Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2343 – Arquitectura de Computadores (II/2018)

Tarea 7

Fecha de entrega: viernes 26 de octubre de 2018 a las 11:59 AM

Parte teórica

El objetivo de esta parte será diseñar, desde cero, el funcionamiento de un dispositivo IO, de forma que pueda ser utilizado en conjunto con un computador de arquitectura `x86-16`.

Hace poco se estrenó en el mercado el nuevo dispositivo *Germoji*, a través del que usted puede crear sus propios *emojis* animados y almacenarlos directamente en su computador para su uso en aplicaciones novedosas como *Missingir*. Este dispositivo se conecta con un computador `x86-16` mediante *memory mapping* y *port* I/O.

Cada *emoji* se compone de un máximo de 8 *frames* (*i.e.* imágenes) de 16x16 en color RGB. Para la transferencia el dispositivo envía, para cada pixel de cada *frame* del *emoji*, el valor de color en cada canal (R, G o B) a través de *buffers* de datos. Al ser tres canales, el dispositivo cuenta con tres *buffers* distintos y, como cada canal de color tiene un rango de valores de 0 a 255, cada pixel de una imagen usa 3 bytes (*i.e.* un byte para el valor de color de cada canal). Considere, además, que *Germoji* hace envío del *emoji* generado a través de transferencias de 3 *frames*. Un *frame*, en este contexto, se puede ver como tres arreglos (uno por cada canal de color) de largo 16x16, los que en conjunto contienen toda la información de una de las imágenes que componen al *emoji* animado. El dispositivo, en su interior, posee un registro que indica la cantidad de *frames* total a enviar. De esta forma, cada vez que termina una transferencia, este registro es actualizado según la cantidad de *frames* que haya sido enviada y, además, el dispositivo actualiza sus *buffers* con los bytes de las siguientes imágenes a enviar.

Por otra parte, suponga que posee dos dispositivos adicionales dentro de su computador:

- Una `DMA` (*Direct Memory Access*) que trabaja con *memory mapped* I/O, la que posee: registro de comandos, registro de dirección base, registro de dirección destino y registro de tamaño de copia. La `ISR` (*Interrupt Service Routine*) de este dispositivo se convoca a partir de la dirección `7h` del vector de interrupciones y se encarga solo de llevar a cabo la transferencia desde el dispositivo hacia el disco, por lo que la escritura de datos en sus registros debe ser realizada a través de código.

- Un disco duro (`HDD`) que posee: *buffer* de datos, registro de comandos que indica si debe leer o escribir, registro de dirección base y registro de tamaño de copia. Este es controlado por la `ISR` ubicada en la dirección `8h` del vector de interrupciones, la que según el valor de registro de comandos, se encarga de escribir directamente en el disco el contenido del *buffer* de datos. Para el almacenamiento de imágenes, se debe guardar de forma consecutiva el valor de cada canal de color de cada pixel según el orden RGB. El *buffer* posee el tamaño necesario para recibir, como máximo, una imagen.

Para este computador considere las siguientes direcciones reservadas, entre las que además se incluye el *mapeo* del vector de interrupciones y, por otra parte, los puertos utilizables:

Dirección	**Función/Contenido**
0-6	Interrupciones de *Hardware*
7	`ISR` transferencia `DMA`
8	`ISR` escritura en `HDD`
9-31	Interrupciones de *software* de uso libre
32-37	*Exception handlers*
38-805	*Buffer* Rojo *Germoji*
806-1573	*Buffer* Verde *Germoji*
1574-2341	*Buffer* Azul *Germoji*
2342-3109	*Buffer* lectura/escritura `HDD`
3110-65535	Memoria de uso libre

Tabla 1: Tabla de direcciones reservadas, donde 0-31 son las direcciones del vector de interrupciones.

Puerto	**Función**
0-5	Otros dispositivos del computador
6	Registro de estado *Germoji*
7	Registro de comandos *Germoji*
8	Registro de comandos `DMA`
9	Registro de dirección base `DMA`
10	Registro de dirección destino `DMA`
11	Registro de tamaño de copia `DMA`
12	Registro de comandos `HDD`
13	Registro de dirección base `HDD`
14	Registro de tamaño de copia `HDD`

Tabla 2: Tabla de puertos del computador.

