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

Saturday 3rd February 2018

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 11 questions.
- The maximum number of points that can be achieved is 154.
- This exam paper consists of 33 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	9	10	11

Total:

Name: ______________________________ Leginr: ________________

Question 1 [13 points]

Consider the following function for computing the product of an array of n integers. Notice that the loop has been unrolled by a factor of 3.

```
int aprod(int a[], int n)
{
    int i, x, y, z;
    int r = 1;
    for (i = 0; i < n-2; i+= 3) {
        x = a[i];
        y = a[i+1];
        z = a[i+2];
        r = r * x * y * z; // Product computation
    }
    for (; i < n; i++)
        r *= a[i];
    return r;
}
```

The code is run on a single-core processor which can issue one (1) new integer multiply instruction every cycle.

The latency of an integer multiply on this machine is 4 cycles.

Assume that the run time of the function for an array of length n, measured in clock cycles, is given by the formula:

$$Cn + K$$

– where C is the Cycles per Element (CPE) of the computation.

Suppose we replace the line marked "`Product computation`" with the following code:

```
        r = ((r * x) * y) * z;
```

Assuming that the only limiting factors for the runtime are the latency and issue time of the integer multipler, what is the expected CPE for this version of the program?

Explain your answer.

(3 points)

[Question continues on the next page]

[continued]

Suppose instead we replace the line marked “`Product computation`” with the following code:

```
        r = (r * (x * y)) * z;
```

What theoretical CPE would we expect now?

Explain your answer.

(5 points)

[Question continues on the next page]

[continued]

What about the following substitution for the line marked "`Product computation`":

```
        r = r * ((x * y) * z);
```

What theoretical CPE would we expect now?

Explain your answer.

(5 points)

Question 2 [18 points]

Consider the following C function, where ARG is a decimal constant defined elsewhere:

```
#include <stdint.h>
int64_t multiply( int64_t a )
{
    return a * ARG;
}
```

The following disassemblies are of versions of the `multiply` function, compiled with different values for `ARG` and different optimization settings in the compiler.

Recall that the first (in this case, only) argument to a function is passed in the `rdi` register, and the result returned in the `rax` register.

For each one, say what the value of `ARG` was, and explain why.

```
0000000000000000 <multiply>:
   0: 55                      push   %rbp
   1: 48 89 e5                mov    %rsp,%rbp
   4: 48 89 7d f8             mov    %rdi,-0x8(%rbp)
   8: 48 8b 45 f8             mov    -0x8(%rbp),%rax
   c: 48 c1 e0 02             shl    $0x2,%rax
  10: 5d                      pop    %rbp
  11: c3                      retq
```

(3 points)

Answer:

[Question continues on the next page]

[continued]

```
0000000000000000 <multiply>:
   0: 48 8d 04 bd 00 00 00  lea    0x0(,%rdi,4),%rax
   7: 00
   8: c3                    retq
```

(3 points)

Answer:

```
0000000000000000 <multiply>:
   0: 48 8d 04 7f           lea    (%rdi,%rdi,2),%rax
   4: 48 8d 04 87           lea    (%rdi,%rax,4),%rax
   8: c3                    retq
```

(3 points)

Answer:

[Question continues on the next page]

[continued]

```
0000000000000000 <multiply>:
   0: 31 c0                  xor    %eax,%eax
   2: c3                     retq
```

(3 points)

Answer:

```
0000000000000000 <multiply>:
   0: 48 8d 04 fd 00 00 00  lea    0x0(,%rdi,8),%rax
   7: 00
   8: 48 29 f8               sub    %rdi,%rax
   b: 48 8d 04 c7            lea    (%rdi,%rax,8),%rax
   f: c3                     retq
```

(3 points)

Answer:

[Question continues on the next page]

[continued]

```
0000000000000000 <multiply>:
   0: 48 89 f8              mov    %rdi,%rax
   3: 48 f7 d8              neg    %rax
   6: 48 c1 e0 02           shl    $0x2,%rax
   a: 48 29 f8              sub    %rdi,%rax
   d: 48 c1 e0 02           shl    $0x2,%rax
  11: c3                    retq
```

(3 points)

Answer:

Name: __________________________________ Leginr: ____________________

Question 3

[20 points]

Consider a floating point format which uses 9 bits but otherwise follows IEEE standard format. 4 bits are used for the fractional part, and 4 bits to represent the exponent.

What is the *bias* for this format?

(2 points)

What integer is the largest positive normalized value below infinity?

Give your answer as a decimal number, and show your working.

(4 points)

What real number is represented by the binary value 100000000 in this system? Show your working

(2 points)

[Question continues on the next page]

[continued]

How is the real number 1 represented in binary in this system? Show your working.

