



IP PARIS

Introduction à l'Architecture des Microprocesseurs

O.Hammami

ENSTA ParisTech
828 Bvd des Maréchaux 91762 Palaiseau cedex
www.ensta-paristech.fr

Plan

Plan	3
Table des figures	5
Liste des Tableaux.....	8
1 Chapitre 1 : Systèmes à Base de Microprocesseurs	9
1.1 Natures et utilités	10
1.2 Eléments matériels de base structurants	10
1.2.1 Exécution : Processeurs.....	11
1.2.2 Communication : Bus et réseaux d'interconnexions.....	12
1.2.3 Mémorisation : Mémoires	15
1.2.4 Interfaces : Modules d'entrée sorties	16
1.3 Méthodologies de Conception et Implémentation.....	17
1.4 Modèles de Performance	21
1.5 Conclusion	31
2 Chapitre 2 Structure d'un Microprocesseur	34
2.1 Introduction	35
2.2 Jeu d'instructions et éléments de structure	35
2.3 Définition codage d'un jeu d'instructions:	39
2.4 Automates d'états finis et systèmes synchrones.....	40
2.5 Structure cycle unique	41
2.5.1 Construction du chemin de données.....	41
2.5.2 Conception Unité de Contrôle.....	46
2.6 Structure multicycle.....	48
2.7 Extensions du jeu d'instructions.....	51
3 Chapitre 3 Hiérarchie Mémoire.....	53
3.1 Le « Mur Mémoire »	54
3.2 Mémoires Caches	56
3.2.1 Définitions	57
3.2.2 Structure de base de caches	58
3.2.3 Méthode d'accès au cache.....	60
3.3 Performance de cache et outil d'évaluation.....	61
3.4 Surface Silicium des caches.....	63
3.5 Mémoire Virtuelle	66
3.6 Mémoire DDR3, mode burst et défaut de cache	69
Chapitre 4 Structure Parallèle: Pipeline et Superscalaire	72
3.7 Comment accroître les performances ?	73
3.8 Chemin de donnée pipelinée.....	76
3.9 Contrôle Pipelinée	79
3.10 Aléas	80
3.11 Processeur Superscalaire.....	83
3.12 Processeur Superscalaire: compromis prediction de branchement, taille de la fenêtre d'instructions, taille de caches	88
3.13 Evaluation de performances et dimensionnement de microprocesseur superscalaire par simulation : Simplescalar	91
3.14 Environnements automatisés	92
3.15 Optimisation de processeur Superscalaire, Recherche Opérationnelle et Optimisation Multiobjectifs	93
3.16 Conclusion	93

Chapitre 5 - Multiprocesseurs	94
3.17 Introduction.....	95
3.18 Réseaux d'interconnexions	97
3.19 Architectures parallèles et cohérence de caches	99
3.20 Architectures parallèles et cohérence de caches : mémoire partagée avec bus ...	100
3.21 Exemple : Multiprocesseur ARM MP11Core.....	102
3.22 Architectures parallèles et cohérence de caches : mémoire distribuée et réseau d'interconnexion.....	102
3.23 Architectures parallèles et cohérence de caches : mémoire distribuée et réseau d'interconnexion Performance d'applications	104
3.24 Evaluation de performances et dimensionnement de multicore par Simulation.	108
L'évaluation de performances et le dimensionnement des multicore comme pour les architectures de type superscalaire fait appel à des simulateurs (du type Gem5 www.gem5.org) et McPAT (An integrated power, area, and timing modeling framework for multicore and manycore architectures www.hpl.hp.com/research/mcpat/).	109
3.25 Multicore dans les systèmes embarqués	111
3.26 Evaluation de performances par émulation FPGA	112
3.27 Conclusion	113
4 Conclusion.....	114
5 Références	120
6 Appendice A – Rappel sur la logique.....	121
Appendice B – Fabricants de microprocesseurs, microcontrôleurs et DSP. (<i>liste non exhaustive</i>).....	127
7 Glossaire.....	129

Table des figures

Figure 1.Exemples de configurations de systèmes à base de microprocesseurs (1) PC (2) téléphone portable (<i>note : les différents éléments ne sont pas à l'échelle</i>)	11
Figure 3. Exemple de configurations de bus dans un PC	13
Figure 2. Configurations d'associations de bus (a) bus unique (2) deux types de bus : bus processeur-mémoire/bus d'entrées-sorties (3) trois types de bus.....	12
Figure 4. Réseau en anneau	13
Figure 5.Réseaux d'interconnexions : (a) crossbar (b) omega (c) élément de commutation pour le réseau Omega	14
Figure 6.Topologie de réseau fat-tree avec 16 noeuds.....	14
Figure 7 Module Mémoire SDRAM (www.elpida.com)	15
Figure 8.Variations de performance entre les processeurs et la mémoire.....	15
Figure 9.Circuit de codage-decodage MPEG-4 avec DRAM sur puce.....	16
Figure 10.Exemple de Configuration complète	17
Figure 11.Niveaux d'abstraction dans la conception de microprocesseur	17
Figure 12. Flot de Conception.....	19
Figure 13. Processeur Intel P4 (1) vue blocs fonctionnels (2) vue du circuit avant mise en boîtier (3) en boîtier.....	19
Figure 14.Flot de Génération de Processeur et Environnement Logiciel Associe (Tensilica).21	21
Figure 15.Interface de Génération de Processeur (Tensilica www.tensilica.com)	21
Figure 16.Performance Relative de 5 processeurs embarqués pour 3 des 5 benchmarks	23
Figure 17.Prix-performance pour 5 différents processeurs embarqués et 3 des 5 benchmarks EEMBC utilisant uniquement le prix du processeur	23
Figure 18.Performance Relative par Watt consommé pour 5 processeurs embarqués	23
Figure 19 Performance de 7 ordinateurs différents pour les SPEC CINT 2000.	25
Figure 20.Performance de 7 ordinateurs différents pour les SPEC CFP 2000.....	26
Figure 21.Performance sur TPC-C (www(tpc.org)	27
Figure 22.Cout-Performance TPC -C (www(tpc.org)	27
Figure 23.Analyseur Logique (Agilent www.agilent.com).....	28
Figure 24.Mécanisme des compteurs de performance au sein des microprocesseurs (1) mécanisme au sein du P4 (2) microarchitecture du P4	28
Figure 25.Flot de Compilation.	36
Figure 26.Modes d'adresses du processeur MIPS.....	36
Figure 27.Statistiques d'utilisation des modes d'adresses pour 3 programmes	37
Figure 28. Formats de codage des Instructions MIPS.....	39
Figure 29.Mécanisme de lecture instructions (a) éléments structurants (b) connexions des éléments.....	42
Figure 30.Banc de Registres (1) 2 ports de lecture 1 port en écriture (2) Accès Ecriture (3) Accès Lecture.	43
Figure 31.Connexion banc de registres et ALU	44
Figure 32.éléments structurants pour les instructions d'accès à la mémoire (2) mécanisme d'adressage des données.....	44
Figure 33.Chef de donnée	45
Figure 34.mode de calcul de l'adresse pour les instructions de branchements (2) modification associée du chef de donnée	45
Figure 35.Chef de donnée	45

Figure 36. Unité de Contrôle	46
Figure 37. Processeur Monocycle	47
Figure 38. Chemin de donnée modifié	48
Figure 39. Chemin de donnée du Processeur Multicycle	49
Figure 40. Machine d'états de l'unité de contrôle Processeur Multicycle	49
Figure 41. Machine d'états détaillée – Processeur Multicycle	50
Figure 42. Processeur Multicycle	51
Figure 43. Processeur AMD K6-III Incluant des unités 3D Now !	52
Figure 44. Evolution des Performances Microprocesseurs et Mémoires	54
Figure 45. Hiérarchie Mémoire : (a) relation entre taille et temps d'accès (2) relation entre taille et distance au microprocesseur	55
Figure 46. Organisation générale	56
Figure 47. Exemple de microprocesseur avec cache de second niveau intégré	56
Figure 48. Découpage de l'adresse	57
Figure 49. Structure de caches et fonction de correspondances	58
Figure 50. Structure de cache Direct-mapped	59
Figure 51. Structures de cache 4-way set associative	59
Figure 52. Temps d'accès vs associativité	62
Figure 53. Miss rate vs Taille du bloc	63
Figure 54. CACTI : évaluation de la surface en fonction de la taille de cache et de l'associativité	64
Figure 55. CACTI : évaluation de la surface en fonction de la taille de cache et du nombre de ports	65
Figure 56. CACTI : évaluation de la consommation en fonction de la taille de cache, de l'associativité et du nombre de ports (1, 2, 4)	65
Figure 57. CACTI : évaluation du temps d'accès en fonction de la taille de cache et de l'associativité	65
Figure 58. Exemple de circuit avec cache 16KL1I, 16K L1D, 1MB L2I, 256 K L2D, 12MB L3	66
Figure 59. Exemple de circuits avec caches	66
Figure 60. Mémoire Virtuelle	67
Figure 61. Mécanisme d'accès	68
Figure 62. Traduction d'adresses	68
Figure 63. TLB	69
Figure 64. Architecture composant mémoire DDR3	70
Figure 65. Chronogramme opération de lecture mode burst DDR3	70
Figure 66. Chronogramme opération d'écriture DDR3	71
Figure 67. Exécution Séquentielle vs Exécution Pipeline	74
Figure 68. Cas d'un branchement conditionnel : (a) condition non vérifiée (b) condition vérifiée	75
Figure 69. Branchement retardé (delayed branch)	75
Figure 70. Principe du Forward	76
Figure 71. Dépendance entre instruction de lecture mémoire et instruction de type R	76
Figure 72. vDécoupage préliminaire du chemin de donnée	77
Figure 73. Découpage du Chemin de donnée par introduction de registres	78
Figure 74. Modification du chemin de donnée : prise en compte du registre destination	78
Figure 75. Introduction du support pour beq/alu control	79
Figure 76. Transmission des signaux de contrôle	79
Figure 77. Processeur Pipeline v1	80
Figure 78. Unité de Forward	81

Figure 79. Unité de Détection d'Aléas (Hazard Detection Unit)	82
Figure 80. Potentiel en Instructions exécutables en parallèle	83
Figure 81. Structure de Processeur Superscalaire	84
Figure 82. Impact de la Prédiction de Branchement sur le nombre d'instructions exécutables en parallèle	85
Figure 83. Machine d'états (2 bits) pour la prédiction de branchement	85
Figure 84. Machine d'états (2 bits) pour la prédiction de branchements (variante Hysteresis scheme).....	85
Figure 85. Structure de BTB	86
Figure 86. Procédure de Prédiction de Branchement a base de BTB	87
Figure 87. Processeur AMD Athlon (www.amd.com).....	87
Figure 88. IPC en fonction de la taille de cache donnee et de la taille du RUU	89
Figure 89. IPC en fonction de la taille du cache donnee et de la taille du RUU avec un predicteur parfait.	90
Figure 90. IPC en fonction de la taille du cache instruction et de la taille du RUU.	91
Figure 91. Architecture multiprocesseur à base de bus.....	95
Figure 92. Architecture multiprocesseur avec plusieurs niveaux de cache.....	96
Figure 93. Architecture multiprocesseur à mémoire distribuée	96
Figure 94. Evolution Architecture multiprocesseur - Intel	97
Figure 95. Architecture multiprocesseur	98
Figure 96. temps de propagations porte logique et connexions en fonction de la technologie semiconducteur.....	98
Figure 97. Réseaux d'interconnexions pour architecture multiprocesseur	99
Figure 98. Cohérence de cache: exemple.....	99
Figure 99. Architecture multiprocesseur à base de bus avec cohérence de caches.....	100
Figure 100. Protocole de cohérence de caches write-invalidate et cache write-back.	101
Figure 101. Architecture multiprocesseur avec cohérence de caches	101
Figure 102. Architecture ARM	102
Figure 103. Architecture multiprocesseur à mémoire distribuée et protocole de cohérence de type directoire.....	103
Figure 104. Protocole de cache pour directoire.....	103
Figure 105. Architecture dual core avec cache L2 et quadcore avec cache L3 partagé (intel)	104
Figure 106. Impact sur le taux de défaut de cache du nombre de processeur	105
Figure 107. Impact sur le taux de défaut de la taille du cache	105
Figure 108. Analyse de l'impact de la taille du bloc sur le défaut de cache.....	106
Figure 109 Impact sur le trafic sur le bus de l'augmentation de la taille du bloc	106
Figure 110. Analyse de l'impact du nombre de processeurs sur le défaut de cache.....	107
Figure 111. Analyse de l'impact de la taille du cache sur le défaut de cache	107
Figure 112. Analyse de l'impact de la taille du bloc sur le défaut de cache.....	108

Liste des Tableaux

Tableau 1.Catégories de Processeurs	11
Tableau 2.exemples de Bus	12
Tableau 3.Mémoires	15
Tableau 4 Modules Interfaces homme-machine	16
Tableau 5.Langage de Description pour Microprocesseurs	18
Tableau 6. – Caractéristiques des trois niveaux essentiels d'abstraction pour des propriétés intellectuelles (IP cores)	20
Tableau 7.Types d'IPs disponibles (IBM Microelectronics)	20
Tableau 8– Exemples de Benchmarks	22
Tableau 9.Benchmark pour processeurs Embarqués (EEMBC)	22
Tableau 10.Benchmarks pour Systèmes Généralistes	24
Tableau 11.– Benchmarks SPEC CINT2000	24
Tableau 12.Benchmarks SPEC CFP 2000	25
Tableau 13.Paramètres requis pour valider un résultat d'évaluation avec les benchmarks SPEC	26
Tableau 14.Exemples de programme d'analyse de performance	29
Tableau 15.Exemples de Microprocesseurs Commerciaux	35
Tableau 16.Jeu d'instructions du MIPS	38
Tableau 17.Codage des instructions assembleur MIPS	40
Tableau 18.Elements Structurants de base	41
Tableau 19.Codage en entrée de l'ALU	44
Tableau 20.Description des Signaux de Contrôle	46
Tableau 21.Formats des Instructions Assembleur MIPS	47
Tableau 22.Spécification des signaux de contrôle pour Processeur Monocycle.....	47
Tableau 23 Instructions de gestion du cache sur IBM/Motorola PowerPC	61
Tableau 24. Exemple: Cache 4M octets + bits de parité, bloc de cache 32 octets, associativité 4, nombre de ports de lecture/écriture 1 - technology 90 nm: surface 128.7 mm ²	64
Tableau 25.Formats des Instructions Assembleur MIPS	81

Chapitre 1 : Systèmes à Base de Microprocesseurs

1.1 Natures et utilités

Un système à base de microprocesseurs est un système électronique programmable composé d'au moins un microprocesseur et d'un ensemble de composants et structures électroniques permettant l'exécution de programmes informatiques. Ce système est obligatoirement ouvert en permettant la communication avec le monde extérieur. Cette communication peut prendre la forme classique d'entrées sorties du type écran ou clavier mais aussi de capteurs divers (chimiques, mécaniques) ou d'actionneurs. Les systèmes à base de microprocesseurs sont omniprésents dans la vie économique et sont utilisés pour des tâches variées dont de manière non exhaustive:

- Contrôle (automobile, aviation et espace , ...)
- Calcul scientifique (modélisation, simulation ...)
- Analyse de données (analyse financière, économique, statistique...)
- Divertissement (console de jeux, ...)
- Biomédical , IoT (Internet-of-things),

L'industrie et les services des systèmes à base de microprocesseurs sont une des premières industries dans le monde en termes de revenu et représentent une valeur ajoutée technologique stratégique fondamentale. Les possibilités de progression de cette industrie sont encore très nombreuses dans des domaines en cours d'adoption des techniques à base de systèmes à microprocesseurs (exemple : automobile) ou dont les potentialités en termes de marché sont larges (domotique).

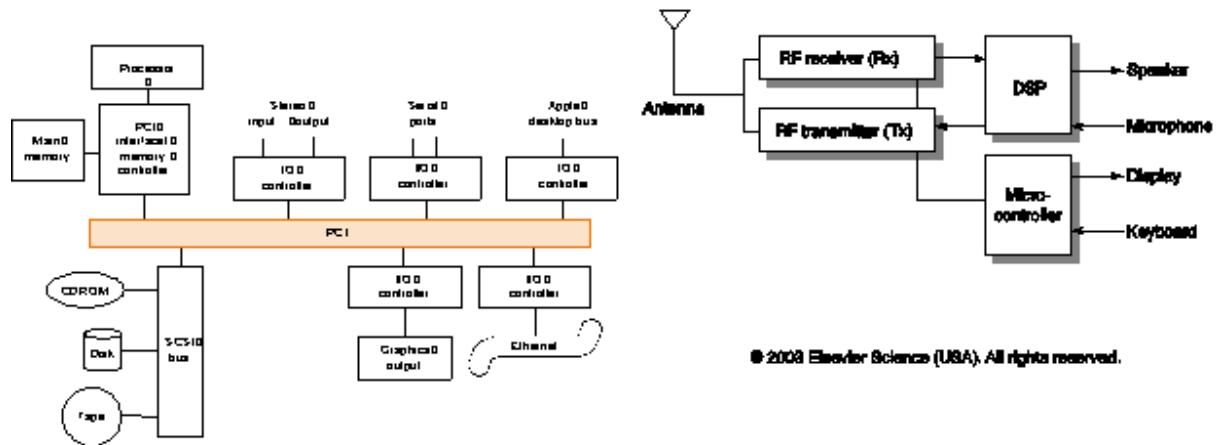
L'industrie des systèmes à base de microprocesseurs et ses dérivés (industrie du semiconducteur) sont caractérisés par des progrès réguliers sur les critères principaux: (1) du coût (2) de la consommation d'énergie (3) du temps d'exécution. La très célèbre Loi de Moore stipulant le doublement des capacités tous les 18 mois et qui a été régulièrement vérifiée au cours des 30 dernières années en est la meilleure illustration. Ces critères principaux permettent à leur tour d'augmenter les domaines d'applications ainsi que les fonctionnalités associées. L'impact est aussi une durée de vie des produits de plus en plus courte et une compétition extrêmement sévère entre les différents acteurs de l'industrie.

La notion de temps d'apparition au marché (Time To Market – TTM) d'un produit y est encore plus stricte que dans d'autres secteurs économiques et domine l'ensemble des activités et leurs rythmes.

1.2 Eléments matériels de base structurants

Les systèmes à base de microprocesseurs sont composés d'éléments matériels électroniques de base. Ces éléments électroniques peuvent être classés dans une des catégories suivantes suivant leurs fonctionnalités : (1) calcul : processeur (2) mémorisation : mémoire (3) communication : bus ou réseau d'interconnexion (4) interfaçage : entrée-sortie. Au sein de chaque catégorie il existe une grande variété d'éléments plus ou moins compatibles avec les éléments des autres catégories et qui permettent de construire un système à base de microprocesseurs. Quoique l'assemblage d'éléments disponibles commercialement (composants sur étagère) peut à priori apparaître accessible relativement aisément la complexité des fonctionnalités et caractéristiques de chaque élément au sein de chaque classe rend leur aggrégation très complexe.

Ainsi la construction à partir d'éléments préexistants peut apporter une valeur ajoutée importante dans l'identification des éléments les plus adaptés et une construction finale efficace en rapport prix-performance.



**Figure 1.Exemples de configurations de systèmes à base de microprocesseurs (1) PC (2) téléphone portable
(note : les différents éléments ne sont pas à l'échelle)**

Un téléphone portable incluant un DSP et un microcontrôleur pourra être considéré comme un système à base de microprocesseurs.

1.2.1 Exécution : Processeurs

Les processeurs sont les uniques éléments calculatoires programmable d'un système à base de microprocesseurs. Il existe 5 grandes catégories de microprocesseurs : (1) généraliste (2) embarqué (3) traitement de signal (4) multimédia (5) réseaux. Cette catégorisation informelle se fait sur la base des conditions d'utilisation ou des applications visées. Les processeurs généralistes ont pour vocation d'exécuter les applications dans un environnement non spécialisé comme celui supporté par un ordinateur personnel. Cette vocation généraliste fait que ces processeurs sont sous-optimaux pour des tâches spécialisées mais reste compétitifs dans le cas général. Les processeurs embarqués sont des processeurs ayant pour vocation d'être inclus dans un système à priori mobile sous contrainte de consommation d'énergie et sont utilisés de manière très intensive dans les industries de l'aviation et de l'espace , de l'automobile, des télécommunications et de l'armement. Les DSP développés depuis 1980 sont plus spécialement orientés pour les applications de traitement de signal. Enfin deux catégories plus récentes sont apparues : (1) les processeur multimédia dont l'émergence a suivi l'émergence du multimédia dans les applications informatiques (vidéo, etc...) et (2) les processeurs réseaux qui découlent de l'émergence et de la prolifération des réseaux informatiques et que l'on retrouve en particulier dans les routeurs de réseaux.

Tableau 1.Catégories de Processeurs

Type de processeur	Exemple
Généraliste	AMD Opteron, Intel core i7 ,PowerPC G7
Embarqué	ARM T9TDMI, MIPS32, PowerPC 403
Traitement du signal	DSP TI C6711, Analog Devices SHARC
Multimédia	Philips Trimedia TM-1300, Nvidia
Réseau	Intel IXP2400

Un système à base de microprocesseurs peut utiliser différentes catégories de processeurs chacune dédiée à une ou partie des applications qui le compose. De même il est tout a fait possible qu'un système possède un grand nombre de processeurs du même type. Les combinaisons de catégories et le nombre de représentants par catégories sont des choix du concepteur du système.

1.2.2 Communication : Bus et réseaux d'interconnexions

Un processeur doit obligatoirement communiquer pour lire/fournir des données et cette communication se fait par des bus ou des réseaux d'interconnexions. Un bus ou un réseau d'interconnexion peut être vu sous ses deux angles : (1) l'angle physique : la matière qui le compose (fils de cuivre, câble coaxial, fibre optique, etc) et ses limitations physiques (nombre d'éléments pouvant être connectés/bande passante des données) (2) l'angle fonctionnel : ses fonctionnalités et son protocole de communication. Le bus est l'élément de communication le plus élémentaire dans sa topologie puisque c'est essentiellement un segment de droite. Il existe de nombreux types de bus dans leur grande majorité standardisés (exemple : SCSI standard ANSI X3.131). Ces standards garantissent entre autres la compatibilité entre des composants périphériques et les ordinateurs.

Tableau 2. Exemples de Bus

Type	Utilisation
PCI	Bus PC
PCI-X	Bus PC (extension)
VME	Bus industriel, militaire

Les spécifications fonctionnelles d'un bus peuvent évoluer avec les besoins mais il est possible qu'un standard de bus atteigne les limites de cette évolution et soit abandonné au profit de nouveaux standards. L'utilisation d'un bus peut se faire de manière conjointe avec d'autres bus de types différents et l'interfaçage entre bus se fait par des adaptateurs de bus (bus adapter). Cela permet de concevoir différentes topologies pour les systèmes permettant de répondre au mieux aux besoins en performance.

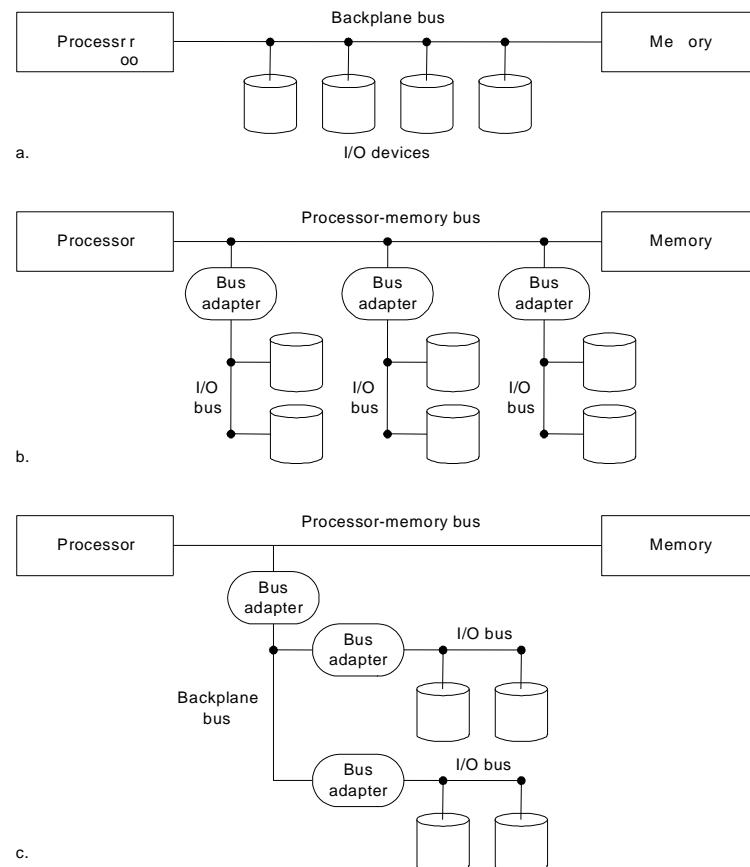


Figure 2. Configurations d'associations de bus (a) bus unique (2) deux types de bus : bus processeur-mémoire/bus d'entrées-sorties (3) trois types de bus.

La figure 2 montre trois topologies de systèmes construites par l'association de bus principaux (processor-memory bus) et des bus d'entrée-sorties (I/O bus).

Un exemple pratique de PC consiste à avoir comme décrit dans la figure 3 une carte électronique permettant la connection de cartes externes à différents bus de types PCI, et ISA (bus ancien) coexistants.

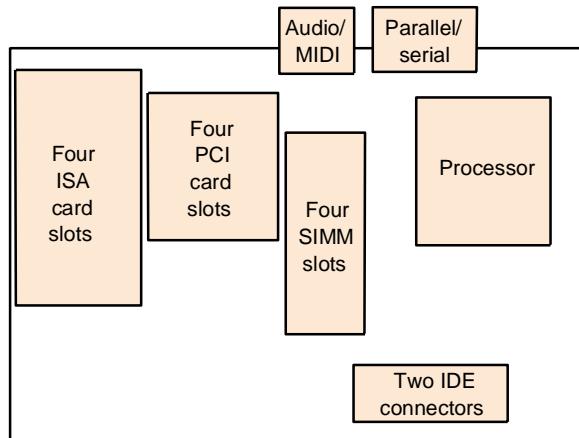
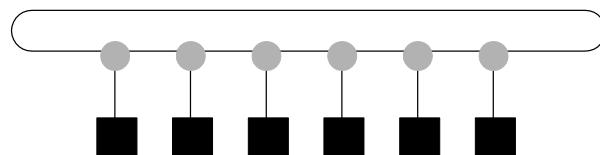


Figure 3. Exemple de configurations de bus dans un PC

Les réseaux d'interconnexions sont une extension de la notion de bus. Les réseaux d'interconnexion ont leurs justifications dans les limitations physiques des bus (nombre maximum de composants pouvant être connectés) et dans les possibilités de communications parallèles. L'extension la plus simple d'un bus passe par un réseau en anneau qui permet de nombreux transferts simultanés de données.



© 2003 Elsevier Science (USA). All rights reserved.

Figure 4. Réseau en anneau

A l'autre extrême de cette topologie se situerait le réseau crossbar décrit Figure 5 qui permet de faire communiquer n'importe quelle paire de noeuds du système en une étape. Le coût du réseau crossbar (n^2 commutateurs) fait qu'il est utilisé généralement pour des tailles relativement petites à moyenne de réseau. Le réseau Omega est moins coûteux ($2n\log_2 n$ commutateurs) mais du fait de sa topologie rend possible des conflits d'accès lorsque deux noeuds du système souhaitent transiter par le même commutateur obligeant à un arbitrage et donc à une réduction de performance.

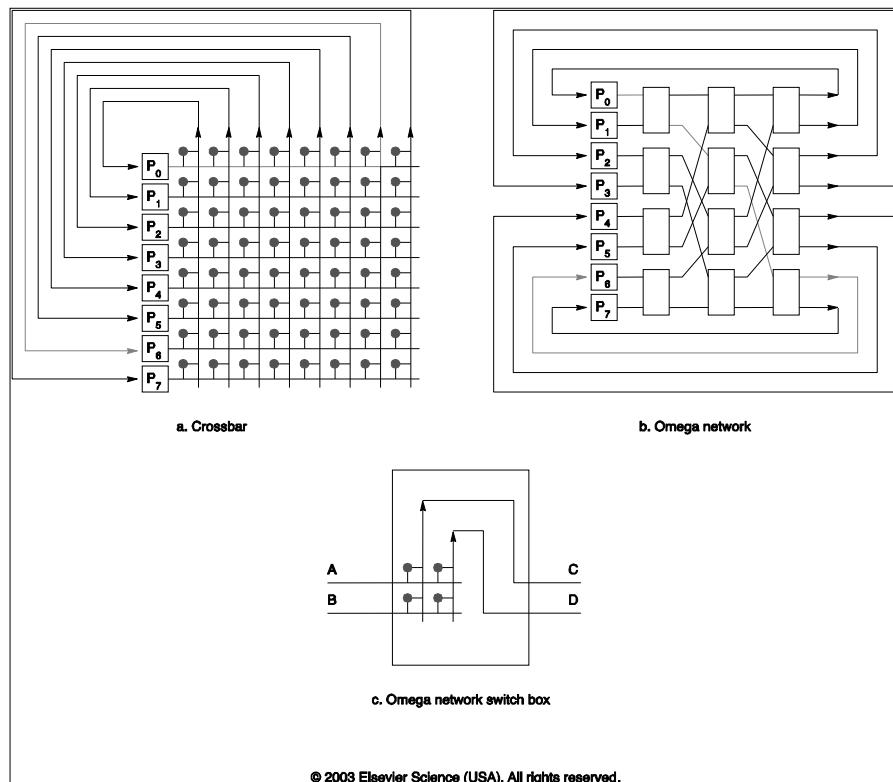


Figure 5.Réseaux d'interconnexions : (a) crossbar (b) omega (c) élément de commutation pour le réseau Omega

D'autres types de réseaux d'interconnexions ont été proposés et implémentés dans des systèmes commerciaux comme les configurations en grille 2D (2D grid, 2D Torus) et les hypercubes.

Des systèmes de grande taille comme la Connection Machine CM-5 ont utilisés le réseau fat-tree ci-dessous.

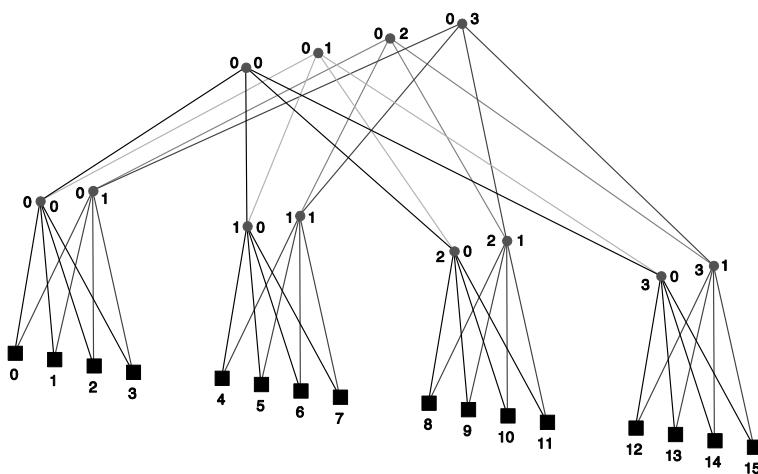


Figure 6.Topologie de réseau fat-tree avec 16 nœuds

Le choix d'un réseau d'interconnexion est basé sur : (1) les caractéristiques des communications entre nœuds du système (débit des échanges de données, simultanéité des échanges) (2) le nombre de nœuds du système et bien entendu (3) le coût.

1.2.3 Mémorisation : Mémoires

Les mémoires dans un système à base de microprocesseurs servent à mémoriser l'ensemble des données et des codes des applications utilisées. Il existe plusieurs catégories de mémoire classifiables essentiellement par la technologie utilisée pour les fabriquer mais aussi par leur volatilité (capacité à conserver les données).

Tableau 3.Mémoires

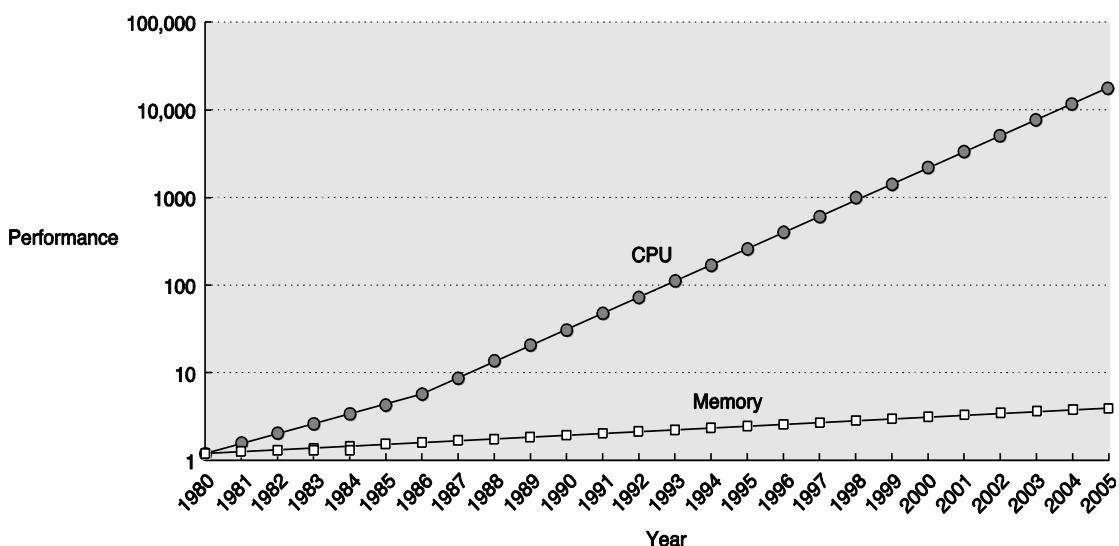
Type	Description
SRAM	Static Random Access Memory Mémoire volatile haute performance - Utilisée pour les mémoires - caches
DRAM	Dynamic Random Access Memory Mémoire volatile utilisée pour la mémoire centrale (SDRAM/RDRAM/ DDR/DDR2, DDR3, DDR4, etc...)
FLASH	Mémoire non volatile Utilisée pour données/programme résidents (portable/pda/...)

La vue la plus commune des mémoires est celle sous forme de module (barrettes) utilisées pour l'extension des mémoires centrales d'ordinateurs personnels.



Figure 7 Module Mémoire SDRAM (www.elpida.com)

La sélection des mémoires les plus appropriées pour un système à base de microprocesseurs est un problème difficile et connu comme le problème de la hiérarchie mémoire. Le problème de la hiérarchie mémoire est de construire une organisation mémoire qui puisse compenser les différences en performance dues aux technologies semiconducteurs entre les processeurs et la mémoire. Comme le montre la figure suivante cette différence tend à s'accroître avec le temps.

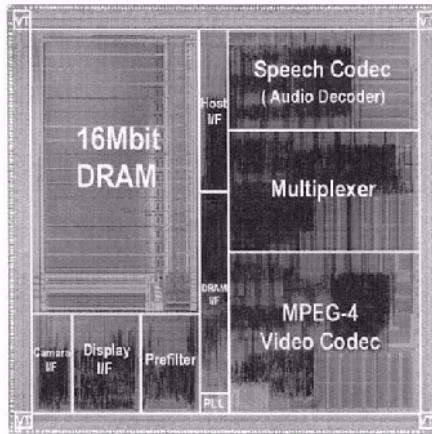


© 2003 Elsevier Science (USA). All rights reserved.

Figure 8.Variations de performance entre les processeurs et la mémoire

La construction de cette hiérarchie doit effectuer des compromis nécessaires entre les différents types de mémoire en termes de performance (temps d'accès /lecture/écriture) – densités – coûts.

De plus les différents types de mémoires sont disponibles en composants externes (barrettes mémoires Figure 8) mais peuvent aussi être implantés directement dans le circuit comme pour cet exemple de circuit effectuant le codage MPEG-4 et qui contient de la DRAM en interne.



Chip micrograph.

Figure 9.Circuit de codage-decodage MPEG-4 avec DRAM sur puce

1.2.4 Interfaces : Modules d'entrée sorties

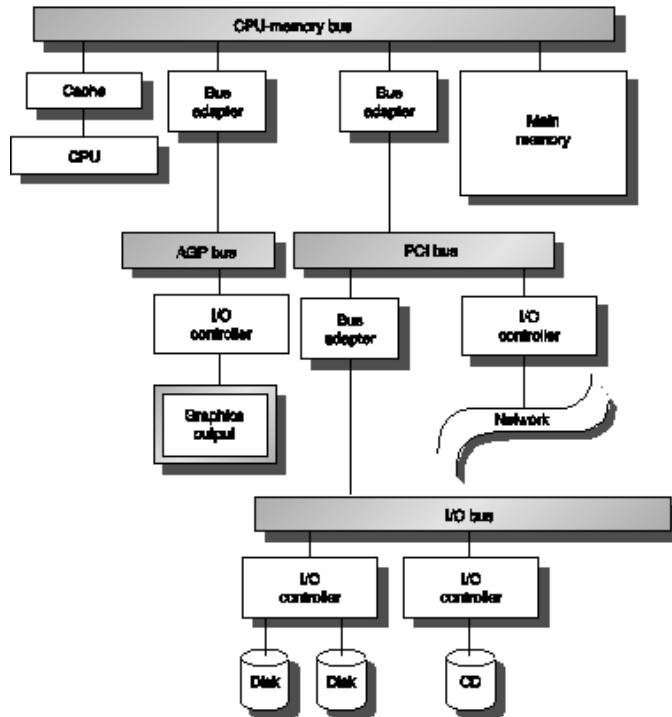
Les modules d'entrées-sorties dans un système à base de microprocesseurs sont extrêmement divers. Ils incluent les traditionnels éléments nécessaires à l'interface homme-machine (clavier/écran/souris/microphone/webcam...) mais aussi les modules d'instrumentation (capteurs chimiques/mécaniques/...), les modules de contrôle (contrôle de moteurs/...), les modules de communication (réseau : Ethernet/sans fil/...).

Les modules d'entrées-sorties se caractérisent globalement par : (1) leur débit : le débit de données par seconde pouvant être transférés entre le module d'entrée-sortie et le processeur et la mémoire (2) type d'accès : lecture seule/écriture seule/lecture écriture.

Tableau 4 Modules Interfaces homme-machine

Type	Description
Interface homme-machine	Clavier/écran/souris/microphone...
Instrumentation	Capteurs (chimiques/mécaniques/...)
Communication	Réseau Ethernet/sans fil (WLAN)
Sauvegarde	Cd-rom/disque optique-magnétique/DAT/...

La configuration finale d'un système à base de microprocesseurs doit donc considérer les quatre fonctionnalités (catégories) décrites précédemment et sélectionner les éléments structurants permettant d'obtenir le meilleur rapport prix-performance. La Loi d'Amdhal spécifiant qu'une amélioration de performance pouvant être obtenue par l'utilisation d'un mode d'exécution plus rapide est limitée par le pourcentage du temps pendant lequel ce mode peu être utilisé oblige à une distribution des ressources dans le système pour l'amélioration globale du rapport cout-performance.



© 2005 Elsevier Science (USA). All rights reserved.

Figure 10.Exemple de Configuration complète

1.3 Méthodologies de Conception et Implémentation

La conception de systèmes à base de microprocesseurs peut être dirigée par des standards et normes de conception dans des produits de type PC ou libre comme pour les systèmes embarqués dédiés.

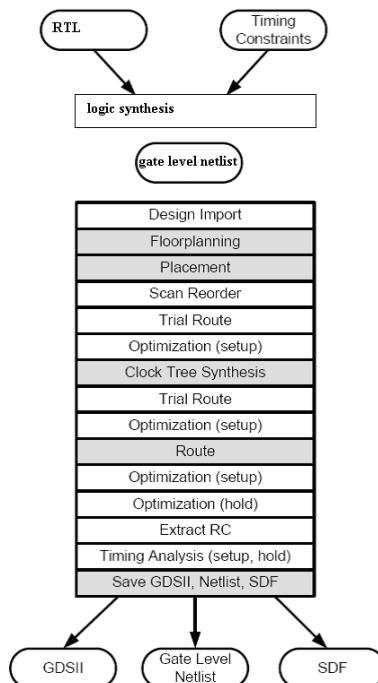


Figure 11.Niveaux d'abstraction dans la conception de microprocesseur

La conception de microprocesseur suit la méthodologie de conception de circuits semiconducteurs. Cette conception est fortement liée aux niveaux d'abstractions de représentation d'un circuit comme décrit dans la figure 11.

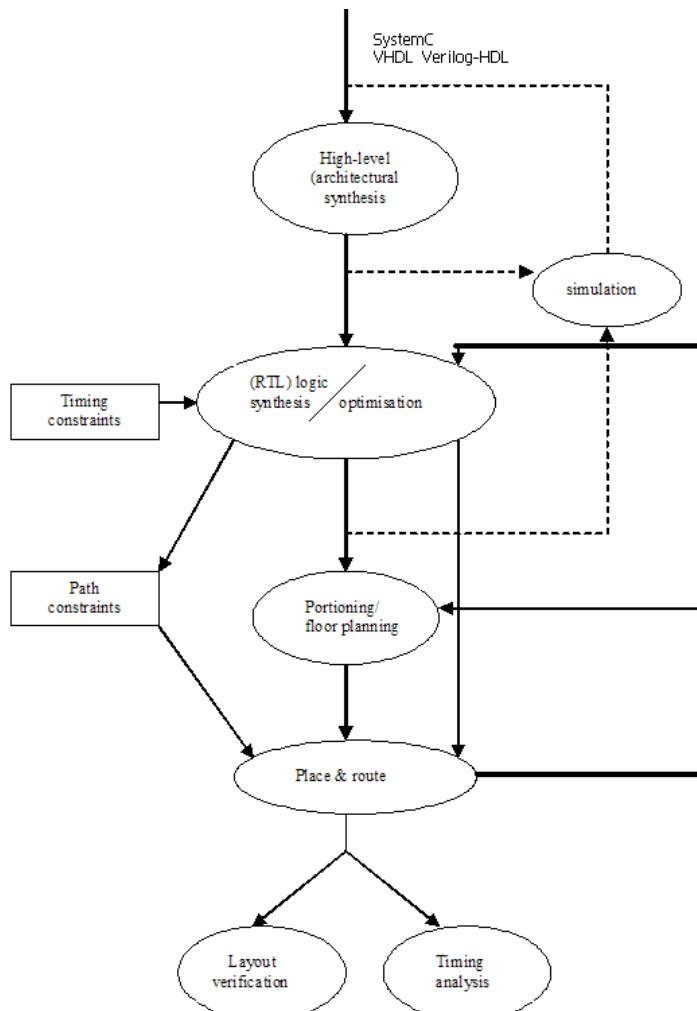
Les outils logiciels de synthèse transforment automatiquement une description à un niveau plus élevé dans la hiérarchie d'abstraction en une description moins élevée dans la hiérarchie. Ces outils de synthèse font partie de l'ensemble des outils de la conception assistée par ordinateur orientés électronique (EDA – Electronic Design Automation). Au plus haut niveau de la hiérarchie de description se trouve la description fonctionnelle : le comportement de l'ensemble du système (du microprocesseur) est décrit de manière comportementale et cela inclut le cœur du microprocesseur, la mémoire, les éléments d'entrée sorties. Les langages de description au niveau fonctionnel incluent les langages de description matériel (Hardware Description Langage) comme VHDL et Verilog-HDL mais aussi de nouveaux langages basés sur C++ comme SystemC utilisé de manière croissante en conception de systèmes sur puce.

Tableau 5.Langage de Description pour Microprocesseurs

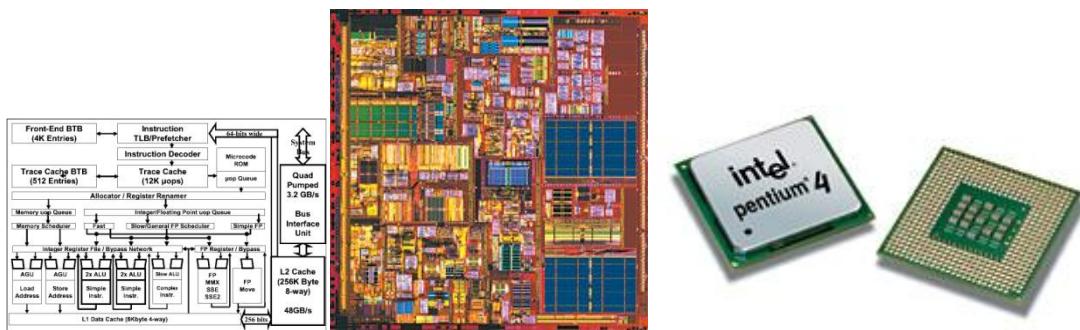
Langage de description	commentaires
VHDL	www.accellera.org
Verilog-HDL	www.accellera.org
SystemC	www.systemc.org

Un ensemble d'outils commerciaux de synthèse existe acceptant en entrée ces langages (entre autres de Synopsys (www.synopsys.com) et Cadence (www.cadence.com)) dans le but d'être synthétisés.

Le niveau suivant dans la hiérarchie d'abstraction est le niveau RTL (Register Transfer Level). La description en RTL se base sur le fait que la majorité des systèmes peut être considéré comme une collection de registres qui mémorisent des données binaires et qui sont échangées et traitées entre ces registres. Le niveau suivant est le niveau porte logique (Logic Gate level). L'outil de synthèse appliqué au niveau RTL effectue une correspondance avec une bibliothèque de cellules de portes logiques (logic gates). Ces bibliothèques incluent les portes de base de type AND/OR/XOR/NOT/NOR/NAND mais aussi des cellules plus complexes. Le niveau transistor est l'avant dernier niveau et cette description dépend de la technologie choisie (CMOS, ...) et le style de logique choisie telle que CMOS statique ou dynamique. Le nombre de transistors est souvent utilisé comme une mesure de la complexité d'un circuit semi-conducteur ainsi que des progrès effectués d'une génération de microprocesseur à l'autre. A titre d'exemple le processeur Intel Pentium III fait 28 millions de transistors tandis que son successeur le processeur Intel Pentium 4 fait 42 millions de transistors. Cette mesure en fait est une mesure strictement quantitative des ressources silicium affectées à l'implémentation d'un processeur. Le fait que des ressources supplémentaires soient affectées n'implique bien évidemment pas que ces ressources représentent une architecture plus efficace en termes de performances. On peut au mieux supposer qu'a ressource supplémentaire on soit en droit d'attendre davantage de performances mais il ne peut exister de loi de performance qui relie une quantité matérielle brute (nombre de transistors) et une mesure de performance et ce a cause de l'utilisation large qui peut être faite de ces transistors. Le dernier niveau est le niveau layout. A ce niveau les bibliothèques de cellules sont complètement dépendantes de la technologie semi-conducteur employée (HCMOS . BICMOS,...). Les niveaux d'abstraction réduisent la complexité des tâches de conception à des niveaux acceptables mais sont aussi accompagnés par des problèmes de vérification. La simulation reste la technique de base pour la vérification et des simulateurs commerciaux pour chacun des niveaux existent. Néanmoins cette tâche de vérification croît de manière exponentielle avec la croissance exponentielle des circuits et constitue un des défis majeurs pour les futures générations de circuits.

**Figure 12. Flot de Conception**

La figure 12 donne une vue plus détaillée de la figure 11 et spécifie de plus la possibilité d'inclure des contraintes et les itérations possibles entre niveaux. La forme finale d'un microprocesseur en tant que produit est un boîtier comme décrit dans la figure suivante.

**Figure 13. Processeur Intel P4 (1) vue blocs fonctionnels (2) vue du circuit avant mise en boîtier (3) en boîtier**

Il est néanmoins possible d'acheter un processeur sous une autre forme suivant le niveau d'abstraction dans la hiérarchie. Les trois principaux niveaux sont définis comme : (1) soft (2) firm et (3) hard. Le tableau suivant décrit les caractéristiques de ces trois formes.

Tableau 6. – Caractéristiques des trois niveaux essentiels d'abstraction pour des propriétés intellectuelles (IP cores)

	Design flow	Representation	Libraries	Technology	probability
Soft Not predictable Very flexible	System design	Behavioural	N/A	Technology independent	unlimited
	RTL design	RTL			
Firm Flexible predictable	floor planning synthesis placement	RTL & blocks netlist	reference library -footprint -timing mode -wiring model	Technology generic	Library mapping
Hard Not Flexible very predictable	Routing verification	Polygon data	Process-specific Library & design Rules - characterized cells - process rules	Technology fixed	Process mapping

Il existe un marché croissant pour les services de conception de propriétés intellectuelles et de nombreux constructeurs proposent des catalogues de composants aux niveaux décrits précédemment.

Tableau 7.Types d'IPs disponibles (IBM Microelectronics)

Category	Examples of cores available
Processors	IBM PowerPC® with CoreConnect™ interfaces, ARM, ZSP400 DSP
Processor peripherals	DMA controllers, memory and peripheral controllers, interrupt controller, general purpose I/O and timers with CoreConnect interfaces
IBM CoreConnect bus structures	CoreConnect processor local bus and on-chip peripheral bus arbiters and bridges; bridges to other on-chip bus architectures (AMBA, ZSP)
Embedded FPGA	Xilinx
Communications	Ethernet and HDLC functions
High-speed links	High-speed SERDES , Hypertransport, InfiniBand, rapid I/O, XAUI, system packet interface functions
Serial interfaces	I ² C, IrDA, serial host transport layer, serial communication port, smart card interfaces, UART
Bus interfaces	PCI, PCI-X, Arapahoe, GTL, HSTL, IDE, IEEE 1284, SCSI and USB functions
Data compression	ALDC, ELDC
Multimedia	LCD, NTSC/PAL functions
Data converters	ADC, DAC functions
Security	Cryptography functions

Le tableau précédent montre une offre de propriétés intellectuelles pour différentes fonctionnalités chez IBM Microelectronics.

La présence de coeurs de processeurs soft (décrits dans un langage de type VHDL ou SystemC) donc paramétrables oblige donc aussi à générer l'ensemble de l'environnement de développement logiciel qui doit l'accompagner. En effet chaque variation du processeur peut nécessiter des outils logiciels de développement adaptés (exemple compilateur). Un exemple d'environnement générant le processeur ainsi que son environnement de développement logiciel est proposé par exemple par Tensilica .

