Architecture matérielle et logicielle : introduction.

Un ordinateur ne peut exécuter qu'un nombre resteint d'*instructions* :

- additionner deux nombres,
- tester l'égalité d'un nombre à zéro,
- copier des données d'une zone de la mémoire à une autre, ...

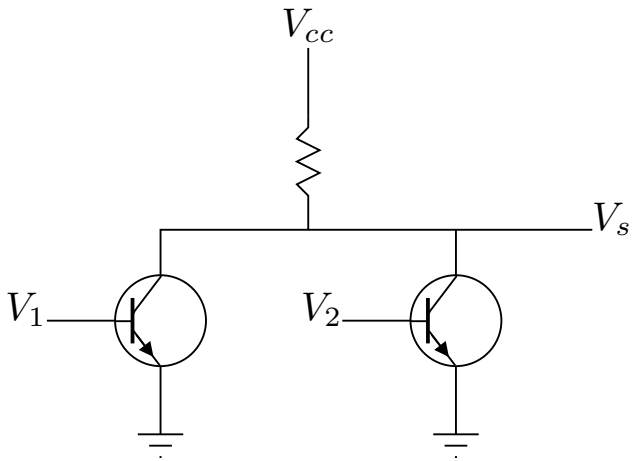
L'ensemble de ces instructions de base forme un langage appelé *langage machine*, qui permet de programmer l'ordinateur : on appelle *programme* la séquence d'instructions décrivant la réalisation d'un certain traitement.

Le langage machine est si élémentaire qu'il est fastidieux de programmer ainsi. Pour faciliter la tâche du programmeur, des niveaux d'abstraction s'empilent donc au dessus du matériel : on parle d'*organisation par niveaux de l'ordinateur*.

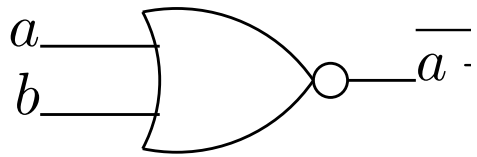
Niveau 0 : circuits logiques

Ce niveau comprend les *portes logiques* qui permettent à l'ordinateur d'effectuer des traitements de données élémentaires. Ces portes logiques sont aujourd'hui réalisées à l'aide de transistors.

Chaque porte présente des entrées prenant comme valeur 0 ou 1, et applique à ces entrées une fonction logique simple : NOT, AND, OR... Les ordinateurs manipulent donc de façon privilégiée des informations codées en binaire.



