UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Departamento de
Ingeniería Electrónica

Técnicas Digitales III

Guía de Trabajos Prácticos

Primer cuatrimestre

Versión 1.1

Ciclo lectivo 2022

Plan 95A

Tabla de contenido

La presente guía puede ser resuelta utilizando el lenguaje que considere más adecuado. La cátedra recomienda el uso de ensamblador (NASM) y C (C11).

1. Interacción con las herramientas

Instalar la máquina virtual ***Bochs*** siguiendo las instrucciones indicadas en **https://sge.frba.utn.edu.ar/wiki/td3/doku.php?id=bochs**
Configurar la máquina virtual para simular una PC con 512MB de memoria RAM y 64kB de ROM y un procesador x86 genérico.

Objetivos conceptuales

I. *Familiarizarse con los comandos de **Bochs**.*
II. *Entender el funcionamiento de la imagen de la máquina virtual y su generación*
III. *Familiarizarse con la estructura de los archivos Makefile y de configuración de Bochs*

2. Inicialización básica utilizando solo ensamblador con acceso a 1MB

Escribir un programa que se ejecute en una ROM de 64kB y permita copiarse a sí mismo en cualquier zona de memoria. A tal fin se deberá implementar la función

void *td3_memcopy(void *destino, const void *origen, unsigned int num_bytes);

Para validar el correcto funcionamiento del programa, el mismo deberá copiarse en las direcciones indicadas a continuación y mediante Bochs verificar que la memoria se haya escrito correctamente.

i. 0x00000
ii. 0xF0000

Objetivos conceptuales

i. *Familiarizarse con el lenguaje ASM y las herramientas asociadas (NASM).*
ii. *Familiarizarse con las herramientas de depuración provistas por el Bochs.*
iii. *Comprender el mapa de memoria del procesador.*
iv. *Identificar todos los registros y datos que utiliza el procesador para acceder a cada instrucción de código y dato. [1][2]*

Referencias

[1] Volumen 1, Capítulo 3, sección 3, Intel® 64 and IA-32 Architectures Software Developer Manuals,
[2] Volumen 1, Capítulo 3, sección 4, Intel® 64 and IA-32 Architectures Software Developer Manuals,

3. INICIALIZACIÓN BÁSICA UTILIZANDO EL LINKER

Modificar el código del ejercicio anterior para satisfacer los siguientes requerimientos:

a. El programa debe situarse al inicio de la ROM (0xFFFF0000)
   i. Copiarse y ejecutarse a la dirección 0x00007C00 donde debe establecer estado de ***halted*** en forma permanente.

El mapa de memoria propuesto

Sección	Dirección inicial
Binario copiado	00007C00h
Pila	00050000h
Secuencia inicialización ROM	FFFF0000h
Vector de reset	FFFFFFF0h

Tabla 1. Mapa de memoria ejercicio 3

Objetivos conceptuales

I. *Familiarizarse con el lenguaje ASM y las herramientas asociadas (NASM).*
II. *Familiarizarse con las herramientas de depuración provistas por el Bochs.*
III. *Familiarizarse con el "linker", su lenguaje de "script" y las herramientas asociadas (ld).*
IV. *Identificar todos los registros y datos que utiliza el procesador para acceder a cada instrucción de código, dato y pila. [1][2]*

Condiciones generales

A partir de este punto de la guía se recomienda fuertemente plantear el ejercicio con el siguiente esquema de archivos.

Makefile o ***make.sh*** : comandos necesarios para construir el binario
linker.lds : *script* para el *linker*
bochs.cfg : configuración utilizada para el Bochs en cada ejercicio
init.s : solo el código necesario para inicializar al procesador en modo protegido y en
ejercicios posteriores la paginación.
main.s : funcionalidad solicitada en cada ejercicio
functions.s : funciones auxiliares y/o frecuentemente implementadas en ensamblador
functions.c : funciones auxiliares y/o frecuentemente implementadas en C
sys_tables.s : tablas de sistema.

