Arquitectura de Computadoras

Tips y Errores Comunes

Guia de referencia rapida para el parcial
11 de October de 2025

---

1 Tips Generales

Tip: Todas las posiciones de memoria de Intel ocupan 1 Byte. Esto es importante para entender el direccionamiento y el uso de registros.

Tip: La estandarizacion de espacio busca eficiencia de acceso a costa de espacio en memoria.

Tip: Acceso a letra dentro de un string: usar registros de 4 bits (AL, BL, etc). Esto hace que no necesitemos agregar explicitamente la keyword BYTE ya que el compilador sabe el tamaño de AL, BL, etc

2 Assembler x86

2.1 Syscalls

Error Comun: No olvidarse del `int 80h` al hacer syscalls. Es muy facil olvidarlo y el programa no va a funcionar

```
;; Syscall correcta
mov eax, 1    ; sys_exit
mov ebx, 0    ; codigo de retorno
int 80h       ; NO OLVIDAR ESTO
```

Tip: Usar `strace` para interceptar y debuggear las system calls de un programa:

```
strace ./programa
```

Importante: Hacer la syscall de exit permite no tener que hacer armado y desarmado de stack frame en la funcion `_start`.

2.2 Size de Operandos

Error Comun: Cuando se hace un `mov` con punteros, SI o SI especificar la cantidad de bytes:

```
mov byte [esi], 0   ; BIEN -> especificar 1 byte
mov [esi], 0        ; MAL -> El compilador no sabe cuantos bytes tiene que copiar
; Le surge la duda de "Puntero a que?". Aclarando con la keyword, se elimina esa
; ambiguedad
```

Nota: Las opciones de size son:

- `byte` → 1 byte
- `word` → 2 bytes
- `dword` → 4 bytes
- `qword` → 8 bytes

2.3 Registros

Importante: Registros que se deben preservar (caller-saved):

- `ebx`
- `esi`
- `edi`
- `ebp`
- `esp`

Notemos que si nos dicen que preservemos estos registros, quiere decir que todas las funciones que llames desde assembly de libc, van a preservar estos registoros.

Tip: Si el sistema necesita optimizaciones, se puede evitar el backup de registros, pero esto depende de cada caso particular. Y esta abierte la eleccion para que el programador elija

Tip: Para copiar una direccion de memoria con un offset, podemos usar la instruccion `lea`, que tambien **es importante saberla** ya que es comun que la pongan en ejemplos de assembler. *Ambas opciones a continuacion son equivalentes*

```
mov eax, 0x6c6c6c6c ; Supongamos que es una direccion valida
mov esi, eax        ; Movemos la direccion a esi (source index)
add esi, 4          ; Es comun hacer esto cuando queremos recorrer un array
;; ====== Seria exactamente igual hacer ======
lea esi, [eax + 4]  ; Notemos que es mas corto y practico
```

Tip: Para recorrer un string (leer byte a byte con AL) e ir aumentando ESI, es posible hacerlo en una sola instruccion:

```
; Leer byte a byte de [ESI] e ir avanzando
cld
lodsb               ; AL = [ESI], ESI++
; ... procesar AL ...
```

3 Stack y Stack Frames

3.1 Armado y Desarmado

Importante: Siempre armar el stack frame al inicio de una funcion y desarmarlo antes de retornar:

Armado:

```
push ebp
mov ebp, esp
```

Desarmado:

```
mov esp, ebp
pop ebp

; Otra opcion equivalente:
leave
```

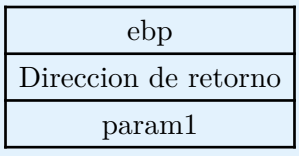
Tip: Hacer el seguimiento de la pila y **no borrar lo que ya se escribio**

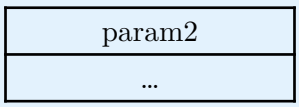
3.2 Acceso a Parametros

Nota: En x86 (32 bits), los parametros se pasan por la pila. Para acceder al primer parametro (**en caso de haber hecho armado de stack frame!!!!!!!**):

```
mov ax, [ebp + 8]
```

Pues tenemos:





Tip: En arquitectura de 64 bits, los parametros se pasan primero por registros y luego por la pila. Esto ayuda a evitar inyeccion de codigo.

3.3 Valores de Retorno

Importante: Convencion de retorno en C:

- Menor a 32 bits: retorna en `EAX`
- Mayor a 32 bits: parte alta en `EDX`, parte baja en `EAX`
- Estructuras complejas: retorna un puntero formado por `EDX:EAX`

3.4 Errores Comunes

Error Comun: Si una funcion altera flags (como el flag Z), puede romper ciclos que dependen de ese flag. :

```
dec ecx
call procesar    ; si procesar altera el flag Z
jz fin           ; este salto puede no funcionar correctamente
```

La solucion sencilla es que **todas las funciones se encarguen de dejar el estado del procesador como se recibio**, es decir backupear las flags

Error Comun: La informacion queda intacta despues de hacer `POP`. Esto puede ser un problema de seguridad (analisis forense de memoria).

