Національний технічний університет України “Київський політехнічний інститут”
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Розрахунково-графічна робота з дисципліни “Архітектура комп'ютерів”

Тема: Обробка інформації на програмному, мікропрограмному та апаратному рівнях

Виконав:
Грибенко Д. В.
ФІОТ гр. ІО-72
Залікова книжка: 7202

Перевірив:
Жабін В. І.

Київ — 2009

Зміст

1 Завдання

1.1 Вступ

Завдання розрахункової роботи передбачає розробку мікропрограми для обчислювальної системи, побудованої на базі комплекта КМ1804 з мікропрограмним управлінням. Розроблена мікропрограма повинна забезпечувати виконання команд, які зберігаються в оперативній пам'яті. Використовуючи розроблену систему команд, необхідно розробити програму обчислення функції відповідно до варіанту.

1.2 Вихідні дані

Розроблена програма повинна виконувати обчислення виразу

$$A = (X + Y) \cdot Z$$

Операнди X, Y, Z вводяться з пристрою введення, результат виводиться у пристрій виведення. Програма та її дані повинні розміщуватись в ОП починаючи з відповідних заданих адрес. Відповідно до варіанту (лабораторна робота розробки мікропрограми множення), X та Y мають інтерпретуватись як числа в ДК, а Z — як число в ПК. Результат множення повинен бути в ПК. Порядок виводу результату у пристрій виведення не заданий та обирається самостійно.

Вихідні дані для розробки мікропрограми та програми відповідно до варіанту:

- Початкова адреса програмного коду в ОП: 0x1A.
- Початкова адреса даних в ОП: 0xDA.
- Зовнішній пристрій вводу: номер 7, адреса РСС 0xE0.
- Зовнішній пристрій вводу: номер 0, адреса РСС 0xE4.
- Зовнішній пристрій виводу: номер 2, адреса РСС 0xEA.

2 Опис архітектури та схема алгоритму обчислень

Відповідно до варіанту, адресний простір ОП використовується так, як зазначено на рисунку 1.

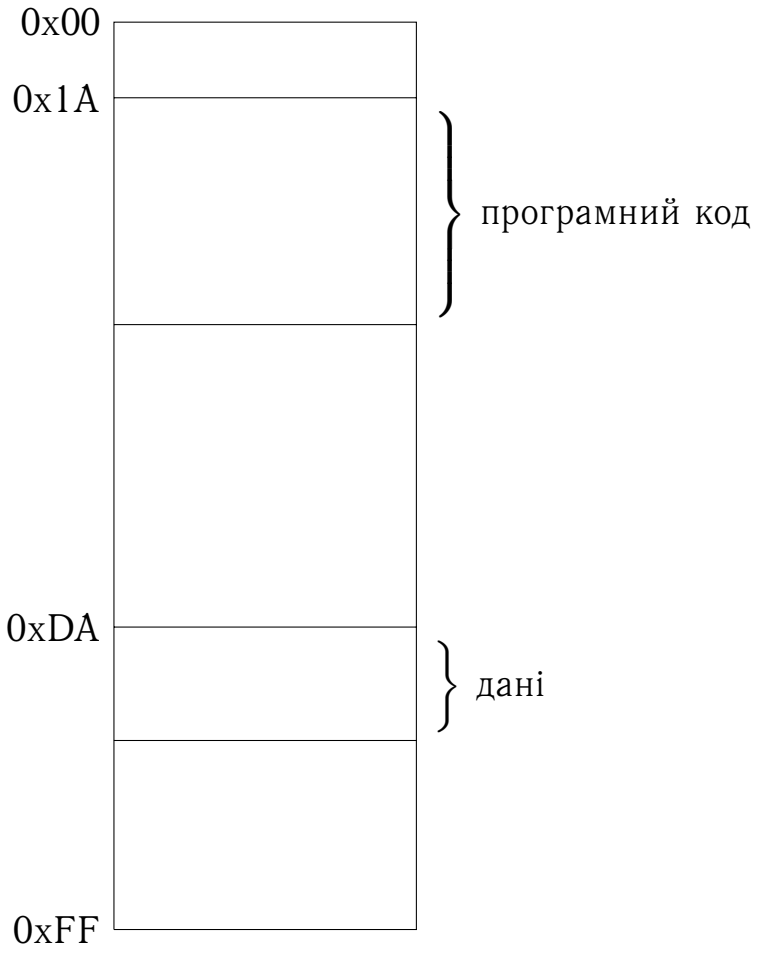

Рис. 1 – Адресний простір ОП

Адресний простір зовнішніх пристроїв, відповідно до варіанту, представлений в таблиці 1. Скорочення: РСС — регістр слова стану, РД — регістр даних.

Табл. 1 – Адресний простір зовнішніх пристроїв

Номер	Тип	Адреса РСС	Адреса РД
7	Input	0xE0	0xE2
0	Input	0xE4	0xE6
2	Output	0xEA	0xEC

В мікропрограмному комплексі доступні 16 регістрів. Розподіл їх призначення зазначений в таблиці 2. Програмно доступними є ті регістри, які можуть бути операндами (явними або ні) в розроблюваній системі команд. В даному випадку програмно доступними є регістри `r0` — `r7`, а також `rlo`, `rhi`. Регістри `r8` — `r13` є робочими, тобто використовуються виключно мікропрограмою. Так, регістр `rcom` (`r8`) містить поточну команду, що виконується.

Регістри `r0` — `r7` доступні за допомогою прямої регістрової адресації. Регістри `rlo`, `rhi` є спеціальними регістрами, в які записується результат множення. Крім того, обмін даними з пам'яттю та зовнішніми пристроями відбувається тільки через `rhi`. Регістри `rhi`, `rlo` доступні для читання за допомогою команд `mfhi`, `mflo`. Запис в `rhi` виконується за допомогою `mthi`.

Табл. 2 – Модель програміста — призначення регістрів системи

Регістр	Призначення

`r0`
`r1`
`r2`
`r3`
`r4`
`r5`
`r6`
	Регістри загального призначення (РЗП)

`r7`	Лічильник команд `pc` (доступний програмно)
`r8`	Регістр команди `rcom` (недоступний програмно)

`r9`
`r10`
`r11`
`r12`
`r13`
	Робочі регістри

`r14`	`rlo` (доступний програмно)
`r15`	`rhi` (доступний програмно)

3 Система команд і програма обчислень

Розроблена система команд представлена в таблиці 3.

