LABORATOIRE D'INFORMATIQUE THéORIQUE
& APPLICATIONS DE MARSEILLE
L.I.T.A.M
Département de Mathématiques-Informatique
Luminy
UNIVERSITé AIX-MARSEILLE II

ISSN 0291 - 5413

INFORMATIQUE FONDAMENTALE ET APPLICATIONS

SOMMAIRE

DéCEMBRE 1987

Adresse postale : FACULTé DES SCIENCES DE LUMINY
Mathématiques-Informatique LITAM BAT TPR 2 3e etage
Case 901 70 route Léon LACHAMP 13 288 MARSEILLE Cedex 9
91 26 90 81

EDITORIAL

e. bianco

Théatre d'ombres. Défilé de marionnettes. Pour exister la marionnette doit défiler, son inexistence d'immobilité la pousse à ce mouvement éperdu qui tend à l'agitation et à la frénésie, simulacre de vie.

Un mouvement d'opinion, une forme de pensée, sont parfaitement inexistants tant qu'ils n'ont pas provoqué le petit ballet de marionnettes qui marchent, qui scandent. L'envol de la pensée, délicate vapeur tôt diluée au soleil de la vérité, peut s'amonceler en lourds nuages noirs, quand les vents froids contraires, porteurs de lunettes sombres balayent tout avenir.

Alors dans les tourbillons qui font voler les papiers gras, les marionnettes à brandebourgs voltigent gaiement dans la grêle et la pluie.

Et puis tout semble s'apaiser...

C'est au tour des marionnettes tristes de payer les violons du bal. Et pendant ce temps-là, les marionnettes de luxe voltigent de partout sur des tapis volants grands comme des billets de banque.

Dans une orgie de parfums et de courants d'air qui jaillissent de nulle part, c'est le vent des grands sables qui sent bon le champagne chaud, dans lequel tourbillonnent tous les paris et tous les drakkars, c'est le vent des émirs lourd d'un parfum des mille et une nuits, souligné d'un relent de

pétrole; soudain d'une urne en pur cristal d'orient, jaillit la sarabande des petits bruissants billets verts.

Tout au fond de la scène, des marionnettes sombres s'agitent en silence et soufflent dans leurs mains où craquent des éclairs.

De ci, de là, dans des tourbillons frénétiques des marionnettes d'argent vif s'affolent tout autour de la corbeille.

Et dans un crescendo strident s'activent les éclairs, paroxysme de flamme. Poupées futiles, les marionnettes d'argent éclatent comme des feux d'artifice. Le vent d'est, lourd des essences de Tchernobyl projette de petits pantins rouges dans d'énormes volutes moirées.

Et le ballet des marionnettes tristes s'étire tristement sur fond de banderolles du temps des jours de fête. Seules les gaies banderolles s'agitent avec joie aux grands vents de l'espace.

De nouveau le calme est revenu.

Les marionnettes sont fatiguées. D'autres marionnettes sombres débarrassent les détritus de tous les défilés, torchent les dernières flammèches, rentrent le feu-follet du pantin inconnu.

Sur les murs clairs de la cité d'autres défilés mûrissent, claires icônes au large sourire, défilé des promesses, valse de l'avenir,

donnez-moi tout pouvoir ! et vous verrez comment je vous ferai danser !

Le théatre Guignol est d'actualité, le grand et le petit. Qui ne se réjouit de voir rosser Pandore. Mais celui-ci prudent depuis l'opération Icare, se déguise en Guignol. Il fait la politique, du beau temps fait la pluie, bonheur de la grenouille à la chaussette à clou, jette les arcs-en-ciel au fond des océans s'investit de partout, marionnette qui tire les fils des marionnettes.

Les marionnettes du silence au fond de leur caverne vivent du vent des autres , troublées par les lueurs, frémissent aux rumeurs, ombres parmi les ombres, défileront un jour et puis brûlées par l'air et le soleil, dans l'ombre reviendront , rites et soubressauts, lentement, lentement se consumer.

LA MACHINE PLATE

C.R. Subject Classification Informatics : C53 C54 D31 D41

RéSUMé

Il existe diverses façons de réaliser des processeurs dont le langage puisse être évolutif. Il en est présenté une, ici qui est originale en ce sens qu'elle ramène un programme interne au même niveau linguistique que le programme utilisable.

LA MACHINE PLATE

Il en est d'une structure de processeur comme de celle d'un programme. Et d'ailleur tel est bien le point de vue que j'adopte pour un essai d'abstraction fondamentale. C'est quelquechose qui est modelable à volonté. Bien sur, des contraintes matérielles extérieures qu'il faut bien respecter, ne permettent pas toutefois tout-à-fait n'importe quoi.

Cependant si l'on respecte quelques lois fondamentales, et c'est précisément en cela qu'elles sont fondamentales, on s'apperçoit qu'on peut faire bien des choses en variant le ton. Et c'est ce que je voudrais montrer en l'occurence en construisant un processeur, qui fonctionne comme tous les processeurs, mais qui possède en outre une petite propriété supplémentaire.

Je vais décrire un processeur à langage extensible. Normalement quand un processeur est mis sur le marché, "son langage" , je dirai les jeux de codes qu'il reconnait et qui rythment son travail, sont fixés une fois pour toutes. Et l'utilisateur se doit de "programmer" en récrivant ses algorithmes sous forme de suites de codes utilisables par le processeur.

Ce que je veux faire c'est qu'en programmant, je puisse construire tous les algorithmes dont j'ai besoin, mais en plus, accroître autant que je le désire le potentiel langage du processeur. Je veux pouvoir lui rajouter à son langage, toutes les instructions qui peuvent m'être utiles. Pour cela je me définis une méthode.

MéTHODE

Pour parvenir au résultat cherché je mets en œuvre juste ce qu'il me faut de moyens. D'abord il faut bien voir que le "processeur plat" doit être organisé pour accepter sous certaines conditions des codes qu'il aura, à partir du moment où il les a acceptés, à les utiliser exactement comme les codes qui lui sont fournis d'origine.

