МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ
імені адмірала Макарова

І. В. УСТЕНКО, Л. О. ЛАТАНСЬКА, С. А. УСТЕНКО

МЕТОДИЧНІ ВКАЗІВКИ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ З ДИСЦИПЛІНИ «АРХІТЕКТУРА КОМП'ЮТЕРІВ»

Миколаїв
2018

УДК 004.2
ББК 32.973.26-02
У 79

Автор:

І. В. Устенко, кандидат технічних наук, доцент, доцент кафедри програмного забезпечення автоматизованих систем Національного університету кораблебудування імені адмірала Макарова;

Л. О. Латанська, кандидат фізико-математичних наук, доцент, доцент кафедри програмного забезпечення автоматизованих систем Національного університету кораблебудування імені адмірала Макарова;

С. А. Устенко, доктор технічних наук, доцент, завідувач кафедри комп'ютерної інженерії Миколаївського національного університету імені В. О. Сухомлинського.

Рецензент:

І. І. Коваленко, доктор технічних наук, професор, професор кафедри інженерії програмного забезпечення Чорноморського національного університету імені Петра Могили.

Затверджено та рекомендовано до друку рішенням методичної ради Національного університету кораблебудування.

Устенко І. В., Латанська Л. О., Устенко С. А.

У79 Методичні вказівки до виконання лабораторних робіт з дисципліни «Архітектура комп’ютера» / І. В. Устенко, Л. О. Латанська, С. А. Устенко. – Миколаїв : НУК імені адмірала Макарова, 2018. – 46 с.

Методичні вказівки містять теоретичний матеріал, завдання для лабораторних робіт з прикладами виконання, питання для самоконтролю, список рекомендованої літератури, необхідної для вивчення курсу «Архітектура комп’ютера». Рекомендовано для студентів спеціальностей напряму 12 – «Інформаційні технології».

УДК 004.2
ББК 32.973.26-02

ВИМОГИ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ З ДИСЦИПЛІНИ «АРХІТЕКТУРА КОМП'ЮТЕРА»

При виконанні лабораторних робіт кожен студент отримує індивідуальне завдання. Номер варіанта відповідає порядковому номеру студента у списку групи. Створювані в рамках лабораторних робіт додатки повинні мати графічний інтерфейс і можуть бути написані будь-якою мовою програмування.

По кожній лабораторній роботі оформлюється звіт, що повинен задовольняти зазначеним далі вимогам, частина з яких обов'язкова для виконання. Також в кожній лабораторній роботі можуть з'являтися додаткові вимоги до змісту звіту – вони наводяться в описі роботи.

З дисципліни «Архітектура комп'ютера» передбачена бальна система оцінювання успішності студентів. Відповідно до неї за кожну виконану лабораторну роботу студент отримує бали за факт, своєчасність і якість виконання та захисту лабораторної роботи. Якість виконання роботи оцінюється за кількома критеріями, частина з яких загальна для всіх лабораторних робіт (ці критерії наводяться далі), а частина – специфічна для конкретної лабораторної роботи (ці критерії наводяться в описі кожної роботи). Виконання необов'язкових частин дозволяє набирати бали за якість захисту лабораторної роботи.

Бали за своєчасність виконання та захисту лабораторної роботи нараховують за принципом "чим раніше здана робота, тим більше балів набрано". Прийом викладачем кожної лабораторної роботи включає три етапи (в зазначеному порядку):

1) демонстрація роботи програми на комп'ютері (перевіряється правильність роботи програми);

2) пояснення вмісту звіту (перевіряється розуміння логіки роботи програми, знання теоретичного матеріалу, на основі якого написана програма, відповідність звіту встановленим вимогам);

3) захист лабораторної роботи (відповіді на питання).

Загальні вимоги до оформлення звіту

Звіт повинен містити:

– титульний лист (обов'язково);
– опис завдання на лабораторну роботу і варіанти (обов'язково);
– всі проміжні обчислення (якщо є відповідні завдання);
– опис розробленої програми;
– програмний код, написаний безпосередньо студентом (обов'язково);
– результати виконання всіх завдань (обов'язково);
– тестування програми;
– висновки щодо виконання лабораторної роботи.

Всі розділи звіту повинні бути витримані в єдиному стилі оформлення.

Робота №1

БАЗОВІ СИСТЕМИ ЧИСЛЕННЯ

Мета роботи: закріплення знань про системи числення та перетворення між ними; оволодіння навичками складання програм з перетворення чисел в різні позиційні системи числення.

Стислі теоретичні відомості

В якості стислих теоретичних відомостей рекомендується використовувати главу 1 із навчального посібника [2].

Завдання

1. Розробіть мовою програмування C/C++ функції перетворення числа з однієї системи числення в іншу відповідно до варіанту (табл. 1). Числа задавати та отримувати у вигляді рядка.

2. Розробіть функції для додавання та віднімання чисел в даних системах числення.

Таблиця 1

№	Схема перетворення	№	Схема перетворення	№	Схема перетворення
1	$2 \Rightarrow 3$	11	$5 \Rightarrow 2$	21	$10 \Rightarrow 2$
2	$2 \Rightarrow 5$	12	$5 \Rightarrow 3$	22	$10 \Rightarrow 3$
3	$2 \Rightarrow 8$	13	$5 \Rightarrow 8$	23	$10 \Rightarrow 5$
4	$2 \Rightarrow 10$	14	$5 \Rightarrow 10$	24	$10 \Rightarrow 8$
5	$2 \Rightarrow 16$	15	$5 \Rightarrow 16$	25	$10 \Rightarrow 16$
6	$3 \Rightarrow 2$	16	$8 \Rightarrow 2$	26	$16 \Rightarrow 2$
7	$3 \Rightarrow 5$	17	$8 \Rightarrow 3$	27	$16 \Rightarrow 3$
8	$3 \Rightarrow 8$	18	$8 \Rightarrow 5$	28	$16 \Rightarrow 5$
9	$3 \Rightarrow 10$	19	$8 \Rightarrow 10$	29	$16 \Rightarrow 8$
10	$3 \Rightarrow 16$	20	$8 \Rightarrow 16$	30	$16 \Rightarrow 10$

Контрольні питання

1. Які типи систем числення Ви знаєте?
2. Які типи даних Ви знаєте? Скільки байтів кожен з них займає в пам’яті комп’ютера?
3. Назвіть основні принципи концепції Джона фон Неймана.

Робота №2

ОЗНАЙОМЛЕННЯ З КОМПІЛЯТОРОМ NASM

Мета роботи: навчитися користуватись компілятором NASM для створення програм мовою Assembler.

Стислі теоретичні відомості

Для компіляції програм можна використовувати компілятор NASM (Netwide Assembler), який вільно розповсюджується (разом зі своїми вихідними кодами) за ліцензією LGPL. Його можна знайти на сайті `https://www.nasm.us/`.

Синтаксис компілятора подібний до синтаксису комерційних компіляторів MASM (Microsoft Assembler) і TASM (Turbo Assembler від Borland). NASM легко переноситься на різні операційні системи в межах x86-сумісних процесорів.

Формат вихідного файлу визначається за допомогою ключа командного рядка `-f`. Деякі формати розширюють синтаксис певних інструкцій, а також додають свої інструкції.

Створення виконуваного файлу складається з двох етапів. Перший – це компіляція (трансляція) початкового коду програми в деякий об'єктний формат. Об'єктний формат містить машинний код програми, але символи (змінні і інші ідентифікатори) в об'єктному файлі поки не прив'язані до адрес пам'яті.

На другому етапі, який називається компоновкою або лінковкою (linking), з одного або декількох об'єктних файлів створюється виконуваний файл. Процедура компоновки полягає в тому, що компонувальник зв'язує символи, визначені в основній програмі, з символами, які визначені в її модулях (враховуються директиви `EXTERN` і `GLOBAL`), після чого кожному символу призначається остаточна адреса пам'яті або забезпечується його динамічне обчислення.

Формат `bin` не є об'єктним форматом. Він містить результат перекладу команд асемблера в машинні коди. Цей формат дозволяє оголошувати секції (коду, даних, неініціалізованих даних тощо). Після імені секції може стояти директива `ALIGN`, що вимагає розташовувати цю секцію за адресами, кратними вказаному числу. Наприклад, наступна директива задає вирівнювання секції коду за адресами, кратними 16:

```
section .text align=16
```

Формат bin також можна використовувати для створення файлів, що виконуються під DOS (.com і .sys) або для завантажувачів. Значення за умовчанням директиви BITS рівне 16. У коді, призначеному для виведення у форматі bin, може бути присутньою директива ORG.

```
nasm -f bin hello.asm
```

OMF (Object Module Format) – це старий формат, розроблений корпорацією Intel, в якому до цих пір створює об'єктний код Turbo Assembler. Компілятори MASM і NASM теж підтримують цей формат. Файл у форматі OMF має розширення .obj, і сам формат часто називається також OBJ. Файли з розширенням .obj можуть бути скомпоновані у виконуваний файл.

Не дивлячись на те, що формат OBJ був спочатку розроблений для 16-бітового режиму, компілятор NASM також підтримує його 32-бітове розширення. Тому NASM можна використовувати разом з 32-бітовими компіляторами від Borland, які теж підтримують цей розширений 32-бітовий формат, а не інший об'єктний формат, підтримуваний Microsoft. Формат OBJ розширює синтаксис деяких директив, зокрема, SEGMENT (SECTION). Крім того, формат OBJ додає нову директиву IMPORT, яка дозволяє імпортувати вказаний файл з DLL, їй потрібно передати два параметри – ім'я ідентифікатора і DLL. У кожного OBJ-файлу повинна бути своя точка входу (як мінімум одна). Точка входу визначає позицію, на яку буде передано управління після завантаження програми в пам'ять. Точка входу (entry point) позначена спеціальним ідентифікатором (або міткою) ..start:. Щоб отримати файл у форматі OBJ, потрібно викликати компілятор з ключем командного рядка -f obj.

Для компоновки програми за допомогою Microsoft Visual C++ використовується 32-бітовий формат Win32. Цей формат повинен бути сумісний з форматом COFF (Common Object File Format), але насправді такої сумісності немає. Якщо програма компонуватиметься компонувальником, заснованим на форматі COFF, то краще компілювати асемблерний код в об'єктний файл формату COFF.

Щоб отримати файл у форматі Win32, потрібно викликати компілятор з ключем командного рядка -f Win32.

Формат виконуваного файлу `a.out` (Assembler and link editor OUTput files) переважно використовується в Linux. Розширена версія цього формату `a.outb` використовується групою BSD-сумісних операційних систем (NETBSD, FREEBSD і OPENBSD). NASM може генерувати файли обох форматів, якщо вказати ключ `-f aout` для Linux або `-f aoutb` для BSD.

Формат `COFF` (Common Object File Format) є істотно вдосконаленою і допрацьованою версією формату `a.out`. Він поширений як в світі UNIX, так і в Windows, і підтримується деякими вільними (не комерційними) середовищами розробки, наприклад, DJGPP. Використовується для розробки додатків на C і C++. Якщо компілятор викликати з параметром `-f coff`, на виході буде отримано об'єктний файл у форматі `COFF`.

Формат `ELF` (Executable and Linkable Format) – це формат як об'єктного, так і виконуваного файлу. Він застосовується в багатьох UNIX-подібних системах. Його можна використовувати, щоб відкомпілювати програму для операційних систем сімейства Linux, причому не тільки з мови Assembler. Розширення об'єктного файлу за умовчанням `.o`. Для отримання файлу в цьому форматі компілятор потрібно запускати з ключем `–f elf`.

