MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

U. S. T. A.

ECOLE NATIONALE POLYTECHNIQUE

DEPARTEMENT ELECTRONIQUE ET ELECTROTECHNIQUE

PROJET DE FIN D'ETUDES

THESE D'INGENIORAT

RESOLUTION
d'EQUATIONS DIFFERENTIELLES
par **MICROPROCESSEUR**

Proposé par :

H. TEDJINI

Docteur Ingénieur

Etudié par :

A. CHERIFI

A. DELASSI

...VIER 1981

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

U. S. T. A.

ECOLE NATIONALE POLYTECHNIQUE

DEPARTEMENT ELECTRONIQUE ET ELECTROTECHNIQUE

PROJET DE FIN D'ETUDES

THESE D'INGENIORAT

RESOLUTION d'EQUATIONS DIFFERENTIELLES par MICROPROCESSEUR

Proposé par :

H. TEDJINI

Docteur Ingénieur

Etudié par :

A. CHERIFI

A. DELASSI

JANVIER 1981

D E D I C A C E S

§§§§§§§§§§§§§§§§§§§§

- A mon PERE
- A ma MERE
- A toute ma FAMILLE
- ET à mes AMIS

C.ABDELKADER

- A la memoire de mon PERE
- A ma MERE
- A mon oncle BACHIR
- A toute ma FAMILLE et mes AMIS.

D.AHMED

REMERCIEMENTS

§§§§§§§§§§§§§§§§§§§§§§§§§§

Nous remercions Monsieur B. SANSAL responsable de la division " SIMULATION ET CONTROLE " au C.S.T.N. où notre projet de fin d'etudes a pris forme.

Notre profonde reconnaissance à Monsieur H.TEDJINI ,Docteur Ingenieur et chargé de cours à l'Ecole Nationale Polytechnique d'Alger, pour nous avoir conseillé tout au long de notre travail.

Nous remercions MM. :A.BOURKEB et R.OUIGUINI pour leur contribution,ainsi que toute l'equipe de la division V

Nos remerciements vont egalement à MM. :

M.KADI ,AAZIB etABDI qui se sont occupés respectivement de la frappe et du tirage.

Que tous ceux qui ont contribués ,de prés ou de loin à notre formation trouvent dans ce modeste mémoire l'expression de notre gratitude.

§§§§§§§§§§§§§§§§§§

§§§§§§§§§§

§§§

§

S O M M A I R E

§§§§§§§§§§§§§§§§

§§§§§§§§§

I N T R O D U C T I O N

§§§§§§§§§§§§§§§§§§§§§§

Le microprocesseur est le plus important développement de l'electronique au cours de l'actuelle décennie, mais aussi, par ces applications, une véritable révolution dans tous les domaines d'activite.La compréhension du fonctionnement et des méthodes de mise en oeuvre des systemes à microprocesseurs est donc devenue capitale pour tous les techniciens et ingénieurs .

Les prouesses de calcul des microprocesseurs, l'augmentation de la compléxité des problémes téchnologiques actuels à savoir l'accroissement de la production et les considérations économiques font des calculateurs digitaux un outil de recherche inestimable .Depuis l'arrivée de cette machine trés compléxe, elle a été toujours d'un intérêt majeur pour les ingénieurs et scientists dans l'application de la simulation digitale pour les problèmes insolubles par des méthodes classiques .

La simulation digitale consiste à :

-definir d'abord les informations caractéristiques de l'evolution dynamique du système physique .

-La détèrmination du modèle mathématique qui décrit éxactement ce système régit génèralement par une ou plusieurs équations différentielles .

-Développer une approximation discrête du système réel en solutionnant les équations différentielles préalablement programmées pour l'ordinateur .

Ce dernier point de la simulation est développé dans cet ouvrage débutant par un exposé sommaire du logiciel du microprocesseur MC 6800 et d'une explication détaillée de la Bibliothèque Mathématiques utilisée par la résolution de l'equation différentielle, qui rentre dans le cadre du projet "SIMULATION ET CONTROLE D'UN REACTEUR NUCLEAIRE" de la division V du centre des Sciences et Technologie Nucléaire(C.S.T.N.).

CHAPITRE I

GENERALITES SUR LE MC 6800

I : A - CONFIGURATION INTERNE DU M6800

I) Les registres internes : (voir figure 1)

Les registres sont de petites mémoires qui permettent de conserver temporairement des informations. Tous les microprocesseurs possèdent des registres qui leurs sont propres et qui facilitent beaucoup la programmation à l'utilisateur.
Ces registres contribuent à diminuer notablement le nombre d'accès à la mémoire et par conséquent le nombre d'allées et retours des informations.

1°) Les accumulateurs A et B :

Les 2 accumulateurs sont des registres privilégies. De la taille des mots à traiter, c'est à dire 8 bits chacun, ils sont constitués de 8 bascules à accès parallèle.
L'accumulateur stocke temporairement les données (opérandes) traitées par l'unité arithmétique et logique (U.A.L). Bien entendu, seule la dernière information stockée dans l'accumulateur est conservée.
L'utilisateur a accès aux accumulateurs, par conséquent, il a la possibilité de :

- stocker un mot binaire directement dans l'accumulateur
- lire le contenu de l'accumulateur
- transférer un mot contenu dans l'accumulateur vers une adresse mémoire et inversement.

2°) Le registre d'index :

C'est un registre de 16 bits destiné à contenir une adresse mémoire souvent utilisée dans le mode d'adressage indexé.

3°) Le registre d'instruction : R.I.

Il est d'une longueur correspondante à 2 fois le format des mots binaires en mémoire, c'est à dire 16 bits. Il est donc nécessaire d'effectuer 2 aller et retour en mémoire pour disposer de l'instruction

.../...

complète. Cette dernière passe ensuite dans le registre à décodage.

4°) Le compteur de programme (C.P.)

C'est un registre de 16 bits, son contenu représente l'adresse de la case mémoire contenant la première instruction du programme. Lors de l'accès mémoire, le contenu de ce compteur est transmis dans le registre d'adresses, le compteur de programme est ainsi à nouveau libre peut être incrémenté (Plus 1) de façon à permettre l'adressage de la case mémoire suivante.

5°) Le registre d'état : C.C.R. (Code condition register)

composé de 8 bits et contient 6 informations utiles à la gestion du programme.

- C (bit indicateur de retenue) : nous renseigne si le mot considéré contient une retenue au 7ème bit.

- V (bit indicateur de dépassement) : Il indique à l'utilisateur que l'on a dépassé la capacité de traitement de l'U.A.L.

- Z (bit indicateur de zéro) : Ce bit signale si le résultat d'une opération effectuée par l'U.A.L est nul. Il sera à 1 si le résultat est nul et à 0 dans le cas contraire.

- N (bit indicateur désigne) : Un (0) indique que le résultat est positif et un (1) indique que le résultat est négatif.

- I (bit indicateur d'interruption) : Ce bit est positionné en présence d'un signal d'interruption.

- H (bit indicateur de demi-retenue) : Le mot considéré contient une retenue au 3eme bit.

II - Les instructions du M6800 : (Voir tableaux 1,3,4)

Le MC 6800 possède un jeu de 72 instructions d'une longueur variable de 1 à 3 octets, permettant d'effectuer les opérations suivantes :

.../...

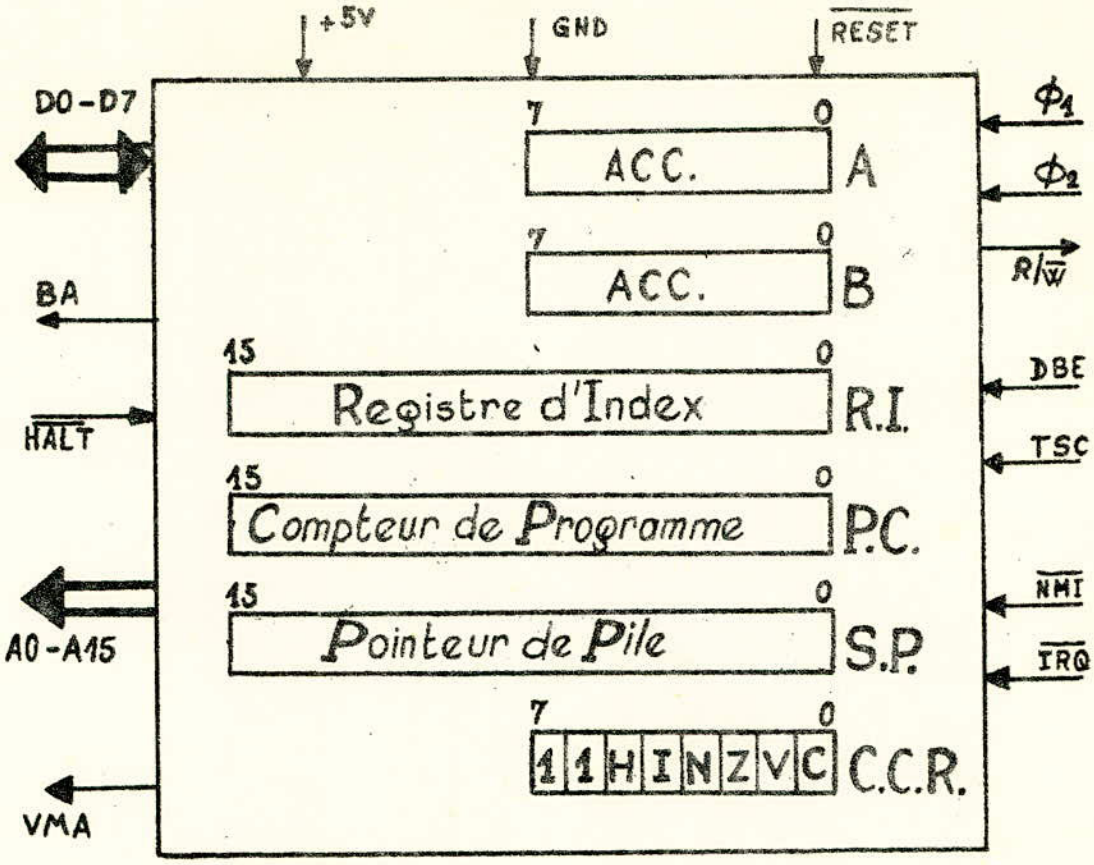


Fig 1 : Configuration des registres

Tableau 1 :
Instructions du M6800

ABA	Addition de l'accumulateur B à l'accumulateur A
ADC	Addition avec retenue
ADD	Addition
AND	"ET" logique
ASL	Décalage arithmétique de un vers la gauche
ASR	Décalage arihtmétique de un vers la droite
BCC	Branchement s'il n'y a pas de retenue
BCS	Branchement s'il y a retenue
BEQ	Branchement si égal (à zéro)
BGE	Branchement si supérieur ou égal à zéro
BGT	Branchement si plus grand que zéro
BHI	Branchement si supérieur
BIT	Test de bits
BLE	Branchement si inférieur ou égal à zéro
BLS	Branchement si inférieur ou égal
BLT	Branchement si inférieur à zéro
BMI	Branchement si négatif
BNE	Branchement si non nul
BPL	Branchement si positif ou nul
BRA	Branchement inconditionnal
BSR	Branchement à un sous-programme
BVC	Branchement si pas de dépassement
BVS	Branchement si dépassement
CBA	Comparaison des accumulateurs
CLC	Mise à zéro du bit de retenue
CLI	Mise à zéro du masque d'interruption
CLR	Mise à zéro
CLV	Mise à zéro du bit de dépassem. en compl. à deux
CMP	Comparaison
COM	Complément à un
CPX	Comparaison du registre d'index
DAA	Ajustement décimal sur l'accumulateur A
DEC	Décrémentation
DES	Décrémentation du pointeur de pile
DEX	Décrémentation du registre d'index
EOR	"OU" exclusif
INC	Incrémentation
INS	Incrémentation du pointeur de pile
INX	Incrémentation du registre d'index
JMP	Saut inconditionnel
JSR	Saut à un sous-programme
LDA	Chargement accumulateur
LDS	Chargement du pointeur de pile
LDX	Chargement du registre d'index
LSR	Décalage logique vers le droite d'une position
NEG	Complément à deux (opposé)
NOP	Passage en séquence (non opération)
ORA	"OU" logique
PSH	Mise d'un octet dans la pile
PUL	Dépliement d'un octet
ROL	Décalage circulaire à gauche
ROR	Décalage circulaire à droite
RTI	Retour de séquence d'interruption
RTS	Retour de sous-programme
SBA	Soustraction entre accumulateurs
SBC	Soustraction avec retenue
SEC	Mise à un de la retenue
SEI	Mise à un du masque d'interruption
SEV	Mise à un du bit de dépassement en complément à deux
STA	Mise en mémoire d'un accumulateur
STS	Mise en mémoire du pointeur de pile
STX	Mise en mémoire du registre d'index
SUB	Soustraction
SWI	Interruption programmée
TAB	Transfert de l'accumulateur A dans l'accumulateur B
TAP	Transfert de l'accumulateur A dans le Registre Codes Conditions
TBA	Transfert de l'accumulateur B dans l'accumulateur A
TPA	Transfert du registre d'Etat dans l'accumulateur A
TST	Test
TSX	Transfert du pointeur de pile dans l'Index
TXS	Transfert du registre d'index dans le pointeur de pile
WAI	Attente d'interruption

Tableau : 2

ACCUMULATOR AND MEMORY		ADDRESSING MODES														BOOLEAN/ARITHMETIC	COND. CODE REG					
		IMMED			DIRECT			INDEX			EXTND			INHER			OPERATION	5	4	3	2	1	0
OPERATIONS	MNEMONIC	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	(All register labels refer to contents)	H	I	N	Z	V	C
Add	ADDA	8B	2	2	9B	3	2	AB	5	2	BB	4	3				A + M → A	↕	•	↕	↕	↕	↕
	ADDB	CB	2	2	DB	3	2	EB	5	2	FB	4	3				B + M → B	↕	•	↕	↕	↕	↕
Add Acmltrs	ABA													1B	2	1	A + B → A	↕	•	↕	↕	↕	↕
Add with Carry	ADCA	89	2	2	99	3	2	A9	5	2	B9	4	3				A + M + C → A	↕	•	↕	↕	↕	↕
	ADCB	C9	2	2	D9	3	2	E9	5	2	F9	4	3				B + M + C → B	↕	•	↕	↕	↕	↕
And	ANDA	84	2	2	94	3	2	A4	5	2	B4	4	3				A · M → A	•	•	↕	↕	R	•
	ANDB	C4	2	2	D4	3	2	E4	5	2	F4	4	3				B · M → B	•	•	↕	↕	R	•
Bit Test	BITA	85	2	2	95	3	2	A5	5	2	B5	4	3				A · M	•	•	↕	↕	R	•
	BITB	C5	2	2	D5	3	2	E5	5	2	F5	4	3				B · M	•	•	↕	↕	R	•
Clear	CLR							6F	7	2	7F	6	3				00 → M	•	•	R	S	R	R
	CLRA													4F	2	1	00 → A	•	•	R	S	R	R
	CLRB													5F	2	1	00 → B	•	•	R	S	R	R
Compare	CMPA	81	2	2	91	3	2	A1	5	2	B1	4	3				A − M	•	↕	↕	↕	↕	↕
	CMPB	C1	2	2	D1	3	2	E1	5	2	F1	4	3				B − M	•	•	↕	↕	↕	↕
Compare Acmltrs	CBA													11	2	1	A − B	•	•	↕	↕	↕	↕
Complement, 1's	COM							63	7	2	73	6	3				M̄ → M	•	•	↕	↕	R	S
	COMA													43	2	1	Ā → A	•	•	↕	↕	R	S
	COMB													53	2	1	B̄ → B	•	•	↕	↕	R	S
Complement, 2's (Negate)	NEG							60	7	2	70	6	3				00 − M → M	•	•	↕	↕	①	②
	NEGA													40	2	1	00 − A → A	•	•	↕	↕	①	②
	NEGB													50	2	1	00 − B → B	•	•	↕	↕	①	②
Decimal Adjust, A	DAA													19	2	1	Converts Binary Add. of BCD Characters into BCD Format	•	•	↕	↕	↕	③
Decrement	DEC							6A	7	2	7A	6	3				M − 1 → M	•	•	↕	↕	④	•
	DECA													4A	2	1	A − 1 → A	•	•	↕	↕	④	•
	DECB													5A	2	1	B − 1 → B	•	•	↕	↕	④	•
Exclusive OR	EORA	88	2	2	98	3	2	A8	5	2	B8	4	3				A⊕M → A	•	•	↕	↕	R	•
	EORB	C8	2	2	D8	3	2	E8	5	2	F8	4	3				B⊕M → B	•	•	↕	↕	R	•
Increment	INC							6C	7	2	7C	6	3				M + 1 → M	•	•	↕	↕	⑤	•
	INCA													4C	2	1	A + 1 → A	•	•	↕	↕	⑤	•
	INCB													5C	2	1	B + 1 → B	•	•	↕	↕	⑤	•
Load Acmltr	LDAA	86	2	2	96	3	2	A6	5	2	B6	4	3				M → A	•	•	↕	↕	R	•
	LDAB	C6	2	2	D6	3	2	E6	5	2	F6	4	3				M → B	•	•	↕	↕	R	•
Or, Inclusive	ORAA	8A	2	2	9A	3	2	AA	5	2	BA	4	3				A + M → A	•	•	↕	↕	R	•
	ORAB	CA	2	2	DA	3	2	EA	5	2	FA	4	3				B + M → B	•	•	↕	↕	R	•
Push Data	PSHA													36	4	1	A → M_{SP}, SP − 1 → SP	•	•	•	•	•	•
	PSHB													37	4	1	B → M_{SP}, SP − 1 → SP	•	•	•	•	•	•
Pull Data	PULA													32	4	1	SP + 1 → SP, M_{SP} → A	•	•	•	•	•	•
	PULB													33	4	1	SP + 1 → SP, M_{SP} → B	•	•	•	•	•	•
Rotate Left	ROL							69	7	2	79	6	3	M				•	•	↕	↕	⑥	↕
	ROLA													49	2	1	A	•	•	↕	↕	⑥	↕
	ROLB													59	2	1	B	•	•	↕	↕	⑥	↕
Rotate Right	ROR							66	7	2	76	6	3	M				•	•	↕	↕	⑥	↕
	RORA													46	2	1	A	•	•	↕	↕	⑥	↕
	RORB													56	2	1	B	•	•	↕	↕	⑥	↕
Shift Left, Arithmetic	ASL							68	7	2	78	6	3	M				•	•	↕	↕	⑥	↕
	ASLA													48	2	1	A	•	•	↕	↕	⑥	↕
	ASLB													58	2	1	B	•	•	↕	↕	⑥	↕
Shift Right, Arithmetic	ASR							67	7	2	77	6	3	M				•	•	↕	↕	⑥	↕
	ASRA													47	2	1	A	•	•	↕	↕	⑥	↕
	ASRB													57	2	1	B	•	•	↕	↕	⑥	↕
Shift Right, Logic	LSR							64	7	2	74	6	3	M				•	•	R	↕	⑥	↕
	LSRA													44	2	1	A	•	•	R	↕	⑥	↕
	LSRB													54	2	1	B	•	•	R	↕	⑥	↕
Store Acmltr	STAA				97	4	2	A7	6	2	B7	5	3				A → M	•	•	↕	↕	R	•
	STAB				D7	4	2	E7	6	2	F7	5	3				B → M	•	•	↕	↕	R	•
Subtract	SUBA	80	2	2	90	3	2	A0	5	2	B0	4	3				A − M → A	•	•	↕	↕	↕	↕
	SUBB	C0	2	2	D0	3	2	E0	5	2	F0	4	3				B − M → B	•	•	↕	↕	↕	↕
Subtract Acmltrs	SBA													10	2	1	A − B → A	•	•	↕	↕	↕	↕
Subtr. with Carry	SBCA	82	2	2	92	3	2	A2	5	2	B2	4	3				A − M − C → A	•	•	↕	↕	↕	↕
	SBCB	C2	2	2	D2	3	2	E2	5	2	F2	4	3				B − M − C → B	•	•	↕	↕	↕	↕
Transfer Acmltrs	TAB													16	2	1	A → B	•	•	↕	↕	R	•
	TBA													17	2	1	B → A	•	•	↕	↕	R	•
Test, Zero or Minus	TST							6D	7	2	7D	6	3				M − 00	•	•	↕	↕	R	R
	TSTA													4D	2	1	A − 00	•	•	↕	↕	R	R
	TSTB													5D	2	1	B − 00	•	•	↕	↕	R	R

Code d'instruction du 6800 (1)

Tableau :3

INDEX REGISTER AND STACK POINTER OPERATIONS	MNEMONIC	IMMED OP	IMMED ~	IMMED #	DIRECT OP	DIRECT ~	DIRECT #	INDEX OP	INDEX ~	INDEX #	EXTND OP	EXTND ~	EXTND #	INHER OP	INHER ~	INHER #	BOOLEAN/ARITHMETIC OPERATION	5 H	4 I	3 N	2 Z	1 V	0 C
Compare Index Reg	CPX	8C	3	3	9C	4	2	AC	6	2	BC	5	3				$(X_H/X_L) - (M/M+1)$	•	•	⑦	↕	⑧	•
Decrement Index Reg	DEX													09	4	1	$X - 1 \rightarrow X$	•	•	•	↕	•	•
Decrement Stack Pntr	DES													34	4	1	$SP - 1 \rightarrow SP$	•	•	•	•	•	•
Increment Index Reg	INX													08	4	1	$X + 1 \rightarrow X$	•	•	•	↕	•	•
Increment Stack Pntr	INS													31	4	1	$SP + 1 \rightarrow SP$	•	•	•	•	•	•
Load Index Reg	LDX	CE	3	3	DE	4	2	EF	6	2	FE	5	3				$M \rightarrow X_H$, $(M+1) \rightarrow X_L$	•	•	⑤	↕	R	•
Load Stack Pntr	LDS	8E	3	3	9E	4	2	AE	6	2	BE	5	3				$M \rightarrow SP_H$, $(M+1) \rightarrow SP_L$	•	•	⑤	↕	R	•
Store Index Reg	STX				DF	5	2	EF	7	2	FF	6	3				$X_H \rightarrow M$, $X_L \rightarrow (M+1)$	•	•	⑤	↕	R	•
Store Stack Pntr	STS				9F	5	2	AF	7	2	BF	6	3				$SP_H \rightarrow M$, $SP_L \rightarrow (M+1)$	•	•	⑤	↕	R	•
Indx Reg → Stack Pntr	TXS													35	4	1	$X - 1 \rightarrow SP$	•	•	•	•	•	•
Stack Pntr → Indx Reg	TSX													30	4	1	$SP + 1 \rightarrow X$	•	•	•	•	•	•

JUMP AND BRANCH OPERATIONS	MNEMONIC	RELATIVE OP	RELATIVE ~	RELATIVE #	INDEX OP	INDEX ~	INDEX #	EXTND OP	EXTND ~	EXTND #	INHER OP	INHER ~	INHER #	BRANCH TEST	5 H	4 I	3 N	2 Z	1 V	0 C
Branch Always	BRA	20	4	2										None	•	•	•	•	•	•
Branch If Carry Clear	BCC	24	4	2										C = 0	•	•	•	•	•	•
Branch If Carry Set	BCS	25	4	2										C = 1	•	•	•	•	•	•
Branch If = Zero	BEQ	27	4	2										Z = 1	•	•	•	•	•	•
Branch If ≥ Zero	BGE	2C	4	2										$N \oplus V = 0$	•	•	•	•	•	•
Branch If > Zero	BGT	2E	4	2										$Z + (N \oplus V) = 0$	•	•	•	•	•	•
Branch If Higher	BHI	22	4	2										C + Z = 0	•	•	•	•	•	•
Branch If ≤ Zero	BLE	2F	4	2										$Z + (N \oplus V) = 1$	•	•	•	•	•	•
Branch If Lower Or Same	BLS	23	4	2										C + Z = 1	•	•	•	•	•	•
Branch If < Zero	BLT	2D	4	2										$N \oplus V = 1$	•	•	•	•	•	•
Branch If Minus	BMI	2B	4	2										N = 1	•	•	•	•	•	•
Branch If Not Equal Zero	BNE	26	4	2										Z = 0	•	•	•	•	•	•
Branch If Overflow Clear	BVC	28	4	2										V = 0	•	•	•	•	•	•
Branch If Overflow Set	BVS	29	4	2										V = 1	•	•	•	•	•	•
Branch If Plus	BPL	2A	4	2										N = 0	•	•	•	•	•	•
Branch To Subroutine	BSR	8D	8	2											•	•	•	•	•	•
Jump	JMP				6E	4	2	7E	3	3				} See Special Operations	•	•	•	•	•	•
Jump To Subroutine	JSR				AD	8	2	BD	9	3					•	•	•	•	•	•
No Operation	NOP										01	2	1	Ad Prog Cntr. Only	•	•	•	•	•	•
Return From Interrupt	RTI										3B	10	1		⑩					
Return From Subroutine	RTS										39	5	1	} See special Operation	•	•	•	•	•	•
Software Interrupt	SWI										3F	12	1		•	S	•	•	•	•
Wait for Interrupt	WAI										3E	9	1		•	⑪	•	•	•	•

CONDITION CODE REGISTER OPERATIONS	MNEMONIC	INHER OP	INHER ~	INHER #	BOOLEAN OPERATION	5 H	4 I	3 N	2 Z	1 V	0 C
Clear Carry	CLC	0C	2	1	$0 \rightarrow C$	•	•	•	•	•	R
Clear Interrupt Mask	CLI	0E	2	1	$0 \rightarrow I$	•	R	•	•	•	•
Clear Overflow	CLV	0A	2	1	$0 \rightarrow V$	•	•	•	•	R	•
Set Carry	SEC	0D	2	1	$1 \rightarrow C$	•	•	•	•	•	S
Set Interrupt Mask	SEI	0F	2	1	$1 \rightarrow I$	•	S	•	•	•	•
Set Overflow	SEV	0B	2	1	$1 \rightarrow V$	•	•	•	•	S	•
Acmltr A → CCR	TAP	06	2	1	$A \rightarrow CCR$	———			⑫		———
CCR → Acmlt A	TPA	07	2	1	$CCR \rightarrow A$	•	•	•	•	•	•

LEGEND

OP	Operation Code (Hexadecimal)	00	Byte = Zero
~	Number of MPU Cycles	H	Half carry from bit 3
#	Number of Program Bytes	I	Interrupt mask
+	Arithmetic Plus	N	Negative (sign bit)
−	Arithmetic Minus	Z	Zero (byte)
·	Boolean AND	V	Overflow, 2's complement
M_{SP}	Contents of memory location pointed to be Stack Pointer	C	Carry from bit 7
		R	Reset Always
+	Boolean Inclusive OR	S	Set Always
⊕	Boolean Exclusive OR	↕	Test and set if true cleared otherwise
$\overline{M}$	Complement of M	•	Not Affected
→	Transfer Into	CCR	Condition Code Register
0	Bit = Zero	LS	Least Significant
		MS	Most Significant

CONDITION CODE REGISTER NOTES

(Bit set if test is true and cleared other wise)

① (Bit V) Test Result = 10000000 ?

② (Bit C) Test Result = 00000000 ?

③ (Bit C) Test Decimal value of most significant BCD Character greater than nine ? (Not cleared if previously set)

④ (Bit V) Test Operand = 10000000 prior to execution ?

⑤ (Bit V) Test Operand = 01111111 prior to execution ?

⑥ (BIT V) Test Set equal to result of N⊕C after shift has occured

⑦ (Bit N) Test Sign bit of most significant (MS) byte of result = 1 ?

⑧ (Bit V) Test 2's complement overflow from subtraction of LS bytes ?

⑨ (Bit N) Test Result less than zero ? (Bit 15 = 1)

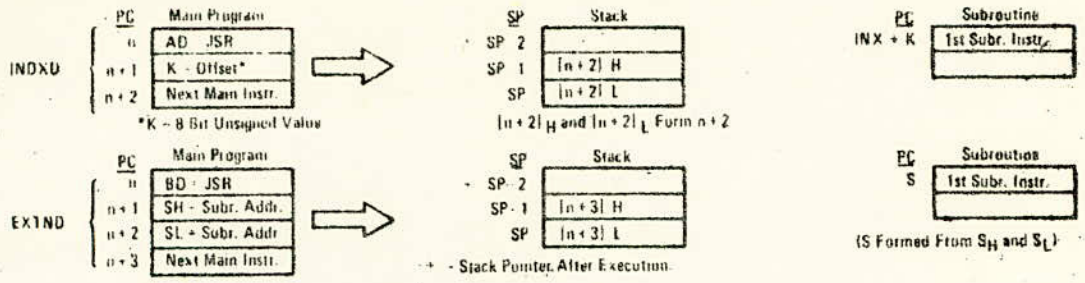
⑩ (All) Load Condition Code Register from Stack (See Special Operations)

⑪ (Bit I) Set when interrupt occurs. If previously set, a Non Maskable interrupt is required to exit the wait state

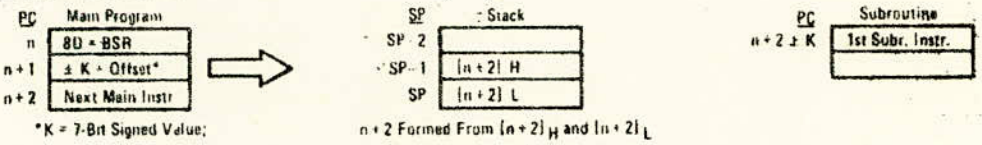
⑫ (ALL) Set according to the contents of Accumulator A

SPECIAL OPERATIONS

JSR, JUMP TO SUBROUTINE:

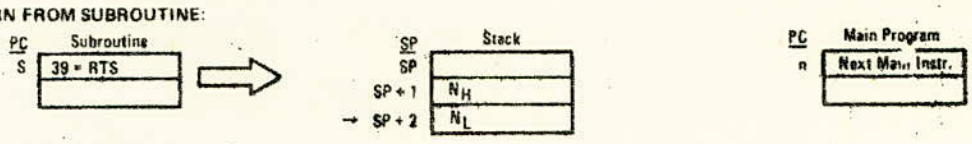


BSR, BRANCH TO SUBROUTINE:



JMP, JUMP:

RTS, RETURN FROM SUBROUTINE:



RTI, RETURN FROM INTERRUPT:

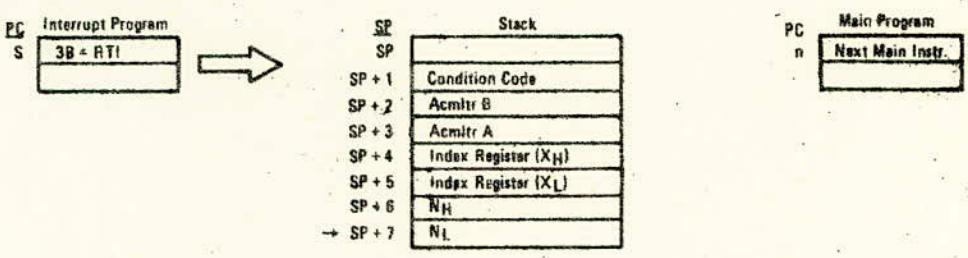


TABLE 6 – CONDITION CODE REGISTER MANIPULATION INSTRUCTIONS

OPERATIONS	MNEMONIC	IMPLIED OP	~	#	BOOLEAN OPERATION	COND. CODE REG. 5 H	4 I	3 N	2 Z	1 V	0 C
Clear Carry	CLC	0C	2	1	0 → C	•	•	•	•	•	R
Clear Interrupt Mask	CLI	0E	2	1	0 → I	•	R	•	•	•	•
Clear Overflow	CLV	0A	2	1	0 → V	•	•	•	•	R	•
Set Carry	SEC	0D	2	1	1 → C	•	•	•	•	•	S
Set Interrupt Mask	SEI	0F	2	1	1 → I	•	S	•	•	•	•
Set Overflow	SEV	0B	2	1	1 → V	•	•	•	•	S	•
Acmltr A → CCR	TAP	06	2	1	A → CCR	——— (12) ———					
CCR → Acmltr A	TPA	07	2	1	CCR → A	•	•	•	•	•	•

CONDITION CODE REGISTER NOTES: (Bit set if test is true and cleared otherwise)

1 (Bit V) Test: Result = 10000000?
2 (Bit C) Test: Result = 00000000?
3 (Bit C) Test: Decimal value of most significant BCD Character greater than nine? (Not cleared if previously set.)
4 (Bit V) Test: Operand = 10000000 prior to execution?
5 (Bit V) Test: Operand = 01111111 prior to execution?
6 (Bit V) Test: Set equal to result of N⊕C after shift has occurred.
7 (Bit N) Test: Sign bit of most significant (MS) byte = 1?
8 (Bit V) Test: 2's complement overflow from subtraction of MS bytes?
9 (Bit N) Test: Result less than zero? (Bit 15 = 1)
10 (All) Load Condition Code Register from Stack. (See Special Operations)
11 (Bit I) Set when interrupt occurs. If previously set, a Non-Maskable Interrupt is required to exit the wait state.
12 (All) Set according to the contents of Accumulator A.

- Arithmétiques (binaires et décimales)
- Logiques (ET, OU, ...)
- Décalages
- Chargement
- Stockage
- Branchements conditionnels et inconditionnels (Bouche, saut...)
- Instructions relatives aux interruptions (SWI,RII...)
- Manipulation dans la pile.

Ces instructions sont classées en quatre catégories suivant qu'elles agissent sur :

1°) Les accumulateurs et les mémoires

2°) Les registres RI (Instructions) ou SP (pointeur de pile)

3°) Le registre d'état C.C.R.(code condition register)

4°) Les instructions de branchement et de saut.

L'éxécution d'une instruction se réalise en deux temps successifs :

- Recherche de l'instruction qui consiste à lire en mémoire l'instruction à éxécuter.
- L'éxécution de cette instruction.

Le temps d'éxécution d'une instruction est plus ou moins long suivant la complexité de celle-ci. Le minimum de cycles mémoires pour réaliser une instruction est de 2 microsecondes (μs) pour par exemple charger l'accumulateur A (LOAD : LDAA); alors que le maximum est de 12 microsecondes (μs) pour par exemple une interruption (software interrupt : SWI).

