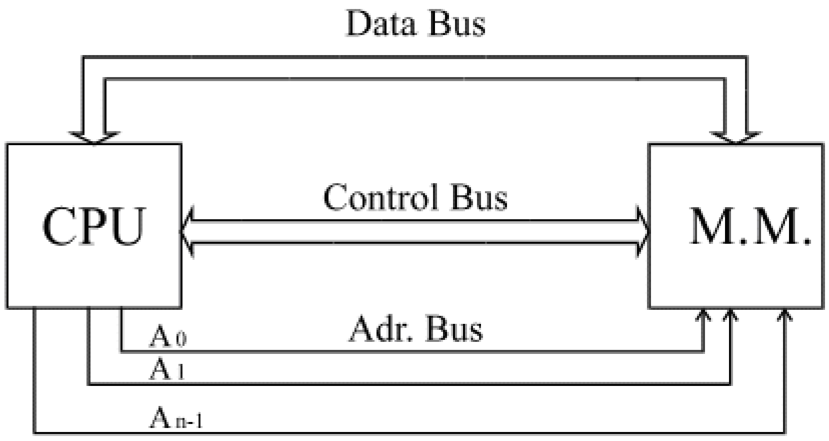
MEMORY ADDRESSING TEHNIQUES

General considerations concerning the memory addressing

It has been presented the fact that in modern computers the memory unit is implemented with integrated modules. The connection between CPU and main memory is realized by means of three buses:

- The address bus ADRBUS;
- The data bus DATA BUS;
- The control bus CONTROL BUS.



The address bus is unidirectional; the data bus is bidirectional, while the control bus contains command and status lines. The memory is organized on locations:

0	B_{m-1}	………	B_1	B_0
1				
	…	…	…	…
k-1				
k				
k+1				
	…	…	…	…
$2^n - 1$				

A memory location has **m** ranks noted B_{m-1}, … B_1, B_0.

Reading and writing the memory is performed in parallel, in other words one reads or writes an **m** bit word at a time. Addressing the memory is done via the address bus (ADRBUS) and it is assumed this bus comprises **n** lines. Each combination of bits of this bus defines a certain address. The connection between the bus content and a specific location is done via a decoding process. If the address bus has **n** lines then the address set is $\mathbf{2^n}$.

$$A = \{0,1,...2^n -1\}$$

where: A = address space

On the other hand, given a certain memory geometry, it results the set of physical locations where **m** bit words are stored.

Definition: The set of all physical locations defined by a geometry is called the space of the memory M.

The connection between the address space and the memory space is given by a translation function, defined as: $f_T : A \rightarrow M$.

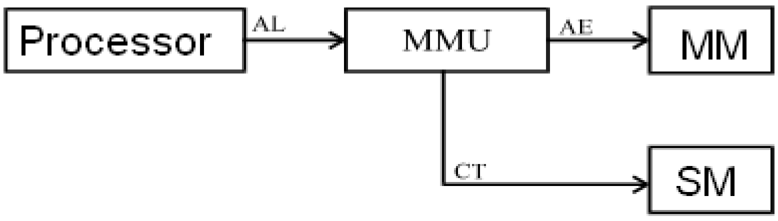
In accordance to the previously mentioned notions, this function corresponds to a decoding function. In case of simple systems (microcomputers) the two sets will be identical M = A so **f** is a simple function.

When the width of the memory word is not enough to represent the data, then several consecutive locations are used. If for instance, a datum is representable on two words in memory – AIU, meaning addressable informational unit, then, in order to represent it, the first location is used for the least significant word (from address **j**), and the next location (address **j+1**) for the most significant word.

Modern processors present memory management facilities. These memory management operations refer to virtual memory implementation, memory protection, etc.

The Memory Management Unit – MMU can be built in the control unit inside the CPU or can be external (with respect to the same control unit). In particular, in case of microprocessors it can be incorporated or sold separately as an additional module – the MMU module.

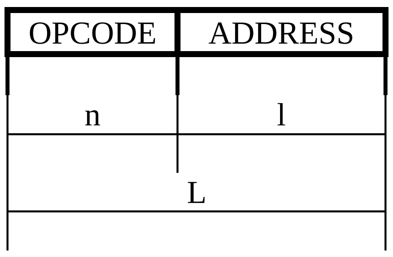
In order to implement the concept of virtual memory the next simplified scheme is used:



AL represents the logic address generated by the CPU when reading the current instruction. After processing the logic address inside the MMU it results the physical address. MMU realizes another translation function $f_T^* : AL \rightarrow AE$. f_T^* realizes translation between logic addresses (virtual ones) and physical addresses (the real ones). In case the logic address has no physical correspondent in the main memory then transfer commands (CT) are generated, for transfer between external (secondary) memory and main memory.

The practical implementation of the virtual memory concept is based on the transfer of blocks of information between secondary and main memories. The most widespread method is memory segmentation.

It has been already shown that each instruction at machine level comprises two main fields:



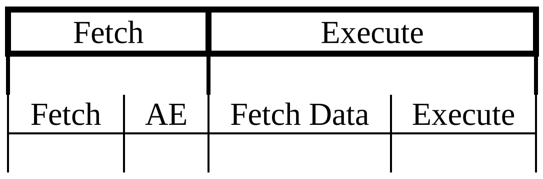
The address field comprises those constitutive elements of the logic address; processing this logic address one finds the effective or physical address. In case of the effective address:

AE = f(AL)
where:
AL – logic address;
AE – effective address

Besides these constitutive elements in the address field forming the logic address, there may exist other elements stored in CPU registers or memory.

In modern computers usually the effective and logic addresses are not the same. Quite the opposite, this function is more and more complex. The purpose is to ensure greater flexibility in programming, as well as shortening the length of the programs. It is also envisaged shortening the instructions length, even if the total volume of addressed memory (the memory space) is constantly increasing.

It has been shown that the instruction cycle comprises two main phases. Each of these phases is divided in sub-phases as in the next figure:



The second phase (AE) is important in the calculation of the operands effective addresses (determination of AE).

I. Assumed operand addressing

There exist situations in which the operand is assumed as known, based on the OPCODE, so it doesn’t need to be explicitly emphasized in the instruction, neither as value, nor as address.

For example:

- Increment instruction – the increment 1 is assumed as already known, and if this instruction is executed then the value of the accumulator register is increased with 1.
- SHIFT instruction – one assumes the shift value is 1, there is no need for supplementary specifications.

II. Implicit addressing

It has been presented that in the case of instructions with one or two addresses there is a special register in the CPU, the accumulator, which is used when performing the operations. This is because of the way the computer itself is designed and built. It is not necessary to specify it in any other way.

