21st CSEC – Past Year Paper Solution (2020 – 2021 Semester 2)
CE/CZ 3001 – Advanced Computer Architecture

1 (a)

$$\text{Speed-up of M1 over M2} = \frac{\text{Execution Time}_{M2}}{\text{Execution Time}_{M1}} = 2$$

$$\Rightarrow \text{Execution Time}_{M2} = 2 \cdot \text{Execution Time}_{M1}$$

According to Amdahl's Law,

$$\text{Execution Time}_{P1} = \frac{(\text{Execution Time}_{M1})/3}{2} + \frac{(\text{Execution Time}_{M1}) \cdot 2}{3} = \frac{(\text{Execution Time}_{M1}) \cdot 5}{6}$$

$$\text{Execution Time}_{P2} = \frac{(\text{Execution Time}_{M2})/2}{3} + \frac{(\text{Execution Time}_{M2})}{2} = \frac{(\text{Execution Time}_{M2}) \cdot 4}{6} = \frac{(\text{Execution Time}_{M1}) \cdot 4}{3}$$

$$\text{Speed-up of P1 over P2} = \frac{\text{Execution Time}_{P2}}{\text{Execution Time}_{P1}} = \frac{4/3}{5/6} = \frac{8}{5} = 1.6$$

Answer: The speed-up of P1 over P2 is 1.6

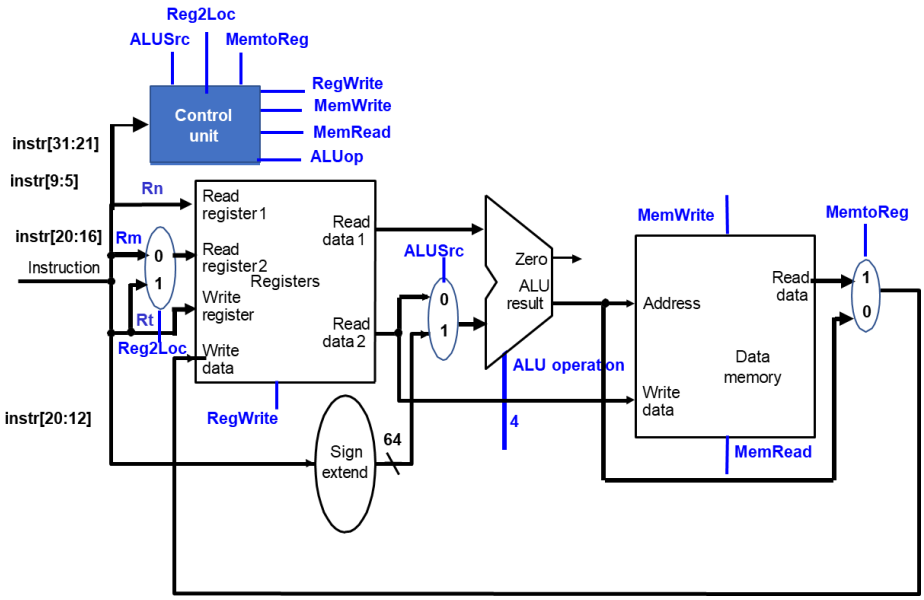
(b)

The addressing mode is PC-relative. It is represented as PC value + offset.
The maximum possible address of the instruction memory to which it could branch = PC value + $2^{\text{Size of offset-1+2}}$ = 0xCC + 2^{20} = 0x1000C8
-1 for sign bit, +2 for Left shift 2

(c)

LDUR X0, [X1, #8]
The value of X1 is retrieved from the register file as register 1. The value of #8 is sign extended and sent to the mux. The control will output a control signal at ALUSrc = 1 to add the two to obtain the data memory address desired. The IM receives a MemRead = 1 and outputs the data to the mux. The mux then receives a signal MemToReg = 1. This data is then sent to the Register File which stores it in the register X0 specified at Rd. The control also outputs a RegWrite signal which allows this writing.

XOR X2, X1, X3
The control sends a signal at Reg2Loc = 0. The output of our register file will then give us the data at X1 and X3. The control also send a control signal ALUSrc = 0 to use the register file output data. The ALU then operates on this two data based on the ALUOp code which would indicate a XOR operation. The data is then sent to the mux which receives a control signal MemToReg = 0. This data is then stored in the register file at the register X0 specified at Rd. The control also outputs a RegWrite signal which allows this writing.



