Orga II cheatsheet

L.B, T.F, A.R

Secciones del programa (section)

.data	declaraciones de variables globales con valores iniciales
.rodata	declaraciones de constantes globales con valores particulares
.bss	contiene declaraciones de variables globales de valor no determinado; indicadores RESB, RESW, RESD, RESQ
.text	contiene la lista de instrucciones que compone el texto del programa a ejecutar

Directivas al ensamblador

Datos	DB, DW, DD, DQ, RESB, RESW, RESD, RESQ
Posicin	$
Macro	EQU
Constantes	%define

Llamadas al SO

- El ndice de funcin debe cargarse en rax
- Los parmetros se ingresan en los registros rbx, rcx, rdx, rsi, rdi y rbp sucesivamente
- Se genera la interrupcin de software int 0x80
- El resultado est en RAX

Convencion C

64 bits:

- Preservar RBX, R12, R13, R14 y R15
- Retornar el resultado en RAX o XMM0
- No romper la pila
- Para llamar a funciones de C, pila alineada a 16B

32 bits:

- Preservar EBX, ESI y EDI
- Retornar el resultado en EAX
- No romper la pila
- Para llamar a funciones de C, pila alineada a 4B

Llamado de funciones

64 bits:

Enteros	RDI, RSI, RDX, RCX, R8 y R9
Punto Flotante	XMM
Resto	Pila, para que se popeen en orden

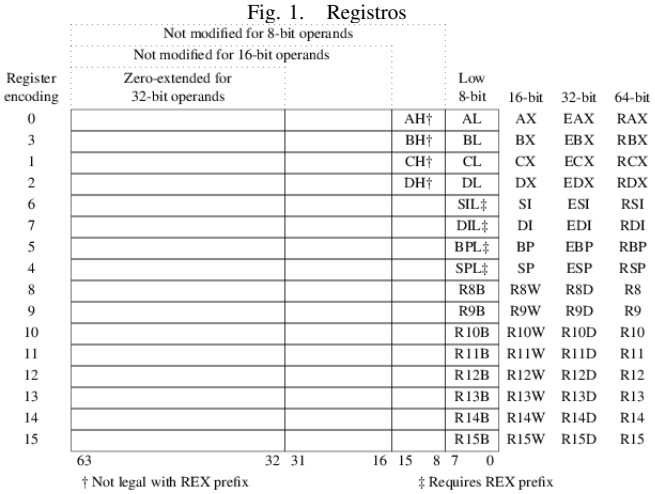
32 bits: Por pila, se pushean de derecha a izquierda

Tamaos

ASM - C:

void*	64 bits, 8 bytes. Addr direccionable.
int	32 bits, 4 bytes
char	8 bits, 1 byte
int8_t	8 bits, 1 byte
int32_t	32 bits, 4 bytes
float	32 bits, 4 bytes
double	64 bits, 4 bytes

Fig. 1. Registros



xmms	128 bits, 16 bytes
xmms	4 floats, 2 double, 4 unsigned int, 4 int
float	bit sign, exponent, significand
0xFF	4 bits, 1 byte
word	2 bytes 16 bits
dw	4 bytes 32 bits
qword	8 bytes 64 bits
byte	1 byte 8 bits
direcc	[reg+reg*scalar+desp], scalar=1,2,4,8 desp=imm32bits

Listing 1. Define, funciones externas, equ

```
extern free
extern malloc
%define LISTA_LONGITUD 0
```

La alineacion de la pila esta definida por RSP.
[pushes]
[8 local]
[8 local]
[8 local]
[viejo rbp]
[dir retorno]

VECTORES

[Base + Indice*scala +/- Desplazamiento]
Base = algun registro
Indice = algun registro
scala = 1, 2, 4 u 8
Desplazamiento = inmediato de 32 bits

MATRICES

puntero al inicio de la matriz + cantidad elementos de la fila * indice de fila * tamao dato + indice de columna * tamao dato

INSTRUCCIONES ASM

lea reg, [reg1 + reg2*t + k]

Tiene el mismo formato que un modo de direccionamiento Donde t es 1,2,4,8

k es un inmediato de 64 bits; y los registros son de 64 bits. Nota importante: No modifica las flags,

a veces es util usar lea en lugar de add cuando se quieren preservar las flags.

