Name: _____ Student ID:_____

# CS 354 Exam 2

Exam code: A

Instructions:

1. No electronic devices allowed while taking this exam. No calculators, no cell phones.
2. Do not open this exam until the examiners instruct to you.
3. Fill in your name and UW Student ID on all the pages of this exam.
4. Do not detach the sheets of this exam. If you do so, please let the proctors know so they can staple it back for you.
5. You have 1 hour and 30 minutes to answer all the questions.
6. Please check that this exam contains X printed pages.
7. The total points achievable in this exam is X pts. The actual grade will be for 100 points with 5 points as bonus.
8. All the x86 instructions in this exam follow the AT&T syntax discussed in the class lectures. Your answers must also follow the AT&T syntax.
9. Assume a little endian machine in case you need to in order answer a question.
10. All sizes are specified as a power of 2 (e.g., 1 KB is $2^{10}$ B or 1024 B).
11. In case you need a clarification or any other help during the exam, raise your hand and one of the proctors will assist you.
12. Last page of this exam is intentionally empty and you can use it for your rough work and other calculations.

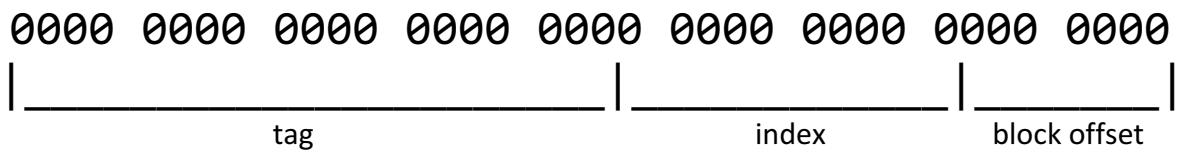
| | Worth | Points Earned |
|---|---|---|
| Question 1 | 10 points | _____ |
| Question 2 | 16 points | _____ |
| Question 3 | 16 points | _____ |
| Question 4 | 20 points | _____ |
| Question 5 | 22 points | _____ |
| Question 6 | 10 points | _____ |
| Question 7 | 10 points | _____ |
| Total | 104 points | _____ |

Name: _____ Student ID:_____

## Question 1: Cache design (10 points)

Answer the following questions.

An address is split into the following parts to access a cache:




(A) What is the maximum amount of memory you can address with an address of this size? Express your answer as 2 to some power (e.g., $2^{10}$ Bytes)

(B) What is the block size of the cache in bytes?

(C) How many sets does the cache have?

(D) Assume the cache is 1 MB ($2^{20}$ Bytes). What is the associativity?

(E) Ignore part D. What would the cache size be if it was 2-way set associative?

Name: _____ Student ID:_____

## Question 2: Assembly Function (16 points)

The following is an implementation of a recursive factorial function with some missing lines. Fill in the blank lines. A C version of factorial is given as a reference. Note: %eax, %ecx, %edx are caller-saved and %ebx, %esi, %edi are callee-saved.

```
int factorial(int N) {
    if (N == 1) {
        return 1;
    }
    int answer = N * factorial(N-1);
    return answer;
}
```

```
factorial:
    # Create the stack frame for this function and store
    # the previous function's stack frame (2 insts)


    # Save the callee saved regs (1 inst)


    movl 8(%ebp), %ebx          # Put N in %ebx
    cmpl $1, %ebx               # Compare N to 1
    je return1                  # Jump to return 1 if the same
    movl %ebx, %ecx             # Copy N to %ecx
    subl $1, %ebx               # Subtract 1 from N (in %ebx)
    # Save the caller saved regs (1 inst)


    # Push the parameter on to the stack and call factorial (2 insts)



    movl 4(%esp), %ecx          # Restore the caller-saved regs
    mull %ecx                   # Perform the multiply (%eax=%eax*%ecx)
    jmp return                  # Return from the function

return1:
    movl $1, %eax               # Move 1 into return value
return:
    movl -4(%ebp), %ebx         # Restore the callee-saved registers
    # Restore the stack of caller (1 inst)
    # Return (1 inst)
```

Name: _____ Student ID:_____

## Question 3: Short answer cache questions (16 points)

(A) Assume an L1 cache has a 5 cycle access time and an average miss time of 100 cycles. What is the average memory access latency (AMAT) if the hit rate of the cache is 90%?

