UNIVERSITE DES SCIENCES
ET DE LA TECHNOLOGIE
USTHB

ECOLE NATIONALE
POLYTECHNIQUE
D'ALGER

Département d'Electronique et d'Electrotechnique

PROJET DE FIN D'ETUDES

Thèse d'Ingéniorat

MICROPROCESSEUR ET UNITE ARITHMETIQUE RAPIDE

Proposé par :

H.TEDJINI
Docteur Ingénieur

Etudié par :

T. ABDENNEBI
Z. KOUIDER
née YAHIAOUI

JUIN 1981

UNIVERSITE DES SCIENCES
ET DE LA TECHNOLOGIE
USTHB

ECOLE NATIONALE
POLYTECHNIQUE
D'ALGER

Département d'Electronique et d'Electrotechnique

PROJET DE FIN D'ETUDES

Thèse d'Ingéniorat

Proposé par :

H.TEDJINI
Docteur Ingénieur

Etudié par :

T. ABDENNEBI
Z. KOUIDER
née YAHIAOUI

JUIN 1981

REMERCIEMENTS

Nous remercions Monsieur B. SANSAL ,responsable de la division "Simulation et Contrôle" au C.S.T.N. où notre projet de fin d'étude a pris forme.

Nous formulons l'expression de notre profonde reconnaissance à Mr. H. TEDJINI pour nous avoir dirigé tout au long de notre travail.

Nous remercions également Mrs Boumghar, Bourkeb, Halimi pour leur contribution; sans oublier Mrs. Chérif , Meguenni et le personnel de la ronéo Mrs. Laazib et Abdi qui se sont occupés respectivement de la frappe et du tirage.

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

§ § § § § § § § § §
§ § § § § § § §
§ § § §
§

D E D I C A C E S

§§§§§§§§§§§§§§§§§§§§§§

A mes parents

A toute ma famille

A tous mes amis

Tounsia

A la mémoire de mon père

A ma mère

A mon mari

A toute ma famille ,à tous mes amis

Zohra

S O M M A I R E

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

§§§§§§§§§§§§§§§§§§§§§§§§§§§§

Le microprocesseur est le développement le plus important dans l'industrie électronique, au cours de l'actuelle décennie. Il a été créé pour remplacer les systèmes en logique cablée et les miniordinateurs dans d'innombrables applications telles que: contrôle et automatisme, instrumentations, informatique...

Pour son utilisation, il peut être relié par des circuits d'inter--face appropriés, à une grande variété de périphériques qui lui fournissent ses signaux d'entréé et sont controlés par ses signaux de sortie.
Il apparait toutefois que le développement de système microprocesseurs a été entravé par une insuffisance trés nette dans le domaine du trai--tement arithmétique.
Dans le cadre du projet "SIMULATION et CONTROLE", la division V du CSTN a adopté pour réaliser un simulateur numérique de réacteur nucléaire la technique du multiprocessing. Les microprocesseurs à 8 bits se sont avérés trop lents, vu la longueur du mot dans le calcul numérique; il a été jugé nècessaire d'augmenter la rapidité de calcul du système par l'association d'une unité arithmétique rapide aux microprocesseurs utilisés.

Le but de notre projet va être donc l'étude d'une unité arithmétique rapide en virgule flottante l'Am 9512 et son couplage au microprocesseur , le MC 6800 , dans une première partie.
La deuxième partie est consacrée à l'utilisation de ce système, dans la résolution des équations différentielles par la méthode de Runge-Kutta.

^^^^^^^^^^^^^^^^^

CHAPITRE I

MICROPROCESSEUR ET CALCUL NUMERIQUE

A - INTRODUCTION AU MICROPROCESSEUR MC 6800

A.1. - Hardware du 6800

A.1.1. - Caractéristiques

Le MC 6800 est un microprocesseur monobloc (ou monolithique) se présentant sous forme d'un boitier DIL (dual in line) de 40 broches constituant ses lignes d'entrées et sorties. Ce up est un circuit intégré LSI de 2ème génération, réalisé en technologie MOS à canal N (N MOS). Il traite des mots de 8 bits et possède une capacité d'adressage de 64 Kmots. Il nécessite une alimentation unique de + 5 V et sa consommation varie autour de 0,25 W. Il travaille à une fréquence de 1 Mhz et est dôté de la possibilité d'arrêt et d'exécution pas à pas d'un programme.

A.1.2. - Organisation interne du MC 6800

Le microprocesseur comprend essentiellement :

a) une unité arithmétique et logique (ALU) : c'est l'ensemble des circuits combinatoires capables d'effectuer les opérations arithmétiques et logiques nécessaires au traitement des informations.

b) une unité de contrôle et décodage : son rôle est de décoder et d'analyser les informations présentes dans le programme et les faire traiter par les organes exécutifs (ALU, ACCU...) au rythme d'une impulsion d'horloge.

c) les registres internes : Ils sont au nombre de 7.

* les accumulateurs A et B : ce sont des registres à 8 bits ou viennent se ranger les données intermédiaires et les résultats nécessaires à l' ALU.

* Le compteur ordinal PC (progam counter) : C'est un registre à 16 bits qui détermine la séquence de l'instruction à exécuter. Il est incrémenté à chaque fois qu'on passe à l'instruction suivante.

* Le registre d'index : C'est un registre à 16 bits destiné à contenir une adresse souvent utilisée dans le mode d'adressage indexé.

* Le pointeur de pile SP (stack pointer) : Ce registre à 16 bits est utilisé lors d'un passage à un sous-programme. Il permet de stocker l'adresse de l'emplacement mémoire du programme initial pour un retour ultérieur.

ORGANISATION INTERNE DU MC6800

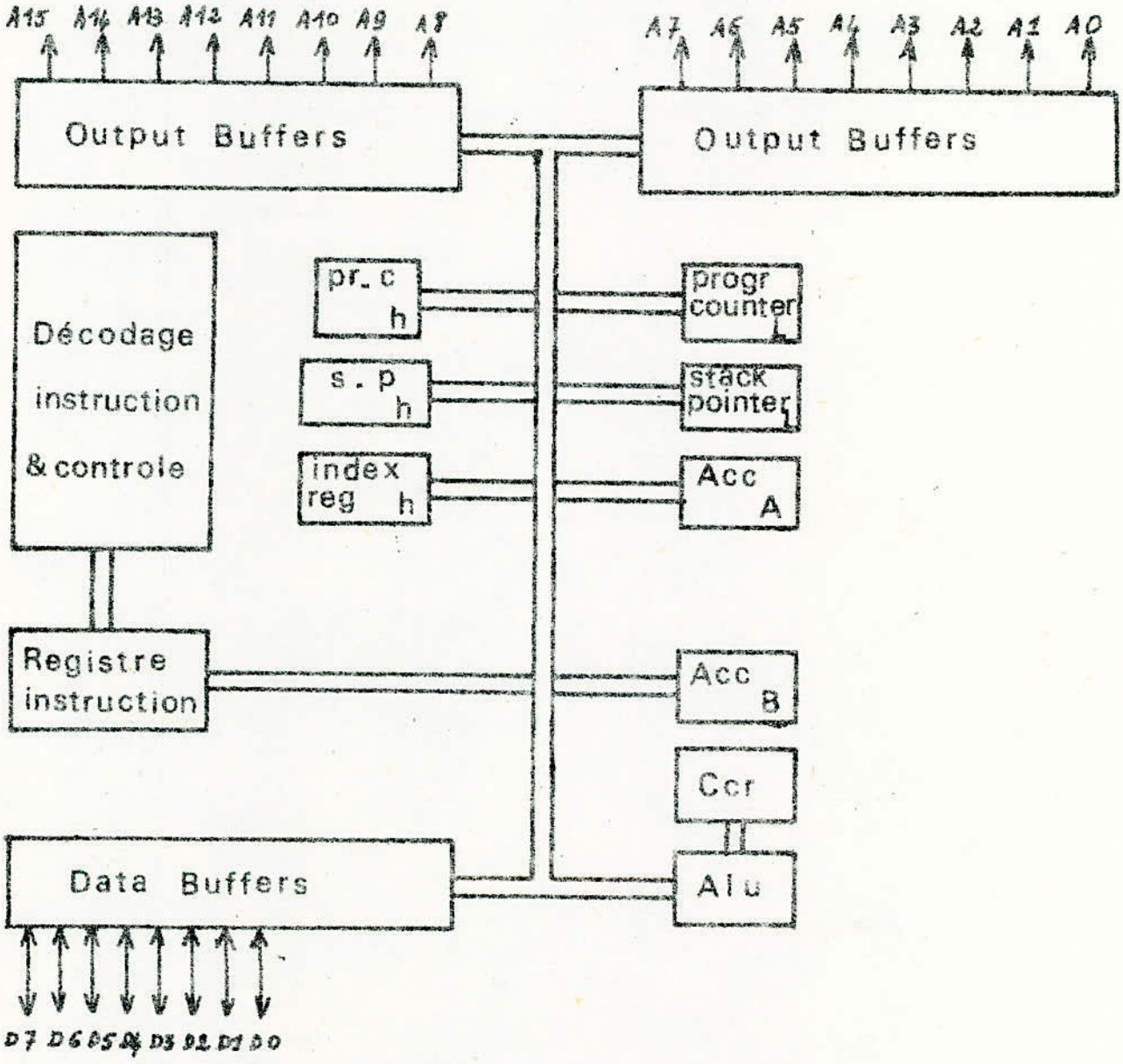


fig 1.a

* Le registre d'instruction (RI) : C'est un registre à 16 bits où vient se ranger chaque instruction de la mémoire pendant le temps nécessaire à son exécution.

* Le registre d'état CCR (code conditions register) : Ce registre de 8 bits permet de disposer de 6 informations utiles à la gestion du programme. 5 informations concernent le résultat des opérations effectuées par l' ALU :

  - N : résultat négatif
  - Z : résultat nul
  - V : dépassement de capacité (overflow)
  - C : retenue (carry)
  - H : demi - retenue (half carry)

  La 6ème information est le bit de masquage de l'interruption $\overline{IRQ}$. Les 2 derniers bits sont constamment égaux à 1.

A.1.3. - Lignes d'entrées et sorties

Pour le fonctionnement correct du MPU, des lignes d'entrées/sorties sont nécessaires pour accomplir des fonctions spécifiques et déterminer l'état du up. Elles se subdivisent en 3 ensembles groupés en bus.

a) Bus de données : D0 - D7. C'est un bus bidirectionnel de 8 bits fonctionnant en logique 3 états; c'est lui qui fait transiter les données, c'est à dire les opérandes, les résultats de calculs, ...

b) Bus d'adresse : A0 - A15 : C'est un bus unidirectionnel de 16 bits permettant d'achever 64 Koctets et pouvant être mis en haute impédance.

c) Bus de contrôle et de commande : Il englobe 3 sortes de signaux :

1°) Signaux de synchronisation : Φ_1 et Φ_2 qui sont les 2 phases de l'horloge. Sur ces signaux sont synchronisés l'adressage et le transfert des données.

2°) Signaux de contrôle :

* R/W (read/write) de lecture, écriture. R/W = 1 ⟹ lecture
* VMA : validation d'adresse mémoire
* DBE : activation du bus de données
* BA : disponibilité du bus adresse du up
* TSC : contrôle 3 états : qui met le bus adresse du up et la ligne R/W en haute impédance.

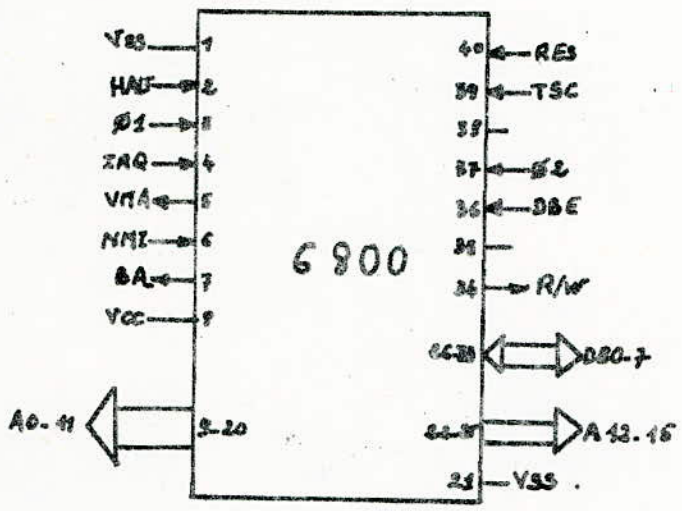


fig 1-b BROCHAGE DU MC 6800.

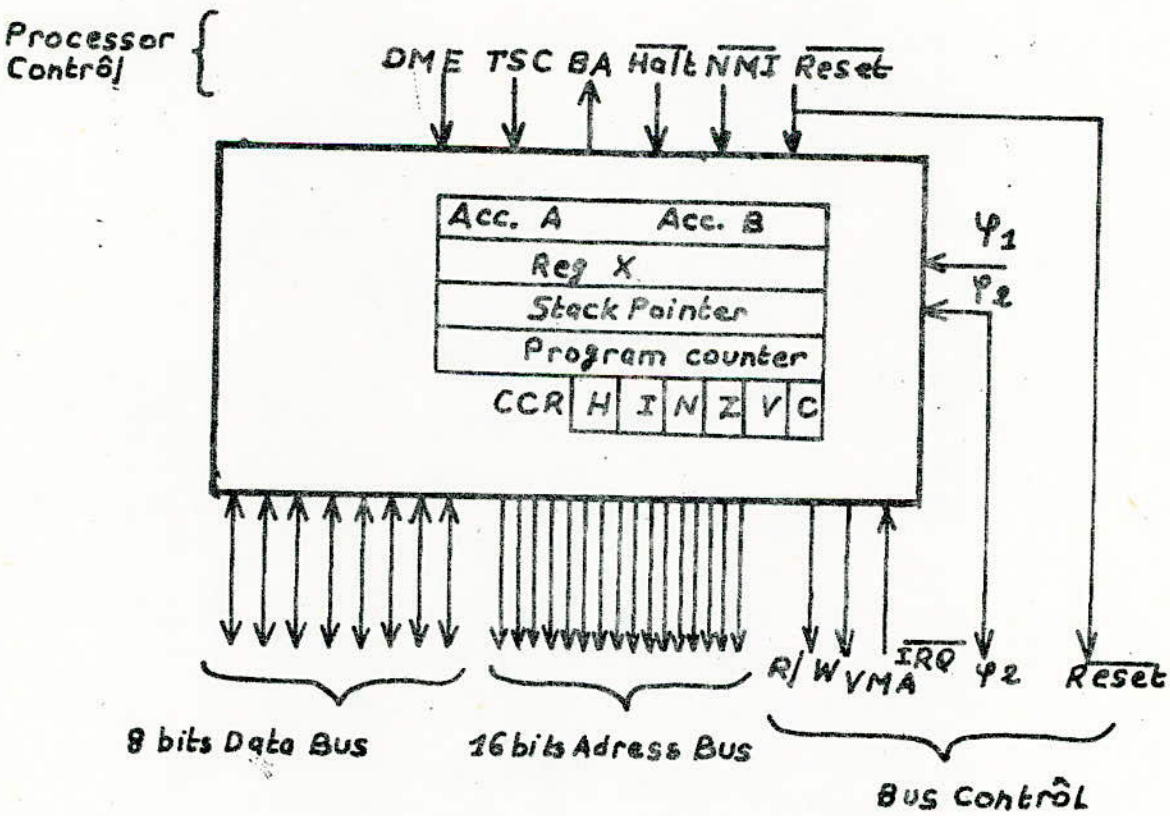


fig 1-c L'UNITE CENTRALE MPU MC6800

3°) Signaux d'interruption :

* $\overline{\text{Halt}}$: arrêt du up pour un temps indéfini
* $\overline{\text{Reset}}$: de remise à zéro du up
* $\overline{\text{IRQ}}$: de demande d'interruption masquable
* $\overline{\text{NMI}}$: d'interruption non masquable.

A.2. - Le logiciel

A.2.1. - Jeux d'instruction

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

- Arithmétiques (binaires et décimales)
- Logiques (et, ou, ...)
- Décalages (à droite ou à gauche)
- Chargement (de certains registres)
- Stockage (en mémoire ou autres)
- Branchements conditionnels ou inconditionnels (JUMP, BEQ, BPL, ...)
- Instructions relatives aux interruptions (SWI, RTS, ...)

La réalisation de toute instruction se décompose en 2 temps successifs :

* La recherche de l'instruction, qui consiste à lire en mémoire l'instruction à exécuter
* L'exécution de cette instruction. Ce temps peut être plus ou moins long suivant la complexité de l'instruction. Le minimum de cycles mémoires pour une instruction est de 2 us (LDA A), et le maximum de cycles mémoires est 12 us (SWI)

A.2.2. - Traduction du programme

Le programme est rédigé de la façon la plus courante en utilisant un langage ignoré par le up, qui est le langage mnémonique. La machine électronique ne comprend que le numérique, on passe donc par une étape de traduction qui est l'assemblage.

L'assemblage est un dictionnaire utilisé pour traduire le code mnémonique en code machine; c'est un programme enregistré sur un support approprié (bande perforée ou magnétique ou disque souple) en lagage machine.

A.2.3. - L'adressage

Lorsqu'une instruction fait référence à un opérande, elle peut repérer celui-ci en mémoire de différentes façons appelées modes d'adressage. Par un mode judicieux, il est possible de réduire la longueur du programme, la capacité et le temps d'exécution. Le MC 6800 possède 7 modes d'adressage dont les plus utilisés sont :

A.2.3.1. - Adressage immédiat :

L'opérande se trouve dans le 2ème ou 3ème Octet de l'instruction selon qu'on s'adresse aux accumulateurs ou aux registres.

A.2.3.2. - Adressage direct :

C'est le mode d'adressage le plus utilisé. Dans le 2ème Octet se trouve l'adresse de l'opérande. On peut adresser les locations de 0 à 255 (instructions à 2 Octets).

A.2.3.3. - Adressage indéxé :

L'adresse contenue dans le second octet du bus d'adresse est ajoutée à une valeur particulière du registre d'index. L'adresse résultante est utilisée pour accéder à la position mémoire désirée.

A.2.3.4. - Adressage étendu

L'adresse est formée par le 2ème et 3ème octet venant après l'instruction. Ce mode d'adressage permet de balayer toutes les mémoires de 0000 à FFFF.

Les autres modes : le mode simplicite, le mode relatif et le mode indirect sont rarement utilisés.

A.2.4. - Structure de la pile

Le MC 6800 dispose d'une pile de registres volatifs, qui permet de mémoriser les informations et de les utiliser selon le mode LIFO (last in, firts out) : dernier entré, premier sorti. Le pointeur de pile permet d'adresser les registres de la pile. Son organisation est schématisée par la fig.

1°) Gestion de la pile :

Le registre printeur de pile SP permet la gestion de la pile. Il contient l'adresse de la 1ère 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 mémoir, les adresses sont décroissantes vers le sommet de la pile, le registre est donc décrémenté à chaque entrée d'information et incrémenté à chaque sortie. Des instructions de stockage en pile (PUSH) ou de sortie en pile (PULL) permettent son utilisation; mais cette structure est aussi utilisée pour les appels de retours de sous - programmes.

2°) SAUVEGARDE EN PILE

Une pile peut être utilisée pour sauvegarder l'adresse de retour d'un sous-programme. Lorsqu'un 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.

B - FAMILLE DU 6800

La famille M 6800 comprend le MPU MC 6800 et des mémoires dynamiques et statiques RAM MC 6810, ROM MC 6830.

La nécessité d'utiliser des fonctions spéciales pour le transfert des données a conduit à la conception des circuits d'interface tel que PIA MC 6820, ACIA MC 6850. Ces circuits sont entièrement programmables à travers le bus, et leur statut en temps réel estt disponible sur le bus. Le MPU adresse tous ces circuits comme des locations mémoires d'où la simplification de l'interface entre les mémoires et les périphériques, et l'élimination des instructions spéciales d'entrée/sortie.

B. 1. - LES MEMOIRES :

* B.1.1. Les mémoires RAM (ramdom accès mémory)
  Dans ces mémoires, ont peut y lire ou écrire à volonté.

* B.1.2. Les mémoires ROM (read only mémory)
  C'est des mémoires mortes à lecture seulement. Les instructions sont stockées au stade de la fabrication. Le contenu est immuable et ne peut être effacé, sauf dans le cas d'une destruction de la mémoire.