Listing 2. Loop

```
mov ecx, 5
 start_loop:
 ; the code here would be executed 5 times
 loop start_loop
```

Listing 3. Otras

PUNPCKLDQ

```
Shift Double Quadword Left Logical
instruction with 128-bit operands:
Destination[0..31] = Destination[0..31];
Destination[32..63] = Source[0..31];
Destination[64..95] = Destination[32..63];
Destination[96..127] = Source[32..63];
```

PMULLW

```
toma la parte baja de la multiplicacion,
enteros sin signo
Temporary0[0..31] = Destination[0..15]
                  * Source[0..15];
Temporary1[0..31] = Destination[16..31]
                  * Source[16..31];
Destination[0..15] = Temporary0[0..15];
Destination[16..31] = Temporary1[0..15];
```

PHADDW suma horizontal

```
xmm1[15-0] = xmm1[31-16] + xmm1[15-0];
xmm1[31-16] = xmm1[63-48] + xmm1[47-32];
xmm1[47-32] = xmm1[95-80] + xmm1[79-64];
xmm1[63-48] = xmm1[127-112] + xmm1[111-96];
xmm1[79-64] = xmm2/m128[31-16]
        + xmm2/m128[15-0];
xmm1[95-80] = xmm2/m128[63-48]
        + xmm2/m128[47-32];
xmm1[111-96] = xmm2/m128[95-80]
        + xmm2/m128[79-64];
xmm1[127-112] = xmm2/m128[127-112]
        + xmm2/m128[111-96]
```

PHADDD

```
xmm1[31-0] = xmm1[63-32] + xmm1[31-0];
xmm1[63-32] = xmm1[127-96] + xmm1[95-64];
xmm1[95-64] = xmm2/m128[63-32]
 + xmm2/m128[31-0];
xmm1[127-96] = xmm2/m128[127-96]
 + xmm2/m128[95-64];
```

PSRLW/PSRLQ

```
; Shift Packed Data Right Logical
;PSRLQ instruction with 128-bit operand:
if(Count > 15) Destination[128..0] = 0;
else {
Destination[0..63]
= ZeroExtend(Destination[0..63] >> Count);
=ZeroExtend(Destination[64..127] >> Count);
}
psllw/pslld/psllq (para ints)
Shift Packed Data left Logical
if(Count > 63) Destination[0..128] = 0;
else {
Destination[0..63]
= ZeroExtend(Destination[0..63] << Count);
Destination[64..127]
= ZeroExtend(Destination[64..127] << Count);
}
```

```
psraw/psrad/psraq
(para floats) NO EXISTE LEFT
Shift Packed Data Right Arithmetic
psrad
if(Count > 31) Count = 32;
Destination[0..31]
 = SignExtend(Destination[0..31] >> Count);
Destination[32..63]
 = SignExtend(Destination[32..63] >> Count);
Destination[64..95]
 = SignExtend(Destination[64..95] >> Count);
Destination[96..127]
 = SignExtend(Destination[96..127] >>Count);
```

```
pandn packed and not
pxor packed xor
por packed or
pand packed and
```

```
Insert Byte/Dword/Qword
pisrb/pinsrd/pinsrq
pinsrd xmm4, r10d, 0
; inserta r10 en pos 0 del xmm [x,x,x,r10]
pinsrq xmm1, r/m64, imm8

;maskleft: [a,b,c,d] -> [a,0,c,0]
maskLeft: DD 0, -1, 0, -1
;maskleft: [a,b,c,d] -> [0,b,0,d]
maskRightt: DD -1, 0, -1, 0
```

```
mover entre registros:
movdqa (align)
movdqu (unalign)
```

```
paddd
packed add doubleword
paddw
paddb
psubb
psubd
psubq
psubb

ejemplo:
xmm0[1,2,3,4,5,6,7,8]
xmm1[9,10,11,12,13,14,15,16]
phaddw xmm1, xmm2
phaddw xmm1, xmm1
xmm1=[9+10+11+12, 13+14+15+16,
1+2+3+4, 5+6+7+8, repetido basura]
punpckhwd desempacketa high words to double.
punpcklwd desempaqueta low words to double.
[a7,a6,a5,a4] a era de 0 a 7

pslldq xmm2, 4
; xmm2 = [ P2 | P1 | P0 | 0 ]

pextrd r15d, xmm3, 1
; r15 = P1

pinsrd xmm3, r15d, 3

; xmm3= [ P1 | P2 | P1 | P0 ]
; xmm2 = [ P2 | P1 | P0 | 0 ]

pavgb xmm3, xmm2;
;xmm3 = [(P1 + P2)/2|X|(P1 + P0)/2|X]

pand xmm3, xmm15;
;xmm3=[(P1 + P2)/2|0|(P1 + P0)/2|0]

por xmm1, xmm3
;xmm1=[(P1+P2)/2|P1|(P1+P0)/2|P0]

cvtdq2ps xmm0, xmm0 ;[r, g, b.0] en float

mulps xmm0, xmm11
;xmm0=[var*maxr, var*maxg, ..]

cvtps2dq xmm0, xmm0 ; en entero

packssdw xmm0, xmm9 ; xmm9 = [0 ,0 ,0]

packuswb xmm0, xmm9
; [0 ,0 ,0 ,...R,G,B,0 ]
; xmm9 = [0 ,0 ,0]
```

