МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 27.00.00.000 ПЗ

Група ІПм-22-1

Чичул Наталія

2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Чичул Наталія Вікторівна

(прізвище, ім'я, по-батькові)

УДК 004.942

(індекс)

МАГІСТЕРСЬКА РОБОТА

Алгоритмічний аналіз мікропроцесорних основ операційних систем

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 – Інженерія програмного забезпечення

(шифр і назва спеціальності)

Чичул Н. В..

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Царева Олександра Степанівна, асистент

(прізвище, ім'я, по-батькові, науковий ступінь, вчене звання)

Допущено до захисту

В. о. завідувача кафедри

доц. Бандура В. В.

(посада) (підпис) (дата) (ініціали та прізвище)

Рецензент

доц.

(посада) (підпис) (дата) (ініціали та прізвище)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Івано-Франківськ – 2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітній рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

В. о зав. кафедрою ____________ ІПЗ

доц. В.В. Бандура

“ 04 ” вересня 2023 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Чичул Наталії Вікторівні

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “Алгоритмічний аналіз мікропроцесорних основ операційних систем ”

керівник проекту (роботи) Царева Олександра Степанівна, асистент

затверджені наказом закладу вищої освіти від “18 ” грудня 2023 р. №738/7

2. Строк подання студентом проекту (роботи) 16 січня 2024 р.

3**. Вихідні дані до проекту (роботи)** Теоретичні концепції та формальні моделі побудови та функціонування інформаційних та програмних технологій певного класу

4. Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)

1. Дослідження архітектурних особливостей мікропроцесора

2. Дослідження алгоритмічних основ роботи мікропроцесора

3. Розробка програмного забезпечення на мікропроцесорній основі

Перелік графічного матеріалу (із точним зазначенням обов’язкових креслень)

1. Загальна організація цифрової обчислювальної системи (рис. 1.1)

2. Типовий сегмент програми машинного коду 6800 (рис. 1.2)

3. Внутрішня архітектура типового 8-розрядного мікропроцесора (рис. 1.3)

4. Типова архітектура однокристалічного мікропроцесора (рис. 1.4)

5. Код умови або регістр стану мікропроцесора 6800 і функціональний розподіл його бітів даних (рис. 1.5)

6. Блок-схема, що показує просте використання біта N у циклі віднімання

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата
Нормоконтроль	доц., к.т.н. Вовк Р. Б.	
Перевірка на плагіат	доц., к.т.н. Вовк Р. Б.	

7. Дата видачі завдання 02 вересня 2023 р.

Керівник ___________________________
(підпис)

Завдання прийняв до виконання ___________________________
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури	01.10.2023	виконано
2	Долсідження архітектурних особливостей мікропроцесора	25.10.2023	виконано
3	Дослідження алгоритмічних основ мікропроцесора	10.11.2023	виконано
4	Етапи та принципи розробки програмного забезпечення на мікропроцесорній основі	22.11.2023	виконано
5	Застосування точки зупинки при розробці програмного забезпечення	04.12.2024	виконано
6	Особливості застосування інтерпретаторів та компіляторів	22.12.2023	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	15.01.2024	Виконано

Студент – магістр ____________________
(підпис)

Керівник роботи ____________________
(підпис)

АНОТАЦІЯ

Магістерська робота: 109 с., 33 рис.,40 джерел.

Тема:“ Алгоритмічний аналіз мікропроцесорних основ операційних систем

Об’єкт дослідження: мікропроцесорні основи, на яких функціонують операційні системи.

Мета роботи: дослідження та оцінка ефективності алгоритмів операційних систем, адаптованих під мікропроцесорні архітектури.

Предмет дослідження: аналіз архітектурних особливостей різних мікропроцесорів та їх вплив на функціонування операційних систем.

Результати дослідження:

Результати магістерського дослідження визначили оптимальні алгоритми та стратегії управління ресурсами операційних систем для різних мікропроцесорів, що дозволяє розробити конкретні рекомендації для їх оптимізації та практичної реалізації.

Висновок:

У результаті проведеного алгоритмічного аналізу мікропроцесорних основ операційних систем виявлено ключові фактори, впливаючи на їх ефективність та взаємодію з апаратним середовищем. Дослідження архітектурних особливостей мікропроцесорів дозволило визначити оптимальні алгоритми та стратегії управління ресурсами, сприяючи поліпшенню продуктивності операційних систем на різноманітних апаратних платформах. Результати експериментального аналізу підтверджують високий потенціал оптимізації, що виражається в пропозиціях конкретних рекомендацій для розробників та адміністраторів операційних систем.

МІКРОПРОЦЕСОР, ОПЕРАЦІЙНА СИСТЕМА, АРХІТЕКТУРНІ ОСОБЛИВОСТІ

SUMMARY

Master's thesis: 109 pages, 33 figures, 40 sources.

Thesis Subject: "Algorithmic analysis of microprocessor bases of operating systems

Object of Research: Microprocessor bases on which operating systems function.

Reseach goal: Research and evaluation of the effectiveness of algorithms of operating systems adapted to microprocessor architectures.

Subject of research: Analysis of architectural features of various microprocessors and their influence on the functioning of operating systems.

The results

The results of the master's research determined the optimal algorithms and strategies for managing the resources of operating systems for various microprocessors, which allows for the development of specific recommendations for their optimization and practical implementation.

Conclusion

As a result of the p research algorithmic analysis of microprocessor bases of operating systems, key factors influencing their efficiency and interaction with the hardware environment were identified. The study of architectural features of microprocessors made it possible to determine optimal algorithms and resource management strategies, contributing to the improvement of the performance of operating systems on various hardware platforms. The results of the experimental analysis confirm the high potential of optimization, which is expressed in the proposals of specific recommendations for developers and administrators of operating systems.

MICROPROCESSOR, OPERATING SYSTEM, ARCHITECTURAL FEATURES

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ………………………………………… 9
ВСТУП…………………………………………………………………………. 10

РОЗДІЛ 1. ДОСЛІДЖЕННЯ АРХІТЕКТУРНИХ ОСОБЛИВОСТЕЙ МІКРОПРОЦЕСОРА…………………………………………………………….. 12
1.1 Характеристика мікропроцесора та його ролі в сучасних системі……………………..………………………………………. 12
1.2 Тести, розгалуження та адресні режими………………………. 26
Висновки до розділу………………………………………………… 36

РОЗДІЛ 2. ДОСЛІДЖЕННЯ АЛГОРИТМІЧНИХ ОСНОВ РОБОТИ МІКРОПРОЦЕСОРА... 37
2.1 Принципи роботи та функції арифметико-логічного блоку мікропроцесора ……………………………………………………… 37
2.2 Ознаки та конвертація вісімкових та шістнадцяткових чисел…… 38
2.3 Обробка від'ємних чисел в двійковій системі…………………….. 39
2.4 Підпрограми та переривання…………………………………... 56
Висновки до розділу……………………………………………..…… 66

РОЗДІЛ 3. РОЗРОБКА АЛГОРИТМІЧНОГО ЗАБЕЗПЕЧЕННЯ НА МІКРОПРОЦЕСОРНІЙ ОСНОВІ………………………………………………. 67
3.1 Опорні пристрої…………………………………………………….. 67
3.2 Проектування та реалізація структури……………………………. 85
3.3 Етапи та принципи розробки програмного забезепчення………… 95
3.4 Застосування точки зупинки при розробці програмного забезпечення.. 98
3.5 Застосування інтерпретаторів і компіляторів……………………… 103

Висновки до розділу……………………………………………………………. 105

ВИСНОВКИ…………………………………………………………………………. 106
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ………………………………………... 107

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ЦП – центральний процесор

СК – система керування

ALU – арифметико-логічний блок

АЦП – аналого-цифровий перетворювач

ПЗ – програмне забезпечення

ОС – операційна система

ВСТУП

Актуальність роботи. Зростання обчислювальної потужності мікропроцесорів та поширення операційних систем в різних сферах життя створюють потребу в глибокому розумінні алгоритмів, які лежать в основі їхньої роботи.

Розуміння алгоритмів, які використовуються в операційних системах, дозволяє розробникам створювати більш ефективні та оптимізовані програми, які працюють на мікропроцесорах.

Зі зростанням потужності мікропроцесорів з'являються нові можливості для розробки більш складних та продуктивних операційних систем. Аналіз алгоритмів може сприяти створенню ефективних рішень для використання цих можливостей.

В контексті зростання кількості кіберзагроз та атак на операційні системи, розуміння алгоритмів допомагає в розробці більш стійких та безпечних систем.

З розвитком Інтернету речей та вбудованих систем важливо розуміти, як алгоритми взаємодіють з обмеженими ресурсами мікропроцесорів для ефективного управління пристроями та передачі даних.

З розширенням використання хмарних технологій важливо аналізувати алгоритми, які використовуються для оптимізації віддалених операційних систем та забезпечення їх ефективної роботи.

Мета і задачі дослідження

Метою магістерської роботи є глибоке вивчення та аналіз алгоритмів, що лежать в основі мікропроцесорних основ операційних систем. Робота спрямована на розкриття та розуміння принципів функціонування цих алгоритмів, а також їхнього впливу на продуктивність та ефективність операційних систем в різних умовах.

Мета магістерської роботи визначає необхідність виконання таких **завдань:**

- дослідити архітектурні особливості мікропроцесора;
- дослідити алгоритмічні основи роботи мікропроцесора;
- описати етапи розробки прогамного забезпечення на мікропроцесорній основі;
- описати принципи розробки прогамного забезпечення на мікропроцесорній основі.

Предметом дослідження є алгоритмічний аналіз мікропроцесорних основ операційних систем, зокрема їх структура, ефективність та вплив на функціонування операційних систем в різних умовах.

Методи дослідження

Для вирішення завдань дослідження використано методи аналізу літературних джерел,експериментальних досліджень, порівняльного аналізу, математичного моделювання.

Наукова новизна отриманих результатів виявляється у глибокому аналізі алгоритмів мікропроцесорних основ операційних систем та комплексному порівнянні їх ефективності, що приводить до визначення оптимальних стратегій використання для максимальної продуктивності.

Практичне значення магістерської роботи полягає в конкретних рекомендаціях щодо оптимального програмного забезпечення систем, що сприяє підвищенню продуктивності та ефективності комп'ютерних систем. Ці результати можуть бути застосовані в розробці операційних систем, програмного забезпечення та архітектур комп'ютерів для оптимізації роботи на різноманітних пристроях та умовах експлуатації.

Структура магістерської роботи. Робота складається зі вступу, трьох розділів та висновків. Загальний обсяг роботи становить 109 сторінок, і містить 33 рисунки та список використаних джерел із 40 найменувань.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ АРХІТЕКТУРНИХ ОСОБЛИВОСТЕЙ МІКРОПРОЦЕСОРА

1.1 Характеристика мікропроцесора та його ролі в сучасних системах

Мікропроцесор — це велика інтегральна схема (LSI), яка містить складну цифрову логіку, необхідну для центрального процесорного блока (CPU) цифрового комп'ютера. Набір, можливо, з двох або три пристрої LSI, що утворюють центральний процесор,який також можна назвати мікропроцесор.

Основна характеристика та роль у сучасних системах Мікропроцесор — це велика інтегральна схема (LSI), яка містить складну цифрову логіку, необхідну для центрального процесорного блока (CPU) цифрового комп'ютера. Набір, можливо, з двох або три пристрої LSI, що утворюють центральний процесор,який також можна назвати мікропроцесор.

Сам по собі мікропроцесор практично марний і інші схеми потрібно додати навколо нього, щоб створити робочий цифровий комп’ютер.

На рис.1.1 показано основні елементи, з яких складається будь-який цифровий пристрій від маленького мікрокомп’ютера, як-от ZX81, до великого мейнфрейм, наприклад DEC VAX11.

Серцем комп’ютера є центральний процесор, який містить усі логічні схеми, які керують синхронізацією та роботою комп’ютера і виконує арифметичні та логічні функції.

Поза ЦП є розділ пам'яті, який зберігає дані для обробки, а також містить список інструкцій, що називається Програма, яка вказує процесору, що робити.

Комп’ютер повинен мати можливість спілкуватися із зовнішнім середовищем світу, і це робиться через вхідні та вихідні канали, які називаються порти.

Ці порти дозволяють комп’ютеру приймати дані з різних джерел введення, такі як перемикач або клавіатура, а також виводити дані на принтер, плотер або блок візуального відображення (VDU).

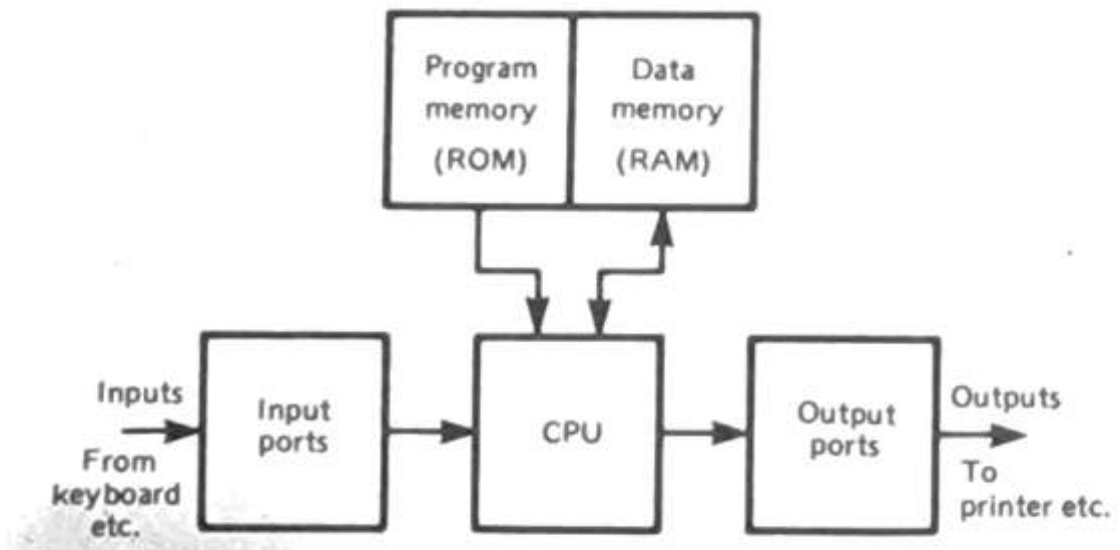

Рис. 1.1. Загальна організація цифрової обчислювальної системи

1.1.1 Відмінності мікрокомп’ютера і мікропроцесора.

Назва мікрокомп'ютер використовується для опису будь-якої малої системи, яка використовує мікропроцесор як центральний процесор. Зазвичай мікрокомп'ютер використовує кілька інтегральних схем, які називаються в народі як «чіпи», тому що кожен побудований на крихітній часточці або «чіпі». Іноді всі компоненти для мікрокомп’ютера містяться в одній інтегральній схемі і цього типу пристрій називають однокристальним мікрокомп'ютером, щоб відрізнити його від більш поширеного мікропроцесорного пристрою.

Однокристальний мікрокомп'ютер містить управління і арифметичні схеми, що утворюють центральний процесор, деяка пам’ять для даних інструкції та набір вхідних і вихідних сигнальних ліній для спілкування із

зовнішнім світом. Однокристальний мікрокомп'ютер ідеально підходить для використання в невеликих спеціалізованих системах, оскільки вони виробляють дуже просту апаратну систему та зменшитують розмір і вартість цілого продукту.

Приклад застосування однокристальної мікроЕОМ знаходиться в знайомому електронному кишеньковому калькуляторі. Інші поширені контролери застосовуються в електронних іграшках і побутовій техніці.

1.1.2 Біт. Основні аспекти та роль у цифрових технологіях

Біт, що є абревіатурою для двійкової цифри, є найменшим елементом даних, який обробляється мікропроцесором.

Біт даних може мати один із двох станів, які називаються «0» і "1". Альтернативні назви для стану 0 — низький або хибний. Так само 1 стан можна назвати високим або справжнім.

Використання біта даних із двома станами ідеально підходить для електронних систем, таких як мікропроцесор, де для одного стану є схема увімкнуто, а для іншого стану ланцюг вимкнено. Логіка схеми сигналу з двома станами називається двійковою (два) логікою та всі доступні на даний момент мікропроцесори та мікрокомп'ютери використовують його.

У типовому мікропроцесорі стан 0 позначається символом а рівень напруги між OV та +0,5 В, тоді як стан 1 має а рівень напруги, можливо, від +3 В до +5 В.

1.1.3 Word Data: сутність та роль у інформаційних технологіях

Один біт даних здатний представляти лише числа 0 і 1. In щоб працювати з числами, вищими за 1, може бути набір бітів даних згруповані разом, щоб утворити слово даних. Якщо розглядати слово складається з чотирьох бітів даних, буде 16 різних комбінацій станів 0 і 1 для чотирьох бітів, взятих як група. Ці 16 комбінацій можна використовувати для представлення 16

різних чисел, наприклад від 0 до 15. Будь-яка кількість бітів може бути об’єднана разом для формування даних слово, і чим більша кількість бітів, тим більшим буде діапазон різних чисел, які можна представити словом. Оскільки окремі біти мають два стани, кожен додатковий біт подвоюється кількість можливих комбінацій. Таким чином, 8 біт дадуть 256 значення, 9 біт дають 512, а 16 біт забезпечать понад 65000 значень.

У мікропроцесорних системах загальна довжина слів зустрічається 4, 8, 16 і 32 біти, хоча є один або два процесори, які використовують 12-бітне слово даних.

1.1.4 Байт: пояснення та значення у сфері обчислювальних технологій

8-бітове слово даних отримує спеціальну назву байт. Спочатку термін байт не означав конкретно 8-бітове слово, а загальне використання тепер зробило це стандартним терміном для посилання на 8-розрядне data word.

Байт може приймати будь-яке числове значення від 0 до 255 і є зручною довжиною слова для представлення текстових символів. Типовий текстовий набір символів міститиме близько 96 верхніх і нижніх регістрів символи, включаючи цифри та спеціальні знаки. Крім того, може бути 32 різних керуючих коду, що дає в цілому 128 кодів символів, які можуть бути представлені 7 з 8 біт байта.

Назва nybble, яку іноді записують як nibble, широко використовується для опису 4-бітного слова (півбайта). Цей термін також може використовуватися для опису 4-бітового сегмента слова. Таким чином, байт можна розділити на верхній і нижній блоки.

Інструкції для процесора складаються з рядка чисел які зберігаються як двійкові слова даних у розділі пам’яті мікрокомп'ютерна система. Кожна інструкція містить код операції або код операції, який визначає тип операції ЦП повинен працювати, коли виконується інструкція. Типові операції можуть бути LOAD, STORE, ADD, INCREMENT і так далі.

Кожен інший код операції представлений у пам’яті унікальним таким чином ми можемо мати число 96 для інструкції LOAD тоді як 97 може бути інструкцією STORE.

Крім коду операції більшість інструкцій також матимуть операнд, який займає наступне одне або два слова в пам'яті після слова коду операції. Цей операнд може представляти дані для обробки за інструкцією або адресою, яка надає розташування в пам'яті деяких даних або іншої інструкції.

Хоча код операції інструкції є лише числом у пам'яті, зазвичай для цього використовують мнемонічний код коли програма написана на папері. Це робить програму простіше для розуміння людиною-програмістом. Таким чином код операції для «Завантажити число в акумулятор» можна записати як LDA. Так само дані та адреси, пов’язані з кодом операції також можуть бути надані назви, щоб полегшити програму.

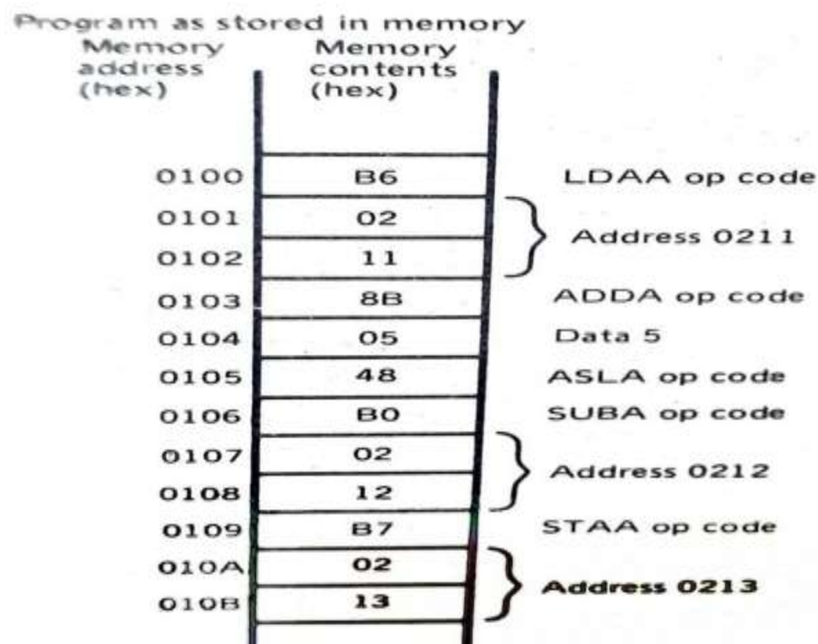

Рис.1.2. Типовий сегмент програми машинного коду для типу 6800

Загальна ідея показана на рис. 1.2, де показано мнемонічний список і фактичний вміст пам'яті для розділу програма для мікропроцесора Motorola MC6800. У мікропроцесорі загального призначення їх може бути від 50 до 200

різні коди операцій для визначення доступних операцій. На практиці там мікропроцесор, який показує, як він зберігається в пам'яті, може бути кількома різними версіями інструкції, наприклад LOAD, кожен має інший код операції та працює по-різному при виконанні.

Для більшості типів процесорів є 8-розрядне слово код операції інструкції, але деякі процесори використовують 16-бітні слова для програмних інструкцій. У цих 16-розрядних процесорах код операції займає лише частину повного командного слова, а інші частини слова можуть визначати різні параметри та надавати інформацію про адресу.

1.1.5 Вплив довжини слова на роботу мікропроцесора: аналіз та перспективи

Для простих програм керування, таких як машини, що працюють на монетах, іграшки, калькулятори та контролери приладів 4-розрядне слово дуже зручний. Слово такої довжини може легко надати цифри від 0 до 9 плюс 5 або 6 додаткових кодів, які можуть визначати вид операції, яку необхідно виконати. Багато комп’ютерів, такі як серія Texas Instruments TMS1000, використовують 4-розрядне слово для даних, хоча інструкції все ще можуть використовувати 8-розрядні слова.

Для програм, що включають обробку текстової інформації та мікропроцесори, такі як Серії Intel 8085, Motorola 6800 і 6809, Zilog Z80 і MOS Technology серії 6500 використовує 8-бітні слова для обох даних та інструкції.

Для настільних комп'ютерів загального призначення або персональних комп'ютерів зазвичай є 8-розрядне слово даних. Коли процесор буде використовуватися у великій системі або має дуже складні обчислення для виконання є перевага у використанні 16-бітна або навіть 32-бітова довжина слова.

Перевагою використання більшого слова є те, що тоді як 8-бітові процесори зазвичай використовують два або три байти для кожної інструкції а

16- або 32-бітна машина може використовувати лише одне слово, що економить час налаштування та виконання інструкцій. Загалом, довші машини довжини слова, як правило, швидше працюють і більш ефективні.

1.1.6 Архітектура мікропроцесора: вплив на функціональність та ефективність систем

Внутрішній пристрій мікропроцесора або мікрокомп'ютера називається його «архітектурою». Логіка мікропроцесора надзвичайно складна, тому діаграма архітектури зазвичай показує лише основні функціональні блоки логічної системи. Діаграма покаже всі робочі регістри, які можуть використовуватися як інструкції в програмі.

На рис. 1.3 показана архітектура типового загального призначення мікропроцесора, схожого на той, який може використовуватися в популярних комп'ютерних системах, таких як APPLE, PET або Sinclair ZX81.

Логіку можна умовно розділити на одну частину, яка контролює послідовність операцій та інша, яка обробляє дані виконання арифметичних або логічних операцій. Ці два розділи тісно пов'язані між собою і в деяких випадках їх функції мають тенденцію до перекриття.

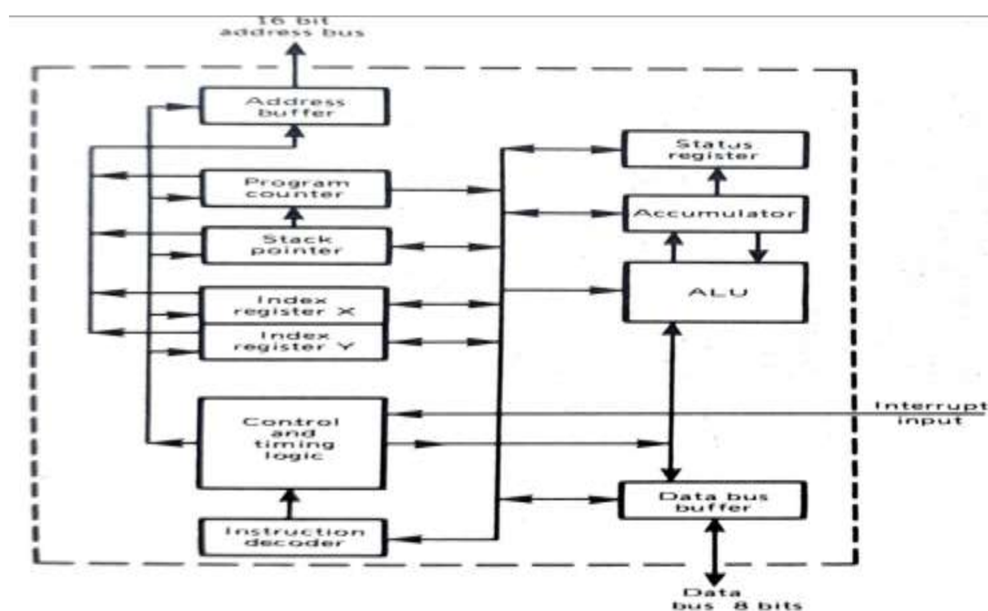

Рис.1.3. Внутрішня архітектура типового 8-розрядного мікропроцесора

В основі секції керування лежить регістр, який називається програмним лічильником. Тут може бути корисно пояснити, що регістр — це логічна схема, яка зберігає стани бітів даних одного слова даних. Регістр лічильника програми використовується для зберігання адреси ділянки пам'яті, яка містить код операції для наступної інструкції, яка має бути виконана. Дані коду операції зберігаються в регістрі інструкцій під час виконання, а декодер інструкцій встановлює необхідну логічну послідовність у ЦП для виконання необхідної операції.

