**МІНІСТЕРСТВО ОСВІТИ УКРАЇНИ**

##### **ЧЕРНІВЕЦЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ**

**ім. Ю. ФЕДЬКОВИЧА**

ФІЗИЧНИЙ ФАКУЛЬТЕТ

КАФЕДРА ЕОМ

###### СТРУКТУРНЕ ПРОГРАМУВАННЯ

**НА МОВІ АСЕМБЕЛЕРА**

**Курсова робота з курсу “Системне програмування”**

**Студент 3-го курсу**

**Беднарський І.Я.**

Керівник

**Федорук В.І.**

**1999**

**ЗМІСТ**

ЧАСТИНА і

Вступ 4

Глава 1. Структурні оператори і структури логіки керування 5

умови у структурах логіки керування 6

Глава 2. Структура if 5

структура if з чаСткою else 9

функціонування структури if 9

команди, що генеруються операторами $if-$endif 10

команди, що генеруються операторами $if-$else-$endif 10

варіанти операнду 11

операції and i or 11

дальні передачі управління 13

глава 3. структура do 13

структура do until 14

структура do while 15

структура do complex 16

додаткові операнди 16

параметр loop 17

операції and и or 18

глава 4. структура search 18

структура search until 19

структура search while 20

структура search complex 21

додаткові операнди 21

глава 5. обмеження на використання умов ncxz і cxz 23

глава 6. упорядкування структурованих програм 23

процедура 23

розробка структурованих програм по методу “зверху вниз” 24

використання програми salut 25

пакет команд для програми salut 25

переформатування вихідних текстів програмою salut 26

переформатування неструктурованих програм 26

ЧАСТИНА іі

ЛІСТИНГ ПРОГРАМИ ДРУКУ ДИСКОВОГО asCII ФАЙЛУ 28

список використаної літератури 32

СТРУКТУРНЕ ПРОГРАМУВАННЯ

Одним із найцінніших якостей Макроасемблера версії 2 є те, що він дозволяє вам писати структуровані *програми* на мові асемблера. Під «структурованими» ми розуміємо програми, що містять оператори високого рівня і виконуючі цикли, умовні передачі керування та інші задачі керування, що при відсутності оператора зажадали б декількох команд на мові асемблера. У дійсноcті структурні оператори забезпечують ті ж види складаних операцій, що і мова Бейсік. Наприклад, одне із сімейств структурних операторів (або *логічних структур*) виконує ті ж дії, що й оператор IF-THEN-NEXT.

Але структурне програмування зовсім не зводиться до простої заміни команд на структурні оператори. Воно, скоріше, являє собою *філософію* розробки програмного забезпечення, в основу якої покладені такі елементи:

* розробка за методу «зверху вниз». (нагадаємо, що при цьому методі розробка програм починається з коментарів, до яких послідовними кроками додаються все нові і нові деталі).
* Програмування без міток.
* Програмування без оператора GOTO (тобто без команд умовної або безумовної передачі керування).
* Запис тексту програми з відступами, що показують рівень вкладеності логічних структур.
* У достатній мірі лістинги, що самодокументуються.

Все це призначено для підвищення *якості* програм на мові асемблера. Якісні поліпшення повинні бути явними в наступних аспектах:

* розробник повинен краще розуміти логіку програми;
* лістинг повинен бути зрозумілий тим, хто намагається внести власні зміни в чужі програми;
* оскільки програма більш зрозуміла, у ній повинно бути менше помилок;
* підвищується гнучкість і полегшується супровід програми. Це означає, що програма виявиться більш стійкою до зміни зовнішніх ситуацій, ніж це передбачено в проекті, і наступні модифікації органічно ввійдуть в існуючу структуру програми і не внесуть нових помилок;
* програма повинна бути зручною для читання, це допоможе зробити лістинг кращою документацією програми.

Можливість користуватися структурними операторами забезпечується програмою SALUT (Structured Language Utility - сервісна програма структурного програмування на мові асемблера). Ви повинні набрати текст своєї програми, а потім викликати програму SALUT для перетворення структурних операторів у стандартні команди на мові асемблера. Після цього треба як завжди відтранслювати програму і завантажити її.

У цій главі описані структурні оператори програми SALUT і приведені приклади і вказівки по їхньому застосуванню. У ній також обговорюються кроки, що треба виконати для трансляції і форматування структурованих програм.

## СТРУКТУРНІ ОПЕРАТОРИ І СТРУКТУРИ ЛОГІКИ КЕРУВАННЯ

Існує більше десяти різних структурних операторів, але ними треба користуватися в строго визначених сполученнях, що у посібнику з Макроасемблеру називаються *структурами логіки керування*. Нижче визначені вісім різних структур логіки управління, але ми можемо розбити їх на три групи: структура IF (якщо) -- реалізує прийняття рішень Вона змушує мікропроцесор виконувати або пропускати групу команд у залежності від того, задовольняється задана умова чи ні. Структура IF аналогічна оператору IF-THEN на мові Бейсік.

* Структура DO (виконати), що забезпечує повторення групи команд. Вона змушує мікропроцесор циклічно виконувати одну або декілька груп команд, поки не задовольниться задана умова. Структура DO аналогічна операторам FOR-NEXT у мові Бейсік.
* Структура SEARCH (шукати) також пов'язана з повторенням команд, але використовується в тому випадку, коли вам необхідно знати, яка саме з декількох умов привела до завершення виконання циклу. Структура SEARCH виконує свій блок команд для кожної умови виходу з циклу.

