Universidad del Valle de Guatemala
Organización de computadoras y Assembler
Sección 10
Martha Ligia Naranjo

Guatemala, 26 de abril del 2019
Proyecto 3 de diseño
Gustavo Méndez 18500
Diego Estrada 18540
Temario: No. 3: Cuatro en línea

Proyecto No.3 de Diseño: Análisis

Uso de registros:

Para poder realizar el juego fue imprescindible conocer y determinar los registros a utilizar. Estos fueron utilizados tanto para realizar impresiones como para guardar valores de variables. A continuación se presenta una síntesis de los registros más usados y la función que cumplieron en el programa del juego *4 en Línea*:

R0 y R1→ Utilizados para:

Cargar en él los formatos de impresión de mensajes, tanto en el main como en la subrutina.

Además, es utilizado para pasar valores como el *player* (1,2) como parámetro a la subrutina desde el main. Ejemplo: en el *insertInput* de la subrutina, es imprescindible recibir el número del jugador con el fin de llenar el espacio en la matriz. También para la impresión del mensaje ganador, se imprime el formato de ganador más el dato almacenado en R1.

R2-R5 → Utilizado para:

Variables de las columnas de la matriz 4x4 dentro de la función de *getWinner,* en la cual se realiza la comparación de estados y se obtiene el jugador que ha ganado la partida. .Realizado de la siguiente manera:

```
column1 .req r2
column2 .req r3
column3 .req r4
column4 .req r5
```

De esta manera se le da un “Alias” a cada registro. Uso de *.req* [1]

R6-R9 → Utilizado para:

Variables de las filas de la matriz 4x4 dentro de la función de *getWinner,* en la cual se realiza la comparación de estados y se obtiene el jugador que ha ganado la partida. Realizado de la siguiente manera:

```
row1 .req r2
row2 .req r3
row3 .req r4
row4 .req r5
```

De esta manera se le da un “Alias” a cada registro. Uso de *.req* [1]

R10 → Utilizado para:

Variable que contendrá el valor del ganador, este valor es un entero cuyo valor puede ser 1 o 2, representando el número de jugador homónimo. Posteriormente el valor que almacena el registro será pasado de nuevo al programa *main* para realizar la impresión del ganador y terminar el programa.

R11 → Utilizado para:

Contador iniciado en 0, el cual se utiliza en la verificación horizontal de la matriz, ya que son cuatro filas, se realizará la verificación Horizontal que se encuentra en el algoritmo narrativo mientras este contador sea menor a 4.

Algoritmo narrativo

Para la ejecuciòn del juego se utilizarà un programa principal denominado *main.s*, dentro del cual se referenciarà y utilizará una subrutina llamada *game.s,* donde se almacenan funciones que manejan el flujo del programa.

Proceso de input

a. Para la subrutina *game.s*, se plantea un algoritmo como el siguiente:
    1. Se crean cuatro listas de nùmeros enteros, cada una llamada columna#, se ve de esta manera: columna1 = [0,0,0,0][3]. Estas listas forman una columna de la matriz 4x4 que representa el tablero del juego.
    2. La variable llamada *currentColumn* se empieza en 0, y será la encargada de controlar y manejar las inserciones.
        a. input
            i. Se le asigna al registro r5 el alias de “player”
            ii. Se le asigna al registro r6 el alias de “columna”
            iii. Realizar un push en la pila del registro actual
            iv. Por último se realiza un *mov* de r0 al *player*, de manera que es pasado como un parámetro para siguientes instrucciones.
        b. Cuerpo del Input
            i. Se realiza un *load* del mensaje del Input en el registro r0, con el fin de ser mostrado posteriormente en un print.

ii. Se guarda en el r1 la variable *player*, de manera que servirá para indicar en el display de la consola el jugador que debe de realizar el input.
iii. Se imprime mediante un printf.
iv. Ahora para proceder con el input, se cargan el formato de entrada en r0 se solicitando el input del usuario y en r1 se carga la dirección de la columna actual para luego ser almacenado el input.
v. Mediante un branch *bl* se realiza la instrucción scanf para pedir y almacenar la columna del usuario de turno.
vi. Usando un *load*, se almacena el input en el registro 0, para posteriormente realizar otro load y guardar el valor del input que se encuentra en r0 en la variable *columna*.
vii. Una vez ingresado la columna, verifica si ésta es menor que 1 o mayor que 4, es decir una columna que no existe (input inválido), mediante un cmp y valores inmediatos #1 y #4. Luego se realiza un *branch less than (blt)* y *branch greater than (bgt)*, para que en dado caso suceda, se realiza lo siguiente:
    1. Se carga con *ldr* en r0 el formato del mensaje de error para posteriormente mostrarlo.
    2. Se imprime con *bl printf*.
    3. Regresa al *Cuerpo del Input*

