Final Exam

601.229 Computer Systems Fundamentals

Fall 2019
Johns Hopkins University
Instructors: Philipp Koehn and David Hovemeyer

13 December 2019

Complete all questions.

Use additional paper if needed.

Time: 120 minutes.

Name of student: ______________________________

Q1. MIPS *20 points*

(a) [8 points] Consider the following sequence of MIPS instructions:

```
1:   add $t0, $t1, $t2
2:   add $t0, $t0, $t3
3:   sw $t0, 0($a0)
```

Assume that before the sequence is executed, the correct values of `$t1`, `$t2`, `$t3`, and `$a0` are immediately available in the register file.

Do any pipeline stalls occur in this sequence? If so, explain which stalls occur and why they occur. If there are no stalls, briefly explain why and mention any pipeline implementation details relevant for avoiding stalls for this sequence.

(b) [8 points] Consider the following sequence of MIPS instructions:

```
1:   lw $t0, 0($a0)
2:   lw $t1, 4($a0)
3:   add $t2, $t0, $t1
4:   sw $t2, 8($a0)
5:   lw $t3, 0($a1)
6:   lw $t4, 4($a1)
7:   add $t5, $t3, $t4
8:   sw $t5, 8($a1)
```

Explain how to reorder instructions in order to avoid pipeline stalls, insofar as that is possible. Briefly justify your answer.

(c) [4 points] Briefly explain why MIPS uses separate caches for instructions and data.

Q2. Caches *20 points*

(a) [6 points] On a system with 32 bit addresses, the data cache has 32 byte blocks, is 4-way set associative, and is 32,768 (2^{15}) bytes in size (excluding tags and metadata.)

Fill in the table below to indicate the ranges of address bits where the offset, index, and tag are located in an address, according to the cache parameters described above. Assume that bit 0 is the least significant bit of an address, and bit 31 is the most significant bit of the address.

Field	Range of bits
Address	0–31
Offset	
Index	
Tag	

(b) [4 points] Consider the following C function:

```
void scaleMatrix(double matrix[ROWS][COLS], double fac) {
  for (int j = 0; j < COLS; j++)
    for (int i = 0; i < ROWS; i++)
      matrix[i][j] *= fac;
}
```

Assume that ROWS and COLS correctly describe the sizes of the first and second dimensions of the parameter matrix. Briefly explain (1) the performance issue with this function, and (2) how to change the code to fix the issue.

(c) [10 points] Complete the following table. For each address in the *Request* column, indicate the tags of cached blocks after handling the request. Addresses are 8 bits, blocks are 8 bytes, there are 4 sets, and the cache is 2-way set associative. All slots are initially empty. When a block is evicted, select the least-recently-used (LRU) block as the victim.

	Set 0		Set 1		Set 2		Set 3	
Request	Slot 0	Slot 1	Slot 0	Slot 1	Slot 0	Slot 1	Slot 0	Slot 1
	empty	empty	empty	empty	empty	empty	empty	empty
10110001								
10100011								
10011000								
00100100								
10011010								
01000111								
11110000								
00111011								
10110100								
10010111								

Q3. x86-64 assembly, linking *20 points*

(a) [10 points] Write an x86-64 assembly language function called `swapInts` which swaps the values of two `int` variables. The C function declaration for this function would be

```
    void swapInts(int *a, int *b);
```

Hints:

- Think about which registers the parameters will be passed in
- Think about what register(s) would be appropriate to use for temporary value(s)
- Consider that `int` variables are 4 bytes (32 bits), and use an appropriate operand size suffix

Important: Your function should follow proper x86-64 Linux register use conventions. Be sure to include the label defining the name of the function.

(b) [10 points] Consider the following x86-64 assembly language instruction:

```
    callq some_function
```

Assume that `some_function` is not defined in the assembly language module containing this instruction. How is the address of `some_function` resolved (given that the assembler cannot know the eventual address of the function), and what are the roles of the assembler and linker in resolving the address? Explain briefly.

Q4. Virtual Memory *20 points*

(a) [8 points] In the 32 bit x86 architecture, addresses have 32 bits, pages are 4096 bytes, and the page tables defining an address space are structured as a tree, where

- the root node is the *page directory* containing an array of 1024 *page directory entries* containing the physical addresses of *page tables*, and
- each page table is an array of 1024 *page table entries* containing the physical addresses of physical memory pages.

The physical pages are the leaves of the tree. Note that $4096 = 2^{12}$ and $1024 = 2^{10}$.

What is the size of the portion of the overall virtual address space corresponding to a single page directory entry? Explain briefly.

What is the size of the portion of the overall virtual address space corresponding to a single page table entry? Explain briefly.

(b) [6 points] Consider the following sequence of x86-64 assembly instructions:

```
1:   movq %rdi, %r10
2:   addq %rsi, %r10
3:   movq %r10, (%rdx)
4:   movq $1, %rax
```

Explain (1) which instruction could cause a page fault, and (2) which instruction control will return to once the page fault handler completes. (Assume that the page fault handler does not terminate the running program.)

(c) [6 points] Briefly describe a situation in which a page fault will cause the OS kernel to both write data to disk and also load data from disk. Use specific details as appropriate.

Q5. Networks/threads *20 points*

(a) [8 points] Complete the following function called `chat_with_client`. It implements the server's role in the following network protocol:

- Clients will send messages consisting of 4 lower or upper case letters followed by a newline
- For each received client message, the server will send back a response in which each letter is converted to upper case
- As a special case, if the client sends `quit` followed by a newline, the session should be ended

Example session:

Client sends:

```
CSFi
sDon
eYAY
quit
```

Server sends back:

```
CSFI
SDON
EYAY
```

Note that `clientfd` is a file descriptor referring to a TCP socket that can be used for both receiving data and sending data. You may assume the existence of `read_fully` and `write_fully` functions, which are like `read` and `write`, but will read and write the requested number of bytes if possible. You may assume that I/O routines will not fail. You may assume that messages sent by the client will be properly formed.

Continue on next page if necessary.

```
void chat_with_client(int clientfd) {
```

[Continue code for Q5 part (a) here if necessary, and note that Q5 continues to parts (b) and (c) on the following pages]

(b) [8 points] Consider the following data type and functions:

```
typedef struct {
  int a;
  int b;
} IntPair;


IntPair *intpair_create(int aval, int bval) {
  IntPair *pair = malloc(sizeof(IntPair));
  pair->a = aval;
  pair->b = bval;
  return pair;
}


void intpair_destroy(IntPair *pair) {
  free(pair);
}


void intpair_swap(IntPair *pair) {
  int tmp = pair->a;
  pair->a = pair->b;
  pair->b = tmp;
}


void intpair_adjust(IntPair *pair, int adelta, int bdelta) {
  pair->a += adelta;
  pair->b += bdelta;
}


int intpair_getsum(IntPair *pair) {
  int sum = pair->a + pair->b;
  return sum;
}
```

Annotate the above code to show how to make instances of the `IntPair` type safe to access from multiple threads. (I.e., show where code should be inserted.)

(c) [4 points] The purpose of the Internet is pictures of cats. Draw a picture of a cat.

We hope you enjoyed CSF! Have a great break!