UCLA Computer Science 33 (Spring 2017)
Midterm 2, 100 minutes, 100 points, open book, open notes.
Put your answers on the exam, and put your name and student ID at the top of each page.

Name:_____ Student ID:_____

1 a-c	1 d-e	2 a	2 b	2 c	3	4	5 a-c	5 d	total

1a (6 minutes). Write two C functions with the following APIs:

```
unsigned f2u (float x);
float u2f (unsigned y);
```

f2u(X) should yield an unsigned integer Y that has the same 32-bit representation as X. For example, f2u(-0.1f) should yield 0xbdcccccd, because -0.1f has a sign bit 1, an exponent field 0x7b, and a fraction field 0x4ccccd, and ((1u<<31) | (0x7b<<23) | 0x4ccccd) == 0xbdcccccd. The u2f function should be the reverse operation, i.e., f2u(u2f(Y)) == Y should be true for all unsigned values Y.

```
unsigned f2u (float x){
    union {
        float f;
        unsigned un;
    } u;
    u.f = x;
    return u.un;
}
```

```
float u2f (unsigned y){
    union {
        float f;
        unsigned un;
    } u;
    u.un = y;
    return u.f;
}
```

+6

1b (3 minutes). Does the C expression u2f(f2u(X)) == X yield 1 for all float values X? If so, briefly justify why; if not, give a counterexample.

No. If X is a NaN, then u2f(f2u(x)) returns a NaN, but NaNs are never equal.

+3

1c (3 minutes). Give two unsigned values Y1 and Y2 such that the C expression (Y1 != Y2 && u2f(Y1) == u2f(Y2)) yields 1.

~~Two unsigned values are y1 = 0x7F800001 and y2 = 0x7F800002~~
~~Since both these values have a representation of NaN in float representation~~

0

9

Name:_____ Student ID:_____

1d (4 minutes). Which of the following machine-language functions, if any, are plausible x86-64 implementations of f2u and of u2f, respectively?

u2f

A:
```
movl  %edi, -4(%rsp)
movss -4(%rsp), %xmm0
ret
```

B:
```
movl  %edi, 4(%rsp)
movss 4(%rsp), %xmm0
ret
```

C:
```
movss %xmm0, -4(%rsp)
movl  -4(%rsp), %eax
ret
```

f2u

D:
```
pxor      %xmm0, %xmm0
movl      %edi, %edi
cvtsi2ssq %rdi, %xmm0
ret
```

E:
```
cvttss2siq %xmm0, %rax
ret
```

F:
```
movd %xmm0, %eax
ret
```

f2u

u2f: WA

1e (9 minutes). For each machine-language function (A)-(F) that is not a valid implementation of either u2f or f2u, explain why not. If there is some other C-language function that the machine-language function is a valid implementation of, give such a function; if not, explain why not.

B does not work because it could be overwriting important information on the stack, such as the return address of the caller. This is because the stack grows in a negative direction. Because of this, it is not a valid implementation of any C function. conflict with choice 1d, at least one should lose points

D is not a valid representation because it does not preserve the bits as they are because of the cvtsi2ssq instruction. This would however be valid for a function that converts ints into float representation.

