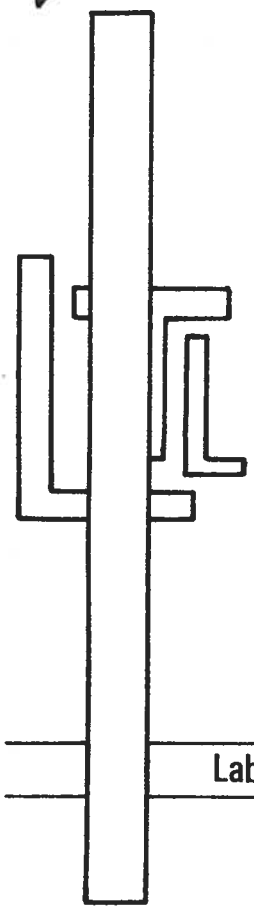
Publication du Département de Mathématique & d´Informatique

BULLETIN d'INFORMATIQUE

approfondie & applications



Nº 0

Laboratoire d´Informatique

Avant-Propos.

Le plus simple, nous semble-t-il, pour présenter un nouveau bulletin est encore d'en réaliser un premier numéro. N'en déplaise aux jeteurs de sort, nous essayerons de faire en sorte qu'il y en ait un second. Au moins. Pourquoi vouloir à tout prix essayer d'expliquer, essayer d'introduire : ce bulletin existe et s'il doit vivre il vivra, sinon tant pis.

Nous pourrions dire, par exemple, que nous sommes un tout petit groupe rassemblé autour de cette tâche, beaucoup plus par intérêt, même passionné (oui pourquoi pas) pour la chose réalisée, que par un fait administratif précis. Il y a parmi nous, n'ayons pas honte, des gens qui ne sont pas Informaticiens de métier, qui doivent donc faire autre chose pour vivre. D'autres qui sont bien loin, vers les marches de l'est, et qui, malgré tout, apportent leur petite contribution.

Mais pourquoi répéter toutes ces choses. Pourquoi redire que la recherche présente toujours son petit côté folklorique.

Nous avons essayé, au cours des temps, d'amasser quelques brins de savoir, d'accumuler quelques bribes de savoir-faire, sans trop jamais nous préoccuper d'engranger la vraie moisson, cette manne qui tombe du ciel, ce crédit que nous accorde la Capitale, cette faveur.

Aussi ne peut-il être question ici d'exhaler une quelconque plainte à propos de la modicité de nos moyens. Pourquoi arguer de notre éloignement de Paris pour pouvoir baisser les bras, en justifiant, foi maligne, du calme provincial loin de la vraie vie active. Nous n'avons pas arpenté les trottoirs du pouvoir, les gazettes ne nous ont pas spontanément ouvert leurs colonnes. Alors nous essayons d'ouvrir les nôtres.

Nous aurions pu effectivement dire tout cela mais il n'est nullement dans notre intention de tenir de tels propos, bien trop pessimistes, alors qu'une tentative comme la nôtre ne réclame qu'optimisme, beaucoup d'optimisme.

Qu'aimerions-nous dire ? Nous aimerions noter, pour nous en souvenir, que, depuis l'invasion du marché par le micro-processeur, on a un peu tendance à oublier que les problèmes fondamentaux de l'informatique demeurent. Ce n'est pas parce que l'on possède un ordinateur au même titre qu'une 2 CV, que l'écriture d'un compilateur s'en trouve simplifiée. Ce n'est pas parce que l'ordinateur est sorti de la forteresse du pouvoir et de l'argent, pour se retrouver simple jouet dans la hôte du Père Noël, qu'il devient inutile de se poser des problèmes fondamentaux.

C'est ainsi que, passée la courte période libérale pendant laquelle l'ordinateur était vendu nu, donc était potentiellement totalement utilisable, on recommence à trouver sur le marché ces O.S. (operating systems) frappés du sceau du secret, verrouillés destinés à tout contrôler, énormes, qui se mélangent sournoisement aux compilateurs qu'ils sont prétendus gérer. Et qui montrent seulement que l'expérience acquise a beaucoup de mal à resservir.

Et pourtant...

Et si nous pensons qu'il existe une large catégorie de problèmes indépendants de l'aspect micro ou macro du processeur, et que ces problèmes sont ceux de l'informatique, nous pensons également qu'il existe une part d'entre eux qui peuvent et doivent être soumis à l'analyse mathématique ; mais cela c'est déjà de l'épiinformatique. Quant à nous, nous nous préoccupons plus modestement de l'informatique qui contient les moyens de résoudre ses propres problèmes, cette informatique existe, nous aussi nous l'avons rencontrée et c'est elle que nous essayerons de faire parler dans ces lignes.

Il existe une certaine informatique, il existe un cadre où la décrire, il existe un projet, il existe un programme qui, et ensuite après avoir montré qu'il existe un être qui, le chercheur devient trouveur, et trouve des utilisations à ses trouvailles. C'est toute une méthodologie, tout un programme.

Voilà bien des choses que nous aurions pu dire également si nous n'étions pas si respectueux des instants de nos lecteurs, aussi nous limiterons-nous à l'essentiel.

La Rédaction

Nos Séminaires.

Fidèles à notre habitude, cette année encore nous organisons tous les mercredis de 15H jusqu'à l'heure de l'apéritif des réunions de travail.

Ces séminaires, puisque tel est le nom de ces réunions dont les thèmes consacrés à l'Informatique sont très diversifiés, permettent à des personnes intéressées par les différents sujets que nous proposons de venir apprendre mais aussi et surtout discuter.

C'est ainsi que depuis le début de l'année scolaire nous avons traité de la théorie des compilateurs ; sujet qui a d'ailleurs fait l'objet de discussions parfois très explosives mais ô combien profitables.

L'intérêt a été tel qu'à l'heure actuelle notre petite équipe est en train de construire le compilateur d'un langage simple dont la réalisation débouchera sur l'élaboration des critères de "bon choix" dans la construction de compilateurs.

Parallèlement à ce travail, nos amis des Mesures Physiques de l'I.U.T. de St Jérôme ont présenté deux cartes Texas Instruments : le microprocesseur 16 bits TMS 9900 et sa carte interface programmable TMS 9901.

Les prochains séminaires nous permettront d'étudier les thèmes suivants :

- Poursuite de la construction du compilateur précédemment cité jusqu'à son implantation sur le microprocesseur 16 bits TMS 9900.
- Théorie des systèmes.
- Conduite de processus : grafcet, Réseaux de Petri, Automates programmables.
  A ce sujet, nous souhaitons à notre ami Fumanal, victime d'un accident de ski, un prompt rétablissement et nous espérons que sa convalescence lui permettra de rédiger un petit article dans ce domaine où il excelle.

- L'Informatique à la Banque.

Avant de tourner la page, nous rappelons que ces réunions sont ouvertes à tous. L'information concernant la date et le sujet de chaque séminaire se fait par affichage au Département de Mathématique-Informatique de LUMINY.
Si vous êtes intéressé ... Venez. Notre petit groupe ne demande qu'à s'agrandir.

Le Processeur Expérimental

MCA_0

C.R. Subject classifications Informatics 4.21, 4.34, 6.22, 6.3

Résumé. Le processeur expérimental MCA-0 fait l'objet d'une double description, d'un point de vue informatique et d'un point de vue hardware. Le langage qui permet de l'utiliser a été choisi pour des critères pédagogiques d'enseignement et de formation à la recherche. Sa structure hardware elle-même est décrite à l'aide de moyens de l'informatique. Le but est d'obtenir un langage de programme qui permette une description aussi complète que possible de processeurs. La réalisation de l'objet décrit montre, lorsqu'il fonctionne, l'efficacité du langage. C'est le cas du prototype dont il est question ici.

PRELIMINAIRE

L'article qui suit est une introduction à la construction d'un langage de programmation spécifique : celui qui permet de décrire des processeurs. Mais l'objet de cet article est avant tout la description d'un processeur expérimental effectivement construit et qui marche.

Il s'agit donc d'une réalisation qui déborde largement de notre qualification officielle d'Informaticiens. Et c'est un assez gros travail d'équipe. Hélas, les moyens de notre petit laboratoire d'Informatique théorique étaient quelque peu insuffisants. Heureusement, la notion de pluridisciplinarité ne semble pas être un vain mot au sein de l'Université, au moins pour quelques Universitaires véritables, et nous avons pu bénéficier de collaborations sans lesquelles notre but aurait eu peu de chances d'être atteint. Je remercie à ce propos tout particulièrement Monsieur Chaîne, du Département de Physique de Luminy, et Messieurs Fumanal et Lazzarini, du Laboratoire des Mesures Physiques de St Jérôme, qui nous ont tous apporté leurs compétences et prêté leur matériel avec beaucoup de gentillesse et d'efficacité.

INTRODUCTION

Depuis de nombreuses années déjà, j'avais envie de regarder d'un peu près un processeur réel. J'enseignais des structures de processeurs fictifs décrits en termes de programmation. Et puis, la technique évoluant et se commercialisant, il s'est trouvé sur le marché des circuits qui pouvaient me permettre de satisfaire ma curiosité. La plus longue partie de mon travail a consisté, pour moi, à établir un lien entre mes structures informatiques que j'avais baptisées "unités centrales" et qui n'étaient autres que des compilateurs de déroulement, et cette circuiterie aux niveaux de contrôle standardisés.