4. MODO PROTEGIDO 32BITS

En base al ejercicio anterior, adecuarlo para que el mismo se ejecute en modo protegido 32bits.

a) Crear una estructura GDT mínima con modelo de segmentación FLAT [3][4][5]
b) En la zona denominada Kernel solo debe copiarse código.
c) Definir una pila dentro del segmento de datos e inicializar el par de registros **SS:ESP** [3][4] adecuadamente. Realice la definición de forma dinámica de modo que pueda modificarse su tamaño y ubicación de manera simple.

El mapa de memoria para las diferentes secciones debe ser el siguiente

Sección	Dirección inicial
Rutinas	00037000h
Kernel	00137000h
Pila	1FFF0000h
Secuencia inicialización ROM	FFFF0000h
Vector de reset	FFFFFFF0h

Tabla 2. Mapa de memoria ejercicio 4

Objetivos conceptuales

I. *Comprender los requerimientos necesarios para que un programa pueda ejecutarse en modo protegido.*
II. *Identificar todos los registros y datos que utiliza el procesador para acceder a cada instrucción de código, dato y pila.*
III. *Analizar el esquema de direccionamiento entre modo protegido y real, identificando diferencias y similitudes.*
IV. *Comprender el significado y la implicancia de cada campo de las tablas de descriptores y los mecanismos de protección que activan.*

Referencias

[3] Volumen 3, Capítulo 2, sección 2, Intel® 64 and IA-32 Architectures Software Developer Manuals
[4] Volumen 3, Capítulo 3, sección 2, Intel® 64 and IA-32 Architectures Software Developer Manuals,
[5] Volumen 3, Capítulo 3, sección 4, Intel® 64 and IA-32 Architectures Software Developer Manuals,

5. CONFIGURACIÓN DEL SISTEMA DE INTERRUPCIONES

Tomando el ejercicio anterior, agregar una IDT [6][7] capaz de manejar todas las excepciones del procesador e interrupciones mapeadas por ambos PIC, es decir de la 0x00 hasta el tipo 0x2F y cumpla con los siguientes requerimientos.

a) Configurar el PIC maestro y esclavo de manera que utilicen el rango de tipos de interrupción **0x20-0x27** y **0x28-0x2F**, respectivamente.
b) Inicializar el registro de máscaras [8], de modo que estén deshabilitadas, todas las interrupciones de hardware en ambos PIC's.
c) Implementar en todas las excepciones una rutina que permita identificar el número de excepción generada y finalice deteniendo la ejecución de instrucciones mediante la instrucción "**hlt**". Se propone como método de identificación almacenar en ***dl*** el número de excepción.
d) Generar de **manera apropiada** [9] (no emulándolas por interrupciones de software) para comprobar su funcionamiento las excepciones #UD, #DF, #SS y #AC. Se recomienda asociar la generación de cada una de las excepciones indicadas previamente a la pulsación de diferentes teclas. A tal fin se propone la siguiente correspondencia: #UD=**U**, #DF=**I**, #SS=**S**, y #AC=**A**.
e) La IDT se debe ubicar en *Tablas de sistema*

El mapa de memoria debe ser el siguiente:

Sección	Dirección inicial
Tablas de sistema	00000000h
Rutina de teclado e ISR	00157000h
Datos	00127000h
Kernel	00137000h
Pila	1FFF0000h
Secuencia inicialización ROM	FFFF0000h
Vector de reset	FFFFFFF0h

Tabla 3 Mapa de memoria ejercicio 5

Objetivos conceptuales

I. *Comprender el esquema de numeración de los vectores de interrupción y su asociación con la decodificación realizada por los PIC.*
II. *Entender la diferencia entre IRQ y tipo de interrupción.*
III. *Identificar los tipos de excepciones y las condiciones que las generan.*
IV. *Comprender el significado y la implicancia de cada campo de las tablas de descriptores y los mecanismos de protección que activan.*
V. *Analizar la gestión de la pila realizada por cada excepción.*
VI. *Identificar todos los registros y datos que utiliza el procesador para acceder al controlador de una excepción.*
VII. *Identificar las diferencias principales entre una excepción y una interrupción*

