1. (a) In the context of optimizing machine performance, an engineer at Company C introduces enhancement E1, which accelerates 45% of the original instructions by a factor of 3. Concerned about the complexity and cost effectiveness of E1, the management proposes an alternative enhancement E2. If E2 is applied to an as-yet-undetermined fraction of the original instructions, and speeds them up by a factor of 2, what percentage of all instructions should be optimized using enhancement E2 to achieve an equivalent overall speedup as obtained with enhancement E1.

(8 marks)

Answer

E1 = 45% = 0.45

S1 = 3

$$\text{Speedup (E1, S1)} = \frac{1}{(1 - E1) + \frac{E1}{S1}} = \frac{1}{(1 - 0.45) + \frac{0.45}{3}} = \frac{10}{7}$$

S2 = 2

$$\text{Speedup (E2, S2)} = \frac{1}{(1 - E2) + \frac{E2}{S2}} = \frac{1}{(1 - E2) + \frac{E2}{2}}$$

E2 speedup equivalent to E1,

$$\frac{1}{(1 - E2) + \frac{E2}{2}} = \frac{10}{7}$$

$$7 = 10(1 - E2) + 5(E2)$$

$$7 = 10 - 10(E2) + 5(E2)$$

$$-3 = -5(E2)$$

$$E2 = \frac{3}{5}$$

$$= 0.6$$

= 60%

Hence 60% of all instructions should be optimized using enhancement E2

1. (b) The hexadecimal value of the current content of the program counter (PC) in a LEGv8 processor is 0x100100A8 as shown in Table Q1b. The code intends to change the PC value to 0x100000FC. State the addressing mode used and calculate the required offset (loop value) in hexadecimal for the control instruction.

Find the maximum address of the instruction memory to which the control of execution of a LEGv8 code could be moved backward by the unconditional control instruction.

Table Q1b

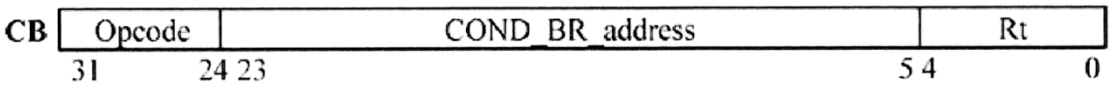
Program counter value in hexadecimal	Instruction
0x100000FC	Loop: LDUR X5, [X2, #0]
…	…
0x100100A8	CBNZ X1, Loop

(9 marks)

Answer

Conditional branch is using PC-relative addressing mode.

Offset (loop value) = (0x100000FC – 0x100100A8) /4
= 0xFFFF0054 / 4
= 0xFFFFC015



No. of address bits for conditional branch = 23 – 5 + 1 = 19 bits

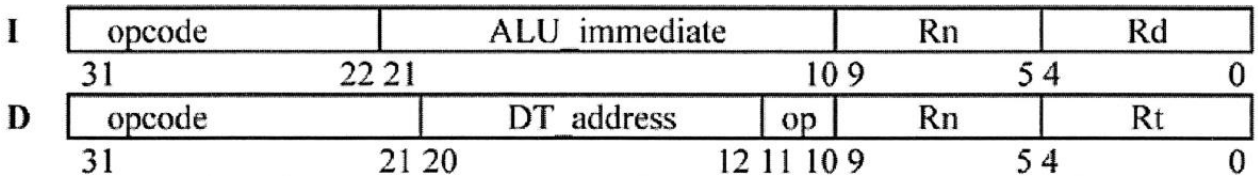
The maximum negative number for 19 bits
= 100 0000 0000 0000 0000 (0x40000)

