НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Кафедра комп’ютерних систем, мереж та кібербезпеки

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт
з дисципліни
«Комп’ютерні системи». Частина 1

для студентів спеціальності 123 «Комп’ютерна інженерія»

всіх форм навчання

2021

Методичні вказівки до лабораторних робіт з дисципліни «Комп'ютерні системи». Частина 1 для студентів спеціальності 123 «Комп'ютерна інженерія» всіх форм навчання / Укл.: М.Д. Місюра – Київ: НУБіП, 2021. – 54 с.

Укладачі: М.Д. Місюра, к.т.н., доцент каф. комп'ютерних систем, мереж та кібербезпеки НУБіП України

Рецензенти: Б.Л. Голуб, к.т.н., завідувач каф. комп'ютерних наук НУБіП України
В.В. Шкарупило, к.т.н., доцент доцент каф. комп'ютерних систем, мереж та кібербезпеки НУБіП України

Відповідальний за випуск: М.Д. Місюра, к.т.н., доцент каф. комп'ютерних систем, мереж та кібербезпеки НУБіП України

Затверджено:
Рекомендовано до видання:
НМК факультету ІТ
Протокол №__ від ___._ __.2021 р.

Зміст

ВСТУП

Матеріал, пов'язаний з вивченням різних навичок щодо побудови складових комп'ютерних систем, складає значну частину курсу «Комп'ютерні системи». Дисципліна «Комп'ютерні системи» є одною з базових в системі знань та вмінь, що формують бакалавра та спеціаліста із спеціальності «Комп'ютерна інженерія». Мета викладення дисципліни «Комп'ютерні системи» – в сформувати в студентів знання, уміння і навички, необхідні для подальшого використання сучасних апаратних і програмних засобів комп'ютерних систем та їх компонентів.

Зважаючи на важливість і обширність матеріалу, пов'язаного з питаннями методів аналізу і синтезу, принципами організації і функціонування, викладання дисципліни «Комп'ютерні системи – Частина 1», займає провідну позицію.

У даних методичних вказівках, у відповідності до навчального плану, містяться лабораторні роботи, які у певній мірі дозволяють студентам оволодіти методами аналізу і синтезу функціонування компонентів (складових частин) комп'ютерних систем.

Ціль методичних вказівок – з метою сприяння вирішенню завдань підготовки спеціалістів у галузі комп'ютерних технологій, допомогти студентам оволодіти практичними навиками роботи з різними підходами, методиками та програмними середовищами.

Спробою досягти цієї цілі й визначається структура даних методичних вказівок. Частина лабораторних робот виконується у відповідності до індивідуальних варіантів, а частина робот виконується студентом у відповідності до завдання, яке міститься у розділі – хід роботи.

Після виконання лабораторної роботи кожен студент повинен оформити звіт на аркуші формату А4 якій містить наступні пункти:

- *Назву лабораторної роботи*
- *мета роботи;*
- *варіант завдання;*
- *Розв'язок чи послідовність її виконання згідно рекомендацій;*
- *результати (допускається подавати результати як PrintScreen екранів відповідних програмних середовищ, за допомогою яких виконується лабораторна робота);*
- *висновок по роботі.*

При захисті роботи студент відповідає на питання, які наведені після лабораторні роботи (питання для самоперевірки), демонструє послідовність виконання, результати лабораторної роботи.

Критерії оцінювання робот. Оцінюється відсоток виконаних робіт.

A –	"5"	— 4,5 – 5,0 (10) — 90% – 100%
B –	"4"	— 4,0 – 4,4 (9) — 82% – 89%
C –	"4"	— 3,5 – 3,9 (8) — 74% – 81%
D –	"3"	— 3,0 – 3,4 (7) — 64% – 73%
E –	"3"	— 2,5 – 2,9 (6) — 60% – 63%
FX –	"2"	— 2,0 – 2,4 (4-5) — 35% – 59%
F –	"2"	— 1,5 – 1,9 (0-3) — 0% – 34%

Методичні вказівки призначені для виконання лабораторних робіт студентами всіх форм навчання 123 «Комп'ютерна інженерія».

Дані методичні вказівки охоплюють всі модулі, що відповідають робочій програмі «Комп'ютерні системи». Частина 1.

Лабораторна робота №1. Оцінка трудомісткості алгоритму

Мета роботи: Засвоєння засобів аналізу трудомісткості обчислювальних алгоритмів.

Теоретичні відомості

Задача, що підлягає рішенню на ЕОМ, може бути охарактеризована кількістю даних, складністю алгоритму, трудомісткістю алгоритму. Під трудомісткістю алгоритму розуміється кількість інформації, необхідної для його реалізації. Оцінка складності алгоритму може бути дана в бітах, байтах, кількості символів певної мови. Для оцінки можуть використовуватися також кількість операторів конкретної алгоритмічної мови, кількість машинних кодів і т. д. Попередня оцінка складності алгоритму і кількості даних виконується експертним шляхом, а точні значення можуть бути відомі тільки після завершення програмування. Складність алгоритму і кількість даних характеризують потребу задачі в оперативній і зовнішній пам'яті. Під трудомісткістю алгоритму розуміється кількість обчислювальної роботи, необхідної для його реалізації. Трудомісткість характеризує витрати часу для реалізації алгоритму на деякій сукупності технічних засобів. Звичайно трудомісткість оцінюється кількістю процесорних операцій і операцій введення-виводу. Слідує відразу обмовитися, що трудомісткість алгоритму є в загальному випадку випадковою величиною і залежить від вхідних даних. Тому трудомісткість алгоритму може бути визначена тільки наближено в термінах теорії імовірності: математичними сподіваннями, дисперсією і т. д.

Трудомісткість алгоритму в першому наближенні може бути охарактеризована набором параметрів:

Θ - середня кількість процесорних операцій, необхідних для однієї реалізації алгоритму;

$N_1, N_2, \ldots, N_H$ – середня кількість звернень до файлів за один прогін програми через ЕОМ;

$L_1, L_2, \ldots, L_H$ – середня кількість інформації, що передається за одне звернення до файлів. $F_1, F_2, \ldots, F_H$.

При необхідності набір параметрів, що характеризують трудомісткість алгоритму може бути розширений. Вхідна інформація для розрахунку трудомісткості алгоритму може бути взята з схеми алгоритму, що використається звичайно для розробки програми. Приклад такий схеми алгоритму наведений на рис. 1.1.

Для розрахунку трудомісткості алгоритму необхідно знати імовірності переходів з логічних вершин при одиничному значенні логічної умови. Якщо відповідну імовірність визначити через *p*, тоді імовірність виходу з логічної вершини при нульовому логічному значенні умови, що перевіряється буде рівна 1 - *p*.

Для подальших розрахунків схему алгоритму раціонально зображати в вигляді графа алгоритму. Для цього пропонується перенумерувати всі оператори схеми алгоритму. У логічних операторів замість логічних умов '1' і '0' будемо записувати відповідну даному виходу імовірність. Імовірність виходу з операторної вершини рівна 1. Отриманий таким чином граф алгоритму зображений на рис. 1.2.

Граф алгоритму можна істотно спростити, якщо трудомісткість виконання логічних вершин незначна в порівнянні з трудомісткістю виконання операторних вершин. Тоді стани, що відповідають логічним вершинам, можна злити з станами, що випереджають відповідні операторні вершини. В нашому прикладі можна злити стан (1.2), (3.4), (7.8), (10.11).

Після цілком зрозумілої перенумерації отримаємо мінімізований граф алгоритму, зображений на рис. 1.3.

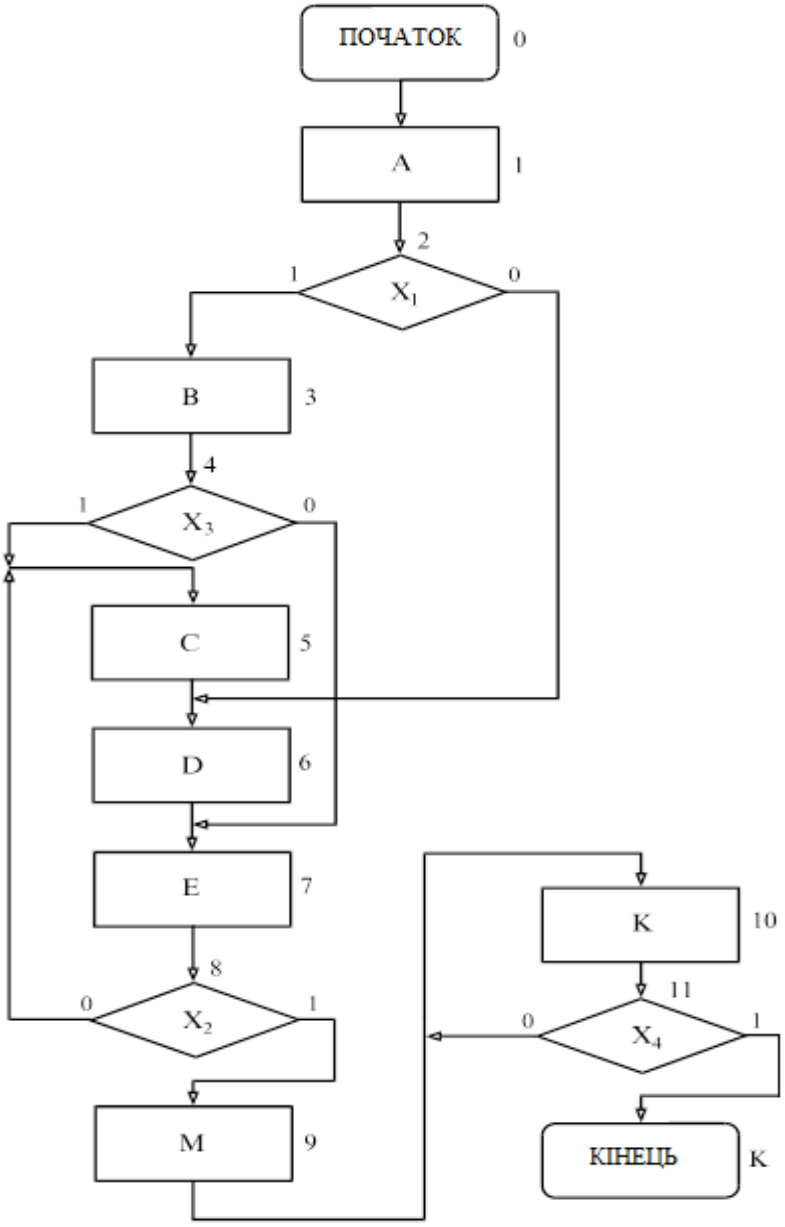


Рис. 1.1. Блок-схема алгоритму

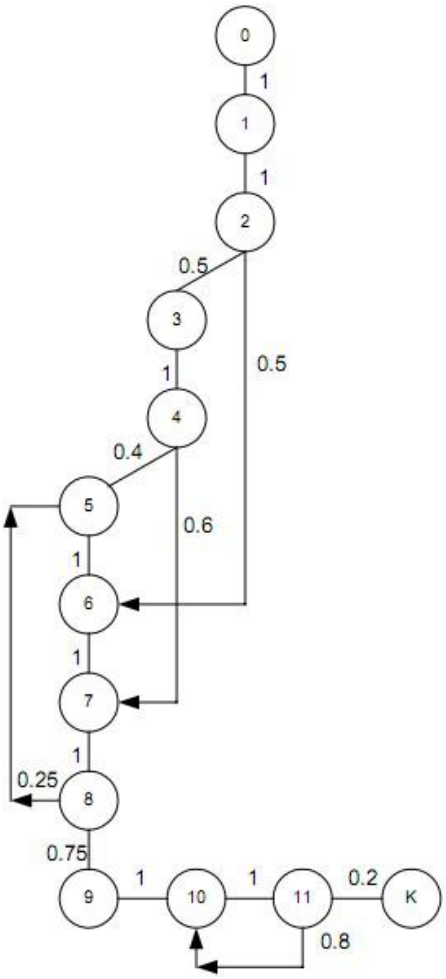


Рис. 1.2. Граф алгоритму

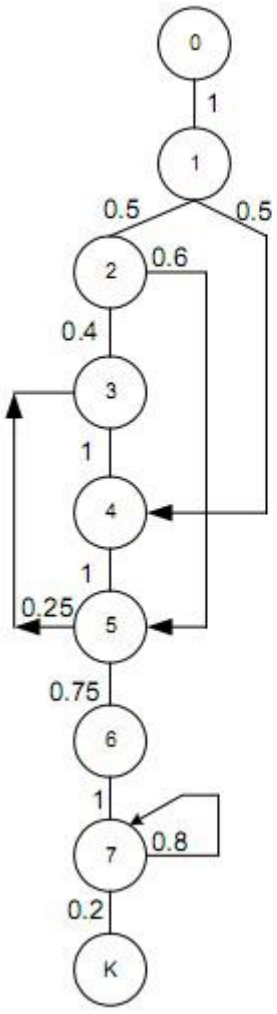


Рис. 1.3. Мінімізований граф алгоритму

При достатньому досвіді до нього можна було прийти відразу від блок-схеми алгоритму, наведеної на рис. 1.1. Позначимо через $n_1, n_2, \ldots, n_{k-1}$ середнє число звернень до операторів $V_1, V_2, \ldots, V_{k-1}$ відповідно. Кожний з операторів $V_i (i = 1,2,\ldots,k-1)$ будемо характеризувати наступним набором чисел:

k_i - середня кількість процесорних операцій, що виконуються в операторі V_i

m_{ij} - середня кількість запитів до файлу F_j(j=1,2,...,h) з оператора V_i

l_{ij} - середня кількість інформації, що передається при одному запиті до файлу F_j(j=1,2,...,H) з оператора V_i

Тоді трудомісткість алгоритму може бути оцінена по наступним формулам:

$$\Theta = \sum_{i=1}^{k-1} n_i k_i \tag{1.1}$$

$$N_j = \sum_{i=1}^{k-1} n_i m_{ij} \tag{1.2}$$

$$L_j = \frac{1}{N_j} \sum_{i=1}^{k-1} n_i l_{ij} \quad (j = 1,2,\ldots,H), \tag{1.3}$$

де Θ - середнє число процесорних операцій, що виконуються при одному виконанні алгоритму;

N_j - середнє число звернень до файлу $F_j(j=1,2,\ldots,H)$ при одному виконанні алгоритму;

l_{ij} - середня кількість інформації, що передається при кожному зверненні до файлу $F_j(j=1,2,\ldots,H)$ при одному виконанні програми.

Для визначення середнього числа звернень n_i до оператора V_i (i=1,2,...,k-1) звичайно робляться наступні припущення:

- імовірність виконання після оператора V_i оператора V_j рівна P_{ij} і є постійною величиною;
- імовірність P_{ij} залежить тільки від оператора V_i, але ніяк не зв'язана зі способом попадання в оператор V_j, т.е. не залежить від передісторії обчислювального процесу;
- $\sum_{i=1}^{k-1} P_{ij} = 1$.

При виконанні вищезазначених допущень обчислювальний процес стає Марківським процесом з станами S_1, S_2, ... ,S_k. При цьому оператори V_1, V_2, ... ,V_{k-1} відповідають станам S_1, S_2, ... ,S_{k-1}.

Стан Sk відповідає кінцевій вершині граф алгоритму. Стани

S1,S2,...,Sk-1 називаються незворотними, бо процес, неодмінно, з них виходить.

Сучасна теорія алгоритмів налічує декілька способів оцінки трудомісткості алгоритмів.

Оцінка трудомісткості алгоритмів засобами теорії марківських ланцюгів

Для визначення середнього числа процесорних операцій, що виконуються за один прогін програми необхідно подати граф алгоритму у вигляді стохастичної матриці P. Для рис.1.3 вона прийме вигляд:

	S_1	**S_2**	**S_3**	**S_4**	**S_5**	**S_6**	**S_7**	**S_k**
S_0	1	0	0	0	0	0	0	0
S_1	0	0.5	0	0.5	0	0	0	0
S_2	0	0	0.4	0	0.6	0	0	0
S_3	0	0	0	1	0	0	0	0
S_4	0	0	0	0	1	0	0	0
S_5	0	0	0.25	0	0	0.75	0	0
S_6	0	0	0	0	0	0	1	0
S_7	0	0	0	0	0	0	0.8	0.2

Елементами матриці **P** є імовірності переходу з стану **i** в стан **j**, що наведені на графі алгоритму. З стохастичної матриці слідує, наприклад, що в стані S_4 процес може виявитися при переході з S_1 з імовірністю 0.5 або при переході з стану S_3 з імовірністю 1.

Якщо відомі середні числа звернень до вершин V_1 й V_3 , то число звернень до вершини V_4 буде відповідно рівним: $n_4 = p_{14} n_1 + p_{34} n_3 = 0.9 n_1 + n_3$.

В самому загальному випадку можна записати (складання по стовпчику стохастичної матриці):

$$n_i = \sum_{j=0}^{k-1} p_{ji} n_j \quad , (n_0 = 1,\ i = 1,2,\ldots,k-1) \qquad (1.4)$$

Останній запис являє собою систему лінійних алгебраїчних рівнянь, розв'язок яких дає це середнє число звернень до операторів. Для прикладу з стохастичної матриці маємо:

$n_1 = 1 n_0 = 1*1 = 1$;
$n_2 = 0.5 n_1 = 0.5*1 = 0.5$;
$n_3 = 0.4 n_2 + 0.25 n_5 = 0.2 + 0.25 n_5$;
$n_4 = 0.5 n_1 + n_3 = 0.5 + n_3$;

$n_5 = 0.6n_2 + n_4 = 0.3 + n_4$;
$n_6 = 0.75n_5$;
$n_7 = n_6 + 0.8n_7$;

Вирішуючи систему рівнянь, отримаємо:

$n_1 = 1;\ n_2 = 0.5;\ n_3 = 0.533;\ n_4 = 1.033;$
$n_5 = 1.333;\ n_6 = 1;\ n_7 = 5.$

Для перевірки правильності рішення системи лінійних рівнянь можна використати очевидну тотожність:

$$n_k = \sum_{j=0}^{k-1} p_{jk} n_j = 1 \quad \text{, при } n_0 = 1. \tag{1.5}$$

В наведеному прикладі це виконується так: $n_k = 0.2 * n_7 = 0.2 * 5 = 1$.

Даний засіб є універсальним (у випадку Марківського процесу), але вимагає більших витрат часу при значних розмірах стохастичної матриці.

Мережевий підхід до оцінки трудомісткості алгоритмів

Мережевий підхід зручний для аналізу трудомісткості алгоритму вручну, та дозволяє обчислювати середню, мінімальну і максимальну трудомісткість алгоритму на графах, не що містять цикли. Для прикладу розглянемо граф алгоритму, зображений на рис.1. 4.

Використовуючи формулу (1.4) для рис. 1.4 можемо записати:

n0 = 1 ;
п1 = 1 * n0 = 1 ;
n2 = 0.5 * п1 = 0.5 ;
n4’ = 0.5 * n1 = 0.5 ;
n5’ = n4’ + 0.6n2 = 0.5 + 0.6 * 0.2 = 0.8 ;
n3’ = 0.4n2 + 0.25n5’ = 0.4 * 0.5 + 0.25 * 0.8 =0.4;
n6 =n3’ + 0.75n5’ = 0.4 + 0.75 * 0.8 = 1 ;
n7’ = 1 * n6 =1.

