UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Departamento de Ingeniería Electrónica

Técnicas Digitales III

Guía de Trabajos Prácticos

Primer cuatrimestre

Ciclo lectivo 2024 **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.
Requisitos previos:
Haber completado el tutorial de instalación de QEMU: "Guía de Instalación de QEMU",
Haber completado las dos prácticas Guiadas para instalar el tool chain de desarrollo:

1. **https://gitlab.frba.utn.edu.ar/td_piloto/qemu_primer_test**
2. **https://gitlab.frba.utn.edu.ar/td_piloto/qemu_segundo_test**

Una vez completados servirán de base a la siguiente secuencia incremental de trabajos que culminarán en un escueto y modesto micro kernel bare metal.

1. ASSEMBLER ARMV7

Escribir un código en ensamblador capaz de ordenar un array de enteros.
Para ello generar un espacio de 10 enteros inicializados y un espacio de 10 enteros para ubicar el resultado ordenado.
Verificar el funcionamiento mediante el debugger.

Objetivos conceptuales

I. *Familiarizarse con los comandos de **Qemu** y su utilización*
II. *Familiarizarse con las instrucciones de ensamblador de ARM*
III. *Familiarizarse con el uso del debugger*
IV. *Familiarizarse con el uso de GIT*

2. LINKER SCRIPTS

Modificar el ejemplo del segundo test provisto en Gitlab en el item 2. de párrafo anterior, que en el vector de arranque pueda copiar una sección de código de lo que será el micro kernel de nuestro sistema operativo, en la dirección 0x70030000. Para tal fin implementar:

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

Una vez que se copia esta sección, el programa en el vector de arranque debe saltar al código de la sección copiada, en donde quedará en un ciclo infinito.

La pila debe ubicarse en 0x70020000.

Las dos secciones de código deben quedar ensambladas una atrás de la otra. Para tal fin se sugiere utilizar un archivo de linker script de modo de gestionar las áreas de memoria durante el ensamblado del binario.

Objetivos conceptuales

V. *Familiarizarse con los comandos de **Qemu** y su utilización*
VI. *Familiarizarse con Linker Script*
VII. *Entender manejo de pila **R13***
VIII. *Entender como cargar el registro **LR** para poder retornar de las funciones.*

3. INTERRUPCIONES BÁSICAS - SWI

Se pide continuar el ejercicio anterior incorporando un sistema básico de atención de interrupciones/excepciones.

Para tal fin, se deberá definir un "handler" genérico para atender la interrupción **SVC.**

Para corroborar su funcionamiento, debemos poder llamar a la instrucción **SWI y validar que atendemos el pedido correctamente.**

Objetivos conceptuales

i. *Familiarizarse con la inicialización de interrupciones/excepciones*
ii. *Entender el concepto de servicio de sistema operativo, que luego será usado por código de usuario para acceder a acciones privilegiadas, como hace un S.O. real.*

4. INTERRUPCIONES DE HW Y EXCEPCIONES – TIMER TICK

Se requiere continuar el ejercicio anterior habilitando únicamente la IRQ asociada al temporizador y la atención de excepciones de los siguientes tipos:
* Invalid instructions and trap exceptions
* Memory accesses
* Exception-generating instructions. Supervisor Call (SVC) solamente

Cada rutina de atención de excepción deberá almacenar en **r10** los caracteres del tipo que la generó ("INV", "MEM", "SVC"), y permitir que el procesador puede continuar con la ejecución normal del código.
Cada excepción se debe generar por una función específica. Se sugiere que la ejecución de dichas funciones pueda des/habilitarse mediante una opción de construcción del binario, como por ejemplo *make exceptions*.

Se debe configurar el TIMER0 de forma que genere una interrupción cada 10ms y la rutina de atención a la IRQ asociada (#36) debe almacenar en **r10** la cantidad de veces que fue invocada. Recordar configurar adecuadamente el GIC0.

El código, una vez finalizada la configuración de todos los recursos de HW descriptos y generadas las excepciones, según la metodología de construcción del binario escogida, debe quedarse en un bucle que suspenda la ejecución (detener el reloj principal del core) hasta recibir un evento IRQ.

Objetivos conceptuales

i. *Familiarizarse con IRQ vs FIQ.*
ii. *Comprender la atención de excepciones.*
iii. *Comprender la importancia de poner el procesador en alta impedancia para reducir el consumo de energía mientras esperamos por eventos.*

5. MMU – Paginación Básica

Continuar el ejercicio anterior, habilitando paginación básica para correr el mismo código sin cambios, en un modelo de paginación conocido como **identity mapping**.

Objetivos conceptuales

i. *Comprender los conceptos de dirección lógica, lineal y física.*
ii. *Comprender el mecanismo de la unidad de paginación del procesador.*
iii. *Entender los periféricos asociados (coprocesadores)*

6. Task Scheduler Simple

Tomar el ejercicio anterior, y agregar dos tareas.

- Tarea 1: deberá corroborar el correcto funcionamiento de la RAM (escribiendo y luego leyendo lo escrito) en el rango 0x70A00000 a 0x70A0FFFF, utilizando el word 0x55AA55AA. Debe resguardar la información antes de escribir cada bloque.
- Tarea 2: deberá recorrer la RAM en el rango 0x70A10000 a 0x70A1FFFF. En cada porción de memoria, deberá invertir de lo leído, es decir, si lee un bit en 1 lo debe pasar a 0, y viceversa.

La administración de la ejecución de las tareas se realizará desde un scheduler que operará dentro de una ventana de tiempo de 100ms. Las tareas se ejecutarán durante 10ms antes de ser suspendidas hasta la próxima ventana de tiempo.
Las tareas corren indefinidamente.
El tiempo remanente de la ventana de tiempo lo ocupará una tercera tarea que ponga el procesador en un estado de bajo consumo de energía, hasta el inicio de la próxima ventana de tiempo, o hasta que ocurra un evento asincrónico.

El mapa de memoria propuesto es el siguiente:

Sección	Dirección inicial
Tarea 2 . readingArea	70A10000h
Tarea 1 .readingArea	70A00000h
Tarea 1 .text	70F50000h
Tarea 1 .data	70F51000h
Pila Tarea 1	70F52000h
Tarea 1 .bss	70F53000h
Tarea 1 .rodata	70F54000h
Tarea 2 .text	70F40000h
Tarea 2 .data	70F42000h
Tarea 2 .bss	70F43000h
Tarea 2 .rodata	70F44000h
Pila Tarea 2	70F45000h
Kernel .data (datos y tablas)	70031000h
Kernel .text	70030000h
Kernel .rodata	70023000h
Kernel .bss	70022000h
Pila Kernel	70020000h
Secuencia inicialización ROM	70010000h
Vector de INT	70000000h

Observar que no se define un área para la tercera tarea.

Objetivos conceptuales

i. *Comprender la protección de espacios de memoria por paginación.*
ii. *Comprender como funciona un sistema operativo básico.*

7. MANEJO DE PRIVILEGIOS

Para el ejercicio anterior, se requiere ahora ubicar las tareas 1 y 2 a modo USR.

Para poder leer la RAM, ahora las tareas tendrán que usar un servicio privilegiado (usando SWI), que permita realizar las tareas del ejercicio anterior en modo supervisor.

Objetivos conceptuales

i. *Comprender la importancia del uso de servicios del sistema operativo.*
ii. *Comprender la importancia de separar tareas de kernel y de usuario, con diferentes niveles de privilegio.*