Clase Pre-Parcial

Organización del Computador II

Kevin F. G.

Departamento de Computación

2022-09-13

¿Qué vamos a hacer?

- Ejercicio C + ASM
- Ejercicio SIMD

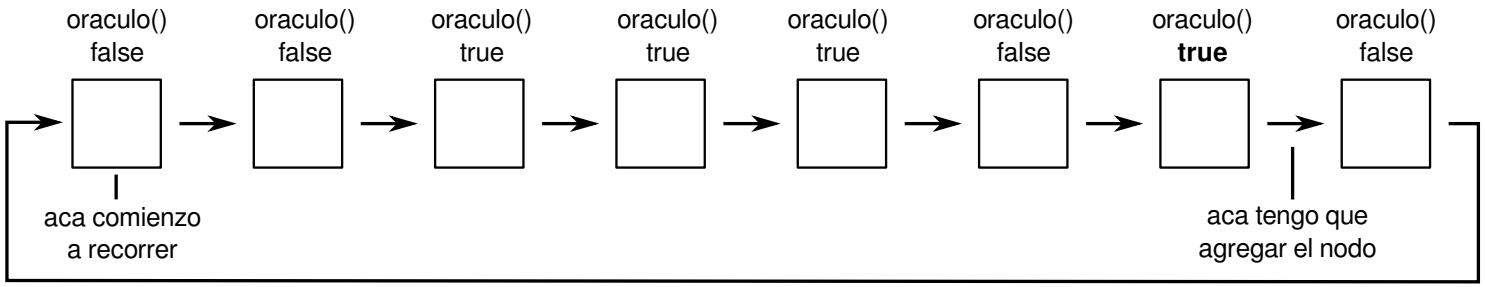
Sea una lista circular que respeta la siguiente estructura:

```
struct nodo {
  struct nodo* siguiente,
  int dato,
  bool (*oraculo)()
}
```

Donde `siguiente` es un puntero al siguiente nodo, `dato` es un valor entero almacenado y `oraculo` es un puntero a función que no recibe nada y devuelve un valor de tipo `bool`. `bool` es un entero de 4 bytes, que se interpreta como `false` si vale `0` y `true` en caso contrario.

Se nos pide implementar la función en C y ASM

- `void insertarDespuesDelUltimoTrue(nodo* listaCircular, int nuevoDato, bool (*nuevoOraculo)())`:
  Inserta un nuevo nodo después del último nodo para el cual el llamado a su `oraculo` devuelva `true`. El nuevo nodo debe contener los campos `dato` y `oraculo` pasados por parámetro. Si el orácul de ningún nodo devuelve `true`, no se debe insertar nada.



```
void insertarDespuesDelUltimoTrue(nodo* lC, int nD, bool (*nO)())
```

¿Qué datos necesitamos? ¿Cuál es la idea de la solución?

`void insertarDespuesDelUltimoTrue(nodo* lC, int nD, bool (*nO)())`

¿Qué datos necesitamos? ¿Cuál es la idea de la solución?

- **encontrar** cuál es el último true

`void insertarDespuesDelUltimoTrue(nodo* lC, int nD, bool (*nO)())`

¿Qué datos necesitamos? ¿Cuál es la idea de la solución?

- **encontrar** cuál es el último true
- insertar el **nuevo** nodo

`void insertarDespuesDelUltimoTrue(nodo* lC, int nD, bool (*nO)())`

¿Qué datos necesitamos? ¿Cuál es la idea de la solución?

- **encontrar** cuál es el último true
- insertar el **nuevo** nodo
- asegurarnos de mantener la circularidad

`void insertarDespuesDelUltimoTrue(nodo* lC, int nD, bool (*nO)())`

¿Qué datos necesitamos? ¿Cuál es la idea de la solución?

- **encontrar** cuál es el último true
- insertar el **nuevo** nodo
- asegurarnos de mantener la circularidad

Pensar el pseudo-código!

C + ASM

`void insertarDespuesDelUltimoTrue(nodo* lC, int nD, bool (*nO)())`

```
ultimoTrue <- NULL
para cada nodo en lC:
    si nodo.oraculo() > 0:
        ultimoTrue <- nodo
    fin si
fin para
si ultimoTrue != NULL:
    insertarNuevoNodo(ultimoTrue, nD, nO)
fin si
```

¿Qué hace insertarNuevoNodo?