Інші частини системи керування включають стек або регістр покажчика стека, регістр індексу або регістр покажчика даних і деяку логіку для обробки операцій переривання, де зовнішній; сигнал можна використовувати для впливу на потік виконання програми.

Обробка даних виконується арифметико-логічним блоком (ALU), який працює разом із регістром, що називається накопичувачем. Деякі процесори, наприклад серія Motorola 6800, мають два накопичувальних регістри. З акумулятором пов'язаний регістр стану, в якому окремі біти використовуються для вказівки стану процесора.

Зв'язок між центральним процесором, пам'яттю та портами введення/виведення здійснюється через серію шин, які є паралельними групами проводів. Зазвичай існують три системи шин, які називаються шиною даних, шиною адреси та шиною керування. Дані надходять до ЦП і з нього через шину даних, тоді як шина адреси передає сигнали від ЦП, які вибирають конкретне місце пам’яті або канал введення/виведення та підключають його до шини даних. Шина керування, як випливає з її назви, передає набір сигналів керування та синхронізації між різними частинами системи.

На рис. 1.4 показана типова архітектура невеликого мікрокомп’ютера, такого типу, який можна використовувати в кишеньковому калькуляторі. Тут мікросхема містить окремі пам'яті для програми та для зберігання даних. Існують також додаткові регістри для обробки вхідних і вихідних сигналів.

Інша частина логіки, яка може бути показана на архітектурі, - це тактовий генератор, який забезпечує основну синхронізацію для мікропроцесора. Також може бути показаний регістр адреси, який керує адресною шиною. Багато однокристальних мікрокомп'ютерів мають окремі адресні регістри та шинні системи для пам'яті програм і даних, а адреса даних може бути встановлена програмними інструкціями.

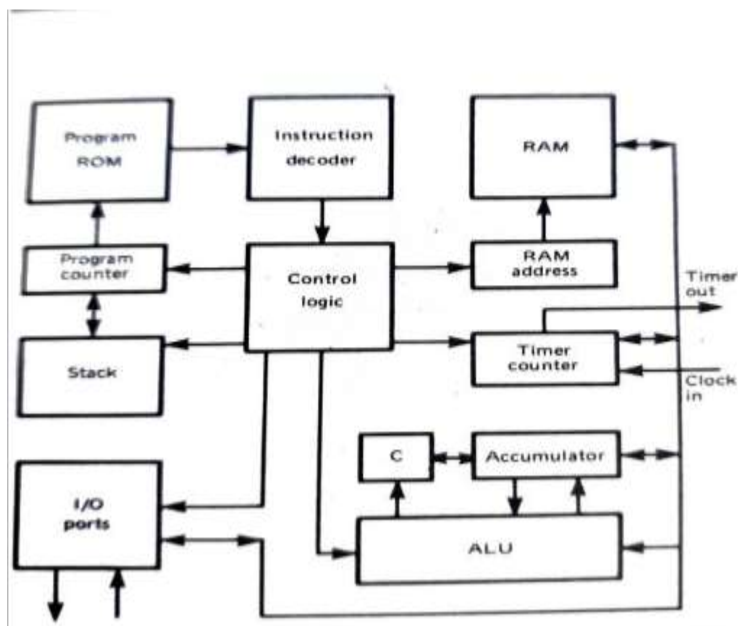

Рис.1.4. Типова архітектура однокристального мікрокомп'ютера

1.1.6 Керування шиною даних у мікропроцесорній архітектурі: організація та ефективність передачі інформації

Оскільки дані можуть надходити до ЦП або з нього, важливо правильно контролювати напрямок потоку даних, і це досягається за допомогою лінії керування читанням/записом (R/W) від ЦП. Коли слово даних має бути надіслано з ЦП, лінія R/W встановлюється в стан запису, і ЦП управляє шиною даних. Якщо слово має бути передане до ЦП, рядок R/W встановлюється на читання, і шиною керуватиме або пам’ять, або лінія введення/виведення.

Більшість мікропроцесорів використовують логіку трьох станів для схем, які керують шиною даних. На додаток до звичайних логічних рівнів 0 і 1 а можна встановити третю умову, яка фактично відключає ЦП від шини даних.

Коли вибрано цей третій стан, ЦП звільняє шину даних і може використовуватися іншими пристроями. Ця схема особливо корисна, коли кілька мікропроцесорів працюють разом і можуть використовувати спільну шину даних.

1.1.7 Роль та функції адресної шини в мікропроцесорних системах

Адресна шина зазвичай керується адресним регістром у центральному процесорі. Сигнал на адресній шині використовується для вибору певного місця в пам'яті або одного з каналів введення/виведення та підключення його до шини даних, готової для передачі даних до або з ЦП.

Логіка декодування адреси буде потрібна для керування вибором пам'яті або пристрою введення/виведення. Це декодування може також використовувати сигнали синхронізації та керування, що виводяться ЦП на його керуючу шину, щоб гарантувати, що синхронізація будь-яких передач даних буде синхронізована з роботою внутрішньої логіки ЦП.

У простій системі потрібно декодувати лише частину адресного слова. Це може означати, що ділянка пам'яті або лінія введення/виведення відповідатиме на будь-який із цілої групи адресних кодів ЦП. Усе буде добре за умови, що ці альтернативні адреси не перекриватимуть адреси інших місць пам’яті чи введення/виведення.

Як і шина даних, шина адреси зазвичай керується схемами з трьома станами. Зазвичай центральний процесор керує адресною шиною, але, вибравши з’єднання з трьома станами, шину можна звільнити для використання зовнішніми пристроями чи іншими процесорами. Одна з проблем, з якою доводиться стикатися при виробництві будь-якого мікропроцесора, полягає в тому, що з чіпа виводиться відносно велика кількість сигнальних проводів.

Для механічної зручності більшість сучасних інтегральних схем, як правило, використовують корпус із двома контактами або DIL. Ряд, штифти, розташовані на відстані 0,1 дюйма (2,54 мм) один від одного, розташовані

вздовж кожного боку упаковки. Відстань між двома рядами становить 0,3, 0,5 або 0,6 дюйма відповідно до розміру упаковки. Загальні розміри корпусів DIL варіюються від 14- та 16-вивідних версій, що використовуються для стандартних логічних мікросхем, до 24- та 28-контактних розмірів і до 40-контактного розміру, який зазвичай використовується для мікропроцесорів. Деякі з найновіших 16-розрядних мікропроцесорів використовують 64-контактний корпус, щоб забезпечити достатні сигнальні з'єднання.

Розмір упаковки залежить від кількості необхідних штифтів, а не від розміру кремнієвого чіпа всередині нього. Менші пакети мають переваги, оскільки вони дешевші у виготовленні та займають менше місця на друкованій платі. Оскільки потрібно з'єднати менше проводів, час і вартість складання друкованої плати скорочуються, а менша кількість паяних з'єднань, як правило, забезпечує більшу надійність.

Одним зі способів зменшення кількості контактів на упаковці є використання набору контактів між шинами адреси та даних за допомогою «мультиплексування». У такій схемі сигнали на цих лініях іноді будуть адресою, а іноді словом даних. Керуючий сигнал від мікропроцесора вказує зовнішнім схемам, чи використовується шина для даних або адреси в будь-який конкретний час. Хоча мультиплексування дійсно спрощує пакет мікропроцесора, воно збільшує складність зовнішньої логіки, яка повинна демультиплексувати сигнали адреси та даних. Декілька типів мікропроцесорів здійснюють мультиплексування сигналів адреси та шини даних через загальний набір пакетних контактів. Деякі 16-розрядні мікропроцесори використовують 8-розрядну шину даних і мультиплексують 16-розрядні комп'ютерні слова як два послідовні байти даних на 8-рядковій шині. Це зменшує кількість пінів, але за рахунок швидкості, оскільки процесору тепер потрібні два цикли інструкцій для передачі повного слова даних.

Першим кроком у виконанні інструкції є розміщення адреси, яка зберігається в програмному лічильнику, на адресну шину. Це вибирає місце в

пам'яті, яке містить код операції інструкції, яку потрібно виконати. Код операції зчитується з пам'яті та передається через шину даних у регістр інструкцій. Код операції декодується логікою керування ЦП, яка потім встановлює внутрішні контури та послідовність синхронізації, необхідну для виконання інструкції. У цей момент програмний лічильник оновлюється, щоб надати адресу наступної інструкції. Деякі інструкції складатимуться просто з коду операції, і в таких випадках програмний лічильник просто збільшується на одиницю. Інша інструкція матиме операнд з одного або двох байтів, який завжди слідуватиме за кодом операції. У таких випадках програмний лічильник збільшується на два або три, щоб вказати на наступний код операції в пам’яті. Іноді операнд інструкції надає адресу наступної інструкції, яку потрібно виконати, а дані операнда передаються безпосередньо з шин даних у регістр лічильника програми. Після того, як код операції декодовано та будь-які операнди оброблені, інструкція виконується, і процес продовжується з інструкцією, вибраною новим вмістом лічильника програми.

Робота ЦП під час виконання інструкції зазвичай займає два або більше тактових періодів ЦП. Перший крок завантаження та декодування коду операції називається циклом отримання інструкції, а останній цикл зазвичай є циклом виконання. Завантаження операндів займе наступні цикли. Деякі процесори, такі як серії 6800 і 6500, мають дуже простий двофазний тактовий сигнал із внутрішніми операціями центрального процесора, що відбуваються під час фази 1, і доступом до шини даних під час фази 2.

Таким чином, проста інструкція використовуватиме два тактові цикли, перший вибирає код операції та другий виконує інструкцію. Інші процесори мають складніші схеми синхронізації, де кожна повна операція складається з кількох циклів інструкцій, що складаються з кількох тактів відповідно до типу дії, що виконується.

1.1.8 Технологія виробництва: основні принципи та процеси у контексті мікропроцесорних систем

Кілька різних методів використовуються для виробництва цифрових піщаних еквітів, але, що стосується мікропроцесорів, MOS, то вони поділяються на дві основні групи, а саме біполярні та (металооксидний кремній).

У біполярних типах використовуються звичайні переходові транзистори, але, на відміну від типових транзисторних радіоприймачів, вони виготовлені з таким невеликим фізичним розміром, щоб розмістити їх на крихітній кремнієвій кришці розміром приблизно 6 мм. Типовий чіп мікропроцесора такого розміру цілком може містити від 5000 до 10 000 транзисторів, і для того, щоб побачити окремий транзистор, знадобиться мікроскоп.

Більшість біполярних мікропроцесорів використовують транзисторно-транзисторну логіку (TTL), яка може бути знайома деяким читачам у формі популярних логічних пристроїв серії 74, які зазвичай використовуються в цифрових логічних системах. Інша популярна логічна схема полягає в тому, що транзистори з’єднані разом за допомогою емітерних схем і називаються логікою з емітерним зв’язком (ECL).

Інтегральні логічні схеми MOS типу використовують польові транзистори для формування логічних елементів. Зазвичай зустрічаються три різновиди логіки MOS, і вони називаються PMOS p-channel MOS, NMOS (n-channel MOS) і CMOS (complementary MOS) NMOS MOS).

У логічних схемах NMOS використовувані польові транзистори мають кремній n-типу для провідного каналу та працюють від позитивної шини джерела живлення. Пристрої PMOS використовують транзистори, у яких канал зроблено з кремнію p-типу, а струми та напруги є від’ємними. Логіка CMOS використовує пари транзисторів PMOS і NMOS у комбінації. Цей тип логіки може бути знайомим, оскільки логічні схеми CMOS серії 4000, які часто використовуються в логіці, були зведені до мінімуму. На відміну від іншої

біполярної логіки MOS, схема CMOS практично не споживає струму, коли вона не перемикається. Загалом логічні пристрої біполярного типу мають вищу робочу швидкість, ніж типи MOS, причому ECL є швидшим за МГц. Типові робочі швидкості для біполярних типів становлять від 10 до 15 у порівнянні з приблизно 1-4 МГц для типів MOS. Одним із недоліків технології hipolar є те, що схеми фізично більші, ніж у логіки MOS. Таким чином, складність пристрою, яку можна досягти, є біполярною для мікросхеми, що використовує технологію типу MOS. Як досягається, є простіші пристрої в результаті використання набору інтегральних схем, а не одного пристрою.

Серед процесів MOS PMOS був першим, хто був використаний для створення мікропроцесорів. Його недоліком є потреба у двох окремих напругах живлення та відносно повільна робота. Пізніші мікропроцесори використовують процес NMOS, який може працювати від однієї шини живлення +5 В і забезпечує найвищу робочу швидкість типів MOS. До недавнього часу лише один або два типи мікропроцесорів були побудовані з використанням процесу CMOS, але тепер доступні версії CMOS багатьох популярних мікропроцесорів NMOS.

Основною перевагою пристрою типу CMOS є те, що він практично не споживає струм, коли логіка не перемикається, і навіть при роботі на повній швидкості він зазвичай споживає менше енергії, ніж його еквівалент PMOS або NMOS. Таким чином, типи CMOS ідеально підходять для використання в портативному або іншому обладнанні, що працює від батарей, де споживання енергії є важливим фактором.

Пристрої типу PMOS мають ту перевагу, що вони можуть працювати з більш високою напругою, ніж типи NMOS і CMOS, і це може бути корисним у деяких додатках, таких як управління вакуумними флуоресцентними цифровими дисплеями. Серед типів MOS найбільш популярні типи NMOS і CMOS, і, ймовірно, типи PMOS з часом зникнуть.

1.2 Тести, розгалуження та адресні режими

Усі мікропроцесори мають певну форму реєстру стану, який показує поточний стан ЦП. Кожен біт у цьому регістрі використовується як прапорець, який сигналізує про наявність певного стану в системі процесора. Іноді цей регістр називають регістром коду прапора або умови.

На рис. 1.5 показано розподіл бітів у регістрі стану мікропроцесора Motorola 6800. Біт 3 — це нуль або біт Z, який, якщо встановлено в 1, вказує, що результат останньої операції був нульовим. Якщо біт Z дорівнює 0, результат не був нульовим. 1 у розряді.

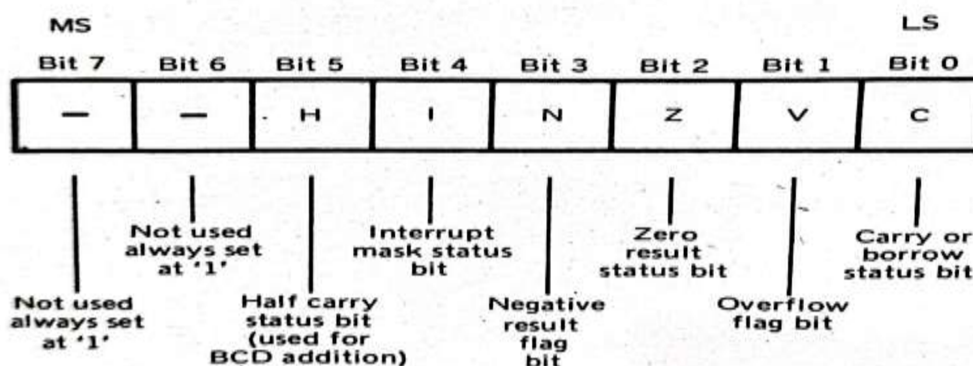

Рис.1.5. Код умови або регістр стану мікропроцесора 6800 і функціональний розподіл його бітів даних

Позиція 4 означає, що результат був від’ємним числом. Цей біт називається «негативним» або N-бітом, і 0 тут означає позитивний результат.

Під час арифметичної операції, наприклад додавання або віднімання, може бути створено перенесення або запозичення, і це вказується С або бітом перенесення. Якщо попередня інструкція включала зсув шаблону бітів в накопичувачі або іншому регістрі, біт С може бути встановлений у стан біта, який був виштовхнутий з кінця регістра. Ще одна умова, яка може виникнути під час арифметичної операції, — переповнення, коли результат операції виходить за межі діапазону чисел, які можуть бути представлені комп’ютерним

словом. Цей стан сигналізується бітом V у регістрі стану 6800. Інші типи процесорів можуть визначати це як O або біт переповнення. Умова переповнення виникає, коли арифметика виконується на числах зі знаком, і для 8-розрядного процесора це означало б, що результат був більшим за 127 або більш негативним, ніж -127. Окрім звичайного біта переносу, 6800 має додатковий біт переносу (H), який вказує на те, що відбулося перенесення з біта 3 на біт 4 під час інструкції додавання. Цей біт H використовується, коли обробляються двійкові десяткові числа (BCD).

Інші біти регістра стану можуть використовуватися для ввімкнення або вимкнення операцій переривання. Процесор 6502 використовує біт регістра стану, щоб змінити свої арифметичні операції зі звичайного двійкового режиму на режим, який обробляє числа BCD.

Деякі біти регістра стану, такі як перенесення, переповнення та переривання, можуть бути безпосередньо встановлені або скинуті командами програми, а також на них впливають результати виконання арифметичних або логічних операцій.

1.2.1 Умовні операції - визначення та важливість в програмуванні

Мабуть, найважливішою властивістю будь-якого мікропроцесора є його здатність приймати рішення щодо послідовності операцій, які мають бути виконані у відповідь на результати його обчислень. Ця функція досягається шляхом проведення перевірок даних або результатів арифметичної чи логічної операції, а потім вибору однієї з двох альтернативних послідовностей інструкцій відповідно до результатів перевірки.

Таким чином, у нас може бути частина програми, яка послідовно віднімає число в пам’яті від числа в накопичувачі, поки результат не буде негативним. Це грубий спосіб проведення ділення. Після кожного віднімання проводиться перевірка, щоб перевірити, чи є результат негативним. Коли біт N дорівнює 0, програма повертається, щоб виконати ще одне віднімання. Коли біт

N дорівнює 1, програма перейде до іншої точки в послідовності інструкцій, як показано на рис. 1.6.

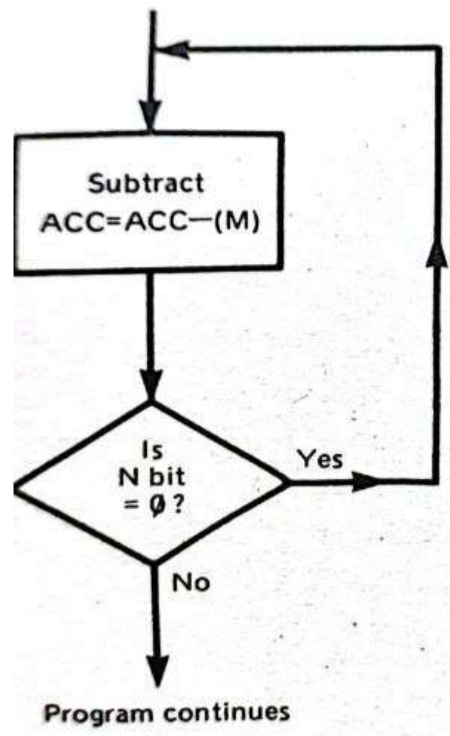

Рис.1.6 Блок-схема, що показує просте використання біта N у циклі віднімання

У всіх процесорах є тест на нульовий результат від останньої операції. Якщо результат був нульовим, виконання програми переходить до вказаної інструкції, тоді як ненульовий результат змушує виконання продовжуватися з наступною інструкцією.

Більшість процесорів надають тест на нульовий результат і тест на негативний результат, і часто надаються зворотні тести, тобто на ненульовий результат і на позитивний.

Усі процесори забезпечують перевірку стану переносу, і розгалуження може бути здійснено або для набору переносу (1), або для очищення переносу (0). Більшість процесорів загального призначення мають тест на арифметичне переповнення, де результатом є число поза межами діапазону акумулятора. Цей тест дозволяє усунути потенційні арифметичні помилки.

Також можуть бути тести, що включають комбінації двох або більше бітів стану, наприклад більше або менше, а також логічні перевірки, наприклад вище і менше. Вони також можуть бути поєднані з нулем, щоб умова перевірки могла бути, наприклад, «Нижчим» або «Те саме значення», коли порівнюються два числа.

У багатьох процесорах вміст регістра стану може бути переданий до накопичувача та з нього, що дозволяє маніпулювати бітами стану за потреби.

Операції пропуску, розгалуження та переходу відрізняються між собою за їхнім призначенням та впливом на виконання програмного коду. Основні відмінності між операціями Skip, Branch і Jump полягають у способі обчислення адреси для наступної інструкції, яка має бути виконана.

Почнемо з інструкції SKIP, яка в основному дуже проста. Якщо перевірка, пов'язана з інструкцією пропуску, виявляється вірною, тоді виконання програми пропускає наступну інструкцію в послідовності до наступної. Як приклад розглянемо наступну програмну послідовність

SKIP Z

JMP LABEL 2

LABEL1 LDA ITEM1

....

....

LABEL 2 LDA ITEM2

....

Коли виконується інструкція SKIP, якщо нульовий біт статусу встановлено на 1, тоді виконання переходить через наступну інструкцію до інструкції, позначеної LABEL1, і в накопичувач. Якщо біт статусу Z дорівнює 0, тоді буде виконано інструкцію JUMP після SKIP, що спричинить прямий перехід до інструкції, позначеної LABEL 2, і наступною операцією є завантаження ITEM2 в накопичувач.

Більшість випадків використання інструкції SKIP дотримується цього шаблону, хоча в деяких випадках може бути можливим виконати необхідну альтернативну дію в інструкції, наступній за інструкцією SKIP, а потім виконання продовжується в послідовності. У такому випадку умова пропуску просто змушує процесор пропускати одну операцію, коли виконується умова перевірки. Прикладом цього може бути ситуація, коли всі числа мають бути додатними для наступних операцій. Тут пропуск перевіряє на позитивний результат, і якщо це не так, наступна інструкція заперечує число, щоб зробити його позитивним. Якщо перевірка пройшла успішно, інструкція заперечення буде пропущена.

У інструкції JUMP операнд інструкції визначає безпосередньо адресу інструкції, до якої має бути здійснений перехід. Таким чином, у наведеному вище прикладі операнд LABEL2 інструкції JUMP буде адресою пам'яті інструкції, позначеної LABEL2. Деякі типи процесорів, такі як серія 6800, допускають лише безумовний стрибок, коли стрибок виконується незалежно від стану бітів у регістрі стану. Інші процесори, такі як типи 8085 і Z80, забезпечують умовні переходи, коли, якщо задана умова задовольняється, здійснюється переход до адреси, заданої операндом, але в іншому випадку операція продовжується з наступною інструкцією в послідовності.

Інструкція типу BRANCH подібна до переходу, але метод адресації використовується інший. Інструкції розгалуження використовують режим, відомий як відносна адресація, у якому адреса інструкції, до якої переходить програма, обчислюється відносно поточної точки в програмі, а не абсолютно визначена інструкцією розгалуження.

Тип адресації, який використовується для інструкції розгалуження, відомий як «відносна адресація». Тут операнд інструкції розгалуження надає те, що називається зміщенням адреси. Коли обчислюється ефективна адреса (EA), яку фактично використовуватиме ЦП, це буде можна зробити шляхом додавання зміщення до поточного вмісту програмного лічильника, а потім

розміщення результату в програмному лічильнику, щоб надати нову адресу для наступної інструкції, яка буде виконана.

Зсув зазвичай розглядається як двійкове число зі знаком, так що розгалуження можна робити як вперед, так і назад у програмі. Якщо зміщення від'ємне, виконання програми розгалужується до точки перед інструкцією розгалуження в послідовності програми.

Позитивне зміщення надішле виконання програми вперед. Для 8-розрядного процесора зсув зазвичай знаходиться в діапазоні від -127 до +127. Деякі процесори можуть також надавати довгу інструкцію розгалуження, де зсув використовує два байти після коду операції. У цьому випадку гілка може простягатися від -32767 до +32767 відносно поточної адреси лічильника програми.

Під час обчислення адреси інструкції, до якої розгалужується виконання, важливо пам'ятати, що початкова точка (нульовий зсув) є адресою коду операції для наступної інструкції в послідовності після інструкції розгалуження.

Для 8-розрядного процесора адресна шина зазвичай має ширину 16 біт, що дає діапазон адрес пам'яті 65536 байт. Якщо використовується інструкція переходу, повна 16-бітна адреса повинна бути вказана як операнд після коду операції інструкції. Оскільки слова даних мають ширину 8 біт, це означає, що два байти повинні бути прочитані з переходу, але пам'ять перед переходом може бути виконана. Таким чином, інструкція повинна займати не менше трьох машинних циклів для виконання.

Інструкція розгалуження, з іншого боку, має відносне зміщення адреси, яке може бути визначено однобайтовим операндом і, отже, виконуватиметься швидше, ніж інструкція переходу. Гілка також використовує на один байт програми менше, що призводить до ефективнішого використання програмної пам'яті.

Крім переваги швидкості виконання та більш компактного програмного коду, є більш важлива перевага відносної адресації. З інструкцією JUMP адреса, на яку програма переходить, абсолютно визначена операндом інструкції. Однак для BRANCH адреса призначення обчислюється відносно поточної адреси в програмі. Частину програми, яка використовує лише інструкції розгалуження, можна завантажити будь-де в пам'ять, і вона працюватиме правильно. Таким чином можна писати сегменти програмного коду, які є позиційно незалежними.

