SC3050 Examination 2021-2022 Semester 2

1. (a) If fraction T of a program is enhanced by a factor of F when executed on an enhanced machine, derive the expression of speedup over the original machine. Based on that expression, find the value of speedup with T=0.4, for F=1000.

(4 marks)

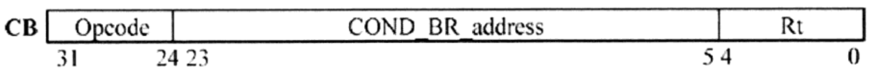
Answer

$$\text{Speedup} = \frac{T_{\text{unenhanced}}}{T_{\text{enhanced}}} = \frac{T_{\text{original}}}{T_{\text{enhanced}}}$$

$$\text{Speedup (T, F)} = \frac{1}{(1 - T) + \frac{T}{S}} = \frac{1}{(1 - 0.4) + \frac{0.4}{1000}} = \frac{2500}{1501} = 1.67$$

(b) State the addressing mode for conditional branching. Determine the minimum and maximum possible values to which an instruction "CBZ X31, address" can branch for a LEGv8 architecture, given that the content of the 64-bit program counter value of the branch instruction is 0xF000 0000 000C.

(9 marks)



Answer

Conditional branch is using PC-relative addressing mode.

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

The maximum positive number for 19 bits

= 011 1111 1111 1111 1111 (0x3FFFF)

Maximum PC = PC + Sign extended (offset) << 2

= 0xF000 0000 000C + (-0x3FFFF)*4

= 0xF000 0010 0008

The maximum negative number for 19 bits

= 100 0000 0000 0000 0000 (0x40000)

