A Hierarchical Memory System Environment

*J. Djordjevic, A. Milenkovic, S.Prodanovic**
Faculty of Electrical Engineering, University of Belgrade
P.O.Box 3554, 11120 Belgrade, Yugoslavia
**Informatika, Jevrejska 32, 11000 Belgrade, Yugoslavia*
E-mail: jdjordjevic@kiklop.etf.bg.ac.yu

Abstract: *The paper presents an environment for teaching elements of a computer system memory hierarchy. It is made up of a hierarchical memory system, a reference manual, a software package and a set of laboratory experiments. The hierarchical memory system is devised to cover the virtual memory and translation lookaside buffer, the cache memory, and the main memory. The reference manual provides all implementation details with the appropriate circuits drawings and detailed descriptions. For the devised hierarchical memory system a software package, which includes the graphical simulator with the accompanying tools, is developed. They allow one to carry out the simulation down to the register transfer level by executing a set of laboratory experiments*

1 Introduction

The hierarchical memory system is one of the topics extensively covered in the third year course in Advanced computer architecture and organization at the Faculty of Electrical Engineering, University of Belgrade, attended by the students at the Department of Computer Science. The students acquire the knowledge necessary for following such a course through the first year course in Fundamentals of Computer Science and the second year courses in Programming methodology and languages and Computer architecture and organization. The course in Computer architecture and organization covers the basic concepts related to the most commonly found structure of a computer which includes the processor, the memory, the input/output subsystem and the bus. The course in Advanced computer architecture and organization goes one step further covering topics such as the architecture and organization of CISC and RISC processors, the organization of pipelined processors, the storage system, the interconnection networks, and the memory system [1].

The memory system is presented as a hierarchical one, made up of the virtual memory and translation lookaside buffer, the cache memory and the main memory. The secondary memory, although part of the hierarchical memory system, is studied within the storage system. The virtual memory and translation lookaside buffer is studied by introducing the notions of virtual and real address spaces and the need for mapping the virtual into real addresses. Special attention is given to the separation of activities carried out by the operating system and the translation lookaside buffers. The cache memory begins by pointing out at the behavior of programs, which display temporal and spatial locality. It follows by the study of the associative, direct and set associative mapping techniques of main memory blocks into cache memory blocks, the cache memory replacement algorithms, the main memory updating policies and some other techniques that might improve the performance of the cache memory. The main memory study concentrates at the techniques, such as the memory interleaving, used to increase the memory bandwidth.

Generally, a great problem in teaching any course in the field of computer architecture and organization is how to provide means which would facilitate the students to make a cognitive leap from the blackboard description of the architecture and organization of a computer to its utilization as a programmable device and connect their theoretical knowledge with practical experience. This problem the authors had treated in another paper concerning their course in Computer architecture and organization and came to the decision to devise and develop their own educational environment [2, 3]. Based on the number of discussions conducted with the students that have used it in the laboratory experiments and the results of the exams, the authors are convinced that the educational environment has been a power aid in teaching the course in Computer architecture and organization. This resulted in the development of similar environments as help in teaching other topics for some other courses in the field of computer architecture and organization. One of them, the hierarchical memory system environment used within the course in Advanced computer architecture and organization, is the subject of this paper.

The hierarchical memory system environment (the HMS environment), which includes the Hierarchical Memory System (HMS), the reference manual for it, and the Software Package of the HMS (SPHMS), is used by the students to carry out the laboratory experiments. The HMS is devised to cover all concepts concerning the virtual memory and translation lookaside buffer, the cache memory and the memory interleaving and its detailed description is given in the accompanying manual [4]. The HMS is described in Section 2. The SPHMS includes the tools for initializing the appropriate parts of the HMS, and the graphical simulator of all implementation details of the HMS down to the register transfer level. A brief description of the SPHMS is given in Section 3. A set of laboratory experiments, devised to demonstrate to the students the situations of interest during their practical work with HMS, is presented in Section 4. Section 5 contains the conclusion.