I. B : LE LOGICIEL

I - PROGRAMMATION

L'évolution du logiciel est étroitement liée aux travaux des mathématiciens tels que Church et Turing, cela dit, il est indiscutable que la paternité du logiciel revient à des gens qui étaient aussi bien des théoriciens que des praticiens; Par exemple on cite Van Neumann.
Un des problèmes soulevé par les premiers calculateurs électronique : s'est naturellement posé en terme de relations homme-machine. Dès 1947 malgré le matériel disponible qui permettait déjà d'effectuer des multiplications de 2 nombres de 16 bits en 16 µs, on avait pas encore trouvé de méthode bien commode pour communiquer à la machine les informations dont-elle avait besoin; cette situation mettait en évidence le fossé qui séparait les compétences du technicien et les capacités de la machine. Ce problème de communication réduisait dans de larges proportions les avantages du calcul électronique.

Les idées de Von Neumann et autres s'imposèrent. Elles se reposaient sur la constatation simple qu'un programme pouvait être considéré comme une collection de données qu'on pouvait charger en mémoire et traiter.

Cette notion de programme enregistré permit d'accroître substantiellement le caractère automatique du calculateur.

1°) L'organigramme.

L'étape de la rédaction d'un programme est précédée par une analyse du problème à traiter, ce qui se traduit par un organigramme qui représente les étapes logiques conduisant à la solution du problème.
Le programmeur va décomposer cette opération en une série de petites instructions successives par lesquelles la machine electronique devra passer, c'est là que demeure la difficulté majeure.

.../...

Chacune de ces instructions est composée de 3 parties :

- Code opération : qui représente l'opération de base que peut exécuter le microprocesseur; par exemple LDA A (charger l'accumulateur A ≡ Load A accumulator). STA A (ranger le contenu de l'accumulateur A ≡ stock Accumulateur A).

- Adresse de l'opérande, qui désigne l'adresse de la mémoire de l'opérande ou bien la valeur de l'opérande.

- Champ de commentaire à vocation purement informative, le programmeur expliquera dans cette dernière colonne, ce qu'il a voulu faire, non seulement afin que son programme puisse être lu et compris par une troisième personne mais encore afin que lui même se retrouve dans difficulté.

2°) Traduction du programme :

Malheureusement, le microprocesseur ne comprend pas le language mémonique, par conséquent, il va falloir lui traduire tout cela en binaire qui est son propre language. Chaque instruction de deux ou trois octets sera représentée par une série de zéro et de uns ; dans le cas où le programme est constitué d'une centaine d'instructions ou plus, on comprendra facilement que l'opération est peu agréable car l'erreur est inévitable. C'est pour cette raison que le programmeur préfère coder son programme mémonique par l'hexadécimal (4 bits correspondent à un digit hexadecimal :
(0;1;2;3;4;5;6;7;8;9;AB;C;D;E;F)
Exemple $(1001\ 1110)_2 = (9E)_{16}$

Les instructions en hexadécimal sont plus faciles à utiliser par le programmeur.
L'éxécution de la conversion mnémonique à hexadecimal se fait déjà au niveau du microprocesseur en utilisant un code enregistré fourni par le constructeur c'est un genre de dictionnaire mnémonique ---) numérique relatif au 72 instructions du MC 6800. Ce dictionnaire appelé Assembleur,l'opération s'appelle Assembler le programme dont en parlera plus loin.

3º Introduction du programme en memmoire

Le programme a été rédigé en mnémonique puis traduit en numérique va maintenant être logé dans les mémoires internes RAM ou ROM du microprocesseur; pour cela, il convient d'affecter une adresse mémoire à chaque instruction. Chaque cellule mémoire est capable de stocker un octet; c'est à dire deux digits hexadécimal.

Une séquence d'un programme est contrôlée par le contenu du compteur de programme P.C. (programme counter). Sous l'éxécution normale du programme, le PC relient la prochaine adresse de l'instruction suivante l'on doit aller chercher dans la mémoire centrale contenant le programme à éxécuter et il est incrémenté automatiquement par le contrôle interne du MPU chaque fois que chaque octet d'une instruction a été pris en compte.

A chaque fois que le contenu d'une adresse mémoire est lu, il sera transféré dans le registre d'instruction, puis décodé. On constate que chaque instruction déclenche, pour son éxécution une serie micro opérations qui ne dépendent plus du programmeur: On remarquera que le programmeur peut d'une manière ou d'une autre améliorer son programme en utilisant les avantages particulers qu'offrent les registres complémentaires tels que :

- 2 Accumulateurs de 8 bits chacun.

- Un registre d'index permettant une méthode d'adressage mémoire supplémentaire par le calcul de l'adresse (méthode très utilisée).

- Une pile de stockage en mémoire. Il s'agit d'une position de mémoire adressée au moyen d'un registre du pointeur de pile (stock pointer) qui est utilisé pour stocker temporairement le contenu des registres lorsqu'une interruption (Branchement, saut à une sous-broutine) est demandée. Ceci sera décrit plus loin d'une manière plus détaillée.

.../...

II - ASSEMBLEUR ET ASSEMBLAGE

Le programme est rédigé de façon la plus courante en utilisant un langage ignoré par le microprocesseur qui est le langage mnémonique. Comme la machine electronique ne comprend que le numérique il faudrait obligatoirement passer par l'étape de traduction.

1°) L'assembleur.

C'est un dictionnaire utilisé pour traduire le code mnémonique en code machine (numérique). Un tel dictionnaire est un programme qui provient du créateur du langage. Ce programme est enregistré sur un support approprié (Bande perforée ou magnétique ou disque souple et rigide) en langage machine.

Ainsi, la traduction du programme mnémonique peut être assurée par l'ordinateur lui-même, cette opération est plus efficace car :

1°) Elle permet aussi de confier au programme de traduction le soin de depister les erreurs de syntaxe ou du codage.

2°) Elle permet l'exclusion du risque d'introduction d'erreurs manuelles.

3°) Cet avantage est capital, elle permet de représenter les adresses sous forme symbolique appelées étiquettes. Le programmeur n'a pas à se soucier de l'adresse réelle des instructions, au moment où il écrit un programme.

On définit donc :

- Le programme source qui est le programme rédigé par le programmeur.

- Le programme objet est ce même programme mais traduit en lagage machine (numérique). Il est translatable car ces adresses

.../...

d'implantation en mémoire ne sont pas définitives et le programme est ensuite repris par un chargeur éditeur de lien. L'éditeur de liens relie entre eux les différents modules objets et donne naissance à un seul module qui sera rangé par le chargeur à son emplacement définitif avant éxécution.

2°) Langage d'assemblage.

C'est un langage symbolique, c'est à dire constitué d'un texte en mnémonique qui diffère d'un fabricant à un autre.

Un programme écrit en langage d'assemblage est une suite de lignes sources. Des commentaires peuvent êtres ajoutés pour faciliter la relecture. Ils ne seront pas pris en considération dans l'éxécution du programme résultant. L'ensemble constitue un programme symbolique. Afin que l'assembleur puisse effectuer correctement la traduction, certaines conventions sont à respecter au moment de l'écriture en ce qui concerne l'orthographe et la syntaxe. Après un numéro de ligne on laisse un blanc pour l'étiquette avant d'écrire le mnémonique et un autre blanc sera laissé après ce dernier.
Une ligne source (soit une instruction) se compose de 5 Zones :

N °ligne	Etiquette	Mnémonique	Opérande	Commentaire

a) Zone étiquette

C'est un ensemble de un à six caractères alphanumériques, le 1er étant alphabétique. Une étiquette doit être unique, elle repère une destination de branchement.

b) Zone mnémonique.

Elle contient la représentation des codes opérations grâce à des côdes (mnémonique). On y trouve les mnémoniques des opérations à exécuter. Certaines de ces directives seront traduites, d'autres seront de simples renseignements (nom, origine et fin NAM, ORG et END).

.../...

La liste des 72 jeux d'instructions en mnémonique dont dispose le MC 6800 (Tableau 1)?

c) La zone opérande.

Le champ opérande est réservé à la partie variable de l'instruction. On y trouve le mode d'adressage, une adresse, une valeur numérique, un symbole ou une expression (sous - broutine).

d) La zone commentaire.

Celle-ci est facultative et non traduite, son rôle est de faciliter la lecture des programmes sur listing.

3°) Assemblage.

1°) Format d'assemblage.

Une fois écrit un programme source, il faut l'introduire dans le micro-ordinateur pour le traduire. Quelque soit le périphérique employé à cet usage (teletype, lecteur de ruban , ou imprimante ...) il faut dans tous les cas respecter les conventions de formats, fixées par le constructeur qui permettront à l'assembleur d'identifier la nature des informations. Les principales conventions en usage pour l'écriture d'un programme source sur une teletype sont les suivantes :

- On sépare une zone de la suivante en frappant un "blanc"

-Instruction sans étiquette, le 1er caractère frappé est un "blanc" après le numéro de la ligne.

- Instruction avec étiquette est écrite directement après le numéro de la ligne.

- Séparation des instructions, chaque fois qu'une instruction est complète on frappe le caractère " retour chariot" et un numéro de ligne apparaît.

.../...

2°) Principe de l'Assemblage.

Le rôle essentiel de l'assemblage va être double :

a) Traduction des mnémoniques (LDAA, MUL ...) dans le code binaire du microprocesseur à l'aide de l'assembleur. Toutes les instructions seront traduites en binaire dans le code machine LDAA soit 0101; ADD soit 0110. Pour cela l'assembleur dispose d'une table d'équivalence suivante :

Code sourse (Chaine de caractères ASCII)	Code Objet (Code machine)	
Code ASCII de LDA A	0101	Zonne de définition des mnémoniques (ZONE FIXE)
Code ASCII de ADD A	0110	
Code ASCII de NEWLIN	Valeur correspondante du compteur adresse.	Zone de définition des étiquettes (ZONE VARIABLE)

Table de symboles et Etiquettes.

b) Affectation des valeurs numériques aux étiquettes et aux opérandes symboliques. C'est au moment de l'assemblage que sont déterminées les adresses réelles des instructions. Pour cela, l'un des registres du microprocesseur est utilisé comme compteur, ce dernier est initialisé au début de l'assemblage, puis s'incrémente à chaque nouvelle instruction. Pour chaque instruction, l'assembleur range le code machine à l'adresse indiqué par le compteur d'adresses.

Lorsqu'une instruction comporte une étiquette : soit par exemple

NEWLIN LDA A ## $ 13

.../...

et puis une autre instruction qui fait appel.

JMP NEWLIN

Au moment de l'assemblage, la valeur de l'adresse réelle correspondante à l'étiquette NEWLIN est établie. Pour cela l'assembleur dispose dans la table des étiquettes et symboles d'une zone libre permettant la définition des étiquettes.

Lorsque l'assembleur traduit l'instruction LDA A, il détecte la présence de l'étiquette, il range alors dans la table le code ASCII de l'étiquette NEWLIN aussi que la valeur d'adresse à cet instant, grâce à cette mémorisation, il saura en arrivant à l'instruction JMP NEWLIN, retrouver dans la table la valeur réelle à affecter à l'opérande NEWLIN.

En résumé, comme l'assembleur ne peut pas généralement traiter simultanément les étiquettes les mnémoniques et les opérandes; l'assemblage se fera en plusieurs phases ou (passes) c'est à dire que le programme source est exploré plusieurs fois. A chaque fois un traitement différent est effectué. Généralement, il y a 3 phases.

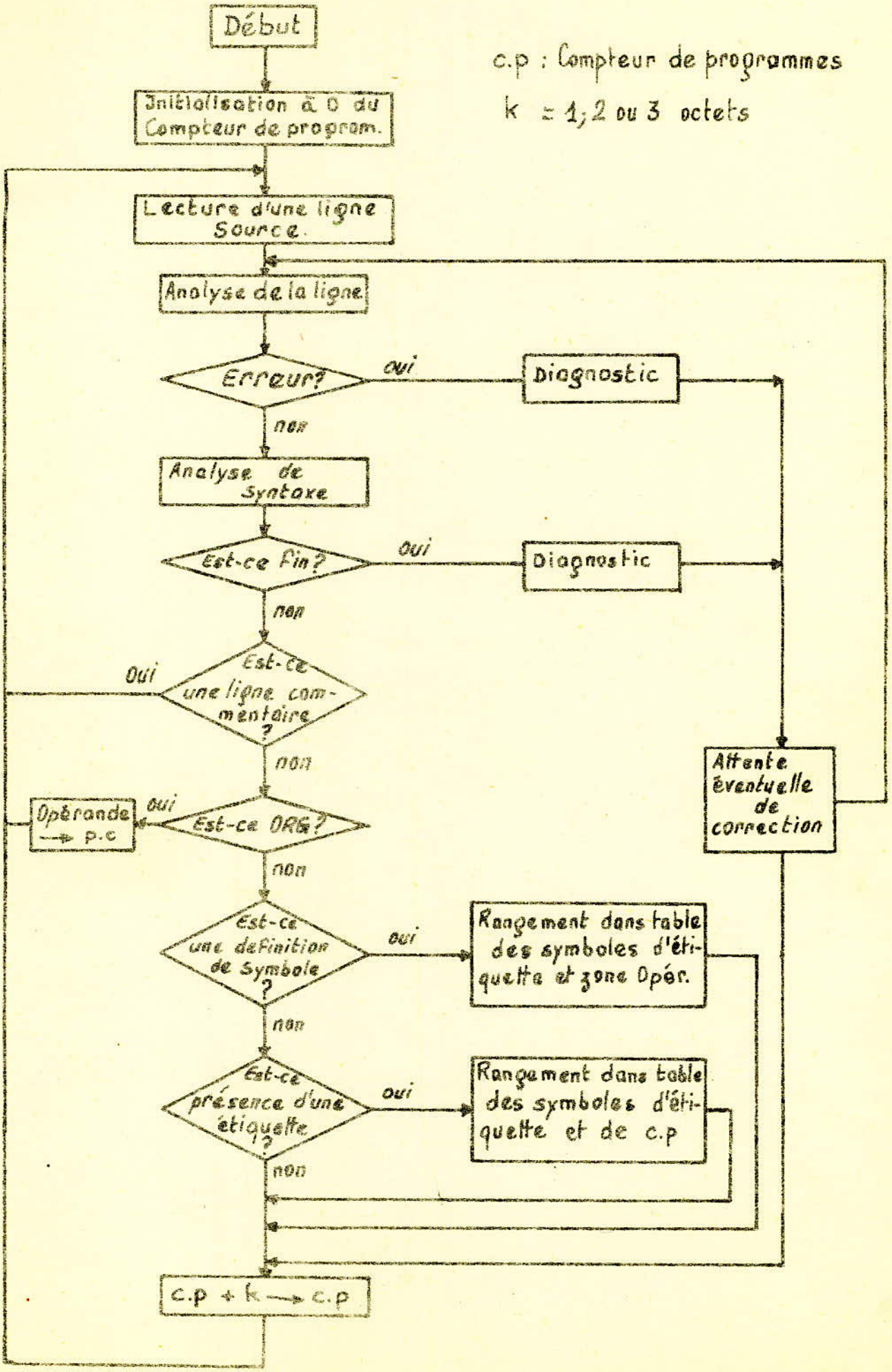
1er) Phase (passe) L'assembleur traduit les mnémoniques en code machine et range les étiquettes dans la table des étiquettes. L'assemblage détecte ainsi l'absence d'étiquettes si elle est nécessaire, sa présence si elle est interdite, un code opération erroné. qui suit illustre cette première phase. ↘ l'organigramme

2°) Phase (passe) L'assembleur affecte aux opérandes symboliques leurs valeurs réelles. Ceci n'est possible qu'à une 2ème passe, car il faut que les étiquettes correspondantes aient été préalablement définie à la 1ere passe.

3ème phase (passe). Elle commande la frappe d'un listage du programme objet accompagné du programme source et des commentaires. C'est le télé-imprimeur qui servira à cet effet.

...4..

Assemblage d'un programme se fait en 3 phases
cet organigramme est celui de la 1ère phase.



c.p : Compteur de programmes

k = 1, 2 ou 3 octets

Exemple d'assemblage d'un programme
(Addition de 2 nombres)

Programme Source à Assembler

PSEUDO-INSTRUCTION		ORG	10
	A1	EQU	20
	A2	EQU	30
	A3	RST	40
INSTRUCTION EXECUTABLE		LDA A	A1
		ADD A	A2
		STA A	A3
PSEUDO-INSTRUCTION		END	

Table des Symboles et Etiquettes

[LDA A] en ASCII	0101	zone Symboles
[ADD A] en ASCII	0110	
[STA A] en ASCII	1011	
[A1] en ASCII	100	zone étiquettes
[A2] en ASCII	101	
[A3] en ASCII	102	

1ere Passe d'assemblage

- traduction des mnemoniques
- définition des étiquettes

Compteur Adresse	Mémoires de Programme	
10	20	1ere Opérande
11	30	2eme Opérande
12	40	Resultat
13	0101	[A1] ASCII
14	0110	[A2] ASCII
15	1011	[A3] ASCII

2eme Passe d'assemblage

- identification des opérandes symboliques
- on obtient le programme objet

10	20	
11	30	
12	40	
13	0101	10
14	0110	11
15	1011	12

3eme Passe d'assemblage

0010	0010			ORG	$10
0020	0010		A1	EQU	20
0030	0011		A2	EQU	30
0040	0012		A3	RST	40
0050	0013	96 10		LDA A	A1
0060	0014	9B 11		ADD A	A2
0070	0015	97 12		STA A	A3
0080	0016			END	

III - MODES D'ADRESSAGE

Lorsqu'une instruction fait référence à un opérande, elle peut repérer celui-ci en mémoire de différentes façons appelées : modes d'adressage.

Les informations (instructions et données) sont stockées dans des cellules des mémoires RAM ou ROM. Chaque cellule est caractérisée par son adresse. Pour atteindre une cellule précise, on dispose de plusieurs manières " modes d'adressage " que l'on examinera un à un.

Ainsi par un choix judicieux, il est possible d'améliorer le programme en réduisant :

- la longueur du programme
- la capacité d'éxécution
- le temps d'éxécution.

Ces qualités recherchées en micro - informatique conférent une très grande souplesse à la programmation.

Dans ce chapitre, on expose les différentes caractéristiques et le fonctionnement de ces modes d'adressages ontrouve :

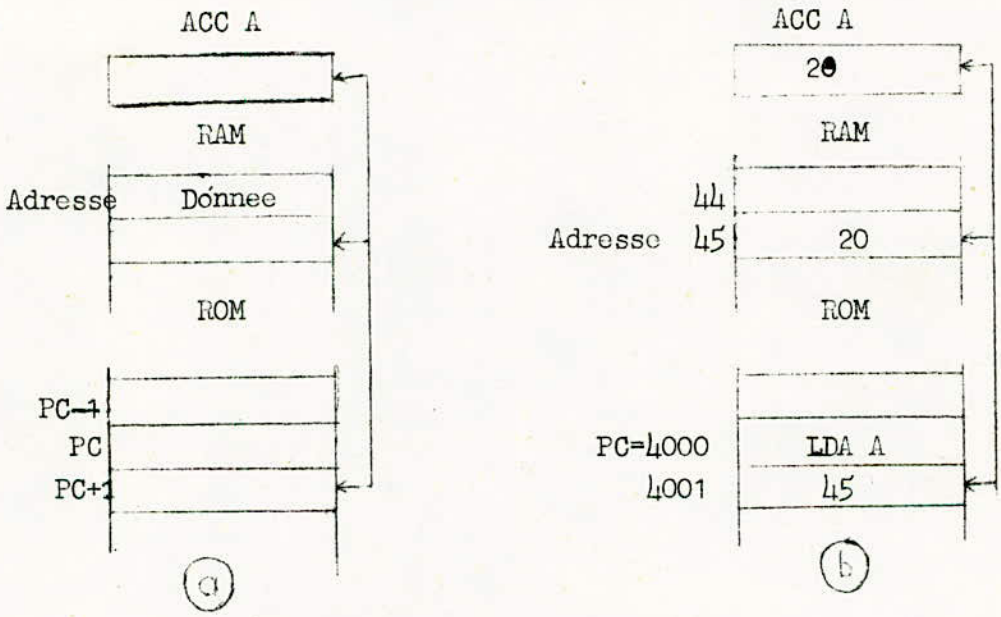
1°) Adressage direct
2°) " étendu
3°) " indirect
4°) " immédiat
5°) " indexé
6°) " implicite
7°) " inhérent.

Par définition on appelera adresse effective, l'adresse réelle finale de la cellule mémoire dans laquelle se trouve la donnée recherchée.

1) Adressage direct

L'instruction contient au moins 2 Octets; l'adresse de l'opérande est contenue dans le 2ème Octet de l'instruction; le 1er Octet est réservé à l'opération. C'est le mode d'adressage le plus utilisé et le plus simple; il consiste à utiliser les adresses fournies sur le bus adresse pour acceder directement à des données contenues dans les positions correspondantes de la mémoire.

Exemple : LDA A $ 45 charger le contenu de l'adresse 45 (en hexadécimal) dans l'accumulateur A. Dans l'adresse 45 de la RAM on a l'opérande $(20)_{10}$. Le compteur ordinal (P.C) nous fournit l'adresse de l'instruction, codée par exemple sur 2 cellules mémoires (2 octets). On lira donc les cellules représenté par (PC) puis (PC+1). Ce mot d'instruction de 16 bits passera dans le registre d'instruction, il sera décodé, sa partie adresse contient l'adresse effective de la donnée; celle-ci est alors lue directement dans la cellule mémoire indiquée et le mot $(20)_{10}$ sera logé dans l'accumulateur A.



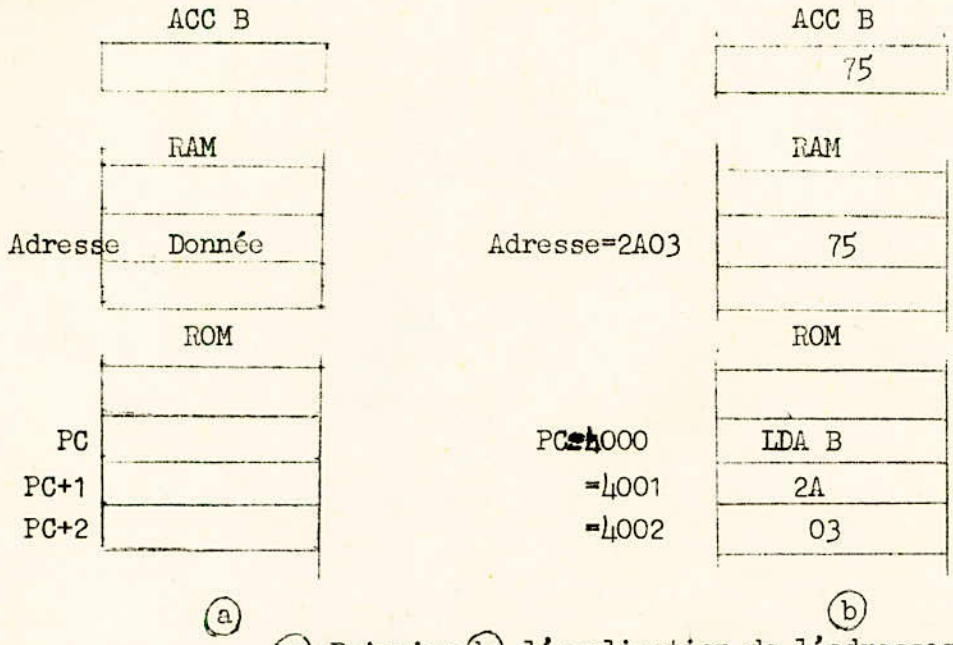
Dans le schéma (a) on a le principe
Dans le schéma (b) on a l'application de l'adressage direct.
L'ordre LDA A (charger l'accumulateur A) est lu à la cellule d'adresse 4000 pointé par le (P.C) puis incrémenté de 1 à la cellule 4001 où se trouve l'adresse effective de la donnée $(20)_{10}$ est lu aussi. Le décodage renvoie à la cellule 45 de la RAM où se trouve $(20)_{10}$ qui chargera l'ACCA.

2°) Adressage en étendu.

L'instruction contient 3 octets; l'adresse de l'opérande est contenue dans les 2ème (poids fort) et 3ème (poids faible) octet de l'instruction. Ce mode d'adressage permet le balayage de toutes les mémoires de 0000 à FFFF. Pour lire une adresse sur 16 bits on passera 2 tours de lecture en mémoire, c'est évidemment plus long que le

.../...

mode d'adressage direct. On appelera donc le mode d'adressage étendu, un adressage direct mais ou le mot adressage est codé sur une longueur double, soit 2 Octets.



(a) (b)

(a) Principe (b) l'application de l'adressage en étendu

Le compteur ordinal adresse une instruction sur 3 Octets

- Un octet : champ opération
- 2 octets : champ adresse

Cette instruction sera décodée puis éxécutée.

Soit à charger l'acc B avec la valeur effective 75 (HEX.) contenue dans la cellule mémoire d'adresse 2A03 (HEX.).

LDA B $ 2A03

Le compteur ordinal pointe PC = 4000 où l'on lit (LDA B), puis (PC + 1) = 4001 où l'on lit 2A et enfin (PC + 2) où l'on lit 03,

A cette adresse 2A03 on lit dans la RAM 75 qui sera logée dans l'accumulateur B.

3º) Adressage Indirect.

L'instruction se présente au moins sur 2 champs : le code opération et le champ adresse qui est logé dans l'opérande. La cellule désignée par le mot d'instruction ne contient pas la donnée utile, mais simplement une autre adresse qui sera l'adresse effective qui contient la donnée. Grâce à l'adressage indirect, il est facile de stocker un programme complet dans une mémoire morte ROM. Les adresses utiles de sousbroutines qui composent ce programme seront logées dans une RAM. Ainsi à partir de mêmes instructions de programme on pourra traiter des données implantées dans toute la mémoire. Cet adressage indirect présente un inconvénient qui est celui du temps de recherche de l'information.

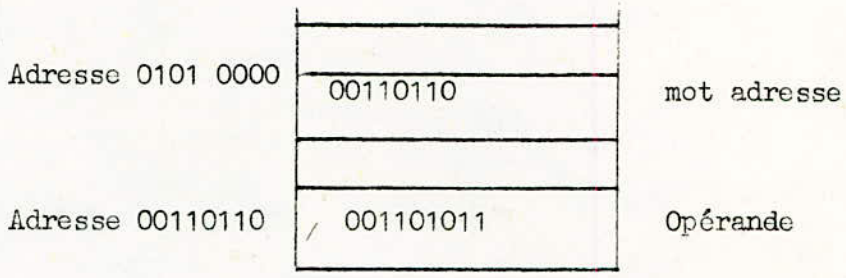
.../...

Pour obtenur l'adresse effective de l'opérande, il faut deux cycles de lecture de la mémoire; le temps de traitement est ainsi augmenté d'une durée de cycle; nommée phase d'indirection.

Exemple STA A 80_{10} soit 0101 0000(Binaire)--) 50(HEX.)

OPERATION ADRESSE

STAA	0101 0000

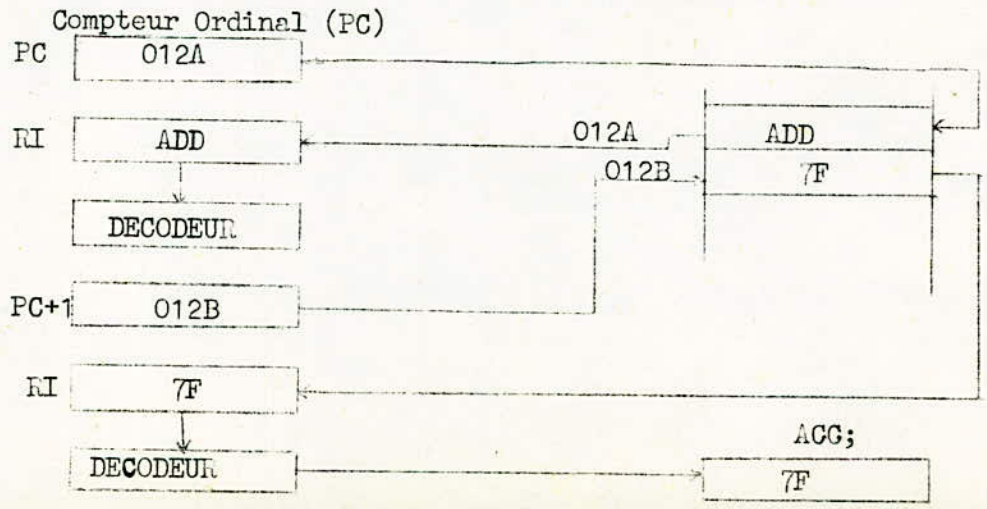


La 1ère adresse lue est $(80)_{10}$ soit 01010000 en binaire. Elle renvoie à la cellule $(0\ 0110110)_2$ soit $(54)_{10}$. Dans cette dernière cellule mémoire, on lit enfin l'opérande = 01101011 soit $(107)_{10}$.

4º) Adressage immédiat.

L'opérande est contenu dans le 2ème ou 3ème et 3ème octet de l'instruction, selon que l'on s'adresse aux accumulateurs ou au registre. Le champ adresse de l'opérande contient la valeur de l'opérande il est appelé opérande immédiat; donc ce mode d'adressage ne permet de traiter que les opérandes dont les valeurs sont des constantes dans le programme.
Exemple : ADD ## $ 7F cette valeur est additionnée à Accumulateur.

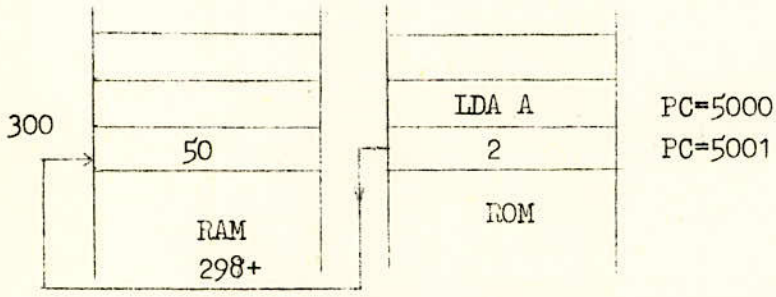
Le compteur de programme reçoit l'adresse 012A, il vise la 1ère partie de l'instruction puis la 2ème partie en 012B.
L'ordre à éxécuter est donné. ADD est transféré dans le registre d'instruction puis décodé. Entre temps le compteur de programme est décrémenté et le transfert de la 2ème partie de l'instruction s'effectue, passe par le décodeur puis transmet à l'accumulateur qui est censé contenir 00 :



5º) Adressage Indexé.

L'instruction est composée de l'opération sur le 1º Octet et de l'adresse sur le 2ème Octet. On fait intervenir le registre d'index de 16 bits; l'adresse effective de l'instruction est obtenue en ajoutant le contenu du registre d'index à la valeur du 2ème Octet de l'instruction. Ce mode d'adressage est particulièrement adapté au traitement de tableau de données par incrémentation ou décrémentation des instructions particulières.

Exemple : LDA A 2,X charger l'Acc. A du contenu de l'adresse mémoire 2 plus le contenu du registre d'index.



L'instruction indexé de l'exemple suivant consiste à charger l'acc.A avec le contenu de la mémoire dont l'adresse est 300 qui est obtenu en faisant la somme du contenu du registre d'index (298) et le contenu de la position mémoire 5001 qui est (2). Le contenu de cette mémoire ainsi formé est (50) qui chargera l'accumulateur A.

6º) Adressage en Implicite.

L'instruction est composée ainsi de 2 ou 3 octets;l'opérande est indiqué par le code opération de l'instruction. Dans l'adressage implicite, on s'adresse implicitement à un registre donné.

Exemple PSH A

Sauvegarder le contenu de l'acc.A implicitement dans la pile.

SUB $ C4

Ordonne de soustraire implicitement $ C4 du contenu de l'ACC.B.

7º) Adressage Inhérent.

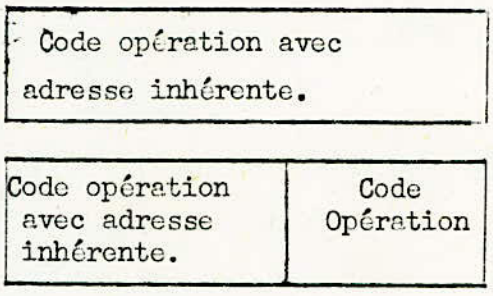
C'est parfois l'instruction elle-même plus exactement son code opération, qui contient l'adresse où se trouve la donnée sur laquelle va porter l'opération.

Ex. ADD A $ 3C2F

Addtionner au contenu de l'ACC. ACCA la valeur 3C2F (HEX.)

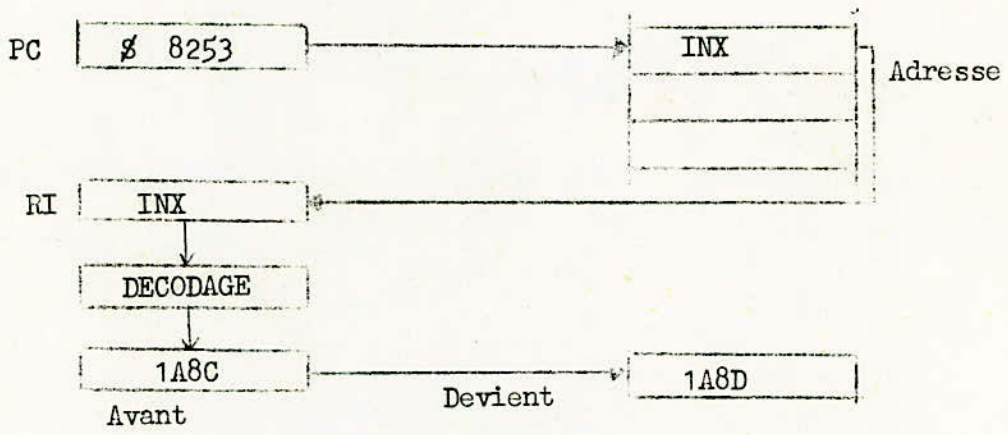
.../...

Comme on vient de le voir, cet adressage inhérent n'exclut pas la présence d'un champ opérande, qui contient le second opérande s'il y a lieu (ou son adresse). Ainsi, on trouve deux types de formats de mots d'instruction.



