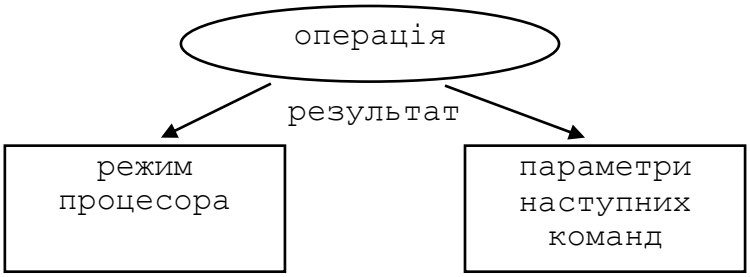
Трансляція і завантаження програм

Схеми виконання команд мікропроцесора

Команда може мати нуль, один, два або три операнди. Загальні правила визначають способи адресування операндів і місце для результату виконання команди.

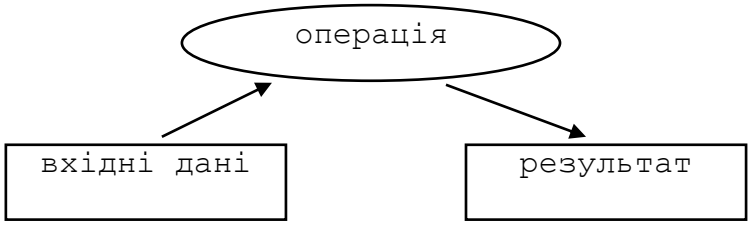
1) Команди без операндів (нуль-операндні). Такі команди призначені для керування режимом роботи цілого процесора або способом виконання наступних команд.



Приклади команд:

REP	префікс; означає повторення рядкової операції наступної команди ECX разів; різновиди: REPZ (REPE) – виконувати так само, поки виконується умова рівності (ZF=1), закінчити повторення, якщо після чергової ітерації ZF=0 (до знайдення іншого значення); REPNZ (REPNE) - так само, але закінчити при ZF≠0 (до знайдення такого самого значення)
CLD	ознака напряму DF регістра ознак встановлюється в нуль; це означає автоматичне збільшення SI і DI на 1 за кожною ітерацією операцій над рядками
STD	DF=1, SI і DI зменшуються на 1 за кожною ітерацією

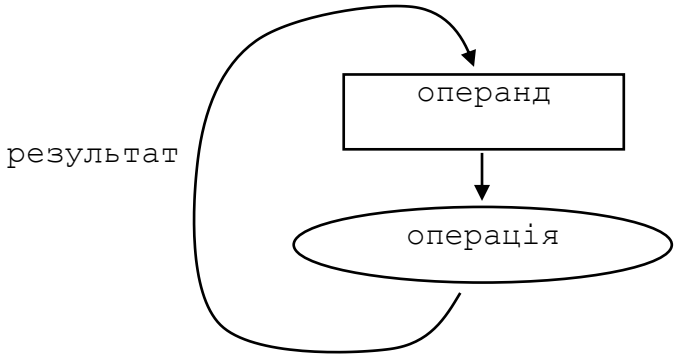
2) Нуль-операндні команди з неявними операндами, визначеними окремо. Такі команди виконують групу елементарних операцій за одне виконання, самі операнди визначають попередньо або зафіксовані командою.



Приклади команд:

XLAT	завантажити в AL байт з таблиці сегменту даних, на початок якої вказує EBX (BX), а початкове значення AL є зміщенням; тобто, це є заміна одного коду іншим за таблицею
PUSHAD	записати в стек регістри EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP
CWD	розширення слова (AX) до подвійного слова (DX:AX) з копіюванням знакового біту
CWDE	розширення слова (AX) до подвійного слова (EAX) з копіюванням знакового біту

3) Команди операцій з одним операндом. Команда опрацьовує заданий операнд відповідно до свого призначення, результат записує на місце операнда. Отже, після виконання команди операнд буде модифікований.



