

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Запорізький національний технічний університет

Архітектура комп'ютера

Програма, самостійна робота та контрольні завдання
для студентів напрямку 6.050103 “*Програмна інженерія*”
заочної форми навчання

2011

Архітектура комп'ютера. Програма, самостійна робота та контрольні завдання для студентів напряму 6.050103 "Програмна інженерія" заочної форми навчання /Уклад.: Рисіков В.П., Степаненко О.О., Качан О.І. – Запоріжжя: ЗНТУ, 2011. – 42 с.

Укладачі: В.П. Рисіков, доцент, к.т.н.,
О.О. Степаненко, доцент, к.т.н.,
О.І. Качан, асистент

Рецензенти: В.Ф. Оніщенко, доцент, к.т.н.

Відповідальний
за випуск: А.В. Притула, доцент, к.т.н.

Затверджено
на засіданні кафедри
"Програмні засоби"

Протокол № 1 від 29.08.2011 р.

ЗМІСТ

Вступ	4
1 Мета і задачі дисципліни.....	4
2 Загальні вказівки.....	5
3 Програма дисципліни і методичні вказівки по її вивченню	6
4 Контрольні завдання.....	39
5 Список літератури, що рекомендується.....	42

ВСТУП

Розвиток народного господарства вимагає значного збільшення виробництва і широкого застосування високопродуктивних електронно-обчислювальних машин, промислових роботів і гнучких виробничих модулів, створення автоматизованих виробництв. Отже, експлуатація машин, устаткування і приладів, висуває специфічні вимоги до професійної підготовки фахівців даної спеціальності.

1 МЕТА І ЗАДАЧІ ДИСЦИПЛІНИ

1.1 Мета викладання дисципліни

Метою даного курсу є вивчення загальних принципів побудови і функціонування електронно-обчислювальних систем обробки даних і керування, придбання практичних навичок розробки і налагодження програмного забезпечення.

1.2 Задачі вивчення дисципліни

У результаті вивчення дисципліни студенти повинні:

- вивчити загальні принципи побудови електронно-обчислювальних систем;
- знати архітектуру ЕОМ;
- придбати навички в програмуванні і налагодженні програм.

1.3 Зв'язок з іншими дисциплінами

Отримані по розглянутій дисципліні знання будуть використовуватися і доповнюватися спеціальними дисциплінами "Мікроелектроніка і схемотехніка ЕОМ", "Мікропроцесори в системах обробки даних і керуванні", "Комп'ютерні мережі", "Архітектура обчислювальних систем", "Системне програмне забезпечення", а також бути корисними в курсовому і дипломному проектуванні.

2 ЗАГАЛЬНІ ВКАЗІВКИ

Дана дисципліна включає лекційні, лабораторні, практичні заняття, а також виконання двох контрольних завдань.

Працювати над дисципліною необхідно систематично, вивчаючи теми в зазначеному порядку. При цьому необхідно користатися не тільки основний, але і додатковою літературою.

Виклад матеріалу в навчальній літературі по темах звичайно не завжди відповідає програмі дисципліни. Тому рекомендується вивчати матеріал вибірково, керуючись програмою теми і не обмежуючись одним джерелом.

При самостійному проробленні матеріалу варто також врахувати, що контрольні завдання складені так, щоб результат їх виконання був покладений в основу виконання лабораторного практикуму.

Підведення підсумків вивчення дисципліни здійснюється у вигляді заліку й іспиту. До здачі іспиту допускаються студенти, що успішно виконали всі контрольні завдання, лабораторний практикум і ті, хто отримав залік.

3 ПРОГРАМА ДИСЦИПЛІНИ І МЕТОДИЧНІ ВКАЗІВКИ ПО ЇЇ ВИВЧЕННЮ

3.1 Системи числення. Застосування систем числення в обчислювальній техніці

Поняття системи числення. Позиційні і непозиційні с/ч. Переклад чисел з однієї с/ч в іншу. Арифметичні дії з числами в різних с/ч. Двійково-десятькова с/ч.

Системою числення (с/ч) називається спосіб представлення числа за допомогою цифр. Будь-яке число може бути представлене у виді послідовності цифр. Існують позиційні і не позиційні с/ч. У позиційній системі с/ч значення будь-якої цифри в записі числа визначається не тільки самою цифрою, але і позицією, що ця цифра займає в числі. У не позиційній с/ч значення цифри не залежить від її положення в числі (Номер: "римська" с/ч XXX).

В обчислювальній техніці застосовують позиційні с/ч (Номер "десятькова" 129,36). Назва "десятькова", "двійкова", пояснюється тим, що в підставі цієї с/ч лежить десять, два. У двійковій с/ч для запису числа застосовуються цифри: 0 і 1, у восьмеричній: 0, 1, 2, 3, 4, 5, 6, 7, у десятковій: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, у шістнадцятиричній: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Для кожної цифри числа підстава с/ч зводиться в ступінь, залежний від позиції цифри і збільшується на цю цифру. У якості початку рахунка позицій використовується кома, причому при рахунку позицій цифр від коми вліво ступінь с/ч приймає послідовно значення 0, 1, 2,..., а при рахунку вправо - значення: -1, -2, -3... Номер для числа 129,36₁₀:

$$1 \cdot 10^2 + 2 \cdot 10^1 + 9 \cdot 10^0 + 3 \cdot 10^{-1} + 6 \cdot 10^{-2}$$

Для перекладу з десяткової с/ч у будь-яку іншу поділяють число на підставу цієї с/ч із запам'ятовуванням залишку. Потім виписують усі залишки починаючи з останнього, це і буде число в новій с/ч. Номер 891₁₀ записати у восьмеричної с/ч:

$$\begin{array}{r} 891 \overline{) 8} \\ 888 \quad 111 \overline{) 8} \\ 3 \quad 104 \quad 13 \overline{) 8} \\ \quad 7 \quad 8 \overline{) 1} \\ \quad \quad 5 \end{array}$$

$$891_{10} = 1573_8$$

Арифметичні операції з числами в будь-який с/ч здійснюється по тим же правилам, що і з десятковими числами, за винятком того, що переноси в наступні розряди виробляється при досягненні підстави с/ч. Номер:

Перенос	1 111	11	11	1
1-ий додаток	101001102	166 ₁₀	246 ₅	A6 ₁₆
2-ий додаток	001011112	47 ₁₀	57 ₈	2F ₁₆
Результат	11010101	213	325	D5

У цифрових пристроях використовується двійково-десяткова с/ч. У двоїчно-десяткової с/ч десяткові цифри від 0 до 9 представляють 4-розрядними двійковими комбінаціями від 0000 до 1001. Перетворення з десяткової в двійково-десяткову і назад виконуються шляхом прямої заміни десяткової цифри чотирма двійковими чи навпаки.

Для зручності виконання ручних операцій по підготовці і налагодженню програм для ЕОМ двійкові коди команд і даних звичайно заміняють на восьмеричні і шістнадцятирічні.

Для перекладу двійкового числа у восьмеричне, число розділяють з права наліво на групи з трьох двійкових цифр, потім кожен групу цифр виражають у виді восьмеричної. Номер: $10.100.010_2 - 10_2 = 28$, $100_2 = 4_8$, $010_2 = 2_8$, $10100010_2 = 242_8$, аналогічно перетворюють у шістнадцятирічну з тією лише різницею, що двійкове число поділяють на групи з чотирьох двійкових цифр. Перетворення числа з восьмеричної чи шістнадцятирічної с/ч у двійкову здійснюється перекладом кожної цифри вихідного числа в групу з трьох (восьмерична) чи чотирьох двійкових цифр.

Двійкове число можна перетворити в десяткове, подавши його у виді суми ступенів числа 2.

Основу с/ч часто вказують у виді індексу для числа, узятото в круглі дужки.

Контрольні питання і вправи

1. Що розуміється під поняттям с/ч?
2. Чим позиційна с/ч відрізняється від непозиційної?
3. Які знаки використовуються як цифри в двійкової, восьмеричної, шістнадцятирічної с/ч?
4. Як виконуються арифметичні операції в різних с/ч?

5. Як перевести з десяткової с/ч в іншу?
6. Як перевести з двійкової у восьми-, шістнадцятиричну с/ч й назад?

3.2 Структура і принципи функціонування ЕОМ

Структура ЕОМ. Пристрій процесора. Регістри. Пам'ять. Пристрої введення і виводу інформації.

