Tomasulo et Hyperthreading

_____

Luminy - Mai 2003

_____

Denis Cazor

Mots clés : common data bus, hazard detection, multiport memory, register renaming mechanism, reservation stations, scheduler buffer, Tomasulo.

I) Utilité et objectifs

Exemple de structure

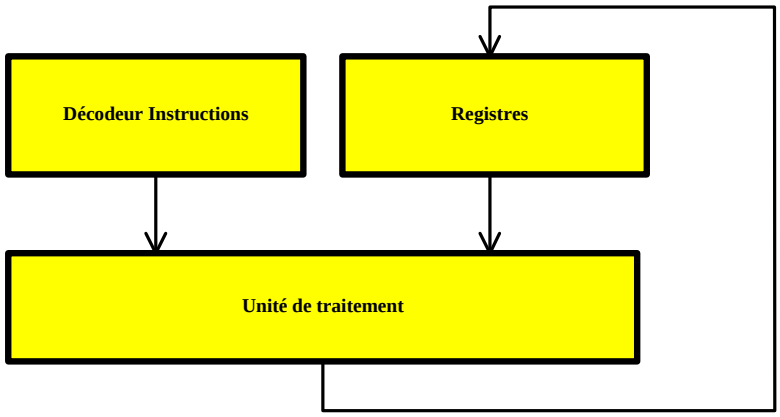
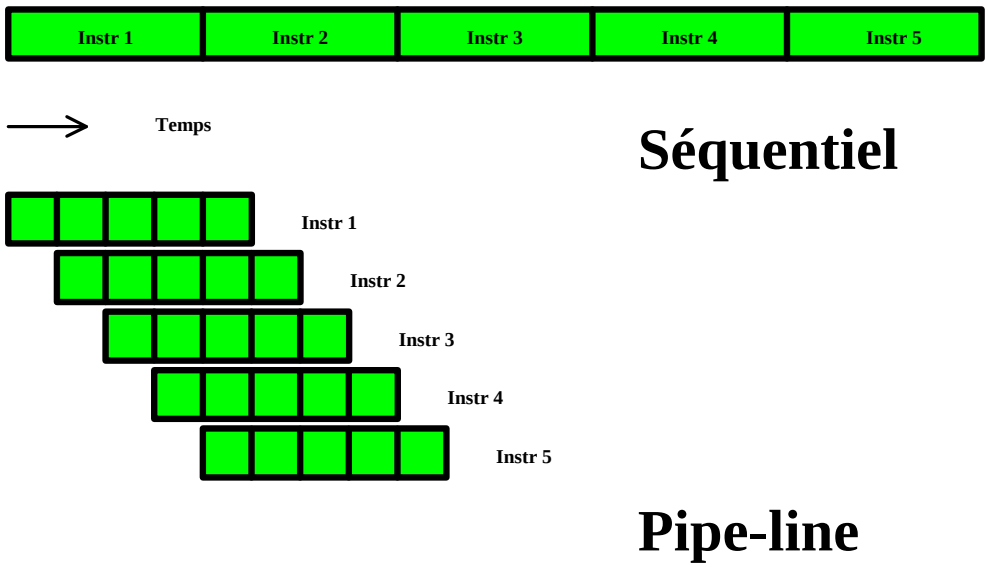


Schéma fonctionnel

Une solution très simple consisterait à calibrer toute opération, sur l'opération la plus lente. Cette solution est très peu performante. En fait, on introduit un pipe-line (c'est à dire la décomposition des opérations les plus longues en plusieurs cycles élémentaires) puis différentes unités de traitement indépendantes.

L'idée du fonctionnement séquentiel et d'un pipe-line



Comparaison du fonctionnement Séquentiel / Pipe-line

Difficultés avec un pipe-line et différentes unités de traitement : la latence des opérations n'est pas la même avec chacune des unités de traitement (entiers, flottants, adressage, registres ...). Souvent la latence d'une instruction dépend des données présentes dans les registres, ou bien des données qui se trouvent dans le cache. Ceci rend les prévisions très difficiles.

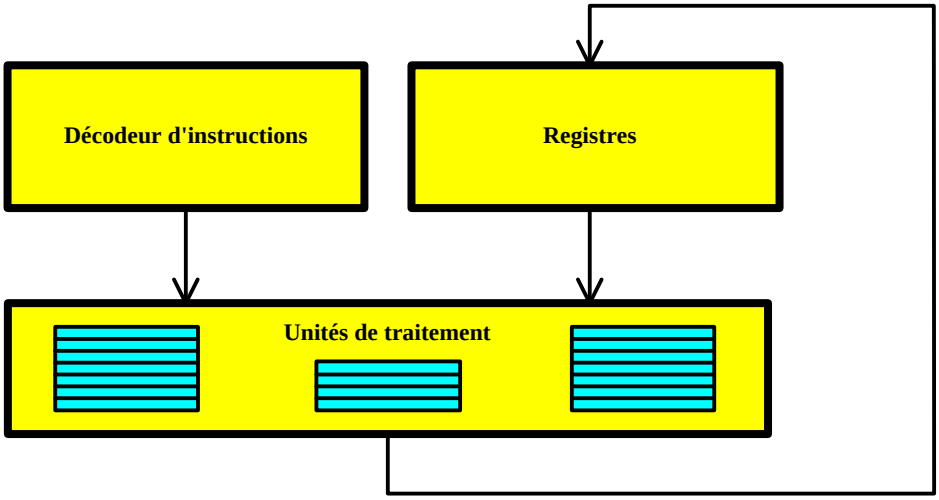


Schéma fonctionnel avec différentes unités de traitement

Exemple : produit scalaire

```
s := 0 ;
pour i = 1 à N faire s := s + a[i] * b[i] ;
```

1	LD R_0, 0	1 cycle
2	LD R_1, 1	1 cycle
3	LD R_5, N	1 cycle
4	LD R_2, $m_{ai}(R_1)$	30, puis 2 cycles
5	LD R_3, $m_{bi}(R_1)$	30, puis 2 cycles
6	LD R_4, $R_2 \times R_3$	5 cycles
7	LD R_0, $R_0 + R_4$	2 cycles
8	LD R_1, $R_1 + 1$	2 cycles
9	LD R_5, $R_5 - 1$	2 cycles
10	JNZ R_5, 4	3 cycles

Exemples de durées d'exécutions :

Integer pipe-line : 10 cycles AMD Athlon, 12 cycles AMD Opteron,
Floating point pipe-line : 15 cycles AMD Athlon, 17 cycles AMD Opteron, 20 cycles Intel Pentium 4.