V_1	V_2	V_s
0	0	1
0	1	0
1	0	0
1	1	0



Niveau 1 : micro-architecture

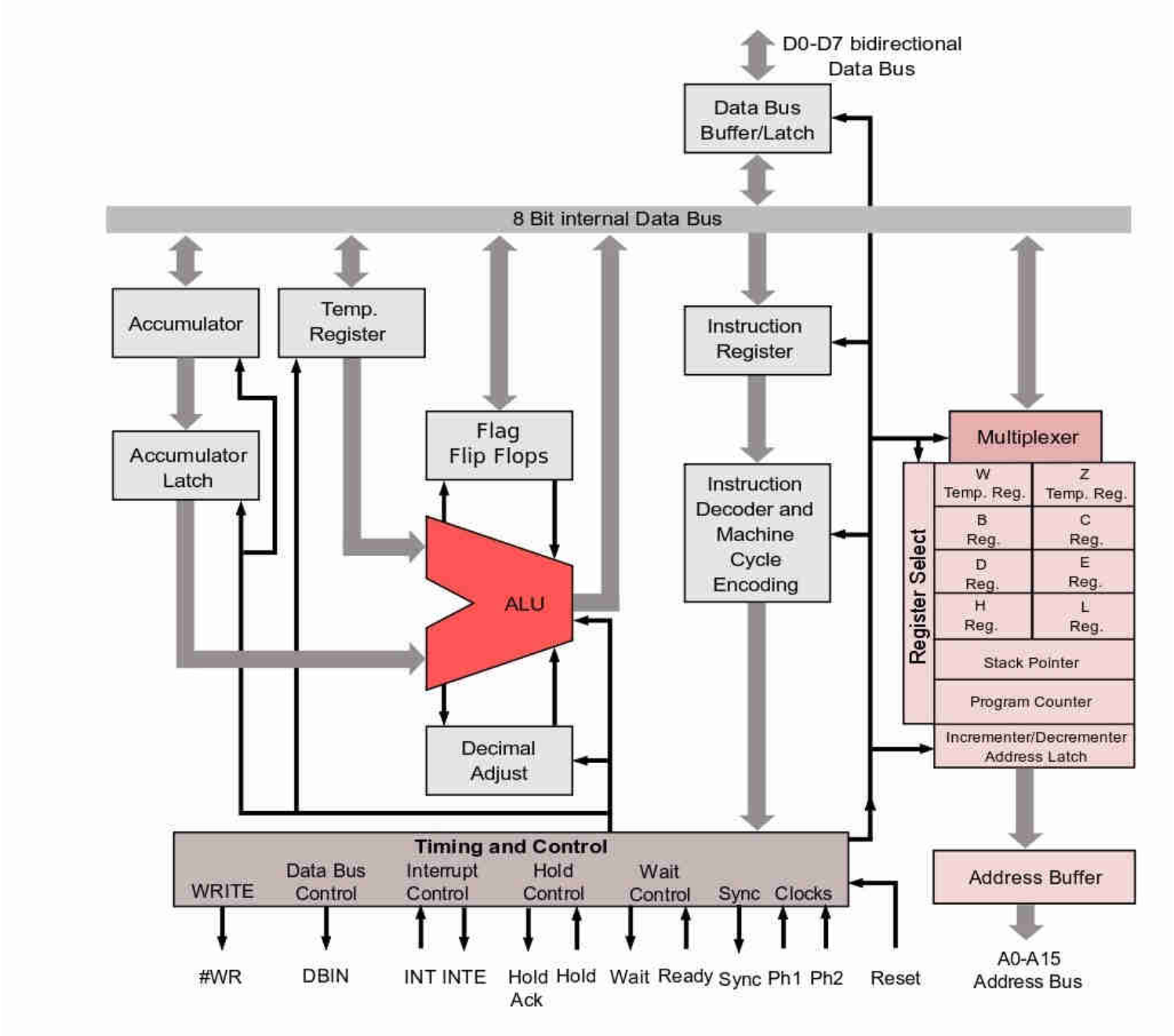
Avec les portes logiques, on peut concevoir des unités de traitement évoluées.

Chaque ordinateur dispose d'un *processeur* (appellé aussi *unité centrale de traitement*), qui comprend de nombreux circuits, dont :

- une *unité arithmétique et logique* qui effectue les opérations de base,
- une *unité de contrôle*, chargée de gérer l'exécution des opérations élémentaires dans un certain ordre.

Le *niveau micro-architecture* comprend tout le matériel chargé de l'exécution effective des instructions du langage machine.

Ex : diagramme repésentant la micro-architecture du processeur Intel 8080 :



Niveau 2 : architecture

L'*architecture* décrit le principe général de fonctionnement de l'ordinateur, pour permettre à un programmeur de l'utiliser au travers du langage machine.

Définir une architecture, c'est définir tous les attributs d'un ordinateur qui sont visibles au programmeur lorsqu'il programme en langage machine.

Citons quatre attributs définissant une architecture :

- le jeu d'instructions qui compose le langage machine,
- les registres que le programmeur peut utiliser,
- la manière dont la mémoire de l'ordinateur est organisée,
- les types de données élémentaires (entiers, nombres flottants).

On parle aussi d'*ISA* pour *Instruction Set Architecture*.

