«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки

АРХІТЕКТУРА КОМП'ЮТЕРІВ - 3. МІКРОПРОЦЕСОРНІ ЗАСОБИ

Методичні вказівки до виконання лабораторних робіт

для підготовки бакалавра

спеціальності 123 Комп'ютерна інженерія

кафедри обчислювальної техніки

денної та заочної форми навчання

ЧАСТИНА 1

Рекомендовано
кафедрою обчислювальної
техніки
ФІОТ КПІ ім. Ігоря Сікорського
Протокол № 1 від 30.08.2017 р.

Київ

КПІ ім. Ігоря Сікорського

2017 – 2018

Архітектура комп'ютерів 2. Процесори. Методичні вказівки до виконання лабораторних робіт. [Текст] / Уклад.: В.І.Жабін, В.В. Ткаченко, І.А. Клименко – К.: НТУУ «КПІ», 2017. – 59 с.

Методичні вказівки призначені для підготовки бакалаврів спеціальності 123 Комп'ютерна інженерія кафедри обчислювальної техніки всіх форм навчання. Наведено завдання та методичні вказівки до виконання лабораторних робіт і курсового проектування з кредитного модуля «Архітектура комп'ютерів – 2. Процесори» навчальної дисципліни «Архітектура комп'ютерів», теоретичні відомості, питання для самоконтролю, список необхідної літератури.

Укладачі: В.І. Жабін, д.т.н., професор
В.В. Ткаченко, к.т.н., доцент
І.А. Клименко, д.т.н., доцент

Рецензент: Теленик С.Ф., доктор технічних наук, професор, завідувач кафедри автоматики і управління в технічних системах

За редакцією укладачів

ЛАБОРАТОРНА РОБОТА №1

ВЗАЄМОДІЯ ПРОЦЕСОРА З ОСНОВНОЮ ПАМ'ЯТТЮ. СПОСОБИ АДРЕСАЦІЇ ОПЕРАНДОВ БЕЗ ВИКОРИСТАННЯ РОН.

Мета роботи: Изучить основные принципы взаимодействия процессора с общей памятью в системах с объединенными и разделенными шинами адреса и данных. Изучить алгоритмы основных циклов обращения процессора к ОП. Изучить способы адресации операндов без использования РОН.

Теоретичні відомості []

Додаткові теоретичні відомості

Структурная организация ВС

Для изучения основных принципов взаимодействия процессора (П) и основной памяти (ОП) рассмотрим фрагмент вычислительной системы (рис. 1.1), основні характеристики якої наведені далі.

Системна шина (СШ) - разділена шина адреси (ША) и шина даних (ШД);

Основна пам'ять (ОП) об’ємом $2\text{М} = 2 \times 2^{20} = 2^{21}$. Минимальная информационная единица составляет 16-розрядное слово, то есть доступ к отдельному байту невозможен. Другими словами в системе реализована *выборка данных словами*.

При использовании нумерации байтов справа-налево адрес слова в памяти соответствует адресу младшего байта (рис.1.2). Формирование адреса (А) следующего слова выполняется по формуле $\text{A}_{i+1} = \text{A}_i + 2$, где значение 2 соответствует ширине выборки данных – 2 байта.

Таким образом необходимо адресовать 1М 16-розрядных слов или $2\text{М} : 2 = 2^{21} : 2 = 2^{20}$ слов. Для передачи 2^{20} слов потребуется 20-разрядная ША.

ОП виконує дві мікрооперації: запис слова за заданою адресою (ініціюється сигналом $\overline{W}$ (*Write*)) та читання слова (ініціюється сигналом $\overline{R}$ (*Read*)). Мнемонічно позначатимемо ці операції як {w;} та {r;} відповідно.

Разрядность шины данных 16 разрядов.

Разрядность шины адреса 20 разрядов.

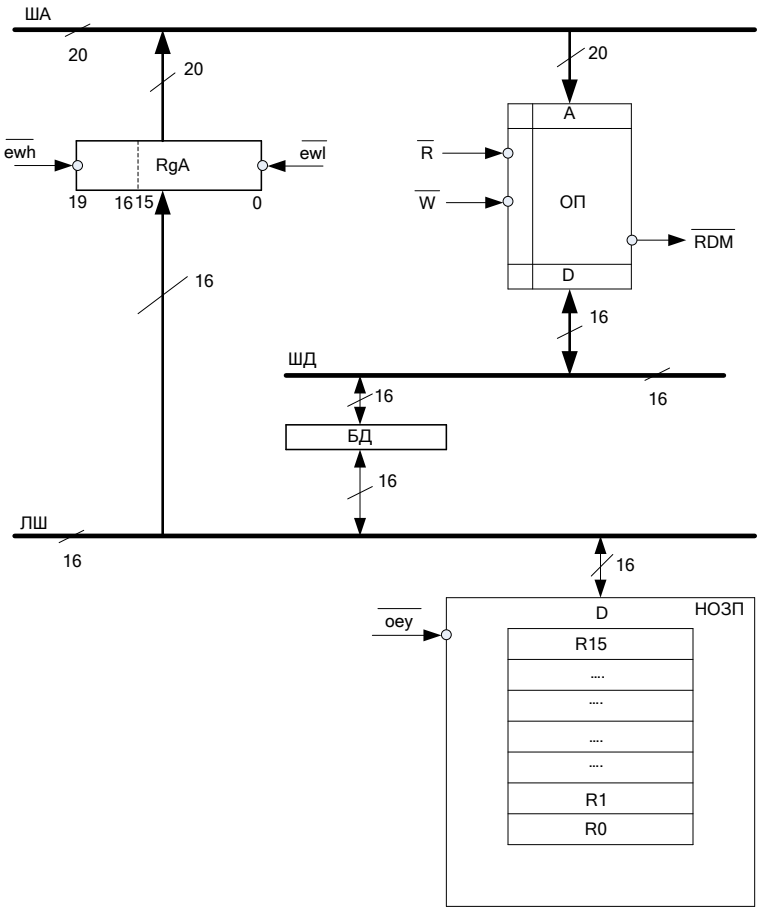


Рис. 1.1. Структурна організація обчислювальної системи

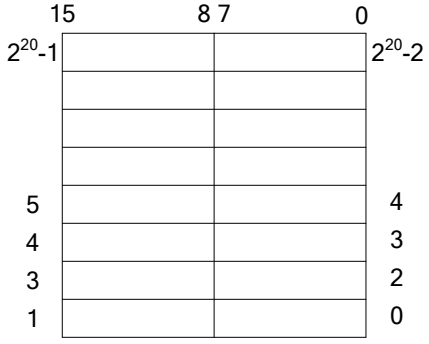


Рис. 1.2. Структурна основної пам’яті

Регистр адреси

Адреса ОП формується з використанням ЛШ. Оскільки ЛШ 16-розрядна, а ША 20-розрядна, то для формування адреси ОП необхідно два такти.

Інформація з ЛШ в регістр адреси (*RgA*) записується за два прийоми. Для цього застосовуються відповідні сигнали дозволу запису: $\overline{EWH}$ (*Enable Write High*) – дозвіл запису в старшу частину *RgA*, та $\overline{EWL}$ (*Enable Write Low*) – дозвіл запису в молодшу частину.

Мнемонічно ці сигнали позначаються наступним чином: `{ewh;}` та `{ewl;}`.

Поділ *RgA* на дві частини здійснюється за допомогою директиви `LINK EWH`.

Приклад:

```
link ewh : 16      \ молодша частина RgA включає розряди 15…0,
                   \ старша частина RgA – розряди 19…16
```

Спочатку на ЛШ формують, наприклад, молодші розряди адреси пам‘яті, та за допомогою сигналу `{ewl;}` записують в молодшу частину *RgA*, а потім на ЛШ формують старші розряди адреси й за допомогою сигналу `{ewh;}` записують в старшу частину *RgA*.

Як окремий випадок можна передавати 16-розрядну адресу, яка формується в одному з регістрів НОЗП. В цьому випадку в регістр *RgA* записується тільки молодша частина адреси за допомогою сигналу `{ewl;}`, при цьому за замовчуванням старша частина адреси приймається такою, що дорівнює нулю. Але в цьому випадку неможливо адресувати частину адресного простору, що адресується більш ніж шістнадцятьма розрядами (рис. 1.3).

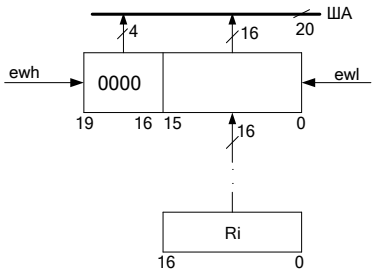


Рис. 1.3. Поділ регістру адреси на старшу і молодшу частини.

Буфер данных застосовується для передавання слів даних між НОЗП і ОП.

16-разрядное НОЗП входить у склад П і містить шістнадцять 16-розрядних регістрів.

Директиви мікроасемблера для роботи з пам'яттю

Крім директиви `LINK EWH` для роботи з ОП передбачено ще дві директиви мікроасемблера.

`DW` – директива завдання значень в комірках ОП. Ця директива (*Define Word*) дозволяє задати значення в одній або кількох суміжних комірках ОП одночасно.

Загальний вигляд директиви:

```
DW <адреса>:<значення>
```

Приклад:

```
dw 0A000h:36      \ за адресою A000 задати слово 36
dw 20h :2,4,8      \ за адресами 20h, 22h, 24h записати
                   \дані 2, 4, 8
```

`ACCEPT RDM_DELAY` – директива завдання швидкодії ОП.

Загальний вигляд директиви:

```
ACCEPT RDM_DELAY:<кількість тактів>
```

Приклад:

```
accept          \ швидкодія ОП дорівнює 3 тактам
rdm_delay:3     \ роботи БОД
```

Після завершення циклу роботи (запис або читання слова) ОП формує сигнал «Готовність пам'яті» $\overline{RDM}$ (*ReaDy Memory*), нульовий рівень якого свідчить про готовність ОП до наступного циклу. Цей сигнал в умовних мікрокомандах ФАМ мнемонічно позначають rdm.

Приклад: Встановити в нуль 4 старші розряди *RgA*, а в молодші записати адресу з *R*14. За даною адресою зчитати слово з ОП в регістр НОЗП *R*10. Виконати його декримент вмісту регістру *R*10 і результат записати у наступну комірку ОП. Швидкодія ОП вдвічі нижча за швидкодію процесора.

Виконання завдання:

```
\ область налагодження зв'язків
    link l2:rdm              \ сигнал RDM подається на
                             \ другий вхід МУ
    link ewh:16              \ поділ RgA
    accept rdm_delay:2       \ сигнал RDM формується
                             \ через 2 такти (затримка
                             \ пам'яті)
\ область завантаження даних
    dw 0024h:0aaaah          \ в ОП за адресою 24h
                             \ завантажується слово AAAA
\ обрасть завантаження регістрів
    accept r14:24h           \ вихідна адреса записується в
                             \ R14
\ область програми
    {xor nil,r3,r3;oey;ewh;} \ RgA[19…16] := 0000,
                             \ завантаження нулів в старшу
                             \ частину регістру адреси
```

```
    {or nil,r14,z;oey;ewl;}    \ RgA[15…0]:=R14,
                               \ завантаження адреси із
                               \ регістру R14 в молодшу
                               \ частину регістру адреси
ss1 {r;cjp rdm,ss1;or r10,bus_d,z;}
                               \ R10:=ОП <24h>, зчитування
                               \ даних із ОП за адресою 24h
    {sub r10,r10,z,z;}         \ R10:=R10 – 1
    {add r14,r14,nz,nz;}       \ R14:=R14 + 2, формування
                               \ адреси наступної комірки
                               \ пам’яті
    {or nil,r14,z;oey;ewl;}    \ RgA[15…0]:=R14

ss2 {w;cjp rdm,ss2;or nil,r10,z;oey;}
                               \ ОП <26h> :=R10, запис слова
                               \ із регістру R10 НОЗП в ОП за
                               \ адресою 26h
    {}                         \ кінець
```

Способи адресації операндов без використання регістрів загального призначення

Виконавчою (фізичною) адресою називають послідовність нулів й одиниць, що видається на адресні шини. Такі послідовності можуть мати більшу довжину (32 і більше розрядів), тому не завжди вдається розмістити фізична адреса в слові команди, або одному з регістрів процесора. У таких випадках адресне поле в команді містить необхідну інформацію для обчислення виконавчої адреси.

Існує три способи адресації операндів без використання регістрів загального призначення (РЗП):

– Пряма (абсолютна) адресація операндів;
– Непряма адресація;
– Безпосередня адресація.

В багатоадресних командах спосіб адресації кожного операнда може бути окремим.

Розглянемо всі перераховані типи адресації на прикладі одноадресної команди.

Пряма адресація операндів

Узагальнена структура команди із прямою адресацією має такий вигляд:

КОП	E

де КОП – код операції, E – виконавча адреса (E – адреса операнда x – Ax).

За такого способу адресації команда містить адресу операнда в ОП. Схема вибірки операнда представлена на рис. 1.4, де РК – регістр команд; *AC* – акумулятор; *SM* – суматор. Для завантаження даних у процесор відбувається одне звертання до ОП.