Pour demeurer entièrement maître de mes structures, je me suis limité aux circuits les plus simples en bannissant ALU , séquenceurs et autres. Un simple additionneur, avec des buffers, latches et décodeurs ont fait amplement l'affaire.

Côté informatique j'ai également restreint mon ambition à une excellente machine d'enseignement : la machine à cases adressables qui ne possède pas, bien sûr, tous les gadgets d'un ordinateur moderne, mais dont il est prouvé toutefois qu'elle est aussi puissante ; et qui joint les avantages d'un langage rustique et simple à une structure fondamentale des ordinateurs : la mémoire centrale.

Le produit obtenu, le MCA-0 , est un prototype expérimental qui ne sera pas destiné au calcul mais à l'expérimentation sur ses structures, et dont l'intérêt est d'amorcer un deuxième projet plus important. En fait, nous nous sommes fait la main à son propos dans un domaine qui n'était pas le nôtre.

Je vais essayer dans les pages qui suivent de décrire les moyens que je me suis donné et le résultat obtenu, en tentant de ne pas sortir du champ informatique. Je ne traiterai des structures électroniques que ce que je pourrai mettre en forme à l'aide de sortes de langages de programmation.

Dans un premier chapitre, je définis un langage de câblage qui me sert à m'exprimer pour la description des structures électroniques. Comme on peut le constater, un tout petit nombre de types élémentaires de cellules suffit auxquelles on rajoute, bien sûr, les portes habituelles NAND, NOR et OUX.

C'est à l'aide de cet outil que je décris au chapitre suivant un circuit de calcul destiné à réaliser matériellement tous les calculs du processeur. Dans ce circuit, ne se produisent que des échanges de contenus avec éventuellement une transformation au passage qui est l'addition. Sur cette structure matérielle j'applique deux langages qui se correspondent point à point. L'un sert à décrire les échanges internes de cellule à cellule, c'est un véritable langage de programmation interne. L'autre décrit les commandes internes nécessaires pour que s'effectuent les échanges.

Le chapitre IV est, en fait, introduit par ce langage de commandes, car il est la description du circuit capable de générer les jeux de commandes nécessaires au fonctionnement du circuit de calcul.

On trouve au chapitre III le langage du processeur tel que l'utilisateur doit le manipuler, en fait, sous les deux formes habituelles : la première, symbolique, est le langage tout simple de la machine à cases adressables ; l'autre en est le code tel que l'on doit l'enregistrer en mémoire centrale pour que le processeur puisse calculer. Or, le choix de ce code a des implications sur la structure du circuit pilote. Et le chapitre III est également une introduction au chapitre IV.

Il reste à expliquer le mécanisme moteur qui régit les divers changements de phases, c'est le sujet du chapitre V. C'est là que se trouvent rassemblés les aspects "timing" ou "séquenciel" de l'organisation.

Là s'arrête la structure du processeur. Il serait important d'envisager la circuiterie de la mémoire centrale ainsi que les moyens d'échange avec l'extérieur qui lui sont normalement liés.

Cela fera l'objet d'un article ultérieur. Les mémoires utilisées pour constituer une mémoire centrale, pour être suffisamment économiques, sont en général plus lentes que les cellules du processeur. Pour essayer d'utiliser un peu moins mal celui-ci, on s'engage dans une organisation plus complexe. J'aborderai donc ce domaine en même temps que le langage descripteur de processeurs.

Pour l'instant, étudions simplement notre prototype.

I. LE LANGAGE DE CABLAGE

En termes informatiques je vais essayer de décrire une construction électronique. Pour cela, je définis un petit ensemble de sémantèmes que je prends commun aux deux champs, et qui va me permettre d'obtenir une double image d'un même objet : le processeur. L'un de ces champs est constitué par le langage de câblage, l'autre par le langage interne du processeur.

Canal et flux.

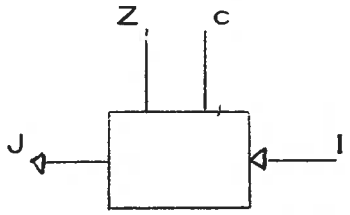
L'information circule sur un "canal", qui est l'image d'un fil, ou d'un ensemble de fils montés en parallèle. Le "flux" désigne une configuration telle qu'elle se présente, stable au bout d'un temps en général très court, sur un canal. Ce sont des potentiels établis sur des fils, qui servent à noter physiquement une quantité d'information. Il y a pratiquement un potentiel Haut et un potentiel Bas possible par fil. Je m'en tiens à cet usage technologique qui a fait ses preuves, et toujours conformément à l'usage, je fais correspondre le "0" et le "1" de l'informatique, respectivement avec les potentiels Bas et Hauts de l'électronique. Et, enfin, "moment" me sert à désigner la capacité de porter un élément d'information : le 0 ou le 1 . Ainsi un canal à 8 moments est un canal à 8 fils, et un tel canal peut porter 256 flux différents.

Un canal est orienté, et parfois dans les deux sens.

La Cellule-Mémoire.

Cette cellule est attaquée par 4 canaux, dont trois en entrée et 1 en sortie. Deux de ces canaux servent de commande et sont à 1 moment seulement. Le tableau qui définit la fonction de cette cellule n'est valable qu'au bout du temps nécessaire à la stabilisation des potentiels. Ce temps est en général court mais il est non nul et il faut en tenir compte dans l'organisation du processeur, comme on le verra au chapitre du "timing". De plus, toutes les cellules ne réagissent pas à la même vitesse.

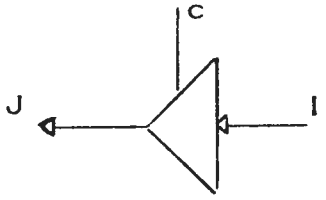
La commande "c" , selon le flux apporté sert à mémoriser le flux en entrée, ou à stabiliser en sortie "J" le flux mémorisé dans une phase antérieure, et quel que soit le flux actuel en entrée "I" . La commande "Z" sert à mémoriser la valeur zéro et à stabliliser le flux zéro en sortie.

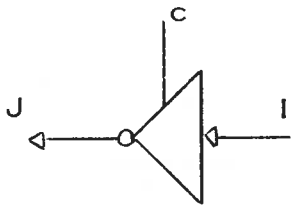


Les canaux I et J peuvent être à plusieurs moments pourvu qu'ils soient à un nombre identique de moments.

Le Buffer trois-états.

Le buffer trois états est une sorte de robinet qui sert à isoler le canal de sortie par rapport à celui de l'entrée. Ce circuit est indispensable quand plusieurs cellules attaquent en sortie un même canal. Selon la valeur de la commande appliquée en "c" , le flux en entrée se retrouve identique en sortie, voire amélioré, ou alors l'entrée se trouve isolée de la sortie. Une autre variante de ce circuit est la seconde présentée, qui, lorsqu'elle est passante, donne en sortie le flux inverse du flux en entrée.

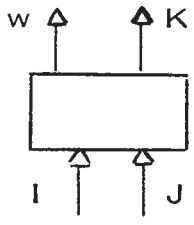




En fait, c'est seulement cette dernière cellule qui est indispensable. Technologiquement d'ailleurs, il se trouve que les buffers inverses sont plus performants.

L'Additionneur.

Je prends un additionneur simple, "l'adder", dont les trois canaux I, J et K sont identiques. Mais la sortie comporte un canal supplémentaire w qui véhicule le flux dû au dépassement de capacité. Ce canal est à un seul moment. Si je désigne par k le maximum de flux des canaux I, J, K, l'adder fonctionne ainsi :



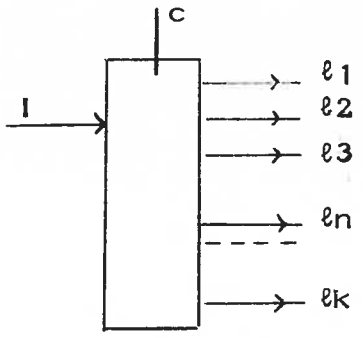
si $f(I) + f(J) > k$ alors $f(K) = f(I) + f(J) - k$ et $w = 1$

si $f(I) + f(J) \leq k$ alors $f(K) = f(I) + f(J)$ et $w = 0$

C'est le seul circuit arithmétique de ce processeur.

Le décodeur.

Ce circuit a pour rôle d'établir une relation entre un contenu de canal et un numéro de ligne. Par ligne j'entends canal de moment 1 . Si k est le maximum de flux que transporte I , et que :



$$0 \leq n , j \leq k - 1$$

pour $f(I) = n$, alors $f(In) = 0$
et si $n \neq j$, alors $f(I\,j) = 1$.

En d'autres termes, ce circuit excite une ligne de sortie dont le numéro d'ordre est égal à la valeur transportée par le canal I . Dans l'exemple ci-contre il y a k canaux en sortie.

Transfert de l'information.

Tous ces circuits prennent en charge un ou plusieurs flux en entrée, et maîtrisent un ou plusieurs flux en sortie. Les commandes interviennent pour modifier l'état de la cellule et le rapport qui existe entre flux d'entrée et flux de sortie. Je pourrais dire également que l'information est amenée à traverser le circuit. Une transition, réalisée par un système matériel se déroule dans un laps de temps non nul et non négligeable, bien que bref en général. Par exemple, pour simuler les cellules mémoires, j'ai utilisé des circuits TTL dont le temps de réaction est de l'ordre de 10 nanosecondes. Encore faudrait-il bien préciser ce que représente un tel intervalle de temps. Je peux dire en première approximation que, supposant les flux d'entrée stabilisés, l'intervalle commence au moment précis où le potentiel de la commande attaque la borne du circuit, et il s'achève au moment où on considère que le nouveau flux est stabilisé en sortie.