Більшість процесорів мають ряд регістрів у ЦП, і вони можуть бути визначені самим кодом операції. Такий спосіб адресації називається неявним або властивим. Таким чином, у процесорах серії 6800 є два акумулятори, які можна вказати за допомогою таких інструкцій, як INCA Збільшення вмісту ACC A DECB Зменшення вмісту ACC B, де акумулятор, який буде використовуватися, визначено в інструкції. У Z80 до регістрів D, E, H і L так само можна отримати прямий доступ за допомогою коду операції.

Для багатьох інструкцій, особливо тих, що включають арифметику чи логіку, може знадобитися конкретне значення даних для роботи, і зручно зберігати ці дані як операнд інструкції. Таким чином, якщо ми хочемо додати число 10 до деяких даних у пам'яті, програма може бути

......

LDA DATA

ADDA £1Ø

STA RESULT

....

Тут DATA – це адреса пам'яті, де зберігається число, а РЕЗУЛЬТАТ – місце, де буде збережено відповідь. Знак £ перед 1Ø означає, що це фактичні дані, а не адреса. Такий спосіб адресації називається негайною адресацією.

Режим негайної адреси часто використовується для встановлення бітових шаблонів і може використовуватися під час порівняння результатів із заданими граничними значеннями. Негайна адресація також може

використовуватися для завантаження конкретної адреси. Як приклад, LDX £ NUMBER означає завантажити в індексний регістр X адресу, за якою зберігається NUMBER, але NUMBER LDX спричинить завантаження вмісту комірки пам'яті NUMBER в індексний регістр.

Коли дані передаються в пам'ять або з пам'яті, найпростішим методом надання адресної інформації є збереження адреси в байті або байтах, які слідують безпосередньо за кодом операції в пам'яті. Це називається прямою адресацією.

Коли використовується 8-розрядний процесор, якщо адреса вказана одним байтом, тоді вона може мати значення від 0 до 255. Для покриття повного діапазону адрес у 64 Кб слів, дозволеного 16-розрядною шиною адреси, необхідно використовувати два байти. для адресної інформації. Цей режим двобайтової адреси часто називають розширеною адресацією, але в основному два байти після коду операції визначають 16-бітну пряму адресу. У 8-розрядних мікропроцесорних системах пам'ять можна вважати поділеною на сторінки, кожна з яких складається з 256 слів. Тепер старший байт 16-бітної адреси пам'яті дасть номер сторінки, а нижній байт визначає позицію на сторінці. Для першої сторінки в пам'яті (Сторінка 0) старший байт дорівнює нулю, тому для визначення адреси пам'яті потрібно лише 8 біт. Більшість процесорів мають режим адреси нульової сторінки, який використовує однобайтову адресу замість звичайної двобайтової прямої адреси. Деякі типи, такі як 6809, дозволять цей однобайтовий режим адреси для будь-якої вибраної сторінки в пам'яті. Тут прямий регістр сторінки в ЦП встановлюється програмою на номер сторінки в пам'яті, де має використовуватися однобайтова адресація. Головна перевага режиму адресації прямої сторінки або нульової сторінки полягає в тому, що, оскільки потрібен лише один байт, читання інструкції займає менше часу, пам'яті і тому швидше виконується. У 16-бітній процесорній системі адреса може бути включена в командне слово. Якщо в

інструкції вказано фактичну адресу, яка буде використана, це все одно буде пряма адресація.

Досить складніший спосіб адресації відомий як непрямий. У цьому режимі операндом інструкції є адреса місця в пам'яті, яка сама містить адресу, яку має використовувати інструкція. Таким чином, для інструкції переходу операнд дає адресу комірки пам’яті, яка містить як дані адресу наступної інструкції, яка має бути виконана.

Перевага непрямої адресації полягає в тому, що ефективна адреса, яка буде використовуватися інструкцією, може бути змінена під час виконання програми, тоді як пряма адресація абсолютно визначається операндом інструкції програми, заданим, коли була складена програма. Існує кілька 8-розрядних процесорів із можливістю непрямої адресації, але цей режим досить поширений серед 16-розрядних типів.

Більшість мікропроцесорів мають спеціальний регістр, званий індексним регістром, і це дозволяє нову форму адресації, яка називається індексованою адресацією. Коли інструкція вказує індексовану адресацію, вміст регістра індексу додається до адреси операнда для отримання ефективної адреси, яка фактично використовується під час виконання інструкції. Цей режим адресації особливо корисний при роботі з таблицями даних, що зберігаються в пам'яті.

Вмістом індексного регістра можна керувати так само, як і вмістом будь-якого іншого регістра загального призначення, і фактично індексний регістр також можна використовувати як регістр загального призначення в більшості мікропроцесорів. Інкремент і декремент індексного регістра іноді можна комбінувати з іншою інструкцією для надання режиму, який називається автоіндексуванням. До таблиці чисел можна отримати прямий доступ за допомогою такої інструкції, як *LDA START, X ,* де START - це адреса початку таблиці даних, а X вказує на індексовану адресацію за допомогою регістра X. Число в індексному регістрі (X) визначає, наскільки далеко внизу таблиці розташовані дані. Таким чином число можна ввести з клавіатури і розмістити в

регістрі X, щоб вказати позицію в таблиці даних, які будуть використовуватися інструкцією.

Деякі процесори мають два індексні регістри (IX і IY), що полегшує роботу з двовимірною таблицею з одним індексним регістром для рядків, а іншим для стовпців. Все ще можна працювати з такою таблицею, використовуючи один індексний регістр, але вміст потрібно буде перенести до пам’яті та з неї, де можна використовувати два окремі місця для відстеження поточного рядка та інформацію про стовпці.

Якщо той самий процес має використовуватися з кожним записом у таблиці даних, та сама послідовність інструкцій повторюється для кожного запису, а потім індексний регістр збільшується, щоб вказати на наступний елемент у таблиці. Зазвичай це робиться за допомогою операції типу циклу.

1.2.2 Цикли в програмуванні: означення та принципи роботи

Майже в усіх програмах будуть випадки, коли послідовність інструкцій повинна бути виконана кілька разів поспіль. Фактична послідовність може коливатися від однієї чи двох інструкцій до повного розділу програми. Цей тип операції називається циклом.

Перед тим, як буде досягнуто розділ циклу, регістр або місце пам'яті встановлюється на кількість разів, які цикл має бути виконано. Далі починається сам цикл, і перша інструкція позначається або зазначається її адреса. Основна частина циклу подібна до будь-якої іншої частини програми. Наприкінці послідовності циклу кількість, що зберігається в регістрі або пам’яті, зменшується, а результат перевіряється на нуль. Якщо результат не дорівнює нулю, програма повертається до першої інструкції послідовності циклу, яка потім виконується знову. Якщо результат дорівнює нулю, необхідну кількість проходів виконано, і програма продовжує виконання наступної інструкції. Потік основної програми показаний на рис.1.7.

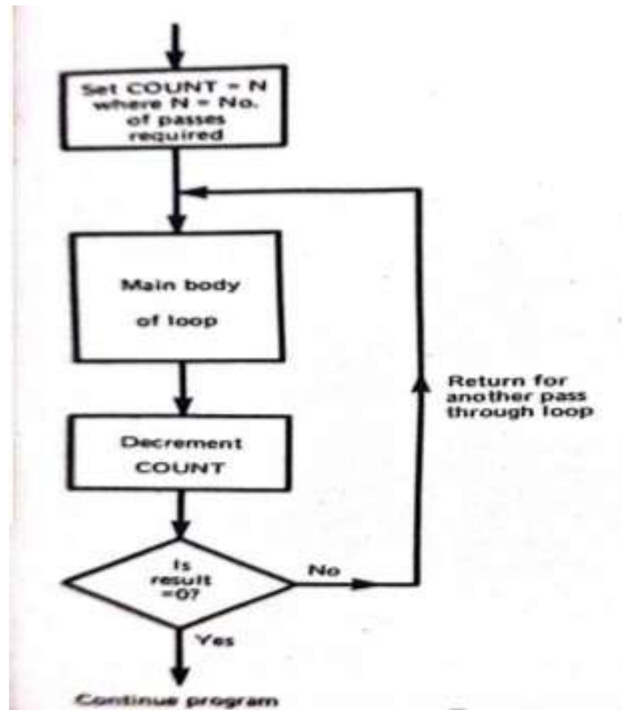

Рис.1.7. Основний процес виконання програми для виконання циклу

Цикл може бути використаний для забезпечення простої затримки часу, і витрачений час буде часом виконання інструкцій у циклі, включаючи інструкції тестування та розгалуження, помноженим на кількість разів, коли цикл виконується.

Висновки до розділу

В першому розділі виявлено ключові компоненти та концепції, що лежать в основі функціонування мікропроцесорів. Отримані знання будуть важливими для подальшого розуміння та розробки програмного забезпечення для мікропроцесорних систем

РОЗДІЛ 2
ДОСЛІДЖЕННЯ АЛГОРИТМІЧНИХ ОСНОВ РОБОТИ МІКРОПРОЦЕСОРА

2.1 Принципи роботи та функції арифметико-логічного блоку мікропроцесора

Практично вся обробка чисел у центральному процесорі виконується в арифметико-логічному блоці (ALU), який працює разом із спеціальним регістром, який називається накопичувачем. Загальне розташування цієї частини центрального процесора показано на рис. 2.1.

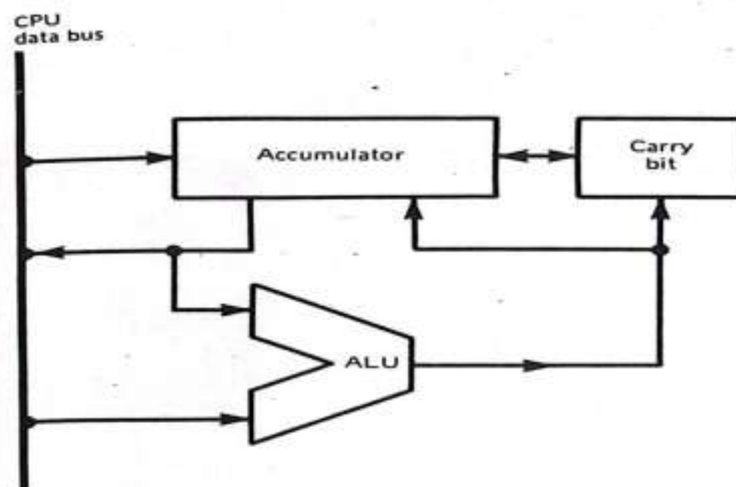

Рис.2.1. Принципове розташування арифметично-логічної частини комп'ютера

Саме ALU є складним логічним масивом, який може виконувати такі операції, як Додавання, Зсув, І, Збільшення тощо. Є два входи даних до ALU, один з яких надходить від накопичувача, а інший може бути отриманий через шини даних, з пам'яті або іншого регістра ЦП. Результат операції записується в накопичувач, замінюючи його попередній вміст. Однорозрядне розширення акумулятора використовується для збереження будь-якого біта переносу, створеного арифметичною або логічною операцією в ALU.

У повсякденному житті ми звикли мати справу з десятковими числами, де кожна цифра, що представляє одиниці, десятки, сотні, тисячі тощо, може мати значення від 0 до 9. Мікропроцесори використовують двійкову (двійкову) систему числення, де цифри (біт) мають значення 0 (вимкнено) або 1 (увімкнено). У двійкових числах кожна цифра або біт позначає одиниці, двійки, четвірки, вісімки, шістнадцятки тощо. Будь-яке десяткове число можна представити у вигляді рядка з 1 і 0 бітів у двійковому числі.

Припустімо, ми беремо число 13. У десятковій формі це представлено як

$$(1 \times 10) + (3 \times 1) = 13$$

У двійковій системі еквівалентним числом буде

$$(1 x 8) + (1 x 4) + (0 \times 2) + (1 x 1) = 1101$$

Таким чином, 1101 двійковий = 13 десятковий. Основні відмінності між двома системами у тому, що для подібного значення двійкове число містить більше цифр і двійкові цифри (біти) можуть мати лише значення 0 або 1.

2.2 Ознаки та конвертація вісімкових та шістнадцяткових чисел

Двійкові числа можна записати у вигляді рядка з Os і 1, але якщо їх більше п’яти бітів, цей підхід стає досить громіздким. Альтернативні та більш зручні методи запису двійкових даних використовують або вісімкове, або шістнадцяткове число. У вісімковій системі кожна цифра має значення від 0 до 7, і ці цифри позначають одиниці, вісімки, шістдесят чотири і так далі.

Тепер кожна вісімкова цифра замінює групу з трьох двійкових бітів, тому двійкове число 101011 можна записати як 53 у вісімковій формі. Для перетворення двійкової у вісімкову двійкові біти беруться наборами по три біти, що працюють справа, і кожен набір перетворюється на число від 0 до 7. Перетворення з вісімкового у двійкове просто передбачає запис кожної вісімкової цифри в її еквівалент трибітного двійкового числа.

У шістнадцятковій системі запису використовується шкала шістнадцять, і кожна цифра представляє групу з чотирьох двійкових бітів. Цифри можуть мати значення від 0 до 15. Тут цифри зі значеннями від 0 до 9 пишуться нормально, але для значень від 10 до 15 використовуються літери від A до F. Таким чином, 10 стає A, 11 — B і так далі до 15, яке стає F. Шістнадцяткові цифри представляють одиниці, 16, 256 і так далі. Перетворення між шістнадцятковим і двійковим в основному таке ж, як і для вісімкового, за винятком того, що чотири двійкові біти стають однією шістнадцятковою цифрою і навпаки. Припустимо, ми розглядаємо число 200 у десятковій системі, яке буде 11001000 у двійковій системі. Для перетворення в шістнадцяткову форму ділиться на групи по чотири біти, напр. (1100) (1000), щоб отримати C8 у шістнадцятковій формі.

Коли записуються шістнадцяткові числа, вони відрізняються від звичайних десяткових чисел записом знака $ перед числом або H після числа. Таким чином, наше шістнадцяткове значення для десяткового числа 200 буде написано $C8 або C8H.

2.3 Обробка від'ємних чисел в двійковій системі

У десятковій системі для позначення від'ємного числа, тобто одиниці нижче нуля, ми просто ставимо знак перед, наприклад, -65. Це так само добре можна зробити з двійковою системою (-1000001), якщо ми пишемо на папері, але потрібні деякі засоби для вказівки від'ємних чисел у комп'ютері.

Якщо у нас є чотирирозрядне двійкове число, яке починається з 0000, результати будуть 0000, 0001, 0010, 0011 тощо. Якщо число зменшується від 0000, перший крок призведе до того, що всі біти зміняться, даючи 1111, а продовження зменшення дає послідовність 1110, 1101, 1100 і так далі. Таким чином, число 1111 представляє -1, а 1110 дорівнює -2, і слід зазначити, що

лівий біт усіх від'ємних чисел тепер дорівнює 1. Цей біт можна розглядати як знаковий біт.

У цих двійкових числах зі знаком перший біт позначає знак, а решта три біти визначають значення, тому для 4-розрядного числа значення може коливатися від +7 (0111) до 0 (0000) до -7 (1001). Значення 1000 є неоднозначним і може розглядатися як +8 або -8, тому може знадобитися перевірка вихідних чисел та їхніх знаків, щоб визначити, яке значення є правильним для цього результату.

Перетворення чисел із додатних у від'ємні чи навпаки відоме як заперечення й досягається зміною стану всіх бітів числа та додаванням 1 до результату. Процес зміни стану всіх бітів називається доповненням. Таким чином, якщо ми візьмемо +1 (0001) і доповнимо його, ми отримаємо 1110, що насправді дорівнює -2. Тепер, додавши до цього 1, ми отримаємо 1111, яке, як ми бачили вище, є двійковим числом зі знаком для -1. Багато процесорів матимуть інструкцію NEGATE, але якщо вона недоступна, того самого результату можна досягти, спершу використавши COMPLEMENT, а потім збільшивши результат.

Додавання двійкових чисел подібне до додавання десяткових чисел, за винятком того, що цифри або біти в кожному числі та в результаті можуть мати лише значення 0 або 1.

Додавання одного біта з кожного числа, яке потрібно додати, дає один із наступних чотирьох результатів

$$0+0=0$$

$$0+1=1$$

$$1+0=1$$

1 + 1 = 0 і перенести 1 до наступного біта

Якщо є 1, перенесена від додавання попереднього біта, то результати стають

$$0+0+C=1$$

0+1+C = 0 і перенести 1 до наступного біта

1+0+ C = 0 і перенести 1 до наступного біта

1 + 1 + C = 1 і перенести 1 до наступного біта

ALU автоматично оброблятиме перенос між бітами, коли він додає два числа, і будь-який перенос із найбільш значущого кінця буде зберігатися в біті переносу регістра стану ЦП. Якщо в програмі слідує наступна інструкція додавання, цей біт переносу буде додано в кінці додавання з найменшим значенням.

Деякі процесори забезпечують два типи інструкцій Add, одна з яких враховує перенесення від попередньої операції, а інша ігнорує будь-який попередній стан перенесення.

Послідовність програм для додавання з використанням процесора 6502 буде виглядати так:

LDA NUMBER1

ADD NUMBER2

STA RESULT

Ця послідовність завантажує ЧИСЛО1 з пам’яті в накопичувач, потім додає з пам’яті ЧИСЛО2, і сума повертається з АЛУ в накопичувач. Нарешті РЕЗУЛЬТАТ зберігається в пам'яті.

Двійкове віднімання відбувається за подібною схемою до двійкового додавання. Існує чотири можливі результати віднімання з використанням одного біта з кожного числа, і ці умови застосовуються до кожного біта чисел по черзі, починаючи з молодшого кінця.

Чотири результати є

0-0=0

1-0=1

0-1=1 і запозичення = 1

1-1=0

якщо запозичення від попереднього віднімання біта дорівнює 1, то результати стають

0-0=1 і запозичення = 1

1-0=0 і запозичення = 0

0-1=0 і запозичення = 1

1-1=1 і запозичення = 1

Для більшості процесорів загального призначення буде інструкція, яка змушує віднімати число від вмісту акумулятора. Ця інструкція може враховувати стан біта переносу, який вона використовуватиме як запозичений вхідний сигнал у молодшому кінці, і будь-яке остаточне запозичення у найбільш значимому кінці після віднімання буде зберігатися як новий біт переносу. Деякі процесори також мають версію інструкції віднімання, яка припускає, що початковий вхід запозичення дорівнює нулю. Типова послідовність інструкцій для віднімання

LDA NUMBER1

SUB NUMBER2

STA RESULT

Деякі процесори не мають інструкції віднімання, але бажаного результату можна досягти шляхом заперечення числа, яке потрібно відняти, а потім додавання його до числа в накопичувачі.

2.3.1 Числа BCD: збереження та використання в обчисленнях

Часто мікропроцесору доведеться обробляти числа, які вводяться як десяткові цифри, скажімо, з клавіатури. Таким же чином числа можуть бути виведені як десяткові цифри на дисплеї або роздруківці.

Перетворення великих чисел із десяткової системи в двійкову й навпаки є відносно складним, і краще, якщо числа можна обробляти по одній десятковій цифрі за раз. Цього можна досягти, використовуючи двійкові кодовані десяткові чи двояково-десятичні числа.

У схемі BCD кожна десяткова цифра перетворюється на чотирибітний двійковий еквівалент, і ці чотирибітні коди потім об’єднуються, щоб утворити число BCD, у якому кожна група з чотирьох бітів є однією десятковою цифрою.

Таким чином, число 725 перетворюється на 0111 0010 0101. Людині-оператору не надто важко прочитати числа, представлені в цьому форматі, і, звичайно, легше, ніж спробувати прочитати чисті двійкові числа.

Форма кодування BCD зазвичай використовується в невеликих системах, де введення здійснюється з клавіатури, наприклад у калькуляторах і касових апаратах.

2.3.2 Обробка чисел BCD центральною процесорною одиницею: методи та призначення

Оскільки кожна десяткова цифра представлена у форматі BCD чотирибітним словом, вона обробляється окремо в 4-бітній мікропроцесорній системі, що робить цей тип ЦП досить зручним для використання з числами BCD. У 8-розрядних процесорах зазвичай в кожне 8-розрядне слово вкладають дві цифри BCD.

Коли дві цифри BCD обробляються разом, потрібні деякі відмінності в операції додавання. Деякі процесори мають спеціальну інструкцію додавання BCD або можуть використовувати інструкцію зміни режиму, щоб процесор автоматично обробляв слово як дві цифри BCD, щоб отримати результат, який є правильним у BCD. Багато процесорів надають цю можливість для додавання BCD, а деякі також можуть виконувати віднімання BCD.

Деякі процесори, такі як серія 6800, не мають можливості прямого додавання BCD, але виконують пряме двійкове додавання. Щоб отримати правильний результат BCD, після додавання слідує додаткова інструкція під назвою DAA (Decimal adjust arithmetic). Ця інструкція DAA використовує половинний біт переносу (H), який виявляє переноси з молодших чотирьох бітів до п’ятого біта. Шляхом додавання коригувального коефіцієнта, який

регулюється станом половини та повного біта переносу, можна отримати правильний результат BCD. Ці процесори не мають можливості віднімання BCD, і це має бути досягнуто шляхом написання спеціальної процедури для віднімання BCD, яка також використовуватиме статус половини та повного біта переносу для визначення поправки, необхідної для простого двійкового віднімання. 16-розрядні процесори можуть обробляти чотири цифри BCD на слово, і більшість із них мають ті чи інші засоби арифметики BCD, які зазвичай охоплюють як додавання, так і віднімання.

2.3.3 Операції SHIFT і ROTATE: визначення та застосування в обчисленнях

Практично всі мікропроцесори мають інструкції, які дозволяють зміщувати шаблон бітів даних у регістрі на одну позицію вправо або вліво. Цей тип операції показано на рис. 2.2. Розглянемо спочатку операцію зсуву вліво, показану на рис. 2.2. На початку (вгорі) в регістрі зберігається число 00110001, яке має десяткове значення 49. Коли виконується зсув вліво, число стає 01100010, яке має значення 98. Зверніть увагу, що всі біти слова переміщуються одночасно. Переміщення візерунка ще на одну позицію ліворуч дасть значення 196 і так далі. Кожна операція зсуву вліво подвоює значення даних. На кроці 4 крайня ліва 1 переходить у біт переносу, вказуючи число більше 256.

Якщо ми тепер подивимося на операцію зсуву вправо (рис. 2.2(b)), то побачимо, що число знову починається зі значення 49. Після першої операції зсуву вправо значення стає 24, що дає ділення на два та а 1 залишок переходить у біт переносу. Процес продовжує виробляти значення 12 після двох змін. Таким чином, зсув вправо поступово ділить значення даних на два для кожного зсуву.

Якщо під час операції зміни біт висипається в положення для перенесення, він буде втрачений під час наступної операції зміни, тому, якщо ці

біти надлишку є важливими, їх потрібно вирішити одразу після інструкції щодо зміни.

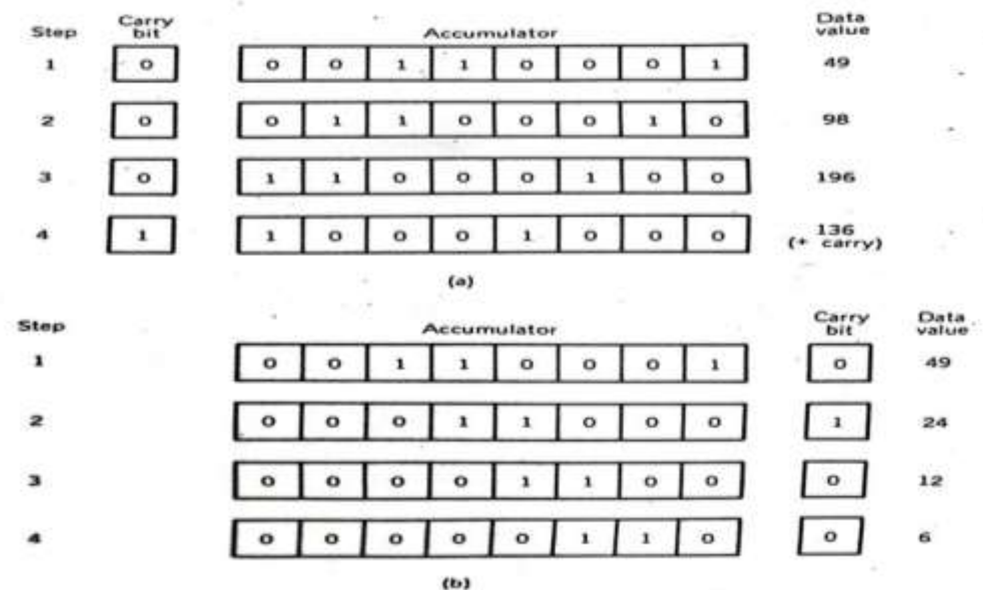

Рис. 2.2. Переміщення даних через акумулятор і перенесення для (а) послідовності операцій зсуву вліво та (б) зсуву вправо

Для операції зсуву вправо новий стан повинен бути призначений старшому біту. Для простої операції логічного зсуву цей біт встановлюється на 0, щоб регістр заповнювався зліва Os. Аналогічно для операції зсуву вліво Os зсуваються з лівого кінця.

Один різний тип зсуву, передбачений у багатьох процесорах, — це арифметичний зсув праворуч (ASR), де, якщо старший біт дорівнює 0, операція виконується так само, як і для логічного зсуву праворуч, але якщо старший біт дорівнює 1, то 1 зсувається праворуч, як зазвичай, але 1 також вставляється у найважливішу позицію, щоб слово поступово заповнювалося одиницями з лівого кінця. Ця, начебто, дивна операція використовується для збереження правильного співвідношення поділу на два для чисел зі знаком, де старший біт є знаковим, а 1 тут вказує на від’ємне число.