C + ASM

`void insertarDespuesDelUltimoTrue(nodo* lC, int nD, bool (*nO)())`

```
ultimoTrue <- NULL
para cada nodo en lC:
    si *nodo.oraculo() > 0:
        ultimoTrue <- nodo
    fin si
fin para
si ultimoTrue != NULL:
    // insertarNuevoNodo(ultimoTrue, nD, nO)
    nuevoNodo <- NuevoNodo() // -> *Nodo
    *ultimoTrue.siguiente <- nuevoNodo
    *nuevoNodo.dato <- nD
    *nuevoNodo.oraculo <- nO
fin si
```

¿Es la solución correcta?

C + ASM

`void insertarDespuesDelUltimoTrue(nodo* lC, int nD, bool (*nO)())`

```
ultimoTrue <- NULL
para cada nodo en lC:
    si *nodo.oraculo() > 0:
        ultimoTrue <- nodo
    fin si
fin para
si ultimoTrue != NULL:
    // insertarNuevoNodo(ultimoTrue, nD, nO)
    nuevoNodo = NuevoNodo() // -> *Nodo
    *nuevoNodo.siguiente <-  ultimoTrue.siguiente
    *nuevoNodo.dato <- nD
    *nuevoNodo.oraculo <- nO
    *ultimoTrue.siguiente <- nuevoNodo
fin si
```

Ahora sí, pasamos a C.

C + ASM

```
insertarDespuesDelUltimoTrue(nodo* lC, int nD, bool (*nO)()) {
  nodo* ultimoTrue = NULL;
  for nodo in lC {
    if (*nodo.oraculo() > 0) {
        ultimoTrue = nodo
    }
  }
  if (ultimoTrue != NULL) {
    insertarNuevoNodo(ultimoTrue, nD, nO)
  }
}
```

¿Como arreglamos el `for nodo in lC`?

¿Como implementamos `insertarNuevoNodo`?

C + ASM

```
insertarDespuesDelUltimoTrue(nodo* lC, int nD, bool (*nO)()) {
  nodo* nodoActual = lC;
  nodo* primerNodo = lC;
  nodo* ultimoTrue = NULL;
  do {
    if (*nodoActual.oraculo() > 0) {
      ultimoTrue = nodoActual;
    }
    nodoActual = *nodoActual.siguiente;
  } while (nodoActual != primerNodo);
  if (ultimoTrue != NULL) {
    nodo* nuevoNodo = malloc(sizeof(nodo));
    nuevoNodo->siguiente = ultimoTrue->siguiente;
    nuevoNodo->oraculo = nO;
    nuevoNodo->dato = nD;
    ultimoNodoTrue->siguiente = nuevoNodo;
  }
}
Ahora en ASM
```

C + ASM

`void insertarDespuesDelUltimoTrue(nodo* lC, int nD, bool (*nO)())`

¿Qué consideraciones hay que tener?

- cómo desplazarse dentro del nodo

C + ASM

`void insertarDespuesDelUltimoTrue(nodo* lC, int nD, bool (*nO)())`

¿Qué consideraciones hay que tener?

- cómo desplazarse dentro del nodo
- alinear la pila en los llamados a funciones externas

C + ASM

`void insertarDespuesDelUltimoTrue(nodo* lC, int nD, bool (*nO)())`

¿Qué consideraciones hay que tener?

- cómo desplazarse dentro del nodo
- alinear la pila en los llamados a funciones externas
- respetar la convención C en todos sus sentidos

Recordando la estructura

```
struct nodo {
  struct nodo* siguiente,
  int dato,
  bool (*oraculo)()
}
```

¿Cómo queda en memoria? ¿Cuál es el offset de cada elemento?
`nodo*` - `8 bytes`, por ser un puntero
`int` - `4 bytes`, por ser un entero
`bool*` - `8 bytes`, por ser un puntero
Quedando alineado a 8 bytes con un tamaño total de...