# УМОВИ У СТРУКТУРАХ ЛОГІКИ КЕРУВАННЯ

У табл. 1 показані умови, якими можна скористатися для керування структурами IF, DO і SEARCH.

## СТРУКТУРА IF

Структура IF змушує мікропроцесор виконувати або пропускати групу команд в залежності від того, істинна чи хибна задана умова. Її основна форма:

$IF умова

А

$ENDIF

де **умова** -- умова що перевіряється (одна з тих, що перераховані в табл. 1), а А -- блок команд. Якщо **умова** істинна, то мікропроцесор виконує ці команди, у протилежному випадку він переходить до оператора, що слідує безпосередньо за оператором $ENDIF. (звернемо увагу, що цей порядок дій протилежний до того, по якому слідують команди умовної передачі керування: останні *пропускають* команди, якщо умова істинна). На мал. 1, *а* показана блок-схема структури IF.

Наприклад, для перетворення числа, що знаходиться в регістрі BX на його абсолютне значення ми повинні скористатися операторами

CMP BX, 0 ;Від’ємне значення?

$IF L

NEG BX ;Якщо так, змінити знак

$ENDIF

**Таблиця 1. Умови в структурах логіки управління**

|  |  |  |
| --- | --- | --- |
| Умова | Опис | Істинно, якщо … |
| А | Above – вище | CF=0 і ZF=0 |
| AE | Above or Equal - вище або рівне | CF=0 |
| B | Below – нижче | CF=1 |
| BE | Below or Equal - нижче або рівне | CF=1 або ZF=1 |
| C | Carry – перенос | CF=1 |
| CXZ | CX Register is Zero - регістр СХ рівний нулю | (CX)=1 |
| E | Equal – рівне | ZF=1 |
| \*G | Greater – більше | ZF=0 і SF=OF |
| \*GE | Greater or Equal - більше або рівне | ZF=OF |
| \*L | Less – менше | SF<>OF |
| \*LE | Less or Equal - менше або рівне | ZF=1 або SF<>OF |
| NA | Not Above - не вище | CF=1 або ZF=1 |
| NAE | Not Above nor Equal - не вище або рівне | CF=1 |
| NB | Not Below - не нижче | CF=0 |
| NBE | Not Below nor Equal - не нижче або рівне | CF=0 i ZF=0 |
| NC | No Carry - німа переносу | CF=0 |
| NCXZ | CX Register Not Zero - регістр СХ не нуль | (CX)<>0 |
| NE | Not Equal - не рівне | ZF=0 |
| \*NG | Not Greater -- не більше | ZF=1 або SF<>OF |
| \*NGE | Not Greater nor Equal - не більше, не рівне | SF<>OF |
| \*NL | Not Less - не менше | SF=OF |
| \*NLE | Not Less nor Equal -- не менше, не рівне | ZF=O i SF=OF |
| \*NO | No Overflow - німа переповнення | OF=0 |
| NP | No Parity (Odd) - неспівпадання парності (непарне) | PF=0 |
| \*NS[[1]](#footnote-1)\* | No Sign - німа знаку | SF=0 |
| NZ | Not Zero - не нуль | ZF=0 |
| \*O | Overflow – переповнення | OF=0 |
| P | Parity (Even) - співпадання парності (хлопцю) | PF=1 |
| PE | Parity Even – хлопцю | PF=1 |
| PO | Parity Odd – непарне | PF=0 |
| \*S | Sign – знак | SF=1 |
| Z | Zero – нуль | ZF=1 |

# СТРУКТУРА IF З ЧАСТКОЮ ELSE

Як і в мові Бейсік, ви можете вставити ELSE і змусити мікропроцесор виконувати альтернативну групу команд у випадку, якщо умова не істинна. Тоді структура IF буде мати вигляд:

$IF умова

A (здійснюється, якщо умова істинна)

$ELSE

B (здійснюється, якщо умова не істинна)

$ENDIF

На мал. 1,*б* показана блок-схема, що пояснює дію структури $IF-$ELSE-$ENDIF.

# ФУНКЦІОНУВАННЯ СТРУКТУРИ IF

А

Умова

Ні

$IF

Так

$ENDIF

А

Умова

Ні

$IF

Так

$ENDIF

B

$ELSE

Мал. 1. Структури IF (*a*), IF ELSE (*б*)

*a*)

*б*)

Насправді оператори $IF, $ELSE і $ENDIF є командами для програми SALUT. Коли ми викликаємо програму SALUT, то вона перетворює ці оператори в рядки коментарів (так що Асемблер буде їх ігнорувати), а потім створює їхній еквівалент на мові асемблера. Після цього програма може бути відтрансльована Асемблером, тому що операторів $IF, $ELSE і $ENDIF в ній уже немає. У наступних розділах ми покажемо, як програма SALUT перетворить спочатку структуру $IF-$ENDIF, а потім структуру $IF-$ELSE-$ENDIF.

## КОМАНДИ, ЩО ГЕНЕРУЮТЬСЯ ОПЕРАТОРАМИ $IF-ENDIF

Коли програма SALUT виявляє оператор $IF, вона генерує команду умовної передачі управління Jx, що змушує мікропроцесор пропустити оператори, що знаходяться між оператором $IF і наступним оператором $ENDIF. Звернемо увагу на слово «пропустити». Воно повідомляє нам, що програма SALUT повинна підставити команду Jx, *обернену* стосовно змісту операнда в операторі $IF. Отже, вона повинна замінити $IF Z на JNZ, а $IF на JNA. Звичайно, у кожної команди умовної передачі управління повинний бути адресат, і тому програма SALUT уставить мітку після оператора $ENDIF. Наприклад після того, як програма SALUT опрацює оператори $IF і $ENDIF, наш попередній приклад прийме такий вид:

CMP BX,0 ;Негативне значення?

