Chapter 5

Elementary Educational Computer

§1. General structure of the Elementary Educational Computer (EEC)

- The EEC conforms to the 5 units structure defined by von Neumann's model (1.1)
- All units are presented in a simplified form consisting of only basic components. (1.2)
- Structure of the EEC presented in **Annex 5**. (1.3)

§2. Presentation of the EEC units

§2.1. Memory Unit (MU)

- One level memory consisting of the Main Memory (MM) (2.1.1)
- Every location identified by its own *address on k bits.* (2.1.2)
- Communication with other units through: (2.1.3)
  - a) **MAR**–Memory Address Register
  - b) **MBR**–Memory Buffer Register or Memory Data Register
- Organization of the memory: 2^k locations of *n* bits, thus **2^k ×n memory array** (2.1.4)
- Two operations are allowed: READ and WRITE, controlled by the Control Unit (2.1.5)
- Description of the READ cycle (2.1.6)
  1) Address placed in MAR
  2) READ control signal
  3) Extraction from the addressed location
  4) Store data in MBR

- Description of the WRITE cycle (2.1.7)
  1) Address placed in MAR
  2) Data transferred in MBR
  3) WRITE control signal
  4) Store data in the addressed location
- Types of WRITE and READ commands issued by the Control Unit: two independent(R,W) or one common ($R / \overline{W}$) (2.1.8)

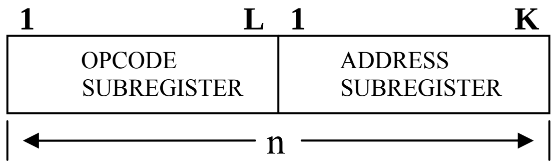
§2.2. Arithmetic and Logic Unit (ALU)

- Implements *binary arithmetic on n bits* (2.2.1)
- Dimension of ALU operational units is assumed n (2.2.2)
- All registers inside ALU are *n*-dimensional (2.2.3)
- ALU contains a simple register file and a processing device (2.2.4)
- Processing section consists of an Adder/Subtractor and a Shifter (2.2.5)
- Register file consists of an Accumulator, three auxiliary registers RX1, RX2, RX3 and a Flag (Status) register (FR) (2.2.6)
- ALU performs a limited set of primitive operations (2.2.7)
- Communication between ALU and CU: CU sends the commands via control lines, whereas ALU sends the status of the registers content (status signals, flags, condition signals), usually of the Accumulator. (2.2.8)
- Possible set of *status bits*: zero, parity, sign, overflow etc. (2.2.9)
- Operands are extracted either from register file (local memory) or from MM. Extraction from MM implies a READ cycle. (2.2.10)
- Role of the Accumulator: it is a special register communicating directly with the processing device, that contains one of the operands and where the result after processing is stored. (2.2.11)

- The Arithmetic and Logic operations performed in ALU are on one or two operands (monadic or diadic operations). (2.2.12)

§2.3. Control Unit (CU)

- The CU is formed of the following blocks: (2.3.1)
  1) Program Counter (**PC**), on *k* bits
  2) Instruction Register (**IR**), on *n* bits
  3) Function decoder (**DEC** L/2^L)
  4) **Control Block** (Logic Sequencer, Control Sequencer)
- Program Counter contains a memory address where the next instruction to be executed is stored; since the addressing space of MM is 2^k , the dimension of PC is *k* (identical with the dimension of MAR). (2.3.2)
- PC has the incrementing facility, as well as a parallel load facility. (2.3.3)
- Instruction Register (IR) contains the current instruction which is in execution. (2.3.4)
- The width of IR coincides with the width of an instruction and in the case of EEC it is *n*. (2.3.5)
- IR is divided in two subregisters according to the format of the instruction. (2.3.6)
- Structure of the IR: (2.3.7)



- The OPCODE subregister communicates with the function decoder to interpret the current instruction (to decide which function must be executed) (2.3.8)