Табл. 3 – Система команд

Код операції	Мнемоніка	Призначення
Одноадресні команди		
0000	—	Заборонений
0010	`mul src`	`rhi, rlo := rhi * src`
0100	`b addr`	Безумовний перехід на вказану адресу (branch)
0101	`bdr addr`	Перехід на вказану адресу якщо встановлений 7-й біт rhi (branch if device ready)
0110	`load addr`	Зчитування даних з пам'яті за адресою addr в rhi
0111	`store addr`	Запис в пам'ять за адресою addr даних із rhi
1000	`in addr`	Зчитування даних з зовнішнього пристрою з адресою addr в rhi
1001	`out addr`	Запис в зовнішній пристрій з адресою addr даних із rhi
1111	`halt`	Закінчити виконання програми
Двоадресні команди		
10000	`mfhi dest`	`dest := rhi` (move from rhi)
10001	`mflo dest`	`dest := rlo` (move from rlo)
10010	`mthi src`	`rhi := src` (move to rhi)
10011	`add dest, src`	`dest := dest + src`

Важливо відмітити, що завдання передбачає різну кількість біт для кода операції в одно- та двоадресних командах (4 та 5 біт). Так как код виконання одноадресної команди не буде працювати у випадку, коли команда двоадресна, і навпаки, прийнято наступне рішення: встановити старший (5-й) біт КОП двоадресних команд в 1, а в одноадресних командах вважати 5-й біт КОП рівним 0. Таким чином, повні КОП-и одноадресних команд ма-

ють вигляд 0xxxx, причому в команді зберігається тільки xxxx. Повний КОП двоадресної команди має вигляд 1xxxx і зберігається в команді повністю. Це дозволить мати до 15 одноадресних та до 16 двоадресних команд, всього 31. КОП 0000 є забороненим, тому що по відповідній адресі знаходиться код основного циклу виконання команди.

Для одноадресних команд розроблено пряму та непряму адресацію. Для двоадресних команд розроблено тільки пряму регістрову адресацію (відповідно до варіанту).

У вищевказаних командах розроблена програма обчислення виразу $A = (X + Y) \cdot Z$. Операнди X, Y, Z (у такій послідовності) вводяться з пристрою введення номер 7. Введені X та Y інтерпретуються як числа в ДК, а Z — як число в ПК. Обчислений результат (в ПК) виводится у пристрій виводу номер 2 починаючи зі старшого подвійного слова.

Програма в мнемонічних та машинних кодах представлена в наступному лістингу. Зліва в коментарях від кожної команди та слова даних вказана адреса в ОП, а справа — відповідний вміст ОП. Алгоритм програми предствлений на рисунку 2

```
.text
                wait_1:                         /* 5432109876543210 */
/* 0x1A */              in      rsw_1           /* 0100000011011010% */
/* 0x1B */              bdr     read_x_addr     /* 0010100011011101% */
/* 0x1C */              b       wait_1_addr     /* 0010000011011100% */
                read_x:
/* 0x1D */              in      rdata_1         /* 0100000011011011% */
/* 0x1E */              mfhi    %r0             /* 1100000000000000% */

                wait_2:
/* 0x1F */              in      rsw_1           /* 0100000011011010% */
/* 0x20 */              bdr     read_y_addr     /* 0010100011011111% */
/* 0x21 */              b       wait_2_addr     /* 0010000011011110% */
                read_y:
/* 0x22 */              in      rdata_1         /* 0100000011011011% */
/* 0x23 */              mfhi    %r1             /* 1100000000100000% */

/* 0x24 */              add     %r1, %r0        /* 1100110000100000% */
/* 0x25 */              mthi    %r1             /* 1100100000000001% */
/* 0x26 */              store   tmp_1_addr      /* 0011100011100011% */

                wait_3:
```

```
/* 0x27 */              in      rsw_1           /* 0100000011011010% */
/* 0x28 */              bdr     read_z_addr     /* 0010100011100001% */
/* 0x29 */              b       wait_3_addr     /* 0010000011100000% */
                read_z:
/* 0x2A */              in      rdata_1         /* 0100000011011011% */

/* 0x2B */              mul     tmp_1           /* 0001000011100010% */
/* 0x2C */              mfhi    %r0             /* 1100000000000000% */

                wait_4:
/* 0x2D */              in      rsw_3           /* 0100000011100100% */
/* 0x2E */              bdr     write_a_hi_addr /* 0010100011100111% */
/* 0x2F */              b       wait_4_addr     /* 0010000011100110% */

                write_a_hi:
/* 0x30 */              mthi    %r0             /* 1100100000000000% */
/* 0x31 */              out     rdata_3         /* 0100100011100101% */

                wait_5:
/* 0x32 */              in      rsw_3           /* 0100000011100100% */
/* 0x33 */              bdr     write_a_lo_addr /* 0010100011101001% */
/* 0x34 */              b       wait_5_addr     /* 0010000011101000% */

                write_a_lo:
/* 0x35 */              mflo    %r0             /* 1100010000000000% */
/* 0x36 */              mthi    %r0             /* 1100100000000000% */
/* 0x37 */              out     rdata_3         /* 0100100011100101% */

/* 0x38 */              halt                    /* 0111100000000000% */

.data
                rsw_1:
/* 0xDA */              .short 0x00E0           /* 0x00E0 */
                rdata_1:
/* 0xDB */              .short 0x00E2           /* 0x00E2 */
                wait_1_addr:
/* 0xDC */              .short wait_1           /* 0x001A */
                read_x_addr:
/* 0xDD */              .short read_x           /* 0x001D */
                wait_2_addr:
/* 0xDE */              .short wait_2           /* 0x001F */
                read_y_addr:
/* 0xDF */              .short read_y           /* 0x0022 */
                wait_3_addr:
/* 0xE0 */              .short wait_3           /* 0x0027 */
                read_z_addr:
```

```
/* 0xE1 */              .short read_z           /* 0x002A */
                tmp_1:
/* 0xE2 */              .short 0x0000           /* 0x0000 */
                tmp_1_addr:
/* 0xE3 */              .short tmp_1            /* 0x00E2 */
                rsw_3:
/* 0xE4 */              .short 0x00EA           /* 0x00EA */
                rdata_3:
/* 0xE5 */              .short 0x00EB           /* 0x00EC */
                wait_4_addr:
/* 0xE6 */              .short wait_4           /* 0x002D */
                write_a_hi_addr:
/* 0xE7 */              .short write_a_hi       /* 0x0030 */
                wait_5_addr:
/* 0xE8 */              .short wait_5           /* 0x0032 */
                write_a_lo_addr:
/* 0xE9 */              .short write_a_lo       /* 0x0035 */
```

