8086 Microprocessor

The 80886 is a 16-bit microprocessor. It has a 16-bit data bus with 20-bit address bus. Figure 1 shows the pin-outs of the microprocessor. It is packaged in 40-pins dual in-line package.

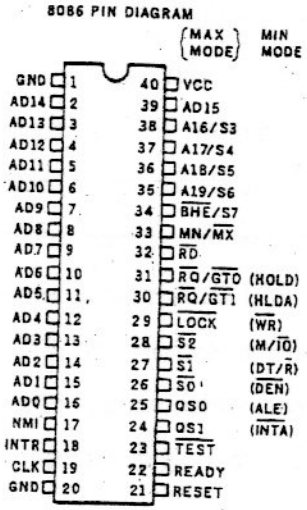


Figure 1: 8086 Pin Diagram.

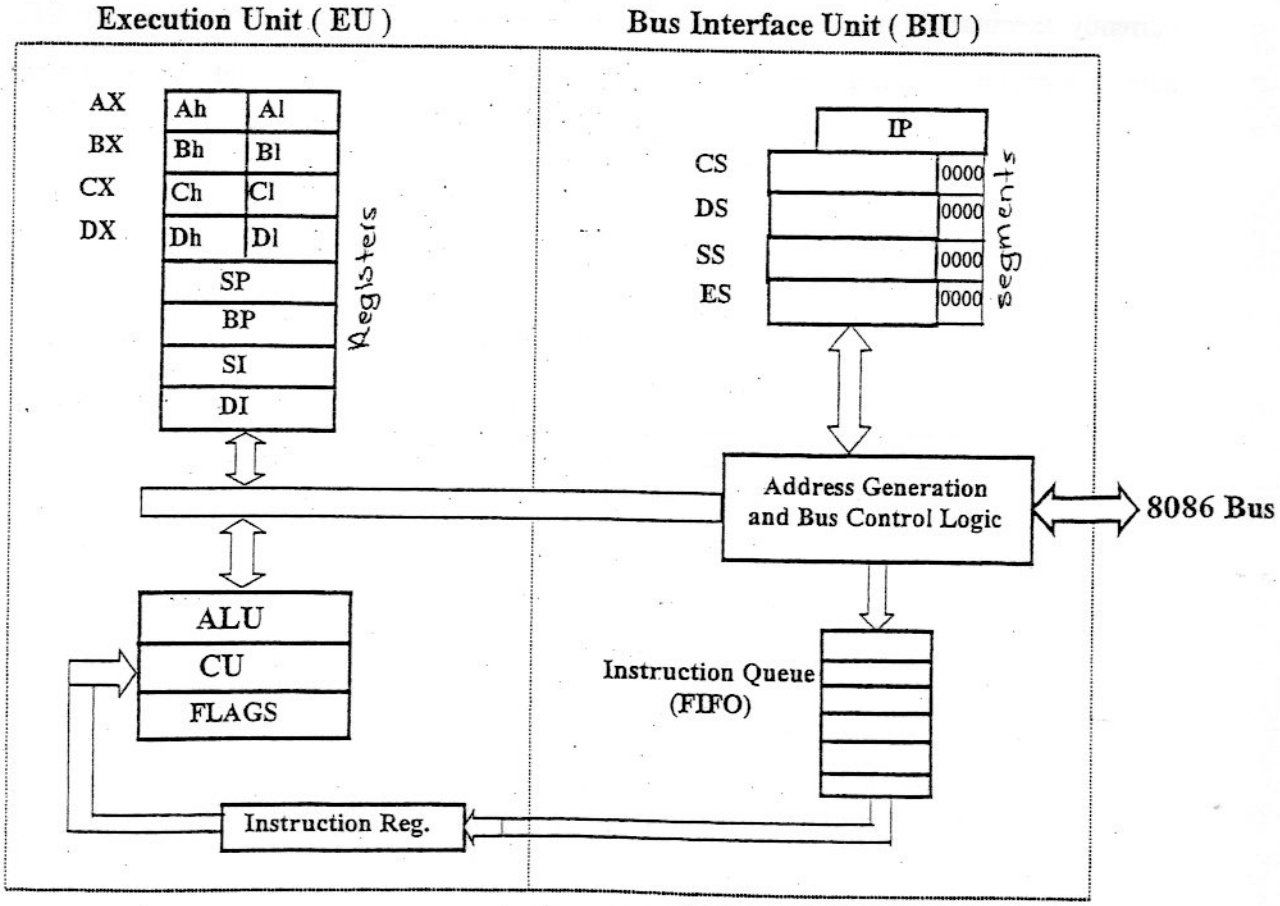


Figure 2 : The 8086 Architecture.

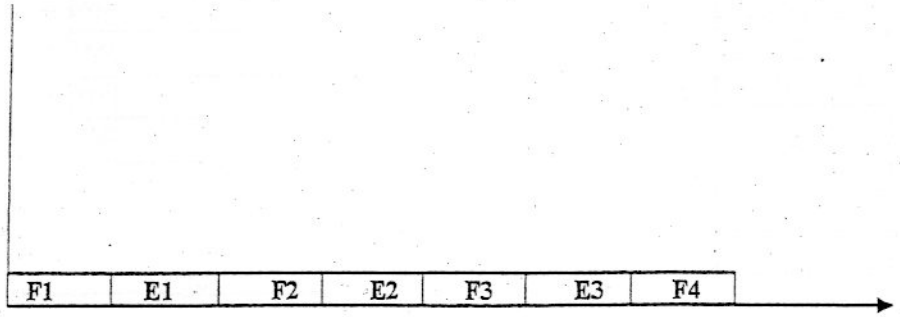
Execution Unit and Bus Interface Unit (8086)

As illustrated in Figure 2, the processor is partitioned into two logical units: an execution unit (EU) and a **bus interface unit (BIU)**. The role of the EU is to execute instructions, whereas the BIU delivers instructions and data to the EU. The EU contains an arithmetic and logic unit (ALU), a control unit (CU), and a number of registers. These features provide for execution of instructions and arithmetic and logical operations.