OPCODE	ADR1	ADR2

ACC ← (ADR1)*(ADR2)

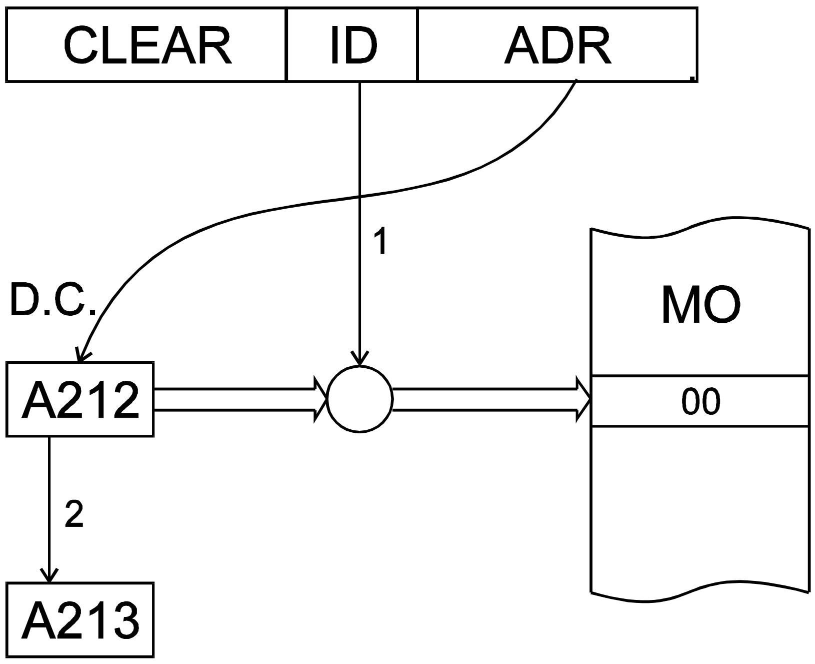
OPCODE	ADR

ACC ← (ADR)*(ACC)

Moreover, in modern CPUs, one can designate certain general registers to store an operand’s address. These general registers are generally referred to as DATA COUNTER. When designing a CPU, the implicit addressing is considered in the following manner: the operand is read from memory from the address specified by the Data Counter, this register acts as a counter with increment/decrement facilities. This way, it allows addressing of a whole block of data. One may notice implicit addressing through self-incrementation and implicit addressing through self-decrementation. Additionally, a load operation must be implemented, in order to be able to load the start address of the data block.

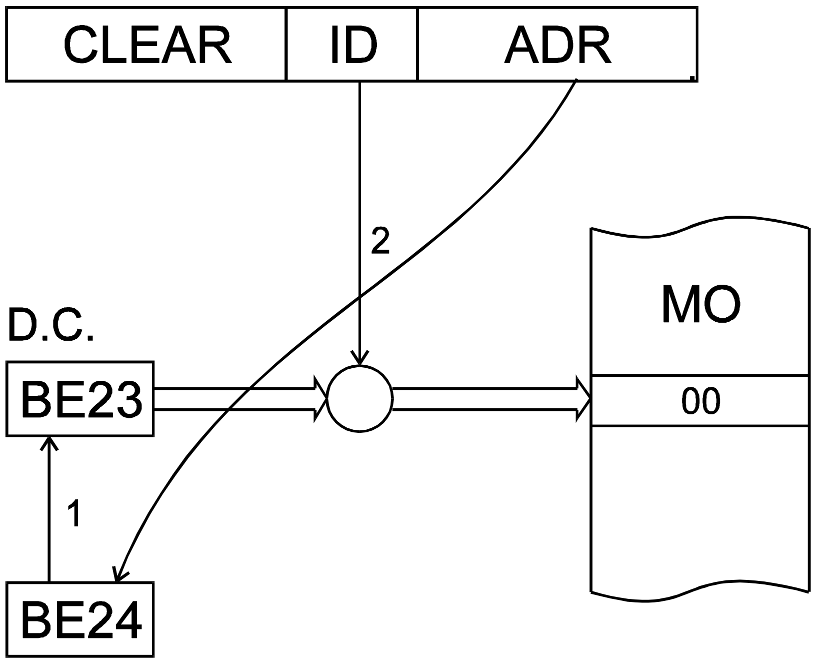
One can notice two variants in which the operand read operation is performed:

- Before incrementation/decrementation; incrementation/decrementation post-operation;
- After incrementation/decrementation; incrementation/decrementation pre-operation.



Implicit addressing with post operation incrementation

This is an implicit addressing with post operation incrementation.
Step no 1 – the location A212 is addressed, and a CLEAR is performed
Step no 2 – the Data Counter is incremented, thus having the value A213.



Implicit addressing with pre operation decrementation

This is an implicit addressing with pre operation decrementation.
Step no 1 – the DC content is modified/decremented, to the value BE23
Step no 2 – a CLEAR operation is performed at the address BE23.

Remark: There are CPUs that can define simultaneously several DCs, therefore it is possible to process several data blocks.

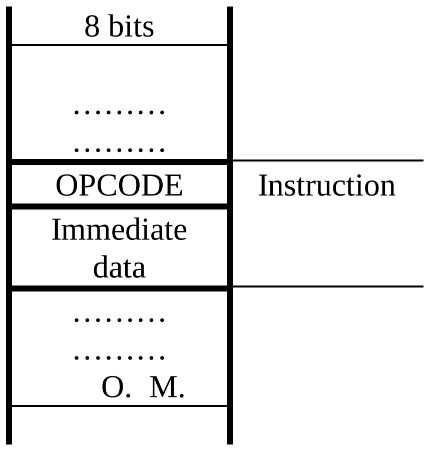
III. Immediate addressing

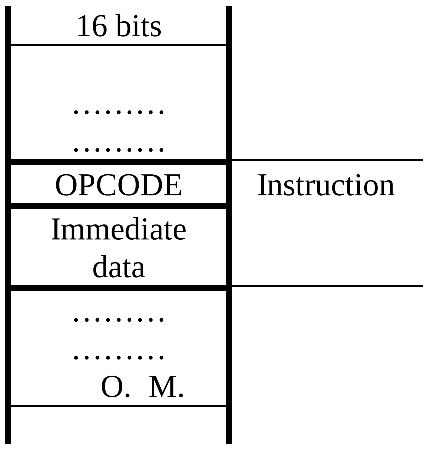
This addressing mechanism violates the general addressing principle; the operand is not separately stored but is an integral part of the instruction. This type of operand is called immediate data.

OPCODE	ID	OPERAND

Considering that there is the OPCODE field in the instruction format, it results that the length of the operand is shorter than the standard operand length, but the main advantage is that this data is immediately available after the **fetch instruction** phase, the **fetch data** phase being no longer necessary. Each **fetch** implies a memory read, a time consuming operation, but the execution is faster when the data is available together with the instruction itself.

