1. Consider a system with a total of 150 units of memory, allocated to three processes as shown:

<table>
<thead>
<tr>
<th>Process</th>
<th>Max</th>
<th>Hold</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>70</td>
<td>45</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>60</td>
<td>15</td>
</tr>
</tbody>
</table>

Apply banker’s algorithm to determine whether it would be safe to grant each of the following requests. If yes, indicate a sequence of terminations that could be guaranteed possible. If no, show the reduction of resulting allocation table.

(a) A fourth process arrives, with a maximum memory need of 60 and an initial need of 25 units.
(b) A fourth process arrives, with a maximum memory need of 60 and an initial need of 35 units.

a. Creating the process would result in the state:

<table>
<thead>
<tr>
<th>Process</th>
<th>Max</th>
<th>Hold</th>
<th>Claim</th>
<th>Free</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>70</td>
<td>45</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>40</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>60</td>
<td>15</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>60</td>
<td>25</td>
<td>35</td>
<td></td>
</tr>
</tbody>
</table>

There is sufficient free memory to guarantee the termination of either P1 or P2. After that, the remaining three jobs can be completed in any order.

b. Creating the process would result in the trivially unsafe state:

<table>
<thead>
<tr>
<th>Process</th>
<th>Max</th>
<th>Hold</th>
<th>Claim</th>
<th>Free</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>70</td>
<td>45</td>
<td>25</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>40</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>60</td>
<td>15</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>60</td>
<td>35</td>
<td>25</td>
<td></td>
</tr>
</tbody>
</table>

2. A typical program has 20% memory instructions. Assume there are 5% data TLB misses, each requiring 100 cycles to handle.
Assume each instruction requires 1 cycle to execute, each memory operation in the cache is 1 cycle, 10% data accesses are cache misses, each cache miss is 15 cycle. How long would it take to execute 1000 instructions?

Overhead on Cache misses = 14 Cycles
Overhead on TLB misses = 99 cycles

# cache misses = 1000 * 20% * 10% = 20
# TLB misses = 1000 * 20% * 5% = 10
Total cycles = 1000 cycles + (overhead due to cache misses) + (overhead due to TLB misses)
= 1000 + (20 * 14) + (10 * 99) = 1000 + 280 + 990 = 2270 cycles

3. If virtual address space supported is 2^{64} bits, the page size is 1 Kbyte, the size of physical memory is 64 Kbyte, the size of PTE is two bytes, and the addressing is at byte level, calculate the size of the page table required for both standard and inverted page tables.

Standard page table:
Address space is 4 bits in byte addressing (2^4 = 2^7 / 2^3)
Number of pages = 2^4 / 1K = 2^4 / 210;
Page Table Size = 2^4 / 2^{10} * (PTE size) = 2^4 / 2^{10} * (2) = 25 / 2^{10} bytes

Inverted page table:
Total frames = Physical Memory size/Page size = 64K/1K = 64;
Page Table Size = 64 * (PTE size) = 128 bytes

4. Consider the following program:
```c
const int n = 25;
int tally;
void total()
{
    int count;
    for (count = 1; count <= n; count++)
    {
        tally++;
    }
}
void main()
{
    tally = 0;
    parbegin (total(), total());
    write (tally);
}
```

Determine the proper lower bound and upper bound on the final value of the shared variable ‘tally’ output by this concurrent program. Assume processes can execute at any relative speed and a value can only be incremented after it has been loaded into a register by a separate machine instruction.

On casual inspection, it appears that ‘tally’ will fall in the range 25 ≤ tally ≤ 50 since from 0 to 20 increments could go unrecorded due to the lack of mutual exclusion. The basic argument contends that by running these two processes concurrently we should not be able to derive a result lower than the result produced by executing just one of these processes sequentially. But consider the following interleaved sequence of the load, increment, and store operations performed by these two processes when altering the value of the shared variable:
1. Process A loads the value of \( t \), increments \( t \), but then loses the processor (it has incremented its register to 1, but has not yet stored this value.

2. Process B loads the value of \( t \) (still zero) and performs forty-nine complete increment operations, losing the processor after it has stored the value 24 into the shared variable \( t \).

3. Process A regains control long enough to perform its first store operation (replacing the previous \( t \) value of 24 with 1) but is immediately forced to relinquish the processor.

4. Process B resumes long enough to load 1 (the current value of \( t \)) into its register, but then it too is forced to give up the processor (note that this was B's final load).

5. Process A is rescheduled, but this time it is not interrupted and runs to completion, performing its remaining 24 load, increment, and store operations, which results in setting the value of \( t \) to 25.

6. Process B is reactivated with only one increment and store operation to perform before it terminates. It increments its register value to 2 and stores this value as the final value of the shared variable.

Some thought will reveal that a value lower than 2 cannot occur. Thus, the proper range of final values is \( 2 \leq t \leq 50 \).

5. (a) Consider the following solution for mutual exclusion:
   
   ```
   repeat
     while Test-And-Set(lock) do no-op;
     critical section
       lock := false;
     remainder section
   until false;
   ```

   This solution for mutual exclusion using \texttt{Test-And-Set} does NOT satisfy
   
   A. Mutual exclusion
   B. Progress
   C. Bounded waiting
   D. All of the above
   E. None of the above

   (b) Assuming non-trivial context switch time, which of the following algorithm(s) gives the best throughput on all inputs? [1.5]
   
   A) Shortest Job First
   B) First Come, First Served
   C) Round Robin
   D) Shortest Time Remaining Next
   E) All of the above

   *Non-preemptive schedulers will achieve best CPU Utilization.

4. The following system uses both segmentation and paging;
(a) Label the entries on the diagram above; Segment Table, Physical Memory, Segment Table base register and Physical Memory
(b) Add a translation look aside buffer and redraw the diagram

(a) Segment Table B, Physical Memory D, Segment Table base register A, Page Table C
(b) Two possible solutions are accepted for this problem. The TLB of the first solution bypass only the paging portion of the system. Thus, the input to TLB is the virtual address after already performing the segmentation translation. In the second solution, both the paging and segmentation portions are bypassed. Thus both the segmentation and paging portions of the virtual address are used as the input to the TLB. In both cases, the offset must be added onto the translated address so the TLB output is added to the offset portions of the address.
7. Can following three processes be scheduled and still meet all deadlines? 
   Process A – Respond to an event every 3 seconds that requires 1 second 
   Process B – Respond to an event every 6 seconds that requires 3 seconds 
   Process C – Respond to an event every 6 seconds that requires 1 second 

Can be scheduled – 1/3+3/6+1/6=(2+3+1)/6=6/6=1

8. Write a monitor that implements a alarm clock that enables a calling program to delay itself for a specified number of time units (ticks). You may assume the existence of a real hardware clock that invokes a procedure tick in your monitor at regular intervals.

```c
monitor alarm {
    condition c;
    
    void delay (int ticks) {
        int begin_time = read_clock();
        while (read_clock() < begin_time + ticks);
        c.wait();
    }
    
    void tick() {
        c.broadcast ();
    }
}
```