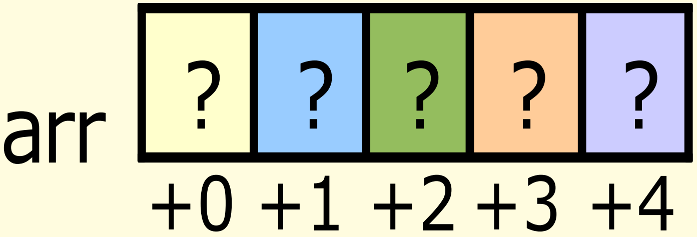
III. STRUCTURED ASSEMBLY LANGUAGE PROGRAMMING TECHNIQUES

Structured Data Types

Review - Arrays

```
i db 2
arr times 5 db 0

mov ecx, 3
for:
mov esi, ecx
mov al, cl
mul byte[i]
mov byte[arr+esi], al
loop for
```

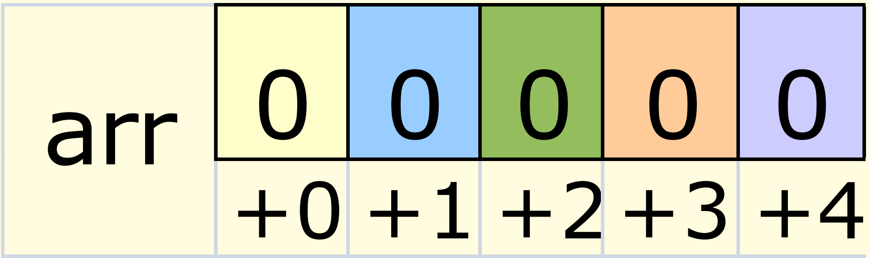


arr	?	?	?	?	?
	+0	+1	+2	+3	+4

Review - Arrays

```
i db 2
arr times 5 db 0

mov ecx, 3
for:
mov esi, ecx
mov al, cl
mul byte[i]
mov byte[arr+esi], al
loop for
```

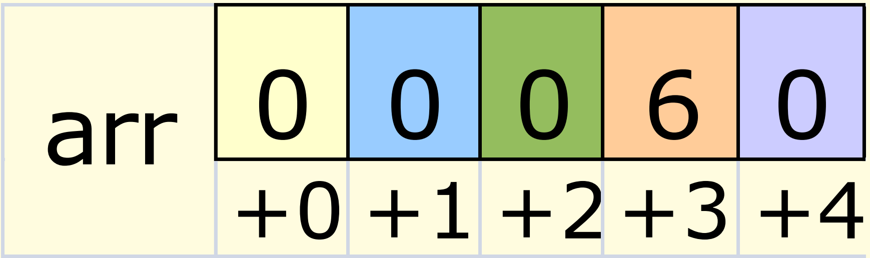


arr	0	0	0	0	0
	+0	+1	+2	+3	+4

Review - Arrays

```
i db 2
arr times 5 db 0

mov ecx, 3
for:
mov esi, ecx
mov al, cl
mul byte[i]
mov byte[arr+esi], al
loop for
```



arr	0	0	0	6	0
	+0	+1	+2	+3	+4

Review - Arrays

```
i db 2
arr times 5 db 0

mov ecx, 3
for:
mov esi, ecx
mov al, cl
mul byte[i]
mov byte[arr+esi], al
loop for
```

arr	0	0	4	6	0
	+0	+1	+2	+3	+4

Review - Arrays

```
i db 2
arr times 5 db 0

mov ecx, 3
for:
mov esi, ecx
mov al, cl
mul byte[i]
mov byte[arr+esi], al
loop for
```

arr	0	2	4	6	0
	+0	+1	+2	+3	+4

Review - Strings

```
string1 resb 20
string2 resb 20
strLen resd 1

mov eax, 3
mov ebx, 0
mov ecx, string1
mov edx, 20
int 80h
mov [strLen], eax
```

```
mov ecx, [strLen]
mov esi, string1
mov edi, string2
cld
rep movsb
```