Деякі вихідні формати дозволяють зберігати разом з машинними кодами символи початкового тексту програми. Така можливість дуже корисна при відладці, вона дозволяє відладчику відображати адреси пам'яті у вигляді імен змінних, міток тощо. Символьна інформація значно збільшує розмір вихідного файлу, тому після відладки програму наново компілюють вже без неї. Для додавання символічної інформації потрібно запустити NASM з ключем `-g`. Цей ключ допустимий тільки для форматів `OBJ` і `ELF`.

Контрольні питання

1. Для якої операційної системи призначений компілятор NASM мови програмування Assembler?
2. Що таке компіляція та компоновка?
3. За допомогою якого ключа в компіляторі NASM визначається формат вихідного файлу?
4. Який ключ дозволяє додати до машинних кодів символічну інформацію?

Робота №3

ПРЕДСТАВЛЕННЯ ДАНИХ. АРИФМЕТИКО-ЛОГІЧНІ ОПЕРАЦІЇ. ПОДАННЯ ЦІЛИХ ЧИСЕЛ В КОМП'ЮТЕРІ

Мета роботи: вивчення архітектури x86, структури найпростішої програми, ознайомлення з системою арифметико-логічних команд процесора, організація обчислень мовою Assembler; закріплення знань про подання додатних і від'ємних чисел в комп'ютері; оволодіння навичками написання програм обробки знакових та без знакових даних.

Стислі теоретичні відомості

При виконанні арифметико-логічних команд найбільшої швидкодії та зручності програмування можна досягти за рахунок використання акумулятора (регістра `AX`) для зберігання проміжних результатів. Наприклад, обчислення виразу `D = A + B – C` можна записати так:

```
MOV   AX, A
ADD   AX, B
SUB   AX, C
MOV   D, AX
```

При реалізації операцій ділення необхідно пам'ятати про те, що якщо результат не поміщається цілком в регістрі-приймачі (наприклад, при діленні `8B00h` в регістрі `AX` на 3 в регістрі `BL`), виникає помилка "ділення на нуль" і програма аварійно завершується. Щоб уникнути подібних ситуацій, слід збільшувати розмірність чисел. У даному прикладі слід використовувати команду `CWD` і ділити на `BX`.

Ділення та множення на степінь двійки слід виконувати за допомогою команд зміщення. Ці команди найбільш ефективні при їх виконанні для регістра `AX`.

Завдання

1. Сформувати числові значення, визначити мінімальний формат представлення початкових даних. Значення початкових даних, які повинні зберігатися в сегменті даних, визначаються такими виразами:

$$X_1 = N*(-1)^N;\ X_2 = (-1)^N + 1*(M*N);$$

$$X_3 = (-1)^N + 2*(M*N + M);\ X_4 = (-1)^N + 3*M,$$

де N – номер варіанта, M – номер групи.

2. По заданому алгоритму (рис. 1) скласти та виконати програму роботи з даними.

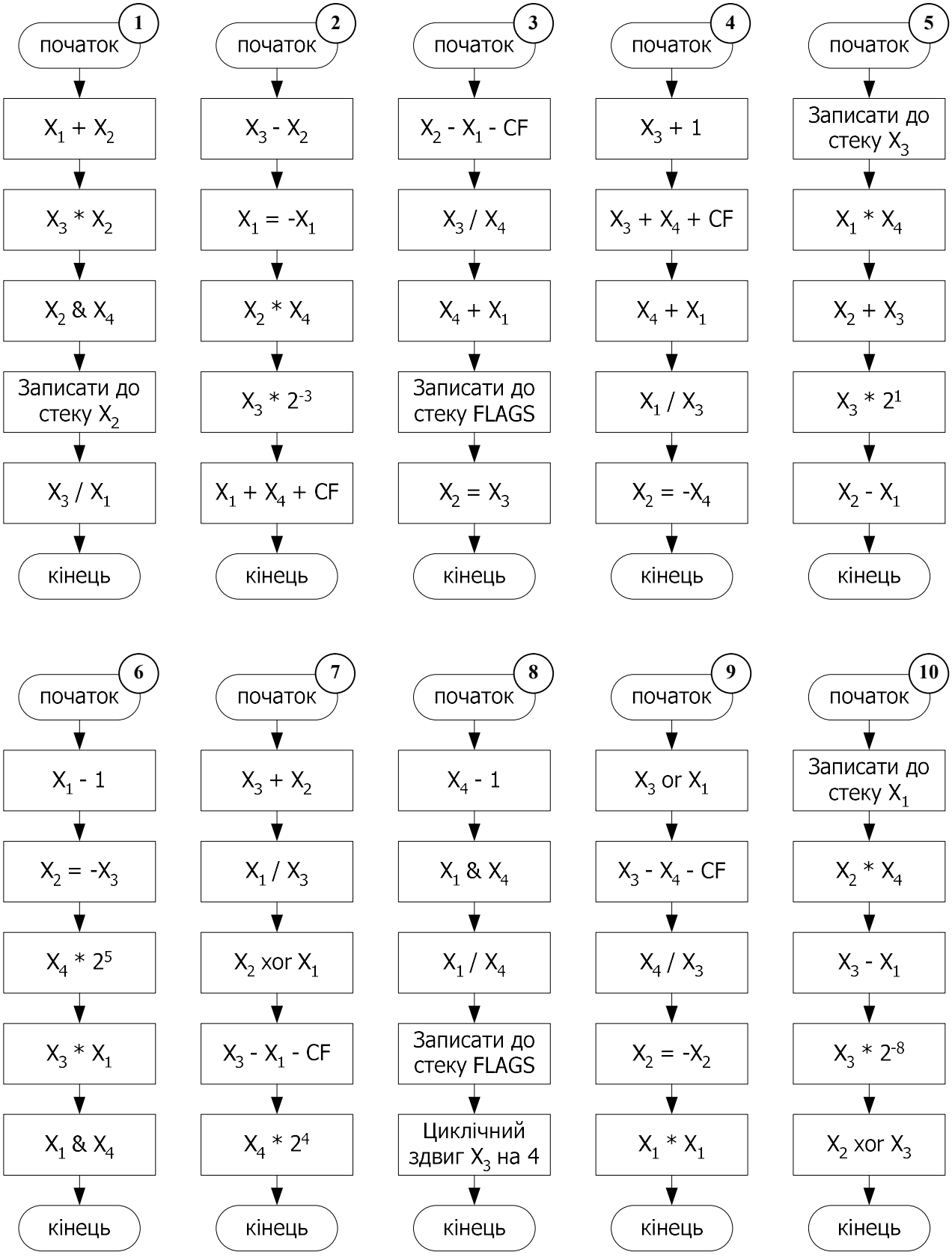


Рисунок 1

3. Показати внутрішнє (машинне) уявлення чисел X_1 і X_2 відповідно до діапазону в знакових та беззнакових форматах типів даних: `char` (8 біт), `unsigned char` (8 біт), `short int` (16 біт), `unsigned short int` (16 біт), `int` (32 біта), `unsigned int` (32 біта).

$$X_1 = N - M; X_2 = N + M.$$

де N – номер варіанта, M – номер групи.

Приклад виконання завдання

Реалізуйте мовою Assembler наступну програму:

```
int A = 015;
char B;
B = A % 4;
```

Розв'язання:

Оскільки змінна `A` має тип `int`, то мовою Assembler вона буде описана як `dw`, а змінна `B` – `db`. Змінна `A` записана в системі числення з основою 8. Для того щоб отримати залишок від ділення, слід скористатись командою `div`. Програма, що реалізує наведену програму мовою С, наведена нижче:

```
Sseg    SEGMENT STACK 'stack'
        db 256 dup(?)
Sseg    ENDS
Dseg    SEGMENT 'data'
A       DW 13
B       DB ?
Dseg    ENDS
Cseg    SEGMENT 'code'
        ASSUME CS:Cseg, DS:Dseg, SS:Sseg
Main    PROC
        mov ax, Dseg
        mov ds, ax
        xor ax, ax
        mov ax, A
        mov bh, 4
        div bh
```

```
        mov B, ah
        mov ax, 4c00h
        int 21h
Main    ENDP
Cseg    ENDS
END     Main
```

Для перегляду результатів можна завантажити в Turbo Debugger, отриманий після компіляції та компонування, exe-файл з розробленою програмою.

Контрольні питання

1. З яким регістром процесор працює найшвидше?
2. Яким чином помножити швидко число на 8?
3. За допомогою чого можна переглянути результати програми?

Робота №4

РЕАЛІЗАЦІЯ БУЛЕВИХ ФУНКЦІЙ ЗА ДОПОМОГОЮ ЛОГІЧНИХ ВЕНТИЛІВ

Мета роботи: закріпити знання про застосування логічних вентилів для практичної реалізації логічних функцій, що забезпечують роботу пристрою у відповідності до технічного завдання.

Стислі теоретичні відомості

При всій складності цифрових пристроїв, вони засновані на принципі багаторазового повторення простих базових схем. Зв'язок між схемами будуються на основі формальних методів. Інструментом такої побудови служить булева алгебра, що у цифровій техніці називається також алгеброю логіки.

Для реалізації булевих операцій над двійковими сигналами, представленими у вигляді рівнів напруги, використовують спеціальні електронні схеми, що називаються логічними вентилями.

Основне призначення вентилів: для практичної реалізації логічних функцій, що забезпечують роботу пристрою у відповідності до технічного завдання. Якщо можна переформулювати технічне завдання в термінах логіки висловлювань, то не важко буде записати вираз, що описує логіку роботи розроблювального пристрою.

Основи алгебри логіки

Математичним апаратом сучасних обчислювальних систем є булева алгебра, названа так на честь шотландського математика Джона Буля. В алгебрі Буля безліч змінних складається із двох елементів: "0" і "1", які одержали назву логічного нуля і логічної одиниці.

Базові комбінаційні цифрові елементи

В основі побудови обчислювальних систем перебувають три базових елементи: логічне множення (кон'юнкція); логічне додавання (диз'юнкція); логічне заперечення (інверсія).

Логічне множення (кон'юнкція) або логічна операція "І" – результатом функції буде логічна одиниця тільки в тому випадку, якщо значення всіх аргументів дорівнює логічній одиниці.

У формулах (виразах) логічних операцій логічне множення може позначатися одним з наступних символів: & (амперсенд), ×, •, ∧.

Логічне додавання (диз'юнкція) або логічна операція "АБО". Логічною сумою вхідних змінних X_i називається двійкова змінна Y, яка приймає нульове значення тільки тоді, коли всі доданки одночасно дорівнюють нулю.

Аналітичний опис логічної функції диз'юнкції: $Y = X_1 + X_2 = X_1 \vee X_2$.

Логічне заперечення (інверсія) або логічний елемент "НІ" – це логічна функція, що кожному аргументу ставить у відповідність нове значення, що заперечує вихідне.

Аналітичний опис заперечення "НІ": $Y = \overline{X_1}$.

Похідні логічні елементи

Крім трьох базових елементів у цифровій техніці широко застосовуються ще чотири логічних елементи, що одержали назви похідних: "АБО, що виключає", заперечення кон'юнкції (штрих Шеффера), заперечення диз'юнкції (стрілка Пірса).

