



Assembler Intel 8086

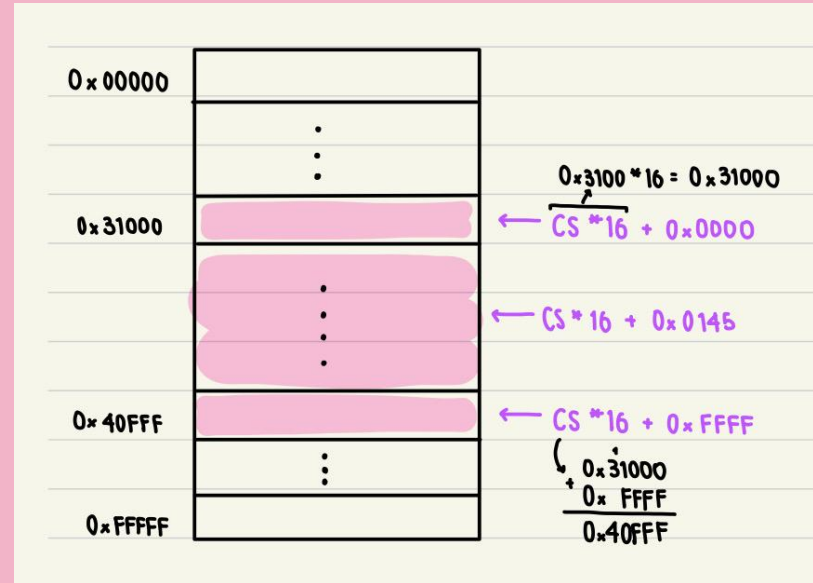
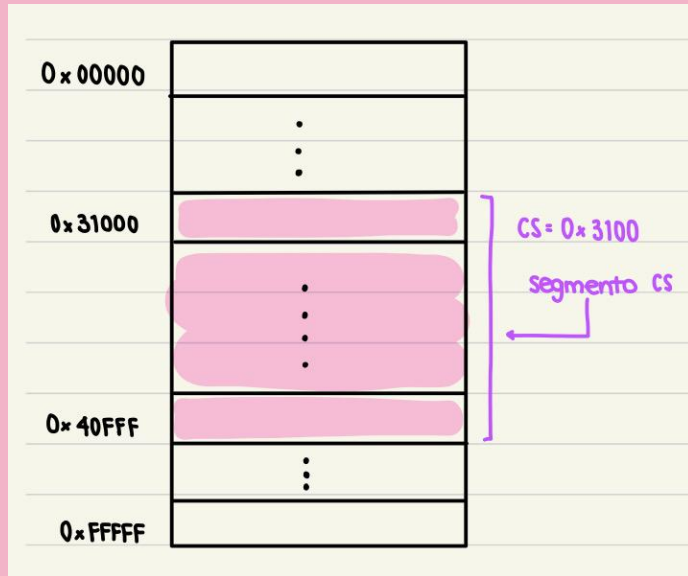
Arquitectura de Computadoras - Práctico 7

Características principales

- Diseño CISC
- Procesador de 16 bits: que implica?
- Memoria segmentada
- 12 registros visibles, con personalidad
- Little Endian
- Stack implementado por hardware sobre memoria
- Memoria direccionada de a byte

Memoria segmentada

$\text{direccion_fisica}[20] = \text{registro_segmento} * 16 + \text{desplazamiento}$



ejercicio: Presente tres formas de lograr la dirección 0xCAFE en el modo segmentado