За допомогою простих інструкцій зсуву накопичувач або інший регістр заповнюється або Os, або 1s у міру просування процесу зсуву. Іноді потрібно,

щоб дані в регістрі переміщалися через регістр, а біти, які виливаються з одного кінця, потім знову вставляються в інший кінець. Шаблон даних тепер циркулює через реєстр безперервно. Цей процес дозволяє перевіряти кожен біт по черзі, і в кінці дані повертаються на своє початкове місце в регістрі. Цей тип операції називається Rotate.

2.3.4 Логічні операції і їх використання

На додаток до двійкових або BCD арифметичних функцій більшість мікропроцесорів також забезпечать серію булевих логічних операцій. Типовими з них є функції AND, OR і EXCLUSIVE-OR.

В операції AND шаблон бітів у двох словах даних порівнюється побітово. Якщо обидва відповідні біти в словах мають значення 1, результуючий біт у відповіді буде встановлено на 1. Якщо біт в одному або обох словах дорівнює 0, тоді 0 буде записаний у цій позиції біта в слові результату. Це показано на рис. 2.3.

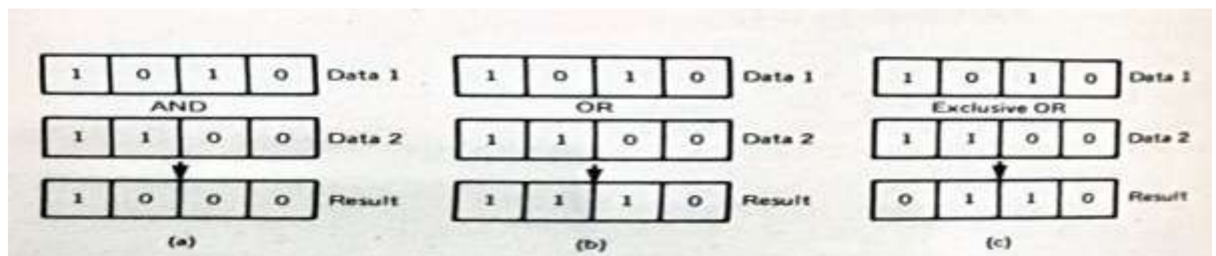

Рис. 2.3. (a) Результати операції І для можливих бітових комбінацій у двох словах даних. (b) Результати операції АБО для можливих бітових комбінацій у двох словах даних. (c) Результати операції EXCLUSIVE-OR для можливих бітових комбінацій у двох data words.

Для операції АБО шаблони бітів двох слів знову порівнюються, але тут, якщо будь-який біт слова дорівнює 1, результуючий біт дорівнює 1, тоді як лише якщо обидва біти мають значення 0, у результаті буде отримано 0. Це показано на рис. 2.3.

Функція EXCLUSIVE-OR створює вихід 1, лише якщо стан біта в одному слові відрізняється від стану біта в другому слові. Якщо обидва біти мають значення 1 або обидва мають значення 0, тоді результуючий біт встановлюється на 0.

Іншою корисною логічною функцією є функція NOT або COMPLEMENT, яка інвертує стан усіх бітів у слові, при цьому 1s стає 0, а Os стає 1. Нарешті, зазвичай є функція CLEAR, яка встановлює всі біти в слові на 0.

Окрім її використання як суто логічної функції, одним із основних застосувань функції AND є надання можливості «маскування» або «вибору». Під час операції маскування будь-які біти в шаблоні даних, які потрібно ігнорувати, можуть бути об’єднані AND з бітом 0, тоді як усі інші біти об’єднані AND проти 1. Це може бути використано, коли цифровий код вводиться з клавіатури в коді ASCII, де лише чотири молодші значущі біти позначають значення числа, тоді як старші біти використовуються для відмінності чисел від букв.

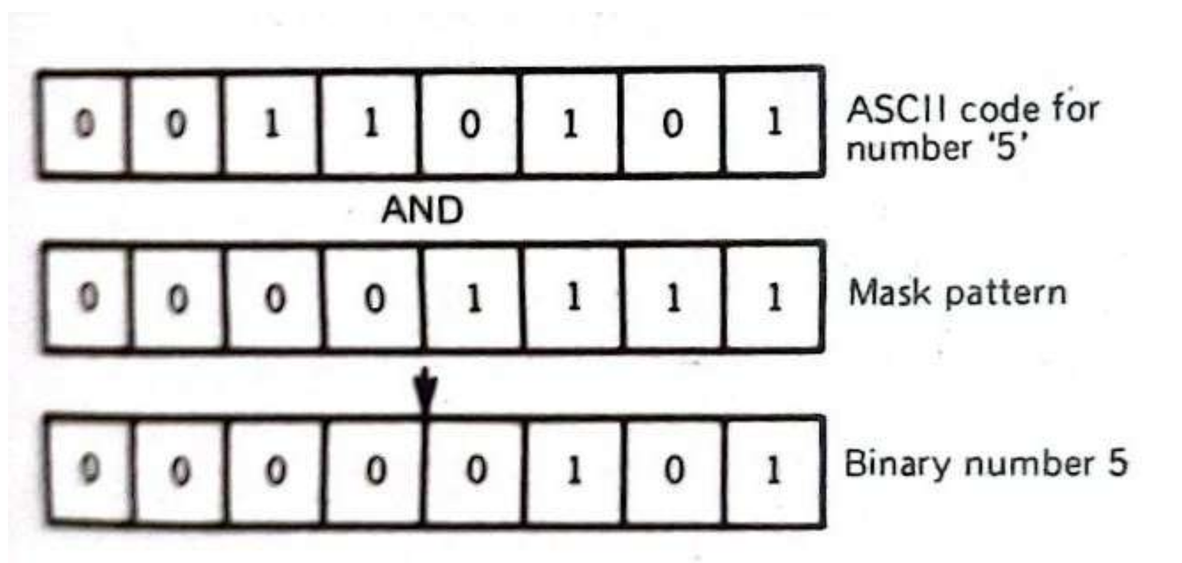

Рис. 2.4. Використання логічної функції І для маскування частини слова даних для перетворення коду ASCII у двійкове число

У цьому випадку, коли дані були прочитані в ньому, було б об’єднано AND із шаблоном 00001111, який ефективно пригнічує старші біти, замінюючи їх на Os, як показано на рис.2.4.

Щоб вибрати окремі біти з шаблону, ці біти слід об’єднати І проти 1 біта, тоді як усі інші біти слова об’єднати І проти Os, щоб придушити небажані біти. Зазвичай функція АБО, окрім її звичайного логічного використання, вибирається там, де певні біти слова мають бути примусово переведені в стан 1 без зміни станів інших бітів у слові. Ця функція може бути використана для перетворення однієї цифри BCD в її еквівалент коду ASCII, готовий для виведення на принтер. Тут додаткові кодові біти, які використовуються для ідентифікації цифр у коді ASCII, будуть встановлені в старших чотирьох бітах слова даних, і це слово буде потім об’єднано АБО з шаблоном цифр BCD, щоб отримати бажаний результат. Це показано на рис. 2.5.

Основне використання функції Exclusive-OR полягає в порівнянні бітових шаблонів у двох словах. Якщо шаблон бітів у кожному слові ідентичний, вхідними даними для операції « Exclusive-OR » будуть пари з 1 або 0 бітів, а на виході всі біти будуть дорівнювати 0. Якщо виникає невідповідність між відповідними бітами у двох словах, 1 буде виведено в цю позицію біта, так що перевірка також покаже, які біти не збігаються.

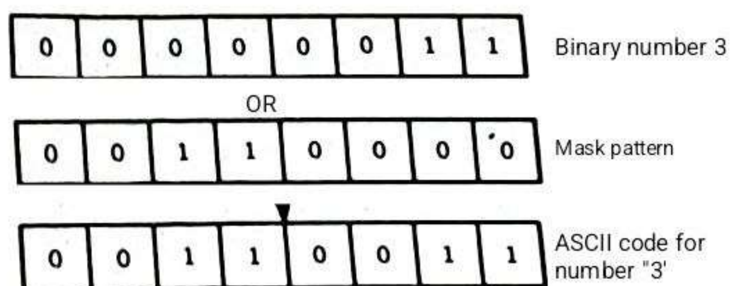

Рис.2.5. Використання логічної функції АБО для вставки додаткових логічних бітів під час перетворення даних двійкового числа у форму коду ASCII

У функції «Exclusive OR», якщо один із вхідних бітів дорівнює 1, тоді вихід буде доповненням до другого вхідного біта. Таким чином, якщо другий біт дорівнює 0, вихід стає 1 і навпаки. Якщо один вхідний біт дорівнює 0, тоді вихід стає станом другого вхідного біта. Цю функцію можна використовувати для вибіркового інвертування бітів у слові даних, встановивши відповідні біти в слові-масці на 1, а всі інші біти на 0. Тепер, виконавши операцію «Виключне АБО» між словом, яке потрібно обробити, і словом-маскою бажані біти можна інвертувати, а інші біти залишаються незмінними. Це показано на рис. 2.6.

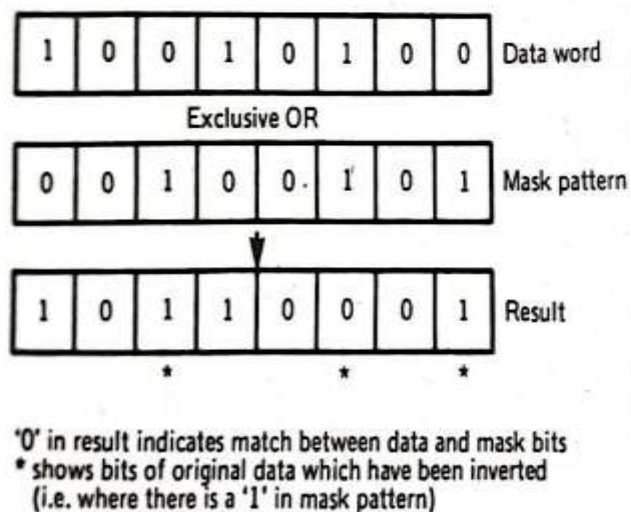

Рис. 2.6 Використання EXCLUSIVE-OR для виявлення збігу між двома словами даних або для вибіркового доповнення окремих бітів слова даних

2.3.5 Множення та ділення двійкових чисел центральною процесорною одиницею: процес та алгоритми

Процес множення набагато складніший, ніж додавання чи віднімання, і фактично є серією додавання. Подумайте, як два десяткові числа перемножуються разом, як показано на рис. 2.7. Перше число, відоме як множене, множиться на молодшу цифру другого числа, яке називається множником, і виходить частковий результат. Потім множене множиться на другу цифру множника, щоб отримати подальший частковий результат і так

далі. Нарешті часткові результати додаються разом для отримання продукту. Зауважте, що для цього додавання другий частковий результат зсувається на одну цифру вліво перед додаванням. Це фактично помножує його значення на 10. Інші часткові результати поступово зсуваються ліворуч перед додаванням до суми.

У двійковій арифметиці процес множення дещо спрощений, оскільки будь-який біт множника може мати лише значення 0 або 1, що означає, що множене буде додано до суми (біт = 1) або ні (біт = 0). Таким чином, біти множника перевіряються послідовно, починаючи з молодшого значущого кінця, і множене або додається до суми, або ні, відповідно до стану біта, що перевіряється. Кожне додавання повинно бути зроблено на один біт ліворуч у сумі, але це зазвичай досягається шляхом зміщення суми на один біт праворуч між додаваннями, що досягає того самого результату. Одна з проблем множення полягає в тому, що добуток завжди буде більшим за множник або множене і фактично зазвичай матиме вдвічі більше бітів.

12345 Multiplicand
678 Multiplier
98760 1st partial product
86415 2nd partial product
74070 3rd partial product
8369910 Product
(a)

11001 Multiplicand (25)
101 Multiplier (5)
11001 1st partial product (25)
00000 2nd partial product (zero)
11001 3rd partial product (100)
1111101 Product (125)
(b)

Рис.2.7 а Десяткове множення Рис.2.7b Двійкове множення

Тому знадобиться другий регістр, щоб утримувати частину продукту, яка переливається з накопичувача. Проста процедура множення двох 8-розрядних чисел за допомогою числа 6800 виглядає так:

```
                                  LDAA £8
                                 STAA COUNT                    Shift product (low)
                                 LOOP
LDAB MPLR                                                      DEC COUNT
                                 NCRY                          Shift product (high)
CLRA
                                 ASRB                          Decrement count
Multiplier to ACCB               RORA PRODL
                                                               BNE LOOP
Set up count of 8                BCC NCRY                      Add multiplicand
                                                               Test for zero
CLR PRODL                        ADDA MPCD
                                                               STAA PRODH
Product (high) = 0               ASRA
                                                               Store product (high)
                                 Product (low) = 0
```

```
                               LDAA £8

                               STAA COUNT              Set up count of 8
                               LDAB MPLR              Multiplier to ACCB
                               CLRA                        Product (high=0
                               CLR PRODL             Product (low)=0
LOOP                           ASRB
                               BCC NCRY
                               ADDA MPCD                   Add multiplicand
NCRY                           ASRA                  Shift product (high)
                              RORA PRODL             Shift product (low)
                              DEC COUNT              Decrement count
                              BNE LOOP              Test for zero
                              STAA PRODH            Store product (hidh)
```

Ділення фактично є процесом послідовного віднімання. Число, яке потрібно поділити, називається діленим, і воно ділиться на дільник, щоб отримати відповідь, яка називається часткою. Якщо ми припустимо, що дробові числа не дозволені, то також буде залишок.

Найпоширеніший спосіб ділення показаний на рис.2.8. Тут ділене спочатку завантажується в накопичувач, а дільник віднімається з нього. Другий регістр або місце пам'яті використовується для зберігання частки, яка спочатку

встановлена на нуль. Якщо результат віднімання додатний, 1 зміщується вліво в найменший кінець частки. Якщо результат був негативним, 0 переноситься в приватне. Якщо отримано негативний результат, ділене повертається до попереднього стану шляхом додавання до нього дільника. Потім ділене зміщується вліво на один біт, що дає такий самий ефект, як ділення значення дільника на два. Потім процес продовжується ще сім разів, якщо числа, що обробляються, є 8-розрядними. Важливий тест має бути проведений до того, як почнеться фактична процедура поділу. Це перевіряє «дільник» нуля. Результат такого поділу має бути нескінченним. На практиці найкраще, що можна зробити, це встановити «частку» на максимальне значення. Процедура ділення передбачає два позитивних числа. Якщо використовуються числа зі знаком, знак результату можна визначити на початку. Якщо знаки дільника та дільника однакові, частка, якщо два знаки різні, має бути сумарною. Визначивши знак результату, від'ємний дивіденд або дільник можна перетворити на додатний шляхом віднімання його від нуля. Деякі процесори мають спеціальні інструкції для простих операцій множення та ділення.

2.3.6 Множинна точність та арифметика з плаваючою комою: означення та застосування

З 8-розрядним процесором діапазон чисел, які може обробляти безпосередньо накопичувач, становить від 0 до 255 або альтернативно від -127 до +127 для чисел зі знаком. Більші числа можна обробляти, розглядаючи число як послідовність 8-розрядних сегментів.

Припустимо, ми вирішили використовувати 24-розрядні числа. Для 8-розрядного процесора це число потрійної точності. Два числа будуть збережені в пам'яті як два набори з трьох 8-розрядних слів, як показано на рис. 2.8. На початку завантажується одне з найменш значущих слів і до нього додається найменш значуща частина другого числа. Цей результат тепер зберігається як найменш значуще слово результату. Далі дві середні частини чисел додаються з

урахуванням будь-якого переносу, створеного першим додаванням, і цей результат зберігається як середнє слово результату.

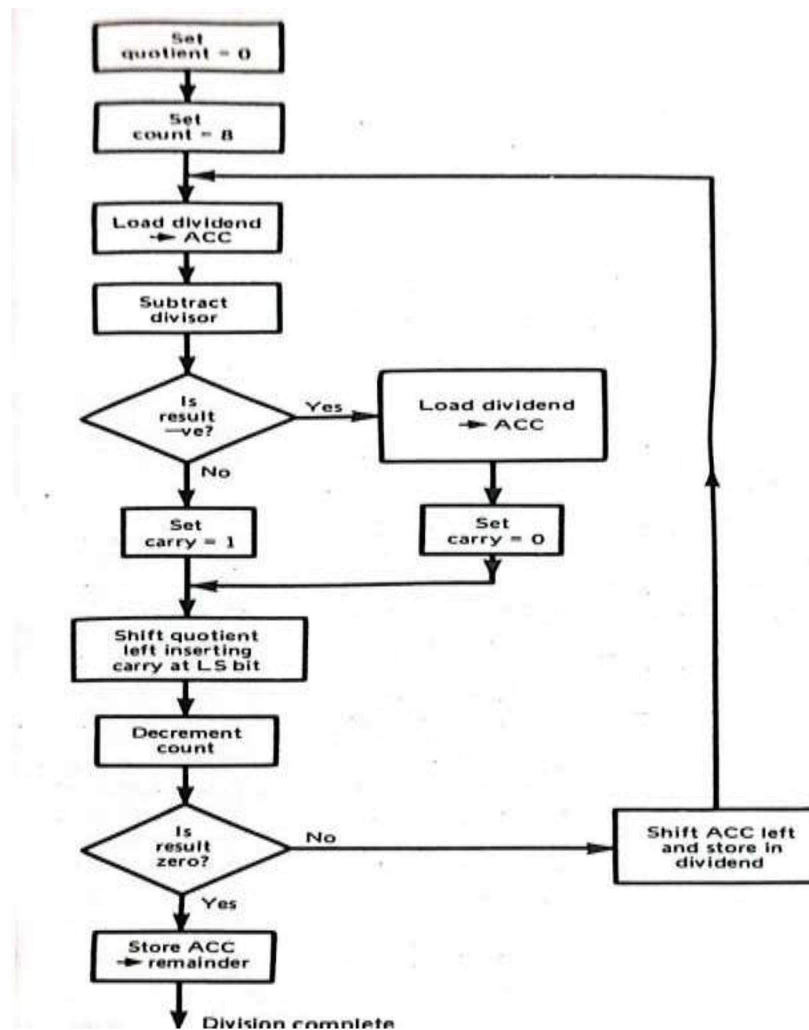

Рис.2.8. Потік програми для простої 8-бітної процедури ділення

Нарешті найбільш значущі частини додаються знову з урахуванням будь-якого переносу від попереднього додавання, і цей результат зберігається як найважливіша частина четвертої частини, у більшості випадків будь-який створений перенос може знадобитися зберегти у відповіді. Віднімання чисел у кількох розділах може бути виконано нашим аналогічним способом між кожною наступною операцією. Подібні прийоми можна використовувати для множення та ділення. Там, де дроби потрібно обробляти та ділити, використовуйте певну форму арифметичної системи з плаваючою комою.

У схемі з плаваючою комою кожне число ділиться на два елементи. Одна частина — це дріб із значенням від 0 до 1. Вона забезпечує основне числове значення та визначає точність результатів. Друга частина числа є показником ступеня. Показник степеня може бути числом, що представляє ступінь 2, на який потрібно помножити дріб щоб отримати фактичне значення числа.

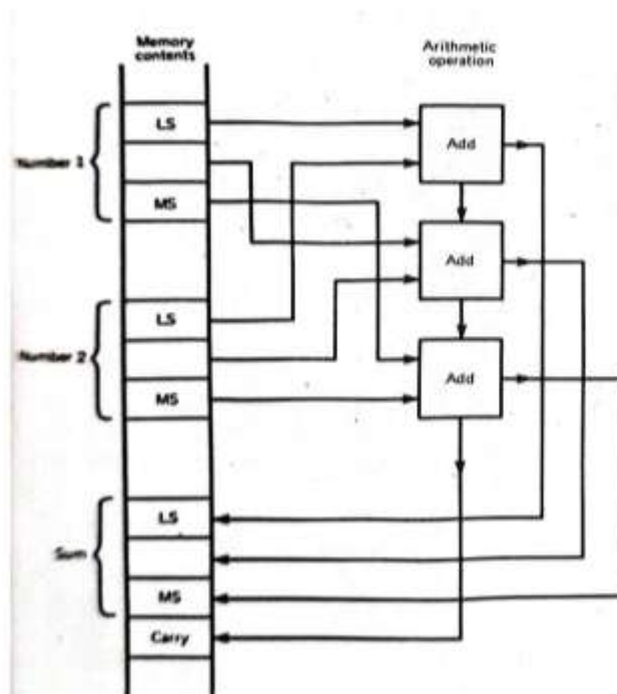

Рис.2.9. Додавання двох 24-розрядних чисел, кожне з яких зберігається як три 8-розрядних слова

У частці біт праворуч від знакового біта представляє значення 0,5, а наступні біти, що рухаються праворуч, представляють 0,25, 0,125, 0,0625 і так далі. Також можна мати дріб, представлений як число BCD, і в цьому випадку послідовні цифри BCD будуть еквівалентними десятим, сотим і так далі.

Припустімо, ми хочемо встановити значення для 32, використовуючи 8-бітний експонент і 8-бітний дріб. Тут частка може бути встановлена як 0,5 (01000000 у двійковій системі), а експонента – як 6 (00000110 у двійковій

системі). Таким чином, ми отримуємо дріб 0,5, помножений на 26, що дає необхідне значення 32. Використання цієї техніки дає майже астрономічний діапазон чисел, і з 8-розрядною машиною ми можемо варіювати від 2126 (величезний) до 2-126 (надзвичайно крихітний) і всі числа залишаться точними з точністю до 8 біт, що становить приблизно ±0,5%. З плаваючою комою можна обробляти великий діапазон чисел, але точність залежатиме від кількості бітів, які використовуються для дробу.

Слід зазначити, що якщо для дробу використовується BCD, то показник степеня буде десятковим, тобто це ступінь 10, а не ступінь 2. Використання десяткового показника степеня та дробу корисно, якщо результати потрібно надрукувати поза.

Коли числа представлені у формі з плаваючою комою, послідовність арифметичних операцій стає складнішою. Припустимо, ми беремо додавання і віднімання. Перш ніж можна буде здійснити фактичне додавання або віднімання, два числа повинні бути скориговані таким чином, щоб їхні показники були однаковими. Це означає, що масштабне значення двох дробів буде однаковим. У цей момент частки дробу додаються або віднімаються за бажанням, а результат зберігається як частка відповіді. Показник відповіді буде таким же, як і в інших чисел.

Щоб зробити показники однаковими, частка меншого числа послідовно ділиться на 2, а його показник збільшується, доки два показники не стануть рівними. На рис. 2.10 показано приклад цієї операції. Дріб просто зсувається праворуч на одну позицію, щоб забезпечити ділення на два.

Для множення показники степеня додаються разом, щоб отримати показник степеня відповіді, тоді як дробові частини множаться разом, щоб отримати дріб для відповіді. Ділення передбачає віднімання показника степеня дільника від показника діленого, щоб отримати показник степеня для частки, тоді як частка дільника ділиться на частку від діленого, щоб утворити частку від частки.

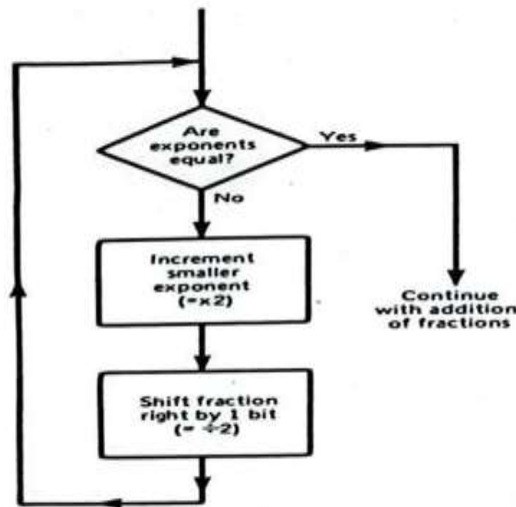

Рис.2.10. Блок-схема циклу для прирівнювання експонент до додавання або віднімання з плаваючою комою

Наприкінці будь-якої операції з плаваючою комою зазвичай виконується процес, відомий як нормалізація. Це фактично перевіряє, чи старший біт результату дорівнює 1, і якщо ні, дроб зміщується ліворуч і коригує експоненту, доки ця умова не буде виконано. Це гарантує, що використовується повна точність дробу.

2.4 Підпрограми та переривання

2.4.1 Підпрограми: визначення та принципи роботи в програмуванні

Більшість комп'ютерних програм мають послідовності інструкцій, які повторюються в різних місцях програми. Прикладом є набір інструкцій для обробки даних, що вводяться з клавіатури кожного разу, коли натискається клавіша. Звичайно, цей набір інструкцій можна вставляти за потреби в програму, але це витрачає простір пам'яті, а більш ефективна техніка використовує підпрограму.

Послідовність інструкцій для підпрограми з клавіатури зберігається в пам'яті як окрема частина програмного коду, яка називається інструкція JSR

(перехід підпрограмою, і кожного разу, коли функція клавіатури має бути виконана, спеціальна до підпрограми) вставляється в основну програму.

Для деяких процесорів замість JSR використовується інструкція CALL. Інструкція JSR або CALL має своїм операндом початкову адресу коду підпрограми. Кожного разу, коли зустрічається інструкція JSR, здійснюється перехід до першої інструкції послідовності підпрограм, і починається виконання підпрограми. Остання інструкція в підпрограмі є інструкцією RTS або RET (повернення з підпрограми).

