Laboratory work No 5

Operand addressing modes in the microprocessor Intel® 8086

1. The aim of the work

To acquaint with instruction formats of the Intel® 8086, modes of operand addressing and formation of instruction codes.

2. General knowledge

2.1. Instruction formats

There are 133 basic instructions in the instruction set of the microprocessor Intel® 8086 using data both of 8 and 16 bits. Instructions can be without operands and can have 1 or 2 operands. A general instruction format is shown in Figure 1.

Repetition prefix	Segment modification prefix	Operation code byte (OCB)	Mod Reg R / M	Address	Data
0 or 1	0 or 1	1	0 or 1	0, 1, 2, 4	0, 1, 2
Length in bytes					

Fig 1. General format of the instruction

The repetition prefix (there can be no prefix) is applied in some cyclic procedures (e.g., in symbol line processing instructions) by repeating one or another instruction until the predetermined condition is fulfilled.

The segment modification prefix is applicable when instead of the segment which is used another segment should be used, e.g., instead of **DS** – **CS**.

The operation code byte (**OCB**) is in all instructions. If the instruction is of one byte, this byte is **OCB**.

The further byte of the instruction code (there can be no byte) is called the addressing byte (**AB**). Its format is shown in Figure 2.

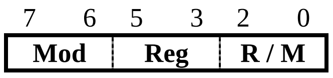


Fig 2. Format of the addressing byte

Field **Mod** of this byte determines the addressing mode applied by the instruction. How field **R / M,** which can mean either the register or the memory address, will be interpreted depends on its content. Field **Reg** indicates a register as one of the instruction operands. This field can also be applied to the operation code extension.

If **Mod** = 11, the instruction operands are in processor registers, which are indicated by fields **Reg, R / M** and the lowest **OCB** bit **W** (see Table 1).

Table 1. Addresses of microprocessor registers

Field **Reg** or **R / M**	Registers		Segment registers
	W = 0	**W** = 1	
000	**AL**	**AX**	**ES**
001	**CL**	**CX**	**CS**
010	**DL**	**DX**	**SS**
011	**BL**	**BX**	**DS**
100	**AH**	**SP**	–
101	**CH**	**BP**	–
110	**DH**	**SI**	–
111	**BH**	**DI**	–

Segment registers are addressed only by two bits, i.e. 00, 01, 10 and 11.

In other cases, fields **Mod**, **R / M** and the instruction code are determined by the addressing mode (see Table 2).

Table 2. Modes of operand addressing

R / M	Addressing mode		
	Mod = 00	**Mod** = 01	**Mod** = 10
000	**BX+SI**	**BX+SI+Disp L**	**BX+SI+Disp H Disp L**
001	**BX+DI**	**BX+DI+Disp L**	**BX+DI+Disp H Disp L**
010	**BP+SI**	**BP+SI+Disp L**	**BP+SI+Disp H Disp L**
011	**BP+DI**	**BP+DI+Disp L**	**BP+DI+Disp H Disp L**
100	**SI**	**SI+Disp L**	**SI+Disp H Disp L**
101	**DI**	**DI+Disp L**	**DI+Disp H Disp L**
110	**Disp H DipL**	**BP+Disp L**	**BP+Disp H Disp L**
111	**BX**	**BX+Disp L**	**BX+Disp H Disp L**

In this table **Disp L** is a lower byte with respect to the start of the displacement segment, **Disp H** is the higher byte of this displacement.

The addressing byte is unnecessary if:

- the instruction has no operands (e.g., **PUSH, RET**);
- the first operand of the instruction is the register, the second one is part of the instruction code (Immediate addressing);
- one operand is the memory address at direct addressing; the second one is register **AX** (**AL**);
- a direct addressing is applied in transition instructions;
- input / output instructions **(IN**, **OUT**) are used.

The address field of the instruction code can be of up to 4 bytes length. Here the displacement **Disp L**, **Disp H** and the basic segment address **Seg L, Seg H** are written.

In the last field of the instruction code direct data of the one or two byte length are written.

The instruction code length in bytes is equal to the sum of lengths of separate fields and is usually of 1 – 6 bytes.