Or, les circuits sont un tant soit peu différents les uns des autres. Ce temps risque de n'être pas exactement égal pour tous, j'entends tous ceux d'un même type. Car d'un type de cellule à l'autre alors ils peuvent être franchement différents.

Pour une même série de circuits j'adopterai un Δt dont je sois sûr qu'il maximalise tous les intervalles, et je prendrai même encore une petite marge supplémentaire. En effet, nous verrons plus loin que respecter ces marges est une des conditions de sécurité du fonctionnement du processeur.

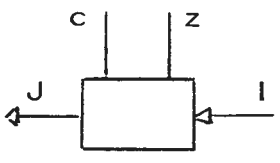
Flux d'information.

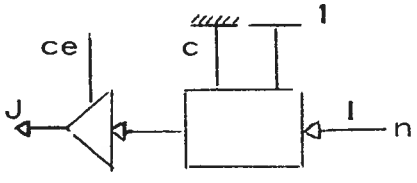
Physiquement on utilise, au moins dans les technologies qui m'ont servi, deux potentiels électriques différents pour représenter les deux flux différents possibles sur un même fil. Ces potentiels je les appellerai "zéro" et "un" ou bien 0 et 1 . Je néglige momentanément le fait que, selon la technologie utilisée et selon qu'il s'agit d'entrées ou de sorties, ces potentiels s'expriment de manière complexe.

Cellules composées.

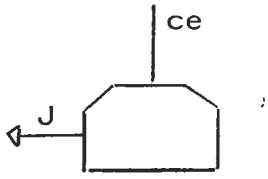
A partir de ces quelques cellules élémentaires, on peut construire des cellules plus complexes qui vont représenter les propriétés de circuits réels que j'ai utilisés. C'est en particulier le cas des "packs" de mémoires que je classe en deux catégories : les mémoires définitives (les ROM) et les mémoires momentanées (les RAM). Pour des raisons pratiques épaulées par des possibilités technologiques dans certains cas il est bien commode d'utiliser les mémoires, non par cellules séparées mais par paquets importants de ces cellules élémentaires.

Je reprends donc le schéma de la cellule-mémoire, avec un canal d'entrée I et canal de sortie J , à un ou plusieurs moments, mais identiques, et ses deux commandes c et z . J'ai là l'image d'une mémoire momentanée élémentaire. A partir de cette idée, j'imagine la construction qui utilise deux cellules élémentaires. Si $0 \leq n \leq k-1$ et que n soit imposée une fois pour toutes, par exemple en reliant les fils du canal à la masse ou au potentiel 1 , je relie c à la masse pour rendre la cellule transparente à n et Z au potentiel 1.





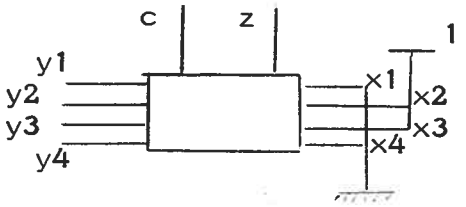
Et c'est alors la commande "ce" qui va permettre de stabiliser n sur le canal de sortie ou d'isoler la cellule de ce dernier.



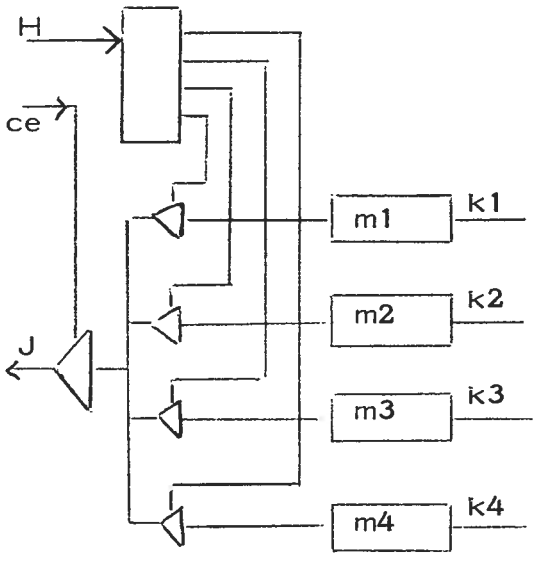
Je donne une nouvelle représentation pour ce genre de cellules, dont il faut connaître par ailleurs le contenu. Je vais alors construire des paquets de telles cellules.

Mémoires définitives.

J'imagine une cellule à flux à 4 moments, par exemple : chaque fil x1, x2, x3, x4 pouvant être porté soit à 0 soit à 1 , ainsi que les fils y1, y2, y3, y4 de sortie. Et je suppose que je veuille imposer la configuration 0110 à cette cellule pour qu'elle devienne définitive ; je présente une solution qui, si elle ne correspond pas exactement à ce qui se fait dans la réalité, a un but explicatif dans cet exposé. x1 et x4 sont reliés à la masse, potentiel zéro, et x2, x3 reliés au potentiel 1 . De la même manière la commande est reliée à la masse, et le Z au potentiel 1. Ainsi, dès que la cellule est mise sous tension, la configuration en entrée est imposée en sortie.

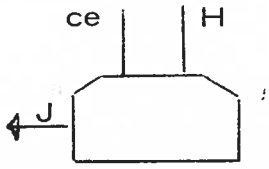


Je vais construire une cellule quadruple, et la technique est valable pour "intégrer" un nombre quelconque de cellules élémentaires.



Si k1, k2, k3, k4 sont des constantes fixées en entrée de chaque cellule, pour choisir l'une de ces valeurs afin de la stabiliser en sortie, sur J , il suffit d'envoyer sur H une valeur qui sera considérée comme l'adresse de la cellule choisie parmi les autres. Bien sûr, le canal H comporte ici deux fils. Là commence à s'introduire la notion d'adresse, en effet le décodeur transforme une valeur en une commande qui libère la sortie de la cellule choisie alors que les autres demeurent bloquées.

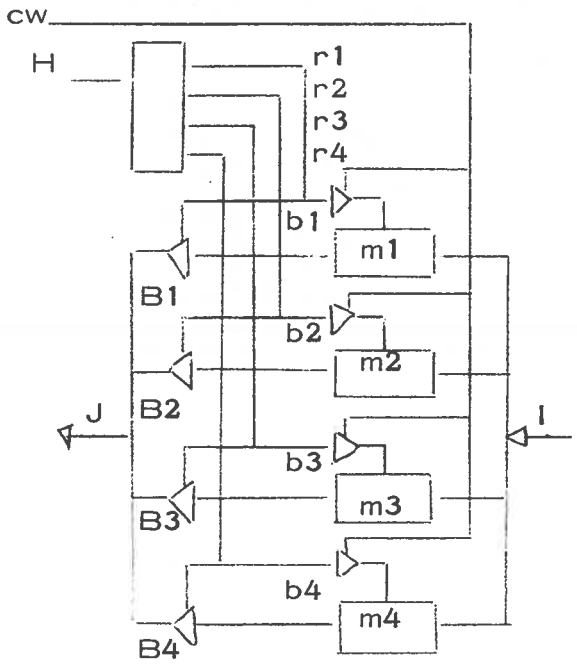
La commande "ce" sert à contrôler la sortie du résultat sur J quand l'information est stabilisée en sortie de la cellule sélectée.



Je condense ce schéma sous la forme ci-contre.

Mémoires momentanées.

Comme dans le cas précédant, je constitue un pack d'un certain nombre de cellules élémentaires, ici quatre, dans l'intention que, lorsque je m'adresse au pack, c'est une seule cellule que je veux atteindre, soit pour la lire soit pour lui imposer un nouveau contenu. Dans le schéma ci-contre, la cellule-décodeur a un canal d'entrée à 2 moments. Chacun des 4 fils de sortie r1, r2, r3, r4 est relié à la commande d'une cellule mémoire m1, m2, m3, m4.



Ainsi, si je stabilise sur H le flux 1, le décodeur sort la valeur 0 sur la ligne r2, ce qui a pour effet de sélecter en sortie m2, en rendant également passant le buffer B2. Alors J véhicule le contenu de m2, les autres cellules restant bloquées puisque les lignes r1, r3, r4 maintiennent un 1 sur les commandes des autres buffers et des autres mémoires, les bloquant par là-même.

f (H)	0	1	2	3
Mem.	m1	m2	m3	m4

Tableau T1

Dans le tableau ci-contre, qui met en correspondance d'une manière arbitraire les flux de H avec les cellules m1, on voit apparaître la notion d'adresse telle qu'elle va nous être utile.

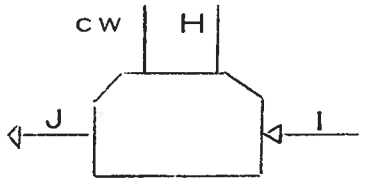
Je vais faire une hypothèse sur "cw" :

si f (cw) = 0 alors on écrit dans la cellule sélectée par f (H)
si f (cw) = 1 -- lit --- --- --- ---

En effet, 1 sur la commande "cw" bloque les buffers b1, b2, b3, b4 et les commandes des cellules sont maintenues à 1 par les "pull-up" R1, R2, R3, R4 . Dans ce cas, les cellules ne peuvent être qu'en émission et le flux de J est égal à ce qu'émet la cellule sélectée par H .