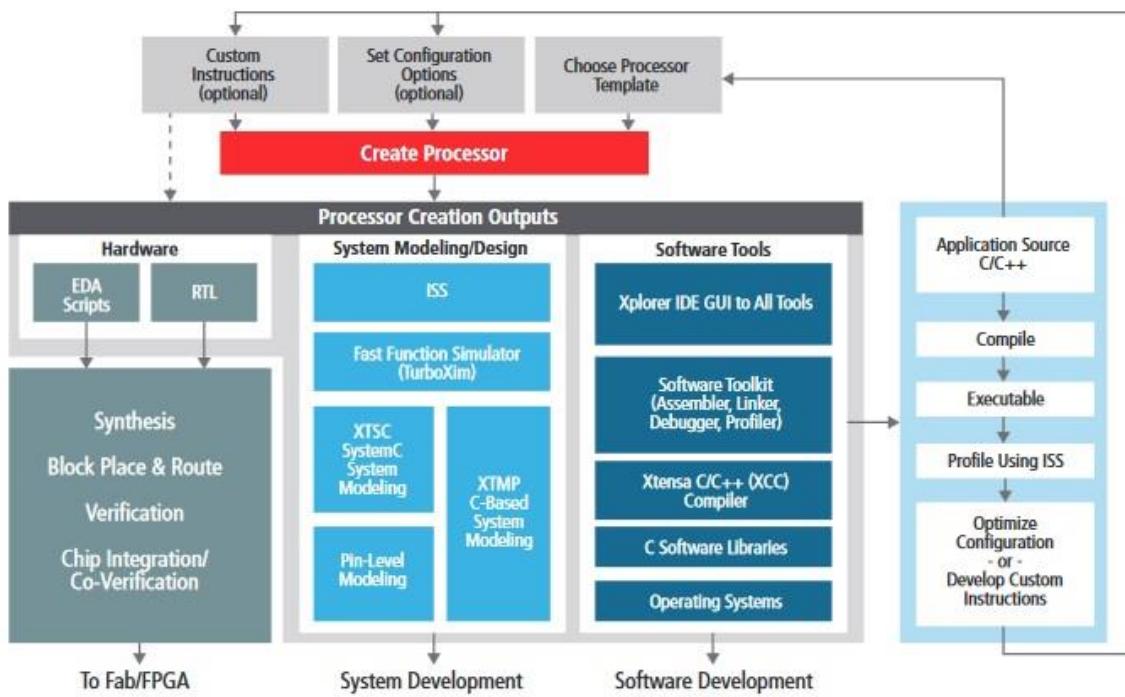
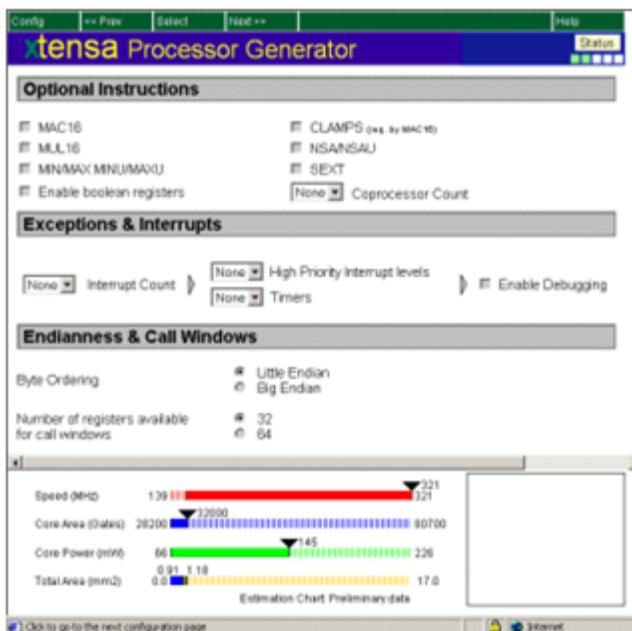


Figure 14.Flot de Génération de Processeur et Environnement Logiciel Associe (Tensilica/Cadence)



La principale mesure de performance d'un système à base de microprocesseurs est le temps d'exécution. Les unités de temps d'exécution sont les unités habituelles du temps.

Néanmoins étant donné que la vocation principale des systèmes à base de microprocesseurs est d'exécuter une ou des applications cette mesure n'est pas possible dans l'absolu et se fait toujours par rapport à une ou des applications. Ce point est fondamental. Il n'existe donc pas de système unique optimal en termes de performances. La conséquence directe est l'existence de familles de systèmes adaptées à chaque domaine d'application et le travail d'ingénierie principal consiste à définir et concevoir des systèmes qui soient les plus adaptées aux besoins des applications. La dimension économique reste pourtant incontournable et l'adaptation "totale" représentées par les systèmes totalement dédiés n'est pas économiquement viable du au coût extrêmement important du développement. La tendance étant plutôt dans la réutilisation de composants sur étagères et leurs adaptations.

Les Programmes d'Evaluation (Benchmarks)

Les programmes d'évaluation (Benchmarks) sont des programmes utilisés pour évaluer des systèmes à base de microprocesseurs. Comme on peut s'y attendre ces programmes d'évaluations sont issues et sont sensés représenter les domaines variés. Le tableau suivant en présente un certain nombre suivant les domaines.

Tableau 8– Exemples de Benchmarks

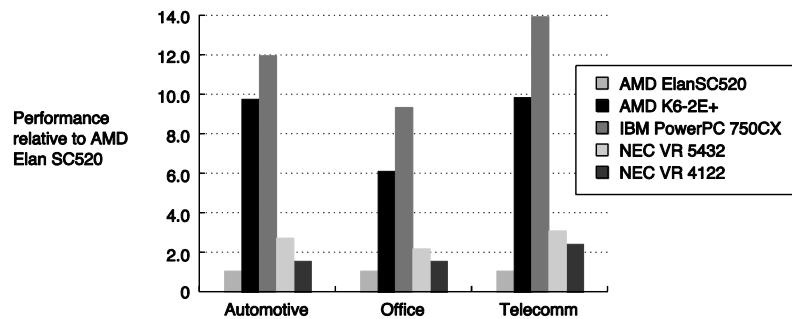
Benchmark	Description
CPU généraliste	SPEC CPU 2017
Systèmes embarqués	IoTMARK, ULPMARK
Serveurs de fichiers	SPECSFS 2014
Serveurs Web	SPECWeb
Serveurs de Transactions	TPC
Graphiques	SPECviewperf/SPECapc

Plus précisément pour les systèmes embarqués l'ensemble des benchmarks EEMBC est décrit comme suit:

Tableau 9.Benchmark pour processeurs Embarqués (EEMBC)

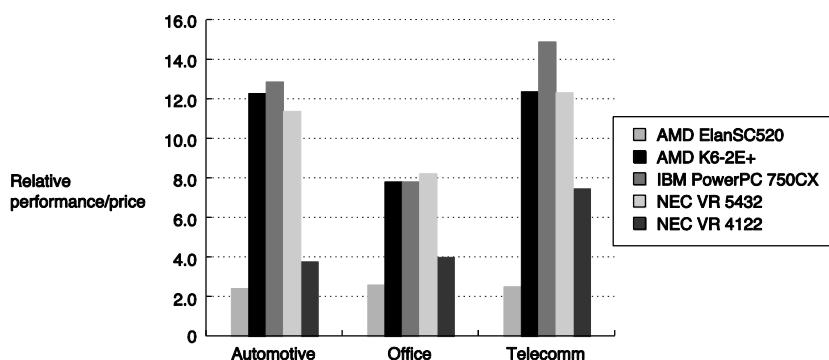
Domaine	nombre	Benchmarks
Automobile/industrie	16	6 microbenchmarks (opérations arithmétiques, multiplication de matrices,...), 5 benchmark de contrôle automobile et 5 benchmarks de filtrage ou FFT
Elect. Grand public (consumer)	5	5 benchmark multimedia (JPEG compress/decompress/filtrage/...)
Télécom	6	Filtrages/FFT/auto corrélation/décodage/codage/
Réseau	3	Plus court chemin, routage IP, opérations de flot de paquet
Bureautique	4	Benchmarks graphiques et textes

L'évaluation de systèmes embarqués sur les EEMBC recherche non seulement la performance mais aussi le rapport coût performance très important dans les applications portables et aussi la consommation d'énergie de ces applications. Les figures suivantes montrent respectivement la performance relative de 5 microprocesseurs pour 3 catégories de benchmarks (automotive, office, télécom), le rapport performance/prix et enfin le rapport performance/énergie.



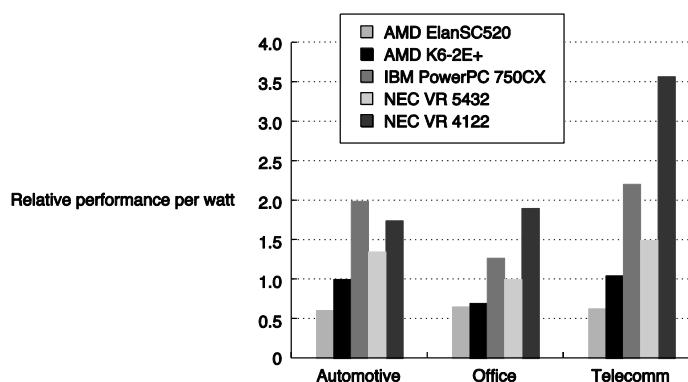
© 2003 Elsevier Science (USA). All rights reserved.

Figure 16. Performance Relative de 5 processeurs embarqués pour 3 des 5 benchmarks



© 2003 Elsevier Science (USA). All rights reserved.

Figure 17. Prix-performance pour 5 différents processeurs embarqués et 3 des 5 benchmarks EEMBC utilisant uniquement le prix du processeur



© 2003 Elsevier Science (USA). All rights reserved.

Figure 18. Performance Relative par Watt consommé pour 5 processeurs embarqués

Les systèmes généralistes CPU utilisent pour l'évaluation de performances les benchmarks SPEC comme principaux benchmarks.

Tableau 10.Benchmarks pour Systèmes Généralistes

Benchmarks	Commentaires
SYSmrk	www.bapco.com
SYSmrk DB	www.bapco.com
TPC	www.tpc.org

Les benchmarks les plus reconnus pour l'évaluation des microprocesseurs restent les benchmarks SPEC CPU 2000 qui incluent une partie en arithmétique entière SPEC CINT 2000 et une partie en arithmétique flottante SPEC CFP 2000. Le tableau suivant décrit les programmes utilisés pour l'évaluation de performances en arithmétique entière.

SPEC CINT2000 Benchmark
CINT2000 (Integer Component of SPEC CPU2000)

Tableau 11.– Benchmarks SPEC CINT2017

Benchmark	Langage	Category
600.perlbench_s	C	Perl interpreter
602.gcc_s	C	GNU C compiler
605.mcf_s	C	Route planning
620.omnetpp_s	C++	Discrete Event simulation - computer network
623.xalancbmk_s	C++	XML to HTML conversion via XSLT
625.x264_s	C	Video compression
631.deepsjeng_s	C++	Artificial Intelligence: alpha-beta tree search (Chess)
641.leela_s	C++	Artificial Intelligence: Monte Carlo tree search (Go)
657.xz_s	Fortran	Artificial Intelligence: recursive solution generator (Sudoku)
648.exchange2_s	C	General data compression

Ces benchmarks sont évalués et permettent d'établir des comparaisons entre ordinateurs. La procédure d'évaluation est la suivante:

Procédure d'évaluation SPEC CPU2000

Début

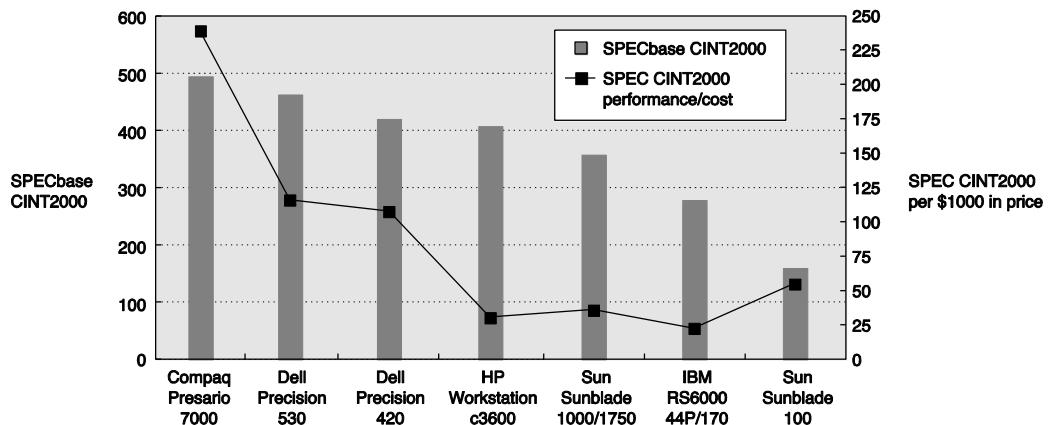
1. exécuter chaque benchmark un nombre impair de fois > à 3,
2. mesurer le temps de chaque exécution en secondes,
3. calculer le médian et calculer le rapport de ce temps avec le temps d'exécution pour ce même benchmark pour la station Sun Ultra 10 (rapport d'exécution i – execution time ratio i),
4. calculer la moyenne géométrique de ces rapports d'exécution pour les programmes entiers (SPECint_base2000),
5. calculer la moyenne géométrique de ces rapports d'exécution pour les programmes flottants (SPECfp_base2000).

Fin

moyenne géométrique :

$$\sqrt[n]{\prod_{i=1}^n \text{Execution time ratio i}}$$

La figure 20 décrit les résultats de l'évaluation de performances de 7 systèmes permettant ainsi non seulement de comparer leurs performances sur les benchmarks SPEC CPU entiers mais aussi leurs rapports performance/coût.



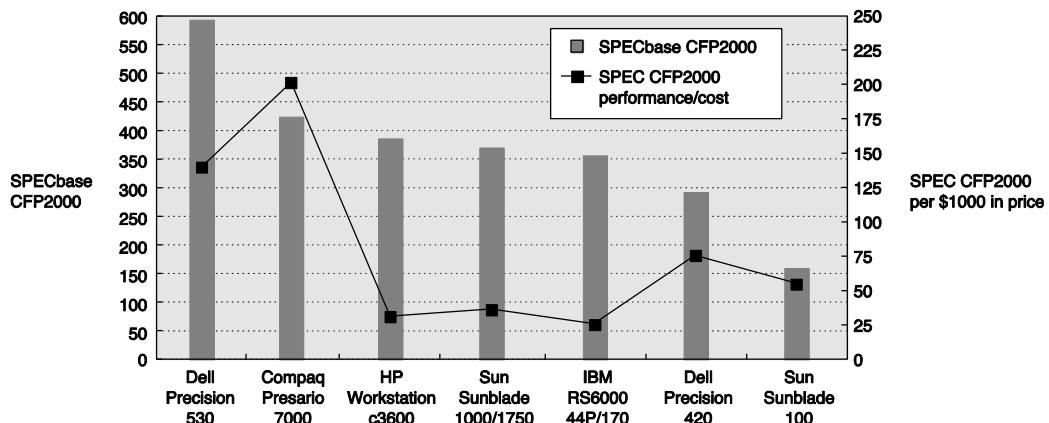
© 2003 Elsevier Science (USA). All rights reserved.

Figure 19 Performance de 7 ordinateurs différents pour les SPEC CINT 2000.

Les performances en arithmétique flottante peuvent être évaluées grâce au benchmark SPEC CFP 2000. La suite des applications constituants le benchmark est décrite dans le tableau suivant et inclut de nombreuses applications de type scientifique.

Benchmark	Language	Category
Tableau 12.Benchmarks SPEC CFP 2017		
503.bwaves_r	Fortran	Explosion modeling
507.cactuBSSN_r	Fortran	Physics: relativity
508.namd_r	C++	Molecular dynamics
510.parest_r	C++	Biomedical imaging: optical tomography with finite elements
511.povray_r	C++, C	Ray tracing
519.lbm_r	C	Fluid dynamics
521.wrf_r	Fortran, C	Weather forecasting
526.blender_r	C++, C	3D rendering and animation
527.cam4_r	Fortran	Atmosphere modeling
628.pop2_s	C	Wide-scale ocean modeling (climate level)
538.imagick_r	C	Image manipulation
544.nab_r	C	Molecular dynamics
549.fotonik3d_r	Fortran	Computational Electromagnetics
554.roms_r	Fortran	Regional ocean modeling

De même que pour les SPEC CINT 2017 on obtient des comparaisons pour les SPEC CFP 2017.



© 2003 Elsevier Science (USA). All rights reserved.

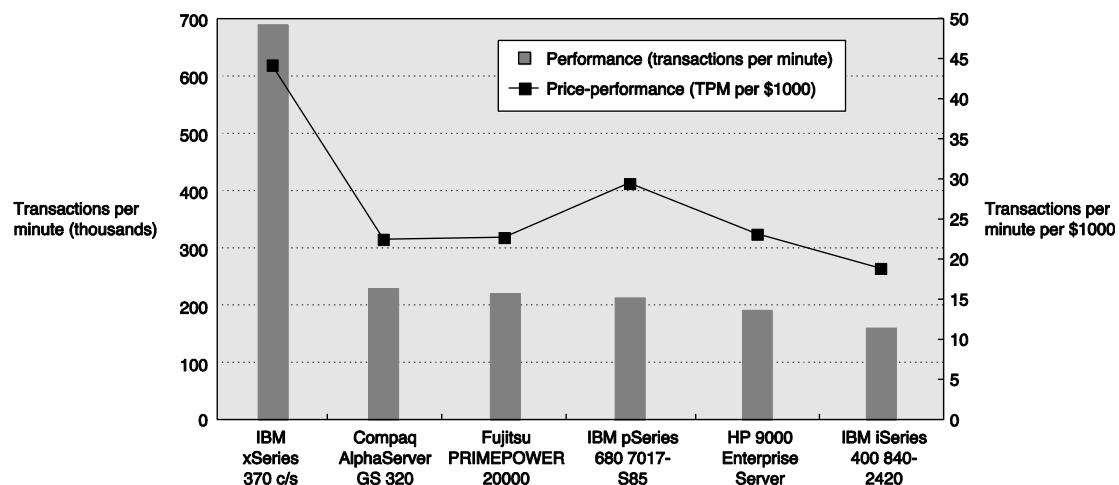
Figure 20. Performance de 7 ordinateurs différents pour les SPEC CFP 2000.

On remarquera qu'aucun des benchmarks SPEC CPU n'est écrit en Java. Les applications Java ont leur propres benchmarks avec SPEC qui sont les SPECjbb2000 (Java Business Benchmark) qui sont les premiers benchmarks pour l'évaluation de performances de la partie serveur de Java tandis que les benchmarks JVM98 évaluent les performances du côté client. L'évaluation de performances exige que le rapport de résultats spécifie clairement le contexte d'exécution de telle sorte à ce que ces résultats soient reproductibles. En ce sens SPEC exige les informations suivantes pour valider les résultats d'évaluation de performance à base des benchmarks SPEC CPU.

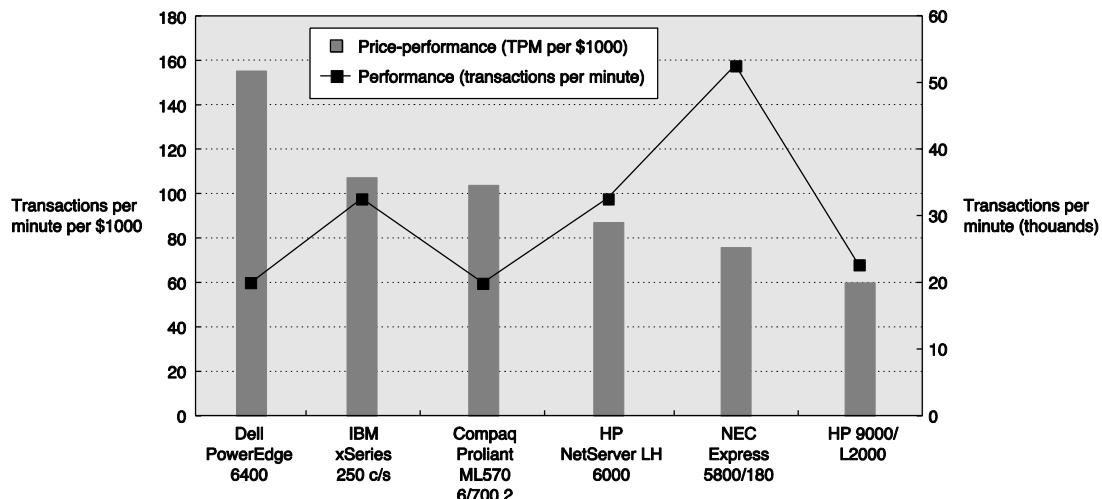
Tableau 13. Paramètres requis pour valider un résultat d'évaluation avec les benchmarks SPEC	
Hardware	HarDWARE
CPU (Processor Name)	CPU MHz
FPU	Number of CPUs in System.
Level 1 Cache (Size and Organization)	Level 2 Cache (Size and Organization)
Other Cache (Size and Organization)	Memory (Size in MB/GB)
Memory Configuration	Disk (Size (MB/GB), Type (SCSI, Fast SCSI etc.)
Other Hardware (Additional equipment added to improve performance	
	SOFTWARE
	Operating System (Name and Version)
	System State (e.g. Single User, Multi-user, Init 3, Default)
	File System Type
	Compiler (Name and Version)
	Pre-processors (Name and Version) if used
	Other Software (Additional software added to improve performance)

En effet les performances des microprocesseurs dépendent fortement du contexte et ceci est d'autant plus vrai avec les nouvelles générations de microprocesseurs. Des fiches types de rapport d'évaluations de deux processeurs commerciaux sont données en fin de chapitre.

Le cas des systèmes de transactions (OLTP – Online Transactions Processing) de types bancaires ou réservation d'avions est traité par les benchmarks TPC.



© 2003 Elsevier Science (USA). All rights reserved.
Figure 21. Performance sur TPC-C (www.tpc.org)



© 2003 Elsevier Science (USA). All rights reserved.
Figure 22. Cout-Performance TPC -C (www.tpc.org)

Les benchmarks SPEC CPU présentent toutefois un problème important pour les architectes concepteurs de microprocesseurs qui utilisent la simulation comme technique d'évaluation de nouvelles machines : les temps de simulation sont très importants et peuvent représenter plusieurs mois de temps machine sans interruption. Le problème de trouver des benchmarks "équivalents" mais nécessitant un temps de simulation moindre reste un problème ouvert de recherche. Les techniques de réduction proposées jusqu'à présent consistent à : (1) échantillonner un benchmark (2) à effectuer une combinaison de simulation statistique et fonctionnelle. Malheureusement aucune technique ne permet de refléter le comportement d'un benchmark de manière précise.

Techniques de mesure

La mesure de performances d'un microprocesseur doit être aussi précise qu'elle doit ne pas perturber la mesure.

Les méthodes de mesure peuvent s'appuyer sur des outils matériels ou logiciels ou exploiter le support matériel existant dans les microprocesseurs récents pour effectuer des mesures de performances.

Les techniques matérielles pures détournent des instruments initialement utilisés pour le diagnostic et le test comme les analyseurs logiques à des fins d'évaluation de performance. Un analyseur logique est un instrument de mesure multi-entrées avec capture de signal et circuits intelligents de détection de séquence permettant de développer, débugger et maintenir des systèmes digitaux. Un ensemble d'accessoires comme des sondes de bus ou de processeurs permettent en connexion directe sur le composant semi-conducteur d'obtenir des traces d'exécution et donc de pouvoir analyser les performances.

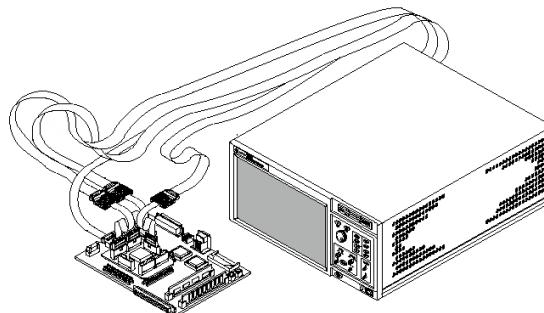


Figure 23. Analyseur Logique (Agilent www.agilent.com)

La technique de mesure reste limitée du fait de son principe : (1) seuls les signaux externes du composant sont accessibles et par conséquent seule l'information associée est disponible (même si des analyses de corrélation permettent de dériver certaines conclusions sur les sous-ensembles matériels n'ayant pas de communication avec le monde extérieur par des broches mais participant au processus de génération des signaux en amont) (2) les analyseurs logiques restent limités dans leur capacité de mémorisation et restent donc incapables de mémoriser des traces d'exécution de benchmarks du type SPEC CPU 2017.

Les techniques logicielles incluent bien évidemment la simulation pour les processeurs en cours de développement et les moniteurs logiciels pour les microprocesseurs existants. La simulation reste l'outil indiscuté pour la recherche et la conception de nouveaux processeurs mais reste une technique très coûteuse et lente. Finalement pour les microprocesseurs existants les constructeurs ont depuis de nombreuses années ajouté sur le silicium des registres spécialisés pour compter des événements spécifiques durant l'exécution d'un programme.

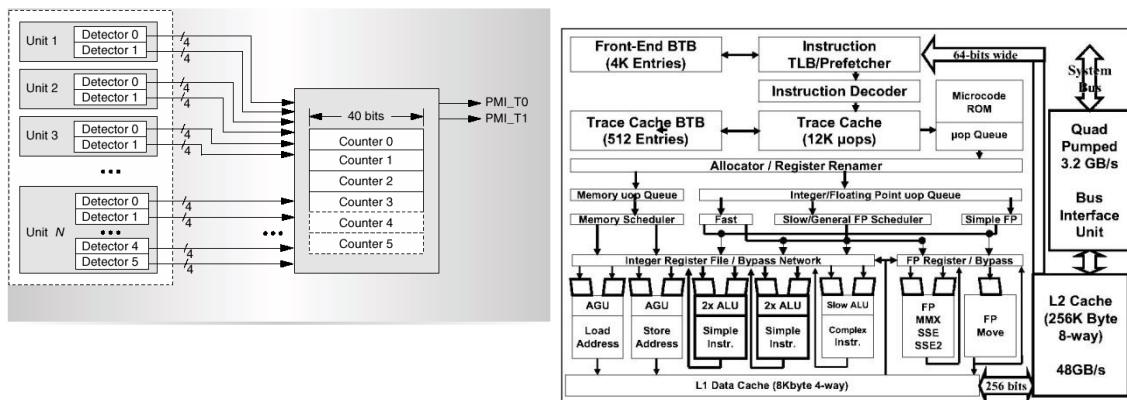


Figure 24. Mécanisme des compteurs de performance au sein des microprocesseurs (1) mécanisme au sein du P4 (2) microarchitecture du P4

Les événements pouvant entre autres être mesurés : nombre de cycles d'exécution, nombre de cycles par instruction (CPI), nombre de défauts de caches (instruction, données), distribution des instructions par unité fonctionnelle, etc.... Des programmes d'analyse et de mesures de performance comme ceux mentionnés dans le tableau suivant sont disponibles chez les constructeurs.

Tableau 14.Exemples de programme d'analyse de performance

CONSTRUCTEUR	PROGRAMME
AMD	AMD Code Analyst
INTEL	VTune Performance Analyzer

Les programmes précédents sont à utiliser avec une bonne connaissance de la microarchitecture associée pour mieux analyser les très probables corrélations entre les différents paramètres affectant la performance.

Modèle Analytique

De manière générale les modèles de performance analytiques ne sont pas utilisés pour l'évaluation de performance des microprocesseurs.

Néanmoins certaines équations analytiques permettent de faire des calculs simples de performance.

Loi d' Amdhal

La loi d'Amdhal définit le speedup comme le rapport des temps d'exécution pour une application sur une configuration donnée (old) et une proposition de configuration améliore. (new).

$$\text{Speedup overall} = \frac{\text{Execution time old}}{\text{Execution time new}}$$

Cette équation peut se réécrire en:

$$\text{Speedup overall} = \frac{1}{(1 - \text{Fraction enhanced}) + \frac{\text{Fraction enhanced}}{\text{Speedup enhanced}}}$$

ou Fraction enhanced est la fraction du temps de calcul dans la configuration donnée qui peut être améliorée grâce à l'amélioration proposée
et Speedup enhanced est l'amélioration obtenue dans la nouvelle configuration.

Exemple: Supposons que l'on envisage un accroissement de performance pour un processeur d'un serveur. La nouvelle CPU est 10 fois plus rapide en calcul pour l'application Web. Supposons que la CPU initiale est occupée 40% du temps en attente d'entrée/sortie 60% du temps.

Quel est le speedup global si la nouvelle CPU est installée ?

$$\text{Fraction enhanced} = 0.4$$

$$\text{Speedup enhanced} = 10$$

$$\text{Speedup}_{\text{overall}} = \frac{1}{(1 - 0.4) + \frac{0.4}{10}} = \frac{1}{0.64} = 1.56$$

Une autre équation relative à l'évaluation de performances CPU (processeur) consiste à évaluer le temps d'exécution CPU (CPU time) comme le produit du nombre de cycles horloges nécessaires à l'exécution d'un programme (un microprocesseur est une machine synchrone) et de la période d'horloge.

Equation de Performance CPU

$$\text{CPU time} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

Où

$$\text{CPU time} = \text{CPU clock cycles for a program} / \text{clock rate}$$

$$\text{CPU time} = \frac{\text{CPU clock cycle for a program}}{\text{clock rate}}$$

Une mesure additionnelle est le nombre moyen de cycles par instruction (**CPI** - cycles per instruction). Pour cela il est nécessaire de connaître le nombre d'instructions assemblées (IC – Instruction Count) par un programme donné.

Le calcul du CPI se fait de la manière suivante:

$$\text{CPI} = \frac{\text{CPU clock cycles for a program}}{\text{Instruction count}}$$

Cette mesure est utilisée comme critère de mérite sur les différents types de jeu d'instructions et leurs implémentations et résume en recherche en architecture l'efficacité d'une implémentation.

On peut aussi écrire que:

$$\text{CPU time} = \text{Instruction count} \times \text{Clock cycle time} \times \text{Cycles per instruction}$$

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{Clock cycle time}}{\text{Clock rate}}$$

Où

$$\text{CPU time} = \frac{\text{Instructions}}{\text{program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{clock cycle}}$$

Le premier paramètre définit le nombre d'instructions dans un programme et est strictement dépendant de l'architecture du jeu d'instruction et de la technologie du compilateur.

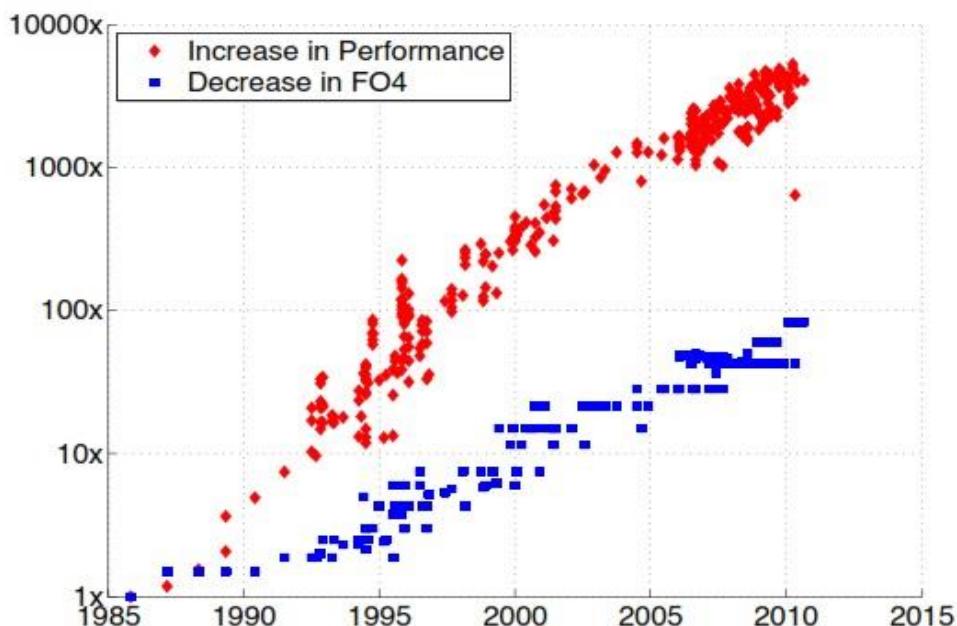
Le deuxième paramètre définit le CPI qui définit la microarchitecture du microprocesseur et l'architecture du jeu d'instruction et enfin la période de l'horloge qui elle définit la technologie semiconducteur et la microarchitecture.

On peut conclure des ces équations que la fréquence d'horloge d'un microprocesseur ne peut en aucun cas constituer à elle seule une mesure de performance d'un microprocesseur.

Le jeu d'instructions et la microarchitecture du microprocesseur objets des chapitres suivants jouent un rôle central.

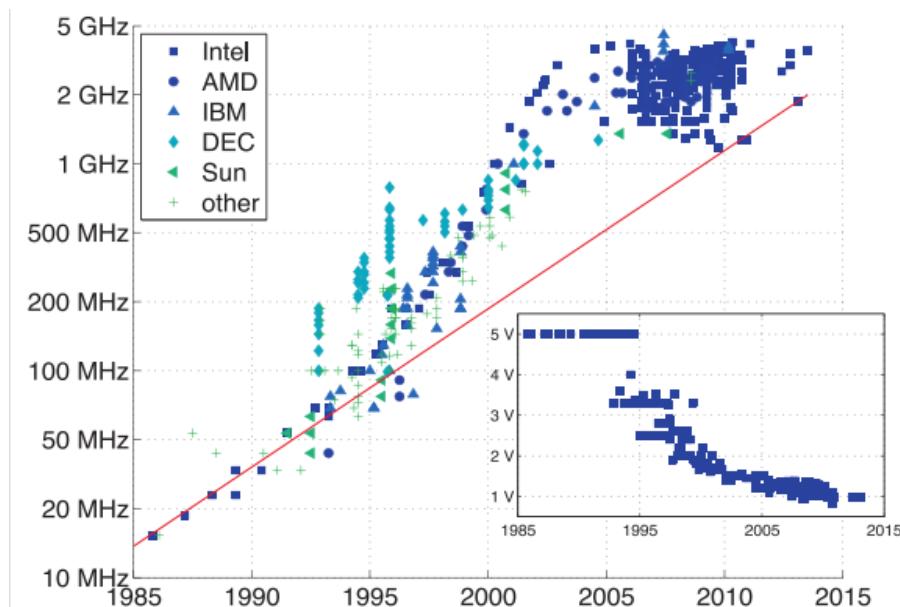
1.5 Conclusion

Les microprocesseurs ont connu une évolution spectaculaire de leurs performances dans les 30 dernières années.



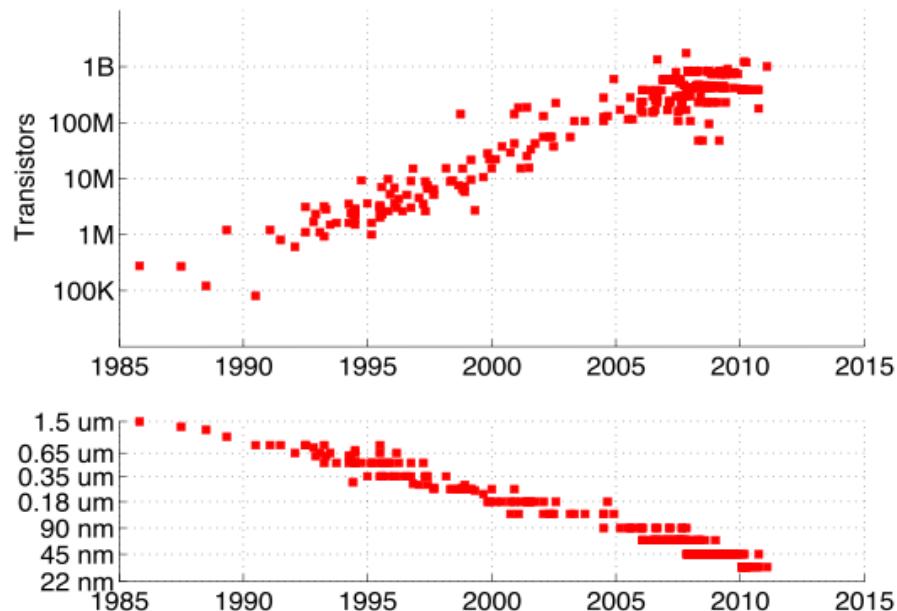
Improvement in microprocessor and gate performance vs. year.

Cette performance a été soutenue par les progrès de la physique du semiconducteur qui a permis des fréquences de fonctionnement et des surfaces de silicium en évolution permanente.



Clock frequency vs. year. The red line indicates frequency increase due to gate speed. The insert plot is Vdd vs. year.

Les progrès ont ainsi permis une augmentation spectaculaire du nombre de transistors et actuellement plusieurs milliards de transistors sont disponibles pour les concepteurs de microprocesseurs



Number of transistors and feature size vs. year.
ouvrant la voie à l'implémentation de multiprocesseurs complexes.

Les architectures de microprocesseurs ont su exploiter cette ressource brute de la physique en proposant régulièrement de nouveaux mécanismes permettant de pousser davantage les performances.

La montée en puissance régulière et importante de la généralisation de l'internet comme support aux activités économiques et l'émergence de L'Internet des objets (IoT pour *Internet of Things*) représentant l'extension d'Internet à des choses et à des lieux du monde physique renforcera les besoins en puissance des microprocesseurs et de nouveaux défis comme le Big Data apparaissent .

En 2018 les microprocesseurs sont omniprésents et amenés à accroître leurs présences dans de nombreuses nouvelles applications.

2 Chapitre 2 Structure d'un Microprocesseur

2.1 Introduction

Un microprocesseur a pour fonctionnalité principale l'exécution de programmes informatiques écrits dans des langages variés comme Java, C++, Ada ou Fortran mais qui pour être exécutés doivent être au préalable compilés vers le seul langage compris par un microprocesseur : son propre jeu d'instructions. Le processus de compilation a pour fonctionnalité principale la transformation d'un texte (fichier) représentant un programme écrit dans un langage de haut niveau en un texte (fichier) représentant le même programme écrit dans le langage machine (jeu d'instructions) du microprocesseur sur lequel doit s'exécuter ce même programme. La définition d'un jeu d'instructions entraîne obligatoirement le développement d'un ensemble d'outils de développement logiciels pour permettre l'exploitation maximale des ressources du microprocesseur et l'optimisation des programmes : compilateurs optimiseurs, débuggeur, outil d'analyse de performances, simulateurs. Cet effort représente un investissement financier et humain très important qui combiné au fait que les utilisateurs d'ordinateurs souhaitent pouvoir réutiliser leurs logiciels en passant d'une génération à une autre de microprocesseurs fait que les modifications de jeu d'instructions sont des décisions stratégiques majeures devant être profondément justifiées à tout point de vue. Dans tous les cas la compatibilité ascendante doit être scrupuleusement respectée.

Tableau 15.Exemples de Microprocesseurs Commerciaux

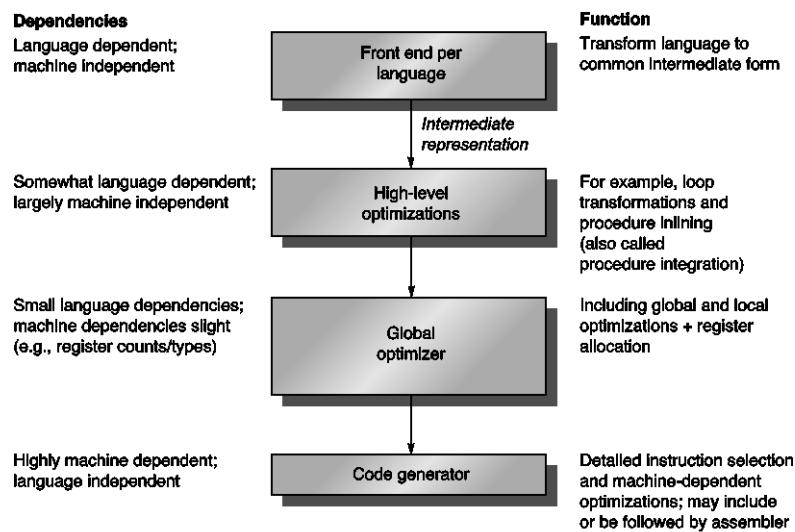
Constructeur	Microprocesseur
AMD	Ryzen, Athlon, Opteron, Embedded Series G
ARM	Cortex-A7X, Cortex-A5X, Cortex-M
Compaq	Alpha 21164, 21264, 21364
ST	STM32F2, SPC56, STM8AF
Intel	Core i7, Core i5, PIII, P4, Itanium, Nehalem
MIPS	M-Class M51xx, I-Class I6500-F
NXP	Kinetis, i.MX, P-Series
Oracle	UltraSparc II, III

Face aux intérêts majeurs que représente la compatibilité un axe de recherche récent s'est développé qui a pour objectif de pouvoir exécuter un programme binaire pour un microprocesseur de type X sur un processeur de type Y cela de manière totalement transparente pour l'utilisateur. Cet objectif de traduction binaire dynamique (*binary translation*) a bien entendu une contrainte technologique forte : effectuer la traduction et l'exécution du programme en un temps compétitif avec celui obtenu par l'exécution du programme sur le microprocesseur cible. Cet objectif n'a pas pu être atteint pour les processeurs à hautes performances mais a trouvé une niche dans les processeurs embarqués à travers l'exemple commercial Transmeta Crusoe (www.transmeta.com). Il n'en reste pas moins que la traduction binaire dynamique reste un défi technologique majeur.

2.2 Jeu d'instructions et éléments de structure

Définition jeu d'instructions : le jeu d'instructions d'un microprocesseur est un ensemble fini de N instructions opérant sur un ensemble M fini de types de données et permettant l'exécution de programmes.

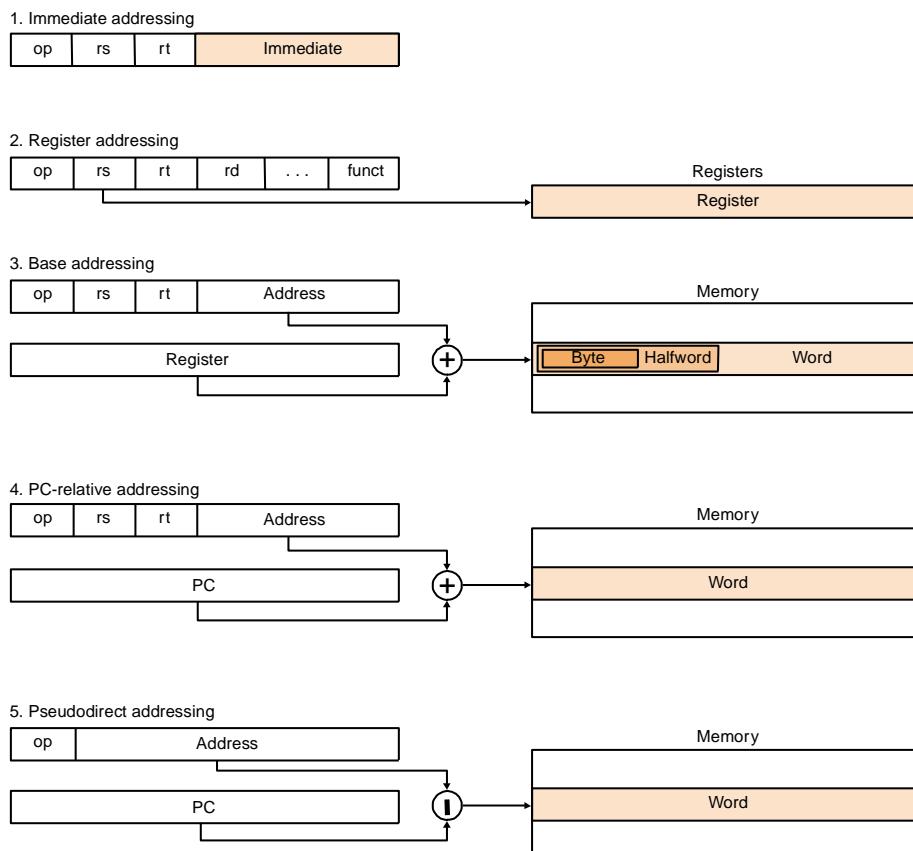
Le jeu d'instructions d'un microprocesseur est le seul langage que le microprocesseur puisse interpréter. Le rôle d'un compilateur est de transformer des programmes écrits dans un langage de haut niveau (exemple : C, C++, Fortran) en des programmes écrits dans le jeu d'instructions du microprocesseur cible. La version binaire obtenue en transformant les instructions du microprocesseur par leurs codages binaires constitue le binaire exécutable. Les jeux d'instructions des processeurs du commerce ont un nombre distinct d'instructions mais on retrouve les instructions les plus courantes dans chaque jeu. Il en est de même pour les types de données.



© 2003 Elsevier Science (USA). All rights reserved.

Figure 25.Flot de Compilation.

Définition Mode d'Adressage : un mode d'adressage est une procédure de spécification de l'adresse de l'objet en mémoire auquel le microprocesseur souhaite accéder au cours de l'exécution.

**Figure 26.Modes d'adresses du processeur MIPS**

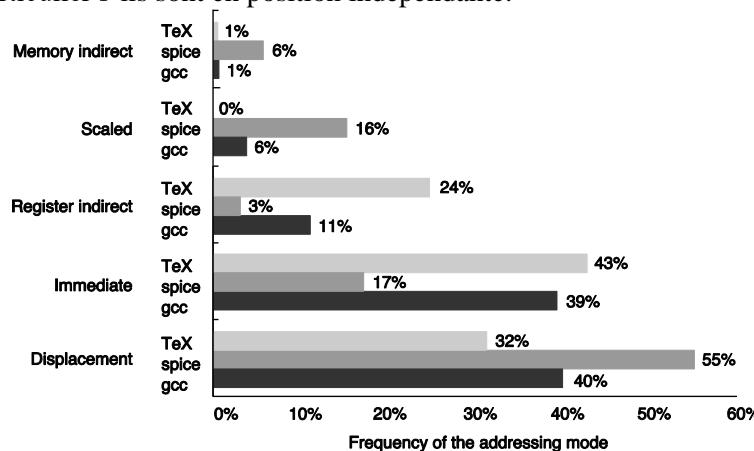
Il n'existe aucune contrainte spécifique sur les modes d'adressage et les modes d'adressage ne sont ni "normalisés" ni "standardisés". De ce fait certains microprocesseurs peuvent proposer des modes

d'adressages que d'autres microprocesseurs ne proposeront pas mais les modes d'adresses les plus simples se retrouvent dans tous les microprocesseurs.

La méthodologie de sélection (d'inclusion) de modes d'adresses effectuée par les architectes de microprocesseurs est dominée par : (1) l'utilisation que les compilateurs en font (2) le gain en performance (3) le coût de leurs implémentations.

La conception de compilateurs se faisant souvent de manière concurrente avec la conception du microprocesseur, des mesures statistiques sont effectuées de manière itérées sur l'utilisation des modes d'adressage et déterminent au final leurs sélections. Cela n'oblige bien entendu pas les concepteurs de compilateurs indépendants ("third party") de les utiliser. Les analyses de code générée par les compilateurs montrent malheureusement de manière quasi-invariable que c'est toujours la voie de la simplicité qui est privilégiée au détriment fut il de la performance.

Autant un concepteur de microprocesseur aura intérêt à développer un compilateur optimiseur permettant de démontrer l'utilité ou le gain de performance apporté par une nouveauté microarchitecturale autant les concepteurs indépendants seront regardants par rapport au gain que cela leur rapporte en particulier s'ils sont en position indépendante.



© 2003 Elsevier Science (USA). All rights reserved.

Figure 27. Statistiques d'utilisation des modes d'adresses pour 3 programmes

Néanmoins les trois points principaux se retrouvent dans les modes d'adresses :

1. les modes d'adresses suivants : déplacement, immédiat et registre indirect,
2. la taille d'un déplacement de 12-16 bits
3. la taille des valeurs immédiates 8-16 bits

Le jeu d'instruction doit permettre d'effectuer un ensemble d'opérations : (1) opérations de calcul (2) opérations logiques (3) opérations de lecture écriture de données (4) opérations de contrôle du flux des opérations. Les opérations de calcul incluent les opérations arithmétiques de base (+, -, *, /) sur des entiers naturels (non signés) ou relatifs (signés). Les opérations logiques sont des opérations de ET, OU, OU Exclusif, négation et de décalage logique et arithmétique.

L'ensemble des opérandes de ces instructions se trouve dans des registres regroupés dans une structure située dans le microprocesseur, le banc de registres. Le banc de registres du processeur MIPS compte 32 registres de taille 32 bits numérotés de 0 à 31. Le registre 0 conserve la valeur 0. Les autres registres peuvent être utilisés pour conserver des données. Les opérations de lecture et écriture de données en mémoire sont les opérations permettant dans le cas de la lecture de lire une donnée en mémoire et la placer dans un registre sur le microprocesseur. À l'inverse l'opération d'écriture consiste à écrire une donnée existante dans un registre sur le microprocesseur en mémoire. Les microprocesseurs dont les

seules instructions pouvant référencer une donnée en mémoire sont les instructions de lecture et écriture sont appelées architectures load-store.