Методичні вказівки

ЕОМ складається з процесора, пам'яті, пристроїв введення-виводу. Процесор складається з:

1. Арифметико-логічного пристрою, що робить операції над даними.
2. Регістрів, що здійснюють тимчасове збереження даних і стану процесора.
3. Пристрої керування, що здійснює процес вибірки команд із пам'яті і їхнього виконання.

Процесор організує обробку інформації шляхом вибірки і послідовного виконання команд програми, що знаходиться в пам'яті в ЕОМ.

Призначення регістрів процесора:

1. Лічильник команд - служить для організації звертання до комірок пам'яті, у яких зберігається програма. Наприкінці кожного циклу виконання команди лічильник вказує на адресу комірки пам'яті, що містить наступну команду програми. Тому що команди заносяться у послідовні комірки пам'яті, а при виконанні поточної команди вміст лічильника збільшується на одиницю і він автоматично вказує адресу наступної команди. У такий спосіб здійснюється передача керування іншій частині програми.

- 2.Регістр команд (РК) - цей n-розрядний регістр використовується для збереження команди, яка безпосередньо використовується машиною. Код операції команди пересилається з РК у пристрій керування і декодується. Після цього відбувається дія по реалізації команди: зчитування операнда і виконання операції.

- 3.Регістр адреси (РА) - він містить значення виконавчої адреси комірки пам'яті. Якщо ЕОМ здійснює вибірку команди, то в РА

пересилається вміст вказівки адреси комірки, де зберігається команда. Якщо ж ЕОМ робить вибірку даних, адреса може надійти з РК.

4.Регістр даних (РД) - їх небагато, вони використовуються для тимчасового збереження слів при обміні інформацією між пам'яттю і процесором. При пересиланні даних із процесора в пам'ять вони спочатку містяться в РД і лише потім записуються в потрібну комірку пам'яті.

5.Акумулятор (А) - один з головних елементів процесора. ЕОМ може виконувати операції над двома операндами одночасно. Звичайно перший операнд витягається з пам'яті в РД, а другий завжди знаходиться в А. Результат операції міститься в А.

6.Регістр переносу (З) - однорозрядний регістр, виступає як продовження А и заповнюється при переповненні А. Також використовується при організації циклічних зсувів.

Пам'ять ЕОМ включає пристрій, що забезпечує збереження команд і даних. Основна пам'ять ЕОМ складається з двох частин - оперативного запам'ятовуючого пристрою (ОЗУ) і постійного запам'ятовуючого пристрою (ПЗУ). Пам'ять складається з блоків однакового розміру, що іменуються комірками пам'яті. Вони призначені для збереження одного слова інформації. У свою чергу, комірка пам'яті складається з елементів пам'яті (бітів). Сукупність нулів і одиниць «збережених» в елементах однієї комірки являють собою вміст цього осередку. Комірки пам'яті звичайно нумерують числами десяткової чи шістнадцятиричної с/ч.



Рисунок 3.1 – Узагальнена схема типової ЕОМ

Якщо необхідно записати в пам'ять слово, варто подати на шину адреси пам'яті сигнали, що відповідають адресі комірки, у яку треба помістити записуване слово і подати саме слово на шину даних.

Пристрої введення забезпечують зчитування інформації та її представлення у формі електричних сигналів, що сприймали іншими пристроями ЕОМ: процесором, пам'яттю.

Пристрої виводу представляють результати обробки інформації у формі, зручній для візуального сприйняття.

Контрольні питання і вправи

1. Які пристрої входять у типову ЕОМ?
2. Призначення і складові процесора?
3. Які типи і призначення регістрів процесора?
4. Призначення і принцип дії пам'яті ЕОМ?
5. Які функції пристроїв вводу-виводу інформації?

3.3 Команди EOM

Система команд ЕОМ. Класифікація команд. Способи адресації.

Методичні вказівки

Система команд - це фіксований, обмежений по розмірах список команд. Частина команди називаються - полями команди. Кожна команда містить поле коду операції (КОП). Крім того, багато команд мають одне чи більш полів операндів. У залежності від їхнього числа розрізняють формати одне-, двох- і триадресних команд:

15 12 11 0

КОП	Розширення коду операції
-----	--------------------------

-безадресова

$$15 \quad 12 \ 11 \quad 0$$

КОП.	Адреса операнда, результату або переходу
------	---

-одноадресова

15 12 11 9 8 6 5 3 2 0

КОП	Адреса 1-го операнда	Адреса 2-го операнда	Адреса результату	Адреса наступної команди
-----	----------------------------	----------------------------	----------------------	--------------------------------

- чотирьохадресна

При написанні програми можуть кодуватися чотирма способами:

1. У двійковій системі.
2. У восьмеричній с/ч.
3. У шістнадцятирічній с/ч.
4. Мнемонічне (символічне) кодування.

Записувати команду в двійковому коді незручно. Середня програма складається із сотень команд, що спричинить запис тисяч нулів і одиниць. Для скорочення використовується система з підставою 2^3 і 2^4 . Вибір для кодування команд тієї чи іншої с/ч зв'язаний з форматом команд ЕОМ. Зручніше шістнадцятирічна с/ч, тому що більшість полів у командах кратно чотирьом, при мнемонічному кодуванні кожна команда представляється простим двох-, трьох- чи чотирибуквеним мнемонічним символом, що легше зв'язати з машинними операціями.

У залежності від того до яких блоків машини звертається чи команда на які блоки вона посилається, команди поділяють на три групи:

1. звертання до пам'яті (адресні команди);
2. звертання до реєстрів (реєстрові або безадресні);
3. команди вводу-виводу.

Команди звертання до пам'яті наказують машині робити дії з вмістом комірки пам'яті, адреса якого зазначена в адресній частині.

Безадресні команди виконують різні дії без посилання на комірку пам'яті. Команди вводу-виводу здійснюють обмін даними між ЕОМ і зовнішніми пристроями. В залежності від функцій, виконуваних командою, команди поділяються на групи:

1. Пересилань чи обміну.
2. Арифметичні.
3. Логічні.
4. Зсувів.
5. Переходів.
6. Звертання до підпрограми.
7. Керування.
8. Вводу-Виводу.

По способу адресації команди поділяються на дві групи:

1. З прямою адресацією.
2. З непрямою адресацією.

При прямому адресуванні в команді зберігатися адреса комірки пам'яті де зберігається операнд. При непрямому адресуванні в команді зберігається не сама адреса операнда, а його покажчик, тобто адреса комірки пам'яті в якій зберігається адреса операнда.

Для вказівки виду адресації в командах використовується біт з номером 11, у який при прямій адресації записується 0, а при непрямої -1.

Контрольні питання і вправи

1. Основні частини команди?
2. Які формати команд бувають у залежності від числа полів операндів?
3. Способи кодування команд?
4. Класифікація команд у залежності від того до яких блоків ЕОМ йде звертання?
5. Класифікація команд у залежності від виконуваних функцій?
6. Які є способи адресації?

3.4 Логічні операції і команд

Логічне множення. Логічне додавання. Логічне заперечення. Застосування логічних команд в ЕОМ.

Методичні вказівки

Алгебра висловлень - найпростіший розділ мат.логіки, об'єктом якого є пропозиція, щодо якої істи можливість говорити правдиве воно чи помилкове.

Логічне множення. з'єднання двох висловлень А и В в одне за допомогою союзу "І" називається операцією логічного множення, а отримане складене висловлення - логічним множенням.

$$A \bullet B \Leftrightarrow AB \Leftrightarrow A * B$$

A	B	A*B
1	1	1
1	0	0
0	1	0
0	0	0

Логічне додавання. З'єднання двох висловлень А и В в одне за допомогою союзу "ЧИ" наз. логічним додаванням, а нова складова висловлення - логічною сумою. $(A+B)$

A	B	A+B
1	1	1
1	0	1
0	1	1
0	0	0

Логічне заперечення. Вироблятися не над двома, а над одним висловленням. Приєднання до присудка, що виражає дію, частки "НІ" чи до усього висловлення "НЕВІРНО" називається операцією логічного заперечення, а отримане нове висловлення - логічним запереченням вихідного.

A	$\overline{A}$
1	0
0	1

Якщо дане висловлення правдиве, то його заперечення помилкове і навпаки. Таблиці істинності демонструють зв'язку між собою багатьох висловлень. Таблицею істинності - називається таблиця за допомогою якої підраховується істинність складних за структурою висловлень.

