**КПІ ім. Ігоря Сікорського**

**Інститут прикладного системного аналізу**

**Кафедра Системного проектування**

Лабораторна рoбота №4

«Дослідження адресації даних в «реальному режимі»»

Виконав:

Студент групи ДА-92

ННК «ІПСА»

Насікан Дмитро Юрійович

Варіант № 11

Київ – 2021 рік

**ЗАВДАННЯ**

1. Скласти схему обчислення адресу процесорів архітектури х86.

2. Пояснити призначення регістрів ЦПУ, використовуваних для адресації.

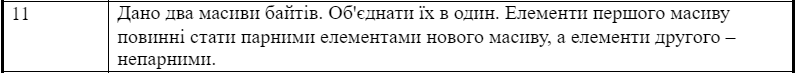
3. Визначити склад і пояснити призначення засобів обчислення адресу даних.

4. Перерахувати способи адресації даних і пояснити їх особливості.

5. Скласти схеми обчислення адресу для різних способів адресації даних.

6. Написати і відлагодити програму, що виконує дії, вказані в таблиці. Для адресації елементів масиву використовувати непрямі режими адресації. Програма має містити процедуру з передачею параметрів через стек. Навести приклади використання різних способів адресації даних і команд в програмі, зафіксувавши вміст відповідних регістрів і адрес пам'яті.

7. Написати програму мовою С або С++, яка виконує ті самі дії. Передивитись сгенерований компілятором код в режимі дебагеру як дезасемблований. Порівняти з власною реалізацією.



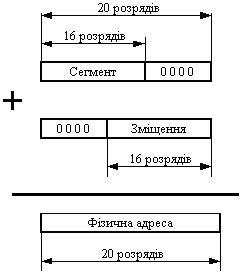
**ХІД РОБОТИ**

1. Складемо схему обчислення адресу процесорів архітектури х86:

Фізична адреса передається з процесора в пам'ять по шині адреси, ширина якої визначає максимальний обсяг фізичної пам'яті, що безпосередньо адресується процесором. У реальному режимі при наявності 20-розрядної шини адреси, процесор може адресуватися до 216 байт, або 1 мегабайту пам’яті.

Логічна адреса пам’яті складається з двох 16-розрядних компонент: компоненти сегмента пам’яті та зміщення відносно сегмента.

При побудові фізичної адреси з логічної, компоненту сегмента зсувають вліво на 4 розряди, а компоненту зміщення вправо на 4. Потім їх додають.



Важливо підмітити, що одній фізичній адресі може відповідати декілька логічних.

Межі для логічних та фізичних адрес:

0000h:0000h <= [логічна адреса] <= FFFFh:000Fh

00000h <= [фізична адреса] <= FFFFFh

1. Пояснити призначення регістрів ЦПУ, використовуваних для адресації:

Сегментова компонента може зберігатися в одному з наступних регістрів:

**CS – Code Segment -** вказує на початок блоку пам'яті об'ємом 64К, або сегмент коду, в якому знаходиться наступна інструкція.

**DS – Data Segment** - вказує на початок сегмента даних, який являє собою блок пам'яті об'ємом 64К, в якому знаходиться більшість розміщених в пам'яті операндів.

**ES – Extra Segment** - вказує на початок блоку пам'яті об'ємом 64К, який називається додатковим сегментом. Як зрозуміло з його назви, додатковий сегмент не служить для якоїсь конкретної мети, але доступний тоді, коли в ньому виникає необхідність. Іноді додатковий сегмент використовується для виділення додаткового блоку пам'яті об'ємом 64К для даних.

**SS – Stack Segment** - вказує на початок сегмента стека, який являє собою блок пам'яті об'ємом 64К, в якому знаходиться стек.

Компонента зміщення може знаходитися у одному з наступних регістрів:

**IP - Instruction Pointer** - зберігає адресу зміщення наступної команди, яку потрібно виконати. IP у поєднанні з регістром CS (як CS:IP) дає повну адресу поточної інструкції в сегменті коду.

**SP – Stack Pointer** – зберігає значення зміщення програмного стека. SP у поєднанні з регістром SS (SS:SP) означає поточну позицію даних або адреси в програмному стеку.

