AN OBJECT-ORIENTED PROTOTYPING ENVIRONMENT FOR ARCHITECTURES AND OPERATING SYSTEMS

BY

KHALED M. HASSAN

Bachelor of Civil Engineering

Cairo University

Cairo, Egypt

1986

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 1992

AN OBJECT-ORIENTED PROTOTYPING ENVIRONMENT
FOR ARCHITECTURES AND OPERATING SYSTEMS

Thesis Approved:

M. Samadzadeh-H.

Thesis Advisor

G. E. Hedrick

Blayne E. Mayfield

Thomas C. Collins

Dean of the Graduate College

ACKNOWLEDGEMENTS

I would like to express my appreciation to Dr. Mansur H. Samadzadeh for his advisement, guidance, dedication, encouragement, and instruction throughout my thesis research work. Without his support, motivation, and patience it would have been difficult to complete this work as it is now.

My sincere thanks to Drs. G. E. Hedrick and B. E. Mayfield for serving on my graduate committee. Their suggestions and support were very helpful throughout the study.

I would like to extend my deepest appreciation to Mr. Ik-Jeong Jhun for his help and advice that helped me improve this work. I also wish to thank my family for their support in completing my graduate studies.

TABLE OF CONTENTS

Chapter Page

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The dramatic increase in the speed of computers in performing a typical instruction - from one tenth of a second to nanoseconds - has come about mainly as a result of the progress in electronic logic technology. During the last few years, it appears that it is progressively more difficult to increase the speed of computers only by upgrading the switching logic technology, hence there is a need to have parallel processing as a way to have faster computers. In other words, we can say that MIMD multiprocessors - multiple instruction-streams and multiple data-streams - are going to be the computers of the future.

This relatively new technique of programming - concurrent programming - introduces new kinds of correctness and performance problems that do not occur in sequential programming for programmers. Mutual exclusion, deadlock, and starvation are examples of correctness difficulties. The main issue in performance is the dramatic difference in execution of a given partitioned program on different multiprocessors. In addition to these problems there are many details in the creation of tasks, mutual exclusion, and waiting for events, that the programmer needs to be aware of [Sarkar89].

Decomposition into tasks is a common approach utilized to organize programs that have a number of independent "parts" (i.e., units that can be executed at the same time). Such programs can be implemented as a set of tasks to make their implementation more efficient. Each program may need to spawn other subtasks, thus making the program's set of tasks change dynamically. The encapsulated declaration of tasks keeps many details inside their implementations, so that a programmer can concentrate on the implementation of one task or the communication among tasks.

Object-oriented programming, because of its support of sharing interfaces, sharing code, and reusable software, seems to be a promising solution to the increase in software complexity especially the increase in the complexity of operating systems [Russo91a], [Russo91b], [Cahill91], [Shapiro91]. Different methods of implementation for object-oriented operating systems are discussed by Finlayson [Finlayson91].

Object-oriented operating systems are an attempt to address the problem of operating system complexity and provide support for distributed systems. There is a need for an object-oriented package or prototyping system to help prototype these new operating systems.

In this thesis a package of classes was created to give the user the ability to prototype various operating system models. In each prototyped operating system, jobs written in hexadecimal code can be handled and the instructions constituting each job can execute. In this first version of the package, the input jobs are in hexadecimal to avoid the added burden of implementing a compiler for a specific language and tie the package to this language. The package gives its user the ability to create complicated models which have more than one memory, loader, and/or cpu (as mentioned in section 4, in this thesis, cpu denotes class cpu and CPU denotes the simulation of the central processing unit of the architecture being modeled). The system also provides a debugging option to give the package user the ability to follow the execution of jobs, instruction by instruction, through four windows in the default debugger. One window contains the user options, the second the register values, the third the instruction decoding information, and the fourth general information about a job such as its id, the memory allocated to it, and the CPU executing it.

Chapter II of this thesis gives a brief discussion about the object-oriented programming concepts. Chapter III briefly discusses parallel programming and parallelism in object-oriented languages. The implementation of the package is discussed in detail in Chapter IV. Evaluation of the package is included in Chapter V. Chapter VI contains the summary and some possible areas of future work.

CHAPTER II

OBJECT-ORIENTED PROGRAMMING

Object-oriented programming is derived from Simula 67 [Kirkerud89] and is based on the concept of an "object" [Nygaard86]. Object-oriented programming is a technique that facilitates code reuse. Encapsulation, data abstraction, and inheritance are examples of its important features, which are defined below.

Classes of objects and operations on objects are the main components of object-oriented languages. Operations, when invoked, operate on multiple type of objects, and classes may share components by inheritance.

The designer of an object-oriented package defines classes and declares objects as instances of those classes. Each object has its own state consisting of instance variables and methods implemented in the object's class. A class (a child or a descendant class) can inherit the definition of other classes (a parent or an ancestor class). The object client of the package does not need to understand anything about the implementation of the package classes. The object client just uses objects operations. Changes in the implementation of the package classes, which support the old external interface, do not affect the object client code.

In object-oriented languages, data abstraction is implemented by the encapsulation technique. This means that the code of the object client of a class depends only on the class interface. Hence, the implementation of a class' methods can change without affecting the object client code, while the new implementation supports the old interface.

Inheritance helps form new classes from the existing classes. A new class can inherit methods, functions in other classes, by reusing parts of the implementations of these existing classes. The external interface for inheritance clients in general is less restrictive than the external interface for object clients, this is a potential weak point in class encapsulation [Snyder86].

2.1 Inheriting Instance Variables Safely

If the code in an inheriting class can access the instance variables of one of its ancestors, then the designer of that ancestor class will not be able to rename or remove those instance variables without affecting the inheritance client code or descendant classes. By using the same technique as the one used in objects interfaces, this problem can be solved. In other words, the descendant classes need to use certain operations to access the instance variables of their ancestor classes instead of their direct use of these variables [Snyder86].

Using the self invocation (e.g., the "this" invocation in C++) is not adequate to call an operation from a parent class. Object-oriented languages need to support a new invocation for parent classes such as a super invocation, or use the name of the parent class in front of the operation name [Snyder86].

Some object-oriented languages tie subtyping with inheritance. In such cases, if the designer changes the inheritance of a class from one parent to another, the object client code will be illegal if it uses the subtyping relation between the existing classes;

although the class still has the same external interface. Subtyping may need to be separated from inheritance and depend only on the class behavior [Snyder86].

2.2 Multiple Inheritance

Multiple inheritance means that a class may have more than one parent. Compilers of object-oriented languages have different strategies to handle multiple inheritance [Snyder86]. Three strategies, namely graph-oriented, linear-oriented, and tree-oriented inheritance, are briefly described below.

The inheritance graph is modeled directly in the graph-oriented strategy. A problem arises when a class inherits operations with the same name from more than one parent. One solution of this name resolution problem is to redefine the operation in the descendant class. By doing so, the parent operation can be invoked unambiguously in this new definition. Another technique for name resolution is choosing the operation of the first parent that has been implemented. However, this technique seems rather arbitrary. A third technique is to make any conflict an error, except if it is for the same inherited operation. However, this method puts inheritance in the external interface of the class. Thus, the object client code will be illegal, as a result of the name conflict between two different methods, if the class under consideration reinherits from another ancestor class which has an operation with the same name.

The linear-oriented strategy flattens the inheritance graph to a linear chain. One of its drawbacks is that one of the parents of a class will be its immediate parent in the linear order and conflicting operations among the parents of the class will be selected from this immediate parent without any good choice between this parent and others except the text order, also the difficulty of communication between a class and its "real" parent(s), because another parent may be between the class and its "real" parent(s).

In the tree-oriented inheritance strategy, the conflict between different parents is always an error. Each parent in multiple inheritance has a set of its instance variables for each inheritance path.

2.3 Separation of Interface and Implementation

There are a number of representations for separating the interface from the implementation in different object-oriented languages. Each representation is built based on a different set of rules. Two representations are discussed in the following subsections.

2.3.1 Separate Class and Implementation Modules

In this representation, subtyping is defined in the "class" module, where the supertype is a parameter of the class (see Figure 1). For example, the interface of class "A" is inherited in class "B" without its implementation.

Operations are inherited virtually, so they use dynamic binding to find the object type to be executed on at run time, while the implementation can be inherited in the implementation module by the "use" clause or reimplemented in the implementation module [Ancona91]. For example, in class B, operation D reimplemented, while operation C inherits its implementation from class A. It is clear that multiple inheritance can be implemented by selecting operations from different parents.

```
class A;
procedure C(...);
function D(...):...;
end A.
class B(A);
procedure E(...);
...
end B.
module B; of B use C;
procedure E(...);
begin
...
end E;
function D(...):...;
begin
...
end D;
end B.
```

Figure 1. Example of Separate interface and implementation modules

2.3.2 Extension of C++

By adding new keywords such as "interface", "implement", and "reuse", by using virtual base classes, virtual functions, and multiple inheritance features, and by implementing a C++ preprocessor to convert the new C++ code to a standard C++ code, the new C++ extension supports classes with separate interface and implementation [Martin91]. This strategy gives a program the capability of executing in distributed systems by having one global interface and multiple local implementations on each node.

In the example in Figure 2 (similar to an example in [Martin91]), the interface Stop_place inherits both PeopleWait and Port interfaces and adds its needed methods.

The class Queue_people uses the PeopleWait interface and has the implementation of its methods. The class Bus_stop uses the Stop_place interface, inherits the needed implementation from the class Queue_People, and adds the implementation for the rest of its methods.

```
interface PeopleWait {
        put(person *);
        person *get();
        int size();
};
interface Port {
        city *distance();
        time *time_needed();
};
interface Stop_place : PeopleWait, Port {
        boolean covered();
};

class Queue_people implements PeopleWait {
        person *head, *tail;
public:  Queue_people() { head=tail=NULL; }
        put(person *p);
        person *get();
        int size();
};

class Bus_stop implements Stop_place reuses
        public:Queue_people { boolean cover;
public:  Bus_stop(boolean cv) { cover=cv; }
        city *distance();
        time *time_needed();
        boolean covered() { return cover; }
};
```

Figure 2. Example of separate interface and implementation classes

It is easy to have different implementations for the Stop_place interface at the same time. This technique needs some of the extensions provided by "C++" such as its support of the inheritance of pure virtual functions [Martin91].

All these methods add to the cost overhead in terms of storage, compiling time, and execution time. More work is needed to devise languages that support, in their design philosophy, the separation of interface (subtyping) and implementation (inheritance).

2.4 Classes Versus Prototypes

As a result of the class representation problems of object-oriented languages, researchers have tried to find new object-oriented representations. There is a belief that prototype-based languages will be free of many class-based languages' problems [Borning86].

In the class-based languages, because of the objects' message protocol, there must be at least one class created for any object. The language will be more complicated if classes themselves are objects. In this case, classes need to be instances of metaclasses to understand initialization messages. On the other hand, object clients have to move to the abstract (class) level to create new classes whenever they need to design a new object.

In the prototype-based languages, a new object is a modified copy of a prototype. Each object has its state and behavior and can change both of them. The state of an object is a set of named fields. Its behavior has two components, a method dictionary and a protocol. An object contains a protocol for message description, a protocol for message arguments, and a protocol for the messages' returned results.

In prototype-based languages, there is a separate inheritance method for each part of the object. The inheritance client can inherit object field names, behaviors, or protocols separately. This means that we have separate subtyping (protocol reuse) and inheritance (implementation reuse).

When using the prototype technique, it does not mean that there are no classes in the prototype-based languages at all. Objects, which have the same field names, methods dictionary, and protocols, may be put together in one class. Similarly, instead of having multiple methods in a dictionary, they may be divided into subclasses and superclasses.

Prototype-based and class-based languages can be compared in terms of their advantages and disadvantages. The prototype-based technique is simpler than the class-based technique because in the prototype-based techniques the prototypes can be considered objects and hence there are no classes or metaclasses. Each object is created with its initial values and may have its unique behavior. This technique also gives the ability to separate the interface and the implementation.

On the other hand, prototypes may not be reasonable for some types such as integers, stacks, or queues. Prototype-based languages have their own problems, for example, a programmer may modify a prototype intentionally, as an object, which will affect all objects created from that prototype. Copying an object just to change a few values in it may not be efficient in some cases. Generally, it is easy to avoid these drawbacks by imposing some protection on the prototype messages and having more efficient ways of copying objects [Borning86].

2.5 Other Concepts

Encapsulation, inheritance, and separation of interface and implementation are some of the important aspects of object-oriented design. The object-oriented design of a software package must incorporate other principles such as exception handling, type parameterization, and reflection. These concepts are at the center of the design of most object-oriented software libraries that have been implemented so far [Gorlen87] and [Booch90].

CHAPTER III

PARALLEL PROGRAMMING

There are many sequential languages (e.g., C, FORTRAN, and Pascal) that have been upgraded to have parallelism, but few new basically parallel languages (e.g., Ada) have been defined. The use of upgraded languages can lead to highly obscure and unportable code. That is part of the reason why there are not many parallel programs running on the available multiprocessor computers [Sarkar89].

Programmers are being pushed towards parallel programming to exploit the proliferating hardware (i.e., the multiprocessors) to perform peripheral processing on slow networks and devices such as disks, terminals, and printers, and to capitalize on the users' ability to do more than one thing at the same time [Birrell89].

3.1 Multiprocessors

Multiprocessors are general-purpose, asynchronous parallel machines with multiple instruction-streams and multiple data-streams. Multiprocessors can be classified into two main classes, tightly-coupled and loosely-coupled [Sarkar89] as described in the following sections.

3.1.1 Classification

Tightly-coupled and loosely-coupled multiprocessors are briefly discussed in this subsection. In tightly-coupled multiprocessors, processors communicate through a shared memory (e.g., in Alliant FX8, BBN Butterfly, Denelcor HEP, ELXSI 6400, Encore Multimax, IBM RP3, and Sequent Balance). There are different types of tightly-coupled multiprocessor structures including the following.

1- Shared bus: The bus connects the processing elements to a global shared memory. The local memory is used as a private cache.

2- Shared multiple buses: Multiple buses support more processors than a single bus, but the complexity of the system increases.

3- Hierarchical clusters: The multiprocessor structure is interconnected with an inter-cluster bus.

4- Interconnection network: It avoids bus problems. It connects a number of processing elements to a number of shared memory modules. Because it is expensive to build a network for a large number of processors, a multistage network is built from smaller networks.

In loosely-coupled multiprocessors, processors communicate by exchanging messages (e.g., in Caltech Cosmic Cube, Intel iPSC, NCUBE-10, and workstation-based distributed systems). There are different types of loosely-coupled multiprocessor structures including the following.

1- LANs: A local area network can work as a loosely-coupled multiprocessor because it has a bus to handle inter-processor messages.

2- Distributed systems: These systems generally have the same bus structure as LANs and can work for programs with large granularity.

3.1.2 Properties

There are tradeoffs between the size of program execution granularity and multiprocessor scalability. A multiprocessor can support more programs efficiently, if it has small granularity (the minimum program granularity value below which performance degrades significantly). On the other hand, a parallel program in general can be executed more efficiently on a larger number of processors, if it has larger granularity. An increase in scalability in general can come at the cost of larger granularity [Sarkar89].

3.2 Partitioning and Scheduling

Partitioning is the process of dividing a program into sequential units called tasks. Partitioning is an important issue because of its effect on parallel program execution granularity. On the other hand, scheduling (assigning the tasks of the partitioned program to processors) is important in attaining a good utilization of the processors. There are in general three ways of automatic partitioning and scheduling [Sarkar89] as briefly discussed below.

Although the run-time partitioning and scheduling strategy adds extra overhead during program execution, it gives better partitioning and scheduling because of the available run-time information which leads to simpler partitioning and scheduling algorithms.

Compile-time partitioning and run-time scheduling is the strategy commonly used. The programmer explicitly partitions the program into tasks, while the scheduling of tasks on processors is done at run time.

The compile-time partitioning and scheduling strategy may lead to inefficient scheduling because of the possible errors in the estimation of task execution times and the associated overhead.

3.3 Writing Concurrent Programs

Writing concurrent programs can be difficult compared with writing sequential programs, but if the programmer works carefully with a specific technique, (s)he can avoid common errors. Birrell [Birrell89] discussed many of these difficulties and developed programming writing strategy using threads.

It is important to have a library of functions as part of the run-time support of the operating system to support the programmer. UNIX has its standard library of heavyweight processes. Each process has its own resources of time (execution and linking time), space (virtual store), and external environment (access to disks and files).

These types of heavyweight processes have a high overhead. For example, in BSD UNIX, because processes can share environments but not space, creating new processes means initializing them with their parents' state. In UNIX System V, although processes may share space, each process has its mapping table and registers. Switching between these processes is a high-overhead operation. This heavyweight processes' overhead has motivated researchers to present lightweight tasks (Gautron in C++ [Gautron91] and Finkel [Finkel87]).

Lightweight processes are resident in a specific address space. Different from heavyweight processes, lightweight processes are much faster, because they do not create new mapping tables during their calls nor need special instructions during switching among them.

3.4 Concurrent Programming on Personal Computers

Systems programmers were among the first group of people to take up concurrent programming. More recently, programmers in other fields such as database systems and expert systems have become interested in using concurrent programming. As a result of this proliferation, the need to have concurrent programming on personal computers has increased also. ENSEMBLE [Santo91] is an example of a concurrent programming library on personal computers.

ENSEMBLE is a system library written in Turbo Pascal for concurrent programming on personal computers. The implementation of ENSEMBLE allows a user to create two important abstractions for an application, coroutines and tasks. In this system, a user can create coroutines dynamically. The coroutine mechanism is supported by an "Interrupt Handling" procedures to terminate one coroutine, transfer control to another one, and later return to the interrupted coroutine.

As for tasks, a user can create them dynamically and control context switches. Tasks may have one of the four states: Sleeping, Ready, Running, or Terminated. The task mechanism is supported by a scheduler and queue management routine. Implementations for "Semaphores" and "Monitors" are also presented under this system.

3.5 Parallelism in Object-oriented Programming Languages

Parallelism in object-oriented environments is a relatively new research area. The idea is to create objects that have parallel execution and communication capabilities. Two kinds of communication are introduced: synchronous and asynchronous. The ability to have many activities within an object, is also introduced in parallel object-oriented environment [Corradi90].

There are two kinds of objects in such environment: active and passive. Passive objects are analogues to objects in other object-oriented languages. Active objects have an active role in synchronous and asynchronous communication. Active objects are similar to actors in the Actor paradigm [Agha86]. There are two subtypes of these active objects, inter-objects that execute in parallel and communicate during their execution, and intra-objects that can execute tasks inside themselves concurrently. Intra-objects are presented in the active object language PO (Parallel Objects) [Corradi87].

The interesting issue in these objects is the ability to inherit this kind of behavior from other classes. Communication between objects can be divided into three classes depending on the nature of passing and execution [Koivisto91].

In asynchronous passing and synchronous execution, the relation between the client and server objects continues from the request to the reply (see Figure 3). This case is similar to operation calls in passive objects.

In asynchronous passing and asynchronous execution, the client object does not wait for the reply (see Figure 4). This scheme is used when no reply is needed from the call. A client object can send many requests to different servers just to activate them. Each server object has a queue of requests that may be served FIFO.

```
  Client     Server
   |          *
   | request  *
   o-------->* reception
waiting       *
              x acceptance
              |
              | operation execution
     reply    |
   o<--------x end
   |          *
   |          *
   |          *
   |          *
```

Figure 3. Asynchronous passing and synchronous execution

```
  Client     Server
   |           *
   | request   *
   o--------->* reception
   |           *
   |           *
   |           x acceptance
   |           |
   |           | operation execution
   |           |
   |           x end
   |           *
   |           *
   |           *
   |           *
```

Figure 4. Asynchronous passing and asynchronous execution

```
              Client    Server
               |          *
               |   request*
intermediate M o--------->* reception
entity         |          *
               |          *
               |          x acceptance
               |          |
               |          | operation execution
               |  reply   |
               |<---------x end
    use of     |          *
    reply by M o          *
               |          *
               |          *
```

Figure 5. Synchronous passing and asynchronous execution

In synchronous passing and asynchronous execution, the client object does not wait for the reply (see Figure 5), although the result may be needed in the future. So there is an intermediate entity that receives the answer while the client object can get the reply whenever it is needed. The client object can check to see whether the entity has the answer or not before suspending itself and waiting for the answer. The intermediate entity can be presented as a separate object or as an internal variable in the client object.

