelp.4gl

This function is a help function to get the system configuration from the reference table.

deletwin() returning IS_DEL

deletwin.4gl

This function can be used to verify whenever the user chooses to delete a record from any table. It gives a RING MENU option of "YES" or "NO" to choose from. If "YES" is chosen, then IS_DEL is set to TRUE; if "NO" is chosen, then IS_DEL is set to FALSE. After a particular choice is made, the function returns IS_DEL. Any program calling this function can work on the value returned by IS_DEL. For information sake,the window delwin is displayed at 21,3 with 2 rows and 30 columns.

mainmenu()

mainmenu.4gl

This is the main function of the whole system. It displays the main menu of the system and it is used as a driver for all functions. This is the only one which contains main; all other files are functions only. Here, the defer interrupt is used so that when the user presses (CONTROL-C), it sets int_flag to true, and based on that one can decide whether the user aborted the process or selected a row.

menu1()

menu1.4gl

This function is used as a submenu of a main menu. When the user selects item 1 from the main menu, then this menu is displayed.

menu3()

menu3.4gl

This function is used as the submenu of the main menu. The basic functionality is based on the user selection from the menu. It calls the function corresponding to the user's choice.

menu4()

menu4.4gl

This function is used as a submenu of the main menu. The basic functionality is based on the user selection from the menu. It calls the function corresponding to the user's choice.

menu5()

menu5.4gl

This function is used as a submenu of the main menu. The basic functionality is based on the user selection from the menu. It calls the function corresponding to the user's choice.

get_mic()

michelp.4gl

This function is used to get the help for the name of the microprocessor. It will list the name of the microprocessor from the reference table; one can select the microprocessor by pressing the ESC key.

get_chars()

multchlp.4gl

This function is used as a help function to get the name of the characteristics. The data is fetched from the characteristic reference table and is displayed as a program array on the screen.

multperf()

multperf.4gl

This function is used as a maintenance of the performance of the multiprocessor. The basic logic is the input array of some elements.

mult_chr()

mult_chr.4gl

This function is used to maintain the characteristics of the multiprocessor.

get_net()

nethelp.4gl

This function is a help function to get the network name from the reference table.

net_char()

netwchar.4gl

This module is used to maintain the characteristics of a network reference table. It is also used as the view in the main menu.

prt_char()

prt_char.4gl

This function is used to print the multiprocessor characteristics in report form so that one can keep the readable copy.

prt_net()

prt_net.4gl

This function is use to print the multiprocessor network configuration in report form so that one can keep the readable copy.

5.3 FORMS

The forms related to the system are described below:

Mainmenu.per: This form is used with the main menu module to display the various items of selection.

Menu1.per: This is a submenu from the main menu.

Menu3.per: This is also a submenu from the main menu.

Menu4.per: This is a maintain reference table screen from the main screen.

Menu5.per: This is a report screen from the main screen.

Multchar.per: This is the Multiprocessor Characteristics maintenance screen from the maintain reference table.

Design.per: This form is used to get the design data from the user.

Netwchar.per: This function is used to maintain the network characteristics from the maintain reference table.

Michelp.per: This is a microprocessor selection help used as an interface between the maintain screen and menu selection.

Nethelp.per: This is the same as michelp.per but used as a help to choose a network name.

Configsm.per: A system configuration maintenance screen.

Confhelp.per: A help screen that acts as an intermediate screen between the maintenance and menu.

CHAPTER 6

SYSTEM OPERATION

6.1 USER AND REFERENCE TIPS FOR OPERATION

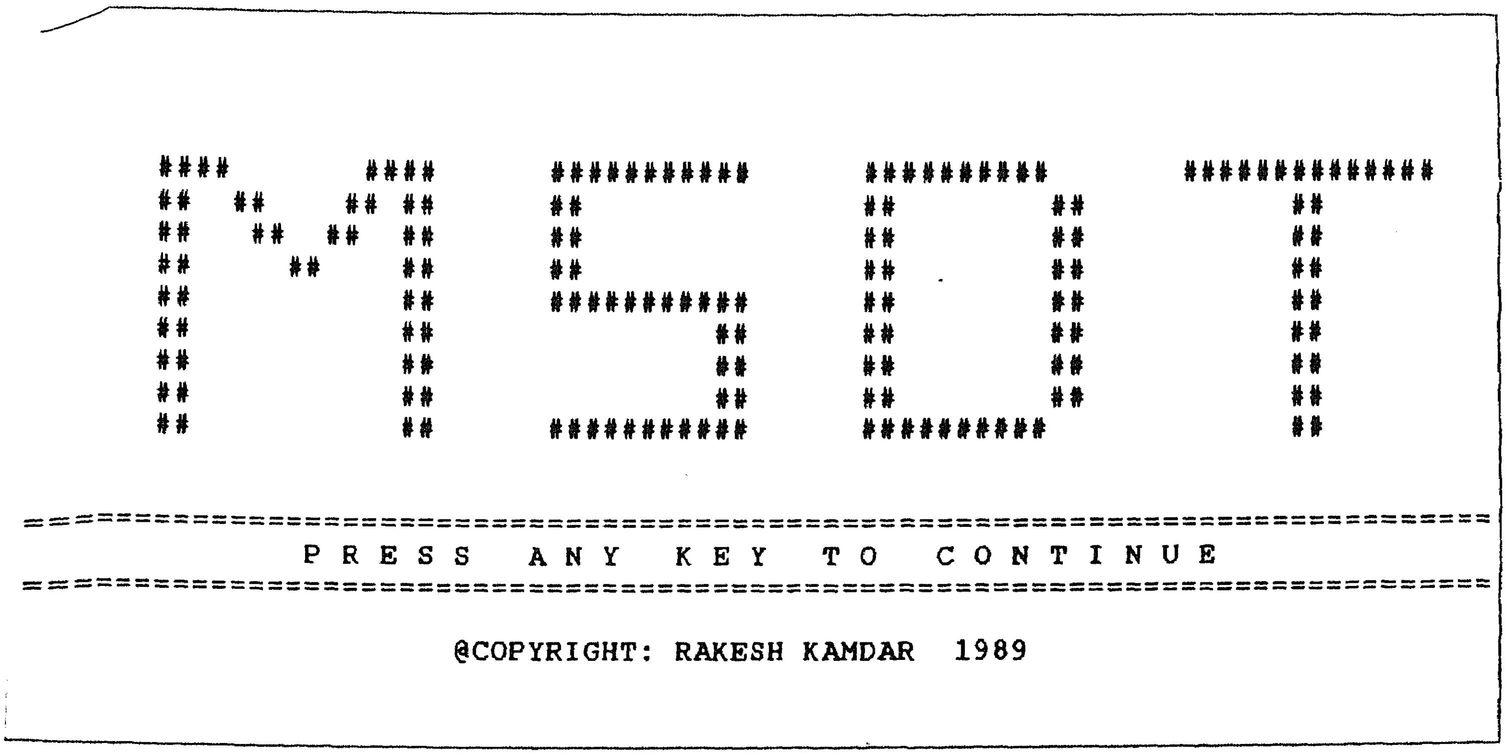
The basic operation of this system is menu driven. All menus are very user friendly, and display most of the messages on the screen while the user is in data entry mode.

Some of the standards of this system are as follows:

1. [ESC] is always an accept key, i.e. if you press [ESC] then the corresponding action will be taken. e.g. If the user is in ADD mode and after the complete data entry, if the user presses [ESC], the data will be inserted in the database.
2. [CONTROL-C] is always an abort key, i.e. if you presses [CONTROL-C] while you are in help function the corresponding row will not be selected and the control passes back to the menu.
3. [CONTROL-J] will be used as an up arrow key.
4. [CONTROL-K] will be used as a down arrow key.
5. [CONTROL-H] will be used as back space.
6. [CONTROL-L] will be used as the right arrow.
7. All the menu forms will be displayed within the window.
8. Always enter the number corresponding to menu selection.