Tableau 16.Jeu d'instructions du MIPS

Category	Instruction	Example	Meaning	Comment
Arithmetic	add	add \$s1,\$s2,\$s3	\$s1= \$s2+\$s3	Three operands ;overflow detected
	subtract	sub \$s1,\$s2,\$s3	\$s1= \$s2-\$s3	Three operands ;overflow detected
	add immediate	addi \$s1,\$s2,100	\$s1= \$s2+100	+constant; overflow undetected
	subtract unsigned	subu \$s1,\$s2,\$s3	\$s1= \$s2-\$s3	Three operands ;overflow undetected
	add immediate unsigned	addiu \$s1,\$s2,100	\$s1= \$s2+\$s3	+constant; overflow undetected
	move from coprocessor register	mfc0 \$s1,\$epc	\$s1=\$epc	Used to copy exception PC plus other special registers
	multiply	mult \$s2,\$s3	Hi, Lo=\$s2x \$s3	64-bits signed product in Hi, Lo
	multiply unsigned	multu \$s2,\$s3	Hi, Lo=\$s2x \$s3	64-bits unsigned product in Hi, Lo
	divide	div \$s2,\$s3	Lo=\$s2 / \$s3 Hi=\$s2 mod \$s3	Lo = quotient, Hi = remainder
	divide unsigned	divu \$s2,\$s3	Lo=\$s2 / \$s3 Hi=\$s2 mod \$s3	Unsigned quotient and remainder
	move from Hi	mfhi \$s1	\$s1=Hi	Used to get copy of Hi
	move from Lo	mflo \$s1	\$s1=Lo	Used to get copy of Lo
Logical	and	and \$s1,\$s2,\$s3	\$s1= \$s2 & \$s3	Three register. operands ; logical AND
	or	or \$s1,\$s2,\$s3	\$s1= \$s2 \$s3	Three register. operands ; logical OR
	and immediate	andi \$s1,\$s2,100	\$s1= \$s2 & 100	Logical AND reg, constant
	or immediate	ori \$s1,\$s2,100	\$s1= \$s2 100	Logical OR reg, constant
	shift left logical	sll \$s1,\$s2,10	\$s1= \$s2 << 10	Shift left by constant
	shift right logical	srl \$s1,\$s2,10	\$s1= \$s2 >> 10	Shift right by constant
Data transfer	load word	lw \$s1,100(\$s2)	\$s1=Memory[\$s2+100]	Word from memory to register
	store word	sw \$s1,100(\$s2)	Memory[\$s2+100] =\$s1	Word from register to memory
	load byte unsigned	lbu \$s1,100(\$s2)	\$s1=Memory[\$s2+100]	byte from memory to register
	store byte	sb \$s1,100(\$s2)	Memory[\$s2+100] =\$s1	byte from register to memory
	load upper immediate	lui \$s1,100	\$s1=100*2 ¹⁶	Load constant in upper 16 bits
Conditional branch	branch on equal	beq \$s1,\$s2,25	If (\$s1==\$s2) go to PC+4+100	Equal test; PC-relative branch
	branch on not equal	bne \$s1,\$s2,25	If (\$s1 !=\$s2) go to PC+4+100	Not Equal test; PC-relative
	set on less than	slt \$s1,\$s2,\$s3	If (\$s2<\$s3) \$s1=1 ; else \$s1=0	Compare less than; two's complement
	set less than immediate	slti \$s1,\$s2,100	If (\$s2<100) \$s1=1 ; else \$s1=0	Compare < constant; two's complement
	set less than unsigned	sltu \$s1,\$s2,\$s3	If (\$s2<\$s3) \$s1=1 ; else \$s1=0	Compare less than; natural numbers
	set less than immediate unsigned	sltiu \$s1,\$s2,100	If (\$s2<100) \$s1=1 ; else \$s1=0	Compare < constant; t natural numbers
Unconditional jump	jump	j 2500	Go to 10000	Jump to target address
	jump register	jr \$ra	Go to \$ra	For switch, procedure return
	jump and link	jal 2500	\$ra = PC+4; go to 10000	For procedure all

Le jeu d'instruction du MIPS qui est le processeur que nous étudions suit un modèle d'architecture load-store.

Les instructions de contrôle sont les instructions permettant de modifier le flot d'exécution des instructions. Ce sont les instructions qui implémentent les opérations de contrôle de flot dans les langages de haut niveau. Ces opérations de contrôle sont if-else, switch(), les boucles (for(), while()) et appels de fonction.

Les 2 premières catégories sont implémentées par les instructions dites de branchements conditionnels (*conditional branch*) tandis que les appels de fonctions sont implémentés par les instructions de branchements inconditionnels (*unconditional jump*).

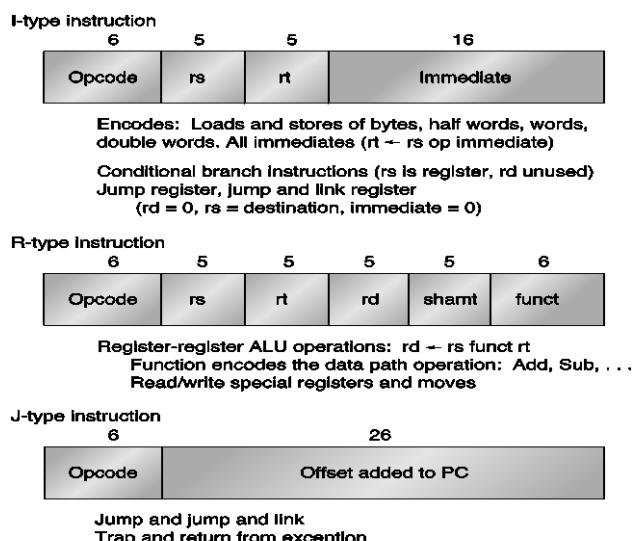
2.3 Définition codage d'un jeu d'instructions:

Le codage du jeu d'instructions définit le codage binaire de l'ensemble des combinaisons possibles de modes d'adressage avec les instructions.

Ce codage binaire est bien entendu indispensable puisque les seules valeurs reconnues sont les valeurs binaires. Ce codage affectera les éléments suivants :

1. implémentation du processeur : partie décodage d'instructions
2. la taille du code compilé (l'exécutable)

Au cours de l'exécution les instructions seront décodées par l'unité de contrôle pour générer les signaux de contrôle associés. Le décodage est implanté en matériel en utilisant des décodeurs qui occupent une surface de silicium et consomme de l'énergie. Plus un jeu d'instructions comportera d'instructions et de combinaisons particulières plus le décodage sera compliqué et coûteux en surface silicium, en temps d'exécution et en consommation d'énergie. Les principes qui doivent guider la conception d'un jeu d'instructions sont la simplicité et la régularité. La taille du code compilé affecte de manière accessoire l'espace disque dans le cas de PC mais de manière fondamentale dans le cas des systèmes embarqués ou l'espace mémoire est précieux. La taille du code compilé affecte aussi les performances de la hiérarchie mémoire car plus un code est large plus il est difficile d'optimiser son placement en mémoire. Le jeu d'instruction du processeur MIPS que nous étudions est codé sur 3 formats distincts d'instruction : (1) I-type (2) R-type (3) J-type. Tous les trois formats sont codés sur 32 bits et dédient les bits 31-26 à l'opcode qui est le code de l'instruction et qui permet d'identifier l'instruction. La figure suivante décrit les trois codages.



© 2003 Elsevier Science (USA). All rights reserved.

Figure 28. Formats de codage des Instructions MIPS

Les champs rs, rt présents dans le R-type et I-type et le champ rd présent dans le R-type permettent de spécifier 1 registre du banc de registre. Ces champs sont codés sur 5 bits et permettent donc d'adresser n'importe lequel des 32 registres du banc de registres. Le registre spécifié par le champ rs est systématiquement un registre utilisé comme opérande source pour les deux formats R et I tandis que le registre spécifié par rd est un registre destination c'est-à-dire amené à recevoir un résultat d'instruction. Ce rôle est joué par le registre rt dans le format I tandis que ce registre est la seconde opérande dans le format R. Le champ *immediate* du format I-type permet d'y placer une valeur constante. Le champ shamt (shift amount) est utilisé uniquement dans les instructions de décalages pour spécifier le nombre de décalages à effectuer. Finalement le champ funct (*function code*) permet de distinguer une variante particulière de l'instruction définie par le code opcode.

Tableau 17. Codage des instructions assebleur MIPS.

Name	Format	Example						comments
		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	
add	R	0	2	3	1	0	32	add \$s1,\$s2,\$s3
sub	R	0	2	3	1	0	34	sub \$s1,\$s2,\$s3
addi	I	8	2	1		100		addi \$s1,\$s2,100
addu	R	0	2	3	1	0	33	addu \$s1,\$s2,100
subu	R	0	2	3	1	0	35	subu \$s1,\$s2,\$s3
addiu	I	9	2	1		100		addiu \$s1,\$s2,100
mfc0	R	16	0	1	14	0	0	mfc0 \$s1,\$epc
mult	R	0	2	3	0	0	24	mult \$s2,\$s3
multu	R	0	2	3	0	0	25	multu \$s2,\$s3
div	R	0	2	3	0	0	26	div \$s2,\$s3
divu	R	0	2	3	0	0	27	divu \$s2,\$s3
mfhi	R	0	0	0	1	0	16	mfhi \$s1
mflo	R	0	0	0	1	0	18	mflo \$s1
and	R	0	2	3	1	0	36	and \$s1,\$s2,\$s3
or	R	0	2	3	1	0	37	or \$s1,\$s2,\$s3
andi	I	12	2	1		100		andi \$s1,\$s2,100
ori	I	13	2	1		100		ori \$s1,\$s2,100
sll	R	0	0	2	1	10	0	sll \$s1,\$s2,10
srl	R	0	0	2	1	10	2	srl \$s1,\$s2,10
lw	I	35	2	1		100		lw \$s1,100(\$s2)
sw	I	43	2	1		100		sw \$s1,100(\$s2)
lui	I	15	0	1		100		lui \$s1,100
beq	I	4	1	2		25		beq \$s1,\$s2,25
bne	I	5	1	2		25		bne \$s1,\$s2,25
slt	R	0	2	3	1	0	42	slt \$s1,\$s2,\$s3
slti	I	10	2	1		100		slti \$s1,\$s2,100
sltu	R	0	2	3	1	0	43	sltu \$s1,\$s2,\$s3
sltiu	I	11	2	1		100		sltiu \$s1,\$s2,\$s3
j	J	2			2500			j 10000
jr	R	0	31	0	0	0	8	jr \$31
jal	J	3			2500			jal 10000

2.4 Automates d'états finis et systèmes synchrones

L'implémentation d'un microprocesseur peut se faire dans des modèles d'exécution théoriques distincts et dans des technologies distinctes. Architecture, implémentation et technologie sont des notions

orthogonales. Nous nous restreignons dans ce cours aux implémentations à base de machine synchrone qui est l'implémentation dans la quasi-totalité des microprocesseurs commerciaux. La base théorique de la conception d'un microprocesseur se trouve dans la théorie des automates et des machines synchrones. Les systèmes séquentiels contiennent des états mémorisés dans des éléments mémoires et sont représentés par des machines d'états finis. Ces machines d'états finis acceptent des signaux d'entrée et génèrent des signaux de sortie. Une machine d'états finis contient un ensemble d'états et utilisent deux fonctions l'une permettant de passer à l'état suivant et l'autre de générer les valeurs des signaux de sortie. Une machine d'états dont la fonction de sortie ne dépend que de l'état courant est une machine de Moore. La machine de Moore sera utilisée pour implémenter entre autres la machine d'états du microprocesseur dans sa version multicycle.

La définition d'un système synchrone ne présuppose aucune valeur de fréquence d'horloge. La définition de la fréquence d'horloge dépend : (1) de la technologie semi-conducteur utilisée (2) des techniques de conception logiques utilisées (3) de la microarchitecture implémentant le système synchrone. Tous ces paramètres étant liés mais non complètement permet de grandes variations dans la configuration finale d'un microprocesseur pour un jeu d'instructions donné.

2.5 Structure cycle unique

Définition : une structure de microprocesseur en cycle unique implique que toutes les instructions s'exécutent en un seul cycle d'horloge.

L'automate d'états finis correspondant possède en fait un seul état qui correspond en fait à la lecture, le décodage l'exécution ainsi que la mémorisation des résultats obtenus en mémoire centrale. L'exécution d'une instruction se faisant à travers un chemin de donnée du microprocesseur et ce chemin étant différent pour chaque instruction la durée du cycle d'horloge sera dominée par le chemin le plus long. Ce chemin qui est donc le Max(T_i) $i = 1..N$ où T_i représente la durée d'exécution d'un chemin est appelé le chemin critique et définit la fréquence maximale de l'horloge.

2.5.1 Construction du chemin de données

Le chemin de données représente l'ensemble des chemins pouvant être pris par les données au cours de l'exécution des instructions présentes dans le jeu d'instruction. Par conséquent une méthodologie de conception de chemin de donnée consiste à analyser le jeu d'instructions et identifier les composants structurants nécessaires à l'exécution pour exécuter chaque d'instruction.

La définition d'un jeu d'instructions implique des éléments structurants de base du microprocesseur permettant l'exécution des différentes instructions.

Tableau 18.Elements Structurants de base

Element structurants	Utilité
Additionneur binaire à 2 entrées de N bits	Addition de deux données codées sur N bits
Multiplexeur n entrées 1 sortie	Sélection d'une entrée parmi n
Registre	Mémorisation d'une donnée
Mémoires	Mémorisation de données et code

Les éléments structurants de base sont les éléments principaux à partir desquels nous allons construire le chemin de donnée du microprocesseur.

Les microprocesseurs que nous étudions suivent le modèle d'exécution de Von Neumann dans lequel le programme à exécuter se trouve dans une mémoire. Il est donc nécessaire d'avoir une mémoire pouvant mémoriser les instructions du programme. Les paramètres principaux d'un composant mémoire virtuel

sont : (1) la taille du bus de donnée (2) la taille de cette mémoire et par conséquent la taille de son bus d'adresse (3) la possibilité d'effectuer des opérations de lecture et/ou d'écriture.

Nous allons considérer dans un premier temps que la taille du bus de donnée est égale à la taille d'une instruction (32 bits) et que la taille de la mémoire est indéterminée ce qui signifie que l'ensemble d'un programme peut y résider. L'exécution d'un programme exige la lecture des instructions en mémoire en utilisant leur adresse. L'adresse d'une instruction doit être conservée dans un élément d'état en l'occurrence un registre appelé le compteur de programme (Program Counter - PC). La progression dans l'exécution du programme se fera par la mise à jour du compteur de programme qui dans la version la plus simple consiste à passer à l'instruction suivante (exécution séquentielle). Le passage à l'instruction suivante consiste alors à incrémenter l'adresse de l'instruction courante pour obtenir l'adresse de l'instruction suivante. Cette opération d'incrémentation peut s'effectuer électriquement par un additionneur. La figure suivante décrit les éléments nécessaires à cette opération de lecture mémoire instructions.

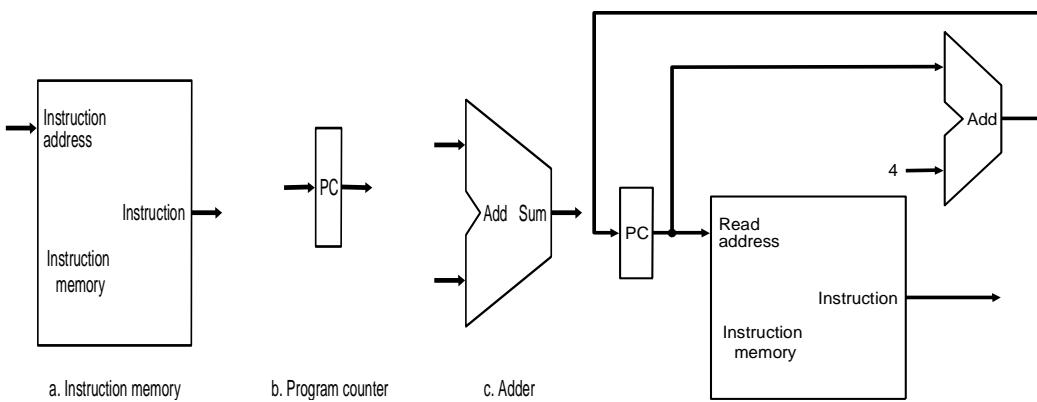


Figure 29. Mécanisme de lecture instructions (a) éléments structurants (b) connexions des éléments

Le calcul au sein du microprocesseur comme l'indique le jeu d'instruction se base sur des opérandes placés dans des registres. Ces registres sont physiquement regroupés ensemble pour former un banc de registres (register file). La figure suivante représente un banc de registres avec 2 ports de lectures (Read register number 1, Read Register number 2) et 1 port d'écriture (Write register, Write data). Cela signifie dans la pratique qu'il est possible de simultanément lire 2 registres distincts et écrire dans un registre distinct. Clairement la simultanéité ou le parallélisme d'accès augmente la performance par rapport à une version purement séquentielle où tous les accès se feraient de manière séquentielle dans ce cas avec un banc de registre à un seul port de lecture écriture.

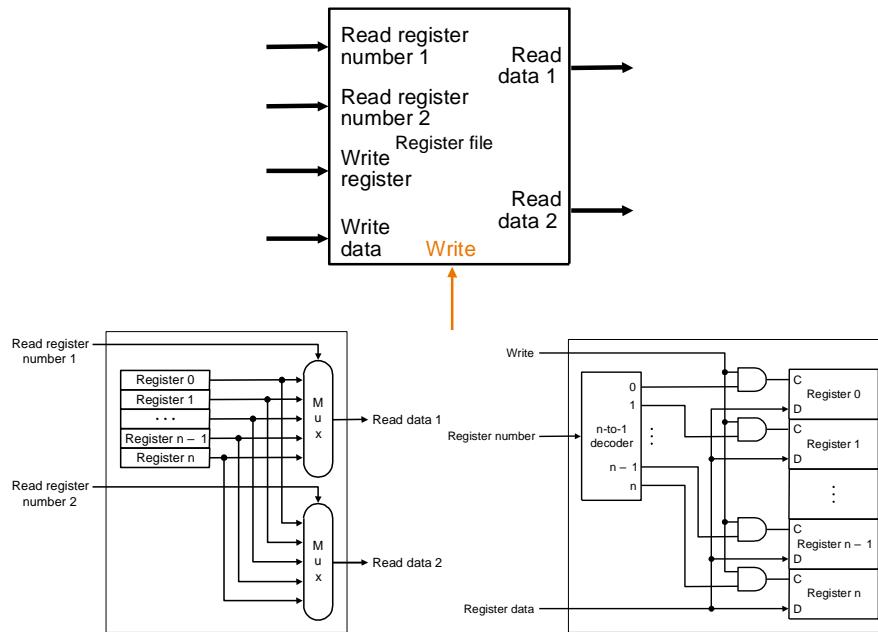


Figure 30.Banc de Registres (1) 2 ports de lecture 1 port en écriture (2) Accès Ecriture (3) Accès Lecture.

L'opération de lecture d'un registre consiste à placer sur un des ports de lecture Read register number 1 ou Read register number 2 le numéro de registre que nous souhaitons lire. La donnée correspondante apparaîtra sur le port de sortie correspondant Read data1 ou Read data 2. Une opération d'écriture dans un registre consiste à placer la donnée que nous souhaitons écrire sur le port donnée Write data et le numéro du registre ou nous souhaitons écrire la donnée sur Write register. L'opération est effectuée par l'activation du signal Write qui en passant à 1 active l'écriture dans le registre. La structure interne du banc de registres révèle un ensemble n de registres, un décodeur en entrée utilisé pour sélectionner le registre dans lequel on souhaite écrire et deux multiplexeurs en sortie utilisés en lecture pour sélectionner la sortie (la valeur) du registre que l'on souhaite lire. Il apparaît donc assez trivial d'ajouter des ports supplémentaires de lecture par ajout de multiplexeurs supplémentaires en sortie. L'ajout de ports d'écriture supplémentaire est un peu plus compliqué même si dans un premier temps on ajoute un décodeur supplémentaire pour sélectionner le registre dans lequel on souhaite écrire. Par contre en entrée du signal d'écriture du registre il faudra ajouter la possibilité d'une provenance d'un signal d'écriture supplémentaire. Cette structure simple n'est pas protégée contre les écritures simultanées distinctes sur le même registre ou les lectures et écritures simultanées sur le même registre. Le temps d'accès à un banc de registre en mode lecture ou écriture est dominé par les temps de propagation dans le décodeur, le temps d'accès au registre le temps de propagation dans le multiplexeur. Les temps de propagation sont eux dépendants du nombre de registres dans le banc de registres. Par conséquent l'augmentation du nombre de registres dans un banc de registre a un coût temporel au delà du coût en surface de silicium supplémentaire.

Comment ce banc de registres sera-t-il utilisé ? par exemple dans le cas d'une opération arithmétique add \$s1,\$s2,\$s3 l'instruction décodée sélectionnera sur le port de lecture 1 le registre s2 et sur le port de lecture 2 (read register2) le registre s3. Les valeurs correspondantes seront donc disponibles en sortie du banc de registre respectivement sur le port de sortie 1 (Read data1) et le port de sortie 2 (Read data2). Ces sorties étant directement connectées aux entrées d'une unité arithmétique et logique une sélection au préalable de l'opération permettra d'obtenir le résultat en sortie de l'ALU (result).

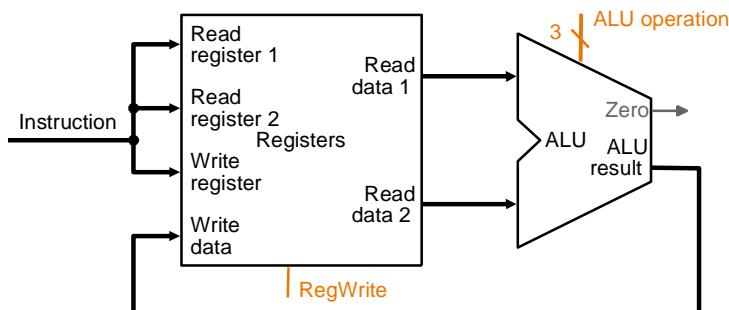


Figure 31. Connexion banc de registres et ALU

Pour conserver le résultat obtenu dans un registre du banc de registre il suffira alors de connecter directement la sortie de l'UAL au port donnée d'écriture du banc de registres. Le signal d'écriture RegWrite écrira le résultat obtenu dans le banc de registres.

L'ALU permet d'effectuer 5 opérations distinctes définies par les codages suivants.

Tableau 19. Codage en entrée de l'ALU

Signaux d'entrée de l'ALU	Fonction souhaitée
000	and
001	or
010	add
110	sub
111	slt

Les instructions de lecture et d'écriture nécessitent deux éléments structurants supplémentaires : (1) une mémoire donnée (2) un mécanisme d'extension de signe. La mémoire donnée est utilisée pour mémoriser les données. Le mécanisme d'extension de signe accepte en entrée une donnée signée codée sur 16 bits et l'étend sur 32 bits en conservant le signe.

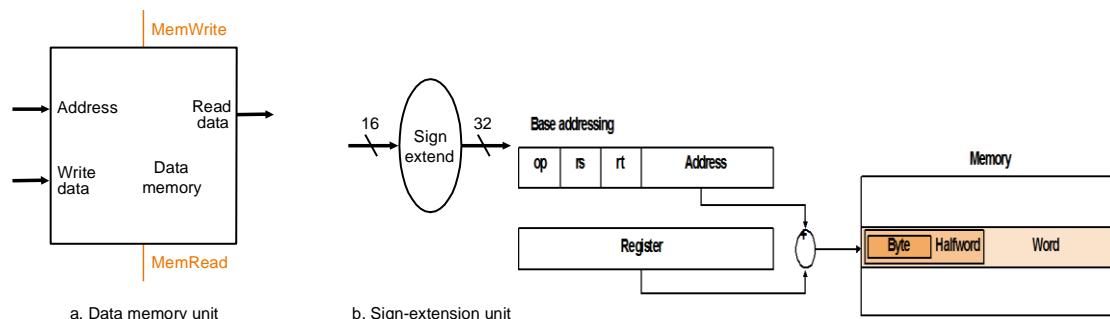


Figure 32. éléments structurants pour les instructions d'accès à la mémoire (2) mécanisme d'adressage des données.

Ce mécanisme est utilisé dans le mode d'adressage utilisé par les instructions de lecture et d'écriture des données. En effet ces instructions au format I composent l'adresse de la donnée à lire ou écrire comme la somme d'une valeur dans un registre et la valeur immédiate placée dans les 16 bits de faible poids du codage de l'instruction. L'addition nécessite des opérandes codées sur un même nombre de bits il est alors nécessaire d'étendre la valeur immédiate codée sur 16 bits à 32 bits. Le chemin de donnée constitué est décrit dans la figure suivante.

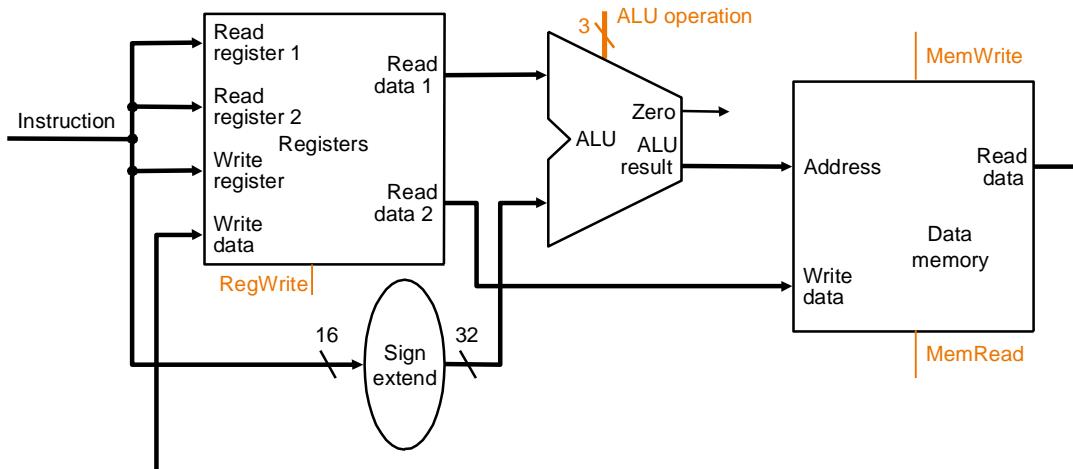


Figure 33.Chemin de donnée

La dernière partie à inclure est relative aux instructions de branchements conditionnelles. Pour ces instructions le mode d'adressage se fait par ajout de la valeur immédiate figurant dans les 16 bits de faible poids étendue à 32 bits et décalé de 2 bits pour tenir compte d'un adressage mémoire mot (32 Bits) taille des instructions MIPS.

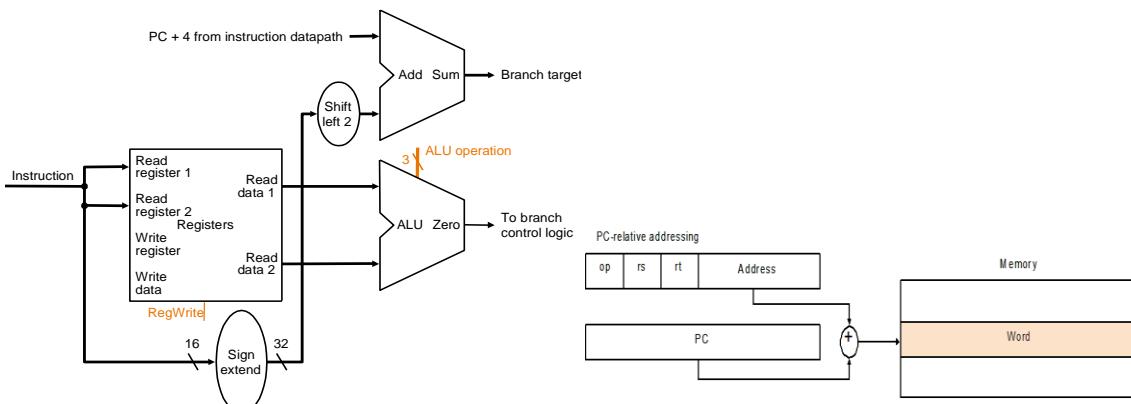


Figure 34.mode de calcul de l'adresse pour les instructions de branchements (2) modification associée du chemin de donnée

Le chemin de données obtenu par l'analyse du jeu d'instructions donne en connectant les différents sous ensembles obtenus la structure suivante :

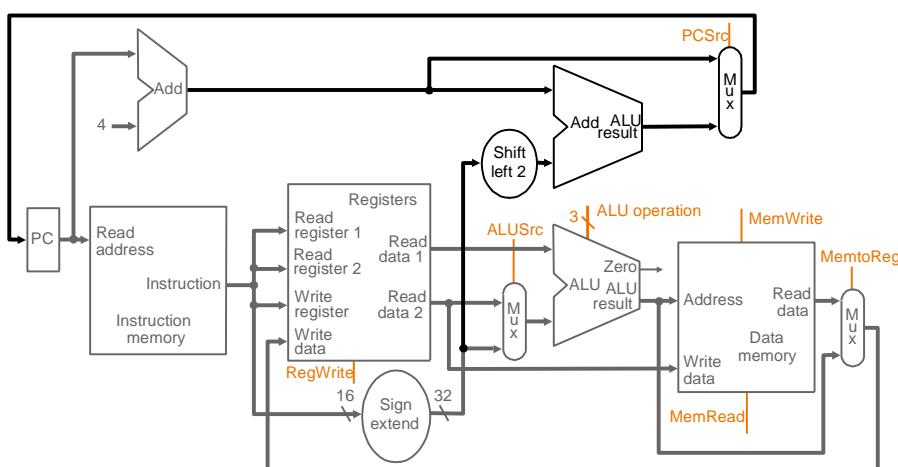


Figure 35.Chemin de donnée

L'ajout de 2 multiplexeurs l'un en entrée de la deuxième entrée de l'ALU et l'autre en entrée du PC permet dans chaque cas de tenir compte des différentes possibilités. Ce chemin de donnée ainsi constitué fait naturellement apparaître un ensemble de signaux de contrôle permettant de pouvoir activer ce chemin de donnée. La prochaine étape est donc la conception de l'unité de contrôle.

2.5.2 Conception Unité de Contrôle

L'unité de contrôle du microprocesseur accepte en entrée le code opération de l'instruction et génère en sortie les signaux de contrôle du chemin de donnée.

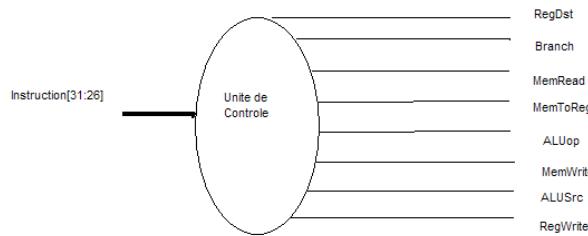


Figure 36. Unité de Contrôle

Les signaux de contrôle suivant leurs valeurs à 0 ou à 1 ont les effets décrits dans le tableau suivant.

Tableau 20. Description des Signaux de Contrôle

Nom de signal	Effet avec valeur égale à 0	Effet avec valeur égale à 1
RegDst	Le registre destination vient du champ rt	Le registre destination vient du champ rd
Branch	l'instruction décodée n'est pas une instruction branchement	l'instruction décodée est une instruction de branchement
MemRead	Aucun	Lecture mémoire donnée des données situées en mémoire à l'adresse fournie sur le port adresse
MemtoReg	La valeur fournie pour l'écriture dans le banc de registres provient de l'ALU	La valeur fournie pour l'écriture dans le banc de registres provient de la mémoire donnée
MemWrite	Aucun	Écriture mémoire donnée de la donnée située sur le port donnée à l'adresse fournie sur le port adresse
ALUSrc	La seconde opérande de l'ALU vient du deuxième port lecture de sortie du banc de registres (Read data2)	La seconde opérande de l'ALU est le mot de 16 bits extrait des 16 bits inférieurs de l'instruction et étendu en signe à 32 bits
RegWrite	Aucun	Écriture dans le banc de registre de la donnée en entrée du port d'écriture au numéro de registre spécifié sur le port WriteRegister du port en écriture.

L'unité de contrôle génère les valeurs appropriées à chaque signal en fonction du code opération de l'instruction. Quelque soit le format de l'instruction le code opération se situe dans les bits 31-26 des instructions comme rappelé dans le tableau suivant.

Tableau 21. Formats des Instructions Assembleur MIPS

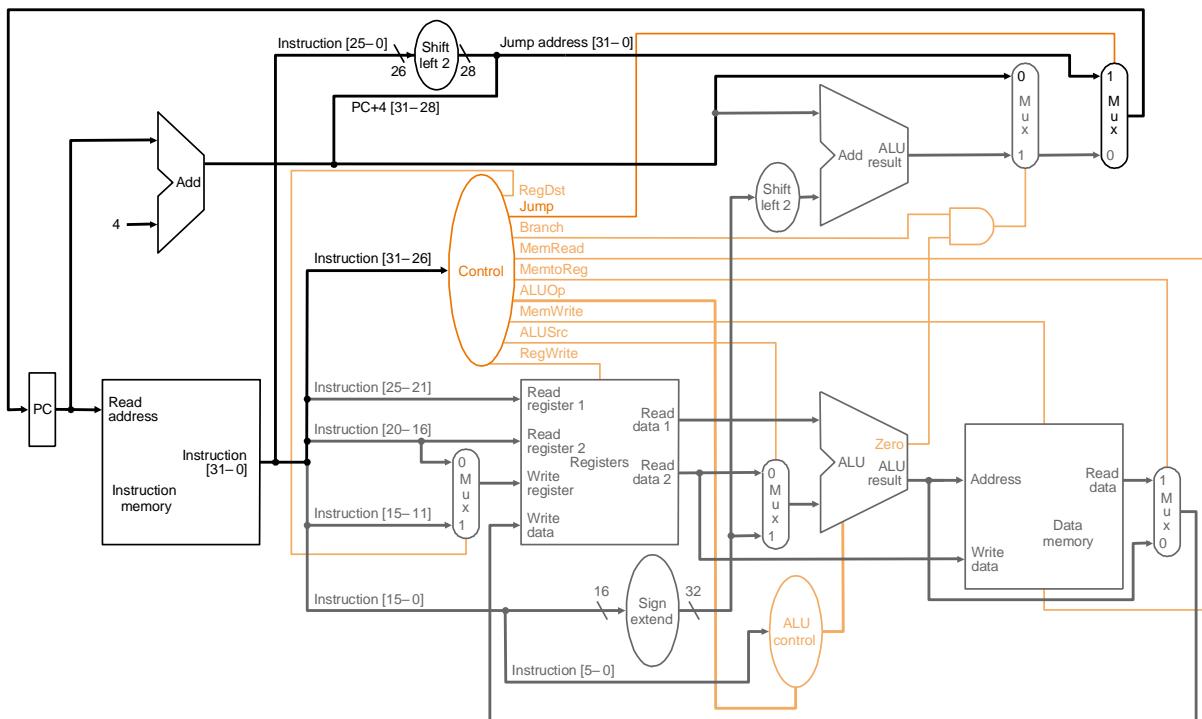
name		Fields						comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions 32 bits	
instruction	31-26	25-21	20-16	15-11	10-6	5-0	Instruction 32 bits Instruction[31:0]	
R-format	op	rs	rt	rd	shamt	Funct	Arithmetic instruction format	
I-format	op	rs	rt	Address/immediate			Transfer, branch, imm, format	
J-format	op	Target address				Jump instruction format		

C'est ce champ qui sera utilisé en entrée de l'unité de contrôle. En sortie de l'unité de contrôle on retrouvera l'ensemble des signaux associés au chemin de donnée.

Tableau 22. Spécification des signaux de contrôle pour Processeur Monocycle

Signaux en entrée		Signaux de contrôle en sortie								
Instruction	RegDst	Branch	Mem Read	MemtoReg	Mem Write	ALUSrc	Reg Write	ALUOp1	ALUOp0	
R-format	1	0	0	0	0	0	1	1	0	
lw	0	0	1	1	0	1	1	0	0	
sw	X	0	0	X	1	1	0	0	0	
beq	X	1	0	X	0	0	0	0	1	

Le processeur dans sa version finale monocycle est décrit dans la figure suivante.

**Figure 37. Processeur Monocycle**

En résumé la méthodologie de conception a suivi l'approche suivante :

Entrée: jeu d'instructions

Debut

1. Analyser le jeu d'instructions et identifier les éléments structurants de base du chemin de donnée
2. Etablir le chemin de donnée du processeur en connectant les différents éléments structurants entre eux par des bus ou des signaux
3. Identifier les signaux de contrôle du chemin de donnée
4. Etablir la table de vérité de l'unité de contrôle
5. Connecter les signaux de contrôle à l'unité de contrôle

Fin

L'inconvénient majeur d'une structure à cycle unique est que les durées d'exécution de toutes les instructions sont identiques quelque soit leurs complexités et que la fréquence maximale d'une telle structure est en fait déterminée par l'instruction ayant le temps d'exécution le plus long. Cela évidemment pénalise le temps d'exécution global d'un programme s'exécutant sur une structure de ce type.

2.6 Structure multicycle

Définition : une structure multicycle implique que les instructions s'exécutent en un nombre distinct de cycles.

Pour implémenter une structure multicycle il est nécessaire de découper le chemin de donnée de telle sorte à ne pas dépendre de l'ensemble de la logique combinatoire. Ce découpage s'effectue en plaçant des éléments de mémorisation à certains points dans la combinatoire permettant ainsi de s'affranchir de la partie amont de la combinatoire située avant le point de mémorisation. Ce découpage et découplage de différentes sous parties du chemin de donnée permet de faire de la réutilisation de ressources et évitent donc la duplication de ressources. C'est le cas de la mémoire instruction et données qui dans le traitement d'une instruction n'ont pas à être utilisés de manière simultanée. En reprenant le chemin de donnée de base du processeur monocycle nous pouvons spécifier un premier chemin de donnée pour processeur multicycle.

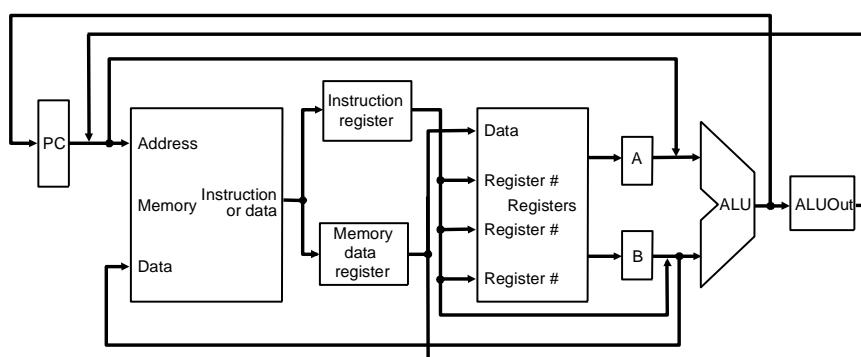


Figure 38.Chemin de donnée modifié

On remarquera : (1) qu'une seule mémoire est utilisée (2) il existe une seule ALU en lieu d'une ALU et deux additionneurs comme présents dans la figure 13 (3) 1 ou plusieurs registres sont placés en sortie de chaque unité fonctionnelle majeure dans notre cas, la mémoire, le banc de registres et l'ALU.

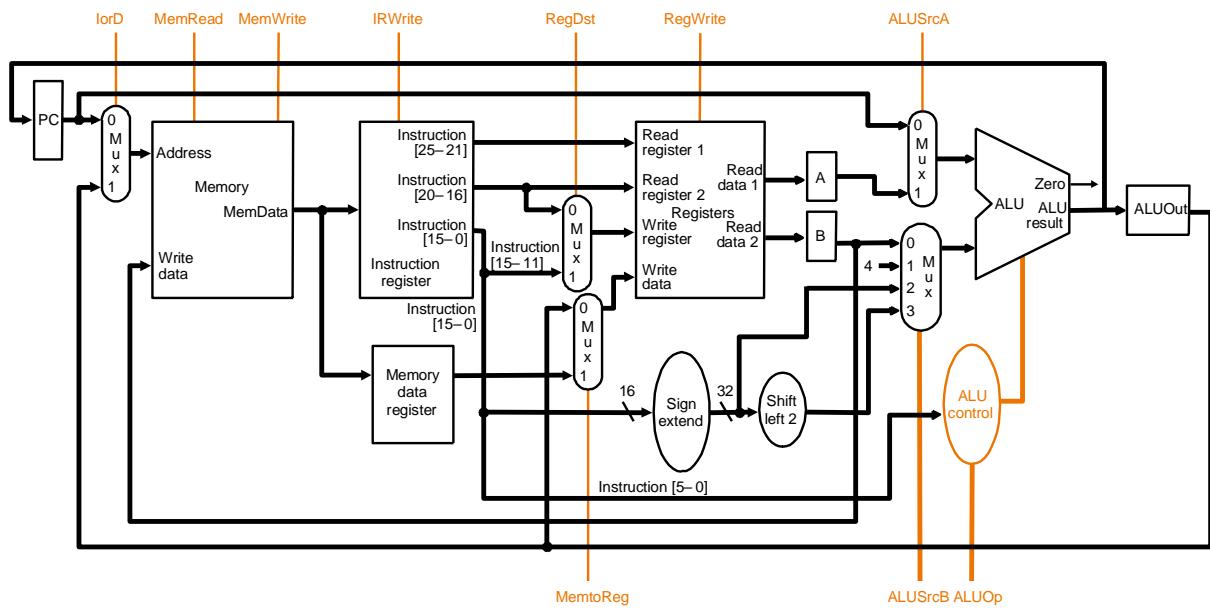


Figure 39. Chemin de donnée du Processeur Multicycle

La version étendue de ce chemin de donnée est spécifiée par la figure 15. Un ensemble de signaux de contrôle ont été bien entendus ajoutés comme conséquence de ce nouveau découpage du chemin de donnée mais le principe le plus important à respecter sera l'activation de ces signaux suivant un mode multicycle. Pour cela une machine d'état sera nécessaire contrairement au processeur monocycle ou seule de la logique combinatoire était suffisante pour l'unité de contrôle. La structure générale de cette machine d'état se base sur un (des) état(s) initiaux communs à toutes les instructions correspondant aux actions de lecture d'instruction en mémoire, décodage d'instructions suivis par des chemins distincts dans la machine d'états qui sont propres à chaque classe d'instructions. Ces chemins distincts garantissent un traitement et un nombre de cycles adapté à chaque classe d'instructions.

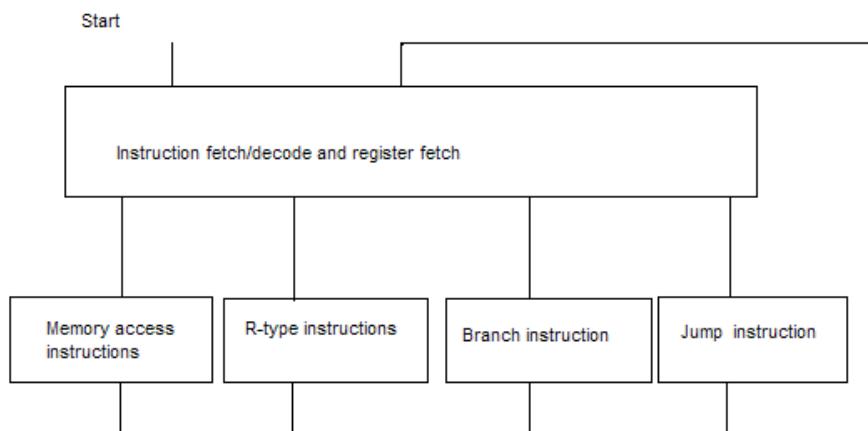


Figure 40. Machine d'états de l'unité de contrôle Processeur Multicycle

La version détaillée est donnée par la figure suivante.

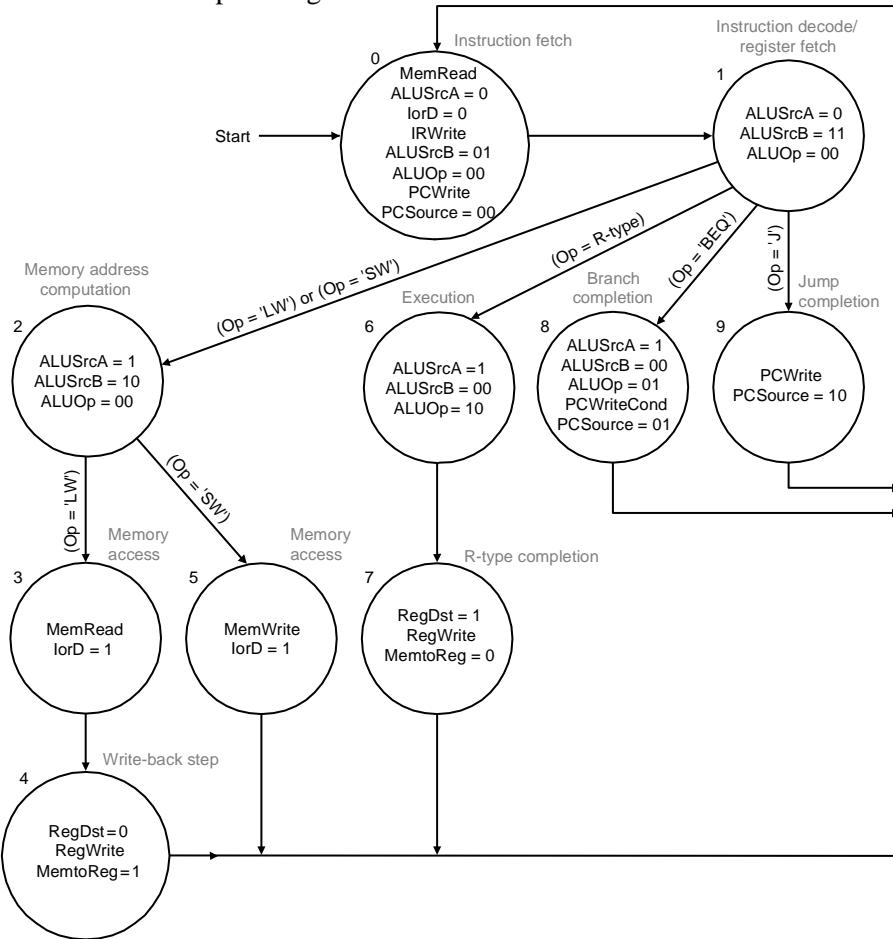


Figure 41. Machine d'états détaillée – Processeur Multicycle

L'ajout de nouvelles classes d'instructions s'effectue en modifiant la machine d'états pour y ajouter les états supplémentaires nécessaires au traitement de ces instructions.

Le processeur multicycle complet est donné par la figure suivante.

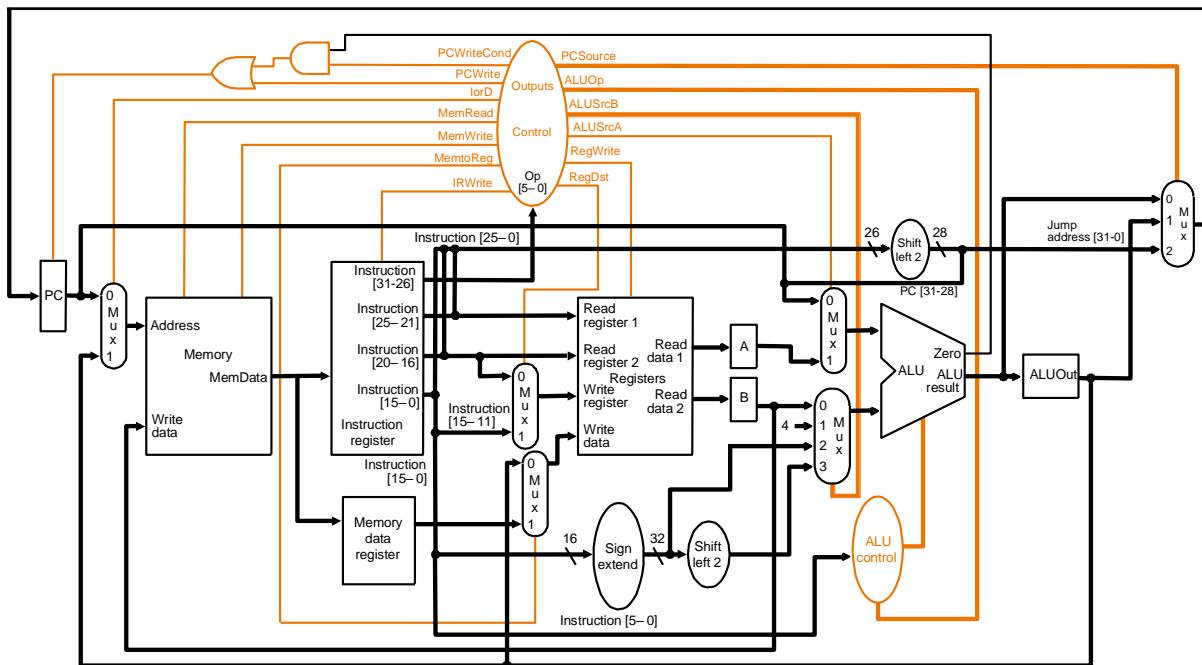


Figure 42. Processeur Multicycle

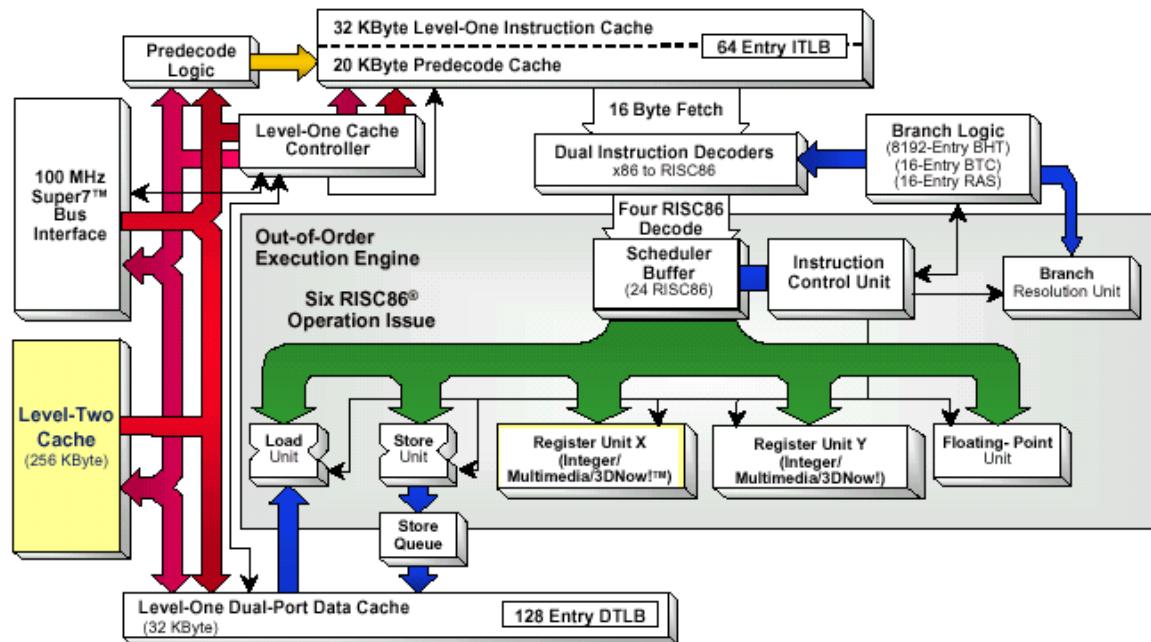
2.7 Extensions du jeu d'instructions

Les extensions du jeu d'instruction sont possibles et exigent par rapport au modèle de machine synchrone la modification de l'automate d'états finis de contrôle du microprocesseur ainsi que la modification du chemin de données avec l'ajout éventuel d'unités fonctionnelles supplémentaires permettant l'exécution des instructions supplémentaires.

Les ajouts les plus significatifs des dix dernières années ont été des instructions spécialisées pour le traitement des applications Multimédia présentant des caractéristiques et des exigences différentes des programmes entiers classiques.

