Nº Orden	Apellido y nombre	L.U.	Cantidad de hojas
66	HERTZULIS NICOLAS	811/15	OCHO (8)

Organización del Computador 2
Primer parcial – 02/10/18

1 (40)	2 (40)	3 (20)	
37	29	30	86 (A)

Paula

Normas generales

- Numere las hojas entregadas. Complete en la primera hoja la cantidad total de hojas entregadas.
- Entregue esta hoja junto al examen, la misma **no** se incluye en la cantidad total de hojas entregadas.
- Está permitido tener los manuales y los apuntes con las listas de instrucciones en el examen. Está prohibido compartir manuales o apuntes entre alumnos durante el examen.
- Cada ejercicio debe realizarse en hojas separadas y numeradas. Debe identificarse cada hoja con nombre, apellido y LU.
- La devolución de los exámenes corregidos es personal. Los pedidos de revisión se realizarán por escrito, antes de retirar el examen corregido del aula.
- Los parciales tienen tres notas: I (Insuficiente): 0 a 59 pts, A- (Aprobado condicional): 60 a 64 pts y A (Aprobado): 65 a 100 pts. No se puede aprobar con A- ambos parciales. Los recuperatorios tienen dos notas: I: 0 a 64 pts y A: 65 a 100 pts.

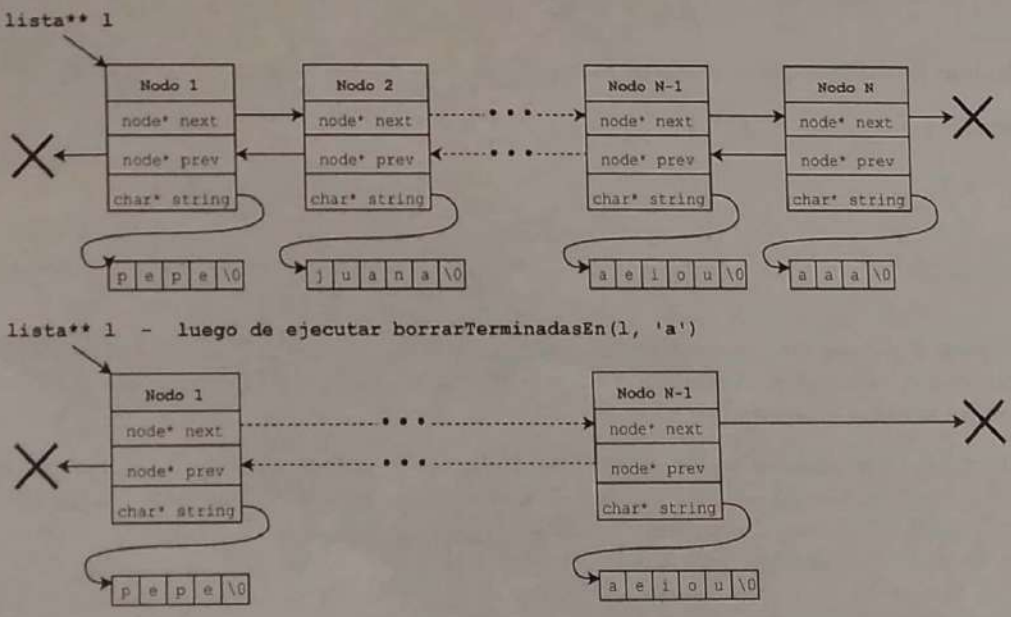
Ej. 1. (40 puntos)

Considerar una estructura de lista doblemente enlazada en donde cada nodo almacena un string de C, es decir, un arreglo de caracteres finalizado en el caracter nulo (`'\0'`).

```
typedef struct node_t {
    struct node_t *next;
    struct node_t *prev;
    char *string;
} node;
```

(20p) a. Escribir en ASM una función que reciba como parámetros `l`: *doble puntero a nodo* y `c`: *un caracter*, y borre todos los nodos, junto con la string, para los que el último caracter sea igual a `c`. Utilizar la función `free` para borrar tanto los nodos como las strings. Su aridad es: `void borrarTerminadasEn(node** l, char c)`. En el caso de borrar el primer elemento de la lista se debe actualizar el puntero recibido.

