НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра системного програмування і спеціалізованих комп‟ютерних

систем

**КУРСОВА РОБОТА**

з дисципліни «Системне програмування»

на тему: Розробка компілятора програм мовою Асемблера

Студента 2 курсу КВ-42 групи

напряму підготовки

6.050102 «Ком`ютерна інженерія»

Сахніка Івана Сергійовича

Керівник\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_Оцінка: ECTS \_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Київ – 2016 рік

**Технічне завдання**

**Тема курсової роботи** – Розробка компілятора програм мовою Асемблера. Основним завданням і результатом роботи програми компілятора повинно бути створення для програм мовою Асемблера текстового файла лістинга (розширення .lst), подібного до файла лістинга компілятора MASM або TASM. Враховуючи, що створення компілятора є трудомістким процесом, який потребує значних витрат часу, у варіантах завдань на курсову роботу використовуються суттєві обмеження на перелік допустимих машинних інструкцій, режимів адресації даних і команд та допустимих директив. Будь який із запропонованих студентам варіантів індивідуальних завдань фактично вказує на підмножину стандартної мови Асемблера процесорів фірми Intel. Тим не менш, цієї підмножини достатньо для забезпечення мети розробки.

**Варіант 17 (КВ-42):**

*Ідентифікатори*

Містять великі і малі букви латинского алфавіту та цифри. Починаються з букви. Великі та малі букви не відрізняються. Довжина ідентифікаторів не більше 8 символів

*Константи*

Шістнадцятерічні, десяткові, двійкові та текстові константи

*Директиви*

END,

SEGMENT - без операндів, ENDS, программа може мати тільки один сегмент кодів і тільки один сегмент даних

EQU

IF-ELSE-ENDIF

DB,DW,DD з одним операндом - константою (строкові константи тільки для DB)

*Розрядність даних та адрес*

32 - розрядні дані та зміщення в сегменті, 16 -розрядні дані та зміщення не використовуються

*Адресація операндів пам'яті*

Базова індексна адресація ([edx+esi],[ebx+ecx] і т.п.) з оператором визначення типу (ptr) при необхідності

*Заміна сегментів*

Префікси заміни сегментів можуть задаватись тільки явно

*Машинні команди*

movsd

Inc **reg**

Dec **mem**

Add **reg,reg**

Cmp **reg,mem**

And **mem,reg**

Mov **reg,imm**

Or **mem,imm**

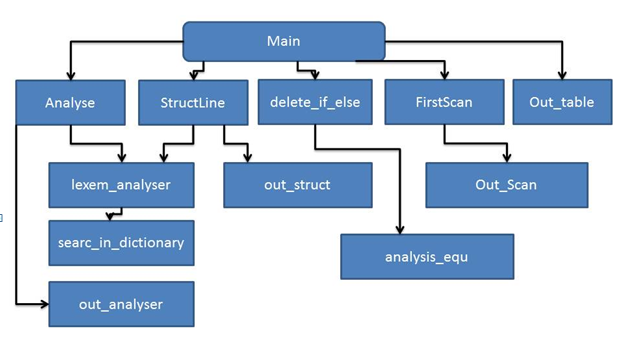
Jbe

Де **reg –** 8 або 32-розрядні РЗП

**mem –** адреса операнда в пам’яті

**imm -** 8 або 32-розрядні безпосередні дані (константи)

**Опис загальної структури програми:**



**Ф-ція main:** Ініціалізація програми. Викликає ф-ції **Analyse** та **StructLine** (лексичний аналізатор та структура речень відповідно)

**Ф-ція Analyse**: Виконує роль лексичного аналізатору. Принцип роботи полягає в тому, що построково зчитуємо тестовий файл після чого розбиваємо строку на окремі лексеми та за допомогою ф-ції **lexem\_analyser** визначаємо її тип. Після чого викликаємо ф-цію **out\_struct** яка записує отримані дані у файл лексичного аналізатора.