**BP – Base Pointer** – допомагає посилатися на змінні параметрів, передані підпрограмі. Адреса в регістрі SS поєднується зі зміщенням в BP, щоб отримати розташування параметра. BP також можна поєднувати з DI та SI як базовий регістр для спеціальної адресації.

**SI (Source Index), DI (Destination Index) –** індексні регістри, у яких зберігаються адреси операндів під час виконання програми.

**BX – Base Pointer** - базовий регістр в обчисленнях адреси, часто вказує на початкову адресу (званий базою) структури в пам'яті.

Типи адресації:

**Безпосередня адресація**

Операндом виступає саме значення, над яким одразу виконується операція. Процесор не звертається до пам’яті, тому цей тип адресації ще називають моментальним.

Приклад: ADD BYTE\_VALUE, 65

**Регістрова адресація**

Операнд знаходиться у регістрі.

Пряма – значення операнда знаходиться у регістрі.

Непряма – адреса на значення операнда знаходиться у регістрі.

Приклад: АDD EX, BX;

**Пряма адресація пам’яті**

Абсолютну адресу операнда задають у вигляді СЕГМЕНТ:ЗМІЩЕННЯ.

Приклад: MOV AX, SS:0037h

**Непряма адресація**

Адресу операнда знаходиться в:

Регістрова – адреса знаходиться у регістрах SI, DI, BX, BP.

Пам’яті – адреса знаходиться у комірці пам’яті.

Зрозуміло, що такий тип адресації є не ефективним.

Приклад:

MOV AX, [SI]

**Відносна адресація**

Адреса визначається як значення операнда + значення Instruction Pointer.

**Базова адресація**

Адреса визначається як сума значення базового регістра та зміщення.

Якщо базовий регістр – BP, то адреса операнда відраховується відносно сегмента, на який вказує SS. Якщо BX, то – DS.

Приклад:

MOV AX, [BX+2] ; береться слово, що знаходиться у сегменті, на який вказує DS, зі зміщенням BX + 2.

**Індексна адресація**

Адреса операнда обраховується як сума зміщення, яке задається в інструкції, та значення SI або DI.

Приклад:

MOV AX, [DI+05]

**Базово індексна адресація**

Зміщення операнда обраховується, як сума одного з базових регістрів (BP або BX) та індексних регістрів (SI або DI).

Приклад:

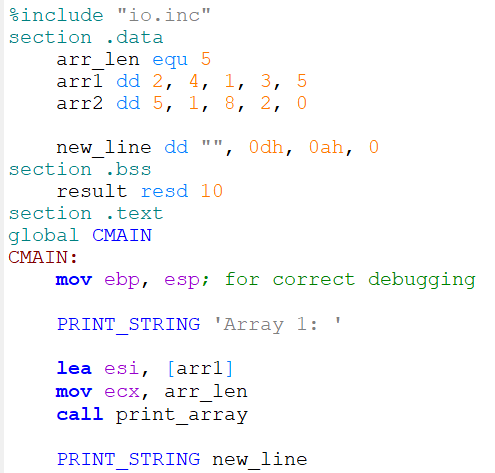
ADD AX, [BX+SI]

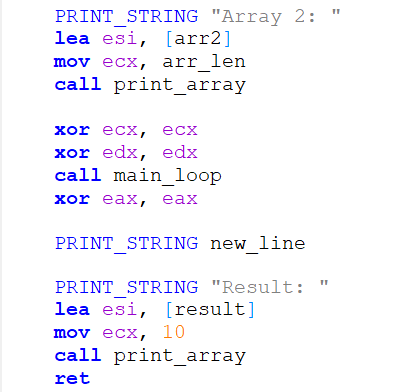
Якщо використовується базовий регістр – BP, то адреса операнда відраховується відносно сегмента, на який вказує SS. Якщо BX, то – DS.

