Facultad de Informática
Universidad Complutense de Madrid

Examen final extraordinario - Fundamentos de Computadores II
29 de Junio de 2023

Las tiras LED RGB son unos dispositivos ampliamente utilizados en la iluminación de hoy en día. Para controlarlas desde un procesador RISC-V, se asigna a cada LED una posición de memoria consecutiva. Por ejemplo, si la dirección del primer led es la `0x0`, la del segundo será la `0x4` y así sucesivamente. En dichas direcciones se escriben 32 bits de información que codifican el color elegido.

Para cambiar el color de la tira led se deben escribir todas las posiciones de memoria consecutivamente. Se dispone del siguiente código, ejecutado en un micro-controlador RISC-V. El código contiene una función `set_led_colors` que recibe tres argumentos:

- La dirección de memoria donde están los datos (colores) que queremos copiar a los leds. (D_c)
- La dirección de memoria donde comienzan los leds (D_l)
- El tamaño de la tira de leds (N)

En resumen, la función realiza la copia de un array D_c a otro D_l, ambos de tamaño N:

```
.equ N, 8
.data
    COLORS_A: .word 0x00ff0000, 0x00ff0000, 0x000000ff, 0x00ffffff,
                    0x00ffffff, 0x000000ff, 0x0000ff00, 0x0000ff00
    COLORS_B: .word 0x000000ff, 0x000000ff, 0x000000ff, 0x00ffffff,
                    0x00ffffff, 0x00ff0000, 0x00ff0000, 0x00ff0000
.bss
    LEDS: .space 4*N

.text

#función SET_LED_COLORS
set_led_colors:
bucle:
    lw t1, 0(a0)        #cargo color
    sw t1, 0(a1)        #grabo color
    addi a0, a0, 4      #incremento dir de array fuente
    addi a1, a1, 4      #incremento dir de array destino
    addi a2, a2, -1     #decremento iterador
    bne a2, zero, bucle #compruebo terminación
fin_bucle:
    ret                 #devuelvo control

#función MAIN
.global main
main:
    #alterno entre los dos patrones de colores
main_loop:
    #cargo y llamo con patrón A
    la a0, COLORS_A
    la a1, LEDS
    li a2, N
    j set_led_colors
    #cargo y llamo con patrón B
    la a0, COLORS_B
    j set_led_colors
    #repito el bucle indefinidamente
    j main_loop
```

Responda a las siguientes preguntas:

1. **ASM [0.4pt]** El código proporcionado para la función `main` tiene fallos. Indíquelos y proponga una solución.
2. **ASM [0.3pt]** Supongamos que la sección `.text` se ubica a partir de la dirección `0x0200`, y por error llamo a la función `set_led_colors` con `a1=0x0200`. ¿Qué ocurre?
3. **ASM [0.3pt]** Convierta las instrucciones `addi a0, a0, 4` y `addi a1, a1, 4` a código máquina y señale en qué difieren sus codificaciones hexadecimales.
4. **CPU [1pt]** En la ruta de datos del procesador monociclo ejecutamos la última instrucción del bucle `bne a2, zero, bucle`. Indique los valores que toman las señales indicadas, en la primera y en la última iteración, recalcando las diferencias observadas. **NOTA:** Asuma que `.text` se ubica en `0x200`.

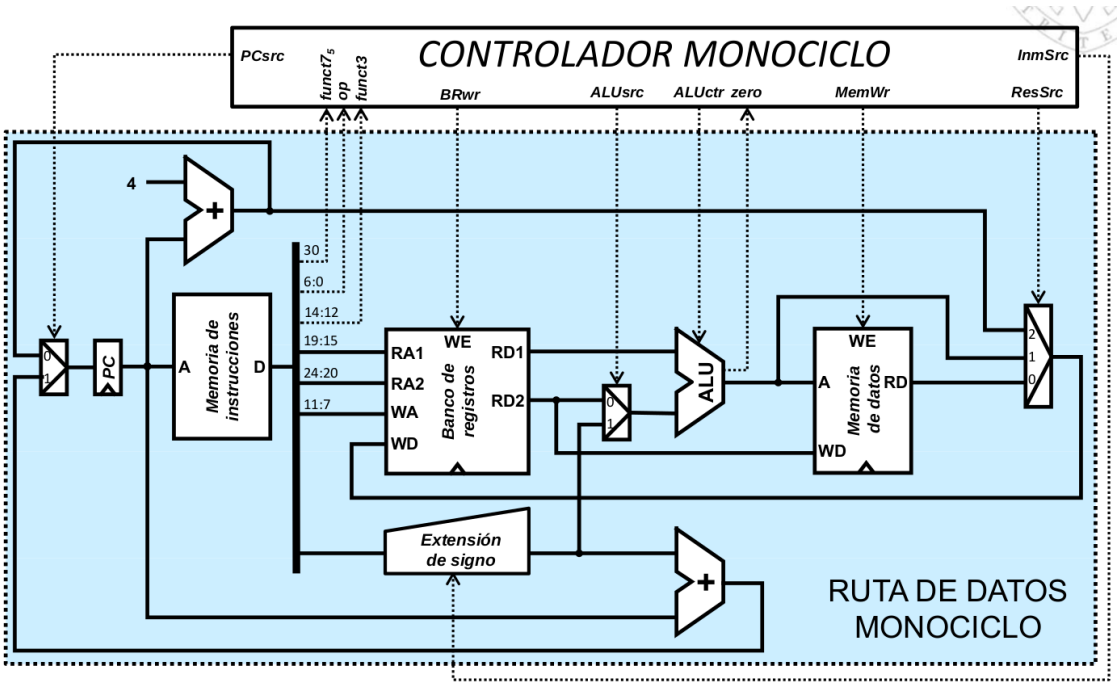
Iteración	PCSrc	Entrada PC	BRwr	ALUsrc	Zero	Salida ALU	MemWr	ResSrc
Primera								
Última								