;$IF L

JNL $$IF1

NEG BX ;Якщо так, змінити знак

;$ENDIF

$$IF1:

(Ми вставили мітку $$IF1 тільки для ілюстрації, тому що не знаємо яку мітку програма SALUT вставить насправді).

## КОМАНДИ, ЩО ГЕНЕРУЮТЬСЯ ОПЕРАТОРАМИ $IF-$ELSE-$ENDIF

При опрацюванні структури $IF-$ELSE-$ENDIF програма SALUT замінює оператор $IF на команду Jx, що передає керування мітці, що слідує за $ELSE, заміняє $ENDIF на іншу мітку і підставляє замість $ELSE команду *короткої* передачі керування JMP, що адресується до мітки при $ENDIF Таким чином, як показано на наступному схематичному лістингу, програма SALUT перетворить оператори лівої частини в ті, що зазначені справа.

$IF умова JN умова $$IF1

А

$ELSE JMP SHORT $$EN1

$$IF1:

В

$ENDIF $$EN1:

# ВАРІАНТИ ОПЕРАНДУ

Загальна форма операторів $IF і $ELSE

$IF умова [,AND/OR][,LONG]

$ELSE [LONG]

де операції AND і OR дозволяють Вам указувати додаткові умови, а атрибут LONG забезпечує можливість використання довгих блоків команд у цій структурі.

## ОПЕРАЦІЇ AND І OR

Дотепер ми обговорювали структури IF, що залежать від виконання тільки однієї умови, але (як і в мові Бейсік) ми можемо домогтися того щоб у структурі IF розглядалася комбінація умов у сполученні з операціями AND і OR

Проте між умовами в мові Бейсік і умовами на мові асемблера є визначені розходження. З одного боку, мікропроцесор розглядає кожний член вираження в порядку його появи; тут немає дужок, що задають порядок дій. З іншого боку, для кожного застосування операції AND або OR потрібно окремий оператор $IF, так що між ними можна поміщати команди.

Нижче приводиться загальний вид оператора $IF із двома умовами, сполученими операцією IF:

... (Обчислити умову1)

$IF умову1,AND

... (Обчислити умову2)

$IF умову2

... (Обидві умови виконані)

...

$ENDIF

Врахуємо, що команди обчислення **умови 2** виконуються тільки в тому випадку, якщо **умова 1** істинна. Якщо вона хибна, то мікропроцесор пропустить всі оператори аж до оператора $ENDIF. (Звернемо увагу і на те, що ми використовували тільки один оператор $ENDIF, оскільки операція AND об'єднує оператори $IF в один *складний* оператор.)

За допомогою команд, що слідують за першим оператором $IF, забезпечується установка флагу (або флагів) для другого оператора $IF. Наприклад, для видачі повідомлення "Вага в нормі", якщо значення WEIGHT знаходиться між 31 і 34, треба використовувати оператори

СМР WEIGHT,31

$IF AE,AND

СМР WEIGHT,34

$IF BE

(видати повідомлення "Вага в нормі")

$ENDIF

Звичайно, ми можемо включити сюди і ELSE, щоб виконати визначені дії (наприклад, видачу повідомлення "Вага за межами норми"), якщо одна з умов не виконується.

Нижче приводиться загальний вигляд оператора $IF, в якому умови пов'язані з операцією OR:

... (Обчислити умову1)

...

$IF умова1,OR

... (Обчислити умову2)

$IF умова2

... (умова1 або умова2 виконана)

$ENDIF

Тут мікропроцесор виконає команди обчислення **умови2** тільки в тому випадку, якщо **умова1** виявиться невиконаною.

## ДАЛЬНІ ПЕРЕДАЧІ КЕРУВАННЯ

Через те що програма SALUT заміняє оператори $IF і $ELSE на команди Jx і JMP типу SHORT (тобто команди короткої передачі керування), то розмір блоку команд не може перевищувати 127 байт, тобто біля 50-60 команд, що звичайно не є істотним обмеженням. Проте у випадку, якщо наша структура IF включає багато команд, то треба присвоїти операторам $IF або $ELSE атрибут LONG.

Наприклад, можна вказати

$IF A,LONG

Атрибут LONG змушує програму SALUT замінити оператор $IF А на команди

JA $$XL1

JMP $$IF1

$$XL1:

замість команди JNA $$IF1.

## СТРУКТУРА DO

* Структура DO змушує мікропроцесор повторювати блоки команд до тих пір, поки не виконається задана умова. Існують такі три її форми: структура DO UNTIL (виконувати ... до) повторює блок команд до тих пір, поки умова вкінці цього блока не стане істинною.
* Структура DO WHILE (поки не ..., виконувати) повторює блок команд, поки умова на початку цього блоку не стане істинною. (Іншими словами, вона виконує блок, поки умова не виконується). Якщо початкова умова істинна, то вона пропускає блок, жодного разу не виконавши його.
* Структура DO COMPLEX (виконувати комплексно) являє собою комбінацію структур DO UNTIL і DO WHILE, що включає в себе два блоки команд. Вона виконує перший блок, потім перевіряє умову. Якщо вона істинна, то виконується другий блок і процес повторюється ще раз; у протилежному випадку другий блок пропускається і цикл завершується.