(B) You are a computer architect at Entil Inc. Your boss comes up to you and says that the cache design team has two new designs.
In the first design, they were able to **reduce the average miss time to 70 cycles**.
In the second design, they were able **to increase the L1 hit rate to 95%.**

Unfortunately, there is only area on the chip for one of these two changes.
If the only goal is to reduce AMAT, **which design do you choose**?
**What is the new AMAT** with this design?

(C) Increasing the **associativity** of a cache reduces what kinds of misses (circle one)?

Cold or compulsory Conflict Capacity

(D) In a **single sentence** why do caches need a valid bit?

Name: _____ Student ID:_____

(E) Caching is used in all layers of the computing stack. Here are a few examples of caches that we haven't covered in lecture. For each example, circle temporal or spatial locality as the main type of locality the cache is exploiting.

1) memcached is a software cache often used in datacenters (e.g., the cloud). Every time a user requests a website, first memcached is checked, and if the webpage is there it is returned to the user. Otherwise, the webpage is requested from the backend server.
Temporal / Spatial

2) As we discussed in class, disks are really slow. Some recent disks have increased their sector sizes from 512 B to 4096 B. Each time a single byte or word is read off of the disk an entire sector is read into the disk cache (in memory).
Temporal / Spatial

3) You are not allowed to get books yourself at the library. You have to request them from the librarian. The librarian is smart and doesn't put up the books she fetches immediately. She keeps a ***cache*** of recently requested books at her desk. When a new library user requests a book, she first checks her cache before going to the stacks.
Temporal / Spatial

(F) State **one** advantage of having a split I/D cache (two caches, one for instructions and one for data).

Name: _____ Student ID:_____

# Question 4: Cache structure (20 points)

Consider a direct-mapped cache with S = 16 sets and each cache line of size B = 16 bytes in an x86 machine with a total memory of size M = 64 KB ($2^{16}$ B).