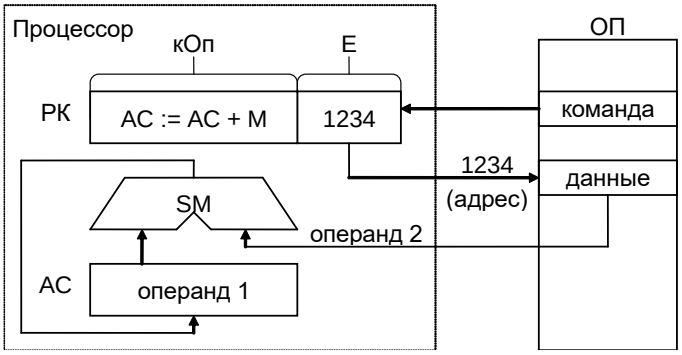


Рис. 1.4. Схема вибірки операнда за прямої адресації

Непряма адресація операндів

Узагальнена структура команди з непрямою адресацією має такий вигляд:

КОП	@A

де КОП – код операції, *@A* – адреса області ОП (покажчик адреси операнда, або адреса комірки пам'яті, де зберігається адреса операнда, тобто *A*(*Ax*)).

За такого способу адресації в команді зберігатися адреса області ОП, з якої можна одержати виконавчу адресу операнда, або інформацію, необхідну для його обчислення. Схема вибірки операнда представлена на рис. 1.5. Для вибірки даних відбувається два звертання до ОП.

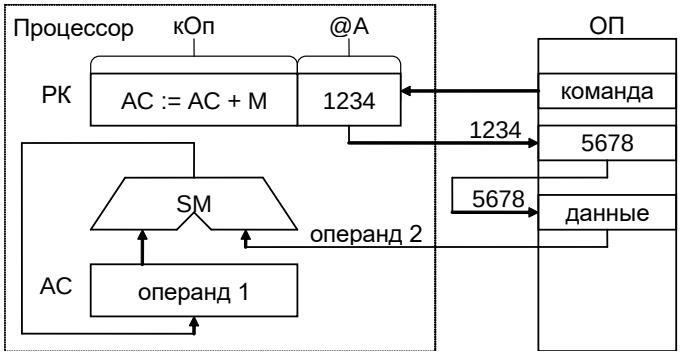


Рис 1.5. Схема вибірки операнда за непрямої адресації

Непряма адресація застосовується під час роботи з підпрограмами, записаними в ПЗУ.

Безпосередня адресація операндів

Узагальнена структура команди з безпосередньою адресацією має такий вигляд:

КОП	I

де *I* – операнд, використовуваний у команді.

За такого способу адресації необхідні дані (операнди) вказуються безпосередньо в самій команді. Дані вибираються з командою, тому додаткових звертань до ОП не потрібно, що показано на схемі на рис. 1.6. Однак цей спосіб адресації застосовується тільки для роботи з константами.

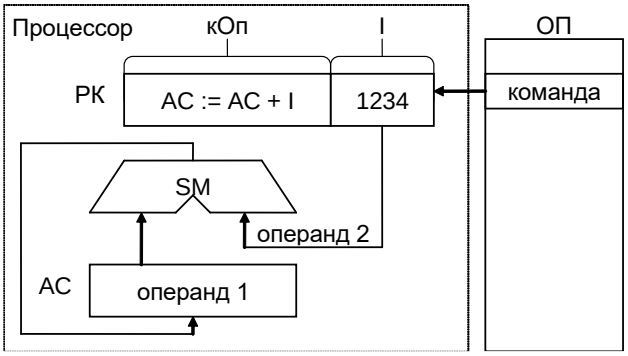


Рис 1.6. Схема вибірки операнда за безпосередньої адресації

Основні цикли звернення процесора до основної пам'яті

До основних циклів звернення до ОП належать цикли читання, запису, читання-модифікація запис.

Мікроалгоритми читання даних из ОП в регістри НОЗП

Формат команди читання:

```
MOV Ri,<A> / ОП<Ax> → Ri пряма адресація
           / ОП<A(Ax)> → Ri непряма адресація,
```

де *Ri* – номер регістру НОЗП, *A* – адреса комірки пам'яті, A_x – адреса комірки пам'яті де зберігається значення *x*.

Основні етапи циклу читання для прямої адресації даних зображені за допомогою мікроалгоритму на рис. 1.7.

1. A_x – адреса операнда *x*, що у вихідному стані знаходиться в одному з регістрів НОЗП, записується в регістр адреси, при цьому виконуються наступні дії:

$$R_i \rightarrow \text{ЛШ} \rightarrow RgA \rightarrow \text{ША} \rightarrow \text{А}_{\text{ОП}},$$

де $\text{А}_{\text{ОП}}$ – вхід адреси пам'яті.

```
{or nil,ri,z;oey;ewl;}
```

2. Процесор генерує сигнал *r*: R → ОП (читання даних за адресою, що виставлена на ША) і переходить у режим очікування.

3. Система знаходиться в режимі очікування готовності поки немає сигналу «Готовність». Через визначений час очікування, що дорівнює часу читання даних ($t_{чит.}$) дані видаються на ШД ($D_{ОП}$ → ШД → БД → ЛШ) а пам'ять видає сигнал *RDM* – «готовність».

4. Якщо є сигнал «Готовність», дані записуються у заданий регістр НОЗП (БД → *Ri*).

```
ss1  {r;cjp rdm,ss1;or r10,bus_d,z;}
```

Основні етапи циклу читання для непрямої адресації даних зображені за допомогою мікроалгоритму на рис. 1.8.

Під час застосування непрямої адресації, на першому етапі в регістр адреси *RgA* записується адреса адреси операнда *A*(*Ax*) (рис. 1.8). На третьому етапі в БД зчитується адреса операнда, яка має бути знов записана у регістр адреси *RgA,* що відбувається на етапі 5 відповідного мікроалгоритму.

```
ss1  {r;cjp rdm,ss1;or nil,bus_d,z;oey;ewl;}
```

Далі повторюються етапи 1 – 4, аналогічно застосуванню прямої адресації.

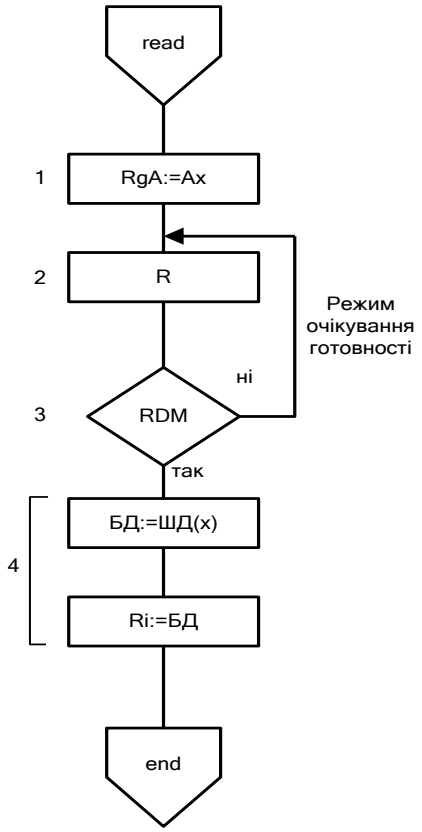


Рис. 1.7. Мікроалгоритм читання даних за прямої регістрової адресації

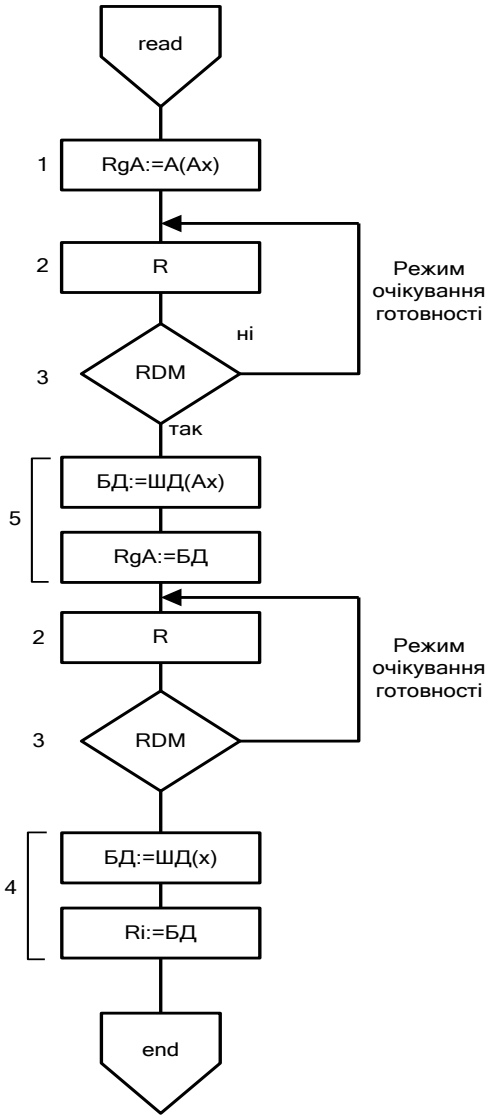


Рис. 1.8. Мікроалгоритм читання даних за непрямої регістрової адресації

Мікроалгоритми запису даних в ОП з регістрів НОЗП

Формат команди запису:

MOV <A>,Ri / *Ri* → *ОП<Ax>* пряма адресація

/ *Ri* → *ОП<A(Ax)>* непряма адресація,

де *Ri* - номер регістру НОЗП, *A* – адреса комірки пам'яті, A_x – адреса комірки пам'яті де зберігається значення *x*.

Основні етапи циклу запису для прямої адресації даних зображені за допомогою мікроалгоритму на рис. 1.9.

1. A_x – адреса операнда *x*, що у вихідному стані знаходиться в одному з регістрів НОЗП, записується в регістр адреси, виконуються наступні дії:

$$R_i \rightarrow \text{ЛШ} \rightarrow RgA \rightarrow \text{ША} \rightarrow \text{А}_{\text{ОП}},$$

де $\text{А}_{\text{ОП}}$ – вхід адреси пам'яті.

```
{or nil,ri,z;oey;ewl;}.
```

2. Процесор виставляє дані, що мають бути записані на ШД:

$$R_i \rightarrow \text{БД} \rightarrow \text{ ШД} \rightarrow D_{\text{ОП}},$$

де *D* – вхід даних ОП.

```
ss2  {w;cjp rdm,ss2;or nil,ri,z;oey;}.
```

3. Процесор генерує сигнал запису *w*: $W \rightarrow$ ОП (запис даних за адресою, що виставлена на ША) і переходить у режим очікування.

Система знаходиться в режимі очікування готовності поки немає сигналу «Готовність».

Через визначений час очікування, що дорівнює часу запису даних ($t_{\text{зап.}}$) пам'ять видає сигнал *RDM* – «готовність».

4. Якщо є сигнал «Готовність» цикл запису закінчений.

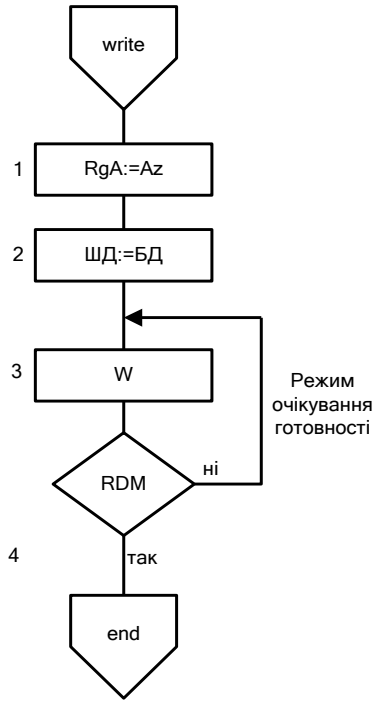
Основні етапи циклу запису для прямої адресації даних зображені за допомогою мікроалгоритму на рис. 1.10.

У випадку застосування непрямої адресації цикл запису даних в ОП можна поділити на два кроки. На першому кроці необхідно вибрати адресу операнда із ОП, для чого застосувати етапи 1 – 3 та 5 аналогічно до мікроалгоритму, зображеного на рис. 1.4. Далі, на другому кроці, виставляються

дані на ШД та відбувається звичайний цикл запису, аналогічно етапам 2 – 4 мікроалгоритму на рис. 1.5.

Наступний фрагмент мікропрограми виконує запис вмісту регістру *Rj* в ОП за адресою, розташованою в регістрі *Ri*, за застосування непрямої регістрової адресації.

```
                {and nil, ri,z;oey; ewl;}
          ss1 {r;cjp rdm ss1;and nil,bus_d,z;oey;ewl;
ss2 {w; cjp rdm ss2;and nil,rj,z,oey;}
```



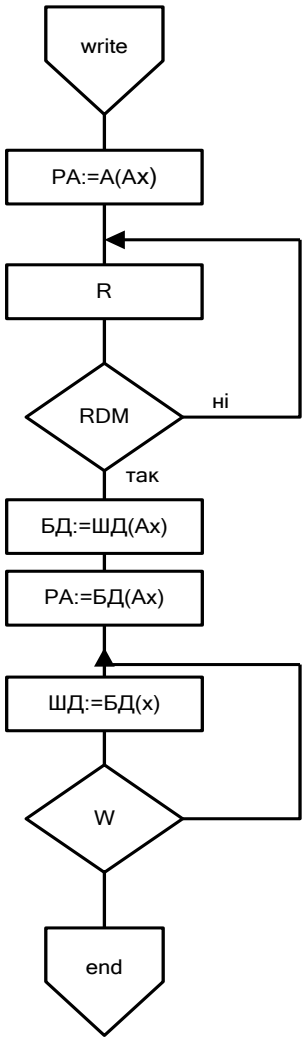


Рис. 1.9. Мікроалгоритм циклу запису за прямої регістрової адресації

Рис. 1.10. Мікроалгоритм циклу запису за непрямої регістрової адресації

Приклад: Обчислити значення функції $z = x + y$. Дані розміщуються в ОП, для формування адреси наступної комірки пам'яті застосувати автоінкрементний спосіб. Адресу першого операнда задати у регістрі R8. Для операнда *x* застосувати пряму адресацію даних, для операнда *y* та результату – непряму.