Plusieurs types de difficultés sont liées aux durées d'exécution différentes des instructions, si on ne veux pas attendre la fin d'exécution de chacune d'entre elles.

Soit I et J deux instruction consécutives :

RAW (read after write), J accède au contenu d'un registre après qu'il a été modifié par I.
WAR (write after read), J modifie le contenu d'un registre après qu'il a été lu par I.
WAW (write after write), J modifie le contenu d'un registre après qu'il a été modifié par I.

Différents mécanismes sont mis en œuvre pour contrecarrer ces difficultés et améliorer la vitesse de traitement du processeur :

Contrôle statique effectué à la compilation, dépendant de la machine, code très développé avec de nombreux NOPs, compliqué à mettre en œuvre (static scheduling).

Contrôle dynamique effectué à l'exécution, code compact, indépendant de la machine, facile à mettre en œuvre au niveau du compilateur, (dynamic scheduling).

Avantages du mécanisme de contrôle dynamique :

Il permet d'optimiser l'exécution même si les dépendances ne sont pas connues à la compilation (par exemple dans le cas des accès mémoire dans un système bi-processeur).

Il simplifie l'écriture du compilateur, le code compilé peut être utilisé sur différents pipe-lines, sans modification.

Il permet le début et la fin d'exécution des instructions dans le désordre, ce qui permet un gain de performance substantiel.

Les contrôles et tampons sont distribués au niveau des unités de traitement.

Les moyens pour réaliser le contrôle dynamique, sur une proposition de Robert Tomasulo en 1963, sont appliqués dans la réalisation de l'ordinateur IBM 360/91 en 1967.

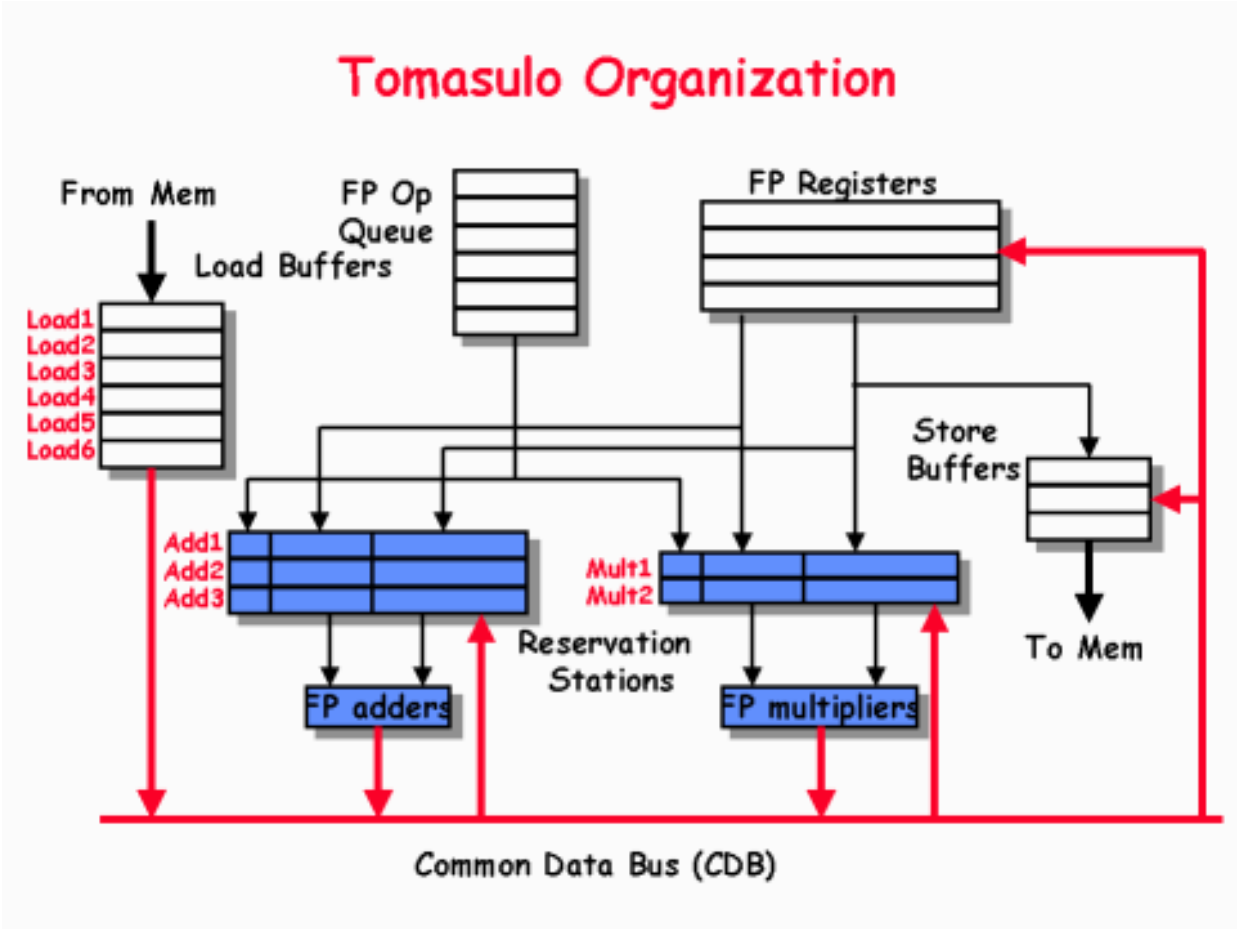
Toutes les instructions traversent l'unité de décodage dans l'ordre prédéfini par le compilateur.

Les instructions peuvent commencer et se terminer dans le désordre.

On distingue le début et la fin d'exécution d'une instruction. Entre ces deux instants l'instruction est en cours de traitement.

On utilise :

Les registres à assignation unique, le dépassement d'instruction.



