Replacement Techniques

A number of replacement techniques can be used . These include
1- Random selection of a cache block for replacement is done based on the output of the random number generator at the time of replacement. This technique is simple and does not require much additional overhead

2- Frst-In-Frst-Out(FIFO) technique takes the time spent by a block in the cache as a measure for replacement . The block that has been in the cache the longest is selected for replacement regardless of the recent pattern of access to the block. This technique requires keeping track of the life time of a cache block. Therefore, it is not as simple as the random selection technique. Intuitively, the FIFO technique is reasonable to use for straight line programs where locality of reference is not of concern.

3- Least Recently Used (LRU)replacement technique, the cache block that has been recently used the least is selected for replacement. Among the three replacement techniques, the LRU technique is the most effective. This is because the history of block usage (as the criterion for replacement) is taken into consideration. The LRU algorithm requires the use of a cache controller circuit that keeps track of references to all blocks while residing in the cache. This can be achieved through a number of possible implementations. Among these implementations is the use of counters. In this case each cache block is assigned a counter. Upon a cache hit,

Input/ output (I/O)&Direct Memory Access(DMA)

1- Input/ Output (I/O) Concept

Figure(1) shows a simple arrangement for connecting the processor and the memory in a given computer system to an input device, for example, a keyboard and an output device such as a graphic display. A single bus consisting of the required address, data, and control lines is used to connect the system‟s components in the Figure below.
Figure: Single Bus System
here concerned with the way which the processor and the I/O devices exchange data. There exists a big difference in the rate at which a processor can process information and those of input and output devices. One simple way to accommodate this speed difference is to have the input device, for example, a keyboard, deposit the character struck by the user in a register (input register), which indicates the availability of that character to the processor. When the input character has been taken by the processor, this will be indicated to the input device in order to proceed and input the next character, and so on. Similarly, when the processor has a character to output (display), it deposits it in a specific register dedicated for communication with the graphic display (output register). When the character has been taken by the graphic display, this will be 18

indicated to the processor such that it can proceed and output the next character, and so on. This simple way of communication between the processor and I/O devices, called I/O protocol, requires the availability of the input and output registers. In a typical computer system, there is a number of input registers, each belonging to a specific input device. There is also a number of output registers,
each belonging to a specific output device. More than one arrangement exists to satisfy the above mentioned requirements. Such as shared I/O arrangement figure(2), and Memory_Mapped arrengement. In addition, a mechanism according to which the processor can address those input and output registers must be adopted such as programmed I/O, Interrupt-Driven I/O, and Direct Memory access.
Figure: Shared I/O arrengment.

2. Programmed I/O

This protocol has to be programmed in the form of routines that run under the control of the CPU. Consider, for example, an input operation from 19

Device 6 (could be the keyboard) in the case of shared I/O arrangement. Let us also assume that there are eight different I/O devices connected to the processor in this case figure(3). The following protocol steps (program) have to be followed:
1. The processor executes an input instruction from device 6, for example, INPUT 6. The effect of executing this instruction is to send the device number to the address decoder circuitry in each input device in order to identify the specific input device to be involved. In this case, the output of the decoder in Device #6 will be enabled, while the outputs of all other decoders will be disabled.
2. The buffers (in the figure we assumed that there are eight such buffers) holding the data in the specified input device (Device #6) will be enabled by the output of the address decoder circuitry.
3. The data output of the enabled buffers will be available on the data bus.
4. The instruction decoding will gate the data available on the data bus into the input of a particular register in the CPU, normally the accumulator. Output operations can be performed in a way similar to the input operation explained above. The only difference will be the direction of data transfer, which will be from a specific CPU register to the output register in the specified output device
They are performed under the CPU control. A complete instruction fetch, decode,and execute cycle will have to be executed for every input and every output operation. Programmed I/O is useful in cases where by one character at a time is to be transferred, for example, keyboard and character mode printers. Although simple, programmed I/O is slow. 20

Figure: I/O connection to the processor

3. Interrupt-Driven I/O

It is often necessary to have the normal flow of a program interrupted, for example, to react to abnormal events, such as power failure. In the case of an I/O interrupt, serving an interrupt means to perform the required data transfer. Upon finishing serving an interrupt, the processor should restore the original status by popping the relevant values from the stack. Once the processor returns to the normal state, it can enable sources of interrupt again.

One important point that was overlooked in the above scenario is the issue of serving multiple interrupts, for example, the occurrence of yet another interrupt while the processor is currently serving an interrupt. 21

Response to the new interrupt will depend upon the priority of the newly arrived interrupt with respect to that of the interrupt being currently served.
- If the newly arrived interrupt has priority less than or equal to that of the currently served one, then it can wait until the processor finishes serving the current interrupt.
- If, on the other hand, the newly arrived interrupt has priority higher than that of the currently served interrupt.
For example, power failure interrupt occurring while serving an I/O interrupt, then the processor will have to push its status onto the stack and serve the higher priority interrupt.

4-Direct Memory Access (DMA)
We have discussed the data transfer between the processor and no devices. We have discussed two different approaches namely prograrhmed I/O and Intpt-driven tro Both the methods require the active intervention of the processor to transfer data between memory and the I/O module, and any data transfer must transverse a path through the processor. Thus both these forms of tro suffer from two inherent drawbacks.
1- The I/O transfer rate is limited by the speed with which the processor can test and service adevice.
2- The processor is tied up in managing an I/O transfer; a number of instructions must be executed for each I/O transfet.
To transfer large block of data at high speed, a special control unit may be provided to allow transfer of a block of data directly between an external device and the main memory, without continuous intervention by the processor. This approach is called direct memory access or DMA. 22

DMA transfers are performed by a control circuit associated with the I/O device and this circuit is referred as DMA controller. The DMA controller allows direct data transfer between the device and the main memory without involving the processor.
To transfer data between memory and I/O devices, DMA controller takes over the control of the system from the processor and transfer of data take place over the system bus. For this purpose, the DMA controller must use the bus only when the processor does not need it, or it must force the processor to suspend operation temporarily. The later technique is more common and is referred to as cycle stealing, because the DMA module in effect steals a bus cycle.
Figure: DMA Block daigram
When the processor wishes to read or write a block of data, it issues a command to the DMA module, by sending to the DMA module the following information.
1-Whether a read or write is requested, using the read or write control line between the processor and the DMA module.
2-The address of the I/O devise involved, communicated on the data lines.