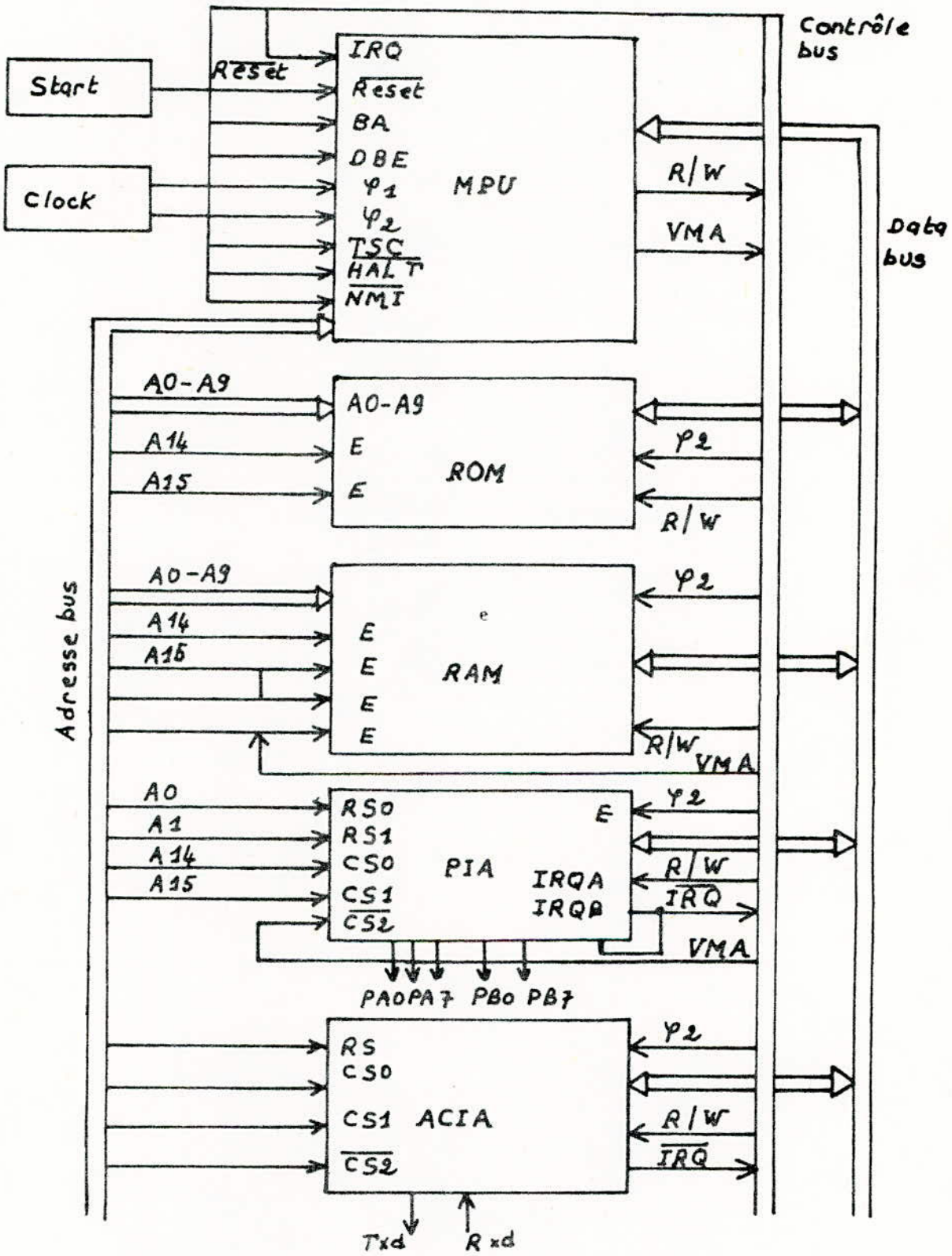
* B.1.3. Les mémoires PROM (ROM Programmable)

Ce sont des mémoires ROM programmables d'une façon définitive par l'utilisation au moyen d'un "programmeur de PROM".

B.2. - LES INTERFACES

Les microprocesseurs ne peuvent dialoguer directement avec les unités périphériques; pour celà il est nécessaire de prévoir des étages tampons appelés interfaces. Ces dispositifs sont de 2 sortes :

fig 1-d SYSTEME MINIMUM M 6800



Les interfaces parallèles qui assurent une transmission en parallèle des informations (PIA) et les interfaces série (ACIA) qui transmettent les données en série bit par bit.

B. 3. - STRUCTURE ELEMENTAIRE D'UN SYSTEME M 6800

Le système minimum comprend une ROM, une RAM, un ou 2 adaptateurs d'interface PIA, ACIA et les circuits d'horloge.

Le MPU commuique avec les mémoires et les interfaces par l'intermédiaire de 3 bus : Le bus de données et le bus de contrôle bidirectionnels, et le bus adresse unidirectionnel. En effet non seulement le bus adresse sélectionne la mémoire, mais il devient un outil pour sélectionner les circuits d'entrée/sortie.

Par l'intermédiaire des bus de données, de contrôle et d'adresse, l'interface entrée/sortie est considéré comme une location mémoire. Comme conséquence à celà, l'utilisateur peut converser avec les I/O en utilisant n'importe quelle instruction faisant référence à la mémoire, en sélectionnant le périphérique désiré à l'aide d'une adresse mémoire.

Le système élémentaire contient la ROM dans laquelle est implanté le programme de service. La RAM est utilisée pour l'exécution du programme. Le PIA sert au couplage des périphériques avec le MPU. Ces périphériques peuvent être : la télétype, le lecteur de cartes ou l'imprimante, ... L'ACIA sert à communiquer avec des périphériques en mode asynchrone.

C - CALCUL NUMERIQUE PAR LE M 6800

Les algorithmes d'addition, de multiplication et de division en virgule flottante 24 bits pour le système M 6800, possèdent respective--ment un temps d'exécution de 586, 680 et 736 cycles MPU.

Si un cycle MPU égale une microseconde (cas du MC 6800), nous constatons que l'exécution des opérations d'addition, multiplication et division est trop lente.

Lorsqu'il s'agit d'une utilisation intensive de ces opérations, dans les problèmes de résolution des équations différentielles par exemple, le temps d'exécution influe sur la rapidité de traitement du problème.

L'idéal pour le MPU serait que l'opération additionner, multiplier ou diviser occupent les dimensions de quelques instructions. Nous adoptons donc l'utilisation d'une unité arithmétique rapide en virgule

flottante : l'AM 9512, qui va résoudre le problème de lenteur du microprocesseur et augmenter la précision des calculs.

CHAPITRE II

L'OPERATEUR EN VIRGULE FLOTTANTE AM 9512

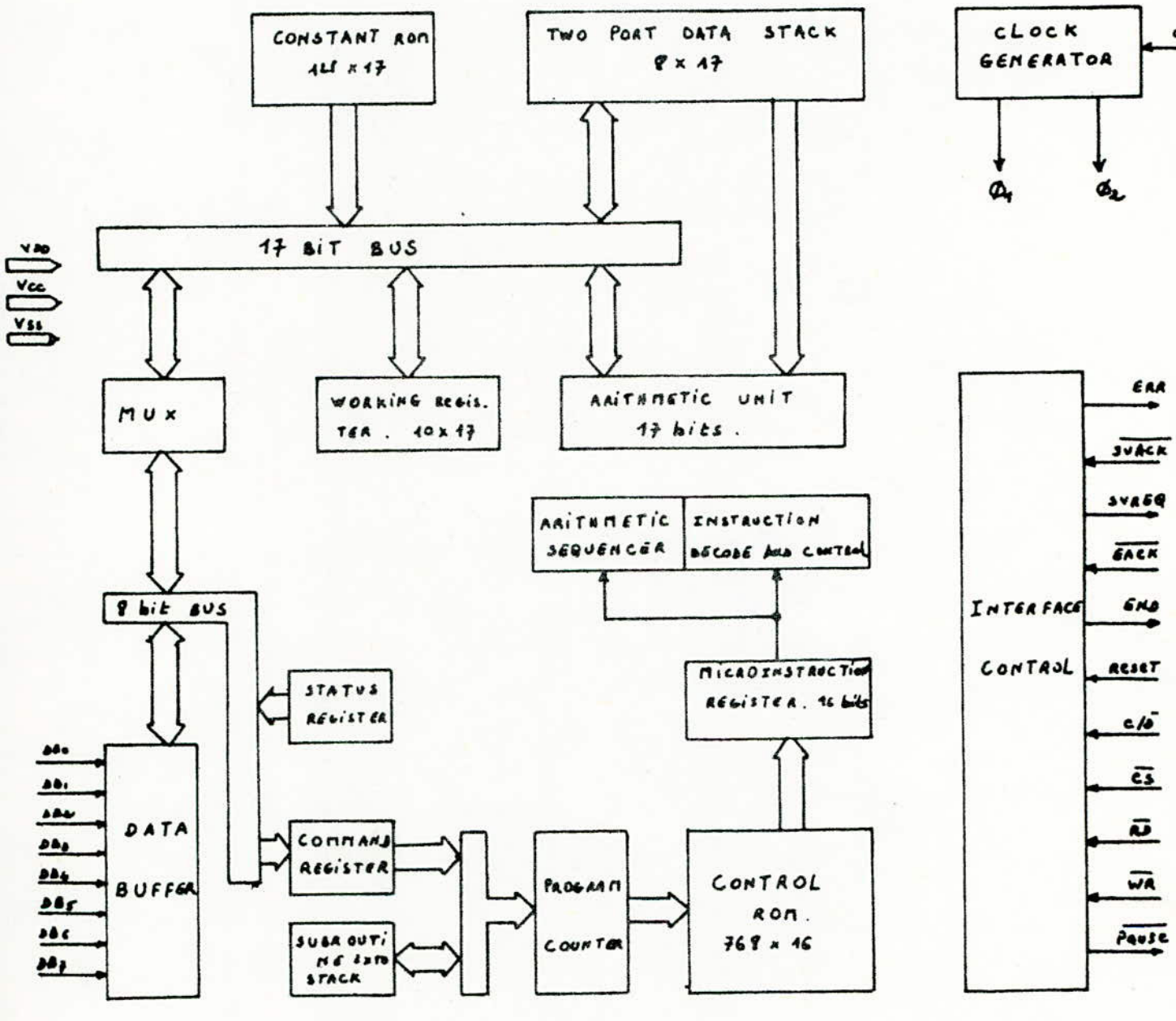


fig 2-a **Bloc diagramme de l'Am 9512**

II A. - PRESENTATION DE L'UNITE ARITHMETIQUE AM 9512

2. 1. - Introduction

Malgré l'apparition récente des 16 bits, les up MOS 8 bits consituent , encore ajourd'hui, les standards de l'industrie. Le services rendus dans les domaines aussi divers que le test, l'instrumentation, le contrôle de processus ne sont plus à vanter. un très gros effort a été réalisé par les fabricants pour faciliter la mise en oeuvre et le développement des programmes par les utilisateurs. De nombreux circuits périphériques d'entrée/sortie ont, en outre, permis de connecter les up à des lignes série, synchrones ou asynchrones, ou à des bus parallèles standards. Il apparait, toutefois, que le développement de systèmes microordinateurs de hautes performances a, jusqu'à présent, été entravé par 2 facteurs :

* Une insuffisance très nette dans le domaine du traitement arithmétique,
* Un manque d'efficacité au niveau du système.

Les circuits périphériques spécialisés pour up 8 bits dont l'AM 9512, permettent de combler ces différentes lacunes.

2.2. - Caractéristiques générales de l'unité AM 9512 :

L' AM 9512 est un monochip LSI, en technologie MOS canal N à grille de silicium, qui se présente dans un boîtier de 24 broches. Il est destiné à accroître sensiblement la capacité de calcul arithmétique des up 8 bits. Son caractère universel permet de le connecter indifféremment à n'importe quel up. Il travaille à une fréquence maximum d'horloge de 3 Mhz et demande 2 tentions d'alimentation + 5 V et + 12 V

L'unité arithmétique AM 9512 exécute les 4 opérations fondamentales (addition, soustraction, division, multiplication) sur des nombres représentés en code binaire, en virgule flottante, en 2 formats : format simple précision 32 bits et format double précision 64 bits.

2.3. - Broches d'interface

En dehors du bus de données DB 0 - DB7 et du signal d'horloge CLK, l'AM 9512 comprend les broches d'interface suivantes :

a) Lignes d'entrée :

- Reselt : sert à initialiser le composant. L'activation de cette ligne a pour conséquences : l'arrêt de toute opération en cours, le RAZ du registre d'état, le positionnement du composant dans un état d'attente.
- $C/\overline{D}$: command/data : Il indique le type de transfert effectué.
  la fig donne le codage des transferts.

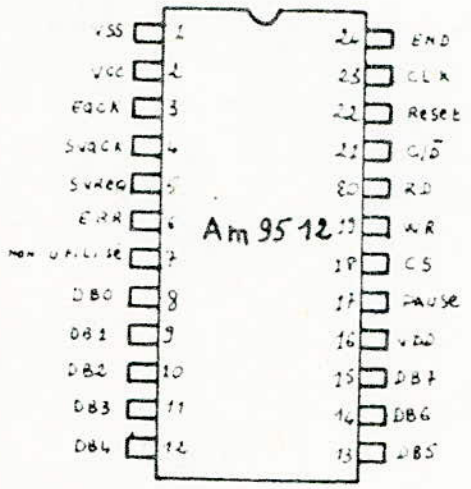


fig-2 b **Brochage de L'AM 9512**

C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	FONCTION
L	H	L	entrée d'un byte donnée dans la pile
L	L	H	Sortie d'un byte donnée de la pile
H	H	L	entrée d'une commande
H	L	H	lecture du registre d'état
X	L	L	indefini .

L : Low

H : HIGH

x : indefini

commande des transferts

- $\overline{CS}$: Chip select (sélection du boîtier) : valide les échanges d'informations avec l'extérieur.

b) Lignes de sortie :

- $\overline{END}$: End exécution (fin d'exécution). Indique que l'exécution de la dernière opération demandée est terminée.
- $\overline{SVREQ}$: Service request (demande de service) : Indique qu'un service particulier post opération a été demandé.
- $\overline{PAUSE}$: pause. Cette ligne est activée dans les conditions suivantes : une entrée est demandée, alors qu'une opération est en cours d'exécution; une entrée de données est demandée alors que la donnée précédente n'est pas encore rangée en pile; une sortie (de données ou de mot d'état) est demandée, et l'information n'est pas encore valide sur le bus. Elle permet donc de synchroniser avec le up maître par l'intermédiaire d'une procèdure d'attente.
- $\overline{ERR}$: Error (erreur). L'activation de cette sortie indique une erreur de condition dans l'exécution de la commande en cours. Les erreurs de condition sont : une division par zéro, un overflow de l'exposant ou un underflow de l'exposant. La sortie ERR est mise à zéro après lecture du registre d'état ou une reset.

II B. - DESCRIPTION FONCTIONNELLE :

Le synoptique du composant est donné par la fig. a. La totalité des transferts (données opérandes, données résultat, mot de commande ou mot d'état) s'effectuent par l'intermédiaire d'un bus bidirectionnel de 8 bits (DB0 - DB7). Ces signaux entrent à travers des buffers de données. La structure interne est celle d'un processeur 17 bits parallèles; sur ce bus interne de 17 bits sont connectés :

- l'unité arithmétique
- une pile où sont stockés les opérandes et les résultats de calcul. Cette pile est organisée en 8 mots de 17 bits, avec le dernier entré, premier sorti (LIFO)
- les registres de travail au nombre de 10, utilisés pour le stockage des valeurs intermédiaires pendant l'exécution des opérations.
- une mémoire ROM de 128 constantes.

Les opérations de l'unité AM 9512 sont contrôlées par le microprogramme contenu dans une mémoire CONTROL ROM. Le program counter

MOT DE COMMANDE 7 6 5 4 3 2 1 0	MNEMONIQUE	FONCTION RÉALISEE	
x 0 0 0 0 0 0 1	SADD	(A) + (B) ⟶ B SP	
x 0 0 0 0 0 1 0	SSUB	(B) - (A) ⟶ B SP	POP STACK
x 0 0 0 0 0 1 1	SMUL	(A) × (B) = P ⟶ B SP	
x 0 0 0 0 1 0 0	SDIV	(B) : (A) = Q ⟶ B SP	
x 0 0 0 0 1 0 1	CHSS	INVERSION SIGNE (A) SP	
x 0 0 0 0 1 1 0	PTOS	PUSH STACK (A) ⟶ A SP	
x 0 0 0 0 1 1 1	POPS	POP STACK SP	
x 0 0 0 1 0 0 0	XCHS	ECHANGE entre (A) et (B) SP	
x 0 1 0 1 1 0 1	CHSD	INVERSION SIGNE (A) DP	
x 0 1 0 1 1 1 0	PTOD	PUSH STACK (A) ⟶ A DP	
x 0 1 0 1 1 1 1	POPD	POP STACK DP	
x 0 0 0 0 0 0 0	CLR	REGISTRE D'ETAT = 0	
x 0 1 0 1 0 0 1	DADD	(A) + (B) ⟶ B DP	
x 0 1 0 1 0 1 0	DSUB	(B) - (A) ⟶ B DP	POP STACK
x 0 1 0 1 0 1 1	DMUL	(A) × (B) = P ⟶ B DP	
x 0 1 0 1 1 0 0	DDIV	(B) : (A) = Q ⟶ B DP	

fig-e FONCTIONS RÉALISEES par l'AM 9512

fournit les adresses des microinstructions et peut être partiellement chargé par le registre de commande. Une pile de soubroutine, connectée au program counter stocke les adresses de retour, lors d'un appel de soubroutine. Le chargement du registre de commande fournit l'adresse de départ dans la mémoire du microprogramme. Un registre d'état rassemble un certain nombre d'indications, indiquant l'état du composant à la suite d'un calcul.

2.1. - Fonctionnement de la pile de données :

Les opérandes et les résultats de calcul sont rangés dans une pile LIFO organisée en 4 niveaux de 32 bits ou 2 niveaux de 64 bits, en fonction du format de travail. Avant le début d'une opération, les opérandes sont rangés dans la pile, octet de poids faible en tête. L'opération elle même s'effectue implicitement sur les opérandes situés au sommet de la pile TOS (Top of the stack) et à lemplacement suivant NOS (Next of the stack). A la fin de l'opération, le résultat se retrouve au sommet de la pile, après une éventuelle opération de rotation (Pop stack) implicitement contenue dans le code opératoire. L'acquisition du résultat par le up s'effectue par lecture de la pile, octet de poids fort en tête, suivant le format choisi 4 bytes pour la simple précision et 8 bytes pour la double précision.

2.2. - Registre de commande :

Le registre de commande (voir fig. d - 2) est chargé par l'octet de commande spécifiant le type de traitement à effectuer. Le format de commande est de 8 bits, le bit 7 est le service réquest SUREQ = SR, indique si la commande est assortie ou non d'une demande de service; les 6 premiers bits indiquent le code opération. Les mots de commande de l'AM 9512 se divisent en 3 catégories : simple précision, double précision et manipulation de données. Les 4 opérations de base (addition, soustraction, multiplication, division) se font en virgule flottante en simple ou double précision.

Le tableau (fig. e) représente les fonctions réalisées par l'AM 9512 les notations suivantes ont été adoptées

- A = TOS représente le sommet de la pile et (A) son contenu
- B' = NOS représente l'emplacement suivant et (B) son contenu
- Push stack : indique une opération d'enfoncement dans la pile (avec perte de l'opérande situé à la base de la pile)
- Pop stack : indique une opération d'extraction de la pile.

REGISTRE D'ETAT

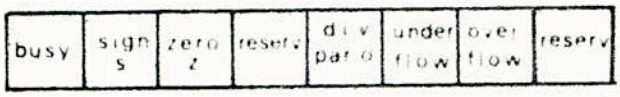


fig d-1

REGISTRE de COMMANDE

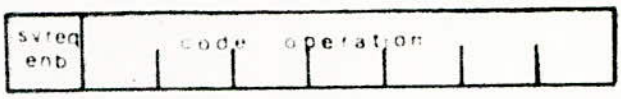


fig d-2

FORMAT des DONNÉES

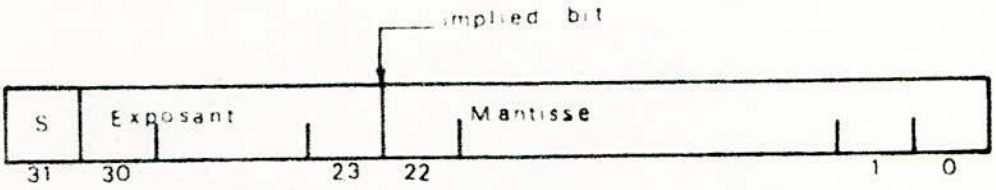


fig d-3 Simple précision

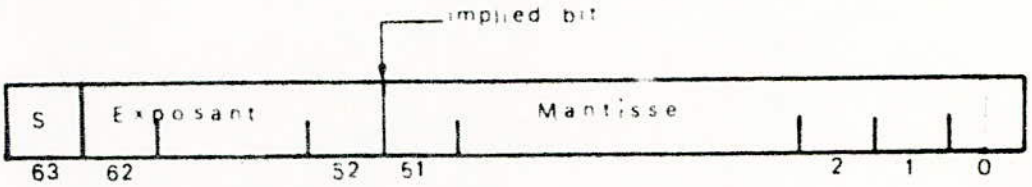


fig d-4 Double précision

TEMPS D'EXECUTION DES OPÉRATIONS en cycles

	Min	Typ	Max
ADD	58	220	512
SOUST	56	220	512
MULT	192	220	254
DIV	228	240	264

fig d-5 SIMPLE PRECISION

	Min	Typ	Max
ADD	578	1200	3100
SOUST	578	1200	3100
MULT	1720	1770	1860
DIV	4560	4920	5120

DOUBLE PRECISION

2.3. - Registre d'état

Le résultat de la dernière opération effectuée par l'AM 9512 affecte le contenu du registre d'état. Celui-ci comprend 8 bits, dont 6 indicateurs. (voir fig. d - 1)

Bit 0 : réservé
Bit 1 : dépassement de capacité de l'exposant (overflow)
Bit 2 : sous dépassement de capacité de l'exposant (underflow)
Bit 3 : division par 0
Bit 4 : réservé
Bit 5 : zéro, indique si le TOS est nul
Bit 6 : signe du TOS
Bit 7 : busy, quand il est à 1, indique qu'une opération est en cours d'exécution. Les différents bits du registre d'état sont validés quand le bit busy est égal à 0.

