Namn:_____

Personnummer:_____

Datorarkitektur, 2008

Tentamen 2008-06-03

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 plus 3 possible bonus points.
- The aproximate limits for grades on this exam are:
  - To pass (grade E): 30 points.
  - For grade D: 37 points.
  - For grade C: 45 points.
  - For grade B: 52 points.
  - For grade A: 59 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 but no computer or telephone. Good luck!

Problem 1. (12 points):

Consider the following 12-bit floating point representation similar on the IEEE floating point format:

- There is a sign bit in the most significant bit.
- The next six bits are the exponent. The exponent bias is 31.
- The last five bits are the significand.

The rules are like those in the IEEE standard (normalized, denormalized, representation of 0, infinity, and NAN).
We consider the floating point format to encode numbers in a form:

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

where m is the *mantissa* and E is the exponent.
Fill in the table below for the following numbers, with the following instructions for each column:

Hex: The 4 hexadecimal digits describing the encoded form.

m**:** The fractional value of the mantissa. This should be a number of the form x or x/y, where x is an integer, and y is an integral power of 2. Examples include: 0, 23/16, and 1/32.

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

Value: The numeric value represented. Use the notation x or $x \times 2^z$, where x and z are integers.

As an example, to represent the number $7/2$, we would have $s = 0$, $m = 7/4$, and $E = 1$. Our number would therefore have an exponent field of `0x20` (decimal value $31 + 1 = 32$) (binary 100000_2) and a significand field `0x18` (binary 11000_2), giving a binary representation 010000011000_2 which in hex becomes `0x418`.
You need not fill in entries marked "—".

Description	Hex	m	E	Value
-0				-0
Smallest value > 1				
Largest Denormalized				
$-\infty$		—	—	$-\infty$
Number with hex representation `AA0`	`AA0`			

Problem 2. (8 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 the above code generates the following assembly code:

```
copy:
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp), %edx
        movl    %edx, %eax
        sall    %eax
        addl    %edx, %eax
        sall    $2, %eax
        addl    %edx, %eax
        movl    %eax, %ecx
        addl    12(%ebp), %ecx
        movl    12(%ebp), %edx
        movl    %edx, %eax
        sall    $3, %eax
        addl    %edx, %eax
        addl    8(%ebp), %eax
        movl    array2(,%eax,4), %eax
        movl    %eax, array1(,%ecx,4)
        leave
        ret
```

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

`M` =

`N` =

Problem 3. (9 points):

Consider the following C declarations:

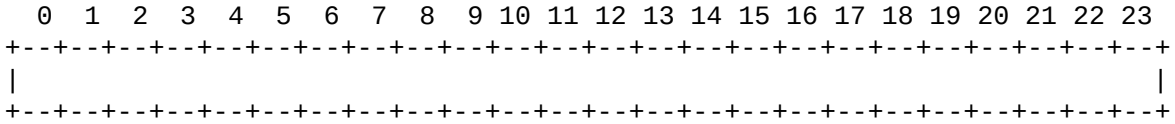
```
typedef struct {
    short code;
    char raw[3];
    long start;
    double data;
} OldSensorData;

typedef struct {
    short code;
    short start;
    short sense;
    char raw[5];
    short ext;
    double data;
} NewSensorData;
```

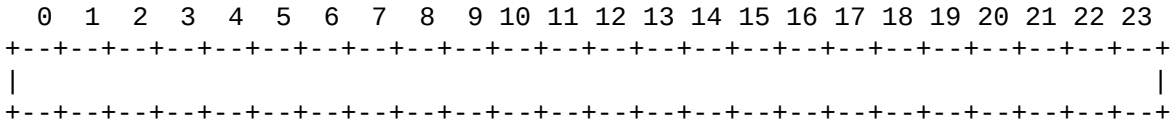
A. Using the templates below (allowing a maximum of 24 bytes), indicate the allocation of data for structs of type `OldSensorData` `NewSensorData`. Mark off and label the areas for each individual element (arrays may be labeled as a single element). **Cross hatch the parts that are allocated, but not used (to satisfy alignment).**