Referencias

[6] Volumen 3, Capítulo 6, sección 10, Intel® 64 and IA-32 Architectures Software Developer Manuals,
[7] Volumen 3, Capítulo 6, sección 11, Intel® 64 and IA-32 Architectures Software Developer Manuals,
[8] Volumen 3, Capítulo 2, sección 3, Intel® 64 and IA-32 Architectures Software Developer Manuals,
[9] Volumen 3, Capítulo 6, sección 15, Intel® 64 and IA-32 Architectures Software Developer Manuals.

6. INTERRUPCIONES DE HARDWARE

Utilizando lo realizado anteriormente, implementar las siguientes modificaciones:

a) Incluir una rutina de adquisición de teclas. Esta debe realizarse en el controlador de teclado (**IRQ1** [10], dirección de E/S 0x60 datos y 0x64 estado/comando). Se debe tener en cuenta que por cada presión de una tecla se producen dos interrupciones, una por el ***make code*** y otra por el ***break code.***

b) Implementar la lectura de la **IRQ1**, mediante un buffer circular cuyas direcciones y funcionamiento s muestra en la Figura 1.

c) Solo se tomarán como válidos dígitos numéricos y el retorno de carro. Las teclas restantes se descartarán.

d) El handler de **IRQ1** conformará con los sucesivos caracteres un número de hasta 64bits.

   En caso de ingresarse una cantidad menor de 16 teclas numéricas consecutivas, completará el número con ceros a la izquierda, es decir si se presionan las teclas 12345678, se debe almacenar en la tabla de dígitos como una entrada que contiene al número 0000000012345678h. Cada nuevo número se insertará en la Tabla de datos cuando se presione ENTER. Por razones de simplicidad el buffer circular de teclado dispondrá de una longitud de 17 bytes. En caso de no haber recibido ENTER hasta el caracter Nº 16, al ingresarse el caracter Nº. 17 el handler lo reemplaza por ENTER y almacena el número de 16 dígitos en la posición correspondiente de Tabla.

e) Escribir el controlador del temporizador (**IRQ0** [10]) de modo que interrumpa cada 100ms. Verifique el correcto funcionamiento almacenando en alguna dirección de *Datos* el número de veces que se produce la interrupción. Tenga en cuenta que la implementación del *timer tick* en ***Bochs*** no garantiza ejecución del tipo tiempo real, es decir observará una falta de correspondencia temporal entre la unidad de tiempo calculada y la que ***Bochs*** ejecuta en la práctica

El mapa de memoria es el que se muestra en la Tabla 4.

Sección	Dirección inicial
Tablas de sistema	00000000h
Rutina de teclado e ISR	00157000h
Tabla de Dígitos	00100000h
Datos	00127000h
Kernel	00137000h
Pila	1FFF0000h
Secuencia inicialización ROM	FFFF0000h
Vector de reset	FFFFFFF0h

