Лабораторна робота №5

Програмування множення чисел підвищеної розрядності

Мета: Навчитися програмувати на асемблері множення чисел підвищеної розрядності, а також закріпити навички програмування власних процедур у модульному проекті.

Завдання:

1. Створити у середовищі MS Visual Studio проект з ім'ям **Lab5**.
2. Написати вихідний текст програми згідно варіанту завдання. У проекті мають бути три модуля на асемблері:
   - головний модуль: файл **main5.asm**. Цей модуль створити та написати заново, частково використавши текст модуля main4.asm попередньої роботи №4;
   - другий модуль: використати **module** попередніх робіт №3, 4;
   - третій модуль: модуль **longop** попередньої роботи №4 доповнити новим кодом відповідно завданню.
3. У цьому проекті кожний модуль може окремо компілюватися.
4. Скомпілювати вихідний текст і отримати виконуємий файл програми.
5. Перевірити роботу програми. Налагодити програму.
6. Отримати результати – кодовані значення чисел згідно варіанту завдання.
7. Проаналізувати та прокоментувати результати, вихідний текст та дизасембльований машинний код програми.

Теоретичні відомості

Обчислення факторіалу

Факторіалом $n!$ зветься добуток цілих чисел від 1 до n

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n$$

Факторіал часто використовується у різноманітних галузях математики. Наприклад, з комбінаторики відомо, що кількість усіх можливих перестановок n елементів дорівнює $n!$

Числові значення факторіалу стрімко зростають при збільшенні n. Це ускладнює розрахунки там де потрібна велика точність. Можна розглянути декілька прикладів значень факторіалу (табл. 1).

Таблиця 1

Деякі значення факторіалу

n	$n!$ (приблизні значення для $n>10$)	Кількість бітів, потрібних для точного представлення $n!$
1	1	1
2	2	2
3	6	3
4	24	5
5	120	7
6	720	10
7	5040	13
8	40320	16
9	362880	19
10	3628800	22
20	$2.4329 \cdot 10^{18}$	62
30	$2.65253 \cdot 10^{32}$	108
40	$8.15915 \cdot 10^{47}$	160
50	$3.04141 \cdot 10^{64}$	215
60	$8.32099 \cdot 10^{81}$	273
70	$1.1979 \cdot 10^{100}$	333
80	$7.1569 \cdot 10^{118}$	395
90	$1.4857 \cdot 10^{138}$	459
100	$9.3326 \cdot 10^{157}$	525

Для обчислення приблизного значення факторіалу можна скористатися формулою Стірлінга. У першому наближенні оцінка має вигляд:

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

Алгоритм для точного обчислення факторіалу є дуже простим:

```
factorial = 1;
for (i=2; i<=n; i++)
   factorial *= i;
```

проте, щоб реалізувати точні обчислення, потрібно виконувати множення чисел великої розрядності. Наприклад, при обчисленні 100! потрібно на останньому кроці перемножувати 519-бітне значення щоб отримати 525-бітовий результат.

Взагалі, для представлення будь-якого цілого позитивного числа N потрібно не менше $(1+\log_2 N)$ двійкових розрядів.

Множення підвищеної розрядності

У якості прикладу розглянемо множення двох 96-бітових операндів A та B. Результат буде мати удвічі більшу розрядність – 192 біти. Один з можливих алгоритмів виконання множення групами по 32 біт полягає у множенні одного 96-бітного операнду (A) на групи 32 бітів іншого операнду (B_i). У свою чергу множення 96-бітного A на групу бітів B_i виконується за три кроки – на кожному кроці отримується 64-бітовий добуток двох 32-бітових груп. Для отримання результату потрібно додати усі 64-бітові добутки відповідно їхньому розташуванню у 192-бітовій розрядній сітці. Алгоритм відображений на рис. 1.