**All numbers in the table below are in hex.**

| Direct Mapped Cache | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Index** | **Tag** | **Valid** | **Byte 0** | **Byte 1** | **Byte 2** | **Byte 3** | **Byte 4** | **Byte 5** | **Byte 6** | **Byte 7** | **Byte 8** | **Byte 9** | **Byte A** | **Byte B** | **Byte C** | **Byte D** | **Byte E** | **Byte F** |
| 0 | 29 | 0 | 34 | 29 | 34 | 34 | 34 | 29 | 34 | 34 | 29 | 34 | 34 | 34 | 34 | 34 | 34 | 29 |
| 1 | F3 | 1 | 0D | 8F | 0D | 0D | 0D | 8F | 0D | 0D | 8F | 0D | 0D | 0D | 0D | 0D | 0D | 8F |
| 2 | A7 | 1 | E2 | 4 | E2 | E2 | E2 | 4 | E2 | E2 | 4 | E2 | E2 | E2 | E2 | E2 | E2 | 4 |
| 3 | 3B | 0 | AC | 1F | AC | AC | AC | 1F | AC | AC | 1F | AC | AC | AC | AC | AC | AC | 1F |
| 4 | 80 | 1 | 60 | 35 | 60 | 60 | 60 | 35 | 60 | 60 | 35 | 60 | 60 | 60 | 60 | 60 | 60 | 35 |
| 5 | EA | 1 | B4 | 17 | B4 | B4 | B4 | 17 | B4 | B4 | 17 | B4 | B4 | B4 | B4 | B4 | B4 | 17 |
| 6 | 3B | 0 | 3F | A4 | 3F | 3F | 3F | A4 | 3F | 3F | A4 | 3F | 3F | 3F | 3F | 3F | 3F | A4 |
| 7 | 29 | 0 | 34 | 29 | 34 | 34 | 34 | 29 | 34 | 34 | 29 | 34 | 34 | 34 | 34 | 34 | 34 | 29 |
| 8 | 29 | 0 | 34 | 29 | 34 | 34 | 34 | 29 | 34 | 34 | 29 | 34 | 34 | 34 | 34 | 34 | 34 | 29 |
| 9 | F3 | 1 | 0D | 8F | 0D | 0D | 0D | 8F | 0D | 0D | 8F | 0D | 0D | 0D | 0D | 0D | 0D | 8F |
| A | 23 | 1 | E2 | 4 | E2 | E2 | E2 | 4 | E2 | E2 | 4 | E2 | E2 | E2 | E2 | E2 | E2 | 4 |
| B | 3B | 0 | AC | 1F | AC | AC | AC | 1F | AC | AC | 1F | AC | AC | AC | AC | AC | AC | 1F |
| C | 80 | 1 | 60 | 35 | 60 | 60 | 60 | 35 | 60 | 60 | 35 | 60 | 60 | 60 | 60 | 60 | 60 | 35 |
| D | EA | 1 | B4 | 17 | B4 | B4 | B4 | 17 | B4 | B4 | 17 | B4 | B4 | B4 | B4 | B4 | B4 | 17 |
| E | 1C | 0 | 3F | A4 | 3F | 3F | 3F | A4 | 3F | 3F | A4 | 3F | 3F | 3F | 3F | 3F | 3F | A4 |
| F | 29 | 0 | 34 | 29 | 34 | 34 | 34 | 29 | 34 | 34 | 29 | 34 | 34 | 34 | 34 | 34 | 34 | 29 |

(A) In the boxes below, indicate the bits that are used to determine the following: O (the block offset within the cache line), I (the cache index), and T (the cache tag). Each box should have an O, an I, or a T.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | |

Name: _____ Student ID:_____

(B) Now, for the following 16 bit addresses, perform lookup on the cache and fill in the table below. Assume each access is a single byte load (e.g., movb <addr>, %eax). Refer to your answer in part A for the tag/index/offset bits.

| Address | Address in binary | | | | | | | | | | | | | | | | Hit? Y/N | Data returned (if applicable) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | F | E | D | C | B | A | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |
| 0x3B62 | | | | | | | | | | | | | | | | | | |
| 0x23A7 | | | | | | | | | | | | | | | | | | |
| 0x23A0 | | | | | | | | | | | | | | | | | | |
| 0x41C5 | | | | | | | | | | | | | | | | | | |

Refer to the following code for part C.

```
void mmkji(){
  int i,j,k;
  for (k = 0; k < n; k++)
    for (j = 0; j < n; j++) {
      double r;
      r = B[k][j];
      for (i = 0; i < n; i++)
         C[i][j] += A[i][k]*r;
    }
  }
}
```

```
void mmkij(){
  int i,j,k;
  for (k = 0; k < n; k++)
    for (i = 0; i < n; i++) {
      double r;
      r = A[i][k];
      for (j = 0; j < n; j++)
        C[i][j] += r*B[k][j];
    }
  }
}
```

(C) Which of the above functions will execute more quickly **and why (in one sentence)**? Assume a 16 KB ($2^{14}$ B) direct-mapped cache with 64-byte block, and assume the A, B, and C matrices are much larger than the cache.