Resultado de aplicar `borrarTerminadasEn(l, 'a')`



(20p) b. Escribir en ASM la función `char* superConcatenar(node* n)`, que toma un puntero a un nodo y retorna la concatenación de todas las strings almacenadas en la lista. Se cuenta con la función `uint32_t strlen(char* s)` que dada una string, retorna la cantidad de caracteres válidos de la misma.

Ej. 2. (40 puntos)

Suponiendo un arreglo de números enteros con signo de 16 bits, tal que su longitud sea múltiplo de 8. Escribir en ASM utilizando instrucciones de SIMD las siguientes funciones:

(20p) a. Una función que cuente la cantidad de valores del arreglo que su codificación binaria comienza con `101` o `1001`. La aridad de la función es: `int contar(short* a, int n)`, donde `a` es el arreglo y `n` la cantidad de elementos.

(20p) b. Una función que realice sobre punto flotante la operación:

$$\sum_{i=0}^{n-2} a[i] - f \cdot a[i+1]$$

La aridad de la función es: `float sumarDiferencias(short* a, int n, float f)`, donde `a` es el arreglo, `n` su cantidad de elementos y `f` un valor en punto flotante pasado por parámetro.

Nota: Procesar la máxima cantidad posible de elementos en simultáneo.

Ej. 3. (20 puntos)

Suponer código donde todas las funciones construyen el *stack-frame* de la siguiente forma:

```
PUSH RBP
MOV RBP, RSP
...
POP RBP
RET
```

(5p) a. Explicar los cambios que se producen en la pila cuando se ejecuta el siguiente código genérico:

```
main() {
    ...
    A();
    ...
}
```

```
A() {
    ...
    B();
    ...
}
```

```
B() {
    ...
    C();
    ...
}
```

```
C() {
    ...
}
```

Donde `A`, `B` y `C` son funciones cualesquiera. Dibujar el estado de la pila antes de ejecutar la primer instrución de la función `C`. Indentificar para cada función dentro de la pila, la dirección de retorno, los registros salvados y las variables locales.

(15p) b. Entre llamados a funciones, se utiliza en la pila una cierta cantidad de bytes determinada por el código de cada función. Escribir en ASM una función que calcule el promedio de bytes utilizados dentro de la pila desde el inicio de la ejecución y a través de los sucesivos llamados a funciones, y retorne dicho valor en float de doble precisión. Considerar para ello que inicialmente el valor de `RBP` fue cero, y que la función no debe considerarse a sí misma en la cuenta. La aridad de la función es: `double promedioDeBytesUtilizadosEnLaPila()`.

NICOLAS HERTZULIS 8/11/15 HOJA 1 DE 8

1) a)

```
void borrarTerminadasEn(node **l, char c)
    node* actual = *l;
    node* aux;
    while (actual != NULL) {
        if (TerminaEn(actual->string, c)) {
            if (actual->prev != NULL) {
                actual->prev->next = actual->next;
            } else {
                *l = actual->next;
            }
            aux = actual;
            actual = actual->next;
            if (actual != NULL) {
                actual->prev = aux->prev;
            }
            free(aux->string);
            free(aux);
        } else {
            actual = actual->next;
        }
    }
} //void
```

(A)

(B)

```
int TerminaEn(char* s, char c) {
    if (*s == 0) {
        return 0;
    }
    while (*(s+1) != 0) {
        ++s;
    }
    return (*s == c);
}
```

ASM: % define NULL, 0

```
borrarTerminadasEn:
; rdi = node** l
; sil = char c
push rbp
mov rbp, rsp
push rbx
push r12
push r13
push r14
mov rbx, rdi        ; **l
mov r12, [rbx]      ; actual
mov r13b, sil       ; c

.ciclo:
cmp r12, NULL
je .fin
```

```
mov rdi, [r12+offset_string]
cmp byte [rdi], 0
je .siguiente

.ciclo_interno:
cmp byte [rdi+1], 0
je .fin_ciclo_interno
inc rdi
jmp .ciclo_interno
.fin_ciclo_interno:

cmp [rdi], r13b
jne .siguiente

mov rsi, [r12+offset_next]
mov rdi, [r12+offset_prev]

cmp rdi, 0
je .else_1
    mov [rdi+offset_next], rsi
    jmp .end_if_1
.else_1:
    mov [rbx], rsi
.end_if_1:
```

ok último elemento de la string

todo esto es la guarda de la string actual termina en (c)

parte (A) del código C

```
mov r14, r12
mov r12, rsi
cmp r12, 0
je .end_if_2
    mov [r12+offset_prev], rdi
.end_if_2:
mov rdi, [r14+offset_string] ok
call free ok
mov rdi, r14
call free ok
```

