Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Department of Computer Science

Computer Architecture and Systems Programming

252-0061-00

Saturday 1st February 2013, 9:00-12:00

Last Name : ______________________________

First Name : ______________________________

Leginr. : ______________________________

Rules

- You have 180 minutes for the exam.
- Please write your name and Legi-ID number on all sheets of paper.
- Please write your answers on the exam sheet. Please also use the reverse sides of the exam sheets. 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 10 questions. The maximum number of points that can be achieved is 160.
- This exam paper consists of 25 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

Name: ______________________ Leginr: ______________

Question 1 [15 points]

Explain the difference between a processor's *architecture*, and its *microarchitecture*.

(4 points)

Now consider the following function, which calculates the product of all the elements in an array of n integers.

```
int product(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;
}
```

What loop unrolling factor has been used in this function?

(1 point)

For the line labeled `Product computation`, we can use parentheses to create 5 different associations of the computation, as follows:

```
r = ((r * x) * y) * z; // A1
r = (r * (x * y)) * z; // A2
r = r * ((x * y) * z); // A3
r = r * (x * (y * z)); // A4
r = (r * x) * (y * z); // A5
```

[Question continues on the next page]

[continued]

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 CPE (cycles per element) and K is a constant.

Suppose we measure all 5 versions of this function on a process where the integer multiplication operation has a latency of 4 cycles and an issue time of 1 cycle.

The following table shows some values of the CPE, and other values missing. The measured CPE values are those that were actually observed. “Theoretical CPE” means that performance that would be achieved if the only limiting factor were the latency and issue time of the integer multiplier.

Fill in the missing entries. For the missing values of the measured CPE, you can use the values from other versions that would have the same computational behavior. For the values of the theoretical CPE, you can determine the number of cycles that would be required for an iteration considering only the latency and issue time of the multiplier, and then divide by the unrolling factor.

(10 points)

Version	Measured CPE	Theoretical CPE
`A1`	4.00	
`A2`	2.67	
`A3`		4/3 = 1.33
`A4`	1.67	
`A5`		8/3 = 2.67

Question 2 [18 points]

Alice is writing code to gather statistics on a large table of floating-point numbers in main memory.

Her computer has the following relevant characteristics:

- Four 32-bit x86 processor cores, running Linux
- Each core has its own 1MB, 4-way set-associative, write-back cache with 64-byte line (block) size.
- MESI-based cache coherence using bus snooping
- A PRAM memory consistency model

Her program gathers the statistics on the table in a single C structure declared as:

```
struct stats {
    char label[5]
    float sum;
    float sumsq;
    int   num;
    char  flag;
};
```

Fill in the following table, which gives the offset (in bytes) of each field of a `struct stats` value in memory from the start of the struct. You may find it helpful to draw a diagram of the layout.

(2 points)

Field	Offset
`label`	0
`sum`	
`sumsq`	
`num`	
`flag`	

.

What is `sizeof(struct stats)` ?

(1 point)

[Question continues on the next page]

[continued]

The input table Alice uses is as follows (the code to initialize the values of the table is not shown):

```
#define TABLE_LENGTH 200000000

static float table[TABLE_LENGTH];
```

Alice writes the following function to fill in the fields in a struct:

```
void calc_stats(int start, int end, struct stats *sp)
{
  int i;
  for( i=start; i<end; i++) {
    sp->num++;
    sp->sum += table[i];
    sp->sumsq += table[i] * table[i];
  }
}
```

She then calls `calc_stats( 0, TABLE_LENGTH, &s )` to calculate the statistics, where `s` is a static `struct stats` (suitably initialized) used to hold the result.

To get the most accurate result from this code, what should Alice do to the table before calling this function, and why?