Il est bon de faire une remarque d'ordre terminologique. Tout processeur est organisé sur deux niveaux, en bas le niveau du calcul, et au dessus avec pour rôle d'en diriger le fonctionnement, le niveau du séquenceur. L'organe séquenceur est constitué d'une sorte de programme enregistré qui, quand il est déroulé active à chaque étape le niveau du calcul. Le programme du séquenceur est quelquefois appelé micro-programme et il est enregistré définitivement dans une mémoire spéciale.

Dans la situation que je présente ici, ce programme devient modifiable, c'est en quelque sorte comme si les deux étages que je viens de dire se retrouvaient au même niveau, d'où la dénomination de machine plate.

Visiblement la mémoire du micro-programme doit se retrouver au même niveau que celle sur laquelle travaille le processeur. L'utilisateur va se trouver dans la situation où il dispose d'un code du langage existant à l'instant où il travaille, mais il faut qu'il dispose d'un moyen pour étendre ces codes au fur et à mesure. Donc le processeur doit lui fournir ce moyen sous deux formes, un code opérateur disponible différent de ceux qui existent déjà, et une instruction d'intégration.

Pour le code opérateur, la solution que j'ai déja choisie pour le MCA-0* convient très bien, puisqu'elle consiste à en prendre pour valeur l'accès au traitement correspondant dans le programme interne.

D'une manière tout-à-fait classique, je me donne au départ un langage interne, celui qui sert à décrire précisément le programme interne qui pilote le séquenceur. Avec ce langage il me faut évidemment une structure câblée. C'est tout cela que je vais maintenant détailler.

INSTRUCTION D'INTéGRATION

L'utilisateur, lorsqu'il programme, se sert bien évidemment des codes d'instruction dont il dispose dans le langage du processeur. Imaginons qu'il veuille rajouter une instruction à ce langage, son seul moyen de communication avec le processeur est l'utilisation d'une instruction. Il en faut donc une qui serve à construire une nouvelle instruction. Je l'appelle "instruction d'intégration". Son code est très simple,

('int') (n) (c1) (c2) ... ... (cn)

quand le processeur rencontre le code 'int' ,l'information n lui indique combien d'éléments d'information suivent, qu'il transfère dans sa mémoire, après avoir attribué un code opérateur. Ce code-là est précisément l'adresse de la première case de libre dans la configuration du programme interne. Cette adresse est contrôlée par un registre spécial "mext" couplé à un visualisateur pour que l'utilisateur puisse le consulter avant sa modification. C'est cette valeur-là qui devient le code opérateur de la nouvelle instruction.

cf. Bull. d'Informatique Approfondie et Applications N° 0 et 1.

C'est donc la case mext qui contient la première adresse disponible dans la mémoire du programme interne. Et c'est ce même contenu qui devient le code de l'instruction à introduire. Le travail de mise en place de la nouvelle instruction est réalisé par le calcul suivant:

	instructions	commandes	ligne
Entrée :	cme1 := cMpc	ce1,sAB,eμb	00
	Aiguillage (me1)	etm	01
Eext :	cma1 := cpc	ca1,spc	02
	cpc := cma1 + 1	cpc,a1,un,add	03
	cm4 := cM_{pc}	c4,sAB,eμb	04
eext2 :	si cm4 = 0 vers eext1	ext1	05
	cma1 := cm4	ca1,s4	06
	cm4 := cma1 - 1	c4,a1,mun,add	07
	cma1 := cpc	ca1,spc	08
	cpc := cma1 + 1	cpc,a1,un,add	09
	cm3 := cM_{pc}	c3,sAB,eμb	0A
	cd := cmext	cd,sn	0B
	cM_d := cm3	dAB,R/W,s3,sμb	0C
	cma1 := cmext	ca1,sn	0D
	cmext := cma1 + 1	cn,a1,un,add	0E
	vers eext2	ext2	0F
eext1 :	cma1 := cpc	ca1,spc	10
	cpc := cma1 + 1	cpc,a1,un,add	11
	vers entrée	com,zéro	12

La colonne de gauche décrit les opérations à l'aide des noms des cases qui constituent l'étage calcul du processeur, tandisque la colonne de droite donne les noms des commandes qui agissent sur les circuits qui matérialisent les cases. Ce sont les deux aspects du même langage interne.

COMMUTATION INTERNE

L'adjonction d'autant d'instructions nouvelles qu'on le désire a pour conséquence une difficulté non négligeable, qui apparaît quand l'instruction nouvelle contient une condition. En effet dans un tel cas il faut un moyen pour choisir entre deux numéros de ligne pour l'instruction à suivre.

Une solution consiste à faire le calcul dans l'instruction introduite elle-même, à condition d'avoir un moyen de disposer du numéro de ligne actuel. C'est-à-dire de pouvoir utiliser une instruction qui se localise elle-même pour pouvoir calculer ensuite le numéro de ligne auquel se renvoyer si la condition est vérifiée.

La récupération de la localisation est possible grâce à la case AI qui peut avec la commande ec enregistrer l'adresse en cours sur le bus adresse du séquenceur. Les calculs d'adresse se font alors dans les cases me1 et me2 qui peuvent attaquer directement en entrée de L1 à cause des commandes etm, et etd et cond.

La case me1 est chargée au contenu de AI sur une étiquette, de telle sorte que parvenu en fin de boucle, il suffit de prendre pour numéro de ligne suivant le contenu de me1.

Par contre sur une condition descendante, il faut calculer à l'avance le numéro de ligne auquel se renvoyer sur la condition. Le calcul se fait en me2.

