

National University of Computer and Emerging Sciences, Lahore Campus



Course:	Computer Architecture	Course Code:	EE204
Program:	BS(Computer Science)	Semester:	Spring 18
Duration:	180 Minutes	Total Marks:	70
Date:	16-05-2018	Weight	45%
Section:	All	Page(s):	8
Exam:	Final Solution	Student Id:	

NOTE: Answer in the space provided. You can ask for rough sheets but they won't be graded.
In case some information is missing, make assumption and mention it at the top of your solution.

Question 1 [Marks: 2+2+2+2+2]

Provide reasoning for the following questions!

1. We may need to add a cycle before and a cycle after branch instruction to solve hazards. Does both of these stalls are due to control hazard?

Answer:

No. A cycle before a branch instruction may be required due to data hazard if branch instruction is dependent on some previous instruction. Cycle after the branch instruction are usually due to control hazard.

2. Stalls can be added to solve hazards. Can we solve false dependencies (WAR, WAW) using stalls?

Answer:

No. False dependencies does not cause any harm to the output of the program unless and until we change the flow of the program. So stall does not provide any help to remove false dependencies. A simple solution to resolve them is 'renaming the registers'. By renaming we remove false dependencies and now these independent instructions can be used to remove stalls in the program.

3. What is execution hazard in RAW? If forwarding is implemented, do we still need stalls to remove this hazard?

Answer:

An instruction could be dependent on previous instruction. If the previous instruction is not load instruction then the result of this instruction will be available after execution and current instruction required it at the start of execution. Such a RAW is called execution hazard.

If forwarding is implemented then no stalls is required as execution stage of second instruction will start when the execution of first stage is completed.

4. Why we need loop unrolling? Suppose a loop has 22 iterations and we want to unroll it by degree 3. In this scenario, how we will execute 22 iterations?

Answer:

An efficient way to solve RAW dependency instead of stalls is to find independent instructions and put them between dependent instructions.

If dependency is among instructions within a loop then normally we don't have enough independent instructions to replace all stalls. One way to increase the number of independent instructions is to unroll the loop in which the working instructions are repeated without repeating loop overhead instructions.

If an original loop has 22 iterations and loop is unrolled by degree 3 then this loop can be run 7 times thus completing 21 iteration of the original loop and remaining one iteration can be completed by running the original loop 1 time.

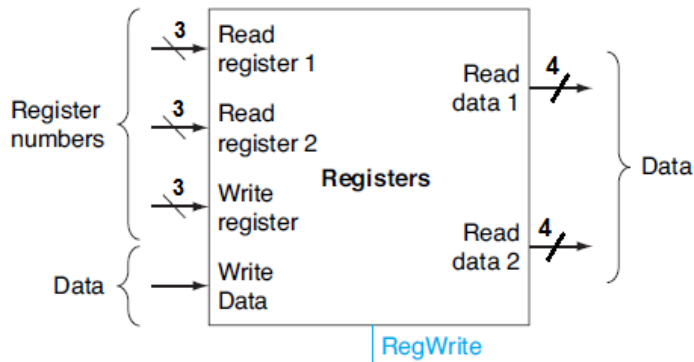
5. What is a VLIW processor? Does VLIW processor always provide speedup of more than 1?

Answer:

Very large instruction Word is a type of processor where multiple instructions are combined into a large instruction in order to achieve parallelism. Number of instructions depends on the architecture of the processor and the dependency among instructions. In order to combine multiple instructions into a single instruction all of them should be independent from each other.

If all instructions are dependent on each other and we cannot combine instructions then there is no speed up and performance would be equal to a scalar processor.

Question 2: [Marks: 2 +3 + 5]



This is the diagram of Register file which is built using different components (Registers, Decoders, multiplexers).

- a) How many registers are there in this register file and what is the size of each register?

