МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ»
КАФЕДРА ЕЛЕКТРОННОЇ ТЕХНІКИ

Методичні вказівки

до практичних робіт

з дисципліни

«СИСТЕМНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ»

для студентів спеціальності

123 комп'ютерна інженерія

усіх форм навчання

Луцьк
2023

УДК 004.45(072)
М 54

Методичні вказівки до практичних робіт з дисципліни «Системне програмне забезпечення» для студентів спеціальності 123 Комп’ютерна інженерія усіх форм навчання. [Електронний ресурс] / уклад. В.В. Шамаєв, С.О. Ковальов, А.С. Любимов. – Луцьк : ДонНТУ, 2023. – 102 с.

Методичні вказівки містять основні положення щодо вимог, виконання, структури та оформлення практичних робіт з дисципліни «Системне програмне забезпечення» студентами бакалаврату ДонНТУ, що навчаються за спеціальністю 123 Комп’ютерна інженерія усіх форм навчання. Наведено зразки виконання робіт, видів текстового матеріалу, таблиць, формул та ілюстрацій.

Методичні вказівки розроблено за участю О.Г. Шевченко. Укладачі щиро вдячні їй за співпрацю та надані матеріали.

Призначено для бакалаврів спеціальності 123 Комп’ютерна інженерія усіх форм навчання

Укладачі: В.В. Шамаєв, доц., к.т.н., доц. каф. ЕТ
С.О. Ковальов, доц., к.т.н, доц. каф. ЕТ
А.С. Любимов, асс., каф. ЕТ

Рецензент: Н.О. Маслова, доц., к.т.н., доц. каф. ПМІ.

Відповідальний за випуск: С.О. Ковальов, проф., к.т.н., зав. каф. ЕТ.

Затверджено навчально-методичним відділом ДонНТУ
Протокол № 3 від 28.11.2023 р.

Розглянуто на засіданні кафедри електронної техніки
Протокол № 4 від 13.11.2023 р.

ЗМІСТ

ВСТУП

Методичні вказівки включають рекомендації до виконання практичних робіт з дисципліни «Системне програмне забезпечення».

Практичні роботи присвячені різним темам, що викладаються в даному курсі. На прикладі першої практичної роботи студенти мають можливість втілити на практиці не тільки початкові знання з програмуванням на мові Асемблер, а також опанувати навичками роботи з пакетом програм MASM32, який, використовується розв’язання багатьох задач у цьому курсі.

Складність завдань наведених у практичних роботах збільшується поступово, що, при їх своєчасному виконанні, сприяє позитивному засвоєнню теоретичного матеріалу. Кожна з практичних робіт має чітку структуру: теоретичні свідомості у сукупності з лекційними та при необхідності звернення до літератури являють достатній обсяг матеріалу для виконання вказаних завдань. Окремо виділяється сьома практична робота, яка знайомить студентів з офіційними документами з специфікації команди процесору CPUID різних виробників [2].

Надані методичні рекомендації дозволяють використати цю інформацію для реалізації поставлених завдань. Остання (восьма) практична робота допоможе студентам набути практичних навиків з обробці текстових даних та за бажанням переглянути попередні роботи (де виконувалась обробка масивів, матриць) з метою вдосконалення їх знань з програмування [2].

ПРАКТИЧНА РОБОТА №1.
ОРГАНІЗАЦІЯ ПРОСТИХ ЛІНІЙНИХ ОБЧИСЛЕНЬ

Мета. Придбання навичок використання арифметичних команд для організації цілочислових обчислень.

1.1 Структура програми

Структурно програма на мові Асемблер складається з декількох програмних сегментів певного типу: даних, стека і коду. При цьому тільки один з них є обов'язковим - сегмент коду. Кількість необхідних програмних сегментів різного типу визначає програміст [2]. Шаблон програми 32-ох розрядного додатку представлено нижче:

```
.586                          ; визначення «типу (покоління)» процесору, на якому можливе
                    ; виконання програми
.model flat, stdcall                  ; модель пам’яті та узгодження про виклики
Option casemap: none                  ; враховувати регістр символів в ідентифікаторах користув.
.stack                                ; опис заголовку сегмента стеку
.data                                 ; опис заголовку сегмента даних
a   db  5                             ; опис змінної у сегменті даних
……..
.code                                 ; опис заголовку сегмента коду
Begin  :        xor eax, eax          ; вміст сегмента коду ( інструкції Асемблеру)
……
end begin                             ; завершення сегменту коду та визначення точки входу
                              ; програми
```

Сегмент даних містить опис вхідних та вихідних змінних необхідних для роботи програми, сегмент коду являє закодований на мові Асемблера алгоритм програми. Основне призначення сегмента стека - організація взаємозв'язку між програмними одиницями, що іменуються підпрограмами [2].

1.2 Призначення та структура регістру прапорів

Регістр прапорів - один з небагатьох регістрів, який немає імені. Прапорами називаються окремі біти цього регістра. Виходячи з особливостей використання, всі прапори можна розділити на три групи:

- п'ять прапорів стану;
- прапор управління;
- вісім системних прапорів.

Прапори стану відображають результат виконання арифметичних і логічних операцій. Прапор управління використовується в ланцюжкових командах, задаючи

напрямок обробки операндів символьного типу. Системні прапори управляють введенням / виводом, налагодженням, перериваннями тощо [2,3].
Перелік розташування окремих прапорів в регістрі прапорів представлено в табл.1.1.

Таблиця 1.1 Розташування прапорів в регістрі прапорів [2].

Біт	Позначка	Назва	Пояснення	Група
0	CF	Carry Flag	Прапор переносу	Стан
1	1	Зарезервировано		
2	PF	Parity Flag	Прапор парності	Стан
3	0	Зарезервирован		
4	AF	Auxiliary Carry Flag	Допоміжний прапор переносу	Стан
5	0	Зарезервирован		
6	ZF	Zero Flag	Прапор нульового результату	Стан
7	SF	Sign Flag	Прапор знаку	Стан
8	TF	Trap Flag	Прапор трасування	Системний
9	IF	Interrupt Enable Flag	Прапор дозволу переривання	Системний
10	DF	Direction Flag	Прапор напряму	Управління
11	OF	Overflow Flag	Прапор переповнення	Стан
12 13	IOPL	I/O Privilege Level	Рівень пріоритету введення/ виведення	Системний
14	NT	Nested Task	Прапор вкладеності задач	Системний
15	0	Зарезервирован		
16	RF	Resume Flag	Прапор відновлення	Системний
17	VM	Virtual-8086 Mode	Режим віртуального процесора 8086	Системний
18	AC	Alignment Check	Перевірка вирівнювання	Системний
19	VIF	Virtual Interrupt Flag	Віртуальний прапор дозволу переривання	Системний
20	VIP	Virtual Interrupt Pending	Очікуюче віртуальне переривання	Системний
21	ID	ID Flag	Перевірка доступу до інструкції CPUID	Системний
22		Зарезервовано		

Біт	Позначка	Назва	Пояснення	Група
...				
31				

1.3 Команди групи «пересилання даних»

Деякі команди, що відносяться до групи пересилання наведені у табл. 1.2 [2].

Таблиця 1.2 Команди пересилання [2].

№ п/п	Формат команди	Операнди	Пояснення
1	MOV op1,op2	op1,op2 – мають бути однакового розміру (reg,reg), (reg,mem) (mem,reg), (reg,data), (mem,data)	Дані пересилаються з op2 у op1 Результат op1:=op2
2	MOVSX op1,op2	op2 за розміром у 2 або 4 рази менший за op1 (reg,reg), (reg,mem)	Дані пересилаються з op2 у op1, всі інші розряди op1 встановлюються у значення знакового розряду op2.
3	MOVZX op1,op2	op2 за розміром у 2 або 4 рази менший за op1 (reg,reg), (reg,mem)	Дані пересилаються з op2 у op1, всі інші розряди op1встановлюються в 0.
4	CBW	AX,AL (неявні операнди)	Знакове розширення AL до AX
5	CWDE	EAX, AX (неявні операнди)	Знакове розширення AX до EAX
6	CWD	DX:AX, AX (неявні операнди)	Знакове розширення AX до DX:AX
7	CDQ	EDX:EAX, EAX (неявні операнди)	Знакове розширення EAX до EDX:EAX
8	XCHG op1,op2	op1,op2 – мають бути однакового розміру (reg,reg)	Обмін вмісту op1 та op2
9	BSWAP op1	op1 -32-ух розрядний регістр загального призначення	Міняє порядок байт на зворотний (1,2,3,4 на 4,3,2,1)

Позначення у таблиці 1.2:

- reg – операнд задано у регістрі;
- mem – операнд задано в сегменті даних/стеку.
- data - операнд задано константою.

Команди пересилання не впливають на регістр прапорів [2].

1.4 Арифметичні команди

Формат, призначення команд двійкової арифметики наведено у табл. 1.3[2,4].

Таблиця 1.3. Арифметичні команди] [2].

№ п/п	Формат команди	Операнди	Пояснення	Прапори, що змінюються
1	ADD op1,op2	op1,op2 – мають бути однакового розміру (reg,reg), (reg,mem) (mem,reg),(reg,data),(mem,data)	op1:=op1+op2	CF,PF,ZF SF,OF,AF
2	ADC op1,op2	op1,op2 – мають бути однакового розміру (reg,reg), (reg,mem) (mem,reg),(reg,data),(mem,data)	op1:=op1+op2+CF	CF,PF,ZF SF,OF,AF
3	SUB op1,op2	op1,op2 – мають бути однакового розміру (reg,reg), (reg,mem) (mem,reg),(reg,data), (mem,data)	op1:=op1-op2	CF,PF,ZF SF,OF,AF
4	SBB op1,op2	op1,op2 – мають бути однакового розміру (reg,reg), (reg,mem) (mem,reg),(reg,data),(mem,data)	op1:=op1-op2-CF	CF,PF,ZF SF,OF,AF
5	CMP op1,op2	op1,op2 – мають бути однакового розміру (reg,reg), (reg,mem) (mem,reg),(reg,data),(mem,data)	Порівняння op1 з op2. Виконує операція op1-op2, при цьому вміст операндів не змінюється	CF,PF,ZF SF,OF,AF
6	DEC op1	mem / reg	op1:=op1-1	PF,ZF SF,OF,AF,
7	INC op1	mem / reg	op1:=op1+1	PF,ZF SF,OF,AF
8	MUL op2	op1(неявний операнд)- регістр акумулятора, op2 - mem / reg. op1,op2 – мають бути однакового розміру	Без знакове множення op1*op2 Результат- AX або DX:AX або EDX:EAX в залежності від розміру явного операнду (op2).	CF=OF=0, або CF=OF=1, якщо розмір результату більш за розмір операндів
9	IMUL op2	op1(неявний операнд)- регістр акумулятора, op2- mem / reg. op1,op2 – мають бути однакового розміру	Знакове множення op1*op2 Результат- AX\ DX:AX\ EDX:EAX	Як для команди MUL
№	Формат ко-	Операнди	Пояснення	Прапори,

п/п	манди			що змінюються
10	IMUL op1,op2	op1 -тільки reg, op2 – data / reg	op1=op1*op2	Як для команди MUL, можлива, часткова втрата результату.
11	IMUL op1,op2,op3	op1 -тільки reg, op2 – mem / reg op3 – тільки data	op1=op2*op3	Як для команди MUL, можлива часткова втрата результату
12	DIV op2	op1(неявний операнд) – тільки регістри (AX / DX:AX / EDX:EAX) op2 – mem /reg, розміром в 2 рази меншим за op1	Без знакове ділення. Результат- частка, залишок op1/ op2= частка в AL, залишок в AH; або частка в AX, залишок в DX; або частка в EAX, залишок в EDX в залежності від розміру явного операнду (op2)	Непевний стан
13	IDIV op2	op1 – тільки регістри (AX, DX:AX, EDX:EAX) op2 – mem або reg, розміром в 2 рази меншим за op1	Знакове ділення Результат- частка, залишок op1/ op2= частка в AL, залишок в AH; або частка в AX, залишок в DX; або частка в EAX, залишок в EDX в залежності від розміру явного операнду (op2)	Непевний стан

1.5 Команди зсуву та ротації

Операції порозрядного зсуву переміщають окремі біти у межах вказаного операнду. Напрямок зсуву може бути ліворуч або вправоруч. При зсуві вправо на

один розряд крайній правий біт втрачається, а крайній лівий заміщається нулем. При зсуві вліво втрачається крайній лівий біт, а крайній правий заміщається нулем [2].

При зсуві на кілька розрядів «виштовхується» вказана кількість бітів, а звільнені заміщуються нулями. Ротація - циклічний зсув, не виштовхує біти, а повертає їх на місце звільнених. В результаті ротації вправо крайній правий біт встане на місце крайнього лівого, а інші біти будуть зрушені на одну позицію вправо [3,4].

Команди зсуву та ротації при виконанні використовують прапор CF, куди потрапляє зміщений розряд. Команди зсуву, їх формат та пояснення виконання наведено в табл. 1.4. [2].

Таблиця 1.4 Команди зсуву та ротації [2].

н.п	Формат команди	Операнди	Пояснення
1	SHR op1,op2	op1 - mem / reg, op2 – data / CL	Зсув op1 вправо на кількість розрядів, вказаних op2
2	SHL op1,op2	op1 – mem / reg, op2 – data / CL	Зсув op1 вліво на кількість розрядів, вказаних op2
3	SAR op1,op2	op1 - mem / reg, op2 – data / CL	Арифметичний зсув op1 вправо на кількість розрядів, вказаних op2. Звільнені зліва розряди op1 встановлюються у значення його знакового розряду .
4	SAL op1,op2	op1 - mem / reg, op2 – data / CL	Зсув op1 вліво на кількість розрядів, вказаних op2 (аналог SHL)
5	ROR op1,op2	op1 - mem / reg, op2 – data / CL	Ротація op1 вправо на op2 розрядів
6	ROL op1,op2	op1 - mem / reg, op2 – data / або CL	Ротація op1 вліво на op2 розрядів
7	RCR op1,op2	op1 - mem / reg, op2 – data / CL	Ротація op1 вправо на op2 розрядів разом з CF
8	RCL op1,op2	op1 - mem / reg, op2 – data / або CL	Ротація op1 вліво на op2 розрядів разом з CF
9	SHLD op1,op2,op3	op1,op2 – мають бути однакового розміру (reg,reg), (reg,mem) (mem,reg), op3 – тільки data, або CL	Зсув op1 та op2 на op3 розрядів вліво, при цьому op1 змінюється, op2-ні.
10	SHRD op1,op2,op3	op1,op2 – мають бути однакового розміру (reg,reg), (reg,mem) (mem,reg), op3 – тільки data, або CL	Зсув op1 та op2 на op3 розрядів вправо при цьому op1 змінюється, op2-ні.

1.6 Завдання

Скласти програму для обчислення заданої формули. Вхідні та вихідні данні - цілочислові зі знаком, розміром у чотири байти [2]:

1. a=d+2*f+(b+1)/6
2. a= (b-d)*7+f/2-1
3. a=b*4-d/13+10
4. a=(b+f)/16-25
5. a=100/(d-f)+b
6. $a=8*b+f-12_{16}$
7. a=f+d+b/2-30
8. a=(f-d)*4+b/234
9. $a=(b+f)/3-d*2-50_{8}$
10. a=120-d*6+b/2+f
11. a=f+(d-b)*8-13
12. a=(16-f)/2+b*4
13. $a=d-200_{16}+(b+f)/8$
14. a=d/2+(b-f)*b
15. a=d+f-b/4+40
16. a=d-7*f-b+123
17. $a= b*d-f/3-56_{16}$
18. a=b/5-(d*4-1)/2
19. $a=(b-f)*2+25_{16}/ f$
20. $a=256/b+(d-f)^2$
21. $a=(b*512-f/12_{8})+2$
22. $a=f-d-b/2+d*100_{16}$
23. a=(f+d)*16-b/f
24. $a=2*(b-f+d)/45+523_{16}$
25. $a=d+b/2-345_{8}+f*d$
26. $a=93_{16}-f*b+(d-b)*2$
27. a=(120-d)/5-b*2+f
28. $a=d+700_{8}+b/(4+f)$
29. a=(d-2)*b-(f+b)/d
30. $a=(b-f)/b+f*876_{16}$
31. $a=(2*d+b)/75_{8}-f$
32. a=f+(d-b)*39/b

1.7 Порядок виконання

1. Визначити початкові вхідні параметри.
2. Обчислити формулу.
3. Визначитись з командами мови, які потрібні для обчислення формули.
4. Скласти програму.
5. Отримати виконавчий файл.
6. Завантажити виконавчий файл засобами відладчика OLLYDBG [2].
7. Виконати програму.
8. Порівняти отримані результати з результатами п.2.
9. Протестувати програму з різними вхідними параметрами.

1.8 Структура звіту

Звіт оформлюється на окремих аркушах формату А4 та має наступний зміст:

1. Титул
2. Постанова завдання
3. Текст програмного додатку
4. Результати роботи.
5. Висновки.

1.9 Питання до захисту

До захисту лабораторної роботи допускаються студенти, що:

- довели працездатність розробленого ПЗ;
- довели достовірність отриманих результатів;
- готові продемонструвати свою спроможність виконати модифікацію програмного коду за вимогою викладача;
- оформили належним чином звіт.

Питання:

1. Поняття програмного сегменту.
2. Опис програмних сегментів.
3. Типи даних.
4. Загальний формат команди.
5. Призначення та структура регістру прапорів.
6. Команди пересилання даних.
7. Арифметичні команди додавання та віднімання.
8. Арифметичні команди множення та ділення.
9. Команди зсуву.
10. Команди ротації.

11. Література : [2], [3], [4], [5], [8].

ПРАКТИЧНА РОБОТА №2.
ОРГАНІЗАЦІЯ РОЗГАЛУЖЕНИХ ОБЧИСЛЕНЬ

Мета. Придбання навичок програмування розгалужених обчислень.

Розгалуженою э програма, в якій за певними умовами змінюється лінійний хід виконання. Програмування подібних ситуацій виконується з використанням операторів «Якщо - То», які в конкретній мові мають свій синтаксис та семантику. На Асемблері для цього призначені команди порівняння та умовного переходу. Однак, враховуючи конвеєрну обробку команд на сучасних процесорах, слід уникати не обґрунтованого використання команд умовного переходу.

2.1 Команди умовного переходу по значенню прапорів

Якщо команда стан свого виконання відображає у регістрі прапорів, то для аналізу результату такої команди зовсім не обов'язково використовувати команду CMP, оскільки Асемблер має в своєму арсеналі спеціальні команди переходу по значенню прапорів (таблиця 2.1). Всі команди мають один операнд, що вказує на команду передачі управління [2,3,4].

Таблиця 2.1 Команди переходу по значенню прапорів стану. [2].

№ п/п	Формат команди	Значення прапору	Пояснення
1	JC op1	CF=1	Перейти за адресою op1, якщо CF=1
2	JP op1	PF=1	Перейти за адресою op1, якщо PF=1
3	JZ op1	ZF=1	Перейти за адресою op1, якщо ZF=1
4	JS op1	SF=1	Перейти за адресою op1, якщо SF=1
5	JO op1	OF=1	Перейти за адресою op1, якщо OF=1
6	JNC op1	CF=0	Перейти за адресою op1, якщо CF=0
7	JNP op1	PF=0	Перейти за адресою op1, якщо PF=0
8	JNZ op1	ZF=0	Перейти за адресою op1, якщо ZF=0
9	JNS op1	NS=0	Перейти за адресою op1, якщо SF=0
10	JNO op1	OF=0	Перейти за адресою op1, якщо OF=0

2.2 Команди переходу при порівнянні операндів

Команда порівняння CMP завжди формує прапори стану виконання, отже розгалуження може бути зроблено командами з таблиці 2.1. Але для зручності читання програми Асемблер дозволяє використовувати еквівалентні за призначенням інші команди для знакових операндів, а також команди переходу при порівнянні без

знакових операндів. Формат та пояснення виконання таких команд наведено в таблиці 2.2 [2].

Таблиця 2.2 Команди переходу при порівнянні операндів [2].

№	Формат команди	Пояснення	*Операнди*
1	JE op1	Перехід, якщо порівняльні операнди рівні.	Не має значення
2	JNE op1	Перехід, якщо порівняльні операнди нерівні	Не має значення
3	JL/JNGE op1	Перехід, якщо перший з порівняльних операндів менше за другий	Знакові
4	JLE/JNG op1	Перехід, якщо перший з порівняльних операндів менше за другий або дорівнює йому.	Знакові
5	JG/JNLE op1	Перехід, якщо перший з порівняльних операндів більше за другий	Знакові
6	JGE/JNL op1	Перехід, якщо перший з порівняльних операндів більше за другий або дорівнює йому	Знакові
7	JB/JNAE op1	Перехід, якщо перший з порівняльних операндів менше за другий	Без знакові
8	JBE/JNA op1	Перехід, якщо перший з порівняльних операндів менше за другий або дорівнює йому	Без знакові
9	JA/JNBE op1	Перехід, якщо перший з порівняльних операндів більше за другий	Без знакові
10	JAE/JNB op1	Перехід, якщо перший з порівняльних операндів більше за другий або дорівнює йому	Без знакові
11	JCXZ op1	Перехід, якщо регістр CX=0	Не має значення
12	JECXZ op1	Перехід, якщо регістр ECX=0	Не має значення

Приклад,

```
        CMP  eax, 10          ; Вміст регістру EAX порівнюється з 10
        JE     M1             ; Перехід на команду з міткою M1, якщо EAX=10
        INC eax               ; Якщо EAX!=10, встановити EAX=EAX+1
        .....
M1:     DEC eax               ; EAX=EAX-1
        JZ   M2               ; Перехід на мітку M2, якщо EAX=0
        ......                ; Продовження виконання програми, якщо EAX!=0
```

M2: ..... ; Продовження виконання програми, якщо EAX=0

2.3 Логічні команди

В основі логічної обробки даних лежить логіка висловлювань. Висловлення це твердження, яке може бути або істинним або хибним. Над висловленнями можуть виконуватись логічні операції такі, як:

- логічне заперечення – логічне НІ (таблиця 2.3);
- логічне додавання - логічне АБО (таблиця 2.4);
- логічне множення – логічне І (таблиця 2.5);
- логічне виключне додавання – логічне виключне АБО (таблиця 2.6) [2].

Таблиця 2.3. Логічне заперечення [2].

Операнд	Результат
0	1
1	0

Таблиця 2.4 Логічне додавання [2].

Операнд 1	Операнд 2	Результат
0	0	0
0	1	1
1	0	1
1	1	1

Таблиця 2.5 Логічне множення [2].

Операнд 1	Операнд 2	Результат
0	0	0
0	1	0
1	0	0
1	1	1

Таблиця 2.6 Логічне виключне додавання [2].

Операнд 1	Операнд 2	Результат
0	0	0
0	1	1
1	0	1
1	1	0

Логічні команди процесору виконують логічні операції над бітами операндів. Формат команд та пояснення до виконання наведено у таблиці 2.7. [2].

Таблиця 2.7 Логічні команди [2].

№ п/п	Формат команди	Операнди	Пояснення
1	NOT op1	op1 – mem / reg	Виконується операція логічного заперечення для кожного біта op1 op1=! op1
2	OR op1,op2	op1,op2 – мають бути однакового розміру (reg, reg), (reg, mem), (mem, reg),(reg, data),(mem, data)	Виконується операція логічного додавання над кожними відповідними бітами op1 та op2 op1=op1 | op2
3	AND op1,op2	op1,op2 – мають бути однакового розміру (reg, reg), (reg, mem), (mem, reg),(reg, data),(mem, data)	Виконується операція логічного множення над кожними відповідними бітами op1 та op2 op1=op1 & op2
4	XOR op1,op2	op1,op2 – мають бути однакового розміру (reg, reg), (reg, mem), (mem, reg),(reg, data),(mem, data)	Виконується операція логічного виключного додавання над кожними відповідними бітами op1 та op2 op1 = op1 ^ op2
5	TEST op1,op2	op1,op2 – мають бути однакового розміру (reg, reg), (reg, mem), (mem, reg),(reg, data),(mem, data)	Виконується операція логічного множення над кожними відповідними бітами op1 та op2, але op1 не змінюється

За допомогою логічних команд можливе виділення окремих бітів в операнді з метою їх установки, скидання, інвертування або перевірки на відповідне значення. Для організації такої роботи з бітами другий операнд зазвичай грає роль маски. Маска містить одиничні біти в тих розрядах, які потрібно перевірити у першому операнді, розряди в інших позиціях маски повинні бути нульовими [2].

