Muhammed Emin Deniz - Onur Can Avcı

2017400186 - 2016400393

CMPE 230 First Assignment Assembly Interpreter - 2020 Spring

In this project, we implemented an interpreter for an assembly language of a hypothetical 8086- like CPU called HYP86 which does some basic instructions given in the project description. We use C++ for this project.

First, we define global variables which are memory and registers and flags. Then, we start to read the given input line by line. According to the line, if the line is empty, we continued the line. If the line contains “code segment”, we assign zero to our counter and we checked the counter like if counter smaller than zero waits until code segment is seen. If there is an “int 20h” string in the line, we start to defining variables. If the line contains “code ends”, it means that stop and break reading line.

In label parsing, we determine the type of variable, and store them in a map, in the program we can easily get lines that labels exist. We use the same technique for variable parsing, storing an index of memory and type with variables name in map.

We read other lines according to is there any comma or not. If there is a comma, it means that there are two parameters. If not, there is a parameter. In the line, the first string is determined the type of instructions and other second and third string generally destination and source which are mov, cmp, add and sub instructions. Other instructions which are xor, or, and, rcl, rcr, shl and shr instructions have two parameters too.

If there is not comma at the line, it means that there is a parameter for the instruction. These are push, pop, div, mul, int, not, nop, inc, dec and jump (jz, jnz, je, jne, ja, jae, jb, jbe, jnae, jnb, jnbe, jnc and jc operations) instructions.

In the development stage, we face some strange things that are out “cout” outputs are overwriting, we can not find the solution easily. We used “std:::getline(str, delimiter)” for parsing line, indeed “getline” gets “\r” return carriage and this causes the problem. And we use a method named with “strip” like python standard library method, it does two things, the first one is getting rid of “\r”, the second one is deleting whitespaces from left and right side that is unnecessary. It makes us faster.

Also, we implement debug mode for seeing the current state of stack, memory, registers, variables, flags. It can be open or closed at the beginning of main.cpp. we need it due to “int 21h” doesn’t work perfect when our debug outputs exist.

We have a lower case function for matching functions name and reading data.

We create a function for defining type of operand (typeofoperand), value of memory, registers, values and variables and amount (getvalue) of bit numbers (bitnumberof). In the getvalue function, we determine is string is 16 bit or 8 bit. In the typeofoperand function, we define register as “reg”, memory as “memory”, offsets as “offset” and others as “value”. Particularly “getvalue” method is very helpful, it returns value of whatever type of operand.

We also code a method which is called “getmemoref”, it returns the reference of memory, it can be an assignable method, due to reference.

Mov method is a crucial method for hyp-86, due to most instructions needs to store new data somewhere, so mov has used in many other instructions.

Push and Mov provides using stack, stack starts from at the end of memory, and come back.

In “and”, “or” and “xor” instructions, we check the one of parameters is value or not with typeofoperand function. If one of them value, operated mov function to operated parameters with and (&), or (|) and xor(^). We also, operated not instruction with not(~).

In shl and shr instructions, we left shift the first parameter with “<<” second parameter times and right shift the first parameter with ”>>” second parameter times.

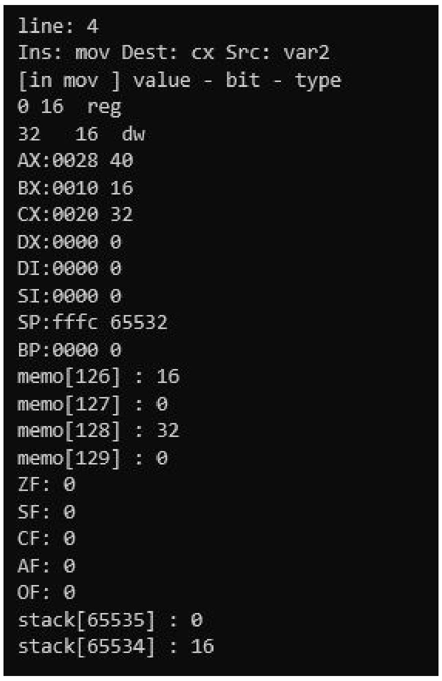
In cmp instruction, we check one of parameters is value or not with typeofoperand function. If one of their values, we compare the value of parameters. If they are equal to each other zero flag (zf) is true and carry flag (cf) is false. If value of second parameter bigger than first parameter zf is false and cf is true. If value of second smaller than first parameter zf and cf is false. If type of parameters are not value, first we compare the number of parameter bits and if they are equal, we check the flags.