Basándose en la descripción de estos dispositivos y sus conexiones:

- Realice un diagrama que muestre la comunicación de los dispositivos con la CPU y la memoria del computador `x86-16`. Debe incluir todos los elementos que considere necesarios para llevar a cabo la funcionalidad descrita. [**0.75 pts**]

- Diseñe una tabla de estados y comandos para utilizar el dispositivo *Germoji*, la `DMA` y el `HDD`. Considere que es usted el que define los estados y comandos a utilizar que le sean de utilidad para programar las `ISRs` solicitadas, así como los valores que tendrá cada uno de ellos. Las direcciones a utilizar también pueden ser escogidas por usted, pero no deben causar conflictos entre ellas. [**0.75 pts**]

- Haciendo uso de la tabla antes definida, escriba una `ISR` (*Interrupt Service Routine*) en *Assembly* `x86` que controle *Germoji* según el funcionamiento descrito al comienzo. Esta puede ubicarse en cualquier posición del vector de interrupciones que no esté siendo utilizada por la `ISR` de otro dispositivo. **[1.5 pts]**
- Escriba un programa en *Assembly* `x86` que escriba en el `HDD` un *emoji* (*i.e.* todos los *frames* del contenido) desde *Germoji* haciendo uso de la `DMA`. Debe hacer uso de las subrutinas de estos dispositivos para llevar a cabo la funcionalidad. **[1.5 pts]**
- Diseñe el microcontrolador de *Germoji* mediante un diagrama y especifique, en pseudocódigo, qué debe hacer este controlador. Sea explícito con sus supuestos e incluya todos los comentarios que sean necesarios para entender el funcionamiento de su diseño. **[1.5 pts]**

A considerar

Además de lo anterior, deben considerar lo siguiente:

- Un *emoji* tendrá un *frame* como mínimo.
- Puede definir todos los estados y comandos por dispositivo que estime convenientes, siempre que en conjunto puedan abarcar la funcionalidad antes descrita.
- La instrucción `IN Reg,(Port)` almacena en el registro `Reg` el valor del registro I/O asociado al puerto `Port`, mientras que `OUT (Port),Reg` realiza el proceso inverso.
- La instrucción `INT dir` en `Assembly x86` se encarga de ejecutar la `ISR` que se encuentra en la posición `dir` del vector de interrupciones. Debe hacer uso de ella para el programa pedido.
- La instrucción `IRET` se encarga de hacer el retorno de una `ISR`. Es decir, debe ser la última instrucción de una `ISR`, a diferencia del `RET` utilizado en otros programas de *Assembly* `x86`.
- El estándar para escribir una `ISR` entonces es como sigue:

```
;ISR ubicada en la posicion XYh del vector de interrupciones.
;Por ejemplo, ISR16 corresponde a la ISR ubicada en 16h.
ISRXY:
 ;Instrucciones que trabajan directamente con las direcciones
 ;utilizadas dentro de su dispositivo.
 ...
 ;Finalizado este manejo, se hace uso de la instruccion
 ;IRET para retornar
 IRET
;Programa que hacer uso de la ISR.
program:
 ;Instrucciones tipicas.
 ...
 ;Luego, llamado a la ISR 16h.
 INT 16
 ...
 ;Mas instrucciones tipicas.
 ;Finalizamos con RET.
 RET
```

Supuestos

Puede hacer uso de **supuestos** en casos límite o elementos que no hayan sido explicitados en el enunciado. No obstante, para que estos sean válidos durante la corrección, **debe** explicitarlo en su `README`.

> **No se aceptarán supuestos sobre criterios que hayan sido establecidos en el enunciado.**

Parte práctica

Objetivo

Implementar en el *hardware* del computador básico un *Address decoder* para trabajar con las luces *led* y los *switches* de forma directa con direcciones y, además, dar soporte para la conexión con los pulsadores y *display* de 7 segmentos mediante *port* I/O.

Descripción de la actividad

Para esta tarea deberá implementar sobre el computador básico visto en clases, es decir, la versión Harvard, los mecanismos de acceso a *Input/Output*. En particular, deberá implementar:

- Comunicación *memory mapped* I/O con un *Address decoder* programable.
- Comunicación *port* I/O.

Si lo desea, puede usar como base el proyecto `basic computer 2`. Tendrá libertad sobre el cómo implementar estas características en su computador básico, respetando la ISA entregada más adelante.

