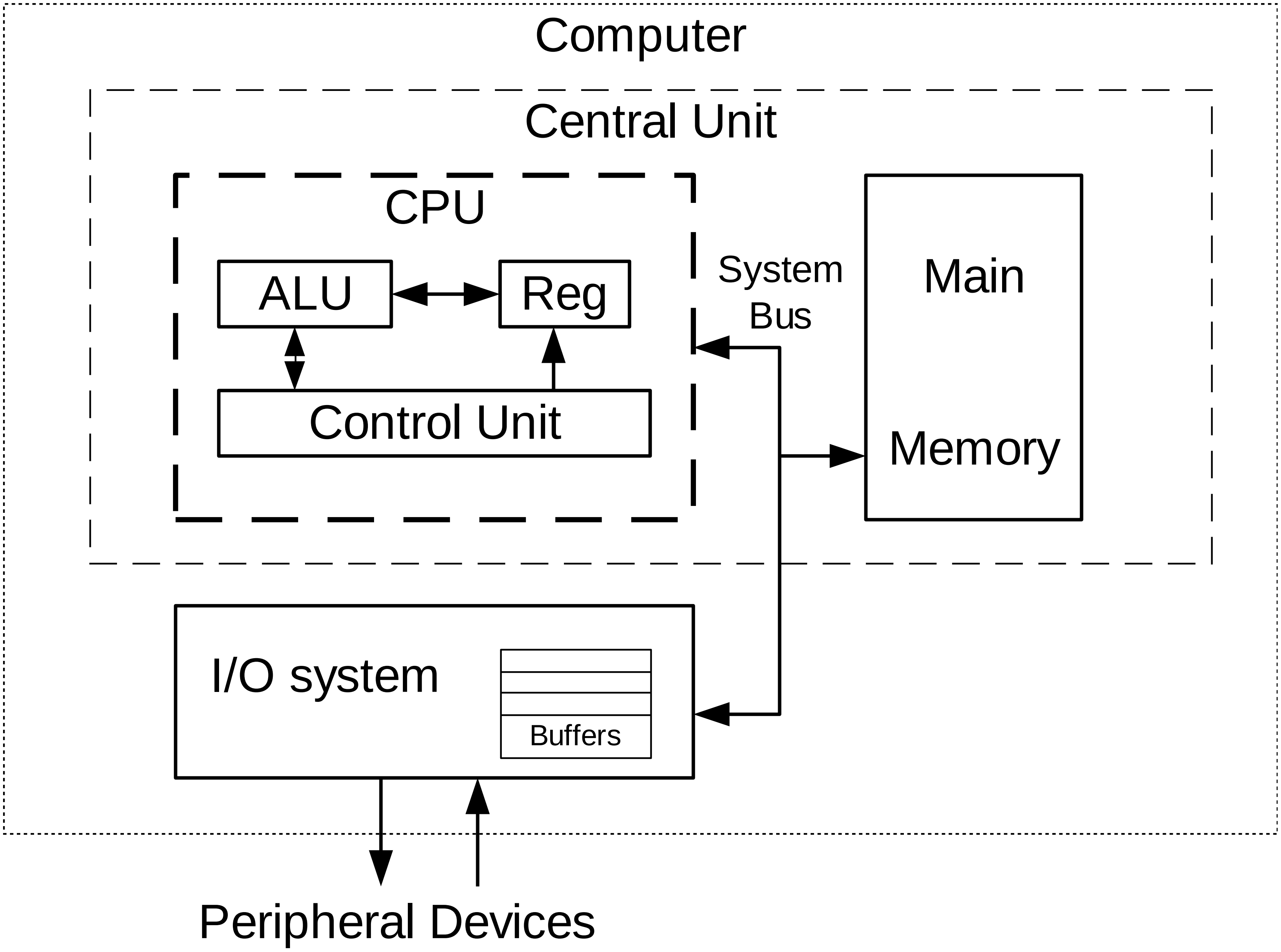
1. Computer components

Virtually, all contemporary computer designs are based on concepts developed by **John von Neumann.**

Such a design is referred to as the von Neumann architecture and is based on three concepts:

- Data and instructions are stored in a single read/write memory.
- The contents of this memory are addressable by location, without regard to the type of data contained there.
- Execution occurs in a sequential fashion from one instruction to the next.

The basic **Von Neumann** architecture:



CPU consists of Control Unit, ALU (Arithmetic and Logic Unit) and registers and represents a general purpose processor. in contrast with specialised processors (I/O processor, arithmetic processor) with a set of instructions, which means that it recognize and execute a set of instructions in a binary form.

CPU and main memory forms a **Central Unit**.

A Central Unit, Input/Output System and a set of system programs forms a **computer.**

A computer and peripheral devices forms a **computer system**.

Main memory (also named and internal memory) consists of a set of locations, defined by sequentially numbered addresses.

Each location contains a binary number that can be interpreted as either an instruction or data.

The internal memory can be of 2 types:

ROM (Read Only Memory)

RAM (Random Access Memory).

I/O system transfers data from external devices to CPU and memory and vice versa. It contains internal buffers for temporarily holding these data until they can be sent on.

Peripheral devices:

External memory devices **(**hard-disc, floppy-disc, compact-disc);

Input devices (keyboard, mouse);

Output devices (printer, monitor).

Data and instructions are communicated with the computer using input devices, the results are sent to output devices.

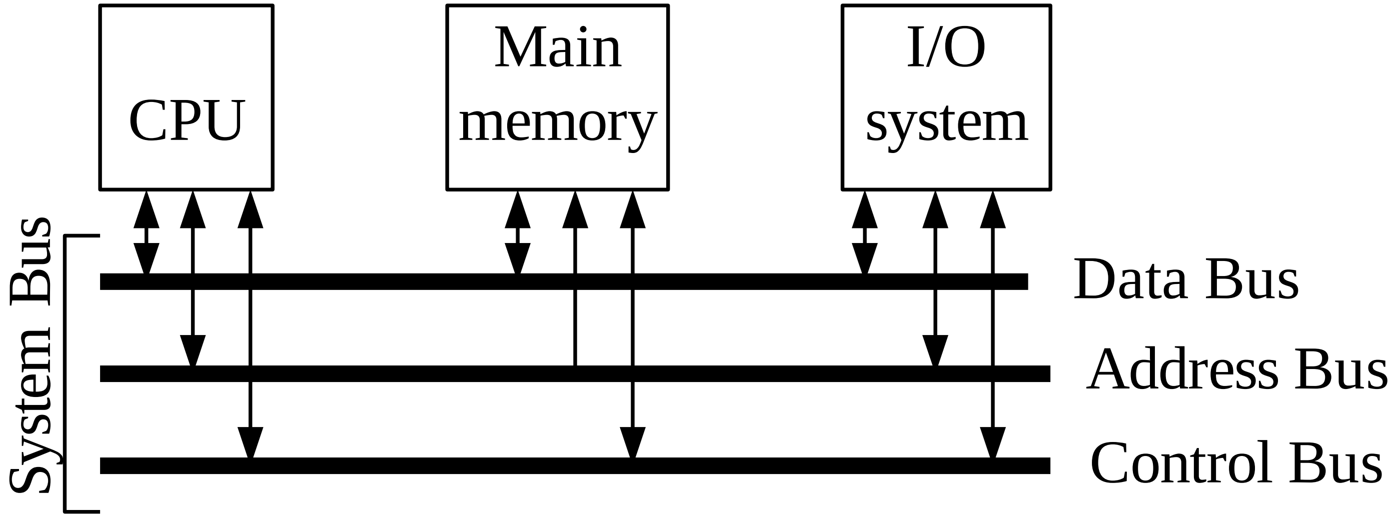
CPU interchanges with other components with data (operands and results), instructions, addresses, control signals. The communication is executed through buses.