6.2 MENU STRUCTURE

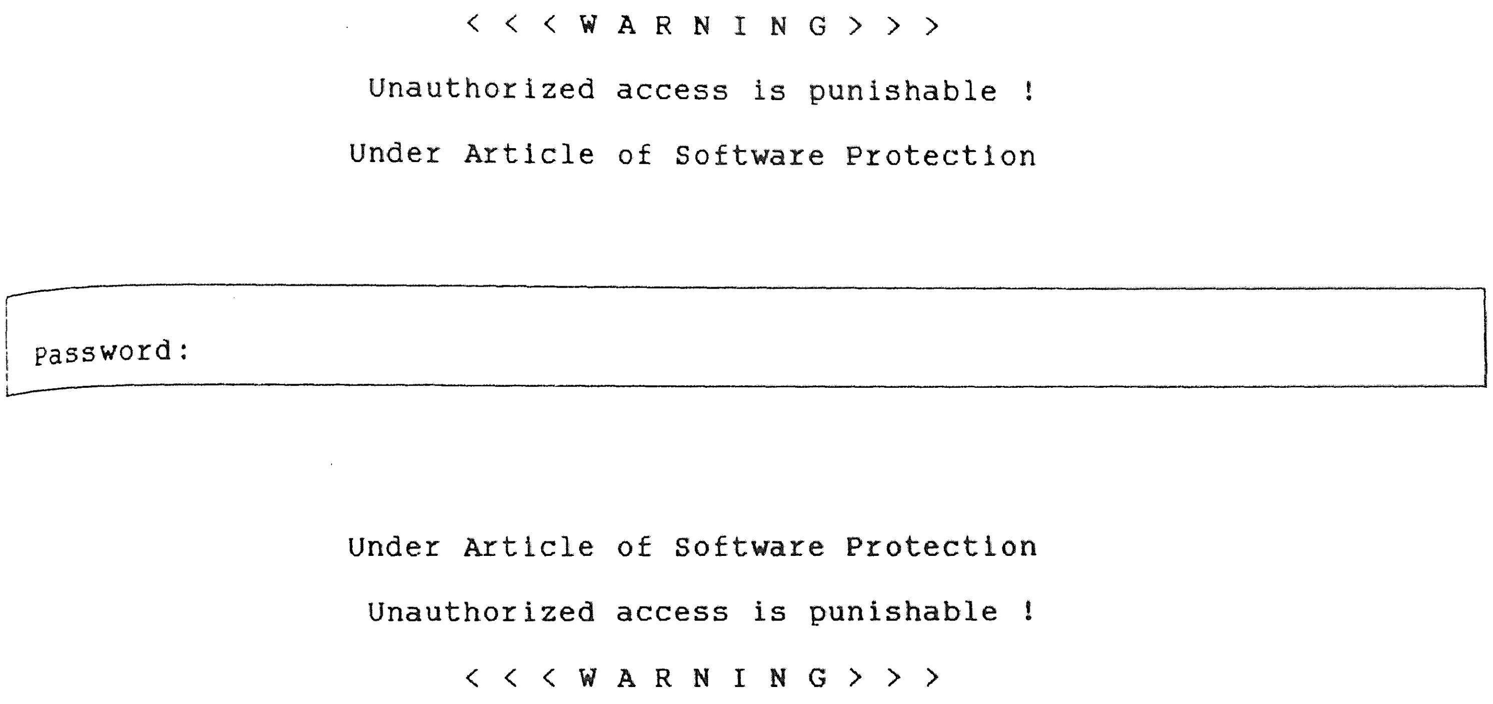
Figures 6.1 thru 6.14 on the following pages show the hierarchical menu structure. These figures are created using the print screen command of the computer while the system is running.



```
####       ####      ###########      ##########        ##############
##  ##    ## ##      ##               ##        ##            ##
##   ##  ##  ##      ##               ##        ##            ##
##     ##    ##      ##               ##        ##            ##
##           ##      ###########      ##        ##            ##
##           ##               ##      ##        ##            ##
##           ##               ##      ##        ##            ##
##           ##               ##      ##        ##            ##
##           ##      ###########      ##########              ##

================================================================================
              P R E S S   A N Y   K E Y   T O   C O N T I N U E
================================================================================

                      @COPYRIGHT: RAKESH KAMDAR  1989
```

Figure 6.1 Main title screen

Figure 6.1 shows the main title screen when user first enters into the system. This screen allows the user to press any key to continue.



< < < W A R N I N G > > >

Unauthorized access is punishable !

Under Article of Software Protection

Password:

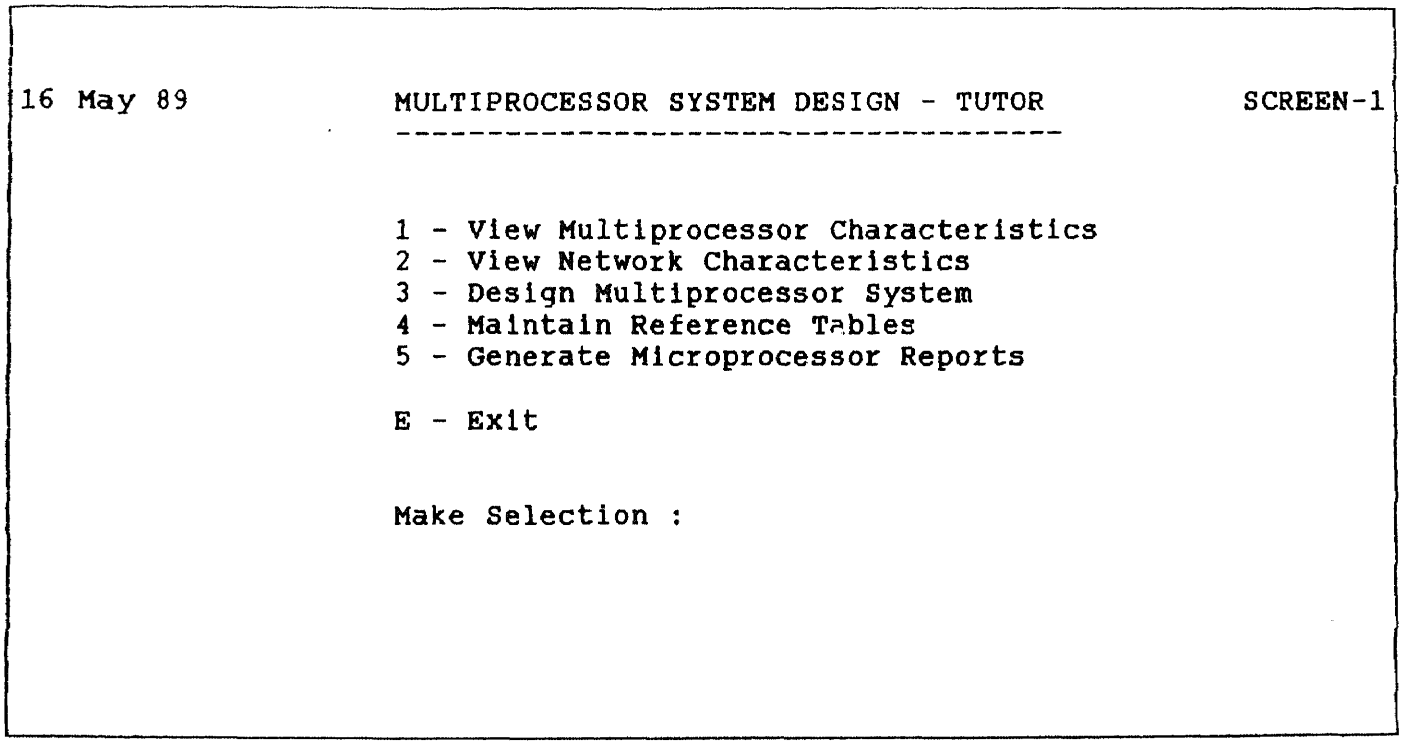
Under Article of Software Protection

Unauthorized access is punishable !

< < < W A R N I N G > > >

Figure 6.2 Password entry screen

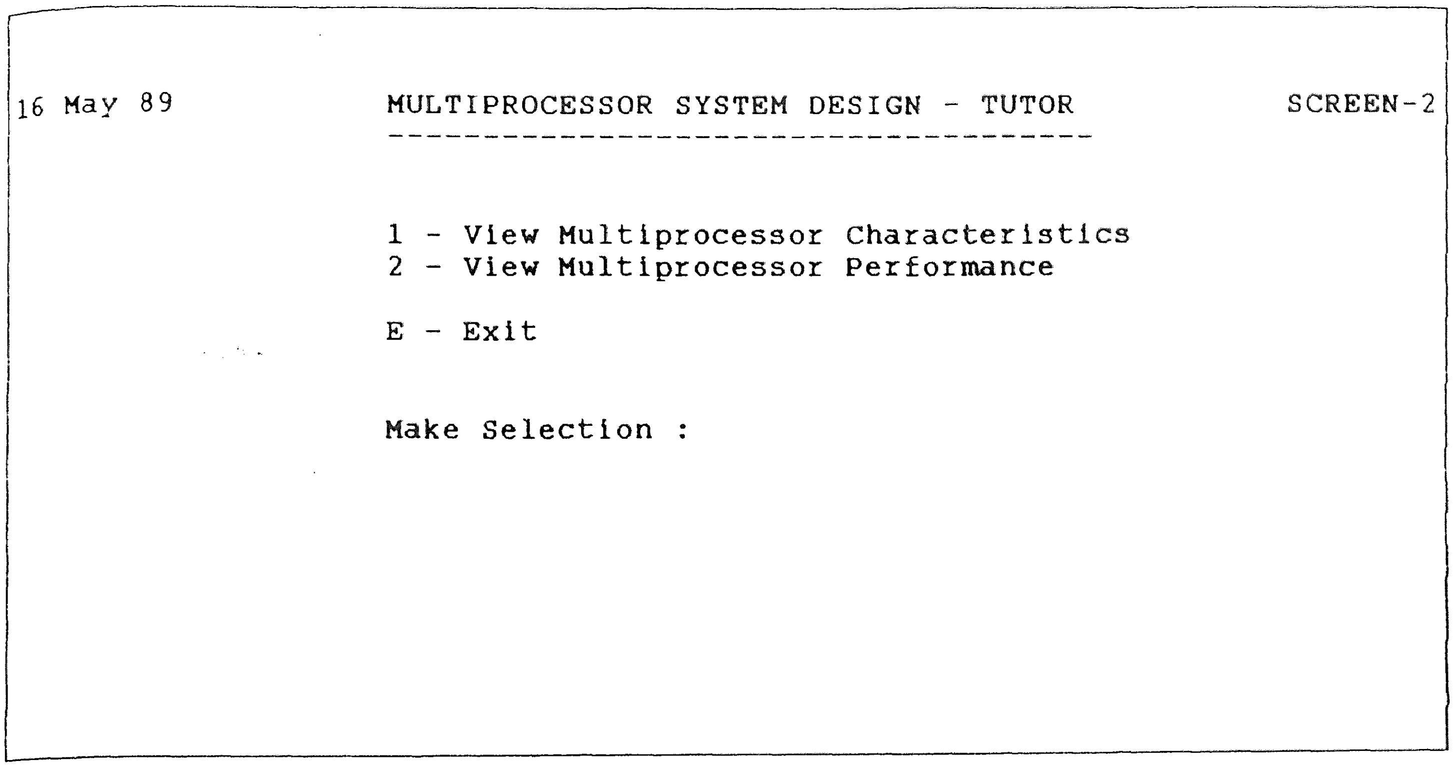
Figure 6.2 shows the password entry screen. This screen allows the user to enter a valid password to get entry into the first menu screen.



```
16 May 89            MULTIPROCESSOR SYSTEM DESIGN - TUTOR          SCREEN-1
                     ------------------------------------

                     1 - View Multiprocessor Characteristics
                     2 - View Network Characteristics
                     3 - Design Multiprocessor System
                     4 - Maintain Reference Tables
                     5 - Generate Microprocessor Reports

                     E - Exit


                     Make Selection :
```