A	$\overline{B}$	C	$\overline{B}$	A+B	$(A+B)*C$
1	1	1	0	0	0
1	1	0	0	0	0
1	0	1	1	0	0
1	0	0	1	0	0
0	1	1	0	1	1
0	1	0	0	1	0
0	0	1	1	0	0
0	0	0	1	0	0

Основні еквівалентності:

$\overline{\overline{A}} = A$ (правило зняття подвійного заперечення) $A * A = A$ (кон'юнкція)

$A + A = A$ (диз'юнкція)

$A * (C + B) = (A * C) + (A * B)$

$A + (C * B) = (A + C) * (A + B)$

$A * B = A + B$ и $A + B = A * B$ (правила Де Моргана)

$A + (A * B) = A$ и $A * (A + B) = A$ (правила поглинання)

Керування обчислювальним процесом і логічними операціями.

Майже всі програми ЕОМ повинні мати можливість перевіряти вихідні дані, проміжні й остаточні результати, щоб на підставі результатів перевірки змінювати потрібним образом послідовність виконання операцій. Такі задачі зважаються за допомогою команд переходу, збільшення, зупинок. Усі ці команди, крім зупинок, є адресними. У цих командах вказується адреса тієї комірки пам'яті, з якого повинна бути обрана наступна команда при виконанні наступного умови. Якщо ж ця умова не виконується, виконується наступна команда, розташована за поточною командою керування. Розгалуження при виконанні програм організується за допомогою команд переходу. Ці команди не змінюють стан акумулятора і регістра переносу. Вони можуть змінювати вміст лічильника команд, помістивши туди адресу обумовлений адресною частиною команди. Циклічні програми використовуються в тих випадках, коли необхідно кілька разів виконати набір однакових дій над деяким складом даних, що змінюється.

Побітова обробка даних забезпечується в ЕОМ командами логічного множення, циклічного зсуву вліво, вправо а також командами інвертування й очищення регістра переносу. Ці команди часто використовуються для упакування інформації в пам'яті ЕОМ (розміщення в одному слові декількох невеликих чисел або кодів) і її розпакування, обробки текстової інформації, множення (розподілу) на 2, 4, 8... і т.п. Команда логічного множення $ANA\ r$ ($ANA\ m$) здійснює порозрядне "І" над вмістом регістра r (комірки пам'яті m) і акумулятора.

Контрольні питання і вправи

1. Як виконується логічне множення?
2. Як виконується логічне додавання?
3. Як виконується лог. заперечення?
4. Які задачі здійснюється за допомогою команд переходу, збільшення, зупинок?
5. Які команди застосовуються для побітової обробки даних?

3.5 Виконання машинних команд

Стан ПК ЕОМ. Вибірка команди. Вибірка адреси. Виконання команд.

Методичні вказівки

В процесі виконання машинних команд пристрій керування (ПК) робить аналіз і пересилання команди, окремих її частин (коду операції, ознаки адресації й адреси) з одного регістра машини в інший її регістр, АЛУ, чи пам'ять пристрій вводу-виводу. Ці дії інакше наз. мікроопераціями і протікають у визначеній тимчасовій послідовності і скоординовані між собою. Будь-яка ЕОМ може знаходитися в чотирьох можливих станах:

1. Вибірки команди.
2. Вибірки адреси.
3. Виконання.
4. Переривання.

Цикл команди. Для реалізації однієї команди потрібно виконати визначену кількість мікрокоманд, кожна з яких ініціює одночасне здійснення однієї чи декількох мікрооперацій за час робочого такту ЕОМ. Машинним циклом наз. час у плині якого виконується читання команди з чи пам'яті вибірка операнда. Отже, цикл команди містить у собі від одного до декількох машинних циклів.

Вибірка команди. У даному машинному циклі виконується читання команди з пам'яті, її часткове декодування й іноді виконання:

- а) уміст комірки пам'яті, що вказується лічильником команд, читається з пам'яті в регістр даних;
- б) уміст лічильника команд збільшується на одиницю;
- в) уміст регістра даних пересилається в регістр команд, код операції команди частково декодується для виявлення типу команди (адресна, безадресна, вводу-виводу). Аналізується біт ознаки адресації

і відбувається підготовка ланцюгів, необхідних для виконання команди;

д) якщо обрана адресна команда, то здійснюється перехід до мікрокоманд наступного машинного циклу; у противному випадку виконуються дії по завершенню одноциклової команди.

Вибірка адреси. Цей машинний цикл йде за вибіркою команди для адресних команд із непрямою адресацією. Цикл використовується для читання адреси операнда (адреса результату або переходу) з пам'яті і складається з наступних кроків:

а) адресна частина команди пересилається з реєстра даних у реєстр адреси;

б) вміст комірки пам'яті, що вказується реєстром адреси, читається в реєстр даних; тепер у цьому реєстрі знаходиться адреса операнда, що буде використовуватися при виконанні команди;

в) вміст реєстра даних збільшується на одиницю;

д) змінений вміст реєстра даних пересилається в комірку пам'яті за адресою, що вказується реєстром адреси;

ж) вміст реєстра даних зменшується на одиницю.

Виконання. Послідовність дій, виконуваних у цьому машинному циклі, визначається типом виконуваної адресної команди. Послідовність кроків у цьому циклі наступна:

а) для команд, при виконанні яких потрібно вибірка операнда з пам'яті ЕОМ, машинний цикл використовується для читання операнда в реєстр даних і виконання операції, що вказується кодом операції;

б) по команді пересилання в цьому машинному коді виробляється запис вмісту акумулятора в комірку пам'яті з адресою, розташованною у реєстрі даних, для чого вміст РД пересилають у РА, а вміст акумулятора - у РД і далі в комірку пам'яті, що вказується реєстром адреси;

в) при виконанні команд переходів виробляється перевірка відповідності умови і пересилання адреси з РД у СК при виконанні цієї умови. Якщо виконуються команди безумовного переходу, то пересилання адреси переходу в СК здійснюється без перевірки;

д) для команди звертання до підпрограми під час цього машинного циклу здійснюється пересилання вмісту З К в комірку пам'яті, адреса якої міститься в РД, занесення в З До збільшеного на одиницю вмісту РД.

Контрольні питання і вправи

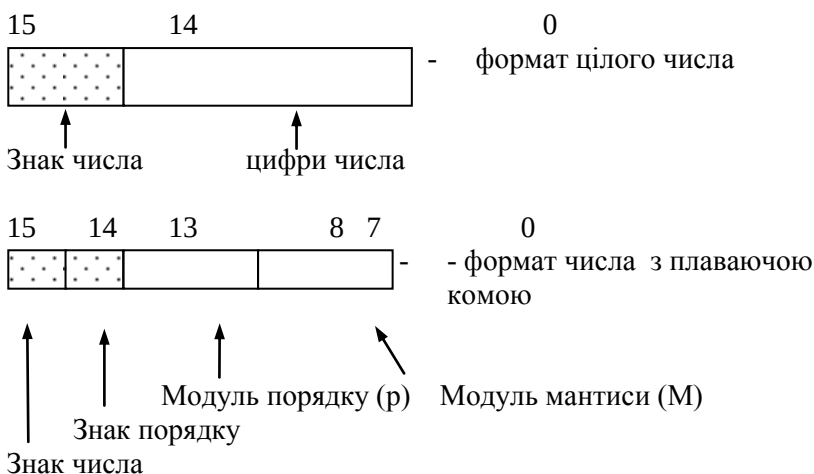
1. У якому з чотирьох станів може знаходитися ПК?
2. Що таке машинний цикл, цикл команди?
3. Яка послідовність виконання машинного циклу "Вибірка команди"?
4. Які мікрооперації виконуються при "Вибірці адреси"?
5. Яка послідовність дій у машинному циклі "Виконання"?

3.6 Арифметичні операції

Представлення чисел в ЕОМ. Додавання двійкових чисел. Різні випадки додавання чисел зі знаком.

Методичні вказівки

Формат представлення чисел в ЕОМ



Якщо необхідно скласти два двійкових числа, то операція відбувається за правилом десяткової арифметики. Якщо число перевершує шістнадцять розрядів, то зайва цифра переходить в регістрі переносу.

