6.823 Computer Architecture DUE: 5/8/2002

Group 18: Timothy Danford, Kilian Pohl, Bryan Russell

Problem Set 6 *Instructors: Krste Asanovic, Srinivas Devadas*

Problem 1

(a) Here's the code to implement `EXCH R2,0(R1)`, where `RT1` and `RT2` are temporary registers:

```
        ADD     RT2,R2,R0
LOOP:   ADD     R2,RT2,R0
        LL      RT1,0(R1)
        SC      0(R1),R2
        BEQZ    R2,LOOP
        ADD     R2,RT1,R0
```

(b)

Only slightly more bandwidth would be needed **per message** when writing to memory with a store conditional instead of a typical store word. Store conditionals, like store word operations, require that the target memory be invalidated in all other caches. However, an additional bit needs to be appended to these invalidate messages, to indicate that the "store" was a "store conditional", and that the link registers of the other processors should be examined and possibly invalidated as well.

There may also be a higher traffic level generated by store conditionals: Ten store words in a row to the same address will only cause (at best) one invalidate message to be generated on the bus to the other caches (the first time). However, every time the store conditional is executed, the other link registers must be invalidated. This means that every store conditional generates an invalidate request along the bus, a higher rate (if not so much size) of bus messages.

(c) Here is the optimized code:

```
TRY:    LL      R2,0(R1)
        BNEZ    R2,TRY
        ADDI    R2,R0,#1
        SC      0(R1),R2
        BEQZ    R2,TRY
```

The major advantage here using `LL/SC` over `EXCH` is that the majority of time we check to see if the mutex is free, we only do a `LL`. The `EXCH` instruction does both a load and a store which adds more traffic to the bus and increases processor cycles. So, with this implementation we save bus cycles and decrease processor cycles.

(d) Suppose that we are executing the lock above in two tasks on the same processor. If both tasks get to the `ADDI` instruction (via context switching) and then both tasks execute the `SC` command then they both will obtain the lock–very bad. The problem here is that one task cannot cancel the other task's R_{link} register. One fix to this is to have multiple R_{link} registers and a task is assigned to one of these registers. Then, when the `SC` instruction is executed in one task all of the inter and intra registers are cancelled. Another option would be to keep a stack that points to where the contexts are stored in memory (and hence points to the R_{link} register.