viii. Si se encuentra en el rango de 1-4, entonces se realiza lo siguiente:
    1. Se mueve el valor de la *columna* al registro r0 (*mov*).
    2. Se desenlaza con *unreq* las variables *player* y *column* para poder proceder a la inserción y luego cambiar de jugador.
    3. Hacer pop {lr} a la pila para recuperar el registro en el que se encontraba.
    4. Se realiza un *mov* para recuperar el registro de *lr* a *pc*.

Inserciones de juego

c. Inserciones de input:
    i. Se empieza por declarar las variables al colocar alias a los registros de r5 a r9. Estas variables serán: *columna*, *box*, *player*, *count* y *columnInput*.
    ii. Se hace push a la pila del registro actual.
    iii. Se obtienen ahora los datos del *Cuerpo del Input* como parámetros. Se almacena con *mov* el *player* en r0 y *column* en r1.
    iv. Mediante un *switch case*, se realizará la verificación y asignación de la columna según el input del usuario; todo esto usando *cmp* con valores inmediatos 1-4, luego de cada

comparación, se asigna la *columna* ingresada por el usuario a las columnas 1-4 establecidas al inicio de la subrutina:

1. ejemplo: if(columna == 1):
   columna = columna1

v. trasladarse al proceso ***Loop Vector***

d. Loop Vector:

i. Cargar el valor de la columna actual a la variable *box*, se comprueba si esta está vacía, de ser así, almacena el *player* en dicho elemento.

ii. Si fue así, se realiza lo siguiente:

1. Se almacena la columna actual en r0
2. Desenlazar las variables con *unreq*

iii. De lo contrario, pasar al siguiente elemento de la columna elegida y sumar 1 al contador, cuyo valor máximo es 4, puesto que solo hay cuatro elementos en la columna. Repetir el proceso de Loop Vector mientras el contador sea diferente de 4.

iv. Si todas están ocupados o no vacíos, entonces se despliega el mensaje de columna llena.

Impresión del tablero

e. Se establecen las variables de las columnas. Se crea un contador y se inicia en 4.

f. Se cargan en cada una de las variables anteriores el formato

g. Ya que el juego debe de agregar la "ficha" hasta el fondo de la columna, se revierte la matriz, por lo que se empieza por un espacio más del último elemento, por lo tanto se suma un valor inmediato de #16.

h. Hacer un ciclo[2] en el cual se impriman los valores de cada columna, formando una fila, luego de los 4 elementos de la fila, se imprime un "enter" al cargar en r0 un formato de salto de línea. Antes de cada impresión se debe realizar la resta de #4 a cada columna para acceder al siguiente elemento (Va de atrás para adelante, por eso la resta). Se resta 1 al contador. Se repite esto mientras el contador sea diferente de 0.

i. Desenlazar las variables.

Verificación de ganador

a. Para representar los estados de victoria, el 0 significa que no hay ganador, 1 para el jugador 1 y 2 para el jugador 2.

b. Se declaran las variables para elementos de cada columna, para cada fila, una variable *ganador*, la cual tendrá los estados (0,1 o 2), y un contador.

c. Cargar en las variables creadas los formatos de las columnas. Iniciar el contador en 0

d. Verificación horizontal:

i. Cargar en las variables de fila los elementos de las columnas, de manera que se obtienen los cuatro elementos de una fila entera.
ii. Se compara si el primer elemento de la fila es igual al segundo, si se cumple, comparar el tercer elemento con el cuarto, si se cumple, comprobar si el primer elemento es igual al cuarto. Si todo esto se cumple, almacenar con *mov* el valor del elemento de la fila en la variable de *ganador*.
iii. Si *ganador* es distinto de 0:
    1. Se almacena en r0 el ganador
    2. Desenlazan las variables.
iv. Si es igual a 0:
    1. Sumar #4 a cada variable de columna para pasar a los elementos de la siguiente fila.
    2. Sumar 1 al contador.
v. Repetir mientras el contador sea menor a 4.

