1. (a) Two systems S1 and S2 have clock frequencies of 300 MHz and 200 MHz, respectively. A given program P consists of 2×10^9 instructions, which are distributed among two instruction types T1 and T2, each contributing 1×10^9 instructions. Cycles-Per-Instruction (CPI) of T1, while executing on S1 and S2, are 2 and 3, respectively. CPI of T2, while executing on S1 and S2, are 4 and 3, respectively.

(i) Determine the execution time of P on S1 and S2.

(8 marks)

Answer

Freq of S1 = 300 MHz
Freq of S2 = 200 MHz

P1 instructions = 2×10^9
T1 instructions = 1×10^9
T2 instructions = 1×10^9

CPI of T1 on S1 = 2
CPI of T1 on S2 = 3

CPI of T2 on S1 = 4
CPI of T2 on S2 = 3

Execution time $T = IC * CPI * Tc = \frac{IC * CPI}{Freq}$

T1 of P on S1 $= \frac{1\times10^9 * 2}{300\times10^6} = \frac{20}{3}$ sec

T2 of P on S1 $= \frac{1\times10^9 * 4}{300\times10^6} = \frac{40}{3}$ sec

Execution time of P on S1 $= \frac{20}{3} + \frac{40}{3} =$ 20 sec

T1 of P on S2 $= \frac{1\times10^9 * 3}{200\times10^6} =$ 15 sec

T2 of P on S2 $= \frac{1\times10^9 * 3}{200\times10^6} =$ 15 sec

Execution time of P on S2 = 15 + 15 = 30 sec

1. (a) (ii) Comment which system is slower after finding the speed-up of S1 over S2. (3 marks)

Answer

Speedup of S1 over S2 = $\frac{\text{Time}_{S2}}{\text{Time}_{S1}} = \frac{30}{20} = 1.5$

S1 is 1.5 times faster than S2. Therefore, S2 is slower.

1. (b) Briefly explain the working of the instructions "STUR X0, [X1, #8]" and "CBZ X2, #8". Figure Q1 shows the LEGv8 architecture, where one line is marked with "X". "X" indicates there is a hardware failure due to a broken bus. Indicate which of the instructions above will have issue in executing. Explain in detail the reason by indicating the path(s) taken by the instruction(s) with issue. (10 marks)

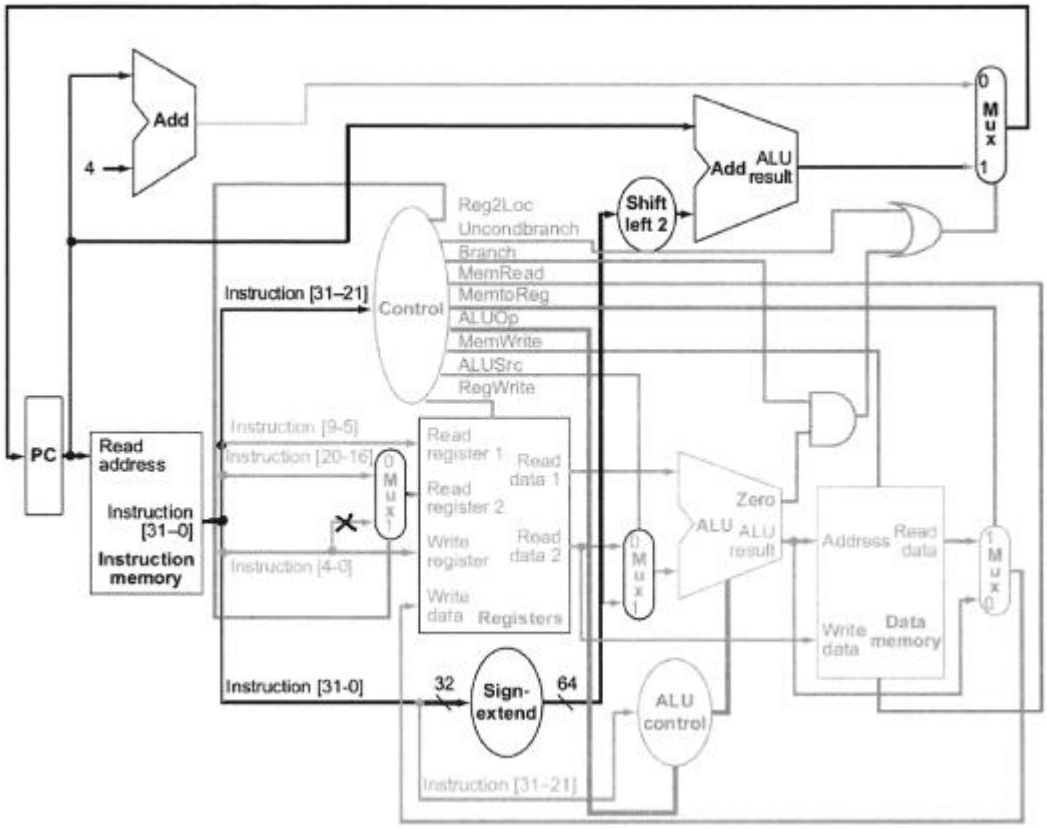
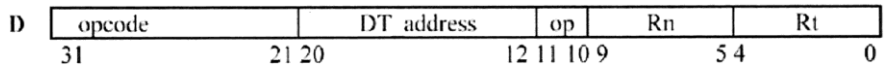