The system bus:

Address bus: carries the address of a unique memory or input/output (I/O) device.

Data bus: carries data stored in memory (or in I/O device) to the CPU or from the CPU to the memory (or I/O device)**.**

Control bus: is a collection of control signals that coordinate and synchronize the whole system**.**



2. Central Processing Unit
2.1. CPU basics

A typical CPU has three major components:

(1) register set,

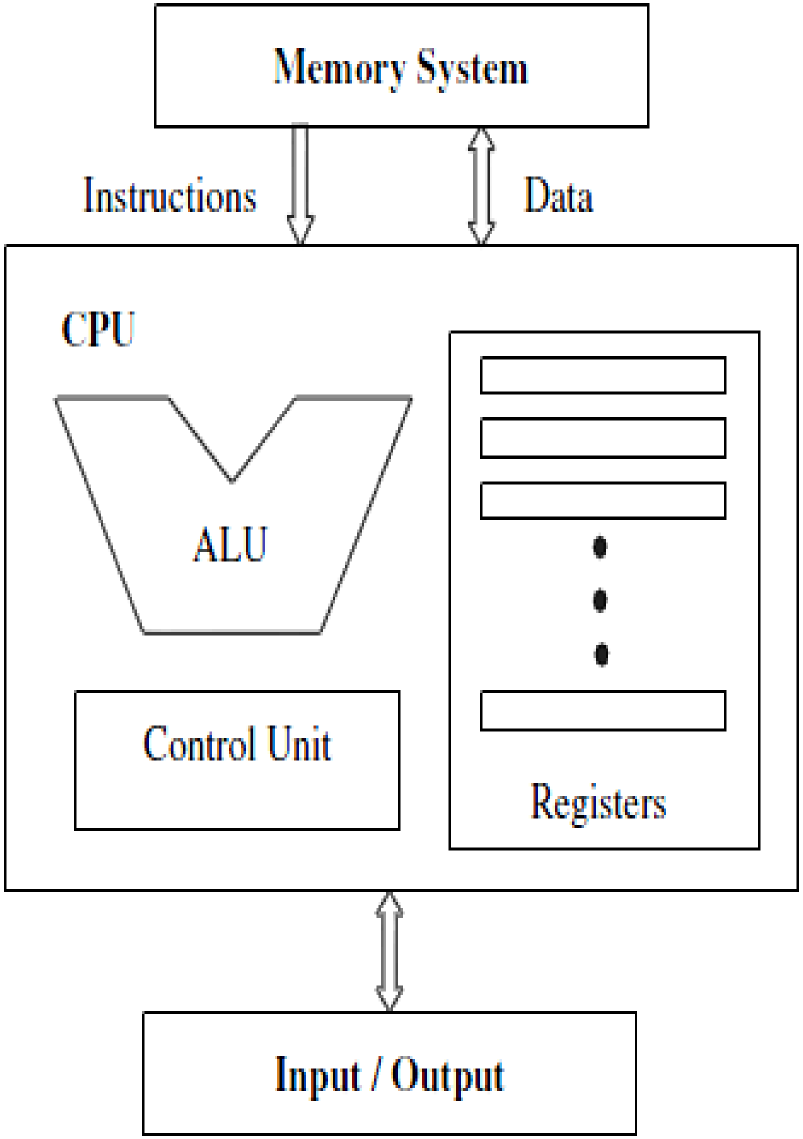
(2) arithmetic logic unit (ALU),

(3) control unit (CU).

The register set: **general-purpose** and special **purpose registers.**

The ALU provides the circuitry needed to perform the arithmetic, logical and shift operations. It consists of combinational logic circuits: **adders, decoders, encoders, multiplexers and a set of registers** (ex. acumulator), used as a fast memory in arithmetic and logic operations.

The control unit is the entity responsible for **fetching the instruction** to be executed from the main memory and **decoding** and then **executing** it.



Instruction cycle

The basic function performed by a computer is execution of a program, which consists of a set of instructions stored in memory.

The CPU reads (fetch) instructions from memory one at a time and executes each instruction. Program execution consists of repeating the process of instruction fetch and execution.

The processing required for a single instruction is called an **instruction cycle**. It consists of two steps:

fetch cycle

execute cycle

A typical instruction cycle :

1. **Instruction address calculation**: determine the address of the next instruction to be executed by adding a fixed number to the address of the previous instruction in PC.

2. **Instruction fetch**: Read the instruction from its memory location and store it into IR.

3. **Instruction decoding**: analyse instruction to determine type of operation to be performed and operands to be used.

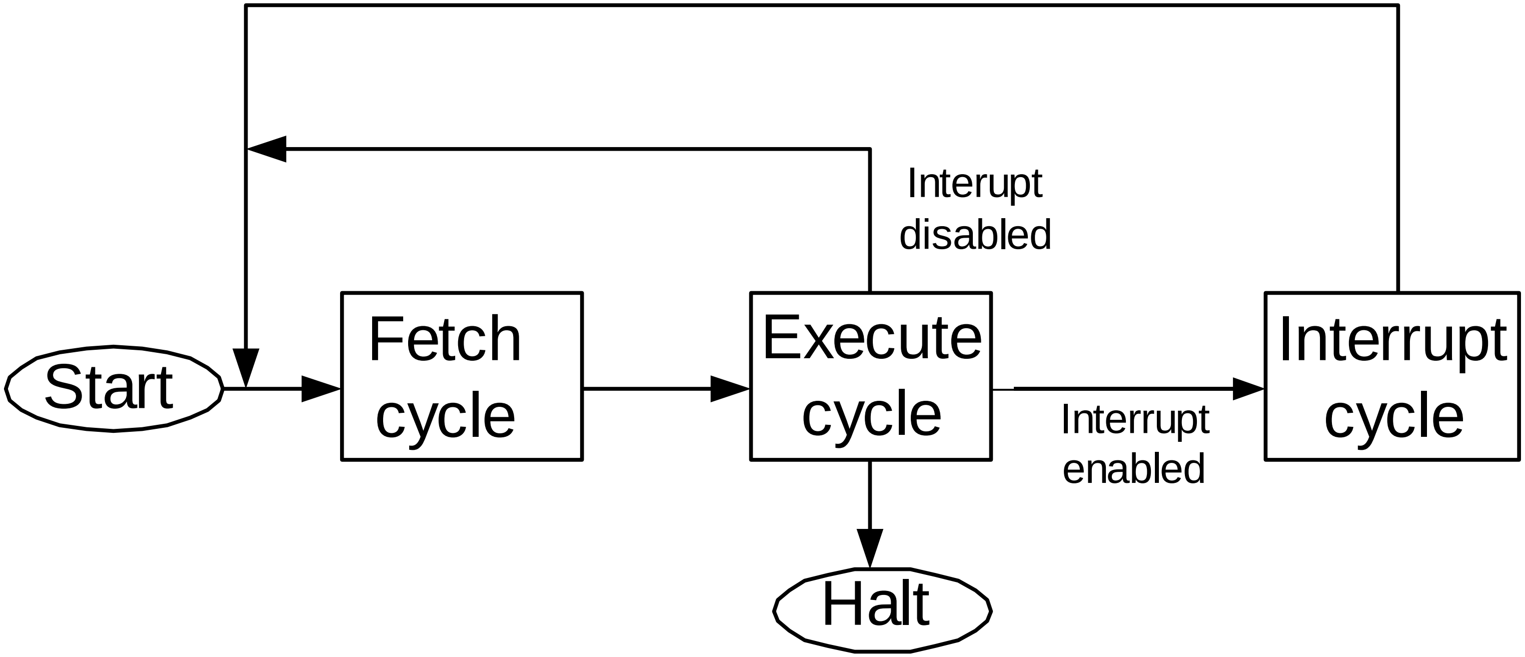
4. **Operands address calculation**, if needed.

