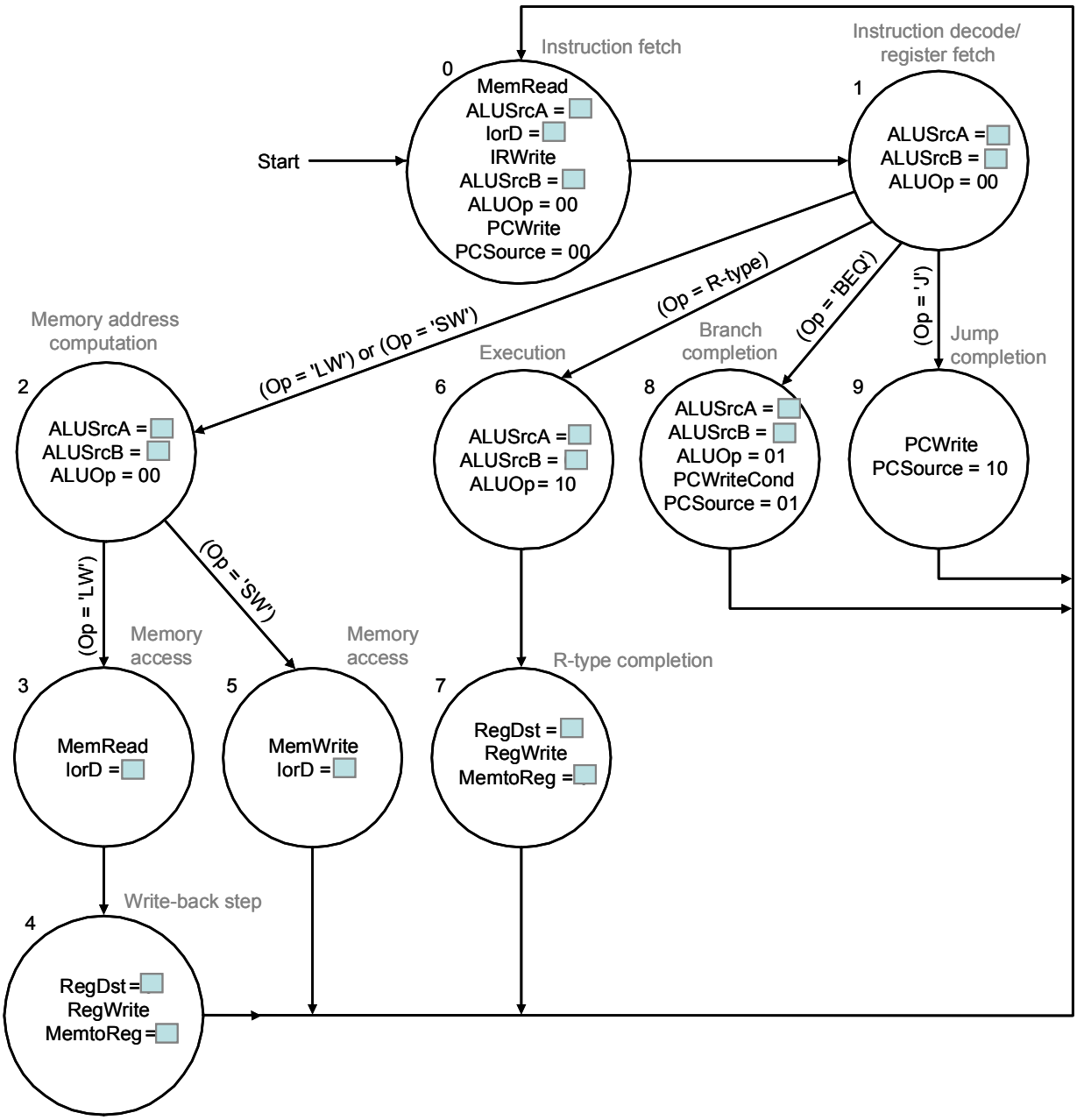
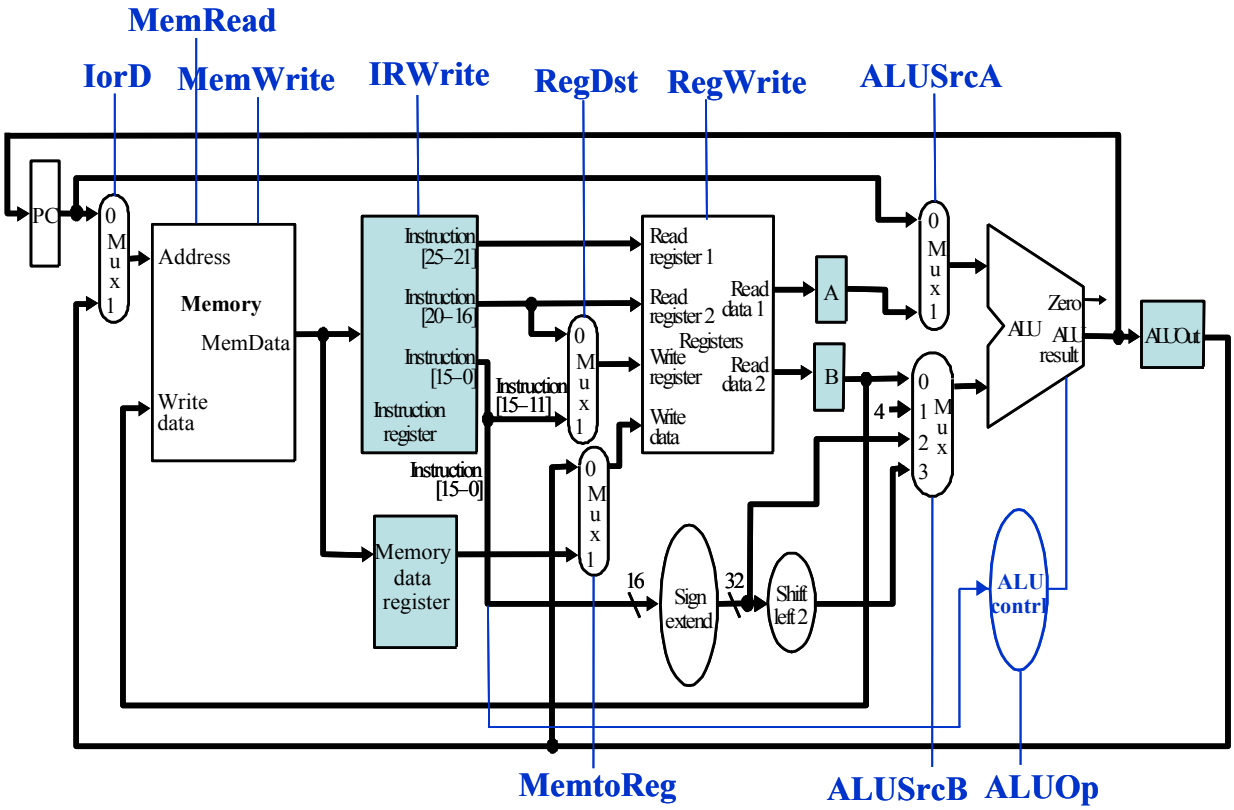
1. (Chapter 5) Fill in the values for ALUSrcA, ALUSrcB, IorD, RegDst and MemtoReg to complete the Finite State Machine for the multi-cycle datapath shown below.