Виконання завдання:

```
\ область налагодження зв'язків
    link l2:rdm                 \ сигнал RDM подається на
                                \ другий вхід МУ
    link ewh:16                 \ поділ RgA
    accept rdm_delay:2          \ сигнал RDM формується
                                \ через 2 такти (затримка
                                \ пам'яті)
\ область завантаження даних
    dw 00e0h:0011h              \ x
    dw 00e2h:0a24h              \ A(y)
    dw 00e4h:0c64h              \ A(z)
    dw 0a24h:0008h              \ y
    dw 0c64h:00                 \ z
\ область завантаження регістрів
    accept r8:00e0h             \ адреса x
\ область програми
\ виборка значення x в регістр R10 (пряма регістрова адреса-
ція)
    {add nil, r8, z; oey; ewl;} / R8 → ЛШ → RgA
```

```
/ якщо немає rdm, повернення на мітку, інакше записуємо дані з
/ БД у регістр R10
ss1 {r;cjp rdm,ss1;add r10,bus_d,z;oey;}
/ формування адреси наступної комірки пам'яті
    {add r8,r8,002h;}
/ выборка y в регістр R11 (непряма регістрова адресация)
    {add nil,r8,z;oey;ewl;}     / A(Ay) → RgA
/ якщо встановився rdm, Ay → RgA
ss2 {r; cjp rdm,ss2;add nil,bus_d,z;oey;ewh;}
ss3 {r;cjp rdm,ss3;add r11,bus_d,z;oey;} / y → R11
    {add r10, r10, r11;}        \ підсумовування
/ формування адреси наступної комірки пам'яті
    {add r8,r8,002h;}
/ запис z в ОП (непряма регістрова адресация)
    {add nil,r8,z;oey;ewl;}     / A(Az) → RgA
/ якщо встановився rdm, Az → RgA
ss4 {r; cjp rdm,ss4;add nil,bus_d,z;oey;ewh;}
/ видача у БД вмісту регістру R10, якщо є сигнал rdm запис у ОП
ss5 {w; cjp rdm,ss5;add nil,r10,z;oey;}
    {}                          \ кінець
```

На рис. 1.11 представлена карта розміщення операндів у ОП.

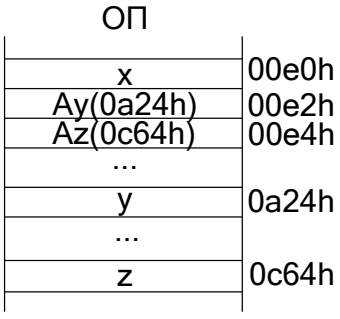


Рис. 1.11. Карта розміщення операндів у ОП

Підготовка до виконання завдання

1. Вивчити способи адресації операндів, в протоколі представити описи із застосування структурних схем способів адресації операндів без застосування РЗП.

2. Вивчити та представити у протоколі основні мікроалгоритми взаємодії П із ОП для різних способів адресації, навести часові діаграми виконання основних циклів взаємодії.

3. Для обчислення функції, заданої в табл. ХХ, розробити мікроалгоритми читання та запису операндів із ОП. Дані розміщуються у сусідніх комірках області даних основної пам'яті. Початкова адреса області даних у пам'яті $A_п$ задана у табл. 1.2. Спосіб формування наступної адреси комірки пам'яті обрати із таблиці 1.1. Способи адресації операндів обрати з табл. 1.2: П – пряма адресація, НП – непряма адресація.

Під час виконання завдання номери робочих регістрів обрати самостійно. Адресу початкової комірки пам'яті $A_п$ розмістити в заданий регістр НОЗП (табл. 1.2). Для формування наступної адреси застосувати формули $A_{i+1} = A_i + 2$ та $A_{i+1} = A_i + 2$ для автоінкрементної і автодекриментної адресації відповідно.

Таблиця 1.1. Варіанти завдання

a_2	a_0	Функція
0	0	x + y = z
0	1	x & y = z
1	0	x V y = z
1	1	x – y = z

Таблиця 1.1. Варіанти завдання

a_0	Спосіб формування наступної адреси
0	автоінкрементний
1	автодекриментний

Таблиця 1.2. Варіанти завдання

a_4	a_2	a_0	X	Y	Z	Початкова адреса	Лічильник адреси
0	0	0	П	П	П	D0h	*R*7
0	0	1	П	П	НП	A0h	*R*10
0	1	0	П	НП	П	28h	*R*11
0	1	1	П	НП	НП	4Ah	*R*8
1	0	0	НП	П	П	20h	*R*9
1	0	1	НП	П	НП	F0h	*R*14
1	1	0	НП	НП	П	B2h	*R*13
1	1	1	НП	НП	НП	1Ah	*R*12

4. В протоколі навести структурну схему обчислювальної системи, карту розміщення даних у ОП, мікроалгоритми виконання операцій читання та запису даних.

Виконання лабораторної роботи

Налагодити розроблену мікропрограму із застосуванням моделюючої програми COMPEX. Під час налагодження виконати приклади із різними значеннями операндів.

ЛАБОРАТОРНА РОБОТА №2
ВИКОНАННЯ КОМАНД В ЕОМ

Мета роботи: Вивчити етапи виконання команд в ЕОМ. Навчитися розробляти мікроалгоритми вибірки, розпакування команд, виконання операцій і формування адреси наступної команди. Одержати навички розробки мікропрограм для ЕОМ з використанням мнемонічного мікроасемблера.

Теоретичні відомості []

Додаткові теоретичні відомості

Системи команд ЕОМ. Різновиди команд

Залежно від реалізованого в обчислювальній системі кількості команд, їх прийнято ділити на два типи:

– RISC - система зі скороченою системою команд;

– CISC - система з комплексною системою команд.

Для RISC процесорів кількість команд визначається десятками, а для CISC - сотнями.

Всі команди по функціональній ознаці діляться на групи:

– основні команди - забезпечують перетворення інформації, тобто змінюють уміст регістрів. До них відносять: команди пересилання, зрушення, арифметичні й логічні команди;

– команди передачі керування - забезпечують безумовні й умовні переходи, виклик і повернення з підпрограм;

– команди вводу-виводу - забезпечують взаємодія процесора із зовнішніми пристроями. Для систем зі сполученим адресним простором ОП і ВУ команди даного типу можуть отсутствовать;

– системні команди - змінюють режими роботи процесора. Наприклад, дозволяють і забороняють переривання, блокую й розблокують операції вводу-виводу, установлюють і скидають режим покрокового виконання й т.п.

Кожна група може ділитися на підгрупи:

безадресна:

одноадресна:

КОП	А1

двухадресная:

КОП	А1	А2

трехадресная:

КОП	А1	А2	А3

де КОП – код операції; А1, А2, А3 – адреса або інформація, використовувана для обчислення адреси операнда.

Залежно від кількості адрес довжина команди може змінюватися. Якщо команда безадресна (без операндів), то це, як правило, системна команда. Одноадресні команди широко використаються як команди передачі керування, команди вводу-виводу. Двухадресні команди - команди основної групи. Трєхадресні команди дозволяють міняти приймач.

Адресація команд

Адреса команд формується на лічильнику команд (СК). Значення, що зберігається в СК, безпосередньо або після об'єднання із умістом іншого регістра СОЗУ (базою) визначає фізична адреса команди в ОП.

Виконання команд основної групи

Для команд основної групи можна виділити чотири основних етапи виконання:

- Вибірка команди з ОП;
- Розпакування (обчислення виконавчих адрес і вибірка операндов);
- Виконання заданої операції;
- Формування адреси наступної команди.

Зазначені етапи можуть виконуватися одночасно з використанням паралельно функціонуючих апаратур, тому такий розподіл на етапи є умовним.

Так, у системах з випереджальна вибірка команд, під час виконання однієї команди відбувається зчитування іншої команди з ОП. Лічена команда записується в чергу команд. Черга являє собою пам'ять із адресацій типу FIFO. Для роботи таких процесорів, крім лічильника команд, необхідно мати додаткові покажчики, за допомогою яких здійснюється доступ до буфера команд.

Розглянемо Ф-микроалгоритмы виконання кожного з етапів виконання команди. Уважаємо, що вибірка команди з пам'яті відбувається за один цикл звертання до системної магістралі, адреса команди зберігається в СК, ША й ШД розділені.

Вибірка команди

У якості СК може використатися будь-який регістр СОЗУ. Для вибірки команди необхідно виконати цикл читання даних з ОП, використовуючи як адреса вміст СК. Якщо в системі використається один з різновидів базової адресації (див. наступну лабораторну роботу), то виконавча адреса повинен бути обчислений на основі СК і вмісту базового регістра. МА виконання цього етапу команди наведений на Рис. 5.1. Результатом завершення даного етапу є команда в регістрі команди процесора.

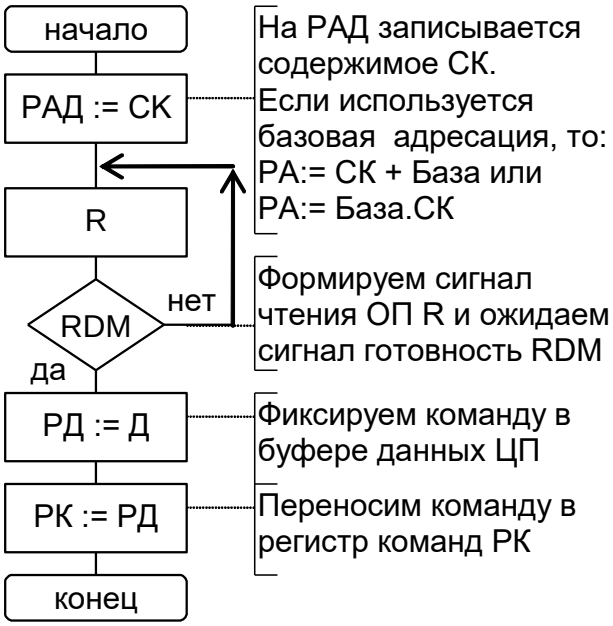


Рис. 5.1. МА вибірки команди з ОП.

Розпакування команд

У коді операції (кіп) можна виділити кілька функціональних полів:

– Формат команди (ФК) - дозволяє визначити довжину команди, кількість полів й їхнє положення в коді команди;

– Тип адресації (ТА) - задає способи адресації для кожного адресного поля А;

– Операція (Оп) - визначає виконувану операцію.

У деяких командах частина із цих полів може отсутствовать.

Мікропрограма розпакування аналізує вміст поля ФК, на основі чого визначає кількість полів типу адресації ТА й адреси А. Після цього здійснює вибірку операндов з ОП.

– Припустимо, що команда, завантажена в РК у попередньому пункті має наступний формат:

кіп (ФК.Оп)	ТА1	А1	ТА2	А2

де ТА1, ТА2 – поля задающие тип адресації відповідно для першого й другого адресного поля; А1, А2 – адресні поля.

Одна з гілочок МА розпакування даної команди наведена на Ошибка! Источник ссылки не найден..

– Передбачається, що для адресації першого операнда використається прямий тип адресації (задається полем ТА1), а для другого - непрямий (поле ТА2).

Як видно з мікроалгоритму, мікропрограми розпакування одне-, двох- і трехадресных команд будуть мати однакові ділянки. Кожна така ділянка здійснює вибірку операндов з ОП відповідно до одним зі способів адресації. Для зменшення обсягу мікропрограми розпакування, її повторювані участи можуть бути реалізовані у вигляді мікропідпрограм. Вони ж можуть виконувати аналіз полів типу адресації ТА.

Результатом завершення цього етапу є операнды, записані в робочі регістри процесора. У нашому прикладі це регістр РР1 і РР2, відповідно для першого й другого операнда.

