

Memoria Estática

Punteros, Vectores y Matrices

Organización del Computador II

2º Cuatrimestre 2015

18 de agosto

Repaso de punteros

¿Qué, cómo y para qué?

- Un puntero es una variable que referencia una posición de la memoria. (ejemplo: una variable cuyo valor es una dirección de memoria)
- Como toda variable, tiene un tipo y un nombre.
- Sirve para guardar datos en memoria.
- Operadores:
 - & → Da como resultado la dirección de memoria de una variable.
 - * → Da como resultado el valor apuntado por un puntero.
(Además de ser el indicador del tipo puntero)

Repaso de punteros

Ejemplos:

- `int *pepe`

Crea un puntero de tipo entero con nombre *pepe*.

Repaso de punteros

Ejemplos:

- `int *pepe`

Crea un puntero de tipo entero con nombre *pepe*.

- `int x = 5`
`pepe = &x`

Guarda en la dirección del puntero *pepe* la dirección de *x*.
Se dice que *pepe* apunta a *x*.

Repaso de punteros

Ejemplos:

- `int *pepe`

Crea un puntero de tipo entero con nombre *pepe*.

- `int x = 5`
`pepe = &x`

Guarda en la dirección del puntero *pepe* la dirección de *x*.
Se dice que *pepe* apunta a *x*.

- `*pepe = 8`

Guarda 8 en la dirección apuntada por el puntero *pepe*.

Repaso de punteros

Ejemplos:

- `int *pepe`

Crea un puntero de tipo entero con nombre *pepe*.

- `int x = 5`

`pepe = &x`

Guarda en la dirección del puntero *pepe* la dirección de *x*.

Se dice que *pepe* apunta a *x*.

- `*pepe = 8`

Guarda 8 en la dirección apuntada por el puntero *pepe*.

- `int y`

`y = *pepe`

Guarda en *y* el valor apuntado por *pepe*.

Repaso de punteros

Ejemplos más visuales:

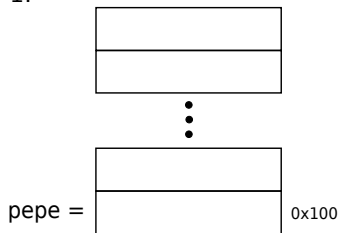
❶ `int *pepe`

❷ `int x = 5`
`pepe = &x`

❸ `*pepe = 8`

❹ `int y`
`y = *pepe`

1.



Repaso de punteros

Ejemplos más visuales:

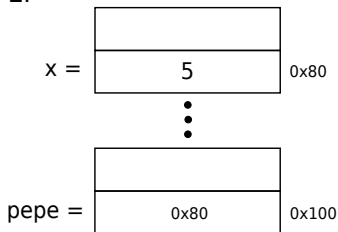
❶ `int *pepe`

❷ `int x = 5`
`pepe = &x`

❸ `*pepe = 8`

❹ `int y`
`y = *pepe`

2.



Repaso de punteros

Ejemplos más visuales:

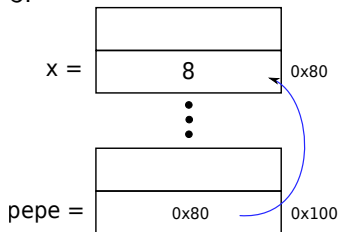
❶ `int *pepe`

❷ `int x = 5`
`pepe = &x`

❸ `*pepe = 8`

❹ `int y`
`y = *pepe`

3.



Repaso de punteros

Ejemplos más visuales:

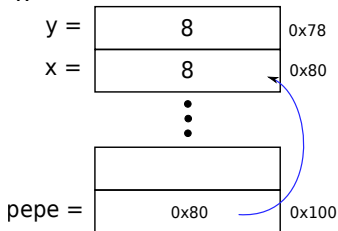
❶ `int *pepe`

❷ `int x = 5`
`pepe = &x`

❸ `*pepe = 8`

❹ `int y`
`y = *pepe`

4.

