Solution

Problem 1: (10 points)

1. A: LRU/temporal locality B: Prefetching/spacial locality

   C: Write-back/temporal locality

2. Use sigpromask(SIG_BLOCK, some_set, some_old_set) to block a signal.

   Use signal(signum, SIG_IGN) to ignore a signal.

Problem 2: (7 points)

1. 012102: impossible

   200212: impossible

   210012: impossible

   201022: possible

2. [1] 5 [2] 2 [3] 7

Problem 3: (9 points)

1. Non-deterministic

   Because if the previous signal is blocked, the following signals will be just thrown away. So we don't know how many signals are handled actually.

2. Call waitpid(-1, null, NOHANG|WUNTRACED) to see if there's some child needed to be reaped.

Problem 4: (8 points)

1. C

   The store buffer is provided so that a series of store operations can be executed without having to wait for each one to update the cache. In A, there's no store operation. In B, there're much more arithmetic instructions than the store operations.

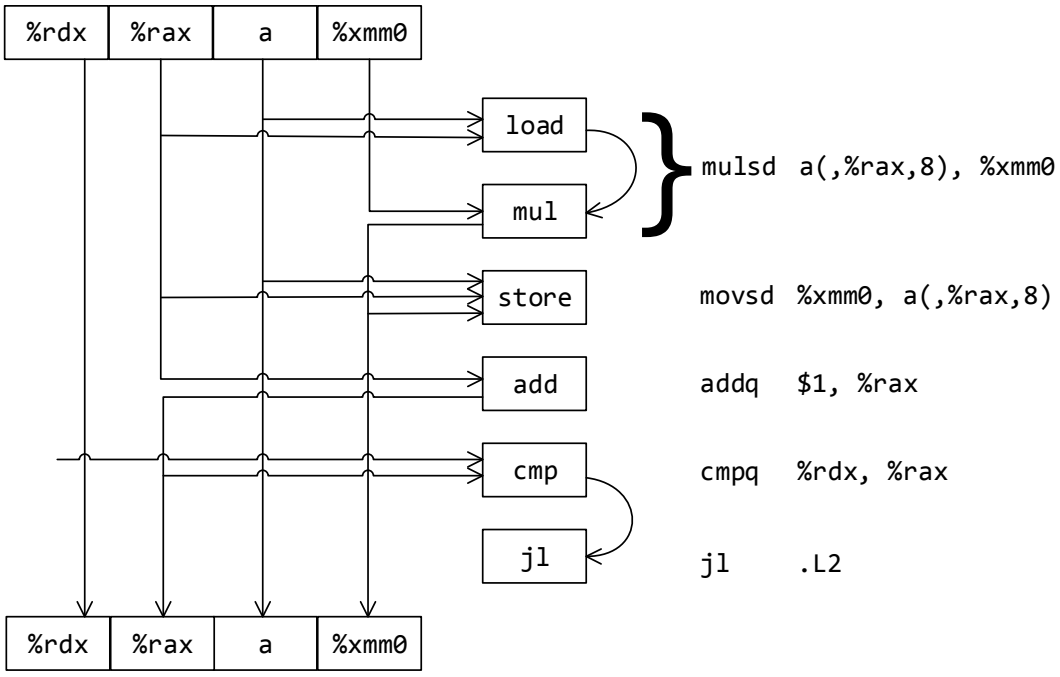
2. C

   The data stored in the store buffer and the data in the FP add unit's pipeline will be cleared if the branch prediction is incorrect.

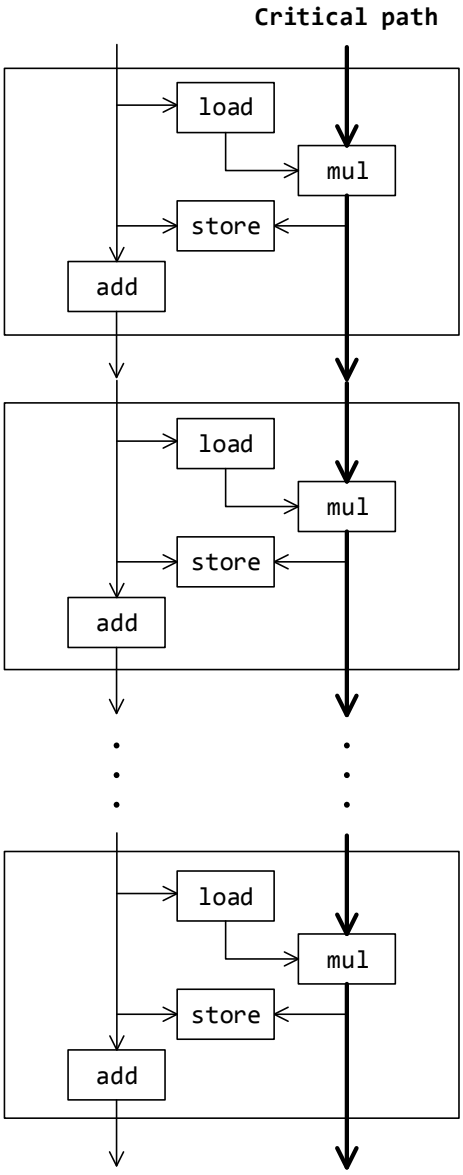
   While the register file will not be affected because it is only updated when the instruction is ensured to execute and the execution has finished.

Problem 5: (17 points)

1. (4')



2. (5')



3. 5 5

4. 3 + 5 + 3 = 11
   There's a dependency between the load operation of one iteration and store operation of the previous iteration. So the load won't start until the previous store finishes.

Problem 6: (28 points)

1. [1] 4 [2] 8 [3] 2 [4]2

2. [1] 0 [2] No [3] 4 [4] No [5] --
   [6] 3 [7] No [8] 7 [9] Yes [10] 0x13
   [11] 0 [12] No [13] 4 [14] No [15] --
   [16] 0 [17] No [18] 4 [19] No [20] --/0x12

3. [1] No [2] No [3] --
   [4] No [5] Yes [6] 0x13
   [7] No [8] No [9] --
   [10] Yes [11] Yes [12] 0x12

4. [1] T [2] T [3] T [4] F

Problem 7: (21 points)

1. [1] 1 [2] 1/80

2. [1] j [2] i [3] k

3. [1] 1/4 [2] 1/4800

4. (9')

```
int func(int* sum){
  int i, j, k;
  int temp_sum = 0; // eliminate unneeded memory references (3')

  for (j = 0; j < Y; j++)
    for (i = 0; i < X; i++)
        for (k = 0; k < Z; k+=4){    // loop unrolling (3')
            temp_sum += A[i][j][k] + B[j];
            temp_sum += A[i][j][k+1] + B[j];
            temp_sum += A[i][j][k+2] + B[j];
            temp_sum += A[i][j][k+3] + B[j];
        }
  // reduce procedure calls (3')
  if (temp_sum < 511037 && temp_sum > -511037)
    for (i=0; temp_sum < 511037; i++)
        temp_sum += i;
  *sum = temp_sum;
  return 1;
}
```