(4 points)

What real number is represented by the binary value 111100000 in this system? Show your working

(4 points)

[Question continues on the next page]

[continued]

How is the real number -1/32 represented in binary in this system? Show your working.

(4 points)

Question 4 [22 points]

You have been asked to design the format of a page table for a 32-bit machine. The machine has 32-bit word size, so the Page Table Entry (PTE) size should be 32 bits and the virtual address space is 32 bits wide.

The page size is set at 1kB (1024 bytes).

The page table has 2 levels, and decodes the 32-bit Virtual Address Space into a 34-bit wide Physical Address Space.

Assuming that the level 1 and level 2 page tables are the same size (and so decode the same number of bits of the virtual address), how many bits does each level decode? Explain why.

(2 points)

How large (in bytes) is a single page table at each level as a result?

(3 points)

[Question continues on the next page]

Name: ______________________ Leginr: ______________

[continued]

Assuming that each page table is stored in memory at a “naturally aligned” physical address (i.e. an address which is an integer multiple of the size of the table), what is the *minimum* number of physical address bits that must be stored in the first (top) level Page Table Entry?

(2 points)

What is the *minimum* number of physical address bits that must be stored in the second-level Page Table Entry?

(2 points)

[Question continues on the next page]

[continued]

In addition to the **valid** or **present** bit, give 4 examples of other metadata bits in the PTE that would be useful, and say why.

(8 points)

Now consider the following virtual address space layout for a process on this machine:

- The stack starts at the top of the virtual address space and grows down. It consists of a single page whose physical address is 0x1E000.
- The lowest page in the virtual address space is not mapped.
- Above that is a single page of program text whose physical address is 0x....
- Above that is a single page holding the data segment at ...
- Above that is a single page holding the heap at ...
- All other virtual addresses are invalid (i.e. not mapped).

[Question continues on the next page]

[continued]

How many page tables (at both levels) are needed to create this virtual address space?

(3 points)

Explain why is the lowest page is not mapped

(2 points)

Name: ______________________ Leginr: ______________

Question 5 [9 points]

Consider the design of a fast cache subsystem for the system in the previous question:

- 32-bit word size
- 32-bit virtual address space
- 1kB (1024 byte) page size
- 34-bit wide Physical Address Space.

Which of the following dimensions of L1 data cache allow an efficient **virtually-indexed, physically-tagged** implementation with the Memory Management Unit designed above, and why?

A 32-kilobyte, 8-way set associative cache with 64 bytes per line/block?

(3 points)

[Question continues on the next page]

[continued]

A 8-kilobyte, 16-way set associative cache with 16 bytes per line/block?

(3 points)

A 4-kilobyte, 2-way set associative cache with 8 bytes per line/block?

(3 points)

Question 6

[18 points]

Consider the following code to add up the elements of a matrix of floating point numbers

```
#define ROWS 256
#define COLS 64
float sum_matrix( float *f )
{
  float acc = 0.0;

  for( int c=0; c < COLS; c++) {
    for( int r=0; r < ROWS; r++) {
      acc += f[c + r*COLS];
    }
  }
  return acc;
}
```

This function compiles to the following machine code:

```
0000000000000000 <sum_matrix>:
   0: be 00 00 00 00        mov    $0x0,%esi
   5: 66 0f ef c0           pxor   %xmm0,%xmm0
   9: eb 17                 jmp    22 <sum_matrix+0x22>
   b: 48 63 d0              movslq %eax,%rdx
   e: f3 0f 58 04 97        addss  (%rdi,%rdx,4),%xmm0
  13: 83 c0 40              add    $0x40,%eax
  16: 39 c8                 cmp    %ecx,%eax
  18: 75 f1                 jne    b <sum_matrix+0xb>
  1a: 83 c6 01              add    $0x1,%esi
  1d: 83 fe 40              cmp    $0x40,%esi
  20: 74 0a                 je     2c <sum_matrix+0x2c>
  22: 89 f0                 mov    %esi,%eax
  24: 8d 8e 00 40 00 00     lea    0x4000(%rsi),%ecx
  2a: eb df                 jmp    b <sum_matrix+0xb>
  2c: f3 c3                 repz retq
```

Suppose this code executes on a 64-bit x86 processor with a 1-kilobyte, 1-way set associative data cache with 32 bytes per line/block.

Also assume that the function starts with a cold (i.e. empty) cache, and the cache uses a “Least-Recently Used” (LRU) replacement policy.

[Question continues on the next page]

[continued]

After the first 8 iterations through the *inner* loop (i.e. the first 8 rows of the first column of the matrix), how many cache misses have occurred? Show your working.