II C - REPRESENTATION DES NOMBRES

L' AM 9512 travaille sur des nombres binaires représentés en virgule flottante. Dans ce type de représentation, un nombre N s'écrit de la façon suivante : $N = M.2^E$ où M est la mantisse et E l'exposant. Le nombre de bits de la mantisse fixe la précision limite des calculs et le nombre de bits de l'exposant détermine l'étendue du codage. La forme dite normalisée, permet de conserver le maximum de bits significatifs.

C.1. - Format simple précision

Le format simple utilisé ici, est de 32 bits. voir fig d - 3

- le bit 31 = s représente le signe dela mantisse.
- bits 23 à 30 : ces 8 bits représentent l'exposant codé, qui est égal à sa valeur augmentée du bias exposant.
  bias exposant = $2^7 - 1 = 127$ et l'exposant codé égal à $E + (2^7 - 1)$
- bits 0 à 22 : ces 23 bits représentent l'amplitude de la mantisse en binaire, la virgule se trouvant à gauche de la mantisse. Une conséquence de la normalisation fait que le 1er bit après la virguleest toujours positionné à 1. Ce bit appelé implied bit, est à chaque fois sous-entendu; donc le premier bit significatif est le bit 22 et la valeur absolue de la mantisse est comprise entre 1 et 2.

Pendant l'exécution interne d'une opération, l'implied bit est restauré, et à la fin de l'opération, le résultat est normalisé et

l'implied bit sous entendu. La virgule de la mantisse se situe entre l'implied bit et le bit 22.

On représente le nombre codé N par la notation :

$$N = (-1)^s \; 2^{E - (2^7 - 1)} \; (1, M)$$

($E - (2^7 - 1)$: bias ; (1, M) : virgule)

Exemple de conversion :

Exemple 1 : N = 0 10000011 11000000000000000000000

signe — exposant — mantisse

exposant codé = 10000011

exposant actuel = 10000011 - 01111111 = 00000100 = 4 (10)

mantisse = 1.11000000000000000000000

$$= 1 + \frac{1}{2} + \frac{1}{4} = 1{,}75 \; (10)$$

le nombre N en décimal = $2^4 \times 1{,}75 = 28 \; (10)$

Exemple 2 : N = 1 01111010 01100000000000000000000

signe — exposant — mantisse

exposant codé = 01111010

exposant actuel = 01111010 - 01111111 = 11111011 = -5 (10)

mantisse = 1.01100000000000000000000

$$= 1 + \frac{1}{4} + \frac{1}{8} = 1{,}375 \; (10)$$

N en décimal = $-2^{-5} \times 1{,}375 = -0{,}4296875 \; (10)$

C. 2. - Format double précision :

Pour ce format le nombre de bits est de 64 (fig. d - 4)

- le bit 63 = s est le signe de la mantisse
- bits 52 à 62 : ces 11 bits représentent l'exposant codé, dont la valeur est supérieure à la valeur réelle de celle du bias exposant. Ici le bias est égal à $2^{10} - 1 = 1023$ donc l'exposant codé est $E + (2^{10} - 1)$
- Bits 0 à 51 : Ces 52 bits représentent la mantisse. Le bit 51 est précédé de l'implied bit égal à 1 et sous-entendu. Un nombre en double précision est représenté par :

$$N = (-1)^s \; 2^{E - (2^{10} - 1)} \; (1, M)$$

($E - (2^{10} - 1)$: exposant codé ; (1, M) : virgule)

COMMANDE	TOS	NOS	RESULTAT	PÉRIODES
SADD	3F800000	3F800000	40000000	58
SSUB	3F800000	3F800000	00000000	56
SDIV	40400000	3FC00000	40900000	198
SMUL	40000000	3F800000	3F000000	231
CHSS	3F800000	—	BF800000	10
PTOS	3F800000	—	—	16
POPS	3F800000	—	—	14
XCHS	3F800000	40000000	—	26
CHSD	3FF0000000000000	—	BFF0000000000000	24
PTOD	3FF0000000000000	—	—	40
POPD	3FF0000000000000	—	—	26
CLR	3FF0000000000000	—	—	4
DADD	3FF0000000000000	8000000000000000	3FF00000A0000000	578
DSUB	3FF0000000000000	8000000000000000	3FF00000A0000000	578
DMUL	BFF8000000000000	3FF8000000000000	C002000000000000	1748
DDIV	BFF8000000000000	3FF8000000000000	BFF0000000000000	4560

TOS, NOS et RESUL en HEXADECIMAL
PÉRIODES en DECIMAL

Execution de certains exemples

II D - ALGORITHMES DES OPERATIONS EN VIRGULE FLOTTANTE :

Dès leur entrée dans le CI, les opérandes sont rangés dans la pile, le signe et l'exposant occupent un et 8 bits pour la simple précision, et la mantisse 24 bits avec le MSB égal à 1. Pour les différents algorithmes on utilisera les abréviations suivantes :

MSB : bit le plus significatif du nombre
sign. : signe du résultat
exp. : exposant du résultat
MAN. : mantisse du résultat
sign. (TOS) : signe du sommet de la pile
exp. (TOS) : exposant du sommet de la pile
MAN. (TOS) : mantisse du sommet de la pile
sign. (NOS) : signe du nombre dans l'emplacement suivant de la pile
exp. (NOS) : exposant du nombre dans l'emplacement suivant de la pile
MAN. (TOS) : mantisse du nombre dans l'emplacement suivant de la pile.

D. 1. - ADDITION, SOUSTRACTION EN VF :

L'addition et la soustraction utilisent essentiellement le même algorithme. La seule différence est que pour la soustraction, le signe de TOS est changé avant de poursuivre l'algorithme d'addition (fig e - 1) Celui - ci se résume à :

a - ranger TOS et NOS
b - l'exposant de TOS est comparé à l'exposant de NOS.
c - si les exposants sont égaux, passer à f
d - décalage à droite de la mantisse du nombre dont l'exposant est le le plus petit
e - incrémenter l'exposant le plus petit et revenir en b
f - mettre le signe du résultat égal au signe du nombre le plus grand
g - mettre l'exposant du résultat égal à l'exposant du nombre le plus grand
h - si les signes des 2 nombres sont différents, aller en m
i - addition des mantisses des 2 nombres
j - décalade d'une position à droite de la mantisse du résultat et incrémentation de son exposant
k - si le MSB de l'exposant du résultat passe de 1 à 0, à la suite de l'incrémentation, le bit d'état overflow est mis à 1
l - arrondissement de la mantisse si nécessaire et sortie du résultat
m - soustraction des mantisses des 2 nombres
n - décalage à gauche de la mantisse du résultat et décrémentation de son exposant

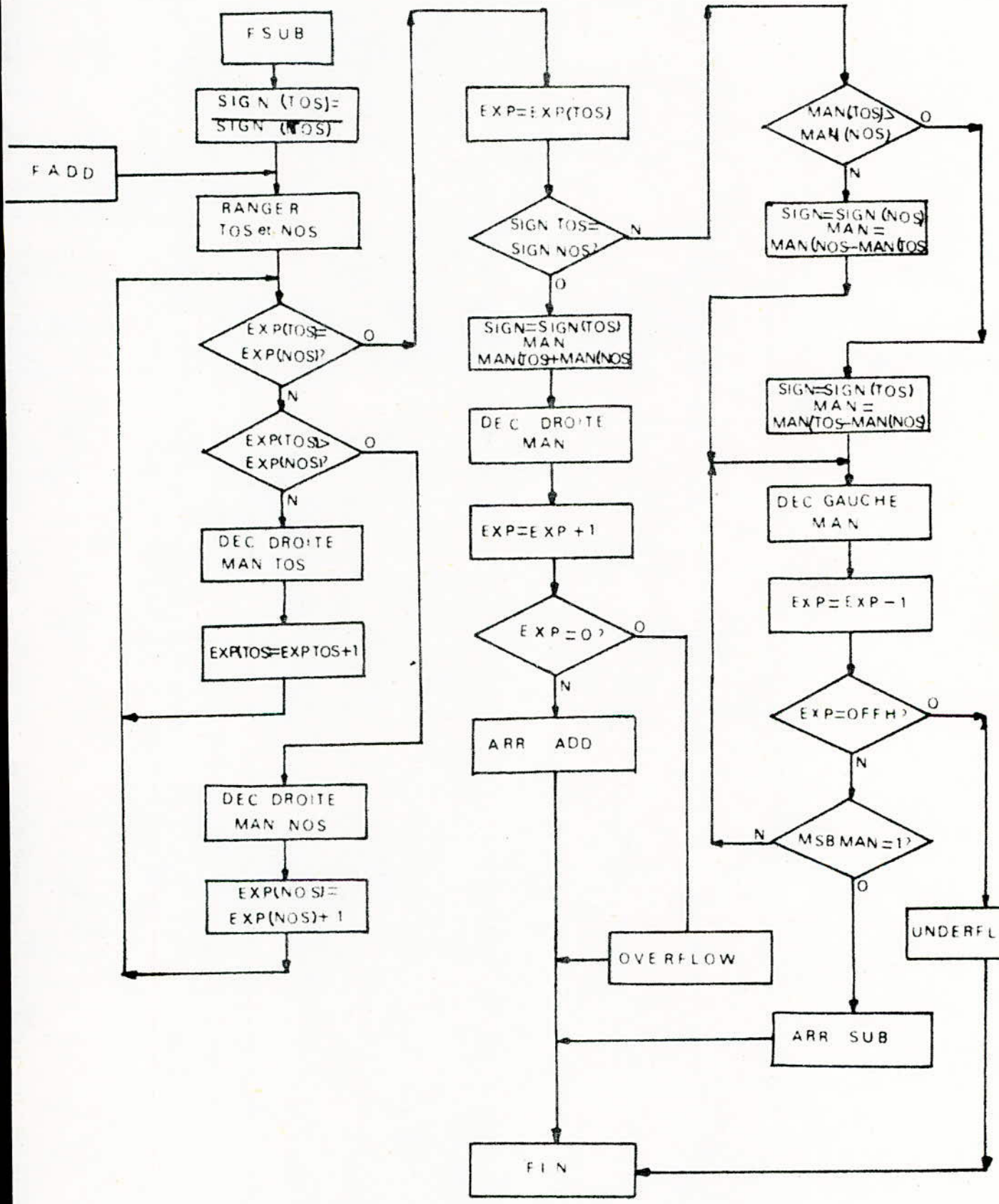


fig e-1 ADDITION - SOUSTRACTION VF

o - si le MSB de l'exposant passe de 0 à 1 à la suite de la décrémentation mise à 1 du bit d'état underflow et sortie du résultat

p - si le MSB de la mantisse du résultat est égal à 0, aller en n

q - arrondissement de la mantisse si nécessaire et sortie du résultat

D. 2. MULTIPLICATION EN VF

La multiplication en VF se base sur l'addition des exposants et la multiplication des mantisses (voir fig. e - 2)

a - tester si TOS ou NOS est égal à 1

b - si TOS ou NOS est nul, sortie du résultat égal à 0

c - ranger TOS et NOS

d - conversion des exposants de TOS et NOS à la forme unbiased.

exp. (TOS) = exp. (TOS) - $127_{(10)}$

exp. (NOS) = exp. (NOS) - $127_{(10)}$

e - exposant du résultat égal à la somme des exposants

exp. = exp. (TOS) + exp. (NOS)

f - si le MSB de l'exposant de TOS et le MSB de l'exposant de NOS sont égaux à 0, avec le MSB de l'exposant du résultat égal à 1, mise à 1 du bit d'overflow et sortie du résultat

g - si le MSB de l'exposant de TOS et le MSB de l'exposant de NOS sont égaux à 1, et si le MSB de l'exposant du résultat égal à 0, mise à 1 du bit d'underflow et sortie du résultat

h - reconversion de l'exposant du résultat à la forme biased

exp. = exp. + $127_{(10)}$

i - si les signes de TOS et NOS sont égaux, mettre le signe du résultat égal à 0, sinon le mettre à 1

j - multiplication des mantisses des 2 nombres

k - si le MSB de la mantisse du résultat est égal à 1, décalage de celle - ci à droite d'une position et incrémentation de l'exposant du résultat

l - si le MSB de l'exposant du résultat passe de 1 à 0 à la suite de l'incrémentation, mise à 1 du bit d'overflow

m - arrondissement de la mantisse si nécessaire et sortie du résultat

D. 3 - DIVISION EN VF

Celle - ci se résume à la division des mantisses et la soustraction des exposants (fig e - 3)

a - si TOS et nul, mise à 1 du bit d'état division par zéro et sortie du résultat

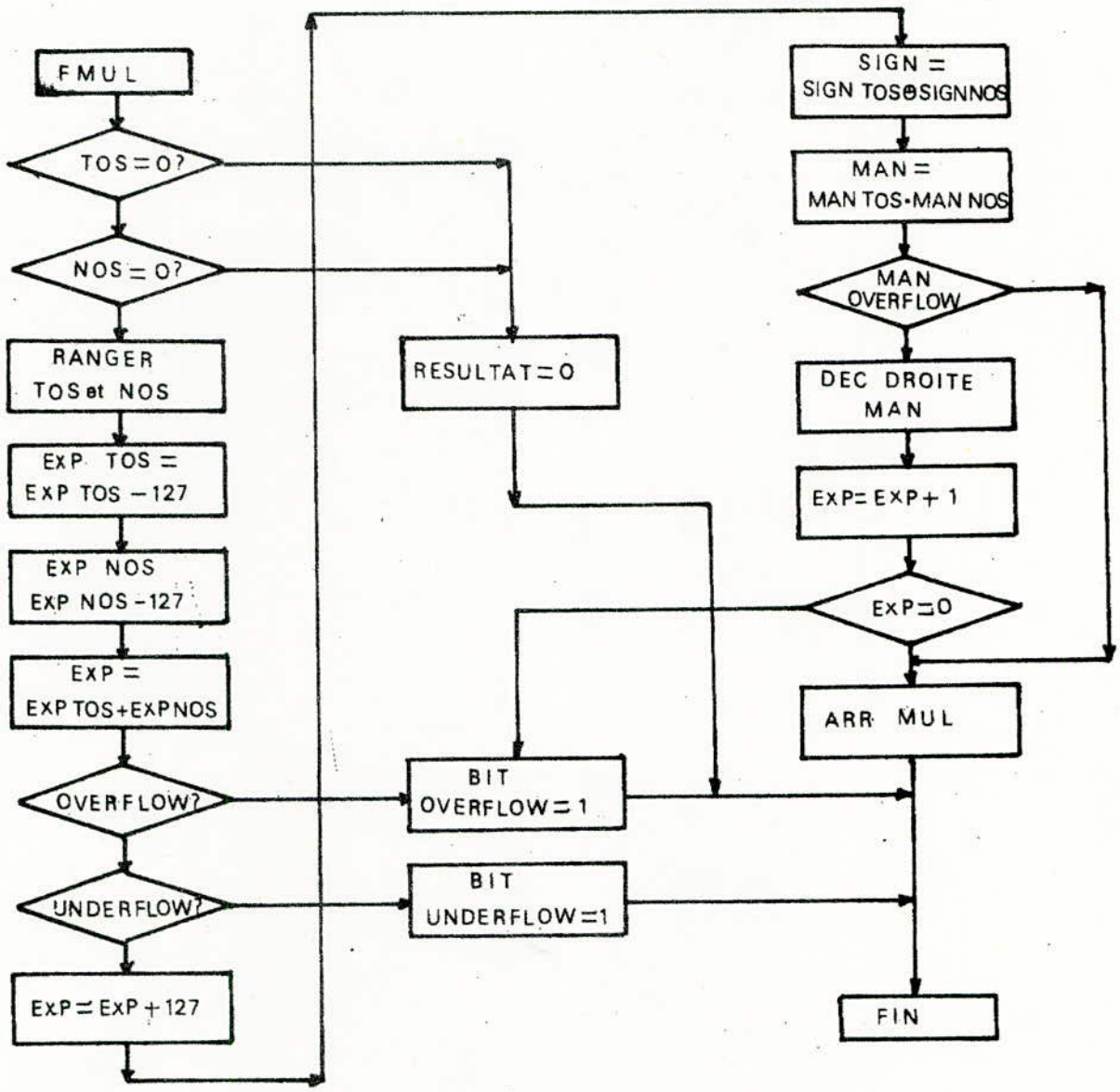


fig e-2 MULTIPLICATION EN VIRGULE FLOTTANTE

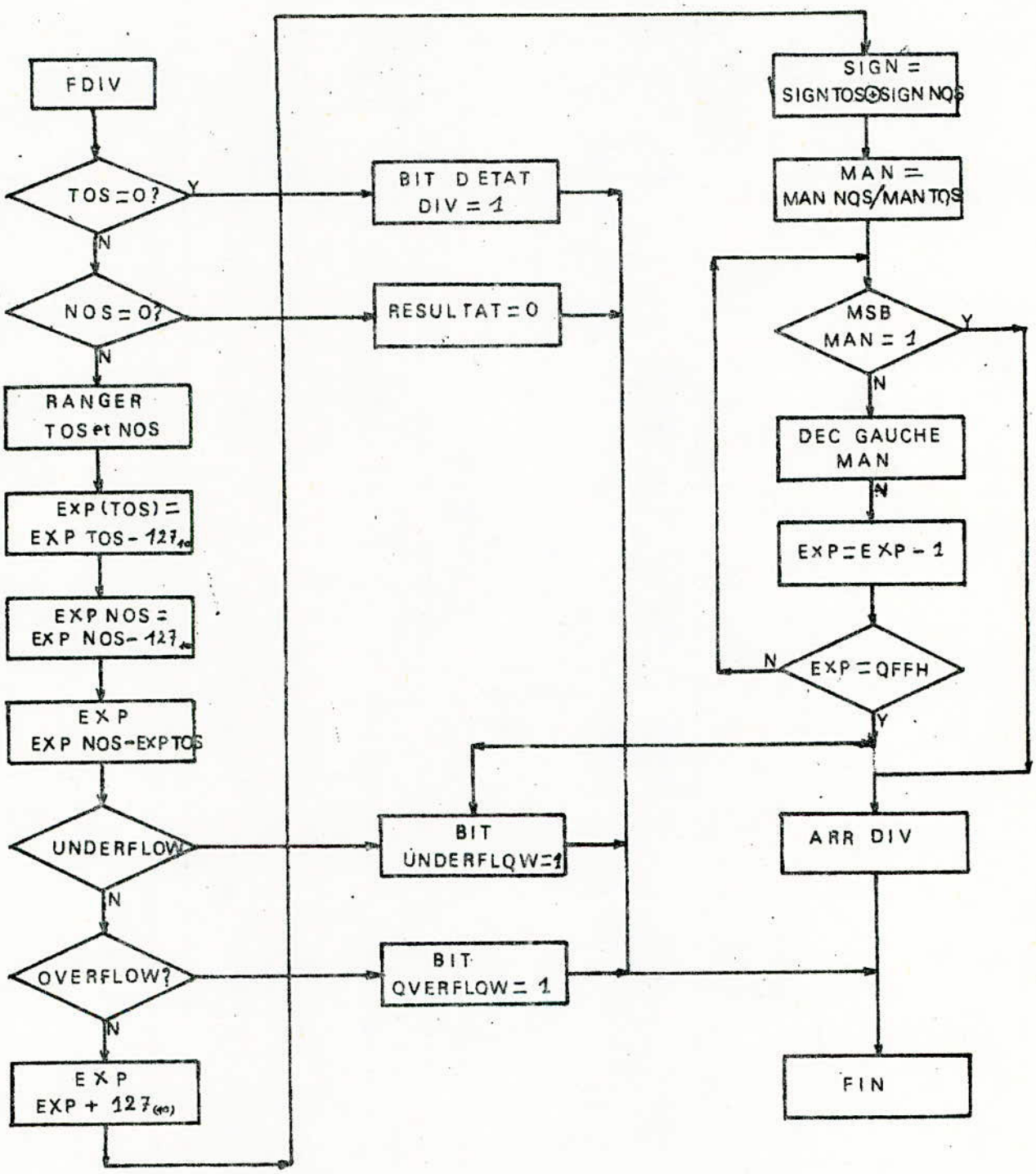


fig e-3 DIVISION EN VIRGULE FLOTTANTE

b - si NOS est nul, sortie du résultat égal à 0

c - ranger TOS et NOS

d - conversion des exposants de TOS et NOS à la forme unbiased
exp (TOS) = exp (TOS) - 127(10)
exp (NOS) = exp (NOS) - 127(10)

e - soustraction des exposants de TOS et NOS
exp = exp (NOS) - exp (TOS)

f - si le MSB de l'exposant de NOS est nul, le MSB de l'exposant de TOS est à 1, si le MSB de l'exposant du résultat est égal à 1, mise à 1 du bit d'overflow et sortie du résultat.

g - si le MSB de l'exposant de NOS est égal à 1, si le MSB de l'exposant de TOS est égal à 0, et si le MSB de l'exposant du résultat est égal à 0, mise à 1 du bit d'underflow et sortie du résultat.

h - addition du bias à l'exposant du résultat
ex = exp + 127(10)

i - si les signes de TOS et NOS sont égaux, mettre le signe du résultat égal à 0, s'ils sont différents, mettre le signe du résultat à 1

j - division de la mantisse de NOS par celle de TOS

k - si le MSB de la mantisse du résultat est nul, décalage à gauche de celle - ci est décrémentation de l'exposant du résultat, sinon aller en n

l - si le MSB de l'exposant du résultat passe de 0 à 1 à la suite de la décrémentation, mise à 1 du bit d'underflow

m - aller en k

n - arrondissement de la mantisse si nécessaire et sortie du résultat.