Логічний елемент *штрих Шеффера* складається з послідовно з'єднаних логічних елементів "І" та "НІ" (логічний елемент "І-НІ"). Функція буде приймати значення логічної одиниці в тому випадку, коли хоча б один із вхідних сигналів буде логічним нулем.

Алгебраїчний запис операції Шеффера: $Y = X_1 / X_2 = \overline{X_1 \bullet X_2} = \overline{X_1 \wedge X_2}$.

Логічний елемент *стрілка Пірса* складається з послідовно з'єднаних логічних елементів "АБО" та "НІ" (логічний елемент "АБО-НІ"). Функція буде приймати значення логічної одиниці в тому випадку, коли всі вхідні сигнали будуть логічним нулем.

Алгебраїчний запис стрілки Пірса: $Y = X_1 \downarrow X_2 = \overline{X_1 + X_2} = \overline{X_1 \vee X_2}$.

Логічний елемент "*АБО, що виключає*" (суматор по модулю 2) – функція, що приймає значення логічної одиниці, якщо складові її аргументи мають різне значення.

Алгебраїчний запис суматора по модулю 2:

$$Y = X_1 \oplus X_2 = (\overline{X_1} + X_2) \bullet (X_1 + \overline{X_2}) = (X_1 \bullet X_2) + (\overline{X_1} \bullet \overline{X_2}).$$

Завдання

1. Спроектувати схему, шляхом поєднанням різних вентилів, що реалізує логічний вираз, відповідно до варіанту (табл. 2):

   а) за допомогою ДДНФ;

   б) на основі тільки вентилів "І-НІ" (для парних варіантів) або тільки вентилів "АБО-НІ" (для непарних варіантів).

Таблиця 2

N	Завдання	N	Завдання	N	Завдання
1	$(A\wedge B)\wedge C$	7	$(A\sim B)\vee C$	13	$(A\vee B)\sim C$
2	$(A\vee B)\wedge C$	8	$(A\wedge B)\to C$	14	$(A\to B)\sim C$
3	$(A\to B)\wedge C$	9	$(A\vee B)\to C$	15	$(A\sim B)\sim C$
4	$(A\sim B)\wedge C$	10	$(A\to B)\to C$	16	$(A\wedge B)\vee C$
5	$(A\vee B)\vee C$	11	$(A\sim B)\to C$		
6	$(A\to B)\vee C$	12	$(A\wedge B)\sim C$		

2. Спроектувати схему для заданої логічної операції, відповідно до варіанту (табл. 3), за допомогою:

   а) кон'юнкції і заперечення – для парних варіантів;

   б) диз'юнкції і заперечення – для непарних варіантів.

Таблиця 3

N	Завдання	N	Завдання	N	Завдання
1	$\neg(\neg A\vee B)\wedge(C\sim D)$	7	$(A\to B)\vee(C\to D)$	13	$(A\sim B)\to(C\sim D)$
2	$(A\to B)\wedge(C\to D)$	8	$(A\sim B)\vee(C\sim D)$	14	$(A\vee B)\to(C\to D)$
3	$(A\sim B)\wedge(C\sim D)$	9	$(A\wedge B)\vee\neg(C\to D)$	15	$(A\vee B)\to(C\sim D)$
4	$(A\vee B)\wedge(C\to D)$	10	$(A\wedge B)\vee(C\to D)$	16	$(A\to B)\to(C\sim D)$
5	$(A\vee B)\wedge(C\sim D)$	11	$(A\wedge B)\vee(C\sim D)$		
6	$(A\to B)\wedge(C\sim D)$	12	$(A\to B)\vee(C\sim D)$		

Контрольні питання

1. Що таке логічний вентиль?
2. Що таке тригер?
3. Що таке цифрова схема?
4. Які логічні вентилі існують?
5. Базові комбінаційні цифрові елементи?
6. Логічний елемент заперечення кон'юнкції (штрих Шеффера).
7. Логічний елемент заперечення диз'юнкції (стрілка Пірса).

Робота №5

ТИПОВІ КОМБІНАТОРНІ ЦИФРОВІ СХЕМИ

Мета роботи: вивчення особливостей функціонування та моделювання найбільш поширених видів типових комбінаційних цифрових схем – шифраторів, дешифраторів, мультиплексорів і демультіплексорів, а також, їх синтез і реалізація в системі схемотехнічного моделювання Electronics Workbench.

Стислі теоретичні відомості

Шифратор (CD) перетворює число з десяткової системи числення до двійкового коду при подачі сигналу 1 (CD високого рівня) або сигналу 0 (CD низького рівня) на вхід з номером цього десяткового числа. Шифратор має *m*-входів і *n*-виходів. Максимальне число входів CD:

$$m = 2^n.$$

Якщо використовуються всі входи, то це повний CD, якщо їх частина – неповний. Згідно зі слів [1], шифратори підрозділяються на CD "з 4 в 2", "з 8 в 3", "з 16 в 4" тощо.

Дешифратор (декодер) (DC) – схема, яка отримує на вході *n*-розрядне число і використовує його для того, щоб вибрати (тобто встановити в значення 1) одну з 2^n вихідних ліній. Розрізнюють дешифратори "з 2 в 4", "з 3 в 8", "з 4 в 16" тощо. Якщо використовують всі виходи, то це повний DC, якщо їх частину – неповний.

Мультиплексор представляє собою схему з 2^n входами, одним виходом і *n* лініями управління, які дозволяють вибрати один з входів. Обраний вхід з'єднується з виходом.

Демультиплексор з'єднує єдиний вхідний сигнал з одним з 2^n виходів в залежності від значень сигналів в *n* лініях управління. Якщо бінарне значення ліній управління дорівнює *k*, то вибирається вихід *k*.

Система схемотехнічного моделювання Electronics Workbench (EW)

Досліджувана схема збирається на робочому полі з використанням миші і клавіатури. При побудові та редагуванні схем виконуються наступні операції:

– вибір компонента з бібліотеки компонентів;
– виділення об'єкта;
– переміщення об'єкта;
– копіювання об'єктів;
– видалення об'єктів;
– поєднання компонентів схеми провідниками;
– установка значень компонентів;
– підключення вимірювальних приладів.

Після побудови схеми і підключення приладів, аналіз роботи схеми починається з натискання вимикача в правому верхньому куті вікна програми (при цьому в нижньому лівому кутку екрана показуються моменти схемного часу). Повторне натискання вимикача припиняє роботу схеми. Зробити паузу при роботі схеми можна натисканням клавіші F9 на клавіатурі; повторне натискання F9 відновлює роботу схеми (аналогічного результату можна домогтися натискаючи кнопку Pause, розташовану під вимикачем).

Вибір компонента необхідного для побудови схеми проводиться після вибору поля компонентів, що містить необхідний елемент. Цей елемент захоплюється мишею і переміщається на робоче поле.

Виділення об'єкта. При виборі компонента необхідно клацнути на ньому лівою клавішею миші. При цьому компонент стає червоним. Зняти виділення можна клацанням миші в будь-якій точці робочого поля.

Переміщення об'єкта. Об'єкт виділяють, встановлюють на нього вказівник і, тримаючи ліву кнопку миші, перетягують об'єкт.

Поворот об'єкта. Для цього об'єкт потрібно попередньо виділити, потім натиснути правою кнопкою миші і вибрати необхідну операцію:

– Rotate (поворот на 90 градусів);
– Flip vertical (переворот по вертикалі);
– Flip horizontal (переворот по горизонталі).

Копіювання об'єктів здійснюється командою Copy з меню Edit. Перед копіюванням об'єкт потрібно виділити. При виконанні команди виділений об'-

єкт копіюється в буфер. Для вставки вмісту буфера на робоче поле потрібно вибрати команду Paste з меню Edit

Видалення об'єктів. Виділені об'єкти можна видалити командою Delete.

З'єднання компонентів схеми провідниками. Для з'єднання компонентів провідниками потрібно підвести покажчик миші до контакту компонента (при цьому на контакті з'явиться чорна точка). Натиснувши ліву кнопку миші, треба перемістити її покажчик до контакту компонента, з яким потрібно з'єднатися, і відпустити кнопку миші. Контакти компонентів з'єднуються провідником. Колір провідника можна змінити, якщо двічі клацнути по провіднику мишею і вибрати з вікна, що з'явилося необхідний колір.

Видалення провідника. Якщо з будь-якої причини провідник потрібно видалити, необхідно підвести покажчик миші до контакту компонента (повинна з'явитися чорна точка). Натиснувши ліву кнопку миші, треба перемістити її на порожнє місце робочого поля і відпустити кнопку миші. Провідник зникне.

Установка значень параметрів проводиться в діалоговому вікні властивостей компонента, яке відкривається подвійним клацанням миші по зображенню компонента (закладка Value). Кожному компоненту можна присвоїти ім'я (закладка Label).

Підключення приладів. Для підключення приладу до схеми потрібно мишею перетягнути прилад з панелі інструментів на робоче поле і підключити контакти приладу до досліджуваних точок. Деякі прилади необхідно заземлювати, інакше їх показання будуть невірними. Розширене зображення приладу з'являється при подвійному натисканні по зображенню.

Компоненти EW

Після запуску на екрані з'являються рядок меню і панель компонентів. Панель компонентів складається з піктограм полів компонентів, а поле компонентів – з умовних зображень компонентів.

Клацанням миші на піктограмі компонентів відкривається поле відповідне цій піктограмі. На рис. 2 наведені деякі елементи з полів компонентів, що треба використовавути при виконанні лабораторної роботи.

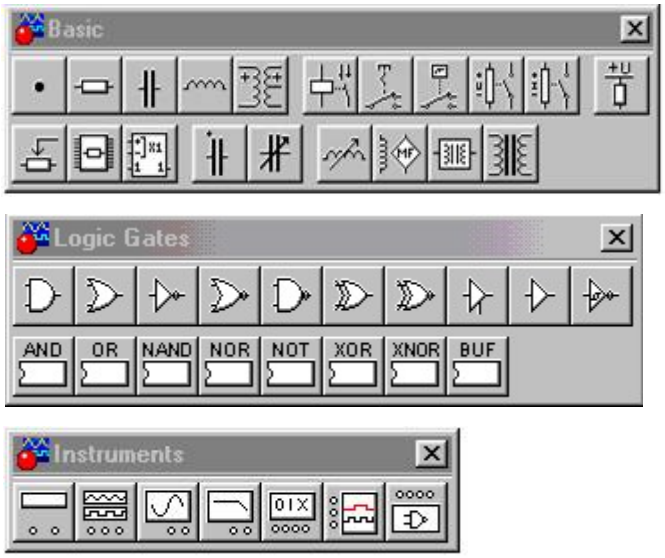


Рисунок 2

Категорія Basic

– вузол служить для з'єднання провідників і створення контрольних точок.

Категорія Logic Gates служить для вибору необхідного типу вентиля.

Категорія Instruments

– *генератор слів*. На схему виводиться зменшене зображення генератора слів. На 16 виходів в нижній частині генератора паралельно подаються біти (генерується слово). На вихід тактового сигналу (правий нижній) подається послідовність тактових імпульсів із заданою частотою. Вхід синхронізації використовується для подачі імпульсу, що синхронізує, від зовнішнього джерела. Подвійним клацанням миші відкривається розширене зображення генератора (рис. 3).

