School of Computer Science and Information Technology
Computer Systems Architecture (G51CSA) Autumn 2008
Thorsten Altenkirch

Coursework 4 (cw id 120)
Monday, 20 October 2008
Deadline: 27 October 08, 12:00

Collaborating in small groups of up to three students is permitted, but you must implement your own programs (absolutely *do not* copy and paste from others) and provide your own answers where appropriate.

Part 1 has to be submitted on paper to the school office before the deadline. Clearly write CSA, coursework 4, *your name*, *your student id*, and the email address of your CSA tutor, i.e. either *asg* or *lyh* on the top of your submission. If your submission is more than one page then the pages have to be stapled together.

Parts 2 and 3 have to be submitted using the departmental coursework submission system, see
http://support.cs.nott.ac.uk/coursework/cwstud/.

Create a directory `ex04` and put all the files to be submitted (but nothing else) into this directory before submitting the directory.

Multiple submission before the deadline are allowed, only the last one will be taken into account.

1. The following exercise is about binary arithmetic. It is important that you carry out these calculations by hand on paper. Hence, the answers to this question have to be **submitted to the school office on paper!**

   Given

$$\begin{aligned} a &= 123 \\ b &= -45 \\ c &= 61 \end{aligned}$$

   Translate a, b, c into 8-bit twos complement and calculate in binary:

   (a) $a + b$
   (b) $b + c$
   (c) $b - c$
   (d) $a + c$

   Show all the steps of your calculation. Use addition of the twos complement for subtraction. Write clearly and use vertical pencil lines to layout your calculations.

   Check your results by translating back into decimal. In which cases did an overflow occur?

2. The following C program reads two numbers, adds them and prints the result.

   ```
   #include <stdio.h>

   int main() {
     int x,y,z;
     /* input */
     printf("x=");
     scanf("%d",&x);
     printf("y=");
     scanf("%d",&y);
     /* calculation */
     z=x+y;
     /* output */
     printf("x+y = %d\n",z);
   }
   ```

   Translate this program into MIPS assembler using `add` for addition. Call your program `add.asm`

   Test your program using SPIM with $a+b$,$a+c$,$b+c$ using a,b,c from part 1.

   Modify your program by using `addu` instead of `add` calling the new program `addu.asm`. Can you find an input for which this program behaves differently from the previous one? Summarize your observations in a text file called `add.txt`

3. Modify your previous program for a super-riscy version of MIPS which doesn't implement addition but only shift and logical operations. Call your program `myadd.asm`.

   Hint: In C this could be achieved by replacing the line `z=x+y` by:

   ```
     int xi,yi,ci,zi,i;
     ci=0;
     z=0;
     for(i=1;i!=0;i=i<<1) {
       xi=x&1;
       x=x>>1;
       yi=y&1;
       y=y>>1;
       zi=xi^yi^ci;
       ci=(xi&yi)|(yi&ci)|(xi&ci);
       if(zi) z = z|i;
     }
   ```

   Test your program using SPIM with $a+b$,$a+c$,$b+c$ using a,b,c from part 1.

4. (*) The logical operations are performed in parallel on all bits of a word but the program from the previous question always operates only on one bit. Can you implement a more efficent version of the program (either in C or MIPS assembler) which exploits this parallelism to reduce the number of operations it has to perform to calculate the sum of two numbers.