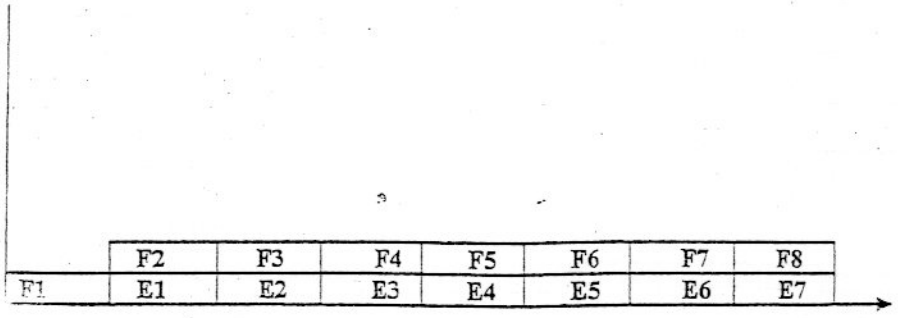
The most important function of the BIU is to manage the bus control unit, segment registers, and instruction queue. The BIU controls the buses that transfer data to the EU, to memory, and to external input/output devices, whereas the segment registers control memory addressing.

Another function of the BIU is to provide access to instructions. Because the instructions for a program that is executing are in memory, the BIU must access instructions from memory and place them in an ***instruction queue***, which varies in size depending on the processor (6 bytes in the 8086). This feature enables the BIU to look ahead and prefetch instructions so that there is always a queue of instructions ready to execute.