CHAPTER IV

IMPLEMENTATION ISSUES

The main focus of this thesis is the implementation of a package, or a prototyping system, to simulate architectures and operating systems [Hassan92]. The package, which is written in C++, attempts to give its user the flexibility to create a model of the machine needed. The package implementation gives the user enough flexibility to prototype conventional, parallel, and object-oriented systems. The package uses the object-oriented approach to contain classes as basic encapsulated components, which the package user can use to build (i.e., simulate) a system. These classes include: hex_digit, byte, word, registers, storage, memory, disk, page_table, memory_table, loader, instructions, cpu, and clock.

4.1 Main Elements of the Package

Besides the default sizes of all elements, an object client can declare his/her system's elements with needed sizes. For example, the object client may have a 2 or 4-byte "word", declare a memory with different sizes of words and different page sizes, overload the system instruction set to have a new instruction set for his/her CPU (henceforth, as a notational convention, we will use cpu for "the class cpu" and CPU for "the object cpu" in the object client prototype), or even create a new debugger with different windows from the default ones, or overload the class window to have new features.

4.2 Parallel Processing in the Package

Creating loader, memory, and/or CPU is done independently. In other words, different memories, loaders, and/or CPU's can be created each with its unique features (see Figure 6). These instantiations can communicate easily in a parallel processing environment. For example, a loader can load jobs into a memory while one (or more) CPU executes other jobs in the same memory at the same time, also probably in other memories at the same time.

```
loader l1,l2,l3;
            //declare three loaders
memory m1(128,5),m2,m3;
            //declare three memories
ins_set inst1;
ins_set2 inst2;
            //declare two instruction sets
cpu c1(&ins1),c2(&ins2),c3(&ins2);
            //declare three cpu's
```

Figure 6. Examples of component declarations

In Figures 7 and 8, a parallel processing case is simulated to show how easy it is to use this package to model a multi-processor system. After declaring a system's components of (loaders, memories, and CPU's), it is straightforward to have different processes each use its own loader as well as its own memory and CPU.

Furthermore, in a more complicated system we may have a shared memory in which more than one loader can load new jobs at the same time, and more than one CPU may execute different ready jobs from this memory. To guarantee the integrity of the memory contents and to guard against the race condition and the readers/writers problem, the contents of memory must be protected. There are two ways that this protection can be enforced. The addition of a semaphore (one for all of the memory_table) in the class memory_table to protect its elements from being accessed by other CPU's and loaders, or the addition of a semaphore to class mem_element so that no more than one CPU or loader can access the mem_element at the same time, but more than one mem_element object can be accessed at the same time.

```
loader l1,l2;
                //declare two loaders
memory m1,m2;
                //declare two memories
ins_set inst1;
                //declare an instruction set
cpu c1(&ins1),c2(&ins1);
                //declare two CPU's
int i=fork();
if (i=0) {
        l1.load(jobs1,m1);
                //load jobs in memory m1
        c1.run_job_from(m1);
                //execute jobs from memory m1
        }
else {  l2.load(jobs2,m2);
                //load jobs in memory m2
        c2.run_job_from(m2);
                //execute jobs from memory m2
        }
```

Figure 7. Two separate systems load and execute their jobs in parallel

```
loader l1;
                //declare a loader
memory m1(128,5),m2,m3;
                //declare three memories
ins_set inst1;
ins_set2 inst2;
                //declare two instruction sets
cpu c1(&ins1),c2(&ins2),c3(&ins2);
                //declare three CPU's
l1.load(jobs1,m1);
l1.load(jobs2,m2);
l1.load(jobs3,m3);
                //load jobs in memories m1, m2, and m3
int i=fork();

if (i=0) {
        c1.run_job_from(m1);
                //execute jobs from memory m1
        c2.run_job_from(m2);
                //execute jobs from memory m2
        }
else c3.run_job_from(m3);
                //execute jobs from memory m3
```

Figure 8. Load jobs in sequence using one loader, and execute two of them in parallel with the third

4.3 Relations among Classes

Different relations among the package's classes are represented using object-oriented programming features such as the following.

1- We have single inheritance in this package or prototyping system (the "is a" relation). Examples include the classes byte, pt_element, and pcb_element from the class vect, the class register from the class word, and the class memory from the class storage.

2- We also have multiple inheritance of the class mem_element from both classes pcb_element and page_table.

3- There is another relation among classes besides inheritance (see Figure 9), some objects have object instance variables from other classes (the "has a" relation).

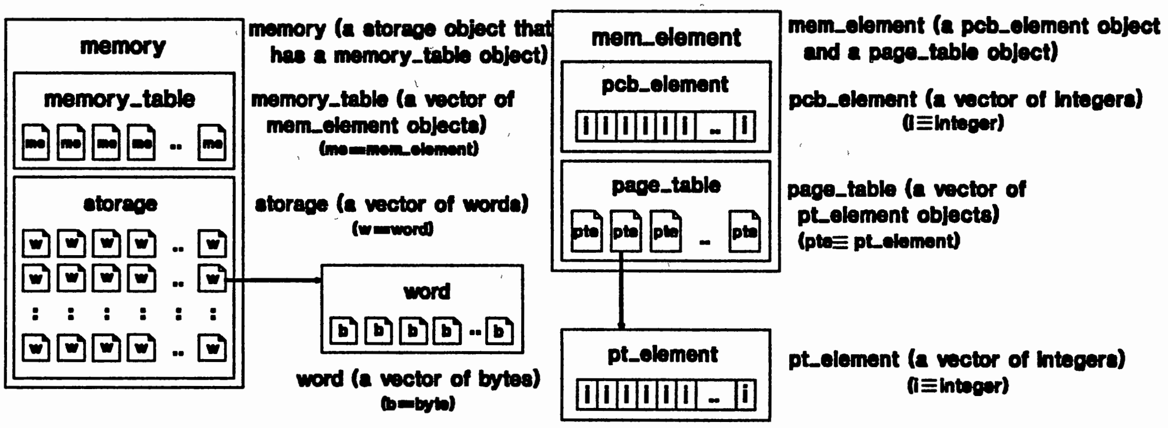


Figure 9. Relations among classes

For example, an object of the class storage has an array of word objects, an object of the class mem_element has a pcb_element object and the page_table object has an array of pt_element objects. This case is more complicated in the memory object which has a memory_table object, array of mem_element objects, and its body is an array of word objects.

4.4 Communication among Objects

The processing of each prototype system using the package is based on the communication among its objects as outlined below (see Figure 10).

1- The class loader and its interaction with the class memory and instances of the class memory:

- A loader object interacts with a memory object by calling loader.load(jobs_file,memory).
- The load() function communicates with the memory element, i.e., memory_table, by using memory.put(vect) to write the new job information into a memory_table element, i.e., pcb_element.
- The load() function also uses the write() function to communicate with the memory body to write a new word into the memory location by using the overloading operator = in the class word.

2- The CPU communicates with the memory to find a ready job from the memory_table and calls its inst_set object to execute the ready job's instructions one by one from the memory body.

3- The inst_set communicates with the CPU's registers and the memory body during each instruction's execution.

4- The Debugger communicates with the ins_set to receive its needed information about the current instruction to be displayed to the user of the package.

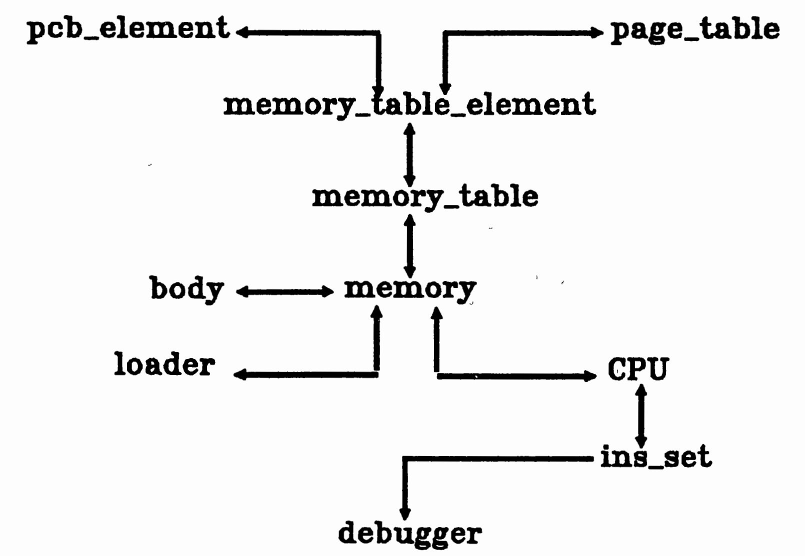


Figure 10. Communication among classes.

4.5 Interface

A default debugger was implemented to serve as an interface to the prototype system. The inheritance client can design his/her own debugger as needed.

The default debugger (see Figure 11) has four windows: REGISTER window, which displays the current register values, INST. INFO. window which explains the current instruction, JOB INFO. window which contains general information about the job, and Options window which contains user options.

Since the class debugger uses the class my_window, the inheritance client simply can overload it to add more features or create his/her new customized debugger by using the class my_window.

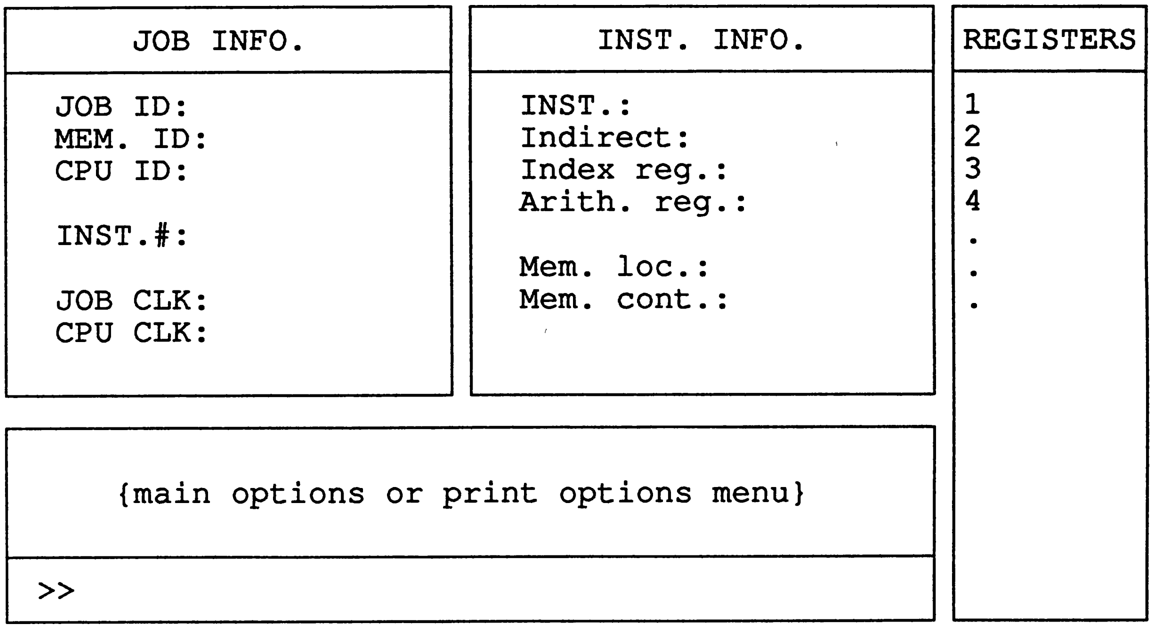


Figure 11. The debugger interface

The class my_window is easy to use because of the following reasons: Its constructor has the number of its variables, it has set() method to set the location of each variable, and to update a variable, it just needs to call the update() method with the variable number and its new value.

By using parallel processing functions we can run both parts of the package in parallel. While the debugger is displaying for the user a job's execution steps, the main program can execute another job and prepare its execution information in a special file for the debugger.

4.6 Help Option

Since this program constitutes a package, it should have some documentation as a help option. A user can use the help option to choose the class that (s)he needs to know about and the package will display a window containing information about the class and its methods. In the current implementation, help is a separate program as an application of using the class my_window from the package's classes.

4.7 Stochastic Processes and Queueing

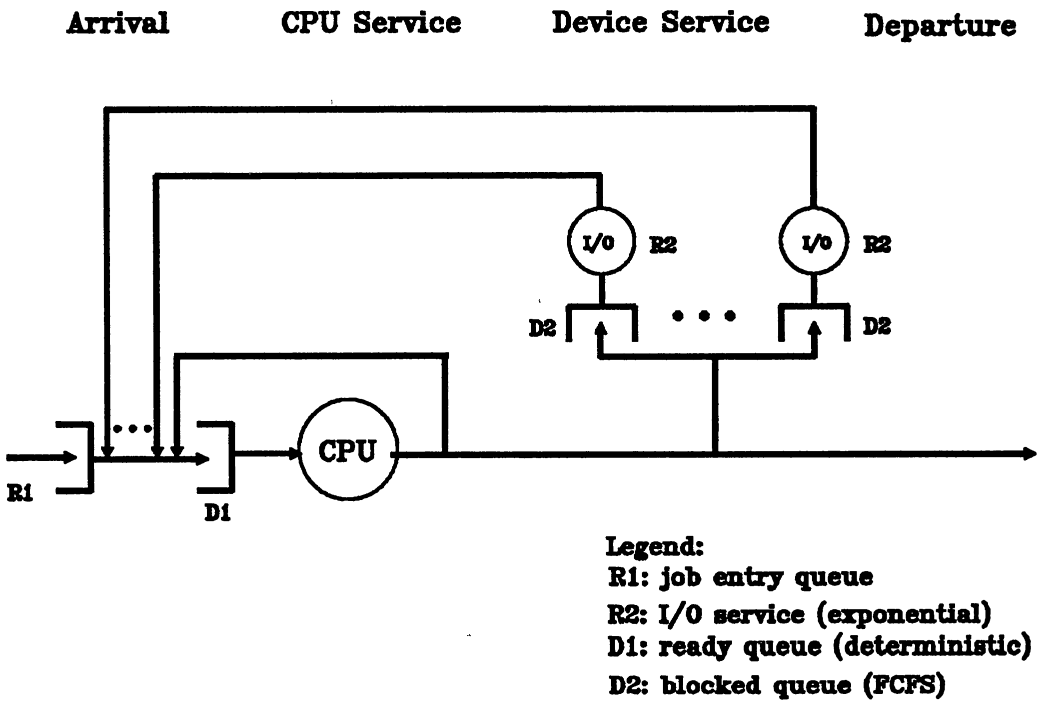


Figure 12. A queueing model of the dynamics of the current implementation of the package

It is important for operating system designers to effectively analyze how a system manages its resources. Therefore, users of the package need to be able to monitor and evaluate the performance of the systems they simulate to tune and streamline how the system uses its resources.

In order to make the simulation package more realistic, a pseudo-random-number generator class has been included [Jhun92]. The code for this class is listed in Appendix D. This class has "inter-arrival times" and "service times" methods..

Figure 12 shows a queueing model of the package. In this figure, queue R1 is used to hold the incoming jobs. The arrival times of the jobs is generated with the help of exponentially distributed inter-arrival times using the pseudo-random-number generator object. The distribution of the I/O service times is exponential. The deterministic queue D1, is the queue for ready jobs. Various scheduling mechanisms can be used on the D1 queue. The deterministic queues labeled D2 in Figure 12, are the I/O servers queues that use a FIFO scheduling mechanism.

4.8 Object-Oriented Approach

Object-oriented programs usually concentrate only on the inheritance relation among classes. However, it is clear from this package that using objects elements in a class maximizes some of the object-oriented programming advantages such as encapsulation.

This relation is not less important than inheritance. This idea is useful in conventional languages, using structures inside structures, but it is more important in object-oriented languages because of the natures of objects themselves. It is realistic to built new objects from different simpler objects with different behaviors. The unique behavior of each new object will be partially based on the result of its components' behaviors.

During the implementation of classes an attempt was made to:

- have as much overloading for the classes' constructors as we can to have more flexibility to meet the user needs;
- have as much overloading for the operators as we can to simplify the use of classes;
- avoid using friend classes as much as we can because it affects the encapsulation of the classes; and
- have as few arguments as possible in the methods to give the programmer more freedom to change his/her class implementations without affecting the application code.

CHAPTER V

EVALUATION

The package has been used in class project of the graduate level operating system course in the Computer Science Department in Oklahoma State University during Spring Semester 1992.

5.1 Using the Package

About 35 students in the class used the package for the course project which consisted of three phases. The goal of the first phase was to give students a chance to become familiar with the package and its use as a new way to simulate (i.e., to write simulation programs). The first phase consisted of simulating a simple machine with a simple operating system to execute one assembly job.

The second phase of the course project consisted of simulating a uniprocessor multiprogrammed machine with input and output spooling disks to execute a batch file of about 50 jobs. The goal in this phase was to find which configuration (i.e., memory size, spooling disk size, quantum size, compaction interval, etc.) of the machine/operating system will give the best system utilization (i.e., cpu, memory, and spooling disk utilization, among other things). There were a total of over 1000 configurations to be compared. The third phase of the course project consisted of comparing uniprocessor and dual processor machine simulations using stochastic arrival patterns and service queues.

5.2 Lessons Learned

During the use of the package in the graduate-level operating system course, three types of problems were encountered. The first type of problems was due to the fact that students were generally not familiar with this type of programming, which involved using the ready-made elements of a package, in their C code. Familiarity with C++ was not a prerequisite for the course. 34 out of 35 students used C as their language of simulation. One student used C++. Because of hiding the implementation details of the package, the clean design of the interface, and the capability of C++ for incorporating routines written in other languages, knowledge of C++ did not appear to be a significant advantage. By separating the package elements' interface and implementation, and providing small examples in the package's documentation and project specification, most of the students in the class were able to use the package elements and their different methods relatively easily.

The second type of problems was rooted in the students' belief that errors and bugs in their simulations were a result of bugs in the package code itself, which they could not access. This is a common type of problem that routinely occurs when dealing with a new and essentially untested software package. In fact, the number of real bugs found during the semester was unexpectedly low (as few as three bugs). So students gradually were convinced that the package code was robust enough to be trusted, specially when they found out that other students had not had similar problems in their simulation programs.

The third type of problems encountered was because of the platform used for the implementation of the package and for the course project. The package had been tested under ULTRIX on a VAX 8350; but during the course, the students used the package under a derivation of UNIX V (DYNIX/ptx) on a Sequent S81 with twenty four 80386-20MHz processors. The number of different configurations of the simulated machines that each student had to execute for the performance study part of the second phase of the course project was more than 1000. As a result of the execution of the 35 students' simulation programs basically at the same time as foreground jobs, there was a high rate of swapping between the main memory and the hard disks of the platform machine. This situation caused a serious degradation of the platform machine's performance for all users including interactive users. During the same semester, the platform machine was being used for several other programming courses as well. Although this problem was partially solved by limiting the number of configurations each student could execute simultaneously, the problem still persisted at a reduced level of intensity. It should be added that other than the memory problems, the platform machine performed very well. It seems that more memory on the platform machine (Sequent S81) will enable it to match the high processing power of its 24 processors.

CHAPTER VI

SUMMARY AND FUTURE WORK

The package was tested under ULTRIX on a VAX 8350 and under a derivation of UNIX V (DYNIX/ptx) on a Sequent S81 with twenty four 80386-20MHz processors. With minor changes, it can be (and in fact it has been) used under DOS on a personal computer.