On citera les ajouts les plus significatifs : Alpha MAX, AMD (3D Now), Intel (MMX, SSE, SSE2), Motorola-IBM PowerPC Altivec, HP PA-RISC MAX2, Sun (VIS). Le problème initial soulevé par ces ajouts est l'absence d'outils logiciels permettant l'exploitation optimisé de ces instructions. Il existe un certain temps de latence avant que ces optimisations soient intégrées dans ces outils logiciels.

Ces instructions ont été en général implémentées sur un mode d'exécution de type SIMD (Single instruction Multiple data) permettant l'exécution en parallèle de plusieurs instructions identiques sur des données différentes. Ce type d'instructions est bien adapté aux opérations effectuées en traitement de signal ou d'images ou souvent des opérations répétitives sont effectuées sur une grande structure de donnée.

**AMD-K6®-III Processor Block Diagram****Figure 43. Processeur AMD K6-III Incluant des unités 3D Now !**

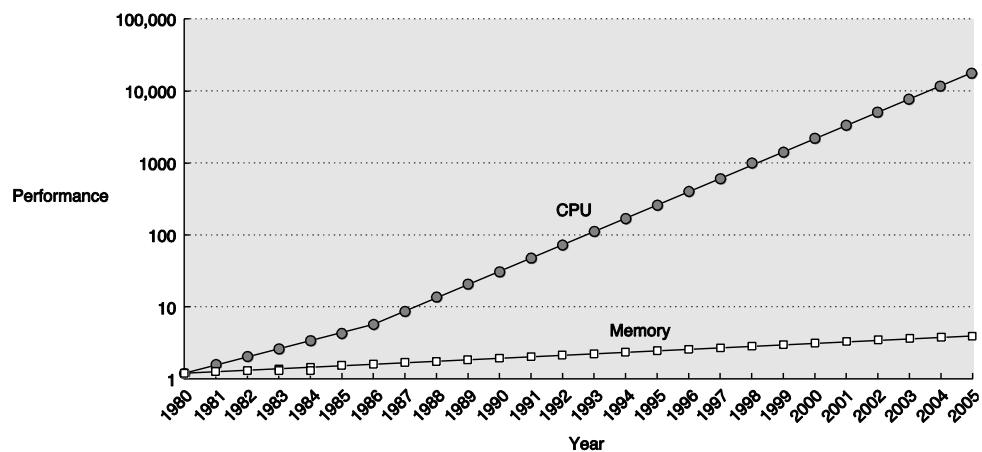
Ces instructions ont stimulés l'écriture de bibliothèques spécialisées et optimisées pour le traitement de signal, les manipulations mathématiques.

3 Chapitre 3 Hiérarchie Mémoire

3.1 Le « Mur Mémoire »

La structure de base de la machine de von Neuman exige que le programme ainsi que les données associées se trouvent dans une mémoire. Cette mémoire sera alors accédé en cours d'exécution pour lire les instructions et les données associées. La nature de cette mémoire n'est pas spécifiée mais il est souhaitable que cette mémoire soit telle que son utilisation ne génère pas ou peu de délais dans l'exécution du programme car elle ne contribue pas au calcul et sert uniquement de sauvegarde.

Malheureusement la technologie semi-conducteur des mémoires n'a pas réussi à fournir une telle mémoire et c'est la situation inverse qui prévaut c'est-à-dire qu'il existe un fossé significatif entre les fréquences d'utilisation des microprocesseurs et les temps d'accès mémoire. Ce fossé s'est accru avec le temps et continue de s'accroître et la conséquence en est que les accès mémoires sont très pénalisants pour les performances.



© 2003 Elsevier Science (USA). All rights reserved.

Figure 44. Evolution des Performances Micropcesseurs et Mémoires

La figure précédente montre la disparité croissante entre les performances des microprocesseurs et celles des mémoires.

Ce problème structurel est aggravé par deux contraintes :

- économique : il existe plusieurs types de mémoires (SRAM/SDRAM/DDR2/DDR3) et deux relations simples les dominent « plus une mémoire est rapide et plus elle est chère »,
- physique : plus une mémoire est grande plus son temps d'accès est important

Cette situation crée le problème considéré comme le problème No. 1 dans l'architecture des microprocesseurs : le mur mémoire (*Memory Wall*).

Il est donc absolument nécessaire de proposer une solution à ce problème sans quoi l'activité principale d'un microprocesseur consisterait à attendre les données et instructions demandées à la mémoire et nécessaires à l'exécution d'un programme. La solution proposée et existante dans toutes les machines actuelles est la hiérarchie mémoire.

Définition Hiérarchie mémoire : une hiérarchie mémoire est une hiérarchie de mémoires de taille et de temps d'accès décroissants.

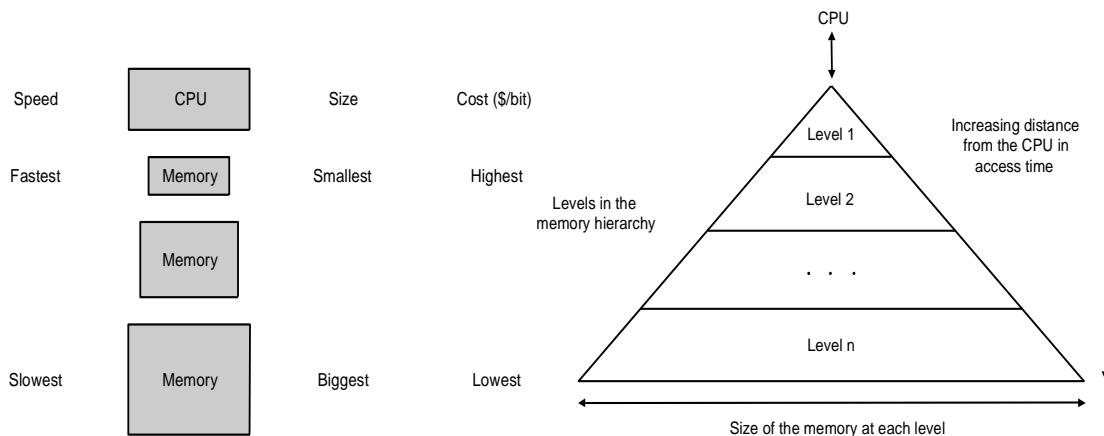


Figure 45.Hiérarchie Mémoire : (a) relation entre taille et temps d'accès (2) relation entre taille et distance au microprocesseur

La hiérarchie mémoire est telle que la mémoire la plus proche du microprocesseur est la plus petite mais aussi la plus rapide. L'hypothèse principale sur laquelle repose cette organisation est qu'à un instant donné seul un sous-ensemble des données et du programme est nécessaire à l'exécution et que ce sous-ensemble peut se placer dans cette mémoire la plus rapide donnant l'illusion d'accès quasi permanent à une mémoire de grande taille et rapide. Cette hypothèse a été systématiquement vérifiée et est décrite par le *principe de localité*. Le principe de localité exprime le fait que la majorité des programmes n'accèdent pas le code ou les données de manière uniforme (une règle empirique indique que 90% de l'exécution se fait dans 10% du code du fait de l'utilisation des structures de boucles (while/for)).

Deux types de localité de références ont été observés : (1) la localité temporelle (2) la localité spatiale.

Définition Localité Temporelle : la localité temporelle est une propriété sur la distribution des accès aux adresses mémoires indiquant que des éléments mémoires « récemment » accédés le seront probablement à nouveau dans un futur proche.

Exemple : un exemple trivial est une variable de boucle *i* dans une boucle de type *for(i= 0 ; i < N ; i++) {corps de la boucle}*.

Définition Localité Spatiale: la localité temporelle est une propriété sur la distribution des accès aux adresses mémoires indiquant que des éléments mémoires proches dans l'espace mémoire ont une tendance à être accédés de manière proche dans le temps .

Exemple : calcul sur des vecteurs et matrices.

Comme on peut le constater les deux définitions ne sont pas quantifiées et par conséquent la notion de localité est une notion relative à la distribution des accès aux adresses mémoire d'un programme. Une manière de visualiser les localités consiste à « tracer » un programme à l'exécution c'est à dire extraire à l'exécution en conservant l'ordre des accès la séquence complète des accès mémoires au code et la séquence complète des accès mémoires aux données et de visualiser sur un graphe ayant en abscisse le numéro d'accès et en ordonnée l'adresse mémoire. De nombreux outils logiciels

existent dont certains dans le domaine public permettant de tracer un programme à l'exécution.
(Exemple : ATOM, simplescalar)

Le principe de localité couplé aux contraintes structurelles des mémoires citées plus haut conduit à une organisation générale comme suit.

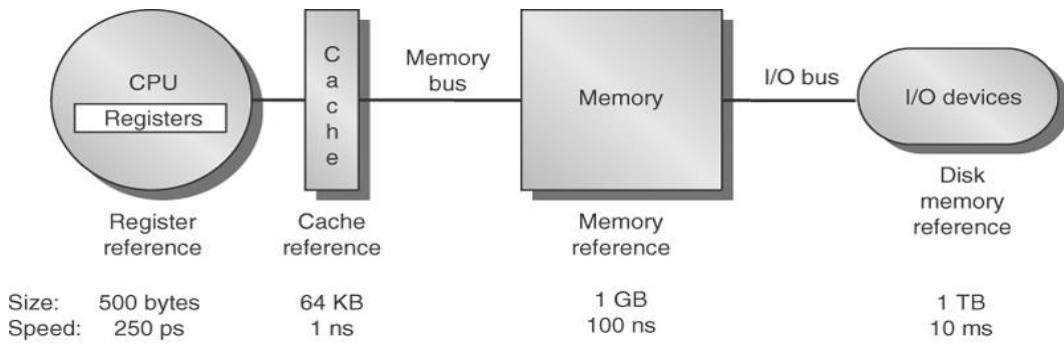


Figure 46. Organisation générale

Dans un premier niveau le plus proche du microprocesseur on trouvera la ou les mémoires caches suivies de la mémoire centrale puis du disque. Le mouvement initial des données dans une hiérarchie mémoire se fait depuis la mémoire la plus large vers la plus petite suite bien entendu à une requête sur ces données. De manière classique dans un ordinateur les données sont conservées sur disque puis lors de l'appel du programme par l'utilisateur le système opératoire charge ce programme du disque dans la mémoire centrale. Par la suite le processeur par ses requêtes successives amènera progressivement au cours de l'exécution les données utiles vers les niveaux les plus proches pour arriver au plus proche. Les mémoires ayant des capacités limitées et décroissantes va s'opérer au cours de l'exécution un ensemble d'échanges de données entre les différents niveaux. Il faut réaliser que ces échanges se font dans le cadre d'une bande passante mémoire offerte par un medium de communication entre les niveaux qui est elle même limitée.

Se pose ainsi un problème général de gestion de la hiérarchie mémoire. Ce problème dans le cas idéal (ou l'ensemble des accès sont connus à l'avance) peut être posé comme un problème mathématique d'optimisation ayant pour fonction d'objectif la minimisation des échanges inter mémoires.

La définition de la hiérarchie mémoire n'ayant pas spécifier de limites sur le nombre de niveaux dans la hiérarchie il existe des variations de cette organisation générale ayant davantage de niveaux en particulier sur le nombre de niveaux de caches actuellement au nombre de 3. Sur un ordinateur personnel ou une station de travail il est habituel d'augmenter la taille de la mémoire centrale ou du disque mais il est généralement impossible de modifier les tailles de caches qui se trouvent souvent intégrés sur la puce du microprocesseur.

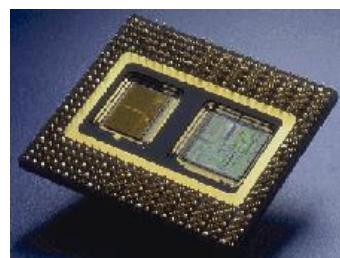


Figure 47. Exemple de microprocesseur avec cache de second niveau intégré

3.2 Mémoires Caches

3.2.1 Définitions

Définition Cache : la mémoire cache de niveau i est la mémoire située au niveau i de la hiérarchie mémoire. La mémoire cache de niveau 1 (L1) est celle directement accédée par le microprocesseur.

Définition Cache Hit : lorsque le microprocesseur accède le cache pour y accéder une donnée et qu'elle s'y trouve c'est un **cache hit**.

Définition Cache Miss: lorsque le microprocesseur accède le cache pour y accéder une donnée et qu'elle ne s'y trouve pas c'est un **cache miss**.

Définition Miss Rate : le miss rate est le taux de cache miss sur l'ensemble des accès au cache.

Définition Miss Penalty : lorsqu'un cache miss advient la donnée doit être recherchée au niveau suivant de la hiérarchie et cette recherche a un coût temporel exprimable en nombre de cycles processeurs. Ce coût est le miss penalty.

Un modèle simple de performance de microprocesseur (CPU) prenant en compte l'impact des caches est donné par l'équation suivante :

$$\text{CPU Exécution Time} = (\text{CPU clock cycles} + \text{Memory stall cycles}) \times \text{clock cycle time}$$

Le temps d'exécution CPU est partagé entre le temps de calcul effectif et d'accès mémoire sans défaut (CPU clock cycles) et le temps pendant lequel la CPU est en attente de données de la mémoire (*Memory Stall cycles*).

Ce temps peut lui-même être décomposé de la manière suivante avec IC (*Instruction Count*) représentant le nombre d'instructions :

$$\begin{aligned}\text{Memory stall cycles} &= \text{number of misses} \times \text{Miss Penalty} \\ &= \text{IC} \times [\text{Misses}/\text{Instruction}] \times \text{Miss Penalty} \\ &= \text{IC} \times [\text{Memory accesses}/\text{instruction}] \times \text{Miss rate} \times \text{Miss penalty}\end{aligned}$$

ce qui donne en équation finale:

$$\text{Memory stall cycles} = \text{IC} \times [\text{Memory accesses}/\text{instruction}] \times \text{Miss rate} \times \text{Miss penalty}$$

Définition Bloc : un bloc (block) est un ensemble de mots d'adresses mémoire consécutives. Le bloc est l'unité minimum échange entre les différents niveaux de la hiérarchie mémoire.

Le nombre de mots (ou d'octets) composant un bloc mémoire est variable suivant les microprocesseurs et est un des paramètres à optimiser dans la spécification d'une hiérarchie mémoire.

Lors d'une opération de lecture ou d'écriture un microprocesseur accède un mot qui se situe dans un bloc mémoire. Le cache miss (ou hit) représente un cache miss (hit) sur le bloc.

L'adresse d'un mot en mémoire peut être vue comme l'adresse du bloc qui contient le mot suivi du déplacement dans le bloc (block offset) pour atteindre le mot.

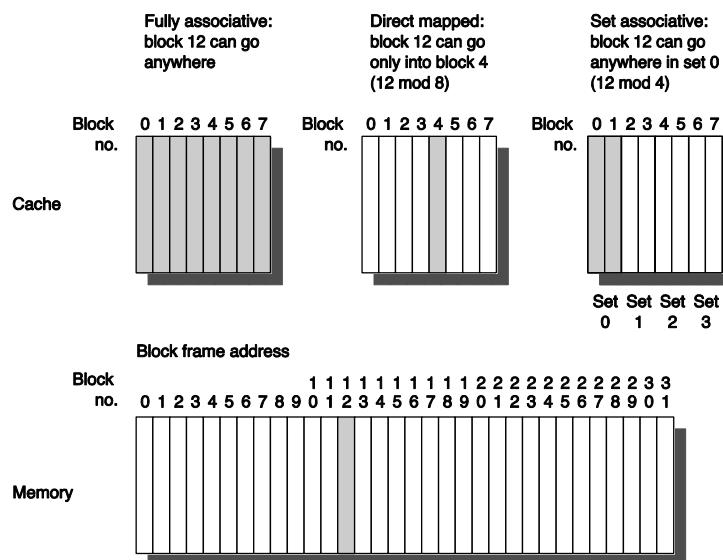


Figure 48.Découpage de l'adresse

Cette adresse bloc peut être elle-même découpée 2 champs supplémentaires : (1) tag (2) index. L'index servira à accéder la mémoire cache et le tag complétera l'identification du bloc.

3.2.2 Structure de base de caches

Il apparaît évident a partir de la structure de hiérarchie mémoire qu'il n'existe pas de fonction bijective entre l'espace d'adresse mémoire de niveau i et l'espace d'adresse mémoire de niveau j. Il est donc nécessaire de trouver une fonction de placement qui préserve autant que possible le principe de localité tout en étant : (1) simple (2) rapide : la simplicité devant se traduire par une implémentation matérielle ayant un temps de propagation minimal.



© 2003 Elsevier Science (USA). All rights reserved.
Figure 49. Structure de caches et fonction de correspondances

Il existe essentiellement 3 types d'organisations décrites schématiquement dans la figure ci-dessus. Quoique la littérature scientifique sur les caches est extrêmement abondante et que les caches restent un sujet de recherche actif la pratique n'a conservé que deux types essentiels d'organisation. La première organisation est dite **direct-mapped** et consiste simplement à utiliser comme fonction de correspondance une fonction modulo comme suit :

Adresse du bloc *mod* (nombre de blocs dans le cache)

De cette manière il n'existe qu'une seule correspondance possible entre un bloc situé en mémoire et un bloc dans le cache. Du fait du modulo plusieurs blocs en mémoire auront le même emplacement dans le cache mais seulement un d'entre eux pourra à un moment donné être placé dans le cache. Si par exemple au cours de l'exécution deux blocs partageant le même emplacement mémoire étaient accédés par le microprocesseur cela créerait des caches misses car seul l'un des deux pourra être présent dans le cache à un moment donné.

Un exemple de structure direct-mapped est donné dans la figure suivante. La structure de cache est constituée de 3 parties pouvant être vues comme des tableaux (dans cet exemple de taille 1024). La première partie de taille 1 bit sert à spécifier si la donnée accédée est valide ou non. La deuxième partie indique le Tag (identifiant) du bloc et finalement la troisième et dernière partie contient la donnée que l'on souhaite accéder.

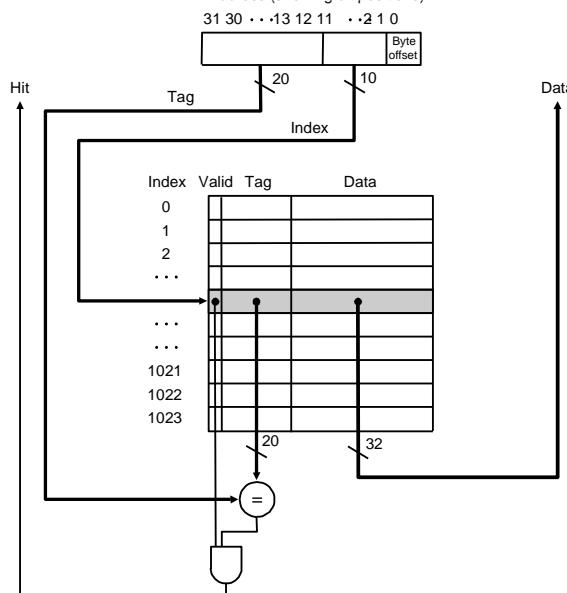


Figure 50. Structure de cache Direct-mapped

La deuxième structure de cache dite set-associative utilise aussi une fonction modulo mais cette fois ci sur un nombre d'ensemble comme suit :

Adresse du bloc *mod* (nombre d'ensemble dans le cache)

Dans cette structure la première décomposition du cache se fait par ensembles. Ces ensembles sont des ensembles de blocs du cache. Par conséquent la fonction de correspondance associe à chaque bloc un ensemble dans le cache et le bloc peut être placé n'importe où dans l'ensemble ce qui augmente le nombre de possibilités de placement et réduit le nombre de conflits. On dit que la fonction de placement est **n-way set-associative** s'il existe n blocs dans l'ensemble. L'associativité d'un cache est ce nombre n. Une associativité de 1 correspond à un cache direct.

La figure suivante montre un cache d'associativité 4.

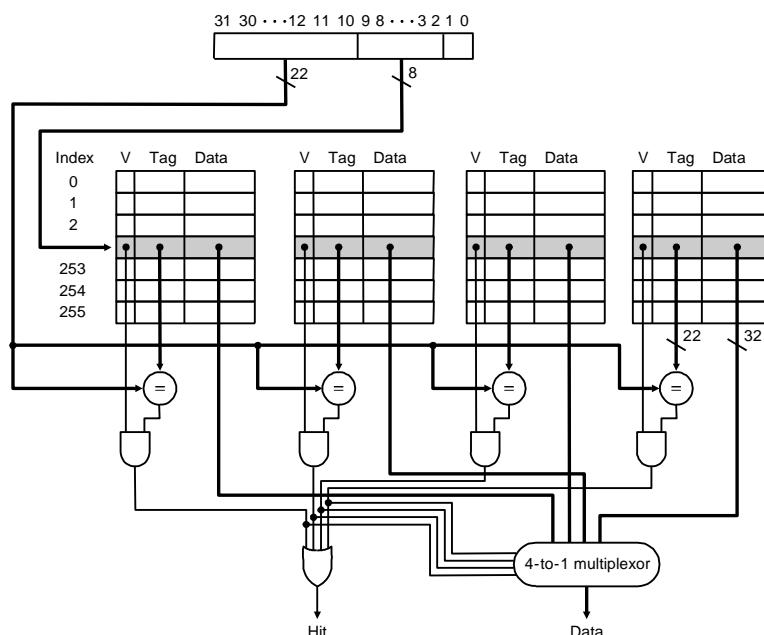


Figure 51. Structures de cache 4-way set associative

Si la taille du cache est gardée constante l'accroissement de l'associativité augmente le nombre de blocs par ensemble et par conséquent décroît la taille de l'index.

La méthode d'accès est donc comme suit :

3.2.3 Méthode d'accès au cache

Début

1. Accéder le cache avec l'index et lire le(s) tag(s) associé
2. Comparez ces tags avec le tag de l'adresse
3. **Si** les tags sont identiques **alors** {c'est un cache hit ; utiliser cette information pour sélectionner la donnée correspondante}
4. **Sinon** {c'est un cache miss ; rechercher la donnée au niveau supérieur de la hiérarchie.}

Fin

L'étape 1 a un temps d'accès qui est celui de l'accès à la mémoire utilisée. L'étape 2 a un temps équivalent au temps de propagation d'un comparateur. L'étape 3 a un temps équivalent dans les caches associatifs au temps de propagation dans un multiplexeur.

L'intérêt du cache à accès direct par rapport aux caches associatifs est le fait que la donnée est présente au même moment que le test. Ceci revient à paralléliser ces actions et donc revient à décroître le temps d'accès moyen au cache.

Lorsque le processeur lors d'un accès au cache rencontre un miss il est nécessaire de lire la donnée du niveau suivant de la hiérarchie mémoire et le placer dans le cache. Si l'emplacement associé dans le cache est déjà occupé il est nécessaire de sélectionner une donnée à remplacer. Le problème de la sélection ne se pose dans le cas d'un cache direct mapped car un seul emplacement est possible et c'est donc cette donnée qui sera remplacée. Dans le cas d'un cache associatif plus d'un candidat peut être sélectionné.

Trois stratégies de base sont utilisées : (1) aléatoire (*random*) (2) le moins récemment utilisé LRU (*Least Recently Used*) (3) FIFO. L'évaluation de performances des trois stratégies sur 10 benchmarks de SPEC2000 et une taille de bloc de 64 octets montre que les trois stratégies se valent sur des caches de taille importante (256 KB) mais que LRU l'emporte sur des caches de plus petite taille. L'implémentation matérielle de LRU peut être compliquée à calculer et dans la pratique est approximée.

L'écriture dans un cache généralement suit une des deux options suivantes : (1) **write through** (2) **write back**. Write through exige qu'écrire dans un bloc du cache implique aussi écrire en mémoire. Dans write-back l'écriture ne se fait que dans le bloc. Ce bloc ne sera écrit en mémoire que lorsqu'il sera remplacé. Cette dernière option résulte à un instant donné en une incohérence des données dans la hiérarchie mémoire mais pas pour le processeur.

La politique de remplacement décrite ci-dessus exige l'écriture du bloc remplace que dans le cas où le cache répond à une politique de write-back. Lors d'un write miss le processeur a deux options : (1) **write allocate** : le bloc est alloué suite à un write miss suivi des opérations d'écriture (2) **no-write allocate** : il n'y a pas d'allocation de bloc et seul le bloc au niveau inférieur est modifié. Par conséquent les blocs ne sont pas dans le cache tant qu'ils ne sont pas lus tandis que les blocs le seront dans write allocate même s'ils ne sont pas lus par la suite.

Toutes les combinaisons sont possibles entre les techniques d'écriture et les options sur write miss. Dans la pratique les caches à base de write-back utilisent write-allocate tandis que write-through utilisent souvent le no-write allocate. Le raisonnement effectué dans le premier cas est basé sur l'espoir que dans

le cas d'écritures successives elles se feront dans le cache tandis que dans le second cas il n'existe tout simplement pas d'intérêt puisque de toutes manières l'écriture doit se faire systématiquement dans le niveau inférieur.

Enfin il apparaît clairement qu'il n'existe pas d'organisation et de gestions uniques qui soient optimales pour tous les programmes. La flexibilité est nécessaire.

C'est dans cet esprit la que les constructeurs de microprocesseurs ont introduit la flexibilité dans l'organisation des caches en permettant à l'utilisateur de paramétriser le cache et dans la gestion en proposant dans le jeu d'instruction des instructions permettant de gérer le cache.

Le tableau suivant présente des instructions spécialisées ajoutées au jeu d'instructions du microprocesseur de la famille IBM/Motorola PowerPC et pouvant servir de base à diverses techniques de gestion du cache.

Tableau 23 Instructions de gestion du cache sur IBM/Motorola PowerPC

instruction	commentaire
dcbf rA,rB Data cache block flush	Flush the data cache line
dcbi rA, rB Data cache block invalidate	Invalidate the data cache line
dcbst rA, rB Data cache block store	Store the data cache line to memory
dcbtst rA, rB Data cache block touch for store	Touch for store the data cache line
dcbz rA, rB Data cache block zero	Clear the data cache line

L'instruction **dcbf** écrit le bloc en mémoire s'il a été modifié et invalide la ligne de cache. L'instruction **dcbi** invalide la ligne de cache. Si le bloc a été modifié les changements sont éliminés. **dcbst** écrit le bloc en mémoire. **dcbt** donne l'information au processeur que le bloc en référence sera probablement chargé dans un futur proche. Le processeur peut ignorer cette information ou peut charger sur son initiative ce bloc. **dcbtst** est similaire à dcbt mais cette fois ci donne l'information que le bloc spécifié sera probablement écrit dans un futur proche. **dcbz** met tout simplement l'ensemble des données du bloc à zéro. On réalise que ces instructions permettent de pouvoir effectuer par l'intermédiaire d'instructions ce qui est normalement câblé. Cette opportunité d'agir sur la gestion du cache est encore un des nombreux exemples d'interaction entre la compilation et l'architecture des microprocesseurs. Ces instructions peuvent donc être utilisées par un compilateur pour appliquer des algorithmes sophistiqués de gestion de la mémoire sur la base du programme.

3.3 Performance de cache et outil d'évaluation

Une mesure de la performance d'une hiérarchie mémoire est donnée par le temps d'accès moyen à la mémoire (*AMAT - Average Memory Access Time*). Ce temps s'exprime de la manière suivante :

$$\text{AMAT} = \text{Hit Time} + (\text{Miss Rate} \times \text{Miss Penalty})$$

Avec Hit time étant le temps nécessaire pour effectuer un hit dans le cache. L'équation de calcul de performance d'un processeur s'écrit :

$$\text{CPUtime} = \text{IC} \times (\text{CPIexe} + [\text{Memory stall clock cycles/instruction}] \times \text{clock cycle time})$$

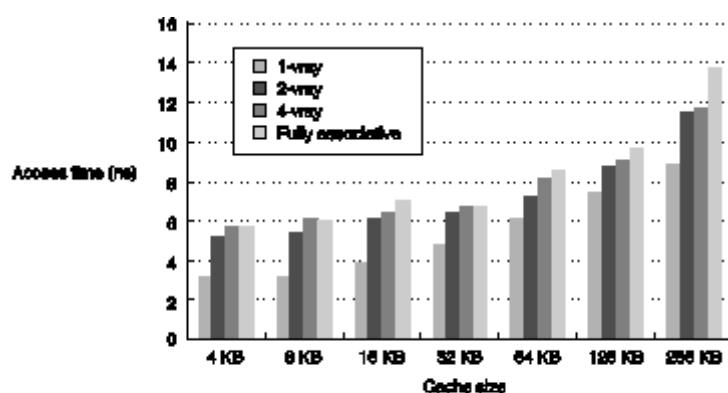
Où CPIexe est le nombre de cycles par instructions (CPI – Cycle per Instruction) pour les instructions exécutées

L'équation peut se réécrire en :

$$\text{CPUtime} = \text{IC} \times (\text{CPIexe} + \text{Miss rate} \times [\text{Memory accesses/instruction}] \times \text{Miss Penalty}) \times \text{clock cycle time}$$

Il apparaît donc de manière claire que les caches ont un impact direct sur la performance.

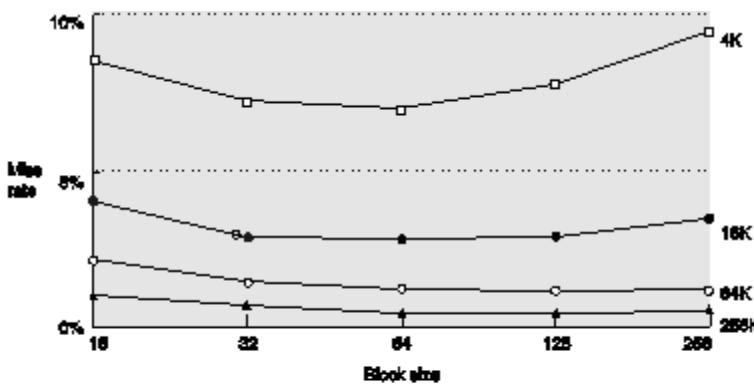
La spécification des paramètres d'une hiérarchie mémoire se fait par des études à base de simulation en utilisant divers programmes généralement des benchmarks comme décrit dans le chapitre 1. Cette exploration de l'espace des paramètres dans sa généralité est un problème d'optimisation multi objectifs : (1) temps d'accès (2) surface silicium dédiée aux caches (3) consommation d'énergie des caches. La résolution du problème général dépasse le cadre de ce cours mais on peut observer sur deux exemples comment au cours de cette exploration l'augmentation de la valeur d'un paramètre peut affecter d'autres paramètres. Le premier exemple décrit dans la figure suivante montre qu'une augmentation de l'associativité d'un cache souhaitable pour la réduction du miss rate se traduit par un temps d'accès plus important à taille de cache égale.



© 2003 Elsevier Science (USA). All rights reserved.

Figure 52.Temps d'accès vs associativité

Le deuxième exemple montre qu'une augmentation de la taille d'un bloc peut réduire le miss rate mais pour certaines tailles de cache se traduire au delà d'un certain point par un accroissement du miss rate.



© 2003 Elsevier Science (USA). All rights reserved.

Figure 53. Miss rate vs Taille du bloc

La recherche en architecture des ordinateurs se base essentiellement sur les techniques de simulation pour l'évaluation de performances. Dans le cas de l'évaluation de performance de la hiérarchie mémoire il est nécessaire d'écrire un simulateur de hiérarchie mémoire efficace, flexible et précis. Il existe deux types de simulation : (1) à base de traces de programmes (*trace-based simulation*) (2) dirigée par l'exécution (*execution driven simulation*).

La technique à base de traces de programme consiste à utiliser comme entrée du simulateur des fichiers contenant des traces de références mémoires de programmes benchmarks. Cela exige tout d'abord une technique de collecte des références mémoires qui n'affecte pas la mesure. La deuxième technique exécute un programme et génère les références comme le ferait un programme à l'exécution. Le simulateur dans les deux cas génère différentes statistiques dont entre autres les miss rates des différents caches composant la hiérarchie mémoire. Un simulateur dans le domaine public est sim-cache de la distribution simplescalar (www.simplescalar.com).

3.4 Surface Silicium des caches

Les études d'évaluation de performances de cache se traduisent au final par des spécifications des caches en termes de tailles, d'organisation (hiérarchie (cache L1 (séparé, unifié), cache L2 (séparé, unifié), cache L3), tailles de blocs,) qui seront directement utilisés pour la conception des circuits correspondants. Cela se traduit au final en surface de silicium dont le coût est directement évalué en fonction de cette surface. Le problème de la hiérarchie mémoire doit donc être vu comme un problème d'optimisation multiobjectif dont les critères sont: (1) la performance, (2) le coût (la surface) et (3) la consommation d'énergie. Des outils logiciels comme CACTI 4.2 permettent de calculer en fonction des paramètres suivants:

- taille du cache,
- taille du bloc de cache,
- nombre de bancs,
- associativité,
- technologie de fabrication semi-conducteur en nm (130, 90, 65, 45 nm),
- nombre de ports en lecture/écriture/lecture-écriture
- mode d'accès

la surface nécessaire à l'implémentation du cache, sa consommation d'énergie et les différents temps d'accès.

Tableau 24. Exemple: Cache 4M octets + bits de parite, bloc de cache 32 octets, associativité 4, nombre de ports de lecture/ecriture 1 - technology 90 nm: surface 128.7 mm².

Capacity (bytes)	4194304
Line size (bytes)	32
Number of banks	1
Associativity	4
Tech node (micron)	0.09
Number of read ports	0
Number of write ports	0
Number of read/write ports	1
Number of output bits	256
Number of single-ended read ports	0
Change tag	No
Access mode	Serial

Ainsi comme décrit dans la figure suivante CACTI permet d'évaluer la surface du cache en fonction de la taille de cache et de l'associativité et l'on constate par exemple que pour une taille de cache donnée l'associativité augmente la surface.

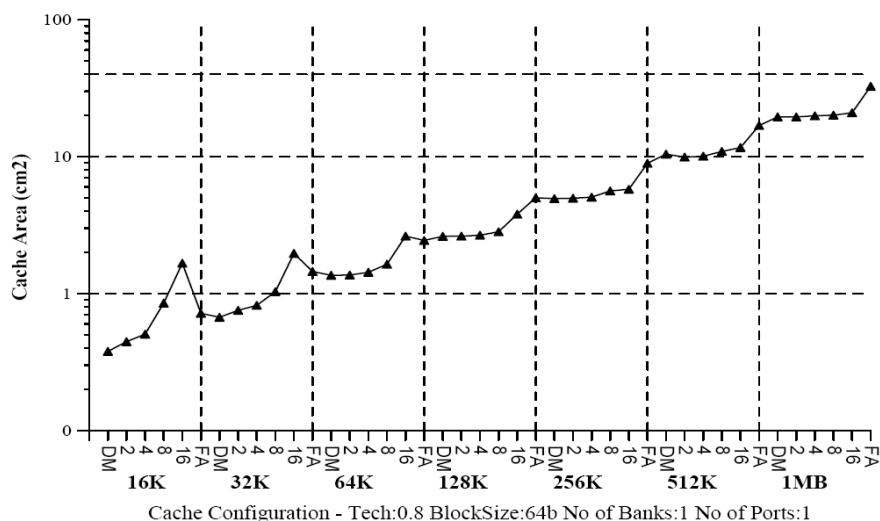


Figure 54.CACTI : évaluation de la surface en fonction de la taille de cache et de l'associativité

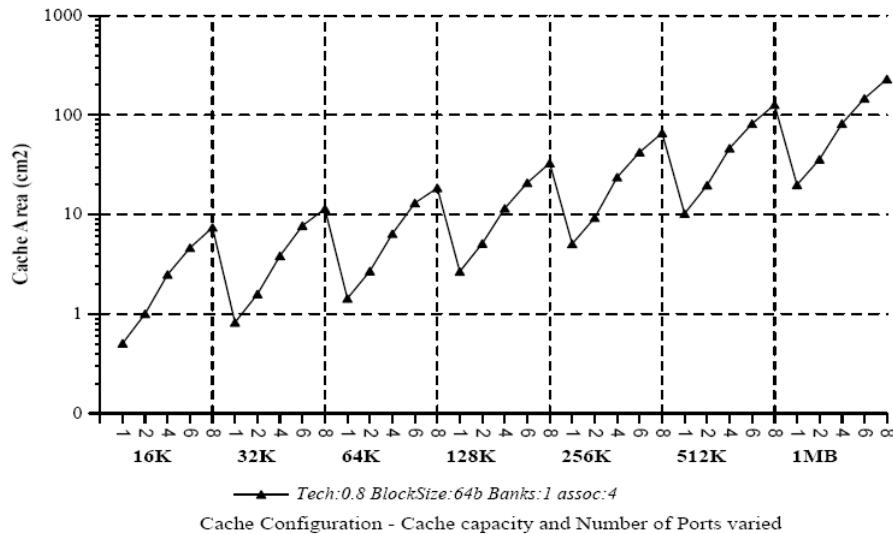


Figure 55.CACTI : évaluation de la surface en fonction de la taille de cache et du nombre de ports

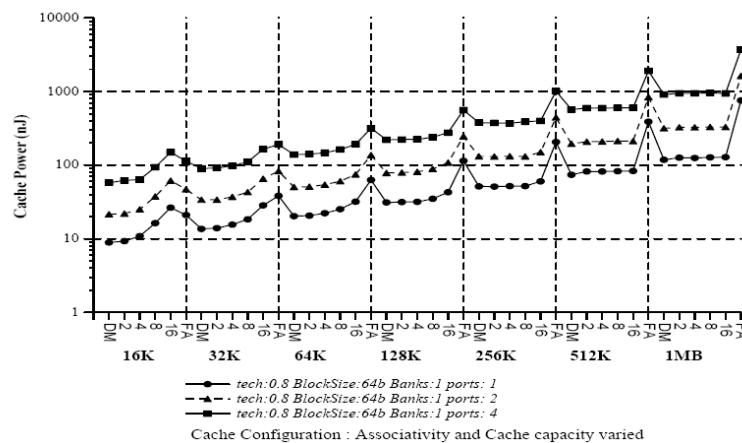


Figure 56.CACTI : évaluation de la consommation en fonction de la taille de cache, de l'associativité et du nombre de ports (1, 2, 4)

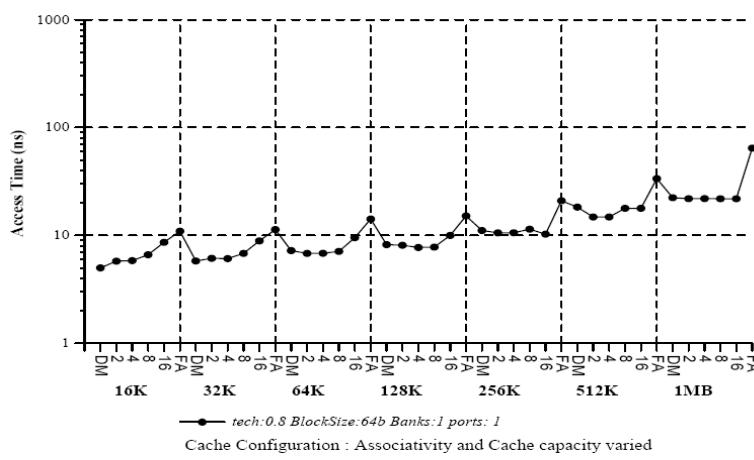


Figure 57.CACTI : évaluation du temps d'accès en fonction de la taille de cache et de l'associativité

CACTI permet d'évaluer le temps d'accès en fonction de la taille du cache et de l'associativité et la figure 59 décrit cette relation. Ainsi augmenter l'associativité d'un cache à taille donnée permet d'améliorer le hit rate mais au détriment de la surface silicium (donc le cout) et du temps d'accès au cache (donc de la fréquence du microprocesseur). La surface des mémoires caches est devenue majoritaire dans les circuits processeurs comme en témoignent les différentes photos suivantes de circuits.

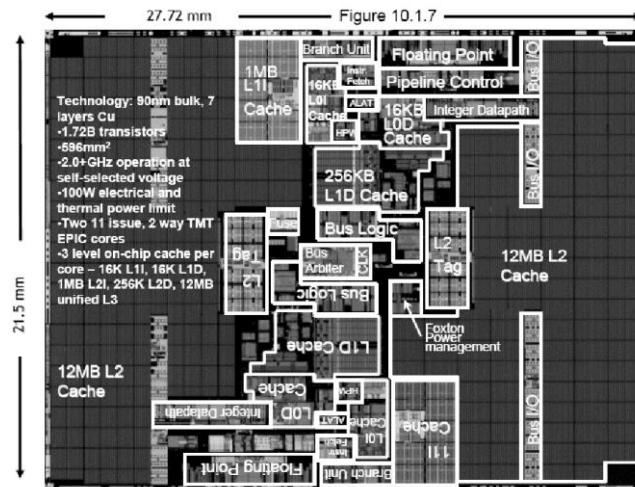


Figure 58.Exemple de circuit avec cache 16KL1I, 16K L1D, 1MB L2I, 256 K L2D, 12MB L3

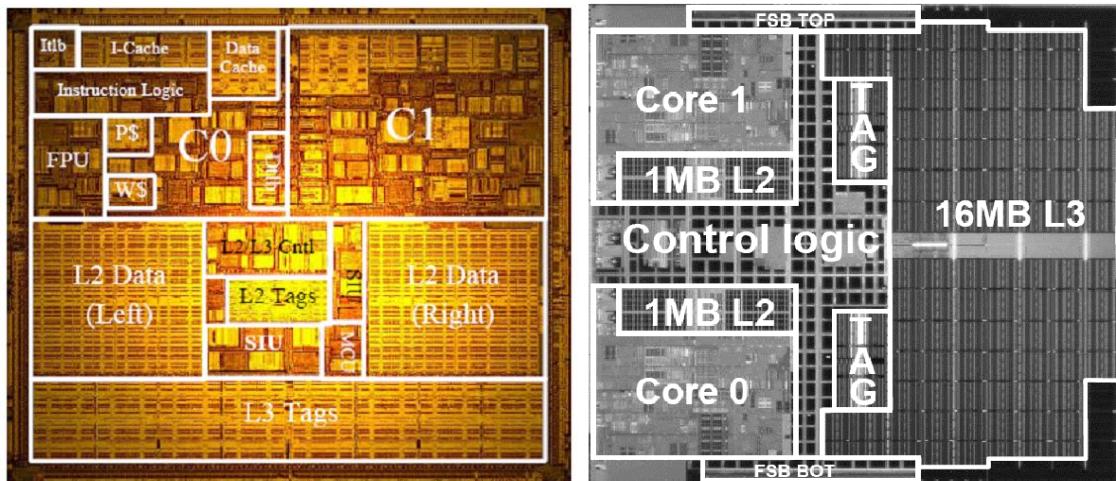


Figure 59.Exemple de circuits avec caches

L'optimisation de la hiérarchie mémoire reste un problème très complexe et multiobjectif. Cette optimisation dépend de nombreux facteurs de la microarchitecture du processeur et donc doit être évalué dans ce contexte.

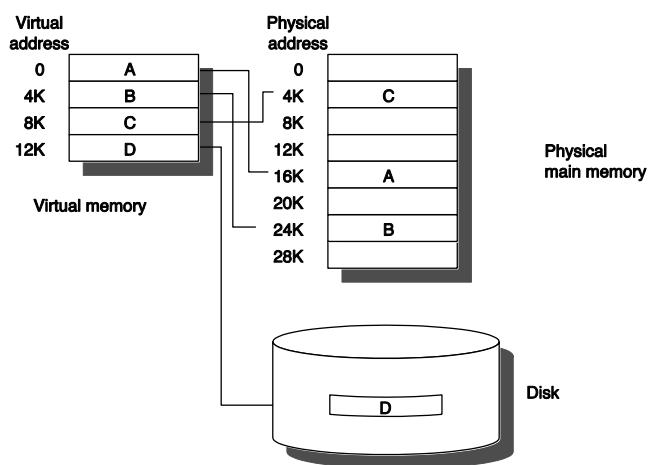
3.5 Mémoire Virtuelle

Le but ici n'est pas de présenter de manière approfondie la mémoire virtuelle qui est généralement abordée en cours de Systèmes Opératoires mais plutôt de décrire très brièvement le support matériel associé dans les microprocesseurs en particulier pour le TLB.

La capacité relativement limitée de la mémoire physique oblige à une gestion de celle-ci durant l'exécution d'un programme. Un programme dont les besoins en mémoire excèdent cette capacité physique ne peut être chargé en mémoire en un seul transfert et devra faire l'objet de multiples transferts de taille moindre en opérant des remplacements. Le problème est globalement analogue à celui rencontré entre les caches et la mémoire centrale.

La mémoire virtuelle a pour objectif principal de proposer des mécanismes pour gérer ce problème de manière à minimiser les transferts de données entre la mémoire centrale et le disque. Cela débute par l'affectation à chaque programme d'un espace d'adressage propre virtuelle qui correspond à l'espace maximum pouvant être adressé par un microprocesseur (ex : $2^{64} - 1$). Dans cet espace d'adressage les données sont accédées par une adresse virtuelle à qui l'on fera correspondre une adresse physique dans l'espace physique réel de la machine. Ce mécanisme est le mécanisme de traduction d'adresse (address translation). L'adresse virtuelle est décomposée en un numéro de page virtuel et un déplacement dans la page. Une page est un bloc de données contigus dans l'espace d'adressage. Cette page a des tailles variables suivant les systèmes opératoires et peut varier de 4K octets à 64K octets. Une donnée se trouve donc obligatoirement dans une page virtuelle unique et on lui fera correspondre une page physique unique. L'accès à la donnée dans la page se fera par un déplacement dans cette même page.

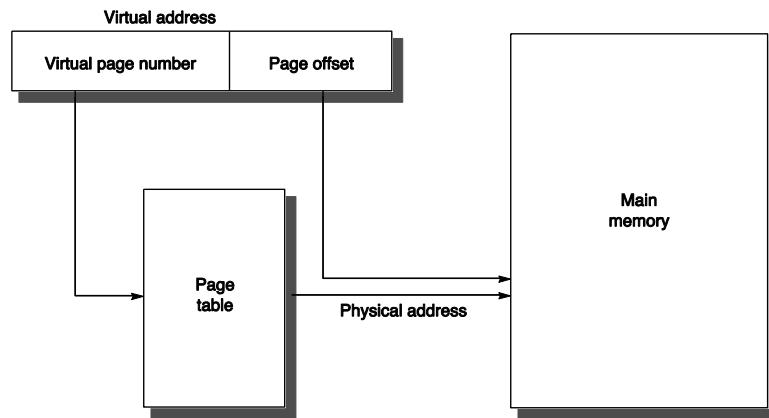
La figure suivante montre des pages contigus dans l'espace d'adressage virtuel mais qui dans l'espace physique représentés par la mémoire physique et le disque sont séparés.



© 2003 Elsevier Science (USA). All rights reserved.

Figure 60. Mémoire Virtuelle

Ce mécanisme de découpage de l'espace d'adressage en pages amené à une décomposition de l'adresse virtuelle en un numéro d'adresse virtuelle (*VPN – Virtual Page Number*) et un déplacement dans la page (*page offset*). La taille de la page étant identiques dans les deux espaces le mécanisme de traduction d'adresses consistera donc à faire correspondre à un numéro de page virtuel un numéro de page physique. Ceci est implémenté par une table de page qui est indexée par un numéro de page virtuel et fait correspondre le numéro de page physique associé.



© 2003 Elsevier Science (USA). All rights reserved.

Figure 61. Mécanisme d'accès

L'adresse de cette table de page en mémoire est donnée par un registre spécialisé le registre de table de pages (*page table register*). Un exemple complet est donné dans la figure suivante sur la base d'une page de taille 4Koctets, d'un espace virtuel de 4 Goctets (2^{32}) et d'un espace physique de 1 Goctets (2^{30}).

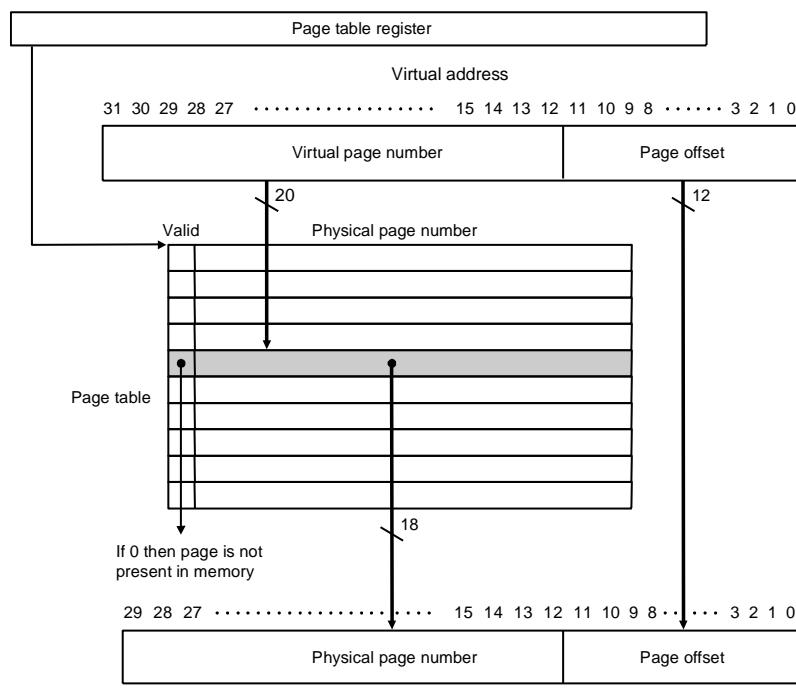
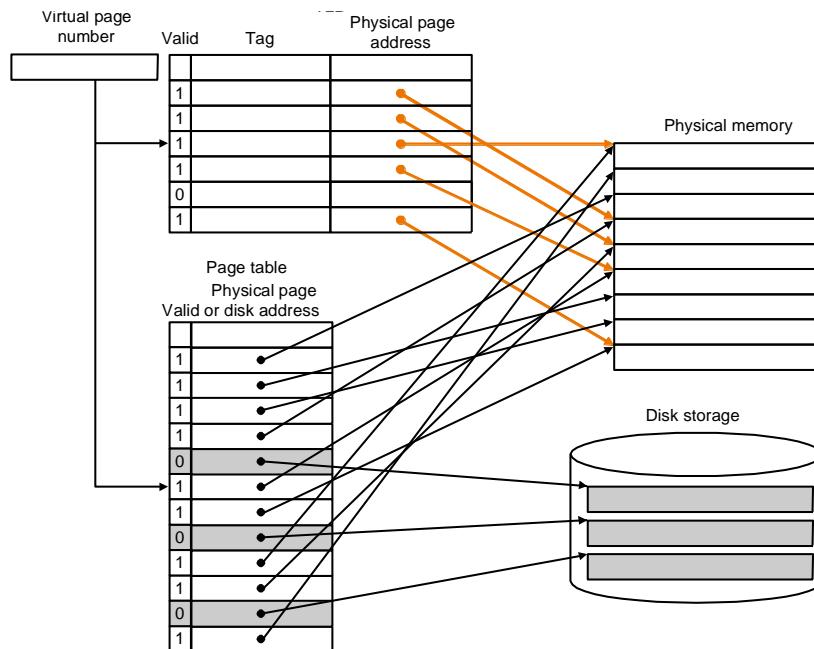


Figure 62. Traduction d'adresses

Dans la mesure où les tables de pages sont en mémoire un accès mémoire pour le microprocesseur se traduit par un double accès mémoire : (1) obtention de l'adresse physique et (2) obtention de la donnée. Ceci est trop pénalisant en termes de performances et il est souhaitable de trouver un mécanisme pour accélérer l'accès mémoire. Ce mécanisme est offert par le TLB (*Translation Lookaside Buffer*) qui est un cache de la table de pages.