Приклад. Якщо 4-ий та 14 розряди деякого параметру встановлено в 1, виконати інверсію цього параметру [2]. Нумерація розрядів починається з 0.

```
.......parametr    DD ?                ; опис параметру в сегменті даних

.code
.....
  Mov parametr, EAX
 test parametr, 4010h ; операція логічного множення на константу, в якій 4-й та 14-й
```

```
jz m1          ; біти дорівнюють 1. Перехід на M1, якщо 4-ий та 14-ий розряди у
                 parametr; не 1.
not prametr    ; виконання операції логічного заперечення
.....
```

Виконання команди виключного додавання у форматі XOR op,op, де в якості обох операндів вказано один і той же, призводить до встановлення в 0 всіх бітів операнду [2].

Крім означених вище, Асемблер підтримує інші побітові команди деякі з них наведено у таблиці 2.8. [2].

Таблиця 2.8 Побітові команди [2].

№ п/п		Формат команди	Операнди	Пояснення
1		BSF op1,op2	op1,op2 – мають бути однакового розміру (reg, reg), (reg, mem), (mem, reg),(reg, data),(mem, data)	Сканування вперед – пошук першого одиничного біту в op2, починаючи з молодшого розряду та запис в op1 номера знайденого біту
2		BSR op1,op2	op1,op2 – мають бути однакового розміру (reg, reg), (reg, mem), (mem, reg),(reg, data),(mem, data)	Сканування назад – пошук першого одиничного біту в op2, починаючи зі старшого розряду та запис в op1 номера знайденого біту
3		BT op1,op2	op1,op2 – мають бути однакового розміру (reg, reg), (reg, mem), (mem, reg),(reg, data),(mem, data)	Занесення перевіряючого біту в регістр прапорів - CF. op2 задає номер перевіряє мого біту в op1
4		BTS op1,op2	op1,op2 – мають бути однакового розміру (reg, reg), (reg, mem), (mem, reg),(reg, data),(mem, data)	Занесення перевіряючого біту в регістр прапорів - CF, після чого встановлюється перевіряючий біт в 1. op2 задає номер перевіряє мого біту в op1
5		BTR op1,op2	op1,op2 – мають бути однакового розміру (reg, reg), (reg, mem), (mem, reg),(reg, data),(mem, data)	Занесення перевіряючого біту в регістр прапорів - CF, після чого встановлюється перевіряючий біт в 0. op2 задає номер перевіряє мого біту в op1

Приклад,

```
btr param, eax      ;  перевірити та встановити в 0 20-ий біт в param
jc M1                       ; перехід на M1, якщо 20-ий біт в param був 1
```

2.4 Завдання [2].

Скласти програму, що реалізує завдання у відповідності до варіанту. Вхідні параметри – знакові 32-ох розрядні цілочислові дані:

1. Дано два числа, якщо перше більше другого у 2 рази, обчислити різницю цих чисел, інакше - суму.
2. Дано три числа, знайти найбільше серед них.
3. Дано два числа, не виконуючи порівняння цих чисел, знайти найбільше з них.
4. Дано три числа, визначити чи є серед них однакові за значенням.
5. Дано три числа, визначити чи є серед них кратні 2.
6. Дано три числа, визначити чи є серед них непарні.
7. Дано три числа, знайти найменше серед них.
8. Дано два числа обміняти старший розряд одного числа з молодшим другого.
9. Дано два числа, якщо значення 5-го розряду в першому числі 0, змінити друге число на суму цих чисел.
10. Дано три числа. Знайти серед них мінімальне за модулем.
11. Дано три числа, визначити чи є серед них однакові за знаком.
12. Дано три числа, якщо сума їх позитивна, змінити друге число на протилежне.
13. Дано три числа, якщо різниця їх негативна, обміняти значення крайніх чисел.
14. Дано три числа, якщо сума їх дорівнює нулю, збільшити кожне число у 2 рази.
15. Дано два числа, якщо значення 12-го розряду у другому числі 1, змінити перше число на різницю цих чисел.
16. Дано два числа, якщо вони парні обчислити суму цих чисел.
17. Дано два числа, якщо обидва вони непарні – перетворити їх на парні.
18. Дано два числа, якщо перше більше за друге, виконати операцію ділення, інакше – множення.
19. Дано два числа. Обміняти в кожному з них молодший та старший розряди. Обчислити суму чисел до та після обміну.
20. Дано два числа. Обміняти пару розрядів (2-ий та 13-тий) одного числа з такою ж парою другого числа.
21. Дано три числа. Відсортувати їх в порядку зростання.
22. Дано три числа. Знайти серед них найбільше та обміняти містами з другим.

23. Дано два числа. Якщо перше менше другого встановити в 0 непарні за розкладом розряди другого числа. Обчислити суму чисел до та після змін.
24. Дано два числа. Якщо перше більше другого, а друге – парне, перетворити друге на непарне, а перше зробити менше другого.
25. Дано два числа. Якщо їх різниця більша за суму, обміняти числа місцями.

2.5 Порядок виконання

1. Визначити початкові вхідні параметри.
2. Розробити алгоритм програми.
3. Аналітично отримати результат.
4. Визначитись з командами мови програмування.
5. Скласти програму.
6. Отримати виконавчий файл.
7. Завантажити виконавчий файл засобами відладчика OLLYDBG.
8. Виконати програму.
9. Порівняти отримані результати з результатами п.3.
10. Протестувати програму з різними вхідними параметрами

2.6 Структура звіту

Звіт оформлюється на окремих аркушах формату А4 та має наступний зміст:

1. Титул.
2. Постанова завдання.
3. Текст програми.
4. Результати роботи.
5. Висновки.

2.7 Питання до захисту

До захисту лабораторної роботи допускаються студенти, що:

- довели працездатність розробленого ПЗ;
- довели достовірність отриманих результатів;
- оформили належним чином звіт.

Питання:

1. Операції логічного заперечення та виключного додавання.
2. Операція логічного множення.
3. Операція логічного додавання.
4. Алгоритм перевірки зазначених бітів.
5. Команди переходу за прапорами.

6. Операції відношення.
7. Команди переходу при порівнянні знакових даних.
8. Команди переходу при порівнянні без знакових даних.
9. Команди сканування бітів.
10. Команди тестування та встановлення біту.
11. Література : [2], [4], [10], [11], [12], [13].

ПРАКТИЧНА РОБОТА №3.
ОРГАНІЗАЦІЯ ЦИКЛІЧНИХ ОБЧИСЛЕНЬ

Мета. Придбання навичок програмування циклічних обчислень.

3.1 Обчислення ефективної адреси

При виконанні програми процесор звертається до пам’яті за операндами та командами. Звернення відбувається шляхом виставлення необхідної адреси на шину адресу. Засіб формування адреси називається режимом адресації. Для 32 – розрядних додатків, що функціонують на процесорах архітектури IA-32, адреса формується за схемою селектор: зміщення. Зміщення називають ефективною адресою (EA) [2].

Схема обчислення ефективної адреси представлена на рис.3.1 [2].

Рисунок 3.1 – Схема формування ефективної адреси [2].

В загальному випадку ефективна адреса визначається як:

EA = База + Індекс * Множник + Зсув.

База, Індекс та Зсув можуть використовуватись між собою в різних поєднаннях, Множник – тільки з Індексом. Якщо у якості бази зазначено регістри ESP, EBP то звернення йде до сегменту стека [2] [4].

3.2 Режими адресації

Пряма адресація. Ефективна адреса дорівнює зсуву, що зазначений в команді.

Приклад:

.Data

```
  A  dd 10
  B  dd  100
.....
.Code
.............
        MOV eax, A          ;  ЕА=Зсув А у сегменті даних
        ADD eax, B          ; ЕА=Зсув В у сегменті даних
    JMP  M1          ; ЕА=Зсув М1 у сегменті коду
```

Адресація - пряма з індексуванням. Ефективна адреса обчислюється як сума Зсуву та Індексу. [2].

Приклад:

```
        XOR esi, esi          ; звернення до пам'яті за операндами команди не відбувається
        MOV eax, D[esi]     ; ЕА= Зсув D+( esi), к зсуву операнда D у сегменті даних додається вміст регістру ESI
        ADD edx, A[esi][edi] ; ЕА= Зсув А+(esi)+(edi),  к зсуву операнда А у сегменті даних додається вміст регістрів ESI, EDI. [2].
```

Непряма адресація – адресація по базі. За адресою, що вказана в команді знаходиться не операнд, а його адреса [2].

Приклад:

```
.Data
  A  dd 10
.....
.Code
.........
        LEA ebx, A          ; у регістр ebx занесли зсув операнду А в сегменті даних
        MOV eax, [ebx]      ; після виконання команди eax = 10
```

Адресація по базі з індексуванням та множником.

Приклад,

```
Data
  A  dd 10,-12,0
.....
.Code
.........
        LEA ebx, A          ; у регістр ebx занесли зсув операнду А в сегменті даних
        INC dword ptr[ebx+esi*4] ; EA=( ebx)+( esi*4).
```

В останній команді модифікатор dword ptr вказується для уточнення, що за ефективною адресою буде обиратися 4-ри байти. [2].

Адресація по базі з індексуванням, множником та зсувом. [2].

Приклад:

```
Data
  A  dd 10,-12,0
.......
.Code
.........
        LEA ebx, A          ; у регістр ebx занесли зсув операнду A в сегменті даних
        INC dword ptr[ebx+esi*4-4] ; EA=( ebx)+( esi*4)-4
```

Безпосередня адресація.

Приклад:

```
        MOV eax, 10             ; EA – не обчислюється, так як операнд (10) безпосередньо
                                ;  знаходиться в самій команді
```

Регістрова адресація.

Приклад:

```
        MOV eax, ecx            ; обидва операнди задані регістрами, тобто відсутнє звер-
нення                                   ; до пам’яті, що потребує визначення EA [2].
```

3.3 Опис масивів

Масив – послідовність елементів однакового типу та розміру. Кожен елемент крім значення має порядковий номер (індекс). Декларувати масив можна порізному.

Приклад:

```
.DATA
mas dd 12, 7, 11, 5, 24, 13 ; опис ініціалізованого масиву розміром 24 байти, кількість
                            ; елементів дорівнює 6
```

Приклад,

```
.DATA
mas dd 12, 7, 11, 5, 24, 13, 10 DUP(?)    ; опис масиву розміром 64 байти, перші
                                          ; 6 елементів ініціалізовані, останні 10 – ні
```

Приклад:

```
.DATA
buf1 dd 20 dup(-1)          ; опис масив розміром 80 байт,
                            кожен елемент масиву встановлено в -1.
```

Приклад:

```
.DATA

buf1 db ' string massage ',13,10    ; опис ініціалізованого масиву розміром 18 байт ( з
                            ; урахуванням 3 пробілів)
```

Приклад:

```
.DATA?
buf1 dd 5 dup(?)                    ; опис неініціалізованого масиву розміром 20 байт
```

Приклад:

```
.DATA
buf2 dd 10 dup(1,2)         ; опис ініціалізованого масиву розміром 80 байт, кількість
                            ; елементів дорівнює 20, кожен непарний встановлюється в 1,
                    ; парний –в 2.
```

3.4 Команди циклу

Якщо виконання яких не будь дій потребує їх повторень, то даний алгоритм іменується ітераційним або циклічним. Для організації подібних операцій використовуються команди, що приведені в таблиці 3.1 [2].

Таблиця 3.1. Команди циклу. [2].

Н.п	Формат	Операнди	Пояснення
1	LOOP op1	op1 – визначає адресу переходу, задається з використанням прямого або непрямого режиму адресації	1. ECX =ECX-1 2. Якщо ECX !=0, то перехід згідно op1 3. Якщо ECX дорівнює 0, виконується наступна за LOOP команда
2	LOOPZ op1	op1 – визначає адресу переходу, задається з використанням прямого або непрямого режиму адресації	1. ECX =ECX-1 2. Якщо ECX!=0 та ZF=1-перехід згідно op1 3. Вихід з циклу : якщо ECX=0, або ZF=0
3	LOOPNZ op1	op1 – визначає адресу переходу, задається з використанням прямого або не-	1. ECX =ECX-1 2. Якщо ECX!=0 та ZF=0-перехід згідно op1

		прямого режиму адресації	3. Вихід з циклу :якщо ECX =0, або ZF=1

Організувати циклічні дії можна і без використання означених команд, через команди порівняння та переходу [2].

Приклад. Скласти фрагмент програми, який в цілочисловому масиві знаходить максимальний за значенням елемент [2].

```
.586
.MODEL FLAT, stdcall
option casemap: none

        .data?
max  dd ?                           ; для збереження максимального елементу

        .data
buf  dd  -10,4,-2,6,8,-1,3,12,-7,5  ; опис ініціалізованого масиву
nx dd 10                                    ; кількість елементів масиву
Pos_max  dd -1              ;       ; для збереження зсуву максимального елементу
        .code
start:                              ; початок програми
; ------------------------------------------------------------
   mov ecx, nx              ; для організації циклу
   xor esi,esi          ; esi :=0
   mov eax, buf[esi] ; в eax перший елемент масиву
   mov max,eax              ; в  max перший елемент масиву
c1:  mov eax, buf[esi]      ;  в eax поточний елемент масиву
   cmp   eax, max           ; порівняння елементу масиву з максимальним
    jle   m1                ;  перехід, якщо елемент масиву менший або дорівнює
                            ; максимальному
   mov max, eax;            ; оновити максимальне значення
  mov pos_max, esi ; запам'ятати зсув максимального елементу
m1: add   esi,4             ; обчислити зсув наступного елементу масиву
   loop  c1                 ; якщо не всі елементи оброблені перейти на C1
.....                       ; в pos_max зсув, в max значення максимального елементу
.....
end start
```

У наведеному прикладі для доступу до елементів масиву використовується режим адресації «пряма з індексуванням», де в регістрі esi вказується зсув елементу відносно початку масиву, спираючись на розмір одного елементу. Слід зазначити,

що запам’ятовувати і зсув і саме значення максимального елементу немає потреби, оскільки значення завжди можна отримати, якщо відома адреса елементу [2].

3.5. Завдання

Скласти програму згідно свого варіанту, вхідні параметри – знакові 32-х розрядні цілочислові дані [2].

1. Визначено довільне число і масив, впорядкований по зростанню. Включити задане число в масив, не порушивши умови впорядкованості.
2. Перший і останній непарні елементи масиву замінити відповідно сумою і різницею цих елементів. Визначити середнє арифметичне елементів до та після змін.
3. У масиві видалити всі елементи, які менші за середнє арифметичне елементів масиву.
4. У масиві після кожного негативного елементу, вставити елемент, рівний модулю негативного елемента.
5. Видалити з масиву максимальний по модулю елемент. Враховуючи, що їх може бути декілька.
6. Якщо сума парних елементів в масиві більше, ніж сума непарних, обміняти перший парний і останній непарний елементи.
7. Знайти в масиві елементи, що найменше відрізняються від середнього арифметичного.
8. Видалити з масиву всі максимальні та мінімальні елементи.
9. Знайти суму елементів, що позиційно знаходяться між максимальним і мінімальним елементами в масиві, враховуючи, що максимальний та мінімальний елементи не повторюються.
10. Кожну пару елементів в масиві (1 та 2, 3 та 4, ...) розташувати так, щоб першим стояв елемент, який має більше за модулем значення.
11. Виконати сортування масиву за зростанням.
12. З масиву видалити всі негативні елементи.
13. З масиву видалити всі елементи кратні вказаному.
14. Перший максимальний по модулю негативний елемент масиву обміняти з першим додатнім.
15. Якщо сума додатних елементів парна, змінити значення елементів масиву за зворотним порядком.
16. У масиві визначити положення і розмір останньої послідовності негативних елементів, до складу якої входить не менше трьох елементів.

17. Визначити, чи є в масиві три числа поспіль, упорядкованих по спадаючій. Визначити початок (індекс) такої послідовності.
18. У масиві в першій серії позитивних елементів (серія - послідовність більше трьох поспіль елементів) поміняти елементи в зворотному порядку.
19. Перетворити масив, розташувавши спочатку непарні елементи, а потім парні.
20. У масиві переставити перший і другий, третій і четвертий ... негативні елементи.
21. Всі числа, що знаходяться в масиві між максимальним і мінімальним елементами, розташувати в зворотному порядку, враховуючи, що максимальний та мінімальний елементи присутні тільки по одному разу.
22. З масиву видалити всі нульові елементи, що розташовані на непарних позиціях крім першого нульового елемента.
23. Обміняти в масиві максимальний парний елемент і мінімальний непарний.
24. Перший елемент масиву замінити сумою парних елементів, а останній - сумою непарних .
25. Визначити середнє арифметичне парних елементів масиву, розташованих на парних позиціях.
26. Якщо сума елементів, що стоять на парних позиціях, більше суми елементів, що стоять на непарних позиціях, то обміняти перший і останній елементи масиву.
27. Визначити, скільки елементів масиву розташовано в діапазоні [a, b] і знайти їх середнє арифметичне.
28. Підрахувати кількість пар сусідніх елементів (1,2; 2,3; ...), для яких перший елемент пари більше другого елемента пари.
29. Знайти першу пару сусідніх елементів (1,2; 3,4; ...), сума значень якої максимальна.
30. У масиві знайти два елементи, різниця значень яких мінімальна.
31. Елементи, позиційно розташовані між першим негативним і останнім позитивним, поставити в зворотному порядку.
32. Якщо сума непарних елементів масиву парна, то додати одиницю до всіх непарних елементів.

3.6 Порядок виконання

Визначити початкові вхідні параметри.

1. Розробити алгоритм програми.
2. Аналітично отримати результат.
3. Визначитись з командами мови програмування.

4. Скласти програму.
5. Отримати виконавчий файл.
6. Завантажити виконавчий файл засобами відладчика OLLYDBG.
7. Виконати програму.
8. Порівняти отримані результати з результатами п.3.
9. Протестувати програму з різними вхідними параметрами

3.7 Структура звіту

Звіт оформлюється на окремих аркушах формату А4 та має наступний зміст:

1. Титул.
2. Постанова завдання.
3. Текст програми.
4. Результати роботи.
5. Висновки

3.8 Питання до захисту

До захисту лабораторної роботи допускаються студенти, які:

- довели працездатність розробленого ПЗ;
- довели достовірність отриманих результатів;
- готові продемонструвати свою спроможність виконати модифікацію програмного коду за вимогою викладача;
- готові захищати свої рішення;
- оформили належним чином звіт.

Питання:

1. Схема обчислення ефективної адреси.
2. Режим адресації « регістрова», « безпосередня».
3. Режим адресації « пряма», « непряма».
4. Режим адресації « пряма з індексуванням».
5. Режим адресації « по базі».
6. Засоби опису масивів.
7. Команда LOOP.
8. Команда LOOPZ
9. Команда LOOPNZ

10.Доступ до елементів масиву при різних режимах адресації.

11. Література : [2], [5], [6], [10], [12], [13].

ПРАКТИЧНА РОБОТА № 4.
ОРГАНІЗАЦІЯ ПІДПРОГРАМ

Мета. Придбання навичок з програмування додатків, що складаються з декількох програмних одиниць. Засвоєння засобів передачі параметрів підпрограмам.

4.1 Поняття та типи підпрограм.

Алгоритмічно завершена частина коду програми, яка може використовуватись неодноразово в залежності від вхідних даних, може бути оформлена як самостійна програмна одиниця – підпрограма (процедура). В залежності від властивостей підпрограма може бути локальною у межах тільки одного додатку, або глобальною, доступною для використання у різних додатках. Глобальні процедури, як правило, організовані у бібліотеки - сукупність процедур, об’єднаних за функціональним призначенням [2].

При використанні підпрограми треба враховувати:

- можливість використання на обраній мові програмування;
- тип виклику;
- засоби активізації підпрограми;
- засоби передачі параметрів;
- правила завершення роботи з підпрограмою.

По типу виклику процедури поділяються на підпрограми прямого або зворотного виклику. У подальшому будуть розглядатися процедури прямого виклику.

Для оформлення фрагменту коду як процедуру використовують директиви опису заголовку та кінця процедури [2]. В загальному випадку формат заголовку наступний:

Name_fun PROC

де, **Name_fun** - ім’я процедури, **PROC** - назва директиви.

Закінчується процедура директивою формату **Name_fun ENDP,**

де **Name_fun** - ім’я процедури. [2].

4.2 Команди активізації та завершення процедур.

Активізувати процедуру означає передати їй управління на виконання. З цього приводу можуть використовуватись такі команди:

- команда безумовного (найчастіше) або умовного переходу;
- команда виклику процедури;
- команда виклику обробника переривання. [2].

Для передачі управління процедурі прямого виклику використовується команда наступного формату:

CALL op1

де, у якості операнду може бути вказано ім’я процедури, що викликається, або її адреса [2].

При роботі з процедурами активно використовується програмний сегмент стеку, через який можуть передаватися параметри процедурі, та обов’язково адреса повернення в викликаючу програму [2].

Дії, що відбуваються при виконанні команди **CALL:**

- запам’ятовується в стеку,згідно регістрам SS: ESP, адреса повернення;
- регістри CS та EIP встановлюються у відповідності до операнду op1, тим самим забезпечується передача управляння на виконання за вказаною адресою [2].

Повернення до викликаючої програми відбувається по команді **RET [n]**, що присутня у підпрограмі. Дії, що виконуються по даній команді:

- згідно регістрам SS: ESP зі стеку зчитується вміст, що використовується для встановлення регістру EIP і, якщо у команді відсутній параметр, управління передається в викликаючу програму;
- при наявності у команді параметру **n** ESP збільшується на вказану кількість байт.

Оскільки для додатків Win32 використовується плоска модель пам’яті, то при використання команд **CALL** в стек запам’ятовується вміст регістру EIP, а по команді **RET** зчитуються 4-ри байти, та заносяться в EIP [2].

Якщо процедура являє собою обробник переривання, то для її виклику використовується команда **INT n,** де **n –** номер обробника переривання, повернення з якого відбувається командою **IRET.** Команда **INT** виконує дії аналогічніі команді **CALL,** але спочатку до стека заноситься регістр прапорів. Команда **IRET** вибирає зі стеку адресу повернення та регістр прапорів [2].

Якщо передача управління процедурі відбувається через команди переходу, а повернення через команду **RET,** то за наявність в стеку адреси повернення повинна потурбуватися викликаюча програма [2].

Слід зазначити, що команда **RET** «не знає» про адресу повернення і буде вибирати зі стеку 4-ри байти, на які вказуватиме регістр ESP. Тому, якщо на момент повернення з процедури регістр ESP буде позначати щось інше, а не адресу повернення, команда **RET** виконається без порушень, але повернення в викликаючу програму не відбудеться, більш того, подальша поведінка додатку – непрогнозуєма [2].

4.3. Засоби передачі параметрів.

Виконання будь якої підпрограми потребує вхідних даних. При використанні процедур на Асемблері вхідні (фактичні) параметри можуть передаватися наступним чином [2]:

- через регістри;
- через стек;
- через загальну пам’ять.

При цьому припустимо будь яке поєднання цих засобів. Передача параметрів через загальну пам’ять означає, що викликаюча програма і всі підпрограми мають доступ до одних і тих же сегментів даних. Використання першого або другого засобу передачі параметрів означає, що між викликаючою програмою і підпрограмами існує певна домовленість для розуміння дій кожної з сторін. Перший засіб самий швидкий, але має обмеження на кількість передаваємих параметрів. Другий засіб – найбільш загальний, але уступає першому за швидкістю та витратою пам’яті. Обирання того чи іншого засобу повинно враховувати вимоги, що висуваються при розробці додатку [2].

Передача параметрів завжди передує виклику підпрограми. Тобто, при використанні першого засобу до виконання команди **CALL** у обрані регістри повинні бути занесені вхідні параметри, що будуть оброблятися в підпрограмі. При використання другого засобу до команди **CALL** в стек необхідно послідовно занести всі фактичні параметри, які в підпрограмі будуть зчитуватись зі стеку для подальшої обробки. Для роботи зі стеком використовуються спеціальні команди. Запис в стек відбувається командою **PUSH op1,** де op1 – 32-розрядний операнд будь якого типу (регістр, пам'ять, константа) [2].