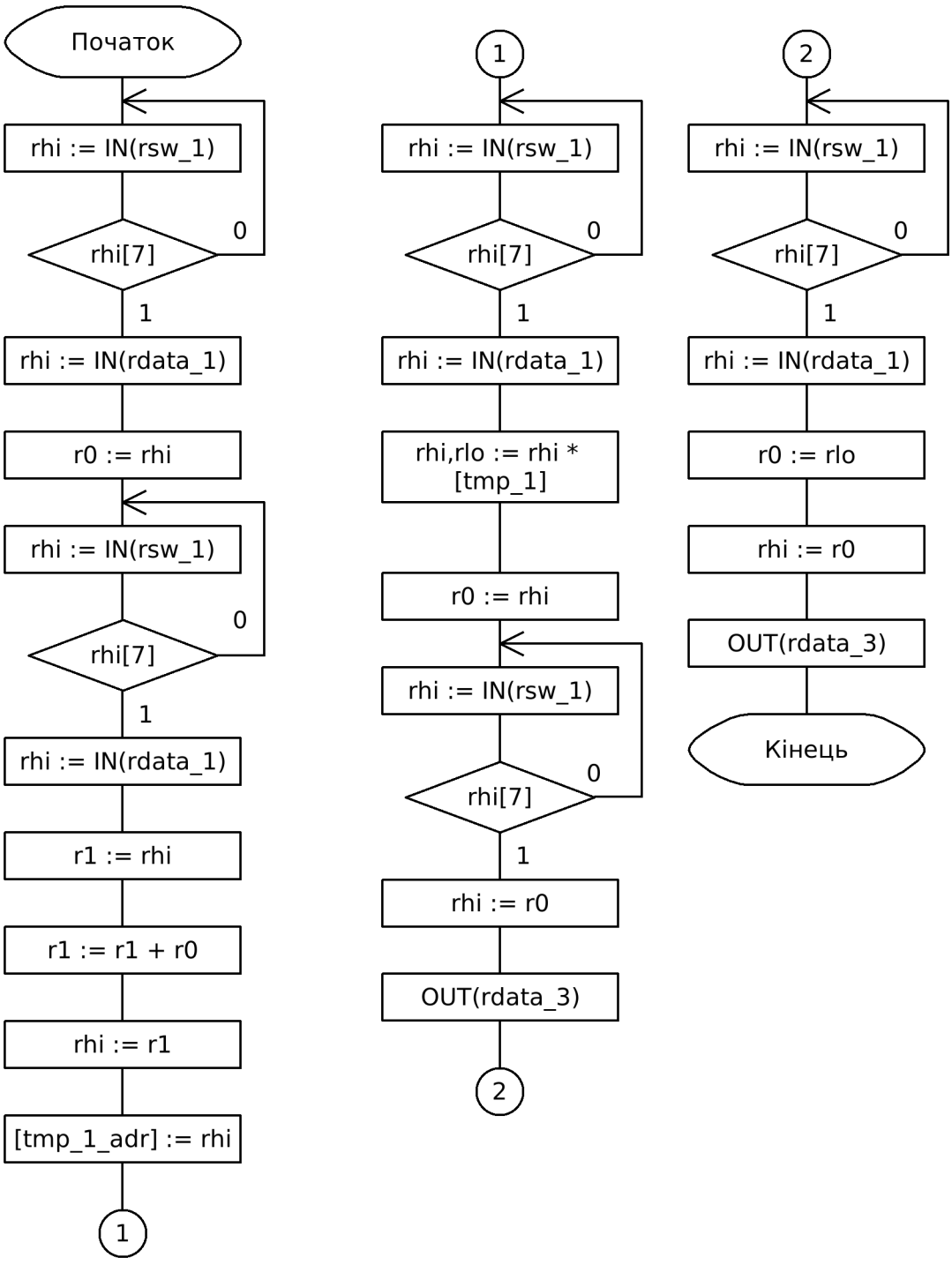

Рис. 2 – Алгоритм програми обчислення заданої функції

4 Структура системи і мікропрограма

Структурна схема ЕОМ, з зазначенням керуючих сигналів, що використовувались при розробці мікропрограми, наведена на рисунку 3. На рисунку не показаний блок пріорітетних переривань, так як він не використовувався для розобки даної мікропрограми.

На рисунку використовуються наступні скорочення:

- ША — шина адреси,
- ШД — шина даних,
- BUS_D — локальна шина,
- ОП — оперативна пам'ять,
- ПВВ — пристрої вводу/виводу,
- РАД — регістр адреси,
- БД — буфер даних,
- МУ — мультиплексор умов,
- СФАМ — схема формування адреси мікрокоманди,
- ПМК — пам'ять мікрокоманд,
- БМ — буфер M,
- БР — буфер P,
- БК — буфер K,
- БРС — блок регістра стану,
- АЛБ — арифметико-логічний блок,
- БЗ — блок зсуву,
- БУ — буфер Y,
- НОЗП — надоперативний запам'ятовуючий пристрій,
- EWH — дозвіл запису старшої частини адреси в РАД,
- EWL — дозвіл запису молодшої частини адреси в РАД,
- OEY — дозвіл видачі результату з АЛБ на BUS_D,
- MSA — куруючий сигнал мультиплексору вибора операнду в АЛБ по каналу A,
- MSB — куруючий сигнал мультиплексору вибора операнду в АЛБ по каналу B,
- CO, ZO, VO, NO — ознаки, що формуються в АЛБ (перенос, нуль,

переповнення, від'ємний результат).

Блок-схема мікроаглогитму циклу виконання команди (читання, розпакування, виконання, формування наступної адреси) представлена на рисунках 4, 5, 6, 7, 8.