**Figure 63.TLB**

Le TLB est intégré sur le microprocesseur et fait partie de la MMU (Memory Management Unit) l'unité en charge de la gestion mémoire. Il est aussi possible de séparer le TLB en TLB instruction et TLB donnée pour de nouveau exploiter les localités de références propres au code et aux données.

3.6 Mémoire DDR3, mode burst et défaut de cache

Clairement les mémoires caches contribuent de manière fondamentale à l'amélioration des performances des microprocesseurs par réduction du mur mémoire. Néanmoins les mémoires ont encore un rôle à jouer en réduisant le temps de pénalité d'un défaut de cache en fournissant de la manière la plus efficace et la plus rapide les blocs de donnée depuis la mémoire lors d'un défaut.

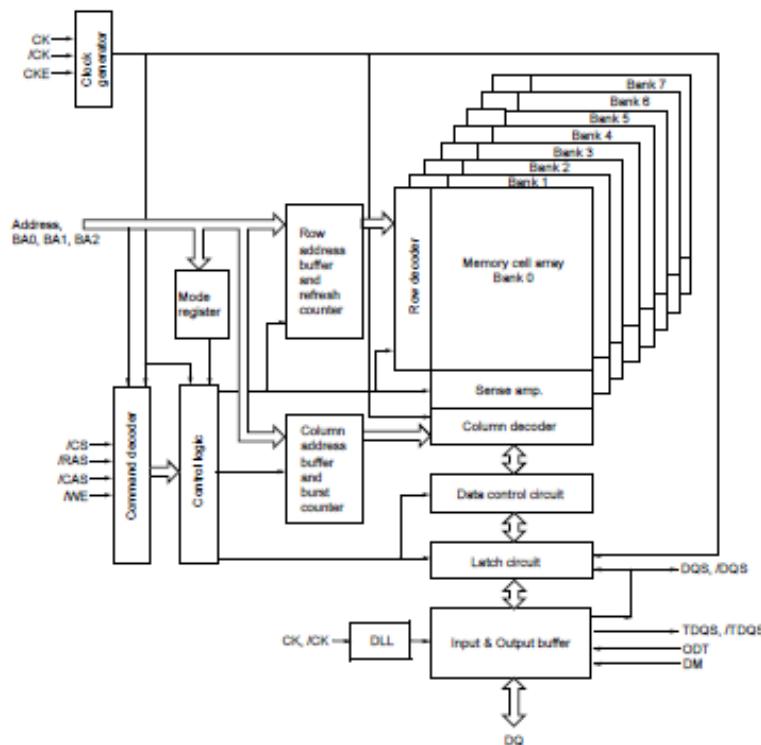
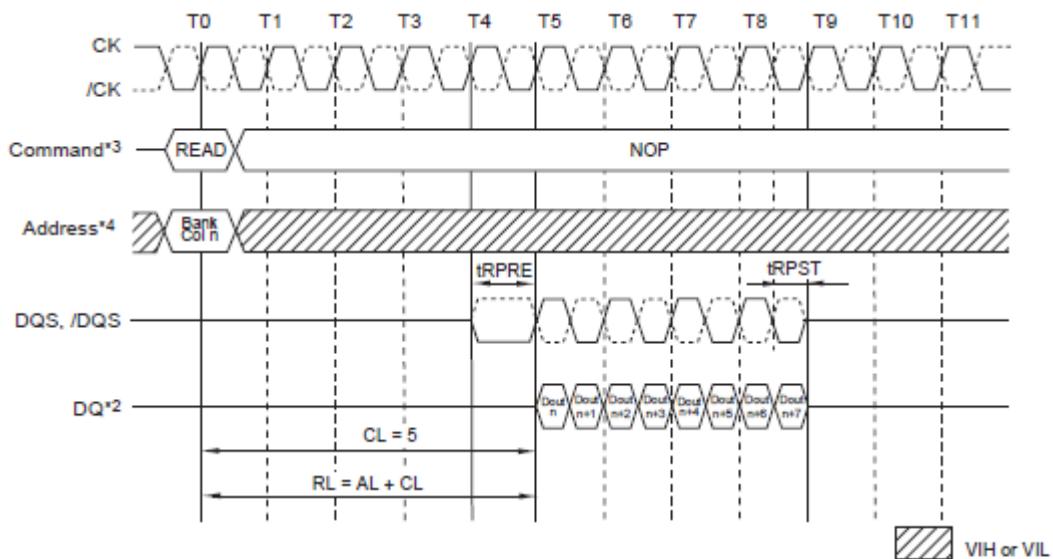


Figure 64. Architecture composant mémoire DDR3

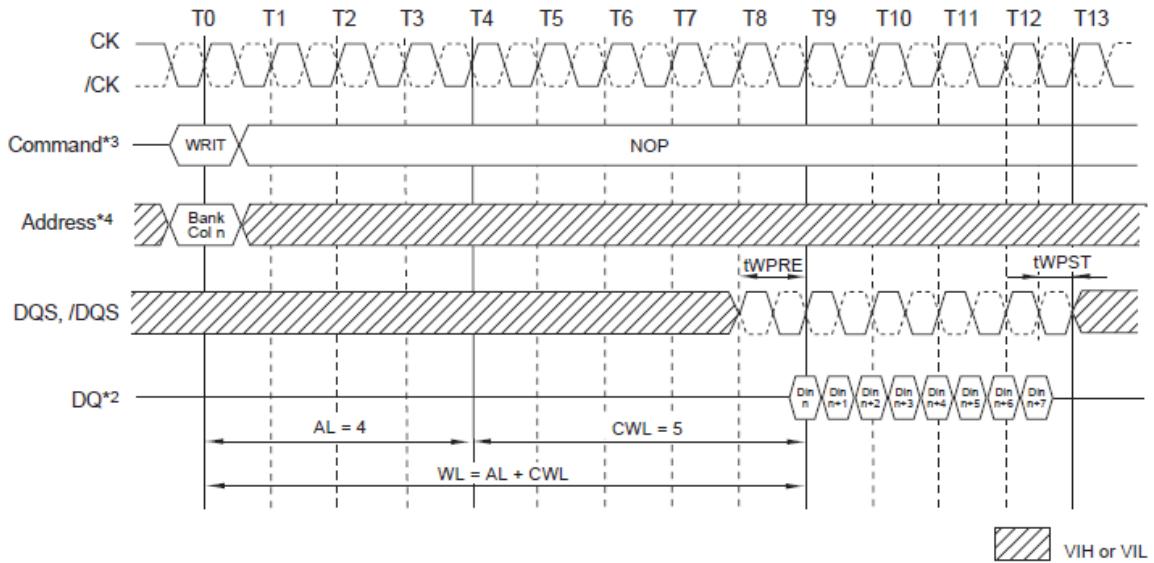
La manière la plus efficace est le mode burst. Le mode burst permet de pouvoir lire ou écrire une donnée en mémoire dans un mode optimisé qui évite de respecter le protocole entier pour chaque mot d'un bloc de donnée. Le protocole habituel nécessite l'activation d'un ensemble de signaux tout en respectant les contraintes temporelles de maintien de ces signaux. Le » mode burst permet de s'affranchir de l'activation de certains signaux sur la base de la localité. Les chronogrammes suivants montrent des exemples d'opérations de lecture et d'écriture en mode burst sur une DDR3.



- Notes:
1. BL8, AL = 0, RL = 5, CL = 5
 2. Dout n = data-out from column n .
 3. NOP commands are shown for ease of illustration; other commands may be valid at these times.
 4. BL8 setting activated by MRO bit [A1, A0] = [0, 0] or MRO bit [A1, A0] = [0, 1] and A12 = 1 during READ command at T0.

Burst Read Operation, RL = 5

Figure 65. Chronogramme opération de lecture mode burst DDR3



Notes:

1. BL8, WL = 9 (AL = (CL - 1), CL = 5, CWL = 5)
2. Din_n = data-in from column n .
3. NOP commands are shown for ease of illustration; other commands may be valid at these times.
4. BL8 setting activated by MRO bit $[A1, A0] = [0, 0]$ or MRO bit $[A1, A0] = [0, 1]$ and $A12 = 1$ during WRIT command at T0.

Burst Write Operation, WL = 9

Figure 66. Chronogramme opération d'écriture DDR3

4 . Chapitre 4 Structure Parallèle: Pipeline et Superscalaire

4.1 Comment accroître les performances ?

Les processeurs monocycle et multicycle ne permettent pas d'augmenter les performances sans modifier le modèle de base de traitement des instructions. Ce modèle de base est toujours le même :

1. Lire une instruction
2. Décoder l'instruction/lire les opérandes
3. Exécuter l'instruction
4. Ecrire le résultat dans les registres/en mémoire

Une approche naturelle à l'accroissement des performances est de se poser la question suivante :

Peut-on traiter plus d'une instruction en parallèle ?

Le premier des parallélismes est un parallélisme de traitement partiel. Ce parallélisme partiel est un recouvrement dans le temps des traitements des instructions i et $i+1$. Ainsi on peut décider de commencer à lire l'instruction suivante $i+1$ non pas lorsque l'instruction i a terminé son exécution comme c'était le cas pour le processeur monocycle et multicycle mais dès que l'instruction i est dans l'étape de décodage. En poursuivant par récurrence on aboutit à la situation où l'instruction i se trouverait dans l'étape 4, l'instruction $i+1$ dans l'étape 3, l'instruction $i+2$ dans l'étape $i+3$ et enfin l'instruction $i+4$ dans l'étape 1. Cette situation dans laquelle nous traitons plusieurs instructions simultanément mais chacune à une étape distincte du traitement s'appelle le traitement en mode pipeline.

La figure suivante montre le traitement du code suivant :

Iw \$1, 100(\$0)

Iw \$2,200(\$0)

Iw \$3,300(\$0)

dans le cas séquentiel classique et dans le cas d'un traitement pipeline. L'axe des X dans la figure représente le temps et l'axe des Y les instructions dans l'ordre séquentiel du programme.

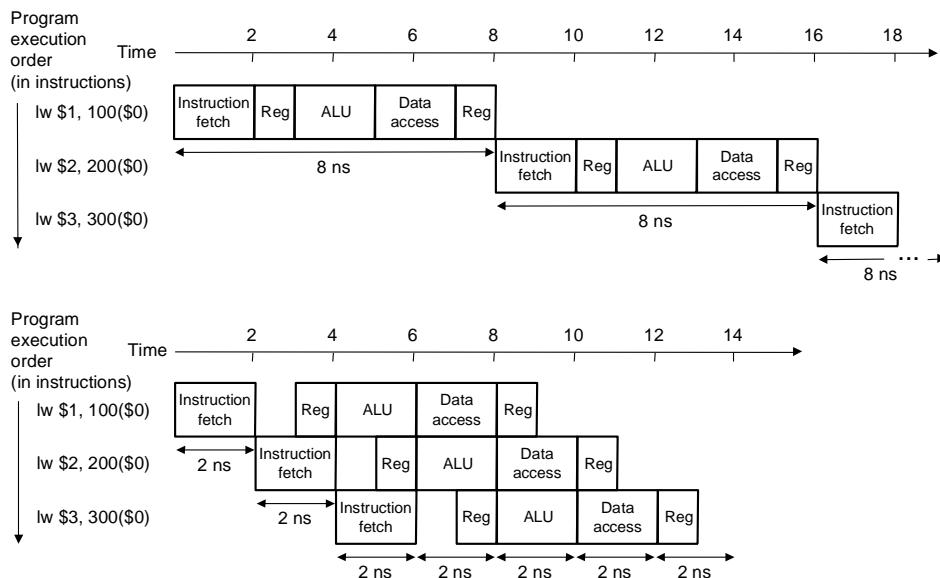


Figure 67.Exécution Séquentielle vs Exécution Pipeline

Comme on peut le voir dans le deuxième cas il existe un recouvrement des durées d'exécution qui constitue un parallélisme partiel et donc une réduction du temps d'exécution. Ce gain de performance à travers le mode pipeline est le mode d'exécution prévalant sur tous les processeurs actuels. Le pipelining est une méthode de traitement des instructions qui améliore les performances en accroissant le débit des instructions par opposition à une amélioration grâce à une réduction du temps d'exécution d'une instruction.

Il apparaît clairement que ce mode de traitement à la chaîne ne peut être efficace qu'en régime de croisière c'est-à-dire lorsqu'il n'existe pas d'interruptions de ce traitement. Il existe trois situations dans lesquelles une interruption du flux continu du traitement pourrait être possible et qui sont dues à des aléas (hazards) de nature distinctes :

- Aléas de structure (structural hazards)
- Aléas de contrôle (control hazards)
- Aléas de donnée (data hazards)

Ces trois situations expriment une dépendance entre deux ou plusieurs instructions distinctes situées à différentes étapes de traitement pour des raisons de conflit d'accès à une ressource matérielle (aléas de structure), parce qu'une instruction typiquement de branchement va éventuellement changer le flux des instructions (aléas de contrôle) et enfin à cause d'une relation de type producteur-consommateur sur une donnée particulière (aléas de donnée). Dans le premier cas la solution simple consiste à dupliquer la ressource ou à étendre l'accès (augmenter le nombre de ports de lectures/écritures sur un banc de registres par exemple). Dans le second cas le problème se pose à cause des instructions de branchements conditionnels. En effet lors du traitement d'une instruction de branchement conditionnel (ex : beq \$1,\$2,100) nous ne pourrons savoir si le branchement sera pris que lorsque l'on aura testé la condition (ex. \$1 = \$2) c'est-à-dire après lecture des registres c'est-à-dire dans l'étape d'exécution. Or le mode d'exécution pipeline fait que dans cette étape les deux instructions i+1 et i+2 qui suivent beq dans le flux des instructions se trouvent respectivement dans l'étape de décodage et dans l'étape de lecture de l'instruction. L'exécution complète de ces deux instructions dépend de l'instruction beq et donc constitue un aléa de contrôle.

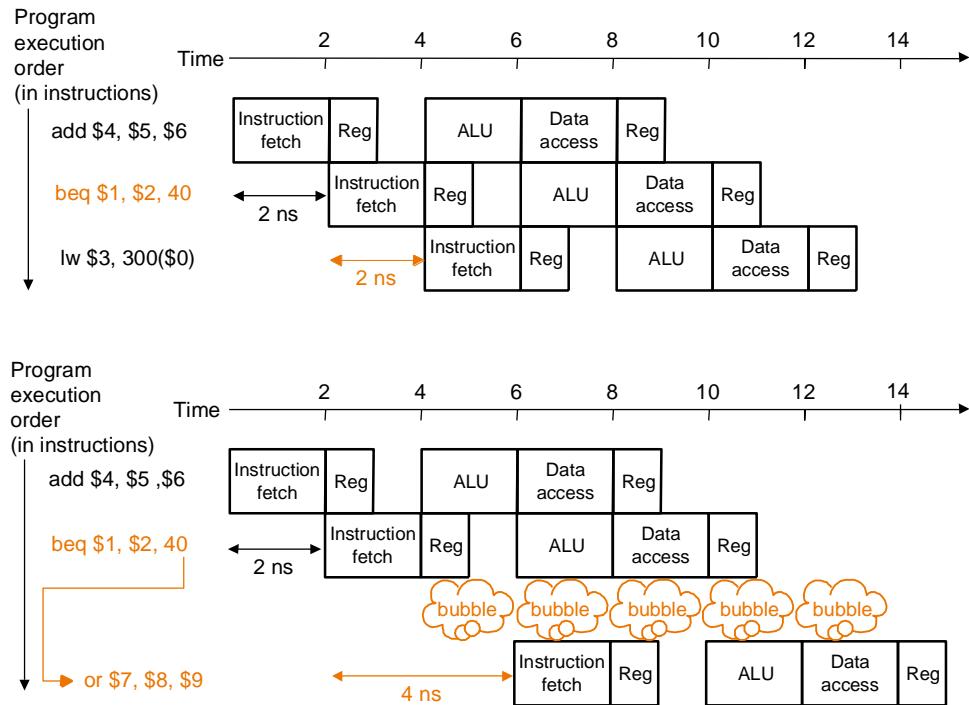


Figure 68.Cas d'un branchement conditionnel : (a) condition non vérifiée (b) condition vérifiée

Une proposition de solution simple à ce problème consiste tout simplement à considérer que systématiquement l'instruction après le branchement sera exécutée. Ce principe appelle delayed branch n'est efficace que si l'on peut toujours placer une instruction utile à exécuter après le branchement. Dans l'exemple de la figure précédente l'instruction naturellement candidate est l'instruction add \$4,\$5,\$6 située avant le branchement qui devra nécessairement être exécutée. C'est le premier cas d'amélioration de performances qui s'appuie sur une exécution qui ne soit pas dans l'ordre du programme. Ce réordonnancement des instructions dans ce cas se fait de manière statique à la compilation par le compilateur.

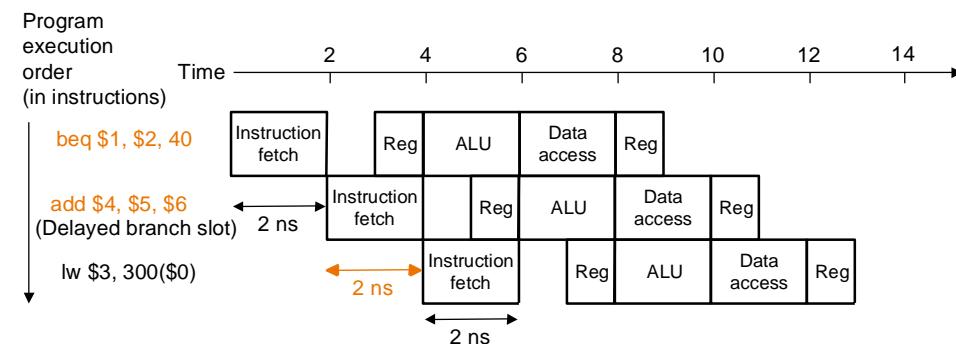


Figure 69. Branchement retardé (delayed branch)

Ce réordonnancement n'est pas toujours possible et lorsqu'il ne l'est pas le compilateur introduit des instructions NOP (no operation) ne faisant rien d'autre que de retarder le flot des instructions suivantes en attendant la résolution de la condition de branchement. Le troisième aléas existe lorsque entre deux instructions existe une relation de producteur-consommateur. L'instruction consommatrice ne pouvant s'exécuter que lorsque l'instruction productrice aura produit et transmit la donnée.

Par exemple les deux instructions suivantes :

```
add $s0, $t0, $t1
sub $t2,$s0,$t3
```

sont dépendantes sur \$s0 la première produisant le résultat tandis que la seconde l'utilise. Dans le mode d'exécution séquentielle monocycle et multicycle l'instruction sub \$t2,\$s0,\$t3 ne lit le registre \$s0 dans le banc de registres qu'après que l'instruction add \$s0, \$t0, \$t1 eut écrit le résultat de l'addition effectuée dans l'ALU dans le banc de registres. Dans une exécution en mode pipeline l'instruction sub \$t2,\$s0,\$t3 se trouve dans l'étape de lecture des opérandes du banc de registres lorsque l'instruction add \$s0, \$t0, \$t1 se trouve dans l'étape d'exécution. Or le mode d'exécution adopté jusqu'à présent exige que les opérandes ne peuvent être lues que du banc de registres. Ce qui signifie dans le cas présent que l'instruction doit attendre l'écriture dans le banc de registres de \$s0 pour pouvoir poursuivre son exécution. Cette situation de dépendance de données est un aléas de donnée et oblige à un arrêt du pipeline et donc à une réduction de performances. Une solution consiste à transmettre la donnée directement après l'obtention du résultat en sortie de l'ALU et sans attendre l'écriture dans le banc de registres. Cette transmission (forwarding) est illustrée dans la figure suivante qui reprend le traitement d'une instruction en 5 étapes : (1) IF : Instruction fetch (lecture d'instruction) (2) ID : Instruction decode (décodage d'instructions) (3) EX : Execution d'instructions (4) MEM : memory access (accès mémoire pour les instructions lw/sw) et (5) WB : Write Back écriture des données dans le banc de registres.

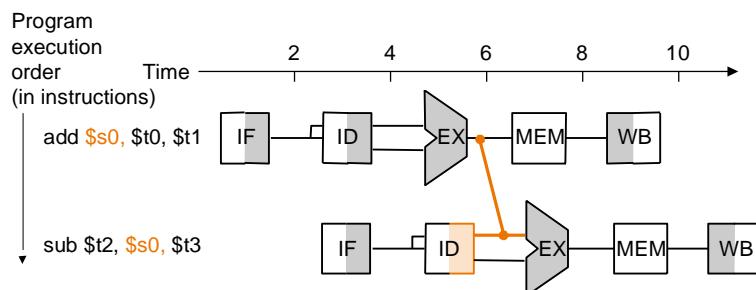


Figure 70. Principe du Forward

Malheureusement ce forward n'est possible que parce que la consommation se fait directement après la production ce qui n'est pas le cas pour les instructions produisant un résultat tardivement dans les étapes de traitement. Ceci est le cas des instructions de lecture qui produisent leurs résultats dans la dernière étape du traitement après lecture mémoire. Une instruction se situant tout de suite après une opération de lecture ne peut s'appuyer sur le mécanisme de forward car au moment où l'instruction de lecture se trouve en position productrice l'instruction suivante est sensée déjà avoir consommé la donnée et l'avoir utilisée comme opérande dans un calcul. Dans ce cas et dans le mode pipeline statique (ordonnancement statique des instructions) il n'existe pas d'autres solutions que d'interrompre le flot des instructions en attente de la donnée.

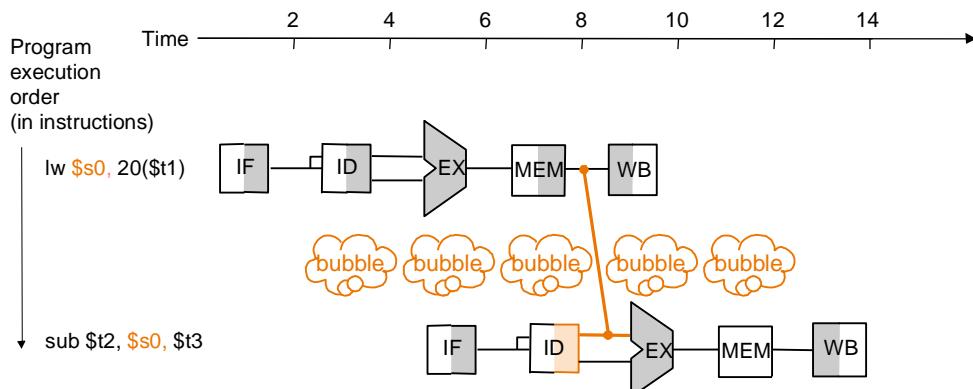


Figure 71. Dépendance entre instruction de lecture mémoire et instruction de type R

4.2 Chemin de donnée pipeliné

Le mode d'exécution pipeliné exige que plusieurs instructions peuvent être simultanément dans le chemin de donnée à des étapes de progression différentes. Ceci n'est possible que si l'on délimite clairement (spatialement) dans le chemin de donnée les étapes de traitement et les ressources qui leur sont affectées et que ce découpage permet une autonomie maximale de l'instruction durant cette étape de traitement. Le découpage adopté dans le cas du processeur MIPS est un découpage qui correspond aux 5 étapes de traitement des instructions : (1) Etape 1 : IF Instruction fetch (lecture instruction) (2) Etape 2 : ID Instruction decode/register file read (décodage instruction/lecture des registres) (3) Etape 3 : EX Execute/address calculation (Exécution/calcul des adresses) (4) Etape 4 : MEM Memory access (accès mémoire) (5) WB : Write Back (Écriture des résultats dans le banc de registres). Un découpage virtuel du chemin correspondant à ces 5 étapes est donné dans la figure suivante.

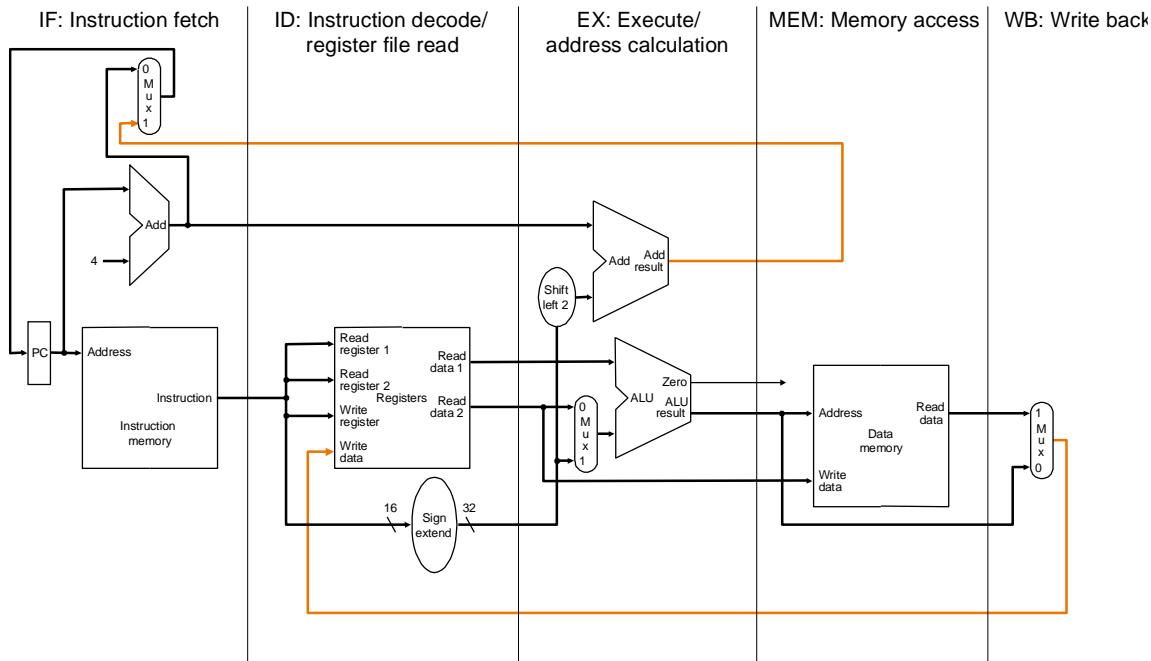


Figure 72. vDécoupage préliminaire du chemin de donnée

Le découpage physique effectif se fait à travers des registres connectés à une horloge commune et définissent ainsi un pipeline synchrone. Les registres étant placés en sortie des circuits nécessaires à chaque étape jouent un double rôle : (1) empêcher la propagation des signaux entre étapes tant qu'un front d'horloge n'a pas validé le transfert (2) dès qu'un front d'horloge a validé le transfert, mémoriser les résultats produits par ces circuits et propager les valeurs dans les circuits de l'étape suivante. Le chemin de donnée avec les registres est donné dans la figure suivante.

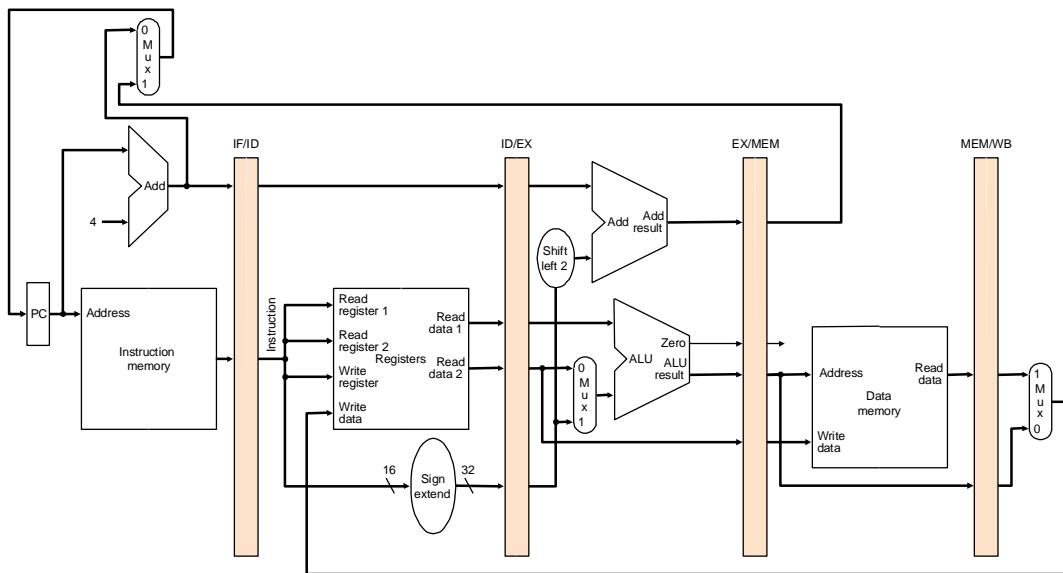


Figure 73. Découpage du Chemin de donnée par introduction de registres

Dans ce premier découpage il est néanmoins nécessaire de vérifier que les instructions puissent effectivement véhiculer au cours de leur progression l'ensemble des paramètres nécessaire à leurs exécutions. Ainsi dans les processeurs monocycle et multicycle le numéro du registre destination était directement fourni par l'instruction et pouvait être connecté à ce niveau au port d'écriture du banc de registres comme indiqué dans la figure précédente. Ceci n'est plus correct dans un processeur pipeline qui par son découpage physique du chemin de donnée et sa simultanéité de traitement des instructions fait que l'instruction fournissant le numéro de registre destination (étage ID) n'est pas celle voulant écrire sa donnée dans le banc de registres (étage WB). La correction du chemin de donnée impose donc de faire propager avec l'instruction le numéro de registre destination et ce comme indiqué dans la figure suivante.

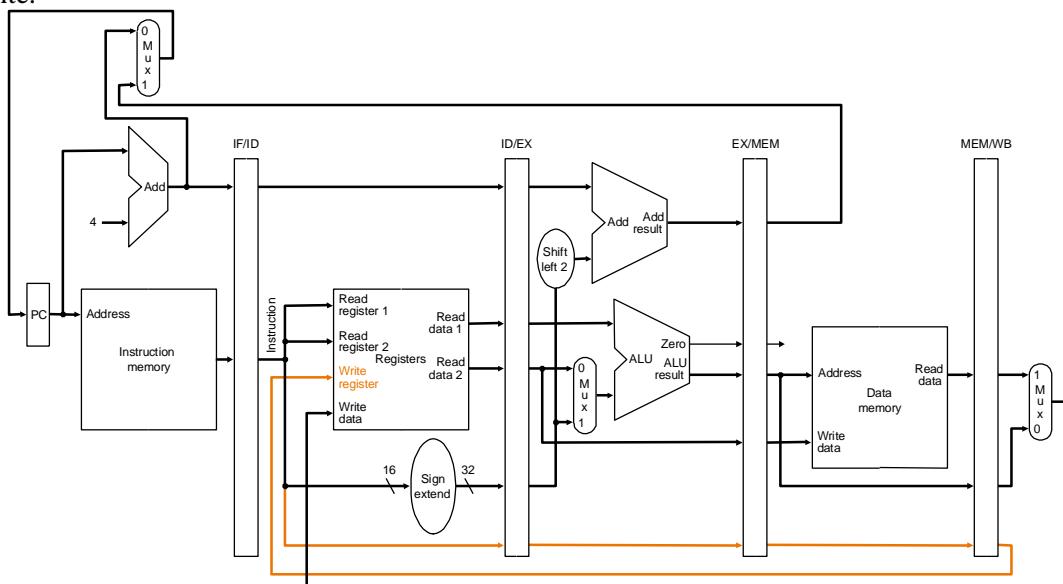


Figure 74. Modification du chemin de donnée : prise en compte du registre destination

Ce principe se retrouvera lors de la conception de l'unité de contrôle. Il reste à compléter le chemin de donnée avec un multiplexeur pour le registre destination différent suivant les formats I et R et rajouter le signal d'indication de branchement pour la sélection de la prochaine valeur du PC. Cette sélection est la conjonction d'un signal en sortie de l'ALU indiquant que le résultat est égal à 0 (dans un

branchement conditionnel la comparaison est effectuée par soustraction, une valeur 0 indiquant l'égalité et le fait que l'instruction décodée est un branchement.

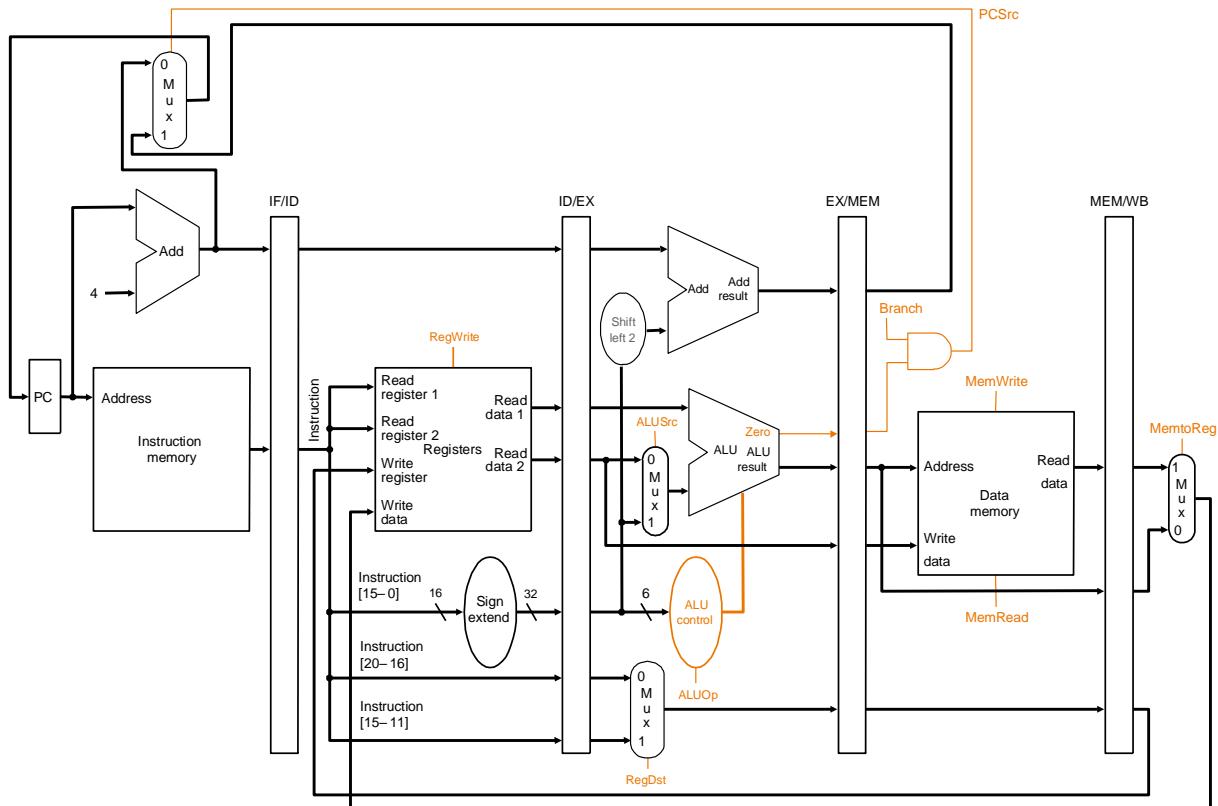


Figure 75. Introduction du support pour beq/alu control

4.3 Contrôle Pipeliné

Une unité de contrôle prend en entrée une instruction et génère les signaux de contrôle associés. Comment traiter plusieurs instructions simultanément et conserver leurs signaux de contrôle jusqu'à leurs terminaisons ? Reprenant le principe de la propagation avec l'instruction de ses signaux associés

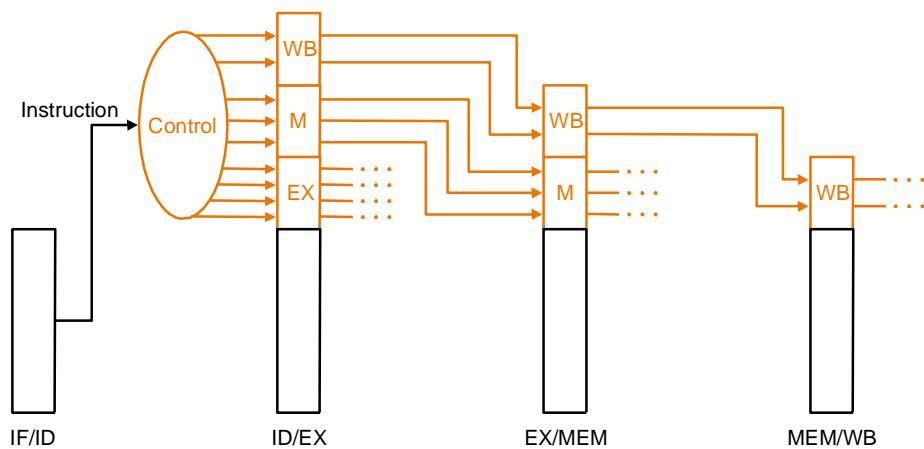


Figure 76. Transmission des signaux de contrôle

l'unité de contrôle générera tous les signaux de contrôle nécessaires à l'exécution de l'instruction et les laissera se propager avec l'instruction. Pour cela on rajoute des registres supplémentaires aux registres

existants utilisés pour le découpage du chemin de donnée et ce pour la mémorisation des signaux de contrôle. Néanmoins sachant que les signaux de contrôle seront progressivement consommés au cours des étapes subséquentes et si l'on regroupe les signaux par étapes (EX, MEM, WB) cette mémorisation se réduira au fur et à mesure de la progression de l'exécution enlevant progressivement un groupe de signaux et donc la mémorisation associée après chaque étape. Le chemin de donnée avec son contrôle associé est donné dans la figure suivante.

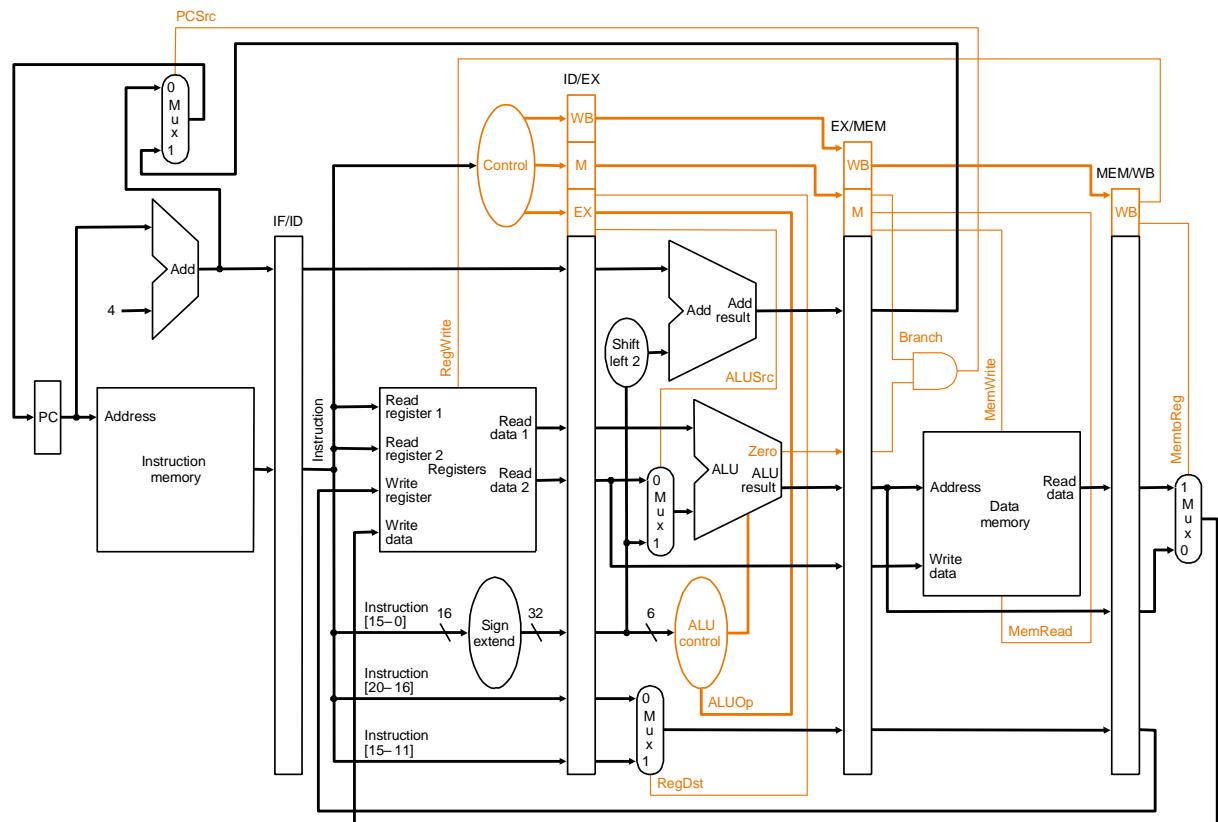


Figure 77. Processeur Pipeline v1

Ce processeur pipeline dans sa version 1 n'a aucun support pour le traitement des aléas de donnée. Il impose toujours d'attendre qu'une instruction écrive son résultat dans le banc de registres pour pouvoir lire la donnée. Etant donné la fréquence des dépendances dans un programme quelconque le mode pipeline dans l'état est loin de pouvoir être dans un régime continu. Il est nécessaire d'introduire des mécanismes dans le chemin de donnée pour répondre à cela.

4.4 Aléas

L'aléas de donnée peu être résolu par un mécanisme de transmission de données entre instructions successives. Ce mécanisme de transmission a pour objectifs : (1) de détecter une dépendance de données entre instructions (producteur-consommateur) (2) de transmettre la donnée entre les étages de traitement concernés.

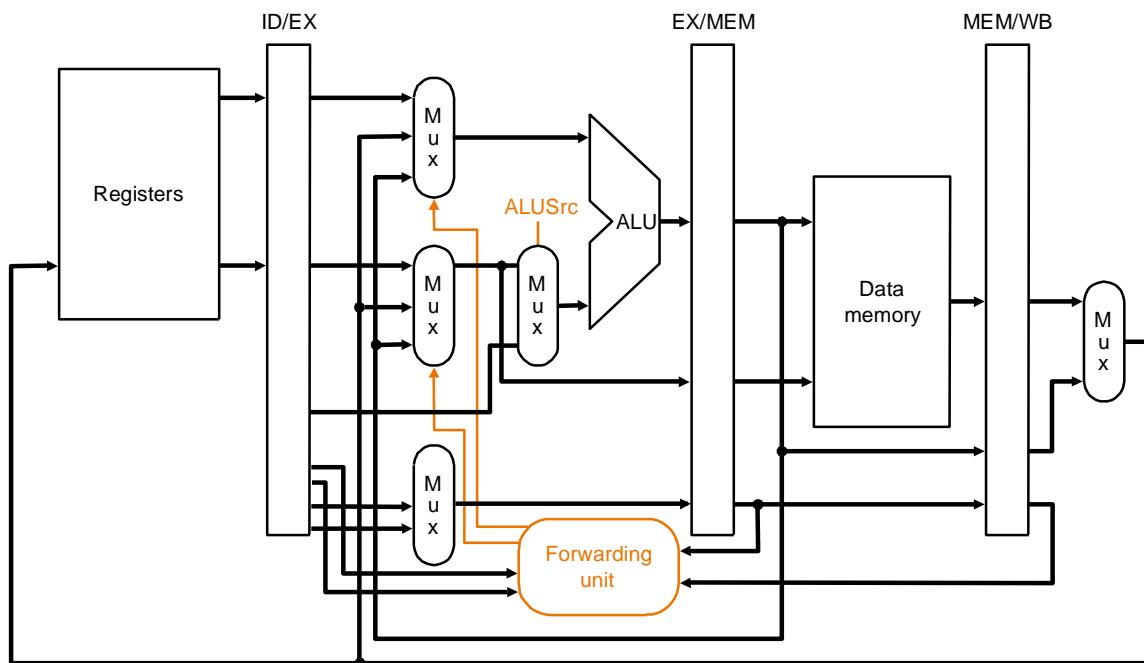
La détection se fait tout simplement par comparaison des numéros de registre sources de l'instruction en entrée de l'étage EX et les numéros des registres destinations des instructions dans les étages MEM et WB. La table suivante qui rappelle les formats des instructions montre que l'utilisation des champs des instructions peut être utilisé à cet effet.

Tableau 25.Formats des Instructions Assembleur MIPS

Fields							comments
name	Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
instruction		31-26	25-21	20-16	15-11	10-6	5-0
							All MIPS instructions 32 bits Instruction 32 bits Instruction[31:0]
R-format	op	rs	rt	rd	shamt	Funct	Arithmetic instruction format
I-format	op	rs	rt	Address/immediate			Transfer, branch, imm, format
J-format	op	Target address					Jump instruction format

Cette détection doit donc amener à une action de transmission des données. Pour cela et sachant que la donnée à transmettre doit être directement réinjectée en entrée de l' ALU et ce dans éventuellement n'importe laquelle des entrées il faut rajouter : (1) des multiplexeurs aux entrées de l' ALU (2) modifier le chemin de donnée en ajoutant des bus de données en sortie des étages MEM et WB et se connectant à ces multiplexeurs. L'unité de transmission de donnée (Forwarding unit) devra simplement en fonction de la combinaison de registres générer les signaux appropriés de sélection des multiplexeurs en entrée de l' ALU. Tout ceci se fera sans interrompre l'exécution dans le pipeline et donc de manière parallèle entre autres de l'écriture dans le banc de registre de la donnée par ailleurs directement réinjectée dans le calcul.

La position de l'unité de forward dans le chemin de donnée ainsi que les modifications apportées sont décrites dans la figure suivante.

**Figure 78.Unité de Forward**

Dans le cas d'une opération de lecture suivie d'une autre instruction il n'est malheureusement pas possible dans le cas d'un pipeline statique de s'appuyer sur le data forwarding unit pour résoudre la dépendance de données. La raison est que tout simplement l'instruction de lecture ne génère son

résultat qu'après lecture de la mémoire c'est à dire bien après que les instructions qui suivent en aient besoin. Il est alors nécessaire d'introduire une unité chargée de : (1) détecter ce type d'aléas (2) arrêter le pipeline en amont pour éviter une lecture incohérente du banc de registres. Cette unité appelle Hazard detection unit est placée à l'étage ID pour permettre une influence directe sur l'unité de contrôle. La détection est aisée et consiste à vérifier qu'il existe une dépendance entre l'instruction i et $i+1$ (par comparaison des registres destination et sources) et que l'instruction i est une instruction de lecture mémoire (par test de l'activité du signal de lecture mémoire). L'action associée consiste à arrêter le pipeline en amont ce qui est réalisé par (1) empêcher la progression (mise à jour) du PC et (2) la transmission des instructions vers l'aval (action sur les registres de séparation d'étages). L'autre aspect concerne l'aval où il est nécessaire de permettre la progression normale dans le pipeline de l'instruction de lecture. Néanmoins le pipeline devra traiter des instructions dans les étages suivants. Dans un pipeline à ordonnancement statique le choix se portera sur une instruction ne faisant rien NOP (no operation) déjà utilisée dans le cas de branchement retardé. Cette instruction NOP n'a pas à être présente dans le jeu d'instructions et est réalisée en mettant à 0 les valeurs des signaux de contrôle. Ceci se fait en rajoutant un multiplexeur dont l'une des entrées récupère l'ensemble habituel des signaux de contrôle des étages suivants (EX/MEM/WB) et l'autre entrée les mêmes signaux avec les valeurs 0. L'unité Hazard detection sélectionnera l'entrée avec valeurs égales à 0 lors d'aléas sinon les valeurs normales générées par l'unité de contrôle. Le processeur est donc complété par cette unité et est décrit dans la figure suivante.

