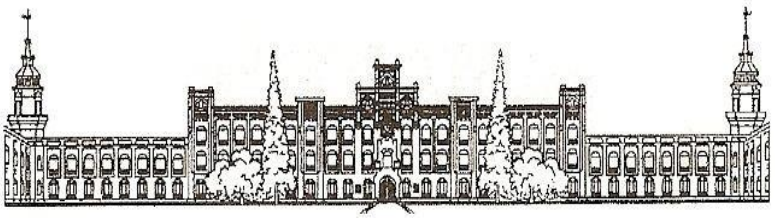

Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут» ім. Ігоря Сікорського

Курсова робота

з дисципліни «Системне програмування»

Тема: ***«Розробка компілятора програм написаних мовою Асемблера»***

Виконав: Лашков Денис
Група: КВ-33
Номер залікової книжки:

Допущений до захисту

_____

Київ 2025

Індивідуальне завдання

Варіант 10 Лашков Денис

Ідентифікатори
Містять великі і малі букви латинського алфавіту та цифри. Починаються з букви. Великі та малі букви не відрізняються. Довжина ідентифікаторів не більше 5 символів
Константи
Шістнадцяткові, десяткові та двійкові константи
Директиви
END, SEGMENT - без операндів, ENDS, програма може мати тільки один сегмент кодів і тільки один сегмент даних
DB, DW, DD з одним операндом - довільний арифметичний вираз над константами
Розрядність даних та адрес
16 - розрядні дані та зміщення в сегменті, у випадку 32 -розрядних даних та зміщень генеруються відповідні префікси заміни.
Адресація операндів пам'яті
Індексна адресація (Val[si], Val1[eax], Val1[edi] і т.п.). Пряма адресація
Заміна сегментів
Префікси заміни сегментів можуть задаватись явно, а при необхідності автоматично генеруються транслятором
Машинні команди
Cli
Push **mem**
Bt **reg, reg**
Lea **reg, mem**
Out **imm, reg**
Cmp **mem, imm**
Jnz
Jmp (внутрішньо-сегментний прямий)
Де **reg –** 8, 16 або 32-розрядні РЗП
mem – адреса операнда в пам’яті
imm – 8, 16 або 32-розрядні безпосередні дані (довільний арифметичний вираз над константами)

Тестовий файл test1.asm

```
data SEGMENT

    Val1 db 101b
    Valb1 db 10
    Valb2 db 0Ah
    Valb3 db 12+12*(6-4)/2

    val2 dw 0111b
    valw1 dw -16
    valw2 dw 45d
    ABCDE dw 32h
```

```
    val3 dd 12345678h
    vald1 dd 1101b
    vald2 dd 45d
    vald3 dd 12+6*(6-4)/4-3

    data ends

code SEGMENT
 cli

 jmp L2
    CodeV dw 0ABh

    push CodeV
    push cs:CodeV
    push ds:CodeV

    push val2
    push es:val3
    push ds:vald3

    bt edi, ebx
    bt di, ax

    lea dx, abcde[si]
    lea bx, vald1
    lea eax, val3[ebp]
    lea edi, val2[esi]

    out 10h, al
    out 20h, ax
    out 30h, eax

    cmp val1, 1
    cmp valw2, 0B2Ch
    cmp vald1, 0DEADh
    cmp val3, 0101b
    cmp vald3,12+6*(6-4)/4
    cmp ABCDE, 32h

    jnz mark2
    mark2:
    jnz mark2

    L1:
    jmp l1
    L2:
code ends
END
```

Тестовий файл test2.asm для компілятора TASM

```
.486
data segment use16
    Val1 db 101b
    Valb1 db 10
    Valb2 db 0Ah
    Valb3 db 12+12*(6-4)/2

    val2 dw 0111b
    valw1 dw -16
    valw2 dw 45d
    ABCDE dw 32h

    val3 dd 12345678h
    vald1 dd 1101b
    vald2 dd 45d
    vald3 dd 12+6*(6-4)/4

    data ends
code segment use16
assume ds:data, cs:code
 cli

 jmp L2
    CodeV dw 0ABh

    push CodeV
    push cs:CodeV
    push ds:CodeV

    push val2
    push es:val3
    push ds:vald3

    bt edi, ebx
    bt di, ax

    lea dx, abcde[si]
    lea bx, vald1
    lea eax, val3[ebp]
    lea edi, val2[esi]

    out 10h, al
    out 20h, ax
    out 30h, eax

    cmp val1, 1
    cmp valw2, 0B2Ch
    cmp vald1, 0DEADh
    cmp val3, 0101b
    cmp vald3,12+6*(6-4)/4-3
    cmp ABCDE, 32h

    jnz mark2
    mark2:
    jnz mark2

    L1:
    jmp l1
    L2:
code ends
END
```

