FATIMA KHAN
CT-23024

9.5/10

20/Jan/25

Lab # 01

Exploring Instruction Set Architecture x86 Machines

1. Instruction Set Architecture (ISA)

The ISA of a machine is the set of its attributes a system programmer needs to know in order to develop system software or an assembler requires for translation of an Assembly Language (HLL) code into machine language. Examples of such attributes are (but not limited to):

- *Instruction Set*
- *Programmer Accessible Registers:* these are the general purpose registers (GPR) within a processor in contrast to some special purpose registers only accessible to the system hardware and Operating System (OS)
- *Memory-Processor Interaction*
- Addressing Modes: means of specifying operands in an instruction (e.g. immediate mode, direct mode, indirect mode, etc.)
- *Instruction Formats:* breakup of an instruction into various fields (e.g. opcode, specification of source and destination operands, etc.) ISA is also known as the programmer"s view or software model of the machine.

ISA is also known as the programmer"s view or software model of the machine.

2. ISA of x86 Machines

This represents a family of machines beginning with 16-bit 8086/8088 microprocessors. (An n-bit microprocessor is capable of performing n-bit operations). As an evolutionary process, Intel continued to add capabilities and features to this basic ISA. The 80386 was the first 32-bit processor of the family. The ISA of 32-bit processor is regarded as IA-32 (IA for Intel Architecture) or x86-32 by Intel. IA-64 was introduced in Pentium-4F and later processors. Operating Systems are now also categorized on the basis of the architecture they can run on. A 64-bit OS can execute both 64-bit and 32-bit applications. We will limit scope of our discussion to IA-32. The ISA of 8086/88 processor shown in Figure 1-1

Scanned with OKEN Scanner

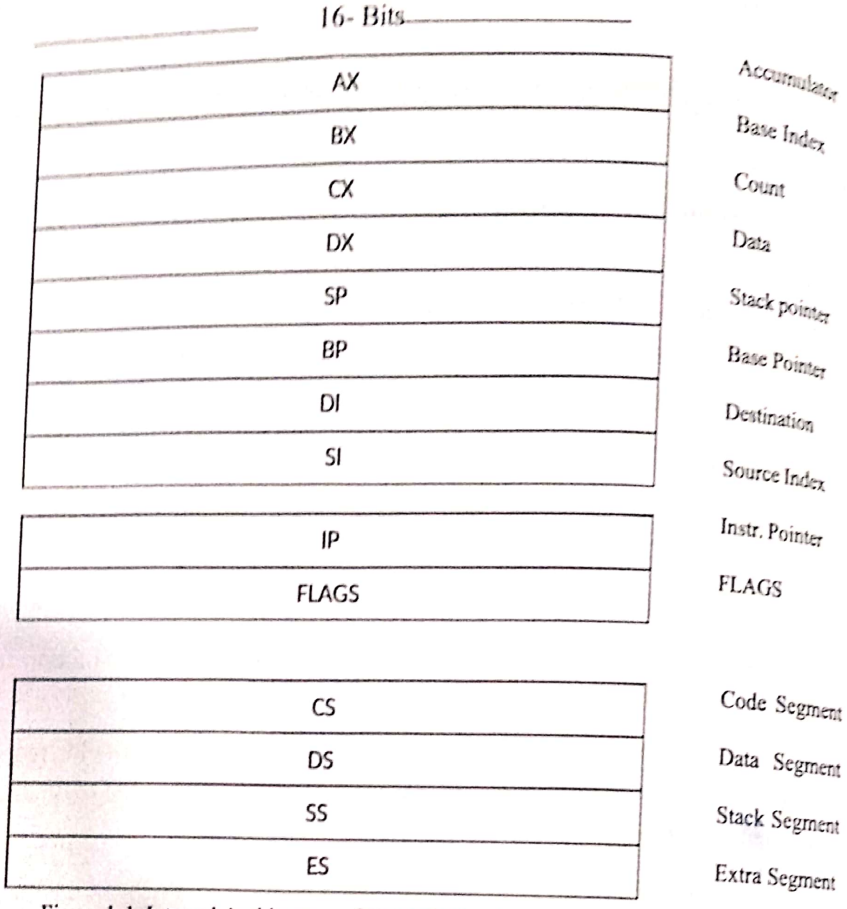


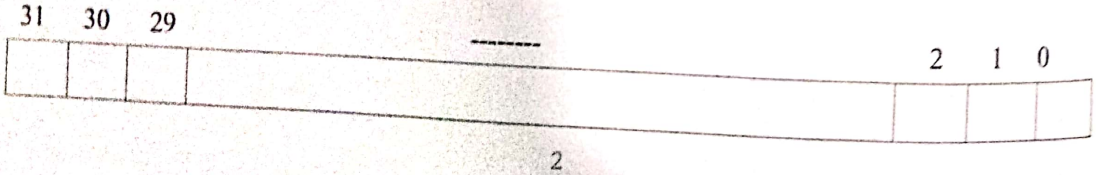
Figure 1-1 Internal Architecture of 8088/8086 microprocessor