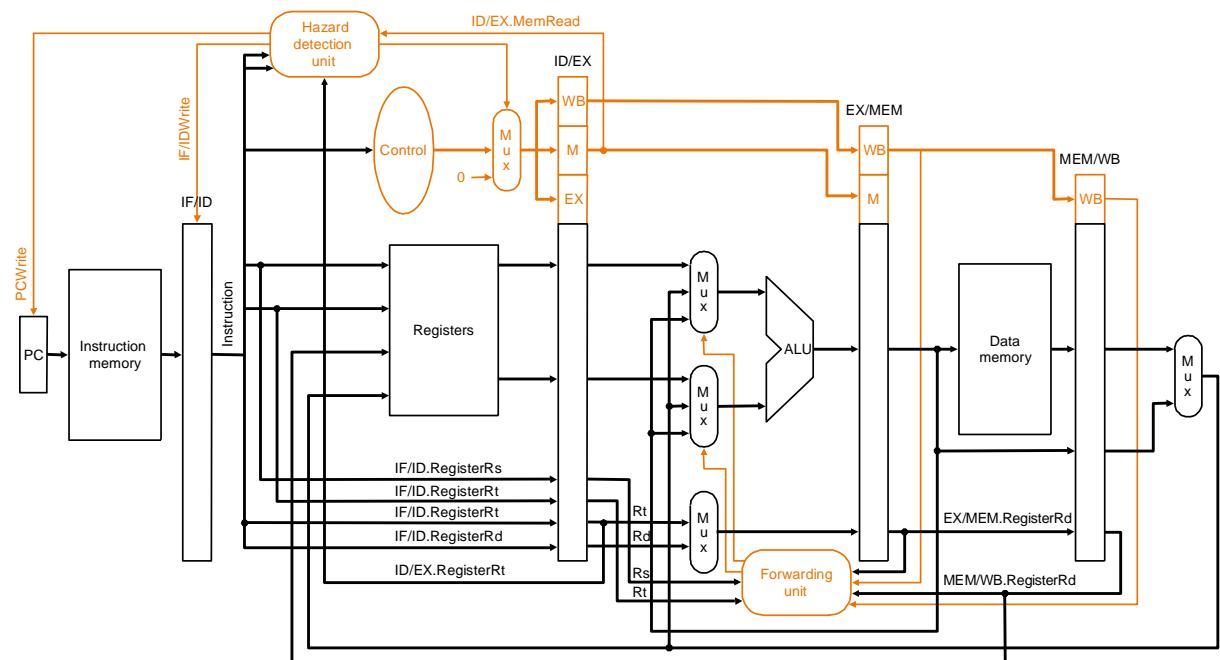


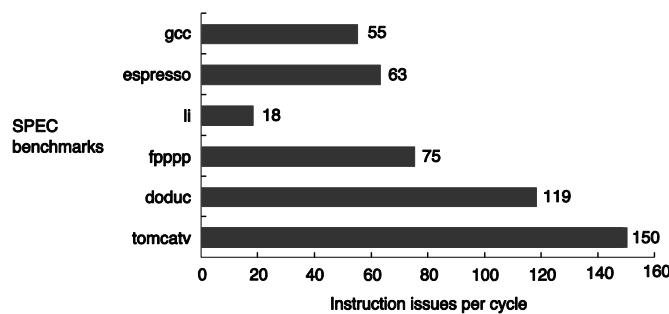
Figure 79. Unité de Détection d'Aléas (Hazard Detection Unit)

Le processeur ainsi obtenu peut être amélioré par l'ajout d'unités fonctionnelles supplémentaires de calcul entiers ou flottants tout en conservant à l'esprit que le mécanisme de transmission de donnée augmentera sa combinatoire mais aussi que l'inclusion d'unités fonctionnelles relativement lentes (ex.calcul flottant vs entier) réduira la fréquence d'horloge du pipeline sauf si ces unités sont elles mêmes pipelinees ce qui est systématiquement le cas des unités flottantes. Le processeur présenté ici n'a pas de structures de caches pour des raisons de simplicité de présentation. Il est absolument indispensable de remplacer les mémoires données et instruction de la figure par respectivement un cache donnée et un cache instruction. En effet, un processeur pipeline ne peut être efficace que si les étages du pipeline ont un temps de propagation quasiment identiques ce qui n'est pas le cas entre les étages IF et MEM et les autres étages. Nous avons déjà vu que les accès mémoires sont très lents et donc inclure un accès

mémoire dans un pipeline oblige le pipeline à avoir une fréquence d'horloge dominée par le temps d'accès à la mémoire. La solution passe par les caches beaucoup plus rapides. Auquel cas le processeur devra être complété par une interface mémoire permettant la communication entre les caches et la mémoire lors des transferts de données dus aux défauts de caches où à la stratégie de remplacement. Les mémoires caches trouvent donc un environnement naturel dans les structures pipelines même si parfois les mémoires caches sont elles mêmes pipelinees ce qui se traduit par deux étages pour IF, IF1 et IF2.

4.5 Processeur Superscalaire

Les pipelines statiques se privent d'un vivier d'instructions exécutables simplement parce que le mode d'exécution suit un modèle strictement séquentiel en d'autres termes un ordonnancement statique des instructions. Différentes études ont été réalisées pour quantifier le potentiel d'instructions exécutables en parallèle dans un parallélisme réel non partiel comme dans le cas pipeline. Pour cela ces études analysent des programmes types extraits de suite de benchmarks et effectuent une analyse sans contraintes de ressources et basée sur une connaissance a priori parfaite du flot des instructions. Dans ces conditions idéales le parallélisme instruction (Instruction Level parallelism) peut atteindre de très grandes valeurs en particulier sur les programmes scientifiques flottants. L'exemple extrême de la figure suivante est donné par le programme Tomcatv de SPEC CFP (www.spec.org) qui permettrait d'exécuter jusqu'à 150 instructions en parallèle.



© 2003 Elsevier Science (USA). All rights reserved.
Figure 80. Potentiel en Instructions exécutables en parallèle

Même un programme classique comme le compilateur gcc permettrait dans le cas idéal l'exécution d'une cinquantaine d'instructions en parallèle. Pour pouvoir exécuter en parallélisme réel plusieurs instructions il faudrait fournir les ressources en unités fonctionnelles permettant l'exécution simultanée de ces instructions. Les processeurs superscalaires sont des processeurs qui exécutent plusieurs instructions par cycle d'horloge et s'appuie sur le principe du pipeline pour le parallélisme temporel et sur la duplication multiple du chemin de donnée pour le parallélisme spatial. Pour une exploitation efficace des ressources les processeurs superscalaires sont associés avec la technique d'ordonnancement dynamique des instructions qui permet au processeur d'exécuter les instructions pratiquement en flot de donnée c'est-à-dire dès qu'elles sont prêtes à être exécuter pour éviter les blocages dus aux aléas. Le principe de l'ordonnancement dynamique peut s'implémenter par des stations de réservations associées à chaque unité fonctionnelle et qui jouent le rôle de file d'attente. Dès qu'une instruction a été décodée elle est envoyée vers une des unités fonctionnelles pouvant l'exécuter et s'exécute dès que possible sinon elle se met en attente soit de ses opérandes soit de l'unité fonctionnelle. Un processeur superscalaire avec reordonnancement dynamique peut être vu comme un système à trois niveaux ou le premier consiste à lire et décoder des instructions dans l'ordre séquentiel du programme, le deuxième consiste en une partie exécution qui s'effectue dans l'ordre d'exécution permettant le minimum de blocages qui n'est pas l'ordre initial mais qui respecte la sémantique du programme (out of order exécution) et finalement un dernier niveau qui rétablit

l'ordre initial et valide l'exécution. Ce mode de fonctionnement est supérieur au pipeline statique dans le sens où il affaiblit les contraintes de dépendances ainsi que le mode d'exécution d'une instruction par cycle mais ne s'affranchit pas de l'aléas de contrôle. Les instructions de branchements ont un effet encore plus importants dans une structure comme celle des superscalaires. Une solution à ce problème passe par la prédition des branchements. Le principe de base consiste à effectuer une prédition de branchement pour éviter d'attendre l'évaluation de la condition de branchement et connaître la direction du flot des instructions donc des prochaines instructions à lire en mémoire. La prédition de branchement est effectuée par une unité supplémentaire dédiée à la prédition de branchement et située au niveau des étapes de lecture instructions pour influencer la mise à jour du PC et donc des prochaines instructions à lire. Une structure d'architecture superscalaire avec reordonnancement dynamique et prédition de branchement est décrite dans la figure suivante.

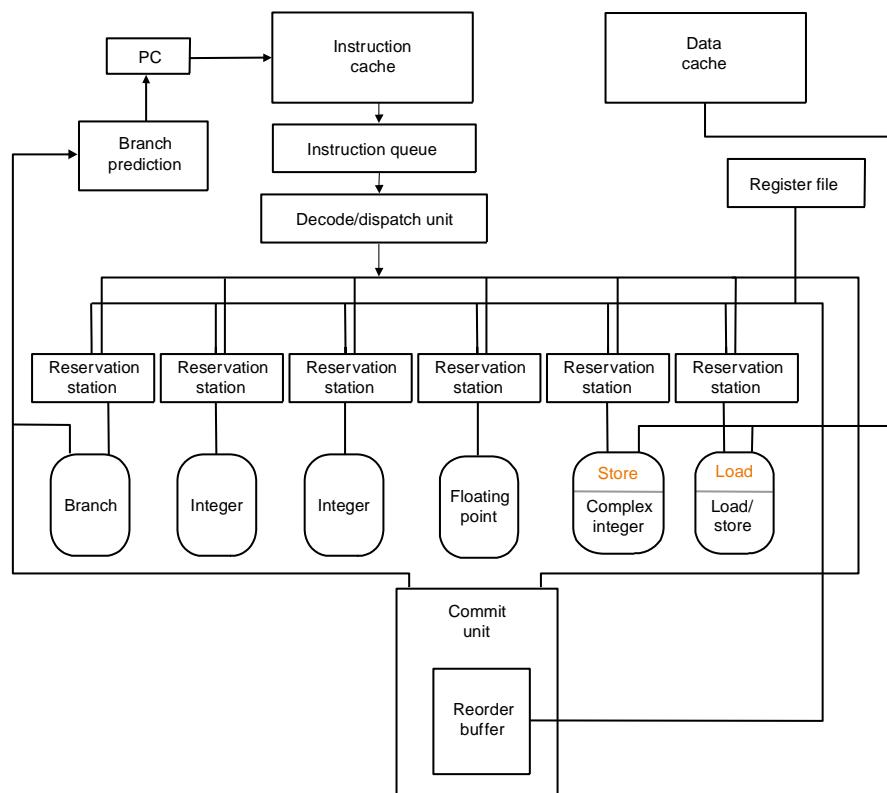
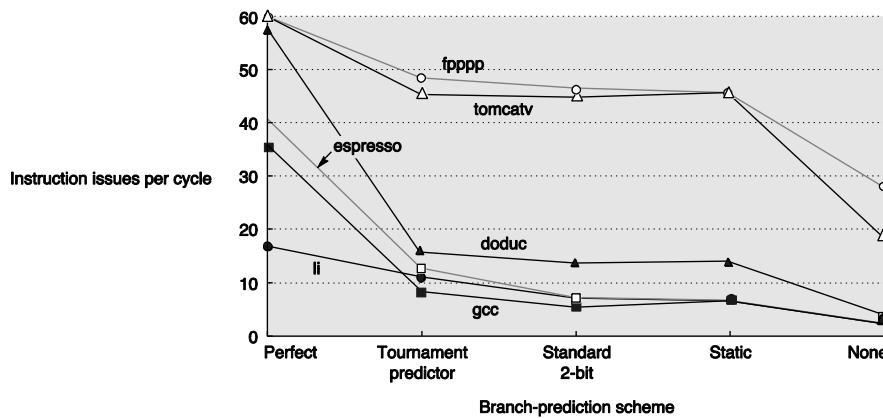


Figure 81. Structure de Processeur Superscalaire

Le mécanisme de prédition n'étant pas parfait le processeur peut être amené à faire des prédictions incorrectes qui se traduiront par le traitement et l'exécution d'instructions dans le mauvais chemin d'exécution. Le processeur aura spéculé sur la direction prise par un branchement et aura ainsi effectué une exécution spéculative (speculative execution). Dans ce cas il doit donc garantir que cette exécution spéculative n'aura pas affecté l'état du processeur au sens machine d'état de manière irréversible et qu'au contraire il puisse reprendre l'exécution sur le chemin correct.



© 2003 Elsevier Science (USA). All rights reserved.

Figure 82. Impact de la Prédition de Branchement sur le nombre d'instructions exécutables en parallèle

Malgré cette pénalité de performances l'exécution spéculative reste un mécanisme important pour l'accroissement des performances pour autant que le mécanisme de prédition est performant. La figure précédente établit le lien entre la qualité de la prédition de branchement et le nombre d'instructions pouvant être exécutés en parallèle et ce pour 5 mécanismes variant de gauche à droite sur l'axe des X du prédicteur parfait à une absence totale de prédition (None). Le prédicteur statique (static) consiste à adopter une stratégie de prédition statique toujours la même sur la direction d'un branchement. Les deux autres techniques sont dynamiques car la décision de prédition se fait au cours de l'exécution du programme et tient compte du comportement de l'instruction de branchement.

Cette prédition s'appuie sur une machine d'état de 2 bits et dont le comportement est décrit dans la figure ci-dessous.

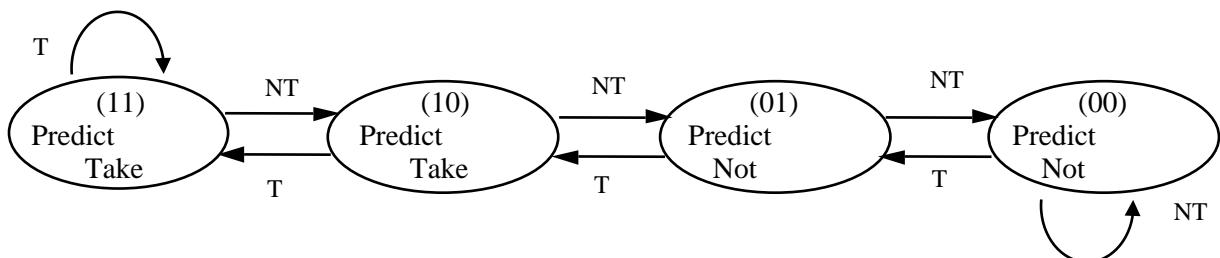


Figure 83. Machine d'états (2 bits) pour la prédition de branchement

Des variantes existent

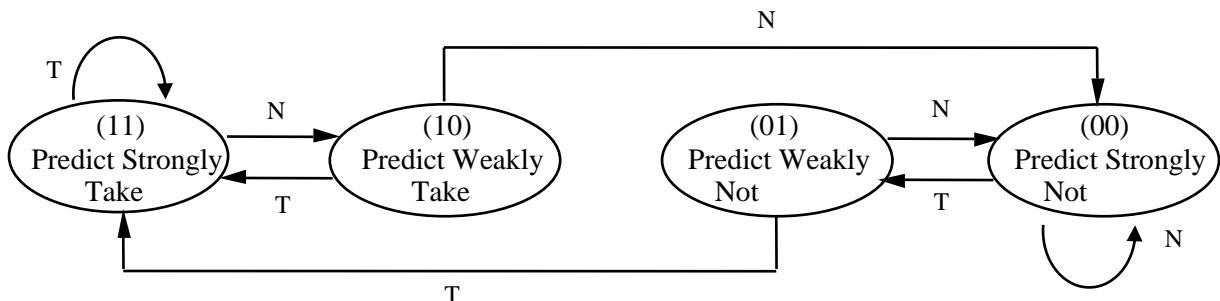
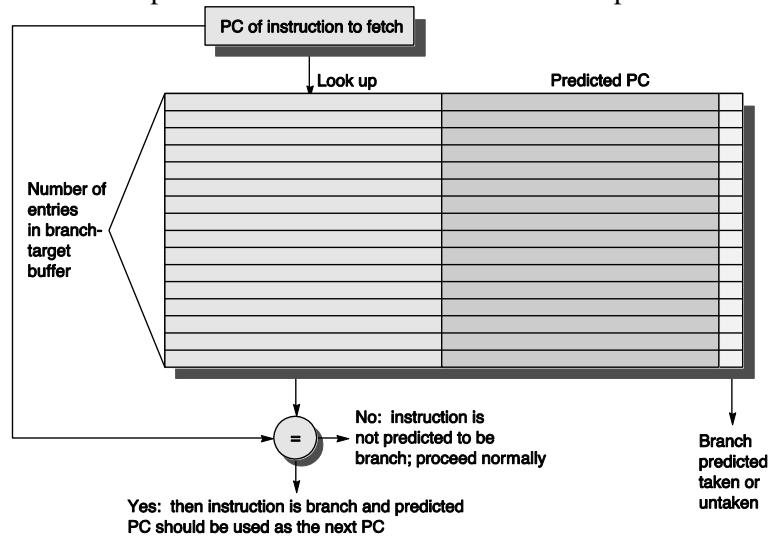


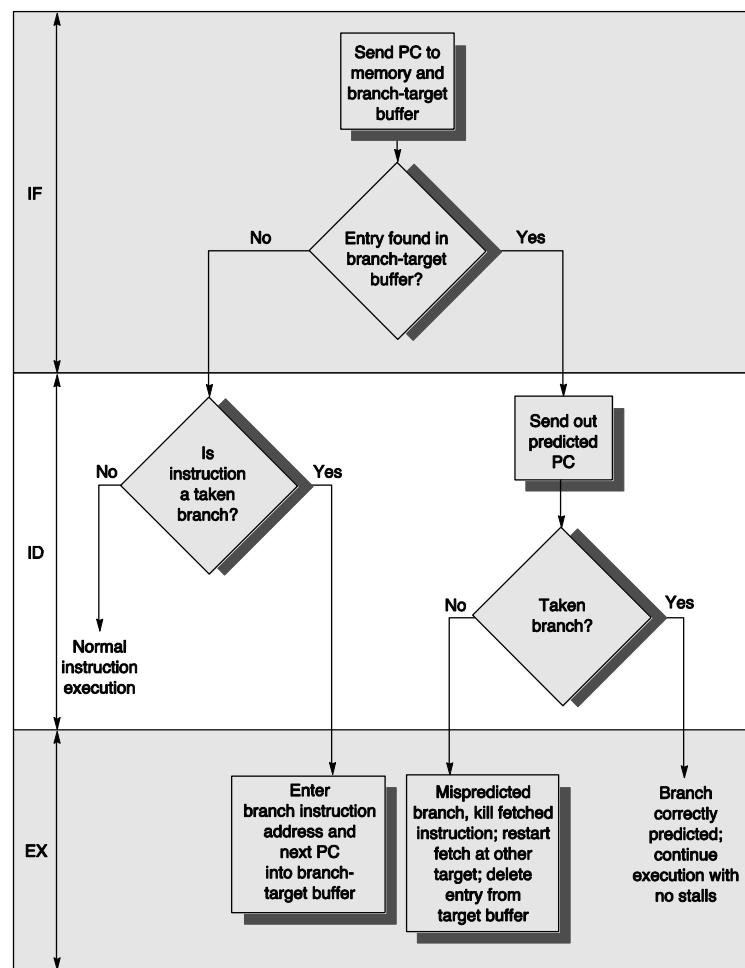
Figure 84. Machine d'états (2 bits) pour la prédition de branchements (variante Hysteresis scheme)

La prédition de branchement est un sujet de recherche très actif et de nombreuses propositions tentent d'améliorer les mécanismes par hybridation de prédicteurs ou composition de prédicteurs global ou local. Tous les processeurs haute performances du commerce utilisent la prédition de branchement.



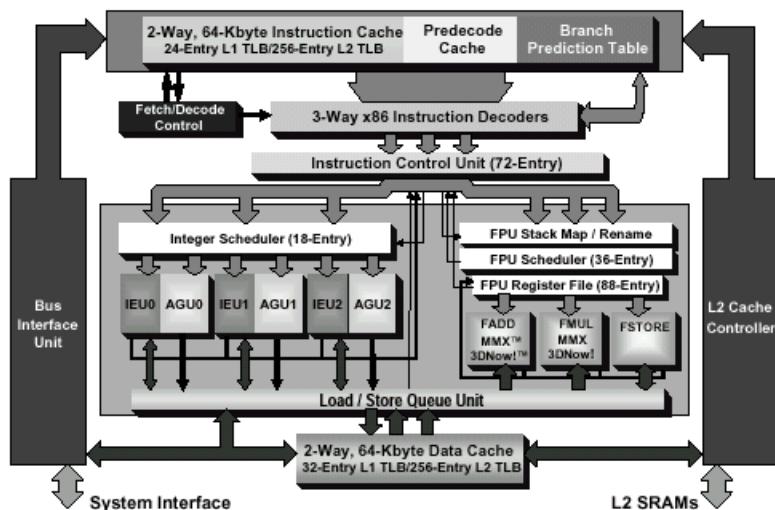
© 2003 Elsevier Science (USA). All rights reserved.

Figure 85. Structure de BTB



© 2003 Elsevier Science (USA). All rights reserved.

Figure 86. Procédure de Prédition de Branchement à base de BTB



AMD Athlon™ Processor Block Diagram

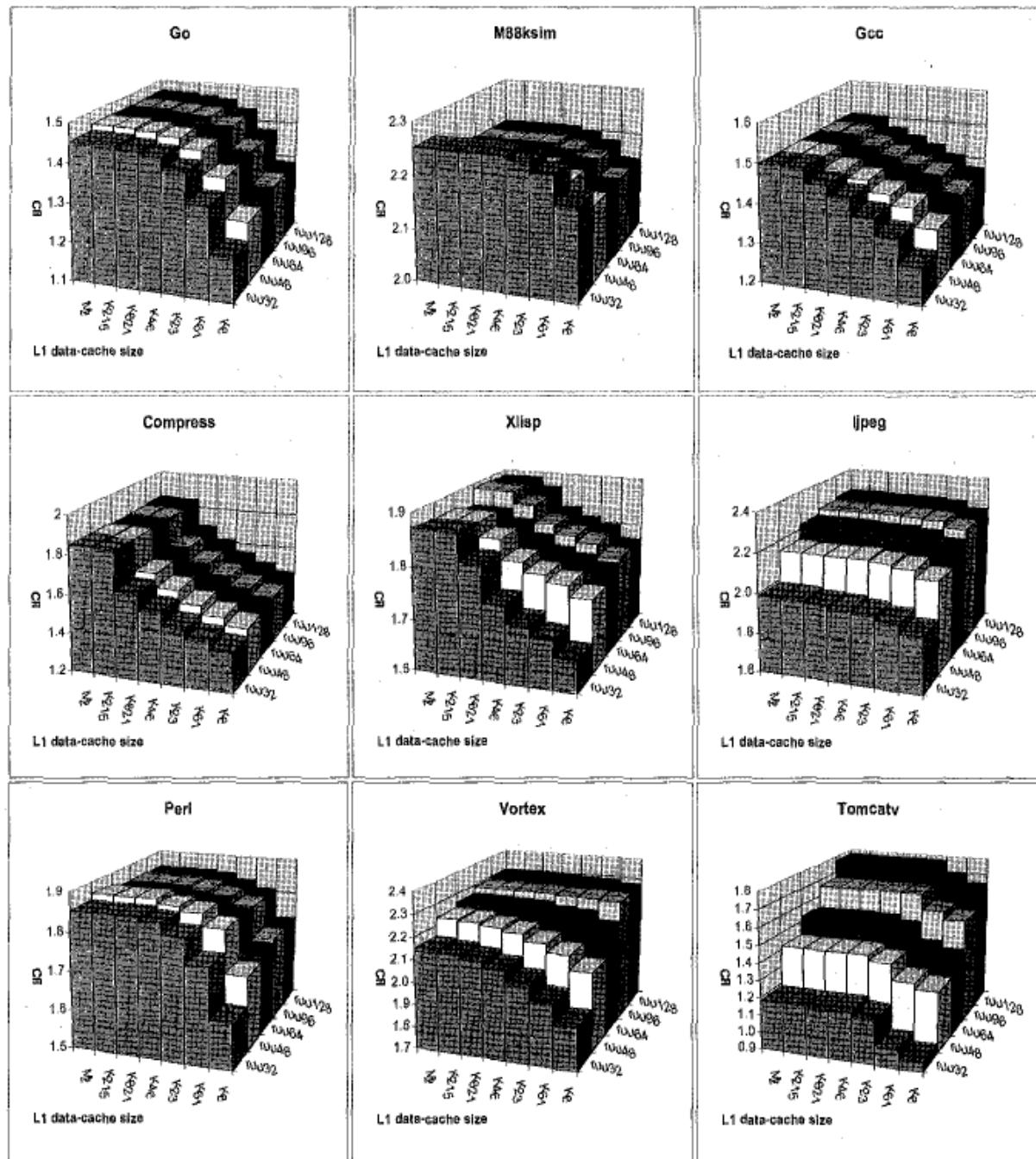
Figure 87. Processeur AMD Athlon (www.amd.com)

4.6 Processeur Superscalaire: compromis prediction de branchement, taille de la fenetre d'instructions, taille de caches

La prédition de branchement, la taille de la fenetre d'instructions et la taille des caches sont étroitement liés. De nombreuses études sur des applications benchmarks de SPEC CPU (go, M88ksim, Gcc, compress, xlisp, Perl, vortex, tomcatv) ont été réalisées qui mettent en évidence ces relations. Les figures suivantes décrivent les résultats obtenus en IPC sur un microprocesseur superscalaire lorsque l'on fait varier :

- taille du cache donnée et taille du RUU (fenêtre d'instructions) avec un prédicteur hybride,
- taille du cache donnée et taille du RUU (fenêtre d'instructions) avec un prédicteur parfait,
- taille du cache instruction et taille du RUU (fenêtre d'instructions) avec prédicteur parfait.

Les différentes applications ont des évolutions distinctes mais des tendances générales peuvent être observées.



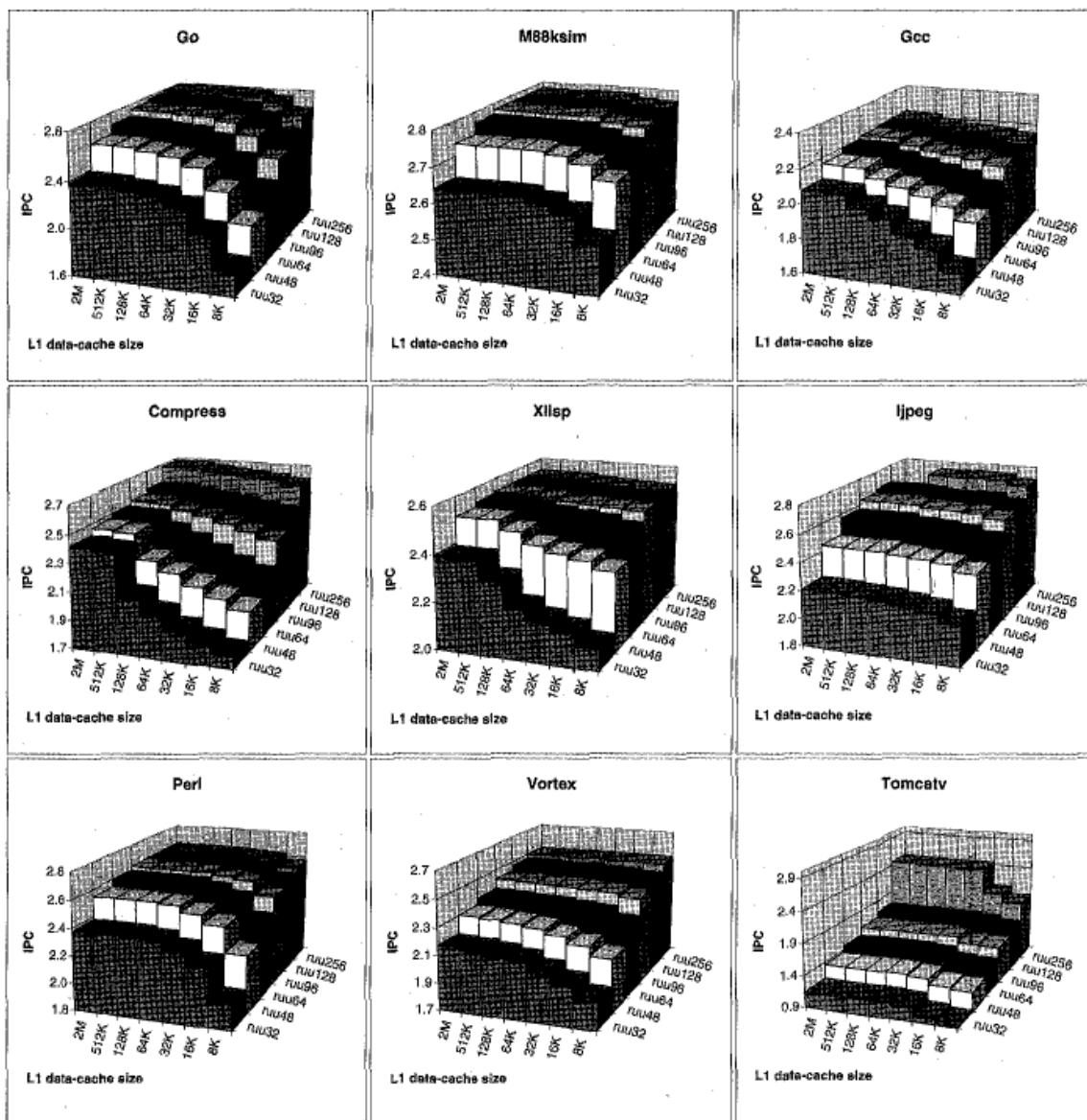


Fig. 2. IPC as a function of data cache size and RUU size with the 100 percent-direction-accuracy predictor. Simulator difficulties prevented us from obtaining the two ruu256 values for *linen* with large caches.

Figure 89. IPC en fonction de la taille du cache donnée et de la taille du RUU avec un prédicteur parfait.

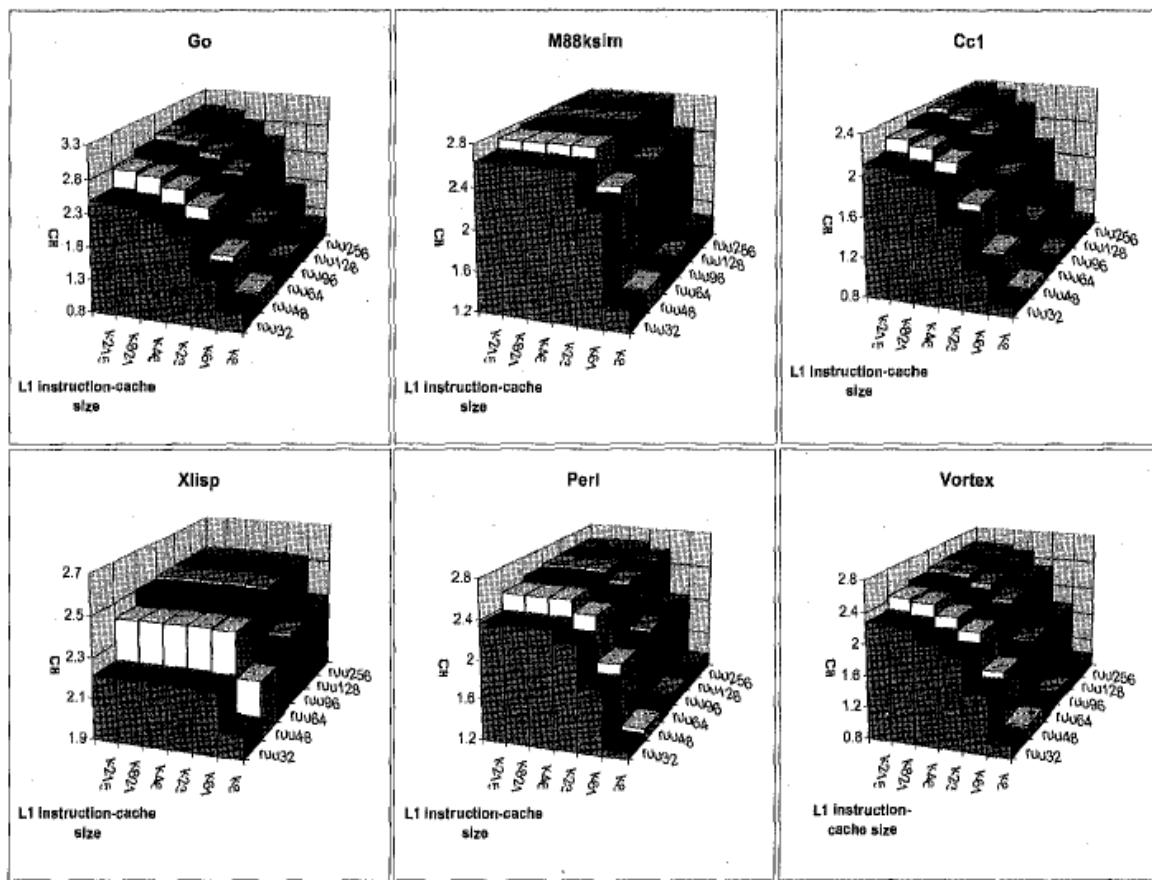
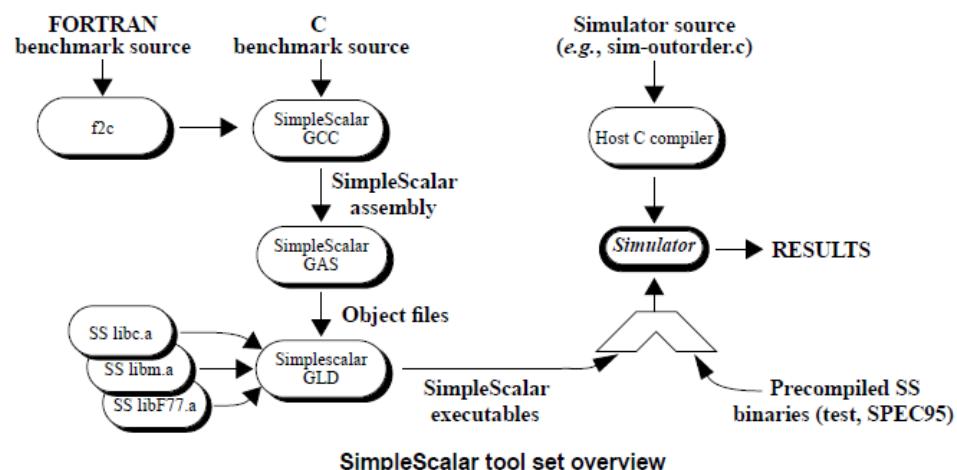


Fig. 6. IPC as a function of instruction cache size and RUU size with the 100 percent-direction predictor.

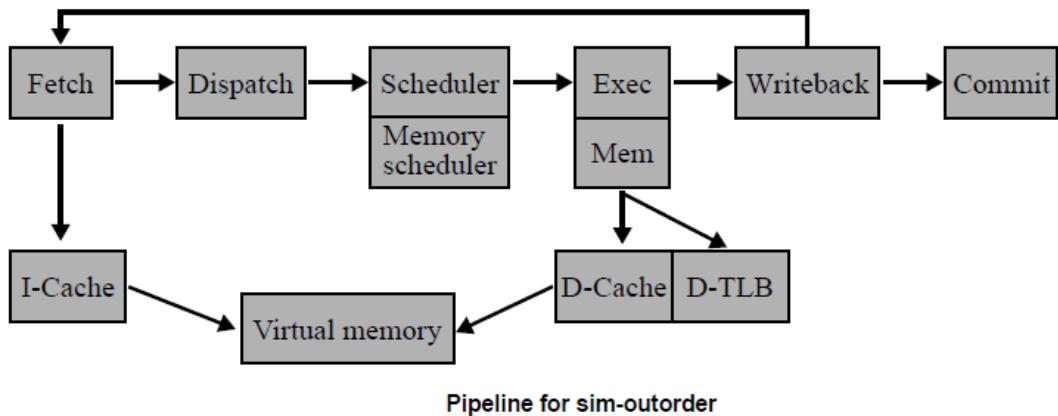
Figure 90.IPC en fonction de la taille du cache instruction et de la taille du RUU.

4.7 Evaluation de performances et dimensionnement de microprocesseur superscalaire par simulation : SimpleScalar

L'évaluation des performances d'un microprocesseur superscalaire et son dimensionnement est essentiellement basé sur des simulateurs de type ISS (Instruction Set Simulator) du type



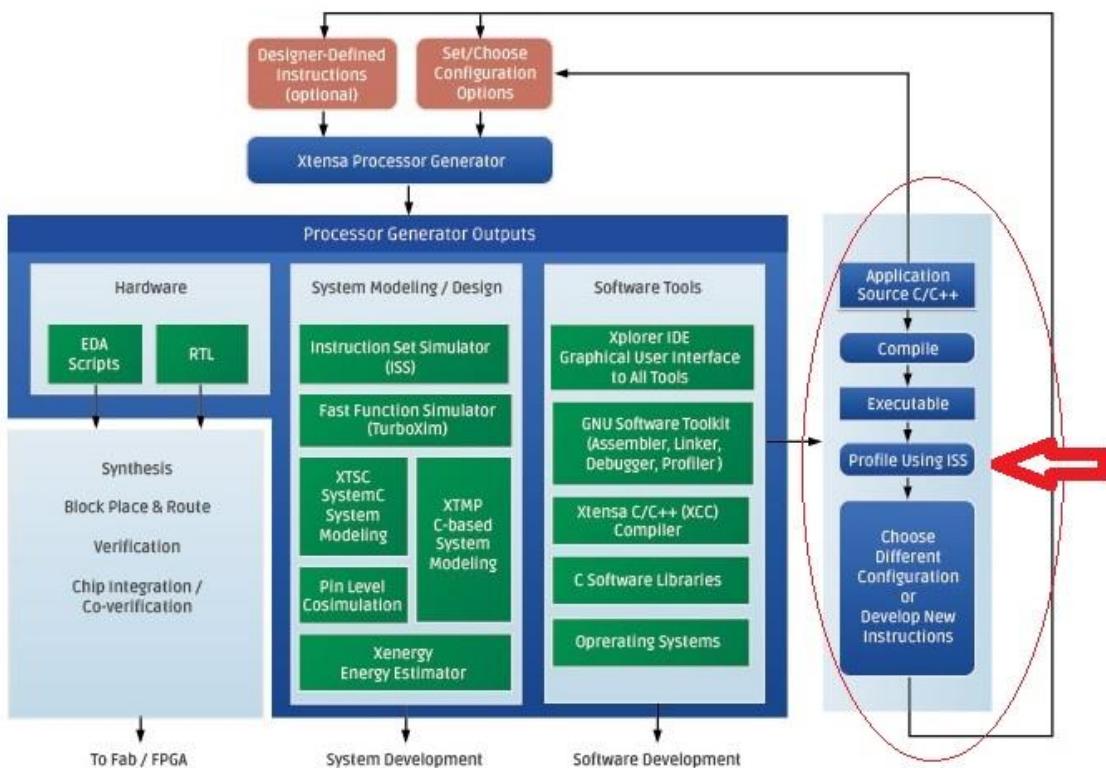
Simplescalar. Le simulateur prend en entrée des applications qui seront exécutées et reproduit le comportement architectural du processeur permettant de collecter un ensemble de statistiques sur les performances des différents éléments microarchitecturaux.



Le processeur modélisé par SimpleScalar est un processeur superscalaire dynamique incluant tous les mécanismes avancés des processeurs avec exécution spéculative. Le dimensionnement du processeur consiste à dimensionner la structure du pipeline, des unités de calcul, de l'architecture des caches, de l'unité de prédiction de branchement et de la fenêtre des instructions.

4.8 Environnements automatisés

Cette problématique du dimensionnement de processeur a fait l'objet de recherches intensives et certaines solutions sont apportées par des outils industriels comme l'outil de Cadence



<http://ip.cadence.com/ipportfolio/tensilica-ip/xtensa-customizable>

L'outil utilise un ISS du type simplescalar pour configurer la microarchitecture souhaitée pour être ensuite fournie à une générateur automatique d'environnement de développement, de circuits et d'outils divers de modélisation système.

4.9 Optimisation de processeur Superscalaire, Recherche Opérationnelle et Optimisation Multiobjectifs

L'optimisation multiobjectif est une branche de l'optimisation combinatoire dont la particularité est de chercher à optimiser simultanément plusieurs objectifs d'un même problème (contre un seul objectif pour l'optimisation combinatoire classique).

Un problème multiobjectif est un problème d'optimisation qui implique plusieurs fonctions d'objectifs.

Mathématiquement, un problème d'optimisation multiobjectif peut être formulé comme suit:

$$\begin{aligned} \min & (f_1(x), f_2(x), \dots, f_k(x)) \\ \text{s.t. } & x \in X, \end{aligned}$$

tel que l'entier $k \geq 2$ représente le nombre d'objectifs et l'ensemble X est l'ensemble réalisable des vecteurs de décision. Cet ensemble est défini par des fonctions de contraintes.

L'ensemble des points Pareto optimal est appelé le front de **Pareto**.

De nombreuses techniques issues des mathématiques appliquées permettent d'apporter une solution à la résolution des problèmes de dimensionnement de microprocesseurs.

4.10 Conclusion

Le problème général de la conception de microprocesseurs superscalaire exige une approche multiobjective prenant pour objectifs la performance, la surface du circuit et la consommation d'énergie.

La méthodologie générale s'appuie sur : 1. des techniques d'optimisation multiobjectifs 2. la simulation des performances à base d'ISS et de la consommation d'énergie 3. des outils d'estimation de la surface de circuit sur la base d'une technologie semiconducteur .

La microarchitecture d'un microprocesseur inclut tout en ensemble de supports sophistiqués issus des modélisations mathématiques comme la prédition des branchements.

La maîtrise de la conception des microprocesseurs superscalaires est limitée pour le parallélisme d'instructions (ILP – Instructions Level Parallelism) effectivement exploitable lors de l'exécution. De nombreux processeurs commerciaux comme les processeurs ARM exploitent l'architecture superscalaire dynamique.

Chapitre 5 - Multiprocesseurs

5.1 Introduction

La recherche de la performance par l'accroissement de la fréquence a été l'approche suivie pendant de nombreuses années mais cette stratégie est désormais stérile pour des raisons de consommation d'énergie. En effet, la consommation d'énergie est proportionnelle à la fréquence d'horloge et par conséquent aux fréquences actuelles une poursuite dans cette voie généreraient des microprocesseurs ayant des densités d'énergie inacceptables. A ce phénomène s'ajoute les difficultés de concevoir un microprocesseur exploitant le parallélisme d'instructions tout en conservant une complexité acceptable et un rendement relatif à l'investissement en surface silicium. Les architectures monoprocesseurs dans leur mode d'exécution actuel ne peuvent résoudre ce triple défi (performance, consommation d'énergie et surface) et les investissements retournent des gains en diminution constante (Law of diminishing returns). La solution adoptée a été de suivre une voie multiprocesseur dans laquelle plusieurs microprocesseurs sont implantés sur un seul circuit avec des fréquences d'horloge relativement moindre le gain en performance étant attendu par le parallélisme d'exécution offert par la duplication des ressources. On parle alors de multiprocesseurs sur puce (MPSOC - Multiprocesseurs System On Chip ou CMP- Chip Multiprocessors).

Dans un modèle de multiprocesseur à mémoire partagée (shared memory multiprocessors) les processeurs communiquent entre eux par des variables partagées en mémoire. Ce type d'architecture nécessite :

- un système opératoire pour multiprocesseur,
- une algorithmie parallèle qui exige de repenser les algorithmes développés pour les architectures monoprocesseurs,
- des compilateurs prenant en compte de manière explicite ou implicite ce parallélisme et enfin,
- des modèles de programmation adéquats.

Ce n'est qu'avec ces environnements adaptés qu'il est possible d'exploiter le parallélisme d'une application à travers son implémentation sur une architecture multiprocesseur. L'exécution d'applications totalement indépendantes sur un multiprocesseur revient à utiliser ces machines comme des serveurs ou le débit des applications est le but recherché.

L'étude d'implémentation parallèle efficace d'algorithmes parallèles nécessite une connaissance approfondie des points précédents. La tendance des constructeurs de microprocesseurs d'augmenter le nombre de processeurs sur une puce (8 processeurs en 2009) ne fera que renforcer cette exigence.

Dans sa forme la plus classique une architecture multiprocesseur à base de bus consiste à connecter plusieurs processeurs sur un bus qui sera utilisé pour la communication entre les processeurs et la mémoire et les entrées/sorties.

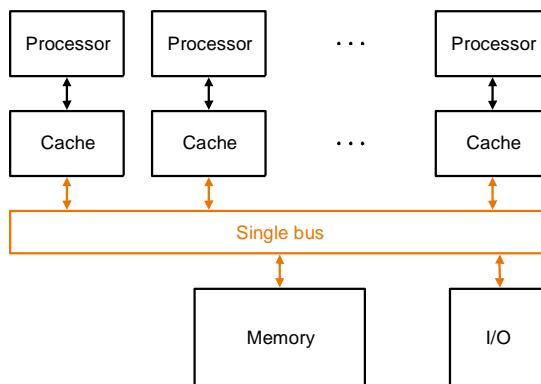
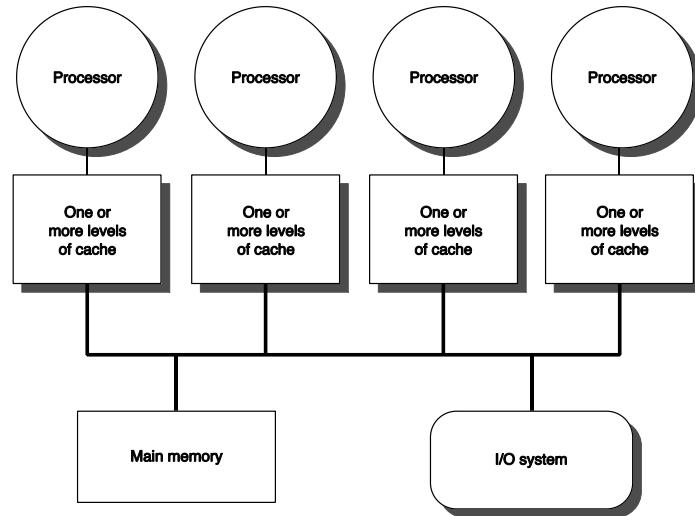


Figure 91. Architecture multiprocesseur à base de bus

Le cas général de ce modèle est donné dans la figure suivante celui des architectures parallèles à mémoire centralisée et plusieurs niveaux de caches.



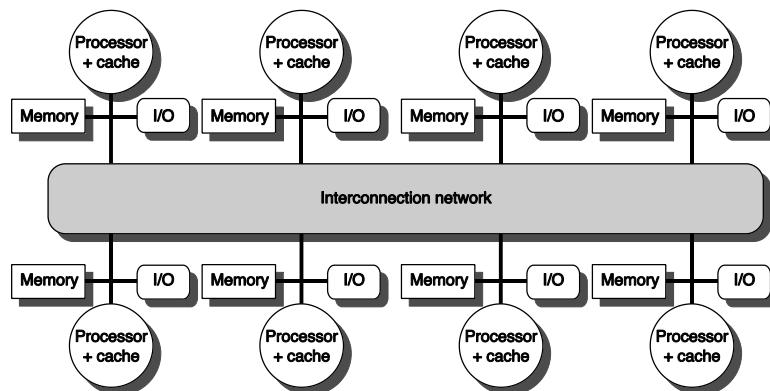
© 2003 Elsevier Science (USA). All rights reserved.

Figure 92. Architecture multiprocesseur avec plusieurs niveaux de cache

Ces architectures présentent une symétrie par rapport à la mémoire et plus particulièrement le temps d'accès entre les processeurs et la mémoire est uniforme. Pour cette raison ces architectures sont aussi appelées multiprocesseurs symétriques (symmetric multiprocessors – SMP) et la caractéristique d'uniformité des temps d'accès est spécifiée par architectures avec accès mémoire uniforme (Uniform Memory Access – UMA). Ce type d'architecture est actuellement la plus diffusée et particulièrement pour sa facilité de programmation.

Ce modèle a malheureusement ses limites et n'est pas en mesure de passer l'échelle lorsque le nombre de processeurs augmente. En effet le temps d'accès à la mémoire (temps de latence mémoire) devient prohibitif et il est donc nécessaire de trouver une organisation mémoire qui permette de pouvoir accéder aux mémoires.

La seconde catégorie est celle des architectures à mémoire distribuée.



© 2003 Elsevier Science (USA). All rights reserved.

Figure 93. Architecture multiprocesseur à mémoire distribuée

Ce modèle distribue la mémoire dans une configuration basée sur un noeud de calcul composé d'un processeur et de sa hiérarchie mémoire, d'accès à des entrées-sorties et une partie de la mémoire. Le temps d'accès à la mémoire pour un processeur quelconque dépendra de la position de la mémoire dans le système et donc le temps d'accès n'étant plus uniforme ces architectures sont appellées architectures à accès non uniforme (NUMA – Non Uniform Memory Access). La distribution de la mémoire selon cette organisation a néanmoins ses avantages. Cela représente une approche efficace d'un point de vue cout d'augmenter la bande passante mémoire si sur le principe de localité des références la majorité des références se fait sur la mémoire locale. D'autre part le temps de latence à la mémoire est réduit lors des accès à la mémoire locale. Par contre, naturellement la communication de données entre processeurs devient plus complexe et présente une plus grande latence.

Les architectures à mémoire distribuée peuvent adopter deux types de communication entre les processeurs. Le premier considère que malgré le fait que les mémoires sont physiquement distribuées elles constituent ensemble un espace d'adressage partagé. Elles sont dans ce cas appellées architectures distribuées à mémoire partagée (Distributed Shared Memory – DSM). Ce modèle permet une programmation qui est la plus proche du modèle classique de communication des données par des variables présentes dans un espace d'adressage commun. Par conséquent dans ce modèle chaque mémoire locale a une partie de l'espace d'adressage global. La deuxième approche considère que chaque mémoire locale a son propre espace d'adressage séparé des autres mémoires. De ce fait, chaque noeud de calcul (processeur – mémoire) peut être considéré comme une entité indépendante ce qui en termes de programmation a une implication fondamentale. Dans ce type de configuration la communication devient explicite et non plus implicite par variables dans un espace d'adressage partagé. Cette communication consiste à remplacer les opérations de lecture et d'écriture sur des variables d'un autre espace d'adressage par l'envoi de messages à destination du noeud de calcul responsable de cet espace d'adressage. Cette communication par passage de message a un standard de programmation MPI (Message Passing Interface) qui représente la communication par passage de messages.

5.2 Réseaux d'interconnexions

Les architectures parallèles ont fait l'objet de recherches intensives pour déterminer les réseaux d'interconnexions les plus appropriés entre les processeurs et les mémoires et entre processeurs.

Cette problématique bien étudiée dans le domaine des supercalculateurs est revisité dans le cadre des multi-coeurs.

Un exemple d'évolution commencé il y a déjà 10 ans est donné par Intel qui a connu un accroissement de deux coeurs de processeurs par année comme le montre la figure ci-dessous.

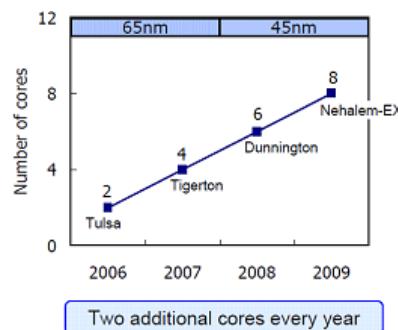


Figure 94. Evolution Architecture multiprocesseur - Intel

et la communication entre ces coeurs a nécessité un réseau différent d'un bus pour permettre un parallélisme des communications et une meilleure bande passante. La figure suivante montre le processeur Nehalem d'intel et les choix effectués.

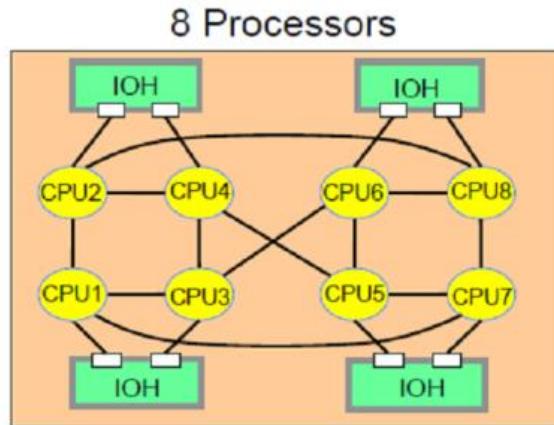


Figure 95. Architecture multiprocesseur

La tendance actuelle en technologie semiconducteur se traduit à chaque passage à une nouvelle technologie (transistor) à une réduction du temps de propagation d'une porte logique (gate delay) mais en même temps à une accroissement relatif du temps de propagation dans les fils d'interconnexions. Comme on peut le voir sur la figure suivante les seuls fils d'interconnexions qui se rapprochent de l'évolution des portes logiques sont les fils locaux.

