Setayesh Saeidi 40131851

Assembly project phase 2

Overview :

this project continues last phase project which was assembler for make assembly code into machine code for system , in this phase we visualize the way data of machine code goes to memory and also data's of data and stack segment.

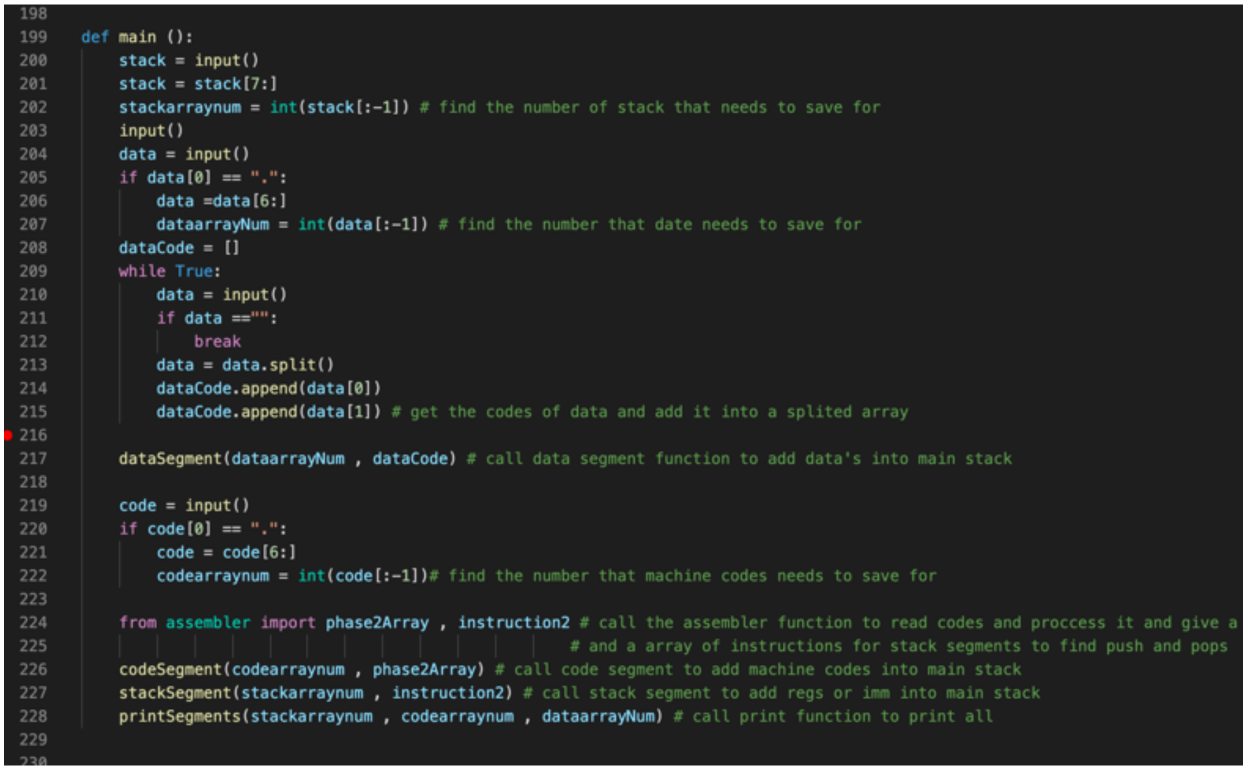
How does it work ?

Well first of all I'll give a general idea about code till we break it into little parts and explain it with all of the details:
1.getting inputs : our program gets a very specific way of input to work properly as it should so we provide the code for that
2. make assembly code into machine code : for this part we use our last project code to provide us with this data
3. save .data code into memory : it will process the code of data segment and put the data into stack
4. process stack data : it will process push and pop codes and put it into memory
5.visual the outputs : it will show a picture of memory in terminal

Now let's explain each part with lots of details!

How we get the proper input?