2 The hierarchical memory system

The hierarchical memory system is made up of the virtual memory and translation lookaside buffer, the cache memory and the main memory. The initial idea was to develop such a hierarchical memory system where the process of accessing the translation lookaside buffer, the cache memory and the interleaved memory is realized as a whole. In addition to that, the hierarchical memory system was envisaged to be such that all most commonly used techniques for realizing the virtual memory and translation lookaside buffer, the cache memory and the interleaved memory could be demonstrated. Finally, the aim was to develop a user friendly environment which would make it possible to follow the above process at the clock level and examine, at any time, the values of all signals of the hierarchical memory system down to the register transfer level. Although the realization of such an HMS environment was feasible, the authors felt that such a system would be too complex and very difficult to be used by the students. Therefore, it was decided to split up the HMS environment into three separate entities for the virtual memory and translation lookaside buffer, the cache memory and the interleaved memory. Their detailed description is given in the accompanying manual written specifically for the course in Advanced computer architecture and organization [4].

2.1 The virtual memory and translation lookaside buffer

The virtual memory and translation lookaside buffer (TLB) are demonstrated by using three types of lookaside buffer and three types of virtual memory. The TLB holds certain number of descriptors of most frequently accessed pages or segments. For any address translation request, coming from the processor, the TLB checks, according to the mapping technique implemented, whether the descriptor is in it. If the descriptor is in the TLB, the virtual to real address translation is carried out and the real address returned to the processor. In the case of the TLBs for segmented and segment paged virtual memories the access and address violation checks is performed. If any or both checks fail, the TLB generates an interrupt. If the descriptor is not in the TLB, it goes into the appropriate segment or/and page table(s) and checks the appropriate descriptor. If the segment or page is in the main memory, its descriptor is loaded in the appropriate entry in the TLB. Now, for the same address translation request the TLB check is carried out again. The activities are now the same as for the above explained case when the descriptor is in the TLB. If the segment or page is not in the main memory, the TLB generates an interrupt. The TLB entry to be replaced with a new descriptor is selected according to the Least Recently Used (LRU) algorithm for the TLB with the associative mapping and according to either the LRU or First In First Out (FIFO) algorithm for the TLB with the set associative mapping.

The processor sending address translation requests to the TLB is simulated. The HMS environment allows a user to initialize a table in the simulated processor with address translation requests and time intervals between them before the simulation starts. Once the simulation is started, the simulated processor according to the information obtained from the table will generate address translation requests.

The virtual memory is realized by interrelated activities of the TLB and the operating system. Therefore, some of the operating system activities dealing with situations when the processor and the TLB are switched from a process to a process, when an interrupt is generated by the TLB etc. are also simulated. Here again the HMS environment allows a user to specify parameters relevant for the operating system activities in such situations.

2.2 The cache memory

The cache memory is demonstrated by using three types of cache memory realized with the direct, associative and set associative mapping. The cache memory holds certain number of most frequently accessed blocks from the main memory. For any main memory access request, coming from the processor, the cache memory checks, according to the mapping technique implemented, whether the block is in it. If the block is in the cache memory, the requested access is carried with the cache memory, and, in the case of the read access request, the data read is returned to the processor. If the block is not in the cache memory, the block is transferred from the main memory into the cache memory. For the same memory access request the cache memory check is carried out again. The activities are now the same as for the above explained case when the block is in the cache memory. The cache memory block to be replaced with a new block is selected according to the First In First Out (FIFO) algorithm for the cache memories with the associative and set associative mapping.

Some other activities in the cache memory depend on whether the write back or store through technique for updating the main memory is used. The cache memory with the direct mapping uses the write back technique. As a consequence, the cache memory block, selected to be replaced with a new block, is first returned to the main memory, and then the new block is transferred from the main memory into the cache memory. However, this is done only if at least one write access for the cache memory block to be replaced has been carried out. The cache memory with the associative mapping uses the store through technique. Therefore, there is no need to return to the main memory the block selected to be replaced with a new block. The cache memory with the set associative mapping uses the write back technique with buffering. The cache memory block, selected to be replaced with a new block, is returned to the main memory only if at least one write access for that cache memory block has been carried out. However, this block is first written only into a buffer block, then the new block is transferred from the main memory into the cache memory, and, finally, the block selected to be replaced is transferred from the buffer block into the main memory.