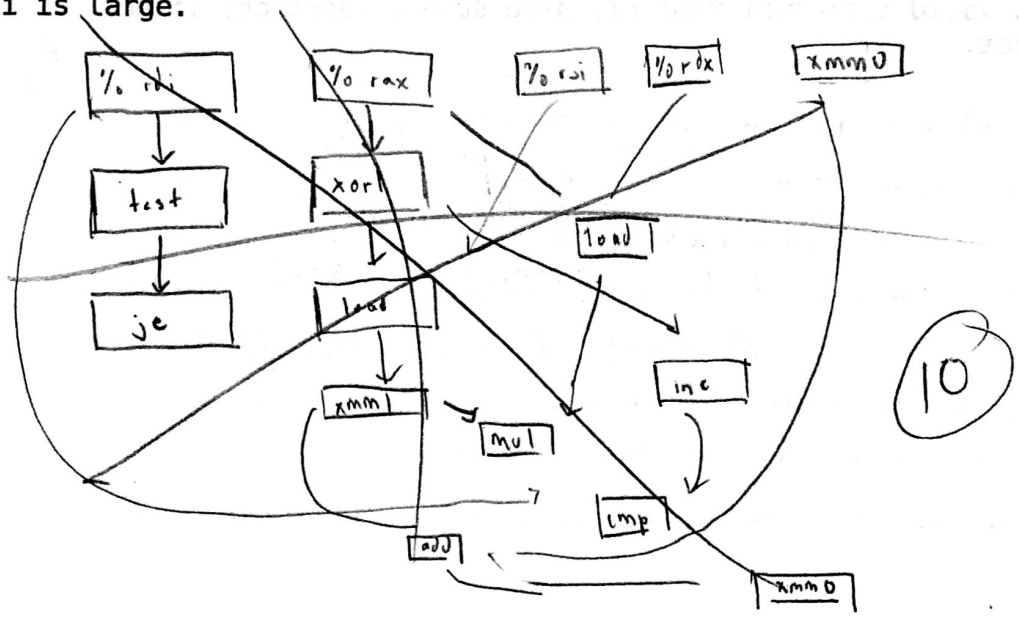
E is not a valid representation because cvttss2siq instruction does not preserve the bits as they are. However, this function would work for converting floats into their integer representations

Name:_____ Student ID:_____

Consider the following x86-64 function foo:

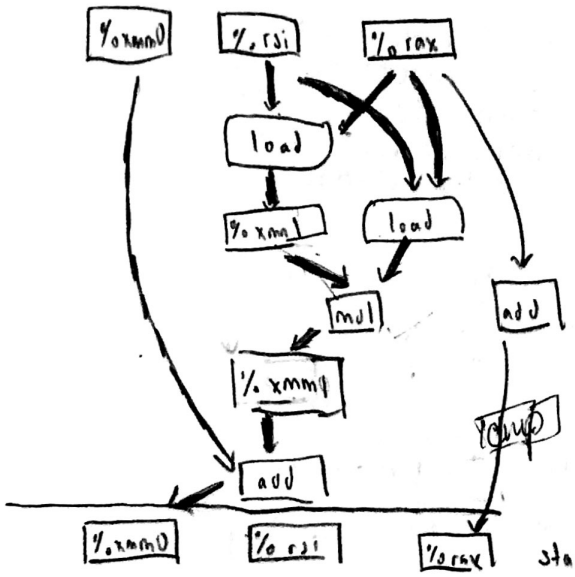
```
  foo:
          testq %rdi, %rdi
          je    .L4
          xorpd %xmm0, %xmm0
          xorl  %eax, %eax
  .L3:
          movsd (%rsi,%rax,8), %xmm1
          mulsd (%rdx,%rax,8), %xmm1
          incq  %rax
          cmpq  %rax, %rdi
          addsd %xmm1, %xmm0
          jne   .L3
          ret
  .L4:
          xorpd %xmm0, %xmm0
          ret
```

2a (11 minutes). Construct a data-flow representation of the micro-operations for the inner loop of the function foo. Identify any critical paths that are likely to be a performance bottleneck when %rdi is large.



10

Answer on back



(I apologize, this chart is a little hard to read!)

start of next iteration

The critical path is shown in bold arrows. This must be the critical path because %xmm1 cannot be used for the next iteration until its results are added into %xmm0. However, this path requires two seperate load operations and a multiply operation, which is much greater than the add operation to rax. So the center path, with 2 loads, a mul, and an add is likely to bottle neck the program in the case of a large %rsi.

Name:_____ Student ID:_____

2b (11 minutes). Suppose the function foo is executing on a Kaby Lake processor. Kaby Lake processors have the same set of functional units as the Haswell. How well will the inner loop be parallelized on this processor, using instruction-level paralellism? Briefly justify your answer by appealing to your answer to the previous subquestion.

The inner loop should be parallelised fairly well. This is because there are a large amount of multiplication and addition, and load operations, but the Kaby Lake has at least two functional units for each one of those operations. Thus the inner loop should be successfully pipelined, with each set of micro operations staggered as to keep all functional units busy at a time. We can see from the critical path of the previous question that it contains two loads, an add and a multiply. The Kaby Lake has enough functional units to successfully pipeline this instruction.