`void insertarDespuesDelUltimoTrue(nodo* lC, int nD, bool (*nO)())`

Recordando la estructura

```
struct nodo {
  struct nodo* siguiente,
  int dato,
  bool (*oraculo)()
}
```

¿Cómo queda en memoria? ¿Cuál es el offset de cada elemento?
`nodo*` - `8 bytes`, por ser un puntero
`int` - `4 bytes`, por ser un entero
`bool*` - `4 bytes`, por ser un puntero
Quedando alineado a 8 bytes con un tamaño total de... 24 bytes, por el padding luego del int, al no ser packed

C + ASM

insertarDespuesDelUltimoTrue(nodo* lC, int nD, bool (*nO)())

```
% define OFFSET_SIGUIENTE 0
% define OFFSET_DATO 8
% define OFFSET_ORACULO 16
extern malloc
; lC [?], nD [?], nO [?]
insertarDespuesDelUltimoTrue:
    ;stackframe prologo

    ;desarrollo

    ;stackframe epilogo
```

```
% define SIZE_NODO 24
% define NULL 0
```

C + ASM

insertarDespuesDelUltimoTrue(nodo* lC, int nD, bool (*nO)())

```
% define OFFSET_SIGUIENTE 0
% define OFFSET_DATO 8
% define OFFSET_ORACULO 16
% define SIZE_NODO 24
% define NULL 0
extern malloc
; lC [rdi], nD [rsi], nO [rdx]
insertarDespuesDelUltimoTrue:
    ;stackframe prologo
    push rbp          ; alineada
    mov rbp, rsp
    push rbx          ; desalineada, uso de nuevoDato
    push r12          ; alineada, uso de nuevoOraculo
    push r13          ; desalineada, uso de nodoActual
    push r14          ; alineada, uso de primerNodo
    push r15          ; desalineada, uso de ultimoTrue
    sub rsp, 8        ; alineada

    ;desarrollo
    ...
```

```
insertarDespuesDelUltimoTrue:
    ...
    ;desarrollo
    ...
    ;stackframe epilogo
    add rsp, 8
    pop r15
    pop r14
    pop r13
    pop r12
    pop rbx
    pop rbp
    ret
```

C + ASM

insertarDespuesDelUltimoTrue(nodo* lC, int nD, bool (*nO)())

```
; lC [rdi], nD [rsi], nO [rdx]
insertarDespuesDelUltimoTrue:
    ...      ; r13 nodoActual, r14 primerNodo, r15 ultimoTrue
    ;desarrollo
    mov rbx, rsi     ; nuevoDato
    mov r12, rdx     ; nuevoOraculo
    mov r13, rdi     ; nodoActual
    mov r14, rdi     ; primerNodo
    mov r15, NULL    ; ultimoTrue
    .ciclo:
        call [r13 +OFFSET_ORACULO] ; llamo al oraculo
        cmp rax, 0 ; if (nodoActual.oraculo() > 0)
        jz .esFalse
        mov r15, r13 ; guardo el actual como ultimoTrue
        .esFalse:
            mov r13, [r13 + OFFSET_SIGUIENTE] ;voy al siguiente
        cmp r13, r14 ; valido si di la vuelta
        jnz .ciclo
    ...
```

C + ASM

insertarDespuesDelUltimoTrue(nodo* lC, int nD, bool (*nO)())

```
; lC [rdi], nD [rsi], nO [rdx]
insertarDespuesDelUltimoTrue:
    ...      ; rbx nuevoDato, r12 nuevoOraculo
    ...      ; r13 nodoActual, r14 primerNodo, r15 ultimoTrue
    cmp r15, NULL
    jz .fin ; ningun oraculo dijo true
    mov rdi, SIZE_NODO
    call malloc
    mov r8, [r15 + OFFSET_SIGUIENTE]
    mov [rax + OFFSET_SIGUIENTE], r8
    mov [rax + OFFSET_DATO], rbx
    mov [rax + OFFSET_ORACULO], r12
    mov [r15 + OFFSET_SIGUIENTE], rax
    .fin:
    ; stackframe epilogo
    ...
```

C + ASM

insertarDespuesDelUltimoTrue(nodo* lC, int nD, bool (*nO)())

```
; lC [rdi], nD [rsi], nO [rdx]
insertarDespuesDelUltimoTrue:
    ...      ; rbx nuevoDato, r12 nuevoOraculo
    ...      ; r13 nodoActual, r14 primerNodo, r15 ultimoTrue
    cmp r15, NULL
    jz .fin ; ningun oraculo dijo true
    mov rdi, SIZE_NODO
    call malloc
    mov r8, [r15 + OFFSET_SIGUIENTE]
    mov [rax + OFFSET_SIGUIENTE], r8
    mov [rax + OFFSET_DATO], rbx ; por que esto funciona?
    mov [rax + OFFSET_ORACULO], r12
    mov [r15 + OFFSET_SIGUIENTE], rax
    .fin:
    ; stackframe epilogo
    ...
```