2.1 Registers

Registers are temporary storage locations inside the processor. A register can be accessed more quickly than a memory location. Different registers serve different purposes. Some of them are described below:

2.1.1 General-Purpose Registers

EAX, EBX, ECX and EDX are called data or general purpose registers. (E is for extended as they are 32-bit extensions of their 16-bit counter parts AX, BX, CX and DX in 16-bit ISA). Other General purpose registers includes BP, DI and SI registers . Bits in a register are conventionally numbered from right to left, beginning with 0 as shown below.



Apart from accessing the register as a whole, EAX, EBX, ECX, EDX registers can be accessed in pieces as illustrated in Figure 1-2

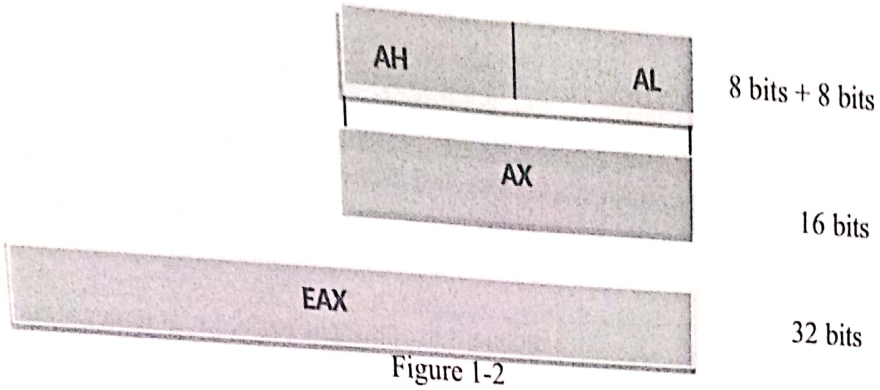


Figure 1-2

Register	Purpose
AX	For 16/32 bits operations, AX registers stores operands for arithmetic operations.
BX	Used to hold starting location of a memory region within data segment.
CX	It defined as a counter, primarily used in loop instructions.
DX	It is used to hold the part of result from multiplication or a part of dividend before division.

2.1.2 Index Registers

ESI(Extended Source Index) and EDI(Extended Destination Index) registers are respectively used as source and destination addresses in string operations. They can also be used to implement array indices.

2.1.3 Pointer Registers

The EIP (Extended Instruction Pointer) register contains the offset in the current code segment for the next instruction to be executed. (Segments will be explained shortly). ESP(Extended Stack Pointer) and EBP(Extended Base Pointer) are used to manipulate stack - a memory area reserved for holding parameters and return address for procedure calls. ESP holds address of top of stack, location where the last data item was pushed. EBP is used in procedure calls to hold address of a reference point in the stack.

Scanned with OKEN Scanner

2.1.4 Flags Register

EFLAGS register is never accessed as a whole. Rather, individual bits of this register either control the CPU operation (control flags) or reflect the outcome of a CPU operation (status flag). Table below gives some of the commonly used control and status flags.

BIT	Name	Type	Description
0	Carry Flag (CF)	Status	Contains carry from a high-order (leftmost) bit following an arithmetic operation; also, contains the contents of the last bit of a shift or rotate operation.
2	Parity Flag (PF)	Status	It indicates the parity of result i.e. if lower order 8 bits of the result contains eve number of 1"s the parity flag is set to 1.0 otherwise.
4	Auxiliary Carry (AF)	Status	If an operation performed in ALU generates a carry/borrow from lower nibble (D_0-D_3) to upper nibble (D_4-D_7), the AF is set to1, 0 otherwise.
6	Zero Flag (ZF)	Status	Indicates the result of an arithmetic or comparison operation (0 = nonzero and 1 = zero result)
7	Sign Flag (SF)	Status	Contains the resulting sign of an arithmetic operation (0 = positive and 1 = negative).
8	Trap Flag (TF)	Control	Permits operation of the processor in single-step mode.
9	Interrupt Flag (IF)	Control	Indicates that all external interrupts, such as keyboard entry, are to be processed or ignored.
10	Direction Flag (DF)	Control	Determines left or right direction for moving or comparing string (character) data.
11	Overflow Flag (OF)	Status	Indicates overflow resulting from some arithmetic operation

Table below gives the clear understanding that how these registers can be view and use in programming.