Лістинг тестового файлу test2.asm (TASM)

```
Turbo Assembler     Version 2.5        05/18/25 17:27:03        Page 1
test2.ASM


      1                    .486
      2
      3        0000             data segment use16
      4        0000  05                 Val1 db 101b
      5        0001  0A                 Valb1 db 10
      6        0002  0A                 Valb2 db 0Ah
      7        0003  18                 Valb3 db 12+12*(6-4)/2
      8
      9        0004  0007                 val2 dw 0111b
     10        0006  FFF0                 valw1 dw -16
     11        0008  002D                 valw2 dw 45d
     12        000A  0032                 ABCDE dw 32h
     13
     14        000C  12345678             val3 dd 12345678h
     15        0010  0000000D             vald1 dd 1101b
     16        0014  0000002D             vald2 dd 45d
     17        0018  0000000F             vald3 dd 12+6*(6-4)/4
     18
     19        001C                 data ends
     20
     21        0000             code
 segment use16
     22                        assume    ds:data, cs:code
     23        0000  FA              cli
     24
     25        0001  EB 72    90          jmp L2
     26        0004  00AB                 CodeV dw 0ABh
     27
     28        0006  2E: FF 36 0004r         push CodeV
     29        000B  2E: FF 36 0004r         push cs:CodeV
     30        0010  FF 36    0004r            push ds:CodeV
     31
     32        0014  FF 36    0004r            push val2
     33        0018  66| 26: FF 36    000Cr         push es:val3
     34        001E  66| FF 36 0018r         push ds:vald3
     35
     36        0023  66| 0F A3 DF             bt    edi, ebx
     37        0027  0F A3    C7            bt    di, ax
     38
     39        002A  8D 94    000Ar            lea dx, abcde[si]
     40        002E  BB 0010r             lea bx, vald1
     41        0031  66| 67| 8D 85          +         lea eax, val3[ebp]
     42          0000000Cr
     43        0039  66| 67| 8D BE          +         lea edi, val2[esi]
     44          00000004r
     45
     46        0041  E6 10                out 10h, al
     47        0043  E7 20                out 20h, ax
     48        0045  66| E7 30             out 30h, eax
     49
     50        0048  80 3E    0000r 01         cmp val1, 1
     51        004D  81 3E    0008r 0B2C         cmp valw2,    0B2Ch
     52        0053  66| 81 3E 0010r     +         cmp vald1,    0DEADh
     53          0000DEAD
     54        005C  66| 83 3E 000Cr 05         cmp val3, 0101b
     55        0062  66| 83 3E 0018r 0C         cmp vald3,12+6*(6-4)/4-3
```

```
     56        0068  83 3E    000Ar 32          cmp ABCDE,     32h
     57
Turbo Assembler     Version 2.5        05/18/25 17:27:03        Page 2
test2.ASM


     58        006D  75 02    90 90              jnz mark2
     59        0071                    mark2:
     60        0071  75 FE                     jnz mark2
     61
     62        0073                    L1:
     63        0073  EB FE                     jmp l1
     64        0075                    L2:
     65        0075              code ends
     66                      END

Turbo Assembler     Version 2.5        05/18/25 17:27:03        Page 3
Symbol Table



Symbol Name              Type     Value

??DATE                   Text     "05/18/25"
??FILENAME               Text     "test2      "
??TIME                   Text     "17:27:03"
??VERSION                Number 0205
@CPU                    Text     0D1FH
@CURSEG                    Text     CODE
@FILENAME                Text     TEST2
@WORDSIZE                Text     2
ABCDE                    Word     DATA:000A
CODEV                    Word     CODE:0004
L1                      Near     CODE:0073
L2                      Near     CODE:0075
MARK2                     Near     CODE:0071
VAL1                     Byte     DATA:0000
VAL2                     Word     DATA:0004
VAL3                     Dword     DATA:000C
VALB1                     Byte     DATA:0001
VALB2                     Byte     DATA:0002
VALB3                     Byte     DATA:0003
VALD1                     Dword     DATA:0010
VALD2                     Dword     DATA:0014
VALD3                     Dword     DATA:0018
VALW1                     Word     DATA:0006
VALW2                     Word     DATA:0008

Groups & Segments          Bit Size Align  Combine Class

CODE                     16  0075 Para      none
DATA                     16  001C Para      none
```

