Universidad Tecnológica Nacional

Facultad Regional Buenos Aires

Departamento de Electrónica

Cátedra: Técnicas Digitales III - Plan 95A

GUIA DE TRABAJOS PRACTICOS

PARTE I: 1er Cuatrimestre

Ciclo Lectivo 2011

Indice

Trabajos Prácticos de Técnicas Digitales III

Introducción y Régimen de aprobación

La presente guía de Trabajos Prácticos tiene por objeto llevar a la práctica los contenidos vistos en las clases teóricas. De este modo se espera una realimentación entre la aplicación y la lectura de los diferentes conceptos teóricos que permita desarrollar en el alumno un enfoque metodológico para resolver problemas de Ingeniería utilizando como herramientas los diferentes componentes digitales, subsistemas y sistemas aprendidos a lo largo del presente ciclo lectivo.

El grado de complejidad irá creciendo a través de los diferentes ejercicios planteados para cada Unidad Temática.

Cada alumno deberá presentar aquellos ejercicios que lleven la indicación ***Entrega Obligatoria***. La entrega de cada ejercicio se efectuará sin excepciones en las fechas estipuladas en el cronograma de clase que se entregará en la primera clase del ciclo lectivo.

Los calendarios de entrega de los prácticos obligatorios estarán diseñados para que todos los Trabajos Prácticos correspondientes a los contenidos que se incluyen en cada parcial sean revisados por los docentes auxiliares antes del examen. De este modo los alumnos tendrán una devolución con las correcciones de los errores detectados, como forma de realimentación necesaria para el examen parcial.

La no entrega de un ejercicio en la fecha establecida equivale a considerar al alumno o al grupo **Ausente** en ese práctico. En consecuencia se considerará **No Aprobado** dicho práctico. De acuerdo con el reglamento vigente, la aprobación de los Trabajos Prácticos requiere el 80% de los Prácticos Aprobados. En el caso de esta guía de Trabajos Prácticos se requiere la aprobación del 80% de los estipulados de ***Entrega Obligatoria***.

Formato de presentación

- Los archivos fuentes deben tener en todos los casos los comentarios necesarios para clarificar su lectura.

- Deben llevar por cada subrutina / función, un encabezado con la descripción de la operación que realiza, los parámetros que espera como entrada, y en que forma y donde entrega sus resultados.

- Como encabezado del programa, debe haber un comentario que explique claramente que hace dicho programa, y las instrucciones detalladas (comandos) para su compilación y linkeo.

IMPORTANTE:

FORMA DE ENTREGA DE LOS TRABAJOS PRACTICOS

La entrega se realizará en primer lugar en el repositorio de versiones SVN de la cátedra.

El aumno deberá ir volcando en ella los prácticos intermedios que lo conducirán al entregable, con la frecuencia en que vaya dedicándose a las tareas de ejercitación previstas en la asignatura.

La versión final del Trabajo Práctico se compondrá de los programas fuentes necesarios, el makefile que permita su compilación y un archivo de texto plano readme con las instrucciones adicionales que el alumno considere pertinente para la ejecución en bochs de su programa. Además, cada archivo fuente deberá estar correctamente documentado utilizando para ello la herramienta estándar Doxygen. Sin excepciones.

Por otra parte aquellos ejercicios que lo requieren tienen condiciones específicas que hacen a lo que se denomina buena práctica de programación.

La no presencia de la versión final completa del TP en la fecha estipulada por parte del alumno en el repositorio indicado se considerará ausente.

Cada equipo de docentes auxiliares establecerá si lo desea como alternativa el envío de las prácticas vía e-mail, sin que esto signifique el relevo e responsabilidad para el alumno de subir su práctica al server SVN de la cátedra.

Solo bajo prueba fundada de indisponibilidad del servicio de internet de la Facultad se aceptarán entregas via e-mail

T.P. Nº 1.Herramientas de Desarrollo

Ejercicio 1.1. Compilar el Bochs, habilitando las opciones de Debugging, Disassembly y SIMD.

Utilizaremos el entorno de emulación x86 Bochs: http://bochs.sourceforge.net/

Bochs es una máquina virtual del tipo vmware aunque algo mas rústico, pero free, y tiene una virtud mayúscula: permite debuggear la PC. Es decir que podremos arrancar en modo debug y avanzar la ejecución de la máquina desde la dirección 0xFFFF:0xFFF0, es decir la primer dirección en donde empieza la rutina del POST pudiendo si lo deseamos examinar el código hasta que se cargue el sistema operativo y una vez cargado seguir dentro del sistema, hasta que se cargue una aplicación a la que también obviamente se podrá debuggear. Es posible establecer breakpoints, y tracear a través de rutinas o pasarlas de un paso como en cualquier debugger normal.

Los comandos del debugger de Bochs están basados en el debugger gdb (GNU Debugger). Podríamos decir que es un subset del gdb.

El manual detallado de comandos del debugger de bochs está en el siguiente link: http://bochs.sourceforge.net/doc/docbook/user/internal-debugger.html

Existen versiones para Windows y Linux. Esta práctica al igual que el resto de la guía se desarrollará íntegramente en Linux.

En la distribución de Debian, el paquete que está en el repositorio es el emulador sin opción de debugging. Para habilitar la función de debugging, entonces es necesario bajar la última versión de fuentes del repositorio dado al principio de esta sección, y compilarlo con las opciones adecuadas. Por lo tanto, hay que compilarlo, pasándole una opción correspondiente al shell script ***./configure***...

1. Bajar el paquete de archivos fuentes del repositorio correspondientes a la última versión (2.4.6 al momento).

2. Extraer su contenido y descomprimirlo usando el comando tar.

3. Ejecutar en el directorio en el que se descomprimió el script configure con las siguientes opciones:

./configure --enable-debugger --enable-disasm --prefix=<poner aquí el path en el que se desea instalar> –-enable-fpu --enable-configurable-msrs –-enable-cpu-level=6 --enable-pci --enable-usb

Estos parámetros establecen el tipo de CPU a emular, habilitación de la fpu (aunque de acuerdo a la documentación de esta versión por default se habilita), entre otras cosas. Lo mas importante es que habilitamos con --enable-debugger al debugger interno de bochs (luego en el archivo de configuración podremos habilitar una práctica interfaz gráfica para no usarlo en línea de comandos), y con –enable-disasm preparamos al debugger para desensamblar código a partir de una dirección de memoria dada.

A continuación enumeramos la lista de opciones y su significado extraída de la documentación.