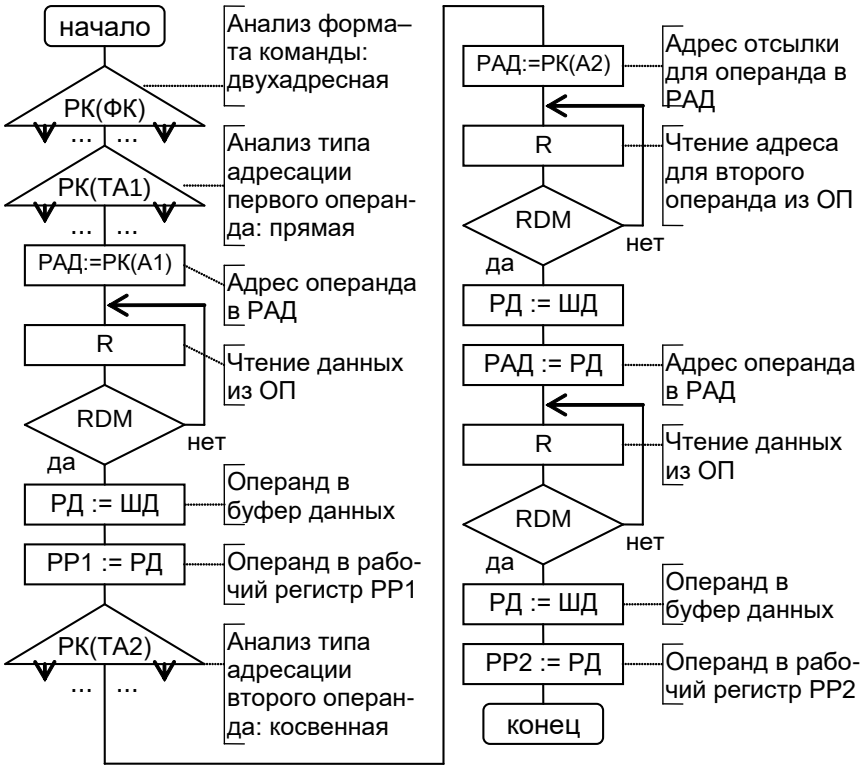


рис. 5.5. МА распаковки двухадресной команды.

Виконання заданої операції

На даному етапі необхідно проаналізувати вміст поля операції (Оп) у коді операції (кіп) і перейти до відповідної мікропідпрограми.

Для аналізу коду операції можуть використатися мікрооперації порівняння поля Оп з константами. У випадку формування в АЛУ результату “дорівнює”, БМУ звертається до відповідної мікропідпрограми. Недоліком такого способу є низька швидкодія, особливо для CISC систем. Іншим способом переходу до мікропідпрограми є обчислення її адреси на основі поля Оп. У такому випадку із РК виділяється поле операції Оп, що обробляється в АЛУ. Результатом цієї обробки є адреса мікропідпрограми операції, що видається на ЛШ процесора. БМУ зчитує ця адреса через буфер М и переходить до необхідної мікропідпрограми. (см. Структуру системи в розділі “опис лабораторного комплексу”). Такий спосіб аналізу поля операції дозволяє скоротити час до декількох тактів синхронізації.

Приклад мікроалгоритму розпакування поля операції наведений на Ошибка! Источник ссылки не найден. а) і б). На Ошибка! Источник ссылки не найден.,а вважається, що аналіз поля операції ведеться його порівнянням з константами,а на

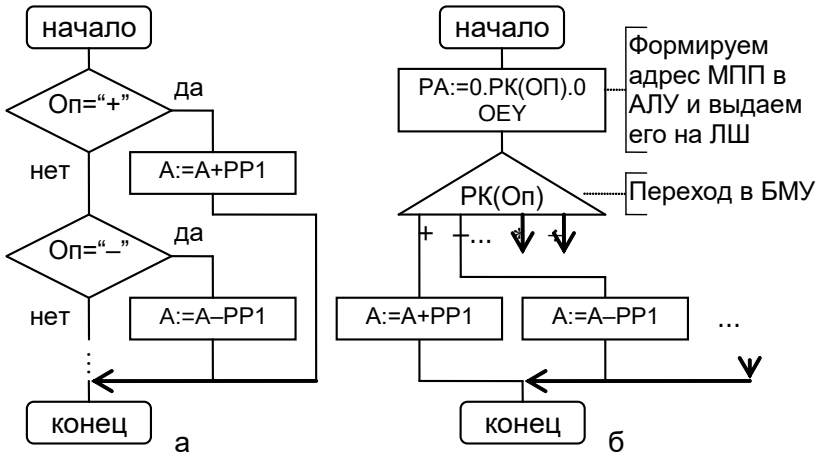


рис.5.6. МА выполнения команды.

Ошибка! Источник ссылки не найден.,б - адреса мікропідпрограми обчислюється в АЛУ.

Формування адреси наступної команди

У лічильнику команд перебуває адреса команди під час її виконання. Для визначення адреси наступної команди необхідно збільшити значення СК. Тому що довжина команд може бути різної (залежно від її формату й адресности), те СК инкрементируется на кількість байт, рівна довжині команди. Для визначення довжини команди може бути проаналізоване поле ФК, а також поля ТА.

МА обчислення адреси наступної команди представлена на Ошибка! Источник ссылки не найден., де l – довжина команди в байтах.

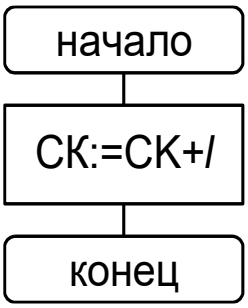


Рис. 5.7. МА вычисления адреса следующей команды.

Підготовка до лабораторної роботи

– Розробити Ф- і Фс-микроалгоритмы виконання всіх етапів одноадресної команди множення (вибірка, розпакування, виконання операції, формування адреси наступної команди). З використанням мікроасемблера написати мікропрограму, що реалізує Фс-микроалгоритм. Операція множення виконується за умовою лабораторної роботи №2.

– Вважати, що система команд містить команди двох форматів (одне- і двухадресные). Формат одноадресних команд показаний на Ошибка! Источник ссылки не найден.. Код операції множення прийняти рівним a_5 a_4 a_2 a_1, де a_i – цифри номера залікової книжки, представленого у двійковій системі числення.

– Команда множення повинна забезпечити виконання наступного перетворення R14,R15 := R15 (M, тобто 32-розрядний результат повинен бути записаний у парі регістрів, в одному йз яких перебуває перший 16-розрядний операнд. Другий 16-розрядний операнд M вибирається з пам'яті.

Для мікроалгоритмів і мікропрограми докладно розробляються тільки ті галузі, які відповідають заданій команді й типу адресації. Тип адресації для розробки визначений у Табл. 4.

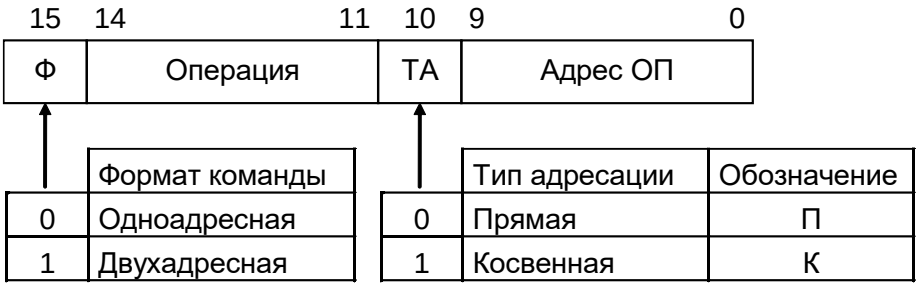


рис. 5.8. Обобщенный формат одноадресной команды

Табл. 5.1

a_3	Тип адресації
0	П (Пряма)
1	ДО (Непрямого)

Регістри R0...R7 є програмно доступними регістрами загального призначення (РОН), причому R7 є також лічильником команд, а регістр R6 зарезервований для подальшого використання. Регістр R15 використається як акумулятор, R8 й R9 як регістр команд. Інші регістри є робочими.

Порядок виконання роботи

– Налагодити розроблену мікропрограму з використанням програмного емулятора.

– Зробити виводи по роботі.

Контрольні питання

– Охарактеризуйте етапи виконання команд, приведіть мікроалгоритми їхньої реалізації.

– Охарактеризуйте основні способи адресації операндов з використанням і без використання Ронов.

– Як забезпечити правильне зчитування й запис даних на згадку?

– Яким образом можна управляти записом інформації в RA й RB, навіщо використаються зазначені регістри?

– Поясните призначення директив мікроасемблера, що визначають роботу з пам'яттю.

ЛАБОРАТОРНА РОБОТА № 3

СПОСОБИ АДРЕСАЦІЇ ОПЕРАНДОВ З ВИКОРИСТАННЯМ РЕГІСТРІВ ЗАГАЛЬНОГО ПРИЗНАЧЕННЯ

Ціль роботи. Вивчити способи адресації операндов, етапи виконання двухадресных команд в ЕОМ. Одержати навички розробки мікропрограм з використанням мнемонічного мікроасемблера.

Теоретичні відомості

Способи адресації операндов з використанням РоновЗагальним недоліком розглянутих у попередній лабораторній роботі способів адресації є більша довжина команди, і як наслідок зниження швидкодії системи. Для зменшення довжини команди застосовується адресації операндов з використанням РОНов.

Існують наступні основні способи адресації операндов з використанням Ронов:

– пряма регістрова адресація;
– непряма регістрова адресація;
– автоинкрементная адресація;
– автодекрементная адресація;
– індексна адресація;
– базова сегментна адресація;
– базова сторінкова адресація.

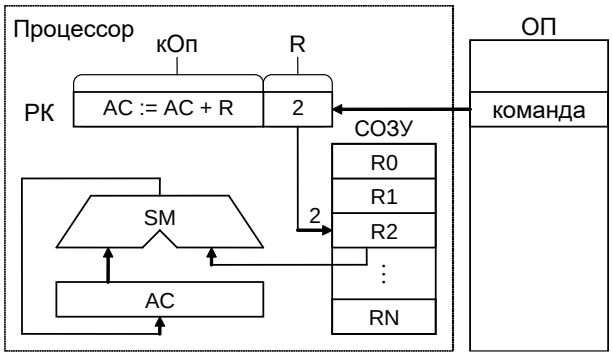
Пряма регістрова адресація

Узагальнена структура команди із прямою регістровою адресацією має такий вигляд:

кіп	R

де кіп – код операції, R – номер регістра (РОНа).

Схема вибірки операнда представлена на **Ошибка! Источник ссылки не найден.**.

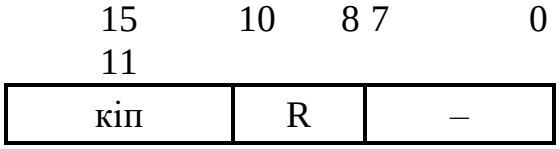


мал. 1. Схема прямої регістрової адресації.

де РК - регістр команд; АС - акумулятор; SM - суматор.

При такому способі адресації різко зменшується довжина команди. Команда може бути однобайтной. Довжина поля R не перевищує п'яти двійкових розрядів, але цього досить для завдання номер будь-якого Рона. Вибірка такої команди може відбуватися за одне звертання до пам'яті. Даний тип адресації придатний тільки для проміжних дій у процесорі, і не дозволяє обмінюватися інформацією між процесором й ОП.

Розглянемо приклад мікропрограми виконання одноадресної операції додавання вмісту акумулятора й РОНа, у випадку використання для адресації останнього прямої регістрової адресації. Як акумулятор використаємо R15, лічильника команд – R9, регістра команд – R10, для зберігання адреси РОНа скористаємося регістром RA (див. опис лабораторного комплексу). Уважаємо що довжина команди дорівнює слову (16 біт), а її формат наведений на мал. 6.2.



мал. 6.2. Формат команди із прямою регістровою адресацією.

де кіп – код операції; R – номер адресуемого регістра.

Нижче наведений приклад мікропрограми вибірки й виконання команд додавання акумулятора з Роном.

Приклад 6.1

```
\ ********************  Настроювання системи
**********************
```

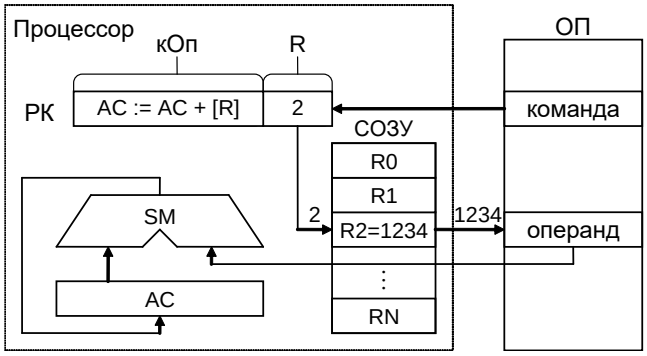
```
Link    L1:RDM              \ Комутируємо RDM на
                            БМУ
Link    RA: z,10,9,8        \ Комутируємо чотири
                            входи
                             \ RA з константою
                            нуль й
                             \ 10, 9 й 8 розрядами
                            ЛШ
ACCE    RDM_DELAY:5         \ Час формування RDM
PT
Accept  R15:10              \ Значення в акумуля-
                            торі
Accept  R9:20h              \ Значення лічильника
                            команд
DW      20h                 \ Заносимо команду в
        :1111101011111111   ОП
        %                    \ кіп = 11111%
                             \ R = 010% – викорис-
                             таємо R2
\ ********************* Мікропрограма
*******************
   { SetAddr R9; }          \ Формуємо адресу на ША зі
                            СК
   { ReadMem R10;           \ Читаємо команду в регістр
   Load ra; }               команд
                             \ і фіксуємо поле R в RA
   { add R15,RA,Z; }        \ Виконуємо додавання
                            AC:=AC+R2
```

Непряма регістрова адресація

При непрямій регістровій адресації узагальнена структура команди має такий вигляд:

кіп	R

При такому способі адресації виконавчим (фізичним) адресою операнда виступає вміст РОНа, зазначеного в команді. Схема вибірки представлена на **Ошибка! Источник ссылки не найден.**.



