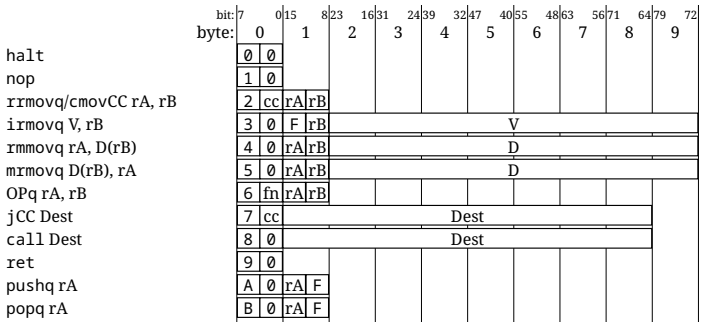
Spring 2022 CS 3330 Final

Name: _____

Computing ID: _____

- Time limit: 180 minutes.
- No notes, books, or calculators. Scratch paper is allowed.
- Because the test is given in multiple locations, we cannot answer questions about the test during the test. If you find a question ambiguous or confusing, explain that on the page near the question and it will be considered during grading.
- Grading occurs in gradescope, which shows one question to the grader at a time. Keep your comments and answers near the question they apply to.
- You may use pencil or pen, and may mark answers in any way a human grader can easily understand.
- Please write clearly. Ambiguous answers, such as `T` when `T` or `F` was expected, are graded as wrong.
- Partial credit is awarded for answers that demonstrate partial understanding.
- Grading is not linear: earning 50% of the available points may result in more than a 50% grade on the test.
- Unless otherwise specified, questions that refer to pipelines assume the 5-stage pipeline from the textbook and your homeworks, namely:
    1. Fetch (with one instruction access)
    2. Decode (with two register reads)
    3. Execute (with one ALU operation)
    4. Memory (with one memory access)
    5. Writeback (with two register updates)
- Some select-all-that-apply questions may have a correct answer of none or all
- For your reference:
    - 2^{10} is abbreviated "K" or "Ki" (kilo)
    - 2^{20} is abbreviated "M" or "Mi" (mega)
    - 2^{30} is abbreviated "G" or "Gi" (giga)
    - 2^{40} is abbreviated "T" or "Ti" (tera)
    - 2^{50} is abbreviated "P" or "Pi" (peta)
    - 2^{60} is abbreviated "E" or "Ei" (exa)
    - byte or octet is abbreviated "B"
    - a "cold cache" is one with no data (i.e. all lines are invalid)
    - in assembly,
        - the order is `operation, source, destination`
        - `123(%rax, %rbx, 8)` means the memory address `123 + rax + (8 * rbx)`
        - `cmp X,Y` sets the flags as if comparing the result of `Y -= X` to 0
        - `jg` and `jl` do signed-integer comparisons; `ja` and `jb` do unsigned-integer comparisons

The Y86 instructions and their encoding as machine code are:



We call the nyble that contains a number identifying the instruction above "`icode`" and the the nyble named `fn` and `CC` above "`ifun`".

Pledge:
On my honor, I pledge that I have neither given nor received help on this assignment.

Signature: _____

There are 17 questions with 30 parts worth 81 total points.

Question 1 (2 pt)
Write a C expression that reverses the bytes of a 16-bit `unsigned short x`. For example, given `x = 0x1234` it should produce `0x3412`

Answer: _____

Question 2 (2 pt)
Consider the following assembly snippet, which uses the AT&T-syntax x86-86 assembly notation discussed in class and the textbook.

```
loop:
    addq    $1, %r14
    movq    %rbx, %rdi
    callq   f
    addq    $2, %rbx
    testl   %eax, %eax
    jne loop
```

Which of the following C snippets could create this assembly?
Pick One

- ○ `for(int i=0; i != 0; i = f(s)) { t += 1; s += 2; }`
- ○ `for(int i=0, j=0; f(j) & f(j); i++, j+=2) {}`
- ○ `while(f(s)) { t += 1; s += 2; }`
- ○ `while(f(s+2) != 0) { t += 1; }`

Question 3 (8 pt)
Consider a set-associative cache with

- 16-bit addresses
- 256-byte blocks
- 16 sets
- 2 lines per set, with LRU replacement
- write-back policy (also write-allocate)

Suppose the following memory accesses happen in the order listed, starting with a cold cache (all lines invalid). For each, write its set-index, tag, if it's a hit, if it evicts a valid line, and if this access causes a write to be sent up the cache hierarchy towards RAM. All numbers are in hexadecimal.

