Full Name:_____

Datorarkitektur, 2006

Tentamen 2006-03-10

Instructions:

- Make sure that your exam is not missing any sheets, then write your full name on the front.
- Write your answers in the space provided below the problem. If you make a mess, clearly indicate your final answer.
- The exam has a maximum score of 60 points.
- The aproximate limits for grades on this exam are:
  - To pass (G or 3): 30 points.
  - For grade 4: 43 points.
  - For grade VG: 50 points.
  - For grade 5: 55 points.
- The problems are of varying difficulty. The point value of each problem is indicated. Pile up the easy points quickly and then come back to the harder problems.
- This exam is OPEN BOOK. You may use any books or notes you like. Good luck!

Problem 1. (10 points):

Consider the following 5-bit floating point representation based on the IEEE floating point format:

- There is a sign bit in the most significant bit.
- The next two bits are the exponent. The exponent bias is 1.
- The last two bits are the significand.

The rules are like those in the IEEE standard (normalized, denormalized, representation of 0, ∞, and NAN). As described in lectures, the floating point format encodes numbers in a form:

$$(-1)^s \times m \times 2^E$$

where m is the *mantissa* and E is the *exponent*.
The table below enumerates the entire non-negative range for this 5-bit floating point representation. Fill in the blank table entries using the following directions:

E**:** The integer value of the exponent.

m**:** The fractional value of the mantissa. **Your answer must be expressed as a fraction of the form** $x/4$**.**

Value: The numeric value represented. **Your answer, except for the last value, must be expressed as a fraction of the form** $x/4$.

You need not fill in entries marked "—".

Bits	E	m	Value
0 00 00	—	—	0
0 00 01			
0 00 10			
0 00 11			
0 01 00			
0 01 01			
0 01 10			
0 01 11			
0 10 00	1	$4/4$	$8/4$
0 10 01			
0 10 10			
0 10 11			
0 11 00	—	—	

Problem 2. (10 points):

Consider a **6-bit** two's complement representation. Fill in the empty boxes in the following table:

Number	Decimal Representation	Binary Representation
Zero	0	
n/a	-1	
n/a	5	
n/a	-10	
n/a		01 1010
n/a		10 0110
TMax		
TMin		
TMax+TMax		
TMin+TMin		
TMin+1		
TMin−1		
TMax+1		
−TMax		
−TMin		

Problem 3. (8 points):

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

```
int mat1[M][N];
int mat2[N][M];

int copy_element(int i, int j)
{
    mat1[i][j] = mat2[j][i];
}
```

This generates the following assembly code:

```
copy_element:
        pushl %ebp
        movl %esp,%ebp
        pushl %ebx
        movl 8(%ebp),%ecx
        movl 12(%ebp),%ebx
        movl %ecx,%edx
        leal (%ebx,%ebx,8),%eax
        sall $4,%edx
        sall $2,%eax
        subl %ecx,%edx
        movl mat2(%eax,%ecx,4),%eax
        sall $2,%edx
        movl %eax,mat1(%edx,%ebx,4)
        movl -4(%ebp),%ebx
        movl %ebp,%esp
        popl %ebp
        ret
```

A. What is the value of `M`:

B. What is the value of `N`:

Problem 4. (8 points):

This problem tests your understanding of how `for` loops in C relate to IA32 machine code
Consider the following IA32 assembly code for a procedure `foo()`:

```
foo:
        pushl %ebp
        movl %esp,%ebp
        movl 12(%ebp),%ecx
        xorl %eax,%eax
        movl 8(%ebp),%edx
        jmp .L3
        .align 4
.L5:
        addl %edx,%eax
        decl %edx
.L3:
        cmpl %ecx,%edx
        jg .L5
        leave
        ret
```

Based on the assembly code above, fill in the blanks below in its corresponding C source code. (Note: you may only use symbolic variables x, y, i, and $result$, from the source code in your expressions below — do *not* use register names.)

```
int foo(int x, int y)
{
  int i, result=0;

  for (i=________; _____________________________; _________) {

    _______________;

  }

  return result;
}
```

The next problem concerns the following C code:

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

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

Here is the corresponding machine code on a 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   # 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
```

Problem 5. (8 points):

This problem tests your understanding of the stack discipline and byte ordering. Here are some notes to help you work the problem:

- `strcpy(char *dst, char *src)` copies the string at address `src` (including the terminating '`\0`' character) to address `dst`. It does **not** check the size of the destination buffer.
- Recall that Linux/x86 machines are Little Endian.
- You will need to know the hex values of the following characters:

Character	Hex value	Character	Hex value
'a'	0x61	'f'	0x66
'b'	0x62	'g'	0x67
'c'	0x63	'h'	0x68
'd'	0x64	'i'	0x69
'e'	0x65	'\0'	0x00

- It might be a good idea to make a drawing of the stack.

Now consider what happens on a Linux/x86 machine when `callfoo` calls `foo` with the input string "`abcdefghi`".

A. 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.

```
buf[0] = 0x__________
buf[1] = 0x__________
buf[2] = 0x__________
```

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

```
%ebp  = 0x__________
```

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

```
%eip  = 0x__________
```

Problem 6. (10 points):

Consider the following function for computing the product of an array of n integers. We have unrolled the loop 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;
}
```

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
```

We express the performance of the function in terms of the number of cycles per element (CPE). As described in the book, this measure assumes the run time, measured in clock cycles, for an array of length n is a function of the form $Cn + K$, where C is the CPE.

We measured the 5 versions of the function on an Intel Pentium III. Recall that the integer multiplication operation on this machine 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.

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

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 3.

Problem 7. (6 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).

Cache	m	C	B	E	S	t	s	b
1.	32	1024	4	4				
2.	32	1024	4	256				
3.	32	1024	8	1				
4.	32	1024	8	128				
5.	32	1024	32	1				
6.	32	1024	32	4				