Midterm 2 - Answers

PLEASE MENTION WHO GRADED WHICH QUESTION

Q1 (Anurag)

1a (6 points)

3 possible ways -

a. union
b. memcopy from address
c. cast float pointer to int while dereferencing

1b (3 points)

No it might be a NaN

1c

-0, +0 (or 0x8000… or left shift 1 by 31)

1d (4 points)

f2u: **A** (2 points); other answer (0 point)

u2f: **C,F** (2 points); C (1 point); F (1 point); other answer (0 point)

Note:

For each subquestion, including wrong choice will lead to 0 point.

Not distinguish f2u and u2f: at most 2 points

1e (9 points)

B: invalid, go to the wrong direction of stack layout, visit unknown memory

D: float D(unsigned x){return x;}

E: unsigned E(float t) {return t;}

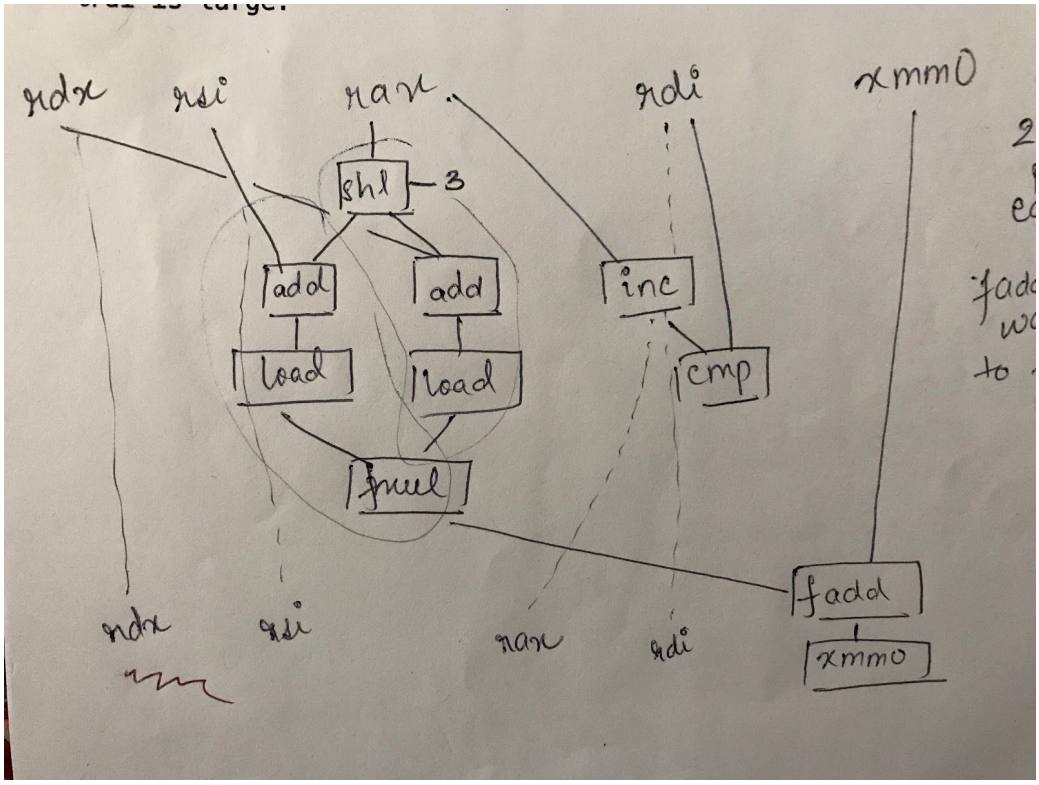
Note:

Each of above answer is worthy 3 points.

Minor error: -1 point/each part.

If you include wrong answer, you will get at most 8 points.

Q2a (11 points) Aanchal Dalmia



This is one of the possible data-flow representations. There can be other solutions as well. The critical paths are circled in the image. It involves the floating point addition and multiplication operations. It is a critical path since fadd has to wait for fmul to finish.

Q2b Mingda Li

The program can be parallelized. You can answer one technique to the instruction-level parallelism:

E.g

1. Parallelism among iterations
2. In one iteration, the two additions and loadings before multiplying them can be parallelized
3. The inc can be parallelized with the mul instruction.

Any technique can win the whole credit. If you only answer the program can be parallelized but no technique, you will only get 3 points.

(Nikita)

Q.3

Multiplexing: Good for activities with delays. For example, applications which need to wait for user inputs.

Process Parallelism: Good for loosely coupled systems. Different parts run independently.

Thread Parallelism: Good for systems where different parts share information like in web servers.

(For process and thread they are expected to give any logical example with explanation of why process or thread suits it better)

SIMD: Single instruction multiple data. Works best for operations on arrays, matrices where same instruction has to be performed on all data. ML is an application.

Instruction Level Parallelism: Can be used in most cases for optimisation where there are less data dependencies and not too many branches. (This is useful whenever the other four are not.)

Note: There are many possible correct answers for this question. Marks s given wherever the student has given examples correctly and explained why the example is best for that type of parallelism.

(Shikhar)
Q.4
Pros for exclusive: More capacity, hence more data can be cached; Less redundancy of data; Holds one copy for each data value
Cons for exclusive: More management to keep track of uniqueness; Different cache lines are difficult to support; Have to search L1 and L2 to get hold of specific data
Pros for inclusive: Simpler in design; Different cache line sizes are supported (like 16 for L1 and 64 for L2, resulting in faster access); Faster data access
Cons for inclusive: More time to cache; Cache may get filled quickly with data being replicated; Need to maintain correct state of data in both caches

Points mentioned above are not exhaustive. Marks have been given for other **logical** points too. Sensible 1 pro and 1 con each for exclusive and inclusive fetches you 8 marks.

(Harshada)

Q.5 a. The callee saved register RBX is being stomped on. %rbx value being altered.
Q.5 b. For Stack Alignment. Need memory for PUTS function. Partial credit if any of the 2 points missing.
Q.5 c. Call pthread_join with the appropriate arguments. Possible solution to resolve the bug would be having an array of thread IDs, initialized by the loop that calls pthread_create, and then used by a later loop that calls pthread_join. Partial marks for stating that the output string is being printed in random (maybe even interleaved) order (race condition). Partial credit for stating that threads are not being cancelled or no exit or not detatched.

(Swathi)
Q5 d.

```c
void *f(const char *s)
{
    puts(s);
}
int main(int argc, char *argv)
{
    pthread *x;
    char **p = &argv[1];
    while(*p)
    {
        pthread_create(0,&x,f,*p++);
    }
    return 0;
}
```

Looking for the presence of puts() function, the use of argv[1] and the right call for pthread_create() within a loop.