

**Programmation du microprocesseur
en langage machine
Version PC et outils MS-DOS**

Patrick Cégielski

Octobre 2018

Pour Irène et Marie

Legal Notice

Copyright © 2018 Patrick CÉGIELSKI

Université Paris Est - Créteil - IUT

Route forestière Hurtault

F-77300 Fontainebleau

cegielski@u-pec.fr

Préface

De nos jours, tout le monde ou presque a déjà utilisé un *ordinateur*¹, et en tous les cas certainement ceux qui désirent lire ce livre. Mais, de même qu'on conduit une automobile sans savoir comment elle fonctionne (tout au moins dans les moindres détails), on se sert d'un ordinateur sans savoir comment il a été conçu. On peut donc distinguer deux attitudes à l'égard des ordinateurs : **conception** et **utilisation**.

Les deux attitudes sont importantes : on ne peut pas utiliser quelque chose qui n'existe pas, d'où l'intérêt de la conception ; à quoi bon concevoir quelque chose qui n'est pas utilisé, d'où l'intérêt d'un marché d'utilisateurs. On imagine bien que la conception doit être complexe, que ce soit celle d'un ordinateur ou d'une automobile. Mais contrairement à l'automobile, dont l'utilisation est relativement simple (sauf exception comme les formules 1, qui sont des automobiles à part), l'utilisation des ordinateurs connaît toute une gradation : simple (utiliser les ressources d'Internet, bureautique, jeux vidéo), un peu moins simple (programmation de petits utilitaires) jusqu'à extrêmement complexe (conception des logiciels de prévision du temps en météorologie, par exemple).

L'utilisateur a, quant aux ordinateurs, seulement besoin de savoir ce que peut faire un ordinateur et considérer celui-ci comme une boîte noire. Ce qu'il peut faire peut rester à un niveau informel ou se modéliser comme *machine de Turing* (dont nous parlerons ci-après) ou tout autre modèle équivalent. On peut, et on doit, partir sur cette dernière base pour une utilisation intelligente.

Si, donc, de nos jours, on aborde les ordinateurs en les utilisant on peut, dans une seconde étape, s'intéresser à leur conception, soit par simple curiosité, soit parce qu'une connaissance même partielle permet quelquefois d'améliorer l'implémentation² de certains algorithmes ou la paramétrisation (administration système), soit parce qu'on veut effectivement concevoir une partie

1. Il s'agit plus spécifiquement d'un *micro-ordinateur*, mais ceci est un raffinement de classification qui n'a rien à voir avec notre sujet.

2. Mais jamais la conception, insistons sur ce point !

d'un nouveau système informatique.

Les ordinateurs sont l'aboutissement d'une longue quête d'outils d'aide aux calculs³. Bien entendu les plus utilisés et de très loin (puisque les autres sont soit d'une technologie dépassée, soit au stade expérimental) sont les ordinateurs électroniques⁴. Nous ne nous intéresserons qu'à ceux-ci dans la suite.

Les ordinateurs électroniques ont eux-mêmes connu une évolution au cours de leur histoire : ordinateurs utilisant des lampes électroniques (celles-ci ayant été créées à l'origine pour les besoins de la TSF), ordinateurs à transistors, ordinateurs à circuits intégrés. Seuls ces derniers sont utilisés de nos jours, sauf cas très spécial.

On distingue dès l'abord deux aspects du point de vue de la conception : l'aspect **matériel** et l'aspect **logiciel**. Le premier concerne les circuits électroniques, les circuits électriques et les parties mécaniques (*hardware* en anglais, ce qui signifie *quincaillerie*), le second concerne tout ce qui est programmation (*software* en anglais, jeu de mot consistant à remplacer *hard* par *soft*).

La distinction de ces deux aspects provient de ce qu'ils correspondent à deux métiers différents à l'origine, et même en grande partie de nos jours : l'aspect matériel est lié à la physique, plus particulièrement à l'électronique, alors que l'aspect logiciel est plutôt lié aux mathématiques.

0.1 Bibliographie

- [Wil-85] WILLIAMS, Michael R., **A History of Computing Technology**, IEEE Computer Society Press, 1985, second edition, 1997, xi + 426 p.

3. Cet aspect est malheureusement très mal traité dans la littérature. On pourra lire [Wil-85] pour une introduction.

4. Mais ce n'est pas la seule possibilité : le premier ordinateur, celui de BABBAGE (conçu en 1834 mais non réalisé), était entièrement mécanique ; puis vinrent les calculateurs électro-mécanique (avec des relais téléphoniques) et enfin électroniques ; il existe aussi des ordinateurs pneumatiques, optiques et biologiques.

Table des matières

Préface	v
0.1 Bibliographie	vi
I Modèle théorique	1
1 Modélisation théorique des ordinateurs	3
1.1 Données, opération et résultat	4
1.2 Notion de registre	5
1.3 Instructions	7
1.3.1 Notion d'instruction	7
1.3.2 Registres de travail	8
1.3.3 Instructions primitives	9
1.3.4 Structures de contrôle	10
1.3.5 Unité arithmétique	12
1.3.6 Programme	12
1.3.7 Programme enregistré	13
1.4 Fonctions calculables	14
1.5 Historique	15
1.5.1 Machines d'aide aux calculs	15
1.5.2 Fonctions calculables	19
1.6 Bibliographie	24
1.7 Exercices	27
II Les principes de la réalisation physique	29
2 Codage de l'information	31
2.1 Le contexte	32
2.2 Codage des entiers naturels	35
2.3 Codage des textes	36
2.4 Appendice : conversions	37
2.5 Historique	38
2.5.1 Utilisation de l'électricité	38
2.5.2 Utilisation de la numération binaire	38
2.5.3 Le codage des textes	39
2.6 Bibliographie	47

3	Les circuits combinatoires	49
3.1	Les circuits combinatoires	50
3.1.1	Fonctions logiques	50
3.1.2	Table de vérité d'une fonction logique	50
3.1.3	Les fonctions logiques naturelles	50
3.1.4	Circuits électroniques	51
3.1.5	Définition des circuits combinatoires	52
3.2	Analyse des circuits combinatoires	52
3.3	Synthèse des circuits combinatoires	53
3.3.1	Réalisation des circuits combinatoires de base	53
3.3.2	Réalisation des autres circuits combinatoires	55
3.3.3	Notation des circuits logique élémentaires	55
3.4	L'additionneur	56
3.4.1	Demi-additionneur	56
3.4.2	Étage d'additionneur	57
3.4.3	Additionneur binaire complet	58
3.5	Historique	60
3.5.1	SHANNON et les circuits de commutation	60
3.5.2	Le premier additionneur (électro-mécanique)	61
3.5.3	Application systématique	63
3.6	Bibliographie	64
4	Réalisation des circuits combinatoires	65
4.1	Les circuits électromécaniques	66
4.1.1	Principe de réalisation des circuits	66
4.1.2	Les relais	66
4.1.3	Réalisation des portes logiques	67
4.2	Les circuits à lampes radio	68
4.2.1	Diode et triode	68
4.2.2	Réalisation des portes logiques à l'aide des lampes électroniques	72
4.3	Les circuits à éléments semi-conducteurs discrets	74
4.3.1	Les circuits à transistors	74
4.3.2	Les circuits à transistors et diodes semi-conductrices	77
4.4	Utilisation des circuits intégrés	79
4.4.1	Notion de circuit intégré	79
4.4.2	Utilisation des circuits intégrés pour les circuits combinatoires	83
4.5	Historique	84
4.5.1	Les circuits mécaniques	84
4.5.2	Les circuits électromécaniques	84
4.5.3	Les circuits à lampes électroniques	87
4.5.4	Les circuits à transistors	90
4.5.5	Les circuits intégrés	91
4.5.6	Lecture 1 : l'ENIAC	93
4.6	Bibliographie	101
5	Réalisation des registres et des mémoires	103
5.1	Réalisation des registres	104
5.1.1	Bistable R.S.	104
5.1.2	Registres à bistables	105
5.2	Des registres à la mémoire	106

5.3	Historique	108
5.3.1	Premières réalisations des registres	108
5.3.2	Les registres à bascule	113
5.4	Bibliographie	116
6	Les bus	119
6.1	Réalisation des transferts	120
6.1.1	Transfert direct entre registres	120
6.1.2	Le problème topologique de la liaison entre registres	121
6.1.3	Les bus	122
6.1.4	Réalisation des transferts <i>via</i> un bus	124
6.1.5	Pupitre de commande	124
6.2	Réalisation des instructions primitives	125
6.2.1	L'unité arithmétique	125
6.2.2	Réalisation des opérations	126
6.3	Historique	127
6.3.1	Calculateurs à pupitre de commande	127
6.3.2	Transcodage décimal–binaire	127
6.4	Bibliographie	128
6.5	Exercices	128
7	Enregistrement des programmes séquentiels	129
7.1	Codage et décodage des instructions primitives	130
7.1.1	Codage des instructions primitives	130
7.1.2	Notion de registre d'instruction	131
7.1.3	Décodage des instructions primitives	131
7.2	Programmes simples	132
7.2.1	Nouveaux composants électroniques	132
7.2.2	Exécution du programme	135
7.2.3	Déroulement d'un programme	136
7.3	Historique	137
7.3.1	Aspect matériel	137
7.3.2	Aspect logiciel	151
7.4	Bibliographie	156
8	Programmes à rupture de séquence	159
8.1	Codage et décodage des ruptures de séquence	160
8.1.1	Codage de la rupture de séquence	160
8.1.2	Décodage des ruptures de séquence	161
8.1.3	Déroulement d'un programme	161
8.2	Historique	162
8.2.1	Notion de programme enregistré avec rupture de séquence (1945)	162
8.2.2	Construction du premier ordinateur (Angleterre, 1948)	163
8.2.3	Le déploiement des ordinateurs	166
8.3	Bibliographie	171
9	Architecture des ordinateurs	173
9.1	Vue d'ensemble	174
9.1.1	Le microprocesseur	174
9.1.2	Architecture primaire	174

9.2	Synchronisation et horloge	174
9.3	Les bus	176
9.3.1	Notion	176
9.3.2	Carte mère	178
9.4	Interfaçage de la mémoire	178
9.4.1	Hierarchie des mémoires	178
9.4.2	Éléments de mémoire centrale	179
9.5	Démarrage d'un micro-ordinateur	180
9.6	Historique	181
9.6.1	Synchronisation	181
9.6.2	Apparition des microprocesseurs	181
9.6.3	Apparition des micro-ordinateurs	186
9.7	Bibliographie	197
III La programmation des microprocesseurs		199
10	La programmation des microprocesseurs	201
10.1	Microprocesseur, mémoire et périphériques	202
10.1.1	Une modélisation des ordinateurs	202
10.1.2	Les registres	202
10.1.3	Mémoire de masse	202
10.1.4	Première classification des instructions d'un microprocesseur	203
10.2	Un exemple de microprocesseur : Intel 8088	204
10.2.1	Aspect extérieur	205
10.2.2	Les broches	205
10.2.3	Les registres internes	206
10.2.4	Les unités du microprocesseur	207
10.3	Façons de programmer un microprocesseur	208
10.3.1	Programmation physique du microprocesseur	208
10.3.2	Kit spécial	208
10.3.3	Programmation sur un système informatique	208
10.4	Historique	209
10.5	Bibliographie	210
10.6	Appendice : MS-DOS sur une clé USB	210
10.6.1	Première étape : récupérer MS-DOS	210
10.6.2	Seconde étape : créer une clé USB bootable	210
10.6.3	Troisième étape : version française complète	213
11	Accès à la mémoire vive du 8088	215
11.1	Modèles de mémoire	216
11.1.1	La mémoire plate du i8085	216
11.1.2	Segmentation de la mémoire du 8086/8088	216
11.2	Écriture et lecture en mémoire avec <code>debug</code>	218
11.2.1	Le logiciel <code>debug</code>	218
11.2.2	Contenu en hexadécimal	218
11.2.3	Analyse d'une zone de mémoire avec <code>debug</code>	218
11.2.4	Changer une zone de mémoire avec <code>debug</code>	219
12	Codage en langage machine	221