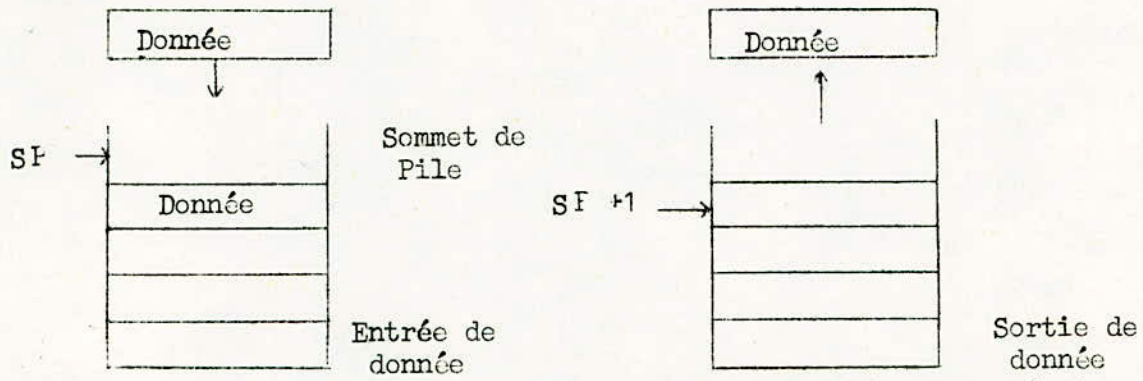
Exemple INX incrémenter le registre d'index

Le compteur ordinal contient l'adresse $ 8253 et pointe à celle-ci.



IV - STRUCTURE DE PILE

Une pile est un mécanisme câblé ou programmé, qui permet de mémoriser des informations et de les restituer selon le mode " dernier entré, premier sortie". Son organisation est schématisée par la figure suivante :

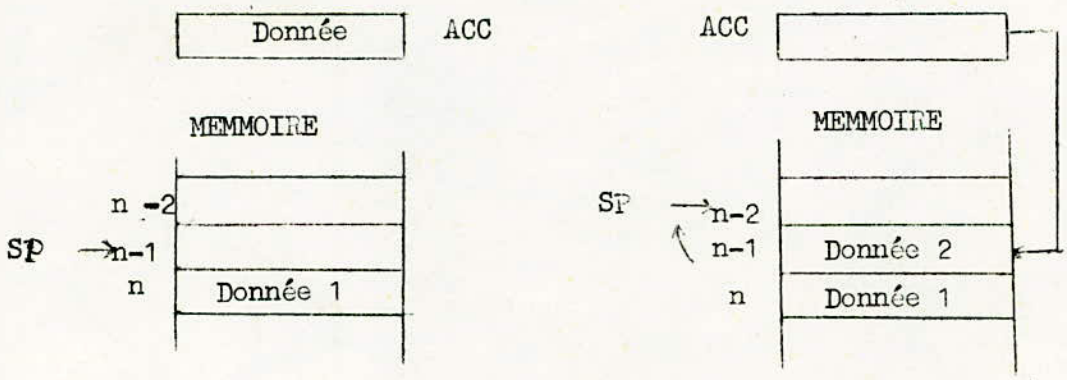


La pile peut être constituée par une mémoire spécialisée, rapide ou être incluse dans la mémoire centrale de la machine électronique; dans ce cas sa capacité n'est pratiquement pas limitée (sauf par la capacité de la mémoire centrale elle-même).

1°) Gestion de la pile.

Un registre, nommé pointeur de pile (stock pointer) SP contient l'adresse de la première position libre au sommet de la pile; il est modifié à chaque entrée ou sortie d'information, de sorte qu'il désigne toujours ce sommet de pile.

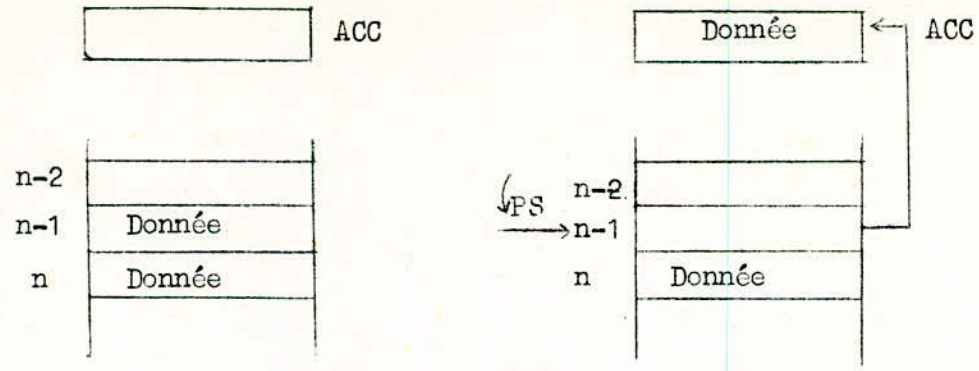
Si l'on considère une pile descendante en mémoire les adresses sont décroissantes vers le sommet de pile, le registre est donc décrémenté à chaque entrée d'information, et incrémenté à chaque sortie. La figure ci-dessous illustre le principe.



Entrée de donnée en provenance de l'Acc. Acc. -----) Pile

(PS ← n - 1) -------)(PS = n-2)

.../...



Sortie d'une donnée vers l'Acc.(PS=n-2)---) (PS=n-1)
(Ps) ---) ACC.

Une pile consiste donc en une zone mémoire,essentiellement utilisée pour les stockages temporaires de données, et qui est gérée par un mécanisme d'adressage particulier utilisant un registre pointeur de pile. Des instructions de stockages en pile (PUSH) ou de sortie de pile (PULL) permettent son utilisation; mais cette structure est aussi utilisée pour les appels de retours de sous-programmes.

2º) Sous-programme.

Lors d'un appel de sous-programme, le problème essentiel résidé dans la mémorisation de l'adresse de retour. Il y a deux modes principaux de sauvegarde de cette adresse.

- Utilisation d'une position mémoire incluse dans le sous--programme.
- Utilisation d'une pile.

a) Utilisation d'une position mémoire de sauvegarde.

L'éxécution de l'instruction d'appel provoque le transfert de l'adresse de retour dans la première position mémoire du sous-programme. Cette dernière est simplement le contenu du compteur du programme qui a été incrémenté enfin de décodage de l'instruction d'appel.
Dans la seconde phase, a lieu le transfert de l'adresse de début de sous-programme + 1 dans le cmpteur de programme (P.S); en effet les instructions éxécutables du sous-programme ne commencent qu'a sa deuxième position;
La première étant la mémoire de sauvegarde.

Cette structure ne permet pas la réentrance direct des sous-programmes,tout nouvel appel détruisant le contenu précédent de la mémoire de sauvegarde. De plus, ce mode de sauvegarde est mal adapté aux systèmes à microprocesseur dont les programmes et sous programmes sont fréquemment figés en mémoire morte.

.../...

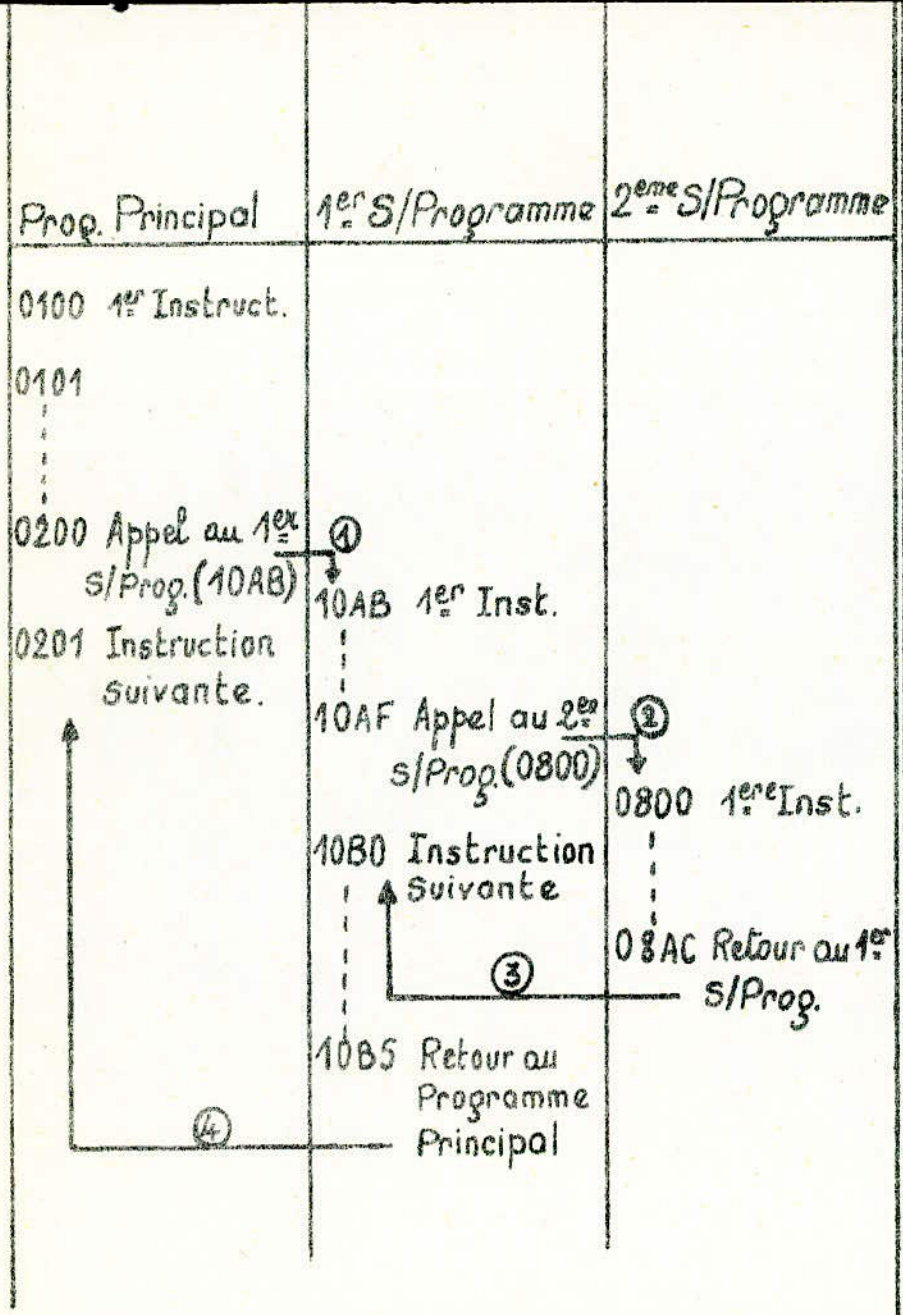
Le retour au sous-programme appelant s'effectue au moyen d'une rupture de séquennce par adressage indirett de la mémoire de sauvegarde (EX. BRA : branchement au point de départ du programme).

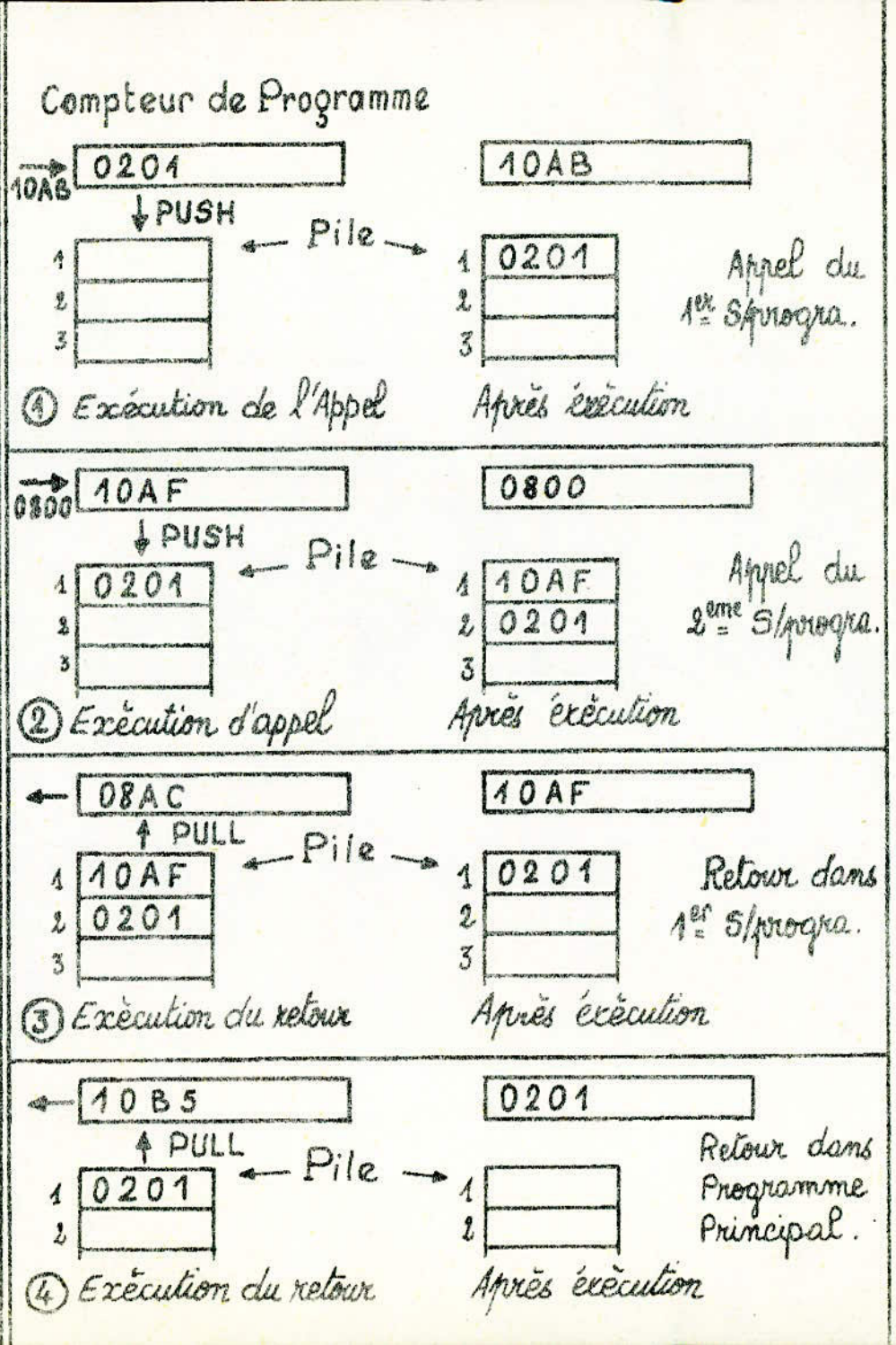
b) Sauvegarde en pile

Une pile peut être utilisée pour sauvegarder l'adresse de retour d'un sous-programme. Lorsqu'un sous-programme est appelé, l'adresse de retour dans le programme appelant est stockée dans la pile, et restituée dans le compteur de programme à la fin du sous-programme.

La figure de la page suivante nous montre le principe. Notons que cette pile peut en outre stocker les contenus des regis tres utilisés dans le programme appelant, ils ne sont ainsi pas perturbés par l'appel du sous-programme et peuvent être utilisés sans restriction dans celui-ci

EXEMPLE D'UTILISATION D'UNE PILE D'ADRESSE DE SOUS-PROGRAMMES (3 programmes emboîtés)





CHAPITRE II

§§§§§§§§§§§§§

BIBLIOTHEQUE MATHEMATIQUES DE MOTOROLA

§§§

II - A . PRESENTATION DE LA BIBLIOTHEQUE MATHEMATIQUES.

I - I N T R O D U C T I O N

Ce programme de Motorola éxécutable sur micro Ordinateur utilisant le microprocesseur MC 6800, est conçu de manière à éxécuter les différentes opérations en virgule flottante. Un chiffre quelconque positif, négatif, décimal ou entier est écrit en virgule flottante de 24 bits, soit 16 bits pour la mantisse et 8 bits pour l'exposant.
Le 15ème bit de la mantisse représente le bit de signe; la virgule décimale vient juste après ce bit de signe. Le 7ème bit de l'exposant représente le bit de signe ainsi l'exposant et la mantisse sont tous les deux signés.

Ce programme est un support nécessaire pour permettre le contrôle des différentes fonctions directement sur l'écran de visualisation ou sur une télétype.

II - LES DIFFERENTES FONCTIONS

1°) Fonctions d'entrées :

Elles permettent d'introduire en mémoire trois bytes, soit 6 caractères (y compris le signe moins et la virgule) et de les convertir en un chiffre de 24 bits en virgule flottante. Ces fonctions sont les suivantes:

- READ 3B : (lire 3 bytes)
- STFP : (String to flotting point) conversion de la file de caractère en un chiffre en virgule flottante.

2°) Fonctions de sorties :

Celles-ci permettent la conversion d'un chiffre de 3 bytes écrit en virgule flottante en une file de caractères hexadécimaux (5 plus significatifs caractères) et les imprime sur l'écran de visualisation.

- FPST : (Flotting Point To String) conversion d'un chiffre en virgule flottante en une file de caractères

- WRIT 3B : (Ecriture de 3 bytes)

3°) Opérations réentrantes :

Ces opérations peuvent êtres appelées à l'aide de programm appelant ou de leur code opération :

Code opération	Opération.
1	FPPADD addition en virgule flottante
2	FPPSUB Soustraction en virgule flottante
3	FPP MUL Multiplication " "
4	FPPDIV Division " " "
5	FPP FLT Conversion d'un entier positif(16 bits) en virgule flottante (24 bits)
6	Conversion d'un nombre positif en virgule flottante en un entier (16 bits) FPPFIX
7	FPP NEG Négation d'un nombre
8	FPPCLR Effacer 3 mémoires consécutives.

Ces fonctions seront détaillées plus loins,avec leurs algorithmes.

4°) Opérations auxiliaires.

Ces opérations ne sont pas reentrantes, elles participent au contrôle des autres opérations, mais ne sont pas visualisées sur l'écran.

FPP CMP Comparaison de 2 nombres en virgule flottante

FPP CPY Transfert d'un nombre en virgule flottante.

III - CARACTERISTIQUES DE LA BIBLIOTHEQUE MATHEMATIQUES

La virgule flottante présentée dans ce chapitre possède les caractéristiques suivantes :

.../...

- La mantisse est de 16 bits soit 4 caractères significatifs, et l'exposant est de 8 bits soit 2 caractères hexadécimaux.

- Le format est compatible avec l'architecture du MC 6800.

- Exécution rapide

- Les chiffres négatifs sont complémentés à 2 :

16 bits de la mantisse complément à 2.

8 bits de l'exposant complément à 2 .

- La virgule du chiffre binaire en flottant apparaît juste après le bit de signe c'est à dire le 15ème bit.

- Disposition du chiffre en mémoire.

0 -----) le plus significatif byte de la mantisse(MSbyte)

1 ----) le moins significatif byte de la mantisse(LS Byte)

2 -----) exposant.

- Les nombres normalisés ont les 2 premiers bits de la mantisse différents pour cela la mantisse est décalée soit à droite ou à gauche.

- Dans les chiffres normalisés, le plus significatif byte (MSB) de la mantisse détermine nécessairement si le nombre est positif, négatif ou nul.

- Si on additionne un à l'exposant, le nombre est multiplié par 2 et si on soustrait un à l'exposant, le nombre est divisé par 2 .

Exemple 1 : $1 = \frac{1}{2} \times 2^1$ -----) 40 00 01

0.100	0 000	0 000	0 000	0 000	0 001
4	0	0	0	0	1

Exemple 2 : $3 = (\frac{1}{2} + \frac{1}{4}) \times 2^2 = \frac{3}{4} \times 4 = 3$

0.110	0 000	0 000	0 000	0 000	0 010
6	0	0	0	0	2

Exemple 3 : 18 (dec.) ---) 12(Hex.) ce chiffre est directement écrit après le bit de signe.

0.100	1 000	0 000	0 000	0 000	0 101	480 005
4	8	0	0	0	5	

$= \left(\frac{1}{2} + \frac{1}{16}\right) \times 2^5 = \frac{(8+1)}{16} \times 32 = 18$ (dec)

Exemple 4 : FF (Hex.) ----) 1 111 1 111 (Binaire)

0.111	1 111	1 000	0 000	0 000	1 000	7F80 08
7	F	8	0	0	8	Mant exp.

$\left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{64} + \frac{1}{128}\right) \times 2^8 = \frac{(64 + 32 + 16 + 8 + 4 + 2 + 1)}{128}$

$\frac{127 \times 256}{128} = 255$ (dec.)

Exemple 5 : - 1 (dec.)=-(1 Hex.) s'écrit aussi FFFF

1.100	0 000	0 000	0 000	0 000	0 001	C0 00 01
Signe débit C	0	0	0	0	1	

ici le complément à 2 du chiffre formé reste le même

Exemple 6 : - 25 (dec.) ----) -(19 Hex.) ----) on écrit 19 en binaire 0 001 1 001

1.110 0 100 0 000 0 000 0 000 0 101

Pour lire il faut d'abord complémenter à 2 car c'est un nombre négatif.

1.001	1 100	0 000	0 000	0 000	0 101	9C 00 05
9	C	0	0	0	5	

$-\left(\frac{1}{2} + \frac{1}{4} + \frac{1}{32}\right) \times 2^5 = -\frac{25 \times 32}{32} = -25$ (dec)

.../...

Dans le cas où les nombres sont inférieurs à 1 l'exposant devient signé. On peut représenter un nombre allant jusqu'à $2^{-127} \simeq 5,410^{-39}$ car l'exposant le plus petit est : 1 000 0001 le 7è bit est le bit de signe après complémentation à 2 : 1111 1111 ----) 1111 1111 (binaire) =-127 (dec.) d'où l'exposant = $2^{-127} \simeq 5,41\ 10^{-39}$

Exemple 7 : 0,8 (dec) ---) 6666 00 (Flottant)

0.110 0 110 0 110 0 110 0000 0000

$$= \left(\frac{1}{2}\right) + \frac{1}{4} + \frac{1}{32} + \frac{1}{64} + \frac{1}{512} + \frac{1}{1024} + \frac{1}{8192} + \frac{1}{16384}\Big)2^{0} = 0,799987792$$

$\simeq 0,8$ (dec) .

Exemple 8 : 0,1 -----; 6666FD (Flottant)

0.110 0110 0110 0110 1111 1101

L'exposant est négatif, on prend le complément à 2 pour voir à combien est - il égal : 1111 1101 Complet. à 2) 1 000 0011 = 2^{-3}

d'où notre nombre : 0, 799987792 X $\frac{1}{2^3}$ = 0,099998474

On remarquera que le complément à 2 de 8000 est inchangé

8000 : 1000 0000 0000 0000

8000 : 1000 0000 0000 0000 Complément à 2.

IV - MANIPULATION DANS LA PILE.

Les différentes réservations des arguments et des adresses de retour dans la pile sont disposées de la manière suivante :

17 LS Byte moins significatif byte de l'adresse de retour.
16 MS Byte plus " " " " "
15 LS Byte moins " " " " de l'argument 1.
14 MS Byte plus " " " " " " 1.
13 Exposant de l'argument 1
12 LS Byte moins significatif byte de la mantisse de l'argument 1.
11 MS Byte plus " " " " " " " 1.
10 LS byte moins " " " l'adresse de l'argument 2.
09 MS Byte plus " " " " " " "
08 Exposant de l'argument 2
07 LS Byte moins significatif byte de la mantisse de l'argument 2
06 MS Byte du résultat " " " " " " "
05 Exposant du résultat
04 LS Byte de la mantisse du résultat
03 MS Byte " " " " "
02 Différence des exposants pour l'addition et soustraction des exposants, ou signe du résultat pour la multiplication et la division.
01 Résultat temporaire pour la multiplication pour ensuite prendre seulement les 16 bits significatifs.
00 Compteur de Boucle.

V - ALGORITHME DES OPERATIONS

Tous les résultats occupent la place du premier argument.

1°) FPP ADD.

Le nombre 1 est additionné au nombre 2.

a) si un argument est nul, l'autre est redonné comme résultat.

b) si les exposants sont différents, la mantisse de l'argument ayant le plus petit exposant est déplacée vers la droite, jusqu'à avoir les exposants des 2 arguments égaux.

c) le résultat sera la somme bit à bit des mantisses et l'exposant du résultat sera égal au plus grand exposant des 2 arguments.

d) s'il y a plus de 15 déplacements à faire à l'un des arguments, le plus grand nombre est donné comme résultat.

e) Le dernier bit décalé servira à l'arrondissement du même nombre; cette opération consiste à ajouter le bit décalé au bit de plus faible poids de la mantisse.

f) Opération de normalisation qui portera sur le résultat si ce dernier a 2 mêmes bits consécutifs de poids le plus fort (1.1) ou (0.0).Ce résultat sera déplacé vers la gauche ou vers la droite jusqu'à la configuration (0.1) ou (1.0).

Les points e et f sont valables pour toutes les opérations.

2°) FPPSUB

Il suffit de rendre l'argument a substitué négatif donc prendre son complément à 2 et le même algorithme que pour l'addition est utilisé.

.../...

3°) FPPMUL

a) Les arguments sont tous les 2 convertis en 2 nombres positifs.

b) On calcul la somme des exposants qui sera l'exposant du résultat.

c) Les mantisses sont multipliées bit par bit, les 16 bits les plus significatifs sont gardés pour former le résultat.

d) Ces 16 bits sont arrondis à 15 bits.

e) Le 16ème bit de la mantisse portera le signe du nombre qui sera calculé en tenant compte des 2 signes des arguments 1 et 2.

4°) FPP DIV

Division de 2 arguments.

a) Les 2 arguments sont convertis en 2 nombres positifs par complémentation à 2 s'ils sont négatifs.

b) Si le diviseur est nul, le ~~dividande est redonné comme~~ résultat est infini.

c) Les 16 bits du résultat sont arrondis à 15.

d) Le résultat portera le signe dans le 16ème bit qui sera plus ou moins selon le diviseur et dividande.

5°) FPP CLR.

Le nombre en virgule flottante est désigné par son adresse dans l'accumulateur A, puis ces 3 positions mémoires sont remplacées par des zéros.

6°) FPPCPY

Déplacera un nombre d'une position mémoire à une autre.

.../...

a) Dans l'accumulateur B on charge l'adresse du nombre en virgule flottante à déplacer.

b) Dans l'accumulateur A on charge l'adresse de la position mémoire où le nombre doit être déplacé.

7°) FPPCMP.

Comparer 2 nombres en virgule flottante.

a) On charge les accumulateurs A et B par les adresses des 2 nombres à comparer.

b) Une soustraction est appliquée à ces 2 nombres à comparer.

c) Le MS Byte du résultat est stocké dans l'accumulateur A, celui-ci sera testé pour voir l'état du nombre trouvé.

d) Le registre d'état ou code condition nous renseignera si ce nombre est positif, négatif ou nul.

8°) FPPFLT.

Elle convertit un nombre de 4 caractères hexadécimaux (16 bits = 4 chiffres) en un nombre en virgule flottante de 24 bits (6 caractères) et en décimal.

a) Le chiffre est écrit à partir du bit 0 de la mantisse.

b) le byte de l'exposant est positionné à 15(1111)

c) L'opération de normalisation est appliquée au nombre.

d) Le nombre est lu après normalisation.

.../...

Exemple 1 : 0026 (Hex.) ----) 0010 0110(binaire) = 38 (dec)

0.000 0000 0010 0110 0000 1111

Pour normaliser le chiffre, on décale de 9 rangs vers la gauche, l'exposant sera ainsi decrementé de 9 il devient égal à 6.

$$\underbrace{0.100}_{4}\ \underbrace{1100}_{C}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0110}_{6}\quad 4C0006$$

Exemple 2 : 12,5 n'est pas converti car il n'est pas en Hexadécimal, le caractère virgule (,) n'est pas reconnu.

Pour les nombres négatifs, le caractère moins (-) n'est pas reconnu.

Pour convertir ces nombres on prend leur complément à 2

Exemple 3 : - 12 (dec), en hexadecimal 12 s'écrit C ---) 1100 (binaire).

Comme il est négatif on prend son compl. à 2 pour le considérer comme positif.

1.111 1111 1111 0100 = FFF4 ce chiffre sera reconnu.

← 1.111 1111 1111 0100 0000 1111 il n'est pas normalisé.

$$\underbrace{1.010}_{A}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0100}_{4}\quad A0\ 0004$$

Pour retrouver - 12 en décimal.

Il faut recomplémenter à 2 le résultat car il a déjà été complémenté à 2.

1.010 $\xrightarrow{\text{Complet. 2}}$ 1.11 $= (\frac{1}{2} + \frac{1}{4})$

$$-(\tfrac{1}{2} + \tfrac{1}{4}) \times 2^4 = -\tfrac{3}{4} \times 16 = -12 \text{ (dec)}$$

.../. .

9) FPPFIX

Un nombre en virgule flottante est converti en un nombre en hexadecimal composé de 4 caractères.

a) Le nombre en virgule flottante (normalisé) est dénormalisé (l'exposant est alors incrémenté jusqu'à 15).

b) Le chiffre entier donné sera celui de la mantisse qui représente un nombre hexadécimal.

c) Si l'exposant est supérieur à 15; la mantisse est décalée à gauche et l'exposant est décrémenté jusqu'à 15, ainsi le nombre lu sera différent de celui qui est donné.

Exemple 1 : 4C0006 ----) 26 (Hex.)

0.100	1100	0000	0000	0000	0110
4	C	0	0	0	6

On décale la mantisse à droite jusqu'à avoir l'exposant égal à 15.

0.000	0000	0010	0110	0000	1111
0	0	2	6	0	15

0010 0110 (binaire) = 38 (décimal).

Exemple 2 : A00004 -----) FFF4 (Hex.) ----) - (C Hex.)

1.010	0000	0000	0000	0000	0100
A	0	0	0	0	4

On décale la mantisse y compris le bit de signe jusqu'à l'exposant égal à 15.

1111	1111	1111	0100	0000	1111
F	F	F	4	0	15

.../...

Pour convertir en décimal on prend le complément à 2 de FFF4.

FFF4 comple. à 2) 0000 1100(binaire) = - 12(dec.)

Exemple 3 : Si l'exposant est supérieur à 15 (F)

61A711 ----) résultat est : 869C(Hex.) -----) 31076 (déc.) au lieu de 99995 (dec.)

10) FPPNEG

a) Le complément à 2 de la mantisse est formé sans transformer l'exposant.

b) la mantisse est ensuite lue avec le même exposant

Exemple : 10 en décimal ----) A (Hex.)

0.101 0000 0000 0000 0000 0100

complément à 2 de la mantisse :

1.011 0000 0000 0000 0000 0100

B 0 0 0 0 4

10 (50 0004) --------) - 10 (B0 0004)

11) FPST.

C'est une soubroutine dont on n'a pas directement accés, elle n'est pas réentrante.

- Le chiffre à convertir se trouve dans une adresse mémoire (00,01,02).

- Tester le nombre s'il n'est pas nul ou négatif

- si le nombre est nul, le résultat est directement donné sur l'écran, s'il est négatif, il sera converti en un nombre positif.

- La soubroutine FPPFIX est appliquée à ce nombre.

.../...

- Le résultat en décimal est donné seulement sur l'écran de visualisation.

12) STFP

C'est aussi une soubroutine non réentrante.

- Le chiffre en décimal est écrit sur l'écran de visualisation.

- Le premier caractère est testé si c'ast un signe moins, une virgule ou un chiffre supérieur ou inférieur à 9.

- Le soubroutine FPPFLT est appliquée au nombre

- le résultat est déplacé à sa place réservée (00,01,02).

II - B - PROGRAMMES APPELANTS.

1°) Nécessité de ces programmes.

Tlle qu'elle se présente, la bibliotèque mathématiques (BIBMAT) fonctionne comme étant un tout indivisible. L'ensemble est formé de plusieurs sous - programmes qui sont appelés par n'importe qu'elle opération et à chaque fois qu'elle en a besoin.

Pour exécuter une opération, il suffit de donner son code opération (O.C) après avoir appelé tout le programme BIBMAT. Ainsi tout accès à une opération n'est possible qu'après un OC sur l'écran de visualisation; Ce phénomène se répette indéfiniment après l'éxécution de l'opération juste avant (voir exemple page :41)

Une autre caractéristique du programme BIBMAT est relative aux résultats des opérations qui sont à chaque fois donnés aux adresses mémoires 00; 01; 02: remarque donc, que deux inconvénients se présentent, le cycle fermé de BIBMAT d'une part et l'emplacement des résultats d'autre part.

Pour résoudre une équation différentielle, il est nécessaire d'introduire d'abord une table de données en virgule flottante (3 bytes par donnée), cette table doit avoir un début et une fin. La résolution d'une équation va évidemment utiliser les différentes opérations de ce programme Pour les calculs intermédiaires. Les résultats sont ensuite écris sous forme de table.

Après ce bref apperçu sur le déroulement de la résolution d'une équation différentielle, on constate que la bibliothèque mathématiques ne se prête pas facilement. Pour cela il faut concevoir un programme appelant pour chaque fonction en tenant compte à chaque fois de l'initialisation de la pile (Stack) et encore le chargement du registre d'index (Index Register) pour FPPFLT, FPPFIX et FPPNEG.

.../...

Ces programmes appelants doivent être conçus de manière à donner le résultat dans une position d'adresse voulu de notre table.

Ainsi à l'aide de ses programmes appelants, on pourra travailler en dehors du cycle de BIBMAT mais en la gardant comme source pour l'appel des séquences.

2°) Appel de FPPAD; FPPSUB; FPPMUL; FPPDIV

Dans cette partie, on distingue 2 sortes de programmes. D'abord des programmes qui exécutent des opérations indépendantes mais en dehord du cycle de la bibliothèque mathématiques (voir exemple page 39). Puis une 2ème catégorie de programmes qui sont conçus de manière à être directement utilisés par le programme principal de résolution de l'équation différentielle.

3°) Appel de FPPNEG;FPPFIX;FPPCPY;FPPCMP et FPPFLT.

