UNIT IV

Memory Organization

12-1 MEMORY HIERARCHY

Memory hierarchy in a computer system :

Memory hierarchy system consists of all storage devices employed in a computer system from the slow but high capacity auxiliary memory to a relatively faster main memory, to an even smaller and faster cache memory accessible to the high speed processing logic.

- **Main Memory**: memory unit that communicates directly with the CPU (RAM)
- **Auxiliary Memory**: device that provide backup storage (Disk Drives)
- **Cache Memory**: special very-high-speed memory to increase the processing speed (Cache RAM)

Figure 12-1 Memory hierarchy in a computer system.

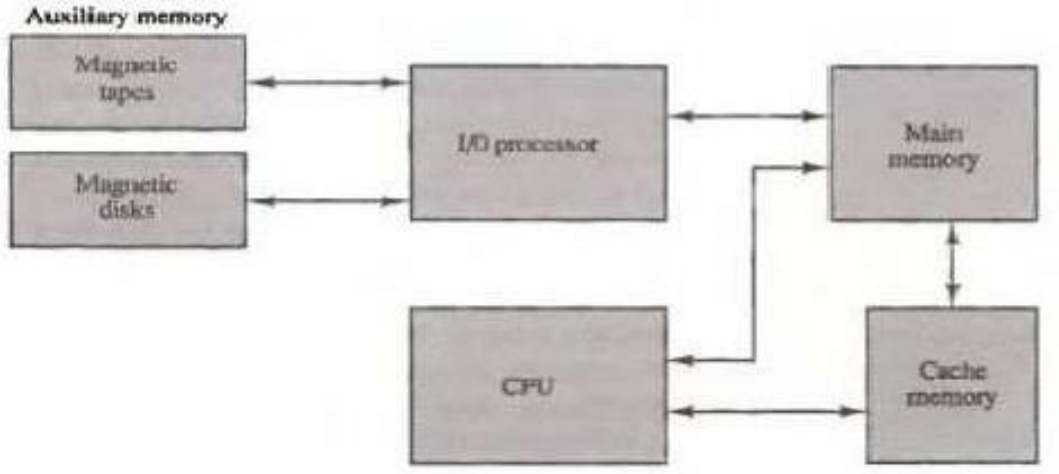


Figure12- 1 illustrates the components in a typical memory hierarchy. At the bottom of the hierarchy are the relatively slow magnetic tapes used to store removable files. Next are the Magnetic disks used as backup storage. The main memory occupies a central position by being able to communicate directly with CPU and with auxiliary memory devices through an I/O process. Program not currently needed in main memory are transferred into auxiliary memory to provide space for currently used programs and data.

The cache memory is used for storing segments of programs currently being executed in the CPU. The I/O processor manages data transfer between auxiliary memory and main memory. The auxiliary memory has a large storage capacity is relatively inexpensive, but has low access speed compared to main memory. The cache memory is very small, relatively expensive, and has very high access speed. The CPU has direct access to both cache and main memory but not to auxiliary memory.

Multiprogramming:

Many operating systems are designed to enable the CPU to process a number ofindependent programs concurrently.

Multiprogramming refers to the existence of 2 or more programsin different parts of the memory hierarchy at the same time.

Memory management System:

The part of the computer system that supervises the flow of information between auxiliary memory and main memory.

12 – 2 MAIN MEMORY

Main memory is the central storage unit in a computer system. It is a relatively large and fast memory used to store programs and data during the computer operation. The principal technology used for the main memory is based on semi conductor integrated circuits. Integrated circuits RAM chips are available in two possible operating modes, static and dynamic.

- Static RAM – Consists of internal flip flops that store the binary information.
- Dynamic RAM – Stores the binary information in the form of electric charges that are applied to capacitors.

Most of the main memory in a general purpose computer is made up of RAM integrated circuit chips, but a portion of the memory may be constructed with ROM chips.

- Read Only Memory –Store programs that are permanently resident in the computer and for tables of constants that do not change in value once the production of the computer is completed.

The ROM portion of main memory is needed for storing an initial program called a Bootstrap loader.

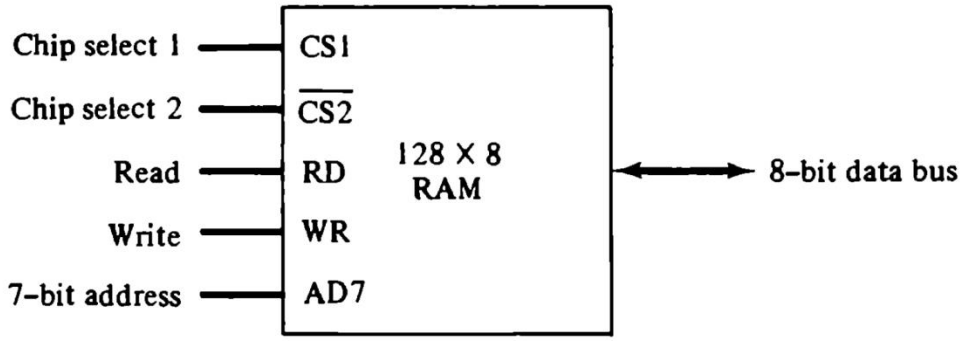
- Boot strap loader –function is start the computer software operating when power is turned on.
- Boot strap program loads a portion of operating system from disc to main memory and control is then transferred to operating system.

with CPU

RAM and ROM CHIP

- RAM chip –utilizes bidirectional data bus with three state buffers to perform communication

Figure 12-2 Typical RAM chip.



(a) Block diagram

CS1	$\overline{CS2}$	RD	WR	Memory function	State of data bus
0	0	×	×	Inhibit	High-impedance
0	1	×	×	Inhibit	High-impedance
1	0	0	0	Inhibit	High-impedance
1	0	0	1	Write	Input data to RAM
1	0	1	×	Read	Output data from RAM
1	1	×	×	Inhibit	High-impedance

(b) Function table

The block diagram of a RAM Chip is shown in Fig.12-2. The capacity of memory is 128 words of eight bits (one byte) per word. This requires a 7-bit address and an 8-bit bidirectional data bus. The read and write inputs specify the memory operation and the two chips select (CS) control inputs are enabling the chip only when it is selected by the microprocessor. The read and write inputs are sometimes combined into one line labelled R/W.

The function table listed in Fig.12-2(b) specifies the operation of the RAM chip. The unit is in operation only when CS1=1 and CS2=0.The bar on top of the second select variable indicates that this input is enabled when it is equal to 0. If the chip select inputs are not enabled, or if they are enabled but the read or write inputs are not enabled, the memory is inhibited and its data bus is in a high-impedance state. When CS1=1 and CS2=0, the memory can be placed in a write or read mode. When the WR input is enabled, the memory stores a byte from the data bus into a location specified by the address input lines. When the RD input is enabled, the content of the selected byte is placed into the data bus. The RD and WR signals control the memory operation as well as the bus buffers associated with the bidirectional data bus.

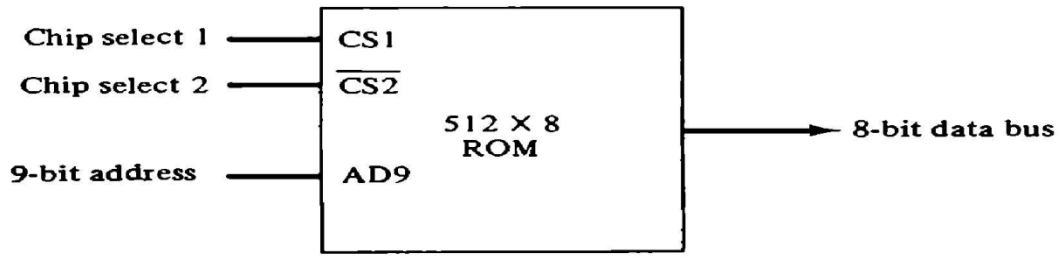


Figure 12-3 Typical ROM chip.

A ROM chip is organized externally in a similar manner. However, since a ROM can only read, the data bus can only be in an output mode. The block diagram of a ROM chip is shown in fig.12-3. The nine address lines in the ROM chip specify any one of the512 bytes stored in it. The two chip select inputs must be CS1=1 and CS2=0 for the unit to operate. Otherwise, the data bus is in a high-impedance state.

Memory Address Map

The interconnection between memory and processor is then established from knowledge of the size of memory needed and the type of RAM and ROM chips available. The addressing of memory can be established by means of a table that specify the memory address assigned to each chip. The table called Memory address map, is a pictorial representation of assigned address space for each chip in the system.

TABLE 12-1 Memory Address Map for Microprocomputer

Component	Hexadecimal address	Address bus									
		10	9	8	7	6	5	4	3	2	1
RAM 1	0000–007F	0	0	0	x	x	x	x	x	x	x
RAM 2	0080–00FF	0	0	1	x	x	x	x	x	x	x
RAM 3	0100–017F	0	1	0	x	x	x	x	x	x	x
RAM 4	0180–01FF	0	1	1	x	x	x	x	x	x	x
ROM	0200–03FF	1	x	x	x	x	x	x	x	x	x

The memory address map for this configuration is shown in table 12-1. The component column specifies whether a RAM or a ROM chip is used. The hexadecimal address column assigns a range of hexadecimal equivalent addresses for each chip. The address bus lines are listed in the third column. The RAM chips have 128 bytes and need seven address lines. The ROM chip has 512 bytes and needs 9 address lines.