Name: _____ Student ID:_____

## Question 5: Random stuff about functions (22 points)

(A) Jason has made up a new ISA. The registers have very simple names (%R0-%R15). Other than the register names, it looks a lot like x86. Below is a snippet of code for two functions.
For each register circle one of caller or callee saved.

```
main:
    ...
    pushl %R0
    pushl %R12
    call some_func
    popl %R12
    popl %R0
    ...
```

```
some_func:
    pushl %R8
    pushl %R9
    pushl %R2
    ...
    popl %R2
    popl %R9
    popl %R8
    ret
```

%R0 is caller / callee saved
%R2 is caller / callee saved
%R8 is caller / callee saved
%R9 is caller / callee saved
%R12 is caller / callee saved

(B) Stack smashing relies on a buffer overflow of an array on the stack. The attacker is able to overwrite the return address on the stack. Name **one** possible outcome of a **successful** stack smashing attack.

(C) Name **one** way we discussed in lecture to **mitigate** stack smashing attacks.

(D) Write a short (2-4 instructions) sequence of assembly instructions that emulates the "call label" instruction. Note: %eip holds a pointer to the **current** executing instruction.

Name: _____ Student ID:_____

(E) For each statement write either “True” or “False” under the statement.

A. Function local variables that do not fit into the CPU cache are stored in the stack.

B. All function local variables are stored in the callee’s frame in the stack.

C. Function return value is stored in the caller’s frame in the stack.

D. Function parameters are stored in the callee’s frame in the stack.

F. Certain registers that are live across function calls are stored in the caller’s stack.

G. The **global** variables that a function writes have a copy in the callee’s stack.

(F) At what offset from the contents of the register %ebp is the 3rd argument accessible from a callee function in the x86 architecture? (i.e., movl X(%ebp), %eax loads the 3rd argument into %eax, what is X?). Assume all arguments and addresses are 4 bytes.

(G) Name the register used for return value of a function in the x86 architecture.

Name: _____ Student ID:_____

## Question 6: Exceptions, I/O, and Signals (10 points)

For each statement below, write either "True" or "False" under the statement.

(A) Reentrant code is often required in interactive kernels (operating systems) because interrupts are **asynchronous** and can happen during execution of the interrupt handler.

(B) The ret instruction performs two actions, it pops the return address off the stack and jumps to that address. An interrupt **can** occur in the middle of this instruction (i.e., after popping the return address but before the jump) rendering the system unable to return from the function.

(C) The file descriptor table (the per-process table that tracks the open files) points directly to the physical location on the disk of the current file.

(D) Systems often use DMA (direct memory access) to transfer data from I/O devices into memory while the CPU is free to do other things.

(E) One reason for multiple privilege levels in the hardware is to prevent the operating system (kernel) from accessing user data.

(F) A signal handler that you write for your application may be executed at any time (asynchronously).

(G) Certain types of exceptions bypass the kernel (operating system) and instead immediately execute a signal handler.

(H) **All** exceptions are asynchronous and there is no way to reproduce an exception.

(I) Without interrupts, all I/O would have to by **synchronous.** This would waste CPU resources decreasing system throughput and/or its energy efficiency.

(J) Signals are one method for inter-process communication.

Name: _____ Student ID:_____

## Question 7: Disks and Storage (10 points)

(A) Order the following five (5) storage technologies in increasing order of latency of access (lowest latency first).
1. DRAM 2. Magnetic Disk 3. Remote Magnetic Disk 4. SRAM 5. Flash (SSD)

Answer here. Include both the number (above) and the name in a list.

For parts B and C, assume that there is a given disk with the following key parameters: a 10 millisecond (ms) maximum seek time, a 6 ms rotation time (for a full rotation), and a 100 MB/s transfer rate when reading or writing to the surface.

(B) Assume you have a request, A, that is sent to the disk. It is a request to read 1 MB of data. What is the **maximum** time in milliseconds (ms) this request can take to complete (including seek, rotation, and transfer) on the disk above? (assume no other requests are being sent to the disk, i.e., this is the only request you have to think about)

(C) Assume that a request, B, was issued to disk immediately following request A (i.e., A is serviced by the disk, and then B is sent to the disk to be serviced immediately after). It is also a read of 1 MB of data, and the request is to the 1 MB on disk immediately following where A is (i.e., if A is at address N, B is at N + 1 MB). How long will the request to B take?