Opción	Default	Comentarios
--enable-cpu-level={3,4,5,6}	6	Selecciona cual nivel de CPU emular. Las opciones son 3, 4, 5, y 6, que corresponden respectivamente a los procesadores 386, 486, Pentium, o Pentium Pro y posteriores.
--prefix= *	None	Setea el path en el que se instalarán los archivos ya compilados.
--enable-fpu	yes	Si deseamos utilizar la FPU del procesador es necesario compilar con esta opción. (Escrita por Stanislav Shwartsman).
--enable-configurable-msrs	no	Soporte para los MSR's (ver ejemplos en msrs.def)
--enable-vbe **	no	Habilita el uso de las VGA BIOS Extensions (VBE) (por Jeroen Janssen), Ver Sección 8.16 de la documentación para ampliar.
--enable-debugger	no	Compila con soporte para el Bochs internal command-line debugger. Se trata de un debugger nativo, no intrusivo, poderoso, y basado en el ***gdb*** (mas bien es un subset del ***gdb*** a nivel de comandos

		). Habilitarlo necesariamente hará mas lenta la emulación pero es una herramienta fundamental para los desarrolladores de un kernel. Ver Sección 8.11 para mas información.
--enable-disasm	yes	Compila con soporte para el desensamblador interno. Es útil para trabajar en conjunto con el debugger interno (--enable-debugger), o para desensamblar la instrucción actual cuando bochs genera un panic.
--enable-pci	no	Habilita un soporte limitado compatible con i440FX PCI. Aún está incompleto, pero es utilizable.
--enable-usb	no	Habilita soporte (UHCI) para i440FX PCI USB. El host controller con un root hub de 2 ports y 6 tipos de USB devices disponibles.
--enable-gdb-stub	no	Habilita una conexión vía socket con gdb. Esta opción es mutuamente excluyente con --enable-debugger.

Nota *: Utilizar /opt/bochs-2.4.6, como argumento de modo de instalarlo en el path que se ha adoptado como estándar en la Cátedra.

Nota **: VBE se refiere al modo gráfico emulado por Bochs (VESA Bios Extensions). Sin embargo si bien en la documentación figura entre las opciones disponibles, en la práctica está obsoleta (deprecated), y su inclusión en el script ***configure***, generará que éste aborte su ejecución al momento de aplicar ésta opción, dejando incompleto el proceso de configuración, hecho que impide compilar con éxito la aplicación. Por lo tanto las VBE a partir de la versión 2.4.6 vienen habilitadas y no requieren opciones de compilación específicas.

En http://bochs.sourceforge.net/doc/docbook/user/compiling.html#CONFIG-OPTS se pueden ver el resto de las opciones.

Es interesante jugar con diferentes opciones y ver el comportamiento.

Las sugeridas aquí aplican a los modestos fines de esta Guía de Trabajos Prácticos.

Una vez ejecutado el script ***configure*** con las opciones indicadas, se ejecutan las siguientes líneas:

make

para construir el binario de bochs con las opciones indicadas al ejecutar el script configure previamente.

sudo make install

Para copiar en el path indicado en la opción --prefix=

Si respetó el path ***/opt/bochs-2.4.6*** que se sugiere en la práctica, su usuario no tendrá allí permisos de escritura, por eso se debe utilizar sudo.

Para mas información sobre estos parámetros consultar la página de bochs mencionada anteriormente.

Ejercicio 1.2. Compilar el Bochs, habilitando las mismas opciones, pero reemplazando el debugger interno por el gdb.

Ejecutar los mismos pasos que en el ejercicio anterior pero sustituyendo la opción --enable-debugger del script ***configure*** por la opción --enable-gdb-stub.

Por otra parte se deberá cambiar el path para la opción --prefix, para que se instale sin pisar el anterior y tener ambas alternativas disponibles. Utilizar ***/opt/bochs-2.4.6-gdb***.

Ejercicio 1.3. Configuración de la versión de bochs compilada con debugger interno

El Bochs se configura mediante el archivo ***.bochsrc***. En la instalación base se incluye un archivo ejemplo que detalla muchas de las opciones disponibles.

Con motivo de testear la instalación, también se provee de una imagen de DOS para verificar que la instalación de Bochs funciona correctamente: http://www.electron.frba.utn.edu.ar/~afurfaro/files/dosboot.rar

Existe una lista de imágenes con un SO ya pre-instalado en http://bochs.sourceforge.net/diskimages.html.

Tomar el archivo .bochsrc del paquete descomprimido y copiarlo en el directorio de trabajo (directorio desde donde se va a ejecutar bochs, o en un directorio fijo. En este último caso habrá que aclararlo como un argumento al comando ***bochs***.

Editar la copia del archivo ***.bochsrc***. Todas las líneas que comienzan con el carácter '#' son comentarios. Observando el archivo vemos que lleva incluida alguna documentación adicional mas una generosa cantidad de opciones de configuración a las que simplemente habrá que descomentar para que queden habilitadas, y eventualmente modificar algún parámetro para adaptar a nuestra configuración.

Para completar el archivo de configuración vamos desdde el inicio del archivo ***.bochsrc*** habilitando o des comentando líneas y completando con la información de configuración adicional:

Descomentar o escribir si no se encuentra disponible la siguiente línea:

```
display_library: x, options="gui_debug" # use GTK debugger gui
```

Configurar el archivo de emulación del BIOS de la PC a emular. Observar el path que es consistente con la opción --prefix del script ***configure***.

```
romimage: file=/opt/bochs-2.4.6/share/bochs/BIOS-bochs-latest
```

Configuración de la CPU:

```
cpu: count=1, ips=50000000, reset_on_triple_fault=1, ignore_bad_msrs=1, msrs="msrs.def"
```

Configuración de memoria (los detalles se explican en los comentarios del propio archivo ***.bochsrc***.):

```
memory: guest=32, host=32
```

Lo mismo para el BIOS de VGA estándar que tiene cualquier PC:

```
vgaromimage: file=/opt/bochs-2.4.6/share/bochs/VGABIOS-lgpl-latest
```

Habiliación de la VGA BIOS Extensions:

```
vga: extension=vbe
```

Necesitamos un floppy disk. Puede habilitarse un disquette físico des comentando y modificando la siguiente línea:

floppya: 1_44=/dev/fd0, status=inserted

O bien utiliar una imagen (sumamente útil en máquinas que ya no vienen con disquette, como una notebok)

floppya: 1_44=../diskette.img, status=inserted

Por si vamos mas adelante a habilitar un Hard disk virtualizado o el CD ROM físico des comentar la siguiente línea:

ata0: enabled=1, ioaddr1=0x1f0, ioaddr2=0x3f0, irq=14

Necesitamos al igual que en una PC definir el dispositivo desde el cual se va a bootear el kernel.

boot: floppy

El timer tick siempre es un dispositivo que no suele funcionar a la velocidad correcta. Una buena aproximación aunque algo mas lenta que el de la PC Host, es des comentar esta línea:

clock: sync=realtime, time0=local

La siguiente línea ya viene des comentada

floppy_bootsig_check: disabled=0

significa que el BIOS chequeará la firma 0xAA55 en el boot sector desde donde se cargue el kernel del Sistema Operativo. **La dejamos tal cual como está**.

