VICHARAK COMPUTERS LLP
Assignment

1. Setup the 8-bit CPU Simulator

• Clone the 8-bit CPU repository from https://github.com/lightcode/8bit-computer.

• Read through the README.md to understand the CPU architecture and its instruction set.

• Run the provided examples to see how the CPU executes assembly code.

2. Understand the 8-bit CPU Architecture

• Review the Verilog code in the rtl/ directory, focusing on key files such as machine.v.

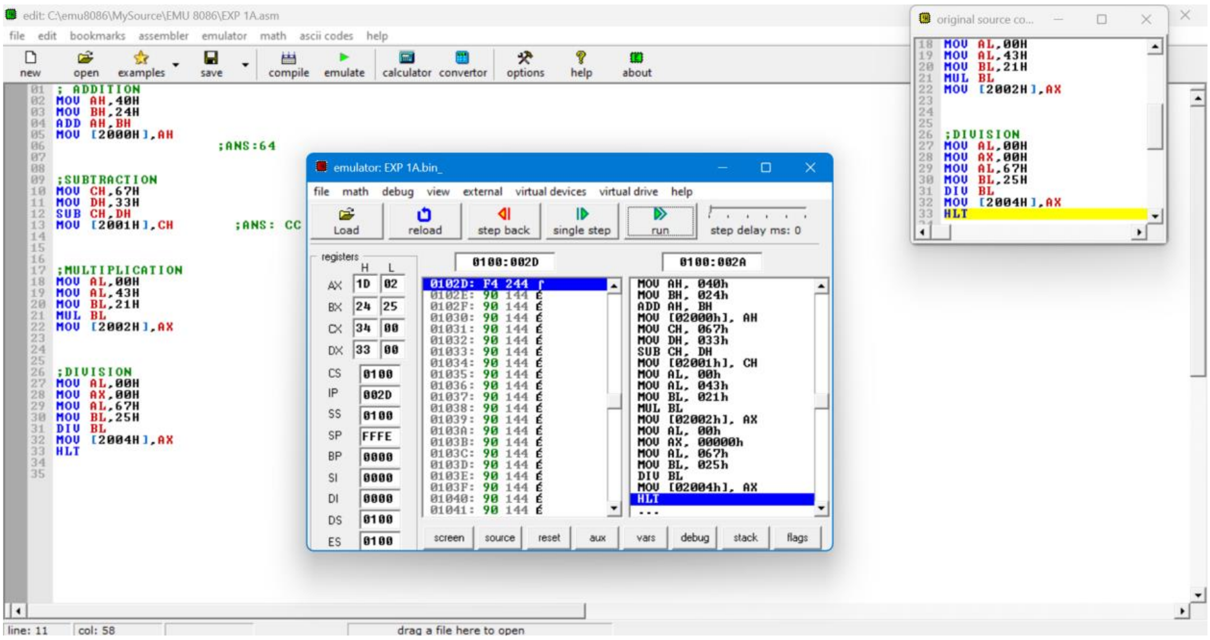
• Identify the CPU's instruction set, including data transfer, arithmetic, logical, branching, machine control, I/O, and stack operations.

Here in this steps I have used the emu 8086 emulator tool as a simulator for assemble program to run

And the below mentioned code is the output for the requirements asked in the 1st and 2nd steps in the task given

```
; ADDITION

MOV AH,40H

MOV BH,24H

ADD AH,BH

MOV [2000H],AH

;SUBTRACTION

MOV CH,67H

MOV DH,33H

SUB CH,DH

MOV [2001H],CH

;MULTIPLICATION

MOV AL,00H

MOV AL,43H

MOV BL,21H

MUL BL

MOV [2002H],AX
MOV AL,00H

MOV AX,00H

MOV AL,67H

MOV BL,25H

DIV BL

MOV [2004H],AX

HLT
```

