No. 937 811E Engineer's Computation Pad

STAEDTLER"

Processor A takes 54 x 2 cycles = 108 cycles

Processor B)

LDRI LDRI LDRI LDRY LDRS CDR6 CDR7 LDRS BEQ LDRI LD LD LD LBEQ 一分的45640 TTTTTT DOUBLE HE HELD LD LD LD 11 16 DADDIU PADDIY DADDIU BLI BLI BLT BLT BCT

If we count no overlap of last BCT for TY with other work, the last loop iteration takes 3 extra cycles. 48+51=99 cycles If overlap, 2 × 48 = 96 cycles.

Process	or C)	
Time 1 2	+ /	LP LD LD LD LD LD LD
234567890	TI	BEQ
890	TI	BEQ
12 m 3 m 4 h	TI	BEQ
3456789	T + 1	DAPDICA
141	(41	x 2) +1 = 83 cycles x9 threads [nest loop iteration = 332 cycles cache miss But, we don't need T4's last LD therefore 332-4=331cycles

```
ELE 475 PS#5 Solution Problem # 2:
```

One of the interesting things about compare and exchange is that atomically operates on a single memory address, but does not have any place to store a lock variable because the locking is implicit. Hence, if you want to build compare and exchange out of test and set, you will need another address. Therefore below, we create a struct which contains both a lock and the value.

```
typedef struct cae_t
   int value;
   int lock; // 1 denotes locked
} cae_t;
// prototype of test_and_set
// old value returned
int test_and_set(int * mem);
// returns 1 if compare success
// returns 0 if compare failure
int compare_and_exchange(cae_t * mem, int compare_value, int swap_value)
{
   int ret_value = 0;
   // grab lock
   while(test_and_set(&(mem->lock)){}
   if(compare_value == mem->value)
   {
      mem->value = swap_value;
      ret_value = 1;
   }
   else
   {
      ret_value = 0;
   }
   // release lock
   mem \rightarrow lock = 0;
}
```

ELE 475 PS#5 Solution

Problem # 3:

We first investigate the final value for 'i'. Because thread T3 only has one memory operation, that memory operation is either ordered before or after the store to 'i' in thread T1, therefore 'i' is either 30 or 100 and these outcomes are valid with all outcomes of 'j'.

Now we investigate valid values for 'j'.

Case 1:

```
T2 LW R6, 0(j) j = 10
T2 SW R7, 0(j) j = 109
T1 LW R2, 0(j) j = 109
T1 SW R2, 0(j) j = 109
```

Case 2:

```
T2 LW R6, 0(j) j = 10
T1 LW R2, 0(j) j = 10
T2 SW R7, 0(j) j = 109
T1 SW R2, 0(j) j = 10
```

Case 2:

```
T2 LW R6, 0(j) j = 10
T1 LW R2, 0(j) j = 10
T1 SW R2, 0(j) j = 10
T2 SW R7, 0(j) j = 109
```

Case 3:

```
T1 LW R2, 0(j) j = 10

T2 LW R6, 0(j) j = 10

T2 SW R7, 0(j) j = 109

T1 SW R2, 0(j) j = 10
```

Case 4:

```
T1 LW R2, 0(j) j = 10
T2 LW R6, 0(j) j = 10
T1 SW R2, 0(j) j = 10
T2 SW R7, 0(j) j = 109
```

Case 5:

```
T1 LW R2, 0(j) j = 10
T1 SW R2, 0(j) j = 10
T2 LW R6, 0(j) j = 10
T2 SW R7, 0(j) j = 109
```