Figure Q1

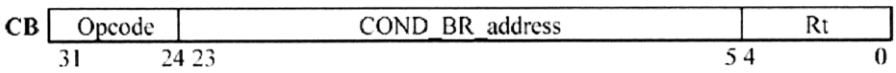
1. (b) Answer

Working of the Instructions



"LDUR X0, [X1, #8]"

1. Read the data from register X1.
2. Compute the memory address by adding the values of register X1 to the immediate value 8 using the ALU.
3. Write the values from the computed memory address to the register X0 to.
4. [X0] ← mem[[X1] + [8]]

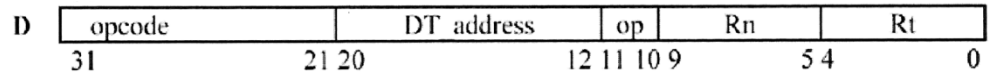


"CBZ X2, #8"

1. Read the data from register X0.
2. If values from X2 == 0, the PC is updated to the branch target address (current PC + #8<<2).
3. Otherwise, the next instruction is executed.
4. [new PC] ← PC + 8*4; if X2 = 0

Bus is broken

"STUR X0, [X1, #8]"



Instructions fetched:

Instructions [31-21] - opcode

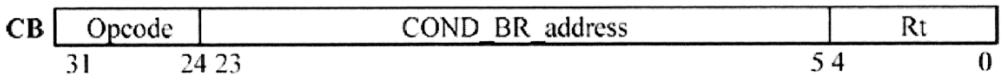
Instructions [20-12] - immediate value

Instructions [9-5] - read register 1 (X1)

Instructions [4-0] - write register (X0)

Since LDUR does not require to read register 2, hence LDUR has no issue.

"CBZ X0, #8"



Instructions fetched:

Instructions [31-24] - opcode

Instructions [23-5] - immediate value

Instructions [4-0] - read register 2 (X0)

As CBZ requires to read the data from register X0, it is unable to read the register when the bus is broken. Hence, CBZ will have issue in executing the instruction.

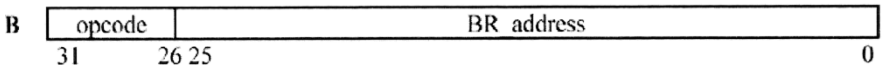
1. (c) With the example given in Table Q1, clearly explain the operation of instruction with reference to its addressing mode. You need to clearly emphasize the minimum and maximum range of branching (by suggesting a value for offset).

Table Q1

PC value	Instruction
0X10F0FF0C	B offset

(8 marks)

Answer



B offset is using PC-relative addressing mode.

Working instructions:
1. Fetch the immediate "offset value".
2. The PC is updated to the branch target address (current PC + "offset value"<<2).
3. [new PC] ← PC + "offset value"*4.