Variables

variable: es un espacio en memoria con un nombre asignado

tipo: se refleja con el tamaño que ocupan en memoria

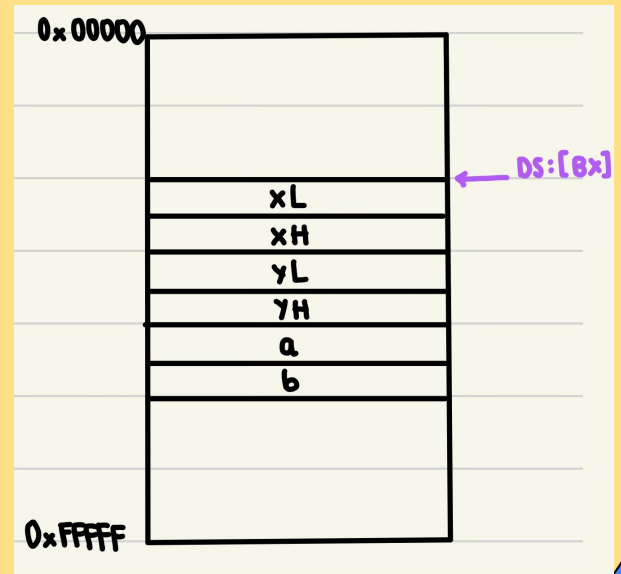
```
struct ejemplo {  
    short x,y;  
    char a,b;  
}; // 6 bytes
```

```
mov AX, [Bx] // Ax = x
```

```
mov AX, [Bx+2] // Ax = y
```

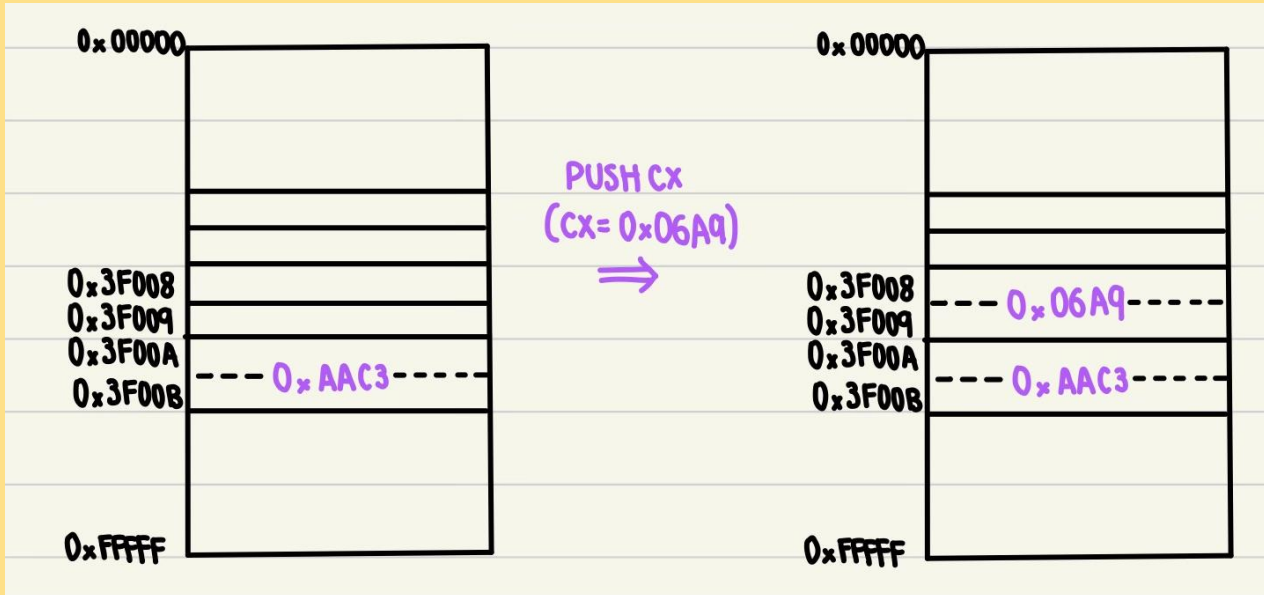
```
mov AL, [Bx+4] // AL = a
```

```
mov AL, [Bx+5] // AL = b
```



