Trabajos Prácticos para entregar
Comisión 2
V4

Organización de Computadoras 2020

Universidad Nacional de Quilmes

Introducción

El **pacman** es un videojuego que consiste en comer monedas que les permite a lxs jugadorxs acumular puntos. Mientras se mueve, el pacman debe huir de los fantasmas, pero si sus esfuerzos son en vano y es alcanzado por uno de ellos, entonces pierde una vida. Cuando la cantidad de vidas llega al valor cero entonces el juego ha finalizado.

La persona que juega utiliza las flechas del teclado para mover el avatar por el espacio de juego, limitado por las paredes del tablero.

Sistema de puntaje:
Cuando el pacman come una moneda se suman 20 puntos. A su vez, los casilleros también pueden tener premios especiales, que al ser comidos, suman 50 puntos. Tener en cuenta, que un casillero del laberinto puede contener tanto una moneda como un premio especial (*bonus*). Por lo tanto, en caso que el casillero tenga ambas fichas, el puntaje final se actualizará con la suma de ambos. Al acumular 1024 puntos se suma una vida y se reinicia el puntaje.

1. Entrega 1

1.1. Coordenadas del Pacman (Sistema de numeración)

La posición del pacman en el espacio del tablero se representa mediante dos coordenadas, denominadas **X** e **Y**, que están representadas en un sistema BSS(5). Para la representación del estado del juego se utiliza el formato `PacStatus` de **32 bits** que se estructura de esta manera:

X (5b)	Y(5b)	W(0000)	V (2b)	N (6b)	P (10b)

1. El campo X representa el valor de la coordenada **X**,
2. El campo Y representa el valor de la coordenada **Y**,
3. El campo W está reservado para uso futuro y se completa con un relleno `0000`,
4. El campo V representa la cantidad de **vidas** que le quedan,
5. El campo N representa el **nivel** actual,
6. El campo P representa la cantidad de **puntos** que acumuló.

Esta información normalmente se encuentra comprimida en hexadecimal, y en ese caso se la denomina `HexaStatus`.

(a) ¿Qué tamaño puede tener el tablero?

(b) ¿Cuál es la cantidad máxima de vidas que pueden tenerse?

(c) Dar 2 ejemplos de datos en formato **PacStatus** que reflejen una posible representación del juego. Justificar explicando que valor representa cada campo.

(d) Si se quiere duplicar el tamaño del tablero ¿Cómo se debe modificar el sistema de representación para lograr este propósito?

(e) Representar el **estado inicial** del pacman en el centro del tablero, con la máxima cantidad de vidas, en el primer nivel y sin puntos acumulados, con el formato **PacStatus**.

(f) Comprimir la cadena anterior en el formato **HexaStatus**.

1.2. Puntaje del pacman (Aritmética)

Teniendo el cuenta el objetivo del juego, responder las siguientes preguntas en base a las consideraciones planteadas en cada caso:

1. Suponiendo que en este momento, el estado del pacman en formato hexaStatus es `3A83 1D55`, ¿cómo queda el hexaStatus luego de descontar una vida?

2. Suponiendo que en este momento, el estado del pacman en formato hexaStatus es `3A82 1FFF`, y el pacman come otra moneda ¿cómo queda el hexaStatus luego de incrementar el puntaje?

1.3. Init (código fuente y código máquina)

El estado de los 4 fantasmas se representa en los 4 registros R0, R1, R2 y R3 respectivamente, con el formato `GhostStatus`: `000000xxxxxyyyyy`, donde los bits `xxxxx` representan la coordenada **X** y los bits `yyyyy` representan la coordenada **Y** de cada fantasma. El estado del juego se almacena en los registros R5 y R6.

1. Escribir en Q1 un programa `init` que inicialice el estado del juego teniendo en cuenta la representación del inciso e del ejercicio 1.1 y la posición de los 4 fantasmas, ubicándolos en cada una de las esquinas del tablero.

2. Ensamblar el programa.

1.4. Desplazar2d (Diseño de circuitos)

El Pacman puede moverse en sólo 4 direcciones (N,S,E,O) (no puede moverse de manera diagonal), y un sólo casillero a la vez en función de la flecha que se haya oprimido en el teclado (ver ejemplos en la figura 1). Se necesita un circuito `desplazar2d` que permita moverse en la dirección deseada.

