Seminar 1
C and Assembly Programming

Computer Organization and Components / Datorteknik och komponenter (IS1500), 9 hp
Computer Hardware Engineering / Datorteknik, grundkurs (IS1200), 7.5 hp

KTH Royal Institute of Technology

Introduction

The purpose of seminars is to enable active learning of the more theoretical tasks that are typically part of the final written exam. Seminars are optional. However, we strongly recommend that you perform these seminar exercises and attend the seminar.

Rules. You may receive up to 1 extra point on the fundamental part of the written exam if:

- you make an honest *attempt* to solve *all* the seminar exercises on your own. You may discuss the exercises with your friends, but you are not allowed to copy any solutions from anyone or anywhere. You need to have written down a potential solution on all assignments. You are not allowed to skip some exercises, you need to try to provide a solution for all exercises.
- you write down your solutions *by hand* on this exercise form. You are not allowed to hand in machine printed solutions, copies, or handwritten solution on another paper format.
- you bring your solution *personally* to the seminar and attend the whole seminar. This means that you are not allowed to hand in a solution on behalf of someone else.
- you need to have signed this form before you hand it in.
- you are not allowed to attend the seminar if you are not bringing a solution, that is, if you do not bring this form filled out with your own solutions, you cannot attend the seminar.
- you must come to the seminar on time when it starts. If you are not there from the beginning, the assistants may refuse that you participate in the seminar.

Note that the extra point is only valid on the next ordinary exam, and the following two retake exams. During the seminar, the teaching assistant or teacher will go through the solutions and you will correct the solution done by another student. You need to have received at least 50% of the total number of points to pass the seminar. In such a case, you get one extra bonus point on the exam. We recommend that you take a photo of your solutions before you hand it in.

By signing the following, I hereby guarantee that I follow the rules above.

Your name (printed): Philip Salqvist

Signature: Philip S

Date: 2021-01-26

Exercises

1. Assume that the two numbers `-49`$_{10}$ and `113`$_{10}$ are encoded as 8-bit signed values in two's complement form. Sign extend *and* zero extend each of them into 12-bit values. Do it by hand and answer in hexadecimal form. Show the main steps of your solution.

   Your solution:

   $-49_{10} = 11001111_2$

   Sign extended: $1111\ 1100\ 1111_2$ = 0xFCF

   Zero extended: 0x0CF

   $113_{10} = 0111\ 0001_2$

   Sign extended: $0000\ 0111\ 0001_2$

   Zero extended: 0x071

2. Assume that you have a C program with signed integer (`int`) variables `x`, `y`, and `z`. All variables contain some arbitrary values. Write a C-statement that extracts the bits with index 17 to 13 from `x` and places them as the least significant bits in `z`, and extracts the least 3 significant bits of `y` and places them in the bits with index 7 to 5 in `z`. No other bits of `z` should be changed, besides the 8 bits that were extracted from `x` and `y`. Note that the bit index 0 is the least significant bit. Your answer should contain one single C statement together with short notes of what the different parts of the statement do.

   Your solution:

   cleaning up z: z &= 0xFFFFFF00

   getting least significant bits from y and shifting 5 positions:

   ((y & 0x00000007) << 5)

   extracting bits from x and shifting so that lsb starts at position 0:

   ((x & 0x0003E000) >> 13)

   all in one statement:

   z = (z & 0xFFFFFF00) | ((y & 0x3) << 5) | ((x & 0x3E000) >> 13);

3. Write down the function body of the two following C functions. Function `adder` should add together the two integer values that the pointers `x` and `y` points to, and then write the result to where `z` points to. Function `foo` should use function `adder` to add together `a` and `k` and then return the resulting value. For instance, if expression `foo(7)` is executed, value `17` should be returned.

   Your solution:

```
void adder(const int *x, const int *y, int *z){

      *z = *x + *y;

}

int foo(int a){
  const int k = 10;

    int p;
    adder( &k , &a , &p);
    return p;

}
```

4. Write out the MIPS assembly instruction that has the machine code `0x2d28fff9`. You should include the main steps of how you computed your solution.

   Your solution:

machine code: 0x2d28fff9 =

= 0010 1011 0010 1000 1111 1111 1111 1001$_2$

opcode = 001010$_2$ = 10$_{10}$ => slti , I-type instruction

rs = 11001$_2$ = 25$_{10}$ => $t9

rt = 01000$_2$ = 8$_{10}$ => $t0

imm = 1111 1111 1111 1001$_2$ = 65529$_{10}$

MIPS instruction: slti $t0, $t9, 65529

5. Assume that the MIPS machine encoded word of the following instruction is located at address `0x00400000` in the program code memory.

```
j       foo
```

Assume further that label `foo` is located at address `0x0040002c`. What is then the machine encoding of the jump instruction? Include a short explanation of the different parts of encoding.

Your solution:

Opcode: 000010

32-bit jump address:

0000 0000 0100 0000 0000 0000 0010 1100

26-bit address becomes:

0000 0100 0000 0000 0000 0010 11

Machine code becomes:

000010 0000 0100 0000 0000 0000 0010 11

6. Create a C function named `square_reverse` with three parameters. The two first parameters are 64-bit floating-point pointers `x` and `y`, and the third parameter is an integer parameter called `len`. The function must not return any value. Pointer `x` points to an array of length `len` of floating-point values. The function reads out each element of the array, computes the square value of the element (x^2) and then writes back the result into the array `y` in reverse order. That is, the output array `y` has also the length `len`.

For example, if we have the following declarations

```
double in[] = {11.0, 20.0, 100.0};
double out[3];
```

a function call `square_reverse(in,out,3);` should result in that the three elements 10000.0, 400.0, and 121.0 are the content of `out`. Note that your function should also declare parameters as `const` when appropriate.

Your solution:

```
void square_reverse (const double *x, double *y, int len) {
    int i;
    for ( i=0; i<len; i++) {
        y[i] = x[len-1-i];
    }
}
```

Corrected by ____________________________. Total number of points: _________