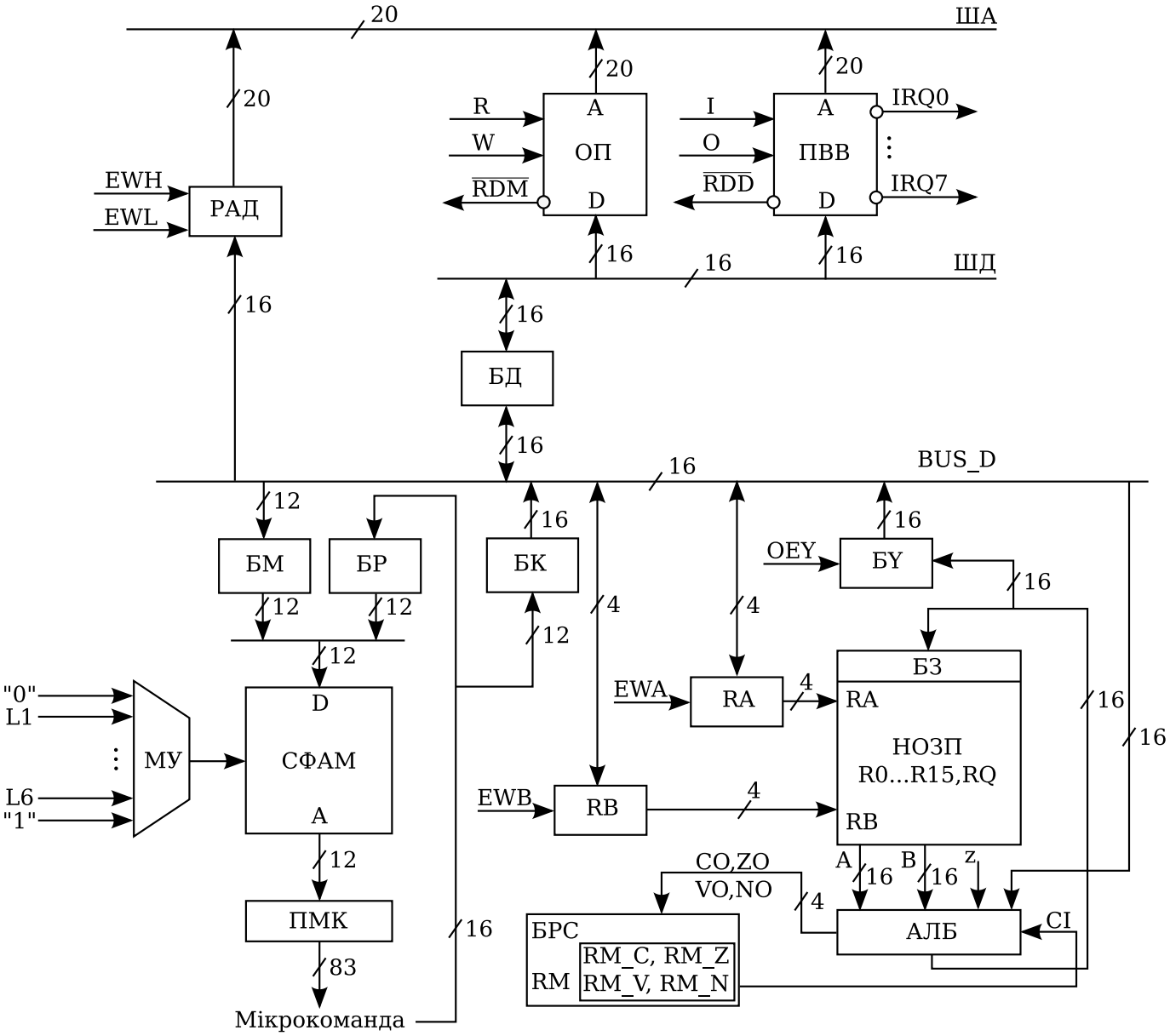

Рис. 3 – Структура ЕОМ

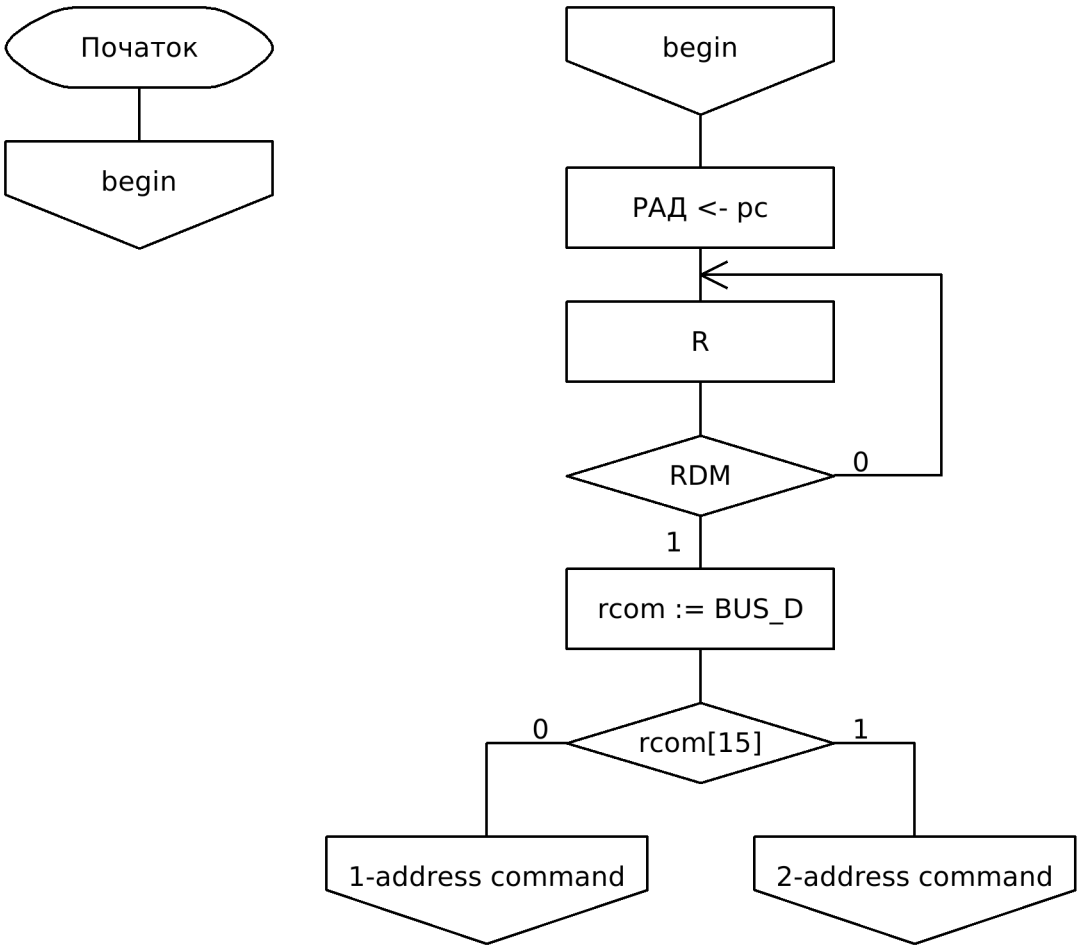

Рис. 4 – Мікроалгоритм циклу виконання команди. Читання та розпакування команди