Listing 4. preservar pila

```
global fun
section .text
fun:
  push rbp
  mov rbp, rsp
  sub rsp, 24
  push rbx
  push r12
  push r13
  push r14
  push r15
  ;....
  pop r15
  pop r14
  pop r13
  pop r12
  pop rbx
  add rsp, 24
  pop rbp
  ret
```

Listing 5. moviendo desde memoria

```
mascara: DQ 0xF2, 0xF9
movdqu xmm1, [mascara]
;xmm1 = {15}[0Xf9 | 0xf2]{0}
```

Listing 6. Hola Mundo

```
section .data
msg: DB Hola Mundo , 10
largo EQU $ msg
global _start
section .text
_start :
  mov rax , 4 ;sys write
  mov rbx , 1 ;fd stdout
  mov rcx , msg ;buf
  mov rdx , largo ; count
  int 0x80
  mov rax , 1 ;exit
  mov rbx , 0 ;error code=0
  int 0x80
```

Listing 7. String Copy

```
extern malloc
;funcion externa
;char* str copy(char* a)
;(27 lineas)--(19)
  push rbp
  mov rbp, rsp
  push rdi

  mov al, 0
  mov rcx, -1
  cld;clear direction
  repne scasb
  ;string compare, repne to al

  neg rcx
```

```
  dec rcx
  ;la respuesta tiene 1 menos y negada
  push rcx

  mov rdi, rcx
  call malloc

  pop rcx
  pop rsi
  cld

  mov rdi, rax
  ;rdi tiene destino, rsi fuente
  ;y rcx contador
  rep movsb
  pop rbp
ret
```

Listing 8. Uso de SHUFFLES y otras yerbas

```
section .rodata
val0703: dq 0.7, 0.3
val255: dq 255.0, 255.0

section .text
ej1: ; rdi = a, esi = n;
  push rbp
  mov rbp,rsp
  mov ecx, esi
  shr ecx, 2
  movdqu xmm8, [val0703]
  movdqu xmm9, [val255]
  .ciclo:
    movdqu xmm0, [rdi]
    ; xmm0 = [fp3|fp2|fp1|fp0]
    cvtPS2PD xmm1, xmm0
    ; xmm1= [fp1|fp0]
    psrldq xmm0, 8
    ; xmm0 = [0|0|fp3|fp2]
    cvtPS2PD xmm2, xmm0
    ; xmm2= [fp3|fp2]
    mulpd xmm1, xmm8
    ;  xmm1= [.3*fp1|.7*fp0]
    mulpd xmm2, xmm8
    ; xmm2= [.3*fp3|.7*fp2]


    movdqu xmm3, xmm1
    ; xmm3 = [.3*fp1|.7*fp0]
    shufpd xmm3, xmm2, 1
    ; xmm3 = [.7*fp2|.3*fp1]
    shufpd xmm1, xmm2, 2
    ; xmm1 = [.3*fp3|.7*fp0]
    addpd xmm1, xmm3
    ;xmm1 = [.3*fp3+.7*fp2|.3*fp1+.7*fp0]

    sqrtpd xmm1, xmm1
    ;xmm1 = [sqrt(.3*fp3+.7*fp2)
    ;        |sqrt(.3*fp1+.7*fp0)]
    mulpd xmm1, xmm9
    ;xmm1 = [255*sqrt(.3*fp3+.7*fp2)
    ; |255*sqrt(.3*fp1+.7*fp0)]
    movdqu [rdi], xmm1
    add rdi, 16
    loop .ciclo
  pop rbp
ret
```