Esto es la parte (B) del código C

```
.siguiente:
mov r12, [r12+offset_next]
jmp .ciclo

.fin:
pop r14
pop r13
pop r12
pop rbx
pop rbp
ret
```

NICOLAS HERTZULIS 811/15 HOJA 3 DE 8

1) b)

```
char * superConcatenar(node * n) {
    uint32_t len = 0
    node* m = n;
    while (m != NULL) {
        len += strlen(m→string);
        m = m→next;
    }
    char * s = malloc(len+1);
    uint32_t i = 0
    while (n != NULL) {
        char* s2 = n→string;
        while (*s2 != 0) {
            s[i] = *s2;
            i++; s2++;
        }
        n = n→next;
    }
    s[i] = 0;
    return s;
}
```

A

B

Super Concatenar:

;rdi = nodo* n;

push rbp ✓

mov rbp, rsp ✓

push rbx ; push r12 ; push r13 ✗ pila desalineada

mov rbx, rdi ; mov r12, rdi ; ~~xor~~ r13, r13 ok

~~mov rcx, rdi~~
~~xor rbx, rbx~~

.ciclo_len:

~~cmp r12, 0~~ ✓
cmp r12, 0
je .fin_ciclo_len
mov rdi, [r12+offset_string] ✓
call str_len ✓
add r13d, eax ✓
mov r12, [r12+offset_next] ✓
jmp .ciclo_len
.fin_ciclo_len

parte A del código C

inc r13 ok
mov rdi, r13 ✓
call malloc ✓
xor rcx, rcx

.ciclo:
cmp rbx, NULL ?
je .fin_ciclo

NICOLAS HERTZULIS 8/11/15 HOJA 4 DE 8

parte B del código C

```
mov rdx, [rbx + offset_string]
mov r8b, [rdx]
.ciclo_interno:
cmp r8b, 0
je .fin_ciclo_interno
   mov [rax+rcx], r8b
   inc rcx
   inc rdx
   mov r8b, [rdx]
jmp .ciclo_interno
.fin_ciclo_interno:

mov rbx, [rbx + offset_next]
jmp .ciclo
.fin_ciclo:
mov byte[rax+rcx], 0

pop r13
pop r12
pop rbx
pop rbp
ret
```

NICOLAS HERTZOIS 841/15 11/04 6 17 $

2) (a) cantar: ¿Comentar C?

;rdi = short * a
;esi = int n

mov ecx, esi
shr ecx, 3
pxor xmm7, xmm7
pxor xmm8, xmm8
.ciclo:
movdqu xmm0, [rdi]
pand xmm0, xmm15 → paso (1) Pisar con cero los bytes menos signif de c/ short

movdqa xmm15, [mask1]
movdqa xmm14, [mask2]
movdqa xmm13, [mask3]
movdqa xmm12, [mask4]
movdqa xmm11, [mask5]

pxor xmm1, xmm1
punpcklwd xmm1, xmm0
pxor xmm2, xmm2
punpckhwd xmm2, xmm0
} paso (2) Desempaquetar a word double

movdqa xmm3, xmm1
pcmpeqd xmm3, xmm14
movdqa xmm4, xmm2
pcmpeqd xmm4, xmm14
} 0xC000

movdqa xmm5, xmm1
pcmpeqd xmm5, xmm13
movdqa xmm6, xmm2
pcmpeqd xmm6, xmm13
} 0xD000

paso (3) Comparar con 0xC000, 0xD000, 0x9000

Falta explicar las comparaciones

xmm1 := 1011 0000 0000 0000 = -0
xmm15 := 1101 0000 0000 0000 = -0
xmm5 := 0000 0000 0000 0000 = 0

