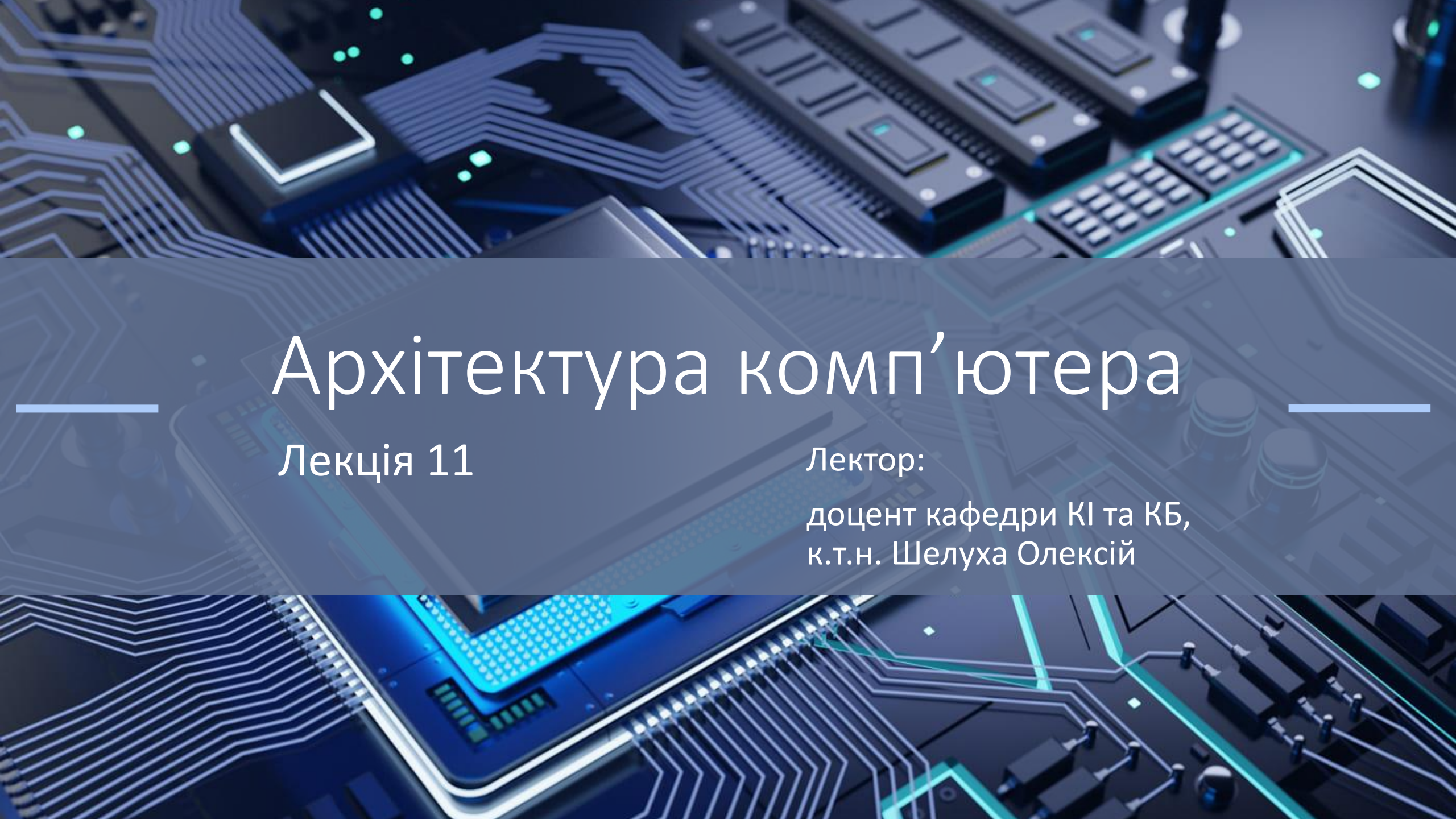


Архітектура комп’ютера

Лекція 11

Лектор:

доцент кафедри КІ та КБ,
к.т.н. Шелуха Олексій

Зміст лекції

1. Організація циклів на мові Assembler
2. Приклади циклів з використанням команд умовних переходів
3. Команди LOOPx
4. Робота зі стеком
5. Обробка масивів

Організація циклів на мові Assembler

- Типи циклів:
  - Цикл з передумовою
  - Цикл з постумовою
  - Цикл з лічильником

- Організація циклів:
  - з використанням команд умовних та безумовних переходів
  - з використанням групи команд LOOPx

Приклади циклів. Цикл з постумовою

```
        …               ; попередні оператори програми
        MOV CX, n       ; n – кількість повторень циклу
begin:                  ; початок тіла циклу
        …               ; тіло циклу
        DEC CX          ; перевірка умови продовження циклу
        CMP CX, 0       ; якщо CX = 0, то вийти з циклу
        JNE begin       ; якщо  CX <> 0, то перейти на початок
        …               ; наступні оператори програми
```