Floating point unit - IBM 360/91 d'après Randy H. Katz

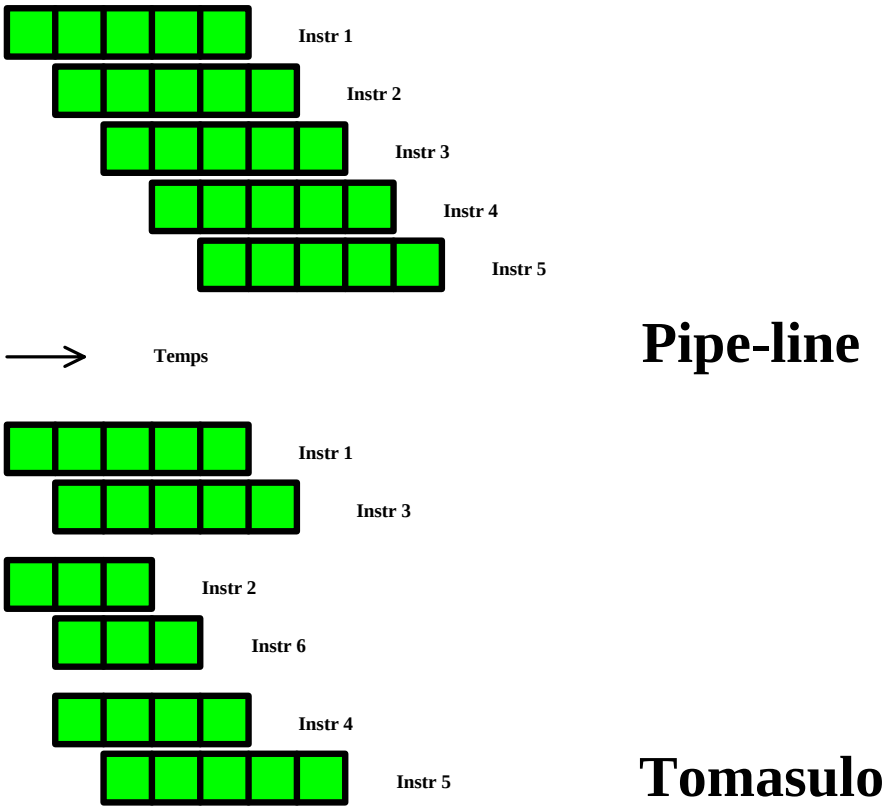
Les contrôles et les tampons sont distribués dans toutes les unités de traitement du processeur.

Les données sont accompagnées d'une étiquette et placées dans des files d'attente (reservation stations - unités de dépassement). L'utilisation de noms de registres "virtuels" (register renaming - assignation unique) permet d'éviter les conflits de type WAR et WAW.

Le nombre de registres physiques est supérieur au nombre de registres logiques. Ainsi le hardware peut réaliser des optimisations que ne peut envisager le compilareur.

Dans la réalisation de la société IBM, tous les registres (FP registers et reservation stations) ont des étiquettes fixées par le câblage. On envisage dans la suite de l'exposé, une configuration un peu différente, dans laquelle les étiquettes qui apparaissent au niveau des unités de dépassement dépendent de l'assignation des registres.

L'idée du fonctionnement d'un pipe-line et du "Tomasulo" :



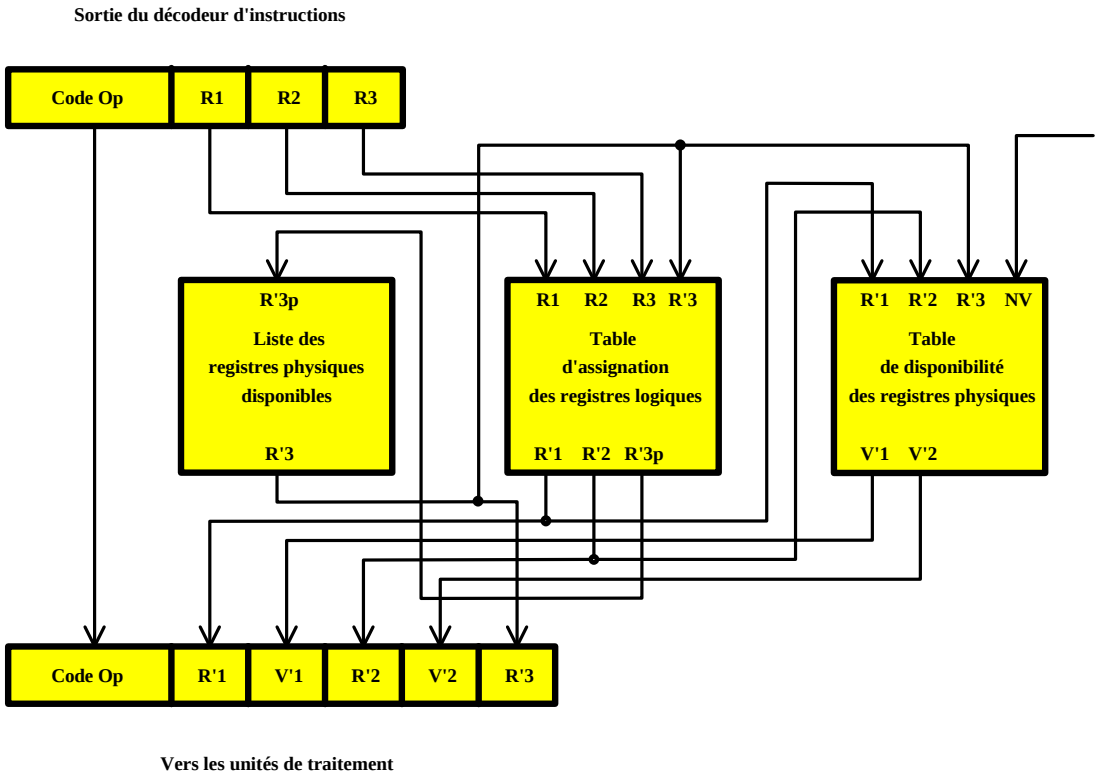
Comparaison du fonctionnement Pipe-line / Tomasulo

II) Registres à assignation unique (register renaming)

Permet d'éviter les contraintes de type WAR, et WAW. Le nombre de registres physique est supérieur au nombre de registres logiques, ce qui permet au hardware d'effectuer des optimisations que ne peut réaliser le compilateur.

Les registres logiques sont numérotés à la volée, au passage des instructions dans l'unité de décodage. On attribut un numéro physique à chaque registre logique. Un registre physique n'est écrit qu'une seule fois, lorsque sa valeur est déterminée. L'assignation du numéro physique persiste tant que les instructions décodées ne modifient pas la valeur du registre logique associé. Le registre physique est libéré lorsqu'une nouvelle instruction modifie le contenu du registre logique associé, si aucune instruction en attente n'y fait référence.