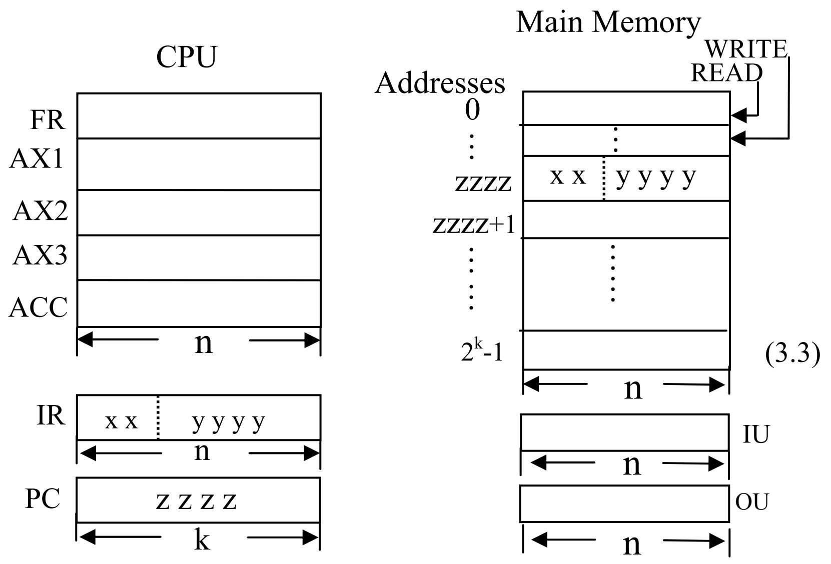
- The Address subregister contains an address of the MM where one operand is stored. In case of two operands operation it is assumed that the other operand is in the Accumulator. (2.3.9)
- For reason of simplicity, there are missing the Function Register and the Address Register. Also, the address field contains always the *effective address* of the operand (not the logical address). (2.3.10)
- The central role in the CU is played by the Control Block (Control Sequencer), which generates the control signals for the other units according to the operation (function) to be executed. (2.3.11)
- The inputs in the Control Block are the decoded (interpreted) function, master clock (from a Clock Generator) and status flags (from ALU). (2.3.12)
- Control Block is a *complex sequential machine*, that is why it is also called Control Sequencer. (2.3.13)

§2.4. Input/Output units (I/O)

- In case of EEC the I/O system is composed of simple Input/Output devices, at the lowest level a register on *n* bits. They are communicating with the ALU (Accumulator) and MM (MBR), as well as with the Control Unit (Instruction Register). (2.3.14)

§3. The register structure of the EEC

- Any digital system can be viewed as a *union of generalized registers and the data paths interconnecting* them. Even the memory formed of 2^k locations can be considered as formed of 2^k registers, as each location is an *n*-bit register. (3.1)
- By merging ALU with CU, to form the CPU, the entire structure of the EEC can be reduced to the following set of registers (3.3): (3.2)



(3.3)

Where:

- ➢ FR = flag register (status register), on *n* bits
- ➢ ACC = Accumulator, on *n* bits
- ➢ AX1, AX2, AX3 = auxiliary registers, on *n* bits
- ➢ IR = Instruction Register, on *n* bits
- ➢ xx = the opcode field of the instruction, on *L* bits
- ➢ yyyy = the address field of the instruction, on *k* bits
- ➢ PC = Program Counter, on *k* bits
- ➢ IU = Input unit, on *n* bits
- ➢ OU = Output unit, on *n* bits

(3.4)

- This register view of the EEC is useful for explaining the flow of operations that take place for execution of instructions. (3.5)

§4. Mode of operation

§4.1 General considerations

- According to the von Neumann's principles, both instructions and data are located in memory, in binary coded form. (4.1.1)
- Any instruction is executed in two major phases
  - ➢ FETCH phase, consisting in extracting the current instruction from the memory and decoding the OPCODE field
  - ➢ EXECUTE phase, consisting in effective execution of the operation on the defined operands (data).

(4.1.2)