Assume the Linux alignment rules described in book and discussed in lectures. **Clearly indicate the right hand boundary of the data structure with a vertical line.**

`OldSensorData:`



`NewSensorData:`



B. Now consider the following C code fragment:

```
void foo(OldSensorData *oldData)
{
    NewSensorData *newData;

    /* this zeros out all the space allocated for oldData */
    bzero((void *)oldData, sizeof(oldData));

    oldData->code = 0x3A5B;
    oldData->start = 0x935028EC;
    oldData->raw[0] = 0xB1;
    oldData->raw[1] = 0x2D;
    oldData->raw[2] = 0x76;
    oldData->raw[-2] = 0xDC;
    oldData->data = 3.25;

    newData = (NewSensorData *) oldData;

    ...
```

Once this code has run, we begin to access the elements of newData. Below, give the value of each element of `newData` that is listed. Assume that this code is run on a Little-Endian machine such as a Linux/x86 machine. You must give your answer in hexadecimal format. **Be careful about byte ordering!**.

(a) `newData->code  = 0x_____`

(b) `newData->raw[0] = 0x_____`

(c) `newData->raw[2] = 0x_____`

(d) `newData->raw[4] = 0x_____`

(e) `newData->sense  = 0x_____`

Problem 4. (10 points):

A C function `looper` and the assembly code it compiles to on an IA-32 machine running Linux/GAS is shown below:

```
looper:
  pushl %ebp
  movl %esp,%ebp
  pushl %esi
  pushl %ebx
  movl 8(%ebp),%ebx
  movl 12(%ebp),%esi
  xorl %edx,%edx
  xorl %ecx,%ecx
  cmpl %ebx,%edx
  jge .L25
.L27:
  movl (%esi,%ecx,4),%eax
  cmpl %edx,%eax
  jle .L28
  movl %eax,%edx
.L28:
  incl %edx
  incl %ecx
  cmpl %ebx,%ecx
  jl .L27
.L25:
  movl %edx,%eax
  popl %ebx
  popl %esi
  movl %ebp,%esp
  popl %ebp
  ret
```

```
int looper(int n, int *a) {
  int i;
  int x = ________________;

  for(i = ______________;

      ____________________;

      i++) {

    if (_____________________)

       x = ___________________;


    ________________________;
  }

  return x;
}
```

Based on the assembly code, fill in the blanks in the C source code.

Notes:

- You may only use the C variable names `n`, `a`, `i` and `x`, not register names.
- Use array notation in showing accesses or updates to elements of `a`.

Problem 5. (4 points):

Consider the following C functions and assembly code:

```
int fun4(int *ap, int *bp)
{
    int a = *ap;
    int b = *bp;
    return a+b;
}

int fun5(int *ap, int *bp)
{
    int b = *bp;
    *bp += *ap;
    return b;
}

int fun6(int *ap, int *bp)
{
    int a = *ap;
    *bp += *ap;
    return a;
}
```

```
pushl   %ebp
movl    %esp, %ebp
subl    $4, %esp
movl    12(%ebp), %eax
movl    (%eax), %eax
movl    %eax, -4(%ebp)
movl    12(%ebp), %ecx
movl    12(%ebp), %edx
movl    8(%ebp), %eax
movl    (%eax), %eax
addl    (%edx), %eax
movl    %eax, (%ecx)
movl    -4(%ebp), %eax
leave
ret
```

Which of the functions compiled into the assembly code shown?

Problem 6. (7 points):

This problem will test your understanding of stack frames. It is based on the following recursive C function:

```
int silly(int n, int *p)
{
    int val, val2;

    if (n > 0)
        val2 = silly(n << 1, &val);
    else
        val = val2 = 0;

    *p = val + val2 + n;

    return val + val2;
}
```

This yields the following machine code:

```
silly:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $8, %esp
        cmpl    $0, 8(%ebp)
        jle     .L2
        subl    $8, %esp
        leal    -8(%ebp), %eax
        pushl   %eax
        movl    8(%ebp), %eax
        sall    $2, %eax
        pushl   %eax
        call    silly
        addl    $16, %esp
        movl    %eax, -4(%ebp)
        jmp     .L3
.L2:
        movl    $0, -8(%ebp)
        movl    $0, -4(%ebp)
.L3:
        movl    12(%ebp), %edx
        movl    -8(%ebp), %eax
        addl    -4(%ebp), %eax
        addl    8(%ebp), %eax
        movl    %eax, (%edx)
        movl    -8(%ebp), %eax
        addl    -4(%ebp), %eax
        leave
        ret
```

A. Is the variable `val` stored on the stack? If so, at what byte offset (relative to `%ebp`) is it stored, and why is it necessary to store it on the stack?