Bochs posee un log, en donde se guarda la salida de su operación, particularmente útil cuando hay problemas o fallos como panic, por ejemplo. Viene seteado por default. No es relevante modificarlo sabiendo donde queda.

log: bochsout.txt

El resto del archivo puede dejarse con las opciones default tal como vienen. No hacen a nuestra aplicación particular sino mas bien al funcionamiento de bochs en si mismo.

Una vez configurado y ejecutado en el mismo directorio que contiene el archivo ***.bochsrc***, Bochs debería verse una pantalla similar a:

```
% /opt/bochs-2.4.6/bin/bochs
```

```
========================================================================
                      Bochs x86 Emulator 2.4.6.cvs
             Build from CVS snapshot, after release 2.4.6
                   Compiled at Mar 15 2011, 00:26:47
========================================================================
------------------------------
Bochs Configuration: Main Menu
------------------------------

This is the Bochs Configuration Interface, where you can describe the
machine that you want to simulate.  Bochs has already searched for a
configuration file (typically called bochsrc.txt) and loaded it if it
could be found.  When you are satisfied with the configuration, go
ahead and start the simulation.

You can also start bochs with the -q option to skip these menus.

1. Restore factory default configuration
2. Read options from...
3. Edit options
4. Save options to...
5. Restore the Bochs state from...
6. Begin simulation
7. Quit now

Please choose one: [6]
```

entonces Bochs leyó el archivo de configuración ***.bochsrc*** que describe la PC virtual x86, cargando la imagen correspondiente. Está detenido para permitir cambiar opciones antes de iniciar la ejecución. Para iniciar, presionar Enter.

Si se lo ejecuta desde un directorio diferente de aquel que contiene el archivo .bochrc, el resultado es similar, salvo por la opción que aparece como default en la última línea:

```
Please choose one: [2]
```

Esto significa que si damos Enter se ingresa a "Read options from...", esto es se deberá especificar la ruta y nombre del archivo de configuración.

Tal como indica la pantalla anterior de Bochs, puede saltearse este paso e ingresar directamente al programa ejecutándolo con la opción -q.

Verificarlo tipeando:

/opt/bochs-2.4.6/bin/bochs -q

Si el archivo de configuración está en otro lado se puede especificar con la opción -f la ruta completa y nombre del archivo.

Ejecutarlo desde un directorio hijo del actual mediante la siguiente línea y verificar que funciona igual que en el caso anterior.

/opt/bochs-2.4.6/bin/bochs -q -f ../.bochsrc

La opción -f se utiliza además siempre que el nombre del archivo de configuración sea diferente de ***.bochsrc***.

Una vez arrancada la imagen, Bochs inicia la consola de debugging:

```
Next at t=0
(0) [0x000ffff0] f000:fff0 (unk. ctxt): jmp f000:e05b         ; ea5be000f0
<bochs:1>
```

Ejercicio 1.4. Configuración de la versión de bochs compilada con soporte para gdb

gdb por ***G***NU ***D***e***B***ugger es un potente debugger simbólico cuya operación se efectúa por línea de comandos.

Existen algunas aplicaciones de interfaz gráfica que se montan sobre gdb parahacer su operación mas amena al usuario (aunque si se quere sacar el máximo potencial del gdb, lo indicado es usar línea de comandos). Estas son ***D***ata ***D***isplay ***D***ebugger, ***ddd***, y una aplicación que integra el paquete full de kde, conocida como ***kdbg***.

Esto hace atractivo el uso de Bochs con este debugger en especial si se pretende mezclar C y Assembly en las aplicaciones a debuggear.

Las modificaciones son mínimas para el archivo de configuración, de modo pongamos manos a la obra.

En el directorio donde se generó el archivo .bochsrc anterior, ejecutar el siguiente comando:

cp .bochsrc .bochsrcgdb

Esto nos proveerá de un segundo archivo de configuración que será utilizado con la versión binaria de Bochs que tiene habilitada la interfaz con gdb: /opt/bochs-2.4.6-gdb/bin/bochs. Por eso creamos dos versiones, ya que ambas opciones de debugging son mutuamente excluyentes.

Comentar nuevamente la línea:

display_library: x, options="gui_debug" # use GTK debugger gui

agregándole al inicio el carácter '#'. Esto deshabilita que Bochs intente levantar su interfaz gráfica.

Casi al final del archivo .bochsrcgdb estará comentada la siguiente línea que deberemos descomentar:

gdbstub: enabled=1, port=1234, text_base=0, data_base=0, bss_base=0

enable = 1 nos exime de cualquier explicación. El parámetro *port*, es el port tcp a través del cual se realizará la conexión con el ***gdb***. Los demás parámetros son la base de los segmentos entendidos como parte del archivo *elf* del ejecutable a debuggear.