Memory Connection to CPU:

RAM and ROM chips are connected to a CPU through the data and address buses. The low order lines in the address bus select the byte within the chips and other lines in the address bus select a particular chip through its chip select inputs.

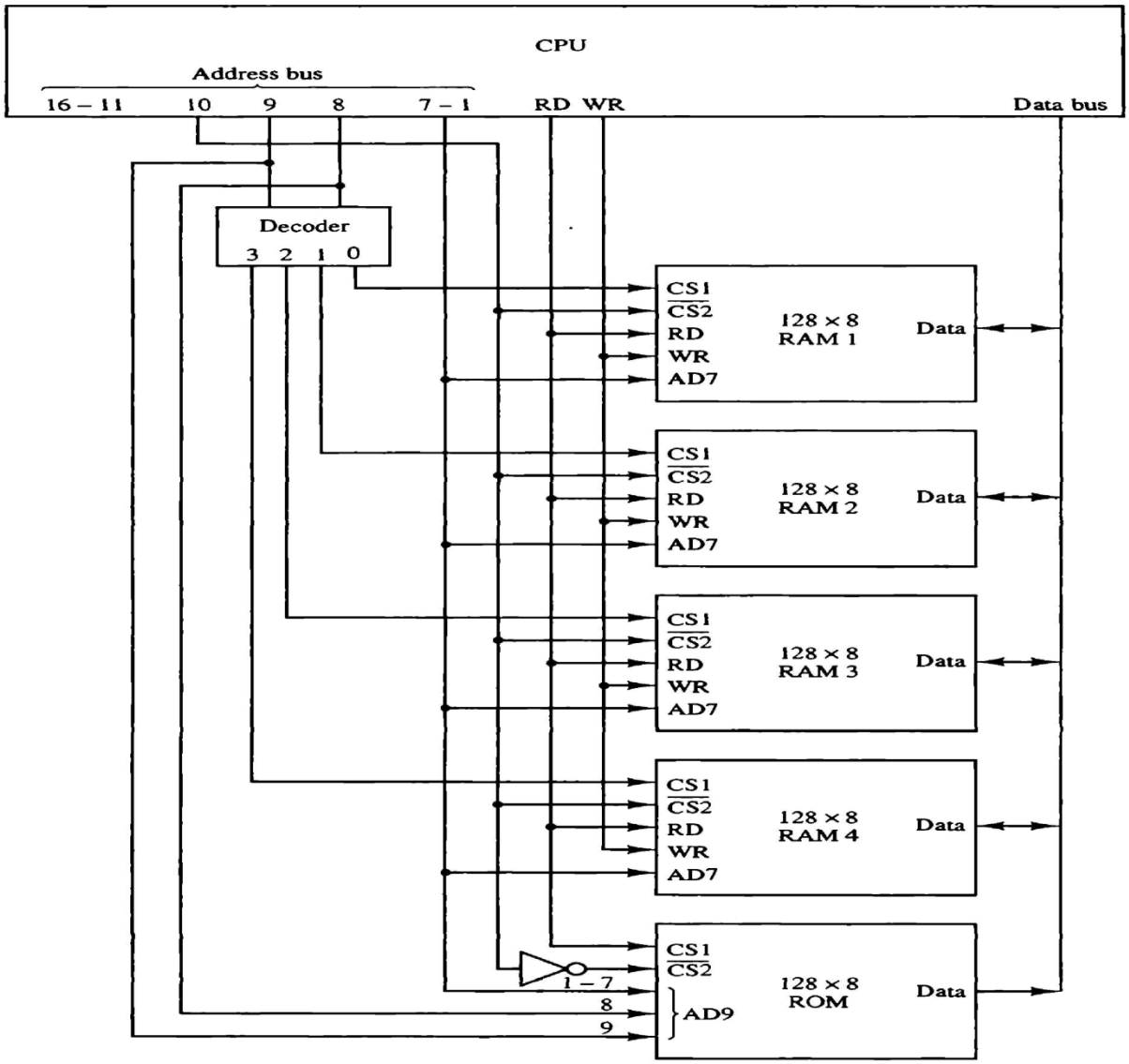


Figure 12-4 Memory connection to the CPU.

The connection of memory chips to the CPU is shown in Fig.12-4. This configuration gives a memory capacity of 512 bytes of RAM and 512 bytes of ROM. Each RAM receives the seven low-order bits of the address bus to select one of 128 possible bytes. The particular RAM chip selected is determined from lines 8 and 9 in the address bus. This is done through a 2 X 4 decoder whose outputs go to the CS1 inputs in each RAM chip. Thus, when address lines 8 and 9 are equal to 00, the first RAM chip is selected. When 01, the second RAM chip is select, and so on. The RD and WR outputs from the microprocessor are applied to the inputs of each RAM chip. The selection between RAM and ROM is achieved through bus line 10. The RAMs are selected when the bit in this line is 0, and the ROM when the bit is 1. Address bus lines 1 to 9 are applied to the input address of ROM without going through the decoder. The data bus of the ROM has only an output capability, whereas the data bus connected to the RAMs can transfer information in both directions.

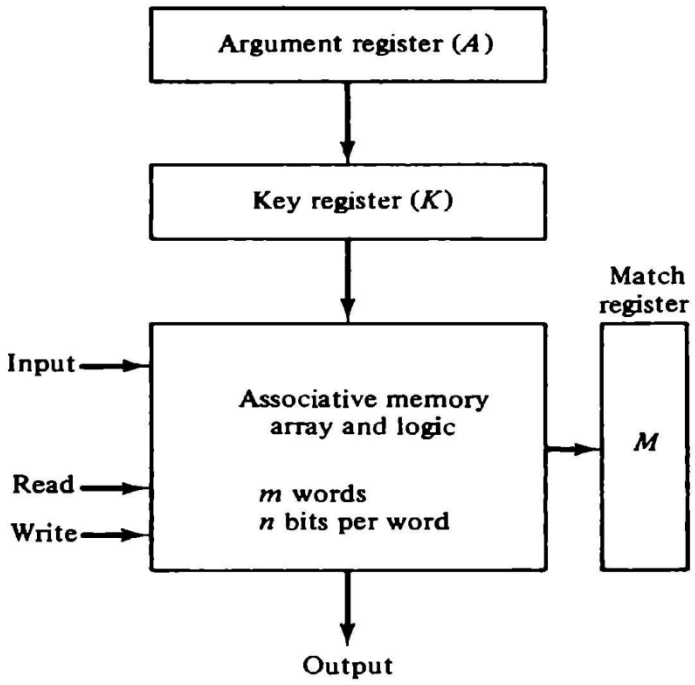
12-3 AUXILIARY MEMORY

The time required to find an item stored in memory can be reduced considerably if stored data can be identified for access by the content of the data itself rather than by an address. A memory unit accessed by content is called an associative memory or content addressable memory (CAM).

- CAM is accessed simultaneously and in parallel on the basis of data content rather than by specific address or location
- Associative memory is more expensive than a RAM because each cell must have storage capability as well as logic circuits
- Argument register –holds an external argument for content matching
- Key register –mask for choosing a particular field or key in the argument word

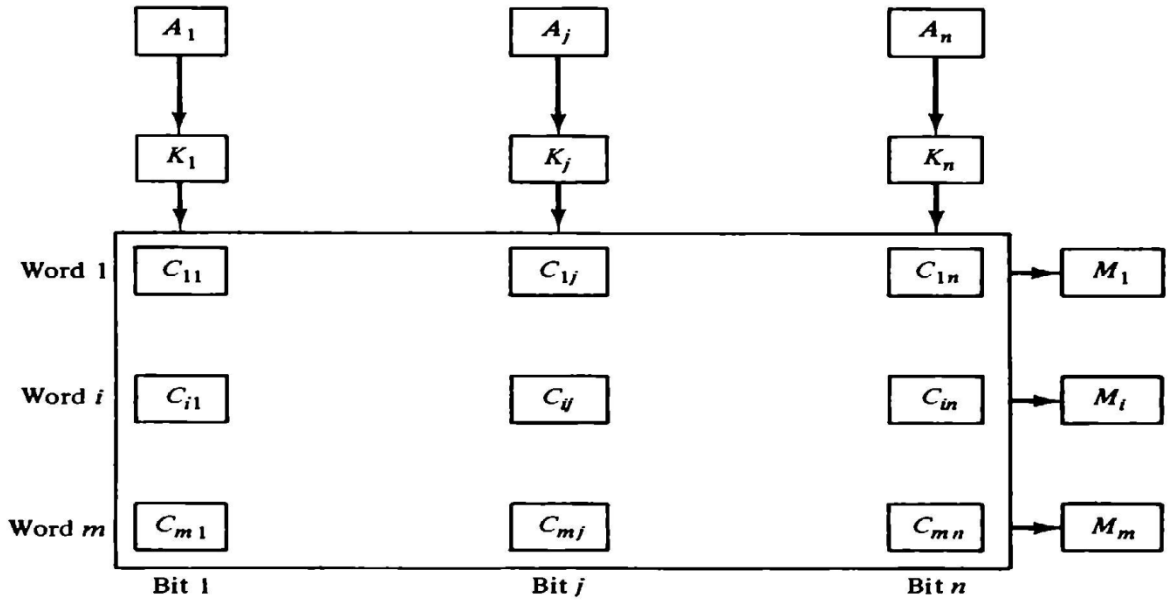
Hardware Organization

Figure 12-6 Block diagram of associative memory.



