Desarrollo de una solución a la asignación de horarios de universitarios en lenguaje MIPS

Camila Gárate (6 horas) , Benjamín Sandoval (6 horas)
Departamento de Ingeniería Informática
Universidad de Santiago de Chile, Santiago, Chile
camila.garate@usach.cl, benjamin.sandoval.b@usach.cl

***Resumen*—Este documento aborda una problemática ficticia creada con fines educativos. Dicha problemática tiene lugar en el departamento de ingeniería informática de la Universidad de Santiago de Chile y consiste en la imposibilidad de designar un horario para los estudiantes, debido a problemas en la plataforma LOA, la cual normalmente cumple esa función. La solución consiste en un programa que facilita la asignación de cursos al detectar similitudes y diferencias automáticamente.**

***Palabras claves*—MIPS, sistema binario, registros de memoria**

I. Introducción

En la búsqueda de una solución se decide utilizar un sistema de tablas que permitan representar los bloques de clases. Estas tablas contienen valores binarios. El 0 representa que el bloque no está ocupado mientras que el 1 indica que si lo está. El objetivo principal de este trabajo consiste en fundamentar la creación de un programa en lenguaje ensamblador MIPS que permita comparar los bloques de clase entre dos horarios para saber qué bloques son iguales y cuáles no, con el fin de agilizar el proceso de asignación de cursos. Los números ingresados deben ser de 8 o menos bits. Como objetivo secundario es lograr interiorizarse con la herramienta Latex para la realización de este informe.

II. Antecedentes

Los conceptos necesarios para realizar este laboratorio son los siguientes:

II-A. MARS

"MARS es un entorno de desarrollo interactivo (IDE por sus siglas en inglés) para programación en lenguaje ensamblador MIPS, destinado a uso de nivel educativo".[1] "El lenguaje ensamblador depende directamente de la arquitectura del computador. Por tanto cada arquitectura tiene su propio lenguaje ensamblador".[2] Este IDE permite programar en el lenguaje asociado a las arquitecturas MIPS.

II-B. MIPS

Es un lenguaje ensamblador el cual es la representación simbólica de la codificación binaria del computador (lenguaje de máquina). [2]

II-C. Registros de memoria

MIPS dispone , entre otros, de los siguientes registros: 32 registros en la CPU, cada uno de 32 bits. Contiene 32 registros en la unidad de coma flotante, cada uno de 32 bits. Un contador de programa (PC) de 32 bits, que indica, al principio de cada ciclo, la dirección de memoria de la instrucción del programa que se va a ejecutar.[3]

II-D. Sistema Binario

"Es el sistema numérico de las computadoras o aparatos electrónicos en su mayoría. Dicho sistema numérico sólo cuenta con dos elementos : 1 y 0. Al igual que el sistema decimal se denomina como base 10, este sistema binario se fundamenta en tener como base el número 2".[4]

"La unidad más elemental de información en el interior de un ordenador es un valor binario (0 ó 1). Esta unidad elemental de información se denomina BIT (BInary uniT). Un bit representa la información correspondiente a la ocurrencia de un suceso de dos posibilidades (opciones) distintas".[5]

II-D1. Operador XOR: El operador xor opera 2 binarios bit a bit y genera como resultado un nuevo binario modificado. "El resultado de aplicar el operador xor en cada posición es 1 si el par de bits son diferentes y 0 si el par de bits son iguales".[6]

II-E. RAM

"La memoria es un dispositivo electrónico en el que se almacenan datos. Permite operaciones de escritura en las que los datos se almacenan en un lugar dado. La memoria también permite operaciones de lectura en las que dado un lugar, se obtiene el contenido previamente almacenado en dicho lugar. A este tipo de memoria se le conoce como memoria RAM (Random Access Memory)".[7] Ese lugar al que se hace referencia corresponde a las direcciones de memoria que son valores en sistema hexadecimal (base 16) que representan los distintos espacios en la RAM.

III. Materiales y métodos

III-A. Materiales

1. Sistema operativo: Windows 8.1 y 10 Pro
2. Procesador: Intel Pentium (4 núcleos) y Intel Core i5-4310u

3. RAM: 4 GB Y 16 GB
4. MARS 4.5 (ensamblador de MIPS)
5. Java oracle versión 8 actualización 111 y versión 8 actualización 51
6. Software LaTeX, editor Overleaf

III-B. Métodos

El programa lee 2 binarios que son ingresados por el usuario bit a bit, son alojados en las direcciones 0x100100a0 y 0x100100c0, y muestra el resultado alojado en 0x100100e0 por consola.

```
.data
        mensaje: .asciiz "Ingrese 2 binarios de izq a der. Presione 2 si el binario está completo"
        entrada1: .asciiz "Binario1:"
        entrada2: .asciiz "Binario2:"
        mensajeError: .asciiz "Por favor ingrese 0, 1 o un 2 si ha finalizado"
        msg_resultado: .asciiz "Resultado: "
        newline: .asciiz "\n"
        largo: .asciiz "largo:"
```