Exemple:

```
                cma1 := cAI
                cmA := cma1 + 10
                cma1 := cmA
                cmA := cma1 + 5
                cme2 := cmA
ex2 :           cme1 := cAI
                si <condition> vers c(cme2)    etd (2 points
                I1             (aller à ex1)   d'attaque)
                I2
                I3
                I4
                I5
                I6
                I7
                vers c(me1)    (aller à ex2)   etm (2 points
ex1 :           I9                             d'attaque)
                etc...
```

HORLOGE

C'est l'horloge, ou plutôt l'ensemble des signaux qui constituent

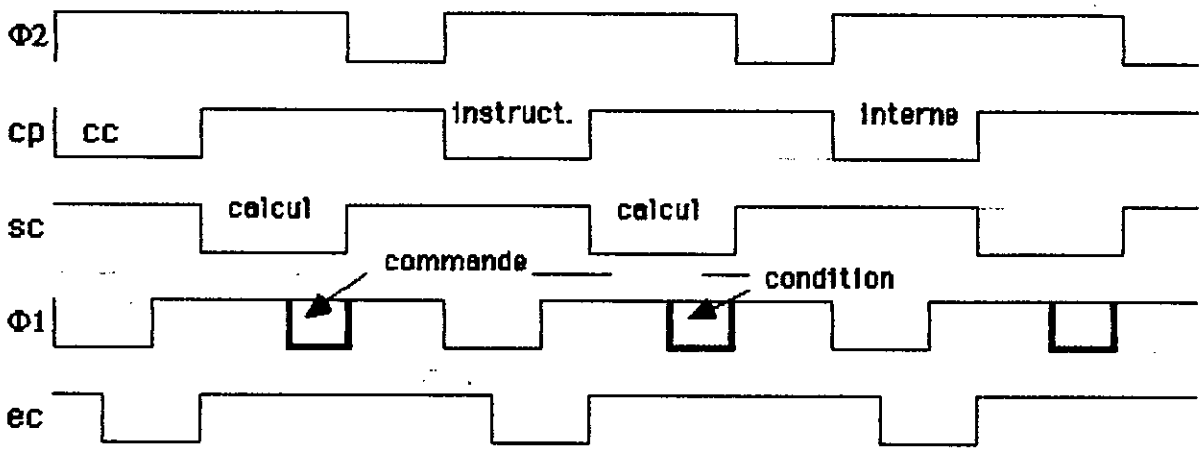
l'horloge qui servent à l'ordonnancement des tâches dans le temps. Ainsi, Φ2 quand elle est haute, commande le temps global de calcul qui se subdivise en deux parties, la première, contrôlée par le signal cp,cc et pendant laquelle se fait en mémoire la récupération du code instruction, qui est logé en LC, la seconde contrôlée par sc, et pendant laquelle le groupe de prom GP attaqué en adresse par le contenu de LC, émet les commandes du calcul.

En un cycle la mémoire est attaquée successivement deux fois. La première pour en extraire le code de l'instruction interne, qui est choisi pour être un numéro de ligne dans le groupe GP. La seconde lorsque ce qu'émet le groupe GP sous forme du jeu de commandes du calcul, maintient en état de calcul l'étage de calcul. Dans cette partie, cependant la mémoire ne sera effectivement sollicitée que si l'instruction interne déroulée fait un appel mémoire sous l'effet de l'un des trois registres adresse pc, d et P.

Par exemple: cma1 := cMpc ou cma1 := cMP ou encore cm1 := cMd .

Le signal ec active l'enregistrement en LC du code de l'instruction interne qui est émis par la mémoire.

Le signal Φ2 sert à enregistrer le numéro de ligne suivant en L1. Ici je fais l'hypothèse qu'il est créé par l'addition de 1 au travers de AS à partir de l'adresse en cours. Dans le cas du déroulement d'une instruction de condition telle que: si cmA = 0 vers c(me2) ou une simple commutation comme: vers c(me1) le numéro de ligne suivant doit venir d'ailleurs, par exemple des cases me1 ou me2. Mais alors il faut récupérer l'information dans la deuxième phase, celle du calcul d'où la nécessité de superposer au signal Φ1 la partie notée en trait renforcé qui



P4

ne doit apparaitre que dans ces cas-là. C'est le signal rΦ1 qui apporte cette perturbation.

INSTRUCTIONS INTERNES

Les diverses cellules de l'étage de calcul, qu'il s'agisse des buffers ou des latches sont commandées chacune par un signal qui est issu des prom GP. Chaque signal est repéré par un nom. La liste de ces noms est la suivante:

eμb	sμb	sAl	sn	cn
zéro	ca2	a2	sa2	sm2
un	mun	cinq	dix	deux
ca1	a1	sa1	ce1	se1

ce2	se2	c4	s4	ext1
c3	s3	c2	s2	c1
s1	cA	sA	add	cω
cd	sd	dAB	cpc	spc
sAB	R/W	cP	sP	PAB
cond				

Le principe de fonctionnement de l'étage de calcul est simple chaque instruction correspond à la mise en communication de deux cellules mémoires. Qu'il s'agisse d'une communication de latch à latch, de latch à cellule de mémoire centrale, ou l'inverse. La mémorisation dans un latch se fait sous l'action de sa commande, ainsi pour me1 c'est la commande ce1. L'information est supposée stabilisée en entrée, au préalable. Par contre, à partir du moment où un latch a acquis une information il la réémet en permanence sur sa sortie. C'est la raison pour laquelle en sortie de latch on met un buffer pour l'isoler du bus. Seul le latch devant émettre, aura son buffer de sortie excité.

Je donne ci-dessous quelques exemples d'instructions internes, c'est-à-dire en fait quelques éléments de calcul interne, avec la liste des commandes qui doivent être activées pour que le calcul se réalise.

Pour chaque élément de calcul, les commandes sont émises par la batterie de proms GP, avec au niveau actif les commandes qui assurent les échanges d'information entre les cellules désignées, et au niveau inactif toutes les autres commandes afin que rien ne se passe entre les autres cellules.

cma1 := cAl sAl,ca1 cmA := cma1+10 dix,a1,cA

cma1 := cmA	ca1,sA	cme2 := cmA	ce2,sA
cme1 := cAl	sAl,ce1	cm3 := cMd	eμb,c3,dAB
cma1 := cd	ca1,sd	cd := cma1+1	un,a1,add,cd
cma2 := cm3	ca2,s3	cd := cma1 + cma2	a2,a1,add,cd
cMd := cm3	sμb,s3,dAB,R/W	cma1 := cd	ca1,sd
cd := cma1_1	mun,a1,cd	cma1 := cpc	ca1,spc
cpc := cma1+1	un,a1,cpc	cm2 := cP	c2,sP
cme2 := cma1+10	dix,a1,ce2	cma1 := cme2	ca1,se2
cma2 := cMpc	eμb,ca2,sAB	cma1 := cMP	eμb,ca1,PAB
cmA := cma1-cma2	sm2,a1,cA	cmA := cma1+1	un,a1,cA
vers c(me1)	etm	si cmA = 0 vers c(me2)	sA,cond

Table des codes des instructions internes.