Приклади циклів. Цикл з передумовою

```
        …                ; попередні оператори програми
 MOV CX, n    ; n – кількість повторень циклу
begin:
        CMP CX, 0     ; перевірка умови продовження циклу
        JE end          ; якщо CX = 0, то вийти з циклу
        …               ; тіло циклу
        DEC CX
        JMP begin     ; перейти на початок
end:   …
       …                ; наступні оператори програми
```

Команди LOOPx

- Синтаксис команди LOOP:

  LOOP мітка

- Логіка роботи команди
- <CX> = кількість повторень (Counter)
- мітка:
- … ;тіло циклу
- *<CX> = <CX> – 1*
  якщо <CX> <> 0 перейти на мітку

- LOOP самостійно виконує декремент лічильника CX.

Команди LOOPx

- LOOPZ або LOOPE (Loop if Zero/Equal): Ця інструкція виконує наступну ітерацію циклу, якщо лічильник циклу не дорівнює нулю і прапорець Zero/Equal (ZF) встановлений. Значення лічильника зменшується на одиницю.

- LOOPNE або LOOPNZ (Loop if Not Zero/Not Equal): Ця інструкція виконує наступну ітерацію циклу, якщо лічильник циклу не дорівнює нулю і прапорець Zero/Equal (ZF) не встановлений. Значення лічильника зменшується на одиницю.

Приклад циклу з використанням команд LOOP. Приклад 1.

- Розрахувати значення факторіалу
- result1 = n! = 1 x 2 x 3 x … x n. Факторіал не може бути від’ємним.
- __asm {; якщо p буде більше 16 бітів, розрахунок цієї програми буде некоректним
- MOV CX, n ; внесення кількість повторень
- MOV BX, 1 ; BX регістр для множення
- MOV AX, BX ; <AX>=1
- JCXZ end ; Перевірка чи <CX> = 0
- begin: MUL BX ; <DX:AX> = <AX>x<BX>
- INC BX ; Регістр BX = BX+1
- LOOP begin ; перевірка чи <CX> = 0,
- якщо не дорівнює, цикл повтрюється
- end: MOV result1, AX
- }

Приклад циклу з використанням команд LOOP. Приклад 2

- Розрахувати значення факторіалу
- result2 = n! = 1 x 2 x 3 x … x n. Факторіал не може бути від’ємним.
- __asm {; якщо p буде більше 16 бітів, розрахунок цієї програми буде некоректним
- MOV CX, n ; внесення кількість повторень
- MOV AX, 1
- JCXZ end ; Перевірка чи <CX> = 0
- begin: MUL CX ; <DX:AX> = <AX>x<CX>
- LOOP begin ; перевірка чи <CX> = 0,
- якщо не дорівнює, цикл повтрюється
- end: MOV result2, AX
- }

Команди LOOPE (LOOPZ)

- Команди LOOPE (LOOPZ) перехід по лічильнику та якщо дорівнює. Аналізують ознаку регістра EFLAGS ZF = 1
- Синтаксис команди LOOPE (LOOPZ)
- LOOPE (LOOPZ) мітка
- Логіка роботи команди
- <CX> = кількість повторень
- мітка:
- … ;тіло циклу
- *<CX> = <CX> – 1*
  якщо (<CX> <> 0 та <ZF> = 1) перейти на мітку

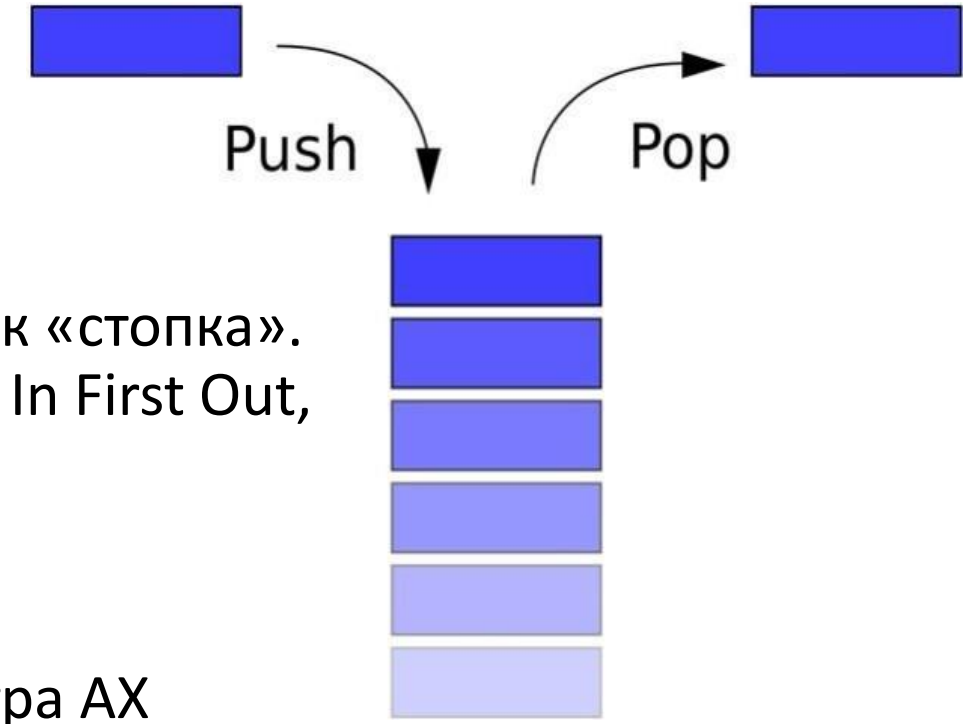
Команди LOOPNE (LOOPNZ)