Es decir que, dada una cadena de 10 bits que representan las coordenadas X e Y del pacman, calcule las nuevas coordenadas luego de moverlo hacia la nueva posición según las líneas de control: $A/\overline{R}$(Avanzar/Retroceder) [1] y $H/\overline{V}$ (Horizontal/Vertical).

Asumir como precondición que el tablero siempre le permite desplazarse, es decir que la posición deseada también se encuentra dentro de los límites del tablero.

1. Dibujar la caja negra del circuito (ver libro) indicando el nombre, las entradas y salidas necesarias. Explicar qué representa cada entrada y cada salida en el contexto del juego.

2. Construir el circuito `desplazar2d`. Para hacerlo cuentan con una caja de herramientas con diferentes circuitos. Deben usar los necesarios.

[1] La notación $\overline{R}$ indica que el valor 0 en la entrada representa el valor remarcado (en este caso Retroceder)

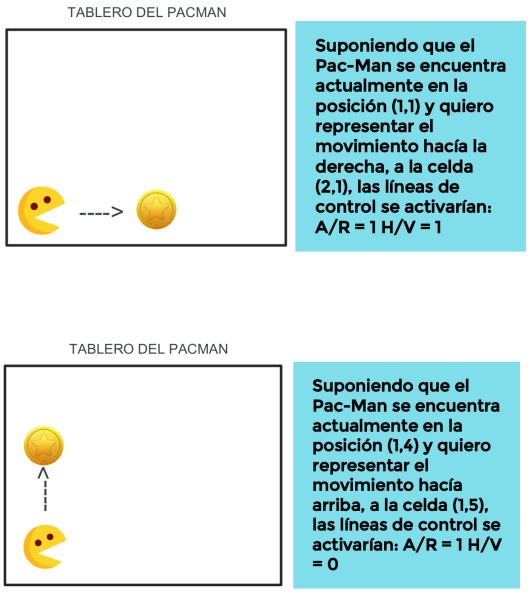
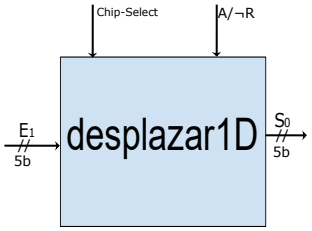


Figura 1: Ejemplos de movimiento del pacman

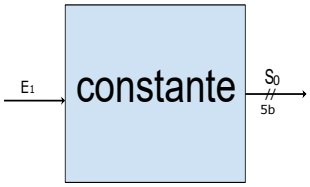
3. Construir el circuito `desplazar1D` siguiendo la especificación que se describe en la caja de herramientas.

Caja de Herramientas

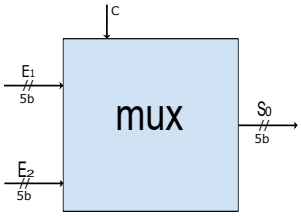
`desplazar1D`
Si la línea chip-select es 0, el circuito es neutral (la salida es igual a la entrada). Si la línea A/R es 1 y chip-select es 1: se suma 1 a la cadena de entrada. Si A/R es 0 y chip-select es 1: se resta 1 a la cadena de entrada.



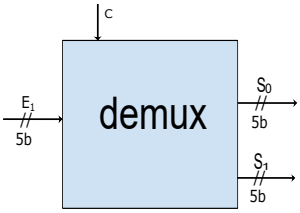
`constante`
Si la entrada e es 0 se construye la constante `00000`. Si en cambio es 1 se construye la constante `00001`



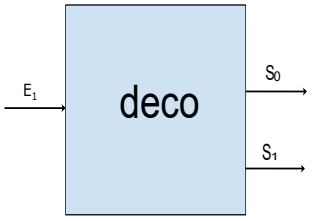
`mux`
Proyecta en la salida una de las 2 entradas, según lo que se indique en la línea de control c.



`demux`
Proyecta la entrada en alguna de las dos salidas, según lo que se indique en la línea de control c.