Expected output of the assembly level code in the emu 8086 emulator


edit: C:\emu8086\MySource\EMU 8086\EXP 1A.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
emulator: EXP 1A.bin_
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms: 0
screen source reset aux vars debug stack flags
original source co...
line: 11 col: 58 drag a file here to open

```
; ADDITION
MOV AH,40H
MOV BH,24H
ADD AH,BH
MOV [2000H],AH
                              ;ANS:64


;SUBTRACTION
MOV CH,67H
MOV DH,33H
SUB CH,DH
MOV [2001H],CH                  ;ANS: CC


;MULTIPLICATION
MOV AL,00H
MOV AL,43H
MOV BL,21H
MUL BL
MOV [2002H],AX


;DIVISION
MOV AL,00H
MOV AX,00H
MOV AL,67H
MOV BL,25H
DIV BL
MOV [2004H],AX
HLT
```

```
registers
    H  L
AX  1D 02
BX  24 25
CX  34 00
DX  33 00
CS  0100
IP  002D
SS  0100
SP  FFFE
BP  0000
SI  0000
DI  0000
DS  0100
ES  0100
```

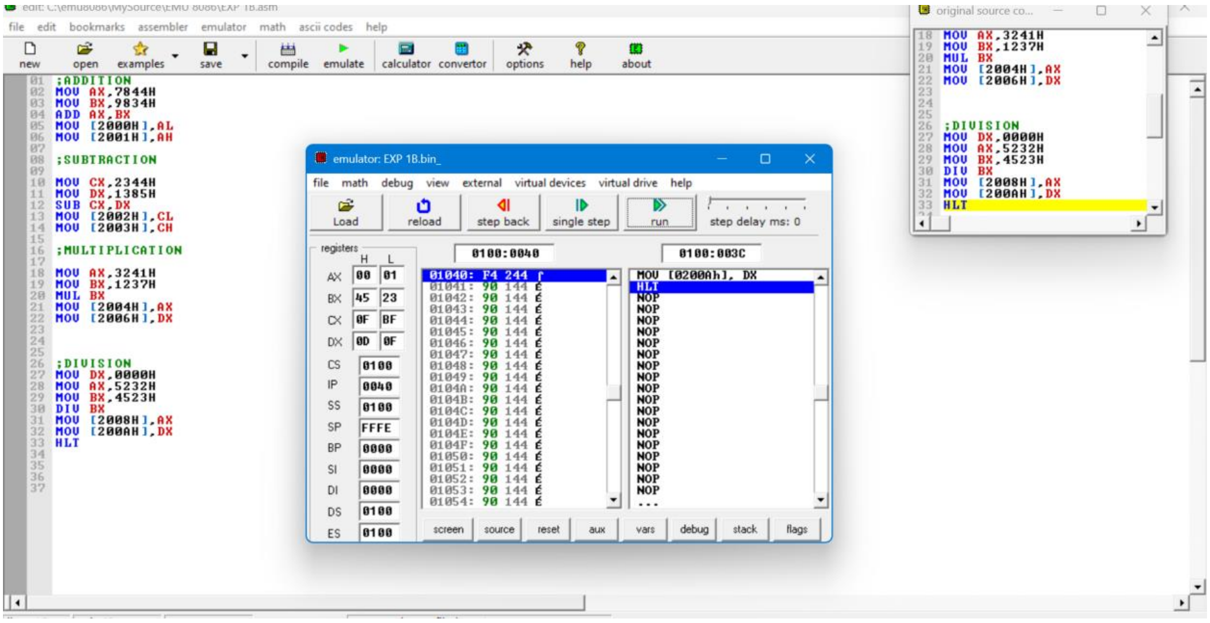
```
0100:002D
0102D: F4 244 ſ
0102E: 90 144 É
0102F: 90 144 É
01030: 90 144 É
01031: 90 144 É
01032: 90 144 É
01033: 90 144 É
01034: 90 144 É
01035: 90 144 É
01036: 90 144 É
01037: 90 144 É
01038: 90 144 É
01039: 90 144 É
0103A: 90 144 É
0103B: 90 144 É
0103C: 90 144 É
0103D: 90 144 É
0103E: 90 144 É
0103F: 90 144 É
01040: 90 144 É
01041: 90 144 É
```

```
0100:002A
MOV AH, 040h
MOV BH, 024h
ADD AH, BH
MOV [02000h], AH
MOV CH, 067h
MOV DH, 033h
SUB CH, DH
MOV [02001h], CH
MOV AL, 00h
MOV AL, 043h
MOV BL, 021h
MUL BL
MOV [02002h], AX
MOV AL, 00h
MOV AX, 00000h
MOV AL, 067h
MOV BL, 025h
DIV BL
MOV [02004h], AX
HLT
...
```

```
;ADDITION

MOV AX,7844H

MOV BX,9834H

ADD AX,BX

MOV [2000H],AL

MOV [2001H],AH

;SUBTRACTION

MOV CX,2344H

MOV DX,1385H

SUB CX,DX
```

```
MOV [2002H],CL
MOV [2003H],CH
;MULTIPLICATION
MOV AX,3241H
MOV BX,1237H
MUL BX
MOV [2004H],AX
MOV [2006H],DX
;DIVISION
MOV DX,0000H
MOV AX,5232H
MOV BX,4523H
DIV BX
MOV [2008H],AX
MOV [200AH],DX
HLT
```