Total 8 registers of 4 bit each

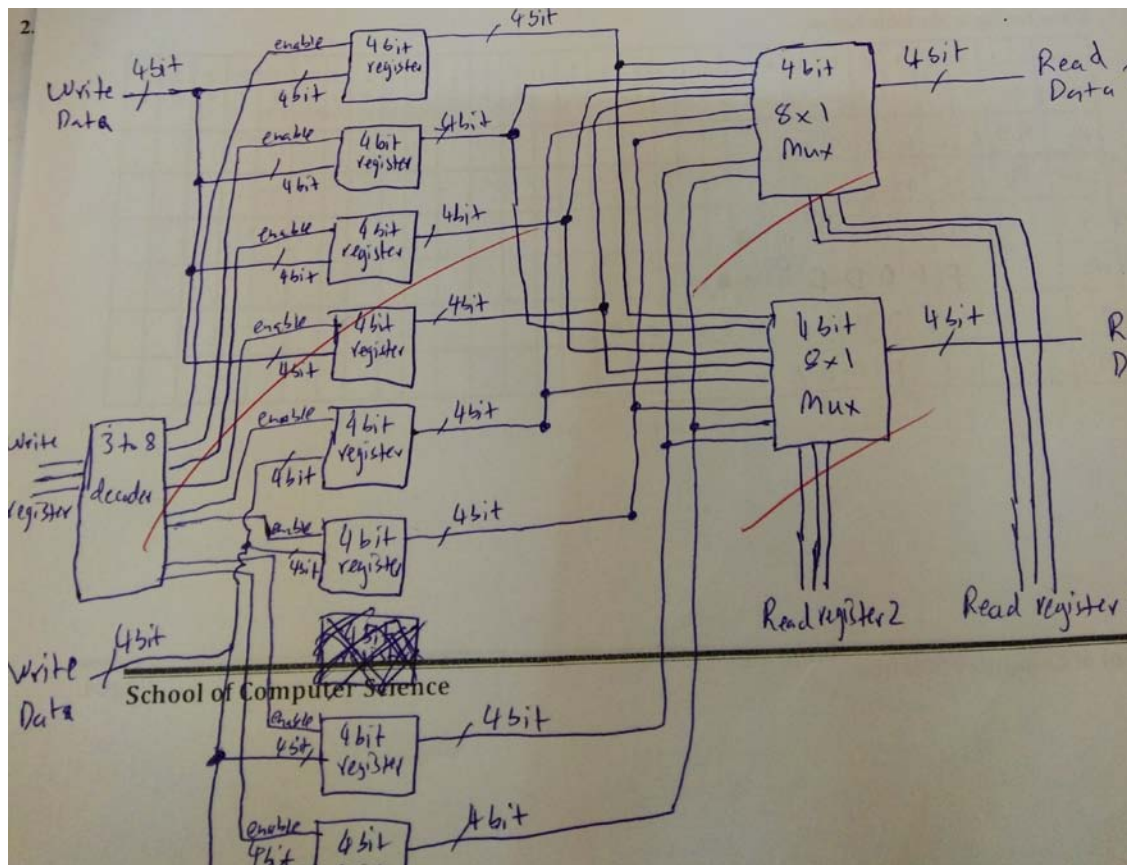
- b) In order to implement complete functionality of this register file, which components are required, in what quantity and what would be the size of each component (number of inputs and outputs).

8 Registers

2 multiplexors with 8 input and 1 output of 4 bit each

1 3-8 decoder

- c) Draw circuit with all Registers and circuitry required to select a particular register by reading port **Read register 1** and providing data on **Read data 1**. Detail of other functionalities is not required!



Question 3: [Marks: 8+2+3+2]

Consider the following program fragment executing on a basic 5-stage MIPS pipeline **with no data forwarding**. All stages take 1 cycle, except the Execute stage, which takes a variable number of cycles, depending on the functional unit used:

Functional unit	Number of EX cycles
Integer ALU	1
Floating Point Add	2
Floating Point Load/Store	2
Floating Point Multiply	4

- a. For each instruction, determine in which clock cycle the instruction is safely completed (i.e., when does its write back stage occur) after inserting stalls. **F** at the end of instruction op-code denotes floating point instruction e.g. MULTF.

Assume no data forwarding occurs. Ignore false dependencies while scheduling.