Memory Mapped I/O

Los dispositivos I/O que se usarán para esta parte son los *switches* y *leds*, para un total de 32 dispositivos.

Dado que el *Address decoder* será programable, es necesario reservar espacio en memoria para utilizar las operaciones. Se reservarán permanentemente las direcciones 192 a 255 de la memoria de datos para este cometido. Notar que son 64 espacios, no 32.

Para definir el mapeo de memoria se usaran dos palabras para cada dispositivo. El formato será una primera palabra para identificar la dirección de memoria que será ocupada por el *input/output* y una segunda palabra para identificar el *puerto físico* de conexión del dispositivo, es decir, identificar a qué dispositivo se hace referencia. La asignación de puertos para esta parte es la siguiente:

Dispositivo	Puerto
led_0	0
...	...
led_{15}	15
$switch_0$	16
...	...
$switch_{15}$	31

Tabla 3: Puertos del *Address decoder* con su entrada en la tabla de redirección.

Port I/O

Los dispositivos I/O que se usarán para esta parte son los botones y *diplays*, para un total de 9 dispositivos. La asignación de puertos es la siguiente:

Dispositivo	Puerto
d_btn$_0$	32
d_btn$_1$	33
d_btn$_2$	34
d_btn$_3$	35
d_btn$_4$	36
dis_a	37
dis_b	38
dis_c	39
dis_d	40

Tabla 4: *Port* I/O.

ISA

Instrucciones de carga, aritméticas y lógicas

Instrucción	Operandos	Operación	*Opcode*
MOV	A, B	A=B	00000000
	B,A	B=A	00000001
	A,Lit	A=Lit	00000010
	B,Lit	B=Lit	00000011
ADD	A,B	A=A+B	00000100
	B,A	B=A+B	00000101
	A,Lit	A=A+Lit	00000110
SUB	A,B	A=A-B	00000111
	B,A	B=A-B	00001000
	A,Lit	A=A-Lit	00001001
AND	A,B	A=A and B	00001010
	B,A	B=A and B	00001011
	A,Lit	A=A and Lit	00001100
OR	A,B	A=A or B	00001101
	B,A	B=A or B	00001110
	A,Lit	A=A or Lit	00001111
NOT	A,A	A=not A	00010000
	B,A	B=not A	00010001
	A,Lit	A=not Lit	00010010
XOR	A,B	A=A xor B	00010011
	B,A	B=A xor B	00010100
	A,Lit	A=A xor Lit	00010101
SHL	A,A	A=shift left A	00010110
	B,A	B=shift left A	00010111
	A,Lit	A=shift left Lit	00011000
SHR	A,A	A=shift right A	00011001
	B,A	B=shift right A	00011010
	A,Lit	A=shift right Lit	00011011

Instrucciones de salto y comparación

Instrucción	Operandos	Operación	Condición	*Opcode*
CMP	A,B	A-B	-	00011100
	A,Lit	A-Lit	-	00011101
JMP	Dir	PC = Dir	-	00011110
JEQ	Dir	PC = Dir	Z=1	00011111
JNE	Dir	PC = Dir	Z=0	00100000
JGT	Dir	PC = Dir	N=0 y Z=0	00100001
JLT	Dir	PC = Dir	N=1	00100010
JGE	Dir	PC = Dir	N=0	00100011
JLE	Dir	PC = Dir	Z=1 o N=1	00100100
JCR	Dir	PC = Dir	C=1	00100101
JOV	Dir	PC = Dir	V=1	00100110

Instrucciones de memoria y direccionamiento

Instrucción	Operandos	Operación	*Opcode*
MOV	A,(Dir)	A=Mem[Dir]	00100111
	B,(Dir)	B=Mem[Dir]	00101000
	(Dir),A	Mem[Dir]=A	00101001
	(Dir),B	Mem[Dir]=B	00101010
	A,(B)	A=Mem[B]	00101011
	B,(B)	B=Mem[B]	00101100
	(B),A	Mem[B]=A	00101101

Instrucciones adicionales

Instrucción	Operandos	Operación	*Opcode*
INC	B	B=B+1	00101110
NOP	Ninguno	Nada	11111111