Цілі двійкові позитивні і негативні числа. Щоб відрізнити негативні числа від позитивних можна використовувати або знак (таке представлення чисел називається прямим кодом), або величину числа (додатковий код). Додавання чисел у прямому коді складно реалізувати технічно, тому числа представляють у виді додаткових

кодів, що при підсумовуванні двох чисел із протилежними знаками дозволяє замінити вирахування на додавання. Доповнення M n -розрядного цілого числа K можна представити у виді $M = b^n - K = ((b^n - 1) - K) + 1$, де b -основа с/ч. У машині негативне число, перетвориться в додатковий код по наступному алгоритму:

1. Одержати інверсію заданого числа (усі його 0 замінити на 1, а всі 1 - на 0). Номер: 0010100101 - число (прямий код) 1101011010 - інверсія числа (зворотний код)

2. Утворити додатковий код заданого числа шляхом додавання 1 до інверсії цього числа (додатковий код). Перевірити правильність переводу, якщо скласти додатковий код і число, вийде одиниця в сімнадцятому розряді і шістнадцять нулів.

Додавання цілих двійкових чисел зі знаком виконується по тим же правилам, що і для чисел без знака.

Для розширення діапазону представлення великих чисел і збільшення точності розрахунку використовується розміщення цілого двійкового числа в декількох словах пам'яті. Підсумовування багатослівних чисел виконується в наступній послідовності:

1. Зробити додавання молодших слів чисел, при цьому в регістр переносу автоматично занесеться 1 чи 0 у залежності від відсутності чи наявності переносу, що виникає при додаванні старших розрядів підсумованих слів;

2. Записати отриману суму в молодше слово результату;

3. Зробити додавання наступних по старшинству слів і вмісту регістра переносу, наприкінці операції в регістр переносу заноситься перенос зі старшого розряду підсумованих слів (1 або 0);

4. Отриману суму записати у відповідне слово результату;

5. Повторювати п. 3 і п. 4 доти, поки не будуть складені всі слова доданків.

Контрольні питання і вправи

1. У якому форматі представляються числа в ЕОМ?
2. Поняття прямого і зворотного коду числа?
3. Як утвориться додатковий код числа?
4. Коли видається сигнал переповнення при виконанні арифметичних операцій?
5. Як виробляється підсумовування багатослівних чисел?

3.7 Додаткові операції

Збільшення на одиницю і зменшення на одиницю. Зміна знака числа. Вирахування. Множення і розподіл.

Методичні вказівки

Збільшення на одиницю (INCrement і зменшення на одиницю (DECrement). По команді INC до вмісту акумулятора додається одиниця, а по команді DEC - одиниця віднімається. Якщо при цьому з'являється перенос зі старшого розряду, то в регістр переносу (3) заноситься 1, у протилежному випадку - 0.

Зміна знака числа. Для зміни знака числа, записаного в додатковому коді, необхідно його інвертувати, а потім додати одиницю до молодшого розряду.

Вирахування (X-Y). Вирахування вимагає виконання декількох команд і може виконуватися шляхом додавання зменшуваного X і додаткового коду що віднімається Y. Для скорочення програми і часу виконання в ЕОМ передбачена команда вирахування SUB Y, що реалізує ті ж дії за менший час. При виконанні вирахування можливе переповнення, ознакою чого є одержання позитивного результату при $X < 0$ і $Y > 0$ чи негативний результат при $X > 0$ і $Y < 0$.

Множення та ділення. У базовій ЕОМ немає команд для здійснення множення і ділення, тому користувач повинний організовувати одержання чи добутку частки програмним шляхом. При бінарному множенні частковий добуток зсувається на один розряд уліво для кожного наступного розряду множника. Якщо множник дорівнює 0, то частковий добуток дорівнює нулю, якщо множник дорівнює 1, той частковий добуток дорівнює множеному.

Контрольні питання і вправи

1. Як виконуються операції збільшення і зменшення числа на одиницю?
2. Які дії виконуються для зміни знака числа?
3. Як виконується операція вирахування в машині?
4. Операції множення і ділення в базовій ЕОМ?

3.8 Мікропрограмний рівень. Багаторівнева ЕОМ

Компіляція. Інтерпретація. Схема виконання задачі по етапах. Рівні мов програмування.

Методичні вказівки

Є два способи організації процесу перекладу вихідної програми з алгоритмічної мови на машинну мову і процесу її виконання:

1. Компіляція. Полягає в тому, що процес виконання алгоритму здійснюється лише після завершення процесу перекладу вихідної програми. При цьому в процесі виконання отриманої в результаті перекладу машинної програми вже не потрібна не тільки вихідна програма, але і програма-компілятор, що здійснив її переклад у машинне представлення.

2. Інтерпретація. Полягає в тому, що кожен оператор вихідної програми замінюється програмою-інтерпретатором на еквівалентну послідовність машинних команд, що відразу виконується і здійснюється послідовний переклад і реалізація команд програми. Отже, при рішенні задачі на машині використовується і вихідна програма і програма-інтерпретатор.

Витрати на написання програм-трансляторів залежать від ступеня відмінності компільованої й одержуваної мов. Тому сама процедура перекладу алгоритмічної мови на машинну може виконуватися по етапах у кілька циклів.

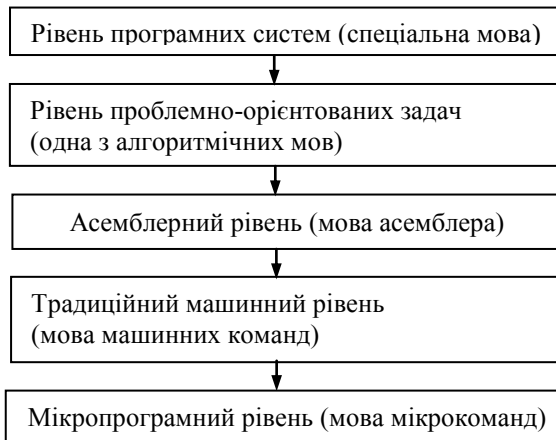


Рисунок 3.2 – Багаторівнева ЕОМ

Багаторівнева ЕОМ - це обчислювальна машина, що має засоби для роботи з різними рівнями мов програмування. Нижній рівень, є найбільш простим, верхній - найбільш складним. Таку машину можна розглядати як n різних віртуальних машин, кожна з яких має свою машинну мову. Схема виконання задачі по етапах:

1. Постановка задачі.
2. Методика рішення.
3. Алгоритм.
4. Програма алгоритмічною мовою.
5. Трансляція.
6. Програма машинною мовою.

Контрольні питання і вправи

1. Які розходження між компіляцією й інтерпретацією?
2. Що таке багаторівнева ЕОМ. З яких мов вона складається?
3. Які етапи виконання задачі?

3.9 Периферійні пристрої

Класифікація периферійних пристроїв. Кодування інформації. Способи обміну даними між ЗП й ЕОМ.

Методичні вказівки

Зовнішні (периферійні) пристрою можна розділити на три групи:

1. Пристрою для зв'язку людина - ЕОМ (клавіатура, індикатори,...)
2. Пристрою для зв'язку з об'єктами керування (різні датчики і виконавчі органи, а також пристрою для перетворення безупинних сигналів з датчиків у цифрові сигнали і зворотне перетворення)
3. Зовнішні запам'ятовуючі пристрої великої ємності.

Більшість ЗП дозволяє людині спілкуватися з ЕОМ на звичній для неї мові слів і десяткових чисел, а ЕОМ розуміє лише мову фізичних станів, то у ЗП звичайно виробляється кодування (декодування) інформації, що пересилається в (з) ЕОМ. Для кодування російських (33 літери) і латинських (26 літер) прописних і рядкових, а також графічних і службових символів необхідно мати більш ніж

$2^7=128$ різних кодів. Тому кодування виробляється у восьми бітовому (1-байтовому) коді обміну інформацією. В усіх закордонних ЕОМ кожному символу відповідає код від 0 до 255. Символи від 0 до 127 - однакові для всіх. Для знаходження помилок обміну даними між процесором (пам'яттю) і ЗП коди символів перетворюють у яку-небудь стандартну формулу. Найчастіше це додавання одного біта. При передачі такого стандартного коду, виробляється контроль визначеного виду. Наприклад, контроль парності по його результатам приймається рішення про справність отриманого символу. Якщо під час передачі помилково зміниться значення одного біта, то число одиниць в отриманому символі стане непарним і користувач одержить сигнал про помилку.

Способи зв'язку між ЕОМ і зовнішніми пристроями(ЗП).

Існує два способи передачі слів інформації з ліній даних:

1. рівнобіжний, коли одночасно пересилаються всі біти слова;
2. послідовний, коли всі біти слова пересилаються по черзі.

Існує два режими послідовної передачі: синхронний і асинхронний. При синхронній передачі кожен переданий біт даних супроводжується імпульсом синхронізації, що інформує приймач про наявність на лінії інформаційного біта. Між передавачем і приймачем повинне бути три лінії: дві для передачі бітів даних і одна заземлений провідник.