insertarDespuesDelUltimoTrue(nodo* lC, int nD, bool (*nO)())

```
; lC [rdi], nD [rsi], nO [rdx]
insertarDespuesDelUltimoTrue:
    ...      ; rbx nuevoDato, r12 nuevoOraculo
    ...      ; r13 nodoActual, r14 primerNodo, r15 ultimoTrue
    cmp r15, NULL
    jz .fin ; ningun oraculo dijo true
    mov rdi, SIZE_NODO
    call malloc
    mov r8, [r15 + OFFSET_SIGUIENTE]
    mov [rax + OFFSET_SIGUIENTE], r8
    mov [rax + OFFSET_DATO], rbx ; por que esto funciona?
    mov [rax + OFFSET_ORACULO], r12
    mov [r15 + OFFSET_SIGUIENTE], rax
    .fin:
    ; stackframe epilogo
    ...
```

porque la estructura Nodo tiene padding luego del dato

¿Preguntas?

¿Preguntas?
Vamos al siguiente ejercicio

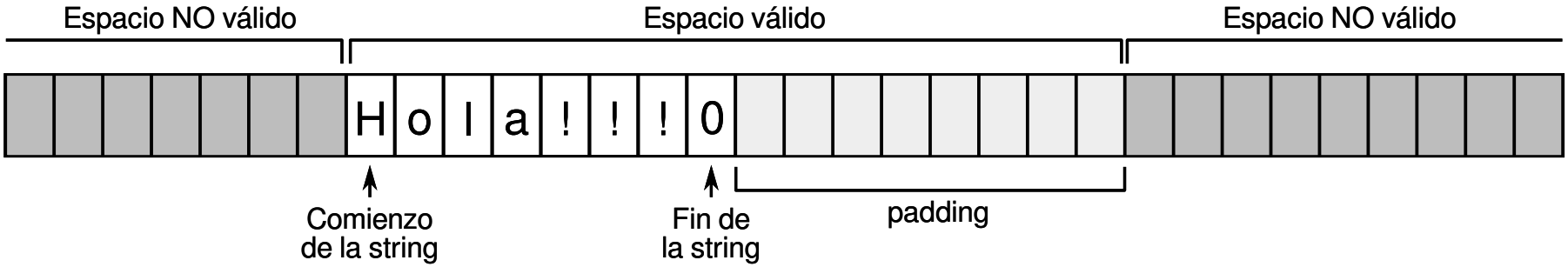
Un string de C es una cadena de caracteres de un byte codificados usando ASCII que termina en 0 (nulo). Por ejemplo, la cadena `Hola!!!` se codifica:

`H`	`o`	`l`	`a`	`!`	`!`	`!`	
`0x48`	`0x6f`	`0x6c`	`0x61`	`0x21`	`0x21`	`0x21`	`0x00`

Se pide implementar usando SIMD:

- `double promedio(char* str)`:
  Calcula el promedio de los caracteres del string, sin considerar el caracter nulo.
  Por ejemplo, para la string anterior:
  $\frac{\texttt{0x48}+\texttt{0x6f}+\texttt{0x6c}+\texttt{0x61}+\texttt{0x21}+\texttt{0x21}+\texttt{0x21}}{7} = 69{,}57\ldots.$

Nota: La longitud de la string puede ser arbitrariamente grande, considerar que el área ocupada por la string es múltiplo de 16 bytes, es decir, se agrega *padding* al final de la misma que puede ser leído y modificado. Suponer a demas que este *padding* es inicialmente cero.



SIMD

`double promedio(char* str)`

¿Qué información nos da el enunciado? ¿Cuál es la idea de la solución?

SIMD

`double promedio(char* str)`

¿Qué información nos da el enunciado? ¿Cuál es la idea de la solución?

- cada string ocupa un espacion total múltiplo de 16

SIMD

`double promedio(char* str)`

¿Qué información nos da el enunciado? ¿Cuál es la idea de la solución?

- cada string ocupa un espacion total múltiplo de 16
- si es necesario, el string tiene padding luego del caracter nulo

SIMD

`double promedio(char* str)`

¿Qué información nos da el enunciado? ¿Cuál es la idea de la solución?

