ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Department of Computer Science

Last Name : ______________________________

First Name : ______________________________

Leginr. : ______________________________

Systems Programming and Computer Architecture

252-0061-00

Tuesday 7th February 2017, 14:00-17:00

Rules

- You have 180 minutes for the exam.
- Please write your name and Legi-ID number on all sheets of paper.
- Please write in a blue or black pen. Do not use pencil.
- Please write your answers on the exam sheet. If you need more paper, please raise your hand so that we can provide you with additional paper. Write your name and Legi-ID number on those extra sheets of paper.
- Write as clearly as possible and cross out everything that you do not consider to be part of your solution. You must give your answers in either English or German.
- The exam consists of 8 questions. The maximum number of points that can be achieved is 160.
- This exam paper consists of 21 pages in addition to this title page. Please read through the exam paper to ensure that you have all the pages, and if not, please raise your hand.
- You are not allowed to use any electronic or written aids in this exam, except for a German-English dictionary and the x86 reference sheet that should be on your desk. If the reference sheet is missing, please raise your hand.

For examiners' use only:

1	2	3	4	5	6	7	8

Total: ____

Name: ____________________ Leginr: ____________________

Question 1

[20 points]

You have been asked to analyze the cache performance of code to manipulate large matrices.

The particular hardware this code will run on has a **direct mapped** cache of total size 64K bytes, with a block (line) size of 16 bytes.

Furthermore, the cache is **write-back** and **write-allocate**.

First consider the following code to copy one matrix to another.

Assume that the `src` matrix is aligned on a 1 MB boundary, and the `dest` matrix follows immediately follows it. Also, throughout this question assume that the cache starts cold (i.e. empty) and that local variables and computations take place completely within the registers and do not spill onto the stack.

```
void copy_matrix(int dest[ROWS][COLS], int src[ROWS][COLS])
{
    int i, j;

    for (i=0; i<ROWS; i++) {
        for (j=0; j<COLS; j++) {
            dest[i][j] = src[i][j];
        }
    }
}
```

Remember that `sizeof(int) == 4`.

What is the cache miss rate if `ROWS = 128` and `COLS = 128`?

Show your working.

(3 points)

What is the percentage cache miss rate if `ROWS = 128` and `COLS = 192`?

Show your working.

(3 points)

What is the percentage cache miss rate if `ROWS = 128` and `COLS = 256`?

(2 points)

[Question continues on the next page]

[continued]

Now consider two different implementations of a matrix copy-and-transpose operation. Make the same assumptions as before (cold cache, 1MB-aligned `src`, followed in memory by `dest`).

The first implementation is as follows:

```
void copy_n_flip_matrix1(int dest[ROWS][COLS], int src[ROWS][COLS])
{
    int i, j;

    for (i=0; i<ROWS; i++) {
        for (j=0; j<COLS; j++) {
            dest[i][COLS - 1 - j] = src[i][j];
        }
    }
}
```

What is the percentage cache miss rate if `ROWS = 128` and `COLS = 128`?

Show your working.

(3 points)

What is percentage cache miss rate if `ROWS = 128` and `COLS = 192`?

Show your working.

(3 points)

[Question continues on the next page]

[continued]

The second implementation is as follows:

```
void copy_n_flip_matrix2(int dest[ROWS][COLS], int src[ROWS][COLS])
{
    int i, j;

    for (j=0; j<COLS; j++) {
        for (i=0; i<ROWS; i++) {
            dest[i][COLS - 1 - j] = src[i][j];
        }
    }
}
```

What is the percentage cache miss rate if `ROWS = 128` and `COLS = 128`?

Show your working.

(3 points)

What is the cache miss rate if `ROWS = 192` and `COLS = 128`?

Show your working.

(3 points)

Name: ______________________________ Leginr: ________________

Question 2 [19 points]

Consider the source code below, where `M` and `N` are constants declared with `#define`.

```
int array1[M][N];
int array2[N][M];

void copy(int i, int j)
{
    array1[i][j] = array2[j][i];
}
```

Suppose that compiling the above code on a 64-bit x86 Linux system, *without* the compiler optimizer on, generates the following assembly code:

```
copy:
        pushq   %rbp
        movq    %rsp, %rbp
        movl    %edi, -4(%rbp)
        movl    %esi, -8(%rbp)
        movl    -4(%rbp), %eax
        movslq  %eax, %rcx
        movl    -8(%rbp), %eax
        movslq  %eax, %rdx
        movq    %rdx, %rax
        salq    $3, %rax
        addq    %rdx, %rax
        addq    %rax, %rax
        addq    %rdx, %rax
        addq    %rcx, %rax
        movl    array2(,%rax,4), %ecx
        movl    -8(%rbp), %eax
        movslq  %eax, %rsi
        movl    -4(%rbp), %eax
        movslq  %eax, %rdx
        movq    %rdx, %rax
        addq    %rax, %rax
        addq    %rdx, %rax
        salq    $2, %rax
        addq    %rdx, %rax
        addq    %rsi, %rax
        movl    %ecx, array1(,%rax,4)
        nop
        popq    %rbp
        ret
```

[Question continues on the next page]

[continued]

What are the values of `M` and `N`?