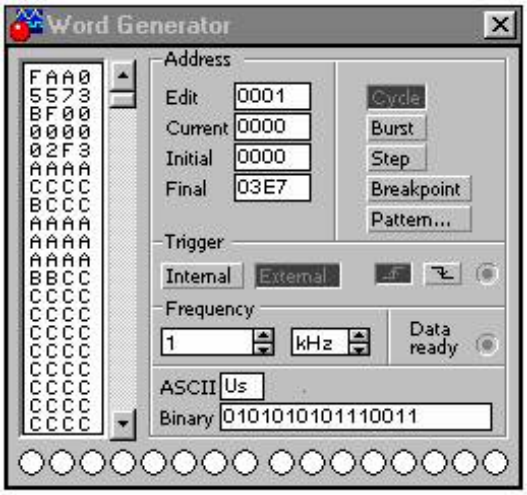


Рисунок 3

Ліва частина генератора містить 16 розрядні слова, що задаються в шістнадцятковому коді. Кожна кодова комбінація заноситься за допомогою клавіатури. Номер редагованої комірки (від 0 до 03FF, тобто від 0 до 1023) висвічується в віконці Edit. В процесі роботи генератора в комірці Address відображається адреса поточної комірки (Current), початкової комірки (Initial) і кінцевої комірки (Final). Видані на 16 виходів (внизу генератора) кодові комбінації відображаються в коді ASCII і двійковому коді.

Генератор може працювати в покроковому, циклічному і безперервному режимах:

– кнопка Step переводить генератор в покроковий режим;

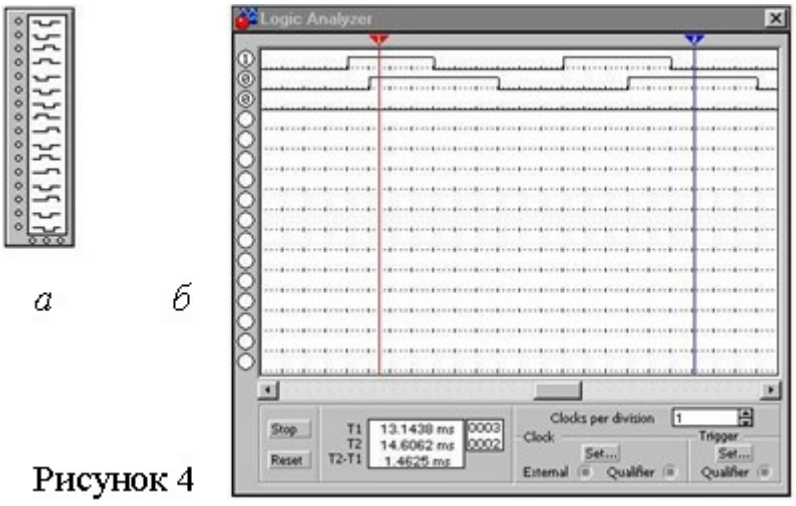
– кнопка Burst – в циклічний режим (на вихід генератора одноразово послідовно надходять всі слова);

– кнопка Cycle – в безперервний режим. Для того щоб перервати роботу в безперервному режимі, потрібно ще раз натиснути кнопку Cycle.

При виконанні лабораторної роботи необхідно:

– входи вентилів з'єднати з X-ми в відповідності зі структурними формулами вихідних функцій, наприклад, (1);

– в ліву частину поля необхідно заносити вхідні дані для роботи схеми. Наприклад, для моделювання кодера ("з 8 в 3") вхідними даними будуть номери бітів, які використовуються для запису десяткових чисел (від 0 до 7), де 0 – 0000 0001, 1 – 0000 0010, 2 – 0000 0100, ..., тобто, індекс відповідає десятковому числу $X_0 = 0$, $X_1 = 1$, $X_2 = 2$ тощо, які потім будуть кодуватися двійковими числами.



а *б*

Рисунок 4

Логічний аналізатор – на схему виводиться зменшене зображення логічного аналізатора (рис. 4, *а*).

Логічний аналізатор підключається до схеми за допомогою контактів в його лівій частині.

У даній роботі використовується для виведення двійкового представлення введеного на лініях входу десяткового числа.

Завдання

1. Виходячи з мінімального числа логічних елементів, розробити структурні схеми наступних типових комбінаторних цифрових схем згідно варіанту (табл. 4).

Таблиця 4

N	Завдання	N	Завдання	N	Завдання
1	Шифратор із 8 в 3	7	Дешифратор із 3 в 8	13	Шифратор із 2 в 4
2	Шифратор із 2 в 4	8	Дешифратор із 2 в 4	14	Мультиплексом із 8 в 1
3	Мультиплексор із 8 в 1	9	Демультиплексор із 1 в 4	15	Мультиплексор із 4 в 1
4	Мультиплексор із 4 в 1	10	Демультиплексор із 1 в 8	16	Шифратор із 8 в 3
5	Демультиплексор із 1 в 8	11	Дешифратор із 2 в 4		
6	Демультиплексор із 1 в 4	12	Дешифратор із 3 в 8		

У звіті повинні бути представлені:

а) умовні позначення і таблиці істинності;

б) алгебраїчні вирази функцій, які виконуються комбінаторними цифровими схемами;

в) розроблені структурні схеми.

2. Реалізувати структурні схеми в системі схемотехнічного моделювання EW. Зібрати схеми згідно п. 1 завдання, включити їх і перевірити на функціонування відповідно до таблиці істинності (ТІ). Що стосується розбіжності отриманих результатів з даними таблиці, знайти і усунути помилки в зібраній схемі.

У звіті повинні бути представлені скрін-шоти покрокового процесу моделювання схеми і отриманого результату.

Приклад виконання завдання для шифратора "із 4 в 2"

Умовне графічне позначення повного CD високого рівня (CD "із 4 в 2") наведено на рис. 5, ТІ – в табл. 5.

Структурні формули вихідних функцій CD "із 4 в 2" запишемо безпосередньо з ТІ у вигляді:

$$Y_0 = X_1 + X_3;\ Y_1 = X_2 + X_3. \quad (1)$$

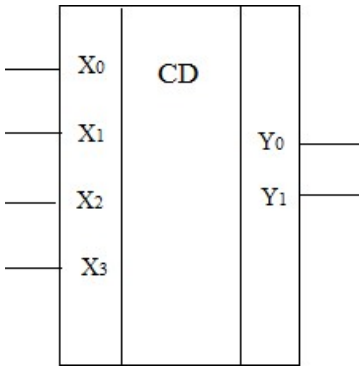


Рисунок 5

Таблиця 5

	X_3	X_2	X_1	X_0	Y_1	Y_0
0	0	0	0	1	0	0
1	0	0	1	0	0	1
2	0	1	0	0	1	0
3	1	0	0	0	1	1

Видно, що цей CD реалізується на основі трьох 2-х входових елементів АБО, як це показано на рис. 6.

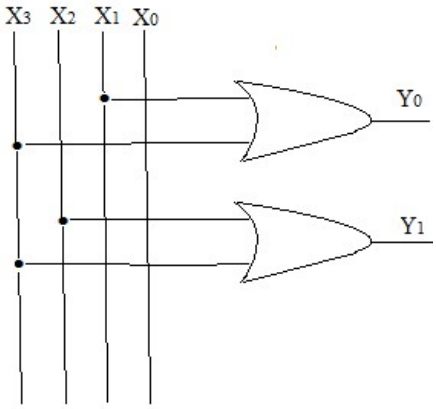


Рисунок 6

Цей шифратор можна реалізувати і на основі елементів І-НЕ. Однак, для такої реалізації шифратора потрібно набагато більша кількість логічних елементів.

Контрольні питання

1. Що таке комбінаторна схема?
2. Що таке шифратор?
3. Що таке дешифратор?
4. Цільове призначення дешифратора?
5. Що таке мультіплексор?
6. Цільове призначення демультіплексора?

Робота №6

ПАМ'ЯТЬ ПК. КОД ВИПРАВЛЕННЯ ПОМИЛОК

Мета роботи: закріпити знання про зберігання інформації в пам’яті комп’ютера; оволодіння навичками реалізації запропонованих алгоритмів завадозахисного кодування.

Стислі теоретичні відомості

Пам'ять комп'ютера через сплески напруги та з інших причин час від часу може помилятися. Щоб боротися з помилками, використовуються спеціальні коди, які вміють виявляти і виправляти помилки. В цьому випадку до кожного слова в пам'яті особливим чином додаються додаткові біти. Коли слово зчитується з пам'яті, ці додаткові біти перевіряються, що і дозволяє виявляти помилки.

Завадозахисні коди діляться на дві групи:

1. *Код з виявленням помилок* має на меті виявити з імовірністю близькою до одиниці наявність помилок.

2. *Код з виправленням помилок* має на меті відновити з ймовірністю близькою до одиниці вихідну інформацію.

Код з виявленням помилок

Коди Геммінга є такі, що самоконтролюються, тобто кодами, що дозволяють автоматично виявляти найбільш ймовірні помилки. Для їх побудови досить приписати до кожного слова один додатковий (контрольний) двійковий розряд і вибрати цифру цього розряду так, щоб загальна кількість одиниць в зображенні будь-якого числа було, наприклад, парних. Одиночна помилка в будь-якому розряді слова (в тому числі, може бути, і в контрольному розряді) змінить парність загальної кількості одиниць. Лічильники по модулю 2, що підраховують кількість одиниць, які містяться серед двійкових цифр числа, можуть давати сигнал про наявність помилок.

При цьому, зрозуміло, що не буде отримано ніяких вказівок про те, в якому саме розряді сталася помилка, і, отже, не має можливості виправити її. Залишаються непоміченими також помилки, що виникають одночасно в двох, в

чотирьох або взагалі в парній кількості розрядів. Втім, подвійні, а тим більше чотириразові помилки є малоймовірними.

Код з виправленням помилок

Коди, в яких можливе автоматичне виправлення помилок, називаються такими, що самокорегуються. Для побудови такого коду, розрахованого на виправлення одиночних помилок, одного контрольного розряду недостатньо. Для того, щоб можна було виправляти помилки, кодова комбінація повинна мати деяку надмірність, яка досягається за рахунок додавання кількох контрольних розрядів. Припустимо, що слово складається з m біт даних, до яких додатково додаються r біт (контрольних розрядів). Тоді загальна довжина слова складе n біт (тобто $n = m + r$). Одиницю з n біт, що містить m біт даних і r контрольних розрядів, називають кодовим словом.

Для будь-яких двох кодових слів, наприклад 10001001 та 10110001, можна визначити, скільки відповідних бітів в них різняться. В даному прикладі таких біта *три*. Щоб визначити кількість бітів, що розрізняються, потрібно над двома кодовими словами виконати логічну операцію *АБО, що виключає*, і порахувати кількість бітів зі значенням 1 в отриманому результаті.

Число бітових позицій, за якими розрізняються два слова, називається інтервалом Геммінга. Якщо інтервал Геммінга для двох слів дорівнює d, значить, що досить d бітових помилок, щоб перетворити одне слово в інше. Наприклад, інтервал Геммінга для кодових слів 11110001 та 00110000 дорівнює 3, оскільки для перетворення першого слова в друге досить 3 бітові помилки.

Виникнення одноразової помилки можливо в будь-якому з елементів слова. Кількість таких ситуацій $C_n^1 = n$. Таким чином, для того, щоб визначити місце виникнення помилки, кількість комбінацій перевірочних елементів 2^r має бути не менше кількості можливих помилкових ситуацій в коді плюс ситуація, коли помилка не виникає. Тому кількість контрольних розрядів r повинно бути вибрано так, щоб задовольнялося нерівність $2^r \geq n + 1$.

