Сьогодні ми розпочинаємо новий блок у вивчені нашої дисципліни.

Це блок який буде пов’язано з використанням мови ассемблер для вирішення тих чи інших задач.

Мова асемблер – не та мова, на якій пишеться програма цілком.

Дозволяє:

- Написати просту програму з використанням мінімальної кількості ресурсів
- Та програму яка буде працювати дуже швидко.
  За рахунок того, що ми будемо працювати з ресурсами напряму

- Написання елементів програмного забезпечення драйверів
- Написання елементів програмного забезпечення необхідного для реалізації певних компонентів операційних систем

Важливо розуміти, що асемблер потрібен фахівціям які працюють з невеликими пристроями, наприклад, інтернету речей. В них досить часто немає інтегрованих середовищ написання програм і потрібно писати програми короткі, але продуктивні. Немає великої кількості ресурсів, наприклад гігабайтів оперативної пам’яті та швидкодіючих процесорів. Тому мова асемблер в сфері інтернету речей застосовується досить широко.

Окрім того, знання мови асемблер потрібно для проведення дезасемблювання з подальшим аналізом відповідних виконуваних програмних продуктів. Відповідно, якщо ми в системі Віндовс хочемо проаналізувати якийсь шкідливий код, зрозуміти як він працює, до чого звертається, з якими ресурсами, з якими файлами, папками і таке інше, які операції виконує – ми, нажаль не можемо цей код повернути до тієї мови на якій він написаний, до компіляції, мова С, чи ще перекомпільовано з якоїсь іншої мови. Ми можемо лише повернути його до вигляду асемблерного і далі все аналізувати, як з ним працювати. Тому знання мови асемблер є дуже важливим.

Перелічувати питання лекції

Синтаксис мови асемблер

Програма на асемблер повинна бути відкомпельована.

Тобто перетворена із текстового запису у виконуваний код. У певні числові команди, які будуть поступати на процесор комп’ютера і далі виконуватись.

В чому специфіка компіляції – відкомпільована програма, після трансляції команд, підключення бібліотек може виконуватись самостійно. І їй не потрібні ніякі додаткові засоби щоб бути запущеною.

Прикладами компільованих мов є С, С++ та деякі інші.

Є мови трансльовані (або скриптові). Відповідно вони вимагають для запуску наявності транслятора. Прикладом такої мови може бути використання мови Python. Яка предбачає, що для запуску написаної на пайтоні програми обов’язково має бути транслятор – тобто програма яка буде транслювати програмний код у виконуваний код. Іншими мовами є HTML та джаваскрипт. Бо для перегляду, наприклад, програми на JS потрібно обов’язково запускати браузер, який буде запускати ті чи інші елементи нашої програми.

Звичайно трансльовану програму можна також перетворити у виконувану програму, але це передбачає включення до неї самого транслятора, як складової програми.

Повернемось для програми на мові асемблер.

Програма на асемблер – це сукупність блоків пам'яті, або сегментів (сегменти коду, сегменти даних і таке інше.

Кожен сегмент програми складається з певних речень мови – сукупності відповідних рядків програми.

Якими є синтаксичні конструкції мови Асемблер:

- Команди – мінімальний елемент програми, символьні аналоги машинних команд.

По суті це числова послідовність, яка поступає на виконавчий блок в ядро процесора і процесор отримуючи послідовність розуміє, яку операцію потрібно виконати.

- Макрокоманди – певним чином оформленні речення програми. Досить часто це певна сукупність окремих команд. Яка в процесі компіляції може замінюватись на складові команди.
- Директиви – спеціальні конструкції мови, які вказують компіляторові, як проводити ті чи інші дії, або операції. Наприклад, можна вказати, компілювати для 32х бітної або для 64х бітної операційної системи. Відповідно там будуть свої особливості роботи з регістрами, з пам'яттю тощо.
- Коментар – довільна текстова/символьна послідовність, яка використовується для пояснення роботи програми.

Слід звернути увагу, що в мовах більш високого рівня по тексту команди можна більш-менш зрозуміти, які дії виконуються – де є початок циклу, де кінець, перевірка умови чи яка операція виконується. В асемблері це все набагато складніше, тому коментарі грають дуже важливу роль в поясненні та розумінні коду програми та процесів, запрограмованих програмістом.

Розглянемо, як записується речення мови асемблера

Команда (Макрокоманда, директива) ;коментар

Якщо розглядати більш детально, то команду можна записати у наступному вигляді:

Мітка : **КОД ОПЕРАЦІЇ** {Операнд1, Операнд 2} ; коментар

Код операції – мнемонічне (скорочене) позначення машинної команди

Операнд – параметр команди.

start1: ADD AX, BX ; Додавання значення регістра BX до значення регістра AX із збереженням результату в регістрі AX

Алфавіт Ассемблер:

Літери латинського алфавіту (A-Z, a-z)

Для мови асемблер не суттєво, які літери ми використовуємо для позначення команд.