Figure 6.3 Main menu

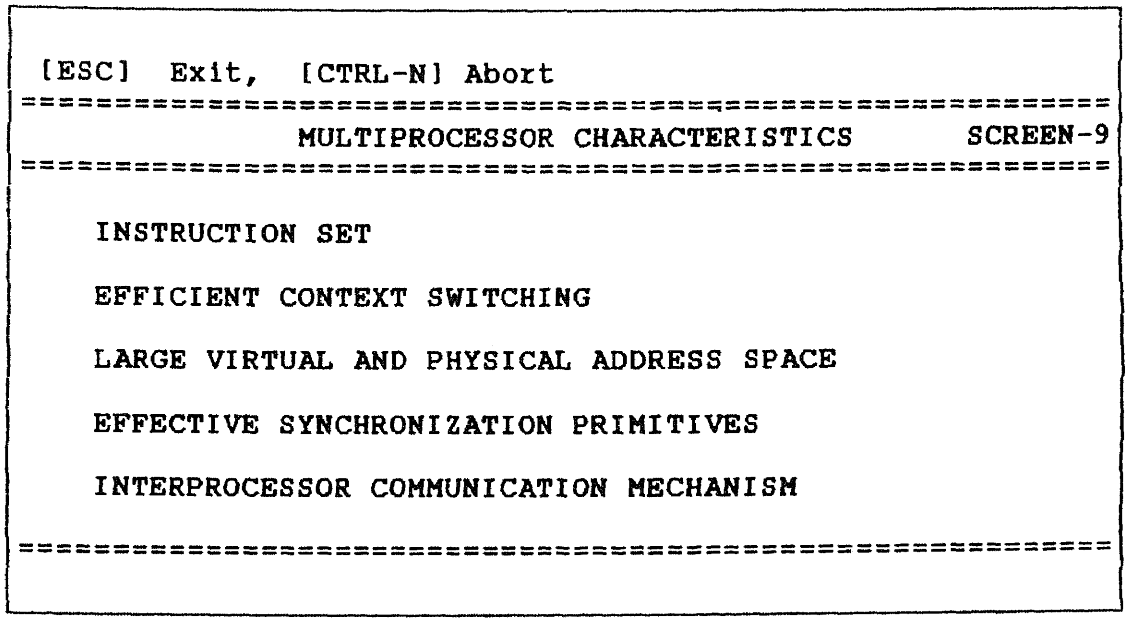
Figure 6.3 shows the main menu of the system. The user can select any of the items by pressing the number corresponding to that item.



```
16 May 89           MULTIPROCESSOR SYSTEM DESIGN - TUTOR          SCREEN-2
                    ------------------------------------

                    1 - View Multiprocessor Characteristics
                    2 - View Multiprocessor Performance

                    E - Exit


                    Make Selection :
```

Figure 6.4 Submenu 1

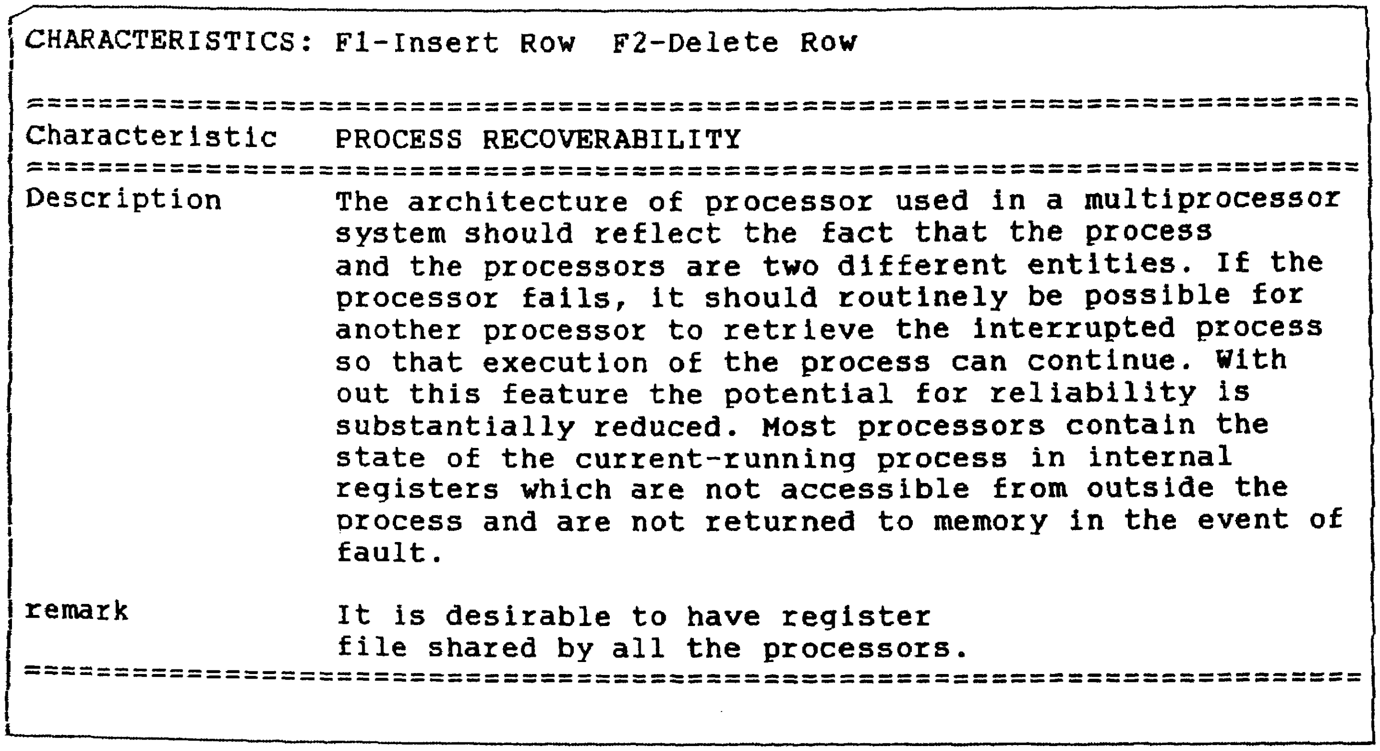
Figure 6.4 shows the submenu for the selection number 1 from the main menu. The user can view multiprocessor characteristics or multiprocessor performance by selecting appropriate number from this submenu.



```
[ESC]  Exit,  [CTRL-N] Abort
==========================================================
              MULTIPROCESSOR CHARACTERISTICS      SCREEN-9
==========================================================

    INSTRUCTION SET

    EFFICIENT CONTEXT SWITCHING

    LARGE VIRTUAL AND PHYSICAL ADDRESS SPACE

    EFFECTIVE SYNCHRONIZATION PRIMITIVES

    INTERPROCESSOR COMMUNICATION MECHANISM

==========================================================
```

Figure 6.5 Multiprocessor characteristics

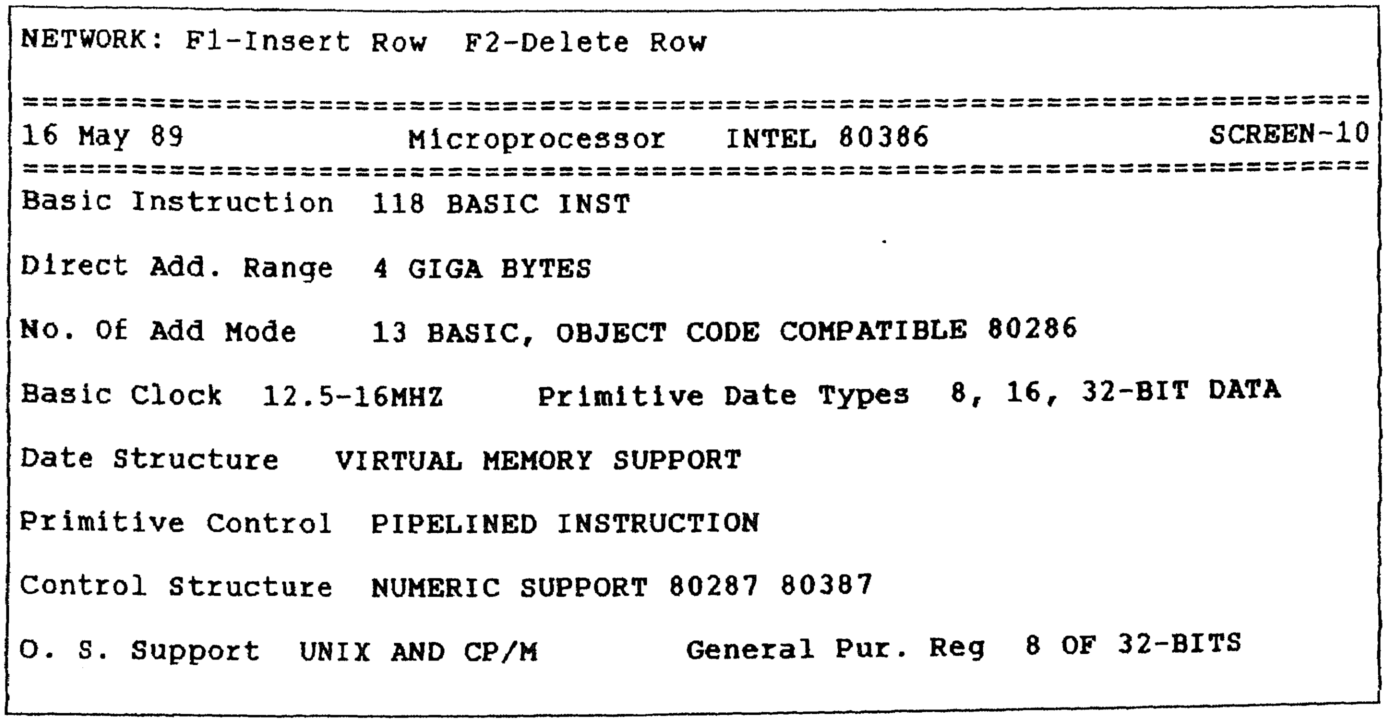
Figure 6.5 lists multiprocessor characteristics available in the system. The user can view these characteristics in detail by pressing the ESCAPE key.



```
CHARACTERISTICS: F1-Insert Row  F2-Delete Row

========================================================================
Characteristic   PROCESS RECOVERABILITY
========================================================================
Description      The architecture of processor used in a multiprocessor
                 system should reflect the fact that the process
                 and the processors are two different entities. If the
                 processor fails, it should routinely be possible for
                 another processor to retrieve the interrupted process
                 so that execution of the process can continue. With
                 out this feature the potential for reliability is
                 substantially reduced. Most processors contain the
                 state of the current-running process in internal
                 registers which are not accessible from outside the
                 process and are not returned to memory in the event of
                 fault.

remark           It is desirable to have register
                 file shared by all the processors.
========================================================================
```