5. **Operand fetch**: fetch the operand from memory and store it in CPU registers, if needed.

6. **Instruction execution**.

7. **Results store**: results are transferred from CPU registers to memory, if needed.

A check for pending interrupts is usually included in the cycle. Examples of interrupts include I/O device request, arithmetic overflow, division by zero, etc. Interrupts are provided primarily as a way to impove processing efficiency.



The actions of the CPU during an instruction cycle are defined by micro-orders issued by the control unit.

These micro-orders are individual control signals sent over dedicated control lines.

Example:

Let us assume that we want to execute an instruction that moves the contents of register X to register Y and both registers are connected to the data bus, D.

The control unit will issue a control signal to tell register X to place its contents on the data bus D.

After some delay, another control signal will be sent to tell register Y to read from data bus D.

2.4. I8086 microprocessor architecture

The I8086 microprocessor architecture consists of two sections:

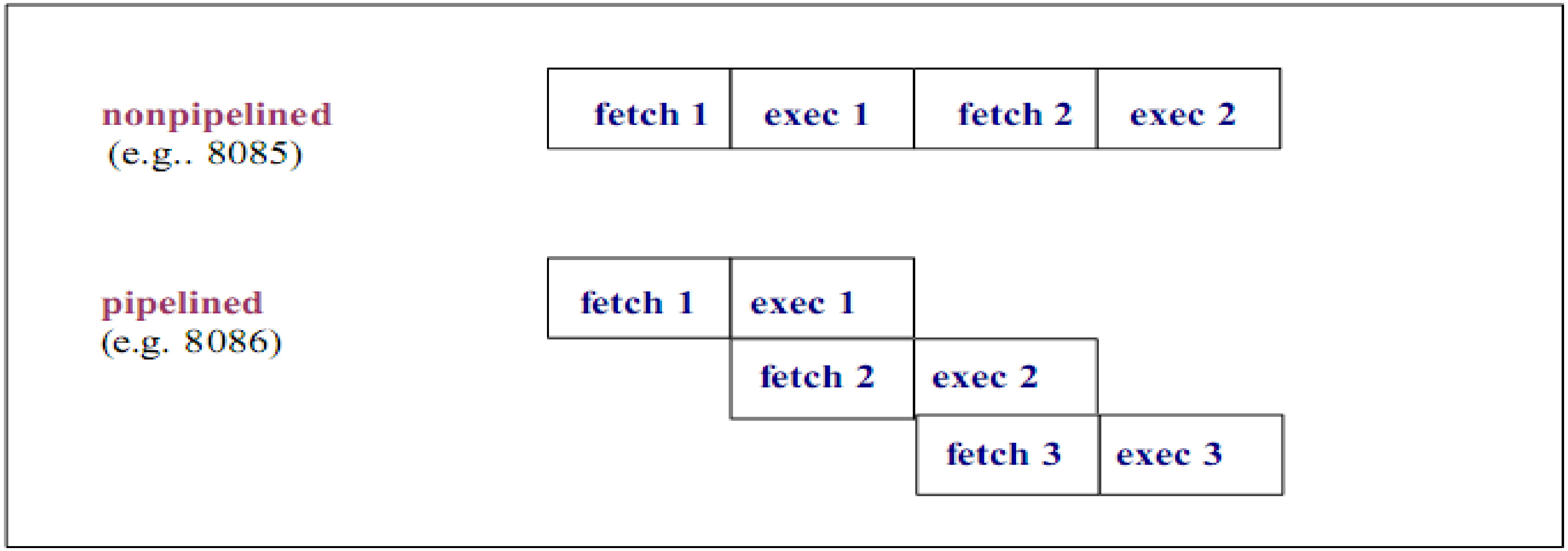
- the execution unit (EU)
- the bus interface unit (BIU)

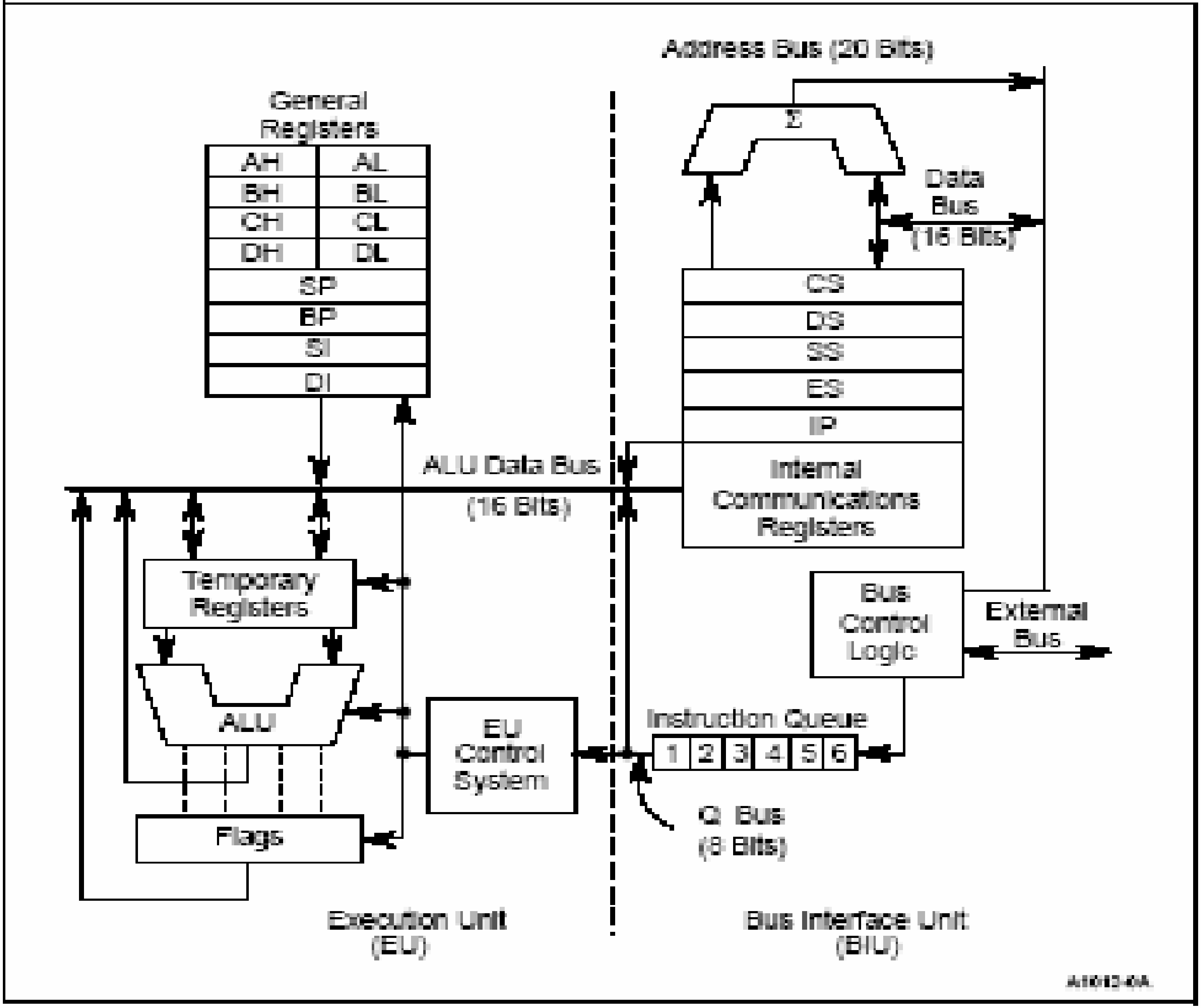
These two sections work simultaneously. BIU accesses memory and peripherals while the EU executes the instructions previously fetched.