Instruction	Write Back Cycle
1: MULTF F1, F2, F3	8
2: ADD R1, R1, R2	6
3: LF F5, 0(R1)	10
4: SUBF F5, F6, F1	12
5: SF F5, 0(R1)	16
6: MULTF F5, F3, F4	13

Show timing in the table below:

Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Inst										0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0
(1)	F	D	E	E	E	E	M	WB																						
(2)		F	D	E	M	WB																								
(3)			F	D	D	D	E	E	M	WB																				
(4)				F	D	D	D	D	E	E	M	WB																		
(5)					F	D	D	D	D	D	D	E	E	M	WB															
(6)						F	D	E	E	E	E	M	WB																	

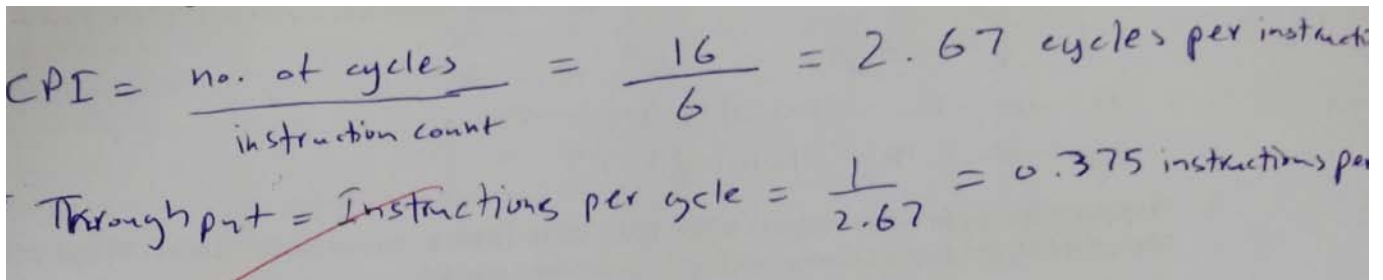
- b. Identify false dependencies in the above code and tell whether they created any problem in the above code schedule or not.

3 & 4: WAW

5 & 6: WAR

No, they have not created any problem in the above code!

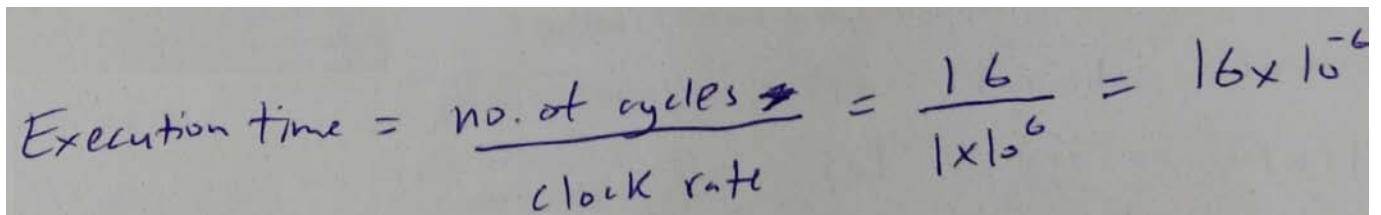
- c. Calculate the average CPI and throughput (in instructions per cycle) of the processor for the above code.



Handwritten calculation for CPI and Throughput:

$$\text{CPI} = \frac{\text{no. of cycles}}{\text{instruction count}} = \frac{16}{6} = 2.67 \text{ cycles per instruction}$$
$$\text{Throughput} = \text{Instructions per cycle} = \frac{1}{2.67} = 0.375 \text{ instructions per cycle}$$

- d. If we run this program on a processor with frequency of 1 MHZ, what would be its execution time?



Handwritten calculation for Execution time:

$$\text{Execution time} = \frac{\text{no. of cycles}}{\text{clock rate}} = \frac{16}{1 \times 10^6} = 16 \times 10^{-6} \text{ seconds}$$

Question No. 4 [Marks: 10]

Consider a machine with a byte addressable main memory of 64 Kbytes. Assume we have a 2-way set associative cache of 4K data bytes size and 8 bytes block size that is being used with this machine.

- a. Calculate the tag, index, and offset bits?

$$\text{offset bits} = \log_2(8) = 3 \text{ bits}$$

for index:-

$$\text{no. of sets} = \frac{4096}{8} = 2^{12-3} = 2^9 = 2^8$$

$$\text{index bits} = \log_2(2^8) = 8 \text{ bits}$$

$$\text{As main memory address} = \log_2(64 \text{ Kbytes}) = 16 \text{ bits}$$

$$\text{tag} = 16 - 8 - 3 = 5 \text{ bits}$$

- b. Into what set would bytes with each of the following addresses be stored

- 0001 0001 0001 1011
- 1010 1010 1010 1010

Set number

35

85

$$1) \text{ index bits} = 0010011 = 35$$

$$2) \text{ index bits} = 01010101 = 85$$

- c. Suppose the byte with address 0001 1010 0001 1010 is stored in the cache. What are the addresses of the other bytes stored along with it in the same block?