(D) State an advantage of SSDs when compared to magnetic disks.

Name: _____ Student ID:_____

Blank space for extra work

Name: _____ Student ID:_____

Blank space for extra work

Name: _____ Student ID:_____

# x86 Cheat Sheet

**x86 cheat sheet**

**general purpose registers**

```
%eax     (%ax,%ah,%al)
%ecx     (%cx,%ch,%cl)
%edx     (%dx,%dh,%dl)
%ebx     (%bx,%bh,%bl)
%esi
%edi
%ebp [base pointer]
%esp [stack pointer]
```

**program counter**

```
%eip
[instruction pointer]
```

**condition codes (CCs)**

```
cf (carry flag)
zf (zero flag)
sf (sign flag)
of (overflowing flag)
```

**jump**

```
jmp dst      always jump
je dst       jump if equal/zero
jne dst      … not eq/not zero
js dst       … negative
jns dst      … non-negative
jg dst       … greater (signed)
jge dst      … >= (signed)
jl dst       … less (signed)
jle dst      … <= (signed)
ja dst       … above (unsigned)
jb dst       … below (unsigned)
```

dst is address of code (i.e., jump target)

**comparison**

```
cmpl src2, src1
   // like computing src1 - src2
cf=1 if carry out from msb
zf=1 if (src1==src2)
sf=1 if (src1-src2 < 0)
of=1 if two's complement
        under/overflow
```

**testing**

```
testl src2, src1
   // like computing src1 & src2
zf set when src1&src2 == 0
sf set when src1&src2 < 0
```

**set**

```
sete dst      equal/zero
setne dst     not eq/not zero
sets dst      negative
setns dst     non-negative
setg dst      greater (signed)
setge dst     >= (signed)
setl dst      less (signed)
setle dst     <= (signed)
seta dst      above (unsigned)
setb dst      below (unsigned)
```

`dst` must be one of the 8 single-byte reg (e.g., %al)

often paired with `movzbl` instruction
(which moves 8-byte reg into 32-bit & zeroes out rest)

**data movement**

```
movl src, dst
```

src or dst can be:
- immediate (e.g., $0x10 or $4)
- register (e.g., %eax)
- memory (e.g., an address)

limits:
- dst can never be an immediate
- src or dot (but not both) can be memory

**general memory form:**

```
N (register1, register2, C)
```

which leads to the memory address:

```
N + register1 + (C * register2)
```

N can be a large number;
C can be 1, 2, 4, or 8

**common shorter forms:**

```
N              absolute (reg1=0,reg2=0)
(%eax)         register indirect (N=0,reg2=0)
N(%eax)        base + displacement (reg2=0)
N(%eax,%ebx)   indexed (C=1)
```

**example:**

```
movl 4(%eax), %ebx
```

takes value inside register %eax, adds 4 to it, and then fetches the contents of memory at that address, putting the result into register %ebx; sometimes called a "load" instruction as it loads data from memory into a register

**Load Effective Address:**
Does not do any memory reference at all. Loads the effective address of src into dst. dst must be a register.

leal src,dst dst = address of src

**arithmetic**
**two operand instructions**

```
addl src,dst   dst = dst + src
subl src,dst   dst = dst - src
imull src,dst  dst = dst * src
sall src,dst   dst = dst << src (aka shll)
sarl src,dst   dst = dst >> src (arith)
shrl src,dst   dst = dst >> src (logical)
xorl src,dst   dst = dst ^ src
andl src,dst   dst = dst & src
orl src,dst    dst = dst | src
```

**one operand instructions**

```
incl dst      dst = dst + 1
decl dst      dst = dst - 1
negl dst      dst = -dst
notl dst      dst = ~dst
```

**arithmetic ops set CCs implicitly**

```
cf=1 if carry out from msb
zf=1 if dst==0,
sf=1 if dst < 0 (signed)
of=1 if two's complement
     (signed) under/overflow
```