Final

20% of midterm material, all as multiple choice

Given binary sequence what is its decimal value?

Given signed negative number, what is its binary representation?

Pointer arithmetic (let's not forget strings are arrays of char)

Correct/incorrect instructions for x86 64 bit (remember size of registers and instruction sizes, postfixes)

20% Cache

Concepts

Cache friendly code

Given a 64 bit single core machine, CPU has 256 2-way local cache and cache line is 64 bytes, how many sets? (2-way: each set has 2 cache lines)

2 sets (each set is 128 bytes)

Assume cache line is indexed and tagged by *physical address*. What are the bit positions in address used to represent the offset of the cache line, index set in the cache?

Using the address: 64 bit address = 8 bytes = 2ˆ64 possible ways

64 bytes cache line = 2ˆ6 bytes, that means we need 6 bits for the indexing of the offset

1 more bit for the set (2ˆ1 for set)

If each (both physical and virtual) page size is 4KB, can we use the virtual address to index the cache cache set? Why? If the answer is "yes", is there any advantage?

Yes, you can. with 4KB pages, last 7 bits of virtual address are exactly the same as the physical address 2ˆ10 for page offset, and 2ˆ10 offset (4k bits = 1k bytes) can be used for the cache offset (10 > 6 needed bits).

Given a struct:

```
typdef struct {
char (1)
unsigned long (8)
char (1)
unsigned long (8)
char (1)
unsigned long (8)
} node

node n[6];

for (int i = 0; i < 6; i++) {
  access(n[i]);
}
```

What is the size of the node? 48 bytes

Assume the address of n[0] is cache line alignment, how many cache misses does the program have? (same cache as previously) ceil(48 * 6 / 64) = 5 misses, since 64 byte cache line size, 6 accesses of 48 bytes

How do we reduce the cache misses? Restructure it so the chars are together and the longs are together (32 bytes)

ceil(32 * 6 / 64) = 3 misses

Customize this question: size of cache, cache line ~~32 bit machine~~ Page size Structure layout, array length

Virtual Memory

Given a 16 bit machine, both virtual and physical are 16 bit, uses 2 level page table for address translation and page size is 64 bytes.

For each page in the page table how many entries does it have? Each entry is the physical address of the page in the next level. page is 64 bytes: 64 / 2 = 32 entries each page (16 bit address = 2 bytes)

Page table is made up of pointers in the pages.

How to divide the address to index the page tables? 2ˆ5 = 32 5 bits for offset (6 bits to index byte in page) 2ˆ6 = 64 5 | 5 | 6

Given the virtual address 0x82, what is the virtual page number? How do you get the physical address with the 2 level page table?

1. Get the root page physical address. (Register %CR3)
2. Leftmost 5 bites, calculate offset of the entry. (0)
3. With the physical address in the first entry, we find the page in the in the second level.
4. Then offset of next level: (2)

If the cache line size in the CPU cache is 64 bytes, in the worst scenario, how many cache misses will you get to get the physical

address and how many cache misses will you have to access data? 2 misses, 3

Customize: virtual address, physical address Page size, 3 level page table, cache line, best scenario Note: if there is a difference between virtual address length and physical address length, we use the physical address length to get page table size.

Multithreading

```
void transfer(int x, int y, int amt) {
  if (account[x]->val > amt) {
    accounts[x]->val -= amt;
    accounts[y]->val += amt;
  }
}

int sum(int x, int y) {
  return accounts[x]->val + accounts[y]->val;
}
```

Requirement: sum should not observe middle state of transfer.

If two threads concurrently invoke transfer/sum, what interleaving can break the requirement? Trivial

Use a fine-grained read-write lock to protect the code.

Write code the implement the read write code that prefers write over read.

Add the fairness to the readwrite lock, in order acquired. Queue based lock with read write lock implementation.