Chapitre 13

L'ordinateur en papier

Voici un ordinateur qui fonctionne sans électricité ; il suffit d'un papier, d'un crayon et d'une gomme (et d'un peu de patience) pour le faire fonctionner. (Ce chapitre est une retranscription du support de cours conçu par Vincent Lesbros en octobre 1990.)

Son schéma est réduit au minimum, mais le principe de fonctionnement est inspiré de celui de petits processeurs séquentiels qui existent réellement.

13.1 Anatomie

L'ordinateur en papier possède une mémoire (la grille carrée) dont chaque case est numérotée. Chaque case peut contenir une valeur numérique (notée en base seize ou système de numération hexadécimal, avec deux chiffres) variant de 00 à FF.

La mémoire contient deux registres :
- Le *Registre de Sélection* (RS) permet de désigner une case de la mémoire par son numéro (ou adresse).
- Le *Registre Mot* (RM) permet d'échanger des valeurs entre la mémoire et l'unité centrale.

La mémoire est de 256 cases, donc les adresses des cases vont de 00 à FF (et peuvent être également contenues dans des cases de la mémoire pour constituer un *pointeur*).

L'unité centrale comporte une unité de contrôle et une Unité Arithmétique et Logique (UAL). L'ordinateur est doté en outre des indispensables moyens de communication avec l'extérieur : une entrée pour saisir des données et une sortie pour exprimer les résultats des traitements.

L'unité de contrôle contient :
- une horloge (non figurée) qui cadence le déroulement des opérations
- le registre (PC) *Program Counter* qui désigne l'adresse en mémoire de la

prochaine instruction du programme à exécuter.
- le séquenceur a accès au *Registre Instruction* (RI), lui-même constitué de deux registres (OP) et (AD) contenant respectivement le *code opératoire* de l'instruction en cours et l'adresse de la donnée à traiter.

L'unité arithmétique et logique est capable d'effectuer des additions, des soustractions et des opérations booléennez.

L'opération à faire s'inscrit dans la case trapézoïdate, les opérandes sont l'*Accumulateur* (le registre (A)) et le registre mot (RM) ; les résultats des opérations sont toujours rendus dans l'accumulateur.

Les différents registres sont des cases pouvant contenir une valeur de huit bits (deux chiffres hexadécimaux) comme les cases de la mémoire. Les registres sont reliès entre eux par des lignes (les bus) et les transferts de valeur par ces lignes sont gouvernés par le séquenceur (suivant l'instruction en cours).

13.2 Fonctionnement

Cette petite machine universelle ne peut fonctionner sans programme : il faut inscrire dans la mémoire une suite d'instructions à effectuer (ou exécuter) et placer dans le registre (PC) l'adresse de la première instruction.

Pour écrire le programme, reportez-vous aux rubriques suivantes : *langage d'assemblage* et *modes d'adressage*.

Quand le programme est écrit, faites *tourner* le programme en répétant le cycle. Le cycle est constitué de trois phases : La première phase est la recherche d'instruction, la deuxième phase contient le décodage de l'instruction et sa réalisation proprement dite (recherche de l'opérande et calcul éventuel), la troisième phase (qui peut être omise dans certains cas) permet de pointer vers l'instruction suivante.

La rubrique *cycle* vous indique pour chaque phase la séquence de microcode à effectuer. (Les microcodes sont notés dans des cercles.) Pour la deuxième phase, la séquence de microcode dépend du code opératoire inscrit dans le registre OP.

La rubrique *les microcodes* vous donne, pour chaque microcode, les choses à faire avec votre crayon et votre gomme.

13.3 Le langage d'assemblage

Mnémoniques	Code opératoire	description
arithmétique		
ADD ♯	20	$A \leftarrow A + V$
ADD α	60	$A \leftarrow A + (\alpha)$
ADD $*\alpha$	E0	$A \leftarrow A + *(\alpha)$
SUB ♯	21	$A \leftarrow A - V$
SUB α	61	$A \leftarrow A - (\alpha)$
SUB $*\alpha$	E1	$A \leftarrow A - *(\alpha)$
logique		
NAND ♯	22	$A \leftarrow \neg[A\&V]$
NAND α	62	$A \leftarrow \neg[A\&(\alpha)]$
NAND $*\alpha$	E2	$A \leftarrow \neg[A\& * (\alpha)]$
transferts		
LOAD ♯	00	$A \leftarrow V$
LOAD α	40	$A \leftarrow (\alpha)$
LOAD $*\alpha$	C0	$A \leftarrow *(\alpha)$
STORE α	48	$(\alpha) \leftarrow A$
STORE $*\alpha$	C8	$*(\alpha) \leftarrow A$
Entrées / Sorties		
IN α	49	$(\alpha) \leftarrow$ Entrée
IN $*\alpha$	C9	$*(\alpha) \leftarrow$ Entrée
OUT α	41	Sortie $\leftarrow (\alpha)$
OUT $*\alpha$	C1	Sortie $\leftarrow *(\alpha)$
Branchement inconditionnel		
JUMP α	10	PC $\leftarrow \alpha$
conditinonels		
BRN α	11	si $A < 0$ alors PC $\leftarrow \alpha$
BRZ α	12	si $A = 0$ alors PC $\leftarrow \alpha$