Виконання команди **PUSH:**

- вміст регістру ESP зменшується на 4 ESP:=ESP-4;
- за новим значенням ESP у стек виконується запис операнду op1.

Для читання зі стеку використовується команда **POP op1**, вимоги до op1 такі, як і у команді **PUSH** [2].

Виконання команди **POP:**

- за значенням регістру ESP вибирається зі стеку 4-ри байти та записуються до операнду op1;
- вміст регістру ESP збільшується на 4 ESP:=ESP+4. [2].

4.4 Види фактичних параметрів

Параметри процедурі можна передавати як по значенню так і за адресою. У першому випадку процедура отримує копію параметра і таким чином будь які зміни параметру не позначаються на подальшої роботі викликаючої програми, оскільки зміни параметру були локальні тільки для підпрограми. Тобто, якщо процедура обробляє фактичний параметр в режимі читання, то такий параметр може бути переданим за значенням. Якщо ж фактичний параметр підлягає модифікації у процедурі, його треба передавати за адресою. Крім цього за адресою передаються масиви даних.

Приклад 1. Оформити як процедуру пошук максимального елементу масиву (див. Лабораторна робота № 3). Вхідні та вихідні дані передати через регістри [2].

```
.586
.MODEL FLAT, stdcall
option casemap: none
```

```
        .stack
        .data?
max  dd ?                           ; для збереження максимального елементу

        .data
buf  dd  -10,4,-2,6,8,-1,3,12,-7,5  ; опис ініціалізованого масиву
nx dd 10                                    ; кількість елементів масиву
pos_max  dd -1              ;       ; для збереження зсуву максимального елементу
        .code
start:                              ; початок програми
; ------------------------------------------------------------
…….
   mov ecx, nx                  ; кількість елементів масиву
   lea ebx, buf         ; в ebx- адреса масиву
   call pr_max          ; виклик процедури пошуку макс. Значення
   mov max, eax                 ; максимальне значення повертається в eax
   mov pos_max, edi  ; номер максимального елементу повертається в edi
……                      ; продовження роботи з масивом
   ret                  ; повернення з головної програми
;-------------------------------------------------------------------------------
pr_max          proc            ; заголовок підпрограми
;
; Вхідні параметри передані через регістри ecx, ebx
; Вихідні будуть повернені через eax, edi
; Регістри edx, ecx, esi - змінюються у підпрограмі, тому їх треба зберегти у стекі.

   push edx
   push ecx
   push esi
   xor esi,esi          ; esi :=0
   xor edi,edi          ; edi :=0
   mov edx, [ebx+esi*4]     ; в edx перший елемент масиву
   mov eax, edx                 ; в  eax перший елемент масиву
c1:  mov edx, [ebx+esi*4]    ;  в eax поточний елемент масиву
   cmp edx, eax                 ; порівняння елементу масиву з максимальним
    jle   m1                    ;  перехід, якщо елемент масиву менший або дорівнює
                        ; максимальному
   mov eax, edx;                ; оновити максимальне значення
  mov edi, esi          ; запам’ятати номер максимального элементу
m1: inc esi                     ; індекс наступного елементу масиву
   loop  c1                     ; якщо не всі елементи оброблені перейти на C1
   pop esi                      ; відновлення esi
   pop ecx                      ; відновлення ecx
   pop edi                      ; відновлення edi
   ret                  ; повернення з підпрограми
pr_max          endp            ; завершення підпрограми
end start                       ; завершення головної програми
```

Приклад 2. Оформити як процедуру пошук максимального елементу масиву Вхідні та вихідні дані передати через стек [2].

```
.586
.MODEL FLAT, stdcall
option casemap: none

        .stack
        .data?
max  dd ?                               ; для збереження максимального елементу

        .data
buf  dd  -10,4,-2,6,8,-1,3,12,-7,5      ; опис ініціалізованого масиву
nx dd 10                                        ; кількість елементів масиву
pos_max  dd -1                  ;       ; для збереження зсуву максимального елементу
        .code
start:                                  ; початок програми
; ------------------------------------------------------------
…….
   lea ebx, buf          ; в ebx- адреса масиву
   push nx                      ; кількість елементів масиву – в стек
   push ebx                     ; адреса масиву        - в стек
   lea ebx, pos_max
   push ebx                     ; адреса pos_max - в стек.
   call pr_max          ; виклик процедури пошуку макс. Значення
……..
   mov esi, pos_max  ; номер максимального елементу
   mov eax, buf[esi*4]
   mov max, eax                 ; максимальне значення
……                      ; продовження роботи
    ret                 ; повернення з головної програми
;-------------------------------------------------------------------------------
pr_max          proc            ; заголовок підпрограми
        push esi    ; при вході в підпрограму регістр ESP вказує на адресу повернення
        push edi        ; кожна з вказаних команд push зменшує ESP на 4.
        push eax
        push ebx
        push ecx
        push edx

; Вибірка параметрів із стеку з урахуванням виконаних вище команд push.

   mov edi, dword ptr [ esp+28]     ; в edi - адреса pos_max
   mov ebx, dword ptr [ esp+32]     ; в ebx – адреса масиву
   mov ecx, dword ptr [ esp+36]     ; в ecx – кількість елементів масиву

         xor esi,esi            ; esi :=0
```

```
   mov edx, [ebx+esi*4]       ; в edx перший елемент масиву
   mov eax, edx               ; в  eax перший елемент масиву
c1:  mov edx, [ebx+esi*4]     ;  в eax поточний елемент масиву
   cmp edx, eax               ; порівняння елементу масиву з максимальним
    jle   m1                  ;  перехід, якщо елемент масиву менший або дорівнює
                          ; максимальному
   mov eax, edx;              ; оновити максимальне значення
  mov [edi], esi              ; запам'ятати номер максимального элементу в pos_max
m1: inc esi                   ; наступний елемент масиву
   loop  c1                   ; якщо не всі елементи оброблені перейти на C1
         pop edx
         pop ecx
         pop ebx
         pop eax
         pop edi
         pop esi
..... ret 12                  ; з урахуванням 3-х вхідних параметрів по 4 байти кожний
pr_max          endp
end start
```

4.5 Завдання

Для реалізації завдання створити не менш 3-х процедур (одна головна та дві додаткових), використовуючи різні засоби передачі параметрів, а саме – регістри та стек. В завданнях використовуються знакові 32-х розрядні цілочислові дані [2].

1. Перший елемент масиву замінити сумою парних елементів, а останній - сумою непарних елементів масиву.
2. Визначити середнє арифметичне парних елементів масиву, розташованих на парних позиціях.
3. Обміняти в масиві перший негативний елемент з останнім позитивним.
4. Якщо кількість 0-их елементів масиву – непарна,видалити з масиву кожен другий нульовий елемент.
5. Якщо масив має 2 або більше нульових елементів, то перший і останній нульові елементи замінити різницею максимального і мінімального елементів.
6. Видалити перший парний елемент зі складу масиву.
7. Перший елемент масиву обміняти з максимальним парним елементом.
8. Останній елемент масиву обміняти з першим мінімальним непарним елементом.

9. Обнулити елементи, які найбільше відрізняється від максимального по модулю елемента масиву
10. За один прохід масиву знайти два елементи, що мають найбільші за модулем значення.
11. Останній елемент масиву обміняти місцями з останнім максимальним негативним елементом.
12. Видалити зі складу масиву перший та останній позитивний елемент.
13. Видалити зі складу масиву останні два парні елементи.
14. Якщо сума елементів масиву парна, зрушити всі елементи циклічно на одну позицію ліворуч, в іншому випадку - праворуч.
15. Переставити кожну пару сусідніх елементів (1 та 2, 3 та 4 ...) між собою. Визначити, як при цьому змінилася кількість пар, в яких перший елемент менше другого.
16. Видалити з масиву максимальні непарні елементи.
17. Якщо в масиві максимальні від'ємні елементи є парними, то обнулити їх.
18. Знайти останню серію (більше 3 елементів поспіль) парних елементів.
19. Перший негативний обміняти місцями з останнім парним елементом масиву.
20. Видалити другий нульовий елемент масиву, якщо нульовий елемент один обміняти його з першим елементом масиву.
21. Замінити всі негативні елементи на множення їх сусідів (сусідами першого елемента є останній та другий, останнього – передостанній та перший). Визначити, як змінилася кількість негативних елементів.
22. У масиві є один нульовий елемент. Всі елементи, які знаходяться після нього, переставити в зворотному порядку.
23. Обміняти місцями максимальний негативний з першим негативним, якщо це один і той же елемент, обнулити його.
24. До всіх непарних елементів додати 5 і визначити, як змінилася кількість позитивних елементів в масиві.
25. Видалити мінімальні парні елементи масиву.
26. Обміняти останній елемент масиву з першим максимальним парним.
27. Видалити з масиву мінімальні позитивні елементи.

28. Обміняти в масиві перший максимальний парний елемент з першим максимальним непарним.
29. Елементи масиву, позиційно розташовані після першого парного елемента, розташувати в зворотному порядку.
30. Підрахувати суму елементів, розташованих позиційно між останнім максимальним парним і останнім негативним елементами.
31. Визначити елемент масиву, який відрізняється від сусіднього, розташованого праворуч, на мінімальне значення (сусідом останнього вважати перший).
32. Видалити з масиву максимальні і мінімальні негативні елементи.

4.6 Порядок виконання

1. Визначити початкові вхідні параметри.
2. Розробити алгоритм програми.
3. Аналітично отримати результат.
4. Визначитись з командами мови програмування.
5. Скласти програму.
6. Отримати виконавчий файл.
7. Завантажити виконавчий файл засобами відладчика OLLYDBG.
8. Виконати програму.
9. Порівняти отримані результати з результатами п.3.
10. Протестувати програму з різними вхідними параметрами

4.7 Структура звіту

Звіт оформлюється на окремих аркушах формату А4 та має наступний зміст:

1. Титул.
2. Постанова завдання.
3. Текст програми.
4. Результати роботи.
5. Висновки

4.8 Питання до захисту

До захисту лабораторної роботи допускаються студенти, якщо вони:

- довели працездатність розробленого ПЗ;
- довели достовірність отриманих результатів;
- готові продемонструвати свою спроможність виконати модифікацію програмного коду за вимогою викладача;
- готові захищати свої рішення;
- оформили належним чином звіт.

Питання:

1. Поняття процедури.
2. Директиви оформлення процедури.
3. Засоби активізації процедур.
4. Засоби передачі параметрів.
5. Особливості виконання команди CALL.
6. Особливості виконання команди RET.
7. Команди роботи зі стеком.
8. Особливості виконання команди PUSH.
9. Особливості виконання команди POP.
10. Особливості передачі параметрів за значенням та адресою.
11. Література: [2], [3], [5], [7], [10], [11], [12].

ПРАКТИЧНА РОБОТА №5.
ОРГАНІЗАЦІЯ ВВОДУ ВИВЕДЕННЯ

Мета. Придбання навичок з програмування консольних додатків, що потребують вводу вхідних даних та виводу результатів. Засвоєння засобів використання API- функцій Windows.

5.1 Інтерфейс прикладного програмування

Програмування в Windows ґрунтується на використанні інтерфейсу прикладного програмування (Application Program Interface - API). Він надає програмісту набір готових класів, функцій, структур даних та констант. API-функції забезпечують взаємодію додатка з зовнішніми пристроями та ресурсами операційної системи. API-функції розташовані в системних DLL- бібліотеках [2]:

- kernel32.dll - для взаємодії з ОС;
- user32.dll - користувацький інтерфейс;
- gdi32.dll – підтримка графіки.

Бібліотека kernel32.dll призначена для роботи з об’єктами ядра ОС, її функції дозволяють керувати пам’яттю та іншими ресурсами системи. Бібліотека user32.lib відповідає за управління вікнами, обробку повідомлень, роботу з таймерами тощо. Бібліотека gdi32.dll забезпечує графічний інтерфейс. До складу бібліотеки входять функції виводу на екран, принтер, функції по роботі зі шрифтами та інші [2].

При виклику функцій Win32 API можна використовувати один із двох наборів символів. Це набір 8-розрядних символів ASCII/ANSI і набір 16-розрядних символів Unicode. Функції Windows API, що працюють із текстом, зазвичай мають дві версії: одна із них призначена для роботи з 8-розрядними символами ASCII/ANSI (імена закінчуються символом A), а інша – для роботи з розширеним набором символів, включаючи Unicode (імена закінчуються символом W). Набір символів Unicode є вбудованим для сучасних Windows систем. Тому коли викликається функція, що закінчується на символ A, наприклад WriteConsoleA (відображення інформації на консоль), операційна система спочатку перетворює символи з набору ANSI в Unicode, а потім викликає функцію WriteConsoleW [2].

В додатках Win32 при зверненні до API-функцій використовується погодження про виклик stdcall, що може бути вказано як глобальний параметр в директиві MODEL або локально при виклику конкретної функції [2].

У відповідності до цього погодження параметри функції передаються через стек в зворотному порядку (по відношенню до вказаного при виклику). Функція, що викликається виконує « очистку» стеку, тобто закінчується командою RET N, де N – кількість байт, на які збільшується значення регістру ESP при умові, що функції були передані параметри. Значення, що повертається передається через регістр EAX - 32-ох розрядне ціле або EDX:EAX – 64-ох розрядне ціле.

5.2 Типи додатків Windows

Windows підтримує два типи додатків: графічні та консольні. У додатках першого типу зовнішній інтерфейс – графічний. Ці додатки створюють вікна, меню, взаємодіють з користувачем через діалогові вікна. Консольні додатки виконуються в текстовому режимі, хоча на екрані також розміщуються у вікні – консолі. Слід зазначити, що межа між цими типами додатків дещо умовна, бо консольні програми здатні створювати вікна, в той же час як графічні – виводити текстові рядки у консольне вікно, відправляючи до нього, наприклад, відладочну інформацію, що пов’язана з виконанням програми. Тип додатку вказується у виконавчому файлі, куди він заноситься компонувальником. Коли користувач запускає додаток на виконання, операційна система перевіряє даний параметр і або створює консольне вікно, якщо запускається консольна програма, або ні – для графічної.

Консольні додатки при, необхідності, можуть створювати власну консоль. Консоль складається з одного вхідного і декількох екранних буферів. Вхідний буфер є чергою, кожен запис якої містить інформацію щодо окремої вхідної події консолі [2]:

- натиснення і відпускання клавіш;
- маніпуляцію мишею – рух, натиснення і відпускання кнопок;
- зміну розміру активного екранного буфера, стан прокручування.

Екранний буфер представляє двомірний масив, що містить символи, які виводяться у вікно консолі, і дані про їх колір [2].

В Windows передбачені три стандартні пристрої - для вводу, виводу та виводу помилок. Доступ до них відбувається через дескриптори типу HANDLE, для отримання яких призначені спеціальні функції. Дескриптор є 32-розрядним цілим числом без знаку, що унікально ідентифікує об’єкт, у якості якого може бути таймер, вікно, файл, вхідні і вихідні пристрої та інше. Дескриптор використовується зазвичай при роботі через деякий інтерфейс (API), причому сенс значення дескриптора прихований за цим інтерфейсом [2].

5.3 Функції по роботі з консоллю

5.3.1 Створення та звільнення консолі

Створення власної консолі додатку відбувається через наступну функцію:

BOOL WINAPI AllocConsole(void)

По завершенню роботи додатка всі створені ним консолі автоматично знищуються, але доречним це роботи самостійно, коли консоль стає непотрібною. Для цього призначена функція [2]:

BOOL WINAPI FreeConsole(void)

Обидві функції повертають 0 – у разі помилкового завершення.

5*.3.2 Отримання дескриптора пристрою та установка заголовку консолі*

Для отримання дескриптора стандартного пристрою використовується наступна функція [2]:

HANDLE WINAPI GetStdHandle(DWORD nStdHandle)

де вхідний параметр nStdHandle може приймати наступні значення:

- STD_INPUT_HANDLE - пристрій вводу;
- STD_OUTPUT_HANDLE - пристрій виводу;
- STD_ERROR_HANDLE - пристрій для виводу помилок.

Функція повертає дескриптор пристрою у відповідності до вхідного параметру [2].

Для установки заголовку консольного вікна використовується функція [2]:

BOOL WINAPI SetConsoleTitle(LPCTSTR lpConsoleTitle)

де вхідний параметр lpConsoleTitle вказівник на рядок заголовку.

5.3.3 Функції вводу виводу

Для виводу текстової інформації на консоль використовується функція [2]:

BOOL WINAPI WriteConsole(HANDLE ConsoleOutput, const VOID* lpBuffer, DWORD nNumberOfCharsToWrite, LPDWORD lpNumberOfCharsWritten, LPVOID lpReserved)

де:

- ConsoleOutput – дескриптор буферу виводу консолі, який був отриманий функцією GetStdHandle;
- lpBuffer – вказівник на буфер, що містить текст виводу;
- nNumberOfCharsToWrite – кількість символів виводу;
- lpNumberOfCharsWritten – вказівник на змінну, куди буде записано фактична кількість виведених символів;
- lpReserved – резервний параметр (повинен бути NULL)

Для читання рядка з буферу консолі (тобто для вводу даних), використовують функцію [2]:

BOOL WINAPI ReadConsole(HANDLE ConsoleInput, LPVOID lpBuffer, DWORD NumberOfCharsToRead, LPDWORD lpNumberOfCharsRead, LPVOID InputControl)

де:

- ConsoleInput – дескриптор буферу вводу консолі, який було отримано функцією GetStdHandle;
- lpBuffer – вказівник на буфер для вводу даних
- NumberOfCharsToRead – кількість символів вводу;
- lpNumberOfCharsRead – вказівник на змінну куди буде записано фактична кількість введених символів;
- InputControl – резервний параметр (повинен бути NULL)

Функції повертають 0 – у разі помилки, якщо операція виконана, то параметр lpNumberOfCharsRead / lpNumberOfCharsWritten містить кількість фактично зчитаних або записаних символів. Порівняння цих значень з NumberOfCharsToRead / nNumberOfCharsToWrite дозволяє зробити висновок щодо якості виконання операції [2].

5.3.4. Функції по встановленню розміру консолі, положення курсору та атрибутів символів

Для визначення розміру консольного вікна використовується функція [2]:

```
BOOL WINAPI SetConsoleScreenBufferSize( HANDLE ConsoleOutput,
                                        COORD dwSize)
```

де: ConsoleOutput – дескриптор буферу виводу консолі;
dwSize – структура, що задає розмір консолі:

```
COORD STRUC
    X DW ?
    Y DW ?
COORD ENDS
```

Для позиціювання курсору використовується функція [2]:

```
BOOL WINAPI SetConsoleCursorPosition( HANDLE ConsoleOutput,
                                      COORD dwCursorPosition)
```

де: ConsoleOutput – дексриптор буферу виводу;
dwCursorPosition – структура координат COORD, що визначає позицію курсору.

Для установки атрибутів символів, що виводяться на консоль використовують функцію [2]:

```
BOOL WINAPI SetConsoleTextAttribute(HANDLE hConsoleOutput,
                                    WORD wAttributes)
```

де:

- ConsoleOutput – дескриптор буферу виводу;
- wAttributes – атрибути символів, що визначаються комбінацією констант:

```
FOREGROUND_BLUE          equ 1h          синій
FOREGROUND_GREEN         equ 2h          зелений
FOREGROUND_RED           equ 4h          червоний
FOREGROUND_INTENSITY     equ 8h          насичений
BACKGROUND_BLUE          equ 10h     синій фон букв
BACKGROUND_GREEN   equ 20h     зелений фон букв
BACKGROUND_RED           equ 40h     червоний фон букв
BACKGROUND_INTENSITY     equ 80h     насичений фон букв
```

Якщо потрібно змінити атрибути визначеної кількості символів, починаючи з деякої координати екрану, можна скористатися функцією [2]:

```
BOOL WINAPI FillConsoleOutputAttribute(HANDLE ConsoleOutput,
        WORD wAttribute, DWORD nLength, COORD
    dwWriteCoord,
        LPDWORD lpNumberOfAttrsWritten);
```

де:

- ConsoleOutput – дескриптор буферу виводу ;
- wAttribute – атрибут кольору символу и його фона в консолі;
- nLength – кількість символів, які встановлюються в визначений колір;
- dwWriteCoord – координати першого символу ;
- lpNumberOfAttrsWritten – вказівник на змінну, куди буде записано дійсна кількість символів зазначеного кольору.

5.4. Приклад оформлення програми з використанням функцій по роботі з консоллю.

Для створення консольного додатку засобами MASM32 необхідно внести зміни у bat – файли, що розташовані за шляхом \masm32\bin, додавши для компанувальника (LINK) опцію /SUBSYSTEM:CONSOLE, або скористатися відповідними пунктами вкладки «Project» програми qeditor [2].

Визначення прототипів функцій консолі може бути здійснено як безпосередньо скориставшись директивою PROTO, так і через використання директиви INCLUDE, вказавши імена відповідних файлів. Визначення бібліотек, що необхідно підключити, відбувається через директиву INCLUDELIB [2].

Приклад. Розробити програму, що виконує обробку масиву у відповідності до завдання. Вимоги до розробки: максимальна кількість елементів масиву - 100. Елементи масиву та їх фактична кількість задається зі стандартного пристрою вводу. Вивід результатів обробки масиву здійснюється на стандартний пристрій виводу – спочатку відображається первинний масив потім оброблений масив. Обидва масиву виводяться у вигляді матриці. Додаткові вихідні та вихідні данні супроводжуються коментарем [2].

```
.586
.model  flat,  stdcall

option  casemap : none

; Підключення файлів з константами та прототипами функцій

include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
include \masm32\include\user32.inc
include \masm32\include\msvcrt.inc

; Підключення файлів бібліотек

includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\user32.lib
includelib \masm32\lib\msvcrt.lib

.STACK

.DATA
ConsoleTitle   db       " Лабораторна робота №5 ",0  ; заголовок вікна консолі
Name_Title      db 30 dup(0)
ComSizeMas          db      " Введіть розмір масиву ",13,10
Len_ComSize         equ     $- ComSizeMas ; різниця між адресами змінних є розмір    ;
                    попереднього поля  ComSizeMas
ComElemMas          db      " Введіть елементи масиву ",13,10
Len_ComElem         equ     $- ComElemMas
ComMasBefore        db      " Масив до обробки ",13,10
```

```
Len_MasBefore       equ     $- ComMasBefore
ComMasAfter         db      " Масив після обробки ",13,10
Len_MasAfter        equ     $- ComMasAfter
format_size_buf     db      "%d",0
format_print_buf db  " %5d ",0
print_buf_b         db      "Buf [ ",0
print_buf_e         db      "]=",0
print_13_10    db  13,10,0
print_13         db  13,0


.DATA?
Buf             dd      100 dup(?)  ; максимальний обсяг масиву
Size_buf        dd      ?       ; фактичний розмір масиву
h_input         dd      ?       ;  дескриптор пристрою вводу
h_output        dd      ?       ;  дескриптор пристрою виводу
nWrite          dd      ?       ; змінна для функції WriteConsole
number_element      dd      ?   ; поточний номер елементу вводу
element_buf   dd      ?       ;  змінна для вводу елементу масиву
.CODE
main  proc

 call     AllocConsole   ;  створення власної консолі

;  перетворення символів в формат Oem для виводу заголовку

invoke  CharToOemA, addr ConsoleTitle, addr Name_Title

invoke                  SetConsoleTitle,  addr Name_Title    ; вивід заголовку
invoke          GetStdHandle,  STD_INPUT_HANDLE
mov h_input, eax                ; отримали дескриптор пристрою вводу
invoke          GetStdHandle,  STD_OUTPUT_HANDLE
mov h_output, eax         ; отримали дескриптор пристрою виводу


 invoke SetConsoleOutputCP,1251             ; підтримка кирилиці
 invoke SetConsoleCP,1251

call input_mas   ; Ввід початкових даних
call work_mas    ;  Обробка масиву
call print_rezult  ;  Вивід результатів

 invoke Sleep,50000 ; Затримка зображення на екрані

call FreeConsole
invoke ExitProcess, 0  ; Завершення програми

main endp
```

```
;-------------------------------------------------------------------------------------------------

input_mas  proc

;   Запрошення для вводу розміру масиву

invoke          WriteConsole, h_output, ADDR ComSizeMas, Len_ComSize, ADDR nWrite, 0

invoke   crt_scanf, ADDR format_size_buf, ADDR Size_buf   ; форматний ввід

;  !!!!  Виконати перевірку    1<= Size_buf<=100 у разі помилки – повторити ввід

mov number_element, 0
m_input_buf:

invoke WriteConsole, h_output, ADDR print_buf_b, 7, ADDR nWrite, 0   ; Вивід buf [

invoke   crt_printf, ADDR format_size_buf, number_element        ; вивід № елементу

  invoke WriteConsole, h_output, ADDR print_buf_e, 2, ADDR nWrite, 0  ; Вивід ]=

 invoke   crt_scanf, ADDR format_size_buf, ADDR element_buf  ; Ввід елем.масив.
or eax,eax
  jnz m_1   ; немає помилки

; помилка вводу - очистка буферу вводу

 m_2: invoke crt_getchar
   cmp eax,10  ; доки не enter
   jne m_2
  invoke   crt_printf,  ADDR print_13
   jmp m_input_buf
m_1:
  mov eax,  element_buf
  mov esi,  number_element
  shl esi,2
  mov Buf [esi], eax
  inc number_element
  mov edx,  number_element
  cmp edx,  Size_buf
  jnz m_input_buf   ;  цикл без використання команди  LOOP

  ret
input_mas  endp
;-------------------------------------------------------------------------------------------------

work_mas proc    ;  Основний алгоритм, реалізується декількома процедурами
```

```
  ret
work_mas endp
;-------------------------------------------------------------------------------------------------

print_rezult proc   ; Вивід результатів

 invoke WriteConsole, h_output, ADDR ComMasBefore, Len_MasBefore, ADDR nWrite, 0

  mov number_element,0
m1:  mov ecx,5                  ;   вивід по 5 елементі в рядок
m2:  push ecx

  mov eax,number_element
  shl eax,2
  mov esi,eax

  invoke   crt_printf,  ADDR format_print_buf , Buf[esi]
  pop ecx
  inc number_element
   mov edx, number_element
  cmp edx, Size_buf
  jz m_ret                ;    всі елементи виведені
    dec ecx
   jnz m2
   invoke   crt_printf,  ADDR print_13_10    ; перехід на новий рядок
   jmp m1
m_ret: ret
print_rezult  endp
;-------------------------------------------------------------------------------------------------
   end main
```

5.5 Завдання.