(3 points)

[Question continues on the next page]

[continued]

Dissatisfied with the performance of her code, Alice decides to use multiple threads on her multicore machine to parallelize the work. She uses an array of statistics structures, and partitions the array among the threads.

```
static struct stats sa[4] = {
  {"1", 0.0, 0.0, 0, 1},
  {"2", 0.0, 0.0, 0, 2},
  {"3", 0.0, 0.0, 0, 3},
  {"4", 0.0, 0.0, 0, 4}
};

void *calc_thread(void *arg)
{
  int p = (int)arg;
  int start = p * TABLE_LENGTH / NUM_THREADS;
  int end   = (p+1) * TABLE_LENGTH / NUM_THREADS;
  printf("Thread %d running!\n", p);
  calc_stats(start, end, &sa[p]);
  printf("Thread %d ending\n", p);
  return NULL;
}

...

  // In main function:
  for( i=0; i < NUM_THREADS; i++ ) {
      pthread_create( &threads[i], NULL, calc_thread, (void *)i );
  }

  for( i=0; i < NUM_THREADS; i++ ) {
      pthread_join( threads[i], NULL );
  }
```

Alice's code spawns each new thread to run the function `calc_thread`, with a number indicating which partition of the array to work on.

[Question continues on the next page]

[continued]

To her disappointment, this code runs slower with multiple threads than with a single thread.

Explain in detail what is happening here, and give the name of the phenomenon.

(7 points)

Since Alice has taken the Systems Programming course at ETH, she immediately realizes what the problem is. She makes a *single, one-line change* to the definition of `struct stats`, and now her code runs much faster (and scales in performance with number of cores).

What is the change, and why does it solve the problem?

(5 points)

Question 3

[18 points]

What is a device driver, and what does it do?

(5 points)

Most of the time, a device driver is not executing any code. Name three events which cause the device driver to start executing.

(3 points)

Give two ways in which a device driver can transfer data to a hardware.

(2 points)

[Question continues on the next page]

[continued]

In a particular type of operating system known as a *microkernel*, device drivers execute in user-space (rather than kernel mode). Given what you know about virtual memory and caches, it should be clear that achieving this requires a set of features of, and interactions with, the computer's memory system.

Give as many examples as you can of how the device driver might need to interact with the memory system.

(8 points)

Question 4 [10 points]

Consider the following source code, where the numeric constants `C` and `R` are already defined as CPP macro definitions:

```
static int x[C][R];
static int y[R][C];

void copy(int i, int j)
{
  x[i][j] = y[j][i];
}
```

When compiled on a 32-bit x86 Linux machine with a recent version of the `gcc` compiler (4.7.2), the following assembly code is generated:

```
copy:
pushl %ebp
movl %esp, %ebp
movl 12(%ebp), %edx
movl %edx, %eax
addl %eax, %eax
addl %edx, %eax
sall $2, %eax
addl %edx, %eax
movl 8(%ebp), %edx
addl %edx, %eax
movl y(,%eax,4), %ecx
movl 8(%ebp), %edx
movl %edx, %eax
sall $4, %eax
addl %edx, %eax
movl 12(%ebp), %edx
addl %edx, %eax
movl %ecx, x(,%eax,4)
popl %ebp
ret
```

[Question continues on the next page]

[continued]

What are the values of `C` and `R`?

Show how you arrived at your answer, either by annotating the assembly code above or explaining the code below.

(10 points)

Question 5 [12 points]

Consider the following function:

```
unsigned int mul27(unsigned int v)
{
    return (27 * v);
}
```

Assuming a 32-bit machine, give a C expression which is equivalent to the multiplication by 27, but only uses C addition and shift operators.

Your answer should use the minimum number of operators required.

(4 points)

In general, given a constant unsigned integer multiplier `k`, what is smallest total number of shifts and adds required to express multiplication by `k`, and why?

(4 points)

[Question continues on the next page]

[continued]

Now consider the following, slightly different function:

```
unsigned int mul25(unsigned int v)
{
    return (25 * v);
}
```

In the following assembly code, fill in the gaps with appropriate `addl`, `movl`, and `sall` instructions **only** to complete the code.