0001 1010 0001 000
0001 1010 0001 001
0001 1010 0001 010
0001 1010 0001 011
0001 1010 0001 100
0001 1010 0001 101
0001 1010 0001 110
0001 1010 0001 111

⇒ The offset would change if the other addresses, the index and tag bits will remain same.

- d. Write two addresses that would map to same set in cache.

00010 00100011 011

11011 00100011 101

↳ The index bits need to be same.

- e. Calculate the size of the tag array?

As no. of sets = 2^8 and it is 2 way set associative.
so no. of ^{times} tag bits occur in cache line is 2.

$$\text{tag bits} = 5$$

$$\text{so, } 5 \times 2^8 \times 2 = 2560 \text{ bits.}$$

Question No. 5 [Marks: 15]

Suppose we have virtual memory in a system with page size of 4KB, Virtual address is 17 bits and physical address is 16 bits. TLB cache is fully associative with 4 entries and least recently used replacement policy. Following is the snapshot for Page table and TLB cache.

Tag	Valid	Physical Page #
00000		
00001	0	
00010	1	0011
00011	1	1001
00100	1	0000
00101	1	0010
00110	0	1010
00111	1	0100
01000	0	1011
01001	1	0101
01010	1	1000
01011	1	0110
01100	1	1111
01101	1	1101
01110	0	0111
01111	1	1110
01111	1	1100
01111	1	0001

TLB cache:

Valid	LRU	Tag	Physical Page #
1	3 4 1 2	01111	0001
1	4 1 2 3	00011	0010
1	2 3 4 1	01000	1000
1	1 2 3 4	00100	1010

- a. For each of the following addresses write the corresponding physical addresses and tell whether it is TLB hit/miss or page fault.

Assuming that TLB is not updated at this point, when a miss occurs.

	Virtual address	Physical address	TLB hit/miss	Page fault
A	01110001000100100	11000000100100	miss	no
B	00011101001100000	001010101100000	hit	no
C	01110010100010001	1100101001001	miss	no
D	00101000111000101	01000011100101	miss	yes
E	01000100110100011	10001011010011	hit	not

- b. In case of a TLB miss, new entry comes in the TLB cache replacing the least recently used value. Value of 1 in LRU in above table represents the most recently used value and 4 represent the least recently used value. If the above addresses are accessed in same sequence, what would be the final state of TLB cache?

Valid	LRU	Tag	Physical Page #
1	2	00011	0010
1	3	01110	1100
1	1	00101	0100
1	4	01000	1000

Note: In case of a page fault, you can change valid bit to 1 in page table, after making sure that same physical page number is not assigned to another virtual page number. In that scenario first change the valid bit of that entry to 0.

Question No. 6 [Marks: 10]

Each instruction fetch means a reference to the instruction cache and 35% of all instructions reference data memory (Load/Store instructions).

With the first implementation (separate instruction and data cache):

- The average miss rate in the L1 instruction cache was 2%
- The average miss rate in the L1 data cache was 10%
- In both cases, the miss penalty (time to access main memory) is 90 CCs

For the new design, with unified memory (same cache for instruction and data), the average miss rate is 3% for the cache as a whole, and the miss penalty is again 90 CCs.

a)

Which design is better and by how much?

for L1:- memory access stall
 for instruction cache $= I \times \frac{2}{100} \times 90 = 1.8I$, where I is instructions per program.

memory access stall
 for data cache $= I \times \frac{35}{100} \times \frac{10}{100} \times 90 = 3.15I$

So L1 Total memory access stall $= 1.8I + 3.15I = 4.95I$

for new cache:-
 memory access stall $= \left(I \times \frac{3}{100} \times 90 \right) + \left(I \times \frac{35}{100} \times \frac{3}{100} \times 90 \right) = 2.7I + 0.945I = 3.645I$

The new cache is better. by 1.358 times.

$\frac{4.95I}{3.645I} = 1.358$

b) If we add L2 cache in the new design with access time equal to 10cc, it reduces the miss rate to 1.5 % from 3 %. What is new average memory access time?

$$\begin{aligned}
 &= I * (3/100 * (10 + 35/100 * 10)) + (1.5/100 * (90 + 35/100 * 90)) \\
 &= I * (0.405 + 1.82) \\
 &= 2.228I
 \end{aligned}$$