Review - Structures

```
struct  student {
     char name[10];
     int age;
     int scores[3];
};
struct student x[5];

for(i = 5,j = 0;i > 0;i--,j++)
   x[j].scores[2] = i;
```

```
array_size   equu    5
integer      equ     2
student      equ    18
name         equ     0
age          equ    10
scores       equ    12

x  resb array_size*student
```

Review - Structures

```
for(i = 5,j = 0;i > 0;i--,j++)
    x[j].scores[2] = i;


mov ecx, 5
mov esi, 0
for:
    mov word[x+esi*student+scores+integer*2], cx
    inc esi
    loop for
```

Review - Structures

```
for(i = 5,j = 0;i > 0;i--,j++)
    x[j].scores[2] = i;


mov ecx, 5
mov esi, 0
for:
                    esi*18                          2*2
    mov word[x+esi*student+scores+integer*2], cx
    inc esi
    loop for
```

Review - Structures

```
for(i = 5,j = 0;i > 0;i--,j++)
    x[j].scores[2] = i;
```

```
mov ecx, 5
mov esi, 0
for:
    mov word[x+esi+scores+integer*2], cx
    add esi, student
    loop for
```

Review - Structures

```
mov ecx, 5
mov esi, 0
for2:
    add word[x+esi+scores+integer*2], 30h
    push ecx
    mov eax, 4
    mov ebx, 1
    lea ecx, [x+esi+scores+integer*2]
    mov edx, 1
    int 80h
    add esi, student
    pop ecx
    loop for2
```

Review - Structures

```
struct  course {
        char code[9];
        int units;
};
struct course c[5];
```

```
code        =     9
units       =     1
                 10
```

```
course    equ    10

code      equ    0
units     equ    9

size      equ    5

c resb size*course;
```

Review - Structures

- c[3].units = 3;

  mov byte[c+30+units], 3
- c[1].units = 2;

  mov byte[c+10+units], 2

Review - Structures

- scanf("%s",c[1].code);

```
mov eax, 3
mov ebx, 0
lea ecx, [c+10+code]
mov edx, 9
int 80h
```

Sets

Set Operations:

- Add element
- Remove element
- Is element

- logic instructions/bitwise operations are used to implement set operations

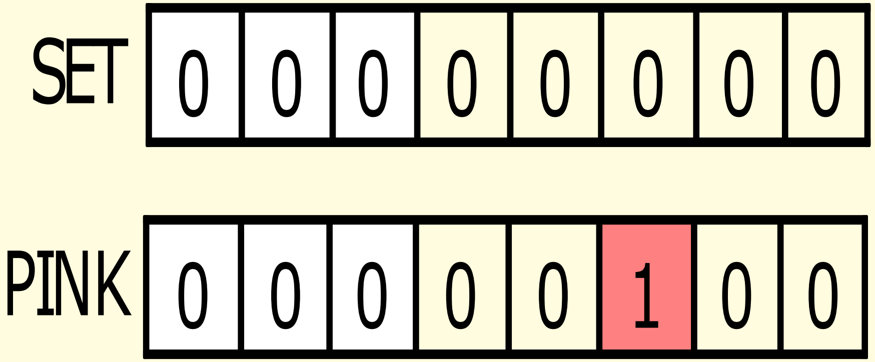
Sets – Add Element

```
BLUE        equ 1
GREEN       equ 2
PINK        equ 4
YELLOW      equ 8
ORANGE      equ 16
SET         db 0
```

SET	0	0	0	0	0	0	0	0

Sets – Add Element

```
BLUE        equ 1
GREEN       equ 2
PINK        equ 4
YELLOW      equ 8
ORANGE      equ 16
SET         db 0
```

