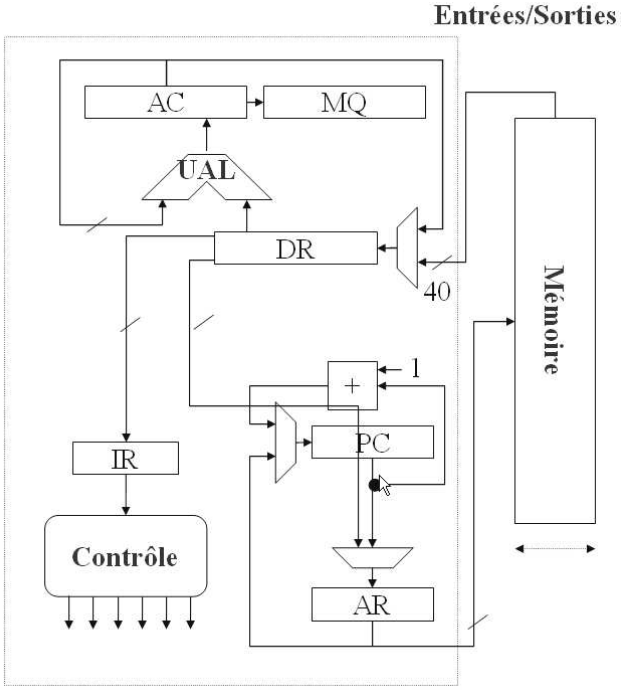
7 Les processeurs

La structure des processeurs actuels est le résultat de la convergence progressive de plusieurs domaines :

- **machines à calculer** ; machine de Pascal (addition/soustraction mécanique), Leibniz (machines mécaniques réalisant les quatre opérations élémentaires), Babbage (calculateur mécanique pour effectuer des calculs itératifs, puis proposition d'une machine générale permettant d'effectuer un enchaînement quelconque d'opérations arithmétiques logiques)…
- **technologie** ; représentation et modification d'un état de façon mécanique (e.g., roue de Pascal), de façon électrique (interrupteur à commande mécanique), de façon électronique (interrupteur à commande électrique : tube à vide, puis transistor)
- **logique** ; travaux de Boole, de Shannon et de Turing…
- **commande automatique** (Jacquard, Hollerith…)

Ces travaux ont progressivement fait émerger la notion d'**ordinateur**, une machine «universelle» capable d'exécuter un algorithme quelconque, puis ils ont permis de réaliser les premiers prototypes vers la fin des années 1940. La structure des ordinateurs a rapidement évolué vers ce que l'on nomme aujourd'hui le **modèle de Von Neumann** :

- l'ordinateur est composé d'un coeur de calcul, d'une mémoire, d'une unité de contrôle et d'entrées/sorties,
- l'exécution d'un algorithme se décompose en instructions élémentaires qui commandent le fonctionnement de l'ordinateur ; cet ensemble d'instructions forme le **programme** ; le programme guide donc l'exécution de l'ordinateur,
- les données et le programme sont stockés dans la mémoire (proposer de stocker le programme en mémoire est la principale contribution de Von Neumann).



Sur les premiers processeurs comme sur les processeurs actuels, les étapes de l'exécution d'une instruction d'un programme se décomposent (schématiquement) de la façon suivante :

- le processeur dispose d'un registre appelé PC (*Program Counter*) qui contient l'adresse en mémoire de la prochaine instruction à exécuter ; le processeur envoie donc cette adresse à la mémoire,
- la mémoire renvoie l'instruction au processeur,
- l'unité de contrôle du processeur analyse l'instruction à exécuter : elle détermine ses opérandes et le calcul à effectuer,
- l'unité de contrôle récupère les opérandes de l'instruction,
- le processeur effectue le calcul correspondant,
- l'unité de contrôle stocke le résultat, et passe à l'instruction suivante du programme.