Результат лексичного аналізу

```
№1 | "data SEGMENT "
Lexeme table:
    Index    Lexeme        Length        Type
    1    DATA        4        identifier
    2    SEGMENT        7        segment_start
SENTENCE STRUCT:
label: 1,1 | mnem: 2,1 |
```

```
№2 | "<empty line>"


№3 | "    Val1 db 101b"
Lexeme table:
    Index    Lexeme        Length        Type
    1    VAL1         4        identifier
    2    DB         2         var-size
    3    101B         4        imm_bin
SENTENCE STRUCT:
label: 1,1 | mnem: 2,1 | op: 3,1 |


№4 | "    Valb1 db 10"
Lexeme table:
    Index    Lexeme        Length        Type
    1    VALB1         5        identifier
    2    DB         2         var-size
    3    10         2         imm_dec
SENTENCE STRUCT:
label: 1,1 | mnem: 2,1 | op: 3,1 |


№5 | "    Valb2 db 0Ah"
Lexeme table:
    Index    Lexeme        Length        Type
    1    VALB2         5        identifier
    2    DB         2         var-size
    3    0AH          3         imm_hex
SENTENCE STRUCT:
label: 1,1 | mnem: 2,1 | op: 3,1 |


№6 | "    Valb3 db 12+12*(6-4)/2"
Lexeme table:
    Index    Lexeme        Length        Type
    1    VALB3         5        identifier
    2    DB         2         var-size
    3    12         2         imm_dec
    4    +         1        symbol
    5    12          2         imm_dec
    6    *         1        symbol
    7    (         1        symbol
    8    6         1        imm_dec
    9    -         1        symbol
    10     4          1         imm_dec
    11     )          1         symbol
    12     /          1         symbol
    13     2          1         imm_dec
SENTENCE STRUCT:
label: 1,1 | mnem: 2,1 | op: 3,11 |


№7 | "<empty line>"


№8 | "    val2 dw 0111b"
Lexeme table:
    Index    Lexeme        Length        Type
    1    VAL2         4        identifier
    2    DW         2         var-size
```

```
    3    0111B        5        imm_bin
SENTENCE STRUCT:
label: 1,1 | mnem: 2,1 | op: 3,1 |


№9 | "    valw1 dw -16"
Lexeme table:
    Index    Lexeme        Length        Type
    1    VALW1        5        identifier
    2    DW        2        var-size
    3    -        1        symbol
    4    16        2        imm_dec
SENTENCE STRUCT:
label: 1,1 | mnem: 2,1 | op: 3,2 |


№10 | "    valw2 dw 45d"
Lexeme table:
    Index    Lexeme        Length        Type
    1    VALW2        5        identifier
    2    DW        2        var-size
    3    45D        3        imm_dec
SENTENCE STRUCT:
label: 1,1 | mnem: 2,1 | op: 3,1 |


№11 | "    ABCDE dw 32h "
Lexeme table:
    Index    Lexeme        Length        Type
    1    ABCDE        5        identifier
    2    DW        2        var-size
    3    32H        3        imm_hex
SENTENCE STRUCT:
label: 1,1 | mnem: 2,1 | op: 3,1 |


№12 | "<empty line>"


№13 | "    val3 dd 12345678h"
Lexeme table:
    Index    Lexeme        Length        Type
    1    VAL3        4        identifier
    2    DD        2        var-size
    3    12345678H        9        imm_hex
SENTENCE STRUCT:
label: 1,1 | mnem: 2,1 | op: 3,1 |


№14 | "    vald1 dd 1101b"
Lexeme table:
    Index    Lexeme        Length        Type
    1    VALD1        5        identifier
    2    DD        2        var-size
    3    1101B        5        imm_bin
SENTENCE STRUCT:
label: 1,1 | mnem: 2,1 | op: 3,1 |


№15 | "    vald2 dd 45d"
Lexeme table:
    Index    Lexeme        Length        Type
    1    VALD2        5        identifier
```

```
    2    DD         2         var-size
    3    45D         3         imm_dec
SENTENCE STRUCT:
label: 1,1 | mnem: 2,1 | op: 3,1 |


№16 | "    vald3 dd 12+6*(6-4)/4-3"
Lexeme table:
    Index    Lexeme         Length         Type
    1    VALD3         5         identifier
    2    DD         2         var-size
    3    12         2         imm_dec
    4    +         1         symbol
    5    6         1         imm_dec
    6    *         1         symbol
    7    (         1         symbol
    8    6         1         imm_dec
    9    -         1         symbol
    10    4         1         imm_dec
    11    )         1         symbol
    12    /         1         symbol
    13    4         1         imm_dec
    14    -         1         symbol
    15    3         1         imm_dec
SENTENCE STRUCT:
label: 1,1 | mnem: 2,1 | op: 3,13 |


№17 | "<empty line>"


№18 | "    data ends"
Lexeme table:
    Index    Lexeme         Length         Type
    1    DATA         4         identifier
    2    ENDS         4         segment_ends
SENTENCE STRUCT:
label: 1,1 | mnem: 2,1 |


№19 | "<empty line>"


№20 | "code SEGMENT "
Lexeme table:
    Index    Lexeme         Length         Type
    1    CODE         4         identifier
    2    SEGMENT         7         segment_start
SENTENCE STRUCT:
label: 1,1 | mnem: 2,1 |


№21 | " cli"
Lexeme table:
    Index    Lexeme         Length         Type
    1    CLI         3         instruction
SENTENCE STRUCT:
mnem: 1,1 |


№22 | "<empty line>"
```

```
№23 | " jmp L2"
Lexeme table:
    Index    Lexeme        Length        Type
    1    JMP         3         instruction
    2    L2        2        identifier
SENTENCE STRUCT:
mnem: 1,1 | op: 2,1 |


№24 | "    CodeV dw 0ABh"
Lexeme table:
    Index    Lexeme        Length        Type
    1    CODEV         5        identifier
    2    DW        2        var-size
    3    0ABH         4        imm_hex
SENTENCE STRUCT:
label: 1,1 | mnem: 2,1 | op: 3,1 |


№25 | "<empty line>"


№26 | "    push CodeV      "
Lexeme table:
    Index    Lexeme        Length        Type
    1    PUSH         4        instruction
    2    CODEV         5        identifier
SENTENCE STRUCT:
mnem: 1,1 | op: 2,1 |


№27 | "    push cs:CodeV   "
Lexeme table:
    Index    Lexeme        Length        Type
    1    PUSH         4        instruction
    2    CS        2        seg_reg
    3    :         1        symbol
    4    CODEV         5        identifier
SENTENCE STRUCT:
mnem: 1,1 | op: 2,3 |


№28 | "    push ds:CodeV "
Lexeme table:
    Index    Lexeme        Length        Type
    1    PUSH         4        instruction
    2    DS        2        seg_reg
    3    :         1        symbol
    4    CODEV         5        identifier
SENTENCE STRUCT:
mnem: 1,1 | op: 2,3 |


№29 | "<empty line>"


№30 | "    push val2 "
Lexeme table:
    Index    Lexeme        Length        Type
    1    PUSH         4        instruction
    2    VAL2         4        identifier
SENTENCE STRUCT:
mnem: 1,1 | op: 2,1 |
```

```
№31 | "    push es:val3 "
Lexeme table:
    Index    Lexeme        Length        Type
    1    PUSH        4        instruction
    2    ES        2        seg_reg
    3    :        1        symbol
    4    VAL3        4        identifier
SENTENCE STRUCT:
mnem: 1,1 | op: 2,3 |


№32 | "    push ds:vald3"
Lexeme table:
    Index    Lexeme        Length        Type
    1    PUSH        4        instruction
    2    DS        2        seg_reg
    3    :        1        symbol