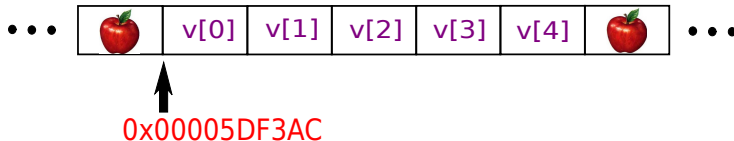


Vectores

- Declaramos en C un vector v :

```
int v[5];
```

- ¿Cómo está guardado en memoria?
- Como 5 enteros (*doublewords* / 4 bytes) **consecutivos**:

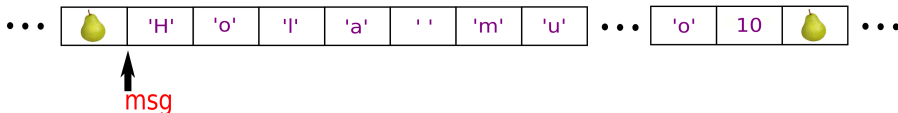


Vectores

- Si rememoramos el ejemplo de la primera clase:

```
section .data:  
    msg: DB 'Hola mundo', 10  
    largo: EQU $-msg
```

msg es una etiqueta que, vista como un puntero, es un vector de caracteres almacenados de la siguiente manera:



msg es un `char*`, es como si en C hiciéramos:

```
char msg[11] = "Hola mundo\n";
```

Vectores

Volviendo al ejemplo del vector v

- Si suponemos que el primer elemento se encuentra almacenado en la dirección $0x200$ de memoria; ¿cómo se realiza la siguiente asignación?

```
int v[5];  
v[2] = 8;
```



Vectores

Volviendo al ejemplo del vector v

- Si suponemos que el primer elemento se encuentra almacenado en la dirección $0x200$ de memoria; ¿cómo se realiza la siguiente asignación?

```
int v[5];  
v[2] = 8;
```

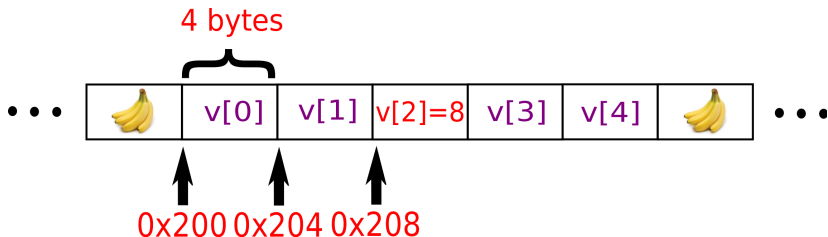


Vectores

Volviendo al ejemplo del vector v

- Si suponemos que el primer elemento se encuentra almacenado en la dirección $0x200$ de memoria; ¿cómo se realiza la siguiente asignación?

```
int v[5];  
v[2] = 8;
```

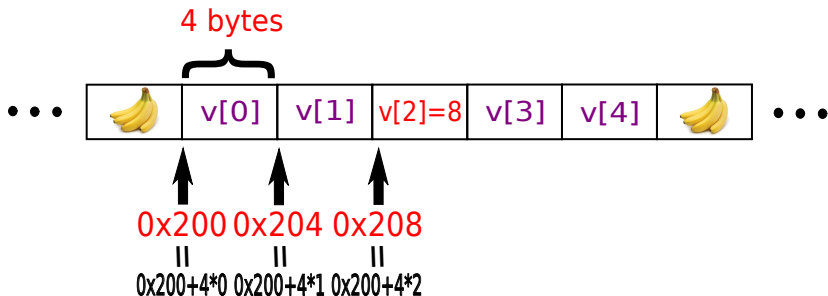


Vectores

Volviendo al ejemplo del vector v

- Si suponemos que el primer elemento se encuentra almacenado en la dirección $0x200$ de memoria; ¿cómo se realiza la siguiente asignación?

```
int v[5];  
v[2] = 8;
```

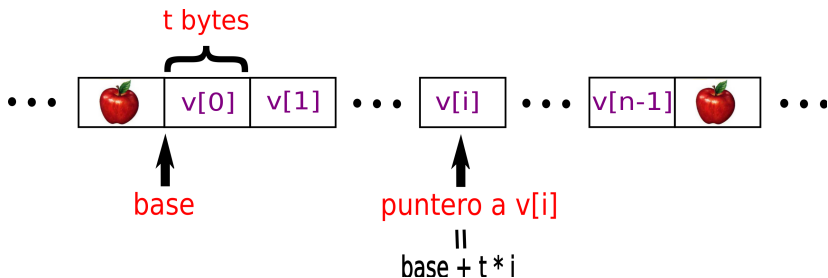


Vectores

La regla

- En general, para indizar dentro de un vector, la “regla” es:

“puntero al inicio del arreglo” +
+ “tamaño del dato” * “índice del elemento al que queremos acceder”



El modo de direccionamiento que se utiliza para indizar un vector es:

base + índice + desplazamiento

El formato general para direccionar en Intel es:

`[Base + Índice*scala +/- Desplazamiento]`

Base = algún registro

Índice = algún registro

scala = 1, 2, 4 u 8

Desplazamiento = inmediato de 32 bits

ver: Manual Intel - Vol.1 - 3.7.5 - Specifying an Offset

Vectores y Punteros

Si tenemos:

```
int v[5];
```

Para C: v es un puntero al primer elemento del vector.