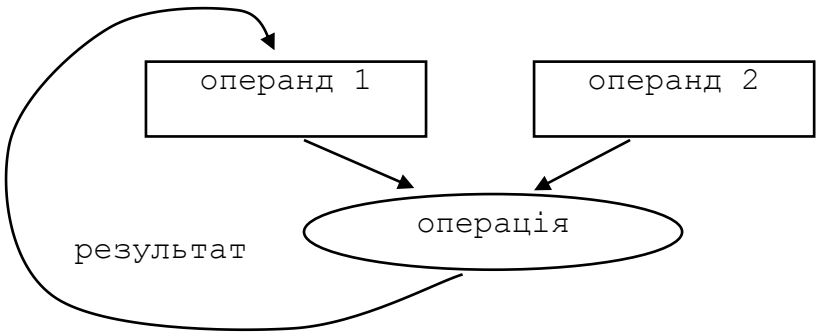
Приклади команд:

INC EAX	інкремент операнда
NEG testvalue	змінити знак операнда

4) Команди дії за одним операндом. Такі команди виконують задану дію, використовуючи операнд. Сам операнд не міняється. Приклади комад:

PUSH EDI	записати в стек значення регістра
POP memx	прочитати з стеку слово чи подвійне слово
JMP target	перейти в програмі до вказаної адреси (різні форми адресування)
JGE / JNL target	перейти, якщо більше або рівне
CALL target	передача керування процедурі за вказаною адресою; в стек зберігається адреса команди, наступної за CALL-командою
INT n	двобайтова команда; спочатку в стек записує регістр ознак, після нього – повну адресу повернення; крім того, скидає ознаку TF; після цього виконує неявний перехід через *n*-й елемент дескрипторної таблиці переривань

5) Команди з двома операндами. Можливі комбінації операндів розглянуто далі. Результат записують на місце першого операнда, отже, він набуває нового значення, тому порядок запису операндів команди є важливий.



Приклади команд:

ADD EDX,alfa	додавання, результат – в регістрі EDX
OR beta,ECX	логічне порозрядне «або», результат – в пам’яті за адресою beta

6) Команди з трьома операндами. Такі команди є лише для окремих різновидів операцій. Результат буде на місці операнда, визначеного структурою команди.

Приклади команд:

IMUL regDWdst, regDWsrc | memDWsrc, immDW	множення зі знаком
regDWdst ← (regDWsrc | memDWsrc) * immDW	
IMUL EAX, EDX, 12 IMUL EAX, [dword BX], 35700	
SHLD / SHRD regDWdest | memDWdest, regDWsrc, count	зсув вліво / вправо
regDWdest ← (regDWdest ∪ regDWsrc) SHLD на count	спочатку об'єднати біти
SHLD [BX], EAX, 2 SHLD [EDI], EDX, CL	

Комбінації операндів в командах

Використаємо такі позначення: R – регістр загального призначення; M – комірка пам'яті, адресована одним з допустимих способів; I – безпосереднє значення, записане в самій команді, тобто, є частиною коду команди. Відповідно до архітектури процесора допускають такі комбінації операндів:

R	один операнд – регістр
M	один операнд – адресована комірка пам'яті
I	операнд є частиною самої команди
R - R	регістр – регістр
R - M	регістр – пам'ять
R - I	регістр – безпосереднє
M - R	пам'ять - регістр
M - I	пам'ять – безпосереднє

Зауважимо, що команд виду «пам'ять»-«пам'ять» архітектурою процесора не передбачено. Тобто, в одній команді не можна адресувати зразу два операнди в пам'яті.