**Ф-ція out\_analyser**: приймає такі параметри(номер, лексема, довжина лексеми, тип лексеми). Дана ф-ція записує у файл отримані параметри та будує таблицю лексичного аналізатора.

**Ф-ція StructLine**: виконує роль структури речення. За допомогою лексичного аналізатора та початкового тестового файлу аналізуємо вміст кожної строки та заповнюємо відповідні поля (мнемокод, довжина мнемокоду, операнд1, довжина операнду1, операнд2, довжина операнду2) після чого передаємо їх до ф-ції **out\_struct**.

**Ф-ція out\_struct**: записує отримані дані в таблицю структури речення.

**Ф-ція lexem\_analyser**: визначає тип лексеми.

**Ф-ція search\_in\_dictionary**: пошук типу лексеми за ключем в словнику.

**Ф-ція analysis\_equ:** аналізує змінні що описані директивою EQU (запиує їх ідентифікатор та значення в список EQU\_list.

**Ф-ція delete\_if\_else:** за допомогою даних списку EQU\_listвидаляє вітки програми в які вона не заходить.

**Ф-ція FirstScan:** ф-ція що виконує роль першого перегляду. Генерує байти команд, виявляє помилки( звертання до неініціалізованого ідентифікатора, повторне входження команди, виявляє помилкові операнди аналізуючи окремо кожну команду.

**Ф-ція Out\_table:** дописує у файл MyLst(сформований першим переглядом) таблицю ідентифікаторів.

**Додаток 1:**

#Coursework made by Ivan Sakhnik

import re,string

#ініціалізація структур для роботи з лексемами

Registers\_32=('eax','ebx','edx','ecx','esi','ebp','esp','edi') #32-ох розрядні регістри

Registers\_8=('ah','al','bh','bl','dh','dl','ch','cl') #8-ми розрядні регістри

Segment\_Reg=('cs','ds','es','fs','gs','ss') #сегментні регістри

Command=('movsd','inc','dec','add','cmp','and','mov','or','jbe') #команди

Directive=('segment','equ','ends','end','db','dw','dd','if','else','endif') #директиви

TypeOperator=('ptr',) #тип оператора

Number=('0','1','2','3','4','5','6','7','8','9') #десяткові числа

Number\_16=('a','b','c','d','e','f','h') #16-теричні числа

Binary=('0','1') #двійкові числа

SingleSymbol=('+','\*',':','[',']',',') #односимвольні лексеми

Identifier=('byte','dword')

dictionary={Registers\_32:'32\_bit\_register', Registers\_8:'8\_bit\_register', Segment\_Reg:'segment\_register', #словник що зберігає в собі вищеописані кортежі

Command:'command', Directive:'directive', TypeOperator:'operator\_ident\_type\_def',

SingleSymbol:'single\_symbol', Identifier:'type\_definitor'}

EQU\_list=[] #список [ім'я, значення] для директиви EQU

def search\_in\_dictionary(lexem,\_list): #пошук лексеми за ключем в словнику

if lexem in \_list:

return 1

return 0

#Ф-ція що визначає тип лексеми

def lexem\_analyser(lexem):

if lexem=='':

return -1

for key in dictionary: #якщо лексема не є числом то шукаємо її по ключу в словнику

if search\_in\_dictionary(lexem,key):

return dictionary[key]

flag=0

if lexem[0] in Number: #перевірка на те чи є лексема 16-ричним числом

for i in range(0,len(lexem)):

if lexem[i] in Number\_16 or lexem[i] in Number:

continue

else:

flag=1

else:

flag=1

if lexem[len(lexem)-1] != 'h':

flag=1

if flag==0:

return 'hexadecimal\_number'

flag=0 #перевірка чи є двійковим числом

for i in range(0,len(lexem)-1):

if lexem[i] not in Binary:

flag=1

if lexem[len(lexem)-1] != 'b':