Розробити консольний додаток, що виконує обробку масиву у відповідності до завдання. Вимоги до розробки: максимальна кількість елементів масиву -100. Елементи масиву та їх фактична кількість задається зі стандартного пристрою вводу. Вивід результатів обробки масиву здійснюється на стандартний пристрій виводу – спочатку відображається первинний масив потім оброблений масив. Обидва масиви виводяться у вигляді матриці. Додаткові вихідні данні супроводжуються коментарем, консольне вікно повинно мати заголовок, підтримка кирилиці - обов’язкова. Кожна функціональна одиниця алгоритму оформлюється у вигляді окремої процедури, загальна кількість процедур не менш 4-х, використовуючи різні засоби передачі параметрів [2]:

1. Якщо в масиві максимальний від'ємний елемент є парним, то обнулити його.
2. Знайти останню серію (більше 3 елементів поспіль) парних елементів.
3. Перший негативний обміняти місцями з останнім парним елементом масиву.
4. Видалити другий нульовий елемент масиву, якщо нульовий елемент один обміняти його з першим елементом масиву.
5. Замінити всі негативні елементи на множення їх сусідів (сусідом першого елемента є останній та другий, останнього – передостанній та перший і навпаки). Визначити, як змінилася кількість негативних елементів.
6. У масиві є один нульовий елемент. Всі елементи, які знаходяться після нього, переставити в зворотному порядку.
7. Обміняти місцями останній максимальний негативний з першим негативним, якщо це один і той же елемент, обнулити його.
8. До всіх непарних елементів додати 5 і визначити, як змінилася кількість позитивних елементів в масиві.
9. Видалити мінімальні парні елементи масиву.
10. Обміняти останній елемент масиву з першим максимальним парним.
11. Видалити з масиву мінімальні позитивні елементи.
12. Обміняти в масиві останній максимальний парний елемент з останнім максимальним непарним.
13. Елементи масиву, позиційно розташовані після першого парного елемента, розташувати в зворотному порядку.
14. Підрахувати суму елементів, розташованих позиційно між останнім максимальним парним і останнім негативним елементом.
15. Визначити елемент масиву, який відрізняється від сусіднього, розташованого праворуч, на мінімальне значення (сусідом останнього вважати перший).
16. Видалити з масиву максимальний і мінімальний негативні елементи
17. Перший елемент масиву замінити сумою парних елементів, а останній - сумою непарних елементів масиву.
18. Визначити середнє арифметичне парних елементів масиву, розташованих на парних позиціях.
19. Обміняти в масиві перший негативний елемент з останнім позитивним.
20. Видалити з масиву кожен другий нульовий елемент.

21. Якщо масив має 2 або більше нульових елементів, то перший і останній нульові елементи замінити різницею максимального і мінімального елементів.
22. Видалити перший парний елемент зі складу масиву.
23. Перший елемент масиву обміняти з останнім максимальним парним елементом.
24. Останній елемент масиву обміняти з першим мінімальним непарним елементом.
25. Обнулити елементи, що найбільш відрізняються від максимального по модулю елемента масиву
26. За один прохід масиву знайти два елементи, що мають найбільші за модулем значення.
27. Останній елемент масиву обміняти місцями з останнім максимальним негативним елементом.
28. Видалити зі складу масиву перший та останній позитивний елемент.
29. Видалити зі складу масиву останні два парні елементи.
30. Якщо сума елементів масиву парна, зрушити все елементи циклічно на одну позицію ліворуч, в іншому випадку - праворуч.
31. Переставити кожну пару сусідніх елементів (1 та 2, 3 та 4 ...) між собою. Визначити, як при цьому змінилася кількість пар, в яких перший елемент менше другого.
32. Видалити з масиву максимальні непарні елементи.

5.6 Порядок виконання

1. Визначити початкові вхідні параметри
2. Розробити алгоритм програми.
3. Аналітично отримати результат.
4. Визначитись з командами мови програмування.
5. Скласти програму.
6. Отримати виконавчий файл.
7. Виконати програму.
8. Порівняти отримані результати з результатами п.3.
9. Протестувати програму з різними вхідними параметрами.

5.7 Структура звіту

Звіт оформлюється на окремих аркушах формату А4 та має наступний зміст:

1. Титул.
2. Постанова завдання.
3. Текст програми.
4. Результати роботи.
5. Висновки.

5.8 Питання до захисту

До захисту лабораторної роботи допускаються студенти, що:

- довели працездатність розробленого ПЗ;
- довели достовірність отриманих результатів;
- готові продемонструвати свою спроможність виконати модифікацію програмного коду за вимогою викладача;
- готові захищати свої рішення;
- оформили належним чином звіт.

Питання:

1. Інтерфейс прикладного програмування. Призначення.
2. Погодження про виклики.
3. Типи додатків Windows.
4. Поняття консолі та її склад.
5. Стандартні пристрої вводу виводу та поняття дескриптору.
6. Функції створення та звільнення консолі.
7. Функції отримання дескрипторів стандартних пристроїв.
8. Функції читання та запису з/на консоль.
9. Функції управління курсором.
10. Функції встановлення кольорової палітри.
11. Література: [2], [5], [7], [10], [11], [12], [13].

ПРАКТИЧНА РОБОТА №6.
ОРГАНІЗАЦІЯ ВЗАЄМОДІЇ РІЗНОМОВНИХ ПРОГРАМ

Мета. Придбання навичок з програмування додатків, що об'єднують модулі на різних мовах програмування : C++ (C) та Асемблер. Обробка двомірних масивів даних.

6.1 Опис та доступ до елементів двомірних масивів

Для опису двомірних масивів мова Асемблера не має спеціальних директив. Будь-яка послідовність байт може трактуватися програмістом по-різному в залежності від задачі, що вирішується. Двомірні масиви, що найчастіші іменуються матрицями, декларуються як звичайні одномірні масиви. Окремо задаються параметри кількості елементів в рядку та стовпцю. Розглядати одномірний масив як послідовність рядків чи послідовність стовпчиків залежить від програміста, але дотримуючись правил представлення матриць на мовах високого рівня (C++, C) перевага віддається першому варіанту – послідовності рядків [2].

Для доступу до елементів матриці доцільно використовувати пряму адресацію з індексуванням, або адресацію по базі с індексуванням. При чому перший індекс відповідає зміщенню початку рядка, а другий індекс – зміщенню елемента в цьому рядку [2].

Приклад 1. Дана матриця A[4][5]. Програма демонструє організацію доступу до елементів матриці [2]:

```
.586
.model flat, stdcall
option  casemap : none

.data

A  dd 1,2,3,4,5,11,12,13,14,15,21,22,23,24,25,31,32,33,34,35  ; ініціалізація матриці
len_str dd 0              ; довжина рядка в байтах
n_str  dd 4               ; кількість рядків
n_colum  dd 5             ; кількість стовпців

.code
start:
```

```
mov eax,n_colum
shl eax,2
mov len_str,eax   ; довжина рядка в байтах

Xor esi,esi
Mov ecx,n_str                ; кількість рядків
        m2:   Xor edi,edi
              Push ecx
              Mov ecx,  n_colum       ; кількість стовпців = кількість елементів в рядку
        m1:   Mov eax,  A[esi][edi*4]      ; в еах спочатку перший елемент матриці
              Inc edi                      ; потім на кожній ітерації циклу наступний
              Loop m1                      ;  елемент рядка ( цикл по стовпцям)
              add esi,len_str              ; початок наступного рядка

              Pop ecx                      ; кількість оброблених рядків
              Dec ecx
              Jnz m2                       ; цикл по рядкам
 ……
 end start
```

З використанням адресації по базі вибірка елементу матриці відбувається наступним чином:

```
              Lea ebx, A
……
              Add esi, ebx
              …..
              Mov eax, dword ptr [esi][edi*4]
```

Модифікатор dword ptr означає, що з пам’яті, адреса якої вказана другим операндом команди Mov, обирається елемент розміром 4-ри байти [2].

6.2 Особливості програмування модулів мови C++ (C) та Асемблеру при поєднанні їх у межах одного проекту. [2].

Розподіл функцій між різномовними компонентами відбувається наступним чином:

- оформлення консольного вікна - модуль C++ (C);
- ввід вхідних параметрів (кількість рядків та стовпців) та динамічне виділення пам’яті необхідного обсягу - модуль C++ (C);
- ввід елементів матриці – модуль C++ (C);
- обробка матриці у відповідності до завдання – модуль Асемблеру;
- вивід результатів обробки – модуль C++ (C);

- завершення роботи – модуль C++ (C)

Головний модуль C++ (C) повинен містити опис прототипів функцій Асемблеру, к яким буде звернення з програм C++ (C). Формат опису прототипів [2]:

Extern "C" P_r Name ([p1] [,p2]... [, pn])

де: extern - директива, яка визначає, що процедура, опис прототипу якої представлено, знаходить за межами файлу модулю C++ (C);

параметр "C" є вказівкою компілятору C++ не змінювати ім’я вказаної процедури;

P_r Name ([p1] [,p2]... [, pn]) – власне прототип процедури Асемблеру:

де: P_r – тип значення, що повертається процедурою;

Name - ім’я процедури

[p1] [,p2]... [, pn] – типи вхідних параметрів, якщо такі передаються процедурі.

Приклад 2. При зверненні до процедури Асемблеру їй передається 3 параметри – адреса початку матриці, кількість рядків та кількість стовпців. Процедура повертає номер стовпця, що задовольняє якимсь умовам [2].

Опис прототипу:

extern "C" int my_matrix (int *, int, int);

де: my_matrix - ім’я процедури на Асемблері

int (до імені процедури) - процедура повертає параметр вказаного типу.

Перший параметр процедурі my_matrix передається за адресою, тому від буде доступний на читання\запис, інші – за значенням, доступні тільки на читання.

При програмуванні Асемблерних модулів слід враховувати [2]:

1. Узгодження про виклики - «C» (.model flat, C).
2. Параметри процедурі від програми C++ (C) передаються тільки через стек.
3. 32-ох розрядне значення, що повертається процедурою Асемблеру передається в регістрі EAX, 64-х розрядне - в EDX:EAX (в EDX старша частина, в EAX молодша)
4. Згідно узгодження «C» «очистку» стеку виконує процедура, що викликає, тобто процедура Асемблеру, що викликається з C++, повинна завершуватися командою RET незалежно від кількості переданих їй параметрів.
5. Асемблерний модуль, в межах якого може бути декілька процедур повинен завершуватись командою END без вказівки параметру [2].

6.3 Особливості налаштування проекту в Visual Studio

Файли з текстами програм на мовах C++ (C) та Асемблеру можна підготувати в редакторі Visual Studio і в будь якому іншому. Порядок подальших дій [2]:

1. Створити в середовищі Visual Studio порожній проект консольного додатку Win32.
2. Додати (або створити) до проекту файли з програмами на обох мовах.
3. Виконати зміни у властивостях проекту: *«Свойства конфигурации»* - *«Общие»* - *«Набор символів»* встановити в «*Использовать багатобайтовую кодировку»* (Рис.6.1)
4. Відкрити контекстне меню для об'єкту з назвою проекту (правий клік мишки) та обрати команду *«Настройка построения»* та встановити відмітку у рядку *«masm(.targets,.props)»* (Рис.6.2, Рис.6.3)
5. Відкрити контекстне меню для об'єкту з назвою ASM програми та обрати команду *«Свойства»* (Рис.6.4)
6. Змінити властивості для програми ASM: *«Свойства конфигурации»* -*« Общие»* - *«Тип элемента »* на *Microsoft Macro Assembler* (Рис.6.4) [2].
7. Побудувати рішення проекту [2].

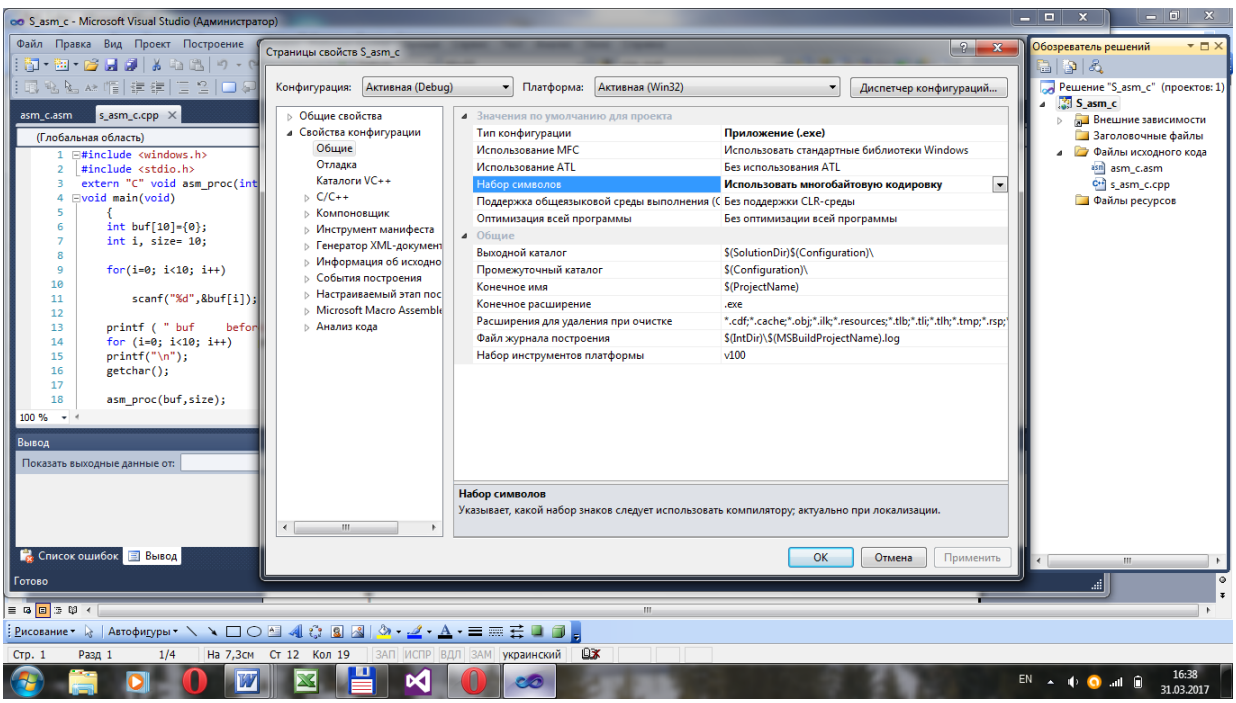

Рис.6.1 Встановлення кодировки [2].

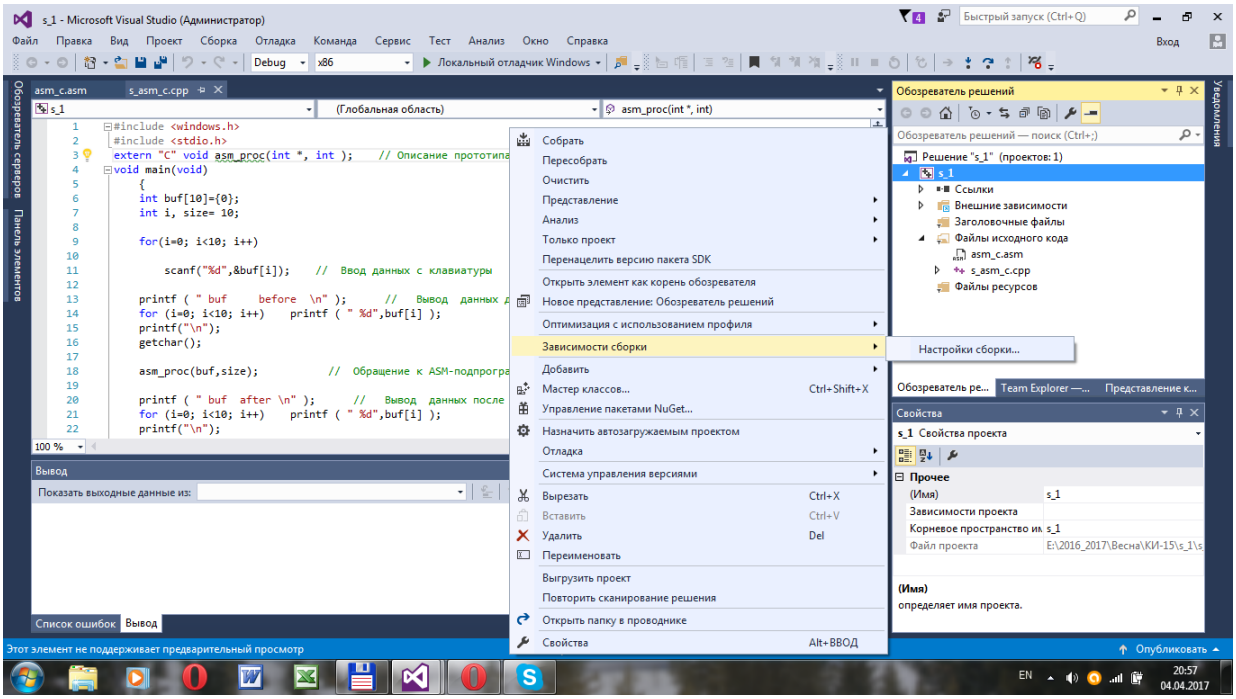

Рис.6.2. Вибір настройок побудови рішення [2].

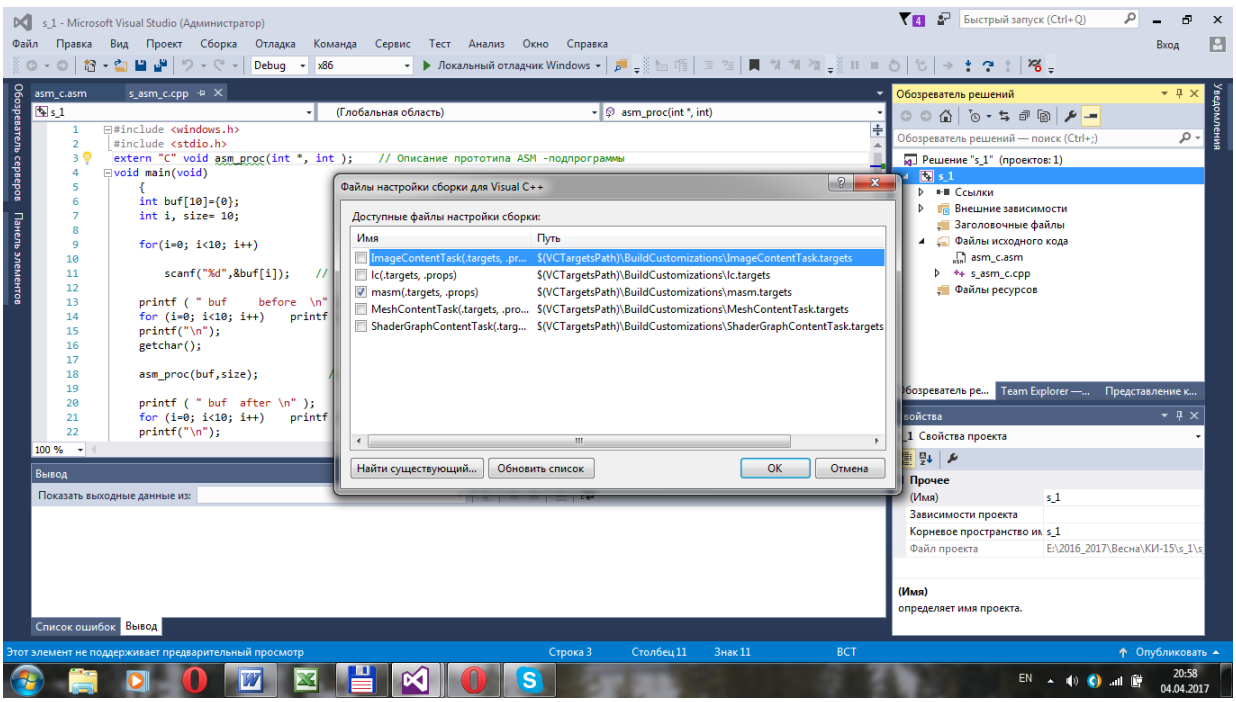

Рис. 6.3. Додавання до настройок рішення [2].

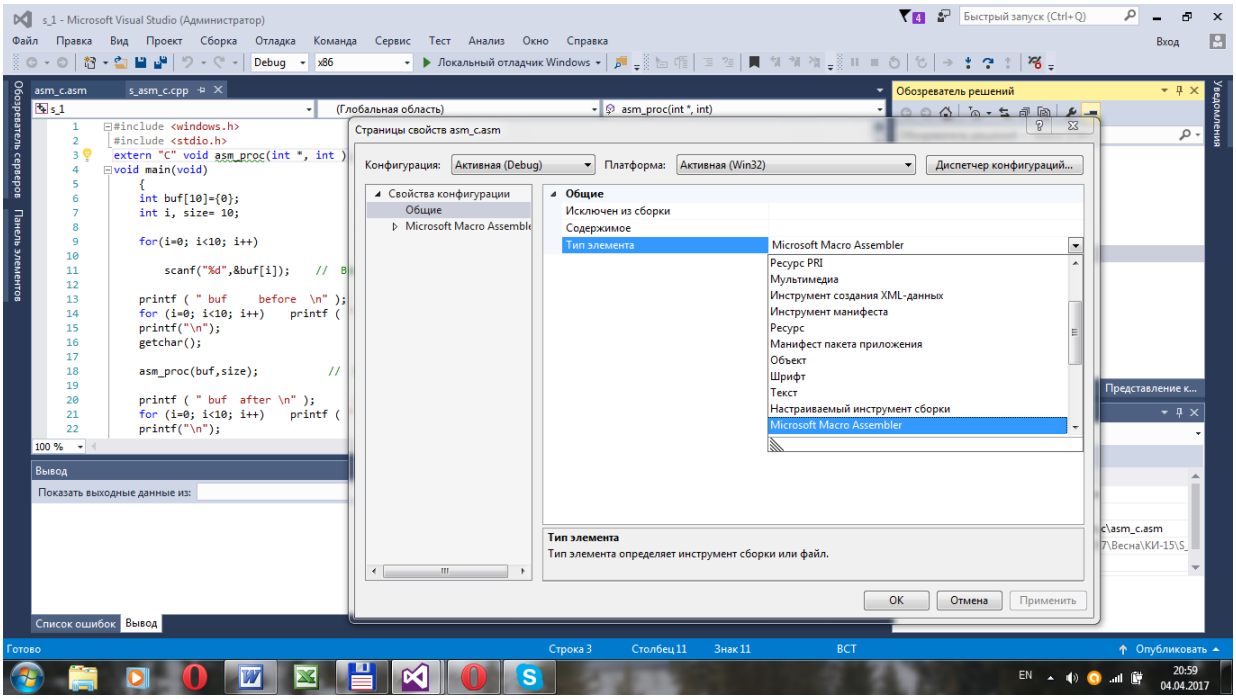

Рис.6.4. Вибір відповідного компілятору [2].