- cada string ocupa un espacion total múltiplo de 16
- si es necesario, el string tiene padding luego del caracter nulo
- hay que usar simd

SIMD

`double promedio(char* str)`

¿Qué información nos da el enunciado? ¿Cuál es la idea de la solución?

- cada string ocupa un espacion total múltiplo de 16
- si es necesario, el string tiene padding luego del caracter nulo
- hay que usar simd
- necesitamos el largo del string?

SIMD

`double promedio(char* str)`

¿Qué información nos da el enunciado? ¿Cuál es la idea de la solución?

- cada string ocupa un espacion total múltiplo de 16
- si es necesario, el string tiene padding luego del caracter nulo
- hay que usar simd
- necesitamos el largo del string

vamos con pseudo-código

SIMD

double promedio(char* str)

```
strLen <- calcularLong(str)
actual <- 0
res <- 0
mientras actual < strLen:
    xmm <- strLen[actual]
    res <- res + calcularSumaParcial(xmm)
    actual <- actual + 16
fin mientras
devolver res/strLen
```

¿Qué hace calcularSumaParcial?

SIMD

double promedio(char* str)

```
strLen <- calcularLong(str)
actual <- 0
res <- 0
mientras actual < strLen:
    xmm <- strLen[actual]
    res <- res + calcularSumaParcial(xmm)
    actual <- actual + 16
fin mientras
devolver res/strLen
```

¿Qué hace calcularSumaParcial?

Vamos con ASM

- punpckhbw, punpcklbw

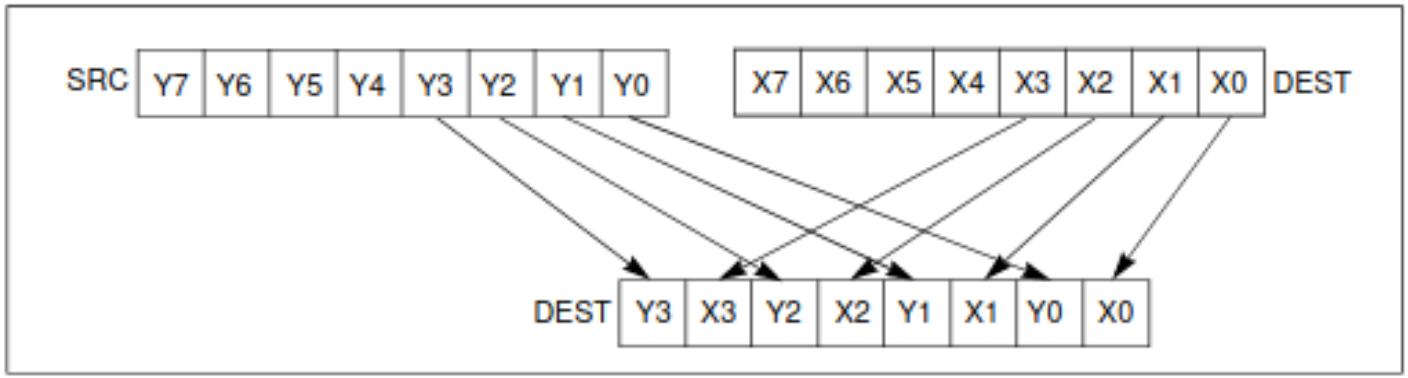
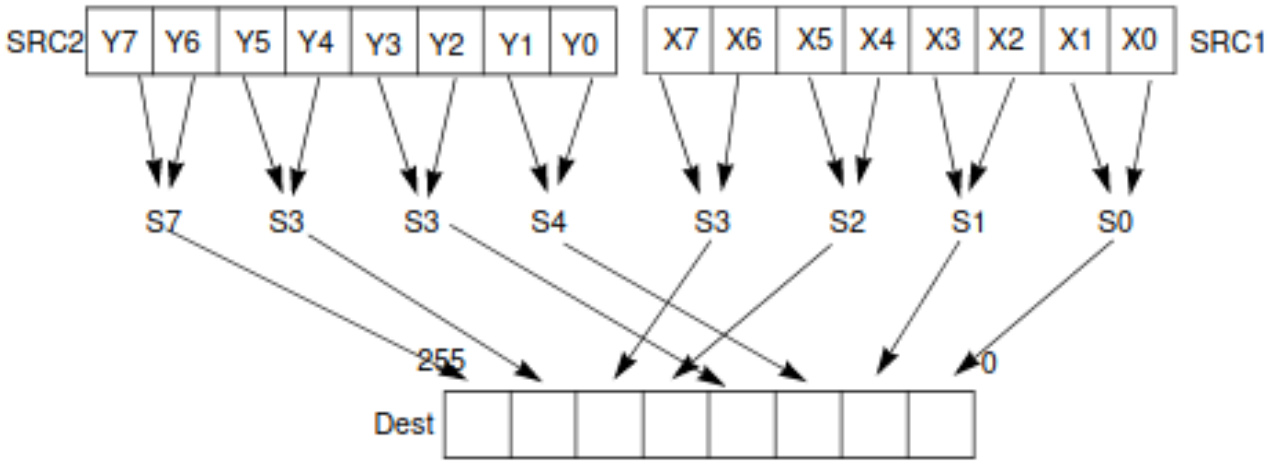