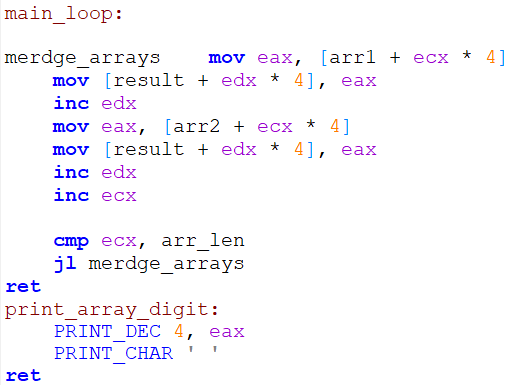
Загальна схема адресації:

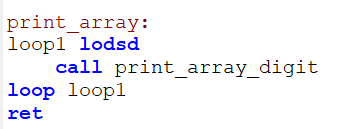
Segment(в одному з сегментних регістрів):база + індекс + зміщення.

1. Напишемо програму, що об’єднує два масиви байтів в один, причому елементи першого масиву стають парними елементами нового масиву, а другого – непарними:

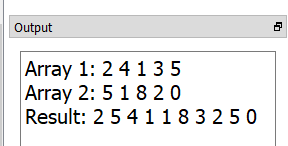




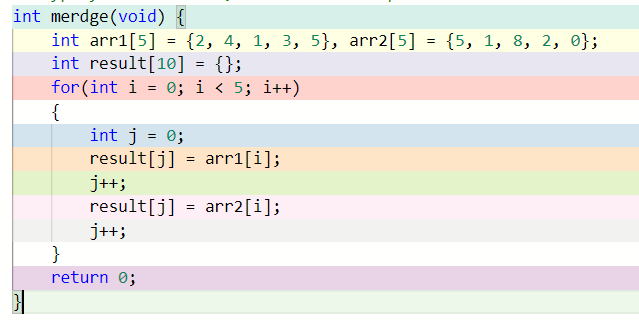




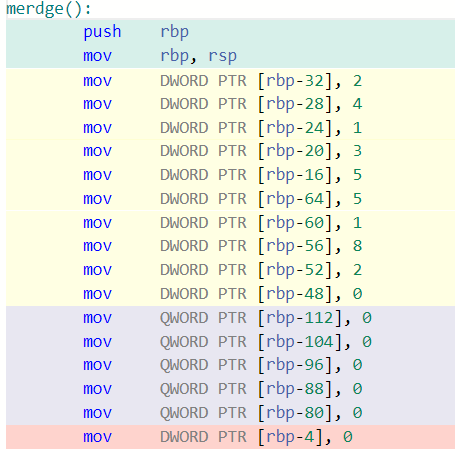
Як бачимо, при виконанні, програма об’єднує 2 масиви, причому елементи першого масиву стають парними елементами нового масиву, а другого – непарними.

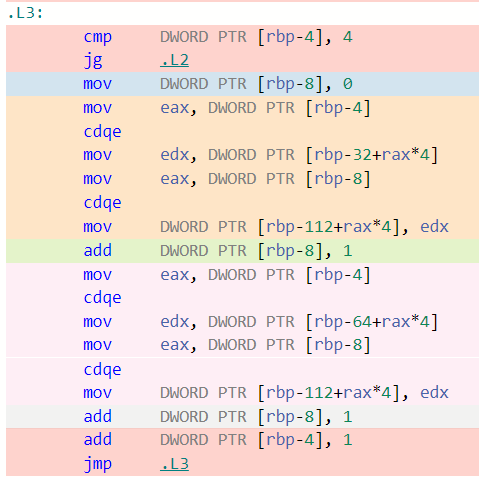


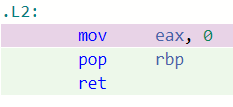
Складемо програму на мові С++, що виконує ті самі дії, та дизасемблюємо її:



Дизасембльований код:







**ВИСНОВКИ**

У ході цієї лабораторної роботи було досліджено механізми адресації пам’яті у реальному режимі. Спочатку було сконструйовано схему обчислення фізичної адреси, потім пояснене призначення регістрів, що використовуються для адресації. Далі, був наведений опис основних способів адресації даних з прикладами на мові програмування асемблер. Була складена програма, що виконує дії згідно із завданням, у якій були використані різні типи адресації даних, такі як регістрова, безпосередня, базово-індексна та різні типи стрибків. Нарешті, еквівалентна програма була написана на мові програмування С++, та дизасембльована для вивчення.