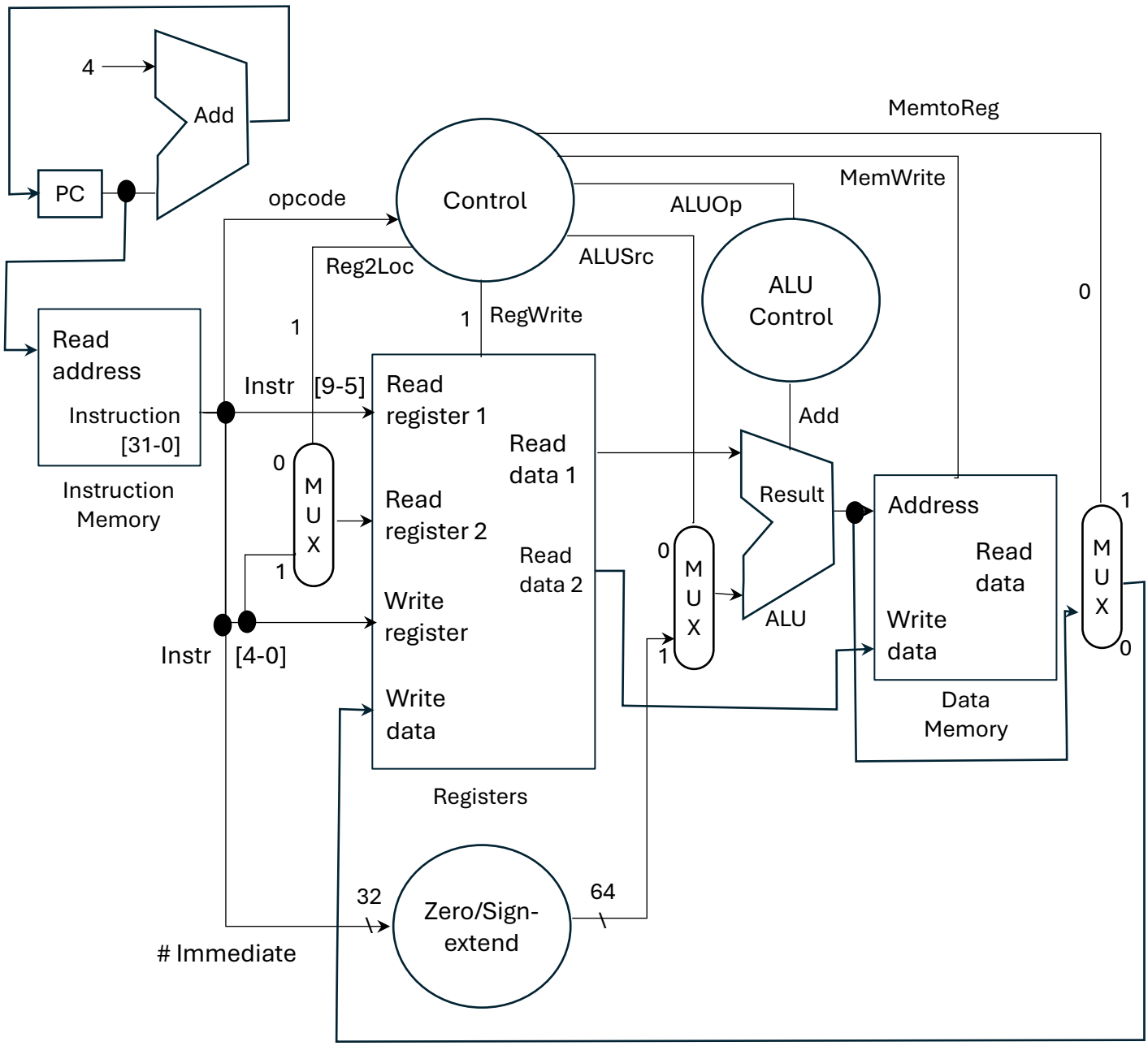
Maximum address backward PC = PC + Sign extended (offset) << 2
= 0x100100A8 + (-0x40000)*4
= 0x0FF100A8

1. (c) Use a neat diagram to show the datapath of a single-cycle architecture that supports the execution of both "XORI X5, X6, #5" and "STUR X3, [X4, #16]". Note that the datapath needs to have only minimal number of multiplexers (control signals can be simplified).

(8 marks)

Answer





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 Execute stage. Let the initial values in hexadecimal be X7=0x0000000000010000 and X8=0x0000000000001100 (CBNZ: *branch if not equal to 0*).

Listing Q2

```
I1    loop: LDUR  X0,  [X7,  #0]
I2          LDUR  X1,  [X8,  #0]
I3          ADD   X2,  X1,   X0
I4          XORI  X3,  X2,   0x00F
I5          STUR  X3,  [X7,  #0]
I6          ADDI  X7,  X7,   #8
I7          SUBI  X8,  X8,   #16
I8          CBZ   X8,  Finish
I9          B     loop

      Finish
```

(a) Calculate the steady-state Cycles Per Instruction (CPI) of the code segment in Listing Q2 with the help of a reservation table for the execution of the code if full data forwarding is allowed. Show the forwarded paths and the dependencies. Find the total number of loop iterations.