6.4 Завдання [2].

Розробити консольний додаток, що містить різномовні (C++, Асм) програмні модулі, розподіл функцій між якими визначено у п.6.2:

1. В матриці знайти стовпчики з мінімальною кількістю нульових елементів. Вивести номери цих стовбців та кількість нульових елементів.
2. В матриці знайти рядки з максимальною кількістю нульових елементів. Вивести номери цих рядків та кількість нульових елементів.
3. В матриці знайти стовпчики з мінімальною кількістю парних елементів. Вивести номери цих стовпців та кількість парних елементів.
4. В матриці знайти рядки з мінімальною кількістю негативних елементів. Вивести номери цих рядків та кількість негативних елементів.
5. В матриці знайти стовпчики з максимальною кількістю додатних елементів. Вивести номери цих стовпців та кількість додатних елементів.
6. В матриці знайти рядки з однаковою сумою елементів. Вивести номери цих рядків та суму елементів.
7. В матриці знайти стовпчики з однаковою сумою елементів. Вивести номери цих стовпців та суму елементів.
8. В матриці знайти рядки, в яких кількість додатних та негативних елементів однакова. Вивести номери цих рядків та кількість додатних елементів.
9. Якщо сума елементів головної діагоналі дорівнює нулю, переставити елементи цієї діагоналі в зворотному порядку.
10. Якщо суми елементів першого рядка та першого стовпця квадратної матриці однакові, обміняти елементи першого рядка з елементами першого стовпця.
11. Переставити стовпці матриці місцями, рухаючись від крайніх до центрального.
12. Якщо кількість не нульових стовпців матриці більша за кількість нульових, змінити відповідні елементи нульових стовпців на інверсні значення елементів ненульових стовпців так, щоб кількість ненульових та нульових стовпців була однакова.
13. Матриця містить тільки 1 та 0. Сприймаючи вміст стовпця як двійковий код, відсортувати стовпці за зростанням коду, розглядаючи, елементи першого рядка, як старші розряди коду.

14. Матриця містить одиничну (всі елементи дорівнюють 1) підматрицю (прямокутну або квадратну). Вивести координати лівого верхнього куту та розмір під матриці.
15. В матриці всі нульові елементи замінити сумою сусідніх.
16. Виконати транспонування вхідної матриці.
17. Впорядкувати рядки матриці за убуванням сум їх елементів. Вивести підсумкову матрицю та масив сум.
18. Відсортувати стовпці матриці у зростаючому порядку сум їх елементів. Вивести матрицю та масив сум.
19. Якщо всі елементи головної діагоналі дорівнюють 0, обміняти місцями елементи трикутних матриць.
20. Визначити чи є задана матриця діагональною.
21. Якщо елемент на пересіченні головної та побічної діагоналі дорівнює 0, обміняти місцями елементи верхнього та нижнього трикутника матриці.
22. Якщо суми елементів бокових трикутників, що утворені перетином головної та побічної діагоналей, однакові за знаком, обнулити перший елемент матриці.
23. Якщо сума елементів головної діагоналі більша за суму елементів побічної, обміняти елементи цих діагоналей.
24. Обміняти рядки з непарними номерами (1,3; 5,7; ...)
25. Якщо суми елементів першого та останнього рядка однакові, обміняти ці рядки з відповідними стовпцями (першим та останнім).
26. Визначити чи є задана матриця ступінчастою.
27. В матриці знайти стовпчики, всі елементи яких є непарними.. Вивести номери цих стовбців.
28. В матриці переставити рядки, що містять тільки негативні елементи (рядок може бути переставлено тільки один раз).
29. В матриці для всіх стовпчиків з непарними номерами переставити елементи в зворотному порядку.
30. В матриці обміняти місцями непарні рядки з парними стовпцями.
31. Якщо сума елементів першого стовпця негативна, а останнього додатна, обміняти їх місцями, переставляючи елементи в зворотному порядку
32. В матриці обміняти місцями парні рядки з непарними стовпцями.

6.5 Порядок виконання

1. Визначити початкові вхідні параметри.
2. Розробити алгоритм програми.
3. Скласти програму.
4. Отримати результат і виконавчий файл.
5. Виконати програму.
6. Порівняти отримані результати з результатами п.3.
7. Протестувати програму з різними вхідними параметрами

6.6 Структура звіту

Звіт оформлюється на окремих аркушах формату А4 та має наступний зміст:

1. Титул.
2. Постанова завдання.
3. Текст програми.
4. Результати роботи.
5. Висновки.

6.7 Питання до захисту

До захисту лабораторної роботи допускаються студенти, які:

- довели працездатність розробленого ПЗ;
- довели достовірність отриманих результатів;
- готові продемонструвати свою спроможність виконати модифікацію програмного коду за вимогою викладача;
- готові захищати роботу та оформили належним чином звіт.

Питання:

1. Опис багатомірних масивів на Асемблері.
2. Представлення в пам’яті двомірних масивів.
3. Засоби адресації для доступу до елементів рядка матриці.
4. Засоби адресації для доступу до елементів стовпця матриці.
5. Узгодження про виклики «С».
6. Особливості опису прототипу функцій Асемблеру при звернені до них з мови С++
7. Особливості оформлення процедур на Асемблері при зверненні до них з С++(С).
8. Особливості створення проекту, що включає різномовні модулі.
9. Налаштування властивостей проекту у VS.

10. Використання вбудованого відладчика для відстежування виконання.
11. Література: [2], [8], [10], [11], [13].

ПРАКТИЧНА РОБОТА №7.
ОТРИМАННЯ ХАРАКТЕРИСТИК ПРОЦЕСОРУ

Мета. Придбання навичок з використання команди CPUID для визначення різних параметрів процесору.

7.1 Загальні відомості про команду CPUID

Інструкція CPUID надає широкі можливості для визначення різних характеристик процесору від його назви до штатної тактової частоти. На відміну від інших команд, CPUID – багатофункціональна інструкція. Дії, що підлягають виконанню ідентифікуються вхідними параметрами, в якості яких виступають регістри процесору загального використання, найчастіше регістр EAX. Значення вхідного параметру визначає рівень команди CPUID [1]. Розрізнюють наступні рівні [2]:

- стандартний ;
- розширений ;
- спеціальний ;
- недокументований.

Для всіх рівнів підтримується єдиний інтерфейс команди, а саме - спочатку визначається максимальний номер функції, що підтримується процесором для конкретного рівня. Це надає можливість надалі бути впевненим, що при використанні функцій з номерами від 1 до максимального процесор адекватно відреагує на команду CPUID з відповідними вхідними параметрами [2].

Діапазони значень номерів функцій для кожного рівня визначаються як [2]:

- 0x0000 0000 - 0x0000 00** , для стандартного рівня;
- 0x8000 0000 - 0x8000 00** , для розширеного рівня;
- 0x8086 0000 - 0x8086 00** , для спеціального рівня (процесор Transmeta);
- 0xC000 0000 - 0xC000 00** , для спеціального рівня (процесор Centaur);
- 0x8FFF 0000 - 0x8FFF 00** , для не документованого рівня

де, ** може приймати значення від 00 до максимального номеру функції, що підтримується на даному рівні. Зазначимо, що вказані номери спеціального та не документованого рівня характерні тільки для процесорів Intel [5]. Для процесорів інших виробників треба звертатися до відповідної документації (команда CPUID не стандартизована і обов’язково потребує звернення до документації на процесор) [2].

Всі сучасні процесори підтримують команду CPUID, про що свідчить можливість зміни 21-біту регістру EFLAGS [5]. Переконатися в цьому, можна через виконання наступного фрагменту програми [2]:

```
pushfd                  ; занести EFLAGS в стек
pop eax                 ; зчитати EFLAGS в EAX
mov ebx, eax
xor eax, 00200000h      ; переключити 21-біт в 1
push eax                ; занести результат в стек
popfd                   ; зчитати EAX в EFLAGS
pushfd                  ; занести EFLAGS в стека
pop eax                 ; зчитати EFLAGS в EAX
cmp eax, ebx                    ; перевірити змінився 21-й біт чи ні
jz NO_CPUID             ; змін немає, тобто CPUID не підтримується
……                      ; CPUID підтримується
```

7.2 Формат команди та методика використання

Робота с командою CPUID виконується за наступним сценарієм [2]:

1. Обирання необхідного рівня команди в залежності від того, яку інформацію потрібно отримати.
2. Визначення максимального номера функції, що підтримується обраним рівнем.
3. Визначення необхідних функцій обраного рівня.

Перший пункт цієї схеми виконується аналітично на підставі документації за командою. Для виконання другого пункту використовується загальний формат команди, згідно якого вхідний параметр передається через регістр EAX. У якості вхідного параметру для кожного рівня вказується ліва межа діапазону значень функцій кожного рівня [2]:

- 0x0000 0000 - для стандартного рівня;
- 0x8000 0000 - для розширеного рівня;
- 0x8086 0000 - для спеціального рівня (процесор Transmeta);
- 0xC000 0000 - для спеціального рівня (процесор Centaur);
- 0x8FFF 0000 - для не документованого рівня.

Приклад. Визначити максимальний номер функції стандартного та розширеного рівнів [2].

```
…..; Фрагмент коду
    Mov EAX,0          ; для стандартного рівня
    CPUID
    Mov max_num_st, EAX ; отримали максимальний номер функції стандартного
 рівня
……..                   ; продовження роботи з функціями стандартного рівня
    Mov EAX,80000000h     ; розширений рівень
    CPUID
    Xor EAX,80000000h     ; отримали максимальний номер функції розшир. рів-
ня
 Mov max_num_ext, EAX
……; продовження роботи з функціями розширеного рів-
ня
```

Оскільки CPUID являє собою інструкцію процесору, то як вхідні так і вихідні дані можуть відрізнятися для різних виробників [1]. Однак стандартний рівень в більшості вихідних даних підтримується процесорами різних виробників, інші рівні виявляються більш залежними від типу процесору. Тому для отримання будь-яких характеристик процесору спочатку треба визначитись з його виробником. Цю інформацію можна отримати скориставшись командою CPUID с вхідним параметром 0 (EAX=0). Крім максимального значення функції стандартного рівня команда повертає в регістрах EBX, EDX, ECX рядок назви виробника процесору, для Intel це – GenuineIntel. Розподіл символів між регістрами наступний [2]:

- Genu – регістр EBX (символ “G” у молодшому байті EBX);
- ineI - регістр EDX (символ “i” у молодшому байті EDX);
- ntel - регістр ECX (символ “n” у молодшому байті ECX).

Визначившись з виробником, можна далі отримати дані щодо назви процесору. Для цього слід звернутися до функцій розширеного рівня (0x80000002h, 0x80000003h, 0x80000004h). Інформація (Brand String) повертається в регістрах eax, ebx, ecx, edx, при чому крім назви може містити і частоту процесору. Якщо Brand String не підтримується, то перевіряється Brand ID (функція 0x80000001h, вихідний регістр ebx). Не нульове значення Brand ID свідчить про його підтримку. Тоді за таблицями значень Brand ID для відповідного виробника ідентифікується назва (тип) процесору [2].

Документація по команді CPUID для процесорів Intel наведена у Додатку А [5], для процесорів AMD – у Додатку Б [11].

7.3 Завдання [2].

Розробити програму, що визначає параметри процесору згідно номеру варіанта. Завдання містять по два питання, відповіді на які належить оформити окремими функціями з використанням відповідного звернення до команди CPUID [2].

Мова програмування – C\C++, використання команди CPUID оформити «asm» - вставками [2]:

1. Назва виробника, наявність математичного сопроцесору.
2. Модель процесору, підтримка сторінок розміром 4 MB.
3. Сімейство процесору, лічильник міток реального часу.
4. Степінг процесору, підтримка команд rdmsr и wrmsr.
5. Назва процесору, підтримка фізичної адреси більш за 32 біти.
6. Назва виробника, підтримка інструкції cmpxchg8b.
7. Модель процесору, наявність програмно доступного контролера переривань.
8. Сімейство процесору, підтримка інструкцій швидких системних викликів sysenter та sysexit.
9. Степінг процесору, підтримка глобальних сторінок.
10. Назва процесору, підтримка архітектури машинного контролю.
11. Назва виробника, підтримка інструкцій умовної пересилки cmov, fcmovcc, fcomi.
12. Модель процесору, підтримка таблиці атрибутів сторінок.
13. Сімейство процесору, підтримка 36-разрядної фізичної адресації з 4-MB сторінками.
14. Степінг процесору, наявність унікального серійного номеру процесору.
15. Назва процесору, підтримка інструкції CLFLUSH.
16. Наявність датчику температури та можливість температурного моніторингу процесору.
17. Назва виробника, підтримка запису відладочної інформації.
18. Модель процесору, підтримка управління охолодженням процесору за допомогою порожніх циклів.
19. Сімейство процесору, підтримка MMX.

20.Степінг процесору, підтримка інструкцій FXSAVE, FXRSTOR .
21.Назва процесору, підтримка SSE.
22.Назва виробника, підтримка SSE2.
23.Модель процесору, підтримка управління конфліктуючими типами пам’яті.
24. Сімейство процесору, підтримка Hyper-Threading.
25.Степінг процесору, підтримка автоматичного моніторингу температури.
26.Назва процесору, підтримка сигналу припинення.
27.Назву процесору та кількість біт для фізичної адреси даних.
28.Для процесорів виробника Intel визначити значення базової, та максимальної частоти.
29.Для процесорів виробника AMD визначити наявність апаратної підтримки виконання на одному ядрі процесору більш за один програмний потік.
30.Для процесорів виробника Intel визначити наявність у процесорі розширеного програмованого контролеру переривань.
31.Для процесорів виробника Intel визначити можливість отримання інформації про різні типи частоти процесору.
32.Назва процесору, серійний номер.

7.4 Порядок виконання

1. Визначити рівень(ні) команди CPUID.
2. Визначити номери функцій кожного рівня.
3. Визначитись з командами мови програмування.
4. Скласти програму.
5. Отримати виконавчий файл.
6. Виконати програму.
7. Порівняти отримані результати з результатами відповідних програм сторонніх виробників.
8. Протестувати програму на ПК з різними процесорами.

7.5 Структура звіту

Звіт оформлюється на окремих аркушах формату А4 та має наступний зміст:

1. Титул.
2. Постанова завдання.

3. Текст програми.
4. Результати роботи.
5. Висновки.

7.6 Питання до захисту

До захисту лабораторної роботи допускаються студенти, які:

- довели працездатність розробленого ПЗ;
- довели достовірність отриманих результатів;
- готові продемонструвати свою спроможність виконати модифікацію програмного коду за вимогою викладача;
- готові захищати свої рішення;
- оформили належним чином звіт.

Питання:

1. Призначення команди CPUID.
2. Рівні та функції команди.
3. Визначення підтримки процесором команди CPUID.
4. Загальний інтерфейс команди.
5. Схема використання команди.
6. Визначення максимального номера функції кожного рівня команди.
7. Отримання назви виробника процесору.
8. Отримання назви процесору.
9. Визначення Brand ID процесору.
10. Визначення первинної та розширеної сигнатури процесору.
11. Література: [1], [2], [10], [11], [12], [13].

ПРАКТИЧНА РОБОТА №8.
ОБРОБКА ТЕКСТОВИХ ДАНИХ

Мета. Придбання навичок з програмування додатків, що реалізують алгоритми обробки текстових даних.

8.1 Загальні відомості про команди роботи з рядками

З точки зору структур даних рядок представляє масив байтових елементів. Для роботи з рядками в системі команд процесорів архітектури IA-32 присутні спеціальні інструкції, що відносяться до групи ланцюгових команд. Ці команди дозволяють виконувати дії над блоками пам'яті, що представляють собою послідовність елементів однакового розміру – 8, 16 або 32 біти. При цьому вміст елементів значення не має, це можуть бути як символьні так і числові дані. Для FLAT моделі обсяг блоків пам'яті досягає 4Гбайт [2,4].

Ланцюгові команди дозволяють виконувати п'ять операцій [2]:

- пересилка ланцюгів;
- порівняння ланцюгів;
- сканування ланцюгу;
- завантаження елементу з ланцюга;
- збереження елемента в ланцюгу.

Команди мають наступний формат [2]:

[ПП] КОП [ОП1] [,ОП2]

де: ПП - префікс повторення;
КОП - мнемонічне позначення команди;
ОП1, ОП2 – операнди команди.

Наявність префіксу повторення змушує процесор виконувати відповідну команду декілька разів. Кількість повторень задається регістром ECX, значення якого після кожної ітерації зменшується на 1, а адреса елементу в ланцюгу модифікується в залежності від розміру елементу. Адреса наступного елементу для виконання чергової ітерації може як збільшуватися, так і зменшуватись по підношенню до адреси поточного елементу [2]:

$$\mathbf{ADR_{i+1} = ADR_i \pm K} \qquad (8.1)$$

де: ADR_{i+1} – адреса наступного елементу ланцюга;

ADR_i – адреса поточного елементу ланцюга;

К – може приймати значення 1,2,4 відповідно, якщо елементи ланцюга байти, слова, подвійні слова [2].

Знак дії «+» або «-» у формулі (8.1) залежить від значення прапору DF в регістрі прапорів. Якщо DF=0, адреса наступного елементу обчислюється як сума, якщо DF=1 – як різниця. Для встановлення значення прапора DF використовуються команди [2]:

CLD - встановити DF в 0
STD - встановити DF в 1

В загальному випадку циклічність виконання припиняється коли вміст регістру ECX становить рівним 0. Якщо префікс повторень відсутній процесор виконує дії тільки над одним елементом [2].

У відповідності до КОП команди мають різну кількість явних операндів [2].

8.2 Формати команд

Для кожної операції, що виконують ланцюгові команди виділяються чотири мнемонічні позначення: одне – загальне і три – в залежності від розміру елементу блоку пам'яті: байт, слово, подвійне слово. Для визначення операції виконання команди можуть бути задані в загальному або спеціалізованому форматі. Перелік команд загального формату наведено в таблиці 8.1. [2].

Таблиця 8.1. Ланцюгові команди загального формату [2].

	Операція	Формат команди	Підготовчі дії	Пояснення
1	Пересилка ланцюга	Movs adr1,adr2	Edi := adr1 Esi := adr2	Пересилка елемента за адресою adr2 у елемент за адресою adr1
2	Порівняння ланцюгів	Cmps adr1,adr2	Edi := adr1 Esi := adr2	Порівняння елементів за вказаними адресами. Встановлює прапори ZF, SF,OF.
3	Сканування ланцюгу	Scas adr	Edi := adr	Пошук вказаного елементу, який знаходить в регістрі акумуляторі в ланцюгу, що заданий адресою adr .
4	Завантаження елемента з ланцюга	Lods adr	Esi := adr	Переслати елемент з ланцюга за адресою adr в регістр акумулятор
5	Збереження елемента в ланцюгу	Stos adr	Edi := adr	Переслати вміст регістру акумулятора в елемент ланцюга за адресою

				adr

Ланцюгові команди спеціалізованого формату наведено у таблиці 8.2 [2].

Таблиця 8.2 Спеціалізовані формати команд [2].

	Операція	Формат команди	Підготовчі дії	Пояснення
1	Пересилка ланцюга байт	Movsb	Edi := adr1 Esi := adr2	Пересилка байта за адресою adr2 у байт за адресою adr1
2	Пересилка ланцюга слів	Movsw	Edi := adr1 Esi := adr2	Пересилка слова за адресою adr2 у слово за адресою adr1
3	Пересилка ланцюга подвійних слів	Movsd	Edi := adr1 Esi := adr2	Пересилка подвійного слова за адресою adr2 у подвійне слово за адресою adr1
4	Порівняння ланцюгів байт	Cmpsb	Edi := adr1 Esi := adr2	Порівняння байтів за вказаними адресами. Встановлює прапори ZF, SF, OF.
5	Порівняння ланцюгів слів	Cmpsw	Edi := adr1 Esi := adr2	Порівняння слів за вказаними адресами. Встановлює прапори ZF, SF, OF
6	Порівняння ланцюгів подвійних слів	Cmpsd	Edi := adr1 Esi := adr2	Порівняння подвійних слів за вказаними адресами. Встановлює прапори ZF, SF, OF
7	Сканування ланцюгу байт	Scasb	Edi := adr	Пошук байта, вміст якого знаходить в регістрі AL, в ланцюгу, що заданий адресою adr .
8	Сканування ланцюгу слів	Scasw	Edi := adr	Пошук слова, вміст якого знаходиться в регістрі AX в ланцюгу, що заданий адресою adr .
9	Сканування ланцюгу подвійних слів	Scasd	Edi := adr	Пошук подвійного слова, вміст якого находиться в регістрі EAX в ланцюгу, що заданий адресою adr .
10	Завантаження байта з ланцюга	Lodsb	Esi := adr	Переслати байт з ланцюгу за адресою adr в AL
11	Завантаження слова з ланцюга	Lodsw	Esi := adr	Переслати слово з ланцюгу за адресою adr в регістр AX
12	Завантаження подвійного слова з ланцюга	Lodsd	Esi := adr	Переслати подвійне слово з ланцюгу за адресою adr в регістр EAX
13	Збереження байта в ланцюгу	Stosb	Edi := adr	Переслати вміст регістру AL в ланцюг за адресою adr
14	Збереження слова в ланцюгу	Stosw	Edi := adr	Переслати вміст регістру AX в ланцюг за адресою adr
15	Збереження подвій-	Stosd	Edi := adr	Переслати вміст регістру EAX в ла-

	ного слова в ланцю-гу			нцюг за адресою adr

8.3 Префікси повторень

Префікси повторень, як і команди мають свої мнемонічні позначення [2]:

- Rep
- Repe або Repz
- Repne або Repnz

Різниця між ними полягає в тому, на якій підставі буде закінчуватися (продовжуватися) циклічне виконання команди, яка має вказаний префікс [2].

У першому випадку (Rep) дії завершуються коли значення регістру ECX становить рівним 0. Найчастіше префікс Rep використовується з командами Movs.

Префікси Repe або Repz – по функціональному призначенню еквівалентні і визначають повторення команди доки ECX не дорівнює 0, або прапор FZ у регістрі прапорів дорівнює 1, що може бути доречним при виконанні операцій пошуку або порівняння [2].

Префікси Repne або Repnz, як і попередні однаково впливають на циклічність виконання команди, а саме повторення відбувається доки ECX не дорівнює 0, або прапор FZ у регістрі прапорів дорівнює 0, що також буде у нагоді при виконанні операцій пошуку або порівняння [2].

Приклад 1. Виконати копіювання символьного рядка [2].

```
.586
.model  flat,  stdcall
option  casemap : none
  .Data
  String_source          db     “ abcde1234fgh “          ; рядок, що потрібно скопіювати
  Len_source             equ    $- String_source          ; розмір рядка
  String_dest            db     Len_source dup (“ “)      ; пам'ять, куди потрібно скопіювати рядок

  .Code
  start:
        cld                              ; встановити DF=0
        lea      esi, String_source      ; визначити регістр ESI
        lea      edi, String_dest        ; визначити регістр EDI
        mov      ecx, Len_source         ; визначити кількість повторень
                 rep    movsb                    ; виконується копіювання (пересилка)
```

```
                                        ; рядка string_source у String_dest.
    ; rep   movs  String_dest, String_source ; можна використати команду загального
                                              ; формату
    End   start
```

Після виконання програми рядки String_source та String_dest будуть мати однаковий вміст, значення регістрів Esi, Edi зміняться на Len_source, тобто, їх значення відповідають адресам, що вказують на елемент пам’яті за кінцем кожного з рядка String_source, String_dest:

Esi := Esi + Len_source, Edi := Edi + Len_source

Приклад 2. Виконати копіювання рядка з видаленням з 6-го по 9-ий елементи. Нумерація елементів починається з 0 [2].

```
.586
.model  flat,  stdcall
option  casemap : none
.Data
String_source     db     “ abcde1234fgh “          ; рядок, що потрібно скопіювати
Len_source        equ    $- String_source          ; розмір рядка
String_dest       db     Len_source dup (“ “)      ; пам'ять, куди треба скопіювати рядок
K_esi             equ    9-6+1                     ; для пропуску непотрібних елементів

.Code
start:
cld                       ; встановити DF=0
lea  esi, String_source   ; визначити регістр ESI
lea  edi, String_dest     ; визначити регістр EDI
mov       ecx, 6                ; визначити кількість повторень
rep movsb                 ; виконується копіювання (пересилка) 6 байт рядка
; string_source у String_dest.

    Add esi, K_esi                ; визначення адреси необхідного елементу
    Mov ecx, Len_source-6- K_esi     ; визначення кількості елементів, що потрібно до-
копіювати
Repmovsb
End       start
```

Після виконання програми рядок String_dest має вміст abcdefgh.

