CS 6810 - Assignment 9

3) (i) LL instruction - When an LL [load-linked] instruction is executed, it reads a value from the memory and updates a special tabb indicating that you have read this address. This helps keep track of the changes done to the address as the table is always notified on some update. It then allows to perform any number of computations.

(ii) SC instruction - When ~~an~~ an SC [store-conditional] instruction is executed, it attempts to store a result into the same memory location accessed by the LL instruction. The store instruction succeeds only if the special table updated during LL indicates that no other processes attempted a store since the local LL. i.e. succeeds only if the operation was effectively atomic.

(iii) Benefit of using LL-SC instead of test-and-test-and-set:
↳ LL-SC is easier to implement in hardware as it is not required to implement atomicity since both the instructions can be executed ~~ad~~ independently.
↳ LL-SC gives programmer more flexibility to execute instructions in between LL & SC as compared to test-and-test-and-set where one has to do read and write simultaneously.
↳ LL-SC gives better performance as it doesn't lead to coherence traffic.

2) High-performance techniques like out-of-order scheduling can yield unintuitive/unexpected outputs in multi-threaded programs. To resolve this issue, we can go for a ~~Rela~~ software-hardware based approach called Relaxed Consistency Model. According to this model, the programmer has to inform the hardware where ~~at~~ should the optimizations in the form of out-of-order execution can take place. The programmer can do so with the help of fence instructions which are special instructions that require all previous memory accesses to complete before proceeding (sequential consistency). Here the programmer has to identify the Racy code and acquire a lock before the execution of the racy code. This will tell the hardware that it can't perform out-of-order execution during the time a lock is being acquired. This means it has to complete all the instructions prior to acquiring lock and can go for OOO executions ~~in~~ between the locks. This solution offers a relatively simple programming & relatively high performance.

1) Sequentially consistent ~~exe~~ execution. $A=B=0$.

Thread 1

$a = A = 20$

$b = B = 40$

Thread 2

$p = \text{if } (B+A) > 30)$

$q = \text{then } B = A+B;$

$r = \text{print } B.$

Possible scenarios. $= {}^5C_2 = 10$

abpqr	apbqr	apqbr	apqrb	paqrb	pqarb
60	40	40	0	0	0
pqrab	pabqr	pqabr	~~pabqr~~ paqbr		
0	40	40	40		

P.1

Thread 1.
lock (L1)
a= A = 20
b= B = 40.
unlock (L1)

Thread 2
lock (L1)
p= if (B+A) > 30
q= then B = A+B;
r= print B
unlock (L1);

c2
$= \frac{2!}{2!0}$
$= {}^2C_1$
$\frac{2!}{1!1!} = 2.$
${}^3C_2$
$= \frac{3!}{2!1}$
$= 3.$

~~ab pqr~~ / ~~bapqr~~ / ~~ab qpr~~ / ~~abprq~~ ~~abrqp.~~

Possible outcomes = $^2P_2 \times {}^3P_3 = 2 \times 6 = 12.$

ab pqr. ba pqr. ab qpr ba qpr ab prq, ba prq
60 60
abrqp barqp abqrp baqrp

~~abrpq~~ ~~barpq~~

Possible outcomes :-
abpqr pqrab.
60 0.

4) Networking
Dimension-order routing - It is an example of deterministic routing where packet is sent along 1st dimension until destination co-ord (in that dimension) is reached, then next dimension etc.
Adaptive Routing - Here a switch may alter the route in order to deal with unexpected ~~events~~ like faults, congestion..
Both these types of routings may lead to deadlocks as the turns can lead to cycles. or there is a cycle of resource dependencies. West-First routing algorithm can eliminate cycles or ~~dep~~ deadlocks by preventing just 2 turns. Hence it is better.