Légende :

La flèche $\leftarrow$représente l'affectation.
(α) représente le contenu de la case d'adresse alpha.
$*(\alpha)$ représente le contenu de la case dont l'adresse est dans la case d'adresse alpha.
V est la valeur donnée immédiatement.
$\neg$ représente le non logique.
& représente le *et* logique.
Les crochets [ ] groupent les termes sur lesquels s'applique une opération.

Chaque instruction est codée sur deux octets, le premier est le code opératoire (dans un rectangle), le second est la valeur ou l'adresse de l'opérande.

Par exemple :
LOAD 1F (charger le contenu de la case 1F dans A)
s'encode avec : 40 1F dans deux cases consécutives.
LOAD ♯00 (mettre 00 dans A)
s'encode avec : 00 00, toujours sur deux octets.

13.4 Les Modes d'addressage

Les modes d'adressages correspondent aux dífférentes façons d'accéder aux données à traiter.

Adressage immédiat

Dans l'adressage immédiat, on donne directement la valeur avec laquelle l'opération est à effectuer.

Exemples

Mnémoniques	hexadécimal	description
LOAD ♯FF	00	Charge l'accumulateur avec la valeur FF.
	FF	
Mnémoniques	hexadécimal	description
ADD ♯01	20	Incrémente l'accumulateur (c'est à dire ajoute 1
	01	à son contenu)

Adressage absolu

L'adressage absolu permet de désigner une valeur par son adresse en mémoire (ce mode est qualifié d'*absolu* par opposition aux modes dits *relatifs* qui spécifient une adresse en mémoire grâce à un déplacement par rapport à une adresse de référence ; ici, la référence est le début de la mémoire).

Exemple

Mnémoniques	hexadécimal
LOAD 22	40
	22
ADD 20	60
	20

Ce programme charge l'accumulateur avec la valeur contenue dans la case mémoire d'adresse 22 puis ajoute à cette valeur le contenu de la case mémoire 20 ; le résultat est dans l'accumulateur.

Adressage indirect

L'adressage indirect permet de faire une *indirection*, c'est à dire de désigner la valeur à traiter en spécifiant l'adresse d'une case qui contient l'adresse de la valeur.

Une case mémoire contenant l'adresse d'une autre case mémoire est appelée *pointeur*. Ce mode permet de spécifier une valeur en donnant l'adresse du pointeur vers la valeur.

Exemple

Mnémoniques	hexadécimal
SUB * 1F	E1
	1F

Par cette instruction, l'accumulateur va être décrémenté de la valeur dont l'adresse est dans la case mémoire d'adresse 1F. Si, avant cette instruction, l'accumulateur contenait 10, la case mémoire 1F contenait AF et la case AF contenait 2, alors l'accumulateur contiendra 0E après l'exécution de l'instruction (10 − 02 = 0E).

13.5 Les microcodes

Cette rubrique contient la descrption des opérations à effectuer pour chacun des microcodes.

Transferts

Gommez le contenu du registre à gauche de la flèche et y inscrire le contenu de celui de droite.

(1) (RS) ←(PC)

(2) (PC) ←(RM) ; ne pas faire la phase III.

(3) (A) ←(RM)

(4) (RM) ←(A)

(5) (OP) ←(RM)

(6) (AD) ←(RM)

(7) (RS) ←(AD)

(8) (RM) ←(Entrée)

(9) (Sortie) ←(RM)

Préparation du calcul

(10) Écrire + dans UAL (la case trapézoïdale)

(11) Écrire − dans UAL (la case trapézoïdale)

(17) Écrire NAND dans UAL (la case trapézoîdale)

Calculer

(12) Effectuer l'opération affichée sur l'UAL en prenant le contenu de (A) comme premier opérande et le contenu de (RM) comme second. Ecrivez le résultat dans l'accumulateur (A).

Commandes de la mémoire

(13) Lecture de la mémoire :
Décodez l'adresse contenue dans le registre de sélection (RS) en prenant le chiffre gauche (ou Haut) pour numéro de colonne et le droit (ou Bas) pour numéro de ligne. Reportez le contenu de la case mémoire ainsi désignée dans le registre mot (RM).