Thus, Intel implemented the concept of pipelining.

Pipelining is the simplest form to allow the CPU to fetch and execute at the same time.

Pipelined vs. Nonpipelined Execution




Address Bus (20 Bits)
Σ
Data Bus (16 Bits)
General Registers
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
DI
CS
DS
SS
ES
IP
ALU Data Bus (16 Bits)
Internal Communications Registers
Temporary Registers
ALU
Flags
EU Control System
Bus Control Logic
External Bus
Instruction Queue
1
2
3
4
5
6
Q Bus (8 Bits)
Execution Unit (EU)
Bus Interface Unit (BIU)
A10124-0A

Execution Unit

The Execution Unit executes all instructions, provides data and addresses to the Bus Interface Unit and manipulates the general registers and the Processor Status Word (Flags register).

The 16-bit ALU performs arithmetic and logic operations, control flags and manipulates the general registers and instruction operands.

The Execution Unit does not connect directly to the system bus. It obtains instructions from a queue maintained by the Bus Interface Unit. When an instruction requires access to memory or a peripheral device, the Execution Unit requests the Bus Interface Unit to read and write data.

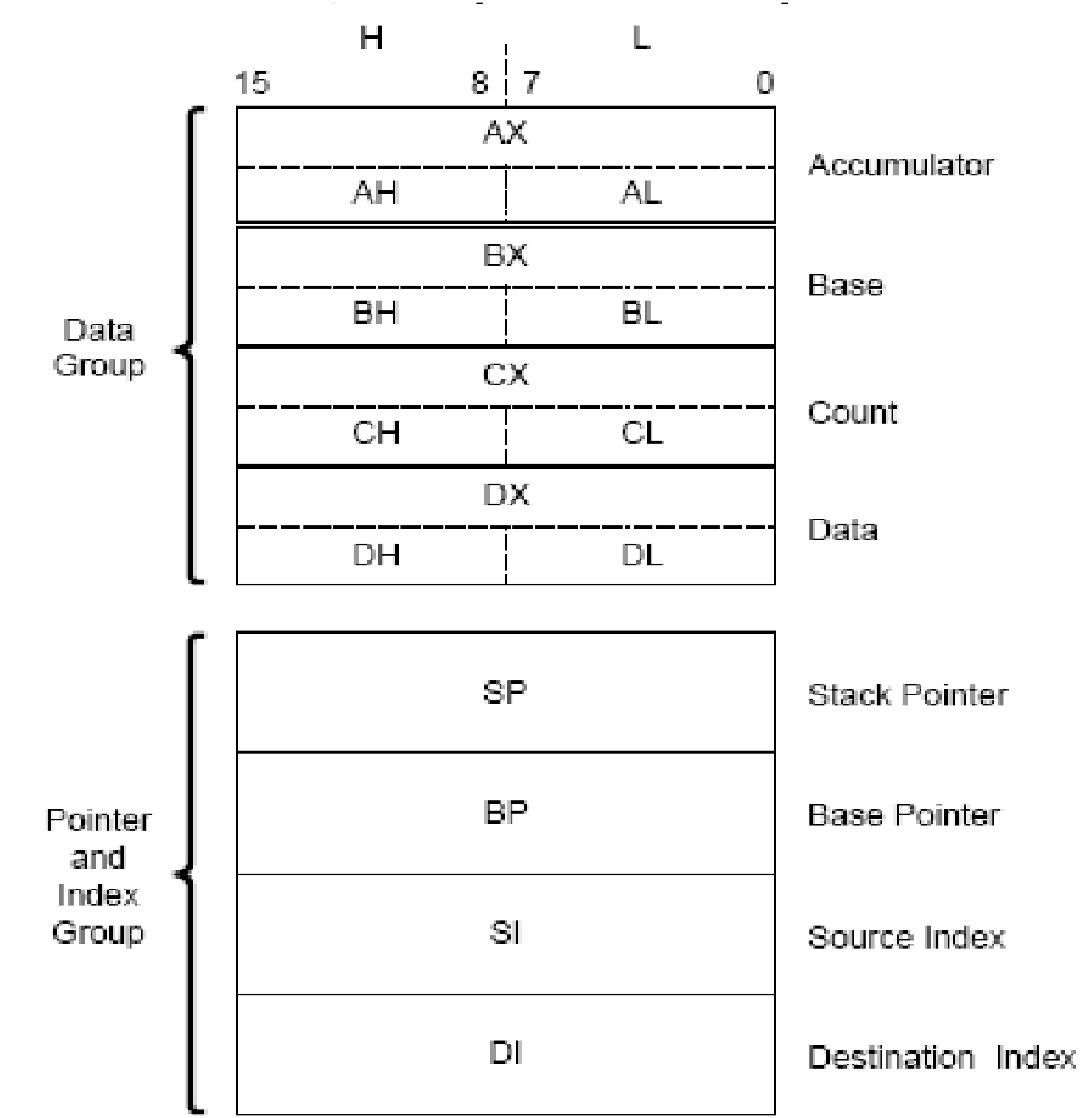
Bus Interface Unit

The Bus Interface Unit facilities communication between the EU and memory or I/O circuits.

It is responsible for transmitting address, data, and control signals on the buses.

This unit consists of the segment registers, the Instruction Pointer (or Program Counter), internal communication registers, a logic circuit to generate a 20 bit address, bus control logic that multiplexers data and address lines, the instruction code queue (6 bytes RAM).

2.5. Registers set of I8086



The data registers can be addressed by their upper or lower halves.

Each data register can be used interchangeably as a 16-bit register or two 8-bit registers.

The pointer and index registers are always accessed as 16-bit values.

The µ**p** can use data registers without constraint in most arithmetic and logic operations. Arithmetic and logic operations can also use the pointer and index registers. Some instructions use certain registers implicitly allowing compact encoding.

