CS305 Assignment-8

Team:
Guttu Sai Abhishek 180050036
Hrithik Vikas Samala 180050038
Mulinti Shaik Wajid 180050063
Sai Phanindra Ramasahayam 180050084
Sanapathi Sumanth Balaji 180050091

Question 1

(a) and (b) parts

$AMAT_{L1} = HT_{L1} + (1-h_{L1})*AMAT_{L2}$ ------(1)
$AMAT_{L2} = HT_{L2} + (1-h_{L2})*MT_{L2}$ ------(2)

From equation 2, $AMAT_{L2} = 10 + (1-0.5)*200$. So $AMAT_{L2} = 110$ cycles
Substituting value of $AMAT_{L2}$ in equation 1 gives $AMAT_{L1} = 1 + (1-0.96)*110$.
So $AMAT_{L1} = 5.4$ cycles

(c)

In non load-store instructions there will be one access to Instruction memory and it takes 5.4 cycles on average, without stalls it takes only 1 cycle, so there is a stall of 5.4-1=4.4 cycles. In load store instructions there will be two accesses to memory, one access to data memory and one access to instruction memory, so it takes 5.4*2=10.8 cycles, without a stall it takes 2 cycles, so there is a stall of 10.8-2=8.8 cycles. Assuming that there are no other types of stalls, the average number of stalls per instruction = 0.5*4.4 + 0.5*8.8 = 6.6 cycles

(d)

$AMAT_{L1} = HT_{L1} + (1-h_{L1})*MT_{L1}$
Assuming the miss time of L1 is equal to miss time of L2(because both involve time to access main memory)
$AMAT_{L1} = 1 + (1-0.96)*200 = 9$ cycles
In non load-store instructions there will be one access to Instruction memory and it takes 9 cycles on average, without stalls it takes only 1 cycle, so there is a stall of 9-1=8 cycles. In load store instructions there will be two accesses to memory, one access to data memory and one access to instruction memory, so it takes 9*2=18 cycles, without a stall it takes 2 cycles, so there is a stall of 18-2=16 cycles. Assuming that there are no other types of stalls, the average number of stalls per instruction = 0.5*8 + 0.5*16 = 12 cycles

Question 2

a) 10
b) 7
c) 5
d) 10
e) 28, 23, 26

Explanation:

Note:

1. There a total of 5 iterations(since $3 is set to 5 initially and decremented by 1 each time till 0) for every part i.e, 5 times evaluating the ‘here’ label i.e, 10 memory access are made to data memory since 2 accesses are made during each iteration
2. The 5 line pattern for below parts represent the 5 iterations. 2 elements in each line of that 5 line pattern represent the named(named wherever required) quantity value during each of the two data memory accesses in the ‘here’ label
3. Start physical address of array arr is 0x100
4. M mean Miss, H mean Hit during memory access
5. Number of hits, misses does not depend on whether we use write through or write back. But it depends on what strategy is used while write miss is encountered. Here write allocate is used which implies that a write miss lead to its presence in D cache just after writing in MM and before executing next instruction
6. Initially D cache is assumed to be empty for all parts
7. Almost all numbers presented in a,b,c parts are in binary except the index of arr
8. If the most significant bits are 0 in a binary representation, they are ignored in below calculation, but it is to be noted that physical address is 16 bit

a)

Index of arr accessed:
0, 4
20, 24
40, 44
60, 64
80, 84

Physical address of data accessed from Data memory:
1 0000 0000, 1 0000 0100
1 0001 0100, 1 0001 1000
1 0010 1000, 1 0010 1100
1 0011 1100, 1 0100 0000
1 0101 0000, 1 0101 0100

4 byte line in 32 byte D cache implies 8 lines in D cache numbered 0 to 7

So physical address should be viewed as [tag(remaining most significant bits) | line number in D cache(3 bits) | offset inside the line(least significant 2 bits)]

Line number(in binary) the physical address maps to, in D cache:
000, 001
101, 110
010, 011
111, 000
100, 101

Hit or Miss along with tag in that corresponding line(line number mentioned above) of D cache just before executing next instruction(there are no sets, hence each element below means the hit or miss to respective line number mentioned above AND each miss either evicts nothing if there is an invalid line in D cache(at starting) and evicts the previous allocated line of the same index if there is a valid line))
M^ 1000, M 1000
M* 1000, M 1000
M 1001, M 1001
M 1001, M 1010(evicts the line entered during the miss marked ^)
M 1010, M 1010(evicts the line entered during the miss marked *)

b)