Figura 1: (Data segment + mensajes mostrados al usuario)

```
# Instrucciones de entrada
la $a0, mensaje
jal imprimirMensaje

la $a0, newline
jal imprimirMensaje
la $a0, newline
jal imprimirMensaje
```

Figura 2: (Instrucciones de uso del programa)

Se solicita al usuario ingresar el primer binario bit a bit. Los bits se almacenan en la dirección 0x100100a0. Cada bit ingresado se valida y se guarda.

```
pedirBinario1:
# Mensaje de solicitud del primer binario
la $a0, entrada1
jal imprimirMensaje

la $a0, newline
jal imprimirMensaje

# Pide el primer binario
 la $s0, 0x100100a0 # Dirección primer binario
 addi $t0, $zero, 4 # puntero
 addi $t1, $zero, 0 # contador
 addi $t3, $zero, 0 # limpieza $t3
# Se solicita el binario bit a bit, se comprueba que el valor sea válido y se guarda en memoria
loop1:
 beq $t1, 8, salida1
 jal pedirValor
 jal bitApropiado1
 beq $t3, 2, salida1

 sb $t3, 0($s0) # Carga en memoria
 addi $t1, $t1, 1 # contador + 1
 add $s0, $s0, $t0 # direccion + 4
 j loop1

salida1:
 la $a0, newline
 jal imprimirMensaje
```

Figura 3: (Se utiliza para que el usuario ingrese datos por consola)

Esto se hace de igual manera para solicitar al usuario el segundo binario, pero almacenando los bits en la direccion 0x100100c0.

Calcula el 'largo' de cada uno. Por ejemplo: '01001' tiene largo 5.

```
# Compara la cantidad de bits de cada binario
definirMenor:
 blt $t1, $t2, primeroMenor
 blt $t2, $t1, segundoMenor
 beq $t1, $t2, iguales
```

Figura 4: (Compara la cantidad de bits de cada binario)

Si el largo del primer binario es menor, por ejemplo: binario1 = '11', binario2 = '101', entonces adapta binario1 quedando '011'. Ahora binario1 tiene el mismo largo que binario2.

```
# Adapta el primer binario al largo del segundo usando la direccion 0x10010100 para realizar el desplazamiento
primeroMenor:
 sub $t6, $t2, $t1 # obtiene la diferencia de cantidad de bits
 addi $t4, $zero, 4 # puntero
 mul $t6, $t6, $t4 # adapta la diferencia a la arquitectura (4 bits)
 la $s0, 0x100100a0 # dirección del primer binario
 la $s1, 0x10010100 # direccion auxiliar para realizar el desplazamiento de bits
 add $s1, $s1, $t6 # realiza el desplazamiento
 addi $t0, $zero, 0 # limpieza $t0
 addi $t3, $zero, 0 # i=0
```

Figura 5: (Adapta el primer binario al largo del segundo usando la dirección 0x10010100)

Si el largo del segundo binario es menor se hace lo mismo que antes, pero esta vez con el binario2. Si son iguales no requiere igualar los largos.

Se ajustan los binarios en el data segment. Del ejemplo anterior: binario1 = '11' y binario2 = '101'. Luego de este paso quedan binario1 = '00000011' y binario2 = '00000101' (cada bit en una 'cajita' del data segment)

```
# Ubica ambos binarios ordenados en el data segment para ser operados
iguales:
 addi $t8, $zero, 8 # cantidad de bloques de 4 bits disponibles
 sub $t6, $t8, $t2 # se obtiene la cantidad de ceros que se requieren para rellenar a la izquierda
 addi $t4, $zero, 4 # puntero
 mul $t6, $t6, $t4 # cantidad de bits de diferencia
 la $s0, 0x100100a0 # direccion binario 1
 la $s1, 0x100100c0 # direccion binario 2
 la $s2, 0x10010100 # direccion auxiliar 1 para realizar el desplazamiento de bits
 la $s3, 0x10010120 # direccion auxiliar 2 para realizar el desplazamiento de bits
 add $s2, $s2, $t6 # realiza el desplazamiento 1
 add $s3, $s3, $t6 # realiza el desplazamiento 2
 addi $t0, $zero, 0 # limpieza $t0
 addi $t3, $zero, 0 # i=0
```

Figura 6: (Ubica ambos binarios ordenados en el data segment para ser operados)

Se realiza la operación XOR bit a bit entre los dos binarios almacenados en las direcciones de memoria mencionadas anteriormente, este resultado se guarda en la dirección 0x100100e0.

```
operarXor:
addi $t0, $zero, 0 # i=0
addi $t1, $zero, 4 # puntero
addi $t3, $zero, 0 # limpia $t3
addi $t4, $zero, 0 # limpia $t4
la $s0, 0x100100a0
la $s1, 0x100100c0
la $s2, 0x100100e0
cicloXor:
 beq $t0, 8, imprimirResultado

 lb $t3, 0($s0)
 lb $t4, 0($s1)
 xor $t5, $t3, $t4
 sb $t5, 0($s2)

 add $s0, $s0, $t1 # direccion + 4
 add $s1, $s1, $t1 # direccion + 4
 add $s2, $s2, $t1 # direccion + 4
 addi $t0, $t0, 1 # i++
 j cicloXor
```

Figura 7: (Operación XOR)

Se crea un procedimiento para mostrar el resultado de la operación XOR por consola.