- Команди LOOPNE (LOOPNZ) перехід по лічильнику та якщо НЕ дорівнює. Аналізують ознаку регістра EFLAGS ZF = 0
- Синтаксис команди LOOPNE (LOOPNZ)
- LOOPNE (LOOPNZ) мітка
- Логіка роботи команди
- <CX> = кількість повторень
- мітка:
- … ;тіло циклу
- *<CX> = <CX> – 1*
  якщо (<CX> <> 0 та <ZF> = 0) перейти на мітку

Приклад циклу з використанням команд LOOP. Приклад 3

- Знайти значення суми чисел S = 1+2+3+…+n. Обчислення припинити якщо S=K або перебрані всі n чисел.
- __asm {
- MOV CX, n
- XOR AX, AX ; очистити регістр, фактично <AX>=0
- XOR BX, BX ; очистити регістр, фактично <BX>=0
- JCXZ end
- begin: INC BX
- ADD AX, BX
- CMP AX, k
- LOOPNE begin
- end: MOV S, AX
- }

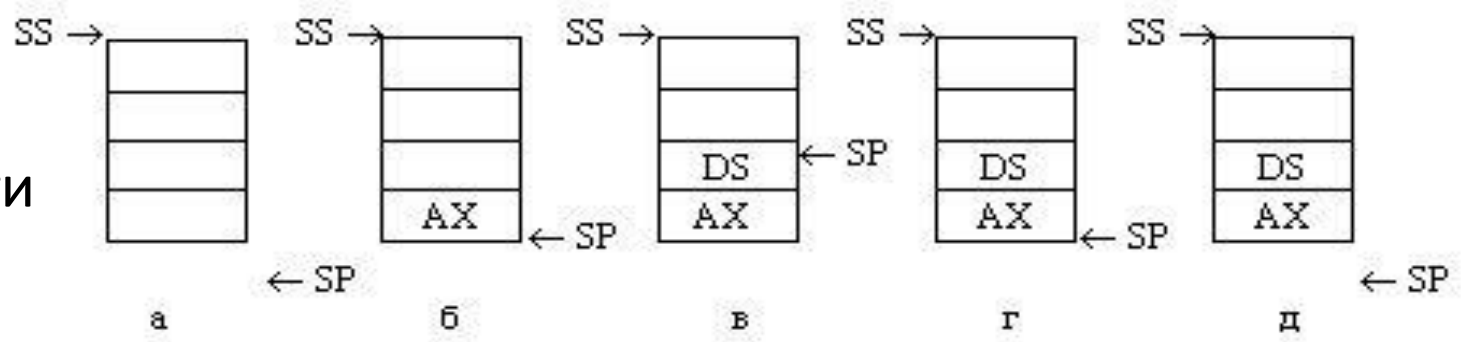
Робота зі стеком



- Stack в перекладі з англійської перекладається як «стопка». Відповідно він працює за принципом LIFO – Last In First Out, тобто останній зайшов – перший вийшов.
- Відповідно для запису «проштовхування» в стек використовується команда push %sourse%, наприклад push ax. Відповідно значення з регістра AX записується у верхню вільну комірку стека.
- Для читання зі стеку доступна тільки саме ця верхня комірка стеку. Прочес отримання називаєтсья «Виштовхування» і має синтаксис pop %destination%, наприклад pop bx. Значення з верхівки стеку переписуєтсья в регістр BX і теперь доступною стає наступна комірка.

Робота зі стеком

- Розміри комірок відповідають розмірності даних які ми проштовхуємо або виштовхуємо:
- Push al => char = 1 байт
- Push ax => short int = 2 байти
- Push eax => int = 4 байти.



- Початкова точка запису для конкретного фрейму програми (потоку) вказується в регістрі BP – base pointer. Він використовується коли викликається якась функція і відповідно для неї записи починаються з пустої комірки, яка не завжди може бути першою в основному фреймі програми. При цьому по завершенню функції дані записані в стек теж мають бути доступні.
- На конкретну комірку для запису вказує регістр SP – stack pointer. І перед кожним записом він збільшується на розмір комірки у яку ми записували дані. Автоматично для команд push та pop.

Приклад 4

- Для обчислення суми масиву чисел у асемблері.

Дякую за увагу!