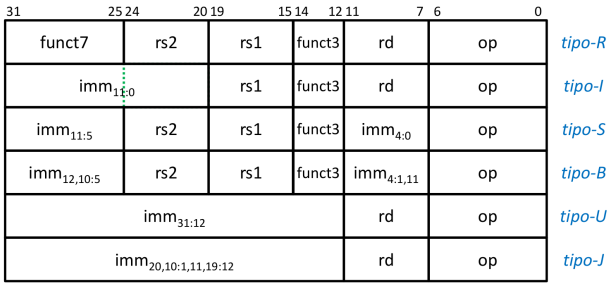
5. **CPU [1pt]** Ejecutamos la función `set_led_colors` (con `a2=16`) en un procesador multiciclo, y observamos que tarda un total de 387 ciclos. Sabiendo que las instrucciones `lw` y `sw` tardan el doble de ciclos que `addi`, y que el resto de instrucciones tardan siempre 3 ciclos, calcule cuánto tardan las instrucciones `lw`, `sw` y `addi` individualmente.
6. **CPU [1pt]** ¿Cuánto mayor debe ser la frecuencia del procesador multiciclo anterior frente a la de un procesador monociclo, para que consiga un menor tiempo de ejecución en la función? ¿Por qué?
7. **CPU [1.5pt]** Supongamos que se ejecuta una iteración del bucle de `set_led_colors` en el procesador segmentado RISC-V. El procesador tiene gestión de conflictos de datos por anticipación (con parada cuando no sea posible), predicción de salto no tomado, y escritura en banco de registros a mitad de ciclo, como el visto en clase.
   a) Indique los conflictos que se dan durante la iteración.
   b) Realice el diagrama de ejecución, indicando paradas y anticipaciones si las hubiera.
   c) ¿Cuántos ciclos tarda la ejecución de la iteración?
8. **CPU [0.5pt]** Calcula y comenta qué ocurre con el CPI del procesador segmentado anterior en la función `set_led_colors` al llamar con un valor bajo y alto (ej: 2 y 16) en `a2`. ¿Con cuál se obtiene menor CPI? ¿Por qué?
9. **CPU [1pt]** Si en el anterior procesador se elimina sólo la unidad de anticipación (y no se para el pipeline ante los riesgos), habría operaciones que se ejecutarían con datos erróneos al no poder anticiparse.
   a) Reordene el código del bucle de `set_led_colors` para que el resultado de la ejecución siga siendo correcto.
   b) Calcule el número de ciclos de la función reordenada si se llama con `a2=2`.
10. **MEM [1pt]** Disponemos de un procesador multiciclo RISC-V con una memoria principal de 64KB dividida en 2048 Bloques. La caché es de 128B y de emplazamiento directo. Indica los formatos de dirección para ambas.
11. **MEM [2pt]** Se coloca la sección `.data` en la dirección `0x2000` y a continuación la sección `.bss`. `.text` se coloca en la dirección `0x3020`. Rellene la siguiente tabla, teniendo en cuenta que la caché es unificada para datos e instrucciones:

Elemento	Dir. inicial	Dir. final	Bloque MP	Etiqueta	Marco MC
`COLORS_A`					
`COLORS_B`					
`LEDS`					
`set_led_colors`					

Queremos analizar el comportamiento de la caché durante la primera invocación de `set_led_colors` (`a0=dir(COLORS_A)`).

a) Escriba la secuencia de direcciones de memoria que genera el procesador multiciclo al ejecutar la primera y segunda iteración del bucle, indicando a qué bloque MP y marco MC corresponden.
b) Indique el número de accesos y fallos que se producen durante la ejecución completa de la función en el procesador multiciclo, calculando la tasa de fallos.
c) Si el tiempo de acierto $t_h = 1ns$ y el tiempo de penalización por fallo $t_p = 10ns$, calcule el tiempo medio de acceso a memoria T_{MEM}





op	funct3	funct7*	Instrucción	Tipo
0010011	000	-	**addi**	I
	001	0000000*	**slli**	I
	010	-	**slti**	I
	011	-	**sltiu**	I
	100	-	**xori**	I
	101	0000000*	**srli**	I
	101	0100000*	**srai**	I
	110	-	**ori**	I
	111	-	**andi**	I

Nombre	Número	Código
zero	x0	00000
ra	x1	00001
sp	x2	00010
gp	x3	00011
tp	x4	00100
t0	x5	00101
t1	x6	00110
t2	x7	00111
s0/fp	x8	01000
s1	x9	01001
a0	x10	01010
a1	x11	01011
a2	x12	01100
a3	x13	01101
a4	x14	01110
a5	x15	01111

Nombre	Número	Código
a6	x16	10000
a7	x17	10001
s2	x18	10010
s3	x19	10011
s4	x20	10100
s5	x21	10101
s6	x22	10110
s7	x23	10111
s8	x24	11000
s9	x25	11001
s10	x26	11010
s11	x27	11011
t3	x28	11100
t4	x29	11101
t5	x30	11110
t6	x31	11111