    4    VALD3        5        identifier
SENTENCE STRUCT:
mnem: 1,1 | op: 2,3 |


№33 | "<empty line>"


№34 | "    bt edi, ebx     "
Lexeme table:
    Index    Lexeme        Length        Type
    1    BT        2        instruction
    2    EDI        3        reg32
    3    ,        1        symbol
    4    EBX        3        reg32
SENTENCE STRUCT:
mnem: 1,1 | op1: 2,1 | op2: 4,1 |


№35 | "    bt di, ax "
Lexeme table:
    Index    Lexeme        Length        Type
    1    BT        2        instruction
    2    DI        2        reg16
    3    ,        1        symbol
    4    AX        2        reg16
SENTENCE STRUCT:
mnem: 1,1 | op1: 2,1 | op2: 4,1 |


№36 | "<empty line>"


№37 | "    lea dx, abcde[si] "
Lexeme table:
    Index    Lexeme        Length        Type
    1    LEA        3        instruction
    2    DX        2        reg16
    3    ,        1        symbol
    4    ABCDE        5        identifier
    5    [        1        symbol
    6    SI        2        reg16
    7    ]        1        symbol
SENTENCE STRUCT:
```

```
mnem: 1,1 | op1: 2,1 | op2: 4,4 |


№38 | "    lea bx, vald1 "
Lexeme table:
    Index    Lexeme        Length        Type
    1    LEA         3         instruction
    2    BX         2         reg16
    3    ,         1         symbol
    4    VALD1         5         identifier
SENTENCE STRUCT:
mnem: 1,1 | op1: 2,1 | op2: 4,1 |


№39 | "    lea eax, val3[ebp] "
Lexeme table:
    Index    Lexeme        Length        Type
    1    LEA         3         instruction
    2    EAX         3         reg32
    3    ,         1         symbol
    4    VAL3         4         identifier
    5    [         1         symbol
    6    EBP         3         reg32
    7    ]         1         symbol
SENTENCE STRUCT:
mnem: 1,1 | op1: 2,1 | op2: 4,4 |


№40 | "    lea edi, val2[esi]"
Lexeme table:
    Index    Lexeme        Length        Type
    1    LEA         3         instruction
    2    EDI         3         reg32
    3    ,         1         symbol
    4    VAL2         4         identifier
    5    [         1         symbol
    6    ESI         3         reg32
    7    ]         1         symbol
SENTENCE STRUCT:
mnem: 1,1 | op1: 2,1 | op2: 4,4 |


№41 | "<empty line>"


№42 | "    out 10h, al"
Lexeme table:
    Index    Lexeme        Length        Type
    1    OUT         3         instruction
    2    10H         3         imm_hex
    3    ,         1         symbol
    4    AL         2         reg8
SENTENCE STRUCT:
mnem: 1,1 | op1: 2,1 | op2: 4,1 |


№43 | "    out 20h, ax"
Lexeme table:
    Index    Lexeme        Length        Type
    1    OUT         3         instruction
    2    20H         3         imm_hex
    3    ,         1         symbol
    4    AX         2         reg16
```

```
SENTENCE STRUCT:
mnem: 1,1 | op1: 2,1 | op2: 4,1 |


№44 |  "    out 30h, eax"
Lexeme table:
    Index    Lexeme        Length        Type
    1    OUT         3         instruction
    2    30H         3         imm_hex
    3    ,         1         symbol
    4    EAX         3         reg32
SENTENCE STRUCT:
mnem: 1,1 | op1: 2,1 | op2: 4,1 |


№45 | "<empty line>"


№46 |  "    cmp val1, 1 "
Lexeme table:
    Index    Lexeme        Length        Type
    1    CMP         3         instruction
    2    VAL1          4          identifier
    3    ,         1         symbol
    4    1         1         imm_dec
SENTENCE STRUCT:
mnem: 1,1 | op1: 2,1 | op2: 4,1 |


№47 |  "    cmp valw2, 0B2Ch         "
Lexeme table:
    Index    Lexeme        Length        Type
    1    CMP         3         instruction
    2    VALW2         5         identifier
    3    ,         1         symbol
    4    0B2CH         5         imm_hex
SENTENCE STRUCT:
mnem: 1,1 | op1: 2,1 | op2: 4,1 |


№48 |  "    cmp vald1, 0DEADh      "
Lexeme table:
    Index    Lexeme        Length        Type
    1    CMP         3         instruction
    2    VALD1         5         identifier
    3    ,         1         symbol
    4    0DEADH          6          imm_hex
SENTENCE STRUCT:
mnem: 1,1 | op1: 2,1 | op2: 4,1 |


№49 |  "    cmp val3, 0101b"
Lexeme table:
    Index    Lexeme        Length        Type
    1    CMP         3         instruction
    2    VAL3          4          identifier
    3    ,         1         symbol
    4    0101B         5         imm_bin
SENTENCE STRUCT:
mnem: 1,1 | op1: 2,1 | op2: 4,1 |


№50 |  "    cmp vald3,12+6*(6-4)/4"
```

```
Lexeme table:
    Index    Lexeme        Length        Type
    1    CMP         3         instruction
    2    VALD3         5         identifier
    3    ,         1        symbol
    4    12         2         imm_dec
    5    +         1        symbol
    6    6         1        imm_dec
    7    *         1        symbol
    8    (         1        symbol
    9    6         1        imm_dec
    10    -         1         symbol
    11    4         1         imm_dec
    12    )         1         symbol
    13    /         1         symbol
    14    4         1         imm_dec
SENTENCE STRUCT:
mnem: 1,1 | op1: 2,1 | op2: 4,11 |


№51 |  "    cmp ABCDE, 32h"
Lexeme table:
    Index    Lexeme        Length        Type
    1    CMP         3         instruction
    2    ABCDE         5         identifier
    3    ,         1        symbol
    4    32H         3         imm_hex
SENTENCE STRUCT:
mnem: 1,1 | op1: 2,1 | op2: 4,1 |


№52 | "<empty line>"


№53 |  "    jnz mark2"
Lexeme table:
    Index    Lexeme        Length        Type
    1    JNZ         3         instruction
    2    MARK2         5         identifier
SENTENCE STRUCT:
mnem: 1,1 | op: 2,1 |


№54 |  "    mark2: "
Lexeme table:
    Index    Lexeme        Length        Type
    1    MARK2         5         identifier
    2    :         1        symbol
SENTENCE STRUCT:
label: 1,1 |


№55 |  "    jnz mark2"
Lexeme table:
    Index    Lexeme        Length        Type
    1    JNZ         3         instruction
    2    MARK2         5         identifier
SENTENCE STRUCT:
mnem: 1,1 | op: 2,1 |


№56 | "<empty line>"
```

```
№57 | "    L1:"
Lexeme table:
    Index    Lexeme        Length        Type
    1    L1        2        identifier
    2    :        1        symbol
SENTENCE STRUCT:
label: 1,1 |


№58 | "    jmp l1"
Lexeme table:
    Index    Lexeme        Length        Type
    1    JMP        3        instruction
    2    L1        2        identifier
SENTENCE STRUCT:
mnem: 1,1 | op: 2,1 |


№59 | "    L2:"
Lexeme table:
    Index    Lexeme        Length        Type
    1    L2        2        identifier
    2    :        1        symbol
SENTENCE STRUCT:
label: 1,1 |


№60 | "code ends"
Lexeme table:
    Index    Lexeme        Length        Type
    1    CODE        4        identifier
    2    ENDS        4        segment_ends
SENTENCE STRUCT:
label: 1,1 | mnem: 2,1 |


№61 |
 "END"
Lexeme table:
    Index    Lexeme        Length        Type
    1    END        3        program_end
SENTENCE STRUCT:
mnem: 1,1 |
```