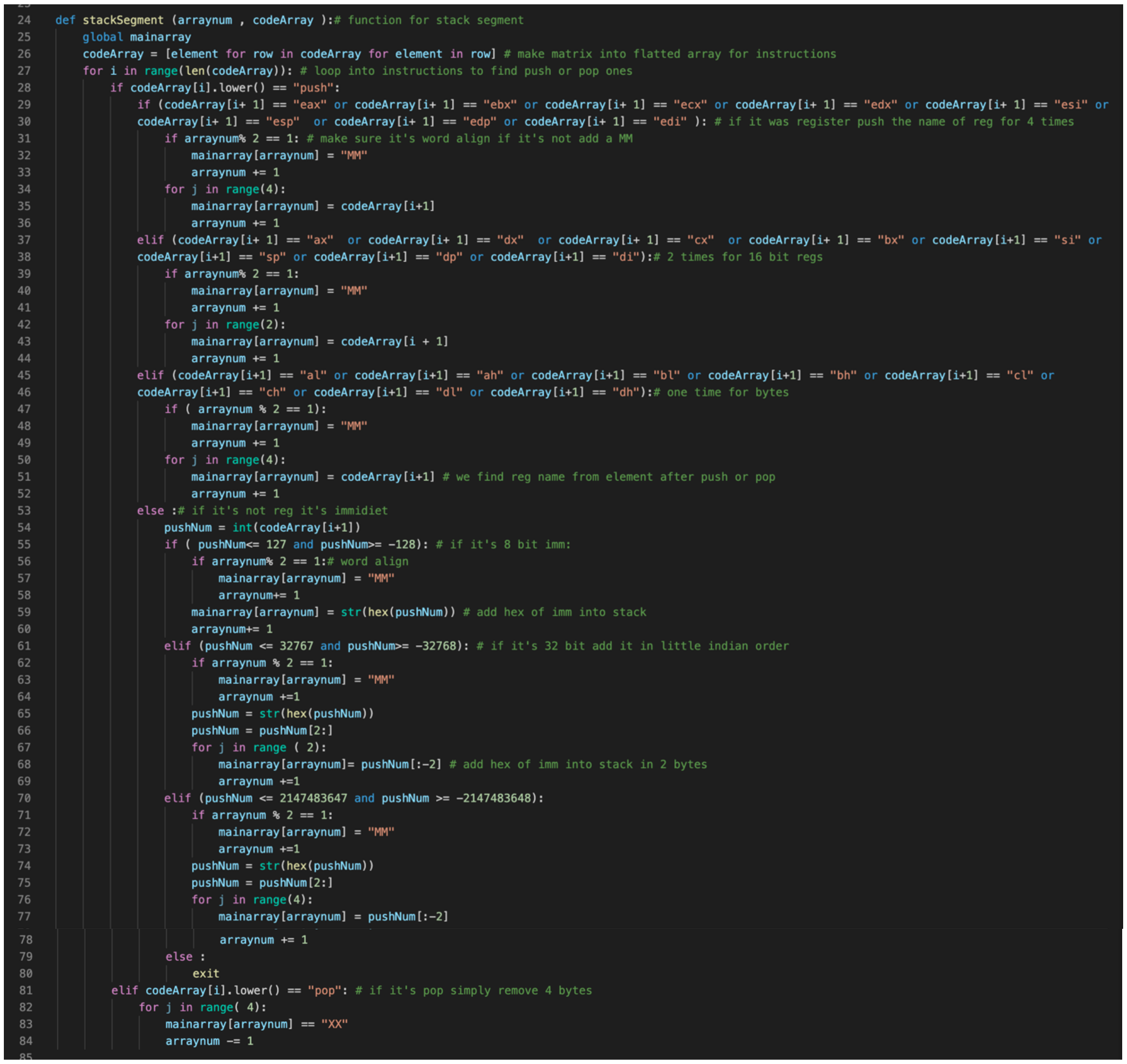
I wrote this part of my code in main function so as you can tell we a specific rules for our input , first for stack part we try to trim the number out of .stack(number), and then go to the data segment in there we again try to trim number of where part of memory should be saved out of .data(number) for .data we have some codes to so we get the codes and split it into an array that will be needed in processing data , after that we will be arrive into .code segment in this part we first trim the number too and call the assembler.py file from last project to process the code and give us a list of the instructions and a list of all machine codes that where generated ,the codes will be got in the other module and outputs will be printed too
After that we will generate memory management into different functions.



```python

def main ():
    stack = input()
    stack = stack[7:]
    stackarraynum = int(stack[:-1]) # find the number of stack that needs to save for
    input()
    data = input()
    if data[0] == ".":
        data =data[6:]
        dataarrayNum = int(data[:-1]) # find the number that date needs to save for
    dataCode = []
    while True:
        data = input()
        if data =="":
            break
        data = data.split()
        dataCode.append(data[0])
        dataCode.append(data[1]) # get the codes of data and add it into a splited array

    dataSegment(dataarrayNum , dataCode) # call data segment function to add data's into main stack

    code = input()
    if code[0] == ".":
        code = code[6:]
        codearraynum = int(code[:-1])# find the number that machine codes needs to save for

    from assembler import phase2Array , instruction2 # call the assembler function to read codes and proccess it and give a
                                                      # and a array of instructions for stack segments to find push and pops
    codeSegment(codearraynum , phase2Array) # call code segment to add machine codes into main stack
    stackSegment(stackarraynum , instruction2) # call stack segment to add regs or imm into main stack
    printSegments(stackarraynum , codearraynum , dataarrayNum) # call print function to print all