The EU and BIU work in parallel (***Pipeline Processing***), with the BIU keeping one step ahead. The EU notifies the BIU when it needs access to data in memory or an I/O device. Also, the EU requests machine instructions from the BIU instruction queue. The top instruction is the currently executable one and, while the EU is occupied executing an instruction, the BIU fetches another instruction from memory. This fetching overlaps with execution and speeds up processing.



Non-Pipelined Processing



Pipelined Processing

The Programming Model

The programming model of the 8086 is considered to be program visible because its registers are used during application programming and specified by the microprocessor instructions.

1. Segment Registers

A ***segment register*** is 16-bit long and provides for **addressing an area of memory** known as the current segment. Because a segment aligns on a paragraph boundary, its address in a segment register assumes 4 0-bits to its right.

CS (Code Segment) register: Contains the starting address of a program's code segment. This segment address, plus an offset value in the instruction pointer (IP) register, indicates the address of an instruction to be fetched for execution. For normal programming purposes, you need not reference the CS register.

DS (Data Segment) register: Contains the starting address of a program's data segment. Instructions use this address to locate data: This address, plus an offset value in an instruction, causes a reference to a specific byte location in the data segment.

SS (Stack Segment) register: Permits the implementation of a stack in memory, which a program uses for temporary storage of addresses and data. The system stores the starting address of a program's stack segment in the SS register. This segment address, plus an offset value in the stack pointer (SP) register, indicates the current word in the stack being addressed. For normal programming purposes, you need not directly reference the SS register.

ES (Extra Segment) register: Used by some string (character data) operations to handle memory addressing. The ES register is associated with the DI (index) register. A program that requires the use of the ES may initialize it with an appropriate segment address.

2. Pointer Registers

The three pointer registers are the **IP, SP**, and **BP**.

IP (Instruction Pointer) register. The 16-bit IP register contains the offset address of the next instruction that is to execute. The **IP** is associated with the **CS** register in that the IP indicates the current instruction within the currently executing code segment.

In the following example, the CS register contains 39B4[0]H and the IP contains 0514H. To find the next instruction to be executed, the processor combines the address in the CS and with the offset in the IP:

Segment address in CS	39B40H
Plus offset address in IP	+ 0514H
Address of next instruction	3A054H

*The **SP (stack pointer)** and **BP (base pointer)** registers are associated with the **SS register** and permit the system to access data in the stack segment.*

SP register: The 16-bit SP register provides an offset value, which when associated with the SS register, refers to the current word being processed in the stack. The system automatically handles these registers.

In the following example, the SS register contains segment address 4BB3[0]H and the SP contains offset 412H. To find the current word being processed in the stack, the processor combines the address in the SS with the offset in the SP:

Segment address in SS	4BB30H
Plus offset in SP	+412H
Address in stack	4BF42H

BP register: The 16-bit BP facilitates referencing parameters, which are data and addresses that a program passes via the stack. The processor combines the address in the SS with the offset in the BP.

3. General-Purpose Registers

The **AX, BX, CX,** and **DX** general-purpose registers are the workhorses of the system. They are unique in that you can address them as one word or as a **1-byte portion**. The leftmost byte is the "**high**" portion and the rightmost byte is the "**low**" portion. ***For example***, the **AX** register consists of an **AH** (high) and an **AL** (low) portion, and you can reference any portion by its name.

AX register: The AX register, the primary **accumulator**, is used for operations involving input/output and most arithmetic. ***For example***, the multiply, divide, and translate instructions assume the use of the AX. Also, some instructions generate more efficient code if they reference the AX rather than another register.

AX: AH AL

BX register: The BX is known as the **base register** since it is the only general-purpose register that can be used as an index to extend addressing. Another common purpose of the BX is for computations.

BX: BH BL

CX register: The CX is known as the **count register**. It may contain a value to control the number of times a loop is repeated or a value to shift bits left or right. The CX may also be used for many computations.

CX: CH CL

DX register: The DX is known as the **data register**. Some input/output operations require its use, and multiply and divide operations that involve large values assume the use of the DX and AX together as a pair.