Ця інструкція RTS спричиняє відновлення основної програми за інструкцією, що йде відразу після JSR або CALL. Кожного разу, коли потрібна функція підпрограми, ця послідовність подій відбувається з виконанням програми, що переходить від основної програми до підпрограми та назад. Це показано на рис. 5.1

Перевага використання підпрограми полягає в тому, що, незважаючи на те, що послідовність інструкцій підпрограми може виконуватися багато разів протягом основної програми, вона зберігається в пам’яті один раз, що економить місце.

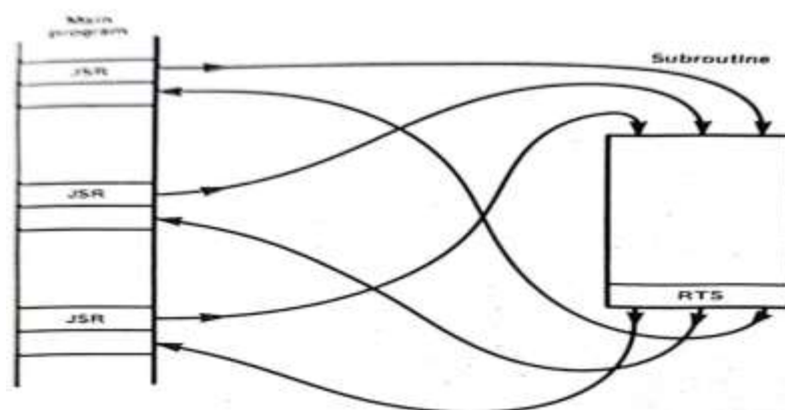

Рис.2.11. Потік програми для підпрограм

Очевидно, щоб повернутися до правильного місця в основній програмі після виконання підпрограми, ЦП повинен запам'ятати, де він був, коли

виникла інструкція JSR або CALL. Це робиться шляхом збереження вмісту регістра програмного лічильника (PC) у спеціальному регістрі SAVE перед фактичним переходом до підпрограми.

Після збереження вмісту ПК завантажується з початковою адресою підпрограми і виконання переходить до підпрограми. Наприкінці підпрограми, коли збережена адреса є лічильником програми, а наступною виконаною інструкцією буде та, що йде після JSR в основній програмі.

У простому мікропроцесорі, де є лише один регістр збереження ПК, виклик іншої підпрограми з підпрограми буде катастрофічним.

Адже початкова адреса повернення буде перезаписана і, отже, втрачена, а при поверненні з першої підпрограми ЦП втратить її місце в основній програмі. Щоб подолати цю проблему, більшість мікропроцесорів використовують стек для збереження адреси повернення під час виконання підпрограми.

2.4.2 Стек: означення та роль у системах зберігання даних

Стек є розширенням регістра збереження програмного лічильника, який використовується в простих мікропроцесорах. У невеликих процесорах стек може складатися просто з трьох або чотирьох регістрів збереження з деякою логікою керування, організованою так, що коли перший регістр заповнюється, наступний включається в гру як регістр збереження для нової підпрограми, і цей процес продовжується вниз по банку регістрів. поки вони не заповняться. Після завершення кожної підпрограми вміст лічильника програми відновлюється з останнього заповненого регістру збереження. Цей процес триває, доки не будуть завершені всі підпрограми. Загальна ідея проілюстрована на рис. 2.12. мікропроцесори загального призначення Closet реалізують функцію стека в пам'яті читання/запису. У цьому випадку спеціальний регістр ЦП, званий регістром покажчика стека, оцінюється як поточна вершина.

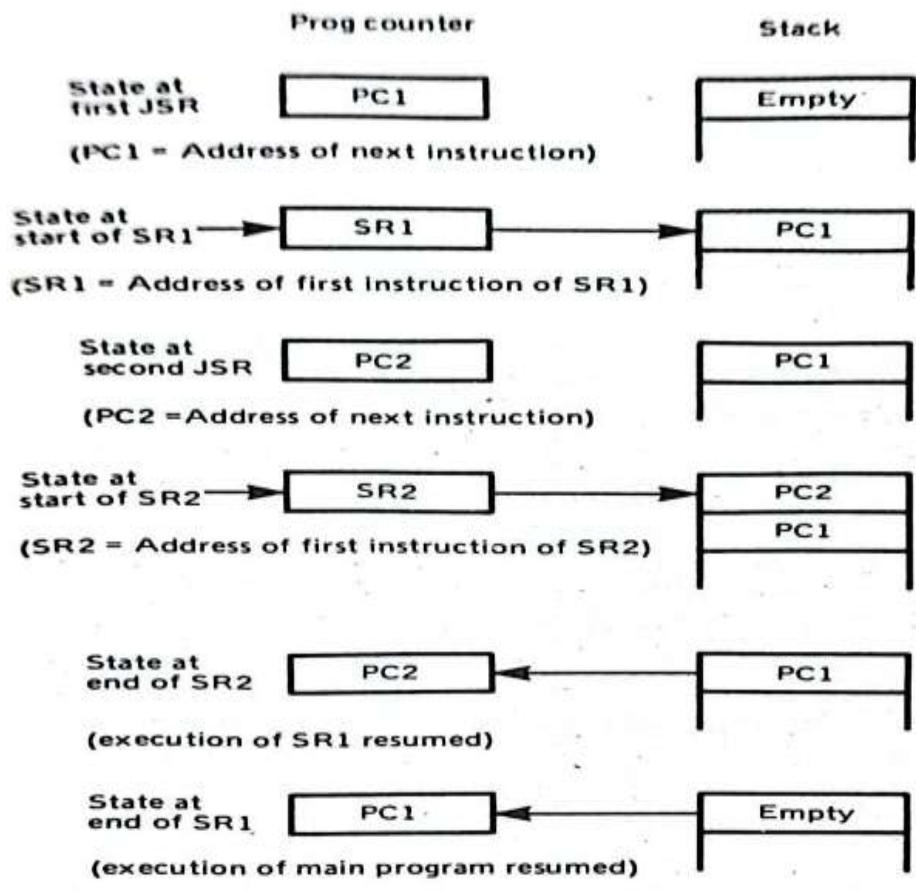

Рис.2.12. Робота стека під час виконання підпрограм

Коли слово даних передається в стек, вказівник стека налаштовується, щоб стек використовував його. Коли слово видаляється зі стеку, SP коригується, щоб дозволити доступ до нижчих рівнів у стеку.

У практичному комп’ютері стек, як правило, будується вниз у пам’яті, так що коли кожен новий елемент записується в стек, вміст покажчика стека і, отже, поточна адреса пам’яті в стеку зменшується на одиницю.

У 8-розрядному процесорі, який використовує 16-розрядні адреси, наприклад 8085 або 6800, стек буде зменшено на два місця під час виклику підпрограми, оскільки два 8-розрядні байти будуть потрібні для зберігання адреси повернення підпрограми.

У мікропроцесорі загального призначення стек може використовуватися для збереження даних, а також вмісту програмного лічильника. Для цього є дві інструкції. Інструкція PUSH змушує одне слово даних передаватись з накопичувача або іншого регістра на вершину стека, а потім стек налаштовується, щоб дати адресу наступного вільного розташування. Щоб відкликати слово даних зі стеку, натисніть PULL. (або для деяких типів POP) можна використовувати інструкцію. Тут покажчик стека спочатку налаштовується на останнє слово, записане в стек, а потім слово даних зчитується зі стеку. Слід зазначити, що читання зі стеку не змінює вміст розташування стека в пам'яті. Іноді виникне потреба передати одне або декілька слів даних між основною програмою та підпрограмою. Це можна зробити, використовуючи виділену область пам'яті, яку спільно використовують підпрограма та основна програма. Альтернативний, а часом і більш зручний метод, передбачає використання стека. У цьому випадку до досягнення інструкції JSR або CALL необхідні слова даних надсилаються в стек. Після входу програми в підпрограму вказівник стека можна налаштувати так, щоб він вказував на слова даних, які потім викликаються зі стеку та використовуються в підпрограмі. Будь-які дані, які потрібно передати назад, також можуть бути передані в стек до завершення підпрограми. Важливим моментом, на який слід звернути увагу під час виконання таких передач, є переконатися, що вказівник стека було змінено правильно, щоб вказувати на адресу повернення до завершення підпрограми, інакше призведе до хаосу. Також важливо переконатися, що адреса повернення для підпрограми не буде перезаписана під час підпрограми.

У багатьох програмах мікропроцесор може виконувати безперервний процес, наприклад, керувати машиною, але він також повинен реагувати на зовнішні події, такі як робота перемикача. Звичайно, центральний процесор може регулярно перевіряти стан комутатора, але більш ефективна техніка передбачає використання так званого переривання.

Спеціальний рядок введення, який зазвичай називається INT або IRQ (запит на переривання), надається на ЦП. За нормальних умов ЦП виконує свою програму, але коли в рядку IRQ відбувається введення, ЦП завершує виконання поточної інструкції, а потім виконує послідовність переривань.

Переривання змушує програму переходити до розділу коду, відомого як підпрограма обслуговування переривань. Багато в чому це схоже на підпрограму, за винятком того, що вона ініціюється сигналом IRQ від зовнішнього обладнання. Інструкція повернення з переривання (RTI) наприкінці процедури переривання змушує програму продовжити роботу в точці, де відбулося початкове переривання.

Проста схема переривання викликає операцію переривання кожного разу, коли апаратний сигнал подається на лінію IRQ. Іноді може бути бажано, щоб дія переривання була придушена під час тих частин програми, де процесор не має вільного часу для обслуговування нових переривань. Це досягається за допомогою маскуваного переривання.

Щоб розрізнити два типи, просте переривання називається немаскованим перериванням (NMI). Насправді тип, що маскується, зазвичай розглядається як звичайний тип, і його вхідні дані часто називають просто IRQ, тоді як типи, що не маскуються, мають вхідні дані, позначені як NMI.

У маскуваному перериванні біт прапора, як правило, у регістрі стану ЦП, використовується для вказівки, чи активне переривання чи ні. Встановлюючи або очищаючи цей біт маски, лінію переривання можна зробити активною або проігнорувати за потреби. Якщо є два або більше входів, які можна маскувати, кожному може бути призначено окремий біт маски. Коли переривання маскується, його присутність помічається встановленням прапора, і коли воно знову стає активним, переривання може бути обслуговано, якщо потрібно.

Деякі порти вводу/виводу також матимуть засоби маскування переривання, так що, встановивши біт у регістрі керування портом, сигнал запиту на переривання від цього порту можна зробити неактивним.

2.4.3 Програмні переривання: сутність та використання в програмуванні

Багато мікропроцесорів мають інструкцію програмного переривання (SWI), яка викликає дію переривання, коли вона зустрічається в програмі. Насправді програмне переривання буде діяти так само, як стрибок підпрограми, за винятком того, що воно ініціює програму обслуговування переривання замість підпрограми.

Це може здатися досить зайвим, але насправді програмне переривання в основному використовується для налагодження програми, де код операції для SWI може бути замінений на звичайний код операції в програмі, щоб дозволити зупинити виконання з метою діагностики.

Різновидом програмного переривання є операція типу пастки. Тут ЦП може виявити недопустимий код інструкції або недопустиму умову, наприклад «поділити на нуль», і ініціювати програмне переривання. В якості альтернативи інструкція 'trap' може бути використана для виявлення конкретного стану регістра стану і, таким чином, для створення переривання.

Процедура обслуговування переривань для цих переривань-переривань буде розроблена для перевірки помилки та внесення відповідних виправлень перед відновленням основної програми.

2.4.4 Обробка одночасних переривань: вплив та заходи безпеки

У більшості систем переривання всі зовнішні пристрої, які можуть викликати переривання, будуть підключені паралельно до однієї лінії введення IRQ ЦП. Як правило, використовується система Wired-OR, коли будь-який окремий пристрій може знизити напругу на лінії переривання до стану 0, але не

пошкоджується, коли інший пристрій підтягує лінію до 0. Якщо на лінії є кілька пристроїв, то час від часу два чи більше пристроїв викликатимуть переривання одночасно. Для роботи з такими кількома перериваннями тепер може знадобитися певна система пріоритетів.

Кожен пристрій переривання зазвичай містить певну форму регістра стану, і зазвичай до нього може отримати доступ ЦП за певною унікальною адресою пам'яті. Деякі процесори мають окрему схему адреси для пристроїв введення/виведення, але, незважаючи на це, регістр стану пристрою все ще може читатися ЦП. Один або більше бітів у регістрі стану можуть використовуватися для вказівки того, що пристрій запитав увагу та, можливо, якої уваги воно вимагає.

Зовнішні пристрої, як правило, працюють на різних швидкостях. Очевидно, що повільніші робочі пристрої, такі як принтери, не потребують негайного вирішення, тоді як перетворювач у контурі керування може потребувати негайної реакції на своє переривання, щоб контур керування працював правильно. Таким чином, кожному пристрою можна надати відносний пріоритет із швидкісними пристроями, які є найвищими у списку.

Коли два переривання відбуваються разом, процесор і підпрограми переривань повинні визначити, який пристрій має вищий пріоритет, і обслуговувати цей пристрій першим.

Найпростіший підхід полягає у використанні процедури опитування, яка перевіряє кожен пристрій по черзі, починаючи з найвищого пріоритету, щоб побачити, чи спричинило воно переривання. Якщо виявлено запит на переривання, пристрій обслуговується. Якщо в кінці цієї процедури обслуговування все ще існує переривання, програма опитування відновлюється, доки не буде оброблено всі запити.

Альтернативна схема полягає у використанні зовнішньої апаратної логіки для визначення пріоритету переривання та представлення необхідних запитів на переривання процесору з правильним пріоритетом. Деякі процесори,

такі як серія Texas 9900, мають логіку пріоритету переривання, вбудовану в ЦП. Як і підпрограми, переривання можуть бути вкладеними, але потрібна певна увага, щоб гарантувати, що нове переривання може бути оброблено без перешкод для переривання, яке зараз обробляється. Таким чином, зовнішньому пристрою, який має дуже швидку роботу, може бути дозволено переривати повільніший пристрій, але зворотне буде неприпустимим.

Зазвичай, якщо нове переривання виникає під час обробки існуючого переривання, нове переривання буде перевірено на пріоритет, і якщо воно має вищий рівень пріоритету, ніж поточне переривання, воно буде оброблено негайно. Якщо його пріоритет нижчий, то він буде утримуватися до завершення поточної процедури переривання. Робота з такими проблемами пріоритету та часу може стати досить складною, особливо якщо програма повинна працювати швидко та ефективно.

У деяких випадках усі інші переривання можуть бути замасковані, як тільки вводиться програма обслуговування переривань; будь-які нові переривання будуть ігноруватися, але можуть бути встановлені біти прапорів у регістрі стану, щоб показати, що вони відбулися. Після завершення поточної операції переривання маска переривання видаляється, і тепер система реагує на будь-які нові переривання, які були позначені.

2.4.5 Вектор переривань: означення та роль в системах обробки переривань

Вектор переривання - це область пам'яті, яка містить адресу початку програми обслуговування переривання. Коли відбувається переривання та приймається процесором, поточна інструкція буде завершена, а потім лічильник програми буде завантажено з розташування вектора переривань, так що наступною інструкцією, яка буде виконана, буде перша інструкція відповідної програми обслуговування переривань.

Метод, за допомогою якого вибирається вектор переривання, відрізняється від одного типу ЦП до іншого. Багато ЦП, такі як серії 6800, 6500 і Z80, автоматично отримують початкову адресу переривання з фіксованого вектора, який зазвичай розташований у верхній частині пам'яті, або, у випадку Z80, у нижній частині пам'яті.

Деякі процесори, такі як 8080, коли вони приймають вихід переривання, видають сигнал підтвердження переривання і у відповідь на це очікують отримання від пристрою, що перериває, через шину даних адресу вектора переривання, який потім завантажується в програмний лічильник.

2.4.6 Вплив переривань на поточні дані програми: аналіз та можливі впливи

Переривання може виникнути в будь-який момент під час виконання програми. Зазвичай, коли переривання виявлено, ЦП завершить виконання своєї поточної інструкції, а потім перейде до підпрограми обробки переривань. Оскільки акумулятор та інші регістри можуть використовуватися для обробки переривання, важливо, щоб поточний вміст цих регістрів було збережено, щоб його можна було відновити, коли головна програма відновить роботу після обробки переривання.

Деякі процесори, такі як серія Motorola 6800, автоматично зберігають вміст усіх регістрів процесора, надаючи дані в стек. В інших процесорах можуть бути збережені лише програмний лічильник і регістри стану, а вміст інших регістрів повинен бути збережений програмістом відповідними інструкціями на початку процедури переривання. Подібним чином наприкінці процедури переривання інструкція RTI призведе до автоматичного відновлення вмісту регістра за допомогою процесорів Motorola, тоді як для інших типів ця операція повинна виконуватися за допомогою інструкцій у програмі переривання.

Висновки до розділу

У другому розділі було розглянуто ключові аспекти послідовної передачі даних, включаючи роль регістрів зсуву та синхронізації для забезпечення стабільності передачі. Також висвітлені питання формату слова, інтерфейсних шин та важливості вибору оптимальної швидкості передачі даних.

РОЗДІЛ 3
РОЗРОБКА АЛГОРИТМІЧНОГО ЗАБЕЗПЕЧЕННЯ НА МІКРОПРОЦЕСОРНІЙ ОСНОВІ

3.1 Опорні пристрої

3.1.1 Опорні пристрої: визначення та застосування

Крім самого центрального процесора, мікрокомп'ютерна система зазвичай включає інші пристрої, які називаються мікросхемами підтримки. Деякі з них, такі як пам'ять і пристрої введення/виведення, вже обговорювалися. Інші допоміжні пристрої включають таймери, лічильники, арифметичні пристрої, аналогові схеми вводу/виводу, контролери дисплеїв та інші контролери периферійних пристроїв, наприклад для дискової пам’яті.

Багато функцій, що забезпечуються допоміжними чіпами, фактично можуть оброблятися самим ЦП, але залишатимуть мало можливостей ЦП для обчислювальних завдань. Завдяки перенесенню завдання підтримки на інший чіп ЦП може працювати ефективніше. Насправді деякі допоміжні чіпи самі по собі є однокристальними мікрокомп’ютерами.

Деякі допоміжні чіпи розроблені спеціально для роботи з одним типом або серіями мікропроцесорів, тоді як інші призначені як типи загального призначення. У більшості випадків допоміжні пристрої для використання з конкретним мікропроцесором можна використовувати з іншими типами процесорів шляхом додавання кількох дискретних компонентів або логічних мікросхем.

Хоча мікропроцесор може досить легко виконувати завдання синхронізації та підрахунку, використовувати ЦП для цієї мети неефективно, і краще, якщо ці функції виконуються зовнішнім пристроєм, залишаючи ЦП вільним для виконання інших обчислювальних завдань до періоду затримки або підрахунку, операція завершена. Організація типової мікросхеми лічильника

таймера показана на рис. 3.1. Є три внутрішні регістри, які можуть використовуватися мікропроцесором. Для налаштування режиму роботи мікросхеми використовується керуючий регістр. Це реєстр лише для запису. З регістром керування пов’язаний регістр стану, який доступний лише для читання та часто має ту саму адресу, що й регістр керування. Регістр стану надає біти прапорів, які показують, коли лічильник проходить через нуль.

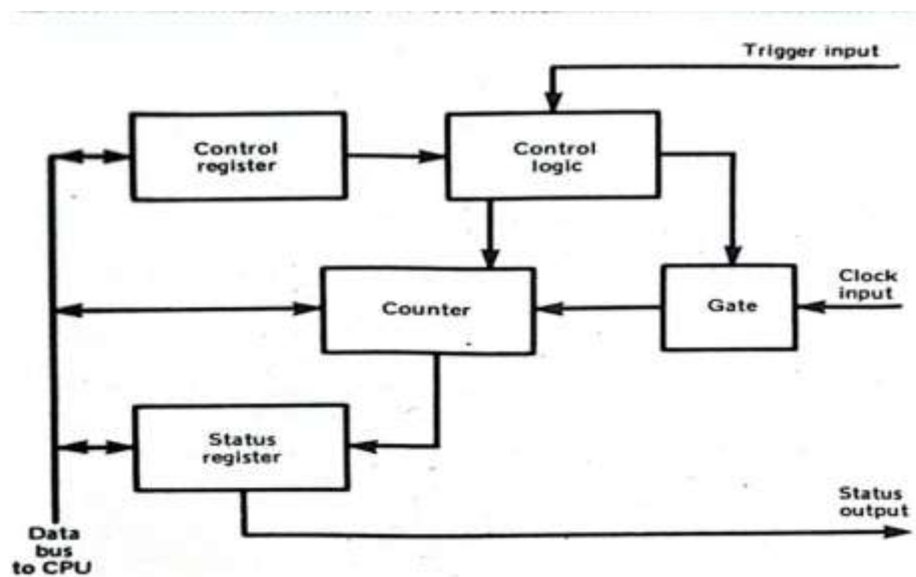

Рис.3.1. Базова організація типового пристрою лічильника таймерного типу

Регістр лічильника — це звичайний двійковий лічильник, який зазвичай веде зворотний відлік. ЦП може записувати дані в цей регістр або читати з нього. Деякі типи також можуть мати четвертий регістр, до якого можна передати поточне значення підрахунку.

Крім регістра підрахунку та регістра керування/статусу, пристрій зазвичай містить деяку логіку стробування на вході регістра підрахунку для забезпечення різних режимів роботи, таких як порівняння частоти або вимірювання ширини імпульсу. Існує також вихід лічильника, який зазвичай є однією лінією від схеми тригера.

Найпростішим режимом роботи є одиночний режим хронометражу. Тут ЦП записує число в регістр лічильника, який потім починає зменшуватися з кожним синхронізуючим імпульсом. Рядок виходу лічильника стає високим і залишається високим, доки регістр лічильника не досягне нуля, коли вихід стає низьким і рахунок зупиняється. Варіант полягає в тому, що регістр завантажується, але підрахунок не починається, поки імпульс не буде подано на вхід затвора.

Більшість таймерів лічильників мають безперервний режим відліку часу, коли лічильник перезавантажується зі своїм початковим значенням після досягнення нуля та починає зворотний відлік. У цьому випадку вихідний рядок просто змінює стан кожного разу, коли кількість досягає нуля, даючи на виході квадратний сигнал.

Варіант безперервного режиму синхронізації створює короткий вихідний імпульс щоразу, коли відлік досягає нуля. Зазвичай ширина імпульсу дорівнює одному циклу годинника, який керує лічильником. Цей режим корисний для створення коротких стробів або імпульсів вибірки через регулярні проміжки часу.

Періоди часу можна виміряти, запустивши лічильник одним імпульсом і зупинивши його наступним, поданим на вхід затвора. Різниця між значеннями лічильника — це кількість тактів, що минули.

Для вимірювання частоти потрібні два лічильники часу. Один використовується для забезпечення фіксованого періоду таймера та контролює логіку затвора другого лічильника, який фактично підраховує цикли вхідного сигналу протягом періоду часу.

Типовим лічильником таймера є Motorola 6840, який містить три незалежних 16-розрядних лічильника таймера. Інші типи — це Intel 8253 і Zilog Z80-CTC, які забезпечують подібні можливості.

Блок відеодисплея (VDU), який використовується з мікрокомп’ютерною системою, має дисплей телевізійного типу та використовує схеми, подібні до

тих, що в домашньому телевізійному приймачі, для створення зображення на екрані. Дисплей вимальовується на екрані як ряд горизонтальних ліній, яскравість яких змінюється в міру того, як вони проходять по екрану. У США, Канаді та Японії дисплей мають 525 ліній сканування, тоді як у Великобританії, Європі та більшості інших частин світу використовується 625 ліній сканування. Щоб зменшити мерехтіння, весь дисплей виводиться 30 разів на секунду в системі США та 25 разів на секунду в інших країнах.

Коли потрібно відобразити текст, область екрана ділиться на масив маленьких прямокутників із зазвичай 24 рядками, кожен з яких містить 40 прямокутників. Згодом кожен прямокутник міститиме один текстовий символ. Ці окремі символи створюються за допомогою матриці точок. Вибірково підсвічуючи точки в прямокутнику дисплея, можна отримати форму потрібного символу, як показано на рис. 3.2.

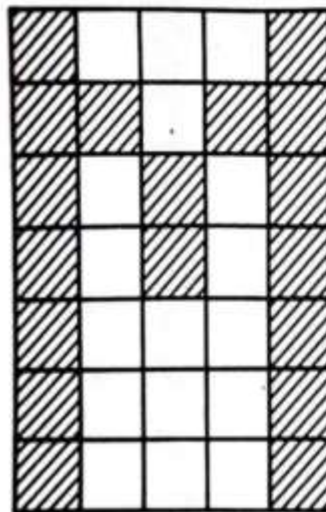

Рис.3.2. Текстові символи, що відтворюються на екрані телевізора, використовують вибірково освітлені точки в матриці (затінені області підсвічуються для відображення літери M)

Багато дисплеїв використовують формат із шести крапок у ширину та вісім точок у висоту для кожного символу, причому фактичний символ займає матрицю 5 х 7, один рядок і один стовпець залишаються порожніми, щоб відокремити символ від сусідніх у тому самому рядку та від інших у рядки зверху і знизу. Однак за допомогою матриці 5 х 7 важко правильно відобразити малі літери, якщо вони мають низхідні символи, які простягаються нижче нижньої частини інших символів у рядку. Рішенням є використання більшої точкової матриці, як правило, 5 × 9 або, можливо, 7 × 9, де звичайні символи все ще мають формат 5 × 7, але нижні регістри відображаються у правильному місці під лінією символів. Цей формат також забезпечує більше розділення між рядками тексту та покращує читабельність.

