ДОДАТОК А

ВКАЗІВКИ ДО ВИКОРИСТАННЯ МІКРОАСЕМБЛЕРУ

А.1. Трансляція мікропрограми

Мнемонічний двопроходний мікроасемблер призначений для розробки мікропрограм.

Результатом роботи мікроасемблера є файл даних з розширенням «**.pmk*», який є вихідним для програмного емулятора системи на рис. А.1.

Процес трансляції вихідного файлу здійснюється за два проходи. Під час першого проходу відбувається визначення обсягу вихідного файлу, формуються таблиці міток і відповідностей, а також проводиться попередній синтаксичний аналіз. Під час другого проходу безпосередньо формуються коди мікрокоманд. У випадку виявлення синтаксичної або семантичної помилки в вихідному тексті процес трансляції припиняється з видачею повідомлення про характер помилки і рядка тексту, на якому вона була виявлена.

Вихідним файлом для мікроасемблеру є текстовий файл в кодах *ASC*II з розширенням «**.asm*». Розходження між заголовними і малими літерами мікроасемблером не сприймаються. Між окремими мнемоніками може бути будь-яке число службових символів, наприклад, пробіл, табуляція, повернення каретки, переклад рядка і таке інше.

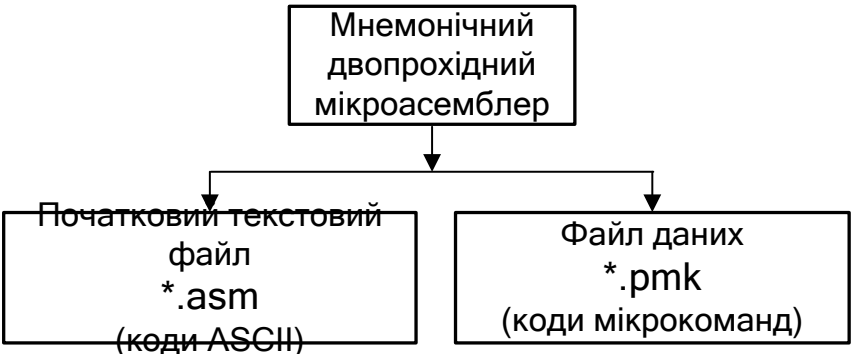


Рис. А.1. Процес трансляції мікропрограми

Строге дотримання правил написання мікропрограми, акуратність в наборі тексту прискорюють трансляцію і налагодження. Більшість помилок виникає насамперед через недбалий стиль написання і неточне знання самого об'єкта розробки.

Приклад текстового файлу (в кодах *ASCII*), який є початковим для мнемонічного двопрохідного мікроасемблера:

```
link l1:ct                          \ завдання зв’язків
accept r1:00ffh                     \ завантаження регістрів
org 100h                            \ розміщення МП у ПМК
macro inc reg:{add reg,reg,z,nz;}   \ створення макрокоманди
                                    \ інкременту
accept ra:12                        \ завантаження номеру
                                    \ регістру АЛП у RA
{add r10,ra;}                       \ R10:=R10 + R12
inc r1                              \ інкремент регістру R1
equ znach:125                       \ параметру znach
                                    \ присвоюється значення
                                    \ 125
{sub r1,znach,nz;}                  \ R1:=R1 – ZNACH
```

Коментарі

Коментарі використовуються для пояснень. Ознакою початку коментарю є символ «\ ». Далі мікроасемблер ігнорує всі символи, які зустрічаються, до наступного «\» або до кінця рядка.

Наприклад:

```
{add r11,r11,r10,z;} \додати до вмісту R11 вміст R10
```

Числові константи

Числові константи застосовуються під час завдання значень операндів і адрес. Ознакою константи є цифра на початку мнемоніки.