Приклад 3. Підрахувати кількість слів в рядку та знайти слово максимальної довжини. Роздільник між словами - один або декілька пробілів [2].

```
.586

.model  flat,  stdcall
option  casemap : none

.Data
; Вхідні дані - дозволяють перевірити різні ситуації
;---------------------------------------------------------------------
string_source  db       “  Загальний випадок коли пробіли “
               db       “ присутні на початку та в кінці рядка  "
; string_source   db  '   '
; string_source   db 'На початку пробіли відсутні  '
; string_source   db '   В кінці  немає пробілів '
; string_source   db  'A'
; string_source   db  ' '
;---------------------------------------------------------------------
len_source      equ     $- string_source        ; розмір рядка
count_word      dd  0           ; кількість слів
max_len         dd  0           ; довжина максимального слова
number_word dd  0           ; номер слова з максимальною довжиною
adr_end_string  dd  0         ; для перевірки на кінець обробки

.Code

;---------Процедура пошуку слова в рядку------------
; Вхідні дані:
;   EDI - адреса початку пошуку
;   ECX - поточний розмір рядка
; Вихідні дані:
;   EAX - довжина знайденого слова
;   ESI - адреса початку слова, або -1, якщо після останнього слова були тільки пробіли

Find_Word  proc
        mov     esi, -1
        mov     al, ' '                 ; символ пошуку
Repe    scasb       ; пошук символу, що відрізняється від " ", тобто пошук початку слова
        cmp     ecx, 0   ; перевірка умови виходу з scasb
        je      m_f1    ; в рядку немає слів, не знайдено символу, що відрізняється від « »
        inc     ecx
        dec     edi     ; зсув на початок слова
        mov     esi, edi  ; далі шукаємо кінець слова
Repne   scasb       ; пошук символу, що співпадає з " " або кінець рядка
        cmp     ecx,0
        je      m_f2      ;  символ « » не знайдено, кінець рядка
        dec     edi
m_f2:   mov     eax, edi
        sub     eax, esi            ;  довжина слова
m_f1:   ret
Find_Word  endp
```

```
;-------------------------------------------------------------------------------
start:
;-------Перевірка на окремі випадки---

        mov     ecx, len_source
        cmp     ecx,0
        je      m_end       ; порожній рядок
        cmp     ecx, 1
        jne     m_2      ; у рядку більш за один символ
        mov     al, string_source
        cmp     al, ' '
        je      m_end      ; у рядку тільки один пробіл
        inc     count_word    ; у рядку тільки один символ
        inc     number_word
        inc     max_len
        jmp     m_end
;-------------------------------------------------------------
m_2:
        lea     edi, string_source
        mov     ebx,edi
        add     ebx,ecx
        mov     adr_end_string, ebx   ; адреса за кінцем рядка
        CLD             ; перегляд рядка від молодших адрес до старших
m_continue:
        call    Find_Word
        cmp     esi, -1             ; перевірка на кінець рядка ( після останнього слова були
        je      m_end                 ; пробіли) або рядок складається тільки з пробілів
        inc     count_word    ; підрахунок кількості слів
        cmp     eax, max_len
        jl      m_1         ; Max_len не змінюється
        mov     max_len, eax
        mov     ebx, count_word
        mov     number_word, ebx  ; номер поточного максимального слова

m_1:    mov     ebx,esi       ; перевірка на кінець обробки
        add     ebx,eax
        cmp     ebx, adr_end_string
        je      m_end
        mov     edi, ebx
        sub     ebx,offset string_source
        mov     ecx,len_source
        sub     ecx, ebx     ; перерахунок довжини рядка, що підлягає подальшому аналізу
        jmp     m_continue
m_end: End      start
```

8.4 Завдання

Розробити консольний додаток, що складається з різномовних модулів (C++, ACM) та виконує обробку текстових даних у відповідності до завдання [2].

Вимоги до розробки [2]:

- вміст рядка задано кирилицею;
- введення вхідних даних, та виведення результатів реалізується на мові високого рівня;
- необхідний обсяг пам’яті для вхідних та вихідних даних виділяється в модулі C++.
- обробка текстового рядка виконується модулем ACM;
- модуль ACM складається з декілька процедур;
- вхідні дані вводяться в режимі діалогу;
- перед виводом результату обов’язковий вивід вхідного рядка;
- вивід результатів супроводжується коментарем.

1. У рядку замінити кілька поспіль пробілів одним. Вивести номери слів, між якими була виконана заміна.
2. У рядку знайти слово, що повторюється. Вивести знайдене слово та кількість повторень. Примітка: повторюватись може тільки одне слово.
3. У рядку знайти слова, в яких буква «а» зустрічається більш одного разу. Вивести знайдені слова.
4. У рядку знайти першу послідовність слів, довжини яких складають арифметичну прогресію з кроком 2. Вивести ці слова, та їх розмір.
5. У рядку знайти слова, в яких початкова буква збігається з останньою. Вивести знайдені слова.
6. Розташувати слова в рядку в порядку зростання довжини слів. Вивести перетворений рядок.
7. У рядку видалити слова однакової довжини. Вивести перетворений рядок.
8. Збільшити довжину рядка, до вказаної, додавши між максимальною кількістю слів пробіли.
9. У рядку знайти слова, які закінчуються на вказану цифру, що задається з пристрою введення. Вивести знайдені слова.
10. Визначити чи мають перше та останнє слово рядка однакову довжину. Якщо так, вивести ці слова.
11. У рядку переставити слова місцями (1-2, 3-4 ...). Вивести перетворений рядок.
12. У рядку знайти всі слова, довжина яких мінімальна. Вивести ці слова.
13. Виконати інверсію першого і останнього слова в рядку. Вивести перетворений рядок.
14. У рядку замінити кожен другий символ слова на вказаний, що задається з пристрою введення. Вивести перетворений рядок.
15. Визначити чи задовольняє рядок шаблону [а - м] '\ 3' \ $ (кінець рядка повинен закінчуватись на 3 символи з діапазону від «а» до «м»).

16. У рядку знайти останнє входження заданого слова. Вивести номер заданого слова в рядку.
17. Рядок містить перелік назв файлів. Назва складається з імені та розширення, що відокремлюються один від одного «.» . В розширенні може бути 1, 2 або 3 символи, але воно (відповідно і «.») може бути і відсутнім . Знайти назви файлів з максимальною кількість символів в розширенні. Вивести знайдені назви.
18. Якщо в рядку відсутнє вказане слово, що задається з пристрою введення, то додати його в кінець рядка. Вивести перетворений рядок.
19. У рядку вирівняти довжини всіх слів, додавши, при необхідності, зліва і справа слова символ «*». Вивести перетворений рядок. Примітка: довжина вихідного рядка не може перевищувати довжину вхідного рядка більш ніж у 2 рази, та повинна бути введена з пристрою введення
20. У рядку знайти слова однакової довжини. Вивести номери цих слів.
21. У рядку видалити задане слово. Вивести перетворений рядок.
22. Підрахувати кількість вказаної букви, що задається з пристрою введення, в кожному слові. Вивести слово та кількість вказаних букв.
23. У рядку видалити слова, якщо їх довжини менше вказаної, що задається з пристрою введення. Вивести перетворений рядок.
24. У рядку після кожного другого слова вставити вказане, що задається з пристрою введення. Вивести перетворений рядок.
25. У рядку видалити найдовше слово. Вивести слово, яке видаляється, його довжину та перетворений рядок.
26. У рядку знайти останню послідовність слів, довжини яких складають арифметичну прогресію з кроком 1. Вивести ці слова та їх розмір.
27. Рядок може містити телефонні номери в форматі «**********», де «*» - це цифра. Знайти телефонні номери, які починаються на 095. Вивести знайдені номери.
28. Виконати послівний реверс рядка. Вивести перетворений рядок.
29. Рядок являє собою перелік деяких текстових параметрів (слів) розділених символом «:». Якщо параметр відсутні символи «:» розташовані один за одним. Вивести номери відсутніх параметрів.
30. У рядку знайти перше входження заданого слова. Вивести номер заданого слова в рядку
31. Визначити чи задовольняє рядок шаблону ^ [к-р] '\ 2' \ (рядок починається двома символами з діапазону від «к» до «р»).
32. Визначити чи задовольняє рядок умові : всі слова повинні закінчуватись на вказаний символ, що задається з пристрою введення. Якщо, ні – вивести перше слово, що не задовольняє умові.

8.5 Порядок виконання

1. Визначити початкові вхідні параметри.

2. Розробити алгоритм програми.
3. Аналітично отримати результат.
4. Скласти програму.
5. Отримати виконавчий файл.
6. Виконати програму.
7. Порівняти отримані результати з результатами п.3.
8. Протестувати програму з різними вхідними параметрами

8.6 Структура звіту

Звіт оформлюється на окремих аркушах формату А4 та має наступний зміст:

1. Титул.
2. Постанова завдання.
3. Текст програми.
4. Результати роботи.
5. Висновки.

8.7 Питання до захисту

До захисту лабораторної роботи допускаються студенти, які:

- довели працездатність розробленого ПЗ;
- довели достовірність отриманих результатів;
- готові захищати свої рішення;
- оформили належним чином звіт.

Питання:

1. Призначення ланцюгових команд.
2. Типи даних, що використовують ланцюгові команди.
3. Загальний та спеціалізований формат команд.
4. Адресація операндів в ланцюгових командах.
5. Вплив прапору DF на виконання команд.
6. Призначення префіксів повторення.
7. Види префіксів повторення та їх вплив на виконання команди.
8. Особливості створення проекту, що включає різномовні модулі.
9. Налаштування властивостей проекту у VS.
10. Використання вбудованого відладчика для відстежування виконання.
11. Література: [1], [2], [8], [10], [11], [12].

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Методичні вказівки до виконання курсового проекту з дисципліни «Проектування систем реального часу» для студентів спеціальності 123 Комп'ютерна інженерія усіх форм навчання [Електронний ресурс] / уклад. В.В. Шамаєв. – Луцьк : ДонНТУ, 2022. – 36 с.
2. Методичні вказівки до лабораторних робіт з дисципліни «Системне програмування» для студентів спеціальностей: 121 Інженерія програмного забезпечення, 122 Комп'ютерні науки, 123 Комп'ютерна інженерія денної та заочної форми навчання. [Електронний ресурс] / уклад. О.Г. Шевченко. – Покровськ : ДонНТУ, 2020. – 93 с.
3. Мосіюк О. Операційні системи та системне програмування: навч.-метод. посіб. / О. Мосіюк, А. Федорчук. - Житомир: ЖДУ ім. Івана Франка, 2022. - 76 с.
4. Практикум з системного програмного забезпечення :навч. посіб. / Я. Савицька, В. Смолій, Н. Чичикало та ін. - Київ: НУБіП України, 2020. – 262 с.
5. Операційні системи : навч. посіб. / В. Зайцев, І. Дробязко. – Київ: КПІ ім. І. Сікорського, 2019. – 240 с.
6. Вступ до алгоритмів / Томас Г. Кормен, Чарлз Е. Лейзерсон, Роналд Л. Рівест. - Київ: К. І. С., 2019. - 1288 с.
7. UNIX and Linux System Administration Handbook [Електронний ресурс] / Trent R. Hein, Evi Nemeth, Garth Snyder, Ben Whaley, Dan Mackin :5th Edition. – Режим доступу : http://surl.li/oqvjd. - Назва з екрана.
8. Stallings William Operating Systems: Internals and Design Principles [Електронний ресурс] : 9th Edition. – Pearson, 2018. - Режим доступу : https://learn.ztu.edu.ua/mod/book/tool/print/index.php?id=27881. - Назва з екрана.
9. AMD Developer Center [Електронний ресурс] : сайт. - Режим доступу : https://www.amd.com/en/developer.html. - Назва з екрана.
10. AMD64 Architecture Programmer's Manual [Електронний ресурс] : Volume 3: General-Purpose and System Instructions - Режим доступу : http://surl.li/oqwyy. - Назва з екрана.
11. CPUID [Електронний ресурс] : сайт. - Режим доступу : https://www.cpuid.com. - Назва з екрана.
12. Codecademy [Електронний ресурс] : офіц. сторінка. - Режим доступу : https://www.codecademy.com. - Назва з екрана.
13. Linux [Електронний ресурс] : офіц. сторінка. - Режим доступу : https://www.linux.org. - Назва з екрана.

14. CPUID Function specification [Електронний ресурс]. - Режим доступу : https://kib.kiev.ua/x86docs/AMD/AMD-CPUID-Spec/25481-r2.28.pdf. - Назва з екрана.
15. Canonical UBUNTU [Електронний ресурс] : офіц. сторінка. - Режим доступу : https://ubuntu.com. - Назва з екрана.

ДОДАТОК А.
СПЕЦИФІКАЦІЯ КОМАНДИ CPUID ДЛЯ ПРОЦЕСОРІВ INTEL

Інформація, що надана в додатку представляє собою окремі фрагменти з офіційної документації процесорів Intel [5].

"Caching Translation Information" in Chapter 4, "Paging," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

Table 3-17. Information Returned by CPUID Instruction

Initial EAX Value		Information Provided about the Processor
	Basic CPUID Information	
0H	EAX EBX ECX EDX	Maximum Input Value for Basic CPUID Information (see Table 3-18) "Genu" "ntel" "inel"
01H	EAX	Version Information: Type, Family, Model, and Stepping ID (see Figure 3-5)
	EBX	Bits 07-00: Brand Index Bits 15-08: CLFLUSH line size (Value ∗ 8 = cache line size in bytes) Bits 23-16: Maximum number of addressable IDs for logical processors in this physical package*. Bits 31-24: Initial APIC ID
	ECX EDX	Feature Information (see Figure 3-6 and Table 3-19) Feature Information (see Figure 3-7 and Table 3-20)
		NOTES: * The nearest power-of-2 integer that is not smaller than EBX[23:16] is the number of unique initial APIC IDs reserved for addressing different logical processors in a physical package. This field is only valid if CPUID.1.EDX.HTT[bit 28]= 1.
02H	EAX EBX ECX EDX	Cache and TLB Information (see Table 3-21) Cache and TLB Information Cache and TLB Information Cache and TLB Information
03H	EAX EBX ECX	Reserved. Reserved. Bits 00-31 of 96 bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.)
	EDX	Bits 32-63 of 96 bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.)
		NOTES: Processor serial number (PSN) is not supported in the Pentium 4 processor or later. On all models, use the PSN flag (returned using CPUID) to check for PSN support before accessing the feature. See AP-485, *Intel Processor Identification and the CPUID Instruction* (Order Number 241618) for more information on PSN.
	CPUID leaves > 3 < 80000000 are visible only when IA32_MISC_ENABLE.BOOT_NT4[bit 22] = 0 (default).	
	Deterministic Cache Parameters Leaf	
04H		**NOTES:** Leaf 04H output depends on the initial value in ECX.* See also: "INPUT EAX = 4: Returns Deterministic Cache Parameters for each level on page 3-182.
	EAX	Bits 04-00: Cache Type Field 0 = Null - No more caches 1 = Data Cache 2 = Instruction Cache 3 = Unified Cache 4-31 = Reserved

Table 3-17. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
		Bits 07-05: Cache Level (starts at 1) Bit 08: Self Initializing cache level (does not need SW initialization) Bit 09: Fully Associative cache Bits 13-10: Reserved Bits 25-14: Maximum number of addressable IDs for logical processors sharing this cache**, *** Bits 31-26: Maximum number of addressable IDs for processor cores in the physical package**, ****, *****
	EBX	Bits 11-00: L = System Coherency Line Size** Bits 21-12: P = Physical Line partitions** Bits 31-22: W = Ways of associativity**
	ECX	Bits 31-00: S = Number of Sets**
	EDX	Bit 0: Write-Back Invalidate/Invalidate 0 = WBINVD/INVD from threads sharing this cache acts upon lower level caches for threads sharing this cache. 1 = WBINVD/INVD is not guaranteed to act upon lower level caches of non-originating threads sharing this cache. Bit 1: Cache Inclusiveness 0 = Cache is not inclusive of lower cache levels. 1 = Cache is inclusive of lower cache levels. Bit 2: Complex Cache Indexing 0 = Direct mapped cache. 1 = A complex function is used to index the cache, potentially using all address bits. Bits 31-03: Reserved = 0
		NOTES: * If ECX contains an invalid sub leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n+1 is invalid if sub-leaf n returns EAX[4:0] as 0. ** Add one to the return value to get the result. ***The nearest power-of-2 integer that is not smaller than (1 + EAX[25:14]) is the number of unique initial APIC IDs reserved for addressing different logical processors sharing this cache **** The nearest power-of-2 integer that is not smaller than (1 + EAX[31:26]) is the number of unique Core_IDs reserved for addressing different processor cores in a physical package. Core ID is a subset of bits of the initial APIC ID. ***** The returned value is constant for valid initial values in ECX. Valid ECX values start from 0.
	MONITOR/MWAIT Leaf	
05H	EAX	Bits 15-00: Smallest monitor-line size in bytes (default is processor's monitor granularity) Bits 31-16: Reserved = 0
	EBX	Bits 15-00: Largest monitor-line size in bytes (default is processor's monitor granularity) Bits 31-16: Reserved = 0
	ECX	Bit 00: Enumeration of Monitor-Mwait extensions (beyond EAX and EBX registers) supported Bit 01: Supports treating interrupts as break-event for MWAIT, even when interrupts disabled Bits 31 - 02: Reserved

Table 3-17. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	EDX	Bits 03 - 00: Number of C0* sub C-states supported using MWAIT Bits 07 - 04: Number of C1* sub C-states supported using MWAIT Bits 11 - 08: Number of C2* sub C-states supported using MWAIT Bits 15 - 12: Number of C3* sub C-states supported using MWAIT Bits 19 - 16: Number of C4* sub C-states supported using MWAIT Bits 23 - 20: Number of C5* sub C-states supported using MWAIT Bits 27 - 24: Number of C6* sub C-states supported using MWAIT Bits 31 - 28: Number of C7* sub C-states supported using MWAIT **NOTE:** * The definition of C0 through C7 states for MWAIT extension are processor-specific C-states, not ACPI C-states.
	Thermal and Power Management Leaf	
06H	EAX	Bit 00: Digital temperature sensor is supported if set Bit 01: Intel Turbo Boost Technology Available (see description of IA32_MISC_ENABLE[38]). Bit 02: ARAT. APIC-Timer-always-running feature is supported if set. Bit 03: Reserved Bit 04: PLN. Power limit notification controls are supported if set. Bit 05: ECMD. Clock modulation duty cycle extension is supported if set. Bit 06: PTM. Package thermal management is supported if set. Bit 07: HWP. HWP base registers (IA32_PM_ENALBE[bit 0], IA32_HWP_CAPABILITIES, IA32_HWP_REQUEST, IA32_HWP_STATUS) are supported if set. Bit 08: HWP_Notification. IA32_HWP_INTERRUPT MSR is supported if set. Bit 09: HWP_Activity_Window. IA32_HWP_REQUEST[bits 41:32] is supported if set. Bit 10: HWP_Energy_Performance_Preference. IA32_HWP_REQUEST[bits 31:24] is supported if set. Bit 11: HWP_Package_Level_Request. IA32_HWP_REQUEST_PKG MSR is supported if set. Bit 12: Reserved. Bit 13: HDC. HDC base registers IA32_PKG_HDC_CTL, IA32_PM_CTL1, IA32_THREAD_STALL MSRs are supported if set. Bits 31 - 15: Reserved
	EBX	Bits 03 - 00: Number of Interrupt Thresholds in Digital Thermal Sensor Bits 31 - 04: Reserved
	ECX	Bit 00: Hardware Coordination Feedback Capability (Presence of IA32_MPERF and IA32_APERF). The capability to provide a measure of delivered processor performance (since last reset of the counters), as a percentage of the expected processor performance when running at the TSC frequency. Bits 02 - 01: Reserved = 0 Bit 03: The processor supports performance-energy bias preference if CPUID.06H:ECX.SETBH[bit 3] is set and it also implies the presence of a new architectural MSR called IA32_ENERGY_PERF_BIAS (1B0H). Bits 31 - 04: Reserved = 0
	EDX	Reserved = 0
	Structured Extended Feature Flags Enumeration Leaf (Output depends on ECX input value)	
07H		Sub-leaf 0 (Input ECX = 0). *
	EAX	Bits 31-00: Reports the maximum input value for supported leaf 7 sub-leaves.

Table 3-17. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	EBX	Bit 00: FSGSBASE. Supports RDFSBASE/RDGSBASE/WRFSBASE/WRGSBASE if 1. Bit 01: IA32_TSC_ADJUST MSR is supported if 1. Bit 02: Reserved Bit 03: BMI1 Bit 04: HLE Bit 05: AVX2 Bit 06: Reserved Bit 07: SMEP. Supports Supervisor-Mode Execution Prevention if 1. Bit 08: BMI2 Bit 09: Supports Enhanced REP MOVSB/STOSB if 1. Bit 10: INVPCID. If 1, supports INVPCID instruction for system software that manages process-context identifiers. Bit 11: RTM Bit 12: Supports Platform Quality of Service Monitoring (PQM) capability if 1. Bit 13: Deprecates FPU CS and FPU DS values if 1. Bit 14: Reserved. Bit 15: Supports Platform Quality of Service Enforcement (PQE) capability if 1. Bits 17:16: Reserved Bit 18: RDSEED Bit 19: ADX Bit 20: SMAP Bits 24:21: Reserved Bit 25: Intel Processor Trace Bits 31:26: Reserved
	ECX	Bit 00: PREFETCHWT1 Bit 31-01: Reserved
	EDX	Reserved
		NOTE: * If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX.
	Direct Cache Access Information Leaf	
09H	EAX	Value of bits [31:0] of IA32_PLATFORM_DCA_CAP MSR (address 1F8H)
	EBX	Reserved
	ECX	Reserved
	EDX	Reserved
	Architectural Performance Monitoring Leaf	
0AH	EAX	Bits 07 - 00: Version ID of architectural performance monitoring Bits 15- 08: Number of general-purpose performance monitoring counter per logical processor Bits 23 - 16: Bit width of general-purpose, performance monitoring counter Bits 31 - 24: Length of EBX bit vector to enumerate architectural performance monitoring events
	EBX	Bit 00: Core cycle event not available if 1 Bit 01: Instruction retired event not available if 1 Bit 02: Reference cycles event not available if 1 Bit 03: Last-level cache reference event not available if 1 Bit 04: Last-level cache misses event not available if 1 Bit 05: Branch instruction retired event not available if 1 Bit 06: Branch mispredict retired event not available if 1 Bits 31- 07: Reserved = 0
	ECX	Reserved = 0

Table 3-17. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	EDX	Bits 04 - 00: Number of fixed-function performance counters (if Version ID > 1) Bits 12- 05: Bit width of fixed-function performance counters (if Version ID > 1) Reserved = 0
	Extended Topology Enumeration Leaf	
0BH		**NOTES:** Most of Leaf 0BH output depends on the initial value in ECX. The EDX output of leaf 0BH is always valid and does not vary with input value in ECX. Output value in ECX[7:0] always equals input value in ECX[7:0]. For sub-leaves that return an invalid level-type of 0 in ECX[15:8]; EAX and EBX will return 0. If an input value n in ECX returns the invalid level-type of 0 in ECX[15:8], other input values with ECX > n also return 0 in ECX[15:8].
	EAX	Bits 04-00: Number of bits to shift right on x2APIC ID to get a unique topology ID of the next level type*. All logical processors with the same next level ID share current level. Bits 31-05: Reserved.
	EBX	Bits 15 - 00: Number of logical processors at this level type. The number reflects configuration as shipped by Intel**. Bits 31- 16: Reserved.
	ECX	Bits 07 - 00: Level number. Same value in ECX input Bits 15 - 08: Level type***. Bits 31 - 16:: Reserved.
	EDX	Bits 31- 00: x2APIC ID the current logical processor. **NOTES:** * Software should use this field (EAX[4:0]) to enumerate processor topology of the system. ** Software must not use EBX[15:0] to enumerate processor topology of the system. This value in this field (EBX[15:0]) is only intended for display/diagnostic purposes. The actual number of logical processors available to BIOS/OS/Applications may be different from the value of EBX[15:0], depending on software and platform hardware configurations. *** The value of the "level type" field is not related to level numbers in any way, higher "level type" values do not mean higher levels. Level type field has the following encoding: 0: invalid 1: SMT 2: Core 3-255: Reserved
	Processor Extended State Enumeration Main Leaf (EAX = 0DH, ECX = 0)	
0DH		**NOTES:** Leaf 0DH main leaf (ECX = 0).
	EAX	Bits 31-00: Reports the valid bit fields of the lower 32 bits of XCR0. If a bit is 0, the corresponding bit field in XCR0 is reserved. Bit 00: legacy x87 Bit 01: 128-bit SSE Bit 02: 256-bit AVX Bits 31- 03: Reserved
	EBX	Bits 31-00: Maximum size (bytes, from the beginning of the XSAVE/XRSTOR save area) required by enabled features in XCR0. May be different than ECX if some features at the end of the XSAVE save area are not enabled.