No. of address bits for unconditional branch = 25 – 0 + 1 = 26 bits

The maximum positive number for 26 bits
= 01 1111 1111 1111 1111 1111 1111 (0x1FFFFFF)

Maximum PC = PC + Sign extended (offset) << 2
= 0x10F0FF0C + (0x1FFFFFF)*4
= 0x18F0FF08

The maximum negative number for 26 bits
= 10 0000 0000 0000 0000 0000 0000 (0x2000000)

Minimum PC = PC + Sign extended (offset) << 2
= 0x10F0FF0C + (-0x2000000)*4
= 0x08F0FF0C

The maximum positive "offset values" is 0x01FFFFFF.
The maximum negative "offset value" is -0x02000000 (0xFE000000).

Hence, I would suggest a "offset values" within the range of:
0x01FFFFFF to -0x02000000 (0xFE000000).

2. Listing Q2 shows a code segment that is intended to be executed in a 5-stage pipelined LEGv8 processor. The program counter is updated with the branch target address at the Decode stage. No data forwarding is allowed but write-back and register-read operations of different instructions can be performed in the same clock cycle. Let the initial values in hexadecimal X8 be `0x0000000000001000` in hexadecimal (`CBZ`: *branch if equal to 0*).

Listing Q2

```
I1    loop: LDUR  X1,  [X8,  #0]
I2          ANDI  X2,  X1,   0xFFFF
I3          STUR  X2,  [X8,  #0]
I4          SUBI  X8,  X8,   0x0010
I5          CBZ   X8,  loop
I6          B     Finish

      Finish
```

(a) Calculate the steady-state CPI of the code segment in Listing Q2 with the help of a reservation table. Show the forwarded paths and the dependencies. Also find the total number of loop iterations.

(7 marks)

(b) The code segment shown in Listing Q2 is now intended to be executed in a two-way superscalar processor. In the superscalar processor, both ways can be used for all the instructions. Find the CPI achieved by the superscalar architecture. Note that full data forwarding is allowed.

(7 marks)

(c) Perform unrolling by a factor of 4 and do the necessary reordering. Use the unrolled and reordered code segment to be executed in a two-way superscalar machine. Find the CPI achieved by the superscalar architecture. Comment on the change in CPI when compared to Q2(b).

(10 marks)

(d) Briefly comment on the impact of applying static branch prediction of "Always Not Taken" in Q2(a).

(2 marks)

2 (a) Answer

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
loop: LDUR X1, [X8, #0]	I1.	F	D	E	M	W												
ANDI X2, X1, 0xFFFF	I2.		F	S	S	D	E	M	W									
STUR X2, [X8, #0]	I3.					F	S	S	D	E	M	W						
SUBI X8, X8, 0x0010	I4.								F	D	E	M	W					
CBZ X8, loop	I5.									F	S	S	D	E	M	W		
B Finish	I6.												S	F	D	E	M	W

Data dependency:
RAW: I1, I2 at X0
I2, I3 at X2
I4, I5 at X8

WAR: I1, I5 at X8

Assuming the single instruction negligible

$$\text{Steady state CPI} = \frac{\text{No. of instructions + No. of stalls + No. of control}}{\text{No. of instructions}}$$

$$= \frac{6 + 6 + 1}{6} = \frac{13}{6} = 2.17$$

X8 = 0x0000000000001000 (hex)
= 4096 (ten)

0x0010 (hex) = 16 (ten)

$$\text{Total number of loop iterations} = \frac{4096}{16} = 256$$

2 (c) Answer

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
loop: LDUR X1, [X8, #0]	I1.	F	D	E	M	W												
ANDI X2, X1, 0xFFFF	I2.		F	S	S	D	E	M	W									
STUR X2, [X8, #0]	I3.					F	S	S	D	E	M	W						
SUBI X8, X8, 0x0010	I4.								F	D	E	M	W					
CBZ X8, loop	I5.									F	S	S	D	E	M	W		
B Finish	I6.												S	F	D	E	M	W