flag=1

if flag==0:

return 'binary\_number'

if lexem[0]=='"': #перевірка чи є текстовою константою

if lexem[len(lexem)-1]=='"':

return 'text\_const'

flag=0 #чи є десятковим числом

for i in range(0,len(lexem)):

if lexem[i] not in Number:

flag=1

if flag==0:

return 'decimal\_number'

flag=0 #якщо не було знайдено лексему в словнику перевіряємо це це ідентифікатор користувача

if len(lexem)<=8: #за умовою іденифікатори повинні бути менше 8 символів >>

if lexem[0] in list(string.ascii\_lowercase): #та починатися з великої або малої літери

for i in range(1,len(lexem)):

if lexem[i] in list(string.ascii\_lowercase) or lexem[i] in Number:

continue

else:

flag=1

else:

flag=1

else:

flag=1

if flag==1:

return 'ERROR'

else:

return 'user\_identifier'

# """ф-ція запису у файл результату виконання лексичного аналізатора"""

def out\_analyser(Nomber,lexem,type\_lexem,file):

if type\_lexem==-1:

return

if type\_lexem=='text\_const':

lexem=lexem[1:len(lexem)-1] #видаляємо '"' на початку та в кінці текстової константи

lexem=lexem.strip() #видаляє початкові і кінцеві пробіли лексеми

#приклад \_\_\_Ivan\_\_ >> Ivan

"""

file.write('|%6d'%Nomber); file.write(' | %21s'%lexem); file.write(' | %11d'%len(lexem));

file.write(' | %24s'%type\_lexem); file.write(' |\n');

file.write('-'\*75); file.write('\n');

"""

file.write('%d'%Nomber); file.write(' %s'%lexem); file.write(' %d'%len(lexem));

file.write(' %s'%type\_lexem); file.write('\n');

#Лексичний аналізатор

def Analyse(filename,filename1):

analyser=open(filename1, "w"); #відкриття файлу для записування

with open(filename) as test: #відкриття файлу для читання

"""

analyser.write('-'\*75); analyser.write('\n') #запис заголовку таблиці лексичного аналізатора

analyser.write('|%2s'%'Number'); analyser.write(' | %21s'%'Lexeme');

analyser.write(' | %8s'%'len(lexeme)');

analyser.write(' | %27s'%'type lexeme'' |'); analyser.write('\n')

analyser.write('-'\*75); analyser.write('\n');

"""

#count=0 #лічильник що рахує номер лексеми

for line in test:

count=0

line=line.split(';',1)[0] #видаляємо зі строки коментарій

line=line.lower()

i=0

while i < len(line):

if line[i] in SingleSymbol:

ch=line[i]

line=line.split(ch,1)

line=line[0]+' '+ch+' '+line[1] #розділяємо всі лексеми пробілами

i=i+3

else:

i=i+1

lexem=''

line=line.split() #розбиваємо строку на список лексем

for item in line: #прохід по списку

lexem=item

type\_lexem=lexem\_analyser(lexem) #передаємо лексему на аналіз

count+=1

out\_analyser(count,lexem,type\_lexem,analyser)

lexem=''

test.close() #закриття файлів

analyser.close()

# """ф-ція запису у файл структури речень"""

def out\_struct(label,mnemocode,len\_mnemo,op1,len\_op1,op2,len\_op2,file):

file.write('|%6d'%label); file.write(' | %10d'%mnemocode); file.write(' | %10d'%len\_mnemo);

file.write(' | %10d'%op1); file.write(' | %10d'%len\_op1);

file.write(' | %10d'%op2); file.write(' | %10d'%len\_op2); file.write(' |\n');

file.write('-'\*87); file.write('\n');

#Структура речень

def StructLine(filename,filename1):

Struct=open(filename1, "w"); #відкриття файлу для запису структури речень

with open(filename) as test: #відкриття файлу для читання

Struct.write('-'\*87); Struct.write('\n') #запис заголовку таблиці структури речень