Par exemple : R1, R2 et R3 : **5 bits** ; NV, R'1, R'2, R'3 et R'3p : **10 bits** ; V'1 et V'2 : **1 bit**.



Unité de gestion des assignations

a) *Liste des registres physiques disponibles* : on utilise une mémoire FIFO, contenant les numéros des registres physiques non assignés. Cette mémoire est réalisée à l'aide par exemple de 1K×10 bits pour 1024 registres physiques ou 64×6 bits pour 64 registres physiques. Elle est initialisée au moment de l'opération de démarrage par des valeurs consécutives. Chaque écriture de registre logique R3 s'effectue en extrayant un numéro (R'3 en tête de FIFO) de registre physique, et en recyclant son ancienne affectation (R'3p en queue de FIFO).

b) *Table d'assignation des registres logiques* : elle contient les adresses physiques associées aux adresses logiques des registres. Avec 32 registres logiques et 1024 registres physiques, on réalise une mémoire de 32×10 bits par exemple. À chacune des opérations de transcription d'une instruction (de 1 à 6 instructions décodées à chaque cycle suivant les processeurs) on accède ***en lecture*** aux adresses de R1 et R2 (réassignations R'1 et R'2), R3 (recyclage de l'assignation précédente R'3p), et ***en écriture*** à l'adresse R3 (nouvelle assignation R'3).

c) *Table de disponibilité des registres physiques* : elle permet de savoir, au moment du transcodage des adresses registres si le contenu du registre est déjà affecté et donc valide, ou bien si la donnée correspond à une instruction dont l'exécution n'est pas encore terminée. Dans le premier cas on accédera à la donnée en lisant le contenu du registre, dans le deuxième cas la donnée sera présente sur le bus de donnée partagée (CDB common data bus) à un instant ultérieur. Avec 1024 registres physiques par exemple, on utilise une mémoire 1K×1 bits. À chacune des opérations de transcription d'une instruction, on accède ***en lecture*** aux adresses de R'1 et R'2 (bits de validations V'1 et V'2), et ***en écriture*** à l'adresse R'3 (invalidation de la

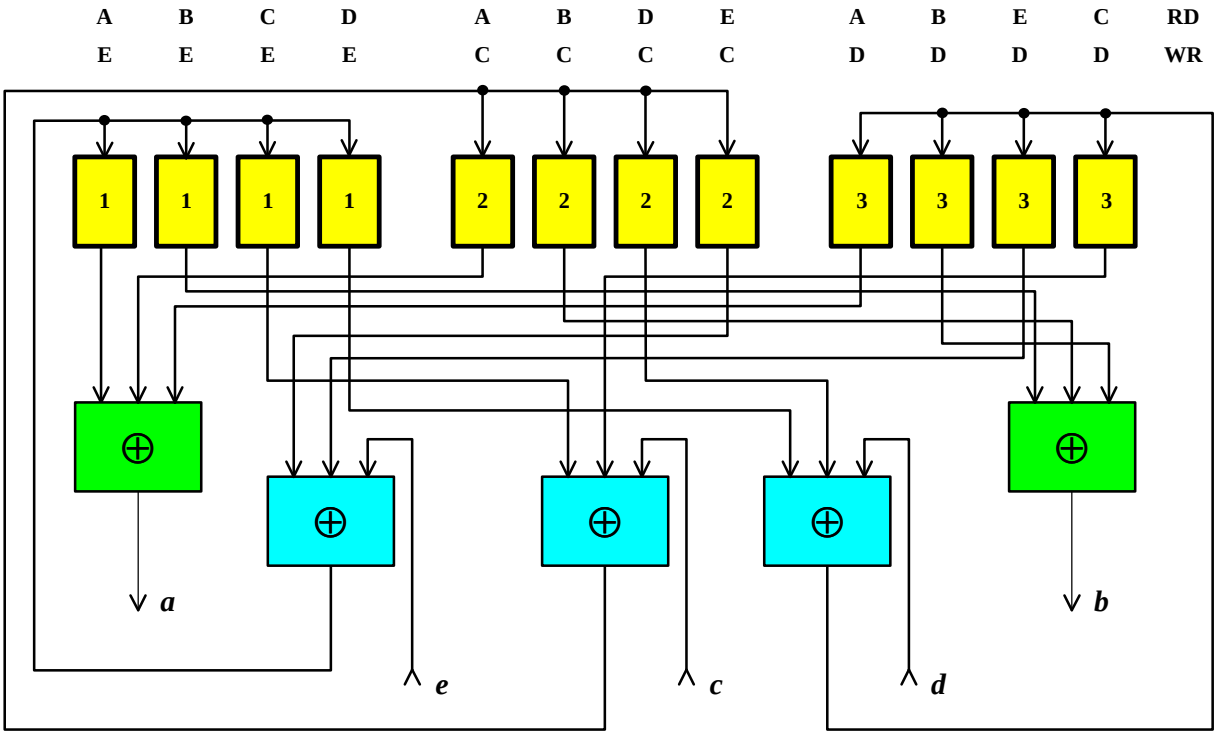
nouvelle assignation R'3). Il est nécessaire d'ajouter un accès en écriture pour la donnée disponible sur le bus partagé.

III) Lectures et écritures multiples

Comme on vient de le constater au paragraphe précédent, il est nécessaire de disposer (pour la réalisation des tables d'assignation et de disponibilité) de mémoires à accès multiples (multiport memory), si l'on désire obtenir un débit d'information important.

Ceci peut aussi être utile au niveau des registres physiques, auxquels on doit accéder rapidement en lecture (plusieurs instructions décodées simultanément), comme en écriture (plusieurs unités de traitement produisant un résultat dans le même cycle).

Soit à effectuer la lecture de données *a* et *b* aux adresses A et B, et l'écriture de données *c*, *d* et *e* aux adresses C, D et E (distinctes). On réalise un demi cycle de lecture (RD) suivi d'un demi cycle d'écriture (WR).



Mémoire à accès multiples

Les mémoires du banc numéroté 1 contiennent toutes la même information, de même pour les mémoires des bancs 2 et 3. Ceci permet d'effectuer plusieurs lectures de la même information simultanément à des adresses identiques ou distinctes. On obtient l'information globale en calculant le ou exclusif (ou la parité - XOR) des informations lues dans les mémoires 1, 2 et 3. On obtient ainsi la possibilité de modifier l'information dans toute la mémoire en n'accédant qu'à l'un des bancs.

