**Лабораторна робота 3. Способи адресації команд і даних**

3.1. Мета роботи

3.2. Принципи адресації команд і даних

## 3.2.1. Безпосередня адресація

## 3.2.2. Пряма адресація

## 3.2.3. Пряма регістрова адресація

## 3.2.4. Непряма адресація

## 3.2.5. Непряма регістрова адресація

## 3.2.6. Базова адресація

## 3.2.7. Індексна адресація

## 3.2.8. Автоінкрементна і автодекрементна адресація

## 3.2.9. Адресація стекової пам'яті

3.3. Порядок виконання роботи

3.4. Вміст звіту

3.5. Контрольні питання і завдання

## 3.1. Мета роботи

Вивчення методів адресації команд і даних в комп'ютерах, принципів логічної організації апаратних засобів процесора, що забезпечують доступ до пам'яті.

## 3.2. Принципи адресації команд і даних

У комп'ютерах, процесори яких оснащені регістрами загального призначення, метод адресації об'єктів, якими маніпулює команда, може задавати константу, регістр або елемент пам'яті. Для звернення до комірки пам'яті процесор повинен, перш за все, обчислити ефективну або виконавчу адресу пам'яті Ав, яка визначається використовуваним в команді методом адресації. Виконавча адреса – це номер комірки пам'яті, до якої проводиться фактичне звернення.

При розгляді методів адресації слід виділити поняття неявного операнда (що мається на увазі), і неявної адреси (що мається на увазі).

Поняття неявного операнда передбачає, що команда не містить в явному вигляді адреси операнда; операнд мається на увазі і фактично визначається кодом операції. Даний спосіб має кілька важливих випадків застосування. Як приклад можна привести команди підрахунку, в якій до деякого числа, що знаходиться в одному з регістрів, додається фіксоване прирощення. В результаті, один з операндів команди (номер регістра) адресується явним методом, а другий операнд – прирощення – адресується неявно, і ймовірно, в пам'яті комп'ютера не міститься.

Поняття неявної адреси передбачає, що в команді не міститься явних вказівок про адресу, яка бере участь в операції операнда, або адреси, за якою записується результат операції. У той же час ця адреса мається на увазі. Прикладом може служити команда, яка містить адреси обох операндів, що беруть участь в операції, при цьому мається на увазі, що результат операції буде поміщений на адресу одного з операндів. Іншим прикладом може служити команда, в якій вказана адреса тільки одного операнда, а адреса другого, яким є вміст спеціального регістра, мається на увазі.

У таблиці 3.1 на прикладі команди додавання (Add) наведені найбільш уживані назви методів адресації операндів, які реалізовані в комп'ютерах загального призначення.

Таблиця 3.1 – Основні методи адресації операндів

|  |  |  |  |
| --- | --- | --- | --- |
| Метод адресації | Приклад команди | Опис команди | Використання |
| Безпосередня | Add R4, **#3** | R4 := R4 + 3 | Для завдання констант |
| Пряма | Add R1, **(1000)** | R1 := R1 + M[1000] | Застосовується для звернення до статичних даних |
| Пряма регістрова | Add **R4**, **R5** | R4 := R4 + R5 | Обидва операнди і результат перебувають в регістрах |
| Непряма регістрова | Add R4, **(R1)** | R4 := R4 + M[R1] | Для звернення за вказівником або обчисленою адресою |
| Базова (відносна) | Add R4, **(R1), D** | R4 := R4 + M[R1 + D] | Перший операнд – в регістрі, другий в пам'яті |
| Індексна | Add R3, **(R1+R2), D** | R3 := R3 + M[R1 + R2 + D] | Застосовується при роботі з масивами: R1 – база, R2 – індекс |
| Автоінкрементна | Add R1, **(R2)+** | R1 := R1 + M[R2]; R2 := R2 + d | Використовується для проходу в циклі по масиву з кроком. R2 – початок масиву. В кожному циклі R2 бере прирощення d |
| Автодекрементна | Add R1, **(R2)-** | R1 := R1 + M[R2]; R2 := R2 - d | Аналогічна попередній. Обидві можуть використовуватися для реалізації стека |
| Безадресна |  |  | Безадресне завдання операндів |

Зазвичай в комп'ютерах одночасно використовується кілька типів адресації. У команді тип адресації може задаватися або неявно кодом операції, або в явній формі в спеціальному полі адресної частини команди.