Stack

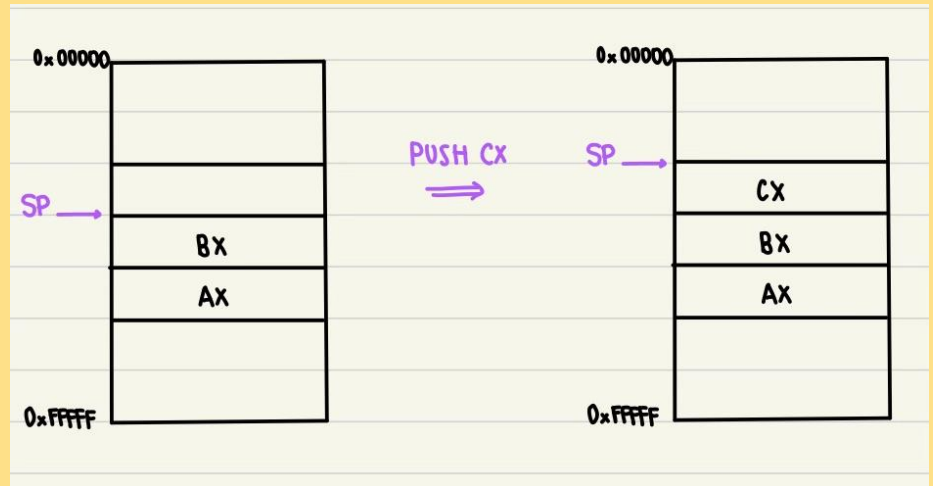
- posiciones ocupan 2 bytes SIEMPRE
- crece hacia las posiciones más bajas de memoria



Stack (2)

El stack se implementa mediante dos punteros relativos al segmento SS:

- stack pointer (SP): última posición del stack
- base pointer (BP): para direccionar en el stack



Punteros

¿Cómo se representan los punteros?

- en el mismo segmento: son 2 bytes para el desplazamiento
- en otro segmento: son 4 bytes (2 para el segmento y 2 para el desplazamiento)
- como direcciones absolutas de memoria

Operaciones con punteros en C

- `&`: devuelve la dirección de memoria donde se almacena la variable, un puntero a esa variable.
 - `*`: accede al valor almacenado en la dirección de memoria a la que apunta el puntero, para acceder al valor real que se encuentra en esa ubicación de memoria.
 - `new`: asigna un bloque de memoria dinámico para un tipo de dato y devuelve un puntero
 - `delete`: es necesario liberar esa memoria manualmente con el operador `delete` para evitar goteras de memoria.
- no existen
son
funciones
de alto nivel

Ejemplo

```
void main () {  
    int a = 4;  
    int b = getAndSet(&a, 3);  
    printf("a = %d, b = %d\n", a, b);  
}
```

```
int getAndSet(int *xp, int y) {  
    int x = *xp;  
    *xp = y;  
    return x;  
}
```

```
void main () {  
    int a = 4;  
    int b = getAndSet(&a, 3);  
    printf("a = %d, b = %d\n", a, b);  
}
```

```
int getAndSet(int *xp, int y) {  
    int x = *xp;  
    *xp = y;  
    return x;}  
}
```

```
.data    segmento de datos  
a dw 3;  definimos variables  
b dw ?;  en memoria  
.code  
; pasaje parámetros  
MOV BX, OFFSET a;  &a  
MOV AX, 3;  
CALL getAndSet;  
; en CX está el resultado de la función
```

```
proc getAndSet  
    MOV CX, [BX]; // x = *xp  
    MOV [BX], AX; // *xp = y  
ret  
endproc
```

Ejercicio

Considere la siguiente estructura de datos para manejar inscripciones a cursos en un sistema x86:

```
struct _asignatura{
    short codA;
    char* nombre;
}

struct curso{
    short año;
    _asignatura* asig;
    short cantEstudiantes;
}
```

Y las siguientes variables globales:

```
short cantCursos;
curso[1024] cursos;
```

Donde la variable 'cantCursos' indica la cantidad de posiciones válidas del arreglo.

- Implementar en un lenguaje de alto nivel, la función:

short cantidadTotalEstudiantes(short año);

Que dado un año, indica la cantidad total de estudiantes en cursos de dicho año.