FIgure 6.6 Detail description of characteristics

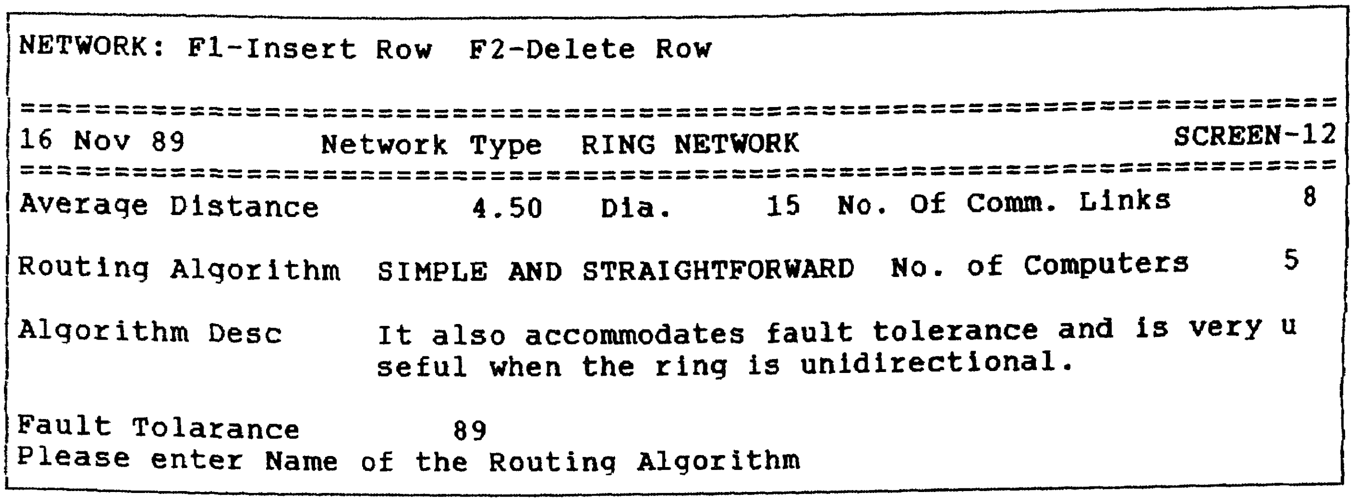
Figure 6.6 shows the detailed description of the process recoverability of multiprocessor. The user can scroll through all the characteristics to view in detail.



```
NETWORK: F1-Insert Row  F2-Delete Row

============================================================================
16 May 89             Microprocessor   INTEL 80386               SCREEN-10
============================================================================
Basic Instruction  118 BASIC INST

Direct Add. Range  4 GIGA BYTES

No. Of Add Mode    13 BASIC, OBJECT CODE COMPATIBLE 80286

Basic Clock  12.5-16MHZ     Primitive Date Types  8, 16, 32-BIT DATA

Date Structure   VIRTUAL MEMORY SUPPORT

Primitive Control  PIPELINED INSTRUCTION

Control Structure  NUMERIC SUPPORT 80287 80387

O. S. Support  UNIX AND CP/M        General Pur. Reg  8 OF 32-BITS
```

Figure 6.7 Microprocessor characteristics

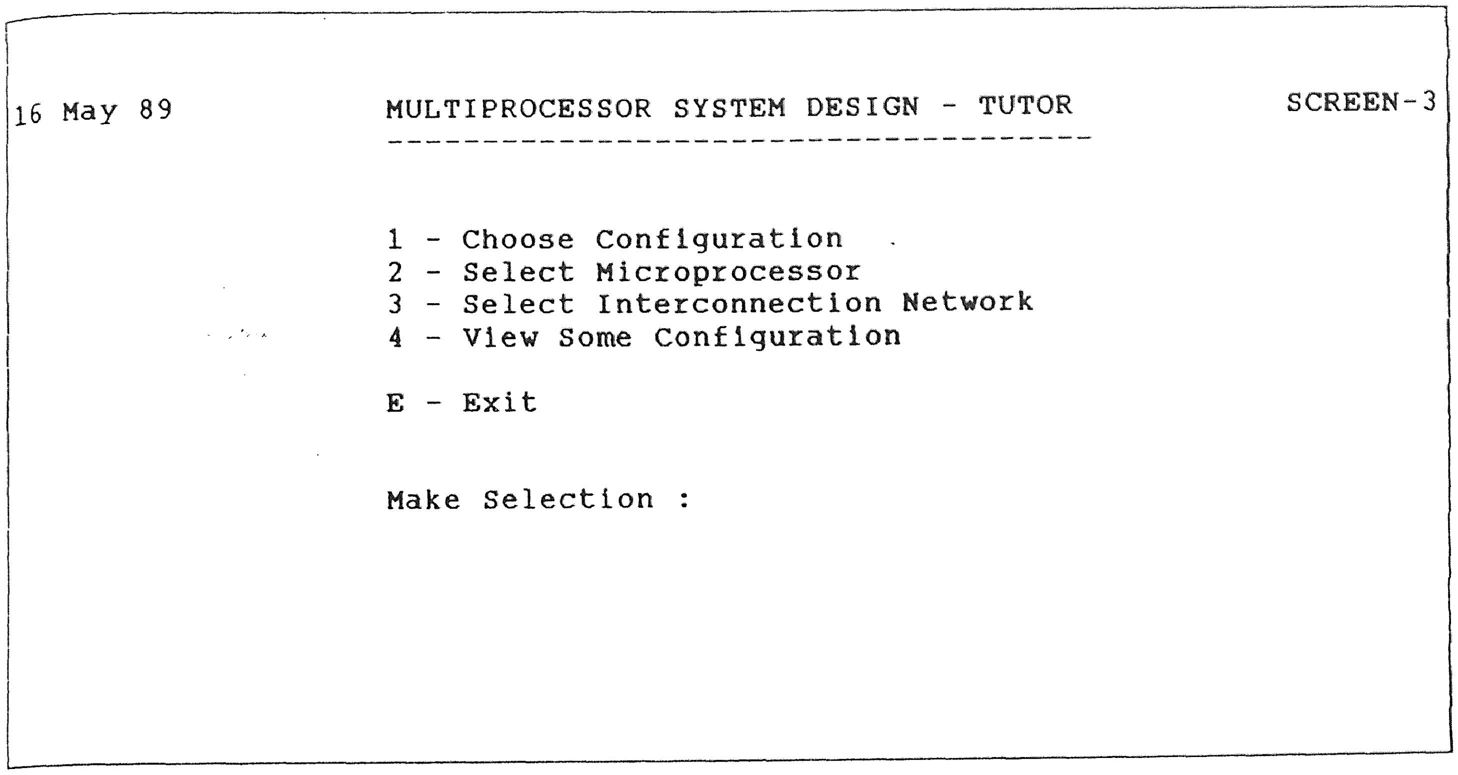
Figure 6.7 shows the characteristics of the Intel 80386 microprocessor.



```
NETWORK: F1-Insert Row  F2-Delete Row

==========================================================================
16 Nov 89       Network Type  RING NETWORK                        SCREEN-12
==========================================================================
Average Distance        4.50   Dia.     15  No. Of Comm. Links          8

Routing Algorithm  SIMPLE AND STRAIGHTFORWARD  No. of Computers       5

Algorithm Desc     It also accommodates fault tolerance and is very u
                   seful when the ring is unidirectional.

Fault Tolarance        89
Please enter Name of the Routing Algorithm
```

Figure 6.8 Network type

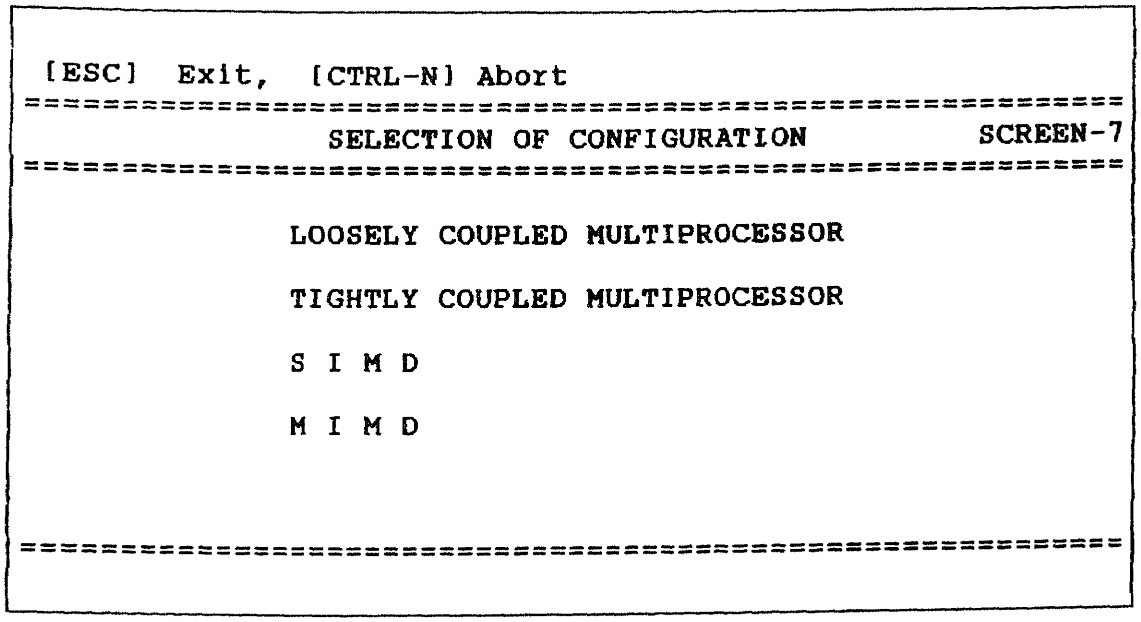
Figure 6.8 shows all the parameters of a ring network. The user can scroll through the details of other network configurations.



```
16 May 89          MULTIPROCESSOR SYSTEM DESIGN - TUTOR          SCREEN-3
                   ------------------------------------

                   1 - Choose Configuration
                   2 - Select Microprocessor
                   3 - Select Interconnection Network
                   4 - View Some Configuration

                   E - Exit


                   Make Selection :
```