The processor is realized in a similar way as for the virtual memory.

2.3 The interleaved main memory

The interleaved main memory is demonstrated in a system made up 16 units, 16 memory modules, a split transactions synchronous bus and the arbiter.

Each unit can be configured to generate either the single word accesses or the block accesses. The single word accesses are typical for a processor fetching instructions or accessing scalar values or a direct memory access controller working with a slow peripheral device. The block accesses are typical for a processor transferring blocks between the cache memory and the main memory or a direct memory access controller working in the burst mode with a fast peripheral device. A unit includes the bus interface and the requester. The bus interface includes all circuitry necessary to perform the appropriate operations when the unit is either a master or a slave. When the unit is a master, it sends a bus request to the arbiter and performs a read request or write request cycle when it gets the grant. When the unit is a slave, it accepts data, requested by an earlier read request cycle, in a data available cycle. The requester simulates parts of a processor or a direct memory access controller which generate the memory read and write requests and accepts data returned as a response to a read request. It is realized in a similar way as the processor for the virtual and cache memories.

A memory module contains the bus interface and the RAM memory. The bus interface contains all circuitry necessary to perform the appropriate operations when the unit is either a slave or a master. When the module is a slave, it accepts a write or read request from the bus and initiates the write or read operation in the RAM module. As the result of the read operation performed in the RAM module, the bus interface obtains data requested by a bus read request cycle, and starts with activities as a bus master. It sends a bus request to the arbiter and performs a data available cycle when it gets the grant. The RAM memory of the module is used to store data. The HMS environment allows a user to initialize locations of interest in the simulated RAM memory and specify the desired access time for each memory module separately. Each memory module can obtain its number in the range from 0 to 15. In addition to that, five possible ways of interleaving memory modules can be specified. One of them is consecutive addresses in the same module, the other one is consecutive addresses in 16 consecutive modules and the remaining three are possible cases of mixtures of the first two.

The bus is realized as a split transactions synchronous bus. The bus cycles performed are the write request, the read request and the data available. In order to make it possible for the module to return data to the unit that initiated a read request cycle, the unit sends its identifier with the read request cycle. When the data requested are read, the module as the master uses the identifier in the data available cycle to send it to the appropriate unit.

The arbiter realizes the parallel arbitration between all units and modules. Each of them is connected with the arbiter with a pair of lines. One of them is used for sending a request to the arbiter, and the other one is used for receiving a grant from the arbiter. The memory modules are given higher priority than the units.

3 The Software Package of the HMS

The Software Package of the HMS (SPHMS) contains the simulators that provide the graphical presentation of all implementation details of the HMS down to the register transfer level and makes it possible to follow the functioning of all its parts. The SPHMS offers various facilities such as the interactive setting and examination of the contents of memory locations, registers etc., the drawing of timing diagrams of any of the selected signals of the HMS during the execution of the appropriate operation, etc. All these features are provided in a user friendly manner by extensive use of modern tools for the development of Windows applications.

The facilities offered by the SPHMS make it possible to initialize the HMS and run the simulators.

3.1 Initialization of the HMS

The first step in the initialization of the HMS is the selection of one of three simulators for the virtual memory and TLB, the cache memory, and the interleaved memory. For the virtual memory and the cache memory one must further select one of three types of TLBs and cache memories, respectively. For the interleaved memory one must specify one of five possible ways of interleaving memory modules, assign numbers to modules, define for each unit whether it is the one with the single word or the block access, assign identifiers to units, etc.

The next step in the initialization of the HMS is the loading of the appropriate register and memory locations with initial values. In the case of the virtual memory it includes the initialization of the table of the simulated processor, some registers in the TLB, and parts of the main memory containing the page or/and segment tables. In the case of the cache memory it includes the initialization of the table of the simulated processor, some registers in the cache memory and parts of the main memory accessed by the cache memory. For the interleaved main memory it includes the loading of the table of the simulated requester and the memory module locations accessed by the units.

The initialization of the HMS can be done either interactively or by invoking a file. The file may be created by using the **Save** command either immediately after the HMS is interactively initialized or at any moment of the simulation. The interactive initialization is predominantly used by instructors to carefully prepare laboratory experiments. The initialization by invoking a file is normally used by students.