Para comprobar su funcionamiento, simplemente ejecutamos

/opt/bochs-2.4.6-gdb/bin/bochs -q -f .bochsrcgdb

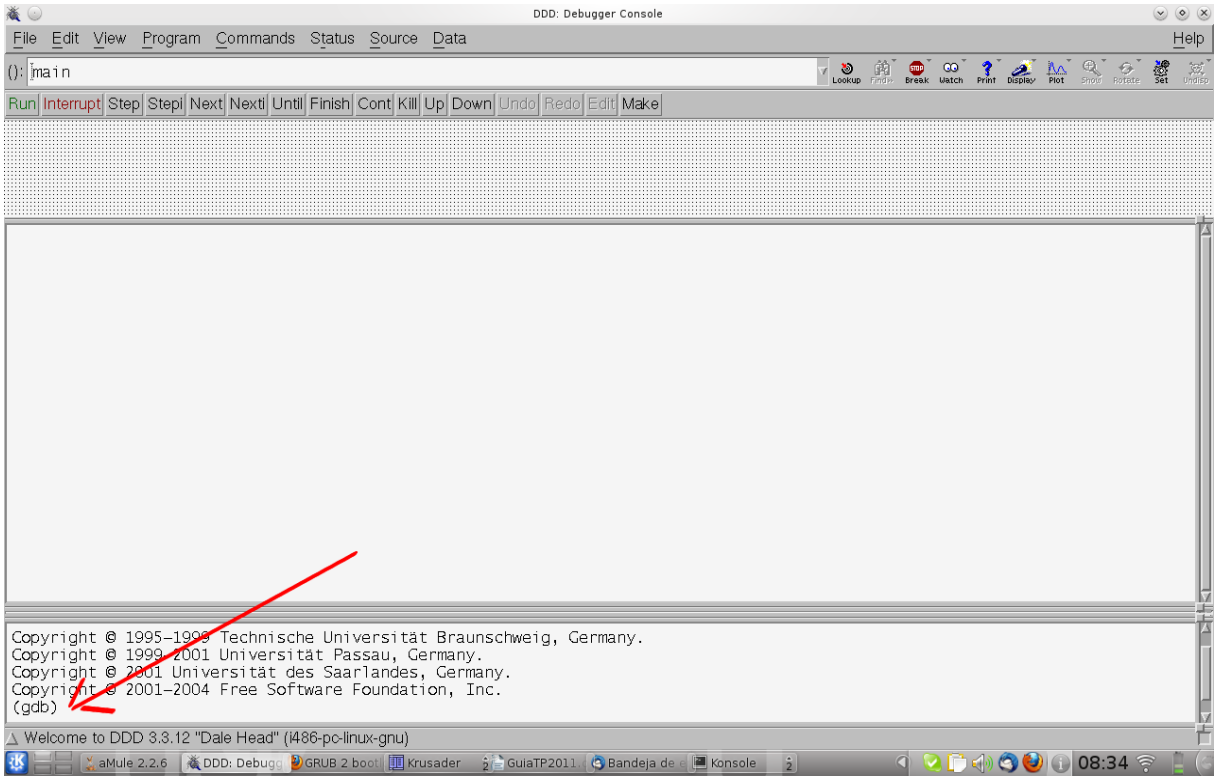
La salida es la que se ve a continuación

```
========================================================================
                      Bochs x86 Emulator 2.4.6.cvs
              Build from CVS snapshot, after release 2.4.6
                    Compiled at Mar 15 2011, 07:33:12
========================================================================
00000000000i[     ] reading configuration from ../.bochsrcgdb
00000000000i[     ] Enabled gdbstub
00000000000i[     ] installing x module as the Bochs GUI
00000000000i[     ] using log file bochsout.txt
Waiting for gdb connection on port 1234
```

Como es lógico, Bochs queda esperando la conexión por el port 1234 que le seleccionáramos en el archivo de configuración.

A continuación en el menú de KDE, Seleccionamos "Aplicaciones->Desarrollo->ddd

Se despliega la siguiente pantalla de inicio de ddd:



En donde indica la flecha es la ventana de comandos donde se puede acceder al gdb en forma directa. Allí debemos escribir el siguiente comando:

target remote localhost:1234

El comando *target remote*, del ***gdb*** sirve para conectarlo a diversos clientes remotos. Nosotros lo vamos a conectar a Bochs, como es obvio.

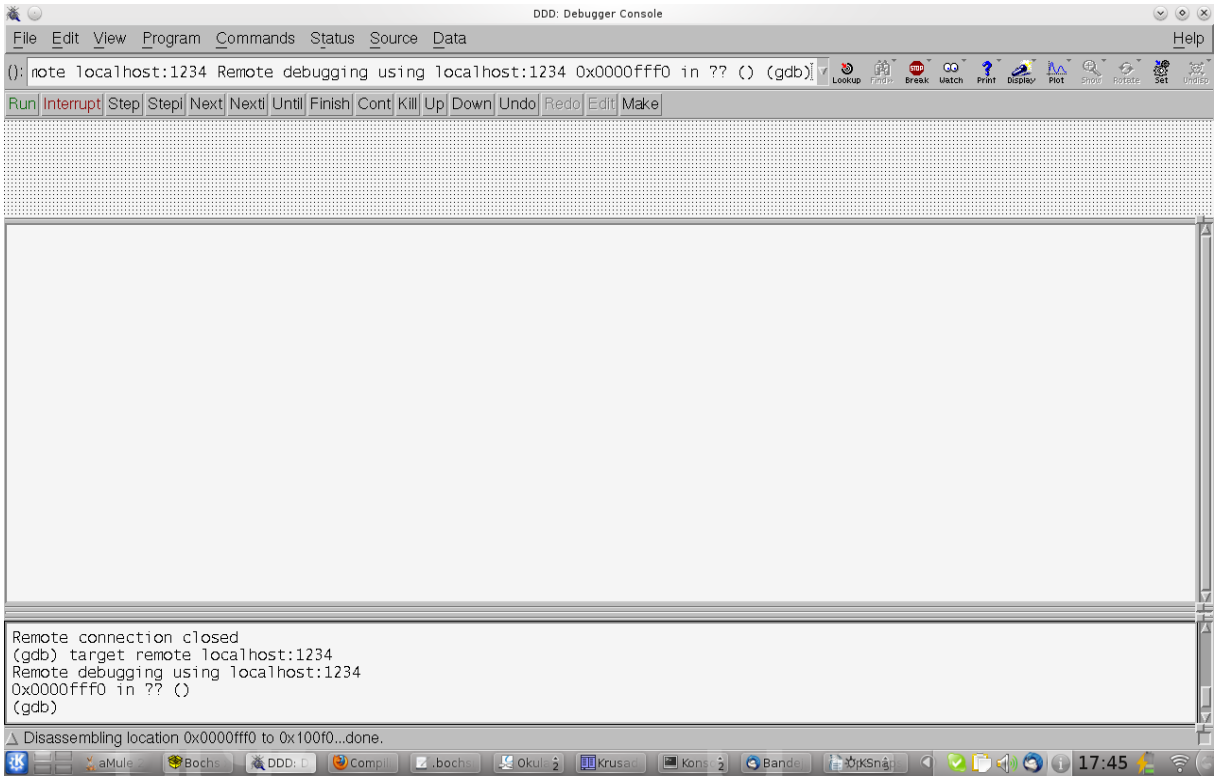
localhost:1234, es la forma de especificar una conexión: dirección IP : port. En este caso usamos *localhost* ya que la conexión es en la misma máquina.