Наприклад:

```
\способи завдання констант у різних системах числення
65535              \десяткова константа
0FFFFh             \шістнадцатирічна константа
177777o            \вісімкова константа
1111111111111111%  \двійкова константа
```

Мітки

Мітки включають до 10 символів (букв, цифр та символів «_»), причому першим символом не повинна бути цифра. Ознакою кінця мітки служить будь-який зазначений далі розділювач: пробіл, повернення каретки, переведення рядка, табуляція. Мітка не повинна збігатися з зарезервованою мнемонікою. Приклади написання міток:

Наприклад:

```
\застосування міток
ll1 {or sll r1,r1,0;}   \логічний зсув вмісту регістру вліво
    {cjp not rm_z,ll1}  \якщо вміст регістру не дорівнює
                        \повторюємо зсув
    {add r1,r1,r3,z}    \додавання
    {cgp nz, ll1}       \безумовний перехід на мітку
```

Мнемонічний запис мікрокоманд

Арифметичні мікрокоманди, що виконуються в АЛП, записуються в вигляді

```
<мнемоніка>(<оператор_зсуву>,)(<приймач_результату>,)
      <джерело_1>,<джерело_2>,<вхідний_перенос>
```

Тут і далі в круглих дужках вказуються необов'язкові елементи конструкцій.

Запис логічних мікрокоманд відрізняється від арифметичних відсутністю операнду вхідного переносу, тому що перенос не бере участь в логічних мікрооперацiях.

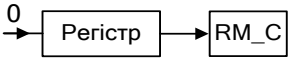
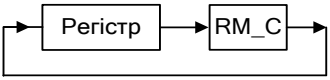
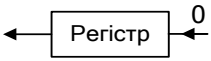
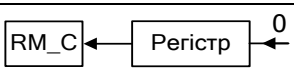
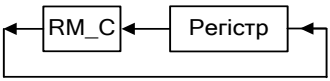
Мнемоніка арифметичних і логічних мікрокоманд зазначена в табл. А.1. Як приймач результату може бути зазначений кожний з регістрів *R*0 – *R*15, а також `nil`, коли результат в НОЗП не записується, але може бути виданий на локальну шину через БУ. Якщо приймач результату не вказується, то результат міститься на місце першого джерела операндів. Джерелами операндів можуть бути два регістри НОЗП, а також один регістр (він указується як перше джерело операндів) в комбінації з константою, `bus_d` або нулем. Нуль в полі джерела операнда позначається буквою `z`. Регістри НОЗП можуть адресуватися непрямо. Якщо в якості джерел операндів зазначені *RA* та/або *RB*, то операнди вибираються з регістрів, коди яких записані в *RA* і *RB*. Вхідний перенос може

приймати значення 0, 1 (записується відповідно через `z` і `nz`), а також `rm_c` і `not rm_c`. Мнемоніки операторів зсуву зазначені в табл. А.2.

Таблиця А.1. Мнемоніка мікрооперацій в АЛБ

Мнемоніка	Мікрооперація в АЛБ
`add`	$R + S + CI$
`sub`	$R - S - 1 + CI$
`or`	*R or S*
`and`	*R and S*
`nand`	*not*(*R and S*)
`xor`	*R xor S*
`nxor`	*not*(*R xor S*)

Таблиця А.2. Мнемоніка операторів зсуву

Мнемоніка	Найменування зсуву	Розряди МК $I_{10} - I_6$	Схема зсуву
`sra`	Зсув вправо арифметичний	00010	
`srl`	Зсув вправо логічний		
`sr.9`	Зсув вправо з переносом	01001	
`sla`	Зсув вліво арифметичний	10010	
`sll`	Зсув вліво логічний	10000	
`sl.25`	Зсув вліво з переносом	11001	

Приклади мікрокоманд:

```
{add slr,r10,r2,z;}  \ R10 := 0.r[R1 + R10]
{xor r5,r5,r5;}      \ встановлення в нуль вмісту регістру R5
{SUB r7,r7,bus_d,nz;} \ віднімання з вмісту регістру R7
                      \ даних, що надходять з ЛШ
{add r9,z,nz;}        \ інкремент регістру R9
```

А.2. Директиви мікроасемблера для блока обробки даних

Директиви мікроасемблера – це службові мнемоніки, які не транслюються в мікрокоманди мікропрограми.

Вони служать для задання початкових значень в регістрах, початкової адреси мікропрограми в пам‘яті мікрокоманд, для настроювання окремих вузлів обчислювальної системи тощо.