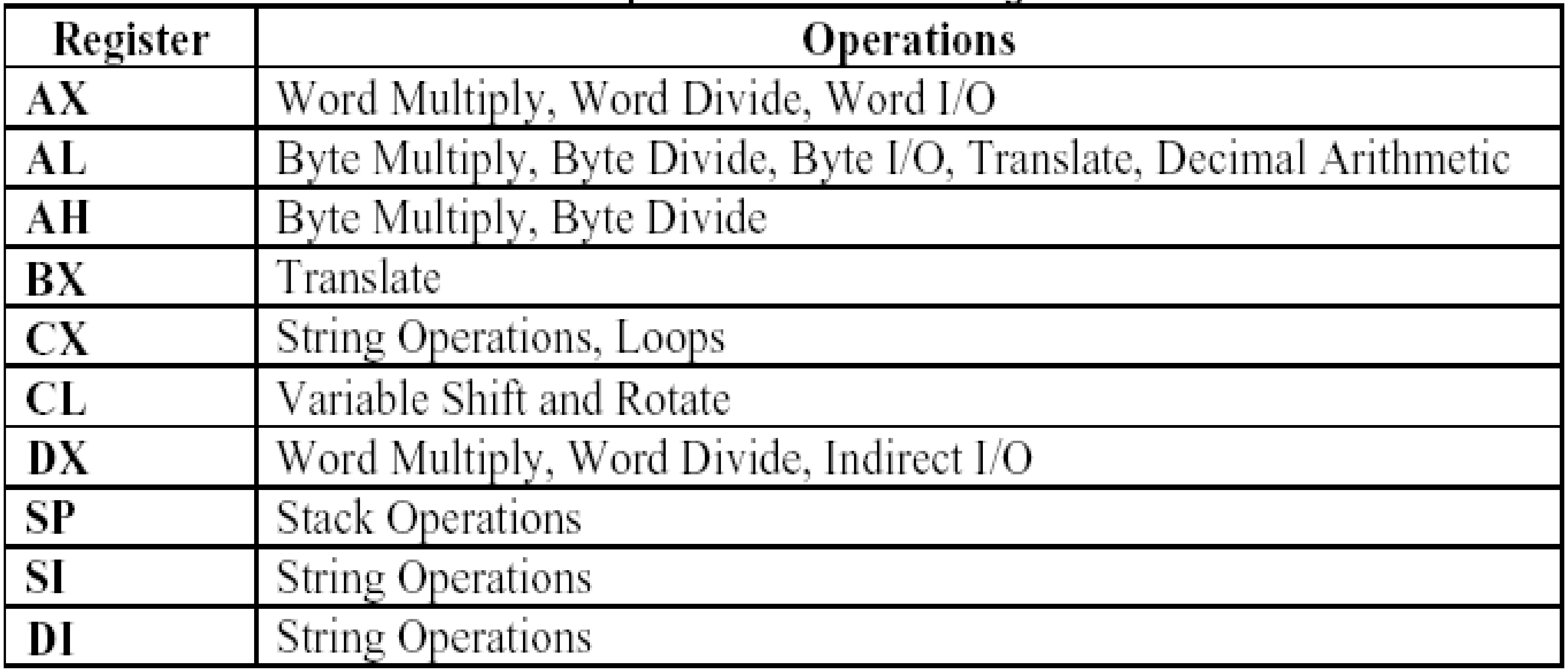
SP - Stack Pointer : Always points to top item of the stack.

BP - **Base Pointer**: It is used to access any item in the stack;

SI - Source Index: Contains the address of the current element in the source string;

DI - Destination Index: Contains the address of the current element in the destination string;

Table 1. Implicit Use of General Registers



Register	Operations
AX	Word Multiply, Word Divide, Word I/O
AL	Byte Multiply, Byte Divide, Byte I/O, Translate, Decimal Arithmetic
AH	Byte Multiply, Byte Divide
BX	Translate
CX	String Operations, Loops
CL	Variable Shift and Rotate
DX	Word Multiply, Word Divide, Indirect I/O
SP	Stack Operations
SI	String Operations
DI	String Operations

2. *Segment registers*

The mp 8086 has a 20-bit address bus for 1 Mbyte external memory but inside the CPU registers have 16 bits that can access 64 Kbytes.

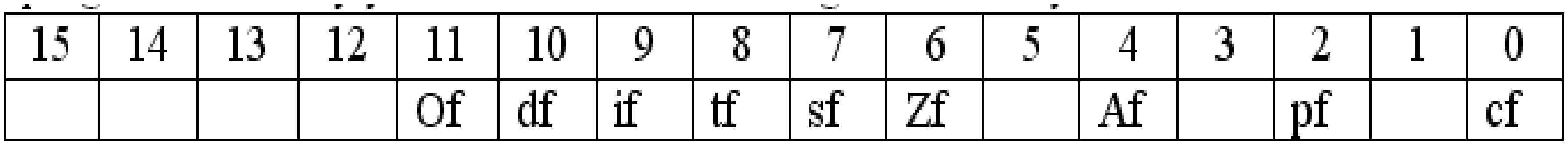
The 8086 family memory space is divided into logical segments of up to 64 Kbytes each. The segment registers contain the base addresses (starting locations) of these memory segments.

- **CS** (code segment) - points at the segment containing the current program.
- **DS** (data segment)- generally points at the segment where variables are defined.
- **ES** (extra segment)- extra segment register, it's up to a coder to define its usage.
- **SS** (stack segment)- points at the segment containing the stack.

3. Special purpose registers

IP - the instruction pointer or program counter: Always points to next instruction to be executed. It contains the offset (displacement) of the next instruction from the start address of the code segment.

Flags Register - determines the current state of the processor. From 16 bits are used only 9.



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				Of	df	if	tf	sf	Zf		Af		pf		cf

Condition flags:

0 bit -Carry Flag (CF) - this flag is set to **1** when there is a carry (borrow) from the 8 or 16 bit in addition or subtraction operation.

2 bit - Parity Flag (PF) - this flag is set to **1** when there is even number of one bits in result, and to **0** when there is odd number of one bits. Even if result is a word only 8 low bits are analyzed!

4 bit - Auxiliary Flag (AF) - set to **1** when there is an **unsigned overflow** for low nibble (4 bits).

6 bit - Zero Flag (ZF) - set to **1** when result is **zero**. For none zero result this flag is set to **0**.

7 bit - Sign Flag (SF) - set to **1** when result is **negative**. When result is **positive** it is set to **0**. Actually this flag take the value of the most significant bit.

11 bit - Overflow Flag (OF) - set to **1** when there is a **signed overflow**.

Control flags:

8 bit - Trap Flag (TF) System flag - Used for on-chip debugging when TF=1. In this case the interrupt is generated (int 1) which calls a special routine to show the state of internal registers.

9 bit - Interrupt enable Flag (IF) System flag - when this flag is set to **1** CPU reacts to interrupts on INTR input of the mp from external devices. When IF=0 interrupts are not allowed (masked). IF do not react to NMI (non maskable) interrupts and to internal interrupts performed by instruction INT.

Instructions CLI (clear interrupt) and STI (set interrupt) are used to control this flag.

10 bit - Direction Flag (DF) - this flag is used by some instructions to process data chains, when this flag is set to **0** - the processing is done forward (increment of SI and DI registers), when this flag is set to **1** the processing is done backward - decrement

(instructions CLD and STD).

The FLAGS register is the status register in Intel x86 microprocessors that contains the current state of the processor. This register is 16 bits wide. Its successors, the EFLAGS and RFLAGS registers are 32 bits and 64 bits wide, respectively. The wider registers retain compatibility with their smaller predecessors.

Intel x86 FLAGS Register

FLAGS

0	CF	Carry flag	S
1	1	Reserved	
2	PF	Parity flag	S
3	0	Reserved	
4	AF	Auxiliary flag	S
5	0	Reserved	
6	ZF	Zero flag	S
7	SF	Sign flag	S
8	TP	Trap flag (single step)	X
9	IF	Interrupt enable flag	X
10	DF	Direction flag	C
11	OF	Overflow flag	S
12, 13 only)	IOPL X	I/O privilege level (286+	
14 only)	NT X	Nested task flag (286+	
15	0	Reserved	

EFLAGS

16	RF	Resume flag (386+ only)	X
17	VM	Virtual 8086 mode flag (386+ only)	X
18	AC	Alignment check (486SX+ only)	X
19	VIF	Virtual interrupt flag (Pentium+)	X
20	VIP	Virtual interrupt pending (Pentium+)	X
21	ID	Identification (Pentium+)	X

22-31	0	Reserved

RFLAGS

32-63	0	Reserved

S: Status flag

C: Control flag

X: System flag

Main memory model

Instructions and data are stored in main memory.

