ANSWERS TO THE 2013 COMPUTER SYSTEMS EXAM QUESTIONS

1. (a) We can summarise as `((r1)+1)*((r2)*2)*((r1)+3)`, where `(ri)` denotes the content of the memory location whose address is in `ri`.

   (b) Immediate addressing: used, e.g., in `I2`, where `#1` refers to `1`.

   Register addressing: used, e.g., in `I3`, where `r4` refers to the content of register `r4`.

   Register indirect addressing: used, e.g., in `I1`, where `(r1)` refers to the content of the memory cell whose address is in register `r1`.

   (c) Write–read (or true data) dependency: all the pairs of occurrences of a register `ri` where instruction `Ij` writes to `ri` and instruction `Ik` reads from `ri` with `j` < `k`. For instance, `I4-I6` on `r2`.

   Write–write dependency: all the pairs of occurrences of a register `ri` where instruction `Ij` writes to `ri` and instruction `Ik` also writes to `ri` with `j` < `k`. For instance, `I2-I5` on `r4`.

   Read–write dependency: all the pairs of occurrences of a register `ri` where instruction `Ij` reads from `ri` and instruction `Ik` writes to `ri` with `j` < `k`. For instance, `I3-I8` on `r4`.

2. (a) Superscalar processors try to exploit the instruction-level parallelism, the degree of independence, in the code. They apply branch prediction techniques (to determine which instructions will be executed), register renaming (to remove false dependencies like write–write and read–write from the code) and reorder instructions so that several instructions can be executed at the same time on the available execution units.

   (b) First we remove the false dependencies by renaming registers:

```
 I1: LOAD r3 (r1)
 I2: MOVE r4 #1
 I3: ADD r3 r3 r4
 I4: LOAD r2 (r2)
 I5: MOVE r41 #2
 I6: MUL r2 r2 r41
 I7: MUL r3 r3 r2
 I8: LOAD r42 (r1)
 I9: MOVE r1 #3
I10: ADD r42 r42 r1
I11: MUL r3 r3 r42
```

   Then the code can be reordered, for instance, as follows:

```
 I1: LOAD r3 (r1)
 I2: MOVE r4 #1
 I4: LOAD r2 (r2)
 I5: MOVE r41 #2
 I3: ADD r3 r3 r4
 I6: MUL r2 r2 r41
 I8: LOAD r42 (r1)
 I9: MOVE r1 #3
 I7: MUL r3 r3 r2
I10: ADD r42 r42 r1
I11: MUL r3 r3 r42
```

We make the following assumptions for the diagram below:
1. For load and move instructions the address calculation is done on the in the decode phase, hence the adder and multiplier are not involved in the execution of these instructions.
2. There is an instruction window where decoded instructions can be stored.

	F-D	IW	Ad	Mul	Wb
1	I1,I2				
2	I4,I5				I1,I2
3	I3,I6				I4,I5
4	I8,I9		I3	I6	
5	I7,I10	I8,I9			I3,I6
6	I11	I10		I7	I8,I9
7		I11	I10		I7
8		I11			I10
9				I11	
10					I11

3. (a) Physical addresses directly reference locations in the system's (main) memory. Virtual addresses are the addresses that programs use in their load and store operations. The virtual memory system translates virtual addresses to physical addresses. Some virtual addresses may not have corresponding physical addresses; in this case the referenced data must be brought in from the hard disk.

(b) In a paging system, virtual addresses consist of a page number and an offset inside the page. The virtual memory system translates the page number to a (physical) page frame number by a look up in the TLB or in the page table (in case of a TLB miss). The offset is simply copied (pages and page frames have the same size).

(c) 4 K = 2^{12}, so 12 bits are used for the offset. The reset of the address identifies the page or page frame. Thus there are 2^{36} pages and 2^{24} page frames. The main memory can contain 128 MB/4 KB = 32 K = 2^{15} page frames.

4. (a) Programmed I/O (when the CPU constantly polls the I/O device), interrupt-driven I/O (when the I/O device sends an interrupt to the CPU whenever it is ready to continue with the I/O) and DMA (when there is a dedicated chip to supervise the I/O). Programmed I/O wastes CPU cycles, since the CPU polls the I/O device. The constant polling is avoided by the use of interrupts, but these can happen frequently (depending of the size of the buffer in the controller of the I/O device). With DMA only one interrupt is needed, but cycle stealing can occur when the DMA accesses the I/O device or main memory.

   (b) i. The average seek time is half of the time the head needs to cross all tracks: 50 ms.

   ii. There are 250 revolutions per second, thus one rotation takes 4 ms. The average latency is half of this: 2 ms.

   iii. There are 2048 sectors, so reading one takes 4/2048 ms, which is roughly 2 microseconds.

   iv. We have the seek time (50 ms), the rotational delay (2 ms) and the reading time 1024 $\times$ 2 microseconds, roughly 2 ms. Summing up: 54 ms is the service time.

5. (a) Race condition is when several processes run concurrently and the outcome depends on the order the processes are scheduled. Critical section is that part of the code where shared resources are accessed. Mutual exclusion is the requirement that only one process can be in its critical section at one moment.

   (b) Binary semaphores are usually used to guard critical sections of code. When a process tries to go into the critical section it performs a wait on the semaphore — it becomes blocked if there is another process already in the critical section, otherwise it enters the critical section and updates the semaphore value so that a later process would become blocked trying to enter the critical section. When the process leaves the critical section it preforms a signal on the semaphore and wakes up a process that is blocked on the semaphore so that this process can enter the critical section.

   (c) Deadlock is avoided by making the procedure of taking the forks a critical section: only one process can be blocked on a fork and when the process using that fork finishes eating, the blocked process can advance. Starvation cannot occur either, since the processes can pick up both forks in the order they pass through the critical section guarded by the semaphore `s`.