In jump operations, we check the flags and according to flag conditions, we jump to label which comes in parameter with program counter (PC).

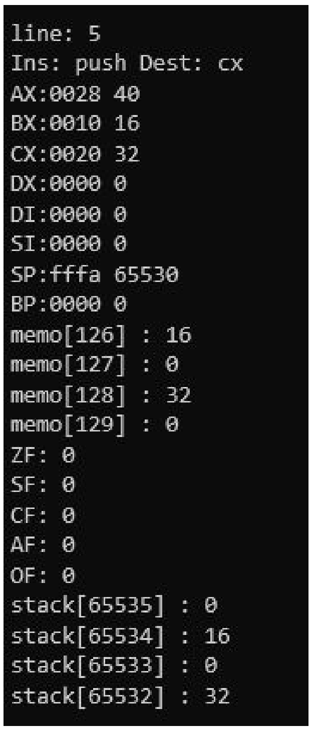
In add and sub instructions, we check the type of parameters and according to the types we compare the number of bits. Then we operate to mov operations and changed flags for appropriate conditions.

In rcl and rcr operations, we rotate the value bit by bit and we assign a value according to operation and then we assign value from carry flag. We operate this second parameter times.

Finally, we learned the basic operations in assembly languages thanks to this project. We comprehend when the flag changes, how to assign a value to registers, memories and others, how to operate in assembly. We understand that it is a very strong correspondence between the instructions in the language and the architecture's machine code instructions. Because assembly depends on the machine code instructions, every assembler has its own assembly language which is designed for exactly one specific computer architecture.

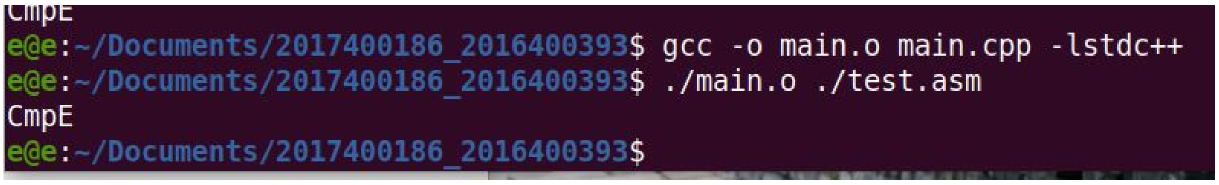


```
line: 4
Ins: mov Dest: cx Src: var2
[in mov ] value - bit - type
0 16  reg
32   16  dw
AX:0028 40
BX:0010 16
CX:0020 32
DX:0000 0
DI:0000 0
SI:0000 0
SP:fffc 65532
BP:0000 0
memo[126] : 16
memo[127] : 0
memo[128] : 32
memo[129] : 0
ZF: 0
SF: 0
CF: 0
AF: 0
OF: 0
stack[65535] : 0
stack[65534] : 16
```



```
line: 5
Ins: push Dest: cx
AX:0028 40
BX:0010 16
CX:0020 32
DX:0000 0
DI:0000 0
SI:0000 0
SP:fffa 65530
BP:0000 0
memo[126] : 16
memo[127] : 0
memo[128] : 32
memo[129] : 0
ZF: 0
SF: 0
CF: 0
AF: 0
OF: 0
stack[65535] : 0
stack[65534] : 16
stack[65533] : 0
stack[65532] : 32
```

Our Debug Mode Outputs



```
CmpE
e@e:~/Documents/2017400186_2016400393$ gcc -o main.o main.cpp -lstdc++
e@e:~/Documents/2017400186_2016400393$ ./main.o ./test.asm
CmpE
e@e:~/Documents/2017400186_2016400393$
```