Результат першого перегляду

```
Denys Lashkov KV-33
Listing | First view
   1            DATA SEGMENT
2
3     0000         VAL1 DB 101B
4     0001         VALB1 DB 10
5     0002         VALB2 DB 0AH
6     0003         VALB3 DB 12+12*(6-4)/2
7
8     0004         VAL2 DW 0111B
9     0006         VALW1 DW -16
10    0008         VALW2 DW 45D
11    000a         ABCDE DW 32H
12
13    000c         VAL3 DD 12345678H
14    0010         VALD1 DD 1101B
```

```
15    0014        VALD2 DD 45D
16    0018        VALD3 DD 12+6*(6-4)/4-3
17
18                  DATA ENDS
19
20              CODE SEGMENT
21    0000      CLI
22
23    0001      JMP L2
24    0004         CODEV DW 0ABH
25
26    0006         PUSH CODEV
27    000b         PUSH CS:CODEV
28    0010         PUSH DS:CODEV
29
30    0014         PUSH VAL2
31    0018         PUSH ES:VAL3
32    001e         PUSH DS:VALD3
33
34    0023         BT EDI, EBX
35    0027         BT DI, AX
36
37    002a         LEA DX, ABCDE[SI]
38    002e         LEA BX, VALD1
39    0031         LEA EAX, VAL3[EBP]
40    0039         LEA EDI, VAL2[ESI]
41
42    0041         OUT 10H, AL
43    0043         OUT 20H, AX
44    0045         OUT 30H, EAX
45
46    0048         CMP VAL1, 1
47    004d         CMP VALW2, 0B2CH
48    0053         CMP VALD1, 0DEADH
49    005c         CMP VAL3, 0101B
50    0062         CMP VALD3,12+6*(6-4)/4
51    0068         CMP ABCDE, 32H
52
53    006d         JNZ MARK2
54    0071         MARK2:
55    0071         JNZ MARK2
56
57    0073         L1:
58    0073         JMP L1
59    0075         L2:
60              CODE ENDS
61    0075    END
```

```
Variables list:
Name:    CODEV    Segment:CODE    Offset:  0004
Name:    VAL1    Segment:DATA    Offset:  0000
Name:    VALB1    Segment:DATA    Offset:  0001
Name:    VALB2    Segment:DATA    Offset:  0002
Name:    VALB3    Segment:DATA    Offset:  0003
Name:    VAL2    Segment:DATA    Offset:  0004
Name:    VALW1    Segment:DATA    Offset:  0006
Name:    VALW2    Segment:DATA    Offset:  0008
Name:    ABCDE    Segment:DATA    Offset:  000a
Name:    VAL3    Segment:DATA    Offset:  000c
Name:    VALD1    Segment:DATA    Offset:  0010
Name:    VALD2    Segment:DATA    Offset:  0014
Name:    VALD3    Segment:DATA    Offset:  0018
```

```
Labels:
Name:    MARK2    Segment:CODE    Offset:  0071
Name:    L1    Segment:CODE    Offset:  0073
Name:    L2    Segment:CODE    Offset:  0075

Errors list:
23 Warning: JMP target label 'L2' not defined in the current pass
53 Warning: JNZ target label 'MARK2' not defined in the current pass

Total errors: 2
```