Tabla 4. Mapa de memoria ejercicio 7

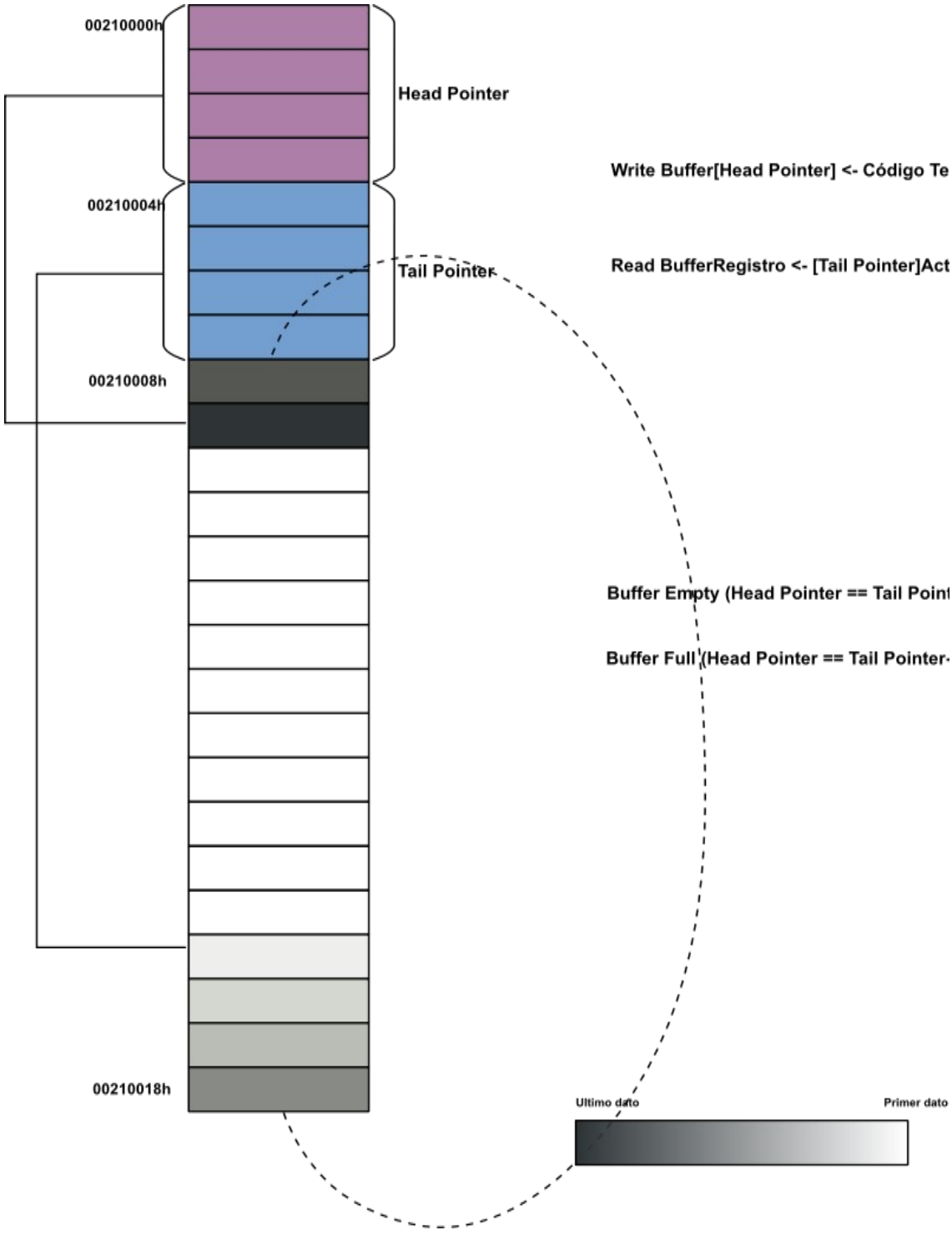

Figura 1. Buffer circular de teclado

Objetivos conceptuales

I. *Identificar todos los registros y datos que utiliza el procesador para acceder al controlador de una interrupción.*
II. *Comprender la implicancia del término "enmascarable"*
III. *Analizar la gestión de la pila realizada por una interrupción y compararla con la de una excepción.*

Referencias

[10] Volumen 3, Capítulo 6, sección 12, Intel® 64 and IA-32 Architectures Software Developer Manuals.

