SYS3 Exam

SYS3 Exam

Y3891128

Part 1

1.i

False

- Data bus is bidirectional – the processor can read and write data to and from memory.
- Address bus is unidirectional – processor addressing a specific memory location, no device can write to the microprocessor.

1.ii

True: The MAR is responsible for holding the memory address of data being transferred, while the MDR stores the actual data transferred to and from the memory, from the location specified by the MAR, acting as a buffer between memory and CPU.

1.iii

False: In a Load-Store Architecture, ALU operations only occur between registers.

1.iv

False: Register renaming is used to resolve WAR & WAW hazards

Part 2

2.i

$$Speedup = \frac{1}{(1 - \text{Fraction}_A - \text{Fraction}_B - \text{Fraction}_C) + \frac{Fraction_A}{Speedup_A} + + \frac{Fraction_B}{Speedup_B} + + \frac{Fraction_C}{Speedup_C}}$$

$$8 = \frac{1}{(1 - 0.35 - 0.35 - x) + \frac{0.35}{20} + + \frac{0.35}{30} + + \frac{x}{25}}$$

$\implies x = 0.2127\ (4.dp),\ (21\%)$

2.ii

$Speedup_A = \frac{1}{(1-0.2)+\frac{0.2}{20}} = 1.23\ (2.dp)$
$Speedup_B = \frac{1}{(1-0.2)+\frac{0.2}{30}} = 1.24\ (2.dp)$
$Speedup_C = \frac{1}{(1-0.6)+\frac{0.6}{25}} = 2.36\ (2.dp)$

2.ii.a To achieve the highest performance, we must choose the enhancement that achieves the highest speedup.

Therefore, we implement enhancement C, with a speedup of 2.36.

2.ii.b We implement the two enhancements with the highest speedups, B (1.24) and C (2.36)

2.iii

A processor could use pipeline scheduling by simultaneously execute stages from multiple instructions in one cycle. This can dramatically increase performance compared to serial execution by reducing the number of cycles required to execute a set of instructions.

Implementing a cache can lead to a large speedup in memory operations, such as load and store, by having a small memory component close to the processor that is very fast and can be queried by the CPU with less overhead than querying main memory.

Part 3

3.i

	Instruction sequences	Dependencies
A	1. LW R1, 40(R6) 2. Add R6, R2, R2 3. SW R6,50(R1)	RAW on R1, instruction 1 to 3 RAW on R6, instruction 2 to 3

	Instruction sequences	Dependencies
B	1. LW R5, -16(R5) 2. SW R5, -16(R5) 3. ADD R5, R5, R5	2 RAWs on R5, instructions 1 (cycle 4) to instruction 2 (cycle 3 & 4) RAW on R5, instruction 1 to 3 Potiential WAW on R5, instruction 1 to 3 Potiential WAR on R5, Instruction 2 to 3

3.ii

	Instruction sequence	Dependencies
A	1. LW R1, 40(R6) 2. Add R6, R2, R2 3. NOP 4. NOP 5. SW R6,50(R1)	Eliminated RAW on instruction 1 to 5 using NOP stalls Eliminated RAW on instruction 2 to 5 using NOP stalls
B	1. LW R5, -16(R5) 2. NOP 3. NOP 4. SW R5, -16(R5) 5. ADD R5, R5, R5	Eliminated 2 RAW on instructions on 1 (cycle 4) to 4 (cycle 3 & 4) using NOP stalls Eliminated RAW on instruction 1 to 5 using NOP stalls Potiential WAW on R5 Instruction 1 to Instruction 5 Potiential WAR on R5 on Instruction 4 to 5

3.iii

	Instruction sequence	Dependencies
A	1. LW R1, 40(R6) 2. Add R6, R2, R2 3. NOP 4. SW R6,50(R1)	Eliminated RAW hazard R6 using operand forwards, on instruction 2 to 4 Eliminated RAW on instruction 1 to 4 using NOP stalls
B	1. LW R5, -16(R5) 2. NOP 3. SW R5, -16(R5) 4. ADD R5, R5, R5	Eliminated RAW hazard on R5 using operand forwards, instruction 1 to 3 Eliminated another RAW hazard on R5 using NOP stalls, instruction 1 to 3 Eliminates RAW on R5 using NOP stalls, instruction 1 to 4 Potiential WAW on R5 Instruction 1 to Instruction 4 Potiential WAR on R5 on Instruction 3 to 4