Результат другого перегляду

```
Denys Lashkov KV-33
Listing | Second view

1                                                   DATA segment
2
3  0000 05                                 VAL1 DB 101B
4  0001 0a                                 VALB1 DB 10
5  0002 0a                                 VALB2 DB 0AH
6  0003 18                                 VALB3 DB 12 + 12 * ( 6 - 4 ) / 2
7
8  0004 00 07                              VAL2 DW 0111B
9  0006 ff f0                              VALW1 DW - 16
10  0008 00 2d                              VALW2 DW 45D
11  000a 00 32                              ABCDE DW 32H
12
13  000c 12 34 56 78                        VAL3 DD 12345678H
14  0010 00 00 00 0d                        VALD1 DD 1101B
15  0014 00 00 00 2d                        VALD2 DD 45D
16  0018 00 00 00 0c                        VALD3 DD 12 + 6 * ( 6 - 4 ) / 4 - 3
17
18                                                   DATA ends
19
20                                                   CODE segment
21  0000    fa                                  CLI
22
23  0001    eb 72 90                            JMP L2
24  0004 00 ab                              CODEV DW 0ABH
25
26  0006    2e: ff 36 00 04r                        PUSH CODEV
27  000b    2e: ff 36 00 04r                        PUSH CS:CODEV
28  0010    ff 36 00 04r                        PUSH DS:CODEV
29
30  0014    ff 36 00 04r                        PUSH VAL2
31  0018    66| 26: ff 36 00 0cr                        PUSH ES:VAL3
32  001e    66| ff 36 00 18r                        PUSH DS:VALD3
33
34  0023    66| 0f a3 df                            BT EDI, EBX
35  0027    0f a3 c7                            BT DI, AX
36
37  002a    8d 96 00 0ar                        LEA DX, ABCDE[SI]
38  002e    bb 00 10r                           LEA BX, VALD1
39  0031    66| 67| 8d 85 00 00 00 0cr                LEA EAX, VAL3[EBP
]
40  0039    66| 67| 8d be 00 00 00 04r                LEA EDI, VAL2[ESI]
41
42  0041    e6 10                               OUT 10H, AL
43  0043    e7 20                               OUT 20H, AX
44  0045    66| e7 30                               OUT 30H, EAX
45
```

```
46  0048    80 3e 00 00r 01                 CMP VAL1, 1
47  004d    81 3e 00 08r 0b 2c              CMP VALW2, 0B2CH
48  0053    66| 81 3e 00 10r 00 00 de ad        CMP VALD1, 0DEADH
49  005c    66| 83 3e 00 0cr 05                 CMP VAL3, 0101B
50  0062    66| 83 3e 00 18r 0f                 CMP VALD3, 12+6*(6-4)/4
51  0068    83 3e 00 0ar 32                 CMP ABCDE, 32H
52
53  006d    75 02 90 90                     JNZ MARK2
54  0071                                MARK2:
55  0071    75 fe                           JNZ MARK2
56
57  0073                                L1:
58  0073    eb fe                           JMP L1
59  0075                                L2:
60                                               CODE ends
61  0075                                    END


SYMBOLS TABLE:
VARIABLES:
Name:CODEV      Segment:CODE   Offset: 0004
Name:VAL1       Segment:DATA   Offset: 0000
Name:VALB1      Segment:DATA   Offset: 0001
Name:VALB2      Segment:DATA   Offset: 0002
Name:VALB3      Segment:DATA   Offset: 0003
Name:VAL2       Segment:DATA   Offset: 0004
Name:VALW1      Segment:DATA   Offset: 0006
Name:VALW2      Segment:DATA   Offset: 0008
Name:ABCDE      Segment:DATA   Offset: 000a
Name:VAL3       Segment:DATA   Offset: 000c
Name:VALD1      Segment:DATA   Offset: 0010
Name:VALD2      Segment:DATA   Offset: 0014
Name:VALD3      Segment:DATA   Offset: 0018
LABELS:
Name:MARK2      Segment:CODE   Offset: 0071
Name:L1         Segment:CODE   Offset: 0073
Name:L2         Segment:CODE   Offset: 0075
Errors list:
No errors found.
```