In case of minicomputers with 16 bit words, the immediate data that comes with the OPCODE is a 16 bit word (first the OPCODE and then the data).





IV. Direct addressing

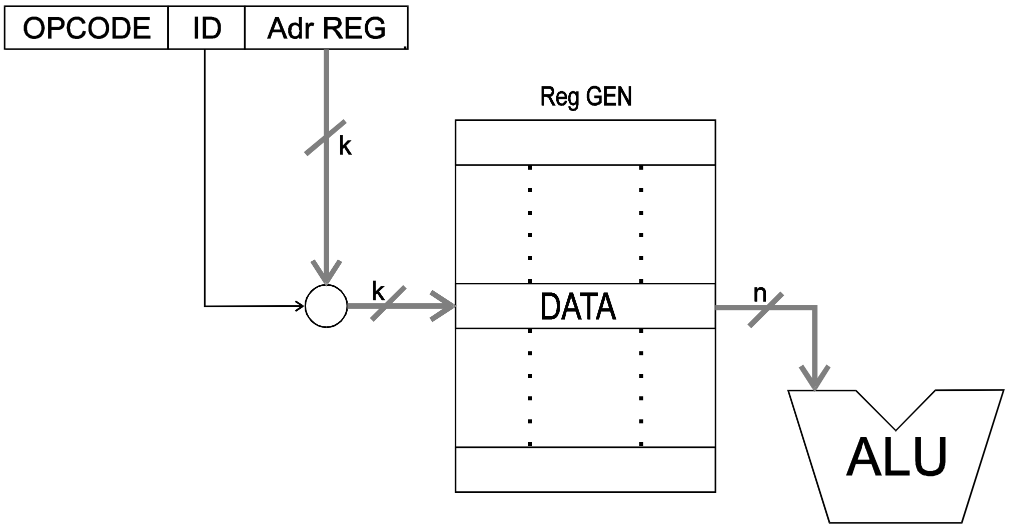
The direct addressing represents the natural memory addressing mode: the logic address coincides with the effective address.

One can notice the following direct addressing sub classes:
1. direct addressing to a register;
2. direct addressing to memory;
3. direct addressing to an input-output device;
4. combined direct addressing.

1. Direct addressing to register

All modern processors contain a certain number of general registers with very fast access, whose reunion forms the ***local memory***. The current data to be processed is brought into these registers. Because the number of these registers is limited (small), the register address is also small (*important advantage*).

On the other hand the access to data is much faster, because the memory read cycle is not started (because data isn't read from main memory). ***Reading data from these local registers*** / local memory ***is performed extremely fast***, due to the fact that they are CPU components.

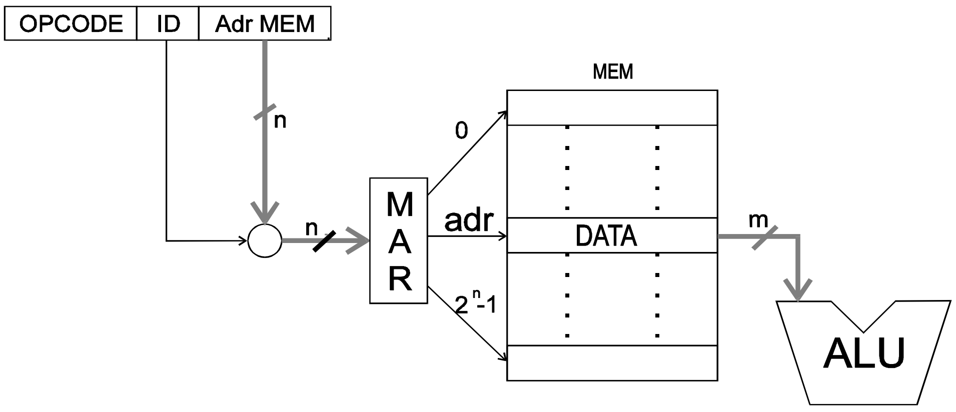


Direct addressing to register

2. Direct addressing to memory

In this case, the operand (data) is located in the main memory, in the data. Addressing the main memory is done by means of ***MAR*** (memory address register). If the memory has $\mathbf{2^n}$ locations then the address field must be on **n** bits.

Because the modern memory modules are getting larger with each day, it results that the number of ranks used for address is steadily increasing also, therefore the instructions whose operands are in memory and are retrieved in this manner necessitate a large number of bits (*longer length*).

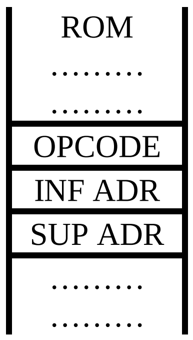


Direct addressing to memory

Executing such an instruction assumes transfer of the logic address from the address field into the MAR register, start of a memory read cycle and extraction of the operand from the addressed location.

In case of the 8 bit microcomputers such an instruction has 3 bytes:

- the first byte is the OPCODE;
- the second byte is the inferior half of the address;
- the third byte is the superior half of the address.



After reading the first byte (OPCODE) the control unit decodes the instruction and thus knows it is a 3 byte instruction with direct addressing to memory.

3. Direct addressing to input-output devices
This category of instructions is very diverse and should contain the following fields:

OPCODE	ID	Channel address	Peripheral device address

4. Combined addressing

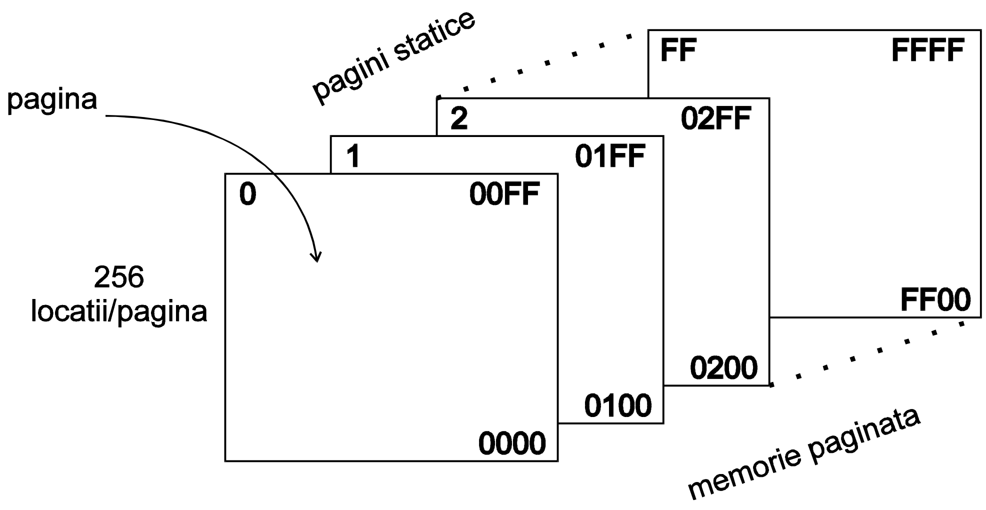
There are situations when one operand is located in the memory and the second in a general register so the instruction must contain both addresses. It is a combination of the first two cases. If both operands would be in the memory the instruction would be very long (*a very inconvenient case*).