З цієї нерівності випливає мінімальне співвідношення перевірочних та інформаційних розрядів, необхідне для виправлення одноразових помилок:

$$2^r - 1 = n.$$

Таким чином, для розрахунку основних параметрів коду Геммінга можна задати кількість перевірочних елементів r, тоді довжина кодових слів $n \le 2^r - 1$, а кількість інформаційних елементів $m = n - r$. Співвідношення між r, n і m наведені нижче (табл. 6).

Таблиця 6

n	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
r	2	3	3	3	3	4	4	4	4	4	4	4	4	5	5	6
m	1	1	2	3	4	4	5	6	7	8	9	10	11	11	12	12

Припустимо далі, що всі $m + r$ розрядів коду розбиті на контрольні групи, які частково перекриваються, причому так, що одиниці в двійковому поданні номера розряду вказують на його приналежність до певних контрольних груп. Наприклад, розряд 5 належить до 1-ої та 3-ої контрольних груп, тому що в двійковому поданні його номера, 1-й і 3-й розряди містять одиниці ($5=000101_2$).

Серед $m + r$ розрядів коду є r розрядів, кожен з яких належить тільки до однієї контрольної групи:

Розряд 1: належить тільки до 1-ої контрольної групи ($1 = 000001_2$).

Розряд 2: належить тільки до 2-ої контрольної групи ($2 = 000010_2$).

Розряд 4: належить тільки до 3-ої контрольної групи ($4 = 000100_2$).

Ці r розрядів вважаються контрольними. Решта m розрядів, кожен з яких належить, щонайменше, до двох контрольних груп, будуть інформаційними розрядами. Введемо умовні позначення для кодових слів (табл. 7):

Таблиця 7

Номер позиції в кодовому слові	1	2	3	4	5	6	7	…
Умовне позначення	u_1	u_2	u_3	u_4	u_5	u_6	u_7	…
Розряди	r_1	r_2	m_1	r_3	m_2	m_3	m_4	…

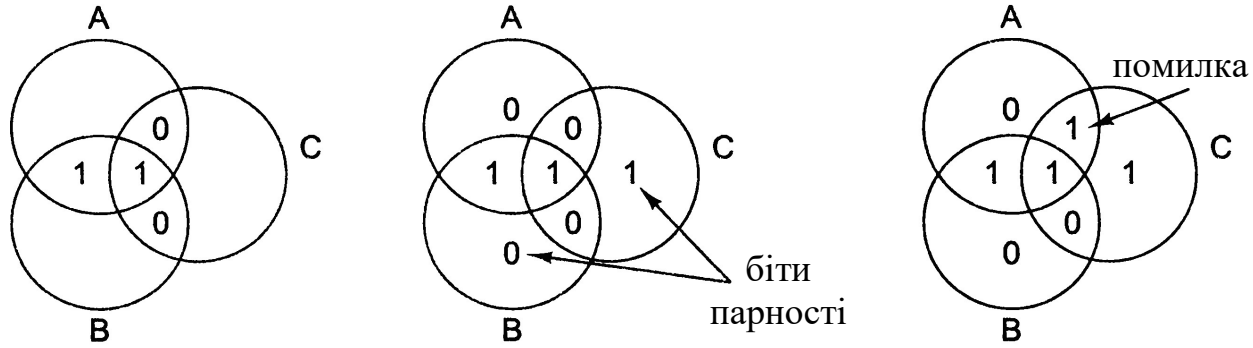
Таким чином, для коду Геммінга в кожному кодовому слові $n(u_1, u_2, u_3, u_4, ..., u_8, ...)$, $r = n - m$ бітів з номерами ступеня 2 є перевірочними, а решта – бітами повідомлення, тобто кодування здійснюється так:

$$E(m_1, m_2, ..., m_k) = (u_1, u_2, ..., u_n) =$$

$$= (r_1, r_2, m_1, r_3, m_2, m_3, m_4, r_4, m_5, m_6, ..., m_k).$$

У кожній з *r* контрольних груп матимемо по одному контрольному розряду. У кожен з контрольних розрядів (біт парності) помістимо таку цифру (0 або 1), щоб загальна кількість одиниць в його контрольній групі була парною.

Перш, ніж звернутися до зазначеного алгоритму, розглянемо просту графічну схему, яка ілюструє ідею коду виправлення помилок для 4-розрядних слів. Діаграма Венна на рис. 7, *а* містить 3 кола *A*, *B* і *C*, які разом утворюють сім секторів. Закодуємо, як приклад, слово з 4 біт 1100 в секторах *AB*, *ABC*, *AC* і *BC*, по одному біту в кожному секторі (в алфавітному порядку).



а *б* *в*

Рисунок 7

Далі додамо біт парності до кожного з трьох порожніх секторів, щоб сума бітів в кожному з трьох кіл, *A*, *B* і *C*, вийшла парною, як показано на рис. 7, *б*. Рисунок відповідає кодовому слову, що складається з 4 біт даних і 3 біт парності.

Проведемо аналогію з контрольними групами, щоб зрозуміти, які інформаційні біти вони контролюють. Перша група контролює розряди 3, 7, 5 вихідного коду, друга – 3, 7, 6, третя група – 5, 7, 6 (№ групи = № контрольного розряду). Очевидно, що вони частково перекриваються (рис. 8).

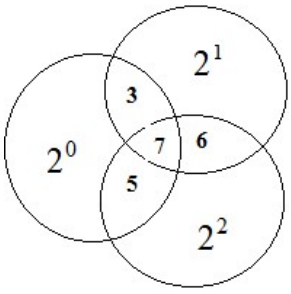


Рисунок 8

При формуванні коду Геммінга конкретне значення кожного контрольного розряду (контрольні суми) знаходять з перевірочних рівнянь. Для контролю парності при *r* = 3 отримаємо наступну систему рівнянь:

$$u_1 \oplus u_3 \oplus u_5 \oplus u_7 = 0;$$
$$u_2 \oplus u_3 \oplus u_6 \oplus u_7 = 0;$$
$$u_4 \oplus u_5 \oplus u_6 \oplus u_7 = 0.$$

Оскільки, $x \oplus x = 0$, то із системи випливає, що

$$u_1 = u_3 \oplus u_5 \oplus u_7;$$
$$u_2 = u_3 \oplus u_6 \oplus u_7;$$
$$u_4 = u_5 \oplus u_6 \oplus u_7.$$

Звідси перевірочні розряди (контрольні суми) знаходяться як

$$\begin{aligned} r_1 = u_1 = u_3 \oplus u_5 \oplus u_7 = m_1 \oplus m_2 \oplus m_4; \\ r_2 = u_2 = u_3 \oplus u_6 \oplus u_7 = m_1 \oplus m_3 \oplus m_4; \\ r_3 = u_4 = u_5 \oplus u_6 \oplus u_7 = m_2 \oplus m_3 \oplus m_4. \end{aligned} \quad (1)$$

Отже, щоб закодувати повідомлення m, як u_i, де i не дорівнює ступеню 2, беруться відповідні біти повідомлення, а перевірочні розряди з індексами ступеня 2 знаходяться з системи перевірочних рівнянь коду. У кожне рівняння системи входить лише одна контрольна сума.

Для декодування і перевірки правильності отриманого каналами зв'язку кодового слова треба провести контроль парності за допомогою наведених вище рівнянь. Якщо всі три результати дорівнюють 0, значить слово прийнято без помилок (або помилка є, але не одноразова). Якщо отримані відмінні від нуля значення, комбінація $r_1r_2r_3$ дасть номер біта, прийнятого з помилкою. Для виправлення помилки досить замінити значення цього біту на протилежне. Після виправлення помилки інформаційне слово отримують з кодового, шляхом відкидання контрольних розрядів.

Приклад 1. Закодуємо повідомлення $m = 0111_2$ кодом Геммінга з 3-ма перевірочними бітами.

Дане повідомлення буде подано такою послідовністю (u_1 u_2 0 u_4 1 1 1), де u_1, u_2, u_4 – контрольні суми, що відповідають r_1, r_2 і r_3.

1	2	3	4	5	6	7
*	*	0	*	1	1	1

Із системи (1) знайдемо контрольні суми:

$$r_1 = u_1 = u_3 \oplus u_5 \oplus u_7 = m_1 \oplus m_2 \oplus m_4 = 0 \oplus 1 \oplus 1 = 0;$$
$$r_2 = u_2 = u_3 \oplus u_6 \oplus u_7 = m_1 \oplus m_3 \oplus m_4 = 0 \oplus 1 \oplus 1 = 0;$$
$$r_3 = u_4 = u_5 \oplus u_6 \oplus u_7 = m_2 \oplus m_3 \oplus m_4 = 1 \oplus 1 \oplus 1 = 1.$$

Таким чином, кодовим словом буде послідовність 0001111_2.

1	2	3	4	5	6	7
0	0	0	1	1	1	1

Припустимо, що при передачі виникла помилка, і було прийнято невірну комбінацію 0001110_2.

Перевіримо її:

$$r_1 = u_1 \oplus u_3 \oplus u_5 \oplus u_7 = 0 \oplus 0 \oplus 1 \oplus 0 = 1;$$
$$r_2 = u_2 \oplus u_3 \oplus u_6 \oplus u_7 = 0 \oplus 0 \oplus 1 \oplus 0 = 1;$$
$$r_3 = u_4 \oplus u_5 \oplus u_6 \oplus u_7 = 1 \oplus 1 \oplus 1 \oplus 0 = 1.$$

Перевірочна комбінація $r_1 r_2 r_3 = 111_2 = 7$. Отже, помилка в сьомому розряді. Для виправлення помилки треба інвертувати сьомий біт.

Завдання

Відновити вихідне інформаційне повідомлення, закодоване кодом Геммінга, виправивши (якщо необхідно) помилку (табл. 8). Вказати в якому біті (бітах) була помилка. Використовувати 866-у кодову сторінку.

Таблиця 8

N	*n*	Отримане повідомлення	N	*n*	Отримане повідомлення	N	*n*	Отримане повідомлення
1	12	♥!	5	12	♫♦	9	12	☼/
2	7	♠j	6	7	<4	10	7	8▲
3	12	☼:	7	12	♂↔	11	12	☺з
4	7	)з	8	7	:]	12	7	►ц

Контрольні питання

1. Класифікація запам’ятовуючих пристроїв за засобом доступу.
2. Чому пам’ять має ієрархічну багатоступеневу організацію?
3. Дайте визначення пам’яті довільного доступу.
4. Дайте стислий опис кєш-пам’яті.
5. Який тип пам’яті зберігає дані, використовувані при виконанні поточної програми?
6. Який тип пам’яті призначений для короткочасного зберігання, запису та видачі інформації, безпосередньо в найближчі такти роботи машини?

Робота №7

ВПЛИВ РІВНІВ КЕШ-ПАМ'ЯТІ НА ЧАС ОБРОБКИ МАСИВІВ ДАНИХ

Мета роботи: дослідження залежності часу доступу до даних в пам'яті від їх обсягу; дослідження залежності часу доступу до даних в пам'яті від порядку їх обходу; оволодіння навичками розробки інструментів дослідження.