Форми запису констант в командах

1. Двійкова: послідовність цифр 0 і 1, яка закінчується буквою B:
   1 1 0 0 1 0 1 0 B
2. Десяткова: послідовність цифр 0-9, яка може мати суфікс D:
   419 або 419D
3. Шістнадцяткова: послідовність цифр 0-9 і букв латинського алфавіту A-F, яка має обов'язковий суфікс латинською буквою H. Першим символом обов'язково має бути цифра:
   105H 0C04BH
4. Літерал: рядок букв, цифр та інших символів, записаний в лапках або апострофах. Дві форми передбачені для того, щоб вставляти в текст самі лапки чи апострофи:
   "Значення функції: " 'Значення функції: '
   "Відповідь на запитання 'Вибір варіанта?' такий: "
5. Від'ємні числа. Якщо число десяткове, то перед ним записують знак мінус:
   -26

Якщо число двійкове або шістнадцяткове, то його треба записати в доповнювальному коді, тобто вручну зобразити машинний код числа:
для числа -32 : 11100000B або 0E0H

Директиви асемблера

В простішій формі директиви виглядають так:

< ім'я > SEGMENT	визначає ім'я сегмента (частини) програми
< ім'я > ENDS	вказує на кінець сегмента
< ім'я > PROC	фіксує початок процедури
< ім'я > ENDP	фіксує кінець процедури
END < ім'я >	вказує на кінець тексту програми і визначає першу команду програми (точка входу)
ORG < число >	зміна лічильника адреси до вказаного числа, і, відповідно, адреси наступного елемента програми; LOCCTR:= <число>
[< ім'я >] DW < число >	формує двобайтову константу – число; константа може бути іменована, тобто мати власну адресу, або неіменована
[< ім'я >] DW ?	резервує 2 байти (слово) пам'яті
[< ім'я >] DB < число >	формує однобайтову константу - число
[< ім'я >] DB ?	резервує 1 байт пам'яті
[< ім'я >] DB 'рядок'	формує в пам'яті заданий рядок; довжина рядка (кількість байтів) дорівнює кількості літер рядка
[< ім'я >] DD < значення > [< ім'я >] DD ?	аналогічно до вказаного вище, визначає подвійне слово (4 байти)
[< ім'я >] DF < значення > [< ім'я >] DF ?	аналогічно, визначає 6 байтів (адреса в форматі 16-бітний селектор і 32-бітове зміщення
[< ім'я >] DQ <значення> [< ім'я >] DQ ?	аналогічно, визначає почетверене слово (8 байтів)
[< ім'я >] DT < значення > [< ім'я >] DT ?	визначити 10 байтів (80-бітові типи даних, які використовує FPU для операцій з форматами дійсних чисел)

В складнішій формі поле значення може мати декілька чисел, рядків літер, операторів ? і DUP, розділених комами. Всі такі дані будуть записані в пам'ять підряд, а ім'я буде відповідати адресі найпершого значення. Наприклад:

```
table  db   1,2,3,4,5,6,7,8,9,0AH,0BH,0CH,0DH,0EH,0FH
tabtwo  db   512 dup  (?)
```

Трансляція програм за першим і другим переглядом

За <u>першим переглядом</u> транслятор послідовно рядок за рядком переглядає текст програми. При цьому виконує таку роботу: 1) керує лічильником поточної адреси LOCCTR; 2) обчислює довжину кожної команди чи даних в байтах; 3) відстежує адресу (зміщення) кожної команди і даних в сегменті; 4) виконує основне перетворення в машинні коди за кожною командою чи даними; 5) будує таблицю символів SYMTAB з адресами всіх іменованих ділянок пам'яті, які будуть дописані в коди команд за другим переглядом.

За <u>другим переглядом</u> транслятор виконує підстановку обчислених адресів операндів пам'яті, які побудовані в таблиці SYMTAB в результаті першого перегляду.

Далі записані два приклади програм з показом процедури трансляції за першим і другим переглядом.

Приклад 1 (програма з декількох сегментів).