3.1.2 Символьний генератор: принцип роботи та застосування

Для відображення тексту система відображення повинна мати можливість викликати двійковий код. Фактичні шаблони крапок можуть бути легко збережені як шаблони 1s та Os у пам’яті лише для читання, а використовуючи код символу як адресу, досить легко викликати будь-який бажаний шаблон крапок за потреби. Таке ПЗУ називається відповідний шаблон точок для кожного текстового символу, який відображається на екрані. Зазвичай кожному символу в наборі, який можна відобразити, призначають 7-бітний ПЗУ символьного генератора.

Кожному шаблону точок символу виділяється ділянка пам’яті шириною, наприклад, вісім біт і, можливо, з 10 послідовними словами для кожного символу з одним словом для кожного горизонтального ряду крапок. Один повний ряд точок виводиться паралельно за один раз, і окремі ряди точок у кожному шаблоні символу можуть бути обрані за допомогою 5-бітної адреси, яка буде частиною звичайного введення адрес для ПЗУ.

Щоб вибрати фактичний шаблон для символу, який буде відображатися, 7-бітна адреса, що дозволяє до 128 різних текстових символів, застосовується

до ПЗУ. Ця адреса вибирає певну групу послідовних слів у ПЗП, які містять шаблон для потрібного символу. Коли простежуються послідовні рядки сканування, подальша адреса вибирає правильний рядок точкового шаблону для відображення.

3.1.3 Принципи функціонування типової системи відображення відео

Логіка, задіяна у створенні відеодисплею з текстом або графікою, досить складна, але відповідає загальній структурі, показаній на рис. 7.4.

В основі системи відображення лежить генератор синхронізації, який контролює синхронізацію всієї логіки та сигналів, задіяних у створенні дисплея. Початковою точкою часто є «точковий годинник», який визначає час для окремих точок, що відображаються на екрані. Це може мати типову частоту близько 6 МГц. Точковий годинник можна розділити за частотою на шість, щоб отримати символьний годинник, оскільки символи складаються з груп із шести послідовних точок у рядках сканування.

Подальший поділ частоти та стробування використовується для створення імпульсу синхронізації на початку кожного рядка сканування, а також для гасіння лівого та правого полів навколо екрана. Кожен текстовий рядок складається, можливо, з десяти послідовних рядків сканування, тому буде годинник рядка, який відстежує положення внизу екрана.

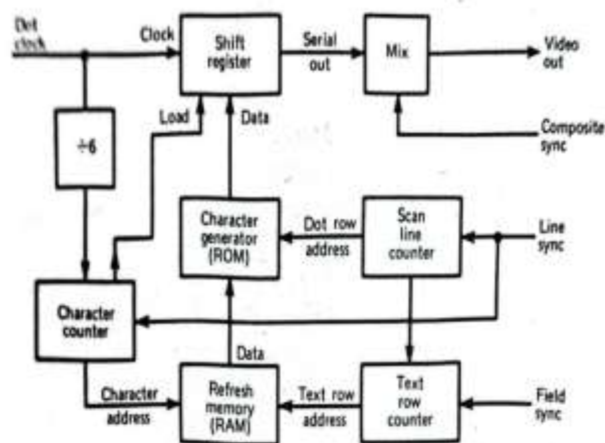

Рис.3.3. Структурна схема типової логічної системи відображення відео

Оскільки для усунення мерехтіння телевізійний екран потрібно сканувати приблизно 25 разів на секунду, дані для всього тексту на екрані мають бути постійно доступними. У деяких системах комп'ютер сам керує створенням відображення, але для більшості систем дані розміщуються в пам'яті оновлення екрана та викликаються відповідно до вимог логіки керування та синхронізації.

Оскільки кожен рядок символів сканується по екрану, лічильник використовується для відстеження позиції символів у рядку, а також надає частину адреси екранної пам'яті. Цей лічильник символів повторюється для кожного рядка сканування. Кожен рядок символів зазвичай займає 10 послідовних рядків сканування. Коли має бути відображено наступний рядок тексту, новий набір даних має бути прочитаний з пам'яті, тому лічильник адреси рядка використовується для відстеження того, який текстовий рядок відображається. Під час сканування кожного рядка тексту виконується підрахунок рядків, і це вибере правильний ряд точок із матриці в ПЗП генератора символів.

Затримка в кілька рядків використовується у верхній частині дисплея, щоб створити верхнє порожнє поле, і аналогічне гасіння використовується для забезпечення поля в нижній частині екрана. У кінці кожного вертикального сканування вниз по екрану вставляється імпульс синхронізації поля, щоб дозволити системі телевізійного сканування йти в ногу з відеосигналами. На практичному дисплеї є 50 полів на секунду, і вони чергуються під час послідовних сканувань. Таким чином, на першому скануванні непарні рядки сканування виводяться, тоді як на другому скануванні парні рядки виводяться між рядками першого сканування. Таким чином, у системі з 625 рядків буде 312,5 рядків у кожному полі сканування. З 312,5 рядків 240 або 256 використовуються для відображення, а решта гаснуть.

Для створення відеосигналу ряд точок для кожного текстового символу в рядку паралельно завантажується в регістр зсуву, а потім зсувається у вигляді

серії імпульсів увімкнення-вимкнення. Потім імпульси синхронізації додаються для завершення відеосигналу. У системі кольорів кольори складаються з комбінацій червоного, зеленого та синього. Тут використовуються три пам’яті даних, по одній для кожного кольору, і створюються три окремі точкові відеосигнали. Якщо використовується кольоровий монітор, ці сигнали червоного, зеленого та синього кольорів (RGB) керують безпосередньо трубкою дисплея. Якщо використовується телевізійний приймач, кольорові сигнали кодуються так само, як і для телевізійного мовлення, а потім будуть правильно декодовані телевізійним приймачем. Дві більш поширені схеми кодування: NTSC для США та Японії та система PAL для Великобританії та Європи.

3.1.4 Постійне збереження програм та даних: ефективні методи та практичні рекомендації

Програма в мікрокомп’ютерній системі загального призначення або системі розробки зазвичай зберігається в пам’яті для читання/запису та буде втрачена, якщо вимкнеться живлення або якщо в пам’ять буде завантажено іншу програму. Очевидно, що програму не бажано вводити вручну з клавіатури кожного разу, коли її потрібно використовувати, тому потрібна якась система постійної пам’яті.

У більш простих системах і недорогих персональних комп'ютерах ця функція забезпечується за допомогою касетного магнітофона. Сигнали даних перетворюються насамперед у послідовний потік даних зазвичай використовує асинхронний формат. Оскільки аудіокасета не дуже добре записує рівні даних, використовується система тональної модуляції. Тут використовуються два звукові сигнали, один для рівня 1, а другий для рівня О. Це показано на рис. 3.4.

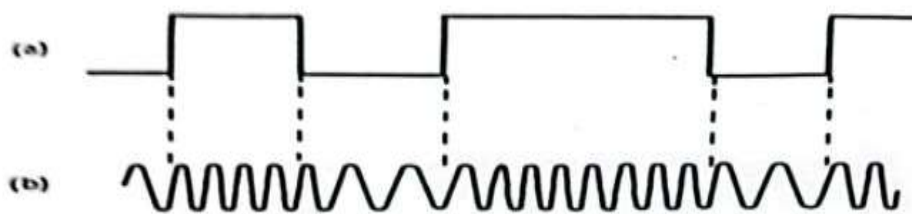

Рис.3.4. (a) Цифровий сигнал даних. (b) Модульований тональний сигнал

Під час повторного відтворення тони перетворюються назад у послідовні цифрові логічні сигнали, а потім поєднуються з центральним процесором через послідовний порт. Зазвичай ці системи працюють зі швидкістю 300 бод, що забезпечує відносно повільну передачу даних.

Щоб пришвидшити передачу даних, деякі системи використовують синхронний формат даних із фазовим кодуванням сигналів зі швидкістю, можливо, 1200 або 2400 бод. Ці сигнали мають достатньо високу частоту для запису як аудіосигнали, і хоча вони дещо спотворені системою запису та відтворення, зазвичай можуть бути декодовані без помилок за допомогою відповідних схем обробки сигналу в каналі відтворення.

Можна передбачити надання імен файлам на касеті та пошук певного файлу на касеті. У деяких касетних операційних системах файли даних, а також програми можуть зберігатися на стрічці, і може бути передбачено автоматичне керування двигуном рекордера для запуску та зупинки стрічки.

Якщо на касету записано кілька програм або файлів даних, пошук потрібного файлу може бути повільним процесом, і більш ефективною системою зберігання в таких випадках є дискета. У цьому випадку запам'ятовуючий пристрій – це тонкий гнучкий пластиковий диск, покритий магнітним оксидом, подібним до того, що використовується на аудіокасетах. Гнучкий диск міститься в тонкому пластиковому конверті. Коли диск вставляється в дисковод, шпиндель приводу зчеплюється з отвором у центрі диска, і диск обертається зі швидкістю приблизно 500 об/хв у межах своєї оболонки.

Сигнали даних можуть бути записані на диск або зчитані з диска за допомогою магнітної головки, подібної до тієї, що використовується в аудіокасетному рекордері. Ця головка контактує з диском через проріз у захисній оболонці та утримується в контакті за допомогою натискної подушечки на протилежному боці диска. Інформація записується змінним магнітним полем так само, як і для касети, і прокладається вздовж кругової доріжки на диску. Радіально переміщаючи головку поверхнею диска, на диску можна створити серію цих доріжок. По суті, диск чимось схожий на аудіодиск, за винятком того, що замість спіралі доріжки на дискеті є концентричними колами.

Існує три основних розміри дискет. Стандартна версія має діаметр диска 216 мм (8,5 дюйма) і іноді називається дискетою. Для більшості персональних комп'ютерів і комп'ютерів малого бізнесу можна використовувати меншу версію діаметром 133 мм (5,25 дюйма). Це називається міні-дискетою або міні-дискетою, і, нарешті, є версія розміром 82 мм (3,25 дюйма), відома як мікро-дискета.

Зазвичай на стандартній дискеті буде 70 або 80 доріжок, а на міні-дискеті — від 35 до 40 доріжок. Деякі диски мають магнітне покриття з обох сторін і називаються двосторонніми дисками. Двосторонній дисковод має дві головки, по одній з кожного боку диска, і диск затиснутий між ними, а неактивна головка діє як притискна подушка.

Стандартна дискета може використовуватися для зберігання від 100 Кб до 500 Кб даних, тоді як міні-дискета зазвичай має ємність від 100 Кб до 200 Кб. Мікродискети випускаються з ємністю 100 КБ. На відміну від звичного аудіодиска, де інформація зберігається на безперервній спіралі, дані на дискеті зберігаються на серії концентричних кругових доріжок. Головка для читання/запису встановлена на кронштейні, що приводиться в рух кроковим двигуном, і її можна переміщати прямо з доріжки на доріжку по поверхні диска.

Кожна доріжка на диску поділена на ряд секторів, кожен з яких зазвичай містить 256 байт даних. На міні-дискеті зазвичай є 16 секторів на трек, що дає близько 4000 байт даних на трек. Ці диски мають 35 доріжок або в деяких випадках 40 доріжок, що дає приблизно від 120 до 150 Кбайт даних на дискету. Стандартні дискети розміром 203 мм (8 дюймів) зазвичай мають подібне розташування доріжок, але мають 70 або 80 доріжок, що забезпечує від 250 000 до 300 000 байт пам'яті.

Більшість дискових систем використовують лише одну сторону диска для зберігання даних, хоча насправді обидві сторони покриті магнітним матеріалом. Деякі дисководи спеціально виготовлені з головкою для запису/відтворення на кожній стороні диска. Лише одна головка буде активною одночасно, тоді як інша діє як притискна подушка, яка утримує поверхню диска на активній головці. Цей тип системи називається «двостороннім» зі зрозумілих причин і забезпечує вдвічі більший обсяг пам'яті, ніж стандартна дискова система. Доступні спеціальні диски для двосторонньої роботи, які спеціально перевіряються, щоб гарантувати відсутність помилок у даних на будь-якій поверхні диска.

Інший спосіб збільшення ємності пам'яті – використання запису подвійної щільності. Тут кожен сектор містить 512 байт даних замість звичайних 256, щоб забезпечити ємність близько 250 Кбайт для 133 мм (5,25 дюйма) диска та 500 Кбайт для 203 мм (8 дюймового) диска. І знову спеціально перевірені диски використовуються для забезпечення мінімальних помилок при вищій щільності запису. Використовуючи подвійну щільність і обидві сторони, ємність одного 203-міліметрового дисковода можна збільшити приблизно до одного мегабайта.

3.1.5 Диск та касетна система: порівняльний аналіз засобів зберігання

Основними відмінностями між касетними та дисковими накопичувачами є швидкість передачі даних і доступність окремих файлів даних. У касетній

системі швидкість передачі даних зазвичай знаходиться в діапазоні від 300 до 1200 бод, що значною мірою залежить від смуги пропускання звукового сигналу систем запису та відтворення. З іншого боку, дисковий блок працюватиме зі швидкістю передачі, яка зазвичай становить від 10 000 до 40 000 бод, тому завантаження файлу з диска на комп'ютер зазвичай відбувається приблизно в 100 разів швидше.

У касетній системі файли записуються послідовно вздовж стрічки, так що для досягнення наступного файлу необхідно прочитати всі попередні файли. Якщо диктофон має лічильник стрічки, тоді можна пришвидшити доступ до файлу, використовуючи швидку перемотку вперед, щоб перемістити стрічку до точки біля початку потрібного файлу за допомогою зчитування лічильника стрічки. Дискова система забезпечує прямий доступ до будь-якої доріжки, а потім доступ до файлу на стійці відбувається протягом одного оберту диска. Для типових мініфлоппі-систем доступ до файлу займає, можливо, десяту частку секунди, а стандартні дискетні системи працюють швидше.

Для зберігання файлів, і особливо для файлів із довільним доступом, дискові системи набагато ефективніші, ніж системи на основі касет. У дисковій системі одна або дві доріжки зазвичай виділяються в каталог, який містить індекс усіх імен файлів і їх розташування на диску. Таким чином, коли запитується файл, дискова система перевіряє каталог, а потім переходить безпосередньо до запитаного файлу. Якщо програми або файли завантажуються лише час від часу, касетна система може бути задовільною. Основна перевага касетної системи полягає в тому, що вона відносно проста і набагато дешевша у застосуванні, ніж система на основі дисків.

Хоча в простіших мікрокомп'ютерних системах є сам ЦП часто використовується для керування диском або касетою, багато систем використовують спеціальну мікросхему контролера для виконання цього завдання. Це дозволяє центральному процесору виконувати інші завдання обробки, поки дані знаходять і передають між диском або касетою та пам'яттю.

На рис. 3.5 показано загальне розташування контролера диска, а для касети схоже, але менш складне.

Як диск, так і касета зберігають свої дані в блоках, можливо, з 256 байтами даних у кожному блоці. Тому частиною логіки контролера є, в основному, послідовний інтерфейс даних, призначений для передачі одного блоку даних між касетою або диском і пам’яттю.

У дисковій системі регістри в мікросхемі контролера зберігають номери доріжок і секторів для поточної позиції диска, а також номери даних, до яких потрібно отримати доступ і передати їх. Потрібну доріжку можна знайти, порівнюючи її з поточним положенням треку голови, а потім крокуючи головою вперед або назад, щоб досягти потрібної доріжки. Коли диск обертає перші кілька байтів даних у кожному секторі читаються. Вони ідентифікують номери секторів, і коли потрібний сектор знайдено, дані або зчитуються з диска, або записуються на нього. У більшості систем новий диск необхідно ініціалізувати або відформатувати перед використанням. Цей процес записує ідентифікаційні дані доріжки та сектора на диск, але залишає частину даних кожного сектора порожньою або з набором даних для всіх 0с.

Касетні системи простіше. На початку кожного файлу, після сигналів синхронізації, зазвичай буде блок заголовка, який містить назву файлу та код, який вказує, який це файл. Цей блок також містить дані, які вказують кількість блоків даних у файлі, щоб контролер знав, коли припинити читання або запис даних.

Звук можна отримати, просто видавши серію імпульсів із комп’ютера та подаючи їх через відповідний підсилювач на невеликий гучномовець. Більшість мікрокомп'ютерних систем, призначених для використання як персональних комп'ютерів, мають цю можливість. На найпростішому рівні мікропроцесор може бути налаштований на роботу в безперервному циклі, в якому він вмикає та вимикає один рядок порту введення/виведення через регулярні проміжки часу.

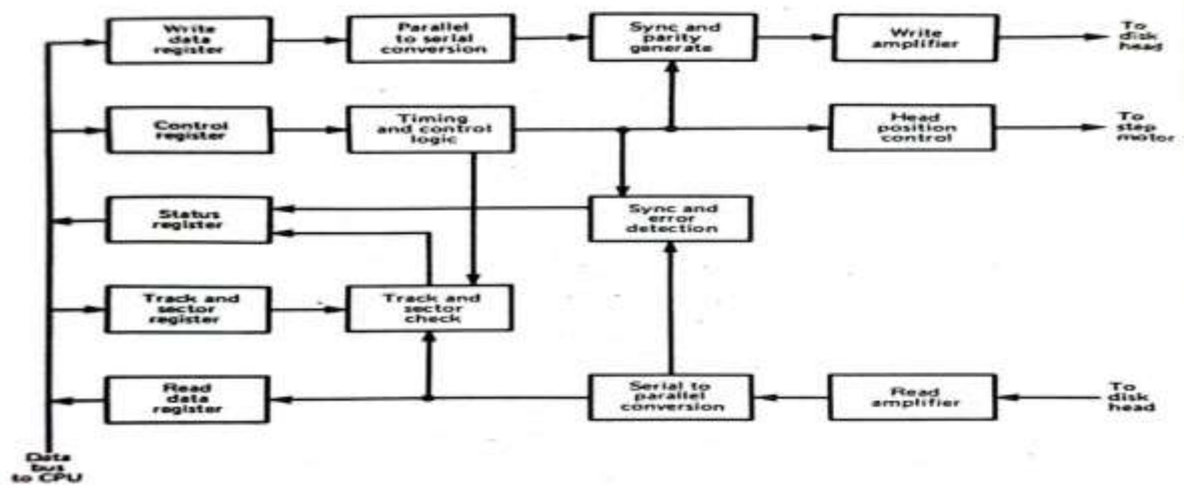

Рис.3.5. Основні організації системи керування дискетою

Цей вихідний сигнал може бути поданий на гучномовець для створення звукового сигналу. Частота тонального сигналу змінюється шляхом зміни часової затримки циклу підрахунку між точками перемикання. Другий цикл підрахунку в програмі можна використовувати для контролю тривалості звуку.

Прості схеми створення звуку з використанням циклів підрахунку можна використовувати для звукових ефектів під час програми, щоб, можливо, привернути увагу користувача до певної дії, яка може знадобитися. Функція «дзвінок» на більшості персональних комп’ютерів використовує цю техніку.

Як і у випадку підрахунку та синхронізації, генерація звуку зв’язує ЦП, тому краще використовувати окремий чіп для генерування звуків і просто керувати ними з ЦП. Типові мікросхеми звукових генераторів забезпечать один або більше генераторів тонів і, ймовірно, генератор «білого шуму». Зазвичай вихідні сигнали можна запрограмувати як по частоті, так і по амплітуді, а потім об’єднати для створення майже нескінченної різноманітності звукових ефектів. Деякі пристрої також включають формування огинаючої для контролю наростання та спаду амплітуди сигналу (також відомого як «атака» та «згасання») і тривалості звуків.

Найкращими генераторами звуку є пристрої виведення голосу. У багатьох із них вибірка вимовлених слів була записана як цифрові шаблони

даних у ПЗУ, і вони відтворювалися через спеціальні схеми генерування звуку для відтворення мови. Слова чи, можливо, фрази можна викликати з пам'яті під керуванням програми, щоб забезпечити голосовий вихід. Щоб зменшити необхідний обсяг пам'яті, використовуються спеціальні методи кодування, серед яких одним із найпопулярніших є лінійне прогнозоване кодування, яке використовується в голосових чіпах Texas Instrument. Типові пристрої голосового виведення можуть мати словниковий запас до 256 окремих слів або фраз. Альтернативна схема, яка створює мову досить нижчої якості — це метод «фонеми», де замість слів ПЗП може містити дані для фонетичних звуків, які називаються «фонемами». Потрібні слова складаються шляхом поєднання цих фонем. Перевагою цього методу є необмежений словниковий запас.

У реальному світі майже всі речі, які ми, ймовірно, хочемо виміряти, знаходяться в аналоговій формі. Зазвичай сигнал, доступний для входу мікрокомп'ютера, - це напруга або струм, який пропорційний деяким фізичним властивостям, таким як довжина, вага, температура або тиск. Щоб використовувати такий вхідний сигнал, його потрібно спочатку перетворити в цифрову форму, сумісну з даними, які використовує комп'ютер. Це робиться за допомогою аналого-цифрового перетворювача (АЦП). Дані, виведені комп'ютером, може знадобитися перетворити в аналогову форму для використання в управлінні зовнішнім обладнанням, і це досягається за допомогою цифрово-аналогового перетворювача (DAC).

Розглянемо цифрово-аналогове перетворення. Діапазон вихідного аналогового сигналу розділений на серію рівних кроків, що відповідають діапазону значень, які можуть бути прийняті цифровим словом. Припустимо, що у нас є 8-розрядне слово, яке має 256 можливих значень, тоді діапазон аналогового виведення поділено на 256 кроків. Тепер ми могли б призначити аналоговий рівень, скажімо, 10 мВ для кожного кроку. Кожен біт у цифровому слові створюватиме «зважений» внесок у кінцевий аналоговий вихід. Таким чином, молодший біт (біт 0) створює вихідний сигнал 10 мВ, біт 1 дає 20 мВ,

біт 2 дає 40 мВ і так далі. Коли біт дорівнює 0, він не створює аналогового виходу, але коли він дорівнює 1, він додає свій «зважений» сигнал до виходу.

Базова схема цифрово-аналогового перетворювача показана на рис. 3.6. Тут виробляється серія двійкових зважених напруг, і ці напруги передаються через серію перемикачів до мережі додавання, а потім через підсилювач для отримання вихідного сигналу. Перемикачі керуються бітами даних слова, виведеного з ЦП. Зазвичай це слово зберігається в регістрі фіксатора на D-A .

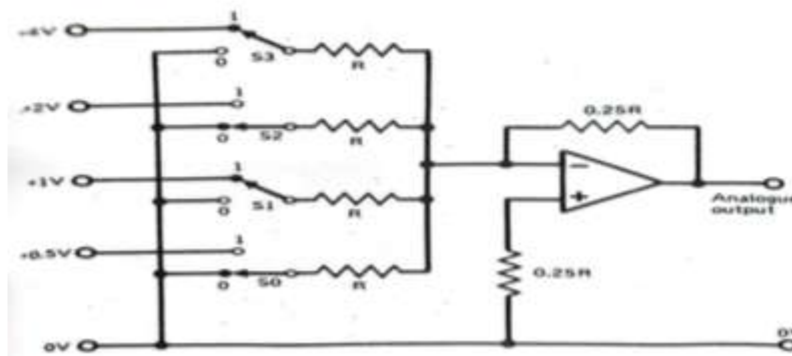

Рис.3.6. Простий цифрово – аналоговий перетворювач з використанням комутованої опорної напруги

Якщо біт дорівнює 0, відповідний перемикач розімкнутий і сигнал на вихід не надходить. Коли біт дорівнює 1, відповідний перемикач замикається, а зважена напруга додається до виходу. Таким чином вихідна аналогова напруга буде пропорційною числовому значенню слова даних.

У практичному пристрої D-A система, подібна до показаної на рис.3.7 використовується. Тут зважені сигнали виробляються резистивом

сходова мережа, де кожна секція розділяє вхідне посилання напруги в два рази. Цей прийом використовується тому, що він є набагато легше отримати точні співвідношення між значеннями резисторів інтегральної схеми, ніж для отримання точних значень резисторів. Для перетворення з аналогового на цифровий використовується більш складна система. Один із підходів показаний на рис. 3.8. Тут вихідний сигнал лічильника подається на цифрово-аналоговий перетворювач, так що в міру збільшення лічильника аналоговий

вихідний сигнал цифрово-аналогового перетворювача збільшує симпатію. Цей аналоговий вихід порівнюється з вхідним сигналом, що підлягає вимірюванню, і коли вони рівні, лічильник зупиняється і число міститься в регістрі лічильника буде пропорційне вхідному сигналу. Потім ЦП зчитує вміст лічильника.

Недоліком методу лічильника є те, що він відносно повільний, і більш популярна система використовує техніку, яка називається послідовним наближенням. Це подібно до підходу, який використовується під час зважування речей на зважувальній машині. Загальна схема показана на рис.3.9.