```

Functionality of stack segment :

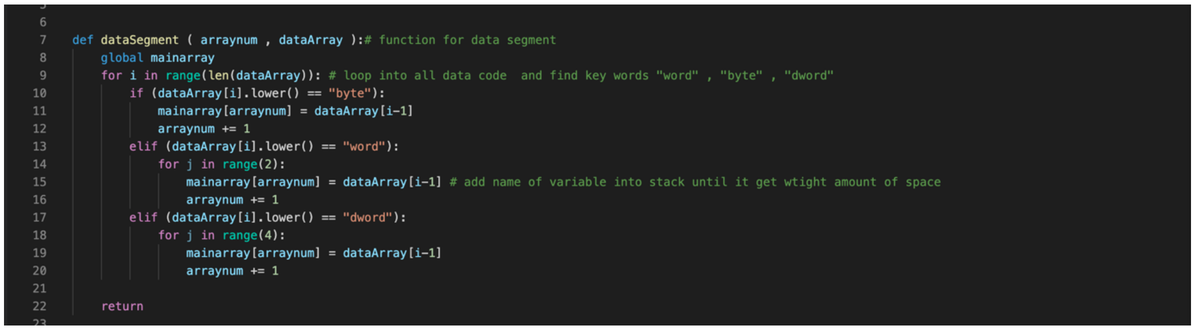
Only two instructions get to have place into stack segment memory ; push and pop ,for push we cover immediate and register in 8 ,16 and 32 bits
But for pop we only cover 32 bits register so how does it work ? we first recognize push and pop instructions in arrays of instructions named "codeArray" then we look the next element of code array and if it's regs we recognize how many bits there are and give it proper storage for immediate we recognize how many bits we need to storage that from the number for example if the number is between -128,127 we know it needs 8 bits
After that we store the immediate in little Indian order in stack segment of the memory
For pop instruction we simply just remove 4 bytes from memory



```python
def stackSegment (arraynum , codeArray ):# function for stack segment
    global mainarray
    codeArray = [element for row in codeArray for element in row] # make matrix into flatted array for instructions
    for i in range(len(codeArray)): # loop into instructions to find push or pop ones
        if codeArray[i].lower() == "push":
            if (codeArray[i+ 1] == "eax" or codeArray[i+ 1] == "ebx" or codeArray[i+ 1] == "ecx" or codeArray[i+ 1] == "edx" or codeArray[i+ 1] == "esi" or
            codeArray[i+ 1] == "esp"  or codeArray[i+ 1] == "edp" or codeArray[i+ 1] == "edi" ): # if it was register push the name of reg for 4 times
                if arraynum% 2 == 1: # make sure it's word align if it's not add a MM
                    mainarray[arraynum] = "MM"
                    arraynum += 1
                for j in range(4):
                    mainarray[arraynum] = codeArray[i+1]
                    arraynum += 1
            elif (codeArray[i+ 1] == "ax"  or codeArray[i+ 1] == "dx"  or codeArray[i+ 1] == "cx"  or codeArray[i+ 1] == "bx" or codeArray[i+1] == "si" or
            codeArray[i+1] == "sp" or codeArray[i+1] == "dp" or codeArray[i+1] == "di"):# 2 times for 16 bit regs
                if arraynum% 2 == 1:
                    mainarray[arraynum] = "MM"
                    arraynum += 1
                for j in range(2):
                    mainarray[arraynum] = codeArray[i + 1]
                    arraynum += 1
            elif (codeArray[i+1] == "al" or codeArray[i+1] == "ah" or codeArray[i+1] == "bl" or codeArray[i+1] == "bh" or codeArray[i+1] == "cl" or
            codeArray[i+1] == "ch" or codeArray[i+1] == "dl" or codeArray[i+1] == "dh"):# one time for bytes
                if ( arraynum % 2 == 1):
                    mainarray[arraynum] = "MM"
                    arraynum += 1
                for j in range(4):
                    mainarray[arraynum] = codeArray[i+1] # we find reg name from element after push or pop
                    arraynum += 1
            else :# if it's not reg it's immidiet
                pushNum = int(codeArray[i+1])
                if ( pushNum<= 127 and pushNum>= -128): # if it's 8 bit imm:
                    if arraynum% 2 == 1:# word align
                        mainarray[arraynum] = "MM"
                        arraynum+= 1
                    mainarray[arraynum] = str(hex(pushNum)) # add hex of imm into stack
                    arraynum+= 1
                elif (pushNum <= 32767 and pushNum>= -32768): # if it's 32 bit add it in little indian order
                    if arraynum % 2 == 1:
                        mainarray[arraynum] = "MM"
                        arraynum +=1
                    pushNum = str(hex(pushNum))
                    pushNum = pushNum[2:]
                    for j in range ( 2):
                        mainarray[arraynum]= pushNum[:-2] # add hex of imm into stack in 2 bytes
                        arraynum +=1
                elif (pushNum <= 2147483647 and pushNum >= -2147483648):
                    if arraynum % 2 == 1:
                        mainarray[arraynum] = "MM"
                        arraynum +=1
                    pushNum = str(hex(pushNum))
                    pushNum = pushNum[2:]
                    for j in range(4):
                        mainarray[arraynum] = pushNum[:-2]
                        arraynum += 1
                else :
                    exit
        elif codeArray[i].lower() == "pop": # if it's pop simply remove 4 bytes
            for j in range( 4):
                mainarray[arraynum] == "XX"
                arraynum -= 1
```