Si, au contraire, la ligne cw est au potentiel 0 , alors les b1, b2, b3, b4 sont débloqués, et seule la cellule attaquée par la ligne r1 qui est à zéro sera à la fois en émission et en réception. Les autres demeurent bloquées.

C'est ainsi qu'on trouve couramment des circuits de mémoires momentanées qui comprennent 1024 cellules à 1 moment, et d'autres à 256 cellules à 8 moments.



Je me donne une forme pour représenter de tels empilements de mémoires. "cw" est la commande de mise en lecture ou écriture. H est le canal adresse qui sert à choisir la cellule et I et J les flux d'information en entrée et sortie.

II. STRUCTURE DE CALCUL

On dispose à ce moment de ce qu'il nous faut pour décrire une structure de calcul. C'est un ensemble de circuits qui, convenablement commandés, vont permettre de réaliser chacune des phases exigées pour le calcul ; par le moyen d'échanges de mémoire à mémoire et de transformations qui se font au passage.
En fait de transformation l'addition nous suffira.

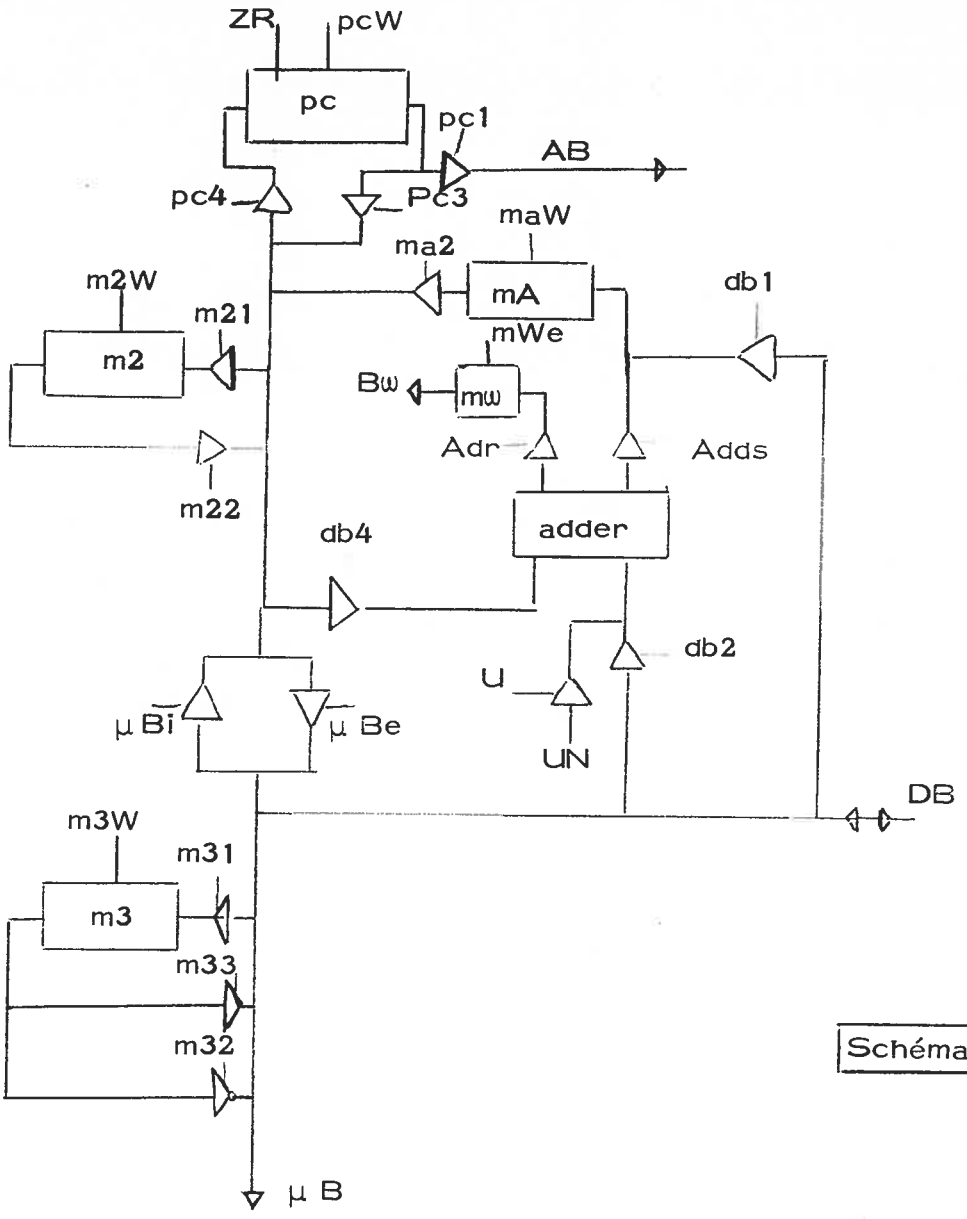


Schéma I

Deux mémoires sont spécialisées, ce sont Pc et mA , par contre m2 et m3 ne servent qu'aux manoeuvres d'échanges et stockages. Le canal AB sert à émettre une adresse vers un paquet de cellules-mémoires momentanées, tandis que le canal DB permet d'en extraire ou d'y envoyer un contenu de cellule élémentaire, par simple mise en communication avec l'un des 3 registres m2, m3 ou mA . Ceci se fait par un jeu de connexions que je vais détailler.

Je me place dans une situation précise : m2 contient une certaine information "x" et m3 contient "y" . Je veux en faire une addition dans mA et mw . En utilisant les diverses cellules, comme je l'ai défini au chapitre du langage de câblage, je vais créer des chemins de communication entre m2 et m3 et chacune des entrées de l'adder, et en même temps je mets mA en état de réception ; pour cela il suffit de porter toutes les commandes suivantes au potentiel 0 :

m33, m22, db4, db2, Adds, Adr, maw

Le chargement de PC à partir de mA se décrirait ainsi : mettre à 0

ma2, pc4, pcw

Bien entendu, chaque fois, toutes les autres commandes sont maintenues au potentiel 1 .

Il est possible de décrire de cette manière tous les échanges réalisables dans la structure donnée. De cette manière de faire je dirai : ce langage est commode pour construire les jeux de manipulations de commandes mais mal commode pour donner une idée de ce qui ressort de l'opération. A chacune de ces phrases je fais correspondre une autre phrase de signification identique, mais appartenant à un autre langage. Ainsi, dans l'ordre, les images de la première et de la seconde des deux phrases ci-dessus :

$$c\,(mA) := c\,(m2) +_k c\,(m3)$$

$$c\,(Pc) := c\,(mA)$$

Le $+_k$ s'explique par la nécessité de tenir compte du dépassement de capacité. Tous les canaux sont à 8 moments, à l'exception de celui qui alimente mw qui est à 1 moment. Ainsi $k = 256$ pour cet exemple, et le tableau ci-dessous résume la sémantique de l'addition modulo - k .

	c (mA)	c (mw)
$c\,(m2) + c\,(m3) \leq k - 1$	$c\,(m2) + c\,(m3)$	0
$c\,(m2) + c\,(m3) > k - 1$	$c\,(m2) + c\,(m3) - k$	1

	Pcw	Pc1	Pc3	Pc4	m2w	m21	m22	m3w	m31	m32	m33	maw	Adds	mA2	cM	ω w	db0	db1	db2	db3	db4	μBe	μBd	Un	Z	Mw	CμB	Cw	CcμB		
si cω = 0 alors ef2																												0			i1
c(mA) : = c(M)		0										0			0			0													i2
Aig ( )														0								0					0				i3
c(mA) : = c(Pc)+1			0									0	0								0			0							i4
c(Pc) : = c(mA)	0			0										0																	i5
c (m2) : = c(Pc)			0		0	0																									i6
c(mA) : = 0																									0						i7
c(M) : = c(mA)		0												0	0					0		0				0					i8
c(mA) : = c(m2) + 1							0					0	0								0			0							i9
c(m3) : = c(M)		0						0	0						0		0														i10
c(m2) : = c(M)		0			0	0									0		0					0	0								i11
c(Pc) : = c(m2)	0			0			0																								i12
c(m2) : = c(mA)					0	0								0																	i13
c(mA) : = c(m2)+$_k$c(m3)							0				0	0	0			0			0	0	0										i14
c(m3) : = c(Pc)			0					0	0													0									i15
c(mA) : = c(m3)+ 1											0	0	0								0	0	0	0							i16
c(mA) : = c(m2)+$_k\overline{c(m3)}$							0				0	0	0			0			0	0	0										i17
si c(mA)= 0 alors ef1														0								0							0		i18
c(mA) : = c(Pc) + c(m3)			0								0	0	0						0	0	0										i19
c(Pc) : = c(m3)	0			0							0											0	0								i20
c(m3) : = c(mA)								0	0					0								0									i21
c(mA) : = 1 +$\overline{c(m3)}$										0		0	0			0					0	0	0	0							i22

Tableau T2

La machinerie dont je dispose ainsi est suffisante pour que je puisse constituer le tableau T2 . Les opérandes complémentés s'obtiennent à partir du registre m3 en utilisant en sortie le buffer négatif de commande : m32 .