Entonces podemos hacer en C:

```
int *p_v = v;
```

```
int *p_v = &v[0];
```

Matrices

- No son muy distintas a los vectores.
- Se representan igual en memoria.
- Se almacenan como varios vectores. En C se almacenan por filas.
- Si la matriz tiene dimensión $M \times N$ entonces sabemos que está formada por M vectores de N elementos cada uno. Se lo conoce como almacenamiento por filas.
- Estos vectores están almacenados en memoria uno al lado del otro.

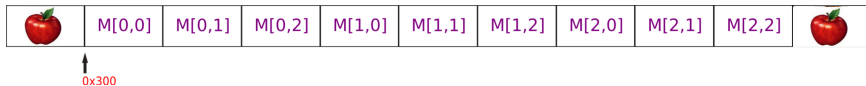
Matrices

Ejemplo: almacenamiento por filas

- Si tenemos M , una matriz de enteros de 3×3 :

$M[0,0]$	$M[0,1]$	$M[0,2]$
$M[1,0]$	$M[1,1]$	$M[1,2]$
$M[2,0]$	$M[2,1]$	$M[2,2]$

- En memoria se representa:

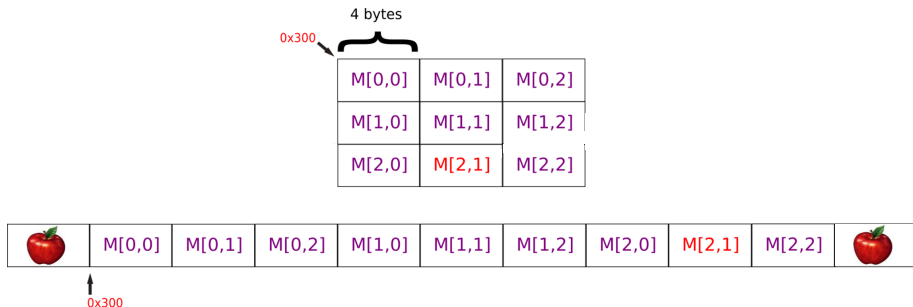


Matrices

- Suponiendo que el primer elemento de M se encuentra almacenado en la posición de memoria $0x300$ y queremos asignar un valor en $M[2, 1]$ entonces en C hacemos:

```
int m[3][3];  
m[2][1] = 7;
```

- ¿Y en ensamblador?

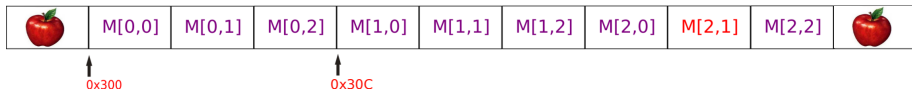
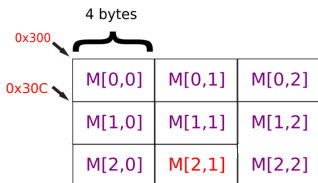


Matrices

- Suponiendo que el primer elemento de M se encuentra almacenado en la posición de memoria $0x300$ y queremos asignar un valor en $M[2,1]$ entonces en C hacemos:

```
int m[3][3];  
m[2][1] = 7;
```

- ¿Y en ensamblador?

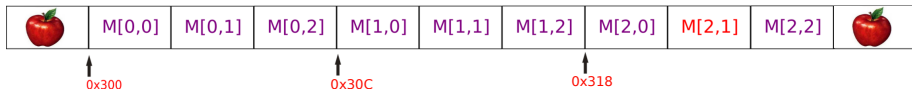
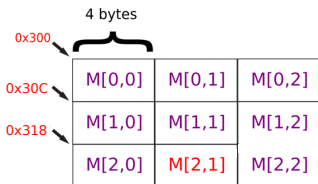


Matrices

- Suponiendo que el primer elemento de M se encuentra almacenado en la posición de memoria $0x300$ y queremos asignar un valor en $M[2,1]$ entonces en C hacemos:

```
int m[3][3];  
m[2][1] = 7;
```

- ¿Y en ensamblador?