DADD

SADD

ADDITION EN VIRGULE FLOTTANTE

DOUBLE PRECISION (D.P.)

Binary coding:

7	6	5	4	3	2	1	0
SRE	0	1	0	1	0	0	1

Hexadécimal coding:

A9 S1 SRE = 1

29 S1 SRE = 0

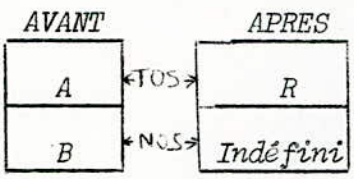
SIMPLE PRECISION (S.P)

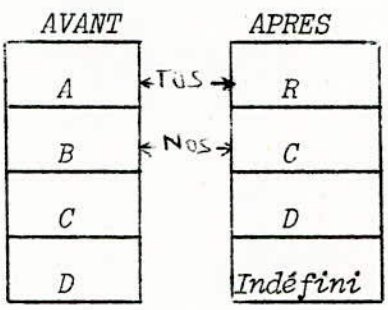
7	6	5	4	3	2	1	0
SPE	0	0	0	0	0	0	1

81 S1 SRE = 1

01 S1 SRE = 0

CONTENU DE LA PILE





DESCRIPTION

L'opérande A de TOS (DP ou SP) est additionné à l'opérande B de NOS ((DP ou SP).Le résultat est arrondi pour obtenir le résultat final R qui est remis dans TOS.Les bits d'état S;Z,U et V sont affectés pour re reporter le signe du résultat si le résultat est zéro;l'exposant under--fow et l'exposant overflow respectivement.Le bit d'état D est mis à zé zéro;les bits d'état affectés sont S,Z,U,V (D est toujours à zéro)

DSUB

SSUB

SOUSTRACTION EN VIRGULE FLOTTANTE

DOUBLE PRECISION (D.P)

code binaire

7	6	5	4	3	2	1	0
SRE	0	1	0	1	0	1	0

SIMPLE PRECISION

7	6	5	4	3	2	1	0
SRE	0	0	0	0	0	1	0

code hexadécimal

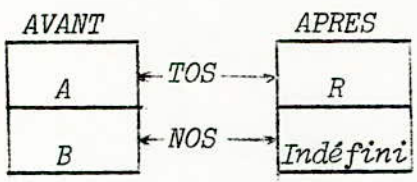
AA S1 SRE = 1

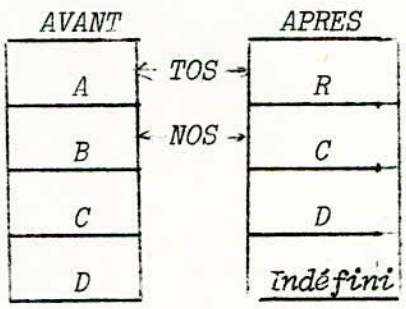
2A S1 SRE = 0

82 S1 SRE = 1

02 S1 SRE = 0

CONTENU DE LA PILE





DESCRIPTION

L'opérande A de TOS (DP ou SP)est soustrait à l'opérande B de NOS. Le résultat est arrondi pour obtenir le résultat finalRR (DP ou SP) qui est remis dans TOS.

Les bits d'états S, Z, U et V sont affectés pour reporter le signe du résultat. Si le résultat est zéro, l'exposant underflow et l'exposant overflow respectivement le bit D est mis à zéro.

Les bits d'états affectés sont/: S, Z, U, V (D est toujours à zéro)

DMUL

SMUL

MULTIPLICATION EN VIRGULE FLOTTANTE

DOUBLE PRECISION (D.P)

code binaire

7	6	5	4	3	2	1	0
SRE	0	1	0	1	0	1	1

code hexadécimal

AB S1 SRE = 1

2B S1 SRE = 0

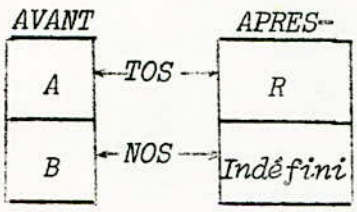
SIMPLE PRECISION (S.P)

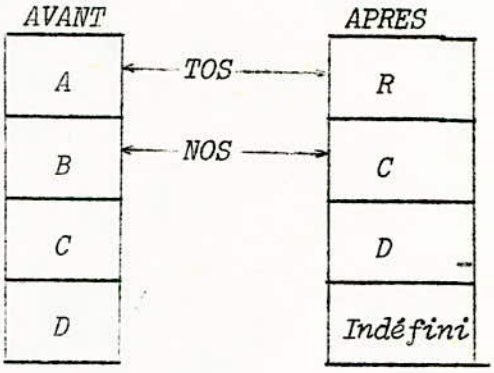
7	6	5	4	3	2	1	0
SRE	0	0	0	0	0	1	1

83 S1 SRE = 1

03 S1 SRE = 0

CONTENU DES LA PILE





DESCRIPTION

L'opérande A de TOS (DP ou SP)est multiplié par l'opérande B de NOS.Le résultat est arrondi pour obtenir le résultat final R (DP ou SP) qui est remis dans TOS.

Les bits d'états S,Z,U et V sont affectés pour reporter le signe du résultat,si le résultat est zéro,l'exposant underflow et l'exposant overflow respectivement.Le bit d'état D est mis à zéro.

DDIV

SDIV

DIVISION EN VIRGULE FLOTTANTE

DOUBLE PRECISION (D.P)

code binaire

7	6	5	4	3	2	1	0
SRE	0	1	0	1	1	0	0

code héxadécimal

AC S1 SRE = 1

2C S1 SRE = 0

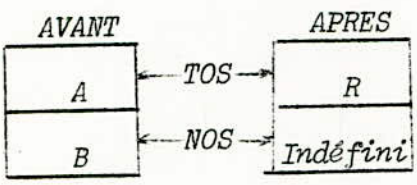
SIMPLE PRECISION(S.P)

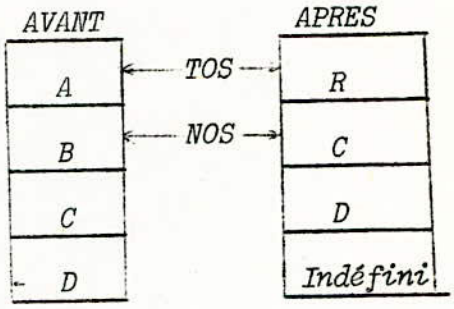
7	6	5	4	3	2	1	0
SRE	0	0	0	0	1	0	0

84 S1 SRE = 1

04 S1 SRE = 0

CONTENU DE LA PILE





DESCRIPTION

L'opérande B de NOS (DP ou SP) est divisé par l'opérande A de TOS. Le résultat (quotient) est arrondi pour obtenir le résultat final R (SP ou DP) qui est remis à TOS.

Les bits d'état S, Z, D, U et V sont affectés pour reporter le signe du résultat si le résultat est zéro, un essai de division par zéro, l'exposant underflow et l'exposant overflow respectivement.

CHSD

CHSS

CHANGEMENT DE SIGNE

DOUBLE PRECISION (D.P)

SIMPLE PRECISION (S.P)

code binaire

7	6	5	4	3	2	1	0
SRE	0	1	0	1	1	0	1

7	6	5	4	3	2	1	0
SRE	0	0	0	0	1	0	1

code hexadécimal

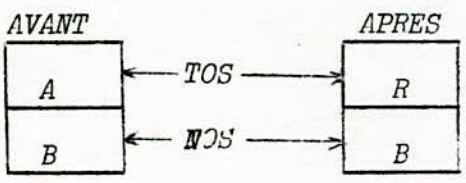
AD Si SRE = 1

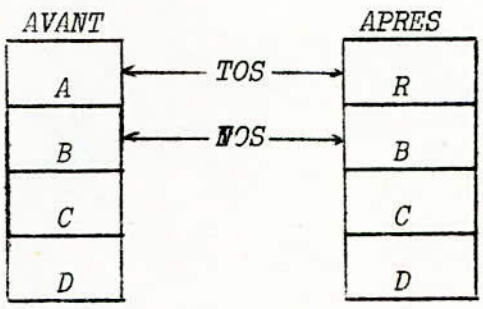
2D Si SRE = 0

85 Si SRE = 1

05 Si SRE = 0

CONTENU DE LA PILE





DESCRIPTION

Le signe de l'opérande A de TOS (DP ou SP) est complémenté à 2; le résultat R (DP ou SP) se trouve à TOS. Si l'opérande (DP) est nul alors le signe n'est pas affecté. Si l'exposant de A (SP) est nul, tous les bits de R sont mis à zéro. Les bits d'état S et Z indiquent le signe du résultat et si le résultat est nul.

Les bits U, V et D sont toujours à zéro.

Les bits d'états affectés sont S, Z (U, V, D = 0)

PTOD

PTOS

ENFONCEMENT DANS LA PILE

DOUBLE PRECISION (D.P)

SIMPLE PRECISION (S.P)

code binaire

7	6	5	4	3	2	1	0
SRE	0	1	0	1	1	1	0

7	6	5	4	3	2	1	0
SRE	0	0	0	0	1	1	0

code hexadécimal

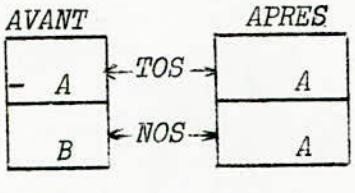
AE Si SRE = 1

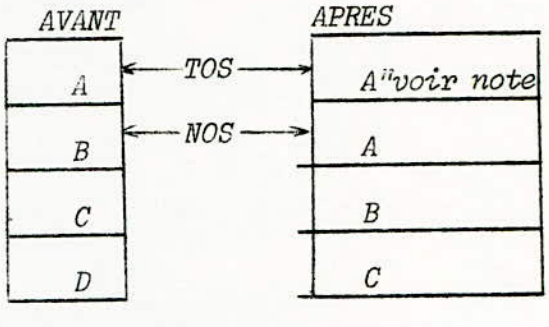
2E Si SRE = 0

86 Si SRE = 1

06 Si SRE = 0

CONTENU DE LA PILE





NOTE : A" = A si le champ de l'exposant de A est non nul

A" = 0 si le champ de l'exposant de A est nul

DESCRIPTION

L'opérande de A de TOS (DP ou SP) est enfoncé dans la pile , ceci est effectivement une duplication de A dans deux positions consé--cutives dans la pile .Cependant si l'opérande (SP) de TOS qui précède la commande PTOS a seulement son champ d'exposant nul alors le nouveau TOS sera mis à 0 . Le contenu de NOS sera une copie exacte de l'ancien TOS .

Les bits d'état S et Z sont affectés pour reporter le signe du nouveau TOS ,et si le contenu de TOS est 0,respectivement les bits d'état U,V et D sont mis à zéro.Les bits d'état affectés sont : S,Z (U,V,D toujours à zéro).

POPD POPS

EXTRACTION DE LA PILE

DOUBLE PRECISION (D.P) SIMPLE PRECISION (S.P)

code binaire

7	6	5	4	3	2	1	0
SRE	0	1	0	1	1	1	1

7	6	5	4	3	2	1	0
SRE	0	0	0	0	1	1	1

code hexadécimal

AF Si SRE = 1 87 Si SRE = 1

2F Si SRE = 0 07 Si SRE = 0

CONTENU DE LA PILE

AVANT		APRES
A	TOS	B
B	NOS	A

Cette opération a le même effet que X.C.H.S

AVANT	APRES
A	B
B	C
C	D
D	A

DESCRIPTION

L'opérande A (DP ou SP)est extrait de la pile et il est réecrit au bas de la pile.

Les bits d'état S et Z sont affectés pour reporter le signe du nouvel opérande de TOS et s'il est égal à zéro,respectivement les bits U,V,D sont mis à 0.

Si le champ de l'exposant du nouveau TOS (S.P) est nul alors si le bit d'état Z est zéro,il sera mis à 1;les bits d'états affectés sont:

S,Z (U ,V,D toujours à 0).

XCHS

ECHANGER TOS ET NOS SIMPLE

code binaire

7	6	5	4	3	2	1	0
SRE	0	0	0	1	0	0	0

code hexadécimal

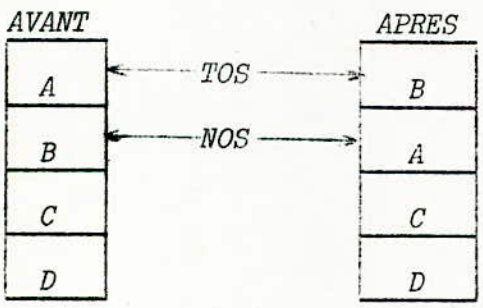
88 Si SRE = 1

08 Si SRE = 0

DESCRIPTION

L'opérande A de TOS (SP) et l'opérande B de NOS (SP) sont échangés. Aprés l'éxécution, B est à TOS et A est à NOS. Tous les autres opérandes sont inchangés. Les bits d'état affectés sont: S, Z, (U, V et D toujours à zéro)

CONTENU DE LA PILE



Les bits d'états S, Z, D, U, V sont mis à 0. La pile n'est pas affectée.

Les bits d'Etats sont à zéro tant qu'il n'y a pas d'opération de commande qui met en jeu les opérandes. Les bits d'Etat affectés sont: S, Z, D, U, V toujours à zéro

CLR

MISE A ZERO DU REGISTRE D'ETAT

code binaire

7	6	5	4	3	2	1	0
SRE	0	0	0	0	0	0	0

code hexadécimal

80 Si SRE = 1

00 Si SRE = 0

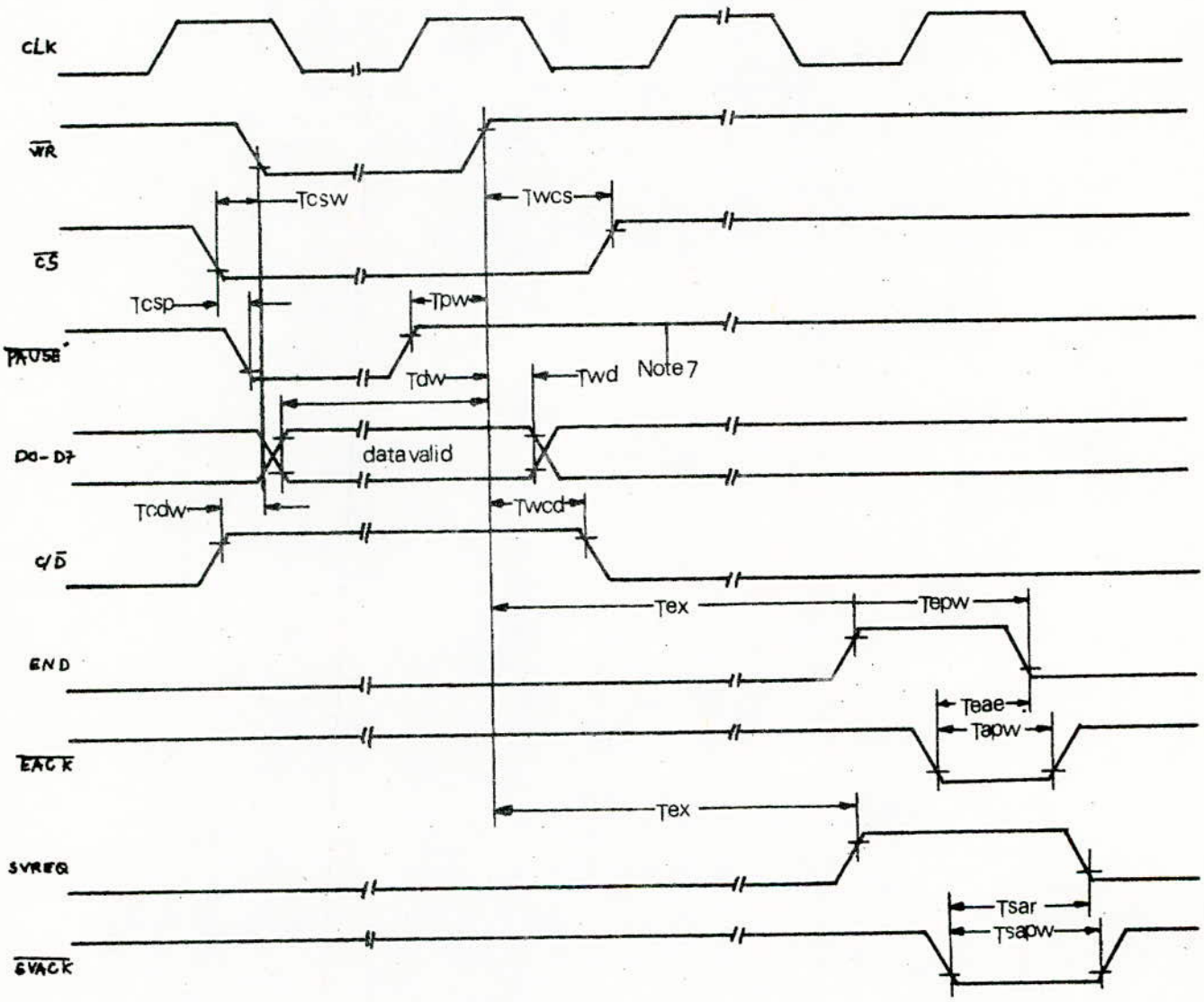
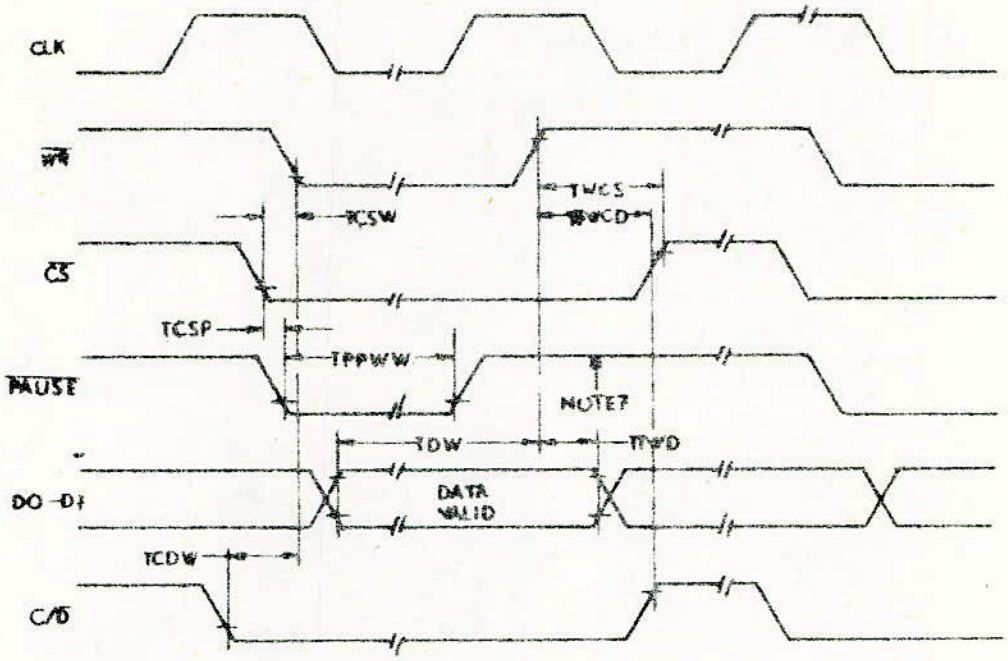