Dans la liste de ces instructions internes, on en remarque trois particulières qui expriment des conditions. Leur traitement est également particulier et est expliqué dans les chapitres qui suivent.

si c (mA) 0 alors ef1
si c (m w) 0 alors ef2
Aig (c (mA) : (e1, " : 0"), (e2, " : -"), (e3, " : "), (e4, "si) , (e5, " : wo"), (e6, " : w1"), (e7, "si wo"), (e8, "vers"), (e9, "comp 2), (e10, " - 1"), (e11, "IV"))

Les deux premières instructions se passent de commentaire, la troisième, aiguillage, signifie que lorsqu'on tombe sur tel code opération, et là il faut se référer au chapitre du langage LUP, alors on se renvoit à l'étiquette qui lui correspond dans le doublet. Par exemple, si sur l'instruction "Aig" on trouve un code : := 0 , on se renvoit à l'étiquette e1 .

Pour le détail du traitement en langage de câblage se reporter au chapitre du Circuit pilote.

Autre remarque, dans l'expression des instructions de ce langage interne, le symbole "M" de c (M) me sert à désigner la cellule du bloc de mémoire momentanée, qui est adressée par le c (Pc) via l'AB , et dont le contenu est véhiculé par le DB (Adress Bus et Data Bus).

III. LANGAGE DU PROCESSEUR MCA-0

Depuis que l'informatique existe, et avant même qu'elle n'existe sous ce nom, une forte opposition séparait les constructeurs de matériel des utilisateurs de ce matériel. Je pense, pour m'exprimer brièvement, qu'une grande partie de cette opposition est dûe au décalage qui existe forcément entre la mise au point, la commercialisation d'un nouveau modèle d'ordinateur et le rôdage de son utilisation qui exige beaucoup de temps. De telle sorte que la mise en chantier d'une nouvelle "génération" ne peut pas vraiment tenir compte de l'expérience de la génération précédente.

Dans les années 60 les Américains tentèrent de raccourcir le circuit de "contre-réaction" lors du gigantesque MAC project. Une bonne partie de propriétés d'ordre logiciel furent câblées portant notamment sur les calculs d'adresses. Cette expérience n'eût pas, semble-t-il, le succès espéré.

Le dialogue est-il possible entre informaticiens et facteurs d'ordinateurs (je préfère ce terme à celui de constructeur qui désigne plutôt le système commercial nidificateur) ?

Personnellement je ne le pense pas si le problème est posé en ces termes. Aucun dialogue n'a jamais résolu de lutte de clans.
Par contre, essayer de parcourir le chemin que l'autre a parcouru me semble bien plus fructueux. Je ne vois pas vraiment d'autre moyen de saisir la difficulté à laquelle l'autre se heurte.

C'est donc ce que j'ai voulu faire ici. Tentative largement facilitée par les commodités désormais offertes par la technique, je dois le reconnaitre.

Je suis parti de mon expérience d'informaticien et d'enseignant pour choisir une machine qui se place bien dans un cadre déterminé. J'ai choisi un chainon dans la suite qui rattache la machine de Turing aux ordinateurs réels. Je me suis arrêté sur le modèle où apparait la mémoire centrale, ce morceau de ruban qui présente l'avantage d'avoir une longueur fixée d'avance.

Car c'est enfin cette propriété qui permet d'atteindre chacune des cases indépendamment. L'étape suivante qui comporte l'apparition de l'index exige l'introduction de notions importantes pour assurer avec cet index une gestion de l'accès aux cases. Ce chainon suivant, la machine formelle, avec son langage, la procédure formelle, fera l'objet d'une réalisation ultérieure (cf. [2]).

J'ai donc choisi un langage simple avec déjà des instructions à trois opérandes, et des conditions portant sur des comparaisons de case à case.

J'attribue donc au MCA-0 un langage que je dirai d'utilisation du processeur, par opposition au langage interne et à celui de câblage. Le LUP, donc par plaisir d'utiliser un sigle, traite de contenus de cases de la mémoire centrale, bien sûr, et non plus de registres.

(1) ca : = 0
(2) ca : = cb $+_k$ cd
(3) ca : = cb
(4) Si ca = cb vers ei
(5) Si c ω = 0 vers ei
(6) vers ei
(7) ca : = cb
(8) c ω : = 0
(9) c ω : = 1
(10) ca : = cb $+_k$ 1
(11) Instruction Vide (IV)

Ces instructions ont la signification décrite en (ref. [2]).

Codage du LUP.

Je dispose d'un support physique qui me sert de mémoire, c'est le bloc de mémoires momentanées. Considéré du point de vue de l'utilisation du processeur, il ne s'agit de rien d'autre que de la mémoire centrale.

Et dans le cas qui nous occupe je dis que je dispose d'une suite de cases ou d'octets que je repère par leur numéro : 0, 1, 2, ..., k - 1 .(J'en dispose ici de 256).

Je vais donc faire une hypothèse sur le codage du programme, sur la manière de le noter en mémoire, qui a des conséquences directes sur l'organisation du processeur, comme on va le voir.

A. D'abord le programme commence toujours à l'adresse zéro.
(Ceci n'est pas fondamental, c'est une convention).

B. Chaque instruction est représentée dans une suite contigüe d'octets d'adresses croissantes :

B1) Dans l'octet d'adresse la plus faible se trouve le code-opération, qui est caractéristique du traitement à appliquer aux opérandes.

B2) Les adresses des opérandes et les étiquettes viennent à la suite.

B3) Les codes d'une suite d'instructions sont rangés en mémoire de manière contigüe et par adresses croissantes.

Le tableau T3 donne les codes pour chacune des instructions du LUP.

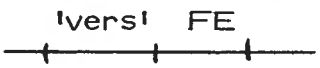
Instruction	1er octet	2ème octet	3ème octet	4ème octet
(1)	': = 0'	a		
(2)	': = +'	b	d	a
(3)	': ='	a	b	
(4)	'si ='	a	b	ei
(5)	'si c ω'	ei		
(6)	'vers'	ei		
(7)	': = -'	a	b	
(8)	'c ω0'			
(9)	'c ω1'			
(10)	': = + 1'	a	b	
(11)	'IV'			

Tableau T3

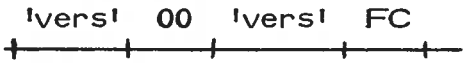
Les codes occupent donc 1, 2, 3 ou 4 octets selon les types d'instructions. Les quantités a et b désignent une adresse quelconque d'octet en mémoire. ei est une étiquette et se traduit ainsi en contenu d'octet :

c'est la distance qui sépare le premier octet qui suit le code de l'instruction qui contient l'étiquette, de l'octet qui contient le code opération de l'instruction étiquetée.

Exemple I : e1 : vers e1 se code :



Exemple II : e1 : vers e2 se code :
e2 : vers e1



Les valeurs sont écrites en hexadécimal, et les distances comptées en remontant sont complémentaires à 2.

Choix du code opération.

D'après ce que je viens d'en dire, il suffit que les codes des diverses instructions soient différents les uns des autres. Mais ils doivent jouer un rôle de commutation : ayant rencontré en mémoire un code-opération, il va falloir dérouler le programme interne qui lui correspond et doit, s'il y a lieu, traiter de l'information qui suit (par exemple prendre l'adresse de l'octet à mettre à zéro pour l'instruction (1)).

Plusieurs solutions se présentent, basées sur un calcul de reconnaissance du code. J'ai pris la plus simple pour le présent prototype : je choisis des codes qui sont directement les adresses des programmes de traitement. Ainsi, le code du ':=0' est l'adresse qui correspond à l'étiquette e1 dans l'instruction (i3) , le code du ':=+' est l'adresse qui correspond à e2 , etc...

Plus loin est expliqué comment le programme interne de traitement est enregistré dans la mémoire définitive P2 .

Ce programme interne contient en effet autant de parties qu'il y a d'étiquettes dans l'instruction d'aiguillage (i3) .
Ainsi, la structure du programme interne est la suivante :

```
        c (mA) : = c (M)
    Aig (c (mA) : (e1, ': = 0'), (e2, ': = +') ,..., (e11, 'IV'))
        e1 : c (mA) : = c (pc) + 1
             ..................
             ..................
        e2 : c (mA) : = c (pc) + 1
             ..................
             ..................
        e3 :
             ..................
             ..................

        e11 : c (mA) : = c (pc) + 1
              c (pc) : = c (mA)
```

Cf. Annexe Programme Interne

L'explication du fonctionnement de l'aiguillage est donnée au chapitre du circuit pilote. Le branchement du schéma I au schéma II se fait par l'intermédiaire du bus interne (μB) qui est relié à la circulation d'adresse sur la ligne $\mathscr{L}$ par l'intermédiaire d'un buffer C μ B .

Comme on l'a vu ci-dessus le code-opération se trouve dans mA avant de dérouler l'aiguillage. Réaliser ce travail consiste donc à émettre des zéros sur les lignes de commande suivantes :

ma, μ B c, C μ B

toutes les autres commandes doivent être à 1. (cf. le tableau T2).