12.1	Premières instructions du 8088	222
12.1.1	Premières instructions de transfert du 8088	222
12.1.2	Instruction de retour	225
12.2	Programmer en langage machine avec <code>debug</code>	227
12.2.1	Le programme	227
12.2.2	Un aparté : examen du contenu des registres	228
12.2.3	Codage en langage machine avec <code>debug</code>	230
12.2.4	Visualisation d'un programme	232
12.3	Assembler avec <code>debug</code>	233
12.4	Historique	233
12.4.1	Les langages symboliques	233
12.4.2	CP/M et MS-DOS	234
12.4.3	Debug	235
12.5	Bibliographie	235
IV	Le langage machine du microprocesseur 8088	237
13	Les instructions de transferts	239
13.1	Principe du stockage des mots chez <i>Intel</i>	239
13.2	Autres instructions de transfert	240
13.2.1	Classification des instructions de transfert	240
13.2.2	Stockage du contenu de l'accumulateur dans un élément de mémoire vive	240
13.2.3	Chargement d'un élément de mémoire vive dans l'accumulateur	241
13.2.4	Transfert entre registres ou registre et case mémoire	242
13.2.5	Choix du segment	243
13.3	Historique	244
13.4	Bibliographie	244
14	Accès aux périphériques	245
14.1	Principe de l'accès aux périphériques	246
14.2	Cas du 8088	247
14.2.1	Aspect matériel	247
14.2.2	Entrée dans l'accumulateur	247
14.2.3	Sortie du contenu de l'accumulateur	247
14.3	Un exemple : voyants lumineux du clavier MF II	248
14.4	Historique	249
15	Calculs sur les entiers naturels	251
15.1	Incrémentation et décrémentation	251
15.1.1	Les opérations arithmétiques	251
15.1.2	Représentation des entiers naturels	252
15.1.3	Incrémentation	252
15.1.4	Décrémentation	254
15.2	Addition	256
15.2.1	Addition sur un octet ou sur un mot	256
15.2.2	Addition sur plusieurs mots	259
15.3	Soustraction	262
15.3.1	Soustraction sur un octet ou un mot	262
15.3.2	Cas d'une différence négative	264

15.3.3	Soustraction sur plusieurs mots	265
15.4	Multiplication	267
15.5	Division euclidienne	269
15.6	Historique	272
15.6.1	Les tous premiers programmes (en langage machine)	272
15.6.2	Mise en place	285
15.7	Bibliographie	285
16	Structures de contrôle	287
16.1	Saut inconditionnel	288
16.1.1	Saut inconditionnel intrasegmentaire, direct et relatif	288
16.1.2	Autres sauts inconditionnels	290
16.2	Sauts conditionnels	290
16.2.1	Les différents cas	290
16.2.2	Exemple : l'exponentiation	292
16.2.3	La comparaison	294
16.3	Historique	295
16.4	Bibliographie	295
17	Les entiers relatifs	297
17.1	Entiers relatifs	298
17.1.1	Représentation des entiers relatifs	298
17.1.2	Cas du microprocesseur 8086/8088	300
17.2	Le décimal codé binaire	305
17.2.1	Notion	305
17.2.2	Addition en DCB	306
17.2.3	Soustraction en DCB	307
17.2.4	Cas de la multiplication et de la division	307
17.3	Historique	307
17.3.1	Blaise PASCAL et le complément à neuf	307
18	Programmation modulaire	309
18.1	Notions générales	310
18.2	Cas du microprocesseur 8086/8088	311
18.2.1	Appel du sous-programme	311
18.2.2	Retour d'un sous-programme	311
18.2.3	Un exemple	312
18.3	Historique	313
19	La pile	315
19.1	La pile	315
19.1.1	Étude générale de la pile	315
19.1.2	Cas du microprocesseur 8086/8088	316
19.1.3	Un exemple	319
19.2	Le passage des paramètres à un sous-programme	320
19.2.1	Utilisation des registres de données	320
19.2.2	Utilisation de la mémoire	321
19.2.3	Utilisation de la pile	321
20	Manipulation des bits	325

20.1	Instructions logiques	326
20.1.1	Mise en place	326
20.1.2	Application au masquage	328
20.2	Les décalages logiques	329
20.3	Les rotations	331
20.3.1	Les rotations sur un octet ou un mot	331
20.3.2	Les rotations sur un octet ou un mot et l'indicateur CF	333
20.4	Changer l'indicateur de retenue	335
20.5	Historique	335
21	Les interruptions	337
21.1	Types d'interruptions	337
21.1.1	Trois types d'interruption	337
21.1.2	Les interruptions internes	338
21.2	Les interruptions logicielles	338
21.2.1	Mise en place	338
21.2.2	Un exemple d'appel d'interruption logicielle	340
21.2.3	Un exemple de routine d'interruption logicielle	341
21.3	Masquage des interruptions externes masquables	343
22	Les chaînes de caractères	345
22.1	Définition des chaînes de caractères	346
22.1.1	Représentation des caractères	346
22.1.2	Représentation des chaînes de caractères	346
22.1.3	Jeu de caractères utilisé	347
22.2	Adressage indirect par registre et tableaux	347
22.2.1	Notion	347
22.2.2	Un exemple	348
22.3	Primitives de manipulation des mots	350
22.3.1	Copie	350
22.3.2	Remplissage	355
22.3.3	Chargement	358
22.3.4	Comparaison	363
22.3.5	Recherche	366
22.3.6	Choix de la direction	369
22.4	Historique	369
23	Dernières instructions	371
23.1	Adressage indexé	372
23.2	Instructions d'échange	373
23.3	Pas d'opération	373
23.4	Arrêt du programme	374
23.5	Tables de traduction	374
23.6	A	375
23.7	Historique	375
24	Assemblage et désassemblage	377
24.1	Assemblage	378
24.1.1	Champs d'une instruction	378
24.1.2	Instructions sans opérande	378

24.1.3	Instructions avec un seul opérande	379
24.1.4	Instructions avec deux opérandes	384
24.2	Désassemblage	388
24.2.1	Instructions sans préfixe	388
V	Évolution	391
25	Évolution des microprocesseurs	393
25.1	Le tout premier microprocesseur : le 4004	394
25.2	Le 8008	395
25.3	Le 8080	396
25.4	Le 8085	397
VI	Programmation en langage d'assemblage	399
26	Initiation aux langages d'assemblage	401
26.1	Notion et mise en place de l'assembleur	402
26.1.1	Notion générale	402
26.1.2	Les assembleurs pour PC et MS-DOS	402
26.1.3	Installation de MASM	403
26.1.4	Un premier exemple de programme en langage d'assemblage	403
26.2	Structure d'un programme source en langage d'assemblage	405
26.2.1	Instructions et directives d'assemblage	405
26.2.2	Segments d'un programme	406
26.2.3	Présentation d'un programme d'assemblage	408
26.2.4	Programme et DOS	409
26.2.5	Exemples de programmes	410
26.2.6	Structure d'un programme	410
26.2.7	Définition simplifiée des segments	411
26.2.8	Définition des segments	412
26.3	Les fichiers utilitaires de l'assembleur	414
26.3.1	Listing	414
26.4	Forme d'un programme en langage d'assemblage	417
26.4.1	Identificateurs	417
26.4.2	Syntaxe d'une instruction de langage d'assemblage	419
26.4.3	Les nombres entiers en langage d'assemblage	420
26.4.4	Initialisation d'un registre avec un caractère	420
26.4.5	Accès à la mémoire vive grâce aux variables	420
26.4.6	Les constantes en langage d'assemblage	423
26.4.7	Les structures de contrôle	423
26.5	Les sous-programmes en langage d'assemblage	428
26.5.1	Sous-programme intra-segmentaire	428
26.5.2	Sous-programme inter-segmentaire	430
26.5.3	Utilisation de la pile pour conserver les registres	431
26.6	Les chaînes de caractères	432
26.6.1	Déclaration des chaînes de caractères	432
26.6.2	Affichage d'une chaîne de caractère	432
26.6.3	Entrée-sortie des chaînes de caractères	433

26.7 Entrées-sorties des entiers	437
26.7.1 Affichage d'un entier	437
26.7.2 Saisie d'un entier	439
26.8 La programmation modulaire	441
26.8.1 Le problème de l'interface	441
26.8.2 Un exemple	442
26.8.3 Les bibliothèques	446
26.9 Langage d'assemblage et langage évolué	448
26.9.1 Lier un programme C et un programme en langage d'assemblage	448
26.10Bibliographie	450
Index	455

Table des figures

1.1	Données, opération et résultat	4
1.2	Machine à registres	5
1.3	Machine à registres avec registres de travail	8
1.4	Machine avec unité arithmétique	12
1.5	Vase de Darius ([Wil-85], p. 56)	15
1.6	Abaque de Salamis ([Wil-85], p. 57)	16
2.1	Principe de la transmission d'un bit	32
2.2	Principe de la transmission de l'information	33
2.3	Numération binaire	35
2.4	Tableau de Frederic REMINGTON	39
2.5	Télégraphe Chappe [Fig-68]	40
2.6	Code utilisé pour le télégraphe Chappe	41
2.7	Code Morse	42
2.8	Clavier télégraphique à 5 touches d'Émile Baudot, Journal télégraphique , vol. 8, n° 12, décembre 1884	43
2.9	Disposition des mains sur le clavier Baudot	44
2.10	Code Baudot	46
3.1	Représentation d'un circuit électronique	51
3.2	Principe d'un relais	53
3.3	Représentation schématique d'un relais	53
3.4	Principe de réalisation d'un inverseur	54
3.5	Principe de réalisation d'une porte OU	54
3.6	Principe de réalisation d'une porte ET	55
3.7	Notation des portes logiques	55
3.8	Réalisation d'un demi-additionneur	56
3.9	Notation d'un demi-additionneur	57
3.10	Réalisation d'un étage d'additionneur	57
3.11	Notation d'un étage d'additionneur	58
3.12	Notation d'un additionneur	58
3.13	Additionneur binaire parallèle	59
3.14	Additionneur binaire parallèle 16 bits	60
3.15	Schéma du demi-additionneur de Stibitz ([Lig-87], p.227)	61
3.16	Réplique du demi-additionneur de STIBITZ ([Aug-84], p. 101)	62
4.1	Principe d'un relais	66
4.2	Représentation schématique d'un relais	66

