ensia The National School of Artificial Intelligence المدرسة الوطنية العليا للذكاء الاصطناعي

Computer Architecture Midterm Solution

Exercise 1:

1) Assume the delay for each stage in the datapath is as follows: **(1.5)**

IF: 200 ps ID: 100 ps EX: 200 ps MEM: 200 ps WB: 100 ps

Mark with **X** the stages of the datapath that the following instructions use and calculate the total time needed to execute the instruction.

	IF	ID	EX	MEM	WB	Total Time	
add	X	X	X		X	600ps	0.25
ori	X	X	X		X	600ps	0.25
lw	X	X	X	X	X	800ps	0.25
sw	X	X	X	X		700ps	0.25
beq	X	X	X			500ps	0.25
jal	X	X	X		X	600ps	0.25

2) Datapath designers are interested in reducing the phases necessary for execution such that instead of accessing both the EXE phase and the MEM phase, instructions access either one or the other, but not both. This would create a 4-stage, single-cycle datapath with the following stages: IF, ID, EXE_OR_MEM, and WB. **(1)**

IF	ID	EXE	MEM	WB
100ps	150ps	200ps	350ps	150ps

a. Given the table above and the described datapath above, how much time does it take for a single instruction that utilizes all stages to execute on the typical 5-stage, single-cycle datapath?

100 + 150 + 200 + 350 + 150 = 950ps 0.25

b. How much time does it take for a single instruction that utilizes all stages to execute on the new 4-stage, single-cycle datapath?

100 + 150 + MAX(200, 350) + 150 = 750ps 0.25

c. Which instructions will not function correctly with this new design? Why? Please limit your answer to two sentences or less.

Load and store instructions because they require both the ALU and Memory phases. The address must first be calculated before memory can be accessed. 0.25

d. Propose a program-level modification that will fix the issue. Do not propose a modification to the datapath. Please describe your modification in two sentences or less.

The compiler (or the programmer) can expand load/store instructions into a load/store + addi pair. This way, the address is calculated by a separate instruction before the memory access takes place. 0.25

3) Explain what happens in each datapath stage. **(1.25)**

a) IF Instruction Fetch

Send the address to the instruction memory, and read IMEM at that address. 0.25

b) ID Instruction Decode

Generate control signals from the instruction bits, generate the immediate, and read registers from the RegFile. 0.25

c) EX Execute

Perform ALU operations, and do branch comparison. 0.25

d) MEM Memory

Read from or write to the data memory. 0.25

e) WB Writeback

Write back the PC + 4, the result of the ALU operation, or data from memory to the RegFile. 0.25

4) Given the RISC-V code below and a pipelined CPU with no forwarding, how many hazards would there be? What types are each hazard? Consider all possible hazards from all pairs of instructions. **(2)**

Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9
1. sub t1, s0, s1	IF	ID	EX	MEM	WB				
2. or s0, t0, t1		IF	ID	EX	MEM	WB			
3. sw s1, 100(s0)			IF	ID	EX	MEM	WB		
4. bgeu s0, s2, 1				IF	ID	EX	MEM	WB	
5. add t2, x0, x0					IF	ID	EX	MEM	WB

There are four hazards:

1) between instructions 1 and 2 0.25 (data hazard from t1), 0.25

2) instructions 2 and 3 0.25 (data hazard from s0), 0.25

3) instructions 2 and 4 0.25 (data hazard from s0), 0.25

4) instructions 4 and 5 0.25 (a control hazard). 0.25

5) How would you fix each hazard? How many stalls would need to be added? **(1.5)**

For data hazards, insert stalls between instructions. 0.25 For control hazards, use branch prediction. 0.25