Struct.write('|%6s'%'label'); Struct.write(' | %10s'%'mnemocode');

Struct.write(' | %10s'%'LenMnemo'); Struct.write(' | %10s'%'operand\_1');

Struct.write(' | %10s'%'len\_oper1'); Struct.write(' | %10s'%'operand\_2');

Struct.write(' | %10s'%'len\_oper2'); Struct.write(' | \n')

Struct.write('-'\*87); Struct.write('\n');

for line in test: #зчитуємо строки та розбиваємо по пробілах

if line=='' or line=='\n':

continue

line=line.split(';',1)[0]

line=line.lower()

i=0

while i < len(line):

if line[i] in SingleSymbol:

ch=line[i]

line=line.split(ch,1)

line=line[0]+' '+ch+' '+line[1]

i=i+3

else:

i=i+1

lexem='' #ініціалізація циклу

line=line.split() #обнуляємо всі дані необхідні для запису структури речення

label=0

mnemocode=0

len\_mnemo=0

len\_op1=0

len\_op2=0

op1=0

op2=0

flag=0

count=0

for item in line: #аналізуємо окремо кожну строку

lexem=item

type\_lexem=lexem\_analyser(lexem)

if type\_lexem=='user\_identifier' and flag==0: #заповнюємо поле label

count+=1

label=1

flag=1

lexem=''

continue

if type\_lexem=='command' or type\_lexem=='directive': #заповнюємо поле mnemocode

count+=1

mnemocode=count

len\_mnemo=1 #заповнюємо поле LenMnemo

lexem=''

flag=2

continue

if type\_lexem=='single\_symbol' and flag==1: #якщо в нас в строці лише ідентифікаор і розділовий знак

break #то виходимо з циклу

if lexem==',':

count+=1

flag=4

continue

if lexem!=',' and flag==3: #рахуємо довжину операнду1 якщо >1

count+=1

len\_op1+=1

continue

if type\_lexem!= 'command' and type\_lexem!= 'directive' and flag==2:

count+=1

op1=count #заповнюємо поле операнд\_1

len\_op1=1

flag=3

continue

if op1!=0 and type\_lexem!= 'command' and type\_lexem!= 'directive' and flag==4:

count+=1

op2=count #заповнюємо поле операнд\_2

len\_op2=1

flag=5

continue

if op1!=0 and flag==5:

count+=1

len\_op2+=1 #рахуємо довжину операнду2 якщо >1

continue

#передаємо отримані результати в ф-цію запису у файл

out\_struct(label,mnemocode,len\_mnemo,op1,len\_op1,op2,len\_op2,Struct)

test.close()

Struct.close()

#аналіз EQU

def analysis\_equ(test\_file):

test\_file=open(test\_file, "r")

EQU=0

ident=0 #змінна що запамятовує ідентифікатор

value=0 #змінна що запамятовує значення ідентифікатора

for line in test\_file: #прохід по файлу

#line=line.lower()

line=line.split()

if 'EQU' in line: #якщо зустрічається EQU, запамятовуємо ідентифікатор та його значення

ident=line[0]

value=line[2]

EQU\_list.append(ident) #отримані значення добавляємо в список

EQU\_list.append(value)

test\_file.close()

#ф-ція що видяляє зайві гілки програми

def delete\_if\_else(test, new\_test):

test=open(test, "r")

new\_test=open(new\_test, "w")

erase\_flag1=0

erase\_flag2=0

else\_flag=0

analysis\_equ('test.asm')

for line in test:

line\_copy=line

i=0

while i < len(line):

if line[i] in SingleSymbol:

ch=line[i]

line=line.split(ch,1)

line=line[0]+' '+ch+' '+line[1]

i=i+3

else:

i=i+1

line=line.split()

if 'if' in line:

value=EQU\_list[EQU\_list.index(line[1])+1]

if value!='0':

erase\_flag1=1

else:

erase\_flag2=1

if 'else' in line:

else\_flag=1

if erase\_flag1==1 and else\_flag==1:

if line[0]=='endif':

new\_test.write(line\_copy)

erase\_flag1=0

else\_flag=0

continue

if erase\_flag2==1:

if else\_flag==1:

new\_test.write(line\_copy)

erase\_flag2=0

else\_flag=0

continue

new\_test.write(line\_copy)

new\_test.close()

test.close()