4.3	Inverseur à relais	67
4.4	Porte OU à relais	67
4.5	Porte ET à relais	68
4.6	L'émission thermoélectronique	68
4.7	Principe de la diode	69
4.8	Très faible passage de courant	69
4.9	Passage de courant	70
4.10	Une diode ([Bow-53], p. 40)	70
4.11	Principe de la triode	71
4.12	Symbole de la triode	71
4.13	Train d'impulsions négatives ([Bow-53], p. 42)	72
4.14	Réalisation d'une porte ET avec deux diodes ([Bow-53], p. 43)	72
4.15	Réalisation d'une porte OU avec deux diodes ([Bow-53], p. 44)	73
4.16	Réalisation d'un inverseur avec une deux triodes ([Bow-53], p. 46)	73
4.17	Courbes caractéristiques d'un transistor	74
4.18	Utilisation d'un transistor comme inverseur	75
4.19	Utilisation de deux transistors comme porte NAND	76
4.20	Utilisation de deux transistors comme porte NOR	76
4.21	Présentation des transistors à l'utilisateur	77
4.22	Symbole et caractéristiques d'une diode semi-conductrice	77
4.23	Réalisation des portes ET et OU avec des diodes semi-conductrices	78
4.24	Aspect d'une puce électronique	79
4.25	Boîtier de circuit intégré avec des broches DIL	79
4.26	Schéma fonctionnel du circuit intégré SN7400 ([TI-73], p. 62)	80
4.27	Durée de propagation à travers une porte	81
4.28	Le premier électro-aimant, inventé par STURGEON in 1824. Dessin de l'article de 1824 dans <i>British Royal Society of Arts, Manufactures, and Commerce</i> . Il y a 18 tours de fil de cuivre (non isolé).	84
4.29	Le premier relais électrique ([Lig-87], p. 184)	85
4.30	Relais électromagnétique	85
4.31	Fig. 1 de [Ata-40]	89
4.32	Table dans [Ata-40]	90
4.33	Le premier prototype de circuit intégré ([Aug-84], p. 237)	91
4.34	Schéma général de l'ENIAC ([Lig-87], p. 321)	98
5.1	Principe du bistable R.S.	104
5.2	Notation du bistable R.S.	104
5.3	Principe d'un registre à bistables	105
5.4	Mise à zéro d'un registre	105
5.5	Schéma d'un élément de mémoire ROM	106
5.6	Entrée d'un nombre bit par bit ([Aug-84], p. 126)	108
5.7	Réplique de la machine Z3 de ZUSE ([Wil-85], p. 217)	109
5.8	Le micro-ordinateur Altair ([Aug-84], p. 274)	109
5.9	Une unité de lignes à retard du BINAC ([Aug-84], p. 158)	110
5.10	Le calculateur IAS à la Smithsonian Institution à Washington ([Aug-84], p. 155)	111
5.11	Mémoire à tores de ferrite ([Aug-84], p. 194)	112
5.12	Le premier basculeur ([Lig-87], p. 268)	113
5.13	Le premier bistable sur circuit intégré ([Aug-83], p. 11)	114
5.14	La première RAM : le 4100 de <i>Fairchild</i> ([Aug-83], p. 25)	115
5.15	La première RAM dynamique, de 1024 bits : le 1103 de <i>Intel</i> ([Aug-83], p. 27)	116

6.1	Transfert entre registres par deux impulsions	120
6.2	Premier système de transfert entre registres à entrée forcée	120
6.3	Deuxième système de transfert entre registres à entrée forcée	121
6.4	Liaisons registres d'entrée – bus des données	122
6.5	Liaisons registres de sortie – bus des données	123
6.6	Liaisons bus–registres de travail	123
6.7	L'unité arithmétique	125
6.8	Transcodage décimal–binaire ([Lig-87], p. 228)	127
7.1	Codage et décodage	131
7.2	Bistable J.K.	132
7.3	Notation d'un bistable J.K.	133
7.4	Élément de mémoire J.K.	133
7.5	Notation d'un élément de mémoire J.K.	134
7.6	Bascule J.K.	134
7.7	Train d'impulsions	134
7.8	Registre d'incrémentatation	135
7.9	Compteur de programme	135
7.10	Programmation par câblage ([Bur-80], p. 328)	137
7.11	Programmation par câblage ([Bur-80], p. 317)	138
7.12	([WR-50], p. 419)	148
7.13	([WR-50], p. 421)	149
7.14	Le code de l'EDSAC (1949)	154
7.15	Format d'instruction de l'EDSAC ([Cam-01], p. 14)	155
9.1	Architecture primaire d'un ordinateur	174
9.2	Circuit non asynchrone	175
9.3	Délai de propagation	175
9.4	Circuit synchrone	175
9.5	Délai synchrone	176
9.6	Architecture à trois bus	177
9.7	Aspect extérieur du microprocesseur 4004 ([Mal-95], p. 11)	181
9.8	La première calculatrice avec microprocesseur ([Mal-95], p. 4)	182
9.9	Vue interne du microprocesseur 4004 ([Aug-84], p. 266)	183
9.10	Publicité pour le microprocesseur 4004 ([Aug-84], p. 264)	184
9.11	Vue interne du microprocesseur 8008 ([Aug-84], p. 266)	185
9.12	Vue interne du microprocesseur 8080 ([Aug-84], p. 266)	185
9.13	PDP-1 ([Aug-84], p. 256)	186
9.14	PDP-8 ([Aug-84], p. 256)	187
9.15	Le premier Micral photographié à côté de sa carte mère ([Lil-03], p. 93)	188
9.16	La couverture de <i>Radio-Electronics</i> de juillet 1974 ([Aug-84], p. 269)	189
9.17	La couverture de <i>Popular Electronics</i> de janvier 1975 ([Aug-84], p. 273)	190
9.18	IMSAI 8080 ([Lil-03], p. 114)	191
9.19	Apple I ([Aug-84], p. 278)	192
9.20	Apple II ([Lil-03], p. 116)	193
9.21	KIM-I (Bolo's Computer Museum, http://www.bolo.ch)	194
9.22	Le Pet de Commodore ([Lil-03], p. 124)	195
9.23	IBM PC (1981)	196
10.1	Boîtier du 8088	203

10.2	Cœur du 8088	204
10.3	Broches du 8088	205
10.4	Structure fonctionnelle du 8088	207
10.5	Fenêtre de HP USB Disk Storage Format Tool	211
10.6	Fenêtre d'avertissement de HP USB Disk Storage Format Tool	212
11.1	Segmentation de la mémoire	217
12.1	Désignation des registres généraux du 8088	223
12.2	Désignation des registres de segment du 8088	224
13.1	Modes d'adressage du 8088 lorsque mod = 00	242
15.1	Le premier programme (18 juillet 1948)	272
15.2	Première routine (Worsley 1950)	273
15.3	Deuxième routine (Worsley 1950)	274
15.4	Troisième routine (Worsley 1950)	275
15.5	Premier organigramme (Worsley 1950)	276
15.6	Second organigramme (Worsley 1950)	277
15.7	Troisième organigramme (Worsley 1950)	278
15.8	Table des carrés (Worsley 1950)	279
15.9	Table des nombres premiers (Worsley 1950)	280

Première partie

Modèle théorique

Chapitre 1

Modélisation théorique des ordinateurs

La programmation des ordinateurs exige une connaissance d'un modèle de ceux-ci. Un *modèle théorique* peut et doit être suffisant. Les modèles théoriques sont d'ailleurs apparus historiquement avant la réalisation.

1.1 Données, opération et résultat

On définit en mathématiques la notion générale de *fonction*, résumée par la notation bourbakiste :

$$f : E \subseteq A \rightarrow B$$

où A et B sont des ensembles. L'ensemble de définition E est l'ensemble des éléments x de A pour lesquels $f(x)$ est défini.

Un ordinateur est une « boîte noire » qui permet d'obtenir la valeur $f(x)$ lorsque f et x sont donnés : on entre une suite de nombres¹, appelée les **données**, on lui indique une **opération** ou **fonction** à effectuer (dans un sens très large, bien au-delà des quatre opérations arithmétiques que sont l'addition, la multiplication, la soustraction et la division) et il en sort une autre suite de nombres², appelée le **résultat**, conformément au schéma de la figure 1.1 :

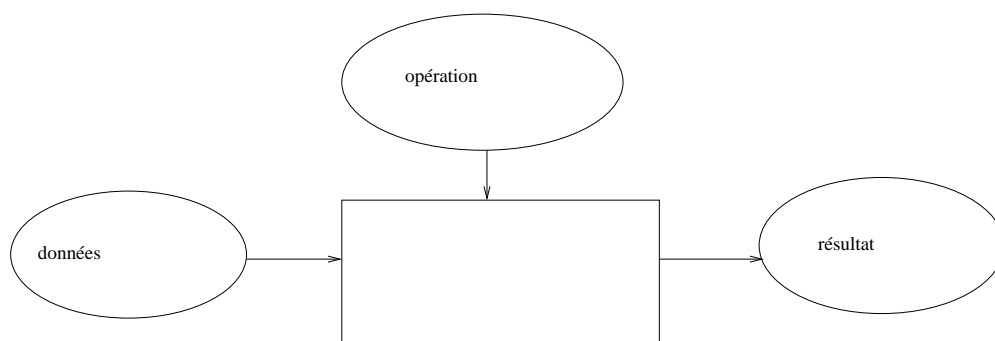


FIGURE 1.1 – Données, opération et résultat

Pour les opérations courantes (addition, soustraction, multiplication, division...) les données sont au nombre de deux : on parle de **fonction binaire**. Elles sont quelquefois au nombre de un (carré, cube, racine carrée...) : on parle de **fonction unaire**. Elles peuvent même être d'un nombre plus grand (opérations sur les matrices par exemple). Il en est de même pour les résultats qui se présentent souvent sous la forme d'un nombre unique (cas de l'addition, de la multiplication, de l'extraction de racine carrée...) mais peut comporter plusieurs nombres (cas des opérations sur les matrices mais aussi plus simplement division euclidienne où l'on donne le quotient et le reste).

1. ou de caractères ou de ce qu'on veut.

2. ou ce qu'on veut.

1.2 Notion de registre

Appelons **registre** tout mécanisme capable de stocker une donnée et de la restituer au moment voulu.