Figure 4-22. PUNPCKLBW Instruction Operation Using 64-bit Operands

- phaddw


SRC2
Y7
Y6
Y5
Y4
Y3
Y2
Y1
Y0
X7
X6
X5
X4
X3
X2
X1
X0
SRC1
S7
S3
S3
S4
S3
S2
S1
S0
255
0
Dest

SIMD

double promedio(char* str)

```
; char* str [rdi]
promedio:
;stackframe prologo
push rbp
mov rbp, rsp

;desarrollo
; calcular largo
; hacer suma

;stackframe epilogo
pop rbp
ret
```

SIMD

double promedio(char* str)

```
; char* str [rdi]
promedio:
...
;desarrollo
xor rcx, rcx ; strLen
mov rsi, rdi
.largo:
    cmp byte [rsi], NULL
    je .finLargo
    inc rcx
    inc rsi
    jmp .largo
.finLargo: ;
mov rax, rcx ; copio strLen
pxor xmm0, xmm0 ; limpio 2 registros que voy a usar
pxor xmm2, xmm2
.ciclo:
...
```

SIMD

double promedio(char* str)

```
; char* str [rdi]
; rcx y rax son strLen, xmm0 y xmm2 limpios
.ciclo:
    movdqu xmm1, [rdi] ; muevo los 16 bytes de str a xmm1
    movdqa xmm3, xmm1  ; copio de xmm1 a xmm3
    ; extiendo la parte baja de xmm1 de byte a word
    punpcklbw xmm1, xmm2 ; quedan 8 words
    ; extiendo la parte alta de xmm3 de byte a word
    punpckhbw xmm3, xmm2 ; quedan 8 words
    call sumaParcial ; es una funcion propia
    cvtdq2pd xmm1, xmm1 ; convierto a double
    addpd xmm0, xmm1 ; sumo a res la suma parcial
    lea rdi, [rdi + 16] ; avanzo el *char
    sub rcx, 16 ; avanzo 16 en el contador
    cmp rcx, 0
    jg .ciclo ; si no llegue a 0, vuelvo
.finCiclo:
...
```

double promedio(char* str)

```
sumaParcial:
    phaddw xmm1, xmm3
    ; |15|14|13|12|11|10|9|8|, |7|6|5|4|3|2|1|0|
    ; |15+14|13+12|7+6|5+4|11+10|9+8|3+2|1+0|
    phaddw xmm1, xmm1
    ;|15+14+13+12|11+10+9+8|15+14+13+12|11+10+9+8
    ;   |7+6+5+4|3+2+1+0|7+6+5+4|3+2+1+0|
    phaddw xmm1, xmm1
    ;|15+14+13+12+11+10+9+8|=|=|=|7+6+5+4+3+2+1+0|=|=|=
    movdqa xmm3, xmm1 ; copio resultado a xmm3
    punpcklwd xmm1, xmm2 ; extiendo parte baja a dw
    punpckhwd xmm3, xmm2 ; extiendo parte alta a dw
        ; 15+14+13+12+11+10+9+8|=|=|=| xmm3
        ; 7+6+5+4+3+2+1+0|=|=|= xmm1
    paddd xmm1, xmm3 ;|15+14+13+12+11+10+9+8...+0|=|=|=
ret
```

SIMD

double promedio(char* str)

```
; char* str [rdi]
promedio:
...
;desarrollo
... ; rcx es strLen, xmm0 sumaParcial
.finCiclo:
   cvtsi2sd  xmm2, rax ; paso strLen a double en xmm2
   divpd xmm0, xmm2 ; divido
;stackframe epilogo
...
```

donde queda el resultado?