Rapidement, les processeurs adoptent plusieurs caractéristiques communes : codage binaire, décomposition de la mémoire en octets (8 bits), calculs entiers en complément à 2... Si les premiers processeurs ont une structure très simple, l'évolution rapide de la technologie (notamment la densité des transistors sur une puce qui double tous les 18 à 24 mois) engendre des architectures de plus en plus complexes : nombreuses instructions, longueur variable des instructions, nombreuses façons d'accéder à une donnée en mémoire (modes d'adressage), nombreux types de données... Au début des années 1980, on constate que cette évolution a les conséquences suivantes :

- les processeurs et en particulier leurs circuits de contrôle deviennent très complexes, et les temps de développement s'allongent ; en conséquence, les constructeurs de processeurs ne sont plus capables de faire évoluer leurs architectures suffisamment rapidement pour prendre en compte l'évolution de la technologie ;
- en raison du nombre croissant d'instructions, la complexité des compilateurs augmente également, allongeant encore le temps de développement ; des chercheurs constatent qu'en pratique, un faible pourcentage des instructions disponibles est effectivement utilisé dans les programmes.

Ces chercheurs proposent de revoir entièrement l'architecture des processeurs, et de la simplifier considérablement :

- utiliser peu d'instructions, peu de modes d'adressage, peu de formats de données ;
- utiliser des instructions de taille fixe ;
- placer les opérandes des instructions dans des registres uniquement (et non en mémoire) pour un accès rapide à l'intérieur de la puce du processeur ;
- utiliser un contrôle câblé plus complexe mais plus rapide que le contrôle microprogrammé introduit dans plusieurs processeurs.

Les conséquences de ces transformations sont les suivantes :

- ces processeurs peuvent être construits plus rapidement parce que leur architecture est plus simple, et ils peuvent bénéficier donc plus rapidement de l'évolution de la technologie ;
- grâce à une utilisation d'instructions simples, et à une segmentation de toutes les instructions en nombreuses étapes élémentaires, le temps de cycle du processeur peut être bas (fréquence élevée), et le débit des instructions devient plus élevé ;
- en raison du faible nombre d'instructions, certains aspects de la compilation sont simplifiés.

Ces processeurs sont appelés des processeurs **RISC** (*Reduced Instruction Set Computer*) par opposition aux processeurs de l'époque baptisés CISC par la suite *Complex Instruction Set Computer*). En raison des progrès en performance qu'ils permettent de réaliser, ces architectures deviennent très populaires dès la fin des années 80. L'architecture de la plupart des processeurs actuels repose sur les principes des processeurs RISC, même si ce modèle a

dû être considérablement altéré depuis pour réaliser des gains supplémentaires en performance. Dans les sections suivantes, nous introduisons un exemple de processeur RISC simple (le LC-2) proposé dans [6] à titre pédagogique.

7.1 Jeu d'instructions

La première étape de la conception d'un processeur est la définition de son jeu d'instructions. Le jeu d'instructions décrit l'ensemble des opérations élémentaires que le processeur pourra exécuter. Il va donc déterminer en partie l'architecture du processeur à réaliser (en anglais, le jeu d'instructions est appelé un ISA pour *Instruction Set **Architecture***). Cependant, à un même jeu d'instructions peut correspondre un grand nombre d'implémentations différentes du processeur (par exemple, le jeu d'instructions x86 et les processeurs d'Intel).
Pour un processeur de type Von Neumann, le jeu d'instructions doit permettre de réaliser les fonctions suivantes :

- des opérations de calcul arithmétique logique,
- l'écriture et la lecture des données en mémoire,
- le contrôle de l'exécution du programme (tests, branchements).

La première étape de la conception du processeur consiste à déterminer la taille du mot et des instructions (ici, nous utiliserons la même taille pour le mot et les instructions). La taille du mot détermine notamment l'intervalle des nombres qui pourront être représentés, la taille des adresses et donc de l'espace mémoire qui pourra être accédé, et dans une certaine mesure, la performance du processeur (plus le nombre de bits que l'on peut traiter en une seule opération est grand, moins il faut d'opérations pour certains calculs). La taille des instructions détermine notamment le nombre d'instructions, le nombre de registres du processeur, et la taille de certains opérandes.
La taille du mot et des instructions est essentiellement contrainte par le coût : la taille et donc le coût des opérateurs arithmétiques et logiques, des chemins de données, des registres sont corrélés à la taille du mot et des instructions. Le premier microprocesseur utilisait un mot de 4 bits, les processeurs actuels utilisent des mots de 32 bits, et plusieurs processeurs utilisant des mots de 64 bits sont apparus depuis quelques années. Pour réaliser un compromis raisonnable entre coût et fonctionnalités, on a fixé la taille du mot du processeur LC-2 à 16 bits.