2 stalls are needed for instruction 1 and 2. 0.25

2 stalls are needed for instruction 2 and 3. 0.25

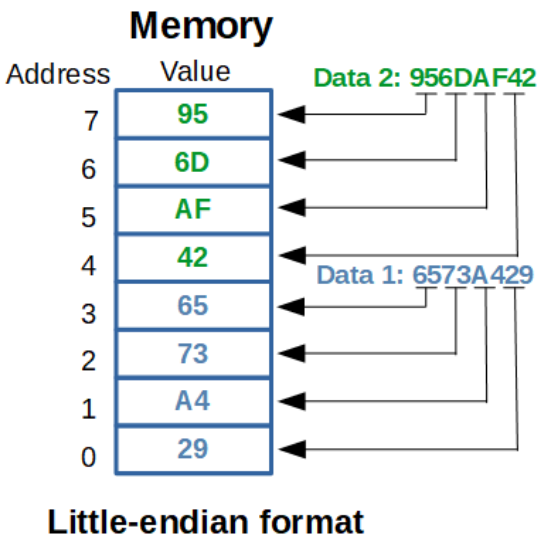
No stall is needed for instruction 2 and 4. 0.25

No stall is needed for branch prediction. 0.25

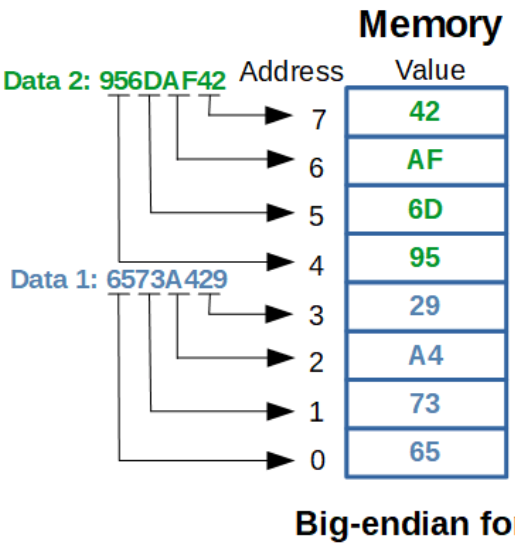
Total 4 stalls.

6) Represent the following words in memory with their addresses. Assume that the address starts at 0. **(1)**

a) ox6573A429 **0.5** b) ox956DAF42 **0.5**



Little-endian format



Big-endian format

Exercise 2:

1) For each statement of the following RISC-V machine language code, determine the immediate value in decimal. **(1.5)**

0x008000EF	**8 0.25**
0x00892023	**0 0.25**
0x00051463	**8 0.25**
0xFF5FF06F	**-12 0.25**
0xFFF50513	**-1 0.25**
0xFF1FF06F	**-16 0.25**

2) In the following RISC-Vcode, the numbers to the left of each instruction indicate the instruction address. **Translate the instruction sequence into machine code in hexadecimal.** **(2)**

0x00400028 addi a0, a1, 0 : **0x00058513 0.25**

0x0040002C jal f2 : **0x008000EF 0.25**

0x00400030 f1: jr ra : **0x00008067 0.25**

0x00400034 f2: sw s0, 0(s2) : **0x00892023 0.25**

0x00400038 bne a0, x0, else : **0x00051463 0.25**

0x0040003C j f1 : **0xFF5FF06F 0.25**

0x00400040 else: addi a0, a0, -1 : **0XFFF50513** **0.25**

0x00400044 j f2 : **0XFF1FF06F** **0.25**

3) If we assume we place the following loop starting at location 80000 in memory, what is the RISC-V machine code in hexadecimal for this loop? **(1.5)**

Loop: slli x10, x22, 2

add x10, x10, x25

lw x9, 0(x10)

bne x9, x24, Exit

addi x22, x22, 1

beq x0, x0, Loop

Exit:

Address	Instruction
80000	**0x002B1513** **0.25**
80004	**0x01950533** **0.25**
80008	**0x00052483** **0.25**
80012	**0x01849663** **0.25**
80016	**0x001B0B13** **0.25**
80020	**0xFE0006E3** **0.25**

4) Decode the following machine code: **(1)**

00000000001100010010001010110011	**slt x5, x2, x3** **0.25**
00000000100000100000011110000011	**lb x15, 8(x4)** **0.25**

00000000001000011001001000100011	**sh x2, 4(x3) 0.25**
11101110000101010001100100010111	**auipc x18, 0xEE151 //0xEE151000 0.25**

Exercise 3:

(1.25)