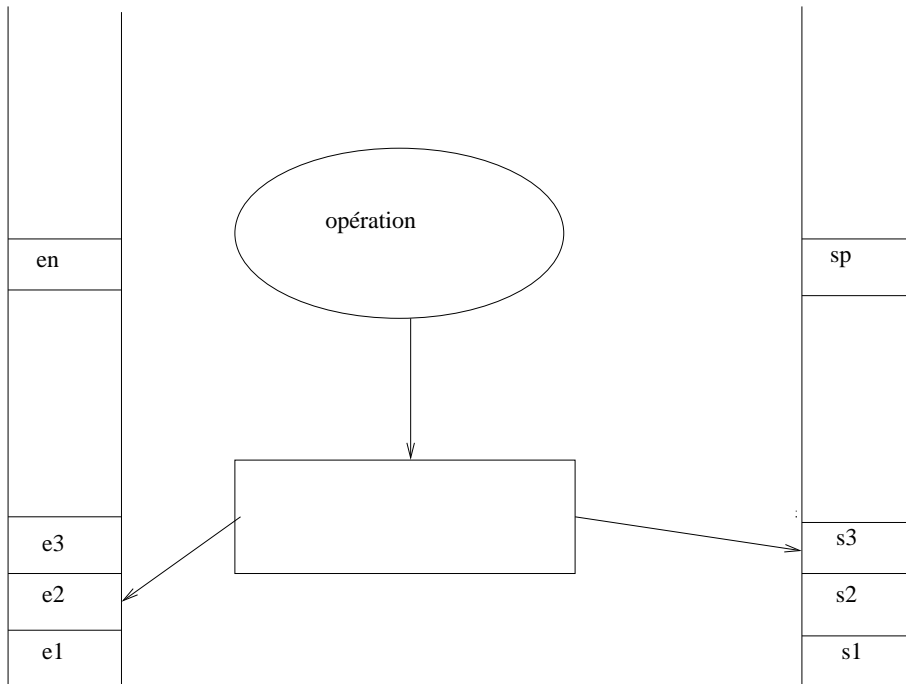


FIGURE 1.2 – Machine à registres

Un ordinateur peut alors être décrit d'une façon un peu plus fine de la façon suivante :

- on a une pile de **registres d'entrée**, numérotés $e_1, e_2, e_3, \dots, e_n, \dots$, potentiellement infinie, chaque registre pouvant contenir un entier naturel;
- on a une pile de **registres de sortie**, numérotés $s_1, s_2, s_3, \dots, s_n, \dots$, potentiellement infinie également, chaque registre pouvant contenir également un entier naturel.

Si on veut faire effectuer l'opération op dont on sait qu'elle a besoin de n entrées et de p sorties, on place les n entrées dans les registres e_1 à e_n (dans cet ordre si l'ordre est important) et on indique à la machine qu'elle doit effectuer l'opération op : celle-ci va chercher les n entrées dans les n premiers registres d'entrée, les triture par un procédé que l'on n'a pas à connaître pour l'instant et place les résultats dans les p premiers registres de sortie. On peut alors aller récupérer les résultats dans les registres de sortie quand on veut.

Le traitement de l'opération n'a évidemment aucune raison d'être immédiat. Il faut donc prévoir un **signal de fin** pour indiquer que la machine a terminé d'effectuer l'opération afin qu'on n'aille pas chercher des résultats erronés.

Remarque.- L'ordinateur est décrit d'une façon un peu plus fine mais beaucoup s'exclameront « mais pas de façon réaliste ». En effet on ne voit pas comment concevoir une pile infinie de registres. En fait nous avons parlé d'une pile potentiellement infinie : on peut partir d'une pile finie plus ou moins grande; si elle n'est pas suffisante on ajoute des registres. Cependant la

théorie de la relativité générale a pour conséquence qu'il n'existe qu'un nombre fini de particules élémentaires dans l'univers ; il est donc impossible d'étendre une telle pile au-delà d'une certaine taille.

Il n'en demeure pas moins qu'il est plus simple de raisonner sur une pile (potentiellement) infinie. Nous étudierons plus tard l'incidence de cette hypothèse.

De même on ne voit pas comment concevoir un registre capable de recevoir un entier naturel aussi grand que l'on veut. Le même commentaire s'applique à cette deuxième restriction.

1.3 Instructions

1.3.1 Notion d'instruction

Problème.- Comment indiquer l'opération à effectuer ?

Voilà la question que l'on se pose naturellement à ce point de l'exposé. Pour les opérations élémentaires (addition, multiplication, extraction des racines carrées, logarithme, sinus...) on peut penser à munir la machine d'une série de touches sur lesquelles sont indiqués les noms des opérations. Il suffit d'appuyer sur la touche désirée pour déclencher l'opération (c'est ce qui se fait pour les *calculettes*). Ce système n'est pas envisageable dans le cas d'un ordinateur car le nombre des fonctions calculables est très grand, voire même infini.

Un espoir : génération finie du nombre infini d'opérations.- On a de la chance. Bien que le nombre d'opérations soit infini, toute opération calculable s'obtient à partir d'un nombre fini d'*opérations primitives* (ou *élémentaires*) grâce à un nombre fini de *constructeurs d'opérations composées*.

Instruction.- On préfère parler d'**instruction** en informatique plutôt que d'opération, bien que ce soit la même chose, car on donne des instructions à la machine comme on le ferait à une personne pour effectuer un certain travail.

1.3.2 Registres de travail

On a quelquefois besoin d'entreposer des résultats intermédiaires : lorsqu'on effectue une soustraction, par exemple, on a besoin de place pour marquer les retenues sur une feuille de papier. On considèrera donc qu'on dispose, outre des deux piles de registres de donnée et de registres de résultat, d'une pile de **registres de travail**, notés $r_1, r_2, r_3, \dots, r_n, \dots$, chacun étant capable de contenir un entier naturel, comme le montre la figure 1.3.

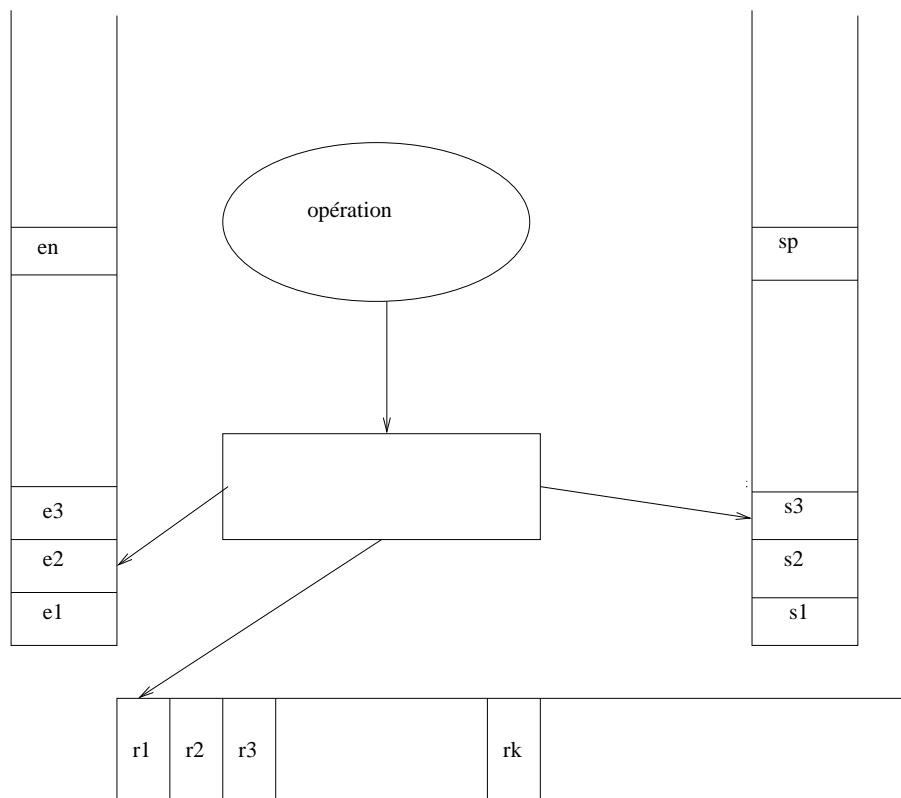


FIGURE 1.3 – Machine à registres avec registres de travail

1.3.3 Instructions primitives

Classification des instructions primitives.- On peut voir immédiatement un certain nombre d'instructions primitives utiles, que l'on peut répartir en trois types :

- Les **instructions de transfert** permettent de copier le contenu d'un registre dans un autre :
 - du registre d'entrée n vers le registre de travail k , notée $r_k := e_n$;
 - du registre de travail n vers le registre de travail k , notée $r_k := r_n$;
 - du registre de travail k vers le registre de sortie n , notée $s_n := r_k$.
 - Il n'y a pas en général de transfert du registre d'entrée n vers le registre de sortie k .
 - Remarquons que le transfert d'un registre de travail vers un registre d'entrée ou d'un registre de sortie vers un registre de travail n'a pas vraiment de sens.
- Les **instructions arithmétiques** permettent des calculs élémentaires :
 - La **remise à zéro** du registre de travail k , notée $r_k := 0$.
 - L'**incréméntation** du registre de travail k , c'est-à-dire le fait de lui ajouter 1, notée $r_k := r_k + 1$.
 - La **décréméntation** du registre de travail k , c'est-à-dire le fait de lui retrancher - 1 s'il est non nul (il reste à zéro s'il est nul si on veut que le résultat soit un entier naturel), notée $r_k := r_k - 1$.
- La seule **instruction de contrôle** primitive est :
 - l'instruction d'**arrêt** de la machine, notée **STOP**, qui fait, par exemple, clignoter une lampe pour indiquer que l'on peut aller chercher les résultats dans les registres de sortie.

Exercice.- *Voyez-vous d'autres instructions primitives possibles?*

1.3.4 Structures de contrôle

On appelle **programme** le concept précis d'*instruction complexe* construite à partir des instructions primitives à l'aide des constructeurs d'instructions.

On appelle **constructeur d'instructions** ou **structure de contrôle** ce qui permet d'obtenir d'autres instructions à partir des instructions primitives. Voyons quelques structures de contrôle naturelles.

1.3.4.1 Séquencement

Notion.- La première structure de contrôle qui nous vient à l'esprit est certainement le **séquencement** : il s'agit d'exécuter une suite d'instructions l'une après l'autre.

Exemple.- L'instruction composée suivante permet de placer 3 dans le premier registre de sortie :

```
 $r_1 := 0$   
 $r_1 := r_1 + 1$   
 $r_1 := r_1 + 1$   
 $r_1 := r_1 + 1$   
 $s_1 := r_1$   
STOP
```

1.3.4.2 Rupture de séquence

Étiquetage des instructions.- On va, pour aller plus loin, ce que dont n'avions pas besoin jusqu'ici, repérer les instructions par des **étiquettes**, de la forme q_n suivi de deux points, où n est un entier naturel.

Le programme précédent peut être réécrit de la façon suivante en utilisant des étiquettes :

```

 $q_0 : r_1 := 0$ 
 $q_1 : r_1 := r_1 + 1$ 
 $q_2 : r_1 := r_1 + 1$ 
 $q_3 : r_1 := r_1 + 1$ 
 $q_4 : s_1 := r_1$ 
 $q_5 : \text{STOP}$ 

```

Ceci est optionnel : on peut mettre des étiquettes, ne pas en mettre, en mettre certaines et pas d'autres.

Branchement.- L'**instruction de branchement** :

```

 $\text{SI } r_k = 0 \text{ ALLER\_A } q_k$ 

```

permet une **rupture de séquence** : si le contenu du registre de travail r_k est nul alors on va, non pas à l'instruction suivante dans l'ordre dans lequel les instructions sont écrites, mais à celle d'étiquette q_k .

Bien que les étiquettes soient facultatives, il faut nécessairement, dans le cas d'une rupture de séquence, que l'instruction à laquelle on veut renvoyer soit étiquetée.