V. Paged addressing

Paged addressing is a variant of the direct. It was created out of the need to shorten the instruction length. It was shown that in case of addressing the memory the address field is long (as the memory volume increases the address field is longer).

There were searched solutions to shorten the address field. One efficient solution is ***dividing the whole memory space in memory pages***. This division is only conceptual and simplifies memory addressing.

Each page has a fixed number of locations, and the number of pages depends on the global volume of the memory and size of a page. The size of a page is linked to the length of the address field in the instruction. If the address field has k bites then the recommended size of a page is 2^k locations.



Example of memory organisation on pages

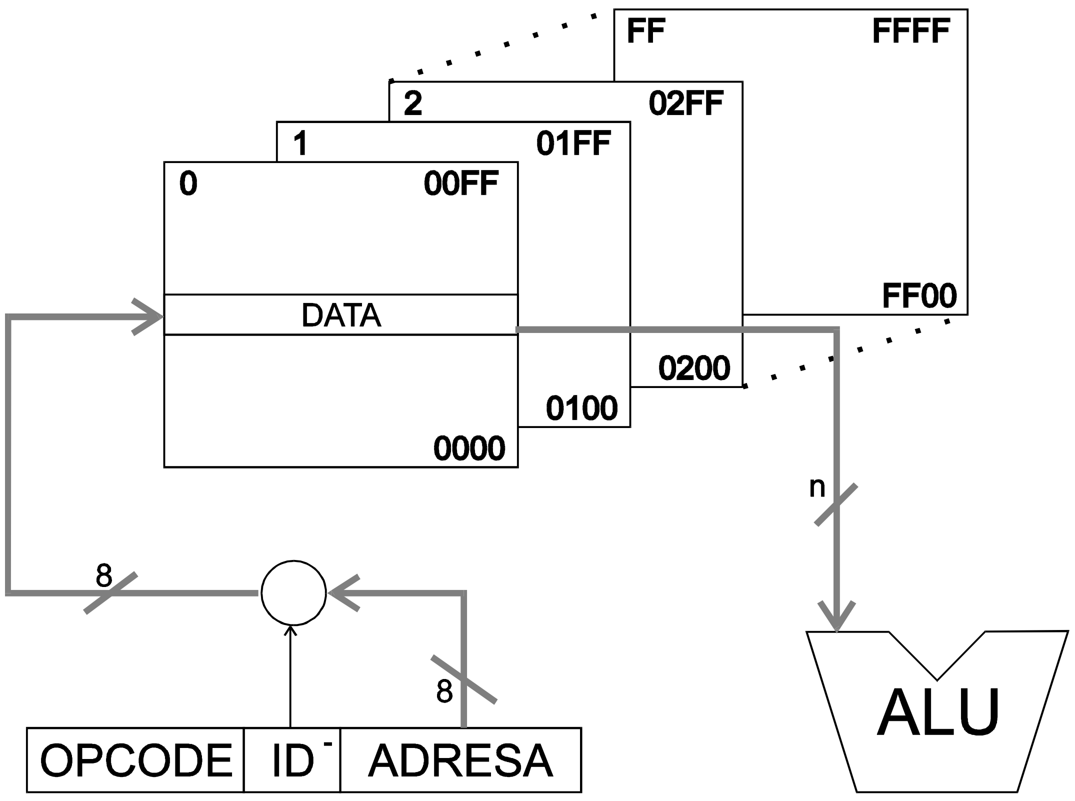
Paged organisation of the memory is used in virtual memory as well. The pages can be located at fixed addresses in case of a fixed page or at mobile addresses in case of a dynamic page.

Example: it is considered an instruction with an 8 bit address field; then a page will contain 256 locations.

1. Direct addressing to page zero

It is used the address field on **8** bits (in general **k** bits) of the instruction to address the **256** locations, but only in page zero. The selected location is read and the operand transferred into the ALU.

The advantage of addressing into the page zero is a small length of the. Instead of using 16 bits for the address (in general **n** bits) one uses only 8 bits (in general **k** bits where ***n*** > ***k***).

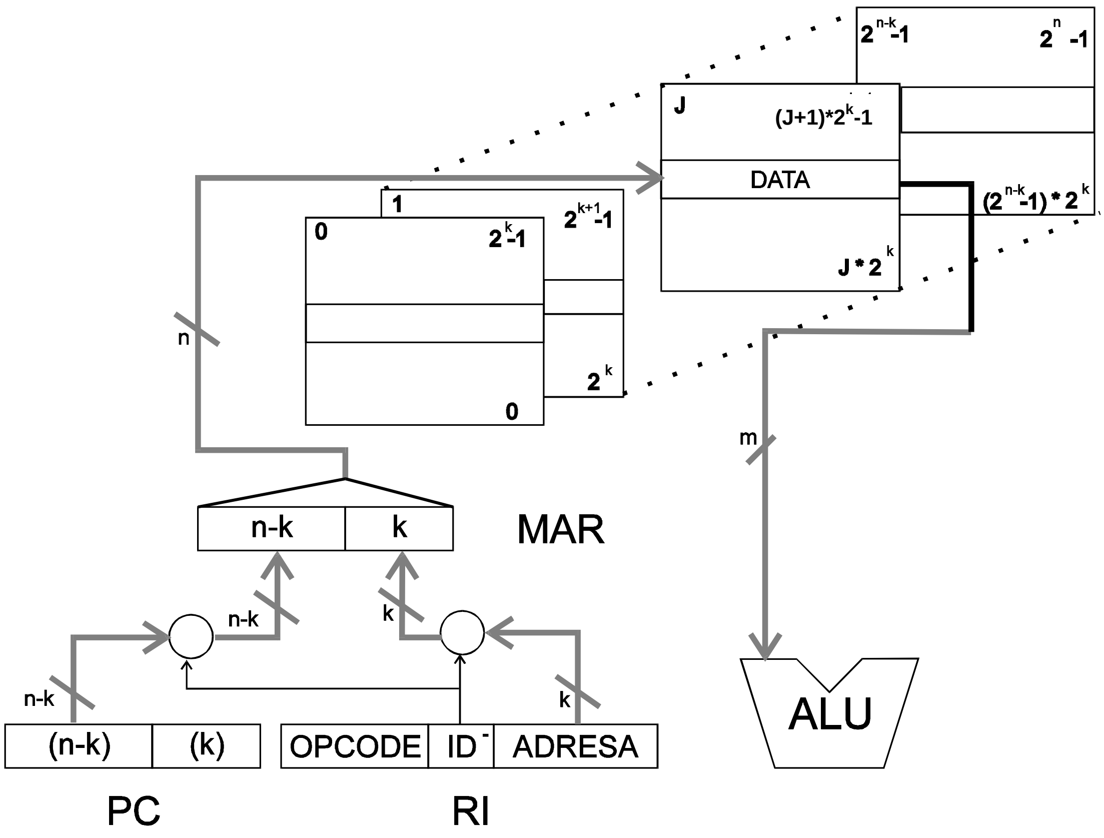


Direct addressing to page zero

The main disadvantage is the access only to this page, with no possibility to access another page.