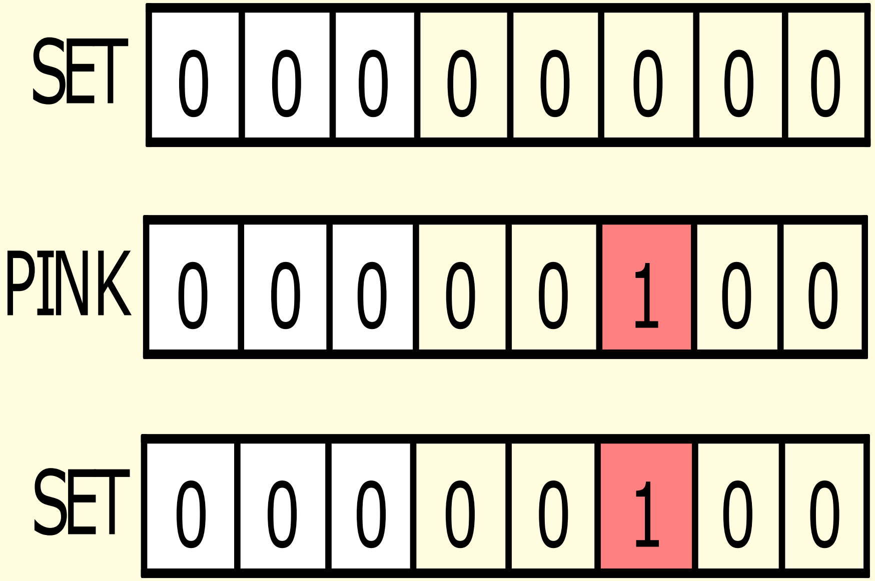


SET	0	0	0	0	0	0	0	0
PINK	0	0	0	0	0	1	0	0

```
OR byte[SET], PINK
```

Sets – Add Element

```
BLUE        equ 1
GREEN       equ 2
PINK        equ 4
YELLOW      equ 8
ORANGE      equ 16
SET         db 0
```

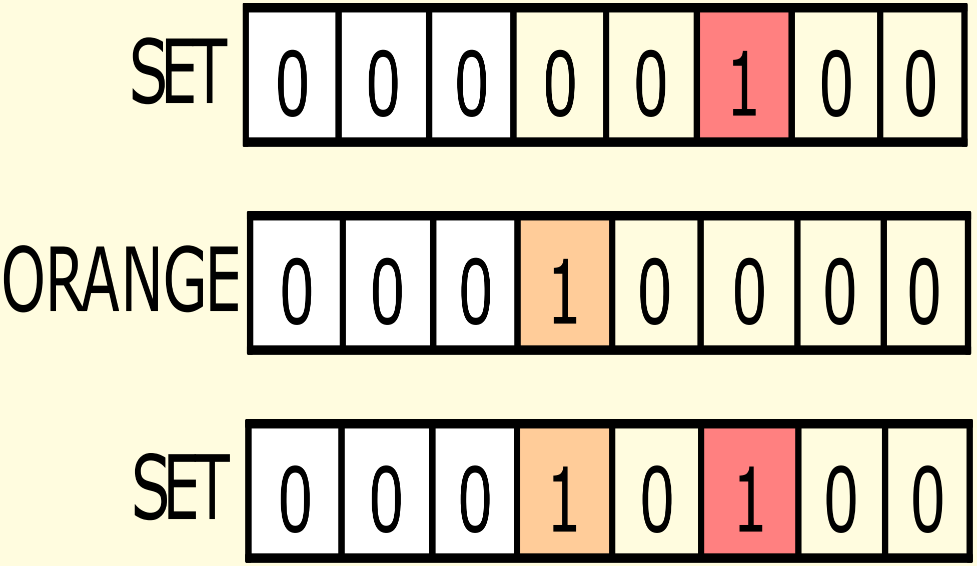


SET	0	0	0	0	0	0	0	0
PINK	0	0	0	0	0	1	0	0
SET	0	0	0	0	0	1	0	0

```
OR byte[SET], PINK
```

Sets – Add Element

```
BLUE      equ 1
GREEN     equ 2
PINK      equ 4
YELLOW    equ 8
ORANGE    equ 16
SET       db 0
```