(8 points)

Now suppose that the code is recompiled, with *different* values for `M` and `N`, and with the optimizer turned on. The result is now as follows:

```
copy:
        movslq  %esi, %rsi
        movslq  %edi, %rdi
        leaq    (%rsi,%rsi,2), %rdx
        leaq    (%rdi,%rdi,8), %rax
        leaq    (%rsi,%rdx,4), %rdx
        leaq    (%rdi,%rax,2), %rax
        addq    %rdx, %rdi
        addq    %rsi, %rax
        movl    array2(,%rdi,4), %edx
        movl    %edx, array1(,%rax,4)
        ret
```

(8 points)

What are the values of `M` and `N` now?

[Question continues on the next page]

[continued]

Even with a different calculation due to the different values for `M` and `N`, it's clear that the optimizer has also removed a lot of unnecessary code. What code functionality has been removed, and why was it safe to do so?

(3 points)

Question 3 [16 points]

Recall that a **device driver** is a piece of system software which communicates directly with a hardware device such as a UART or network adaptor.

Also, recall that the *hardware device* and the software *device driver* can be thought of as finite state machines (FSMs). In this view, *data* is transferred between the two state machines, and *events* are used by one FSM (either the driver or the device) to signal a state transition in the other FSM.

Give two ways in which data is transferred **from** the driver to the device:

(2 points)

Give two ways in which data is transferred **to** the driver from the device:

(2 points)

[Question continues on the next page]

Name: ______________________ Leginr: ______________

[continued]

How can the driver signal events to the device?

(2 points)

The device signals events to the device driver using *interrupts*. Explain the function of the Programmable Interrupt Controller (PIC) in a computer system, and list the problems it solves.

(3 points)

[Question continues on the next page]

[continued]

Explain how the presence of processor caches can complicate the implementation of device drivers, and what operations must be done to the processor cache by the device driver to ensure correct operation.

(7 points)

Question 4

[26 points]

Explain the difference between a cache **placement** policy and a cache **replacement** policy.

(4 points)

In a typical running computer system, processor cache misses can be divided into 4 different categories, whose names all begin with the letter 'C'. Give the name and definition of each one of them.

(8 points)

[Question continues on the next page]

[continued]

In the MSI cache coherency protocol, a line (block) in a local cache can be in one of three states: **Invalid**, **Modified**, and **Shared**.

For each of the following transitions, what *local operation* (i.e. something that the local core does) will cause the transition to occur?

(3 points)

Invalid $\rightarrow$ Modified?

Invalid $\rightarrow$ Shared?

Modified $\rightarrow$ Shared?

For each of the following transitions, what *remote operation* (i.e. something that a different core does) will cause the transition to occur locally?

(3 points)

Modified $\rightarrow$ Invalid?

Modified $\rightarrow$ Shared?

Shared $\rightarrow$ Invalid?

[Question continues on the next page]

[continued]

The MSI protocol is very simple, and performance can be dramatically improved by adding more states and more signals. The MESI protocol is one such extension.

What specific extra functionality does the MESI protocol provide, and what is its main advantage over MSI?

(5 points)

Explain the problem of *false sharing* in multiprocessor systems.

(3 points)

Name: ______________________________ Leginr: __________________

Question 5

[25 points]

Consider the following tiny 8-bit floating point format:

```
+---+---+---+---+---+---+---+---+
| S | E | E | E | E | M | M | M |
+---+---+---+---+---+---+---+---+
```

This format uses 1 bit for the sign, 4 bits to represent the exponent, and 3 bits to represent the mantissa (fractional part). Assume that, apart from the different size, the standard IEEE conventions for floating-point representation are used.

What is the largest positive number (excluding infinity) representable in this format? Explain in detail why this is the case.

(8 points)

Not including "Not-a-Number" (NaN) values, how many distinct numbers are representable in this format? Explain your answer.

(7 points)

[Question continues on the next page]

[continued]

How many different *positive* integers can be exactly represented in this format?