Exemple.- (**Addition**)

Le programme suivant permet d'additionner les contenus des deux premiers registres d'entrée et d'en placer le résultat dans le premier registre de sortie :

```

 $r_1 := e_1$ 
 $r_2 := e_2$ 
 $r_3 := 0$ 
 $q_1 : \text{SI } r_2 = 0 \text{ ALLER\_A } q_2$ 
 $r_2 := r_2 - 1$ 
 $r_1 := r_1 + 1$ 
 $\text{SI } r_3 = 0 \text{ ALLER\_A } q_1$ 
 $q_2 : s_1 := r_1$ 
 $\text{STOP}$ 

```

1.3.5 Unité arithmétique

Pour ne pas tenir compte du choix spécifique des instructions primitives, présentons un nouveau modèle.

On appelle **unité arithmétique** (*arithmetic unit* en anglais) l'organe de la machine capable d'effectuer les instructions arithmétiques primitives que l'on a choisies, pas nécessairement celles ci-dessus.

On considère en général, pour simplifier, que l'unité arithmétique ne travaille qu'avec les registres de travail. Nous obtenons alors le schéma de la figure 1.4 :

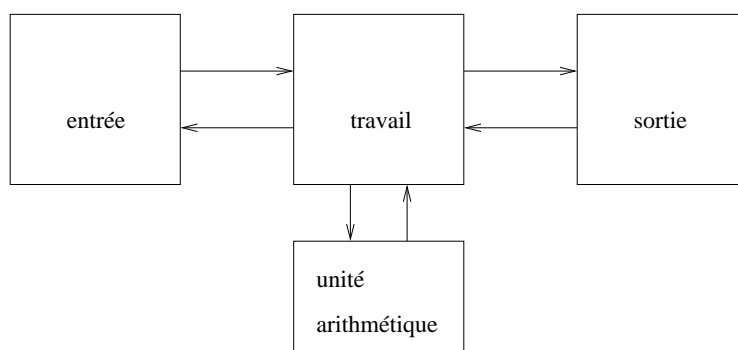


FIGURE 1.4 – Machine avec unité arithmétique

1.3.6 Programme

Programme.- On appelle **programme** toute suite d'instructions, que ce soient des instructions primitives ou des structures de contrôle.

L'idée est qu'un programme permet d'effectuer une opération. On a donc une façon finie de générer un grand nombre d'opérations.

Exercice.- Montrer que l'ensemble des programmes et celui des opérations calculables sont infinis.

⊢ Pour tout entier naturel a l'application constante f_a définie par $f_a(n) = a$ peut être calculée par un programme analogue au premier programme, calculant 3, que nous avons considéré ci-dessus. ⊣

1.3.7 Programme enregistré

Une idée fondamentale de VON NEUMANN a été de placer les instructions, et donc le programme, dans des registres.

Registres d'instruction.- Nous avons précédemment numéroté les instructions afin de pouvoir effectuer des ruptures de séquence. Introduisons maintenant de nouveaux registres, appelés **registres d'instruction**, notés $i_1, i_2, i_3, \dots, i_n, \dots$. Mettons des étiquettes à toutes les instructions. Plaçons l'instruction d'étiquette q_1 (sans son étiquette) dans le registre i_1 , celle d'étiquette q_2 dans i_2 et ainsi de suite.

Pointeur d'instruction.- Introduisons un nouveau registre, appelé **pointeur d'instruction** qui contient une **adresse d'instruction**, c'est-à-dire un entier naturel non nul.

Le programmeur n'a accès au registre d'instruction que de façon indirecte. Au début de l'exécution du programme, le registre d'instruction a la valeur 1. Après l'exécution de chaque instruction, la valeur du registre d'instruction est incrémentée sauf s'il s'agit d'une rupture de séquence : dans ce dernier cas, le registre d'instruction prend la valeur indiquée.

Mise en œuvre d'un programme.- On voit alors comment faire exécuter un programme : on place les instructions et les données dans les registres adéquats et on fait démarrer le programme. L'instruction i_1 est alors effectuée; on passe ensuite à l'instruction suivante, celle de numéro suivant en général sauf si l'instruction est une instruction de rupture de séquence; le programme s'arrête lorsqu'on rencontre l'instruction **STOP**.

Remarque.- Que se passe-t-il si on ne tombe jamais sur l'instruction **STOP**? Le programme ne s'arrête pas : on dit qu'il **boucle**. Vous êtes certainement déjà tombés (involontairement) sur un tel cas si vous avez programmé. Voici par exemple un programme qui boucle :

```
q1 : r1 := 0
q2 : SI r1 = 0 ALLER_A q1
q3 : STOP
```

1.4 Fonctions calculables

Introduction.- Nous venons de voir le principe de réalisation des ordinateurs. Une telle machine peut-elle calculer tout ce qui est calculable? N'avons-nous pas oublié une instruction primitive ou une structure de contrôle? Pire, sommes-nous sûrs qu'il existe un jeu en nombre fini bien choisi d'entre elles?

Il a existé au cours de l'histoire de nombreuses machines d'aide aux calculs : abaques et bouliers, machine arithmétique de Pascal, ... L'idée intuitive de ce qu'est une *opération* (nous dirons *fonction* dorénavant) *calculable* est une opération dont le résultat peut être obtenu grâce à une machine (d'aide aux calculs), qu'importe la nature de celle-ci. Rien ne dit d'ailleurs *a priori* qu'il existe des fonctions non calculables (nous verrons cependant plus tard qu'il en existe).

Un programme utilise n entrées et p sorties. On peut considérer que les entrées et les sorties sont des entiers naturels. À tout programme est donc associé une fonction, visiblement calculable en notre sens intuitif.

*On dit qu'une **fonction** sur les entiers naturels et à valeurs dans les entiers naturels est **récursive** si, et seulement si, il existe (au moins) un programme (au sens ci-dessus) permettant de la calculer.*

Remarques.- 1°) La restriction apparente concernant les entiers naturels n'est pas fondamentale. Les mots, par exemple, peuvent être représentés par des entiers naturels en utilisant un codage adéquat.

- 2°) Une fonction récursive peut être calculée par plusieurs programmes. Il y en a même toujours une infinité : il suffit, si on a un programme dans lequel le registre r_n n'est jamais utilisé, d'introduire, par exemple au début, autant de fois que l'on veut l'instruction :

$$r_n := r_1$$

- 3°) Les fonctions récursives sont visiblement calculables, mais est-ce que toutes les fonctions calculables sont récursives? Il est délicat d'y répondre puisqu'on est en présence, d'une part, d'une notion mathématique précise (celle de fonction récursive) et, d'autre part, d'une notion intuitive (celle de fonction calculable). Cependant on a tout lieu de penser que l'assertion suivante est vraie.

Thèse de Church (1936)

Les fonctions calculables sont les fonctions récursives.

Justification de la thèse de Church.- La thèse de Church est justifiée par les faits suivants :

- Dans les années 1930 plusieurs auteurs ont essayé de mathématiser la notion de fonction calculable.
- Ils sont arrivés à des résultats différents, certes, mais on est toujours arrivé à montrer que l'ensemble des fonctions correspondant à un modèle donné est identique à l'ensemble des fonctions correspondant à au moins un autre et donc à tous les autres.
- L'un des modèles, celui des *machines de Turing*, a réussi à convaincre entièrement.
- On s'aperçoit jour après jour que toute fonction intuitivement calculable est bien calculable sur ordinateur.

1.5 Historique

Nous avons déjà vu, lors de l'introduction à la programmation, deux moments clés de l'histoire de la notion d'ordinateur :

- le contexte, c'est-à-dire les besoins en calculs de plus en plus complexes, depuis la Préhistoire ;
- la création des bureaux de calculs pour effectuer les calculs complexes.

Nous allons voir maintenant deux autres moments clés :

- la réalisation de machines d'aide aux calculs ;
- la définition de ce qui peut être résolu algorithmiquement au milieu des années 1930, cristallisée par la thèse de Church-Turing.

A drawing of the Greek abacus in use. (Source: From an illustration on the "Darius" vase.)

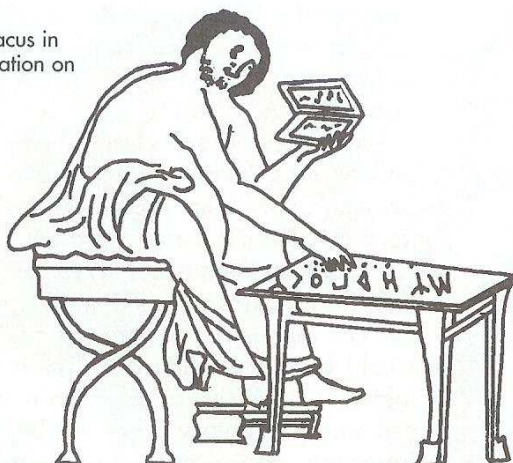


FIGURE 1.5 – Vase de Darius ([Wil-85], p. 56)

1.5.1 Machines d'aide aux calculs

Nous avons déjà signalé le livre [Wil-85] pour une initiation à l'historique des outils d'aide aux calculs, y compris les ordinateurs qui n'en sont qu'une étape, la plus évoluée. On pourra s'y référer.

1.5.1.1 Abaques

Des outils d'aide aux calculs apparurent dans l'Antiquité sous le nom d'**abaque**, ce qui a désigné des techniques différentes et dont l'histoire est en grande partie perdue (voir [Sch-01]).

DÉMOSTHÈNE (~384 av. J.C., ~322 av. J.C.) écrit ([227] et [229]) que nous avons besoin d'utiliser des cailloux pour effectuer les calculs qui sont trop difficiles à faire à la main.

HÉRODOTE ([Her], II 36) écrit à propos des Égyptiens : « *Ils écrivent leurs caractères et calculent avec des cailloux, de droite à gauche là où les Grecs le font de gauche à droite* ».

Le vase grec dit de Darius, trouvé en 1851 et maintenant conservé au musée de Naples, montre un trésorier tenant une tablette à la main alors qu'il semble manipuler des compteurs sur une table avec l'autre (figure 1.5, voir aussi une photo dans [Smi-25], vol. II, p. 161).

Une table d'abaque a été trouvée dans l'île de Salamis près du Pirée (figure 1.6).

A drawing of the Salamis abacus.
Note that the original stone has been
broken in two pieces.

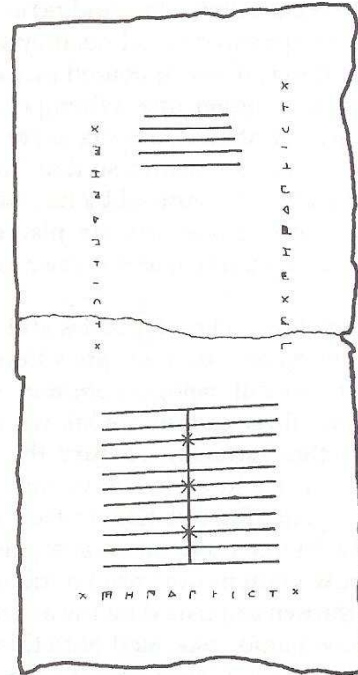


FIGURE 1.6 – Abaque de Salamis ([Wil-85], p. 57)

On sait que la manipulation des cailloux sur de la poussière, ou l'utilisation d'un doigt ou d'un stylet sur de la poussière fine ou du sable déposé sur une table, est utilisée depuis les temps les plus anciens. Le mot sémite *abaq* (poussière) semble être à l'origine du mot *abaque*. Les Grecs utilisaient le mot *abax* pour désigner une surface plane sur laquelle ils effectuaient leurs calculs. Le terme *abaque* a désigné plusieurs choses au cours de son histoire, y compris les bouliers au XIX^e et XX^e siècles.