## 3.2.1. Безпосередня адресація

При безпосередній, або літеральній, адресації в команді міститься не адреса операнда, а сам операнд. При виконанні такої команди не потрібно звернення до пам'яті для вибірки операнда. Безпосередня адресація використовується при виконанні арифметичних операцій, операцій порівняння, а також для завантаження констант в регістри. Застосування безпосередньої адресації дозволяє значно зменшити час виконання програм, а також займаного ними об'єму пам'яті. З огляду на те, що частота використання команд з безпосередньою адресацією в різних програмах коливається від 17% до 43%, можна зробити висновок про ефективність такого методу адресації.

## 3.2.2. Пряма адресація

При прямій, або абсолютній, адресації виконавча адреса збігається з адресною частиною команди:

MAR := 1000; MBR := M[MAR].

Абсолютна адресація була загальноприйнятою в перших обчислювальних машинах і продовжує застосовуватися в даний час в поєднанні з іншими методами.

## 3.2.3. Пряма регістрова адресація

Застосування прямої регістрової адресації передбачає, що виконавча адреса збігається з адресною частиною команди, яка містить посилання на регістр ЦП. Цей спосіб адресації був загальноприйнятим в машинах перших поколінь і продовжує застосовуватися в сучасних комп'ютерах в комбінації з іншими способами.

## 3.2.4. Непряма адресація

При непрямій адресації адресне поле команди вказує адресу комірки пам'яті, в якій знаходиться адреса операнда або команди. Таким чином, непряма адресація може бути визначена як "адресація адреси". Наявність непрямої адресації в команді може визначатися кодом операції. Крім того, в деяких машинах в команді відводиться спеціальний розряд (покажчик адресації – ПА) і цифра 0 або 1 в ньому вказує, що адресна частина команди є прямою адресою або непрямою. Процедуру звернення до оперативної пам'яті за операндом при непрямій адресації можна описати таким чином:

MAR := IR[D];

зчитування: MBR := M[MAR];

якщо ПА = 0, то йти до А, інакше MAR := [MBR];

зчитування: MBR := M[MAR];

А: ALU := [MBR].

В цьому випадку в регістр адреси оперативної пам'яті заноситься вміст поля D команди. Виконується процедура зчитування вмісту комірки D і в буферному регістрі ОП розміщується один з операндів. Далі перевіряється вміст покажчика адреси. Якщо ПА дорівнює 0 (пряма адресація), відбувається перехід до мітки А, і в регістр АЛП передається код операнда з буферного регістра оперативної пам'яті. Якщо ж поле ПА містить 1 (непряма адресація), то вміст буферного регістра пам'яті завантажується в адресний регістр пам'яті. Відбувається повторне зчитування з оперативної пам'яті (але з іншої комірки) операнда, який передається в регістр АЛП.

## 3.2.5. Непряма регістрова адресація

На рисунку 3.1 показана логічна організація пристроїв, що виконують команду передачі числа 1234 з Рг5 в комірку оперативної пам'яті 5432 з використанням непрямої адресації. Адреса комірки знаходиться в Рг3. При цьому необхідно враховувати, що довжина адресних полів команди достатня лише для розміщення тільки номерів регістрів, тобто не дозволяє поставити в команді повну адресу комірки пам'яті.

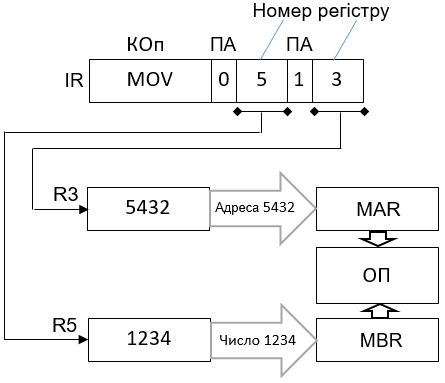


Рисунок 3.1 – Приклад реалізації непрямої адресації

## 3.2.6. Базова адресація

При базовій, або відносній, адресації (адресації з базуванням) виконавча адреса визначається сумою базової адреси Аб і зміщення D. Для зберігання базових адрес в процесорах передбачені спеціальні регістри. У прикладі, наведеному в таблиці 3.1, для зберігання базової адреси використовується регістр R1. Крім того, в адресній частині команди виділяється поле, в якому розміщується зміщення D. Виконавча адреса формується: Ав = Аб + D.