Стислі теоретичні відомості

На практиці часто зустрічається завдання обробки масивів даних. З точки зору швидкості виконання програми, важливо в якому порядку обробляються елементи масиву, тому що від цього буде залежати, наскільки ефективно буде працювати кеш-пам'ять. Основними особливостями організації кеш-пам'яті, які відіграють важливу роль при обробці масивів даних, є блочне кешування даних і апаратна передвибірка даних в кеш. Крім того, істотно, чи вміщається новий масив в кеш-пам'яті.

Залежність часу доступу до елементів масиву від рівнів кеш-пам'яті

Ієрархія пам'яті включає кілька рівнів кеш-пам'яті різного розміру і з різним часом доступу. Припустимо, деяка програма виконує багаторазову обробку елементів масиву. Якщо побудувати графік залежності середнього часу доступу до одного елементу масиву від розміру масиву, то він повинен мати нелінійний характер. При малих розміру масиву, коли всі дані вміщаються в кеш-пам'яті першого рівня, час доступу до елементу буде найменшим, і не буде змінюватися при збільшенні розміру масиву. Якщо розмір масиву перевищить розмір кеш-пам'яті першого рівня, то весь масив цілком уже не зможе в ньому розміститися. Тому при зверненні до деяких елементів масиву в кеш-пам'яті першого рівня будуть траплятися кеш-промахи, і елементи будуть завантажуватися з кеш-пам'яті другого рівня (або оперативної пам'яті). В результаті, чим більше кеш-промахів відбувається, тим більше буде середній час доступу до елементу, аж до часу доступу до наступного рівня ієрархії пам'яті. В результаті зі збільшенням розміру масиву середній час доступу до елементу буде поступово зростати. Таким чином, аналіз графіка може показати, які обсяги різних рівнів кеш-пам'яті, наявних в системі.

Дані з оперативної пам'яті в кеш-пам'ять зчитуються цілими блоками. Розмір блоку дорівнює одному або декільком кеш-рядкам. Якщо елементи в масиві обробляються послідовно один за одним, то спроба читання першого елемента кеш-ланцюжка призводить до копіювання всього блоку з повільної оперативної пам'яті в кеш-пам'ять. Читання кількох наступних елементів виконується набагато швидше, тому що вони вже знаходяться в швидкій кеш-пам'яті.

Проведення дослідження

Для вивчення впливу кеш-пам'яті на швидкість доступу до даних в лабораторній роботі потрібно написати програму, що виконує обробку даних різного розміру трьома характерними способами: послідовно в бік збільшення адрес, послідовно в бік зменшення адрес і в довільному порядку. Дослідження часу виконання програми в залежності від порядку обходу і обсягу оброблюваних даних дозволить спостерігати вплив кеш-пам'яті.

Вибір розмірів оброблюваного масиву

Для визначення середнього часу доступу до даних, програма повинна багаторазово виконувати читання елементів масиву заданого розміру (N) в заданому порядку. Діапазон зміни розмірів шуканого масиву визначається наступними межами:

– мінімальне значення N_{min} вибирається свідомо менше, ніж розмір кеш-пам'яті даних 1-го рівня (наприклад, 1 Кбайт – 256 елементів типу int);

– максимальне значення N_{max} вибирається свідомо більше, ніж розмір кеш-пам'яті останнього рівня (наприклад, 32 Мбайт).

Програма повинна перебирати розміри масиву від N_{min} до N_{max}, для кожного конкретного N, визначаючи середній час доступу до елементу масиву. Крок зміни N повинен бути досить маленьким, щоб побачити всі істотні вигини на графіку, і досить великим, щоб час виконання тесту не було занадто довгим. Можна використовувати змінний крок, який росте разом із зростанням N.

Виконання обходу

Для кожного конкретного розміру масиву N програма повинна виконати багаторазове читання елементів масиву в заданому порядку. Щоб під час обходу не витрачати час на обчислення індексу кожного наступного елемента, використовується

наступний прийом. Значення елементів масиву заповнюються таким чином, щоб сформувати однозв'язний циклічний список, в якому значення чергового елемента є номером наступного. Обхід тоді може бути виконаний циклом такого вигляду:

```
for (k = 0, i = 0; i < N*K; i++)
        k = x[k];
```

тут `N` – розмір масиву, `K` – кількість обходів масиву, `x[k]` – елемент масиву.

Отже, для кожного конкретного *N* програма повинна виконувати такі дії:

1. Заповнити значення елементів, щоб вони утворили циклічний однозв'язний список відповідно.

2. Виконати багаторазовий обхід масиву. Виміряти час обходу.

Способи обходу елементів масиву

В рамках лабораторної роботи пропонується порівняти час доступу до даних для трьох характерних способів обходу:

– прямий обхід в бік збільшення адресу;

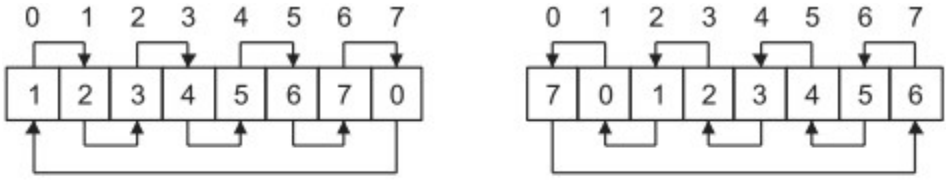
– зворотний обхід в сторону зменшення адресу;

– обхід елементів у випадковому порядку.

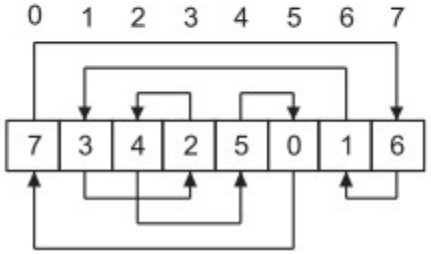
Для кожного способу обходу в програмі необхідно заповнити значення елементів масиву таким чином, щоб вони утворили однозв'язний циклічний список, захоплюючий всі елементи масиву (рис. 9). Особливу увагу слід звернути на заповнення масиву для випадкового обходу. Наприклад, якщо у варіанті для випадкового масиву, наведеному на рис. 9, *в*, поміняти місцями значення комірок 5 і 6, то в обході братимуть участь тільки елементи з номерами 0, 6 і 7. Необхідно, щоб при обході масиву брали участь всі елементи.

На рис. 10 наведено приклад графіків, отриманих на процесорі Intel Xeon E5420 (L1: 32 KB, L2: 6 MB). Видно різницю в швидкості послідовного і випадкового обходів масиву. На відповідних місцях графіка випадкового обходу видно зростання часу звернення до елементів.

Примітка 1. Перед вимірюванням часу для кожного розміру *N* необхідно здійснити одноразовий обхід масиву, щоб "прогріти кеш-пам'ять", тобто вивантажити з кеш-пам'яті сторонні дані, розмістивши там (по можливості) необхідні дані.



а *б*



в

Рисунок 9

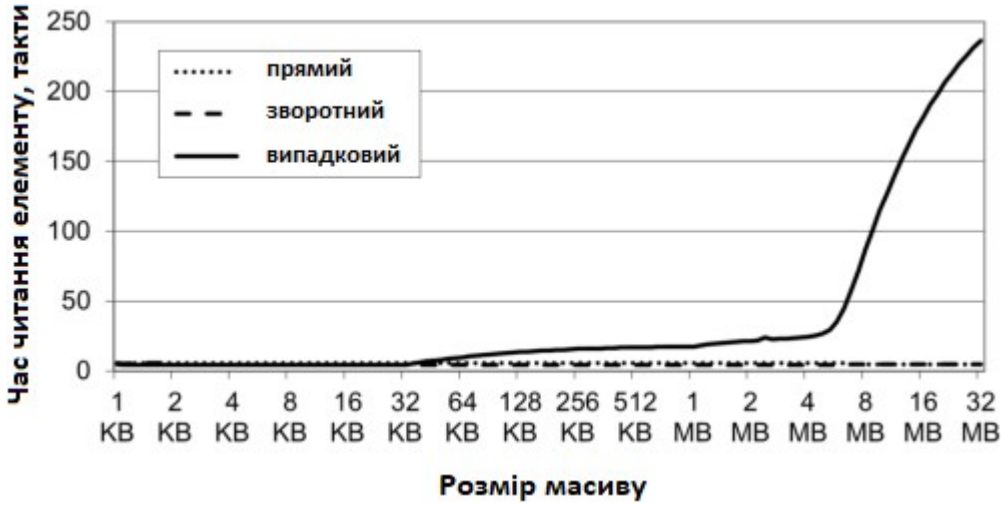


Рисунок 10

Примітка 2. Результуючий графік може виявитися сильно "засміченим" високими піками. Таке буває, якщо на машині працюють сторонні "важкі" програми. В цьому випадку рекомендується вбудувати в програму додатковий цикл, який би проганяв тест кілька разів для кожного розміру масиву, і автоматично зберігав мінімальний час за кількома вимірами.

Завдання

1. Написати програму, що багаторазово виконує обхід масиву заданого розміру трьома способами.

2. Для кожного розміру масиву і способу обходу виміряти середній час доступу до одного елементу (в тактах процесора). Побудувати графіки залеж-

ності середнього часу доступу від розміру масиву. Вказати для якого процесора проводилося дослідження.

3. На основі аналізу отриманих графіків: визначити розміри кеш-пам'яті різних рівнів, обґрунтувати відповідь, зіставити результат з відомими реальними значеннями; визначити розміри масиву, при яких час доступу до елементу масиву при випадковому обході більше, ніж при прямому або зворотному; пояснити причини цієї різниці в часах.

Контрольні питання

1. Який тип пам’яті призначений для короткочасного зберігання, запису та видачі інформації, безпосередньо в найближчі такти роботи машини?
2. Дайте стислий опис кєш-пам’яті.
3. Які існують рівні кєш-пам’яті?
4. Чому кєш-пам’ять має багаторівневу організацію?
5. Чому час доступу до елементів масиву залежіть від рівнів кеш-пам'яті?
6. Чи є залежність швидкодії процесору від засобу організації даних, що обробляються?
7. Які способи обходу масиву були використані при виконанні лабораторної роботи?

Робота №8

ОБЧИСЛЕННЯ З ВИКОРИСТАННЯМ ММХ РЕГІСТРІВ

Мета роботи: написання програми, для дослідження можливості прискорення обчислювального процесу при використанні команд ММХ (мультимедійного розширення SIMD).

Стислі теоретичні відомості

Intel MMX технологія включає набір розширень до архітектури Intel, які розроблені, щоб збільшити продуктивність засобів мультимедіа та комунікацій.

Ці розширення (які включають нові регістри, типи даних, і інструкції) об'єднані з моделлю виконання "одна інструкція, багато даних" (SIMD), щоб прискорити виконання програм типу рухливе відео, комбіновану графіку з відео, обробкою зображень, звуковим синтезом, синтезом мови і стисненням, телефонією, відео конференц-зв'язком і 2D та 3D графікою, які типово використовують алгоритми з інтенсивними обчисленнями, щоб виконувати повторювані дії на великих множинах простих елементів даних.

ММХ технологія визначає просту і гнучку модель програмного забезпечення, без нового режиму або видимого стану для операційної системи. Все існуюче програмне забезпечення продовжить працювати правильно, без модифікації, на процесорах архітектури Intel, які включають ММХ технологію, навіть в присутності існуючих і нових додатків, які включають цю технологію.