#Перший перегляд

def FirstScan(asm,new\_test, new\_analyser, my\_lst):

first\_test=open(asm, "r")

test=open(new\_test, "r")

analyser=open(new\_analyser, "r")

out=open(my\_lst, "w")

UserIdentifier\_list\_all=[]

flag=0

for line in analyser:

line=line.split()

if line[1]=='code':

flag=1

continue

if flag==1 and line[3]=='user\_identifier' and line[0]=='1' :

UserIdentifier\_list\_all.append(line[1])

continue

if line[1]=='ends':

flag=0

continue

continue

analyser.seek(0)

count=0

line\_test=''

flag=0

UserIdentifier\_list=[]

EQU=0

count\_str=0

for line in analyser:

line=line.split()

if line[0]=='1':

Out\_Scan(out, count, line\_test, EQU)

line\_test=test.readline()

line\_first=first\_test.readline()

EQU=0

while (line\_test!=line\_first):

line\_first=first\_test.readline()

count\_str+=1

while line\_test=='\n':

Out\_Scan(out, count, line\_test, EQU)

line\_test=test.readline()

line\_first=first\_test.readline()

count\_str+=1

count\_str+=1

lab=0

directive=0

command=''

register1=0

register2=0

register=0

p=0

op2=0

EQU=0

ERROR=0

er=0

mem\_flag=0

seg\_reg=0

if line[3]=='user\_identifier' and line[0]=='1':

lab=1

if line[1]=='code':

flag=1

continue

if flag==1 and line[0]=='1' and line[3]=='user\_identifier':

UserIdentifier\_list.append(line[1])

continue

if line[1]=='ends':

flag=0

continue

continue

if line[1]=='equ':

if line[0]=='2':

EQU=1

continue

else:

ERROR=1

if EQU==1:

count=int(line[1],10)

continue

if lab==1:

if line[3]=='directive':

directive=1

if line[1]=='db':

inc=1

continue

if line[1]=='dw':

inc=2

continue

if line[1]=='dd':

inc=4

continue

if line[1]=='segment' or line[1]=='ends':

count=0

continue

if lab==1 and directive==1:

if line[3]=='decimal\_number' or line[3]=='hexadecimal\_number' or line[3]=='binary\_number':

count+=inc

continue

if line[3]=='text\_const':

count+=inc\*int(line[2],10)

continue

if line[3]=='command' and line[0]=='1':

if line[1]=='movsd':

count+=1

continue

if line[1]=='inc':

command='inc'

continue

if line[1]=='dec':

command='dec'

continue

if line[1]=='add':

command='add'

continue

if line[1]=='cmp':

command='cmp'

continue

if line[1]=='and':

command='and'

continue

if line[1]=='mov':

command='mov'

continue

if line[1]=='or':

command='or'

continue

if line[1]=='jbe':

command='jbe'

continue

if line[3]=='command' and line[0]!='1':

print('\nERROR: Excess comand(%d)'%count\_str)

if command=='inc':

if line[3]=='32\_bit\_register':

count+=1

continue

if line[3]=='8\_bit\_register':

count+=2

continue

if er!=1:

print('\nERROR: Invalid registers(%d)'%count\_str)

er=1

continue

if command=='dec':

if line[3]=='segment\_register':

seg\_reg+=1

continue

if line[1]=='byte' or line[1]=='word' or line[1]=='dword' or line[1]=='ptr':

