Facultad de Informática
Universidad Complutense de Madrid

Introduction to Computers II - Exam
June 29th, 2023

RGB LED strips are devices widely used in lightning applications. These can be controlled from a RISC-V processor, assigning consecutive memory positions to each of the leds. For example, if the address of the first led is 0x0, the address of the second one will be 0x4 and so on. In these addresses, 32 bits are written, which encode the chosen color.
In order to change the color of the led strip, all memory positions have to be written in a consecutive way. The following RISC-V code is provided. The code contains the set_led_colors function that receives three arguments:

- The memory address that contains the data (colors) that will be copied to the leds (D_c)
- The initial memory address assigned to the leds (D_l)
- The size of the led strip (N)

In short, the function copies one array (D_c) into another one (D_l), both with size 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
 #SET_LED_COLORS  function
 set_led_colors:
 loop :
        lw t1 , 0(a0)               # color loaded
        sw t1 , 0(a1)               # color saved
        addi a0 , a0 , 4            # source array address incremented
        addi a1 , a1 , 4            # destination array address incremented
        addi a2 , a2 , -1           # loop index decremented
        bne a2 , zero , loop        # termination checked
 end_loop :
        ret                         # return

# MAIN function
.global main
main :
        # two color patterns are alternated
main_loop:
        # A pattern call
        la a0 , COLORS_A
        la a1 , LEDS
        li a2 , N
        j set_led_colors
        # B pattern call
        la a0 , COLORS_B
        j set_led_colors
        # loop is repeated forever
        j main_loop
```

Answer the following questions:

1. **ASM [0.4pt]** The main function code is not correct. Identify why and propose a solution.
2. **ASM [0.3pt]** Let us assume that the .text section is located in address 0x0200, and, by mistake, the set_led_colors function is called with a1=0x0200. What happens?
3. **ASM [0.3pt]** Translate the addi a0, a0, 4 and addi a1, a1, 4 instructions into machine code and indicate the difference in their hexadecimal encodings.
4. **CPU [1pt]** The last instruction of the loop, bne a2, zero, loop, is executed in the single-cycle processor data path. Indicate the values taken by the following signals, in the first and last iterations, highlighting the observed differences. NOTE: Assume that .text is located in 0x200.

Iteration	PCSrc	PC input	BRwr	ALUsrc	Zero	ALU output	MemWr	ResSrc
First								
Last								

5. **CPU [1pt]** The set_led_colors function (with a2=16) is executed in a multicycle processor, which takes a total of 387 cycles. If we assume that the lw and sw instructions take twice as many cycles as addi, and that the rest of instructions take 3 cycles always, calculate how long it would take to execute the lw, sw and addi instructions, individually.