(2 points)

How many elements of the matrix are held in the cache at this point?

Show your working.

(2 points)

After the first full interation through the outer loop (i.e. one column of the matrix), how many cache misses have occurred? Show your working.

(2 points)

[Question continues on the next page]

[continued]

After the first **two** iterations through the outer loop, how many cache misses have occurred? Show your working.

(2 points)

How many of these are *compulsory* cache misses? Show your working.

(2 points)

[Question continues on the next page]

[continued]

Now suppose we instead traverse the matrix row-by-row, as follows:

```
#define ROWS 256
#define COLS 64
float sum_matrix( float *f )
{
  float acc = 0.0;

  for( int r=0; r < ROWS; r++) {
    for( int c=0; c < COLS; c++) {
      acc += f[c + r*COLS];
    }
  }
  return acc;
}
```

After the first 8 iterations through the *inner* loop (i.e. the first 8 columns of the first row of the matrix), how many cache misses have occurred? Show your working.

(2 points)

How many elements of the matrix are held in the cache at this point?

Show your working.

(2 points)

[Question continues on the next page]

[continued]

After the first **two** iterations through the outer loop, how many cache misses have occurred? Show your working.

(2 points)

How many of these are *compulsory* cache misses? Show your working.

(2 points)

Question 7

[6 points]

Recall that the MESI cache coherence protocol, as the name suggests, has 4 states: Modified (M), Exclusive (E), Shared (S), and Invalid (I).

If a line is in the Modified (M) state, what can be concluded about the state of the corresponding data in the rest of the system?

(2 points)

If a line is in the Exclusive (E) state, what can be concluded about the state of the corresponding data in the rest of the system?

(2 points)

If a line is in the Shared (S) state, what can be concluded about the state of the corresponding data in the rest of the system?

(2 points)

Question 8 [17 points]

The following code is part of a parallel program to add up the numbers in a large array of floating-point values. The idea is that each thread runs this function, which operates on a different section of the array. The program runs on a large multiprocessor with MESI-based cache coherence.

```
extern double total;
extern lock_t lock;

void totals_1(double *segment, size_t num)
{
    for(int i=0; i < num; i++) {
        lock_acquire(&lock);
        total += segment[i];
        lock_release(&lock);
    }
}
```

When all threads have finished, the main thread reads the total from the `total` variable.

Unfortunately, this code runs barely faster on two cores than on one core. Explain why.

(2 points)

In an attempt to speed things up, the following code is now used:

```
void totals_2(double *segment, double totals[], size_t num)
{
    for(int i=0; i < num; i++) {
        totals[ thread_id() ] += segment[i];
    }
}
```

Where:

- `thread_id()` is a function that returns the index (a small integer, allocated starting at 1) of the calling thread,
- `totals` is an array of `double` values allocated elsewhere, and
- When all threads have finished, the main thread adds up the numbers in the `totals` array to get the final answer.

[Question continues on the next page]

[continued]

When we run this code in a single thread on a single core, it is faster than `total_1`. However, when run with multiple threads on multiple cores (with one thread per core), it still doesn't get significantly faster with 2, 3, or 4 cores.

Explain why.

(4 points)

Oddly enough, when run on 9 cores the code is significantly faster than with 8 cores, but going to 10 cores doesn't help. Why?

(2 points)

[Question continues on the next page]

[continued]

Would upgrading the system from MESI-based coherence to MOESI-based coherence help the problem? Explain why.(or why not)

(5 points)

The next version of the program is as follows:

```
void totals_3(double *segment, double totals[], size_t num)
{
    double acc = 0.0;
    for(int i=0; i < num; i++) {
        acc += segment[i];
    }
    totals[ thread_id() ] = acc;
}
```

This is much better, up to 18 cores, after which performance stops getting better with more cores.

What's likely to be the limiting factor on performance now?

(2 points)

[Question continues on the next page]

[continued]

What hardware feature might qualitatively change this scaling properties of this code if used correctly, and why?

(2 points)

Question 9 [10 points]

Consider a simple device for storing and retrieving fixed-size 512-byte blocks of data on an old-fashioned disk.

The interface to the device consists of memory-mapped device registers and a single descriptor ring.

The ring holds 64 descriptors in a contigous array in main memory.

The format of each descriptor is as follows:

```
struct descriptor {
    uint64_t    buffer_addr; // Address of 512-byte block in memory
    uint32_t    sector_addr; // Sector number to read/write to/from
    uint32_t    flags;       // Descriptor metadata
};
```

The `flags` field of the descriptor is formatted as follows:

- bit 0: 1 if descriptor is owned by the device, 0 otherwise.
- bit 1: 1 if the requested operation is a read from the device, 0 otherwise.
- bits 2-31: reserved; should be zero.