1ère phase : *lecture*

$$a = a1 \oplus a2 \oplus a3, \quad b = b1 \oplus b2 \oplus b3,$$
$$c' = c1 \oplus c3 \oplus c, \quad d' = d1 \oplus d2 \oplus d, \quad e' = e2 \oplus e3 \oplus e.$$

2ème phase : *écriture*

$$c' = c1 \oplus c3 \oplus c \rightarrow c2, \quad d' = d1 \oplus d2 \oplus d \rightarrow d3, \quad e' = e2 \oplus e3 \oplus e \rightarrow e1.$$

3ème phase : *lecture*

$$c = c1 \oplus c2 \oplus c3 = c1 \oplus c' \oplus c3 = c1 \oplus c1 \oplus c3 \oplus c \oplus c3 = c.$$

Cette configuration permet de créer une mémoire multiport dans laquelle il est possible de réaliser n opérations de lecture et p opérations d'écriture à des adresses distinctes (configuration qui est favorisé par l'assignation unique). Le nombre de bancs mémoire doit être égal à p pour pouvoir réaliser les p écritures. Chaque banc doit comporter $n + p - 1$ mémoires identiques pour pouvoir réaliser les n lectures et préparer les p écritures.

Ainsi il est nécessaire d'utiliser $p \times (n + p - 1)$ mémoires, pour réaliser "simultanément" les n opérations de lecture et les p opérations d'écriture en 1 cycle de lecture/écriture.

RD\WR	1	2	3	4	5
1	1	4	9	16	25
2	2	6	12	20	30
3	3	8	15	24	35
4	4	10	18	28	40
5	5	12	21	32	45

Nombre de circuits mémoires nécessaires

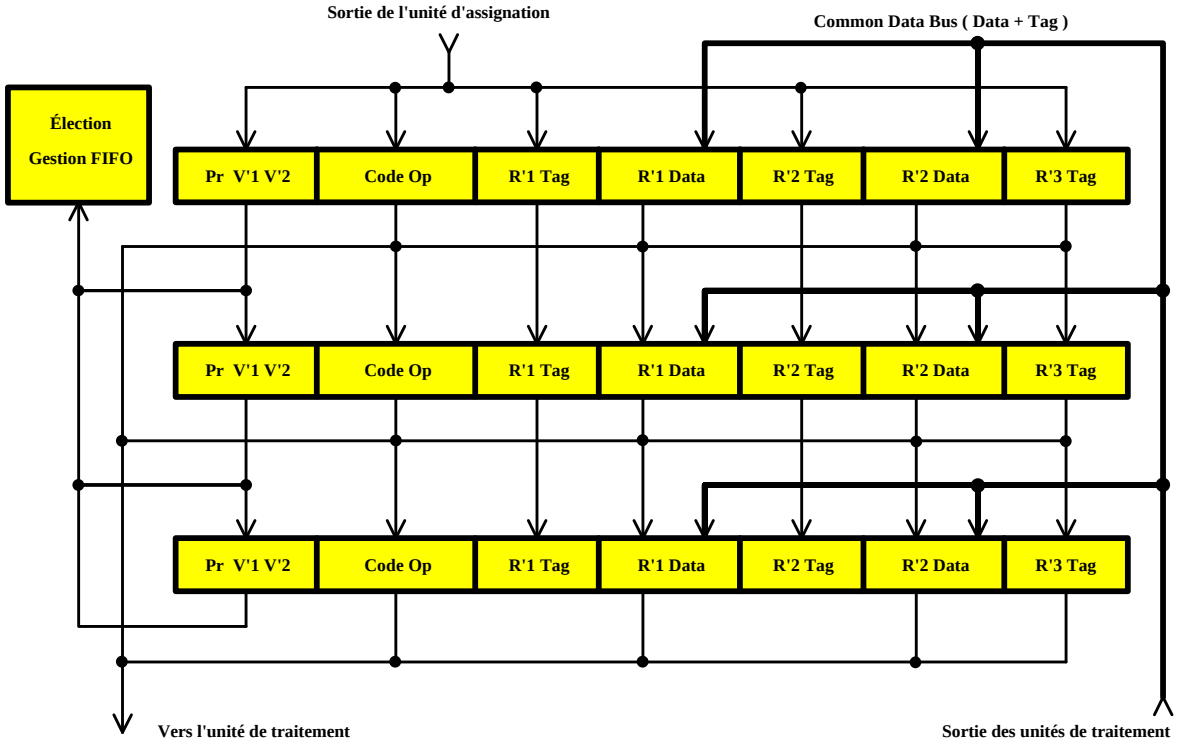
Remarque : cette configuration n'est nécessaire que si l'on dispose de circuits mémoire à accès 1 bit. Dans le processeur, où l'on a accès à tous les bits de la mémoire simultanément, il est plus facile d'utiliser des multiplexeurs et des démultiplexeurs en nombre suffisant pour pouvoir réaliser toutes les opérations de lecture et d'écriture requises.

IV) Unités de dépassement

On retrouve cette unité sous différentes appellations dans la littérature anglo-saxonne, et avec des fonctionnalités diverses (reservation stations / reorder buffer / scheduler buffer).

Chaque unité de dépassement peut être associée à une unité de traitement. Elle traque et enregistre un opérande dès qu'il est disponible, en même temps qu'il peut être écrit dans le registre de destination. Les instructions en attente sont activées dès que tous les opérandes sont disponibles, sans la nécessité du passage de l'information dans les registres.

Le contrôle de l'exécution est décentralisé (*contrôle distribué*) au niveau de chaque unité. Toutes les instructions en attente d'une donnée sont activées simultanément, à l'aide du bus de données partagé, qui diffuse l'information à toutes les unités.



Unités de dépassement

3 + 2 registres sur IBM 360/91 (1967), 8 registres sur Cray 1 (1975), 24 registres sur AMD K6 organisés en une seule unité (1997), 60 registres sur AMD Opteron organisés en plusieurs unités indépendantes (2003).

Les trois étapes de l'algorithme :

- *Production* : s'il existe une place disponible dans l'unité de dépassement, la logique de décodage / réassignation délivre une instruction.

- *Exécution* : exécute l'opération quand les opérandes sont prêts. Si les opérandes ne sont pas prêts, scrute le bus commun pour les enregistrer à leur passage.

- *Écriture* : fin de l'exécution, en plaçant sur le bus commun le résultat de l'opération et son étiquette de destination R'3, vers toutes les unités en attente. Le bit de validité (V'1 ou V'2) est alors activé.