A palindrome is a sequence of characters that reads the same backward and forward. For example, “civic” and “redder” are palindromes, but “wave” and “canal” are not palindromes. Implement the RISC-V function find_palindrome that takes as input a nonempty null-terminated string in a0 and its length (excluding the null-terminator) in a1. The function should return 1 in a0 if the string is a palindrome and 0 in a0 otherwise. Assume the input string contains only lowercase letters.
You may only use registers a0, a1, t5 and t6.

find_palindrome:

add **a1 a0 a1 0.125**

loop:

addi a1 a1 -1 0.125

lb t5 0(a0) **0.125**

lb t6 0(a1) **0.125**

bne t5 t6 not_palindrome **0.125**

addi a0 a0 1 0.125

blt a0 a1 loop 0.125

addi **a0 x0 1 0.125**

jalr x0 ra 0.125

not_palindrome:

mv **a0 x0 0.125**

jalr ra

Exercise 4:

Write a RISC-V program that asks the user to write his name, then calculate the number of characters his name contains, and finally print the following message: “Your name is N characters long”, where N is the number of characters. Recall that string read using system calls ends with the “\n”, represented by 1010 or 10 in decimal. **(3)**

.data 0.125

txt1: .asciz "Please enter your name: "

txt2: **.asciz "Your name is " 0.125**

txt3: **.asciz " characters long" 0.125**

buffer: .space 255

.text 0.125

li a7 4 #It must be a7 0.125

la a0 txt1 #It must be a0 0.125

ecall 0.125

li a7 8 #It must be a7 0.125

la a0 buffer #It must be a0 0.125

li a1 255 #It must be a1 but any number is correct 0.125

ecall 0.125

addi a1 x0 -1 # counter

addi t1 x0 10 # the ending character “\n”

loop:

lb t0, 0(a0) 0.125

addi a1 a1 1 0.125

addi a0 a0 1 0.125

bne t0 t1 loop **0.125**

li a7 4 #It must be a7 0.125

la a0 txt2 #It must be a0 0.125

ecall 0.125

li a7 1 #It must be a7 0.125

add a0 x0 a1 #It must be a0 0.125

ecall 0.125

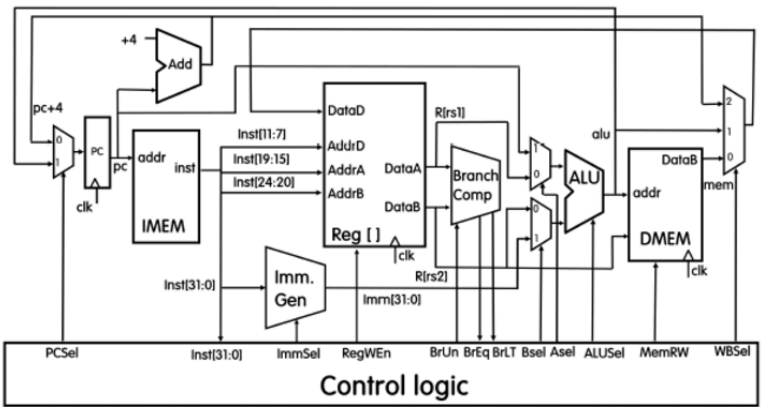
li a7 4 #It must be a7 0.125

la a0 txt3 #It must be a0 0.125

ecall 0.125

Exercise 4: (1.5)

Consider the following datapath and control logic table:



PC Sel	ASel	BSel	WBSel	Regfile
1:ALU ***:** don't care **I**: input dependent **0**: pc+4	**0**: Reg[rs1] **1**: pc ***:** don't care	**0**: Reg[rs2] **1**: imm ***:** don't care	**1**: ALU ***:** don't care **0**: mem **2**: pc + 4	**0**:Write Reg[rd] **1**: Read Reg[rs1] **2**: Read Reg[rs2]

Select all that apply for the following instructions (**write only the numbers**):

	PC Sel	ASel	BSel	WBSel	Regfile	
add	**0**	**0**	**0**	**1**	**0 1 2**	**0.25**
ori	**0**	**0**	**1**	**1**	**0 1**	**0.25**
lbu	**0**	**0**	**1**	**0**	**0 1**	**0.25**
sb	**0**	**0**	**1**	*****	**1 2**	**0.25**
blt	**I**	**0**	**0**	*****	**1 2**	**0.25**
jal	**1**	**1**	**1**	**2**	**0**	**0.25**