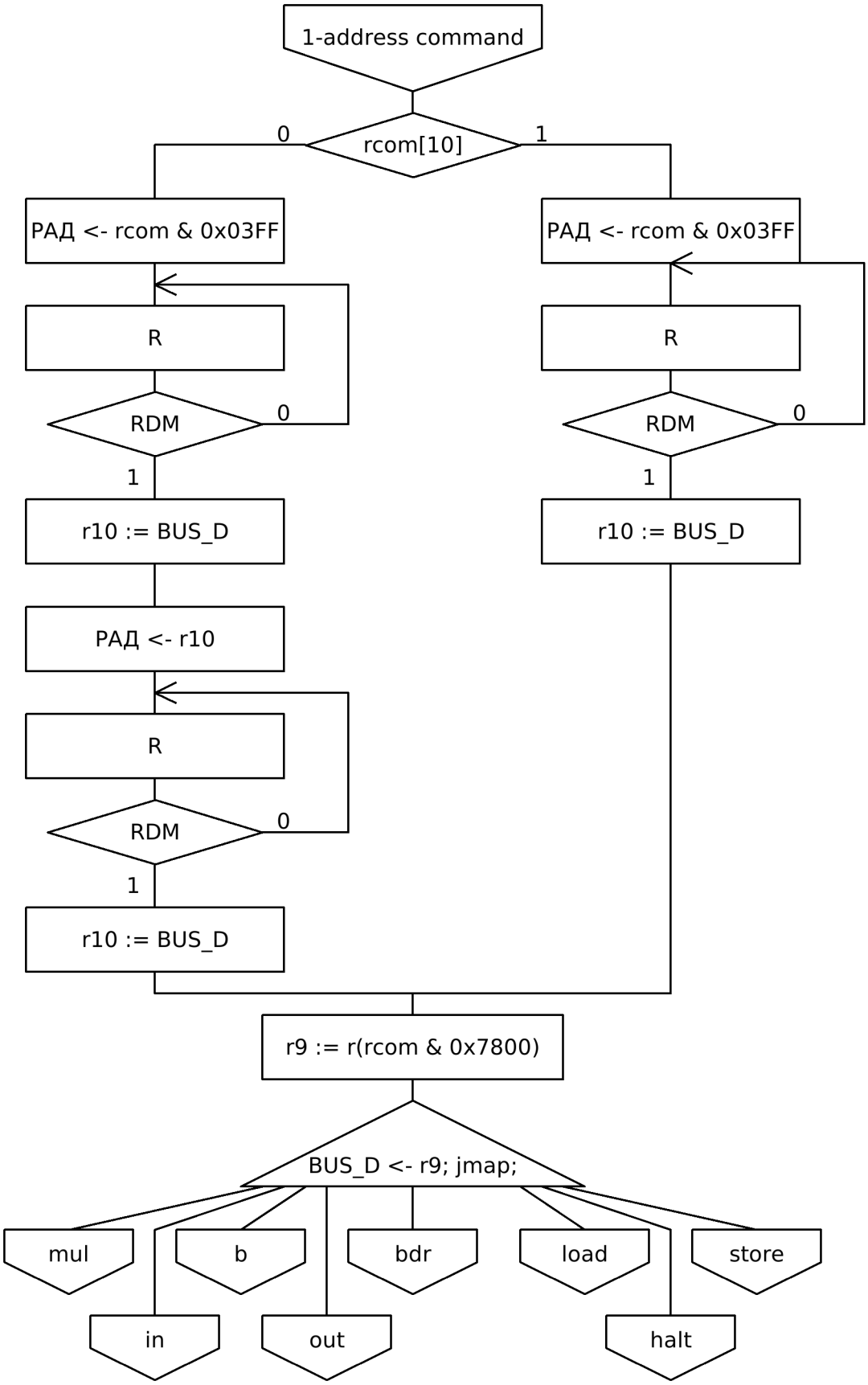

Рис. 5 – Мікроалгоритм циклу виконання команди. Розпакування 1-адресної команди

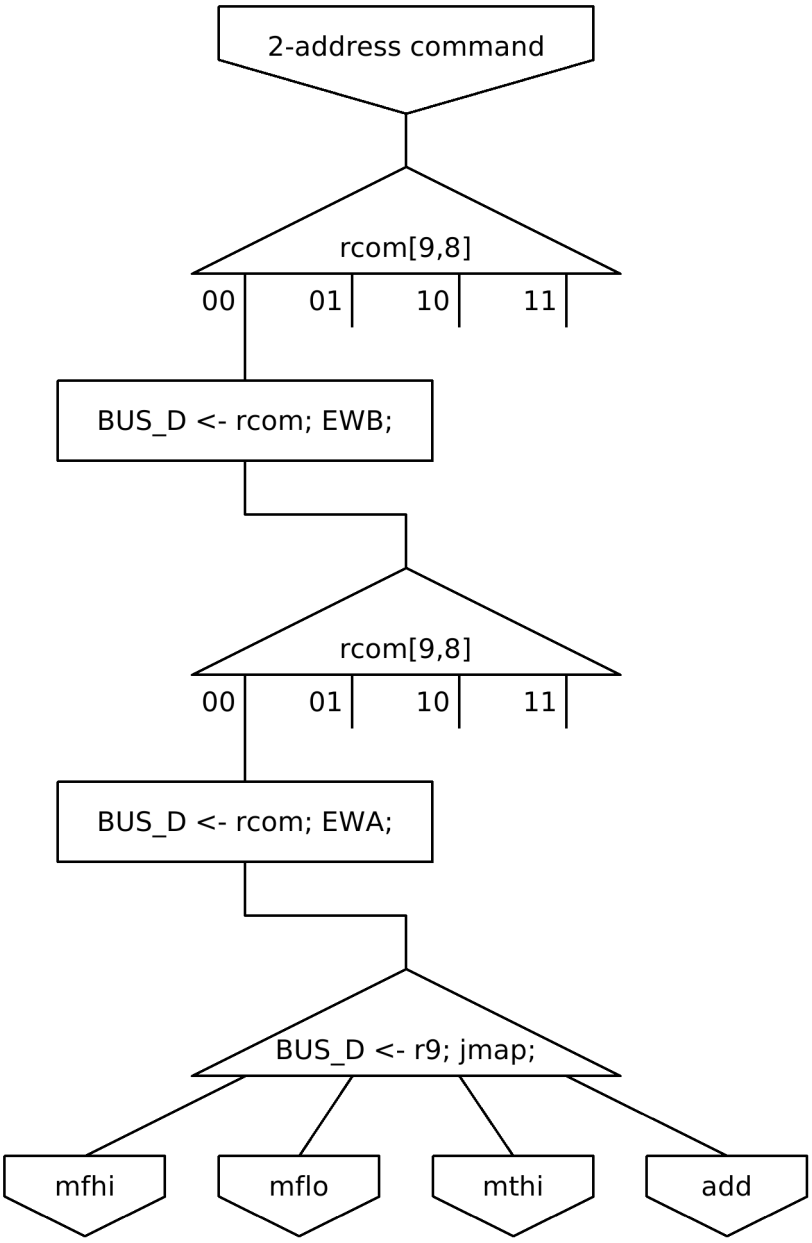