Figure 6.9 Submenu 2

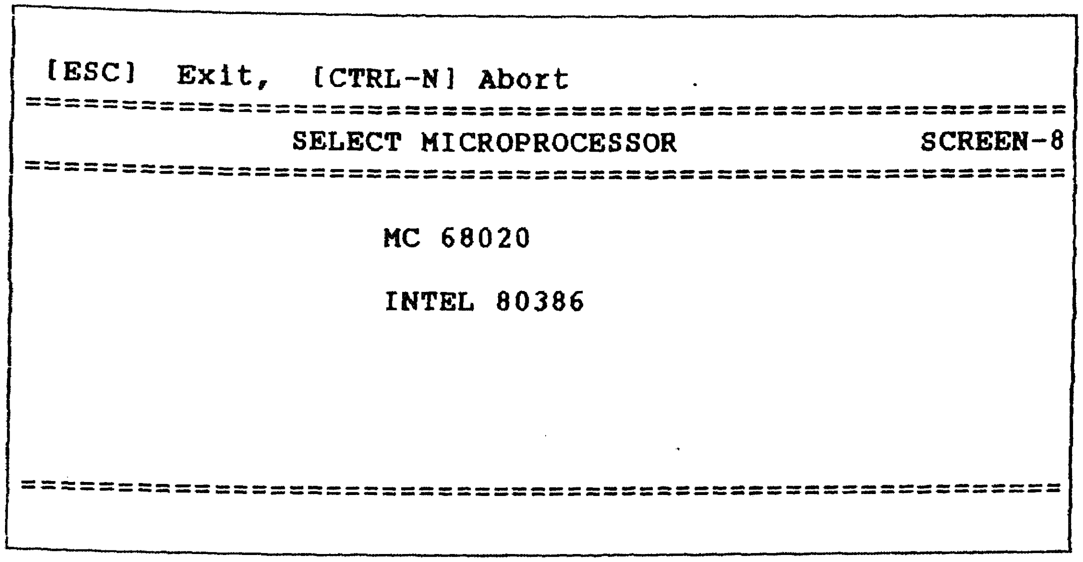
Figure 6.9 shows the submenu for item number 3 of the main menu. The user can select any item by pressing a number corresponding to that item. The user can exit from this submenu by pressing 'E'.



```
 [ESC]  Exit,  [CTRL-N] Abort
============================================================
                SELECTION OF CONFIGURATION         SCREEN-7
============================================================

              LOOSELY COUPLED MULTIPROCESSOR

              TIGHTLY COUPLED MULTIPROCESSOR

              S I M D

              M I M D


============================================================
```

Figure 6.10 Configuration selection

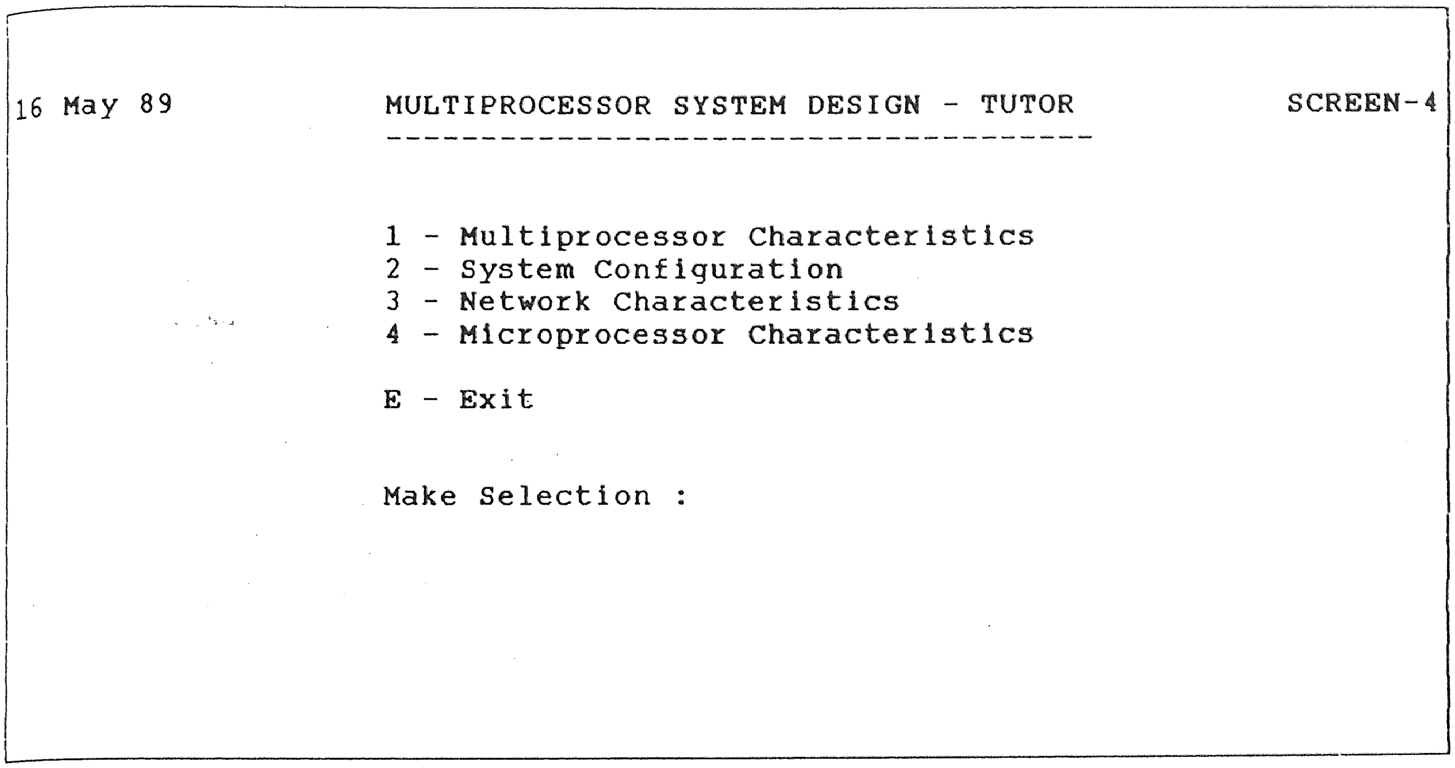
Figure 6.10 shows the various configurations of multiprocessor systems. The user can view these configurations in detail by pressing ESCAPE key.



```
[ESC]  Exit,  [CTRL-N] Abort
=========================================================
              SELECT MICROPROCESSOR              SCREEN-8
=========================================================

                   MC 68020

                   INTEL 80386



=========================================================
```

Figure 6.11 Microprocessor selection

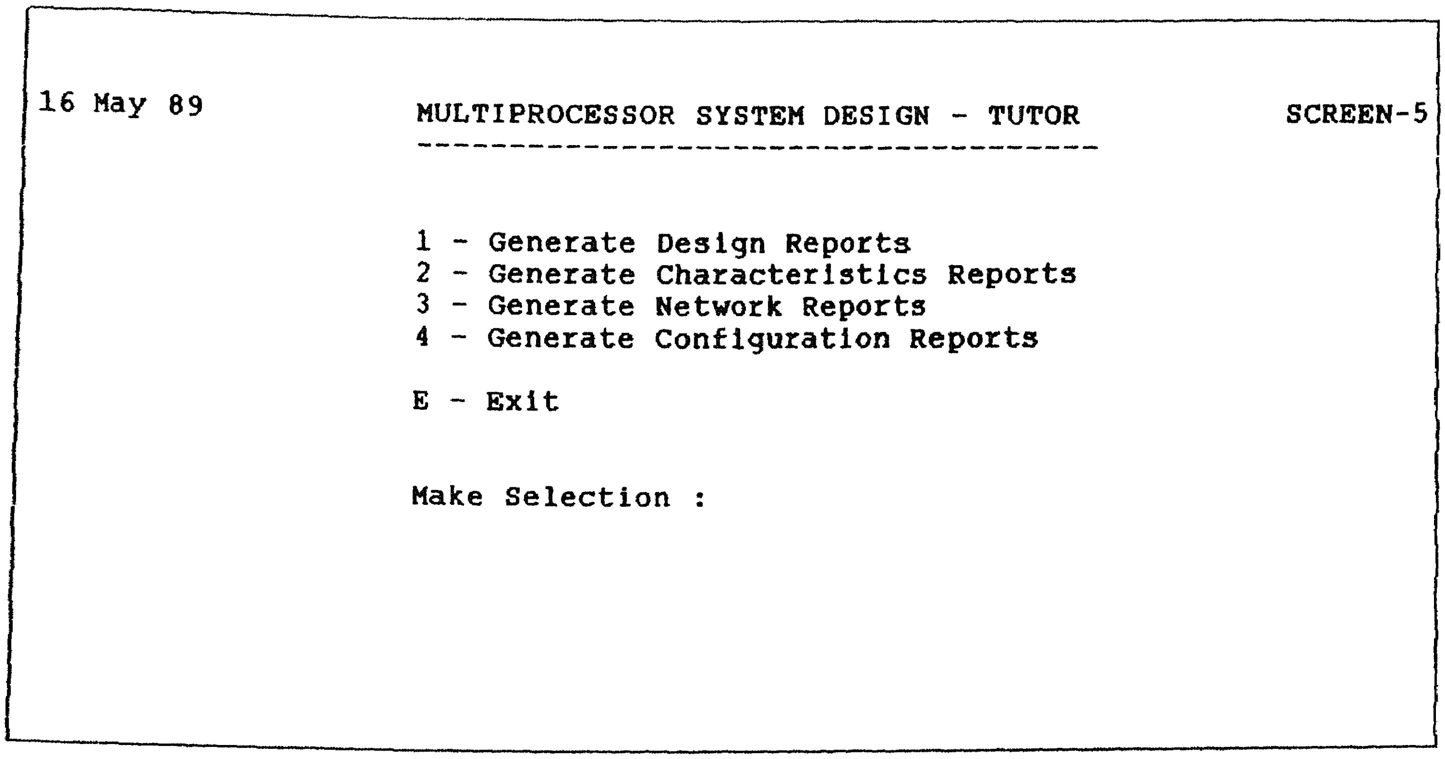
Figure 6.11 lists the microprocessors currently available in the system.



```
16 May 89              MULTIPROCESSOR SYSTEM DESIGN - TUTOR             SCREEN-4
                       ------------------------------------

                       1 - Multiprocessor Characteristics
                       2 - System Configuration
                       3 - Network Characteristics
                       4 - Microprocessor Characteristics

                       E - Exit

                       Make Selection :
```