DX: DH DL

4.Index Registers

The **SI (Source Index)** and **DI (Destination Index)** registers are available for indexed addressing and for use in addition and subtraction.

SI register: The 16-bit source index register is required for some string (character) operations. In this context, the SI is associated with the **DS** register.

DI register. The 16-bit destination index register is also required for some string operations. In this context, the DI is associated with the **ES** register.

5.Flag Register :

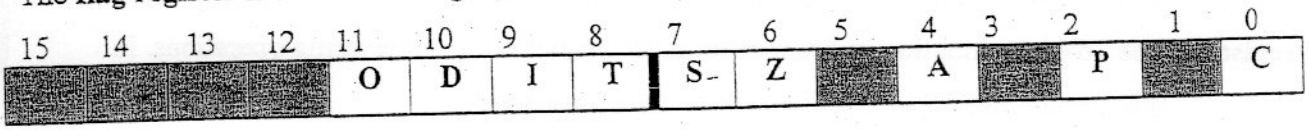
Nine of the 16 bits of the flag register are used in the 8086 processors. **Six** of these bits represent status flags. The Logic State of these status flags indicate conditions that are produced as a result of executing an instruction, such as ADD, specific flag bits are **reset** (logic **0**) or **set** (logic 1) based on the result that is produced. The following summarize the operation of these bits:

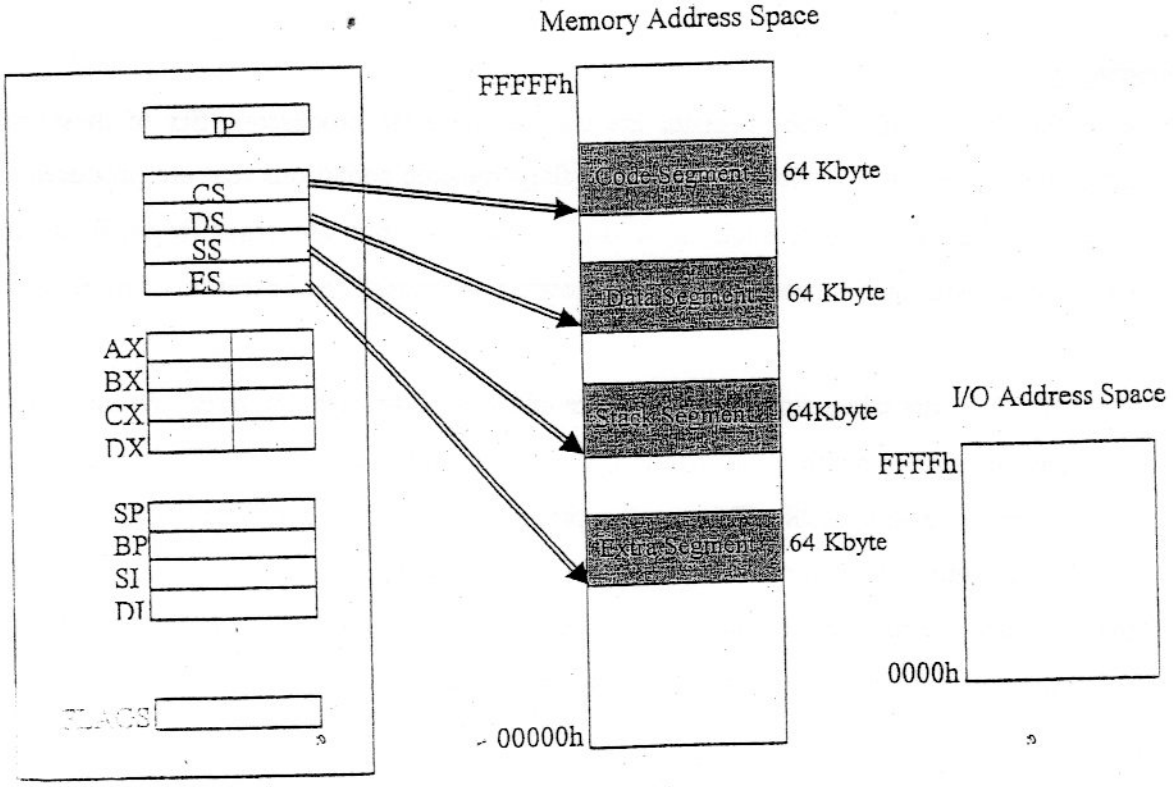
- **OF** (overflow). It occurs when signed numbers are added or subtracted. An overflow indicates that the signed result is out of the range. For example, if a 7F (+127) is added to 1, the result is 80h (-128). This result represents an overflow condition.
  For unsigned operations, the overflow flag is ignored.
- **SF** (sign). Contains the resulting sign of an arithmetic operation (0 =positive and 1 = negative).
- **ZF** (zero). Indicates the result of an arithmetic or comparison operation (0 = nonzero and 1 = zero result).
- **AF** (auxiliary carry). Contains a carry out of bit 3 on 8-bit data, for specialized arithmetic.
- **PF** (parity). Indicates even or odd parity of a low-order (rightmost) 8-bit data operation.
- **CF** (carry). Contains carries from a high-order (leftmost) bit following an arithmetic operation; also contains the contents of the last bit of a shift or rotate operation.