7. RUTINA TEMPORIZADA Y CONTROLADOR DE VIDEO

Modificar el programa desarrollado hasta el momento considerando las siguientes consignas:

a) Escribir un código listo para ser tratado a futuro como una tarea. Debe promediar todos los números de 64 bits almacenados en la Tabla de Dígitos, que fueron ingresados por teclado, y presentar el resultado en el extremo superior derecho de la pantalla, y almacenarlo en alguna variable situada en la sección *Datos*.
   La rutina debe cumplir los siguientes requerimientos:
   a. Ejecutarse cada 500ms. (se puede seguir ingresando datos mientras tanto)
   b. **No implementarse dentro de la IRQ0.**
   c. Situarse en la zona de *Tarea 1.*
   d. Mientras no se ejecute, establecer al procesador en estado ***halted.***

b) Adecuar el código y el *linker script* para satisfacer el siguiente mapa de memoria (Tabla 5):

Sección	Dirección inicial
Tablas de sistema	00000000h
Rutina de teclado e ISR	00157000h
Tabla de Dígitos	00100000h
Datos	00127000h
Kernel	00137000h
Tarea 1	00210000h
Pila	1FFF0000h
Secuencia inicialización ROM	FFFF0000h
Vector de reset	FFFFFFF0h

Tabla 5 Mapa de memoria ejercicio 7

Objetivos conceptuales

I. *Analizar la implementación del direccionamiento realizada por el compilador y el linker.*
II. *Asociar los esquemas de direccionamiento del procesador con los utilizados por el programador y las herramientas (compilador y linker).*

8. PAGINACIÓN BÁSICA

Tomando como base al ejercicio anterior implementar un sistema de paginación [11] en modo *identity mapping* y adecuarlo a los siguientes lineamientos:

a) Estructurar el programa de forma tal que disponga de las siguientes secciones (la denominación se realiza acorde al estándar ELF) con sus respectivas propiedades:
   I. **Sección de código (TEXT):** no debe contener ningún tipo de dato/variable.
   II. **Sección de datos (DATA):** contiene los datos inicializados de lectura/escritura.
   III. **Sección de datos (RODATA):** contiene los datos inicializados de solo lectura.
   IV. **Sección de datos no inicializados (BSS):**
b) Implementar un controlador básico de #PF que indique el motivo de la excepción.
c) El tamaño del binario compactado no debe superar 64kB.
d) El mapa de memoria luego de la expansión del binario debe cumplir con el siguiente esquema:

Sección	Dirección inicial
Tablas de sistema	00000000h
Tablas de Paginación	00010000h
Video	000B8000h
ISRs	00157000h
Datos	00127000h
Tabla de Dígitos	00100000h
Kernel	00137000h
TEXT Tarea1	00210000h
BSS Tarea 1	00220000h
Data Tarea 1	00230000h
RODATA Tarea 1	00240000h
Pila Kernel	1FFF0000h
PIla Tarea 1	1FFF1000h
Secuencia inicialización ROM	FFFF0000h
Vector de reset	FFFFFFF0h

Tabla 6. Mapa de memoria ejercicio 8

Utilizar alguna herramienta interpretación de formato de archivos binarios para analizar los datos de posicionamiento en memoria.

Objetivos conceptuales

i. *Identificar los esquemas de direccionamiento del procesador.*
ii. *Analizar la implementación del direccionamiento realizada por el compilador y el linker.*
iii. *Asociar los esquemas de direccionamiento del procesador con los utilizados por el programador y las herramientas (compilador y linker).*

iv. *Relacionar las características de las diferentes secciones con los tipos de memoria y el mapa de direcciones de la PC.*
v. *Entender el funcionamiento del "linker script".*

Referencias

[11] Volumen 3, Capítulo 4, sección 1, Intel® 64 and IA-32 Architectures Software Developer Manuals.

9. PAGINACIÓN AVANZADA

Modificar el esquema de paginación del ejercicio anterior para satisfacer el mapa de memoria de la .