Part 4

4.i

No.	Last Outcome	BHT N/T	Prediction	Outcome
1	N (initial)	00 / 11	N	T
2	T	01 / 11	T	T
3	T	01 / 11	T	N
4	N	01 / 10	N	N
5	N	00 / 10	N	T
6	T	01 / 10	T	N
7	N	01 / 00	N	T
8	T	11 / 00	N	T
9	T	11 / 01	N	T
10	T	11 / 11	T	N
11	N	11 / 10	T	T
12	T	11 / 10	T	N
13	N	11 / 00	T	T
14	T	11 / 00	N	T

No.	Last Outcome	BHT N/T	Prediction	Outcome
15	T	11 / 01	N	N
16	N	11 / 00	T	T

4.ii

Branches mispredicted: 1, 3, 5, 6, 7, 8, 9, 10, 12, 14

- 10/16 incorrect, misprediction rate of 62.5%

Part 5

5.i

The number of bits in each entry increases by 1 bit – Doubling physical memory means there are now twice as many physical pages, so the physical page number needs to expand by 1 bit.

5.ii

The number of entries will be doubled – Doubling virtual memory size means there are twice as many virtual pages, so twice as many entries.

5.iii

The number of entries will be halved – Doubling page size means there will be half as many virtual pages, which means there will be half as many entries.

Part 6

6.i

Cache size (C) = Block size × Lines per set × $\#sets$

- $\#sets = 2^{index\ bits}$
- Lines per set: the set-associativity

A: $C = 1 \times 1 \times 2^{10} = 1024$ bytes
B: $C = 4 \times 2 \times 2^{7} = 1024$ bytes

- Fully associative, so only has a single set, with 256 blocks

C: $C = 4 \times 256 \times 1 = 1024$ bytes

6.ii

Size of tags = $\frac{\text{Num of tag bits} \times \#sets}{8}$

A: $\frac{6 \times 2^{10}}{8} = 768$ bytes
B: $\frac{7 \times 2^{7}}{8} = 112$ bytes

- Number of sets = 1, however any block can be put in this set, so we need to store the tag from each block
- Size of tags = $\frac{\text{Num of tag bits} \times \text{Num of blocks}}{8}$

C: $\frac{256 \times 14}{8} = 448$ bytes

6.iii

Fully associative cache will have zero conflict misses by definition: mapping of a main memory can be done with any cache block, so there will never be a conflict miss, only capacity and compulsory misses.

A directly mapped cache will have the highest number of conflict misses. It can only store one line in a set, so therefore will have more conflict misses when multiple blocks are mapped to the same set than a set associative cache, which can store multiple lines in a set.

6.iv

A: Service time $= 0.4 \times 1 + (1 - 0.4) \times 20 = 12.4$
B: Service time $= 0.6 \times 2 + (1 - 0.6) \times 20 = 9.2$
C: Service time $= 0.8 \times 5 + (1 - 0.8) \times 20 = 8$

Part 7

7.i

Tomasulo's algorithm uses register renaming, using reservation stations, to prevent an earlier register from overwriting the value of a later register. This removes the potential of name dependence on a register and therefore eliminates the possibility of WAR and WAW hazards.

Scoreboarding is where there is a central location about all active instructions. We can use this table to avoid WAR and WAW hazards that may occur due to name dependence. When we encounter a name dependence, we must stall and wait for the hazard to clear.

7.ii

#	Instruction	Issue	Src_Op1	Src_Op2	EX	WB	C
1	L.D F0, 24(R1)	1	IMM	RF	2	4	5
2	L.D F4, 24(R2)	2	IMM	RF	3	5	6
3	MULT.D F2, F4, F10	3	CDB	RF	6	16	1
4	SUB.D F6, F0, F4	4	ROB/CDB (same cycle)	CDB	5	7	1
5	DIV.D F8, F2, F0	5	CDB	ROB	17	57	5
6	ADD.D F0, F6, F4	6	CDB	ROB/RF (commited cycle 6)	8	10	5

(a) Writes back at cycle 5, commits at 6
(b) Writes back at cycle 16, commits at 17
(c) Writes back at cycle 10, commits at 59

7.iii

The reorder buffer is used in the Tomasulo algorithm and allows for out-of-order execution, while being committed in the in-order, ensuring correctness. It does this by holding instructions between their completion and commit in the re-order buffer, only committing the instruction after the previous instruction has been committed. Subsequent instructions can query the re-order buffer for uncommitted data to continue the speculative execution.

The re-order buffer also enables branch predictions. Until the branch direction is clear, subsequent instructions are added to the re-order buffer but not committed. If the branch is correct, we commit the changes, else flush the re-order buffer back to the branch instruction.