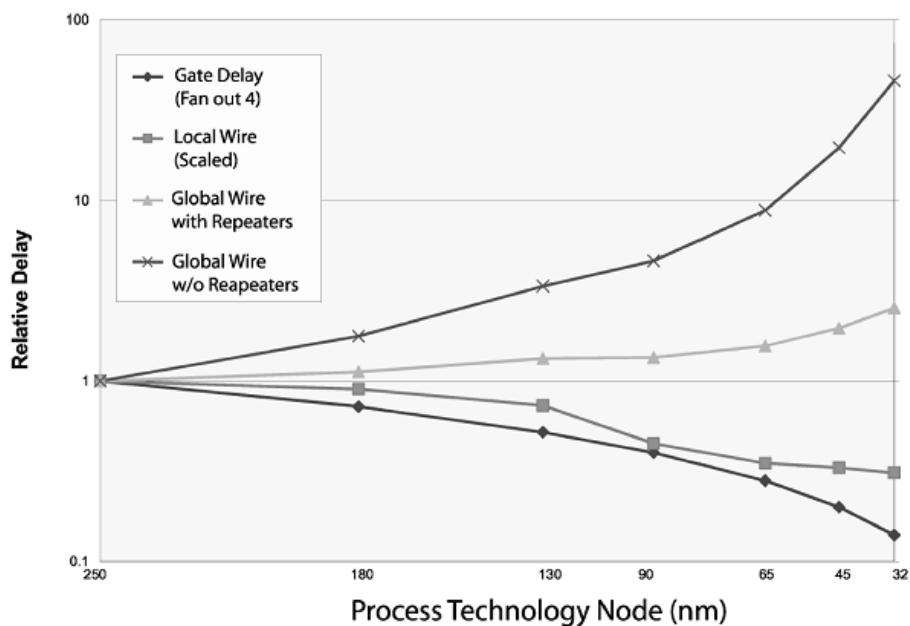
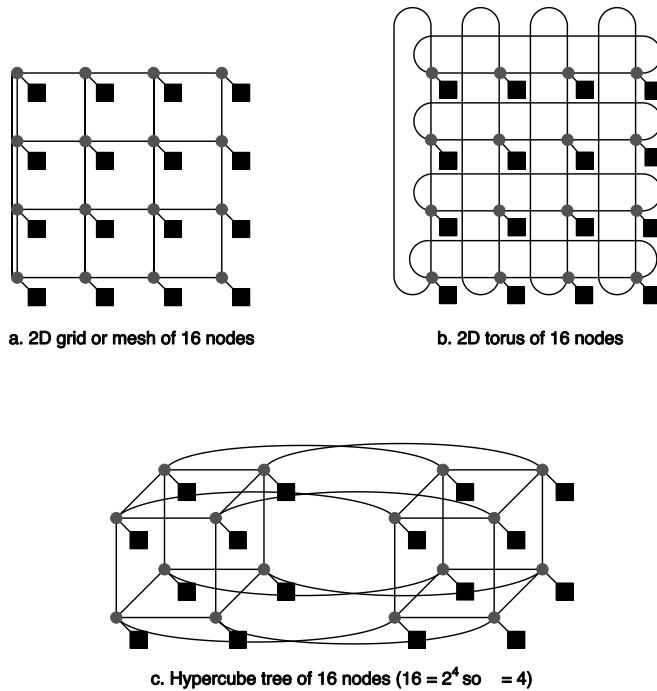


Figure 96. temps de propagations porte logique et connexions en fonction de la technologie semiconducteur

Cette contrainte forte encourage la spécification de topologies et des connexions courtes entre composants sur une même puce. La figure suivante décrit des 3 types de topologies majoritairement adaptées à cette contrainte : (1) les topologies de type grilles (Mesh interconnection network) (2) les topologies de type tore (Torus interconnection network) et les (3) les topologies de type hypercubes (Hypercube interconnection network).



© 2003 Elsevier Science (USA). All rights reserved.

Figure 97. Réseaux d'interconnexions pour architecture multiprocesseur

5.3 Architectures parallèles et cohérence de caches

L'utilisation d'architectures parallèles a pour premier objectif d'exécuter des applications parallèles effectuant des calculs de manière coopérative sur des données partagées. Le partage des données dans des architectures parallèles avec hiérarchie mémoire pose la question de la cohérence des données partagées . On parlera par extension de la cohérence des caches.

Pour illustrer le problème de la cohérence des caches nous allons considérer comme exemple une architecture multiprocesseur à mémoire partagée composée de 3 processeurs (P_1 , P_2 et P_3) avec mémoire cache et utilisant un bus comme moyen de communication.

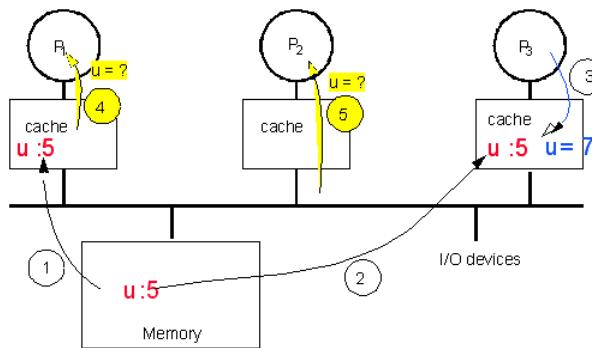


Figure 98. Cohérence de cache: exemple.

Nous effectuons l'hypothèse que cette architecture exécute un programme parallèle et que chaque processeur exécute une tâche sur des données partagées. On considère u comme une variable partagée entre les 3 processeurs qui effectueront des opérations de lecture et d'écriture sur la variable u .

Initialement la variable u se trouve en mémoire et a pour valeur 5. Dans la séquence d'opérations réalisées, le processeur P1 souhaite lire la variable u . Celle ci étant en mémoire mais pas dans son cache sa tentative de lecture de la variable dans son cache donnée se solde par un défaut de cache (cache miss) la donnée n'étant pas présente dans son cache. La donnée est alors lue de la mémoire et une copie est placée dans la mémoire cache de P1.

5.4 Architectures parallèles et cohérence de caches : mémoire partagée avec bus

L'ensemble des actions réalisées pour maintenir la cohérence des caches est appellée le protocole de cohérence de caches. Le protocole snoopy est un protocole adapté aux architectures parallèles avec bus.

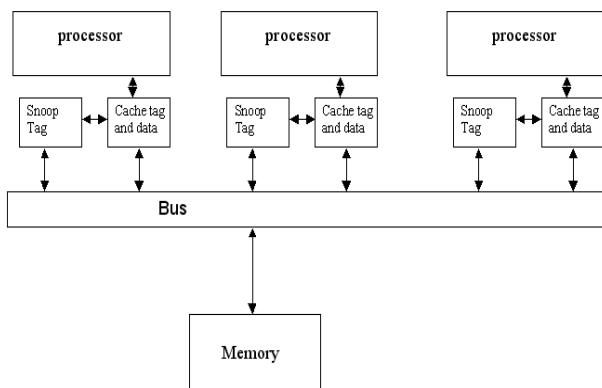


Figure 99. Architecture multiprocesseur à base de bus avec cohérence de caches

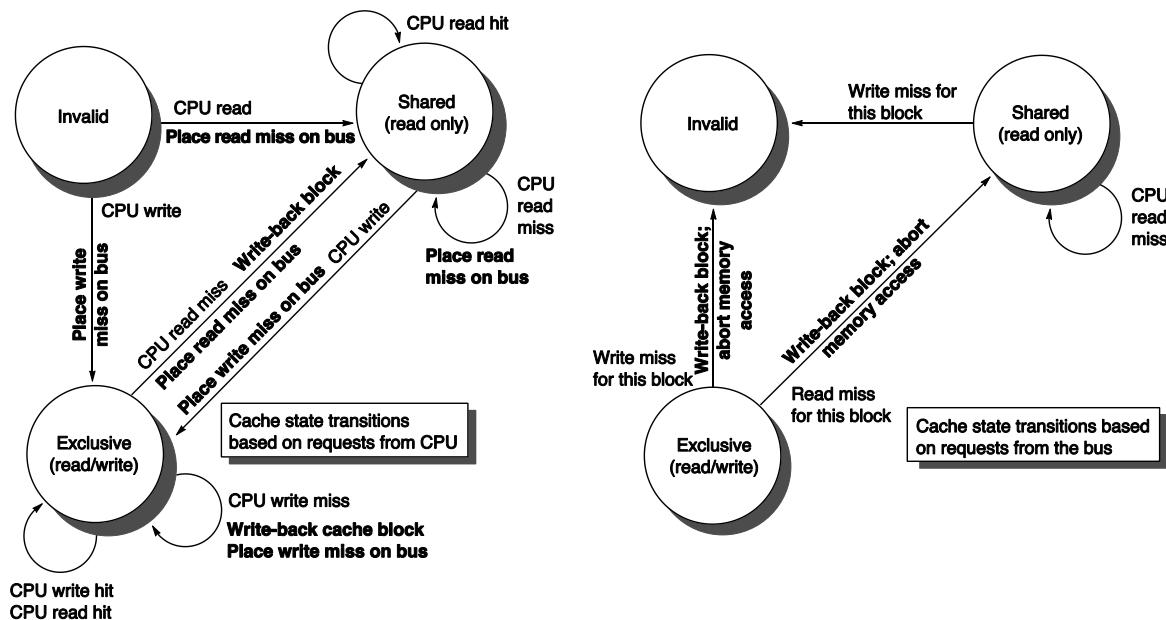
Le premier cas d'étude concerne les architectures parallèles avec structure de communication de type bus. Le bus de par la sérialisation des requêtes vers la mémoire représente un point d'observation unique des opérations de lecture et d'écriture vers la mémoire. La résolution du problème de la cohérence des caches passe donc par : (1) l'observation sur le bus par chaque processeur des requêtes réalisées par les autres processeurs (2) des actions d'invalidation ou de mise à jour des données suivant l'état du bloc de donnée concerné par la transaction et la nature de la transaction. Les mémoires caches utilisées jusqu'à alors dans un cadre monoprocesseur doivent alors être étendues par un contrôleur de cache intégrant le support pour la cohérence des caches. Celui-ci est connecté sur le bus comme décrit dans la figure précédente et a pour fonction la maintien de la cohérence du cache auquel il est rattaché. Ce type de protocole basé sur l'invalidation de données est appelé *write invalidate protocol* car il invalide les autres copies d'un bloc lors d'une opération d'écriture. Ce protocole est le plus commun. Un accès exclusif garantit qu'il n'existe pas d'autres copies du bloc de donnée qu'il soit possible de lire ou d'écrire au moment où une opération d'écriture va se réaliser: toutes les autres copies sont invalidées.

Pour représenter les actions de cohérence à réaliser en fonction de l'état du bloc et de la requête en cours sur le bus le formalisme des machines d'états est utilisé. L'état d'un bloc peut être :

- invalid (*invalide*)
- shared (*partagé*)
- exclusive (*exclusif*)

Initialement un bloc est dans un état invalide car il n'est pas présent dans la mémoire cache. Sur une tentative d'opération de lecture (CPU read), un défaut de cache en lecture advient qui génère une requête de lecture pour read miss sur le bus. Le bloc passe dans l'état shared (partagé) qui pourra être partagé en lecture par plusieurs processeurs et donc plusieurs caches. Le bloc restera dans l'état shared tant que les processeurs n'effectueront que des opérations de lecture sur ce bloc. Si le processeur décide de modifier

le bloc par une opération d'écriture depuis l'état invalid ou depuis l'état shared alors le bloc passe dans l'état exclusive. Cet état exclut tout partage du bloc avec d'autres processeurs. Un bloc ne peut donc être dans un état exclusive que dans une et une seule mémoire cache à la fois.



© 2003 Elsevier Science (USA). All rights reserved.

Figure 100. Protocole de cohérence de caches write-invalidate et cache write-back.

Le passage inverse de l'état exclusive vers shared se réalisera lorsqu'une requête de lecture sur ce bloc provenant d'un autre processeur sera émise. Le processeur qui possède le bloc est donc obligé de partager le bloc et change l'état du bloc d'exclusif à shared. La figure précédente donne 2 vues des transitions : (1) l'une depuis le processeur (2) l'autre depuis le bus.

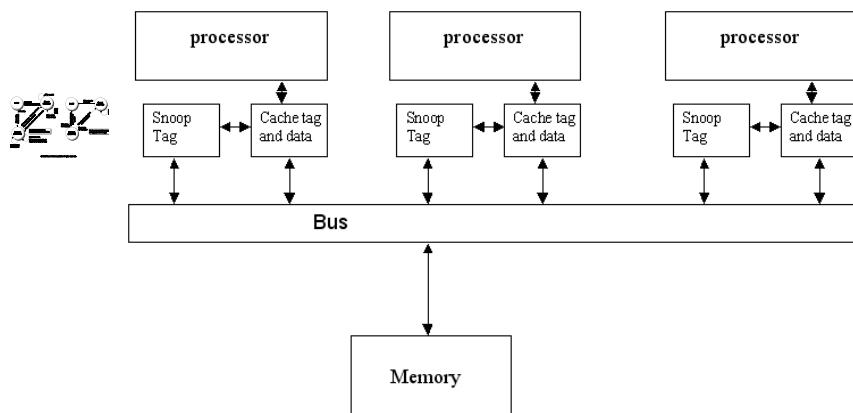


Figure 101. Architecture multiprocesseur avec cohérence de caches

5.5 Exemple : Multiprocesseur ARM MP11Core

Un exemple de multicore avec protocole de cohérence de cache de type snoopy est le processeur MP11Core de la société ARM. Ce multicore est composé de 4 coeurs de processeurs (MP11 CPU0, CPU1, CPU2 et CPU3) qui accèdent au bus par une unité de type snoopy (Snoop Control Unit). Le choix réalisé ici a été donc de centraliser les requêtes et les filtrer plutôt que de laisser chaque processeur accéder séparément. Cette approche est appropriée pour les multicore embarqué que l'on peut aussi composer.

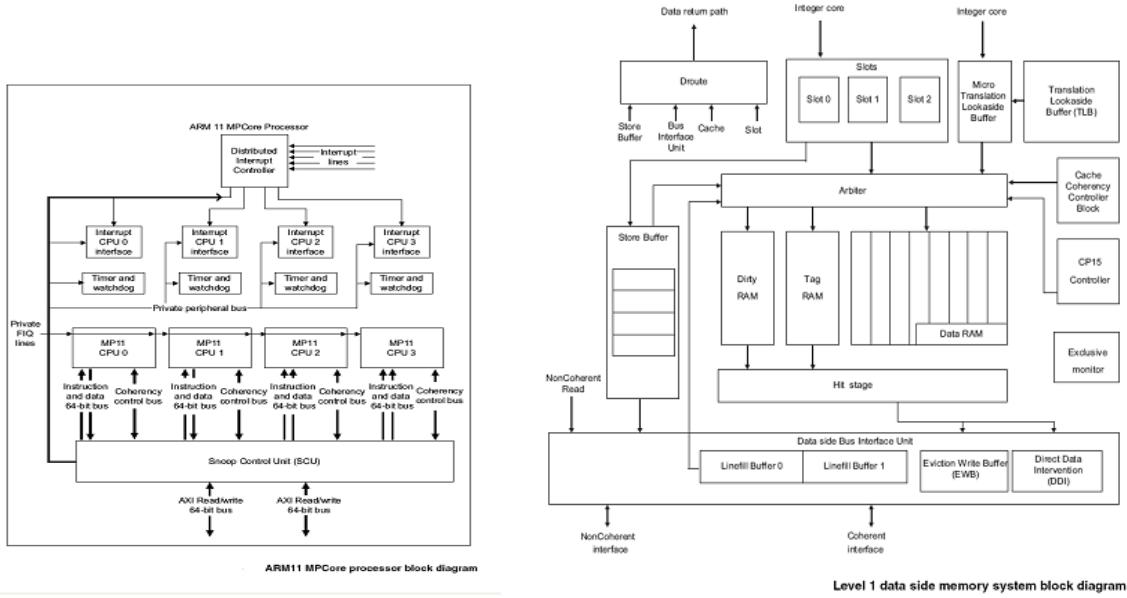
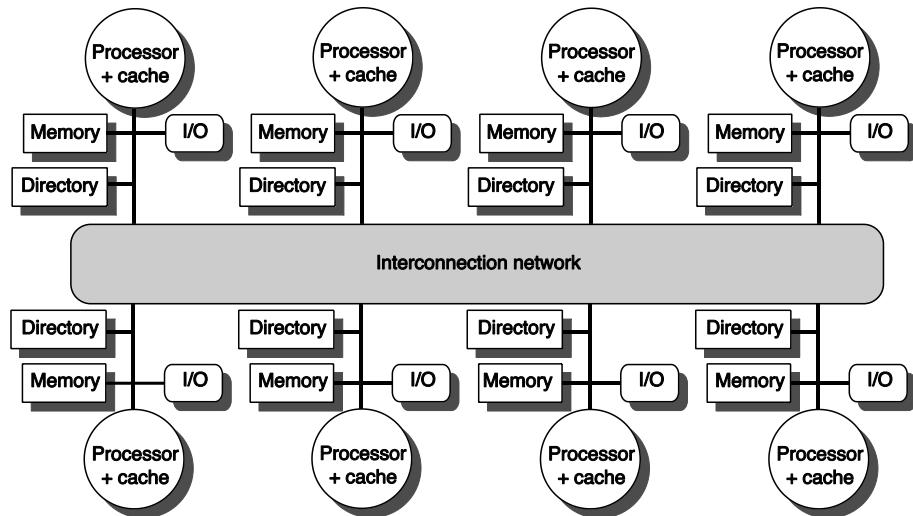


Figure 102. Architecture ARM

On voit que sur le détail de la structure de cache donnée L1 un contrôleur est rajouté pour contrôler les actions de cohérence.

5.6 Architectures parallèles et cohérence de caches : mémoire distribuée et réseau d'interconnexion

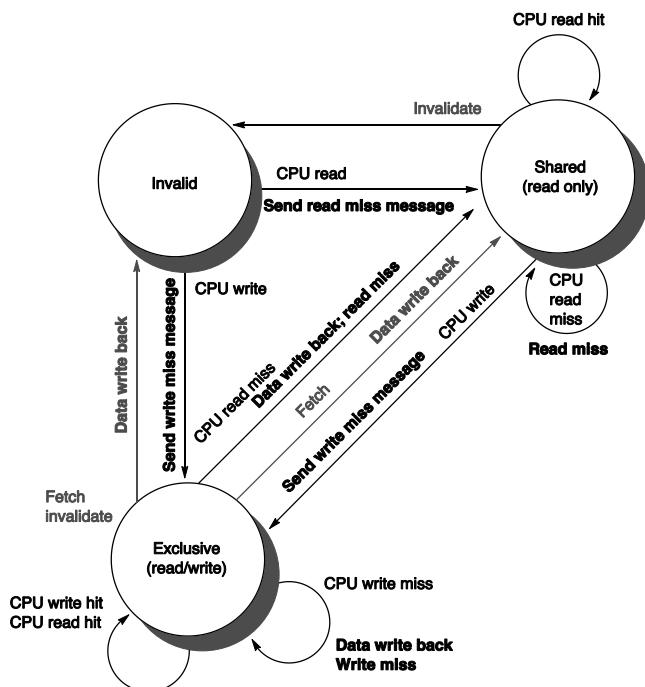
Dans les architectures à mémoire distribuée la solution d'un protocole de type snoopy n'est pas applicable car il n'existe pas de point unique dans l'architecture qui permette de surveiller les opérations de lecture et d'écriture. L'alternative est alors un protocole à base de directoire : directory protocol. La figure suivante décrit une architecture distribuée modifiée pour prendre en compte le protocole à base de directoire. Un directoire conserve l'état de chaque bloc qui est dans une mémoire cache.



© 2003 Elsevier Science (USA). All rights reserved.

Figure 103. Architecture multiprocesseur à mémoire distribuée et protocole de cohérence de type directoire

Le formalisme de description d'un protocole de cohérence de cache suit le même formalisme que précédemment c'est à dire une approche par machine d'état. Par contre, contrairement aux mécanismes de communication sur un bus les mécanismes de communication sur un réseau d'interconnection se fait par message entre processeurs.



© 2003 Elsevier Science (USA). All rights reserved.

Figure 104. Protocole de cache pour directoire

5.7 Architectures parallèles et cohérence de caches : mémoire distribuée et réseau d'interconnexion Performance d'applications

Il est essentiel d'évaluer l'impact des mémoires caches et leur organisation dans le cadre des architectures parallèles. La surface des circuits dédiée aux mémoires caches dans les architectures multicores restent dominantes comme c'était le cas dans le cas monoprocesseur.

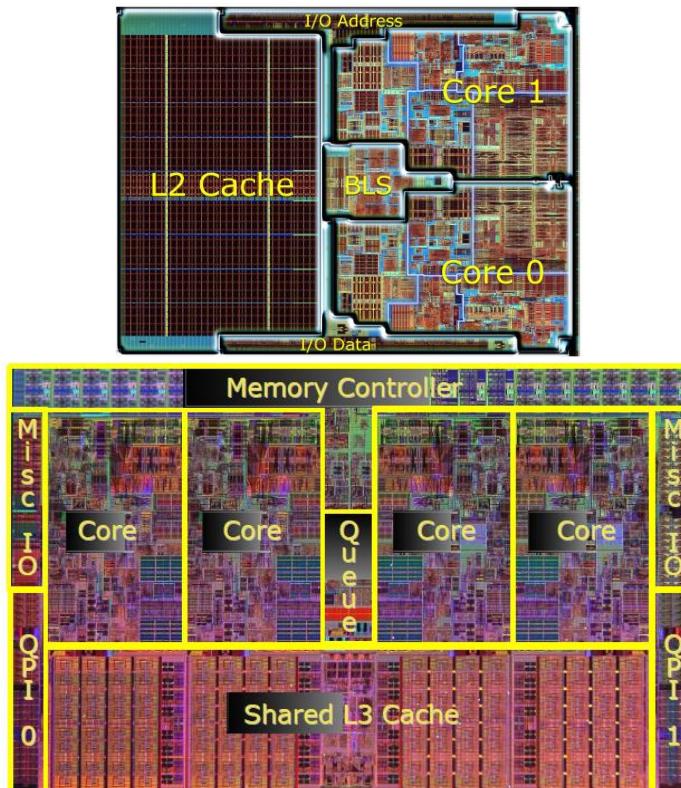
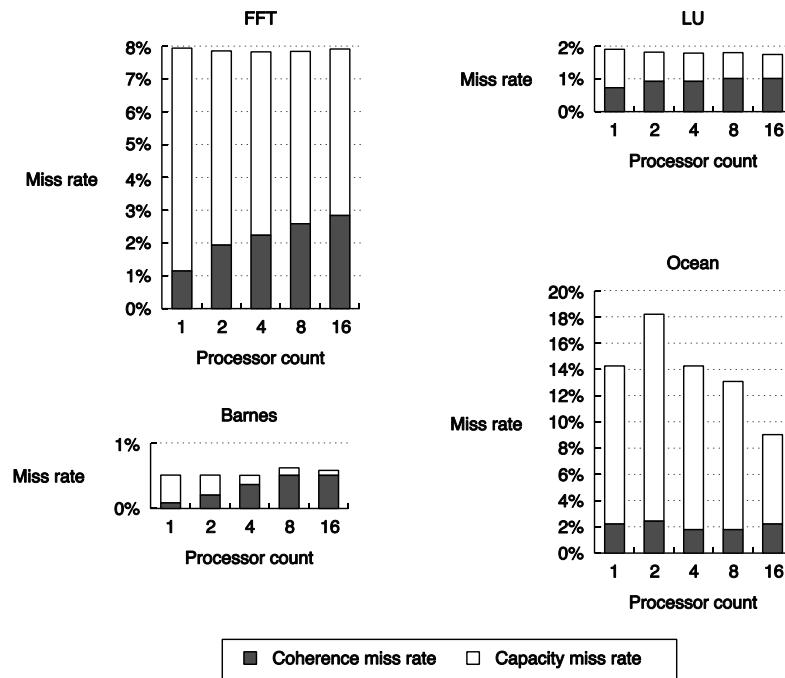


Figure 105. Architecture dual core avec cache L2 et quadcore avec cache L3 partagé (intel)

Dans cet objectif le choix d'applications parallèles (benchmarks) influence naturellement les résultats obtenus. Les applications parallèles choisies sont : (1) FFT : transformée de fourier (Fast Fourier Transform) (2) LU : application scientifique sur les matrices (3) Barnes : application d'astrophysique (4) ocean : application effectuant une modélisation des courants dans les océans.

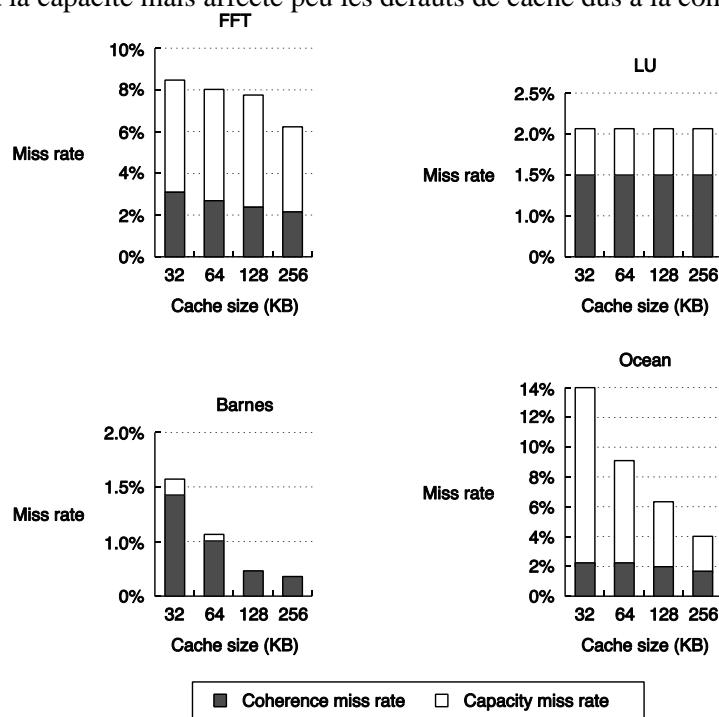
Dans une première analyse on évalue l'impact de la taille du bloc sur le défaut de cache et on réalise que globalement l'augmentation de la taille du bloc réduit le taux de défaut de cache.



© 2003 Elsevier Science (USA). All rights reserved.

Figure 106. Impact sur le taux de défaut de cache du nombre de processeur

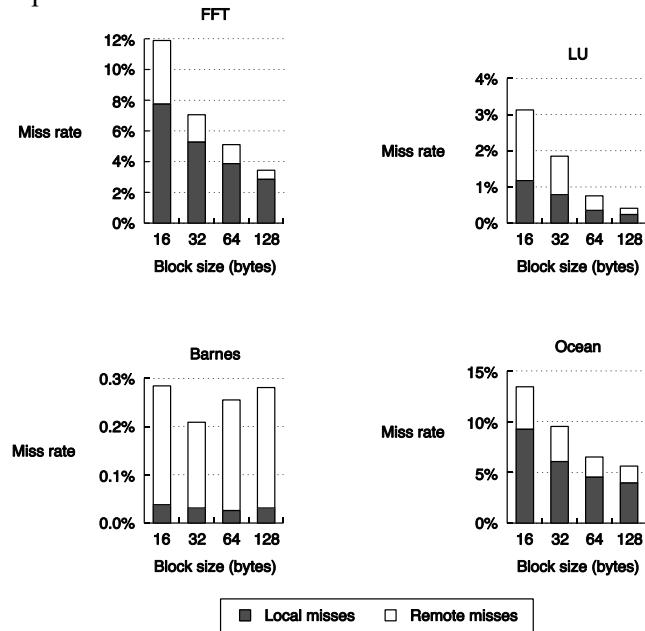
Dans une seconde analyse on évalue l'impact de la taille du cache sur le défaut de cache cohérence et capacité et on réalise que globalement l'augmentation de la taille du cache tend à réduire le taux de défaut de cache du à la capacité mais affecte peu les défauts de cache dus à la cohérence.



© 2003 Elsevier Science (USA). All rights reserved.

Figure 107. Impact sur le taux de défaut de la taille du cache

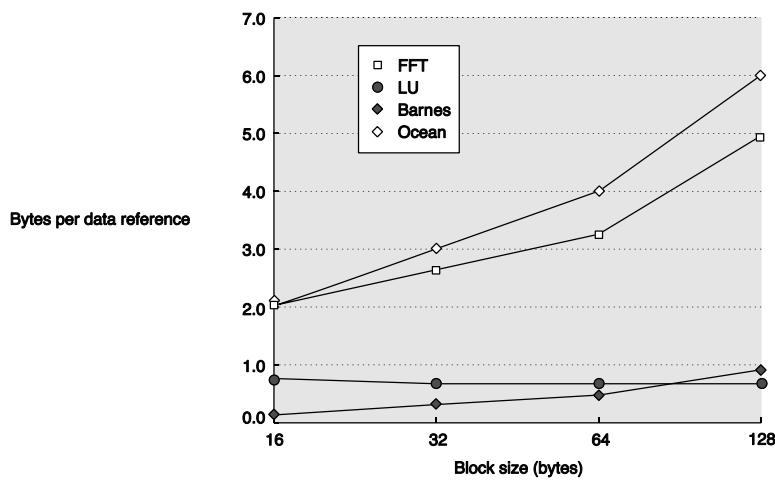
Enfin, une analyse de l'impact de la taille du bloc sur le défaut de cache montre comme on



© 2003 Elsevier Science (USA). All rights reserved.

Figure 108. Analyse de l'impact de la taille du bloc sur le défaut de cache

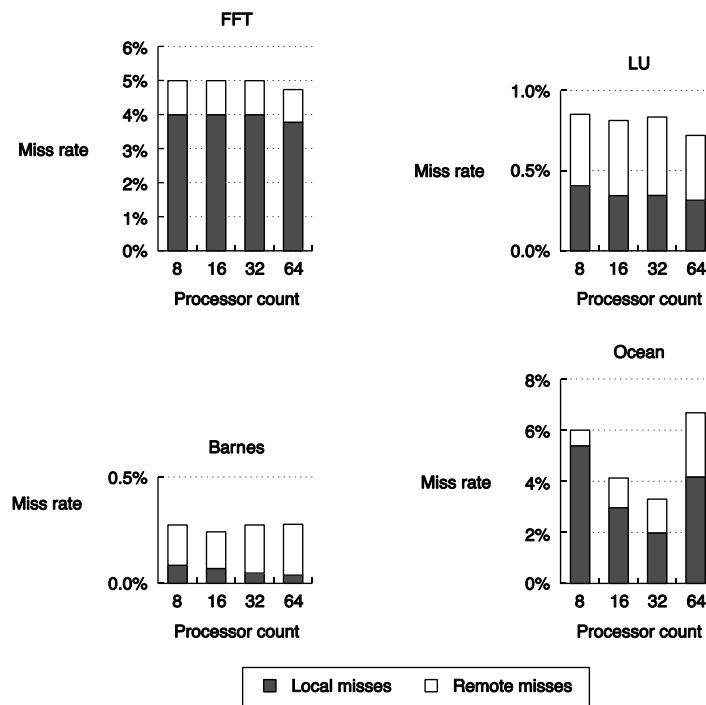
pourrait s'y attendre à une réduction du taux de défauts de cache bien que l'on constate qu'il n'existe pas une taille de bloc optimale pour les 4 applications. Il faut d'autre part lorsque l'on augmente la taille du bloc de cache prendre compte de l'impact sur l'augmentation du trafic sur le bus de communication.



© 2003 Elsevier Science (USA). All rights reserved.

Figure 109 Impact sur le trafic sur le bus de l'augmentation de la taille du bloc

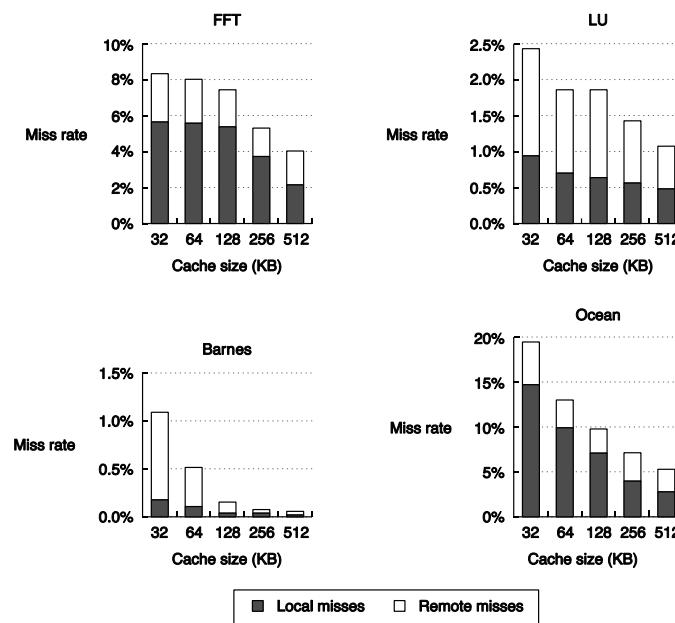
Dans les architectures distribuées une analyse de l'impact du nombre de processeurs



© 2003 Elsevier Science (USA). All rights reserved.

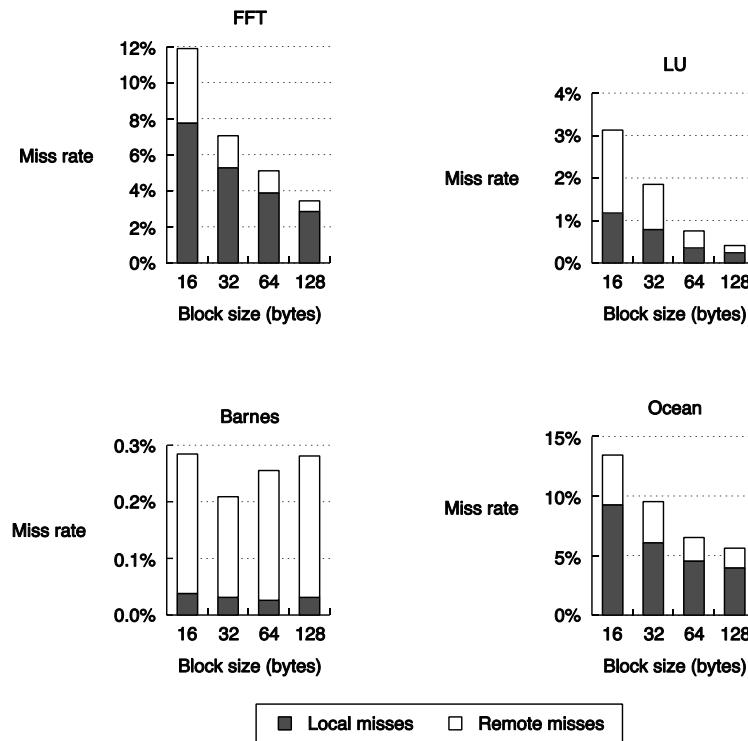
Figure 110. Analyse de l'impact du nombre de processeurs sur le défaut de cache

montre que l'augmentation du nombre de processeurs n'affecte pas significativement le taux de défaut de cache bien que l'on constate que dans l'application Ocean l'application montre un optimum pour 32 processeurs.



© 2003 Elsevier Science (USA). All rights reserved.

Figure 111. Analyse de l'impact de la taille du cache sur le défaut de cache



© 2003 Elsevier Science (USA). All rights reserved.

Figure 112. Analyse de l'impact de la taille du bloc sur le défaut de cache

L'analyse de l'impact de la taille du cache montre dans le cas des architectures distribuées une réduction des défauts de cache dus à la cohérence des données avec un impact plus important sur la dimension capacité des défauts de cache.

5.8 Evaluation de performances et dimensionnement de multicore par Simulation

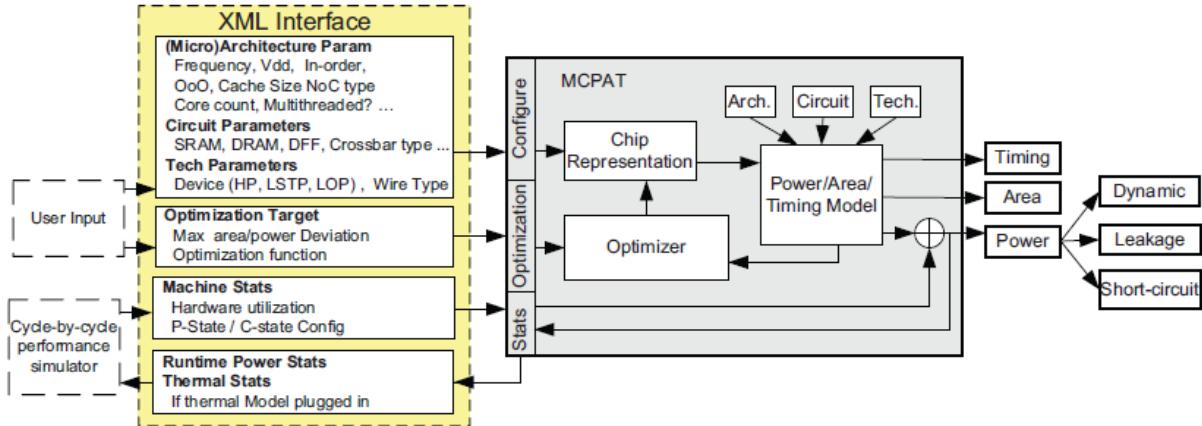
Le simulateur McPAT introduit les 3 critères principaux : 1. Surface, 2. Energie et 3. Performance.

Le dimensionnement est piloté par une fonction d'optimisation sur ces 3 critères.

Le dimensionnement doit être automatisé et parallélisé pour pouvoir explorer de manière statistiquement significative l'espace de conception du multicore.

On parle alors de **Design Space Exploration (DSE)** : exploration de l'espace de conception.

L'évaluation de performances et le dimensionnement des multicore comme pour les architectures de type superscalaire fait appel à des simulateurs (du type Gem5 www.gem5.org) et McPAT (An integrated power, area, and timing modeling framework for multicore and manycore architectures www.hpl.hp.com/research/mcpat/).



Block diagram of the McPAT framework.

Le simulateur McPAT s'appuie sur des hypothèses d'architectures générales et de configurations de mémoires.

Table IV. Parameters of the Manycore Architecture across Technology Generations

Parameters	90nm	65nm	45nm	32nm	22nm
Clock rate (GHz)	2.0	2.3	2.7	3.0	3.5
The number of cores	4	8	16	32	64
The number of memory controllers	2	3	4	6	8
Memory capacity per channel (GB)	2	4	4	8	8
Main memory type	DDR2-667	DDR3-800	DDR3-1066	DDR3-1333	DDR3-1600

Each memory controller has one memory channel.

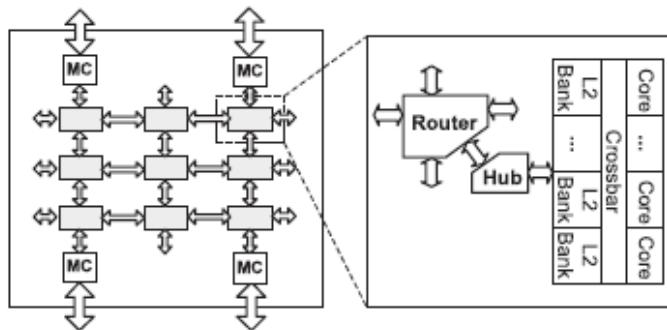
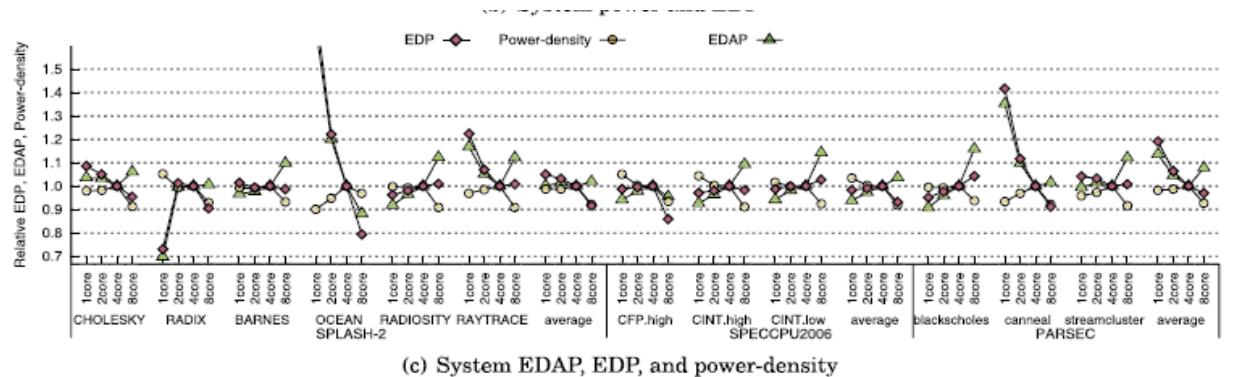


Fig. 7. The manycore system architecture. MCs refer to memory controllers.

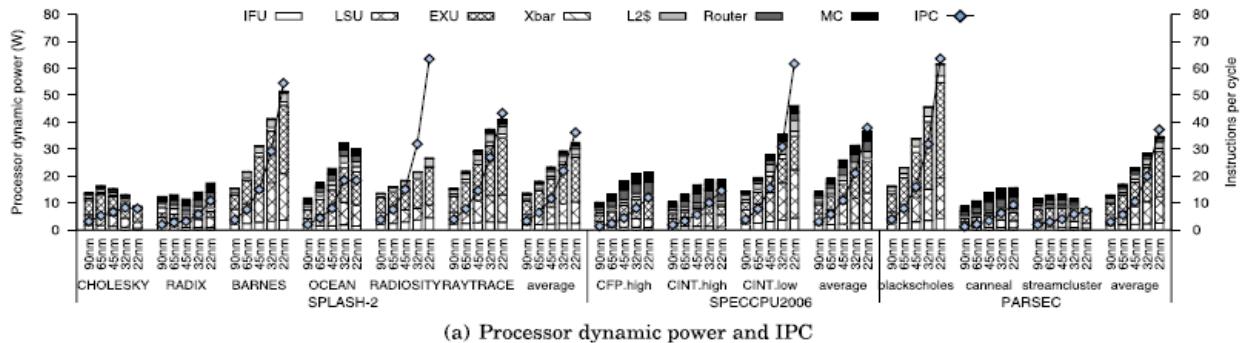
Cette architecture générique fait l'objet du dimensionnement par simulation et se base sur des données de surface et d'énergie comme indiqués dans la table suivante.

Table VI. Area and Maximum Power of Configurations with 4 Cores per Cluster across Technology Generations

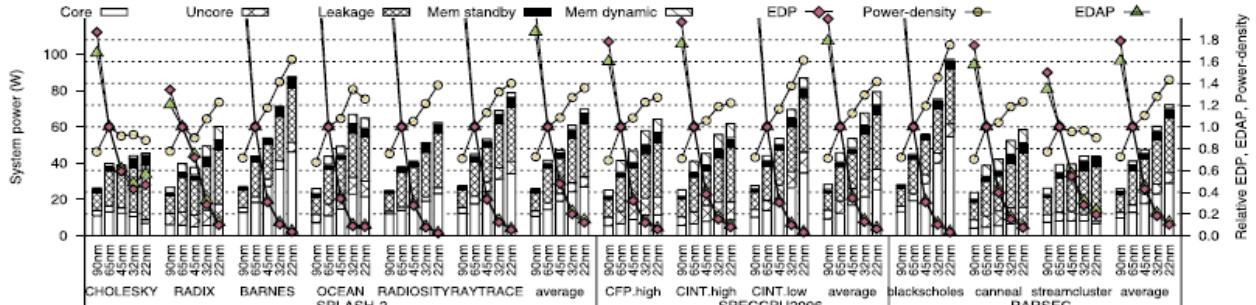
	90nm	65nm	45nm	32nm	22nm
Core area (mm²)	81.9	96.4	113.4	133.5	157.1
Uncore area (mm²)	104.3	111.3	102.7	101.6	93.5
Die area (mm²)	186.3	207.7	216.2	235.1	250.6
Max core dynamic power (W)	24.1	30.7	41.7	48.3	56.4
Max uncore dynamic power (W)	20.6	36.1	45.9	54.5	61.8
Total subthreshold leakage (W)	6.5	11.2	17.6	21.5	25.8
Total gate leakage (W)	2.6	6.7	0.7	1.6	2.5
Chip max power (W)	53.8	84.8	106.0	125.9	146.7



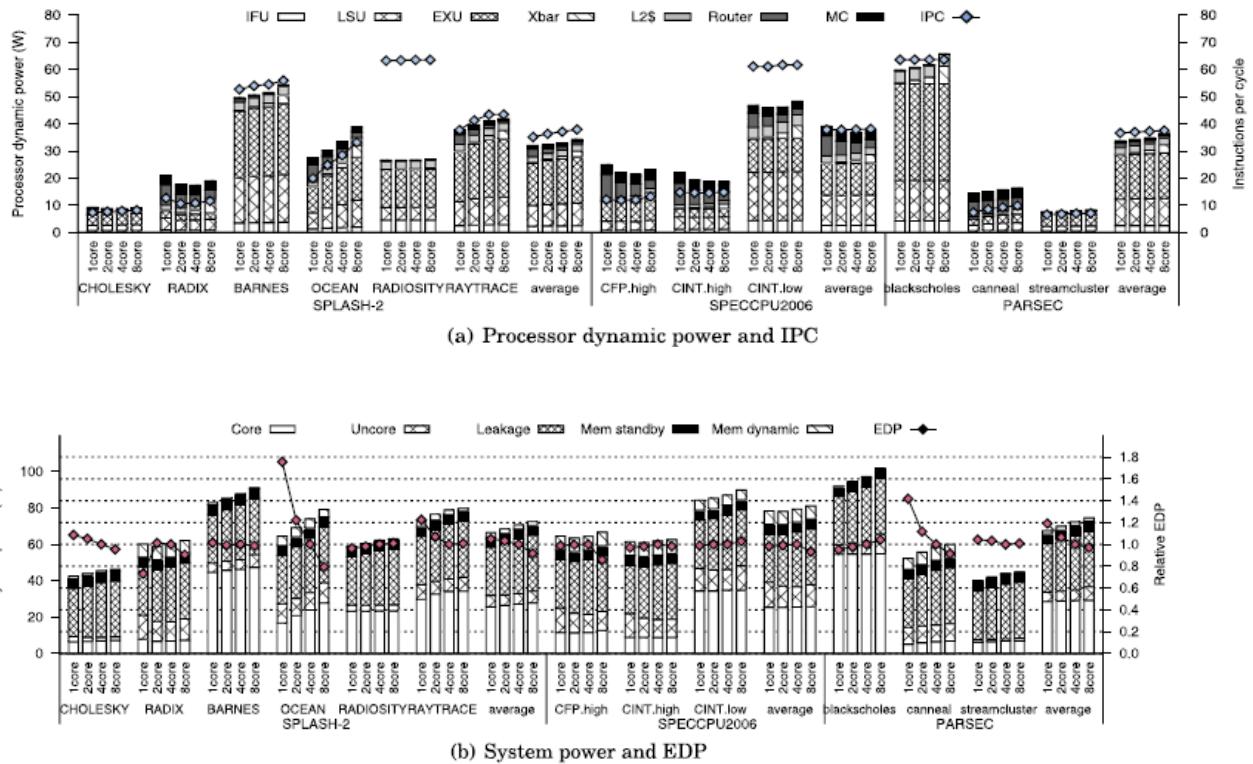
(c) System EDAP, EDP, and power-density



(a) Processor dynamic power and IPC

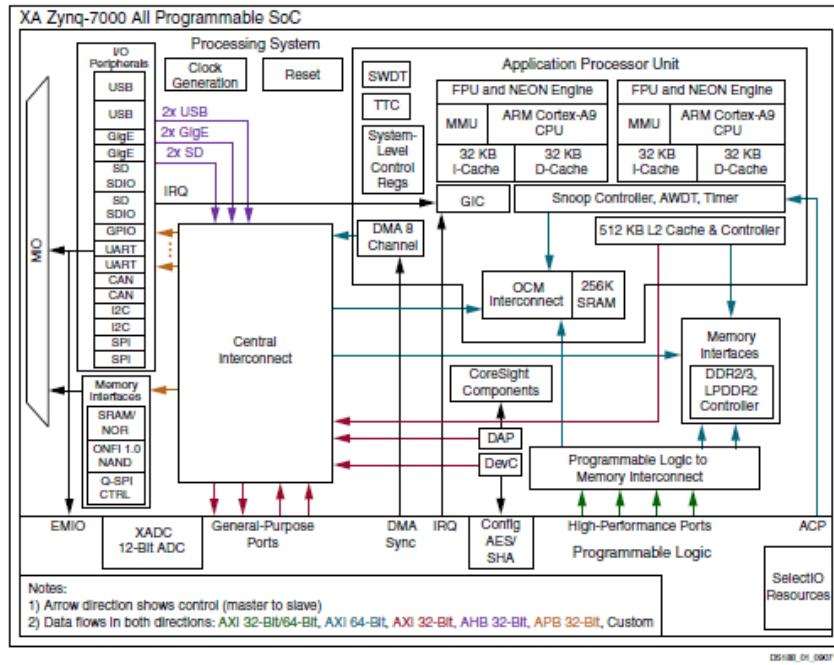


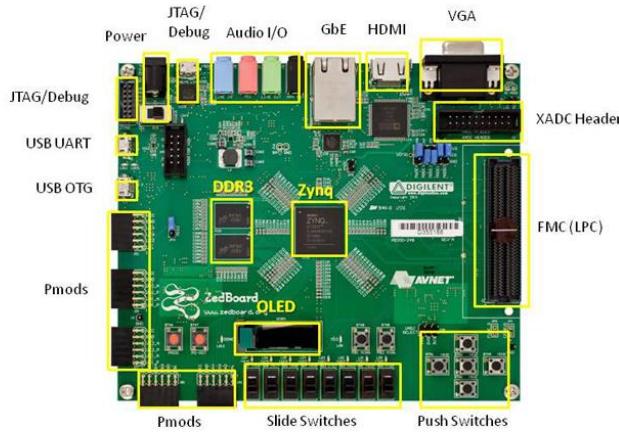
(b) System power, EDP, and EDAP



5.9 Multicore dans les systèmes embarqués

Les multicore se sont généralisés dans les systèmes embarqués et des circuits comme les circuits Zynq (Xilinx) inclut un double cœur Arm 9.



