1. You purchased an Acme computer with the following features:

• 95% of all memory accesses are found in the cache.
• Each cache block is two words, and the whole block is read on any miss.
• The processor sends references to its cache at the rate of 109 words per second.
• 25% of those references are writes.
• Assume that the memory system can support 109 words per second, reads or writes.
• The bus reads or writes a single word at a time (the memory system cannot read or write two words at once).
• Assume at any one time, 30% of the blocks in the cache have been modified.
• The cache uses write allocate on a write miss.

You are considering adding a peripheral to the system, and you want to know how much of the memory system bandwidth is already used. **Calculate the percentage of memory system bandwidth used on the average in the two cases below. Be sure to state your assumptions.**

We know:

* Miss rate = 0.05

* Block size = 2 words (8 bytes)

* Frequency of memory operations from processor = 109

* Frequency of writes from processor = 0.25 ∗ 109

* Bus can only transfer one word at a time to/from processor/memory

* On average 30% of blocks in the cache have been modified (must be written back in the case of the write back cache)

* Cache is write allocate

So:

Fraction of read hits = 0.75 ∗ 0.95 = 0.7125

Fraction of read misses = 0.75 ∗ 0.05 = 0.0375

Fraction of write hits = 0.25 ∗ 0.95 = 0.2375

Fraction of write misses = 0.25 ∗ 0.05 =0.0125

1.1 Write through cache

- On a read hit there is no memory access
- On a read miss memory must send two words to the cache
- On a write hit the cache must send a word to memory

- On a write miss memory must send two words to the cache, and then the cache must send a word to memory

Thus:

Average words transferred = $0.7125 * 0 + 0.0375 * 2 + 0.2375 * 1 + 0.0125 * 3 = 0.35$

Average bandwidth used = $0.35 * 10^9$

Fraction of bandwidth used =

$[0.35 \times 10^9] / 10^9$

$= 0.35$ (1)

1.2 Write back cache

On a read hit there is no memory access

On a read miss:

1. If replaced line is modified then cache must send two words to memory, and then

memory must send two words to the cache

2. If replaced line is clean then memory must send two words to the cache

On a write hit there is no memory access

On a write miss:

1. If replaced line is modified then cache must send two words to memory, and then memory must send two words to the cache

2. If replaced line is clean then memory must send two words to the cache

Thus:

Average words transferred = $0.7125 * 0 + 0.0375 * (0.7 * 2 + 0.3 * 4) + 0.2375 * 0 + 0.0125 *$

$(0.7 * 2 + 0.3 * 4) = 0.13$

Average bandwidth used = $0.13 * 10^9$

Fraction of bandwidth used =

$0.13 \times 10^9/10^9$

$= 0.13$ (2)

Comparing (1) and (2) we notice that the write through cache uses more than twice the cache-memory bandwidth of the write back cache.

2. One difference between a write-through cache and a write-back cache can be in the time it takes to write. During the first cycle, we detect whether a hit will occur, and during the second (assuming a hit) we actually write the data. Let's assume that 50% of the blocks are dirty for a write-back cache. For this question, assume that the write buffer for the write through will never stall the CPU (no penalty). Assume a cache read hit takes 1 clock cycle, the cache miss penalty is 50 clock cycles, and a block write from the cache to main memory takes 50 clock cycles. Finally, assume the instruction cache miss rate is 0.5% and the data cache miss rate is 1%. Assuming that on average 26% and 9% of instructions in the workload are loads and stores, respectively, estimate the performance of a write-through cache with a two-cycle write versus a write-back cache with a two-cycle write

CPU performance equation: CPU_Time = IC $*$ CPI $*$ ClockTime

CPI = CPI_execution + Stall_Cycles_Per_Instruction

We know:

* Instruction miss penalty is 50 cycles

* Data read hit takes 1 cycle

* Data write hit takes 2 cycles

* Data miss penalty is 50 cycles for write through cache

* Data miss penalty is 50 cycles or 100 cycles for write back cache

* Miss rate is 1% for data cache (MRD) and 0.5% for instruction cache (MRI)