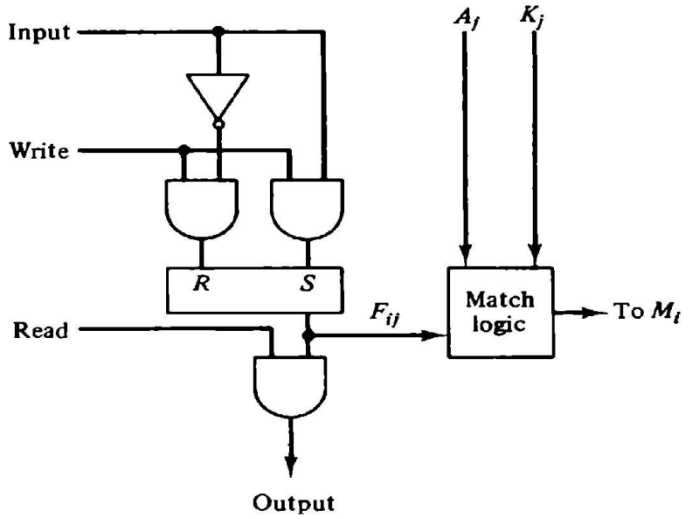
It consists of a memory array and logic for m words with n bits per word. The argument register A and key register K each have n bits, one for each bit of a word. The match register M has m bits, one for each memory word. Each word in memory is compared in parallel with the content of the argument register. The words that match the bits of the argument register set a corresponding bit in the match register. After the matching process, those bits in the match register that have been set bindicate the fact that their corresponding words have been matched. Reading is accomplished by a sequential access to memory for those words whose corresponding bits in the match register have been set.

Figure 12-7 Associative memory of m word, n cells per word.



The relation between the memory array and external registers in an associative memory is shown in Fig.12-7. The cells in the array are marked by the letter C with two subscripts. The first subscript gives the word number and second specifies the bit position in the word. Thus cell C_{ij} is the cell for bit j in word i. A bit A_j in the argument register is compared with all the bits in column j of the array provided that k_j =1.This is done for all columns j=1,2,....n. If a match occurs between all the unmasked bits of the argument and the bits in word I, the corresponding bit M_i in the match register is set to 1. If one or more unmasked bits of the argument and the word do not match, M_i is cleared to 0.

Figure 12-8 One cell of associative memory.



It consists of flip-flop storage element F_{ij} and the circuits for reading, writing, and matching the cell. The input bit is transferred into the storage cell during a write operation. The bit stored is read out during a read operation. The match logic compares the content of the storage cell with corresponding unmasked bit of the argument and provides an output for the decision logic that sets the bit in M_i.

Match Logic

The match logic for each word can be derived from the comparison algorithm for two binary numbers. First, neglect the key bits and compare the argument in A with the bits stored in the cells of the words.

Word i is equal to the argument in A if $A_j = F_{ij}$ for j=1,2,.....,n. Two bits are equal if they are both 1 or both 0. The equality of two bits can be expressed logically by the Boolean function

$$x_j = A_j F_{ij} + A_j \text{ '} F_{ij} \text{ '}$$

where $x_j = 1$ if the pair of bits in position j are equal;otherwise , $x_j = 0$. For a word i is equal to the argument in A we must have all x_j variables equal to 1. This is the condition for setting the corresponding match bit M_i to 1. The Boolean function for this condition is

$$M_i = x_1 x_2 x_{3.....} x_n$$

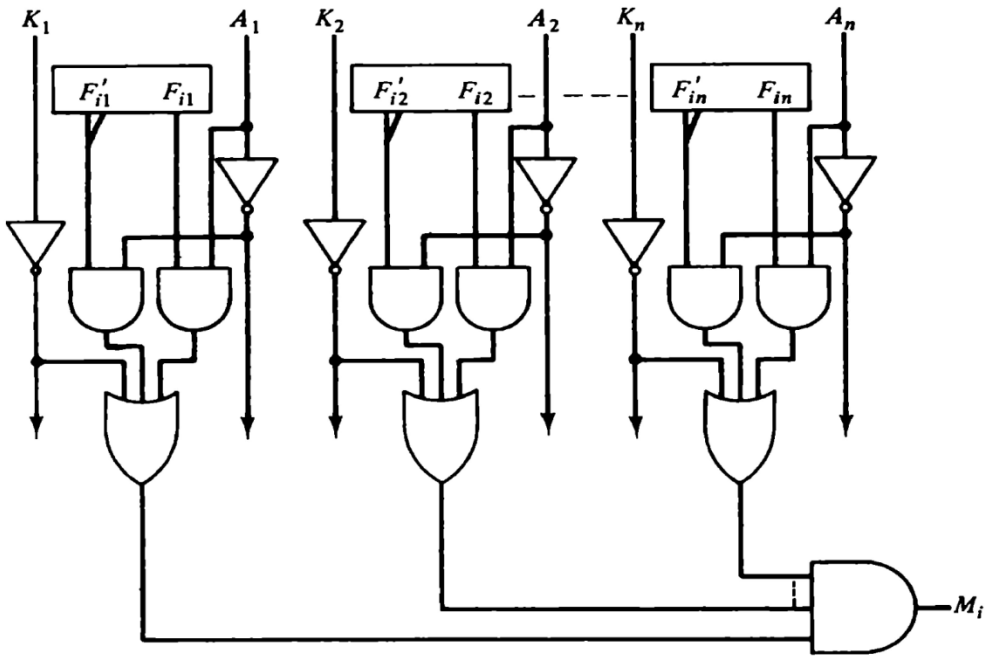


Figure 12-9 Match logic for one word of associative memory.

E

ach cell requires two AND gate and one OR gate. The inverters for A and K are needed once for each column and are used for all bits in the column. The output of all OR gates in the cells of the same word go to the input of a common AND gate to generate the match signal for M_i. M_i will be logic 1 if a match occurs and 0 if no match occurs.

Read and Write operation

Read Operation

- If more than one word in memory matches the unmasked argument field , all the matched words will have 1's in the corresponding bit position of the match register
- In read operation all matched words are read inλ sequence by applying a read signal to each word line whose corresponding Mi bit is a logic 1
- In applications where no two identical items are stored in the memory , only one word may match , in which case we can use Mi output directly as a read signal for the corresponding word

Write Operation

Can take two different forms

1. Entire memory may be loaded with new information

2.Unwanted words to be deleted and new words to be inserted

1.Entire memory : writing can be done by addressing each location in sequence – This makes it random access memory for writing and content addressable memory for reading – number of lines needed for decoding is d Where m = 2 d , m is number of words.

2.Unwanted words to be deleted and newλ words to be inserted :

- Tag register is used which has as many bits as there are words in memory
- For every active (valid) word in memory , the corresponding bit in tag register is set to 1
- When word is deleted the corresponding tag bit is reset to 0
- The word is stored in the memory by scanning the tag register until the first 0 bit is encountered After storing the word the bit is set to 1.

12-5 CACHE MEMORY

- Effectiveness of cache mechanism is based on a property of computer programs called "**locality of reference**"
- The references to memory at any given time interval tend to beλ confined within a localized areas
- Analysis of programs shows that most of their execution time is spent on routines in which instructions are executed repeatedly These instructions may be – loops, nested loops , or few procedures that call each other
- Many instructions in localized areas of program are executed
- repeatedly during some time period and reminder of the program is accessed infrequently This property is called "Locality of Reference".

Locality of Reference

Locality of reference is manifested in two ways :

1. Temporal- means that a recently executed instruction is likely to be executed again very soon.
    - The information which will be used in near future is likely toλ be in use already(e.g. reuse of information in loops)
2. Spatial- means that instructions in close proximity to a recently executed instruction are also likely to be executed soon
    - If a word is accessed, adjacent (near) words are likely to be accessed soon (e.g. related data items (arrays) are usually stored together; instructions are executed sequentially)

3. If active segments of a program can be placed in afast (cache) memory , then total execution time can be reduced significantly
4. Temporal Locality of Reference suggests whenever an information (instruction or data) is needed first , this item should be brought in to cache
5. Spatial aspect of Locality of Reference suggests that instead of bringing just one item from the main memory to the cache ,it is wise to bring several items that reside at adjacent addresses as well (ie a block of information)

Principles of cache

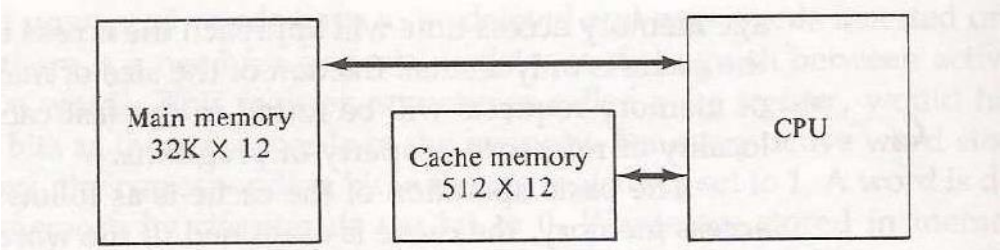


Figure 12-10 Example of cache memory.