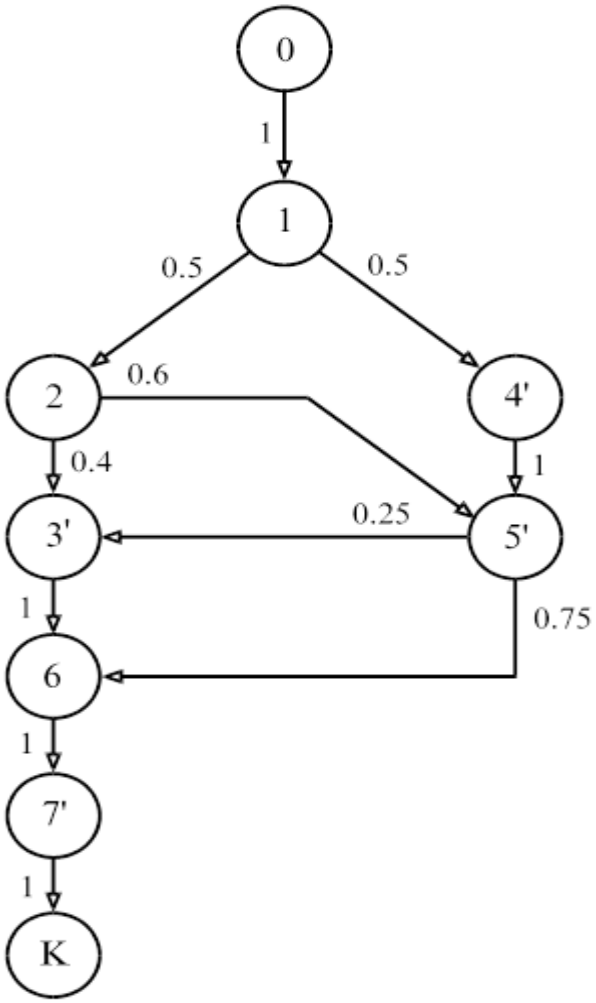


Рис. 1.4. Граф алгоритму без циклів

В зв'язку з відсутністю циклів систему легко вирішувати. Витрати часу на аналіз можуть бути знижені додатково, якщо ввести ефективну нумерацію станів графа алгоритму. Остання полягає в тому, що номер вершини повинен бути таким, щоб дуги що входять в цю вершину починалися в вершинах з меншими номерами. Якщо після такої нумерації рівняння записувати в порядку зростання їхніх номерів, тоді вони будуть відразу вирішуватися, бо невідомі з меншими номерами будуть вже обчислені. В наведеному прикладі нумерація вершин не відповідає означеній вимозі, але рівняння записувалися в порядку їхнього розв'язку. За наявності в графі алгоритму циклів останні слідує замінити операторами з еквівалентної трудомісткістю. Якщо в графі алгоритму є цикли в циклі, тоді, очевидно, перш слідує замінити еквівалентними операторами внутрішні цикли, а після цього переходити до заміни зовнішніх циклів. Розрахунок трудомісткості алгоритму виконується по наведеній вище методиці.

Визначимо через p_c імовірність замикання циклу, т. е. імовірність переходу по дузі з кінця циклу в його початок. Тоді в відповідності з вираженням (1.4) можна записати:

$$n_c = 1 + p_c n_c , \tag{1.6}$$

де n_c - середнє число повторень циклу.

З виразу (1.6) маємо:

$$n_c = \frac{1}{1 - p_c} \tag{1.7}$$

Тоді середнє число процесорних операцій циклу буде рівно: $k_c = n_c k_{тс}$,

де $k_{тс}$ - середня трудомісткість тіла циклу; k_c - середня трудомісткість тіла циклу.

Середнє число звернень до файлів і середня кількість інформації, що передається при зверненні в циклі до файлів, визначається аналогічно.

Приклад 1.

В алгоритмі на рис. 1.3 є два циклу. Перший містить оператори V_3,V_4,V_5, а другий складається тільки з оператора V_7. Перший цикл ускладнюєся входами в середину циклу з операторів V_1 й V_2 .

Для позбавлення від входів в цикл не через початок циклу V_3 обчислимо елементарні перетворення графа алгоритму. Еквівалентний граф після очевидних перетворень зображений на рис. 1.5.

З рис. 1.5 слідує, що оператори V_4 й V_5 , що використовувались для входу в тіло циклу не через його початок, винесені окремо під номерами 4’ й 5’. Кількість повторень циклів з виразу (1.7) дорівнює:

$$n_{c1} = \frac{1}{1 - 0.25} = 4;$$

$$n_{c2} = \frac{1}{1 - 0.8} = 5;$$

де n_{c1}, n_{c2} - число повторень першого і другого циклів.

Граф алгоритму без циклів наведений на рис. 1.4.

Наведений вище підхід дозволяє обчислити середню трудомісткість алгоритму. Для оцінки максимальної і мінімальної трудомісткості алгоритму необхідно перебрати всі можливі шляхи, ведучі з початкової вершини графа алгоритму в кінцеву, і вибрати з них шляхи, що дадуть максимальну і мінімальну трудомісткість. Слідує врахувати, що, наприклад, шлях з мінімальною кількістю процесорних операцій може мати не мінімально можливе число звернень до файлів. За наявності циклів в графі алгоритму для тіла циклу визначається мінімальна і максимальна трудомісткість. Знаходиться мінімальне і максимальне число повторень циклу і формуються відповідні еквівалентні по трудомісткості оператори.

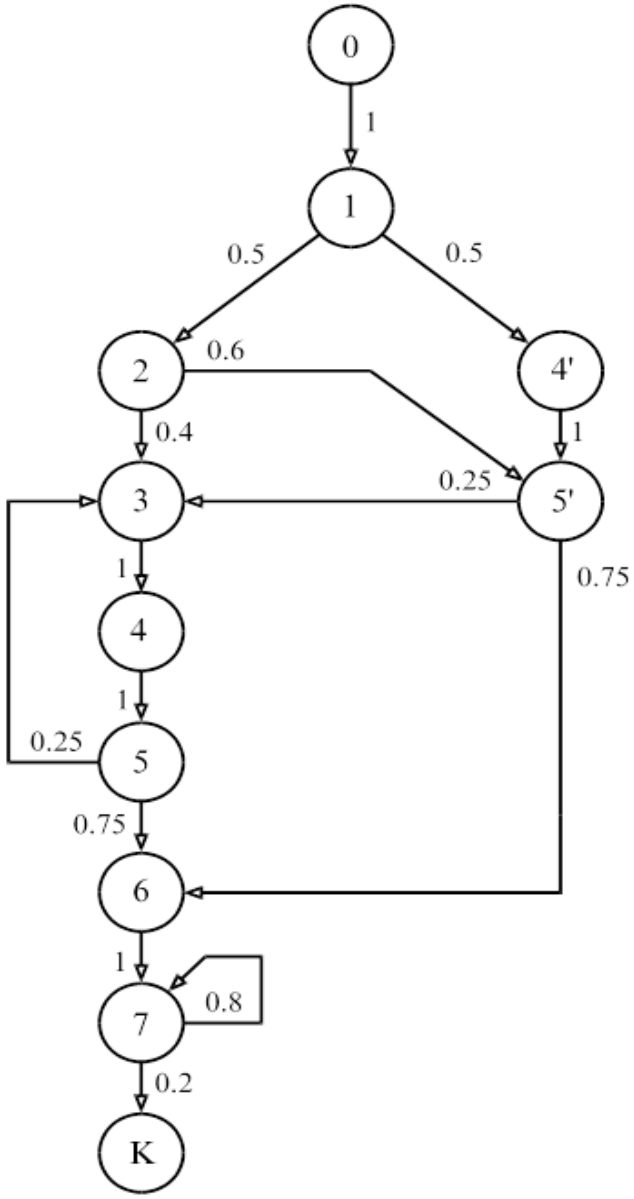


Рис. 1.5. Еквівалентний граф алгоритму

Приклад 2.

Оцінимо мінімальне і максимальне число процесорних операцій для графа алгоритму, зображеного на рис. 1.4. Для спрощення приймемо, що всі оператори мають трудомісткість, рівну 1000 процесорних операцій. Через A_i і B_i будемо значити відповідно мінімальне і максимальне число процесорних операцій, що буде мати місце в момент виходу процесу з і-й вершини графа.

$A_0 = 0;$
$A_1 = \min(A_0) + 1000 = 1000;$
$A_2 = \min(A_1) + 1000 = 2000;$
$A_{4'} = \min(A_1) + 1000 = 2000;$
$A_{5'} = \min(A_2, A_{4'}) + 1000 = 3000;$
$A_{3'} = \min(A_2, A_{5'}) + 1000 = 3000;$
$A_6 = \min(A_{3'}, A_{5'}) + 1000 = 4000;$
$A_7 = \min(A_6) + 1000 = 5000;$
$B_0 = 0;$
$B_1 = \max(B_0) + 1000 = 1000;$
$B_2 = \max(B_1) + 1000 = 2000;$
$B_{4'} = \max(B_1) + 1000 = 2000;$
$B_{5'} = \max(B_2, B_{4'}) + 1000 = 3000;$

$B_{3'} = \max(B_2, B_{5'}) + 1000 = 4000;$
$B_6 = \max(B_{3'}, B_{5'}) + 1000 = 5000;$
$B_7 = \max(B_6) + 1000 = 6000.$

Завдання для виконання

1. Перед виконанням роботи ознайомитись з теоретичними відомостями.
2. У відповідності до отриманого номеру варіанту вибрати логічну схему алгоритму (ЛСА) - таблиця 1. Згідно з ЛСА зобразити графічну схему алгоритму.
3. З таблиці 2 вибираються імовірності переходи при одиничних логічних умовах, відповідно до варіанту.
4. З таблиці 3 знаходяться кількість процесорних операцій в операторах алгоритму.
5. З таблиці 4 по останній цифрі варіанту вибираються кількість запитів і довжина запису в кілобайтах у разі звертання до файлів F_1, F_2, F_3.
6. Зображається блок-схема алгоритму, граф алгоритму і мінімальний граф алгоритму.
7. Визначається трудомісткість алгоритму засобами теорії марківських ланцюгів.
8. Визначається середня трудомісткість алгоритму з допомогою мережевого підходу.
9. Обчислюється мінімальна і максимальна трудомісткість алгоритму.

Контрольні питання

1. Що таке трудомісткість алгоритму?
2. Як визначається трудомісткість алгоритму і з якою метою обчислюється ця величина?
3. Чому трудомісткість алгоритму є, як правило, випадковою величиною?
4. Які параметри можуть бути використані для характеристики трудомісткості алгоритму?
5. Як виконати ефективну нумерацію станів в графі алгоритму для мереженого аналізу?
6. Що таке стохастична матриця?
7. Як визначити імовірності виходу з логічної вершини?
8. Що таке марківський процес?
9. Доведіть правильність формули (1.4).
10. Доведіть правильність тотожності (1.5).
11. Доведіть правильність виразу (1.6).
12. Чи існує різниця в оцінці середньої трудомісткості при використанні теорії марківських ланцюгів та мережевого підходу? Якщо так, то пояснити її.

Таблиця 1. Варіанти схем алгоритмів

Варіант	Логічна схема алгоритму
1	Поч. $Ax_1\uparrow^1Bx_2\uparrow^2C\downarrow^3x_3\uparrow^3D\downarrow^2E\downarrow^4Mx_4\uparrow^4K\downarrow^1$ Кін.
2	Поч. $Ax_2\uparrow^2Bx_3\uparrow^3x_1\uparrow^1D\downarrow^3E\downarrow^1\downarrow^3M\downarrow^2x_3\uparrow^3K$ Кін.
3	Поч. $Ax_2\uparrow^2B\downarrow^3\downarrow^2Cx_3\uparrow^3\downarrow^1Dx_1\uparrow^1Ex_4\uparrow^4MK\downarrow^4$ Кін.
4	Поч. $A\downarrow^2\downarrow^1Bx_1\uparrow^1Cx_2\uparrow^2Dx_4\uparrow^4\downarrow^5Ex_3\uparrow^3M\downarrow^4K$ Кін.
5	Поч. $\downarrow^1Ax_1\uparrow^1Bx_2\uparrow^2Ex_3\uparrow^3C\downarrow^2\downarrow^3Mx_4\uparrow^4D\downarrow^4K$ Кін.
6	Поч. $\downarrow^4Bx_2\uparrow^2Cx_3\uparrow^3D\downarrow^2\downarrow^3Ex_4\uparrow^4Mx_1\uparrow^1K\downarrow^1$ Кін.
7	Поч. $Ax_2\uparrow^2Cx_4\uparrow^4Dx_3\uparrow^3E\downarrow^2\downarrow^3\downarrow^4Kx_4\uparrow^4MB$ Кін.
8	Поч. $Dx_1\uparrow^1Ex_2\uparrow^2M\downarrow^2\downarrow^3Ax_2\uparrow^2Mx_3\uparrow^3K\downarrow^1D$ Кін.
9	Поч. $x_1\uparrow^1A\downarrow^1\downarrow^2Bx_2\uparrow^2C\downarrow^4Dx_3\uparrow^3K\downarrow^3Mx_4\uparrow^4$ Кін.

Варіант	Логічна схема алгоритму
10	Поч. $x_1\uparrow^1 A\downarrow^4\downarrow^2 Bx_2\uparrow^2 Cx_3\uparrow^3 E\downarrow^3 Dx_4\uparrow^4 K\downarrow^1 M$ Кін.
11	Поч. $\downarrow^3 Ax_1\uparrow^1 Bx_2\uparrow^2 C\downarrow^2 D\downarrow^1 Ex_3\uparrow^3 Kx_4\uparrow^4 M\downarrow^4$ Кін.
12	Поч. $x_4\uparrow^4 Ax_1\uparrow^1 B\downarrow^2 Cx_2\uparrow^2 D\downarrow^1 Ex_3\uparrow^3 K\downarrow^4 M\downarrow^3 x_1\uparrow^1$ Кін.
13	Поч. $Ax_1\uparrow^1 Bx_2\uparrow^2 C\downarrow^2\downarrow^1 D\downarrow^3\downarrow^4 EKx_4\uparrow^4 Mx_3\uparrow^3$ Кін.
14	Поч. $\downarrow^5 Ax_1\uparrow^1 Bx_2\uparrow^2 C\downarrow^2 D\downarrow^3 Ex_3\uparrow^3 C\downarrow^1 DEx_3\uparrow^3 Kx_4\uparrow^4 M\downarrow^4$ Кін.
15	Поч. $\downarrow^2 Ax_1\uparrow^1 B\downarrow^1 Cx_2\uparrow^2 DEx_3\uparrow^3 K\downarrow^3 Mx_2\uparrow^2$ Кін.
16	Поч. $\downarrow^1 Ax_1\uparrow^1 B\downarrow^4\downarrow^2 Cx_2\uparrow^2 Dx_4\uparrow^4 Ex_2\uparrow^2 Kx_3\uparrow^3 M\downarrow^3$ Кін.
17	Поч. $\downarrow^1 Ax_1\uparrow^1 Bx_3\uparrow^3 C\downarrow^3 Dx_4\uparrow^4 E\downarrow^2 Kx_2\uparrow^2 M\downarrow^4$ Кін.
18	Поч. $\downarrow^4 Ax_4\uparrow^4 Bx_1\uparrow^1 C\downarrow^1 Dx_3\uparrow^3 Ex_2\uparrow^2 K\downarrow^2 M\downarrow^3$ Кін.
19	Поч. $x_1\uparrow^1 A\downarrow^3 Bx_2\uparrow^2 Cx_4\uparrow^4 D\downarrow^2\downarrow^4 Ex_3\uparrow^3 KM\downarrow^4$ Кін.
20	Поч. $\downarrow^4 Ax_1\uparrow^1 B\downarrow^1 C\downarrow^2\downarrow^3 Dx_1\uparrow^1 Ex_2\uparrow^2 Kx_3\uparrow^3 Mx_4\uparrow^4$ Кін.
21	Поч. $x_1\uparrow^1 A\downarrow^3 Bx_2\uparrow^2 Cx_4\uparrow^4 D\downarrow^4 E\downarrow^2 K\downarrow^1 Mx_3\uparrow^3$ Кін.
22	Поч. $x_2\uparrow^2 A\downarrow^3 Bx_1\uparrow^1 C\downarrow^2 Dx_4\uparrow^4 E\downarrow^1\downarrow^4 KMx_3\uparrow^3$ Кін.
23	Поч. $x_1\uparrow^1 Ax_2\uparrow^2 B\downarrow^2\downarrow^1 Cx_4\uparrow^4 D\downarrow^3 Ex_3\uparrow^3 KM\downarrow^4$ Кін.
24	Поч. $x_1\uparrow^1 A\downarrow^2 B\downarrow^1 Cx_2\uparrow^2 D\downarrow^3 Ex_3\uparrow^3\downarrow^4 Mx_4\uparrow^4$ Кін.
25	Поч. $x_3\uparrow^3 Ax_2\uparrow^2\downarrow^1 Bx_1\uparrow^1 C\downarrow^2\downarrow^3 D\downarrow^4 Ex_4\uparrow^4 MK$ Кін.

Примітки до таблиці 1: В ЛСА символам Поч. та Кін. відповідають початковий і кінцевий оператори алгоритму. Символи A, B, C, D, E, K, M визначають функціональні оператори алгоритму. Символи x_1, x_2, x_3, x_4 визначають логічні умови. Якщо логічна умова рівна одиниці, тоді виконується наступний оператор в ЛСА. Якщо логічна умова рівно нулю, тоді здійснюється перехід по стрілці з відповідним індексом. У логічної умови стрілка направлена вверх. Наприклад, $\uparrow^1$. У місця переходу стрілка направлена вниз - $\downarrow^2$. Для схеми алгоритму, зображеної на рис.1.1 ЛСА має вигляд:

Поч. $Ax_1\uparrow^1 Bx_3\uparrow^3\downarrow^2 C\downarrow^1 D\downarrow^3 Ex_2\uparrow^2 M\downarrow^4 K\ x_4\uparrow^4$ Кін.

Таблиця 2. Імовірності переходу (по X=1)

Варіант	P_1	P_2	P_3	P_4
1	0.1	0.3	0.6	0.9
2	0.2	0.2	0.7	0.8
3	0.3	0.1	0.8	0.7
4	0.4	0.2	0.9	0.6
5	0.5	0.3	0.8	0.5

Варіант	P_1	P_2	P_3	P_4
6	0.6	0.4	0.7	0.4
7	0.7	0.5	0.6	0.3
8	0.8	0.6	0.5	0.2
9	0.9	0.7	0.4	0.1
10	0.8	0.8	0.3	0.2
11	0.7	0.9	0.2	0.3
12	0.6	0.8	0.1	0.4
13	0.5	0.7	0.2	0.5
14	0.4	0.6	0.3	0.4
15	0.3	0.5	0.4	0.3
16	0.2	0.4	0.5	0.2
17	0.1	0.3	0.6	0.1
18	0.2	0.2	0.7	0.2
19	0.3	0.1	0.8	0.3
20	0.4	0.2	0.9	0.4
21	0.5	0.3	0.8	0.5
22	0.6	0.4	0.7	0.6
23	0.7	0.5	0.6	0.7
24	0.8	0.6	0.5	0.8
25	0.9	0.7	0.4	0.9

Таблиця 3. Кількість процесорних операцій (в тисячах)

Варіант	A	B	C	D	E	M	K
1	1	2	3	4	5	6	7
2	8	9	8	7	6	5	4
3	3	2	1	2	3	4	5
4	6	7	8	9	8	7	6
5	5	4	3	2	1	2	3

Варіант	A	B	C	D	E	M	K
6	4	5	6	7	8	9	8
7	7	6	5	4	3	2	1
8	2	3	4	5	6	7	8
9	9	8	7	6	5	4	3
10	2	1	2	3	4	5	6
11	7	8	9	8	7	6	5
12	4	3	2	1	2	3	4
13	5	6	7	8	9	8	7
14	6	5	4	3	2	1	2
15	5	3	4	2	1	5	4
16	3	4	5	6	7	8	9
17	1	3	5	7	9	8	6
18	4	2	2	4	6	8	9
19	7	5	3	1	1	3	5
20	7	9	8	6	4	2	2
21	4	6	8	9	7	5	3
22	1	4	7	9	8	7	6
23	5	4	3	9	7	6	8
24	9	4	3	7	8	8	6
25	2	4	9	7	8	6	2

Таблиця 4. Кількість звернень до файлів та довжина запису

№ п/п	A		B			C			D		E			M			K	
	F_1	F_2	F_1	F_2	F_3	F_1	F_2	F_3	F_1	F_2	F_1	F_2	F_3	F_1	F_2	F_3	F_1	F_2
0	0/0	1/1	2/2	3/3	4/4	3/5	2/6	1/7	0/0	1/8	2/9	3/8	4/7	3/6	2/5	1/4	0/0	1/3
1	2/2	3/1	4/2	3/3	2/4	1/5	0/0	1/6	2/7	3/6	4/9	3/8	2/7	1/6	0/0	1/5	2/4	3/3

№	A		B			C			D		E			M			K	
п/п	F_1	F_2	F_1	F_2	F_3	F_1	F_2	F_3	F_1	F_2	F_1	F_2	F_3	F_1	F_2	F_3	F_1	F_2
2	4/2	0/0	1/1	2/2	3/3	4/4	3/5	2/6	1/7	0/0	1/8	2/9	3/8	4/7	3/6	2/5	1/4	0/0
3	1/3	2/2	3/1	4/2	3/3	2/4	1/5	0/0	1/6	2/7	3/8	4/9	3/8	2/7	1/6	0/0	1/5	2/4
4	3/3	4/2	3/1	2/2	1/3	0/0	1/4	2/5	3/6	4/7	3/8	2/9	1/1	0/0	S	2/3	3/4	4/5
5	1/6	0/0	1/7	2/8	3/9	4/8	3/7	2/6	1/5	0/0	2/4	3/3	4/2	3/1	2/2	1/3	0/0	1/4
6	2/5	1/6	0/0	1/7	2/8	3/9	4/8	3/7	2/6	1/5	0/0	1/4	2/3	3/2	4/1	3/2	2/3	1/4
7	0/0	2/5	3/6	1/7	2/8	3/9	4/8	3/7	2/6	1/5	0/0	1/4	2/3	3/2	4/1	3/2	2/3	1/4
8	4/5	3/6	2/7	1/8	0/0	1/9	2/8	3/7	4/6	3/5	2/4	1/3	0/0	1/2	2/1	3/2	4/3	3/4
9	2/5	1/6	0/0	1/7	2/8	3/9	4/8	0/0	1/7	2/6	3/5	4/4	3/3	2/2	1/1	0/0	1/2	2/3

Примітки до таблиці 4: Кількість звернень до файлів - чисельник; довжина запису в кілобайтах - знаменник

Лабораторна робота №2. Визначення швидкодії ЕОМ

Мета роботи: Засвоєння практичних засобів визначення часу виконання арифметичних і логічних операцій в процесорі..