мал. 6.3. Схема непрямої регістрової адресації.

Як й у попередньому типі адресації, команда має скорочену довжину, але дозволяє забезпечити обмін даними між процесором й ОП.

У відмінності від прямої адресації, у цьому випадку вибірка операндов відбувається швидше. Збільшення швидкодії досягається як за рахунок скорочення довжини команди, так і за рахунок того, що звертання до регістрів СОЗУ відбувається швидше, ніж до комірок пам'яті.

Приклад мікропрограми додавання акумулятора з операндом, обраним з ОП з використанням непрямої регістрової адресації, наведений нижче. Уважаємо що призначення всіх регістрів і настроювання системи аналогічні попередньому прикладу.

Приклад 6.2

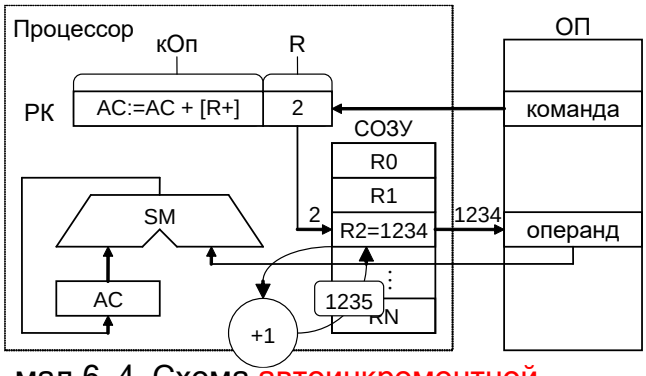
```
 \ ********************* Мікропрограма
 ********************
    { SetAddr R9; }        \ Формуємо адресу на ША зі
                           СК
    { ReadMem R10;         \ Читаємо команду
    Load ra; }
    { or NIL,RA,Z; oey;    \ Установлюємо адресу на
    ewl; }                 ША з R2
    { ReadMem R11;}        \ Зчитуємо операнд із ОП в
                           R11
    { add R15,R11,Z; }     \ Виконуємо додавання
                           AC:=AC+[R2]
```

Різновидами непрямої регістрової адресації є автоинкрементный й автодекрементный способи адресації.

Автоинкрементная адресація

Узагальнена структура команди й спосіб вибірки операнда з ОП відповідають схемі непрямої регістрової адресації. Відмінність полягає в тім, що після зчитування операнда, збільшується (инкрементируется) значення

РОНа, зазначеного в команді. Тип операнда (байт, слово, подвійне слово) визначає на скільки повинне збільшитися вміст РОНа. Даний тип адресації дозволяє ефективно обробляти послідовності (масиви) однотипних даних. Завантаження однобайтового операнда для цього типу адресації представлені на **Ошибка! Источник ссылки не найден.**.



мал.6. 4. Схема автоинкрементной адресації.

Автодекрементная адресація

Відмінністю автодекрементной адресації від автоинкрементной є те, що після вибірки операнда вміст Рона зменшується (декрементируется). Величина зміни вмісту Рона визначається типом оброблюваних даних.

У такий спосіб автодекрементная адресація перебирає слова масиву пам'яті у зворотному порядку.

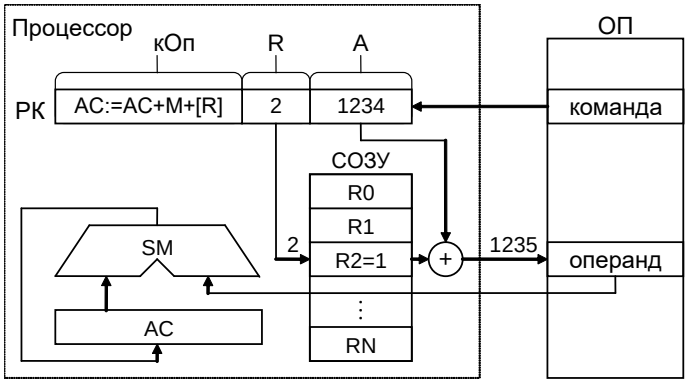
Індексна адресація

Узагальнена структура команди з індексною адресацією має такий вигляд:

кіп	R	A

де кіп – код операції, R – номер регістра, A – адреса.

При такому способі адресації виконавчий (фізичний) адреса операнда формується як сума адреси A і вмісту РОНа з номером R, заданих у команді. Схема вибірки представлена на **Ошибка! Источник ссылки не найден.**.



мал.6. 5. Схема індексної адресації.

Як й у непрямій регістровій адресації, для вибірки операнда потрібне додатковий обіг до пам'яті, але команда більше гнучка.

Цей спосіб адресації зручний при роботі з масивом. У такому випадку поле А інтерпретується як адреса початку масиву, а вміст Рона - як індекс елемента в масиві.

Дана команда використається тільки якщо ширина ШД не менше ширини ША.

Базова сегментна адресація

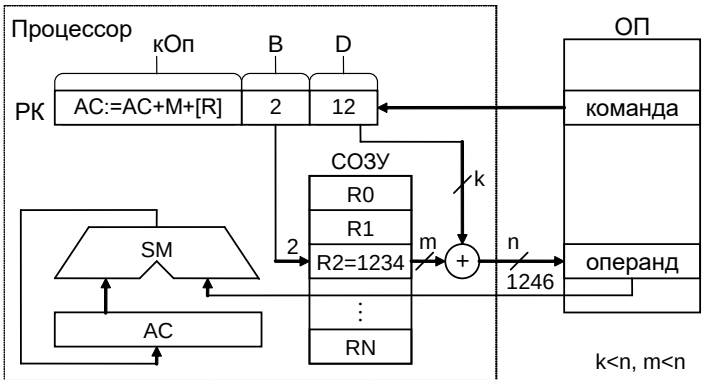
Базова сегментна адресація застосовується коли розрядність ШД або(і) Ронов менше розрядності ША, тобто зсув може бути коротше, ніж повна довжина адреси пам'яті. При базовій адресації узагальнена структура команди має такий вигляд:

кіп	B	D

де кіп – код операції, B – номер регістра бази, D – дані.

Вся ОП ділиться на сегменти. База вказує на адресу початку сегмента. Іноді на величину бази накладаються обмеження по довжині, тоді крок, що вказує сегменти, буде дискретним. Так фізична адреса початку сегмента може виходити зрушенням умісту регістра бази. Наприклад, у процесорах сімейства 86 фірми Intel базові регістри є 16-розрядними, тому для формування 20-розрядної адреси, їхній уміст, перед додаванням зсуву, зрушується на 4 розряди вліво.

На **Ошибка! Источник ссылки не найден.** показано приклад базової сегментної адресації для випадку, коли в поле D утримується байт зсуву, а розрядність Ронов і ША збігається, тобто база не модифікується.



мал. 6.6. Схема базової сегментної

У системі можуть бути стандартні базові регістри, використовувані для роботи з кодом, даними, стеком і т.д. У такому випадку в команді може отсутствовать явне завдання базових регістрів. Використання тих або інших базових регістрів визначаться семантикою виконуваною дією. Так операнды вибираються із сегмента даних, а для завантаження команд залучається базовий регістр сегмента кодів.

Базова сторінкова адресація

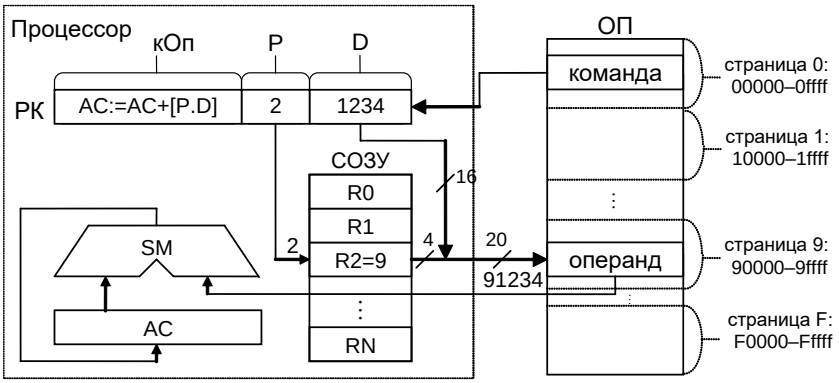
У цьому випадку адреса ОП розділена на дві частини: базу, що визначає номер сторінки, і номер осередку в сторінці. Фізичну адресу знаходимо не підсумовуванням, у відмінності від сегментної адресації, а приєднанням до номера сторінки номера осередку. Отже сторінки, у відмінності від сегментів, не можуть перетинатися. При базовій сторінковій адресації узагаль-

кіп	P	D

нена структура команди має такий вигляд:

де кіп – код операції, P – номер регістра сторінки, D – номер осередку зі сторінці.

На **Ошибка! Источник ссылки не найден.** наведено приклад формування 20-розрядної фізичної адреси для випадку коли номер сторінки є 4-розрядним, а номер осередку, відповідно - 16-розрядним (всі числа наведені в 16-річном коді).



мал. 6.7. Схема базової сторінкової адресації.

На практиці існують комбіновані способи адресації, наприклад, базова індексна, сегментна автоинкрементная й т.д. Кілька різновидів адресації виходить, якщо в якості Рона вказують лічильник команд.

У системах RISC використають спрощені способи адресації, в основному пряма, непряма, спеціальна (через стек), що зменшує час розпакування команди з поля апаратур. У системах (CISC) адреса обчислюється на мікропрограмному рівні.

Підготовка до лабораторної роботи

Розроблені при виконанні лабораторної роботи №3 мікроалгоритми й мікропрограму доповнити фрагментами розпакування й виконання двухадресной команди.

Команда для розробки визначається таблицею варіантів (Табл.6. 2), де a_i ($i = \overline{5...1}$) – молодші двійкові розряди номера залікової книжки. Уважати, що система команд містить команди двох форматів (одне- і двухадресные), як показано на **Ошибка! Источник ссылки не найден.**.

Команди, задані Табл.6. 2 є двухадресными. Вони повинні здійснювати дію DD := DD операція SS, що інтерпретується в такий спосіб. Результат операції повинен бути записаний на місце першого операнда. Перший і другий операнды (відповідно DD й SS) задаються полями типу адресації ТА й номером Рона в коді команди. У командах можуть адресуватися вісім Ронов. Операнды двухадресных арифметичних команд є цілими 16-розрядними числами (з урахуванням знакового розряду), представленими в додатковому коді. Логічні операції виконуються над 16-розрядними словами.

Для мікроалгоритмів і мікропрограм докладно розробляються тільки ті галузі, які відповідають заданим командам і типам адресації. Тип адресації операндов визначений у Табл. 6.3.

мал. 6.8. Узагальнений формат команд

У якості Ронов використати регістри R0...R7. Регістр R15 використати як акумулятор, R6 - як покажчик стека, R7 - як лічильник команд, а R8, R9 - як регістр команд. Інші регістри є робітниками.

Табл.6. 2

a_5	**a_4**	**Операція**
0	0	+ (Додавання)
0	1	– (Вирахування)
1	0	V (Диз'юнкція)
1	1	& (Конъюнкция)

Табл. 6.3

a_2	**a_1**	**Типи адресації**
0	0	ПР
0	1	КР
1	0	АИ
1	1	ПЕКЛО

Порядок виконання роботи

- Налагодити розроблену мікропрограму з використанням програмного емулятора.
- Зробити виводи по роботі

Контрольні питання

- Охарактеризуйте етапи виконання команд, приведіть мікроалгоритми їхньої реалізації.
- Охарактеризуйте основні способи адресації операндов з використанням і без використання Ронов.
- Як забезпечити правильне зчитування й запис даних на згадку?
- Яким образом можна управляти записом інформації в RA й RB, навіщо використаються зазначені регістри?
- Поясните призначення директив мікроасемблера, що визначають роботу з пам'яттю.
- При одночасному виконанні яких мікрооперацій можуть виникати конфлікти на локальній шині ЕОМ?
- Як управляти регістром стану системи?

Додаткова література /5,6/.

ЛАБОРАТОРНА РОБОТА № 4

ВИКОНАННЯ КОМАНД ПЕРЕДАЧІ КЕРУВАННЯ Й КОМАНД РОБОТИ З ПІДПРОГРАМАМИ

Ціль роботи. Вивчити етапи виконання команд передачі керування й команд роботи з підпрограмами. Навчитися розробляти мікроалгоритми й мікропрограми виконання зазначених команд.

Теоретичні відомості

При природному (послідовному) порядку виконання команд, адреса наступної команди може бути обчислений у такий спосіб: СК := СК + *l*, де *l* – довжина команди.

Команди передачі керування дозволяють змінити природний порядок виконання програми. До цієї групи відносять наступні типи команд:

- безумовний перехід;
- умовний перехід;
- умовний і безумовний переходи до підпрограм;
- умовне й безумовне повернення з підпрограми.

Незалежно від того, до якому з перерахованих типів ставиться команда, можна виділити кілька загальних етапів її виконання, а саме:

- вибірка команди;

– розпакування команди;

– виконання команди.

Етапи вибірки й розпакування для команд передачі керування нічим не відрізняється від відповідних етапів для команд основної групи. Вибірка команди й завантаження операндів може здійснюватися кожним з розглянутих раніше способів.

На етапі виконання команди цієї групи модифікують або можуть модифікувати (команди умовного переходу) значення лічильника команд СК.