The main memory can store 32k words of 12 bits each. The cache is capable of storing 512 of these words at any given time. For every word stored , there is a duplicate copy in main memory. The Cpu communicates with both memories. It first sends a 15 bit address to cahache. If there is a hit, the CPU accepts the 12 bit data from cache. If there is a miss, the CPU reads the word from main memory and the word is then transferred to cache.

- When a read request is received from CPU,contents of a block of memory words containing the location specified are transferred in to cache
- When the program references any of the locations in this block , the contents are read from the cache Number of blocks in cache is smaller than number of blocks in main memory
- Correspondence between main memory blocks and those in the cache is specified by a mapping function
- Assume cache is full and memory word not in cache is referenced
- Control hardware decides which block from cache is to be removed to create space for new block containing referenced word from memory
- Collection of rules for making this decision is called "**Replacement algorithm** "

Read/ Write operations on cache

- **Cache Hit Operation**
  - CPU issues Read/Write requests using addresses thatλ refer to locations in main memory
  - Cache control circuitry determines whether requested word currently exists in cache
  - If it does, Read/Write operation is performed on the appropriate location in cache **(Read/Write Hit)**

Read/Write operations on cache in case of Hit

- In Read operation main memory is not involved.
- In Write operation two things can happen.

1.Cache and main memory locations are updated simultaneously **(" Write Through ")** OR

2. Update only cache location and mark it as " Dirty or Modified Bit " and update main memory location at the time of cache block removal **(" Write Back " or " Copy Back ")** .

Read/Write operations on cache in case of Miss Read Operation

- When addressed word is not in cache Read Miss occurs there are two ways this can be dealt with

1.Entire block of words that contain the requested word is copied from main memory to cache and the particular word requested is forwarded to CPU from the cache **(Load Through)** (OR)

2.The requested word from memory is sent to CPU first and then the cache is updated **(Early Restart)**

Write Operation

- If addressed word is not in cache Write Miss occurs
- If write through protocol is used information is directly written in to main memory
- In write back protocol , block containing the word is first brought in to cache , the desired word is then overwritten.

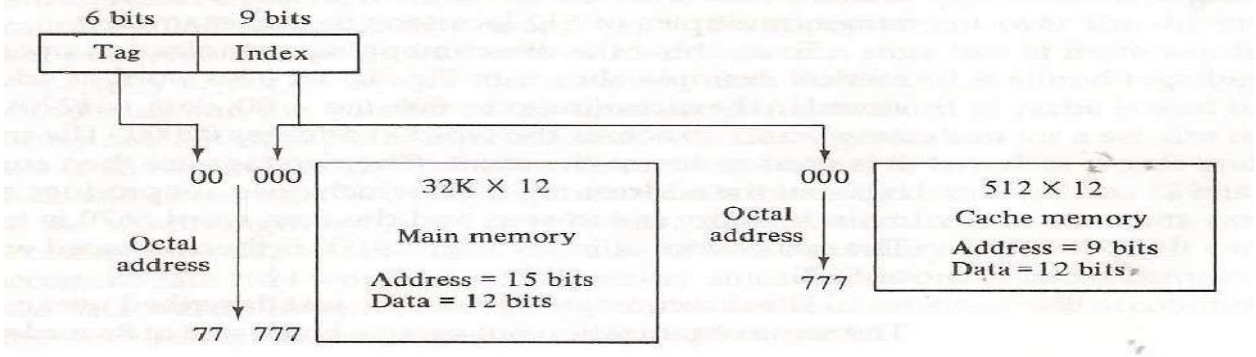
Mapping Functions

- Correspondence between main memory blocks and those in the cache is specified by a memory mapping function
- There are three techniques in memory mapping
    1. Direct Mapping
    2. Associative Mapping
    3. Set Associative Mapping

Direct mapping:

A particular block of main memory can be brought to a particular block of cache memory. So, it is not flexible.

Figure 12-12 Addressing relationships between main and cache memories.



In fig 12-12. The CPU address of 15 bits is divided into two fields. The nine least significant bits constitute the index field and remaining six bits form the tag field. The main memory needs an address

that includes both the tag and the index bits. The number of bits in the index field is equal to the number of address bits required to access the cache memory.

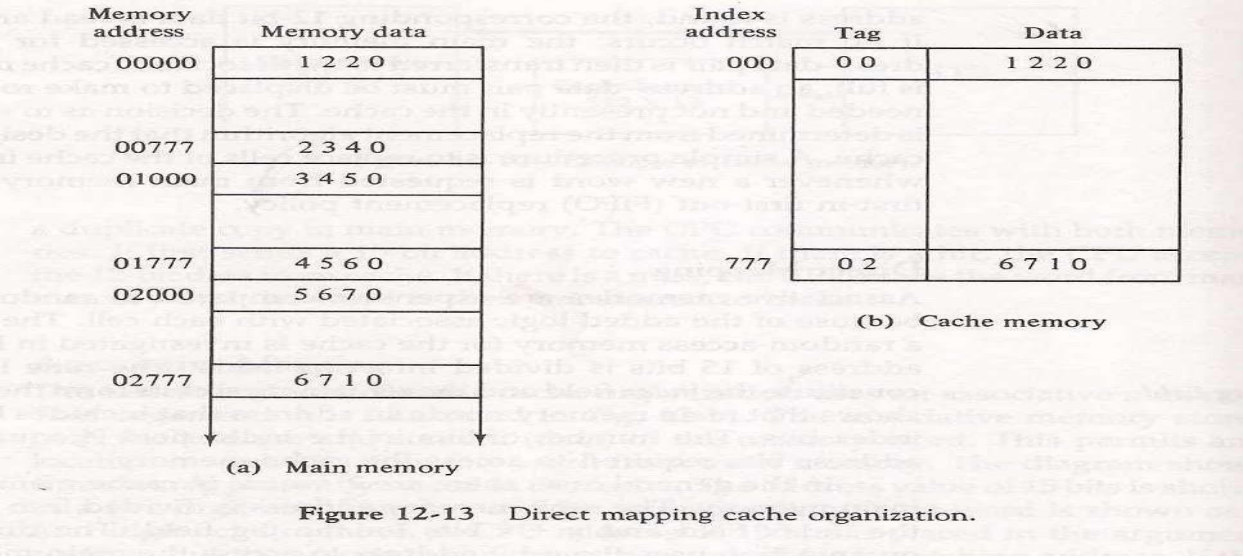


Figure 12-13 Direct mapping cache organization.

The direct mapping cache organization uses the n- bit address to access the main memory and the k-bit index to access the cache.Each word in cache consists of the data word and associated tag. When a new word is first brought into the cache, the tag bits are stored alongside the data bits.When the CPU generates a memory request, the index field is used the index field is used for the address to access the cache. The tag field of the CPU address is compared with the tag in the word read from the cache. If the two tags match, there is a hit anfd the desired data word is in cache. If there is no match, there is a miss and the required word is read from main memory.