Ces codes sont des numéros de ligne dans le groupe de prom GP. La ligne désignée par le code contient le jeu de commandes qui assure le calcul.

Instruction	Code	Instruction	Code
cme1 := cma1+1	00	cme1 := cma1+2	01
cme1 := cma1+3	02	cme1 := cma1+5	03
cme1 := cma1+10	04	cme2 := cma1+1	05
cme2 := cma1+2	06	cme2 := cma1+3	07
cme2 := cma1+5	08	cme2 := cma1+10	09
cmA := cma1+1	0A	cmA := cma1+$_C$1	0B
cmA := cma1+2	0C	cmA := cma1+3	0D
cmA := cma1+5	0E	cmA := cma1+10	0F
cma1 := cAl	10	cme1 := cAl	11

cd := cmext	12	cma1 := cmext	13
cmext := cma1+1	14	cmA := cma1+cma2	15
cmA := cma1+$_C$cma2	16	cpc := cMd	17
cpc := cMP	18	vers c(me1)	19
cma1 := cme1	1A	cmA := cm1	1B
cmA := cm2	1C	cmA := cm3	1D
cmA := cm4	1E	cm1 := cmA	1F
cm2 := cmA	20	cm3 := cmA	21
cm4 := cmA	22	cm1 := cP	23
cm1 := cd	24	cm1 := cpc	25
cm2 := cP	26	cm2 := cd	27
cm2 := cpc	28	cm3 := cP	29
cm3 := cd	2A	cm3 := cpc	2B
cm4 := cP	2C	cm4 := cd	2D
cm4 := cpc	2E	cma1 := cmA	2F
cma1 := cP	30	cma1 := cd	31
cma1 := cpc	32	cma1 := cm1	33
cma1 := cm2	34	cma1 := cm3	35
cma1 := cm4	36	cma1 := cme2	37
cma1 := cMP	38	cma1 := cMpc	39
cma2 := cm1	3A	cma2 := cm2	3B
cma2 := cm3	3C	cma2 := cm4	3D
cma2 := cMP	3E	cma2 := cMd	3F
cma2 := cMpc	40	cd := cMpc	41
cm1 := cMpc	42	cm2 := cMpc	43
cm3 := cMpc	44	cm4 := cMpc	45
cm1 := cMd	46	cm2 := cMd	47

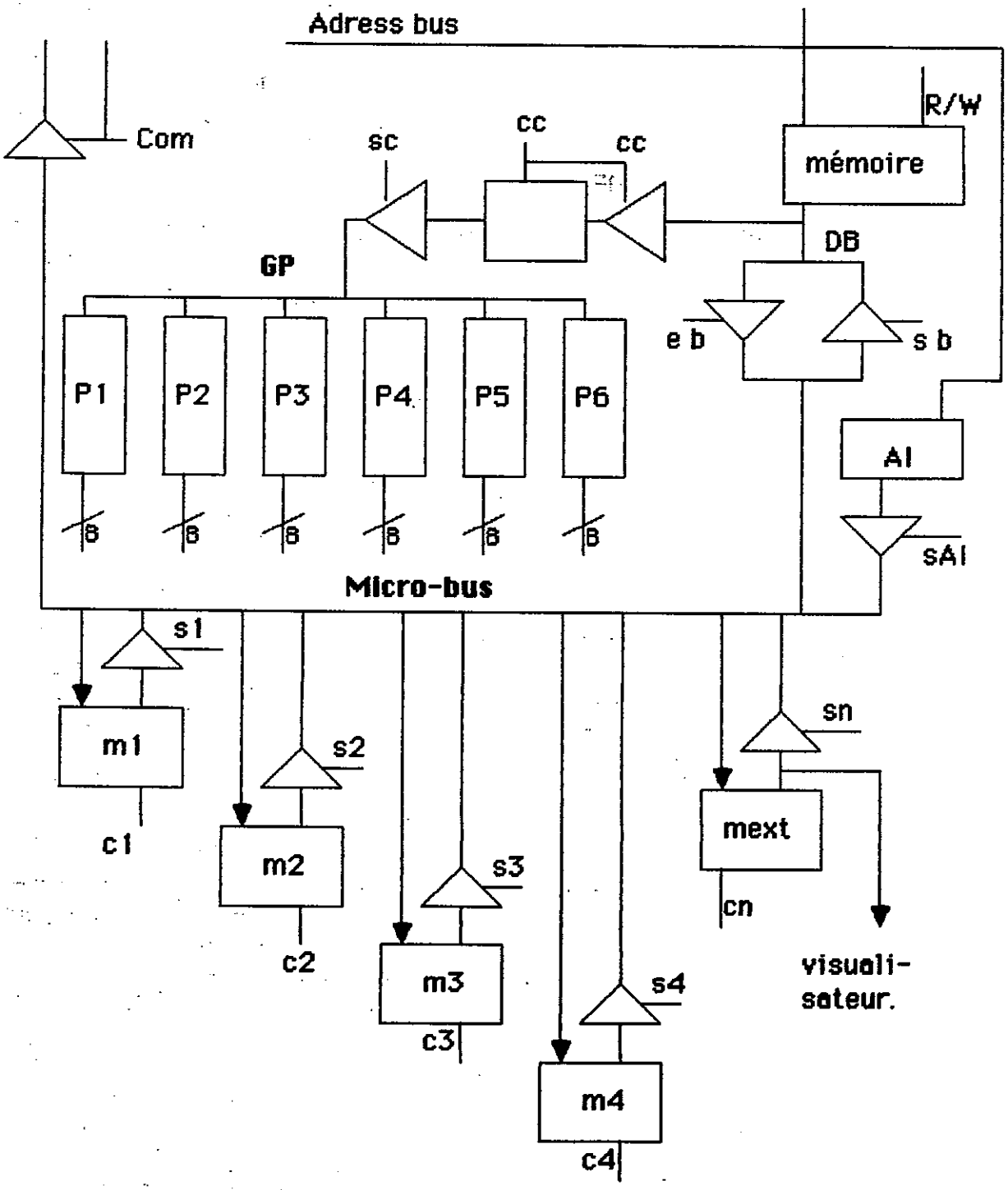
cm3 := cMd	48	cm4 := cMd	49
cP := cm1	4A	cd := cm1	4B
cpc := cm1	4C	cP := cm2	4D
cd := cm2	4E	cpc := cm2	4F
cP := cm3	50	cd := cm3	51
cpc := cm3	52	cP := cm4	53
cd := cm4	54	cpc := cm4	55
cP := cma1+1	56	cd := cma1+1	57
cpc := cma1+1	58	cm1 := cma1+1	59
cm2 := cma1+1	5A	cm3 := cma1+1	5B
cm4 := cma1+1	5C	cm1 := cma2	5D
cm2 := cma2	5E	cm3 := cma2	5F
cm4 := cma2	60	cMP := cma2	61
cMd := cma2	62	cMpc := cma2	63
cMd := cm1	64	cMd :=cm2	65
cMd := cm3	66	cMd := cm4	67
cMd := cma1	68	cMd := cpc	69
cd := cma1+$_C$cma2	6A	cpc := cma1+$_C$cma2	6B
cd := cma1+cma2	6C	cpc := cma1+cma2	6D
cP := cma1+$_C$ 1	6E	cd := cma1+$_C$ 1	6F
cpc := cma1+$_C$ 1	70	cP := cma1+2	71
cd := cma1+2	72	cpc := cma1+2	73
cm1 := cma1+$_C$ 1	74	cm2 := cma1+$_C$ 1	75
cm3 := cma1+$_C$ 1	76	cm4 := cma1+$_C$ 1	77
cd := cma1+3	78	cMpc := cm1	79