Теоретичні відомості

Найважливіші характеристики ЕОМ:

- Продуктивність ЕОМ;
- Ємність ОЗП і ЗЗУ;
- Система команд;
- Програмне забезпечення;
- Надійність;
- Вартість.

Вищенаведені характеристики в загальному випадку зв'язані між собою складними функціональними залежностями, що визначаються прийнятими засобами організації, елементної базою, технологією виробництва і т. д. Характеристики ЕОМ є в більшості випадків векторними величинами, кожна компонента з яких визначає окремі властивості ЕОМ.

Продуктивність ЕОМ - характеристика обчислювальної потужності, визначає кількість обчислювальної роботи, що виконується ЕОМ в одиницю часу.

На практиці користуються різноманітними наборами чисел, що характеризують продуктивність ЕОМ.

Номінальна продуктивність - вектор $V=(V_1, V_2, ... , V_n)$, складатися з швидкодії приладів, що входять в ЕОМ. Наприклад, швидкодія процесора, оперативної пам'яті, каналів, накопичувачів на магнітному диску, принтерів і т. д.

Під швидкодією розуміється кількість операцій, що виконуються в одиницю часу. Поняття номінальної продуктивності зручно для виробників ЕОМ і характеризує потенційні можливості ЕОМ в сенсі швидкості обробки даних.

Наявність загальних ресурсів в ЕОМ не дозволяє повністю використати її номінальну продуктивність. Наприклад, наявність в структурі одного селекторного каналу не дозволяє читати (або записувати) інформацію з двох (або більшого числа) накопичувачів на магнітних дисках водночас. Тому інколи вводиться **поняття комплексної продуктивності** ЕОМ, що характеризує не тільки швидкодію приладів, але і структуру зв'язків між ними, можливість паралельної роботи частини приладів.

Складність структури сучасних ЕОМ призвела до необхідності створення і постійної експлуатації комплексу програм, здійснюючих управління обчислювальним процесом і що реалізують найбільш загальні алгоритми обробки інформації. Цей комплекс програм відомий під назвою “операційна система”. Структура технічних засобів ЕОМ і порядок проходження задачі користувача через ЕОМ, що визначається операційною системою, характеризує **системну продуктивність ЕОМ**.

Врахування параметрів задач користувача (трудомісткість алгоритму, кількість звернень до приладів введення-виведення, необхідна ємність ОЗП і т. д.) призводять до **поняття продуктивності на робітничому навантаженні**.

Остання характеризується числом задач користувача, що вирішуються в одиницю часу.

Продуктивність на робітничому навантаженні важлива для користувача, але малоактуальна для розробників і виробників універсальних ЕОМ.

Відомо, що найважливішим параметром продуктивності ЕОМ є її швидкодія, визначає можливу швидкість обробки інформації. Найчастіше під швидкодією ЕОМ розуміють швидкість або час виконання тільки процесорних операцій. Швидкість і час виконання операцій зв'язані між собою співвідношенням:

$$V = \frac{1}{\tau},$$

де V - швидкість виконання операцій, що вимірюється кількістю операцій за секунду;

τ- час виконання однієї операції.

Швидкодія ЕОМ складається з швидкодії процесора і часу звернення до ОЗП. В загальному випадку швидкодія ЕОМ неоднакова для різних процесорних операцій, що розрізняються між собою числом звернень до ОЗП або регістрів загального призначення, алгоритмами обробки, вхідними даними. Тому для характеристики швидкодії ЕОМ можна ввести **поняття номінальної швидкодії ЕОМ, що визначається вектором значень** $V_1, V_2,..., V_M$ або $\tau_1, \tau_2,..., \tau_M$, де М - число операцій, що виконуються ЕОМ, V_i (i=1.2,..., М) - середнє число операцій **і-го** типу, що виконуються за секунду,τ_i (i=1.2,..., М) - середній час виконання **і-**ї операції.

Порівняння різноманітних ЕОМ по номінальній швидкодії важко із-за великого числа операцій і їхнього різноманітного складу. Для користувача більш простою величиною є середня швидкодія ЕОМ, що визначається наступним чином:

$$V = \frac{1}{\sum_{i=1}^{M} p_i \tau_i},$$

де **pi** (i=1.2,.... М) - імовірність виконання **і-**ї операції.

Очевидно, що

$$\sum_{i=1}^{M} p_i = 1.$$

Тоді в знаменник визначає математичне очікування тривалості операції.

Але в свою чергу величина імовірності виконання операції **pi** визначається задачею, що вирішується. В нинішній час для порівняння різноманітних ЕОМ використають так звані суміші Гібсона, що задають імовірність використання операцій для різноманітних класів задач, що вирішуються.

Практичні значення V_i або τ_i (i=1.2,..., М) для конкретної ЕОМ і при відомому навантаженні можуть бути визначені декількома засобами.

Перший засіб. Використати знання алгоритму виконання **і-**ї операції в ЕОМ. На основі технічної документації складається граф-схема мікропрограми заданої операції і для конкретних вхідних даних визначається число тактових імпульсів. Період слідування тактових імпульсів вимірюється безпосередньо на ЕОМ (з допомогою осцилографа, частотоміру або інших приладів).

Другий засіб. В процесорі заблокувати нарощування лічильника адреси команд (якщо це передбачене конструкцією ЕОМ) і запустити ЕОМ на виконання заданої операції в автоматичному режимі. За допомогою осцилографа або частотоміру визначити період виконання операції. Слідує відзначити, що даний засіб може інколи дати невірний результат для тих команд, що застосовують значення одного або декількох операндів в ході свого виконання.

Третій засіб. Більшість сучасних ЕОМ мають таймери - прилади для виміру часових інтервалів з достатньо високою точністю. Ідея засобу полягає в наступному. Запускається на виконання деяка програма і оцінюється час її виконання по показанням таймеру. Запускається на виконання друга програма, що відрізняється від попередньою тільки наявністю операції, час виконання якої досліджується. Оцінюється час виконання відповідної операції. Очевидно, що різниця часів виконання першої і другій програм дозволяє визначити це значення.

Порядок виконання роботи

1. У відповідності з номером варіанту з таблиці 1 вибирається команда ЕОМ, час виконання якої необхідно визначити з допомогою третього засобу.

2. Розробляються схеми алгоритмів першої і другий програми, що відрізняються одна від іншою наявністю або відсутністю команди ,що досліджується. Програми повинні бути циклічними, щоб різниця часу їхнього виконання була відчутної і фіксувалася операційною системою. Програми слідує написати на мові Асемблер (або іншій, за погодженням з викладачем), передбачивши в них завдання різноманітних вхідних даних. При розробці програм на мовах високого рівня слідує врахувати, що компілятори можуть оптимізувати код програми.

3. Розроблені програми пропускаються через одну і ту же ЕОМ для трьох варіантів вхідних даних. Бажано, щоб вхідні дані були “легкими”, “середніми” і “важкими” для виконання заданої операції в ЕОМ. Наприклад, для операції складання двійкових чисел з плаваючою комою вхідні дані будуть “легкими”, якщо порядки чисел будуть однаковими, “важкими” - для чисел з максимально можливої відмінністю порядків, “середніми” - для проміжних випадків. Тут під важкістю виконання операції розуміють час її виконання для конкретних вхідних даних. Для підбору вхідних даних треба ознайомитися з алгоритмом виконання операції в ЕОМ. Бажано перевірити циклічність результатів.

4. Демонструється виконання розробленої програми та результати викладачу.

5. Оформлюється звіт про лабораторну роботу, де наводять блок-схему алгоритму, тексти програм, аналіз отриманих результатів. Приклад оформлення наведено у додатку.

Таблиця 1. Варіанти завдання

Варіант	Команда Ассемблеру	Операція МВР	Тип Операндів	Тип Результату
1	DIV	Додавання	Ц	Ц
2	MUL	Додавання	В	В
3	AND	Додавання	ВД	ВД
4	OR	Додавання	Д	Д
5	XOR	Віднімання	Ц	Ц
6	NOT	Віднімання	В	В
7	SUB	Віднімання	ВД	ВД
8	ADD	Віднімання	Д	Д
9	ADC	Множення	Ц	Ц
10	INC	Множення	В	В
11	DEC	Множення	ВД	ВД
12	CMP	Множення	Д	Д
13	MOV	Ділення	Ц	Ц
14	XCHG	Ділення	В	В
15	PUSH	Ділення	ВД	ВД
16	POP	Ділення	Д	Д
17	MOVSB	Пересилання	Ц	Ц
18	MOVSW	Пересилання	В	В
19	LODSB	Пересилання	ВД	ВД
20	LODSW	Пересилання	Д	Д
21	CMPSB	Пересилання	Ц	В
22	CMPSW	Пересилання	Ц	ВД
23	SCASB	Пересилання	В	Ц
24	SCASW	Пересилання	ВД	Ц
25	NEG	Логічна Операція		

Примітка до таблиці 1: Ц – цілі числа, В – дійсні числа одинарної точності, ВД – дійсні числа подвійної точності, Д – десяткові.

Контрольні питання

1. Дати визначення швидкодії.
2. Що таке продуктивність ЕОМ?
3. Що таке номінальна продуктивність?
4. Визначити швидкодію пристрою друку.
5. Визначити швидкодію дисплею.
6. Дати визначення комплексної продуктивності і навести приклади.
7. Навести приклади комплексних ресурсів в ЕОМ.
8. Дати визначення системної продуктивності ЕОМ і навести приклади.
9. Дати визначення продуктивності на робітничому навантаженні, навести приклади.
10. Які ЕОМ називаються проблемно - орієнтованими?
11. Які ЕОМ відносяться до класу спеціалізованих?
12. Що таке операційна система і її основне призначення?
13. Що така швидкодія ЕОМ?
14. Що таке суміші Гібсона і навіщо вони потрібні?
15. В яких випадках другий засіб визначення швидкодії надає помилковий результат? Навести приклади.
16. Запропонувати відмінні від викладених засоби оцінки швидкодії ЕОМ.
17. Що таке транслятор який оптимізує код мов високого рівня? Навести приклади оптимізації програм.
18. Чому рекомендується двохкроковий режим виконання програми?

Додаток

Приклад оформлення лабораторної роботи

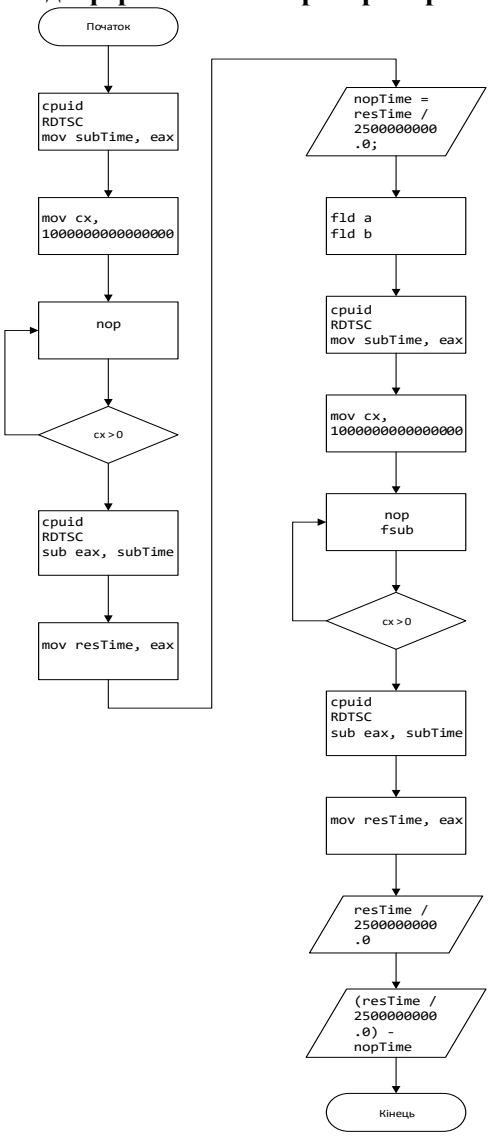


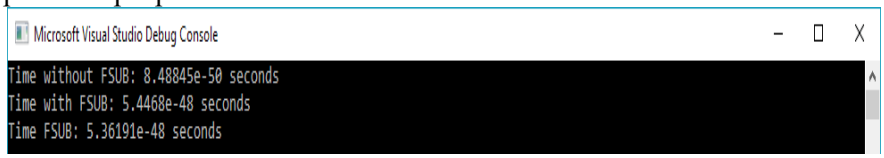
Рис. Д1. Блок схема алгоритму роботи програми

Код програми:

```
#include "pch.h"
#include <iostream>
#include <iomanip>
#include <math.h>
#include <float.h>

int main()
{
	long float nopTime;
	float resTime, subTime;
	double a = 3.14121212, b = 2.52123123;

	__asm {
		cpuid
		RDTSC
		mov subTime, eax

		mov cx, 1000000000000000
```

```
            loopstart:
                nop
                loop loopstart

            cpuid
            RDTSC
            sub eax, subTime
            mov resTime, eax
        }

        nopTime = resTime / 2500000000.0;
        std::cout <<  "Time without FSUB: " << nopTime << " seconds\n";

        __asm {
            fld a
            fld b

            cpuid
            RDTSC
            mov subTime, eax

            mov cx, 1000000000000000
            loopstart1:
                nop
                fsub
                loop loopstart1

            cpuid
            RDTSC
            sub eax, subTime
            mov resTime, eax
        }

        std::cout << "Time with FSUB: " << resTime / 2500000000.0 << " seconds\n";
        std::cout << "Time FSUB: " << (long float)(resTime / 2500000000.0) -
                                                    (long float)nopTime << " seconds\n";
}
```

Результати роботи програми:



Лабораторна робота №3. Ознайомлення з основними можливостями інтерфейсу та реалізація розрахункових задач в середовищі LabVIEW

Мета роботи: Ознайомлення студентів з основними можлівостями середовища LabView. Набуття навичок розробки програм для розв'язку простих розрахункових задач та налагодження зовнішнього інтерфейсу розробленої програми.

Хід виконання роботи

1. Створення програми для відображення результатів простих арифметичних операцій.

Для створення нової програми в середовищі LabVIEW, після запуску середовища, необхідно вибрати пункт Blank VI (рис. 3.1,а), після чого запуститься вікно "Front Panel" (рис. 3.1,б). По замовчанню нова програма має назву "Untiteled".

Рис. 3.1,а – Стартове вікно LabView

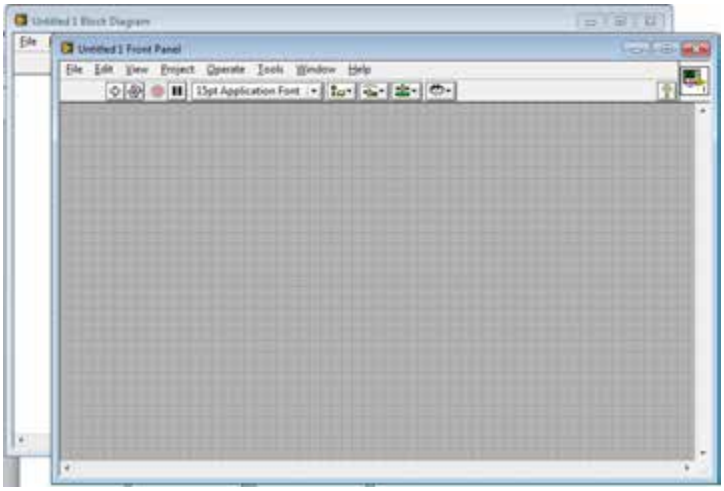


Рис. 3.1,б – Вікно "Front Panel" нової програми

У вікні Fron Panel оберемо елемент керування "Numeric Control" (рис. 3.2,а). Змінюємо його назву на "Chislo 1". Для цього необхідно активувати текстове поле над елементом керування подвійним кліком миші. Аналогічним чином створюємо елемент керування "Chislo 2". Для відображення результату розташуємо на інтерфейсній панелі "Numeric Indicator" (рис. 3.2,б). Змінюємо його назву на "Rezult".

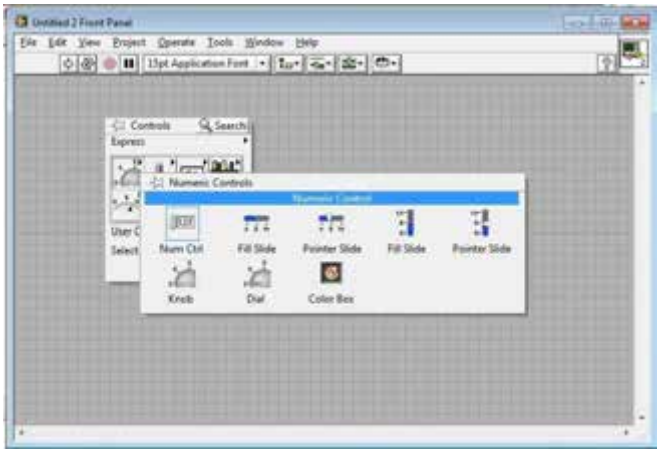


Рис. 3.2, а – Вибір елементу керування "Numeric Control"

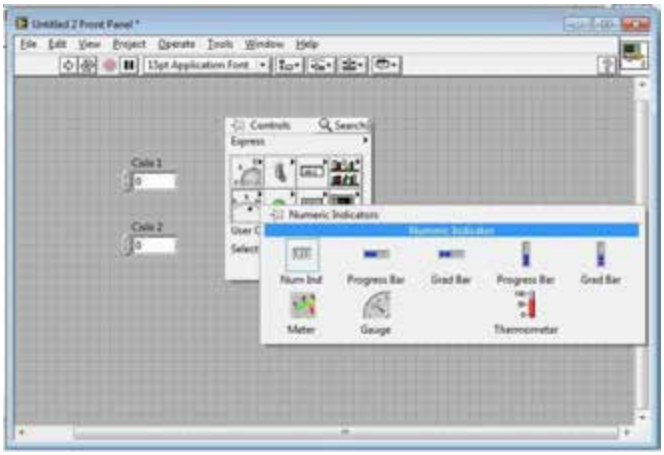


Рис. 3.2, б – Вибір елементу індикації "Numeric Indicator"

Розробивши візуальний інтерфейс з двома полями вводу чисел та одним числовим індикатором, реалізуємо задачу множення даних, що вводяться користувачем.