(4 points)

```
mul25:
pushl %ebp
movl %esp, %ebp
movl 8(%ebp), %edx
```

```
popl %ebp
ret
```

Question 6 [17 points]

Crazy Computing AG of Zurich will shortly announce the *CCZ*, a new low-power 32-bit processor. Most of the power savings come from reducing the size of IEEE float-point registers from 64 to 8 bits, while keeping the integer unit at a word length of 32 bits.

Floating point numbers on the CCZ have the following format:

```
Bit: 7 6 5 4 3 2 1 0
    +-+-+-+-+-+-+-+-+
    |s|e|e|e|e|m|m|m|
    +-+-+-+-+-+-+-+-+
```

In other words, 4 bits are used to represent the exponent and 3 bits used to represent the fractional part of a number.

What range of exponent values can this format express for normalized IEEE floating point numbers?

(3 points)

What is the largest normalized number that can be represented using this format? Show your working.

(3 points)

[Question continues on the next page]

[continued]

When the company was criticized for making the floating point format so small, they replied that it did make conversion from floating point numbers to integers particularly easy.

They also admitted that the first version of the CCZ does not support denormalized numbers, except 0.

Fill in some C code in the following function to convert an 8-bit `float` value (which you can assume is either normalized, or else is zero) on the CCZ into a signed 32-bit `int`:

(7 points)

```
int from_float(float f)
{
    int result;

    int rep = 0xff & (*((int *)&f));









    return result;
}
```

[Question continues on the next page]

[continued]

Someone at PC-AG suggested early on that only 3 bits of mantissa were a bad idea, and that the company should look at a different floating point format:

```
Bit: 7 6 5 4 3 2 1 0
    +-+-+-+-+-+-+-+-+
    |s|e|e|e|m|m|m|m|
    +-+-+-+-+-+-+-+-+
```

If the company had decided to do this, what would have been the largest representable floating point number on the CCZ? Give your answer in decimal and show your working.

(4 points)

Question 7

[16 points]

Define the term “program order” in the context of memory consistency.

(3 points)

Define the term “visibility order” in the context of memory consistency.

(3 points)

[Question continues on the next page]

[continued]

Consider a system with two processors. Assume that:

- All caches write-back and are initially empty.
- The system implements the MESI coherency protocol.
- The memory system implements the PRAM consistency.
- There are no conflict misses.
- No two variables occupy the same cache line.

The following variables have been declarared:

```
int u = 0;
int v = 0;
int *up = &u;
int *vp = &v;
```

Processor A executes the followng code:

```
int l1 = *up;
*vp = 1;
```

Processor B executes the following code:

```
int l2 = *vp;
*up = 1;
```

Suppose that when both processors have finished their respective code sections, `l1` is 0 and `l2` is 1. Fill in the following table showing one possible state for the caches at this point.

(8 points)

Address	Processor A (M, E, S, or I)	Processor B (M, E, S, or I)
`l1`		
`l2`		
`u`		
`v`		

[Question continues on the next page]

[continued]

Now suppose instead that when both processors have finished their respoective code sections, both `l1` and `l2` are 1.

Does this change your answer? If so, say what changes. If not, explain why.

(2 points)

Question 8

[24 points]

This question is about the numerical properties of data types in C.

Consider the following type declarations:

```
uint32_t ux, uy;
int32_t x, y;
float f, g;
double d, e;
```

For each of the following cases, you should either:

- Explain why the statement is true for all possible values of the variables, or
- For integer expressions, give values (expressed in either decimal or hexadecimal) for which the statement is false. For floating-point expressions, describe a case in which the expression is false.