La respuesta al comando es que en la consola de bochs se agrega la siguiente línea:

```
Connected to 127.0.0.1
```

Significa que la conexión está establecida y es posible comenzar a trabajar.

Los comandos se envían desde la ventana de comandos del ***gdb***. A propósito. Observar en dicha ventana el resultado:



Remote debugging using localhost:1234, indica que la conexión está establecida.

0x0000fff0 in ?? Es la línea de código en la que está detenido el debugger. Si observamos al salida de la consola de bochs en el ejercicio anterior veremos que está parado en el mismo lugar. Esperando comandos

(gdb) es el prompt para esperar comandos.

Ejercicio 1.5. Uso de bochs

Leer la documentación de debugging de Bochs, y realizar un seguimiento de la inicialización de la máquina virtual, ejecutando paso a paso, examinando los registros, memoria, colocando breakpoints, etc.

T.P. Nº 2. BIOS y Boot

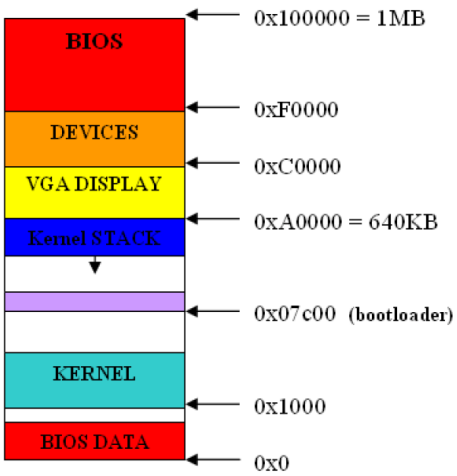
Introducción. Breve síntesis de carga del Sistema Operativo.

El proceso de boot comienza ejecutando el código de la BIOS ubicado en la posición física 0xFFFF0. Allí comienza el POST (Power On Self Test) que es parte de la ROM BIOS. El POST realiza la inicialización básica del hardware (por ejemplo, la placa de video) y su verificación. Luego busca algún dispositivo de booteo: Disco Rígido, Floppy, USB, etc. El orden de búsqueda está determinado en una pequeña memoria RAM estática que se alimenta con una pila de Li. Es lo que se conoce como setup de la PC al que se ingresa mediante alguna tecla específica disponible para tal fin durante algunos instantes cuando recién se enciende la PC. Transcurrido ese breve intervalo se desactiva la opción de ingreso al Setup.

Una vez localizado el dispositivo de arranque, **carga el primer sector de 512 bytes** (excepto en el caso de CDROM, cuyo tamaño es 2048) en la posición de memoria 0x07C00 y salta a esa dirección. Las imagen de arranque es la encargada de cargar el kernel y luego pasarle el control.

IMPORTANTE: Las imágenes de arranque deben ocupar exactamente 512 bytes (excepto en el CDROM), y estar firmada en los últimos dos bytes con 0x55AA.

A continuación se muestra un esquema posible de la distribución de la memoria al arrancar la máquina.



Link interesante: http://en.wikibooks.org/wiki/X86_Assembly/Bootloaders

Ejercicio 2.1.Debuggeando **BIOS**

Iniciar el Bochs, y debuggear el POST de la BIOS. Seguir el código tratando de ver (a grandes rasgos) que operaciones realiza el POST. Boot de un OS

Iniciar el Bochs, y cargar alguna imagen de Linux. Seguir el código e indicar el momento exacto en que el Bootloader pasa el control al Kernel.

Ejercicio 2.2. **Programa autobooteable**

En el link http://en.wikibooks.org/wiki/X86_Assembly/Bootloaders, se presenta una versión "bootloader" del programa Hello World y las líneas necesarias para compilarlo usando NASM. Descárguelo, compile y pruebe su funcionamiento, debuggeando paso a paso.

T.P. Nº 3. Compilación, formato ELF y creación de imágenes para Bochs.

Introducción: Guía de herramientas de desarrollo

Para compilar utilizaremos las herramientas estándar de Linux: gcc, make, objdump, objcopy, ld, etc.

El compilador gcc utiliza por default el formato ELF para los archivos ejecutables.

Se dispone del siguiente material para comprender como está organizado este formato:

Tool Interface Standard (TIS), Executable and Linking Format (ELF) Specification, Version 1.2, TIS Committee, May 1995.
http://www.electron.frba.utn.edu.ar/~afurfaro/descargas/Ejemplos%20programaci%f3n/Boot/elf.pdf

Executable and Linkable Format (ELF), Princeton University.
http://www.electron.frba.utn.edu.ar/~afurfaro/descargas/Ejemplos%20programaci%f3n/Boot/ELF_Format.pdf

Escriba un programa trivial en C. Lo llamaremos do_nothing.c

```
void main (void)
{
        return;
}
```

Compilarlo generando solo el programa objeto. Para ello ejecute:

gcc -c do_nothing.c

Para presentar en pantalla el código assembler de un archivo ejecutable, podemos usar la herramienta *objdump*:

objdump -S do_nothing.o

Capture la salida.

Con la herramienta *objcopy* podemos extraer el código máquina puro de un archivo objeto, filtrando el encabezado elf:

objcopy -S -O binary do_nothing.o do_nothing.bin

Con ayuda de la herramienta ***Apliaciones->Utilidades->Okteta*** visualice en hexadecimal el contenido de ***do_nothing.bin*** y compárelo con la salida del comando objdump

Ahora ejecute:

sign do_nothing.bin kernel.bin

El archivo sign genera el archivo binario de 512 bytes, firmado, listo para incluir en el archivo de configuración de Bochs. Puede bajarse de http://www.electron.frba.utn.edu.ar/~afurfaro/files/sign.

Edite el archivo kernel.bin en ***Apliaciones->Utilidades->Okteta*** y visualice en hexadecimal su contenido. ¿Qu diferencia se observa con do_nothing.bin?

Ejercicio 3.1. Práctica asistida de construcción de un programa booteable.

Una vez generado kernel.bin, se puede armar la imagen bochs de la siguiente manera:

dd bs=512 count=2880 if=/dev/zero of=bochs.img

Esta línea crea una imagen de disco de 2880 sectores de 512 bytes cada uno. Es decir una imagen de floppy disk de 1.44 Mbytes.