`Deco`
Enciende una sola de las dos salidas en función de la entrada



2. Entrega 2: Programación con rutinas y condicional. Sistemas de numeración enteros

2.1. Subsistema de interconexión y ciclo de ejecución

A partir del siguiente estado de memoria:

	...
4003	1940
4004	8403
4005	1980
4006	0400
4007	1800
4008	0000
4009	1840
400A	03E0
400B	1880
400C	001F
400D	18C0
400E	03FF
400F	C000
	...
FFEF	0500
	...

Responder y justificar las siguientes preguntas

1. Elaborar un cuadro de estado de buses (ver sección 6.3.2 del libro) para cada una de las instrucciones del programa, a partir de la celda 4003 y hasta la celda 400E.

2. ¿Cuál es el valor de los registros de **uso general** (visibles a quien programa) al finalizar la ejecución de las instrucciones anteriores?

3. Explicar detalladamente la dirección, el dato y la señal de control de alguna de las filas del cuadro (elegir una de ejemplo).

4. ¿Qué habría que modificar en el programa para que se realice una escritura?

2.2. Simular y analizar ejecución

Para comprender lo que ocurre en cada etapa de ejecución de un programa, se realizan una **simulación** (ejecución paso a paso) del mismo. A partir del estado de memoria descripto mas arriba, y asumiendo que el valor original del registro **SP** es FFEE y de **PC** es 4003, **elaborar un cuadro de simulación** (ver estructura en la sección 7.3 del libro) a partir de la ejecución del programa.

Responder y justificar las siguientes preguntas

1. ¿En qué momento se actualiza el registro SP y con qué objetivo?
2. ¿Qué ocurre con la celda de memoria `FFEF` al finalizar la simulación?
3. ¿Es posible saber en qué celda esta la instrucción siguiente a la última simulada?
4. ¿Cuántos elementos tiene la pila antes de comenzar la simulación?
5. ¿Cuántos elementos tiene la pila luego de terminar la simulación?
6. ¿Cuál es el valor de los registros de **uso específico** (no visibles a quien programa) al finalizar la simulación de ejecución?

2.3. Documentar

1. Documentar la rutina del ejercicio 1.3
2. ¿Tiene parámetros?
3. De manera general ¿Cómo se relaciona el campo requiere con la presencia/ausencia de parámetros?

2.4. Movimiento del pacman (Programación)

Para codificar los movimientos se utiliza la codificación **MoveCode** de 16 bits, con el formato: `0000 0000 0000 00HF`, donde `H` (Horizontal/Vertical) indica si debe haber movimiento horizontal (si vale 0) o bien vertical (si vale 1), y `F` (Foward/Backward) indica si avanza (con el valor 1) o retrocede (con valor 0).

Suponer que se cuenta (no hay que programarlas) con las rutinas `nextToAWall`, `getCoord` y `movePacman` que respetan las siguientes documentaciones:

`nextToAWall`	
`Requiere`	Una coordenada **X en R1**, una coordenada **Y en R2**. Un **MoveCode en R3** que indique en qué dirección y sentido analizar.
`Modifica`	R0
`Retorna`	Un 1 en R3 si el casillero adyacente a (X,Y), en la dirección indicada, está libre, o un 0 si hay un muro

`getCoord`	
`Requiere`	El estado del pacman (en el formato **PacStatus**) en los registros R5 y R6
`Modifica`	R0
`Retorna`	La coordenada X en R1 y la coordenada Y en R2

movePacman	
Requiere	El estado del pacman (en el formato **PacStatus**) en los registros R5 y R6. Un **MoveCode en R3** que indica dirección y sentido del movimiento.
Modifica	R0
Retorna	El **nuevo** estado del pacman (actualizado con la nueva posición) en los registros R5 y R6, en la dirección y sentido solicitados.

Escribir la rutina `checkAndMove` haciendo uso de la rutina anterior, para cumplir con el objetivo descripto en la siguiente documentación:

checkAndMove	
Requiere	El estado del pacman (en el formato **PacStatus**) en los registros R5 y R6. La dirección del movimiento en formato **MoveCode en R3**.
Modifica	??
Retorna	El **nuevo** estado del pacman (actualizado con la nueva posición) en los registros R5 y R6, si es posible mover el pacman en *la dirección solicitada*.