* 50% of cache blocks are dirty in the write back cache

* 26% of all instructions are loads

* 9% of all instructions are stores

Then:

CPI_execution = $0.26 * 1 + 0.09 * 2 + 0.65 * 1 = 1.09$

Write through

Stall_Cycles_Per_Instruction = MRI $* 50 +$ MRD $* (0.26 * 50 + 0.09 * 50) = 0.425$

so:

CPI = $1.09 + 0.425 = 1.515$ (1)

Write back

Stall_Cycles_Per_Instruction = MRI $* 50 +$ MRD $* (0.26 * (0.5 * 50 + 0.5 * 100) + 0.09 * (0.5 * 50 + 0.5 * 100)) =$ **0.5125**

so:

CPI = $1.09 + 0.5125 = 1.6025$ (2)

3.Consider the following program and cache behaviors.

Data Reads per 1K instructions	Data Writes per 1K instructions	Instruction Cache Miss Rate	Data Cache Miss Rate	Block Size (Bytes)
300	150	0.5%	5%	128

3.1 Suppose a CPU with a write-through, write allocate cache achieves a CPI of 2. What are the read and write bandwidths (measured by bytes per cycle) between RAM and the cache? (Assume each miss generates a request for one block.). *For a write-allocate policy, a write miss also makes a read request to RAM – please be sure to consider its impact on Read Bandwidth*

Instruction Bandwidth:
When the CPI is 2, there are, on average, 0.5 instruction accesses per cycle.
0.5 instructions read from Instruction memory per cycle
0.5% of these *instruction* accesses cause a cache ***Read*** miss (and subsequent memory request).
[0.5 instr/cycle] x [0.005 misses/instruction] = missed instructions/cycle
Assuming each miss requests one block and each block is 128 bytes [16 words with 8 bytes (64 bits) per word] , instruction accesses generate an average of
[0.5 instr/cycle] x [0.005 misses/instruction] x[128 bytes/miss] =
= 0.32 bytes/cycle of *read traffic*

Read Data bandwidth:
30% of instructions generate a ***read*** request from data memory.
[0.5 instr/cycle] x [0.3 Read Data Accesses/instruction] = [0.15 Read Data Accesses / cycle]
5% of these generate a cache miss;
[0.15 Read Data Accesses / cycle] x [0.05 misses / Read Data Access] = 0.0075 Read Misses/cycle
Assuming each miss requests one block and each block is 128 bytes [16 words with 8 bytes (64 bits) per word] ,
[0.0075 Read Misses/cycle] x [128 Bytes/block] x [1 block/miss] = 0.0075 x 128 Bytes/cycle
= 0.96 Bytes/cycle
Write Data bandwidth:
15% of instructions generate a ***write*** request into data memory.
[0.5 instr/cycle] x [0.15 Write Data Accesses/instruction] = [0.075 Write Data Accesses / cycle]
All of the words written to the cache must be written into Memory:
[0.075 Write Data Accesses / cycle] x [8 bytes/word] x [1 word/write-through] = 0.6 Bytes/cycle

For a *Write-allocate policy*, a Write miss also makes *a* ***read*** *request* to RAM
[0.5 inst/cycle] x [0.15 Write Data Accesses/instruction] x [0.05 misses/Write Data Access] x [128 Bytes/miss]
= 0.48 Bytes/cycle
Assuming each miss requests one Word (8 bytes) since this is a write-through cache *with only 1 word written per miss into memory*,
[0.00375 Write Misses/cycle] x [8 Bytes/word] x [1 word/miss] = 0.03 Bytes/cycle

Total Read Bandwidth
0.32 (Instruction memory) + **0.96** (data memory) + **0.48** (Write-miss in Write-through cache with Write Allocate) **Bytes/cyle = 1.76 Bytes/cycle**
Total Write Bandwidth:
0.6 Bytes/cycle + 0.03 Bytes/cycle = 0.63 Bytes/cycle

4. Consider the following RISC V Instruction sequence executing in a 5-stage pipeline:

```
or   x13, x12, x11
ld   x10, 0(x13)
ld   x11, 8(x13)
add  x12, x10, x11
subi x13, x12, 16
```