Les instructions sont produites dans l'ordre, exécutées dans le désordre, terminées dans le désordre.

V) Bus de données partagé (CDB - common data bus)

L'utilisation du bus commun permet la diffusion des données sur toutes les unités de dépassement et évite ainsi de nombreuses lectures des registres.

Transitent sur le bus de données partagées les données ainsi que les adresses de destination des informations (numéros des registres physiques et étiquettes mémorisées dans les unités de dépassement (Tags)).

Les accès mémoire sont traités de manière analogue aux autres informations. Les requêtes sont transmises à l'unité qui effectue les opérations de lecture écriture en mémoire (Load / Store), les réponses en provenance de la mémoire sont placées sur le bus commun.

VI) Les inconvénients du "Tomasulo"

- La mécanique est complexe. De nombreuses écritures s'effectuent en mémoire associative, à grande vitesse.

- Les performances sont limitées par le bus de données partagées. Chaque bus de données doit être relié à l'ensemble des unités de dépassement, et aux registres physiques. Ceci induit une grande capacité parasite, et une forte densité de câblage.

- Le nombre d'unité de traitement qui terminent simultanément l'exécution d'une instruction est limité à une unité. L'utilisation de bus de données multiple complique encore le câblage. On doit alors utiliser des mémoires associatives à accès parallèle.

- Les interruptions sont traitées de manière relativement imprécise. Il est nécessaire d'attendre la fin d'exécution de toutes les instructions en attente, pour effectuer une commutation de contexte.

VII) Prédiction de branche

Le mécanisme de Tomasulo permet de superposer différentes itérations d'une même boucle, grâce à la réassignation des registres. Les itérations successives utilisent des registres physiques distincts.

Les unités de dépassement permettent de faire avancer les opérations entières qui gèrent les boucles. L'avancée des itérations permet de saturer les unités de dépassement et les pipe-lines des unités de calcul en virgule flottante, qui présentent le maximum d'étages, et donc la latence maximale.

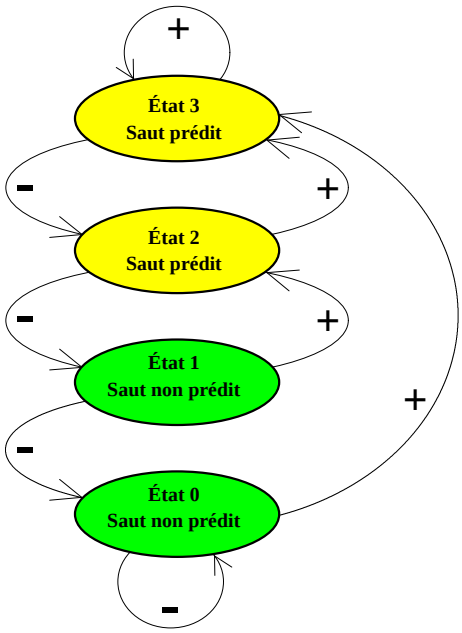
Le résultat d'un test n'est pas nécessairement connus au moment de réaliser un saut conditionnel. Les instructions entières, plus rapides à traiter, permettent de réaliser la prédiction de branche, sans être ralenties par les opérations sur les nombres flottants.

La prédiction de branche s'avère indispensable au bon rendement de toutes les unités de traitement.

Difficulté : que se passe-t-il lorsque la prédiction est erronée ? Le mécanisme doit invalider toutes les instructions qui ont suivi l'instruction de saut. Pour identifier ces instructions, il suffit de numéroter toutes les instructions qui traversent l'unité de décodage.

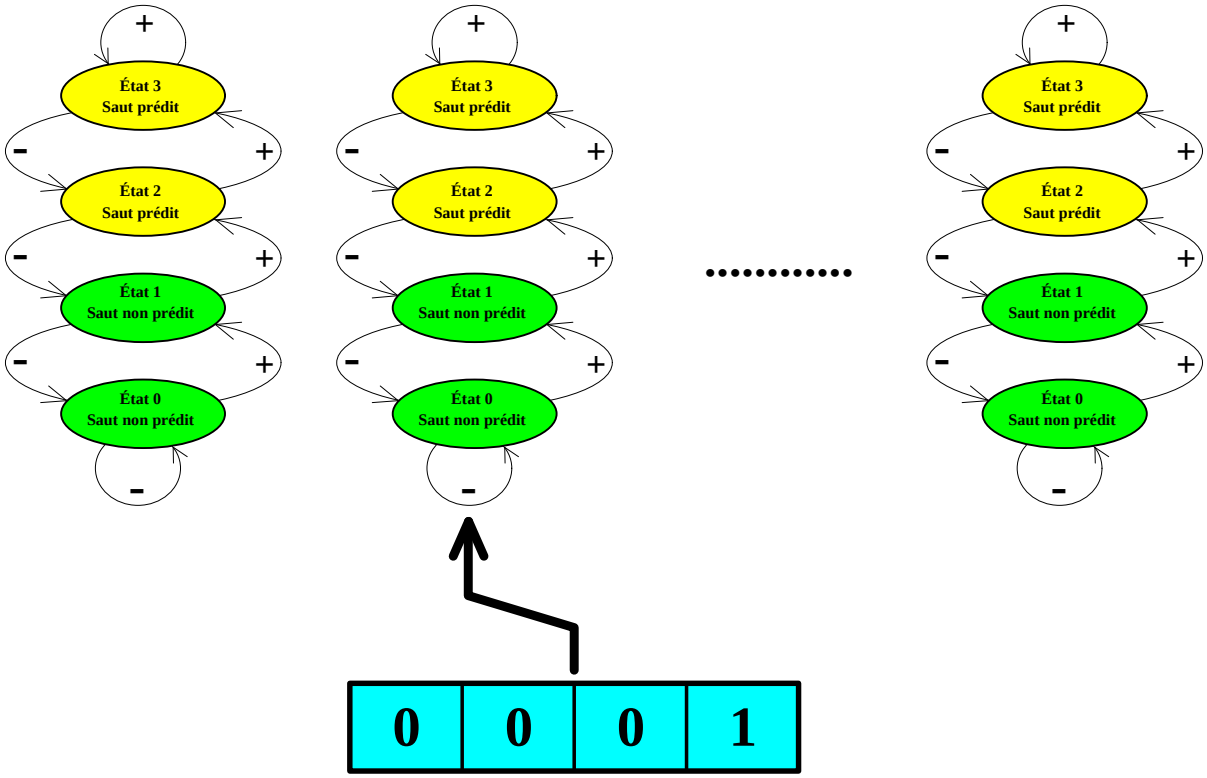
Il faut aussi invalider les écritures de registres physiques et les écritures mémoire qui ont eu lieu. Ceci étant peut être un peu trop difficile on bloquera les écritures mémoire tant que la prédiction n'est pas validée (en utilisant les numéros des instructions de saut comme limite à ne pas franchir).