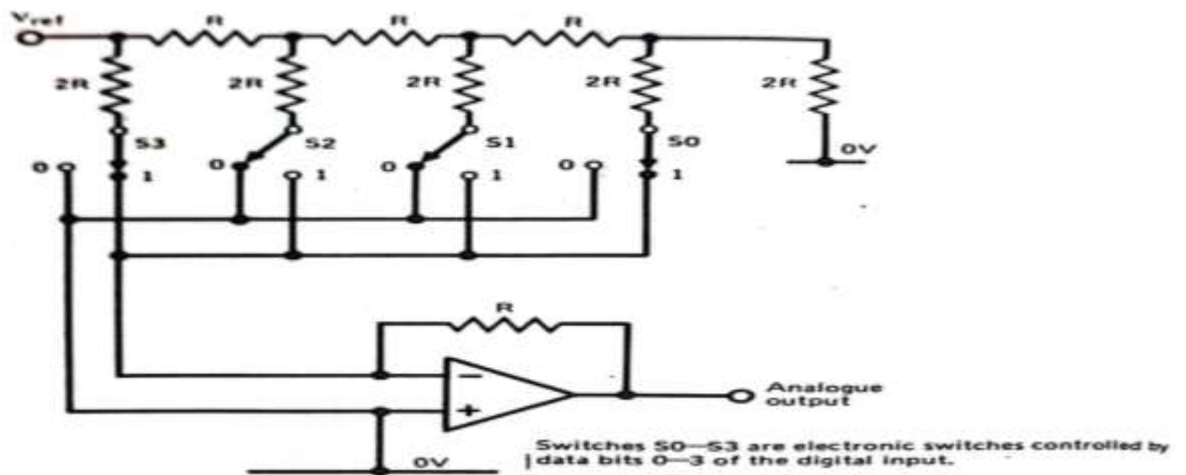

Рис.3.7. Типове розташування практичного цифро-аналогового перетворювача

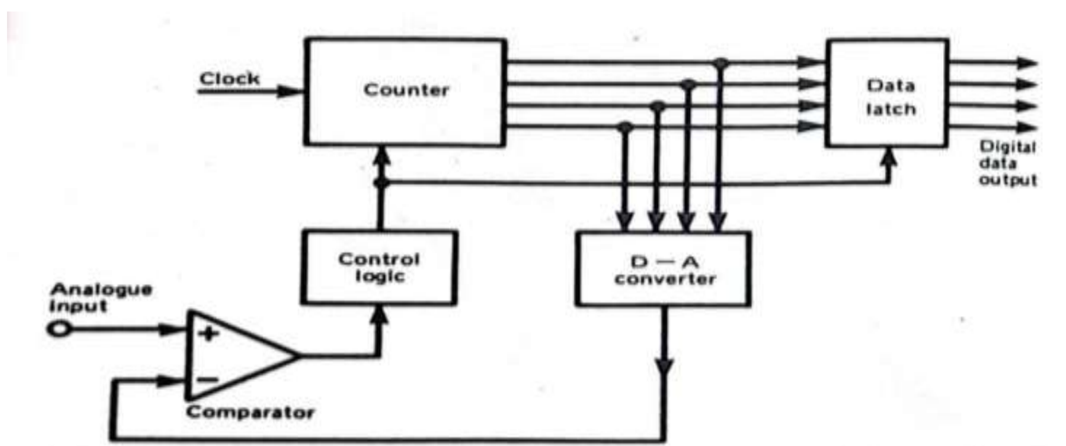

Рис.3.8. Аналого-цифровий перетворювач, що використовує цифровий лічильник і компаратор

Коли починається перетворення, логіка послідовного наближення встановлює старший біт у 1, а всі інші біти – у 0. Ці дані передаються на цифрово-аналоговий перетворювач і видають вихідний сигнал, який становить приблизно половину повної шкали. Тепер вхід порівнюється з виходом D-A. Якщо вхідний сигнал менший, він повинен бути менше половини повної шкали, а старший біт скидається на 0 логікою керування. Якщо вхідний сигнал більший, ніж вихідний сигнал D-A, біт залишається встановленим на 1. Логічна система тепер встановлює кожен біт по черзі та знову перевіряє, чи повинен біт залишатися встановленим чи бути скинутим. Коли останній біт оброблено, вихід цифро-аналогового перетворювача майже точно дорівнює вхідному сигналу, а слово даних, встановлене в регістрі, є потрібним цифровим виходом, який може бути передано в пам'ять центрального процесора за потреби. Ця схема завжди дає однакову кількість кроків, по одному для кожного біта, і типовий час перетворення становитиме від 10 до 25 нас для 12-бітового слова даних, що добре порівнюється з, можливо, 5-25 мс для системи лічильника, що працює на тій самій тактовій частоті.

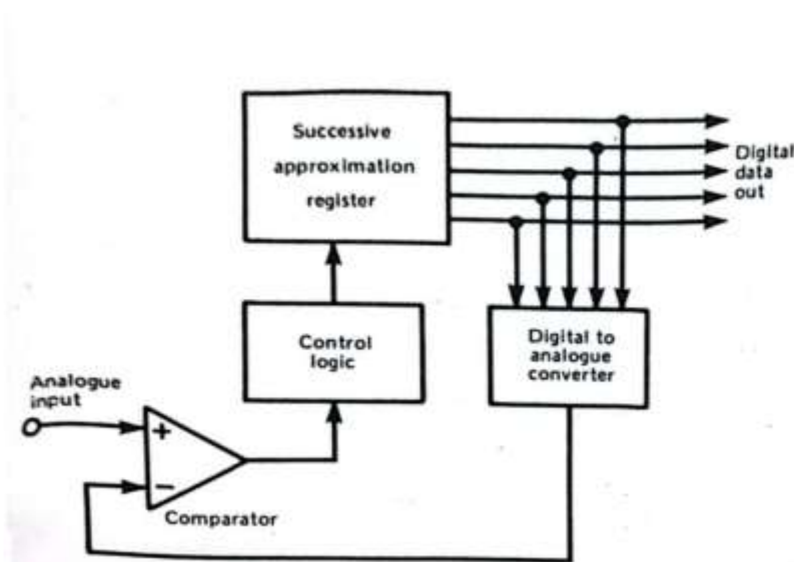

Рис.3.9. Внутрішнє розташування популярного аналого-цифрового перетворювача послідовного наближення

Для простих систем 8-розрядне слово використовується в аналого-цифрових і цифрово-аналогових перетворювачах. Це дає роздільну здатність або розмір кроку приблизно 0,5% повної шкали, що є достатнім для багатьох програм. Промислові системи часто використовують 12-бітове слово даних, щоб отримати роздільну здатність приблизно 0,025% повної шкали.

Кодування слова даних може бути або простим двійковим числом у діапазоні від 0 до, скажімо, 255, або воно може бути організоване як додаткове двійкове кодування, що дає значення від -128 до +127. Деякі типи мають вихід, організований як двійкове кодоване десяткове слово (BCD), і може використовуватися до 16 біт (чотири цифри BCD).

3.2 Проектування та реалізація структури.

3.2.1 Проектування апаратної системи на базі мікропроцесора: структура та розробка

Першим кроком у проектуванні будь-якої мікропроцесорної системи є визначення того, що саме система повинна робити. Після цього має бути можливість визначити, які вхідні та вихідні канали потрібні, а також тип сигналів, які вони оброблятимуть. На цьому етапі також можна оцінити обчислювальні завдання, які вимагаються від ЦП.

Існує понад 100 різних типів мікропроцесорів і мікрокомп’ютерів. Вибір можна дуже швидко звузити, вибравши довжину слова 4, 8 або 16 біт. Загалом 4-розрядні типи є однокристальними мікрокомп’ютерами з масковим програмуванням. Вони ідеально підходять для застосувань, де потрібно виготовити дуже велику кількість одиниць і де потрібна найпростіша апаратна схема. 4-розрядні типи використовуються для таких додатків, як контролери монетних автоматів, керування відеомагнітофонами, касові апарати, телефонні номеронабирачі та електронні іграшки.

Існує кілька сімейств 4-розрядних пристроїв, але всі, як правило, мають подібні можливості, а головні відмінності між типами полягають у діапазоні входів і виходів, розмірі пам'яті та пакеті, що використовується.

Якщо потрібна велика або складна обчислювальна функція або використовується графіка високої роздільної здатності в реальному часі, можна вибрати 16-розрядний процесор. Основними перевагами 16-розрядних типів є їхня висока пропускна здатність інструкцій і їх здатність використовувати дуже велику пам'ять.

Для більшості програм підходить 8-розрядний процесор. Хоча існує чимала кількість 8-розрядних типів, ймовірно, що вибір буде між Intel 8085, Zilog Z80, серією MOS Technology 6500 і Motorola 6800 або 6809. Це всі типи MOS, але якщо потрібна CMOS, то Можна вибрати серію RCA 1800 або National NSC800. Для більшості програм будь-який із цих типів, імовірно, буде однаково ефективним, але з точки зору обчислювальної потужності та універсальності 6809 і Z80 є лідерами. Остаточний вибір часто робиться на основі особистих уподобань або знайомства з одним конкретним сімейством процесорів.

Після вибору типу процесора постає питання дизайну плати. У випадку 4-розрядних пристроїв, імовірно, плата буде створена на замовлення відповідно до кінцевого продукту. З 8- і 16-розрядними типами існує багато готових одноплатних комп'ютерних систем, доступних із використанням плат Eurocards або аналогічних стандартних розмірів плати. Ці системи зазвичай складаються з центрального процесора, пам'яті, таймера/лічильника та різноманітних вхідних і вихідних портів і відповідатимуть більшості системних вимог. Зазвичай доступні інші відповідні плати з A-D або D-A та іншими мікросхемами підтримки. Використання цих готових карт спрощує проектування обладнання, хоча вони містять деякі надлишкові засоби.

3.2.2 Розробка компактної мікрокомп'ютерної плати: процес та особливості

Припустимо, що, подивившись на наявні готові комп'ютерні плати, немає жодної, що відповідає вимогам. Тепер необхідно спроектувати систему. Припустімо, що використовується 8-розрядний ЦП і що для програми та робочої пам’яті потрібно 1 КБ ОЗУ та ПЗУ. Також необхідні порт введення/виведення та лічильник таймера.

Загальна схема буде показана на рис. 3.10. Тут шина даних від процесора підключена паралельно до мікросхем пам'яті, вводу-виводу та таймера. У невеликій системі, такій як ця, ЦП матиме можливість безпосередньо керувати іншими мікросхемами. Якщо було більше пам'яті та мікросхем підтримки або шину даних потрібно було перенести на іншу плату, тоді набір буферів шини даних використовуватимуться на ЦП. Це двонаправлені буфери, і необхідно вжити відповідних заходів, щоб напрямок потоку даних через буфери контролювався лінією керування читанням/записом центрального процесора. На додатковій платі потрібен набір буферів шини, і ними також повинен керувати ЦП. Після роботи з шиною даних наступним кроком є розробка адреси, яка показана як простий логічний блок. У невеликій системі, такій як ця, 16-бітна адреса часто декодується лише частково. Тут є чотири основні пристрої, а саме RAM, ROM, порт введення/виведення та таймер. Ми можемо зручно розділити карту адрес із 64K слів на чотири по 16K сегментів і призначити по одному для кожного пристрою на шині, як показано на рис. 3.11.

Два старші біти адресної шини процесора (A14 і A15) подаються в один із чотирьох дешифраторів для отримання чотирьох рядків вибору адреси, як показано на рис. 8.3. Кожна з цих ліній вибору адреси відповідає на адреси в одному з чотири сегменти карти пам'яті. Таким чином, ми можемо використовувати рядок вибору 1 для адресації ОЗП і вибрати рядок 4 для ПЗУ.

Ці лінії зазвичай використовуються для керування входами вибору мікросхеми мікросхем RAM або ROM відповідно.

У деяких системах, таких як 6800, один або два інших сигнали синхронізації повинні бути стробовані логікою вибору адреси, щоб синхронізація імпульсу вибору виходу була правильною відносно внутрішньої синхронізації процесора.

Для RAM і ROM окремі слова в пам'яті вибираються шляхом переміщення рядків адреси мікросхеми пам'яті з молодших бітів слова адреси. Для пам’яті 1K використовуються молодші 10 бітів адреси (від A0 до A9). Біти A10, A11, A12 і A13 шини адреси не використовуються в цій системі.

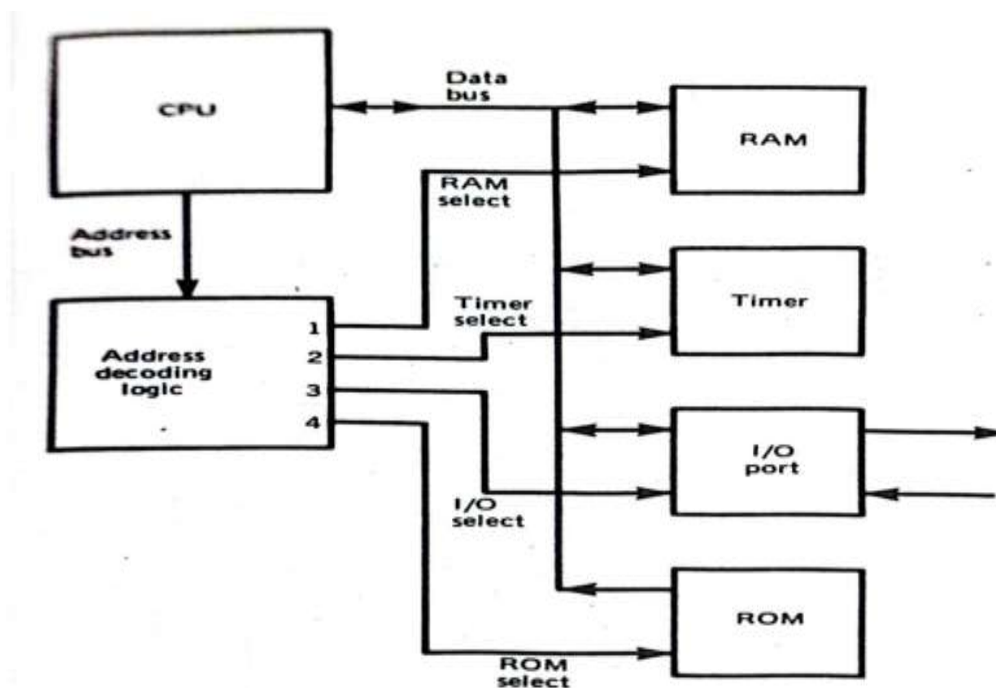

Рис.3.10. Типове розташування невеликої мікрокомп'ютерної системи

Порт вводу-виводу та таймер мають ряд внутрішніх регістрів, які вибираються двома або трьома бітами адреси, тому вони керуються лише адресними лініями від A0 до A2. Насправді порт вводу/виводу відповідає на кожну групу з восьми послідовних адрес у своїй області карти пам’яті, але це неважливо за умови, що його адреса не збігається з адресою іншого пристрою на шині.

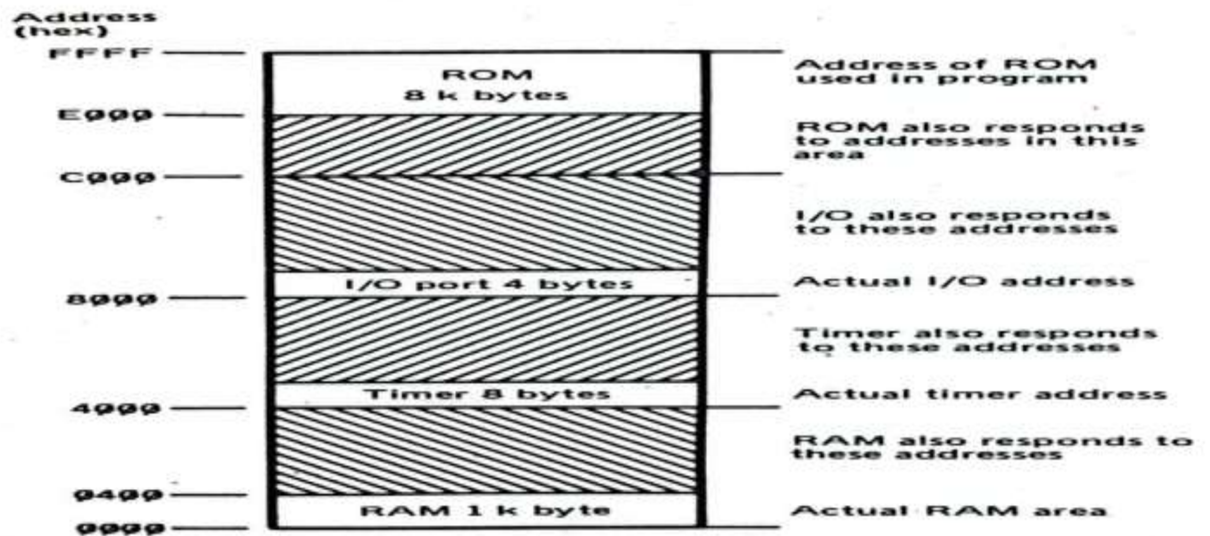

Рис.3.11. Типова карта пам'яті для невеликої системи з використанням часткового декодування адреси

Два порти вводу/виводу можуть бути розміщені в одній області карти пам’яті, якщо біт A4 використовується для вибору одного порту, а зворотний біт A4 вибирає інший порт.

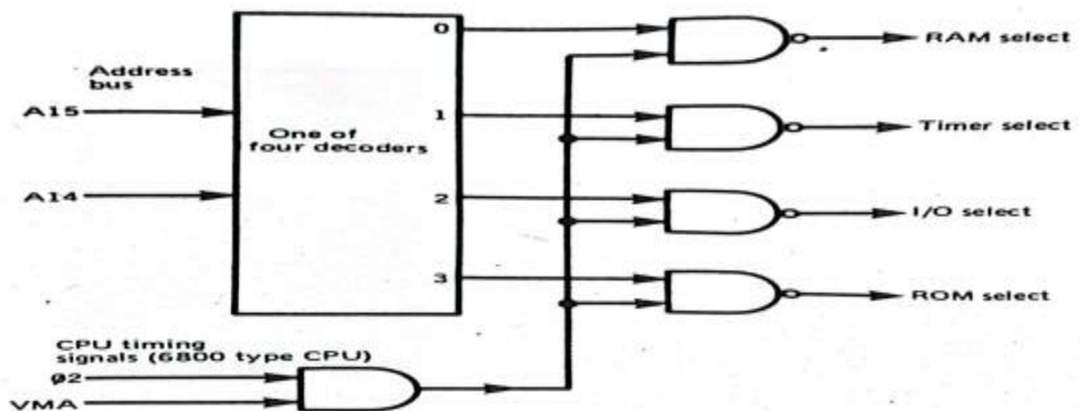

Рис.3.12. Типова схема декодування адреси для малої мікрокомп'ютерної системи

Багато пристроїв вводу/виводу мають два або три входи вибору мікросхеми, які дозволяють цей простий тип декодування адреси.

Розібравшись із адресною системою, потрібно звернути увагу лише на тактовий сигнал центрального процесора та схеми запуску. Годинник сучасних

процесорів вбудований у чіп і вимагає кварцового кристала з потрібною тактовою частотою та, можливо, одного або двох резисторів і конденсаторів.

Щоб запустити систему, потрібна послідовність скидання під час увімкнення живлення. Ця послідовність відбувається автоматично в центральному процесорі, коли на його вхід скидання подається імпульс. Імпульс скидання також може знадобитися застосувати до порту введення/виведення та мікросхем таймера. Часто спрацьовує проста ланцюг резисторів і конденсаторів, як показано на рис. 3.12. Коли подається живлення, лінія скидання утримується на низькому рівні протягом короткого часу, поки конденсатор заряджається через резистор. У деяких системах для надійнішої роботи використовується генератор одноразових імпульсів.

Імпульс скидання запускає послідовність скидання в ЦП і починає виконання програми. У системі 6800 ЦП приймає вміст двох верхніх місць у карті пам'яті як адресу першої інструкції, яка має бути виконана. У такій системі ПЗП має бути у верхній частині пам’яті, щоб забезпечити правильний запуск. У системах 8085 інструкція скидання береться з нижніх місць у пам’яті, тому тут ПЗП має бути розміщено внизу карти пам’яті.

Система розробки мікропроцесора — це комп’ютерна система, яка дозволяє писати та тестувати програму для проекту мікропроцесора, а також дозволяє тестувати або емулювати апаратне забезпечення проекту мікропроцесора.

У більшості випадків система розробки використовує той самий тип мікропроцесора, що й система, яка проектується, і зазвичай дозволяє перевірити та перевірити конструкцію апаратного та програмного забезпечення до того, як буде виготовлено будь-яке апаратне забезпечення для кінцевого продукту. Системи розробки бувають різних розмірів, що надає широкий спектр можливостей. У нижньому кінці розташовані оціночні дошки, які призначені для створення простих систем, а також можуть використовуватися як навчальні посібники для інженерів і програмістів. Деякі системи призначені

в основному для полегшення розробки та тестування програмного забезпечення. Більш комплексні системи зазвичай дозволяють установку апаратного забезпечення в системі розробки, щоб можна було перевірити його базову роботу.

Можуть існувати засоби, які дозволяють перевірити фактичне апаратне забезпечення проекту шляхом заміни функції процесора системою розробки. Також можуть бути надані засоби для аналізу роботи кінцевої системи, і в цьому випадку систему розробки можна використовувати як інструмент діагностики.

Досконаліші системи часто допускають роботу з декількома різними типами мікропроцесорів, що забезпечує більш-менш універсальний інструмент проектування.

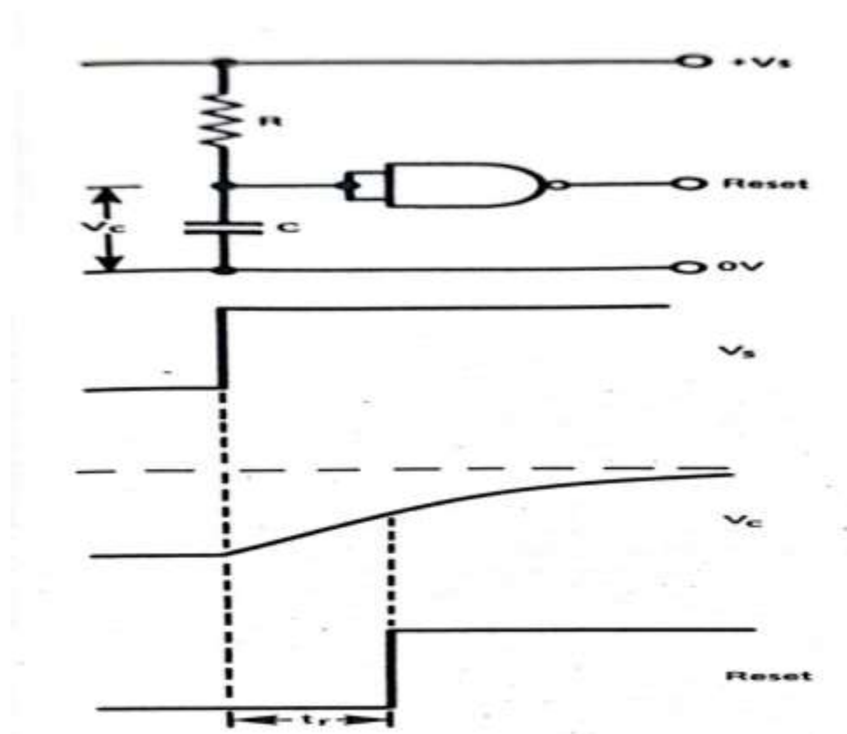

Рис.3.13. Проста схема для генерації сигналу RESET для мікропроцесора, такого як 6800

3.2.3 Оцінна картка: визначення та застосування в управлінні та аналізі.

На найнижчому рівні ряду допоміжних засобів розробки мікропроцесорів знаходиться оціночна картка або система. Зазвичай він складається з однієї або двох друкованих плат, що містять базову мікрокомп'ютерну систему, що використовує певний мікропроцесор. Ці пристрої в першу чергу призначені як допоміжні засоби для навчання, що дозволяє розробнику випробувати різні програмні інструкції, щоб побачити, як вони працюють, і поекспериментувати з основним розташуванням апаратного забезпечення, що дозволяє, можливо, проектувати та тестувати схеми інтерфейсу введення/виведення.

Система оцінювання зазвичай має просту клавіатуру, яка дозволяє перевіряти та змінювати дані в системі. У деяких системах може бути передбачена повна буквено-цифрова клавіатура, що дозволяє вводити як текст, так і числові дані. Простіші системи зазвичай мають певну форму світлодіодного дисплея для відображення вмісту ділянок пам'яті та стану внутрішніх регістрів ЦП. Більш складні версії можуть використовувати маленький принтер або, можливо, дисплей типу VDU. Деякі плати забезпечують можливість використання стандартного комп'ютерного терміналу VDU як вхід/вихід пристрій замість вбудованої клавіатури та дисплея. Апаратне забезпечення зазвичай складається з мікропроцесора та пов'язаних із ним схем годинника, деякої пам'яті для читання/запису та деяких вхідних вихідних портів для забезпечення взаємодії із зовнішніми пристроями. Управління роботою системи зазвичай здійснюється програмою моніторингу, яка постійно міститься в пам'яті лише для читання, наданій на платі. Програми користувача, які підлягають тестуванню, завантажуються в пам'ять для читання/запису для виконання. Ці прості системи зазвичай мають певну форму зберігання програм за допомогою магнітної стрічки та звичайного аудіокасетного диктофона.

Типовими для цих систем оцінювання є комплект Motorola D2 для процесора 6800, комплект SDK85 від Intel для процесора 8085 і плата AIM65 від Rockwell для процесорів серії 6500. Останній забезпечує невеликий принтер на платі для друку.

Хоча невеликі системи оцінювання можна використовувати для розробки досить складних мікрокомп'ютерних продуктів, їх досить важко використовувати для чогось більшого, ніж простий проект, і для серйозної роботи з розробки мікрокомп'ютерів зазвичай потрібна більш комплексна система.

В основному апаратна сторона повної системи розробки — це збільшена версія оцінювальної плати. Існує досить велика кількість пам'яті для читання/запису та більше засобів введення/виведення. Такі системи, як правило, складаються з вибору схемних плат у системі шасі, а конфігурація апаратного забезпечення може бути налаштована за бажанням шляхом вставки відповідних плат.

Для розробки програмного забезпечення існують програми-асемблери, які дозволяють програмне забезпечення для кінцевого продукту писати у мнемонічній формі, а не в машинному коді. Програма асемблера автоматично перекладає програму мнемонічної мови для створення остаточної версії машинного коду. Деякі системи також дозволяють використовувати мови високого рівня, такі як BASIC, FORTRAN і Pascal, які іноді можуть спростити розробку великих і складних програм.

Практично всі ці системи використовують блок пам'яті гнучких дисків для зберігання програм і файлів даних. Система гнучких дисків є набагато швидшою та більш гнучкою, ніж системи аудіокасет, які використовуються на простіших оціночних картках.