Il faut maintenant déterminer plus précisément la structure des instructions du LC-2 ; essentiellement, chaque instruction doit comporter deux parties : l'une indiquant l'instruction à effectuer (le code opération ou *opcode*), l'autre indiquant les opérandes de l'instruction, voir figure ci-dessous.

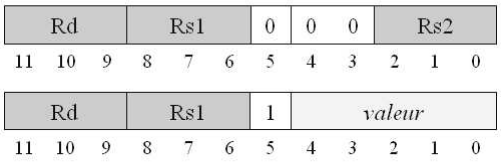
Comme le LC-2 est un processeur RISC, les opérandes doivent être en général contenus dans des registres (et non en mémoire) pour un accès rapide. Il faut donc trois registres pour chaque instruction arithmétique et logique : deux registres sources, et un registre destination. Si le jeu d'instructions contient 2^k instructions[1], il ne reste que $16-k$ bits dans l'instruction pour spécifier les numéros des trois registres, soit $\frac{16-k}{3}$ bits par registre, ce qui signifie que le processeur peut contenir au plus $2^{\frac{16-k}{3}}$ registres. Or, plus le nombre de registres est important, moins il

[1] Le nombre d'instructions n'est pas nécessairement une puissance de 2.

est nécessaire d'accéder à la mémoire, et donc plus la performance est élevée. Il y a donc un compromis à réaliser entre le nombre de registres et le nombre d'instructions. Pour le LC-2, bien que k=4, on n'utilise que 8 registres parce que l'un des bits des opérandes est parfois utilisé pour compléter l'opcode, voir ci-dessous.
On va maintenant construire un jeu d'instructions très simple pour le LC-2, et préciser le format de chaque instruction.

Instructions arithmétiques et logiques. On suppose que les calculs sont effectués en complément à 2. Au minimum, on a besoin de l'opération d'addition (instruction ADD). On a vu que les opérateurs logiques ET et NON permettent de réaliser n'importe quelle fonction logique, et on les ajoute donc au jeu d'instructions (AND, NOT) ; on aurait pu n'ajouter que l'opérateur NON-ET, mais l'opérateur NOT a l'avantage de permettre le calcul de l'opposé d'un nombre en complément à 2.
La pratique montre que l'un des opérandes d'un calcul est fréquemment une valeur simple (0, 1, 2...) ; pour éviter d'avoir à charger fréquemment de telles valeurs depuis la mémoire pour les placer dans un registre, on peut directement coder de telles valeurs dans l'instruction. L'instruction n'utilise plus alors un registre, mais une valeur, on parle d'**adressage immédiat** par opposition à l'adressage **direct** lorsque l'opérande est dans un registre. Outre l'opcode, on a alors besoin d'un bit pour spécifier si l'adressage est immédiat ou direct ; ce bit est interprété par le circuit de contrôle et change la manière dont les opérandes sont récupérés.
Pour ces instructions, on obtient alors le format suivant dans le LC-2 :

• ADD, AND:



• NOT:

- ADD Rd←Rs1,Rs2
  - Rd←Rs1+Rs2
- AND Rd←Rs1,Rs2
  - Rd←ET(Rs1,Rs2)
- ADD Rd←Rs,valeur
  - Rd←Rs+valeur
- AND Rd←Rs,valeur
  - Rd←ET(Rs,valeur)
- NOT Rd←Rs
  - Rd←NOT(Rs)

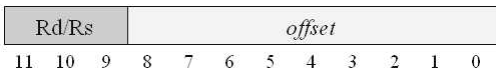
On considérera que les bits d'opcode ont été associés arbitrairement aux instructions de la façon suivante :

Opcode	**Instruction**
0001	**ADD**
0101	**AND**
1001	**NOT**

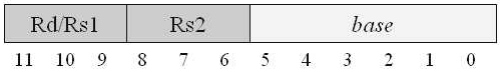
Instructions mémoire. Comme le LC-2 est un processeur RISC, les opérandes sont généralement contenus dans des registres, ce qui signifie qu'il faut des instructions spéciales pour accéder à la mémoire : une instruction de chargement (*load*), est une instruction de rangement (*store*) ; aussi, on parle également d'architectures *load-store*. Ici, on notera LD l'instruction de chargement, et ST l'instruction de rangement.