(8 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 2 and do necessary reordering. Use the unrolled and reordered code segment to be executed in a two-way superscalar machine. 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. Comment on the change in CPI when compared to Q2(b).

(10 marks)

2 (a) Answer

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
loop: LDUR X0, [X7, #0]	I1.	F	D	E	M	W											
LDUR X1, [X8, #0]	I2.		F	D	E	M	W										
ADD X2, X1, X0	I3.			F	D	S	E	M	W								
XORI X3, X2, 0x00F	I4.				F		D	E	M	W							
STUR X3, [X7, #0]	I5.						F	D	E	M	W						
ADDI X7, X7, #8	I6.							F	D	E	M	W					
SUBI X8, X8, #16	I7.								F	D	E	M	W				
CBZ X8, Finish	I8.									F	D	E	M	W			
B loop	I9.										S	S	F	D	E	M	W

Data dependency:
RAW: I1, I3 at X0
I2, I3 at X1
I3, I4 at X2
I4, I5 at X3
I7, I8 at X8

WAR: I1, I6 at X7
I2, I7 at X8
I5, I6 at X7

Assuming the single instruction negligible

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

$$= \frac{9 + 1 + 2}{9} = \frac{4}{3} = 1.33$$

X8 = 0x0000000000001100 (hex)
= 4352 (ten)

$$\text{Total number of loop iterations} = \frac{4352}{16} = 272$$

2 (b) Answer

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Loop: LDUR X0, [X7, #0]	I1.	F	D	E	M	W											
LDUR X1, [X8, #0]	I2.		F	D	E	M	W										
ADD X2, X1, X0	I3.			F	D	S	E	M	W								
XORI X3, X2, 0x00F	I4.				F		D	E	M	W							
STUR X3, [X7, #0]	I5.						F	D	E	M	W						
ADDI X7, X7, #8	I6.							F	D	E	M	W					
SUBI X8, X8, #16	I7.								F	D	E	M	W				
CBZ X8, Finish	I8.									F	D	E	M	W			
B loop	I9.										S	S	F	D	E	M	W

	Way-1 (rest of instructions)	Way -2 (LDUR and STUR only)	Cycle
Loop	nop	LDUR X0, [X7, #0]	1
	nop	LDUR X1, [X8, #0]	2
	SUBI X8, X8, #16	nop	3
	ADD X2, X1, X0	nop	4
	XORI X3, X2, 0x00F	nop	5
	CBZ X8, Finish	STUR X3, [X7, #0]	6
	ADDI X7, X7, #8	nop	7
	B loop	nop	8

2-way super-scalar after instruction reordering

$$CPI = \frac{8}{9}$$

$$= 0.89$$

2 (c) Answer

```
loop: LDUR  X0,  [X7, #0]
      LDUR  X1,  [X8, #0]
      ADD   X2,  X1,  X0
      XORI  X3,  X2,  0x00F
      STUR  X3,  [X7, #0]
      ADDI  X7,  X7,  #8
      SUBI  X8,  X8,  #16
      CBZ   X8,  Finish
      B     loop
```

Unrolling by 2

```
loop: LDUR  X0,  [X7,  #0]
      LDUR  X1,  [X8,  #0]
      ADD   X2,  X1,  X0
      XORI  X3,  X2,  0x00F
      STUR  X3,  [X7,  #0]
      LDUR  X4,  [X10, #8]
      LDUR  X5,  [X11, #-16]
      ADD   X6,  X5,  X4
      XORI  X9,  X6,  0x00F
      STUR  X9,  [X10, #8]
      ADDI  X7,  X7,  #16
      SUBI  X8,  X8,  #32
      CBZ   X8,  Finish
      B     loop
```

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

Unrolling by 2 and reordering

```
loop: LDUR  X0,  [X7,  #0]
      LDUR  X1,  [X8,  #0]
      LDUR  X4,  [X10, #8]
      LDUR  X5,  [X11, #-16]
      ADD   X2,  X1,  X0
      ADD   X6,  X5,  X4
      XORI  X3,  X2,  0x00F
      XORI  X9,  X6,  0x00F
      STUR  X3,  [X7,  #0]
      STUR  X9,  [X10, #8]
      ADDI  X7,  X7,  #16
      SUBI  X8,  X8,  #32
      CBZ   X8,  Finish
      B     loop
```