`accept` – ***директива занесення інформації в регістри БОД***

Загальний вигляд директиви:

```
accept <регістр>: <значення>
```

де `<регістр>`: *R*0, *R*1,…, *R*15, *RQ*,
POH,
RM, *RN*,
RA, *RB*.

Таким чином директива дозволяє задати `<значення>`:

- в будь-якому з регістрів АЛП,
- одночасно у всіх регістрах АЛП,
- значення ознак в регістрах *RM* та *RN* СУСЗ,
- значения в регістрах *RA*, *RB* обрамлення БОД.

Наприклад,

```
accept r1:0affh      \ R1:=AFFH
accept r1:0affh      \ R1:=AFFH
accept rq:12o        \ RQ:=12o
accept rm:1001%      \ RM:=1001%
accept rb:10         \ RB:=10
```

Директива

```
accept poh:<16 значень>
```

дає можливість задати значення в регістрах загального призначення АЛП. Після `poh` необхідно задати 16 значень, перше з яких завантажується в регістр *R*0, останнє – в регистр *R*15 АЛП.

Наприклад:

```
accept poh: 0,1,2,3,4,5,6,7,8,9,0ah,0bh,0ch,0dh,0eh,0fh
```

Наведемо приклади фрагментів мікропрограм із застосуванням директиви `accept`:

Приклад:

```
accept r2:123       \ завантажити значення 123 в R2
accept r10:375o     \ завантажити 8-кове значення 375 в R10
{sub r2,r10,nz;}    \ R2:=R2 – R10
```

Приклад:

```
accept rb:9         \ завантажити значення 9 в RB
{add rb, bus_d;}    \ виконати додавання вмісту регистру,
                    \ номер якого завантажений у RB із
                    \ значенням з локальної шини
                    \ R9:=R9+bus_d
```

`link` – *директива приєднання регістрів RA, RB до ЛШ*

Загальний вигляд директиви:

```
link <регістр>: <4 номери розрядів ЛШ>
```

де `<регістр>` – регістри *RA* або *RB* (рис. А.2).

Директива `link` вказує, номери розрядів 16-розрядної локальної шини які мають бути приєднані до виводів 4-розрядних регістрів *RA, RB* обрамлення БОД.

Наприклад, наступні директиви приєднають виводи *RA* до розрядів молодшої тетради локальної шини, а *RB* до розрядів третьою тетради ЛШ:

```
link rb:3,2,1,0
link ra:11,10,9,8
```

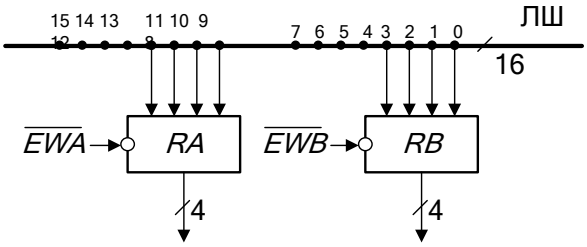


Рис. А.2. Приєднання регістрів *RA, RB* БОД до локальної шини

`load` – ***директива завантаження регістрів RA, RB з ЛШ***

Загальний вигляд директиви:

```
load <регістр>
```

де `<регістр>` регістри *RA* або *RB*.

Директива `load` означає, що в регістр (*RA* або *RB*) буде завантажено значення, яке в даний час перебуває на ЛШ.

Приклади застосування директиви `load`:

```
load ra
load rb
```

Приклад:

Фрагмент мікропрограми з використанням директив `accept`, `link`, `load`:

```
accept r3:0a2d4h      \ R3:=A2D4H
link rb:7,6,5,4       \ виводи регістру RB будуть
                      \ поєднані з розрядами другої
                      \ тетради ЛШ
{and nil,r3,00f0h;oey;} \ ЛШ:=00D0  A3C4
                      \ виконується мікрооперація
                      \ підсумовування вмісту
                      \ регистру R3 із константою,
                      \ результат не фіксується, але
                      \ видаться на локальну шину (oey).
load rb               \ RB:=D  C
                      \ в регістр RB завантажується номер
                      \ регістра АЛП – R13
{sub rb,r10, z;}      \ R13:=R13 – R10 – 1
                      \ виконується мікрооперація
                      \ віднімання. Регістр RB адресує
                      \ перший з операндів. Місце
                      \ розташування результату задано
                      \ неявно. Результат записується за
                      \ місцем розташування першого
                      \ операнду.
```