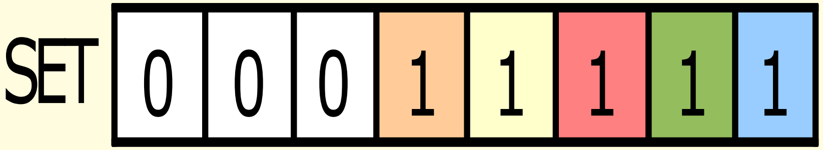


SET	0	0	0	0	0	1	0	0
ORANGE	0	0	0	1	0	0	0	0
SET	0	0	0	1	0	1	0	0

```
OR byte[SET], PINK
OR byte[SET], ORANGE
```

Sets – Remove Element

BLUE	equ 1
GREEN	equ 2
PINK	equ 4
YELLOW	equ 8
ORANGE	equ 16
SET	db 0



SET							
0	0	0	1	1	1	1	1

Sets – Remove Element

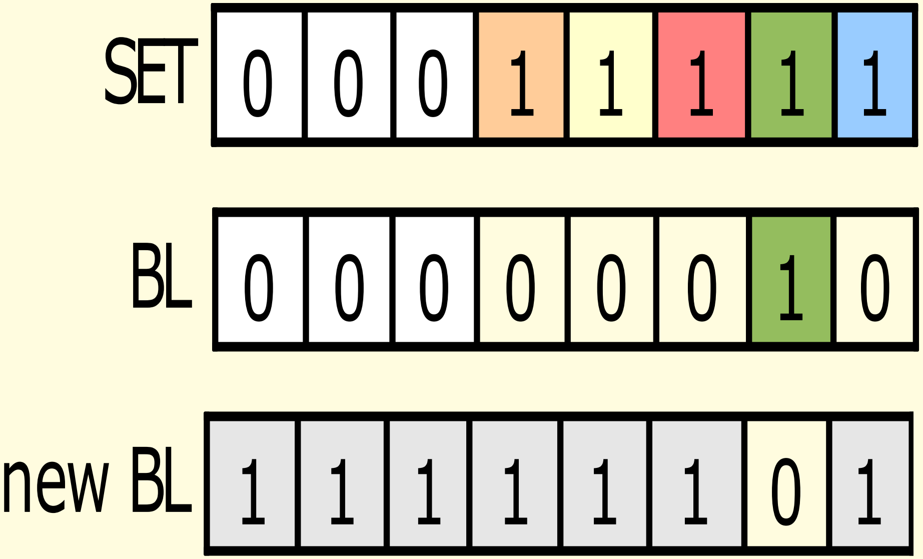
```
BLUE        equ 1
GREEN       equ 2
PINK        equ 4
YELLOW      equ 8
ORANGE      equ 16
SET         db 0
```

SET	0	0	0	1	1	1	1	1
BL	0	0	0	0	0	0	1	0

```
MOV BL, GREEN
```

Sets – Remove Element

```
BLUE        equ 1
GREEN       equ 2
PINK        equ 4
YELLOW      equ 8
ORANGE      equ 16
SET         db 0
```



SET	0	0	0	1	1	1	1	1
BL	0	0	0	0	0	0	1	0
new BL	1	1	1	1	1	1	0	1

```
MOV BL, GREEN
NOT BL
AND byte[SET], BL
```

Sets – Remove Element

```
BLUE        equ 1
GREEN       equ 2
PINK        equ 4
YELLOW      equ 8
ORANGE      equ 16
SET             db 0

MOV BL, GREEN
NOT BL
AND byte[SET], BL
```

SET	0	0	0	1	1	1	1	1
BL	0	0	0	0	0	0	1	0
new BL	1	1	1	1	1	1	0	1
SET	0	0	0	1	1	1	0	1

Sets – Is Element

```
BLUE        equ 1
GREEN       equ 2
PINK        equ 4
YELLOW      equ 8
ORANGE      equ 16
SET         db 0
```