Therefore, valid sequentially consistent outcomes are $\{i, j\} = \{30, 10\}, \{30, 109\}, \{100, 10\}, and \{100, 109\}$

```
ELE 475 PS#5 Solution
Problem # 4:
// Assumes that the input and output arrays are created
// outside of the function. Note that the output lock array is assumed to be
// initialized to 1 (this allows for a mutex)
#include <thread.h>
#include <stdlib.h>
#define MAX VALUE 1023
#define NUM THREADS 100
#define INPUT_ARRAY_ELEMENTS (1024*1024*512) // 512 million entries
struct thread_data{
   int input_array_size;
   int * input_array;
   int * output_array;
   int * output_lock_array;
};
struct thread_data thread_data_array[NUM_THREADS];
void function(int input_array_size, int * input_array, int * output_array,
int * output_lock_array)
   int counter;
   for(counter = 0; counter < input_array_size; counter++)</pre>
      assert(input_array[counter] <= MAX_VALUE);</pre>
      assert(input_array[counter] >= 0);
      P(&output_lock_array[counter]);
      output_array[input_array[counter]]++;
      V(&output_lock_array[counter]);
   }
}
void function_starter(void * thread_args)
{
    struct thread_data * temp_data;
    temp_data = (struct thread_data *) thread_args;
    function(temp_data->input_array_size, temp_data->input_array, temp_data-
>output_array, temp_data->output_lock_array);
    pthread exit();
}
```

```
int main()
{
   int counter;
   pthread t threads[NUM THREADS];
   int * input array;
   input_array = malloc(INPUT_ARRAY_ELEMENTS*sizeof(int));
   // this would be a good place to read a file into input_array
   int * output array = malloc((MAX VALUE + 1) * sizeof(int));
   int * output_lock_array = malloc((MAX_VALUE + 1) * sizeof(int));
   for(counter = 0; counter <= MAX_VALUE; counter++);</pre>
   {
      output_array = 0;
      output_lock_array = 1;
   for(counter = 0; counter < 100; counter++)</pre>
   {
       thread_data_array[counter].input_array_size = INPUT_ARRAY_ELEMENTS /
NUM_THREADS;
       thread_data_array[counter].input_array =
&input array[INPUT ARRAY ELEMENTS / NUM THREADS * counter];
       thread data array[counter].output array = output array;
       thread_data_array[counter].output_lock_array = output_lock_array;
       pthread_create(&threads[counter], NULL, function_starter, (void
*)thread data array[counter]);
   for(counter = 0; counter < 100; counter++)</pre>
      void * dummy;
      pthread_join(threads[counter], &dummy);
   }
}
```

No this program will not see 100x performance speedup for two reasons. First, we require more work to be done in setting and clearing the semaphores. Also, we have used a very fine grain locking scheme which minimized the probability that two threads will use the same location in the output array simultaneously, but it is still possible. When there is contention, a thread will be stalled and performance will decrease. Finally, the coherence protocol along with true and false sharing will cause cache lines to move between the caches of the processors. When a processor is waiting for a cache line to be transferred, these cycles are wasted thereby decreasing performance.

ELE 475 PS5

Problem 5

Solution

MSI Protocol

	P1 Cache		P2 Cache		P3 Cache		Notes
Time	Line Address	State	Line Address	State	Line Address	State	
1	0	Shared	All lines Invalid	d	All lines Invalid	d	
2	0	Shared	0	Shared	All lines Invalid	k	
3	0	Shared	0	Shared	0	Shared	
4	0	Shared	0	Shared	0	Shared	
_	0	Cla a va al	0	Chanad	64	Modified	
5	0	Shared	0	Shared	0	Shared	
6	0	Classical	0	Chanad	64	Modified	
6	0	Shared	0	Shared	0	Shared	
			64	Shared	64	Shared	
7	0	Modified	64	Shared	64	Shared	Line at address 0 is marked invalid in P2 and P3 cache.
8	4096	Shared	64	Shared	64	Shared	Address 4100 (line starts at 4096) aliases with line 0
9	4096	Modified	64	Shared	64	Shared	
10			64	Shared	64	Shared	Line at address 4096 is invalidated in P1 cache
			4096	Modified			
11			64	Shared	64	Shared	Line at address 4096 and address 0 can exist
			4096	Modified	0	Modified	simultaneously in two different caches.