And this is output the above written instructions.


edit: C:\emu8086\MySource\EMU 8086\EXP 1B.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
emulator: EXP 1B.bin_
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms: 0
registers
screen source reset aux vars debug stack flags
original source co...

```
;ADDITION
MOV AX,7844H
MOV BX,9834H
ADD AX,BX
MOV [2000H],AL
MOV [2001H],AH

;SUBTRACTION

MOV CX,2344H
MOV DX,1385H
SUB CX,DX
MOV [2002H],CL
MOV [2003H],CH

;MULTIPLICATION

MOV AX,3241H
MOV BX,1237H
MUL BX
MOV [2004H],AX
MOV [2006H],DX



;DIVISION
MOV DX,0000H
MOV AX,5232H
MOV BX,4523H
DIV BX
MOV [2008H],AX
MOV [200AH],DX
HLT
```

```
     H   L
AX   00  01
BX   45  23
CX   0F  BF
DX   0D  0F
CS   0100
IP   0040
SS   0100
SP   FFFE
BP   0000
SI   0000
DI   0000
DS   0100
ES   0100
```

```
0100:0040                  0100:003C
01040: F4 244 ſ            MOV [0200Ah], DX
01041: 90 144 É            HLT
01042: 90 144 É            NOP
01043: 90 144 É            NOP
01044: 90 144 É            NOP
01045: 90 144 É            NOP
01046: 90 144 É            NOP
01047: 90 144 É            NOP
01048: 90 144 É            NOP
01049: 90 144 É            NOP
0104A: 90 144 É            NOP
0104B: 90 144 É            NOP
0104C: 90 144 É            NOP
0104D: 90 144 É            NOP
0104E: 90 144 É            NOP
0104F: 90 144 É            NOP
01050: 90 144 É            NOP
01051: 90 144 É            NOP
01052: 90 144 É            NOP
01053: 90 144 É            NOP
01054: 90 144 É            ...
```

```
MOV AX,3241H
MOV BX,1237H
MUL BX
MOV [2004H],AX
MOV [2006H],DX



;DIVISION
MOV DX,0000H
MOV AX,5232H
MOV BX,4523H
DIV BX
MOV [2008H],AX
MOV [200AH],DX
HLT
```

3. Design a Simple High-Level Language (SimpleLang)

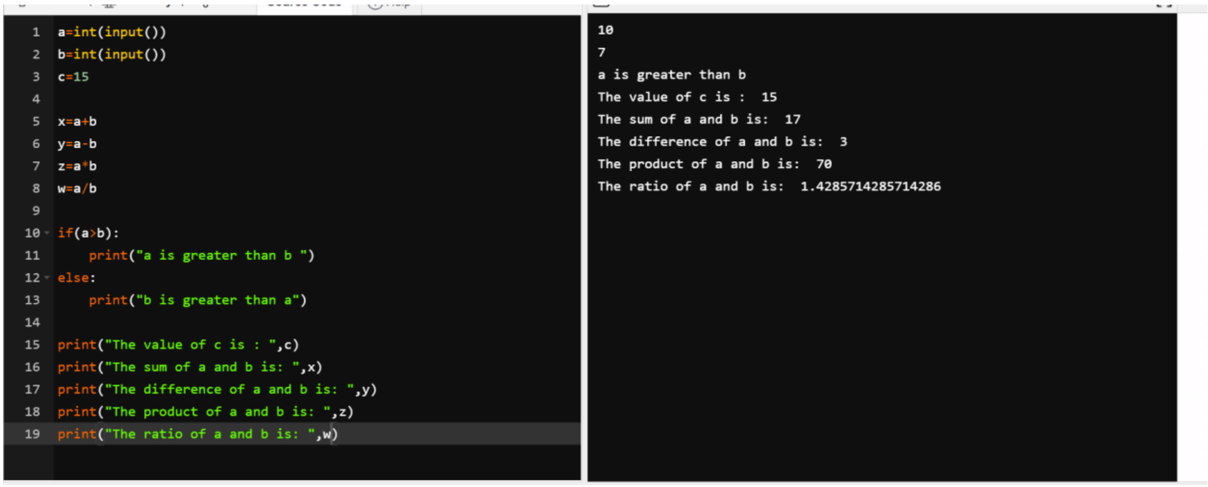
• Define the syntax and semantics for variable declarations, assignments, arithmetic operations, and conditionals.