SET	0	0	0	1	1	1	0	1
YELLOW	0	0	0	0	1	0	0	0

Sets – Is Element

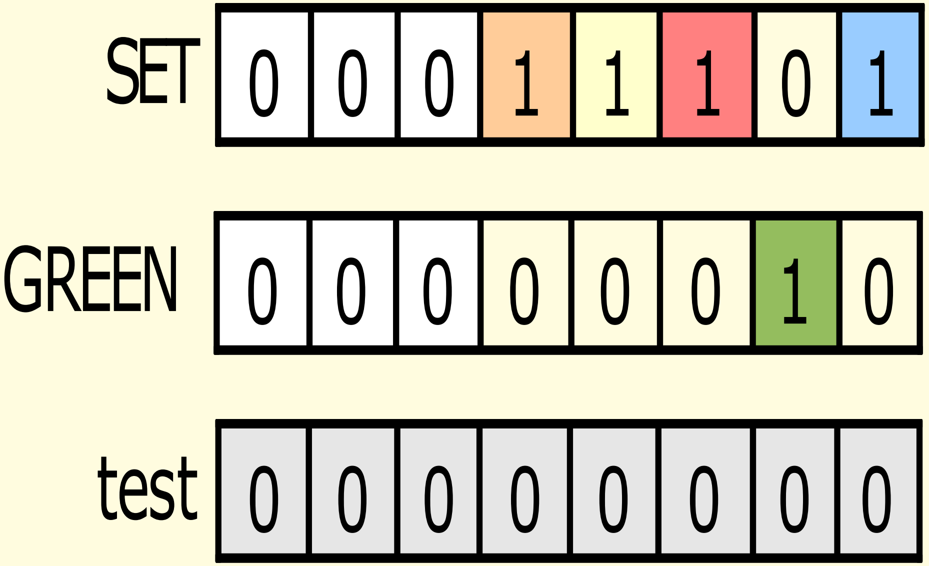
```
BLUE        equ 1
GREEN       equ 2
PINK        equ 4
YELLOW      equ 8
ORANGE      equ 16
SET         db 0
```

SET	0	0	0	1	1	1	0	1
YELLOW	0	0	0	0	1	0	0	0
test	0	0	0	0	1	0	0	0

```
TEST byte[SET], YELLOW
JZ noYellow
```

Sets – Is Element

```
BLUE      equ 1
GREEN     equ 2
PINK      equ 4
YELLOW    equ 8
ORANGE    equ 16
SET       db 0
```



SET	0	0	0	1	1	1	0	1
GREEN	0	0	0	0	0	0	1	0
test	0	0	0	0	0	0	0	0

```
TEST byte[SET], GREEN
JZ noGreen
```

Sets – Add Element

ASM	equ 1
C#	equ 2
COBOL	equ 4
JAVA	equ 8
LISP	equ 16
PLs	db 0

PLs	0	0	0	0	0	0	0	0

Sets – Add Element

ASM	equ 1
C#	equ 2
COBOL	equ 4
JAVA	equ 8
LISP	equ 16
PLs	db 0

PLs	0	0	0	0	0	0	0	0

ASM	0	0	0	0	0	0	0	1

PLs	0	0	0	0	0	0	0	1

Sets – Add Element

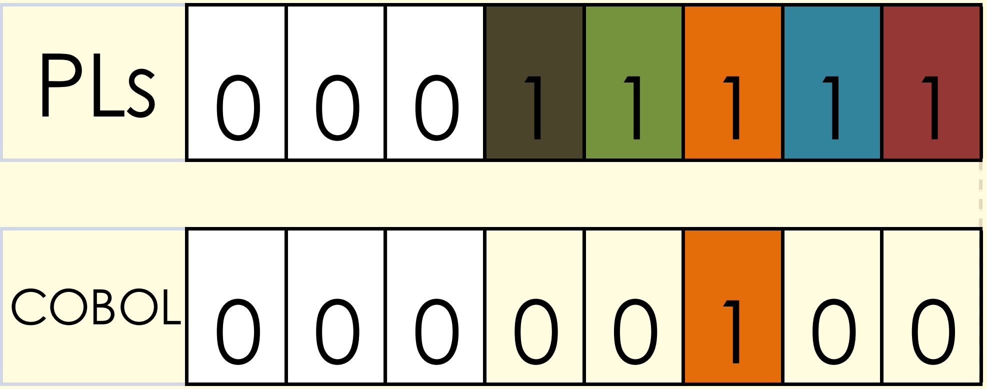
ASM	equ 1
C#	equ 2
COBOL	equ 4
JAVA	equ 8
LISP	equ 16
PLs	db 0

```
OR byte[PLs], ASM
```

