Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Department of Computer Science

Last Name : _____

First Name : _____

Leginr. : _____

Computer Architecture and Systems Programming

252-0061-00

Thursday 28th January 2016, 9:00-12:00

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. 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 9 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	9

Total: ____

Name: _____ Leginr: _____

Question 1

[23 points]

For each of the four cache block states (M, E, S, and I) in the MESI cache coherence protocol, define the state and what it means.

M: (2 points)

E: (2 points)

S: (2 points)

I: (2 points)

[Question continues on the next page]

[continued]

Consider a 4-processor machine which implements the MESI protocol for cache coherence. The cores are named A, B, C, and D, and start with their caches empty. The following table shows a sequence of operations from various cores on a single memory word `p`, and `x`, `y`, and `z` are local register `int` variables.

Fill in the blank entries in the table with the correct cache block states for each core, and also the value held in one or more of the caches, and the value in the memory location `p`.

(10 points)

time	core	operation	cache A state	cache B state	cache C state	cache D state	value in cache	value memory
1	–	–	I	I	I	I	-	0
2	A	`x = *p;`						
3	B	`y = *p;`						
4	C	`*p = 1;`						
5	D	`z = *p;`						
6	A	`*p = 2;`						

[Question continues on the next page]

[continued]

The MOESI protocol adds an additional state, *Owned*, to the four states in MESI. What does this state mean?

(3 points)

What particular limitation of the MESI protocol does the MOESI extension remove?

(2 points)

Question 2 [22 points]

The following C function transposes a `ROWS` × `COLS` matrix into a second `COLS` × `ROWS` matrix:

```
void transpose(double *src, double *dst)
{
    int r,c;
    for( r = 0; r < ROWS; r++) {
        for ( c = 0; c < COLS; c++) {
            dst[c * COLS + r] = src[ r * ROWS + c];
        }
    }
}
```

It compiles into the following code on 64-bit x86 machines:

```
transpose:
        leaq    17672(%rdi), %r8
.L2:
        leaq    264(%rdi), %rcx
        movq    %rsi, %rdx
        movq    %rdi, %rax
.L5:
        movsd   (%rax), %xmm0
        addq    $8, %rax
        addq    $264, %rdx
        movsd   %xmm0, -264(%rdx)
        cmpq    %rcx, %rax
        jne     .L5
        addq    $376, %rdi
        addq    $8, %rsi
        cmpq    %r8, %rdi
        jne     .L2
        rep ret
```

What are the values of `ROWS` and `COLS`? Explain your answers.

(5 points)

[Question continues on the next page]

[continued]

Each of the following assembly language fragments is the 64-bit x86 compiled form of a function which multiplies a single `int` argument and returns the result as another `int`. For each one, say what the multiplier is, and why.

(8 points)

```
mul_a::
        leal    (%rdi,%rdi,2), %eax
        ret
```

```
mul_b::
        movl    %edi, %eax
        sall    $5, %eax
        subl    %edi, %eax
        ret
```

```
mul_c::
        movl    %edi, %eax
        sall    $4, %eax
        addl    %edi, %eax
        ret
```

```
mul_d::
        leal    (%rdi,%rdi,8), %eax
        leal    (%rdi,%rax,2), %eax
        ret
```

[Question continues on the next page]

[continued]

Finally, in the following C function, `DIV` is a non-negative compile-time constant:

```
unsigned char divchar( unsigned char x )
{
     return x / DIV;
}
```

When compiled on a 64-bit x86 machine, the following assembly language is produced:

```
divchar:
        shrb    %dil
        movzbl  %dil, %edi
        leal    (%rdi,%rdi,2), %eax
        sall    $4, %eax
        addl    %edi, %eax
        shrw    $10, %ax
        ret
```

What is the value of `DIV`, and why?

Hint: recall that x86 registers have different names depending on their size, so that `%ax` is the 16-bit register corresponding to `%eax` (and `%rax`), and `%dil` is the 8-bit register corresponding to the least-significant byte of `%edi` (and `%rdi`)

(9 points)

Question 3 [23 points]

The `setjmp()` and `longjump()` functions in the standard C library have the following signatures::

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

Recall that `setjmp()` saves the stack context/environment in `env` and returns zero. `longjmp()` takes the same `env` and then jumps back to `setjmp()`, which this second time returns `val` or 1 if `val` is zero.

Describe what software-visible processor state needs to go in a `jmp_buf` on an x86_64 processor, what processor state does not, and why.

(9 points)

[Question continues on the next page]

[continued]

In particular, does the Instruction Pointer (IP) register need to be stored in a `jmp_buf`? Explain your answer.

(2 points)

The manual page for `longjmp()` says that you can't call it with an `env` if the function which called the corresponding `setjmp()` function has returned. Why?

(1 point)

[Question continues on the next page]

[continued]

Some languages provide a facility called *exceptions* (not to be confused with processor exceptions like interrupts). A function can *raise* (or *throw*) an exception:

```
raise(e1);
```

This causes all the currently executing functions to return immediately up the stack until a *try block* (or *handler*) is encountered::

```
try {
  ...
  // call_some_function;
  ...
} except(e2) {
  // handle exception e2
}
// continue normally.
```

Note that exception handlers can be nested.

It turns out that exceptions like this can be added to C using preprocessor (`cpp`) macros plus `setjmp()` and `longjmp()`.

Show how this can be done. Describe what global variables are required for a (single-threaded) program to raise and catch exceptions, and sketch the macros corresponding to `raise()`, `try`, and `except()`.

Note: for simplicity, you can assume that all exceptions (the `e1` and `e2` in the above example) are non-zero values of type `int`.