2.2. Direct addressing

Direct addressing is such addressing when the address is indicated in the instruction code. In the data processing instructions a direct address is indicated by displacement following the addressing byte (Fig 3).

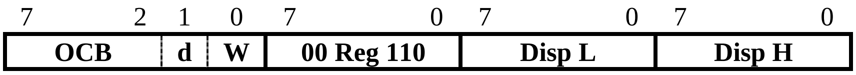


Fig 3. Format of the instruction with direct addressing

Bit “**d**” shows the data transmission direction. If **d** = 1, the operand is transmitted to the register determinable in field **Reg,** i.e., when **d =** 1– the register is a receiver, when **d =** 0 – the register is a transmitter.

In the instructions of the unconditional intersegment transition and of the access to subroutines a direct address is formed of 16-bit displacement **Disp** and the 16-bit segment address **Seg** (Fig 4).

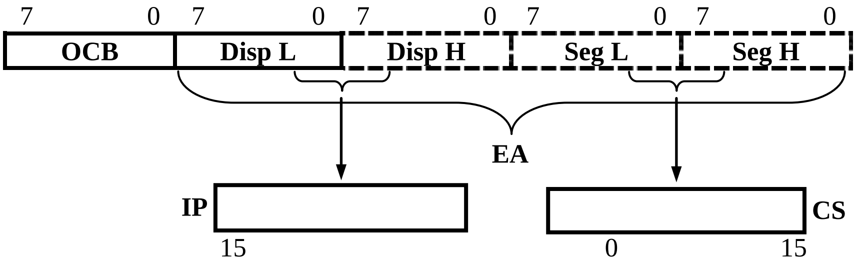


Fig 4. Format of instructions of unconditional transition and access to subroutines when addressing is direct

During the instruction execution, displacement is transmitted to instruction pointer **IP**, and the segment address – to register **CS**. In this case the instruction length is 5 bytes. If the transition is performed inside the segment, bytes **Seg L** and **Seg H** are not applied, while with close transition byte **Disp H** also becomes unnecessary. In such a case, the effective transition address is calculated by an equation:

$$EA = PA - (IP) + 1,$$

where **PA** is a physical address, to which it should be passed, (**IP**) is the last byte of address indicated in the instruction pointer.

2.3. Register addressing

In the instructions with register addressing, the operand is one of general purpose registers. Since 2 – 3 bits of the instruction code are enough for the register addressing, the codes of such instructions are not long (1 – 2 bytes). Besides, for their execution minimal computer time is needed because operands are inside the processor.

The following instruction formats are possible:

- the register address is indicated in the operation code byte;

7 3	2 0
OCB	**Reg**

- the address of a register (registers) is indicated in the addressing byte;

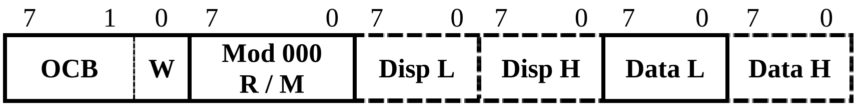
7 2	1	0	7 6	5 3	2 0
OCB	**d**	**W**	**Mod**	**Reg**	**R / M**

When **Mod =** 11, both operands are in registers, which are indicated by fields **Reg** and **R / M**. The purpose of bits **d** and **W** is the same as in the case of direct addressing, i.e. when **d =** 1, the register indicated in field **Reg** is the information receiver, when **W =** 1, operations are performed with 16-bit operands.

2.4. Immediate addressing

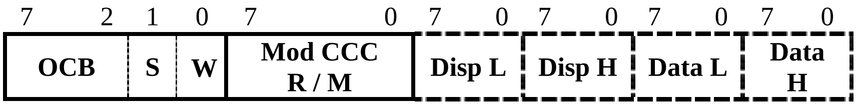
The direct operand **Data** occupies 1 or 2 bytes at the end of the instruction code. The lower byte **Data L** is always indicated as the first. The direct 1-byte operand, if the sign should be evaluated, is written by an adding code and can mean numbers from -128 to +127.