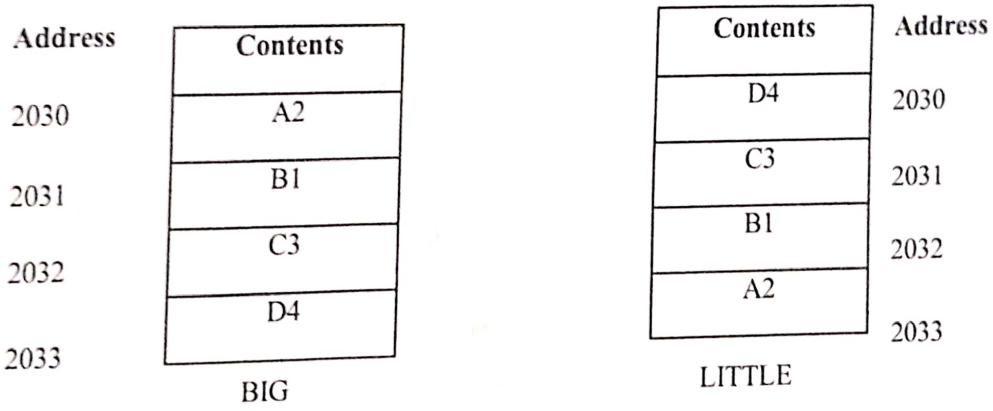
SYMBOL	FLAGS	CLEAR (0)	SET (1)
O	Overflow Flag	NV	OV
D	Direction Flag	UP	DN
I	Interrupt Flag	DI	EI
S	Sign Flag	PL	NG
Z	Zero Flag	NZ	ZR
A	Auxiliary Flag	NA	AC
P	Parity Flag	PO	PE
C	Carry Flag	NC	CY

Scanned with OKEN Scanner

2.2 Memory Addressing

A 32-bit processor uses 32-bit addresses and thus can access 232B = 4GB physical memory. Depending on the machine, a processor can access one or more bytes from memory at a time. The number of bytes accessed simultaneously from main memory is called word length of machine. Generally, all machines are byte-addressable i.e.; every byte stored in memory has a unique address. However, word length of a machine is typically some integral multiple of a byte. Therefore, the address of a word must be the address of one of its constituting bytes. In this regard, one of the following methods of addressing (also known as byte ordering) may be used. 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.

Consider for example, storing 0xA2B1C3D4 in main memory. The two byte orderings are illustrated below.



2.3 Memory Models

IA-32 can use one of the three basic memory models:

Flat Memory Model – memory appears to a program as a single, contiguous address space of 4GB. Code, data, and stack are all contained in this address space, also called the linear address space

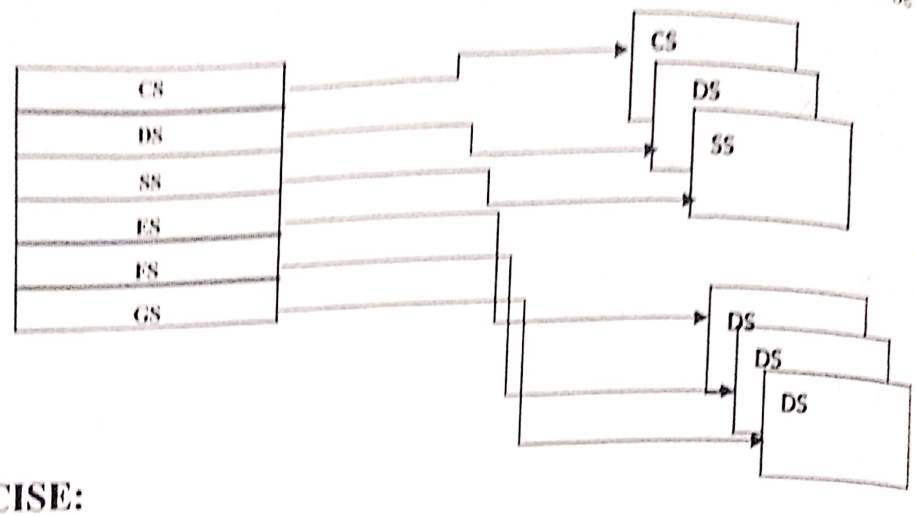
Segmented Memory Model – memory appears to a program as a group of independent memory segments, where code, data, and stack are contained in separate memory segments. To address memory in this model, the processor must use segment registers and an offset to derive the linear address. The primary reason for having segmented memory is to increase the system's reliability by means of protecting one segment from other.

