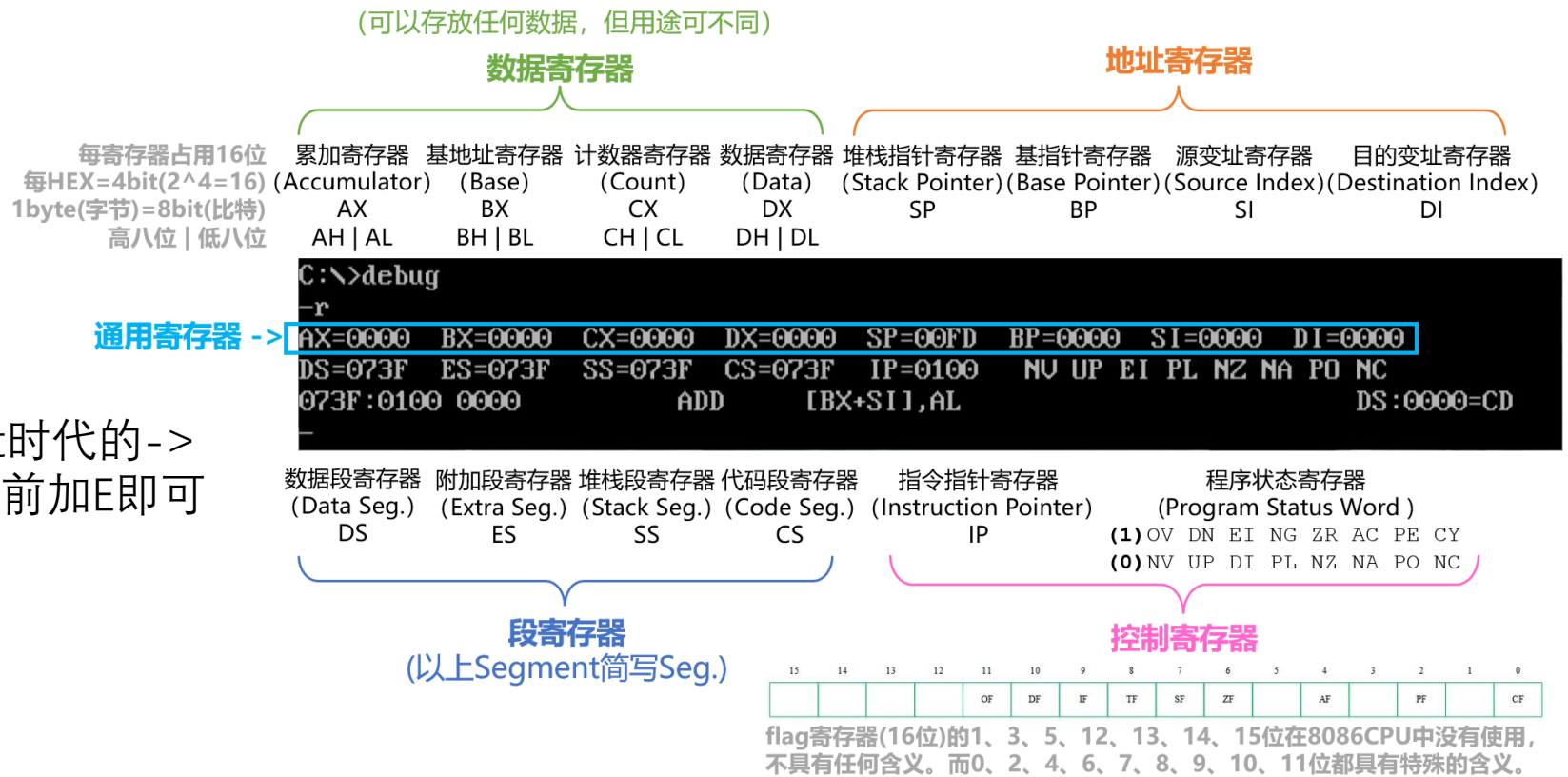
前情提要：
1.在loop指令中，ECX自动成为循环计数器，且循环一次自减1，当ECX==0时停止跳转，不为0时跳转到符号内容中。（颇具8086遗风）
2.LEA指令和mov的不同：LEA赋予的是地址
LEA对于变量，其后面的有无[ ]皆可，都表示取变量地址，相当于指针。对于寄存器而言，无[]表示取地址，有[ ]表示取值。
MOV对于变量，加不加[]都表示取值；对于寄存器而言，无[]表示取值，有[]表示取地址。（大概）
3.任何时刻，SS:SP 都是指向栈顶元素。（32位时代已经不需要偏移地址求和20位了仅需一个地址寄存器如ESP等）



## 把有7个数的数组array中每一位求和并储存在EAX寄存器中：

Drag-and-drop the correct sequence number of the assembly code to form a program where 7 numbers in an array are added and stored in eax register. Note that your sequence must absolutely match the line numbers to the left-most column of the table. The answers for Lines 2, 3 and 5 have been provided. Complete the rest.

| | | |
|---|---|---|
| Line 1 | `mov esi, array` ✖ `#lea esi, array#` | ； 将指针ESI指向array**所在地址** |
| Line 2 | `mov eax, 0` | ； 将累加器设为0 |
| Line 3 | `mov ecx, 0` | ； 将计数器设为0 |
| Line 4 | `sumLoop: add eax, [esi]` ✔ | ； sumLoop子程序开始：将ESI所指地址的值加给EAX |
| Line 5 | `add esi, 4` | ； ESI+=4,指针偏移到下一位int（4byte） |
| Line 6 | `inc ecx` ✔ | ； 计数器自增 |
| Line 7 | `cmp ecx, 7` ✔ | ； 比较ECX与7 |
| Line 8 | `jl sumLoop` ✔ | ； 如果ECX<7，则继续子程序（循环体） |

jnl sumLoop | inc ecx | dec ecx | lea esi, array | jl sumLoop | sumLoop: add eax, [esi] | cmp ecx, 7 | mov esi, array