§4.2. FETCH phase

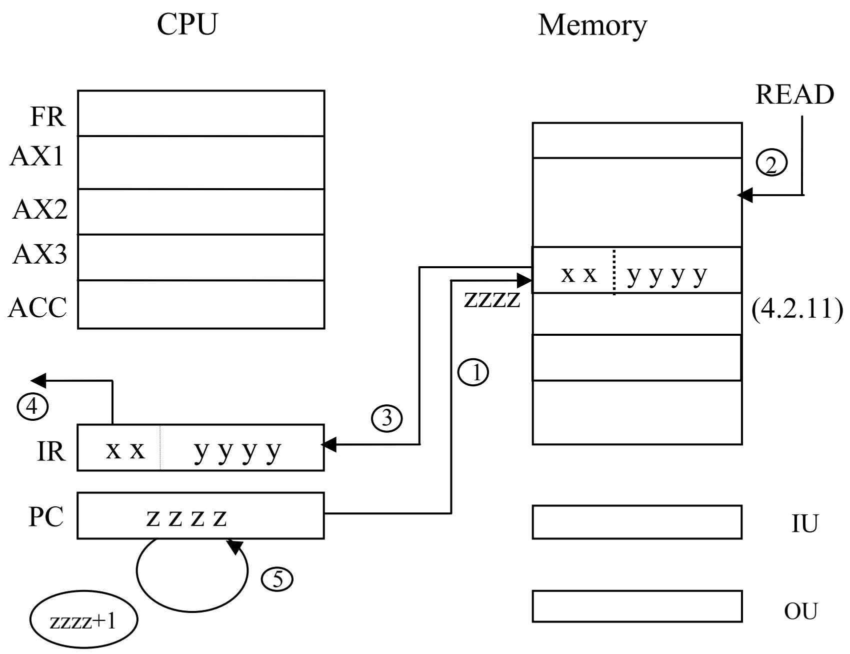
- The initial address of the first instruction to be executed is already stored in PC (4.2.1)
- The content of PC is transferred to MAR. (4.2.2)
- CU is issuing a READ command to MM and a read cycle is initiated. (4.2.3)
- The content of the read location, representing an Instruction, is transferred to MBR. (4.2.4)
- From MBR the instruction is transferred to IR from CU. (4.2.5)
- The subregister containing OPCODE, on *L* bits, is transferred to the Function Decoder. (4.2.6)
- Function Decoder decodes the OPCODE and informs the Control Block of the CU, which, in turn, issues the appropriate control signals to the other units. (4.2.7)
- CU is incrementing the PC to point to the next instruction. (4.2.8)

- In a simplified RTL (Register Transfer Language) the FETCH phase can be described in the following form:
    1. MAR ← (PC)
    2. READ
    3. IR ← (MBR)
    4. DEC ← $(IR)_{OPCODE}$
    5. PC ← (PC)+1
    6. Go to EXECUTE phase

  where:
    - $(IR)_{OPCODE}$ means the content of the OPCODE subregister of the IR;
    - (PC) means the content of the PC;
    - (MBR) means the content of the MBR.

  (4.2.9)

- Schematically, this phase can be represented in the register view of the EEC as follows (4.2.11) (4.2.10)



(4.2.11)

§4.3. EXECUTE phase

- As stated before, the FETCH phase is common for all Instructions, whereas EXECUTE phase is specific for each kind of Instruction. (4.3.1)
- In what follows there are described extensively several simple Instructions that are executed in EEC. (4.3.2)

A) ADD, address

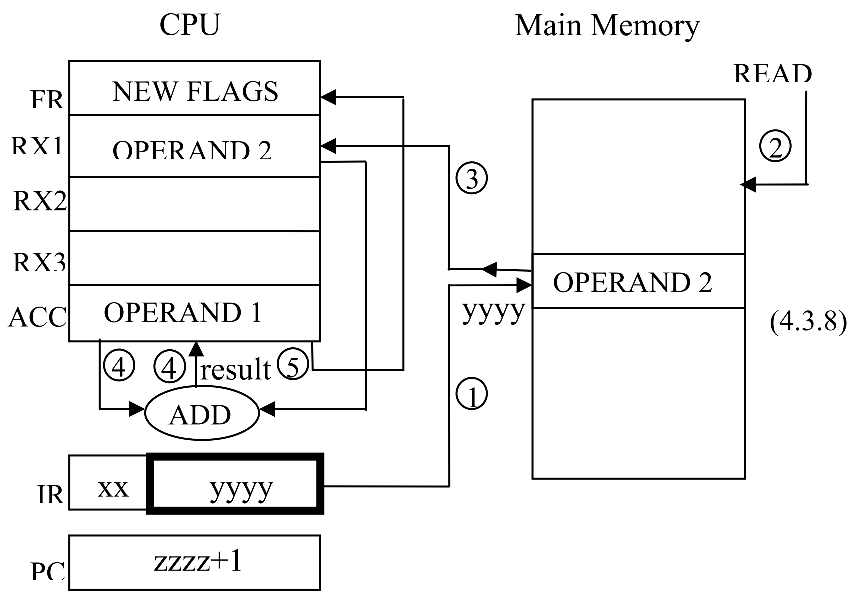
- This represents the addition of two operands Instruction, where the first operand is in the Accumulator, the second operand is in the memory at the address (yyyy), while the sum is saved in the Accumulator. In a symbolic manner this operation can be described as follows: (4.3.3)