2 (a) (i) Steady State CPI = (8 + 8) / 8 = 2
Total number of loops = 4096 / 8 = 512
Loop unrolling by a factor of 2:
Loop: LDUR X0, [X5, #0]
LDUR X1, [X6, #0]
ADD X2, X1, X0, 2 stalls
SUBI X5, X5, #8, 2 stalls
XORI X2, X2, #15, 2 stalls
STUR X2, [X6, #0]
SUBI X6, X6, #8, 2 stalls
CBNZ X6, loop
finish

(ii) Unrolling XORI twice, returns to original value
LDUR X0, [X5, #0]
LDUR X1, [X6, #0]
LDUR X3, [X5, #-8]
LDUR X4, [X6, #-8]
ADD X2, X1, X0
ADD X7, X4, X3
SUBI X5, X5. #16
XORI X3, X3, #15
XORI X7, X7, #15
SUBI X6, X6, #16
STUR X2, [X6, #0]
STUR X4, [X6, #-8]
CBNZ X6, loop
Steady State CPI = (2 + 13) / 13 = 1.15 (2 stalls)

(iii)

	Way 1	Way 2	Cycle
loop	SUBI X5, X5, #8	LDUR X0, [X5, #0]	1
	SUBI X6, X6, #8	LDUR X1, [X6, #0]	2
	NOP	NOP	3
	NOP	NOP	4
	ADD X2, X0, X1	NOP	5
	NOP	NOP	6
	NOP	NOP	7
	NOP	NOP	8
	XORI X2, X3, #15	NOP	9
	CBNZ X6, loop	NOP	10
	NOP	STUR X2, [X6, #0]	11

CPI: 11/8 = 1.375

(b)

N N N N N N N implies start at predict not taken (00)
T N T N T N
Consider starting at predict not taken:
Prediction: 00, 01, 00, 01, 00, 01 (N, N, N, N, N, N) => 50% accuracy

3 (a)

Cache replacement algorithms:

- Optimal/Belady's Algorithm
- First-in-First-out (FIFO) Algorithm
- Least Recently Used (LRU) Algorithm

Write policies:

- Write through
- Write back

(b)

(i)
L1 Cache:
Direct mapped
Number of blocks = 512
Block size = 128 B = 2^7 B
Number of bits in block offset = $\log_2(\text{Block size}) = 7$
Cache size = (Block size) × (Number of blocks) = 512 × 128 = 64 KB
Number of bits in cache index = $\log_2\left(\frac{\text{Cache size}}{\text{Block size}}\right) = \log_2\left(\frac{2^{16}}{2^7}\right) = 9$ bits

Answer: Y = 9; Z = 7

(ii)

Size of main memory = $2^X = 2^{16+Y+Z} = 2^{16+9+7} = 2^{32}$ = 4 GB

Miss rate = 1 – Hit rate = $1 - \frac{\text{Cache Size}}{\text{Size of main memory}} = 1 - \frac{2^{16}}{2^{32}} = 1 - \frac{1}{2^{16}}$ = 0.999985

(iii)

Cache index field size = 11 bits

Size of block offset = 7 bits

Number of sets = $\frac{\text{Size of cache memory}}{\text{Block Size} \times \text{Associativity}} = \frac{\text{Size of cache memory}}{2^{7} \cdot 4}$

⇒ Size of cache memory = $2^9 \times$ Number of sets

Cache index field size = $\log_2$(Number of sets) = 11

Cache size = $2^9 \cdot 2^{11} = 2^{20}$ bytes = 1 MB

Considering valid, dirty, use and LRU bits, additional 5 bits are added per line of cache:

Additional size = Number of lines × 5 = $\frac{\text{Cache Size}}{\text{Block Size}} \times 5 = \frac{2^{20}}{2^7} \times 5$ = 40 KB

Total cache size = 1.04 MB

(iv)

$AMAT = 4 + 0.05 \times 20 + 0.05 \times 0.01 \times 100 = 5.05\ cycles$

4 (a)

SM employs a unique architecture called SIMT (hardware perspective); Single-Instruction, Multiple-Thread i.e. a warp executes one common instruction for all its threads at a time. Within a single thread, its instructions are pipelined to achieve instruction-level parallelism issued in order, with no branch prediction and speculative execution Individual threads in a warp start together, at the same instruction address; but each has its own instruction address counter and registers; free to branch and execute independently when the thread diverges, such as due to data-dependent conditional execution and branch.

(b)

(i) Threads per block = 512

Total threads = 4096

- ⇨ No of blocks = 4096/512 = 8

Code for Line 18: saxpy<<<8, 512>>>(N, A, d_x, d_y);

(ii) 1 warp = 32 threads

- ⇨ 16 warps per block
- ⇨ 128 warps in total

16 warps are generated for each block

(c)

(i) Temporary variable not global; can be solved by changing Line 4 into _shared_ temp[i] = a[i]*b[i]
Threads may not be ready to share data; need to synchronize or else result will be incorrect; can be solved by Line 5, __syncthreads();
function not declared as __global__

(ii) 1 block with 256 threads to make use of intra-block information sharing, which is needed to compute the dot product. As data sharing is more efficiently implemented among threads rather than among blocks, this is the optimal configuration.

Solver: Kamakshi Asuri Simhakutty (kamakshi001@e.ntu.edu.sg)