Si uno de los binarios tiene 8 bits, el programa termina automáticamente sin requerir el 2. Por último, si uno de los bits ingresados no es ni 0, ni 1, ni 2, el programa imprime un mensaje y vuelve a pedir el binario desde el principio.

Finalmente el resultado se muestra en pantalla.

IV. Resultados

Esta sección presenta los resultados obtenidos, siguiendo el mismo orden de las etapas descritas en la sección III.

Se ha llegado a crear un programa que realiza una operación xor entre dos binarios ingresados por el usuario.

Indicaciones de uso: Al ingresar cada binario se requiere que el usuario ponga un 2 cada vez que el binario termine. Teóricamente se aceptan únicamente los valores 1 y 0.

```
Binario1:
13
Por favor ingrese 0, 1 o un 2 si ha finalizado
```

Figura 8: (Manejo de errores, programa indica si hay un dato que no es 1 o 0)

Al ingresar como binario 1: 10110 y como binario 2: 1010 da como resultado: 00011100

```
Ingrese 2 binarios de izq a der. Presione 2 si el binario está completo

Binario1:
101102
largo:5
Binario2:
10102
largo:4
Resultado: 00011100
-- program is finished running --
```

Figura 9: (Ejemplo)

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x72676e49	0x20657365	0x69622032	0x6972616e	0x6420736f	0x7a692065	0x20612071	0x2e726564
0x10010020	0x65725020	0x6e6f6973	0x20322065	0x65206973	0x6962206c	0x6972616e	0x7365206f	0x6320e174
0x10010040	0x6c70696f	0x206f7465	0x616e6942	0x315f6972	0x6942203a	0x6972616e	0x003a326f	0x20726f50
0x10010060	0x6f766166	0x6e692072	0x73657267	0x2c302065	0x6f203120	0x206e7520	0x69732032	0x20616920
0x10010080	0x616e6966	0x617a696c	0x52006f64	0x6c757365	0x6f646174	0x0a00203a	0x72616c00	0x003a6f67
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000001	0x00000000	0x00000001	0x00000001	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000001	0x00000000	0x00000001	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000001	0x00000001	0x00000001	0x00000000	0x00000000

Figura 10: (Operandos y resultado en las direcciones de memoria indicadas)

V. Conclusiones

Se ha cumplido con el objetivo principal de crear un programa funcional que lee 2 binarios, aplica el operador xor y devuelve el resultado por consola.

Además, se han dado a conocer los fundamentos detrás del desarrollo de un programa en el lenguaje MIPS y se ha aprendido a grandes rasgos el uso del entorno de desarrollo MARS, el uso de registros del procesador y el uso de memoria RAM.

También, se han aprendido conceptos básicos como bit, sistemas numéricos, memoria, entre otros.

Referencias

[1] M. university, "An ide for mips assembly language programming)," in *MARS (MIPS Assembler and Runtime Simulator*, Recuperado 20 de Abril de 2024. [Online]. Available: http://courses.missouristate.edu/kenvollmar/mars/

[2] A.Serrano and L.Rincon, "Programador ensamblador mips," in *Estructura y tecnologia de computadores*, Recuperado el 21 de Abril de 2024, pp. 1–24.

[3] J.Gomez, "ensablador mips," in *Estructura de computadores*, 1998, pp. 1–31.

[4] M. T. J. RAMÍREZ, "Sistema numérico binario," in *Sistemas Numéricos*, 2020, pp. 1–31. [Online]. Available: https://repositorio.utn.ac.cr/bitstreams/8e8c99f5-ff4c-40f6-8f8a-2b5b38ddb300/download

[5] L. A. . T. J. C. Prieto A., "Introducción a la informática (vol. 20)," in *Introducción a la Informática (Vol. 20)*, 1989.

[6] H. R. . R. D. G. González, "UtilizaciÓn de operadores a nivel de bit en aplicaciones web de domÓtica con raspberry pi," in *UTILIZACIÓN DE OPERADORES a NIVEL DE BIT EN APLICACIONES WEB DE DOMÓTICA CON RASPBERRY PI*, 2016. [Online]. Available: https://www.researchgate.net/profile/Henry-Gonzalez-Brito/publication/301359298_USING_OF_OPERATORS_BITWISE_IN_WEB_APPLICATIONS_OF_HOME_AUTOMATION_WITH_RASPBERRY_PI/links/571598ad08ae8ab56695b1eb/USING-OF-OPERATORS-BITWISE-IN-WEB-APPLICATIONS-OF-pdf

[7] A. Pardo, "Representación de datos en memoria," in *Representación de Datos en Memoria*, Recuperado el 21 de Abril de 2024, pp. 1–12. [Online]. Available: http://www.it.uc3m.es/luis/fo2/Memory-1x2.pdf