3.2 Running the Simulator

The running of graphical simulators for the virtual memories, the cache memories, and the interleaved memory is similar for all three cases. Therefore, the running of a simulator is demonstrated using the simulator for the interleaved memory.

The simulator is realized as a hierarchical scheme of screens. Each screen is made up of two windows. The larger window in the upper part of the screen, named the block diagram window, contains either a composition of combinational and sequential circuits, if this is a leaf block in the hierarchical scheme, or a composition of subblocks, that can be further selected, and combinational and sequential circuits, if this is not a leaf block in the hierarchical scheme. The smaller window in the lower part of the screen is divided into the information window at its left hand side and the command window at the right hand side. The information window gives the value of the step counter and the control signals generated for a particular clock period and a brief explanation of the actions that are going to take place during that clock period. The command window contains navigation command buttons **Top**, **Back**, and **Up**, simulation command button **Clk+**, and miscellaneous command buttons **Show**, **Save**, **Help**, and **End**. The information field **Clock** displays the number of clock cycles executed.

The **Up** button allows one to move from the current screen to the screen one level up in the hierarchy, **Back** allows one to go back to the previous screen, and **Top** allows one to move directly from the current screen to the top screen (Fig. 1). The **Clk+** button allows one to continue with the simulation for one clock period. The **Show** button opens the window, which allows the interactive initialization and examination of memory modules and units or drawing the timing diagram of selected signals from the beginning of the simulation until the current clock period. The **Reset** button clears the current state of simulation and returns it to the beginning. The **Help** button activates the help system where all details concerning the functioning of the HMS and its simulators are available. The **Save** button makes it possible to save the current status of simulation into a file and use it later for the initialization of the HMS. The **End** button allows one to exit the simulator.

The running of the simulator with the hierarchy of screens of the HMS is described briefly in the following. The first screen, with which the simulation begins, shows the block structure of the system with the interleaved main memory (Fig. 1). The values for groups of lines, such as for the data (DBUS) and address bus (ABUS) lines, are given in the hexadecimal form, while the single lines (WRBUS, RDBUS, DABUS, ACKBUS) are colored either in blue or red depending on whether the signal on that line has logical value zero or one, respectively.

If a more detailed structure of any of the units, the memory modules or the arbiter from Fig. 1. is needed, one can move one level down in the hierarchy by positioning the cursor and clicking the mouse button. As an example of this, one can assume that the cursor is positioned at **Unit0..3** on the screen given in Fig. 1 which includes four units (Unit0 to Unit3). After the mouse button is clicked a pop-up window appears to allow selection of a unit from this group. What appears is the screen giving the block structure of the Unit0, which is configured as a single word unit (Fig. 2). From this screen one can go one level down and get more detailed structure of the Bus Interface (Processing Unit and the Control Unit) and Requester. By positioning the cursor at the Processing Unit and clicking the mouse button, one goes one level down in the hierarchy and gets the design of this block at the register transfer level (Fig. 3). The horizontal navigational boxes (hatched gray on figures) allow one to follow relevant signals at the same level of hierarchy. Thus, by positioning the cursor at the one of the hatched gray boxes labeled as “control unit” and clicking the mouse button one can move to the Control Unit screen (Fig 4). Using the Back command button one can move back to the Processing Unit screen. The same effect can be achieved if one uses the hatched gray boxes labeled as “processing unit”. Since the control unit does not fit into the block diagram window, one can use the scrolling facility to examine all parts of the control unit (Fig. 4). The block structure of the requester of the Unit 0 is shown in Fig. 5. The requester reads the request table, which contains a sequence of, read and write requests and time intervals between adjacent requests. The request table can be filled and examined either during the initialization or at any time during the simulation using the **Show** button (Fig. 6).

If the cursor is positioned at the unit configured for the block access the screen obtained is similar as the one given in Fig. 2. However, since this processing unit is more complex than the one for the single word access, it is divided into three blocks named the input buffer, the output buffer, and the address and identifier registers. By positioning the cursor at the one of them and clicking the mouse each of these blocks can be shown in detail as given in Fig. 7, Fig. 8, and Fig. 9. It should be noted that the input buffer and the output buffer blocks can not be fitted on the screen. Therefore, the scrolling bar facility is being used as shown in Fig. 7 and Fig. 8.