1.5.1.2 Calculatrices mécaniques

On appelle **calculatrice** toute machine permettant d'effectuer rapidement l'une ou la totalité des quatre opérations sur les entiers naturels : addition, soustraction, multiplication et division.

Un bon résumé de l'histoire des calculatrices se trouve dans le livre *A history of Computing Technology* [Wil-85] de Michael WILLIAMS, avec des références bibliographiques. Nous nous contenterons ici d'en donner les grandes étapes.

- La détermination des impôts donne lieu à la naissance de la première machine à calculer mécanique, celle de Blaise PASCAL (1623–1662) en 1642, dont le père était fermier général, c'est-à-dire chargé de la détermination et du recouvrement des impôts. Celle-ci connaît un très grand intérêt de curiosité mais pas le succès commercial escompté.
- On a découvert tardivement que PASCAL a été devancé par Wilhem SCHICKARD (1592–1635), qui a certainement construit une calculatrice permettant les additions en 1623, mais qui est perdue.
- Gottfried Wilhem LEIBNIZ (1646–1716) construit une machine capable d'effectuer des multiplications en 1670.
- La première calculatrice à connaître un succès commercial, inspirée de la machine de Leibniz, est l'*arithmomètre* de Thomas DE COLMAR, mise sur le marché en 1820 et qui fut vendue jusqu'à la première guerre mondiale. D'autres machines, toutes mécaniques, furent commercialisées jusqu'au début des années 1970. Elles sont alors remplacées par les calettes électroniques, sous-produit des ordinateurs.

1.5.1.3 Machines dédiées

On appelle **machine dédiée** tout outil d'aide aux calculs conçu pour un besoin bien défini.

Il s'agit d'abord de *calculateurs analogiques* (et non numériques), tels que ceux qui servent à la prévision des marées (en particulier celui de lord KELVIN, qui a construit en 1878 un des premiers calculateurs analogiques, un prédicteur de marées mécanique appliquant les principes de l'analyse harmonique), qui sont décrits dans la première partie de [Gol-72]. Il s'agit ensuite de calculateurs mécaniques tels que la machine Z1 de Konrad ZUSE (1910–1995) construite en 1938 puis des machines électro-mécaniques (utilisant des relais) tels que le *Model I* de Georges STIBITZ aux établissements Bell pour effectuer des calculs sur les nombres complexes. Une première description de ces machines se trouve dans [Wil-85].

1.5.1.4 Calculateurs généralistes

On appelle **calculateur généraliste** tout outil d'aide aux calculs capable d'effectuer des calculs plus compliqués que les quatre opérations sans être une machine universelle (au sens où nous le verrons à la sous-section suivante).

Il s'agit des machines Z2, Z3 et Z4 de Konrad ZUSE construites avant la fin de la seconde guerre mondiale, des *Model II* à *Model VI* de Georges STIBITZ aux établissements Bell, des machines de Harvard de Howard AIKEN (*Mark I* à *Mark IV*) et des premiers calculateurs IBM. Viennent enfin les calculateurs électroniques telles que la machine ABC de John ATANASOFF (1903–1995) et Clifford BERRY (1918–1963) et surtout l'ENIAC de l'université de Pennsylvanie. Une première description de ces machines se trouve dans [Wil-85].

1.5.1.5 Machines universelles ou ordinateurs

*On appelle **machine universelle**, ou **ordinateur**, tout outil d'aide aux calculs capable d'effectuer tout calcul qui l'est par une machine, qu'importe la nature de celle-ci.*

Le premier ordinateur (conçu mais non construit) est la *machine analytique* de Charles BABAGE (1791–1871), entièrement mécanique. L'idée du premier ordinateur jamais réalisé, due essentiellement à John VON NEUMANN est l'EDVAC conçu en 1945. Mais à cause des dissensions au sein de l'équipe, il ne sera terminé qu'en 1950, après la mise en service de l'EDSAC de Cambridge en Angleterre (1949), conçu par Maurice WILKES. On pourra trouver une introduction à la naissance des ordinateurs, avec des références bibliographiques, dans [Wil-85].

Vint ensuite la commercialisation des ordinateurs, avec la toute puissance d'IBM durant une période, puis l'apparition des micro-ordinateurs.

1.5.2 Fonctions calculables

1.5.2.1 Premiers doutes sur les limites des calculs par programme

Les calculs apparaissent depuis la naissance des Cités pour des raisons de comptabilité (les quatre opérations arithmétiques) puis pour des raisons astronomiques (liées à la détermination des saisons pour l'agriculture). Apparaissent ensuite de plus en plus de programmes de calculs, non pas, historiquement, les programmes d'ordinateur mais les programmes dans les bureaux de calculs. Deux problèmes, cependant, vont inciter à s'interroger sur les limites de ceux-ci.

Attitude générale.- Pendant longtemps les mathématiciens ont résolu des problèmes particuliers par des méthodes générales, des méthodes souvent plus générales que ce pourquoi elles ont été recherchées. On s'interroge, par exemple, sur la contenance d'un tonneau de telle forme. La méthode la plus générale de l'Antiquité, appelée *méthode d'exhaustion*, consiste à deviner la formule (exacte) donnant le volume d'un solide d'une forme déterminée puis à la démontrer en approximant le solide par des polyèdres de plus en plus proches. Au XVII^e siècle, la méthode du calcul intégral de NEWTON et LEIBNIZ permet de calculer des volumes sans avoir d'idée *a priori* sur le résultat. On ne commence à s'interroger sur la notion générale de volume qu'au troisième tiers du XIX^e siècle.

Le XIX^e siècle est un siècle à la fois de rigueur et de classification. On ne s'intéresse plus à tel ou tel problème particulier, on veut formuler le cas général et obtenir des théorèmes généraux sur celui-ci.

Premier problème : théorie des invariants.- Considérons une courbe dans le plan, une surface dans l'espace ou l'analogue (une *hypersurface*) dans un espace de dimension arbitraire. Certains de ces objets sont des *variétés*, dont les coordonnées, dans un repère donné, vérifient une équation polynomiale. Lorsqu'on change de repère, l'équation de la variété change mais on peut mettre en évidence sur l'équation des propriétés invariantes par rapport à un changement de repère. Ces *invariants* traduisent les propriétés géométriques intrinsèques de la variété.

Une *variété algébrique* dans un espace de dimension n est définie, dans un repère donné, par $P(x_1, \dots, x_n) = 0$, où P est un polynôme homogène de degré m . Un *invariant* est un polynôme I en les coefficients des polynômes homogènes à n variables de degré m tel que pour toute transformation linéaire, de déterminant 1, on ait $I(a) = I(a')$, si a est la liste des coefficients de P et a' celle de l'équation P' après la transformation.

Le problème de la *théorie des invariants* consiste à déterminer les invariants de la variété algébrique.

En 1868 [Gor-68], Paul GORDAN (1837–1912) trouve une méthode pour déterminer les invariants d'une courbe algébrique plane. Pendant vingt ans, personne ne réussit à améliorer les résultats de GORDAN. En 1890, David HILBERT [Hil-90] donne une réponse en dimension quelconque, en montrant que, dans un espace de dimension donnée, il existe une famille finie qui engendre l'ensemble des invariants par des opérations simples (*Théorème de la base de Hilbert*). Cependant la démonstration ne donne pas de procédure pour calculer la famille génératrice. Cette voie abstraite rencontre une vive opposition. GORDAN déclare : « *Ce n'est pas des mathématiques, c'est de la théologie* » (cf. [McL-08]). HILBERT reprend le problème et, en 1893 [Hil-93], réussit à donner une démonstration constructive qui permet de calculer la famille génératrice des invariants³.

C'est certainement la première fois que l'on insiste sur l'effectivité ou calculabilité d'un problème.

3. Pour une introduction à l'œuvre de Hilbert, on pourra se reporter à [Cas-01].

Deuxième problème : équations différentielles et axiome du choix.- Le développement des mathématiques classiques est fortement lié à celui de la physique. Ce dont a besoin la physique, c'est la résolution d'équations différentielles. Pour les cas simples, on a pu trouver des solutions « exactes » à certaines d'entre elles, c'est-à-dire exprimable à l'aide des fonctions connues. Au XIX^e siècle, on commence à comprendre que l'on n'a pas toujours de « solution analytique » exacte. Les solutions approchées sont suffisantes pour la physique à condition de disposer de théorèmes sur certaines propriétés générales. Augustin CAUCHY (1789–1857) [Cau-1844], avec des hypothèses précisées par Rudolph LIPSCHITZ (1832–1903), montre l'existence et l'unicité de la solution d'une équation différentielle avec des conditions initiales données [Yus-81], ce qui reflète bien l'expérience physique.

Giuseppe PEANO (1858–1932) améliore le résultat de CAUCHY et LIPSCHITZ en montrant l'existence dans le seul cas de la continuité, mais sans unicité comme le montre l'exemple classique $y' = \sqrt[3]{x}$.

On s'aperçoit rapidement que PEANO utilise implicitement une hypothèse pour démontrer son résultat, hypothèse qui va être connue sous le nom d'*axiome du choix*. Un énoncé simple de celui-ci dit que le produit cartésien (infini) d'ensembles non vides est un ensemble non vide. Émile BOREL s'interroge sur la notion d'effectivité à propos de la justification de l'axiome du choix dans ses *Leçons sur les fonctions* de 1898 [Bor-98].

1.5.2.2 Retour en arrière : émergence des problèmes de possibilité

Au dix-neuvième siècle naît une nouvelle façon de répondre aux problèmes, le type de réponse étant inattendu pour un mathématicien de l'époque. Jusqu'alors, lorsqu'on posait un problème, on s'attendait à ce qu'il soit résolu, tôt ou tard, sous la forme sous laquelle il a été posé. Le nouveau type de réponse est : il n'existe pas de réponse au problème sous la forme sous laquelle il est posé.

Cette nouvelle attitude apparaît à propos de problèmes anciens sur la résolution des équations algébriques et sur le problème des constructions géométriques.

Le premier exemple en est donné par RUFFINI en 1799 [Ruf-04]. Les équations algébriques sont apparues très tôt pour des raisons diverses (mais non vraiment explicitées). La résolution des équations du second degré date de la Mésopotamie ancienne. La résolution des équations du troisième et du quatrième degré date du XVI^e siècle (TARTAGLIA, FERRARI...). On cherche alors activement à résoudre l'équation du cinquième degré, plus exactement par radicaux, c'est-à-dire que les solutions sont exprimées à partir des coefficients en utilisant les quatre opérations arithmétiques et le passage à la racine n -ième. RUFFINI montre que c'est impossible, ou tout au moins l'affirme puisque son article est considéré comme incompréhensible. Qu'importe puisque les démonstrations d'ABEL (en 1824) puis le théorème général de GALOIS de 1832 (qui va plus loin puisqu'indique dans quel cas une équation algébrique est résoluble par radicaux) confirment ce résultat.