La imagen se guardará en el archivo especificado por el argumento del parámetro ***of=*** (***of*** por **O**utput **F**ile). La imagen se inicializará con el contenido del nodo del file system especificado en el argumento ***if=*** (***if*** por **I**nput **F**ile). En nuestro caso ***/dev/zero*** de modo que se completará de 0x00.

dd if=kernel.bin of=bochs.img count=1 seek=0 conv=notrunc

Este segundo comando copia la cantidad de sectores de 512 bytes especificados en el argumento count, desde el archivo de entrada kernel.bin al archivo de salida bochs.img, y no truncará el archivo destino si el origen es de menor tamaño (de hecho lo es, ya que kernel.bin mide 1 bloque de 512 bytes y bochs.img tiene 2880 bloques de 512 bytes).

Edite el archivo bochs.img con ayuda del editor ***Apliaciones->Utilidades->Okteta*** y visualice los resultados de ambos comandos.

Ejercicio 3.2. Segundo programa booteable

Escribir un código que no haga uso de bibliotecas externas (por ejemplo, una serie de instrucciones ASM inútiles), compilarlo, linkearlo y generar una imagen bochs que bootee ese código. Hacer un debugging para verificar que el código cargado esté correcto.

Ejercicio 3.3. Desarrollo de un bootloader

Siguiendo el tutorial:

http://www.osdever.net/tutorials/lba.php,

escribir un bootloader que lea el segundo sector del disco, y lo ejecute. Guardar el hello_world usando la BIOS en dicho sector.

Ejercicio 3.4. Combinación de C y ASM

Verificar la potencia y flexibilidad que brinda la combinación de lenguaje C y ASM según los requerimientos y complejidad del programa.

Entender el uso de los compiladores de C y ASM, de los linkeadores y sus diferencias. Paso de argumentos por la pila.

Para ello escribir dos programas:

a) Cuerpo principal en C que solicite dos operandos por consola, y una operación (s: suma, r:resta) y devuelva por pantalla el resultado.

Las operaciones serán realizadas por funciones auxiliares en ASM (func.asm)

b) Cuerpo principal en ASM. Espera presión de una tecla. Según la tecla presionada (a-z) cambia el color del texto y sale.

El "switch" que evalúa el scan code se realizará en un programa externo en C (evalscancode.c)

T.P. Nº 4. IA-32 - Modo Protegido.

Ejercicio 4.1. Entrada a Modo Protegido

Escriba un programa ***autobooteable*** que ponga al procesador en modo protegido, seguidamente ponga la pantalla en modo de video inverso, y termine su ejecución mediante HLT.

Asuma un controlador de video color.

Ejercicio 4.2. Manejo de Interrupciones en Modo Protegido

Agregue al programa desarrollado en el Ejercicio 4.1 el manejo de la interrupción de teclado de la PC (IRQ1), de modo tal que una vez puesta la pantalla en modo de video inverso espere la presión de una tecla cualquiera. Por cada tecla presionada, incrementará una variable cuyo contenido presentará en pantalla .

Agregue el manejo del COM1, inicializando el 16550 para que funcione por interrupción (IRQ4), y que incremente una variable cada vez que se reciba un carácter por el puerto, mostrando el contenido de la misma en pantalla.

Ejercicio 4.3. Ordenamiento de las Interrupciones

Tome el programa del Ejercicio 4.2. Antes de pasar a modo protegido, reprograme los PICs 1 y 2 de modo que utilicen el rango de tipos de Interrupción INT 20h a INT 2Fh.

Ejercicio 4.4. Manejo de Excepciones en Modo Protegido *Entrega Obligatoria*

Tome el código del Ejercicio 4.3 e inserte un handler para cada una de las excepciones del procesador. Se busca proveer un mínimo manejo de excepciones de modo de evitar que el sistema se estrelle ante fallas de protección. Para ello cada excepción deberá tener un handler que detenga allí al procesador permitiendo examinar los registros con el debugger del bochs.

Genere las situaciones que considere apropiadas para generar cada una de las excepciones y demostrar que sus handlers trabajan adecuadamente.

Nota: es condición que los handlers de interrupción se encuentren en un archivo separado.

Ejercicio 4.5. Cuenta de memoria RAM en el sistema

Tome el código del Ejercicio 4.4. Quite las condiciones para generar excepciones y agregue el código necesario para determinar la cantidad de memoria RAM presente en el sistema. La cantidad total en KBytes deberá imprimirse en el borde superior izquierdo de la pantalla.

Sugerencia: Construya un esquema de segmentos FLAT de 4 Gbytes para datos y códigos separados y en dos niveles de privilegio: 00 y 11

Ejercicio 4.6. Programa de arranque

Utilizando un bootloader, arrancar un sistema almacenado en un diskette virtual , que contendrá el programa del Ejercicio 4.5, con las adaptaciones que correspondan a la condición de ser booteado.

Además deberá:

- Armar una GDT con al menos 4 descriptores de segmento que conformen un modelo flat con dos segmentos de código (uno en DPL=00 y el otro en DPL=11), y dos segmentos de datos (uno en DPL=00 y el otro en DPL=11). Los 4 segmentos deben tener 4 GBytes de tamaño. Como tenemos la gate A20 deshabilitada, la GDT debe colocarse en alguna dirección del primer mega (antes de la dirección 100000h).

- Poner el controlador de video en modo gráfico, imprimir en pantalla un mensaje de bienvenida, y quedarse esperando una tecla (mediante interrupciones).

Ejercicio 4.7. Manejo de arranque con grub

Instale una máquina virtual con una versión mínima de Linux instalada como sistema host. La misma debe tener como boot manager grub2.

Tome el Programa de arranque, y modifíquelo para que pueda agregarse al menú de grub 2, de modo de ser booteable en cualquier máquina física una vez logrado el procedimiento.

Escriba un informe de no menos de 4 carillas con el procedimiento realizado. Explique cual es el estado del procesador cuando llega a su kernel, en lo

referente a modo de trabajo y definición de segmentos. Use interlineado simple, hoja A4 y font Verdana 11.

Links sugeridos para visitar:

http://www.dedoimedo.com/computers/grub-2.html

http://members.iinet.net/~herman546/p20.html

Ejercicio 4.8. Paginación.

Tome el Ejercicio 4.6 y agregue las tablas de paginación correspondientes para trabajar mediante identity mapping. Cubra solamente los primeros 4 Mbytes de RAM. De ser necesario, tome las medidas apropiadas para evitar una "Page Fault Exception".

Mapee la memoria de video del Framebuffer a partir de los 8Mb.