The same procedure one can use to display either the processing unit or the control unit of a memory module. If one selects the processing unit of memory module 0 the screen displayed is shown in Fig. 10.

The control units for the units with the block access and the memory modules are similar as the one for Unit 0 with the single word accesses (Fig. 4.).

The parallel arbiter, made up of the 32 bit priority encoder, decoder, and register, is shown in Fig. 11.

4 The organization of laboratory experiments

The practical work with the HMS is organized through seven laboratory experiments the duration of which is two hours each. The first three laboratory experiments cover the TLB with the associative, direct and set associative mappings for the segmented, paged and segment paged organizations of virtual memories, respectively. The next three laboratory experiments cover the cache memory with the associative, direct, and set associative mappings combined with the write back and the store through memory updating policies, the FIFO and LRU block replacement algorithms, and the techniques such as the write buffering, the accessing critical word first, the bypassing and the early processor start. The seventh laboratory experiment covers the parallel arbitration between 16 units and 16 memory modules, and the bus transactions on the split transactions synchronous bus. This experiment is carried out for each of the five possible ways of interleaving memory modules.

For each laboratory experiment there are certain number of carefully chosen problems. They are prepared in an attempt to cover all typical situations in which the TLBs, the cache memories and the system with the interleaved memory can be found. The students are requested to go through all prepared problems at the clock level. The control signals that appear in the information box (Fig. 4) point out at the parts of the particular unit or module where an action is going to take place. Based on this, the students should locate relevant parts of the HMS and examine the values of signals at the register transfer level using the SPHMS navigation facilities.

At the end of the appropriate laboratory experiment each student starts program TEST that is devised to assess his knowledge. For each laboratory experiment there is a pool of relevant questions. After verification of the student the TEST randomly generates ten questions from the pool. The period of time within which the student should answer the questions is limited to ten minutes. The TEST is connected with the database of students, which attend the course. This makes it possible to update the student's records.

5 Conclusion

The hierarchical memory system environment used for the course in Advanced Computer Architecture and Organization is presented in this paper. It includes the Hierarchical Memory System, the reference manual for it, the Software Package of the Hierarchical Memory System, and a set of laboratory experiments. The Hierarchical Memory System is made up of the virtual memories and TLBs, the cache memories, and the system with interleaved memory. The facilities offered by the Software Package of the Hierarchical Memory System are illustrated using the system with the interleaved memory.

The hierarchical memory system environment is based on the originally developed and already used methodology for creating educational environments for teaching various topics in the field of computer architecture and organization [2]. The first step in the methodology is the logical design of the appropriate part of a computer system down to the register transfer level. The next step is the development of the graphical simulator with accompanying tools. They allow one to initialize the system and carry out the simulation during the execution of the prepared problems, which demonstrate the situations of interest. The simulation is done at the clock level in a user friendly fashion.

The environments developed with the described methodology have been used for teaching courses in computer architecture and organization for several years. They have been useful aids and favorably accepted by the students. The authors feel that the students are rigidly lead in this environment through the predetermined situations leaving little room for their initiative and creativity. Therefore, the related ongoing research is directed towards the development of a user friendly environment that would allow ones to design their own parts of computer systems using the library of standard combinational and sequential modules.

6 References

[1] D. A. Patterson, J. L. Hennessy, *Computer Architecture: A Quantitative Approach*, Morgan Kaufman, San Francisco, California, USA, 1996.

[2] J. Djordjevic, A. Milenkovic, N. Grbanovic, M. Bojovic, "*An Educational Environment for Teaching a Course in Computer Architecture and Organization*," Proceedings of the 4th Annual Workshop on Computer Architecture Education, Las Vegas, NV, January 1998.

[3] J. Djordjevic, "Computer Architecture and Organization – The reference manual for the Educational Computer System," Faculty of Electrical Engineering, University of Belgrade, 1995.

