Assignment 3

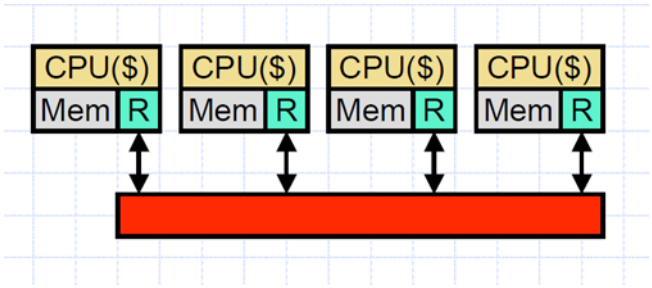
ECSE 420 – Parallel Computing – Fall 2015

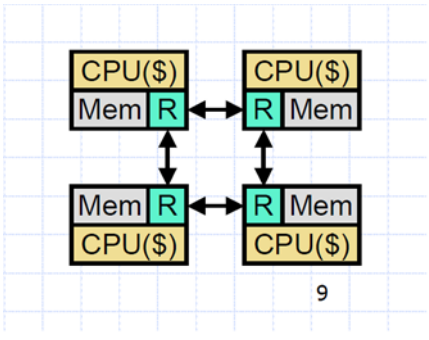
Released: Wednesday, November 4, 2015
Due: Friday, November 13 , 2015 at 11:59 pm

Q1. **Performance Issues & Network Delay** (20%)

Considering a ring network topology of 16 nodes (or processor), with hope latency of 10 ns . If a message communicates through a node in the network to the farthest point, for 80 times, what is the overall network delay? Suggest all possible ways in which you can reduce the network delay.
"Manycore" processors (multicore processors with tens or hundreds of cores) can be arranged in many different topologies. Compare these two topologies by filling the table below (writing +/- as advantages and disadvantages).

	Ring network	Bus network
Cache coherency complexity		
Latency		
Bandwidth		
Wirin cost		





Q2. **Cache memory & bus-based shared memory multiprocessor** (10%)

Consider a bus-based shared memory multiprocessor system with write-through caches. It is constructed using 1.6 GHz processors, and a bus with a peak bandwidth of 50 Mega fetches/s. The caches are designed to support a hit rate of 90%. Only 15 % of program execution time is related to Read and Write commands. Considering each Read/Write instruction, in average, takes 2 clock cycles, What is the maximum number of processors that can be supported by this system?

Q3. **Memory Consistency** (10%)

Suppose there are two process-P1 and process-P2 working on the same counter instance. Explain all possible outcomes from u and v for the following code.

```
int A,B,u,v = 0;

P1() {
[A] = 1 ;  //x1
u = [B] ;  //y1
}

P2() {
[B] = 1 ;  //x2
v = [A] ;  //y2
}
```

Q4. **Illinois Protocol** (35%)

Consider a bus-based shared memory multiprocessor system with 2 Processors, Illinois Protocol and write-back caches ,initially empty, except P1 contains VarB. Both the processors access the shared variables A and B. For the following sequence of commands, write the state of all caches after each executed instruction. Considering the table below and in the last page, indicate bus transaction type and if there is memory access (by writing Yes/No) for each line. Calculate the overall data transfer for whole commands.

Considering MSI protocol, compare the number of Bus Transaction with regard to MESI protocol.

Bus Transaction	Address / Cmd	Data
BusRd	6	64
BusRdX	6	64
BusWB	6	64
BusUpgd	6	--

Ocean Data Cache Frequency Matrix (per 1000)

	NP	I	E	S	M
NP	0	0	1.25	0.96	0.001
I	0.64	0	0	1.87	0.001
E	0.20	0	14.00	0.0	2.24
S	0.42	2.50	0	134.72	2.24
M	2.63	0.00	0	2.30	843.57

Instruction	P0-A	P0-B	P1-A	P1-B	Memory access	Bus transaction
(initially)	I	I	I	E	-	-
P0 reads B						
P1 writes B						
P0 reads A						
P0 write A						
P1 reads A						
P0 reads B						

Table 1: States of caches per executed instruction

Q5. **Test & Set Lock** (25%)

Consider a bus-based shared memory multiprocessor system with 2 Processors. Both the processors execute the same assembly code:

For the given sequence of commands, identify the critical section and provide the needed atomicity. Implement the proper lock-unlock operation using *test&set* instructions:

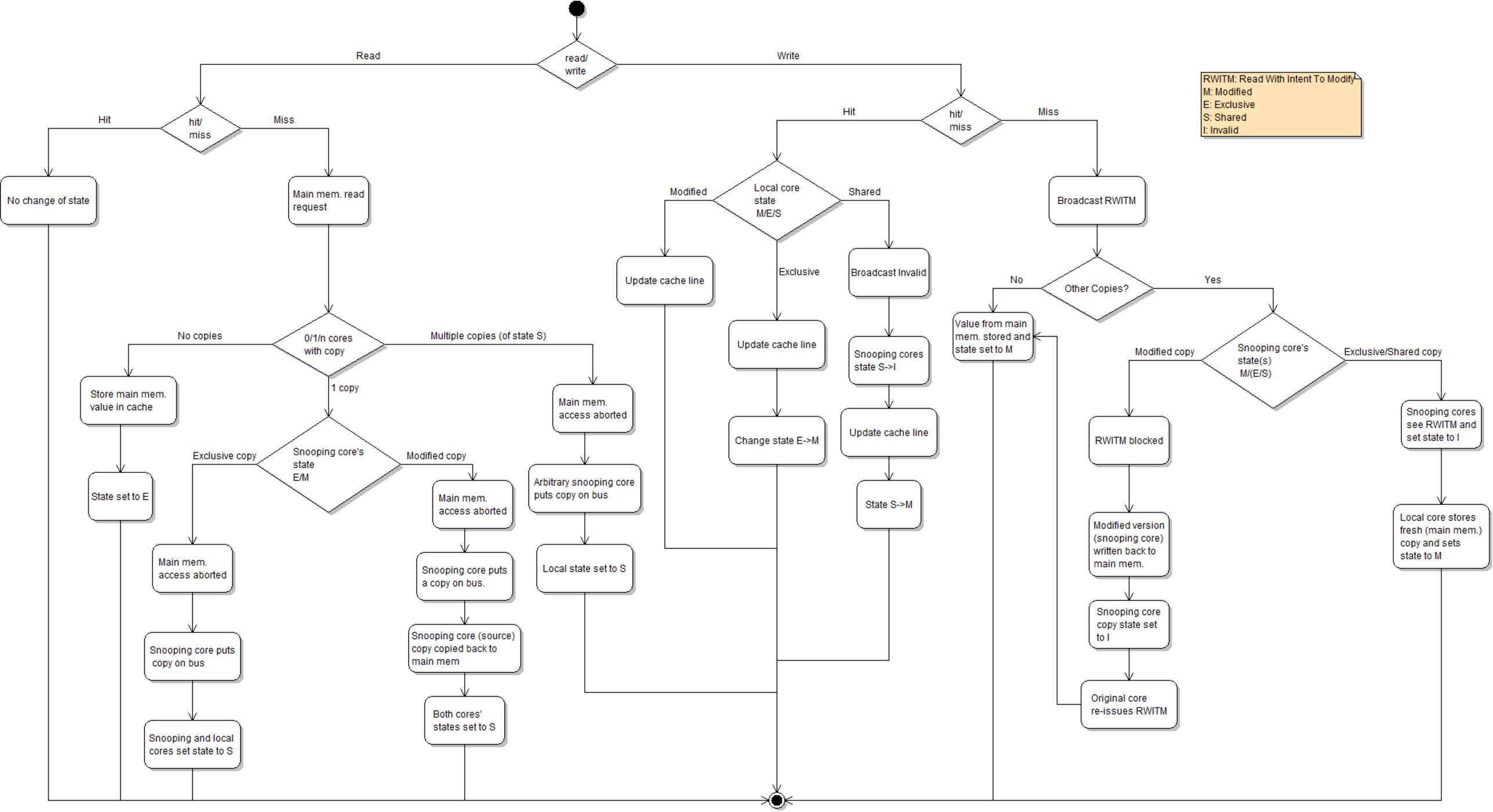
```
lock:    t&s $r0, 0(&lock)
         bnez $r0, lock

unlock:  st $zero, 0(&lock)
```

Assume that both processors have the same $r2 initial values and different $r7 initial values stored in their local register file.

P0		P1	
0:	addi $r1, #5, $r7	0:	addi $r1, #5, $r7
1:	sub $r1, $r1, $r2	1:	sub $r1, $r1, $r2
2:	ld $r5, 2($r2)	2:	ld $r5, 2($r2)
3:	blt $r5, $r3, 8	3:	blt $r5, $r3, 8
4:	sub $r5, $r2, $r1	4:	sub $r5, $r2, $r1
5:	st $r4, 2($r2)	5:	st $r4, 2($r2)
6:	sub $r5, $r6, $r5	6:	sub $r5, $r6, $r5
7:	addi $r6, #4, $r1	7:	addi $r6, #4, $r1
8:	addi $r6, #3, $r1	8:	addi $r6, #3, $r1
9:	addi $r6, #1, $r6	9:	addi $r6, #1, $r6
10:	ld $r4, 0($r1)	10:	ld $r4, 0($r1)
11:	bgt $r4, $r2, 14	11:	bgt $r4, $r2, 14
12:	st $r4, 0($r1)	12:	st $r4, 0($r1)
13:	addi $r5, #2, $r5	13:	addi $r5, #2, $r5
14:	addi $r5, #3, $r5	14:	addi $r5, #3, $r5
15:	...	15:	...

Table 2: Assembly code to be executed


read/ write
Read
Write
hit/ miss
Hit
Miss
No change of state
Main mem. read request
0/1/n cores with copy
No copies
1 copy
Multiple copies (of state S)
Store main mem. value in cache
State set to E
Snooping core's state E/M
Exclusive copy
Modified copy
Main mem. access aborted
Snooping core puts copy on bus
Snooping and local cores set state to S
Main mem. access aborted
Snooping core puts a copy on bus.
Snooping core (source) copy copied back to main mem
Both cores' states set to S
Main mem. access aborted
Arbitrary snooping core puts copy on bus
Local state set to S
hit/ miss
Hit
Miss
Local core state M/E/S
Modified
Exclusive
Shared
Update cache line
Update cache line
Change state E->M
Broadcast Invalid
Snooping cores state S->I
Update cache line
State S->M
Broadcast RWITM
Other Copies?
No
Yes
Value from main mem. stored and state set to M
Snooping core's state(s) M/(E/S)
Modified copy
Exclusive/Shared copy
RWITM blocked
Modified version (snooping core) written back to main mem.
Snooping core copy state set to I
Original core re-issues RWITM
Snooping cores see RWITM and set state to I
Local core stores fresh (main mem.) copy and sets state to M
RWITM: Read With Intent To Modify
M: Modified
E: Exclusive
S: Shared
I: Invalid