2. Addressing the current page

In order to eliminate the disadvantages presented earlier, and to be able to address all pages, a special mechanism will be used, combining a component of the instruction, the **logic address** and a component from a CPU register, namely the **PC**. It is assumed that the address has **n** bits for a total volume of addressable locations of $\mathbf{2^n}$. Then the memory is organised in pages of 2^k locations, for a total of 2^{n-k} pages. The least significant part of the address formed of k bits allows addressing a location inside a page and the most significant part of the address on n-k bits allows determination of the right page. The n-k bits are taken from the most significant n-k bits of the program counter. Combining the two parts is done through concatenation. The newly obtained address is placed in the MAR which is on n bits and then the desired location is read (a certain page, and a certain location).



Addressing the current page

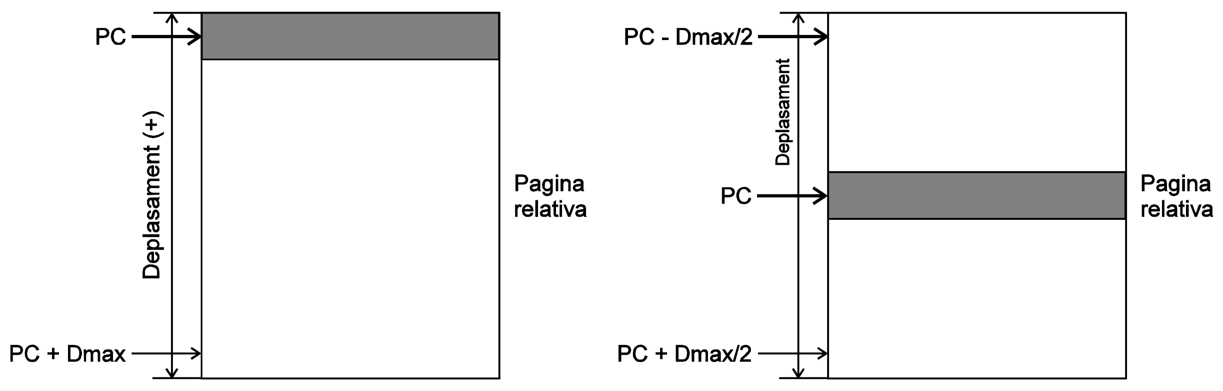
From the way the address is formed it results that ***the data required by an instruction has to be located in the vicinity of that instruction***. This vicinity is materialised by conserving the same page. So, when writing the program, and if using the same addressing technique, the programmer will place the data in the same memory page, because the page address is taken from the PC. When executing the current instruction, the PC has a certain value, out of which the most significant (n-k) bits are taken to form the address for the page containing the operand.

There can result critical situations at the frontier between. It is known that after fetch instruction it is issued the command to increment the PC. If for instance the current instruction is located at the address 01FF (k=2, n - k = 2), then after incrementation 01FF + 1 = 0200 and the page address changes from 01 to 02.

3. Relative paged addressing

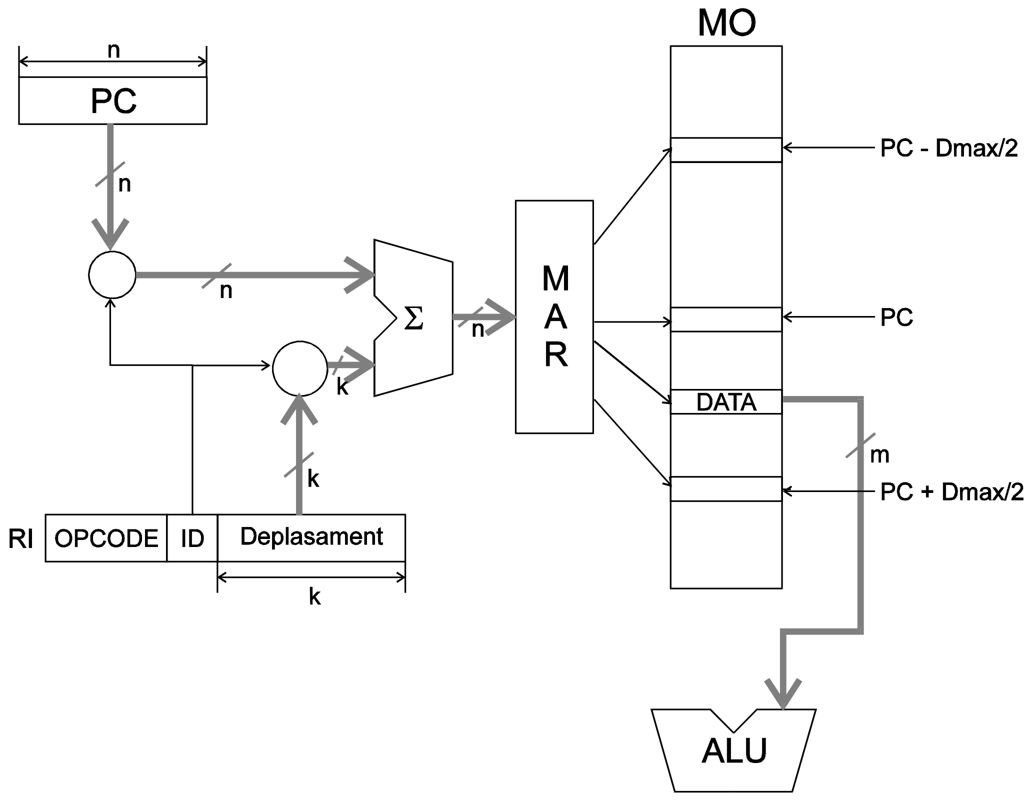
The notion of current page presented above is replaced in this case with the notion of relative page. This relative page is dynamic in the memory space. It is called relative because it is centred on the current value of the PC. So, in this addressing technique the PC is used with all its ranks. In the instruction the previously presented address field (the logic address) is now called **offset**.