3.LEA指令和mov的不同：LEA赋予的是地址

LEA对于变量，其后面有无[ ]皆可，都表示取变量地址，相当于指针。对于寄存器而言，无[]表示取地址，有[ ]表示取值。

MOV对于变量，加不加[]都表示取值；对于寄存器而言，无[]表示取值，有[]表示取地址。（大概）

**将字符串入栈：**

前情提要：
1.在loop指令中，ECX自动成为循环计数器，且循环一次自减1，当ECX==0时停止跳转，不为0时跳转到符号内容中。（颇具8086遗风）

Drag-and-drop the correct arguments and/or instructions to the missing places in the following program segment to push a string onto a stack.

```
char ✔ myArray[MAX_SZ] = "Hello XJTLU";        ; 指针取出来的都是猹

_asm ✔ {                                       ; 开始内联汇编
    mov ecx, MAX_SZ-1 ✔                        ; 赋计数器为数组大小-1（自己算一下）
    mov esi, 0 ✔                               ; 初始化源变址寄存器(作为index)

    cycleIt ✔ :                                ; 开始子程序cycleIt:
        movzx ✔ eax, myArray[ esi ✔ ]          ; 将源变址寄存器所指的字符串的字符补零给EAX
        push ✔ eax                             ; EAX入栈
        inc ✔ esi                              ; Index++
        loop ✔ cycleIt                         ; 循环子程序，直到ECX计数器自动归零
}
```

loop | asm | push | cycleIt | esi | movzx | MAX_SZ | [esi] | inc | char | mov | dec | _asm | 0 | MAX_SZ-1 | jmp | pop | ecx

## 将一个含有7个int的数组通过冒泡排序排为递增：

Drag-and-drop the correct sequence number of the assembly code to form a program that sorts an array of 7 integers in an **ascending** order (Bubble Sort). Note that your sequence must absolutely match the line numbers to the left-most column of the table. Complete Lines 4, 7-10.

| | | |
|---|---|---|
| Line 1 | `lea esi, array` | ; 源变址寄存器指向array |
| Line 2 | `mov ecx, 7` | ; 计数器设置为7 |
| Line 3 | `outerLoop: mov edx,ecx` | ; 外循环开始：edx=当前计数器 |
| Line 4 | `innerLoop: cmp edx, ecx` ✔ | ; 内循环开始：比较edx与计数器大小 |
| Line 5 | `jz noExchange` | ; 若相等则跳转至noEx子程序 |
| Line 6 | `mov eax, [esi + ecx * 4 - 4]` | ; eax=计算后的地址=array+ecx偏移字数-1 |
| Line 7 | `mov ebx, [esi + edx * 4]` ✖ *#mov ebx, [esi + edx * 4 - 4]#* | ; ebx=计算后的地址=array+edx偏移字数-1 |
| Line 8 | `cmp ebx, eax` ✔ | ; 比较ebx与eax的地址的值大小 |
| Line 9 | `jnl noExchange` ✖ *#jl noExchange#* | ; 若ebx>eax，则跳转至noEx子程序 |
| Line 10 | `mov [esi + ecx * 4], ebx` ✖ *#mov [esi + ecx * 4 - 4], ebx#* | ; 不然交换，即将ebx值赋给eax所在地址 |
| Line 11 | `mov [esi + edx * 4 - 4], eax` | ; 将eax值赋给ebx所在地址 |
| Line 12 | `noExchange: dec edx` | ; noEx子程序：edx-- |
| Line 13 | `jnz innerLoop` | ; 如果edx不为0，继续内循环 |
| Line 14 | `loop outerLoop` | ; 直到ecx减为0，停止外循环 |

`mov [esi + ecx * 4 - 4], ebx` `cmp ebx, eax` `jnl noExchange` `jl noExchange` `innerLoop: cmp edx, ecx` `mov ebx, [esi + edx * 4 - 4]` `mov ebx, [esi + edx * 4]` `mov [esi + ecx * 4], ebx`

## 求数组中所有数之和：

Drag-and-drop the correct arguments and/or instructions to the missing places for the following program that sums all the numbers in an array.

```
int arraySize = 5;                                  ; 数组大小为5
int intArray[ arraySize ✔ ] = {12, 3, 7, 23, 9};    ; 初始化数组，每次取出为int整数
int totalAmt = 0 ✔ ;                                ; 初始化totalAmt归零

_asm{                                               ; 开始内联汇编
      lea ✔ edi, intArray ✔                         ; edi指向数组地址
      mov ecx, arraySize ✔                          ; 计数器=数组大小
      mov eax, totalAmt                             ; eax累加器=0

      addTotalAmt ✔ :                               ; addTotalAmt子程序:
            add ✔ eax, [edi]                        ; eax+=edi所指int
            add edi, 4 ✔                            ; 然后edi指向下一个int
            loop ✔ addTotalAmt                      ; 循环子程序，直到计数器归零

      mov totalAmt ✔ , eax                          ; 将eax累加数值赋给totalAmt
}
```

movzx | intArray | loop | arraySize-1 | totalAmt | arraySize | add | addTotalAmt | 4 | [intArray] | lea | 0 | jmp | 8