Для цього переходимо у вікно побудови діаграм (Block Diagram), в якому ми побачимо три іконки, що відповідають двом полям вводу числових значень та індикатору. Для множення чисел необхідно визвати функціональну панель правою кнопкою миші та вибрати функцію "Multiply" (рис. 3.3,а), після чого з'єднати необхідні вхідні та вихідні контакти елементів керування та індикаторів з'єднувальною котушкою "Wiring Tool" (рис. 3.3,б)

Рис. 3.3, а – Вибір функції "Multiply"

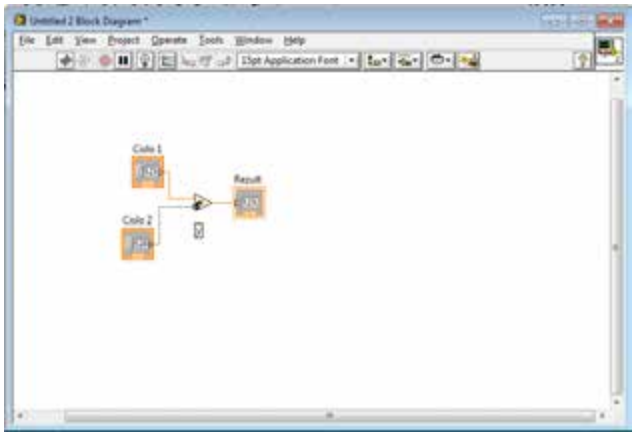


Рис. 3.3, б – Створення зв'язків між елементами програми за допомогою "Wiring Tool"

Програма завершена. Для її запуску переходимо на інтерфейсну панель, активуємо виконання програми в циклічному режимі 1 та запускаємо програму 2 (рис. 3.4). Для зупинки роботи програми натискаємо кнопку "Abort Execution" 3.

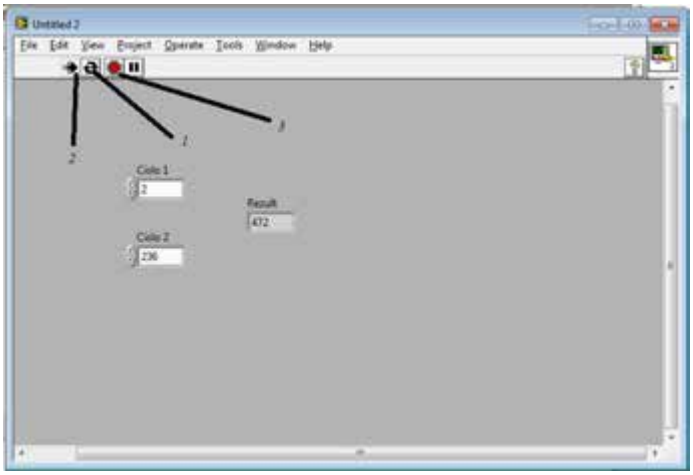


Рис. 3.4. – Результат роботи розробленої програми.

Середовище LabView забезпечує широкі можливості по налаштуванню та зміні зовнішнішнього інтерфейсу розроблених програм. Для прикладу, змінимо зовнішній інтерфейс графічних об’єктів керування та індикаціїї в розробленій програмі. Для цього натискаємо правою кнопкою миші на об’єкті, що буде змінюватись та в контекстному меню, що з’явилося, обираємо опцію заміни “Replace”. В нашому випадку оберемо ручку керування типу “Dial” (рис. 3.5, а), піднімемо назву даного елементу керування та змінимо його розмір (рис. 3.5, б).

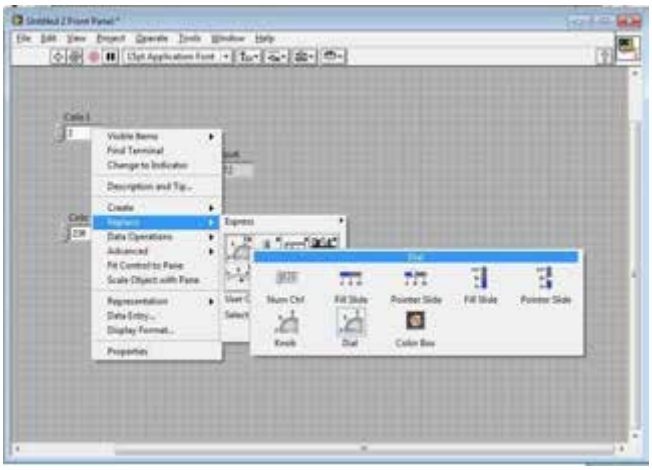


Рис. 3.5, а - Зміна зовнішнього вигляду елемента керування в програмі

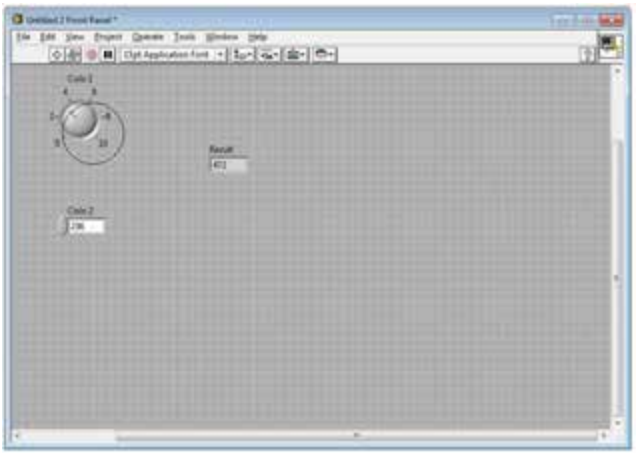


Рис. 3.5, б – Зміна розмірів елементу керування

Аналогічним чином можна реалізовувати зміну зовнішнього інтерфейсу будь-якого зі вставлених нами індикаторів та елементів керування. В результаті проведених змін отримаємо, наприклад, зовнішній інтерфейс прогрми, що приведено на рис. 3.6, а. Для налаштування регуляторів та індикаторів (наприклад, регулювання діапазону данних, що можна вводити або відображати) натискаємо праву кнопку миші та обираємо вкладку “Properties” (рис. 3.6, б).

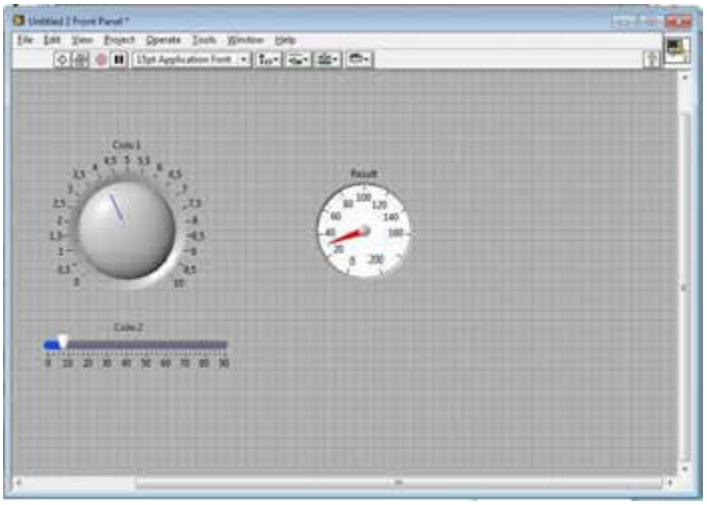


Рис. 3.6, а - Змінений зовнішній інтерфейс програми

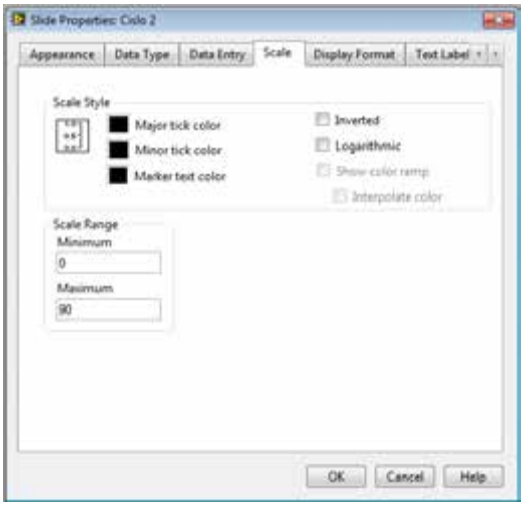


Рис. 3.6, б – Вкладка властивостей (Properties) елементів керування та індикації

Для збереження отриманої програми на жорсткому диску комп'ютера в панелі керування обираємо пункт "Save". Файл буде збережено в обраній вами папці з розширенням .vi.

2. Використання циклів та відображення результатів обчислень в графічному вигляді.

Наступним кроком буде введення в розроблену програму циклу, що буде аналогом оператора "While". Умовою завершення виконання циклу буде натиснення кнопки зупинки.

Для реалізації поставленої задачі обираємо цикл "While-Loop" та розтягуємо його у вікні блок-діаграми (рис. 3.7). Базову структуру програми перемістимо у цикл. Для цього, затиснувши ліву кнопку миші, виділяємо розроблену блок схему, а потім переміщуємо виділені елементи в цикл (рис. 3.8).

Для виконання розробленої програми необхідним елементом є умова виходу з циклу. Для реалізаціїї цієї задачі правою кнопкою миші активуємо вхідний канал іконки виходу з циклу та обираємо пункт "Create Control". В результаті, в нас з'явиться елемент керування "Stop" як в блок-діаграмі, так і на інтерфейсній панелі.

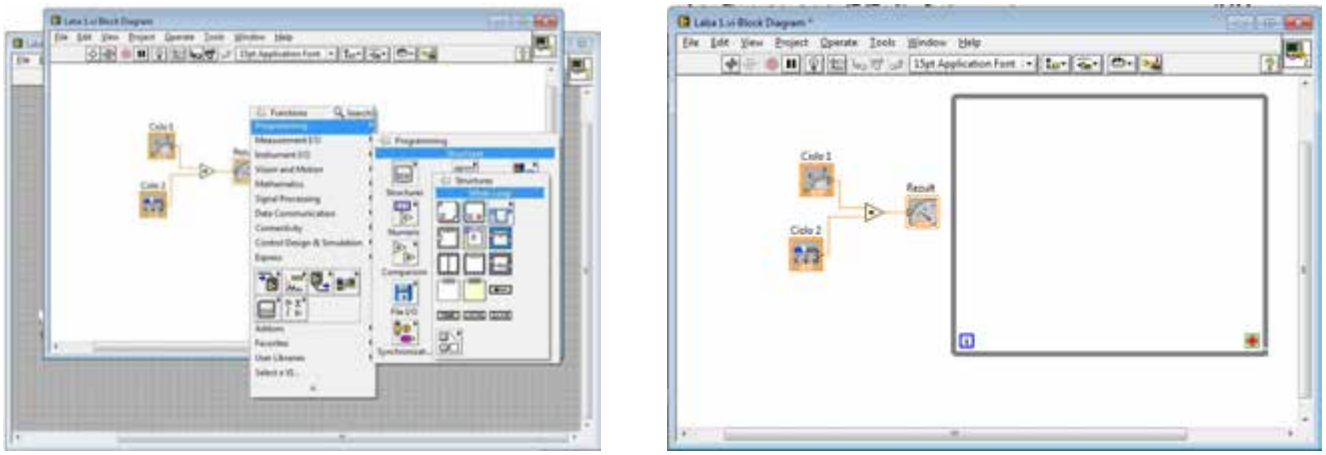


Рис. 3.7. – Вибір та розміщення блоку циклу "While-Loop"

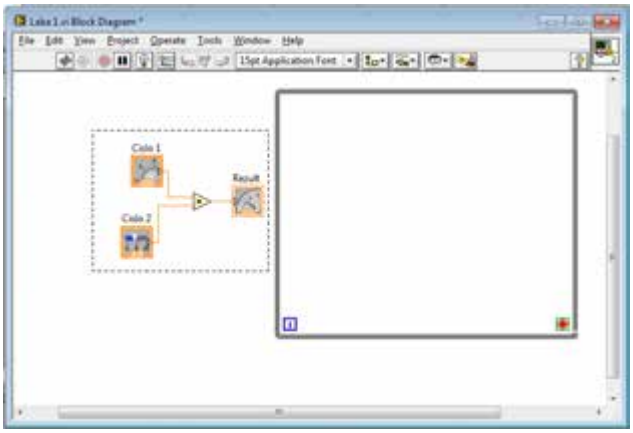


Рис. 3.8. – Розміщення розробленої програми в межах циклу

Отримані теоретичні та експериментальні дані зручно аналізувати у графічному вигляді. Тому наступним кроком буде встановлення графічного екрану, що буде відображати графічну залежність значення добутку введених даних від часу. Для цього обираємо елемент індикаціїї “Waveform Chart” (рис. 3.9, а). Зверніть увагу на те, що після встановлення відповідного елементу індикації, його іконка з’явилась і у вікні блок-діаграми. Переміщуємо даний інструмент в цикл та з’єднуємо з ланцюгом зв’язку, що йде після добутку вхідних даних. Також важливо пам’ятати, що програма в середовищі LabView виконується паралельно. Тобто, ми одночасно маємо можливість виводити дані в різних вікнах індикації. Тому не має необхідності видаляти попередньо встановлений індикатор “Rezult” типу “Gauge” (рис. 3.9, б).

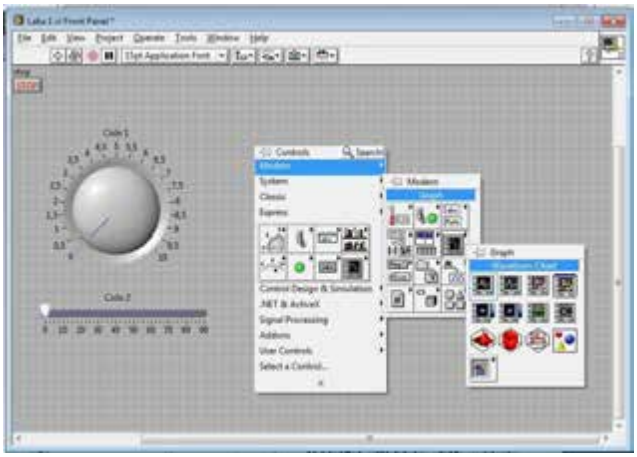


Рис. 3.9, а – Вибір елемента індикації “Waveform Chart”

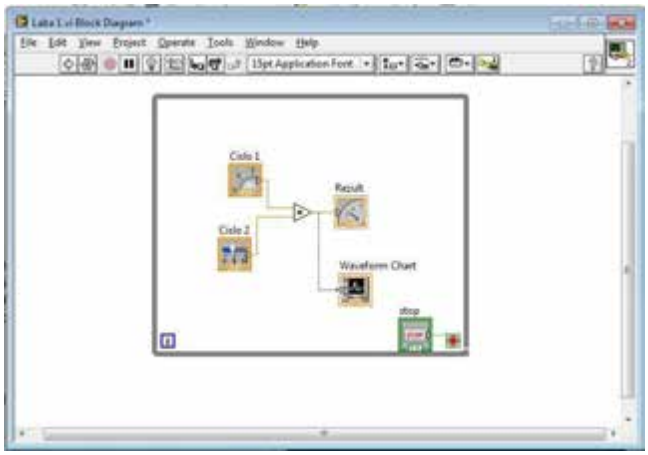


Рис. 3.9, б – Паралельне з’єднання двох елементів індикаціїї у вікні “Block Diagram”

Для того, щоб значення добутку, що виводяться у вікні графіку, відображались з певною затримкою, встановимо оператор “Wait Until Next ms Multiple” (рис. 3.10, а) та створимо для нього константу, значення якої буде відповідати затримці при виведенні результату (аналогічно до створення елемента керування для циклу) (рис. 3.10, б).

Рис. 3.10, а – Вибір оператора“Wait Until Next ms Multiple”

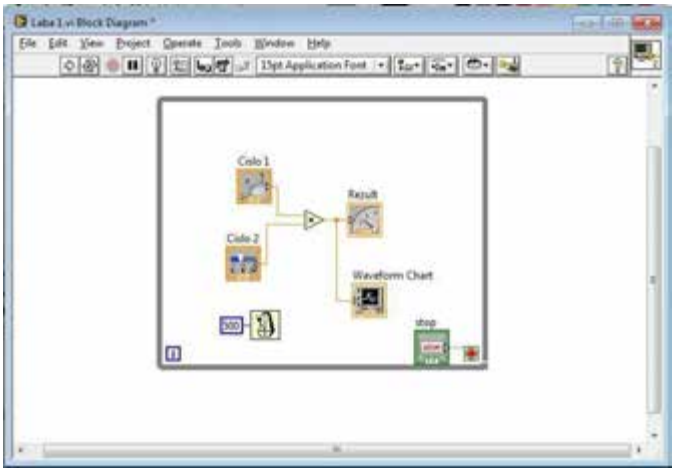


Рис. 3.10, б – Створення константи для регулювання затримки

Програма готова до роботи. Тепер не треба для виконання програми використовувати циклічний режим. Результат роботи програми відображається як на індикаторі, так і на графіку (рис. 3.11, а). Вся по дальша робота над програмою зводиться до зміни зовнішнього інтерфейсу: колір, розмір елементів керування та індикації, їх розташування і т.п. Або заміна констант на елементи керування (наприклад, в даному прикладі готової програмі константу для реалізації затримки відображення результату змінено на елемент керування типу “Fill Slide” (рис. 3.11, б))

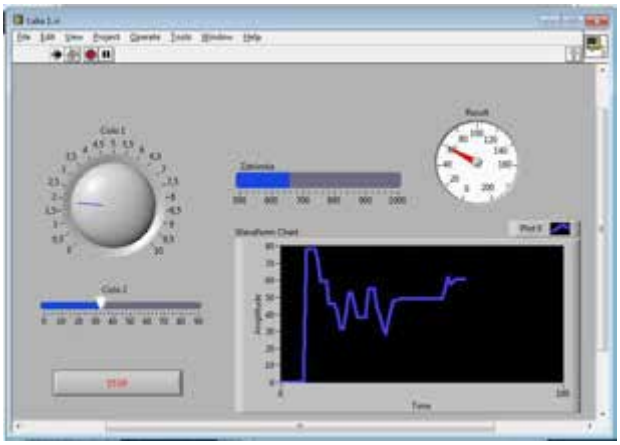


Рис. 3.11, а – Програма для розрахуну та відображення добутку двох числових значень, що задаються користувачем

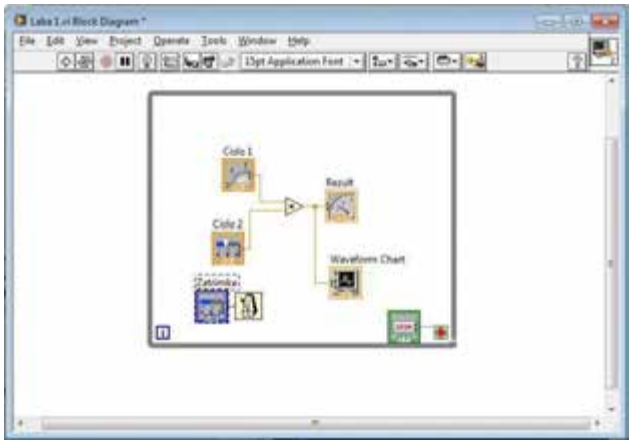


Рис. 3.11, б – Зміна константи на елемент керування для регулювання затримки в онлайн-режимі

Завдання для самостійної роботи

Розробити програму для розрахунку наступних залежностей та відображення результату при зміні значення N в заданому діапазоні у вікні індикатора та на графіку з затримкою T.

Варіант	Завдання		
1	A = N · (35,7 + 13,6)	N = 45 … 89	T = 400
2	A = (N + 3,6)/2	N = 34 … 67	T = 500
3	A = N + 36/7,5	N = 44 … 89	T = 600
4	A = 2,3 · 23 − N	N = 176 … 367	T = 700
5	A = N/(354 − 136)	N = 434 … 767	T = 300
6	A = N · 37 + 46	N = 67 … 234	T = 400
7	A = N − 12 + 13,6	N = 34 … 67	T = 400
8	A = N/3 + 3,4	N = 44 … 89	T = 500
9	A = (N · 3 + 147)/2	N = 176 … 367	T = 600
10	A = (N + 3,6)/2	N = 434 … 767	T = 700
11	A = N + 36/7,5	N = 67 … 234	T = 300
12	A = 2,3 · 23 − N	N = 45 … 89	T = 400
13	A = N · (35,7 + 13,6)	N = 44 … 89	T = 500
14	A = N · 37 + 46	N = 176 … 367	T = 600
15	A = N − 12 + 13,6	N = 434 … 767	T = 700
16	A = N/3 + 3,4	N = 176 … 367	T = 300
17	A = 2,3 · 23 − N	N = 434 … 767	T = 400
18	A = N/(354 − 136)	N = 67 … 234	T = 500
19	A = N · 37 + 46	N = 45 … 89	T = 600
20	A = N − 12 + 13,6	N = 45 … 89	T = 700
21	A = N/3 + 3,4	N = 34 … 67	T = 300
22	A = N/3 + 3,4	N = 44 … 89	T = 400
23	A = (N · 3 + 147)/2	N = 176 … 367	T = 500
24	A = (N + 3,6)/2	N = 434 … 767	T = 600
25	A = N + 36/7,5	N = 67 … 234	T = 700

Лабораторна робота №4. Моделювання роботи АЦП і ЦАП

Мета роботи: вивчення і моделювання роботи АЦП і ЦАП в середовищі LabView.