This offset represents the value with which the PC is added to find the operand. The offset can be a positive binary number or a binary number with sign. The effective address is the sum of the PC an the offset and the result is placed in MAR. If the offset is positive then the base address of the relative page is the address contained in the PC and the top is PC + D_{max}, where D_{max} is the maximum possible value of the offset. In case the offset is a binary number with sign then the relative page is centred around the location with the address given by the PC.



Page types

The PC has n ranks, while the offset has k ranks (k<n); the addition is performed on n ranks.



Relative page addressing

Example: It is considered an instruction already brought in the RI (instruction register) having an OPCODE, an identifier and an offset = 2A, the address is written is hexadecimal digits, PC = 1A00. It is performed a sum between the PC and the offset, thus resulting the effective address.

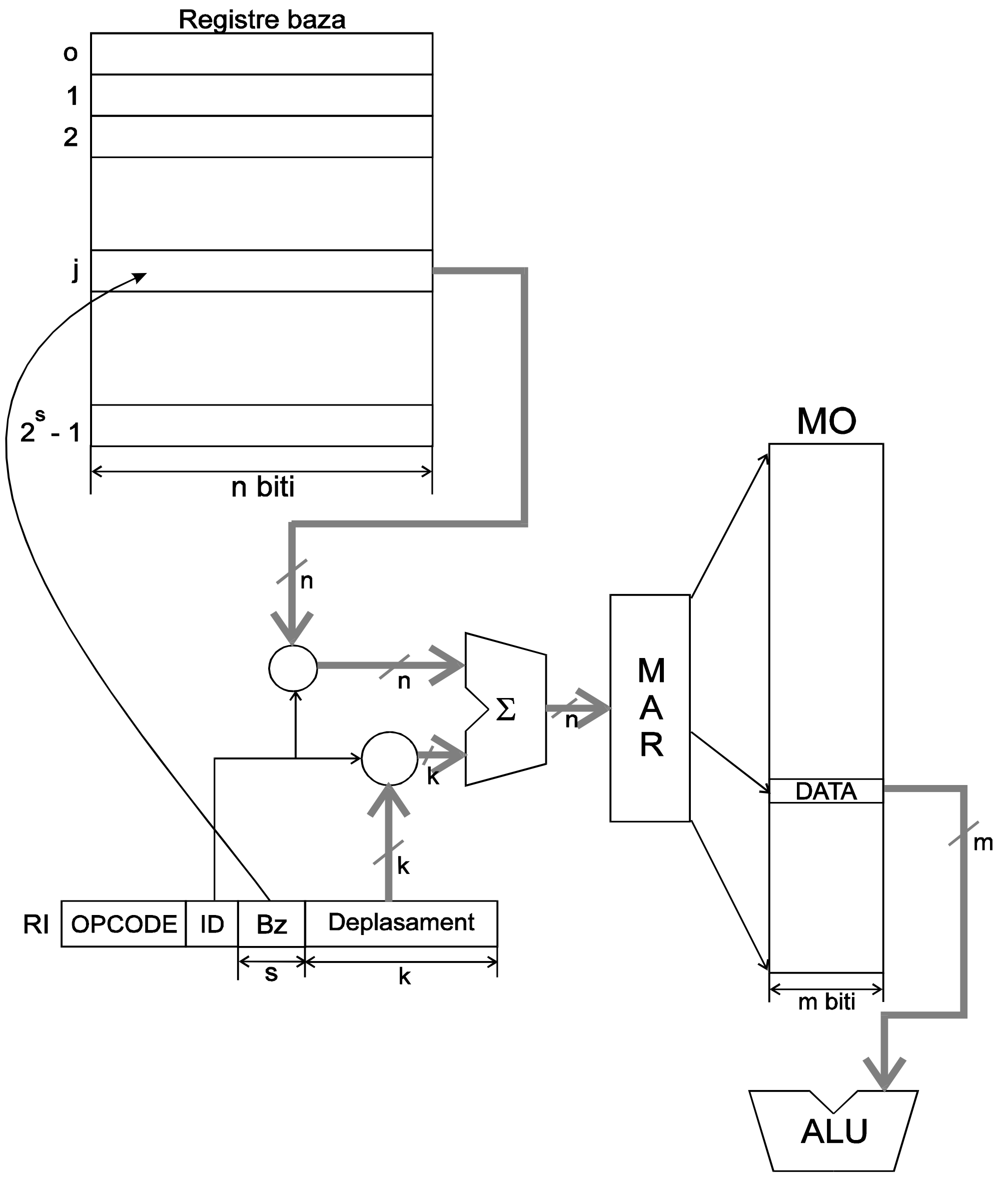
VI. Based addressing

It resembles as mechanism the relative page addressing, except that instead of the PC, it uses the content of another register or several registers called base registers; these registers are part of the processor general registers group.

In this way it is eliminated the dependency between the data location and the program. The base register has the same length as the MAR, and in the instruction, in the address field it is placed the offset, which can be a positive binary number or a signed binary number. The length of the offset is smaller than that of the MAR. The length of the offset gives the width of the window, so a larger number of ranks in the address field yields a larger window (a larger page).

Calculating the effective address is done by summing the content of the base register with the offset. But because usually there are several base registers, a special field must appear in the instruction, called field B, whose value points to the desired base register.

If for instance there are 8 base registers numbered 0 ... 7, then the B field must have the width 3. In some computer architectures it is adopted the convention that if B = 0 then the effective address is given solely by the offset.



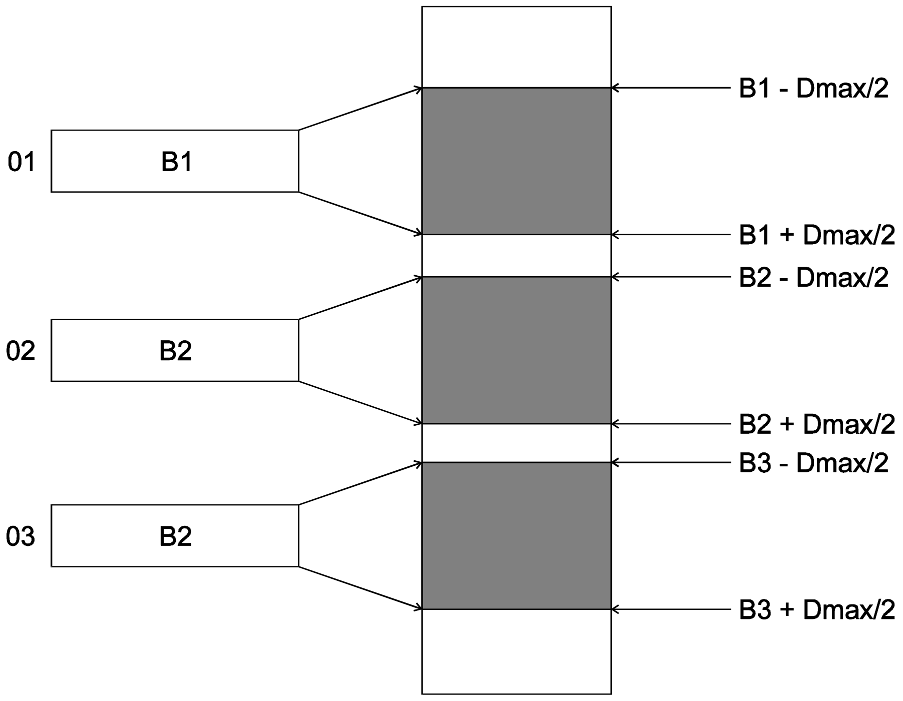
Based addressing

Loading initial values in the base register and modifying them during the execution of the program is done by means of software tools.

