~~Name:~~

Name: Mozeb Ahmed Khan

Roll No: 20F-0161

Section: CS (3A)

Subject: COAL Course

Assignment: 01

Qno1- Why we prefer developing in high level language over low level language and write conversion from HL to LL.

Sol:- We prefer developing in High level language over low level language because:-

1) Program development is faster, as there are fewer instructions to code.

2) Program maintanance is easier, as fewer instuctions are to be coded.

3) Programs are portable and contain less machine dependent details.

4) They can be used with little or no modifications on different machines.

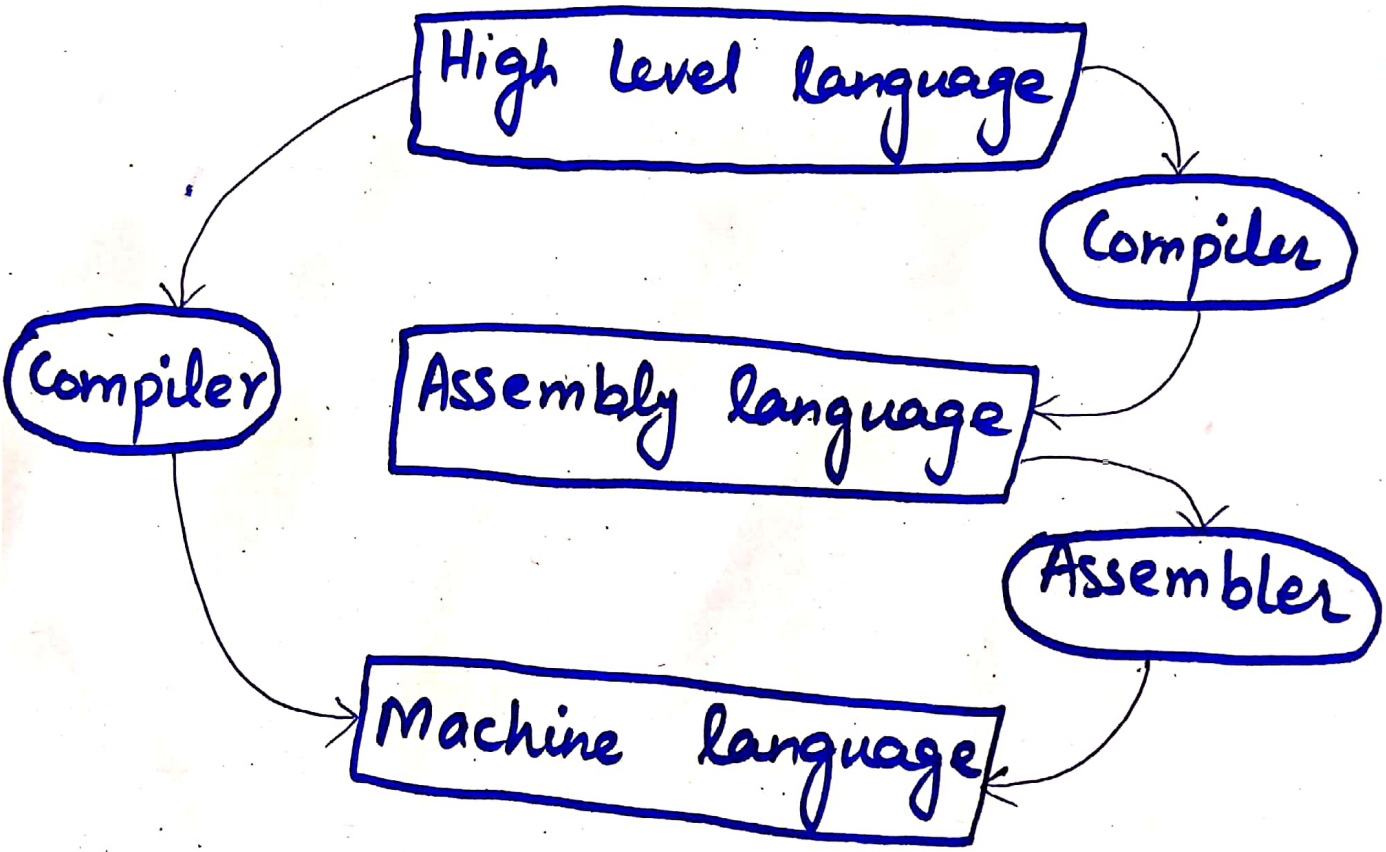
5) HLLs are easier to read, write and understand as they uses natural language.