pcmpeqd xmm1, xmm12
pcmpeqd xmm2, xmm12
} 0x9000

```
por xmm1, xmm3
por xmm1, xmm5      xmm1 := xmm9
por xmm2, xmm4
por xmm2, xmm6
```
paso (4) Unir comparaciones

```
pand xmm1, xmm11
pand xmm2, xmm11
```
paso (5) Transformar los 0xFFFFFFFF en 0x1

```
paddd xmm7, xmm1
paddd xmm8, xmm2
```
paso (6) sumar a la cuenta total

```
add rdi, 16
loop .ciclo
paddd xmm7, xmm8
phaddd xmm7, xmm7
phaddd xmm7, xmm7
movd eax, xmm7
ret

section .rodata
align 16
masc1: times 8 DW 0xF000
masc2: times 4 DD 0xC000 0000
masc3: times 4 DD 0xD000 0000
masc4: times 4 DD 0x9000 0000
masc5: times 4 DD 0x1
```
paso (7) suma horizontal

2) b)

sumarDiferencias:
; rdi = short * a
; esi = int n
; edx = float f ¿Convención C?

```
movss xmm1, edx
pslldq xmm1, 4      ¿bits?
movss xmm1, edi
pslldq xmm1, 4
movss xmm1, edi
pslldq xmm1, 4
movss xmm1, edi
```

paso (1) xmm1 = f|f|f|f ok

```
pxor xmm2, xmm2     ; a[i] parte baja
pxor xmm3, xmm3     ; a[i] parte alta   ok
pxor xmm4, xmm4     } a[i+1]
pxor xmm5, xmm5
```

Medio

```
mov ecx, esi
shr ecx, 3          ; proceso de a 8 short   ok
dec ecx
```

.ciclo:

movdqu xmm0, [rdi] ; a[i] ok
movdqu xmm6, [rdi+1] ; a[i+1] ok

pxor xmm7, xmm7
punpcklwd xmm7, xmm0
pxor xmm8, xmm8
punpckhwd xmm8, xmm0
psrld xmm7, 8
psrld xmm8, 8

} a[i] alta y baja

pxor xmm9, xmm9
punpcklwd xmm9, xmm6
psrld xmm9, 8

pxor xmm10, xmm10
punpckhwd xmm10, xmm6
psrld xmm10, 8

} a[i+1] alta y baja

paddd xmm2, xmm7
paddd xmm3, xmm8

} a[i] parcial

paddd xmm4, xmm9
paddd xmm5, xmm10

} a[i+1] parcial

suma de a[i] y a[i+1]

add rdi, 16
loop .ciclo

~~mulps xmm4, xmm1~~
~~mulps x~~
~~phadd~~

phaddd xmm2, xmm2
phaddd xmm2, xmm2
→ repito esto con xmm3, xmm4 y xmm5 ok

mulps xmm4, xmm1
mulps xmm5, xmm1
→ en xmm1 puse la f cuatro veces pero con una vez alcanzaba ok