Командибезумовного **переходу**

мал. 7.1. МА виконання команди безумовного переходу

Для команд даного типу, отриманий у результаті вибірки й розпакування операнд, використається як адреса переходу. Мікроалгоритм виконання даної команди наведений на **Ошибка! Источник ссылки не найден.**, де БВК - блок вибірки команди - відповідає етапу вибірки, БРК - блок розпакування команди - етап розпакування, Е - ефективний (фізичний) адреса наступної команди.

Тому що БВК і БРК є однаковими для команд основної групи й передачі керування, то для формування ефективної адреси Е можуть використатися будь-які способи адресації, як з використанням Ронов, так і без їхнього використання.

Адреса наступної команди записується в СК. Це порозуміється тим, що стандартна мікропідпрограма вибірки команд із ОП використає вміст СК для обчислення адреси команди.

Команда умовного переходу

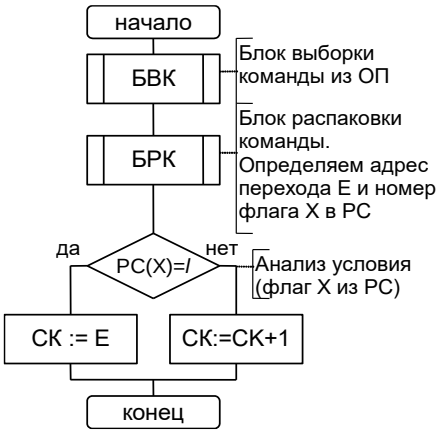
У відмінності від команди безумовного переходу, де в кожному разі СК модифікується обчисленою адресою Е, у командах умовного переходу, завантаження СК відбувається тільки при виконанні аналізованої умови. Як умова може виступати вміст одного або декількох битов у регістрі стану процесора. В останньому випадку говорять що перехід відбувається по масці. МА виконання команди умовного переходу наведений на **Ошибка! Источник ссылки не найден.**.

Якщо умова виконується, то в СК завантажується ефективна адреса Е. У противному випадку адреса наступної команди визначається також, як і для команд основної групи, тобто СК:=СК + *l*, де *l* – довжина команди.

Залежно від способу формування нового значення в СК, виділяють два типи команд переходу:

– абсолютні;

– відносні.



мал. 7.2. МА виконання команди умовного переходу

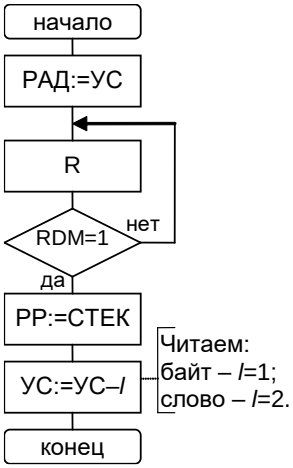
У першому випадку, ефективна адреса обчислюється на основі операндів, завантажених при розпакуванні команди. У другому - отриману адресу Е, може підсумуватися із умістом СК, тобто перехід виконується щодо поточної команди.

Команди роботи з підпрограмами

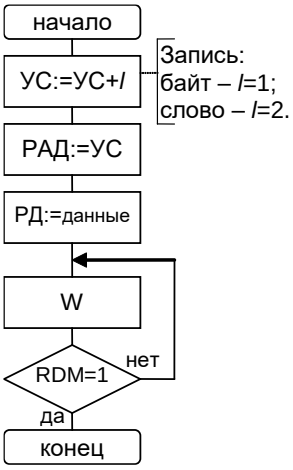
Для реалізації команд даної групи може використатися механізм стека. Як правило, механізм стека застосуються в універсальних ЕОМ, а в спеціалізованих - може отсутствовать.

Стік – частина ОП в адресному просторі процесора, адресація в якій здійснюється за допомогою покажчика стека. Покажчик стека ВУС адресує заповнену вершину стека, тобто вказує на останнє записане в стек значення. У якості ВУС може використатися будь-який регістр СОЗУ. На **Ошибка! Источник ссылки не найден.** і **Ошибка! Источник ссылки не найден.** показані відповідно мікроалгоритми читання й запису в стек.

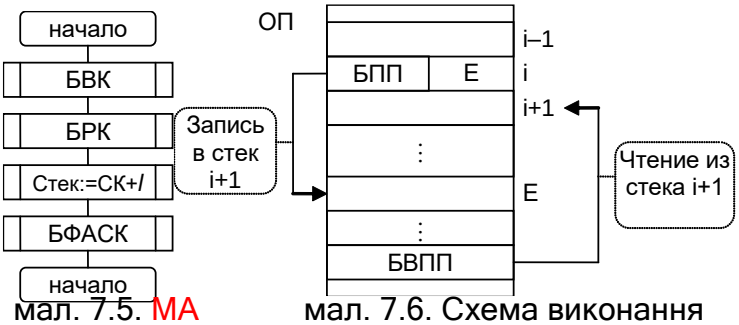
При переході до підпрограм необхідно запам'ятовувати адресу повернення, що відрізняє команди даного типу від команд передачі керування. Мікроалгоритм безумовного переходу до підпрограми БПП представлений на **Ошибка! Источник ссылки не найден.**, а відповідна йому схема на **Ошибка! Источник ссылки не найден.**.



мал. 7.3. МА читання зі стека



мал. 7.4. МА запису в стек

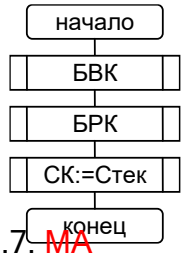


мал. 7.5. МА БПП

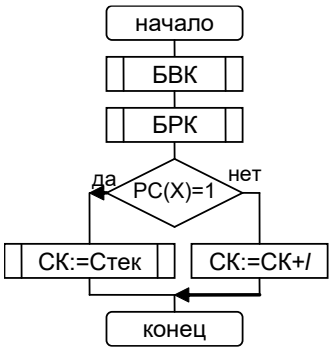
мал. 7.6. Схема виконання БПП

У блоці розпакування команди БРК (**Ошибка! Источник ссылки не найден.**) відбувається обчислення адреси Е першої команди підпрограми, що на етапі формування адреси наступної команди (БФАСК) заноситься в СК, тобто СК := Е. При виконанні команди БПП у стеці зберігається адреса команди, що випливає за командою БПП. На схемі виконання БПП (**Ошибка! Источник ссылки не найден.**) команда безумовного повернення з підпрограми позначена як БВПП. Відмінністю команди умовного переходу до підпрограми є те, що збереження в стеці адреси повернення й перехід до підпрограми здійснюється тільки при виконання умови або їхньої групи.

Для виконання команд безумовного повернення з підпрограм БВПП адреса повернення в команді не потрібний, тому що він автоматично зберігається в стеці, при звертанні до підпрограми. Відмінністю команди умовного повернення УВПП є необхідність аналізу умови (або їхньої групи). МА виконання команд БВПП й УВПП представлені відповідно на **Ошибка! Источник ссылки не найден.** і **Ошибка! Источник ссылки не найден.**.



мал. 7.7. МА безумовного повернення з підпрограми



мал. 7.8. МА умовного повернення з підпрограми

Достоїнством команд роботи з підпрограмами, що використають механізм стека, є практично необмежена вкладеність підпрограм, тому що обсяг стека може бути розширений на всю ОП. Але робота зі стеком вимагає додаткових витрат по звертанню до пам'яті, що знижує швидкодію системи. Тому в спеціалізованих системах застосовуються способи роботи з підпрограмами без використання стека, наприклад спосіб з непрямої адресації або спосіб з індексної адресації.

Підготовка до лабораторної роботи

– Доробити мікропрограму, отриману при виконанні лабораторних робіт №3 й №4, включивши до складу команд одноадресні команди безумовного й умовного переходу. Формат одноадресних команд відповідають заданому в лабораторній роботі №3. Тип адресації визначається Табл. 4. Код

Табл. 4

a_3	**Тип адресації**
0	ДО (Непрямого)
1	П (Пряма)

операції умовного переходу дорівнює $a_5\ a_4\ a_2\ a_1+1$, а безумовного – $a_5\ a_4\ a_2\ a_1+2$ (перенос при підсумовуванні відкидається). Команда умовного переходу здійснює розгалуження програми, якщо встановлено в нуль 7-й розряд акумулятора (7-й розряд R15).

– Включити до складу команд одноадресну команду безумовного виклику й безадресну команду повернення з підпрограми. Формат ко-

манди безумовного виклику й тип адресації відповідають команді безумовного переходу, код операції – a_5 a_4 a_2 a_1+3. Формат операції безумовного повернення відповідає одноадресній команді з кодом операції a_5 a_4 a_2 a_1+4, у якій поля ТА й АдресОП не використаються. Як покажчик стека використати регістр R6.

– Розробити програму в кодах команд (командний рівень), що містять шість команд:

1. умовний перехід;
2. множення;
3. звертання до підпрограми;
4. двухадресная команда, зазначена в роботі №4 (у вигляді підпрограми);
5. повернення з підпрограми;
6. безумовний перехід (на першу команду).

Мнемоніку команд визначити самостійно.

Установка початкових значень у регістрах здійснюється директивою ACCEPT.

Приклад 7.*1* accept r15:0080 \початкове значення R15 дорівнює 80

Для занесення розробленої програми й даних в основну пам'ять використається директива DW.

Приклад7.2

```
d  08h:151  \ за адресою 08h дані MEM=1515h
w  5h
d  10h:723  \ BEGIN: JNZ  LAB1   ; (команда
w  2h       усл. переходу)
d  12h:723  \        CALL SUBR ; (виклик під-
w  4h       програми)
d  14h:6dc  \ LAB1:   MUL  A, MEM
w  2h
d  16h:447  \        JMP   BEGIN
w  eh
            \ ; Підпрограма
d  20h:512  \ SUBR:  ADD   @R1, R4
w  4h
d  22h:723  \        RET    ; (повернення з підп-
w  5h       рограми)
```

Порядок виконання роботи

– Налагодити розроблену мікропрограму з використанням програмного емулятора в режимі трасування.

– Поставити крапку останова на останній мікрокоманді мікропрограми. Виконати в автоматичному режимі записану в пам'яті програму.

– Зробити виводи по роботі.

Контрольні питання

– Охарактеризуйте етапи виконання команд передачі керування, приведіть мікроалгоритми їхньої реалізації.

– Назвіть основні відмінності команд роботи з підпрограмами від команд передачі керування?

– Охарактеризуйте основні способи адресації операндов з використанням і без використання Ронов.

– Як забезпечити правильне зчитування й запис даних на згадку з урахуванням швидкодії пам'яті?

– Яким образом можна управляти записом інформації в RA й RB, навіщо використаються зазначені регістри?

– Поясните призначення директив мікроасемблера, що визначають роботу з пам'яттю.

– Як управляти записом ознак у регістр стану процесора?

Додаткова література /5,6/.

ЛАБОРАТОРНА РОБОТА № 5

ВИКОНАННЯ КОМАНД ВВОДУ-ВИВОДУ

Ціль роботи. Вивчити етапи виконання команд вводу-виводу. Навчитися розробляти мікроалгоритми й мікропрограми реалізації кожного етапу зазначених команд. Вивчити способи взаємодії процесора із зовнішніми пристроями в програмному режимі опитування готовності пристроїв. Одержати навички розробки мікропрограм з використанням мнемонічного мікроасемблера

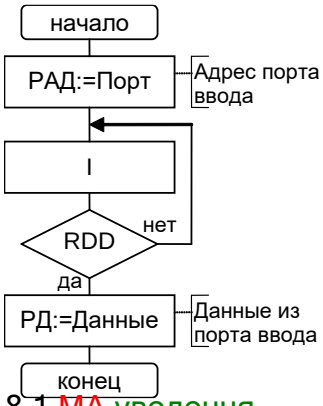
Теоретичні відомості

Команди вводу-виводу

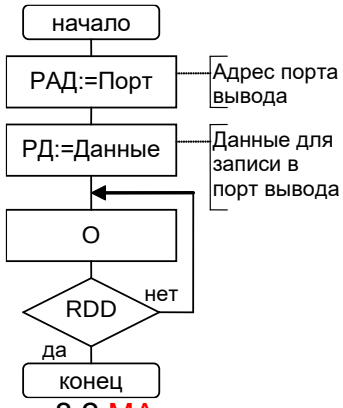
Якщо процесор використає роздільний адресний простір ОП і ВУ, то в його систему команд повинні входити команди роботи із зовнішніми пристроями - команди вводу-виводу. Найпростішими із зазначених команд є:

– команда уведення (IN), що дозволяє вважати дані з порту ВУ в процесор;
– команди виводу (OUT), призначена для запису даних із процесора в порт ВУ.