Застосування в комп'ютерах базової адресації дозволяє при незначній довжині адресного коду команди забезпечити доступ до будь-якої комірки пам'яті. Кількість розрядів регістра зберігання базової адреси Аб вибирається такою, щоб можна було ставити початок масиву у всьому адресному просторі оперативної пам'яті, а адресний код зміщення D визначає положення операнда відносно початку масиву. Процес формування виконавчої адреси при базовій адресації показаний на рисунку 3.2. В цьому випадку виконавча адреса формується на виході суматора відповідно до виразу:

Ав= [B] + D,

де B і D – коди адресної частини, які стоять у відповідних полях команди; [В] – вміст регістра з номером В. У тому випадку, якщо В = 0, виконавча адреса дорівнює D.

Формально звернення до пам'яті при базовій адресації можна представити у вигляді такої послідовності процедур:

якщо IR[В] = 0, то MAR := IR[D];

інакше MAR := [RB] + IR[D];

зчитування: MBR := M[MAR].

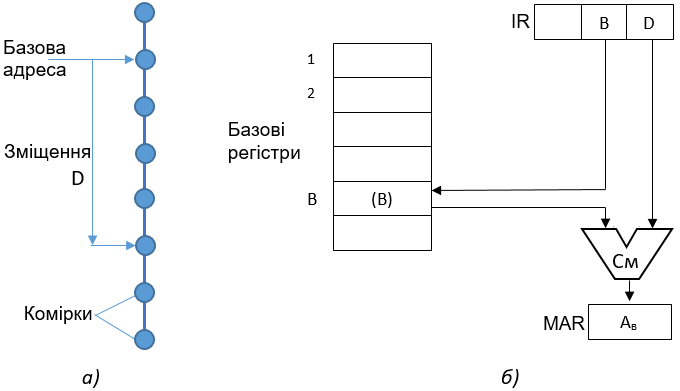


Рисунок 3.2 – Базова адресація; а) формування адреси елемента одновимірного масиву; б) формування виконавчої адреси

Формування виконавчої адреси при базовій адресації шляхом підсумовування пов'язано з втратою часу. Тому іноді застосовується процедура отримання виконавчої адреси, що отримала назву операції конкатенації (суміщення) слова:

MAR := [RB] \ IR[D],

де \ – символ операції конкатенації.

На рисунку 3.3 показана процедура отримання виконавчої адреси на основі операції конкатенації. З огляду на те, що базова адресація для різних програм становить від 32% до 55%, можна зробити висновок, що застосування операції суміщення дозволяє підвищити швидкість формування адреси одного з операндів і продуктивність процесора в цілому.

Базова адресація забезпечує так звану переміщуваність програм в пам'яті комп'ютера, тобто розміщення програм в вільних ділянках пам'яті без внесення змін всередині самої програми.

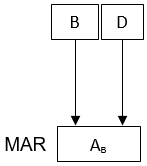


Рисунок 3.3 – Формування виконавчої адреси методом конкатенації

## 3.2.7. Індексна адресація

Індексна адресація є розвитком методу базової адресації, з яким вона часто використовується спільно. Для виконання індексації в машину вводяться спеціальні (індексні) регістри. У форматі команди виділяється поле Х для вказівки номера індексного регістра (в прикладі, наведеному в таблиці 3.1, як індексний використовується регістр R2). Виконавча адреса операнда при індексації формується шляхом підсумовування вмісту базового та індексного регістрів і зміщення D (рис. 3.4).