  ACC ← (ACC) + $(Memory)_{ADDRESS}$
- The address (yyyy) of the second operand is given in the Instruction being stored in the $(IR)_{ADDRESS}$ subregister. (4.3.4)
- The entire operation takes place in the following steps:
    1. Transfer the address field from $(IR)_{ADDRESS}$ into MAR, which means transfer yyyy into MAR.
    2. Initiate a READ operation from the location having the address yyyy.
    3. Transfer of the extracted operand into the ALU, in register RX1.
    4. Perform the addition between the contents of ACC and AX1, then store the result in the Accumulator
    5. Change the corresponding flags from the FR.
    6. Go to the next FETCH phase

  (4.3.5)
- In RTL notation:
    1. MAR ← $(IR)_{ADDRESS}$
    2. READ
    3. (RX1) ← (Memory) $_{ADDRESS}$
    4. ACC ← (ACC) + (RX1)
    5. FR ← New flags
    6. Go to FETCH phase

  (4.3.6)

● In the register view of the EEC the realization of ADD Instruction is presented in (4.3.8) } (4.3.7)



(4.3.8)

B) SUB, address

● This represents the subtraction operation, where the subtrahend, that is the first operand, is in the Accumulator, while the minuend, that is the second operand, is in the memory at the address specified explicitly in the instruction. } (4.3.9)

● The realization of the subtraction assumes reading from the memory of the second operand and transferring it into the ALU, in the register RX1. After that, the subtraction takes place in the processing device and the difference is saved in ACC, accompanied by the corresponding changes of flags in the FR. } (4.3.10)

● Symbolically:

$$\text{ACC} \longleftarrow (\text{ACC}) - (\text{Memory})_{\text{ADDRESS}}$$ } (4.3.11)

● The entire operation takes place in the following steps:

1. Transfer the address field from (IR) $_{\text{ADDRESS}}$ into MAR, which means transfer of yyyy into MAR.
2. Initiate a READ cycle, to extract the content of the location having the address yyyy
3. Transfer of the extracted operand in ALU, in the register RX1.
4. The subtraction operation takes place in the processing device by subtracting the content of RX1 from the content of ACC; the difference is saved in the Accumulator.
5. Change the appropriate status bits in FR
6. Go to the next FETCH phase.

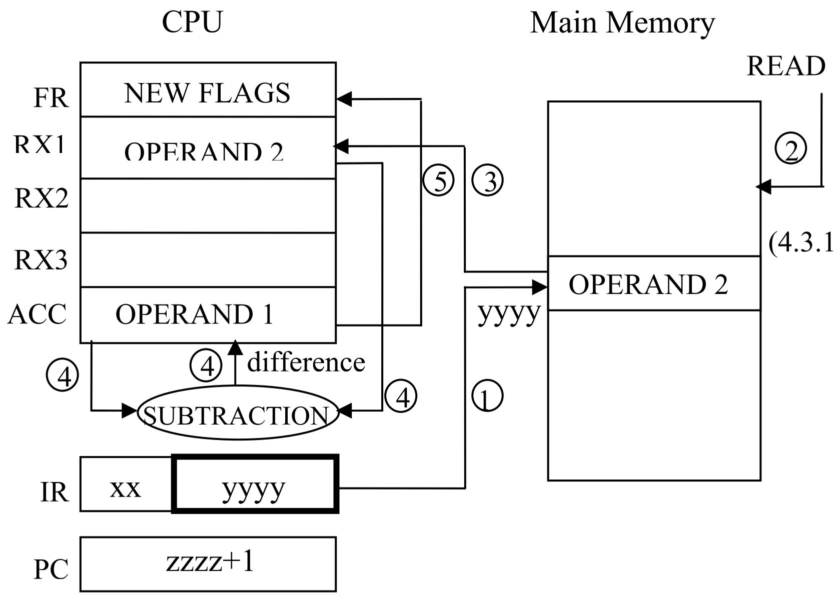
} (4.3.12)

● In RTL notation:

1. MAR $\longleftarrow$ (IR) $_{\text{ADDRESS}}$
2. READ
3. RX1 $\longleftarrow$ (Memory) $_{\text{ADDRESS}}$
4. ACC $\longleftarrow$ (ACC) $-$ (RX1)
5. FR $\longleftarrow$ New Flags
6. Go to FETCH phase

} (4.3.13)

● In the register view the execution of SUB instruction is represented in the next figure (4.3.15) } (4.3.14)

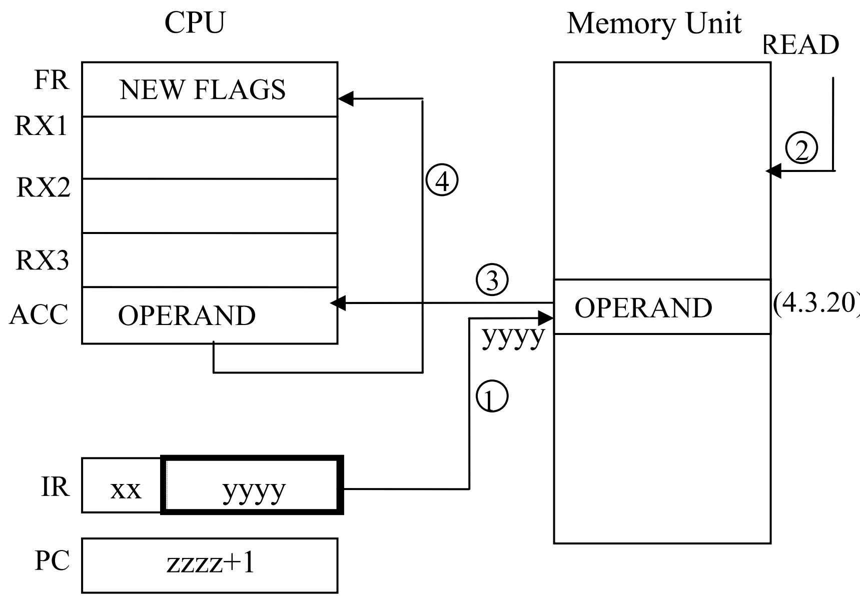


(4.3.15)

C) LOAD, address

- The LOAD Instruction ensures reading of an operand from the memory at the address (yyyy) specified in the instruction and transferring it into the Accumulator. (4.3.16)
- Symbolically: (4.3.17)

  ACC $\longleftarrow$ $(\text{Memory})_{\text{ADDRESS}}$
- The entire operation takes place in the following steps: (4.3.18)
  1. Transfer the address field from $(\text{IR})_{\text{ADDRESS}}$ into MAR, which means transfer yyyy into MAR.
  2. Initiate a READ operation from the location with the address yyyy.
  3. Transfer the extracted operand into the ALU, in the Accumulator and change the flags from FR.
  4. Go to the next FETCH phase.
- In the register view of the EEC the realization of LOAD Instruction is presented in the next figure (4.3.20): (4.3.19)



(4.3.20)

- *Observation*: if in the OPCODE there is provided a subfield specifying the destination register from the ALU, then there can be defined variants of the LOAD instruction: (4.3.21)

  RX1 $\longleftarrow$ $(\text{Memory})_{\text{ADDRESS}}$

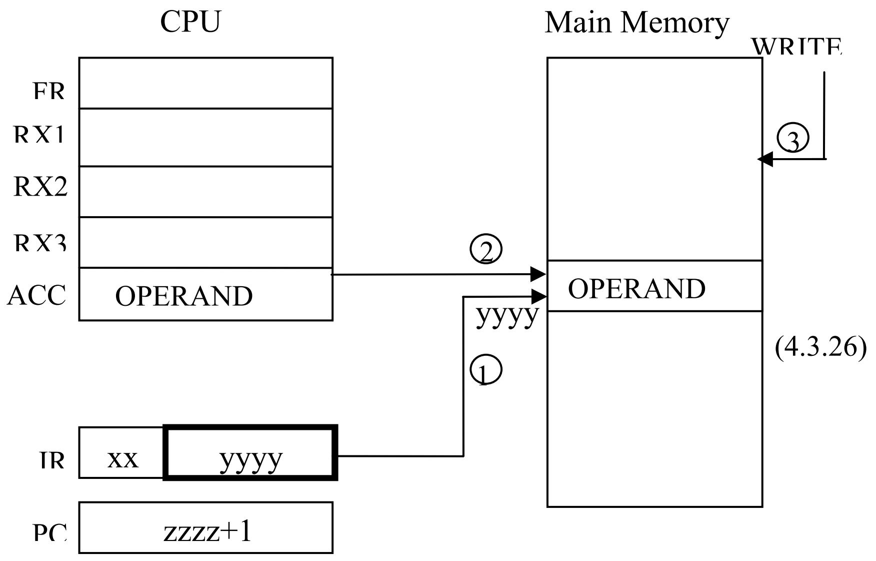
  RX2 $\longleftarrow$ $(\text{Memory})_{\text{ADDRESS}}$

