Consider the following program:

```c
int X[N];
int step = M; // M is some predefined constant
for(int i = 0; i < N; i += step )
    X[i] = X[i] + 1;
```

1. If the program is run on a machine with a 4-KB page size and 64-entry TLB, what values of M and N will cause a TLB miss for every execution of the inner loop?

2. Would your answer in part 1. be different if the loop were repeated many times? Explain.
Consider the following program:

```c
int X[N];
int step = M; // M is some predefined constant
for(int i = 0; i < N; i += step)
    X[i] = X[i] + 1;
```

1. If the program is run on a machine with a 4-KB page size and 64-entry TLB, what values of M and N will cause a TLB miss for every execution of the inner loop?

   Explain.

   1. M has to be at least 4,096 to ensure a TLB miss for every access to an element of X. Since N only affects how many times X is accessed, any value of N will do.

   2. M should still be at least 4,096 to ensure a TLB miss for every access to an element of X. But now N should be greater than 64K to trash the TLB, that is, X should exceed 256KB.
Problem 9

A machine has a 32-bit address space and an 8-KB page. The page table is entirely in hardware, with one 32-bit word per entry. When a process starts, the page table is copied to the hardware from memory, at one word every 100 nsec.

1. If each process runs for 100 msec (including the time to load the page table), what fraction of the CPU time is devoted to loading the page tables?
Problem 9

A machine has a 32-bit address space and an 8-KB page. The page table is entirely in hardware, with one 32-bit word per entry. When a process starts, the page table is copied to the hardware from memory, at one word every 100 nsec.

1. If each process runs for 100 msec (including the time to load the page table), what fraction of the CPU time is devoted to loading the page tables?

The page table contains $2^{32} / 2^{13}$ entries, which are 524,288. Loading the page table takes 52 msec. If a process gets 100 msec, this consists of 52 msec for loading the page table and 48 msec for running. Thus 52% of the time is spent loading page tables.
Problem 16

The TLB on the VAX does not contain an R bit. Why?
Problem 16

The TLB on the VAX does not contain an R bit. Why?

The R bit is never needed in the TLB. The mere presence of the page there means the page has been referenced; otherwise it would not be there. Thus the bit is completely redundant. When the entry is written back to memory, however, the R bit in the memory page table is set.
Problem 20

A student in a compiler design course propose to the professor a project of writing a compiler that will produce a list of page references that can be used to implement the optimal page replacement algorithm.

1. Is this possible? Why or why not?
2. Is there anything that could be done to improve paging efficiency at run time?
Problem 20

A student in a compiler design course propose to the professor a project of writing a compiler that will produce a list of page references that can be used to implement the optimal page replacement algorithm.

1. Is this possible? Why or why not?

2. Is there anything that could be done to improve paging efficiency at run time?

This is probably not possible except for the unusual and not very useful case of a program whose course of execution is completely predictable at compilation time. If a compiler collects information about the locations in the code of the calls to procedures, this information might be used at link time to rearrange the object code so that procedures were located close to the code that calls them. This would make it more likely that a procedure would be on the same page as the calling code. Of course this would not help much for procedures called from many places in the program.
Problem 21

Suppose that the virtual page reference stream contains repetitions of long sequences of page references followed occasionally by a random page reference. For example the sequence: 0, 1, …, 511, 431, 0, 1, …, 511, 332, 0, 1, … consists of repetitions of the sequence 0, 1, …, 511 followed by a random reference to page 431 and 332.

1. Why won’t the standard replacement algorithms (LRU, FIFO, Clock) be effective in handling this workload for a page allocation that is less than the sequence length?

2. If this program were allocated 500 page frames, describe a page replacement approach that would perform much better than LRU, FIFO or Clock algorithms.
Problem 21

Suppose that the virtual page reference stream contains repetitions of long sequences of page references followed occasionally by a random page reference. For example the sequence: 0, 1, ..., 511, 431, 0, 1, ..., 511, 332, 0, 1, ... consists of repetitions of the sequence 0, 1, ..., 511 followed by a random reference to page 431 and 332.

1. Why won't the standard replacement algorithms (LRU, FIFO, Clock) be effective in handling this workload for a page allocation that is less than the sequence length?