B. Is the variable `val2` stored on the stack? If so, at what byte offset (relative to `%ebp`) is it stored, and why is it necessary to store it on the stack?

C. What (if anything) is stored at `-24(%ebp)`? If something is stored there, why is it necessary to store it?

D. What (if anything) is stored at `-8(%ebp)`? If something is stored there, why is it necessary to store it?

Problem 7. (10 points):

After a stressful semester you suddenly realize that you haven't bought a single christmas present yet. Fortunately, you see that one of the big electronic stores has CD's on sale. You don't have much time to decide which CD will make the best presents for which friend, so you decide to automatize the decision process. For that, you use a database containing an entry for each of your friends. It is implemented as an 8×8 matrix using a data structure `person`. You add to this data structure a field for each CD that you consider:

```
struct person{

  char name[16];

  int age;
  int male;

  short nsync;
  short britney_spears;
  short dolly_parton;
  short garth_brooks;
}

struct person db[8][8];
register int i, j;
```

Part 1

After thinking for a while you come up with the following smart routine that finds the ideal present for everyone.

```
void generate_presents(){

  for (j=0; j<8; j++){
      for (i=0; i<8; i++) {
          db[i][j].nsync=0;
          db[i][j].britney_spears=0;
          db[i][j].garth_brooks=0;
          db[i][j].dolly_parton=0;
      }
   }

  for (j=0; j<8; j++){
      for (i=0; i<8; i++) {

          if(db[i][j].age < 30){
             if(db[i][j].male)
                   db[i][j].britney_spears = 1;
             else  db[i][j].nsync = 1;
          }
          else{
             if(db[i][j].male)
                   db[i][j].dolly_parton =1;
             else  db[i][j].garth_brooks = 1;
          }
      }
   }

}
```

Of course, runtime is important in this time-critical application, so you decide to analyze the cache performance of your routine. You assume that

- your machine has a 512-byte direct-mapped data cache with 64 byte blocks.
- `db` begins at memory address 0
- The cache is initially empty.
- The only memory accesses are to the entries of the array `db`. Variables `i`, and `j` are stored in registers.

Answer the following questions:

A. What is the total number of read and write accesses? _____.

B. What is the total number of read and write accesses that miss in the cache? _____ .

C. So the fraction of all accesses that miss in the cache is: _____.

Part 2

Then you consider the following alternative implementation of the same algorithm:

```
void generate_presents(){

  for (i=0; i<8; i++){
      for (j=0; j<8; j++) {

          if(db[i][j].age < 30)
             if(db[i][j].male) {
                db[i][j].nsync=0;
                db[i][j].britney_spears=1;
                db[i][j].garth_brooks=0;
                db[i][j].dolly_parton=0;
             }

             else
                db[i][j].nsync=1;
                db[i][j].britney_spears=0;
                db[i][j].garth_brooks=0;
                db[i][j].dolly_parton=0;
             }
          }

          else{
             if(db[i][j].male) {
                db[i][j].nsync=0;
                db[i][j].britney_spears=0;
                db[i][j].garth_brooks=0;
                db[i][j].dolly_parton=1;
             }

             else{
                db[i][j].nsync=0;
                db[i][j].britney_spears=0;
                db[i][j].garth_brooks=1;
                db[i][j].dolly_parton=0;
             }
          }


      }
   }

}
```

Making the same assumptions as in Part 1, answer the following questions.

A. What is the total number of read and write accesses? _____

B. What is the total number of read and write accesses that miss in the cache? _____

C. So the fraction of all accesses that miss in the cache is: _____.