8 11

(292)

Name:_____ Student ID:_____

2c (11 minutes). Suppose you have several chips that implement the same x86-64 instruction set. Each chip has just one cache for RAM. The cheapest chip uses a direct-mapped cache; the next-cheapest one uses a 2-way set-associative cache; the next-cheapest one a 4-way set-associative cache, and so on for 8-way and 16-way. All the caches use a 64-byte cache line and they all contain 1 MiB of data. If you are interested in executing the function foo on many small arrays (with typical size 256 bytes), which of these chips will you be most interested in buying, if you want to maximize the bang for your buck? Briefly justify your answer.

2

$256/64 = 4$

$1\ \text{MiB} = 2^{20}$ Bytes, Block size $= 2^6$

Direct mapped $2^{20}/2^7 = 2^{13}$ sets

2 way $2^{20}/2^8 = 2^{12}$ sets

4 way $2^{20}/2^n = 2^{11}$ sets

2-way for foo

To maximize bang for your buck, you should buy the 4-way set-associative cache. With the 4 way set associative cache, each set can hold up to 4 address that have 64 bytes after the initial address. Since the typical array size is 256 bytes, a single array can be split into four addresses, so each array will belong to a single set. In 1 or 2 way caches, the set would be full and would need to evict data, but since the arrays are being referenced frequently, this causes thrashing, & and 16 way associations mean more lines can map to a set, meaning it takes longer to search through the set to find the desired address, and arrays are only 256 bytes anyway. Since lines are 64 bytes in size, only four lines are required to store an entire array, meaning the 4-way cache is the optimal choice

Name:_____ Student ID:_____

3 (11 minutes). For each of the following forms of machine parallelism, give an example of an application that will likely work better with this form of parallelism than with any of the other forms listed. Briefly justify your answers.

instruction level parallelism
multiplexing
process parallelism
SIMD
thread parallelism

11

Instruction level parallelism works best with programs that want to take advantage of parallelism without having to rewrite the entire code to use the vector registers required by SIMD. An example is highly object oriented code that must be both reusable and is difficult to translate into SIMD code.

Multiplexing gives more control over program behavior than process parallelism, and has the access to the entire address space of a process making it easy to share data. An example application is an event-driven server that might want to give some preference to clients.

Process parallelism is ideal for programs because it can eliminate the risk of a thread stepping on another threads memory by making processes isolated. An example program might be a simple web server without worrying about prefthread allocs or programs on an operating system.

SIMD is good for complicated programs that need to ~~not a~~ perform the same operation on multiple data at the same time. As such, SIMD is prefthred in graphics applications.

Thread parallelism is a compromise between processes and multiplexing that allow shared address space but must worry about the actions of other parallel threads. An example program is a concurrent echo server, as opposed to an event driven server.

Name:_____ Student ID:_____ (9)

4 (10 minutes). Suppose you are designing a computer with two caches: a smaller L1 cache (one per core), and a larger L2 cache (shared among all the cores). You are trying to decide whether the caches should be *exclusive*, i.e., a data word is never in both L1 and L2, or *strictly inclusive*, i.e., every data word cached in L1 is also cached in L2. Give pros and cons of both approaches.

Exclusive (This assumes a write-back approach)

Pros

More storage available in L2

Evicted words need to be updated in L2 anyway, so it is redundant to have an old copy remaining in L1

A miss in L1 will go to L2 anyway, so it is redundant to store missed words in both L1 and L2

Cons

Latency increased, as there is much more data transfer between L1 and L2

On miss, data is transferred out of L1 and destroyed in L2, instead of just being transferred.

If a word is evicted out of L1, but then brought into L1 again, first the word is destroyed in L1 and placed in L2, and then it is destroyed in L2 and placed back into L1, where it would be much more efficient to just leave the word.

If L1 cache fails, word is lost forever.

Inclusive

Pros

Much less writing/destroying than in L1, so latency decreased and throughput increased

~~Faster [illegible]~~

If for some reason L1 cache fails, and data is lost, data can just be copied out of L2

Cons

Much more space required, L2 must hold data in L1

Dirty bit required to see if L2 has been properly updated, and information changed in L1 is not correct in L2 anyway, so it is redundant to have both