Unrolling by 4

```
loop: LDUR  X1,  [X8,  #0]
      ANDI  X2,  X1,  0xFFFF
      STUR  X2,  [X8,  #0]
      LDUR  X3,  [X10, #-16]
      ANDI  X4,  X3,  0xFFFF
      STUR  X4,  [X10, #-16]
      LDUR  X5,  [X11, #-32]
      ANDI  X6,  X5,  0xFFFF
      STUR  X6,  [X11, #-32]
      LDUR  X7,  [X12, #-48]
      ANDI  X9,  X7,  0xFFFF
      STUR  X9,  [X12, #-48]
      SUBI  X8,  X8,  #64
      CBZ   X8,  loop
      B     Finish
```

```
I1
I2
I3
I4
I5
I6
I7
I8
I9
I10
I11
I12
I13
I14
I15
```

Unrolling by 4 and reordering

```
loop: LDUR  X1,  [X8,  #0]
      LDUR  X3,  [X10, #-16]
      LDUR  X5,  [X11, #-32]
      LDUR  X7,  [X12, #-48]
      ANDI  X2,  X1,  0xFFFF
      ANDI  X4,  X3,  0xFFFF
      ANDI  X6,  X5,  0xFFFF
      ANDI  X9,  X7,  0xFFFF
      STUR  X2,  [X8,  #0]
      STUR  X4,  [X10, #-16]
      STUR  X6,  [X11, #-32]
      STUR  X9,  [X12, #-48]
      SUBI  X8,  X8,  #64
      CBZ   X8,  loop
      B     Finish
```

	Way-1 (rest of instructions)	Way -2 (LDUR and STUR only)	Cycle
loop	nop	LDUR X1, [X8, #0]	1
	nop	LDUR X3, [X10, #-16]	2
	nop	LDUR X5, [X11, #-32]	3
	ANDI X2, X1, 0xFFFF	LDUR X7, [X12, #-48]	4
	ANDI X4, X3, 0xFFFF	nop	5
	ANDI X6, X5, 0xFFFF	nop	6
	ANDI X9, X7, 0xFFFF	STUR X2, [X8, #0]	7
	SUBI X8, X8, #64	STUR X4, [X10, #-16]	8
	nop	STUR X6, [X11, #-32]	9
	nop	STUR X9, [X12, #-48]	10
	CBZ X8, loop	nop	11
	B Finish	nop	12

2-way super-scalar after 4-unrolling and reordering

$$CPI = \frac{12}{15} = \frac{4}{5}$$

$$= 0.8$$

Combined effect of loop unrolling and instruction reordering helped us improve the performance of the system.

2 (d) Answer

- Execute successive instructions in sequence.
- For the conditional branch, if the branch is not taken, there will be no stall.
- Flush the pipeline and read correct instructions if branch actually taken.

3. (a) Name two different write policies for cache systems. Which policy has better performance considering the huge difference in speed between the cache and the main memory?

(5 marks)

Answer

Two basic cache write policies

- Write-through policy
- Write-back policy

Write-back policy has better performance

- CPU writes occur at the speed of the cache memory
- multiple writes within a block require only one write to the lower-level memory
- write back uses less memory bandwidth (good for multiprocessors) and dissipate less power (good for embedded applications)

3. (b) Consider a memory system with Byte-addressable main memory and 32-bit physical addresses. The cache system configuration is as follows:

- L1 Instruction Cache: 64 KB (1 KB = 1024 Bytes) in size, 128-Byte blocks, 4-way set associative cache, indexed and tagged with physical addresses.
- L1 Data Cache: 32 KB in size, 64-Byte blocks, direct mapped cache, indexed and tagged with physical addresses.
- Both caches keep the following information bits for each cache line: a dirty bit, a reference bit, and 3 permission bits.

Specify the number of offset, index, and tag bits for each of these structures, and compute the total size in number of bits for each of the tag arrays (including both information bits and tag bits) in Table Q3a.