Мікрокоманди управління регістрами RM* та *RN

Для завантаження регистрів *RM, RN* СУСЗ застосовуються наступні мікрокоманди

```
{load rm, z;}      \ встановлення всіх розрядів RM в нуль
{load rm, nz;}     \ встановлення всіх розрядів RM в одиницю
{load rm,flags;}   \ завантаження всіх ознак сформованих
                   \ під час виконання мікро операції в АЛП
{load rn,flags;}   \ завантаження ознак у регістр RN
```

Одночасно з зазначеними мікрокомандами застосовуються мікрокоманди заборони зипису у відповідні розряди *RM*, а саме `cem_c, cem_z, cem_n, cem_v`.

Приклад:

```
{load rm,flags;cem_v;cem_n;} \ завантаження тільки
                             \ розрядів rm_c та rm_z.
{load rm,flags;cem_c;}       \ завантаження всіх ознак
                             \ окрім rm_c.
```

`org` – ***директива розміщення виконуваного коду мікропрограми в пам'яті мікрокоманд***

Загальний вигляд директиви:

```
org <мітка>
org <адреса>
```

Директива `org` розміщує виконуваний код мікропрограми в ПМК за вказаною адресою.

Приклади застосування директиви `org`:

```
org 20h
org start
```

Якщо мікропрограма не містить директиви `org`, вона розміщується в ПМК за адресою `000`.

`equ` – ***директива задання відповідності***

Загальний вигляд директиви:

```
equ <ім'я>:<значення>
```

Директива `equ` використовується для присвоєння символічним іменам, що використовуються у мікропрограмі, конкретних числових значень.

Приклади застосування директиви `equ`:

```
equ start:100
equ op1:2150
equ op2:0afh
```

`macro` – ***директива створення макрокоманд***

Загальний вигляд директиви:

```
macro<ім'я><формальні_параметри>:{<мікрокоманда>;}
```

Директива `macro` дозволяє конструювати власні мнемоніки операцій (макрокоманди) і користуватися ними надалі як стандартними.

Приклади застосування директиви `macro`:

```
macro inc reg:{add reg,reg,z,nz;}
macro dec reg:{sub reg,reg,z,z;}
macro mov reg1,reg2:{or reg1,reg2;}
```

Приклад:

```
inc r2          \ R2:=R2 +1
mov r10,r6      \ R10:=R6
dec rq          \ RQ:=RQ – 1
```

Ім'я макрокоманди надалі стає для транслятора звичайною стандартною мнемонікою. В якості імен формальних параметрів макроса не можуть бути застосовані зарезервовані мнемоніки. У мікропрограмі макрокоманда задається своїм власним ім'ям (мнемонікою) та реальними операндами, в тому ж самому порядку, в якому вони вказані в макросі.

`include` – ***директива приєднання файлу***

Загальний вигляд директиви:

```
include <имя_файла>
```

Файл, що указується в директиві повинен знаходитись в одному і тому самому каталозі, що транслюємий.

Приклад:

```
include macro.lib
include routine
```

А.3. Директиви мікроасемблера для блока мікропрограмного управління

`link` – *директива встановлення відповідності між входами МУ та логічними умовами*

Якщо в системі мікрокоманд схеми формування адреси мікрокоманди ФАМ, як умови використовуються сигнали на входах мільтиплексора умов МУ – *L*1, *L*2, …, *L*6 (або *not L*1, *not L*2, …, *not L*6), то сигнал умови необхідно зв‘язати з одним із входів зазначеного мультиплексора (рис. 1) за допомогою директиви `link`. Під час аналізу *Li* (де $i = \overline{1,6}$) як логічної умови в структурі мікрокоманди у частині призначеній для управляння ФАМ поле *MS* буде містити двійковий код відповідний до номеру входу умови – *i*.