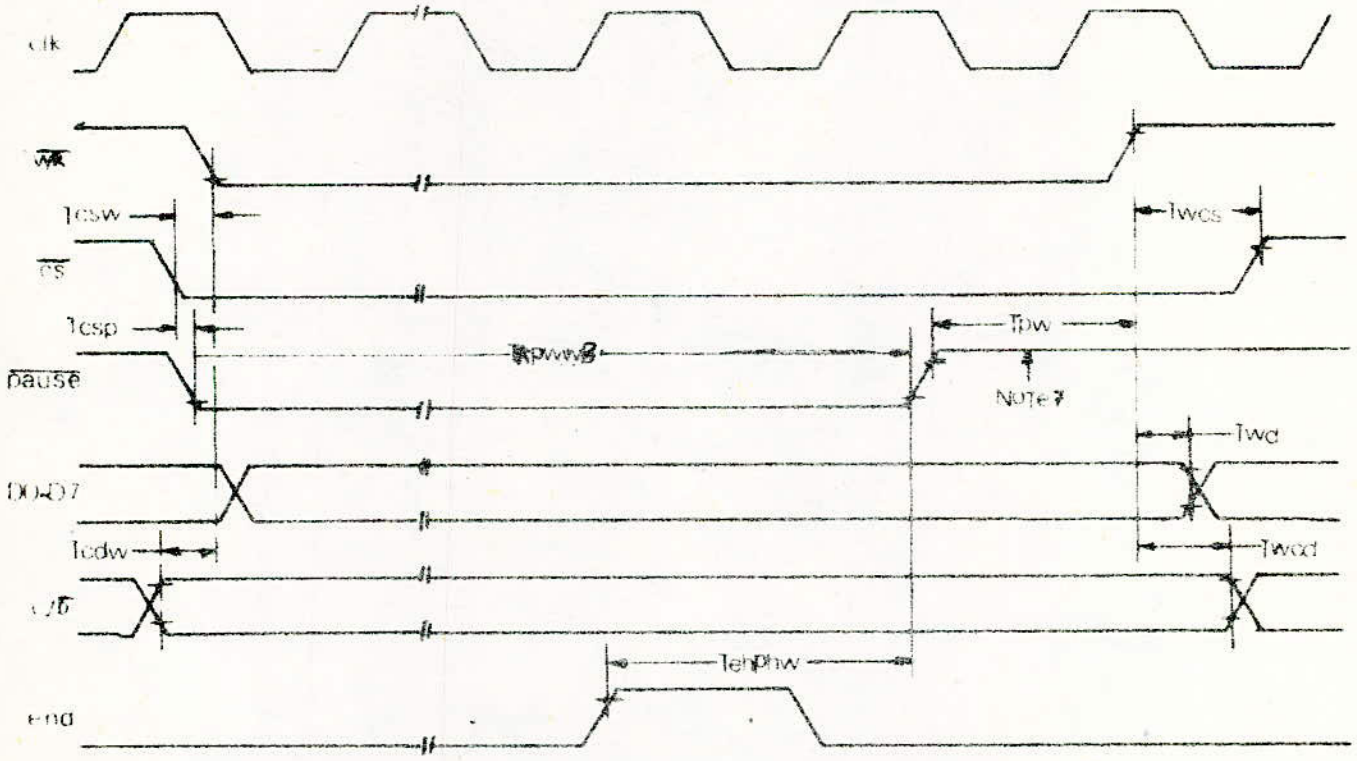
DIAGRAMME DES TEMPS

initialisation d'une commande

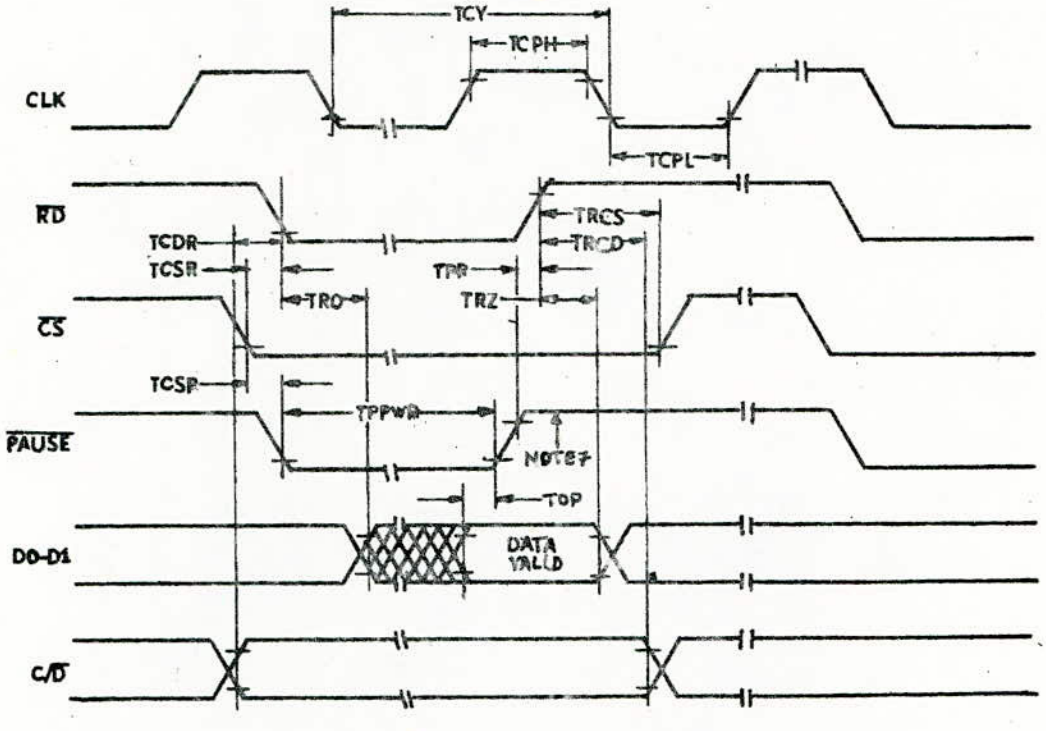
fig 2.f **Entrée d'opérandes**



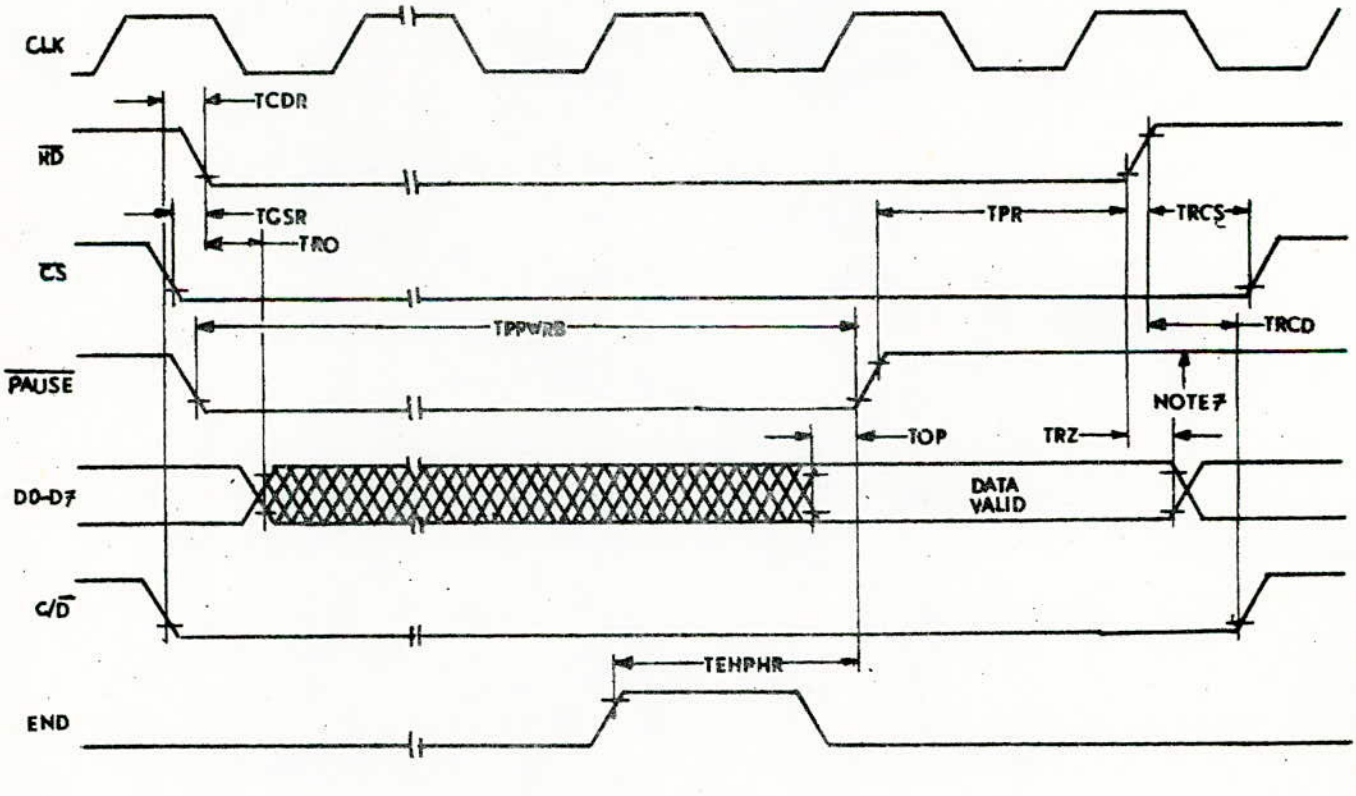
Commande ou écriture d'une donnée quand l'Am9512 est occupé



Opération de lécture



lécture d'operande quand busy = 1



SWITCHING CHARACTERISTICS

Parameters	Description		Am9512DC Min	Am9512DC Max	Am9512-1DC Min	Am9512-1DC Max	Units
TAPW	$\overline{EACK}$ LOW Pulse Width		100		75		ns
TCDR	C/$\overline{D}$ to $\overline{RD}$ LOW Set-up Time		0		0		ns
TCDW	C/$\overline{D}$ to $\overline{WR}$ LOW Set-up Time		0		0		ns
TCPH	Clock Pulse HIGH Width		200	500	140	500	ns
TCPL	Clock Pulse LOW Width		240		160		ns
TCSP	$\overline{CS}$ LOW to $\overline{PAUSE}$ LOW Delay (Note 5)		150		100		ns
TCSR	$\overline{CS}$ to $\overline{RD}$ LOW Set-up Time		0		0		ns
TCSW	$\overline{CS}$ LOW to $\overline{WR}$ LOW Set-up Time		0		0		ns
TCY	Clock Period		480	5000	320	2000	ns
TDW	Data Valid to $\overline{WR}$ HIGH Delay		150		100		ns
TEAE	$\overline{EACK}$ LOW to END LOW Delay			200		175	ns
TEHPHR	END HIGH to $\overline{PAUSE}$ HIGH Data Read when Busy			5.5TCY+300		5.5TCY+200	ns
TEHPHW	END HIGH to $\overline{PAUSE}$ HIGH Write when Busy			200		175	ns
TEPW	END HIGH Pulse Width		400		300		ns
TEX	Execution Time		See Table 2				ns
TOP	Data Bus Output Valid to $\overline{PAUSE}$ HIGH Delay		0		0		ns
TPPWR	$\overline{PAUSE}$ LOW Pulse Width Read	Data	3.5TCY+50	5.5TCY+300	3.5TCY+50	5.5TCY+200	ns
		Status	1.5TCY+50	3.5TCY+300	1.5TCY+50	3.5TCY+200	
TPPWRB	END HIGH to $\overline{PAUSE}$ HIGH Read when Busy	Data	See Table 2				ns
		Status	1.5TCY+50	3.5TCY+300	1.5TCY+50	3.5TCY+200	
TPPWW	$\overline{PAUSE}$ LOW Pulse Width Write when Not Busy			TCSW+50		TCSW+50	ns
TPPWWB	$\overline{PAUSE}$ LOW Pulse Width Write when Busy		See Table 2				ns
TPR	$\overline{PAUSE}$ HIGH to Read HIGH Hold Time		0		0		ns
TPW	$\overline{PAUSE}$ HIGH to Write HIGH Hold Time		0		0		ns
TRCD	$\overline{RD}$ HIGH to C/$\overline{D}$ Hold Time		0		0		ns
TRCS	$\overline{RD}$ HIGH to $\overline{CS}$ HIGH Hold Time		0		0		ns
TRO	$\overline{RD}$ LOW to Data Bus On Delay		50		50		ns
TRZ	$\overline{RD}$ HIGH to Data Bus Off Delay		50	200	50	150	ns
TSAPW	SVACK LOW Pulse Width		100		75		ns
TSAR	SVACK LOW to SVREQ LOW Delay			300		200	ns
TWCD	$\overline{WR}$ HIGH to C/D Hold Time		60		30		ns
TWCS	$\overline{WR}$ HIGH to CS HIGH Hold Time		60		30		ns
TWD	$\overline{WR}$ HIGH to Data Bus Hold Time		20		20		ns

NOTES:
1. Typical values are for T_A = 25°C, nominal supply voltages and nominal processing parameters.
2. Switching parameters are listed in alphabetical order.
3. Test conditions assume transition times of 20ns or less, output loading of one TTL gate plus 100pF and timing reference levels of 0.8V and 2.0V.
4. END HIGH pulse width is specified for $\overline{EACK}$ tied to VSS. Otherwise TEAE applies.
5. $\overline{PAUSE}$ is pulled low for both command and data operations.
6. TEX is the execution time of the current command (see the Command Execution Times table).
7. $\overline{PAUSE}$ will go low at this point if $\overline{CS}$ is low and $\overline{RD}$ and $\overline{WR}$ are high.

III MANIPULATION DE L' OVF AM 9512

Cette unité destiné à accroitre les performances des microprocesseurs, par sa rapidité d'exécution et la précision des résultats; elle sera donc sollicitée pour effectuer certains calcul de méthodes numériques pour la résolution des opérations différentielles. Ainsi pour exécuter l'une des quatres opérations fondamentales sur l'AM 9512, les 2 opérandes se trouvent dans des mémoires fixes M_1 et M_2 disposés de la manière suivante pour la simple précision :

en M_1 ——————→ l'exposant
en M_1 + 1 ——————→ le plus significatif byte de la mantisse (MSB)
en M_1 + 2 ——————→ le 1^{er} moins significatif byte de la mantisse (LSB_1)
en M_1 + 3 ——————→ le $2^{ème}$ moins significatif byte de la mantisse (LSB_2)

Pour l'exécution d'une opération, les opérandes sont envoyés byte par byte, l'un à la suite de l'autre, puis en dernier la commande. Un test par software permet de tester le bit busy du registre d'état, et de voir si le résultat est disponible sur le bus, ce qui permet de le récupérer, de le stocker dans ma mémoire M_1.

La conception d'un programme unique permettrra l'introduction de données dans l'OVF et la réception du résultat stocké en M_1; ce programme s'appellera SAAD, pour une addition en simple précision, SMUL pour une multiplication simple précision, et SDIV pour une division SP et ainsi de suite.

IV - PROGRAMME D'INTRODUCTION DE DONNEES DANS L' AM 9512

Le 1er opérande se trouve à l'adresse M_1
Le 2ème opérande se trouve à l'adresse M_2
Le résultat se trouvera à la mémoire M_1

```
96 63 AVAN  LDA   A  $  0063      lecture du registre d'état
2 B         BMI   AVAN
CE 5018     LDX   #$ 5018        pointer adresse du 1er opérande M1
A6  03      LDA   A  (3,X)
97  62      STA   A  $0062
A6  02      LDA   A  (2,X)
97  62      STA   A  $0062
A6  01      LDA   A  (1,X)
97  62      STA   A  $0062
A6  00      LDA   A  (0,X)
97  62      STA   A  $0062
CE 5014     LDX   #$ 5014        pointer adresse du 2ème opérande M2
A6  03      LDA   A  (3,X)
97  62      STA   A  $0062
A6  02      LDA   A  (2,X)
97  62      STA   A  $0062
A6  01      LDA   A  (1,X)
97  62      STA   A  $0062
A6  00      LDA   A  (0,X)
97  62      STA   A  $0062
86  C.0     LDA   A  #$          code opération
97  63      STA   A  $0063
96  63 TRS  LDA   A  $  0063     test du registre d'état
2 B         BMI   TSR
48          ASL   A
48          ASL   A
48          ASL   A
48          ASL   A
2 B  END    BMI   END            test du bit division par zéro
48          ASL   A
2 B  END    BMI   END            test du bit underflow
48          ASL   A
2 B  END    BMI   END            test du bit overflow
C E 5018    LDX   #$ 5018        pointer adresse du résultat M1
```

```
96  62      LDA  A  $0062
A7  00      STA  A  (0,X)
96  62      LDA  A  $0062
A7  01      STA  A  (1,X)
96  62      LDA  A  $ 0062
A7  02      STA  A  (2,X)
96  62      LDA  A  $ 0062
A7  03      STA  A  $ (3,X)
39  END     RTS                    retour au sous programme.
```

CONCLUSION :

Pour cette unité arithmétique, les temps d'exécution de l'addition, multiplication et division sur 32 bits, sont de 220, 220, 240 cycles MPU; alors qu'ils étaient de 586, 680 et 736 cycles MPU pour une précision de 24 bits seulement. On constate, donc, qu'il y a une nette amélioration dans la rapidité de traitement; de même la précision est augmentée.

CHAPITRE III

COUPLAGE DE L'UNITE DE TRAITEMENT AM 9512

AU µP MC 6800

INTRODUCTION

L'unité de traitement AM 9512 est destinée à accroître sensiblement la capacité de calcul arithmétique des microprocesseurs. Son caractère universel permet de le connecter indifféremment à n'importe quel up. Dans notre cas, le couplage se fait avec le MC 6800. Les échanges avec l'extérieur se font par l'intermédiaire d'un bus bidirectionnel de 8 bits, qui transite les données.

A / CIRCUITS D'INTERFACE :

Un circuit d'interface est un circuit se caractérisant par des des lignes d'entrées et sorties des mêmes informations, et des lignes commandant l'activation de l'interface lui-même. Chaque ligne de bus adresse, de bus de données ou même de contrôle ne peut commander qu'une charge TTL. Lorsque cette ligne doit délivrer un courant supérieur à cette charge, il est nécessaire de prévoir un interface de puissance pour réaliser l'adaptation.

Les circuits d'interface ne servent pas uniquement à fournir de la puissance (amplification, adaptation, leur rôle ne se limite pas, non plus à la protection du up en particulier; ils permettent, et c'est là leur principale caractéristique, d'isoler (déconnecter) les bus d'adresse, de données et certaines lignes de commande ou de contrôle. Ces interfaces sont caractérisés par 3 états (TST) :

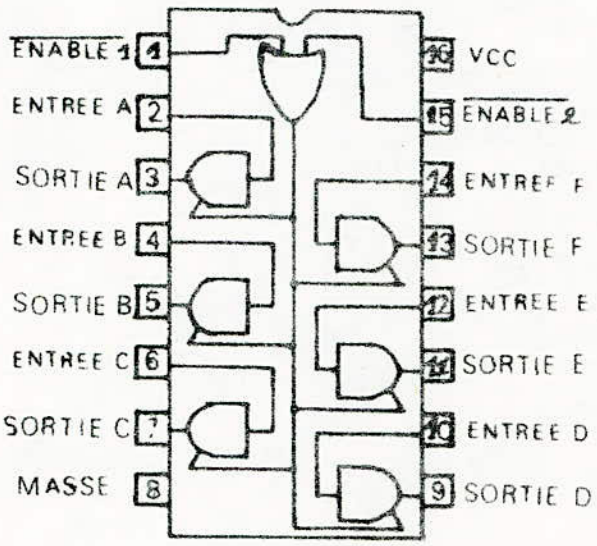
* un état haut : c'est l'état haute impédance, qui signifie que l'interface isole (déconnecte) les bus,
* 2 états bus : { état logique "1" ; état logique "0"

1°) INTERFACE D'ADRESSES :

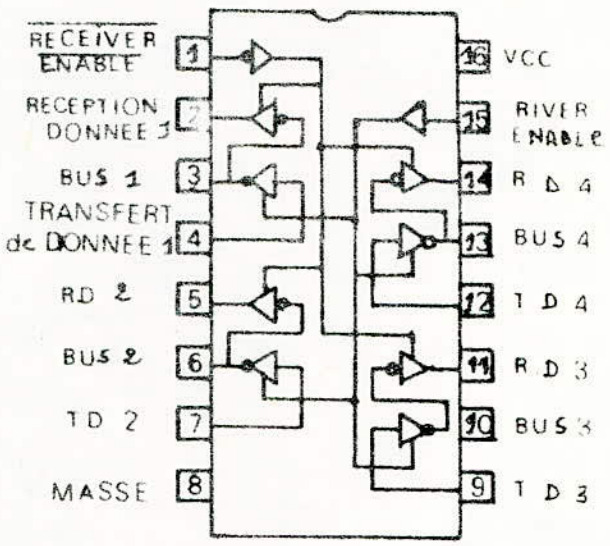
Pour lire ou écrire une donnée dans une mémoire, ou tout autre élément prévu à cet effet, le up muni d'une adresse au moyen de son compteur ordinal, place celle ci sur son interface d'adresse. Ceci signifie, donc que les adresses ne peuvent avoir qu'un seul sens, le sens sortant (du up vers les circuits externes).

Les interfaces d'adresses seront donc, des circuits unidirectionnels de proctection, d'amplification, munis de la possibilité d'être mis en haute impédance. De tels circuits sont appelés "buffers 3 états unidirectionnels". Les buffers qu'on utilisera sont du type MC 8T95. (voir brochage et table de vérité fig.).

MC 8T95



MC 8T26

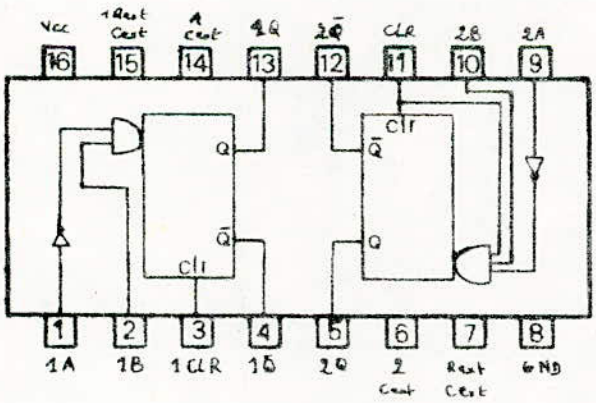


ENABLE	ENABLE	ENTREE	SORTIE
0	0	0	0
0	0	1	1
0	1	Ø	isolée
1	0	Ø	Ø
1	1	Ø	Ø

FIG - a

DREN	RECER	RD	TD	BUS
0	0	0	Ø	0
0	0	1	Ø	1
0	1	Ø	Ø	isolé
1	1	Ø	0	0
1	1	Ø	1	1

FIG - b



entrées			sorties	
CLEAR	A	B	Q	Q
0	x	x	0	1
x	1	x	0	1
x	x	0	0	1
1	0	↑	⎍	⎍
1	↓	1	⎍	⎍
↑	0	1	⎍	⎍

FIG - c MONOSTABLE 74 LS 123

2°) INTERFACE DE DONNEES :

Suivant qu'il s'agisse d'un ordre de lecture ou d'écriture, les données sont sortantes ou rentrantes (le courant est dans un sens ou dans un autre). Les interfaces de données sont donc des "buffers birectionnels 3 états".