Опис структури програми

Програма розробленого двопрохідного компілятора асемблера використовує головний модуль main.cpp та три функціональні модулі для обробки асемблерного файлу.
Модуль main.cpp імпортує та послідовно
запускає Lexical_analysis, FirstPassListing та SecondPassListing.
Модуль Lexical_analysis (Lexical_analysis.h/.cpp) виконує лексичний аналіз, розбиваючи код на токени, що описуються структурою Lexeme.
Модуль FirstPassListing (FirstPassListing.h/.cpp), імпортуючи Lexical_analysis.h, реалізує перший прохід, будуючи таблиці символів (міток, змінних у структурах Label, Variable) та сегментів (Segment), обчислюючи їх адреси та фіксуючи помилки (AssemblyError).
Нарешті, модуль SecondPassListing (SecondPassListing.h/.cpp),
імпортуючи FirstPassListing.h для доступу до таблиць символів та Lexical_analysis.h, виконує другий прохід, генеруючи машинний код (структура MachineCode) та фіксуючи помилки другого проходу (Error).

Опис програмного коду

Lexical_analysis

- isWhitespaceOnly(const std::string& str): Перевіряє, чи рядок складається лише з пробільних символів (пробіл, таб, новий рядок, повернення каретки).
- toUpperLatin(const std::string& input): Конвертує рядок у верхній регістр латинських символів.
- analyzeLine(std::string& line): Розбиває рядок коду на лексеми (токени), визначає їх тип (інструкція, регістр, константа тощо), довжину та індекс. Перед аналізом конвертує рядок у верхній регістр.
- writeSentenceStructure(std::ofstream& outFile, const std::vector<Lexeme>& lexemes): Аналізує послідовність лексем для визначення структури речення (мітка, мнемоніка, операнди) та записує цю структуру у вихідний файл.
- analyzeFile(const std::string& filename): Читає вхідний файл рядок за рядком, викликає analyzeLine для кожного непорожнього рядка та writeSentenceStructure для запису результатів аналізу у файл lexemes_table.txt.

FirstPassListing

- FirstPassListing(): Конструктор, ініціалізує початкову адресу (address) нулем.
- addError(int lineNumber, const std::string& message): Додає повідомлення про помилку, знайдену на певному рядку, до списку помилок.
- getVariableSegment(const std::string& varName): Повертає назву сегменту, в якому оголошена змінна varName.
- is32BitVariable(const std::string& varName): Перевіряє, чи є змінна varName 32-бітною (типу DD).
- extractVariableName(const std::vector<Lexeme>& lexemes, size_t startIndex): Витягує ім'я змінної (ідентифікатор) з послідовності лексем, починаючи з startIndex.
- analyzeOperandType(const std::vector<Lexeme>& lexemes, size_t startIndex): Аналізує тип операнда (регістр 32/16/8-бітний, сегментний регістр, тип змінної DB/DW/DD) за лексемою на позиції startIndex.
- Блок валідації (validate...):
  - validateInstruction(const std::vector<Lexeme>& lexemes, int lineNumber): Перевіряє коректність операндів для інструкцій.
  - validateVariable(const std::vector<Lexeme>& lexemes, int lineNumber): Перевіряє коректність оголошення змінних (чи не визначена вже, чи є ініціалізатор, чи ім'я не конфліктує з міткою).
  - validateSegment(const std::vector<Lexeme>& lexemes, int lineNumber): Перевіряє коректність оголошення сегментів (чи не визначений вже, чи ENDS відповідає поточному сегменту).
  - validateLabel(const std::vector<Lexeme>& lexemes, int lineNumber): Перевіряє коректність оголошення міток (чи не визначена вже, чи ім'я не конфліктує зі змінною).
- calculateSize(const std::vector<Lexeme>& lexemes): Обчислює розмір в байтах для директив визначення даних (DB, DW, DD) або інструкцій (CLI, JMP, JNZ, PUSH, BT, LEA, OUT, CMP) на основі їх мнемоніки та операндів.
- formatAddress(int address): Форматує цілочисельну адресу у 4-знаковий шістнадцятковий рядок з доповненням нулями.
- processLine(const std::string& line, int lineNumber, std::ofstream& outputFile): Обробляє один рядок коду: аналізує лексеми, визначає тип рядка (оголошення сегменту, змінної, мітки, інструкція), обчислює зміщення (адресу), записує інформацію у вихідний файл та зберігає дані про сегменти, змінні, мітки та помилки.
- generateListing(const std::string& inputFile, const std::string& outputFile): Головний метод першого проходу. Читає вхідний файл, обробляє кожен рядок за допомогою processLine, а потім виводить таблиці змінних, міток та список помилок у вихідний файл.
- printVariablesList(std::ofstream& outputFile): Виводить список всіх визначених змінних (ім'я, сегмент, зміщення) у вихідний файл.

- printLabelsList(std::ofstream& outputFile): Виводить список всіх визначених міток (ім'я, сегмент, зміщення) у вихідний файл.
- printErrorsList(std::ofstream& outputFile): Виводить список всіх помилок, знайдених під час першого проходу, у вихідний файл.

