1 DRAM Refresh [80 points]

A memory system is composed of eight banks, and each bank contains 32K rows. The row size is 8 KB.

Every DRAM row refresh is initiated by a command from the memory controller, and it refreshes a single row. Each refresh command keeps the command bus busy for 5 ns.

We define *command bus utilization* as a fraction of the total time during which the command bus is busy due to refresh.

The retention time of each row depends on the temperature (T). The rows have different retention times, as shown in the following Table 1:

Retention Time	Number of rows
$(128 - T)$ ms, $0°C \leq T \leq 128°C$	2^8 rows
$2 * (128 - T)$ ms, $0°C \leq T \leq 128°C$	2^{16} rows
$4 * (128 - T)$ ms, $0°C \leq T \leq 128°C$	all other rows
$8 * (128 - T)$ ms, $0°C \leq T \leq 128°C$	2^8 rows

Table 1: Retention time

1.1 Refresh Policy A [20 points]

Assume that the memory controller implements a refresh policy where all rows are refreshed with a fixed refresh interval, which covers the worst-case retention time (Table 1).

(a) [5 points] What is the maximum temperature at which the DRAM can operate reliably with a refresh interval of 32 ms?

T=96°C.

Explanation. Looking at the first column of Table 1, it is obvious that the DRAM rows with the least retention time (i.e., the first row of the table) can operate at temperatures up to 96°C (128 - 96) when refreshed at fixed 32 ms refresh period.

(b) [15 points] What command bus utilization is directly caused by DRAM refreshes (with refresh interval of 32 ms)?

4.096%.

Explanation. The time that the command bus spends on refresh commands in 32ms is 2^{18}*5ns, where 2^{18} is the number of rows (8 banks, 32K rows per banks), and 5ns is the time that the bus is busy for each refresh command.
Utilization = $\frac{2^{18}*5ns}{32ms}$ = 4.096%

1.2 Refresh Policy B [15 points]

Now assume that the memory controller implements a refresh policy where all rows are refreshed only *as frequently as required* to correctly maintain their data (Table 1).

(a) [15 points] How many refreshes are performed by the memory controller during a 1.024 second period? (with T=64°C)

The number of refreshes in 1.024 seconds is 1313280.

Explanation. 2^8 rows are refreshed every 64ms $\implies$ (1024/64) * 2^8 = 2^{12} refresh commands in 1.024 seconds.
2^{16} rows are refreshed every 128ms $\implies$ (1024/128) * 2^{16} = 2^{19} refresh commands in 1.024 seconds.
2^8 rows are refreshed every 512ms $\implies$ (1024/512) * 2^8 = 2^9 refresh commands in 1.024 seconds.

The remaining rows are $2^{18} - (2^8 + 2^{16} + 2^8) = 2^{18} - 2^{16} - 2^9$, so: $2^{18} - 2^{16} - 2^9$ rows are refreshed every 256ms $\implies$ (1024/256) * $(2^{18} - 2^{16} - 2^9) = 2^{20} - 2^{18} - 2^{11}$ refresh commands in 1.024 second.

Total: $2^{12} + 2^{19} + 2^9 + 2^{20} - 2^{18} - 2^{11}$ = 1313280 refresh commands in 1.024 second.

1.3 Refresh Policy C [25 points]

Assume that the memory controller implements an even smarter policy to refresh the memory. In this policy, the refresh interval is fixed, and it covers the worst-case retention time (64ms), as the refresh policy in part 1.1. However, as an optimization, a row is refreshed only if it has *not* been **accessed** during the past refresh interval. For maintaining correctness, if a cell reaches its maximum retention time without being refreshed, the memory controller issues a refresh command.

(a) [5 points] Why does a row *not* need to be refreshed if it was accessed in the past refresh interval?

The charge of a DRAM cell is restored when it is accessed.

(b) [20 points] A program accesses all the rows repeatedly in the DRAM. The following table shows the access interval of the rows, the number of rows accessed with the corresponding access interval, and the retention times of the rows that are accessed with the corresponding interval.

Access interval	**Number of rows**	**Retention times**
1ms	2^{16} rows	64ms, 128ms or 256ms
60ms	2^{16} rows	64ms, 128ms or 256ms
128ms	all other rows	128ms or 256ms

What command bus utilization is directly caused by DRAM refreshes?

Command bus utilization: 0.512%.

Explanation. The rows that are accessed every 1 ms and every 60 ms do *not* need to be refreshed.

The remaining rows (2^{17}) are accessed once every two refresh intervals (128 ms). So, the command bus utilization is $\frac{2^{17} * 5ns * 100}{128ms} = 0.512\%$

1.4 Refresh Policy D [20 points]

Assume that the memory controller implements an approximate mechanism to reduce refresh rate using Bloom filters, as we discussed in class. For this question we assume the retention times in Table 1 with a constant temperature of $64°C$.

One Bloom filter is used to represent the set of all rows that require a 64 ms refresh rate.