6.1 Summary

Because of the popularity of the object-oriented design and programming, object-oriented operating systems are becoming increasingly more important. They tackle the problem of operating system complexity. This thesis has addressed the need for an object-oriented package (a prototyping system) to help simulate existing systems, and to prototype conventional as well as innovative architectures and operating systems.

Implementing a simulation package generally means that the designer of the package may have to implement a large number of features which may not be used in any one simulation. So, for a single simulation application, it may be faster to implement the application using a conventional language than by implementing and then using any simulation package. However, if there are a large number of simulation applications, then an object-oriented package, similar to the one described in this thesis, should prove more economic in terms of the time and programming effort involved.

A software package consisting of a collection of classes was created to give the user the ability to prototype various operating system/architecture models. The package is designed to give its user the ability to create complicated models (perhaps consisting more than one memory, loader, and/or cpu). A debugging option is included in the package to give its user the ability to trace, through different windows, jobs' execution.

Parallel processing is introduced in the package implementation by executing some of the main components such as the debugger and the help option in parallel with the main simulation. In its current implementation, the package can simulate multiprocessors systems without inter-process communication between the processors.

The static relation between package classes in the inheritance hierarchy is discussed in addition to the communication among package objects during a simulation execution. Since the prototyping system, i.e., the package, has been implemented as a collection of classes utilizing the object-oriented paradigm, a typical simulation application's code typically consists of only a few lines, as our initial testing with the class projects in a graduate-level operating system class suggests.

6.2 Future Work

The package is currently being used to compare different architectural approaches such as RISC, CISC, and microprogramming. The object-oriented nature of the package with its inherent support for (multiple) inheritance, the potential for reuse, and its ability to model different approaches to parallelism provides the possibility of extending the range of the systems to be simulated to multiprocessors, parallel processors, and even distributed systems.

Different Sequential and parallel systems with no inter-process communication have been successfully simulated using the package. The default debugger has proven to be flexible enough to be used to debug the package itself, and to give the users of the package better understanding of their jobs' execution.

Many features can be added to the package. Synchronization among the simulations' components is one of the important features that can be added to the package. Other types of memory management involving virtual memory and cache memory also need to be added to the package classes. The current implementation of the schedular is shared between the memory and the cpu classes in the package. Introducing a separate schedular class will improve the encapsulation of the scheduling functions.

Adding a natural language interface to the package will be a great help for nonprogrammers to be able to use the package. Adding the ability to control the level of details of the simulation can give the user the chance to avoid unneeded overhead in the simulation. Adding different types of debuggers can help meet the different needs of package users for different types of environments that need to be simulated. Adding a database system will give the package the ability to convert the data in its profile files of the simulation elements to a useful and easy-to-understand piece of information which can help the user perform the required analysis.

Another direction is to use new C++ features such as exception handling, or new object-oriented features such as reflection in the package. Also, the package can benefit from a task library, which is available on the Sequent S81 (the C++ task library), to arrange for parallel simulation of multiprocessor architectures.

REFERENCES

[Agha86]
G. Agha, *ACTORS: A Model of Concurrent Computation in Distributed Systems*, The MIT Press, Cambridge, MA, 1986.

[Ancona91]
M. Ancona, "Inheritance and Subtyping", *Proceedings of the 1991 Symposium on Applied Computing (SAC 91),* IEEE Computer Society Press (V. Kumar and E. A. Unger, Eds), Kansas City, MO, pp. 382-388, April 1991.

[Birrell89]
A. P. Birrell, "An Introduction to Programming with Threads", Technical Report, Digital Systems Research Center, Palo Alto, California, 1989.

[Borning86]
A. H. Borning, "Classes versus Prototypes in Object-Oriented Languages", *Proceedings of the Fall Joint Computer Conference*, Dallas, Texas, pp. 36-40, November 1986.

[Booch90]
G. Booch and M. Vilot, "The Design of the C++ Booch Components", *Proceedings of the Object-Oriented Programming Systems, Languages, and Applications Conference (OOPSLA90/ECOOP)*, Ottawa, Canada, pp. 1-11, October 1990.

[Cahill91]
V. Cahill and A. Kramer, "OISIN: Operating System Support for Objects in a Distributed Environment", *IEEE Computer Society Technical Committee on Operating Systems and Application Environments*, vol. 5, no. 1, pp. 4-8, Spring 1991.

[Corradi87]
A. Corradi and L. Leonardi, "An Environment Based on Parallel Objects: PO", *Proceeding of the IEEE Phoenix Conference on Computers and Communications*, Scottsdate, Arizona, February 1987.

[Corradi90]
A. Corradi and L. Leonardi, "Parallelism in Object-Oriented Programming Languages", *Proceedings of the 1990 International Conference on Computer Languages*, New Orleans, LA, pp. 71-80, March 1990.

[Finkel87]
R. A. Finkel, "Lightweight Tasks under UNIX", Technical Report Number 103-87, University of Kentucky, Lexington, KY, 1987.

[Finlayson91]
R. S. Finlayson, "Object-Oriented Operating Systems", *IEEE Computer Society Technical Committee on Operating Systems and Application Environments*, vol. 5, no. 1, pp. 17-21, Spring 1991.

[Gautron91]
P. Gautron, "Porting and Extending the C++ Task System with the Support of Lightweight Processes", *Proceedings of the USENIX C++ Conference*, Washington, DC, pp. 137-146, April 1991.

[Gorlen87]
K. Gorlen, "An Object-Oriented Class Library for C++", *Proceeding of the C++ Workshop*, USENIX Association, Santa Fe, NM, November 1987.

[Hassan92]
K. M. Hassan and M. H. Samadzadeh, "An Object-Oriented Environment for Simulation and Evaluation of Architectures", *Proceedings of the IEEE 25th Annual Simulation Symposium* in Conjunction with *The 1992 SCS Simulation Multiconference*, Orlando, FL, pp. 91-97, April 1992.

[Jhun92]
I. Jhun, K. M. Hassan, and M. H. Samadzadeh, "Simulation of a Computing Environment Using Stochastic Processes and the Object Oriented Technology", *Proceedings of the 23rd Annual Pittsburgh Conference on Modeling and Simulation*, Pittsburgh, PA, April 1992.

[Kirkerud89]
B. Kirkerud, *Object-Oriented Programming with Simula*, Addison-Wesley Publishing Company, Inc., Wokingham, England,1989.

[Koivisto91]
J. Koivisto and J. Malka, "OTSO - An Object-Oriented Approach to Distributed Computation", *Proceedings of the USENIX C++ Conference*, Washington, DC, pp. 163-178, April 1991.

[Martin91]
B. Martin, "The Separation of Interface and Implementation in C++", *Proceedings of the USENIX C++ Conference*, Washington, DC, pp. 51-63, April 1991.

[Nygaard86]

K. Nygaard, "Basic Concepts in Object Oriented Programming", *ACM SIGPLAN Notices*, vol. 21, no. 10, pp. 128-135, October 1986.

[Ralston83]

A. Ralston and D. Reilly, Jr., Encyclopedia of Computer Science and Engineering, Second Edtion, Van Nostrand Reinhold Publishing Company, Inc., NewYork, NY, 1983.

[Russo91a]

V. F. Russo, "Object-Oriented Operating System Design", *IEEE Computer Society Technical Committee on Operating Systems and Application Environments*, vol. 5, no. 1, pp. 34-38, Spring 1991.

[Russo91b]

V. F. Russo, "Process Scheduling and Synchronization in the Renaissance Object-Oriented Multiprocessor Operating System, *Proceedings of the Second USENIX Symposium on Distributed and Multiprocessor Systems (SEDMS II)*, Atlanta, GA, pp. 117-132, March 1991.

[Santo91]

M. D. Santo and W. Russo, "The ENSEMBLE System: Concurrent Programming on a Personal Computer", *ACM SIGPLAN Notices*, vol. 26, no. 2, pp. 99-108, February 1991.

[Sarkar89]

V. Sarkar, *Partitioning and Scheduling Parallel Programs for Multiprocessors*, The MIT Press, Cambridge, MA, 1989.

[Shapiro91]

M. Shapiro, "Object-Support Operating Systems", *IEEE Computer Society Technical Committee on Operating Systems and Application Environments*, vol. 5, no. 1, pp. 39-42, Spring 1991.

[Snyder86]

A. Snyder, "Encapsulation and Inheritance in Object-Oriented Programming Languages", *Proceedings of the Object-Oriented Programming Systems, Languages, and Applications Conference (OOPSLA86)*, Portland, OR, September 1986.

APPENDICES

APPENDIX A

GLOSSARY

Asynchronous Parallel Machine: A computer for simultaneous processing of two or more portions of the same program on two or more processing units [Ralston83].

Coroutines: A mechanism provided in ENSEMBLE [Santo91] which is basically the same as Modula-2 coroutines.

Designer: The designer of an object-oriented System.

General-Purpose Machine: A term used to characterize the capabilities of a computer to be used for a wide variety of tasks [Ralston83].

Granularity: The average size of a sequential computation unit in a program without inter-processor synchronization or communication.

Heavyweight Process: A process that has its own resources of time (execution and linking time), space (virtual store), and external environment (access to disks and files).

Inheritance Client: A user of the Package that inherits new classes from the package classes.

Lightweight Process: A process that does not create new mapping tables during its calls nor needs special instructions during switching to another process.

Object Client: A user of the package that instantiates objects from the package classes.

Prototype-Based Languages: Languages where the only way to make a new object is to make a complete copy of an existing object, copying both state and behavior [Borning86].

Scalability: The ability of a multiprocessor to have a linear increase in speed with an increase in the number of processors under the assumption that the program has sufficient parallelism and a large enough granularity.

Subtyping: The rules by which objects of one type (class) are determined to be acceptable in contexts expecting another type [Snyder86].

Tasks: Lightweight processes that share address spaces [Finkel87].

Thread: A single sequential flow of control.

Trademark Information

DYNIX/ptx is a registered trademark of Sequent Computer System, Inc.

Sequent is a trademark of Sequent Computer System, Inc.

UNIX is a registered trademark of AT&T

iPSC and **Hypercube** are trademarkes of Intel Corporation

VAX is a registered trademark of Digital Equipment Corporation

APPENDIX B

MAIN ELEMENTS OF THE PACKAGE

Some examples of the basic classes in the package and some of their important operations are included in this appendix.

1- The class vect is a vector of integers. This class is the parent class for some other classes in the package such as pcb_element and pt_element. Important operations: vect(), vect(int), vect(int*,int), and print().

2- The class hex_digit is a vector of four bytes. This class is used as the lowest-level class in the package systems' implementation. Important operations: hex_digit(), hex_digit(int), hex_digit(char), and print().

3- The class byte is a vector of 8 bits. Important operations: byte(), assign(hex_digit, hex_digit), and print().

4- The class word is a vector of bytes (the exact number of bytes depends on the system being simulated). Important operations: word(), print(), and operator =(word).

5- The class storage is a vector of words. This class is the parent class of class memory and can be used as the parent class of class disk in the package. Important operations: storage(), storage(int), write(int,word), free(), and release(int).

6- The class pcb_element is a subclass of the class vect. Some of its new operations in addition to the operations of the class vect are: id(), length(), and m_id().

7- The class pt_element is a subclass of the class vect. Some of its new operations in addition to the class vect operations are: valid(), reference(), and modified().

8- The class memory_table is a vector of mem_element objects. Important operations: memory_table(), memory_table(int), operator[](int), print(), put(&vector), and get(int).

9-The class memory is a special kind of storage with its own functions and its memory_table object. Important operation: memory(), memory(int,int), put(vect), get(int), and dump().

10- The class inst_set contains a set of assembly instructions that the system can execute. Important operation: decode(word) and execute(word).

11- The class cpu executes each job's instructions using an object of class inst_set whose type is specified in its constructor. Important operations: cpu(&int_set) and run_job_from(memory).

12- The class loader is a processing class to handle the loading of jobs in memory. Important operations: load(jobs_file,memory).