Ejercicio 4.9. Manejo de tareas simple – scheduler como tarea

Lea el artículo publicado en:

http://embedded.com/columns/technicalinsights/55301875?_requestid=244198

Luego tome el código del Paginación. y agregue el código necesario para administrar dos tareas encargadas de la presentación de información en forma simultánea en las mitades superior e inferior de la pantalla respectivamente, utilizando al timer tick de la PC como base de tiempos.

La información a presentar en cada mitad de la pantalla, consiste en el tiempo acumulado (expresado en décimas de segundos) que lleva escribiendo en cada mitad.

Para simplificar los cálculos en su código, reprograme el Timer Tick para generar una interrupción por mseg. Tenga en cuenta que el Timer 0 de la PC tiene una señal de clock externa establecida por un cristal de 1.19 MHz.

En su condición de arranque (default), el programa dedicará el 50% de los ciclos de timer para cada tarea (de modo que en esta condición el número presentado en cada mitad será el mismo).

Para alterar la prioridad de cada tarea (y por ende desbalancear los valores presentados en cada una) se desea utilizar la tecla F2, y la tecla F3 para aumentar en pasos del 10% la prioridad de la mitad superior e inferior

respectivamente. Cada mitad aumenta su prioridad en desmedro de la otra. Cuando se llega al extremo de tener una tarea al 0% no se la debe invocar hasta que su prioridad aumente al menos al 10%

Se vuelve al modo real limpiando la pantalla cuando se pulsa la tecla F10.

Utilizar una puerta de tarea para IRQ0, y una puerta de interrupción para IRQ1.

Ejercicio 4.10. Manejo de tareas simple – scheduler como rutina de interrupción

Repita el ejercicio 4.9 utilizando una puerta de interrupción en IRQ0.

Ejercicio 4.11. Uso del debugger de Bochs para analizar el comportamiento *Entrega Obligatoria*

Tome los kernels de los Ejercicios 4.9 y 4.10, y mediante la inclusión de breakpoints analice los siguientes aspectos del comportamiento:

1. Punto del código en el que el scheduler reasume su ejecución como respuesta al timer tick en el problema del Ejercicio 4.9. Represente en base al comportamiento observado un diagrama de transiciones entre las diferentes tareas involucradas.

2. Idem para el 4.10

3. Como se comporta el bit Busy y el Bit NT para el programa del 4.9

4. Idem para el 4.10

5. Escriba un informe del comportamiento del procesador para los casos planteados en los diferentes ítems.

Entregable: Diagrama de estados y transiciones para los ítems 1 a 4, en Visio, Powerpoit, o Word. Documento en formato .DOC para el punto 5.

Ejercicio 4.12. Ejecución en dos niveles de privilegio.

Tome el código del 4.9 o del 4.10 (elija el que prefiera), y modifíquelo para que ambas tareas ejecuten, en un segmento de RPL = 11.

Mediante el acceso a dos servicios (Fecha y Hora) implementados en el

segmento de código de RPL = 00, accediendo directamente al Real Time Clock de la PC, las tareas presentarán:

- Tarea 1: Presenta la Hora del Sistema en la posición de pantalla Fila 8 Columna 35, en el formato **hh:mm:ss**.
- Tarea 2: Presenta Fecha y Hora del Sistema en la posición de pantalla Fila 16 Columna 35 en formato **dd:mm:aaaaa** y en Fila 17 Columna 35 en formato **hh:mm:ss**. Ídem Servicio Hora del Sistema.

Las tareas, descriptas se seguirán ejecutando de acuerdo al esquema de manejo de prioridades establecido en el 4.9 o en el 4.10 según haya sido su elección**.**

El sistema finaliza (HLT) cuando se pulsa la tecla F10, o cuando alguna de las dos tareas llega a los 3 minutos de operación efectiva.

Ejercicio 4.13. Manejo de tareas por paginación

En general los diseños de sistemas operativos modernos optan por evitar soluciones dependientes de la Arquitectura de Hardware a fin de facilitar la portabilidad de código entre procesadores diferentes.

El manjeo de tareas visto para el procesador IA-32 es bastante específico, por lo cual se requiere el siguiente cambio en el manejo de tareas:

Sin modificar las políticas de scheduling ni las tareas desarrolladas en el Manejo de tareas simple – scheduler como tareay en el Manejo de tareas simple – scheduler como rutina de interrupción , se pide modificar el despachador de tareas para no efectuar ésta mediante un jmp, sino guardando en una estructura de memoria del kernel los registros que componen el estado del procesador y realizar la conmutación de tareas por simple conmutación del sistema de tablas de páginas.

Sugerencias:

Piense cuales registros tiene sentido cambiar y cuales no son relevantes.

Puede no utilizar la TSS para resguardar el estado de una tarea en función de las conclusiones a las que lo lleve el punto anterior. Aunque, si su decisión es utilizarla tenga presente que deberá llenar y tomar su contenido a mano.

¿Como se puede optimizar memoria en este esquema de acuerdo a las dos sugerencias anteriores?

Ejercicio 4.14.Scheduler con manejo de lista de tareas de longitud dinámicamente variable.

Tome el código del , y realice las siguientes modificaciones / mejoras:

- El scheduler maneja las tareas que se encuentran en una lista de elementos que se definen a través de la siguiente estructura

```
struc task_sel
     selector        resw  1     ;Selector del TSS de la tarea
     prioridad_ini   resb  1     ;Número entero entre 1 y 10 que
                                 ;indica cuantos ciclos de timer tick
                                 ;se asignaron a la ejecución de la
                                 ;tarea al momento de su carga.
     prioridad       resb  1     ;Número entero entre 1 y 10 que
                                 ;indica cuantos ciclos de timer tick
                                 ;debe estar en ejecución la tarea
                                 ;actualmente. Al inicio es igual
                                 ;a prioridad_ini.
endstruc
```

- La lista no es enlazada, sino una lista de estructuras task_sel consecutivas en memoria terminadas en NULL (00h)

- El scheduler debe alojar a las tareas de la lista en un frame de 100 mseg. El tiempo remanente a esta duración máxima lo completará con una tarea denominada Idle que pone al procesador en estado HALT, de modo de minimizar el consumo del sistema.

