Section A.

Question 1

1. Cache holds a copy of section of memory close to the CPU, for the CPU to access it fast.  
   With a cache “Write back” policy, DMA may read a from main memory an old data.  
   DMA may write to main memory a new value, but cache still holds the old value. Causing out of sync problem.  
   Two proposed solutions are:  
   - Cache may be flushed before every DMA operation.  
   - Add a snooping device to the memory bus to check DMA operations and update memory and cache as needed.
2. Compulsory cache miss are the misses that have to happen, usually when we first load the program into memory, or after a cache flush.
3. Flushing the cache before every DMA operation will cause a compulsory cache misses until all values are back into the cache.
4. By adding a snooping device to memory bus, this way we only update values we need to update rather than completely flushing the cache.
5. 95% \* 100 = 95 of the times we will have a hit.  
   Cost of per hit in cache is 1 cycle, so that 95 cycle   
   5 of the time we will have to access main memory, which cost 25 cycle per access.  
   So that is 5 \* (25 + 1) = 130   
   130 + 95 = 225/ 100 = 2.25 access on average.
6. AccessRateL1 = H(L1) \* Accesses   
   AccessRateL2 = Accesses - AccessRateL1  
   Rest = Accesses - (AccessRateL2 + AccessRateL1)  
     
   Total cycle = AccessRateL1\* T(L1) + AccessRateL2 \* T(L1 + L2) + Rest \* T(L1 + L2 + main memory)   
     
   Where H(X) is the hit rate of X cache  
   T(X) is the number of cycle it cost for X chace.

Question 2.

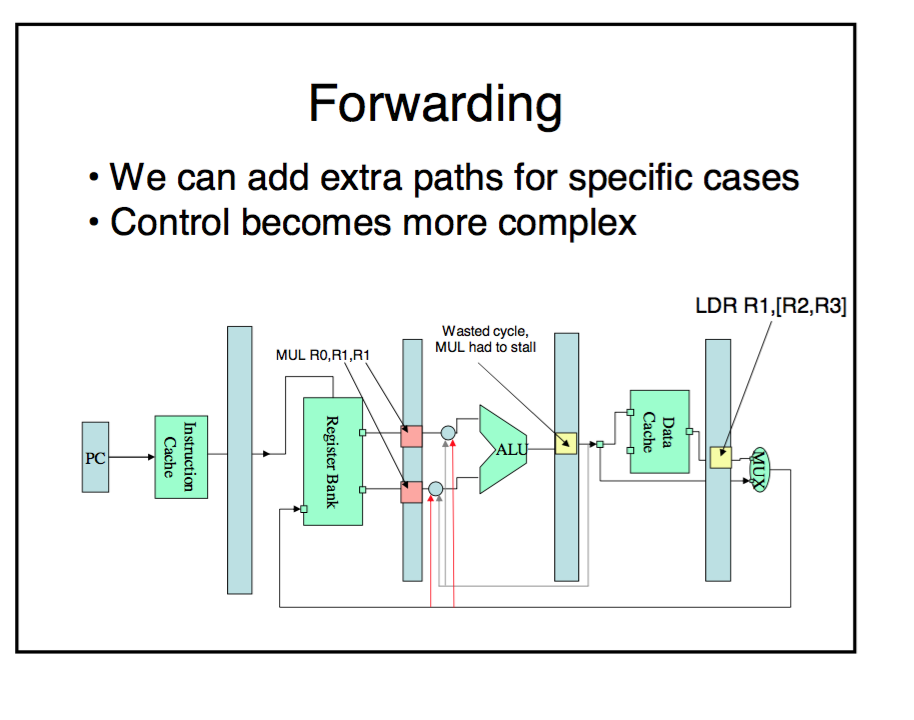
1. Seek time: The time it takes to move the read/write head over the correct partition.   
     
   Search time: The time required between ending the seek and the time required data become available for read/write. On average, half the time of one rotation.  
     
   Transfer time: is the time it takes to transfer required data between computer and disk.
2. Time to transfer file = seek time + search time + transfer time.  
     
   Seek time = 10ms  
   Search time = 1/2 \* 60/7200 = 4.1ms  
   Transfer time = 4/(90 \* 2^10) = 0.000043second = 0.043ms  
     
   10 + 4.1 + 0.043 = 14.143ms
3. Entire disk: seek time and search time can be omitted since they are too small to be of any significant in relevance to entire disk transfer time.  
     
   Transfer time = (3 \* 2^40) / (90 \* 2^20) = 34952.53 seconds   
     
   About 9 hours and 42 minutes
4. 14.143 \* (3/4\*2^30) ms = 11389447962.6ms = 11389447.9626 seconds   
   About 3163.73 hours (WAT)  
   About 131.82 days
5. Disks are slow and unreliable, can fail anytime during those hours of transfer.  
   We can use RAID0 (striping files to multiple disks) to increase the transfer rate.   
   RAID1 to mirror the data in case of failure we will have a backup.  
   And RAID2-6 to store parity data to reconstruct the data after a fail.