Рис. 6 – Мікроалгоритм циклу виконання команди. Розпакування 2-адресної команди

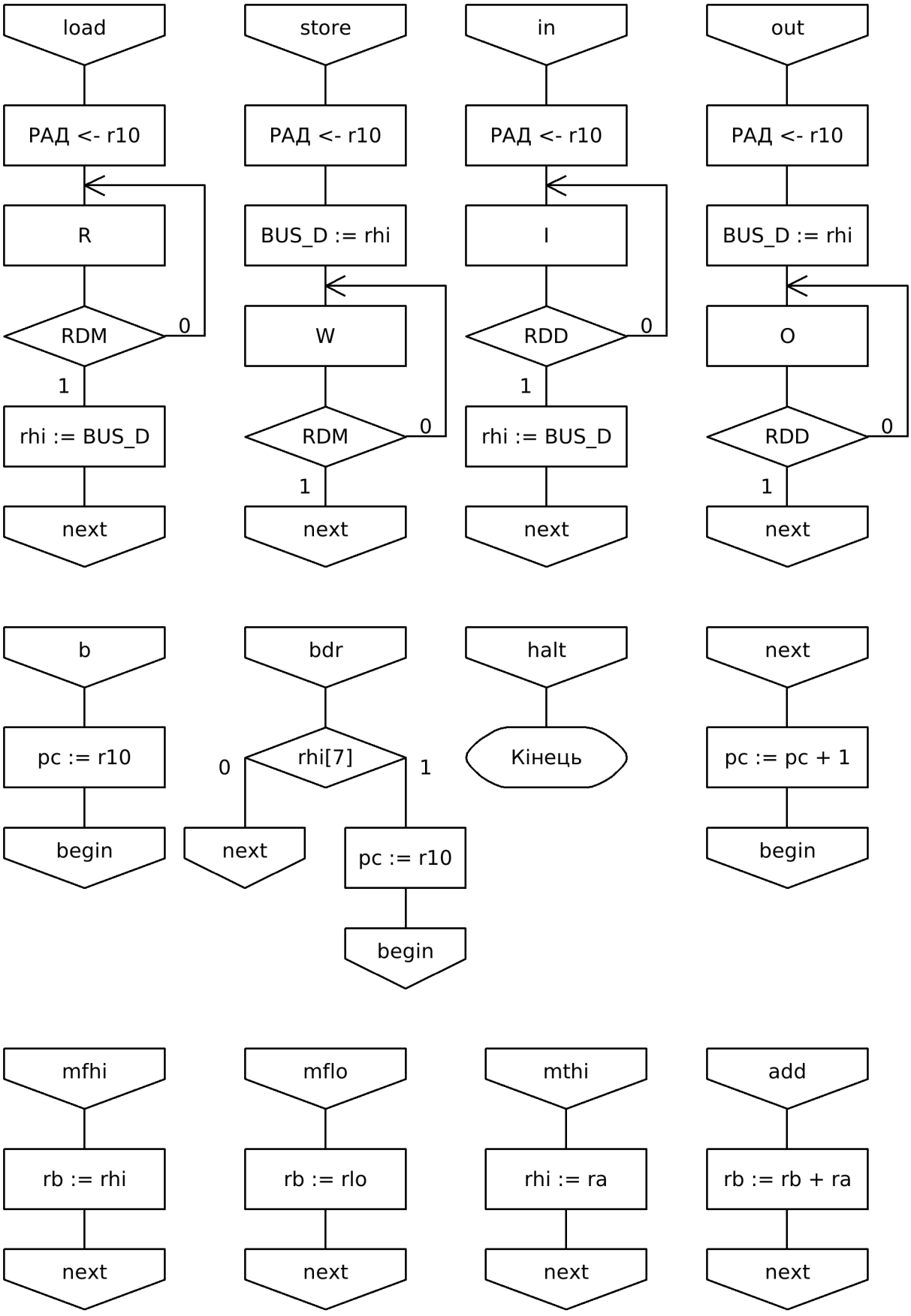

Рис. 7 – Мікроалгоритм циклу виконання команди. Виконання операцій. Формування наступної адреси