Ex : jeu d'instructions de l'architecture LC3 :

instruction	**action**	**nzp**	**codage en langage machine**															
			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					
LDI DR,label	DR <- mem[mem[PC + SEXT(PCoffset9)]]	*	1 0 1 0				DR			PCoffset9								
STI SR,label	mem[mem[PC + SEXT(PCoffset9)]] <- SR		1 0 1 1				SR			PCoffset9								
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								
JMP BaseR	PC <- BaseR		1 1 0 0				0 0 0			BaseR			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										
JSRR BaseR	R7 <- PC ; PC <- BaseR		0 1 0 0				0	0 0		BaseR			0 0 0 0 0 0					
RTI	cf. interruptions		1 0 0 0				0 0 0 0 0 0 0 0 0 0 0 0											
TRAP Trapvect8	R7 <- PC ; PC <- mem[Trapvect8]		1 1 1 1				0 0 0 0						Trapvect8					
Réservé			1 1 0 1															

Niveau 3 : système d'exploitation

Le *système d'exploitation* est un programme qui introduit un niveau d'abstraction entre le programmeur (ou plus généralement l'utilisateur d'une machine), et le matériel.

Il fournit au programmeur une *interface de gestion du matériel* (mémoire, périphériques, exécution des programmes...) identique sur de nombreuses architectures différentes.

Ex : *Linux* a été adapté à de nombreuses architectures : ARM, x86, Power...

Si un programmeur écrit un programme qui réalise des accès au disque dur en utilisant les *primitives* fournies par Linux, ce programme fonctionnera correctement sur ARM, x86 ou PowerPC.

Niveau 4 : langage d'assemblage

Ce niveau fournit un langage intermédiaire permettant au programmeur de coder plus facilement qu'en langage machine, appelé *langage d'assemblage*. En effet, les instructions du langage machine ne sont que des suites de bits, facilement exécutables par le processeur, mais illisibles pour le programmeur.

Les langages d'assemblage peuvent également fournir des facilités pour l'interaction avec le système d'exploitation. Les programmes écrits en langage d'assemblage sont traduits en langages de niveaux 2 et 3 à l'aide d'un programme appelé *assembleur*.

Niveau 5 : langage de programmation

On trouve à ce niveau les *langages de programmation de haut niveau*, tel le C, le C++ ou le Caml. Ces langages fournissent un haut niveau d'abstraction, et permettent aux programmeurs de réaliser facilement des logiciels complexes.

Les programmes codés dans ces langages de haut niveau sont traduits vers les niveaux 3 (système d'exploitation) et 4 (langage d'assemblage) par des *compilateurs* (par exemple GCC, ICC ou CLANG pour le C/C++).

Prenons un exemple pour illustrer tout cela...
On considère par exemple un programme écrit en C, un *langage de haut niveau* :

```
#include <stdio.h>
char car;
int main(void) {
  printf("Hi!\n");                    // appel à une primitive de l'OS
  printf("Entrez un caractere...\n");
  car = getchar();                    // appel à une primitive de l'OS
  printf("Vous avez entre : ");
  putchar(car);                       // appel à une primitive de l'OS
  putchar('\n');
  printf("Bye!\n");
  return(0);
}
```

Le programme traduit dans le *langage d'assemblage* du LC3 par un *compilateur* :

```
        LEA R0,msg0     ; charge l'adresse effective désignée par msg0 dans R0
        TRAP x22        ; affiche la chaine pointée par R0
        LEA R0,msg1     ;
        TRAP x22        ; affiche la chaine à l'adresse msg1
        TRAP x20        ; lit un caractère et le place dans R0
        ST R0,car       ; stocke le caractère lu à l'adresse car
        LEA R0,msg2     ;
        TRAP x22        ; affiche la chaine à l'adresse msg2
        LD R0,car       ; charge le caractère stocké à l'adresse car dans R0
        TRAP x21        ; affiche le caractère qui a été lu
        LEA R0,ret      ;
        TRAP x22        ; affiche un retour à la ligne
        LEA R0,msg3     ;
        TRAP x22        ; affiche la chaine à l'adresse msg3
        TRAP x25        ; termine le programme (rend la main à l'OS)
car:    .BLKW #1        ; case mémoire pour stocker un caractère lu
msg0:   .STRINGZ "Hi!\n"
msg1:   .STRINGZ "Entrez un caractere...\n"
msg2:   .STRINGZ "Vous avez entre : "
msg3:   .STRINGZ "Bye!\n"
ret:    .STRINGZ "\n"
```