• Document the language constructs with examples.

The below is simple high level language code implemented in python to define the asssignments , arithmetic operations and conditional statements.

```
a=int(input())

b=int(input())

c=15

x=a+b

y=a-b

z=a*b

w=a/b

if(a>b):

   print("a is greater than b ")

else:

   print("b is greater than a"

print("The value of c is : ",c)

print("The sum of a and b is: ",x)

print("The difference of a and b is: ",y)

print("The product of a and b is: ",z)

print("The ratio of a and b is: ",w)
```

And this is the expected output of the above simple high level language program.



```
a=int(input())
b=int(input())
c=15

x=a+b
y=a-b
z=a*b
w=a/b

if(a>b):
    print("a is greater than b ")
else:
    print("b is greater than a")

print("The value of c is : ",c)
print("The sum of a and b is: ",x)
print("The difference of a and b is: ",y)
print("The product of a and b is: ",z)
print("The ratio of a and b is: ",w)
```

```
10
7
a is greater than b
The value of c is :  15
The sum of a and b is:  17
The difference of a and b is:  3
The product of a and b is:  70
The ratio of a and b is:  1.4285714285714286
```

4. Create a Lexer

- **Write a lexer in C/C++ to tokenize SimpleLang code.**
- **The lexer should recognize keywords, operators, identifiers, and literals.**

5. Develop a Parser

- **Implement a parser to generate an Abstract Syntax Tree (AST) from the tokens.**
- **Ensure the parser handles syntax errors gracefully.**

This will be the lexer code code of resultant simpleLang code written in above steps:

```
int main() {
    std::string code = R"(a=int(input())
b=int(input())
c=15
x=a+b
y=a-b
z=a*b
w=a/b
if(a>b):
    print("a is greater than b ")
else:
    print("b is greater than a")
print("The value of c is : ",c)
print("The sum of a and b is: ",x)
print("The difference of a and b is: ",y)
print("The product of a and b is: ",z)
print("The ratio of a and b is: ",w))";

    std::vector<Token> tokens = tokenize(code);

    for (const auto& token : tokens) {
        std::cout << "Token Type: " << token.type << ", Value: " << token.value << std::endl;
    }

    return 0;
}
```

6. Generate Assembly Code

• Traverse the AST to generate the corresponding assembly code for the 8-bit CPU.

• Map high-level constructs to the CPU's instruction set (e.g., arithmetic operations to add, sub).

7. Integrate and Test

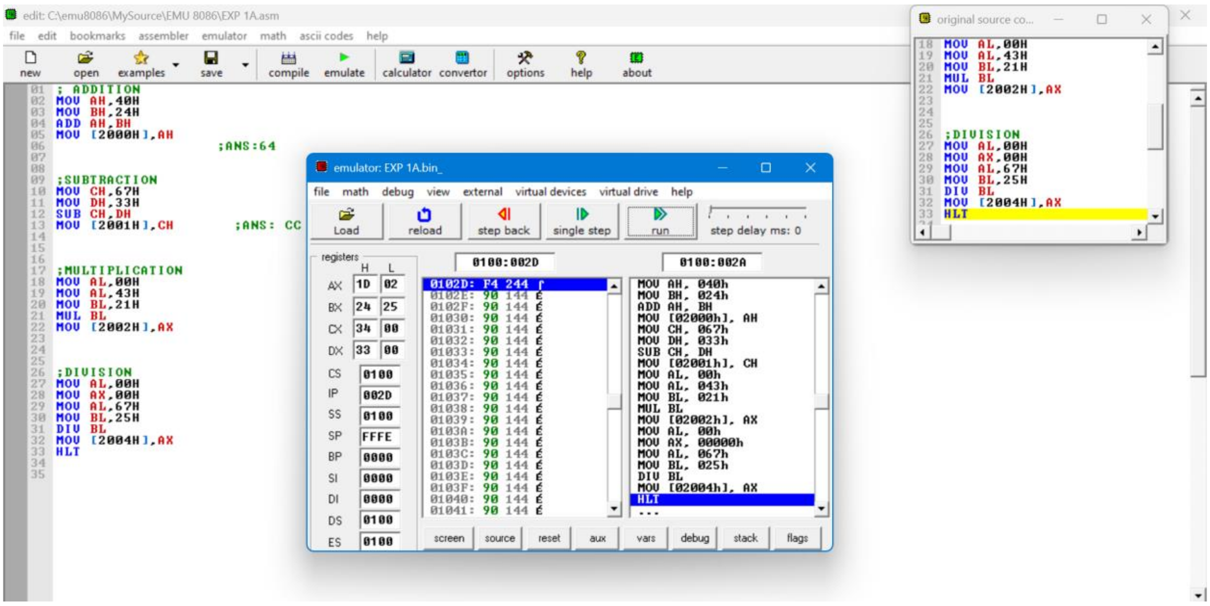
• Integrate the lexer, parser, and code generator into a single compiler program.