Tous ces programmes sont appelés par le programme principal de résolution de l'équation différentielle. Mais le plus utilisé c'est FPPCPY qui nous permet le déplacement d'une valeur d'une adresse à une autre.

```
00100                    ***ADDITION
00110                    *-----------
00120 0100 8E 00FF       LDS     #$FF       INIT. POINT.DE PILE
00130 0103 BD 85E1       JSR     SETUP2     INTRODUIRE LES 2 NBRES DECI.
00140 0106 BD 88D5       JSR     FPPADD
00150 0109 86 20         LDA A   #'         SORTIE D'1 BLANC
00160 010B BD F9CF       JSR     OUTCH
00170 010E 86 00         LDA A   #A1        POINTER ARG.A1
00180 0110 BD 8732       JSR     FPST       SORTIE DU RESULTAT
00190 0113 3F            SWI
00200                    *
00210                    ***SOUSTRACTION
00220                    *--------------
00230 0114 8E 00FF       LDS     #$FF
00240 0117 BD 85E1       JSR     SETUP2
00250 011A BD 887D       JSR     FPPSUB
00260 011D 86 0D         LDA A   #'
00270 011F BD F9CF       JSR     OUTCH
00280 0122 86 00         LDA A   #A1
00290 0124 BD 8732       JSR     FPST
00300 0127 3F            SWI
00310                    *
00320                    ***MULTIPLICATION
00330                    *----------------
00340 0128 8E 00FF       LDS     #$FF
00350 012B BD 85E1       JSR     SETUP2
00360 012E BD 897A       JSR     FPPMUL
00370 0131 86 20         LDA A   #'
00380 0133 BD F9CF       JSR     OUTCH
00390 0136 86 00         LDA A   #A1
00400 0138 BD 8732       JSR     FPST
00410 013B 3F            SWI
00420                    *
00430                    ***DIVISION
00440                    *----------
00450 013C 8E 00FF       LDS     #$FF
00460 013F BD 85E1       JSR     SETUP2
00470 0142 BD 89EA       JSR     FPPDIV
00480 0145 86 20         LDA A   #'
00490 0147 BD F9CF       JSR     OUTCH
00500 014A 86 00         LDA A   #A1
00510 014C BD 8732       JSR     FPST
00520 014F 3F            SWI
00530      0000          END
```

PROGRAMME DES 4 OPERATIONS UTILISEES POUR LA RESOLUTION DE L'EQUATION DIFFERENTIELLE:

```
00010 0000 0003  A1      RMB     3          RESERVER 3 BYTES POUR ARG.A1
00020 0003 0003  A2      RMB     3          RESERVER 3 BYTES POUR ARG.A2
00030      88D5  FPPXXX  EQU     $88D5      (XXX=ADD;SOUS;MUL;DIV)
00040 0200               ORG     $200
00050 0200 8E 00FF       LDS     #$FF       INITIALISATION DE LA PILE
00060 0203 86 00         LDA A  ##A1        POINTER ADRESSE DE ARG.A1
00070 0205 C6 03         LDA B   #A2          "       #     "  ARG.A2
00080 0207 BD 88D5       JSR     FPPXXX     ADD;SOUS;MUL;DIV.
00090 020A 3F            SWI
```

```
                         ***FPPCLR
                         ---------
00010 0277 86 00         LDA A   #A1        POINTER ADRESSE DE ARG.A1
00020 0279 BD 8ACC       JSR     FPPCLR
00030 027C 3F            SWI
00040      0000          END
```

```
00100                    ***FPPFIX
00110                    *----------
00120 0230                       ORG     $230
00130 0230 8E 00FF               LDS     #$FF
00140 0233 CE 0000               LDX     #A1        MANTISSE DE A1 DS REG.INDEX
00150 0236 86 00                 LDA A   #A1        EXPOSANT DE A1
00160 0238 BD 8820               JSR     PSH4       STOCKER DS LA PILE
00170 023B 30                    TSX
00180 023C E6 00                 LDA B   0,X
00190 023E 86 00                 LDA A   #A1        POINTER ADRESSE DE A1
00200 0240 BD 8A94               JSR     FPPFIX     CONVER.DEC. EN ENTIER
00210 0243 31                    INS
00220 0244 31                    INS
00230 0245 31                    INS
00240 0246 31                    INS
00250 0247 3F                    SWI
00260                    *
00270                    ***FPPNEG
00280                    *----------
00290 0248 8E 00FF               LDS     #$FF
00300 024B CE 0000               LDX     #A1        MANTISSE DE A1 DS REG.INDEX
00310 024E 86 00                 LDA A   #A1        EXPOSANT DE A1 DS ACC A
00320 0250 BD 8820               JSR     PSH4       STOCKER DS PILE
00330 0253 30                    TSX
00340 0254 E6 00                 LDA B   0,X
00350 0256 17                    TBA                SAUVER ACC B DS ACC A
00360 0257 BD 8ABB               JSR     FPPNEG
00370 025A DF 20                 STX     $20        20 ADRESSE MEMOIRE QUELCONQUE
00380 025C D6 21                 LDA B   $21
00390 025E 96 00                 LDA A   $00        RESULTAT DS ADRESSE 0
00400 0260 BD 885B               JSR     FPPCPY
00410 0263 3F                    SWI
00420                    *
00430                    ***FPPCMP
00440                    *----------
00450 0264 8E 00FF               LDS     #$FF
00460 0267 86 00                 LDA A   #A1        POINTER ADRESSE DE A1
00470 0269 C6 03                 LDA B   #A2        "  " DE A2
00480 026B BD 883A               JSR     FPPCMP
00490 026E 3F                    SWI
00500                    *LE RESULTAT DS LE REG. CODE CONDITION
00510                    *
00520                    ***FPPCPY
00530                    *----------
00540 026F 86 03                 LDA A   #A2        POINTER ADRESSE MEMOIRE A COPIE
00550 0271 C6 00                 LDA B   #A1        "       "       "     DU NBR.
00560 0273 BD 885B               JSR     FPPCPY
00570 0276 3F                    SWI
00100                    *
00110                    ***FPPFLT:CONVER.ENTIER(16BITS)EN VERG. FLOT.
00120                    *
00140          8820      PSH4    EQU     $8820
00150          8A81      FPPFLT  EQU     $8A81
00160 0210                       ORG     $210
00170 0210 8E 00FF               LDS     #$FF
00180 0213 CE 0000               LDX     #A1        MANTISSE DE ARG.A1 DS REG.IND
00190 0216 96 00                 LDA A   $0         METTRE EXPO. A ZERO(NBRE ENTIER
00200 0218 BD 8820               JSR     PSH4       INTRODUIT DS LA PILE
00210 021B 30                    TSX
00220 021C E6 00                 LDA B   0,X
00230 021E 86 00                 LDA A   #A1        POINTER ADRESSE DE A1
00240 0220 BD 8A81               JSR     FPPFLT     CONVERTIR
00250 0223 31                    INS                INCREMENTER LE POINTEUR DE PILE
00260 0224 31                    INS
00270 0225 31                    INS
00280 0226 31                    INS
00290 0227 3F                    SWI
00300                    *LE RESULTAT EST A LA PLACE DE A1
00310          0000              END
```

```
   8500;G
?0C1
?342
    558009
?34.8
     459A06
5E3309 376.80
?0C1
?.12
    7AF1FD
?-45
    A60006
A63D06 -44.882
?0C2
?7845
     7A940D
?5679
     58BC0D
43B00C 2166.0
?0C2
?.9567
      7A7500
?56.002
       700106
91E906 -55.046
?0C3
?4590
     47B80D
?-10
    B00004
A65A10 -45900
?0C3
?56058
      6D7D10
?.568
     46B400
7C610F 31841
?0C4
?45802
      597510
?78596
       4CC211
4A9700 .58276
?0C4
?4521
      46A40D
?45
    5A0006
647707 100.46
?0C5
?000465
400003 4.0000
?0C6
?12.5
      640004 000C03
?0C7
?54679
       6ACC10
953410 -54680
?0C8
000000 0
?0C
```

```
?0C1
?00456
       720009
?765904
        5D7F14
5D8D14 766360
?0C1
?.004589
         4B30F9
?.056438
         7396FC
7CFCFC .061032
?0C2
?345672
        546513
?785439
         5FE114
94A413 -439740
?0C2
?.004324
         46D8F9
?.005437
         5914F9
B710F7 -.0011129
?0C3
?34567
       438410
?87690
       55A311
5A5720 3031400000
?0C3
?.004583
         4B16F9
?.0003276
           55E1F5
64C1ED .0000015014
?0C4
?54/
     6C0006
?00
    000004
7FFF7F 17014000000000000000000000000000000000000000
?0C4
?00
    000004
?46
    5C0006
0000FD 0
?0C5
?234500
468A0E 9029.2
?0C6
?-2565
        AFD80C F5FB0E
?0C7
?-23
     A40005
5C0005 23.000
?0C8
000000 0
?0C
```

MOTOROLA M6800 CROSS ASSEMBLER, RELEASE 1.0

```
0001                    NAM    TSTFPP
0002                    *
0003                    *
0004                    *      24 BIT REENTRANT FLOATING POINT PACKAGE
0005                    *
0006                    *      WITH AUXILIARY CONTROL/TEST PROGRAM
0007                    *
0008                    *
0009                    *      PROGRAMMER:
0010                    *
0011                    *      DR. BENTON D. WEATHERS
0012                    *      SANGAMO ELECTRIC COMPANY
0013                    *      BOX 3347
0014                    *      SPRINGFIELD, ILLINOIS 62714
0015                    *
0016                    *      217/544-6411 EXT 591
0017                    *
0018 0000               A1     ORG    $0       DEFINITION OF MEMORY ADDRESSES IN RAM
0019 0000 0003          A1     RMB    3        RESERVE 3 BYTES OF RAM
0020 0003 0003          A2     RMB    3        RESERVE THREE BYTES OF RAM
0021 0006 0002          PSH4T1 RMB    2        WORKING LOCATION FOR PSH4
0022 0008 0002          PSH4T2 RMB    2        WORKING LOCATION FOR PSH4
0023 000A 0001          L1     RMB    1        LOOP COUNTER
0024 000B 0001          L2     RMB    1        LOOP COUNTER
0025 000C 0003          A3     RMB    3        HOLDS MAIN COPY OF ARG1
0026 000F 0002          ROUADD RMB    2        HOLDS SUBROUTINE ADDRESS
0027                    *
0028        FA8B        INCH   EQU    $FA8B    SUBROUTINE ADDRESS
0029        F9DC        OUTCH  EQU    $F9DC    SUBROUTINE ADDRESS
```

