Assignment 6

1. LSQ (30 pts)

Assuming that the processor does ***no*** memory dependence prediction to speculatively issue loads.

LD/ST	The register for the address calculation is made available	The register that must be stored into memory is made available	The calculated effective address	Effective address Calculated on cycle	Data memory access time
LD	4	-	abce	5	6
ST	9	3	abdd	10	On Commit
LD	2	-	abcd	3	11
LD	5	-	abdd	6	11
ST	2	3	abdd	3	On Commit
LD	6	-	abdd	7	8
LD	1	-	abce	2	11

1. The effective address will be calculated one cycle after the address operands are made available.
2. The stores will access the data memory on commit
3. The loads will access the memory based on the dependent stores, if same address then it would also depend on store data availability.

2. Memory access times (30 points)

Consider a processor and a program that would have an IPC of 1 with a perfect 1-cycle L1 cache Assume that each additional cycle for cache/memory access causes program execution time to increase by one cycle.

Program Execution times:
Assume 1000 instructions (1Kilo)
1000 instructions -> 1000 cycles
a) L1-L2-L3-L4-memory
L1 Miss (L2) - 40*8 = 320
L2 Miss (L3) - 25*30 = 750
L3 Miss (L4) - 10*60 = 600
L4 Miss (Memory) - 5*250 = 1250
Total exec time- 3920 cycles

b) L1-L2-L3-memory
L1 Miss (L2) - 40*8 = 320
L2 Miss (L3) - 25*30 = 750
L3 Miss (Memory) - 10*250 = 2500
Total exec time- 4570 cycles

c) L1-L2-L4-memory
L1 Miss (L2) - 40*8 = 320
L2 Miss (L4) - 25*60 = 1500
L4 Miss (Memory) - 5*250 = 1250
Total exec time- 4070 cycles

L1-L2-L3-L4-memory cache hierarchy is the best with lowest execution time. It is the best design as before accessing the memory(high access time) we are first checking all the caches (having less access time) and there is a high chance we would find the data in the caches. Number of misses reaching memory and bigger cache is less.

3. Cache Organization (20 points)
Cache size = 48 Mb
Block size = 128 bytes
#ways = 12-way
Total address - 40 bit

#Sets = Cache size / (#ways * Block size)
= 48 / (12*128)
= **2^15 sets**

Offset bits = log(Block size)
= log(128) —(base 2)
= **7 bits**

Index bits = log(#sets)
= log(2^15) —(base2)
= **15 bits**

Tag bits = Total - 7 - 15
= **18 bits**

Tag array size = #Sets * #ways * Tag bits
= 2^15 * 12 * 18bits
= 6912 Kbits/ 8
= **864 Kbytes**

4. Cache Miss Rates (20 points)
Assume an LRU replacement policy.

Access pattern: A B C D E A C E A C E

I)

~~A E C A~~ E	~~C A E~~ C
B	D

M M M M M M M M M M M

II) Hit rate = **0/11**

III) Making 1 set ways

~~A~~ E	~~B~~ A	C	D

M M M M M M H H H H H

Yes the hit rate improves to **5/11.**