A titre d'exemple, voici le programme interne qui traite de l'instruction (1) . Au moment où on atteint à ce programme, on vient de détecter le code-opération qui lui correspond. L'adresse en est encore dans Pc .

On va aller chercher en adresse suivante la quantité "a" , adresse de l'opérande.

```
e1 : c (mA) : = c (Pc) + 1
     c (Pc) : = c (mA)
     c (mA) : = c (M)
     c (m2) : = c (Pc)
     c (Pc) : = c (mA)
     c (mA) : = 0
     c (M)  : = c (mA)
     c (mA) : = c (m2) + 1
     c (Pc) : = c (mA)
```

Ce sont les trois premières instructions qui font cela. La 4ème met en réserve en m2 cette adresse pour la remplacer dans la 5ème par ce qui vient de la mémoire. Les 6ème et 7ème créent le zéro et l'expédient en mémoire, et les deux dernières préparent le contenu de Pc pour pouvoir atteindre au code de l'instruction LUP suivante.

On a là l'essentiel du principe, il faut alors se reporter à l'annexe Programme interne, pour disposer de la description complète du traitement interne.

IV. CIRCUIT PILOTE

Le côté statique du processeur étant décrit, on dispose de la suite des états stables, que j'ai présentés comme des sortes d'instructions internes. Le langage externe du processeur définit les macro-variations d'états, condensées chacune dans l'expression d'une instruction. Et on a vu enfin comment chacune de ces macro-variations pouvait être obtenue par une suite de mini-variations, décrites elles-mêmes chacune par une instruction interne.

Les états stables internes sont obtenus par des jeux de commandes qui créent des chemins entre des registres. Il faut concevoir un support capable d'émettre dans de bonnes conditions tous ces jeux de commandes. En d'autres termes, de réaliser toutes les instructions internes. Et ensuite il faut que ce support soit capable d'assurer l'enchainement de ces suites particulières de jeux de commandes ou d'instructions internes, qui réalisent les macro-variations ou instructions d'utilisation.

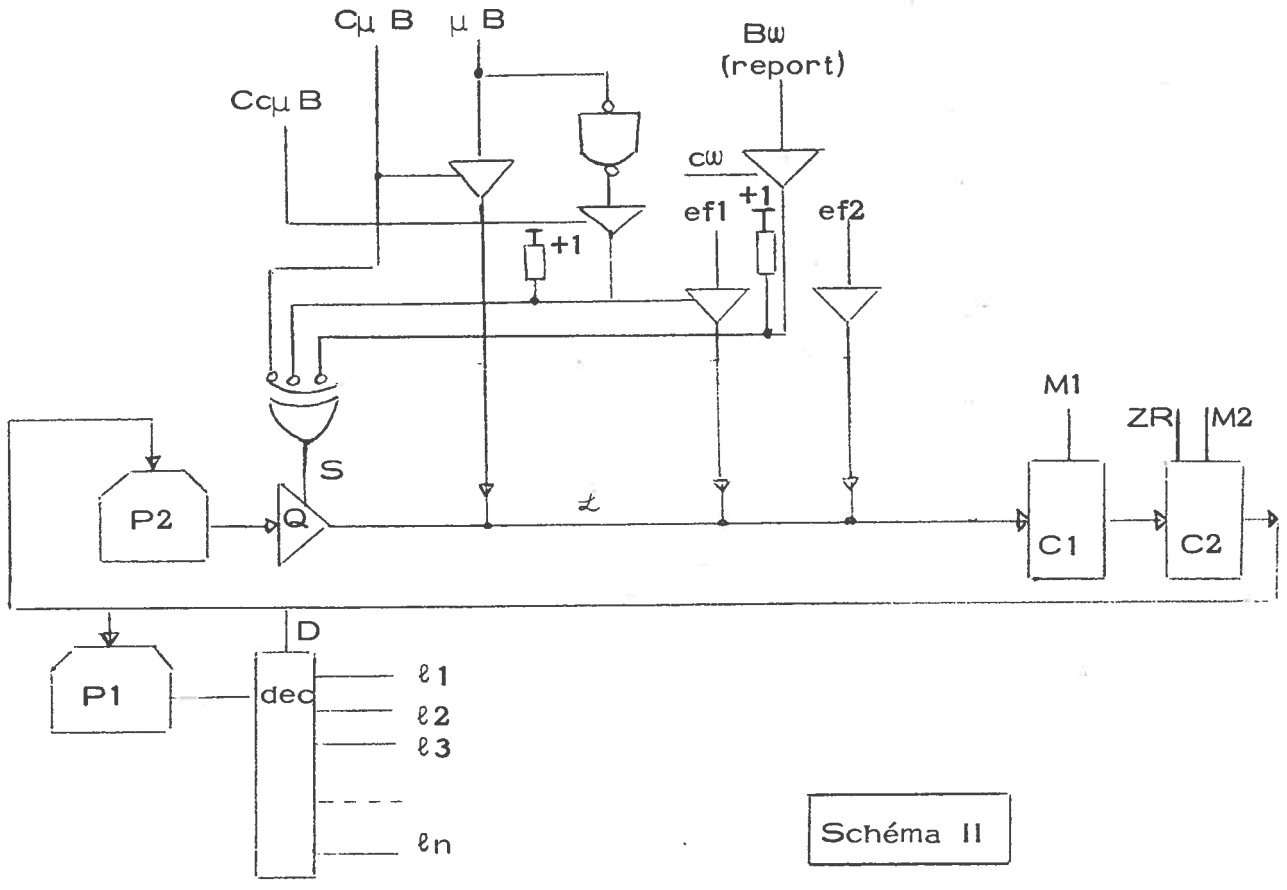


Schéma II

La mémoire définitive P2 contient le numéro de ligne suivante dans la suite des instructions internes contenues dans P1 . La cellule C2 contient l'adresse de l'instruction interne en cours, tandis que C1 va contenir, elle, l'adresse suivante, émise par P2 .

P1 , excitée par l'adresse émise de C2 , donne en sortie un code qui est transformé par le décodeur. Et c'est la ligne qui correspond à ce code qui attaque la circuiterie destinée à générer le jeu de commandes. Ce que je viens de décrire est une phase statique du processeur ; je vais essayer de montrer comment changer de phase.

Phase de calcul.

La commande M2 est au potentiel haut : C2 est bloquée en émission. Cet état doit durer assez pour que tout ce qui suit ait le temps de se réaliser. Ce qui est émis par C2 joue le rôle d'une adresse du programme interne disponible en P1 et P2 . Ces deux mémoires définitives excitées simultanément par cette adresse vont émettre, l'une, P1 , un code image de l'instruction interne (cf. Tableau T2), l'autre, P2 , l'adresse de l'instruction qui devra être prise en compte à la phase suivante.

Je suppose que les flux émis par P1 et P2 sont stables (ce qui finit par arriver au bout d'un certain temps). Il faut alors fixer dans C1 l'adresse suivante, et ceci se réalise en abaissant le potentiel de M1 pendant un laps suffisant pour que C1 ait le temps de mémoriser.

Changer de phase revient à abaisser le potentiel de M2 pour que ce qui est émis par C1 soit pris en charge par C2 .

Je vais maintenant envisager deux autres sortes de phases qui sont les traitements spécifiques de la commutation du programme interne.

Phase d'aiguillage.

Si je me réfère au paragraphe du choix du code opération, je constate que l'instruction d'aiguillage se déroulera toujours après l'instruction : c (mA) : = c (M)
donc on pénètre dans cette phase alors que le registre mA contient le code opération de l'instruction LUP qui est en train d'être prise en compte.

J'ai dit que cette valeur était choisie comme l'adresse du sous-programme interne qui traite de la LUP ainsi reconnue. Du point de vue qui vient d'être exprimé, il s'agit bien de l'adresse suivante.

En conséquence, il me faut connecter le registre mA en émission au canal $\mathscr{L}$. Bien entendu, afin que ce soit cette valeur qui soit prise en compte par C1 et non ce qui est émis par P2 , pour un mélange des deux, il faut alors bloquer l'émission de P2 : c'est le buffer Q qui est utilisé à cet usage. Le signal c μ B qui ouvre le passage du μ B vers $\mathscr{L}$ sert également à bloquer Q à travers le OU exclusif S .

Le jeu de commandes qui matérialise ce transfert est donné au tableau T2 à la ligne i3 . Le chemin que parcourt le flux déclenché par ce jeu est commun aux schémas I et II reliés par le canal μ B .

Phases conditionnelles.

Il y en a deux, ce sont celles du traitement des instructions internes i1 et i18 , tableau T2 .
Le principe est le même dans les deux cas.

Imaginons les schémas I et II reliés par le canal B ω . On voit que le canal est attaqué par un générateur d'adresse fixe : ef2 .
La commande C ω qui libère le flux qui correspond à ef2 vers $\mathscr{L}$ bloque en même temps Q par l'intermédiaire de S .

Le cas de l'instruction i18 est tout semblable. Bien sûr, la condition de choix de l'adresse suivante dépend de la valeur du contenu de mA . Le flux émis par mA est complémenté avant de pénétrer dans un circuit NAND , de sorte que la sortie de ce circuit sera au potentiel zéro, si et seulement si le contenu de mA est lui-même nul. Cette circonstance est validée par le signal cCμB . Le potentiel zéro, s'il est présent, débloque ef1 et bloque Q par le moyen de S .