RECTIFICATIONS : OUTCH F9CF
INCH FA7F

```
00031                    *
00032                    *
00033                    *       MAIN CONTROL PROGRAM FOR TESTING
00034                    *         THE FLOATING POINT PACKAGE
00035                    *
00036                    *
00037 8500                       ORG     $8500
00038 8500 8E 00FF               LDS     #$FF        INITIALIZE STACK POINTER
00039                    *
00040                    *       MAIN TESTING LOOP
00041                    *
00042 8503 BD 868A LOOP          JSR     QUERY       NEWLINE, CR ON TTY
00043 8506 86 4F                 LDA A   #'O         OUTPUT 'O'
00044 8508 BD F9DC               JSR     OUTCH
00045 850B 86 43                 LDA A   #'C         OUTPUT 'C'
00046 850D BD F9DC               JSR     OUTCH
00047 8510 BD FA8B               JSR     INCH        GET OPERATION CODE
00048                    *
00049                    *       SEVEN WAY BRANCH
00050                    *
00051 8513 81 31                 CMP A   #'1         TEST FOR '1'
00052 8515 27 23                 BEQ     ADD
00053 8517 81 32                 CMP A   #'2         TEST FOR '2'
00054 8519 27 27                 BEQ     SUB
00055 851B 81 33                 CMP A   #'3         TEST FOR '3'
00056 851D 27 2B                 BEQ     MUL
00057 851F 81 34                 CMP A   #'4         TEST FOR '4'
00058 8521 27 2F                 BEQ     DIV
00059 8523 81 35                 CMP A   #'5         TEST FOR '5'
00060 8525 27 33                 BEQ     FLOAT
00061 8527 81 36                 CMP A   #'6         TEST FOR '6'
00062 8529 27 40                 BEQ     IFIX
00063 852B 81 37                 CMP A   #'7         TEST FOR '7'
00064 852D 27 4F                 BEQ     NEGATE
00065 852F 81 38                 CMP A   #'8         TEST FOR '8'
00066 8531 27 53                 BEQ     CLEAR
00067                    *
00068 8533 86 58                 LDA A   #'X         INVALID OP-CODE, OUTPUT 'X'
00069 8535 BD F9DC               JSR     OUTCH
00070 8538 20 C9                 BRA     LOOP        GO GET NEXT OP-CODE
00071                    *
00072 853A BD 85E1 ADD           JSR     SETUP2
00073 853D CE 88D5               LDX     #FPPADD     LOAD ADDRESS INTO INDEX REG
00074 8540 20 4B                 BRA     PRINT
00075                    *
00076 8542 BD 85E1 SUB           JSR     SETUP2
00077 8545 CE 887D               LDX     #FPPSUB     LOAD ADDRESS
00078 8548 20 43                 BRA     PRINT
00079                    *
00080 854A BD 85E1 MUL           JSR     SETUP2
00081 854D CE 897A               LDX     #FPPMUL
00082 8550 20 3B                 BRA     PRINT
00083                    *
00084 8552 BD 85E1 DIV           JSR     SETUP2
```

```
00085 8555 CE 89EA          LDX    #FPPDIV
00086 8558 20 33            BRA    PRINT
00087                *
00088 855A BD 868A FLOAT    JSR    QUERY
00089 855D 86 03            LDA A  #A2
00090 855F BD 8600          JSR    READ3B
00091 8562 86 00            LDA A  #A1
00092 8564 C6 03            LDA B  #A2
00093 8566 BD 8A61          JSR    FPPFLT
00094 8569 20 52            BRA    PRIN1
00095                *
00096 856B BD 8502 IFIX     JSR    SETUP1
00097 856E BD 8A94          JSR    FPPFIX   FLOATINGPOINT TO INTEGER
00098 8571 86 20            LDA A  #' '     OUTPUT BLANK CHARACTER
00099 8573 BD F90C          JSR    OUTCH
00100 8576 86 00            LDA A  #A1
00101 8578 BD 8648          JSR    WRIT3B
00102 857B 7E 8503          JMP    LOOP
00103                *
00104 857E BD 85D2 NEGATE   JSR    SETUP1
00105 8581 BD 8A8B          JSR    FPPNEG   FLOATING POINT NEGATE
00106 8584 20 37            BRA    PRIN1
00107                *
00108 8586 86 00   CLEAR    LDA A  #A1
00109 8588 BD 8ACC          JSR    FPPCLR
00110 858B 20 30            BRA    PRIN1
00111                *
00112                *       EXECUTION REPETITION LOOP
00113 858D DF CF   PRINT    STX    ROUADD   SAVE THE SUBROUTINE ADDRESS
00114 858F 86 0C            LDA A  #A3
00115 8591 C6 00            LDA B  #A1
00116 8593 BD 8858          JSR    FPPCPY   A3=A1
00117                *
00118                *       INITIALIZE OUTSIDE LOOP
00119                *
00120 8596 86 02            LDA A  #2
00121 8598 97 CA            STA A  L1
00122 859A 7A 000A LOOP1    DEC    L1       START OF OUTSIDE LOOP
00123 859D 26 02            BNE    LOOPA
00124 859F 20 1C            BRA    PRIN1    GET OUT OF EXECUTION LOOP
00125                *
00126                *       INITIALIZE INNER LOOP
00127                *
00128 85A1 86 02   LOOPA    LDA A  #2
00129 85A3 97 CB            STA A  L2
00130 85A5 7A 000B LOOP2    DEC    L2       DECREMENT INNER LOOP COUNTER
00131 85A8 26 02            BNE    LOOPB
00132 85AA 20 EE            BRA    LOOP1    INNER LOOP DONE SO BACK TO OUTER LOO
  133                *
00134                *       EXECUTION SECTION
00135                *
00136 85AC 86 00   LOOPB    LDA A  #A1
00137 85AE C6 0C            LDA B  #A3
00138 85B0 BD 8858          JSR    FPPCPY   A1=A3
```

```
C0139 85B3 86 00          LDA A   #A1
C0140 85B5 C6 03          LDA B   #A2
C0141 85B7 DE 0F          LDX     R6UADD    SET UP R6UTINE ADDRESS
C0142 85B9 AD 00          JSR     0,X
C0143 85BB 20 E8          BRA     L66P2
C0144              *
C0145              *      PRINT RESULTS
C0146              *
C0147 85BD BD 8693 PRIN1  JSR     NEWLIN
C0148 85C0 86 00          LDA A   #A1       SET UP ADDRESS 6F RESULTS
C0149 85C2 BD 864B        JSR     WRIT3B    PRINT RESULTS
C0150 85C5 86 20          LDA A   #'
C0151 85C7 BD F9DC        JSR     6UTCH
C0152 85CA 86 00          LDA A   #A1
C0153 85CC BD 8732        JSR     FPST      PRINT RESULT IN DECIMAL F6RMAT
C0154 85CF 7E 8503        JMP     L66P      G6 GET NEXT 6P-C6DE
```

```
C0156                    *
C0157                    *
C0158                    *       SETUP1, SETUP2
C0159                    *
C0160                    *
C0161 85D2 BD 868A SETUP1 JSR     QUERY
C0162 85D5 86 03          LDA A   #A2
C0163 85D7 BD 869E        JSR     STFP
C0164 85DA 86 03          LDA A   #A2
C0165 85DC BD 8648 (00)   JSR     WRIT3B
C0166 85DF 20 1A          BRA     SUREG     SET UP POINTERS IN REG'S
C0167                    *
C0168 85E1 BD 868A SETUP2 JSR     QUERY     NEWLINE, CR ON TTY
C0169 85E4 86 00          LDA A   #A1       ARG1 TO A1
C0170 85E6 BD 869E        JSR     STFP
C0171 85E9 86 00          LDA A   #A1
C0172 85EB BD 8648        JSR     WRIT3B
C0173 85EE BD 868A        JSR     QUERY
C0174 85F1 86 03          LDA A   #A2       ARG2 TO A2
C0175 85F3 BD 869E        JSR     STFP
C0176 85F6 86 03          LDA A   #A2
C0177 85F8 BD 8648        JSR     WRIT3B
C0178 85FB 86 00    SUREG LDA A   #A1       SET UP ADDRESS IN A REG
C0179 85FD C6 03          LDA B   #A2       SET UP ADDRESS IN B REG
C0180 85FF 39             RTS               RETURN
C0181                    *
C0182                    *
C0183                    *       READ 3 CONSECUTIVE BYTES (6 CHARACTERS)
C0184                    *       ON ENTRY THE STORAGE ADDRESS IS IN THE A REG
C0185                    *       CHARACTER TO HEX CONVERSION IS PERFORMED AND THE
C0186                    *       BYTES PUT AWAY.
C0187                    *
C0188 8600 36       READ3B PSH A             PUT THE MEMORY ADDRESS ON THE STACK
C0189 8601 4F              CLR A
C0190 8602 36              PSH A
C0191                    *
C0192                    *       SET UP THE OUTER LOOP TO GO FROM 3 TO 1
C0193                    *
C0194 8603 86 03           LDA A   #3
C0195 8605 36              PSH A
C0196 8606 86 02    READ31 LDA A   #2        SET UP INNER LOOP TO GO TWICE
C0197 8608 36              PSH A
C0198 8609 5F              CLR B             USE B REG TO MERGE HEX CHARACTERS
C0199 860A BD FA8B READ32  JSR     INCH      GET CHARACTER FROM TTY
C0200 860D 81 30           CMP A   #'0       IS CHR CODE LT ASC '0'
C0201 860F 2D 08           BLT     READ33    YES
C0202 8611 81 39           CMP A   #'9       IS CHR CODE GT ASC '9'
C0203 8613 2E 04           BGT     READ33    YES
C0204 8615 80 30           SUB A   #'0       SUBTRACT CODE FOR ASC '0'
C0205 8617 20 0C           BRA     READ34    GO TO MERGE SECTION
C0206 8619 81 41    READ33 CMP A   #'A       IS CHAR CODE LT ASC 'A'
C0207 861B 2D 27           BLT     READ35    YES, INVALID CHARACTER
C0208 861D 81 46           CMP A   #'F       IS CHAR CODE GT ASC 'F'
C0209 861F 2E 23           BGT     READ35    YES, INVALID CODE
```

```
00210 8621 80 41          SUB A  #'A      SUBTRACT CODE FOR ASC 'A'
00211 8623 8B 0A          ADD A  #10      ADD 10
00212 8625 58       READ34 ASL B          MERGE A REG WITH B REG
00213 8626 58              ASL B
00214 8627 58              ASL B
00215 8628 58              ASL B
00216 8629 1B              ABA            MERGE
00217 862A 16              TAB            PUT RESULT IN B REG
00218 862B 30              TSX            INDEX REG TO TOP OF STACK
00219 862C 6A 00           DEC    0,X     DECREMENT INNER LOOP COUNTER
00220 862E 6D 00           TST    0,X     IS INNER LOOP CNTR POS
00221 8630 2E D8           BGT    READ32  YES, CONTINUE LOOPING
00222                 *
00223                 *      PUT THE CHARACTER AWAY
00224                 *
00225 8632 EE 02           LDX    2,X     GET ADDRESS OFF THE STACK
00226 8634 E7 00           STA B  0,X     STORE MERGED RESULT
00227 8636 30              TSX            INDEX TO TOP OF STACK
00228 8637 6C 03           INC    3,X     INC LSB OF MEMORY ADDRESS
00229 8639 31              INS            REMOVE INNER LOOP CNTR FRM STACK
00230 863A 6A 01           DEC    1,X     DECREMENT OUTER LOOP COUNTER
00231 863C 6D 01           TST    1,X     IS OUTER LOOP CNTR POSITIVE
00232 863E 2E C6           BGT    READ31  YES, GO GET NEXT BYTE
00233 8640 31              INS            CLEAN UP STACK
00234 8641 31              INS
00235 8642 31              INS
00236 8643 39              RTS            RETURN
00237                 *
00238                 *      INVALID CHARACTER
00239                 *
00240 8644 86 58    READ35 LDA A  #'X      OUTPUT AN 'X' CHARACTER
00241 8646 BD F9DC         JSR    OUTCH
00242 8649 20 BF           BRA    READ32  REREAD CHARACTER
00243                 *
00244                 *
00245                 *      WRITE THREE CONSECUTIVE BYTES
00246                 *
00247                 *
00248 864B 36       WRIT30 PSH A          PUSH DATA POINTER ONTO STACK
00249 864C 4F              CLR A
00250 864D 36              PSH A
00251 864E 86 03           LDA A  #3       SET OUTER LOOP TO GO 3 TIMES
00252 8650 36              PSH A
00253 8651 34              DES            ALLOCATE SPACE ON STACK FOR L2 CNTR
00254 8652 30       WRIT31 TSX            INDEX TO TOP OF STACK
00255 8653 EE 02           LDX    2,X     GET DATA POINTER
00256 8655 E6 00           LDA B  0,X     DATA TO B REG
00257 8657 30              TSX            INDEX TO TOP OF STACK
00258 8658 6C 03           INC    3,X     INCREMENT BYTE POINTER
00259 865A 86 01           LDA A  #1      INNER LOOP INITIALIZATION
00260 865C A7 00           STA A  0,X
00261                 *
00262 865E 17       WRIT32 TBA            UNPACK FOUR BITS, B REG TO A REG
00263 865F 6D 00           TST    0,X     IS INNER LOOP EQUAL TO ZERO
```

```
C0264 8661 27 04          BEQ     WRIT35   YES, SKIP SHIFT OPERATION
C0265 8663 44             LSR A            LOGIC SHIFT RIGHT FOUR TIMES
C0266 8664 44             LSR A
00267 8665 44             LSR A
00268 8666 44             LSR A
C0269 8667 84 0F  WRIT35  AND A   #15      MASK OUT 4 MOST SIGNIFICANT BITS
C0270             *       UNPACKING COMPLETE
C0271 8669 81 0A          CMP A   #10      IS 4-BIT FIELD LT 10
C0272 866B 2D 06          BLT     WRIT33   YES(DIGIT 0 - 9)
C0273 866D 80 0A          SUB A   #10      NO (LETTER A - F)
C0274 866F 8B 41          ADD A   #'A      ADD ASCII 'A' CODE TO RESULT
C0275 8671 20 02          BRA     WRIT34   GO TO OUTPUT SECTION
00276 8673 8B 30  WRIT33  ADD A   #'0      ADD ASCII '0' CODE TO RESULT
C0277 8675 BD F9DC WRIT34 JSR     OUTCH    OUTPUT CHARACTER
C0278 8678 30             TSX              INDEX TO TOP OF STACK
C0279 8679 6A 00          DEC     0,X      INNER LOOP TEST
C0280 867B 6D 00          TST     0,X      IS INNR LOOP CNTR GE 0
C0281 867D 2C DF          BGE     WRIT32   YES, CONTINUE WITH LOOP
C0282 867F 6A 01          DEC     1,X      OUTER LOOP TEST
C0283 8681 6D 01          TST     1,X      IS OUTER LOOP CNTR GT 0
C0284 8683 2E CD          BGT     WRIT31   YES, GET NEXT BYTE
C0285 8685 31             INS              CLEAN UP STACK
C0286 8686 31             INS
C0287 8687 31             INS
C0288 8688 31             INS
C0289 8689 39             RTS              RETURN
C0290             *
C0291             *
C0292             *       QUERY SUBROUTINE
C0293             *
C0294             *
C0295 868A BD 8693 QUERY  JSR     NEWLIN   CAUSE TTY TO SPACE TO NEW LINE
C0296 868D 86 3F          LDA A   #'       OUTPUT QUESTION MARK
C0297 868F BD F9DC        JSR     OUTCH
C0298 8692 39             RTS
C0299             *
C0300             *
C0301             *       NEWLINE
C0302             *
C0303             *
C0304 8693 86 0D  NEWLIN  LDA A   #13      OUTPUT CARRIAGE RETURN
C0305 8695 BD F9DC        JSR     OUTCH
C0306 8698 86 0A          LDA A   #10      OUTPUT LINE FEED
C0307 869A BD F9DC        JSR     OUTCH
C0308 869D 39             RTS              RETURN
```

```
C0310                    *
C0311                    *
00312                    *       STRING TO FLOATING POINT ROUTINE
00313                    *
C0314                    *
00315 869E 36       STFP PSH A              PUSH DATA ADDRESS ONTO STACK
C0316 869F 4F            CLR A
C0317 86A0 36            PSH A
00318 86A1 36            PSH A              STFPDP=0 (DEC PNT FLAG)
C0319 86A2 36            PSH A              STFPSG=0 (NEGATIVE SIGN FLAG)
00320 86A3 CE 0000       LDX   #0
C0321 86A6 BD 8820       JSR   PSH4         STFPA1=0.0
C0322 86A9 CE 4000       LDX   #$4000
C0323 86AC 86 01         LDA A #1
C0324 86AE BD 8820       JSR   PSH4         STFPA2=1.0
C0325 86B1 BD 8820       JSR   PSH4         STFPA3=
C0326 86B4 CE 5000       LDX   #$500C
C0327 86B7 86 04         LDA A #4
C0328 86B9 BD 8820       JSR   PSH4         STFPA4=10.0
00329 86BC 30            TSX                INDEX POINTS TO TOP OF STACK
00330                    *
C0331                    *       STACK OFFSET DEFINITIONS
C0332                    *
C0333      0001     STFPA4 EQU   1
00334      0005     STFPA3 EQU   5
00335      0009     STFPA2 EQU   9
00336      000D     STFPA1 EQU   13
C0337      0010     STFPSG EQU   16         NEGATIVE SIGN FLAG
C0338      0011     STFPDP EQU   17         DECIMAL POINT FLAG
C0339      0012     STFPRS EQU   18         RESULT ADDRESS
00340                    *
C0341                    *       CHARACTER PROCESSING LOOP
C0342                    *
C0343 86BD BD FA8B STFP1 JSR   INCH         GET A CHARACTER INTO A REG
00344 86C0 81 0A         CMP A #$A          LINE FEED CHARACTER
C0345 86C2 27 43         BEQ   STFP4        YES
C0346 86C4 81 2D         CMP A #'-          MINUS SIGN
C0347 86C6 26 04         BNE   STFP2        NO
C0348 86C8 6C 10         INC   STFPSG,X     YES, SET NEGATIVE SIGN FLAG
00349 86CA 20 F1         BRA   STFP1
C0350 86CC 81 2E    STFP2 CMP A #'.         DECIMAL POINT
C0351 86CE 26 04         BNE   STFP3        NO
C0352 86D0 6C 11         INC   STFPDP,X     SET DECIMAL POINT FLAG
C0353 86D2 20 E9         BRA   STFP1
C0354 86D4 81 40    STFP3 CMP A #$40        $40 IS CODE FOR 10
C0355 86D6 2A E5         BPL   STFP1
C0356 86D8 81 30         CMP A #'0
C0357 86DA 2B E1         BMI   STFP1
00358                    *
C0359                    *       DIGIT DETECTED
C0360                    *
C0361 86DC 80 30         SUB A #'0          SUBTRACT CODE FOR ZERO
C0362 86DE A7 06         STA A STFPA3+1,X   STORE INTEGER IN A3
C0363 86E0 6F 05         CLR   STFPA3,X
```

```
C0364 86E2 A6 04          LDA A  STFPA3-1,X
C0365 86E4 16             TAB
C0366 86E5 BD 8A21        JSR    FPPFLT   A3=FLOAT(A3)
C0367 86E8 30             TSX             RESTORE INDEX
C0368 86E9 A6 0C          LDA A  STFPA1-1,X
C0369 86EB E6 00          LDA B  STFPA4-1,X
C0370 86ED BD 897A        JSR    FPPMUL   A1=A1*10*0
C0371 86F0 30             TSX
C0372 86F1 A6 0C          LDA A  STFPA1-1,X
C0373 86F3 E6 04          LDA B  STFPA3-1,X
C0374 86F5 BD 88D5        JSR    FPPADD   A1=A1+A3
C0375 86F8 30             TSX
C0376 86F9 6D 11          TST    STFPDP,X   IS DECIMAL POINT FLAG SET
C0377 86FB 2F C0          BLE    STFP1    NO
C0378 86FD A6 08          LDA A  STFPA2-1,X  YES, CALCULATE PWR OF 10.
C0379 86FF E6 00          LDA B  STFPA4-1,X
C0380 8701 BD 897A        JSR    FPPMUL   A2=A2*10.
C0381 8704 30             TSX
C0382 8705 20 B6          BRA    STFP1
C0383                *
C0384                *    CLEANUP
C0385                *
C0386 8707 A6 0C    STFP4 LDA A  STFPA1-1,X
C0387 8709 E6 08          LDA B  STFPA2-1,X
C0388 870B BD 89EA        JSR    FPPDIV   A1=A1/A2
C0389 870E 30             TSX
C0390 870F 6D 10          TST    STFPSG,X
C0391 8711 27 07          BEQ    STFP5
C0392 8713 A6 0C          LDA A  STFPA1-1,X  SIGN FLAG SET
C0393 8715 16             TAB
C0394 8716 BD 8A8B        JSR    FPPNEG   A1=-A1
C0395 8719 30             TSX
C0396 871A A6 00    STFP5 LDA A  STFPA1,X  MOVE TO RESULT AREA
C0397 871C E6 0E          LDA B  STFPA1+1,X
C0398 871E EE 12          LDX    STFPRS,X
C0399 8720 A7 00          STA A  0,X
C0400 8722 E7 01          STA B  1,X
C0401 8724 30             TSX
C0402 8725 A6 0F          LDA A  STFPA1+2,X
C0403 8727 EE 12          LDX    STFPRS,X
C0404 8729 A7 02          STA A  2,X
C0405 872B 86 14          LDA A  #20
C0406 872D 31       STFP6 INS
C0407 872E 4A             DEC A
C0408 872F 2E FC          BGT    STFP6    LOOP TO CLEAN UP THE STACK
C0409 8731 39             RTS
```

```
C0411                    *
C0412                    *
C0413                    *       FLOATING POINT TO STRING CONVERSION ROUTINE
C0414                    *
C0415                    *
C0416 8732 36            FPST    PSH A           VALUE ADDRESS ONTO STACK
C0417 8733 4F                    CLR A
C0418 8734 36                    PSH A
C0419 8735 30                    TSX             INDEX TO TOP OF STACK
C0420 8736 86 05                 LDA A  #5
C0421 8738 36                    PSH A           SD=5
C0422 8739 36                    PSH A           DP=5
C0423 873A EE 00                 LDX    0,X      GET VALUE ONTO THE STACK
C0424 873C A6 02                 LDA A  2,X      EXPONENT INTO THE A REG
C0425 873E EE 00                 LDX    0,X      MANTISSA INTO THE INDEX REG
C0426 8740 BD 8820               JSR    PSH4     A1=INPUT VALUE
C0427 8743 CE 5000               LDX    #$500C
C0428 8746 86 04                 LDA A  #4
C0429 8748 BD 8820               JSR    PSH4     A2=1C.
C0430 874B CE 4E20               LDX    #$4E2C
C0431 874E 86 0E                 LDA A  #14
C0432 8750 BD 8820               JSR    PSH4     A3=10000.
C0433 8753 CE 61A8               LDX    #$61A8
C0434 8756 86 11                 LDA A  #$11
C0435 8758 BD 8820               JSR    PSH4     A4=100000.
C0436 875B BD 8820               JSR    PSH4     A5=SCRATCH
C0437 875E 30                    TSX             RESTORE INDEX
C0438                    *
C0439                    *       STACK OFFSET DEFINITIONS
C0440                    *
C0441       0001         FPSTA5  EQU    1
C0442       0005         FPSTA4  EQU    5
C0443       0009         FPSTA3  EQU    9
C0444       000D         FPSTA2  EQU    13
C0445       0011         FPSTA1  EQU    17
C0446       0014         FPSTDP  EQU    20
C0447       0015         FPSTSD  EQU    21
C0448       0016         FPSTVL  EQU    22
C0449                    *
C0450 875F 6D 11                 TST    FPSTA1,X  TEST INPUT VALUE
C0451 8761 27 04                 BEQ    FPST1     BRANCH IF ZERO
C0452 8763 2B 0A                 BMI    FPST2     BRANCH IF NEGATIVE
C0453 8765 20 14                 BRA    FPST3     NUMBER IS READY TO BE SCALED
C0454                    *
C0455                    *       ZERO VALUE
C0456                    *
C0457 8767 86 30         FPST1   LDA A  #'0       PRINT A SINGLE ZERO
C0458 8769 BD F9DC               JSR    OUTCH
C0459 876C 7E 8819               JMP    FPST1C    GO CLEAN UP STACK
C0460                    *
C0461                    *       NEGATIVE NUMBER
C0462                    *
C0463 876F 86 2D         FPST2   LDA A  #'-       PRINT MINUS SIGN
C0464 8771 BD F9DC               JSR    OUTCH
```

```
C0465 8774 A6 10          LDA A  FPSTA1-1,X
C0466 8776 16             TAB
C0467 8777 BD 8ABB        JSR    FPPNEG   A1=-A1
C0468 877A 30             TSX
C0469                     *
C0470                     *       NORMALIZING LOOP
C0471                     *
C0472 877B A6 10    FPST3 LDA A  FPSTA1-1,X
C0473 877D E6 08          LDA B  FPSTA3-1,X
C0474 877F BD 883A        JSR    FPPCMP   TEST(A1-10000.)
C0475 8782 30             TSX             RESET INDEX
C0476 8783 2A 0C          BPL    FPST4    BRANCH IF ZERO OR POSITIVE
C0477 8785 A6 10          LDA A  FPSTA1-1,X
C0478 8787 E6 0C          LDA B  FPSTA2-1,X
C0479 8789 BD 897A        JSR    FPPMUL   A1=A1*10.
C0480 878C 30             TSX
C0481 878D 6A 14          DEC    FPSTDP,X   DP=DP-1
C0482 878F 20 EA          BRA    FPST3    CONTINUE LOOP
C0483                     *
C0484 8791 A6 10    FPST4 LDA A  FPSTA1-1,X
C0485 8793 E6 04          LDA B  FPSTA4-1,X
C0486 8795 BD 883A        JSR    FPPCMP   TEST (A1-100000.)
C0487 8798 30             TSX
C0488 8799 2B 0C          BMI    FPST5    BRANCH IF NEGATIVE
C0489 879B A6 10          LDA A  FPSTA1-1,X
C0490 879D E6 0C          LDA B  FPSTA2-1,X
C0491 879F BD 89EA        JSR    FPPDIV   A1=A1/10.
C0492 87A2 30             TSX
C0493 87A3 6C 14          INC    FPSTDP,X   DP=DP+1
C0494 87A5 20 EA          BRA    FPST4    CONTINUE LOOP
C0495                     *
C0496                     *       TEST TO SEE IF DP AND LEADING ZEROS ARE NEEDED
C0497                     *
C0498 87A7 6D 14    FPST5 TST    FPSTDP,X   IS DP CNT NEGATIVE
C0499 87A9 2A 12          BPL    FPST7    NO
C0500 87AB 86 2E          LDA A  #'.      OUTPUT DECIMAL POINT
C0501 87AD BD F9DC        JSR    OUTCH
C0502 87B0 86 30    FPST6 LDA A  #'0      OUTPUT LEADING ZEROS
C0503 87B2 BD F9DC        JSR    OUTCH
C0504 87B5 6C 14          INC    FPSTDP,X
C0505 87B7 2B F7          BMI    FPST6
C0506 87B9 86 FF          LDA A  #$FF     RESET DP CNT TO NEGATIVE VALUE
C0507 87BB A7 14          STA A  FPSTDP,X
C0508                     *
C0509                     *       LOOP TO PRINT FIVE SIGNIFICANT DIGITS
C0510                     *
C0511 87BD 6A 15    FPST7 DEC    FPSTSD,X   SD=SD-1
C0512 87BF 2B 48          BMI    FPST9    GET OUT IF 5 SD HAVE GONE
C0513 87C1 6D 14          TST    FPSTDP,X   SHOULD DECIMAL BE OUTPUT
C0514 87C3 26 05          BNE    FPST8    NO
C0515 87C5 86 2E          LDA A  #'.      OUTPUT DECIMAL POINT
C0516 87C7 BD F9DC        JSR    OUTCH
C0517 87CA 6A 14    FPST8 DEC    FPSTDP,X   DP=DP-1
C0518 87CC A6 0C          LDA A  FPSTA5-1,X  DATA SINK
```

```
00519 87CE E6 10          LDA B  FPSTA1-1,X   DATA SOURCE
00520 87D0 BD 885B        JSR    FPPCPY   A5=A1
00521 87D3 30             TSX
00522 87D4 A6 00          LDA A  FPSTA5-1,X
00523 87D6 E6 08          LDA B  FPSTA3-1,X
00524 87D8 BD 89EA        JSR    FPPDIV   A5=A5/A3
00525 87DB 30             TSX
00526 87DC A6 00          LDA A  FPSTA5-1,X
00527 87DE 16             TAB
00528 87DF BD 8A94        JSR    FPPFIX   A5=IFIX(A5)
00529 87E2 30             TSX
00530 87E3 A6 02          LDA A  FPSTA5+1,X   OUTPUT DIGIT
00531 87E5 8B 30          ADD A  #'0       ADD CHARACTER CODE FOR ZERO
00532 87E7 BD F9DC        JSR    OUTCH
00533 87EA A6 00          LDA A  FPSTA5-1,X
00534 87EC 16             TAB
00535 87ED BD 8A81        JSR    FPPFLT   A5=FLOAT(A5)
00536 87F0 30             TSX
00537 87F1 A6 00          LDA A  FPSTA5-1,X
00538 87F3 E6 08          LDA B  FPSTA3-1,X
00539 87F5 BD 897A        JSR    FPPMUL   A5=A5*A3
00540 87F8 30             TSX
00541 87F9 A6 10          LDA A  FPSTA1-1,X
00542 87FB E6 00          LDA B  FPSTA5-1,X
00543 87FD BD 887D        JSR    FPPSUB   A1=A1-A5
00544 8800 30             TSX
00545 8801 A6 08          LDA A  FPSTA3-1,X
00546 8803 E6 0C          LDA B  FPSTA2-1,X
00547 8805 BD 89EA        JSR    FPPDIV   A3=A3/10.
00548 8808 30             TSX
00549 8809 7E 878D        JMP    FPST7    CONTINUE LOOP
00550                     *
00551                     *      OUTPUT TRAILING ZEROS, IF NECESSARY
00552                     *
00553 880C 6D 14    FPST9 TST    FPSTDP,X   ARE TRAILING ZEROS REQUIRED
00554 880E 2F 09          BLE    FPST10   NO
00555 8810 6A 14          DEC    FPSTDP,X
00556 8812 86 30          LDA A  #'0      OUTPUT ZERO
00557 8814 BD F9DC        JSR    OUTCH
00558 8817 20 F3          BRA    FPST9    CONTINUE LOOP
00559                     *
00560                     *      CLEANUP SECTION
00561                     *
00562 8819 86 18    FPST10 LDA A  #24
00563 881B 31       FPST11 INS              INCREMENT STACK POINTER
00564 881C 4A              DEC A            DECREMENT COUNT
00565 881D 2E FC           BGT    FPST11    CONTINUE WHILE COUNT IS POSITIVE
00566    F 39              RTS              RETURN TO CALL
```

```
C0568                   *
C0569                   *       ROUTINE TO PUSH FOUR BYTES ON THE STACK.  NOT REENTRA
C0570                   *       PUSHES EXPONENT (FROM A REG), LS BYTE, MS BYTE OF MAN
C0571                   *       FOLLOWED BY A POINTER TO MS BYTE OF THE MANTISSA
C0572                   *
C0573 8820 DF 06  PSH4  STX     PSH4T1  SAVE THE MANTISSA
C0574 8822 30           TSX             INDEX TO TOP OF STACK
C0575 8823 EE 00        LDX     0,X     RETURN ADDRESS TO INDEX
C0576 8825 DF 08        STX     PSH4T2  SAVE THE RETURN ADDRESS
C0577 8827 31           INS             BACK UP THE STACK
C0578 8828 31           INS
C0579 8829 36           PSH A           PUSH EXPONENT ONTO STACK
C0580 882A 96 07        LDA A   PSH4T1+1  LS BYTE OF MANTISSA
C0581 882C 36           PSH A           ONTO THE STACK
C0582 882D 96 06        LDA A   PSH4T1  MS BYTE OF MANTISSA ONTO THE STACK
C0583 882F 36           PSH A
C0584 8830 30           TSX             INDEX TO TOP OF STACK
C0585 8831 DF 06        STX     PSH4T1  TRANSFER POINTER TO MEMORY
C0586 8833 96 07        LDA A   PSH4T1+1  GET LS BYTE OF POINTER
C0587 8835 36           PSH A           PUSH ONTO STACK
C0588 8836 DE 08        LDX     PSH4T2  GET THE RETURN ADDRESS
C0589 8838 6E 00        JMP     0,X     RETURN TO CALL
```

```
C0591                    *
C0592                    *
C0593                    *       FLOATING POINT COMPARE
C0594                    *
C0595                    *       THIS PROGRAM IS NOT REENTRANT.  TO MAKE IT REENTRANT,
C0596                    *       INTERRUPT SYSTEM MUST BE DISABLED WHEN THE MEMORY ADI
C0597                    *       PSH4T1 IS BEING USED.
C0598                    *
C0599 883A 36            FPPCMP PSH A            COPY FIRST ARGUMENT ONTO STACK
C0600 883B 4F                   CLR A
C0601 883C 36                   PSH A
C0602 883D 30                   TSX
C0603 883E EE 00                LDX     0,X
C0604 8840 A6 02                LDA A   2,X
C0605 8842 36                   PSH A
C0606 8843 A6 01                LDA A   1,X
C0607 8845 36                   PSH A
C0608 8846 A6 00                LDA A   0,X
C0609 8848 36                   PSH A
C0610 8849 30                   TSX
C0611 884A DF 06                STX     PSH4T1  INDEX TO SCRATCH MEMORY
C0612 884C 96 07                LDA A   PSH4T1+1   GET 8 BIT ADDRESS BACK TO A REG
C0613                    *      8 REG SET UP BY CALLING PROGRAM, AND NOT DISTURBED
C0614 884E BD 887D              JSR     FPPSUB  DESTRUCTIVE SUBTRACT
C0615 8851 30                   TSX
C0616 8852 A6 00                LDA A   0,X     MSB OF MANTISSA TO A REG
C0617 8854 31                   INS             CLEAN UP STACK
C0618 8855 31                   INS
C0619 8856 31                   INS
C0620 8857 31                   INS
C0621 8858 31                   INS
C0622 8859 4D                   TST A           SET STATUS BITS FOR RESULT
C0623 885A 39                   RTS             RETURN
```

```
00625                    *
00626                    *
00627                    *       COPY FLOATING POINT WORD
00628                    *
00629                    *
00630 8858 37    FPPCPY PSH B           SOURCE ADDRESS ONTO STACK
00631 885C 5F          CLR B
00632 885D 37          PSH B
00633 885E 36          PSH A           PUSH SINK ADDRESS ONTO STACK
00634 885F 37          PSH B
00635 8860 30          TSX
00636 8861 EE 02       LDX    2,X      SOURCE ADDR INTO X
00637 8863 A6 00       LDA A  0,X      MANTIXSA INTO A AND B REG
00638 8865 E6 01       LDA B  1,X
00639 8867 30          TSX             NOW, TRANSMIT MANTISSA
00640 8868 EE 00       LDX    0,X
00641 886A A7 00       STA A  0,X
00642 886C E7 01       STA B  1,X
00643 886E 30          TSX             NOW GET EXPONENT.
00644 886F EE 02       LDX    2,X
00645 8871 A6 02       LDA A  2,X
00646 8873 30          TSX
00647 8874 EE 00       LDX    0,X
00648 8876 A7 02       STA A  2,X
00649 8878 31          INS
00650 8879 31          INS
00651 887A 31          INS
00652 887B 31          INS
00653 887C 39          RTS             RETURN TO CALL
```

```
00655                    *
00656                    *       FLOATING POINT NUMBER CONSISTS OF 3 CONSECUTIVE BYTE
00657                    *         BYTE 0 - MS BYTE OF 16 BIT 2'S COMPLEMENT MANTISSA
00658                    *         BYTE 1 - LS BYTE OF MANTISSA
00659                    *         BYTE 2 - 8-BIT 2'S COMPLEMENT BINARY EXPONENT
00660                    *
00661                    *
00662                    *
00663                    *       FLOATING POINT SUBTRACT
00664                    *
00665                    *
00666 887D BD 8A60 FPPSUB JSR     FPPASU    SET UP ARGS
00667 8880 30              TSX               SET INDEX TO TOP OF STACK
00668 8881 08              INX               SET INDEX TO ARG2
00669 8882 08              INX
00670 8883 08              INX
00671 8884 08              INX
00672 8885 08              INX
00673 8886 08              INX
00674 8887 BD 888C         JSR     FPPNE1    NEGATE ARGUMENT 2
00675 888A 20 4C           BRA     FPPADC
00676                    *
00677                    *
00678                    *       BASIC NEGATION ROUTINE
00679                    *
00680                    *
00681 888C 36        FPPNE1 PSH A             SAVE A REG
00682 888D 37               PSH B             SAVE B REG
00683 888E A6 00            LDA A   0,X       MANTISSA TO A AND B REG'X
00684 8890 E6 01            LDA B   1,X
00685 8892 43               COM A             TAKE TWO'S COMPLEMENT
00686 8893 50               NEG B
00687 8894 25 01            BCS     FPPNE2
00688 8896 4C               INC A
00689 8897 81 80     FPPNE2 CMP A   #$80      CHECK FOR SPECIAL CASE $8000
00690 8899 26 09            BNE     FPPNE3
00691 889B C1 00            CMP B   #$00
00692 889D 26 05            BNE     FPPNE3
00693 889F 0C               CLC               SPECIAL CASE FOUND, CLEAR CARRY
00694 88A0 46               ROR A             SHIFT MANTISSA RIGHT
00695 88A1 56               ROR B
00696 88A2 6C 02            INC     2,X       INCREMENT EXPONENT
00697 88A4 A7 00     FPPNE3 STA A   0,X       STORE RESULTS
00698 88A6 E7 01            STA B   1,X
00699 88A8 33               PUL B             RESTORE B REG
00700 88A9 32               PUL A             RESTORE A REG
00701 88AA 39               RTS               RETURN
```

```
00703                    *
00704                    *
00705                    *       SUBROUTINE FOR ROUNDING, AND FOR OVERFLOW NORMALIZATI
00706                    *
00707                    *
00708                    *       ON ENTRY, A NON-ZERO B REG MEANS ROUNDING IS REQUIRED
00709                    *
00710 88AB 5D           FPPRND  TST B           IS ROUNDING REQUIRED
00711 88AC 27 26                 BEQ     FPPRN9  NO, SO RETURN
00712 88AE 60 00         FPPRN1  TST     0,X     YES, IS NUMBER POSIBIVE OR NEGATIVE
00713 88B0 2A 0D                 BPL     FPPRN2  NUMBER IS POSITIVE OR ZERO
00714 88B2 6D 01                 TST     1,X     NUMBER IS NEGATIVE  IS LSB ZERO
00715 88B4 27 03                 BEQ     FPPRN3  YES
00716 88B6 6A 01                 DEC     1,X     NO (NO BORROW GENERATED)
00717 88B8 39                    RTS             RETURN
00718 88B9 6A 01         FPPRN3  DEC     1,X     BORROW WILL BE GENERATED
00719 88BB 6A 00                 DEC     0,X     SO DECREMENT MSB ALSO
00720 88BD 20 06                 BRA     FPPRN4  GO CHECK FOR OVERFLOW
00721 88BF 6C 01         FPPRN2  INC     1,X     NUMBER IS POSITIVE SO ROUND UP
00722 88C1 26 11                 BNE     FPPRN9  GET OUT SINCE THERE IS NO CARRY
00723 88C3 6C 00                 INC     0,X     CARRY SET, SO INCRMT MSB
00724 88C5 28 0D         FPPRN4  BVC     FPPRN9  CHK FO  OVFLW, GET OUT IF CLEAR
00725                    *
00726                    *       OVERFLOW HANDLING ENTRY POINT
00727                    *
00728 88C7 6C 02         FPPOVF  INC     2,X     INCREMENT EXPONENT
00729 88C9 6D 00                 TST     0,X     POS OR NEG OVFLW, CARRY=0
00730 88CB 2B 01                 BMI     FPPRN5  BRANCH FOR POSITIVE OVERFLOW
00731 88CD 0D                    SEC             NEGATIVE OVERFLOW SO SET CARRY
00732 88CE 66 00         FPPRN5  ROR     0,X     ROTATE RIGHT MSB OF MANTISSA
00733 88D0 66 01                 ROR     1,X     ROTATE RIGHT LSB OF MANTISSA
00734 88D2 25 DA                 BCS     FPPRN1  SEE IF FURTHER ROUNDING REQUIRED
00735 88D4 39            FPPRN9  RTS             RETURN TO CALL
```

```
00737                    *
00738                    *
00739                    *       FLOATING POINT ADDITION
00740                    *
00741                    *
00742 88D5 BD 8A60 FPPADD JSR     FPPASU   SETUP ARGUMENTS ON STACK
00743 88D8 30      FPPADC TSX              SET INDEX TO TOP OF STACK
00744 88D9 6D 06          TST     FPPAR2,X   IS ARG2 ZERO
00745 88DB 27 67          BEQ     FPPMOC   YES, RETURN ARG1
00746 88DD 6D 0B          TST     FPPAR1,X   NO, IS ARG1 ZERO
00747 88DF 27 3F          BEQ     FPPRT2   YES, RETURN ARG2
00748 88E1 A6 0D          LDA A   FPPAR1+2,X   FORM DIFFERENCE OF EXPONENTS
00749 88E3 A0 08          SUB A   FPPAR2+2,X
00750 88E5 A7 02          STA A   FPPEXD,X   PUT IT ON THE STACK
00751 88E7 81 0F          CMP A   #15      IS DIFF GT 15
00752 88E9 2E 59          BGT     FPPMOC   YES, RETURN ARG1
00753 88EB 81 F1          CMP A   #0-15    NO, IS DIFF LT -15
00754 88ED 2D 31          BLT     FPPRT2   YES, RETURN ARG2
00755 88EF 4D             TST A            NO, IS ARG2 GT ARG1
00756 88F0 2E 08          BGT     FPPAD1   SET INDEX TO ARG2 ON STACK
00757 88F2 27 0E          BEQ     FPPAD2   EXPONENTS ARE EQUAL
00758 88F4 08             INX
00759 88F5 08             INX
00760 88F6 08             INX
00761 88F7 08             INX
00762 88F8 08             INX
00763 88F9 40             NEG A
00764 88FA 08      FPPAD1 INX              ADJUST INDEX TO ARGUMENT TO BE SHIFT
00765 88FB 08             INX
00766 88FC 08             INX
00767 88FD 08             INX
00768 88FE 08             INX
00769 88FF 08             INX
00770 8900 20 2C          BRA     FPPSFT   SHIFT SMALLER ARGUMENT
00771 8902 30      FPPAD2 TSX              RESTORE INDEX TO TOP OF STACK
00772                    *
00773                    *       ADDITION OF MANTISSAS
00774                    *
00775 8903 A6 0D          LDA A   FPPAR1+2,X
00776 8905 A7 05          STA A   FPPRES+2,X   STORE THE EXPONENT
00777 8907 A6 0C          LDA A   FPPAR1+1,X
00778 8909 AB 07          ADD A   FPPAR2+1,X
00779 890B A7 04          STA A   FPPRES+1,X   STORE LSB OF MANTISSA
00780 890D E6 0B          LDA B   FPPAR1,X   ADD MSB OF MANTISSAS, USE B REG
00781 890F E9 06          ADC B   FPPAR2,X
00782 8911 07             TPA              SAVE V IN TH A REG
00783 8912 E7 03          STA B   FPPRES,X   STORE THE RESULT
00784 8914 06             TAP              RESTORE THE V BIT
00785 8915 28 2F          BVC     FPPNOR   NORMALIZE IF NO OVERFLOW
00786 8917 08             INX              SET INDEX TO RESULT
00787 8918 08             INX
00788 8919 08             INX
00789 891A BD 88C7        JSR     FPPOVF   GO TO OVERFLOW-ROUNDING ROUTINE
00790 891D 30             TSX              RESTORE INDEX REG
```

```
C0791 891E 20 26          BRA    FPPNOR   GO TO NORMALIZATION ROUTINE
C0792                     *
C0793                     *      RETURN ARGUMENT 2 AS RESULT
C0794                     *
C0795 8920 A6 06    FPPRT2 LDA A  FPPAR2,X  MOVE 3 BYTES TO FPPRES ON STACK
C0796 8922 A7 03          STA A  FPPRES,X
C0797 8924 A6 07          LDA A  FPPAR2+1,X
C0798 8926 A7 04          STA A  FPPRES+1,X
C0799 8928 A6 08          LDA A  FPPAR2+2,X
C080C 892A A7 05          STA A  FPPRES+2,X
C080C 892C 20 34          BRA    FPPMOV   MOVE RESULT TO ARG1 MEMORY LOCATION
C0802                     *
C0803                     *      ROUTINE TO SHIFT MANTISSA RIGHT AND ADJUST EXPONENT
C0804                     *
C0805                     *      A REGISTER CONTAINS SHIFT COUNT, AND INDEX POINTS TO
C0806                     *        TO BE ADJUSTED
C0807                     *
C0808 892E 5F       FPPSFT CLR B           SET B TO INDICATE NO ROUNDING
C0809 892F 4D       FPPSF1 TST A           SEE IF FURTHER SHIFTING REQUIRED
C0810 8930 27 0D          BEQ    FPPSF2   NONE REQUIRED, SO GO TO ROUNDING SECT
C0811 8932 4A             DEC A           DECREMENT COUNT
C0812 8933 6C 02          INC    2,X      INCREMENT EXPONENT
C0813 8935 67 00          ASR    0,X      ARITH SHIFT RIGH MSB OF MANTISSA
C0814 8937 66 01          ROR    1,X      ROTATE RIGHT LSB OF MANTISSA
C0815 8939 24 F3          BCC    FPPSFT   SET B REG ACCORDING TO STATE OF CARRY
C0816 893B C6 01          LDA B  #1
C0817 893D 20 F0          BRA    FPPSF1   CONTINUE LOOP
C0818 893F BD 88AB  FPPSF2 JSR    FPPRND   SEE IF ROUNDING REQUIRED
C0819 8942 20 BE          BRA    FPPAD2   GET OUT OF THIS ROUTINE
C0820                     *
C0821 8944 20 2D    FPPM00 BRA    FPPM01   DUMMY TO EXTEND BRANCHING RANGE
C0822                     *
```

```
C0824                    *
C0825                    *       NORMALIZATION ROUTINE (SHIFT MANTISSA LEFT UNTIL TWO
C0826                    *         BITS ARE DIFFERENT.  DO NOT NORMALIZE ZERO RESULT)
C0827                    *
C0828 8946 6D 03    FPPN0R TST     FPPRES,X   IS MSB OF MANTISSA ZERO
C0829 8948 26 04           BNE     FPPN01     NO, SO NORMALIZE
C0830 894A 6D 04           TST     FPPRES+1,X   IS LSB OF MANTISSA ZERO
C0831 894C 27 14           BEQ     FPPM0V     YES, DO NOT NORMALIZE
C0832                    *
C0833                    *       START OF REGULAR NORMALIZATION
C0834                    *
C0835 894E A6 03    FPPN01 LDA A   FPPRES,X   IS SIGN BIT CLEAR
C0836 8950 2C 05           BGE     FPPN02     YES, CHECK FOR BIT 6 SET
C0837 8952 49              ROL A              NO, IS BIT 6 CLEAR
C0838 8953 2B 05           BMI     FPPN03     NO, GO TO SHIFT LEFT ROUTINE
C0839 8955 20 08           BRA     FPPM0V     YES, NORMALIZATION COMPLETE
C0840 8957 49       FPPN02 ROL A              SIGN BIT CLEAR, IS BIT 6 SET
C0841 8958 2B C8           BMI     FPPM0V     YES, NORMALIZATION COMPLETE
C0842                    *
C0843                    *       ROUTINE TO SHIFT RESULT LEFT ONE PLACE
C0844                    *
C0845 895A 68 04    FPPN03 ASL     FPPRES+1,X
C0846 895C 69 03           ROL     FPPRES,X   ROTATE LEFT MSB OF MANTISSA
C0847 895E 6A 05           DEC     FPPRES+2,X   DECREMENT EXPONENT
C0848 8960 20 EC           BRA     FPPN01     BACK TO START OF LOOP
C0849                    *
C0850                    *
C0851                    *       MOVE RESULT FROM STACK INTO MEMORY
C0852                    *
C0853 8962 A6 03    FPPM0V LDA A   FPPRES,X
C0854 8964 E6 04           LDA B   FPPRES+1,X
C0855 8966 EE 0E           LDX     FPPPA1,X
C0856 8968 A7 00           STA A   0,X
C0857 896A E7 01           STA B   1,X
C0858 896C 30              TSX
C0859 896D A6 05           LDA A   FPPRES+2,X
C0860 896F EE 0E           LDX     FPPPA1,X
C0861 8971 A7 02           STA A   2,X
C0862                    *
C0863                    *       CLEAN UP STACK
C0864                    *
C0865 8973 86 12 x  FPPM01 LDA A   #FPPRTN+2
C0866 8975 31       FPPM02 INS                 INCREMENT STACK POINTER
C0867 8976 4A              DEC A               DECREMENT COUNT
C0868 8977 2E FC           BGT     FPPM02     LOOP TEST
C0869 8979 39              RTS                 RETURN FROM SUBROUTINE
```

```
C0871                    *
00872                    *
C0873                    *       FL6ATING P6INT MULTIPLY
C0874                    *
C0875                    *
C0876 897A BD 8A60 FPPMUL JSR     FPPASU.   SET UP ARGUMENTS 6N STACK
C0877 897D 30              TSX               SET INDEX T6 T6P 6F STACK
C0878 897E BD 8A44         JSR     FPPSMD,   PR6CESS ARGUMENTS
C0879 8981 30              TSX               SET INDEX T6 T6P 6F STACK
C0880 8982 E7 02           STA B   FPPSGN,X   ST6RE THE SIGN FLAG
C0881 8984 A6 0D           LDA A   FPPAR1+2,X   ADD EXP6NENTS AND
C0882 8986 AB 08           ADD A   FPPAR2+2,X   ST6RE IN THE
C0883 8988 4A              DEC A             ADJST EXP6NENT T6 ACC6UNT F6R SHIFT W
C0884 8989 A7 01           STA A   FPPCRY,X   SAVE THE RESULT EXP6NENT 6N STACK
C0885                    *
00886                    *       MULTIPLY MANTISSAS
C0887                    *
C0888 898B 68 07           ASL     FPPAR2+1,X
C0889 898D 69 06           R6L     FPPAR2,X   R6TATE ARG2 T6 SAVE A BIT 6F SIGNIF
C0890 898F 6F 0D           CLR     FPPAR1+2,X   CLEAR 3RD BYTE 6F ARG1
C0891 8991 6F 03           CLR     FPPRES,X   CLEAR RESULT L6CATI6N(24 BIT ACCUMU
00892 8993 6F 04           CLR     FPPRES+1,X
C0893 8995 6F 05           CLR     FPPRES+2,X
C0894 8997 86 0F           LDA A   #15        INITIALIZE THE
C0895 8999 A7 00           STA A   FPPCNT,X   L66P C6UNT
00896 899B 68 07   FPPMU1  ASL     FPPAR2+1,X   SHIFT ARG2 MANTISSA LEFT
C0897 899D 69 06           R6L     FPPAR2,X   IS CARRY SET
C0898 899F 24 12           BCC     FPPMU2   N6, SKIP ADD 6PERATI6N
C0899 89A1 A6 05           LDA A   FPPRES+2,X
00900 89A3 AB 0D           ADD A   FPPAR1+2,X
C0901 89A5 A7 05           STA A   FPPRES+2,X
C0902 89A7 A6 04           LDA A   FPPRES+1,X
C0903 89A9 A9 0C           ADC A   FPPAR1+1,X
C0904 89AB A7 04           STA A   FPPRES+1,X
C0905 89AD A6 03           LDA A   FPPRES,X
C0906 89AF A9 08           ADC A   FPPAR1,X
00907 89B1 A7 03           STA A   FPPRES,X
C0908 89B3 64 08   FPPMU2  LSR     FPPAR1,X   SHIFT ARG 1 MANTISSA RIGHT
00909 89B5 66 0C           R6R     FPPAR1+1,X
C0910 89B7 66 0D           R6R     FPPAR1+2,X
C0911 89B9 6A 00           DEC     FPPCNT,X   DECREMENT L66P C6UNT
C0912 89BB 6D 00           TST     FPPCNT,X   IS L66P FINISHED
C0913 89BD 2E DC           BGT     FPPMU1   N6, G6 BACK T6 START 6F L66P
C0914 89BF 6D 03           TST     FPPRES,X   IS SIGN BIT SET
C0915 89C1 2F 0A           BLE     FPPMU8   YES, 6R P6SSIBLY ZER6 RESULT
C0916 89C3 6A 01   FPPMU3  DEC     FPPCRY,X   DECREMENT EXP6NENT
C0917 89C5 68 05           ASL     FPPRES+2,X   SHIFT RESULT MANTISSA LEFT
C0918 89C7 69 04           R6L     FPPRES+1,X
C0919 89C9 69 03           R6L     FPPRES,X
C0920 89CB 2A F6           BPL     FPPMU3   C6NTINUE UNTIL SIGN BIT SET
C0921 89CD A6 01   FPPMU8  LDA A   FPPCRY,X   M6VE EXP6NENT
C0922 89CF A7 05           STA A   FPPRES+2,X
C0923 89D1 6D 03   FPPMU6  TST     FPPRES,X   CHECK F6R ZER6 RESULT
C0924 89D3 2A 07           BPL     FPPMU7   SKIP 6VERFL6W 6PERATI6N F6R ZER6 RES
```

```
C0925 89D5 08              INX             ADJUST FOR OVFLW, AND ROUND RESULT
C0926 89D6 08              INX             SET INDEX TO FPPRES
C0927 89D7 08              INX
C0928 89D8 BD 88C7         JSR     FPPOVF  CALL OVERFLOW ROUTINE
C0929 89DB 30              TSX
C0930 89DC 6D 02   FPPMU7  TST     FPPSGN,X   IS THE RESULT NEGATIVE
C0931 89DE 27 07           BEQ     FPPMU4  NO,SKIP COMPLEMENTING OPERATION
C0932 89E0 08              INX             YES, NEGATE RESULT
C0933 89E1 08              INX
C0934 89E2 08              INX             INDEX TO FPPRES
C0935 89E3 BD 888C         JSR     FPPNE1  NEGATION ROUTINE
C0936 89E6 30              TSX             RESTORE INDEX
C0937 89E7 7E 8946 FPPMU4  JMP     FPPNOR  GO TO NORMALIZING ROUTINE
```

```
C0939                    *
C0940                    *
C0941                    *       FLOATING POINT DIVIDE
C0942                    *
C0943                    *
C0944 89EA BD 8A60 FPPDIV JSR     FPPASU    SET UP ARGS ON STACK
C0945 89ED 30              TSX              INDEX TO TOP OF STACK
C0946 89EE 6D 06           TST     FPPAR2,X   IS DIVISOR ZERO
C0947 89F0 27 45           BEQ     FPPZDV    YES, PROCESS SPECIAL CASE
C0948 89F2 BD 8A44         JSR     FPPSMD    PROCESS ARGS 1 & 2
C0949 89F5 30              TSX              SET INDEX TO TOP OF STACK
C0950 89F6 E7 02           STA B   FPPSGN,X   STORE THE SIGN FLAG
C0951 89F8 A6 0D           LDA A   FPPAR1+2,X   FORM DIFFERENCE OF EXPONENTS
C0952 89FA A0 08           SUB A   FPPAR2+2,X
C0953 89FC A7 05           STA A   FPPRES+2,X   PUT EXPONENT AWAY
C0954 89FE 86 10           LDA A   #16       INITIALIZE THE
C0955 8A00 A7 00           STA A   FPPCNT,X   LOOP COUNTER
C0956                    *
C0957                    *       LOGIC TO MAKE SURE 16 BITS OF QUOTIENT ARE COMPUTED
C0958                    *
C0959 8A02 A6 09           LDA A   FPPAR1,X   COMPUTE DIFFERENCE FOUND FIRST TIM
C0960 8A04 E6 0C           LDA B   FPPAR1+1,X
C0961 8A06 E0 07           SUB B   FPPAR2+1,X
 0962 8A08 A2 06           SBC A   FPPAR2,X
C0963 8A0A 4D              TST A              IS REMAINDER NEGATIVE
C0964 8A0B 2A 11           BPL     FPPDI4    NO, LEADING BIT IS 1
C0965 8A0D 68 0C           ASL     FPPAR1+1,X   YES, LEADING BIT IS ZERO
C0966 8A0F 69 0B           ROL     FPPAR1,X   SO SHIFT NUMERATOR MANTISSA LEFT O
C0967 8A11 6A 05           DEC     FPPRES+2,X   DECREMENT EXPONENT OF RESULT
C0968                    *
C0969                    *       MAIN DIVISION LOOP
C0970                    *
C0971 8A13 A6 0B   FPPDI1 LDA A   FPPAR1,X   LOAD REMAINDER
C0972 8A15 E6 0C           LDA B   FPPAR1+1,X   WHICH STARTS AS THE NUMERATOR
C0973 8A17 E0 07           SUB B   FPPAR2+1,X   SUBTRACT DIVISOR
C0974 8A19 A2 06           SBC A   FPPAR2,X
C0975 8A1B 4D              TST A              IS REMAINDER NEGATIVE  (CARRY=C)
C0976 8A1C 2D 01           BLT     FPPDI2    YES, CARRY IS CLEAR, SKIP
C0977 8A1E 0D      FPPDI4 SEC                NO, SET CARRY
C0978 8A1F 69 04   FPPDI2 ROL     FPPRES+1,X   SHIFT CARRY BIT
C0979 8A21 69 03           ROL     FPPRES,X   INTO THE QUOTIENT
C0980 8A23 4D              TST A              IS REMAINDER NEGATIVE
C0981 8A24 2D 04           BLT     FPPDI3    YES, GO SHIFT OLD REM LEFT
C0982 8A26 A7 0B           STA A   FPPAR1,X   NO, SO STORE NEW REMAINDER
C0983 8A28 E7 0C           STA B   FPPAR1+1,X
C0984 8A2A 68 0C   FPPDI3 ASL     FPPAR1+1,X   SHIFT OLD REMAINDER LEFT
C0985 8A2C 69 0B           ROL     FPPAR1,X
C0986 8A2E 6A 00           DEC     FPPCNT,X   LOOP TEST
 0987 8A30 6D 00           TST     FPPCNT,X
C0988 8A32 2E DF           BGT     FPPDI1    CONTINUE LOOPING
C0989 8A34 7E 89D1         JMP     FPPMU6
C0990                    *
C0991 8A37 86 7F   FPPZDV LDA A   #127      ZERO DIVISION, SET Q TO MAX
C0992 8A39 A7 03           STA A   FPPRES,X   POSITIVE VALUE
```

```
C0993 8A3B A7 05          STA A  FPPRES+2,X
C0994 8A3D 86 FF          LDA A  #$FF
C0995 8A3F A7 04          STA A  FPPRES+1,X
C0996 8A41 7E 8962        JMP    FPPMOV
C0998                *
C0999                *    PROCESS TWO ARGUMENTS
C1000                *
C1001 8A44 5F       FPPSMD CLR B             CLEAR THE SIGN INDICATOR
C1002 8A45 08              INX               SET INDEX TO ARG 2
C1003 8A46 08              INX
C1004 8A47 08              INX
C1005 8A48 08              INX
C1006 8A49 08              INX
C1007 8A4A 08              INX
C1008 8A4B BD 8A57         JSR    FPPPAR     PROCESS ARGUMENT 2
C1009 8A4E 08              INX               SET INDEX TO ARG1
C1010 8A4F 08              INX
C1011 8A50 08              INX
C1012 8A51 08              INX
C1013 8A52 08              INX
C1014 8A53 BD 8A57         JSR    FPPPAR     PROCESS ARGUMENT 1
C1015 8A56 39              RTS               RETURN
C1016                *
C1017                *    SUBROUTINE TO PROCESS MUL ARGS
C1018                *
C1019                *    INDEX POINTS TO FP ARGUMENT
C1020                *
C1021 8A57 6D 00    FPPPAR TST    0,X        IS ARG NEGATIVE
C1022 8A59 2C 04           BGE    FPPPAC     NO, SKIP SIGN CHANGE
C1023 8A5B 53              COM B             COMPLEMENT THE RESULT SIGN
C1024 8A5C BD 888C         JSR    FPPNE1     NEGATE ARGUMENT
C1025 8A5F 39       FPPPAO RTS               RETURN
C1026                *
C1027                *    ROUTINE TO SET UP ARGUMENTS ON THE STACK
C1028                *
C1029                *    A REG CONTAINS POINTER TO ARG1
C1030                *    B REG CONTAINS POINTER TO ARG2
C1031                *
C1032                *    THE B REG WILL BE USED TO CONTROL THE TWO PASS
C1033                *
C1034 8A60 36       FPPASU PSH A             LSB OF ADDRESS ONTO STACK
C1035 8A61 4F              CLR A             CLEAR A REGISTER
C1036 8A62 36              PSH A             MSB OF ARG ADDRESS ONTO STACK
C1037 8A63 30              TSX               LOAD ADDRESS OF ARG INTO X
C1038 8A64 EE 00           LDX    0,X
C1039 8A66 A6 02           LDA A  2,X       ARG EXPONENT ONTO STACK
C1040 8A68 36              PSH A
C1041 8A69 A6 01           LDA A  1,X       LSB OF ARG MANTISSA ONTO STACK
C1042 8A6B 36              PSH A
C1043 8A6C A6 00           LDA A  0,X       MSB OF ARG MANTISSA ONTO THE S
C1044 8A6E 36              PSH A
C1045 8A6F 5D              TST B            IS THE B REG CLEAR
C1046 8A70 27 04           BEQ    FPPAS2    YES, EXIT FROM LOOP
C1047 8A72 17              TBA              NO, MOVE BREG TO A REG
```

```
C1C48 8A73 5F              CLR B           CLEAR B REG T8 INDICATE L88P TERMI
C1C49 8A74 20 EA           BRA   FPPASU    REPEAT SEQUENCE F8R ARG2
C1C50 8A76 34      FPPAS2  DES   INS?      LEAVE SIX ADDITI8NAL P8SITI8NS 8N
C1C51 8A77 34              DES
C1C52 8A78 34              DES
C1C53 8A79 34              DES
C1C54 8A7A 34              DES
C1C55 8A7B 34              DES
C1C56 8A7C 30              TSX             SET INDEX T8 T8P 8F STACK
C1C57 8A7D EE 10           LDX   FPPRTN,X
C1C58 8A7F 6E 00           JMP   0,X       RETURN (RTS CANN8T BE USED HERE)
C1C59                *
C1C60                *     STACK L8CATI8N SYM88LS
C1C61                *
C1C62      0000      FPPCNT EQU   0         L88P C8UNTER
C1C63      0001      FPPCRY EQU   FPPCNT+1    CARRY INDICAT8R
C1C64      0002      FPPSGN EQU   FPPCRY+1    PR8DUCT (Q8TIENT) SIGN
C1C65      0002      FPPEXD EQU   FPPSGN    EXP8NENT DIFFERENCE
C1C66      0003      FPPRES EQU   FPPEXD+1    RESULT 8F 8PERATI8N
C1C67      0006      FPPAR2 EQU   FPPRES+3    ARG 2 (DATA)
C1C68      0009      FPPPA2 EQU   FPPAR2+3    ARG 2 (P8INTER)
C1C69      0008      FPPAR1 EQU   FPPPA2+2    ARG 1 (DATA)
C1C70      000E      FPPPA1 EQU   FPPAR1+3    ARG 1 (P8INTER)
C1C71      0010      FPPRTN EQU   FPPPA1+2    RETURN ADDRESS
C1C72                *
C1C73                *
C1C74                *     FL8AT
C1C75                *
C1C76                *
C1C77 8A81 8D 8A60 FPPFLT JSR   FPPASU    SET UP ARGS
C1C78 8A84 30              TSX             SET INDEX T8 T8P 8F STACK
C1C79 8A85 86 0F           LDA A #15       15 T8 THE A REG
C1C80 8A87 A7 05           STA A FPPRES+2,X   SET THE RESULT EXP8NENT
C1C81 8A89 A6 06           LDA A FPPAR2,X   M8VE MANTISSA
C1C82 8A8B A7 03           STA A FPPRES,X
C1C83 8A8D A6 07           LDA A FPPAR2+1,X
C1C84 8A8F A7 04           STA A FPPRES+1,X
C1C85 8A91 7E 8946         JMP   FPPN8R    G8 T8 N8RMALIZING SECTI8N
C1C86                *
C1C87                *
C1C88                *     IFIX
C1C89                *
C1C90                *
C1C91 8A94 8D 8A60 FPPFIX JSR   FPPASU    SET UP ARGS
C1C92 8A97 30              TSX             SET UP INDEX T8 T8P 8F STACK
C1C93 8A98 A6 08           LDA A FPPAR2+2,X   GET EXP8NENT INT8 A REG
C1C94 8A9A 81 0F   FPPFI1  CMP A #15       C8MPARE EXP8NENT WITH 15
C1C95 8A9C 27 10           BEQ   FPPFI3    EQUAL
C1C96 8A9E 2E 07           BGT   FPPFI2    LESS THAN
C1C97                *     EXP8NENT IS LESS THAN 15 S8 SHIFT RIGHT
C1C98 8AA0 4C              INC A           INCREMENT EXP8NENT
C1C99 8AA1 67 06           ASR   FPPAR2,X   SHIFT MANTISSA RIGHT
C1100 8AA3 66 07           R8R   FPPAR2+1,X
C1101 8AA5 20 F3           BRA   FPPFI1    C8NTINUE L88PING
```

```
C1102                    *       EXPONENT GREATER THAN 15 SO SHIFT LEFT (ERROR)
C1103 8AA7 4A       FPPFI2 DEC A
C1104 8AA8 68 07           ASL     FPPAR2+1,X   SHIFT MANTISSA LEFT
C1105 8AAA 69 06           ROL     FPPAR2,X
C1106 8AAC 20 EC           BRA     FPPFI1   CONTINUE LOOPING
C1107                    *       EXPONENT EQUAL TO 15 SO SHIFT IS COMPLETE
C1108 8AAE A6 06    FPPFI3 LDA A   FPPAR2,X   TRANSFER RESULTS
C1109 8AB0 E6 07           LDA B   FPPAR2+1,X
C110C 8AB2 EE 0E           LDX     FPPPA1,X
C1111 8AB4 A7 00           STA A   0,X
C1112 8AB6 E7 01           STA B   1,X
C1113 8AB8 7E 8973         JMP     FPPM01   GO CLEAN UP STACK
C1114                    *
C1115                    *
C1116                    *       FLOATING POINT NEGATE
C1117                    *
C1118                    *
C1119 8ABB BD 8A60  FPPNEG JSR     FPPASU   SET UP ARGUMENTS
C112C 8ABE 30              TSX              SET INDEX TO TOP OF STACK
C1121 8ABF 08              INX              SET INDEX TO ARG2
C1122 8AC0 08              INX
C1123 8AC1 08              INX
C1124 8AC2 08              INX
C1125 8AC3 08              INX
C1126 8AC4 08              INX
C1127 8AC5 BD 888C         JSR     FPPNE1
C1128 8AC8 30              TSX              INDEX TO TOP OF STACK
C1129 8AC9 7E 8920         JMP     FPPRT2   RETURN ARG2
C1130                    *
C1131                    *
C1132                    *       FLOATING POINT CLEAR
C1133                    *
C1134                    *
C1135 8ACC 36       FPPCLR PSH A             PUT ADDRESS ONTO STACK
C1136 8ACD 4F              CLR A
C1137 8ACE 36              PSH A
C1138 8ACF 30              TSX              INDEX TO TOP OF STACK
C1139 8AD0 EE 00           LDX     0,X      GET ADDRESS INTO INDEX REG
C1140 8AD2 6F 00           CLR     0,X      CLEAR THREE CONSECUTIVE BYTES
C1141 8AD4 6F 01           CLR     1,X
C1142 8AD6 6F 02           CLR     2,X
C1143 8AD8 31              INS              CLEAN UP STACK
C1144 8AD9 31              INS
C1145 8ADA 39              RTS              RETURN
C1146                      END
```