Але є така практика, що команди пишуть маленькими літерами, а директиви – великими.

Але при поєднанні з іншими мовами, а ми будемо працювати з мовою С, то для мови С є суттєвим використання змінної з великої чи з маленької. Тому звертає на цей момент Вашу увагу, щоб потім не було певних проблем при написанні коду.

Цифри (0-9)

Спеціальні символи (?, &, $, @ …)

Роздільники (,), [,], {, }, <,>, +, -, /, * …

Зверну Вашу увагу, що в мові асемблер немає арифметичної операції заданої знаком +, як у звичайному програмуванні.

Ключові слова:

- назви регістрів (AL, AH, AX, EAX, BL, ….)

восьмибітних, шістнадцятибітних, 32бітних і так далі.

Назви регістрів не можна використовувати як змінні.

Рекомендується змінні з мови С позначати маленькими літерами, а регістри в асемблері – великими.

оператори (BYTE, WORD, FAR, NEAR…)

far, near – виклики відповідних функцій, використання ближньої та дальньої пам’яті

- назви команд (ADD, NEG, INC, LOOP, JE …)

зазвичай команди є скороченнями чи абревіатурами від певних англійських слів, що описують дію цієї команди

чому команди такі прості і короткі – оскільки коли розроблювалась ця мова, для збереження тексту програм потрібно було економити пам’ять на носіях. Наприклад я пам’ятаю ще великі дискети об’ємом 720 кілобайт, ну і ви всі швидше за все бачили маленькі, 3.5 дюймові дискети об’ємом в півтора мегабайти. Зараз коли є флешки на десятки і сотні гігабайт та вінчестери на терабайти – це вже не є проблемою.

JE – jump if equal

Ідентифікатори – послідовності символів, що використовуються для позначення змінних (a1, B2) та міток (start1)

З міткою все більш-менш зрозуміло, вони дозволяють нам ідентифікувати початок, кінець чи якісь проміжні елементи коду.

Також мітки використовуються для переходів та організації циклів.

Щодо змінних в мові асемблер не має звичного поняття змінна. Мова асемблер взаємодіє з певною ділянкою пам’яті. Тобто, ще раз, в асемблері змінна – це позначення ділянки пам’яті. Тобто для нього важливою є довжина ділянки – 1 байт, 2 байти, 4 байти і так далі. А що в середині цієї ділянки асемблер не розуміє. І вже програміст має розуміти, з чим він працює – з цілим числом чи дійсним числом, зі знаковим чи беззнаковим числом і так далі.

Ланцюжки символів: ‘text’, “text”

Це певні послідовності символів, які записуються у одинарних або подвійних лапках. По суті – це масиви символів, які позначають ті чи інші символьних змінних.

Цілі числа

Отже з цим синтаксисом ми будемо працювати

Операнди команди – об’єкти, над якими або за допомогою яких можуть проводитися дії, що задаються відповідними командами.

Що може виступати у вигляді операнда команди

Операнди:

- регістр; 8 біт, 16 біт і 32 біти тощо
- пам’ять; це будуть по сути змінні, описані на мові С
- безпосередні операнди (як правило, це цілі числа).

Команди можуть бути: одноопераднимиими, двоопераднимиими та іноді триопераднимиими.

Одноопарандні команди:

Регістр або пам’ять

NEG EAX – що робить?

Як правило, команди Ассемблер двоопарандні:

Регістр – Регістр	SUB AX, BX
Регістр – Пам’ять	ADD AH, a
Пам’ять – Регістр	ADD a, AH

Команди пам’ять – пам’ять немає, ми повинні скористатись проміжним елементом – регістром

Регістр – Безпосередній операнд	ADD BL, 123
Пам’ять – Безпосередній операнд	ADD a, 123

Звертаю увагу, що ми не можемо внести у 8-бітний регістр значення, яке займає більше ніж 8 біт. Тому що перші 8 біт внесуться, а старщі розряди – 9+ будуть відкинуті.

Триопарандні команди – з'явились у більш сучасних версіях асемблера, але в більшості випадків вони є макрокомандами, що скадаються з декількох двоопарандних команд.

ADD AX, BX, CX – ADD AX, BX MOV CX, AX

За таким принципом можуть бути написані і «більшеопарандні» макрокоманди

Типи даних ассемблер

Це не звичні для мови високого рівня поняття.