`x>0 && y>0` $\Longrightarrow$ `x + y >= 0` :

(2 points)

`(d+f)-d == f` :

(2 points)

`d == (float) d` :

(2 points)

`x >= 0` $\Longrightarrow$ `-x <= 0` :

(2 points)

[Question continues on the next page]

[continued]

`d < 0.0` $\Longrightarrow$ `(d*2) < 0.0` :

(2 points)

`x <= 0` $\Longrightarrow$ `-x >= 0` :

(2 points)

`x == (int)(double) x` :

(2 points)

`ux > -1` :

(2 points)

`f == -(-f)` :

(2 points)

`f == (float)(double) f` :

(2 points)

[Question continues on the next page]

[continued]

`(x|-x)>>31 == -1` :

(2 points)

`x * x >= 0` :

(2 points)

Name: ______________________________ Leginr: ______________________

Question 9 [18 points]

The following table gives the parameters for a number of different caches, where m is the number of physical address bits, C is the cache size (number of data bytes), B is the block size in bytes, and E is the number of lines per set. For each cache, determine the number of cache sets (S), tag bits (t), set index bits (s), and block offset bits (b).

(14 points)

Cache	m	C	B	E	S	t	s	b
1.	32	1024	4	4	64	24	6	2
2.	32	1024	8	128				
3.					32	22	5	5
4.	32	1024	32	4				
5.	48	32768		8			6	6
6.	48	65536	64	2				
7.	48				64	36	6	6

(2 points)

Which (if any) of the caches above is direct mapped? Explain your answer.

(2 points)

Which (if any) of the caches above is fully associative? Explain your answer.

Question 10

[12 points]

Consider the following fragment of IA32 code from the C standard library:

```
0x400446e3 <malloc+7>: call   0x400446e8 <malloc+12>
0x400446e8 <malloc+12>: popl   %eax
```

After the `popl` instruction completes, what hex value does register `%eax` contain?

(4 points)

Now consider the following, rather bad, C code:

```
/* copy string x to buf */
void foo(char *x)
{
    int buf[1];
    strcpy((char *)buf, x);
}

void callfoo()
{
    foo("abcdefghi");
}
```

[Question continues on the next page]

[continued]

Here is the corresponding machine code on a 32-bit Linux/x86 machine:

```
080484f4 <foo>:
080484f4: 55                pushl  %ebp
080484f5: 89 e5             movl   %esp,%ebp
080484f7: 83 ec 18          subl   $0x18,%esp
080484fa: 8b 45 08          movl   0x8(%ebp),%eax
080484fd: 83 c4 f8          addl   $0xfffffff8,%esp
08048500: 50                pushl  %eax
08048501: 8d 45 fc          leal   0xfffffffc(%ebp),%eax
08048504: 50                pushl  %eax
08048505: e8 ba fe ff ff    call   80483c4 <strcpy>
0804850a: 89 ec             movl   %ebp,%esp
0804850c: 5d                popl   %ebp
0804850d: c3                ret

08048510 <callfoo>:
08048510: 55                pushl  %ebp
08048511: 89 e5             movl   %esp,%ebp
08048513: 83 ec 08          subl   $0x8,%esp
08048516: 83 c4 f4          addl   $0xfffffff4,%esp
08048519: 68 9c 85 04 08    pushl  $0x804859c   {\em# push string address}
0804851e: e8 d1 ff ff ff    call   80484f4 <foo>
08048523: 89 ec             movl   %ebp,%esp
08048525: 5d                popl   %ebp
08048526: c3                ret
```

Recall that:

- `strcpy(char *dst, char *src)` copies the string at address `src` (including the terminating null character) to address `dst`.
- `strcpy` does not check the size of the destination buffer.
- 32-bit Linux/x86 machines are Little Endian.
- The ASCII character code for 'a' is `0x61`.

Now consider what happens when `callfoo` calls `foo` with the input string "`abcdefghi`".

List the contents of the following memory locations immediately after `strcpy` returns to `foo`. Each answer should be an unsigned 4-byte integer expressed as 8 hex digits.

(4 points)

`buf[0] = 0x____________________`

`buf[1] = 0x____________________`

`buf[2] = 0x____________________`

[Question continues on the next page]

[continued]

Immediately **before** the `ret` instruction at address `0x0804850d` executes, what is the value of the frame pointer register `%ebp`?

(2 points)

```
%ebp   = 0x____________________
```

Immediately **after** the `ret` instruction at address `0x0804850d` executes, what is the value of the program counter register `%eip`?

(2 points)

```
%eip   = 0x____________________
```