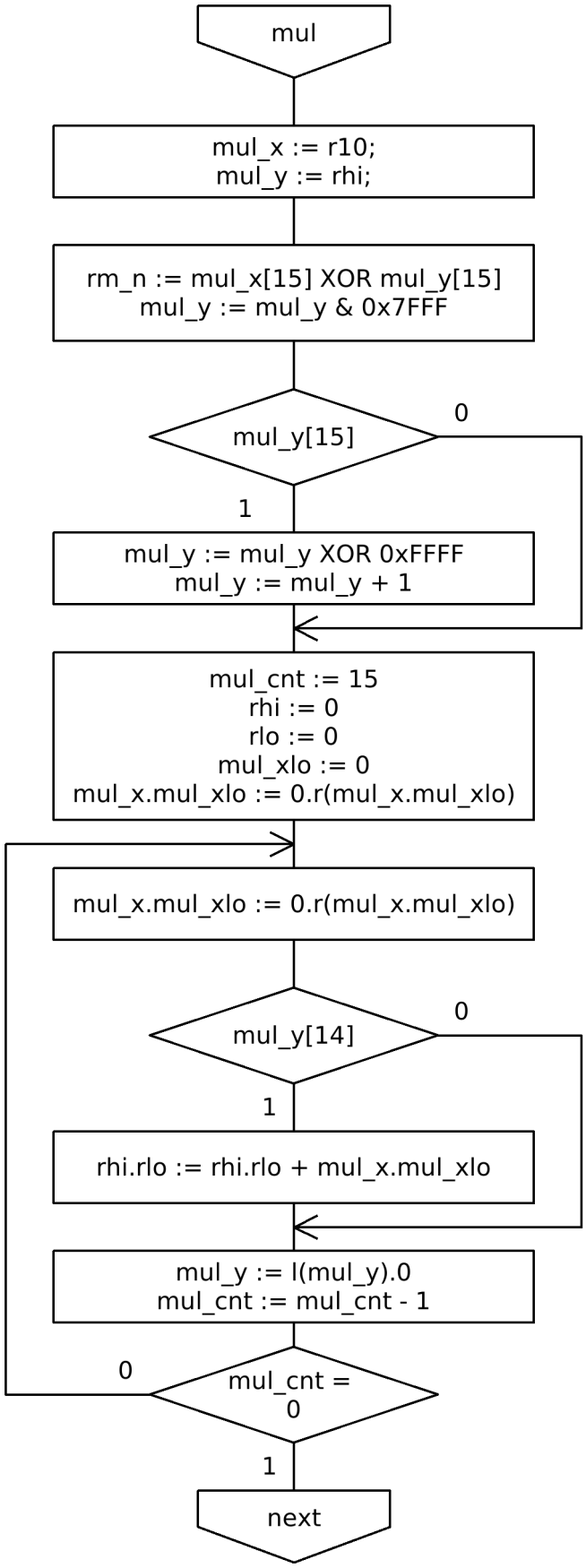

Рис. 8 – Мікроалгоритм циклу виконання команди. Виконання операції множення

Мікропрограма в мнемонічному виді:

```
accept rdm_delay : 4
link l1 : ct
link l2 : rdm
link l3 : rdd
link ewh : 10
link m : z, z, z, 14, 13, 12, 11, 10, z, z, z, z
link ra : z, 2, 1, 0
link rb : z, 7, 6, 5

accept dev[7] : i, 0E0h, 0E2h, 4, 70
accept dev_buf[7] : 00001h, 0FFFEh, 00002h \ result should be 8000 0002h

accept dev[0] : i, 0E4h, 0E6h, 4, 50
accept dev_buf[0] : 01111h

accept dev[2] : o, 0EAh, 0ECh, 4, 100

equ pc   : r7
equ rcom : r8
equ rhi  : r15
equ rlo  : r14

equ mul_cnt : r12
equ mul_y   : r10
equ mul_xlo : r13
equ mul_x   : r11

\               5432109876543210
equ c_fmt     : 1000000000000000%
equ c1_opcode : 0111100000000000%
equ c1_ta     : 0000010000000000%
equ c1_addr   : 0000001111111111%
equ c2_opcode : 0111110000000000%
equ c2_ta1    : 0000001100000000%
equ c2_ro     : 0000000011100000%
equ c2_ta2    : 0000000000011000%
equ c2_rt     : 0000000000000111%

\                     98
equ c2_ta1_dir : 0000000000000000%
equ c2_ta1_ind : 0000000100000000%
equ c2_ta1_ai  : 0000001000000000%
equ c2_ta1_ad  : 0000001100000000%

\                          43
```

```
equ c2_ta2_dir : 0000000000000000%
equ c2_ta2_ind : 0000000000001000%
equ c2_ta2_ai  : 0000000000010000%
equ c2_ta2_ad  : 0000000000011000%

\                  4321
equ addr_mul  : 0000100000%
equ addr_b    : 0001000000%
equ addr_bdr  : 0001010000%
equ addr_load : 0001100000%
equ addr_stor : 0001110000%
equ addr_in   : 0010000000%
equ addr_out  : 0010010000%
equ addr_halt : 0011110000%
\ 2-address commands
equ addr_mfhi : 0100000000%
equ addr_mflo : 0100010000%
equ addr_mthi : 0100100000%
equ addr_add  : 0100110000%
\ other
equ addr_main : 1000000000%

accept pc : 01Ah
dw 01Ah : 0100000011011010%
dw 01Bh : 0010100011011101%
dw 01Ch : 0010000011011100%
dw 01Dh : 0100000011011011%
dw 01Eh : 1100000000000000%
dw 01Fh : 0100000011011010%
dw 020h : 0010100011011111%
dw 021h : 0010000011011110%
dw 022h : 0100000011011011%
dw 023h : 1100000000100000%
dw 024h : 1100110000100000%
dw 025h : 1100100000000001%
dw 026h : 0011100011100011%
dw 027h : 0100000011011010%
dw 028h : 0010100011100001%
dw 029h : 0010000011100000%
dw 02Ah : 0100000011011011%
dw 02Bh : 0001000011100010%
dw 02Ch : 1100000000000000%
dw 02Dh : 0100000011100100%
dw 02Eh : 0010100011100111%
dw 02Fh : 0010000011100110%
dw 030h : 1100100000000000%
```

```
dw 031h : 0100100011100101%
dw 032h : 0100000011100100%
dw 033h : 0010100011101001%
dw 034h : 0010000011101000%
dw 035h : 1100010000000000%
dw 036h : 1100100000000000%
dw 037h : 0100100011100101%
dw 038h : 0111100000000000%
dw 0DAh : 000E0h
dw 0DBh : 000E2h
dw 0DCh : 0001Ah
dw 0DDh : 0001Dh
dw 0DEh : 0001Fh
dw 0DFh : 00022h
dw 0E0h : 00027h
dw 0E1h : 0002Ah
dw 0E2h : 00000h
dw 0E3h : 000E2h
dw 0E4h : 000EAh
dw 0E5h : 000ECh
dw 0E6h : 0002Dh
dw 0E7h : 00030h
dw 0E8h : 00032h
dw 0E9h : 00035h

\ ============================================================================
\ ========== macros
macro mov dest, src :
{ or dest, src, z; }

macro inc reg :
{ add reg, reg, z, nz; }

macro dec reg :
{ sub reg, reg, z, z; }

macro bus_d_out src :
{ or nil, src, z; oey; }

macro bus_d_outz :
{ and nil, pc, z; oey; }

macro read_mem dest :
{
  r;
  mov dest, bus_d;
```

```
  cjp rdm, cp;
}

macro write_mem src :
{
  w;
  bus_d_out src;
  cjp rdm, cp;
}

macro read_dev dest :
{
  i;
  mov dest, bus_d;
  cjp rdd, cp;
}

macro write_dev src :
{
  o;
  bus_d_out src;
  cjp rdd, cp;
}

macro load_rm_z :
{ load rm, flags; cem_n; cem_v; cem_c; }

macro load_rm_n :
{ load rm, flags; cem_z; cem_v; cem_c; }

macro load_rm_c :
{ load rm, flags; cem_z; cem_v; cem_n; }

macro cmp a, b :
{ xor nil, a, b; load_rm_z; }

macro jeq addr :
{ cjp rm_z, addr; }

\ ============================================================================
\ ========== address 0

{ cjp nz, mainloop; }

\ ============================================================================
\ ========== main read-decode-execute loop
```

```
org addr_main
mainloop
\ ========== read command ==========
{ bus_d_out pc; ewl; }
{ bus_d_outz;   ewh; }
{ read_mem rcom; }

\ halt if command is entierly zero
{
  mov nil, rcom;
  cjp zo, addr_halt;
}

\ ========== unpack command ==========
{
  and nil, rcom, c_fmt;
  load_rm_z;
}
{ cjp not rm_z, unpack_ta; }

unpack_oa \ one-address
{
  and nil, rcom, c1_ta;
  load_rm_z;
}
{ cjp not rm_z, oa_indir; }

oa_dir \ direct addressing
{ and r10, rcom, c1_addr; }
{ bus_d_out r10; ewl; }
{ bus_d_outz;    ewh; }
{ read_mem r10; }
{ cjp nz, exec_oa; }

oa_indir \ indirect addressing
{ and r10, rcom, c1_addr; }
{ bus_d_out r10; ewl; }
{ bus_d_outz;    ewh; }
{ read_mem r10; }
{ bus_d_out r10; ewl; }
{ bus_d_outz;    ewh; }
{ read_mem r10; }
{ cjp nz, exec_oa; }

unpack_ta \ two-address
```

```
{ and r10, rcom, c2_ta1; }
{ cmp r10, c2_ta1_dir; }
  { jeq ta_r1_dir; }
{ cmp r10, c2_ta1_ind; }
  { jeq ta_r1_ind; }
{ cmp r10, c2_ta1_ai; }
  { jeq ta_r1_ai; }
{ cmp r10, c2_ta1_ad; }
  { jeq ta_r1_ad; }

ta_r1_dir
{
  bus_d_out rcom; ewb;
  cjp nz, ta_r2;
}

ta_r1_ind
{ cjp nz, error; } \ not implemented
\{ cjp nz, ta_r2; }

ta_r1_ai
{ cjp nz, error; } \ not implemented
\{ cjp nz, ta_r2; }

ta_r1_ad
{ cjp nz, error; } \ not implemented
\{ cjp nz, ta_r2; }

ta_r2
{ and r10, rcom, c2_ta2; }
{ cmp r10, c2_ta2_dir; }
  { jeq ta_r2_dir; }
{ cmp r10, c2_ta2_ind; }
  { jeq ta_r2_ind; }
{ cmp r10, c2_ta2_ai; }
  { jeq ta_r2_ai; }
{ cmp r10, c2_ta2_ad; }
  { jeq ta_r2_ad; }

ta_r2_dir
{