(14) Ecriture en mémoire :
Décodez l'adresse contenue dans le registre de sélection (RS) de la même façon que pour la lecture. Reportez le contenu du registre mot (RM) dans la case mémoire.

Divers

(15) Incrémenter le contenu de (PC) le pointeur ordinal pour pointer sur l'instruction ou la valeur suivante.

(16) Attendre qu'une valeur soit écrite dans l'Entrée.

13.6 Le cycle

Phase I : Recherche de l'instruction

(1) (13) (5) (15) passer à la phase II

Phase II Décodage de l'instruction, recherche d'opérande et calcul

Valeur de (OP)	microcodes
00	(1) (13) (3)
10	(1) (13) (2) puis passer à la phase I
11	Si (A) $<$ 0 (1) (13) (2) puis phase I
12	Si (A) $=$ 0 (1) (13) (2) puis phase I
20	(10) (1) (13) (12)
21	(11) (1) (13) (12)
22	(17) (1) (13) (12)
40	(1) (13) (6) (7) (13) (3)
41	(1) (13) (6) (7) (13) (9)
48	(1) (13) (6) (7) (4) (14)
49	(1) (13) (6) (7) (16) (8) (14)
60	(10) (1) (13) (6) (7) (13) (12)
61	(11) (1) (13) (6) (7) (13) (12)
62	(17) (1) (13) (6) (7) (13) (12)
C0	(1) (13) (6) (7) (13) (6) (7) (13) (3)
C1	(1) (13) (6) (7) (13) (6) (7) (13) (9)
C8	(1) (13) (6) (7) (13) (6) (7) (4) (14)
C9	(1) (13) (6) (7) (13) (6) (7) (16) (8) (14)
E0	(10) (1) (13) (6) (7) (13) (6) (7) (13) (12)
E1	(11) (1) (13) (6) (7) (13) (6) (7) (13) (12)
E2	*A faire* (voir exercice)

Phase III

(15) puis phase I

13.7 Exemple de programme : un boot-strap

Le *boot-strap* est le programme de lancement de l'ordinateur ; il est implanté à une adresse qui est placée dans le PC à chaque démarrage, c'est donc le premier programme à être exécuté, et il est chargé de faire le nécessaire pour la mise en place et le fonctionnement du système d'exploitation.

Le programme qui suit permet de charger en mémoire votre propre programme et de l'exécuter.

Mode d'emploi :
- Ecrivez le programme à partir de l'adresse 00
- Allumez l'ordinnateur : (PC) ← 00
- Entrez l'adresse de début de votre programme
- Entrez la taille (en nombre d'octets) utilisée par votre programme, puis entrez tour à tour chaque code constituant votre programme.

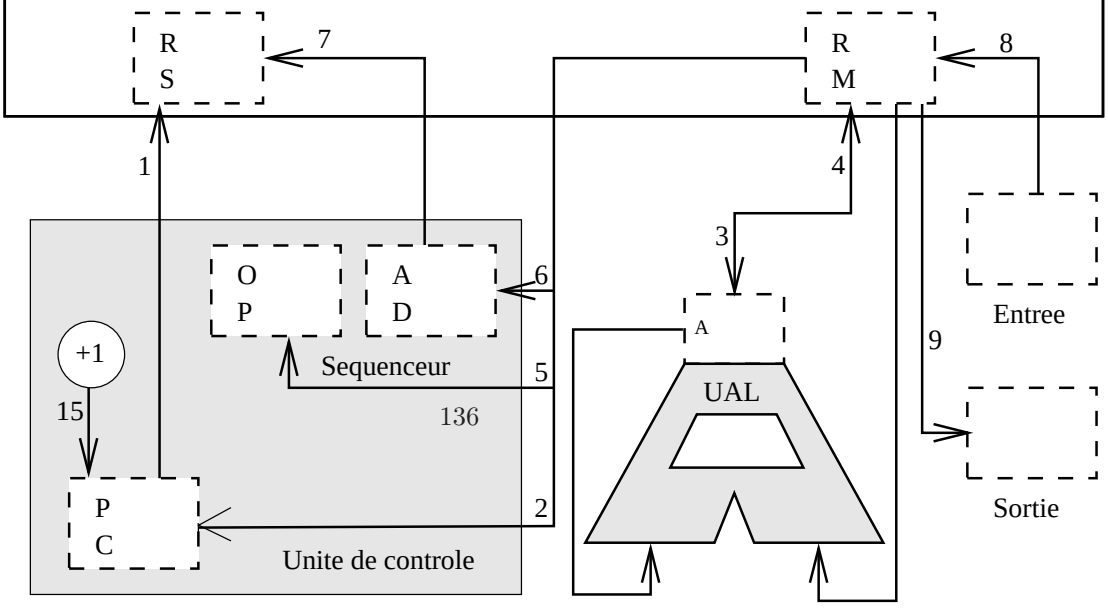
adresse	*code*	*mnémonique*	*commentaire*
00	49	IN 20	Entrez l'adresse de chargement du programme
01	20		
02	49	IN 22	Entrez la taille du programme
03	22		
04	40	LOAD 20	A ←adresse du programme
05	20		
06	48	STORE 21	Adresse courante ←A
07	21		
08	C9	IN * 21	Entrez une instruction ou une valeur
09	21		
0A	40	LOAD 22	A ←Nb. d'octets restant à charger
0B	22		
0C	21	SUB ♯1	Décrémenter le Nb. d'octets
0D	01		
0E	12	BRZ 1F	Exécuter le programme si tout est chargé
0F	1F		
10	48	STORE 22	Ranger le nouveau Nb. d'octets
11	22		
12	40	LOAD 21	Reprendre l'adresse courante dans A
13	21		
14	20	ADD ♯1	l'incrémenter
15	01		
16	10	JUMP 06	Continuer à charger
17	06		
18	1E	00	
1F	10	JUMP ? ?	Brancher sur ...
20	? ?		l'adresse de début
21	? ?		Adresse courante
22	? ?		Taille (ou Nb. d'octets)