Це значення певної довжини

Основні типи даних ассемблер

- Байт, byte (8 розрядів, бітів)
- Слово, word (2 байти) = (Старший байт, Молодший)
- Подвійне слово, double word (4 байти) = (Старше, Молодше)
- Зчетверене слово, quad word (8 байтів)
- Упакований тип даних, double-quad, xmmword (16 байтів)
- Десятибайтний, ten bytes (10 байтів)

Цілі типи даних (8, 16, 32, 64, 128)

- знаковий
- беззнаковий

Дійсні типи даних. Завжди знакові. (32, 64, 80, 128, …)

Ланцюжки символів – можуть використовуватись для роботи з пам'яттю,

Дані MMX-розширень тощо.

Основні арифметичні команди мови Ассемблер

Правила арифметики мови Ассемблер

1. Для виконання операцій додавання та віднімання операнди повинні мати однакову довжину:
   8 біт +/- 8 бітів = 8 бітів
   16 бітів +/- 16 бітів = 16 бітів
   32 біти +/- 32 біти = 32 біти
   64 біти +/- 64 біти = 64 біти
   Якщо нам потрібно від 16 бітного відняти 8 бітного – то друге число потрібно перетворити у шістнадцятибітне
2. При виконанні операції множення операнди повинні мати однакову довжину, а результат довжину в два рази більшу:
   8 біт * 8 бітів = 16 бітів
   16 бітів * 16 бітів = 32 бітів
   32 біти * 32 біти = 64 біти
   64 біти * 64 біти = 128 біти
3. При виконанні операції ділення ділене повинне мати довжину в два рази більше ніж дільник та частка:
   16 бітів : 8 бітів = 8 бітів
   32 біти : 16 бітів = 16 бітів
   64 біти : 32 біти = 32 біти
   128 бітів : 64 біти = 64 біти

Основні арифметичні команди мови Ассемблер

Команди перетворення типів

cbw – розширює регістр AL до регістра AX

cwd – розширює регістр AX до регістра EAX (DX:AX)

cdq – розширює регістр EAX до регістра REAX

Команда пересилки MOV

MOV Операнд1, Операнд2

Команда обміну вмістом операндів

XCHG Операнд1, Операнд2

Команда зміни знаку

NEG Операнд

Команди інкременту та декременту

INC Операнд

DEC Операнд

Команди додавання та віднімання

ADD Операнд1, Операнд2

SUB Операнд1, Операнд2

Приклад.

Розрахувати значення виразу **x = -(a+b-c)-1**, де x, a, b,c, - 8-бітні знакові числа.

```
int main ()
{ signed char a, b, c, x1, x2;
  printf("a = "); scanf_s("%d", &a);
  printf("b = "); scanf_s("%d", &b);
  printf("c = "); scanf_s("%d", &c);
  x1 = -(a+b-c)-1;
  printf("Result C x = %d\n", x1);
  __asm {
      MOV AL, a ; <AL>:=a
      MOV AH, b; <AH>:=b
      MOV BL, c; <BL>:=c
      ADD AL,AH; <AL>:= a+b, <AH>:=b, <BL>:=c
      SUB AL, BL; <AL>:= a+b-c, <AH>:=b, <BL>:=c
      NEG AL; <AL>:= -(a+b-c), <AH>:=b, <BL>:=c
      DEC AL; <AL>:= -(a+b-c)-1, <AH>:=b, <BL>:=c
      MOV x2, AL; x := -(a+b-c)-1, <AL>:= -(a+b-c)-1, <AH>:=b, <BL>:=c
  }
  printf("Result ASM x = %d\n", x2);
  system("Pause");
  return 0;
}
```

Нестандартні ситуації.

Розрахувати вираз **x=a+b**, де x,a,b – 8-бітні беззнакові числа.

8-бітні беззнакові числа належать до діапазону 0 .. 255.

a = 10
b = 10
x = 10 + 10 = 20 – все добре
a = 150
b = 200
x = 150 + 200 = 350 – число не потрапляє в діапазон 0 .. 255
ПЕРЕПОВНЕННЯ !
Фізично (побітово) переповнення

$150_{10} = 10010110_2$
$200_{10} = 11001000_2$
Правила двійкового додавання
0+0 = 0
1+0 = 1
0+1 = 1
1+1 = 10 = ^{1}0
Додавання
10010110
11001000
101011110
$01011110_2 = 94_{10}$

Приклад 2.

- Число **a** 8-бітне беззнакове,
- $a = 10_{10} = 00001010_2$
- Після виконання команди NEG до знач. 00001010_2 отримано результат 10001010_2,
- але число ми використовуємо як 8-бітне беззнакове, тоді результатом буде число 138_{10}

Це приклад того, як логічна помилка програміста може призвести до подальших помилкових обрахунків