Sketch the layout of this descriptor in memory. Assume conventional 64-bit x86 Linux alignment rules for structures.

(2 points)

[Question continues on the next page]

[continued]

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

(1 points)

Fill in the body of the following function, which takes a pointer to a descriptor as an argument and returns a boolean value which is true if the descriptor is “free”, in other words it is owned by software and not by the device:

(1 points)

```
bool desc_is_free( struct descriptor *d)
{







}
```

Now fill in the body of the following function, which takes a pointer to a descriptor as an argument and returns a boolean value which is true if the descriptor is referring to a “read” operation:

(1 points)

```
bool desc_is_read( struct descriptor *d)
{







}
```

[Question continues on the next page]

[continued]

Now fill in the body of the following function, which takes a pointer to a descriptor as an argument and returns the address in memory of the buffer that the descriptor refers to:

(2 points)

```
void *desc_blockaddr( struct descriptor *d)
{









}
```

Finally, fill in the body of the following function, which takes a pointer to a descriptor as an argument and initializes it to refer to an operation on a block of data in memory at address `block`, a storage sector `sector`, and is a read if the argument `read` is True, otherwise a write.

(3 points)

```
void desc_init( struct descriptor *d, void *block, uint32_t sector, bool read )
{














}
```

Name: ______________________________ Leginr: ________________

Question 10 [9 points]

Now consider the design of the driver software for the device in the previous question. For simplicity, we will ignore concurrency issues: assume there is only a single thread accessing the device driver at all times. Also, assume that the device and driver have been correctly initialized (including the formatting of the descriptor ring).

This machine implements the Intel x86 memory model (i.e. Total Store Ordering).

However, in this machine DMA transfers from a device are **not** coherent with main memory, and so the processor cache contents must be explicitly managed by the driver.

The following is some code from the driver to start operations on the disk to write or read a block. The OS calls `dev_start_write_block()` to queue a write operation at the disk (a similar function `dev_start_read_block()` is not shown here).

```
#define NUM_DESCS 64
struct descriptor ring[ NUM_DESCS ];
int next_desc;   // Points to the next descriptor to be sent to the device
int last_desc;   // Last descriptor returned from the device

void dev_start_write_block( void *block, uint32_t sector )
{
    // Wait until there's a descriptor free
    for(;;) {
        <cache op 1>;
        if (desc_is_free( &ring[ next_desc ] )) {
            break;
        }
        dev_block_thread();
    }
    <cache op 2>;
    desc_init( &ring[ next_desc ], block, False );
    <cache op 3>;
    dev_start_polling();
    next_desc = (next_desc + 1) % NUM_DESCS;
}
```

Assume the following additional functions are already provided:

- `void dev_block_thread()`, which blocks the calling thread until an interrupt is received from the device.
- `void dev_start_polling()`, which tells the device to start polling the descriptor ring if it has stopped for some reason.
- `void cache_flush( void *base, size_t len )`, which writes back to memory any dirty lines in the data cache which lie in the address range starting at `base` and of size `len`.
- `void cache_invalidate( void *base, size_t len )`, which discards any lines in the data cache which lie in the address range starting at `base` and of size `len`.

The lines `<cache op 1>`, etc. are placeholders for the operations that must performed on the processor cache.

[Question continues on the next page]

[continued]

What is the minimum set of operations needed for `<cache op 1>`? Explain why.

(3 points)

What is the minimum set of operations needed for `<cache op 2>`? Explain why.

(3 points)

What is the minimum set of operations needed for `<cache op 3>`? Explain why.

(3 points)

Name: ______________________________ Leginr: ____________________

Question 11 [12 points]

Suppose we have three variables of type `int`, on a machine with a 32-bit word size:

```
int x;
int y;
int z;
```

We also two variables of type `unsigned int`:

```
unsigned ux = (unsigned) x;
unsigned uy = (unsigned) y;
```

Consider the following series of C expressions. For each one, state whether the expression will *always* evaluate to 1. If it does not, provide a set of values for the variables `x`, `y`, and `z` for which the expression does *not* evaluate to 1.

Expresssion: `~x+~y+1 == ~(x+y)`

Answer:

(2 points)

Expresssion: `((x+y)<<4) + y-x == 17*y+15*x`

Answer:

(2 points)

[Question continues on the next page]

[continued]

Expresssion: `(x<y) == (-x>-y)`

Answer:

(2 points)

Expresssion: `((x >> 1) << 1) <= x`

Answer:

(2 points)

Expresssion: `(x >= 0) || (x < ux)`

Answer:

(2 points)

Expresssion: `ux-uy == -(y-x)`

Answer:

(2 points)