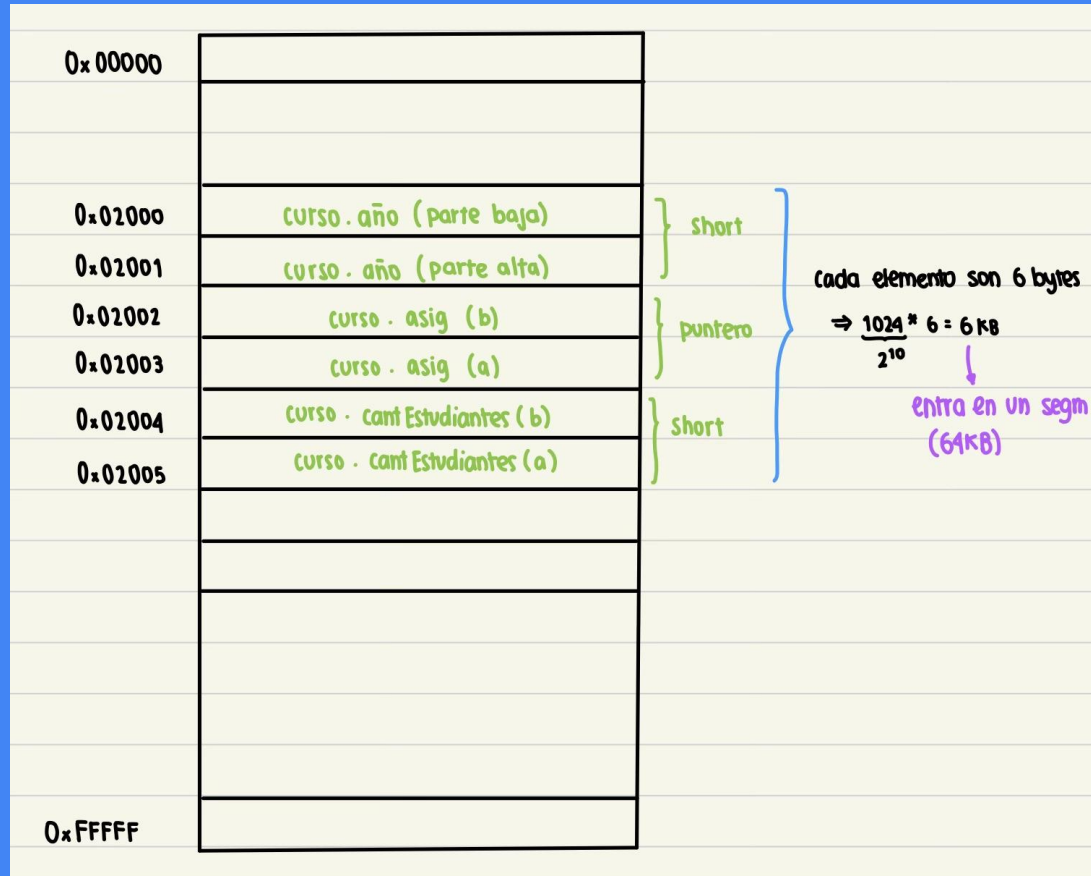
Ejercicio - Parte A

```
Short cantidadTotalEstudiantes (short año){  
    short cant = 0;  
    for (int i=0 ; i < cantCursos ; i++) {  
        if (cursos [i]. año == año)  
            cant += cursos [i]. cantEstudiantes;  
    }  
    return cant;  
};
```


Ejercicio - Parte B

0x00000			
0x02000	curso . año (parte baja)	} short	} cada elemento son ? bytes
0x02001	curso . año (parte alta)		
0x02002	curso . asig (b)	} puntero	
0x02003	curso . asig (a)		
0x02004	curso . cant Estudiantes (b)	} short	
0x02005	curso . cant Estudiantes (a)		
0xFFFFF			

Ejercicio - Parte B



Ejercicio - Parte C

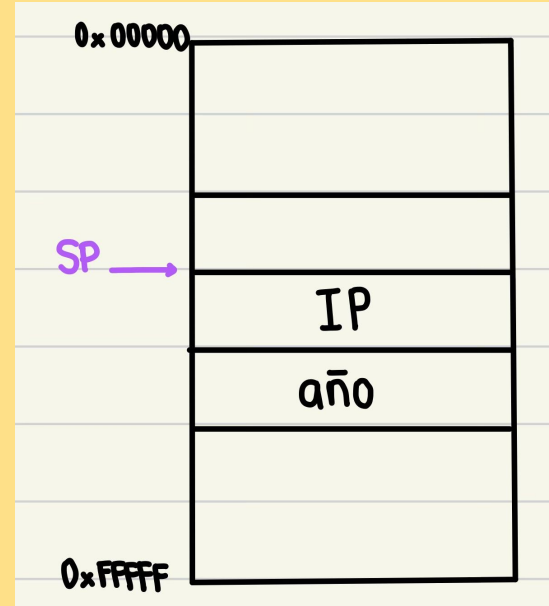
- c. Compilar en assembler 8086 sabiendo que las variables globales se encuentran almacenadas a partir de la dirección 0 del ES. La variable año se recibe en el stack y el resultado se devuelve en el stack. Se debe preservar el valor de todos los registros.

cómo se reciben los parámetros?