6. **CPU [1pt]** How much larger should the previous multicycle processor frequency be respect to a single-cycle processor, so that the function takes less time to execute? Why?
7. **CPU [1.5pt]** Let us assume that a loop iteration of set_led_colors is executed in the RISC-V pipelined processor. The processor uses forwarding to handle data hazards (with pipeline stall when needed), it also uses not-taken branch prediction and the register file is written at the middle of the cycle.
   a) Indicate the hazards produced during the iteration.
   b) Provide the execution diagram indicating the pipeline stalls and forwarding operations (if any).
   c) How many cycles are needed to execute the loop iteration?
8. **CPU [0.5pt]** Calculate and discuss what happens with the CPI of the previous pipelined processor in the set_led_colors function, when calling it with a low and high value (e.g. 2 y 16) in a2. Which one would produce a lower CPU? Why?
9. **CPU [1pt]** If the forwarding unit is removed from the previous processor and the pipepine is not stalled in the hazards, there would be operations executed with the wrong operands, since these could not be forwarded.
   a) Reorder the loop code of set_led_colors so that the final result is correct.
   b) Calculate the number of cycles of the reordered function if it is called with a2=2.
10. **MEM [1pt]** We have a RISC-V multicycle processor with a 64-KB main memory, divided in 2048 blocks. The cache memory has 128B, with direct mapping. Indicate the address format of both the main memory and the cache.
11. **MEM [2pt]** The .data section is placed in address 0x2000 and after that the .bss section. The .text section is placed in address 0x3020. Fill the following table, considering that there is a single cache for data and instructions:

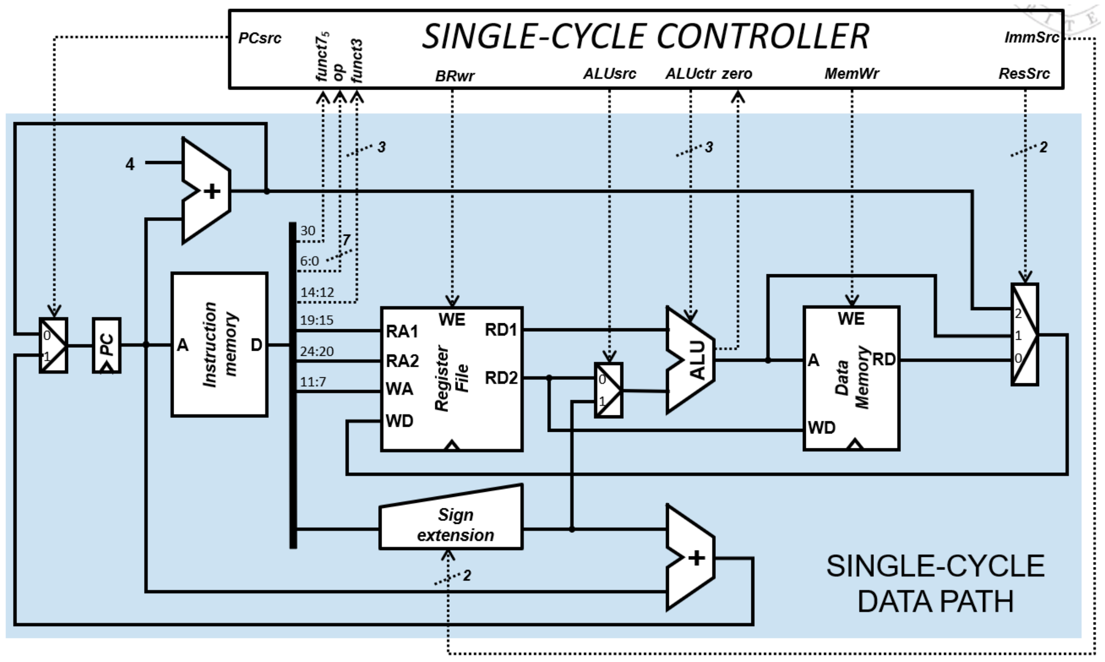
Item	Initial addr	Final addr	MM block	tag	CM block
COLORS_A					
COLORS_B					
LEDS					
set_led_colors					

We want to analyze the behavior of the cache during the first call of the set_led_colors function (a0=addr(COLORS_A)).

a) Write the sequence of memory addresses generated by the multicycle processor when executing the first and second loop iterations, indicating the corresponding MM and CM blocks.

b) Indicate the number of accesses and misses produced during the whole execution of the function in the multicycle processor, calculating the miss rate.

c) If the hit time is t_h = 1ns and the miss penalty time is t_p = 10ns, calculate the average memory access time T_{MEM}.



31 25	24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	op	*R-type*
$\text{imm}_{11:0}$		rs1	funct3	rd	op	*I-type*
$\text{imm}_{11:5}$	rs2	rs1	funct3	$\text{imm}_{4:0}$	op	*S-type*
$\text{imm}_{12,10:5}$	rs2	rs1	funct3	$\text{imm}_{4:1,11}$	op	*B-type*
$\text{imm}_{31:12}$				rd	op	*U-type*
$\text{imm}_{20,10:1,11,19:12}$				rd	op	*J-type*

op	funct3	funct7*	Instruction	Type
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

Name	Number	Code
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

Name	Number	Code
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