e. <u>Verificación vertical</u>:
    i. Cargar en las variables de elementos de fila, los cuatro elementos de una sola columna, haciendo ldr a la variable row del elemento de una sola columna, luego sumarle #4 a la columna para acceder al siguiente elemento y volver a hacer la carga, de manera que row 1-4 tengan los valores de una columna entera.
    ii. Se compara si el primer elemento de la columna es igual al segundo, si el tercero es igual al cuarto y si el primero es igual al último. Si esto se cumple, almacenar con mov el valor del elemento de la columna en la variable ganador (recordar que 1 es para el jugador 1 y 2 para el jugador 2).
    iii. Si el ganador es distinto de 0:
        1. Almacenar dato en r0
        2. Desenlazar variables
    iv. De lo contrario repetir el proceso solo que ahora con los valores de la columna 2 y así sucesivamente hasta tener las 4 columnas.
f. <u>Verificación de diagonales:</u>
    i. Cargar en las variables de columna sus respectivas listas de caracteres.
    ii. Pendiente negativa:
        1. En la variable *row1*, cargar el valor del primer elemento en la llista de la columna 1, en *row 2*, cargar el valor del segundo elemento (+ #4) en la lista de columna 2 , en *row3*, cargar el valor del tercer elemento en la lista de columna 3 (+ #8) y en *row4* cargar el valor del cuarto elemento en la lista de columna 4.
        2. Si los cuatro valores de *row1-4* son iguales, almacenar el valor de row en la variable *ganador*.

3. si el ganador es distinto de 0:
    a. Almacenar dato en r0
    b. Desenlazar variables.
4. De lo contrario, saltar a la comprobación de diagonales con pendiente positiva.

iii. Pendiente positiva:

1. Lo mismo que la pendiente negativa, unicamente que los valores de *row1-4* empiezan por columna 1 + #12 y terminan en columna 4.

Para el archivo *.main*

a. En el programa del main se utilizarán los ciclos creados en la subrutina y programa de *game.s*.
b. En el espacio del *.data* se colocan los formatos de impresión para solicitar los inputs al usuario, también se colocan los mensajes de bienvenida para realizar el ASCII art. También se utiliza *.word* para crear un contador de empate llamado *tieCounter.*
c. En el área del *.text* se empieza por colocar las variables *winner*, *contador*, *tieCounter*, usando *.req*, y se le asigna a un registro específico cada variable.
d. Desplegar el mensaje ASCII de bienvenida al cargar en r0 la dirección de memoria del mensaje a desplegar. Se repetirá esto ya que el mensaje ASCII está compuesto por varias lineas.
e. Se imprimen las instrucciones del juego, de igual forma se cargan a r0 y se imprimen con bl printf
f. Se procede a pedir el input al Jugador 1:
    i. Se guarda en r0 con mov el valor inmediato de #1 para pasarlo como parámetro a la función de input que se encuentra en la subrutina de *.game* .
    ii. Luego se obtiene la columna ingresada por el usuario
    iii. Se imprime el tablero usando la función *printBoard*
    iv. Se utiliza un salto para obtener la función de *getWinner* en la subrutina.
    v. Si el valor de *winner* es distinto de 0, se procede a imprimir el mensaje para el ganador, por lo que se carga el valor de la variable *winner* obtenida en la función anterior.
    vi. Si es igual a 0, se procede a imprimir mensaje de empate.
g. Se repite el proceso para el Jugador 2.
h. Se sale del programa.

Conclusiones

Fue posible realizar de manera exitosa el juego de *4-En-Linea* en lenguaje ARM Ensamblador mediante el uso de subrutinas. Para llevar el programa a cabo se utilizó la lógica de una matriz 4x4, en donde para verificar si un jugador había ganado, se comparaban los elementos de las filas y columnas, así mismo con las diagonales. El hecho de que el juego fuera modificado a un tablero 4x4 facilitó más las cosas, puesto que las

verificaciones y comprobaciones fueron más cortas; esto se puede ver en las diagonales, ya que en este tablero solo existen dos diagonales con cuatro elementos.

La complejidad solamente se encuentra a la hora de simplificar los registros que se usarán en el programa, ya que se cuenta con registros limitados. Para ello fue de utilidad el almacén en memoria de variables que controlan el flujo del programa, tales como el contador de turnos, ganador, contador de empates y la respectiva matriz representada con cuatro arreglos.

