Lab 1
Warm-up : Python and the target machine : LC-3

Credits

This sequence of compilation labs has been inspired by those designed by C. Alias and G. Iooss in 2013/14. In 2016/17 we changed the support language for Python and the target machine LC-3.

Objective

- Start with Python
- Be familiar with the LC-3 instruction set.
- Understand how it executes on the LC-3 processor with the help of a simulator.
- Write simple programs, assemble, execute.

1.1 Quick intro to Python - 1h max

This part is strongly inspired by the Project 1 of ENSL (L3).

Please use a correct text editor ! We don't really care if it is SublimeText, Emacs, Atome or Vim, but please use a text editor made for programming.

`https://www.python.org/` Official tutorial: `https://docs.python.org/2/tutorial/` An amazing interactive one `http://www.learnpython.org/en/Welcome`

Inside the interpreter

And now, let's get to the heart of the matter.

EXERCISE #1 ► **Launch!**
Launch the Python interpreter. Which version is it ? Use a version of Python not older than 2.7. Quit the interpreter with CTRL-D or quit().

EXERCISE #2 ►
Launch the interpreter in interactive mode and use it as a calculator to solve these equations:

$$2+2=x$$

$$11=3k+r$$

where k and r positive or null integers

$$27^{98} \bmod 97 = y$$

EXERCISE #3 ► **Strings**
Try the following code:

```
x = 'na'
'Ba' + 2 * x
```

Then write "j'aime les bons bonbons" with the same technique.

Lists

EXERCISE #4 ► **Lists**
Create a list `li` of integers containing various éléments. Replace one of the elements with a new value. At last, use + or += to add elements at the end of the list.

EXERCISE #5 ► **Sorts**
Sort a list using function `sorted`. What is the complexity in the worst case? In the best case? Use function `len()`; same questions.

Print

EXERCISE #6 ► **Formatting**
Give 3 different ways of building the following character string:
`"2.21 Gigawatts !! 2.21 Gigawatts !! My godness !"` using one variable `x = 2.21`, and another variable that uses `str()`, then the operator `%`, then the method `.format()`.

Tiny programs

Now, write your programs in `.py` files, starting with:

```
# -*- coding: utf-8 -*-
```

to avoid encoding issues

EXERCISE #7 ► **Hello**
Edit a file named `hello.py` with the following content:

```
# -*- coding: utf-8 -*-
print "Hello World"
```

Save, execute with: `python hello.py`.

EXERCISE #8 ► **If then else**
Write a program that initializes an int value to a number given by the user (use `input()`) and prints a different message according to its parity (odd/even).

EXERCISE #9 ► **While**
Write a program that declares two integer values `a` and `b`, then computes and prints their pgcd.

EXERCISE #10 ► **Imperative For**
Using the construction `for i in ...`, write a programs that sums all even *i* from 2 to 42

EXERCISE #11 ► **For expression / Lists**

- Write a program that declares and initialises a list, and computes the sum of all its elements.
- Write a 1-line code that, from a list `l`, returns a list whose elements are the squares of the elements in `l`.
- Write a 1-line code that, from a list `l`, returns a liste containing the even elements of `l.l`.

EXERCISE #12 ► **Dicts**

1. What are the types of `{}`, `{'a'}`, `{'a', 'b'}` and `{'a': 'b'}`?

2. What is the following code doing (where `t` is a dictionary):

```
while id in t:
        id = t[id]
print(id)
```

   What is the problem?

3. Write a code doing the same operation but without the same drawback (*i.e.*: if needed, it doesn't print anything)

EXERCISE #13 ▸ **Functions**

1. Declare a function `fact` that computes the factorial of a number.
2. What returns `help(fact)`? If it is not done, document your function.

1.2 The LC-3 processor, instruction set, simulator

EXERCISE #14 ▸ **Configuration**
To install and run PennSim, you can follow this simple guide up to step 4: `https://www.cis.upenn.edu/~milom/cse240-Fall06/pennsim/pennsim-guide.html`

In the architecture course (LIF6), you already saw a version of the target machine LC-3. The instruction set is depicted in Appendix A.

EXERCISE #15 ▸ **TD : skip if you are late**
Write a program in LC-3 assembly that writes the character 'Z' 10 times in the output.

