COM1006/COM1090 Devices and Networks (Autumn)
Tutorial Sheet #8: Instruction Set Architecture

1. Implement the following instruction

$$[K] \leftarrow ([B]+[C]) / (([E]-[F]\cdot[G])-[H])$$

on different machines with instruction sets as stated below. Remember that multiplication and division is done before addition and subtraction. You must not overwrite variables B-H, but you can use registers D0, D1, etc. as temporary storage. Try to use as few instructions as possible.

Three-address machine	
ADD X, Y, **Z**	[Z]←[X]+[Y]
SUB X, Y, **Z**	[Z]←[X]-[Y]
MULT X, Y, **Z**	[Z]←[X]·[Y]
DIV X, Y, **Z**	[Z]←[X]/[Y]

Two-address machine	
ADD X, **Y**	[Y]←[Y]+[X]
SUB X, **Y**	[Y]←[Y]-[X]
MULT X, **Y**	[Y]←[Y]·[X]
DIV X, **Y**	[Y]←[Y]/[X]
MOVE X, **Y**	[Y]←[X]

One-address machine	
ADD X	[A]←[A]+[X]
SUB X	[A]←[A]-[X]
MULT X	[A]←[A]·[X]
DIV X	[A]←[A]/[X]
LOAD X	[A]←[X]
STORE X	[X]←[A]

2. Now let us compare the resources used by the different address machines above. Assume that every opcode has 8 bits (1 byte) and every operand has 16 bit (2 bytes) length.

   a) What is the length of one instruction in bytes for each of the address machines from question 1? This number describes how large or wide the bus must be to be able to fetch an instruction from memory in one step.

   b) Now look at your solution for question 1 and work out the length of your programs for each address machine in bytes. This tells you how much space your programs would take in memory.

   Considering all the above, including your implementations, what is your favourite instruction set architecture among the above, and why?

3. For each of the following Motorola 68K instructions, name the addressing mode used. Refer to the source operand for instructions with more than one operand:

```
MOVE    1024,D0

CLR     D2

ADD     (A1),D2

OR      $0F,D3

MULU    #42,D4
```

4. Familiarise yourself with the EASy68K assembler and simulator. Start EASy68K from the Start menu and load the following source file:
   https://staffwww.dcs.shef.ac.uk/people/D.Sudholt/campus_only/misc/intro.X68

   Execute the program using the "Assemble Source" button (F9) which looks like a "Play" button. When a dialog pops up, choose "Execute". Now three windows appear:
   - a **console** used for outputting and inputting text and numbers
   - a **memory display** with memory cells in hexadecimal and ASCII
   - a **debugger** showing the assembled program and register contents.

   Find the program's machine code and the defined variables at address `0000` (this location is specified by the `ORG` command at the beginning of the source code).

   The instructions below will ask you to execute your program step-by-step using the "Step over" (F8) button (fifth from left) in the debugger. You can always rewind your program (eighth button) or let it run all the way (second button).

   Note that you cannot edit your program from the debugger, nor open another source file from there. You have to close the debugger or switch between windows. Every time you execute a program a new debugger window will pop up.

   If you hit the "Assemble Source" button and your code contains any syntax errors, the editor lists all errors and the lines where they occur at the bottom. In addition, these lines are set in a brownish colour in your source code.

   Finally, have a look at the help under Help → Help. "Jimmy Mårdells 68000 Reference Guide" contains valuable information such an instruction summary with detailed explanations to each instruction.

   If you want to use EASy68K outside Computer Lab 1, you can download it for Windows from http://www.easy68k.com/, or use any CiCS machine (install it through the Software Center).

   a) Execute the program step-by-step to see what each instruction does. Pay attention to how the affected registers and memory locations change with each instruction.

   b) Rewind and step through the program again. This time look at the status flag register "SR" and how it is affected by instructions. Note how the carry flag "C" is being set after each addition.

   c) Note that machine codes for different instructions may have different lengths. Why is this the case? Would this happen in a RISC architecture?

   d) Step through the program again and watch the program counter "PC" this time. Can you explain how it is being increased, depending on characteristics of the current instruction?