Загальний вигляд директиви:

```
link <ім‘я_входу>:<умова>
```

де <ім‘я входу> відповідає *L*1, …, *L*6.

Приклади застосування директиви `link`:

```
link l2:rdm
link l3:no
link l4:ct
```

До входів *L*1, …, *L*6 МУ можна приєднувати наступні управляючі сигнали:

```
zo, co, no, vo
rm_z, rm_c, rm_n, rm_v
rn_z, rn_c, rn_n, rn_v
ct
rdm, rdd
int
irq0, ..., irq7
```

Завдання відповідності між входами МУ та умовами за допомогою директиви `link` може здійснюватись як окремо для кожного входу МУ, так і для всіх входів одночасно за допомогою наступної директиви

```
link l:<шість_умов>)
```

Приклад:

```
link l:ct,int,irq0,irq2,irq5,irq7
```

За цього до входів мультиплексора умов приєднаються відповідні управляючі сигнали *L*1:=*CT*, *L*2:=*INT*, *L*3:=*IRQ*0, *L*3:=*IRQ*2, *L*3:=*IRQ*5, *L*6:=*IRQ*7.

`accept` – *директива встановлення схеми ФАМ в початковий стан*

Директива `accept` дозволяє встановити в початковий стан наступні вузли ФАМ:

```
accept sp:<значення>          – покажчик стека
accept stack:< значення >     – комірки стека
accept rac:< значення >       – регістр адреси / лічильник циклів (РА/ЛЦ)
accept pcmk:< значення >      – лічильник мікрокоманд
```

Приклад:

```
accept sp:003                 \встановлення покажчика стека у значення 3
accept stack[2]:0afdh         \записати в комірку 2 стека
                              \значення AFDH
accept stack:0a00h,0b00h,0c00h,2d0h,0d00h \5 значень
                              \заносяться в комірки стека
                              \комірка за коміркою
                              \розпочинаючи з верхівки
                              \стека
accept rac:5                  \в РА/ЛЦ записати значення 5
accept rcmk:10                \початковий стан ЛМК – 10.
```

`link m` – *директива приєднання Буфера М до ЛШ*

Дванадцяти розрядний Буфер *М* поєднує 16-розрядну локальну шину та 12-розрядну шину адреси розгалуження (рис. А.3). Інформацію з ЛШ на ШАР можна подавати у довільному порядку послідовності бітів.

Директива `link m` задає відповідність між розрядами ЛШ та розрядами Буфера *М*.

Наприклад наступна директива

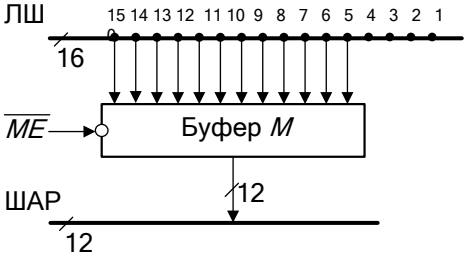
```
link m:15,14,13,12,11,10,9,8,7,6,5,4
```

з'єднає дванадцять старших розрядів ЛШ із входами Буфера М (рис. А.3, *а*).

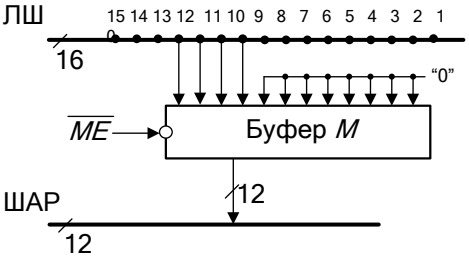
Директива

```
link m: 12,11,10,9,z,z,z,z,z,z,z,z
```

з'єднає 12, 11, 10, 9 розряди ЛШ із старшими чотирма входами Буфера М, інші розряди Буфера М завантажаться нулями (рис. А.3, *б*).



а



б

Рис. А.3. Поєднання локальної шини з Буфером *М*:
а, б – різні варіанти поєднання

`link v` – *директива налагоджування Буфера V*

Директива `link v` задає відповідність між управляючими сигналами, що поступають на вхід Буфера *V* та розрядами Буфера *V*.