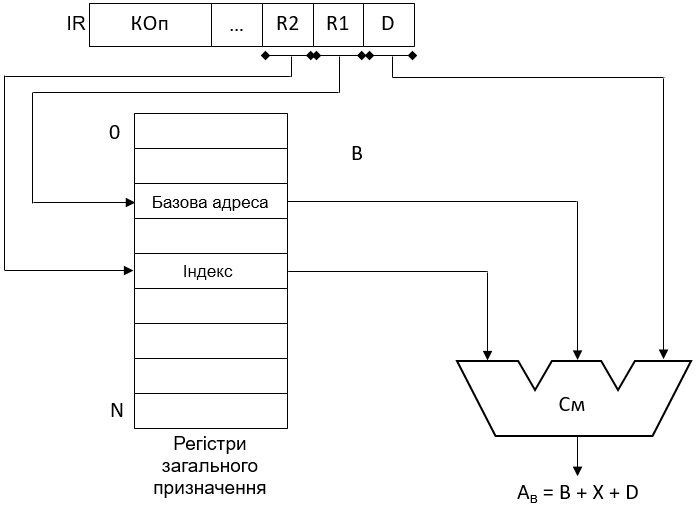


Рисунок 3.4 – Формування виконавчої адреси при індексації

## 3.2.8. Автоінкрементна і автодекрементна адресація

Автоінкрементна і автодекрементна адресації мають механізм автоматичного збільшення чи зменшення вмісту певного регістра при кожному зверненні до нього. Такий тип адресації особливо ефективний при обробці масивів даних.

При автоінкрементній адресації спочатку вміст регістра використовується як адреса операнда, а потім змінюється шляхом додавання константи d, що дорівнює числу байт в елементі масиву. При автодекрементній адресації спочатку вміст зазначеного в команді регістра зменшується на число d, яке дорівнює кількості байт в елементі масиву, а потім використовується як адреса операнда.

Автоінкрементна і автодекрементна адресації є спрощеним варіантом індексації і важливим механізмом перетворення адресних частин команд при організації обчислювальних циклів.

## 3.2.9. Адресація стекової пам'яті

Безадресне завдання операндів реалізується за допомогою стекової пам'яті, яка є ефективним елементом сучасної архітектури комп'ютерів, починаючи персональним комп'ютером і закінчуючи мейнфреймом.

Стек являє собою сховище даних, робота з якими ведеться за наступним принципом: елемент, записаний в стек останнім, зчитується з нього першим. Застосовуються два види реалізації стекової пам'яті:

* апаратний стек, що представляє собою групу послідовно пронумерованих регістрів;
* група комірок ОП, забезпечених покажчиком стека.

Найпоширенішою є така організація стека, при якій він заповнюється зверху донизу: перший елемент записується в самий кінець області стека (в комірку з найбільшою адресою), наступний елемент записується під ним тощо. При читанні інформації зі стека першим витягується слово, яке надійшло до нього останнім.

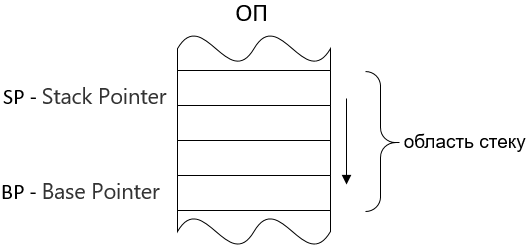


Рисунок 3.5 – Стекова пам'ять

Таким чином, виходить, що низ стека завжди фіксований (це остання комірка області стека), а вершина стека весь час переміщується. Для завдання поточного становища вершини стека використовується регістр SP (Stack Pointer – покажчик стека). У ньому зберігається адреса тієї комірки, в якій знаходиться елемент, записаний в стек останнім. Область пам'яті, що виділяється під стек, називається сегментом стека.

Початок області стеку вказується у виділеному регістрі BP (Base Pointer). У покажчику стека SP встановлюється зміщення вершини стека, відлічене від початку області стека. Тому абсолютна адреса вершини стека задається парою регістрів BP і SP (рисунок 3.5).

У стеці автоматично реалізується правило "останній прийшов – перший пішов", тому при операціях зі стеком можливо безадресне завдання операнда. Команда не містить адреси комірки стека, але повинна містити адресу комірки пам'яті чи регістру, звідки слово передається в стек або куди поміщається зі стека.