One of the possible formats of instructions with a direct operand is the following:



Unnecessary bytes of the instruction code are indicated by a dotted line. For example, if **W** = 0, then **Data H** = 0, and we will have the instruction of 5 byte length. By applying this format we can form a code of the instruction, which will send a direct operand to memory or will perform operation with another operand, which address is indicated in the addressing byte.

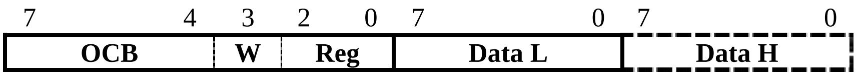
The format of arithmetical instructions with a immediate addressing can also be of up to 6 byte length.



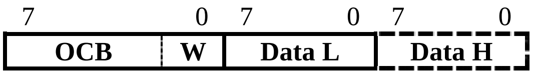
According to 3 bit field **CCC**, alongside with field **OCB** arithmetical operations are identified. If **SW** = 01, the direct operand is of 2 bytes. When **SW** = 11, the direct operand is of 1 byte, is written by the adding code and can mean numbers from -128 to +127.

Special instructions of short format (1 – 3 bytes) with a direct operand are also applied:

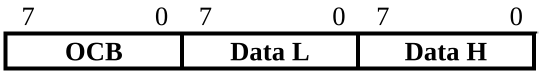
- to send a direct operand to the register;



- to perform arithmetical and logical operations if one of operands is in the accumulator;



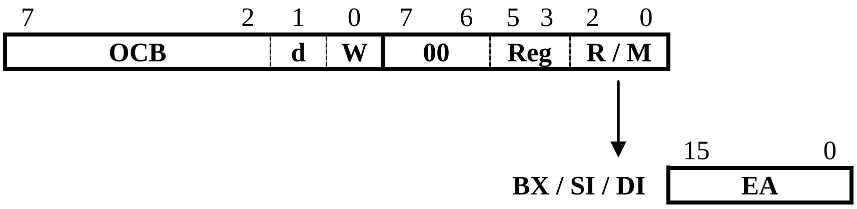
- in instructions of return from subprograms, when a constant should be added to the pointer;



The operating speed is typical of these three instructions because there is no need to access to memory.

2.5. Register indirect addressing

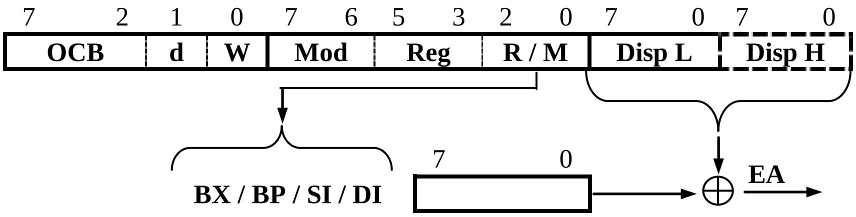
In the instructions with register indirect addressing the 16-bit execution (effective) address **EA** is in one of registers, which is determined by the addressing byte field **R / M**. For this purpose one of the registers **BX, SI** or **DI** can be used.



Such way of addressing is used in the instructions of unconditional transition and access to subroutines.

2.6. Base and index addressing

When addressing is the base the execution address **EA** is obtained by adding contents of registers **BX** or **BP** to 8 or 16-bit displacement.



If the displacement is of 8 bits, it is presented by the additional code from the number interval from -128 to +127.

When addressing is the base, it is possible to work with the data available at different memory sites.

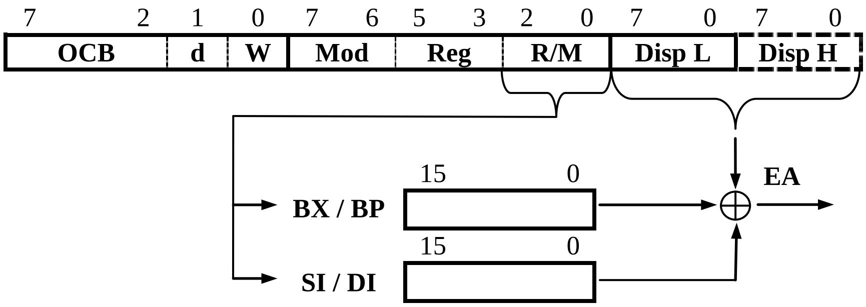
The index addressing differs from the base only in that instead of the base index register **BX** and base pointer register **BP** index registers **SI** and **DI** are used. Such way of addressing is convenient in processing data arrays, when the displacement shows the base address of the array, while the content of the index register – the element index of the array.