На основі цих сигналів під час виконання мікрокоманди

```
{cjv cond;}
```

формується початкова адреса мікропрограми в ПМК, яка саме через Буфер *V* надходить в ФАМ.

На вхід Буфера *V* (рис. А.4) подаються наступні сигнали

`irq0, ..., irq7`	–	запити на переривання від зовнішніх пристроїв,
`ct`	–	сигнал логічної умови,
`int`	–	сигнал вимоги загального переривання,
`rdm, rdd`	–	сигнали готовності пам‘яті та зовнішнього пристрою відповідно (у вигляді мнемонік ці сигнали позначаються без інверсії),
`z, nz`	–	сигнали “0” та “1”.

Наприклад наступна директива

```
link v:z,nz,irq0,irq1,irq2,irq3,irq4,irq5,irq6,irq7,ct,int
```

під’єднає зазначені управляючі сигнали до дванадцяти входів Буфера *V* (рис А.4).

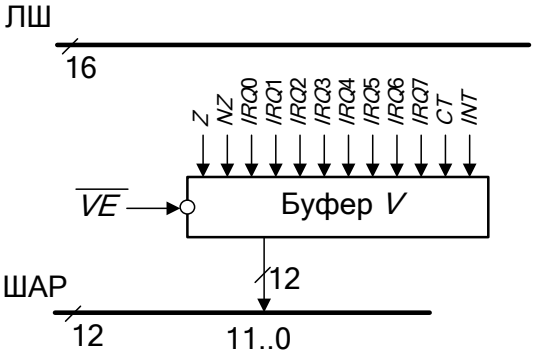


Рис А.4. Приєднання Буфера *V* до шини адреси розгалуження

А.4. Директиви роботи із пам’яттю та зовнішніми пристроями

dw - ***директива завдання значень комірок пам'яті має вигляд***

```
dw <адреса>:<значення>
```

Приклад:

```
dw 12:0ffh
dw 03Fh:15
```

Мікрокоманди управління зовнішніми пристроями

Мнемоніка мікрокоманд управління зовнішніми пристроями, пам'яттю, регістром адреси і буфером БУ збігається з найменуванням відповідного управляючого сигналу:

`r` – мікрокоманда читання з пам'яті;

`w` – мікрокоманда запису в пам'ять;

`i` – мікрокоманда вводу даних із зовнішнього пристрою;

`o` – мікрокоманда виводу в зовнішній пристрій;

`ewh`, `ewl` – мікрокоманди запису відповідно в старші і молодші розряди регістра адреси;

`oey` – мікрокоманда видачі результату *Y* з АЛБ на локальну шину.

Можна задати такі характеристики зовнішніх пристроїв (`dev`):

– тип пристрою (`in` - введення, `out` – виведення);

– адреса регістра стану (РС) в межах 64К (*max* 0*FFFFh*);

– адреса регістра даних (РД) в межах 64К (*max* 0*FFFFh*);

– затримка в тактах формування сигналу `rdd` (*max* 0*FFFFh*);

– затримка в тактах установки біта «Готовність» в регістрі стану (*max* 0*FFFFh*).

Приклад:

```
accept dev[2]:in,30h,32h,3,114
```

де `in` - пристрій вводу даних;

`30h` - адреса РС;

`32h` – адреса РД;

`3` – затримка сигналу *RDM* в тактах;

`114` – затримка установки біта готовності в РС після звертання до РД;

Для пристроїв введення можна задавати внутрішній буфер даних `dev_buf`, обсягом до 16 слів.

Приклад:

```
accept dev_buf[2]:1234h,5678h,89abh,0eeeeh
```

Зазначені після символу «:» дані вводяться в процесор один за одним під час кожного звертання до РД даного пристрою введення.

Директива `accept` дозволяє задати швидкодію пам'яті за допомогою змінної `rdm_delay`.

Приклад:

```
accept rdm_delay: 3
```

В даному випадку сигнал *RDM* буде формуватися з затримкою на 3 такти після видачі на шину керування сигналу *R* або *W*.

Для опису конфігурації зв'язків між компонентами системи використовується директива `link`.