В архітектурі сучасних комп'ютерів стекова адресація широко використовується при обробці підпрограм і в системах переривань.

## 3.2.9. Особливості формування адреси в процесорах Intel

Пам'ять для процесорів Intel розбивається на байти (8 біт), слова (16 біт), подвійні слова (32 біт). Більшими одиницями є сторінки і сегменти. Пам'ять може логічно організовуватися у вигляді одного або множини сегментів змінної довжини. Метод адресації даних в команді може встановлювати константу, регістр або елемент пам'яті. Стосовно до пам'яті розрізняють три адресних простори: логічний, лінійний і фізичний.

***Логічна***або ***віртуальна адреса*** (рисунок 3.6) складається з селектора і зсуву. При зверненні до комірки пам'яті процесор спочатку повинен обчислити ***ефективну (виконавчу) адресу пам'яті***, яка задається обраним в команді методом адресації. Ефективна адреса формується підсумовуванням (в різних поєднаннях) наступних компонентів:

**Base (база)** – вміст одного з універсальних регістрів. Базові регістри зазвичай використовуються транслятором як початок позначки локальної області пам'яті;

**Index (індекс)** – вміст одного з універсальних регістрів. Індексні регістри зазвичай використовуються для доступу до елементів масиву або рядку;

**Scale (масштабний коефіцієнт)** – значення індексного регістра може бути помножене на коефіцієнт 1, 2, 4 або 8, використання якого визначено в спеціальному байті поля команди;

**Disp (displacement – зміщення в команді)** – 8- або 32-розрядне значення, безпосередньо задається в команді.

*Лінійна адреса* складається з двох компонент – базової адреси сегмента і ефективної адреси:

**LA = BAS + EA**.

У загальному випадку виконавча адреса EA операнда визначається так:

**ЕА = Base + (Index \* Scale) + Disp**.

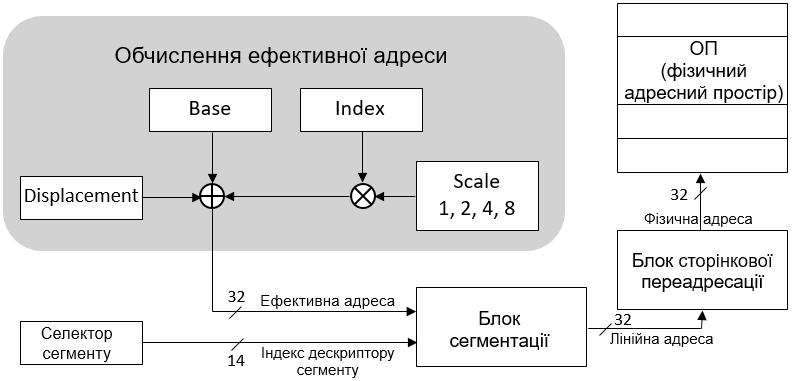


Рисунок 3.6 – Формування адреси 32-розрядних процесорів

Як приклад в розділі "Схема адресації" розглянуті методи адресації, використовувані в процесорі 80386. При цьому опущені два режими адресації: один забезпечує звернення до вмісту будь-якого регістра процесора, інший задає операнд безпосередньо в команді. Операнди можуть мати розмір 8, 16 або 32 біта.

## 3.3. Порядок виконання роботи

Перша частина. Студенти повинні:

1. Уважно прочитати теоретичну частину роботи.
2. Ознайомитися з методами адресації даних в комп'ютерах,
3. З'ясувати причини використання таких понять, як неявний операнд і неявна адреса.

Друга частина.

2.1. Завантажити або застосунок **PC.exe**, або **MMIkurs.exe**.

2.2. Для трьох заданих (відповідно до індивідуального варіанта) режимів адресації:

* переписати вирази для обчислення лінійної адреси;
* навести приклади команд;
* зобразити відповідні схеми формування LA.

2.3. За допомогою навчальної програми (PC.exe або MMIkurs.exe) перейти в розділ "Моделювання" і виконати наступні процедури:

* вибрати розмір даних BYTE; розмір сегмента – 96 байт;
* базову адресу сегмента розрахувати за формулою:

(№ варіанту mod 6 + 1) \* 16;

* ефективну адресу операнду:

EA = (N\_вар mod 5 + 1) \* 14.

