Lab 1

Warm-up : discovering 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. All the material will be on the course webpage (bookmark now).

`http://laure.gonnord.org/pro/teaching/capM1.html`

Objective

- 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.

EXERCISE #1 ▶ **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`

1.1 The LC-3 processor and instruction set

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

EXERCISE #2 ▶ **TD**
On paper, write (in LC-3 assembly language) a program which initializes the $R0$ register to 1 and increments it until it becomes equal to 10.

EXERCISE #3 ▶ **TD : another one**
Write a program in LC-3 assembly that writes the character 'Z' 10 times in the output.

1.2 Assembling, disasembling

EXERCISE #4 ▶ **Hand assembling**
Assemble by hand the two instructions :

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

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

EXERCISE #5 ▶ **Hand disassembling**
In Figure 1.1 we depicted a toy example with its corresponding assembly code. Disasemble the two first instructions in the table. Then...

Fill the first two row of the table, read the rest of the solution, and answer the following questions:

- Which instruction is used to load data from memory?
- Could we do it another way?
- How is the pointer jumping done to create the loop?
- What happens to the labels in the disassemble program?

Address	Content	Binary	Instructions	pseudo-code
x3000	x5020			
x3001	x1221			
x3002	xE404	1110 010 0 0000 0100	LEA R2, Offset9=4	$R_2 \leftarrow x3007$ (label end)
x3003	x6681	010 011 010 00 0001	LDR R3, R2, 1	$R_3 \leftarrow mem[R2+1]$ (label of data $\rightarrow x3008$)
loop:x3004	x1262	0001 001 001 1 00010	ADD R1, R1, 2	$R_1 \leftarrow R_1 + 2$
x3005	x16FF	0001 011 011 1 11111	ADD R3, R3, -1	$R_3 \leftarrow R_3 - 1$
x3006	x03FD	0000 001 1 1111 1101	BRp Offset9=-3	if $R_3 > 0$ goto loop
end:x3007	xF025	1111 0000 0010 0101	TRAP x25	*HALT*
data:x3008	x0006	data	-	

Figure 1.1: A binary/hexadecimal program (`tp1-52.asm`)

1.3 Pennsim Simulator

EXERCISE #6 ► **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 #7 ► **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.

1.4 More advanced assembly code!

EXERCISE #8 ▶ **Algo in** LC-3 **assembly**

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										