The other three flag bits are *control flags*. These three flags provide control functions on the 8086 processor as follows:

- DF (direction). Determines the direction of string operations. When set, the string instruction automatically ***decrements*** the address. On other hand, resetting DF causes the string address to be automatically ***incremented***.
- **IF** (interrupt). Indicates that all external interrupts, such as keyboard entry, are to be processed or ignored.
- **TF**(trap). Permits operation of the processor in single-step mode. Debugger programs such as DEBUG set the trap flag so that you can step through execution a single instruction at a time to examine the effect registers and memory.

The **flag register** has the following format:

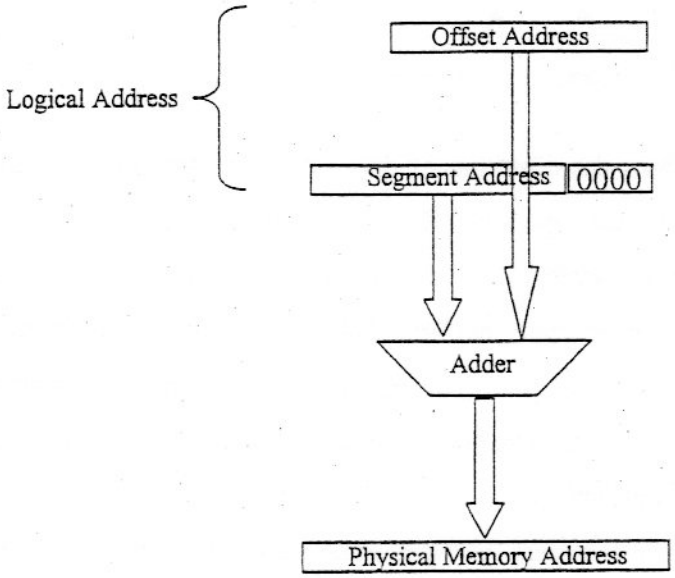




8086 Software Model

Generation of Memory Address

A ***logical address*** is described by a segment address and an offset address, both the segment base and offset are 16-bit quantities. However, the ***physical addresses*** that are used to access memory are 20-bits in length. The generation of the physical address involves combining a 16-bit offset value and a 16-bit segment base value that is located in one of the segment registers.



Physical Address = Segment Value * 10h + Offset Value

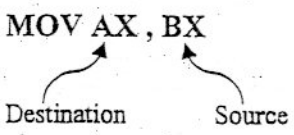
The microprocessor has a set of rules that apply to segments whenever memory is addressed. These rules define the segment register and offset register combination. **For example**, the code segment is always used with the instruction pointer to address the next instruction in program. This combination is **CS:IP**, the **CS** register defines the start of the code segment area and **IP** register defines the offset. Table 1 shows the default segment and offset address combinations.