Значення бази, індексу, зміщення: *довільні ненульові, підбираються відповідно до умов, наведених вище.*

* Відповідно до отриманих значеннями наберіть числові дані у вікнах "Адреса сегмента", "Розмір сегмента", "Зсув" тощо;

2.4. Побудуйте таблицю для всіх 9 методів адресації (за прикладом табл. 3.3).

Таблиця 3.2 – Варіанти завдань

| №№ варіантів | Методи адресації | | |
| --- | --- | --- | --- |
|  | Index | Base index with offset | Base index with scaling |
|  | Direct | Base index with scaling | Base index with offset and scaling |
|  | Base index | Register indirect | Base index with offset |
|  | Base index with offset | Direct | Base index with scaling |
|  | Index | Direct | Base index with offset and scaling |
|  | Base index | Index | Index with scaling |
|  | Base index with scaling | Base index with offset and scaling | Base index with offset |
|  | Index with scaling | Direct | Base index with offset and scaling |
|  | Direct | Base index with scaling | Base index |
|  | Register indirect | Base index with offset | Base |
|  | Index | Index with scaling | Base |
|  | Base index with offset | Index with scaling | Base |
|  | Register indirect | Base | Direct |
|  | Base | Direct | Base index with scaling |
|  | Base index | Register indirect | Index |
|  | Register indirect | Base index with offset | Direct |
|  | Base index with scaling | Base index with offset and scaling | Register indirect |
|  | Register indirect | Index | Direct |
|  | Index | Base index with offset | Base |
|  | Base index | Index with scaling | Direct |
|  | Register indirect | Index | Base index with offset |
|  | Base | Direct | Index |
|  | Index | Base | Base index with offset and scaling |
|  | Base index | Register indirect | Direct |
|  | Register indirect | Base index with scaling | Base index with offset and scaling |
|  | Index | Index with scaling | Base index with offset and scaling |
|  | Base index with offset | Index with scaling | Direct |
|  | Base index | Base index with offset | Index |
|  | Index | Base | Direct |
|  | Index with scaling | Base index with scaling | Base index with offset and scaling |
|  | Register indirect | Direct | Base index with offset and scaling |
|  | Direct | Base index with scaling | Index |
|  | Register indirect | Base index with offset | Index with scaling |
|  | Base index | Register indirect | Base index with scaling |
|  | Index with scaling | Direct | Base index with scaling |
|  | Index | Base index with offset | Direct |
|  | Register indirect | Index | Base |
|  | Direct | Base index with scaling | Index with scaling |
|  | Register indirect | Base index with offset | Direct |
|  | Base index | Index with scaling | Base index with offset and scaling |
|  | Register indirect | Direct | Base index with scaling |
|  | Register indirect | Index with scaling | Base index with scaling |
|  | Index with scaling | Base | Base index with scaling |
|  | Base index | Direct | Base index with scaling |
|  | Index | Index with scaling | Base index with scaling |
|  | Register indirect | Base | Base index with offset and scaling |
|  | Index | Base index with offset | Index with scaling |
|  | Register indirect | Index | Base index with offset |
|  | Base index with offset | Direct | Base index with offset and scaling |
|  | Base index | Register indirect | Base index with offset and scaling |
|  | Register indirect | Index with scaling | Direct |
|  | Base index | Direct | Base index with offset and scaling |
|  | Register indirect | Index with scaling | Base index with offset and scaling |
|  | Index | Base | Base index with scaling |
|  | Register indirect | Base | Base index with scaling |
|  | Base index | Index | Direct |
|  | Base | Direct | Register indirect |
|  | Base index | Base index with offset | Base index with offset and scaling |
|  | Index | Index with scaling | Index |
|  | Base index with offset | Index with scaling | Base index with scaling |
|  | Base | Direct | Base index |
|  | Base index | Base | Base index with scaling |
|  | Base index with offset | Base | Direct |
|  | Register indirect | Index | Index with scaling |
|  | Index | Base index with scaling | Base index with offset and scaling |
|  | Base index | Base index with offset | Base |
|  | Base index with offset | Index with scaling | Base |
|  | Base index | Register indirect | Index with scaling |
|  | Base index with scaling | Base index with offset and scaling | Base index |
|  | Direct | Base index with offset | Register indirect |
|  | Base index with scaling | Base index with offset and scaling | Index with scaling |
|  | Base index | Base index with offset | Base index with scaling |
|  | Base index with offset | Index with scaling | Base index with offset and scaling |
|  | Base index | Index | Base index with offset and scaling |
|  | Direct | Base index with scaling | Register indirect |
|  | Base index | Index | Base index with scaling |
|  | Base index with offset | Base | Base index |
|  | Base index with scaling | Base index with offset and scaling | Base |
|  | Base index | Register indirect | Base |
|  | Base index with scaling | Base index with offset and scaling | Index |
|  | Register indirect | Base index with offset | Base index |
|  | Base index | Index | Base index with scaling |
|  | Register indirect | Base index with offset | Base index with offset and scaling |
|  | Base index with scaling | Base index with offset and scaling | Direct |
|  | Base index | Base | Base index with offset and scaling |
|  | Register indirect | Index | Base index with offset and scaling |
|  | Base | Direct | Base index |
|  | Base index | Index | Base index with offset |
|  | Index with scaling | Base index with scaling | Base |
|  | Base index with offset | Base | Base index with offset and scaling |
|  | Base index | Index with scaling | Base |
|  | Index | Base index with offset | Base index with offset and scaling |
|  | Base index | Index | Base |
|  | Direct | Base index with scaling | Base index with offset |
|  | Index with scaling | Base | Base index with offset and scaling |
|  | Base index | Base index with scaling | Base index with offset and scaling |
|  | Base index | Index with scaling | Base index with scaling |
|  | Base index with offset | Base index with scaling | Base index with offset and scaling |
|  | Register indirect | Index with scaling | Base |
|  | Index with scaling | Base | Direct |