continue

if line[1]==':' and seg\_reg==1:

seg\_reg+=1

continue

if line[1]=='[':

mem\_flag+=1

continue

if line[3]=='32\_bit\_register' and (mem\_flag==1 or mem\_flag==3):

mem\_flag+=1

continue

if line[1]=='+' and mem\_flag==2:

mem\_flag+=1

continue

if line[1]==']' and mem\_flag==4:

mem\_flag+=1

if mem\_flag==5:

if seg\_reg==2:

count+=4

continue

if seg\_reg==0:

count+=3

continue

print('\nERROR: Addressing expected(%d)'%count\_str)

continue

if er!=1:

print('\nERROR: Addressing expected(%d)'%count\_str)

er=1

continue

if command=='add':

if line[0]=='2' and line[3]=='32\_bit\_register':

register1=32

continue

if line[0]=='2' and line[3]=='8\_bit\_register':

register1=8

continue

if line[0]=='3' and line[1]==',':

continue

if line[0]=='4' and line[3]=='32\_bit\_register':

register2=32

if register1==register2:

count+=2

register1=0

register2=0

continue

print('\nERROR: Expected (add reg, reg) (%d)'%count\_str)

continue

if line[0]=='4' and line[3]=='8\_bit\_register':

register2=8

if register1==register2:

count+=2

register1=0

register2=0

continue

print('\nERROR: Expected (add reg, reg) (%d)'%count\_str)

continue

if er!=1:

print('\nERROR: Expected (add reg, reg) (%d)'%count\_str)

er=1

continue

if command=='cmp':

if line[3]=='segment\_register':

seg\_reg+=1

continue

if line[1]==':' and seg\_reg==1:

seg\_reg+=1

continue

if line[1]=='byte' or line[1]=='word' or line[1]=='dword' or line[1]=='ptr':

continue

if (line[3]=='32\_bit\_register' or line[3]=='8\_bit\_register') and mem\_flag==0:

mem\_flag+=1

continue

if line[1]==',' and mem\_flag==1:

mem\_flag+=1

continue

if line[1]=='[' and mem\_flag==2:

mem\_flag+=1

continue

if line[3]=='32\_bit\_register' and (mem\_flag==3 or mem\_flag==5):

mem\_flag+=1

continue

if line[1]=='+' and mem\_flag==4:

mem\_flag+=1

continue

if line[1]==']' and mem\_flag==6:

mem\_flag+=1

if mem\_flag==7:

if seg\_reg==2:

count+=4

continue

if seg\_reg==0:

count+=3

continue

print('\nERROR: Expected (cmp reg, mem) (%d)'%count\_str)

continue

if er!=1:

print('\nERROR: Expected (cmp reg, mem) (%d)'%count\_str)

er=1

continue

if command=='and':

if line[3]=='segment\_register':

seg\_reg+=1

continue

if line[1]==':' and seg\_reg==1:

seg\_reg+=1

continue

if line[1]=='byte' or line[1]=='word' or line[1]=='dword' or line[1]=='ptr':

continue

if line[1]=='[' and mem\_flag==0:

mem\_flag+=1

continue

if line[3]=='32\_bit\_register' and (mem\_flag==1 or mem\_flag==3):

mem\_flag+=1

continue

if line[1]=='+' and mem\_flag==2:

mem\_flag+=1

continue

if line[1]==']' and mem\_flag==4:

mem\_flag+=1

continue

if line[1]==',' and mem\_flag==5:

mem\_flag+=1

continue

if (line[3]=='32\_bit\_register' or line[3]=='8\_bit\_register') and mem\_flag==6:

mem\_flag+=1

if mem\_flag==7:

if seg\_reg==2:

count+=4

continue

if seg\_reg==0:

count+=3

continue

print('\nERROR: Expected (and mem, reg) (%d)'%count\_str)

continue

if er!=1:

print('\nERROR: Expected (and mem, reg) (%d)'%count\_str)