Синхронізація послідовної передачі починається з пересилання в приймач одного чи двох символів синхронізації. Приймач починає прийом даних і перетворює їх у рівнобіжний формат; при такій синхронізації доцільно використовувати лише пересилання масивів слів, а не окремих символів. При асинхронній передачі в передавача і приймача немає загального джерела синхроімпульсів і синхронізуючий сигнал не посиляється разом з даними.

Контрольні питання і вправи

1. Які пристрої відносять до периферійних?
2. На які категорії поділяються периферійні пристрої?
3. Як кодується інформація при пересиланні в (з) ЕОМ?
4. Які способи зв'язку між ЗП й ЕОМ?
5. Які режими послідовної передачі даних?

3.10 Система переривань

Призначення системи переривань. Класи переривань. Дії комп'ютера при виникненні переривань.

Методичні вказівки

Система переривань комп'ютера призначена для того, щоб змусити ЕОМ звернути увагу на особливі умови, що виникають у ній самій чи іншому устаткуванні, і переключитися на виконання іншої програми. Безпосередньою причиною такого (переключення) переривання процесора з однієї програми на іншу є сигнал переривання. Нова програма, що мікропроцесор починає виконувати в результаті впливу сигналу переривання, називається програмою обробки переривання. Характер цієї програми залежить від джерела виникнення переривань. Необхідність переривань диктується тим, що в ЕОМ часто потрібно відкласти виконувану в даний момент програму і терміново виконати іншу, більш важливу програму, для того щоб пізніше повернутися, якщо можна, до продовження перерваної програми.

У залежності від джерела всі переривання в ЕОМ прийнято поділяти на класи (або типи):

1. Внутрішньомікропроцесорні переривання, викликування деякими подіями в МП (наприклад, переривання через апаратний збій чи при спробі МП виконати команду з помилковим кодом операції).

2. Внутрішньосистемні переривання, що виникають унаслідок тих чи інших подій у зовнішніх пристроях (наприклад, переривання від деякого пристрою вводу-виводу в моменти готовності до передачі цим пристроєм чергового байта даних).

3. Переривання, навмисно закладені в програму що виконується за допомогою спеціальних команд МП.

4. Міжмашинні переривання, що виникає, наприклад, при посилю особливих (комунікаційних) сигналів від однієї ЕОМ до іншої.

Кожна система переривань може автоматично або за бажанням програміста заборонити, чи, як ще говорять, замаскувати, весь чи заданий клас переривань МП. Таким чином, маскування переривань дозволяє тимчасово (до тих пір, поки переривання не буде розмасковано, тобто дозволено) не реагувати на всі чи обрані сигнали переривань. Маскування переривань у різних МП

здійснюється по-різному. Нерідко використовується спеціальний реєстр МП, який називається реєстром маски переривань.

Основними діями МП при виникненні переривань є:

1. розпізнання (ідентифікація) джерела сигналу переривання;
2. запам'ятовування в основній пам'яті стану перерваної програми;
3. маскування повторних переривань від даного джерела сигналу переривань;
4. виконання програми обробки переривання;
5. відновлення стану і продовження перерваної програми.

Перша дія необхідна для того, щоб правильно реагувати на даний сигнал переривання, тому що для кожного класу переривання мається окрема програма обробки переривань.

Друга дія дозволяє після завершення програми обробки переривання продовжити перервану програму.

Третя дія призначена для того, щоб виключити небажане повторне переривання від того ж самого джерела до завершення обробки першого переривання.

Четверта дія є основною і, як правило, найбільш тривала. Характер переривання, тобто програма обробки переривання, залежить від класу переривання.

П'ята дія забезпечує продовження перерваної програми після виконання програми обробки переривання.

В ЕОМ використовується, як правило, векторна система переривання. У цій системі для кожного класу переривань в основній пам'яті мається вектор переривання, що звичайно містить два елементи - початкова адреса програми обробки переривання і словостан процесора для цієї програми.

Контрольні питання і вправи

1. Навіщо призначений сигнал переривань, програма переривань?
2. Класи переривань?
3. Дайте визначення поняттям замаскованих і розмаскованих переривань?
4. Визначте основні дії МП при виникненні дозволених переривань?

3.11 Турбо Асемблер.

Загальні зведення про Турбо Асемблере. Режими адресації і команди пересилання даних. Арифметичні операції.

Методичні вказівки

Турбо Асемблер є могутньою мовою, що працює з командним рядком та сприймає ваші вихідні файли (файли з розширенням .ASM) і створює з них об'єктні модулі (файли з розширенням .OBJ). Після цього ви можете використовувати програму - компоновщик фірми Borland Tlink.exe, що відрізняється високою швидкістю компонування, для компонування отриманих об'єктних модулів і створення виконуваних файлів (файлів з розширенням .EXE та .COM).

Турбо Асемблер створений для роботи з процесорами серії 80x86 і 80x87 (більш докладно набір інструкцій процесорів серії 80x86/80x87 описаний у відповідних посібниках фірми Intel). У Турбо Асемблері мається дуже могутній і гнучкий синтаксис командного рядка. Якщо ви запустите Турбо Асемблер, не задавши ніяких аргументів, наприклад:

TASM

тоді на екран виведеться довідкова інформація, (англійською мовою) описуюча безліч параметрів командного рядка та синтаксис для специфікації асемблерних файлів.

Turbo Assembler Version 2.0 Copyright (3) 1990 by Borland International, Inc

Usage:

TASM [параметри] вик_файл [,об'єкт_файл] [,лістинг] [пер_посилки]

/a,/s	Упорядкування елементів по алфавітному порядку чи порядку початкового коду
/c	Генерація у лістингу перехресних посилань
/dSYM[=VAL]	Визначається SYM=0 чи SYM=VAL
/e/r	Емулюємі чи дійсні інструкції з плаваючою точкою
/h/?	Виводиться дана довідкова інформація
/PATH	Файли, які включаються, шукаються по маршруту визначеному PATH

/jCMD (наприклад IDEAL)	Визначає початкову директиву Асемблеру
/kr#/,ks# рядка #	Місткість хеш-таблиці #, місткість об'єму
/l/la розширений	Генерація лістингу: l-звичайний, la-
/ml/mx/mu ml=усі, mx=глобальні, mu=не розрізняються	Розрізнимість у регістрі букв ідентифікаторів:
/mv#	Задає максимальну довжину ідентифікаторів.
/m#	Дозволяє виконання декількох проходів для
дозволення випереджаючих посилань	
/h (ідентифікаторів)	Вилучення у лістингах таблиці символів
/p захищеному режимі	Перевірка перекриття сегменту коду у
/q компоновці	Вилучення записів .OBJ не потрібних при
/t асемблюванні	Вилучення повідомлень при успішному
/w0,/w1,/w2 попереджень, w1=w2=ε попередження	Завдання рівня попередження w0=нема
/w-xxx, /w+xxx	Заборона чи дозвіл попередження типу xxx
/x асемблювання	Включення у листинги блоків умовного
/zi, /zd відлагодження: zi=повна, zd=тільки про номери рядків	Інформація про ідентифікатори для

Директива SEGMENT

Директива SEGMENT визначає початок сегмента. Мітка, що вказується в даній директиві, визначає початок сегмента. Наприклад, директива:

Cseg SEGMENT

визначає початок сегмента з ім'ям Cseg. Директива SEGMENT може також (необов'язково) визначати атрибути сегмента, включаючи вирівнювання в пам'яті на межу байта, слова, подвійного слова, параграфа (16 байт) або сторінки (256 байт). Інші атрибути містять у собі спосіб, за допомогою якого сегмент буде комбінуватися з іншими сегментами з тим же ім'ям і класом сегмента.

Директива ENDS

Директива ENDS визначає кінець сегмента. Наприклад:

Cseg ENDS

завершує сегмент з ім'ям Cseg, що починався по директиві SEGMENT. При використанні стандартних директив визначення сегментів ви повинні явно завершувати кожен сегмент.

Директива ASSUME

Директива ASSUME указує Турбо Асемблеру, що в значення якого сегмента встановлений даний сегментний регістр. Директиву ASSUME CS: потрібно вказувати в кожній програмі, у якій використовуються стандартні сегментні директиви, тому що Турбо Асемблеру необхідно знати про сегмент коду для того, щоб встановити виконувану програму. Крім того, звичайно використовуються директиви ASSUME DS: та ASSUME ES:, дякуючи яким Турбо Асемблер знає, до яких комірок пам'яті ви можете адресуватися в даний момент.