Réalisation : le Pentium utilise un automate à quatre états qui se comporte comme sur la figure (+ correspond à un saut effectué, – correspond à un saut non effectué). Il semble que ce mécanisme asymétrique n'ai pas été très performant. Mais les erreurs de prédiction sont masquées par le mécanisme d'effacement.



Prédiction de branche dans le Pentium

Les Pentium II et III utilisent un registre à décalage quatre bits et 16 automates à quatre états qui permettent de prévoir le comportement de tous les motifs de longueur 5 et certains motifs de longueur 16. Ce mécanisme est efficace car il peut prévoir des motifs complexes comme 00101001010010100101 .....



Prédiction de branche dans le Pentium II

Toute cette information est enregistrée dans une mémoire 36 bits (4 bits du registre à décalage et 16×2 bits pour les automates). Les adresses des instructions de saut sont hachées et servent à l'accès dans une table de grande dimension (BTB - Branch Target Buffer). On peut aussi enregistrer l'adresse à laquelle doit sauter le programme. La principale difficulté vient du retard qui existe entre la mise à jour de la table et la lecture des instructions par le décodeur. Ce problème ne se pose pas si la boucle est assez longue (en assembleur on prévoit une instruction de saut pour 6 instructions d'autres types - dans un jeu d'instructions complexes CISC).

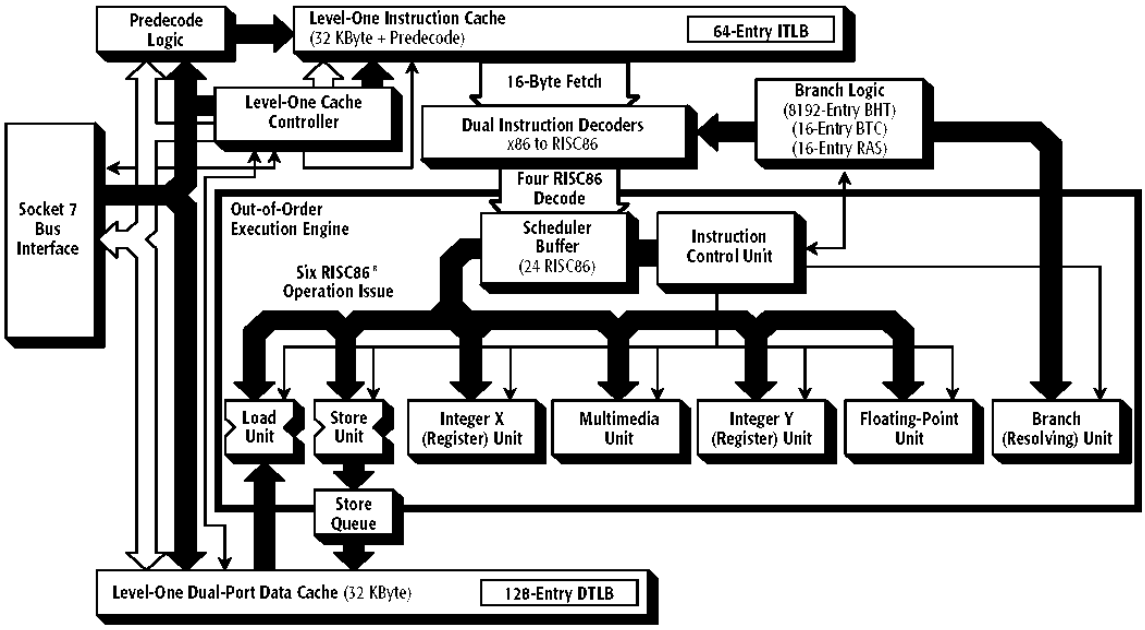
On utilise des tables qui comportent jusqu'à 32K entrées, on obtient des taux de réussite de l'ordre de 95-99 %.

VIII) Hyperthreading

Les ressources du processeur sont attribuées à deux taches indépendantes. L'unité de décodage des instructions lit alternativement le code de chacune des deux taches en exécution. Les autres ressources du processeur sont partagées sans difficultés particulières du fait de l'assignation unique des registres. On limite ainsi les étages de pipe-line qui sont vides, et on espère obtenir deux fois plus de performances qu'avec un processeur ordinaire. Un bit de contrôle permet de déterminer à quelle tache une instruction doit être rattachée.

IX) Réalisations

Dec Alpha 21264, HP 8000, MIPS 10000, Intel Pentium II, III et 4, AMD K6, Athlon, Opteron, Motorola PowerPC 604.