Table 1 8086 default segment and offset combinations.

Segment	Offset	Purpose
CS	IP	Code Address (Instruction)
SS	SP or BP	Stack Address
DS	BX,DI,SI, or 8 or 16 bit number	Data Address
ES	DI for string instructions	String Destination Address

Data-Addressing Modes

Efficient software development for the microprocessor requires a complete familiarity with the addressing modes employed by each instruction. Because the MOV instruction is a common and flexible instruction, it provides a basis for the explanation of data-addressing modes.



The MOV instruction defines the direction of data flow. The **source** is to the right and the **destination** is to the left, next to the opcode MOV (An **opcode,** or operation code, tells the microprocessor which operation to perform). This direction of flow, which is applied to all instructions, is awkward at first. We naturally assume that things move from left to right, whereas here they move from right to left. Notice that a comma always separates the destination from the source in an instruction. Also, note that memory-to-memory transfers are not allowed by any instruction except for the **MOVS** instruction.

The **MOV AX, BX** instruction transfers the word contents of the source register (**BX**) into the destination register (**AX**). The source never changes, but the destination usually changes. It is essential to remember that a **MOV** instruction always *copies* the source data and into the destination. Note that the flag register remains unaffected by most data transfer instructions. The source and destination are often called **operands.**

Figure 3 shows all possible variations of the data-addressing modes using the **MOV** instruction. This illustration helps to show how each data-addressing mode is formulated with the **MOV** instruction and also serves as a reference. The data-addressing modes are as follows:

1. Register Addressing

Transfers a copy of a byte or word from the source register or memory location to the destination register or memory location. (Example: the **MOV CX,DX** instruction copies the word-sized contents of register **DX** into register-**CX**).

2. Immediate Addressing

Transfers the source-immediate byte or word of data into the destination register or memory location. (Example: the **MOV AL, 22H** instruction copies a byte-sized **22H** into register **AL**).

3. Direct Addressing

Moves a byte or word between a memory location and a register. The instruction set does not support a memory-to-memory transfer, except for the **MOVS** instruction. (Example: the **MOV CX,[1000]** instruction copies the word-sized contents of memory location **of offset 1000** into register **CX**).

4. Register Indirect Addressing

Transfers a byte or word between a register and a memory location addressed by an index or base register. The index and base registers are **BP, BX, DI,** and **SI.** (Example: the **MOV AX, [BX]** instruction copies &e word-sized data from the data segment offset address indexed by **BX** into register **AX**).