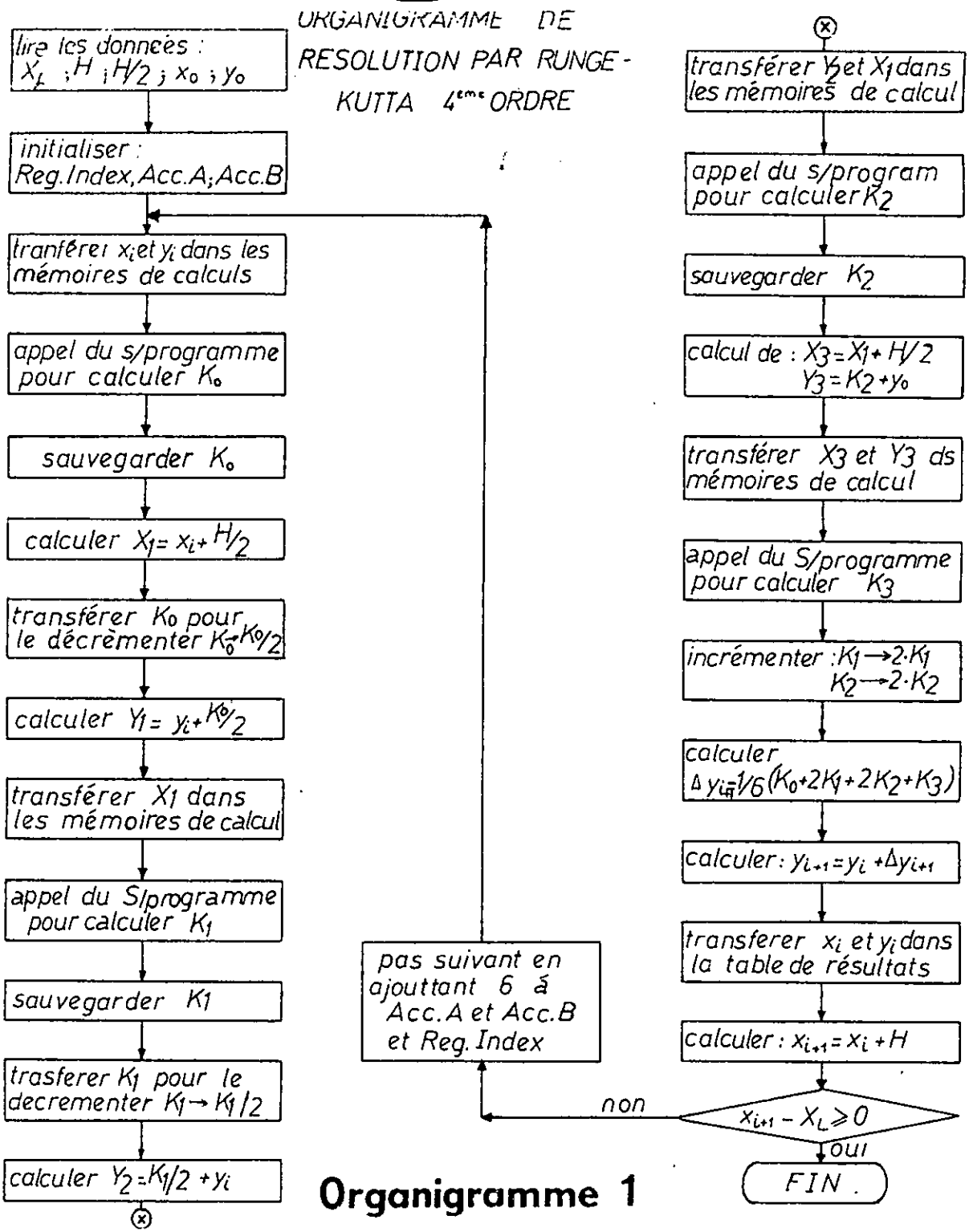
SYMBOL TABLE

A1	0000	A2	0003	PSH4T1	0006	PSH4T2	0008	L1	000A	L2	000B
A3	000C	ROUADD	000F	INCH	FA8B	OUTCH	F90C	LOOP	8503	ADD	853A
SUB	8542	MUL	854A	DIV	8552	FLOAT	855A	IFIX	856B	NEGATE	857E
CLEAR	8586	PRINT	858D	LOOP1	859A	LOOPA	85A1	LOOP2	85A5	LOOPB	85AC
PRIN1	858D	SETUP1	85D2	SETUP2	85E1	SUREG	85FB	READ3B	8600	READ31	8606
READ32	860A	READ33	8619	READ34	8625	READ35	8644	WRIT3B	864B	WRIT31	8652
WRIT32	865E	WRIT35	8667	WRIT33	8673	WRIT34	8675	QUERY	868A	NEWLIN	8693
STFP	869E	STFPA4	0001	STFPA3	0005	STFPA2	0009	STFPA1	000D	STFPSG	0010
STFPDP	0011	STFPRS	0012	STFP1	86BD	STFP2	86CC	STFP3	86D4	STFP4	8707
STFP5	871A	STFP6	872D	FPST	8732	FPSTA5	C001	FPSTA4	0005	FPSTA3	0009
FPSTA2	000D	FPSTA1	0011	FPSTDP	0014	FPSTSD	C015	FPSTVL	0016	FPST1	8767
FPST2	876F	FPST3	8778	FPST4	8791	FPST5	87A7	FPST6	87B0	FPST7	87BD
FPST8	87CA	FPST9	88CC	FPST10	8819	FPST11	881B	PSH4	8820	FPPCMP	883A
FPPCPY	885B	FPPSUB	887D	FPPNE1	888C	FPPNE2	8897	FPPNE3	88A4	FPPRND	88AB
FPPRN1	88AE	FPPRN3	88B9	FPPRN2	88BF	FPPRN4	88C5	FPPOVF	88C7	FPPRN5	88CE
FPPRN9	88D4	FPPADD	88D5	FPPAD0	88D8	FPPAD1	88FA	FPPAD2	8902	FPPRT2	8920
FPPSFT	892E	FPPSF1	892F	FPPSF2	893F	FPPM80	8944	FPPN0R	8946	FPPN01	894E
FPPN02	8957	FPPN03	895A	FPPMOV	8962	FPPM01	8973	FPPM02	8975	FPPMUL	897A
FPPMU1	8998	FPPMU2	89B3	FPPMU3	89C3	FPPMU8	89CD	FPPMU6	89D1	FPPMU7	89DC
PMU4	89E7	FPPDIV	89EA	FPPDI1	8A13	FPPDI4	8A1E	FPPDI2	8A1F	FPPDI3	8A2A
FPPZDV	8A37	FPPSMD	8A44	FPPPAR	8A57	FPPPA0	8A5F	FPPASU	8A60	FPPAS2	8A76
FPPCNT	0000	FPPCRY	0001	FPPSGN	0002	FPPEXD	C002	FPPRES	0003	FPPAR2	0006
FPPPA2	0009	FPPAR1	0008	FPPPA1	000E	FPPRTN	C010	FPPFLT	8A81	FPPFIX	8A94
FPPFI1	8A9A	FPPFI2	8AA7	FPPFI3	8AAE	FPPNEG	8ABB	FPPCLR	8ACC		

ALPHA SYMBOL TABLE

A1	0000	A2	0003	A3	000C	ADD	853A	CLEAR	8586	DIV	8552
FLOAT	855A	FPPAD0	88D8	FPPAD1	88FA	FPPAD2	8902	FPPADD	88D5	FPPAR1	0008
FPPAR2	0006	FPPAS2	8A76	FPPASU	8A60	FPPCLR	8ACC	FPPCMP	883A	FPPCNT	0000
FPPCPY	885B	FPPCRY	0001	FPPDI1	8A13	FPPDI2	8A1F	FPPDI3	8A2A	FPPDI4	8A1E
FPPDIV	89EA	FPPEXD	0002	FPPFI1	8A9A	FPPFI2	8AA7	FPPFI3	8AAE	FPPFIX	8A94
FPPFLT	8A81	FPPM80	8944	FPPM01	8973	FPPM02	8975	FPPMOV	8962	FPPMU1	8998
FPPMU2	8983	FPPMU3	89C3	FPPMU4	89E7	FPPMU6	89D1	FPPMU7	89DC	FPPMU8	89CD
FPPMUL	897A	FPPNE1	888C	FPPNE2	8897	FPPNE3	88A4	FPPNEG	8ABB	FPPN01	894E
FPPN02	8957	FPPN03	895A	FPPN0R	8946	FPPOVF	88C7	FPPPA0	8A5F	FPPPA1	000E
FPPPA2	0009	FPPPAR	8A57	FPPRES	0003	FPPRN1	88AE	FPPRN2	88BF	FPPRN3	88B9
FPPRN4	88C5	FPPRN5	88CE	FPPRN9	88D4	FPPRND	88AB	FPPRT2	8920	FPPRTN	0010
FPPSF1	892F	FPPSF2	893F	FPPSFT	892E	FPPSGN	C002	FPPSMD	8A44	FPPSUB	887D
FPPZDV	8A37	FPST	8732	FPST1	8767	FPST10	8819	FPST11	881B	FPST2	876F
FPST3	8778	FPST4	8791	FPST5	87A7	FPST6	87B0	FPST7	87BD	FPST8	87CA
FPST9	88CC	FPSTA1	0011	FPSTA2	000D	FPSTA3	0009	FPSTA4	0005	FPSTA5	0001
FPSTDP	0014	FPSTSD	0015	FPSTVL	0016	IFIX	856B	INCH	FA8B	L1	000A
L2	000B	LOOP	8503	LOOP1	859A	LOOP2	85A5	LOOPA	85A1	LOOPB	85AC
MUL	854A	NEGATE	857E	NEWLIN	8693	OUTCH	F90C	PRIN1	858D	PRINT	858D
PSH4	8820	PSH4T1	0006	PSH4T2	0008	QUERY	868A	READ31	8606	READ32	860A
READ33	8619	READ34	8625	READ35	8644	READ3B	8600	ROUADD	000F	SETUP1	85D2

```
SETUP2 85E1  STFP   869E  STFP1  86BD  STFP2  86CC  STFP3  86D4  STFP4  870
STFP5  871A  STFP6  872D  STFPA1 000D  STFPA2 C009  STFPA3 0005  STFPA4 000
STFPDP 0011  STFPRS 0012  STFPSG 0010  SUB    8542  SUREG  85FB  WRIT31 865
WRIT32 865E  WRIT33 8673  WRIT34 8675  WRIT35 8667  WRIT3B 864B
*STOP* 0
FIN
TOTAL JOB TIME=00:10:12
  BEGIN IDLE
```



Organigramme 1

CHAPITRE III

METHODES NUMERIQUES POUR LA RESOLUTION D' EQUATIONS DIFFERENTIELLES

III - A : GENERALITES

Ingénieurs et scientifiques, utilisaient depuis plus de quatres siècles, les méthodes numériques pour résoudre des problèmes très complexes à l'aide des régles à calculer et puis récemment avec des calculateurs analogiques et digitaux.

Ces méthodes numériques sont utilisées pour développer le modèle de simulation qui est une approximation discréte du système réel. Dahs la sélection de la technique numérique, l'utilisateur doit considérer l'exactitude et l'éfficacité de la méthode spécifiée à son application correspondante.

Un simple système discret consiste en une série de pulsations en des temps égaux (T). Considérons f (t) une fonction continue, quand cette fonction est échantillonnée à des intervalles égaux dans le temps t, cette fonction peut être déterminée par la séquence de nombre f(o), f(T), f(2T), ... f(nT).

Cette série de nombres donne une description limitée de la fonction de temps. Car les valeurs de f(t) sont seulement connues au temps $0, T, 2T, \ldots nT$. Les autres valeurs de f(t), à d'autres valeurs du temps sont trouvées par extrapolation ou intrapolation.

Un système à temps discréts est défini comme tout système variable (excitation et réponse) mais seulement à des moments discrets du temps.

A chaque pas de la simulation, la réponse simulée differe normalement de la réponse exacte. L'erreur de troncature et l'erreur d'arrondissement sont introduites durant l'intégration de l'équation différentielle.

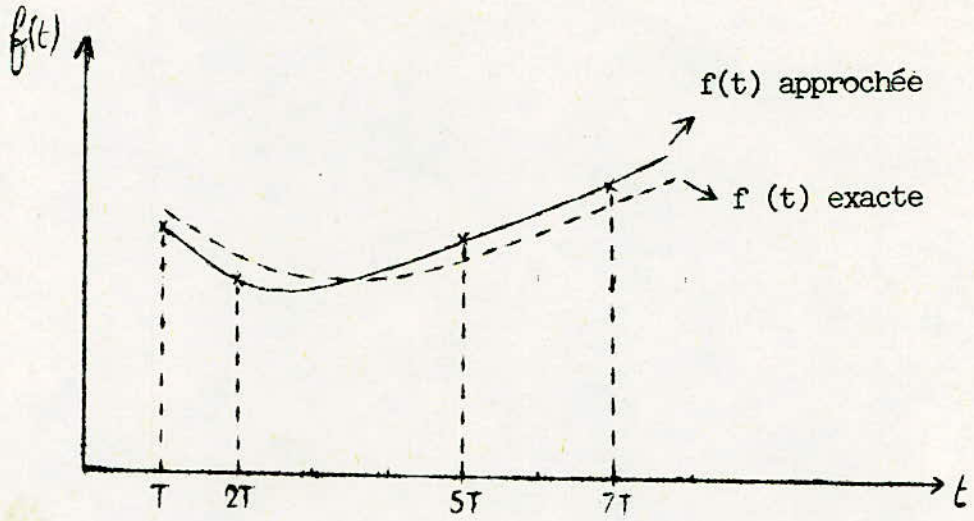
L'erreur de troncature est determinée par la nature des approximations dans la méthode utilisée et les caractéristiques d'équipement

.../...

du calculateur utilisé (nombre de bits, ...)

L'erreur d'arrondissement est due à l'arrondissement du nombre irrationnel.

Ces erreurs vont s'additionner ensemble à chaque pas de la simulation, ceci nous conduit à une réponse approximative de la réponse exacte comme le montre la figure ci-dessous.



III - B : METHODE DE RUNGE - KUTTA 4EME ORDRE

I - ELABORATION DU PROGRAMME DE RESOLUTION

1°) Présentation de la méthode de RUNGE - KUTTA.

Durant des années une large variétés de méthodes numériques ont été développées dans le but de retenir les avantages sur les series de Taylor qui requierent un calcul de hautes derivées. La plus importante et la plus utilisée parmi ces méthodes dans la simulation numérique est celle de RUNGE-KUTTA. Plusieurs ont contribué à ces méthodes numériques, l'approche la plus importante comprise dans toutes celles-ci est de traiter une équation différentielle du type :

$$\frac{dy}{dx} = y'(x) = f(x,y). \qquad (1)$$

avec y = yo et x = xo comme conditions initiales, dans un domaine D où cette fonction est définie et continue.

La solution d'une telle équation est de la forme suivante :

$$y_{i+1} = y_i + \int_{hi}^{hi+1} f(x,y)\,dx = y_i + \int_{x_i}^{xi+1} f(x,y\,dx = y_i + \Delta yi+1 \qquad (2)$$

Supposons que pour un point quelconque du domaine, et pour le pas h de la simulation l'équation peut être développée d'après la formule de taylor :

$$y(x+h) = y(x) + \frac{h}{1!}\,y(x) + \frac{h^2}{2!}\,y''(x) + \frac{h^3}{3!}\,y'''(x) + \ldots + (3)$$

$$y(x+h) = y(x) + \Delta y \text{ ----)} \quad \Delta y = y(x+h) - y(x)$$

$$\text{donc : } \Delta y = \frac{h}{1!}\,y'(x) + \frac{h^2}{2!}\,y''(x) + \frac{h^3}{3!}\,y'''(x) + \ldots \qquad (4)$$

. . ./. . .

Il suffit donc de calculer les différentes dérivées $y';y'';\ \ldots$ les substituer dans l'équation (3) pour trouver la solution de l'équation différentielle. Ce procédé est appelé méthode de RUNGE-KUTTA.

Calcul des dérivées :

$$y' = f(x,y)$$

$$y'' = f'(x,y) = \left(\frac{\delta f}{\delta x} + \frac{\delta f}{\delta y}\cdot\frac{\delta y}{\delta x}\right) = f_x + f_y f$$

$$y''' = \frac{d}{dx}\left(\frac{\delta f}{\delta x} + \frac{\delta f}{\delta y}\cdot\frac{\delta y}{\delta x}\right) = \frac{\delta^2 f}{\delta x^2} + \frac{\delta^2 f}{\delta y\,\delta x}\cdot\frac{\delta y}{\delta x} +$$

$$+ \frac{\delta f}{\delta y}\cdot\frac{\delta f}{\delta x} + \left[\frac{\delta^2 f}{\delta x\delta y} + \frac{\delta^2 f}{\delta y^2}\cdot\frac{\delta y}{\delta x} + \left(\frac{\delta f}{\delta y}\right)^2\right]\frac{\delta y}{\delta x}$$

$$y''' = \left[f_{xx} + F_{yx}.f + f_y.f_x\right] + \left[f_{xy} + f_{yy}.f + f_y^2\right] f.$$

Après substitution de toutes ces dérivées dans l'équation (4), on obtient Δy exprimée en fonction de $f(x,y)$, des dérivées partielles de $f(x,y)$ et des puissances du pas h.

d'où : $$\Delta y = hf + \frac{h^2}{2!}(f_x + f_y f) + \frac{h^3}{3!}(f_{xx} + 2f_{xy}f + f_{yy}f^2 + f_x f_y + f_y f_y f) + \ldots (5)$$

On remarquera, que le but de cette méthode de RUNGE-KUTTA, est une approximation de l'intégrale dans l'équation (2) qui est en toute rigueur de la forme suivante :

$$\int_{x_i}^{x_{i+1}} f(x,y)\,dx = \Delta y_{i+1} = \mu_o k_o + \mu_1 k_1 + \mu_2 k_2 + \ldots$$

$$+ \mu_m k_m \qquad (6)$$

avec : $\mu_o, \mu_1, \mu_2 \ldots \mu_m$: sont les facteurs de poids

.../...

$$ko = h\, f(xo, yo)$$

$$k_1 = h\, f\,(xo + \alpha_0 h;\ yo + \beta_{10}\, ko)$$

$$K_2 = h\, f\,(xo + \alpha_1 h;\ yo + \beta_{20}\, k_1)$$

- - - - - - - - - - - - - - - - - - -

$$km = h\, f\,(xo + \alpha_m h;\ yo + \beta_{mo}\, ko + \ldots + \beta_{mm-1}\, k_{m-1})$$

Pour chacune des expressions ko, k_1 ... km est appliquée le développement des series de Taylor pour les fonctions à 2 variables qui se présente sous cette forme aux points a et b.

$$f(a+u, b+v) = f(a,b) + \left(u\frac{\delta}{\delta x} + v\frac{\delta}{\delta y}\right).f(a,b) + \frac{1}{2!}\left(u\frac{\delta}{\delta x} + v\frac{\delta}{\delta y}\right)^2 f(a,b)$$

$$+ \frac{1}{3!}\left(u\frac{\delta}{\delta x} + v\frac{\delta}{\delta y}\right)^3 . f(a,b) + \ldots \qquad (8)$$

Finalement ko, k_1 ... km sont développées suivant la formule de l'équation (8) et chacune de ces équations est identifiée à l'équation (5). RUNGE - KUTTA est arrivée à un système de 11 équations avec 13 inconnus, qui permettent de trouver ses différentes méthodes 1er Ordre ... qui dépendent du nombre de termes des séries de Taylor.

La méthode du 4ème ordre comprend tous les termes jusqu'à h^4. Plus l'ordre de la méthode est élevé plus l'erreur est réduite à chaque étape; mais encore plus complexe et plus longue. La plus puissante et la plus utilisé en pratique, est la méthode du 4ème Ordre, qui sera le centre de ce modeste travail.

a) méthode du 4ème Ordre de RUNGE - KUTTA :

$$y_{i+1} = y_i + \frac{1}{6}\,(ko + 2k_1 + 2k_2 + k_3)\ .$$

.../...

Avec : $ko = h.f(xo, yo)$

$k1 = h.f\left(xo + \frac{1}{2}h\ ;\ yo + \frac{1}{2}ko\right)$.

$k2 = h.f\left(xo + \frac{1}{2}h ; yo + \frac{1}{2}k1\right)$

$k3 = h.f(xo + h\ ;\ yo + k2)$.

b) Algorithme de cette méthode

$y(xo) = yo$, $x = xo$ et $h = x_{i+1} - x_i$ le pas de la simulation.

1er pas $xo + h = x_1$ -----) $y(xo + h) = y_1$ ----) $y_1 - Y_o = \Delta_{y1}$

2ème pas $x_1 + h = x_2$ ----) $y(x_1 + h) = y_2$ -----) $y_2 - y_1 = \Delta y_2$

- -

i^{eme} pas $x_i + h = x_i$ -----) $y(x_i + h) = y_{i+1}$ ----) $y_{i+1} - y_i = \Delta y_{i+1}$

On commence par calculer ko, k_1, K_2 et K_3, ceci nous permettra le calcul de $\Delta y_1 = \frac{1}{6}(ko + 2k_1 + 2k_2 + k_3)$ qui sera ajoutée à yo, le résultat obtenue est y_1. Cette séquence sera répétée n fois suivant le nombre de pas demandé.

Remarque : Pour le début du calcul, xo et yo sont connues, ko peut être déterminée directement $ko = h.f(xo, yo)$. Le résultat de cette dernière expression est utilisé pour le calcul de k_1 qui nous permettra de calculer k_2 etc ...

C) Estimation du temps de résolution

Pour chaque pas de la simulation, l'estimation du temps sera donnée par la formule suivante :

.../...

$$th = 4\,t_f + 10\alpha + 7\beta$$

Avec tf = temps nécessaire pour le caclul de la fonction f (x,y); ce temps sera multiplié par 4 car nous avons 4 expressions à calculer (k_o, k_1, k_2 et k_3).

¥ α = temps nécessaire pour faire une addition.

¥ β " " " " " multiplication.

2°) Programme de résolution:

Du point de vu construction d'un programme de résolution sur le microprocesseur M6800, la méthode de RUNGE-KUTTA ne présente pas de problèmes spéciaux. La plus grande difficulté se trouve dans le calcul des k_i à chaque étape.

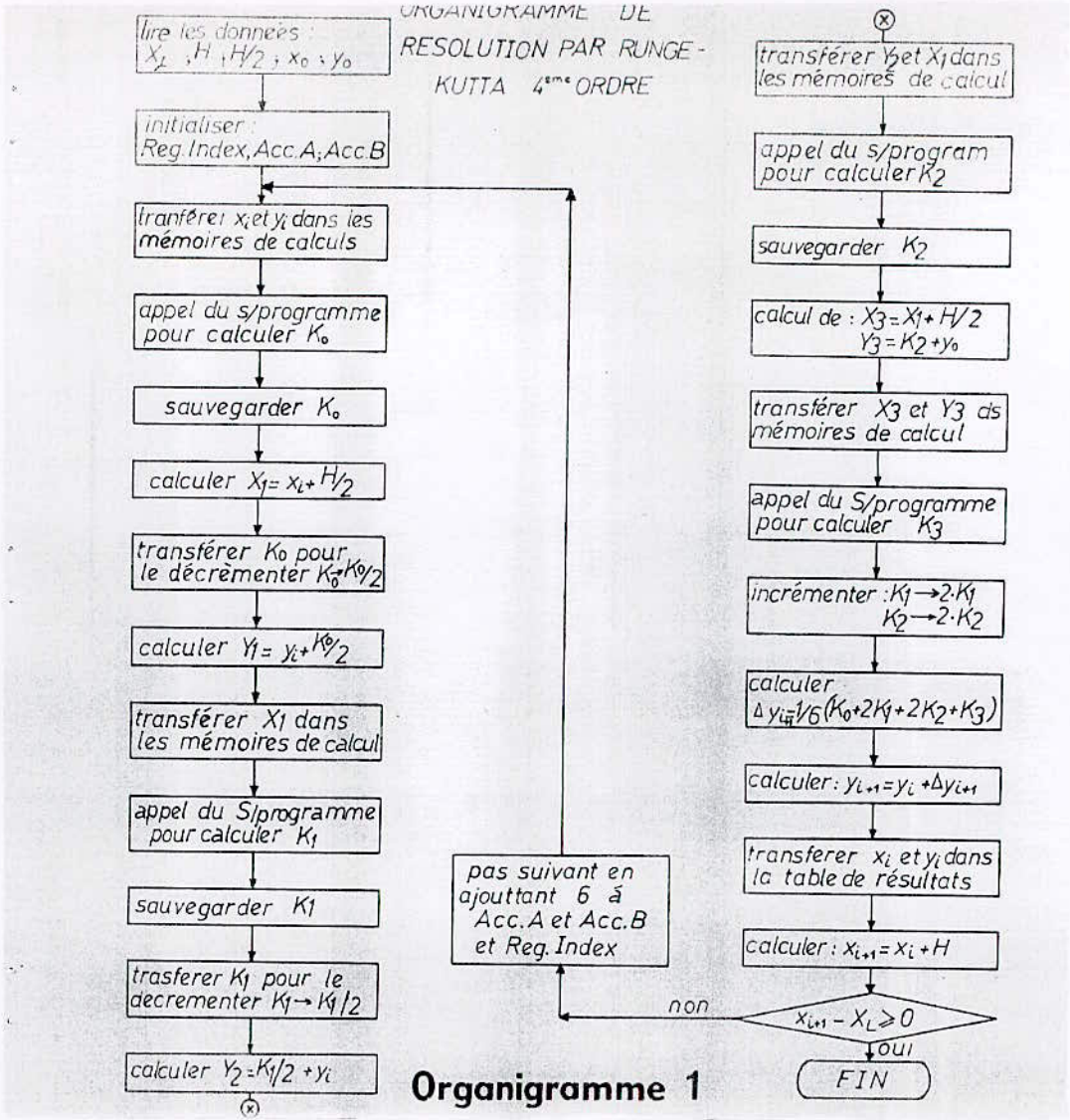
Pour conserver un programme de résolution valable pour une équation différentielle quelconque, on procéde à l'obtention de f(x,y) donnée par x et y comme une soubroutine spéciale (FPP SPK) qui sera appelée par le programme principal de résolution pour le calcul des k_i.

a) Organigramme de résolution (voir organigramme 1)

¥ Conditions initiales, xo,yo, le pas H et la valeur finale X_1 de l'intervale de résolution, sont introduites en mémoire à l'aide du programme de chargement d'une table (page 50).