Ejercicio - Parte C

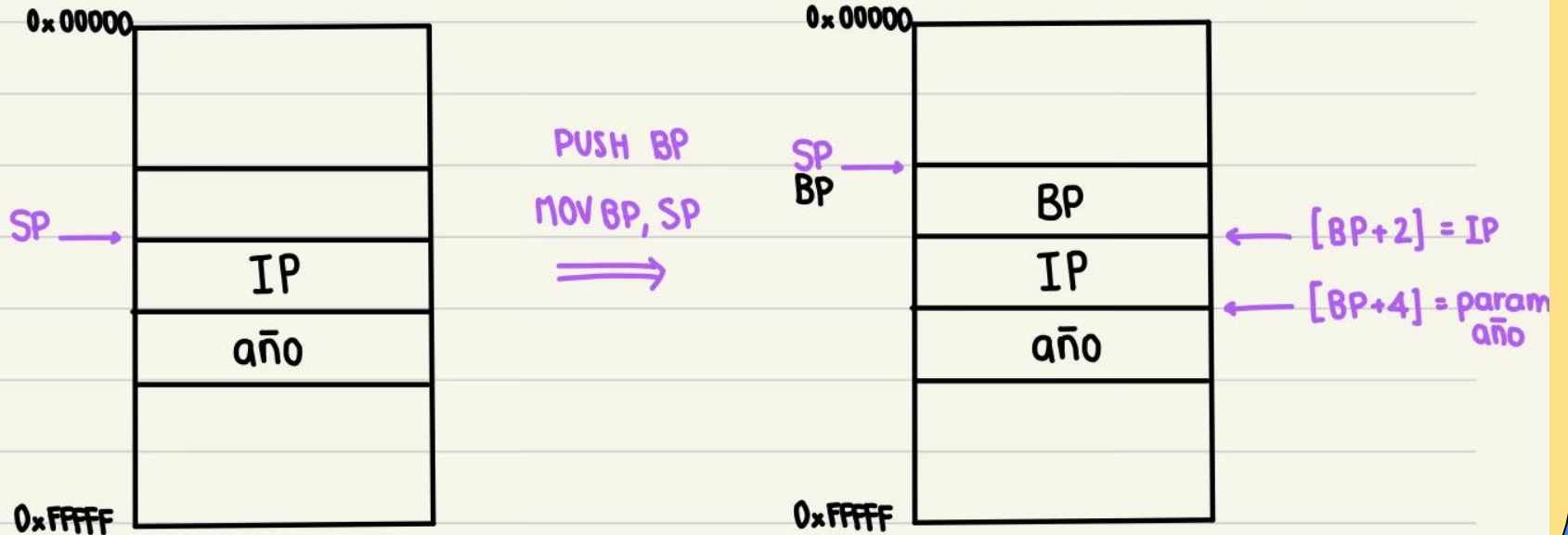
- c. Compilar en assembler 8086 sabiendo que las variables globales se encuentran almacenadas a partir de la dirección 0 del ES. La variable año se recibe en el stack y el resultado se devuelve en el stack. Se debe preservar el valor de todos los registros.

cómo se reciben los parámetros:



Ejercicio - Parte C

referencia al stack:



Ejercicio - Parte C

cantidadTotalEstudiantes PROC

PUSH BP;

MOV BP, SP;

Ejercicio - Parte C

preservar los registros:

CantidadTotalEstudiantes Proc

```
PUSH BP;
```

```
MOV BP, SP;
```

```
PUSH AX;
```

```
PUSH BX;
```

