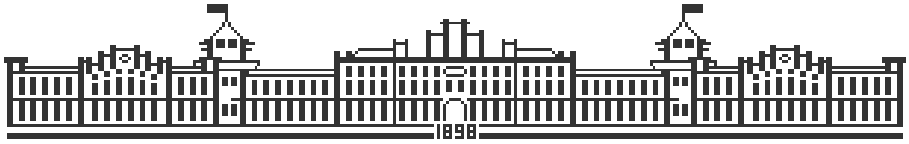


Міністерство освіти та науки України

Національний технічний університет України “Київський політехнічний інститут”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

# ЛАБОРАТОРНА РОБОТА № 3
3 дисципліни «Архітектура комп’ютерів-1»

На тему «ПЕРЕТВОРЕННЯ ДАНИХ В ЕОМ НА МІКРОПРОГРАМНОМУ РІВНІ»

Виконав:
студент 2 курсу ФІОТ
групи ІВ-71
Мазан Я. В.
Залікова – 7109

ПЕРЕВІРИВ:
доц. Верба О. А.

Київ – 2019

**Теоретичні відомості:**

Перетворення даних в ЕОМ може виконуватися на програмному, мікропрограмному та апаратному рівні.

На програмному рівні обробка інформації відбувається за допомогою програми – зв’язаного списку команд. Програма міститься в оперативній пам’ятті. Команди виконуються в певній послідовності, що визначається вихідними даними, проміжними результатами, ознаками тобто. Чергова адреса команди формується в лічильнику команд. Множина всіх команд, що може виконувати процесор, називається системою команд процесора.

Команда – інформаційне слово, що в загальному випадку визначає операцію, що виконується з операндами, місце розташування операндів і результату операції, а також вказує на адресу наступної команди.

Процесор послідовно зчитує команди з пам’ятті і виконує їх за допомогою мікропрограм і апаратних засобів. Як правило, в CISC-системах (CISC – Complex Instruction Set Computing) це відбувається на мікропрограмному рівні, а в RISC-системах (RISC – Reduced Instruction Set Computing) можлива апаратна інтерпретація реалізації команд.

Мікропрограма – це зв’язаний список мікрокоманд, виконання яких в певній послідовності забезпечує необхідне перетворення даних. Мікропрограми записані в памятті мікрокоманд (ПМК), що входить в склад блоку мікропрограмного управління (БМУ).

**Мета:**

Вивчити архітектуру ЕОМ, що містить блок мікропрограмного керування і арифметико-логічний пристрій із зосередженою логікою та двохадресним надоперативним запам’ятовуючим пристроєм (НОЗП), одержати навички розробки мікропрограм.

**Завдання:**

$7409_{10} = 1110011110001_2$

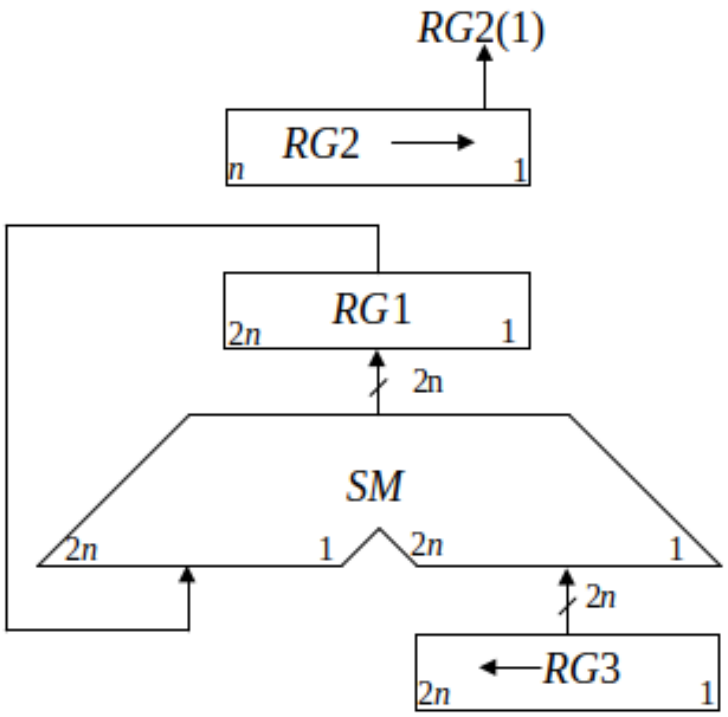
| $a_5$ | $a_4$ | Спосіб множення (при g = 1) |
|---|---|---|
| 1 | 0 | 2 |