6) They are independent of a particular type of processor.

7) They are closer to human language.

Conversion from HL to LL:-

* To convert high level language to machine language, a compiler and assembler is used.
* A compiler first converts high level language to assembly language or object code.
* Then an assembler converts assembly language to machine language.



Qno2- Provide comparison b/w HLL & Assembly language with types of Application

Sol :-

High Level Languages	Assembly Languages	Type of Applications
* Formal structures make it easy to organize and maintain large sections of code. * Language may not provide for direct hardware access. Even if it does, awkward coding techniques must often be used, causing maintanance difficulty. * Usually very portable. The source code can be recompiled on each target OS with minimal changes. * Produces too much executable code.	* Minimal formal structure, so one must be imposed by programmers who have varying levels of experience. This causes difficulty in maintaining code. * Hardware access is straightforward & simple. Easy to maintain when programs are short & well-documented. * Must be recorded separately for each plateform, by assembler with different syntax. Difficult to maintain. * Ideal as executable code is small & runs quickly.	* Business application software, written for single platform medium to large size. * Hardware device driver. * Business application written for multiple platforms (different OS). * Embedded system & computer games requiring direct hardware access.

Qno3. How many machine languages are there and briefly describe them.

Sol:- ~~There is only one mac~~

Machine language is a low-level language that is used specifically for each CPU. It is a sequence of 0s and 1s like:- 10100100, 00101100 etc. Assembly language is very similar to machine language. Assembly language is converted into machine language by assembler. There are many assembly languages like ARM, MIPS, and x86.

* **ARM** is a "Reduced Instruction Set Computer (RISC)".
* **x86** is a "Complex Instruction Set Computer (CISC)".
* **MIPS** is a "Reduced Instruction Set Architecture (RISC)".

Qno 5-

a) Find two's compliment of unsigned:-

1) $(26)_{10}$

First, convert it into binary.

2	26	
2	13	0
2	6	1
2	3	0
	1	1

$= (11010)_2$.

Now, first take 1's compliment.

$$\begin{array}{r} 1\ 1\ 0\ 1\ 0 \\ \hline 0\ 0\ 1\ 0\ 1 \end{array}$$ (flipping)

$= (00101)_2$.

Now, add 1 to 1's compliment for 2's compliment.

$$\begin{array}{r} 0\ 0\ 1\ 0\ 1 \\ +\ \ \ \ \ 1 \\ \hline 0\ 0\ 1\ 1\ 0 \end{array}$$

$= (0\ 0\ 1\ 1\ 0)_2$.

2) $(99)_{10}$

First convert it into binary.

2	99	
2	49	1
2	24	1
2	12	0
2	6	0
2	3	0
	1	1

$= (1100011)_2$.

Now, for 1's compliment,

$$\begin{array}{r} 1\ 1\ 0\ 0\ 0\ 1\ 1 \\ \hline 0\ 0\ 1\ 1\ 1\ 0\ 0 \end{array}$$

$= (0011100)_2$

Add 1,

$$\begin{array}{r} 0\ 0\ 1\ 1\ 1\ 0\ 0 \\ +\ 1 \\ \hline 0\ 0\ 1\ 1\ 1\ 0\ 1 \end{array}$$

$= (0011101)_2$.

3) $(181)_{10}$

First convert it into binary.