2.7. Base-index addressing

In this case the execution address **EA** is formed by adding up contents of base and index registers and the displacement.

The 8-bit displacement is presented by the additional code from the number range from -128 to +127.

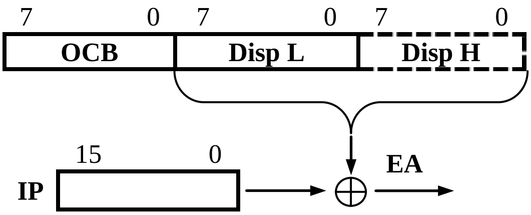
This type of addressing is used when working with the complex structure of data because it allows changing two address components at the same time.



2.8. Relative addressing

During this addressing the execution address **EA** is calculated by adding the current content (the address of the first byte of another instruction) of the instruction pointer **IP** to the displacement indicated in the instruction.

8 or 16-bit displacement is taken from the numbers with a sign in the ranges -128 ...+127 or -32768 ...+32767.



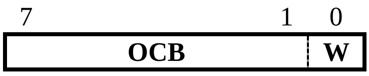
The relative addressing is applied only in instructions of transitions, access to subroutines and the cycle control.

2.9. Addressing applicable to the symbol line processing instructions

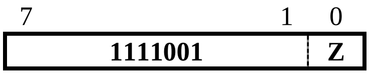
Bytes (**W** = 0) or words (**W** = 1) can be elements of symbol lines. Elements of the initial symbol line are addressed applying register **SI** and the elements themselves by default are placed in the data segment **DS**. The processed symbol line is always placed in the additional data segment **ES** and for addressing of its elements register **DI** is applied.

If before the instruction of the line element processing 1-byte repetition prefix **REP** is inserted, the corresponding operation (primitive) will be repeated until the content of the cycle counter **CX** becomes equal to 0. In executing the line processing instruction, the content of registers **SI** and **DI** is corrected automatically taking into account the value **DF** of the 11th bit of the flag register **FL**. If **DF** = 0, after each operation the content of corresponding index registers is increased by one unit when operations are executed with bytes and by two units – if it is operated with words. If **DF** = 1, the contents of index registers are decreased by one unit or two units.

The instructions of line processing are of 1 byte.



The repetition prefix is also of one byte.

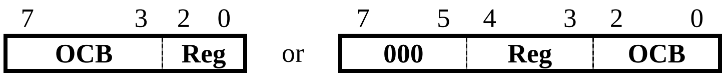


If the flag register flag **Z** (6th bit) does not coincide with bit **Z** of repetition prefix, the repeated operation of the line processing is aborted.

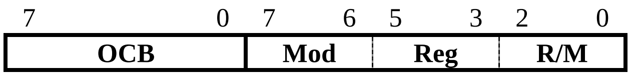
2.10. Stack addressing

The stack addressing is applied in instructions **PUSH** and **POP** (write into a stack and read from the stack). In this case the operand address is placed in the stack pointer **SP** and is automatically decreased or increased by two units by writing into the stack or reading from it. The stack is filled in the direction of the address decreasing.

The stack can exchange data with the general purpose registers and segment registers. Such instructions are of one byte.



The instructions executing the information exchange between the stack and memory are of 2 bytes.



The bit field **Reg** together with the field **OCB** identifies the instruction.

3. Task

1. From the below presented form to draw a table where the instruction execution results of microprocessor Intel® 8086 will be written.

The instruction execution results of microprocessor Intel® 8086

Address	Instruction hexadecimal codes	Performed functions	Operands	Execution results
Direct and register addressing instructions				
0000:1000 0000:1001 0000:1002 0000:1003 ...	**OCB** = **AB** = **Disp L** = **Disp H** = ...	...	...	...
Register indirect and immediate addressing instructions				
...	**OCB** = **AB** = ...	...	...	...