The memory controller's refresh logic is modified so that on every potential refresh of a row (every 64 ms), the refresh logic probes the Bloom filter. If the Bloom filter probe results in a "hit" for the row address, then the row is refreshed. Any row that does *not* hit in the Bloom filter is refreshed at the default rate of once per 128 ms.

(a) [20 points] The memory controller performs 2107384 refreshes in total across the channel over a time interval of 1.024 seconds. What is the false positive rate of the Bloom filter? (NOTE: False positive rate = Total number of false positives / Total number of accesses).

- Hint: $2107384 = 2^3 * (2^{18} + 2^{11} - 2^{10} + 2^9 - 2^8 - 1)$

False Positive rate = $(2^{10} - 1)/2^{18}$.

Explanation. First, there are 2^8 rows that need to be refreshed every 64ms. All the other rows ($2^{18} - 2^8$) will be refreshed every 128ms.

At 64 ms, we have 2^8 rows which actually require this rate, plus P out of the $2^{18} - 2^8$ other rows which will be false-positives. In a period of 1.024s, this bin is refreshed 16 times. The number of refreshes for this bin is $16 * (2^8 + P * (2^{18} - 2^8))$.

At 128 ms we have the rest of the rows which are $2^{18} - 2^8$ minus the false positives refreshed in the 64ms bin. So, the number of rows are $2^{18} - 2^8 - P * (2^{18} - 2^8)$. In a period of 1.024s, this bin is refreshed 8 times. The number of refreshes for this bin is $8 * ((2^{18} - 2^8) - (P * (2^{18} - 2^8))$

Thus, in total we have $16 * (2^8 + P * (2^{18} - 2^8)) + 8 * ((2^{18} - 2^8) - P * (2^{18} - 2^8)) = 2107384$. Solving the equation, P=2^{-8}.

Finally, False positive rate = Total number of false positives / Total number of accesses = $P * (2^{18} - 2^8))/2^{18} = 2^{-8} * (2^{18} - 2^8)/2^{18} = (2^{10} - 1)/2^{18}$

2 DRAM Scheduling and Latency [80 points]

You would like to understand the configuration of the DRAM subsystem of a computer using reverse engineering techniques. Your current knowledge of the particular DRAM subsystem is limited to the following information:

- The physical memory address is 16 bits.
- The DRAM subsystem consists of a single channel and 4 banks.
- The DRAM is byte-addressable.
- The most-significant 2 bits of the physical memory address determine the bank.
- The DRAM command bus operates at 500 MHz frequency.
- The memory controller issues commands to the DRAM in such a way that *no command* for servicing a *later* request is issued before issuing a READ command for the current request, which is the oldest request in the request buffer. For example, if there are requests A and B in the request buffer, where A is the older request and the two requests are to different banks, the memory controller does *not* issue an ACTIVATE command to the bank that B is going to access *before* issuing a READ command to the bank that A is accessing.

You realize that you can observe the memory requests that are waiting to be serviced in the request buffer. At a particular point of time, you take the snapshot of the request buffer and you observe the following requests in the request buffer.

Requests in the request buffer (in descending order of request age, where the oldest request is on the top):

time ↓

```
Read 0x4C80
Read 0x0140
Read 0x4EC0
Read 0x8000
Read 0xF000
Read 0x803F
Read 0x4E80
```

At the same time you take the snapshot of the request buffer, you start probing the DRAM command bus. You observe the DRAM command type and the cycle (relative to the first command) at which the command is seen on the DRAM command bus. The following are the DRAM commands you observe on the DRAM bus while the requests above are serviced.

```
Cycle 0 --- PRECHARGE
Cycle 6 --- ACTIVATE
Cycle 10 --- READ
Cycle 11 --- READ
Cycle 21 --- PRECHARGE
Cycle 27 --- ACTIVATE
Cycle 31 --- READ
Cycle 32 --- ACTIVATE
Cycle 36 --- READ
Cycle 37 --- READ
Cycle 38 --- READ
Cycle 42 --- PRECHARGE
Cycle 48 --- ACTIVATE
Cycle 52 --- READ
```

Answer the following questions using the information provided above.

(a) [15 points] What are the following DRAM timing parameters used by the memory controller, in terms of nanoseconds?

i) ACTIVATE-to-READ latency

8 ns.

Explanation. After issuing the ACTIVATE command at cycle 6, the memory controller waits until cycle 10, which indicates that the ACTIVATE-to-READ latency is 4 cycles. The command bus operates at 500 MHz, so it has 2 ns clock period. Thus, the ACTIVATE-to-READ is $4 * 2 = 8$ ns.

ii) ACTIVATE-to-PRECHARGE latency

30 ns.

Explanation. The bank activated at cycle 6 is precharged at cycle 21. Although the memory controller is idle after cycle 11, it waits until cycle 21, which means the ACTIVATE-to-PRECHARGE latency restricts the memory controller from issuing the PRECHARGE command earlier. Thus, the ACTIVATE-to-PRECHARGE latency is 15 cycles = 30 ns.

iii) PRECHARGE-to-ACTIVATE latency

12 ns.