Sección	Dirección Lineal	Dirección Física
Tablas de sistema	00000000h	00000000h
Tablas de Paginación	00010000h	00010000h
Video	00020000h	000B8000h
ISRs	00100000h	00157000h
Datos	01200000h	00127000h
Tabla de Dígitos	01210000h	00100000h
Kernel	01220000h	00137000h
TEXT Tarea1	01310000h	00210000h
BSS Tarea 1	01320000h	00220000h
Data Tarea 1	01330000h	00230000h
RODATA Tarea 1	01340000h	00240000h
Pila Kernel	2FFF8000h	1FFF0000h
Plla Tarea 1	0078F000h	1FFF1000h
Secuencia inicialización ROM	FFFF0000h	FFFF0000h
Vector de reset	FFFFFFF0h	FFFFFFF0h

Tabla 7. Mapa de memoria ejercicio 9

Objetivos conceptuales

i. *Comprender en profundidad el mecanismo de paginación.*
ii. *Dominar la herramienta "linker script".*

10. PAGINACIÓN REAL

En base al código anterior modificar las siguientes funcionalidades:

a) La rutina de promedio (Tarea 1) debe intentar leer en la dirección que se obtenga como resultado de promedio. En caso de que dicho número supere la RAM del sistema (512MB) se omitirá la lectura.
b) El controlador de **#PF** al detectar que el error se debe a una página no presente, deberá asignar a la dirección que produjo el error una nueva página a partir de la dirección física 0x0C000000h.

Tenga en cuenta que las estructuras de paginación deberán completarse en forma dinámica.

Objetivos conceptuales

I. *Comprender en profundidad el controlador la excepción de Page Fault.*

Referencias

Se recomienda leer nuevamente:
Volumen 3, Capítulo 6, sección 15, Intel® 64 and IA-32 Architectures Software Developer Manuals, la descripción de la excepción de Page Fault (14).

11. CONMUTACIÓN DE TAREAS

Incorpore al programa desarrollado hasta el momento una capacidad mínima de administración de tareas [11]. Para ello se requiere, agregar las siguientes prestaciones:

a) Implementar 3 tareas a saber
   i. **Promedio** (1 tarea). Es la tarea presente al momento. Sigue ejecutando cada 500 ms.
   ii. **Suma** (2 tareas). Estas tareas de ejecutarán cada 100 y 200 ms. respectivamente. En todos los casos utilizarán como sumandos los datos de Tabla de dígitos, presentando el resultado en pantalla y al finalizar debe establecer al procesador en estado *halted*. Deshabilitar (**no borrar**) el código de generación de PF.
   iii. **Idle** (1 tarea). Su única función es establecer al procesador en estado *halted y se debe ejecutar cuando ninguna otra tarea se encuentre en ejecución*.

b) Modificar el valor del temporizador 0 del PIT, para que genere una interrupción cada 10 mseg aproximadamente.

c) Modificar el controlador de la interrupción 32 (**IRQ0, timer tick**), para que actué como conmutador de tareas (**scheduler**). Diseñe dicho mecanismo [12] para que despache las tareas en forma secuencial. El mecanismo de conmutación de contextos deberá implementarlo finalmente de forma manual (transitoriamente puede analizar el mecanismo automático provisto por el procesador) [13] [14].

d) Modificar el mapa de memoria al esquema de la Tabla 8

Sección	Dirección Lineal	Dirección Física
Tablas de sistema	00000000h	00000000h
Tablas de Paginación	00010000h	00010000h
Video	00020000h	000B8000h
ISRs	00100000h	00157000h
Datos	01200000h	00127000h
Tabla de Dígitos	01210000h	00100000h
Kernel	01220000h	00137000h
TEXT Tarea1	01310000h	00210000h
BSS Tarea 1	01320000h	00220000h
Data Tarea 1	01330000h	00230000h
RODATA Tarea 1	01340000h	00240000h
TEXT Tarea2	01410000h	00310000h
BSS Tarea 2	01420000h	00320000h
Data Tarea 2	01430000h	00330000h
RODATA Tarea 2	01440000h	00340000h
TEXT Tarea3	01510000h	00410000h
BSS Tarea 3	01520000h	00420000h
Data Tarea 3	01530000h	00430000h
RODATA Tarea 3	01540000h	00440000h
TEXT Tarea 4	01610000h	00510000h
BSS Tarea 4	01620000h	00520000h
Data Tarea 4	01630000h	00530000h
RODATA Tarea 4	01640000h	00540000h