Listing 9. Tree sample

```
//binary tree:
//       4
//   2       8
//1   3   6   9
void insertarNodo( nodo* N, char* valor ){
  nodo* actual = N;
  bool termine = false;
  while(!termine)
    if(cmpStr(valor, actual->val) < 0){
      if( actual->hijoIzquierdo == NULL )
    actual->hijoIzquierdo=nodo_crear(valor);
        termine = true;
      else
        actual = actual->hijoIzquierdo;
    }
    else {
      if( actuall->hijoDerecho == NULL )
        actual->hijoDerecho = nodo_crear( valor
        termine = true;
      else
        actual = actual->hijoDerecho;
    }
}
```

Listing 10. Tree en ASM

```
extern malloc
global node_insert, node_create, string_compare

;struct node{
; char* val
; nodo* leftChild
; nodo* rightChild
;}

OFFSET_VAL equ 0
OFFSET_RIGHTCHILD equ 16
OFFSET_LEFTCHILD equ 8
NULL equ 0

SIZEOF_STRUCT equ 24

;void node_insert(node* root,
char* val)
```

```
;RDI: puntero a raiz,
;RSI: puntero a palabra a insertar

section .text

node_insert:
 PUSH RBP    ;alineada
 MOV  RBP,RSP
 PUSH R14    ;desalineada
 PUSH R15    ;alineada
 PUSH R13    ;desalineada
 SUB  RSP, 8

 MOV  R15, RDI ;R15 <- puntero actual
 MOV  R14, RSI ;R14 <- puntero a palabra
 MOV  R13, NULL ;puntero a siguiente

.ciclo:
 CMP R15, NULL
 JE .agregar_y_terminar

 MOV RDI, R14
 MOV RSI, [R15 + OFFSET_VAL]
 call string_compare
 CMP qword RAX, -1
 JE .menor
 LEA R13, [R15 + OFFSET_RIGHTCHILD]
 JE .seguir

.menor:
 LEA R13, [R15 + OFFSET_LEFTCHILD]

.seguir:
 MOV R15, [R13]
 JMP .ciclo

.agregar_y_terminar:
 MOV RDI, R14
 call node_create
 MOV [R13], RAX

.terminar:
 ADD RSP, 8
 POP R13
 POP R15
 POP R14
 POP RBP

node_create:
 PUSH RBP;
 MOV RBP, RSP
 PUSH RDI
    SUB RSP, 8

 MOV RDI, SIZEOF_STRUCT
 call malloc
 ADD RSP, 8
 POP RDI

 MOV qword[RAX + OFFSET_RIGHTCHILD], NULL
 MOV qword[RAX + OFFSET_LEFTCHILD], NULL
 MOV qword[RAX + OFFSET_VAL], RDI
 POP RBP
 RET

string_compare:
 PUSH RBP
 MOV RBP, RSP

 .ciclo:
 MOV DL, [RDI]
 MOV CL, [RSI]

 CMP DL, 0
 JE .devuelvePositivo
 CMP CL, 0
 JE .devuelveNegativo

 CMP DL, CL
 JNE .seguir

 INC RDI
 INC RSI
 JMP .ciclo

.seguir:
 CMP DL, CL
 JL .devuelvePositivo
 JMP .devuelveNegativo

.devuelvePositivo:
 MOV RAX, 1
 JMP .terminarStrings

.devuelveNegativo:
 MOV RAX, -1

.terminarStrings:
 POP RBP
 RET
```