Le programme traduit en *langage machine* par l'*assembleur* :

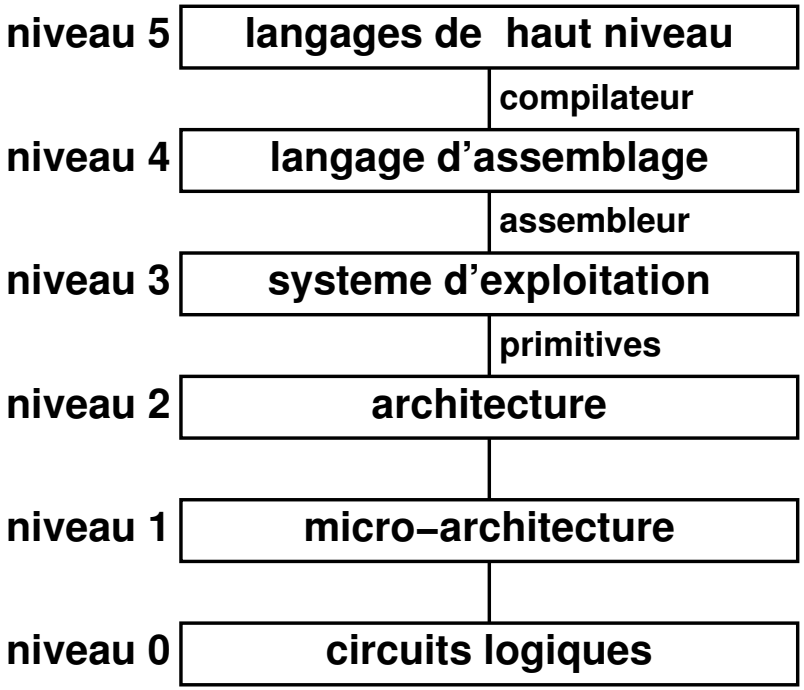
```
E00F F022 E012 F022 F020 3009 E026 F022 2006 ...
```

On a la correspondance suivante pour chaque instruction :

```
langage    langage
machine  d'assemblage

xE00F    LEA R0,msg0
xF022    TRAP x22
xE012    LEA R0,msg1
xF022    TRAP x22
xF020    TRAP x20
x3009    ST R0,car
xE026    LEA R0,msg2
xF022    TRAP x22
x2006    LD R0,car
...
```

Nous sommes descendus du niveau *langage de haut niveau* au niveau *architecture* : les niveaux *micro-architecture* et *circuits logiques* de l'ordinateur ciblé permettront ensuite l'exécution effective du programme.



La compréhension de chacun de ces niveaux permet de maîtriser la complexité de l'ordinateur :)

- On commencera par présenter une vue d'ensemble de l'ordinateur, en se reposant sur le modèle de Von Neumann.
- On repartira ensuite du niveau « circuits logiques » pour remonter jusqu'au niveau « langage d'assemblage. »

Conclusion

Les ordinateurs permettent de couvrir un très large spectre d'applications, et certains offrent une puissance de calcul considérable.

Des interfaces ont été développées pour faciliter l'utilisation des ordinateurs. D'autre part, le modèle « par niveaux » permet au programmeur d'effectuer son travail sans se préoccuper des détails architecturaux de sa machine...

Pourquoi donc étudier l'architecture des ordinateurs ?

Au delà de l'augmentation du nombre de transistors intégrés par processeur, et celle de la fréquence des processeurs, de nombreuses innovations architecturales ont été mises en place *afin d'améliorer la performance des machines.*

Le programmeur doit être capable de comprendre les caractéristiques de sa machine *afin de concevoir des programmes plus performants*.