SecondPassListing
- SecondPassListing(const FirstPassListing& fp): Конструктор, приймає результати першого проходу та викликає initializeRegisters.
- initializeRegisters(): Ініціалізує мапу registers кодами для всіх типів регістрів (8, 16, 32-бітних загального призначення та сегментних).
- addError(int lineNumber, const std::string& errorMessage, const std::string& context): Додає повідомлення про помилку другого проходу.
- getErrorCount() const: Повертає кількість помилок, знайдених під час другого проходу.
- printErrorSummary(): Виводить зведення про помилки другого проходу у вихідний файл.
- checkLabelExists(const std::string& label, int lineNumber): Перевіряє, чи існує мітка (визначена під час першого проходу).
- checkVariableExists(const std::string& variable, int lineNumber): Перевіряє, чи існує змінна (визначена під час першого проходу).
- checkVariableRedefinition(const std::string& varName, int lineNumber): Перевіряє, чи змінна varName не перевизначена або чи її ім'я не конфліктує з міткою.
- checkValidRegister(const std::string& reg, int lineNumber): Перевіряє, чи є ім'я регістра reg дійсним.
- validateOperands(const std::vector<Lexeme>& lexemes, int lineNumber): Валідує операнди інструкцій (наприклад, узгодженість розмірів регістрів). Для JMP/JNZ перевіряє існування мітки; для PUSH (з ідентифікатором) – існування змінної; для LEA – валідність регістрів та існування змінних в операнді пам'яті.
- generateListing(const std::string& inputFile, const std::string& outputFile): Головний метод другого проходу. Читає вхідний файл, обробляє кожен рядок за допомогою processLine для генерації машинного коду та лістингу, а потім виводить таблицю символів та звіт про помилки.
- processLine(const std::string& line, int lineNumber): Обробляє один рядок коду під час другого проходу, визначаючи тип рядка (сегмент, змінна, мітка, інструкція) та викликаючи відповідні обробники.
- processSegmentStart(const std::vector<Lexeme>& lexemes): Обробляє директиву початку сегменту, встановлює поточний сегмент та скидає поточну адресу.
- processSegmentEnd(const std::vector<Lexeme>& lexemes): Обробляє директиву кінця сегменту.
- processVariable(const std::vector<Lexeme>& lexemes, int lineNumber): Обробляє оголошення змінної, обчислює її дані (calculateVariableData) та записує машинний код і рядок у лістинг.
- processLabel(const std::vector<Lexeme>& lexemes, int lineNumber): Обробляє оголошення мітки, зберігає її адресу та записує рядок у лістинг.
- processInstruction(const std::vector<Lexeme>& lexemes, int lineNumber): Обробляє інструкцію, генерує для неї машинний код (generateMachineCode) та записує його разом з рядком інструкції у лістинг.
- Блок генерації машинного коду (generate...Instruction):
  - generateMachineCode(const std::vector<Lexeme>& lexemes): Диспетчер, що викликає відповідний метод генерації коду залежно від мнемоніки інструкції.
  - generateCliInstruction(const std::vector<Lexeme>& lexemes): Генерує машинний код для CLI.
  - generatePushInstruction(const std::vector<Lexeme>& lexemes): Генерує машинний код для PUSH.
  - generateBtInstruction(const std::vector<Lexeme>& lexemes): Генерує машинний код для BT.
  - generateLeaInstruction(const std::vector<Lexeme>& lexemes): Генерує машинний код для LEA.

- generateOutInstruction(const std::vector<Lexeme>& lexemes): Генерує машинний код для OUT.
- generateCmpInstruction(const std::vector<Lexeme>& lexemes): Генерує машинний код для CMP.
- generateJmpInstruction(const std::vector<Lexeme>& lexemes): Генерує машинний код для JMP та JNZ, обчислюючи відносне зміщення до мітки.

• getRegisterCode(const std::string& regName): Повертає числовий код для заданого імені регістра.
• getSegmentRegisterPrefix(const std::string& segRegName): Повертає байт-префікс для перевизначення сегментного регістра.
• getImmediateValue(const std::string& immValue): Конвертує рядкове представлення безпосереднього значення (шістнадцяткове, двійкове, десяткове) в ціле число.
• calculateVariableData(const std::string& type, const std::vector<Lexeme>& lexemes, int lineNumber): Обчислює байтове представлення даних для ініціалізації змінних (рядки, числа, вирази).
• evaluateExpression(const std::vector<Lexeme>& lexemes, size_t startIdx, size_t endIdx, int lineNumber): Обчислює значення простого арифметичного виразу, використовуючи стековий алгоритм (shunting-yard).
• formatAddress(int address): Форматує цілочисельну адресу у 4-знаковий шістнадцятковий рядок.
• formatHexByte(unsigned char byte): Форматує байт у 2-знаковий шістнадцятковий рядок.
• formatMachineCode(const std::vector<unsigned char>& bytes, bool need_ref, int maxWidth): Форматує послідовність байтів машинного коду у рядок для виведення в лістинг, додаючи 'r' для посилань.
• printSymbolsTable(): Виводить таблицю символів (змінних та міток з їх адресами та сегментами) у вихідний файл, використовуючи дані першого проходу.

main
• main(): Головна функція програми. Викликає analyzeFile для лексичного аналізу, потім створює об'єкти FirstPassListing та SecondPassListing і послідовно викликає їх методи generateListing для генерації лістингів першого та другого проходів.