Remarque électronique.

Le canal qui relie le buffer Cω au OUX S , comme on le constate, se trouve en période d'inactivité à un niveau flottant.

Or ce canal attaque en commande le buffer de ef2 . Le niveau flottant est incompatible avec un bon maintien d'un buffer. Il est donc nécessaire d'imposer un potentiel 1 sur ce canal. Ceci est réalisé par un pull-up, une résistance reliée au potentiel haut.

Générateur de commandes.

Le signal qui sort de P1 est transformé par le décodeur ; une seule d'entre les lignes qui émergent de celui-ci sera porteuse du potentiel zéro. Chacune des lignes du décodeur attaque un buffer par sa commande, ce buffer présente en entrée un canal dont le flux est imposé à zéro. Chacun de ces buffers sert à émettre un jeu de commandes nécessaires pour établir une phase de calcul. Les canaux de ces buffers comportent autant de moments qu'il est indispensable pour le fonctionnement d'une instruction interne, cf. tableau T2.

Ces choses-là étant précisées, on s'aperçoit que le générateur de commandes est simplement composé d'une série de buffers dont chacun est relié par la commande à une sortie du décodeur. Il y a évidemment autant de buffers que d'instructions internes.

Remarque.

Il aurait été également possible de réaliser un générateur de commandes plus condensé, en prenant une mémoire définitive qui comporte directement en sortie autant de lignes qu'il y a d'instructions internes à piloter. Les critères de choix entre ces deux solutions sont d'ordre purement technologique.

Pour pouvoir expérimenter le processeur, il me fallait une mémoire définitive rechargeable. Ceci n'existe à ma connaissance qu'en technologie MOS , il s'agit d' EPROM (erasable, programmable read only memory), à 8 moments en sortie. Il m'en aurait fallu 3 montées en parallèle, dans la deuxième solution. A titre de sécurité, il aurait été bon d'interposer des buffers TTL entre ces mémoires et les circuits TTL du processeur.

La balance est à établir entre, d'un côté, première solution : une carte importante qui contient tous les buffers TTL qui jouent le rôle des générateurs de commande ; de l'autre, deuxième solution : deux EPROM de plus avec, malgré tout, quelques buffers TTL et cette difficulté supplémentaire d'avoir à recharger 3 EPROM au lieu de 2 en cas de corrections du programme interne.

Bus de commande et sortie du générateur.

L'information qui sort du générateur de commandes se retrouve disponible sur le bus de commande. Les commandes du schéma I ainsi qu'une partie des commandes du schéma II sont connectées à ce bus. Comme on le verra plus loin au chapitre de la dynamique du processeur, une partie de ces commandes, celles qui verrouillent la mémorisation, sont délimitées dans le temps par l'une des horloges.

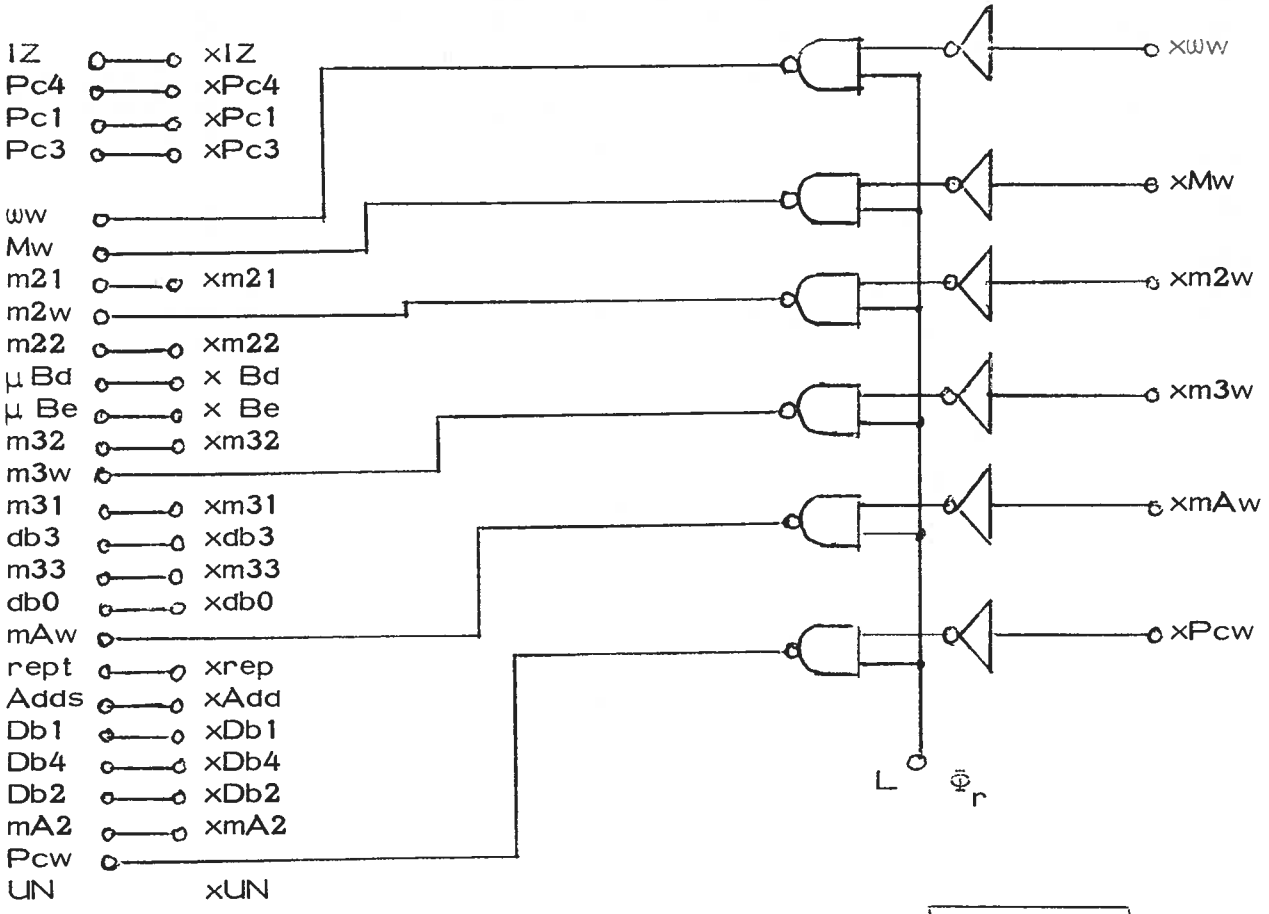


Schéma III

Les signaux désignés par un identificateur précédé d'un "x" sont immédiatement issus du générateur de commandes.

V. DYNAMIQUE DU PROCESSEUR

Cette partie est plus généralement désignée par le vocable "timing". Je pense que ce terme laisse de côté l'aspect impulsionnel, moteur, qui est fondamental en l'occurrence. Je vais essayer d'introduire la mécanique du franchissement des intervalles qui séparent deux phases. Ces intervalles sont des sortes de temps moteurs qui font avancer d'un pas le travail du processeur. C'est quelque chose qui est entièrement différent de ce que nous venons de voir. On appelle également cela séquencement, le moteur est alors une "horloge".

En bref, on se sert de signaux carrés, ce sont des signaux constitués par des potentiels alternativement maintenus hauts et bas, pendant des durées bien déterminées. Il existe des circuits spécialisés dans l'émission de ce genre de signaux, qui peuvent être directement appliqués sur des commandes de cellules telles que décrites plus haut, ou bien servir de déclencheur à un signal en attente sur une entrée de porte en apparaissant sur l'autre entrée.

Je n'insiste pas, pour l'instant, sur ce point de vue, mais je vais plutôt prendre en considération le calcul de la durée de maintien des signaux. Car, si ce sont les basculements d'un potentiel à l'autre qui servent à la commande des changements de phase, ce sont les durées de maintien qui assurent la complétion du déroulement d'une phase.

Je désigne par "descriptif du processeur" cette sorte de programme constitué par la réunion des schémas I et II . Associé à ce descriptif, je vais construire l'ensemble des phases de stabilisation, suffisant pour assurer les échanges prévus.

Phase générale d'émission.

Je définis une phase de durée Δ_{ge} , c'est la durée en potentiel haut appliqué sur M2 pour que C2 émette son signal, l'adresse dans P1 et P2. La commande appliquée à D doit être de durée égale mais de potentiel opposé, pour assurer la transmission du code instruction, cf. la cellule décodeur.

Phase de mémorisation.

Ce sont les cellules mémoires Pc, mA, m1 et m2 qui sont concernées. Il faut leur ajouter la cellule du report : w et la mémoire centrale externe au processeur : M .

Pour des raisons évidentes de validité de valeur mémorisée, la mémoire réceptrice doit être verrouillée avant que le canal d'entrée ne soit lui-même devenu flottant par blocage de l'émission. La phase de durée Δ_m représente le temps suffisant pour que les supports matériels de ces 6 mémoires puissent enregistrer, en toute sécurité, ce qui leur est présenté en entrée.

Phase de changement de phase.

Pendant la phase générale d'émission s'effectue également le calcul de l'adresse de l'instruction suivante, et là plusieurs circonstances se présentent :

a) La phase qui s'écoule est un calcul d'instruction de traitement, l'adresse suivante est celle qui est émise par P2 . Le buffer Q est transparent.