Table Q3a

Structure	Tag Bits	Index Bits	Offset Bits	Size of Tag Array in Bits
L1 Instruction Cache				
L1 Data Cache				

(12 marks)

Answer

Address size = 32 bits

L1 Instruction Cache size = 64 KB
L1 Instruction Block size = 128 Bytes
L1 Instruction No. of way = 4

L1 Data Cache size = 32 KB
L1 Data Block size = 64 Bytes
L1 Data No. of way = 1

Dirty = 1 bit
Reference = 1 bit
Permission = 3 bits

L1 Instruction

No. of block $= \frac{\text{Cache size}}{\text{Block size}} = \frac{64 * 1024}{128} = 512$

No. of set $= \frac{\text{No. of block}}{\text{No. of way}} = \frac{512}{4} = 128$

Offset $= \log_2(\text{Block size}) = \log_2(128) =$ 7 bits
Index $= \log_2(\text{No. of set}) = \log_2(128) =$ 7 bits

Address size = Tag + Index + Offset
Tag = Address size – Index – Offset = 32 – 7 – 7 = 18 bits

L1 Tag Array = (Dirty + Reference + Permission + Tag) * Way * Set
= (1 + 1 + 3 + 18) * 4 * 128
= 11776 bits

L1 Data

No. of block $= \frac{\text{Cache size}}{\text{Block size}} = \frac{32 * 1024}{64} = 512$

No. of set $= \frac{\text{No. of block}}{\text{No. of way}} = \frac{512}{1} = 512$

Offset = $\log_2$(Block size) = $\log_2$(64) = 6 bits
Index = $\log_2$(No. of set) = $\log_2$(512) = 9 bits

Address size = Tag + Index + Offset
Tag = Address size – Index – Offset = 32 – 6 – 9 = 17 bits
L1 Tag Array = (Dirty + Reference + Permission + Tag) * Way * Set
= (1 + 1 + 3 + 17) * 1 * 512
= 11264 bits

Structure	Tag Bits	Index Bits	Offset Bits	Size of Tag Array in Bits
L1 Instruction Cache	18	7	7	11776
L1 Data Cache	17	9	6	11264

3. (c) Tables Q3b and Q3c show the results of cache miss rates and average memory access times when a system is configured with different cache sizes and block sizes. Assume a cache hit takes 2 clock cycles. The cache miss penalties with different block sizes are provided in Table Q3c.

Table Q3b: Cache Miss Rate v.s. Block Size

Block Size (Bytes)	Cache Size			
	4KB	16KB	64KB	256KB
16	9.37%	5.59%	2.86%	1.36%
64	X%	3.87%	1.43%	0.77%
256	7.52%	4.23%	1.58%	0.62%

Table Q3c: Average Memory Access Time (AMAT) v.s. Block Size

Block Size (Bytes)	Miss Penalty	Cache Size			
		4KB	16KB	64KB	256KB
16	42	5.94	4.35	3.20	Y
64	48	4.95	3.86	2.69	2.37
256	72	7.41	Z	3.14	2.45

(i) Compute the missing entries X, Y and Z in Tables Q3b and Q3c.

(5 marks)

Answer

AMAT = Hit time + Miss rate * Miss penalty

4.95 = 2 + X% * 42 ⇒ X = 7.02%

Y = 2 + 1.36% * 48 ⇒ Y = 2.65

Z = 2 + 4.23% * 72 ⇒ Z = 5.04

(ii) Between cache miss rate and average memory access time, which one is the more important performance metric to consider when designing a cache? Briefly justify your answer.

(2 marks)

Answer

Average memory access time (AMAT) is the more important performance to the cache performance because it considers both the time spent on cache hits and misses.

4. (a) Assume a branch predictor uses a two-bit prediction scheme as shown in Figure Q4a. The predictor is initialized to the state of "00". Consider a branch instruction with the following repeating sequence of actual outcome: 'T T T N T T", where T indicates the branch is taken and N indicates the branch is not taken.