  bus_d_out rcom; ewa;
  jmap;                                 \ execute 2-address command
}

ta_r2_ind
```

```
{ cjp nz, error; } \ not implemented
\{ cjp nz, unpack_end; }

ta_r2_ai
{ cjp nz, error; } \ not implemented
\{ cjp nz, unpack_end; }

ta_r2_ad
{ cjp nz, error; } \ not implemented
\{ cjp nz, unpack_end; }

\unpack_end

\ ========== execute command ==========
exec_oa
{ and srl, r9, rcom, c1_opcode; } \ r9 is used only on the following line
{ bus_d_out r9; jmap; }

\ ========== compute next address for non-branch commands ==========
\ after execution, all commands jump here
com_done
{ add pc, 1; }
{ cjp nz, mainloop; }

\ ========== all commands jump here in case of any error ==========
error
{ cjp nz, error; } \ infnite loop

\ ============================================================================
\ ========== multiplication
\ ========== rhi, rlo := mul_x * mul_y
org addr_mul
{ mov mul_x, rhi; }
{ mov mul_y, r10; }

\ ========== prepare the arguments
\ ========== and determine the sign of the result

{
  xor nil, mul_x, mul_y;                \ sign of the result is:
  load_rm_n;                            \ rm_n := sign(mul_x) xor sign(mul_y)
}

{ and mul_x, mul_x, 7fffh; } \ zero the sign bit of X, mul_x := abs(mul_x)

{
```

```
  and nil, mul_y, 8000h; \ test the sign bit
  load_rm_z;
}
{ cjp rm_z, no_y_abs; } \ jump if mul_y >= 0

\ mul_y := -mul_y
{ xor mul_y, mul_y, 0ffffh; } \ invert
{ inc mul_y; }
no_y_abs

\ ========== main multiplication algorithm ==========
{ mov mul_cnt, 15; }
{ xor rhi, rhi; }
{ xor rlo, rlo; }
{ xor mul_xlo, mul_xlo; }

{ or srl, mul_x, mul_x, z; }
{ or sr.9, mul_xlo, mul_xlo, z; } \ mul_x.mul_xlo := 0.r(mul_x.mul_xlo)

mul_loop
{ or srl, mul_x, mul_x, z; }
{ or sr.9, mul_xlo, mul_xlo, z; } \ mul_x.mul_xlo := 0.r(mul_x.mul_xlo)

{
  and nil, mul_y, 4000h; \ test mul_y(14)
  load_rm_z;
}
{ cjp rm_z, no_add; } \ jump if mul_y(14) is 0

\{ inc r0; }  a debug counter
{
  add rlo, rlo, mul_xlo, z; \ rm_c.rlo := rlo + mul_xlo
  load_rm_c;
}
{ add rhi, rhi, mul_x, rm_c; } \ rhi := rhi + mul_x + rm_c

no_add
{ or sll, mul_y, mul_y, z; } \ mul_y := l(mul_y).0

{
  dec mul_cnt;
  cjp not zo, mul_loop; \ jump if mul_cnt = 0
}

\ ========== set the sign of the result ==========
{ cjp not rm_n, sign_end; } \ jump if result is positive
```

```
{ or rhi, rhi, 8000h; }
sign_end
{ cjp nz, com_done; }

\ ============================================================================
\ ========== branch unconditional
org addr_b
{
  mov pc, r10;
  cjp nz, mainloop;
}

\ ============================================================================
\ ========== branch if device ready
\ ========== condition: rhi & 0080h == 0080h
org addr_bdr
{ and nil, rhi, 0080h; load_rm_z; }
{ cjp not rm_z, jdr_do; }
{ cjp nz, com_done; }
jdr_do
{
  mov pc, r10;
  cjp nz, mainloop;
}

\ ============================================================================
\ ========== load from memory
org addr_load
{ bus_d_out r10; ewl; }
{ bus_d_outz;    ewh; }
{ read_mem rhi; }
{ cjp nz, com_done; }

\ ============================================================================
\ ========== store to memory
org addr_stor
{ bus_d_out r10; ewl; }
{ bus_d_outz;    ewh; }
{ write_mem rhi; }
{ cjp nz, com_done; }

\ ============================================================================
\ ========== input from device
org addr_in
{ bus_d_out r10; ewl; }
{ bus_d_outz;    ewh; }
```

```
{ read_dev rhi; }
{ cjp nz, com_done; }

\ ============================================================================
\ ========== output to device
org addr_out
{ bus_d_out r10; ewl; }
{ bus_d_outz;    ewh; }
{ write_dev rhi; }
{ cjp nz, com_done; }

\ ============================================================================
\ ========== move from rhi
\ ========== dest := rhi

org addr_mfhi
{
  mov rb, rhi;
  cjp nz, com_done;
}

\ ============================================================================
\ ========== move from rlo
\ ========== dest := rlo
org addr_mflo
{
  mov rb, rlo;
  cjp nz, com_done;
}

\ ============================================================================
\ ========== move to rhi
\ ========== rhi := src
org addr_mthi
{
  mov rhi, ra;
  cjp nz, com_done;
}

\ ============================================================================
\ ========== addition
\ ========== dest := dest + src
\ ========== rb   := rb   + ra
org addr_add
{
  add rb, ra, rb, z;
```

```
  cjp nz, com_done;
}

\ ============================================================================
\ ========== halt the cpu
\ ========== This should be the last command in the file for COMPLEX to halt.
org addr_halt
{}
```

5 Висновок

В ході виконання даної роботи були отримані практичні навички написання мікропрограм для ЕОМ, побудованої на базі комплекта КМ1804. На практиці була детально розглянута реалізація циклу читання-розпаковки-виконання команди. Були досліджені різні способи адресації, як з використанням РЗП, так і без них. Були розглянуті загальні принципи обміну даними з оперативною пам'яттю та зовнішніми пристроями.

Були розглянуті загальні принципи створення абстракцій для проектування систем команд.

Література

1 Жабін В. І., Жуков І. А., Клименко І. А., Стіренко С. Г. — Арифметичні та управляючі пристрої цифрових ЕОМ: Навчальний посібник. — К.: ВЕК+, 2008. — 176 с.

2 Жабін В. І., Ткаченко В. В. — Цифрові автомати. Практикум. — К.: ВЕК+, 2004. — 160 с., ил.

3 Sivarama P. Dandamudi. — Guide to RISC processors. — Springer, 2005. — ISBN 0-387-21017-2. — 387 p.