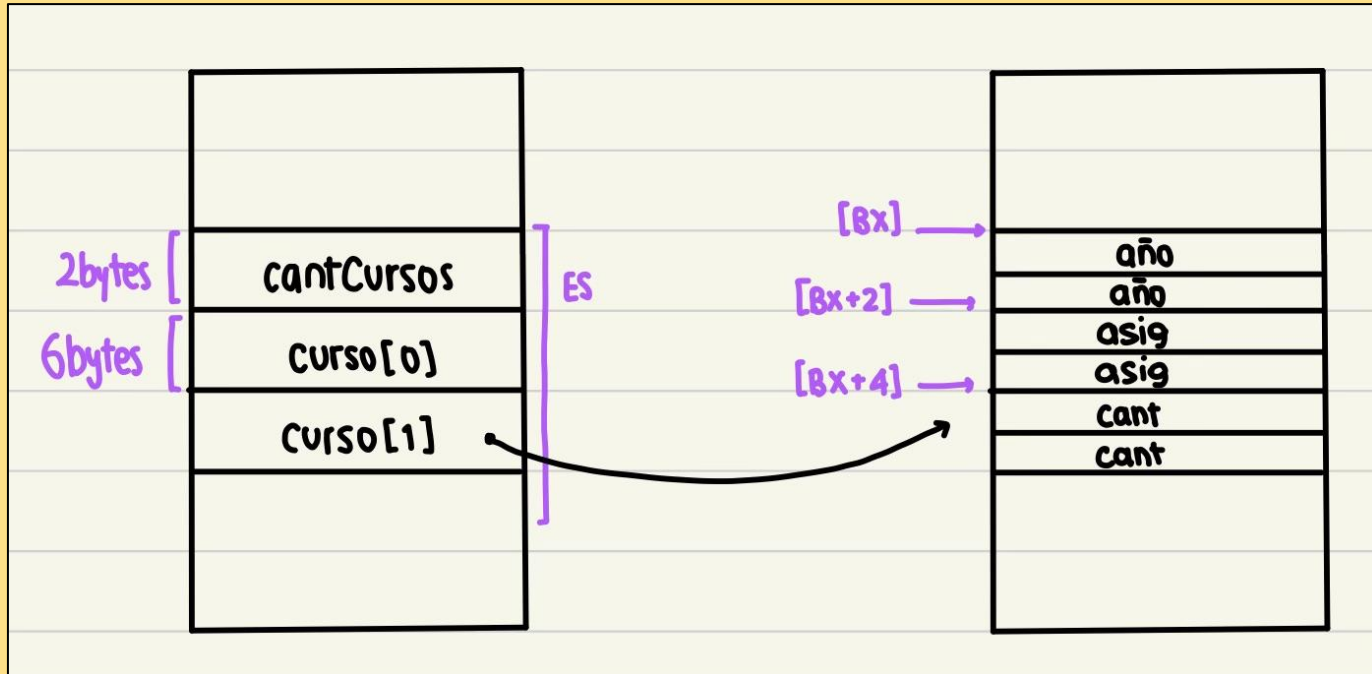
```
PUSH CX;
```

```
PUSH DX;
```

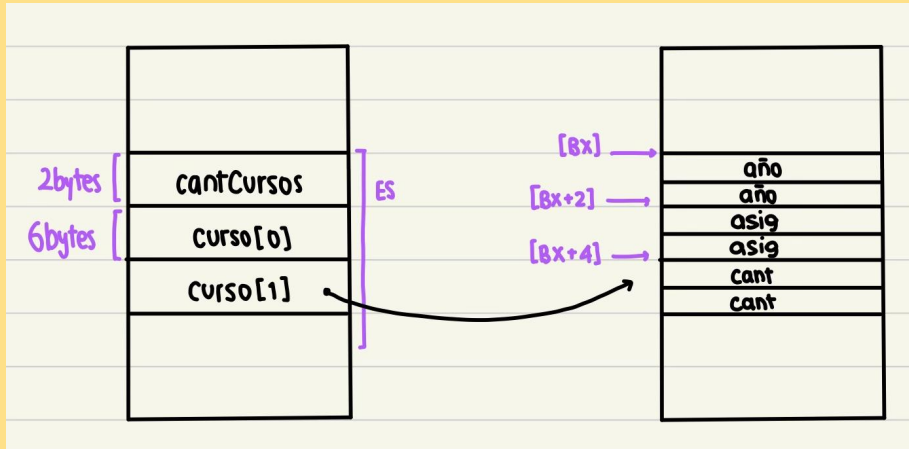


Ejercicio - Parte C

variables globales en memoria:



Ejercicio - Parte C



```
XOR AX, AX ; AX = cant
```

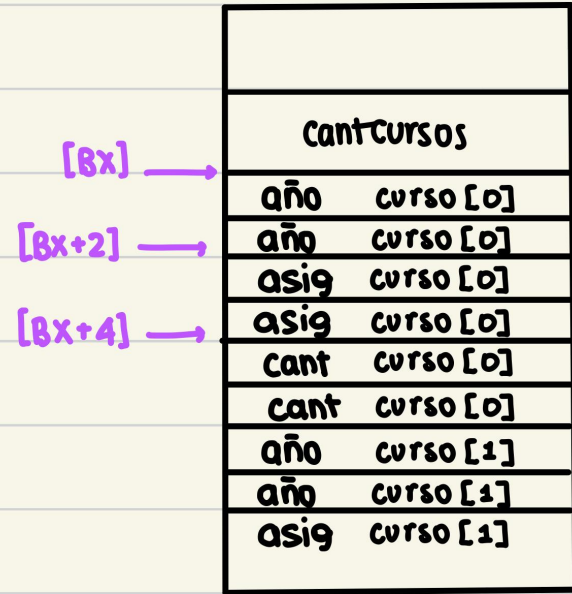
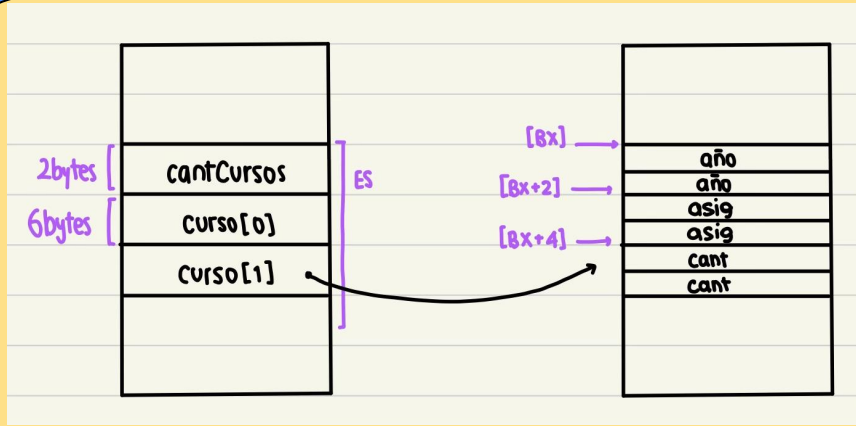
```
XOR DX, DX ; DX = i
```

```
MOV BX, 2
```

```
MOV CX, [BP+4] ; CX = año
```

FOR:

Ejercicio - Parte C



FOR:

```
CMP DX, ES:[0] ; cantCursos (short)
```

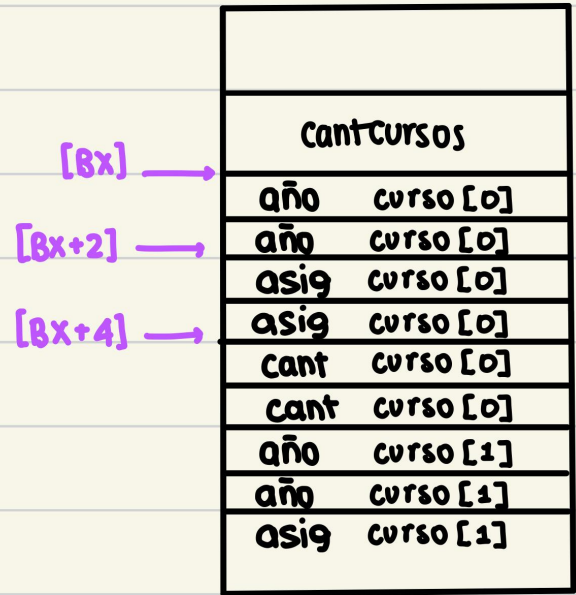
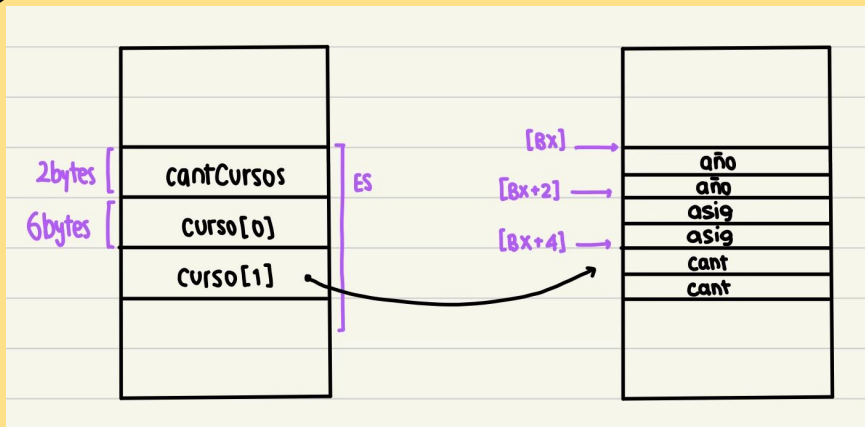
```
JNB FINFOR ; salta si  $DX \geq ES:[0]$  ( $i \geq \text{cantCursos}$ )
```

```
CMP CX, ES:[8x] ; cursos[i].año == año
```

```
JNE FINIF
```

```
ADD AX, ES:[8x+4] ; cant += cursos[i]. CantEstudiantes
```