| Step name | Action for R-type instructions | Action for memory-reference instructions | Action for branches | Action for jumps |
|---|---|---|---|---|
| Instruction fetch | IR <= Memory[PC]<br>PC <= PC + 4 | | | |
| Instruction decode/register fetch | A <= Reg [IR[25:21]]<br>B <= Reg [IR[20:16]]<br>ALUOut <= PC + (sign-extend (IR[15:0]) << 2) | | | |
| Execution, address computation, branch/ jump completion | ALUOut <= A op B | ALUOut <= A + sign-extend (IR[15:0]) | if (A ==B) then PC <= ALUOut | PC <= {PC [31-28], (IR[25:0],2'b00)} |
| Memory access or R-type completion | Reg [IR[15:11]] <= ALUOut | Load: MDR <= Memory[ALUOut]<br>or<br>Store: Memory [ALUOut] <= B | | |
| Memory read completion | | Load: Reg[IR[20-16]] <= MDR | | |

Answers: See Figure 5.38. Please note that there are two errors in state 4 in Figure 5.38. Specifically, RegDst should be 0 and MemtoReg should be 1. Please refer to Figure 5.33 for correct control signal assignment in state 4.

2. (Chapter 6) Consider the pipeline datapath below, and the two instructions, `add $rd, $rs, $rt` and `lw $rt, offset($rs)`.

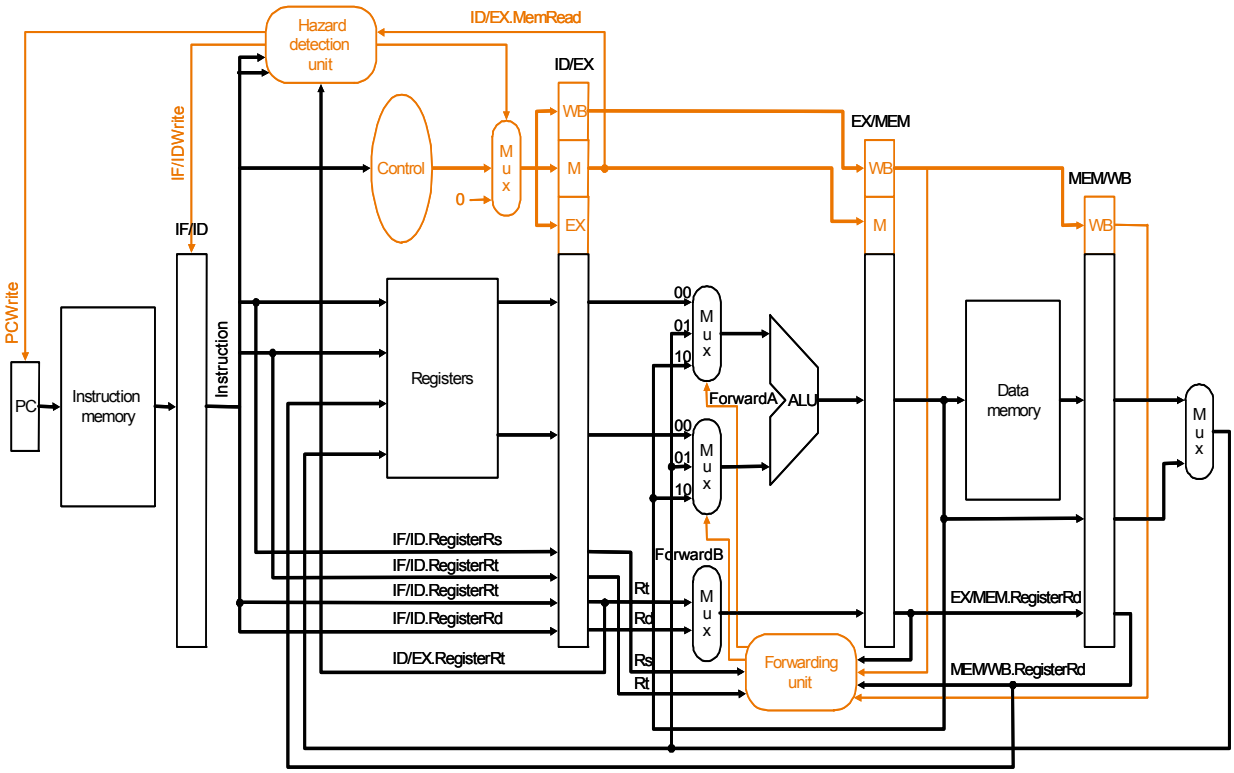
(a) Give a set of code sequences that only consist of `lw` and `add`, in which the pipeline stall can be resolved by forwarding. Justify your answer.

(b) What is the fowardA value (either 01 or 10), if the third instruction of the code sequence

```
add $1, $1, $2
add $1, $1, $3
add $1, $1, $4
```

is at its execution step? Justify your answer.

(c) Modify the register numbers used in the code in (b) so that forwardA becomes the *alternative* value. Note that the ultimate result in $1 after the execution of the code sequence must remain the same. (Hint: Introduce additional register, such as `$5`).