Table 3-17. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	ECX	Bit 31-00: Maximum size (bytes, from the beginning of the XSAVE/XRSTOR save area) of the XSAVE/XRSTOR save area required by all supported features in the processor, i.e all the valid bit fields in XCR0.
	EDX	Bit 31-00: Reports the valid bit fields of the upper 32 bits of XCR0. If a bit is 0, the corresponding bit field in XCR0 is reserved.
	Processor Extended State Enumeration Sub-leaf (EAX = 0DH, ECX = 1)	
0DH	EAX	Bits 31-04: Reserved Bit 00: XSAVEOPT is available Bit 01: Supports XSAVEC and the compacted form of XRSTOR if set Bit 02: Supports XGETBV with ECX = 1 if set Bit 03: Supports XSAVES/XRSTORS and IA32_XSS if set
	EBX	Bits 31-00: The size in bytes of the XSAVE area containing all states enabled by XCR0 | IA32_XSS.
	ECX	Bits 31-00: Reports the valid bit fields of the lower 32 bits of IA32_XSS. If a bit is 0, the corresponding bit field in IA32_XSS is reserved. Bits 07-00: Reserved Bit 08: IA32_XSS[bit 8] is supported if 1 Bits 31-09: Reserved
	EDX	Bits 31-00: Reports the valid bit fields of the upper 32 bits of IA32_XSS. If a bit is 0, the corresponding bit field in IA32_XSS is reserved.
	Processor Extended State Enumeration Sub-leaves (EAX = 0DH, ECX = n, n > 1)	
0DH		**NOTES:** Leaf 0DH output depends on the initial value in ECX. Each valid sub-leaf index maps to a valid bit in either the XCR0 register or the IA32_XSS MSR starting at bit position 2. * If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf n (0 ≤ n ≤ 31) is invalid if sub-leaf 0 returns 0 in EAX[n] and sub-leaf 1 returns 0 in ECX[n]. Sub-leaf n (32 ≤ n ≤ 63) is invalid if sub-leaf 0 returns 0 in EDX[n-32] and sub-leaf 1 returns 0 in EDX[n-32].
	EAX	Bits 31-0: The size in bytes (from the offset specified in EBX) of the save area for an extended state feature associated with a valid sub-leaf index, *n*.
	EBX	Bits 31-0: The offset in bytes of this extended state component's save area from the beginning of the XSAVE/XRSTOR area. This field reports 0 if the sub-leaf index, n, does not map to a valid bit in the XCR0 register*.
	ECX	Bit 0 is set if the sub-leaf index, n, maps to a valid bit in the IA32_XSS MSR and bit 0 is clear if n maps to a valid bit in XCR0. Bits 31-1 are reserved. This field reports 0 if the sub-leaf index, n, is invalid*.
	EDX	This field reports 0 if the sub-leaf index, *n*, is invalid*; otherwise it is reserved.
	Platform QoS Monitoring Enumeration Sub-leaf (EAX = 0FH, ECX = 0)	
0FH		**NOTES:** Leaf 0FH output depends on the initial value in ECX. Sub-leaf index 0 reports valid resource type starting at bit position 1 of EDX
	EAX	Reserved.
	EBX	Bits 31-0: Maximum range (zero-based) of RMID within this physical processor of all types.
	ECX	Reserved.

Table 3-17. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	EDX	Bit 00: Reserved. Bit 01: Supports L3 Cache QoS Monitoring if 1. Bits 31:02: Reserved
	L3 Cache QoS Monitoring Capability Enumeration Sub-leaf (EAX = 0FH, ECX = 1)	
0FH	NOTES: Leaf 0FH output depends on the initial value in ECX.	
	EAX	Reserved.
	EBX	Bits 31-0: Conversion factor from reported IA32_QM_CTR value to occupancy metric (bytes).
	ECX	Maximum range (zero-based) of RMID of this resource type.
	EDX	Bit 00: Supports L3 occupancy monitoring if 1. Bits 31:01: Reserved
	Platform QoS Enforcement Enumeration Sub-leaf (EAX = 10H, ECX = 0)	
10H	NOTES: Leaf 10H output depends on the initial value in ECX. Sub-leaf index 0 reports valid resource identification (ResID) starting at bit position 1 of EDX	
	EAX	Reserved.
	EBX	Bit 00: Reserved. Bit 01: Supports L3 Cache QoS Enforcement if 1. Bits 31:02: Reserved
	ECX	Reserved.
	EDX	Reserved.
	L3 Cache QoS Enforcement Enumeration Sub-leaf (EAX = 10H, ECX = ResID =1)	
10H	NOTES: Leaf 10H output depends on the initial value in ECX.	
	EAX	Bits 4:0: Length of the capacity bit mask for the corresponding ResID. Bits 31:05: Reserved
	EBX	Bits 31-0: Bit-granular map of isolation/contention of allocation units.
	ECX	Bit 00: Reserved. Bit 01: Updates of COS should be infrequent if 1. Bits 31:02: Reserved
	EDX	Bits 15:0: Highest COS number supported for this ResID. Bits 31:16: Reserved
	Intel Processor Trace Enumeration Main Leaf (EAX = 14H, ECX = 0)	
14H	NOTES: Leaf 14H main leaf (ECX = 0).	
	EAX	Bits 31-0: Reports the maximum number sub-leaves that are supported in leaf 14H.
	EBX	Bit 00: If 1, Indicates that IA32_RTIT_CTL.CR3Filter can be set to 1, and that IA32_RTIT_CR3_MATCH MSR can be accessed. Bits 31- 01: Reserved

Table 3-17. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	ECX	Bit 00: If 1, Tracing can be enabled with IA32_RTIT_CTL.ToPA = 1, hence utilizing the ToPA output scheme; IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUTPUT_MASK_PTRS MSRs can be accessed. Bit 01: If 1, ToPA tables can hold any number of output entries, up to the maximum allowed by the MaskOrTableOffset field of IA32_RTIT_OUTPUT_MASK_PTRS. Bit 30:02: Reserved Bit 31: If 1, Generated packets which contain IP payloads have LIP values, which include the CS base component.
	EDX	Bits 31- 00: Reserved
	Time Stamp Counter/Core Crystal Clock Information-leaf	
15H		**NOTES:** If EBX[31:0] is 0, the TSC/"core crystal clock" ration is not enumerated. EBX[31:0]/EAX[31:0] indicates the ratio of the TSC frequency and the core crystal clock frequency. "TSC frequency" = "core crystal clock frequency" * EBX/EAX. The core crystal clock may differ from the reference clock, bus clock, or core clock frequencies.
	EAX	Bits 31:0: An unsigned integer which is the denominator of the TSC/"core crystal clock" ratio.
	EBX	Bits 31-0: An unsigned integer which is the numerator of the TSC/"core crystal clock" ratio.
	ECX	Bits 31:0: Reserved = 0.
	EDX	Bits 31:0: Reserved = 0.
	Unimplemented CPUID Leaf Functions	
40000000H - 4FFFFFFFH		Invalid. No existing or future CPU will return processor identification or feature information if the initial EAX value is in the range 40000000H to 4FFFFFFFH.
	Extended Function CPUID Information	
80000000H	EAX	Maximum Input Value for Extended Function CPUID Information (see Table 3-18).
	EBX	Reserved
	ECX	Reserved
	EDX	Reserved
80000001H	EAX	Extended Processor Signature and Feature Bits.
	EBX	Reserved
	ECX	Bit 00: LAHF/SAHF available in 64-bit mode Bits 04-01 Reserved Bit 05: LZCNT Bits 07-06 Reserved Bit 08: PREFETCHW Bits 31-09 Reserved

Table 3-17. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	EDX	Bits 10-00: Reserved Bit 11: SYSCALL/SYSRET available in 64-bit mode Bits 19-12: Reserved = 0 Bit 20: Execute Disable Bit available Bits 25-21: Reserved = 0 Bit 26: 1-GByte pages are available if 1 Bit 27: RDTSCP and IA32_TSC_AUX are available if 1 Bits 28: Reserved = 0 Bit 29: Intel® 64 Architecture available if 1 Bits 31-30: Reserved = 0
80000002H	EAX EBX ECX EDX	Processor Brand String Processor Brand String Continued Processor Brand String Continued Processor Brand String Continued
80000003H	EAX EBX ECX EDX	Processor Brand String Continued Processor Brand String Continued Processor Brand String Continued Processor Brand String Continued
80000004H	EAX EBX ECX EDX	Processor Brand String Continued Processor Brand String Continued Processor Brand String Continued Processor Brand String Continued
80000005H	EAX EBX ECX EDX	Reserved = 0 Reserved = 0 Reserved = 0 Reserved = 0
80000006H	EAX EBX ECX EDX	Reserved = 0 Reserved = 0 Bits 07-00: Cache Line size in bytes Bits 11-08: Reserved Bits 15-12: L2 Associativity field * Bits 31-16: Cache size in 1K units Reserved = 0 **NOTES:** * L2 associativity field encodings: 00H - Disabled 01H - Direct mapped 02H - 2-way 04H - 4-way 06H - 8-way 08H - 16-way 0FH - Fully associative
80000007H	EAX EBX ECX EDX	Reserved = 0 Reserved = 0 Reserved = 0 Bits 07-00: Reserved = 0 Bit 08: Invariant TSC available if 1 Bits 31-09: Reserved = 0

Table 3-17. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
80000008H	EAX	Linear/Physical Address size Bits 07-00: #Physical Address Bits* Bits 15-8: #Linear Address Bits Bits 31-16: Reserved = 0
	EBX	Reserved = 0
	ECX	Reserved = 0
	EDX	Reserved = 0
		NOTES: * If CPUID.80000008H:EAX[7:0] is supported, the maximum physical address number supported should come from this field.

INPUT EAX = 0: Returns CPUID's Highest Value for Basic Processor Information and the Vendor Identification String

When CPUID executes with EAX set to 0, the processor returns the highest value the CPUID recognizes for returning basic processor information. The value is returned in the EAX register (see Table 3-18) and is processor specific.

A vendor identification string is also returned in EBX, EDX, and ECX. For Intel processors, the string is "GenuineIntel" and is expressed:

```
EBX ← 756e6547h (* "Genu", with G in the low eight bits of BL *)
EDX ← 49656e69h (* "inel", with i in the low eight bits of DL *)
ECX ← 6c65746eh (* "ntel", with n in the low eight bits of CL *)
```

INPUT EAX = 80000000H: Returns CPUID's Highest Value for Extended Processor Information

When CPUID executes with EAX set to 80000000H, the processor returns the highest value the processor recognizes for returning extended processor information. The value is returned in the EAX register and is processor specific.

IA32_BIOS_SIGN_ID Returns Microcode Update Signature

For processors that support the microcode update facility, the IA32_BIOS_SIGN_ID MSR is loaded with the update signature whenever CPUID executes. The signature is returned in the upper DWORD. For details, see Chapter 9 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

INPUT EAX = 1: Returns Model, Family, Stepping Information

When CPUID executes with EAX set to 1, version information is returned in EAX (see Figure 3-5). For example: model, family, and processor type for the Intel Xeon processor 5100 series is as follows:

- Model — 1111B
- Family — 0101B
- Processor Type — 00B

See Table 3-18 for available processor type values. Stepping IDs are provided as needed.

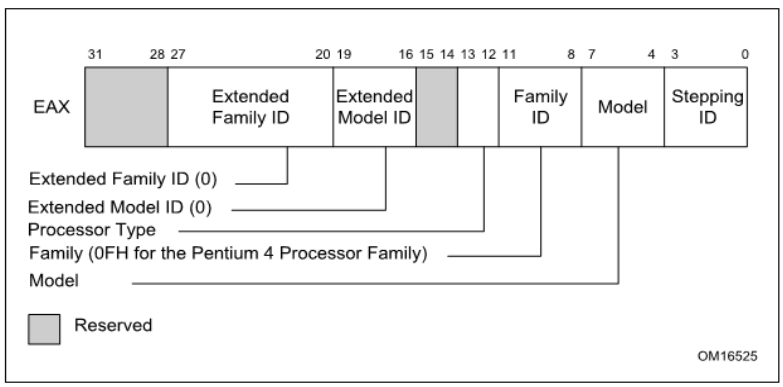

Figure 3-5. Version Information Returned by CPUID in EAX

Table 3-18. Processor Type Field

Type	Encoding
Original OEM Processor	00B
Intel OverDrive® Processor	01B
Dual processor (not applicable to Intel486 processors)	10B
Intel reserved	11B

NOTE

See Chapter 17 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for information on identifying earlier IA-32 processors.

The Extended Family ID needs to be examined only when the Family ID is 0FH. Integrate the fields into a display using the following rule:

```
IF Family_ID ≠ 0FH
    THEN DisplayFamily = Family_ID;
    ELSE DisplayFamily = Extended_Family_ID + Family_ID;
    (* Right justify and zero-extend 4-bit field. *)
FI;
(* Show DisplayFamily as HEX field. *)
```

The Extended Model ID needs to be examined only when the Family ID is 06H or 0FH. Integrate the field into a display using the following rule:

```
IF (Family_ID = 06H or Family_ID = 0FH)
    THEN DisplayModel = (Extended_Model_ID « 4) + Model_ID;
    (* Right justify and zero-extend 4-bit field; display Model_ID as HEX field.*)
    ELSE DisplayModel = Model_ID;
FI;
(* Show DisplayModel as HEX field. *)
```

Table 3-19. Feature Information Returned in the ECX Register (Contd.)

Bit #	Mnemonic	Description
29	F16C	A value of 1 indicates that processor supports 16-bit floating-point conversion instructions.
30	RDRAND	A value of 1 indicates that processor supports RDRAND instruction.
31	Not Used	Always returns 0.

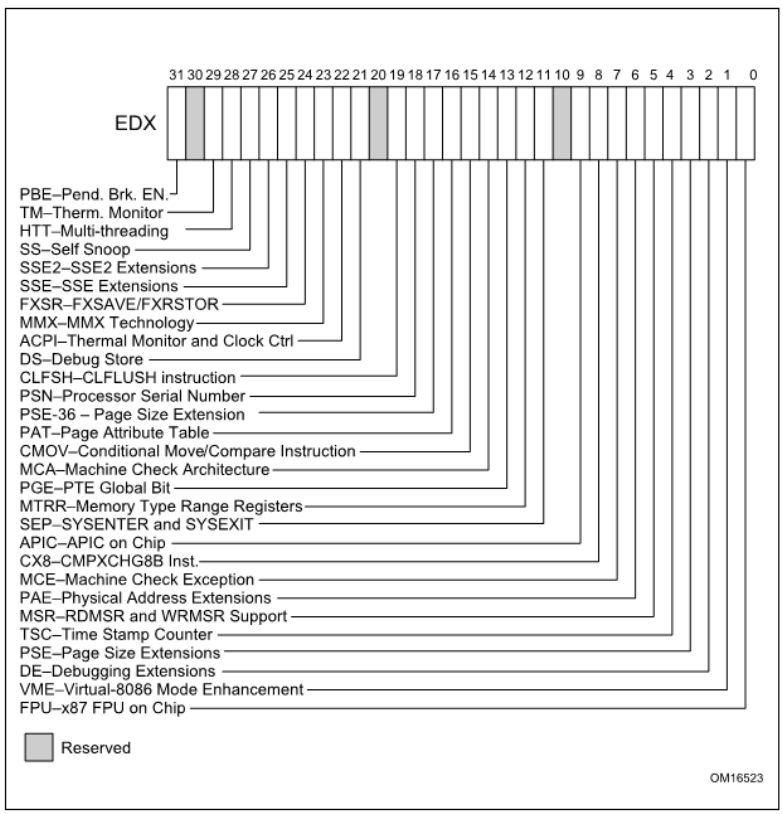

Figure 3-7. Feature Information Returned in the EDX Register

Table 3-19. Feature Information Returned in the ECX Register

Bit #	Mnemonic	Description
0	SSE3	**Streaming SIMD Extensions 3 (SSE3).** A value of 1 indicates the processor supports this technology.
1	PCLMULQDQ	**PCLMULQDQ.** A value of 1 indicates the processor supports the PCLMULQDQ instruction.
2	DTES64	**64-bit DS Area.** A value of 1 indicates the processor supports DS area using 64-bit layout.
3	MONITOR	**MONITOR/MWAIT.** A value of 1 indicates the processor supports this feature.
4	DS-CPL	**CPL Qualified Debug Store.** A value of 1 indicates the processor supports the extensions to the Debug Store feature to allow for branch message storage qualified by CPL.
5	VMX	**Virtual Machine Extensions.** A value of 1 indicates that the processor supports this technology.
6	SMX	**Safer Mode Extensions.** A value of 1 indicates that the processor supports this technology. See Chapter 5, "Safer Mode Extensions Reference".
7	EIST	**Enhanced Intel SpeedStep® technology.** A value of 1 indicates that the processor supports this technology.
8	TM2	**Thermal Monitor 2.** A value of 1 indicates whether the processor supports this technology.
9	SSSE3	A value of 1 indicates the presence of the Supplemental Streaming SIMD Extensions 3 (SSSE3). A value of 0 indicates the instruction extensions are not present in the processor.
10	CNXT-ID	**L1 Context ID.** A value of 1 indicates the L1 data cache mode can be set to either adaptive mode or shared mode. A value of 0 indicates this feature is not supported. See definition of the IA32_MISC_ENABLE MSR Bit 24 (L1 Data Cache Context Mode) for details.
11	SDBG	A value of 1 indicates the processor supports IA32_DEBUG_INTERFACE MSR for silicon debug.
12	FMA	A value of 1 indicates the processor supports FMA extensions using YMM state.
13	CMPXCHG16B	**CMPXCHG16B Available.** A value of 1 indicates that the feature is available. See the "CMPXCHG8B/CMPXCHG16B—Compare and Exchange Bytes" section in this chapter for a description.
14	xTPR Update Control	**xTPR Update Control.** A value of 1 indicates that the processor supports changing IA32_MISC_ENABLE[bit 23].
15	PDCM	**Perfmon and Debug Capability:** A value of 1 indicates the processor supports the performance and debug feature indication MSR IA32_PERF_CAPABILITIES.
16	Reserved	Reserved
17	PCID	**Process-context identifiers.** A value of 1 indicates that the processor supports PCIDs and that software may set CR4.PCIDE to 1.
18	DCA	A value of 1 indicates the processor supports the ability to prefetch data from a memory mapped device.
19	SSE4.1	A value of 1 indicates that the processor supports SSE4.1.
20	SSE4.2	A value of 1 indicates that the processor supports SSE4.2.
21	x2APIC	A value of 1 indicates that the processor supports x2APIC feature.
22	MOVBE	A value of 1 indicates that the processor supports MOVBE instruction.
23	POPCNT	A value of 1 indicates that the processor supports the POPCNT instruction.
24	TSC-Deadline	A value of 1 indicates that the processor's local APIC timer supports one-shot operation using a TSC deadline value.
25	AESNI	A value of 1 indicates that the processor supports the AESNI instruction extensions.
26	XSAVE	A value of 1 indicates that the processor supports the XSAVE/XRSTOR processor extended states feature, the XSETBV/XGETBV instructions, and XCR0.
27	OSXSAVE	A value of 1 indicates that the OS has set CR4.OSXSAVE[bit 18] to enable XSETBV/XGETBV instructions to access XCR0 and to support processor extended state management using XSAVE/XRSTOR.
28	AVX	A value of 1 indicates the processor supports the AVX instruction extensions.

Table 3-20. More on Feature Information Returned in the EDX Register

Bit #	Mnemonic	Description
0	FPU	**Floating Point Unit On-Chip.** The processor contains an x87 FPU.
1	VME	**Virtual 8086 Mode Enhancements.** Virtual 8086 mode enhancements, including CR4.VME for controlling the feature, CR4.PVI for protected mode virtual interrupts, software interrupt indirection, expansion of the TSS with the software indirection bitmap, and EFLAGS.VIF and EFLAGS.VIP flags.
2	DE	**Debugging Extensions.** Support for I/O breakpoints, including CR4.DE for controlling the feature, and optional trapping of accesses to DR4 and DR5.
3	PSE	**Page Size Extension.** Large pages of size 4 MByte are supported, including CR4.PSE for controlling the feature, the defined dirty bit in PDE (Page Directory Entries), optional reserved bit trapping in CR3, PDEs, and PTEs.
4	TSC	**Time Stamp Counter.** The RDTSC instruction is supported, including CR4.TSD for controlling privilege.
5	MSR	**Model Specific Registers RDMSR and WRMSR Instructions.** The RDMSR and WRMSR instructions are supported. Some of the MSRs are implementation dependent.
6	PAE	**Physical Address Extension.** Physical addresses greater than 32 bits are supported: extended page table entry formats, an extra level in the page translation tables is defined, 2-MByte pages are supported instead of 4 Mbyte pages if PAE bit is 1.
7	MCE	**Machine Check Exception.** Exception 18 is defined for Machine Checks, including CR4.MCE for controlling the feature. This feature does not define the model-specific implementations of machine-check error logging, reporting, and processor shutdowns. Machine Check exception handlers may have to depend on processor version to do model specific processing of the exception, or test for the presence of the Machine Check feature.
8	CX8	**CMPXCHG8B Instruction.** The compare-and-exchange 8 bytes (64 bits) instruction is supported (implicitly locked and atomic).
9	APIC	**APIC On-Chip.** The processor contains an Advanced Programmable Interrupt Controller (APIC), responding to memory mapped commands in the physical address range FFFE0000H to FFFE0FFFH (by default - some processors permit the APIC to be relocated).
10	Reserved	Reserved
11	SEP	**SYSENTER and SYSEXIT Instructions.** The SYSENTER and SYSEXIT and associated MSRs are supported.
12	MTRR	**Memory Type Range Registers.** MTRRs are supported. The MTRRcap MSR contains feature bits that describe what memory types are supported, how many variable MTRRs are supported, and whether fixed MTRRs are supported.
13	PGE	**Page Global Bit.** The global bit is supported in paging-structure entries that map a page, indicating TLB entries that are common to different processes and need not be flushed. The CR4.PGE bit controls this feature.
14	MCA	**Machine Check Architecture.** The Machine Check Architecture, which provides a compatible mechanism for error reporting in P6 family, Pentium 4, Intel Xeon processors, and future processors, is supported. The MCG_CAP MSR contains feature bits describing how many banks of error reporting MSRs are supported.
15	CMOV	**Conditional Move Instructions.** The conditional move instruction CMOV is supported. In addition, if x87 FPU is present as indicated by the CPUID.FPU feature bit, then the FCOMI and FCMOV instructions are supported
16	PAT	**Page Attribute Table.** Page Attribute Table is supported. This feature augments the Memory Type Range Registers (MTRRs), allowing an operating system to specify attributes of memory accessed through a linear address on a 4KB granularity.
17	PSE-36	**36-Bit Page Size Extension.** 4-MByte pages addressing physical memory beyond 4 GBytes are supported with 32-bit paging. This feature indicates that upper bits of the physical address of a 4-MByte page are encoded in bits 20:13 of the page-directory entry. Such physical addresses are limited by MAXPHYADDR and may be up to 40 bits in size.
18	PSN	**Processor Serial Number.** The processor supports the 96-bit processor identification number feature and the feature is enabled.
19	CLFSH	**CLFLUSH Instruction.** CLFLUSH Instruction is supported.
20	Reserved	Reserved

Table 3-20. More on Feature Information Returned in the EDX Register (Contd.)