Explanation. The PRECHARGE-to-ACTIVATE latency can be easily seen in the first two commands at cycles 0 and 6. The PRECHARGE-to-ACTIVATE latency is 6 cycles = 12 ns.

(b) [20 points] What is the row size in bytes? Explain your answer.

64 bytes.

Explanation. The Read request to address 0x803F (to Bank 2) does not require an ACTIVATE command, which means there is a row hit for that access. The open row was activated by the command issued for the request to the address 0x8000. That means the target rows of both requests should be the same. When we look at the binary form of those addresses, we see that the least significant 6 bits are different (000000 for 0x8000 and 111111 for 0x803F). That means at least 6 of the least significant bits should be the column bits.
Later, when we look at the commands issued for the requests to 0x4ECO and 0x4E80, we see that for both of those requests, the memory controller has opened a new row. Thus, the target rows of those requests should be different. Since only the 6th bit (assuming the least significant bit is the 0th bit) is different between the two addresses, the 6th bit should be part of the row address. So, of the 6th bit is part of the row address, the number of columns bits should be 6 or less. As we previously found from the requests to 0x8000 and 0x803F that the number of column bits should be at least 6, combining those two findings we can say that the number of column bits should be exactly 6. Thus, the row size is $2^6 = 64$ bytes.

(c) [20 points] What is the status of the banks *prior* to the execution of any of the the above requests? In other words, which rows from which banks were open immediately prior to issuing the DRAM commands listed above? Fill in the table below indicating whether a bank has an open row, and if there is an open row, specify its address. If there is not enough information to infer the open row address, write *unknown*.

	Open or Closed?	**Open Row Address**
Bank 0	Open	5
Bank 1	Open	Unknown
Bank 2	Closed	—
Bank 3	Open	192

Explanation. By decoding the accessed addresses we can find which bank and row each access targets. Looking at the commands issued for those requests, we can determine which requests needed PRECHARGE (row buffer conflict, the initially open row is unknown in this case), ACTIVATE (the bank is initially closed), or directly READ (the bank is initially open and the open row is the same as the one that the request targets).

time ↓

```
0x4C80 → Bank:  1, Row:  50 (PRECHARGE first. Any row other than 50 could be opened.)
0x0140 → Bank:  0, Row:  5 (Row hit, so Bank 0 must have row 5 open.)
0x4EC0 → Bank:  1, Row:  59
0x8000 → Bank:  2, Row:  0 (ACTIVATE is issued first. That means the bank was already closed.)
0xF000 → Bank:  3, Row:  192 (Row hit, so Bank 3 must have row 192 open.)
0x803F → Bank:  2, Row:  0
0x4E80 → Bank:  1, Row:  58
```

(d) [25 points] To improve performance, you decide to implement the idea of Tiered-Latency DRAM (TL-DRAM) in the DRAM chip. Assume that a bank consists of a single subarray. With TL-DRAM, an entire bank is divided into a near-segment and far-segment. When accessing a row in the near-segment, the ACTIVATE-to-READ latency *reduces* by 2 cycles and the ACTIVATE-to-PRECHARGE latency reduces by 5 cycles. When accessing a row in the far-segment, the ACTIVATE-to-READ latency *increases* by 1 cycle and the ACTIVATE-to-PRECHARGE latency increases by 2 cycles.

Assume that the rows in the near-segment have smaller row ids compared to the rows in the far-segment. In other words, physical memory row addresses 0 through $N-1$ are the near-segment rows, and physical memory row addresses N through $M-1$ are the far-segment rows.

If the above DRAM commands are issued 5 cycles faster with TL-DRAM compared to the baseline (the last command is issued in cycle 47), how many rows are in the near-segment? Show your work.

59 rows have to be in the near segment.

Explanation. There should 59 rows in the near-segment (rows 0 to 58) since rows until row id 58 need to be accessed with low latency to get 5 cycle reduction. Rows 59 and 192 are in the far-segment, thus latency for accessing them increases slightly.
Here is the new command trace:

```
Cycle 0 -- PRECHARGE - Bank 1
Cycle 6 -- ACTIVATE - Bank 1, Row 50, near segment
Cycle 8 -- READ - Bank 1
Cycle 9 -- READ - Bank 0
Cycle 16 -- PRECHARGE - Bank 1
Cycle 22 -- ACTIVATE - Bank 1, Row 59, far segment
Cycle 27 -- READ - Bank 1
Cycle 28 -- ACTIVATE - Bank 2, Row 0
Cycle 30 -- READ - Bank 2
Cycle 31 -- READ - Bank 3
Cycle 32 -- READ - Bank 2
Cycle 39 -- PRECHARGE - Bank 1
Cycle 45 -- ACTIVATE - Bank 1, Row 58, near segment
Cycle 47 -- READ - Bank 1
```