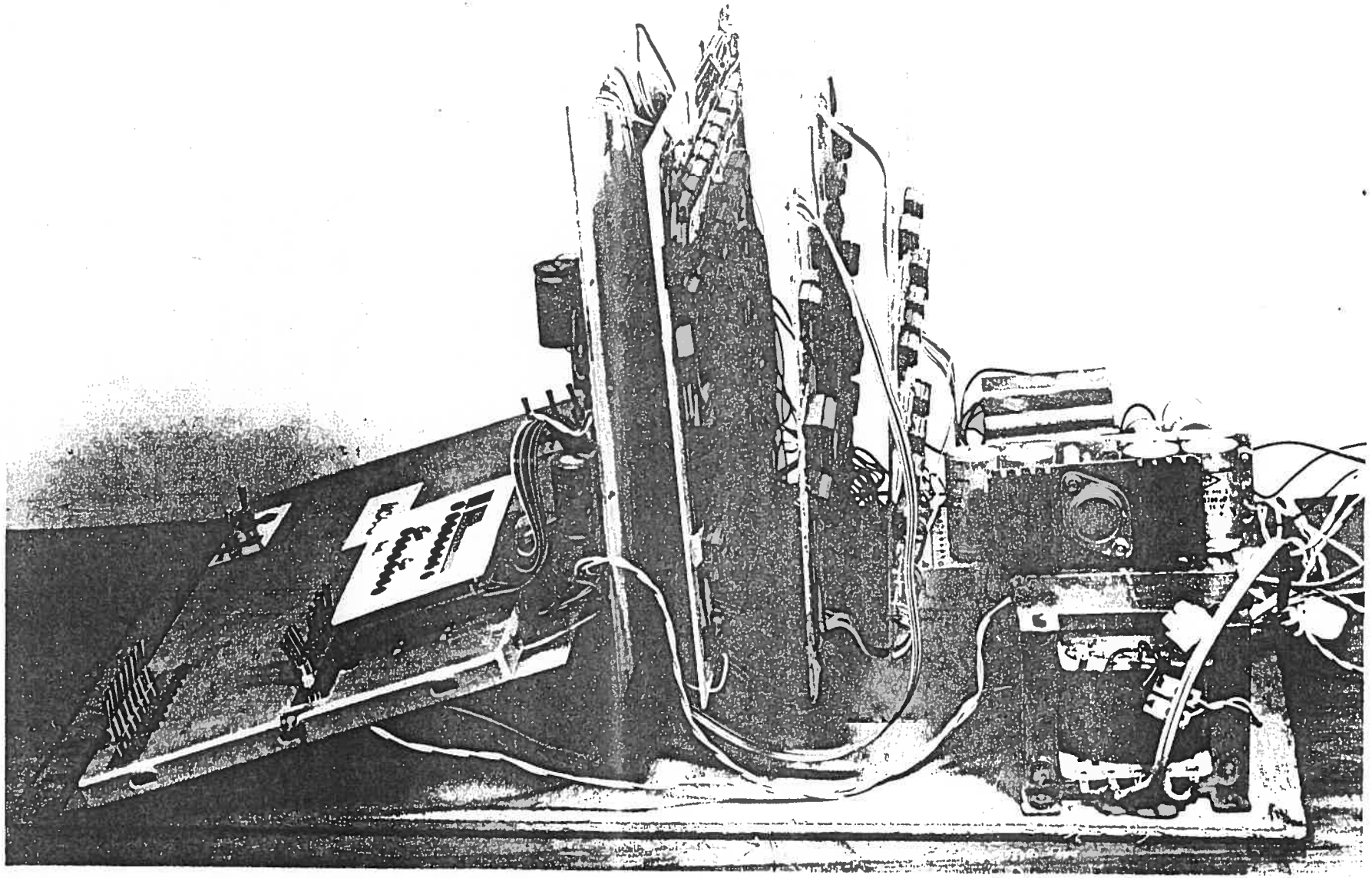
b) La phase en cours est un calcul d'instruction de commutation, et là 3 cas sont possibles :

- b1 : l'instruction est un aiguillage, Q est bloqué, c'est le contenu de mA qui est stabilisé en entrée de C1 .
- b2 : la commutation se fait sur comparaison de c (mA) à zéro, s'il y a égalité, alors Q est bloqué et une certaine adresse ef1 est stabilisée en entrée de C1 . Sinon Q est transparent et l'adresse est celle émise par P2 .
- b3 : sur la présence du report, Q est bloqué et c'est l'adresse ef2 qui est stabilisée sur l'entrée de C1 . Sur son absence l'adresse est celle émise par P2 .

C'est pendant la durée Δ_{ph} qu'est prise en compte l'adresse suivante par C1 . Et Δ_{ph} est la durée pendant laquelle le signal M1 reste bas.

Le changement de phase devient effectif pendant que le signal de M2 reste bas, Δ_c est cette durée. Le cycle de base du processeur est, en conséquence, de durée :

$$\Delta_{ge} + \Delta_c$$



Cadrage des phases.

Les diverses phases étant définies, il reste à les caler les unes par rapport aux autres. Je calcule les durées englobantes en fonction des durées englobées qui doivent être minimales par rapport aux propriétés des circuits utilisés (ici de la TTL sauf les EPROM, du MOS).

Le graphique qui suit donne pour les trois signaux dont j'ai besoin les positions relatives des changements de potentiels. Le démarrage doit se faire avec les trois signaux au potentiel 1 . C'est le premier coup actif de l'oscillateur pilote qui déclenche le jeu des basculements de potentiels.

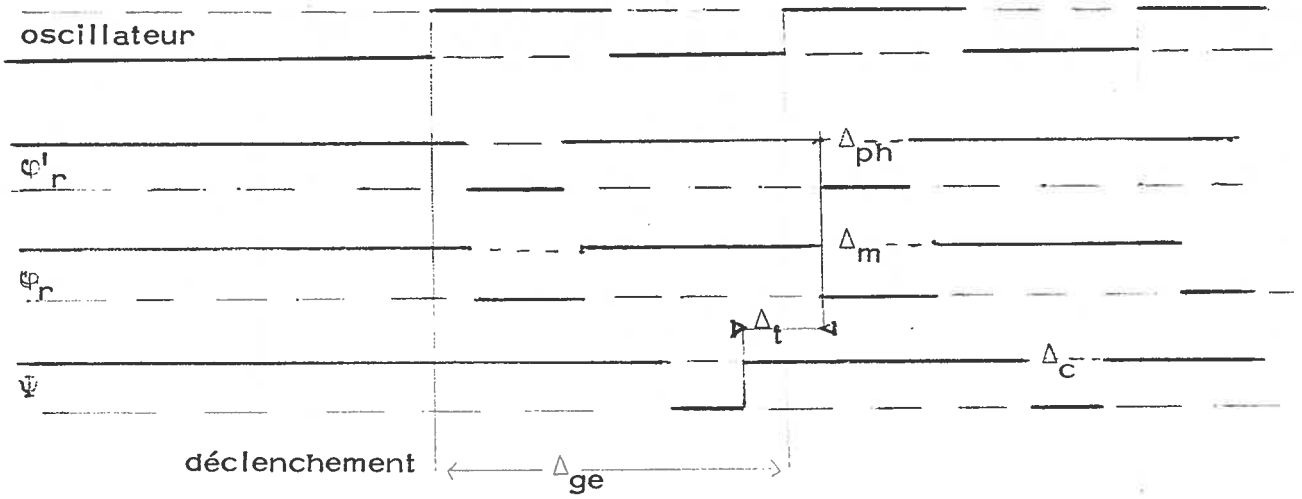


Diagramme 1

La quantité Δ_t doit demeurer suffisante pour être sûr qu'à aucun moment, au cours des dérives possibles, elle ne puisse s'annuler. Car, alors, il lui suffirait de devenir légèrement négative pour bouleverser les travaux d'échanges.

Points de phase.

Je dispose ainsi de trois signaux synchronisés que je dirai émis par trois horloges Ψ , φ_r , φ'_r ; je dispose également des mêmes signaux complémentés et $\overline{\Psi}$, $\overline{\varphi_r}$, $\overline{\varphi'_r}$ leurs horloges.

Dans ce processeur, et à titre expérimental, je n'ai pas séparé la structure propre du processeur de celle de la mémoire centrale. Outre L le point d'entrée de l'horloge qui commande la lecture (cf. schéma III), il y a T le point d'attaque de l'horloge qui commande la mémoire centrale. Il y a ainsi 6 couples horloge-point d'attaque :

P2 , $\overline{\Psi}$
D , $\overline{\Psi}$
M1 , φ'_r
M2 , Ψ
L , $\overline{\varphi_r}$
T , φ_r

Je définis également un état de blocage, phase de démarrage, qui peut se maintenir indéfiniment. Ceci est utile pour l'accès direct de l'extérieur au contenu de la mémoire centrale. Pour que l'état en cours des mémoires se maintienne, il est nécessaire que $\Psi = 1$, également $\varphi_r = 1$, $\varphi'_r = 1$ mais ces deux dernières conditions ne sont pas, elles, nécessaires.

à suivre

Bibliographie

Louis NOLIN — Formalisation des notions de machine et de programme. Gauthier-Villard.

Edmond BIANCO — Informatique fondamentale. ISR 70 - Birkhaüser Verlag.