  RX3 $\longleftarrow$ $(\text{Memory})_{\text{ADDRESS}}$

D) STORE, address

- The STORE Instruction ensures the transfer of the content of the Accumulator into the memory and storing it in the location having the address (yyyy) given in the instruction. (4.3.22)
- Symbolically: (4.3.23)

  $\text{Memory}_{\text{ADDRESS}}$ $\longleftarrow$ (ACC)
- The entire operation takes place in the following steps: (4.3.24)
  1. Transfer the content of the $(\text{IR})_{\text{ADDRESS}}$ into the MAR; the content of MAR becomes yyyy.
  2. Transfer the content of the Accumulator into the MBR; in this way the operand is prepared for further storing in the memory.
  3. Initiate a WRITE operation, realising storing of the content from the MAR into the location with the address yyyy.
  4. Go to the next FETCH phase.
- In the register view of the EEC this instruction is represented in (4.3.26): (4.3.25)



(4.3.26)

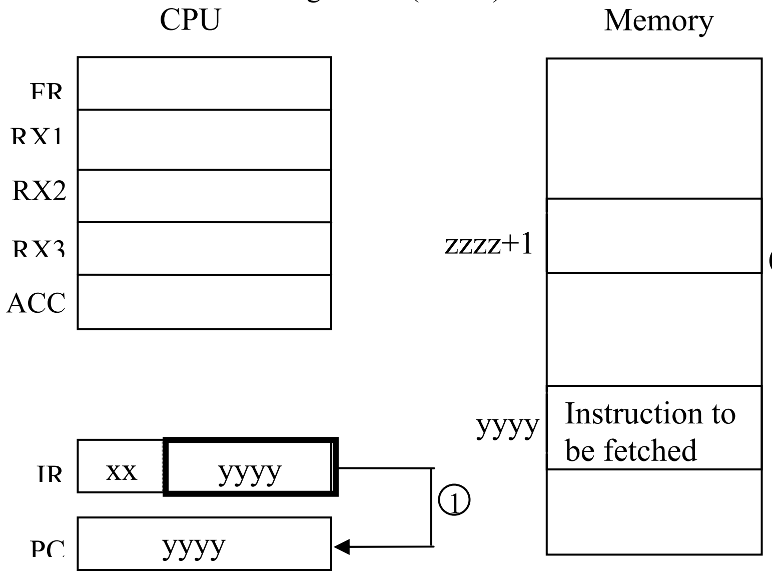
- *Observation*: The STORE Instruction can present variations by including in the OPCODE a subfield specifying the source register from ALU; in this way, the content of RX1, RX2 or RX3 can be stored in the memory at the specified address given in the Instruction (4.3.27)

$$(\text{Memory})_{\text{ADDRESS}} \longleftarrow (\text{RX1})$$
$$(\text{Memory})_{\text{ADDRESS}} \longleftarrow (\text{RX2})$$
$$(\text{Memory})_{\text{ADDRESS}} \longleftarrow (\text{RX3})$$

E) JUMP, address

- This is an unconditional JUMP at the address specified in the Instruction. The implementation is quite simple by transferring the address yyyy from the $(\text{IR})_{\text{ADDRESS}}$ into the PC. In this way, instead of using the address (zzzz+1), the address (yyyy) will be used in the next FETCH phase for extracting the next instruction from the memory. (4.3.28)
- The execution of this unconditional JUMP Instruction is very simple: (4.3.29)
  1. $(\text{PC}) \longleftarrow (\text{IR})_{\text{ADDRESS}}$
  2. Go to the next FETCH phase.