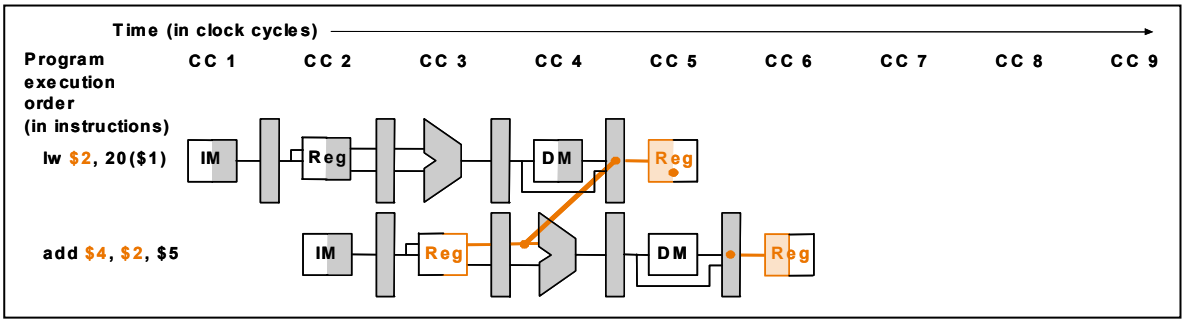

Answers:

(a) It is not possible to do *time-backward* forwarding; hence, the pipeline stall for the code sequence of

```
lw    $2, 20($1)
add   $4, $2, $6
```

cannot be resolved by forwarding technique because `lw` requires at least 4 clocks to obtain the accessed memory content, but the next instruction `add` needs the content one clock ahead. (See the multiple-clock-cycle pipeline diagram below.)

**Notably, the question is asking for a set of code sequences that only consist of `lw` and `add`, in which the pipeline stall can be resolved by forwarding; hence, any sequence of codes that do not require time-backward forwarding is a legitimate answer.**




(b) ForwardA = 10 since MEM/WB.RegisterRd = EX/MEM.RegisterRD = ID/EX.RegisterRs, and we shall use the more recent result.

(c) ForwardA = 01 when MEM/WB.RegisterRd = ID/EX.RegisterRs but EX/MEM.RegisterRD ≠ ID/EX.RegisterRs. Hence, the code

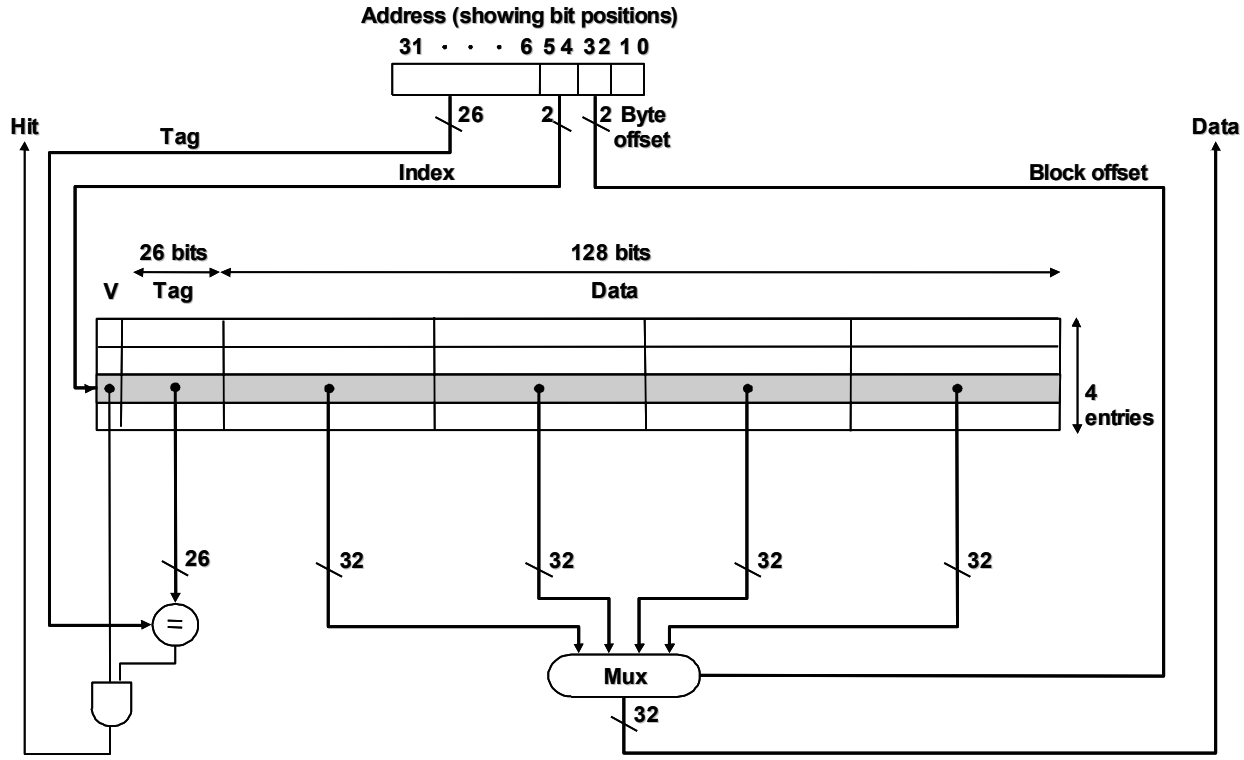
```
add $1, $1, $2
add $5, $3, $4
add $1, $1, $5
```