5. Base-plus-Index Addressing

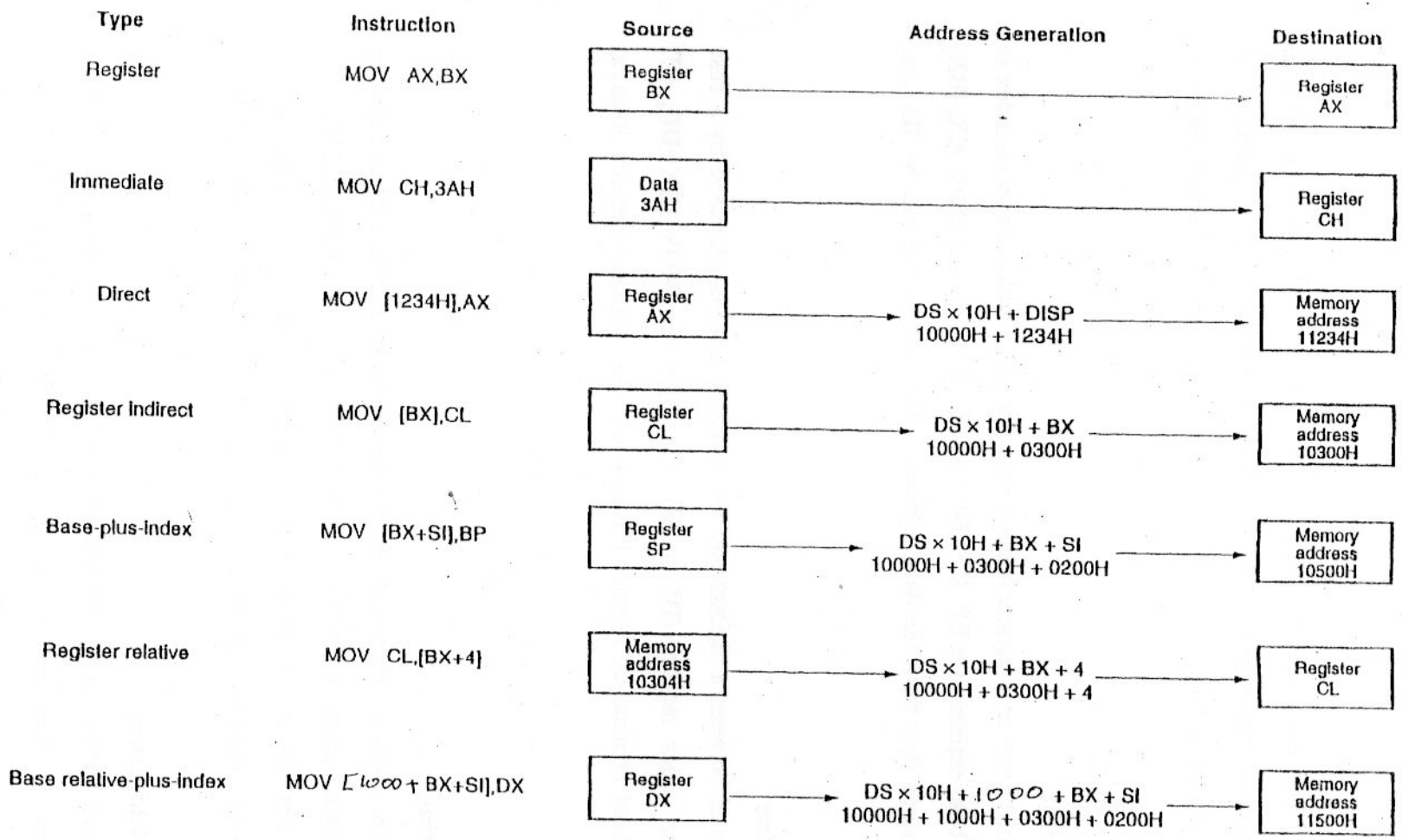
Transfers a byte or word between a register and the memory location addressed by a base register (**BP** or **BX**) plus an index register (**DI** or **SI**). (Example: the **MOV [BX+DI], CL** instruction copies the byte-sized contents of register **CL** into the data segment memory location addressed by **BX** plus **DI**).

6. Register Relative Addressing

Moves a byte or word between a register and the memory location addressed by an index or base register plus a displacement. (Example: **MOV AX,[BX+4]** or **MOV AX,ARRAY[BX]**. The first instruction loads **AX** from the data segment address formed by **BX** plus 4. The second instruction loads **AX** from the data segment memory location in **ARRAY** plus the contents of **BX**).

7.Base Relative-plus-Index Addressing

Transfers a byte or word between a register and the memory location addressed by a base and an index register plus a displacement. (Example: **MOV AX,ARRAY[BX+DI]** or **MOV AX,[BX+DI+4]**. These instructions load **AX** from a data segment memory location. The first instruction uses an address formed by adding **ARRAY, BX, and DI** and the second by adding **BX, DI,** and **4**).



Notes: BX = 0300H, SI = 0200H, ARRAY = 1000H, and DS = 1000H

8086 data-addressing modes.