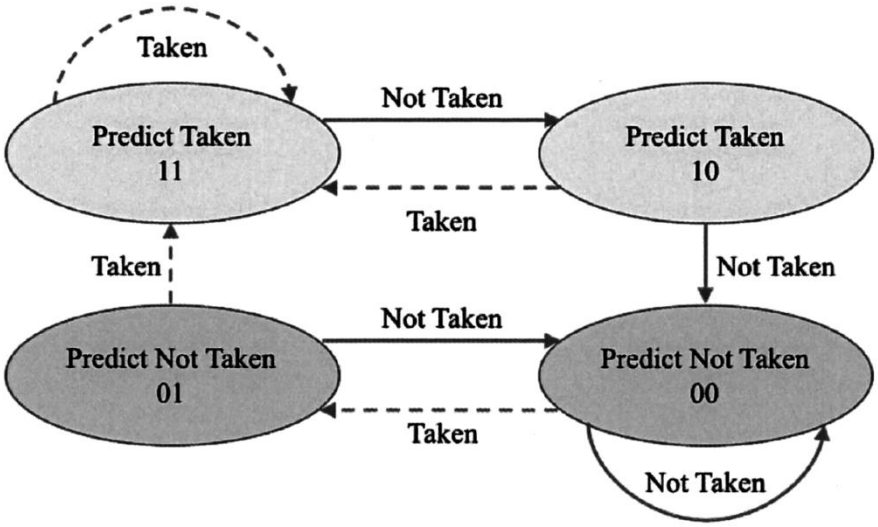


Figure Q4a

(i) Find the prediction accuracy of the two-bit predictor if the repeating sequence of actual outcome is repeated infinitely. Complete Table Q4 below to indicate the prediction decision in each step. The first column has been filled as a reference. What is the prediction accuracy?

Table Q4

Repeating Sequence	1						2					
Predictor State	00											
Prediction Decision	N											
Actual Outcome	T	T	T	N	T	T	T	T	T	N	T	T
Correct Prediction? (Yes/No)	N											

(7 marks)

4. (a) (i) Answer

Repeating Sequence	1						2					
Predictor State	00	01	11	11	10	11	11	11	11	11	10	11
Prediction Decision	N	N	T	T	T	T	T	T	T	T	T	T
Actual Outcome	T	T	T	N	T	T	T	T	T	N	T	T
Correct Prediction? (Yes/No)	N	N	Y	N	Y	Y	Y	Y	Y	N	Y	Y

Prediction accuracy = 8 / 12 = 66.67%

(ii) Compare the prediction accuracy with a static "Always Taken" predictor. Briefly comment on the better choice of predictor in this case using no more than 2 sentences.

(3 marks)

Answer

Prediction for “Always Taken” accuracy (all “T” correct predictions)
= 9 / 12 = 75%

The "Always Taken" predictor has better accuracy in this case because the majority of outcomes are taken (T).

(b) Briefly analyze how GPUs are designed to hide memory access latencies in no more than 4 sentences.

(4 marks)

Answer

- GPUs hide memory access latencies through thread-level parallelism.
- GPUs use warp scheduling to switch between different warps (groups of threads) to ensure that there is always work available to execute

4. (c) Figure Q4b shows a CUDA kernel that runs on a GPU to compute the dot product of two vectors A and B and outputs a scalar value C:

$$C = \mathbf{A} \bullet \mathbf{B} = \sum_{i=1}^{N} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_N b_N$$

```
Line
1  __global__ void dot_prod(int N, int *a, int *b, int *c){
2    __shared__ int temp[N];
3    int i = blockIdx.x;
4    temp [i] = a[i]*b[i];
5
6    // Thread 0 sums the pairwise products
7    if (i == 0) {
8       int sum = 0;
9       for (int j = 0; j < N; j++)
10        sum += temp [j];
11      *c = sum;
12   }
13 }
```

Figure Q4b

(i) Identify two mistakes in the CUDA C code in Figure Q4b that are related to GPU programming only (ignore other general programming bugs, if any). Indicate the correct CUDA C code to fix the mistakes and make the kernel function correctly.

(6 marks)

Answer

Line 3: should use `threadIdx.x` to ensure each thread gets a unique index

Line 5: `__syncthreads()` is not used to synchronizes all threads within a block

Fixed code:

Line 3: `int i = threadIdx.x;`

Line 5: `__syncthreads();`

4. (c) (ii) If N = 32, indicate the CUDA C code to launch the kernel with the number of block(s) and thread(s) to be created. Briefly justify how the number of blocks and threads are determined in no more than 4 sentences.

(5 marks)

Answer

- Launch code: dot_prod<<<1, 32>>>(N, d_a, d_b, d_c);

- There are 32 elements so we would need 32 threads
- A thread block can contain up to 1024 threads
- Launching only one thread block avoids the need for additional synchronization between blocks
- No. of warps = 32 / 32 = 1, hence it is aligned with GPU warp sizes

Appendix – Instruction Formats