Les adresses 00 à 1F sont censées être en mémoire morte et la mémoire vive commence à 20.

13.8 Exercices

Exercice 13.1 : Il manque les microcodes de la phase 2 pour le cas où OP

B \ H	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2																
3																
4																
5																
6																
7																
8																
9																
A																
B																
C																
D																
E																
F																



contient 62. Compléter.

Exercice 13.2 : Étant donné la logique qui existe entre les codes opératoires et le microcode, à quelle code pourrait correspondre l'instruction `IN ♯` ?

Exercice 13.3 : Étant donné la logique qui existe entre les codes opératoires et le microcode, à quelle opération pourrait correspondre le code 01 ?

Exercice 13.4 : (Piégé) Dans le programme de bootstrap, les adresses 18 à 1E sont remplies de 0. Que se passe-t-il au cours du boot-strap quand l'ordinateur exécute ces instructions ?

Exercice 13.5 : Écrire un programme pour l'ordinateur en papier qui lit deux nombres et affiche leur produit. Comme pour le programme de bootstrap donné dans le chapitre, indiquer les adresses, les codes, les mnémoniques et les commentaires. (on peut calculer le produit avec une suite d'additions simple ; faute de décalage à droite, l'ordinateur en papier ne nous permet pas d'utiliser l'algorithme plus rapide vu en cours).

Exercice 13.6 : Même question que précédemment pour la division, avec des soustractions successives.On pourra s'inspirer de l'algorithme suivant :

Etant donné x *et* y*, calculer* a *et* b *tels que* $a \times x + b = y$
$a \leftarrow 0$
$b \leftarrow y$
`tant que` $b > x$
    $a \leftarrow a + 1$
    $b \leftarrow b - x$

Exercice 13.7 : Quelles parties de l'ordinateur faut-il modifier pour ajouter une instruction `SHIFT` de décalage à droite ? Décrire ces modifications, choisir un code opératoire, détailler les modifications à apporter au séquenceur.

Exercice 13.8 : (Assez difficile) Dans la mémoire, on trouve de l'adresse 30 à 45 les valeurs suivantes : 10, 3a, 00 31 40 32 60 33 48 32 49 33 40 33 22 ff 12 34 41 32 10 44. Dans PC se trouve 30. Que fera ce programme quand l'ordinateur en papier va tourner et l'exécuter ?

Exercice 13.9 : (Assez facile) Indiquer les mnémoniques qui correspondent aux valeurs suivantes dans la mémoire.

(Difficile) Commenter le programme. Que fait-il ?

adresse	valeur
50	49 70
52	40 70
54	48 71
56	48 72
58	00 5E
5A	48 8d
5C	10 74
5E	40 71
60	48 72
62	40 70
64	48 71
66	00 6C
68	48 8d
6A	10 74
6C	41 71
6E	10 6E
70	00
71	00
72	00
73	00
74	00 00
76	48 73
78	40 71
7A	12 8c
7C	21 01
7E	41 71
80	40 73
82	60 72
84	48 73
86	10 78
88	40 73
8a	48 71
8c	10 00

Exercice 13.10 : (Amusant mais difficile) Écrire un programme qui met la plus grande partie possible de la mémoire à 0 (en supposant qu'il n'y a pas de mémoire morte).

Exercice 13.11 : (Long) Écrire un programme d'extraction de la racine carrée d'un nombre, avec la méthode de Newton pour l'ordinateur en papier.