- In the register view of EEC representation the execution of this instruction is given in (4.3.30):



(4.3.30)

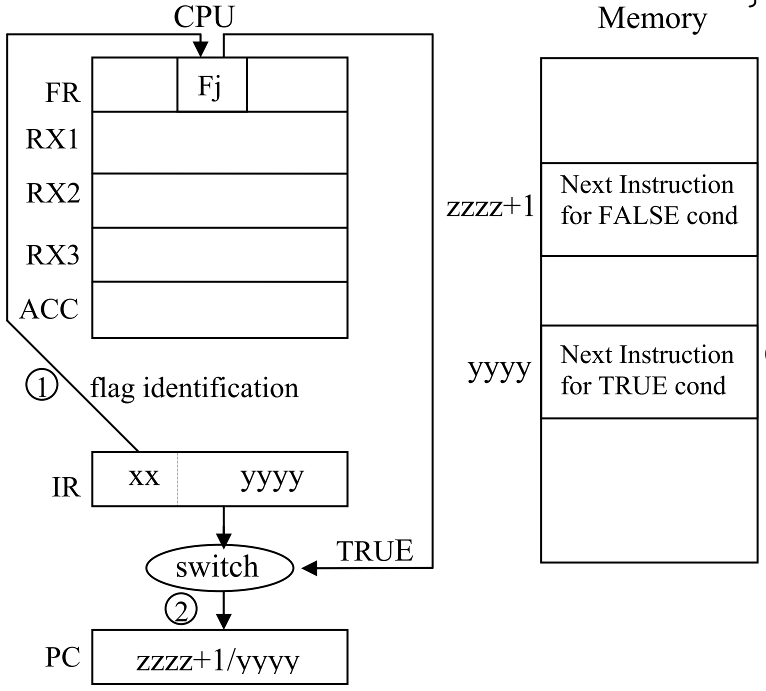
F) Conditional JUMP, address

- The conditional JUMP Instruction tests a condition and if it is true then a jump takes place at the given address in the Instruction; otherwise the normal flow of execution continues, that is the content of PC remains unaltered, so that the next FETCH will take place at the address (zzzz+1). (4.3.31)
- The test operation consists in checking a flag (a condition bit) from the Flag Register, FR. As mentioned before, among the usual flags the following are common: (4.3.32)

ZERO flag	– if the content of the Accumulator is 0
SIGN flag	– reproducing the most significant bit of the Accumulator (if it is 0, then a positive number is in the ACC, if it is 1, then a negative number is in the ACC)

PARITY flag – shows if the number of "1"s in the Accumulator is odd or even
CARRY flag – if after an addition/subtraction operation it was generated a carry from the most significant column. (4.3.32)

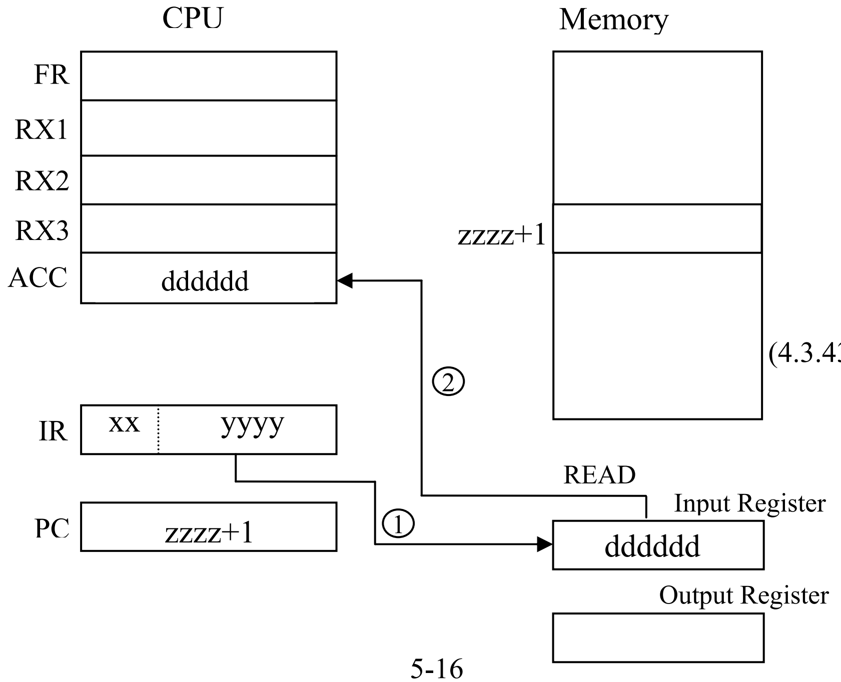
- The OPCODE for this conditional JUMP will contain a subfield to identify which flag is to be tested; thereby there are many Conditional JUMP Instructions depending on how many flags are defined in the architecture. (4.3.33)
- Symbolically, (4.3.34)
  If (condition) *go to* (address) *else* (zzzz+1)
- The steps of implementing these instructions are: (4.3.35)
  1. Test the flag defined by the OPCODE
  2. If the condition is TRUE then transfer the address from $(IR)_{ADDRESS}$ into the PC, and go to 4
  3. If the condition is FALSE then go to 4
  4. Go to next FETCH phase.
- In the register view of the EEC representation (4.3.37): (4.3.36)