Anexos

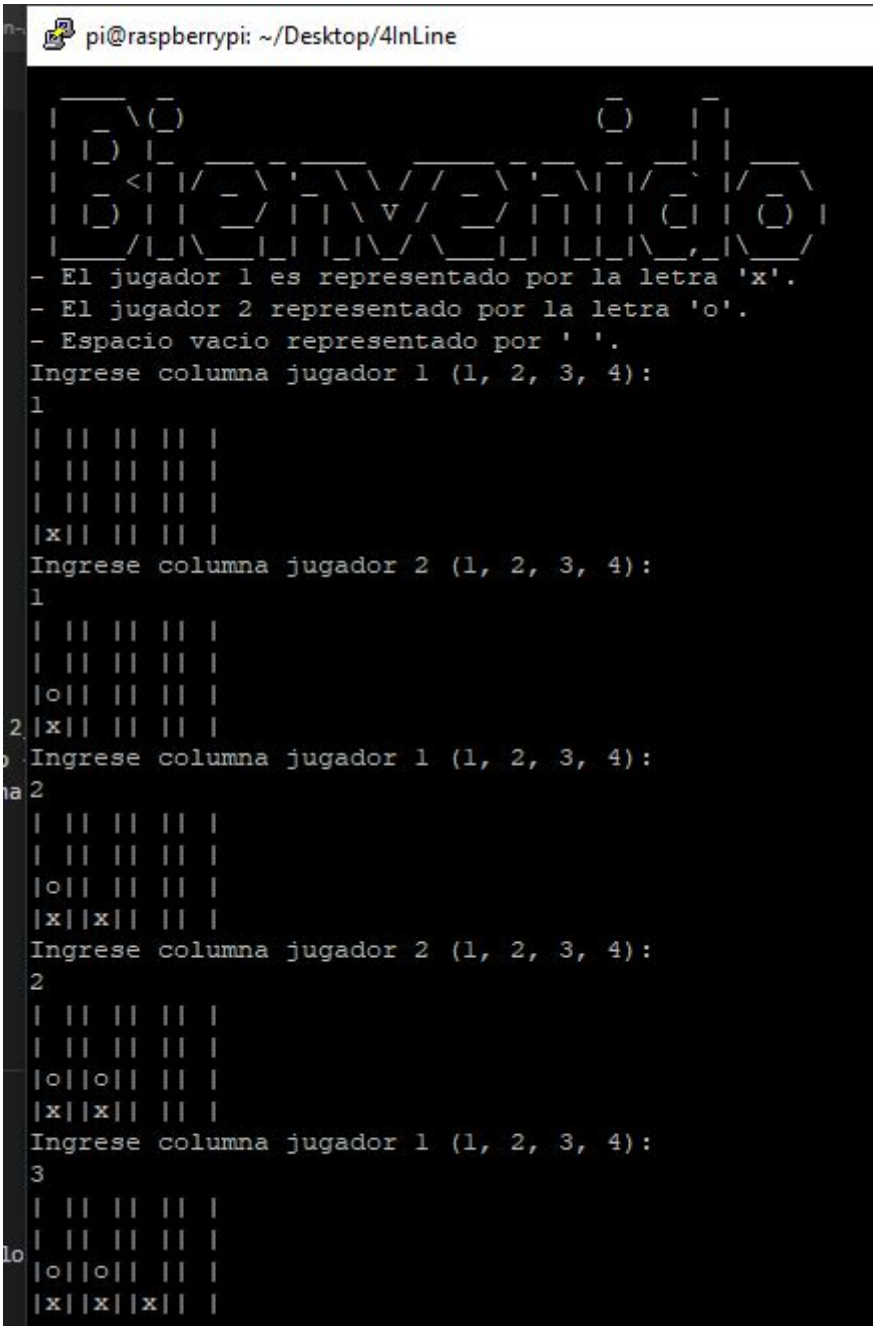

Figura 1. Flujo del programa presentado con ASCII Art

Figura 2. Caso para ganar en diagonal ascendente

Bibliografía:

[1] https://sourceware.org/binutils/docs/as/ARM-Directives.html
[2] https://thinkingeek.com/2013/03/16/arm-assembler-raspberry-pi-chapter-11/
[3] https://thinkingeek.com/2013/03/28/arm-assembler-raspberry-pi-chapter-12/