cMpc := cm2	7A	cMpc := cm3	7B
cMpc := cm4	7C	cm1 := cma1+$_C$ cma2	7D
cm2 := cma1+$_C$ cma2	7E	cm3 := cma1+$_C$ cma2	7F
cm4 := cma1+$_C$ cma2	80	cm1 := cma1+cma2	81
cm2 := cma1+cma2	82	cm3 := cma1+cma2	83
cm4 := cma1+cma2	84	si cmA=0 vers c(me2)	85
cm1 := cma1+3	86	cmA := cMpc	87
cmA := cMd	88	cmA := cMP	89
cd := cma1+cma2	8A	cpc := cma1+cma2	8B
cP := cma1+cma2	8C	vers entrée	8D

Quelques remarques concernant les notations, l'écriture "cM" signifie qu'on s'adresse à une case de la mémoire centrale dont l'adresse est contenue dans le registre dont le nom est désigné à la suite. Ainsi, cMpc désigne la case de mémoire dont l'adresse est dans le registre pc.

L'opérateur arithmétique "+$_C$" indique que l'opérande qui suit est complémenté par rapport à 1 avant d'être introduit dans l'additionneur.

Il me faut rajouter quelques commentaires complémentaires sur l'intérêt des prom GP. Pour surmonter les diverses difficultés qui se posaient à moi j'ai dû accroître sensiblement la quantité de registres dans l'étage du calcul. En conséquence directe, le nombre des commandes a cru. L'image interne de chacune des instructions du séquenceur comporte ainsi un grand nombre de bits. Quarante six en l'occurence, ce qui exige six octets pour l'enregistrer. Noter cette information directement en mémoire, serait prohibitif à la fois et mal commode à manipuler, car il faudrait soit disposer d'un bus à quarante huit moments pour conduire les signaux sur



P1 **En sortie de mémoire par le Data Bus, attaque en adresse des 6 prom émettrices des commandes du circuit de calcul. Le registre mext fournit le nouveau code instruction sur le visualisateur.**

les commandes de registres, soit les collecter par une boucle.

Il est plus simple de les coder sur huit bits car alors l'information tient sur un octet. Et le choix du code est tel que, considéré comme un numéro de ligne dans un jeu de prom ,quand il lui est fourni en adresse ce dernier émet sur quarante six bits l'ensemble de signaux correspondant.