Pila Kernel	2FFF8000h	1FFF0000h
Plla Tarea 1	2FFFF000h	1FF01000h
Plla Tarea 2	30000000h	1FF02000h
Plla Tarea 3	30001000h	1FF03000h
Plla Tarea 4	30002000h	1FF04000h
Secuencia inicialización ROM	FFFF0000h	FFFF0000h
Vector de reset	FFFFFFF0h	FFFFFFF0h

Tabla 8. Mapa de memoria ejercicio 10

Objetivos conceptuales

i. *Identificar todos los registros y datos utilizados para realizar la conmutación de tarea.*
ii. *Comprender que el espacio de direcciones observado por cada tarea es completamente independiente de la dirección física y la traducción solo es conocida por el SO.*

Referencias

[12] Volumen 3, Capítulo 7, sección 1, Intel® 64 and IA-32 Architectures Software Developer Manuals,
[13] Volumen 3, Capítulo 7, sección 3, Intel® 64 and IA-32 Architectures Software Developer Manuals,
[14] Volumen 3, Capítulo 7, sección 2, Intel® 64 and IA-32 Architectures Software Developer Manuals,
[15] Volumen 3, Capítulo 7, sección 7, Intel® 64 and IA-32 Architectures Software Developer Manuals.

12.SIMD

Modificar la Tarea 2 para que realice la suma aritmética saturada en tamaño word y la Tarea 3 para que lo realice en tamaño quadruple word
Implementar el soporte necesario para el resguardo de los registros MMX/SSE mediante la excepción 7 (***#NM***) [15][16], así como también realizar las modificaciones correspondientes en la función de cambio de contexto.

Objetivos conceptuales

i. *Identificar todos los registros y datos que utiliza el procesador para verificar si se ha utilizado una instrucción SIMD.*
ii. *Comprender los diferentes tipos de aritmética utilizados por el procesador.*

Referencias

[16] Volumen 3, Capítulo 6, sección 15, Intel® 64 and IA-32 Architectures Software Developer Manuals,
[17] Volumen 3, Capítulo 12, Intel® 64 and IA-32 Architectures Software Developer Manuals,

13. NIVELES DE PRIVILEGIO

En base a lo elaborado anteriormente implementar un sistema que permita manejar niveles de privilegio. A tal fin modificar/agregar los siguientes ítems:

a) La GDT debe contemplar los descriptores de nivel 3 (**PL=11** usuario) [17], tanto para código como para datos.
b) Adecuar las diferentes entradas de la tabla de paginación según corresponda a usuario o supervisor, verificando además que los permisos de lectura y escritura sean consistentes con la sección asociada.
c) Las tareas 1 a 3 deben ejecutarse en nivel 3. Analizar si es necesario disponer de una pila por cada tarea y realizar las modificaciones pertinentes acorde a su respuesta.
d) Diseñar un mecanismo apropiado de acceso para las siguientes funciones de sistema (*system calls*). Utilice el vector **80h** para los servicios, o bien un **CALL FAR**.
   i. ***void td3_halt(void);***
   ii. ***unsigned int td3_read(void *buffer, unsigned int num_bytes);***
   iii. ***unsigned int td3_print(void *buffer, unsigned int num_bytes);***
e) Modificar el mapa de memoria al siguiente esquema