¥ Pour incrémenter le pas de simulation H, on ajoute le nombre 6 à chacun des accumulateurs A et B. et registre d'index.

¥ Les expressions k_o, k_1, k_2 et k_3 sont



Organigramme 1

calculées par une soubroutine FPP SPK.

¥ y_{i+1} est calculée par substitution des valeurs de k_i obtenues précédemment dans l'équation $y_{i+1} = y_i + \frac{1}{6}(k_o + 2k_1 + 2k_2 + k_3)$

¥ La valeur de x_i est augmentée du pas H, $x_{i+1} = x_i + h$

¥ Les valeurs x_i et y_i d'une étape sont imprimées dans la table des résultats.

¥ Un test est fait si le calcul est complet. Sinon, on reboucle encore une nouvelle fois.

b) Programme principal (page 56)

En fin d'éxécution du programme, les résultats sont stockés dans la table qui est réservé à cette effet (voir page 60). Chaque résultat occupe 3 bytes (2bytes pour la mantisse et 1 byte exposant).

C) Programme de chargement d'une table

Ce programme consiste à charger les valeurs à utiliser dans le programme principal et à réserver des positions mémoires pour sauvegarder les contenus des accumulateurs A et B ainsi que le registre d'index.

¥ Après toutes ces réservations, on introduit les données 2 par 2; X_1, H, H/2, xo et yo. Le procédé est le suivant :

Après le point d'interrogation (?) on écrit la donnée en décimal qui sera directement convertie en virgule flottante est stockée dans la mémoire d'adresse 00 à l'aide de la soubroutine (SETUP2.)

.../...

¥ A l'aide de la soubroutine (FPPCPY), on transfert ce chiffre dans la position mémoire réservée à cet effet.

Programme de chargement d'une table de données :

```
8E  00FF      LDS  #  $FF initialiser le pointeur de pile
86  38        LDA A #  $38 pointer la mémoire de la première donnée
97  16 DEBUT  STA A $16
BD  85E1      JSR SETUP2 introduire les données après le point d'interrogation
                         ?X_L (LF) ? H (LF) ? H/2 (LF) ? xo (LF)? yo(LF)? 0
96  16        LDA A  $16
C6  00        LDA B  #  $ 00 pointer le nombre converti
BD  885B      JSR FPPCPY transfért de 00 à l'adresse pointée par ACC A.
C6  03        LDA B #   $ 03
96  16        LDA A  $ 16
8B  03        ADD A #  $ 03 Ajouter 3 pour pointer la donnée suivante
97  16        STA A  $ 16
BD  885B      JSR FPPCPY Transfert du 2ème nombre converti de 03 vers Adresse
                         pointer par ACC  A
96  16        LDA A  $ 16
8B  03        ADD A #  $ 03 Est ce qu'on pointe la dernière donnée
C6  4A        LDA B #  $ 4A pointe la mémoire de la fin des données
11            CBA          Compare ACC  A à ACC  B
26  E0        BNE DEBUT    Boucler sinon     égaux
3F            SWI          Fin si égaux
```

d) Programme de conversion d'une table de résultats.

¥ On se donne l'adresse du début de la table à convertir (ici 41)

¥ On pointe par ACC A; x_i le chiffre à convertir, puis on appelle la soubroutine de conversion de la virgule flottante à une série de caractères décimaux (FPST).

.../...

¥ On pointe une 2ème fois par l'accumulateur A l'adresse de y_i; à l'aide de (FPST) y_i sera converti en décimal puis écrit juste à côté de x_i correspondant après un espace de 3 blancs.

¥ On appelle la soubroutine NEWLIN qui permet le changement de ligne pour convertir le x_{i+1} et y_{i+1}. Ce processus est répété jusqu'à la fin de la table qui sera déterminée par comparaison de x_i à X_L.

```
                ORG     $ 7000
 8E  00FF       LDS  #  $ FF Initialisation du compteur de pile.
 BD  8693       JSR NEWLIN Nouvelle ligne sur l'écran de visualisation
 86  41         LDA A #  $ 41 Pointer le début de la table
7008 97 16 DEBUT STAA $ 16 Sauvegarder la valeur de ACC.
 BD  8832       JSR FPST  Convertir la valeur x_i en décimal
 C6  03.        LDAB  #  $ 03 Sortie de 3 blancs.
 86  20 BLANC   LDA A #  '
 BD  F9CF       JSR OUTCH
 5A             D EC B
 26  F8         BNE BLANC
 96  16         LDA A  $ 16 Pointer la valeur y_i
 8B  03         ADD A  #  $ 03
 97  16         STA A $ 16
 BD  8832       JSR FPST Convertir la valeur y_i en décimal
 BD  8693       JSR NEWLIN Nouvelle ligne sur écran de visualisation
 96  16         LDA A $ 16 Pointer x_i
 80 03          SUB A #  $ 03
 C6  38         LDA B  #  $ 38 Pointer X_L valeur limité de l'intervale
 BD  883A       JSR FPPCMP Comparer (x_i - X_L) ≤ 0)
 2B  01         BMI PAS  Si négatif
 3F             SWI  si positif ou nul = fin de table
 96  16 PAS     LDAA  $  16
 8B  03         ADDA  #  $ 03 Pointer la valeur suivante des x_i.
 7E  7008       JMP DEBUT
                END.
```

II - EXEMPLES DE RESOLUTION, RESULTATS SUR ECRAN DE VISUALISATION.

Le programme principal de résolution d'équations différentielles est étroitement lié au programme BIBMAT de la bibliothèque mathématiques. Cette dernière constitue la source de toutes les soubroutines utilisées par le programme principal. Il faut ajouter à tout cela, la soubroutine du calcul des k_i (FPPSPK) qui varie d'une fonction à une autre. Tout exemple de résolution se compose de 4 étapes :

* 1ère Etape : Chargement du sous programme de calcul des k_i de la fonction à simuler.

* 2ème étape : A l'aide du programme d'introduction de table (page 50) les valeurs X_L, H, H/2, x_o et y_o et quelques nombre utilisés par les opérations seront stockés dans leurs adresses respectives en virgule flottante.

* 3ème Etape : On appelle le programme principal de résolution; après éxécution, les résultats sont disponibles en virgule flottante dans la table des résultats dont le début est à l'adresse(41)

4ème Etape : Enfin, on appelle le sous-programme de conversion d'une table qui transformera les résultats directement en décimal sous forme de tableau (à chaque x_i correspond y_i).

1°) Exemple 1

$$\frac{dy}{dx} = f(x,y) = \dot{y} = 2x$$

Les données sont :

- valeurs initiales : $x_o = 0$ et $y_o = 0$
- intervale d'étude : $[0;2]$ $X_L = 2$ et $X_I = 0$
- Valeur du pas : $H = \frac{XI - XL}{N}$

  N : Nombre de pas = 20

  $H = \frac{2-0}{20} = 0,1$

a) Organigramme de FPPSPK (Voir organigramme 2)

b) Programme de FPPSPK.

.../...

Algorithme :

$k_i = 2.x_i.h.$

- On calcul xh en virgule flottante

- On incrémente l'exposant pour avoir xh -----) 2 xh

```
ORG $ 8ADC
96 1A    LDA A  $ 1A      Pointer x_i par le contenu de 1 A
C6 3B    LDA B  #  $ 3B   Pointer H le pas de simulation
BD 897A  JSR FPPMUL       Calcul de x_i . H des adresse 35
DE 18    LDX  $ 18        Contenu de 18 dans Reg. Index
6C 00    INC  0,X         Incrémenté l'exposant -----) 2 x_i . H
39       RTS              Retour au programme principal.
```

Remarques : - 1A contient adresse 47 pour le 1er pas

- 1B " " 4A " " " " "

- L'adresse de calcul de FPPSPK s'incrémente de 6 à chaque pas de la simulation.

C) Résultats de la résolution (voir page 61)

2°) Exemple 2

$\frac{dy}{dx} = f(x,y) = \dot{y} = x + y$

Les données sont : - valeurs initiales : xo = 0 et yo = 1

- Intervalle d'étude: $[0, 2]$

- valeur du pas : $H = \frac{2 - 0}{20} = 0,1$

a) Organigramme de FPPSPK (voir organigramme 2)

b) Programme de FPPSPK

* Algorithme : $k_i = H.(x_i + y_i)$

- on calcul $x_i + y_i$ en virgule flottante

- On calcul $H.(x_i + y_i)$.

.../...

```
                ORG     $ 8ADC
96  1A          LDA A   $ 1A Pointer x_i pour l'adresse contenu dans 1A
D6  1B          LDA B   $ 1B "       y_i "      "       "     "  1B
BD  88D5        JSR  FPPADD  calculer x_i + y_i ) des adresse  35
96  1A          LDA A   $ 1A Pointer ce résultat
C6  3B          LDA B   #  $ 3B Pointer H le pas
BD  897A        JSR  FPPMUL Calculer H.(x_i + y_i) dans 35
39              RTS          Retour au programme principal.
```

C) Résultats de la résolution (Voir page 61)

3°) Exemple 3.

$$\frac{dy}{dx} = f(x,y) = \dot{y} = y - \frac{2x}{y}$$

Les données sont : -valeurs initiales : $x_o = 0$; $y_o = 1$

- intervalle d'étude : $[0;2]$

- Valeur du pas
  N = 20

---) $H = \frac{2-0}{20} = 0,1$

a) Organigramme de FPPSPK (Voir organigramme 2)

b) Programme de FPPSPK

Algorithme : $k_i = (y_i - \frac{2xi}{yi}). H$

- On incrémente l'exposant de x_i ---) pour avoir $2x_i$
- On calcul ensuite $2x_i/y_i$
- Puis on calcule la différence $(y_i - 2x_i/y_i.)$
- Finalement on trouve k_i en multipliant la différence par H.

.../...

```
DE  18      LDX  $ 18   Pointer l'exposant de x_i dans adresse 37.
6C  00      INC  0,X    Exposant de x_i ----) 2x_i
96  1A      LDA A  $ 1A Pointer 2.x_i
D6  1B      LDAB  $ 1B  Pointer y_i
BD  89EA    JSR  FPPDIV Calculer 2x_i/y_i
96  1B      LDA A  $ 1B Pointer y_o
D6  1A      LDA  B $ 1A Pointer 2x_i/y_i
BD  887D    JSR  FPPSUB Calculer la différence y_o-2x_i/y_i dans 38
96  1B      LDA A $ 1B  Pointer ce dernier résultat.
C6  3B      LD AB ## $ 3B Pointer le pas H
BD  897A    JSR FPPMUL  Calculer k_i
D6  1B      LDA B $ 1B  Pointer l'adresse de transfert
96  1A      LDA A $ 1A  Pointer le nombre à transférer
BD  885B    JSR FPPCPY  Transférer k_i dans 35 (1er pas)
D6  1A      LDA B $ 1A
96  1B      LDA A $ 1B  Pointer l'adresse de transfert
C0  03      SUB B ## $ 03 Pointer y_i à transférer
BD  885B    JSR  FPPCPY Transférer y_i vers 38 (1er pas)
39          RTS         Retour au programme principal
```

C) Résultats de la résolution (voir page 6;)

ORGANIGRAMMES DU CALCUL DES K_I:(Runge-Kutta)

EXEMPLE: 1

pointer x_i par Acc·A
pointer H par Acc·B

↓

FPPMUL
calcul de $x_i.H$

↓

pointer l'exp. de $x_i.H$
par Reg.Index

↓

incrémenter exp.
de $x_i.H \longrightarrow 2.x_i.H$

↓

RTS

EXEMPLE: 2

pointer x_i par Acc·A
pointer y_i par Acc·B

↓

FPPADD
calculer $(x_i + y_i)$

↓

pointer(x_i+y_i) par AccA
pointer H par Acc B

↓

FPPMUL
calcul $K_i = H.(x_i + y_i)$

↓

RTS

EXEMPLE: 3

pointer x_i par le
Reg. Index

↓

incrémenter exp.
de $x_i \longrightarrow 2.x_i$

↓

pointer $2x_i$ par Acc A
pointer y_i par Acc B

↓

FPPDIV
calcul de $D = 2x_i / y_i$

↓

pointer y_i par Acc·A
pointer D par Acc.B

↓

FPPSUB
calcul de $S = y_i - D$

↓

pointer S par Acc·A
pointer H par Acc·B

↓

FPPMUL
calcul de $K_i = H \cdot S$

↓

pointer K par Acc·B
pointer mémoire de
transfert par Acc·A

↓

FPPCPY
transfert de K_i

↓

pointer y par Acc·A
pointer mémoire de
tranfert par Acc·A

↓

FPPCPY
transfert de y_i

↓

RTS

Organigramme 2

PROGRAMME PRINCIPAL DE RESOLUTION D'EQUATION DIFFERENTIELLES

--- @ METHODE DE RUNGE-KUTTA

PAGE 001 RK4

```
00010                          NAM     RK4
00020 0013                     ORG     $13
00030 0013 0003     N6         RMB     3          RESERVER 3 BYTES POUR 6
00040 0016 0006     TRAV       RMB     6          LOCATION DE MEMOIRES DE TRAVA
00050 0038                     ORG     $38
00060 0038 0003     XL         RMB     3          VALEUR LIMITE DU DOMAINE
00070 003B 0003     H          RMB     3          PAS DE LA SIMULATION
00080 003E 0003     HH         RMB     3          MOITIER DE H
00090 0041 0003     X0         RMB     3          VALEUR INITIALE
00100 0044 0003     Y0         RMB     3          "       "
00110       88D5    FPPADD     EQU     $88I5
00120       897A    FPPMUL     EQU     $897A
00130       89EA    FPPDIV     EQU     $89EA
00140       885B    FPPCPY     EQU     $885B
00150       883A    FPPCMP     EQU     $883A
00160       8ADC    FPPSPK     EQU     $8ADC
00170 8000                     ORG     $8000
00171 8000 CE 6000             LDX     #$6000     MANTISSE DE 6
00172 8003 DF 13               STX     $13        STOCKER MANTISSE DS MEMOIRE 13
00173 8005 86 03               LDA A   #$3        EXPOSANT DE 6
00174 8007 97 15               STA A   $15        EXPOSANT DS 15
00180 8009 CE 0049             LDX     #$49       INITIALISER REG.INDEX
00190 800C C6 41               LDA B   #$41       POINTER XI
00200 800E 86 47               LDA A   #$47       POINTER MEMOIRE DE COPIE
00210 8010 DF 18    DEBUT      STX     $18
00220 8012 97 16               STA A   $16
00230 8014 D7 17               STA B   $17
00240 8016 BD 885B             JSR     FPPCPY     TRANSFERT DE XI
00250 8019 D6 17               LDA B   $17
00260 801B 96 16               LDA A   $16
00270 801D CB 03               ADD B   #$3        AJOUTER 3 A ACC B
00280 801F 8B 03               ADD A   #$3        "     "   ACC A
00290 8021 97 16               STA A   $16
00300 8023 D7 17               STA B   $17
00310 8025 BD 885B             JSR     FPPCPY     TRANSFERT DE YI
00320 8028 96 16               LDA A   $16
00330 802A D6 16               LDA B   $16
00340 802C 80 03               SUB A   #$3        SOUSTRAIRE 3
00350 802E 97 1A               STA A   $1A        SAUVEGARDER POUR FPPSPK
00360 8030 D7 1B               STA B   $1B        "       "    "
00370 8032 BD 8ADC             JSR     FPPSPK     CALCUL DE K0
00380 8035 96 16               LDA A   $16
00390 8037 D6 17               LDA B   $17
00400 8039 CB 03               ADD B   #$3
00410 803B 8B 03               ADD A   #$3
00420 803D 97 16               STA A   $16
00430 803F D7 17               STA B   $17
00440 8041 BD 885B             JSR     FPPCPY     TRANSFERT DE K0
00450 8044 C6 3E               LDA B   #$3E       POINTER H:2
00460 8046 96 17               LDA A   $17        MEMOIRE DE COPIE
00470 8048 97 16               STA A   $16
00480 804A BD 885B             JSR     FPPCPY     TRANSFERT DE (H:2)
00490 804D 96 16               LDA A   $16        POINTER XI
00500 804F D6 17               LDA B   $17
00510 8051 C0 06               SUB B   #$6
00520 8053 D7 17               STA B   $17
00530 8055 BD 88D5             JSR     FPPADD     CALCUL  DE (XI+HH)
00540 8058 96 16               LDA A   $16
```

.../...

```
PAGE  002   RK4

00550 805A D6 16        LDA B  $16
00560 805C CB 06        ADD B  #$6      POINTER K0
00570 805E 8B 09        ADD A  #$9
00580 8060 97 16        STA A  $16
00590 8062 BD 885B      JSR    FPPCPY   TRANSFERT DE K0
00600 8065 DE 18        LDX    $18
00610 8067 6A 09        DEC    9,X      DECREMENT EXP. DE KO
00620 8069 96 16        LDA A  $16
00630 806B D6 16        LDA B  $16      POINTER (K0/2)
00640 806D 80 06        SUB A  #$6      POINTER Y0
00650 806F 97 16        STA A  $16
00660 8071 D7 17        STA B  $17
00670 8073 BD 88D5      JSR    FPPADD   CALCUL DE (YI+K0/2)
00680 8076 96 17        LDA A  $17      MEMOIRE DE COPIE
00690 8078 D6 16        LDA B  $16
00700 807A C0 03        SUB B  #$3      POINTER XI+HH
00710 807C 97 16        STA A  $16
00720 807E D7 17        STA B  $17
00730 8080 BD 885B      JSR    FPPCPY   TRANSFERT DE (XI+HH)
00740 8083 BD 8ADC      JSR    FPPSPK   CALCUL DE K1
00750 8086 96 16        LDA A  $16
00760 8088 D6 17        LDA B  $17      POINTER K1
00770 808A 8B 03        ADD A  #$3      POINTER MEMOIRE DE COPIE
00780 808C 97 16        STA A  $16
00790 808E BD 885B      JSR    FPPCPY   TRANSFERT DE K1
00800 8091 DE 18        LDX    $18
00810 8093 6A 00        DEC    0,X      EXPOSANT DE K1--->K1/2
00820 8095 96 17        LDA A  $17      POINTER K1/2
00830 8097 D6 17        LDA B  $17
00840 8099 C0 03        SUB B  #$3      POINTER YI
00850 809B 97 16        STA A  $16
00860 809D BD 88D5      JSR    FPPADD   CALCUL DE (YI+K1/2)
00870 80A0 96 16        LDA A  $16
00880 80A2 D6 16        LDA B  $16      POINTER (YI+K1/2)
00890 80A4 8B 03        ADD A  #$3      POINTER MEMOIRE DE COPIE
00900 80A6 D7 17        STA B  $17
00910 80A8 BD 885B      JSR    FPPCPY   TRANSFERT DE (YI+K1/2)
00920 80AB 96 17        LDA A  $17      POINTER MEMOIRE DE COPIE
00930 80AD D6 17        LDA B  $17
00940 80AF CB 09        ADD B  #$9      POINTER (XI+HH)
00950 80B1 97 16        STA A  $16
00960 80B3 D7 17        STA B  $17
00970 80B5 BD 885B      JSR    FPPCPY   TRANSFERT DE (XI+HH)
00980 80B8 BD 8ADC      JSR    FPPSPK   CALCUL DE K2
00990 80BB 96 17        LDA A  $17
01000 80BD D6 16        LDA B  $16      POINTER K2
01010 80BF 8B 06        ADD A  #$6
01020 80C1 97 16        STA A  $16
01030 80C3 D7 17        STA B  $17
01040 80C5 BD 885B      JSR    FPPCPY   TRANSFERT DE K2
01050 80C8 96 16        LDA A  $16
01060 80CA C6 3E        LDA B  #$3E     POINTER HH
01070 80CC 80 06        SUB A  #$6      POINTER (XI+HH)
01080 80CE BD 88D5      JSR    FPPADD   CALCUL (XI+H)
01090 80D1 96 17        LDA A  $17      POINTER K2
01100 80D3 D6 17        LDA B  $17
01110 80D5 C0 03        SUB B  #$3      POINTER YI
01120 80D7 97 16        STA A  $16
```

.../...

```
PAGE  003   PK4

01130 80D9 BD 88D5      JSR     PPPADD   CALCUL DE (YI+K2)
01140 80DC 96 16        LDA A   $16
01150 80DE D6 16        LDA B   $16      POINTEUR (YI+K2)
01160 80E0 8B 03        ADD A   #$3      POINTEUR MEMOIRE DE COPIE
01170 80E2 D7 17        STA B   $17
01180 80E4 BD 885B      JSR     PPPCPY   TRANSFERT DE (YI+K2)
01190 80E7 96 17        LDA A   $17      MEMOIRE DE COPIE
01200 80E9 D6 17        LDA B   $17
01210 80EB CB 09        ADD B   #$9
01220 80ED D7 17        STA B   $17
01230 80EF BD 885B      JSR     PPPCPY   TRANSFERT DE (XI+H)
01240 80F2 BD 8A1C      JSR     PPPSPK   CALCUL DE K3
01250 80F5 DE 18        LDX     $18
01260 80F7 6C 0C        INC     12,X     EXPOSANT DE K1--->2K1
01270 80F9 6C 0F        INC     15,X     EXPOSANT DE K2--->2K2
01280 80FB 96 17        LDA A   $17
01290 80FD D6 17        LDA B   $17
01300 80FF 8B 03        ADD A   #$3      POINTEUR 2K1
01310 8101 C0 03        SUB B   #$3      POINTEUR K0
01320 8103 97 16        STA A   $16
01330 8105 D7 17        STA B   $17
01340 8107 BD 88D5      JSR     PPPADD   CALCUL DE S1=2.K1+K0
01350 810A 96 16        LDA A   $16      POINTEUR S1
01360 810C D6 17        LDA B   $17
01370 810E CB 09        ADD B   #$9      POINTEUR 2K2
01380 8110 D7 17        STA B   $17
01390 8112 BD 88D5      JSR     PPPADD   CALCUL DE S2=S1+2.K2
01400 8115 96 16        LDA A   $16      POINTEUR S2
01410 8117 D6 17        LDA B   $17
01420 8119 C0 0F        SUB B   #$F      POINTEUR K3
01430 811B D7 17        STA B   $17
01440 811D BD 88D5      JSR     PPPADD   CALCUL DE S3=S2+K3
01450 8120 96 16        LDA A   $16      POINTEUR S3
01460 8122 C6 13        LDA B   #$13     POINTEUR 6 EN FLOTTANT
01470 8124 BD 89EA      JSR     PPPDIV   CALCUL DE      =S3/6
01480 8127 96 16        LDA A   $16      POINTEUR
01490 8129 D6 17        LDA B   $17
01500 812B C0 03        SUB B   #$3      POINTEUR YI
01510 812D BD 88D5      JSR     PPPADD   CALCUL DE Y(I+1)
01520 8130 96 16        LDA A   $16
01530 8132 D6 16        LDA B   $16      POINTEUR Y(I+1)
01540 8134 80 09        SUB A   #$9      POINTEUR MEMOIRE DE COPIE
01550 8136 97 16        STA A   $16
01560 8138 D7 17        STA B   $17
01570 813A BD 885B      JSR     PPPCPY   TRANSFERT DE Y(I+1)
01580 813D 96 16        LDA A   $16
01590 813F D6 17        LDA B   $17
01600 8141 C0 12        SUB B   #$12
01620 8143 80 03        SUB A   #$3      POINTEUR MEMOIRE DE COPIE
01630 8145 97 16        STA A   $16
01640 8147 D7 17        STA B   $17
01650 8149 BD 885B      JSR     PPPCPY   TRANSFERT DE XI
01660 814C 96 16        LDA A   $16      POINTEUR XI
01670 814E C6 3B        LDA B   #$3B     POINTEUR H
01680 8150 BD 88D5      JSR     PPPADD   CALCUL DE X(I+1)=XI+H
01690 8153 96 16        LDA A   $16      POINTEUR X(I+1)
01700 8155 C6 38        LDA B   #$38     POINTEUR XL
01710 8157 BD 883A      JSR     PPPCMP   COMPARER X(I+1)-XL
```

.../...

```
PAGE  004   RK4

01720 815A 2B 01          BMI     BOUCLE    SI MOINS
01730 815C 3F             SWI               SI POSITIF OU NUL
01740 815D 96 16    BOUCLE LDA A  $16
01750 815F D6 17           LDA B  $17
01760 8161 8B 06           ADD A  #$6
01770 8163 CB 06           ADD B  #$6
01780 8165 DE 18           LDX    $18
01790 8167 08              INX               AJOUTER 6
01800 8168 08              INX
01810 8169 08              INX
01820 816A 08              INX
01830 816B 08              INX
01840 816C 08              INX
01850 816D 7E 8010         JMP    DEBUT      PAS SUIVANT
01860      0000            END

TOTAL ERRORS 00000
```

TABLE DES DONNEES

ADRESSE	Contenu des mémoires	ADRESSE	Contenu des mémoires
00 ↓ 10	Réservées pour BIBMAT	1C	Vide
11	Vide		
12	Vide		
13	60		
14	00		
15	03		
	Sauvegarder 6 en virgule flottante.		
16	Sauvegarder Acc. A	37	Vide
17	Sauvegarder Acc. B	38	X_L Valeur limite du domaine
18	Sauvegarder le Reg. Index.	3B	H pas de la Simulation.
19		3E	H/2 moitié du pas.
1A	Sauvegarder Acc. A et B pour FPPSPK.	41	x_0 Valeur initiale.
1B		44	y_0 Valeur initiale.

TABLE DES RESULTATS EN VIRGULE FLOTTANTE

41	x_0	62	y_5	83	x_{11}	A4	y_{16}
44	y_0	65	x_6	86	y_{11}	A7	x_{17}
47	x_1	68	y_6	89	x_{12}	AA	y_{17}
4A	y_1	6B	x_7	8C	y_{12}	AD	x_{18}
4D	x_2	6E	y_7	8F	x_{13}	B0	y_{18}
50	y_2	71	x_8	92	y_{13}	B3	x_{19}
53	x_3	74	y_8	95	x_{14}	B6	y_{19}
56	y_3	77	x_9	98	y_{14}	B9	x_{20}
59	x_4	7A	y_9	9B	x_{15}	BC	y_{20}
5C	y_4	7D	x_{10}	9E	y_{15}	BF	x_{21}
5F	x_5	80	y_{10}	A1	x_{16}	C2	y_{21}

RESULTATS DU 1er , 2eme, ET 3eme EXEMPLES

PAR LA METHODE DE RUNGE-KUTTA

```
    ↑9-Y4FE-2E C-FA S-FF82 ↑9
EXBUG 1.2 MAID
*7200;G -------->INTRODUCTION DE LA TABLE DE DONNEES
?2    - -------->VALEUR LIMITE DE L'INTERVALLE
  400002
?.1   - - ------->H: PAS DE LA SIMULATION
   6666FD
?.05  - - ------->HH: MOITIER DU PAS
    6666FC
?0 - - - ------->X0: VALEUR INITIALE
  000001
?0- - -  ------->Y0: VALEUR INITIALE (Y0=1 POUR EXEMPLE 3 ET 2)
  000001
?0 - ----------> VALEUR EN PLUS
  000001
BKPT ERROR
P-7225 X-0047 A-4A B-4A C-D4 S-00FF
*8000;G --------> PROGRAMME PRINCIPAL
BKPT ERROR
P-815C X-00F9 A-66 B-40 C-D0 S-00FF
*7000;G -------->TABLE DE RESULTATS
```

0 0	0 1.0000	0 1.0000
.10000 .010000	.10000 1.1104	.10000 1.0955
.20000 .040002	.20000 1.2428	.20000 1.1832
.30000 .090004	.30000 1.3997	.30000 1.2649
.40000 .16000	.40000 1.5837	.40000 1.3416
.50002 .25001	.50002 1.7975	.50002 1.4142
.60008 .36002	.60008 2.0444	.60008 1.4832
.70004 .49006	.70004 2.3276	.70004 1.5491
.80004 .64002	.80004 2.6513	.80004 1.6124
.90004 .81012	.90004 3.0194	.90004 1.6733
1.0000 1.0000	1.0000 3.4368	1.0000 1.7319
1.1000 1.2101	1.1000 3.9086	1.1000 1.7887
1.2000 1.4402	1.2000 4.4408	1.2000 1.8438
1.3000 1.6902	1.3000 5.0390	1.3000 1.8972
1.3999 1.9602	1.3999 5.7110	1.3999 1.9491
1.4999 2.2501	1.4999 6.4640	1.4999 1.9997
1.5999 2.5600	1.5999 7.3072	1.5999 2.0491
1.6999 2.8900	1.6999 8.2496	1.6999 2.0973
1.7999 3.2400	1.7999 9.3012	1.7999 2.1445
1.8999 3.6100	1.8999 10.474	1.8999 2.1905
1.9999 4.0000	1.9999 11.760	1.9999 2.2355
2.0999 4.4100	2.0999 13.235	2.0999 2.2798
Exemple:1	Exemple:2	Exemple:3
$\dot{y}(x)=2x$	$\dot{y}(x)=x+y(x)$	$y(x)=y(x)-\frac{2x}{y(x)}$
$[0\to 2]$, $x_0=0$, $y_0=$	$[0\to 2]$, $x_0=0$, $y_0=1$	$[0\to 2]$, $x_0=0$, $y_0=1$

III - C. METHODE D'EULER 1° ORDRE

I) Elaboration du programme de résolution :

1°) Présentation de la méthode

Comme la méthode de RUNGE-KUTTA, la méthode d'EULER permet de trouver les solutions approchées des équations différentielles du premier ordre. Ces équations sont de la forme :

$$\dot{y}(x) = f(x, y(x))$$

Avec les conditions initiales : $y = y_o$ pour $x = x_o$

dans le domaine $D = (x_o, X_L)$ où la fonction est définie et continue. Cette méthode utilise le développement des séries de Taylor :

$$y(x + h) = y(x) + \frac{h}{1!} y'(x) + \frac{h^2}{2!} y''(x) + \ldots$$

Mais en ignorant les termes contenant h^2 et plus.

Pour cela on découpe le segment (x_o, X_L) en n parties égales et notons

$$\Delta x = h = x_1 - x_o = x_2 - x_1 = \ldots = X_L - x_{n-1}$$

d'où $h = \frac{XL - xo}{N}$

Notons aussi $\Delta y_1 = y_1 - y_o$, $\Delta y_2 = y_2 - y_1$, ..., $\Delta y_n = y_n - y_{n-1}$

à chaque point $x_o, x_1, \ldots, (x_n = X_L)$ de $\frac{dy}{dx} = f(x,y)$ on remplace la dérivée par les différences finies :

soit : $\frac{\Delta y}{\Delta x} = f(x,y)$

Au

Au point x_o on a $\Delta y_1 = (f\ x_o, y_o) \cdot \Delta x$ soit $y_1 - y_o = h.f\ (x_o, y_o)$.

Au point x_1 on a $\Delta y_2 = f\ (x_1, y_1) \cdot \Delta x$ soit $y_2 - y_1 = hf(x_1, y_1)$.

Connaissons x_o, y_o et h on peut déterminer y_1.

Puis connaissons $x_1 = x_o + h$, y_1 et h on détermine y_2.

Et ainsi pas à pas nous déterminons les valeurs approchées de y :

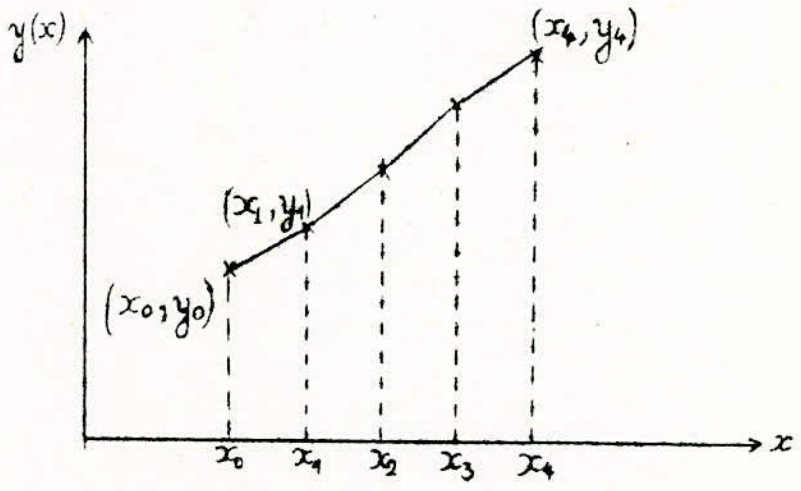
$$y_1 = y_o + hf(x_o, y_o) \quad \text{avec } x_1 = x_o + h$$

$$y_2 = y_1 + hf(x_1, y_1) \quad \text{avec } x_2 = x_1 + h$$

$$\ldots\ldots\ldots\ldots \qquad \ldots\ldots\ldots$$

$$y_n = y_{n-1} + h\ f\ (x_{n-1}, y_{n-1}) \text{ Avec } x_n = x_{n-1} + h.$$

Sur le plan des coordonnées les couples de points (x_o, y_o), ... (x_n, y_n) forment des segments de droite. Cette courbe est appelée la ligne brisée d'EULER.



Ligne brisée d'Euler.

d'autre part on peut représenter l'erreur pour y_1 par :

$$\varepsilon = \frac{h^2}{2!} y_o^{(2)} + \frac{h^3}{3!} y_o^{(3)} + \ldots$$

Et de même on peut introduire une erreur similaire pour chaque pas dans les calculs pour $y_2 \ \ldots \ y_n$.

On remarque déjà que cette erreur est élevée et on peut déjà prévoir que le résultat ne sera pas très exact surtout en comparaison avec la méthode de RUNGE-KUTTA.

2) Estimation du temps de résolution :

La durée d'un pas pour une formule d'intégration peut - être estimée d'après la formule spécifiée pour la méthode.