Listing 11. DOUBLE LINK

```
global ordenarCadena
global ordenarCadenaRecu
extern free
extern malloc

%define NULL 0
%define OFFSET_BASE 0
%define OFFSET_PAREJA 1
%define OFFSET_INFERIOR 9
%define ESLABON_SIZE 17
```

```
ordenarCadena:
;eslabon* ordenarCadena(eslabon* p,
;enum action_e (*cmp_base)
;(char* base1, char* base2))

push rbp
mov rbp, rsp
sub rsp, 40
push rbx
push r12
push r13
push r14
push r15

    cmp rdi, NULL
    je .fin

    ;Guardo el puntero a funcion
    mov r15, rsi

    mov [rbp -8], rdi
    lea rbx, [rbp - 8]

    mov r12, rdi
    mov r14, [r12 + OFFSET_PAREJA]
    mov [rbp - 16], r14
    lea r13, r14

.sig:
    cmp r12, NULL
    je .fin

    lea rdi, [r12 + OFFSET_BASE]
    lea rsi, [r14 + OFFSET_BASE]
    call r15

    cmp eax, 0
    jnz .duplicar

    mov rax, [r12 + OFFSET_INFERIOR]
    mov [rbx], rax

    mov rax, [r14 + OFFSET_INFERIOR]
    mov [r13], rax

    mov rdi, r12
    call free

    mov r12, [rbx]
    mov r14, [r13]

    jmp .sig

 .duplicar:

    mov rdi, ESLABON_SIZE
    call malloc

    ;Actualizo el inferior del lugar de donde v
    ;Si en la primera iteracion del ciclo
    ;se duplica entonces el nodo duplicado es e
    mov [rbx], rax

    ;Copio la base
    mov r8, [r12 + OFFSET_BASE]
    mov [rax + OFFSET_BASE], r8

    ;Lo copio arriba del nodo actual
    mov [rax + OFFSET_INFERIOR], r12

    ;Me quede sin registros para guardar :(
    mov [rbp - 24], rax

    ;Creo el eslabon pareja
    mov rdi, ESLABON_SIZE
    call malloc

    mov [r13], rax

    mov r8, [r14 + OFFSET_BASE]

    mov [rax + OFFSET_BASE], r8

    mov [rax + OFFSET_INFERIOR], r14

    ;Recupero al primer nodo creado
    mov r9, [rbp - 24]

    ;Conecto las parejas
    mov [r9 + OFFSET_PAREJA], rax
    mov [rax + OFFSET_PAREJA], r9

    ;Actualizo los doble punteros

    lea rbx, [r12 + OFFSET_INFERIOR]
    mov r12, [r12 + OFFSET_INFERIOR]

    lea r13, [r14 + OFFSET_INFERIOR]
    mov r14, [r14 + OFFSET_INFERIOR]

    jmp .sig

.fin:
mov rax, [rbp-8]

pop r15
pop r14
pop r13
pop r12
pop rbx
add rsp, 40
pop rbp

ret
```

Listing 12. ejercicio trie

```
global check_trie

%define NULL 0

%define NODO_SIGUIENTE 0
%define NODO_HIJOS 8
%define NODO_LETRA 16
%define NODO_FIN 17
%define NODO_SIZE 18

;La implementacin devuelve
;el primer nodo desordenado
;recorriendo estilo DFS

check_trie:
;nodo** check_trie(trie* t)

push rbp
mov rbp, rsp
sub rsp, 8
push rbx
push r12
push r13

    ;Me fijo si me pasaron NULL
    cmp rdi, NULL
    je .fin

    ;Trie vac o est  ordenado
    mov r12, [rdi]; r8 = raiz*
    cmp r12, NULL
    je .ordenado

    ;Trie* es igual a **nodo
    mov rbx, rdi

;Itero por los nodos de cada
;nivel, y para cada uno hago un
;llamado recursivo para su hijo.
.ciclo:

    mov r8, [r12 + NODO_HIJOS]

    cmp r8, NULL
    je .avanzo_nivel
    ; Si no tiene hijo
    ;avanzo en el nivel

    ;Si existe hago el llamado
    ;recursivo
    ;Creo un doble puntero
    ;que hace de trie*
    ;Mi nueva "raiz" es el
    ;hijo del nodo actual
    mov [rbp - 8], r8
    lea rdi, [rbp - 8]
    call check_trie

    ;Me fijo si me llego el
    ;resultado de mi llamado recursivo
    cmp rax, NULL
    jne .fin

.avanzo_nivel:
    mov r13, [r12 + NODO_SIGUIENTE]

    cmp r13, NULL

    ;Si llegue al final del nivel
    ;entonces todo lo anterior
    ;estaba ordenado
    je .ordenado

    ;No llegue al final, veo si el
    ;nodo actual es el desordenado
    xor rdi, rdi
    mov rdi, [r12 + NODO_LETRA]

    xor rsi, rsi
    mov rsi, [r13 + NODO_LETRA]

    cmp rdi, rsi
    jg .no_ordenado

    ;Avanzo en el nivel e itero de nuevo
    mov r12, r13
    jmp .ciclo

.no_ordenado:
    mov rax, rbx
    jmp .fin

.ordenado:
    mov rax, NULL

.fin:

pop r13
pop r12
pop rbx
add rsp, 8
pop rbp
ret
```