Instrucciones I/O

Instrucción	Operandos	Operación	*Opcode*
IN	A, (Port)	A=Input[Port]	00101111
IN	B, (Port)	B=Input[Port]	00110000
OUT	(Port), A	Output[Port]=A	00110001
OUT	(Port), B	Output[Port]=B	00110010

Finalmente, deberá adjuntar a su proyecto un documento en formato **PDF** que contenga el esquema del computador básico resultante, el esquema detallado de los componentes de su *address decoder* y una explicación de sus decisiones de diseño y el funcionamiento de su unidad. El nombre de dicho archivo debe ser `esquema.pdf`

A considerar

Además de los anterior, deben considerar lo siguiente:

- Pueden usar el bloque `with/select` y `process`.

Supuestos

Puede hacer uso de **supuestos** en casos límite o elementos que no hayan sido explicitados en el enunciado. No obstante, para que estos sean válidos durante la corrección, **debe** explicitarlo en su `README`.

No se aceptarán supuestos sobre criterios que hayan sido establecidos en el enunciado.

ProTips de Vivado

Si es la primera vez que usa Vivado, se recomienda revisar los tutoriales del `Syllabus/Vivado`, especialmente el tutorial 2 y el archivo `intro_vhdl.pdf`. Si siguen teniendo dudas, pueden solicitar una reunión presencial (estas posibilidades están sujetas a la disponibilidad de los ayudantes).
Además, pueden revisar en Vivado el diagrama de su circuito usando lo siguiente:

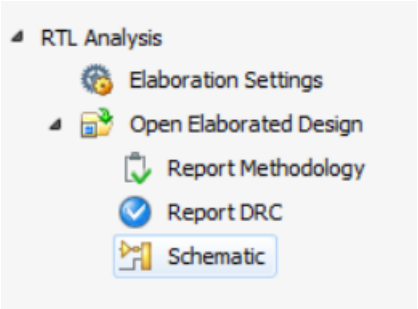

Figura 1: Menú de selección para ingresar al *Schematic*.

Verán un esquema que muestra las instancias de sus componentes y las conexiones entre ellos:

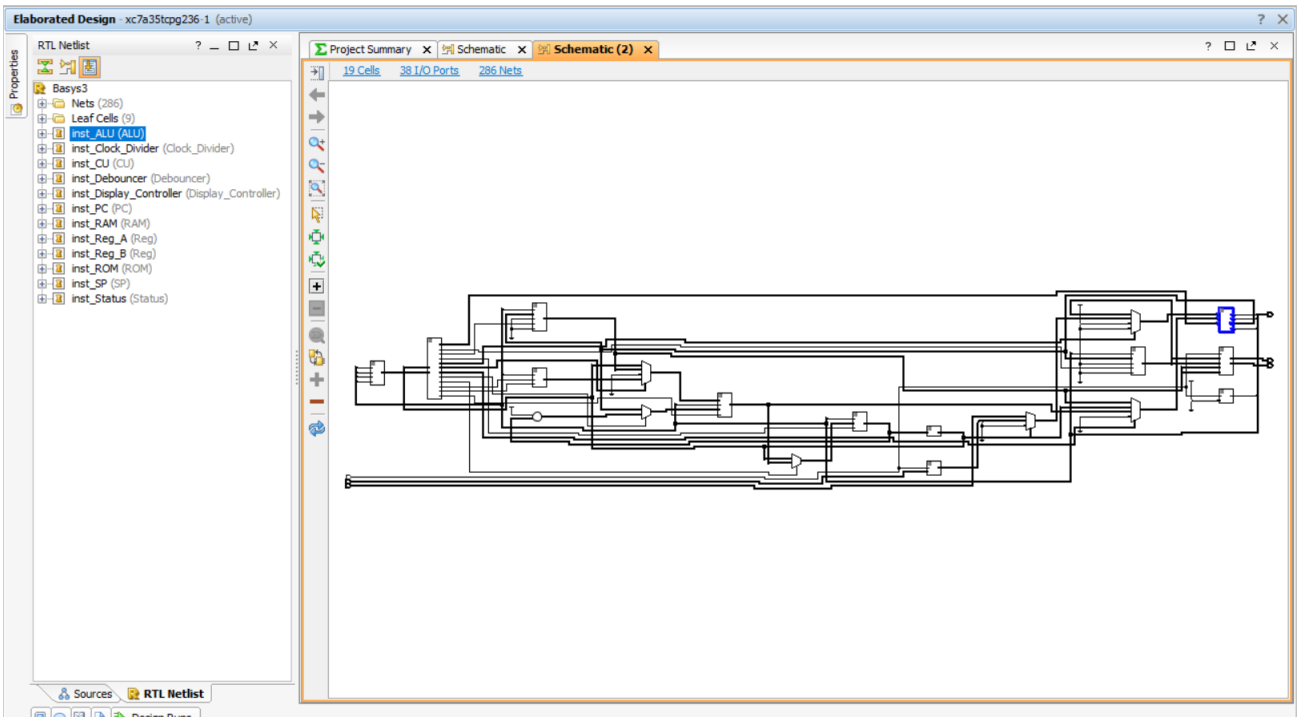

Figura 2: *Schematic* del computador básico.

Entrega y evaluación

Parte teórica

La tarea se debe realizar de **manera individual** y la entrega se realizará a través de GitHub. El formato de entrega corresponde a un archivo llamado `tarea7.tex`, junto con los recursos necesarios para que este pueda ser compilado correctamente. Puede incluir, además, el archivo `PDF` resultante de la compilación del `.tex`.

Parte práctica

La tarea se debe realizada por los **grupos asignados** y la entrega se realizará a través de GitHub. El repositorio debe contener una carpeta con su proyecto de Vivado, el archivo `.bit` y el documento **PDF** llamado `esquema.pdf` con la explicación y diagramas de lo realizado. En el caso de la carpeta del proyecto, deben subir **solo** la carpeta `basic_computer.srcs` y el archivo `basic_computer.xpr`. **Considere que esta parte incluirá una evaluación de pares, que será detallada durante la entrega.**

Independiente de la parte trabajada en esta tarea, el repositorio subido debe contar con un archivo `README.md` escrito en *Markdown* que identifique sus datos y consideraciones que el corrector deba tomar en cuenta, tales como el sistema operativo usado para realizar la tarea. El `README` se puede subir hasta 24 horas después de la entrega. Si lo sube posterior al plazo de entrega establecido, debe crear una *issue* en el repositorio de su tarea indicándolo para que sea considerado en la corrección. Los archivos que no ejecuten o que no cumplan el formato de entrega establecido implicarán nota **1.0** en la tarea. En caso de atraso, se aplicará un descuento de **1.0** punto por cada 6 horas o fracción.

Política de Integridad Académica

Los alumnos de la Escuela de Ingeniería deben mantener un comportamiento acorde al Código de Honor de la Universidad:

"Como miembro de la comunidad de la Pontificia Universidad Católica de Chile me comprometo a respetar los principios y normativas que la rigen. Asimismo, prometo actuar con rectitud y honestidad en las relaciones con los demás integrantes de la comunidad y en la realización de todo trabajo, particularmente en aquellas actividades vinculadas a la docencia, el aprendizaje y la creación, difusión y transferencia del conocimiento. Además, velaré por la integridad de las personas y cuidaré los bienes de la Universidad."

En particular, se espera que mantengan altos estándares de honestidad académica. Cualquier acto deshonesto o fraude académico está prohibido; los alumnos que incurran en este tipo de acciones se exponen a un procedimiento sumario. Específicamente, para los cursos del Departamento de Ciencia de la Computación, rige obligatoriamente la siguiente política de integridad académica. Todo trabajo presentado por un alumno (grupo) para los efectos de la evaluación de un curso debe ser hecho individualmente por el alumno (grupo), sin apoyo en material de terceros. Por "trabajo" se entiende en general las interrogaciones escritas, las tareas de programación u otras, los trabajos de laboratorio, los proyectos, el examen, entre otros. Si un alumno (grupo) copia un trabajo, los antecedentes serán enviados a la Dirección de Docencia de la Escuela de Ingeniería para evaluar posteriores sanciones en conjunto con la Universidad, las que pueden incluir reprobación del curso y un procedimiento sumario. Por "copia" se entiende incluir en el trabajo presentado como propio partes hechas por otra persona. Está permitido usar material disponible públicamente, por ejemplo, libros o contenidos tomados de Internet, siempre y cuando se incluya la cita correspondiente.