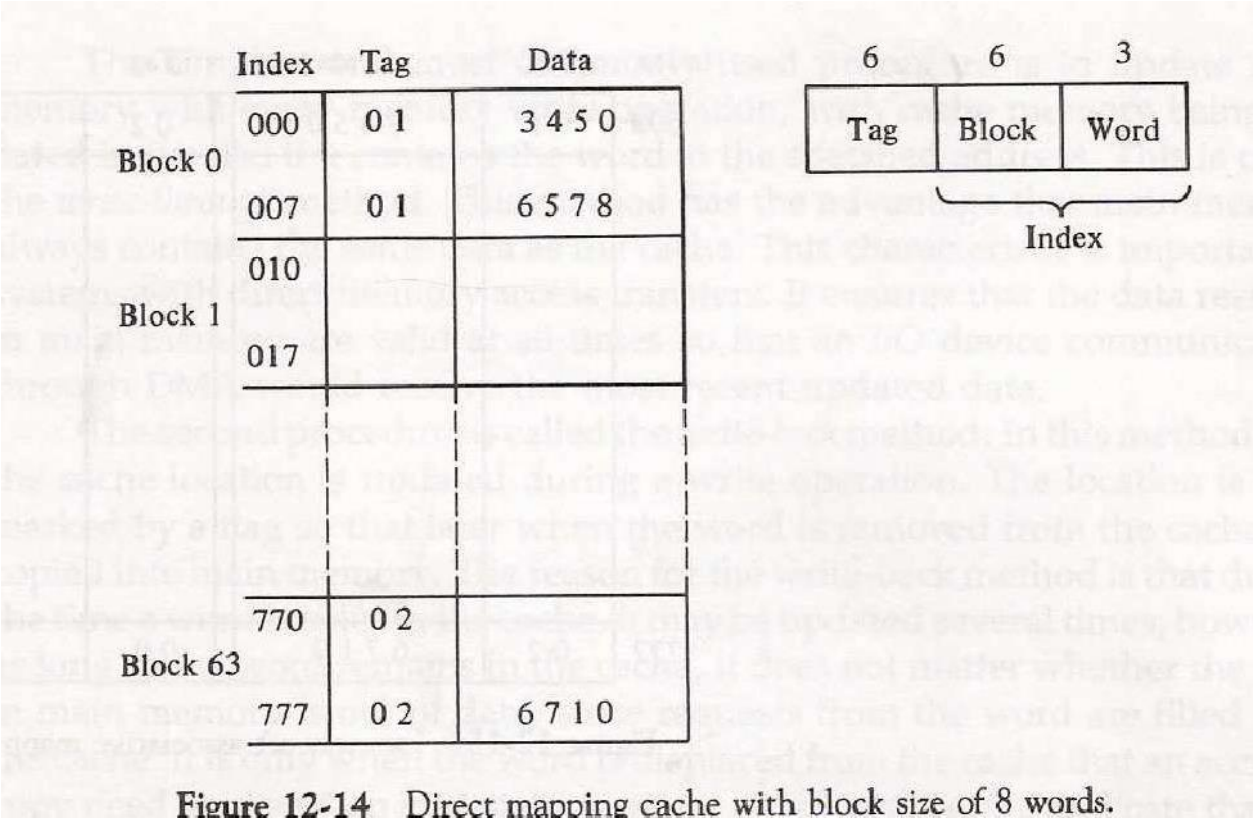
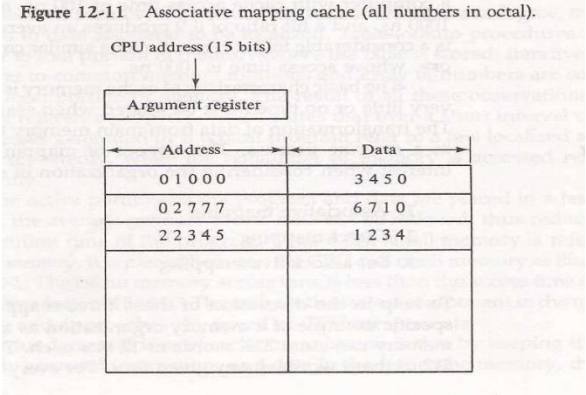


Figure 12-14 Direct mapping cache with block size of 8 words.

In fig 12-14, The index field is now divided into two parts: Block field and The word field. In a 512 word cache there are 64 blocks of 8 words each, since 64X8=512. The block number is specified with a 6 bit field and the word with in the block is specified with a 3-bit field. Th etag field stored within the the cache is common to all eight words of the same block.

Associative mapping:

In this mapping function, any block of Main memory can potentially reside in any cache block position. This is much more flexible mapping method.



In fig 12-11, The associative memory stores both address and content(data) of the memory word. This permits any location in cache to store any word from main memory.The diagram shows three words presently stored in the cache. The address value of 15 bits is shown as a five-digit ctal number and its corressponding 12-bit word is shown as a four-digit octal number. A CPU address of 15-bits is placed in the argument register and the associative memory is searched for a matching address. If address is found, the corresponding 12-bit data is read and sent to the CPU. If no match occurs, the main memory is accessed for the word.

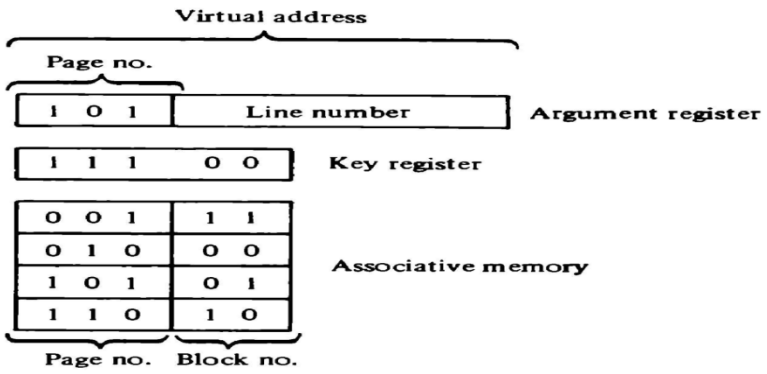
Set-associative mapping:

In this method, blocks of cache are grouped into sets, and the mapping allows a block of main memory to reside in any block of a specific set. From the flexibility point of view, it is in between to the other two methods.

Associative Memory Page Table

A random-access memory page table is inefficient with respect to storage utilization.

Figure 12-20 An associative memory page table.



Replace the random access memory-page table with an associative memory of four words as shown in Fig12-20. Each entry in the associative memory array consists of two fields. The first three bits specify a field for storing the page number. The last two bits constitute a field for storing the block number. The virtual address is placed in the argument register.

Address Translation

- A table is needed to map virtual address to a physical address (dynamic operation) This table may be kept in
    - a separate memory or
    - main memory or
    - associative memory

MEMORY MANAGEMENT HARDWARE

A memory management system is a collection of hardware and software procedure for managing the various programs residing in memory. The memory management software is part of an overall operating system available in many computers.

The basic components of a memory management unit are:

1. A facility for dynamic storage relocation that maps logical memory references into physical memory addresses
2. A provision for sharing common programs stored in memory by different users
3. Protection of information against unauthorized access between users and preventing users from changing operating system functions

A segment is a set of logically related instructions or data elements associated with a given name. Segment may be generated by the programmer or by the operating System. The address generated by segmented program is called a logical address. The logical address may be larger than the physical memory address as in virtual memory, but it may also be equal, and sometimes even smaller than the length of the physical memory address.

Segmented –Page Mapping

The property of logical space is that it uses variable-length segments. The length of each segment is allowed to grow and contract according to the needs of the program being executed.

Page Replacement Algorithms

Paging is a storage mechanism. Paging is used to retrieve processes from secondary memory to primary memory.

The main memory is divided into small blocks called pages. Now, each of the pages contains the process which is retrieved into main memory and it is stored in one frame of memory.

It is very important to have pages and frames which are of equal sizes which are very useful for mapping and complete utilization of memory.

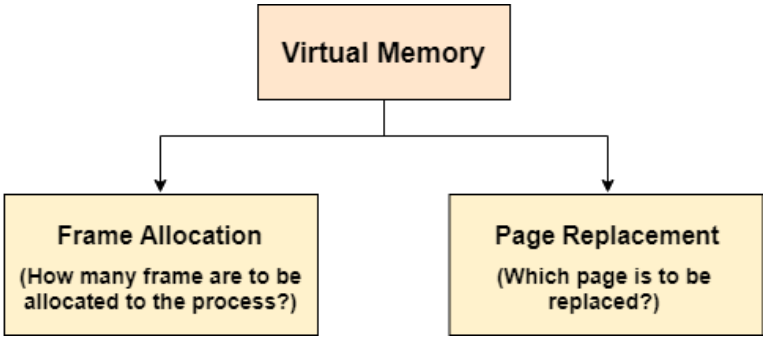
Virtual Memory

A storage method known as virtual memory gives the user the impression that their main memory is quite large. By considering a portion of secondary memory as the main memory, this is accomplished. By giving the user the impression that there is memory available to load the process, this approach allows them to load larger size programs than the primary memory that is accessible. The Operating System loads the many components of several processes in the main memory as opposed to loading a single large process there. By doing this, the level of multiprogramming will be enhanced, which will increase CPU consumption.

Demand Paging

The Demand Paging is a condition which is occurred in the Virtual Memory. We know that the pages of the process are stored in secondary memory. The page is brought to the main memory when required. We do not know when this requirement is going to occur. So, the pages are brought to the main memory when required by the Page Replacement Algorithms.

So, the process of calling the pages to main memory to secondary memory upon demand is known as Demand Paging.



The important jobs of virtual memory in Operating Systems are two. They are:

- Frame Allocation
- Page Replacement.

Frame Allocation in Virtual Memory
Demand paging is used to implement virtual memory, an essential component of operating systems. A page-replacement mechanism and a frame allocation algorithm must be created for demand paging. If you have numerous processes, frame allocation techniques are utilized to determine how many frames to provide to each process.

A Physical Address is required by the Central Processing Unit (CPU) for the frame creation and the physical Addressing provides the actual address to the frame created. For each page a frame must be created.

Frame Allocation Constraints

- The Frames that can be allocated cannot be greater than total number of frames.
- Each process should be given a set minimum amount of frames.
- When fewer frames are allocated then the page fault ration increases and the process execution becomes less efficient
- There ought to be sufficient frames to accommodate all the many pages that a single instruction may refer to

Frame Allocation Algorithms
There are three types of Frame Allocation Algorithms in Operating Systems. They are:

1) Equal Frame Allocation Algorithms

Here, in this Frame Allocation Algorithm we take number of frames and number of processes at once. We divide the number of frames by number of processes. We get the number of frames we must provide for each process. This means if we have 36 frames and 6 processes. For each process 6 frames are allocated. It is not very logical to assign equal frames to all processes in systems with processes of different sizes. A lot of allocated but unused frames will eventually be wasted if a lot of frames are given to a little operation.

2) Proportionate Frame Allocation Algorithms

Here, in this Frame Allocation Algorithms we take number of frames based on the process size. For big process more number of frames is allocated. For small processes less number of frames is allocated by the operating system. The problem in the Proportionate Frame Allocation Algorithm is number of frames are wasted in some rare cases. The advantage in Proportionate Frame Allocation Algorithm is that instead of equally, each operation divides the available frames according to its demands.