La méthode d'EULER requiert une addition et une multiplication durant chaque pas de calcul d'intégration si on désigne par t_T le temps de calcul d'un pas tE le temps de calcul de f(x,y), α le coefficient moyen du temps d'addition et β le coefficient moyen du temps de multiplication, alors l'estimation du temps de calcul du pas d'intégration est donnée par :

$$t_T = t_E + \alpha + \beta$$

3°) Programme de résolution :

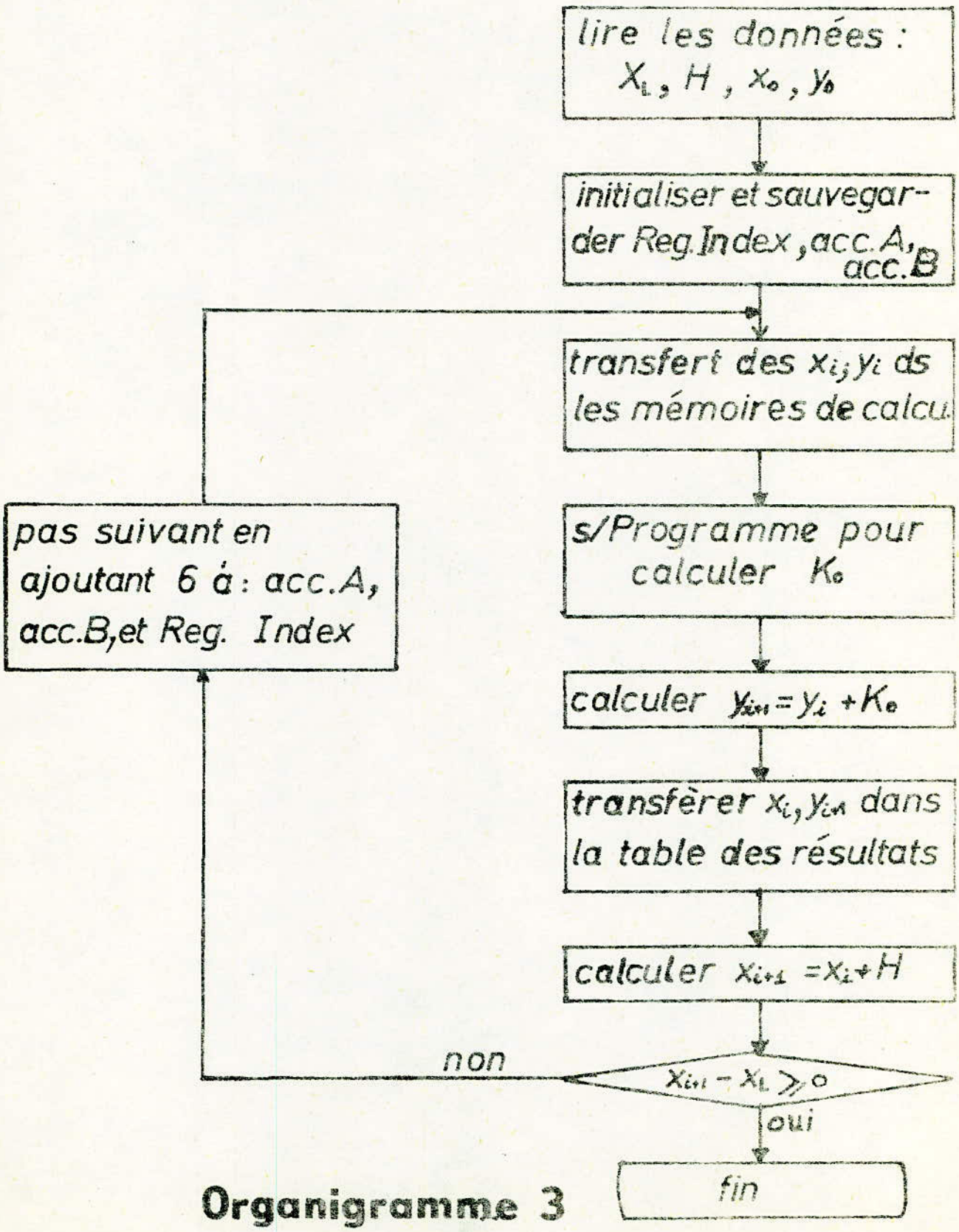
On procède de la même manière que la méthode de RUNGE-KUTTA 4° Ordre c'est à dire en utilisant une soubroutine de calcul de $k_o = h.f\ (x_i\ ,\ y_i)$ (FPPSPK).

a) Organigramme (voir organigramme 3)

* On introduit x_o, y_o, X_L et H en mémoires à l'aide du programme de chargement de table (page 50) qui est encore valable pour la méthode d'EULER.

.../...

ORGANIGRAMME PRINCIPAL DE RESOLUTION (EULER)



Organigramme 3

¥ Pour incrémenter le pas de simulation, on ajoute (6) à chacun des accumulateurs A et B et du registre d'Index.

¥ y_{i+1}, est calculé en ajoutant à y_i la valeur de k_o

¥ x_i est augmentée du pas. H pour obtenir x_{i+1}.

¥ les valeurs x_i et y_i d'un pas sont stockées dans la table des résultats.

¥ un test est fait si le calcul est complet. Sinon en reboucle une nouvelle fois.

b) Programme principal (page 69)

Après éxécution du programme les résultats sont stockés dans la table réservée à cet effet (page 60).

C) Programme de chargement de table.

On utilise le même programme que celui utilisé pour la méthode de RUNGE - KUTTA malgré quelques valeurs inutiles qui n'affectent pas la résolution. (Voir page 50).

d) Programme de sortie de table

Il est commun aux deux méthodes RUNGE - KUTTA et EULER (voir page 51).

II - EXEMPLES DE RESOLUTION, RESULTATS SUR ECRAN DE VISUALISATION.

La marche à suivre pour la résolution est la même que pour la méthode de RUNGE - KUTTA.

1°) Exemple 1.

$\dot{y}(x) = f(x,y) = 2x$

Avec :
- valeurs initiales : $x_o = 0$, $y_o = 0$
- Intervalle d'étude : $(0,2)$; $X_L = 2$
- la valeur du pas : $H = \frac{XL - xo}{N}$
  N = 20 nombre de pas: ----) $H = \frac{2 - 0}{20} = 0,1$

a) Organigramme de FPPSPK (Voir organigramme 4)

b) Programme de FPPSPK

ORG $ 8ADC

96 1A	LDA A $ 1A		Pointer xi par lec contenu de 1 A
C6 3B	LDA B # $ 3B		
BD 897A	JSR FPPMUL		Calcul x_i H dans adresse 47
DE 18	LDX $ 18		Contenu de 18 dans le registre Index
6C 00	INC 0,X		Incrémentér exposant de x_iH ---) $2x_i$H.
39	RTS		Retour au programme principal

c) Résultats (page 70)

2°) Exemple 2

$\dot{y}(x) = f(x, y(x)) = x + y$

avec
- valeurs initiales $y_o = 1$ pour $x_o = 0$
- Intervalle d'étude (0, 2)
- nombre de pas N = 20 -----) $H = \frac{2-0}{20} = 0;1$

.../...

a) Organigramme de FPPSPK (Voir Organigramme 4)

b) Programme de FPPSPK

```
                ORG $ 8 ADC
96 1A           LDA A $ 1 A     Pointer x_i
D6 1B           LDA B $ 1B      Pointer y_i
BD 88D5         JSR FPPADD      Calcul de x_i + y_i
96 1A           LDA A $ 1 A     Pointer (x_i+y_i)
C6 3B           LDA B ≠≠ $ 3B   Pointer H.
BD 897A         JSR FPPMUL      Calcul k_o = H (x_i+y_i)
39              RTS             Retour au programme principal
```

c) Résultats (Voir page 70)

3°)Exemple 3 :

$$\dot{y}\ (x) = f(x,y(x)) = y\ (x) - \frac{2x}{y(x)}$$

avec :

- valeurs initiales : $y_o = 1$ pour $x_o = 0$
- intervalle d'étude: $(0,2)$
- nombre de pas N = 20 ----) $H = \frac{2 - 0}{20} = 0,1$

a) Organigramme de FPPSPK (Voir Organigramme 4)

b) Programme de FPPSPK

```
                ORG $ 8ADC
DE  18      LDX $18
6C  00      INC 0,X         INC exposant de x_i ----) 2x_i
96  1A      LDA A  $1A      Pointer 2 x_i
D6  1B      LDA B  $ 1B     Pointer y_i
BD  89EA    JSR  FPPDIV     Calcul de 2x_i/y_i
D6  1B      LDA B $ 1B      Pointer y_i
96  1A      LDA  $ 1A
8B  03      ADDA ## $ 03    Pointer mémoire de transfert
BD  885B    JSR FPPCPY      Transfert de y_i
96  1A      LDA A $ 1A      Pointer y_i
8B  03      ADDA ## $ 03
D6  1A      LDA B $ 1A      Pointer 2 x_i / y_i
BD  887D    JSR FPPSUB      Calcul y_i - 2 x_i/ y_i
96  1A      LDA A $ 1A
8B  03      ADD A ## 03     Pointer résultat précédent
C6  3B      LDA B ## $ 3B   Pointer H
BD  897A    JSR FPPMUL      Calcul k_o = H ( y_i - 2x_i/ y_i)
D6  1A      LDA B $ 1A
CB  03      ADDB ## $ 03    Pointer k_o
96  1A      LDA A $ 1A      Pointer mémoire de copie
BD  885B    JSR FPPCPY      Transfert de k_o .
39          RTS             Retour au programme principal .
```

c) Résultats de résolution (voir page 70)

ORGANIGRAMME DU CALCUL DES K_I : (Euler)

EXEMPLE : 3

- pointer exposant de x_i par Reg. Index
- increment. expos. de $x_i \longrightarrow 2x_i$
- pointer $2x_i$ par acc. A et y par acc.B
- FPPDIV calcul de $D = 2x_i / y_i$
- pointer memoire de transfert par acc. A
- FPPCPY transfert de y_i
- pointer y_i par acc.A et D par acc. B
- FPPSUB calcul de $S = y_i - D$
- pointer S par acc.A et H par acc. B
- FPPMUL calcul de $K_o = H.S$
- pointer K_o par Acc.B et mem. par Acc.A
- FPPCPY transfert de K_o
- RTS

EXEMPLE : 2

- pointer x_i par Acc.A pointer y_i par Acc. B
- FPPADD calcul de $(x_i + y_i)$
- pointer $(x_i + y_i)$ par Acc.A pointer H par Acc.B
- FPPMUL calcul $K_o = (x_i + y_i)$
- RTS

EXEMPLE : 1

- pointer x_i par Acc.A pointer H par Acc.B
- FPPMUL calcul de $x_i H$
- pointer exposant de $x_i H$ par Reg. Inx.
- increment. exposant de $x_i H \longrightarrow K_o = 2x_i H$
- RTS

Organigramme 4

PROGRAMME PRINCIPAL DE RESOLUTION D'EQUATIONS DIFFERENTIELLES

METHODE D'EULER

```
PAGE  001   EULER

00010                          NAM     EULER
00020 0018                     ORG     $18
00030 0018 0004     TRAV       RMB     4          LOCATION MEMOIRES DE TRAVAIL
00050 0038                     ORG     $38
00060 0038 0003     XL         RMB     3          EXTREMITE DE L'INTERVALLE
00070 003B 0003     H          RMB     3          PAS DE CALCUL
00080 003E 0003     X0         RMB     3          VALEUR INITIALE
00090 0041 0003     Y0         RMB     3          "       "
00100      885B     PRCOPY     EQU     $885B
00110      8ADC     PRSPEK     EQU     $8ADC
00120      88D5     PRPADD     EQU     $88D5
00130      883A     PRCMP      EQU     $883A
00140 7800                     ORG     $7800
00150 7800 CE 0049             LDX     #$49       INITIALISER LE REG.INDEX
00160 7803 C6 41               LDA B   #$41       POINTER X1
00170 7805 86 47               LDA A   #$47       ADRESSE DE COPIE DE X1
00180 7807 DF 18    DEBUT      STX     $18
00190 7809 97 1A               STA A   $1A        SAUVER CONTENU DE ACC A
00200 780B D7 1B               STA B   $1B        "       "    "  ACC B
00210 780D BD 885B             JSR     PRCOPY     COPIER X1
00220 7810 96 1A               LDA A   $1A        POINTER X1
00230 7812 D6 1B               LDA B   $1B        POINTER Y1
00240 7814 CB 03               ADD B   #$3        POINTER Y1
00250 7816 D7 1B               STA B   $1B
00260 7818 BD 8ADC             JSR     PRSPEK     CALCUL DE K0
00270 781B 96 1A               LDA A   $1A        POINTER K0
00280 781D D6 1B               LDA B   $1B        POINTER Y1
00290 781F BD 88D5             JSR     PRPADD     CALCUL DE Y(I+1)
00300 7822 D6 1A               LDA B   $1A        POINTER Y(I+1)
00310 7824 96 1A               LDA A   $1A
00320 7826 8B 03               ADD A   #$3        POINTER MEMOIRE DE COPIE
00330 7828 97 1A               STA A   $1A
00340 782A D7 1B               STA B   $1B
00350 782C BD 885B             JSR     PRCOPY     COPIER Y(I+1)
00360 782F 96 1B               LDA A   $1B
00370 7831 D6 1B               LDA B   $1B
00380 7833 C0 06               SUB B   #$6        SOUSTRAIRE 6
00390 7835 97 1A               STA A   $1A
00400 7837 D7 1B               STA B   $1B
00410 7839 BD 885B             JSR     PRCOPY     COPIER X0 DS 35
00420 783C 96 1A               LDA A   $1A
00430 783E C6 3B               LDA B   #$3B       POINTER LE PAS H
00440 7840 BD 88D5             JSR     PRPADD     CALCUL DE X(I+1)
00450 7843 96 1A               LDA A   $1A        POINTER X(I+1)
00460 7845 C6 38               LDA B   #$38       POINTER XL
00470 7847 BD 883A             JSR     PRCMP      SI X(I+1)-XL >0
00480 784A 2B 01               BMI     BOUCLE     BRANCHER SI NON
00490 784C 3F                  SWI
00500 784D 96 1A    BOUCLE     LDA A   $1A
00510 784F D6 1B               LDA B   $1B
00520 7851 8B 06               ADD A   #$6        AJOUTER 6
00530 7853 CB 06               ADD B   #$6
00540 7855 DF 18               LDX     $18
00550 7857 08                  INX                AJOUTER 6 A REG.INDEX
00560 7858 08                  INX
00570 7859 08                  INX
00580 785A 08                  INX                !
00590 785B 08                  INX
00600 785C 08                  INX
00610 785D 7F 7807             JMP     DEBUT
00620      0000                END
```

RESULTATS DU 1er , 2eme, ET 3eme EXEMPLES

PAR METHODE D' EULER

```
"4HM" X-FF08 A-82 B-2E C-F8 S-FF82 :9
EXBUG 1.2 MAID
*7200;G -------> INTRODUCTION DE LA TABLE DE DONNEES
?2  -  -  ----->VALEUR LIMITE DE L'INTERVALLE:XL
  400002
?.1 - - - ----->H:PAS DE LA SIMULATION
   6666FD
?.05 - - ------>HH:MOITIER DU PAS
    6666FC
?0  -- - ------>X0:VALEUR INITIALE
  000001
?0 - - - ------>Y0:VALEUR INITIALE(Y0=1 POUR EXEMPLE 2 ET 3)
  000001
?0  - - - ----->EN PLUS
  000001
BKPT ERROR
P-723C X-0047 A-4A B-4A C-D4 S-00FF
*7800;G---------->PROGRAMME PRINCIPAL
BKPT ERROR
P-784C X-00F9 A-66 B-40 C-D0 S-00FF
*7000;G --------->TABLE DE RESULTATS
```

0 0	0 1.0000	0 1.0000
.10000 0	.10000 1.1000	.10000 1.1000
.20000 .020000	.20000 1.2199	.20000 1.1918
.30000 .060000	.30000 1.3620	.30000 1.2774
.40000 .12000	.40000 1.5282	.40000 1.3581
.50002 .20000	.50002 1.7210	.50002 1.4350
.60008 .30004	.60008 1.9432	.60008 1.5088
.70004 .42002	.70004 2.1976	.70004 1.5802
.80004 .56008	.80004 2.4874	.80004 1.6496
.90004 .72012	.90004 2.8161	.90004 1.7176
1.0000 .90016	1.0000 3.1876	1.0000 1.7844
1.1000 1.1001	1.1000 3.6064	1.1000 1.8509
1.2000 1.3202	1.2000 4.0772	1.2000 1.9171
1.3000 1.5600	1.3000 4.6052	1.3000 1.9836
1.3999 1.8202	1.3999 5.1958	1.3999 2.0509
1.4999 2.1001	1.4999 5.8552	1.4999 2.1195
1.5999 2.4003	1.5999 6.5912	1.5999 2.1899
1.6999 2.7201	1.6999 7.4096	1.6999 2.2629
1.7999 3.0600	1.7999 8.3204	1.7999 2.3389
1.8999 3.4202	1.8999 9.3332	1.8999 2.4189
1.9999 3.8000	1.9999 10.457	1.9999 2.5038
2.0999 4.2002	2.0999 11.702	2.0999 2.5943
Exemple : 1	**Exemple : 2**	**Exemple : 3**
$\dot{y}(x) = 2 \cdot x$	$\dot{y}(x) = x + y(x)$	$\dot{y}(x) = y(x) - \frac{2 \cdot x}{y(x)}$
$[0 \to 2]$, $x_0 = 0$, $y_0 = 0$	$[0 \to 2]$ $x_0 = 0$, $y_0 = 1$	$[0 \to 2]$, $x_0 = 0$ $y_0 = 1$

III - D -- COMPARAISON DES DEUX METHODES.

La solution d'une équation différentielle ordinaire par un calculateur digital, tels que les résultats sont exactements spécifiés précédemment sont obtenus dans un temps minimum de calcul, exige une attentive considération et un contrôle d'erreurs propre au procédé numérique seulement de cette façon qu'on peut sélectionner judicieusement le pas de simulation et la complexité de la formule d'intégration à savoir le nombre de termes de la série de Taylor, qui pourront être utilisées.
Dans le traitement des erreurs de calcul, on distingue deux types :

- Erreur dûe à chaque pas
- Accumulation de ces erreurs le long du calcul.

Pour caractériser l'erreur introduite à une étape spécffique du calcul il est nécessaire de tenir compte de l'erreur d'arrondissement et de l'erreur de troncature.

L'erreur d'arrondissement est dûe aux performances du calculateur (limite de ces registres) et de la bibliothèque mathématiques utilisée (nombre de bits en virgule flottante).
Cela devient nécessaire d'arrondir tous les nombres à intégrer sous le même format.

L'erreur de troncature provient du fait que pratiquement toutes les méthodes d'intégration constituent une approximation du système réel.. En général cette approximation est intérprêtée en ignorant l'ordre élevé des termes de la série de Taylor. Ces erreurs sont proportionnelles à la puissance du pas H et à l'ordre des dérivées des variables dépendantes.

D'après ce qui précède on peut déduire que l'erreur de troncature pour la méthode de Runge-Kutta est très faible devant celle d'EULER du moment que cette dernière s'arrête à la première puissance de H et à la première dérivée, par contre la première méthode s'arrête à la puissance quatrième de H (H^4) et à la troisième dérivée.
Mais en ce qui concerne l'erreur d'arrondi elle est la même pour les deux méthodes puisqu'on travaille sur le même calculateur et on utilise la même bibliothèque mathématiques de 24 bits en virgule flottante.
L'erreur absolue entre la valeur vraie et la valeur approchée est donnée dans les tableaux de comparaison si-après pour les trois points(x=0;0,5;1).

.../...

On remarque que les résultats donnés par la méthode de Runge-Kutta sont très proches des résultats exacts. Par contre les résultats donnés par la méthode d'Euler sont loins des valeurs exactes. Ceci est confirmé par le tracé point par point des courbes représentatives des deux méthodes.

On remarque encore que les courbes de la méthode de Runge-Kutta sont confondues avec la courbe exacte de la fonction; ce qui n'est pas vrai, mais nous ne pouvons apprécier plus que cela. Par contre la courbe de la méthode d'Euler est loin de la courbe exacte.

Avantages de la méthode de Runge-Kutta.

Cette méthode présente l'avantage de changer le pas de simulation durant l'évolution du système. En effet on voit que l'erreur augmente de plus en plus avec le nombre de pas de sorte que la courbe approchée diverge de la courbe réelle donc on peut remédier à cela en diminuant la valeur du pas H.

Ce phénomène est utilisé lors de l'étude des systèmes à temps réel où le contrôle s'effectue durant l'évolution du processus.

On peut encore améliorer le résultat en utilisant une bibliothèque mathématiques de plus de 24 bits en virgule flottante (32;64 bits ...)

L'inconvénient de cette méthode est le temps d'éxécution. Plus l'ordre des termes de la méthode est élevé plus le programme de résolution se complique, et plus le temps d'éxécution est long.

TABLEAU D'ERREURS

EXEMPLE 1

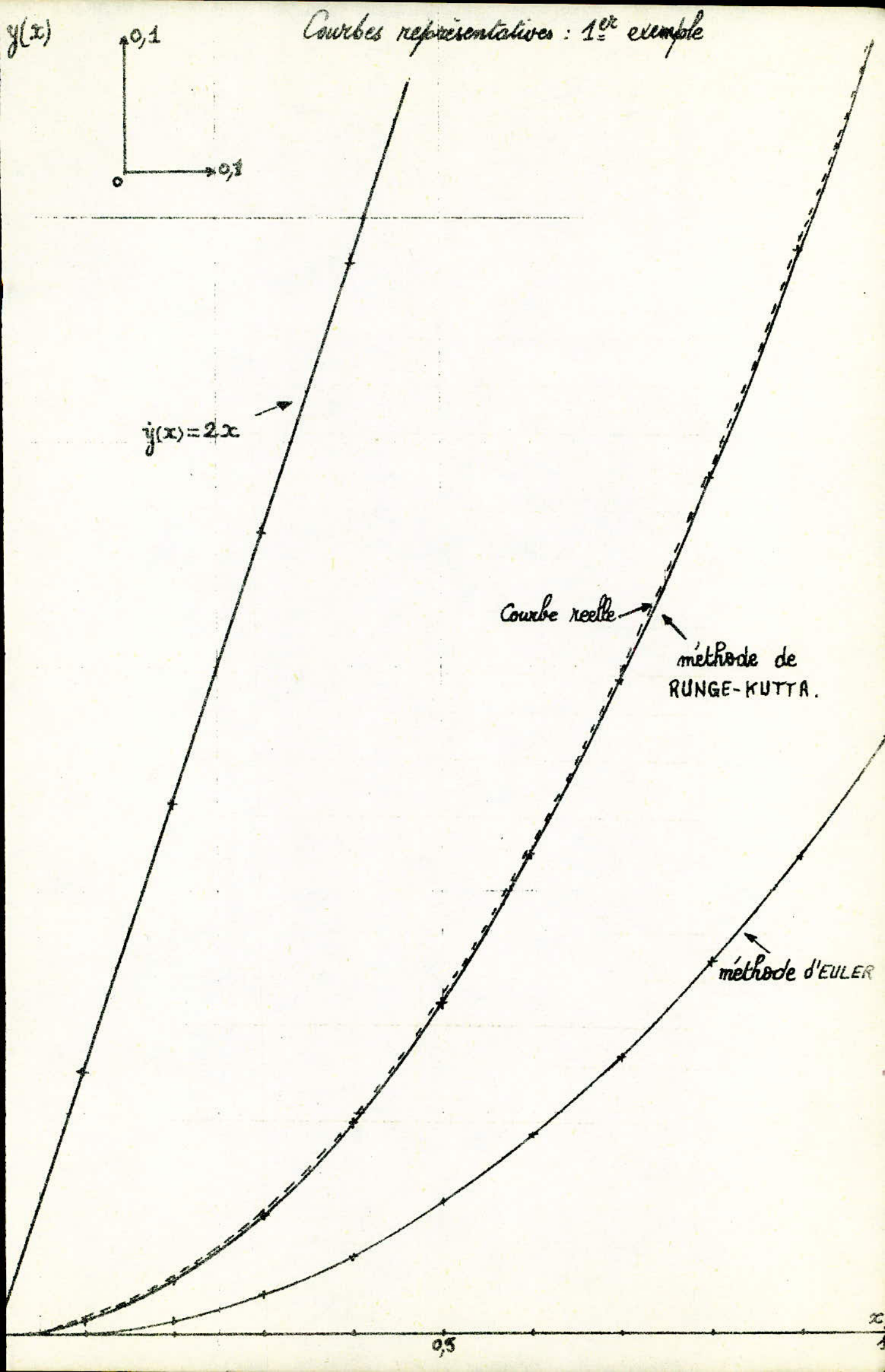
x	RK 4ºORDRE	EULER
0	0	0
0,5000	$0,1000 \cdot 10^{-4}$	0,1500
1,0000	0	0,5500

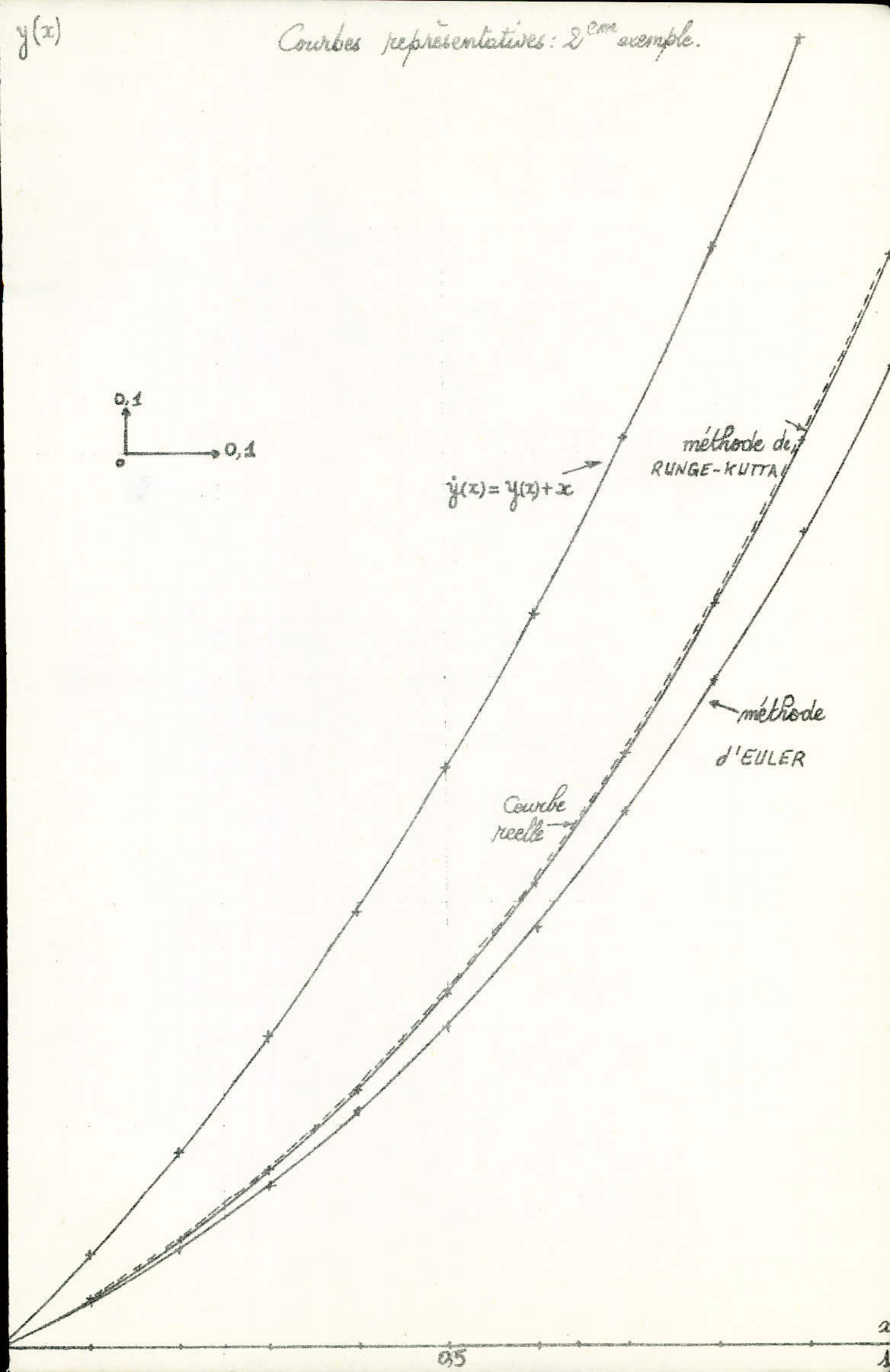
EXEMPLE 2

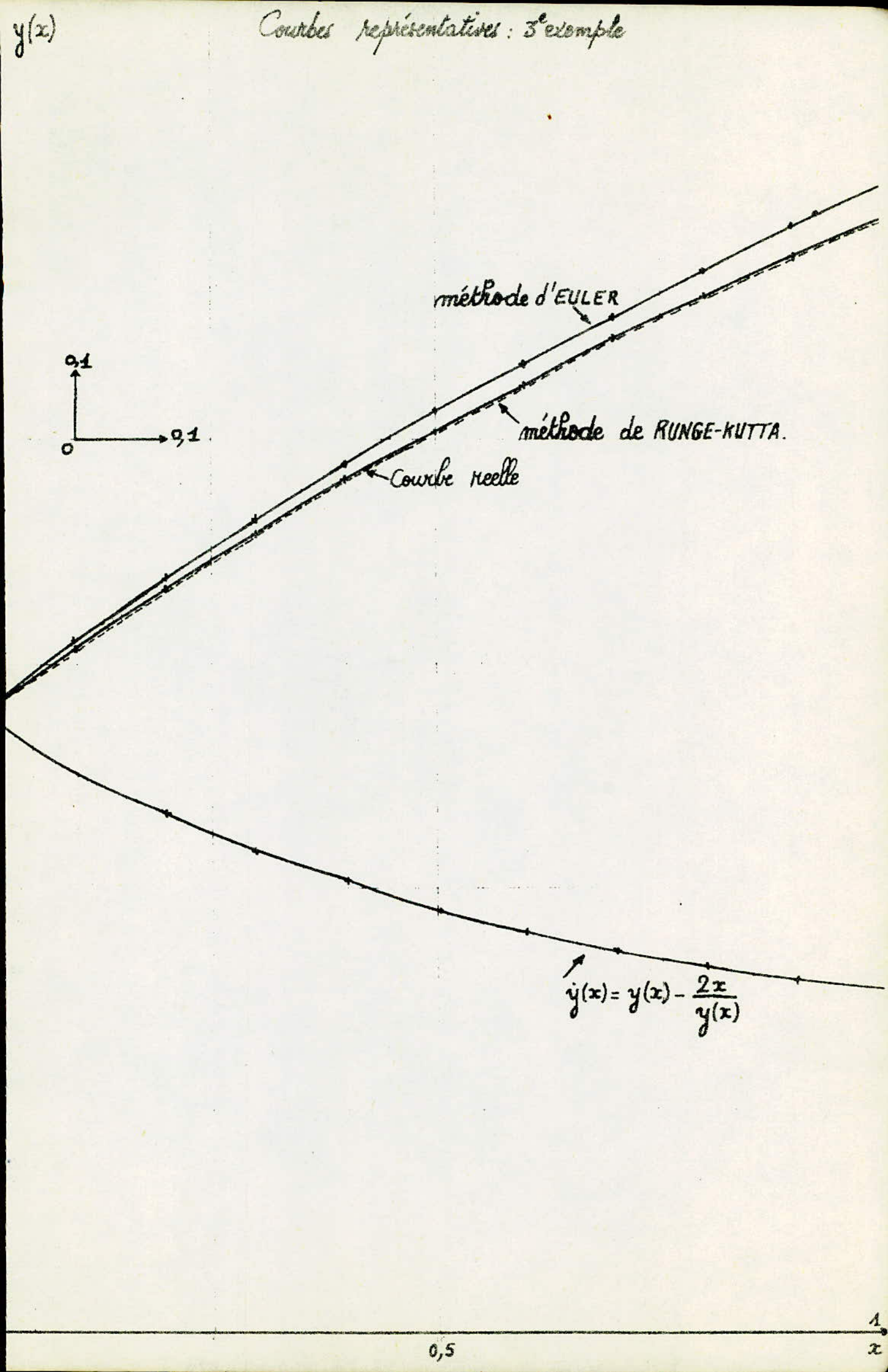
x	RK 4ºORDRE	EULER
0	0	0
0,5000	$0,5740 \quad 10^{-4}$	$0,7640 \quad 10^{-1}$
1,0000	$0,2360 \quad 10^{-3}$	0,2489

EXEMPLE 3

x	RK 4ºORDRE	EULER
0	0	0
0,5000	$0,1360 \quad 10^{-4}$	$0,2080 \quad 10^{-1}$
1,0000	$0,1510 \quad 10^{-3}$	$0,5230 \quad 10^{-1}$


y(x)
0,1
0,1
Courbes représentatives : 1er exemple
y(x) = 2x
Courbe reelle
méthode de
RUNGE-KUTTA.
méthode d'EULER
0,5
x


y(x)
Courbes représentatives : 2ème exemple.
0,1
0,1
ẏ(x) = y(x) + x
méthode de RUNGE-KUTTA
méthode d'EULER
Courbe réelle
x
1


y(x)
Courbes représentatives : 3e exemple
méthode d'EULER
méthode de RUNGE-KUTTA.
Courbe réelle
0,1
0,1
0
$\dot{y}(x) = y(x) - \frac{2x}{y(x)}$
1
x
0,5

C O N C L U S I O N

La programmation en langage machine était pour nous un domaine inconnu avant d'entamer ce projet de résolution d'équations différentielles par microprocesseurs. Après l'étude de plusieurs programmes que renferme la bibliothèque mathématiques de MOTOROLA, nous nous sommes adaptés aux instructions utilisées et à l'élaboration de nos propres programmes.

Ce modeste travail n'est qu'une étape du projet de SIMULATION ET CONTROLE D'UN REACTEUR NUCLEAIRE qui est un processus physique représenté par un système d'équations différentielles dont le nombre dépend du modèle choisi. Notre travail a consisté à concevoire l'opération de l'intégration d'une équation différentielle. Alors le problème se complique encore davantage dans la mise en parallèle de plusieurs opérations de même type gérées par un système informatique permettant de simuler l'ensemble avec des performances supérieures. Il reste donc à élargir cette étude correspondant aux travaux entrepris dans la division Y du C.S.TN.