LANGAGES DE LA MACHINE PLATE

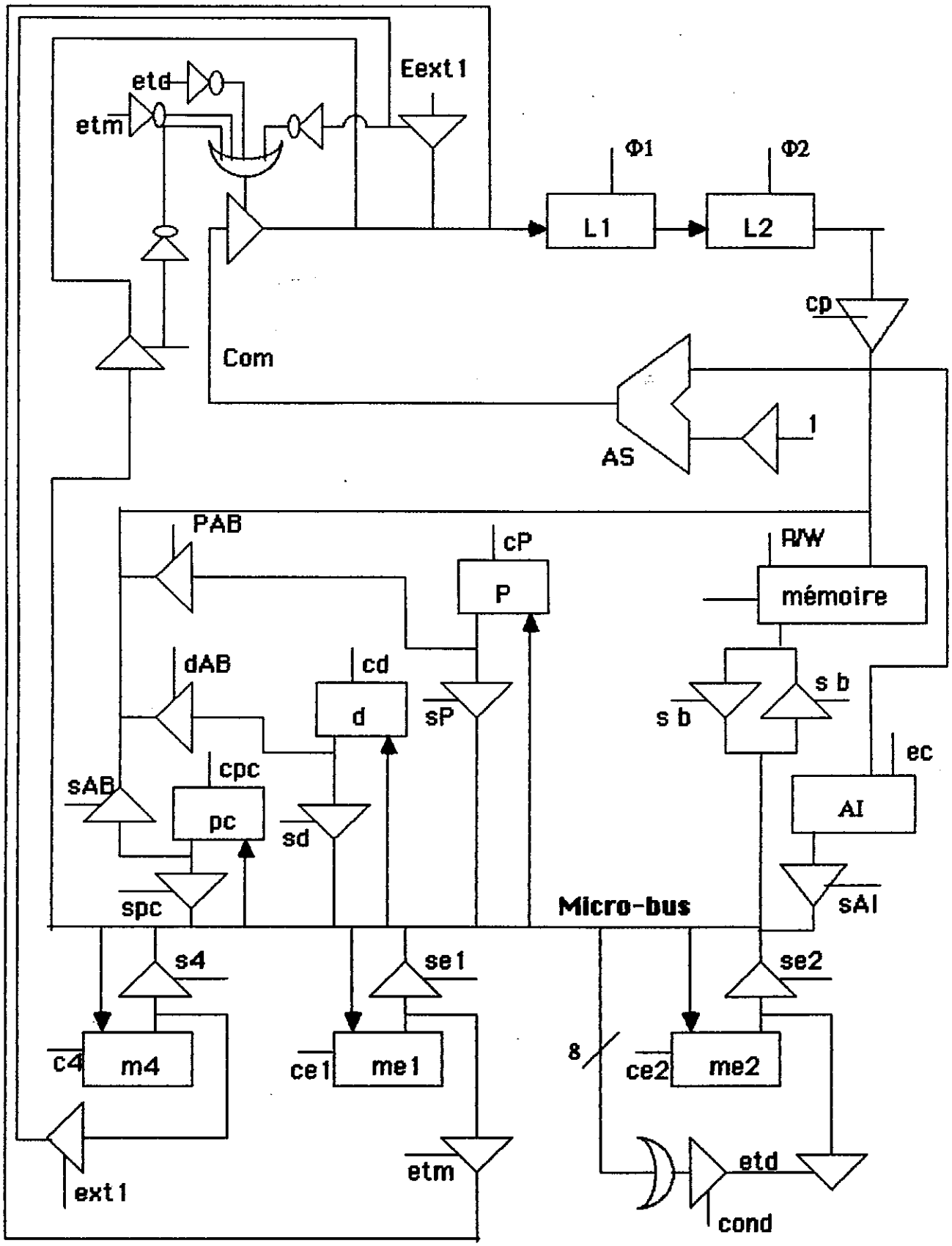
Avec une telle construction, on peut pratiquement rajouter les instructions que l'on veut. Il est ainsi possible de munir cet organe de plusieurs langages différents, tous immédiatement disponibles.

A titre d'exemple je vais montrer comment construire deux langages à sémantique très différente, si la forme en est voisine.

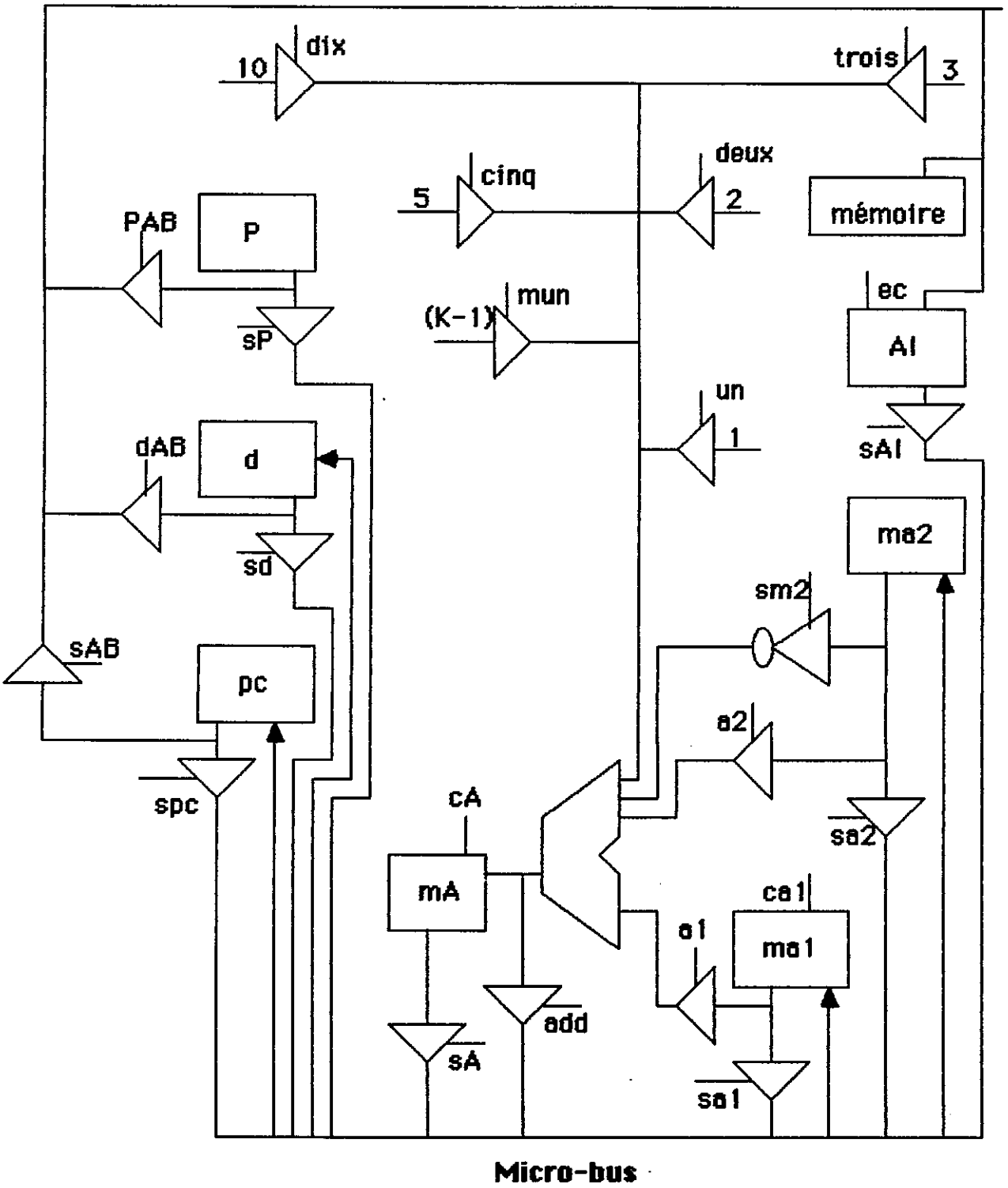
Un premier exemple est une machine de Nolin à laquelle je rajoute une sorte d'index, et quatre instructions supplémentaires. L'autre cas est présenté avec un adressage relatif et l'index qui sert à repérer une origine. Bien que formellement proches, comme on va le voir, ces langages présentent des différences importante de structure. Pour le second je construis une instruction d'insertion de procédure puissante, moyennant une convention forte sur l'organisation des configurations.