APPENDIX C

PROGRAM LISTING

```
/**********************************************************/
     The package attempts to give its user the flexibility to
create a model of the machine (s)he needs. The package
implementation gives the user enough flexibility to prototype
conventional, parallel, and object-oriented systems.  The
package uses the object-oriented approach to contain classes
as basic encapsulated components, which the package user can
use to build his/her system.  These classes include: hex_
digit, byte, word, registers, storage, memory, disk,
page_table, memory_table, loader, instructions, cpu, and
clock.
     In the following program listing, each class
documentation consists of: a class header documentation,
listing of the class.h file, and listing of the class.c file.
     The class header documentation includes: the class name,
the class variables, and the listing of the class operations.
The class.h file contains the declaration of the class
variables and operations. The class.c file contains the
implementation of the class operations.
/**********************************************************/

/**********************************************************/

CLASS: s_clock

int time;                        the clock value
int old;                         the clock value at
                                 last t_time call

Operations:

 1- constructor s_clock()        initialize to 0
 2- constructor s_clock(int)     initialize to a value
 3- void tick()                  increment by 1
 4- int t_time()                 store the time and
                                 return it
 5- int now()                    return the time without
                                 storing it
 6- void set()                   reinitialize to 0
 7- void set(int)                reinitialize to a value
```

```
 8- int past()                    return the time at last
                                  t_time() call
 9- void print()                  print the time to stdio
10- void print(FILE*)             print the time to a file

/**********************************************************/

#ifndef CLOCK
#define CLOCK

class s_clock {
     int time;
     int old;
public:
     s_clock();
     s_clock(int);
     void tick();
     int t_time();
     int now();
     void set();
     void set(int);
     int past();
     void print();
     void print(FILE*);
};
#endif

/**********************************************************/

#include <stream.h>
#include "clock.h"

s_clock::s_clock() { time=0;old=0; }
                         //initialize clock to 0
s_clock::s_clock(int i) { time = i; old=0; }
                         //initialize clock to value i
void s_clock::tick() { time++; }
                         //increment the clock by 1 clock
                         //cycle
int s_clock::t_time() { old = time; return(time); }
                         //return the current clock
                         //and store it in old
int s_clock::now() { return(time); }
                         //return the current clock
void s_clock::set() { time = 0; old=0; }
                         //reinitialize clock to 0
void s_clock::set(int i) { time = i; old=0; }
                         //reset clock value to i
int s_clock::past() { return(time-old); }
                         //return the time past from last
                         //call to t_time operation call
inline void s_clock::print() { cout << time; }
                         //print clock value to stdout
```

```
inline void s_clock::print(FILE* f) {fprintf(f,"%d",time); }
                         //print the current clock value to
                         //a file

/************************************************************/

/************************************************************/

CLASS: vect

int* p;                        pointer to array of integers
int size;                      the size of the vector

Operations:

 1- constructor vect()             create int vector with
                                   size 16
 2- constructor vect(int)          create int vector with any
                                   size
 3- constructor vect(int*,int)     create int vector from an
                                   array
 4- int ub()                       give the upper limit of
                                   the vector (its size)
 5- operator [](int)               return an element from the
                                   vector
 6- operator = (vect)              assign two vectors
 7- void print()                   print the vector to the
                                   stdout
 8- ~vect()                        the class destructor

/************************************************************/

#ifndef VECTOR
#define VECTOR

class vect {
     int* p;
     int size;
public:
     vect();
     vect(int);
     int ub();
     int& operator [](int);
     void operator =(vect&);
     void print();
     ~vect();
};

#endif

/************************************************************/

#include <stream.h>
```

```
#include <stdlib.h>
#include "vect.h"

vect::vect() { size=16; p=new int[size];
                         //create integer array of size 16
               for (int i=0; i<16; i++) p[i]=0;
                         //initialize vector
                         //elements to 0
               }

vect::vect(int sz)
{
if (sz<=0) {             //if size is illegal
     cerr << "illegal vector size " << sz << "\n";
     exit(-1);
     }
size=sz;                 //size of vector = sz
p=new int[size];
for ( int i=0; i<sz; i++) p[i]=0;
                         //initialize vector elements to 0
 }

inline int vect::ub() { return(size-1); }
                         //return the maximum number of
                         //elements that can be in
                         //the vector
int& vect::operator [](int i)
{
     if (i<0 || i>ub()) {   //check for the boundary
          cerr << "illegal vector index: " << i << "\n";
          exit(-1);
          }
     return(p[i]);
}

void vect::print()
{
for (int i=0; i<size; i++) cout << p[i] << " ";
}

void vect::operator =(vect& v)
{
     int s = (size < v.size) ? size : v.size;
                         //check if new vector has a
                         //smaller size
     if (v.size!=size) cerr << "copy different size
vectors\n";
     for (int i=0; i<s; i++) p[i] = v.p[i];
                         //copy elements of the
                         //new vector in this
                         //vector elements
}
```

```
vect::~vect() { delete(p); }

/***********************************************************/

CLASS: hex_digit

char s;                          the hex_digit value
static int lb, int ub, char base[16]

Operations:

 1- constructor hex_digit()      initialize to '0'
 2- constructor hex_digit(char)  initialize to a
                                 character
 3- constructor hex_digit(int)   initialize to an integer
 4- char chr()                   return the value as
                                 character
 5- vect& bin()                  return the value in the
                                 binary format
 6- operator = (hex_digit)       assign from hex_digit
 7- operator = (char)            assign from character
 8- operator = (int)             assign from integer
 9- void print()                 print to stdout
10- void print(FILE*)            print to a file

/***********************************************************/

#ifndef HIX
#define HIX
#include "vect.h"

class hex_digit {
     static char base[17];
     char s;
     int int_val;
     int lb, ub;
     vect cnv;
public:
     hex_digit();
     hex_digit(char);
     hex_digit(int);
     char chr();
     int int_h();
     void operator = (hex_digit& );
     void operator=(char);
     void operator = (int );
     void print();
     void print(FILE* f);
     vect& bin();
};
#endif

/***********************************************************/
```

```
CLASS: byte

hex_digit h1,h2;                 byte contents in two
                                 hex_digits

int intp(int,int,int,int)
Operations:
 1- constructor byte()           create a byte (8 bits)
 2- int int_u()                  return integer value of 4
                                 upper bits

 3- int int_l()                  return integer value of 4
                                 lower bits

 4- int int_b()                  return integer value of the
                                 byte

 5- int m_int_u()                return absolute value of
                                 upper 4 bits

 6- int m_int_l()                return absolute value of
                                 lower 4 bits

 7- void assign(hex_d,hex_d)     assign 2 hex to a byte
 8- void operator =(byte)        assign to a byte
 9- void operator =(char*)       assign to string
10- void print()                 print to stdout

/***********************************************************/

#ifndef BYTE
#define BYTE

class byte {
     int intp(hex_digit& h, int f=0);
     hex_digit h1,h2;
     static char ss[3];
public:
     byte();
     void operator =(byte&);
     void operator =(char*);
     char* str();
     int int_u();
     int int_l();
     int m_int_b();
     int int_b();
     int m_int_u();
     int m_int_l();
     void assign(hex_digit& , hex_digit&);
};
#endif

/***********************************************************/

#include <stream.h>
#include <stdlib.h>
#include <string.h>
#include "vect.h"
#include "hex.h"
```

```
#include "byte.h"

byte::byte() { h1='0';h2='0'; } //initialize the byte to
                            //"00"
void byte::operator =(byte& b) {  h1=b.h1; h2=b.h2; }
                            //assign the contents of byte b
                            //(two hex_digits) to the
                            //contents of this byte

void byte::operator =(char* b) {
                    int len=0;
                    if(strlen(b) != 2)
                         {
                         cerr << "length != 2";
                         exit(-1);
                         }
                    else
                         {
                         h1=b[0]; h2=b[1];
                              //assign two characters to the
                          //contents of this byte
                         }
                    }

 int byte::int_u() {
          int val=h1.int_h();

          if ( val < 8 ) return(val);
          else return( val - 16 );
                         //if the value in the upper 4 bits
                         //is negative,return its complement
          }

 int byte::int_l() {
          int val=h2.int_h();

          if ( val < 8 ) return(val);
          else return( val - 16 );
                         //if the value in the lower 4
                         //bits is negative,return its
                         //complement
          }

 int byte::m_int_b() {
               int val2 = h1.int_h();
               int val1 = h2.int_h();
               return(val1+val2*16);
               }

 int byte::int_b() {
               int val2 = h1.int_h();
               int val1 = h2.int_h();
```

```
               if ( val2 >= 8 ) {
                         //if the value in the upper 4 bits
                         //is negative,return its complement
                         val2=15-val2;
                         val1=16-val1;
                         return(-(val1+val2*16));
                         }
               else return(val1+val2*16);
               }

 int byte::m_int_u() { return(h1.int_h()); }
 int byte::m_int_l() { return(h2.int_h()); }
 void byte::assign(hex_digit& x1, hex_digit& x2)
                         { h1=x1; h2=x2; }
 char* byte::str() { ss[0]=h1.chr();
               //convert the byte contents to string
                ss[1]=h2.chr(); ss[2]='\0'; return(ss); }

/***********************************************************/

CLASS: word

byte* wrd;                       array of bytes
int size;                        number of bytes in the
                                 word
Operations:

 1- constructor word()           create a word (4 bytes)
 2- void get_s(char*)            convert the word to string
 3- int int_w()                  return the integer value of
                                 the word
 4- byte operator[] (int)        return a byte from the
                                 word
 5- int operator +  (word)       add two words
 6- void operator = (word)       assign two words
 7- void operator = (int)        assign integer value to a
                                 word
 8- void print()                 print to stdout
 9- void print(FILE*)            print to a file

/***********************************************************/

#ifndef WORD
#define WORD
#include "vect.h"
#include "hex.h"
#include "byte.h"

class word {
     byte *wrd;
     int size;
public:
     word();
```

```
     byte& operator[] (int);
     void print();
     void print(FILE*);
     void prints(FILE*);
     void get_s(char*);
     int int_w();
     int operator +(word&);
     int operator *(word&);
     int operator -(word&);
     int operator /(word&);
     void operator =(word&);
     int operator =(char*);
     void operator =(int);
     ~word();
};
#endif

/***********************************************************/


#include <stream.h>
#include <stdlib.h>
#include <string.h>
#include "vect.h"
#include "hex.h"
#include "byte.h"
#include "word.h"

word::word() { size=4; wrd=new byte[4]; }
                         //create a 4_byte word
byte& word::operator[] (int i) { return(wrd[i]); }
                         //return a byte
int word::operator +(word& w) {
return((*this).int_w()+w.int_w()); }
                         //add the contents of two words
int word::operator *(word& w) {
return((*this).int_w()*w.int_w()); }
                         //multiply the contents of two
                         //words
int word::operator -(word& w) {
return((*this).int_w()-w.int_w()); }
                         //subtract the contents of two
                         //words
int word::operator /(word& w) {
return((*this).int_w()/w.int_w()); }
                         //divide the contents of two
                         //words

void word::operator =(word& w) {
                         //assign two words by assigning
                         //their bytes
          for (int i=0; i<4; i++)
          (*this)[i] = w.wrd[i]; }
```

```
void word::operator =(int vlu) {
                    int i=0,val=0;
                    hex_digit temp[2];
                    if (vlu < 0 )
                         //if the "vlu" is negative take the
                         //complement
                    val = 2147483647 + vlu +1;
                    else val = vlu;
                    for (int j=0; j <4; j++)
                         { //assign the "vlu" to the 4 bytes
                         for ( i=0; i<2; i++)
                              {
                              temp[i] = val % 16;
                              val = val /16;
                              }
                         (*this)[j].assign(temp[1],temp[0]);
                         }
                    if ( vlu < 0 ) {
                         //if "vlu" was negative let sign
                         //bit = 1
                         temp[1]=temp[1].int_h()+8;
                         (*this)[3].assign(temp[1],temp[0]);
                         }
                    }
word::~word() { if (wrd!=NULL) delete []wrd; }
void word::get_s(char *s)
{
     // convert integers to hex_digits

     hex_digit hl0(wrd[0].m_int_l());
     hex_digit hu0(wrd[0].m_int_u());
     hex_digit hl1(wrd[1].m_int_l());
     hex_digit hu1(wrd[1].m_int_u());
     hex_digit hl2(wrd[2].m_int_l());
     hex_digit hu2(wrd[2].m_int_u());
     hex_digit hl3(wrd[3].m_int_l());
     hex_digit hu3(wrd[3].m_int_u());

     // convert hex_digits to characters

     s[7] = hl0.chr();
     s[6] = hu0.chr();
     s[5] = hl1.chr();
     s[4] = hu1.chr();
     s[3] = hl2.chr();
     s[2] = hu2.chr();
     s[1] = hl3.chr();
     s[0] = hu3.chr();
     s[8]= ’\0’;
}

int word::operator =(char* s)
{
```

```
int len=0,i;
char tm[3];

if(strlen(s) != 8 )
     {
     cerr << "length of a word not equal to 8";
     exit(0);
     return(0);
     }
else
     {
     for( i=0; i<8; i+=2)
          {               //assign the string "s" characters
                          //to the word 4 bytes
          tm[0] = s[i];
          tm[1] = s[i+1];
          tm[2] = '\0';
          (*this)[3-(i/2)] = tm;
          }
     return(1);
     }
}

void word::print() {
          char ss[10];
          // convert to string to print
          strcpy(ss,wrd[3].str());
          strcat(ss,wrd[2].str());
          strcat(ss,wrd[1].str());
          strcat(ss,wrd[0].str());
          strcat(ss," ");
          printf("%s\n",ss);
          }

void word::prints(FILE* f) {
          char ss[10];
          // convert to string to print
          strcpy(ss,wrd[3].str());
          strcat(ss,wrd[2].str());
          strcat(ss,wrd[1].str());
          strcat(ss,wrd[0].str());
          fprintf(f,"%s",ss);
          }

void word::print(FILE* f) {
          char ss[10];
          // convert to string to print
          strcpy(ss,wrd[3].str());
          strcat(ss,wrd[2].str());
          strcat(ss,wrd[1].str());
          strcat(ss,wrd[0].str());
          strcat(ss," ");
          fprintf(f,"%s",ss);
```

```
          }

int word::int_w() {
               int val0=(*this)[0].m_int_b();
               int val1=(*this)[1].m_int_b();
               int val2=(*this)[2].m_int_b();
               int val3=(*this)[3].m_int_b();

               if ( val3 >= 8 ) {
                    //if the sign bit = 1 take the
                    //complement to find the negative value
                    val0=256-val0;
                    val1=255-val1;
                    val2=255-val2;
                    val3=255-val3;

return(-(val0+val1*256+val2*65536+val3*16777216));
                    }
               else
return(val0+val1*256+val2*65536+val3*16777216);

     }

/**********************************************************/


CLASS: s_register

Operations:

 1- constructor s_register()    create a word with the
                                default size (4_byte)
 2- word get()                  return the contents of the
                                register
 3- void put(word)              put a word in the register

/**********************************************************/

#ifndef REGISTER
#define REGISTER

class s_register : public word {
public:
     s_register();
     word& get();
     void put(word&);
};
#endif

/**********************************************************/

#include <stream.h>
#include <stdlib.h>
```

```
#include <string.h>
#include "vect.h"
#include "hex.h"
#include "byte.h"
#include "word.h"
#include "register.h"

s_register::s_register() : word() {}
inline word& s_register::get() { return(*this); }
void s_register::put(word& w) { this->operator=(w); }

/**********************************************************/

CLASS: pcb_element

s_register *rg;               array of process registers

Operations:

 1- constructor pcb_element() create int vector size 16
 2- int& id()                 return job id
 3- int& loc()                return job location in memory
 4- int& start()              return first instruction in
                              the job
 5- int& length_w()           return job length in words
 6- int& trace()              return job trace flag
 7- int& state()              return job state - ready,
                              blocked, etc
 8- int& pc()                 return job's recent
                              instruction
 9- void operator=(vect&)     assign a job information
                              vector

/**********************************************************/

#ifndef TABLES
#define TABLES

class pcb_element : public vect {
     s_register *rg;
public:
     pcb_element();
     int& id();
     int& loc();
     int& start();
     int& length_w();
     int& trace();
     int& pc();
     int& state();
     int& l_id();
     int& m_id();
     int& c_id();
     int& pg_sz();
```

```
     int& rd_nxt();
     int& bl_nxt();
     int& ea();
     s_register** reg();
     void operator=(vect&);
     ~pcb_element();
};

/**********************************************************/

CLASS: pt_element

Operations:

 1- constructor pt_element() create int vector of size 5
 2- int& valid()             return page valid or not
 3- int& resident()          return page is resident or not
 4- int& modified()          return page is modified or not
 5- int& reference()         return is referenced or not
 6- int& address()           return real address of page

/**********************************************************/

class pt_element : public vect {
public:
     pt_element();
     int& valid();
     int& resident();
     int& modefied();
     int& referance();
     int& frame();
     ~pt_element();
};

/**********************************************************/

CLASS: page_table

pt_element *tbl;                 array of pt_elements
int size;                        size of the array

Operations:

 1- constructor page_table()     create an array of 16
                                 pt_elements
 2- constructor page_table(int)  create an array of
                                 pt_elements
 3- pt_element& operator[] (int) return a pcb_element

/**********************************************************/

class page_table {
     pt_element* tble;
```

```
protected:
     int size;
public:
     page_table();
     page_table(int);
     pt_element& operator[] (int);
     ~page_table();
};

/**********************************************************/

CLASS: mem_table_elem

Operations:

 1- constructor mem_table_elem()   create pcb_element and
                                   page_table
 2- void operator=(vect)           assign job information to
                                   the pcb_element

/**********************************************************/

class mem_table_elem : public pcb_elem,public page_table {
public:
     mem_table_elem();
     mem_table_elem(int);
     void operator=(vect&);
     pt_element& pt(int);
     ~mem_table_elem();
};

/**********************************************************/

CLASS: memory_table

mem_table_elem* table;          array of mem_table_elm
int size;                       size of the array

Operations:

 1- constructor mem_table()     create mem_table
                                array of 16 mem_table_elem
 2- constructor mem_table(int)  create mem_table
                                array of mem_table_elem
 3- pcb_element& operator[](int)return pcb_element contents
 4- void print()                print all pcb's contents
 5- int put(vect)               add a job information in
                                the pcb_element
 6- pcb_element& get(int)       get job's pcb_element
                                  information

/**********************************************************/
```

```
class mem_table {
     mem_table_elem* table;
     int size;
public:
     mem_table();
     mem_table(int);
     mem_table_elem& operator[] (int);
     void print();
     int put(vect&);
     pcb_element& get(int);
     void freet(int);
     int pcb(int);
     ~mem_table();
};
#endif

/*********************************************************/

#include <stream.h>
#include <stdlib.h>
#include <string.h>
#include "vect.h"
#include "hex.h"
#include "byte.h"
#include "word.h"
#include "register.h"
#include "tables.h"

pcb_element::pcb_element() : vect(16) {rg=new
s_register[16];}

//following operations return and/or assign elements of the
//pcb_element vector

int& pcb_element::id() { return( (*this)[0] ); }
int& pcb_element::loc() { return( (*this)[1] );}
int& pcb_element::start() { return( (*this)[2] );}
int& pcb_element::length_w() { return( (*this)[3] );}
int& pcb_element::trace() { return( (*this)[4] ); }
int& pcb_element::pc() { return( (*this)[6] ); }
int& pcb_element::state() { return( (*this)[7] ); }
int& pcb_element::l_id() { return( (*this)[8] ); }
int& pcb_element::m_id() { return( (*this)[9] ); }
int& pcb_element::c_id() { return( (*this)[10] ); }
int& pcb_element::pg_sz() { return( (*this)[11] ); }
int& pcb_element::rd_nxt() { return( (*this)[12] ); }
int& pcb_element::bl_nxt() { return( (*this)[13] ); }
int& pcb_element::ea() { return( (*this)[14] ); }
s_register** pcb_element::reg() { return( &rg ); }
void pcb_element::operator=(vect& v) { vect:: operator=(v);
}
pcb_element::~pcb_element() { delete [16]rg; }
```

```
/**********************************************************/

pt_element::pt_element() : vect(5) {}

//following operations return and/or assign elements of the
//pt_element vector

int& pt_element::valid() { return( (*this)[0] ); }
int& pt_element::resident() { return( (*this)[1] ); }
int& pt_element::modefied() { return( (*this)[2] ); }
int& pt_element::referance() { return( (*this)[3] ); }
int& pt_element::frame() { return( (*this)[4] ); }
pt_element::~pt_element(){ vect::~vect(); }

/**********************************************************/

page_table::page_table() { size = 64; tble = new
pt_element[size]; }
page_table::page_table(int i) { size = i; tble = new
pt_element[size]; }
pt_element& page_table::operator[] (int i) { return( tble[i]
); }
page_table::~page_table() { delete [size]tble; }

/**********************************************************/
mem_table_elem::mem_table_elem() : pcb_element(),
page_table() {}
mem_table_elem::mem_table_elem(int i) : pcb_element(),
pt_table(i) {}
void mem_table_elem::operator=(vect& v) { pcb_element::
operator=(v); }
pt_element& mem_table_elem::pt(int i) { return(page_table::
operator[](i)); }
mem_table_elem::~mem_table_elem()
{pcb_element::~pcb_element();}

/**********************************************************/

memory_table::memory_table() { size =64; table = new
mem_table_elem[size]; }
memory_table::memory_table(int i) { size=i; table = new
mem_table_elem[size]; }
mem_table_elem& memory_table::operator[] (int i) {
return(table[i]);
               //return the memory table element of the job
 }
void memory_table::print() { for (int i=0; i<size; i++)
table[i].print(); }
int memory_table::put(vect& j) {
               int i=0;
               while ( table[i].id() != 0 && i < size ) i++;
               if ( i ==  size ) { cerr << "no space\n";
                              return(-1);
```

```
                         }
               table[i] = j;
               return(i); //return the location of the
                         //mem_table_elem of the new job
               }
pcb_element& memory_table::get(int id) {
               int i=0;
               while ( table[i].id() != id && i < size )
               i++;
               if ( i == size )
                    {
                    cerr << "\njob " << id << " not here\n";
                    exit(-1);
                    }
               return(table[i]);
                    //return the mem_table_elem of the job
                    //with id="id"
               }

void memory_table::freet(int id) {
          int i=0;
          while ( table[i].id() != id && i < size ) i++;
          table[i].id()=0;
                    //release the mem_table_elem by
                    //assign its job id = 0
          }

int memory_table::pcb(int id) {
               int i=0;
               while ( table[i].id() != id && i < size )
               i++;
               if ( i == size )
                    {
                    cerr << "\njob " << id << " not here\n";
                    exit(-1);
                    }
               return(i);
                    //return the location of the pcb
                    //of the job with id="id"
               }

memory_table::~memory_table() { delete [size]table; }

/***********************************************************/

CLASS: storage

word* mem;                     array of words
int fragm;                     free words in the memory
int size;                      memory size

Operations:
```

```
 1- constructor storage()      create an array[256] of
                               words
 2- constructor storage(int)   create an array[int] of
                               words
 3- void write(int,word)       put a word into a location
 4- void write(word,word)      put a word into an address
 5- int free()                 return memory fragmentation
 6- void release(int)          release location from the
                               memory
 7- word operator[] (int)      return location contents
 8- word operator[] (word)     return address contents

/************************************************************/

#ifndef STORAGE
#define STORAGE

class storage {
     word *mem;
protected:
     int fragm;
     int size;
public:
     storage();
     storage(int);
     word& operator[] (int);
     word& operator[] (word&);
     void write(int , word& const );
     void write(word& const , word& const );
     int free();
     void release(int);
     ~storage();
};
#endif

/************************************************************/

#include <stream.h>
#include <stdlib.h>
#include <string.h>
#include "vect.h"
#include "hex.h"
#include "byte.h"
#include "byte.h"
#include "word.h"
#include "storage.h"

storage::storage() { size = 256; fragm = size;mem=new
word[size];}        //create a storage with size 256 words
storage::storage(int n) {
          if (n>2048) {cerr << "memory size max=2
M\n";exit(0);}
          size = n; fragm = size;mem= new word[size];}
```

```
                    //create a storage not larger than 4
                    //Mbyte
word& storage::operator[] (int i) { return(mem[i]); }
                    //return contents of storage location
"i"
word& storage::operator[] (word& w) { int i = w.int_w();
return(mem[i]); }   //return contents of storage location
                    //"w"
void storage::write(int loc, word& const w) { mem[loc] = w;
fragm--; }          //load a word in storage location "loc"

void storage::write(word& const loc, word& const w) { int
i=loc.int_w(); mem[i] = w; fragm--; }
                    //load a word in storage location "loc"
int storage::free() { return(fragm); }
                    //return external fragmentation
storage::~storage(){ delete [size]mem; }
void storage::release(int loc) { word init_w; mem[loc] =
init_w; fragm++;}   //release word in location "loc" in the
                    //storage

/**********************************************************/
```

CLASS: memory

memory_table* table

Operations:

1- constructor memory()	create a memory with size 256 of 16 pages
2- constructor memory(int s,int p)	create a memory with size s of p pages
3- int put(vect)	put job's information in the pcb_element
4- pcb_element get(int)	return job's information from the pcb_element
5- pcb_element ready_job()	return first ready job's information
6- void dump()	dump memory contents to stdout
7- void block(int,int ,s_register*)	blocking a job
8- void freem(int)	release job space in the memory
9- word& fetch(s_job* job)	find a ready job to be executed
10- int put_page(char(*)[9],int ,int ,int)	load a job page into the memory
11- int put_page(word* ,int ,int ,int)	load a job page into te memory
12- int pg_size()	return memory page size

```
13- ~memory()                          memory destructor

/*********************************************************/


#ifndef MEMORY
#define MEMORY
#include "word.h"
#include "register.h"
#include "tables.h"
#include "storage.h"
#include "job.h"

class frames { public:int jjob; int next_fr; };
                         //local class
class memory:public storage {
     static int m_id;
     s_job *jobs;
     frames *m_frame;
     memory_table *table;
     int frst_fr,lst_fr;
     int page_size,jbs_no;
     int my_id;
     int frj,lrj,fbj,lbj;
public:
     memory();
     memory(int);
     memory(int ,int );
     int id();
     int put(vect );
     pcb_element& get(int );
     int pcb(int );
     word& operator() (int ,int );
     void dump(FILE* );
     void block(int ,int ,s_register*);
     s_job* ready_job(int ,int ,int =5);
     void freem(int);
     word& fetch(s_job* job);
     int put_page(char(*)[9],int ,int ,int );
     int put_page(word* ,int ,int ,int );
     int pg_size();
     ~memory();
};
#endif

/*********************************************************/

#include <stream.h>
#include <stdlib.h>
#include <string.h>
#include "clock.h"
#include "vect.h"
#include "hex.h"
```

```
#include "byte.h"
#include "word.h"
#include "register.h"
#include "memory.h"

memory::memory():storage(256) {
          frj=lrj=fbj=lbj=-1;
                    //initialize ready and blocked queues to
                    //be empty
          table = new memory_table;
          my_id = ++m_id;
          jobs = new s_job[16];
                    //a table of information
                    //for current jobs in the
                    //memory
          m_frame = new frames[16];
                    //a table of free frames in the
                    //memory
          page_size=16;//the default page size is 16 words
          jbs_no=16;
          for (int i=0;i<15;i++) {
                    //initialize the free frames table
               m_frame[i].jjob=0;
               m_frame[i].next_fr=i+1;
               }
          m_frame[i-1].next_fr=-1;
          frst_fr=0;
          lst_fr=15;
          }

memory::memory(int i):storage(i) {
          frj=lrj=fbj=lbj=-1;
                    //initialize ready and blocked queues to
                    //be empty.
          table = new memory_table;
          my_id = ++m_id;
          jobs = new s_job[16];
                    //a table of information for current
                    //jobs in the memory.
          m_frame = new frames[16];
                    //a table of free frames in the memory
          for (int j=0;j<15;j++) {
                    //initialize the free frames table
               m_frame[j].jjob=0;
               m_frame[j].next_fr=j+1;
               }
          m_frame[j-1].next_fr=-1;
          page_size=i/16;
          jbs_no=16;
          frst_fr=0;
          lst_fr=15;
          }
```

```
memory::memory(int i,int jb):storage(i) {
          frj=lrj=fbj=lbj=-1;
                    //initialize ready and blocked queues to
                    //be empty.
          table = new memory_table(jb);
          my_id = ++m_id;
          jobs = new s_job[jb];
                    //a table of information for current
                    //jobs in the memory.
          m_frame = new frames[jb];
                    //a table of free frames in the memory
          for (int j=0;j<jb;j++) {
                    //initialize the free frames table
               m_frame[j].jjob=0;
               m_frame[j].next_fr=j+1;
               }
          m_frame[j-1].next_fr=-1;
          if ( i%jb!=0 ) {
                    //check if the given memory size and
                    //number of pages give correct page size
                    //for all pages
                    cerr << "memory declaration error\n";
                    exit(0);
                    }
          page_size=i/jb;
          jbs_no=jb;
          frst_fr=0;
          lst_fr=jb-1;
          }

inline int memory::id() { return(my_id); }

int memory::put(vect v) {
          int jid=v[0];
          v[9] = my_id;
          v[11] = page_size;
          v[12] = -1;
          v[13] = -1;
          int job_pg1=v[3];
                    //read page_size from the vector
          int rest1=v[3]%page_size;
          if (rest1>0) job_pg1++;
                    //calculate number of pages needed for
                    //the job
          if ( job_pg1>(fragm/page_size) )
                    //return -1 if no space available
                    //in the memory to load the job
                                    return(-1);
          int pcb_no=table->put(v);
                    //load the job information in a free
                    //mem_table_elem in the memory_table and
                    //return the pcb location
          if ( pcb_no==-1 )
```

```
                     //return -1 if no free  frames in
                    //memory
                            return(-1);

          jobs[pcb_no].live((*table)[pcb_no]);
          jobs[pcb_no].state()=1;
                    //make job state ready
          job_pg=job_pg1;
          for ( i=0;i<job_pg;i++) {
                    //initialize job pages to be valid,
                    //resident, not modified, and not
                    //referenced in its pcb
               ((*table)[pcb_no].pt(i)).valid()=1;
               ((*table)[pcb_no].pt(i)).resident()=1;
               ((*table)[pcb_no].pt(i)).modefied()=0;
               ((*table)[pcb_no].pt(i)).referance()=0;
               ((*table)[pcb_no].pt(i)).frame()=frst_fr;
               m_frame[frst_fr].jjob=jid;
               if ( frst_fr==-1 )
                    {table->freet(jid);return(-1);}
               frst_fr=m_frame[frst_fr].next_fr;
               }
          for ( int i=job_pg;i<64;i++)
                    //initialize the rest of pages to be
                    //invalid for this job
               (((*table)[pcb_no]).pt(i)).valid()=0;

          if ( frj==-1 ) frj=lrj=pcb_no;
                    //put the job pcb in the head of the
                    //ready queue if it is the first arrival
                    //job
          else {    //put the job at the end of the ready
                    //queue
               jobs[lrj].rd_nxt()=pcb_no;
               jobs[pcb_no].rd_nxt()=-1;
               lrj=pcb_no;
               }
          return(pcb_no);
          }

void memory::freem(int id) {
     int pcb_no=table->pcb(id);
     table->freet(id); //release mem_table_elem and
                     //pcb_element for the terminated job
                     //with id = "id"
     int job_pg=(*table)[pcb_no].length_w();
                        //calculate number of pages this job
                     //used
     int rest=(*table)[pcb_no].length_w()%page_size;
     if (rest>0) job_pg++;
     int i=0;
     for ( i=0;i<job_pg;i++) {
```

```
                    //release page frames of the terminated
                    //job from the memory
          if (frst_fr==-1) {
                    //put the free frame at the top of the
                    //free frames queue if it is no other
                    //free frames in the memory.
               frst_fr= ((*table)[pcb_no].pt(i)).frame();
               m_frame[frst_fr].next_fr=-1;
               }
          else {    //put the free frame at the end of the
                    //free frames queue
          int tmpt=frst_fr;
          frst_fr=((*table)[pcb_no].pt(i)).frame();
          m_frame[frst_fr].next_fr=tmpt;
          }
          fragm+=page_size;
                    //return released words to the available
                    //space in the memory
        }
     }

inline pcb_element& memory::get(int id)
                    //return the pcb of the job with id="id"
                 { return( table->get(id) ); }
inline int memory::pcb(int id) { return( table->pcb(id) ); }

word& memory::operator() (int jid,int adr) {
                    //return a word from the memory with a
                    //logical address="adr"
               return( (*this)[(jobs[pcb(jid)].vrt(adr))]);
}

void memory::dump(FILE* out) {
                    //print memory contents to a file
               int j=0;
               for (int i=0; i<size; i+=8)
                    {
                    fprintf(out,"%2.4x",i);
                    for ( j =0; j<8; j++)
                         {
                         fprintf(out," ");
                         (*this)[i+j].print(out);
                         }
                    fprintf(out,"\n");
                    }
               }

void memory::block(int jid,int EA,s_register* R) {
               int j_pcb=pcb(jid);
                    //store blocked job id
               jobs[j_pcb].ea()=EA;
                    //store the effective address of the
                    //blocked job
```

```
               (*(jobs[j_pcb].reg()))=R;
                    //store blocked job registers update
                    //blocked queue
               if ( fbj == -1 ) fbj=lbj=j_pcb;
                    //put blocked job pcb in the blocked
                    //queue
               else{
                    jobs[lbj].bl_nxt()=j_pcb;
                    jobs[j_pcb].bl_nxt()=-1;
                    lbj=j_pcb;
                    }
               }

s_job* memory::ready_job(int cid,int cpu_clk,int sd) {
               int fr;              //first ready job
               int br=-1;           //blocked job
               int fbr=-1;          //first blocked job
               int tmr,tm;
               if ( frj==-1 && fbj==-1 )
                                    //if both ready and
                                    //blocked queue are empty
                                    //return -1
                               return(NULL);
               else if ( frj == -1 ) {//if no ready jobs in
                                        the ready queue
                    tmr=fbj;        // pick first blocked job
                    tm=jobs[fbj].j_clk()-cpu_clk;
                    fr=fbj;
                    while (tmr!=-1) { //update blocked queue
                         if (jobs[tmr].j_clk()-cpu_clk<tm){
                               tm=jobs[tmr].j_clk()-cpu_clk;
                               fr=tmr;
                               fbr=br;
                               br=tmr;
                               tmr=jobs[tmr].bl_nxt();
                               }
                         else
                             {br=tmr;tmr=jobs[tmr].bl_nxt();}
                         }
               if (fr==fbj) fbj=jobs[fr].bl_nxt();
               else if (fr==lbj){ //if this is the last job
                    lbj=fbr;
                    jobs[lbj].bl_nxt()=-1;
                    }
               else
                    jobs[fbr].bl_nxt()=jobs[fr].bl_nxt();
                    jobs[fr].act(cid); //reactivate the job
                    jobs[fr].bl_nxt()=-1;
                    jobs[fr].c_id() = cid;
                    jobs[fr].t_time(); //set job clock
                    return(&jobs[fr]);
                    }
```

```
else {         //check if the ready queue if
               //it has jobs to be executed
     tmr=frj;
     tm=jobs[frj].length();
     fr=frj;
     if (sd==5) fr=frj;
               //if scheduling is FIFO pick
               //first ready job in the ready
               //queue
     else if(sd==7) {
               //if scheduling is longest job
               //first search to find the
               //longest job in the ready
               //queue
     while (tmr!=-1) {
          if (jobs[tmr].length()>tm){
               tm=jobs[tmr].length();
               fr=tmr;
               fbr=br;
               br=tmr;
               tmr=jobs[tmr].rd_nxt();
               }
          else
             {br=tmr;tmr=jobs[tmr].rd_nxt();}
          }
if (fr==frj)
          frj=jobs[fr].rd_nxt();
else if (fr==lrj){
               //if this job is the last job
               //in the ready queue make it
               //empty
     lrj=fbr;
     jobs[lrj].rd_nxt()=-1;
     }
else
     jobs[fbr].rd_nxt()=jobs[fr].rd_nxt();
          }
     else if(sd==6) {
               //if scheduling is shortest
               //job first search to find the
               //longest job in the ready
               //queue
     while (tmr!=-1) {
          if (jobs[tmr].length()<tm){
               tm=jobs[tmr].length();
               fr=tmr;
               fbr=br;
               br=tmr;
               tmr=jobs[tmr].rd_nxt();
               }
          else
             {br=tmr;tmr=jobs[tmr].rd_nxt();}
          }
```

```
                if (fr==frj) frj=jobs[fr].rd_nxt();
                else if (fr==lrj){
                          //if the job is the last job in the
                          //queue make it empty
                     lrj=fbr;
                     jobs[lrj].rd_nxt()=-1;
                     }
                else jobs[fbr].rd_nxt()=jobs[fr].rd_nxt();
                     }
                     jobs[fr].act(cid);//reactivate the job
                     if (sd==5) frj=jobs[frj].rd_nxt();
                     jobs[fr].rd_nxt()=-1;
                     jobs[fr].t_time();
                     return(&jobs[fr]);
                     }
                }

word& memory::fetch(s_job* job) {
          int pc = job->pcv();
                          //find the logical address of the
                          //next instruction to be executed
          if ( job->trace() == 1 )
               (*this)[pc].print(job->dbgf());
          return( (*this)[pc] );
          }

int memory::put_page(word* w,int pcb,int pg,int j)
          {
          ((*table)[pcb].pt(pg)).resident()=1;
                          //set the frame as used by the
                          //loaded job
int strt_addr=((*table)[pcb].pt(pg)).frame()*page_size;
          for ( int i=0;i<page_size;i++){
                          //load page word by word
                          (*this)[strt_addr+i]=w[i];
                          }
          fragm-=page_size; //take page from memory
          return(1);
          }
int memory::put_page(char w[][9],int pcb,int pg,int j)
          {
          ((*table)[pcb].pt(pg)).resident()=1;
                               //set frame as used by the
                           //loaded job
int strt_addr=((*table)[pcb].pt(pg)).frame()*page_size;
          for ( int i=0;i<page_size;i++)
                          //load the page word by word into
                          //the memory frame
               if (strlen(w[i])==8)
                          (*this)[strt_addr+i]=w[i];
               else break;
          fragm-=page_size; //decrement the free space in
                           //the memory by the used page size
```

```
          return(1);
          }
int memory::pg_size() { return(page_size);}
                             //return memory page size
memory::~memory(){           //memory destructor
     delete table;
     delete [jbs_no]jobs;
     delete [jbs_no]m_frame;
     }

/************************************************************/

CLASS: loader

Operations:

 1- int load(FILE* f, memory m)

 Try to read one job from "FILE f" to "memory m" by
reading a job header into a free pcb_element and the job
body into  the memory.

 if succeed, return 1
 if find error, return 2
 if eof, return -1

/************************************************************/

#ifndef LOADER
#define LOADER
#define LOADED 1
#define MEMFULL -1
#define RDERR -2

#include "vect.h"
#include "clock.h"
#include "hex.h"
#include "byte.h"
#include "word.h"
#include "register.h"
#include "tables.h"
#include "storage.h"
#include "job.h"
#include "memory.h"

class loader {
     static int l_id;
     int my_id;                     //loader id
public:
     int load(FILE* ,memory&);
     int id();
};
```

```
#endif

/**********************************************************/

#include <stream.h>
#include <string.h>
#include <stdlib.h>
#include "loader.h"
#define LOADED 1
#define MEMFULL -1
#define RDERR -2

int loader::id() {return(my_id);}
int loader::load(FILE* f,memory& m)
{
     char line[99][81],r_word[99][9];
                         //read job instructions as arrays
                         //of strings
     char buff[99][9];
     int loc,flag=1;
     vect v;
     my_id = ++l_id;

     if (fgets(line[0],80,f) != NULL)
                         //read job instructions
     {
sscanf(line[0],"%x%x%x%x%x%x",&v[0],&v[1],&v[2],&v[3],&v[4],
&v[5]);
     v[6] = v[2];
     v[7] = v[5];
     v[8] = id();
     loc = v[1];
     int pcb=m.put(v);   //put job data in a pcb
     int pz=m.pg_size();
     int no_of_cards = (v[3]/4)+ ( (v[3]%4)>0 );
                         //calculate number of cards
                         //in the job
     int no_of_pages = (v[3]/pz)+ ( (v[3]%pz)>0 );
                         //calculate number of pages
                         //needed by the job
     int j=0;
     for (int i=0; i< no_of_cards; i++)
                         //read job cards from the input
                         //files
          {
               fgets(line[i],80,f);
               sscanf(line[i],"%8s%8s%8s%8s\n",

r_word[0+j],r_word[1+j],r_word[2+j],r_word[3+j]);
               j+=4;      //four words on a card
          }
     for (i=0; i< no_of_pages; i++)
                         //load pages in memory
```

```
          {
          for ( j=0; j <pz; j++)
                         //load a page
               if (strlen(r_word[i*pz+j])==8) {
                    strcpy(buff[j],r_word[i*pz+j]);
                    strcpy(r_word[i*pz+j]," ");
                    }
               else break;
          if( (flag=m.put_page(buff,pcb,i,v[0]))==-1)
                         //if no space available in
                         //memory return fail
                                               return(flag);
          for (int k=0;k<pz;k++) strcpy(buff[k],"00000000");
          }
     }
     else flag=-2;
     return(flag);
}

/************************************************************/

CLASS: cpu

s_clock c, s_register* reg, ins_set *s

Operations:
 1- constructor cpu (ins_set)
 2- int run_job_from(memory m)
 - find a ready job from the memory PCB
 - prepare files for the job - debug, state, trace, and
 output files -
 - set the cpu clock
 - start the job execution by read its instructions from
 memory
 - call the ins_set element to decode the instruction
 - calculate the needed memory address for execution
 - call the ins_set to execute the instruction
 - increment its clock (quantum is 50 clock cycle)
 - if the job trace flag is on send trace information to the
 trace file
 - at the end change the job state to "Halt" in the
pcb_element
 - send appropriate information to its state file
 - return NOTDONE if job is blocked (because of
 read/quantum) and still running
 - return NOMORE if no more ready jobs in memory

/************************************************************/

#ifndef CPU
#define CPU

#include "clock.h"
```

```
#include "vect.h"
#include "hex.h"
#include "byte.h"
#include "word.h"
#include "register.h"
#include "memory.h"
#include "ins_set.h"
#include "ins_set1.h"

#define NOTDONE 4                   //executed job status
#define NOMORE -1
#define RUNTERR 2
#define DONE 3
#define RR 5                        //scheduler mode
#define SF 6
#define LF 7

class cpu {
     static int c_id;
     s_clock clk;                  //system clock
     s_register* reg;              //CPU registers
     ins_set *s;                   //CPU instructions
     FILE* trace;                  //CPU profile file
     int my_id,trflag;
     int rn,trm;                  //id of running and
                          //terminated jobs
     int scdlr;                   //scheduler mode
public:
     cpu(ins_set& ,int qn=50,int = 5,int =0);
     ~cpu();
     int id();
     int running();
     int terminated();
     int m_clock();
     int run_job_from(memory& );
};
#endif

/***********************************************************/

#include <stream.h>
#include <stdlib.h>
#include <string.h>
#include "cpu.h"

extern void itoa(int,char*);
cpu::cpu(ins_set& i,int qn,int sd,int trfl) {
          if ( sd<5||sd>7 ){     //check scheduler
               cerr << "Out of schedulers range\n";
               exit(0);
               }
          scdlr=sd;
          my_id = ++c_id;
```

```
          clk.set(23);      //set clock value > 1st job
                            //arrival time
          reg = new s_register[16]; //CPU registers
          s = &i;
          s->set_qntm(qn);
          trflag=trfl;
          char trcfile[10];
          strcpy(trcfile,"CPU_TR_");
          char tt[10];
          itoa(my_id,tt);
          strcat(trcfile,tt);
          if ( trflag!=0 ) {
               trace=fopen(trcfile,"w");
               fprintf(trace,"CPU_CLK\tJOB_ID\tJOB_CLK\n");
               }
          }
int cpu::m_clock()               //return system clock value
     { return(clk.now()); }
cpu::~cpu() {                    //cpu destructor
          if ( trflag!=0 ) {fclose(trace);
                    delete(trace); }
          }
int cpu::id() { return(my_id); }
int cpu::running() { return(rn); } //return current running
job id
int cpu::terminated() { return(trm); }
                         //return last terminated job
                         //id
int cpu::run_job_from(memory& m) {
               int A,B,ind,p,EA=0,flag=1;
               s_job* job;
               word DADDR,inst,w;

               job = m.ready_job(my_id,clk.now(),scdlr);
                         //receive a ready job
                         //from the memory to be
                         //executed
               if (job==NULL) { //if no more jobs in the
                         //memory, stop
                    cerr << "no more jobs\n";
                    return(-1);
                    }
               rn=job->id();     //store running job id
               if ( trflag!=0 ) {
                    while(job->j_clk()>clk.now()) {
                              clk.tick();
                fprintf(trace,"%d\t__\t__\n",clk.now());}
                    if (clk.now()!=0){
                         //first job context
                         //switching=2 clock cycles
                              clk.tick();
                              clk.tick();
                              }
```

```
fprintf(trace,"%d\t%d\t%d\n",clk.now(),job->id(),job->j_clk(
));
                    }
               EA = job->ea();  //read the first instruction
                              //effective address
               reg = (*(job->reg()));
                              //load job registers from
                              //the pcb to the CPU registers
                    while ( flag == 1 ) {
                              //if the job status is running
                              //execute next instruction
                               //fetch next instruction
                         inst=m.fetch(job);
                               //decode the instruction
                         s->decode(inst,ind,p,A,B,DADDR);
                               //calculate the effective
                              //address
                         if ( ind == 0 )
                              //not indirect addressing mode
                              EA=DADDR.int_w();
                         else
                      EA=m(job->id(),DADDR.int_w()).int_w();
                         if ( B != 0 )
                              //if it is indexing addressing
                              //mode
                              EA += reg[B].int_w();
                         if (EA>job->length()) {
                         cerr<<"page fault core dump\n";
                              m.dump(job->outf());
                              //dump memory contents to
                              //the output file
                              exit(0);
                              }
                         s->execute(EA,m,reg[A],A,job);
                              //instruction execution
                         clk.tick();
                         ins_set::mstr_clk.tick();
                         if (job->state()==1)
                              //if job status still running
                              //continue executing next
                              //instruction
                         continue;
                         if ( trflag!=0 )
fprintf(trace,"%d\t%d\t%d\n",clk.now(),job->id(),job->j_clk(
));
                         if (job->state() >1 &&
                              job->state() < 4 ) {
                              //if job status is halt
                              //terminate the job and
                              //release its memory
                                   job->term();
                                   trm=job->id();
```

```
                                   m.freem(trm);
                                   rn=0;
                                   return(job->state());
                                   }
                         if (job->state()==4) {
                              //if job status is
                              //blocked for I/O, block the
                              //job
                              m.block(job->id(),EA,reg);
                              return(job->state());
                              }
                         }
               }

/***********************************************************/

CLASS: ins_set

Operations:
 1- constructor ins_set()
 2- decode(word&,int&,int&,int&,int&,word&)
 Translate the instruction to indirection, operation, index
 Reg.,
 arithmetic Reg., and memory address.

 3- execute(int&, memory&,word&,FILE*,FILE*)
 Execute the instruction between the memory and the cpu
 registers
 send output to the output FILE and debugging information to
 the debug FILE.
 Return 0 for HALT or 2 for wrong instruction, else return
 -1.

/***********************************************************/

CLASS: ins_set instructions

 0 HALT                                  C branch and link.
 1 LOAD in reg.                          D Binary And.
 2 STORE from reg.                       E Binary Or.
 3 Add mem. to reg.                      F Read from memory.
 4 Sub mem. from reg.                   10 Write to memory.
 5 Mult. mem and reg.                   11 Memory dump.
 6 Div. mem by reg.                     12 ...
 7 Shift reg. left by mem.              13 ...
 8 Shift reg. right by mem.             user can add new
                                        instructions
 9 branch < 0.                          by overloading the
                                        ins_set class
 A branch > 0.
 B branch = 0.

/***********************************************************/
```

```
#ifndef INS_SET
#define INS_SET

class ins_set {
     byte rgs;
     friend class cpu;
protected:
     byte op;
     s_register old[16]; //keep the status of old
                         //registers value to be used in the
                         //file
     int QN;
     static s_clock mstr_clk;
public:
ins_set();
void set_qntm(int);
void decode(word&,int& ,int& ,int& ,int& ,word& );
virtual void execute(int& , memory& ,word& ,int& ,s_job* );
};
#endif

/***********************************************************/

#include <stream.h>
#include <stdlib.h>
#include <string.h>
#include "hex.h"
#include "byte.h"
#include "word.h"
#include "register.h"
#include "memory.h"
#include "job.h"
#include "ins_set.h"
s_clock ins_set::mstr_clk;

ins_set::ins_set() {}
void ins_set::set_qntm(int i) { QN=i; } //set quantum value
void ins_set::decode(word& d,int& indirect,int& p,int&
REG_A,int& REG_B,word& DADDR) {
               hex_digit x1,x2;
               if ( d[3].m_int_b() >= 128 ) {
                                   //if instruction
                                   //addressing mode is
                                   //indirect
                         indirect=1;
                         x1=d[3].str()[0];
                         x2=d[3].str()[1];
                         x1=x1.int_h()-8;
                         d[3].assign(x1,x2);
                         }
               else indirect =0;
               op=d[3];            //read the operator
               p = op.m_int_b();
```

```
               rgs = d[2];
               REG_A = rgs.int_u(); //the arithmetic
                              //register
               REG_B = rgs.int_l(); //the index register
               DADDR[0] = d[0];     //the memory address
               DADDR[1] = d[1];
               }

void ins_set::execute(int& EA, memory& m, word& REG,int&
A,s_job* job)
{
int flag = 1;
int jid=job->id();
if (job->trace()==1) old[A].put(REG);
char tmp[81];
char tmp1[9];
char tmp2[9];
char tmp3[9];
char tmp4[9];
switch (op.m_int_b()) //execute the instruction operator
          {
          case 0: flag = 3; break; //halt instruction
          case 1: REG = m(jid,EA); //load operator
                  break;
          case 2: m(jid,EA) = REG; //store operator
                  break;
          case 3: REG = REG + m(jid,EA);
                                     //addition operator
                  break;
          case 4:
               REG = REG - m(jid,EA);   //subtraction
                                  //operator
               break;
          case 5: REG = REG * m(jid,EA); //multiply operator
                  break;
          case 6: REG = REG / m(jid,EA); //division operator
                  break;
          case 7: REG=REG.int_w()<<EA;   //shift to left
                  break;
          case 8: REG=REG.int_w()>>EA;   //shift to right
                  break;
          case 9: if ( REG.int_w() < 0 ) {
                              //branch if
                              //arithmetic register
                              //is negative
                         job->pc() = EA;
                         flag=-1;
                         }
                  break;
          case 10: if ( REG.int_w() > 0 ) {
                              //branch if
                              //arithmetic register
                              //is positive
```

```
               job->pc() = EA;
               flag = -1;
               }
         break;
case 11: if ( REG.int_w() == 0 ) {
                            //branch if
                            //arithmetic register
                            //is zero

               job->pc() = EA;
               flag = -1;
               }
         break;
case 12: REG = m[job->pcv()];   //branch and link
        job->pc() = EA;
        flag = -1;
        break;
case 13: REG = REG.int_w() & m(jid,EA).int_w();
                                  //bitwice and
         break;
case 14: REG = REG.int_w() | m(jid,EA).int_w();
                                  //bitwice or
         break;
case 15:
     gets(tmp);                 //read operator
sscanf(tmp,"%8s%8s%8s%8s\n",tmp1,tmp2,tmp3,tmp4);
if (strlen(tmp1)<8)
         {cerr<<"Error:In input card\n";exit(0);}
if (strlen(tmp2)<8)
         {cerr<<"Error:In input card\n";exit(0);}
if (strlen(tmp3)<8)
          {cerr<<"Error:In input card\n";exit(0);}
if (strlen(tmp4)<8)
          {cerr<<"Error:In input card\n";exit(0);}
     m(jid,EA)=tmp1;     //read four words in the
                         //memory from the input
                         //card
     m(jid,EA+1)=tmp2;
     m(jid,EA+2)=tmp3;
     m(jid,EA+3)=tmp4;
     job->j_tick(10);        //input delay time
     flag=4;
     break;
case 16:
     for (int i=0; i< 4; i++) //write four words
                         //to the output file
          {
          m(jid,EA+i).print(job->outf());
          }
     fprintf(job->outf(),"\n");
     job->j_tick(10);         //output delay time
     flag=4;
     break;
```

```
          case 17: m.dump(job->outf()); break;
          default: flag = 2;
                     cerr<<"Error:Unexpected command\n";
          }
          if ( flag == 1 || flag == 4 )
//increment job process counter if job succeed to execute
//the instruction
           job->pc()=job->pc()+1;
          else if ( flag == -1 ) flag = 1;

          job->j_tick();
          if (flag != 4 && job->trace() == 1 && job->j_clk()
>= 900) {
//detect infinite loop if the execution time of  the job is
//more than 900 clock cycles
               flag=2;job->t_time();
               cerr<<"Error:Suspected infinite loop job
time>900cc\n";
               cerr<<"Turn trace flag off and try again\n";
               }
          if ( flag != 4 && job->j_past() >= QN )
{flag=4;job->t_time();}  //if the quantum is passed,
                         //suspend the job
          job->state() = flag;

          if ( job->trace() == 1 ) {
                    m(jid,EA).print(job->dbgf());
                    REG.print(job->dbgf());
                    old[A].print(job->dbgf());
                    fprintf(job->dbgf(),"\n");
                    }
}

/***********************************************************/
```