Definition: This addressing technique is called addressing through base and offset.

The content of the designated base register is either the centre of the page containing the data or the beginning of this page.

Because several base registers can be used it results that there can be simultaneously defined several current pages for the data. When defining these pages special attention must be paid to avoid address overlapping.



VII. Indexed addressing

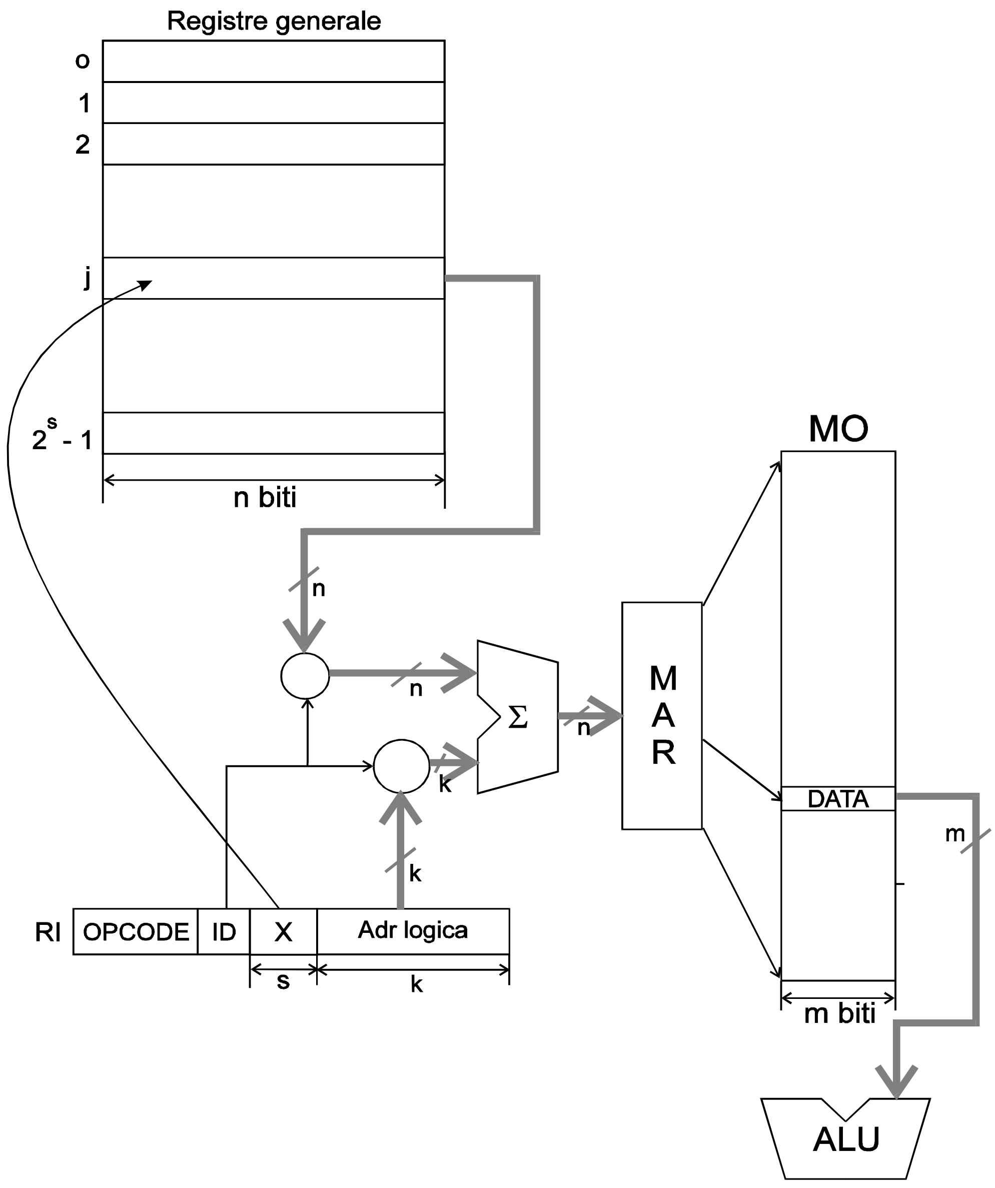
Indexed addressing is a variant of the based addressing. The base registers are replaced with index registers. These registers are part of the processor general registers group. The effective address is obtained in a similar manner as in based addressing: a summing is performed between the content of the index registers and the logic address from the instruction.

Definition. The content of the index register is called index.

What is particular to these index registers is the ability to perform a series of direct operations on them such as: increment, decrement, parallel load, addition, subtraction.

The index registers are frequently used in computer programming, especially in defining cycles. One can simultaneously define several indexes, allowing thus declaration of several cycles and thus nested cycles are possible.

In the X field it is introduced the address of the index register. If there are s ranks then there can be 2^s index registers. In the ID field (identification) there appear combinations specifying a certain addressing technique.