При обміні даними з портом вводу-виводу процесор формує сигнал уведення I (читання даних з порту) або виводу O (запис даних у порт). За змістом ці сигнали аналогічні сигналам читання R і запису W ОП. Типові МА уведення й виводу в порт ВУ наведені на **Ошибка! Источник ссылки не найден.**8.1 і **Ошибка! Источник ссылки не найден.** відповідно. Для визначення вірогідності даних, що вводять, і моменту фіксації вихідних даних у ВУ служить сигнал готовності RDD



мал. 8.1 МА уведення даних з порту уведення

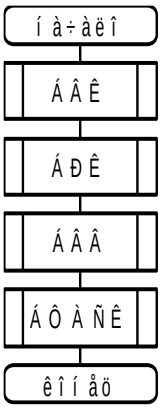


мал. 8.2 МА виводу даних у порт виводу

МА команди вводу-виводу містить чотири стандартних етапи виконання команди: вибірка, розпакування, виконання й обчислення адреси наступної команди. Узагальнений МА команди вводу-виводу представлений на Мал. , де БВВ – блок вводу-виводу, описуваний МА уведення (**Ошибка! Источник ссылки не найден.**8.1) для команди уведення або МА виводу (**Ошибка! Источник ссылки не найден.**8.2) для команди виводу. У блоці розпакування команди БРК обчислюється фізична адреса порту E, з яким здійснюється обмін. У загальному випадку, для формування адреси можуть використатися будь-які способи адресації операндов, однак на практиці найбільше поширення одержали прямі й непряма регістрова адресації.

У ряді систем команди вводу-виводу ставляться до привілейованих команд, і доступні тільки в деяких режимах роботи процесора. У такому випадку, перед початком обміну даними з портом вводу-виводу, у мікропрограмі необхідно перевірити режим роботи процесора (прапор у регістрі стану процесора).

Взаємодія процесора із зовнішніми пристроями



мал. 8.3. Узагальнений МА виконання команди вводу-виводу

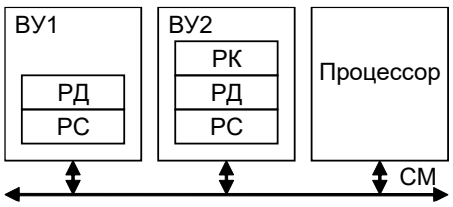
При обміні інформацією з ВУ процесор виступає як активний елемент, тобто управляє процесом обміну. Можна вказати три найбільше що часто використаються режиму обміну інформацією з ВУ:

– Режим програмного опитування готовності ВУ;

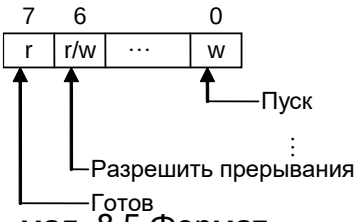
– Режим переривань програм;

– Режим прямого доступу до пам'яті (ПДП).

Будь-який зовнішній пристрій може бути представлене у вигляді сукупності декількох регістрів, відображуваних на порти вводу-виводу або адресний простір ОП (**Ошибка! Источник ссылки не найден.**8.4). Серед цих регістрів можна виділити регістр даних РД, регістр стану РС і регістр команд РК. Для керування простими пристроями досить два регістри: даних



мал. 8.4. Регістри ВУ



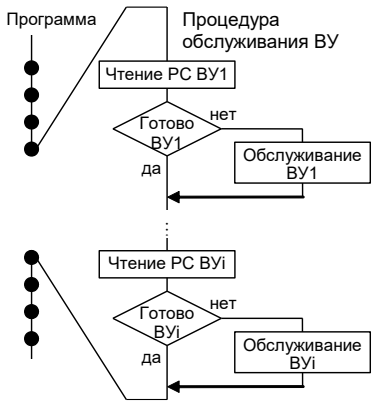
мал. 8.5 Формат регістра стану ВУ

і стану. Для складних пристроїв кількість регістрів може досягати декількох десятків.

РД служить для обміну інформацією між процесором і ВУ. РС дозволяє визначити стан зовнішнього пристрою й задати спосіб його взаємодії із системної (системною магістраллю СМ і процесором). Кожен розряд регістра стану може виконувати певну функцію. Можливий формат регістра стану наведений на **Ошибка! Источник ссылки не найден.**, де розряди позначені буквою r - доступні для читання, w - для запису, а r/w - для читання/запису. РК дозволяє настроїти ВУ на певний режим роботи або виконати задані дії.

Режим програмного опитування готовності ВУ (полинга)

У режимі програмного опитування процесор, що з ініціатором обміну є процесор зчитує вміст РС. певною періодичністю, ВУ, і якщо він У прочитаних даних обслуговування аналізується біт готовності даними установлений, то робить ВУ (обмін даними). Обмін із РС й обміну здійснюється через РД. використаються Для зчитування інформації Схема даними через РД режимі полинга команда вводу-виводу. обслуговування ВУ в наведена на **Ошибка! Источник ссылки не найден.**8.6.



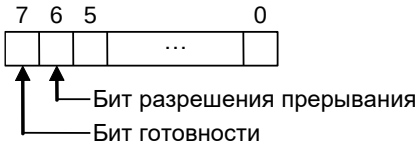
мал. 8.6. Схема обслуговування ВУ в режимі опитування

Головним достоїнством даного режиму обслуговування ВУ є його простота. До недоліків можна віднести: непродуктивні витрати процесорного часу на опитування готовності ВУ, що знижує продуктивність системи, а також підвищену складність обслуговування аварійні ситуації в системі.

Підготовка до лабораторної роботи

– Доробити мікропрограму, отриману при виконанні лабораторних робіт 3, 4 й 5, включивши до складу команд одноадресні команди уведення й виводу. Формат команд зазначений на мал. Х. Код операції команди уведення $a_5\ a_4\ a_2\ a_1+5$ й операції виводу – $a_5\ a_4\ a_2\ a_1+6$.

Зовнішні пристрої містять регістр стану (РС) і регістр даних (РД). Звертання до РС можливо в будь-який момент часу. ДО РД можна звертатися тільки коли в РС установлений біт готовності зовнішнього пристрою. Регістр РС є 8-розрядним, а РД – 16-розрядним. Формат РС показаний на **Ошибка! Источник ссылки не найден.**8.7. Команда уведення здійснює уведення даних в акумулятор, функцію якого в системі виконує регістр R15.



мал. 8.7. Формат регістра стану ВУ

За допомогою команди виводу здійснюється вивід слова з акумулятора в зовнішній пристрій.

Адреси регістрів зовнішніх пристроїв включаються в загальний адресний простір зовнішніх пристроїв. Адреса РД треба за адресою РС. При роботі з регістрами зовнішніх пристроїв використаються сигнали I й O, що формуються в блоці мікропрограмного керування.

– Розробити програму в кодах команд для передачі двох слів із пристрою уведення в пристрій виводу (адреси регістрів зазначені в Табл.

8.8.1). Перед звертанням до РД зовнішнього пристрою варто перевіряти готовність пристрою до обміну. Для цього необхідно прочитати РС пристрою й перевірити біт готовності.

Для занесення розробленої програми в основну пам'ять при налагодженні мікропрограми використається директива DW

Приклад 8.1

```
d   120h:751   \ записати за адресою 120h дані
w   5h         7515h.
```

Пристрою уведення й виводу настроюються директивою ACCEPT.

Приклад 8.2

```
link    l2:rdd          \ підключити до l2 сигнал
                        READY ВУ
link    l1:rdm          \ підключити до l1 сигнал
                        READY пам'яті
accep   dev[2]:o,       \ пристрій виводу
t
        30h,            \ адреса РС
        32h,            \ адреса РД
        3,              \ затримка сигналу RDM у
                        тактах
        14              \ затримка установки біта
                        готовності РС
                         \ після звертання до РД у
                        тактах
accep   dev [1]: i,     20h, 22h, 3, 15
t
accep   dev_buf[1]:12   \ дані, які будуть уводитися
t       34h, 5678h,     в
        89abh, 0eeeeh    \ процесор із РД
```

Порядок виконання роботи

– Налагодити розроблену мікропрограму з використанням програмного емулятора в режимі трасування.

Табл. 8.1

a_3	a_2	a_1	Адреси РС	
			Увв	**УВ ыв**
0	0	0	02H	82H

– Поставити крапку останова наприкінці мікропрограми виконання команд. Виконати в автоматичному режимі записану в пам'яті програму обміну інформацією між зовнішніми пристроями.
– Зробити виводи по роботі.

0	0	1	12H	92H
0	1	0	22H	A2H
0	1	1	32H	B2H
1	0	0	42H	C2H
1	0	1	52H	D2H
1	1	0	62H	E2H
1	1	1	72H	F2H

Контрольні питання

– Охарактеризуйте етапи виконання команд уведення й виводу.
– Як забезпечити правильне зчитування даних із РД зовнішнього пристрою.
– Чи можуть адреси ВУ включатися в адресний простір основної пам'яті?
– Як ураховується при написанні мікропрограм затримка формування сигналу RDD?
– Поясните призначення директив мікроасемблера, що визначають роботу із зовнішніми пристроями.

Додаткова література /5, 6/

ЛАБОРАТОРНА РОБОТА №6

РЕАЛІЗАЦІЯ РЕЖИМУ ПЕРЕРИВАНЬ

Ціль роботи. Вивчити способи обробки векторних і безвекторних переривань на програмному, мікропрограмному й апаратному рівнях. Одержати навички розробки мікропрограм з використанням мнемонічного мікроасемблера.

Теоретичні відомості

Системи переривань процесора

Перериванням називається тимчасове припинення виконання програми, перехід на іншу підпрограму з можливістю повернення в перервану програму.

Переривання бувають:

– внутрішні (програмні й апаратні);

– зовнішні (векторні й безвекторні).

Внутрішні переривання

Внутрішні апаратні переривання відбуваються у випадку виникнення певних подій у процесорі (переповнення розрядної сітки, розподіл на нуль, не надходить сигнал готовність від ВУ довше заданого часу й ін.). При виникненні внутрішнього сигналу вимоги переривання процесор переходить на виконання підпрограми з фіксованою початковою адресою (програмний рівень).

Програмні переривання можуть бути викликані спеціальною командою процесора.

Зовнішні переривання

Сигнали зовнішніх переривань формуються поза процесором і надходять у нього по спеціальних входах. Найбільш пріоритетні - безвекторні переривання. Їхня обробка здійснюється стандартними підпрограмами, розташовуваними по фіксованими початковими адресами. Безвекторні переривання генеруються системою при виникненні в ній певних помилок, наприклад, порушення паритету даних в ОП.

При обміні інформацією між процесором і ВУ використаються зовнішні векторні переривання.

Обмін інформацією з ВУ в режимі переривання

При ініціалізації системи кожному ВУ дозволяється або забороняється працювати в режимі переривання, для чого встановлюється біт дозволу переривання в регістрі стану РС пристрою. При дозволеному перериванні обмін інформацією здійснюється в такий спосіб: якщо процесору дозволено обслуговувати перериванні (прапор дозволу переривання в регістрі словосостояния процесора встановлений, для чого використається спеціальна команда), те після виконання кожної команди перевіряється вимога переривання на вході маскируемого переривання INT. Якщо вимога встановлена, то процесор обслуговує переривання (передає керування програмі обробки переривання), у противному випадку виконує наступну команду.

У більшості процесорів є вхід немаскируемого переривання NMI. Аналіз цього сигналу й виклик відповідної програми обробки відбувається незалежно від того, дозволені або заборонені переривання.

При обслуговуванні переривання виконується наступна послідовність дій:

– Запам'ятовується стан перерваної програми й адреса повернення в крапку переривання (уміст СК);
– Забороняються маскируемые переривання, для чого може бути використаний один із двох способів. Перший - заборона переривань на рівні процесора, тобто установка прапора в регістрі стану процесора. При другому способі - переривання забороняються індивідуально для

кожного ВУ або в РС зовнішнього пристрою, або на рівні контролерів перериванні.

- Для векторного переривання по ШД приймається вектор, що дозволяє обчислити початкову адресу підпрограми обслуговування. Читання вектора стробируется сигналом підтвердження переривання INTA;
- За значенням вектора визначається адреса програми обслуговування переривання, який потім передається керування.

Використання векторних переривань, у відмінності від методу програмного опитування (полинга), дозволяє повністю виключити непродуктивні витрати процесорного часу на аналіз готовності ВУ. Тут ініціатором обслуговування виступає саме ВУ, що дає можливість зменшити час реакції системи. Застосування переривань також дозволяє обслуговувати аварійні ситуації.

Однак використання переривань ускладнює процесор (його мікропрограму), інтерфейс ВУ, а також вимагає введення контролера переривань або їхньої групи.

Апаратна реалізація режиму переривань

Існує два основних способи реалізації апаратних засобів обслуговування переривань:

- Централізована система переривань;
- Децентралізована система переривань.

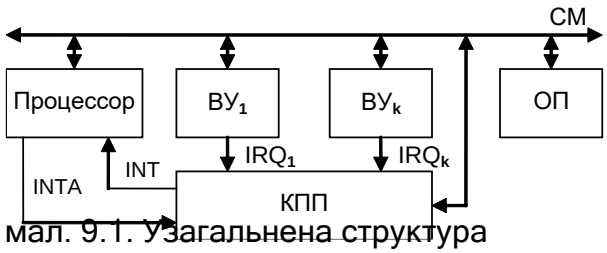
У першому випадку використається спеціальний контролер переривань, у другому - апаратури обслуговування переривань розподілена між інтерфейсами ВУ.

Незалежно від способу побудови, на апаратні засоби покладають наступні завдання:

- аналіз вимог переривань від різних ВУ з урахуванням їх пріоритетності;
- формування загального сигналу переривання INT від ВУ для процесора;
- передача відповідного вектора на ШД по сигналі підтвердження переривання INTA.

