Lecture 09 (Load Store Queue)

1 Issue with Load Store

```
ld r1, 4[r3]
st r2, 10[r5]
```

1. Can't execute OOO
2. Both may resolve to same address
3. In case of exception, we need to make a clean cut
4. But if store has executed OOO, then memory state is corrupted
5. Loads cannot be directly sent to cache since there might be a pending store

1.1 Resolution Ideas

1. Stores need to wait
2. Need to maintain their information somewhere
3. We use a store queue for this

2 Load Store Queue

1. Allocate an entry at decode time
2. Deallocate later
3. Update entry when address is computed

2.1 Computed Address of Store

Scan entries after this entry (if we encounter a load with same address, then forward) until

1. Store to same address
2. Store with unresolved address

2.2 Computed Address of Load

Scan stores before this entry until

1. Found entry with same address - forward

2. Found entry with unresolved address - wait
3. Reached end of queue - request from memory

2.3 Actual Implementation

Have two separate circular queues for load and store

2.3.1 Load Queue Entry

1. Load address
2. Index of tail pointer in store queue when entry was added

2.3.2 Store Queue Entry

1. Store address and value
2. Index of tail pointer in load queue when entry was added

2.3.3 Basic Search

1. If there are n entries, have a n bit vector
2. $prec(i)$ = all locations before i are set to 1
3. $before(j) = \overline{prec(head)} \wedge prec(j)$ (if no wrap around)
4. $before(j) = \overline{prec(head)} \vee prec(j)$ (if wrap around)
5. $after(j) = \overline{prec(j)} \wedge \overline{map(j)} \wedge (prec(tail) \vee map(tail))$ (no wrap around)
6. $after(j) = (\overline{prec(j)} \vee prec(tail) \vee map(tail)) \wedge \overline{map(j)}$ (no wrap around)
7. To search for resolved entries before j: $before(j) \wedge (match \vee \overline{resvd})$
8. Now to choose the leftmost or rightmost entry, we can use a similar to select logic

We use a tree since it is completely parallelizable. Nix problems exist similar to P vs NP. (size n, poly(n) resources, time taken is poly(log(n)))