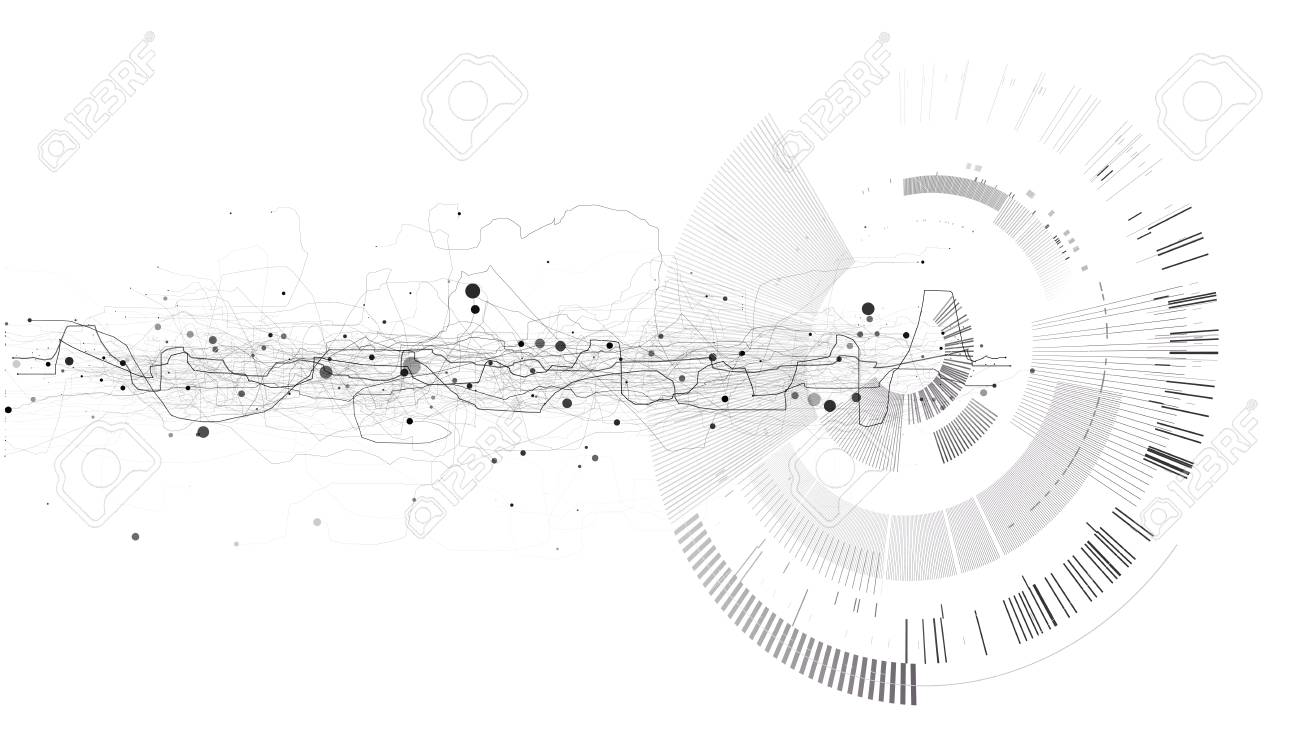
February 17, 2020



**computer organization & architecture**

Lab Report no. 3

Rayan Tahir

17l-4497

EL-B2

**LAB Report no. 3**

**Name:** Rayan Tahir Khan

**Roll no.:** 17L-4497

**Section:** EE-B **Lab:** EL-B2

MEMORY SYSTEM

**Introduction:**

one of the following methods of addressing (also known as byte ordering) is used in most of the machines.

**Big Endian**– the higher byte is stored at lower memory address (i.e. Big Byte first). MIPS, Apple, Sun SPARC are some of the machines in this class.

**Little Endian**- the lower byte is stored at lower memory address (i.e. Little Byte first). Intel’s machines use little endian.

**Objective:**

* To be able to do 32-bit computations using 16-bit registers.
* To learn the x86 memory system segmentation.

**Design/Procedure:**

We had done 3 exercise in this lab

* The first 2 exercises were very simple and just for practice purpose. These were done using calculator and then confirming the answer according to the concept of the memory addressing theory.
* In 3rd exercise we moved data from register to direct memory address and from direct address to register. Data from 1st memory location to 2nd memory location cannot be moved directly. Therefore, registers are used as an intermediate.
* In 4th exercise we used add command to for 32-bit increment using two 16-bit registers. Adding 1 in register A and 0 in register B. Whenever Register A is overflowed, carry becomes 1 and is added to register B.

Conclusion:

In this lab we got familiar with memory structuring on the hardware level.

Application:

* Memory organization plays a key role in defining the performance and computer architecture.
* Understanding the needs of a system is the basis on which memory system is designed for an embedded system.

**POST LAB**

#include<iostream>

using namespace std;

template<typename T>

class Node

{

public:

// constructor

Node(T element)

{

data = element;

next = 0;

prev = 0;

}

//sets the KeyType data in the Node

void setData(T pVal)

{

data = pVal

}

// returns the KeyType data in the Node

T getData()

{

return data;

}

// returns the link to the next node

Node\* GetNext()

{

return next;

}

Node\* GetPrev()

{

return prev;

}

// sets the link to the next node

void SetNext(Node \*x)

{

Next = x;

}

void SetPrev(Node \*x)

{

Prev = x;

}

private:

T data;

Node \*next;

Node \*prev;

};

template<typename T>

class SCList

{ public:

// constructor of the Singly Circular Linked List

DCList()

{

last = 0;

}

/\*Inserts the node pNew after the node pBefore

if the list is empty, it makes pNew the first node of the list\*/

void Insert(Node<T> \*pBefore, Node<T> \*pNew)

{

if (pBefore)

{

pNew->SetNext(pBefore->GetNext());

pNew->SetPrev(pBefore);

pBefore->SetNext(pNew);

if (pNew->GetNext())

pNew->GetNext()->SetPrev(pNew);

if (pBefore == last)

{

last = pNew;

}

}

else

{

if (!last)

{

last = pNew;

pNew->SetNext(pNew);

pNew->SetPrev(pNew);

}

else

{

pNew->SetNext(last->GetNext());

pNew->SetPrev(last);

last->SetNext(pNew);

pNew->GetNext()->SetPrev(pNew);

}

}

}

//Deletes the node pToBeDeleted

void Delete(Node<T> \*pToBeDeleted)

{

pToBeDeleted->GetPrev()->SetNext(pToBeDeleted->GetNext());

pToBeDeleted->GetNext()->SetPrev(pToBeDeleted->GetPrev());

if (pToBeDeleted == last)

last = pToBeDeleted->GetPrev();

delete pToBeDeleted;

}

//prints the contents of the list

void printList()

{

Node<T> \*first = last->GetNext();

Node<T> \*temp = first;

do

{

cout << " " << temp->getData();

temp = temp->GetNext();

} while (temp->GetNext() != first);

}

private:

Node<T> \*last;

};

int main()

{

cout << endl;

int x = 0;

while (x < 1)

{

x++;

}

}