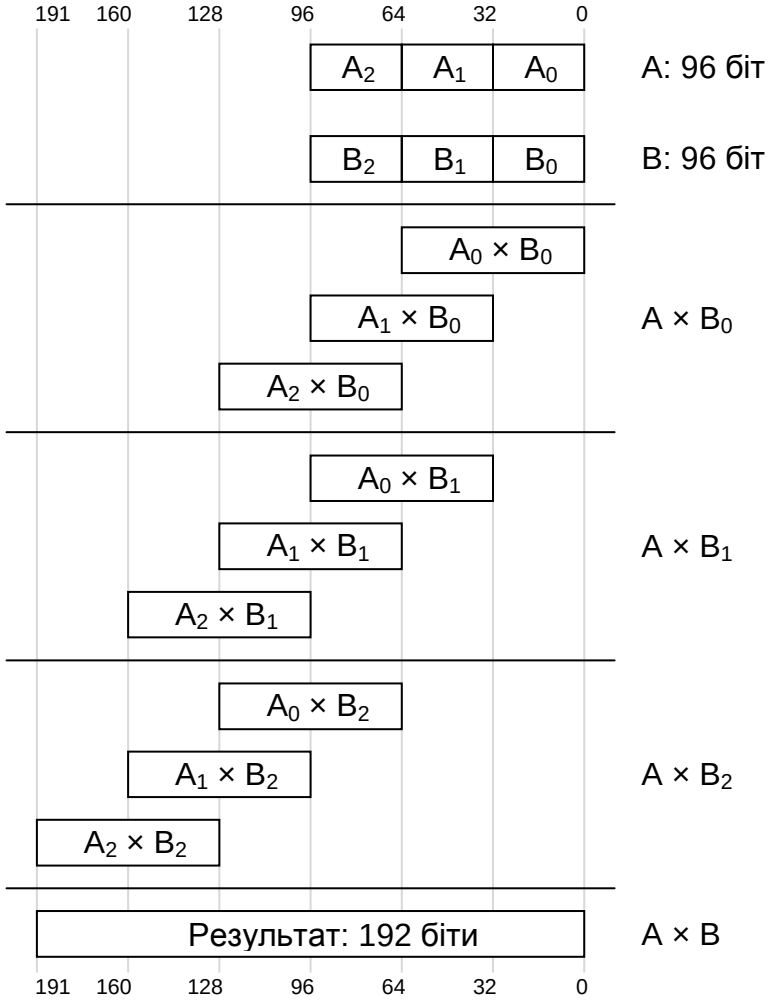

Рис. 1. Множення 96-бітових чисел групами по 32 біти

Наведений тут алгоритм коректно працює тільки для чисел без знаку.

Реалізувати такий алгоритм можна по-різному. Це обумовлено тим, що додавати 64-бітові часткові добутки можна у будь-якій послідовності.

У програмі на асемблері обчислення часткових добутків ($A_i \times B_i$) можна виконати командою MUL – множення чисел без знаку.

```
mul src
```

Ця команда виконує множення операнду src на значення у регістрі AL, або AX, або EAX, або RAX у залежності від розрядності операнду src. Результат записується відповідно у регістр AX, або регістри AX:DX, або у регістри EAX:EDX, або у регістри RAX:RDX. Якщо операнд є 32-бітовим, то результат множення EAX*src буде 64-бітовим: молодші 32 біти результату записуються у EAX, старші – у EDX.

Множення N×32

Розглянемо дещо скорочений варіант множення підвищеної розрядності – коли один з операндів повнорозрядний, а інший операнд 32-бітний. Навіщо потрібен такий різновид множення? Він може бути використаний, коли один з множників гарантовано може представлятися не більш, як 32 бітами. Наприклад, при обчисленні факторіалу $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n$, якщо $n < 2^{32}$.

Порівняно із множенням N×N, множення N×32 є набагато простішим та, відповідно, швидшим. У цьому випадку можна згрупувати часткові добутки так, щоб додавати лише дві порції бітів (рис. 3)

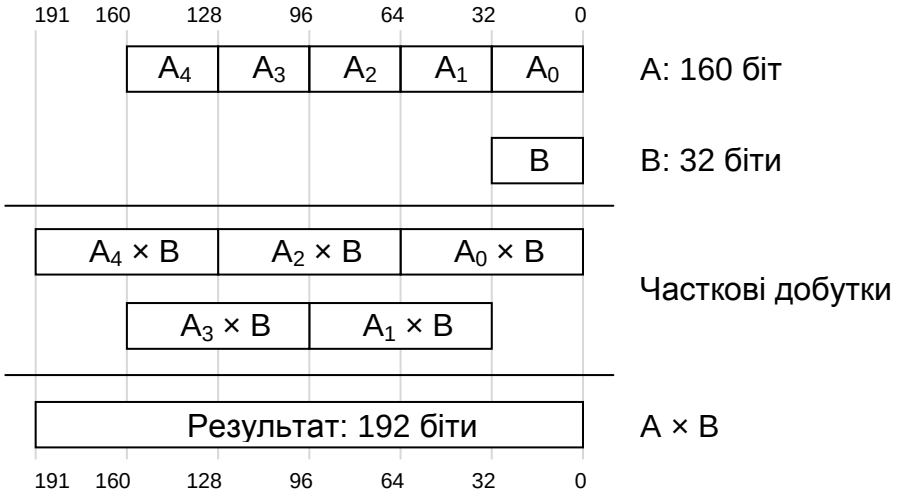

Рис. 3. Множення 160-бітового числа на 32-бітне

Перша порція – добутки $A_0 \times B$, $A_2 \times B$, $A_4 \times B$ прямо записуються у відповідні 32-бітові групи результату. Потім додаються часткові добутки з непарними індексами – $A_1 \times B$, $A_3 \times B$.