ММХ технологія забезпечує наступні нові розширення до Intel-архітектури:

1. Вісім ММХ регістрів (від ММ0 до ММ7).
2. Чотири ММХ типу даних (упаковані байти, упаковані слова, упаковані подвійні слова і четверні слова (quadword)).
3. Система команд ММХ.

Регістри ММХ

Набір регістрів ММХ складається з восьми 64-розрядних регістрів. ММХ команди звертаються до регістрів ММХ безпосередньо, використовуючи

ім'я регістра від MM0 до MM7. Ці регістри можуть використовуватися тільки для виконання обчислень над MMX типами даних; вони не можуть використовуватися, щоб адресувати пам'ять. Адресація операндів MMX команди в пам'яті здійснюється, використовуючи стандартні способи адресації і регістри EAX, EBX, ECX, EDX, EBP, ESI, EDI, і ESP.

Типи даних

Основними типами даних для набору інструкцій MMX є упакований цілочисельний тип, де кілька чисел згруповані в одне 64-х розрядне значення. Ці 64-х розрядні значення завантажуються в регістри MMX. Підтримуються знакові і беззнакові числа: байт, слово, подвійне і четверне слово (byte, word, doubleword і quadword).

Таким чином, є чотири MMX типу:

– упакований байт – 8 байт, упакованих в одне 64-бітове значення;

– упаковане слово – 4 слова, упакованих в одне 64-бітове значення;

– упаковане подвійне слово – два 32-х бітних подвійних слова, упакованих в одне 64-бітове значення;

– четверне слово – одне 64-бітове значення.

На рис. 11 показано схему розміщення даних у 64-х розрядному числі. Байти в упакованому типі даних байтів пронумеровані від 0 до 7, від байта 0, що міститься в наймолодших бітах типу даних (біти 0…7), до байта 7, що міститься в старших бітах (біти 56…63). Слова в упакованому типі даних слів пронумеровані від 0 до 3, від слова 0, що міститься в бітах 0…15 типу даних до слова 3, що міститься в бітах 48…63. Подвійні слова в упакованому типі даних подвійних слів пронумеровані 0 і 1, з подвійним словом 0 містяться в бітах 0…31 і подвійним словом 1, що містяться в бітах 32…63.

MMX команди переміщають упаковані типи даних (упаковані байти, упаковані слова, або упаковані подвійні слова) і тип даних четверне слово в/із пам'яті або в/із універсальних регістрів архітектури Intel в 64-розрядні блоки. Однак, при виконанні арифметичних або логічних операцій над упакованими типами даних, MMX команди оперують паралельно над індивідуальними бай-

тами, словами, або подвійними словами, що містяться в 64-розрядному MMX регістрі. При операціях над байтами, словами, і подвійними словами всередині упакованих типів даних, MMX команди оперують як зі знаковими, так і беззнаковими цілими байтами, словами, подвійними словами.

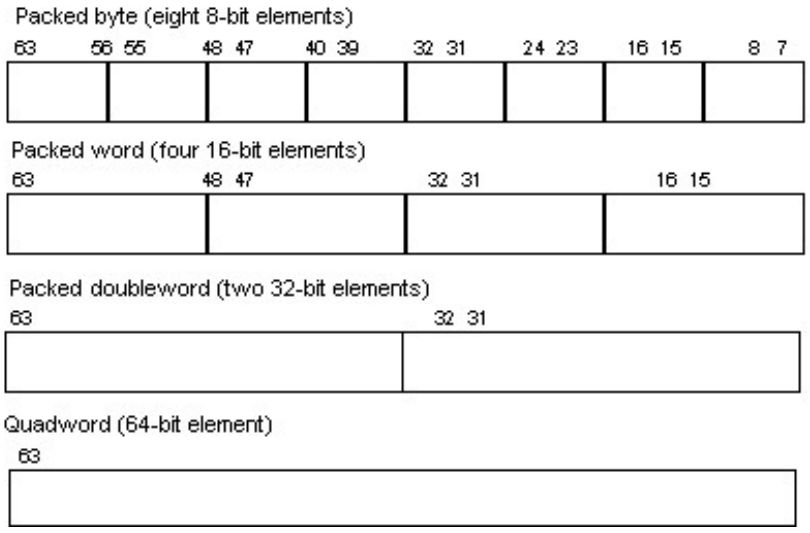


Рисунок 11

Наприклад, колір пікселя зазвичай представляється 8-бітовим цілим (байтом). Використовуючи технологію MMX можна упакувати 8 пікселів в одне 64-ти бітове значення і завантажити в регістр MMX. Потім команди MMX будуть відразу обробляти всі 8 пікселів зображення

Модель SIMD

MMX технологія використовує методику SIMD для виконання арифметичних і логічних операцій над байтами, словами, або подвійними словами, упакованих в 64-розрядні регістри MMX. Наприклад, команда PADDSB складає 8 знакових байтів вихідного операнда з 8 знаковими байтами в операнді адресата і зберігає 8 результуючих байтів в операнді адресата. SIMD методика прискорює ефективність виконання програм, дозволяючи одну і ту ж операцію виконувати паралельно на множині елементів даних.

Сумісність

Технологія MMX створювалася як повністю сумісна з усім попереднім програмним забезпеченням, в тому числі операційними системами. Тому при її реалізації не було додано нових регістрів або станів процесора. Для розташу-

вання 8-й регістрів MMX використані регістри співпроцесора, що дозволило залишити без зміни стандартні механізми перемикання контекстів задач в існуючих операційних системах. Інструкції співпроцесора, що відповідають за збереження/відновлення стану FPU, також обробляють і стан розширення MMX. Таким чином, MMX не додало нових виключень або станів процесора.

Інструкції

Інструкції MMX охоплюють кілька областей:

1. Базові арифметичні операції, такі як ADD – складання, SUB – віднімання, MUL – множення і MUL-ADD – множення-складання.
2. Операції порівняння.
3. Операції перетворення типів, упаковка/розпаковка даних.
4. Логічні операції типу AND, NOT, OR і XOR.
5. Операції зсуву.
6. Операції пересилання даних (MOV).

Арифметичні і логічні інструкції підтримують різні типи упакованих даних. З огляду на це все розширення MMX включає 57 кодів операцій.

Короткий огляд набору інструкцій MMX

Інструкції та відповідні мнемоніки в таблиці згруповані за категоріями. Якщо інструкція підтримує тип даний байт (B), слово (W), подвійне слово (DW) або четверне слово (QW), відповідні абревіатури вказані в квадратних дужках. Наприклад, для базової мнемоніки PADD (packed add) можливі наступні варіанти: PADDB, PADDW і PADDD.

Таблиця 9

Категорія	Мнемоніка	Опис
1	2	3
Арифметичні операції	PADD[B,W,D]	Додавання (Add with wrap-around on [byte, word, doubleword])
	PADDS[B,W]	Додавання знакове с насиченням (Add signed with saturation on [byte, word])
	PADDUS[B,W]	Додавання беззнакове з насиченням (Add unsigned with saturation on [byte, word])
	PSUB[B,W,D]	Віднімання (Subtract with wrap-around on [byte, word, doubleword])

Продовження табл. 9

1	2	3
	PSUBS[B,W]	Віднімання знакове с насиченням (Subtract signed with saturation on [byte, word])
	PSUBUS[B,W]	Віднімання беззнакове с насиченням (Subtract unsigned with saturation on [byte, word])
	PMULHW	Packed multiply high on words
	PMULLW	Packed multiply low on words
	PMADDWD	Множення запакованих байтів и додавання отриманих пар слів (Packed multiply on words and add resulting pairs)
Операції порівняння	PCMPEQ[B,W,D]	Порівняння запакованих на рівність (Packed compare for equality [byte, word,doubleword])
	PCMPGT[B,W,D]	Порівняння запакованих на більше (Packed compare greater than [byte, word, doubleword])
Перетворення розрядності	PACKUSWB	Упаковка слів до байтів (без знакове з насиченням) Pack words into bytes (unsigned with saturation)
	PACKSS[WB,DW]	Упаковка [слів в байти, подвійних слів до слова] (знакове з насиченням) Pack [words into bytes, doublewords into words] (signed with saturation)
	PUNPCKH [BW,WD,DQ]	Розпакування старших [байтів, слів, подвійних слів] з MMX регістру. Unpack (interleave) high-order [bytes, words, doublewords] from MMX register
	PUNPCKL [BW,WD,DQ]	Розпакування молодших [байтів, слів, подвійних слів] з MMX регістру. Unpack (interleave) low-order [bytes, words, doublewords] from MMX register
Логічні	PAND	Бітовий (bitwise) AND
	PANDN	Бітовий (bitwise) AND NOT
	POR	Бітовий й (bitwise) OR
	PXOR	Бітовий (bitwise) XOR
Здвиги	PSLL[W,D,Q]	Логічний зсув вліво запакованих [слів, подвійних слів, четверних слів] на значеня, яке вказане в MMX регістрі або в команді Packed shift left logical [word, doubleword, quadword] by amount specified in MMX register or by immediate value
	PSRL[W,D,Q]	Логічний зсув вправо запакованих [слів, подвійних слів, четверних слів] на значеня, яке вказане в MMX регістрі або в команді Packed shift right logical [word, doubleword, quadword] by amount specified in MMX register or by immediate value

Продовження табл. 9

1	2	3
	PSRA[W,D]	Арифметичний зсув вправо запакованих [слів, подвійних слів, четверних слів] на значеня, яке вказане в MMX регістрі або в команді Packed shift right arithmetic [word, doubleword] by amount specified in MMX register or by immediate value
Пересилання даних	MOV[D,Q]	Пересилання [подвійних, четверних] сліва в/з MMX регістру Move [doubleword, quadword] to MMX register or from MMX register
Управління станом FPU і MMX	EMMS	Скидання стану FPU, переключення між MMX та FPU режимами Empty MMX state

Приклади інструкцій

Опишемо кілька прикладів MMX інструкцій. Для прикладу взято операції з 16-бітними словами, проте більшість цих операцій існують і для байт і подвійних слів.

Приклад 1 ілюструє додавання упакованих слів без насичення (PADD [W]). Обчислюється чотири суми для восьми 16-бітних елементів, всі вони виконуються незалежно і паралельно (рис. 12). В даному випадку результат самого правого складання перевищує значення, яке може бути представлено 16-ма бітами, тобто відбувається переповнення. Результатом складання FFFFh + 8000h буде 17-бітове число. 17-й біт втрачається через усічення результату, відповідно виходить число 7FFFh.

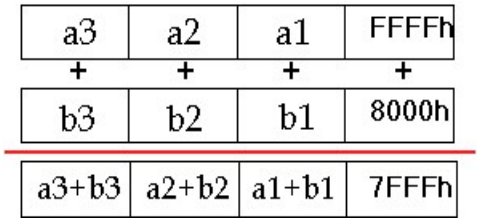


Рисунок 12

У *прикладі 2* показано додавання упакованих слів з беззнаковим насиченням (PADDUS [W]). Використовуються ті ж дані, що і в попередньому прикладі. Саме праве складання дає результат, що не міститься в 16 біт, отже, відбувається насичення (рис. 13). Термін насичення означає, що при переповненні складання (або віднімання) результат обмежений максимальним (або мінімаль-

ним) числом, уявленим в даній розрядності. Для беззнакових 16-бітних слів максимальним і мінімальним будуть FFFFh і 0000h, для знакових це 7FFFh і 8000h відповідно. Насичення може виявитися важливим при роботі з пікселями зображення, коли переповнений може викликати появу точок чорного кольору.