Il existe plusieurs façons de spécifier une adresse en mémoire. La plus simple consiste à spécifier cette adresse directement dans l'instruction, ce qui est analogue à l'adressage immédiat des opérations arithmétiques logiques. Cependant, comme l'opcode utilise 4 des 16 bits de l'instruction et que le registre source ou destination en utilise également 3, il ne reste plus que 9 bits pour spécifier l'adresse ; or le mot de 16 bits permet en théorie d'accéder à 2^{16} octets. Le LC-2 propose de contourner ce problème en prenant les neuf bits de poids fort du PC ; cela signifie que l'adressage a lieu dans le voisinage de l'instruction[2]. Il faut noter que ce mode d'adressage est incorrectement baptisé **direct** dans le LC-2.
Une façon d'étendre l'espace d'adressage est d'utiliser à la fois un registre et une valeur spécifiée dans l'instruction : l'adresse est obtenue en ajoutant registre et valeur. Le registre permet d'utiliser 16 bits pour spécifier une adresse. Le fait de coupler registre et valeur permet de faciliter l'accès à certaines structures de données classiques : pour un tableau, par exemple, on peut imaginer que l'adresse de base du tableau soit stockée dans l'instruction (la valeur), et que le registre soit progressivement incrémenté pour balayer chacune des cases du tableau. On parle d'adressage **indexé**.
Le format des instructions correspondantes est indiqué ci-dessous :

- LD/ST:



- LDR/STR:



- LD Rd←*offset*
- ST Rs→*offset*
  - Adressage direct
  - Adresse = PC[15:9],offset[8:0]
- LDR Rd←Rs,*base*
- STR Rs2→Rs1,*base*
  - Adressage indexé
  - Adresse = Rs1 + Base
- LEA Rd←*offset*
  - Adressage immédiat
  - Rd←PC[15:9],offset[8:0]

On peut remarquer que pour différencier le mode d'adressage des instructions mémoire, le LC-2 utilise l'opcode plutôt qu'un bit dans le champ opérandes, comme pour les instructions arithmétiques et logiques. En fait, il est plus naturel d'utiliser l'opcode pour spécifier le mode d'adressage ; cependant, dans le LC-2 toutes les valeurs possibles de l'opcode sont déjà utilisées pour spécifier une instruction en raison de la taille très restreinte de l'opcode ; si on avait voulu utiliser l'opcode pour spécifier le mode d'adressage des instructions arithmétiques et logiques, il aurait fallu accroître la taille de l'opcode à cinq bits, ce qui aurait restreint les champs *valeur* de toutes les autres instructions ; on a donc logiquement préféré ne restreindre que le champ *valeur* des instructions arithmétiques et logiques. Ce compromis illustre la nature des problèmes pratiques posés par la conception d'un jeu d'instructions.
Les opcodes des instructions mémoire sont les suivants :

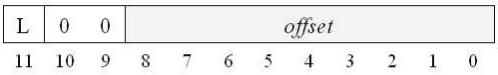
[2] Cette solution est relativement spécifique au LC-2, elle ne doit pas être considérée comme une solution générale.

Opcode	Instruction
0100	**LD**
0110	**LDR**
1110	**LEA**
0011	**ST**
0111	**STR**

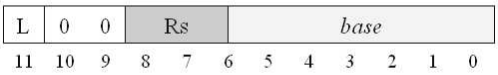
Instructions de contrôle. Les principales instructions de contrôles nécessaires sont les suivantes :