er=1

continue

if command=='mov':

if line[0]=='2' and line[3]=='32\_bit\_register':

register=32

continue

if line[0]=='2' and line[3]=='8\_bit\_register':

register=8

continue

if line[0]=='3' and line[1]==',':

continue

if line[0]=='4' and (register==32 or register==8):

if line[3]=='decimal\_number' or line[3]=='hexadecimal\_number' or line[3]=='binary\_number':

if line[3]=='decimal\_number':

b=int(line[1],10)

if line[3]=='hexadecimal\_number':

j=int(line[2],10)

b=int(line[1][0:j-1],16)

if line[3]=='binary\_number':

j=int(line[2],10)

b=int(line[1][0:j-1],2)

if register==8 and b<256:

count+=2

continue

if register==32:

count+=5

continue

print('\nERROR: Expected (mov reg, imm) (%d)'%count\_str)

continue

if er!=1:

print('\nERROR: Expected (mov reg, imm) (%d)'%count\_str)

er=1

continue

if command=='or':

if line[1]=='byte' or line[1]=='word' or line[1]=='dword' or line[1]=='ptr':

continue

if line[3]=='segment\_register':

seg\_reg+=1

continue

if line[1]==':' and seg\_reg==1:

seg\_reg+=1

continue

if line[1]=='[' and mem\_flag==0:

mem\_flag+=1

continue

if line[3]=='32\_bit\_register' and (mem\_flag==1 or mem\_flag==3):

mem\_flag+=1

continue

if line[1]=='+' and mem\_flag==2:

mem\_flag+=1

continue

if line[1]==']' and mem\_flag==4:

mem\_flag+=1

continue

if line[1]==',' and mem\_flag==5:

mem\_flag+=1

continue

if (line[3]=='decimal\_number' or line[3]=='hexadecimal\_number' or line[3]=='binary\_number') and mem\_flag==6:

if line[3]=='decimal\_number':

b=int(line[1],10)

if line[3]=='hexadecimal\_number':

j=int(line[2],10)

b=int(line[1][0:j-1],16)

if line[3]=='binary\_number':

j=int(line[2],10)

b=int(line[1][0:j-1],2)

if seg\_reg==2:

if b<=255:

count+=5

continue

else:

count+=8

continue

print('\nERROR: Expected (or mem, imm)(%d)'%count\_str)

continue

if seg\_reg==0:

if b<=255:

count+=4

continue

else:

count+=7

continue

print('\nERROR: Expected (or mem, imm)(%d)'%count\_str)

continue

print('\nERROR: Expected (or mem, imm)(%d)'%count\_str)

continue

if er!=1:

print('\nERROR: Expected (or mem, imm)(%d)'%count\_str)

er=1

continue

if command=='jbe':

if line[1] in UserIdentifier\_list:

count+=2

continue

if line[1] in UserIdentifier\_list\_all:

count+=6

continue

print('\nERROR: Unexpected identifier(%d)'%count\_str)

continue

Out\_Scan(out, count, line\_test, EQU)

test.close()

analyser.close()

out.close()

#ф-ція виводу першого перегляду

def Out\_Scan(file\_out, count, line\_test,equ):

if equ==1:

file\_out.seek(file\_out.tell()-8)

file\_out.write('=%04X'%count)

file\_out.write(' ');

file\_out.write('%s\n '%line\_test)

else:

if 'if' in line\_test or 'else' in line\_test or 'END' in line\_test or 'ends' in line\_test:

if 'END' in line\_test:

file\_out.write(' %s'%line\_test)

elif 'ends' in line\_test:

file\_out.write('%s'%line\_test)

else:

file\_out.seek(file\_out.tell()-8)

file\_out.write(' %s'%line\_test)

file\_out.write(' %04X'%count)

file\_out.write(' ');

else:

file\_out.write('%s'%line\_test)

file\_out.write(' %04X'%count); file\_out.write(' ');

def Out\_table(MyLst, out\_label):