| $a_3$ | $a_2$ | Форма подання чисел | | |
|---|---|---|---|---|
| | | X | Y | Z |
| 0 | 0 | ДК | ДК | ПК |

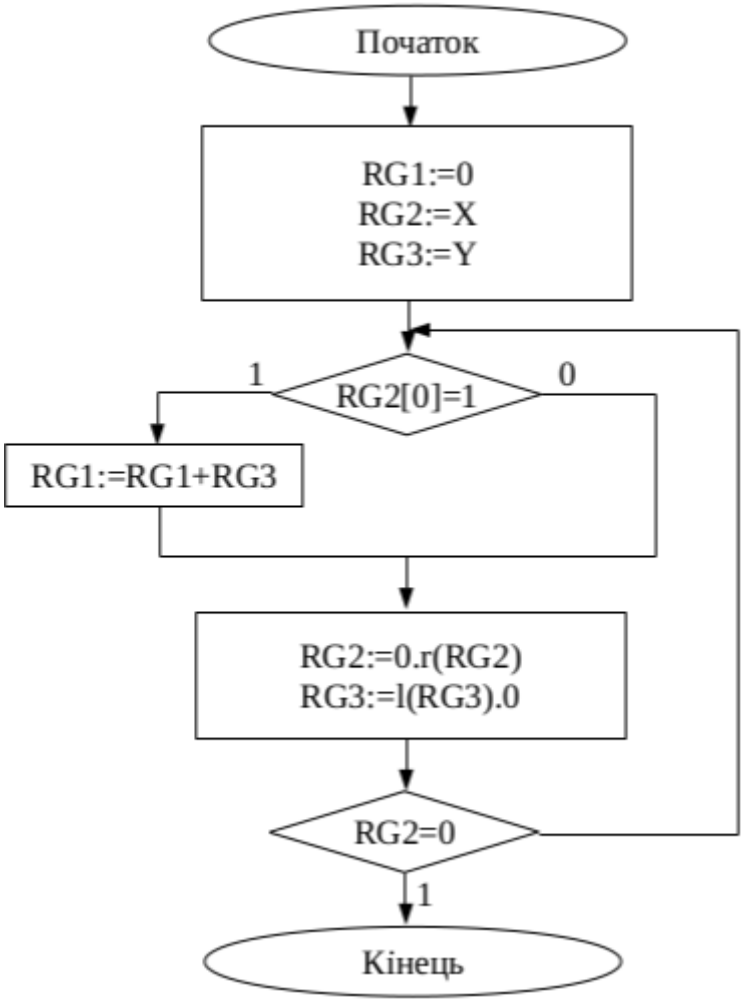
| $a_7$ | $a_1$ | Значення операндів | |
|---|---|---|---|
| | | X | Y |
| 1 | 1 | 7 | -6 |

| $a_6$ | $a_1$ | Арифметична операція (при g = 0) |
|---|---|---|
| 1 | 0 | $2^3X - 2Y$ |