subps xmm2, xmm4
subps xmm3, xmm5
→ xmm3/xmm2 ~~= a[i] - f(a[i])~~

~~addps~~

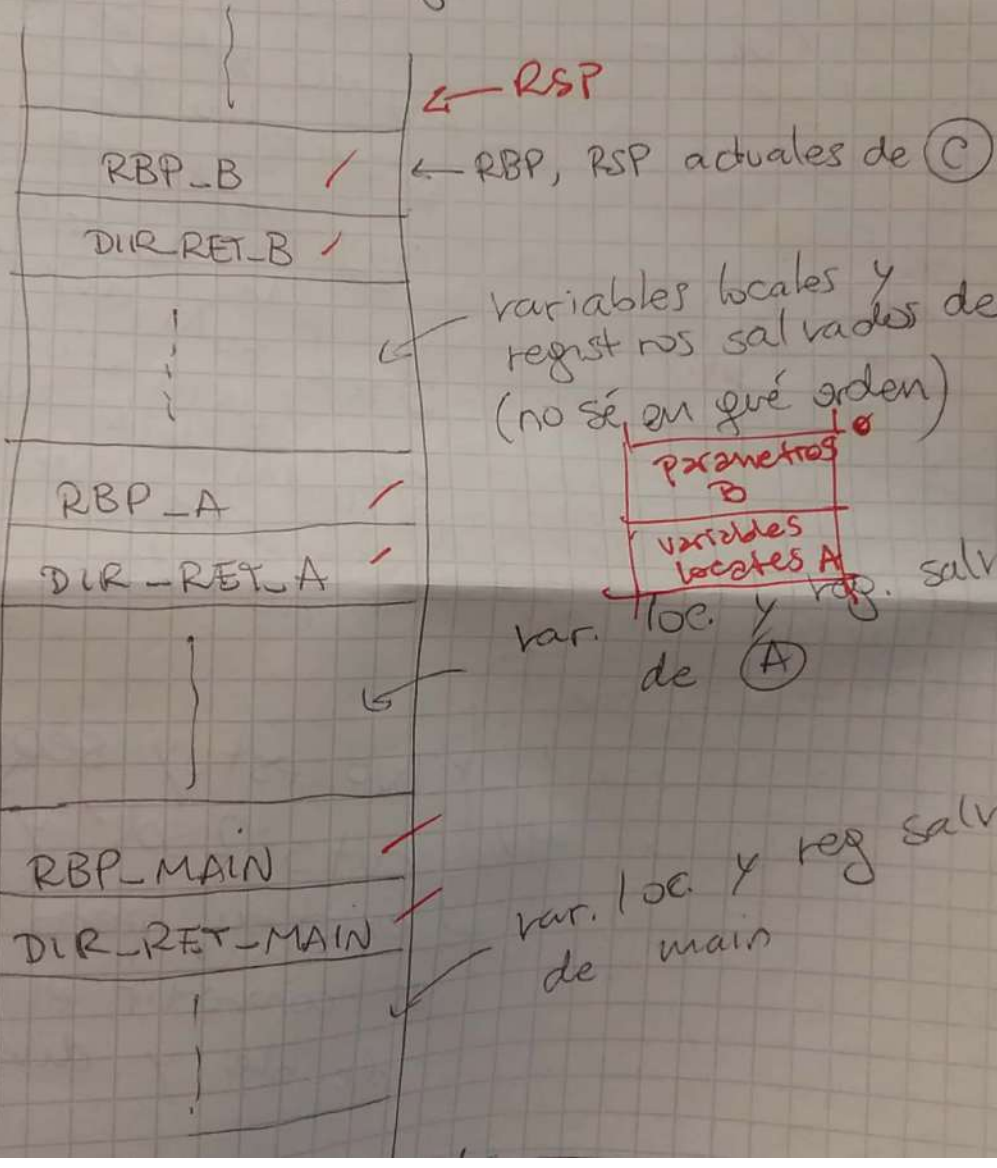
movdqu xmm0, xmm2
addps xmm0, xmm3 ; xmm0 = resultado final
ret ¿convención C?

Nota: Este ejercicio (2b), lo escribí muy rápido. Es posible y muy probable que haya errores menores. En particular me faltó procesar los últimos valores fuera del ciclo pero no tengo más tiempo (21:40)

La idea esta bien

NICOLAS HERTZULIS 8/11/15 HOJA 8 DE 8

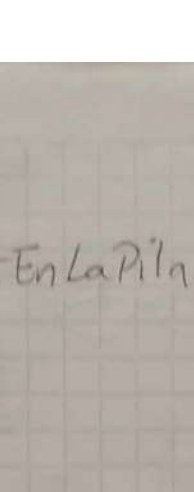
3) a)



3) b)

promedioDeBytesUtilizadosEnLaPila:

```
xor rcx, rcx
xor rax, rax
mov rsi, rsp
mov rdi, rbp
.ciclo:
cmp rdi, 0          ok
je .fin_ciclo       ok
add rax, rdi
sub rax, rsi
sub rax, 8
mov rdi, [rdi]      ok
mov rsi, [rdi+16]
inc rcx
jmp .ciclo
.fin_ciclo:
cvtsi2sd xmm0, rax
cvtsi2sd xmm1, rcx
divpd xmm0, xmm1
ret
```



Asumo que se quiere el promedio de las funciones SIN contar el uso de la pila desde main. O sea, solo doy el promedio de bytes utilizados por funciones llamadas desde main (y llamados internos de cada función) pero no de main. ok

No estoy seguro si cvtsi2sd y divpd existen, pero la idea es convertir el escalar el double y dividirlo ok (divsd)