Повнорозмірна система зазвичай дозволяє перевірити кінцеве апаратне забезпечення за допомогою системи розробки, і часто будуть присутні засоби

для тестування апаратного забезпечення кінцевого продукту, що дозволяє діагностувати несправності та аналізувати роботу кінцевого продукту.

3.2.4 Універсальні розвиваючі системи: визначення та області застосування.

Більшість доступних систем розробки розроблені або на основі конкретного процесора, або на основі процесорів конкретного виробника, наприклад, систем Motorola Exorciser і Intel Intellec.

Якщо існує ймовірність появи мікропроцесорів різних марок, може знадобитися мати дві або більше різних систем розробки для обробки різних типів процесорів. Однак деякі виробники створили так звані універсальні системи розробки, які можуть працювати з широким діапазоном процесорів від різних виробників. Типовими з них є система Futuredata AMDS2300 і системи Tektronix 8500 Microprocessor Lab. Ці системи розроблені для роботи з широким діапазоном типів мікропроцесорів, які використовують загальне системне шасі.

Зазвичай набір друкованих плат потрібно змінити, щоб налаштувати систему на певний тип процесора. Зазвичай це передбачає зміну фактичної плати процесора та завантаження нової операційної системи з диска. Системи пам'яті та введення/виведення зазвичай загальні для всіх типів процесорів. Таким чином, щоб охопити діапазон типів, для кожного типу може бути друкована плата та диск. Очевидно, що система може бути розширена за потреби, коли потрібно працювати з новим типом пристрою.

3.2.5 Внутрішньосхемна емуляція: принципи та механізми роботи

Для тестування кінцевого апаратного продукту корисно, якщо можна використовувати засоби діагностики основної системи розробки. Це дало б можливість вставляти точки зупину та навіть трасування або покрокове виконання.

Ці діагностичні функції можуть бути забезпечені за допомогою внутрішньосхемної емуляторної системи, де чіп центрального процесора видаляється з апаратного забезпечення прототипу, а спеціальний датчик емулятора підключається до порожнього гнізда ЦП. Основну систему розробки тепер можна використовувати так, ніби це ЦП, і її можна змусити реагувати або на вбудовану програму, або на програму, збережену в системі розвитку. Точки зупину можна вставляти за вказаними адресами, оскільки система розробки зараз інтерпретує інструкції програми.

Внутрішньосхемний емулятор може також забезпечувати функцію логічного аналізатора, за допомогою якої можна відображати стан шин даних і адреси за кожним тактом і, можливо, навіть відображати їх у вигляді сигналів, якщо потрібно. Це дає змогу аналізувати роботу зовнішніх ланцюгів по відношенню до мікросхеми процесора в робочих умовах і може бути дуже корисним для діагностики та усунення несправностей у роботі.

3.3 Розробка програмного забезпечення: етапи та принципи

3.3.1. Розробка програмного забезпечення: визначення та процес роботи

Процес проектування мікропроцесорної програми, написання коду інструкцій і забезпечення правильної роботи програми називається розробкою програмного забезпечення.

Перший крок - це точно визначити, що має робити програма. Велика і складна програма зазвичай розбивається на серію модулів. Більшість із цих модулів, імовірно, написані як підпрограми, навіть якщо вони можуть з’являтися лише один раз у повній програмі. Основна перевага використання цієї модульної форми програмування полягає в тому, що окремі модулі можуть бути написані та створені для незалежного виконання своїх функцій. Деякі спеціальні тестові програми можуть знадобитися для тестування окремих модулів, але, як правило, легше знайти та виправити помилки в невеликій

частині програми, ніж мати справу з усією програмою відразу. Якщо над проектом працює більше ніж один програміст, кожен програміст може розробити модуль, який можна об'єднати під час роботи, щоб можна було зібрати та протестувати повну програму.

Після написання розділу програми його можна перетворити у форму машинного коду та перевірити, чи немає помилок. Цей етап відомий як налагодження, а самі помилки в народі називають помилками.

Зазвичай повна програма тестується на певній системі розробки. Нарешті, програмні дані можуть бути записані в один або декілька ПЗП або ППЗУ та протестовані у фактичній апаратній системі для завершеного проекту.

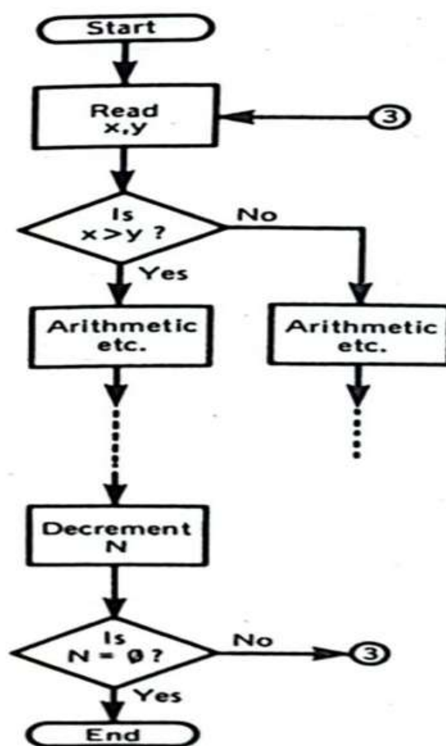

Рис.3.14. Частина типової блок-схеми

Вимоги до мікропроцесорної програми зазвичай починаються з довгого списку дій, але якщо програма довга або складна, може бути важко зрозуміти, як саме мають виконуватися інструкції. Це особливо вірно, якщо є багато тестів і операцій філій. Щоб спростити розуміння роботи такої програми, можна скласти блок-схему. Типовий розділ блок-схеми показано на рис. 3.14.

Зазвичай обчислення, підпрограми та введення/виведення відображаються як дії в прямокутному полі. Умовні випробування та гілки зображені ромбами з позначкою випробування. Основний потік програми входить у верхній частині ромбу і результати тесту «так» або «ні» виходять збоку або знизу.

У деяких випадках може бути дозволено три результати. Там, де програма використовує модулі або підпрограми, кожна з них має власну блок-схему, яка відображається в головній програмі всередині прямокутної рамки. У великій блок-схемі деякі маршрути розгалужень, якщо їх повністю намалювати, можуть заплутати діаграму, тому лінії розриваються поза рамками, які вони з'єднують, а коло з числом у ньому використовується, щоб показати, куди йде гілка на діаграмі. Прикладом цього є точки, позначені «3» на рис. 3.14. У будь-якій мікропроцесорній системі, призначеній для розробки програмного забезпечення та оцінки апаратного забезпечення, є спеціальна керуюча програма, яка називається монітором.

Програма моніторингу починає виконуватися, щойно подається живлення, і налаштовує всі вхідні/вихідні канали. Після цієї фази запуску монітор працює в інтерактивному режимі з користувачем, дозволяючи перевіряти чи змінювати місця пам'яті, завантажувати, виконувати чи зберігати програми та виконувати різноманітні процедури тестування.

Внутрішні регістри центрального процесора також можна перевіряти та змінювати за бажанням.Інші функції, надані монітором, включають вставку та видалення точок зупину в програмі, відстеження розділів програми та виконання однокрокових операцій.

Можуть бути передбачені засоби для відображення або друку вмісту блоків ділянок пам'яті як у шістнадцятковій формі, так і в еквівалентах символів ASCII. Деякі версії також надають можливість дизассемблера, де шістнадцятковий вміст пам'яті транслюється для забезпечення мнемонічної форми коду інструкції.

Роздруківка може бути схожою на оригінальний список мнемонічної програми, за винятком того, що адреси операндів є фактичними числами, а не символічними іменами.

Монітор зазвичай дозволяє завантажувати програму в пам'ять з аудіокасети і зберігати програму в пам'яті на касету.

Деякі системи також дозволяють записувати файли даних на магнітну стрічку або читання з неї. Можуть бути передбачені інші корисні підпрограми для обчислення відносних зміщень адреси або для перетворення чисел між десятковим і шістнадцятковим форматами.

3.4 Застосування точки зупинки при розробці програмного забезпечення

Під час тестування нової або модифікованої програми можуть бути помилки або навіть повний збій у виконанні. Щоб виявити причину, можна вставити контрольні точки за допомогою монітора. Точка зупину змушує виконання програми зупинятися за вказаною адресою інструкції, і керування повертається до монітора, щоб можна було перевірити вміст регістру та пам’яті на цьому етапі виконання програми. Потім виконання можна продовжити за наступною інструкцією. Використовуючи серію точок зупину, користувач може бачити, що відбувається під час виконання, і порівнюючи це з тим, що має відбуватися, зазвичай можна виявити помилку.

Щоб створити точку зупину, монітор може замінити код операції програми за вказаною адресою інструкцією SWI. В якості альтернативи виконується порівняння апаратного забезпечення між адресною шиною та адресою точки зупину, і коли відбувається збіг, генерується переривання. У будь-якому випадку відбувається переривання, і керування повертається до монітора. Після перевірки та, можливо, зміни даних регістра та пам'яті програму можна відновити командою з монітора.

Розширенням ідеї точки зупину є слід. Тут вказуються початкова та кінцева адреси для операції трасування, після чого запускається програма. Коли початкова адреса досягнута, монітор бере на себе керування та викликає відображення або друк вмісту регістру ЦП після виконання кожної інструкції. Тепер користувач може перевіряти роботу регістрів центрального процесора під час виконання програми. По суті, це схоже на точку зупинки для кожної інструкції. Після досягнення кінцевої адреси виконання програми продовжується в звичайному режимі. Трасування дозволяє детально перевірити дію сегмента програми та є цінним для відстеження помилок, викликаних неправильною логікою програми.

Варіантом трасування, який надається в багатьох системах розробки, є одноетапна операція. Тут при кожному натисканні клавіші виконується одна інструкція. Може бути можливим перевірити та змінити регістри або пам'ять на кожному кроці, щоб побачити, як саме працює програма.

Програмна інформація, необхідна мікропроцесору, складається виключно з послідовності чисел, які представляють коди операцій і операнди. Це відомо як машинний код. Надрукована або відображена версія машинного коду програми зазвичай є списком шістнадцяткових чисел.

Однак для зручності розуміння програми програміст-людина зазвичай використовує мову програмування, в якій коди операцій записуються в мнемонічній формі (наприклад, LDA, ADD тощо), а розташування даних і мітки інструкцій можуть мати зручні буквено-цифрові назви, такі як ШВИДКІСТЬ, ГЛИБИНА та ПЕТЛЯ. Однак для створення даних машинного коду мнемонічна версія програми повинна бути перекладена. Це, звичайно, можна зробити вручну, використовуючи таблицю, що показує мнемоніки та їхні еквівалентні номери машинного коду, але набагато швидше та зручніше використовувати комп’ютер для виконання цього завдання. Цей процес перекладу досягається за допомогою спеціальної програми, відомої як асемблер.

Вхідні дані для асемблера називаються вихідним кодом і зазвичай є текстовим файлом із переліком мнемонічної форми програми. Більшість монтажників використовують двопрохідну систему. Під час першого проходу зчитується вихідний код і транслюються коди операцій. Під час цього проходу всім змінним виділяються адреси пам’яті, і будь-які синтаксичні помилки у вихідному коді можуть бути виявлені та роздруковані повідомлення про помилки. Під час другого проходу всі переходи та розгалуження сортуються та створюється остаточний машинний код. Цей остаточний машинний код називається об'єктною програмою.

Програма на асемблері зазвичай має деякі спеціальні інструкції, відомі як директиви, які можна вставити в список програм так, ніби це звичайні інструкції. Ці директиви просто вказують асемблеру щось зробити і не перетворюються на машинний код для об’єктної програми. Типові директиви можуть вказувати початкову адресу для програми, вказувати дані, які потрібно розмістити в місцях пам'яті, або резервувати місця пам'яті для змінної, яка використовується в програмі.

Варіант стандартного асемблера відомий як макроасемблер. Ми вже бачили, що з часто використовуваними послідовностями інструкцій можна працювати, перетворюючи їх на підпрограми. Іноді, однак, програма працюватиме швидше або ефективніше, якщо набір інструкцій фактично повторюється протягом усієї програми. Цього можна досягти за допомогою так званих макрокоманд або макросів. По суті, кожен макрос визначає нову інструкцію для асемблера.

Припустимо, що процесор не має інструкції множення. Тепер ми можемо створити макрос із назвою MULT і визначити набір інструкцій, які виконуватимуть потрібну функцію множення. Ми також можемо визначити, якими будуть операнди. Тепер MULT можна використовувати як звичайний код операції в програмі асемблера. Коли програму буде зібрано, набір інструкцій, пов’язаних з MULT, буде вставлено в програму на цьому етапі.

Основна перевага використання макросів полягає в тому, що це спрощує вихідну програму та може зробити її легшою для розуміння. Немає переваги з точки зору кількості створеного машинного коду.

Звичайний асемблер, коли він перекладає мнемонічну програму, виділить абсолютні адреси машинному коду. Якби цей машинний код було завантажено в певну точку пам’яті, відмінну від його правильного розташування, він не міг би працювати, оскільки адреси, указані в програмі, були б неправильними.

Часто буде багато підпрограм або сегментів програми, які можуть часто використовуватися в багатьох різних програмах. За допомогою звичайного асемблера кожен з них повинен бути вставлений у вихідний код і зібраний разом, щоб сформувати остаточну завершену програму.

Було б зручно, якби ці часто використовувані модулі можна було б уже зібрати, а потім просто зв’язати зі зібраною версією решти програми перед завантаженням. Це означає, що асемблер повинен зарезервувати можливі значення для всіх адрес, але їх абсолютні значення не будуть отримані, доки всі сегменти програми не будуть пов’язані разом.

Асемблер, який може працювати таким чином, називається переміщувальним асемблером, а його вихід називається переміщуваним об'єктним кодом.

Оскільки він має справу з програмним модулем, асемблер створить таблицю символів, яка призначає значення адрес для імен змінних і міток інструкцій, що використовуються цим модулем. Деякі з них, зокрема мітки, будуть визначені модулем, що збирається, і їм буде надано адресу відносно початкової адреси модуля.

Однак багато імен змінних і міток використовуватимуться та визначатимуться іншими модулями. Вони відомі як глобальні імена, і асемблер включить їх у свою таблицю символів, але не призначить для них фактичну адресу.

Зазвичай директива асемблера на початку програмного модуля матиме ці глобальні імена як свої операнди, і вони будуть розглянуті пізніше, коли повна програма буде пов'язана разом.

Коли всі модулі для повної програми зібрані окремо, наступним етапом є їх об'єднання разом і створення фактичної програми машинного коду, яку можна завантажити в пам'ять і виконати. Це робиться програмою під назвою компонувальник.

Користувач вказує порядок, у якому різні модулі мають бути завантажені в пам'ять, а також може вказати, де в пам'яті мають починатися окремі модулі. Компонувальник тепер приймає кожен модуль по черзі та призначає фактичні значення адреси пам'яті його інструкціям, змінним і міткам.

Далі він робить перехресні посилання на всі глобальні імена та призначає значення адрес операндам, які їх використовують. Програма також перевіряє, чи всім іменам у таблиці символів кожного модуля призначено адресу та чи вся програма вміщується в доступний адресний простір, указаний користувачем. На цьому етапі компонувальник створить остаточний машинний код для програми.

Це досить спрощений опис програм асемблера та компонувальника, що переміщуються. Справжні асемблери та компонувальники зазвичай мають багато дуже гнучких опцій, і з побіжного погляду на посібник користувача вони можуть здатися надзвичайно складними та важкими у використанні.

На практиці, однак, основні операції цього типу асемблера досить прості, і його використання може зробити виробництво дуже великих програм простішим і менш схильним до помилок.

Після створення зібраного об'єктного коду використовується інша спеціальна програма, яка називається завантажувачем, щоб розмістити дані машинного коду в правильному місці пам'яті, готові до виконання.

3.5 Застосування інтерпретаторів та компіляторів

У асемблері мнемонічна версія інструкцій перетворюється безпосередньо в машинний код. Альтернативний підхід полягає в тому, щоб змусити процесор інтерпретувати кожну інструкцію під час виконання програми. Типовим прикладом мови програмування типу інтерпретатора є BASIC.

У мові BASIC використовуються команди виду

10 C = A + B

20 PRINT C,

у якому число на початку рядка надає номер оператора, а наступний вираз є простим формулюванням того, що ви хочете зробити від комп'ютера. Тут рядок 10 додає разом змінні A і B і встановлює змінну C рівною результату. Рядок 20 призводить до виведення C на екран дисплея або на принтер.

Під час виконання програми інтерпретатор BASIC зчитує кожен рядок по черзі, переводить його в машинний код, а потім виконує перед переходом до наступного рядка. Головним недоліком інтерпретаторів є те, що оскільки кожен оператор читається та перекладається, поки програма виконує роботу повільно. Набагато кращий підхід полягає в тому, щоб перевести всі оператори програми в машинний код і створити об'єктну програму майже так само, як це робить асемблер. Потім об'єктну програму можна безпосередньо виконати на нормальній швидкості процесора. Цей тип програми перекладу відомий як компілятор.

Мови програмування, призначені для роботи з інтерпретаторами або компіляторами, називаються мовами високого рівня, а типовими прикладами є BASIC, FORTRAN і Pascal. Ці мови набагато прості у використанні, ніж мови асемблера, оскільки дуже складні функції можна легко вказати в операторі, а вбудоване програмне забезпечення піклується про перетворення на рівень машинного коду. BASIC універсально використовується для персональних

комп’ютерів і комп’ютерів малого бізнесу, тоді як FORTRAN і Pascal в основному використовуються на міні-комп’ютерах.

Недоліком мов високого рівня є те, що кожен виробник прагне мати свій власний діалект мови, і програми, написані для однієї машини, зазвичай потрібно модифікувати, перш ніж вони будуть працювати на комп’ютері іншої марки. Передбачається, що Паскаль усуне цю проблему, оскільки його компілятор виробляє об’єктну програму у формі, що називається P-кодом, який теоретично має працювати на будь-якому комп’ютері, що працює з Паскалем. На практиці мова Паскаль, здається, пройшла майже так само, як BASIC і FORTRAN, причому кожен комп’ютер мав власний варіант стандартної версії.

Мови високого рівня можна використовувати для створення програмного забезпечення для невеликих спеціалізованих мікрокомп’ютерних систем, але отримана програма може бути менш ефективною та використовувати набагато більше пам’яті, ніж програма, написана мовою асемблера.

3.5.1 Переваги компілятора в порівнянні з інтерпретатором: аналіз та огляд

Головною перевагою використання компілятора є швидкість виконання. Пам'ятайте, що інтерпретатор, такий як BASIC, фактично читає та перекладає кожен рядок вихідної програми в машинний код під час виконання програми. У випадку компілятора переклад виконується один раз під час компіляції, а потім створюється програма машинного коду. Саме ця програма машинного коду виконується, тому скомпільована програма, як правило, працюватиме набагато швидше, ніж програма, яка виконується з вихідного коду інтерпретатором.

Однією з переваг інтерпретатора є те, що програму можна модифікувати та спробувати знову без необхідності її перекомпілювати. Процес компіляції може передбачати досить складну та трудомістку операцію кожного разу, коли вноситься зміна у вихідний код.

Незважаючи на те, що BASIC зазвичай виконується за допомогою інтерпретатора, також можливо мати компілятор BASIC, щоб розробка програми могла виконуватися за допомогою інтерпретатора, а потім остаточна версія програми могла бути скомпільована для створення швидкої версії машинного коду.

Висновки до розділу

У третьому розділі виявлено, що процеси асемблювання та компіляції є ключовими в створенні програмного забезпечення. Встановлено, що застосування асемблера та компілятора надає можливість написання ефективних програм для конкретної архітектури та забезпечує баланс між швидкістю виконання та зручністю розробки. Визначено, що використання переміщувальних асемблерів та компонувальників дозволяє створювати переміщувані програми, що можуть працювати на різних платформах.

ВИСНОВКИ

У результаті проведеного алгоритмічного аналізу мікропроцесорних основ операційних систем виявлено ключові фактори, впливаючи на їх ефективність та взаємодію з апаратним середовищем.

Дослідження архітектурних особливостей мікропроцесорів дозволило визначити оптимальні алгоритми та стратегії управління ресурсами, сприяючи поліпшенню продуктивності операційних систем на різноманітних апаратних платформах.

В ході проведеного алгоритмічного аналізу мікропроцесорних основ операційних систем виявлено, що висока ефективність та оптимальна робота операційних систем напряму залежать від якісної реалізації алгоритмів, використованих для взаємодії з мікропроцесором. Аналіз структури та ефективності алгоритмів планування завдань, управління ресурсами та обробки переривань свідчить про важливість їхньої оптимізації для забезпечення швидкодії та стабільності операційних систем.

В ході проведеного алгоритмічного аналізу мікропроцесорних основ операційних систем виявлено, що висока ефективність та оптимальна робота операційних систем напряму залежать від якісної реалізації алгоритмів, використованих для взаємодії з мікропроцесором. Аналіз структури та ефективності алгоритмів свідчить про важливість їхньої оптимізації для забезпечення швидкодії та стабільності операційних систем.

Результати алгортмічного аналізу підтверджують високий потенціал оптимізації, що виражається в пропозиціях конкретних рекомендацій для розробників та адміністраторів операційних систем.

Здобуті в ході дослідження знання можуть слугувати основою для подальших розвинених стратегій та інновацій в області оптимізації операційних систем на мікропроцесорних платформах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. "Computer Organization and Design: The Hardware/Software Interface" by David A. Patterson, John L. Hennessy.
2. "Microprocessor Architecture, Programming, and Applications with the 8085" by Ramesh S. Gaonkar.
3. "The Intel Microprocessors: Architecture, Programming, and Interfacing" by Barry B. Brey.
4. "ARM System Developer's Guide: Designing and Optimizing System Software" by Andrew N. Sloss, Dominic Symes, Chris Wright.
5. "MSP430 Microcontroller Basics" by John H. Davies.
6. "Microprocessor Design: A Practical Guide from Design Planning to Manufacturing" by Grant McFarland.
7. "Microprocessor Systems Design: 68000 Family Hardware, Software, and Interfacing" by Alan Clements.
8. "Embedded Systems: Introduction to Arm Cortex-M Microcontrollers" by Jonathan W. Valvano.
9. "PIC Microcontroller and Embedded Systems" by Muhammad Ali Mazidi, Janice Gillispie Mazidi, and Rolin D. McKinlay.
10. "Microprocessor Architecture: From Simple Pipelines to Chip Multiprocessors" by Jean-Loup Baer.
11. "Computer Organization and Design: The Hardware/Software Interface" by David A. Patterson, John L. Hennessy.
12. Assembly Language for x86 Processors" by Kip R. Irvine.
13. "Compilers: Principles, Techniques, and Tools" by Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman.
14. "Introduction to the Theory of Computation" by Michael Sipser.
15. "Programming Language Pragmatics" by Michael L. Scott.

16. "Організація комп'ютера та проектування. Архітектура комп'ютера" Девід Паттерсон, Джон Хеннесі.

17. "Машинне програмування для Intel-сумісних мікропроцесорів" Кіп Ірвайн (за американською книгою "Assembly Language for x86 Processors").

18. "Принципи компіляторів: теорія та практика" А. Ахо, Моніка Лам, Раві Сеті, Джеффрі Ульман.

19. "Вступ до теорії обчислень" автора Майкла Сіпсера.

20. "Прагматика мов програмування" автора Майкла Скотта (за американською книгою "Programming Language Pragmatics").

21. "Мікропроцесорна техніка та мікроконтролери" В. П. Резніченко, В. С. Харченко

22. "Мікропроцесори та мікроконтролери в пристроях автоматики" Ю. М. Рижика, О. О. Гарюк, Л. І. Чередник

23. "Мікропроцесорна техніка" В. І. Лупан

24. "Програмовані логічні контролери" В. П. Ганс

25. "Вбудовані системи. Проектування та програмування" Р. Ешер

26. Embedded Systems Design: An Introduction to Processes, Tools, and Techniques" by Arnold S. Berger.

27. "Programming Embedded Systems: With C and GNU Development Tools" by Michael Barr and Anthony Massa.

28. "Microcontroller Theory and Applications with the PIC18F" by M. Rafiquzzaman.

29. "ARM System Developer's Guide: Designing and Optimizing System Software" by Andrew N. Sloss, Dominic Symes, and Chris Wright.

30. "RTOS Concepts for Embedded Systems" by Dr. Rajkumar S. Thareja.

31. Таненбаум, Е., Бос, Г. (2014). "Операційні Системи: Проектування та Реалізація." Перекладене видання, Київ: Видавництво "Діа".

32. Сільверман, Г. (2015). "Основи мікропроцесорів і мікроконтролерів." Київ: Техніка.

33. Гендель, Т. (2016). "Операційні системи: архітектура та реалізація." Київ: Видавництво "БІНОМ".
34. Таненбаум, Е., Вудхалл, Е. (2014). "Розподілені системи: принципи та парадигми." Київ: Видавництво "Діа".
35. Сталінгс, У. (2018). "Комп'ютерні мережі та Інтернет." Київ: Видавництво "Ваклер".
36. Гусак, І.В. (2017). "Операційні системи та системне програмування." Київ: Ліга-Прес.
37. Скляренко, В. (2019). "Мікропроцесорні системи: архітектура, програмування, взаємодія з периферією." Київ: НТУУ "КПІ".
38. Андерсон, Т., Дал, Б. (2016). "Сучасні операційні системи." Київ: Видавництво "Діа".
39. Лукач, А., Семенець, О. (2015). "Мікропроцесорні технології: архітектура, діагностика, програмування." Київ: Ліга-Прес.
40. Познак, Т.В. (2018). "Архітектура комп'ютера та операційні системи." Київ: Видавничий дім "Ін Юре".