Директива ASSUME дозволяє Турбо Асемблеру перевірити допустимість кожного звертання до іменованої комірки пам'яті з урахуванням значення поточного сегментного регістра. Розглянемо наступний приклад:

```
Data1 SEGMENT WORD 'DATA'
```

```
Var1 DW 0
```

```
Data1 ENDS
```

```
Data2 SEGMENT WORD 'DATA'
```

```
Var2 DW 0
```

```
Data2 ENDS
```

```
Code SEGMENT WORD 'CODE'
```

```
ASSUME CS: CODE
```

```
ProgramStart:
```

```
mov ax, Data1
```

```
mov ds, Ax ; встановити DS у Data1 ASSUME DS: Data1
```

```
mov ax, [Var2] ; спробувати завантажити Var2 в AX – це  
приведе до помилки,
```

```
; тому що Var2 недоступна в сегменті Data1
```

Турбо Асемблер відзначає в даній програмі помилку, тому що в ній робиться спроба одержати доступ до змінної пам'яті Var2, коли регістр DS встановлений у значення сегмента Data1 (до Var2 не можна адресуватися, поки DS не буде встановлений у значення сегмента Data2).

Виділення даних

Тепер, коли ви знаєте, як створювати сегменти, давайте розглянемо, як можна заповнити ці сегменти осмисленими даними. Сегмент стека проблеми не представляє: там знаходиться стек, а до стеку ви можете звернутися за допомогою інструкцій PUSH і POP і адресуватися через регістр BP. Сегмент коду заповнюється інструкціями, що генеруються відповідно до мнемоніки інструкції вашої програми, тому проблеми тут також немає.

Залишається сегмент даних. У Турбо Асемблері передбачена безліч способів визначення змінних у сегменті даних, як ініційованих деякими значеннями, так і неініційованих.

Ініціалізовані дані

Директиви визначення даних DB, DW, DD, DF, DP, DQ і DT дозволяють вам визначити змінні в пам'яті різного розміру:

DB 1 байт
DW 2 байти = 1 слово
DD 4 байти = 1 подвійне слово

DF, DP 6 байт=1 покажчик далекого типу (386)

DQ 8 байт = одне четвертне слово

DT 10 байт

Наприклад:

ByteVar DB 'Z' 1байт

WordVar DW 101b 2 байти (1 слово)

DwordVar DD 2BFh 4 байти (1 подвійне слово) 8 байт (1 четвертне слово)

QwordVar DQ 307o

TwordVar DT 100 10 байт

Ініціалізація масивів

В одній директиві визначення даних може вказуватися кілька значень. Наприклад, директива:

SampleArray DW 0, 1, 2, 3, 4

DD 25, 36, 49, 64, 81

DD 100, 121, 144, 169, 196

створює масив з ім'ям SampleArray, елементи якого мають розмір у слово. У директивах визначення даних можна використовувати будь-як число значень.

Турбо Асемблер дозволяє вам визначити блок пам'яті, ініціалізований зазначеним значенням, за допомогою операції DUP. Наприклад:

```
BlankArray DW 100h DUP (0)
```

Тут створюється масив BlankArray, що складається з 255 (десят.) слів, ініціалізованих значенням 0. Аналогічно, директива:

```
ArrayOfA DB 92 DUP('A')
```

створює масив з 92 байт, кожний з яких ініціалізований символом А.

Ініціалізація рядків символів

Символи являють собою припустимі операнди директив визначення даних, тому рядок символів можна визначити в такий спосіб:

```
String DB 'A', 'B', 'C', 'D'
```

У Турбо Асемблері в цьому випадку передбачена також зручна скорочена форма:

```
String DB 'ABCD'
```

Режими адресації пам'яті

Як при використанні операнда в пам'яті задати ту комірку пам'яті, з яким ви хочете працювати? Очевидна відповідь полягає в тому, щоб привласнити необхідної змінної пам'яті ім'я (як ми це робили в останньому розділі). За допомогою, наприклад, впливають операторів ви можете відняти перемінну пам'яті Debts (борги) з змінної пам'яті Assets (майно):

```
Assets DW?
```

```
Debts DW?
```

...

```
mov ax, [Debts]
```

```
sub [Assets], ax
```

У дійсності мова Асемблера забезпечує кілька різних способів адресації до рядків символів, масивів і буферів даних. Найбільш простий спосіб полягає в тому, щоб вважати дев'ятий по рахунку символ рядка CharString:

CharString DB 'ABCDEFGHJKLM'

....

```
mov ax, @Data
mov ds, ax
mov al, [CharString+ 8]
```

У даному випадку це теж саме, що:

```
mov al, [100+ 8]
```

тому що CharString починається зі зміщення 100. Усе, що укладено в квадратні дужки, інтерпретується Турбо Асемблером, як адреса тому зміщення CharString і 8 складається і використовується як адресу пам'яті. Інструкція приймає вид:

```
mov al, [108]
```

Такий тип адресації, коли комірка пам'яті задається її ім'ям, плюс деяка константа, називається безпосередньою (прямою) адресацією. Хоча безпосередня адресація - це гарний метод, вона не відрізняється достатньою гнучкістю, оскільки звертання виконується щораз по тій самій адресі пам'яті. Тому давайте розглянемо іншої, більш гнучкий шлях адресації пам'яті. Розглянемо наступний фрагмент програми, де в регістр AL також завантажується дев'ятий символ CharString:

```
mov bx, OFFSET CharString+ 8
mov al, [bx]
```

У даному прикладі для посилання на дев'ятий символ використовується регістр BX. Перша інструкція завантажує в регістр BX зміщення CharString (згадайте про те, що операція OFFSET повертає зміщення мітки в пам'яті) плюс 8. (Обчислення OFFSET і додавання для цього виразу виконується Турбо Асемблером під час асемблювання). Друга інструкція визначає, що AL потрібно скласти з вмістом по зміщенні в пам'яті, на яке вказує регістр BX

```
mov al, [108]
```

Квадратні дужки показують, що як операнд - джерело повинен бути використан осередок, на яку вказує регістр BX, а не сам регістр BX. Не забувайте вказувати квадратні дужки при використанні BX як покажчик пам'яті. Наприклад:

```
mov ax, [bx] ; завантажити AL з комірки ;пам'яті, на яку указує
BX
```

```
mov ax, bx ; завантажити в AX вміст регістра BX
```

це дві зовсім різні конструкції. ВХ - це не єдиний регістр, який можна використовувати для посилання на пам'ять. Допускається також використовувати константи, мітки, регістри ВР, SI і DI. Загальний вид операндів у пам'яті виглядає в такий спосіб:

[базовий регістр + індексний регістр + зсув]

де базовий регістр - це ВХ чи ВР, індексний регістр - SI чи DI, а зсув - будь-яка 16-бітова константа, включаючи мітки і вирази. Щораз, коли виконується інструкція, що використовує операнд у пам'яті, процесором 8086, ці три компоненти складаються. Кожна з трьох частин операнда в пам'яті є необов'язковою, хоча (це очевидно) ви повинні використовувати один із трьох елементів, інакше не одержите адреси в пам'яті. От як елементи операнда в пам'яті виглядають в іншому форматі:

ВХ SI
або + або + Зсув
ВР DI
(база) (індекс)

Існує 16 способів завдання адреси в пам'яті:

[зсув]	[bp+ зсув]
[Бх]	[bx+ зсув]
[si]	[si+ зсув]
[di]	[di+ зсув]
[bx+ si]	[bx+ si+ зсув]
[bx+ di]	[bx+ di+ зсув]
[bp+ si]	[bp+ si+ зсув]
[bp+ di]	[bp+ di+ зсув]

де зсув - це те, що можна звести до 16-бітового постійного значення. Може показатися, що 16-режимів адресації - це дуже багато, але якщо ще раз подивитися на цей список, ви побачите, що всі режими адресації виходять усього з декількох елементів, комбінуємих різними шляхами. От ще кілька способів, за допомогою яких можна, використовуючи різні режими адресації, завантажити дев'ятий символ рядка CharString у регістр AL:

CharString DB 'ABCDEFGHJKLM', 0

mov ax @Data
mov ds, ax

```
mov si, OFFSET [CharString+ 8]
mov al, [si]
```

```
mov bx, 8
mov al, [CharString+ bx]
```

```
mov bx, OFFSET CharString
mov al, [bx+8]
```

```
mov si, 8
mov al [CharString+ si]
```

```
mov bx, OFFSET CharString
mov di, 8
mov al, [bx+ di]
```

```
mov si, OFFSET CharString
mov bx, 8
mov al, [si+ bx]
```

```
mov bx, OFFSET CharString
mov si, 7
mov al, [bx+ si+ 1]
```

```
mov bx, 3
mov si, 5
mov al, [bx+ CharString+ si]
```

Усі ці інструкції посилаються на ту саму комірку пам'яті [CharString]+ 8.