satisfies the need.

3. (Chapter 7) (a) For 32-bit byte addressing (in MIPS), how many tag bits are required for a direct-mapped cache with four-word blocks and a total size of 16 words.
(b) Here is a series of address references given as word addresses: 2, 3, 11, 16, 21, 13, 64, 48, 19, 11, 3, 22, 4, 27, 6, and 11. Tabular each reference in the list as a hit or miss and show the final contents of the cache.

Answers:

(a) 26 bits.




(b) Hit or miss for a directly mapped cache is determined by the required tag number and cache block number (i.e., index). The former can be derived by the formula of Integer[Address/16], while the ladder can be obtained using (Integer[Address/4] mod 4). Hence, for word addresses 2,

3, 11, 16, 21, 13, 64, 48, 19, 11, 3, 22, 4, 27, 6, and 11, the required tag-number/block-number are respectively 0/0(2), 0/0(3), 0/2(11), 1/0(16), 1/1(21), 0/3(13), 4/0(64), 3/0(48), 1/0(19), 0/2(11), 0/0(3), 1/1(22), 0/1(4), 1/2(27), 0/1(6), and 0/2(11). Therefore, the hit-and-miss list is as follows.

| (Address referenced) | Hit or miss | Tag | Address for content | Address for content | Address for content | Address for content |
|---|---|---|---|---|---|---|
| 0/0(2) | miss | 0 | 0 | 1 | 2 | 3 |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| 0/0(3) | hit | 0 | 0 | 1 | 2 | 3 |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| 0/2(11) | miss | 0 | 0 | 1 | 2 | 3 |
| | | | | | | |
| | | 0 | 8 | 9 | 10 | 11 |
| | | | | | | |
| 1/0(16) | miss | 1 | 16 | 17 | 18 | 19 |
| | | | | | | |
| | | 0 | 8 | 9 | 10 | 11 |
| | | | | | | |
| 1/1(21) | miss | 1 | 16 | 17 | 18 | 19 |
| | | 1 | 20 | 21 | 22 | 23 |
| | | 0 | 8 | 9 | 10 | 11 |
| | | | | | | |
| 0/3(13) | miss | 1 | 16 | 17 | 18 | 19 |
| | | 1 | 20 | 21 | 22 | 23 |
| | | 0 | 8 | 9 | 10 | 11 |
| | | 0 | 12 | 13 | 14 | 15 |
| 4/0(64) | miss | 4 | 64 | 65 | 66 | 67 |
| | | 1 | 20 | 21 | 22 | 23 |
| | | 0 | 8 | 9 | 10 | 11 |
| | | 0 | 12 | 13 | 14 | 15 |
| 3/0(48) | miss | 3 | 48 | 49 | 50 | 51 |
| | | 1 | 20 | 21 | 22 | 23 |
| | | 0 | 8 | 9 | 10 | 11 |
| | | 0 | 12 | 13 | 14 | 15 |
| 1/0(19) | miss | 1 | 16 | 17 | 18 | 19 |
| | | 1 | 20 | 21 | 22 | 23 |
| | | 0 | 8 | 9 | 10 | 11 |
| | | 0 | 12 | 13 | 14 | 15 |
| 0/2(11) | hit | 1 | 16 | 17 | 18 | 19 |
| | | 1 | 20 | 21 | 22 | 23 |
| | | 0 | 8 | 9 | 10 | 11 |
| | | 0 | 12 | 13 | 14 | 15 |
| 0/0(3) | miss | 0 | 0 | 1 | 2 | 3 |
| | | 1 | 20 | 21 | 22 | 23 |
| | | 0 | 8 | 9 | 10 | 11 |
| | | 0 | 12 | 13 | 14 | 15 |
| 1/1(22) | hit | 0 | 0 | 1 | 2 | 3 |
| | | 1 | 20 | 21 | 22 | 23 |
| | | 0 | 8 | 9 | 10 | 11 |
| | | 0 | 12 | 13 | 14 | 15 |
| 0/1(4) | miss | 0 | 0 | 1 | 2 | 3 |
| | | 0 | 4 | 5 | 6 | 7 |
| | | 0 | 8 | 9 | 10 | 11 |
| | | 0 | 12 | 13 | 14 | 15 |