R/W	Address	index	tag	hit	evict	write to RAM
r	`3330`	_____	_____	☐	☐	☐
W	`3340`	_____	_____	☐	☐	☐
r	`3430`	_____	_____	☐	☐	☐
W	`4330`	_____	_____	☐	☐	☐
r	`5330`	_____	_____	☐	☐	☐
W	`3530`	_____	_____	☐	☐	☐
r	`3330`	_____	_____	☐	☐	☐

Question 4
Consider a system with two-level cache hierarchy where:

- a hit on the L1 cache takes 1 cycle
- a hit on the L2 cache takes 10 cycles
- accessing main memory takes 100 cycles

Suppose a read-only workload is measured to have

- 80% of L1 accesses hit
- 75% of L2 accesses hit

4.a. (4 pt) What percentage of first-level cache accesses require a main memory access? Write your answer to the nearest whole number of percent.

Answer: _____

4.b. (4 pt) Assume a miss on a cache takes as long as hit, plus however long it takes to access the next level of the memory hierarchy. For example, this would mean that an L1 miss but L2 hit would take 11 cycles to complete.

Given this, what is the average memory access time of the program, in cycles, rounded to the nearest tenth of a cycle?

Answer: _____

Question 5 (2 pt)
Which of the following could have a higher hit rate on a **direct-mapped** cache than on a fully-associative cache of the same size? Assume the arrays do not overlap and are large enough that the accesses do not go out of bounds, that all arrays store the same element type, and that `sizeof(array[i])` is a power of 2.
Pick One

- ○ `for(int i=0; i<32; i+=1) use(array[i]);`
- ○ `for(int i=0; i<37; i+=1) use(array[i]);`
- ○
```
for(int i=0; i<8; i+=1) {
    use(array1[i]);
    use(array2[i]);
    use(array3[i]);
    use(array4[i]);
}
```
- ○
```
for(int i=0; i<8; i+=1) use(array1[i]);
for(int i=0; i<8; i+=1) {
    use(array2[i]);
    use(array3[i]);
    use(array4[i]);
}
for(int i=0; i<8; i+=1) use(array1[i]);
```

Question 6
Consider the following four functions, all of which compute the same result

```
long foo1(long *a, long *b, int N) {
    long ans = 0;
    for(int i=0; i<N; i+=1)
        for(int j=0; j<N; j+=1)
            ans += a[i*N + j] * b[j*N + i];
    return ans;
}

long foo2(long *a, long *b, int N) {
    long ans = 0;
    for(int j=0; j<N; j+=1)
        for(int i=0; i<N; i+=1)
            ans += a[i*N + j] * b[j*N + i];
    return ans;
}

long foo3(long *a, long *b, int N) {
    long ans = 0;
    for(int k=0; k<N; k+=4)
        for(int i=0; i<N; i+=1)
            for(int j=k; j<k+4; j+=1)
                ans += a[i*N + j] * b[j*N + i];
    return ans;
}

long foo4(long *a, long *b, int N) {
    long ans = 0;
    for(int k=0; k<N; k+=4)
        for(int j=0; j<N; j+=1)
            for(int i=k; i<k+4; i+=1)
                ans += a[i*N + j] * b[j*N + i];
    return ans;
}
```

For the purposes of this question,

- **data temporal locality** is measured only on the exact same address (`a[0]` is only temporally local to `a[0]`, not `a[1]`)
- **data spatial locality** is measured only on nearby indices in the same array (ignoring fetches and the possibility that argument arrays might be aliases or otherwise adjacent)

6.a. (2 pt) Which function has the best data temporal locality on `a`? If several are tied, select that entire set.
Select all that apply

- [ ] `foo1`
- [ ] `foo2`
- [ ] `foo3`
- [ ] `foo4`

6.b. (2 pt) Which function has the best data spatial locality on `a`? If several are tied, select that entire set.
Select all that apply

- [ ] `foo1`
- [ ] `foo2`
- [ ] `foo3`
- [ ] `foo4`

6.c. (2 pt) Which function has the best overall data cache performance and thus the fastest runtime? If several are tied, select that entire set.
Select all that apply

- [ ] `foo1`
- [ ] `foo2`
- [ ] `foo3`
- [ ] `foo4`

Question 7 (2 pt)
A 3-level virtual memory system has 32-bit virtual addresses and has each array of PTEs in the page table hierarchy exactly fills one page. What is the smallest (positive) **size (in bytes) of a PTE** that can make this possible?