LOCCTR	машинний код	текст	довж. байти	SYMTAB
0000		STACKSG SEGMENT PARA STACK 'Stack'		STACKSG / E
0000	20 [????]	DW 32 DUP (?)	64	
0040		STACKSG ENDS		
0000		DATASG SEGMENT PARA 'Data'		DATASG / E
0000	00FA	FLDA DW 250	2	FLDA = 0
0002	007D	FLDB DW 125	2	FLDB = 2
0004	????	FLDC DW ?	2	FLDC = 4
0006		DATASG ENDS		
0000		CODESG SEGMENT PARA 'Code'		CODESG / E
0000		BEGIN PROC FAR		BEGIN = 0
		ASSUME CS:CODESG, DS:DATASG, SS:STACKSG, ES: NOTHING		
0000	IE	PUSH DS	1	
0001	2B C0	SUB AX,AX	2	
0003	50	PUSH AX	1	
0004	B8 - - - - R	MOV AX,DATASG	3	
0007	8E D8	MOV DS,AX	2	
0009	Al 0000 R	MOV AX,FLDA	3	
000C	03 06 0002 R	ADD AX,FLDB	4	
0010	A3 0004 R	MOV FLDC,AX	3	
0013	CB	RET	1	
0014		BEGIN ENDP		
0014		CODESG ENDS		

Кожний сегмент транслюють окремо. Відлік відносних адресів в межах одного сегмента починають від нуля, якщо немає спеціальних вказівок (ORG тощо). Останнє значення LOCCTR за директовою ENDS є довжиною сегмента.

Адреси і коди записують в шістнадцятковій системі.

Результатом трансляції кожного сегмента є окремий об'єктний файл. Такі об'єктні файли наступним кроком об'єднують в єдиний виконуваний файл за допомогою компонування (linking), що розглянемо далі.

Приклад 2 (COM-програма).

LOCCTR	машинний код	текст	довж. байти	SYMTAB
		.MODEL SMALL		
0000		CODESG SEGMENT DWORD		CODESG / E
		PUBLIC USE16 'CODE'		
0100		ORG 100H		
0100	EB 18 [90 NOP]	BEGIN: JMP MAINPROG	3[2]	BEGIN = 100
0103	54 65 73 74	MESS1 DB 'Testing: print	22	MESS1= 0103
	69 6E 67 3A	a line:'		
	20 70 72 69			
	6E 74 20 61			
	20 6C 69 6E			
	65 3A			
0119	24	DB '$'	1	
011A		MAINPROG PROC NEAR		MAINPROG=011A
011A	B4 09	MOV AH,09	2	
011C	8D 16 0103 R	LEA DX,MESS1	4	
0120	CD 21	INT 21H	2	
0122	E8 012B R	CALL READLINE	3	
0125	E8 0154 R	CALL WRITELN	3	
0128	EB 5B 90	JMP EXIT	3	
012B		MAINPROG ENDP		
		; процедура читання рядка		
012B		READLINE PROC NEAR		READLINE=012B
012B	B4 0A	MOV AH,0AH	2	
012D	8D 16 0134 R	LEA DX,MAXLEN	4	
0131	CD 21	INT 21H	2	
0133	C3	RET	1	
0134		READLINE ENDP		
		; пам'ять та дані для		
		процедури читання рядка		
0134	1E	MAXLEN DB 30	1	MAXLEN= 0134
0135	00	REALLEN DB ?	1	REALLEN=0135
0136	001E [5F]	POLE DB 30 DUP ('_')	30	POLE = 0136
		; процедура вив. на екран		
0154		WRITELN PROC NEAR		WRITELN=0154
0154	B4 09	MOV AH,09	2	
0156	8D 16 0176 R	LEA DX,MESS2	4	
015A	CD 21	INT 21H	2	
015C	. . . . .	; наступні команди	....	

Алгоритм першого перегляду асемблера

```
begin { pass 1 }
 записати 0 в LOCCTR; OKAY:=true
 прочитати перший рядок тексту програми
while(OPCODE<>"END" and OKAY) do begin
  if це не рядок коментар then begin
  if є поле мітки then begin
    пошук мітки в SYMTAB
    if знайшли then begin
      встановити ознаку помилки-повторне визначення
      OKAY:=false
    end
    else занести (< мітка >,LOCCTR) в SYMTAB
  end
  пошук OPCODE в OPTAB
  if знайшли then begin
    аналіз поля операндів
    if формат правильний then begin
      визначення формату та довжини команди
      додати  до LOCCTR довжину команди
    end
    else OKAY:=false (помилка воператорах)
  if OKAY then перевірка наявності та птавильності комантаря
  end { знайшли код операції }
  else if OPCODE="ORG" then LOCCTR:=< число >{ операнд ORG }
  else if OPCODE="DW"  then додати 2 до LOCCTR
  else if OPCODE="DB"  then begin
       аналіз поля операндів DB, визначення довжини
       додати довжину до LOCCTR
       end
  else встановити ознаку помилки - неправильний код операції
     (OKAY:=false)
  end { if це не рядок-коментар }
  записати рядок впроміжний файл
  прочитати наступний рядок тексту програми
end { while }
if not OKAY then сформувати діагностику про помилку
        else begin записати останій рядок в проміжний файл
         запамятати LOCCTR як довжину програми
         end
end { pass1 }
```