• Test the compiler with SimpleLang programs and verify the generated assembly code by running it on the 8-bit CPU simulator.

This is simple assembly level code that we get and its output is also attached by testing in emu 8086 emulator tool.

And I am sorry to say that I do not actually know how to Integrate the lexer, parser, and code generator into a single compiler program. So I am not doing it and I wanted to be genuine.

```
; ADDITION
MOV AH,40H
MOV BH,24H
ADD AH,BH
MOV [2000H],AH
;SUBTRACTION
MOV CH,67H
MOV DH,33H
SUB CH,DH
MOV [2001H],CH
;MULTIPLICATION
MOV AL,00H
MOV AL,43H
MOV BL,21H

MUL BL

MOV [2002H],AX

;DIVISION

MOV AL,00H

MOV AX,00H

MOV AL,67H

MOV BL,25H

DIV BL

MOV [2004H],AX

HLT
```


edit: C:\emu8086\MySource\EMU 8086\EXP 1A.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
emulator: EXP 1A.bin_
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms: 0
registers
screen source reset aux vars debug stack flags
original source co...

```
; ADDITION
MOV AH,40H
MOV BH,24H
ADD AH,BH
MOV [2000H],AH
                              ;ANS:64


;SUBTRACTION
MOV CH,67H
MOV DH,33H
SUB CH,DH
MOV [2001H],CH                  ;ANS: CC



;MULTIPLICATION
MOV AL,00H
MOV AL,43H
MOV BL,21H
MUL BL
MOV [2002H],AX



;DIVISION
MOV AL,00H
MOV AX,00H
MOV AL,67H
MOV BL,25H
DIV BL
MOV [2004H],AX
HLT
```

	H	L
AX	1D	02
BX	24	25
CX	34	00
DX	33	00
CS	0100	
IP	002D	
SS	0100	
SP	FFFE	
BP	0000	
SI	0000	
DI	0000	
DS	0100	
ES	0100	

```
0100:002D

0102D: F4 244 ſ
0102E: 90 144 É
0102F: 90 144 É
01030: 90 144 É
01031: 90 144 É
01032: 90 144 É
01033: 90 144 É
01034: 90 144 É
01035: 90 144 É
01036: 90 144 É
01037: 90 144 É
01038: 90 144 É
01039: 90 144 É
0103A: 90 144 É
0103B: 90 144 É
0103C: 90 144 É
0103D: 90 144 É
0103E: 90 144 É
0103F: 90 144 É
01040: 90 144 É
01041: 90 144 É
```

```
0100:002A

MOV AH, 040h
MOV BH, 024h
ADD AH, BH
MOV [02000h], AH
MOV CH, 067h
MOV DH, 033h
SUB CH, DH
MOV [02001h], CH
MOV AL, 00h
MOV AL, 043h
MOV BL, 021h
MUL BL
MOV [02002h], AX
MOV AL, 00h
MOV AX, 00000h
MOV AL, 067h
MOV BL, 025h
DIV BL
MOV [02004h], AX
HLT
...
```

```
MOV AL,00H
MOV AL,43H
MOV BL,21H
MUL BL
MOV [2002H],AX



;DIVISION
MOV AL,00H
MOV AX,00H
MOV AL,67H
MOV BL,25H
DIV BL
MOV [2004H],AX
HLT
```

Thank You