Figure 6.12 Submenu 3

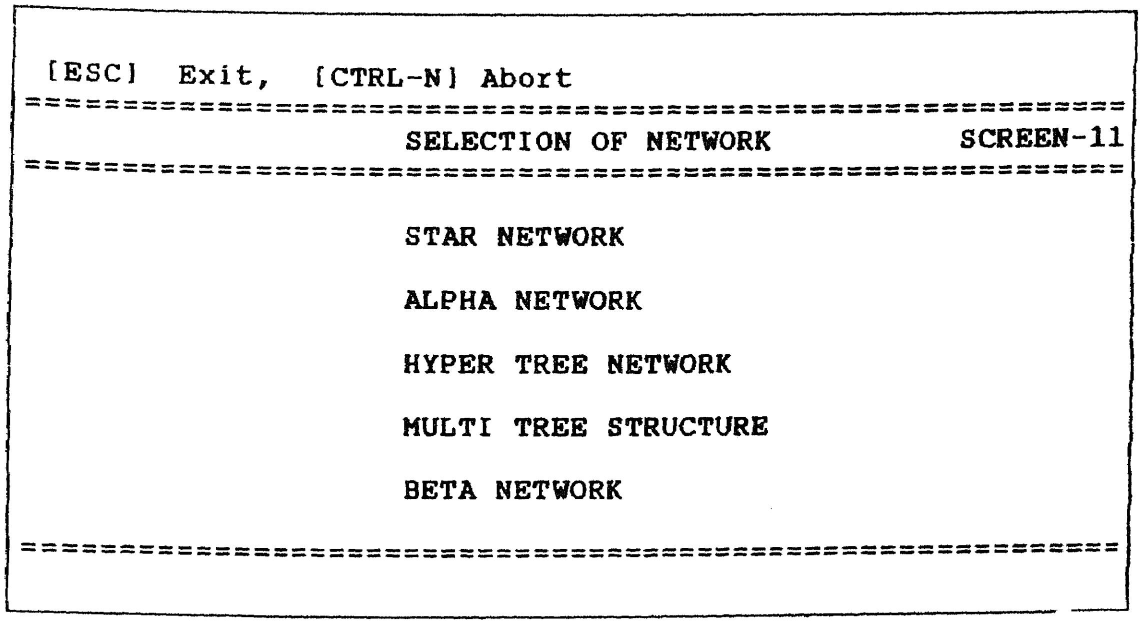
Figure 6.12 shows the submenu for the selection of item number 4 of the main menu. These are the reference tables in the database, which store all the information of multiprocessors and microprocessors.



```
16 May 89              MULTIPROCESSOR SYSTEM DESIGN - TUTOR            SCREEN-5
                       ------------------------------------

                       1 - Generate Design Reports
                       2 - Generate Characteristics Reports
                       3 - Generate Network Reports
                       4 - Generate Configuration Reports

                       E - Exit


                       Make Selection :
```

Figure 6.13 Submenu 4

Figure 6.13 shows the submenu for the selection of item number 5 of the main menu. These are the reports from the database, which print all the information of multiprocessors and microprocessors reference tables.



```
 [ESC]  Exit,  [CTRL-N] Abort
===========================================================
                    SELECTION OF NETWORK          SCREEN-11
===========================================================

                    STAR NETWORK

                    ALPHA NETWORK

                    HYPER TREE NETWORK

                    MULTI TREE STRUCTURE

                    BETA NETWORK

===========================================================
```

Figure 6.14 Network selection

Figure 6.14 lists all the network configurations available in the database. The user can view any of the network configuration by pressing the ESCAPE key.

CHAPTER 7

CONCLUSIONS AND SUGGESTIONS

7.1 CONCLUSIONS

Multiprocessing System Design - Tutor is designed and implemented as a helping tool to system design engineers. MSDT stores and maintains information related to multiprocessor system design, which includes multiprocessor system requirements, microprocessor characteristics and interconnection network configurations and their performance factors. This information is presented to the user via screen building utility of Informix-4GL.

7.2 SUGGESTIONS

This system needs development in the area of evaluation of microprocessor characteristics and memory module interface. Currently, it contains all the information about the MC68020 and Intel 80386. For further development it needs information about other microprocessors. The Informix Rapid Development Tool is commercially available, so one can use that for further 4GL code development.

One can also make this system work for new microprocessors like the Intel 80486 and the MC68030. This system has very good potential to grow into a large multiprocessing system design tool.