Answer: _____

Question 8 (2 pt)
A 1MB cache has 64-byte blocks and 4-way set associativity. How many **bits is its set index**?

Answer: _____

Question 9

Consider a virtual memory system where

- Addresses are 36-bits, made up of
  - a 24-bit VPN (broken into three 8-bit parts)
  - a 12-bit page offset
- There is a two-way set-associative TLB

9.a. (4 pt) Suppose I write a program that uses just three pages of data:

- Code, on page with VPN `0x000800`
- Globals, on page with VPN `0x001000`
- Stack, on page with VPN `0x7FFFFF`

Ignoring kernel-mode memory, but counting pages used to store arrays from the page table, how many total physical pages are used by this program?

Answer as a base-10 integer, like `0` or `256`

Answer: _____

9.b. (2 pt) How big are page table entries in this system? Answer as an integer number of bytes

Answer: _____

9.c. (3 pt) The virtual address space for this system is 2^{36}B = 64GB. The physical address space could be

Select all that apply

- ☐ smaller than 64GB
- ☐ equal to 64GB
- ☐ larger than 64GB

9.d. (4 pt) Suppose a program accesses three addresses repeatedly in a loop. Which set of addresses will cause non-cold misses in the TLB?

Select all that apply

- ☐ `0x112233444`, `0x112233445`, `0x112233446`
- ☐ `0x112234444`, `0x112235444`, `0x112236444`
- ☐ `0x112433444`, `0x112533444`, `0x112633444`
- ☐ `0x142233444`, `0x152233444`, `0x162233444`

Question 10

The following ask about the potential impact of various optimizations that can be applied at different levels. Assume that the code being optimized benefits from the optimization chosen and that any necessary hardware support is available.

10.a. (2 pt) Using *n* accumulators

Pick One

- ○ has linear speed benefits as *n* increases
- ○ has linear speed benefits as *n* increases as long as *n* is small enough not to cause register spilling
- ○ has more impact the larger *n* is, but with diminishing benefit as *n* increases
- ○ has more impact the larger *n* is as long as *n* is small enough not to cause register spilling, but with diminishing benefit as *n* increases
- ○ only has any impact if the loop body is small, and then only for small *n*

10.b. (2 pt) Unrolling the innermost loop *n* times

Pick One

- ○ has linear speed benefits as *n* increases
- ○ has linear speed benefits as *n* increases as long as *n* is small enough not to cause register spilling
- ○ has more impact the larger *n* is, but with diminishing benefit as *n* increases
- ○ has more impact the larger *n* is as long as *n* is small enough not to cause register spilling, but with diminishing benefit as *n* increases
- ○ only has any impact if the loop body is small, and then only for small *n*

10.c. (2 pt) Using vectorized operations with *n*-element vectors

Pick One

- ○ has linear speed benefits as *n* increases
- ○ has linear speed benefits as *n* increases as long as *n* is small enough not to cause register spilling
- ○ has more impact the larger *n* is, but with diminishing benefit as *n* increases
- ○ has more impact the larger *n* is as long as *n* is small enough not to cause register spilling, but with diminishing benefit as *n* increases
- ○ only has any impact if the loop body is small, and then only for small *n*

Question 11 (2 pt)
Amdahl's law most directly motivates which of the following optimization practices?
Pick One

- ○ measuring cache locality with valgrind cachegrind
- ○ measuring instruction sequence latency with a CPU emulator
- ○ measuring what part of code uses the most runtime with a profiler

Question 12 (4 pt)
Sometimes one optimization reduces the impact of another not because they are in conflict, but rather because they are both trying to reduce the same inefficiency so that success on one means there is less left for the other to remove.

Select two of the following optimizations that target the same inefficiency and thus reduce one another's potential for speedups

- ☐ cache blocking
- ☐ loop from `N-1` to `0` instead of `0` to `N-1` to remove one `cmp` per loop
- ☐ loop unrolling
- ☐ moving work outside of loops
- ☐ multiple accumulators
- ☐ reducing aliasing
- ☐ using local variables instead of memory

Question 13
In the following diagrams, indicate the control signals to give each pipeline register by putting a single letter in each box; use N for normal, B for bubble, and S for stall.

13.a. (2 pt) Assume that i_5 and i_4 both resulted from an incorrect speculation and should be squashed, but all other instructions are OK and may continue to execute normally.