Functionality of data segment :

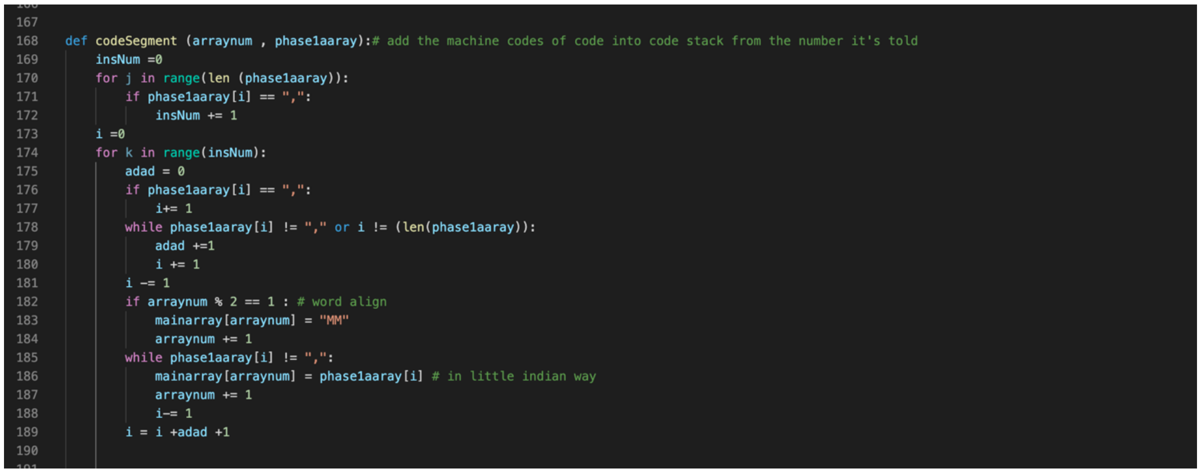
This function give a array of data codes that we splited into array in input plus which part of the stack should be saved
Then it searches for strings word , dword and byte in data codes and depends on the storage it needs put the name of the variable into data segment of the memory for example dword have 4 parts in it note that in all of it the memory is word align



```python

def dataSegment ( arraynum , dataArray ):# function for data segment
    global mainarray
    for i in range(len(dataArray)): # loop into all data code  and find key words "word" , "byte" , "dword"
        if (dataArray[i].lower() == "byte"):
            mainarray[arraynum] = dataArray[i-1]
            arraynum += 1
        elif (dataArray[i].lower() == "word"):
            for j in range(2):
                mainarray[arraynum] = dataArray[i-1] # add name of variable into stack until it get wtight amount of space
                arraynum += 1
        elif (dataArray[i].lower() == "dword"):
            for j in range(4):
                mainarray[arraynum] = dataArray[i-1]
                arraynum += 1

    return
```