2.4 Segment Registers

The segment registers hold the segment selectors which are special pointers that point to start of individual segments in memory. The use of segment registers is dependent on the memory management model in use.

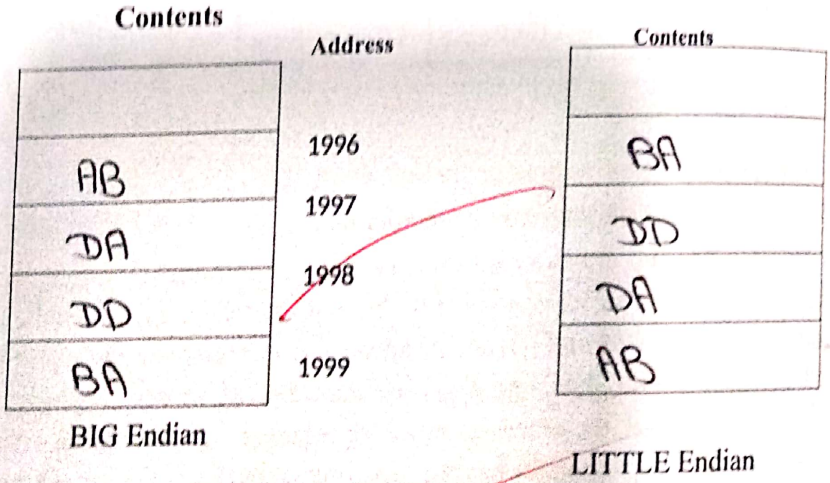
Scanned with OKEN Scanner

The segment registers (CS, DS, SS, ES, FS, and GS) hold 16-bit segment selectors. Each of the segment registers is associated with one of three types of storage: code, data, or stack. For example, the CS register contains the segment selector for the code segment, where the instructions being executed are stored. The processor fetches instructions from the code segment, using a logical address that consists of the segment selector in the CS register and the contents of the EIP register. The EIP register contains the offset within the code segment of the next instruction to be fetched.



EXERCISE:

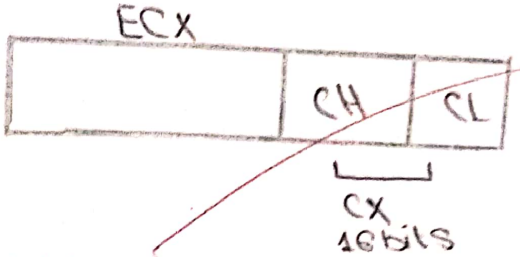
1. Fill in the following tables to show storage of 0xABDADDBA at address 1996 in the memory of a machine using (i) little endian (ii) big endian byte ordering.



2. What is the significance of learning ISA of a processor?

we get the correct understanding of how the instruction work and how the compiler uses those instructions, this is necessary for the development of the application. It means the programmer has complete understanding of the internal processor and how the instructions are handled. It helps understand processor functionality.

3. Show the ECX register and the size and position of the CH, CL, and CX within it.



4. For each add instruction in this exercise, assume that EAX contains the given contents before the instruction is executed. Give the contents of EAX as well as the values of the CF, OF, SF, PF, AF and ZF after the instruction is executed. All numbers are in hex. (Hint: add eax, 45 adds 45 to the contents of register eax and stores the result back in eax)

Contents of EAX (Before)	Instruction	Contents of EAX (After)	CF	OF	SF	PF	AF	ZF
00000045	Add eax,45	0000008A	0	0	0	0	0	0
FFFFFF45	Add eax,45	FFFFFF8A	0	0	1	0	0	0
000000045	sub eax , 46	FFFFFFFF	1	0	1	1	1	0
FFFFFF45	Sub eax, 9A	FFFFFEAB	1	0	1	0	1	0
FFFFFFFF	Add eax, 1	00000000	1	1	0	1	1	1

20/Jan/20

Scanned with OKEN Scanner