The (main) memory can be modeled as an array of millions of adjacent cells, each capable of storing a binary digit (bit), having value of 1 or 0. These cells are organized in the form of groups of fixed number of cells.

An entity consisting of 8 bits is called a byte, of 16 bits – a **word,** of 32 bits – a **double word.**

In order to be able to move a byte in and out of the memory, a distinct address has to be assigned to each byte.

The number of bits, *l*, needed to distinctly address M bytes in a memory is given by

$$l = \log_2 M$$

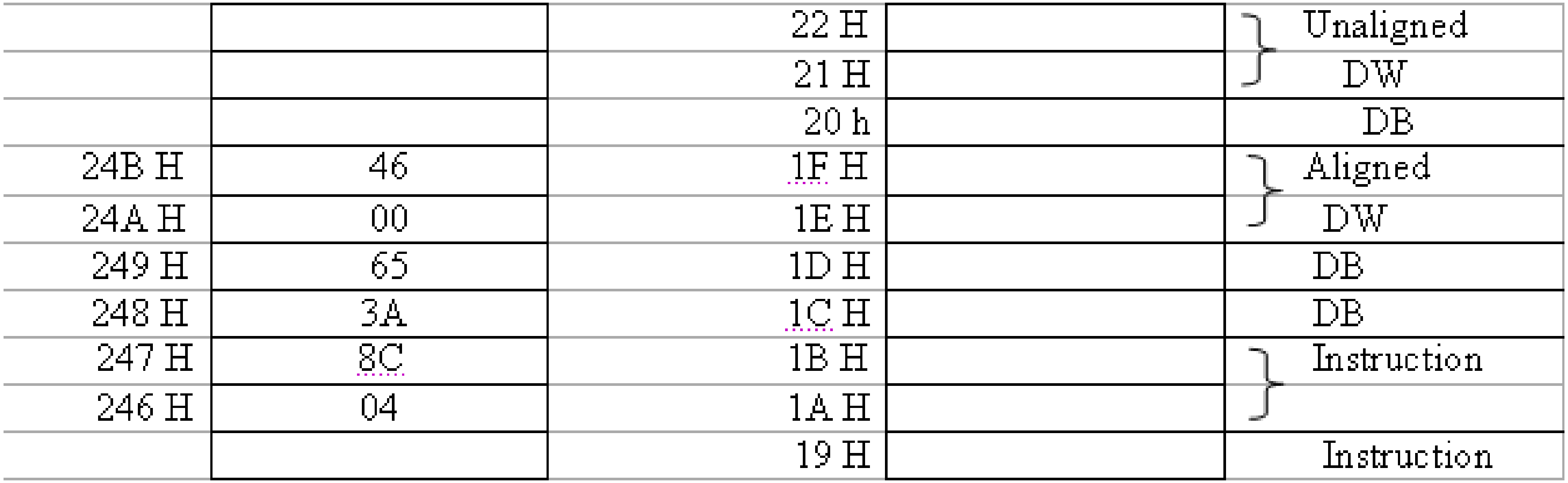
If the size of the memory is 1 MB, then the number of bits in the address is

$$\log_2(2^{20}) = 20 \quad \text{bits.}$$

	7 0
FFFFF	
FFFFE	
FFFFD	
....	
....	
10000	
0FFFF	
....	
00001	
00000	

The addressable memory of I8086 contains 2^{20} bytes (1 Mb). The physical addresses are within the range 00000-FFFFFh.

Locations 0H-7FH (128 bytes) and FFFF0-FFFFF (16 bytes) are reserved for special use (interrupts and system start after reset)

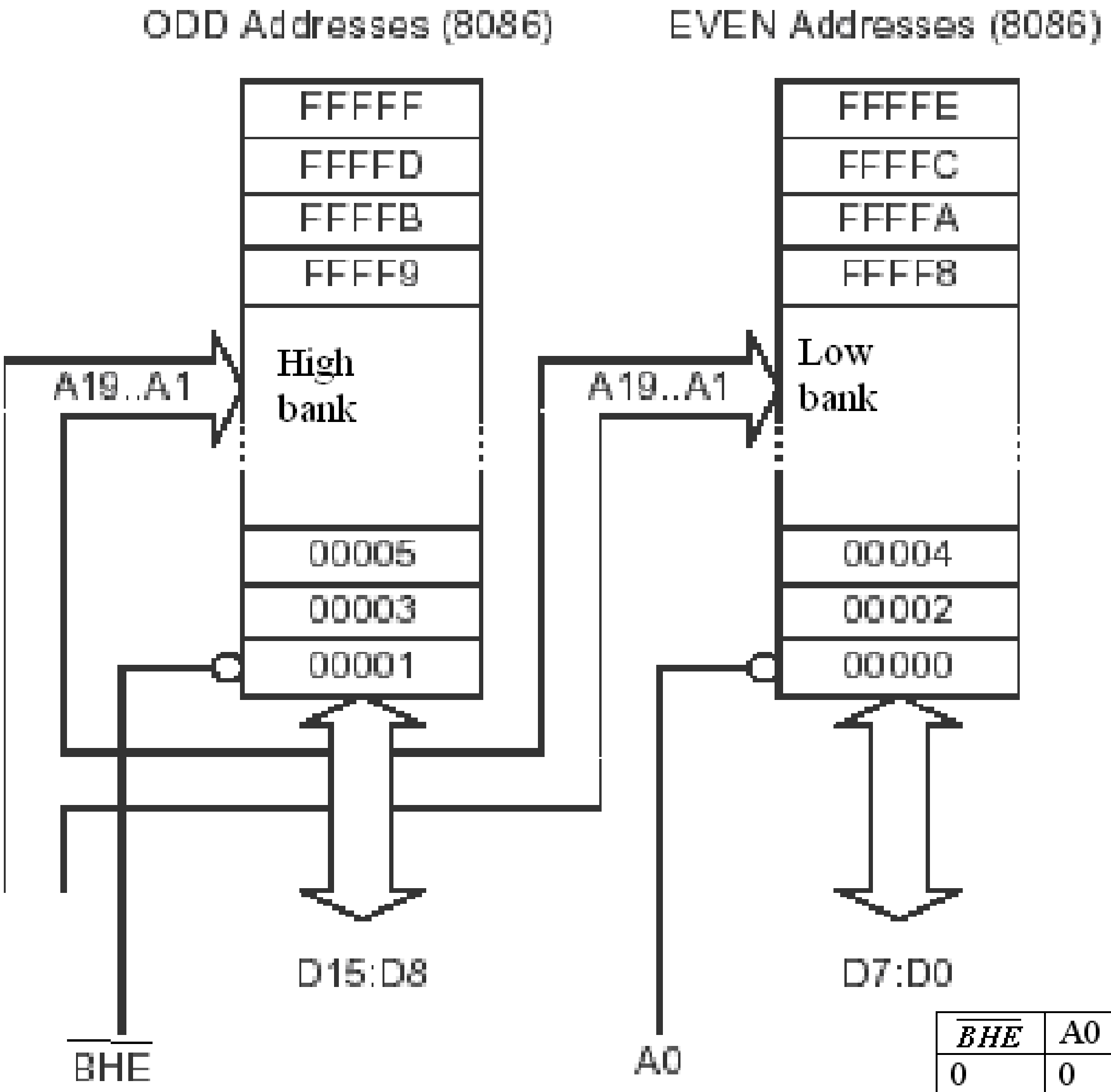


		22 H		Unaligned
		21 H		DW
		20 h		DB
24B H	46	1F H		Aligned
24A H	00	1E H		DW
249 H	65	1D H		DB
248 H	3A	1C H		DB
247 H	8C	1B H		Instruction
246 H	04	1A H		
		19 H		Instruction

Any 2 neighbour bytes can store a word (16 bits). The **smaller address** contains the **smaller byte**. The **address of the word** is the address of its **smaller byte**.