Алгоритм другого перегляду асемблера

```
begin { pass2 }
сформувати запис-заголовок в об'єктній програмі
прочитати перший рядок з проміжного файлу
ініціалізувати перший запис тіла програми
OKAY:=true
while(OPCODE<>"END") and OKAY do begin
  if це не рядок-коментар then begin
    пошук OPCODE в OPTAB
    if знайшли then begin
    аналіз поля операндів
    if в полі операндів є імя then begin
      пошук імені в SYMTAB
      if знайшли then взяти з SYMTAB адресу
      else OKAY:=false( не визначене імя )
    end
    перевести команду в об'єктне зображення
    end ( if знайшли )
    else if OPCODE="DW" або "DB" then
          перетворити константу або рядок в об'єктне зображення
    занести об'єктний код в запис тіла програми
    ініціалізувати новий запис тіла програми
  end { if це не рядок-коментар }
  записати у файл для роздруку черговий рядок
  прочитати наступний рядок з промідного файлу
end { while }
if not OKAY then сформувати дфагностику про помилку
else begin
  занести запис-кінець в об'єктну програму
  надрукувати останній рядок в файл роздруку
end
end { pass2 }
```

Загальні принципи компонування

Компонуванням (linking) називають процес створення фізичного або логічного виконуваного файла (модуля) з набору об'єктних файлів і файлів бібліотек для наступного виконання. У разі створення фізичного виконуваного файла компонування називають статичним, в такому файлі є все потрібне для виконання програми. У разі створення логічного виконуваного файла в часі виконання програми компонування називають динамічним, в цьому випадку образ виконуваного модуля будують «на ходу».