Le second problème concerne les constructions géométriques à l'aide de la règle et du compas. L'Antiquité laisse trois problèmes non résolus à ce propos : la *duplication du cube* (en terme moderne il faut construire la racine cubique de 2), la *trisection d'un angle* quelconque (la construction de la bissectrice est connue depuis longtemps) et la *quadrature du cercle* (étant donné le rayon d'un cercle, autrement dit un segment de droite, construire le côté, un autre segment, d'un carré ayant même aire que le cercle). WANTZEL montre en 1837 [Wan-37] qu'il est impossible de résoudre les deux premiers problèmes : les longueurs des segments constructibles à l'aide de la règle et du compas par rapport à un segment de longueur un sont appelés les *nombre constructibles*; WANTZEL commence par montrer que les nombres constructibles sont ceux des

extensions quadratiques itérées du corps des rationnels ; il montre ensuite que ni $\sqrt[3]{3}$, ni $\sin(10^\circ)$ ne sont des nombres constructibles. LINDEMANN [Lin-82] conclut en montrant l'impossibilité de la quadrature du cercle en 1882, plus précisément en démontrant que le nombre π est transcendant, c'est-à-dire qu'il n'est solution d'aucune équation polynomiale à coefficients rationnels.

Beaucoup d'autres solutions de ce type apparaissent. David HILBERT y fait référence dans son célèbre exposé *Sur les problèmes futurs des Mathématiques* de 1900 :

Il se peut aussi que l'on s'efforce d'obtenir une solution en se basant sur des hypothèses insuffisantes ou mal comprises et que, par suite, on ne puisse atteindre le but. Il s'agit alors de démontrer l'impossibilité de résoudre le problème en se servant d'hypothèses telles qu'elles ont été données ou interprétées. Les Anciens nous ont donné les premiers exemples de pareilles démonstrations d'impossibilité ; ils ont démontré ainsi que dans un triangle rectangle isocèle l'hypoténuse et le côté de l'angle droit sont dans un rapport irrationnel. Dans les Mathématiques contemporaines, la question de l'impossibilité de certaines solutions joue un rôle prépondérant ; c'est à ce point de vue de la démonstration de l'impossibilité que d'anciens et difficiles problèmes, tels que ceux de la démonstration de l'axiome des parallèles, de la quadrature du cercle et de la résolution par radicaux de l'équation du cinquième degré, ont reçu une solution parfaitement satisfaisante et rigoureuse bien qu'en un sens tout différent de celui qu'on cherchait primitivement. Le fait remarquable dont nous venons de parler et certains raisonnements philosophiques ont fait naître en nous la conviction que partagera certainement tout mathématicien, mais que jusqu'ici personne n'a étayée d'aucune preuve, la conviction, dis-je, que tout problème mathématique déterminé doit être forcément susceptible d'une solution rigoureuse, que ce soit par une réponse directe à la question posée, ou bien par la démonstration de l'impossibilité de la résolution, c'est-à-dire de l'insuccès de toute tentative de résolution.

[Hil-00], p. 11–12.

Il est d'ailleurs curieux que, alors qu'il ait bien conscience de solutions inattendues, HILBERT ne remette absolument pas en doute les possibilités du calcul. L'une des preuves les plus flagrantes de cette dernière constatation est l'analyse faite par Yuri MATHIASSEVICH en 1999 de la façon dont David HILBERT pose son dixième problème (sur les mathématiques futures) en 1900 :

Le dixième problème occupe moins d'espace qu'aucun autre problème dans l'article de Hilbert. Durant son exposé il n'en dit pas un mot, ainsi que sur quelques autres problèmes. Son exposé dura deux heures et demi mais ceci ne fut pas suffisant pour présenter les vingt-trois problèmes ; ainsi quelques problèmes, dont le dixième, ne furent pas présentés oralement mais seulement inclus dans la version imprimée de l'exposé.

Ainsi Hilbert ne donna-t-il pas de motivation pour le dixième problème. Nous pouvons seulement deviner pourquoi il ne parla que des solutions en « entiers rationnels ». Nous avons vu que cela est équivalent à rechercher un algorithme pour résoudre les équations diophantiennes en entiers naturels. Mais, en fait, Diophante lui-même ne résolvait les équations ni en entiers naturels, ni en entiers relatifs, il cherchait des solutions rationnelles. Donc pourquoi Hilbert ne demanda-t-il pas une procédure pour déterminer l'existence de solutions rationnelles ? La réponse est plus ou moins évidente. Hilbert était optimiste et croyait en l'existence d'un algorithme pour résoudre les équations diophantiennes en entiers. Un tel algorithme nous permettrait également de résoudre les équations en rationnels.

[Mat-99], p. 297

Bien sûr on ne peut pas en déduire *a priori* à propos d'un problème particulier qu'HILBERT ne se posa jamais la question de la calculabilité. Mais en fait aucun des problèmes ne porte sur la calculabilité et on ne retrouve pas de trace de cette interrogation dans ses œuvres.

1.5.2.3 Le besoin de caractériser les fonctions calculables

Nous avons vu ci-dessus survenir quelques doutes sur la calculabilité de certaines opérations, en particulier par Émile BOREL, mais sans qu'il y ait de définition de ce qui est calculable. La nécessité de définir rigoureusement la *calculabilité* d'une fonction se fait ressentir à propos de deux problèmes : celui de la décidabilité de l'arithmétique élémentaire et l'énoncé du théorème d'incomplétude de GÖDEL.

Le problème de la décidabilité de l'arithmétique élémentaire.- Tout le monde se souvient des problèmes de Géométrie qu'il a rencontrés au collège. Il faut, suivant la capacité de chacun, plus ou moins de temps pour les résoudre. Certains sont même si difficiles qu'il faut attendre la solution du professeur. D'autres, encore plus difficiles, faisaient l'objet d'énoncés dans des revues telle que feue la *Revue de Mathématiques Élémentaires* en France.

Alfred TARSKI montre à la fin des années 1920 (publié tardivement [Tar-48, Tar-51]) qu'en fait il n'y a nul besoin de créativité pour résoudre ces problèmes. On peut automatiser la géométrie élémentaire, c'est-à-dire que l'on peut construire une machine à laquelle on donne l'énoncé et celle-ci en donne la réponse. Il s'agit d'un résultat théorique, une telle machine n'étant pas construite à l'époque. Ce n'est que dans les années 1970 qu'elle commencera à apparaître comme programme d'ordinateur.

Puisque la géométrie élémentaire est automatisable, il doit bien en être également ainsi de l'arithmétique élémentaire, se dit-on à l'époque. Des recherches sont alors effectuées dans ce sens, mais on s'aperçoit rapidement que le cas est plus coriace. On commence même à se dire qu'il ne doit pas en être ainsi. Mais comment le démontrer ? Et d'abord comment démontrer qu'un problème n'est pas automatisable. Il faudrait déjà préciser ce qu'est un problème automatisable.

Le théorème d'incomplétude de Gödel.- En 1931, Kurt GÖDEL répond à HILBERT (plus exactement à son propos que « *la conviction, dis-je, que tout problème mathématique déterminé doit être forcément susceptible d'une solution rigoureuse, que ce soit par une réponse directe à la question posée, ou bien par la démonstration de l'impossibilité de la résolution, c'est-à-dire de l'insuccès de toute tentative de résolution* ») par son célèbre théorème d'incomplétude : *quelle que soit la théorie mathématique considérée, il existe un énoncé du langage de celle-ci qui ne peut ni être démontré, ni réfuté dans cette théorie.*

GÖDEL [God-31] a besoin d'un minimum d'hypothèses sur ce qu'est une théorie mathématique pour démontrer son théorème. Une théorie mathématique est une théorie logique du premier ordre, avec un certain ensemble d'axiomes. Dans ce contexte, l'ensemble des axiomes ne peut pas être fini mais on doit savoir si un énoncé du langage de la théorie est un axiome ou non. Autrement dit on doit pouvoir décider si c'est un axiome.

Pour démontrer son théorème, GÖDEL donne lui-même une définition de ce qu'est une fonction calculable, inspirée, dit-il, d'une suggestion non publiée de Jacques HERBRAND. Il ne sera cependant lui-même convaincu qu'il a bien ainsi appréhendé la notion de fonction calculable qu'à l'apparition du modèle de TURING en 1936 (et de l'équivalence avec son modèle), comme il le dit dans un *postscriptum* à la réédition de son article dans [Dav-65] :

En conséquence d'avancées ultérieures, en particulier au fait que, dus aux travaux de A. M. Turing, une définition précise et adéquate sans discussion du concept général de système formel peut maintenant être donnée, l'existence de propositions arithmétiques indécidables et la non-prouvabilité de la consistance d'un système dans ce système lui-

même peuvent maintenant être démontrés rigoureusement pour tout système formel contenant une certaine quantité de théorie des nombres finitaire.

Les travaux de Turing donnent une analyse du concept de « procédure mécanique » (alias « algorithmique » ou « procédure calculatoire » ou « procédure combinatoire finie »). On a montré que ce concept est équivalent à celui de « machine de Turing ».

[Dav-65], p. 71

1.5.2.4 Caractérisation des fonctions calculables

Venons-en donc à la définition de TURING. Sous la direction de NEUMANN, Alan TURING passe de l'étude de la théorie des groupes à l'hypothèse de Riemann et, pour cela, au calcul des zéros de la fonction zêta. Les moyens théoriques sont hors de portée; il s'agit donc d'effectuer des calculs sur les décimaux (avec les erreurs inhérentes) pour obtenir des résultats sur les réels.

Il abandonne ce problème pour une réflexion globale sur les réels qui peuvent être calculables et plus généralement sur « ce qui est calculable ». Il répond à la question en 1936 [Tur-36] en définissant son célèbre modèle de machines abstraites. Sa réponse est immédiatement acceptée. La même année Alonzo CHURCH [Chu-36] clame que toute notion de calculabilité est réductible aux machines de Turing (ce qui est connu sous le nom de *thèse de Church*) :

Le fait que deux définitions très différentes et (dans l'opinion de l'auteur) également naturelles de la calculabilité effective soient équivalentes ajoute à la force des raisons données ci-dessous pour croire qu'elles constituent une caractérisation générale de cette notion cohérente avec la compréhension intuitive de celle-ci.

[Chu-36], note 3

Plusieurs autres modèles de calculabilité sont proposés, dont on montre facilement qu'ils sont tous équivalents à celui des machines de Turing.

1.5.2.5 Autres modèles et machines RAM

Plusieurs modèles théoriques de caractérisation des fonctions calculables ont été proposés, presque tous la même année 1936.

Modèle des fonctions récursives.- Comme nous l'avons déjà dit, le premier modèle est proposé par HERBRAND.

Kurt GÖDEL est le premier mathématicien à avoir besoin d'une caractérisation de l'ensemble des fonctions calculables. Il utilise une variante de celle de HERBRAND dans [God-31] mais sans être en être satisfait.