CLASS: my_window

WINDOW* w,int no_var,int *mx,int *my.

Operations:

1- constructor my_window(int,int,int,int,int)
create window with width, length, location, and no variables.
2- void add_box(char,char)   add box and refresh the window
3- void add(int,int,char) & (int,int,char*)   add char or string to the window
4- void add_rf(..)   add and refresh the window
5- void add_B(..)   add and highlight
6- void del(int,int)   delete line
7- void set_var(int*,int*)   set variable locations

```
 8- void update(int,char*)     update the variable

/**********************************************************/

#include <stream.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>
#include <curses.h>
#include "my_util.h"

class my_window {
     WINDOW *w;                //the curses library window
     WINDOW *sw;               //subwindow
     int no_var;               //number of variables in the
                        //window
     int *my;
     int *mx;
     friend class debug_window;
public:
     my_window() { }
     my_window(int yl,int xl,int y,int x,int n)
               {
               w = newwin(yl,xl,y,x);
               no_var = n;
               }

     WINDOW *winwin() { return(w); }
     void add_box(char v,char h) //add a frame to the window
                     { box(w,v,h); wrefresh(w); }
     void my_clear()             //erase window contents
                     { werase(w); wrefresh(w); }
     void my_move(int y, int x)  //move curser in the window
                     { wmove(w,y,x); wrefresh(w);}
     void del_ln(int y, int x)   //delete a line
                     { wmove(w,y,x); wdeleteln(w); }
     void del_ln_rf(int y, int x)
                 { wmove(w,y,x); wdeleteln(w); wrefresh(w);}
     void my_delwin() { delwin(w); }
     void add(int y, int x, char c) {
//add character in location (x,y) in the window
//wmove(w,y,x);
                         waddch(w,c);
                    }
     void read(char& c) { scanf("%c",&c); } //read a char
                                           //from the window
     void read(char* c) { scanf("%s",c); }  //read string
                                           //from the window
     void add_rf(int y, int x, char c) {
                    //add char to the window and update it
                         wmove(w,y,x);
                         waddch(w,c);
                         wrefresh(w);
```

```
                   }
    void add(int y, int x, char *s) {
                   //add string to the window
                        wmove(w,y,x);
                        waddstr(w,s);
                   }
    void add_B(int y, int x, char *s) {
                     //add string to the window
                        wmove(w,y,x);
                        wstandout(w);
                        waddstr(w,s);
                        wstandend(w);
                   }
    void add_rf(int y, int x, char *s) {
                      //add string to the window and update
                      //it
                        wmove(w,y,x);
                        waddstr(w,s);
                        wrefresh(w);
                   }
    void add_rf_B(int y, int x, char *s) {
                      //add string to the window and update
                      //it
                        wmove(w,y,x);
                        wstandout(w);
                        waddstr(w,s);
                        wrefresh(w);
                        wstandend(w);
                   }
    void set_var(int *y,int *x) {
                   //set the x,y location of the window
                   //variables
                         my = new int[no_var];
                         for(int i=0;i<no_var;i++)
                          my[i]=y[i];
                         mx = new int[no_var];
                         for( i=0;i<no_var;i++) mx[i]=x[i];
                        }
    void update(int i, char *s) {
                   //move courser and update the window
                   wmove(w,my[i],mx[i]);
                   waddstr(w,s);
                   wrefresh(w);
                   }
    void my_refresh() { wrefresh(w); }
    void touch() { touchwin(w); } //update the window
    int my_read(char*);
    int my_read(char&);
    int my_read(int&);    //overloading to read string,
                    //char,and int
    ~my_window() { delete(my); delete(mx); }
};
```

```
int my_window::my_read(char* i) {
                        //read string from the window
          char temp[99];
          wgetstr(w,temp);
          strcpy(i,temp);
          return(1);
     }
int my_window::my_read(char& i) {
                            //read a char from the window
          char temp[99];
          wgetstr(w,temp);
          i=temp[0];
          return(1);
     }
int my_window::my_read(int& i) {//overloading to read
                    //string, char, and int
          char temp[99];
          wgetstr(w,temp);
          i=atoi(temp);
          return(1);
     }

/**********************************************************/

This program working as an interface to call the debugger
and help subprograms

/**********************************************************/

#include <stream.h>
#include <stdlib.h>
#include <string.h>
#include "wind2.h"


#define W_TRUE 1

main()
{
     initscr();
     my_window w(10,24,14,25,0);
     w.add_box('|','-');
     w.add(4,5,"   WELCOME   ");
     w.add_rf(6,5,"  IN THE Semulation Prototype  ");
     char a;
     noecho();
     w.my_read(a);
     echo();

     while( W_TRUE ) {
     w.my_clear();
     w.add_box('|','-'); //display the user options menu
     w.add_B(1,1,"         PRESS        ");
```

```
     w.add(2,1,"----------------------");
     w.add_B(4,2,"[D]");
     w.add(4,5," for the Debugger");
     w.add_B(5,2,"[H]");
     w.add(5,5," for the Help    ");
     w.add_B(6,2,"[Q]");
     w.add(6,5," to Quit         ");
     w.add_rf(7,2,">> ");

     if (w.my_read(a) && !isalpha(a) && a!='h' && a!='d' &&
a!='q') {
          continue;
          }
     if ( a == 'h' )
          {
          system("/y/hassan/os2/os2help");
                  //call the help subprogram "help"
          continue;
          }
     else
     if ( a == 'q' ) {
          w.my_delwin();
          break;
          }

     else {
          system("/y/hassan/os2/debugger");
                   //call the subprogram "debugger"
          continue;
          }
     }
     endwin();
     exit(0);
}

/**********************************************************/

#include "wind2.h"
#include <pwd.h>
#define MY_TRUE 1
#define MAXJOB 999
#define MAXRUN 50

class debug_window
{
     my_window *w1,*w2,*w3,*w4;      //the four windows of
                           //the debugger
     FILE* dbg,*dbg2;
     char dbga[MAXJOB][82];          //debugger displayed
                           //variables
     char cpu[4],memr[5],jclk[5],cclk[7];
     char reg_val[10];
     char temp_val[10];
```

```
     int no_of_ins,chinst,jck,cck;
     int no_ins;                    //instruction number
     int reg_no;                    //register number
     int chg;
     char job_no[5];                //job id
     static char base[17];          //hex_digit values

public:
     debug_window() {
     reg_no=0;
     chg=1;
     jck=cck=0;
     strcpy(job_no,"0000");
     strcpy(base,"0123456789ABCDEF");
     my_window ww1(22,15,2,64,16);  //display job,
                           //instruction, registers,and
                           //menu windows
     my_window ww2(15,35,2,28,7);
     my_window ww4(15,25,2,2,6);
     my_window ww3(7,61,17,2,0);
     w1 = &ww1;w2 = &ww2;w3 = &ww3;w4 = &ww4;
     no_of_ins=0;
     this->act();                   //put variables in all
                            //windows
     this->refresh_all();           //update all windows
     this->work();                  //start the debugger
     }

     void act1();   //activate job window
     void act2();   //activate instruction window
     void act3();   //activate registers window
     void act4();   //activate options window
     void act_prnt(); //activate print window
     void act()       //activate all windows
      { act4(); act2(); act1(); act3(); }
     void work();     //execute windows to display data
void update(char* s,char* mem,char* regs,int cp,int ck,int
jk,int flag);          //update all windows
void q_update(char* s,char* regs,int cp,int ck,int jk,int
flag);                 //quick update for all windows
     void print_reg(int s, int flag);
     int main_ask();  //read user choice from menu options
     int prnt_ask();  //read user choice from print menu
     int get_job(int); //read job id to debug it
     void refresh_reg() { w1->my_refresh(); }
                       //update registers window
     void refresh_ins() { w2->my_refresh(); }
                       //update instructions window
     void refresh_usr() { w4->my_refresh(); }
                       //update options window
     void refresh_all() { w1->my_refresh();
w2->my_refresh(); w4->my_refresh(); }
                    //update all windows
```

```
     void clear_all() { w1->my_clear(); w2->my_clear();
w4->my_clear(); w3->my_clear();}
                       //erase information from all windows
     void my_del_all() { w1->my_delwin();
                       w2->my_delwin();
                       w4->my_delwin();
                    w3->my_delwin(); } //delete all windows
};

void debug_window::print_reg(int s,int flag=0)
{
     char temp[11],val[11];
     char reg_T[3];
     char sys_reg[16][10];
     FILE *prn;

     prn = fopen("ose_prn","w");
     if ( flag == 0 ) fprintf(prn,"INST#     REG
%2d\n\n",s);
     else {                      //print title of trace file
     fprintf(prn,"INST# REG   0  REG   1  REG   2  REG  3");
     fprintf(prn,"   REG   4  REG   5  REG   6  REG   7\n");
     fprintf(prn,"      REG   8  REG   9  REG  10  REG 11");
     fprintf(prn,"   REG  12  REG  13  REG  14  REG
15\n\n");
          for (int j=0;j < 16; j++)
               strcpy(sys_reg[j],"00000000 ");
          }
     for (int i=1;i <= no_ins; i++) {
          sscanf(dbga[i],"%10s %10s %10s",val,temp,temp);
          temp[8]=' ';temp[9]='\0';
          reg_T[0]=val[2]; reg_T[1]='\0';
          if ( flag == 0 ) {
               if (s == atoi(reg_T) )
                 fprintf(prn,"%3d\t%9s\n",i,temp);
               }
          else {   //print registers contents
               strcpy(sys_reg[atoi(reg_T)],temp);
               fprintf(prn,"%3d  %9s",i,sys_reg[0]);
               for (int j=1;j < 8; j++)
                    fprintf(prn,"%9s",sys_reg[j]);
               fprintf(prn,"\n     ");
               for (j=8;j < 16; j++)
                    fprintf(prn,"%9s",sys_arithmetic[j]);
               fprintf(prn,"\n\n");
               }
          }
     fclose(prn);
}

void debug_window::act1() {
     w1->add_box('|','-');      //draw the window frame
     w1->add_B(1,1,"  REGISTERS  ");
```

```
     char s1[10];
     for ( int i=0; i < 10; i++) //display registers numbers
          {
          char tt[5];
          strcpy(s1," ");
          itoa(i,tt);
          strcat(s1,tt);
          strcat(s1,":");

          w1->add_B(4+i,2,s1);
          }
     for ( i=10; i < 16; i++)
          {
          char tt[5];
          itoa(i,tt);
          strcpy(s1,tt);
          strcat(s1,":");
          w1->add_B(4+i,2,s1);
          }
     w1->add_rf(2,1,"-------------");
     int regy[16],regx[16];

                              //specify register value
                     //location in the window
     for (i =0; i <16; i++) { regy[i] = 4+i; regx[i] = 5; }
     w1->set_var(regy,regx);
     for(i=0;i<16;i++)    //update the register window
                       w1->update(i,"00000000");
     }

void debug_window::act2() {
     int regy[16],regx[16];
     w2->add_box('|','-'); //add window frame
     w2->add_B(1,1,"           INSTRUCTION           ");
     w2->add(2,1,"---------------------------------");

                             //add window content
     w2->add_B(3,3," Instruction Code :");
     w2->add_B(4,3," Indirection      :");
     w2->add_B(5,3," Index Register   :");
     w2->arithmetic_B(ARITHMETIC,3," Arithmatic Reg.  :");
     w2->add_B(7,3," Memory Location  :");
     w2->add_rf_B(8,3," Memory Loc. Cont.:");
                            //specify window content
                         //location
     regy[0] = 3; regy[1] = 4;
      regy[2] = 5; regy[3] = 10; regy[4] = 6;
     regy[5] = 7; regy[6] = 8;
     regx[0] = 23; regx[1] = 23; regx[2] = 23;
      regx[3] = 4;regx[4]=23;
     regx[5] = 23; regx[6] = 23;
     w2->set_var(regy,regx);
     }
```

```
void debug_window::act4() {
     int regy[16],regx[16];
     w4->add_box('|','-'); //draw window frame
     w4->add_B(1,1,"  General Information  ");
     w4->add(2,1,"-----------------------");
                                  //add window contents
     w4->add_B(3,3," JOB ID :");
     w4->add_B(5,3," MEMORY :");
     w4->add_B(7,3," CPU    :");
     w4->add_B(9,3," INST.#:");
     w4->add_B(11,3," JB_CLK:");
     w4->add_rf_B(13,3,"CPU_CLK:");
                                  //add window content locations
     regy[0] = 3; regx[0] = 13;
     regy[1] = 5; regx[1] = 13;
     regy[2] = 7; regx[2] = 13;
     regy[3] = 9 ; regx[3] = 13;
     regy[4] = 11; regx[4] = 13;
     regy[5] = 13; regx[5] = 13;
     w4->set_var(regy,regx);
     }

void debug_window::act3() {
     w3->add_box('|','-');  //draw window frame
     w3->add_B(1,2," nJ "); //add window contents
     w3->add(1,7,"New Job-n");
     w3->add_B(1,21," R  ");
     w3->add(1,26,"Print");
     w3->add_B(1,43," Q ");
     w3->add(1,47,"Quit");
     w3->add_B(2,2," N  ");
     w3->add(2,7,"Next Inst.");
     w3->add_B(2,21," nN ");
     w3->add(2,26,"n Next Inst.");
     w3->add_B(3,2," P  ");
     w3->add(3,7,"Prev Inst.");
     w3->add_B(3,21," nP ");
     w3->add(3,26,"n Prev Inst.");
w3->add(4,1,"------------------------------------------------
------------");
     w3->add_rf(5,2,">> ");
     }

void debug_window::act_prnt() {
     w3->my_clear();      //erase main menu contents
     w3->add_box('|','-');  //redraw print window frame
     w3->add_B(1,2," nR "); //add window frame
     w3->add(1,6,"nReg. History");
     w3->add_B(1,25," RH ");
     w3->add(1,30,"Registers History");
     w3->add_B(2,2," Q ");
     w3->add(2,6,"Quit");
```

```
w3->add(4,1,"-----------------------------------------------
------------");
     w3->add(5,2,">> ");
     w3->my_refresh();
     }

void debug_window::update(char* s,char* mem,char* regs,int
cp,int ck,int jk,int flag=0)
{
     cp=ck=jk=0;
     char indirect[2];      //indirect register
     int oper=0;
     indirect[0] = s[0]; indirect[1] = '\0';
     char ss[5];
     for (int k=0;k<16;k++) if(s[0] == base[k])
                        {oper=16*k;break;}
     for (k=0;k<16;k++) if(s[1] == base[k]) {oper+=k;break;}
     char reg_A[2];
     reg_A[0] = s[2]; reg_A[1] = '\0';
                         //arithmetic and
                         //index registers
     char reg_B[2];
     reg_B[0] = s[3]; reg_B[1] = '\0';
     ss[0] = s[4]; ss[1] = s[5]; ss[2] = s[6];
      ss[3] = s[7];ss[4]='\0';

     w2->update(0,s);        //update instruction
     w2->update(1,indirect);//update indirect mode
     w2->update(2,reg_B);   //update index register
     char ins[17];
     char ins_no[4];
     switch (oper) {     //display instruction meaning
          case 0: strcpy(ins,"Halt            "); break;
          case 1: strcpy(ins,"Load            "); break;
          case 2: strcpy(ins,"Store           "); break;
          case 3: strcpy(ins,"Addition        "); break;
          case 4: strcpy(ins,"Subtraction     "); break;
          case 5: strcpy(ins,"Multiplication  "); break;
          case 6: strcpy(ins,"Division        "); break;
          case 7: strcpy(ins,"Shift to left   "); break;
          case 8: strcpy(ins,"Shift to right  "); break;
          case 9: strcpy(ins,"Branch on<0     "); break;
          case 10: strcpy(ins,"Branch on>0     "); break;
          case 11: strcpy(ins,"Branch on=0     "); break;
          case 12: strcpy(ins,"Goto            "); break;
          case 13: strcpy(ins,"Bin AND         "); break;
          case 14: strcpy(ins,"Bin OR          "); break;
          case 15: strcpy(ins,"Read            "); break;
          case 16: strcpy(ins,"Write           "); break;
          case 17: strcpy(ins,"Memory dump     "); break;
          case 129: strcpy(ins,"Load            "); break;
          case 130: strcpy(ins,"Store           "); break;
          case 131: strcpy(ins,"Addition        "); break;
```

```
     case 132: strcpy(ins,"Subtraction     "); break;
     case 133: strcpy(ins,"Multiplication  "); break;
     case 134: strcpy(ins,"Division        "); break;
     case 135: strcpy(ins,"Shift to left   "); break;
     case 136: strcpy(ins,"Shift to right  "); break;
     case 137: strcpy(ins,"Branch on<0     "); break;
     case 138: strcpy(ins,"Branch on>0     "); break;
     case 139: strcpy(ins,"Branch on=0     "); break;
     case 140: strcpy(ins,"Goto            "); break;
     case 141: strcpy(ins,"Bin AND         "); break;
     case 142: strcpy(ins,"Bin OR          "); break;
     case 143: strcpy(ins,"Read            "); break;
     case 144: strcpy(ins,"Write           "); break;
     case 145: strcpy(ins,"Memory dump     "); break;
     }

w2->update(3,ins);  //update instruction meaning
w2->update(4,reg_A);   //update arithmetic register
w1->update(atoi(reg_A),regs); //update register window
if ( flag != 0 ) { w1->update(reg_no,reg_val);
             strcpy(reg_val,temp_val);}
reg_no = atoi(reg_A);
w2->update(5,ss);
w2->update(6,mem);  //update memory id
w4->update(0,job_no);//update job id
w4->update(1,memr); //update memory location
w4->update(2,cpu);  //update cpu id
if ( flag == 0 ) {  //update instruction number
     ++no_of_ins;     //and clock value
     ++cck; ++jck;
     if (oper==15 || oper==16) {jck+=10; }
     }
else {
     --no_of_ins;
     --cck; --jck;
     }

if ( no_of_ins<=0 || no_of_ins>=no_ins )
               //stop the debugger at the end of
               //the job
                              {endwin();exit(0);}
char tt[9];
itoa(no_of_ins,tt);
strcpy(ins_no,tt);
for (int j=0;j< (3-strlen(tt)); j++)
     strcat(ins_no," ");