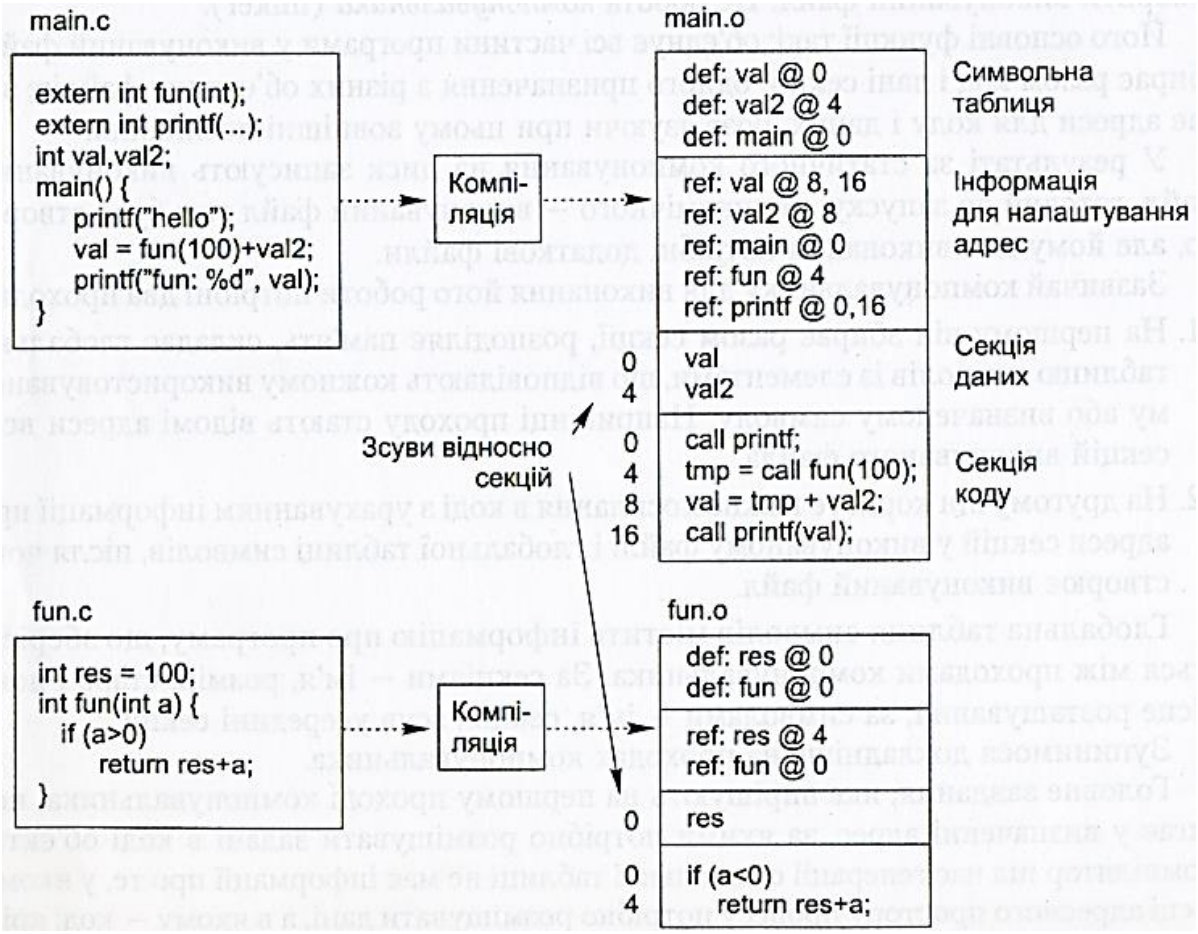
Тут ми розглядаємо статичне компонування виконуваних файлів. Динамічне компонування пов'язане з використанням динамічних бібліотек DLL, це питання будемо розглядати окремим розділом курсу.

Об'єктні файли

Під час компонування виконуваний файл будують з об'єктних файлів (object files), які створює компілятор. Об'єктний файл має: 1) заголовок, де визначено розмір ділянок коду і даних, а також позицію таблиці символів в цьому файлі; 2) об'єктний код (інструкції і дані, згенеровані компілятором), який зазвичай розділений на іменовані ділянки (секції) залежно від призначення; 3) таблицю символів (symbol table), де визначено відносні адреси кожної іменованої комірки пам'яті програми.

Приклад створення об'єктних файлів показано (без заголовку) на рисунку.

Позначення:
`def: name @ offset` – інформація про місце в пам'яті зовнішнього імені `name`;
`ref: name @ offset` – інформація про місце посилання до імені `name`.



Компонувальники і принципи їх роботи

Компілятор не розв'язує зовнішні посилання, а отже, не може створити виконуваний файл. Це робота компонувальника (linker).