Системи із централізованим контролером (арбітром)

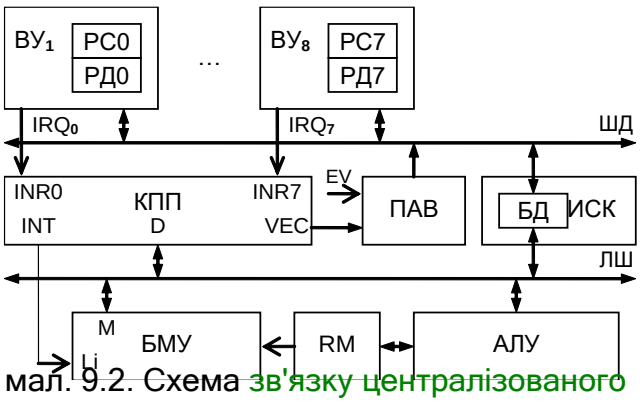
Загальна схема системи із централізованим контролером пріоритетних переривань показана на **Ошибка! Источник ссылки не найден.**9.1, де КПП – контролер пріоритетних переривань, IRQ_i – запит на переривання від і-го ВУ, INT – загальний сигнал вимоги



мал. 9.1. Узагальнена структура системи із централізованим контролером переривань

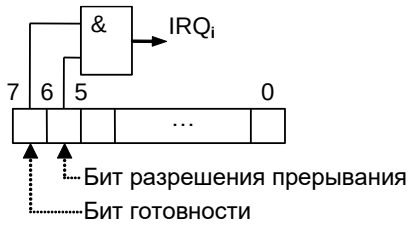
переривань, INTA – сигнал підтвердження переривання.
Готове до обміну ВУ виставляє сигнал запиту переривання (IRQ_i). КПП порівнює ранг пріоритету поточної програми й пристрою, що запросило. Якщо ранг ВУ вище чим ранг програми, то КПП формує сигнал INT. При надходженні сигналу INT на процесор, останній завершує до кінця чергову команду, після чого виробляє сигнал INTA і по ШД СМ читає відповідний вектор. Далі завантажений вектор використається для обчислення початкових команд процедури обробки переривань.

У використовуваному для проведення лабораторних занять емуляторі є централізований КПП, зв'язок якого із системою представлена на **Ошибка! Источник ссылки не найден.**9.2. Сигнали IRQ_i запитів переривання від ВУ надходять на входи INR_i КПП. Контролер здійснює арбітраж вступників сигналів й, при необхідності, формує вимога переривання INT. Сигнал INT можна подати на один із входів Li логічної умови БМУ й проаналізувати на мікропрограмному рівні. Номер переривання (ВУ) установлюється на виході VEC КПП. За допомогою схеми перетворення адреси вектора ВПАВШИ, його можна перетворити в 16-розрядний вектор переривання. При надходженні сигналу EV вектор видається на ШД, і може бути завантажений в АЛУ або БМУ через буфер даних інтерфейсу системного каналу ИСК.



мал. 9.2. Схема зв'язку централізованого КПП із микроЭВМ

Зовнішній пристрій формують сигнал IRQi у тому випадку, коли в його регістрі стану РС (**Ошибка! Источник ссылки не найден.**.3) установлено біт



мал. 9.3. Схема формування запиту на переривання у ВУ

дозволу переривання (6-й розряд) і, крім того, пристрій готовий до обміну (установлений біт готовності). Біт готовності встановлюється автоматично після виконання зазначеного в директиві ACCEPT DEV кількості тактів (від моменту звертання до регістра дані пристрої). Для установки біта дозволу переривання необхідно виконати команду виводу інформації в регістр стану. У виведених даних попередньо повинна бути встановлена одиниця в 6-м розряді. Для читання номера переривання із системі використається спеціальна мікрооперація КПП (аналог описаного раніше сигналу INTA) READ VR, що формує номер вектора на виході VEC КПП. Для перетворення цього номера у вектор переривання необхідно в цьому ж такті сформувати сигнал EV – читання даних з ПАВ. Для заборони переривання, що надійшло, може використатися мікрооперація CLR CLR IR, VR. Використовуваний КПП також дозволяє маскувати групу переривань від ВУ. Для чого через 8-розрядний вхід D необхідно завантажити маску, використовуючи команду LOAD MR, <маска>.

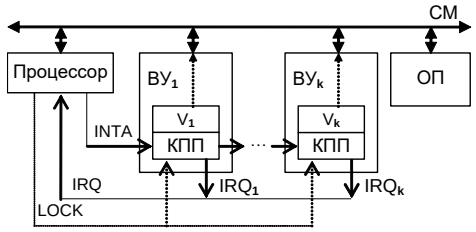
Приклад 9.1 Мікропрограма настроювання КПП, читання в РОН АЛУ вектора переривання з ПАВ і скидання запиту, що надійшов, переривання в КПП.

```
link vec[2]:        \ По перериванню 2 (IRQ₂ від
0CDEFh              ВУ №2)
                     \ сформуємо вектор 0CDEFh
link L[4]:INT       \ Підключаємо сигнал перери-
                    вання від КПП
                     \ до входу L4 логічних умов
                    БМУ
{reset;}            \ скидання КПП
{load               \ дозволити обробку перери-
mr,11111011%;}      вання IRQ2
        .           \ Мікропрограма виконання
        .           \ команди, включаючи форму-
                    вання
        .           \ адреси наступної команди.
{cjp not INT,       \ Якщо INT=0,то перейти до чи-
MakeInt;}           тання вектора
        ⋮           \ інакше почати обробку наступ-
                    ної
```

```
                    \ команди.
MakeInt             \ Мікропрограма читання век-
                    тора
{ev;read vr;or      \ читання вектора в R9
r9,bus_d,z;}
{clr ir,vr;}        \ скидання розряду в регістрі IR,
                    що
                    \ відповідає считтанному век-
                    тору
\ Продовження мікропрограми обслуговування пере-
ривань
```

Системи з розподіленим контролером (дейзи-цепочка)

При використанні розподіленого контролера переривань загальна схема системи має вигляд представлений на **Ошибка! Источник ссылки не найден.**, де КПП – контролер пріоритетних переривань (розподілений арбітр), V_i – регістр для зберігання вектора переривання і-го пристрою, IRQ – загальний сигнал вимоги переривання, INTA -сигнал підтвердження переривання, IRQ_i – запит на переривання від і-го ВУ, LOCK – сигнал підготовки до читання вектора (у деяких системах може отсутствовать).



мал. 9.4. Узагальнена структура системи з розподіленим контролером переривань

Виходи IRQ_i об'єднані через монтажне "**И**". Готове до обміну ВУ виставляє запит на переривання IRQ_i. При надходженні сигналу IRQ процесор завершує команду й формує сигнал підтвердження INTA, що поширюється по пріоритетному ланцюжку (дейзи-цепочке). У системі виконується географічний пріоритет (плата того ВУ, що ближче до процесора, має більший пріоритет). ВУ, що виставило запит IRQ_i, при надходженні сигналу INTA видає вектор переривання з регістра V_i на ШД і блокує подальше поширення сигналу INTA по дейзи-цепочке, тобто в даному ВУ ланцюжок розривається. У деяких системах, перед формуванням сигналу INTA, необхідно зафіксувати вимоги переривань IRQi у тригерах КПП, для чого використається додатковий сигнал LOCK. Така фіксація дозволять уникнути помилкових спрацьовувань при надходженні нових запитів IRQ_i у момент читання вектора. Читання вектора, як і читання будь-якого регістра зовнішнього пристрою, може супроводжуватися формуванням сигналу готовності ВУ RDD.

Достоїнствами дейзи-цепочки є: простота нарощування кількості ВУ й менше число звертань до пам'яті при переході на процедуру обслуговування. До недоліків можна віднести неможливість динамічного (у процесі

роботи системи) зміни рангу пріоритету, а також відсутність можливості одночасного маскирования переривань від трохи ВУ.

У використовуваному для проведення лабораторних занять емуляторі є розподілений КПП, що може працювати як спільно, так і незалежно від централізованого КПП. Для підключення ВУ до дейзи-цепочки застосовується розширений формат директиви настроювання ВУ ACCEPT DEV.

Приклад 9.2 Настроїти ВУ з номером 4 на уведення, ВУ з номером 6 на вивід і підключити їх до дейзи-цепочке.

```
ACCEPT DEV[4]:  I,      \ Пристрій уведення
                1234h   \ Адреса регістра стану РС
                ,       1234h
                1236h   \ Адреса регістра даних РД
                ,       1236h
                10,     \ RDD формується через 10 тактів
                50,     \ Готовність до уведення й формування
                        \ IRQ через 50 тактів
                4444    \ Включено в дейзи-цепь, вектор 4444
accept Dev[6]:  O,6,7,  \ ВУ виводу, адреса РС=6, адреса РД=7
                10,10,  \ Час RDD і готовності по 10 тактів
                6       \ Включено в дейзи-цепь, вектор 6
```

При підключенні декількох ВУ до дейзи-цепочке, пристрій з меншим номером має найбільший пріоритет. Загальний сигнал переривання від дейзи-цепочки називається IRQ. Він може бути підключений до будь-якого входу Li логічних умов БМУ за допомогою директиви LINK. Перед читанням вектора пристрою, підключеного до дейзи-цепочке, необхідно в плині одного такту сформувати сигнал LOCK, після чого заданий директивою ACCEPT DEV вектор буде виданий на шину даних. Формування достовірного значення вектора на ШД супроводжується сигналом RDD.

Підготовка до лабораторної роботи

– Доробити мікропрограму, отриману при виконанні лабораторних робіт 3, 4, 5 й 6, включивши до складу команд безадресну команду безумовного повернення з підпрограми обробки переривання (код операції вибрати самостійно). У відмінності від команди безумовного повернення з підпрограми, нова команда повинна забезпечувати відновлення регістра стану процесора, шляхом зчитування його значення зі стека.

– Розробити й включити в отриману мікропрограму (після етапу формування адреси наступної команди) додатковий етап, пов'язаний з обробкою зовнішніх безвекторних і векторних переривань

Мікропрограма обробки переривань (мікропрограмний рівень) повинна забезпечувати перехід на підпрограму (програмний рівень) обслуговування безвекторного переривання від нульового зовнішнього пристрою й векторного переривання від зовнішнього пристрою з номером a_2*2+a_1+1. Безвекторне переривання має в системі найвищий пріоритет. Зазначена мікропрограма повинна зберегти в стеці адреса повернення й стан перерваної програми (уміст регістра стану), а потім занести в лічильник команд адреса першої команди підпрограми обслуговування переривання. Як покажчик стека використати R6, а регістром стану можна вважати будь-який робочий регістр процесора.

Оброблювач безвекторного переривання

Адреса першої команди підпрограми обробки безвекторного переривання зазначений у табл. . Для аналізу сигналу вимоги безвекторного переривання IRQ0 від 0-го зовнішнього пристрою використати один із входів Li мультиплексора логічних умов (по розсуду розроблювача).

Підпрограма (програмний рівень) обслуговування безвекторного переривання повинна забезпечувати збереження акумулятора (R15) у РД ВУ з номером a_3*2+a_2, після чого повернутися в перервану програму. Для повернення в з оброблювача використати розроблену з відповідності з п. 1 команду безумовного повернення з підпрограми обробки переривання.

Оброблювач векторного переривання

Розроблювальна мікропрограма повинна забезпечити опитування вимоги переривання INT від централізованого КПП або IRQ від розподіленого КПП. Тип контролера визначається відповідно дотабл. 9.29.2. Зчитувальний вектор уважати адресою програми обробки векторного переривання. У мікропрограмі занести його в лічильник команд і зберегти інформації для повернення. Адреса програми оброблювача векторного переривання вибрати з табл. .

Табл. 9.1

a_4	a_3	a_2	Адреса процедури обробки	
			Без-століття.	Вектор.
0	0	0	00AAh	00F0h
0	0	1	00ABh	00E8h
0	1	0	00ACh	00E6h
0	1	1	00ADh	00E4h
1	0	0	00AEh	00D2h
1	0	1	00AFh	00D0h
1	1	0	00B2h	00CFh
1	1	1	00B7h	00C8h

Підпрограмма обслуговування векторного переривання повинна відповідати п. 2 опису лабораторної роботи 9. Пристрій з номером a_2*2+a_1+1 є пристроєм уведення. Пристрій виводу вибрати самостійно.

Скласти в кодах розроблених команд дві підпрограми обробки переривань (векторного й безвекторного). У якості фонової записати циклічну програму (остання команда безумовного переходу передає керування першій команді), що у регістрах стану ВУ, що працюють по перериваннях, по черзі встановлює й скидає біт дозвіл переривань.

Порядок виконання роботи

– Налагодити розроблену мікропрограму й програми з використанням програмного емулятора.

– Зробити виводи по роботі.

Контрольні питання

Табл. 9.2

a_1	Тип КПП
0	Централізований
1	Розподілений

– Охарактеризуйте основні системи прерывани.

– Який тип переривань використається для роботи із зовнішніми пристроями?

– Назвіть основні достоїнства методу обслуговування ВУ в режимі переривань від методу опитування.

– Намалюйте узагальнену структуру системи із централізованим контоллером пріоритетних переривань. Поясните її роботу.

– Намалюйте узагальнену структуру системи з розподіленим контоллером пріоритетних переривань. Поясните роботу системи.

Додаткова література /5,6/.