[4] J. Djordjevic, "Computer Architecture and Organization – The reference manual for the Hierarchical Memory System", Faculty of Electrical Engineering, University of Belgrade, 1998.

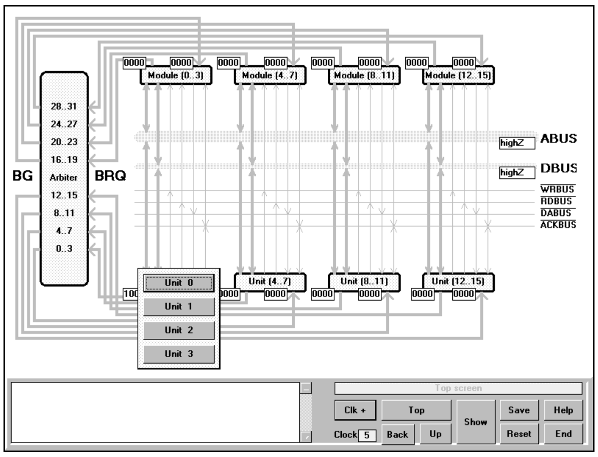
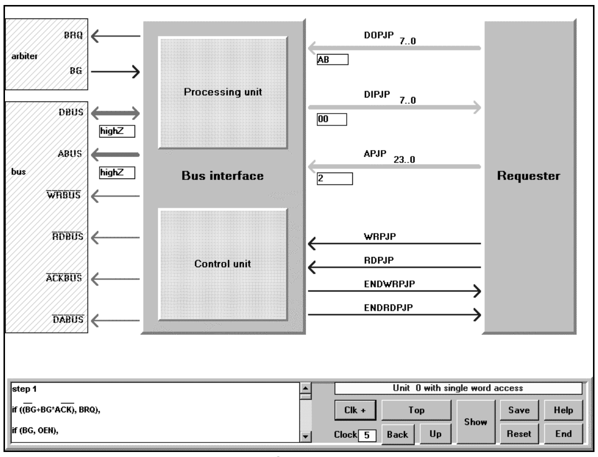
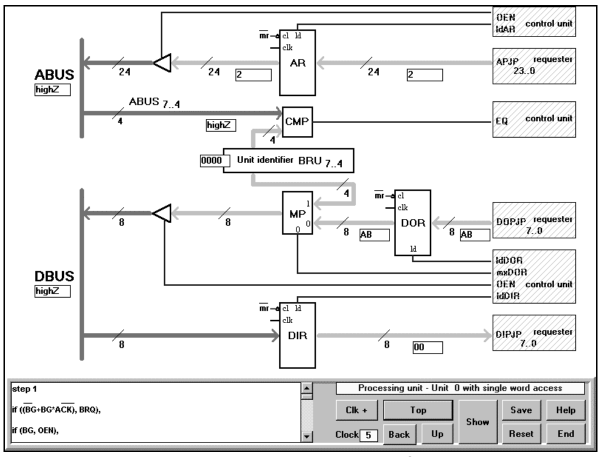


Figure 1. System with memory interleaved organization



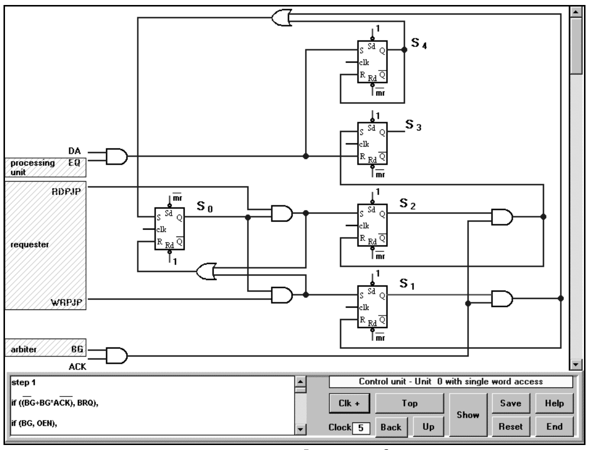
step 1
if ((BG+BG*ACK), BRQ),
if (BG, OEN),