3) Priority Frame Allocation Algorithms

According to the quantity of frame allocations and the processes, priority frame allocation distributes frames. Let's say a process has a high priority and needs more frames; in such case, additional frames will be given to the process. Processes with lower priorities are then later executed in future and first only high priority processes are executed first.

Page Replacement Algorithms

There are three types of Page Replacement Algorithms. They are:

- Optimal Page Replacement Algorithm
- First In First Out Page Replacement Algorithm

- Least Recently Used (LRU) Page Replacement Algorithm

First in First out Page Replacement Algorithm

This is the first basic algorithm of Page Replacement Algorithms. This algorithm is basically dependent on the number of frames used. Then each frame takes up the certain page and tries to access it. When the frames are filled then the actual problem starts. The fixed number of frames is filled up with the help of first frames present. This concept is fulfilled with the help of Demand Paging. After filling up of the frames, the next page in the waiting queue tries to enter the frame. If the frame is present then, no problem is occurred. Because of the page which is to be searched is already present in the allocated frames. If the page to be searched is found among the frames then, this process is known as Page Hit. If the page to be searched is not found among the frames then, this process is known as Page Fault. When Page Fault occurs this problem arises, then the First In First Out Page Replacement Algorithm comes into picture.

The First In First Out (FIFO) Page Replacement Algorithm removes the Page in the frame which is allotted long back. This means the useless page which is in the frame for a longer time is removed and the new page which is in the ready queue and is ready to occupy the frame is allowed by the First In First Out Page Replacement.

Let us understand this First In First Out Page Replacement Algorithm working with the help of an example.

Example:

Consider the reference string 6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 2, 0 for a memory with three frames and calculate number of page faults by using FIFO (First In First Out) Page replacement algorithms.

Reference String:

6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 2, 0

S. no	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
F3				2	2	2	4	4	4	2	2	2	2	2	2	2	2	2	2	0
F2		1	1	1	1	3	3	3	0	0	0	0	0	0	0	3	3	3	3	3
F1	6	6	6	6	0	0	0	6	6	6	1	1	1	1	1	1	1	1	1	1
Hit (H)/ Fault (F)	F	F	H	F	F	F	F	F	F	F	F	H	H	H	H	F	H	H	H	F

Number of Page Hits = 8

Number of Page Faults = 12

The Ratio of Page Hit to the Page Fault = 8 : 12 - - - > 2 : 3 - - - > 0.66

The Page Hit Percentage = 8 *100 / 20 = 40%

The Page Fault Percentage = 100 - Page Hit Percentage = 100 - 40 = 60%

Explanation

First, fill the frames with the initial pages. Then, after the frames are filled we need to create a space in the frames for the new page to occupy. So, with the help of First in First Out Page Replacement Algorithm we remove the frame which contains the page is older among the pages. By removing the older page we give access for the new frame to occupy the empty space created by the First in First out Page Replacement Algorithm.

OPTIMAL Page Replacement Algorithm

This is the second basic algorithm of Page Replacement Algorithms. This algorithm is basically dependent on the number of frames used. Then each frame takes up the certain page and tries to access it. When the frames are filled then the actual problem starts. The fixed number of frames is filled up with the help of first frames present. This concept is fulfilled with the help of Demand Paging

After filling up of the frames, the next page in the waiting queue tries to enter the frame. If the frame is present then, no problem is occurred. Because of the page which is to be searched is already present in the allocated frames.

If the page to be searched is found among the frames then, this process is known as Page Hit.

If the page to be searched is not found among the frames then, this process is known as Page Fault.

When Page Fault occurs this problem arises, then the OPTIMAL Page Replacement Algorithm comes into picture.

The OPTIMAL Page Replacement Algorithms works on a certain principle. The principle is:

Replace the Page which is not used in the Longest Dimension of time in future

This principle means that after all the frames are filled then, see the future pages which are to occupy the frames. Go on checking for the pages which are already available in the frames. Choose the page which is at last.

Example:

Suppose the Reference String is:

0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 2, 0

6, 1, 2 are in the frames occupying the frames.

Now we need to enter 0 into the frame by removing one page from the page

So, let us check which page number occurs last

From the sub sequence 0, 3, 4, 6, 0, 2, 1 we can say that 1 is the last occurring page number. So we can say that 0 can be placed in the frame body by removing 1 from the frame.

Let us understand this OPTIMAL Page Replacement Algorithm working with the help of an example.

Example:

Consider the reference string 6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 4, 0 for a memory with three frames and calculate number of page faults by using OPTIMAL Page replacement algorithms.

Points to Remember

Page Not Found - - - > Page Fault

Page Found - - - > Page Hit

Reference String:

6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 4, 0

S. no	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
F3				2	2	2	4	4	4	4	4	4	4	4	4	3	3	3	4	4
F2		1	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
F1	6	6	6	6	0	0	0	6	6	2	2	2	2	2	2	2	2	2	2	0
Hit (H)/ Faul t (F)	F	F	H	F	F	F	F	H	H	F	F	H	H	H	H	F	H	F	F	F

Number of Page Hits = 8

Number of Page Faults = 12

The Ratio of Page Hit to the Page Fault = 8 : 12 - - - > 2 : 3 - - - > 0.66

The Page Hit Percentage = 8 *100 / 20 = 40%

The Page Fault Percentage = 100 - Page Hit Percentage = 100 - 40 = 60%

Explanation

First, fill the frames with the initial pages. Then, after the frames are filled we need to create a space in the frames for the new page to occupy.

Here, we would fill the empty spaces with the pages we and the empty frames we have. The problem occurs when there is no space for occupying of pages. We have already known that we would replace the Page which is not used in the Longest Dimension of time in future.

There comes a question what if there is absence of page which is in the frame.

Suppose the Reference String is:

0, 2, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 2, 0

6, 1, 5 are in the frames occupying the frames.

Here, we can see that page number 5 is not present in the Reference String. But the number 5 is present in the Frame. So, as the page number 5 is absent we remove it when required and other page can occupy that position.

Least Recently Used (LRU) Replacement Algorithm

This is the last basic algorithm of Page Replacement Algorithms. This algorithm is basically dependent on the number of frames used. Then each frame takes up the certain page and tries to access it. When the frames are filled then the actual problem starts. The fixed number of frames is filled up with the help of first frames present. This concept is fulfilled with the help of Demand Paging

After filling up of the frames, the next page in the waiting queue tries to enter the frame. If the frame is present then, no problem is occurred. Because of the page which is to be searched is already present in the allocated frames.

If the page to be searched is found among the frames then, this process is known as Page Hit.

If the page to be searched is not found among the frames then, this process is known as Page Fault.

When Page Fault occurs this problem arises, then the Least Recently Used (LRU) Page Replacement Algorithm comes into picture.

The Least Recently Used (LRU) Page Replacement Algorithms works on a certain principle. The principle is:

Replace the page with the page which is less dimension of time recently used page in the past.

Example:

Suppose the Reference String is:

6, 1, 1, 2, 0, 3, 4, 6, 0

The pages with page numbers 6, 1, 2 are in the frames occupying the frames.

Now, we need to allot a space for the page numbered 0.

Now, we need to travel back into the past to check which page can be replaced.

6 is the oldest page which is available in the Frame.

So, replace 6 with the page numbered 0.

Let us understand this Least Recently Used (LRU) Page Replacement Algorithm working with the help of an example.

Example:

Consider the reference string 6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 2, 0 for a memory with three frames and calculate number of page faults by using Least Recently Used (LRU) Page replacement algorithms.

Points to Remember

Page Not Found - - - > Page Fault

Page Found - - - > Page Hit

Reference String:

6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 2, 0

S. no	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
F3				2	2	2	4	4	4	2	2	2	2	2	2	2	2	2	2	2
F2		1	1	1	1	3	3	3	0	0	0	0	0	0	0	0	0	1	1	1
F1	6	6	6	6	0	0	0	6	6	6	1	1	1	1	1	3	3	3	3	0
Hit (H)/ Fault (F)	F	F	H	F	F	F	F	F	F	F	F	H	H	H	H	F	H	F	H	F

Number of Page Hits = 7

Number of Page Faults = 13

The Ratio of Page Hit to the Page Fault = 7 : 12 - - - > 0.5833 : 1

The Page Hit Percentage = 7 * 100 / 20 = 35%

The Page Fault Percentage = 100 - Page Hit Percentage = 100 - 35 = 65%

Explanation

First, fill the frames with the initial pages. Then, after the frames are filled we need to create a space in the frames for the new page to occupy.

Here, we would fill the empty spaces with the pages we and the empty frames we have. The problem occurs when there is no space for occupying of pages. We have already known that we would replace the Page which is not used in the Longest Dimension of time in past or can be said as the Page which is very far away in the past.

Pipelining

- Pipelining is a technique of *decomposing a sequential process into suboperations,* with each subprocess being

executed in a special dedicated segment that operates *concurrently* with all other segments.