Minimum PC = PC + Sign extended (offset) << 2

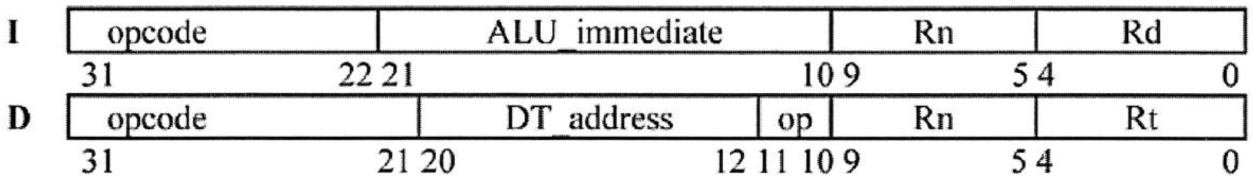
= 0xF000 0000 000C + (-0x40000)*4

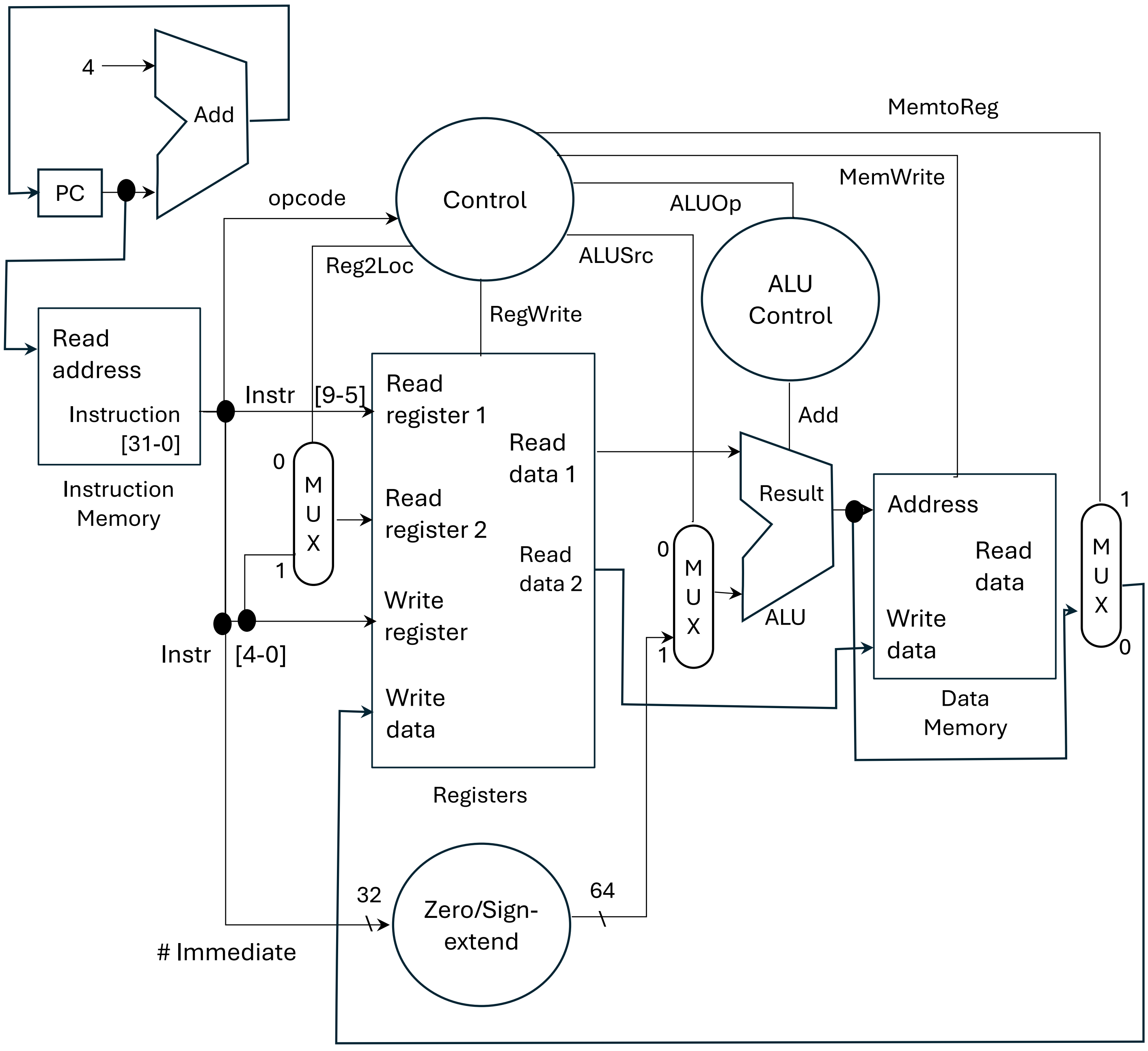
= 0xEFFF FFF0 000C

1. (c) Use a neat diagram to show the datapath of a single-cycle architecture that supports the execution of both "STUR X0, [X1, #8]" and "ADDI X4, X3, #5". Note that the datapath needs to have only minimal number of multiplexers (control signals can be simplified). Also briefly explain the working of the instructions "STUR X0, [X1, #8]" and "ADDI X4, X3, #5".

(12 marks)

Answer





"STUR 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 register X0 to the computed memory address.
4. [X0] $\rightarrow$ mem[[X1] + [8]]

"ADDI X4, X3, #5"

1. Read the data from register X3.
2. Compute the result by adding the values from register X3 to the immediate value 5 using the ALU.
3. Write the computed result to the register X4.
4. [X4] $\leftarrow$ [X3] + [5]

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. Note that, 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 be `X4=0x0000000000000000`, `X7=0x0000000010000000` and `X8=0x0000000000001000`.

Listing Q2

```
I1   loop: LDUR  X1,  [X7,  #0]
I2         LDUR  X2,  [X8,  #0]
I3         ADD   X3,  X2,   X1
I4         ADD   X4,  X4,   X3
I5         SUBI  X7,  X7,   #8
I6         SUBI  X8,  X8,   #8
I7         CBZ   X8,  Finish

    Finish
```

(a) Spot all the flow dependencies in Listing Q2. Also find the total number of loop iterations.

(4 marks)

Answer

Data dependency:

RAW:	I1, I3 at X1
	I2, I3 at X2
	I3, I4 at X3
	I6, I7 at X8
WAR:	I1, I5 at X7

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

Total number of loop iterations $= \frac{4096}{8} = 512$

2. (b) The code segment shown in Listing Q2 is now intended to be executed in a two-way superscalar processor. In the superscalar processor, one way is exclusively for load and store instructions whereas the other way can execute all instructions except load and store. Find the CPI achieved for the code segment shown in Listing Q2 using superscalar architecture.

(7 marks)

Answer

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Loop: LDUR X1, [X7, #0]	I1.	F	D	E	M	W														
LDUR X2, [X8, #0]	I2.		F	D	E	M	W													
ADD X3, X2, X1	I3.			F	S	S	D	E	M	W										
ADD X4, X4, X3	I4.						F	S	S	D	E	M	W							
SUBI X7, X7, #8	I5.									F	D	E	M	W						
SUBI X8, X8, #8	I6.										F	D	E	M	W					
CBZ X8, Finish	I7.											F	S	S	D	E	M	W		
Finish															S	F	D	E	M	W

