8086/8 Emulator

8086/8 is a processor of 16 bit data bus(8086) and 8 bit data bus(8088). We made some of the functions in programming language and implement some of the functions of the processor. We used python in this case and made the GUI for our processor. It takes the input command like in assembly language and do the task that have to be done in assembly language. For instance if it takes the input of mov ax, bx, it moves the content of the bs register to ax. We have implemented some of the instructions of assembly language in it.

What our Program can do?

- Mov instruction in assembly.
- Increment (inc)
- Decrement (dec)
- Rotate left (rol)
- Rotate right(ror)
- Shift right (shr)
- Shift left (shl)
- AND of content
- OR of content
- XOR of content
- NOT of content

MOV:

Mov instruction will deal with the cases like register to register, register to memory, memory to register, immediate to memory and immediate to register.

It will do all the above cases and show the results after execution. It will change the content of registers used in the instruction after execution. It simply copy the content of one to the other.

It will check that registers entered in the instruction is available or not. If not then error will be thrown. Also same is the case with memory. If user entered the wrong memory location, like in our case only 16 memory locations are available 0 to F so if user entered the location beyond this value an error will occurred. Same if user tries to access the content of DS, CS, ES, SS registers separately like higher or lower an error will occurred.

INCREMENT:

INC will deal the cases with register and memory both. It will increment the content of register or memory whichever used by one. It will check both the contents of register and memory. It will also check that if register is correct, available or not.

Also same with the case of memory it will check that only 16 memory locations are available not beyond this. In other case an error will be thrown

DECREMENT:

DEC will deal the cases with register and memory both. It will decrement the content of register or memory, whichever used, by one. It will check both the contents of register and memory. It will also check that if register is correct and available or not.

Also same with the case of memory it will check that only 16 memory locations are available not beyond this. In other case an error will be thrown

ROTATE LEFT:

ROL will deal with the case of rotation. It handle both cases with registers and with memory contents. It will rotate the bits of anyone from both from right to left.

For instance, if in AX there is value stored 10101011. If we apply rol AX, then new content will be 01010111 i.e. first bit will go to an end.

Also it will deal with the case that if the register is correct or not and memory is available or not.

ROTATE RIGHT:

ROR will deal with the case of rotation. It handle both cases with registers and with memory contents. It will rotate the bits of anyone from both from left to right.

For instance, if in AX there is value stored 10101011. If we apply ror AX, then new content will be 11010101 i.e. first bit will go to an end.

Also it will deal with the case that if the register is correct or not and memory is available or not.

SHIFT LEFT:

SHL will handle the cases of shifting the contents. In SHL it will shift the contents of register and memory both to the left. In this case the first bit will be lost and in the end 0 will append. If we keep doing shifting left then if we remain with the last bit in the register or memory, if we shift it as well it will be stored in carry register.

Also It will check the cases of error described above.

SHIFT RIGHT:

SHR will handle the cases of shifting the contents. In SHR it will shift the contents of register and memory both to the right. In this case the last bit will be lost and in the start 0 will append. If we keep doing shifting right, and if we remain with the last bit in the register or memory, if we shift it as well it will be stored in carry register.

Same is the case with this of error handling.

AND:

AND will do the simply ‘and’ operation with all bits. It will take two operands for execution.

TRUTH TABLE OF AND:

A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

It will operate the and operation with register to register contents, register to immediate, memory to immediate etc.

It will handle the cases of the error if register or memory not available.

OR:

OR will do the simply ‘or’ operation with all bits. It will take two operands for execution.

TRUTH TABLE OF OR:

A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

It will operate the ‘or’ operation with register to register contents, register to immediate, memory to immediate etc.

It will handle the cases of the error if register or memory not available.

XOR:

XOR will do the simply ‘xor’ operation with all bits. It will take two operands for execution.

TRUTH TABLE OF XOR:

A	B	A xor B
0	0	0
0	1	1
1	0	1
1	1	0

It will operate the ‘xor’ operation with register to register contents, register to immediate, memory to immediate etc.

It will handle the cases of the error if register or memory not available.

NOT:

NOT will simply do the operation of not in it. It will take only one operand to take not.

Truth table of not:

A	not A
0	1
1	0

It will take only register or memory as a operand. And simply reverse the bit of it.

It will also deal with the case of an error.

REGISTERS:

We Implemented the registers AX, BX, CX, DX, DS, CS, SS, ES and sub-registers of first 4 registers as last 4 will not allow access of higher or lower bit register. So to deal with this we divided the register AX to AH and AL. in this case if user wants to access the data of higher or lower bits of AX it will access also if he wants to access the total content it will allow. In this if he wants to add data in lowers bits of AX only it will enter that data in lower bits and will not effect the higher bits, higher bits will remain the same.

We have created the functions for register that would enter the data in it and assure that the data is correct and if we using 16 bits register then data must not exceed 16 bits. And the function which return the data of certain register (lower and higher separately as well in case of AX, BX, CX, DX).

Same is the case when he wants to access, change the higher bits only it will not effect the lower ones.

In our case AX is of 16 bits register and first 8 bits are higher and next 8 are lower.

MEMORY:

Memory in our case is of 3 bits as only 16 memory location available so three bits are enough to access the memory locations. It is from 0 to F in hex. In this we have created the functions that would enter the data in the memory locations specified. Check the data and enter it in memory. We have created the functions that will get data from certain memory locations and output the data.

SUPPORTIVE FUNCTIONS:

we have created many small functions that would help in coding the logic.

These are:

- Check Register
- Check memory

- Return decimal
- 2’s complement
- Check binary string
- Check if it is memory or not
- Convert string to list
- Convert hex to binary

These are helped in various places in coding

CHECK REGISTER:

This check if entered register is in list of register or not.

CHECK MEMORY:

This check if entered memory location is in our memory or not.

RETURN DECIMAL:

This return a value passed in string to decimal which helped us in conversion from number that in string to decimal to operate certain operations that are doable in integer form.

All of these functions helped us in coding.

MACHINE CODE

Machine Code is very important in this case as it is very challenging. We assign different opcode to the 15 instructions and to the register and memory.

Machine code is function that set the machine code to the instruction user entered.

It assign the correct opcode used like in case of MOV it will use opcode of MOV instruction. It will assign the codes to different registers as well. It contain the DW bits as well to check weather the size of register is 16 (WORD) bit or 8 bit (BYTE) if W bit is 1 it means that we are using 16 bit register in instruction. Same when the content is being moved into the register then D bit is 1 and when it is moved into R/M D bit is 0.

GUI

Our GUI handle with the input by user and done certain operations on it and show results. First it reads the instruction given by user and detect which command user entered either it is MOV or AND or whatever. It detects instructions given by user and send this to backend where all of the required functions to generate opcode are performed. It shows the contents of register and memory as well. If you change the data of some register you will see that change in register block of that register. Same is the case with memory. After execution it will show you the machine code.

THE END