4.1 Identify all of the data hazards and their resolution with NOPs assuming no forwarding or hazard detection hardware is being used

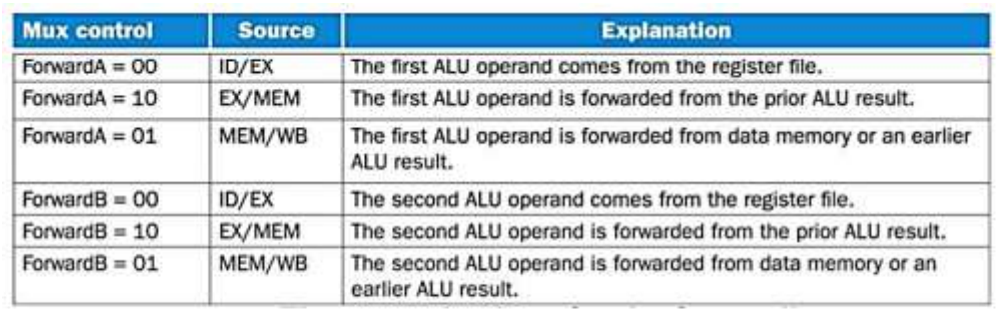
Hazards identified:

`or   x13, x12, x11`	
`ld   x10, 0(x13)`	*EX to 1st RAW Hazard*
`ld   x11, 8(x13)`	*EX to 2nd RAW Hazard*
`add  x12, x10, x11`	*MEM to 1st RAW [load-use-data] & MEM to 2nd Hazards*
`subi x13, x12, 16`	*Ex to 1st RAW Hazard*

NOPS introduced to resolve Hazards:

`or   x13, x12, x11`	
`NOPS`	
`NOPS`	
`ld   x10, 0(x13)`	*EX to 1st RAW Hazard resolution with 2 NOPs*
`ld   x11, 8(x13)`	*EX to 2nd RAW Hazard resolved as well from above 2 NOPs*
`NOPS`	
`NOPS`	
`add  x12, x10, x11`	*MEM to 1st RAW [load-use-data] & MEM to 2nd Hazards resolved with 2 NOPs*
`NOPS`	
`NOPS`	
`subi x13, x12, 16`	*Ex to 1st only RAW Hazard resolved with 2 NOPs*

4.2 If there is forwarding, for the first seven cycles during the execution of this code, *specify which signals are asserted in each cycle by hazard detection and forwarding units* in Figure below.



Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.

	Clock Cycle	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	**or**	IF	ID	**EX**	MEM	WB					
2	**ld**		IF	ID	**EX**	MEM	WB				
3	**ld**			IF	ID	EX	MEM	WB			
4	**NOP**	*mandatory NOP for which no forwarding solution possible: load-data-use*									
5	**add**					IF	ID	EX	MEM	WB	
6	**subi**						IF	ID	EX	MEM	WB

(1) A=x B=x *(no instruction in* `EX` *stage yet)*

(2) A=x B=x *(no instruction in* `EX` *stage yet)*

(3) A=0 B=0 *(both operands of the* `or` *instruction:* `x11,x12` *come from Reg File)*

(4) A=2 B=0 *(base (*`RS1`*) in first* `ld` *(*`x13`*) taken from EX/MEM of previous instruction)*

(5) A=1 B=0 *(base (*`RS1`*) in 2nd* `ld` *(*`x13`*) taken from MEM/WB of a previous instruction)*

(6) A=x B=x *(no instruction in* `EX` *stage yet because NOP introduced to resolve MEM to 1st*

(7) A=0 B=1 *(*`RS2` *in the* `add` *instruction is* `x11` *which is forwarded from MEM/WB of 2nd* `ld`*, the result of the 1st* `ld` *(*`x10`*) has already been written into Reg File in CC 6 - so, no forwarding necessary for first operand)*

(8) A=1 B=0 *(*`RS1` *of* `subi` *instruction forwarded from EX/MEM of* `add` *instruction)*