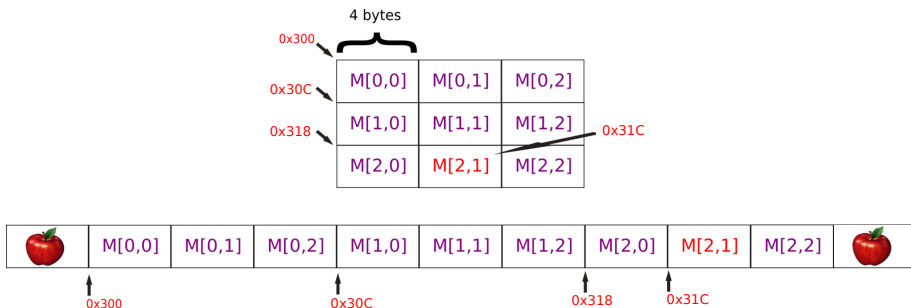


Matrices

- Suponiendo que el primer elemento de M se encuentra almacenado en la posición de memoria $0x300$ y queremos asignar un valor en $M[2,1]$ entonces en C hacemos:

```
int m[3][3];  
m[2][1] = 7;
```

- ¿Y en ensamblador?

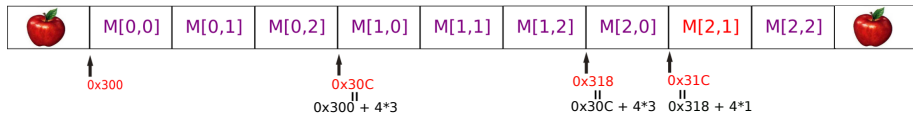
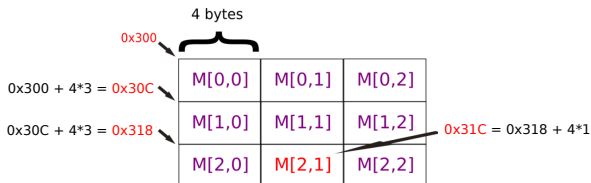


Matrices

- Suponiendo que el primer elemento de M se encuentra almacenado en la posición de memoria $0x300$ y queremos asignar un valor en $M[2,1]$ entonces en C hacemos:

```
int m[3][3];  
m[2][1] = 7;
```

- ¿Y en ensamblador?



Matrices

- En general, para indizar dentro de una matriz $M[i, j]$, la “regla” es:

“puntero al inicio de la matriz” +
+ “cantidad elementos de la fila” * “índice de fila” * “tamaño dato” +
+ “índice de columna” * “tamaño dato”

Matrices y punteros

Las matrices y los punteros también están relacionados. Si tenemos:

```
int m[3][4];
```

Para C: m es un puntero al primer elemento de la matriz.

Entonces podemos hacer en C:

```
int *p_m = m;
```

```
int *p_m = &m[0][0];
```

Ejercicios

Ejercicio 1

Dado un vector de n enteros de 16 bits, hacer una función que devuelva la suma de los elementos del vector.

El prototipo de la función es:

```
short suma(short* vector, short n);
```

Ejercicios

Ejercicio 1 - Solución

```
suma:                                .cicloSuma:
    ; RDI = vector                    add r12w, [rdi]
    ; SI = n                          lea rdi, [rdi+2]
                                      loop .cicloSuma

    push rbp
    mov rbp, rsp
    push r12

    mov rax, r12

    xor r12, r12
    xor rcx, rcx
    mov cx, si

    .fin:
        pop r12
        pop rbp
        ret
```

Ejercicios

Ejercicio 2

Dada una matriz de $n \times n$ enteros de 16 bits, hacer una función que devuelva los elementos de la diagonal en el vector pasado por parámetro.

El prototipo de la función es:

```
void diagonal(short* matriz, short n, short* vector);
```

Ejercicios

Más Ejercicios

Toda la Práctica 0 y la Práctica 1.

- `orga2.exp.dc.uba.ar` > *Prácticas y TPs* > *Prácticas*.
- `http://orga2.exp.dc.uba.ar?pid=7`