Functionality of code segment :

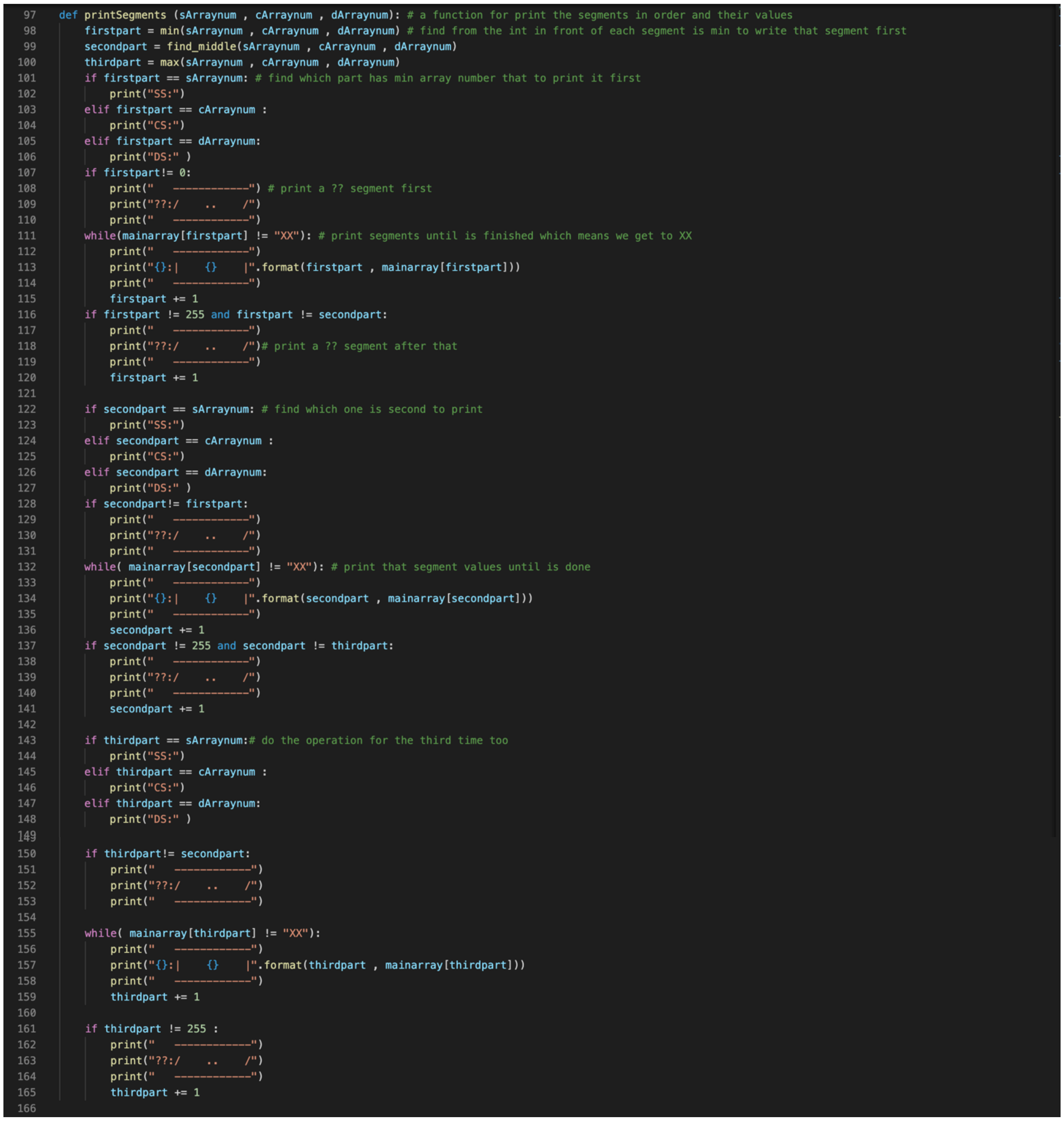
Well we have the machine codes we only need to put it in the code segment of memory
We have to do it in a little Indian way every instruction and it have to be word align so I end up with the code below but node that I have added a "," element before every instruction to find which machine code is for what instruction



```python

def codeSegment (arraynum , phase1aaray):# add the machine codes of code into code stack from the number it's told
    insNum =0
    for j in range(len (phase1aaray)):
        if phase1aaray[i] == ",":
            insNum += 1
    i =0
    for k in range(insNum):
        adad = 0
        if phase1aaray[i] == ",":
            i+= 1
        while phase1aaray[i] != "," or i != (len(phase1aaray)):
            adad +=1
            i += 1
        i -= 1
        if arraynum % 2 == 1 : # word align
            mainarray[arraynum] = "MM"
            arraynum += 1
        while phase1aaray[i] != ",":
            mainarray[arraynum] = phase1aaray[i] # in little indian way
            arraynum += 1
            i-= 1
        i = i +adad +1