Bit #	Mnemonic	Description
21	DS	**Debug Store.** The processor supports the ability to write debug information into a memory resident buffer. This feature is used by the branch trace store (BTS) and precise event-based sampling (PEBS) facilities (see Chapter 23, "Introduction to Virtual-Machine Extensions," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*).
22	ACPI	**Thermal Monitor and Software Controlled Clock Facilities.** The processor implements internal MSRs that allow processor temperature to be monitored and processor performance to be modulated in predefined duty cycles under software control.
23	MMX	**Intel MMX Technology.** The processor supports the Intel MMX technology.
24	FXSR	**FXSAVE and FXRSTOR Instructions.** The FXSAVE and FXRSTOR instructions are supported for fast save and restore of the floating point context. Presence of this bit also indicates that CR4.OSFXSR is available for an operating system to indicate that it supports the FXSAVE and FXRSTOR instructions.
25	SSE	**SSE.** The processor supports the SSE extensions.
26	SSE2	**SSE2.** The processor supports the SSE2 extensions.
27	SS	**Self Snoop.** The processor supports the management of conflicting memory types by performing a snoop of its own cache structure for transactions issued to the bus.
28	HTT	**Max APIC IDs reserved field is Valid.** A value of 0 for HTT indicates there is only a single logical processor in the package and software should assume only a single APIC ID is reserved. A value of 1 for HTT indicates the value in CPUID.1.EBX[23:16] (the Maximum number of addressable IDs for logical processors in this package) is valid for the package.
29	TM	**Thermal Monitor.** The processor implements the thermal monitor automatic thermal control circuitry (TCC).
30	Reserved	Reserved
31	PBE	**Pending Break Enable.** The processor supports the use of the FERR#/PBE# pin when the processor is in the stop-clock state (STPCLK# is asserted) to signal the processor that an interrupt is pending and that the processor should return to normal operation to handle the interrupt. Bit 10 (PBE enable) in the IA32_MISC_ENABLE MSR enables this capability.

INPUT EAX = 2: TLB/Cache/Prefetch Information Returned in EAX, EBX, ECX, EDX

When CPUID executes with EAX set to 2, the processor returns information about the processor's internal TLBs, cache and prefetch hardware in the EAX, EBX, ECX, and EDX registers. The information is reported in encoded form and fall into the following categories:

- The least-significant byte in register EAX (register AL) will always return 01H. Software should ignore this value and not interpret it as an informational descriptor.
- The most significant bit (bit 31) of each register indicates whether the register contains valid information (set to 0) or is reserved (set to 1).
- If a register contains valid information, the information is contained in 1 byte descriptors. There are four types of encoding values for the byte descriptor, the encoding type is noted in the second column of Table 3-21. Table 3-21 lists the encoding of these descriptors. Note that the order of descriptors in the EAX, EBX, ECX, and EDX registers is not defined; that is, specific bytes are not designated to contain descriptors for specific cache, prefetch, or TLB types. The descriptors may appear in any order. Note also a processor may report a general descriptor type (FFH) and not report any byte descriptor of "cache type" via CPUID leaf 2.

ДОДАТОК Б.

СПЕЦИФІКАЦІЯ КОМАНДИ CPUID ДЛЯ ПРОЦЕСОРІВ AMD

Окремі фрагменти з офіційної документації процесорів AMD [12].

CPUID Fn0000 0001 EDX Feature Identifiers

This function contains the following miscellaneous feature identifiers.

Bits	Description
31:29	Reserved.
28	**HTT: hyper-threading technology**. Indicates either that there is more than one thread per core or more than one core per processor
27	Reserved.
26	**SSE2: SSE2 instruction support**
25	**SSE: SSE instruction support**
24	**FXSR: FXSAVE and FXRSTOR instructions**. See “FXSAVE” and “FXRSTOR” in APM4.
23	**MMX: MMX™ instructions**
22:20	Reserved.
19	**CLFSH: CLFLUSH instruction support**
18	Reserved.
17	**PSE36: page-size extensions**. The PDE[20:13] supplies physical address [39:32]
16	**PAT: page attribute table**.
15	**CMOV: conditional move instructions**
14	**MCA: machine check architecture**. See “Machine Check Mechanism” in APM2.
13	**PGE: page global extension**.
12	**MTRR: memory-type range registers**
11	**SysEnterSysExit: SYSENTER and SYSEXIT instructions**.
10	Reserved.
9	**APIC: advanced programmable interrupt controller**. Indicates APIC exists and is enabled.
8	**CMPXCHG8B: CMPXCHG8B instruction**.
7	**MCE: Machine check exception**.
6	**PAE: physical-address extensions**. Indicates support for physical addresses > 32b. Number of physical address bits above 32b is implementation specific.
5	**MSR: AMD model-specific registers**. Indicates support for AMD model-specific registers (MSRs), with RDMSR and WRMSR instructions.
4	**TSC: time stamp counter**. RDTSC and RDTSCP instruction support.
3	**PSE: page-size extensions**.
2	**DE: debugging extensions**.
1	**VME: virtual-mode enhancements**. CR4.VME, CR4.PVI, software interrupt indirection, expansion of the TSS with the software, indirection bitmap, EFLAGS.VIF, EFLAGS.VIP.
0	**FPU: x87 floating point unit on-chip**

CPUID Fn0000_000[4:2] Reserved

Bits	Description
31:0	Reserved.

CPUID Fn0000_0005_EAX Monitor/MWait

This function contains the feature identifiers for the MONITOR and MWAIT instructions.

Bits	Description
31:16	Reserved.
15:0	**MonLineSizeMin: smallest monitor-line size in bytes**.

CPUID Fn0000 0005 EBX Monitor/MWait

Bits	Description
31:16	Reserved.
15:0	**MonLineSizeMax: largest monitor-line size in bytes**.

CPUID Fn0000 0005 ECX Monitor/MWait

Bits	Description
31:2	Reserved.
1	**IBE: interrupt break-event**. Indicates MWAIT can use ECX bit 0 to allow interrupts to cause an exit from the monitor event pending state, even if EFLAGS.IF=0.
0	**EMX: enumerate MONITOR/MWAIT extensions**: Indicates enumeration MONITOR/MWAIT extensions are supported.
Bits	Description
31:0	Reserved.

Далі пропущені функції **Fn0000_0006 - Fn0000_000D**

CPUID Fn8000_0000_EAX Largest Extended Function Number

Bits	Description
31:0	**LFuncExt: largest extended function**. The largest CPUID extended function input value supported by the processor implementation.

CPUID Fn8000_0000_E[D,C,B]X Processor Vendor

Register	Value	Description
CPUID Fn8000_0000_EBX	6874_7541h	The ASCII characters “h t u A”.
CPUID Fn8000_0000_ECX	444D_4163h	The ASCII characters “D M A c”.
CPUID Fn8000_0000_EDX	6974_6E65h	The ASCII characters “i t n e”.

CPUID Fn8000_0001_EAX AMD Family, Model, Stepping

Bits	Description
31:0	See: CPUID Fn0000_0001_EAX.

CPUID Fn8000_0001_EBX BrandId Identifier

This function returns the extended brand ID field.

Bits	Description
31:28	**PkgType: package type**. If (Family[7:0] >= 10h) then the definition of PkgType is contained in the processor BKDG. If (Family[7:0]<10h) then the definition of PkgType is reserved.
27:16	Reserved.
15:0	**BrandId: brand ID**. This field, in conjunction with CPUID Fn0000 0001 EBX[8BitBrandId], is used by BIOS to generate the processor name string. See your processor revision guide for how to

CPUID Fn8000 0001 ECX Feature Identifiers

This function contains the following miscellaneous feature identifiers.

Bits	Description
31:23	Reserved.
22	**TopologyExtensions: topology extensions support**. Indicates support for CPUID Fn8000_001D_EAX_x[N:0]-CPUID Fn8000_001E_EDX.
21	**TBM: trailing bit manipulation instruction support**.
20	Reserved.
19	**NodeId**. Indicates support for MSRC001_100C[NodeId, NodesPerProcessor].
18	Reserved.
17	Reserved.
16	**FMA4: 4-operand FMA instruction support**
15	**LWP: lightweight profiling support**
14	Reserved.
13	**WDT: watchdog timer support**
12	**SKINIT: SKINIT and STGI are supported, independent of the value of MSRC000_0080[SVME].**
11	**XOP: extended operation support**. See APM6.
10	**IBS: instruction based sampling**.
9	**OSVW: OS visible workaround**. Indicates OS-visible workaround support.
8	**3DNowPrefetch: PREFETCH and PREFETCHW instruction support**
7	**MisAlignSse: misaligned SSE mode**.
6	**SSE4A: EXTRQ, INSERTQ, MOVNTSS, and MOVNTSD instruction support**. .
5	**ABM: advanced bit manipulation**. LZCNT instruction support
4	**AltMovCr8: LOCK MOV CR0 means MOV CR8**
3	**ExtApicSpace: extended APIC space**. This bit indicates the presence of extended APIC register space starting at offset 400h from the “APIC Base Address Register,” as specified in the BKDG.
2	**SVM: secure virtual machine**
1	**CmpLegacy: core multi-processing legacy mode**.
0	**LahfSahf: LAHF and SAHF instruction support in 64-bit mode**.

CPUID Fn8000 0001 EDX Feature Identifiers

This function contains the following miscellaneous feature identifiers.

Bits	Description
31	**3DNow: 3DNow!™ instructions**. See Appendix D “Instruction Subsets and CPUID Feature Sets” in APM3.
30	**3DNowExt: AMD extensions to 3DNow! instructions**
29	**LM: long mode**. See “Processor Initialization and Long-Mode Activation” in APM2.
28	Reserved.
27	**RDTSCP: RDTSCP instruction**
26	**PagelGB: 1-GB large page support**
25	**FFXSR: FXSAVE and FXRSTOR instruction optimizations**
24	**FXSR: FXSAVE and FXRSTOR instructions**. Same as CPUID Fn0000_0001_EDX[FXSR].
23	**MMX: MMX™ instructions**. Same as CPUID Fn0000_0001_EDX[MMX].
22	**MmxExt: AMD extensions to MMX instructions**
21	Reserved.
20	**NX: no-execute page protection**.
19:18	Reserved.
17	**PSE36: page-size extensions**. Same as CPUID Fn0000 0001 EDX[PSE36].
16	**PAT: page attribute table**. Same as CPUID Fn0000 0001 EDX[PAT].
15	**CMOV: conditional move instructions**. Same as CPUID Fn0000 0001 EDX[CMOV].
14	**MCA: machine check architecture**. Same as CPUID Fn0000 0001 EDX[MCA].
13	**PGE: page global extension**. Same as CPUID Fn0000 0001 EDX[PGE].
12	**MTRR: memory-type range registers**. Same as CPUID Fn0000 0001 EDX[MTRR].
11	**SysCallSysRet: SYSCALL and SYSRET instructions**
10	Reserved.
9	**APIC: advanced programmable interrupt controller**. Same as CPUID
8	**CMPXCHG8B: CMPXCHG8B instruction**. Same as CPUID
7	**MCE: machine check exception**. Same as CPUID Fn0000 0001 EDX[MCE].
6	**PAE: physical-address extensions**. Same as CPUID Fn0000 0001 EDX[PAE].
5	**MSR: AMD model-specific registers**. Same as CPUID Fn0000 0001 EDX[MSR].
4	**TSC: time stamp counter**. Same as CPUID Fn0000 0001 EDX[TSC].
3	**PSE: page-size extensions**. Same as CPUID Fn0000 0001 EDX[PSE].
2	**DE: debugging extensions**. Same as CPUID Fn0000 0001 EDX[DE].
1	**VME: virtual-mode enhancements**. Same as CPUID Fn0000 0001 EDX[VME].
0	**FPU: x87 floating-point unit on-chip**. Same as CPUID Fn0000 0001 EDX[FPU].

CPUID Fn8000_000[4:2]_E[D,C,B,A]X Processor Name String Identifier

Три функції з Fn8000_0002 до Fn8000_0004 повертають ASCII – рядок 48 символів з 0 в кінці, що відповідає імені процесору. Послідовність з 48 символів відсортована від першої до останньої.

CPUID Fn8000_0005_EAX L1 Cache and TLB Identifiers

Ця функція містить дані про кеш першого рівня процесора і характеристики TLB для кожного ядра.

Поля асоціативності кодуються так:

00h: Reserved.

01h: Direct mapped.

02h-FEh: Associativity. (e.g., 04h = 4-way associative.)

FFh: Fully associative.

Bits	Description
31:24	**L1DTlb2and4MAssoc: data TLB associativity for 2 MB and 4 MB pages**. Data TLB associativity for 2-MB and 4-MB pages. See: CPUID Fn8000_0005_EDX[L1IcAssoc].
23:16	**L1DTlb2and4MSize: data TLB number of entries for 2 MB and 4 MB pages**. Data TLB number of entries for 2-MB and 4-MB pages. The value returned is for the number of entries available for the 2-MB page size; 4-MB pages require two 2-MB entries, so the number of entries available for the 4-MB page size is one-half the returned value.
15:8	**L1ITlb2and4MAssoc: instruction TLB associativity for 2 MB and 4 MB pages**. Instruction TLB associativity for 2-MB and 4-MB pages. See: CPUID Fn8000 0005 EDX[L1IcAssoc].
7:0	**L1ITlb2and4MSize: instruction TLB number of entries for 2 MB and 4 MB pages**. Instruction TLB number of entries for 2-MB and 4-MB pages. The value returned is for the number of entries available for the 2-MB page size; 4-MB pages require two 2-MB entries, so the number of entries available for the 4-MB page size is one-half the returned value.

CPUID Fn8000 0005 EBX L1 Cache and TLB Identifiers

This provides the processor's first level cache and TLB characteristics for each core.

Bits	Description
31:24	**L1DTlb4KAssoc: data TLB associativity for 4 KB pages**. Data TLB associativity for 4 KB pages. See: CPUID Fn8000_0005_EDX[L1IcAssoc].
23:16	**L1DTlb4KSize: data TLB number of entries for 4 KB pages**. Data TLB number of entries for 4 KB pages.
15:8	**L1ITlb4KAssoc: instruction TLB associativity for 4 KB pages**. Instruction TLB associativity for 4 KB pages.See: CPUID Fn8000_0005_EDX[L1IcAssoc].
7:0	**L1ITlb4KSize: instruction TLB number of entries for 4 KB pages**. Instruction TLB number of entries for 4 KB pages.

CPUID Fn8000 0005 ECX L1 Cache and TLB Identifiers

This provides the processor’s first level cache and TLB characteristics for each core.

Bits	Description
31:24	**L1DcSize: L1 data cache size in KB**. L1 data cache size in KB.
23:16	**L1DcAssoc: L1 data cache associativity**. L1 data cache associativity. See: CPUID Fn8000_0005_EDX[L1IcAssoc].
15:8	**L1DcLinesPerTag: L1 data cache lines per tag**. L1 data cache lines per tag.
7:0	**L1DcLineSize: L1 data cache line size in bytes**. L1 data cache line size in bytes.

CPUID Fn8000_0005_EDX L1 Cache and TLB Identifiers

This provides the processor’s first level cache and TLB characteristics for each core.

Bits	Description
31:24	**L1IcSize: L1 instruction cache size KB**. L1 instruction cache size KB.
23:16	**L1IcAssoc: L1 instruction cache associativity**. L1 instruction cache associativity. Bits Description 00h Reserved 01h 1 way (direct mapped) FEh-02h Specifies the associativity; e.g., 04h would indicate a 4-way associativity. FFh Fully associative
15:8	**L1IcLinesPerTag: L1 instruction cache lines per tag**. L1 instruction cache lines per tag.
7:0	**L1IcLineSize: L1 instruction cache line size in bytes**. L1 instruction cache line size in bytes.

CPUID Fn8000_0006_EAX L2 TLB Identifiers

Функція містить кеш другого рівня процесора і характеристики TLB для кожного ядра. Регістр EDX містить характеристики кеш-пам'яті третього рівня процесора, які використовуються усіма ядрами процесора. Поля асоціативності кодуються так:

Table 4: L2/L3 Cache and TLB Associativity Field Definition

Associativity [3:0]	**Definition**
0h	L2/L3 cache or TLB is disabled.
1h	Direct mapped.
2h	2-way associative.
4h	4-way associative.
6h	8-way associative.
8h	16-way associative.
Ah	32-way associative.
Bh	48-way associative.
Ch	64-way associative.
Dh	96-way associative.
Eh	128-way associative.
Fh	Fully associative.

All other encodings are reserved.

Bits	Description
31:28	**L2DTlb2and4MAssoc: L2 data TLB associativity for 2 MB and 4 MB pages**. L2 data TLB associativity for 2-MB and 4-MB pages. See Table 4.
27:16	**L2DTlb2and4MSize: L2 data TLB number of entries for 2 MB and 4 MB pages**. L2 data TLB number of entries for 2-MB and 4-MB pages. The value returned is for the number of entries available for the 2 MB page size; 4 MB pages require two 2 MB entries, so the number of entries available
15:12	**L2ITlb2and4MAssoc: L2 instruction TLB associativity for 2 MB and 4 MB pages**. L2 instruction TLB associativity for 2-MB and 4-MB pages. See Table 4.
11:0	**L2ITlb2and4MSize: L2 instruction TLB number of entries for 2 MB and 4 MB pages**. L2 instruction TLB number of entries for 2-MB and 4-MB pages. The value returned is for the number of entries available for the 2 MB page size; 4 MB pages require two 2 MB entries, so the number of en-

CPUID Fn8000_0006_EBX L2 TLB Identifiers

See CPUID Fn8000 0006 EAX.

Bits	Description
31:28	**L2DTlb4KAssoc: L2 data TLB associativity for 4 KB pages**. L2 data TLB associativity for 4-KB pages. See Table 4.
27:16	**L2DTlb4KSize: L2 data TLB number of entries for 4 KB pages**. L2 data TLB number of entries for 4-KB pages.
15:12	**L2ITlb4KAssoc: L2 instruction TLB associativity for 4 KB pages**. L2 instruction TLB associativity for 4-KB pages. See Table 4.
11:0	**L2ITlb4KSize: L2 instruction TLB number of entries for 4 KB pages**. L2 instruction TLB number of entries for 4-KB pages.

CPUID Fn8000_0006_ECX L2 Cache Identifiers

Bits	Description
31:16	**L2Size: L2 cache size in KB**.
15:12	**L2Assoc: L2 cache associativity**.

Bits	Description	Bits	Description
0000b	Disabled.	1000b	16 ways
0001b	1 way (direct mapped)	1001b	Reserved.
0010b	2 ways	1010b	32 ways
0011b	Reserved.	1011b	48 ways
0100b	4 ways	1100b	64 ways
0101b	Reserved.	1101b	96 ways
0110b	8 ways	1110b	128 ways
0111b	Reserved	1111b	Fully associative

11:8	**L2LinesPerTag: L2 cache lines per tag**.
7:0	**L2LineSize: L2 cache line size in bytes**.

CPUID Fn8000 0006 EDX L3 Cache Identifiers

Bits	Description
31:18	**L3Size: L3 cache size**. Specifies the L3 cache size is within the following range: (L3Size[31:18] * 512KB) ≤ L3 cache size < ((L3Size[31:18]+1) * 512KB).
17:16	Reserved.
15:12	**L3Assoc: L3 cache associativity**. L3 cache associativity. See Table 4.
11:8	**L3LinesPerTag: L3 cache lines per tag**.
7:0	**L3LineSize: L3 cache line size in bytes**.

CPUID Fn8000_0007_E[C,B,A]X Advanced Power Management Information

Bits	Description
31:0	Reserved.

CPUID Fn8000_0007_EDX Advanced Power Management Information

Ця функція забезпечує розширені ідентифікатори функцій управління живленням

Bits	Description
31:11	Reserved.
10	**EffFreqRO: read-only effective frequency interface**. 1=Indicates presence of MSRC000 00E7 [Read-Only Max Performance Frequency Clock Count (MPerfReadOnly)] and MSRC000 00E8 [Read-Only Actual Performance Frequency Clock Count (APerfReadOnly)].
9	**CPB: core performance boost**.
8	**TscInvariant: TSC invariant**. The TSC rate is ensured to be invariant across all P-States, C-States, and stop grant transitions (such as STPCLK Throttling); therefore the TSC is suitable for use as a source of time. 0 = No such guarantee is made and software should avoid attempting to use the TSC

7	**HwPstate: hardware P-state control**. MSRC001_0061 [P-state Current Limit], MSRC001_0062 [P-state Control] and MSRC001_0063 [P-state Status] exist.
6	**100MHzSteps: 100 MHz multiplier Control**.
5	Reserved.
4	**TM: hardware thermal control (HTC)**.
3	**TTP: THERMTRIP.**
2	**VID: Voltage ID control**. Function replaced by HwPstate.
1	**FID: Frequency ID control**. Function replaced by HwPstate.
0	**TS: Temperature sensor**.

CPUID Fn8000_0008_EAX Long Mode Address Size Identifiers

Функція повертає інформацію про максимальну фізичну та лінійну ширину адреси (у бітах), яку підтримує процесор. Повідомлена ширина є максимально підтримуваною в будь-якому режимі. Для процесорів, що підтримують довгий режим, повідомлений розмір не залежить від того, чи ввімкнено довгий режим

Bits	Description
31:24	Reserved.
23:16	**GuestPhysAddrSize: maximum guest physical byte address size in bits**. This number applies only to guests using nested paging. When this field is zero, refer to the PhysAddrSize field for the maximum guest physical address size. See “Secure Virtual Machine” in APM2.
15:8	**LinAddrSize: Maximum linear byte address size in bits**.
7:0	**PhysAddrSize: Maximum physical byte address size in bits**. When GuestPhysAddrSize is zero, this field also indicates the maximum guest physical address size.

CPUID Fn8000 0008 EBX Reserved

Bits	Description
31:0	Reserved.

CPUID Fn8000_0008_ECX APIC ID Size and Core Count

Функція надає інформацію про кількість ядер, що підтримуються процесором.

Bits	Description
31:16	Reserved.
15:12	**ApicIdCoreIdSize: APIC ID size**. Кількість бітів у початковому значенні APIC20 [ApicId], які вказують ідентифікатор ядра в процесорі. Нульове значення вказує на те, що для виведення максимальної кількості ядер потрібно використовувати застарілі методи. Розмір цього поля визначає максимальну кількість ядер (MNC), яку теоретично міг би підтримувати процесор, а не фактичну кількість ядер, які насправді реалізовані .

	if (ApicIdCoreIdSize[3:0] == 0){
	// Used by legacy dual-core/single-core processors MNC = CPUID Fn8000_0008_ECX[NC] + 1;
	} else {
	// use ApicIdCoreIdSize[3:0] field MNC = (2 A ApicIdCoreIdSize[3:0]);
	}

11:8	Reserved.
7:0	**NC: number of physical cores - 1**. The number of cores in the processor is NC+1 (e.g., if NC=0, then there is one core)

CPUID Fn8000_0008_EDX Reserve

Bits	Description
31:0	Reserved.

Далі пропущені функції **Fn8000_0009 - Fn8000_001E.**

НАВЧАЛЬНО-МЕТОДИЧНЕ ВИДАННЯ

Методичні вказівки
до практичних (лабораторних) робіт
з дисципліни «Системне програмне забезпечення»
для студентів спеціальності 123 Комп'ютерна інженерія
усіх форм навчання.

Комп'ютерний набір і верстка:
Шамаєв Віталій Віталійович

Укладачі:
Шамаєв Віталій Віталійович
Ковальов Сергій Олександрович
Любимов Артем Сергійович

Методичні вказівки розроблено за участю Ольги Георгіївни Шевченко.
Автори щиро вдячні їй за співпрацю та надані матеріали.

ДВНЗ «Донецький національний технічний університет»
43003, Україна, Волинська область, м. Луцьк, вул. Потебні, 56.