Ce type de buffer est constitué de 2 amplificateurs en tête bêche comparables au MC 8T95. L'interface utilié est du type MC 8T26 (voir shéma et table de vérité fig.). Les lignes de données étant de huit, deux buffers MC 8T26 sont suffisants.

La broche "1" du MC 8T26 valide la réception de données du périphérique donc l'émission de données du MC 6800, c'est à dire l'écriture; par contre la broche "15" (Driver enable) du MC 8T26 valide le transfert de données du périphérique, donc la réception de données du MC 6800, c'est à dire la lecture.

B/. 1. DECODAGE D'ADRESSE DE L'AM 9512 :

Le périphérique AM 9512 est considéré comme une position mémoire, validée par les bits d'adresse A_1 à A_{15}. Une fois sélectionnée, la ligne Ao reliée au C/$\bar{D}$ (command, data) du CI, indique à l'unité arithmétique si le transfert concerne un octet de commande ou un octet de donnée; ou choisit comme adresse de cette position mémoire 62 ou 63 que Ao = 0 ou Ao = 1. Les lignes A1 à A15 passent à travers des portes NAND, avec le VMA, ce qui permet de sélectionner le boîtier de l'AM 9512, c'est à dire la broche $\overline{CS}$.

C/ LOGIQUE DE COMMANDE ET DE CONTROLE

L'unité arithmétique travaille sur un top d'horloge Φ 2, on relie donc directement le CLK à Φ2 du MC 6800. La ligne $\overline{reset}$ du MC 6800, après être inversée, est reliée à la ligne reset du CI.

2. 1 - CIRCUIT DE LECTURE, ECRITURE :

Ce circuit détermine s'il faut autoriser un transfert ou une une réception de données, suivant qu'il reçoit un ordre de lecture ou décriture sur les interfaces de données.

a - Opération d'écriture

L'opération d'écriture n'a lieu que si le signal d'écriture $\overline{WR}$ est validé (non placé en haute impédance), et la commande R/W = 0. La présence du signal Φ2 est indispensable car les éléments à écrire ou à lire (mémoires) ne sont activées que pendant ce temps.

Le signal permettant l'écriture résulte de la réunion de toutes ces conditions, la table de vérité nous donne :

$\overline{WR} = \overline{\overline{R/W} \cdot \Phi 2}$.

Dans le cas du buffer MC 8T26, le signal d'activiation de la sortie des données, c'est à dire la borne"1" doit être à l'état bas; de ce fait le signal WR est relié directement à la borne "1".

b) Opération de lecture :

Elle n'a lieu que si le boîtier de l'OVF est validé soit $\overline{CS} = 0$, et le signal du MC 6800 doit être nécessairement un signal de lecture soit R/W = 1. Il faudrait aussi que le bus de données du up soit disponible pour lire la donnée, donc $\Phi 2 = 1$. La réunion de toutes ces conditions nous donne le signal de lecture :

$\overline{RD} = \overline{\Phi 2 \cdot R/W \cdot CS}$

Pour le buffer MC 8T26, le signal d'activation de l'entrée des données (interface en mode lecture = borne "15") doit être au niveau logique "1". De ce fait le signal $\overline{RD}$ est relié à la borne "15" à travers un inverseur (voir shéma de cablage)

Les impulsions des signaux $\overline{WR}$ et $\overline{RD}$ obtenus ont même durée que celle du $\overline{CS}$; or, d'après le timing donné par le constructeur, la durée de $\overline{CS}$ devrait être double de celle de $\overline{WR}$ et $\overline{RD}$ (voir diagramme des temps). On utilise donc 2 monostables 74 LS 123 pour diminuer la durée des impulsions de lecture et d'écriture $\overline{RD}$ et $\overline{WR}$.

CHAPITRE IV

PROGRAMMES APPELANTS POUR LA METHODE DE RESOLUTION DES EQUATIONS DIFFERENTIELLES

I - INTRODUCTION

Pour résoudre une équation différentielle, il est nécessaire d'introduire d'abord, une table données en virgule flottante (4 bytes par donnée en simple précision et 8 bytes pour une double précision). La résolution d'une équation va évidemment utiliser les différentes opérations pour les calculs intermédiaires, les résultats sont ensuite écrits sous forme de table.

MOROLA met à notre disposition une bibliothèque mathématique donctionnant comme un tout indivisible, l'esnemble étant formé de plusieurs sous-programmes. Une caractéristique de la bibmat, est qu'un nombre quelconque, positifi, négatif ou décimal est écrit en virgule flottante de 24 bits, soit 16 bits pour la mantisse et 8 bits pour l'exposant.

On constate donc, que la bibmat ne se prête pas facilement à notre utilisation, puisqu'on travaille en 32 bits ou 64 bits; pour celà il faut concevoir un programme appelant pour chaque fonction en tenant compte du fait que l'OVF travaille sur 4 ou 8 bytes.

A - UTILISATION DE LA PILE DU MC 6800 :

La résolution d'équations différentielles utilise plusieurs sous-programmes, parmi eux : sous-programme de conversion d'un nombre en virgule flottante, introduction de ce nombre en pile, transfert d'une mémoire à une autre, conversion d'un nombre négatif en son complément à 2 ....

Chacun de ces sous-programmes utilise la pile du up, les différentes réservations des arguments et des adresses de retour dans la pile sont disposées de la manière suivante :

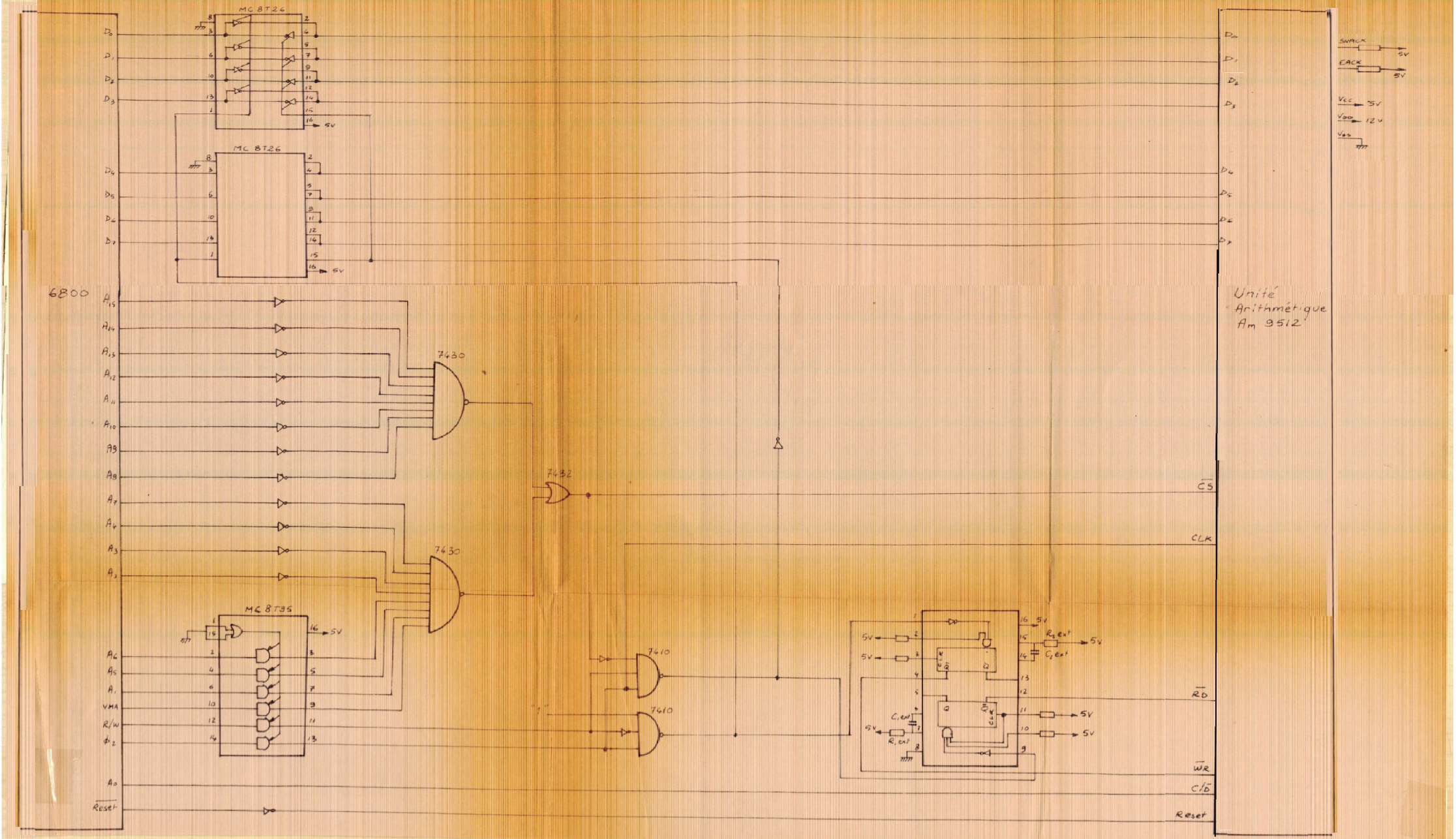
LSB (moins significatif byte) de l'adresse de retour
MSB (plus significatif byte) de l'adresse de retour
LSB_2 (2ème moins significatif byte) de l'argument introduit
LSB_1 (1er moins significatif byte) de l'argument introduit
MSB (plus significatif byte) de l'argument introduit
EXP (exposant) de l'argument introduit.

B - LES DIFFERENTS SOUS PROGRAMMES :

1°) Appel de push 6 :

Il permet d'utiliser 6 registres volatils de la pile, 4 sont occupés par le LSB_2, LSB_1, MSB et l'exposant du nombre introduit; les 2 derniers contiennent le LSB et MSB de l'adresse de l'argument qu'on vient d'introduire. L'adresse de retour est sauvegardée dans la mémoire psh 4 T_2

SCHEMA DE CABLAGE DE L'Am 9512 AVEC LE MC 6800



X : contient l'exposant et le MSB du nombre

B : contient le LSB1

A : contient le LSB_2

```
  psh 4 T1   CQU  501 C
  psh 4 T2   CQU  501 E.
  FF 501 C   STX  psh4 T1      sauvegarde le MSB et l'exposant
  30         TSX
  EE 00      LDX  (0,X)        pointer adresse de retour
  FF 501 E   STX psh4 T2        ranger adresse de retour en psh4 T1
  31         INS
  31         INS
  36         psh A             LSB2 en pile
  37         psh B             LSB1 en pile
  B 6 501 D  LDA A psh4 T1 + 1
  36         psh 4             MSB en pile
  B6 501 C   LDA A psh4 T1
  36         psh A             exposant
  30         TSX
  FF 501 C   STX psh4 T1
  B6 501 D   LDA A psh4 T1 + 1 pointer LQB adresse du nombre
  36         psh A
  B6 501 C   LDA A psh4 T1     pointer MSB adresse du nombre
  36         psh A
  FE 501 E   LDX psh4 T2       pointer adresse de retour
  GE 00      JMP  (0,X)
```

2°) Appel de SPNEG :

Cette soubroutine convertit un nombre négatif, en son complément à 2. Le complément à 2 de la mantisse est formé sans transformer l'exposant. La mantisse est ensuite normalisée si c'est nécessaire, puis lue avec son exposant.

Le registre d'index pointe l'adresse du A_1.

```
63 01      COM (1,X)          complément à 2 du MSB
63 02      COM (2,X)          complément à 2 du LSB1
60 03      NEG (3,X)          complément à 1 du LSB2
25         BCS FPPNE2         C = 1 si R ≠ 00
A6 02      LDA A (2,X)
8B 01      ADD A # $ 01
A7 02      STA A (2,X)
24         BCC FPPNE2
```

```
6C 01      INC (1,X)
E6 01  FINE2  LDA B (1,X)
C1 80  CMP B #$ 80
26         BNE  END
A6 02  LDA A (2,X)
81 00  CMP A #$ 00
26         BNE   END
A6 03  LDA A (3,X)
81 00  CMP A #$ 00
26         BNE  END
0C         CLC                    C = 0
66 01  ROR (1,X)                  décalage à droite de la mantisse
66 02  ROR (2,X)
66 03  ROR (3,X)
6C 00  INC (0,X)                  ajuster l'exposant
39     END  RTS
```

3° Appel de SPPCPY :

Cette soubroutine permet le transfert d'un argument d'une adresse à une autre.

```
30         TSX
EE 02  LDX (2,X)                  pointer l'adresse du nombre
A6 00  LDA A (0,X)                changer l'exposant
E6 01  LDA B (1,X)                changer le MSB
30         TSX
EE 00  LDX (0,X)                  pointer l'adresse de transfert
A7 00  STA A (0,X)
E7 01  STA B (1,X)
30         TSX
EE 02  LDX (2,X)
A6 02  LDA A (2,X)                changer le LSB1
E6 03  LDA B (3,X)                changer le LSB2
30         TSX
EE 00  LDX (0,X)
A7 02  STA A (2,X)
E7 03  STA B (3,X)
31         INS
31         INS
31         INS
31         INS
39         RTS                     pointer adresse de retour
```

4°) Appel de SPFLT :

Ce sous-programme convertit un nombre de 4 caractères héxadécimaux en un nombre en virgule flottante de 32 bits (8 caractères). Cette conversion se fait dans le cas général de la manière suivante :

* le nombre est écrit à partir du bit 0 de la mantisse
* le byte de l'exposant est positionné à 23
* l'opération de normalisation est appliquée au nombre
* le nombre est lu après normalisation.

Mais dans notre cas particulier, la conversion, se fait caractère par caractère, et un chiffre s'écrit au maximum sur un byte en héxadécimal, les autres bits étant des zéros. Les étapes de la conversion seront donc :

a) le chiffre est écrit à partir du bit 0 de la mantisse sur un byte
b) le byte de l'exposant est positionné à 7
c) l'opération de normalisation est appliquée au nombre
d) le nombre est lu après normalisation, il représente le MSB, le LSB_1 et le LSB_2 étant nuls.

Exemple : 9 en décimal ⟶ 0000 1001 (binaire)

0000 1001	0000 0111
MSB	EXP
0.100 1000	0000 0100
4 8	0 4

le nombre s'écrira donc en VF sur 4 bytes : 48 00 0004

Sous-programme SPFLT :

```
30          TSX
86 07   LDA A #$ 07              exp = 07
A7 14   STA A (STFPA3 , X)
6D 11   TST   (STFPA3 + 3,X)
27      BEQ  FLT
A6 11   LDA A (STFPA3 + 3,X)
48      FPPN  ASL A              opération de normalisation
2B      BMI  FPPT
68 11   ASL (STFPA3 + 3,X)
6A 14   DEC (STFPA3 ,X)
20      BRA  FPPN
A6 11   FPPT  LDA A (STFPA3 + 3,X)
A7 13   STA A (STFPA3 + 1,X)
6F 11   CLR (STFPA3 + 3,X)
39      FLT  RTS
```

5°) Appel de QUERY :

Cette soubroutine permet le changement de ligne, après la conversion d'un argument en VF; ce processus est répété jusqu'à la fin de la table de données.

```
                JSR NEWLIN      changement de ligne
86 3F           LDA A #'
BD F9DC         JSR  OUTCH
39              RTS

86 OD  NEWLIN   LDA A #  13
BD F9DC         JSR OUTCH
86 OA           LDA A #  10
BD F9DC         JSR  OUTCH
39              RTS
```

6°) Appel de SETUP 6 :

Ce sous-programme permet après le point d'interrogation d'introduire une donnée en décimal, qui sera directement convertie en virgule flottante et stockée à la mémoire d'adresse M3.

La conversion se fait caractère par caractère en utilisant les sous-programmes donnés précédemment (voir programme)

SETUP 6

```
JSR   QUERY
LDA A #$ 10        LSB de M3
PUSH A
LDA A #$ 50        MSB de M3
PUSH A
CLR A
PUSH A             STFPDP = 0   virgule flottante
PUSH A             STFPSG = 0   Signe
LDX #$ 0000        exp et MSB de A1
LDA A #$ 00        LSB1 de A1
LDA B #$ 00        LSB2 de A1
JSR  PSH6          STFPA1 = 0
LDX #$ 0140        exp et MSB de A2
LDA B #$ 00          LSB1 de A2
LDA A #$ 00          LSB2 de A2
JSR  PUSH6         STFPA2 = 01
CLR A
PUSH A
PUSH A
PUSH A
PUSH A             STFPA3
TSX
STX  PSH4T1
LDA A PSH4T1+1
PUSH A
LDA A PSH4T1
PUSH A             Adresse de A3
LDX #$ 0450        exp et MSB de A4
LDA B #$ 00        LSB1 de A4
LDA A #$ 00        LSB2 de A4
JSR  PUSH6         STFPA4 = 10
TSX
```

Définition des mémoires dans la pile

```
STFPA4   EQU  2
STFPA3   EQU  8
STFPA2   EQU  14
STFPA1   EQU  20
STFPSG   EQU  24    Signe
STFPDP   EQU  25    virgule flottante
STFPRS   EQU  26    Adresse du Résultat
```

Boucle de Conversion Caractère par Caractère

```
STFP1  JSR INCH          .Un Caractère dans ACC A
       CMP A  #$ A       Comparaison avec line feed
       BEQ  STFP4        oui
```

```
        CMPA #'-        Comparaison avec signe moins
        BNE STFP2       Non
        INC (STFPSG,X)  Oui, mise à 1 du signe
        BRA STFP1
STFP2   CMPA #'.        Comparaison virgule flottante
        BNE STFP3       Non
        INC(STFPDP,X)   Oui, mise à 1 de la virgule flottante
        BRA STFP1
STFP3   CMPA #$40       Comparaison à 40($40 : 10 en asc II)
        BPL STFP1
        CMPA #'0
        BMI STFP1
        SUB A #'0       Soustraire le code de zéro au contenu
        STA A (STFPA3+2,X)  Ranger [A] dans A3          [de A
        JSR SPFLT       A3 = A3 en VF
        TSX
        LDX (STFPA,-2,X)    Pointer l'adresse de A,
        LDA A (0,X)
        LDA B (1,X)
        STA A ($5018) M1    Transfert de A, en M1
        STA B (5019) M1+1
        LDA A (2,X)
        LDA B (3,X)
        STA A (501A) ← M1+2
        STA B (501B)   M1+3
        TSX
        LDX (STFPA4-2,X)    Pointer l'adresse de A4
        LDA A (0,X)
        LDA B (1,X)
        STA A (5014) M2     Transfert de A4 en M2
        STA B (5015) M2+1
        LDA A (3,X)
        LDA B (2,X)
        STA A (5016) M2+2
        STA B (5017) M2+3
        JSR SMUL        A1 = A1 x 10 en M1
        TSX
        LDX (STFPA1-2,X)    Pointer l'adresse de A1
        LDA A 5018 M1       Transfert de [M1] = A1 x 10 dans A1
        LDA B 5019 M1+1
        STA A (0,X)
        STA B (1,X)
        LDA A (501A) M1+2
        LDA B 501B M1+3
        STA A (2,X)
        STA B (3,X)
        TSX
        LDX (STFPA3-2,X)    Pointer l'adresse de A3
        LDA A (0,X)
```

```
LDA B (1,X)
STA A (5016)M2          Transfert de A3 en M2
STA B (5016)M2+1
LDA A (2,X)
LDA B (3,X)
STA A (5016) M2+2
STA B 5017 M2+3
JSR SADD                A1 = A1 + A2 en M1
TSX
LDX (STFPA1-2,X)        Pointer l'adresse de A1
LDA A $ 5018 M1
LDA B $ 5019 M1+1
STA A (0,X)
STA B (1,X)
LDA A 501A M1+2
LDA B 5018 M1+3
STA A(2,X)
STA B(3,X)
TSX
TST (STFPDP,X)          la virgule est-elle à 1
BLE STFP1               Non
LDX (STFPA2-2,X)        Oui, pointer l'adresse de A2
LDA A (0,X)
LDA B (1,X)
STA A 5018 M1           Transfert de A2 en M1
STA B 5019 M1+1
LDA A (2,X)
LDA B (3,X)
STA A 501A M1+2
STA B 5018 M1+3
TSX
LDX (STFP4-2,X)         Pointer l'adresse de A4
LDA A (0,X)
LDA B (1,X)
STA A 5014 M2           Transfert de A4 en M2
STA B 5015 M2+1
LDA A (2,X)
LDA B (3,X)
STA A 5016 M2+3
STA B 5017 M2+3
JSR SMUL                A2 = A2 x 10 en M1
TSX
LDX (STFPA2-2,X)        Pointer l'adresse de A2
LDA A 5018 M1           Transfert de A2 = A2x10 de M1
                        [en ad. de A2
```

```
        LDA  B  5019  M1 +1
        STA  A  (0,X)
        STA  B  (1,X)
        LDA  A  501A  M1 +2
        LDA  B  501B  M1 +3
        STA  A  (2,X)
        STA  B  (3,X)
        TSX
        BRA  STFP1
STPP4   LDX  (STPPA1 -2,X)    Pointer l'adresse de A1
        LDA  A  (0,X)
        LDA  B  (1,X)
        STA  A  5018  M1      Transfert de A1 en M1
        STA  B  5019  M1 +1
        LDA  A  (2,X)
        LDA  B  (3,X)
        STA  A  501A  M1 +2
        STA  B  501B  M1 +3
        TSX
        LDX  (STPPA2 -2,X)    Pointer l'adresse de A2
        LDA  A  (0,X)
        LDA  B  (1,X)
        STA  A  5014  M2      Transfert de A2 en M2
        STA  B  5015  M2 +1
        LDA  A  (2,X)
        LDA  B  (3,X)
        STA  A  5016  M2 +2
        STA  B  5017  M2 +3
        JSR  SDIV             A1 = A1/A2 en M1
        TSX
        LDX  (STFPA1 -2,X)    Pointer l'adresse de A1
        LDA  A  5018  M1      Transfer de [M1] à l'ad. de A1
        LDA  B  5019  M1 +1
        STA  A  (0,X)
        STA  B  (1,X)
        LDA  A  501A  M1 +2
        LDA  B  501B  M1 +3
        STA  A  (2,X)
        STA  B  (3,X)
        TSX
        TST  (STFPSG,X)       Test du signe. Le signe est-il à 1
        BEQ  STFP5            Non
        TSX                   Oui
        LDX  (STFPA1 -2,X)
        JSR  SPNEG            A1 = -A1
        TSX
STFP5   LDA  A (STFPA1,X)     Transfert du résultat à l'ad. M3
        LDA  B (STFPA1+1,X)
        LDX  (STFPRS,X)
```

```
        STA A (0,X)
        STA B (1,X)
        TSX
        LDA A (STFPA1+2,X)
        LDA B (STFPA1+3,X)
        LDX (STFPRS,X)
        STA A (2,X)
        STA B (3,X)
        LDA A #$1C
STFPC   INC                          Boucle d'effacement de la pile
        DEC A
        BGT STFPC
        RTS.
```

Definition des mémoires

```
M1          EQU   $501B
M2          EQU   $5014
M3          EQU   $5010
PSH4T1      EQU   $501C
PSH4T2      EQU   $501E
```

CHAPITRE V

METHODES NUMERIQUES POUR LA RESOLUTION DES EQUATIONS DIFFERENTIELLES

A/- INTRODUCTION

Depuis plus de quatre siècles, les méthodes numériques sont utilisées pour résoudre des problèmes complexes à l'aide des règles à calcul, et puis récemment avec des calculateurs analogiques digitaux. Ces méthodes numériques sont utilisées pour développer le modèle de simulation qui est une approximation discrète du système réel.