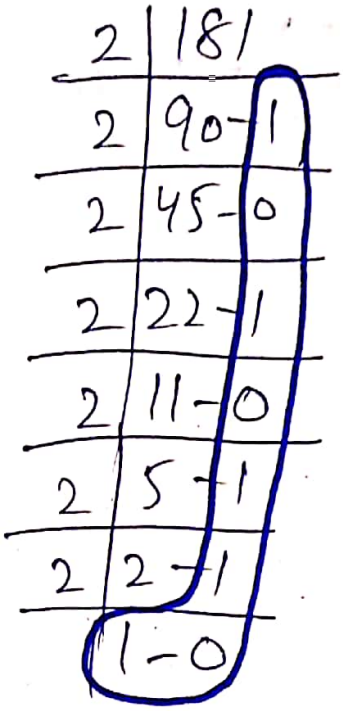
$= (10110101)_2$.

Flip the bits.

$= (01001010)_2$.

Add 1.

$= (01001011)_2$.



Q5-(b) Convert signed numbers:-

1) $(10101001)_2 \rightarrow (?)_{10}$

MSB is '1', so we have to find 2's compliment.

$$\begin{array}{r} 10101001 \\ \hline 01010110 \\ \qquad\qquad +1 \\ \hline 01010111. \end{array}$$

$= (1\times2^6)+(1\times2^4)+(1\times2^2)+(1\times2^1)+(1\times2^0)$

$= 64 + 16 + 4 + 2 + 1$

$= (-87)_{10}$. (As MSB is '1', so number is -ve).

2) $(-85)_{10} \longrightarrow (?)_2$

2	85	
2	42	1
2	21	0
2	10	1
2	5	0
2	2	1
	1	0

Convert into binary.

$= (01010101)_2$.

Now invert, (1's comp).

$= (10101010)_2$.

Now add 1, (2's comp)

$= (10101011)_2$.

3) $(-8124)_{10} \longrightarrow (?)_{16}$

Convert into hex.

16	8124	
16	507	12 = C
16	31	11 = B
16	1	15 = F
	0	1 = 1

$= (1FBC)_{16}$.

```
 F F F F
-1 F B C
---------
 E 0 4 3
     + 1
---------
 E 0 4 4
```

$= (E044)_{16}$.

Qno6- Truth Table of $(\neg X \wedge Y) \vee (Y \wedge \neg Z)$

$(\neg X \wedge Y) \vee (Y \wedge \neg Z)$	$Y \wedge \neg Z$	$\neg X \wedge Y$	$\neg Z$	$\neg X$	Z	Y	X
T	F	T	T	F	F	T	T
T	F	F	T	F	F	T	F
F	F	F	F	F	T	F	T
F	F	F	T	F	T	F	F
T	T	T	F	T	F	T	T
T	F	F	T	T	F	T	F
F	F	F	F	T	T	F	T
F	F	F	T	T	T	F	F

Ans7- Memory Hirarchy:- A memory hirarchy organizes the memory according to the performance of memories, and their time. Memory hirarchy consists of five levels:-

1) Registers 2) Cache memory 3) Main Memory 4) Magnetic Disks 5) Magnetic Tapes.

1) Registers:- Registers are in top of hirarchy. They are ~~storage~~ memory units for storage space. They can hold, control, transfer and perform computation (arithmetic and logical processes) on data or instructions. Registers are volatile and fastest. Registers are located on CPU.

2) Cache Memory:- A cache memory is a static RAM (Random Access Memory). It is located both inside and outside the CPU. Level-1-cache is inside CPU and Level-2-cache is outside CPU. Cache memory is volatile, faster, and expensive. When data to be read is already in cache memory, it is cache hit. When

4) data to be read is not in cache memory, it is cache miss.

3) Main Memory:- It is the primary memory which includes RAM and ROM. RAM is volatile while ROM is permanent. Main memory is very expensive and fastest, but its data is lost when power is switched off. It is larger in size than secondary memory.

4) Magnetic Disks:- It is secondary memory. We can write, read or access data and instructions on magnetic disks. It is covered with magnetic coat and have spots, sectors etc. Examples of magnetic disks are floppy disks, hard disks, zip disks etc.

5) Magnetic Tapes:- Magnetic tapes is also secondary memory that stores data in magnetic way just like magnetic disks. Examples of them are audio cassettes, video cassettes, and recorders etc.