	Way-1 (rest of instructions)	Way -2 (LDUR and STUR only)	Cycle
loop	nop	LDUR X0, [X7, #0]	1
	nop	LDUR X1, [X8, #0]	2
	SUBI X8, X8, #32	LDUR X4, [X10, #8]	3
	ADD X2, X1, X0	LDUR X5, [X11, #-16]	4
	XORI X3, X2, 0x00F	nop	5
	ADD X6, X5, X4	STUR X3, [X7, #0]	6
	XORI X9, X6, 0x00F	nop	7
	CBZ X8, Finish	STUR X9, [X10, #8]	8
	ADDI X7, X7, #16	nop	9
	B loop	nop	10

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

$$CPI = \frac{10}{14} = \frac{5}{7}$$

$$= 0.71$$

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

3. (a) Name two cache organization schemes.

(4 marks)

Answer

- Direct-mapped cache
- Fully associative cache
- Set associative cache

(b) Consider a memory system composed of a 2 GB Byte-addressable main memory, and a 64 MB four-way set-associative cache with a block size of 512 Bytes. Determine the width of the address (in number of bits), and the widths of the tag, index, and offset fields in the address, respectively.

(8 marks)

Answer

Memory size = 2 GB
Block size = 512 B
Cache size = 64 MB
No. of way = 4

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

$$\text{No. of set} = \frac{\text{No. of block}}{\text{No. of way}} = \frac{128 * 1024}{4} = 32 \text{ KB}$$

Address size $= \log_2(\text{Memory size}) = \log_2(2 * 1024 * 1024 * 1024) =$ 31 bits
Offset $= \log_2(\text{Block size}) = \log_2(512) =$ 9 bits
Index $= \log_2(\text{No. of set}) = \log_2(32 * 1024) =$ 15 bits

Address size = Tag + Index + Offset
Tag = Address size – Index – Offset
= 31 - 9 - 15
= 7 bits

Address

Tag	Index	Offset
7	15	9

(c) Following Q3(b), what is the miss rate of the cache when a program accesses the main memory in a totally random fashion, that is, random memory addresses are uniformly accessed for a sufficiently long time?

(6 marks)

Answer

$$\text{Hit rate} = \frac{\text{Cache size}}{\text{Memory size}} = \frac{64 * 1024 * 1024}{2 * 1024 * 1024 * 1024} = \frac{1}{32}$$

$$\text{Miss rate} = 1 - \text{Hit rate} = 1 - \frac{1}{32} = \frac{31}{32} = 0.96875 = 96.875\%$$

3. (d) Tables Q3a and Q3b show the results of cache miss rate and average memory access time when a system is configured with different cache block sizes. Assume a cache hit takes 2 clock cycles. The cache miss penalties with different block sizes are provided in Table Q3b.

(i) Compute the missing entries X, Y and Z in Tables Q3a and Q3b.

(5 marks)

Table Q3a: Cache Miss Rate vs. Block Size

Block Size (Bytes)	Cache Size			
	4KB	16KB	64KB	256KB
16	10.95%	**X%**	1.95%	0.79%
64	6.82%	2.39%	1.02%	0.47%
256	8.36%	2.93%	0.86%	0.32%

Table Q3b: Average Memory Access Time (AMAT) vs. Block Size

Block Size (Bytes)	Miss Penalty	Cache Size			
		4KB	16KB	64KB	256KB
16	62	8.79	3.96	3.21	2.49
64	68	6.64	3.63	2.69	**Z**
256	92	**Y**	4.70	2.79	2.29

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

(2 marks)

Answer

(i) AMAT = Hit time + Miss rate * Miss penalty

3.96 = 2 + X% * 62 ⇒ X = 3.16%
Y = 2 + 8.36% * 92 ⇒ Y = 9.69
Z = 2 + 0.47% * 68 ⇒ Z = 2.32

(ii) 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) List the four types of processor architectures in the processor taxonomy according to Flynn's classification. Which of them are designed to take instruction-level parallelism into account?

(6 marks)

Answer

- Single instruction, single data stream – SISD
- Single instruction, multiple data stream – SIMD
- Multiple instruction, single data stream – MISD
- Multiple instruction, multiple data stream – MIMD

SISD and MIMD are designed to take instruction-level parallelism into account.

(b) Briefly explain the usage of the specifiers `__global__` and `__shared__` in CUDA C programming (in no more than 4 sentences).