Dans la sélection de la technique numérique, l'utilisation doit considérer l'exactitude et l'efficacité de la méthode spécifiée à son application correspondante.

Un système discret consiste en une série de pulsations en des temps égaux (T); soit f (t) une fonction continue, quand elle est échantillonée à des intervalles égaux dans le temps t, cette fonction peut être déterminée par la séquence de nombres f (0), f(T), f (2 T)...f (N T). Cette série de nombre donne une description limitée de la fonction f (t), les valeurs de celles ci sont connues seulement aux temps 0, T, 2 T, ... n T. Les autres valeurs de f (t), à d'autres valeurs du temps sont trouvées par extrapolation ou interpolation.

A chaque pas de la simulation, la réponse simulée diffère légèrement 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 déterminée 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 rationnel. Les erreurs s'additionnent à chaque pas de la simulation, ceci nous conduit à une réponse approximative de la réponse exacte comme le montre la fig. 1 - a.

B: METHODE DE RUNGE - KUTTA 4ème ORDRE

1°) - Présentation de la méthode Runge - Kutta

Durant des années, une large variété de méthodes numériques a été développée dans le but de retenir l'avantage sur les séries de TAYLOR, qui étaient alors, utilisées pour le calcul des hautes dérivées, et des équations différentielles.

La plus importante et la plus utilisée parmi ces méthodes dans la simulation numérique est celle de Runge - Kutta; celle ci étant la plus facile à utiliser sur les machines à calculer.

Soit à résoudre l'équation différentielle : $\frac{dy}{dx} = f(x, y)$ et soient (xo, yo) les conditions initiales dans un domaine D où la fonction est définie, continue. Le problème est de rechercher les valeurs que prend la fonction y des valeurs de x en progression arithmétique de raison h; xo, xo + h, xo + 2 h... xo + nh. La solution d'une telle équation est de la forme :

$$yi + 1 = yi + \int_{hi}^{hi + *} f(x, y)\, dx = yi + \int_{xi}^{xi + 1} f(x, y)\, dx$$

$$yi + 1 = yi + \Delta yi + 1$$

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

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

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

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

Il suffit donc de calculer les différentes dérivées y', y", $y^{(3)}$... et les substituer dans l'équation (1) pour résoudre l'équation différentielle. Ce procédé est appelé méthode de Runge - Kutta. Celle ci au $4^{ème}$ 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 plus complexe et plus longue. La plus puissante et la plus utilisée en pratique, est la méthode du $4^{ème}$ ordre.

Les formules d'approximation de cette méthode sont:

$$yi + 1 = yi + \frac{1}{6}(k_0 + 2\, k_1 + 2\, k_2 + k_3).$$

avec : $k_0 = h.\ f(xo, yo)$

$K_1 = h.\ f(xo + h/2,\ yo + ko/2)$

$k_2 = h.\ f(xo + h/2,\ yo + k_1/2)$

$k_3 = h.\ f(xo + h,\ yo + k_2)$

2°) <u>Algorithme de la méthode</u> :

$$\begin{cases} y(xo) = yo \\ x = xo \end{cases} \qquad h = xi + 1 - xi \text{ pas de la simuation.}$$

Premier pas

On commence par calculer k_o, k_1, k_2, k_3 , ceci nous permettra le calcul de $\Delta y_1 = \frac{1}{6} (k_o + 2\, k_1 + 2\, k_2 + k_3)$ qui sera ajouté à yo, pour donner y_1.

$y_1 = y_o + \Delta\, y_1$.

Cette séquence sera répétée n fois, n étant le nombre de pas choisi. On a donc :

$x_1 = x_o + h$

$y_1 = y\ (x_o + h) = y_o + \Delta\, y_1$.

Deuxième pas

On calcule les ki :

$K_o = h\ f\ (x_1\ ,\ y_1)$

$K_1 = h.\ f\ (x_1 + \frac{h}{2}\ ,\ y_1 + \frac{ho}{2})$

$K_2 = h.\ f\ (x_1 + h/2,\ y_1 + K_1/2)$

$K_3 = h.\ f\ (x_1 + h\ ,\ y_1 + K_2)$

$y_2 = \frac{1}{6} (K_o + 2\, K_1 + 2\, K_2 + K_3)$

$X_2 = X_1 + h$

$y\ (X_1 + h) = y_2 +\quad y_1$.

$(i + 1)^{ème}$ pas :

$xi + 1 = xi + h$

$y\ (xi + h = yi + 1 = yi + \Delta\, yi + 1$

3°) Estimation du temps de résolution

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

$th = 4\ tf + 10\alpha + 7\beta$

avec tf : temps nécessaire pour le calcul 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 , K_3 .

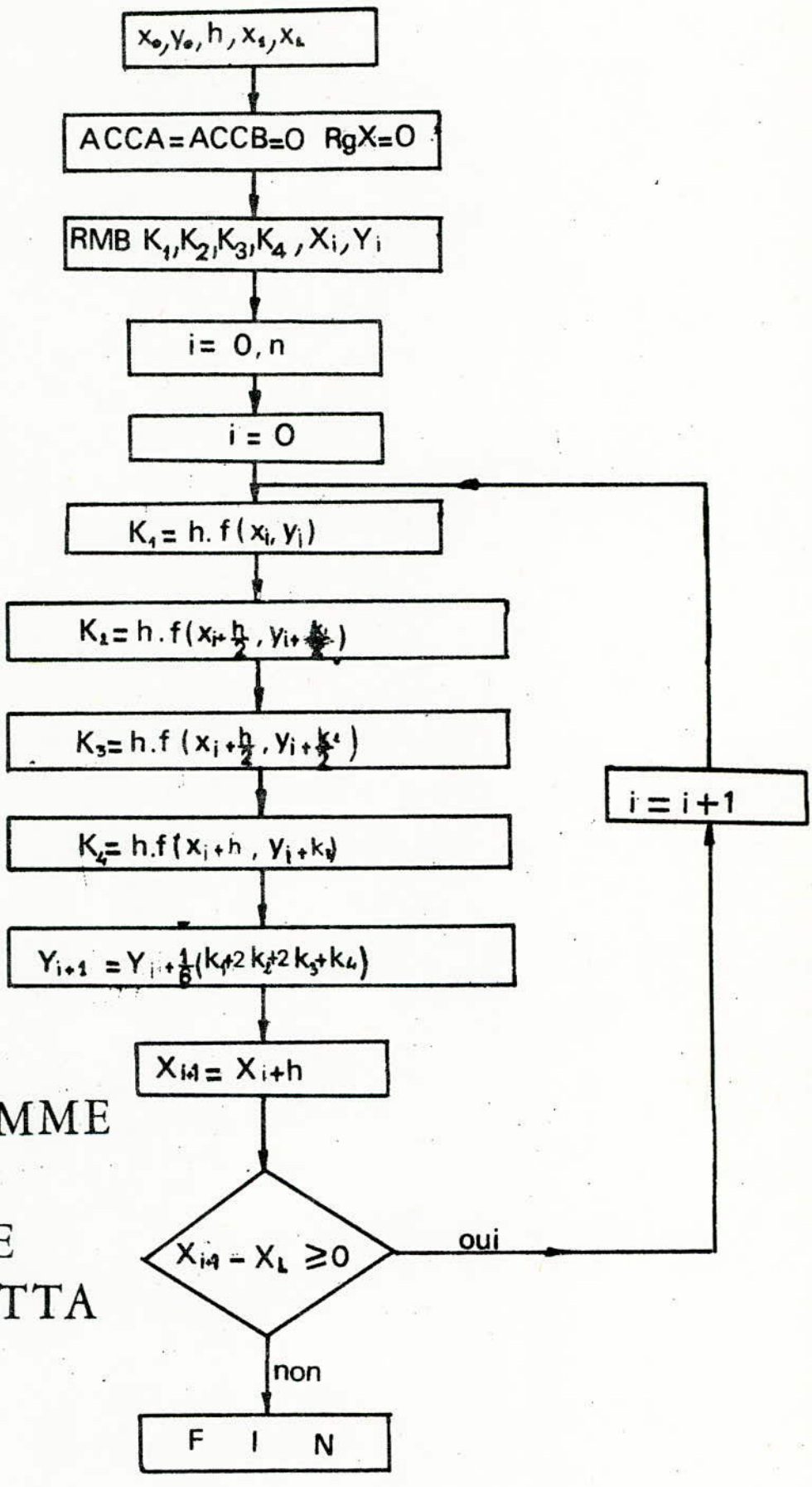
α : temps nécessaire pour faire une addition

β : temps nécessaire pour faire une multiplication

C / PROGRAMME DE RESOLUTION

Pour cette méthode, la difficulté de construction d'un programme de résolution sur le MC 6800, se trouve dans le calcul des ki à chaque pas de la simulation.

ORGANIGRAMME DE LA METHODE RUNGE KUTTA



Pour conserver un programme de résolution valable pour une équation différentielle quelconque, on procède à l'obtention de f (X,y), par une soubroutine spéciale SPSPK, qui sera appelée par le programme principal de résolution pour le calcul des ki. Tout exemple de résolution des équations différentielles par la méthode de Runge - Kutta se compose de 3 étapes :

1ère étape : chargement du sous-programme de calcul des ki de la fonction à simuler.

2ème étape : à l'aide du programme d'introduction de la tablde de données, les valeurs X_L, X_0, Y_0, H et quelques nombres utilisés par les opérations seront stockés dans leurs adresses respectives en virgule flottante.

3ème étape : On appelle le programme principal de résolution; après exécution, les résultats sont disponibles en virgule flottante dans la table des résultats dont le début est à l'adresse 5040 (X_0).

I- Les sous-programmes utilisés :

1°) Programme de chargement d'une table de données

Ce sous-programme consiste à charger les valeurs à utiliser dans le programme principal et à réserver des positions mémoires pour sauve--garder le contenu des accumulateurs A et B, ainsi que de registre d'index. Après toutes ces réservations, on introduit les données une à une; X_L , H, XO , Y_0. Le procédé est le suivant : après le point d'interrogation, on introduit la donnée en décimal, qui sera directement convertie en VF, et stockée dans la mémoire d'adresse M_3, à l'aide de la soubroutine Setup 6. La soubroutine SPEPY, permet le transfert de ce nombre dans la mémoire réservée à cet effet.

```
8E FF       LDS  # 8 FF          initialiser le pointeur de pile
86 50       LDA A # 8 50         pointer la mémoire de la 1ère donnée
87 5004     STA A 8 5004
86 38       LDA A # 8 38
B7 5005 Début STA A 8 5005
BD          JSR SETUP 6
B6 5005     LDA A 8 5005
C6 10       LDA B # 8 10         pointer l'adresse du nombre converti
37          PSH B
C6 50       LDA B # 8 50
```

```
37          PSH B
36          PSH A
37          PSH B
BD          JSR SPCPY       transfert de M3 à l'adresse pointée par
                            ACC A
B6 5005     LDA A $ 5005
8B 04       ADD A # $ 04    pointer la donnée suivante
C6 48       LDA B # $ 48
11          CBA
26          BNEA DEBUT      boucler si non égaux
3F          SWI
```

2°) Programme de calcul des ki :

On pose pour le 1er pas :

$X'0 = X0$
$Y'0 = Y0$ $\quad K0 = H.F(X0, Y0) = H.F(X'0, Y'0)$

$X'_1 = X0 + \frac{H}{2}$
$Y'1 = Y0 + \frac{H0}{2}$ $\quad K_1 = H.F\left(X0 + \frac{H}{2}, Y0 + \frac{K0}{2}\right) = H.F(X'_1, Y'_1)$

$X'2 = X'_1$
$Y'2 = Y_0 + \frac{H_1}{2}$ $\quad K_2 = H.F\left(X0 + \frac{H}{2}, Y0 + \frac{K_1}{2}\right) = H.F(X'2, Y'2)$

$X'_3 = X_0 + H$
$Y'_3 = Y_0 + K_2$ $\quad K_3 = H.F(X_0 + H, Y_0 + K_2) = H.F(X'_3, Y'_3)$

On peut donc écrire pour le premier pas :

$Ki = H.F(X'i, Y'i)$

Pour le ième pas on a :

$X'0 = Xi$
$Y'0 = Yi$ $\quad K_0 = H.F(X'_0, Y'_0)$

$X'_1 = Xi + \frac{H}{2}$
$Y'_1 = Yi + \frac{H_0}{2}$ $\quad K_1 = H.F(X'_1, Y'_1)$

$X'_2 = Xi + \frac{H}{2}$
$Y'_2 = Yi + \frac{H_1}{2}$ $\quad K_2 = H.F(X'_2, Y'_2)$

$X'_3 = Xi + H$
$Y'_3 = Yi + H_2$ $\quad K_3 = H.F(X'_3, Y'_3)$

Pour chaque calcul de Ki, X'i et Y'i sont transférés dans les mémoires M_1 et M_2 avant le calcul de F (X'i, Y'i), à la fin, Ki se trouve à l'adresse M_1.

SPSPK

[500 A] = adresse de X'i
[500 C] = adresse de Y'i

```
86 18       LDA A # ⌀ 18        pointer LSB adresse de M1
D6 500 B    LDA B ⌀ 500 B       pointer LSB adresse X'i
  37        PSH B
D6 500 A    LDA B ⌀ 500 A       pointer MSB adresse X'i
  37        PSH B
  36        PSH A
  37        PSH B
  BD        JSR SPCPY           transfert de X'i en M1
86 14       LDA A # ⌀ 14
D6 500 D    LDA B ⌀ 500 D       pointer LSB adresse de Y'i
  37        PSH B
D6 500 C    LDA B ⌀ 500 C
  37        PSH B
  36        PSH A
  37        PSH B
BD          JSR SPCPY           transfert de Y'i en M2
BD          JSR FCTI            calcul de F (X'i , Y'i) ------> M1
86 14       LDA A # ⌀ 14        pointer adresse de M2
C6 3C       LDA B # ⌀ 3C        pointer adresse de H
  37        PSH B
C6 50       LDA B # ⌀ 50
  37        PSH B
  36        PSH A
  37        PSH B
BD          JSR SPCPY           transfert de H en M2
BD          JSR SMUL            calcul de ki = H. F (X'i, Y'i)
39          RTS
```

Exemple pour SPSPK

Soit à résoudre l'équation différentielle :

$$\frac{dY}{dx} = F(X, Y) = Y - \frac{2X}{y}$$

les données sont : - valeurs initiales : X0 = 0, Y0 = 1
- intervalle d'étude $[0, 2]$
- Valeur du pas
N = 20

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

Ki = H. F (X'i , Y'i)

F (X'i, Y'i) est calculée par la soubroutine FCTi

$$F(X'i, Y'i) = Y'i - \frac{2\ X'i}{Y'i}$$

pour celà on a les étapes suivantes :

* incrémenter l'exposant de X'i $\Longrightarrow$ 2 X'i
* calcul de $\frac{2\ X'i}{Y'i}$
* différence : $Y'i - \frac{2\ X'i}{Y'i}$

pour le calcul des Ki, X'i et Y'i se trouvent dans les mémoires M_1 et M_2 respectivement d'adresses 5018 et 5014.

Soubroutine FCTi

CE 5018	LDX # $ 5018	pointer adresse de X'i
6C 00	INC (0,X)	incrémenter exp de X'i
BD	JSR SDI	
CE 5018	LDX # $ 5018	pointer adresse de 2 X'i/Y'i
A6 00	LDA A (0,X)	
E6 01	LDA B (1,X)	
B7 5014	STA A $ 5014	
E7 5015	STA B $ 5015	
A6 02	LDA A (2,X)	
E6 03	LDA B (3,X)	
B7 5016	STA A $ 5016	
F7 5017	STA B $ 5017	
86 18	LDA A # $ 18	pointer adresse M_1
F6 500 D	LDA B $ 500 D	pointer adresse Y'i
37	PSH B	
F6 500 C	LDA B $ 500 C	
37	PSH B	
36	PSH A	
BD	JSR SPCPY	transfert de Y'i en M_1
BD	JSR SSUB	
39	RTS	

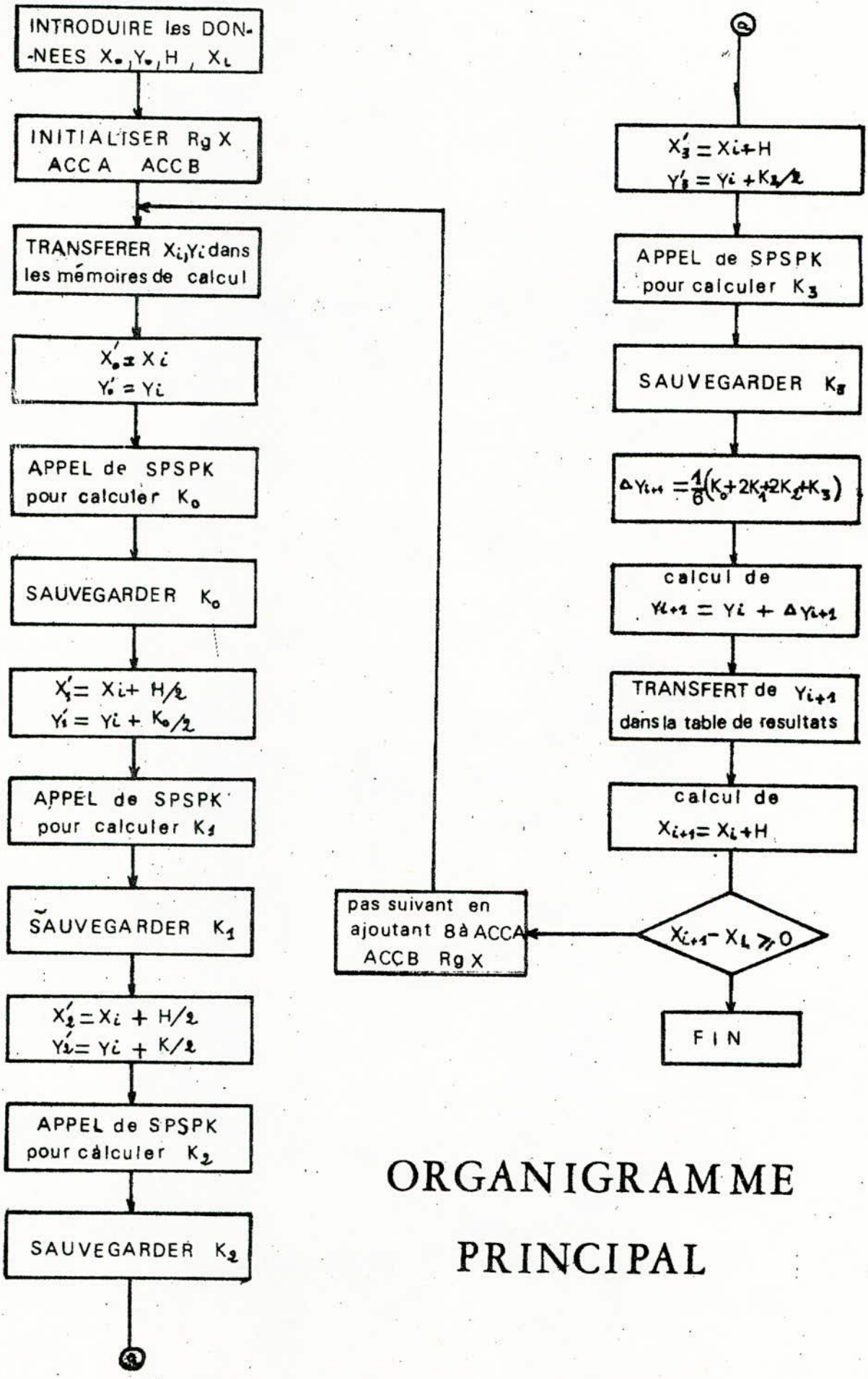
Cette soubroutine FCTi, sera appelée par SPSPK pour le calcul de chaque Ki.