	Way-1 (rest of instructions)	Way -2 (LDUR and STUR only)	Cycle
Loop	nop	LDUR X1, [X7, #0]	1
	SUBI X7, X7, #8	LDUR X2, [X8, #0]	2
	SUBI X8, X8, #8	nop	3
	nop	nop	4
	ADD X3, X2, X1	nop	5
	CBZ X8, Finish	nop	6
	nop	nop	7
	ADD X4, X4, X3	nop	8

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

$$= \frac{7+6+1}{7} = \frac{14}{7} = 2$$

2-way super-scalar after instruction reordering

$$\text{CPI} = \frac{8}{7}$$

$= 1.14$

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Loop: LDUR X1, [X7, #0]	I1.	F	D	E	M	W														
LDUR X2, [X8, #0]	I2.		F	D	E	M	W													
ADD X3, X2, X1	I3.			F	S	S	D	E	M	W										
ADD X4, X4, X3	I4.						F	S	S	D	E	M	W							
SUBI X7, X7, #8	I5.									F	D	E	M	W						
SUBI X8, X8, #8	I6.										F	D	E	M	W					
CBZ X8, Finish	I7.											F	S	S	D	E	M	W		
Finish															S	F	D	E	M	W

2. (c) Perform loop unrolling by a factor of 2 for the code segment in Listing Q2 and do the necessary reordering of instructions to reduce the number of stall cycles to the minimum. You are allowed to use new temporary registers to get rid of hazards.

(6 marks)

Answer

Unrolling by 2

```
loop: LDUR  X1,   [X7,  #0]
      LDUR  X2,   [X8,  #0]
      ADD   X3,   X2,   X1
      ADD   X4,   X4,   X3
      LDUR  X5,   [X11, #-8]
      LDUR  X6,   [X12, #-8]
      ADD   X9,   X6,   X5
      ADD   X10,  X10,  X9
      SUBI  X7,   X7,   #16
      SUBI  X8,   X8,   #16
      CBZ   X8,   Finish
```

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

Unrolling by 2 and reordering

```
loop: LDUR  X1,   [X7,  #0]
      LDUR  X2,   [X8,  #0]
      LDUR  X5,   [X11, #-8]
      LDUR  X6,   [X12, #-8]
      ADD   X3,   X2,   X1
      SUBI  X7,   X7,   #16
      ADD   X9,   X6,   X5
      SUBI  X8,   X8,   #16
      ADD   X4,   X4,   X3
      ADD   X10,  X10,  X9
      CBZ   X8,   Finish
```

No. of stalls
= 2 + 2 + 2 + 2 + 2 + 1
= 11

No. of stalls
= 1

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

$$= \frac{11 + 0 + 1}{11} = \frac{12}{11}$$

= 1.09

2. (d) The unrolled and reordered code segment derived in Q2(c) is now intended to be executed in a two-way superscalar processor. As mentioned earlier, one way is exclusively for load and store instructions whereas the other way can execute all instructions except load and store.

Find the CPI achieved for the unrolled and reordered code segment derived in Q2(c) using superscalar architecture. Comment on the performance enhancement.

(8 marks)

Answer

```
loop: LDUR  X1,   [X7,  #0]
      LDUR  X2,   [X8,  #0]
      LDUR  X5,   [X11, #-8]
      LDUR  X6,   [X12, #-8]
      ADD   X3,   X2,   X1
      SUBI  X7,   X7,   #16
      ADD   X9,   X6,   X5
      SUBI  X8,   X8,   #16
      ADD   X4,   X4,   X3
      ADD   X10,  X10,  X9
      CBZ   X8,   Finish
```

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

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

$$CPI = \frac{10}{11}$$

= 0.9

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

3. (a) Name two basic cache write policies.

(4 marks)

Answer

Two basic cache write policies

- Write-through policy
- Write-back policy

(b) Consider a hierarchical memory system composed of an 8 GB Byte-addressable main memory, and a 16 MB eight-way set-associative cache with block size of 256 Bytes. Determine the lengths (in number of bits) of the tag, index, and offset fields in the address, respectively. (1 GB = 1024 MB; 1 MB = 1024 KB; 1 KB = 1024 Bytes)

(8 marks)

Answer

Memory size	= 8 GB
Block size	= 256 B
Cache size	= 16 MB
No. of way	= 8

$$\text{No. of block} = \frac{\text{Cache size}}{\text{Block size}} = \frac{16 * 1024 * 1024}{256} = 64\text{ KB}$$

$$\text{No. of set} = \frac{\text{No. of block}}{\text{No. of way}} = \frac{64 * 1024}{8} = 8\text{ KB}$$

Address size = $\log_2$(Memory size) = $\log_2(8 * 1024 * 1024 * 1024)$ = 33 bits