Q8- Registers:- Registers are memory units for storage space. They can hold, control, transfer, and perform computation (like logical, arithmetical processes) on data or instructions. Registers are volatile and fastest. They are basically on CPU. Registers are classified as special or general purpose.

1) General purpose registers:- GP registers are used arithmetic and transfer processes. They can access, transfer, and perform arithmetic processes on data. Some GP registers are:- EAX, EBX, ECX, EDX, EBP, ESP, ESI, EDI etc. These 32-bits registers can be divided into upper & lower bits (16-bits) like AX for EAX. AX can further be divided into 8 bits (lower) & ~~8 bits~~ called AL & 8 bits (upper) called AH.

2) Special Registers:- Special registers are the

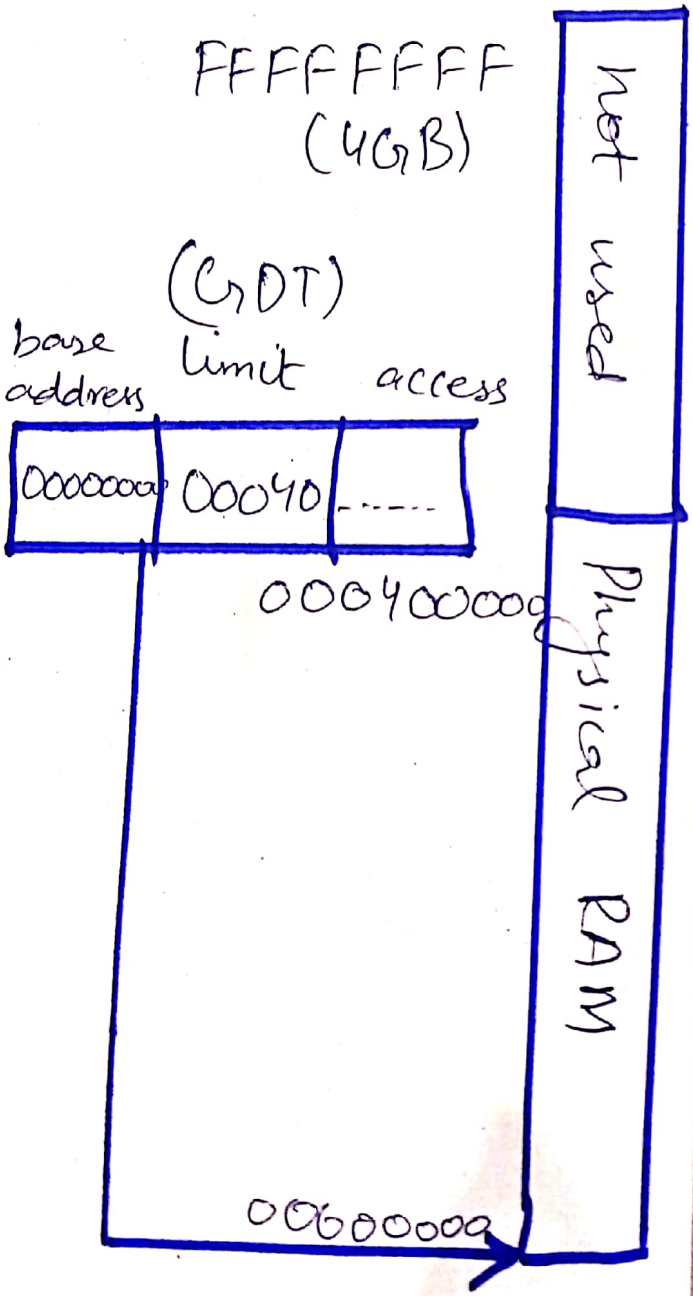
one that controls or handle specific tasks in ~~a~~ CPU. There are a large number of special purpose registers within CPU. They are responsible for the state of program in CPU. Examples of SP registers are :- Program Counter (or Instructions Pointer), Stack Pointer, and Status Registers etc.