У даному прикладі можна знайти кілька цікавих моментів. По-перше, ви повинні розуміти, що знак плюс (+), використовуваний усередині квадратних дужок, має спеціальне значення.

Переміщення даних - це, звичайно, важлива функція, оскільки комп'ютер витрачає істотну частину свого часу на переміщення даних з одного місця в інше. Однак в однаковій мірі важливо мати можливість маніпулювання даними, виконуючи над ними арифметичні і логічні операції. Тому далі ми розглянемо арифметичні і логічні операції, що підтримуються процесором 8086.

Арифметичні операції

Навіть якщо ваш комп'ютер PC і не витрачає увесь час на роботу з числами й обчислювальні операції, ви знаєте, що він може це зробити, якщо вам це буде потрібно. Крім того, на комп'ютері PC може працювати безліч електронних таблиць, програм баз даних і інженерних пакетів. Якщо прийняти все це в увагу, то стає досить очевидним, що комп'ютер IBM PC повинен володіти могутніми обчислювальними здібностями. Крім того, у процесорі 8086 не передбачено арифметичних і логічних інструкцій, що можуть безпосередньо працювати з операндами, розмір яких не перевищує 16 біт.

Додавання і вирахування

Інструкція ADD виконує додавання операнда - джерела (правого операнда) із вмістом операнда - приймача і записує результат в операнд - приймач. Інструкція SUB робить теж саме, тільки вона віднімає операнд - джерело з операнда - приймача.

Наприклад, інструкції: *BaseVal DW 99*

Adjust DW 10

mov dx, [Base Val]

add dx, 11

sub dx, [Adjust]

Спочатку завантажують значення, записане в BaseVal, у регістр DX, потім додають до нього константу 11 (у результаті в DX виходить значення 110) і, нарешті, віднімають з DX значення 10, записане в перемінній Adjust. Отримане в результаті значення 100 зберігається в регістрі DX.

32-розрядні операнди

Операції ADD і SUB працюють з 8- чи 16-бітовими операндами. Якщо ви, приміром, хочете скласти або відняти 32-розрядні операнди, ви повинні розбити операцію на ряд операцій зі значеннями розміром у слово і використовувати інструкції ADC і SBB. Коли ви складаете два операнда, процесор 8086 записує стан у прапор переносу (біт C в регістрі прапорів), що показує, чи був виконаний перенос із приймача. Ви знайомі з принципом переносу в десятковій арифметиці: якщо ви складаете 90 і 10, то одержуєте перенос у третю цифру (розряд).

Інструкція ADC аналогічна інструкції ADD, але в ній враховується прапор переносу (попередньо встановлений попереднім додаванням). Усякий раз коли ви складаєте два значення, що перевищують по розмірі слово, то молодші (менш значущі) слова потрібно скласти за допомогою інструкції ADD, а інші слова цих значень - за допомогою однієї чи декількох інструкцій ADC, останніми складаючи самі значущі слова. Наприклад, необхідні інструкції складають значення в регістрах CX:BX, розміром у подвійне слово, зі значенням, записаним у регістрах DX:AX:

```
add ax, bx
```

```
adc dx, cx
```

а в наступній групі інструкцій виконується додавання четвертного слова зі змінною DoubleLong1 з четверним словом у змінній DoubleLong2:

```
mov ax, [DoubleLong1]
```

```
add [DoubleLong2], ax
```

```
mov ax, [DoubleLong1+ 2]
```

```
adc [DoubleLong2+ 2], ax
```

```
mov ax, [DoubleLong1+ 4]
```

```
adc [DoubleLong2+ 4], ax
```

```
mov ax, [DoubleLong1+ 6]
```

```
adc [DoubleLong2+ 6], ax
```

Інструкція SBB працює по тому ж принципу, що й інструкція ADC. Коли інструкція SBB виконує вирахування, у ній враховується позика, що відбулася в попереднім вирахуванні. Наприклад, наступні інструкції віднімають значення, записане в регістрах CX: BX, зі значення розміром у подвійне слово, записаного в регістрах DX: AX:

```
sub ax, bx
```

```
sbb dx, cx
```

При роботі з інструкціями ADC і SBB ви повинні переконаватися, що прапор переносу не змінився з моменту виконання останнього чи додавання вирахування, інакше стан займу / переносу, що зберігається в прапорі переносу, буде загублено. Наприклад, у наступному фрагменті програми додавання CX:BX із DX:AX виконується некоректно:

```
add ax, bx ;скласти молодші слова
```

```
sub si, si ; очистити SI (прапор
```

```
; переносу скидається в 0)
```

`adc dx, cx` ;скласти старші слова
це буде працювати некоректно, тому що з моменту останньої операції додавання вміст прапора переносу загублено.

Збільшення і зменшення

Іноді в програмі на Асемблері потрібно виконати додавання, що складається просто в додатку до операнду значення 1. Така операція називається збільшенням (інкрементацією). Аналогічно, із вмісту регістрів і змінних у пам'яті іноді потрібно відняти значення 1. Така операція називається зменшенням (декрементацією). Для виконання таких часто потребуючих дій у наборі інструкцій процесора 8086 передбачені дві інструкції - `INC` (збільшити) і `DEC` (зменшити). Наприклад, наступна програма заповнює 10-байтовий масив `TempArray` числами 0, 1, 2, 3, 4, 5, 6, 7, 8, 9:

```
TempArray DB 10DUP (?)
```

```
FillCount DW ?
```

```
mov al, 0 ; перше значення, записане в TempArray
```

```
mov bx, OFFSET TempArray ; BX вказує на TempArray
```

```
mov [FillCount], 10 ; число елементів, якими потрібно
```

заповнити масив

```
FillTempArrayLoop:
```

```
mov [bx], al ; установити поточний елемент TempArray
```

```
inc bx ; посилення на наступний елемент масиву
```

TempArray

```
inc al ; наступне записуване значення
```

```
dec [FillCount] ; зменшити лічильник числа заповнюваних
```

елементів

```
jnz FillTempArray ;обробити наступний елемент, якщо ми ще не заповнили всі елементи
```

Чому переважніше використовувати інструкцію:

`inc bx`, а не **`add bx, 1`** адже вони роблять одне і теж? Інструкція `ADD` займає 3 байти, інструкція `INC` займає тільки 1 байт і виконується швидше. Фактично, більш ошадливо виконати дві операції `INC`, чим додати до регістра розміром у слово значення 2 (збільшення і зменшення регістрів і змінних у пам'яті розміром у байт займає 2 байти, що також коротше, ніж додавання і вирахування). Інструкції `INC` і `DEC` - це найбільш ефективні інструкції, за допомогою яких

можна збільшувати і зменшувати значення перемінні пам'яті і регістрів.

Множення та ділення

Процесор 8086 може виконувати окремі типи операцій множення і ділення. Це одна із сильних сторін даного процесора, оскільки в багатьох мікропроцесорах взагалі відсутня безпосередня підтримка операцій множення і ділення, а ці операції досить складно виконати програмним шляхом.

Інструкція MUL перемножує 8- чи 16-бітові беззнакові співмножники, створюючи 16- чи 32-бітовий добуток. Спочатку розглянемо множення 8-бітових співмножників. При такому множенні один з операндів повинний зберігатися в регістрі AL, а інший може являти собою будь-який 8-бітовий загальний регістр чи змінну пам'яті відповідного розміру. Інструкція MUL завжди зберігає 16-бітовий добуток у регістрах AX. Наприклад, у фрагменті програми:

```
mov    al, 25
mov    dh, 40
mul    dh
```

AL перемножується на DH, а результат (1000) міститься в регістрі AX. В інструкції MUL потрібно вказувати тільки один операнд, інший співмножник завжди зберігатися в регістрі AL (чи в регістрі AX у випадку перемножування 16-бітових співмножників).