Indexed addressing

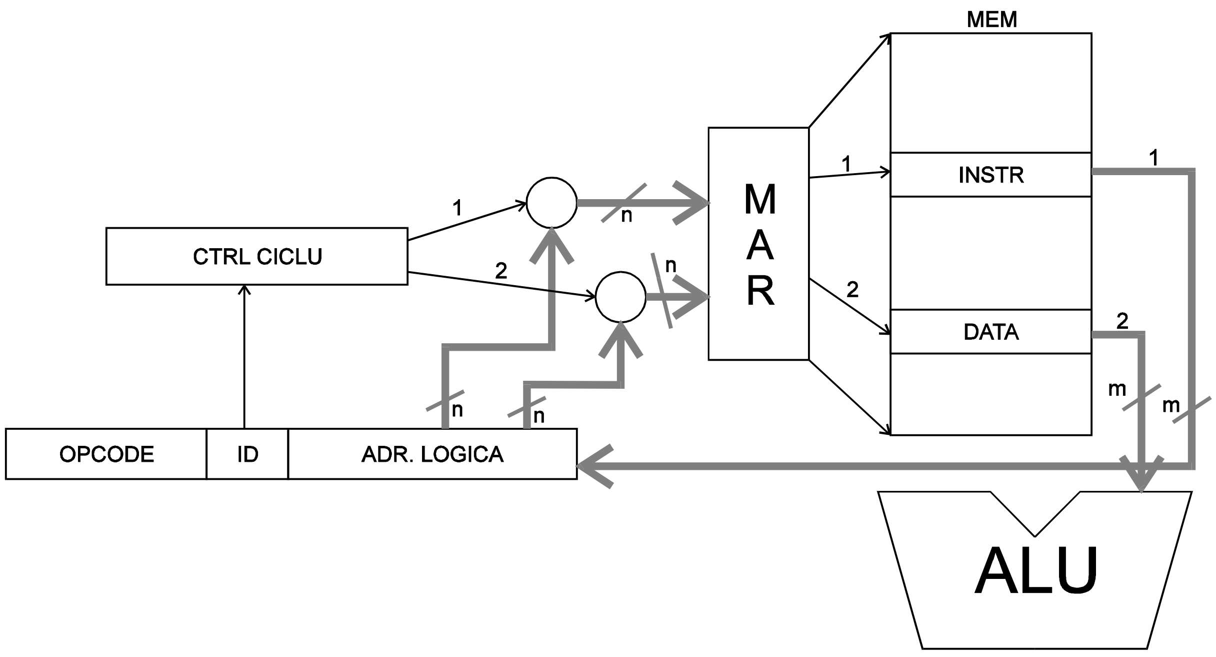
Example of indexed addressing usage: it is required the transfer of a data block with K components from the address area M_1 ... M_k to R_1...R_k. Usually, there should be written k transfer instructions like MOV. Programming is simplified if indexed addressing is used.

In the instructions in the address field the constants R_1 and M_1 appear, and one can perform increment operations on the index register. In the R_x index register the K quantity is introduced. On R_x one can perform decrementations. When the value of the index becomes zero then the data block transfer is complete.

VIII. Indirect addressing

Indirect addressing is a modern addressing technique that allows shortening the programmes length and introduces flexibility in writing complex programmes.

In indirect addressing the logic address doesn't lead us to the effective address of the operand but to the address of a new instruction. The memory location pointed by the address will be read and the data obtained will be interpreted as a new instruction.



Indirect addressing

In case of the microcomputers the indirect addressing is frequently performed as implicit addressing by means of general registers of the CPU. In case of INTEL I8080 the register pair HL, each on 8 bits, is used as logic address for accessing the operative (main) memory from where the operand is extracted. For instance there is the instruction ADD M (which performs an addition between ACC and the content of the memory location whose address is specified implicitly by the register pair HL content.

$$ACC \leftarrow (ACC) + Mem((H)(L))$$

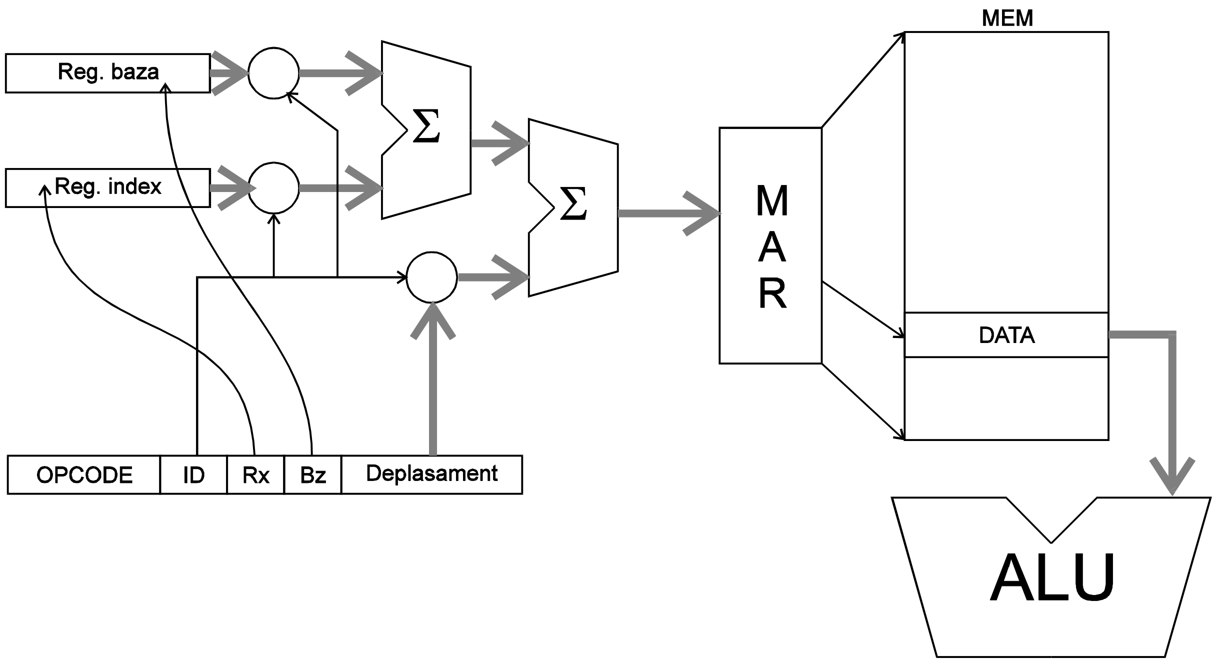
IX. Addressing for a block of data

It has been shown that a computer may operate with variable length fields al well (data blocks). To address them several options are possible:
1. It is given the start and end addresses for the block;
2. It is given the start address and the length of the field;
3. It is given the start address and it is used a block delimiter meaning a particular combination of bits used only for identifying the block end. This is not recommended as part of the memory is used to store the delimiter.

X. Combined addressing techniques

Most of the microcomputers and computers use various combinations of the previously presented addressing techniques, in order to take advantage of their benefits.

1. *Multilevel indirect addressing*. The indirect addressing is performed in several cycles; each time the information extracted from memory is not an operand but another instruction, except for the last read cycle. Usually there is a limit of levels. For instance in case of the FELIX C family of computers the maximum number of levels is 5.
2. *Indirect and indexed addressing*. In accordance with its name, this technique combines the two already presented techniques. The problem is when to perform the indexing. There are 2 cases: indirect with post-indexing and with pre-indexing. In the first situation, the indirect addressing is performed (it can be a multilevel indirect addressing) and at the end it is added the value of the index (from the index register specified in the instruction) thus resulting the effective address. In the second situation it is performed the indexed addressing mechanism at first, adding the logic address to the content of the index register resulting the address where the pointer is located. The pointer is extracted, a new addressing is performed and the operand is finally read.
3. *Based and indexed addressing*. The effective address calculation mechanism is the following:



Based-indexed addressing