- The overlapping of computation is made possible by associating a *register* with each segment in the pipeline.
- The registers provide isolation between each segment so that each can operate on distinct data *simultaneously*.
- Perhaps the simplest way of viewing the pipeline structure is to imagine that each segment consists of an *input register* followed by a *combinational circuit*.
  - The register holds the data.
  - The combinational circuit performs the suboperation in the particular segment.
- A clock is applied to all registers after *enough time* has elapsed to perform all segment activity.
- The pipeline organization will be demonstrated by means of a simple example.
- To perform the combined multiply and add operations with a stream of numbers

  $A_i * B_i + C_i$ for $i = 1, 2, 3, \ldots, 7$.
- Each suboperation is to be implemented in a segment within a pipeline.

  R1 □ A_i, R2 □ B_i Input A_i and B_i

  R3 □ R1 * R2, R4 □ C_i Multiply and input

  C_iR5 □ R3 + R4 Add C_i to product.
- Each segment has one or two registers and a combinational circuit as shown inFig. 9-2.
- The five registers are loaded with new data every clock pulse. The effect ofeach clock is shown in Table 4-1.

- The five registers are loaded with new data every clock pulse. The effect ofeach clock is shown in Table 4-1.

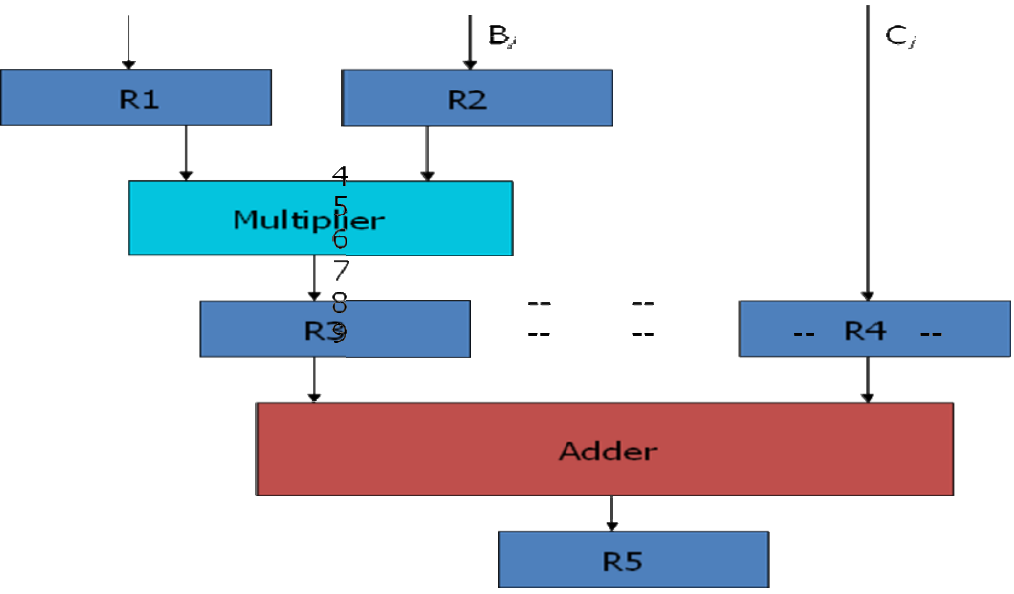


Fig 4-1: Example of pipeline processing

Clock Pulse Number	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A_1	B_1	--	--	--
2	A_2	B_2	$A_1 * B_1$	C_1	--
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
	A_7	B_7	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
			$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
					$A_7 * B_7 + C_7$

Table 4-1: Content of Registers in Pipeline Example

General Considerations

- Any operation that can be decomposed into a sequence of suboperations of about the *same complexity* can be implemented by a pipeline processor.
- The general structure of a four-segment pipeline is illustrated in Fig. 4-2.
- We define a *task* as the total operation performed going through all the segments in the pipeline.
- The behavior of a pipeline can be illustrated with a *space-time* diagram.
  - It shows the segment utilization as a function of time.

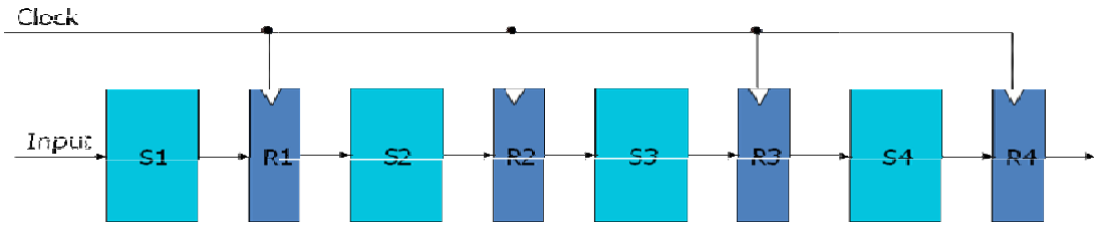


Fig 4-2: Four Segment Pipeline

- The space-time diagram of a four-segment pipeline is demonstrated in Fig. 4-3.
- Where a *k*-segment pipeline with a clock cycle time t_p is used to execute *n* tasks.
  - The first task T_1 requires a time equal to kt_p to complete its operation.
  - The remaining *n*-1 tasks will be completed after a time equal to $(n\text{-}1)t_p$
  - Therefore, to complete *n* tasks using a *k*-segment pipeline requires *k*+(*n*-1)clock cycles.
- Consider a non pipeline unit that performs the same operation and takes a timeequal to t_n to complete each task.
  - The total time required for *n* tasks is nt_n.

Fig 4-3: Space-time diagram for pipeline

- The *speedup of a pipeline processing* over an equivalent non-pipeline processingis defined by the ratio
  $$S = nt_n/(k+n\text{-}1)t_p .$$
- If *n* becomes much larger than *k*-1, the speedup becomes $S = t_n/t_p$.
- If we assume that the time it takes to process a task is the same in the pipeline and non-pipeline circuits, i.e., $t_n = kt_p$, the speedup reduces to $S=kt_p/t_p=k$.
- This shows that the theoretical maximum speed up that a pipeline can provide is *k*, where *k* is the number of segments in the pipeline.

- To duplicate the theoretical speed advantage of a pipeline process by means of multiple functional units, it is necessary to construct *k* identical units that will be operating in parallel.
- This is illustrated in Fig. 4-4, where four identical circuits are connected in parallel.
- Instead of operating with the *input data in sequence* as in a pipeline, the parallel circuits accept four input data items *simultaneously* and perform four tasks at the same time.

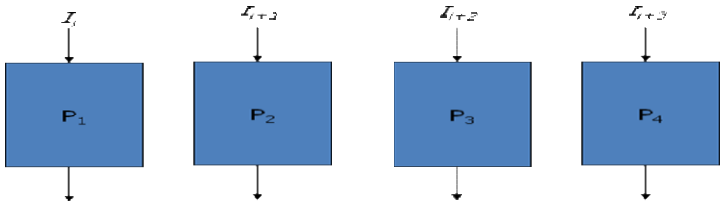


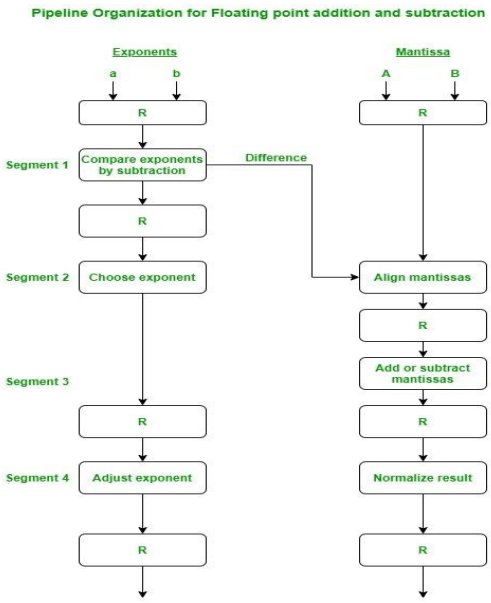
Fig 4-4: Multiple functional units in parallel

- There are various reasons why the pipeline cannot operate at its maximum theoretical rate.
- Different segments may take different times to complete their suboperation.
- It is not always correct to assume that a non pipe circuit has the same timedelay as that of an equivalent pipeline circuit.
- There are two areas of computer design where the pipeline organization is applicable.
- Arithmetic pipeline
- Instruction pipeline

Arithmetic Pipeline and Instruction Pipeline

1. Arithmetic Pipeline :

An arithmetic pipeline divides an arithmetic problem into various sub problems for execution in various pipeline segments. It is used for floating point operations, multiplication and various other computations. The process or flowchart arithmetic pipeline for floating point addition is shown in the diagram.



Floating point addition using arithmetic pipeline :

The following sub operations are performed in this case:
. Compare the exponents.
. Align the mantissas.
. Add or subtract the mantissas.
. Normalise the result

First of all the two exponents are compared and the larger of two exponents is chosen as the result exponent. The difference in the exponents then decides how many times we must shift the smaller exponent to the right. Then after shifting of exponent, both the mantissas get aligned. Finally the addition of both numbers take place followed by normalisation of the result in the last segment.