Ejercicio - Parte C



FINIF :

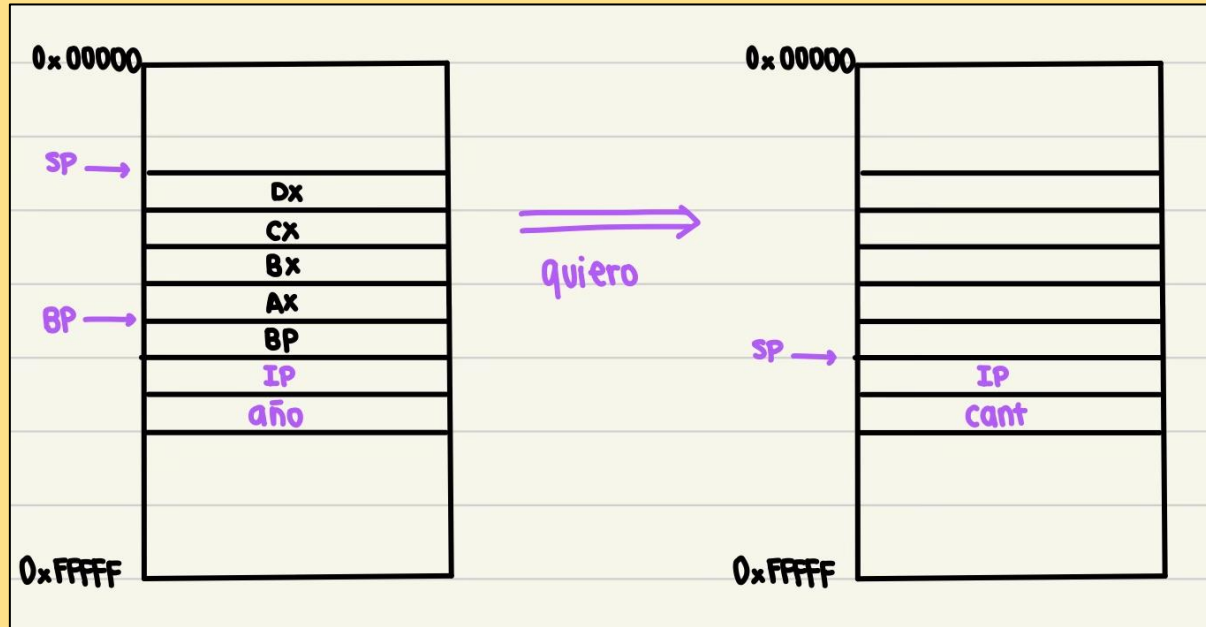
ADD Bx, 6 ; apunta al siguiente curso

INC DX ; i++

JMP FOR

Ejercicio - Parte C

retorno de la función:



Ejercicio - Parte C

FINFOR:

MOV [BP+4], AX ; guardo el res donde vino el parámetro

POP DX

POP CX

POP BX

POP AX

POP BP ; deajo IP en el tope

RET