Figure 2. Block structure of Unit 0 with single word access



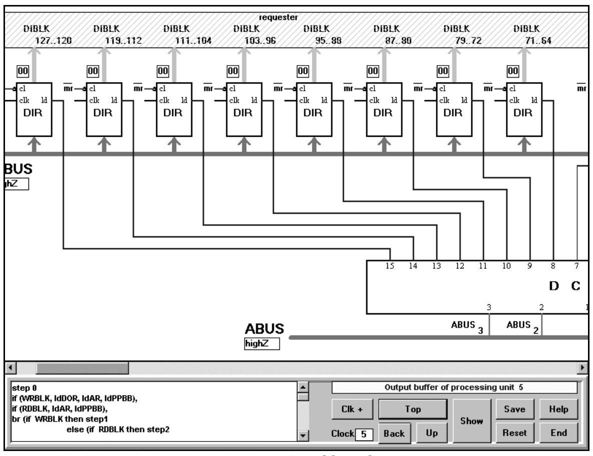
step 1
if ((BG+BG*ACK), BRQ),
if (BG, OEN),

Figure 3. Processing unit of Unit 0



```
step 1
if ((BG+BG'ACK), BRQ),
if (BG, OEN),
```

Figure 4. Control unit of Unit 0



```
step 0
if (WRBLK, ldDOR, ldAR, ldPPBB),
if (RDBLK, ldAR, ldPPBB),
br (if  WRBLK then step1
        else (if  RDBLK then step2
```

Figure 8. Input buffer for Unit 4

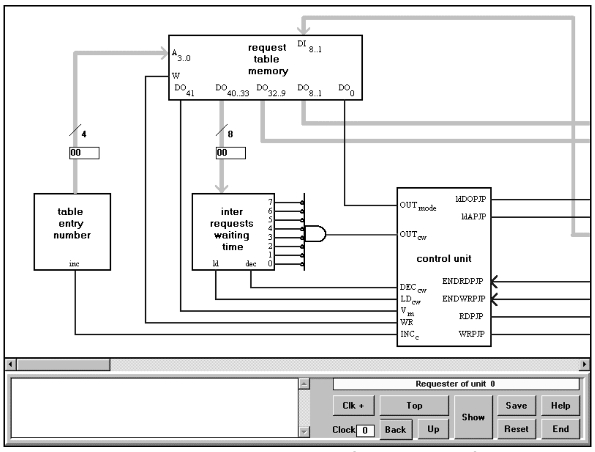
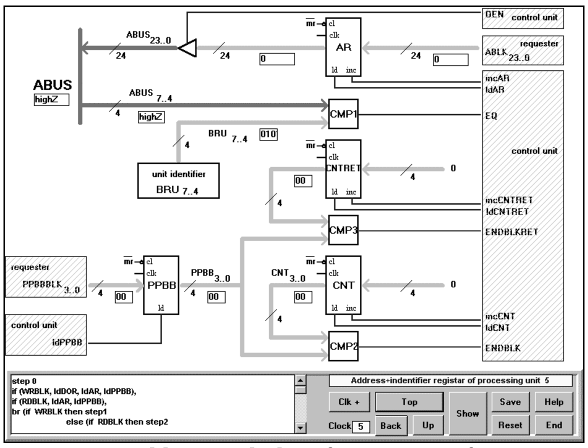
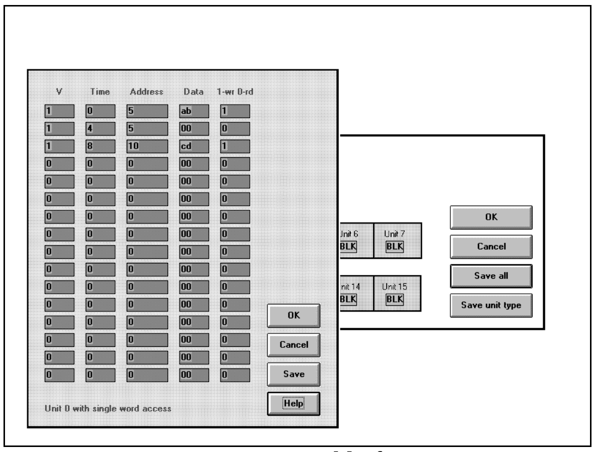