Інструкція перемножування 16-бітових співмножників працює аналогічно. Один зі співмножників повинний зберігатися в регістрі AX, а інший може знаходитися в 16-розрядному загальному регістрі чи змінній пам'яті. 32-бітовий добуток інструкція MUL поміщає в цьому випадку в регістри DX: AX, при цьому молодші (менш значущі) 16 бітів добутку записуються в регістр AX, а старші (більш значущі) 16 біт - у регістр DX. Наприклад, інструкції:

```
mov    ax, 1000
mul    ax
```

завантажують у регістр AX значення 1000, а потім зводять його в квадрат, поміщаючи результат (значення 1000 000) у регістри DX: AX. На відміну від додавання і вирахування, в операції множення не враховується, чи є співмножники операндами зі знаком чи без, тому мається друга інструкція множення IMUL для множення 8- і 16-бітових співмножників зі знаком. Якщо не брати до уваги, що

перемножуються значення зі знаком, інструкція IMUL працює аналогічно інструкції MUL. Наприклад, при виконанні інструкцій:

```
mov    al, -2
mov    ah, 10
imul   ah
```

у регістрі AX буде записане значення -20. Процесор 8086 дозволяє вам з певними обмеженнями розділити 32-бітове значення на 16-бітове значення і на 8-бітове. При беззнаковому діленні 16-бітового значення на 8-бітове ділене повинне бути записане в регістрі AX. 8-бітовий дільник може зберігатися в будь-якому 8-бітовому загальному регістрі чи змінній в пам'яті відповідного розміру. Інструкція DI завжди записує 8-бітову частку в регістр AL, а 8-бітовий залишок - в AH. Наприклад, у результаті виконання інструкцій:

```
mov    ax, 51
mov    dl, 10
div    dl
```

результат 5 (51/10) буде записаний у регістр AL, а залишок 1 - у регістр AH. Частка являє собою 8-бітове значення. Це означає, що результат ділення 16-бітового операнда на 8-бітовий операнд повинен перевищувати 255. Якщо частка занадто велика, то генерується переривання 0 (переривання при діленні на 0). Інструкції:

```
mov    ax, 0ffffh
mov    bl, 1
div    bl
```

генерують переривання при діленні на 0.

При діленні 32-бітового операнда на 16-бітовий операнд ділене повинне записуватися в регістр DX:AX. 16-бітовий дільник може знаходитися в кожному з 16-бітових регістрів загального призначення чи змінній пам'яті відповідного розміру. Наприклад, у результаті виконання інструкцій:

```
mov    ax, 2
mov    dx, 1 ; завантажити в DX:AX 10002h
mov    bx, 10h
div    bx
```

частка 1000h (результат ділення 10002h на 10h) буде записано в регістрі AX, а 2 (залишок від ділення) - у регістрі DX.

Як і при множенні, при діленні має значення, використовуються операнди з знаком чи без знака. Для ділення беззнакових операндів

використовується операція IDI, а для ділення операндів зі знаком - IDI. Наприклад, операції:

TestDivisor DW 100

mov ax, -667

cwd ;установити DX:AX у значення -667

idiv [TestDivisor] ;розділити значення -6 у регістрі AX і
значення -67 у регістрі DX.

4 КОНТРОЛЬНІ ЗАВДАННЯ

4.1 Загальні вказівки

У процесі вивчення дисципліни "Організація і функціонування ЕОМ" студенти заочної форми навчання повинні виконати дві контрольні роботи. Основна мета їхнього виконання – закріплення теоретичного матеріалу і розвиток навичок його практичного використання.

Тематика першої контрольної роботи охоплює питання арифметичних і логічних основ ЕОМ, друга - питання архітектури ЕОМ.

Приступати до виконання контрольних завдань потрібно після вивчення матеріалу по літературі, що рекомендується, і методичних вказівок до контрольних завдань і практичних занять. Орієнтовані витрати часу на виконання однієї роботи і її оформлення складають 8 годин.

Виконана контрольна робота повинна бути представлена на перевірку в строгій відповідності з навчальним графіком. при наявності зауважень рецензента варто виконувати усі виправлення і доповнення так, щоб при повторному рецензуванні можна було зіставляти колишнє і нове рішення.

4.2 Завдання на контрольну і методичну роботу та вказівки по їхньому виконанню

Контрольні завдання видаються в період установчої сесії і виконуються студентами самостійно згідно сьогоденням методичними вказівкам.

4.3 Методичні вказівки по оформленню контрольних робіт

Результати виконання кожної контрольної роботи оформляються у вигляді пояснювальної записки на стандартних аркушах паперу формату А4 чи в окремих учнівських зошитах у клітку. Текстова частина записки повинна бути написана чорнилом (пастою) чітко і розбірливо.

Структура пояснювальної записки: титульний лист, введення, основна частина, висновок, список використаних джерел.

На титульному листі повинні бути зазначене: повне найменування міністерства, ВУЗА, кафедри, прізвище і ініціали студента, його навчальна група, шифр і домашня адреса.

У вступі вказується: ціль роботи, її зміст і вихідні дані.

Основна частина включає результати виконання всіх етапів роботи відповідно до її змісту. Кожен етап виконання повинний мати як текстові, так і ілюстративні пояснення. Текст, малюнки і схеми повинні бути оформлені акуратно, з урахуванням вимог діючих стандартів.

Висновок повинний містити короткі висновки про результати роботи.

Робота повинна бути датована і підписана виконавцем.

4.4 Контрольна робота №1

1. Зарезервувати місце в пам'яті в сегменті даних DSeg_1 для масиву байт з 10 елементів Masb.

2. Зарезервувати місце в пам'яті в іншому сегменті даних DSeg_2 для масиву слів з 30 елементів з ім'ям Masw.

3. Зарезервувати місце в додатковому сегменті ESeg для таблиці байтів, що складається з 10 записів по 4 елементи Tablb.

4. Зарезервувати місце в додатковому сегменті ESeg для таблиці подвійних слів з ім'ям TablAdr 40 елементів.

5. Використовуючи пряму адресацію з індексуванням заповнити масив Masb.

6. Використовуючи базову адресацію зі зсувом заповнити Mas.

7. Використовуючи базову адресацію з індексуванням заповнити TablB у такий спосіб:

як уміст 1 -го елемента кожного запису береться елемент Masb з тим же номером, що і номер запису. Як вміст інших полів береться перший байт елементів Masw один за одним.

8. Використовуючи поперемінно, якщо це можливо, або команду завантаження абсолютної адреси, або псевдооператори SEG, OFFSET, заповнити таблицю TablAdr так, щоб кожен елемент таблиці являв собою адресу відповідного елемента таблиці Tablb.

9. Використовуючи операції зі стеком на початку залишаємо в Tablb порядок елементів у записах незмінним, поміняти в зворотному порядку розташування записів, і на зворотний порядок елементів.

4.5 Контрольна робота №2

1. Помістити в сегменти (DATA 1, DATA2) даних два тексти відповідно.
2. Очистити екран за допомогою строкових команд.
3. Вивести на екран обидва тексти за допомогою строкових команд.
4. У 1- ом тексті підрахувати кількість слів, що починаються з букви "м".
5. В другому тексті підрахувати кількість слів, що починаються з букви "н".
6. Вивести слова яких більше.
7. Знайти і вивести однакові слова.
8. Замінити всі крапки на знак "Г".
9. Вивести на екран обидва тексти.

Список літератури, що рекомендується Основна

1. Обчислювальні комплекси, системи і мережі / А. М.Ларіонов, С. А. Майорів, Г. И. Новиков: Підручник для вузів. Л: Энерговидат. Ленингр. відділення, 1987.
2. Системи рівнобіжної обробки: Пер. с англ./ Під ред. Д. Ивенса -М.: Світ, 1985.
3. Комп'ютери на СБИС: у 2-х кн.: Пер. с япон. / Моотока Т., Томита С., Танака Х. і ін. -М.: Світ, 1988.
4. Мови програмування і схемотехніка СБИС: Пер. с япон. / Фути К., Судзуки Н. -М.: Світ, 1988.
5. Цифрова обробка інформації на основі швидкодіючих БІС / Гамкрелідзе С. А., Зав'ялов В. А. і ін. -М.: Энергоатомвидат, 1988.
6. Мікроаналіз продуктивності обчислювальних систем / Байцер Б. -М.: Радіо і зв'язок, 1983.

Додаткова

1. Рівнобіжні обчислювальні системи / Головкин Б. А. -М.: Наука, 1980.
2. Оцінка продуктивності обчислювальних систем / Ферради Д. - М.: Світ, 1981.
3. Інженерний аналіз функціонування обчислювальних машин і систем /Хорошевский В. Г. -М.: Радіо і зв'язок, 1987.
4. СБИС для розпізнавання образів і обробки зображень / -М.: Світ, 1988.