- La cantidad de elementos de la lista es administrado por un módulo externo, que se encarga de insertar y remover elementos de la misma de manera transparente a su scheduler (lo desarrollará en el Ejercicio 4.16, no aquí). Este mismo módulo se encarga además de modificar las prioridades de todas las tareas de la lista de modo que siempre quepan en 100 mseg., descontando ticks a cada campo prioridad cuando se recarga de tareas el sistema, y restituyéndolos a su valor inicial a medida que se descarga la demanda de ejecución de tareas

Ejercicio 4.15. Kernel DSP *Entrega Obligatoria*

Tome el código del ejercicio 4.14 y reemplace a las tareas por las siguientes:

- Tarea 1: Recibe por COM1 un mapa de bits y lo presenta en pantalla. A partir de entonces de acuerdo a la tecla que se pulse efectuará un procesamiento determinado:

Tecla 'E' ejecuta una erosión de la imagen presentada

Tecla 'B' Muestra los bordes de la imagen presentada

Tecla 'D' ejecuta la Dilatación de la imagen presentada

- Tarea 2: Presenta al lado de la tarea 1 el histograma de brillo de las imágenes

- Tarea Idle. Pone al sistema en HLT

Ejercicio 4.16. Manejo completo de tareas.

Tome el programa del 4.14 e incorpore un sub-sistema que permita agregar y quitar tareas de la lista de ejecución dinámicamente según el procedimiento descrito en el último ítem del mencionado Ejercicio.

Ampliando la especificación el sub-sistema a incorporar debe activarse al pulsar F8. A continuación debe pulsarse la tecla "I" si se desea Insertar una tarea, seguida del número de la tarea (01 a 20), y de la prioridad (01 a 10). No requiere los datos, sino que el operador los ingresa directamente. Por ejemplo, si ingresa la string "I1708", significará Insertar la tarea 17 con 8 como prioridad. Los números ingresados se asumen en decimal. Si desea remover una tarea, luego de F8, ingresará "R", seguido del número de tarea (01 a 20).

La tarea Idle no puede removerse. Es la tarea 0 y ocupa la primer posición de la lista de tareas.

Las funciones que realizan las dos operaciones requeridas funcionarán según el siguiente detalle de especificación:

- Función ***_insert_task***:
  - Recibe como argumentos el selector de la tarea a insertar y su prioridad.
  - Inserta la tarea al final de la lista de ejecución, siempre que la tarea no esté ya incluida en la lista de ejecución. En tal caso no hace nada.
- Función ***_delete_task***:
  - Recibe como argumento el selector de la tarea.
  - Remueve la tarea de la lista de ejecución. Si la tarea no está en la lista, entonces no hace nada.

Para mayor simplicidad, las 20 tareas están todas descargadas en la memoria del sistema y la GDT tiene precargados todos los descriptores de TSS asociados a las mismas.

T.P. Nº 5.Arquitectura de Procesadores *Entrega Obligatoria*

Ejercicio 5.1.Pipeline

Un determinado microprocesador dispone de un "instruction pipeline" lineal, en el cual se implementan, mediante la técnica de "look ahead", las siguientes etapas:

a- Búsqueda de instrucciones (Instruction fetch)

b- Decodificación de instrucciones (Instruction decode)

c- Cálculo de la dirección de operandos (Operand address calculation)

d- Búsqueda de operandos (Operand fetch)

e- Ejecución de la instrucción (Instruction Execution)

f- Almacenamiento del resultado (Result Storage)

Se desea ejecutar 20 instrucciones, si cada etapa demora un ciclo de reloj en procesar la instrucción , determine:

a- La cantidad de ciclos de reloj que demora el "pipeline" en procesar las 20 instrucciones.

b- La cantidad de ciclos de reloj que demora el procesador en ejecutar las 20 instrucciones si no dispusiera de "pipeline". Calcule la optimización lograda por el uso de "pipeline".

c- Sabiendo que el 50% de las etapas demoran 30ns y el restante 50% demora 45ns en procesar cada instrucción y teniendo en cuenta que el tiempo de cada etapa de "latch" es de 5ns. Indique cual es la frecuencia máxima a la cual puede trabajar el "pipeline".

Ejercicio 5.2.Arquitectura Superescalar

Recordando que el procesador P-I de la familia IA-32 de Intel, dispone de 2 "instruction pipeline" (U & V) y sabiendo que opera a 133 MHz. Indique cuánto tiempo demora en procesar el siguiente código:

```
pushf
xor   ax, ax
mov   ax, 60h
popf
```

Ejercicio 5.3. Ejecución fuera de orden

Si a un procesador Pentium IV con motor de ejecución fuera de orden ingresa el siguiente código.

```
Align 64
      mov   edi,14456
      mov   ecx,10000
      mov   esi,[0x1F0E9708]
      cld
otro:
      lodsd
      and   [edi+eax],0xFF7FE806
      jz    true
      mov   edx,12345
      add   edi,100
true:
      loop  otro
```

Considerando que el tamaño de una línea de cache L1 en este procesador es de 64 bytes, y que las direcciones de memoria referidas en el programa no son precacheables, se pide:

a. Analizar el efecto del "cache miss" en las referencias a memoria respecto de su impacto en la ventana de ejecución.

b. De acuerdo a la cantidad y tipos de ports de ejecución del procesador, los "cache miss" que pueden producirse, y considerando que el procesador puede retirar a lo sumo 3 resultados de instrucciones por ciclo de clock, determinar cuántos ciclos de clock insume la ejecución del bloque de código anterior, y escriba la secuencia en que se ejecutarán las instrucciones.

c. Compare el resultado de b., con el de un procesador de generación anterior, es decir, sin motor de ejecución fuera de orden.

Ejercicio 5.4. Memoria Cache

Un microprocesador de 32 bits, tiene incorporado un subsistema de memoria cache Level 1 de 16 Kbytes de capacidad total. El subsistema trabaja en modo

asociativo de 4 vías con líneas de 16 Bytes de tamaño, agrupadas en sets de 8 líneas.

El microprocesador tiene 36 líneas de address hacia la memoria externa.

a. Dibuje el diagrama de organización del subsistema completo: Controlador cache – memoria cache – memoria del sistema, indicando claramente en cuantas páginas se divide ésta última.

b. Indique para una dirección física como se asignan dentro del cache los diferentes campos de bits para determinar si el acceso es un hit o un miss.

c. Donde se asigna dentro del cache la dirección física 0x7FF608EAC.

Ejercicio 5.5. Memoria Cache

Dado el siguiente código

```
while ( semáforo == TRUE)
{
      buffer [i] = buffer [i] * j;
      i++;
}
```

a. Señale donde se tiene vecindad espacial

b. Señale donde se tiene vecindad temporal

Esta Guia cumple con los contenidos del primer cuatrimestre solamente. La segunda parte estará disponible en el segundo cuatrimestre.