Segments:- Segments are memory pointers within CPU, to point out data storage or execution. There are six 16-bit segment registers. These segments point out base address of ~~pre-assignmed~~ pre-assigned memory areas. SS (Stack Segment), DS (Data Segment), CS (Code Segment) are significant. SS hold function variable & parameters. CS holds program instructions. DS holds variables. 3 extra segments are ES, FS, GS.

Q9-

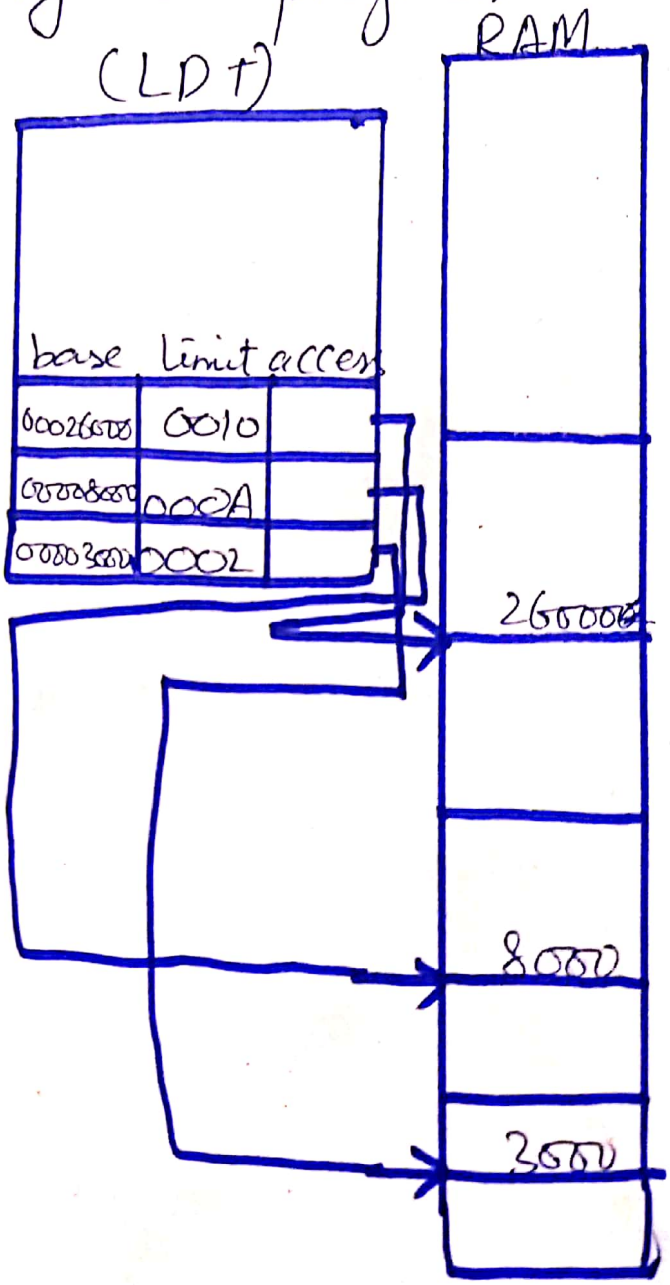
Flat Segment Model

* There is single global discriptor table (GDT).
* All the segments are mapped to entire 32-bits address space.



Multi-Segment Model

* Each program holds a local discriptor table (LDT).
* It holds descriptor for each segment used by the program.



Q9- Data Segment:- In

Q10- Paging :- It is a function of memory management where computer store and retrieve data from main memory to secondary memory. Paging divides linear address spaces into fixed-sized block called 'pages'. For example, IA32 uses pages of sizes 4Kb.

* Pages are allocated space on main memory and if it is full, then on hard disks.

* Complete set of pages mapped by OS is called virtual memory.

* If we want to run many programs in parallel, but main memory cannot load all of them, then paging gives us an illustration that we have plenty of memory as disk storage is cheaper & plentiful. The more of a program depends on main pages, the slower it run. Higher amount of memory means less usage of paging.