APPENDIX 1

INFORMIX FORMS

```
database formonly
screen
{
          MULTIPROCESSOR SYSTEM DESIGN - TUTOR           SCREEN-1
         --------------------------------------


         1 - View Multiprocessor Characteristics
         2 - View Network Characteristics
         3 - Design Multiprocessor System
         4 - Maintain Reference Tables
         5 - Generate Microprocessor Reports

         E - Exit


       Make Selection : [a]
}
attributes
a=formonly.s_choice type char, upshift, autonext, reverse;
instructions
delimiters "  "
end
```

```
database formonly
screen
{
         MULTIPROCESSOR SYSTEM DESIGN - TUTOR          SCREEN-2
         -------------------------------------


       1 - View Multiprocessor Characteristics
       2 - View Multiprocessor Performance

       E - Exit


       Make Selection : [a]
}
attributes
a=formonly.s_choice type char, upshift, autonext, reverse;
instructions
delimiters "  "
end
```

```
database formonly
screen
{
         MULTIPROCESSOR SYSTEM DESIGN - TUTOR           SCREEN-1
         ------------------------------------


                    1 - Linear Array
                    2 - Ring
                    3 - Star
                    4 - Tree
                    5 - Near-Neighbour mesh
                    6 - Systolic Array
                    7 - Completely Connected
                    8 - Cube

                    E - Exit


                    Make Selection : [a]
}
attributes
a=formonly.s_choice type char, upshift, autonext, reverse;
instructions
delimiters "  "
end
```

```
database formonly
screen
{
         MULTIPROCESSOR SYSTEM DESIGN - TUTOR          SCREEN-3
         ------------------------------------


              1 - Choose Configuration
              2 - Select Microprocessor
              3 - Select Interconnection Network
              4 - View Some Configuration

              E - Exit


              Make Selection : [a]
}
attributes
a=formonly.s_choice type char, upshift, autonext, reverse;
instructions
delimiters "  "
end
```

```
database formonly
screen
{
         MULTIPROCESSOR SYSTEM DESIGN - TUTOR          SCREEN-4
         -------------------------------------


          1 - Multiprocessor Characteristics
          2 - System Configuration
          3 - Network Characteristics
          4 - Microprocessor Characteristics

          E - Exit


          Make Selection : [a]
}
attributes
a=formonly.s_choice type char, upshift, autonext, reverse;
instructions
delimiters "  "
end
```

```
database formonly
screen
{
         MULTIPROCESSOR SYSTEM DESIGN - TUTOR          SCREEN-5
         ------------------------------------


             1 - Generate Design Reports
             2 - Generate Characteristics Reports
             3 - Generate Network Reports
             4 - Generate Configuration Reports

             E - Exit


             Make Selection : [a]
}
attributes
a=formonly.s_choice type char, upshift, autonext, reverse;
instructions
delimiters "  "
end
```

```
database MULPROST
screen
{
============================================================
[a|b|c]        MULTIPROCESSOR CHARACTERISTICS        SCREEN-9
============================================================

   [f000                                             ]

   [f000                                             ]

   [f000                                             ]

   [f000                                             ]

   [f000                                             ]

============================================================
}
end
tables
multchar

attributes
a = formonly.b1;
b = formonly.b2;
c = formonly.b3;
f000 = multchar.charname, REVERSE, UPSHIFT;

INSTRUCTIONS

DELIMITERS "  "

SCREEN RECORD s_multchar [5] (multchar.charname)

end
```

```
database MULPROST
screen
{
===================================================================
Characteristic   [f000                                            ]
===================================================================
Description      [f001                                            ]
                 [f002                                            ]
                 [f003                                            ]
                 [f004                                            ]
                 [f005                                            ]
                 [f006                                            ]
                 [f007                                            ]
                 [f008                                            ]
                 [f009                                            ]
                 [f010                                            ]
                 [f011                                            ]
                 [f012                                            ]

remark           [f014                                 ]
                 [f015                                 ]
=====================================================================
}
end
tables
multchar
attributes
f000 = multchar.charname, REVERSE, UPSHIFT;
f001 = multchar.chardesc1, REVERSE, AUTONEXT;
f002 = multchar.chardesc2, REVERSE, AUTONEXT;
f003 = multchar.chardesc3, REVERSE, AUTONEXT;
f004 = multchar.chardesc4, REVERSE, AUTONEXT;
f005 = multchar.chardesc5, REVERSE, AUTONEXT;
f006 = multchar.chardesc6, REVERSE, AUTONEXT;
f007 = multchar.chardesc7, REVERSE, AUTONEXT;
f008 = multchar.chardesc8, REVERSE, AUTONEXT;
f009 = multchar.chardesc9, REVERSE, AUTONEXT;
f010 = multchar.chardesc10, REVERSE, AUTONEXT;
f011 = multchar.chardesc11, REVERSE, AUTONEXT;
f012 = multchar.chardesc12, REVERSE;
f014 = multchar.remark1, REVERSE, AUTONEXT;
f015 = multchar.remark2, REVERSE;
INSTRUCTIONS
DELIMITERS "  "
screen record s_mult (multchar.charname thru multchar.remark2)
end
```

```
database MULPROST
screen
{
==================================================================
Design No [f000       ] Desc [f001                               ]
==================================================================
Microprocessor [f002       ]

Multiprocessor [f003                ]

Network        [f004                ]

Desc.      [f005                                                 ]
           [f006                                                 ]
           [f007                                                 ]
           [f008                                                 ]
           [f009                                                 ]

App        [f010                                                 ]

}
end
tables
design
attributes
f000 = design.designno, REVERSE, NOENTRY;
f001 = design.description, REVERSE;
f002 = design.micropro, REVERSE;
f003 = design.multtype, REVERSE;
f004 = design.nettype, REVERSE;
f005 = design.desdesc1, REVERSE;
f006 = design.desdesc2, REVERSE;
f007 = design.desdesc3, REVERSE;
f008= design.desdesc4, REVERSE;
f009= design.appdesc1, REVERSE;
f010 = design.appdesc2, REVERSE;
end

instructions

delimiters "  "

end
```

```
database MULPROST
screen
{
================================================================
         Network Type [f000                ]           SCREEN-12
================================================================
Average Distance[f001     ]Dia.[f008] No. Of Comm. Links [f002  ]

Routing Algorithm [f003                ] No. of Computers [f007]

Algorithm Desc    [f004                                         ]
                  [f005                                         ]

Fault Tolarance   [f006  ]
}
end

tables
netwchar

attributes
f000 = netwchar.nettype, REVERSE, UPSHIFT,
Comments = "Please enter Network Type";
f001 = netwchar.avgdist, REVERSE, NOENTRY;
f008 = netwchar.diameter, REVERSE,
Comments="Enter Maximum of the minimum distance between any
two pairs of nodes";
f002 = netwchar.commlink, REVERSE,
comments = "Please enter No of Communication Links";
f007 = netwchar.computers, REVERSE,
comments = "Please enter No of Computers";
f003 = netwchar.routalgo, REVERSE,
comments = "Please enter Name of the Routing Algorithm";
f004 = netwchar.algodes1, REVERSE, AUTONEXT,
comments = "Please enter Routing Algorithm short description";
f005 = netwchar.algodes2, REVERSE,
comments = "Please enter Routing Algorithm short description";
f006 = netwchar.faultolr, REVERSE;
end

Instructions

delimiters "  "

screen record s_netwchar(netwchar.nettype thru netwchar.faultolr)

end
```

```
database MULPROST
screen
{
=================================================================
             Microprocessor  [f000       ]            SCREEN-10
=================================================================
Basic Instruction [f001                      ]

Direct Add. Range [f002                 ]

No. Of Add Mode   [f003                                     ]

Basic Clock [f004      ] Primitive Date Types [f005              ]

Date Structure  [f006                           ]

Primitive Control [f007                         ]

Control Structure [f008                         ]

O. S. Support [f009           ]  General Pur. Reg [f010         ]
}
end

tables
multperf

attributes
f000 = multperf.micropro, REVERSE;
f001 = multperf.bascinst, REVERSE;
f002 = multperf.dar, REVERSE;
f003 = multperf.noofadmo, REVERSE;
f004 = multperf.bcf, REVERSE;
f005 = multperf.primdata, REVERSE;
f006 = multperf.datastrc, REVERSE;
f007 = multperf.primcont, REVERSE;
f008 = multperf.contstrc, REVERSE;
f009 = multperf.ossupport, REVERSE;
f010 = multperf.gpr, REVERSE;
end

instructions

delimiters "  "

screen record s_multperf (multperf.micropro thru multperf.gpr)
end
```

```
database MULPROST
screen
{
==========================================================
[a|b|c]       SELECT MICROPROCESSOR              SCREEN-8
==========================================================

                  [f000        ]

                  [f000        ]

                  [f000        ]

                  [f000        ]

==========================================================
}
end
tables
multperf

attributes
a = formonly.b1;
b = formonly.b2;
c = formonly.b3;
f000 = multperf.micropro, REVERSE, UPSHIFT;

INSTRUCTIONS

DELIMITERS "  "

SCREEN RECORD s_michelp [4] (multperf.micropro)

end
```

```
database MULPROST
screen
{
=================================================================
           MAINTAIN SYSTEM CONFIGURATION                SCREEN-6
=================================================================
System Configuration:       [f000                          ]

Description:[f001                                             ]

            [f002                                             ]

            [f003                                             ]

            [f004                                             ]

            [f005                                             ]

            [f006                                             ]

            [f007                                             ]

            [f008                                             ]
}
end
tables
configsm
attributes
f000 = configsm.conname, UPSHIFT, REVERSE;
f001 = configsm.confdes1,  AUTONEXT, REVERSE;
f002 = configsm.confdes2,  AUTONEXT, REVERSE;
f003 = configsm.confdes3,  AUTONEXT, REVERSE;
f004 = configsm.confdes4,  AUTONEXT, REVERSE;
f005 = configsm.confdes5,  AUTONEXT, REVERSE;
f006 = configsm.confdes6,  AUTONEXT, REVERSE;
f007 = configsm.confdes7,  AUTONEXT, REVERSE;
f008 = configsm.confdes8,  REVERSE;
end

instructions

delimiters "  "

SCREEN RECORD s_confmnt (configsm.conname THRU configsm.confdes8)

end
```

```
database MULPROST
screen
{
===========================================================
[a|b|c]        SELECTION OF CONFIGURATION          SCREEN-7
===========================================================

             [f000                          ]

             [f000                          ]

             [f000                          ]

             [f000                          ]

             [f000                          ]

