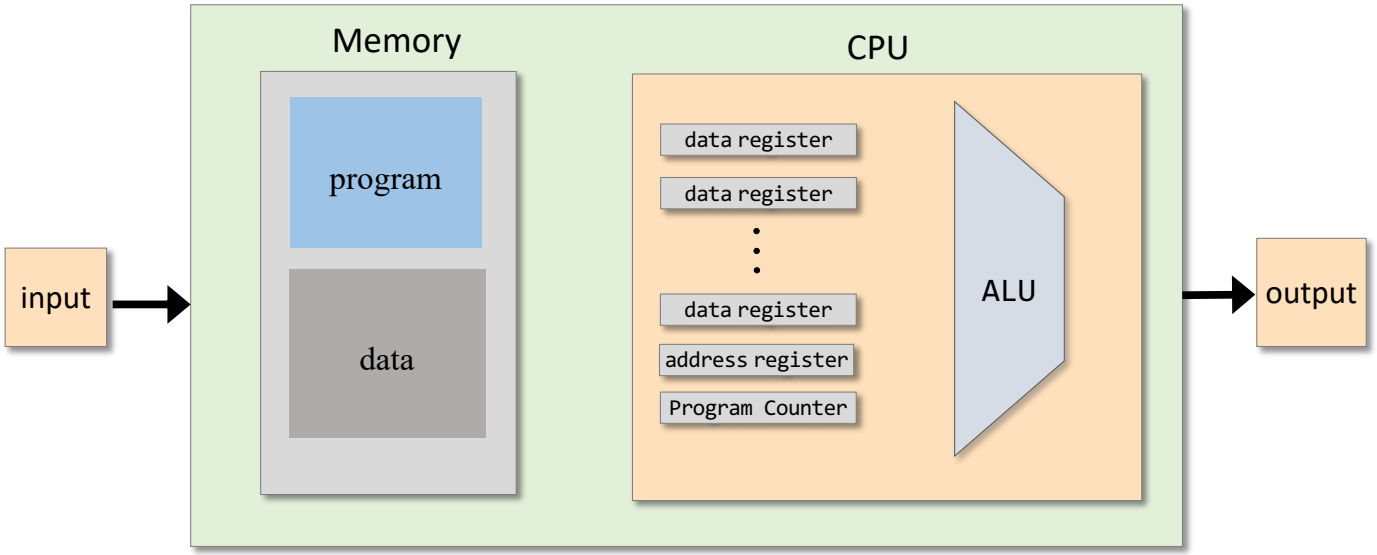
5. Langage machine



Instructions type d'un langage machine

```
// In what follows R1,R2,R3 are registers, PC is program counter,
// and addr is some value.

ADD R1,R2,R3      // R1 ← R2 + R3

ADDI R1,R2,addr   // R1 ← R2 + addr

AND R1,R1,R2      // R1 ← R1 and R2 (bit-wise)

JMP addr          // PC ← addr

JEQ R1,R2,addr    // IF R1 == R2 THEN PC ← addr ELSE PC++

LOAD R1, addr     // R1 ← RAM[addr]

STORE R1, addr    // RAM[addr] ← R1

NOP               // Do nothing

// Etc. – some 50-300 command variants
```

Caractéristiques de la Hack machine

- **Mémoire pour les données : RAM (16 bits)**
- **Mémoire pour les instructions : ROM (16 bits)**
- **Registres : A, D, M avec M=RAM[A]**
- **Opérations : ALU**
- **Compteur ordinal : PC**
- **Jeu d'instruction : A-instruction, C-instruction**

Hack langage

A instruction

Symbolic: `@`*xxx* (*xxx* is a decimal value ranging from 0 to 32767, or a symbol bound to such a decimal value)

Binary: `0` *vvvvvvvvvvvvvvv* (*vv ... v* = 15-bit value of *xxx*)

C instruction

Symbolic: *dest* `=` *comp* `;` *jump* (*comp* is mandatory. If *dest* is empty, the `=` is omitted; If *jump* is empty, the `;` is omitted)

Binary: `111`*accccccdddjjj*

comp		*c*	*c*	*c*	*c*	*c*	*c*
`0`		1	0	1	0	1	0
`1`		1	1	1	1	1	1
`-1`		1	1	1	0	1	0
`D`		0	0	1	1	0	0
`A`	`M`	1	1	0	0	0	0
`!D`		0	0	1	1	0	1
`!A`	`!M`	1	1	0	0	0	1
`-D`		0	0	1	1	1	1
`-A`	`-M`	1	1	0	0	1	1
`D+1`		0	1	1	1	1	1
`A+1`	`M+1`	1	1	0	1	1	1
`D-1`		0	0	1	1	1	0
`A-1`	`M-1`	1	1	0	0	1	0
`D+A`	`D+M`	0	0	0	0	1	0
`D-A`	`D-M`	0	1	0	0	1	1
`A-D`	`M-D`	0	0	0	1	1	1
`D&A`	`D&M`	0	0	0	0	0	0
`D|A`	`D|M`	0	1	0	1	0	1