MESI Protocol

	P1 Cache		P2 Cache		P3 Cache		Notes
Time	Line Address	State	Line Address	State	Line Address	State	
1	0	Exclusive	All lines Invalid	d	All lines Invalid	l	
2	0	Shared	0	Shared	All lines Invalid	i	
3	0	Shared	0	Shared	0	Shared	
4	0	Shared	0	Shared	0	Shared	
_					64	Modified	
5	0	Shared	0	Shared	0	Shared	
_	_		_		64	Modified	
6	0	Shared	0	Shared	0	Shared	
			64	Shared	64	Shared	
7	0	Modified	64	Shared	64	Shared	Line at address 0 is marked invalid in P2 and P3 cache.
8	4096	Exclusive	64	Shared	64	Shared	Address 4100 (line starts at 4096) aliases with line 0
9	4096	Modified	64	Shared	64	Shared	
10			64	Shared	64	Shared	Line at address 4096 is invalidated in P1 cache
			4096	Modified			
11			64	Shared	64	Shared	Line at address 4096 and address 0 can exist
			4096	Modified	0	Modified	simultaneously in two different caches.

ELE 475 PS#5 Solution

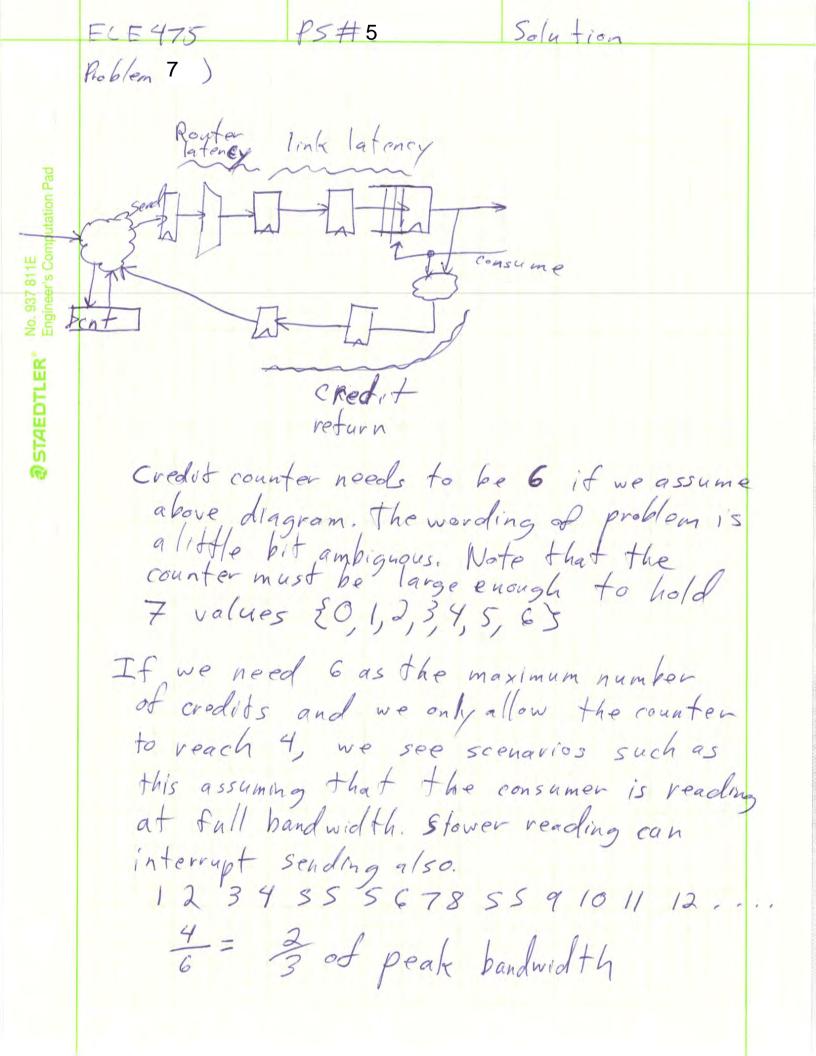
Problem # 6:

We assume that each link is bidirectional even though most mesh networks are made of two sets of unidirectional links. For a 4-ary 3-cube, there are 16 links across the bisection.

16 links * 32bits/link * 800*10^6 cycles/second = 4.096 Tbps.

Each stage of an omega network has the same number of links, but if we examine one stage where a crossover occurs, we see that in a pipelined manner that only 4 links cross the bisection cut of the machine. We assume that each switch is pipelined. Therefore on a given cycle, only 4 links cross the minimum bisection.

4 links * 64bits/link * 1.2*10^9 cycles/second = 307.2Gbps.