PLs	0	0	0	0	0	0	0	0
ASM	0	0	0	0	0	0	0	1
PLs	0	0	0	0	0	0	0	1

Sets – Remove Element

ASM	equ 1
C#	equ 2
COBOL	equ 4
JAVA	equ 8
LISP	equ 16
PLs	db 0



PLs	0	0	0	1	1	1	1	1
COBOL	0	0	0	0	0	1	0	0

Sets – Remove Element

```
ASM      equ 1
C#       equ 2
COBOL    equ 4
JAVA     equ 8
LISP     equ 16
PLs      db 0
```

PLs	0	0	0	1	1	1	1	1
BL	0	0	0	0	0	1	0	0

```
MOV BL, COBOL
```

Sets – Remove Element

```
ASM         equ 1
C#          equ 2
COBOL       equ 4
JAVA        equ 8
LISP        equ 16
PLs         db 0
```

PLs	0	0	0	1	1	1	1	1
BL	1	1	1	1	1	0	1	1

```
MOV BL, COBOL
NOT BL
```

Sets – Remove Element

```
ASM        equ 1
C#         equ 2
COBOL      equ 4
JAVA       equ 8
LISP       equ 16
PLs        db 0
```

```
MOV BL, COBOL
NOT BL
AND byte[PLs], BL
```

PLs	0	0	0	1	1	1	1	1
BL	1	1	1	1	1	0	1	1
PLs	0	0	0	1	1	0	1	1

Sets – Is Element

```
ASM     equ 1
C#      equ 2
COBOL   equ 4
JAVA    equ 8
LISP    equ 16
PLs     db 0
```

PLs	0	0	0	1	1	0	1	1

JAVA	0	0	0	0	1	0	0	0

Sets – Is Element

```
ASM        equ 1
C#         equ 2
COBOL      equ 4
JAVA       equ 8
LISP       equ 16
PLs        db 0

TEST byte[PLs], JAVA
JZ noJava
```

PLs	0	0	0	1	1	0	1	1
JAVA	0	0	0	0	1	0	0	0
test	0	0	0	0	1	0	0	0

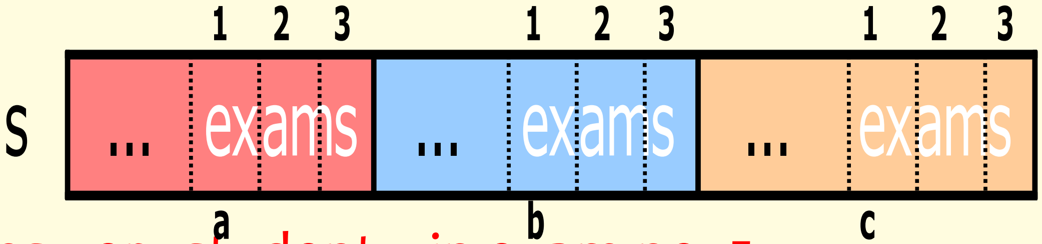
Quiz

In the following assembly program, if there are 3 byte-size exam scores for each student (a,b,c), which student has a perfect score on which exam?

```
size      equ   3
student   equ  15
stdno     equ   0
age       equ  11
exams     equ  12
```

```
s resb size*student

mov ESI, 0
add ESI, student
mov word[s+ESI+exams+1], 100
```



Sample Answer: student x in exam no. 5