```

Functionality of print segments function :

We find number of which segment is first , second and last in memory and then print the segments in the look liked format until you find "XX" in memory cause as you know all of memory initialized with XX



```python
def printSegments (sArraynum , cArraynum , dArraynum): # a function for print the segments in order and their values
    firstpart = min(sArraynum , cArraynum , dArraynum) # find from the int in front of each segment is min to write that segment first
    secondpart = find_middle(sArraynum , cArraynum , dArraynum)
    thirdpart = max(sArraynum , cArraynum , dArraynum)
    if firstpart == sArraynum: # find which part has min array number that to print it first
        print("SS:")
    elif firstpart == cArraynum :
        print("CS:")
    elif firstpart == dArraynum:
        print("DS:" )
    if firstpart!= 0:
        print("   ------------") # print a ?? segment first
        print("??:/    ..    /")
        print("   ------------")
    while(mainarray[firstpart] != "XX"): # print segments until is finished which means we get to XX
        print("   ------------")
        print("{}:|    {}    |".format(firstpart , mainarray[firstpart]))
        print("   ------------")
        firstpart += 1
    if firstpart != 255 and firstpart != secondpart:
        print("   ------------")
        print("??:/    ..    /")# print a ?? segment after that
        print("   ------------")
        firstpart += 1

    if secondpart == sArraynum: # find which one is second to print
        print("SS:")
    elif secondpart == cArraynum :
        print("CS:")
    elif secondpart == dArraynum:
        print("DS:" )
    if secondpart!= firstpart:
        print("   ------------")
        print("??:/    ..    /")
        print("   ------------")
    while( mainarray[secondpart] != "XX"): # print that segment values until is done
        print("   ------------")
        print("{}:|    {}    |".format(secondpart , mainarray[secondpart]))
        print("   ------------")
        secondpart += 1
    if secondpart != 255 and secondpart != thirdpart:
        print("   ------------")
        print("??:/    ..    /")
        print("   ------------")
        secondpart += 1

    if thirdpart == sArraynum:# do the operation for the third time too
        print("SS:")
    elif thirdpart == cArraynum :
        print("CS:")
    elif thirdpart == dArraynum:
        print("DS:" )

    if thirdpart!= secondpart:
        print("   ------------")
        print("??:/    ..    /")
        print("   ------------")

    while( mainarray[thirdpart] != "XX"):
        print("   ------------")
        print("{}:|    {}    |".format(thirdpart , mainarray[thirdpart]))
        print("   ------------")
        thirdpart += 1

    if thirdpart != 255 :
        print("   ------------")
        print("??:/    ..    /")
        print("   ------------")
        thirdpart += 1

```

Thanks for your attention I hope you'd understand the project well.