λ -calcul.- Alonzo CHURCH [Chu-33] et Stephen Cole KLEENE [Kle-35] proposent un second modèle en 1933, qui sera appelé plus tard λ -calcul (prononcez *lambda calcul*). L'équivalence avec le modèle de Herbrand-Gödel est due principalement à KLEENE [Kle-36] mais aussi partiellement à CHURCH et ROSSER [CR-36].

Machines de Turing.- Ce qu'on appelle de nos jours *machines de Turing* a été proposé par Alan TURING [Tur-36] et, indépendamment et de façon plus approfondie, par Emil POST [Pos-36]. L'équivalence avec la λ -définissabilité est esquissée dans l'appendice de l'article même de TURING puis prouvée dans [Tur-37].

Quelques variations Markov et les machines RAM

1.6 Bibliographie

- [Bor-98] BOREL, Émile, **Leçons sur la théorie des fonctions**, Gauthier-Villars, 1898, quatrième édition 1950, XIII + 295 p. Édition de 1898 numérisée par *Internet Archive*.
- [Cas-01] CASSOU-NOGUÈS, Pierre, **Hilbert**, Les Belles Lettres, 2001, 171 p.
- [Cau-1844] CAUCHY, Augustin, **Leçons de calcul différentiel et intégral rédigées principalement d'après les méthodes de M. A.-L. Cauchy par M. l'abbé Moigno**, tome 2 : Calcul intégral, 1^{re} partie, 1844, XLVIII + 783 p., 26^e leçon, pp. 385–396. Numérisé par *Google*.
- [Chu-33] CHURCH, Alonzo, *A Set of Postulates for the Foundation of Logic (Part II)*, **Annals of Mathematics**, vol. 34, 1933, pp. 839–864 (voir p. 863).
- [Chu-36] CHURCH, Alonzo, *An unsolvable problem of elementary number theory*, **American journal of mathematics**, vol. 58, 1936, pp. 345–363. Reprinted in [Dav-65], 1965, pp. 88–107.
- [CR-36] CHURCH, Alonzo et ROSSER, J. B., *Some properties of conversion*, **Trans. American Math. Soc.**, vol. 39, 1936, pp. 472–482.
- [Dav-65] DAVIS, Martin, **The undecidable : Basic papers on undecidable propositions, unsolvable problems and computable functions**, Raven press, New-York, 1965. Reed. Dover, 2004, 413 p., ISBN 0486432289.
- [God-31] GÖDEL, Kurt, *Über formal unentscheidbare Sätze der Principia mathematica und verwandter Systeme I*, **Monatshefte für Mathematik und Physik**, vol. 38, 1931, pp. 173–198.
Engl. tr. in [Van-67], 1967 and in **Collected Works**, vol.1, Oxford University Press, 1986.
Traduction française in **Le théorème de Gödel**, Seuil, 1989, 184 p.
- [Gol-72] GOLDSTINE, Herman H., **The computer : from Pascal to von Neumann**, XI + 378 p., Princeton University Press, 1972.
- [Gor-68] GORDAN, Paul, *Beweis, dass jede Covariante und Invariante einer binären Form eine ganze Function mit numerischen Coefficienten einer endlichen Anzahl solcher Formen ist*, **Journal für die reine und angewandte Mathematik**, vol. 69, 1868, pp. 323–354.
- [Her] HÉRODOTE, **Histoires, Livre II : Euterpe**, texte grec établi et traduit par Ph.-E. Legrand, Société des Belles-Lettres, 1936, 182 pages dédoublées.
- [Hil-90] HILBERT, David, *Über einen allgemeinen Gesichtspunkt für invariantentheoretische Untersuchung*, **Mathematische Annalen**, vol. 28, 1890, pp. 473–534.
- [Hil-93] HILBERT, David, *Über die vollen Invariantensysteme*, **Mathematische Annalen**, vol. 42, 1893, pp. 313–370.
- [Hil-00] HILBERT, David, *Mathematische Problem. Vortrag gehalten auf dem internationalen Mathematiker-Kongress zu Paris, 1900*, **Arch. der Math. und Physik**, vol. 1,

- 1900, pp. 44–63 et 213–237. Aussi *Nachrichten von der Königl. Gesellschaft der Wissenschaften zu Göttingen*, pp. 253–297 et David Hilbert, **Gesammelte Abhandlungen**, pp. 290–329, Springer, Berlin, 1935.
- Traduction française par L. LAUGEL avec corrections et additions *Sur les problèmes futurs des mathématiques* dans **Compte rendu du Deuxième congrès international des mathématiciens tenu à Paris du 6 au 12 août 1900**, Gauthier-Villars, 1902, pp. 58–114. Réédition J. Gabay, Sceaux, 1990.
- Traduction anglaise *Mathematical problems. Lecture delivered before the International Congress of Mathematicians at Paris in 1900*, **Bulletin of the American Mathematical Society**, vol. 8, pp. 437–479, 1902. Repris dans **Mathematical Developments Arising from Hilbert Problems**, volume 28 of *Proceedings of Symposia on Pure Mathematics*, pp. 1–34, American Mathematical Society, 1976.
- [Kle-35] KLEENE, Stephen Cole, *A Theory of Positive Integers in Formal Logic*, Ph.D. diss., Princeton University, September, 1933. Revised June 1934. **American Journal of Mathematics**, vol. 57 (1935), pp. 153–173 et 219–244 (voir p. 219).
- [Kle-36] KLEENE, Stephen Cole, *General recursive functions of natural numbers*, **Math. Annalen**, vol. 112, 1935–6, pp. 727–742. Rééd. in [Dav-65], pp. 236–253.
- [KU-58] KOLMOGOROV, A. N. and USPENSKII, V. A., *On the definition of an algorithm*, en russe, **Uspekhi Mat. Nauk**, vol. 13, pp. 3–28, 1958.
English translation in **AMS translations, 2nd series**, vol. 29, 1963, pp. 217–245.
- [Lin-82] LINDEMANN, C.-L., *Über die Zahl π* , **Mathematische Annalen**, vol. 20, 1882, pp. 213–225. Version numérisée disponible sur *The European Digital Mathematics Library*.
- [Mat-99] MATHIASÉVITCH, Yuri, *Le dixième problème de Hilbert : que peut-on faire avec des équations diophantiennes ?*, in Michel SERFATI, **La recherche de la vérité**, pp. 281–305, 1999, ACL – Les éditions du Kangourou. Disponible en ligne sur :
<http://www.kangmath.com/cite/confC01.html>
- [McL-08] McLARTY, Colin, *Theology and its discontents : the origin myth of modern mathematics*, 2008.
<http://www.institut.math.jussieu.fr/~harris/theology.pdf>
- [Pos-36] POST, Emil, *Finite combinatory processes – formulation 1*, **The Journal of Symbolic Logic**, vol. 1, 1936, pp. 103–105. Rééd. in [Dav-65], pp. 289–291.
- [Ruf-04] RUFFINI, P., *Sopra la determinazione della radici nelle equazioni numeriche di qualunque grado*, Modena, 1804. Réédition dans **Opere Matematiche**, t. II, Edizioni Cremonese, Rome, 1953.
- [Sch-01] SCHÄRLIG, Alain, **Compter avec des cailloux : le calcul élémentaire sur l’abaque chez les anciens Grecs**, Presses polytechniques et universitaires romandes, 2001, 339 p.
- [Smi-25] SMITH, David Eugene, **History of Mathematics**, Ginn and Company, 1925. Reed. Dover, 1958, vol. 1 XII + 596 p., vol. 2 XII + 725 p. Il existe une version électronique téléchargeable.

- [Tar-48] TARSKI, Alfred, **A decision method for elementary algebra and geometry**, Rand Corporation, Santa Monica, 1948, III + 60 p. Traduction française de la première édition (alors inédite) dans [Tar-74], 1974, pp. 203–242.
- [Tar-51] TARSKI, Alfred, **A decision method for elementary algebra and geometry**, 2nd ed., University California Press, Berkeley, 1951, III + 63 p.
- [Tar-74] TARSKI, Alfred, **Logique, sémantique, métamathématique 1923–1944**, vol. 2, Armand Colin, Paris, 1974.
- [Tur-36] TURING, Alan Mathison, *On computable numbers, with an application to the Entscheidungsproblem*, **Proceedings of the London Mathematical Society**, vol. 42, 1936, pp. 230–265, *Correction*, **ibid.**, vol. 43, pp. 544–546. Reprinted in [Dav-65], 1965, pp. 116–154. Tr. fr. in [Tur-95]
- [Tur-37] TURING, Alan Mathison, *Computability and λ -definability*, **The Journal of Symbolic Logic**, vol. 2, 1937, pp. 153–163.
- [Tur-95] TURING, Alan and GIRARD, Jean-Yves, **La machine de Turing**, Seuil, 1995, pp. 47–102.
- [Van-67] VAN HEIJENOORT, Jean, **From Frege to Gödel : A source book in mathematical logic, 1879–1931**, Harvard University Press, 1967, 4th printing, 1981, corrected.
- [Wan-37] WANTZEL, L., *Recherches sur les moyens de reconnaître si un problème de géométrie peut se résoudre avec la règle et le compas*, **Journal de Mathématiques Pures et Appliquées**, vol. 2, 1837, pp. 366–372. Disponible en ligne :
http://portail.mathdoc.fr/JMPA/afficher_notice.php?id=JMPA_1837_1_2_A31_0
- [Wil-85] WILLIAMS, Michael R., **A History of Computing Technology**, IEEE Computer Society Press, 1985, second edition, 1997, XI + 426 p.
- [Yus-81] YUSHKEVICH, A. P., *Sur les origines de la ‘méthode de Cauchy-Lipschitz’ dans la théorie des équations différentielles ordinaires*, **Rev. Histoire Sci. Appl.**, vol. 34 (3-4), 1981, pp. 209–215.

1.7 Exercices

Exercice 1.- (Soustraction)

La soustraction n'est pas une opération totale sur l'ensemble \mathbb{N} des entiers naturels. On parle de **soustraction complète** lorsqu'on prolonge cette opération par 0 pour un premier opérande plus petit que le second opérande.

Écrire un programme qui calcule la différence complète de l'entier naturel contenu dans le registre e_1 par celui contenu dans le registre e_2 et place le résultat dans le registre s_1 .

Exercice 2.- (Codage des mots par des entiers naturels)

Exercice 3.- (Génération des ensembles infinis)

Nous avons dit que nous avions de la chance car, bien que l'ensemble des opérations calculables par un ordinateur soit infini, toute opération calculable peut être engendrée à partir d'un ensemble fini par un nombre fini de constructeurs d'opérations.

- 1°) Tout ensemble infini est-il engendré à partir d'un sous-ensemble fini par un nombre fini de constructeurs ?

- 2°) Tout ensemble dénombrable est-il engendré à partir d'un sous-ensemble fini par un nombre fini de constructeurs ?

Exercice 4.- Nous avons vu que le nombre de programmes et le nombre de fonctions calculables sont infinis.

Se pourrait-il que le nombre de programmes soit infini alors que le nombre de fonctions calculables soit fini ?