F		D		E		M		W
☐	i_5	☐	i_4	☐	i_3	☐	i_2	☐	i_1

13.b. (2 pt) Assume that i_3 requires some information that is not yet available and needs another cycle in the D stage, but all other instructions are OK and may continue to execute normally.

F		D		E		M		W
☐	i_5	☐	i_4	☐	i_3	☐	i_2	☐	i_1

Question 14
Suppose we add a new instruction to Y86-64:

```
vOP %rA1,%rA2, %rB1,%rB2
```

This does two `OPq`s at once; for example

```
vadd %rA1,%rA2, %rB1,%rB2
```

will do the same thing as

```
addq %rA1, %rB1
addq %rA2, %rB2
```

except `vadd`

- is a single instruction
- does not set the condition codes
- does both additions in parallel, so `vadd %r8,%r9, %r9,%r8` will put the same value in both `%r8` and `%r9`

This instruction is given the `icode` 12=`C` and uses the same `ifun` values as `OPq`.

14.a. (1 pt) How many bytes long is the encoding for this instruction?

Answer: _____

14.b. (2 pt) The instruction `vadd %r8,%r9, %r10,%r10` would do the same thing as
Pick One

- ○ `r10 += r8 + r9;`
- ○ `r10 += r8 + r9 + r10;`
- ○ `r10 += r8; r10 += r9;`
- ○ `r9 += r8; r10 += r10;`

Question 15
Consider the five-stage pipeline running the instruction sequence

```
    addq %rax,%rcx  // 1
    subq %rcx,%rcx  // 2
    je L            // 3
    xorq %rcx,%rsi  // 4
L:  andq %rcx,%rax  // 5
```

15.a. (1 pt) The backward-taken forward-not-taken heuristic will predict the jump
Pick One

- ○ correctly
- ○ incorrectly
- ○ cannot tell from the information provided

15.b. (2 pt) Assume the branch predictor predicts "not taken" so the `xorq` is fetched and decoded before the correctness of that prediction is known. The `xorq`'s decode will forward the value of `%rcx`; what stage will it forward the value from?
Pick One

- ○ Fetch
- ○ Decode
- ○ Execute
- ○ Memory
- ○ Writeback
- ○ None; `%rcx` is read from the register file

Question 16
Consider changing the five-stage pipeline into a seven-stage pipeline by splitting execute into three stages:

1. bit-wise work and non-carry part of add/sub
2. carry part of add/sub
3. compare result to 0 and set condition codes

16.a. (5 pt) Which of the following would be hazards in this new seven-stage pipeline?
Select all that apply

- ☐ `pushq` followed by `pushq`
- ☐ `OPq` followed by `OPq`
- ☐ `OPq` followed by `irmovq`
- ☐ `irmovq` followed by `OPq`
- ☐ `OPq` followed by `jXX`

16.b. (3 pt) How many cycles of stalling would the following instruction sequence require? Assume all forwarding occurs during decode.

```
addq       %rax, %rcx
mrmovq 20($rax), %rdx
subq       %rcx, %rdx
rrmovq     %rdx, %r8
rrmovq     %rdx, %r9
pushq      %rdx
```

Answer: _____

Question 17

Many functional units are described by official chip documentation with a latency and a throughput, both measured in units of cycles-per-instruction. Note that this is the inverse of the the usual instructions-per-cycle units for throughput; in this inverted unit, both latency and throughput are better when smaller.

The following ask about the `vsqrtps` instruction on the Skylake architecture which is documented as having latency 12, throughput 6.

17.a. (4 pt) Latency 12 throughput 6 can be achieved by
Select all that apply

- [ ] a 2-stage pipeline, each stage holding the same instruction for 6 cycles
- [ ] a 6-stage pipeline, each stage holding the same instruction for 2 cycles
- [ ] a 12-stage pipeline
- [ ] 2 parallel single-stage units, each holding the same instruction for 12 cycles
- [ ] 2 parallel 6-stage pipelines, each stage holding the same instruction for 1 cycle
- [ ] 2 parallel 6-stage pipelines, each stage holding the same instruction for 2 cycles
- [ ] 6 parallel 2-stage pipelines
- [ ] 6 parallel 12-stage pipelines

17.b. (2 pt) If Intel decide to try to speed up `vsqrtps` by adding additional copies of (parts of) `vsqrtps`'s current circuitry to the chip, they could
Select all that apply

- [ ] reduce cycles-per-instruction latency significantly
- [ ] reduce cycles-per-instruction throughput significantly (i.e. increase more traditional instruction-per-cycle throughput)