**Примітка.** Слова UNTIL (до), WHILE (поки) і COMPLEX (комплексно) використані лише для того, щоб допомогти розрізнити ці три форми структури DO.

# СТРУКТУРА DO UNTIL

Ця структура має загальний вигляд

$DO

A

$ENDDO умова

де **умова --** це умова припинення циклу. Так як мікропроцесор перевіряє **умову** в кінці циклу, то команди блоку А завжди виконуються не менше одного разу. На мал. 2, *а* показана блок-схема, що описує дію структури DO UNTIL.

Структура DO UNTIL часто використовується для повторення команд в залежності від значення лічильника в регістрі СХ. Наприклад

MOV CX,10

$DO

... (Ці команди будуть повторюватися 10 разів)

...

$ENDDO LOOP

Тут команда LOOP зменшує вміст регістра СХ на одиницю і повертає керування оператору $DO, якщо значення регістра СХ відмінне від нуля. Еквівалентом на мові Бейсік є оператори

FOR X=10 TO 1 STEP -1

...

...

NEXT X

# СТРУКТУРА DO WHILE

Ця структура має загальний вигляд

$DO

$LEAVE умова

A

$ENDDO

де **умова** -- це умова завершення циклу. На мал. 2, б показана блок-схема структури DO WHILE.

Мал. 2. Структури DO UNTIL (*a*), DO WHILE (*б*), DO COMPLEX (*в*)

*a*)

*б*)

Умова

Ні

$DO

Так

$ENDDO

B

$LEAVE

A

Ні

Ні

А

Умова

$DO

Так

$ENDDO

Умова

Ні

$DO

Так

$ENDDO

A

$LEAVE

Ні

*в*)

Звернемо увагу на те, що в структурі DO WHILE **умова** перевіряється перед командами циклу, а не після них, як це відбувається в структурі DO UNTIL. Отже, якщо початкова **умова** істинна, то мікропроцесор обходить цикл, не виконавши його ні разу. Структура DO WHILE ніби сигналізує мікропроцесору: "Виконуй наступні команди, поки **умова** не виконується".

Потрібно користуватися структурою DO WHILE для обходу циклу, організованого за допомогою лічильника в регістрі СХ, якщо регістр СХ вже містить нуль, наприклад

$DO

$LEAVE CXZ

...

...

$ENDDO LOOP

# СТРУКТУРА DO COMPLEX

Структура DO COMPLEX схожа на структуру DO WHILE, але дозволяє Вам виконати деякі дії як до перевірки в операторі LEAVE, так і після її. Тим самим забезпечується вихід із середини циклу. Ця структура має такий загальний вигляд:

$DO

А

$LEAVE умова

У

$ENDDO

Як і раніше, **умова** являє собою умову завершення циклу. На мал.2, *в* показана блок-схема структури DO COMPLEX.

Тут блок команд А підготовляє **умову,** а блок команд В виконує те, що потрібно зробити, якщо **умова** не виконується. Наприклад, для зупинки програми доти, поки користувач не введе правильний пароль, можна скористатися таким прийомом:

$DO

(видати на екран "Будь ласка, введіть ваш пароль:")

(Перевірити пароль. Якщо він правильний, покласти флаг CF

рівним 1.)

$LEAVE CF

(видати на екран "Ви схибили. спробуйте ще раз" )

$ENDDO

# ДОДАТКОВІ ОПЕРАНДИ

Подібно до оператора $IF, що описаний в розд. 2, оператори $ENDDO і $LEAVE допускають вказівку операції AND (И) або OR (АБО), що дозволяє комбінувати умови. Крім того, в них можна вказувати атрибут LONG, що забезпечує можливість користуватися довгими блоками команд. Нарешті, в операторі $ENDDO можна вказувати параметр LOOP, що позбуває від необхідності зменшувати значення регістра СХ. Загальна форма цих операторів

$ENDDO [умова[,AND/OR]][,LONG]

$ENDDO [LOOPx][,LONG]

$LEAVE умова[,AND/OR][,LONG]

## ПАРАМЕТР LOOP

Для циклів із заданим числом повторень, лічильником яких служить регістр СХ, у програмі SALUT можна за рахунок застосування форми $ENDDO LOOP опустити команду DEC СХ. Інакше кажучи, замість форми

$DO

...

...

DEC CX

$ENDDO CXZ

Можна використовувати більш просту форму

$DO

...

...

$ENDDO LOOP

Вона змушує програму SALUT замінити оператор $DO на мітку, а оператор $ENDDO -- на команду LOOP **мітка.** Команда LOOP зменшує значення регістра СХ на 1 і змушує мікропроцесор передати керування мітці, якщо значення СХ не нуль.

Щоб врахувати при ухваленні рішення про закінчення циклу стан флага нуля ZF, користуємося одним із таких операторів:

* Оператор $ENDDO -- зменшує значення регістра СХ і повертає керування на початок циклу, якщо значення регістра СХ не дорівнює 0 і прапор ZF дорівнює 1. Це дозволяє виявляти перший ненульовий результат у серії операцій.
* Оператор $ENDDO LOOPNE зменшує значення регістра СХ і повертає керування на початок циклу, якщо значення регістра СХ не дорівнює 0 і флаг ZF рівний нулю. Це дозволяє виявляти перший нульовий результат у серії операцій.
* Оператор $ENDDO LOOPZ є альтернативною формою оператора $ENDDO LOOPE.
* Оператор $ENDDO LOOPNZ є альтернативною формою оператора $ENDDO LOOPNE.