Sección	Dirección Lineal	Dirección Física
Tablas de sistema	00000000h	00000000h
Tablas de Paginación	00010000h	00010000h
Video	00020000h	000B8000h
ISRs	00100000h	00157000h
Datos	01200000h	00127000h
Tabla de Dígitos	01210000h	00100000h
Kernel	01220000h	00137000h
TEXT Tarea1	01310000h	00210000h
BSS Tarea 1	01320000h	00220000h
Data Tarea 1	01330000h	00230000h
RODATA Tarea 1	01340000h	00240000h
TEXT Tarea2	01410000h	00310000h
BSS Tarea 2	01420000h	00320000h
Data Tarea 2	01430000h	00330000h
RODATA Tarea 2	01440000h	00340000h
TEXT Tarea3	01510000h	00410000h
BSS Tarea 3	01520000h	00420000h
Data Tarea 3	01530000h	00430000h
RODATA Tarea 3	01540000h	00440000h
TEXT Tarea 4	01610000h	00510000h
BSS Tarea 4	01620000h	00520000h
Data Tarea 4	01630000h	00530000h

RODATA Tarea 4	01640000h	00540000h
Pila Kernel	2FFF8000h	1FFF0000h
PIla Tarea 1	2FFFF000h	1FF01000h
PIla Tarea 2	30000000h	1FF02000h
PIla Tarea 3	30001000h	1FF03000h
PIla Tarea 4	30002000h	1FF04000h
PIla Kernel Tarea 1	3FFFF000h	1FF05000h
PIla Kernel Tarea 2	40000000h	1FF06000h
PIla Kernel Tarea 3	40001000h	1FF07000h
PIla Kernel Tarea 4	40002000h	1FF08000h
Secuencia inicialización ROM	FFFF0000h	FFFF0000h
Vector de reset	FFFFFFF0h	FFFFFFF0h

Tabla 9. Mapa de memoria ejercicio 13

Objetivos conceptuales

i. *Identificar todos los registros y datos que utiliza el procesador para verificar si es posible efectuar una conmutación de privilegio.*
ii. *Analizar en qué momento actúa cada unidad de direccionamiento para validar el acceso a una región de memoria determinada*
iii. *Identificar todos los registros y datos que utiliza el procesador para verificar si es posible acceder a un dato desde un código de cierto nivel de privilegio.*
iv. *Analizar la gestión de la pila realizada para una interrupción al conmutar de nivel de privilegio.*
v. *Reconocer los campos necesarios para identificar el privilegio de una sección en las tablas de paginación.*
vi. *Analizar la gestión de la pila realizada para una interrupción.*
vii. *Comprender qué información se almacena en el espacio de contexto de cada.*

Referencias

[18] Volumen 3, Capítulo 5, Intel® 64 and IA-32 Architectures Software Developer Manuals.

14. PRIMER RECUPERATORIO

Modificar la Tarea 1 a fin de que implemente mediante SIMD una convolución entre las funciones escalón definidas a continuación

unsigned int signal_a[] = {0, 0, 0, 0, 1, 1, 1, 1, 1, 1}

unsigned int signal_b[] = {0, 0, 1, 1, 1, 1, 1, 1, 1, 1}

Adicionalmente la tarea debe obtener el tiempo consumido por el cálculo y presentarlo en pantalla. A tal fin se debe utilizar la instrucción **rdtsc**. Se brinda a modo de referencia la secuencia de instrucciones a implementar

cpuid

rdtsc

mov mi_var_time, eax

call convolucion

cpuid

rdtsc

sub eax, mi_var_time

15. SEGUNDO RECUPERATORIO

Agregar una tarea que ejecute en modo kernel llamada Monitor de Tareas . Se ejecuta cada 200ms. Debe mostrar en pantalla una tabla en modo texto con la siguiente información de cada tarea. A modo de ejemplo:

Nombre Tarea	% CPU	Memoria (Kb)
Promedio	1	24
Suma 1	1	32
Suma 2	1	32
Monitor de Tareas	1	12
Idle	96	128

El % de CPU se calculará como ticks (running) / ticks totales del sistema. La memoria es la cantidad empleada por la tarea. En el caso de tablas de sistema compartidas con tareas, se computarán a la tarea idle.