Example:
Let us consider two numbers,
X=0.3214*10^3 and Y=0.4500*10^2

Explanation:
First of all the two exponents are subtracted to give 3-2=1. Thus 3 becomes the exponent of result and the smaller exponent is shifted 1 times to the right to give

Y=0.0450*10^3
Finally the two numbers are added to produce

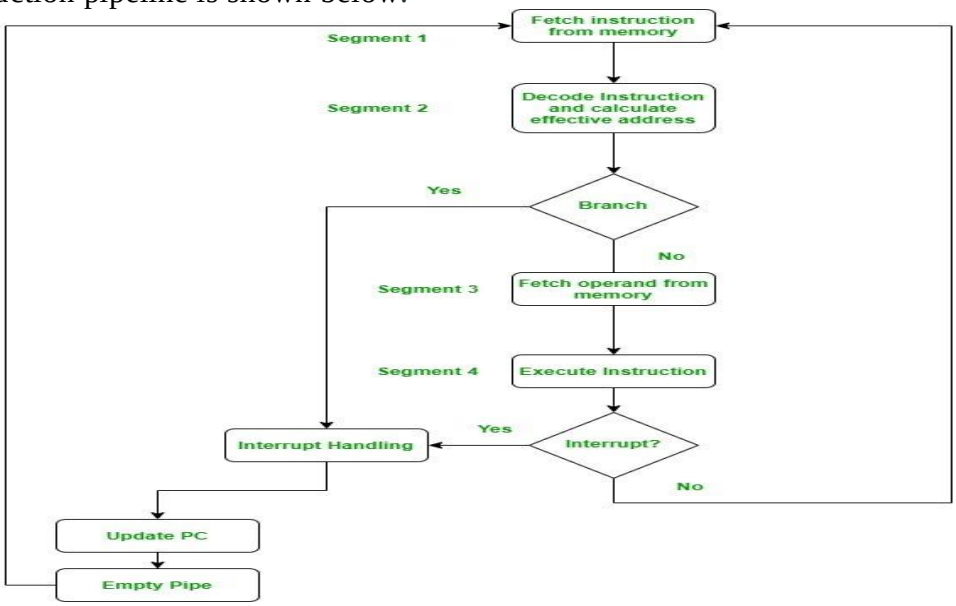
Z=0.3664*10^3
As the result is already normalized the result remains the same.

2. Instruction Pipeline :

In this a stream of instructions can be executed by overlapping fetch, decode and execute phases of an instruction cycle. This type of technique is used to increase the throughput of the computer system. An instruction pipeline reads instruction from the memory while previous instructions are being executed in other segments of the pipeline. Thus we can execute multiple instructions simultaneously. The pipeline will be more efficient if the instruction cycle is divided into segments of equal duration. In the most general case computer needs to process each instruction in following sequence of steps:

. Fetch the instruction from memory (FI)
. Decode the instruction (DA)
. Calculate the effective address
. Fetch the operands from memory (FO)
. Execute the instruction (EX)
. Store the result in the proper place

The flowchart for instruction pipeline is shown below.



Let us see an example of instruction pipeline.

Example:

	Stage	1	2	3	4	5	6	7	8	9	10	11	12	13
	1	FI	DA	FO	EX									
Instruction	2		FI	DA	FO	EX								
Branch	3			FI	DA	FO	EX							
	4				FI	---	---	FI	DA	FO	EX			
	5								FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

Here the instruction is fetched on first clock cycle in segment 1. Now it is decoded in next clock cycle, then operands are fetched and finally the instruction is executed. We can see that here the fetch and decode phase overlap due to pipelining. By the time the first instruction is being decoded, next instruction is fetched by the pipeline. In case of third instruction we see that it is a branched instruction. Here when it is being decoded 4th instruction is fetched simultaneously. But as it is a branched instruction it may point to some other instruction when it is decoded. Thus fourth instruction is kept on hold until the branched instruction is executed. When it gets executed then the fourth instruction is copied back and the other phases continue as usual.

RISC Pipeline

- To use an efficient instruction pipeline
- To implement an instruction pipeline using a small number ofsuboperations, with each being executed in one clock cycle.
    - Because of the fixed-length instruction format, the decoding of the operation can occur at the same time as the register selection.
    - Therefore, the instruction pipeline can be implemented with two orthree segments.
        - One segment fetches the instruction from program memory
        - The other segment executes the instruction in the ALU
        - Third segment may be used to store the result of the ALUoperation in a destination register
- The data transfer instructions in RISC are limited to load and store instructions.
    - These instructions use register indirect addressing. They usually need three or four stages in the pipeline.
    - To prevent conflicts between a memory access to fetch an instruction and to load or store an operand, most RISC machines use two separate buses with two memories.
    - Cache memory: operate at the same speed as the CPU clock
- One of the major advantages of RISC is its ability to execute instructions at the rate of one per clock cycle.
    - In effect, it is to start each instruction with each clock cycle and to pipeline the processor to achieve the goal of single-cycle instruction

execution.
    - RISC can achieve pipeline segments, requiring just one clock cycle.
- *Compiler* supported that translates the high-level language program into machine language program.
    - Instead of designing hardware to handle the difficulties associated with data conflicts and branch penalties.
    - RISC processors rely on the efficiency of the compiler to detect and minimize the delays encountered with these problems.

Example: Three-Segment Instruction Pipeline

- Three are three types of instructions:

o The data manipulation instructions: operate on data in processor registers.
o The data transfer instructions:
o The program control instructions:

- The *control section* fetches the instruction from program memory into aninstruction register.
    - The instruction is decoded at the same time that the registers needed forthe execution of the instruction are selected.
- The processor unit consists of a number of registers and an arithmetic logic unit(ALU).
- A data memory is used to load or store the data from a selected register in theregister file.
- The instruction cycle can be divided into three suboperations and implemented inthree segments:
    - I: Instruction fetch
        - Fetches the instruction from program memory
    - A: ALU operation
        - The instruction is decoded and an ALU operation is performed.
        - It performs an operation for a data manipulation instruction.
        - It evaluates the effective address for a load or store instruction.
        - It calculates the branch address for a program control instruction.
    - E: Execute instruction
        - Directs the output of the ALU to one of three destinations,depending on the decoded instruction.
        - It transfers the result of the ALU operation into a destinationregister in the register file.
        - It transfers the effective address to a data memory for loading orstoring.
        - It transfers the branch address to the program counter.

Delayed Load

- Consider the operation of the following four instructions:
    - LOAD: R1 □ M[address 1]
    - LOAD: R2 □ M[address 2]
    - ADD: R3 □ R1 +R2

- STORE: M[address 3] □ R3

- There will be a *data conflict* in instruction 3 because the operand in R2 is not yetavailable in the A segment.
- This can be seen from the timing of the pipeline shown in Fig. 4-9(a).
  - The E segment in clock cycle 4 is in a process of placing the memory datainto R2.
  - The A segment in clock cycle 4 is using the data from R2.
- It is up to the *compiler* to make sure that the instruction following the loadinstruction uses the data fetched from memory.
- This concept of delaying the use of the data loaded from memory is referred to as *delayed load*.

Clock cycles:	1	2	3	4	5	6
1. Load R1	I	A	E			
2. Load R2		I	A	E		
3. Add R1+R2			I	A	E	
4. Store R3				I	A	E

Fig 4-9(a): Three segment pipeline timing - Pipeline timing with data conflict

- Fig. 4-9(b) shows the same program with a no-op instruction inserted after theload to R2 instruction.

Clock cycle:	1	2	3	4	5	6	7
1. Load R1	I	A	E				
2. Load R2		I	A	E			
3. No-operation			I	A	E		
4. Add R1+R2				I	A	E	
5. Store R3					I	A	E

Fig 4-9(b): Three segment pipeline timing - Pipeline timing with delayed load

- Thus the *no-op instruction* is used to advance one clock cycle in order to compensate for the *data conflict* in the pipeline.
- The advantage of the delayed load approach is that the data dependency is takencare of by the *compiler rather than the hardware.*

Delayed Branch

- The method used in most RISC processors is to rely on the *compiler to redefine the branches* so that they take effect at the proper time in the pipeline. This

method is referred to as *delayed branch*.

- The compiler is designed to analyze the instructions *before and after the branch* and *rearrange the program sequence* by inserting useful instructions in the delay steps.

An Example of Delayed Branch

- The program for this example consists of five instructions.
  - Load from memory to R1
  - Increment R2
  - Add R3 to R4
  - Subtract R5 from R6
  - Branch to address X
- In Fig. 4-10(a) the compiler inserts *two no-op instructions* after the branch.
  - The branch address X is transferred to PC in clock cycle 7.

Clock cycles:	1	2	3	4	5	6	7	8	9	10
1. Load	I	A	E							
2. Increment		I	A	E						
3. Add			I	A	E					
4. Subtract				I	A	E				
5. Branch to X					I	A	E			
6. No-operation						I	A	E		
7. No-operation							I	A	E	
8. Instruction in X								I	A	E

Fig 4-10(a): Using no operation instruction

- The program in Fig. 4-10(b) is rearranged by placing the add and subtractinstructions *after the branch instruction*.
  - PC is updated to the value of X in clock cycle 5.

Clock cycles:	1	2	3	4	5	6	7	8
1. Load	I	A	E					
2. Increment		I	A	E				
3. Branch to X			I	A	E			
4. Add				I	A	E		
5. Subtract					I	A	E	
6. Instruction in X						I	A	E

Fig 4-10(b): Rearranging the instructions