Загальні теоретичні відомості

Цифро-аналоговий перетворювач (скорочено ЦАП) – це одна з найбільш важливих схем сполучення, що застосовуються для організації зв'язку між аналоговими і цифровими пристроями. ЦАП є основою багатьох електронних схем і пристроїв, включаючи цифрові вольтметри, графопобудівники, осцилографи і багато інших пристроїв, керовані комп'ютером.

ЦАП - це електронний пристрій, що перетворює цифровий логічний сигнал в аналоговий сигнал. Вихідна напруга ЦАП - це набір бітів вхідного сигналу, зважених певним чином:

$$DAC = \sum_{i-0} w_i b_i,$$

де, w_i – ваговий коефіцієнт; b_i – значення біту (1 або 0), i – індекс номеру біту. У випадку двійкової системи числення $w_i=2^i$, отримаємо вираз для восьми бітного ЦАП:

DAC = 128*b*7 + 64*b*6 + 32*b*5 + 16*b*4 + 8*b*3 + 4*b*2 + 2*b*1+ 1*b*0

На рис.1 показана реалізація восьми бітного ЦАП в системі LabView.

Вісім булевих перемикачів на лицевій панелі встановлюють вхідні біти b_0…b_7. Коли запущений процес моделювання, вісім індикаторів відображають величину вхідного сигналу, а вихідний сигнал відображається на числовому індикаторі.

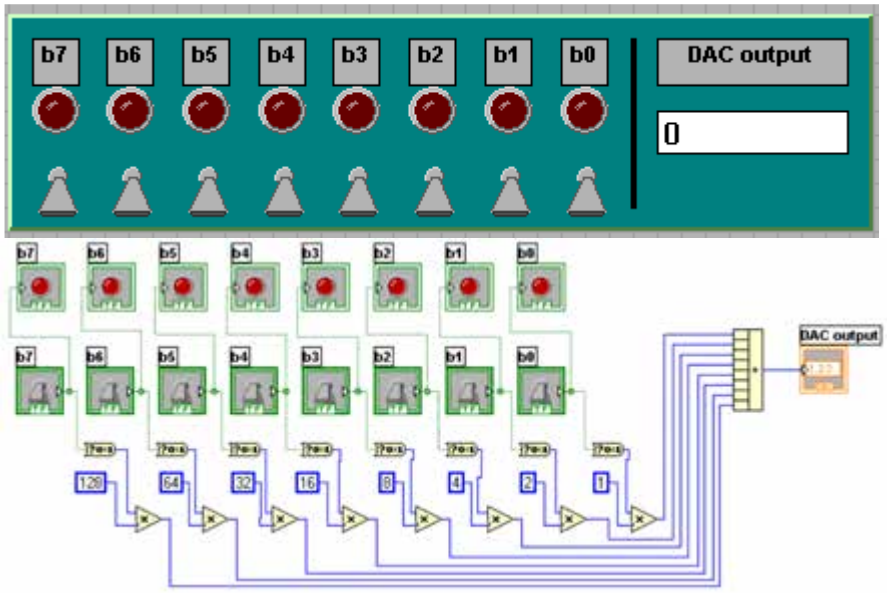


Рис.1 Реалізація ЦАП в системі LabView.

Для генерації аналогового сигналу можна використовувати будь-яку послідовність бітів, що подаються з постійною швидкістю на вхід ЦАП. Найпростіша послідовність виходить на виході 8-бітного двійкового лічильника. З її допомогою генерується цифровий сигнал в межах від 0 до 255 одиниць. Для демонстрації такого процесу необхідно приєднати простий лічильник до ЦАП. Після цього вихідний сигнал подається на графічний індикатор. Швидкість наростання графіка визначається частотою підрахунку: більше частота - більше кут нахилу. Коливальний модуль генерує тактовий сигнал. Коли відбувається переповнення лічильника від (11111111) до (00000000), аналоговий сигнал швидко спадає від 255 до 0. Такий сигнал називають ступінчастим, оскільки він схожий на східці. Модель такого приладу представлена на рис. 2.

Лічильник від 0 до 255 організований за допомогою зсувних регістрів («Shift register»), і вузла порівняння поточного значення лічильника зі значенням 255. Сигнал з цього лічильника

перетворюється в масив біт, а потім в кластер біт, який розбивається на потоки біт і подається на ЦАП.

З виходу ЦАП знімається «аналоговий» сигнал, що відображається на графічному індикаторі «Waveform Chart», для режимів 4- і 6-бітного ЦАП цей сигнал множиться на константу, щоб привести його до масштабу 0 ÷ 255. Значок з годинником - це функціональна одиниця «Wait» - забезпечує затримку виконання коду програми на вказану кількість мілісекунд.

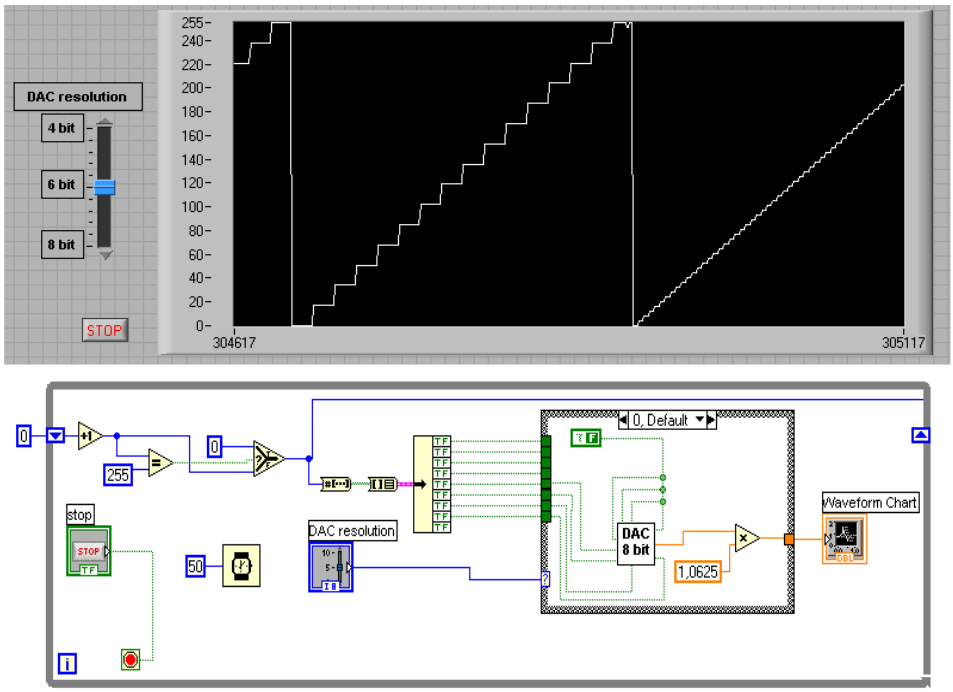


Рис. 2. Модель сигналу 4, 6, 8-бітних ЦАП.

Аналогово-цифровий перетворювач (АЦП) – це другий ключовий елемент, що забезпечує взаємодію аналогових і цифрових пристроїв. АЦП є основою перетворювачів сигналів в цифрову форму, цифрових вольтметрів, багатоканальних аналізаторів, осцилографів і багатьох інших приладів, яким потрібен цифровий код даних. Існує кілька різних типів АЦП. Найбільш поширеними є інтегруючи, спостережні і перетворювачі послідовного наближення. В роботі досліджуються інтегруючи і спостережні аналого-цифрові перетворювачі.

Призначення АЦП полягає в генерації двійкового цифрового коду, пропорційного вхідному аналоговому сигналу (Рис. 3).

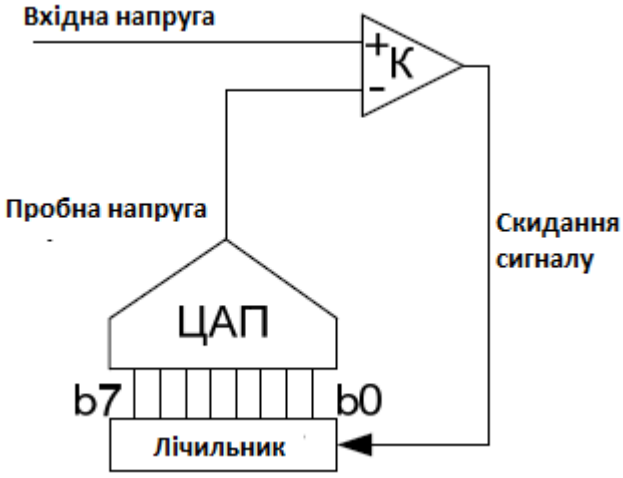


Рис. 3. Схема 8-бітного АЦП.

Лічильник створює пробну двійкову послідовність, яка конвертується в аналогову напругу за допомогою цифро-аналогового перетворювача. ЦАП є базовим елементом багатьох схем АЦП, а його принцип роботи вже обговорювалося вище. Після цього пробне напруга порівнюється з вхідним сигналом. Якщо вхідна напруга більше пробного сигналу, лічильник збільшує значення, щоб наблизити пробний сигнал до рівня вхідної напруги. Якщо ж вхідний сигнал менше пробного, лічильник зменшує своє вихідне значення з тим, щоб рівень пробного сигналу наблизився до рівня вхідного. Цей процес триває до тих пір, поки компаратор не змінить знак. У цей момент рівень пробного сигналу буде в межах одного відліку від рівня вхідного напруги. При збільшенні числа розрядів лічильника буде збільшуватися і дозвіл ЦАП такого типу.

Інтегруючий АЦП. В інтегруючому АЦП для генерації пилкоподібної пробної напруги використовуються двійковий лічильник і цифро-аналоговий перетворювач. Модель приладу представлена на рис. 4.

Двійковий лічильник працює у вільному режимі. Всякий раз, коли пробний сигнал стає більше вхідного, компаратор змінює знак. Таке перетин пилкоподібного сигналу з вхідною напругою можна спостерігати на графічному індикаторі. Двійкова величина лічильника в точці перетину – це оцифрований сигнал. Зміна стану компаратора свідчить про перетин.

Щоб змоделювати дійсно пилкоподібний сигнал АЦП, необхідно, щоб при зміні стану компаратора відбувалося повернення лічильника в початковий стан. Для цього простий двійковий лічильник замінюється на двійковий лічильник з обнуленням. Як тільки рівень пробного сигналу досягає вхідного, двійковий лічильник повертається в початковий стан, а пилкоподібний цикл починається знову. На індикаторі, показаному вище, рівень вхідного сигналу змінювався тричі.

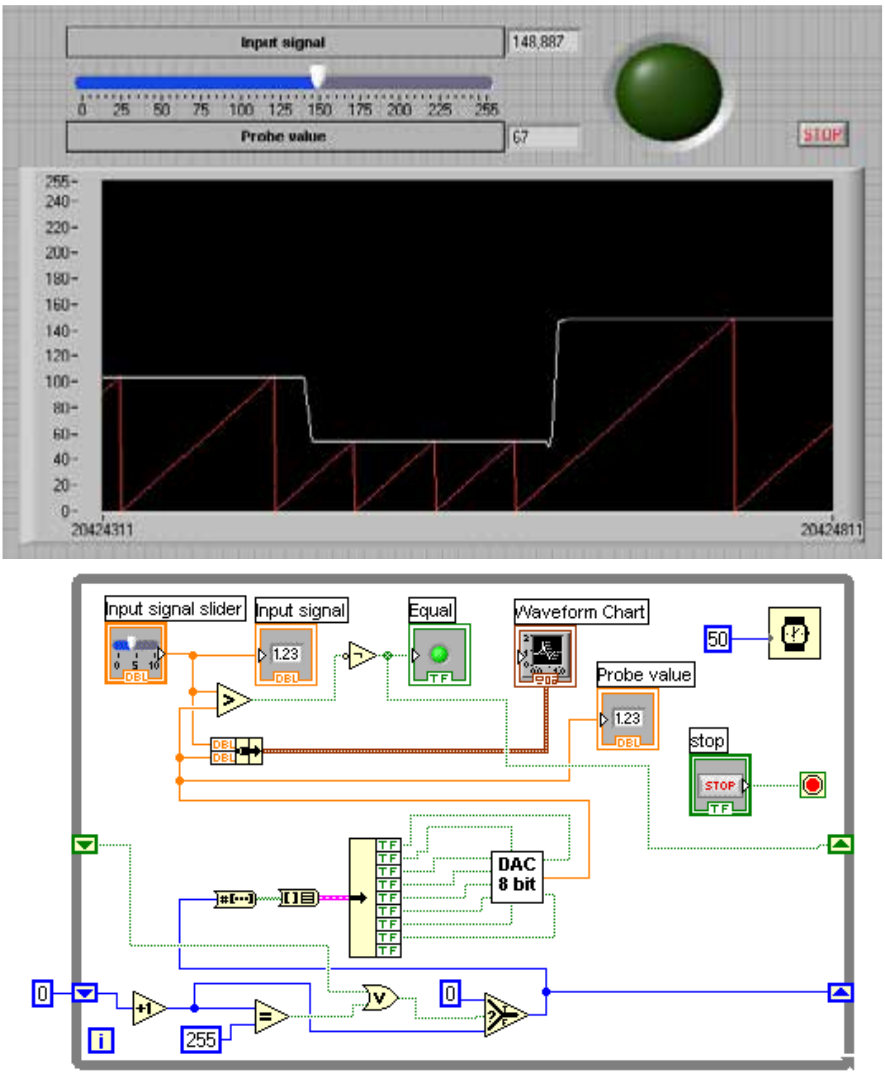


Рис. 4. Модель інтегрованого АЦП.

Спостережні АЦП. Основне завдання спостережних АЦП – швидко наближатися до рівня вхідного сигналу. У точці, яка визначається перетином пилкоподібного і вхідного спостережного сигналів алгоритм закінчує свою роботу.

Спостережний алгоритм досить простий: якщо рівень пробного сигналу більше рівня вхідного сигналу,зменшити число лічильника на одиницю, якщо рівень пробного сигналу менше рівня вхідного сигналу, збільшити число лічильника на одиницю, повторювати нескінченно.

Однак, якщо рівень вхідного сигналу змінюється, АЦП повинен повернутися до генерації пилкоподібної напруги, щоб нагнати зміну вхідного сигналу. У тому випадку, коли тактовий генератор досить швидкий, відстеження відбувається оперативно. Але якщо вхідний сигнал змінюється занадто швидко, оцифрований сигнал пропадає до тих пір, поки пробний сигнал знову не нажене вхідний. Насправді існує гранична швидкість спостереження АЦП. Вона обмежує максимальну частоту зміни вхідного сигналу.

На рис. 5 представлена лицева панель приладу, що моделює спостережний АЦП.

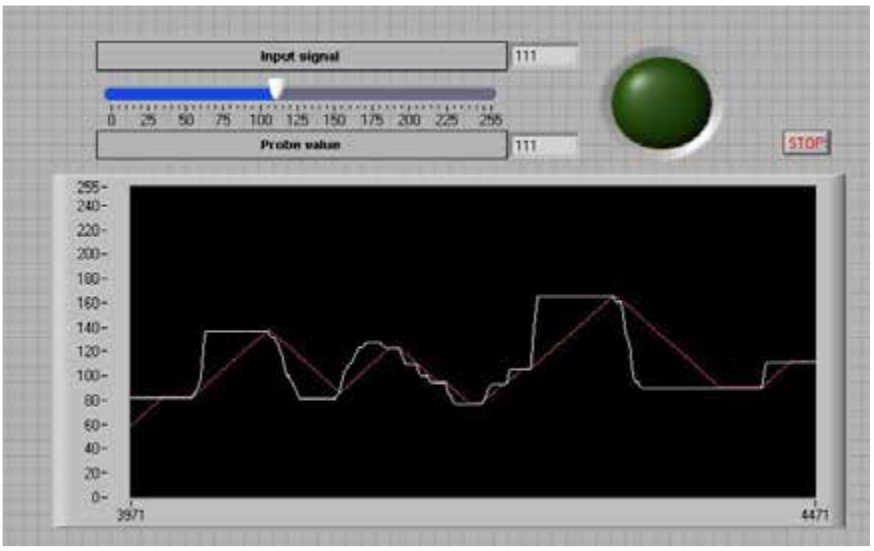


Рис. 5. Робота спостережного АЦП.

Оскільки в спостережному АЦП використовується лічильник, який працює в режимі зростання (зменшення), то коли вхідний сигнал раптово спадає нижче (зростає вище) рівня пробної напруги, спостережний АЦП повертається до генерації спадаючої (зростаючої) пилкоподібної напруги до тих пір, поки пробна напруга досягне рівня вхідного сигналу.

Виконання лабораторної роботи

1. Створити модель ЦАП, що показана на рис. 1. Розібратися з її роботою.
2. Створить бібліотеку підпрограм і збережіть модель, що створення в п.1.
3. Створить модель, що показана на рис. 2. Вивчити принцип роботи нових вузлів. Збережіть файл для демонстрації роботи.
4. Створить модель та лицеву панель як показано на рис. 4 та рис. 5. Розібратися з роботою схеми та її компонентів. Необхідно зберегти файл.
5. Продемонструвати роботу ЦАП і АЦП.

В звіті необхідно подати PrintScreen схем, лицевих панелей. Опис вузлів: «Wait», «Select», «Equal», «Increment», «Number to Boolean array», «Array to cluster», «Unbundle», «Bundle», «Greater», «Waveform chart».

Лабораторна робота №5. Робота арифметико-логічного пристрою центрального процесора

Мета роботи: ознайомитися з роботою, вивчення і моделювання роботи деяких функцій арифметико-логічного пристрою в середовищі Labview.

Теоретичні відомості

Основний елемент будь-якого комп'ютера - це центральний процесор (ЦП). ЦП пов'язаний з оперативною пам'яттю за допомогою шини, що передає дані в двох напрямках. Команди, постійні і змінні величини, які використовуються в програмах і постійно зберігаються в пам'яті, розташовані в певній послідовності. ЦУ звертається до цієї послідовності, керуючи і маніпулюючи адресною шиною. Спеціальні елементи пам'яті, що називаються вхідним/вихідним (I/O) портами, передають двійкову інформацію до або із зовнішнього світу у формі паралельних або послідовних байтів даних. Системний тактовий генератор здійснює управління всіма логічними елементами, тригерами і регістрами, забезпечуючи, щоб всі біти були в необхідний час в потрібному місці і щоб потоки даних не перекривалися. З усіх частин комп'ютера (ЦП, пам'ять, пристрої введення-виведення і тактовий генератор) процесор є найбільш важливим.