w4->update(3,ins_no); //update instruction value
char tc[5];
itoa(jck,tc);
while (strlen(tc)!=4) strcat(tc," ");
w4->update(4,tc);     //update cpu clock
itoa(cck,tc);
while (strlen(tc)!=4) strcat(tc," ");
```

```
     w4->update(5,tc);     //update job clock
     w3->add_rf(5,4,"    ");
     w3->my_move(5,5);
}

void debug_window::q_update(char* s,char* regs,int cp,int
ck,int jk,int flag=0)
{
     cp=ck=jk=0;
     char reg_A[2];
     int oper=0;
     for (int k=0;k<16;k++) if(s[0] == base[k])
                            {oper=16*k;break;}
     reg_A[0] = s[2]; reg_A[1] = ’\0’;
     w1->update(atoi(reg_A),regs); //update register window
     if ( flag != 0 ) { w1->update(reg_no,reg_val);
                                 //step back update
                 strcpy(reg_val,temp_val);
                 --no_of_ins;
                 --cck; --jck;
              if (oper==15 || oper==16) {jck-=10; }
              }
     else {                    //step foreword update
          ++no_of_ins;
          ++cck; ++jck;
          if (oper==15 || oper==16) {jck+=10; }
          }

     if ( no_of_ins<=0 || no_of_ins>=no_ins )
                             {endwin();exit(0);}
     char tt[10];
     itoa(jck,tt);
     w4->update(4,tt);   //update job clock
      itoa(cck,tt);
     w4->update(5,tt);   //update cpu clock
     reg_no = atoi(reg_A);
}

int debug_window::main_ask()
{
     char a[4];
     int ln;

     while (MY_TRUE){
     w3->add_rf(5,4,"    ");
     w3->my_move(5,5);    //move the curser to the input
     w3->my_read(a);      //location
     ln = strlen(a);
     if ( ln==1 )
          switch (tolower(a[0])) {
               case ’n’: return(-2);
                           //step one instruction foreword
               case ’p’: return(-3);
```

```
                           //step back one instruction
               case 'r': return(prnt_ask());
                           //display the print menu
               case 'q': return(-1);
                           //quit the debugger
               default : w1->my_move(5,5);continue;
               }

     else if ( ln > 1 )
          switch (tolower(a[ln-1])) {
               case 'n': a[ln-1]='\0';return(atoi(a));
                              //step execution forward
               case 'p':
                        a[ln-1]='\0';return(atoi(a)+MAXJOB);
                              //step execution backword
               case 'j': a[ln-1]='\0';
                                  //choose a job to debug
                    if (get_job(atoi(a))==0) continue;
                     for(int i=0;i<16;i++)
                         w1->update(i,"00000000");
                      return(-2);
               default : w1->my_move(5,5);continue;
               }
         else {w1->my_move(5,5);continue;}
     }
}

int debug_window::prnt_ask()
{
     char a[4];
     int ln;

     act_prnt();      //start the print options window

     while (MY_TRUE){
     w3->add_rf(5,4,"        ");
     w3->my_move(5,5);
     w3->read(a);   //read user choice
     ln = strlen(a);
     if ( ln==1 )
          switch (tolower(a[0])) {
               case 'q':         //quit print options menu
                          w3->my_clear();
                          act3();  //redisplay main menu
                          w1->my_move(5,5);
                          return(0);
               default : w1->my_move(5,5);continue;
               }

     else if ( ln > 1 )
          switch (tolower(a[ln-1])) {
               case 'r': a[ln-1]='\0';
                           print_reg(atoi(a));
```

```
                                   //print register history
                        w1->my_move(5,5);continue;
                 case 'h': print_reg(0,1);
                                 //print all registers history
                        w1->my_move(5,5);continue;
                                 //move to the input location
                 default : w1->my_move(5,5);continue;
                 }
            else {w1->my_move(5,5);continue;}
       }
}

int debug_window::get_job(int j)
{
     char tt[5];
     itoa(j,tt);
     strcpy(job_no,tt);
     char jobfile[50];
     char job2file[50];

     FILE* log;
     char fullname[81],*name,fnl[50];
                          //open job profile files
     strcpy(job2file,jobfile);
     strcpy(fnl,"date >> ");
     strcat(fnl,jobfile);
     strcat(fnl,"/db");
     system(fnl);
     strcat(jobfile,"/JOB_DB_");
     strcat(job2file,"/JOB_DB2_");
     strcat(jobfile,job_no);
     strcat(job2file,job_no);
     if ( (dbg=fopen(jobfile,"r")) ==NULL) {return(0);}
                          //open debug information file
     fgets(cpu,3,dbg);
     fgets(memr,2,dbg);
     fgets(memr,5,dbg);
     memr[3]='\0';
     no_of_ins=jck=cck=0;

     int i=0;
     while ( !feof(dbg) ) fgets(dbga[++i],80,dbg);
                          //read debug file in an array
     no_ins=i;
     fclose(dbg);
     return(1);
}

void debug_window::work()
{
     char inst[10],memory[10];
     char regs[22];
     int flag=-2,jn=0,cp=0,jk=0,ck=0;
```

```
     if ( (flag=main_ask()) == -1 );
     else {
     while ( no_of_ins <= no_ins )
            //debug the job instruction by instruction
     {
     if ( flag > MAXJOB ) { //quick update of debugger's
                            //windows if user steps more than
                            //one instructions backward
      for (int j=1; j<(flag-MAXJOB); j++)
       {
       sscanf(dbga[no_of_ins-1],"%9s %9s %9s
        %9s",inst,memory,regs,temp_val);
       q_update(inst,regs,cp,ck,jk,1);
       }
      flag = -3;
      }
     if ( flag == -3 ) { //use complete update for one
                         //instruction
       sscanf(dbga[no_of_ins-1],"%9s %9s %9s
       %9s",inst,memory,regs,temp_val);
       update(inst,memory,regs,cp,ck,jk,1);
       }
     if ( flag > 0 ) {   //use quick update for more than
                         //one instruction forward
          for (int j=1; j<flag; j++)
                {
                 sscanf(dbga[no_of_ins+1],"%9s %9s
                %9s",inst,memory,regs);
                 q_update(inst,regs,cp,ck,jk);
                }
          flag = -2;
          }
     if ( flag == -2 ) { //use complete update for one
                         //instruction forward
        sscanf(dbga[no_of_ins+1],"%9s %9s %9s
         %9s",inst,memory,regs,reg_val);
        update(inst,memory,regs,cp,ck,jk);
        }
     if ( (flag=main_ask()) == -1 ) {break;}
                           //stop the debugger
     }
     }
     this->clear_all();    //erase all windows contents
     this->my_del_all();   //delete all debugger windows
}

main() {
initscr();                    //initialize windows session
debug_window wd;              //execute the debuger
endwin();                     //end windows session
}

/************************************************************/
```

```
// This hlp.c program is an application of my_window class.
// It is called by the main program win.c to read the needed
// information from the "man" file about the class that the
// client needs.
/**********************************************************/

#include <stream.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <curses.h>

#define TRUE 1

main()
{
     initscr();
     FILE* ff;
     char addrr[30][1600];
     char buffer[81];
     int i=0;
     char fn[81],*name,final[50];

      system("pwd >> ._temp"); //creat a file for statistics
                                //information
     ff=fopen("._temp","r");
     fgets(fn,80,ff);
     fclose(ff);
     name=strtok(fn,"\/\n");
     name=strtok(NULL,"\/\n");
     strcpy(final,"date >> /x/jhun/ooos/");
     strcat(final,name);
     strcat(final,"/hlp");
     system(final);

     FILE* mn = fopen("/y/hassan/os2/man","r");
                                    //open the manual file
     while ( fgets(buffer,80,mn) != NULL )
                                     //display help pages
     {
     if ( buffer[0] != 'E' ) strcat(addrr[i],buffer);
     else{
      if (i!=0)
     strcat(addrr[i],"\n\n<press any key to go back to the
      menu>\n");
          i++;
           strcpy(addrr[i],buffer);
          }
     }
     strcat(addrr[i],"\n\n<press any key to go back to the
    menu>\n");
     fclose(mn);
     my_window w1(22,76,1,2,0); //draw the user interface
```

```
                         //windows
     my_window w2(22,79,1,2,0);

while( TRUE )
     {                         //display the online help
                         //menu options
     w1.add_box('|','-');
     w1.add_B(1,31," CHOOSE FROM ");
w1.add(2,1,"-----------------------------------------------
--------------------------");
w1.add(19,1,"----------------------------------------------
---------------------------");
                                //display options in the
                        //help menu
     w1.add( 3,5, " 1- ELEM. s_clock");
     w1.add( 4,5, " 2- ELEM. vect");
     w1.add( 5,5, " 3- ELEM. hex_digit");
     w1.add( 6,5, " 4- ELEM. byte");
     w1.add( 7,5, " 5- ELEM. word");
     w1.add( 8,5, " 6- ELEM. s_register");
     w1.add( 9,5, " 7- ELEM. storage");
     w1.add(10,5, " 8- ELEM. memory");
     w1.add(11,5, " 9- ELEM. loader");
     w1.add(12,5, "10- ELEM. ins_set");
     w1.add(13,5, "11- ELEM. cpu");
     w1.add(14,5, "12- ELEM. pcb_element");
     w1.add(15,5, "13- ELEM. pt_element");
     w1.add(16,5,"14- ELEM. pt_table");
     w1.add(17,5,"15- ELEM. mem_table_elem");
     w1.add( 3,44,"16- ELEM. memory_table");
     w1.add( 4,44,"17- ELEM. my_window");
     w1.add( 5,44,"18- ELEM. debug_window");
     w1.add( 6,44,"19- OVERLOAD ins_set");
     w1.add( 7,44,"20- INS_SET instruction");
     w1.add( 8,44,"21- JOB EXAMPLE (ASM)");
     w1.add( 9,44,"22- JOB DATA (ASM)");
     w1.add(10,44,"23- JOB EXAMPLE (HEX)");
     w1.add(11,44,"24- New Ins_set for Phase II");
     w1.add(12,44,"25- New cpu for Phase II");
     w1.add(13,44,"26- New loader for Phase II");
     w1.add(14,44,"27- Random numbers Generator");
     w1.add(16,44,"28- Print Help in ms222");
     w1.add(17,44,"29- Print Help in ms214");
     w1.add(20,44,"Hit return key to quit");

     int a=0,j=2;
     w1.add_rf(20,4,">> ");
     if ( w1.my_read(a) < 0 || a < 0 || a > 29 ) continue;
     if ( a == 0 ) break;
     if ( a == 28 )
      {system("lp -s -dms222 /y/hassan/os2/man");continue;}
     if ( a == 29 )
      {system("lp -s -dms214 /y/hassan/os2/man");continue;}
```

```
     w1.my_clear();              //delete options menu
     w2.add_rf(j,1,addrr[a]); //display menu pages for the
                        //user choice
     getchar();
     w2.my_clear();              //delete menu pages
     }
     w2.my_delwin();             //delete the help window
     w1.my_clear();              //clear the frame window
     w1.my_delwin();             //delete the frame window
     endwin();
}

/**********************************************************/

#include <stream.h>
#include <stdio.h>
#include <string.h>
#include "my_util.h"
extern "C" void exit(int);
#include "memory.h"

class m_data { //a class to store parsing information about
               //the job instructions
     char name[50][20];
     word* val;
     int line_no[50];
     int i;
public:
     m_data() { i=0; val = new word[50]; }
     ~m_data() { delete(val); }

     void put(char* nm,char* v,int ln)//read an instruction
          {
          strcpy(name[i],nm);
          val[i]=v;
          line_no[i]=ln;
          i++;
          }

     word& values(char* nm) { //convert the instruction from
                              //string to a word
          for (int j=0; j<i; j++)
               if (strcmp(name[j],nm)==0) break;
          return(val[j]);
          }

     int value(char* nm) {     //return the variable address
     for (int j=0; j<i; j++)
               if (strcmp(name[j],nm)==0) break;
          return(val[j].int_w());
          }

     int addrs(char* nm) { //return the instruction address
```

```
          for (int j=0; j<i; j++)
               if (strcmp(name[j],nm)==0) break;
          return(line_no[j]);
          }
};

main(int argc,char* argv[])
{
FILE *in,*out;
if ( argc != 3 ) exit(-1);
in = fopen(argv[1],"r");    //open the assembly job file
out = fopen(argv[2],"w");   //open the hex_digit job file

int i=0,max=0;
char prog[100][82];
while(!feof(in)) {       //read the program and its data in
                         //the array "prog"
     fgets(prog[i],81,in);
     if (strlen(prog[i]) > 2 && prog[i][0]!=';') i++;
     }
max=i-1;
char cons[10],DT[5];
char val[9];

m_data Ddata;                                DT
i--;
sscanf(prog[i],"%s%s%8s",cons,DT,val);
while (strcmp(DT,"DATA") == 0 ) { //separate data line from
                         //the program
     Ddata.put(cons,val,i);
     sscanf(prog[--i],"%s%s%8s",cons,DT,val);
     }

m_data Pdata;
char opr[10],opn[10],inst[9],c[5];
static char base[16] =
{'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E'
,'F'};
int temp1,temp2,temp3,temp4;
int k,addr,words=0;
i++;

for (int j=0;j<i;j++) {  //read job instructions
     if ( prog[j][6] != ' ') {
          sscanf(prog[j],"%s",cons);
          Pdata.put(cons,"00000000",j);
          for ( k=0;k<strlen(cons);k++) prog[j][6+k]=' ';
          }
     }

for (j=0;j<=i;j++) {     //read jobs data
     int no_of_reg=0;
     if ( words==4) { fprintf(out,"\n");words=0; }
```

```
          for ( k=0; k<strlen(prog[j]);k++)
               if ( prog[j][k]==',' ) {
                    prog[j][k]=' ';
                    no_of_reg++;
                    }
          int r1=0,r2=0;
          strcpy(c,"0000");
          if ( no_of_reg==1) { //if there is  no indexing
                             //address read the arithmetic
                             //register only
               sscanf(prog[j],"%s%x%s",opr,&r1,opn);
               if (r1>7) {
cerr << "can't use system registers 8-F at: "<< j<<"You may
have an error in your comments\n";
                    exit(0);}
               c[1]='0';
               }

          else if ( no_of_reg == 2) {//if the instruction
                              //contains the index register
                              //read both registers in the
                              //instruction
               sscanf(prog[j],"%s%x%x%s",opr,&r1,&r2,opn);
               if ( r1>7 || r2>7 ) {
cerr << "cann't use system registers 8-F at: "<< j<<"You may
have an error in your comments.\n";
                    exit(0);}
               c[1]=base[r2%16];
               }
          else sscanf(prog[j],"%s%s",opr,opn);
          if ( opn[0] == '(' && opn[strlen(opn)-1] == ')') {
               for ( k=0; k < strlen(opn)-2; k++)
                    opn[k]=opn[k+1];
               opn[strlen(opn)-2]='\0';
               strcpy(inst,"8");
          }
          else strcpy(inst,"0");

          if (strcmp(opr,"RD")==0) {//if the instruction is
                         //read instruction there is no
                         //index register
               strcat(inst,"F00");
               addr=Ddata.addrs(opn);
               temp1=addr%16;
                         //covert data address to hex_digit
               temp2=addr/16;
               temp3=temp2/16;
               temp4=temp3/16;
               c[3]=base[temp1];
               c[2]=base[temp2%16];
               c[1]=base[temp3%16];
               c[0]=base[temp4%16];
               strcat(inst,c);
```

```
     fprintf(out,"%s",inst);
     words++;
     }
else if (strcmp(opr,"WR")==0){//if the instruction
               //is write instructionno index
               //register in it
               //if the instruction use indirect
               //address set indirect bit to 1
     if ( inst[0]=='8' ) strcpy(inst,"9000");
     else strcpy(inst,"1000");
               //if direct address instruction set
               //bit to 0
     addr=Ddata.addrs(opn);
     temp1=addr%16; //covert data address to
                    //hex_digit
     temp2=addr/16;
     temp3=temp2/16;
     temp4=temp3/16;
     c[3]=base[temp1];
     c[2]=base[temp2%16];
     c[1]=base[temp3%16];
     c[0]=base[temp4%16];
     strcat(inst,c);
     fprintf(out,"%s",inst);
     words++;
     }
else {
addr=Ddata.addrs(opn);
          //find the instruction address
             //translate the instruction from
          //assembly to hex_digit value
if ( strcmp(opr,"LD") == 0) strcat(inst,"1");
if ( strcmp(opr,"ST") == 0) strcat(inst,"2");
if ( strcmp(opr,"AD") == 0) strcat(inst,"3");
if ( strcmp(opr,"SB") == 0) strcat(inst,"4");
if ( strcmp(opr,"MPY") == 0) strcat(inst,"5");
if ( strcmp(opr,"DIV") == 0) strcat(inst,"6");
if ( strcmp(opr,"SHL") == 0) strcat(inst,"7");
if ( strcmp(opr,"SHR") == 0) strcat(inst,"8");
if ( strcmp(opr,"BRM") == 0) {strcat(inst,"9");
                    addr=Pdata.addrs(opn);}
if ( strcmp(opr,"BRP") == 0) {strcat(inst,"A");
                    addr=Pdata.addrs(opn);}
if ( strcmp(opr,"BRZ") == 0) {strcat(inst,"B");
                    addr=Pdata.addrs(opn);}
if ( strcmp(opr,"BRL") == 0) {strcat(inst,"C");
                    addr=Pdata.addrs(opn);}
if ( strcmp(opr,"AND") == 0) strcat(inst,"D");
if ( strcmp(opr,"OR") == 0) strcat(inst,"E");
if ( strcmp(opr,"DMP") == 0) strcpy(inst,"11");
if ( strcmp(opr,"HLT") == 0)
 {strcpy(inst,"00000000");
                    fprintf(out,"%s",inst);
```

```
                              words++;
                              continue;
                              }
          c[0] = base[r1%16]; //calculate the arithmatic
                              //register
          c[2] ='\0';
          strcat(inst,c);
          strcat(inst,"00");
          temp1=addr%16;       //calculate the memory
                              //address
          temp2=addr/16;
          c[1]=base[temp1];
          c[0]=base[temp2%16];
          strcat(inst,c);
          fprintf(out,"%s",inst);
          words++;
          }

     }

     for ( k=i+1;k<=max;k++)//print job data
          {
          if ( words==4) { fprintf(out,"\n");words=0; }
          sscanf(prog[k],"%s",cons);
          strcpy(inst,"000000");
          (Ddata.values(cons)).prints(out);
                         //calculate the data address and
                         //print it as hex_digit
          words++;
          }
fclose(in);
fclose(out);
}
```