LANGAGE ABSOLU

J'envisage un langage très simple, avec adressage absolu qui comporte les cinq instructions de la machine de Nolin. Je rajoute une case spéciale X



P2 **Circuit pilote avec la mémoire connectée au data bus et à l'adress bus. Les 3 registres d'émission adresse, pc d et P et les registres de calcul d'adresse du programme interne AI, me1 et me2.**



P3. Additionneur et registres de calcul ma1, ma2 et mA. Registre adresse du programme interne AI.

qui joue un peu le rôle d'un index et qu'on peut charger comme n'importe quelle autre case, en utilisant les deux instructions

cX := ca ca := cX

on peut atteindre à une case de la mémoire par l'intermédiaire du contenu de X, il faut utiliser les instructions:

ca := cM cM := ca

la première place dans la case a le contenu de la case dont l'adresse est dans X, la seconde met dans la case d'adresse contenue en X ce qu'il y a en case a.

On dispose également des instructions classiques :

ca := 0 ca := cb ca := ca $+_K$ 1

si ca = 0 vers ei vers ej

Je ne traiterai ici que le cas des quatre instructions relatives à l'index. Les autres peuvent être facilement reconstituées.

La première:

ca := cM

si je me réfère à l'instruction d'intégration, je constate que son code-opérateur sera obligatoirement 13 , puisque tel est le numéro de la ligne disponible à la suite dans le programme interne. En mémoire le code complet de l'instruction tient en deux cases :

(13),(a)

et, bien entendu je suppose qu'au moment du traitement le contenu de pc pointe sur la première de ces deux cases :

	Instruction	Code	Ligne
Mena :	cma1 := cpc	32	13
	cpc := cma1 + 1	70	14

	cm1 := cMpc	42	15
	cm2 := cMd	47	16
	cpc := cm1	4C	17
	cMpc := cm2	7A	18
	cpc := cma1 + 2	73	19
	vers entrée	8D	1A

C'est d qui joue le rôle de X.

Instruction

cM := ca

aenM :	cma1 := cpc	32	1B
	cpc := cma1 + 1	70	1C
	cm1 := cMpc	42	1D
	cpc := cm1	4C	1E
	cm1 := cMpc	42	1F
	cMd := cm1	64	20
	cpc := cma1 + 2	73	21
	vers entrée	8D	22

Instruction

cX := ca

aenX :	cma1 := cpc	32	23
	cpc := cma1 + 1	70	24
	cm1 := cMpc	42	25
	cpc := cm1	4C	26

cd := cMpc	41	27
cpc := cma1 + 2	73	28
vers entrée	8D	29

LANGAGE RELATIF

Reprenant les mêmes types d'opérations je définis ce coup-ci un adressage relatif. Pour cela je détermine une structuration de la configuration qui représente une convention de programmation. D'abord l'origine est considérée comme repérée par le contenu de l'index. Ensuite les adresses sont calculées par rapport à cette origine, et enfin comme dans le cas de la Procédure formelle, les trois premières cases contiennent toujours dans l'ordre, le volume de la configuration actuelle, le volume de la configuration précédente, et l'adresse de retour dans l'algorithme de la procédure précédente. La deuxième et la troisième de ces valeurs sont mises en place pendant le déroulement de l'instruction d'insertion, et la première pendant le déroulement de l'instruction de déclaration cf.[bibliog.]

Je ne traiterai à titre d'exemple que le cas de l'instruction d'insertion et de son environnement. Elle se présente sous la forme:

insérer P (a1 , a2 , a3 , ... , an)

dont j'adopte le code :

('ins'),(nom P),(n),(a1),(a2),(a3), ... ,(an)

	Instructions	Codes	Lignes
Ins :	cm3 := cMd	48	2A

	cma1 := cd	31	2B
	cd := cma1+1	57	2C
	cma1 := cd	31	2D
	cma2 := cm3	3C	2E
	cd := cma1+cma2	8A	2F
	cMd := cm3	66	30
	cma1 := cd	31	31
	cd := cma1+$_C$1	6F	32
	cma1 := cpc	32	33
	cpc := cma1+1	58	34
	cm2 := cP	26	35
	cma1 := cAI	10	36
	cme2 := cma1+10	09	37
	cma1 := cme2	37	38
	cme2 := cma1+5	08	39
	cma2 := cMpc	40	3A
Ins2 :	cme1 := cAI	11	3B
	cma1 := cMP	38	3C
	cmA := cma1+$_C$cma2	16	3D
	cma1 := cmA	2F	3E
	cmA := cma1+1	0A	3F
	si cmA=0 vers c(me2)	85	40
	cma1 := cP	30	41
	cP := cma1+2	56	42
	vers c(me1)	19	43
Ins1 :	cma1 := cP	30	44

	cP := cma1+1	56	45
	cm1 := cd	24	46
	cma1 := cpc	32	47
	cpc := cma1+1	58	48
	cm4 := cMpc	45	49
	cmA := cm4	1E	4A
	cma1 := cAl	10	4B
	cme2 := cma1+10	09	4C
	cma1 := cme2	37	4D
	cme2 := cma1+10	09	4E
	cma1 := cd	31	4F
	cd := cma1+3	78	50
Ins4 :	cme1 := cAl	11	51
	cma1 := cpc	32	52
	cpc := cma1+1	58	53
	si cmA=0 vers c(me2)	85	54
	cma1 := cMpc	39	55
	cma2 := cm3	3C	56
	cmA := cma1+$_C$cma2	16	57
	cma1 := cmA	2F	58
	cmA := cma1+1	0A	59
	cMd := cmA	8E	5A
	cma1 := cd	31	5B
	cd := cma1+1	57	5C
	cma1 := cm4	36	5D
	cmA := cma1-1	0B	5E
	cm4 := cmA	22	5F

	vers c(me1) (Ins4)	19	60
	_____		61
	_____ 2 lignes vides		62
Ins3 :	cma1 := cm1	33	63
	cd := cma1+2	72	64
	cMd := cpc	69	65
	cd := cm1	4B	66
	cpc := cMP	18	67
	cP := cm2	4D	68
	vers entrée	8D	69

Comme en procédure formelle il faut une instruction de déclaration qui joue le rôle de 'début' et à la fois sert à initialiser la configuration. Le code en est:

('proc'),(n),(m),(v1),(v2), ... ,(vm)

avec n le nombre de paramètres, m le nombre de variables locales, et vi le volume de chacune d'entre elles.