a `==0` *a* `==1`

dest	*d*	*d*	*d*	Effect: store *comp* in:
`null`	0	0	0	the value is not stored
`M`	0	0	1	`RAM[A]`
`D`	0	1	0	`D` register (reg)
`DM`	0	1	1	`RAM[A]` and `D` reg
`A`	1	0	0	`A` reg
`AM`	1	0	1	`A` reg and `RAM[A]`
`AD`	1	1	0	`A` reg and `D` reg
`ADM`	1	1	1	`A` reg, `D` reg, and `RAM[A]`

jump	*j*	*j*	*j*	Effect:
`null`	0	0	0	no jump
`JGT`	0	0	1	if *comp* > 0 jump
`JEQ`	0	1	0	if *comp* = 0 jump
`JGE`	0	1	1	if *comp* ≥ 0 jump
`JLT`	1	0	0	if *comp* < 0 jump
`JNE`	1	0	1	if *comp* ≠ 0 jump
`JLE`	1	1	0	if *comp* ≤ 0 jump
`JMP`	1	1	1	unconditional jump

A-instruction

```
@value          // A ← value
```

où value est un nombre ou une référence symbolique à un nombre.

Entrer une constante	@17 // A=17
	D=A // D=17

Sélectionner une donnée dans la RAM	@17 // A=17
	D=M // D=RAM[17]

Sélectionner une instruction dans la ROM	@17 // A=17 JMP // PC=17

C-instructions

```
dest = x + y
dest = x - y
dest = x
dest = 0
dest = 1
dest = -1
```

x = {A, D, M}

y = {A, D, M , 1}

dest = {A, D, M, MD, A, AM, AD, AMD, null}

Exercices

- Set **D** to **A-1**
- Set both **A** and **D** to **A + 1**
- Set **D** to **19**
- Set both **A** and **D** to **A + D**
- Set **RAM[5034]** to **D - 1**
- Set **RAM[53]** to **171**
- Add **1** to **RAM[7]**, and store the result in **D**.

C-instructions (suite)

```
dest = x + y
dest = x - y
dest = x
dest = 0
dest = 1
dest = -1
```

x = {A, D, M}

y = {A, D, M , 1}

dest = {A, D, M, MD, A, AM, AD, AMD, null}

Exercices

- `sum = 0`
- `j = j + 1`
- `q = sum + 12 – j`
- `arr[3] = -1`
- `arr[j] = 0`
- `arr[j] = 17`

Symbol table:

j	3012
sum	4500
q	3812
arr	20561

C-instructions (suite)

C-command: dest = comp ; jump // dest = and ;jump
// are optional

Where:

comp = 0 , 1 , -1 , D , A , !D , !A , -D , -A , D+1 ,
A+1 , D-1, A-1 , D+A , D-A , A-D , D&A ,
D|A , M , !M , -M ,M+1, M-1 , D+M , D-M ,
M-D , D&M , D|M

dest = M , D , MD , A , AM , AD , AMD, or null

jump = JGT , JEQ , JGE , JLT , JNE , JLE , JMP, or null

Symbol table:

sum	2200
x	4000
i	6151
END	50
NEXT	120

Exercices

- goto 50
- if D==0 goto 112
- if D<9 goto 507
- if RAM[12] > 0 goto 50
- if sum>0 goto END
- if x[i]<=0 goto NEXT.

Instruction if

High level:

```
if condition {
   code block 1}
else {
   code block 2}
code block 3
```

Hack:

```
   D ← not condition
   @IF_TRUE
   D;JEQ
   code block 2
   @END
   0;JMP
(IF_TRUE)
   code block 1
(END)
    code block 3
```

Hack convention:

- True is represented by -1
- False is represented by 0

Instruction while

High level:

```
while condition {
    code block 1
}
Code block 2
```

Hack:

```
(LOOP)
      D ← not condition)
      @END
      D;JEQ
      code block 1
      @LOOP
      0;JMP
(END)
      code block 2
```

Hack convention:

- True is represented by -1
- False is represented by 0

Exercice

Traduire le programme suivant pour une Hack Machine

C language code:

```
// Adds 1+...+100.
 into i = 1;
 into sum = 0;
 while (i <= 100){
    sum += i;
    i++;
 }
```

Hack assembly code:

```
// Adds 1+...+100.
       @i      // i refers to some RAM location
       M=1     // i=1
       @sum    // sum refers to some RAM location
       M=0     // sum=0
```

Questions