Для установки відповідності входів *L*1, *L*2, *L*3 БМК і логічних умов використовується директива

```
LINK <ім'я_входу>:<умова>
```

Приклад:

Для забезпечення зв'язків БМУ з пам’яттю, пристроями вводу/виводу та блоком обробки даних необхідно записати:

```
link L1:rdm
link l2:rdd
link l3:ct
```

Підключення двадцятирозрядного регістру адреси РАД до шістнадцятирозрядної ЛШ описується директивою:

```
link ewh : <номер_розряду>
```

Номер розряду РАД[19..0], зазначений у директиві, розділяє регістр на дві частини. Старша частина, включаючи зазначений розряд, управляється сигналом *EWH*, а молодша – *EWL*.

Приклад:

```
link ewh : 16
```

Директива набудовує зв'язки так, що за сигналом *EWH* чотири молодших розряди з ЛШ записуються в поле РАД[19..16]. За сигналом *EWL* всі шістнадцять розрядів з ЛШ записуються в РАД[15..0], тобто в молодшу частину регістра.

В один момент часу в різних вузлах системи можуть виконуватися різні мікрооперації. Всі мікрокоманди, що управляють мікрооперaціями, які виконуються в одному такті, записуються в операторних дужках { }, утворюючи повну мікрокоманду для даного такту роботи ЕОМ. Окремі мікрокоманди розділяються символом « ; », окрім того, під час запису мікрокоманди можуть використовуватися роздільники типу «пробіл», «повернення каретки», «переведення рядка». Повна мікрокоманда може займати кілька рядків в тексті мікропрограми. За необхідності мітка записується зліва перед операторною дужкою, що відкривається.

А.5. Приклади розробки мікропрограм

Приклад:

Встановити нулі в чотирьох старших розрядах РАД і записати в молодші розряди цього регістра адресу з *R*7. Прийняти слово з ОП в регістр *R*15. Сигнал готовності *RDM* формується рівнем логічного нуля.

```
\Область налагодження зв’язків
     link l1:rdm
     link ewh:16      \установка зв'язків між РАД і ЛШ
\Визначення затримки формування RDM
     accept rdm_delay:2
\-----------------------------------------------
\Область завантаження регістрів
\вихідна установка r7 (для налагодження)
     accept r7:1234h
\-----------------------------------------------
\Область завантаження пам’яті
\Завантаження даних в ОП за адресою 1234h
     dw 1234h:070fh
\-----------------------------------------------
\Область програми
```

```
     {cont;xor nil,r0,r0;oey;ewh;}     \РАД[19...16]:=0
     {cont;or nil,r7,z;oey;ewl;}       \РАД[15...0]:=r7
\Завантеження даних з ОП у регістр R15:=070fh
ll1  {cjp rdm,ll1;r;or r15,bus_d,z;}
     {}                              \кінець мікропрограми
```

Мікроінструкція БМУ `cont`, під час виконання якої відбувається формування адреси наступної мікрокоманди за інкрементом лічильника мікрокоманд (тобто відбувається лінійний перехід до наступної адреси), може бути безпосередньо не указана у мікрокоманді. При цьому формування наступної адреси відбувається за замовчуванням (див. розділ 2.5, табл. 2.27).

Приклад:

Підсумувати коди в регістрах *R*1,*R*2 і *R*15. Записати подвоєний результат в пам'ять за адресою, яка записана в регістрі, зазначеному в *RA*.

```
\Область налагодження зв'язків
     link l1:rdm
     link ewh:16
\Визначення затримки формування RDM
     accept rdm_delay:3
\-----------------------------------------------
\Область завантаження регістрів
     accept ra:3
     accept r3:0004h
     accept r1:4
     accept r2:16
     accept r15:32
\-----------------------------------------------
\Область програми
     {xor nil,r0,r0;oey;ewh;}        \РАД[19...16]:=0
     {add r1,r1,r2,z;}                    \ R1:=R1+R2
     {add sla,r1,r1,r15,z;}          \R1:=l(R1+R15).0
     {or nil,ra,z;oey;ewl;}       \запис адреси в РАД
\запис результату в пам'ять
ll2  {cjp rdm,ll2;w;or nil,r1,z;oey;}
     {}                              \кінець мікропрограми
```