Деякі особливості програмування циклів

При програмуванні на асемблері таких операцій, як множення підвищеної розрядності, може виникнути проблема нестачі регістрів процесора. Для забезпечення високої швидкодії у якості лічильників циклів бажано використовувати регістри процесора, проте часто трапляється так, що регістрів загального призначення не вистачає. У цьому випадку для лічильників циклів залишається використовувати перемінні, розташовані у сегменті даних або у стеку. Це уповільнює роботу, особливо при виконанні великої кількості циклів, у тілі яких міститься мало команд.

Цикли можуть бути вкладеними, тому рекомендується використовувати регістри у першу чергу у внутрішньому циклі. Наприклад:

```
 .data
   counter dd 0              ; ця перемінна буде лічильником циклу1
.code

@cycle1:                     ; початок зовнішнього циклу1
   mov eax, counter          ; завантажуємо значення перемінної
   inc eax                   ; збільшуємо лічильник на одиницю
   cmp eax, maxcycle1        ; порівнюємо з макс. значенням лічильника
   jg @exit                  ; вихід з циклу
   mov counter, eax          ; зберігаємо значення лічильника у пам'яті
   . . .                     ; тіло зовнішнього циклу1
   mov ecx, numcycle2              ; кількість повторень для циклу2
@cycle2:                           ; початок внутрішнього циклу2
   . . .                           ; тіло внутрішнього циклу2
   dec ecx                         ; зменшуємо лічильник циклу2 у регістрі ECX
   jnz @cycle2                     ; перехід на початок циклу2
   . . .                     ; тіло зовнішнього циклу1
   jmp @cycle1               ; перехід на початок циклу1
```

У наведеному вище прикладі є два цикли. У тілі першого циклу міститься вкладений цикл. Для вкладеного циклу лічильник у регістрі ECX. Лічильник для зовнішнього циклу зберігається у перемінній counter.

Порядок виконання роботи та методичні рекомендації

1. Створіть у середовищі MS Visual Studio новий проект з ім'ям **Lab5**.

2. Додайте у проект порожній файл з ім'ям **main5.asm**. Цей файл буде головним файлом програмного коду. Для спрощення виконання роботи скористайтеся

текстом головного файлу *.asm попередньої роботи №4. Скопіюйте текст і у вікні редагування вихідного тексту вилучіть зайві рядки. Запишіть на диск головний файл програми **main5.asm**.

3. Додайте у проект модуль з ім'ям **module**. У проекті використовується файли **module.asm, module.inc** попередніх робіт №3, 4 без будь-яких змін. Рекомендація: для того, щоб у декількох проектах використовувати ті самі модулі, запишіть файли цих модулів у окрему папку. Кожний файл вихідного тексту модулів, які спільно використовується, повинен бути у одному екземплярі.

4. Додайте у проект модуль **longop**. У проекті використовуються файли **longop.asm**, **longop.inc** попередньої роботи №4. У ці файли повинні бути додані програмні коди процедур множення N×N та N×32. Для цього необхідно визначити розрядність для даних та операцій згідно варіанту завдання.

5. У файлі **main5.asm** потрібно запрограмувати цикл для обчислення значення факторіалу згідно варіанту завдання. Рекомендується у циклі обчислення факторіалу використати процедуру множення N×32. Отримане значення $n!$ потім возвести у квадрат – перемножити за допомогою процедури множення N×N.

6. Запрограмувати вивід результатів у діалоговому вікні MessageBox. Запрограмувати вивід потрібних числових значень у шістнадцятковому коді.

7. Компіляція, виклик програми, налагодження, отримання результатів. Виконання цих дій виконується у середовищі MS Visual Studio. Відомості та методичні рекомендації надані у відповідних розділах попередніх робіт.

Зміст звіту:

1. Титульний лист
2. Завдання
3. Роздруківка тексту програми
4. Роздруківка результатів виконання програми
5. Аналіз, коментар результатів, вихідного тексту та дизасембльованого машинного коду
6. Висновки

Варіанти завдання

Для кожного студента своє значення *n*, яке визначається за формулою:

$n = 30 + 2 \times H$,

де H – це номер студента у журналі.

Потрібно запрограмувати на асемблері:

- обчислення факторіалу $n!$
- обчислення квадрату факторіалу $n! \times n!$

Точні цілі значення результатів надати у шістнадцятковій системі числення.

Контрольні питання:

1. Які проблеми виникають при програмуванні обчислення факторіалу?
2. Як оцінити значення факторіалу?
3. Як визначити потрібну розрядність для виконання операцій та представлення результатів?
4. Як виконується множення підвищеної розрядності?
5. Як працює команда MUL?
6. Як запрограмувати цикли на асемблері при обмеженій кількості регістрів?