PROGRAMME PRINCIPAL

A l'aide du programme de changement des données, celles-ci sont stockées dans leurs adresses respectives en virgule flottante.

Les données Xi et Yi sont transférées dans les mémoires de travail pour le calcul des Ki .Ceux-ci sont sauvegardés pour la détermina--tion de Yi+1 ,qui sera stocké dans la table des résultats.

La valeur de Xi est augmentée du pas h, Xi+1 = Xi + h et un test est fait si le calcul est complet,sinon on fait le pas suivant en incrémentant de 8 les accumulateurs A et B et le registre d'index.



ORGANIGRAMME PRINCIPAL

PROGRAMME PRINCIPAL

```
        ORG $5000

        RMB 4          RESERVER 4 bytes pour 6.
        RMB 6          LOCATION de memoires de travail
        ORG $5035
XL      RMB 4          valeur limite du domaine
H       RMB 4          pas de la simulation
X0      RMB 4          valeur initiale
Y0      RMB 4          valeur initiale

        ORG $4000
        LDX  #$0360    exp et MSB de 6
        STX  $5000
        LDX  #$0000    LSB1 et LSB2
        STX  $5002
        LDA A #$50
        STA A $5004
        STA A $5006
        LDX  #$5048    initialiser le ag d'index.
        LDA B #$40     pointer Xi
        LDA A #$48     pointer memoire de copie
DEBUT   STX  $5008
        STA A $6005
        STA B $5007
        PSH B
        LDA B #$50
        PSH B
        PSH A
        PSH B
        JSR  BPCPY     transfert de Xi
        LDA B $5007
        LDA A $5005
        ADD B #$04     pointer adresse de yi
        ADD A #$04     pointer adresse de copie
        STA A $5005
        STA B $5007
        PSH B
        LDA B #$50
        PSH B          transfert de yi dans la
        PSH A          memoire de travail.
        PSH B
        JSR BPCPY
        LDA A $5005
        LDA B $5005
        SUB A #$04
        STA A $500B    sauvegarder AccA pour SPSPK
        STA B $500D    sauvegarder AccB pour SPSPK
        LDA A #$50
```

```
STA  A    $500A
STA  A    $500C
JSR    SPSPK          calcul de Ko
LDA  A    $5005
ADD  A    #$04
STA  A    $5005
LDA  B    #$18        LSB d'adresse de M1
PSH  B
LDA  B    #$50        MSB d'adresse de M1
PSH  B
PSH  A
PSH  B                transfert de Ko
JSR      SPCPY
LDA  B    #$BC        pointer adresse de H
LDA  A    $5007
ADD  A    #$04        pointer adresse de copie
STA  A    $5007
PSH  B
LDA  B    #$50
PSH  B
PSH  A
PSH  B
JSR    SPCPY          transfert de H
DEC    $504B          calcul de H/2
LDA  B  $5007
SUB  B  #$08          pointer adresse de Xi
LDA  A  #$18          pointer adresse de copie
PSH  B
LDA  B  #$50
PSH  B
PSH  A
PSH  B
JSR    SPCPY          transfert de Xi en M1
LDA  B    $5007
LDA  A  #$14
PSH  B
LDA  B  #$50
PSH  B
PSH  A
PSH  B
JSR    SPCPY          transfert de H/2 en M2
JSR    SADD           calcul de Xi + H/2
LDA  B  #$18
LDA  A    $5007
PSH  B
LDA  B  #$50
PSH  B
PSH  A
PSH  B
JSR      SPCPY        transfert de X'i = Xi + H/2
LDA  A    $5005
LDA  B    $5005
```

```
ADD   A     #$ 04
STA   A       $ 5005
PSH   B
LDA   B     #$ 50
PSH   B
PSH   A
PSH   B
JSR       SPCPY                    transfert de K0
LDX     $ 5008
DEC     (C,X)                      K0/2
LDA   B       $ 5005
SUB   B     #$ 08                  pointer adresse de yi
LDA   A     #$ 18
PSH   B
LDA   B     #$ 50
PSH   B
PSH   A
PSH   B                            transfert de yi en M1
JSR   SPCPY
LDA   B       $ 5005
LDA   A     #$ 14
PSH   B
LDA   B     #$ 50
PSH   B
PSH   A
PSH   B                            transfert de K0/2 en M2
JSR   SPCPY                        calcul de y'1 = yi + K0/2
JSR   SADD
LDA   B     #$ 18
LDA   A       $ 5007
ADD   A     #$ 04
PSH   B
LDA   B     #$ 50
PSH   B
PSH   A
PSH   B                            transfert de y'1 = yi + K0/2
JSR   SPCPY                        calcul de K1.
JSR   SPSPK
LDA   B     #$ 18
LDA   A       $ 5006
PSH   B
LDA   B     #$ 50
PSH   B
PSH   A
PSH   B                            transfert de K1
JSR   SPCPY
DEC     $ 5018
LDA   B       $ 5007               pointer adresse yi
SUB   B     #$ 04                  pointer adresse de copie
LDA   A     #$ 14
PSH   B
LDA   B     #$ 50
```

```
PSH   B
PSH   A
PSH   B
JSR     SPCPY                  transfert de y_i en M2
JSR     SADD                   calcul de y'_2 = y_i + K1/2
LDA   B     #$ 18
LDA   A       $ 5007
ADD   A     #$ 04
PSH   B
LDA   B     #$ 50
PSH   B
PSH   A
PSH   B
JSR     SPCPY                  transfert de y'_2.
JSR     SPSPK                  calcul de K2.
LDA   B     #$ 18
LDA   A       $ 5005
ADD   A     #$ 04
STA   A       $ 5005
PSH   B
LDA   B     #$ 50
PSH   B
PSH   A
PSH   B
JSR     SPCPY                  transfert de K2
LDA   B       $ 5007           pointer adresse de xi
SUB   B     #$ 08              pointer adresse de copie
LDA   A     #$ 18
PSH   B
LDA   B     #$ 50
PSH   B
PSH   A
PSH   B
JSR     SPCPY                  transfert de xi en M1.
LDA   B     #$ 3C              pointer adresse de H
LDA   A     #$ 14              pointer adresse de copie.
PSH   B
LDA   B     #$ 50
PSH   B
PSH   A
PSH   B
JSR     SPCPY                  transfert de H en M2
JSR     SADD                   calcul de x'_2 = xi + H
LDA   B   #$ 18
LDA   A     $ 5007
PSH   B
LDA   B   #$ 50
PSH   B
PSH   A
PSH   B
JSR     SPCPY                  transfert de xi + H.
LDA   B   $ 5007
```

```
SUB   B      # $ 04        pointer adresse de yi
LDA   A      # $ 18        pointer adresse de copie
PSH   B
LDA   B      # $ 50
PSH   B
PSH   A
PSH   B
JSR      SPCPY            transfert de y0 en M3.
LDA   B      $ 5005
LDA   A      # $ 14
PSH   B
LDA   B      # $ 50
PSH   B
PSH   A
PSH   B
JSR      SPCPY            transfert de K2 en M2.
JSR      SADD             calcul de y'3 = yi + K2.
LDA   B      # $ 18
LDA   A      $ 5007
ADD   A      # $ 04
PSH   B
LDA   B      # $ 50
PSH   B
PSH   A
PSH   B
JSR      SPCPY            transfert de y'3 pour SPSPK
JSR      SPSPK            calcul de K3.
LDA   B      # $ 18
LDA   A      $ 5005
ADD   A      # $ 04
PSH   B
LDA   B      # $ 50
PSH   B
PSH   A
PSH   B
JSR      SPCPY            transfert de K3
LDX      $ 5008           calcul de 2K1
INC   (12, X)                    2K2.
INC   (16, X)
LDA   B      $ 5007
ADD   B      # $ 08
LDA   A      # $ 18
PSH   B
LDA   B      # $ 50
PSH   B
PSH   A
PSH   B
JSR      SPCPY            transfert de K0 en M3.
LDA   B      $ 5005
SUB          # $ 04
LDA   A      # $ 14
PSH   B
```

```
LDA  B      #$ 50
PSH  B
PSH  A
PSH  B
JSR     SPCPY          transfert de 2K3 en M2
JSR     SADD           calcul de S2 = K0 + 2K3
LDA  B      $5005
LDA  A      #$ 14      pointer adresse de copie
PSH  B
LDA  B      #$ 50
PSH  B
PSH  A
PSH  B
JSR     SPCPY          transfert de 2K2 en M2.
JSR     SADD           calcul de S2 = S3 + 2K2.
LDA  B      $5005
ADD  B      #$ 04
LDA  A      #$ 14
PSH  B
LDA  B      #$ 50
PSH  B
PSH  A
PSH  B
JSR     SPCPY          transfert de K3 en M2.
JSR     SADD           calcul de S3 = S2 + K3
LDA  B      #$ 00      pointer adresse de 6 en VF.
LDA  A      #$ 14      pointer adresse de copie.
PSH  B
LDA  B      #$ 50
PSH  B
PSH  A
PSH  B
JSR     SPCPY          transfert de 6 en VF en M2.
JSR     SDIV           calcul de S5 /6
LDA  B      $5007
STA  B      $5005
SUB  B      #$ 04
STA  B      $5007      pointer adresse de yi.
LDA  A      #$ 14      pointer adresse de copie
PSH  B
LDA  B      #$ 50
PSH  B
PSH  A
PSH  B
JSR     SPCPY          transfert de yi en M2.
JSR     SADD           calcul de yi+1 = yi + S5 /6.
LDA  B      #$ 18
LDA  A      $5005
ADD  A      #$ 04      pointer adresse de copie.
PSH  B
LDA  B      #$ 50
PSH  B
```

```
.PSH A
 PSH B
 JSR     SPCPY                  transfert de Yi+1
 LDA B     $5007
 SUB B     #$ 04                pointer adresse de Xi
 STA B     $5007
 LDA A     #$ 48                pointer adresse de copie
 PSH A
 LDA B     #$ 50
 PSH B
 PSH A
 PSH B
 JSR     SPCPY                  transfert de Xi en M2.
 LDA B     #$ 3C                pointer adresse de H.
 LDA A     #$ 4A                pointer adresse de copie
 PSH A
 LDA B     #$ 50
 PSH B
 PSH A
 PSH B
 JSR     SPCPY                  transfert de H en M2
 JSR     SADD                   calcul de Xi+1 = Xi + H.
 LDA B     #$ 48
 LDA A     $5005
 PSH B
 LDA B     #$ 50
 PSH B
 PSH A
 PSH B
 JSR     SPCPY                  transfert de Xi+1 dans la table
                                de resultats.
 LDA B     #$ 3E                pointer adresse de XL.
 LDA A     #$ 4A                pointer adresse de copie
 PSH B
 LDA B     #$ 50
 PSH B
 PSH A
 PSH B
 JSR     SPCPY                  transfert de XL en M2.
 JSR     SSUB                   calcul de XCal - XL
 LDA A     #$ 43                mise de (XCal - XL).
 BMI     Boucle
 SWI
Boucle  LDA A     $5005          pas SUIVANT.
 LDA B     $5007
 ADD A     #$ 01
 ADD B     #$ 08
 INX     8 fois
 JMP     DEBUT
 END
```

TABLE DE DONNEES

ADRESSE	CONTENU DES MEMOIRES	ADRESSE	CONTENU des MEMOIRES
5000	SAUVEGARDER 6	500C	SAUVEGARDER ACCB
5003	en virgule flottante	500D	pour SPSPK
5004	SAUVEGARDER	5038	X_L valeur limite du domaine
5005	ACC A	503B	
5006	SAUVEGARDER	503C	H pas de La
5007	ACC B	503F	SIMULATION
5008	SAUVEGARDER Le	5040	x_0 valeur initiale
5009	registre d'INDEX	5043	
500A	SAUVEGARDER ACC A	5044	y_0 valeur initiale
500B	pour SPSPK	5047	

TABLE DE RESULTATS EN VF

5040	X_0	506C	Y_5	5098	X_{11}	50C4	Y_{16}
5044	Y_0	5070	X_6	509C	Y_{11}	50C8	X_{17}
5048	X_1	5074	Y_6	50A0	X_{12}	50CC	Y_{17}
504C	Y_1	5078	X_7	50A4	Y_{12}	50D0	X_{18}
5050	X_2	507C	Y_7	50A8	X_{13}	50D4	Y_{18}
5054	Y_2	5080	X_8	50AC	Y_{13}	50D8	X_{19}
5058	X_3	5084	Y_8	50B0	X_{14}	50DC	Y_{19}
505C	Y_3	5088	X_9	50B4	Y_{14}	50E0	X_{20}
5060	X_4	508C	Y_9	50B8	X_{15}	50E4	Y_{20}
5064	Y_4	5090	X_{10}	50BC	Y_{15}	50E8	X_{21}
5068	X_5	5094	Y_{10}	50C0	X_{16}	50EC	Y_{21}

CONCLUSION

On remarque que dans le programme principal de Runge Kutta ,on a une utilisation intensive de l'unité Am 9512 , pour les différentes opérations(addition,soustraction,multiplication et division);cela a pour conséquence ,d'augmenter la rapidité de résolution des équations différentielles et la précision des résul--tats .

Les tests avec l'opérateur Am 9512 n'ont pas été concluants par manque de temps,ce qui nous a empêchées de traiter certains exemples.

C O N C L U S I O N

§§§§§§§§§§§§§§§§§§§§§§§§§§

§§§§§§§§§§

§§

Les progrès dans la réalisation des circuits intégrés ont permis de voir apparaitre en 1980 l'unité arithmétique rapide en virgule flottante l'Am 9512.L'association du MC 6800 avec cette unité arithmétique apporte des possibilités de calcul beaucoup plus importantes que celles obtenues par programmation du MC 6800 seul.

La longueur des mots traités étant de 32 ou 64 bits,ce qui offre une grande précision,et des performances en vitesses très bonnes une addition en virgule flottante (24 bits)sur le 6800 se fait en 586 cycles,alors que l'Am 9512 met 220 cycles pour une précision de 32 bits.

Le couplage trouve une utilisation pratique dans les méthodes numériques de traitements des équations différentielles où la précision et la vitesse d'éxécution sont importantes.

Jeu d'instructions développé du 6800.

ACCUMULATOR AND MEMORY OPERATIONS	MNEMONIC	ADDRESSING MODES: IMMED OP	IMMED ~	IMMED #	DIRECT OP	DIRECT ~	DIRECT #	INDEX OP	INDEX ~	INDEX #	EXTND OP	EXTND ~	EXTND #	INHER OP	INHER ~	INHER #	BOOLEAN/ARITHMETIC OPERATION (All register labels refer to contents)	COND. CODE REG. 5 H	4 I	3 N	2 Z	1 V	0 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				$\overline{M} \rightarrow M$	⊕	⊕	↕	↕	R	S
	COMA													43	2	1	$\overline{A} \rightarrow A$	⊕	⊕	↕	↕	R	S
	COMB													53	2	1	$\overline{B} \rightarrow B$	⊕	⊕	↕	↕	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	⊕	⊕	↕	↕	↕	↕
Subract 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

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

JUMP AND BRANCH OPERATIONS	MNEMONIC	RELATIVE			INDEX			EXTND			INHER			BRANCH TEST	5	4	3	2	1	0
		OP	~	#	OP	~	#	OP	~	#	OP	~	#		H	I	N	Z	V	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 ⊕ V = 0	⊕	⊕	⊕	⊕	⊕	⊕
Branch If > Zero	BGT	2E	4	2										Z + (N ⊕ V) = 0	⊕	⊕	⊕	⊕	⊕	⊕
Branch If Higher	BHI	22	4	2										C + Z = 0	⊕	⊕	⊕	⊕	⊕	⊕
Branch If ≤ Zero	BLE	2F	4	2										Z + (N ⊕ V) = 1	⊕	⊕	⊕	⊕	⊕	⊕
Branch If Lower Or Same	BLS	23	4	2										C + Z = 1	⊕	⊕	⊕	⊕	⊕	⊕
Branch If < Zero	BLT	2D	4	2										N ⊕ 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	Advances Prog. Cntr. Only	⊕	⊕	⊕	⊕	⊕	⊕
Return From Interrupt	RTI										3B	10	1		———	———	———	———	⑩	———
Return From Subroutine	RTS										39	5	1	See special Operations	⊕	⊕	⊕	⊕	⊕	⊕
Software Interrupt	SWI										3F	12	1		⊕	S	⊕	⊕	⊕	⊕
Wait for Interrupt	WAI										3E	9	1		⊕	⑪	⊕	⊕	⊕	⊕

CONDITIONS CODE REGISTER OPERATIONS	MNEMONIC	INHER			BOOLEAN OPERATION	5	4	3	2	1	0
		OP	~	=		H	I	N	Z	V	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	⊕	⊕	⊕	⊕	⊕	⊕

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 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 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)

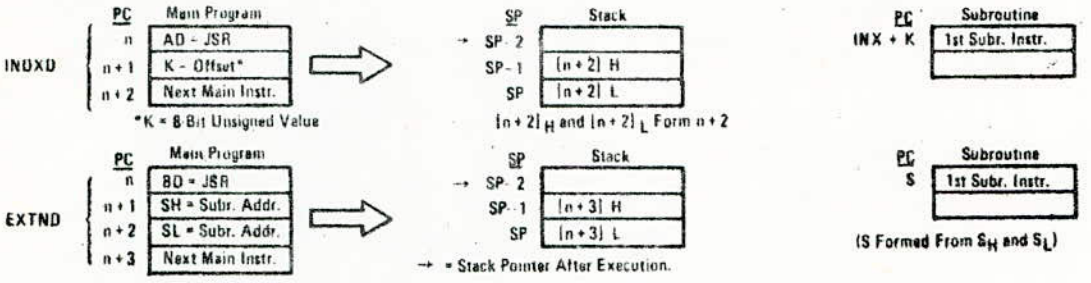
(10) (All) Load Condition Code Register from Stack. (See Special Operations)

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

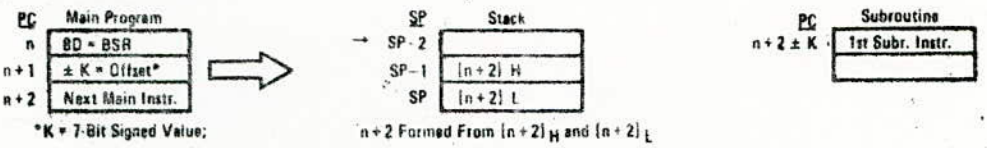
(12) (ALL) Set according to the contents of Accumulator A.

SPECIAL OPERATIONS

JSR, JUMP TO SUBROUTINE:

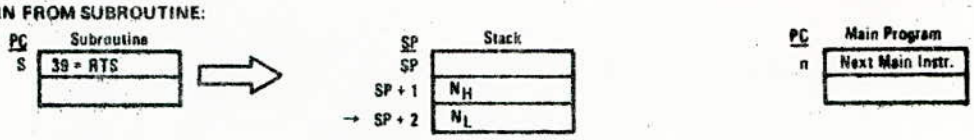


BSR, BRANCH TO SUBROUTINE:



JMP, JUMP:

RTS, RETURN FROM SUBROUTINE:



RTI, RETURN FROM INTERRUPT:

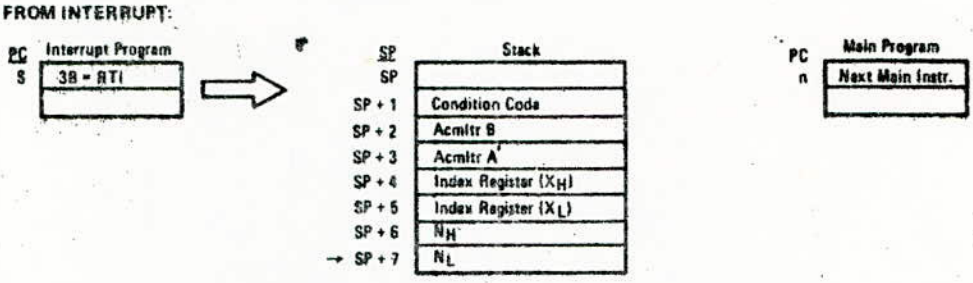


TABLE 6 – CONDITION CODE REGISTER MANIPULATION INSTRUCTIONS

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.

&& B I B L I O G R A P H I E &&

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

MICROPROCESSEURS ET MICROORDINATEURS

R.Lyon J.M. Crozet — *Masson*

EMPLOI DES MICROPROCESSEURS

Aumiaux — *Dunod*

COMPRENDRE LES MICROPROCESSEURS

Queyssac

DOCUMENTATION : ASSISTANCE MICROPRICESSEURS;MICROPROGRAMMATION

REVUES :MINI-MICROS