**Операційна схема:**



**Структурний мікроалгоритм:**




**Код програми:**

```
link l1: rdm
link l2: rdd
link l3: ct
link ewh: 16

macro sh_r reg: {or srr, reg, reg, z;}
macro ah_l reg: {or sll, reg, reg, z;}
macro mov reg1, reg2: {or reg1, reg2;}

accept r1 : 0          \rg1 := 0
\accept r2 : 0111%      rg2 := X
\accept r3 : 1010%      rg3 := Y
accept r14 : 0h        \rg14 := G
dw    0h:0111%
dw    1h:1010%
equ    data:R0
```

```
accept data:0h

\Read data Y to R1-R2(auto set R1 with sign), X to R5(set marker bit)
LWX    {and nil, data, Z; oey; ewh;}     \ Load X to R5 from DATA(R0)
       {add nil, data, Z; oey; ewl;}     \ R0 save position in memory
       {R; or r3, BUS_D, Z; cjp rdm, CP;}                \ where data located
       {add data, data, 1H, Z;}              \ next data

LWY    {and nil, data, Z; oey; ewh;}     \ Load Y to R2 from DATA(R0+1)
       {add nil, data, Z; oey; ewl;}     \
       {R; or r2, BUS_D, Z; cjp rdm, CP;}                \
       {add data, data, 1H, Z;}              \ next data

{and r14, 0001h; load rm, flags;}             \cmp r14, 1
{cjp rm_z, fun;}                          \jz end
{cjp nz, mul;}          \else jmp mul

mul {}      \multiplication
\calculate sign of multiplication and make Y positive
sign {and r4, r3, 1000%;}          \Y sign
     {and r5, r2, 1000%;}          \X sign
     {xor r4, r4, r5;}             \save sign of multiplication into r4
     {xor r5, r5, r5;}             \and make null the additional r5 register
     {add sla, r4, r4, z;}         \ put the sign register into the 8th bit
     {add sla, r4, r4, z;}
     {add sla, r4, r4, z;}
     {add sla, r4, r4, z;}

     {and r2, r2, 0111%;}          \nullify signs of X and Y
     {and r3, r3, 0111%;}
     {sub r3, r3, z;}              \subtract one from Y
     {xor r3, r3, 0111%;}          \reverse bits of Y

cycle {add nil, r2, 0001h; load rm, flags;}     \cmp rg2[0], 1
compare1 {cjp rm_z, next;}                  \rg2[0] == 0 => jmp end
{add r1, r1, r3, z;}
\{cjp nz, next;}                          jmp end
next {add srl, r2, r2, z;}
     {add sll, r3, r3, z;}
compare2 {or nil, r2, 0000h; load rm, flags;}  \cmp rg2, 0
{cjp not rm_z, cycle;}                      \rg2 == 0 => jmp cycle
{cjp nz, sign2;}          \jmp sign2

fun {}
   {add sla, r2, r2, z;}
```

```
    {add sla, r2, r2, z;}
    {add sla, r2, r2, z;}
    {add sla, r3, r3, z;}
    \{and r4, r3, 0010h;}        extract sign of modified r3 and save into r4
    \{and r3, r3, 0ffefh;}       remove currect sign of r3
    \{add sll, r4, r4, z;}         put sign of
    \{add sll, r4, r4, z;}
    {add r1, r1, r2;}


    {add r1, r1, r3;}

    {cjp nz, sign2;}         \jmp sign2

sign2 {or nil, r4, 0000h; load rm, flags;}
      {cjp not rm_z, write;}             \ rg4 == 0 => jmp write

invert {sub r1, r1, z;}         \transform negative number into direct code
       {xor r1, r1, 11111111%;}

write {or r1, r1, r4;}
      {xor r3, r3, r3;}          \nullify our used registers
      {xor r4, r4, r4;}
      {cjp rdm, CP; W; or nil, Z, r3; oey;}

end {}
```

**Висновок:**

У даній роботі побудований алгоритм множення двох 16-розрядних операндів другим способом. Отримані результати моделювання співпадають з прогнозованими. В результаті виконання цієї роботи, я пригадав й закріпив теоретичні аспекти цієї теми.