Index of arr accessed:
0, 4
20, 24
40, 44
60, 64
80, 84

Physical address of data accessed from Data memory:
1 0000 0000, 1 0000 0100
1 0001 0100, 1 0001 1000
1 0010 1000, 1 0010 1100
1 0011 1100, 1 0100 0000
1 0101 0000, 1 0101 0100

8 byte line in 32 byte D cache implies 4 lines in D cache numbered 0 to 3

So physical address should be viewed as [tag(remaining most significant bits) | line number in D cache(2 bits) | offset inside the line(least significant 3 bits)]

Line number(in binary) the physical address maps to, in D cache:
00, 00
10, 11
01, 01
11, 00
10, 10

Hit or Miss along with tag in the corresponding line(line number mentioned above) of D cache just before executing next instruction(there are no sets, hence each element below means the hit or miss to respective line number mentioned above AND each miss either evicts nothing if there is an invalid line in D cache(at starting) and evicts the previous allocated line of the same index if there is a valid line))
M* 1000, H 1000
M` 1000, M^ 1000
M 1001, H 1001

M 1001(evicts the line entered during the miss marked ^), M 1010(evicts the line entered during the miss marked *)
M 1010(evicts the line entered during the miss marked `), H 1010

c)

Index of arr accessed:
0, 4
16, 20
32, 36
48, 52
64, 68

Physical address of data accessed from Data memory:
1 0000 0000, 1 0000 0100
1 0001 0000, 1 0001 0100
1 0010 0000, 1 0010 0100
1 0011 0000, 1 0011 0100
1 0100 0000, 1 0100 0100

8 byte line in 32 byte D cache implies 4 lines in D cache numbered 0 to 3

So physical address should be viewed as [tag(remaining most significant bits) | line number in D cache(2 bits) | offset inside the line(least significant 3 bits)]

Line number(in binary) the physical address maps to, in D cache:
00, 00
10, 10
00, 00
10, 10
00, 00

Hit or Miss along with tag in the corresponding line(line number mentioned above) of D cache just before executing next instruction(there are no sets, hence each element below means the hit or miss to respective line number mentioned above AND each miss either evicts nothing if there is an invalid line in D cache(at starting) and evicts the previous allocated line of the same index if there is a valid line)
M* 1000, H 1000
M^ 1000, H 1000
M` 1001(evicts the line entered during the miss marked *), H 1001
M 1001(evicts the line entered during the miss marked ^), H 1001
M 1010(evicts the line entered during the miss marked `), H 1010

d)

Index of arr accessed:
0, 4

20, 24
40, 44
60, 64
80, 84

Physical address(in binary) of data accessed from Data memory:
1 0000 0000, 1 0000 0100
1 0001 0100, 1 0001 1000
1 0010 1000, 1 0010 1100
1 0011 1100, 1 0100 0000
1 0101 0000, 1 0101 0100

4 byte line in 32 byte D cache implies 8 lines in D cache indexed 0 to 2 where first 3 lines have index 0 and next 3 lines have index 1 and next 2 lines have index 2

So physical address should be viewed as [tag(quotient when (physical address/4) is divided by 3) | line number in D cache(remainder when (physical address/4) is divided by 3) | offset inside the line(least significant 2 bits)]

Physical address(in decimal) divided by 4:
64, 65
69, 70
74, 75
79, 80
84, 85

Remainder(in decimal) when (physical address/4) is divided by 3 OR the index of the D cache, the physical memory maps to
1, 2
0, 1
2, 0
1, 2
0, 1

quotient(in decimal) when (physical address/4) is divided by 3 OR the tag in the D cache
21, 21
23, 23
24, 25
26, 26
28, 28

Hit or Miss along with index(in decimal) of D cache the physical address maps to, then tag(in decimal) of that physical address just before executing next instruction
M^ 1 21, M* 2 21
M 0 23, M 1 23
M 2 24, M 0 25
M 1 26, M 2 26(the line that entered into index 2 during the miss marked *, is evicted during this new allocation as per LRU)

M 0 28, M 1 28(the line that entered into index 1 during the miss marked ^, is evicted during this new allocation as per LRU)

Final tags(in decimal) in each index in D cache
0 -> 23, 25, 28
1 -> 28, 23, 26
2 -> 26, 24

e) Answer is directly borrowed from the calculation done above