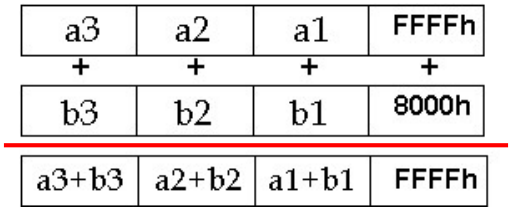


Рисунок 13

Приклад 3 показує роботу інструкції PMADDWD, що виконує операцію множення-додавання (рис. 14). Ця операція часто використовується в алгоритмах обробки сигналів. Ця команда змінює розрядність результату – її операнди мають розмір слова (16 біт), а результат подвійне слово (32 біта).

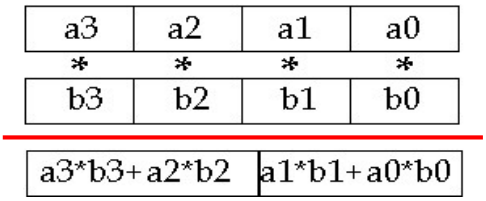


Рисунок 14

Приклад 4 ілюструє паралельне порівняння упакованих даних (PCMPGT [W]). Команда порівнює чотири пари 16-бітних слів (рис. 15). У регістр результату заноситися true/істина (FFFFh) або false/хибність (0000h) окремо для кожного порівняння. На малюнку можна бачити порівняння за умовою "більше" ("greater than"). Ця команда не змінює прапорів процесора.

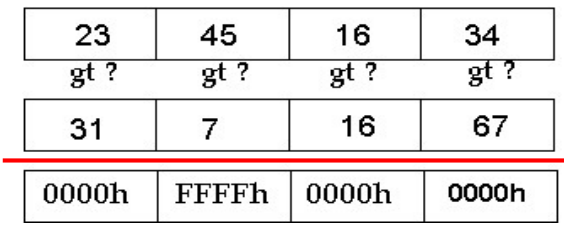


Рисунок 15

Результат такого порівняння може бути використаний як маска для вибору елементів з іншого вхідного потоку з використанням логічних операцій. Такий метод дозволяє в якійсь мірі скоротити кількість інструкцій переходу.

Приклад 5 показує механізм роботи інструкції упаковки (скорочення розрядності – PACKSS [DW]). На вхід подаються чотири 32-бітних подвійних слова в двох різних регістрах, які упаковуються в чотири 16-бітних слова в одному регістрі (рис. 16). Якщо значення вихідного операнду (наприклад, b1) перевершує розрядність результату (b1'), перетворення виконується з насиченням.

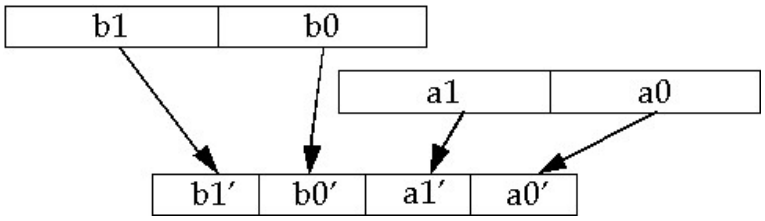


Рисунок 16

Інструкції упаковки/розпаковки зазвичай використовуються, коли розрядність проміжних результатів обчислень перевершує розрядність вихідних даних.

Завдання

Написати програму мовою програмування C/C++ з використанням асемблерних вставок, що виконує наступні дії: провести в циклі близько одного мільйона операцій додавання і заміряти час виконання, потім зробити все те ж саме для операції віднімання і множення з використанням MMX регістрів; поступити таким самим чином з цілочисельними операндами без використання MMX регістрів за аналогією з другою лабораторною роботою; порівняти часи розрахунків, зробити висновки про перевагу використання MMX регістрів.

Контрольні питання

1. Призначення набору розширень Intel MMX технології до архітектури Intel.
2. Які нові розширення до Intel-архітектури забезпечує MMX технологія?
3. Дайте стислий опис регістрів MMX.
4. Які основні типи даних використовуються для набору інструкцій MMX?
5. Що таке модель SIMD?
6. Які області охоплюють інструкції MMX?
7. Правила запису інструкцій MMX.

Робота №9

ОЦІНКА ЧАСУ ВИКОНАННЯ АРИФМЕТИЧНИХ ОПЕРАЦІЙ

Мета роботи: навчитися визначати час виконання різних арифметичних операцій для різних операндів; оволодіти навичками практичної реалізації програми вимірювань.

Стислі теоретичні відомості

У даній лабораторній роботі потрібно оцінити швидкість виконання процесором операцій з операндами різних типів, наприклад, з цілими числами і з числами з плаваючою точкою. Оскільки оцінити час виконання однієї операції для сучасного процесора проблематично, необхідно підрахувати час виконання декількох операцій (для сучасної обчислювальної машини близько одного мільйона операцій).

Заміряти час виконання операцій можна за допомогою функції `ftime()`. Виглядає це наступним чином. За допомогою функції `ftime()` отримуємо деякий проміжок часу виконання. Потім в циклі виконуємо необхідну кількість операцій і знову отримуємо значення часу. Далі знаходимо різницю часів після і до циклу. Отримане значення часу і є час, витрачений процесором на виконання циклу операцій. Оцінювати час виконання циклу операцій необхідно в мілісекундах. Замість функції `ftime()` можна використовувати функцію `GetTickCount()`.

Програма повинна почергово запускати на виконання 8 циклів, кожен з яких буде виконувати різні арифметичні операції (додавання, віднімання, множення і ділення). Перші чотири цикли будуть виконувати операції для операндів першого типу, а другі чотири для операндів другого типу. На початку і в кінці кожного циклу програма буде запам'ятовувати час (в мілісекундах) запуску і зупинки цього циклу. Різниця між цими величинами буде часом виконання циклу конкретної арифметичній операції.

З огляду на високу швидкодію ЕОМ, використовуваних при програмуванні, кількість ітерацій має бути вибрана досить великою для того, щоб помітити очевидну різницю в часі виконання різних операцій. Використання циклу

для оцінки часу виконання вимагає додаткових витрат процесорного часу на організацію циклу. Це означає, що порівнюватися будуть не реальні часи виконання арифметичних операцій, а часи виконання з доданими до них тимчасовими затримками на організацію циклів. Таким чином, число ітерацій для всіх циклів повинно бути вибрано однаковим для того, щоб оцінити реальну залежність в часі виконання операцій в однакових умовах. Тобто це означає, що якщо для всіх операцій використовується однакова кількість ітерацій, то і затримки на організацію циклів для всіх операцій однакові. Це в свою чергу означає що ми можемо зіставляти отримані результати і аналізувати закономірності витрат процесорного часу на виконання арифметичних операцій без необхідності обчислення часу затримки на організацію циклів. Програма повинна виводити на екран результати всіх вимірювань.

Програма повинна виводити на екран результати всіх вимірювань. Приблизний можливий зовнішній вигляд результатів роботи програми представлений нижче на рис. 17. Як видно з рисунку, операції додавання, віднімання і множення значно відрізняються за часом виконання від операції ділення. Крім того, є деякі розбіжності в часі виконання між цілочисельними операндами і операндами з плаваючою точкою.

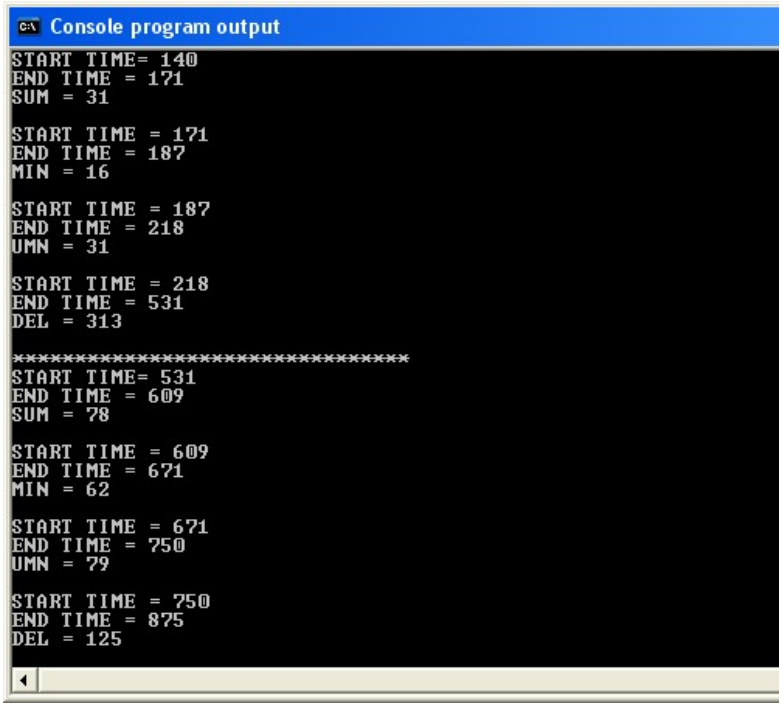


Рисунок 17

Завдання

Написати програму для оцінки часу виконання арифметичних операцій (додавання, віднімання, множення, ділення) для цілочисельних операндів і для операндів з плаваючою точкою мовою програмування C/C++.

Контрольні питання

1. Дайте визначення мікропроцесору.
2. Дайте визначення швидкодії процесору.
3. Що впливає на швидкодію процесору?
4. Чи є залежність швидкодії процесору від типу операндів, що обробляються?
5. Як здійснювався замір часу виконання операцій при виконанні лабораторної роботи?

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. *Абель П.* Язык Ассемблер для IBM PC и программирования / Пер. с англ. Ю.В.Сальникова. – М.: Высш. шк., 1992. – 447 с.
2. *Марек Р.* Ассемблер на примерах. Базовый курс. – СПб.: Наука и техника, 2005. – 240 с.
3. *Смит Б.Э.*, *Джонсон М.Т.* Архитектура и программирование микропроцессора INTEL 80386 / Пер. с англ. В.Л.Григорьева. – М.: Конкорд, 1992. – 334 с.
4. *Юров В.И.* Assembler. Учебник для вузов. – СПб.: Питер, 2003. – 637 с.
5. *Джордейн Р.* Справочник программиста персональных компьютеров типа IBM PC, XT и AT // Пер. с англ. / Предисл. Н.В.Гайского. – М.: Финансы и статистика, 1992. – 544 с.
6. *Юров В.И.* Assembler. Практикум. – СПб.: Питер, 2006. – 399 с.

ЗМІСТ

Начальне видання

УСТЕНКО Ірина Валеріївна
ЛАТАНСЬКА Людмила Олексіївна
УСТЕНКО Сергій Анатолійович

МЕТОДИЧНІ ВКАЗІВКИ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ З ДИСЦИПЛІНИ «АРХІТЕКТУРА КОМП'ЮТЕРІВ»

Надруковано з оригінал-макету,
підготовленого автором видання

Підписано до друку 27.09.2018 р. Формат 60×94⁄16. Папір офсетний.
Друк цифровий. Ум. друк. арк. 2,1. Тираж 100.
Зам. №___ від 27.09.2018 р.

Надруковано ТОВ «ГРУППА КОМПАНІЙ ПЛАНЕТА»
тел. (048) 775-16-30