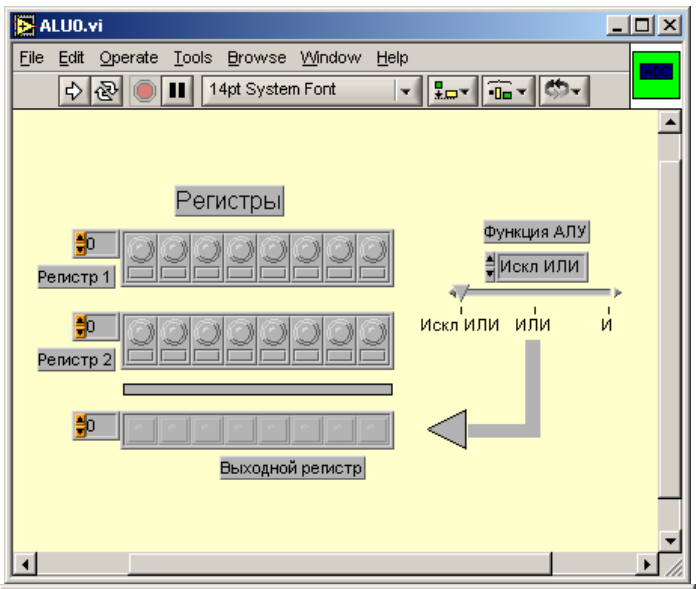
ЦП складається з декількох елементів. Серед них: арифметико-логічний пристрій (АЛП), дешифратор команд, лічильник команд і цілий набір внутрішніх елементів пам'яті, що називаються регістрами. При звичайній роботі ЦП лічильник команд звертається до пам'яті за допомогою адресної шини з метою отримання наступної інструкції. Ця інструкція по шині даних надходить у внутрішні регістри. Перша частина інструкції передається в дешифратор, який вирішує, які канали необхідно відкрити і закрити, щоб виконати інструкцію. У деяких випадках інструкція містить всю необхідну для роботи інформацію. Як приклад такої інструкції можна назвати команду «очистити акумулятор». В інших випадках для виконання інструкції необхідна додаткова інформація, тому знову відбувається звернення до пам'яті за цією інформацією. Як приклад такої інструкції можна привести команду «завантажити в Регістр 2 постійну величину 5». Як тільки вся інформація збереться в одному місці, інструкція виконується за допомогою відкриття і закриття різних логічних елементів.

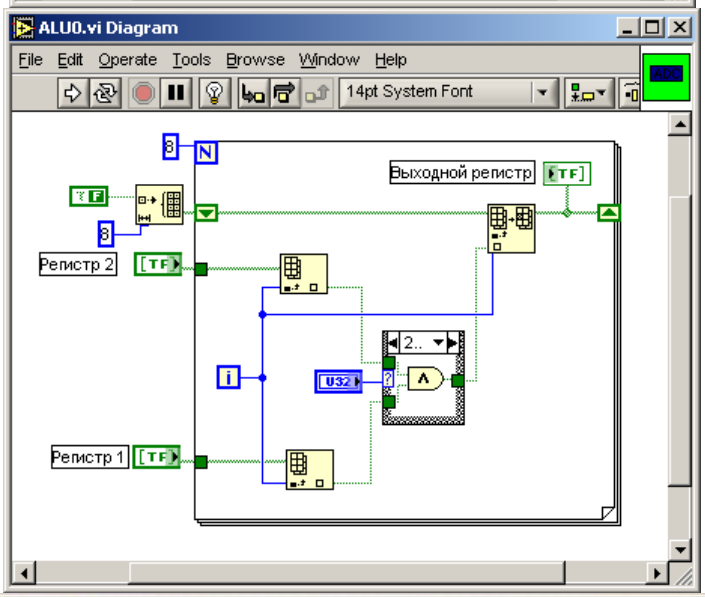
Типові команди, доступні для всіх ЦП, складаються з простих операцій з даними, вже знаходяться всередині самого ЦП. Серед них очищення, доповнення або збільшення суматора. У більш складних інструкціях використовуються два внутрішніх регістра або дані, що надходять з пам'яті. На даному занятті за допомогою основних логічних функцій буде показано, як ЦП виконує прості і деякі більш складні операції.

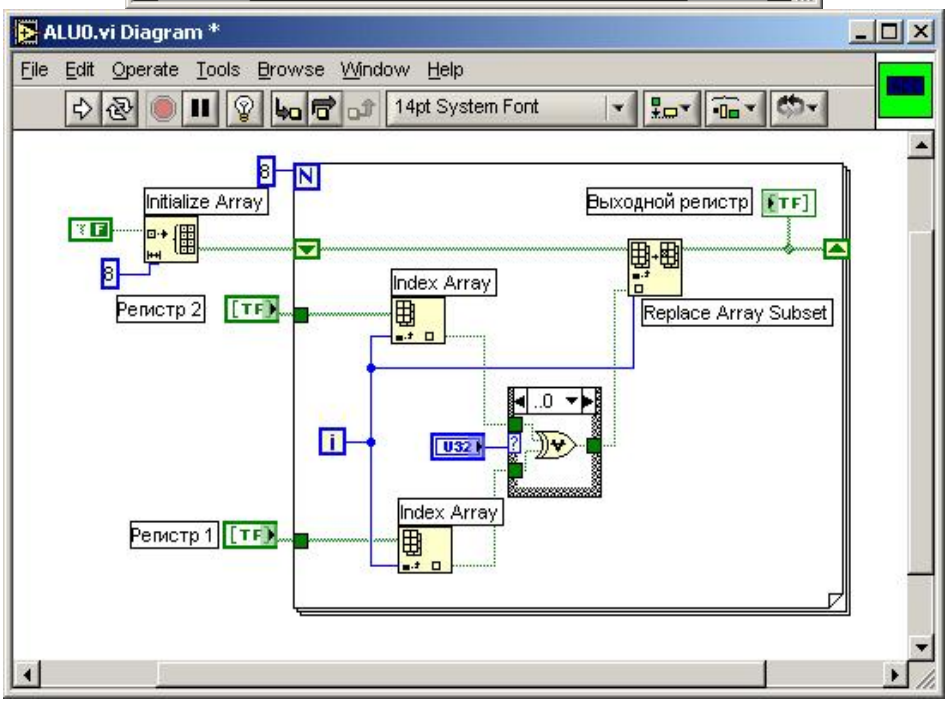
Хід роботи

Робота арифметико-логічного пристрою

Арифметико-логічний пристрій (АЛП) - це набір програмованих двовходових логічних елементів, що оперують з паралельними масивами даних, розміром 4, 8, 16 і 32 біта. В методичних вказівках основну увагу буде сфокусовано на вивченні 8-бітових ЦП. Вхідні регістри будуть називатися Регістр1 і Регістр 2, і для простоти результат операції буде завантажуватися в третій регістр, званий вихідним. Тип команди (I, АБО, що виключає АБО) мнемонічно визначається як I R1, R2.


ALU0.vi
File Edit Operate Tools Browse Window Help
14pt System Font
Регистры
Регистр 1
Регистр 2
Выходной регистр
Функция АЛУ
Искл ИЛИ
Искл ИЛИ ИЛИ И


ALU0.vi Diagram
File Edit Operate Tools Browse Window Help
14pt System Font
Выходной регистр
Регистр 2
Регистр 1
2..
U32


ALU0.vi Diagram *
File Edit Operate Tools Browse Window Help
14pt System Font
Initialize Array
Выходной регистр
Index Array
Replace Array Subset
Регистр 2
Регистр 1
..0
U32
Index Array

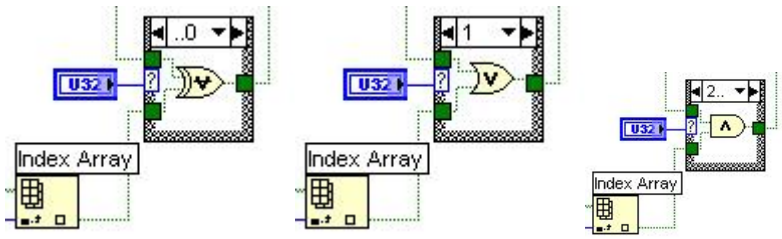


Рис. 1. Моделювання арифметико-логічного пристрою.

ЗАДАЧА 1:

Створити і зберегти віртуальний прилад ALUO.vi. При моделюванні в середовищі LabView - в віртуальному приладі ALUO.vi - для представлення регістрів R1 і R2 використовуються 1D масиви, що складаються з булевих перемикачів. Вихідний регістр представляє з себе масив булевих індикаторів. Функції (І, АБО, виключаюче АБО) можна вибирати за допомогою повзункового перемикача. Дані заносяться у вхідні регістри натисканням на кнопку, що знаходиться під відповідним бітом. При натисканні на кнопку «Run» відбувається виконання обраної логічної операції. Нижче слідує ряд прикладів елементарних операцій ЦП. Що являють собою наступні операції:

И R1[$00], R2[$XX]

ИЛИ R1[$FF], R2[$XX]

Исключающее ИЛИ R1[$55], R2[$FF]?

У всіх зазначених випадках дані, які необхідно ввести, вказані в прямокутних дужках [] і повинні представляти шістнадцятирічне число (наприклад, $ F3). X позначає будь-який шістнадцятковий символ. Досліджуйте представлені операції за допомогою віртуального приладу ALUO.vi.

Операція І встановлює всі розряди вихідного регістру в 0, отже, ця дія еквівалентно команді ОЧИСТИТИ Вихідний Регістр. Операція АБО встановлює всі біти вихідного регістра в 1, отже, це дія еквівалентно команді ВСТАНОВИТИ Вихідний Регістр в 1. Третя операція інвертує розряди R1, тобто, еквівалентна операції ДОПОВНЕННЯ Регістру 1.

Розглянемо команду «Завантажити в вихідний регістр вміст Регістру 1». Мовою звичайного програмування дану команду можна записати в вигляді «Вихід = Регістр 1». Встановіть R1 в ALUO.vi в деякий початковий стан і виконайте операцію I R1, R2 [$ FF].

Ще одна цікава комбінація - Виключне Або R1, R2 - реалізує іншу поширену команду - ОЧИСТИТИ R1. З цих прикладів стає зрозумілим, що багато команд ЦП, що мають специфічний зміст в контексті реалізації програмного забезпечення, виконуються за допомогою основних логічних елементів.

НАКОПИЧУЮЧИЙ СУМАТОР

У віртуальному приладі ALUO.vi команди ЦП виконуються за допомогою «висмикування» одного біта за одну ітерацію за допомогою функції LabView Index Array. Після цього виконується операція арифметико-логічного пристрою над цим бітом. Результат передається в вихідний масив під тим же самим індексом за допомогою функції Replace Array Element. Після восьми циклів кожен біт (0 ... 7) проходить через АЛП. Після цього робота ЦП завершується.

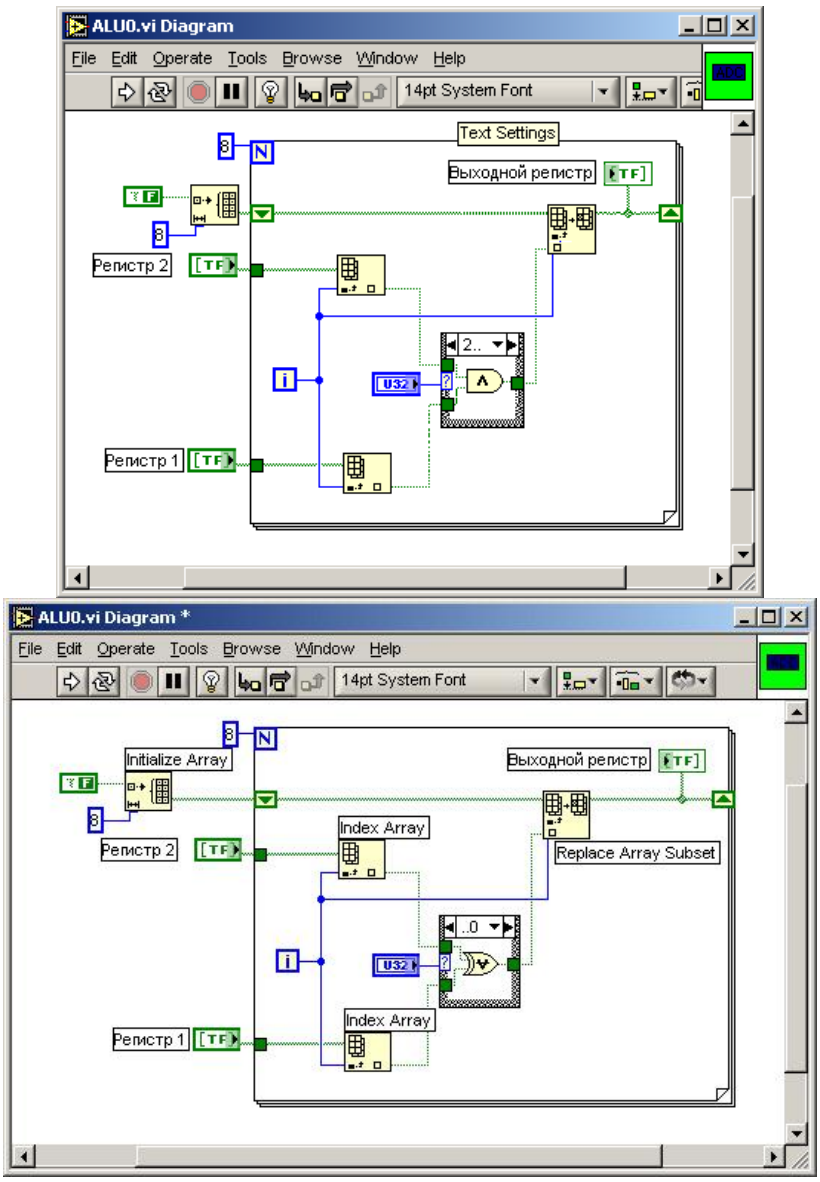


Рис. 2. Прилад, що моделює роботу 8-ми бітового АЛП.

У середовищі LabView немає необхідності «висмикувати» кожен біт, оскільки це завдання може бути зроблена автоматично за допомогою відключення індексації в тунелі структури «Цикл по умові». Провідники даних, що представляють масиви, зображуються жирною лінією, на противагу тонкої лінії, що зображує провідник передачі простих даних всередині циклу. Будьте уважні при розгляді наступного прикладу, в якому використовується ця особливість LabView.

У багатьох процесорах другий вхідний регістр, R2, з'єднаний з вихідним регістром, так що на наступній ітерації вихідний стан стає вхідним. Така структура дозволяє спростити пристрій ЦП, але що більш важливо, при цьому вихідний регістр стає накопичувачем.

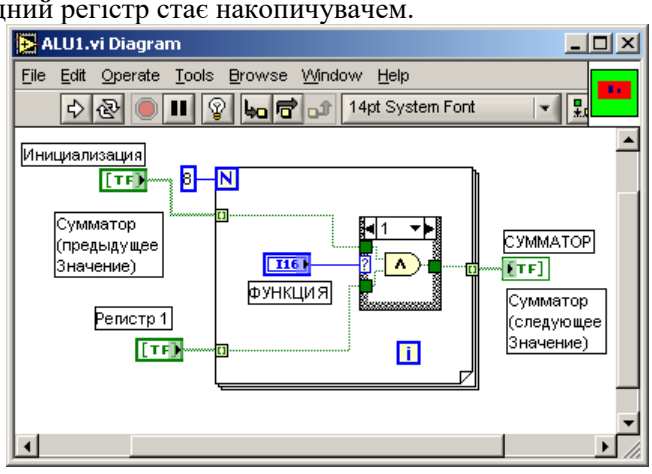


Рис.3. При моделюванні АЛП використовується автоіндексація всередині тунелю циклу по умові.

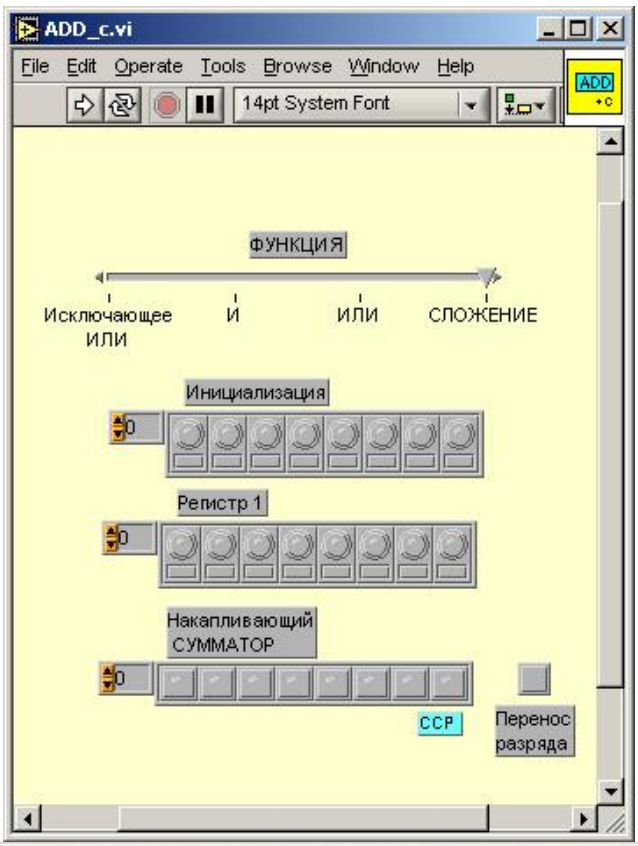
Операція додавання

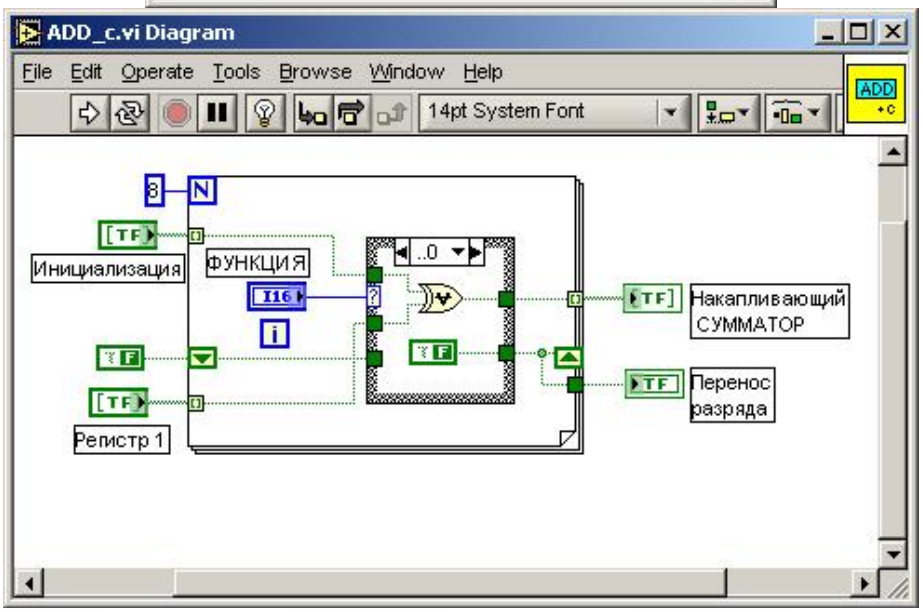
Арифметико-логічний пристрій виконує не тільки логічні операції, він призначений і для виконання арифметичних дій. Відомо, що за допомогою функції виключає Або відбувається додавання окремих бітів при довічним складання, а обчислення сигналу перенесення відбувається за допомогою функції І. При об'єднанні цих функцій виходить напівсуматор (біт 1 + біт 2 = сума + сигнал перенесення (якщо є)). Щоб результат складання зрушити в наступний розряд, використовується повний суматор, в якому відбувається складання бітів, що надходять з двох входів, а також враховується сигнал початку перенесення з попереднього розряду.

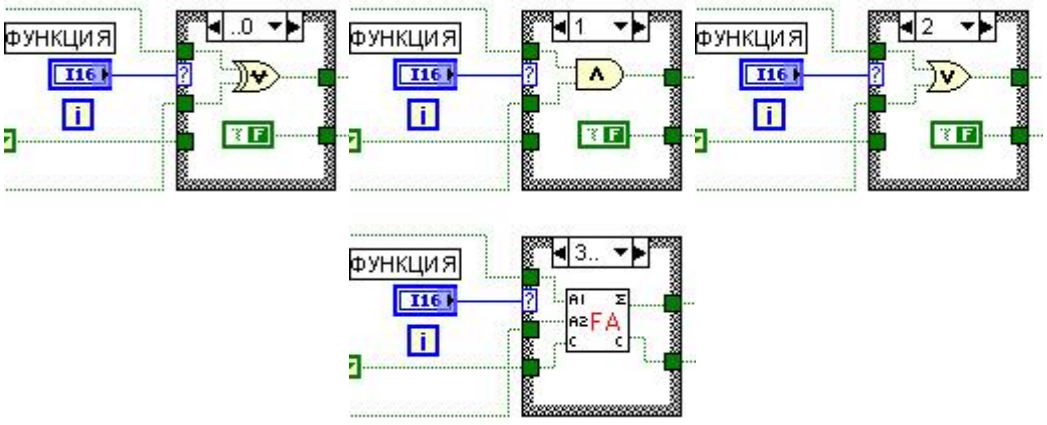
ЗАДАЧА 2:

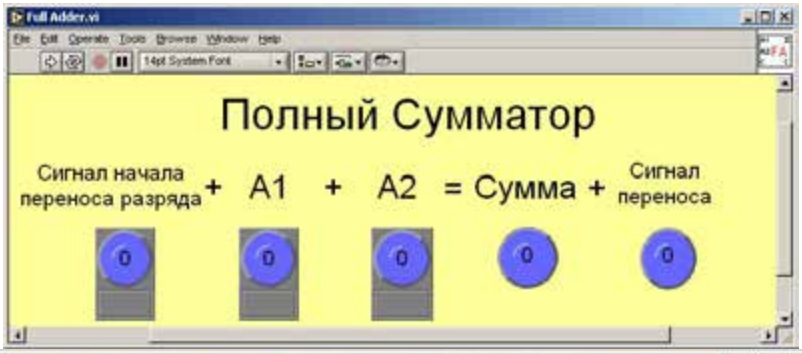
Створіть і збережіть віртуальний прилад **ADD_c.vi**.