4 Memoria

4.1 Conceptos clave

Tip: Formula para calcular **espacio de memoria**:

$$2^{\text{bits bus address}} \times \text{bytes bus de datos}$$

Ejemplo: 32 bits address, 32 bits datos $2^{32} \times 4$ bytes $= 16\text{GB}$

Importante: Las «patitas» del bus de address estan directamente relacionadas a los punteros que usa el procesador. Un bus de 32 bits necesita punteros de 32 bits.

Nota: Tipos de memoria:
- RAM (volatil, rapida, se pierde al apagar)
- ROM (no volatil, lenta, persiste al apagar)
- SRAM (rapida, costosa, usada en cache, no necesita refresco)
- DRAM (mas lenta, economica, necesita refresco)

4.2 Mapeo de Memoria

Tip: Ventajas del mapeo en memoria:
- Mayor cantidad de instrucciones disponibles
- Modificacion directa de registros del periferico sin usar `IN` / `OUT`

Desventajas:
- Reduce cantidad de memoria disponible (impactominimo hoy en dia)

5 Paginacion y Memoria Virtual

5.1 Conceptos Clave

Importante: La paginacion divide la memoria en tamanos fijos (tipicamente 4KB). Esto reduce la fragmentacion de memoria.

Nota: Las paginas siempre empiezan en multiplos de tamano de pagina (esto quiere decir que estan alineadas).

Importante: **Atributos de tablas**
- Las tablas de directiorios tienen atributos, si una tabla de directorio tiene el atributo de presente en off, quiere decir que la tabla de pagina a la que te lleva, no esta presente, entonces no es necesario apagar todos los otros Ps
- Las tablas de paginas tambien tienen los mismos atributos

5.2 Ejercicios de Paginacion

Tip: *Todos los ejercicios de paginacion siguen el mismo patron:*
- Mapa virtual: su tamano depende del bus de address del procesador
- Mapa fisico: su tamano depende de la cantidad de RAM
- Registro CR3: contiene la **direccion fisica** de la **tabla de directorio**
- Bit P (presente): indica si esta en RAM o en disco

Importante: En sistemas de 64 bits con 2MB de tamano de pagina:

- Paginas alineadas a 2MB necesitan 21 bits para offset
- «Tamano de pagina - 1» debe ser el valor maximo obtenido con todos los bits del offset en 1

Error Comun: No poner CR3 (posicion del directorio) en una zona que pueda ser sobrescrita.

6 Seguridad

6.1 Stack Overflow

Error Comun: Nunca usar `gets()`. Esta funcion permite escribir en memoria sin limite, permitiendo ataques de Stack Overflow donde un atacante puede inyectar codigo malicioso.

Tip: El canary de alguna forma reduce el impacto de este problema, interponiendose entre las variables locales (**en caso de haber buffers**) y el ebp/retorno. El esquema seria el siguiente:

Buffer
canary
ebp
Direccion de retorno
param1
param2
...

En caso de que se exceda del limite del buffer, ejemplo:

```
char buff[3] = {0};
buff[4]; // ME EXCEDO DEL LIMITE Y PISO EL CANARY
```

En vez de pisar el **ebp** o la **direccion de retorno** (MUY PELIGROSO POR POSIBLE RCE), pisas el canary. Despues al final de la funcion, antes de popear el ebp y el retorno, te fijas si el canary cambio, en caso de que se hayan excedido del buffer, el canary cambia, por lo que se arroja un error

Importante: El canary debe estar explicito en el codigo ASM si se quiere usar como proteccion. En el caso de que nos pasen un codigo de C que contenga un buffer, **el compilador agrega el canary a menos de que le pidas explicitamente que no lo haga**, esto quiere decir que si piden el seguimiento de pila de un codigo en C y contiene un buffer, **si o si va a estar el canario perez**

Nota: Concepto de inyectar codigo:
- Sigue pasando en otros lenguajes

- Si un programa se puede romper con un error, esta mal programado
- Para encontrar direcciones especificas se usan scripts

Nota: *Como se ve el Canary en codigo assembler?*

- En **64 bits**:

```
  ;; ... (Armado de stackframe incluido)
  mov rax, QWORD PTR fs:40    ; Guardo copia del canary a rax

  mov QWORD PTR -8[rbp], rax  ; Pusheo a partir del rbp (justo encima como
                              ; estamos en 64 bits, cada dir es de 8 Bytes)
  ;; ...
  mov rdx, QWORD PTR -8[rbp]  ; Leo la copia del canary
  sub rdx, QWORD PTR fs:40    ; Hago la resta entre el canary original y lo
                              ; que me quedo en la copia del canary
  je  .L7                     ; En caso de que hayan sido iguales, no llamo a error
  call  __stack_chk_fail@PLT  ; Si no salto, se llama a la funcion de error
  .L7
  ;; ...
```