## 3.4. Вміст звіту

Звіт по роботі повинен містити:

1. **Мета роботи**
2. **Постановка індивідуального завдання**
3. **Теоретичні відомості:**
   1. Використані в комп'ютерах способи адресації команд і даних
   2. Схеми формування LA відповідно до індивідуального варіанту завдання
4. **Експериментально-практичні результати дослідження**

Експериментально-практичні результати обчислення LA в табличному вигляді (табл. 3.3) для всіх 9 методів адресації та відповідні скріншоти, що підтверджують наведені дані.

1. **Аналіз результатів і висновки по роботі.**

Таблиця 3.3 – Таблиця формування лінійної адреси

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Методи адресації | Базова адреса сегмента | Розмір сегмента | Базова адреса | Індекс | Масштаб | Зсув | Лінійна адреса |
| Base index | 2 | 50 | 10 | 5 | 1 | 20 |  |
| Base index with offset | 2 | 50 | 10 | 5 |  | 20 |  |
| Base index with offset and scaling |  |  |  |  |  |  |  |
| Base index with scaling |  |  |  |  |  |  |  |
| Base |  |  |  |  |  |  |  |
| Direct |  |  |  |  |  |  |  |
| Index |  |  |  |  |  |  |  |
| Index with scaling |  |  |  |  |  |  |  |
| Register indirect |  |  |  |  |  |  |  |

***Примітка:*** *в тому випадку, коли у формулі формування лінійної адреси бере участь параметр масштабування (Scale) для кожного розміру операнда необхідно отримати додаткові значення LA для значень параметра Масштаб 1, 2, 4 і 8.*

3.5. Контрольні питання і завдання

1. Чому в сучасних комп'ютерах використовується кілька методів адресації?
2. З якою метою в комп'ютерах застосовуються поняття неявної адреси і неявного операнда?
3. З якою метою в комп'ютерах застосовуються стеки?
4. Наведіть приклади структур одно-, дво- і трьохадресних команд.
5. В яких випадках в комп'ютерах застосовується автоінкрементна і автодекрементна адресація?
6. Яким чином формується виконавча адреса при непрямій адресації?
7. Наведіть приклад використання відносної адресації.
8. Чим відрізняються поняття структура команди і формат команди?