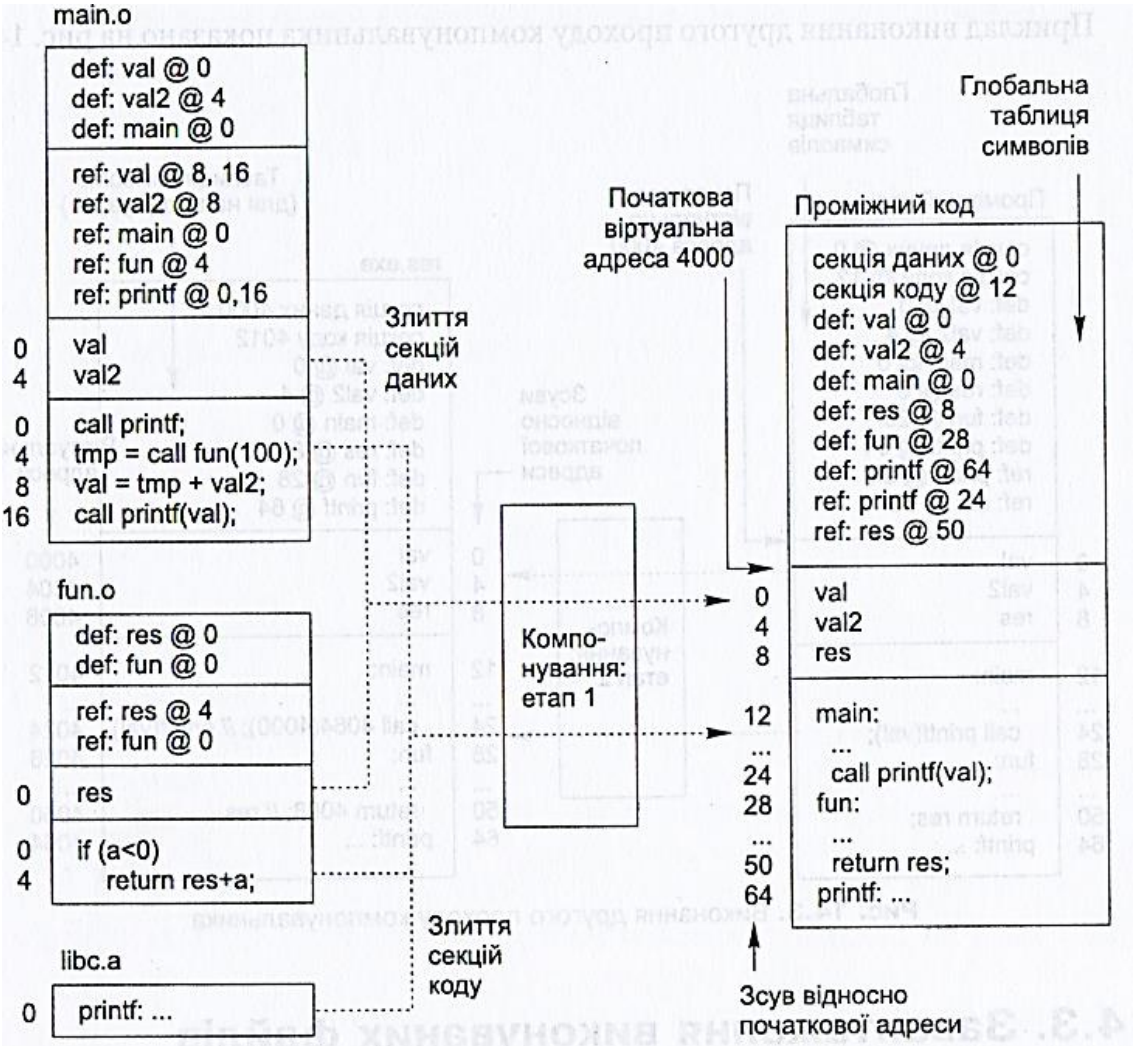
Основні функції компонувальника:

- об'єднати всі частини програми у виконуваний файл;
- зібрати разом код і дані секцій однакового призначення з різних об'єктних файлів;
- визначити остаточно всі адреси для коду і даних, розв'язуючи при цьому зовнішні посилання.

В результаті за статичного компонування на диск записують виконуваний файл, готовий до запуску (EXE-файл), за динамічного – виконуваний файл так само буде створений, але йому для виконання потрібні додаткові файли.