2. To switch the computer on, to start the training program info8086 and get acquainted with the microprocessor instruction set and modes of operand addressing.

3. To switch on the microprocessor training system MDA-8086 and to check the display monitor functioning.

4. By applying direct or register addressing to create and execute programs so that it was possible:

a) to send a word from memory to the accumulator;
b) to sum up two 16-bit operands available in memory;
c) to increment the accumulator content, to pass unconditionally to the address of another segment and once more to increment the accumulator content;
d) to perform the information exchange between the accumulator and registers **CX, DX** and **BX**;
e) to increment (decrement) the content of any register;
f) to send the content of any register to the accumulator;
g) without evaluation of the sign operand available in the accumulator, to multiply with the operand in register **BX**;

5. By applying the register indirect addressing or the immediate addressing to create and execute programs so that it was possible:

a) to send the direct operand of two (one) bytes to memory;
b) to add a direct operand of 2 bytes to the accumulator content;
c) to subtract from the memory cell content the content of another memory cell;
d) to increment the memory cell content;
e) to compare the direct operand with the operand in memory;
f) to determine the logical product of the direct operand and the operand in the accumulator;

6. By applying the base, index, base-index or relative addressing to create and execute programs so that it were possible:

a) to send the operand of two bytes from memory to the accumulator;
b) to multiply the operand available in the accumulator by the operand in memory;

c) if (**CX**) $\neq$ 0, to add a unit to the accumulator and to stop the program operation, if (**CX**) = 0 – to stop the program operation at once;
d) to perform operations of item 4c by applying the relative addressing.

7. To send the data line from segment **DS** to segment **ES**.

8. By applying the stack addressing to create and execute programs so that it was possible:

a) to send the operand form any register to the stack at the indicated address;
b) to send the operand form memory to the stack at the indicated address;
c) to send the operand form the stack to the register;
d) to send the operand from the stack to the random access memory.

4. Contents of the report

1. The aim of the work.
2. Examples of the instruction execution of direct and register addressing.
3. Examples of instruction execution with a register indirect and immediate addressing.
4. Examples of instruction execution of base, index, base-index and relative addressing.
5. Examples of instruction execution of the symbol line processing.
6. Examples of instruction execution of the stack addressing.
7. Conclusions.

5. Test questions

1. To draw a general instruction format of the microprocessor Intel® 8086.

2. To draw the addressing byte structure and explain the purpose of its fields.

3. To explain the principle of direct addressing.

4. To explain the principle of register addressing.

5. To give an example of the instruction with a immediate addressing.

6. To explain the principle of base, index and base-index addressing.

7. How is the physical address calculated in the case of relative addressing?

8. Format of instructions processing symbol lines.

9. To explain the principle of stack addressing.

References

1. Berger A. S. Hardware and Computer Organization. USA, Burlington: Newnes; Book & DVD Edition. May 6, 2005. 512 p.

2. Brey B. B. The Intel Microprocessors 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro Processor, Pentium II, Pentium III, Pentium 4. Architecture, Programming and Interfacing. USA, New Jersey: Pearson Prentice Hall; 7th Edition. March 23, 2005. 912 p.

3. Triebel W. A. The 8088 and 8086 Microprocessors: Programming, Interfacing, Software, Hardware and Applications. USA, New Jersey: Pearson Prentice Hall; 3rd Edition. August 29, 2002. 1040 p.

4. Uffenbeck J. The 80x86 Family: Design, Programming and Interfacing. USA, New Jersey: Pearson Prentice Hall; 3rd Edition. February 14, 2001. 678 p.

5. Antonakos J. L. Introduction to the Intel Family of Microprocessors: A Hands-On Approach Utilizing the 80x86 Microprocessor Family. USA, New Jersey: Pearson Prentice Hall; 3rd Edition. June 3, 1998. 768 p.

6. Gražulevičius A. Mikroprocesoriai. Laboratorinių darbų užduotys ir metodikos nurodymai. Vilnius: Technika, 2000. 60 p.