(4.3.37)

G) INPUT, address

- The address field would specify one of the set of Input Registers representing the INPUT UNIT. In this particular case it was used a simple input register, therefore the address field is irrelevant. But, in general case, there are defined 2^k different addresses of input registers. (4.3.38)
- This Instruction reads the content of the addressed register and transfers it into the CPU, in the Accumulator. (4.3.39)
- Thus, symbolically: (4.3.40)
  ACC $\longleftarrow$ $(\text{Input Register})_{ADDRESS}$
- The steps of implementation:
1. Identify the Input Register from the address stored in $(IR)_{ADDRESS}$ (4.3.41)
2. READ the addressed Input Register and transfer its content into the ACC.
3. Go to next FETCH phase.
- In the register view of the EEC the execution of this instruction is presented in (4.3.43): (4.4.42)



(4.3.43)

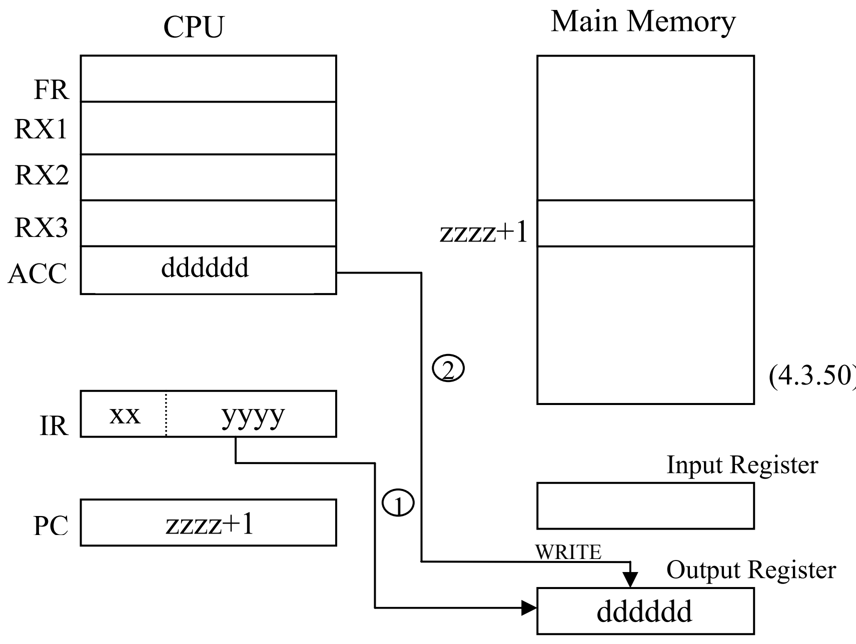
H) OUPUT, address

- The address field would specify one of the set of Output Registers representing the Output Unit. In this particular case of EEC there is provided a single Output Register, so that the address field has no significance. (4.3.44)
- In general case, there can be defined 2^k different addresses of Output Registers. (4.3.45)
- This instruction transfers the content of the Accumulator to the addressed Output Register and writes it in. (4.3.46)
- Thus, symbolically: (4.3.47)

  $$\text{Output Register}_{\text{ADDRESS}} \longleftarrow (\text{ACC})$$
- The steps of the implementation: (4.3.48)

1. Identify the Output Register from the address existing in $(\text{IR})_{\text{ADDRESS}}$
2. Transfer the operand from ACC to the identified Output Register and write it in the register.
3. Go to next FETCH phase.

- In the register view of the EEC the execution of this instruction is presented in (4.3.50): (4.3.49)



(4.3.50)