AMD-K6® Processor Block Diagram

Diagramme logique du processeur K6

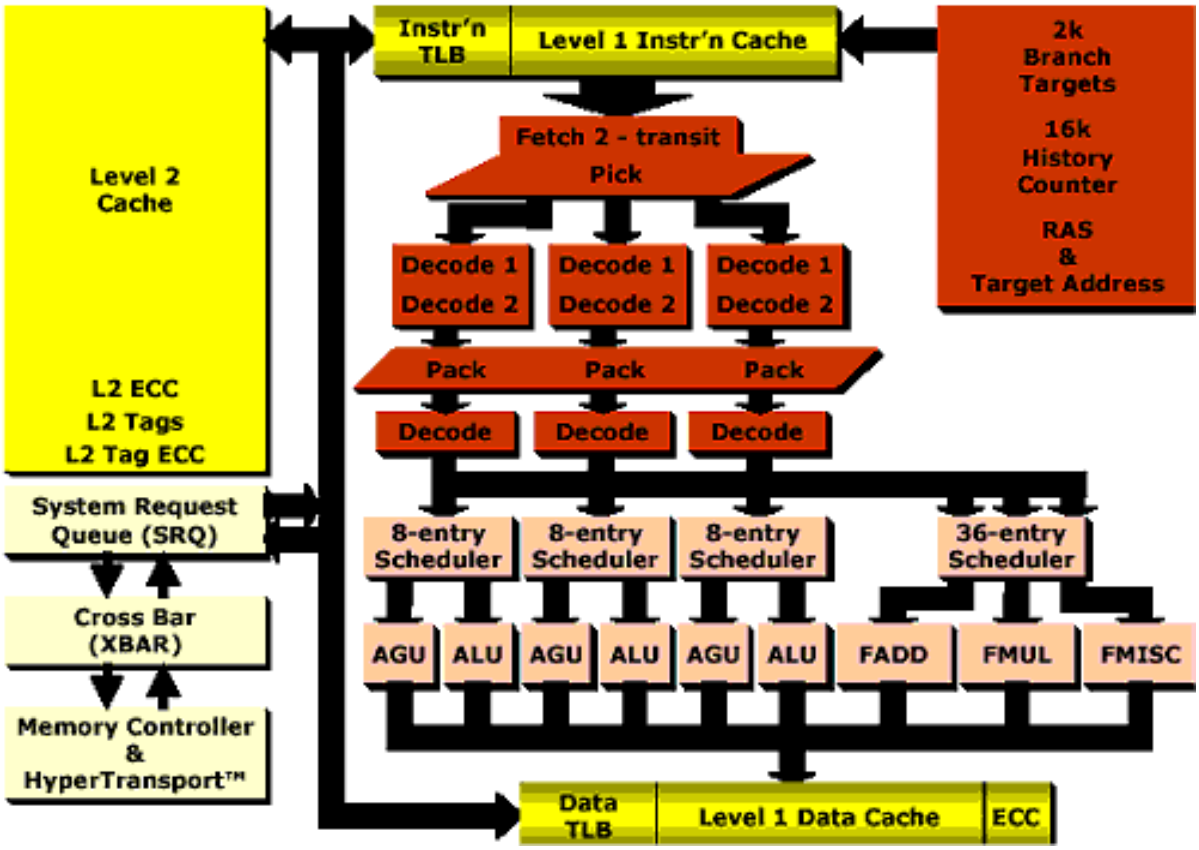


Diagramme logique du processeur Opteron

Références

1. Robert Mikael Tomasulo,
"An Efficient Algorithm for Exploiting Multiple Arithmetic Units"
IBM Journal of R & D, Vol 11, pp 25-33 1967.

2. Exposé CMPUT429/CMPE382 Winter 2001
Author : Randy H. Katz
Email : amaral@cs.ualberta.ca
Home Page : www.cs.ualberta.ca.

3. Design and evaluation of a RISC processor with a Tomasulo scheduler
Diplomarbeit - Universität des Saarlandes - Daniel Kröning 1999.

4. Evaluation of branch prediction methods on trace from commercial applications
R. B. Hilgendorf - G. J. Heim - W. Rosenstiel, IBM Journal of R & D,
Vol 43, No 4 pp579-593 July 1999.

ANNEXES

A Tomasulo Algorithm Example

For this example, the computer begins by executing the following sequence of instructions.

```
DIVD F2, F2, F4     ; F2 <- 1.0, initially F2 = F4 = 1.0
ADDD F4, F2, F4     ; F4 <- 2.0
DIVD F2, F2, F4     ; F2 <- 0.5
ADDD F4, F2, F4     ; F4 <- 2.5
DIVD F2, F2, F4     ; F2 <- 0.2
ADDD F4, F2, F4     ; F4 <- 2.7
DIVD F2, F2, F4     ; F2 <- 0.074
ADDD F4, F2, F4     ; F4 <- 2.774
```

In Tomasulo's algorithm, all registers are modified so that they can hold either data or a tag. In the execution table below, T1 through T7 are tags.

Tags are in effect promissory notes for data that has not yet been computed. They are the key to being able to issue one instruction per cycle. Whenever an instruction takes longer than one cycle to produce a result, a tag is put into the destination register for the instruction. Later instructions that use the register as a source operand get the tag. The functional unit (lets call it FU-L) assigned to the later instruction does not begin its execution phase until the functional unit (lets call it FU-E) assigned to the earlier instruction has computed its result. Then FU-E broadcasts its result on the common data bus along with the tag. All registers that contain tags are listening for their tag, and when they see it they replace their contents with the result. In particular, one of the reservation station source operand registers for FU-L grabs the result broadcast by FU-E.

In the example below at clock 1, a divide instruction is issued with register F2 as its destination register. When the instruction is issued, tag T1 is placed into F2 and the instruction is assigned to DIV Station 1 (this is FU-E). At clock 2, an add instruction is issued that uses F2 as a source operand. It is assigned to ADD Station 1 (this is FU-L), and the tag T1 is copied from register F2 into the reservation station source operand register RS1. ADD Station 1 does not begin executing the add until DIV Station 1 has broadcast its result with tag T1. This happens near the end of clock 5, so the instruction in ADD Station 1 can start executing in clock 6.

		Register contents		DIV Station 1			DIV Station 2			ADD Station 1			ADD Station 2		
Clock	**Issue**	**F2**	**F4**	**Tag**	**RS1**	**RS2**	**Tag**	**RS1**	**RS2**	**Tag**	**RS1**	**RS2**	**Tag**	**RS1**	**RS2**
0		1.0	1.0												
1	1	T1	1.0	1	1.0	1.0									
2	2	T1	T2	Start EX						2	T1	1.0			
3	3	T3	T2				3	T1	T2						
4	4	T3	T4										4	T3	T2
5				End EX			3	1.0	T2	2	1.0	1.0			
6	5	T5	T4	5	T3	T4				Start EX					
7							3	1.0	2.0	End EX			4	T3	2.0
8	6	T5	T6				Start EX			6	T5	T4			
9															
10															
11				5	0.5	T4	End EX						4	0.5	2.0
12	7	T7	T6				7	T5	T6				Start EX		
13				5	0.5	2.5				6	T5	2.5	End EX		
14	8	T7	T8	Start EX									8	T7	T6
15	?														
16	?														
17	?			End EX			7	0.2	T6	6	0.2	2.5			
18	?									Start EX					
19	?						7	0.2	2.7	End EX			8	T7	2.7
20	?						Start EX								
21	?														
22	?														
23	?	0.074	T8				End EX						8	0.074	2.7
24	?												Start EX		
25	?	0.074	2.774										End EX		

Page URL : http://www.d.umn.edu /~gshute/cs2521/arch/tomasuloex.html
Page Author : Gary Shute
Comments to : mailto : gshute@d.umn.edu