EXERCISE #16 ▸ **Hand assembling**
Assemble by hand the two instructions :

```
begin:
  AND r0 r0 #0 ;
  BRp begin
```

You will need the set of instructions depicted in Appendix A and their associated opcode. Verify with Pennsim.

EXERCISE #17 ▸ **Run the simulator with the hex code**

Run the simulation step-by-step on the file `tp1-52.asm` :

Listing 1.1: tp1-52.asm

```
;; Author: Bill Slough for MAT 3670
;; Adapted by Laure Gonnord, oct 2014.
        .ORIG X3000    ; where to load the program in memory
        .FILL x5020
        .FILL x1221
        .FILL xE404
        .FILL x6681
        .FILL x1262
        .FILL x16FF
        .FILL x03FD
        .FILL xF025
        .FILL x0006
        .END
```

. Even if we have "assembled" the program by hand, we still need to use the command `as` in order to create the corresponding binary file `.obj`. Carefully follow each step of the execution. Note that the LC-3 simulator gives an equivalent in assembly language for each instruction.

Until now, we have written programs by putting the encoded instructions directly into the memory. From now on, we are going to write programs using an easier approach. We are going to write instructions using the LC-3 assembly.

EXERCISE #18 ▸ **Execution and modification**

1. Guess the purpose of the following files: `tp1-54a.asm` et `tp1-54b.asm`. Check with the simulator. What is the difference between the primitives PUTS and `OUT`, that are provided by the operating system?

Listing 1.2: tp1-54a.asm

```
;; Author: Bill Slough MAT 3670
;; Adapted by Laure Gonnord, oct 2014.
        .ORIG x3000     ; specify where to load the program in memory
        LEA R0,HELLO
        PUTS
        LEA R0,COURSE
        PUTS
        HALT
HELLO:  .STRINGZ "Hello, world!\n"
COURSE: .STRINGZ "LIF6\n"
        .END
```

Listing 1.3: tp1-54b.asm

```
;; Author: Bill Slough for MAT 3670
;; Adapted by Laure Gonnord, oct 2014.
        .ORIG x3000
        LD R1,N
        NOT R1,R1
        ADD R1,R1,#1   ; R1 = -N
        AND R2,R2,#0
LOOP:   ADD R3,R2,R1
        BRzp ELOOP
        LD R0,STAR
        OUT
        ADD R2,R2,#1
        BR LOOP
ELOOP:  LEA R0,NEWLN
        PUTS
STOP:   HALT
N:      .FILL 6
STAR:   .FILL x2A       ; the character to display
NEWLN:  .STRINGZ "\n"
        .END
```

2. Write a program in LC-3 assembly that computes the min and max of two integers, and store the result in a precise location of the memory that has the label `min`. Try with different values.

EXERCISE #19 ▶ **Algo in** LC-3 **assembly - Bonus**

Write and execute the following programs in assembly :

- Draw squares and triangles of stars (character '*') of size n, n being given by the user.
- Count the number of non-nul bits of a given integer.

Appendix A
LC3

A.1 Installing Pennsim and getting started

To install and use PennSim, read the following documentation :

`http://castle.eiu.edu/~mathcs/mat3670/index/Webview/pennsim-guide.html`

A.2 The LC3 architecture

Memory, Registers The LC-3 memory is shared into words of 16 bits, with address of size 16 bits (from $(0000)_H$ to $(FFFF)_H$).

The LC-3 has 8 main registers : R0, ..., R7. R6 is reserved for the execution stack handling, R7 for the routine return address. They are also specific 16 bits registers: PC (*Program Counter*), IR (*Instruction Register*), PSR (*Program Status Register*).

The PSR has 3 bits N,Z and P that indicate if the last value writen in one of the R0 to R7 registers (viewed as a 16bits 2-complement integer) is strictly negative (N), nul (Z) of strictly positive(P).