Section B.

Question 3:

1. By splitting instruction into different stages in a way that multiple instruction can overlap their execution. This provides a more efficient utilisation of the processor resources, while at same time increasing the clock frequency. Allowing more instructions to be executed at same time.
2. Instruction fetch: get instruction from memory  
   Instruction decode: decode the instruction and select registers.   
   Execute instruction: Perform operation, or calculate address.   
   Memory access: Access an operand in data memory.  
   Write back: Write results to register.  
     
   Can’t find a diagram for this, I will take -2 marks tyvm
3. Control hazards: when there is a branch, the following instruction after the branch won’t be decided until stage 2 (ID) if unconditional branch, or stage 3 (EXE) if conditional branch, so the instructions fetched/decoded might be wasted and need to remove them from the pipeline. Sadpepe.jpeg   
   Either stall the pipeline until branch instruction is decoded, or add NOP instructions after each branch instruction.  
   Data hazards: Instruction need to read from register that is written by a previous instruction before it is stored in register bank. Example loading a value into register, that is used by another instruction to do some calculation.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | t1 | t2 | t3 | t4 | t5 | t6 | t7 | t8 | t9 | T10 | T11 | T12 | T13 |
| MUL R2 | IF | ID | EXE | MEM | WB |  |  |  |  |  |  |  |  |
| LDR R1 |  | IF | ID | EXE | MEM | WB |  |  |  |  |  |  |  |
| ADD R1 |  |  | IF | ID | Stall | Stall | EXE | MEM | WB |  |  |  |  |
| MUL R4 |  |  |  | IF | Stall | Stall | ID | EXE | MEM | WB |  |  |  |
| LDR R3 |  |  |  |  |  |  | IF | ID | EXE | MEM | WB |  |  |
| ADD |  |  |  |  |  |  |  | IF | ID | Stall | EXE | MEM | WB |

1. Forwarding can add extra path so the values of register forwarded to next instruction after the MEM stage rather than waiting to update the memory before reading it.
2. LDR R1 X  
   LDR R3 X  
   MUL R2 R2 R2  
   MUL R4 R4 R4  
   Add R1 R1 R2  
   Add R3 R3 R4  
     
   This will omit the need for any stall operation and hence keeping the cycles busy and avoid wasting resources.

Question 4:

1. Multithreading: control of logic is replicated so instruction from several instruction flows (threads) can be issued but the execution logic is shared among the different threads. AKA replicate instruction, do calculation on different threads on same shared memory space.   
     
   Multicore: all processing logic (control + execution) is replicated so different instruction flows (threads) are executed independently.  
   AKA give each core the same isolated copy of the program and ask it to execute instruction there.  
     
   Superscala: execution logic is replicated to allow issuing several instruction per cycle from a single instruction flow.
2. Course-graind multithreading: instruction are issued from a single thread until there is an expensive operation (cache miss, ICM) in which case the instruction flow is switched to a different thread.
3. Fine-grained multithreading: instruction from several threads are interleaved executing an instruction from each thread unless it is stalled.
4. Simultaneous multithreading: issue several instruction from any ready-marked thread.
5. Cache coherency: in multi core/multiprocessor environment the main memory is shared by different cores, however if a cache hierarchy is implemented, each core will have copies of main memory’s data in their local caches. Keeping all these copies-up to date is cache coherency.  
     
   It is important because if no precaution is taken, two cores could see different values when accessing the same variable which is against semantic principle of shared memory systems which may lead to unstable execution.   
     
   E.g. core1 cache1 have value of X = 4, core2 cache2 reads X and has value = 5. Now sometime depending on availability of the cores we will get X = 4 or X =5
6. CPU rate = 10^9 instruction/second   
   A clock cycle = 1 /10^9 = 1ns per instruction  
   Main memory access time = 75ns = 75 cycle  
   0.1% data cache misses   
     
   For 10000 instruction program, that will have 2000 read.   
   2000 read 0.1% of those will have data cache miss and we will need to access memory  
   That means, Cache hit cycles = 99.99% \* 2000 \* 1 = 1999.8 cycles.   
   And 0.2 will access memory with, 0.2 \* (1+15) = 3.2   
   Total = 1999.8 + 3.2 = 2003/2000 = 1.0015 access on average.   
   Data throughput is 1data per 1.0015 cycle.
7. 2000 read 10% of those will have data cache miss and we will need to access memory  
   That means, Cache hit cycles = 10% \* 2000 \* 1 = 1800 cycles.   
   And 200 will access memory with, 200 \* (1+15) = 3200 cycles  
   Total = 1800 + 3200 = 5000/2000 = 2.5 access on average.   
   Data throughput is 1data per 2.5 access