Figure 5. Block structure of requester for Unit 0



```
step 0
if (WRBLK, ldDOR, ldAR, ldPPBB),
if (RDBLK, ldAR, ldPPBB),
br (if  WRBLK then step1
        else (if  RDBLK then step2
```

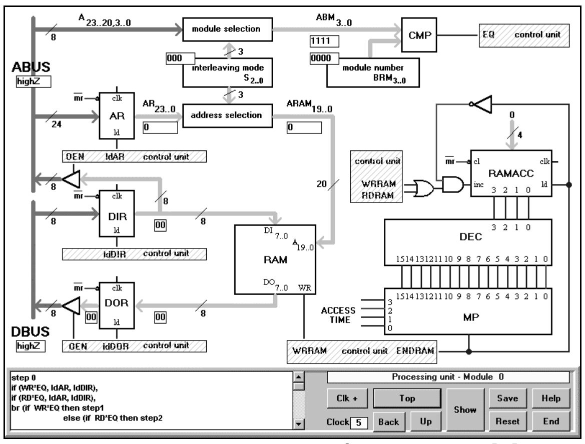
Figure 9. Address and identifier registers for Unit 4



V	Time	Address	Data	1-wr 0-rd
1	0	5	ab	1
1	4	5	00	0
1	8	10	cd	1
0	0	0	00	0
0	0	0	00	0
0	0	0	00	0
0	0	0	00	0
0	0	0	00	0
0	0	0	00	0
0	0	0	00	0
0	0	0	00	0
0	0	0	00	0
0	0	0	00	0
0	0	0	00	0
0	0	0	00	0
0	0	0	00	0

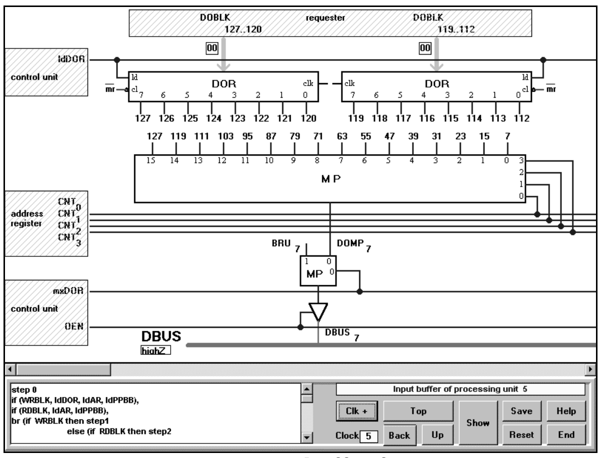
Unit 0 with single word access

Figure 6. Request table for Unit 0



```
step 0
if (WR'EQ, ldAR, ldDIR),
if (RD'EQ, ldAR, ldDIR),
br (if  WR'EQ then step1
        else (if  RD'EQ then step2
```

Figure 10. Processing Unit of memory module 0



```
step 0
if (WRBLK, ldDOR, ldAR, ldPPBB),
if (RDBLK, ldAR, ldPPBB),
br (if  WRBLK then step1
        else (if  RDBLK then step2
```

Figure 7. Output buffer for Unit 4

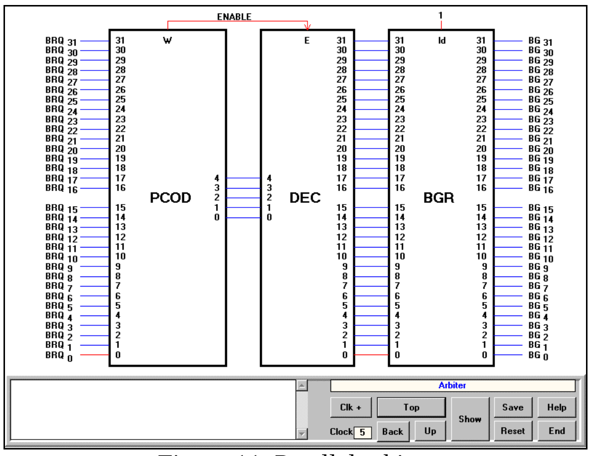


Figure 11. Parallel arbiter