## ОПЕРАЦІЇ AND І OR

Операція AND завершує цикл, якщо декілька умов виконуються одночасно, а операція OR завершує цикл, якщо виконується *будь-яка* з кількох умов. Наприклад, нижче показана загальна форма структури DO, що завершує цикл, якщо у відповідь на запрошення до вводу користувач відповість "Ні", набравши на клавіатурі "Н" або "н":

$DO

...

...

(Видати на екран ”Продовжити? (Д/Н)”)

(Зчитати відповідь користувача в перемінну ANSWER)

СМР ANSWER,”Н”

$ENDDO E,OR

СМР ANSWER,”н”

$ENDDO E

## СТРУКТУРА SEARCH

Структура SEARCH (шукати) схожа на структуру DO, але передбачає два різних варіанти завершення циклу: *успішний* і *неуспішний.* Це зручно для тих додатків, де потрібно знати, *яка* з умов привела до завершення циклу. Наприклад, Вам може бути необхідно дізнатися, завершилася операція вводу-виводу успішно чи була припинена через виникнення помилки.

Як і структура DO, структура SEARCH має три форми:

* Структура SEARCH UNTIL (шукати ... до) повторює блок команд доти, поки одна з двох умов в кінці цього блока не стане істинною.
* Структура SEARCH WHILE (шукати, поки не) повторює блок команд, поки обидві умови помилкові. Якщо одна з них початково істинна, то блок команд не виконується жодного разу.
* Структура SEARCH COMPLEX (шукати комплексно) являє собою комбінацію структур SEARH UNTIL і SEARCH WHILE, що включає в себе блок команд до перевірки умов і блок команд після них. Вона виконує перший блок, потім перевіряє умови. Якщо хоча б одна з них істинно, то цикл завершується. У противному випадку здійснюється другий блок команд і процес повторюється ще раз.

# СТРУКТУРА SEARCH UNTIL

Ця структура має загальний вигляд

$SEARCH-

А (Основний блок)

$EXITIF умова1

В (Якщо умова1 істинна, то виконати і

Вийти з циклу)

$ORELSE

С (Якщо умова1 хибна, то виконати) $ENDLOOP умова2

D (Якщо умова2 істинна, то виконати і

Вийти з циклу)

$ENDSRCH

де **умова1 і умова2** -- умови завершення циклу. Тут **умова1** відповідає успішному завершенню (Ви знайшли те, що шукали), а **умова2** -- неуспішному. На мал. 3, *a* показана блок-схема структури SEARCH UNTIL.

Врахуйте, що основний блок А завжди виконується принаймні один раз. Блоки В і D виконуються відповідно при успішному і неуспішному завершенні операцій (Ви можете користуватися ними для видачі повідомлень), а блок С містить команди, що встановлюють значення флагів для перевірки оператором $ENDLOOP.

Наприклад, в екзаменаційній програмі можна скористатися структурою SEARCH UNTIL і дати студенту три спроби вибрати з запропонованих йому варіантів відповіді правильний. Її застосування може мати такий загальний вид:

MOVE СХ,3 ;Встановити лічильник

(Зобразити на екрані питання)

$SEARCH

(Прочитати відповідь студента)

(Встановити ZF == 1, якщо відповідь правильна)

$EXITIF Z

(Зобразити на екрані "Поздоровляємо, Ви праві” )

$ORELSE

(Зобразити на екрані "Вибачте, Ви помилились, спробуйте знову")

$ENDLOOP LOOP

(Зобразити на екрані 'Всі три відповіді неправильні')

(Зобразити на екрані "Правильна відповідь:" і

вказати відповідь”

$ENDSRCH

# СТРУКТУРА SEARCH WHILE

Ця структура має загальний вигляд

$SEARCH

$LEAVE умова1

А (Якщо умова1 хибна, то виконати)

$EXITIF умова2

В (Якщо умова2 істинна, то

Виконати і вийти з циклу)

$OREELSE

C (Якщо умова2 хибна, то виконати)

$ENDLOOP

D (Якщо умова1 істинна, то виконати і

вийти з циклу)

$ENDSRCH

Через те, що структура SEARCH WHILE здійснює перевірку до виконання інших операторів, то її можна використовувати для обходу всієї структури, якщо початкові умови не виконані. Наприклад, що наступний фрагмент здійснює пошук у таблиці першого нульового значення і обходить пошук, якщо таблиця порожня:

MOV СХ,TABLE LENGTH ;Встановити лічильник

MOV ВХ,0 ;Зупинити вказівник на перший елемент

$SEARCH

$LEAVE CXZ

СМР ТABLE[ВХ],0 Порівняти з нулем

$EXITIF E

(Зобразити на екрані Знайдено нуль )

$OPELSE

INC ВX

$ENDLOOP LOOP

(Зобразити ”Нулів немає”)

$ENDSRCH

# СТРУКТУРА SEARCH COMPLEX

Ця структура має загальний вигляд

$SEARCH

A

$ЕХIТIF умова1

В (Якщо умова1 істинна, то виконати і

вийти з циклу)

$ORELSE

С (Якщо умова1 хибна, то виконати)

$LEAVE умова2

D (Якщо умова2 хибна, то виконати) $ENDLOOP

Е (Якщо умова2 істинна, то виконати і вийти з циклу)

$ENDSRCH

На мал. 3, *в* показана блок-схема структури SEARCH COMPLEX. Звернемо увагу на те, що структура SEARCH COMPLEX аналогічна структурі SEARCH UNTIL, але виконує додатковий *блок.* D, якщо **умова 2** хибна.