2. If this program were allocated 500 page frames, describe a page replacement approach that would perform much better than LRU, FIFO or Clock algorithms.

1) Every reference will page fault unless the number of page frame is 512, the length of the entire sequence.

2) If there are 500 frames, map pages 0-498 to fixed frames and vary only one frame
Problem 29

Consider the following two-dimensional array:

```
int X[64][64];
```

Suppose that a system has four page frames and each frame is 128 words (an integer occupies one word). Programs that manipulate the X array fit into exactly one page and always occupy page 0. The data are swapped in and out of the other three frames. The X array is stored in row-major order (i.e., X[0][1] follows X[0][0] in memory). Which of the two code fragments shown below will generate the lowest number of page faults? Explain and compute the total number of page faults.

Fragment A

```
for (int j=0; j<64; j++)
    for (int i=0; i<64; i++)
        X[i][j] = 0;
```

Fragment B

```
for (int i=0; i<64; i++)
    for (int j=0; j<64; j++)
        x[i][j] = 0;
```
Problem 29

Consider the following two-dimensional array:

```c
int X[64][64];
```

Suppose that a system has four page frames and each frame is 128 words (an integer occupies one word). Programs that manipulate the X array fit into exactly one page and always occupy page 0. The data are swapped in and out of the other three frames. The X array is stored in row-major order (i.e., X[0][1] follows X[0][0] in memory). Which of the two code fragments shown below will generate the lowest number of page faults? Explain and compute the total number of page faults.

<table>
<thead>
<tr>
<th>Fragment A</th>
<th>Fragment B</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>for(int j=0; j&lt;64;j++)</code></td>
<td><code>for(int i=0;i&lt;64;i++)</code></td>
</tr>
<tr>
<td><code>for(int i=0;i&lt;64;i++)</code></td>
<td><code>X[i][j]=0;</code></td>
</tr>
<tr>
<td><code>X[i][j]=0;</code></td>
<td><code>x[i][j]=0;</code></td>
</tr>
</tbody>
</table>

Fragment B since the code has more spatial locality than fragment A. The inner loop causes only one page fault for every other iteration of the outer loop. (There will be 32 page faults.) Aside (Fragment A): Since a frame is 128 words, one row of the X array occupies half of a page (i.e., 64 words). The entire array fits into 64 x 32 / 128 = 16 frames. The inner loop of the code steps through consecutive rows of X for a given column. Thus every other reference to X[i][j] will cause a page fault. The total number of page faults will be 64 x 64 / 2 = 2,048.
Problem 34

A group of operating system designers for the Frugal Computer Company are thinking about ways to reduce the amount of backing store needed in their new operating system. The head guru has just suggested not bothering to save the program text in the swap area at all, but just page it in directly from the binary file whenever it is needed.

1. Under what conditions, if any, does this idea work for the program text?
2. Under what conditions, if any, does it work for the data?
Problem 34

A group of operating system designers for the Frugal Computer Company are thinking about ways to reduce the amount of backing store needed in their new operating system. The head guru has just suggested not bothering to save the program text in the swap area at all, but just page it in directly from the binary file whenever it is needed.

1. Under what conditions, if any, does this idea work for the program text?

2. Under what conditions, if any, does it work for the data?

It works for the program if the program cannot be modified. It works for the data if the data cannot be modified. However, it is common that the program cannot be modified and extreme rare that the data cannot be modified. If the data area on the binary file were overwritten with updated pages, the next time the program was started, it would not have the original data.
Problem 38

Can you think of any situation where supporting virtual memory would be a bad idea, and what would be gained by not having to support virtual memory? Explain.
Can you think of any situation where supporting virtual memory would be a bad idea, and what would be gained by not having to support virtual memory?

General virtual memory is not needed when the memory requirements of all applications are well known and controlled. Some examples are special-purpose processors (e.g., network processors), embedded processors, and super-computers (e.g., airplane wing design). In this situations, we should always consider the possibility of using more real memory. If the operating system did not have to support virtual memory, the code would be much simpler and smaller. On the other hand, some ideas from virtual memory may still be profitably exploited, although with different design requirements. For example, program/thread isolation might be paging to flash memory.