This strategy to store data is called Little Endian (the opposite strategy is called Big Endian and it applied by Motorola, Spark and most RISC machines).

The word with even address is called aligned. The word with odd address is called unaligned. The processor transfer words with even addresses in **1 memory access cycle** and words with odd addresses in 2 cycles.



$\overline{BHE}$	A0	Data
0	0	Word
0	1	Upper byte (odd address)
1	0	Lower byte (even address)
1	1	none

Memory segmentation

Segmentation provides a powerful memory management mechanism:

It allows programmers to partition their programs into modules that operate independently of one another.

Segments provide a way to easily implement object-oriented programs.

Segments allow two processes to easily share data.

It allows extending the addressability of a processor. In the case of the 8086, segmentation let Intel's designers extend the maximum addressable memory from 64KB to 1MB.

- **Disadvantage:** Difficulties with physical address manipulation in programs.

A full segmented address contains a segment component and an offset component

segment:offset.

On the 8086 through the 80286, these two values are 16 bit constants. On the 80386 and later, the offset can be a 16 bit constant or a 32 bit constant.

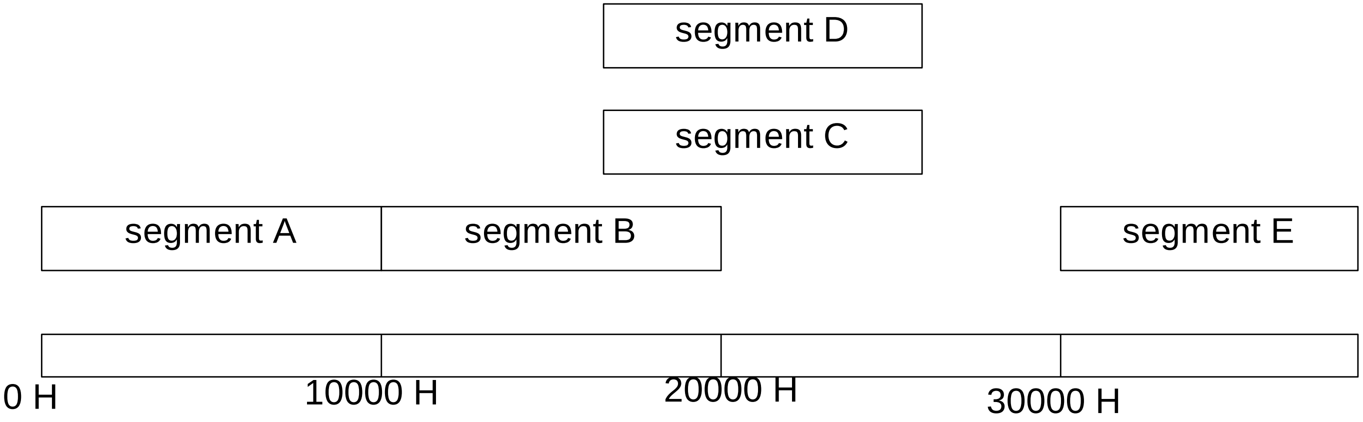
The size of the offset limits the maximum size of a segment.

On the 8086 with 16 bit offsets, a segment may be no longer than $2^{16}=2^{6}*2^{10}$=64KB; The 80386 and later processors allow 32 bit offsets with segments as large as $2^{32}=2^{2}*2^{30}$=4GB.

The segment portion is 16 bits on all 80x86 processors. This lets a single program have up to 65,536 different segments in the program.

All memory space is considered as a set of 64 Kbyte size segments. The segments are defined for each application. Segments are considered to be independent and uniquely addressable. For each program can be currently addressed 4 segments using CS, DS, ES and SS.

Segment registers are initialised at the beginning of the application. They contain the base (low) address of the segment which is always a multiple of 16 (4 low bits are considered 0).



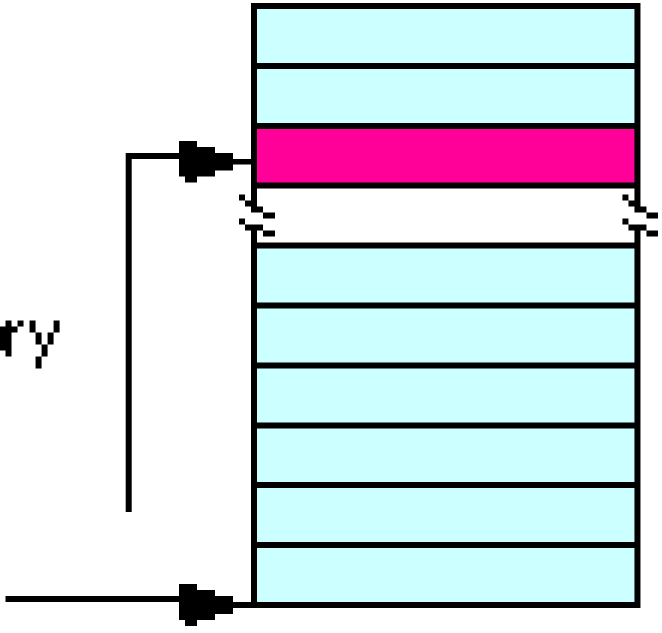
Physical address calculation

Addresses in the programs - **logical addresses**.

The linear address that appears on the address bus - **physical address.**

Logical address notation

segment: offset



Segment:offset

Plus the offset to obtain the address of the actual memory location to access.