- En **32 bits**:

```
  mov     eax, DWORD PTR __stack_chk_guard  ; Cargar el Canary
  mov     DWORD PTR [ebp - 4], eax          ; Guardar el canary al final del stack

  ; ... -> (funciones sin limite, en las que puede haber OF)

  ; luego de la funcion sin limite, checkeamos el valor del canary para ver si se
piso o podemos retornar correctamente
  mov     eax, DWORD PTR __stack_chk_guard  ; valor original del canary
  cmp     eax, DWORD PTR [ebp - 4]          ; comparamos el original con lo que
tenemos en el stack
  jne     .stack_smash_detected

.stack_smash_detected:
  call    __stack_chk_fail                  ; abortamos el programa
```

7 Interrupciones

7.1 Conceptos

Tip: Polling vs Interrupciones:

- Polling: consulta constantemente «tenes un dato?»

- Interrupciones: «cuando tengas un dato me avisas» (mas eficiente)

Importante: IDT (Interrupt Descriptor Table): tabla que contiene punteros a todas las rutinas de atencion de interrupcion. Las primeras 32 son excepciones, solo 20 estan en uso actualmente.

Nota: El flag IF controla si se atienden interrupciones externas:
- IF = 1: habilitado (`sti` - set interrupts)
- IF = 0: deshabilitado (`cli` - clear interrupts)

7.2 Tipos de Interrupciones

Importante:
- NMI (Non-Maskable Interrupt): no pueden ser enmascaradas, son criticas
- IMR (Interrupt Mask Register): interrupciones normales que pueden ser enmascaradas
- Ejemplo: teclado y mouse son interrupciones normales

7.3 Excepciones

Nota: Tipos de excepciones:
1. Faults: pueden corregirse, se guarda la direccion de la instruccion
2. Trap: se usan para acceder al sistema operativo
3. Abort: errores severos, no siempre se puede obtener la instruccion

Importante: Excepciones comunes:
- Divide error
- Invalid opcode
- Stack exception
- General protection (la pantalla azul)
- Page fault

8 Modo Protegido y MMU

Importante: La MMU (Memory Management Unit) es un metodo de interposicion que checkea cada acceso a memoria.

Flujo: Direcciones logicas → Unidad de segmentacion → Direccion Lineal → Unidad de paginacion → Direccion fisica → Memoria fisica

Tip: Memoria virtual: el procesador siempre da memoria, nunca dice que no hay mas. Es una abstraccion sobre la memoria fisica. Cuando accedes a 2000h, probablemente estes accediendo fisicamente a otro lado.

9 Decodificacion

Importante: El pin IO/Mem del microprocesador (**Solo esta en procesadores Intel**):

- Selecciona si escribir en memoria o en el mapa de I/O
- Tiene corriente (True) si hay que escribir en el mapa de I/O

Importante: El pin R/W indica si se esta leyendo o escribiendo.

- El pin de **READ** va enchufado tanto en la **RAM** como en la **ROM**
- El pin de **WRITE** va enchufado solo en la **RAM**, si lo enchufas en la ROM es incorrecto (en parciales bajan puntos)

Nota: Las tablas estan mapeadas en memoria virtual pero no es necesario ponerlas porque lo hace el kernel (Incluso si me piden que meta todo lo que esta en el mapa virtual, implicitamente puedo suponer que me hablan de user space)

Tip: Se pueden dejar entradas de decodificadores sin enchufar, pero puede haber ruido. Solucion: conectar las entradas a ground (0).

9.1 Intel vs Genericos

Nota: Diferencias entre procesadores Intel y genericos:

Intel:

- Tienen parte baja y alta en registros
- Siempre dividen la memoria en bytes

Genericos:

- No tienen «parte baja y alta»
- Siempre tratan todo como tamano de bus. Esto quiere decir que cada «renglon» es una direccion de memoria independientemente del tamaño del bus

10 Optimizaciones y Performance

Tip: Para determinar si multiples syscalls vs una sola syscall es mas eficiente, hacer profiling en ambos casos. La diferencia generalmente no es notable.

Tip: La convencion de backup de registros depende de cada caso. Si el sistema necesita optimizaciones, es preferible evitarlo.

Importante: Cuando se hace `sub esp, 32`, se estan restando 32 bytes. Siempre usar parametros de 4 bytes en programacion de 32 bits.

11 Herramientas Utiles

11.1 Links de Referencia

- `strace`: herramienta para interceptar system calls
- Godbolt (godbolt.org): pasaje en vivo de C a ASM
- NASM docs: directivas para reservar memoria y declarar datos
- Arquitectura PC Intel: stanislavs.org/helppc/

11.2 Comandos Importantes

Tip: Para ver syscalls en Linux:

```
strace ./programa
```

Tip: Verificar compilacion de C a ASM en tiempo real usando Godbolt para entender como el compilador traduce tu codigo.

12 Notas Finales

Importante: Practica haciendo seguimiento de la pila en papel. Esto ayuda a visualizar el estado de la memoria y evitar errores comunes.

Tip: Si hay dudas sobre como el compilador traduce codigo C a ASM, usar Godbolt