# ДОДАТКОВІ ОПЕРАНДИ

Деякі оператори структури SEARCH можуть мати додаткові операнди типу тих, що описувалися для структур IF і DO. Вони мають такі загальні форми:

$EXITIF умова[,AND/OR][,LONG][,NUL]

$ORELSE [LONG]

$ENDLOOP [умова[,AND/OR]][,LONG]

$ENDLOOP [LOOPx],[LONG]

$LEAVE умова[,AND/OR][,LONG]

Єдиний новий операнд -- NUL, що можна використовувати в сполученні з оператором $EXITIF у ситуації, коли в структурі SEARCH треба пропустити блок В *(успішний пошук).* Звичайно, якщо блока В немає, то можна пропустити й оператор $ORELSE. Структура функціонує і в тому випадку, якщо блок В порожній, але вказівка операнда NUL прискорює її виконання.

**Таблиця 2. Допустимі використання умов NCXZ і CXZ**

|  |  |  |
| --- | --- | --- |
| **Структурний оператор** | **NCXZ** | **CXZ** |
| $IF умова | Так | Ні |
| $IF умова, OR | Ні | Так |
| $IF умова, AND | Так | Ні |
| $LEAVE умова | Ні | Так |
| $LEAVB умова, OR | Ні | Так |
| $LEAVE умова, AND | Так | Ні |
| $EXITIF умова | Так | Ні |
| $EXITIF умова, AND | Так | Ні |
| $EXITIF умова, OR | Ні | Так |
| $EXITIF умова, AND, NUL | Так | Ні |
| $EXITIF умова, OR, NUL | Ні | Так |
| $EXITIF умова, NUL | Ні | Так |
| $ENDLOOP умова | Так | Ні |
| $ENDLOOP умова, AND | Так | Ні |
| $ENDLOOP умова, OR | Ні | Так |
| $ENDDO умова | Так | Ні |
| $ENDDO умова, AND | Так | Ні |
| $ENDDO умова, OR | Ні | Так |

## ОБМЕЖЕННЯ НА ВИКОРИСТАННЯ УМОВ NCXZ І CXZ

Як згадувалося вище, програма SALUT транслює структурні оператори в їхні еквіваленти на мові асемблера. Зокрема, вона перетворить умовні оператори в команди Jx або JMP, а оператори $ENDIF, $DO, $SEARCH і $ENDSRCH -- у мітки. При цьому програма SALUT у багатьох випадках повинна перетворювати умовний оператор у протилежну команду умовної передачі керування. Наприклад, вона перетворить оператор $IF А в команду JNA L1 (де L1 -- мітка, що стоїть на місці оператора $ENDIF). Але через це при застосуванні умови CXZ можуть виникнути проблеми, оскільки команди JNCXZ немає. У табл. 2 перераховані ті оператори, в яких можуть використовуватися умови CXZ і NCXZ, і вказано, в яких випадках ці умови допустимі.

## УПОРЯДКУВАННЯ СТРУКТУРОВАНИХ ПРОГРАМ

# ПРОЦЕДУРА

При впорядкуванні структурованих програм можна користуватися тими ж прийомами, що і при впорядкуванні звичайних програм, із тією лише різницею, що текст програми треба опрацювати програмою SALUT для перетворення структурних операторів у стандартні команди на асемблері. Крім того, програма SALUT буде формувати вашу вихідну програму, розташовуючи елементи операторів по визначених стовпчиках. Таким чином, треба виконати такі кроки:

1. Набрати текст програми за допомогою редактора. Дати йому ім'я виду **ім'я\_файла. SAL,** де SAL -- абревіатура від Structured Assembly Language (структурована мова асемблера).
2. Опрацювати файл типу SAL програмою SALUT. В результаті отримається файл з форматованою вихідною програмою **ім'я\_файла. SAL** і проміжний робочий файл **ім'я\_файла. ASM.** програма SALUT перейменує створений вами файл з неформатованою програмою в **ім'я\_файла. BAK.**
3. Відтранслюйте проміжний робочий файл за допомогою Асемблера.
4. Знищити проміжний робочий файл типу ASM. При бажанні можна видалити і файл типу BAK.
5. Скористайтеся завантажувачем для створення виконуваного модуля.

## РОЗРОБКА СТРУКТУРОВАНИХ ПРОГРАМ МЕТОДОМ «ЗВЕРХУ ВНИЗ»

Відповідно до методу розробки “зверху вниз” програма формується послідовною вставкою деталей у її початковий текст (що складається частіше всього лише з коментарів). Цей метод можна застосувати і для розробки структурованих програм. Для цього спочатку потрібно набирати керуючі оператори логічних структур, а потім вставляти команди, що повинні бути між ними. Щоб побудувати, наприклад, структуру IF, треба почати з операторів

$IF ;Якщо вказане правильне значення

$ELSE ;Тому що вказане неправильне значення

$ENDIF ;Кінець перевірки значення на правильність

Ця конструкція поки ще не може бути відтрансльована належною чином, оскільки в операторі $IF треба вказувати умову. Додаючи умову, отримуємо

$IF E ;Якщо вказане правильне значення

$ELSE ;Тому що вказане неправильне значення

$ENDIF ;Кінець перевірки значення на правильність

тепер ми маємо умову “дорівнює”, котрій повинне передувати порівняння значень. Вставляючи це порівняння, отримуємо

CMP AX,100 ;Перевірити на рівність із необхідним

Значенням

$IF E ;Якщо вказане правильне значення

$ELSE ;Тому що вказане неправильне значення

$ENDIF ;Кінець перевірки на правильність

Звернемо увагу на те, що тепер ми одержали щось прийнятне як для трансляції, так і для виконання. Тому ми на якийсь час можемо залишити цей фрагмент програми, а пізніше повернутися до нього для того, щоб заповнити “істинну” і “хибну” частини перевірки. Зрештою, ескіз, що ми одержали, цілком придатний для документування призначення цього фрагмента, а деталі можна додати і пізніше.