Segment points here

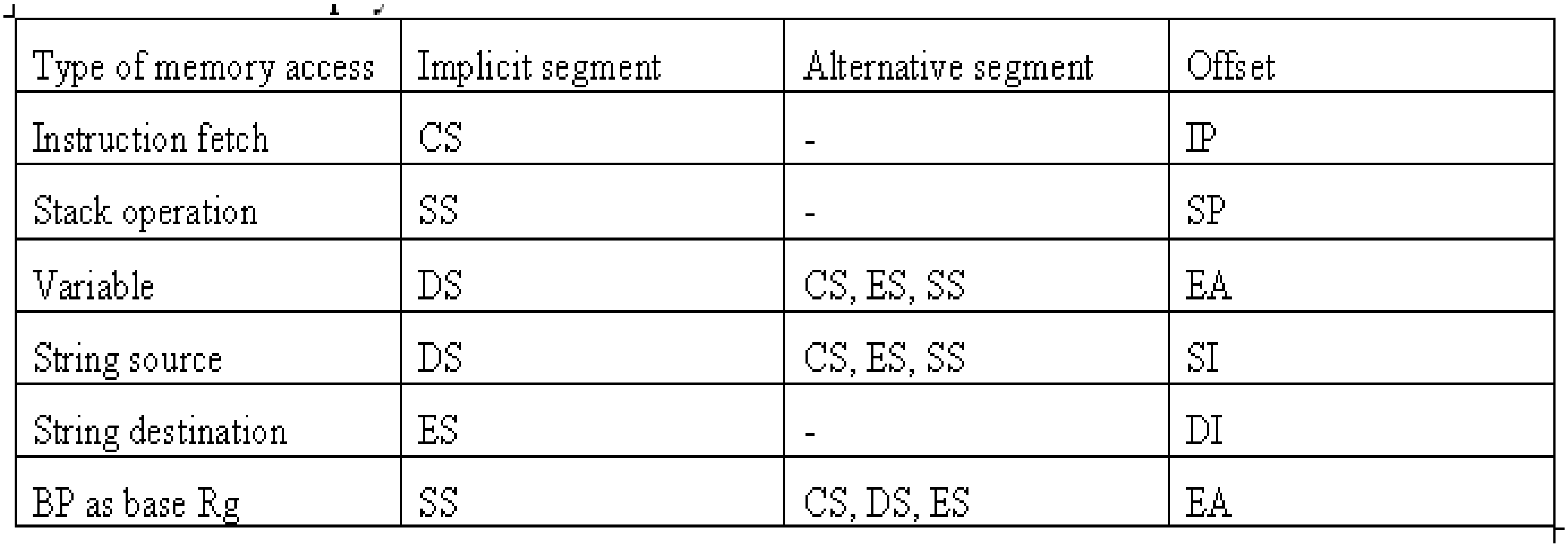
Physical address calculation ***segment*10H+offset***

Segment*10H is equivalent to 1 hexadecimal (4 bits) shift left. To calculate the physical address in BIU the base address is shift 4 bits left and the offset is added.

Example. If (CS)=123A h and (IP)=341B h, the physical address will be

$$\begin{array}{rl} 123A0 & \textbf{\textit{the base address of the segment}} \\ 341B & \textbf{\textit{offset}} \\ \hline 157BB & \textbf{\textit{a physical address}} \end{array}$$

Sources of physical address:



Type of memory access	Implicit segment	Alternative segment	Offset
Instruction fetch	CS	-	IP
Stack operation	SS	-	SP
Variable	DS	CS, ES, SS	EA
String source	DS	CS, ES, SS	SI
String destination	ES	-	DI
BP as base Rg	SS	CS, DS, ES	EA

Stack memory

A stack memory is a small area of reserved memory used in the following cases:

1. To store temporary the data from general purpose registers;
2. To store the content of PSW, CS and IP when an interrupt or a procedure is processed:
3. To transmit the procedures parameters.

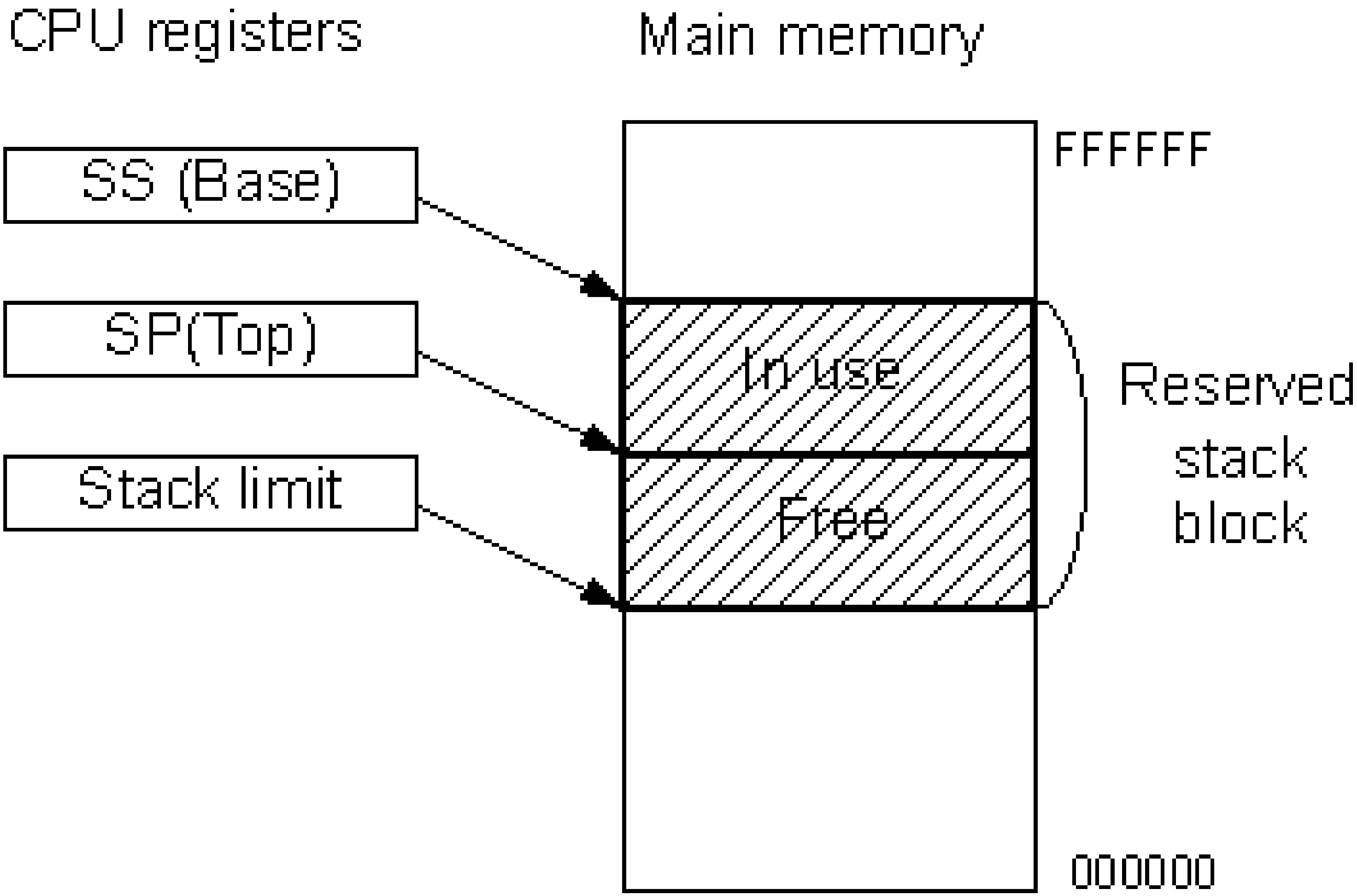
The stack organization principle is **LIFO.**

Stack location is determined by **SS:SP**.

SS holds the base address of stack and **SP holds the offset** of the top of the stack (the most recent stack entry).

Instructions to operate with stack are:

- **PUSH** - Copy specified word to top of the stack.
- **POP** - Copy word from top of the stack to specific location.


CPU registers
Main memory
SS (Base)
SP(Top)
Stack limit
FFFFFF
In use
Free
Reserved stack block
000000

According to Intel convention the stack grows from higher addresses to lower addresses

(according to Motorola convention the stack grows from lower addresses to higher addresses).

The base of the stack (SS) is at the high address end of the reserved stack block and the limit is at the low address end.

If all stack elements are 16-bit words (2 bytes), instruction **PUSH** will cause the **decrement of SP with 2** and **POP** will cause the **increment of SP with 2.**