# Compilation and Program Analysis (#1b) : recalling LC-3 architecture

Laure Gonnord

http://laure.gonnord.org/pro/teaching/

Laure.Gonnord@ens-lyon.fr

Master 1, ENS de Lyon

sept 2016

Lyon 1

ENS DE LYON

1. The LC-3 architecture in a nutshell

2. One example

1 The LC-3 architecture in a nutshell

2 One example

# Our target machine : LC-3

- memory : $2^{16}$ 16-bits words.
- instructions are also encoded in 16-bits words.

We have a simulator (Pennsim, see lab)

Registers **PC**, **IR**, **PSR** (*Program Status Register*) + 8 general purpose registers **R0,...,R7**.

# LC-3 ISA

See companion document.

# Example : ADD instruction

- `ADD DR, SR1, SR2`, does `DR <- SR1 + SR2`.
  - ↪ All operands are registers.
  - ↪ Example : `ADD R1, R2, R3` executes `R1 <- R2 + R3`.
- `ADD DR, SR1, imm5`, does `DR <- SR + imm5`.
  - ↪ The last operand is an immediate value encoded in the instruction.
  - ↪ Example : `ADD R1, R2, 5` executes `R1 <- R2 + 5`.

# LC-3 ADD : encoding

A bit specifies the addressing mode :

| **assembly** | **action** | **code** | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | opcode | | | | arguments | | | | | | | | | | | |
| | | F | E | D | C | B | A | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ADD DR,SR1,SR2 | DR <- SR1 + SR2 | 0 0 0 1 | | | | DR | | | SR1 | | | 0 | 0 0 | | SR2 | | |
| ADD DR,SR1,Imm5 | DR <- SR1 + Imm5 | 0 0 0 1 | | | | DR | | | SR1 | | | 1 | Imm5 | | | | |

# LC-3 Memory access instructions

- `LD DR, add`, does `DR <- mem[add]`.
- `ST SR, add`, does `mem[add] <- SR`.

▶ **Direct addressing** : `add` is an address, `mem[add]` the associated memory cell.
This adress is encoded in the instruction :

| **assembly** | **action** | **encoding** | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | opcode | | | | arguments | | | | | | | | | | | |
| | | F | E | D | C | B | A | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| LD DR,add | DR <- mem[add] | 0 0 1 0 | | | | DR | | | add | | | | | | | | |
| ST SR,add | mem[add] <- SR | 0 0 1 1 | | | | SR | | | add | | | | | | | | |

# LC-3 : branching

**Inconditional branching** :

- `BR add`, does `PC <- add`.

`add` designs the memory address of the instruction that will be executed after the current instruction (direct addressing mode)

| assembly | action | encoding | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | opcode | | | | arguments | | | | | | | | | | | |
| | | F | E | D | C | B | A | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| BR add | PC <- add | 0 0 0 0 | | | | 0 0 0 | | | add | | | | | | | | |

1. The LC-3 architecture in a nutshell
2. One example

# Ex : Assembly code - demo

```
        .ORIG x3000     ; address in memory
;program instructions
        LD R1,a         ;
                        ;
        LD R2,b         ;
                        ;
        ADD R0,R1,R2    ;
                        ;
        ST R0,r         ;
                        ;
        HALT            ; flow back to OS
; data
a:      .FILL #5        ; 5 is stored at this address
b:      .FILL #2        ;
r:      .BLKW #1        ; reservation of a memory cell
        .END
```

# LC-3 Exercise : array sum

```
        .ORIG x3000      ; address in memory
; TODO : sum all elements from beg to end










          .HALT
; data
beg:      .FILL #12
          .FILL #45
          .FILL #16
end:      .FILL #2        ;
r:        .BLKW #1        ; result here!
          .END
```