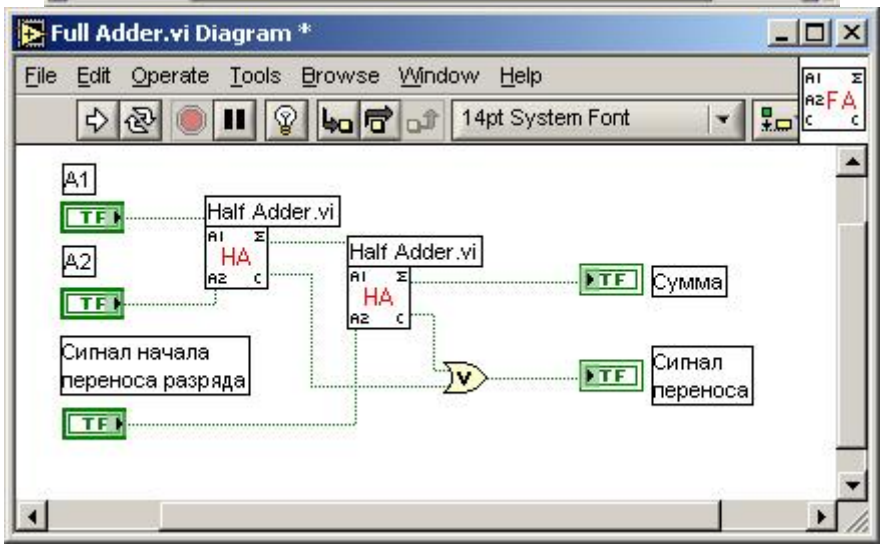
Віртуальний прилад ADD_c.vi додає дію Додавання до операцій АЛП. Повний суматор, показаний нижче, складає два вхідних біта, враховуючи при цьому сигнал про початок перенесення з попереднього розряду. Ця дія моделюється за допомогою булевого зсувного регістру. Нова операція - {СУМА +1, А} - може бути тепер додана в структуру і список опцій в віртуальному приладі.

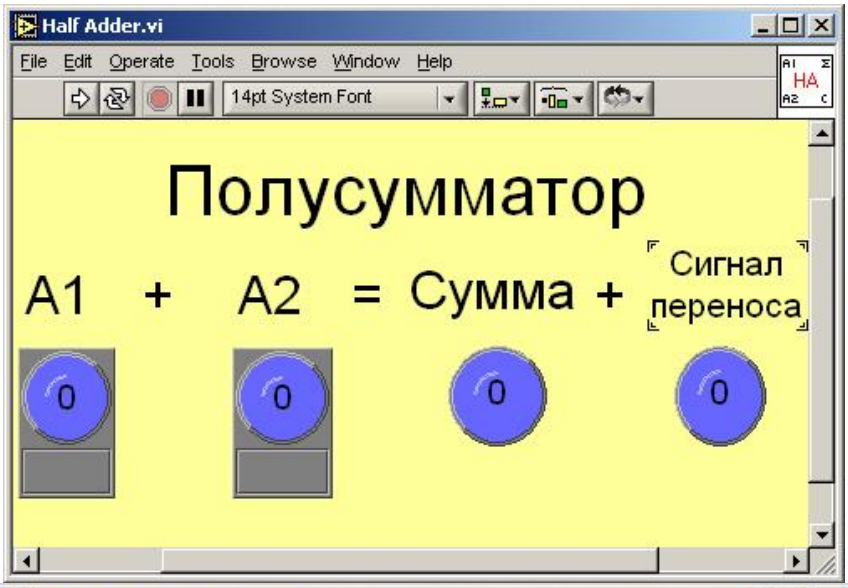

ADD_c.vi
File Edit Operate Tools Browse Window Help
14pt System Font
ADD
+c
ФУНКЦИЯ
Исключающее ИЛИ
И
ИЛИ
СЛОЖЕНИЕ
Инициализация
Регистр 1
Накапливающий СУММАТОР
ССР
Перенос разряда

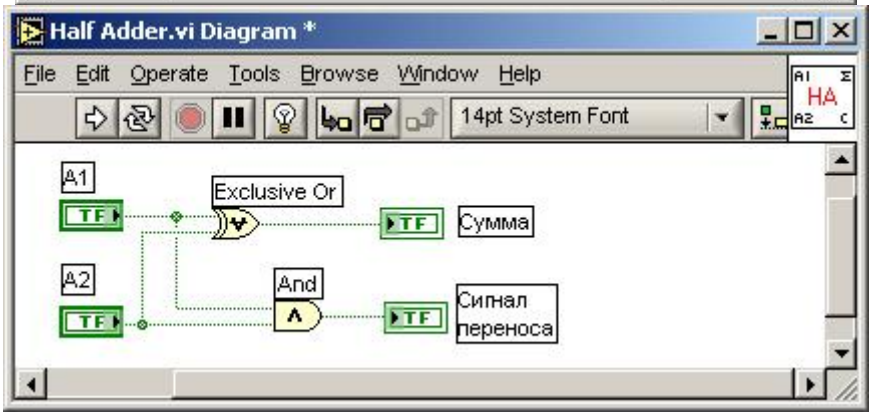

ADD_c.vi Diagram
File Edit Operate Tools Browse Window Help
14pt System Font
ADD
+c
Инициализация
ФУНКЦИЯ
Регистр 1
Накапливающий СУММАТОР
Перенос разряда


ФУНКЦИЯ
..0
1
2
3..
FA


Full Adder.vi
Полный Сумматор
Сигнал начала переноса разряда
+
A1
+
A2
= Сумма +
Сигнал переноса
0
0
0
0
0


Full Adder.vi Diagram *
File Edit Operate Tools Browse Window Help
14pt System Font
A1
A2
Half Adder.vi
HA
Half Adder.vi
HA
TF
Сумма
Сигнал начала переноса разряда
Сигнал переноса


Half Adder.vi
File Edit Operate Tools Browse Window Help
14pt System Font
HA
Полусумматор
A1
+
A2
= Сумма +
Сигнал переноса
0
0
0
0


Half Adder.vi Diagram *
File Edit Operate Tools Browse Window Help
14pt System Font
HA
A1
A2
Exclusive Or
And
TF
Сумма
Сигнал переноса

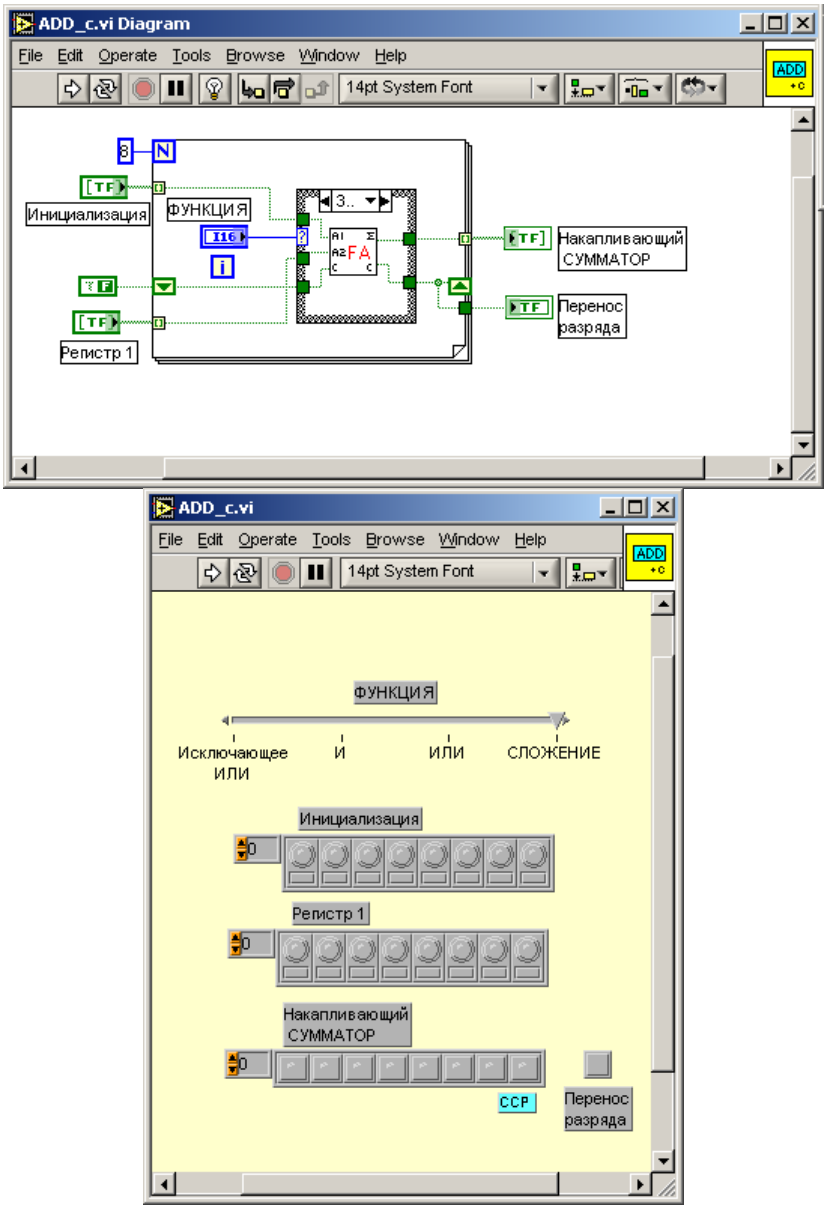


Рис. 5.Операція АЛП – додавання з переносом розряду.

ЗАДАЧА 3:

Двійковий лічильник

Розглянемо програму, яка генерує двійкові числа за допомогою 8-бітового двійкового лічильника. Її роботу можна описати наступним чином: після очищення суматора, додавати до нього одиницю знову і знову протягом n раз.

На будь-якій мові лінійного програмування цю програму можна записати у вигляді:

```
Start  CLEAR A        : обнулить все разряды в сумматоре
Loop   INC A          : увеличить на 1 содержимое сумматора
       REPEAT Loop N  : выполнить последнюю команду n раз
```

Виконання над вмістом регістра і числом $ 00 операції I приведе до очищення суматора, CLEAR A. У списку функцій ЦП операція I має номер 1. Операція INC A - це I 1, A - і йде під номером 3 у списку. Моделювання представлено нижче.

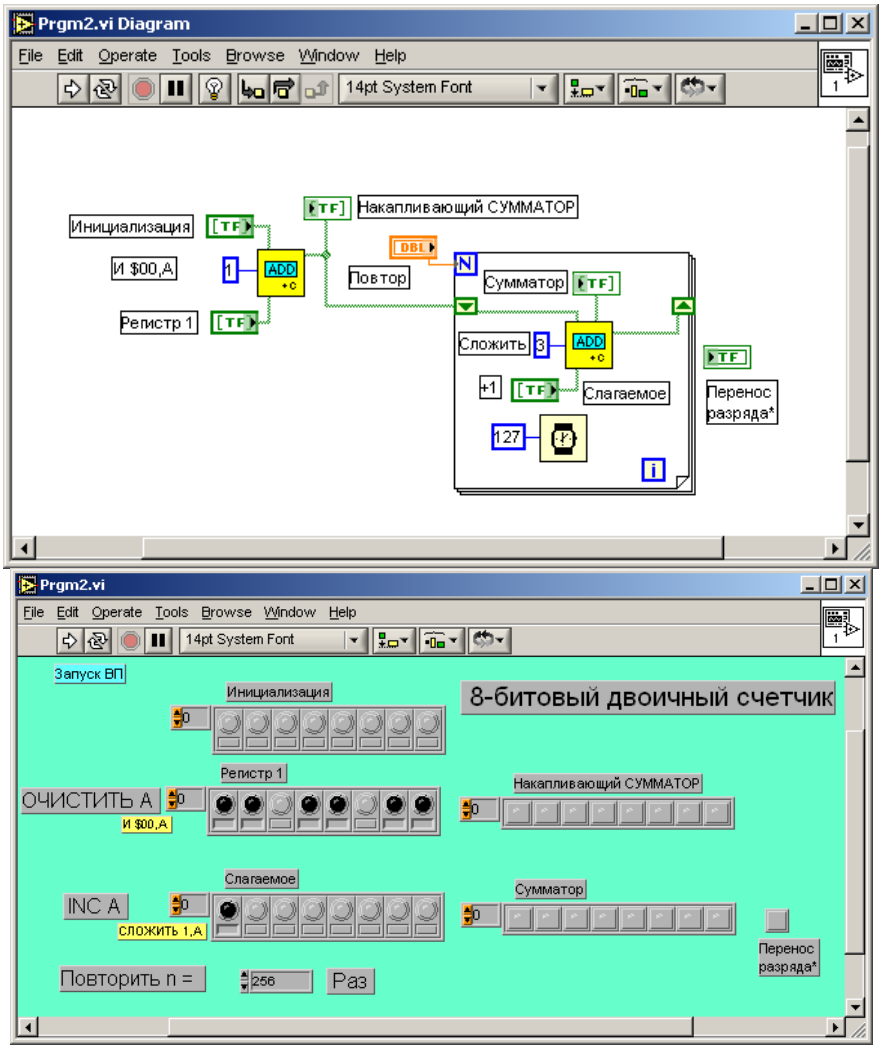


Рис. 6. Віртуальний прилад, що моделює 8-ми бітний лічильник.

Відзначимо, що термінал сигналу про перенесення розряду ні з чим не з'єднаний. Команда збільшень не впливає на перенос. При з'єднанні терміналу сигналу перенесення дана команда правильно запишеться як I + 1, A.

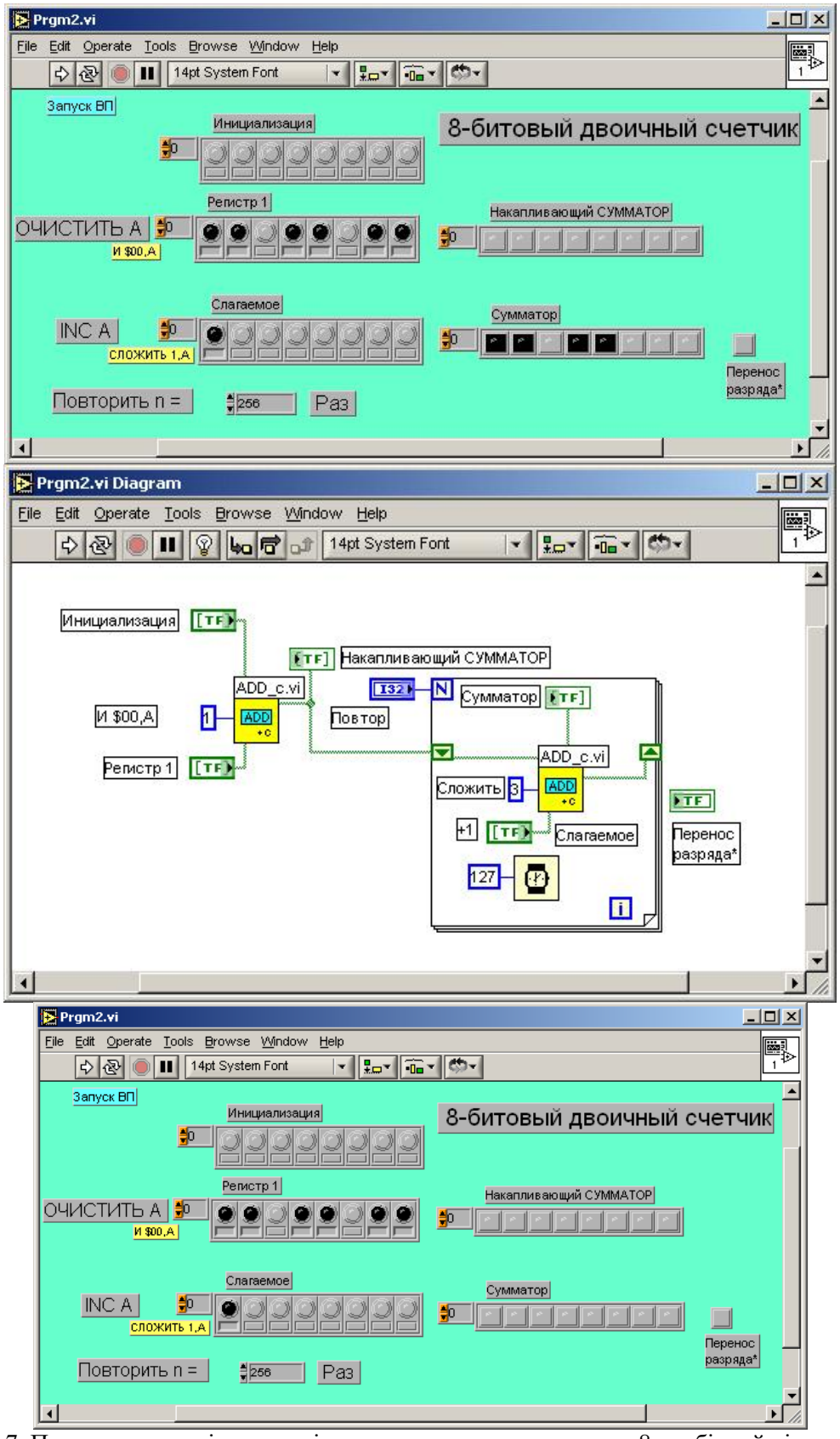


Рис. 7. Панель приладу і панель діаграми приладу, що моделює 8-ми бітний лічильник.

Порядок виконання лабораторної роботи

1. Створити і зберегти віртуальний прилад по задачі 1. При моделюванні в середовищі LabView перевірити роботу.

2. Створити і зберегти віртуальний прилад по задачі 2. При моделюванні в середовищі LabView перевірити роботу.
3. Створити і зберегти віртуальний прилад по задачі 3. При моделюванні в середовищі LabView перевірити роботу.
4. Оформити звіт та пояснити роботу по кожній задачі.

Лабораторна робота №6. Знайомство із системою Paralab

Мета роботи: Вивчення основних функцій програмної системи Paralab. Навчитися правильно виконувати експеримент, згідно с заданими вхідними даними.

Теоретичні відомості

Paralab – інтегроване середовище для проведення паралельних обчислювальних експериментів.

Програмна система Paralab призначена для вивчення та дослідження паралельних методів вирішення складних обчислювальних задач. Призначенням системи є проведення обчислювальних експериментів, з ціллю вивчення паралельних алгоритмів вирішення типових задач, що потребують значних обчислювальних ресурсів.

Можливості системи.

- Моделювання обчислювальної системи. Можливість вибору топології (лінійка, кільце, решітка, гіперкуб, повний граф).
- Постановка задачі. Можливий вибір методу постановки (сортування Шелла, бульбашкове сортування, швидке сортування).
- Виконування експеримент.
- Аналіз результатів обчислень.

Набір існуючих у системі засобів візуалізації дозволяє:

- Вивчити ефективність використання паралельних методів на різних паралельних системах ;
- Зробити висновки о масштабованості алгоритмів;
- Обчислити можливе прискорення процесу паралельних обчислень.

Області застосування.