Важливим моментом цього прикладу є та обставина, що *коментарі* в тексті з'явилися одночасно з командами. Найкраще документувати програму в момент її написання, оскільки саме в цей час ви найкраще розумієте її.

# ВИКОРИСТАННЯ ПРОГРАМИ SALUT

Для виклику програми SALUT одержите на екрані запрошення до вводу С>, а потім введіть команду типу

С>с:salut ім'я\_файлу

Коли програма SALUT завершить свою роботу, у робочій директорії з’являться три файли: початкова вихідна програма **ім'я\_файлу.ВАК,** переформатована версія вихідної програми **ім'я\_файлу.SAL** і проміжний робочий файл **ім'я\_файлу.ASM.** Тепер можна звичайним чином відтранслювати файл **ім'я\_файлу.АSМ,** а потім завантажити об'єктний модуль **ім'я\_файлу.ОВJ** і одержати виконуваний модуль **ім'я\_файлу.ЕХЕ.**

## ПАКЕТ КОМАНД ДЛЯ ПРОГРАМИ SALUT

Взагалі говорячи, після завершення роботи програми SALUT уже не потрібний файл із початковою вихідною програмою (з розширенням ВАК). А після завершення трансляції не потрібний і файл із розширенням ASM (якщо в програмі були помилки, то Вам треба виправляти не файл із розширенням ASM, а файл із розширенням SAL.) Тому потрібно видалити ці два файли. Щоб не робити це щоразу вручну, створіть *пакет команд,* що викликає програму SALUT і Макроасемблер, а потім видаляє файли з розширенням ВАК і ASM. Для цього (припускаючи, що ви хочете дати пакету команд ім'я ASMSAL.BAT) наберіть зазначені нижче рядки:

C>copy con: asmsal.bat

c:salut %1

erase %1.bak

c:masm %1,,;

erase %1. asm (натисніть клавішу F6)

Для виконання цього пакета введіть команду виду

C> c:asmsal ім'я\_файлу

Врахуйте, що третя команда пакета, на жаль, автоматично створює файл із лістингом вихідної програми (із розширенням LST). Якщо він не потрібний, створіть інший пакет команд, указавши команду с:masm % 1; замість цього рядка.

# ПЕРЕФОРМАТУВАННЯ ВИХІДНИХ ТЕКСТІВ ПРОГРАМОЮ SALUT

Якщо задаються інші параметри, то програма SALUT буде переформатовувати вихідну програму в такий спосіб:

1. Початковою позицією *міток* і *самостійних коментарів* буде стовпчик 1.
2. Початковою позицією *структурних операторів* і *мнемокодів* команд буде стовпчик 9.
3. Початковою позицією *операндів* буде стовпчик 17.
4. Початковою позицією *коментарів* буде стовпчик 41.
5. *Всередині структур* оператори зміщуються на чотири позиції вправо. При бажанні можна задати інші початкові позиції.

## ПЕРЕФОРМАТУВАННЯ НЕСТРУКТУРОВАНИХ ПРОГРАМ

Потрібно врахувати що в програми SALUT функція переформатування відділена від функції опрацювання структур. Отже, якщо ми і не користуємося структурами, проте можемо викликати програму SALUT для переформатування своєї програми, написаної на мові асемблера. Для цього потрібно набрати текст програми як звичайно (скажемо, розділяючи елементи операторів одним пропуском або символом табуляції), потім зберегти його у файлі і ввести команду виду

С>с:salut ім'я\_файлу.asm, ім'я\_файлу. asm, nul

Програма SALUT переформатує файл **ім'я\_файлу.АSМ** і перейменує неформатовану версію в **ім'я\_файлу.ВАК.**

**ЛІСТИНГ ПРОГРАМИ ДРУКУ ASCII ФАЙЛІВ print**

; ДРУК ДИСКОВИХ ASCII ФАЙЛІВ

; ТЕХНIЧНI ЗАСОБИ, ЩО ВИКОРИСТОВУЮТЬСЯ:

;Програма виконується на ПК сумiсному з IBM PC базової конфiгурацiї ;монiтор, клавiатура).Мiнiмальнi вимоги: процесор фiрми Intel 8086 i ;старший , наявнiсть 3,5"дисковода або наявнiсть HDD, наявність EPSON-;сумісного матричного принтера.

;Програмне забезпечення: базова версія DOS.