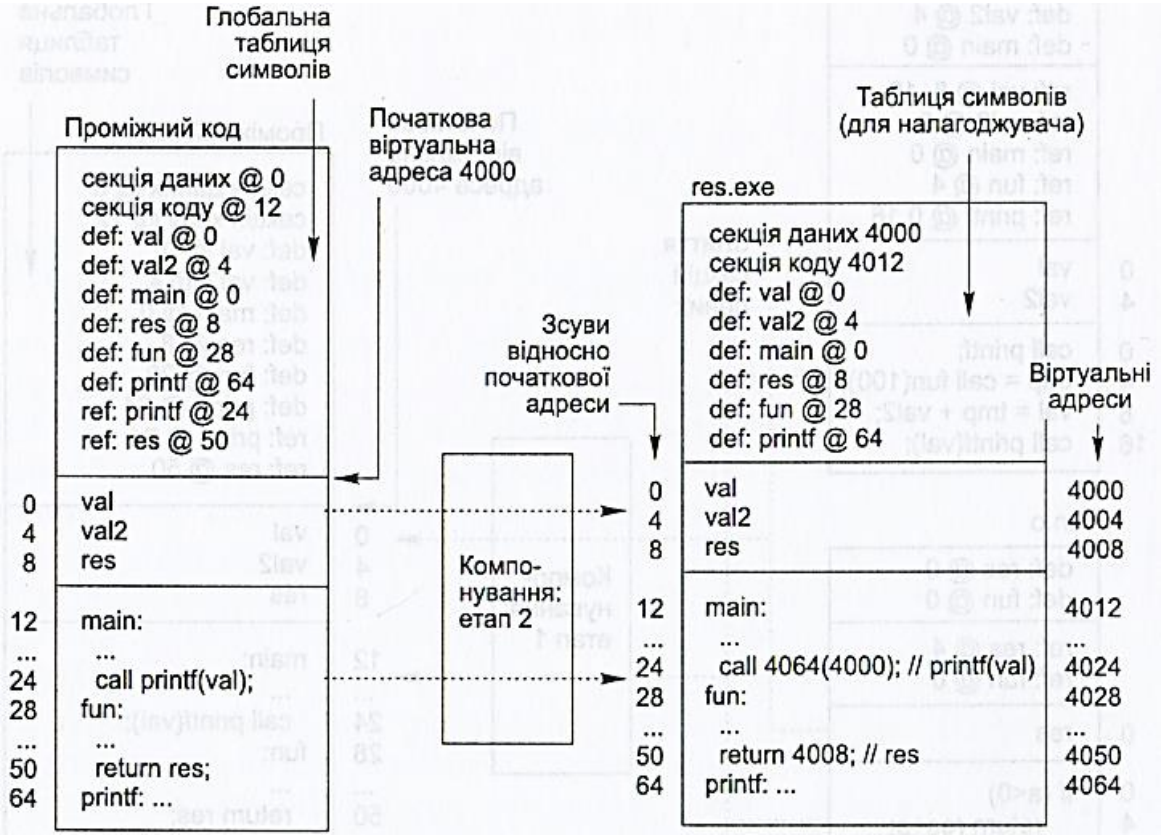
Головне завдання першого проходу компонувальника – визначити відносні адреси, за якими треба розташувати задані в коді об'єкти.

Приклад виконання першого проходу показано на рисунку.



Головне завдання другого проходу компонувальника – корекція всіх адрес в об'єктному коді та розв'язання всіх зовнішніх посилань. Перевіряє, щоб кожний символ мав тільки одне визначення (використати можна багато разів), і виконує корекцію посилань фактичними адресами, з врахуванням отриманої початкової віртуальної адреси.

Приклад виконання другого проходу показано на рисунку.



Завантаження виконуваних файлів за статичного компонування

Виконуваний файл, отриманий внаслідок описаного статичного компонування, має все необхідне для створення процесу. Завантаження такого файла в пам'ять виконує окремий компонент ОС – системний завантажувач. Зазвичай він відображає виконуваний файл в адресний простір процесу та ініціалізує керуючий блок процесу так, щоб процес був в стані готовності до виконання.

Під час відображення виконуваного файла в пам'яті автоматично розташовують його код та ініціалізовані дані. Стек і динамічну ділянку пам'яті зазвичай створюють заново, при цьому для динамічної ділянки компілятор і компонувальник можуть лише задати її початок, а всю інформацію для керування стеком визначає компілятор використовуючи адресування щодо вказівника стеку, який буде встановлений завантажувачем.