APPENDIX D

RANDOM NUMBER GENERATOR CLASS LISTING

```
/************************************************************
In order to make the simulation package more realistic, a
pseudo-random-number generator class has been included. The
code for this class is listed in this appendix.  This class
has "inter-arrival times" and "service times" methods..
************************************************************/

#include <iostream.h>
#include <stdlib.h>
#include <sys/types.h>
#include <math.h>

const int   M1  = 259200;
const int   IA1 = 7141;
const int   IC1 = 54773;
const float RM1 = (1.0/M1);
const int   M2  = 134456;
const int   IA2 = 8121;
const int   IC2 = 28411;
const float RM2 = (1.0/M2);
const int   M3  = 243000;
const int   IA3 = 4561;
const int   IC3 = 51349;


/************************************************************
 * Returns a uniform random number between 0.0 and 1.0. Set
 * idum to any negative value to initialize or reinitialize
 * the sequence
************************************************************/
class ran1  {
   int *idum;
   static long ix1,ix2,ix3;
   static float r[98];
   float  temp;
   static int iff;
   int    j;
   void nrerror(char*);

public:
   ran1 (int iff = 0);
```

```
   float value (int *idum);
};
/***********************************************************
 * Returns an exponentially distributed, positive, random
 * derived of unit mean, using ran1(idum) as the source of
 * the uniform random number
***********************************************************/
class os_rand : public ran1 {
public:
   float value(int*);
   int generate();
   int interarrival_time();
   int service_time();
};

#include <iostream.h>
#include <stdlib.h>
#include <math.h>
#include "/t/opsys/phase3/arr_srv.h"

ran1::ran1(int ff) { iff = ff;}
inline float ran1::value (int *idum) {
   if (*idum < 0 || iff == 0) {
                    /* initialize on first call */
                    /* even if  idum is not negative.     */

      iff=1;
      ix1=(IC1-(*idum)) % M1;
                     /* seed the first routine, */
                     /* and use it to seed the second*/
                     /* and third routines.*/
      ix1=(IA1*ix1+IC1) % M1;
      ix2=ix1 % M2;
      ix1=(IA1*ix1+IC1) % M1;
      ix3=ix1 % M3;

      for (j=1;j<=97;j++) {
                /* fill the table with sequential uniform */
                /* number generated by the first two   */
                /* routines.                  */
       ix1=(IA1*ix1+IC1) % M1;
       ix2=(IA2*ix2+IC2) % M2;
       r[j]=(ix1+ix2*RM2)*RM1;
                /* Low- & high-order are combined here. */
      } /* endfor */
      *idum=1;
   } /* endif */

   ix1=(IA1*ix1+IC1) % M1;
   ix2=(IA2*ix2+IC2) % M2;
          /* start. Generate the next number for each     */
          /* sequence.*/
   ix3=(IA3*ix3+IC3) % M3;
```

```
   j=1 + ((97*ix3)/M3);
          /* use the third sequence to get an integer    */
          /* between 1 and 97.        */
   if (j > 97 || j < 1) nrerror("RAN1: This cannot
happen.");
   temp=r[j];
          /* Return that table entry, */
   r[j]=(ix1+ix2*RM2)*RM1;
   return temp;
}

inline float os_rand::value(int *idum) {
   return -log((float) ran1::value(idum));
}
inline int os_rand::generate()
{  int xidum,
          /* seed */
       xnums,
          /* number of random numbers generated */
      k;
     float temp1;
     int   temp2;
     xidum = time(NULL)%7;
      temp1 = value(&xidum);
      temp2 = temp1*10+15;
      temp2 = temp2/3;
     return(temp2);
}
int os_rand::interarrival_time(){ return(generate()); }
int os_rand::service_time(){ return(generate()); }
void ran1::nrerror(error_text)
char error_text[];
{
     cerr << "Numerical Recipes run-time error...\n";
     cerr << "%s\n",error_text;
     cerr << "...now exiting to system...\n";
     exit(1);
}
```

APPENDIX E

USER MANUAL

1. Simulating an Environment

Creating objects from the loader, memory, or cpu classes is done independently. In other words, different memories, loaders, and/or CPU's can be created each with its unique features as depicted below. These instantiations can communicate easily in a parallel processing environment. For example, one loader can load jobs into a memory while one CPU (or more) execute other jobs in the same memory at the same time, also probably in other memories at the same time.

```
loader l1,l2,l3;
                        //declaration of three loaders
memory m1(128,5),m2,m3;
                        //declaration of three memories
ins_set inst1;
                        //declaration of an instruction set
ins_set2 inst2;
                        //declaration of a differen
                        //instruction set
cpu c1(&ins1),c2(&ins2),c3(&ins2);
                        //declaration of three cpu's
```

Examples of component declarations

Two parallel processing cases are simulated below to show how easy it is to use the package to model of a multi-processor system. After declaring a system's components (loaders, memories, and CPU's), it is straightforward to have different processes each use its own loader on its own memory and CPU.

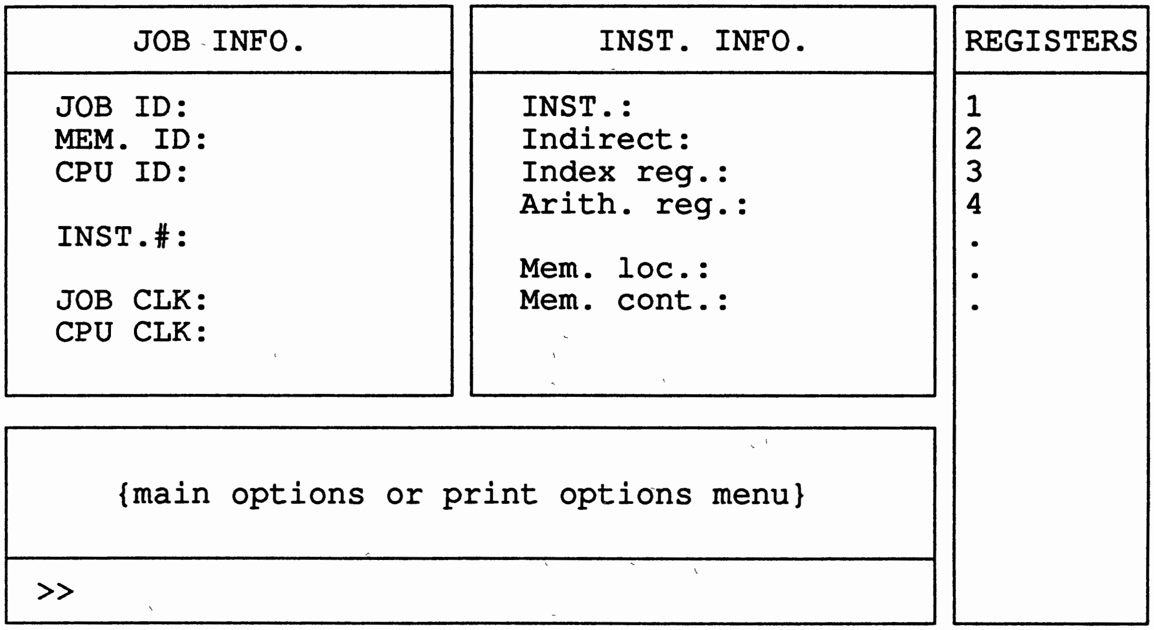
```
loader l1,l2;
                        //declaration of two loaders
memory m1,m2;
                        //declaration of two memories
ins_set inst1;
                        //declaration of an instruction set
cpu c1(&ins1),c2(&ins1);
                        //declaration of two cpu's

int i=fork();

if (i=0) {
        l1.load(jobs1,m1);
                        //load jobs from file jobs1 into memory
        c1.run_job_from(m1);

                        //run a ready job from memory m1 using
                        //cpu1
        }
else {
        l2.load(jobs2,m2);
                        //load jobs from file jobs2 in memory m2
        c2.run_job_from(m2);
                        //run a ready job from memory m2 using
                        //cpu2
        }
```

Two separate systems load and execute their jobs in parallel

Furthermore, in a more complicated system, we may have a shared memory in which more than one loader can load new jobs at the same time, and more than one CPU may execute different ready jobs from this memory. This case will require the addition of a semaphore in the class memory_table to protect its elements from being accessed by CPU's and loaders - one for all the table - or the addition of a semaphore to class mem_element so that no more than one CPU or loader can access it, but more than one memory element can be accessed at the same time.

```
loader l1;
        //declare a loader
memory m1(128,5),m2,m3;
        //declare three memories
ins_set inst1;
        //declare an instruction sets
ins_set2 inst2;
        //declare a different instruction set
cpu c1(&ins1),c2(&ins2),c3(&ins2);
        //declare three cpu's
l1.load(jobs1,m1);
        //load jobs from jobs1 file into memory m1
l1.load(jobs2,m2);
        //load jobs from jobs2 file into memory m2
l1.load(jobs3,m3);
        //load jobs from jobs3 file into memory m3
int i=fork();
if (i=0) {c1.run_job_from(m1);
        //run a ready job from memory m2 using
        //cpu1
        c2.run_job_from(m2); }
        //run a ready job from memory m2 using
        //cpu2
        else c3.run_job_from(m3);
        //run a ready job from memory m2 using
        //cpu3
```

Load jobs in sequence using one loader and execute two of them in parallel with the third



The debugger interface

2. Interface

A default debugger was implemented to serve as an interface to the prototype system. A user can design his/her own debugger as needed.

1- The default Debugger has four windows:

- REGISTER window, which displays the current register values.
- INST. INFO. window, which explains the current instruction.
- JOB INFO. window, which contains general information about the job.
- Options window, which contains user options.

2- Since the class debugger uses the class my_window, the user simply can overload it to add more features or create his/her new customized debugger by using the class my_window.

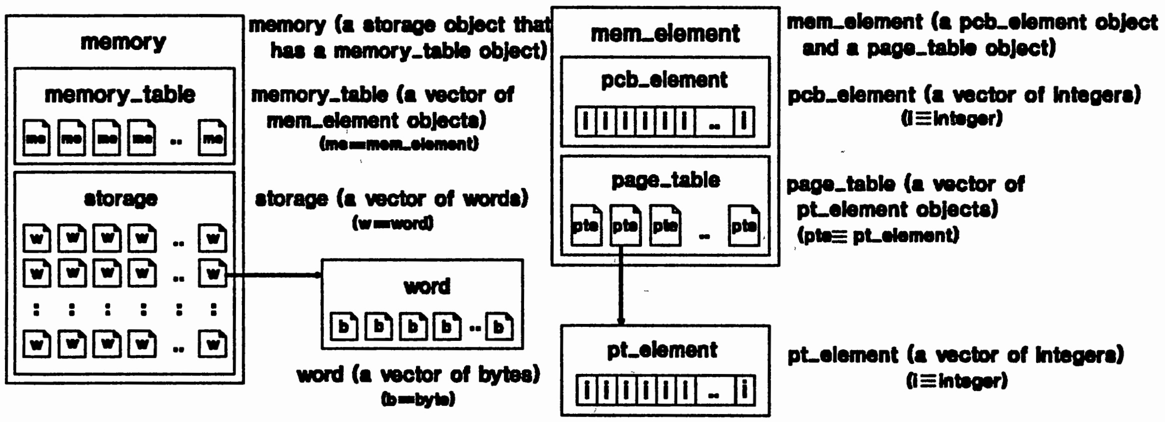
3- The class my_window is easy to use because:

- Its constructor has the number of its variables.
- Has set() method to set the location of each variable.
- To update a variable, it just needs to call the update() method with the variable number and its new value.

By using parallel processing functions we can run both parts of the package in parallel. While the debugger is displaying a job's execution steps for the user, the main program can execute another job and prepare its execution information in a special file for the debugger.

APPENDIX F

PROGRAMMER MANUAL



Relations among classes

1. Relations among Classes

Different relations among the package's classes (see the above depiction) are represented using object-oriented programming features such as the following.

A- We have single inheritance in this prototype system ("is a" relation). Examples include the classes byte, pt_element, and pcb_element from the class vect, the class register from the class word, and the class memory from the class storage.

B- We also have multiple inheritance of the class mem_element from the classes pcb_element and page_table.

C- There is another relation among classes besides inheritance, some objects have object instance variables from other classes ("has a" relation). For example an object of the class storage has an array of "word" objects, an object of the class mem_element has a "pcb_element" object and the class page_table object has an array of "pt_element" objects. This case is more complicated in the class memory object which has a "memory_table" object, array of "mem_element" objects, and its body is an array of "word" objects.

2. Inherit New Classes

If the package classes do not meet the user needs, new classes can be inherited from the package classes and the user can overload the original class' methods. In the example below, a new instruction set class is defined which does not have the RD and WR operations.

```
class my_ins_set:public ins_set {
int execute(int& EA,memory& m,word& REG,FILE* dbg, FILE* out)
        {int flag=0;
        switch(op.int_b())
                {       case 12 :
                        //RD instruction
                        case 13 : flag = 2; break;
                        //WR instruction
                        default : ins_set::execute(EA,m,REG,dbg,out);
                        //call the parent instruction set execute operation
                }
        }
}
```

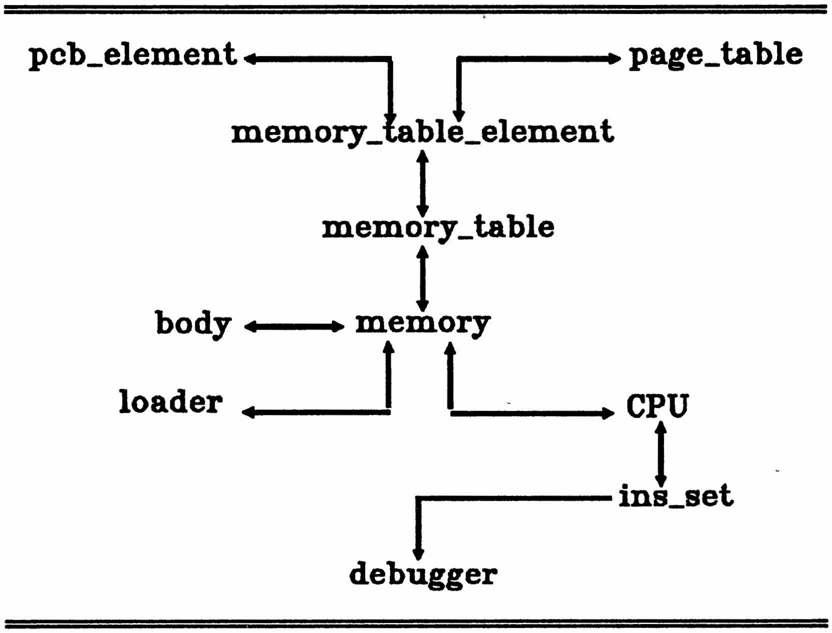
Example of a new ins_set class

3. Communication among Objects

The processing of the prototype system is based on the communication among its objects as outlined below.

A- The class loader and its interaction with the class memory and instances of the class memory:

- A loader object interacts with a memory object by calling loader.load(jobs_file,memory).
- The load() function communicates with the memory element memory_table by using memory.put(vect) to write the new job information into an element of its pcb_elements.
- The load() function also uses the write() function to communicate with the memory body to write a new word into the memory location by using the overloading operator = in the class word.



Communication among classes.

B- The CPU communicates with the memory to find a ready job from the memory_table and calls its inst_set object to execute the ready job's instructions one by one from the memory body.

C- The inst_set communicates with the CPU's registers and the memory body during each instruction's execution.

D- The Debugger communicates with the ins_set to receive its needed information about the current instruction to be displayed to the package user.

VITA

Khaled M. Hassan

Candidate for the Degree of

Master of Science

Thesis: AN OBJECT-ORIENTED PROTOTYPING ENVIRONMENT FOR ARCHITECTURES AND OPERATING SYSTEMS

Major Field: Computer Science

Biographical:

Personal Data: Born in Cairo, Egypt, October 18, 1964, the eldest son of Mr. and Mrs. M. H. Soliman.

Education: Received Bachelor of Civil Engineering from Faculty of Engineering, Cairo University, Cairo, Egypt, in June 1986; completed requirements for the Master of Science degree at Oklahoma State University in December 1992.

Professional Experience: Data Analyst and System Designer in the MIS Department in a Canadiane project with Water Research Center, an Egyptian Research Institute, for hydraulic studies of the River Nile, February 1989 to August 1992.