(4 marks)

Answer

- `__global__` is used to declare a kernel function as device code
  - The kernel function is called from the host

- `__shared__` is used to declare a variable/array in shared memory
  - So that data is shared between threads in a block

4. (c) Assume that there is only one branch instruction in a program. Consider the following sequence of actual outcomes of the branch instruction (N, N, N, N, N, N, N, N, N, N, T, T, N, N, T, N), where T means that the branch is taken, and N means that the branch is not taken. What is the prediction accuracy for the last 6 occurrences of this branch, i.e., (T, T, N, N, T, N), if the 2-bit branch predictor as shown in Figure Q4a is applied where the initial state of the predictor is unknown?

(6 marks)

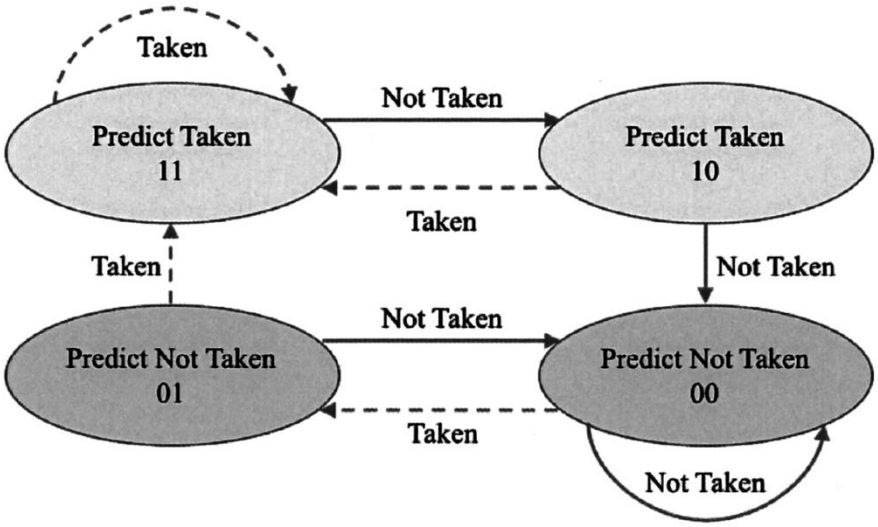


Figure Q4a

Answer

After the first 10 outcomes are not taken (N), we can assume that the state will be at "Predict Not Taken 00".

Predictor State	00	01	11	10	00	01
Prediction Decision	N	N	T	T	N	N
Actual Outcome	T	T	N	N	T	N
Correct Prediction?	N	N	N	N	N	Y

Prediction accuracy = 1 / 6 = 16.67%

4. (d) The code snippet in Figure Q4b shows a C program that uses a CUDA kernel `saxpy()` to compute the SAXPY (Single precision A.X plus Y) operation:

$$\mathbf{Y} = A\mathbf{X}+\mathbf{Y}$$

where A is a scalar, while **X** and **Y** are vectors each consisting of `N` floating-point numbers.

```
Line
1   __global__
2   void saxpy(int n, float a, float *x, float *y){
3     int i = blockIdx.x * blockDim.x + threadIdx.x;
4     if (i < n)
5        y[i] = a*x[i] + y[i];
6     }
7
8   int main(void){
9     int N = 1024;     // size of vectors X and Y
10    float A = 7.0;
11    X = (float *)malloc(N*sizeof(float));
12    Y = (float *)malloc(N*sizeof(float));
13    // get values of vector X and Y
:        :
:        :
n     saxpy<<<.....>>>(N, A, d_X, d_Y);
:        :
:     return 0;
n+k   }
```

Figure Q4b

(i) Complete the code shown in Line ***n*** if the number of threads per block is set as 128. (Hint: indicate the necessary parameters.)

(4 marks)

Answer

No. of threads per block = 128

No. of block $= \frac{1024}{128} = 8$

Code = `saxpy<<<8,128>>>(N, A, d_X, d_Y);`

4. (d) (ii) Following Q4(d)(i), assume a Stream Multiprocessor (SM) in a GPU has sufficient register and shared memory resources to reside all the locks. What is the total number of warps that will be created by launching the kernel?

(5 marks)

Answer

No. of threads per block = 128

No. of warp per block $= \frac{128}{32} = 4$

No. of block = 8

Total no. of warps = No. of warp * No. of block
= 4 * 8
= 32 warps

Appendix – Instruction Formats