Because Store & Forward, no overlap is possible. We assume that there is no router delay at sender side even though this is unrealistic.

therefore, 5 network links where each hop takes 34 cycles yields 170 cycles

Wormhole/cut-through

RIRALLARIRA RIRALALA RIRALA RIRALA

First Flit is recieved at destination after

6 × 5 cycles = 30 cycles

there is 4 cycles for each additional flit (32-bits)

= 28 additional cycles

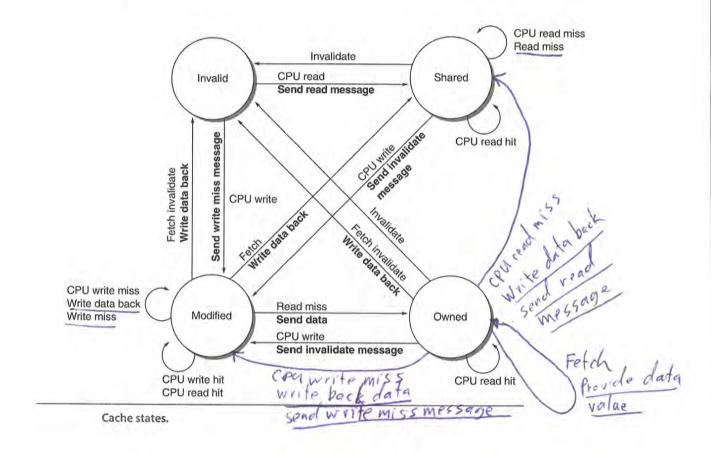
Total = 30 + 28 = 58 cycles

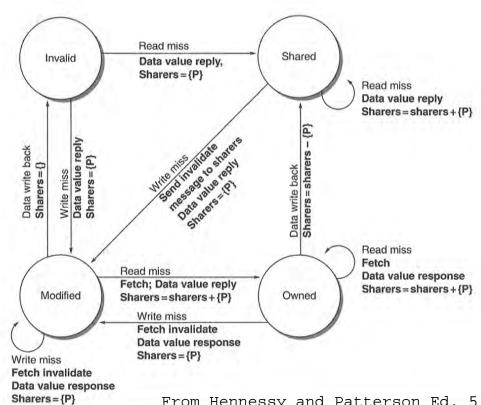
ELE 475 PS5 Problem 9 Solution

ESU Directory Protocol

Note: We assume that writeback of data notifies the directory, but invalidation due to conflict misses on shared data does not.

Time 1	P1 Cache Line Address 0	State Shared	P2 Cache Line Address All lines Invalid		P3 Cache Line Address All lines Invalid		Directory Line Address 0 All other lines Un	State Shared cached	Share List/Owner {P1}	
2	0	Shared	0	Shared	All lines Invalid		0 All other lines Un	Shared cached	{P1, P2}	
3	0	Shared	0	Shared	0	Shared	0 All other lines Un	Shared cached	{P1, P2, P3}	
4	0	Shared	0	Shared	0 64	Shared Modified	0 64 All other lines Un	Shared Exclusive	{P1, P2, P3} {P3}	
5	0	Shared	0	Shared	0 64	Shared Modified	0 64 All other lines Un	Shared Exclusive	{P1, P2, P3} {P3}	
6	0	Shared	0 64	Shared Shared	0 64	Shared Shared	0 64 All other lines Un	Shared Shared	{P1, P2, P3} {P2, P3}	
7	0	Modified	64	Shared	64	Shared	0 64 All other lines Un	Exclusive Shared	{P1} {P2, P3}	
8	4096	Shared	64	Shared	64	Shared	0 64 4096	Invalid Shared Shared	{P2, P3} {P1}	Writeback updates directory
9	4096	Modified	64	Shared	64	Shared	0 64 4096	Invalid Shared Exclusive	{P2, P3} {P1}	all cetery
10			64 4096	Shared Modified	64	Shared	0 64 4096	Invalid Shared Exclusive	{P2, P3} {P2}	
11			64 4096	Shared Modified	64 0	Shared Modified	0 64 4096		{P3} {P2, P3}	





From Hennessy and Patterson Ed. 5 Image Copyright ©

Directory states.

2011, Elsevier Inc. All rights Reserved.