	Instructions	Codes	Lignes
Décl :	cma1 := cpc	32	6A
	cpc := cma1+1	58	6B
	cma1 := cMpc	39	6C
	cm1 := cma1+3	86	6D
	cma1 := cpc	32	6E
	cpc := cma1 +1	58	6F
	cmA := cMpc	87	70
	cma1 := cmA	2F	71

	cma2 := cm1	3A	72	
	cm2 := cma1+cma2	82	73	
	cm3 := cd	2A	74	
	cma1 := cd	31	75	
	cma2 := cm1	3A	76	
	cd := cma1+cma2	8A	77	
	cma1 := cAl	10	78	
	cme2 := cma1+10	09	79	
	cma1 := cme2	37	7A	
	cme2 := cma1+10	09	7B	
Deb2 :	cme1 := cAl	11	7C	
	si cmA=0 vers c(me2)	85	7D	(vers Deb1)
	cMd := cm2	65	7E	
	cma1 := cd	31	7F	
	cd := cma1+1	57	80	
	cma1 := cpc	32	81	
	cpc := cma1+1	58	82	
	cma1 := cMpc	39	83	
	cma2 := cm2	3B	84	
	cm2 := cma1+cma2	82	85	
	cma1 := cmA	2F	86	
	cmA := cma1+$_C$1	0B	87	
	vers c(me1) (Deb2)	19	88	
Deb1 :	cd := cm3	51	89	
	Trois lignes vides			
	cMd := cm2	65	8D	

cma1 := cpc	32	8E
cpc := cma1+1	58	8F
vers entrée	8D	90

C'est l'instruction 'fin' qui complète le jeu de l'insertion. Sur cette instruction, l'indicateur de configuration régresse de la quantité contenue dans la deuxième case de la configuration actuelle, et l'adresse notée dans la troisième case pendant l'insertion, devient l'adresse de la nouvelle instruction à dérouler.

	Instructions	Codes	Lignes
Fin :	cma1 := cd	31	91
	cd := cma1+1	57	92
	cma2 := cMd	3F	93
	cd := cma1+2	72	94
	cpc := cMd	17	95
	cd := cma1+$_C$cma2	6A	96
	cma1 := cd	31	97
	cd := cma1+1	57	98
	vers entrée	8D	99

PROGRAMMATION DE LA MACHINE PLATE

Les codes opérateurs de chacune des instructions des deux langages présentés sont fournis par la mise en place des programmes internes qui leur correspondent.

L'instruction commune, l'intégration est de code : 02

ca := cM :	13
cM := ca :	1B
cX := ca :	23
insertion	2A
déclaration	6A
fin	91

Les autres instructions, si l'on introduisait leur traitement interne se placeraient à partir de 9A. Pour pouvoir introduire l'instruction :

ca := cM il faut construire le code :

(02)(8)(32)(70)(42)(47)4C)(7A)(73)(8D)

où (02) est le code de l'instruction d'intégration, et (8) le nombre d'instructions internes. C'est le processeur qui effecte automatiquement (13) comme code opérateur à cette instruction de telle sorte que si l'on désire coder l'exemple:

c125 := cM on obtient (13)(125)

Dans le cas de:

cM := ca son introduction se fait avec le code :

(02)(8)(32)(70)(42)(4C)(42)(64)(73)(8D)

et une instruction:

cM := c19 se coderait :

(1B)(19)

Pour l'insertion, le code serait plus long:

(02)(64)(48)(31)(57)(31)(3C)(8A)(66)(31)(6F) ....

(72)(69)(4B)(18)(4D)(8D)

quand à fin :

(02)(9)(31)(57)(3F)(72)(17)(6A)(31)(57)(8D)

En manière de conclusion, je peux dire que voila une autre façon de concevoir un processeur à langage variable. Une première technique consistait à enregistrer le programme interne sur une mémoire RAM spéciale qui remplace les classiques PROM ou les réseaux logiques. Mais cela obligeait à disposer à côté, d'un autre petit ordinateur pour le démarrage. Ici on peut concevoir d'utiliser une mémoire CMOS avec batterie auxiliaire, de manière à ne pas perdre l'information.

BIBLIOGRAPHIE

Bulletin d'Informatique Approfondie et Application N° 0 et 1 de mars et juin 1981 : Le MCA-0

Bull. d'Inf. Appr. et Appl. N°9 de décembre 1984 : La machine universelle de la procédure formelle.

Bull. d'Info. Appr. et Appl. N°10 de mars 1985 : L'ordinateur formel.

VOUZZAVEDIBISAR

Tout. Tout est dans tout. Que pourrait-il faire d'autre ? Rien!
Mais alors si tout est dans tout , il y a rien en dehors de tout. Et ainsi quelquechose est en dehors de tout et c'est rien.

Et tout n'est plus tout puisque rien n'est pas dans tout, car si rien n'était dans tout, tout serait en dehors de tout et plus du tout dans tout. Tout serait en quelquesorte le contraire de tout , et tout ça à cause de rien. Donc il n'en faut pas beaucoup pour destabiliser tout.

Revenons au tout est dans tout. Puisque tout est dans tout, tout et son contraire sont dans tout. Dans tout on a le blanc et le noir, le grand et le petit, le gros et le maîgre, Gault et Millau, Bouvart et Ratinet, la croix et le bannière, le vide et le plein.

Tout est-il plein et rien est-il vide ?
Le contraire de tout est-il rien ? mais alors, rien est dans tout, et tout n'est plus tout ... on ne peut plus se fier à rien.