===========================================================
}
end
tables
configsm

attributes
a = formonly.b1;
b = formonly.b2;
c = formonly.b3;
f000 = configsm.conname, REVERSE, UPSHIFT;

INSTRUCTIONS

DELIMITERS "  "

SCREEN RECORD s_confhelp [5] (configsm.conname)

end
```

```
# Rakesh Kamdar
# Makefile for the Multiprocessor System Design Tutor

CFLAG=

LDFLAGS=/st:12288 /se:256 /exepack

MAK=$(MENUo) $(HELPo) $(MNTo) $(REPo)

MENUo=add_date.obj mainmenu.obj menu1.obj menu3.obj
                deletwin.obj menu4.obj menu5.obj init_pro.obj

HELPo=confhelp.obj multchlp.obj nethelp.obj michelp.obj

MNTo=mult_chr.obj netwchar.obj multperf.obj mntconf.obj

REPo=prt_char.4gl prt_net.4gl

cci $*
del $*.c
del $*.ec

form4gl $*

add_date.obj:add_date.4gl

deletwin.obj:deletwin.4gl

mainmenu.obj:mainmenu.4gl

menu1.obj:menu1.4gl

#menu2.obj:menu2.4gl

menu3.obj:menu3.4gl

menu4.obj:menu4.4gl

menu5.obj:menu5.4gl

init_pro.obj:init_pro.4gl

multchlp.obj:multchlp.4gl

mult_chr.obj:mult_chr.4gl

netwchar.obj:netwchar.4gl
```

```
multperf.obj:multperf.4gl

michelp.obj:michelp.4gl

confhelp.obj:confhelp.4gl

mntconf.obj:mntconf.4gl

prt_char.obj:prt_char.4gl

prt_net.obj:prt_net.4gl

nethelp.obj:nethelp.4gl

menu1.frm:menu1.per

menu2.frm:menu2.per

menu3.frm:menu3.per

menu4.frm:menu4.per

menu5.frm:menu5.per

multchar.frm:multchar.per

mainmenu.frm:mainmenu.per

mult.frm:mult.per

netwchar.frm:netwchar.per

multperf.frm:multperf.per

michelp.frm:michelp.per

nethelp.frm:nethelp.per

configsm.frm:configsm.per

confhelp.frm:confhelp.per
```

```
echo off

REM ##############################################################
REM #
REM # FILE NAME:cci.bat
REM #
REM # DESC:    The main functionality of this batch process is to
REM #     compile the Very Huge 4GL function using Huge Library
REM #     of the Microsoft C 4.0. This Batch file will generate
REM #     object file from the 4GL file.
REM #     Usage:cci <4GL filename without extension>
REM #     i. e. cci menu1   for menu1.4gl
REM #
REM ##############################################################

REM # IF first argument to cci is null then display the Usage.
if .%1 == . goto usage

:loop

if .%1 == . goto exit

REM # Check the existence of 4gl module.
if not exist %1.4gl goto usage

if exist %1 goto usage

echo Phase 1 ...

REM # Use fglc of I4GL library to compile .4gl module into .obj .
fglc %1.4gl

REM # If there is an error in compilation that it will go and
REM # search for the error file. If error file is not found than
REM # the compilation is O. K.
if not exist %1.err goto is_ok

BEEP

REM # If error file exist then open error file and show the error

REM # to the user and then show 4gl module to correct the error.
vi %1.err %1.4gl

pause
```

```
REM # Delete error file after correction.
del %1.err
del fg*.

goto loop
:is_ok

echo Phase 2 ...

c4gl -e %1.ec

echo Phase 3 ...

msc -AH %1 /Gt16;

REM del xx457
REM del %1.ec
REM del %1.c

shift
goto loop

:usage
echo Usage:  cci 4gl_file ..... (no .4gl)
:exit
```

```
#####################################################################
# FUNCTION NAME: mainmenu()
#
# FILE NAME:     mainmenu.4gl
#
# DESCRIPTION: This is the main function of the whole system, It
# displays the mainmenu of the system and it is used
# as driver for the all function. This is the only
# which contains main, all other files are functions
# only. Here, we do the defer interrupt so that when
# user presses (CONTROL-C) key it sets int_flag to
# true, and based on that we can take the decision
# wether user aborted the process or selected a row.
#
#####################################################################
#
# Name of the database is abbriviated to mulprost from
#(Multiprocessor System Design Tutor)
database mulprost

main
defer interrupt
call init_prog ()
call mainmenu ()

end main

function mainmenu()

define choice char(1)
define dummy,dummy2, dummy3, dummy4, dummy5 char(3)
define dummy1, done smallint

let done = FALSE

# This opens the form to display on the screen.
open form mainmenu from "mainmenu"
clear screen

while not done
#This window covers the form opend before and it also displys the

#border of the window.
open window win_1 at 2,3 with 22 rows, 75 columns
attribute (border)
display form mainmenu
   call add_date()
```

```
input choice from s_choice

#If user selects option from the mainmenu, then close the window
#with form and call the another menu for the user.

case
when choice = 1
close window win_1
call menu1 ()
when choice = 2
close window win_1
call net_char ()
when choice = 3
close window win_1
call menu3 ()
        when choice = 4
close window win_1
call menu4 ()
        when choice = 5
close window win_1
               call menu5 ()

when choice = "E"
close window win_1
           let done = TRUE
exit case
otherwise
close window win_1
            error "Invalid Menu Selection"
end case
    options input no wrap
initialize choice to null
end while

close form mainmenu
end function
```

```
##################################################################
#
# FUNCTION NAME:  net_char()
#
# FILE NAME:  netwchar.4gl
#
# DESCRIPTION:  This module is use to maintain the
#  characteristics of a Network reference table. It is also used
#  as the view in the mainmenu.
##################################################################
database mulprost

globals

define p_net record like netwchar.*
define p_ddsp record like netwchar.*
define is_del smallint
define eflag smallint
define g_is_err smallint

end globals

function net_char ()

define pa_net array[10] of record like netwchar.*
define idx smallint
define iflag smallint
define scrn smallint
define cnt smallint
define pa_rows smallint
define i smallint
define redraw smallint
define tmp_dat date

initialize p_ddsp.* to null

# defer interrupt

open window net_disp at 6,7 with form "netwchar"
    attribute (border, message line last,prompt line last-1)

let tmp_dat = today

display tmp_dat using "dd mmm yy" at 4, 1

for i = 1 to 10
initialize pa_net[i].* to null
```

```
end for

declare net_cnt cursor for
select * from netwchar

let redraw = true

while redraw
error ""
let idx = 0
let redraw = false
# return to the menu after exitting the input array

foreach net_cnt into p_net.*
let idx = idx + 1
let pa_net[idx].* = p_net.*
end foreach

display
"NETWORK: F1-Insert Row  F2-Delete Row"
at 1,1

call set_count(idx)

input array pa_net without defaults from s_netwchar.*

before row
let idx = arr_curr()
display pa_net[idx].* to s_netwchar.* attribute (reverse)
let scrn = scr_line()
if idx > 1 then
if pa_net[idx-1].nettype is null then
let redraw = true
exit input
end if
end if
let p_net.* = pa_net[idx].*
let iflag = 0

on key (control-N,interrupt)
let pa_net[idx].* = p_net.*
display pa_net[idx].* to s_netwchar.* attribute (reverse)
let int_flag = false
exit input

before insert
initialize p_net.* to null
```

```
after insert
let iflag = -1
let eflag = 0
if  (pa_net[idx].nettype is not null) then
select count(*)
into cnt
from netwchar
where nettype = pa_net[idx].nettype

if cnt != 0 then
error " Code ",pa_net[idx].nettype,
                              " already exists "
clear form
next field nettype
end if
end if

if (pa_net[idx].nettype is not null) then
insert into netwchar
values (pa_net[idx].*)
message "Record added"
sleep 2
message ""
else
error "Entry is required in each field"
next field nettype
end if

before delete
let iflag =-1
call deletewin() returning is_del
if is_del then
message "Deleting record..."
delete from netwchar
where nettype = pa_net[idx].nettype
message "Record deleted"
sleep 2
message ""
end if

after field computers

case
when pa_net[idx].nettype = "RING NETWORK"
let pa_net[idx].avgdist = (pa_net[idx].commlink + 1)/2
display pa_net[idx].avgdist to s_netwchar.avgdist
```

```
when    pa_net[idx].nettype    =    "CUBE    CONNECTION    NETWO"let
pa_net[idx].avgdist =
        (pa_net[idx].commlink + 1)/2 *(2**pa_net[idx].computers)
display pa_net[idx].avgdist to s_netwchar.avgdist

when pa_net[idx].nettype = "ALPHA NETWORK"
let pa_net[idx].avgdist =
        (pa_net[idx].commlink + 1)/2 *(2**pa_net[idx].computers)
display pa_net[idx].avgdist to s_netwchar.avgdist

when pa_net[idx].nettype = "HYPER TREE NETWORK"
let pa_net[idx].avgdist = (pa_net[idx].commlink + 1)/2
display pa_net[idx].avgdist to s_netwchar.avgdist


when pa_net[idx].nettype = "MULTI TREE STRUCTURE"
let pa_net[idx].avgdist =
        (pa_net[idx].commlink + 1)/2 *(2**pa_net[idx].computers)
display pa_net[idx].avgdist to s_netwchar.avgdist

when pa_net[idx].nettype = "BETA NETWORK"
let pa_net[idx].avgdist = (pa_net[idx].commlink + 1)/2
display pa_net[idx].avgdist to s_netwchar.avgdist
end case

after row
let pa_rows = arr_count()
display pa_net[idx].* to s_netwchar.*
attribute(reverse)
if (pa_net[idx].nettype is null) then
let iflag = -1
end if
if(p_net.nettype != pa_net[idx].nettype or
p_net.avgdist != pa_net[idx].avgdist or
p_net.diameter != pa_net[idx].diameter or
p_net.computers != pa_net[idx].computers or
p_net.commlink != pa_net[idx].commlink or
p_net.routalgo != pa_net[idx].routalgo or
p_net.algodes1 != pa_net[idx].algodes1 or
p_net.algodes2 != pa_net[idx].algodes2 or
p_net.faultolr != pa_net[idx].faultolr or
  (p_net.nettype is NULL and pa_net[idx].nettype is NOT NULL) or
  (p_net.avgdist is NULL and pa_net[idx].avgdist is NOT NULL) or
  (p_net.diameter  is  NULL  and  pa_net[idx].diameter  is  NOT  NULL)
or
  (p_net.computers is NULL and pa_net[idx].computers is NOT NULL)
or
  (p_net.commlink is NULL and pa_net[idx].commlink is NOT NULL)
or
  (p_net.routalgo is NULL and pa_net[idx].routalgo is NOT NULL)
or
  (p_net.algodes1 is NULL and pa_net[idx].algodes1 is NOT NULL)
or
  (p_net.algodes2 is NULL and pa_net[idx].algodes2 is NOT NULL)
or
  (p_net.faultolr is NULL and pa_net[idx].faultolr is NOT NULL))
then
  update netwchar set
  netwchar.* = pa_net[idx].*
where nettype = p_net.nettype
message "Record updated"
sleep 2
message ""
end if
on key (escape)
let redraw = false
exit input
end input
end while #redraw
close window net_disp
display "" at 1, 1
display "" at 2, 1
return
end function
```

```
##########################################################
# FILE NAME: lnk.bat
#
# DESCRIPTION: This batch program is use to link all the
# necessary modules.
##########################################################

VERBOSE
#NOVECTOR llibfp.lib,llibc.lib,em.lib,libh.lib
NWIDTH 30
MAP=msdt A
RELOAD FAR 200
STACK 4500
OUTPUT c:\modules\msdt.exe
OVERLAY CODE, FAR_DATA, NIL, ENDCODE
FILE mainmenu, init_pro
SEARCH llib4gl,llibsql,llibform
BEGIN
     section file menu1
     section file multchlp
     section file add_date
     section file deletwin
END
BEGIN
     section file netwchar
END
BEGIN
     section file confhelp
END
BEGIN
     section file menu3
     section file michelp
     section file nethelp
END
BEGIN
     section file mult_chr
END
BEGIN
     section file menu4
     section file multperf
     section file mntconf
END
BEGIN
     section file menu5
     section file prt_net
     section file prt_char
END
```

SELECTED BIBLIOGRAPHY

1. Kai Hwang and Faye A.Briggs, Computer Architecture and Parallel Processing, McGraw Hill Book Company, 1985.

2. D. P. Agarwal and V. K. Jankiram, "Evaluating the Performance of Multicomputer Configuration", IEEE Trans. Computers, Vol. C-30, pp.23-37, May 1981.

3. Hoo-min D. Toong and Amar Gupta, "An architectural comparision of contemporary 16-Bit Microprocessors", IEEE Micro, Vol. 26, pp.26-36, May 1981.

4. Robert H. Kuhn and David A. Padua, Tutorial on Parallel Processing, IEEE Computer Society, 1981.

5. Intel Corporation, 80386 Hardware Reference Manual, 1986.

6. Motorola Inc, MC 68020 32-Bit Microprocessor User's Manual, 1986.

7. Informix Inc., How to Choose an RDBMS, 1988.

8. Manju Bewtra, "Informix and Ingres: A Comparative Look", DATABASE, pp.58-69, August 1988.

9. Informix Inc, Informix-4GL User Guide, 1987.

10. Informix Inc, Informix-4GL Reference Manual 1 - 2, 1987.