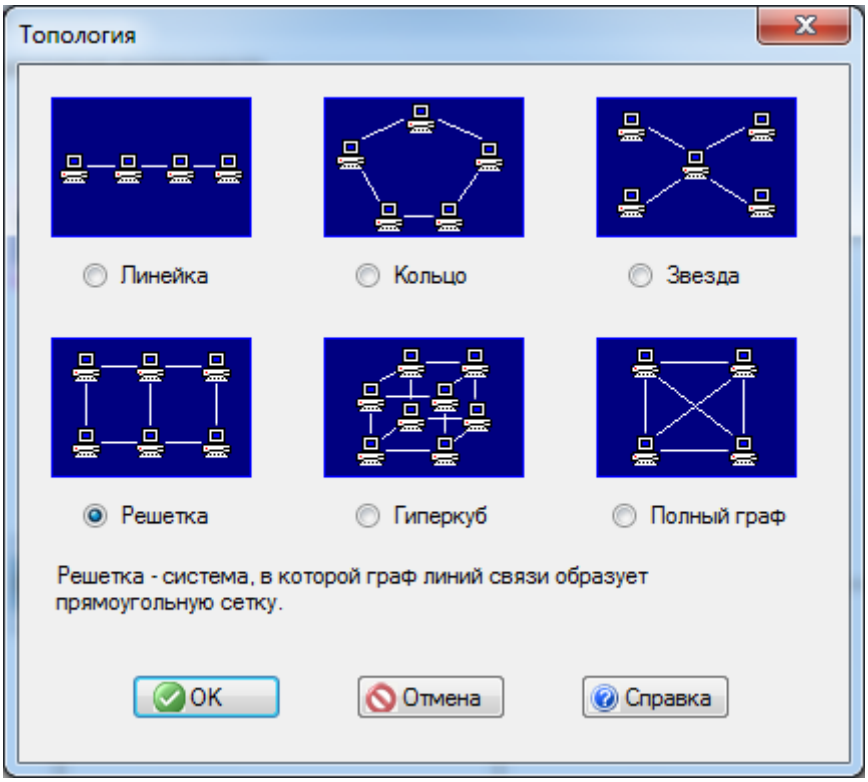
- навчальне застосування;
- наукове використання;
- прикладне застосування.

Проведення експериментів, можуть провадитися:

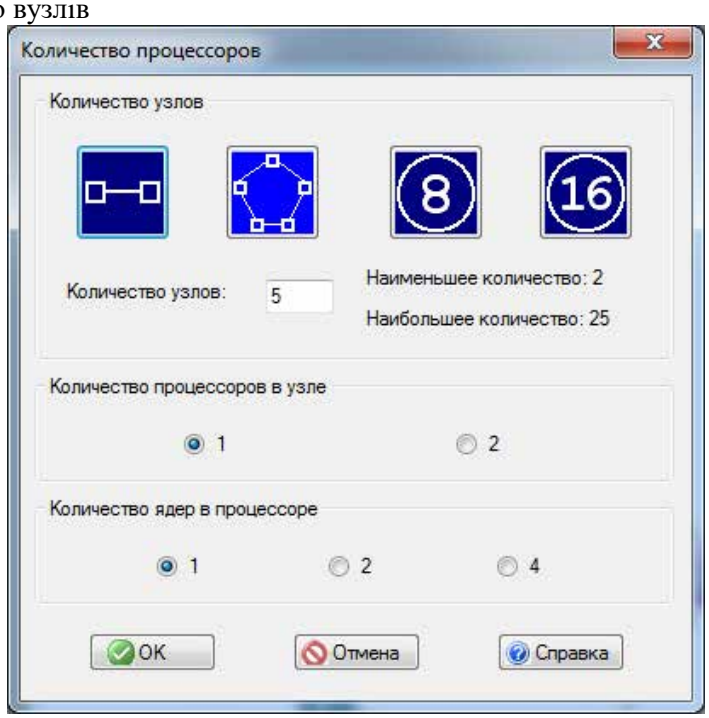
- На одному комп’ютері, де є бібліотека передачі повідомлень MPI (багатопотокове виконання експерименту).
- На реальній багатопотоковій кластерній обчислювальній системі.
- В режимі вилученого доступу до обчислювального кластера.

Хід роботи

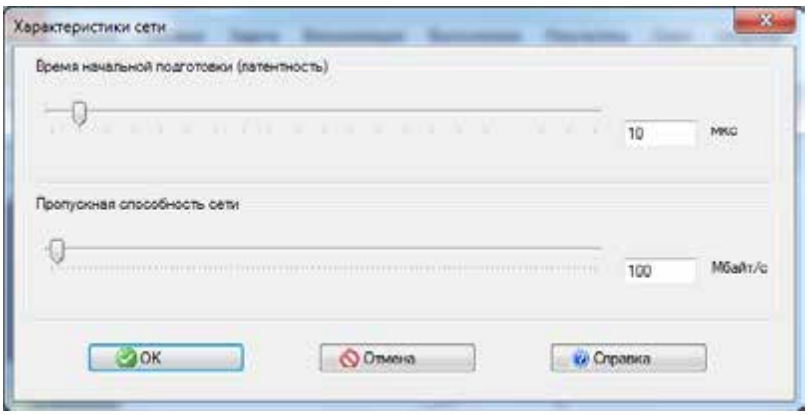
1. Обирається тип топології



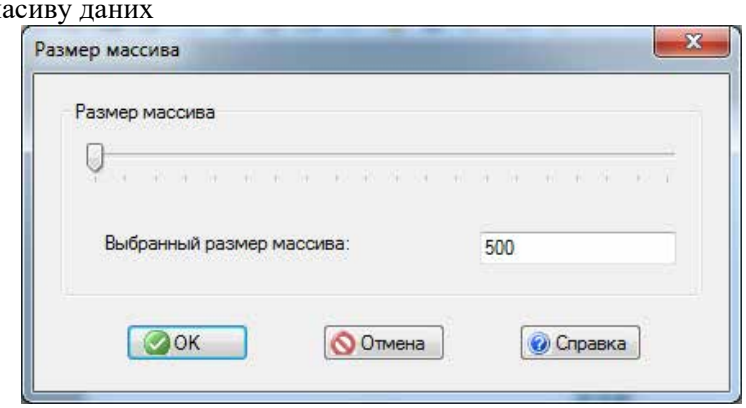
2. Задається число вузлів



3. Задається характеристика мережі



4. Задається розмір масиву даних



Завдання на самостійну роботу

5. Вибрати топологію комп'ютерної системи, залежно від варіанта.
6. Залежно від варіанта, задати кількість процесорів.
7. Визначте продуктивність процесора.
8. Визначте характеристики комунікаційного середовища.
9. Визначте спосіб комунікації
10. З меню вибору, однієї з наявних у системі завдань, виберіть завдання сортування даних
11. З меню визначення обсягу вихідних даних, залежно від варіанта, виберіть розмір масиву.
12. З меню вибору методів рішення завдання, виберіть метод бульбашкове сортування.
13. Виконайте експеримент.
14. Занесіть до протоколу часові характеристики і табличні данні підсумків експериментів.
15. Виберіть демонстрацію роботи процесора. Створіть таблицю результатів. Поясніть результат розв'язуваного завдання, та залежності прискорення від пропускної здібності, латентності, працездатності, кількості процесорів, та розміру масива у графіках
16. Збільшить по черзі один з параметрів на 20% і повторіть пп.1-11.
17. Збільшить по черзі один з параметрів на 20% і повторіть пп.1-11.
18. Зробіть загальний висновок про вплив параметрів на паралельні обчислення комп'ютерної системи.

Таблиця 1. Варіанти виконання лабораторної роботи

№ вар.	Топологія	Число вузлів	Пропускна здатність мережі Мбіт/сек.	Латентність мкс	Розмір масиву
1	Лінійка	10	100	80	1500
2	Кільце	20	10	20	2000

№ вар.	Топологія	Число вузлів	Пропускна здатність мережі Мбіт/сек.	Латентність мкс	Розмір масиву
3	Решітка	9	100	60	500
4	Гіперкуб	8	100	40	1500
5	Повний граф	20	1000	70	3000
6	Гіперкуб	4	1000	50	500
7	Решітка	16	100	30	1500
8	Лінійка	8	1000	40	2000
9	Повний граф	5	100	60	1500
10	Гіперкуб	16	1000	90	500
11	Повний граф	10	100	30	1500
12	Лінійка	4	10	20	500
13	Кільце	12	100	20	2000
14	Решітка	8	1000	90	3000
15	Гіперкуб	16	10	10	3000
16	Повний граф	12	1000	70	2000
17	Лінійка	12	1000	20	1500
18	Кільце	18	100	30	900
19	Решітка	12	10	10	500
20	Гіперкуб	8	100	40	1000
21	Повний граф	6	1000	90	2000
22	Лінійка	14	10	20	2000
23	Кільце	17	100	80	500
24	Решітка	20	1000	60	1000
25	Гіперкуб	19	10	40	2000

Контрольні запитання:

1. Пояснить назначення системи «Paralab».
2. Приведіть свій приклад області застосування системи «Paralab».
3. Яким чином проводяться експерименти, в системі «Paralab»?
4. Які топології можливо використовувати у експерименті?
5. Поясніть поняття терміну «латентність».
6. Поясніть результати вашого експерименту.
7. Як залежить прискорення від зміни кількості процесорів?
8. Яка наукова значність системи «Paralab»?

Лабораторна робота №7. Вивчення кластерних структур

Мета роботи: розуміти, що таке кластер, знати, які бувають типи кластерів і як вони застосовуються, розбиратися в кластерних технологіях і продуктах для їх реалізації.

Теоретичні відомості

Кластерна архітектура

Кластер є два або більше комп'ютерів, часто званих вузлами, об'єднуваних за допомогою мережевих технологій на базі шинної архітектури або комутатора, що представляє перед користувачами як єдиний інформаційно-обчислювальний ресурс. Вузлами кластера, можуть бути вибрані сервери, робочі станції і звичайні персональні комп'ютери.

Перевага кластеризації для підвищення працездатності виявляється, у разі збою якого-небудь вузла: при цьому інший вузол кластера може узяти на себе навантаження несправного вузла, і користувачі не відмітять переривання в доступі. Можливості масштабованості кластерів дозволяють багато разів збільшувати продуктивність додатків для більшого числа користувачів. технологій Fast і Gigabit Ethernet, Myrinet на базі шинної архітектури або комутатора. Такі суперкомп'ютерні системи є найдешевшими, оскільки збираються на базі стандартних комплектувальних елементів «Off the shelf», процесорів, комутаторів, дисків і зовнішніх пристроїв.

Кластеризація може бути здійснена на різних рівнях комп'ютерної системи, включаючи апаратне забезпечення, операційні системи, програми-утиліти, системи управління і додатку. Чим більше рівнів системи об'єднано кластерною технологією, тим вище надійність, масштабованість і керованість кластера.

Типи кластерів

Умовне ділення на класи запропоноване Язеком Радаєвським і Дугласом Едлайном:

Клас I. Клас машин будується цілком із стандартних деталей, комп'ютерних компонентів, перевагою, є низька ціна, просте обслуговування, доступність апаратних компонентів з різних джерел.

Клас II. Система має ексклюзивні або не широко поширені деталі. Цим можна досягти дуже хорошої продуктивності, але при вищій вартості.

Кластери можуть існувати в різних конфігураціях. Типами кластерів, що
найбільш вживаються, є:

•Системи високої надійності.

•Системи для високопродуктивних обчислень.

•Багатопотокові системи.

Межі між цими типами кластерів до деякої міри розмиті, і часто існуючий кластер може мати такі властивості або функції, які виходять за рамки перерахованих типів. Більш того, при конфігурації великого кластера, використовуваного як система загального призначення, доводиться виділяти блоки, що виконують всі перераховані функції.

Кластери для високопродуктивних обчислень призначені для паралельних розрахунків. Ці кластери зазвичай зібрані з великого числа комп'ютерів. Розробка таких кластерів є складним процесом, що вимагає на кожному кроці акуратних узгоджень таких питань як інсталяція, експлуатація і одночасне управління великим числом комп'ютерів, технічні вимоги паралельного і високопродуктивного доступу до одного і тому ж системного файлу або файлів і міжпроцесорний зв'язок між вузлами і координація роботи в паралельному режимі. Ці проблеми найпростіше вирішуються при забезпеченні єдиного образу операційної системи для всього кластера. Проте реалізувати подібну схему вдається далеко не завжди і зазвичай вона зазвичай застосовується лише для не дуже великих систем.

Оцінка трудомісткості операцій передачі даних для кластерних систем

Для кластерних обчислювальних систем одним з широко вживаних способів побудови комунікаційного середовища є використання концентраторів або комутаторів для об'єднання процесорних вузлів кластера в єдину обчислювальну мережу. У цих випадках топологія мережі кластера є повний граф, в якому, є певні обмеження на одночасність виконання комунікаційних операцій. Так, при використанні концентраторів передача даних в кожен теперішній момент часу може виконуватися тільки між двома процесорними вузлами; перемикачі можуть забезпечувати взаємодію декількох непересічних пар процесорів.

Інше часто вживане рішення при створенні кластерів полягає у використанні методу передачі пакетів, що реалізовується, як правило, на основі протоколу TCP/IP, як основний спосіб виконання комунікаційних операцій.

1. Вибираючи для подальшого аналізу кластери даного поширеного типу топології у вигляді повного графа, пакетний спосіб передачі повідомлень, трудомісткість операції комунікації між двома процесорними вузлами може бути оцінена відповідно до виразу (модель А):

$$t_{nd}(m) = t_n + mt_k + t_c \quad (1)$$

де t_n – це час початкової підготовки, який характеризує тривалість підготовки повідомлень для передачі, пошуку маршруту в мережі тощо;

t_k – час передачі одного слова даних по одному каналу передачі даних, тривалість подібної передачі визначається смугою пропускання комунікаційних каналів в мережі.

t_c – час передачі службових даних, між двома сусідніми процесами, тобто для процесорів, між якими є фізичний канал передачі даних; до службових даних може відноситися заголовок повідомлення, блок даних для виявлення помилок тощо.

t_{nd} – час пересилки даних для методу передачі повідомлення розміром m по маршруту довжиною l.

2. Схема побудови тимчасових оцінок може бути уточнена; в рамках нової розширеної моделі, трудомісткість передачі даних між двома процесорами визначається відповідно до наступних виразів (модель В):

$$t_{nd} = \begin{cases} t_{нач0} + t_{нач1} + (mV_c)*t_k, \; n = 1 \\ t_{нач0}\,(V_{max} - V_c) * t_{нач1} + (m + V_c * n)*t_k, \; n > 1, \end{cases} \quad (2)$$

де n – кількість пакетів, на яку розбивається передане повідомлення; величина V_{max} – визначає максимальний розмір пакету, який може бути доставлений в мережі, припустимо, що обчислювання здійснюється для операційної мережі MS Windows в мережі Fast Ethernet, тоді $V_{max} = 1500$ байт;

V_c – об’єм службових даних в кожному з пакетів, що пересилаються, для протоколу TCP/IP, ОС Windows і мережі Fast Ethernet $V_c = 78$ байт;

$t_{нач0}$ – характеризує апаратну складову латентності і залежить від параметрів використовуваного мережевого устаткування; значення $t_{нач1}$ – задає час підготовки одного байту даних для передачі по мережі.

3. Для практичного застосування перерахованих моделей необхідно виконати оцінку значень параметрів використовуваних співвідношень. В цьому відношенні корисним може опинитися використання і простіших способів обчислення тимчасових витрат на передачу даних – серед відомих схем подібного вигляду підхід, в якому трудомісткість операції комунікації між двома процесорними вузлами кластера оцінюється відповідно до виразу 3 (модель С):

$$t_{nd}\,(m) = t_н + m / R \quad (3)$$

де $t_н$ – величина латентності;

R – пропускна здатність мережі передачі даних.

Хід роботи

1. Залежно від варіанту завдання в таблиці 1, необхідно обчислити похибку теоретичної оцінки часу виконання операції передачі даних за формулами (1-3).
2. Оцінка подібного вигляду виходить із співвідношення для методу передачі пакетів при одиничній довжині шляху передачі даних, тобто l = 1.
3. В рамках даної моделі час підготовки даних t_n передбачається постійним, не залежним від об’єму передачі даних, і вказано в таблиці 1.
4. Дайте зрівняльну характеристику трьох моделей по критерію високої точності, результат навести у відсотках.

Таблиця 1. Варіанти для виконання лабораторної роботи

№ варіанту	Об'єм повідомлень в байтах	Час передачі даних в мкс	Латентність, мкс
1	32	172,0269	20
2	64	172,2211	20
3	128	173,1494	20
4	256	203,7902	30
5	1024	334,4392	30
6	2048	481,5397	40
7	4096	770,6155	40
8	256	770,6155	40
9	512	203,7902	60
10	1024	172,0269	60
11	2048	172,0269	60
12	512	242,6845	30

Таблиця 2. Таблиця (приклад) для внесення результату

№ варіанту	Похибка теоретичної оцінки часу виконання операції передачі даних		
	Модель A	Модель B	Модель C
7	22,33%	5,05%	23,73%

Контрольні запитання

1. Для чого потрібні кластери?
2. Які класи кластерних систем визнаєте? Охарактеризуйте їх.
3. Що таке масштабованість кластерів? Опишіть їх можливості.
4. Яке програмне забезпечення використовується, при створення кластера?
5. Наведіть приклад існуючого кластера. Для яких задач використовується, наведений вами приклад кластера?
6. Чим обґрунтована стійкість кластерів до відмов?
7. Яким чином можливо оцінити трудомісткість операцій передачі даних для кластерних систем?
8. Яке комунікаційне обладнання використовується, при створенні кластера?
9. Які переваги кластерної обробки задач, ви можете привести?
10. Чи можливо, для забезпечення необхідної працездатності з'єднати в єдиний кластер, робочі станції з суперкомп'ютерами?

СПИСОК ЛІТЕРАТУРИ

Основна.

1. Васюхин М.И. , С.О.Горбатюк, М.М.Касім, В.Г.Шелестовський Комп'ютерні системи. Навчальний посібник.– К.: ЦП «Компринт», 2017.– 270с.

Додаткова

1. Воеводин В.В., Воеводин Вл. В. Параллельные вычисления. – СПб.: БХВ-Питербург, 2004. – 608с.
2. Гергель В.П. Теория и практика параллельных вычислений. Учебное пособие. - М.: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2007.
3. Головки Б.А. Параллельные вычислительные системы. – М.: Наука, 1980 – 520с.
4. Глушаков С.В., А.В. Коваль, С.В. Смирнов. Язык программирования С++. – Харьков: Фолио. – 2002.
4. Корнеев В.В. Параллельные вычислительные системы. – М.: «Нолидж», 1999., - 320с.
5. Корнеев В. В., Современные микропроцессоры, издательство М.: "Нолидж", БХВ , 2003
6. Мельник А.О. Архітектура комп'ютера.Наукове видання. – Луцьк.: Волинська обласна друкарня, 2008. – 470 с.
7. Немнюгин С., Стесик О. Параллельное программирование для многопроцессорных вычислительных систем СПб:, БХВ-Петербург, 2002
8. Нечаев Ю.И. Искусственный интеллект: концепции и приложения. Спб: Изд.центр СПбГМТУ, 2002.
9. Организация ЭВМ. 5-е изд../ К. Хамахер, З. Вранешич.,С. Заки. . – Спб.: Питер; Киев: изд. группа BXV, 2003. – 848с.
10. Смирнов А.Д. Архитектура вычислительных систем: Учебное пособие для вузов. - М: Наука, 1990.
11. Специализированные процессоры для высокопроизводительной обработки данных. / О.Л. Бадман, Н.Н. Миренков и др. Новосибирск: Наука, 1988.
12. СуперЭВМ. Аппаратная и программная организация. Под ред. С. Фернбаха. - М.: Радио и связь, 1991.
13. Таненбаум, Э.Архитектура компьютера. СПб, Из-во «Питер», 2002.
14. Тарасенко В.П. Надійність комп'ютерних систем / В.П. Тарасенко, А.Ю. Маламан, Ю.П. Черніченко, В.І. Корнійчук. – К., 2007. – 256 с.
15. Хамахер К., Вранешич З., Заки С., Организация ЭВМ. СПб, Из-во «Питер», 2003.

Повний курс дисципліни «Комп'ютерні системи » наведений
https://elearn.nubip.edu.ua/course/view.php?id=800