Instructions :

Syntax	**Action**	**NZP**
NOT DR,SR	DR <- not SR	*
ADD DR,SR1,SR2	DR <- SR1 + SR2	*
ADD DR,SR1,Imm5	DR <- SR1 + SEXT(Imm5)	*
AND DR,SR1,SR2	DR <- SR1 and SR2	*
AND DR,SR1,Imm5	DR <- SR1 and SEXT(Imm5)	*
LEA DR,label	DR <- PC + SEXT(PCoffset9)	*
LD DR,label	DR <- mem[PC + SEXT(PCoffset9)]	*
ST SR,label	mem[PC + SEXT(PCoffset9)] <- SR	
LDR DR,BaseR,Offset6	DR <- mem[BaseR + SEXT(Offset6)]	*
STR SR,BaseR,Offset6	mem[BaseR + SEXT(Offset6)] <- SR	
BR[n][z][p] label	Si (cond) PC <- PC + SEXT(PCoffset9)	
NOP	No Operation	
RET	PC <- R7	
JSR label	R7 <- PC; PC <- PC + SEXT(PCoffset11)	

Assembly directives

`.ORIG add`	Specifies the address where to put the instruction that follows
`.END`	Terminates a block of instructions
`.FILL val`	Reserves a 16-bits word and store the given value at this address
`.BLKW nb`	Reserves nb (consecutive) blocks of 16 bits at this address
`;`	Comments

Predefined interruptions `TRAP` gives a way to implement system calls, each of them is identified by a 8-bit constant. This is handled by the OS of the LC-3. The following macros indicate how to call them:

instruction	**macro**	**description**
`TRAP x00`	`HALT`	ends a program (give back decisions to OS)
`TRAP x20`	`GETC`	reads from the keyboard an ASCII char, and puts its value into R0
`TRAP x21`	`OUT`	writes on the screen the ASCII char of R0
`TRAP x22`	`PUTS`	writes on screen the string whose address of first caracter is stored in R0
`TRAP x23`	`IN`	reads from keyboard an ASCII char, outputs on screen and stores its value in R0

Constants : The integer constants encoded in hexadecimal are prefixed by `x`, in decimal by an optional `#` ; they can appear as parameters of the LC-3 instructions (immediate operands, be careful with the sizes) and directives like `.ORIG`, `.FILL` et `.BLKW`.

Coding tricks

- Initialisation to zero of a given register: `AND Ri,Ri,#0`
- Initialisation to a constant `n` (representable on 5 bits in complement to 2):
  `AND Ri,Ri,#0`
  `ADD Ri,Ri,n`
- Computation of the (integer) opposite $R_i \leftarrow (-R_j)$ (1+ complement to 2):
  `NOT Ri,Rj`
  `ADD Ri,Ri,#1`
- Multiplication $R_i \leftarrow 2R_j$: `ADD Ri,Rj,Rj`
- Copy $R_i \leftarrow R_j$: `ADD Ri,Rj,#0`

A.3 LC3 simplified instruction set

Here is a recap of instructions and their encoding:

syntaxe	**action**	**NZP**	**codage**															
			opcode				arguments											
			F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
NOT DR,SR	DR ← not SR	*	1 0 0 1				DR			SR			1 1 1 1 1 1					
ADD DR,SR1,SR2	DR ← SR1 + SR2	*	0 0 0 1				DR			SR1			0	0 0		SR2		
ADD DR,SR1,Imm5	DR ← SR1 + SEXT(Imm5)	*	0 0 0 1				DR			SR1			1	Imm5				
AND DR,SR1,SR2	DR ← SR1 and SR2	*	0 1 0 1				DR			SR1			0	0 0		SR2		
AND DR,SR1,Imm5	DR ← SR1 and SEXT(Imm5)	*	0 1 0 1				DR			SR1			1	Imm5				
LEA DR,label	DR ← PC + SEXT(PCoffset9)	*	1 1 1 0				DR			PCoffset9								
LD DR,label	DR ← mem[PC + SEXT(PCoffset9)]	*	0 0 1 0				DR			PCoffset9								
ST SR,label	mem[PC + SEXT(PCoffset9)] ← SR		0 0 1 1				SR			PCoffset9								
LDR DR,BaseR,Offset6	DR ← mem[BaseR + SEXT(Offset6)]	*	0 1 1 0				DR			BaseR			Offset6					
STR SR,BaseR,Offset6	mem[BaseR + SEXT(Offset6)] ← SR		0 1 1 1				SR			BaseR			Offset6					
BR[n][z][p] label	Si (cond) PC ← PC + SEXT(PCoffset9)		0 0 0 0				n	z	p	PCoffset9								
NOP	No Operation		0 0 0 0				0	0	0	0 0 0 0 0 0 0 0 0								
RET (JMP R7)	PC ← R7		1 1 0 0				0 0 0			1 1 1			0 0 0 0 0 0					
JSR label	R7 ← PC; PC ← PC + SEXT(PCoffset11)		0 1 0 0				1	PCoffset11										