- Une instruction de branchement JMP/JMPR ; comme pour les instructions mémoire, on utilise l'adressage direct (JMP) ou indexé (JMPR) ; afin de simplifier le circuit du LC-2, les champs *base* et *offset* sont les mêmes que pour les instructions mémoire.
- Une instruction d'appel de procédure (JSR/JSRR) ; elle utilise les mêmes modes d'adressage que l'instruction de branchement. La notion d'appel de procédure est implémentée en stockant l'adresse de l'instruction dans un registre spécifique (R7) lors de l'appel. On distingue l'instruction de branchement et l'instruction d'appel de procédure à l'aide d'un bit L situé dans la zone inutilisée du champ opérande.
- Une instruction de retour de procédure (RET) qui récupère l'adresse de retour dans le registre R7.
- Une instruction de branchement conditionnel (BR) qui utilise trois bits de condition N, Z, P indiquant si le résultat courant de l'UAL est nul, négatif ou positif. Cette instruction comprend un champ spécifique de 3 bits qui permet d'indiquer laquelle (ou lesquelles) des conditions N, Z, P doit être prise en compte pour le test.

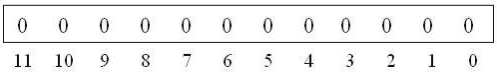
• JSR/JMP:



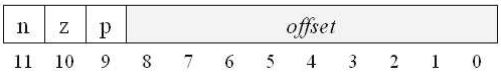
• JSRR/JMPR:



• RET:



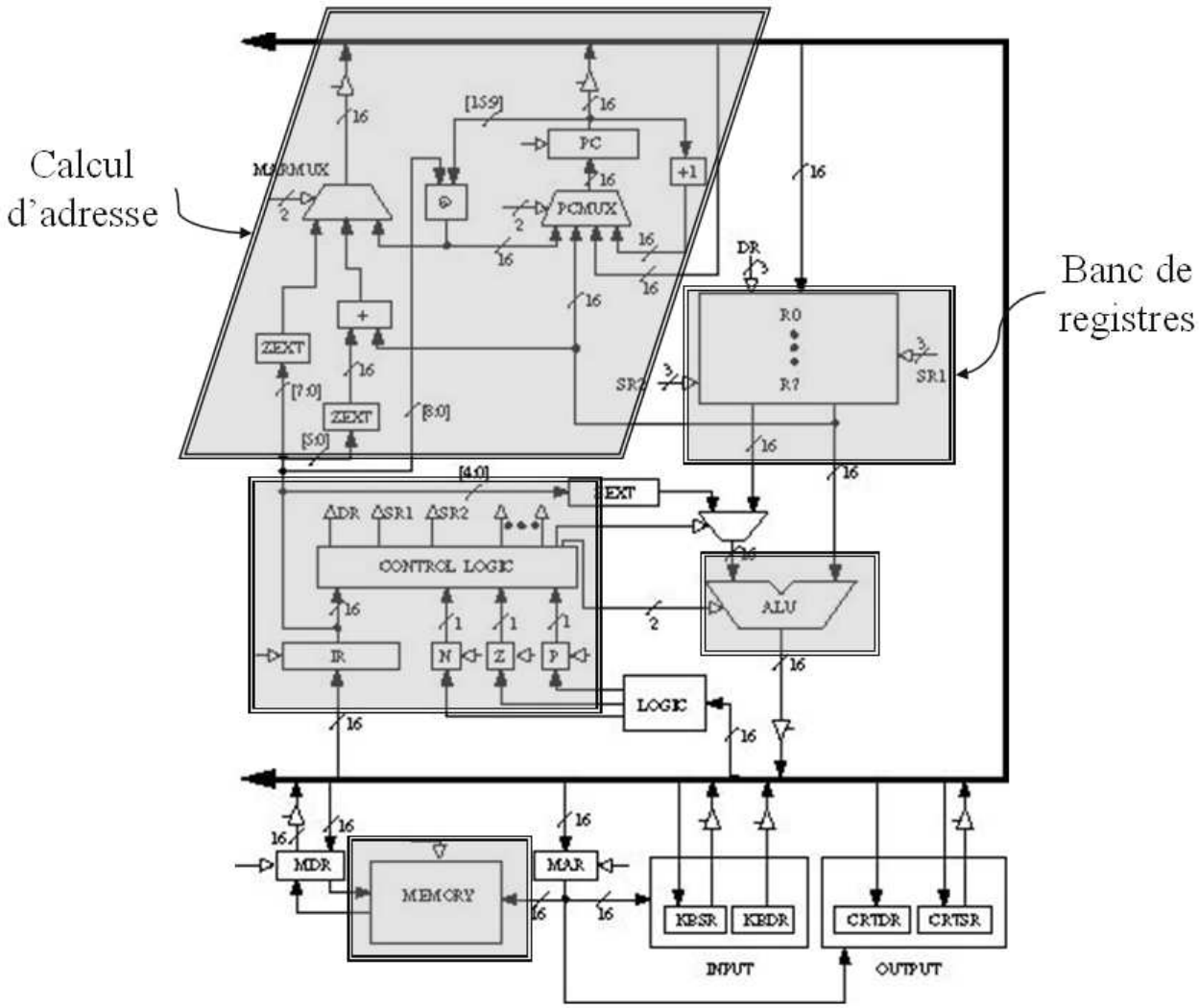
• BR:



- `JMP/JSR L,offset`
  - Adressage direct
  - Si L=1 R7←PC
  - PC←PC[15:9],offset[8:0]
- `JMPR/JSRR L,Rs,base`
  - Adressage indexé
  - Si L=1 R7←PC
  - PC←Rs + base
- `BR nzp offset`
  - Adressage direct
  - PC←PC[15:9],offset[8:0]
  - Test des registres condition dont le bit est à 1.
    - Si bit n=1, test du registre de condition N, valeur condition = valeur du registre N,
    - Si bits n=1 et p=1, valeur condition = OU(N,P).
- `RET`
  - PC←R7

7.2 Une implémentation du jeu d'instructions

La figure ci-dessous décrit une des implémentations les plus simples possibles du jeu d'instructions du LC-2 ; elle est relativement proche du modèle de Von Neumann.



On distingue les composants suivants :

- Unité Arithmétique et Logique,
- banc de registres,
- mémoire,
- unité de contrôle,
- circuit de calcul d'adresse,
- un bus pour relier entre eux les différents composants (des latchs sont utilisés entre le bus et plusieurs des composants, notamment la mémoire (MAR, MDR), pour stocker les informations transmises sur le bus).

L'exécution de chaque instruction est décomposée en huit étapes (un cycle d'horloge par étape), comme indiqué ci-dessous :

1. Envoi de l'adresse de l'instruction à la mémoire
2. Chargement de l'instruction depuis la mémoire
3. Stockage de l'instruction dans le registre IR
4. Décodage de l'instruction (lecture des opérandes)
5. Calcul d'adresse (selon l'instruction)
6. Envoi de l'adresse de l'opérande à la mémoire (selon l'instruction)
7. Chargement/Rangement de l'opérande depuis/dans la mémoire (selon l'instruction)
8. Exécution dans l'UAL (selon l'instruction)
9. Ecriture du résultat ou de l'opérande dans un registre

Bibliographie

[1] Intel 4004, http://www.intel.com/intel/intelis/museum/exhibit/hist_micro/hof/hof_main.htm.

[2] Intel Pentium 4, http://www.intel.com/home/desktop/pentium4.

[3] Intel Itanium, http://www.intel.com/itanium.

[4] C.E. Shannon, *The synthesis of two-terminal switching circuits*, Transactions of the American Institute of Electrical Engineers, 28, 1, 59-98, 1949.

[5] S. B. Furber, D. A. Edwards and J. D. Garside, *AMULET3: a 100 MIPS Asynchronous Embedded Processor*, Proceedings of ICCD'00, Austin, Texas.

[6] Yale N. Patt, Sanjay J. Patel, *Introduction to Computing Systems, from bits and gates to C and beyond*, McGraw Hill International Editions, 2001.

[7] Jean-Michel Muller, *Arithmétique des ordinateurs. Opérateurs et fonctions élémentaires*, Masson, 1989.

[8] David A. Patterson et John L. Hennessy, *Organisation et Conception des Ordinateurs: L'interface Materiel/Logiciel*, chez Dunod.

[9] John L. Hennessy et David A. Patterson, *Architecture des Ordinateurs, une Approche Quantitative*, 2ème édition, chez Morgan Kaufmann.

[10] A. S. Tanenbaum, *Modern Operating Systems*, 2ème édition, chez Prentice Hall