(10 points)

Name: ______________________________ Leginr: ____________________

Question 6 [21 points]

Recall that a linker classifies program symbols as either **strong** or **weak**. Explain the difference between strong and weak symbols.

(4 points)

Consider the following structure definition:

```
struct rec {
    unsigned int id;
    char         flag;
    short        count;
    char         valid;
};
```

Note that x6_64 is a little-endian machine, which stores the least-significant byte of a word in the byte with the lowest address.

Assuming standard alignment rules for a 64-bit x86 Linux C compiler, sketch the layout of this struct in memory.

(4 points)

[Question continues on the next page]

[continued]

What is the value of `sizeof(struct rec)`?

(2 points)

Jo Programmer writes the following C program:

```
#include <stdio.h>

...

struct rec my_counter;

int main(int c, char *argv[])
{
    my_counter.count = 1;
    my_counter.id = 0xFFFF;
    my_counter.flag = 2;

    inc_counter();

    printf("count = %d\n", my_counter.count);
    return 0;
}
```

Her friend Bob tells her that he has helpfully written code which provides the counter Jo wants to use, and in particular he has written the function `inc_counter()` which adds one to Jo's counter.

He has even compiled it into an object (`.o`) file which Jo can simply link her program against.

Unfortunately, Bob write this code:

```
uint64_t my_counter = 0;

void inc_counter(void)
{
    my_counter++;
    return;
}
```

[Question continues on the next page]

[continued]

Jo tries to compiles her program, link it against Bob's object file. It seems to work until she runs it. Explain what is most likely to happen, and why.

(7 points)

Counld the compiler have prevented this problem? Explain why (or why not).

(2 points)

Could the linker have helped? Explain why (or why not).

(2 points)

Name: ____________________ Leginr: ____________

Question 7 [21 points]

Explain what the following two standard C functions do:

```
int setjmp(jmp_buf env);

void longjmp(jmp_buf env, int val);
```

(6 points)

Explain why these two functions need to be implemented (at least partially) in assembly language rather than completely in C?

(2 points)

[Question continues on the next page]

[continued]

The following is an assembly listing for `setjmp` and `longjmp` on a 64-bit x86 Linux machine, taken from a real library implementation. Label the listing with an explanation of what each part of each of the two functions is doing:

(7 points)

```
setjmp:
  movq    rbx,  0 (rdi)
  movq    rbp,  8 (rdi)
  movq    r12, 16 (rdi)
  movq    r13, 24 (rdi)
  movq    r14, 32 (rdi)
  movq    r15, 40 (rdi)
  leaq    8 (rsp), rax
  movq    rax, 48 (rdi)
  movq    (rsp), rax
  movq    rax, 56 (rdi)
  movq    $0, rax
  ret

longjmp:
  movq    rsi, rax
  movq     8 (rdi), rbp
  movq    48 (rdi), rsp
  pushq   56 (rdi)
  movq     0 (rdi), rbx
  movq    16 (rdi), r12
  movq    24 (rdi), r13
  movq    32 (rdi), r14
  movq    40 (rdi), r15
  ret
```

[Question continues on the next page]

[continued]

This example is from real production code, but appears to contain a minor bug. What is it?

(1 point)

What is the size (in bytes) of a `jmp_buf` in this implementation?

(2 points)

What is its layout?

(3 points)

Name: ______________________ Leginr: ______________

Question 8

[12 points]

In the following question assume:

- `a` and `b` are declared as `int` in C.
- The machine uses two's complement format for signed numbers.
- `MAX_INT` and `MIN_INT` are the maximum and minimum representable signed integer values respectively
- `W` is one less than the number of bits needed to represent an `int` (i.e. `W == 31` on a machine with 32-bit `int`s).
- Right-shifts in C are arithmetic, not logical.

Match each of the descriptions on the left with a line of code on the right (write in the letter).

1. `a`.

   ____

2. `a * 7`.

   ____

3. One's complement of `a`

   ____

4. `a / 4 .`

   ____

5. `a & b`.

   ____

6. `(a < 0) ? 1 : -1 .`

   ____

a. `~(~a | (b ^ (MIN_INT + MAX_INT)))`

b. `((a ^ b) & ~b) | (~(a ^ b) & b)`

c. `1 + (a << 3) + ~a`

d. `(a << 4) + (a << 2) + (a << 1)`

e. `((a < 0) ? (a + 3) : a) >> 2`

f. `a ^ (MIN_INT + MAX_INT)`

g. `~((a | (~a + 1)) >> W) & 1`

h. `~((a >> W) << 1)`

i. `a >> 2`