2.5. Reverse Shift2D (Sistemas enteros)

Se necesita rediseñar el circuito **Shift2D** (ver sección 1.4) porque no es posible conseguir circuitos restadores. Para lo cual se pide:

1. Rediseñar circuito sin utilizar los mencionados restadores
2. Probar el circuito con 3 situaciones / ejemplos concretos (datos que representan cada estado del pacman).

2.6. Validar la rutina getCoin (Test)

El contenido de un **casillero del laberinto** se representa en 16 bits con el siguiente formato: `0000000000000LBCP`. Donde, el bit `L` indica si se trata de un casillero que se encuentra en el límite del tablero. `B` indica si el casillero contiene un premio especial (bonus) que permite acumular 50 puntos, el bit `C` indica si el casillero contiene una moneda que permite acumular 20 puntos y el bit `P` indica si se trata de un muro.

getCoin	
Requiere	El contenido de un casillero del tablero en el registro R4, el estado del pacman en los registros R5 y R6
Modifica	la celda A000
Retorna	El estado del pacman actualizado al comer la moneda del casillero, en caso de existir una

Se necesita validar la rutina `getCoin`, para lo cual se les pide escribir tres **rutinas de test** (¡ver libro!)

1. Una rutina de test `testGetCoinSuccessfully` para el caso de un casillero que contenga una moneda pero no tenga bonus.

2. Una rutina de test `testGetBonusSuccessfully` para el caso de un casillero que contenga un premio especial y una moneda

3. Una rutina de test `testNoPrize` para el caso de un casillero vacío.

2.7. Ensamblar test

Ensamblar la rutina `testGetCoinSuccessfully` (del ejercicio anterior) a partir de la celda `7001`, asumiendo que la rutina `getCoin` está ensamblada a partir de `0A0A`.

3. Entrega 3: Programación Avanzada y Sistemas de numeración fraccionarios

3.1. Programar getScore (Uso de máscaras)

Escribir la rutina `getScore` en base a la siguiente documentación y actualizar el campo **Modifica** de la misma.

getScore	
`Requiere`	El estado del pacman (en el formato **PacStatus**) en los registros R5 y R6
`Modifica`	**COMPLETAR**
`Retorna`	El puntaje actual en el registro R4

3.2. Programar MaxScore (Reuso)

Escribir la rutina `MaxScore` en base a la siguiente documentación y actualizar el campo **Modifica** de la misma.

MaxScore	
`Requiere`	Dos estados de pacman (en el formato **PacStatus**) desde las celdas `0000` a `0003` (32 bits cada uno)
`Modifica`	**COMPLETAR**
`Retorna`	El estado con mayor puntaje distribuido en los registros R5 y R6. En caso de que tengan el mismo puntaje, se retorna el primero.

3.3. Programación: theHighestScore (Recorrido de arreglos)

Listado de puntajes ordenados en forma cronológica. Buscar el puntaje más alto (agregar en la práctica un ejercicio de una rutina que retorne el máximo entre 2 valores numéricos).

theHighestScore	
`Requiere`	Un arreglo con un listado de puntajes con formato **PacStatus** (32 bits) ordenado de forma cronológica, cuya dirección inicial se recibe en la celda `5566` y cuya longitud se recibe en la celda `FA56`
`Modifica`	**COMPLETAR**
`Retorna`	El puntaje más alto del arreglo distribuido en las celdas `5563` y `5564`

3.4. Coprocesador 8087

Investigar el coprocesador 8087 para analizar:

1. ¿Que motivó su diseño?
2. En cuanto a su arquitectura ¿Qué aporte brindó este nuevo procesador en comparación al Intel 8086?
3. Dar una cadena de ejemplo en el sistema de punto flotante (IEEE 754) que soportaba (de precisión simple). Interpretar dicha cadena para conocer su valor.

3.5. Caché

Suponer la siguiente arquitectura de caché:

- La cantidad de líneas es 2,
- El tamaño de bloque de 4 celdas,
- Su función de correspondencia es asociativa,
- Su política de reemplazo FIFO,
- Su política de escritura es write-back.

Considerar la ejecución simulada en la sección 2.2, completar la siguiente tabla:

Dirección	Tag	Línea	Palabra	F/A	Etapa