LST=open(MyLst, "r")

out=open(out\_label, "w")

out.write('\n\n')

out.write('- - - - - - - - - - - - - - - - - - - - - - - - -')

out.write('\n')

out.write('%s'%'Name')

out.write('%15s'%'Size')

out. write('%10s'%'Length')

out.write('\n\n')

label=''

len\_label=''

for line in LST:

i=0

ch=''

while i < len(line):

if line[i] in SingleSymbol:

ch=line[i]

line=line.split(ch,1)

line=line[0]+' '+ch+' '+line[1]

i=i+3

else:

i=i+1

line=line.split()

if 'segment' in line:

label=line[line.index('segment')-1]

if 'ends'in line:

len\_label=line[line.index('ends')-2]

if label!='' and len\_label!='':

out.write('%s'%label)

out.write('%15s'%'32 bit')

out.write('%10s'%len\_label)

out.write('\n')

label=''

len\_label=''

continue

continue

out.write('\n')

out.write('- - - - - - - - - - - - - - - - - - - - - - - - -')

out.write('\n')

out.write('%5s'%'Name')

out.write('%12s'%'Type')

out.write('%10s'%'Value')

out.write('%10s'%'Attr')

out.write('\n\n')

LST.seek(0)

label=''

len\_label=''

segment=''

for line in LST:

i=0

ch=''

while i < len(line):

if line[i] in SingleSymbol:

ch=line[i]

line=line.split(ch,1)

line=line[0]+' '+ch+' '+line[1]

i=i+3

else:

i=i+1

line=line.split()

if 'segment' in line:

segment=line[line.index('segment')-1]

if 'ends'in line:

segment=''

lexem=''

for item in line:

lexem=item

lexem\_copy=item

lexem=lexem.lower()

type\_lexem=lexem\_analyser(lexem)

if line[0]=='if' or line[0]=='else' or line[0]=='endif' :

continue

if type\_lexem=='user\_identifier' and line.index(lexem\_copy)==1 and 'segment' not in line and 'ends' not in line:

if line[line.index(lexem\_copy)+1]=='db':

type\_out='BYTE'

elif line[line.index(lexem\_copy)+1]=='dw':

type\_out='WORD'

elif line[line.index(lexem\_copy)+1]=='dd':

type\_out='DWORD'

elif 'EQU' in line:

type\_out='NOMBER'

else:

type\_out='NEAR'

label=lexem\_copy

out.write('%5s'%label);

out.write('%12s'%type\_out)

out.write('%10s'%line[0])

out.write('%10s'%segment)

out.write('\n')

continue

LST=open(MyLst,"a")

out=open(out\_label, "r")

for line in out:

LST.write('%s'%line)

LST.close()

out.close()

#ініціалізація main

if \_\_name\_\_=="\_\_main\_\_":

f=''

f=input('\nEnter identifier file: ')

Analyse(f,'analyser.txt')

StructLine(f,'struct.txt')

delete\_if\_else(f,'new\_test.asm')

Analyse('new\_test.asm','new\_test\_analyser.txt')

FirstScan(f, 'new\_test.asm' , 'new\_test\_analyser.txt', 'MyLst.txt')

Out\_table('MyLst.txt', 'out\_label.txt')

#end

**Тестове завдання з помилками:**

data segment

A1 dd 12h

A2 dw 117

A3 db 01001b

Text1 db "Qwerty167"

data ends

T EQU 13

T1 EQU 0

code segment

movsd add

Point:

jbe erPoint

inc [ebx+ecx]

if T

dec ah

add ebx, [ebx+ecx]

else

cmp eax, [ebx+ecx]

endif

cmp [ebx+ecx], eax

and es:[ebx+ecx], 12h

jbe Point

mov eax

if T1

inc ah

else

mov al, 0h

endif

or eax, 132h

M1:

code ends

END