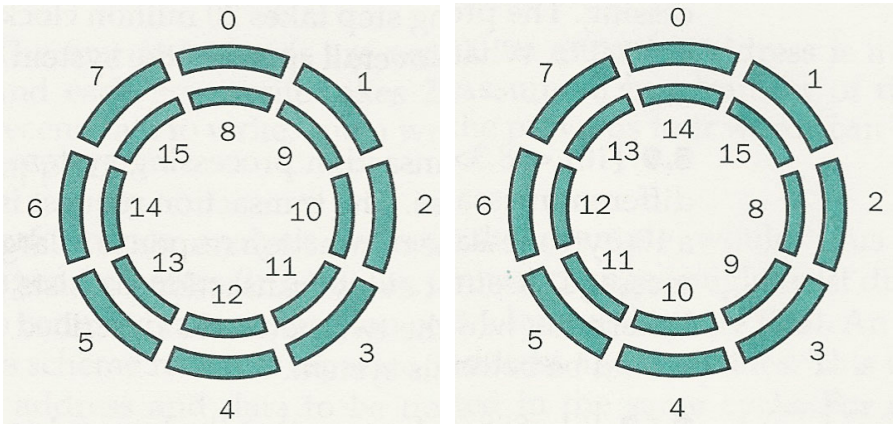
| | | | | | | |
|---|---|---|---|---|---|---|
| 1/2(27) | miss | 0 | 0 | 1 | 2 | 3 |
| | | 0 | 4 | 5 | 6 | 7 |
| | | 1 | 24 | 25 | 26 | 27 |
| | | 0 | 12 | 13 | 14 | 15 |
| 0/1(6) | hit | 0 | 0 | 1 | 2 | 3 |
| | | 0 | 4 | 5 | 6 | 7 |
| | | 1 | 24 | 25 | 26 | 27 |
| | | 0 | 12 | 13 | 14 | 15 |
| 0/2(11) | miss | 0 | 0 | 1 | 2 | 3 |
| | | 0 | 4 | 5 | 6 | 7 |
| | | 0 | 8 | 9 | 10 | 11 |
| | | 0 | 12 | 13 | 14 | 15 |

Note that the numbers in black in the table are not requested parts of the answer. Only those numbers in blue are necessary.

4. (Chapter 8)

(a) Calculate the average time to read a 512-byte sector for a disk with 5ms average seek time, 6000 RPM, 5MB/sec transfer rate and 2ms controller overhead. (Here, $M = 2^{20}$. Assume that the average rotational latency is the time to rotate 180 degree for the disk.)

(b) Consider the following diagram of two potential ways of numbering the sectors of data on a disk. Which way of numbering the sectors will be likely to result in higher performance, if typical reads are contiguous in sector numbers? Justify your answer.




Answer:

(a) Average seek time + average rotational delay + transfer time + controller over head = 5 msec + 0.5/(6000/60) sec + 0.5 KB/ (5MB/sec) + 2 msec = 5ms + 5ms + 0.1ms + 2ms = 12.1ms.

(b) After reading sector 7, a seek is necessary to get to the track with sector 8 on it. This will take some time (on the order of a millisecond, typically), during which the disk will continue to revolve under the head assembly. Thus, in the version where sector 8 is in the same angular position as sector 0, sector 8 will have already revolved past the head by the time the seek is completed and some large fraction of additional revolution time will be needed to wait for it to come back again. By skewing the sectors so that sector 8 starts later on the second track, the seek will have time to complete, and then the sector will soon thereafter appear under the head without additional revolution.