Offset = $\log_2$(Block size) = $\log_2(256)$ = 8 bits

Index = $\log_2$(No. of set) = $\log_2(8 * 1024)$ = 13 bits

Address size = Tag + Index + Offset

Tag = Address size – Index – Offset

= 33 - 8 - 13

= 12 bits

Tag	Index	Offset
12	13	8

3. (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

$$\text{Hit rate} = \frac{\text{Cache size}}{\text{Memory size}} = \frac{16 * 1024 * 1024}{8 * 1024 * 1024 * 1024} = \frac{1}{512}$$

$$\text{Miss rate} = 1 - \text{Hit rate} = 1 - \frac{1}{512} = \frac{511}{512} = 0.9998 = 99.8\%$$

(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 1 clock cycle. The cache miss penalties with different block sizes are provided in Table Q3b.

Table Q3a: Cache Miss Rate vs. Block Size

Block Size	Cache Size			
(Bytes)	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	Miss	Cache Size			
(Bytes)	Penalty	4KB	16KB	64KB	256KB
16	82	9.98	3.96	2.60	1.65
64	88	7.00	3.10	1.90	**Z**
256	112	**Y**	4.28	1.96	1.36

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

(5 marks)

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

$3.96 = 1 + X\% * 82 \Rightarrow X = 3.61\%$

$Y = 1 + 8.36\% * 112 \Rightarrow Y = 10.36$

$Z = 1 + 0.47\% * 88 \Rightarrow Z = 1.41$

(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

(ii) More important performance metric to consider is average memory access time.

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 data-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

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

(b) Briefly discuss the main differences between the typical architecture designs of a CPU and a GPU (in no more than 6 sentences).

(6 marks)

Answer

CPU	GPU
CPU is optimized for low-latency operations and sequential processing.	GPU is optimized for high-throughput operations and parallel processing.
CPU has fewer but powerful cores (2 – 16).	GPU has many simpler cores (hundreds to thousands).
CPU is designed for general-purpose computing.	GPU is designed for parallel processing, such as graphics rendering and scientific computations.

(c) 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
    - Called from the host

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

4. (d) The code snippet in Figure Q4 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 = 8191;     // size of vectors X and Y
10     float A = 2.0;
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 Q4

(i) Complete the code shown in Line ***n*** if the number of threads per block is set as 512.

(4 marks)

Answer

No. of threads per block = 512

No. of block $= \dfrac{8191}{512} = 15.998 \approx 16$

Code = `saxpy<<<16,512>>>(N, A, d_X, d_Y);`

4. (d) (ii) 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

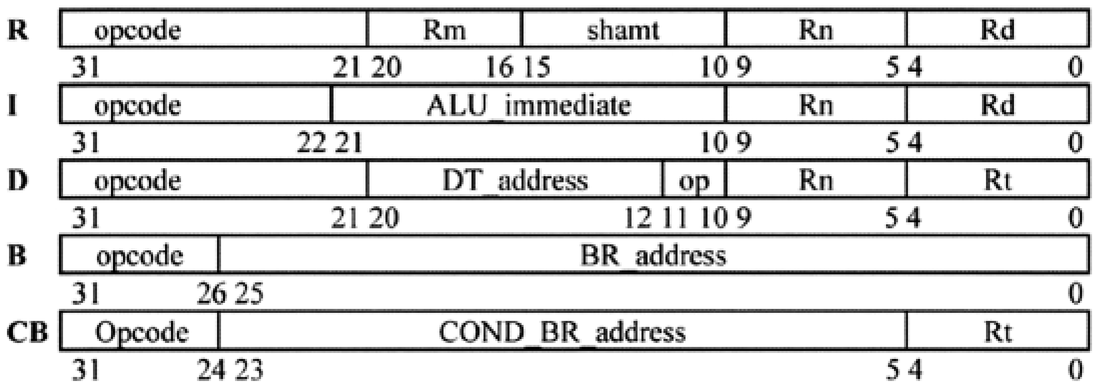
No. of threads per block = 512

No. of warps per block $= \frac{512}{32} = 16$

No. of block = 16

Total no. of warps = No. of warp * No. of block
= 16 * 16
= 256 warps

Appendix – Instruction Formats


R
opcode
Rm
shamt
Rn
Rd
31
21 20
16 15
10 9
5 4
0
I
opcode
ALU_immediate
Rn
Rd
31
22 21
10 9
5 4
0
D
opcode
DT_address
op
Rn
Rt
31
21 20
12 11 10 9
5 4
0
B
opcode
BR_address
31
26 25
0
CB
Opcode
COND_BR_address
Rt
31
24 23
5 4
0