(11 points)

[Question continues on the next page]

[continued]

Question 4

[35 points]

Cache misses are generally classified into four categories, based on the cause of the cache miss. One such category is a "coherence miss", caused by a cache line (i.e. block) having been previously evicted by the cache coherency protocol.

Define the other three types of cache miss, saying what causes them.

(6 points)

What, in general, is the benefit of increasing the associativity of a cache if the overall cache size (in bytes) remains the same? Explain why.

(2 points)

[Question continues on the next page]

[continued]

What is the potential downside of increasing the associativity in this way?

(2 points)

Recall that a cache lookup divides an address into three parts: tag, index, and offset.

Explain how each field is used in the process of looking up an address in the cache.

Cache tag:

(2 points)

Cache index:

(2 points)

Cache offset:

(2 points)

[Question continues on the next page]

[continued]

The MIPS R4400 L1 data cache had the following characteristics:

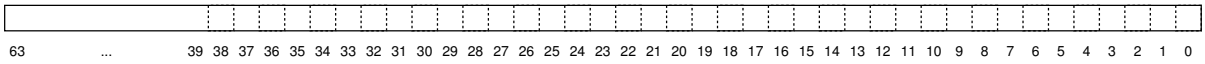
- 16kB size
- 32-byte lines (blocks)
- 2-way set associative

How many sets were there in this cache? Why?

(2 points)

The R4400 had a 39-bit virtual address. Show on the diagram how this address was divided into cache index, cache tag, and cache offset.

(4 points)

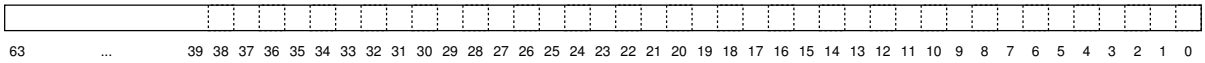


[Question continues on the next page]

[continued]

The R4400 also had a virtual (and physical) page size of 8kB, and a fully-associative TLB. Show on the diagram how a virtual address was divided in to Virtual Page Offset and TLB tag.

(2 points)



In the lectures course we saw how, on a 64-bit x86 architecture processor, the lookup of a virtual address in the TLB and in the cache could be overlapped. This trick does not work on the R4400 – explain why, and also say what problem might occur if the processor tried to overlap TLB and cache lookups.

(6 points)

[Question continues on the next page]

[continued]

In fact, the designers of the MIPS R4400 *did* allow this problem to occur, on the basis that it was much faster to allow it, and the problem could be fixed by careful design of the virtual memory part of the operating system.

Suggest one way this could be achieved.

(5 points)

Question 5

[16 points]

Consider a **5-bit** two's complement representation.

Fill in the empty boxes in the following table.

Addition and subtraction should be performed based on the rules for 5-bit, two's complement arithmetic.

Number	Decimal Representation	Binary Representation
Zero	0	
-2	-2	
9	9	
-14	-14	
		0 1100
		1 0100
TMax		
TMin		
TMin+TMin		
TMin+1		
TMax+1		
−TMax		
−TMin		

Question 6 [16 points]

Consider the following 8-bit floating point representation based on the IEEE floating point format:

- There is a sign bit in the most significant bit.
- The next 3 bits are the exponent. The exponent bias is $2^{3-1} - 1 = 3$.
- The last 4 bits are the fraction.
- The representation encodes numbers of the form: $V = (-1)^s \times M \times 2^E$, where M is the significand and E is the exponent.

The rules are like those in the IEEE standard (i.e. normalized and denormalized numbers, and the same representation of 0, infinity, and NAN).

Fill in the table below for this format. Here are the instructions for each field:

- **Binary:** The 8 bit binary representation.
- **M:** The value of the significand. This should be a number of the form x or $\frac{x}{y}$, where x is an integer, and y is an integral power of 2. Examples include 0, $\frac{3}{4}$.
- **E:** The integer value of the exponent.
- **Value:** The numeric value represented by the number.

  Note: you need not fill in entries marked with "—".

Description	Binary	M	E	Value
Minus zero				-0.0
—	01000101			
Smallest denormalized				
Largest normalized				
One				1.0
—				5.5
Positive infinity		—	—	$+\infty$

Name: _____ Leginr: _____

Question 7

[9 points]

Consider the following C declaration:

```
struct tree_node{
   char c;
   double value;
   struct tree_node* next;
   int flag;
   struct tree_node* left;
   struct tree_node* right;
};

/* tree_nodeTree is an array of N pointers to tree_node structs */
struct tree_node Tree[N];
```

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

(2 points)

Draw a diagram below showing how a `struct tree_node` is laid out in memory. Indicate the offset of each field from the start of the structure, and also any areas of padding introduced by the compiler. Assume standard 64-bit x86 Linux alignment conventions.

(7 points)

Name: _____ Leginr: _____

Question 8 [10 points]

Briefly describe the difference between *synchronous* and *asynchronous* processor exceptions.

(2 points)

Recall that after the handler for a *synchronous* exception has finished, one of the three things can happen:

1. The operating system can cause the entire machine to halt.
2. The operating system can kill or terminate the process causing the exception.
3. The process causing the exception can be resumed by restarting the machine instruction that was executing when the execution occurred.
4. The process causing the exception can be resumed by jumping to the machine instruction immediatately after the one that was executing when the execution occurred.

For each of these, explain why each of these would be appropriate and give an example of an exception that would trigger this behaviour.

1. Halt the machine:

(2 points)

[Question continues on the next page]

[continued]

2. Kill the process:

(2 points)

3. Restart the instruction:

(2 points)

4. Start the next instruction:

(2 points)

Name: _____ Leginr: _____

Question 9

[6 points] ☐

What is a *breakpoint*? What does it do, and how is it used?

(6 points)