; Для використання програми необхiдно запустити файл print.exe

;і на запрошення до вводу ввести повний шлях до файлу, який потрібно ;роздрукувати. Якщо задати тільки ім’я файлу, то програма шукатиме ;його в текучому каталозі.

ideal

model small

stack 256

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dataseg

filenam db 32 dup(' ') ;список параметрiв для

maxlen db 32 ;вводу

namelen db 0 ;iменi файлу

sector db 512 dup(' ') ;область вводу файлу

disarea db 120 dup(' ') ;область виводу

count dw 00

endcde dw 00

handle dw 0

openmsg db '\*\*\*Open error\*\*\*',10,13

prompt db 'Name of file? '

;-------------------------------------------------------------------

codeseg

start:

mov ax,@data

mov ds,ax

mov es,ax

call clrscr ;очистити екран

call cursor ;встановити курсор

loop1:

mov [endcde],00 ;початкова установка

call inp\_filename ;получити iм"я файлу

cmp [namelen],00 ;є запит?

je a90 ;нi - вийти

call open\_file ;вiдкрити файл, встановити DTA

cmp [endcde],00 ;помилка при вiдкриттi?

jne a80 ;так - повторний запит

call readsector ;прочитати 1-й сектор

cmp [endcde],00 ;кiнець файлу, нема даних

je a80 ;так - повторити запит

call printing ;роздрукувати сектор

a80:

jmp loop1

a90: jmp buy

; П/програма запиту iменi файлу:

; ------------------------------

proc inp\_filename

mov ah,40h ;функцiя виводу на екран

mov bx,01

mov cx,13

lea dx,[prompt]

int 21h

mov ah,0ch

mov al,0ah ;функцiя вводу з клвiатури

lea dx,[filenam]

int 21h

mov bl,[filenam+1] ;записати

xor bh,bh ;00 в кiнець

mov [filenam+bx+2],0 ;iменi файлу

mov [byte namelen],bl

c90: ret

endp inp\_filename

; Вiдкриття дискового файлу:

; --------------------------

proc open\_file

mov ah,3dh ;функцiя вiдкриття

mov al,00 ;тiльки читання

lea dx,[filenam+2]

int 21h

jnc e20 ;перевiрити ознаку CF

call errormsg ;помилка, якщо встановлена

ret

e20:

mov [handle],ax ;зберегти номер файлу

lea di,[sector]

mov ax,2020h

mov cx,256 ;очистити пропусками

rep stosw ;область сектора

ret

endp open\_file

; Пiдготовка i друк даних:

; --------------------------

proc printing

cld ;напрям злiва направо

lea si,[sector] ;початкова установка

g20:

lea di,[disarea]

mov [count],00

g30:

lea dx,[sector+512]

cmp si,dx ;кiнець сектора?

jne g40

call readsector ;так - читати наступний

cmp [endcde],00 ;кiнець файлу?

je g80 ;так - вийти

lea si,[sector]

g40:

mov bx,[count]

cmp bx,80 ;кiнець областi виводу?

jb g50 ;нi - обiйти,

mov [Word bx+di],0d0ah ;так - записати CR/LF

call print

lea di,[disarea] ;початок областi вивлду

g50:

lodsb ;записати [SI] в AL, збiльшити SI

mov bx,[count]

mov [di+bx],al ;записати символ

inc bx

cmp al,1ah ;кiнець файлу

je g80 ;так - вийти

cmp al,0ah ;кiнець строки?

jne g60 ;нi - обiйти

call print ;так - друкувати

jmp g20

g60:

cmp al,09h ;символ табуляцiї

jne g70

dec bx ;так - встановити BX:

mov [byte di+bx],20h ;замiнити ТАВ на пропуск

and bx,0fff8h ;обнулити правi 8 бiт

add bx,08 ;i додати 8

g70:

mov [count],bx

jmp g30

g80: mov bx,[count] ;кiнець файлу

mov [byte di+bx],0ch ;прогон сторiнки

call print ;друкувати останню строку

g90: ret

endp printing

; Пiдпрограма друку:

; ------------------

proc print

mov ah,40h ;функцiя друку

mov bx,04

mov cx,[count] ;довжина

inc cx

lea dx,[disarea]

int 21h

mov ax,2020h ;очистити область виводу

mov cx,60

lea di,[disarea]

rep stosw

ret

endp print

; Пiдпрогама зчитування сектора:

; ------------------------------

proc readsector

mov ah,3fh ;функцiя читання з файлу

mov bx,[handle] ;номер файлу

mov cx,512 ;довжина

xor dx,dx

mov dl,[offset sector] ;буфер

int 21h

mov [endcde],ax

ret

endp readsector

; Прокрутка екрану:

; -----------------

proc clrscr

mov ax,0600h

mov bh,0fh ;встановити колiр

mov cx,0000 ;прокрутка (скролiнг)

mov dx,184fh

int 10h

ret

endp clrscr

; Пoдпрограма встановлення курсора:

; ---------------------------------

proc cursor

mov ah,02 ;функцiя установки

mov bh,00 ;курсора

mov dx,00

int 10h

ret

endp cursor

; Вивiд повiдомлення про помилку:

; -------------------------------

proc errormsg

call clrscr

call cursor

mov ah,40h ;функцiя виводу на екран

mov bx,01 ;номер

mov cx,18 ;довжина

lea dx,[openmsg] ;адрес повiдомлення

int 21h

mov [endcde],01 ;ознака помилки

ret

endp errormsg

buy:

call clrscr

call cursor

mov ah,04ch

mov al,[byte endcde]

int 21h

end start

**СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ:**

1. **П. Абель Язык Ассемблера для IBM PC и программирование /Пер. с англ. Ю.В. Сальников.– М.:Высш. шк., 1992.–447с.:ил.**
2. **Л. Скэнлон Персональные ЭВМ IBM PC и ХТ. Програмирование на языке Ассемблера.–М.: Радио и связь.–1989.**

1. \* використовується при виконанні операцій над числами із знаком (в додатковому коді). [↑](#footnote-ref-1)