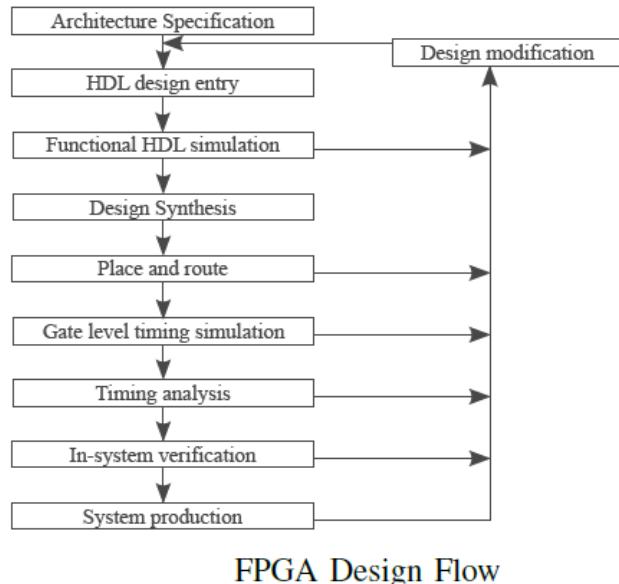


* SD card cage and QSPI Flash reside on backside of board

De ce fait des systèmes très efficaces sont disponibles pour des applications embarquées nécessitant un puissance de calcul significative.

5.10 Evaluation de performances par émulation FPGA

L'évaluation des processeurs et leur dimensionnement se heurte au « mur de la simulation ».



A fur et à mesure de la croissance de la complexité des processeurs les exigences en simulation s'accroissent de manière problématique. La simulation de processeurs complexes avec une précision suffisante dans les performances ne peut être suffisante pour des applications avec plusieurs gigacycles. Plusieurs ordres de grandeur peuvent être obtenus en passant de la simulation à l'émulation et au prototypage. Les méthodes sont alors différentes et s'appliquent à des processeurs déjà décrits dans un langage de description matériel de type VHDL, Verilog HDL. Le processeur est alors implémenté physiquement sur un circuit électronique reconfigurable le FPGA (Field Programmable Gate Array). Plusieurs constructeurs proposent ce type de circuit avec principalement Altera et Xilinx. Le processeur est donc physiquement opérationnel et peut exécuter des programmes à une vitesse largement supérieure à celle pouvant être obtenue sur un simulateur. Cette technique nécessite néanmoins des émulateurs très coûteux.

5.11 Conclusion

Les points fondamentaux des architectures multicores sont : (1) l'organisation de la mémoire et la gestion de la cohérence des caches (2) les réseaux d'interconnexions et (3) les techniques de synchronisation.

La complexité des microarchitectures des processeurs utilisés pour concevoir des multicores est naturellement un facteur important à prendre en compte mais globalement les performances sont dominées par les problématiques de la hiérarchie mémoire et des techniques de communication entre processeurs. La surface silicium des architectures multicores reste largement dominée par la surface mémoire.

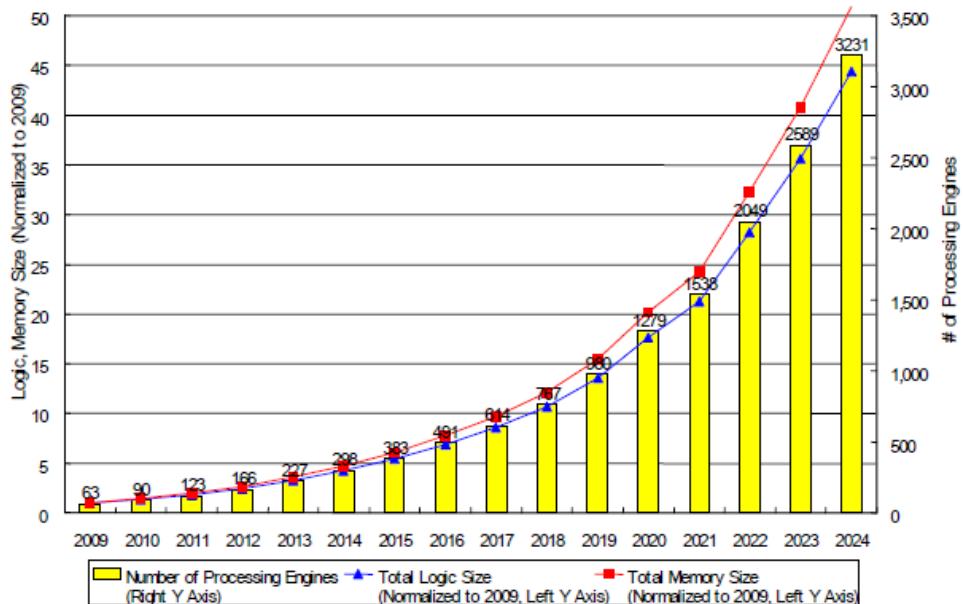
6. Conclusion

Les microprocesseurs ont connus un développement spectaculaire dans leurs performances. Celles-ci ont pu être possible par le progrès exponentiel des semiconducteurs mais aussi par de meilleures architectures et une meilleure compréhension des différentes interactions entre les microprocesseurs et le logiciel.

Cette performance s'est globalement faite de manière transparente pour l'utilisateur et ce pour préserver la compatibilité binaire. Ce n'est donc pas surprenant que l'effort essentiel se soit porté initialement sur le parallélisme au niveau instructions (Instructions Level Parallelism – ILP) car ce mode de parallélisme est celui qui conserve le modèle standard de programmation monoprocesseur. Néanmoins le parallélisme d'instructions a ses limites et il devient indispensable d'exploiter le parallélisme au niveau des threads (Thread Level Parallelism – TLP). Pour les nombreux nouveaux codes écrits dans des applications pouvant clairement bénéficier a priori de l'exécution en parallèle de plusieurs threads comme dans les applications de type serveur la compatibilité binaire n'est pas un problème essentiel. De ce fait le TLP combiné avec l'ILP apporte deux degrés de parallélisme au cours de l'exécution sur microprocesseur.

Ce TLP peut s'exécuter sur des processeurs permettant l'exécution simultanée de threads ou l'on peut aussi chercher la performance fournie par les threads sur des architectures multiprocesseurs sur puce. En ce sens, les systèmes embarqués ont été précurseurs en étant les premiers à fournir des multiprocesseurs sur puce. Les méthodologies associées de conception de système sur puce (System on Chip) basées non seulement sur les IPs de processeurs mais aussi les infrastructures de bus et de périphériques ont permis aux concepteurs d'accéder très rapidement a des technologies clés que les processeurs pour desktop n'auraient jamais permis.

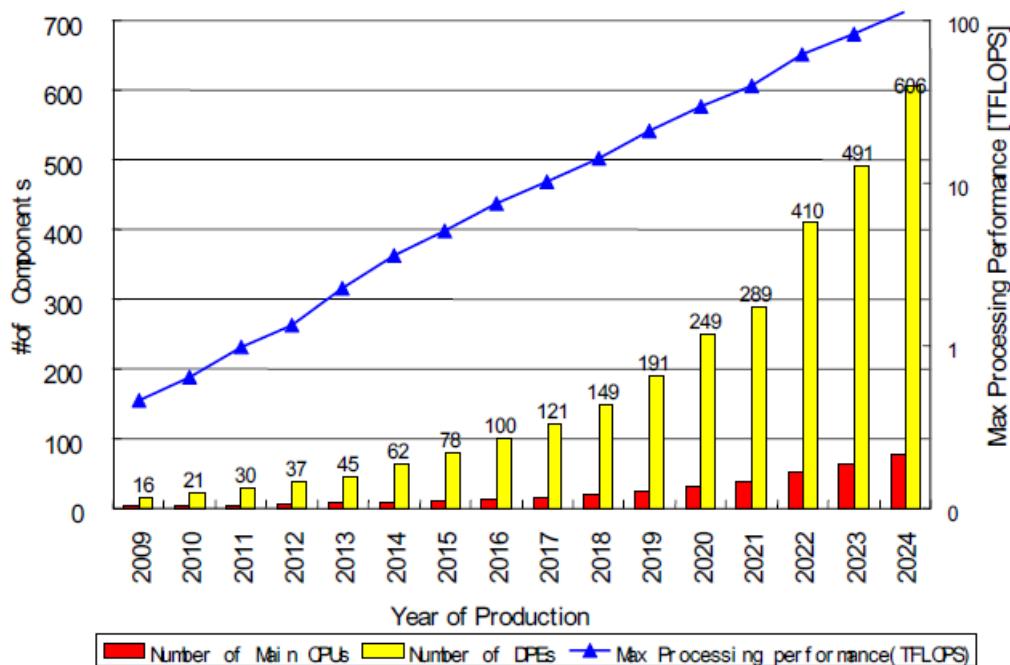
L'ère des multicore est maintenant établie et le défi se trouve à présent dans les environnements de programmation parallèle et une éducation systématique à l'algorithme parallèle. Une bonne compréhension des architectures, de leur richesse et diversité en parallélisme devient indispensable pour en exploiter au mieux les capacités. La multiplication des cores va se poursuivre comme le moyen le plus accessible d'obtenir des performances supplémentaires et par conséquent une algorithmique parallèle efficace sera inévitable.



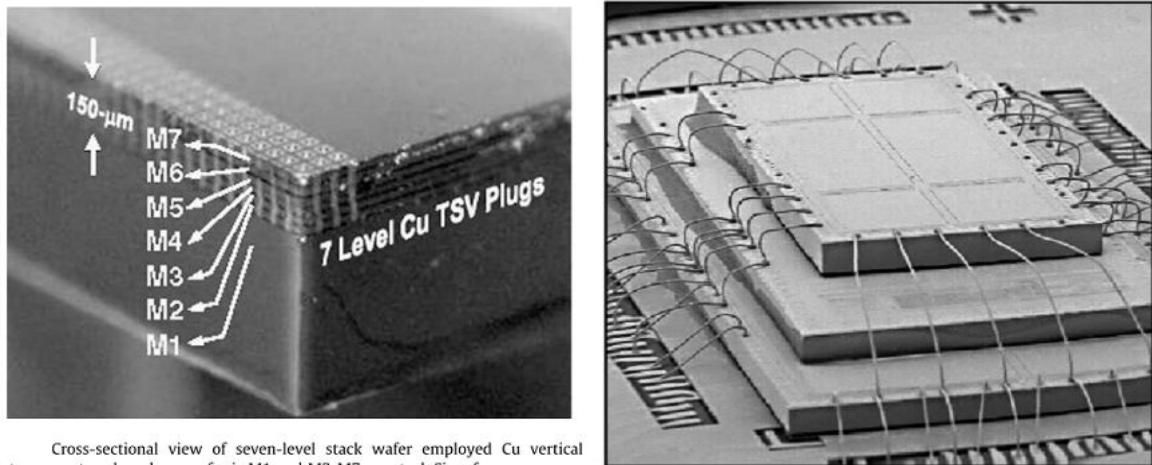
En effet l'ITRS prévoit que la complexité des futures architectures va croître de manière exponentielle et que le parallélisme sera massif.

Ce parallélisme devra s'accompagner par un accroissement de performance effectif sur des centaines de composants.

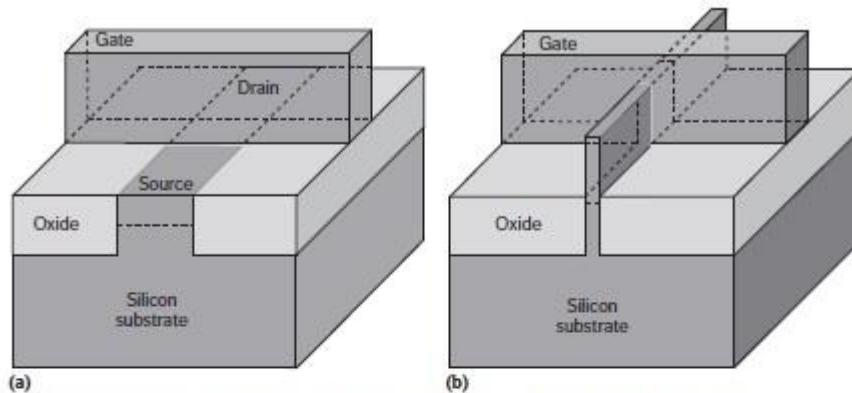
L'efficacité de ces machines sera donc directement liée aux méthodologies de programmation.



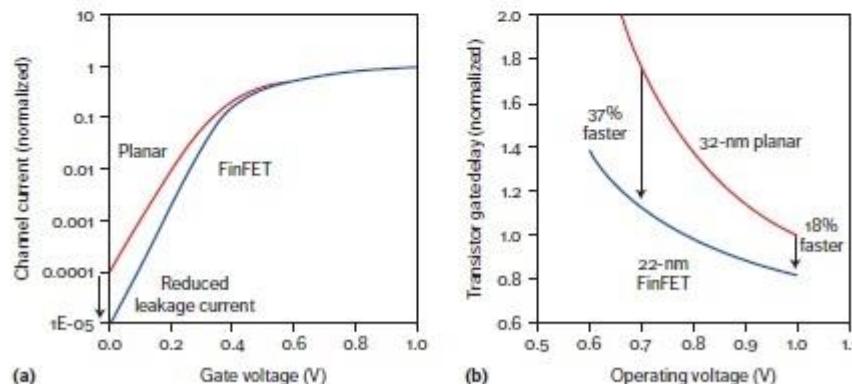
Les prochains défis viendront de la microélectronique et de la physique du semiconducteur et plus particulièrement de la technologie 3D.



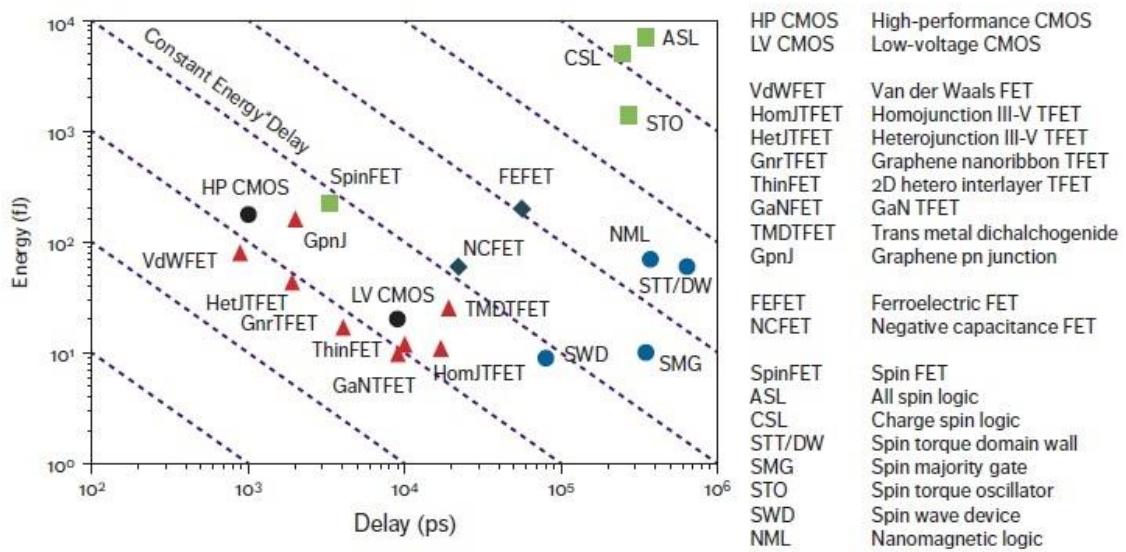
A ce stade cette technologie est en émergence et présente de nombreuses potentialités dont la réduction de la consommation d'énergie.



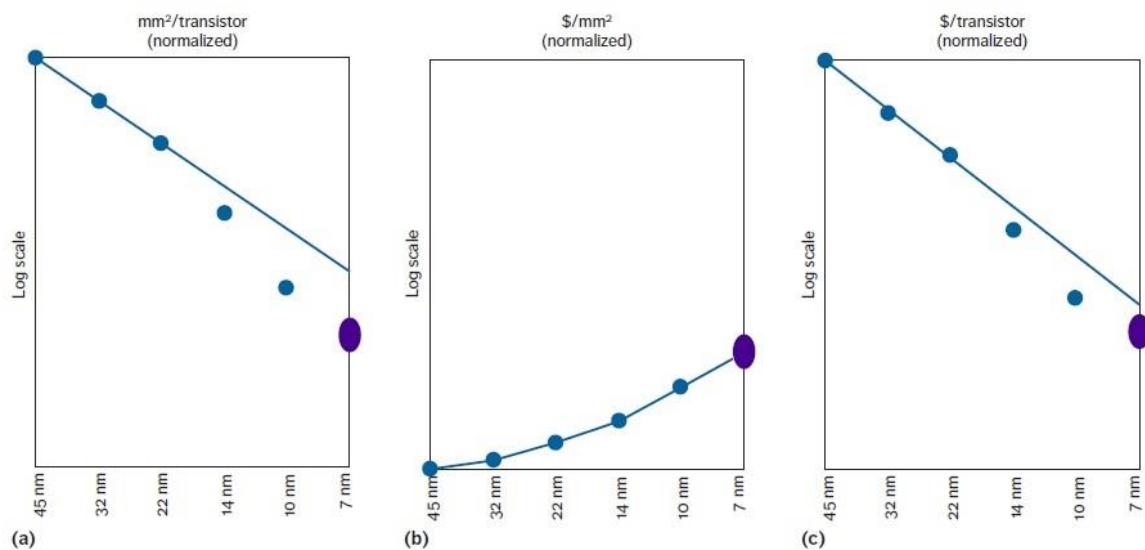
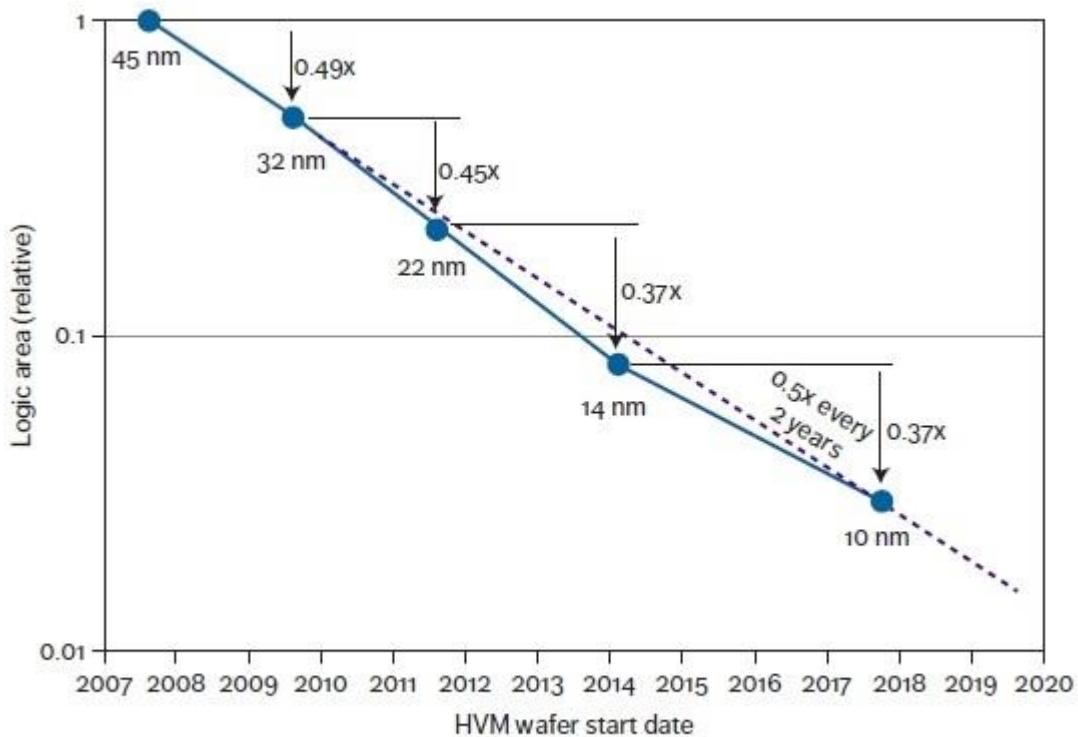
Comparison of transistor structures. (a) Planar transistor. (b) FinFET transistor.



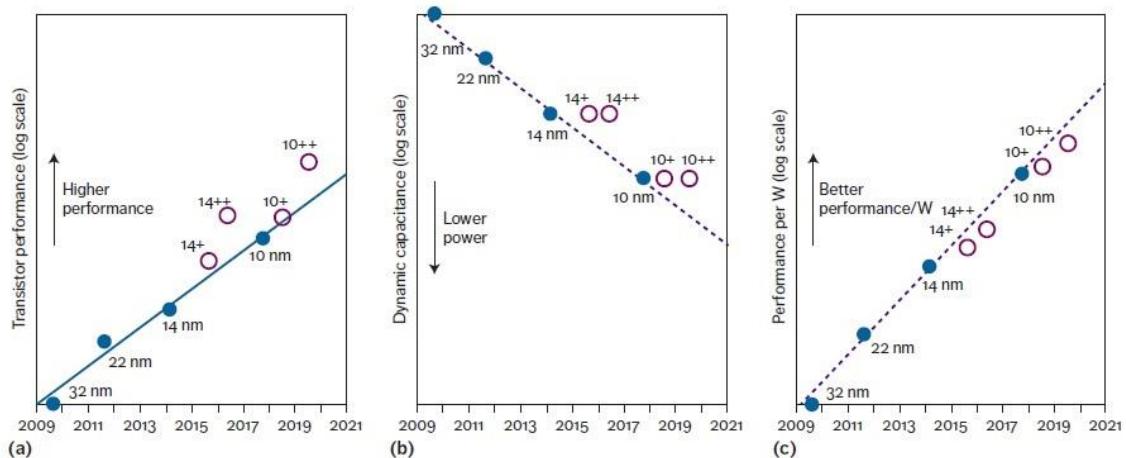
Comparison of planar versus FinFET transistor electrical characteristics. (a) Channel current versus gate voltage. (b) Transistor gate delay versus operating voltage.



Simulated switching energy and delay for a 32-bit arithmetic logic unit circuit for CMOS and for various beyond-CMOS device options.



Trends for improving logic transistor area and cost per transistor. (a) Area per transistor. (b) Wafer cost. (c) Cost per transistor.



Trends for improving transistor performance and reducing active power. (a) Transistor performance. (b) Dynamic capacitance. (c) Performance per watt.

7. Références

- [1] John H. Kelm, Daniel R. Johnson, Matthew R. Johnson, Neal C. Crago, William Tuohy, Aqeel Mahesri, Steven S. Lumetta, Matthew I. Frank, Sanjay J. Patel, Rigel: An Architecture and Scalable Programming Interface for a 1000-core Accelerator, International Symposium on Computer Architecture (ISCA'09), June 2009.
- [2] EEMBC, MultiBench™ 1.0 Multicore Benchmark Software , http://www.eembc.org/benchmark/multi_sl.php
- [3] Christian Bienia and Sanjeev Kumar and Jaswinder Pal Singh and Kai Li, The PARSEC Benchmark Suite: Characterization and Architectural Implications, Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, October 2008
- [4] C. Bienia and K. Li. PARSEC 2.0: A New Benchmark Suite for Chip-Multiprocessors In *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation, June 2009.* <http://parsec.cs.princeton.edu/>
- [5] Rusu, S. Tam, S. Muljono, H. Stinson, J. Ayers, D. Chang, J. Varada, R. Ratta, M. Kottapalli, S. Vora, S. A 45 nm 8-Core Enterprise Xeon® Processor, *Solid-State Circuits, IEEE Journal of* Volume: 45 Issue:1 page(s): 7 – 14, Jan 2010.
- [6] Rattner, J.; [The dawn of terascale computing](#), *Solid-State Circuits Magazine, IEEE* Volume: 1 , Issue: 1 2009 , Page(s): 83 – 89
- [7] J.Hennessy and D.Patterson , *Computer Organization and Design: The Hardware/Software Interface*, Morgan Kaufmann, 3rd Edition 2004
- [8] J.Hennessy D.Patterson , Architecture des ordinateurs Une approche quantitative, vuibert 2003.
- [9] MIPS Architecture www.mips.com
- [10] Keith Cooper , Linda Torczon , Engineering a Compiler, Morgan Kaufmann; (December 2003).
- [11] Ken Kennedy , Randy Allen Optimizing Compilers for Modern Architectures: A Dependence-based Approach , Morgan Kaufmann; 1st edition (October 22, 2001)
- [12] AMD Multicore Technology <http://multicore.amd.com/en/>
- [13] AMD Athlon 64 Processor Technical Documentation http://www.amd.com/us-en/Processors/TechnicalResources/0,,30_182_739_7203,00.html
- [14] IA-32 intel Architecture Optimization Manual <http://www.intel.com/design/Pentium4/manuals/>
- [15] The software optimization cookbook, Intel.
- [16] Intel® Pentium® 4 Processor with 512-KB L2 Cache on 0.13 Micron Process and Intel® Pentium® 4 Processor Extreme Edition Supporting Hyper-Threading(1) Datasheet
- [17]http://www.intel.com/design/Pentium4/datashts/index.htm?id=ipp_dlc_procp4phtxe+prod_datasheet&
- [18] Desktop Performance and Optimization for Intel(R) Pentium(R) 4 Processor www.intel.com
- [19] Richard Gerber, Andrew Binstock ,Programming With Hyper-Threading Technology, Intel Press , Jan. 2004
- [20] The IBM-SONY-TOSHIBA Cell Project http://www.research.ibm.com/cell/cell_chip.html
- [21] Desktop 4th Generation Intel® Core™ Processor Family, Desktop Intel® Pentium® Processor Family, and Desktop Intel® Celeron® Processor Family - December 2013
- [22] Digital Design and Computer Architecture: ARM Edition 1st Edition Morgan Kaufmann; 1 edition (May 6, 2015)
- [23] Embedded Systems: ARM Programming and Optimization 1st Edition Morgan Kaufmann; 1 edition (October 9, 2015)

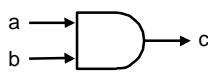
8. Appendice A – Rappel sur la logique

Cet appendice est un rappel très bref des fonctions logiques de base en électronique numérique utilisées pour l'implémentation des fonctions logiques combinatoires et séquentielles et par extension des machines séquentielles. Un processeur est une machine séquentielle synchrone.

- **Portes logiques**

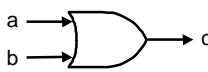
Les portes logiques AND, OR, INV permettent de construire n'importe quelle fonction logique combinatoire en utilisant l'algèbre de Boole. Les portes AND et OR représentées ci-dessous sont à deux entrées mais peuvent facilement être étendues à n entrées.

1. AND gate ($c = a \cdot b$)



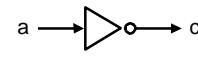
a	b	$c = a \cdot b$
0	0	0
0	1	0
1	0	0
1	1	1

2. OR gate ($c = a + b$)



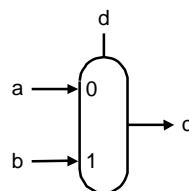
a	b	$c = a + b$
0	0	0
0	1	1
1	0	1
1	1	1

3. Inverter ($c = \bar{a}$)



a	$c = \bar{a}$
0	1
1	0

4. Multiplexor
(if $d == 0$, $c = a$;
else $c = b$)



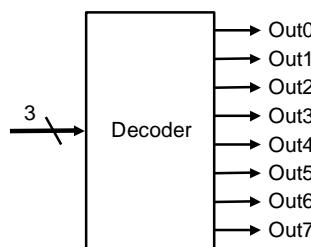
d	c
0	a
1	b

Enfin les portes NAND et NOR sont des portes logiques construites respectivement en couplant un AND (resp. . OR) et un INV.

Toute l'arithmétique classique des microprocesseurs est effectuée à partir de portes élémentaires.

- **décodeur**

Un décodeur est une fonction logique permettant d'activer une et une seule sortie pour chaque code présenté en entrée.



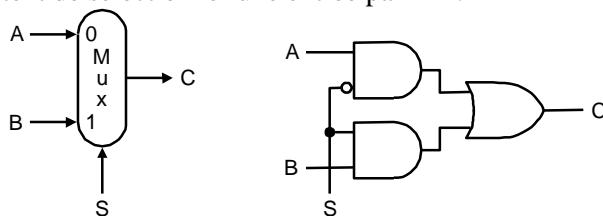
a. A 3-bit decoder

entrées			sorties							
I2	I1	I0	Out7	Out6	Out5	Out4	Out3	Out2	Out1	Out0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

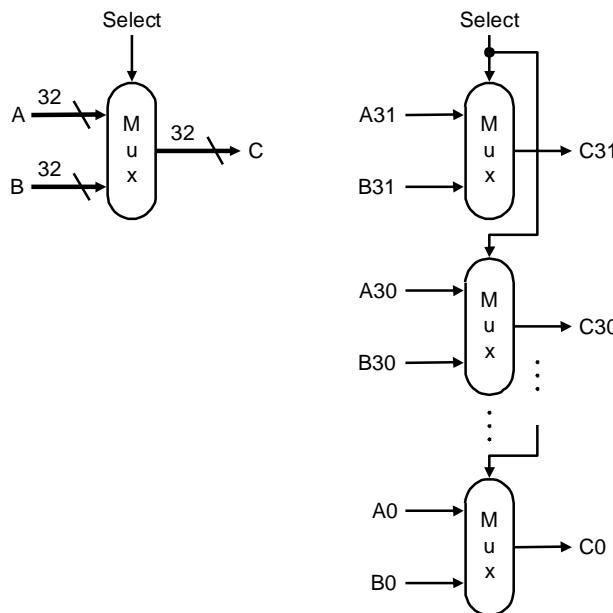
Les décodeurs sont utilisés de manière intensive dans le décodage des adresses mémoires.

- **multiplexeurs 1 bit**

Les multiplexeurs permettent de sélectionner une entrée parmi n.



- **multiplexeurs 32 bits**

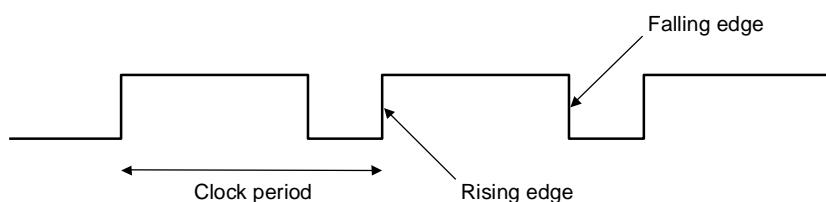


a. A 32-bit wide 2-to-1 multiplexor

b. The 32-bit wide multiplex is actually an array of 32 1-bit multiplexors

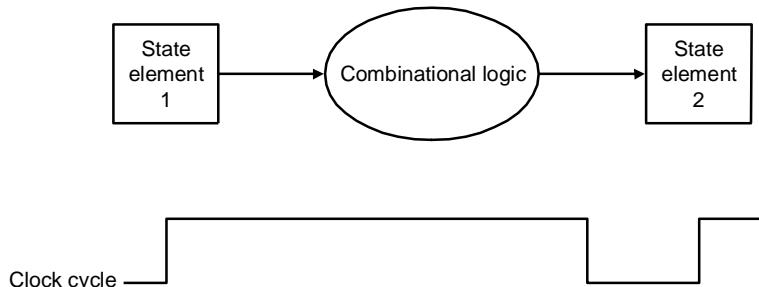
- **Horloge**

Les systèmes synchrones utilisent une horloge qui est un signal périodique défini par sa période (sa fréquence).

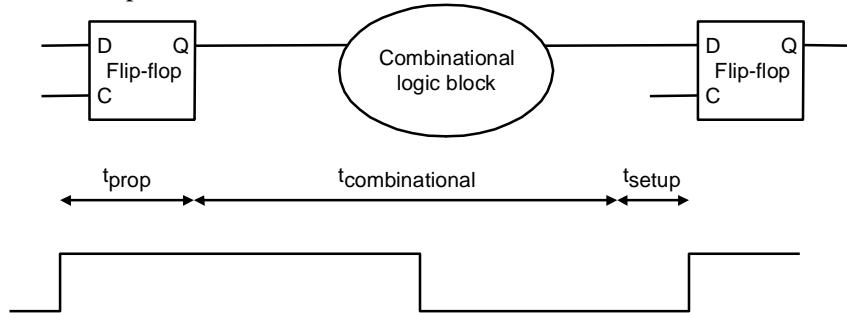


Cette période est divisée en deux : (1) une partie haute (signal à 1) (2) une partie basse (signal à 0). Si la durée de la partie basse est égale à la durée de la partie haute alors l'horloge est dite *symétrique* sinon elle est dite *asymétrique*.

Une horloge possède un front montant (*rising edge*) et un front descendant (*falling edge*). Le front actif (*active edge*) est le front d'horloge sur lequel un changement d'état s'effectue. Le fait que le changement d'état s'effectue sur un front plutôt que sur un niveau est un choix de conception et répond à une approche basée sur le front d'horloge (*edge triggered clocking methodology*). Dans la figure suivante, les front montants de l'horloge activent les états en tenant compte du délai de propagation des signaux à travers la logique combinatoire entre l'élément 1 et l'élément 2.

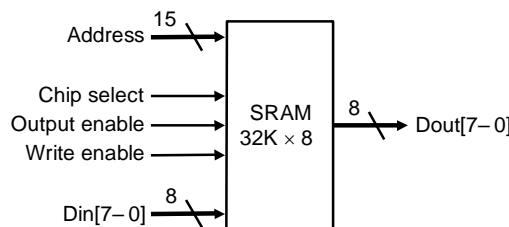


La durée maximale de la logique combinatoire (temps de propagation des signaux à travers les couches logiques) dans un système synchrone définit la période minimale possible et donc la fréquence maximale du système. Ce chemin est le chemin critique. Les éléments d'état sont implémentés en utilisant des éléments de mémorisation : les bascules (flip-flop). Il faut alors inclure dans le temps de propagation deux paramètres temporels supplémentaires : (1) le temps de set-up (t_{setup}) et (2) le temps de propagation (t_{prop}) à travers la bascule. Dans l'implémentation suivante à base de bascule D (D flip-flop) la donnée en entrée de D doit respecter un temps de stabilisation avant de pouvoir être prise en compte et mémorisée.



Enfin la donnée en entrée de D doit être maintenue pendant une durée prédéfinie appelée (Hold time) pour qu'elle soit validée.

- **SRAM**

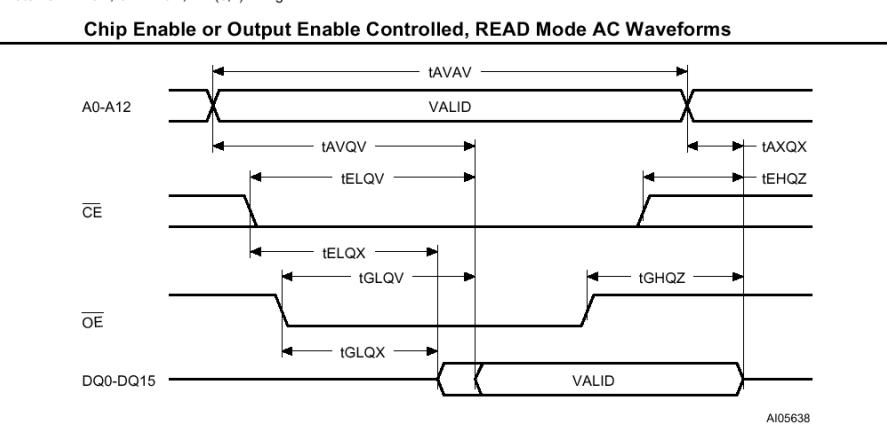
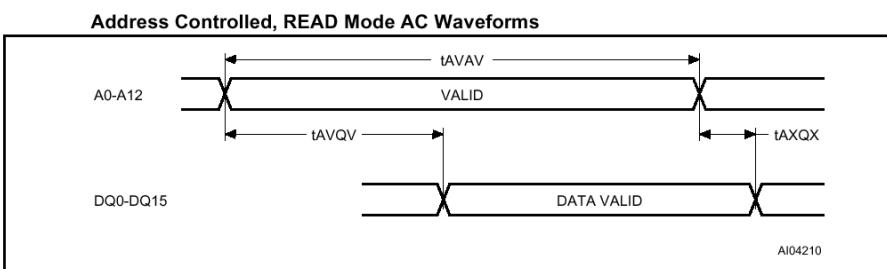
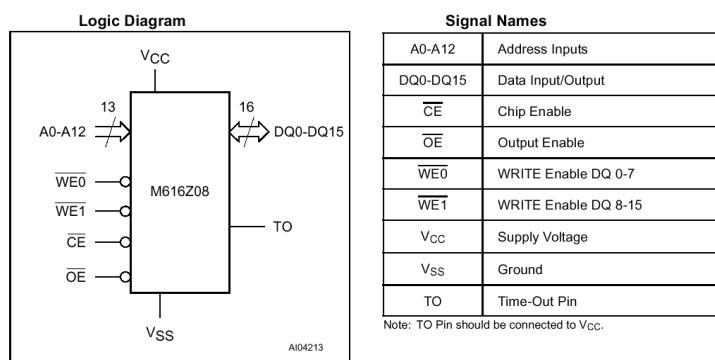


Une SRAM est une mémoire statique (Static Random Access Memory) et sert à mémoriser des données. Une mémoire statique est essentiellement caractérisée par : (1) le nombre d'éléments pouvant être

mémorisés (2) la taille des éléments pouvant être mémorisée (3) le temps d'accès à un élément (4) la durée d'un cycle de lecture (5) la durée d'un cycle d'écriture. Dans la figure ci-dessus le nombre d'éléments pouvant être mémorisés est de 2^{15} ($2^{15} = 32$ K ou 32768) et la taille de ces éléments est de 8 bits d'où la spécification de 32K x 8. Le bus adresse Address[14..0] spécifie l'élément mémoire sur lequel on souhaite effectuer une opération de lecture ou d'écriture. Le bus de données en entrée Din[7..0] permet de fournir à la mémoire une donnée à mémoriser. Le bus de sortie Dout[7..0] permet de lire de la mémoire une donnée mémorisée. Le signal Chip select permet d'activer le composant tandis que le signal Output enable permet d'autoriser la sortie d'une donnée de la mémoire. Enfin le signal Write enable permet l'écriture d'une donnée en mémoire ou sa lecture pour la valeur inverse.

Exemple de SRAM : ST M616Z08 128 Kbit (8Kbit x 16)

<http://www.st.com/stoneline/books/pdf/docs/8231.pdf>

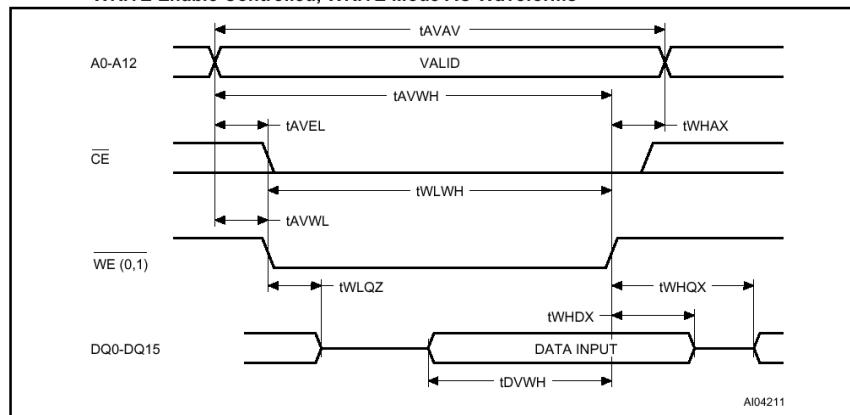


READ Mode AC Characteristics							
Symbol	Parameter ⁽¹⁾	M616Z08				Unit	
		-20					
		2.34 to 3.0V		3.0 to 3.6V			
		Min	Max	Min	Max		
tAVAV	READ Cycle Time	36		20		ns	
tAVQV	Address Valid to Output Valid		36		20	ns	
tELQV	Chip Enable Low to Output Valid		36		20	ns	
tGLQV	Output Enable Low to Output Valid		20		10	ns	
tELQX	Chip Enable Low to Output Transition	0		0		ns	
tGLQX	Output Enable Low to Output Transition	0		0		ns	
tEHQZ ⁽²⁾	Chip Enable High to Output Hi-Z		10		10	ns	
tGHQZ ⁽²⁾	Output Enable High to Output Hi-Z		10		10	ns	
tAXQX	Address Transition to Output Transition	0		0		ns	

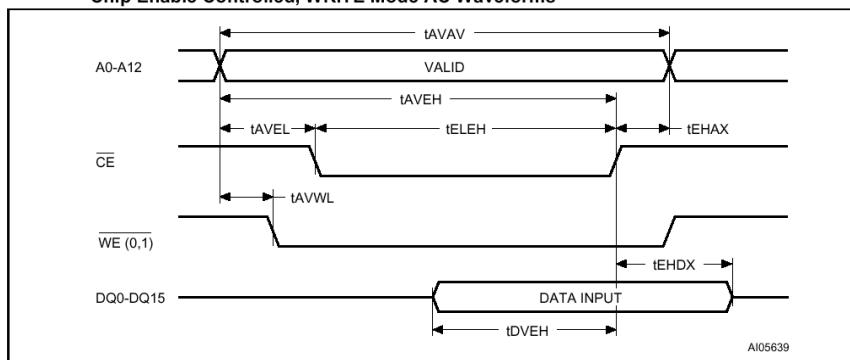
Note: 1. Valid for Ambient Operating Temperature: $T_A = -40$ to 125°C (except where noted).

2. $C_L = 5\text{pF}$.

WRITE Enable Controlled, WRITE Mode AC Waveforms



Chip Enable Controlled, WRITE Mode AC Waveforms



Note: 1. Output Enable (\overline{OE}) = High.

2. If CE goes High with WE0 or WE1 high, the output remains in a high-impedance state.

Symbol	Parameter ⁽¹⁾	M616Z08				Unit	
		-20					
		2.34 to 3.0V		3.0 to 3.6V			
		Min	Max	Min	Max		
t _{AVAV}	WRITE Cycle Time	36		20		ns	
t _{AVWL}	Address Valid to WRITE Enable Low	2		2		ns	
t _{AVWH}	Address Valid to WRITE Enable High	34		18		ns	
t _{AVEH}	Address Valid to Chip Enable High	34		18		ns	
t _{WLWH}	WRITE Enable Pulse Width	25		11		ns	
t _{WHAX}	WRITE Enable High to Address Transition	2		2		ns	
t _{WHDX}	WRITE Enable High to Input Transition	2		2		ns	
t _{WHQX⁽³⁾}	WRITE Enable High to Output Transition	0		0		ns	
t _{WLQZ^(2,3)}	WRITE Enable Low to Output Hi-Z		10		10	ns	
t _{AVEL}	Address Valid to Chip Enable Low	2		2		ns	
t _{ELEH}	Chip Enable Low to Chip Enable High	25		11		ns	
t _{EHAX}	Chip Enable High to Address Transition	2		2		ns	
t _{EHDX}	Chip Enable High to Input Transition	2		2		ns	
t _{DVWH}	Input Valid to WRITE Enable High	20		8		ns	
t _{DVEH}	Input Valid to Chip Enable High	20		8		ns	

Note: 1. Valid for Ambient Operating Temperature: T_A = -40 to 125°C (except where noted).

Appendix B – Fabricants de microprocesseurs, microcontrôleurs et DSP. (*liste non exhaustive*)



www.amd.com



www.analog.com



www.arm.com



www.fujitsu.com



www.ibm.com



www.infineon.com



www.intel.com



<http://www.kalrayinc.com/>



<https://www.qualcomm.com/>



www.nxp.com



www.renesas.com



www.samsung.com



www.st.com



<http://www.ti.com/>



www.toshiba.com

9. Glossaire

1. **Address:** une valeur utilisée pour déterminer l' emplacement dans une mémoire d' une donnée.
2. **Address translation:** processus utilisée pour transformer une adresse dans l 'espace d'adressage virtuel en une adresse physique (réel). Ce processus est effectué par une unité spécialisée la MMU (Memory Management Unit) unité de gestion de la mémoire.
3. **Addressing mode:** processus de génération d' une adresse mémoire.
4. **Amdahl's law:** une règle spécifiant que l'accroissement possible de performance d' un système obtenu par l' amélioration d' une caractéristique du système est limité par le taux d'utilisation de cette caractéristique.
5. **Assembly language:** un langage symbolique pouvant être transformé en binaire.
6. **Base addressing:** un mode d' adressage grâce auquel l'adresse d'une opérande est obtenue par la somme du contenu d'un registre et d'une adresse dans l' instruction.
7. **Basic block:** une séquence d' instructions sans instructions de branchements (sauf en dernière position) et sans destination de branchements ou d' étiquette de branchements sauf à la première instruction.
8. **Block:** l'unité d'information minimum échangé dans une hiérarchie mémoire comprenant au moins 2 niveaux (cache + mémoire centrale)
9. **Branch delay slot:** l'emplacement situé immédiatement après une instruction de branchement retardé. Cet emplacement dans les architectures de type MIPS est utilisé par une instruction qui n'affecte pas le branchement.
10. **Branch hazard:** un aléa de branchement est une situation dans laquelle l' instruction correcte ne peut pas être exécutée dans le bon cycle d'horloge car le flot d'instruction lui est incorrecte et consécutif à un branchement. Aussi appelle control hazard.
11. **Branch history table:** une mémoire qui est indexé par la partie basse de l'adresse d'une instruction de branchement et qui contient un ou plusieurs bits indiquant si le branchement a été récemment pris ou non.
12. **Branch not taken:** un branchement ou la condition de branchement est fausse et par conséquent la prochaine instruction à exécuter est l'instruction séquentiellement après l'instruction de branchement.
13. **Branch prediction:** une méthode pour la résolution des aléas de branchements qui prédit si le branchement sera pris ou non et son adresse de branchement.
14. **Branch prediction buffer:** aussi branch history table.
15. **Branch taken:** une instruction de branchement pour laquelle la condition de branchement est vérifiée et qui génère une modification du flot séquentiel des instructions.
16. **Branch target address:** l'adresse spécifiée dans une adresse de branchement qui deviendra la nouvelle valeur du PC au cas où la condition est vérifiée.
17. **Bubble:** une suspension d'exécution de pipeline dans l'objectif de résoudre un aléa. Aussi appelle pipeline stall.
18. **Bus:** un ensemble de signaux regroupés pour former un signal logique unique et utilisé principalement pour l' échange de données, le contrôle entre plusieurs sources et plusieurs destinations. Exemple : Bus PCI dans les PC.
19. **Cache coherency:** cohérence des données entre plusieurs copies disponibles dans les caches de plusieurs processeurs.
20. **Cache memory:** une petite mémoire d'accès rapide qui sert de tampon intermédiaire avant une mémoire de plus grande capacité et plus lente.
21. **Cache miss:** défaut de cache. Une requête au cache pour l' accès à une donnée qui s'avère ne pas être présente.
22. **Capacity miss:** défaut de cache du à la capacité limitée du cache.
23. **Central processing unit (CPU):** aussi appelé processeur. La partie active d'un ordinateur qui contrôle l'exécution.
24. **Clock cycle:** définit la période d'horloge du processeur (aussi clock period)
25. **Clock cycles per instruction (CPI):** nombre moyen de cycles d'horloge par instruction pour un programme ou une partie de programme.
26. **Clock period:** cf; clock cycle.
27. **Clock rate:** fréquence de l'horloge.
28. **Clock skew:** la différence temporelle séparant les instants en temps absolu avec lesquels des éléments d'état observe le front d'horloge.
29. **Cold start miss:** défaut de cache causé par le premier accès à un bloc de donnée qui n'a jamais été placé dans le cache.

30. **Collision miss:** défaut de cache existant dans les caches (direct-mapped, set-associative) lorsque plusieurs blocs référencent le même emplacement dans le cache.
31. **Commit unit:** unité dans un processeur avec pipeline dynamique ou exécutant dans le désordre chargé de valider l' exécution d'une instruction en autorisant la mise à jour des registres et de la mémoire.
32. **Compiler:** un programme traduisant un programme écrit dans un langage de haut niveau en un langage assembleur.
33. **Compulsory miss:** cf. Cold start miss.
34. **Conditionnal branch:** instruction de branchemet conditionnel.
35. **Conflict miss:** cf. Collision miss.
36. **Context switch:** changement de contexte s'applique au changement de processus dans un processeur et se traduit par une sauvegarde des informations représentant le contexte du processus. (Notion en systèmes opératoires).
37. **Control hazard:** cf. Branch hazard.
38. **Control signal:** signal utilisé pour contrôler les fonctionnalités d'une unité fonctionnelle logique.
39. **Data dependencies:** dépendances de donnée entre instructions.
40. **Data hazard:** aléas de donnée du à une dépendance.
41. **Datapath:** chemin de données d'un processeur. Partie du processeur n'effectuant pas de contrôle.
42. **Datapath element:** élément du chemin de donnée.
43. **Delayed branch:** branchemet retardé. Type de branchemet pour lequel l'instruction situé immédiatement après est toujours exécutée.
44. **Delayed load:** instruction de lecture pour laquelle l'instruction situé immédiatement après est toujours exécutée.
45. **Direct-mapped cache:** une organisation de cache dans laquelle le fonction de correspondance est basée sur le modulo.
46. **Direct memory access:** DMA un mécanisme qui permet à un contrôleur de périphériques d'accéder à la mémoire directement sans faire appel à un processeur.
47. **Directory:** structure de données permettant dans les architectures multiprocesseurs de connaître l'état de chaque bloc mémoire en activité.
48. **Dispatch:** utilisé dans les processeurs à ordonnancement dynamique pour décrire l'action d'envoyer une instruction vers la file d'attente associée à une unité fonctionnelle.
49. **DRAM:** Dynamic Random Access Memory – mémoire dynamique nécessitant un rafraîchissement cyclique de son contenu pour conserver ses données.
50. **Dynamic pipeline scheduling:** technique d'ordonnancement dynamique des instructions à l'intérieur d'une structure de processeur avec pipeline qui évite les suspensions d'exécution en allant rechercher des instructions prêtes à être exécutées situées en aval du flot d'instructions.
51. **Edge-triggered clocking:** méthodologie de séquencement des systèmes synchrones basée sur le front du signal de l'horloge plutôt que sur son niveau.
52. **Exception:** interruption d'exécution. Événement au cours de l'exécution d'un programme nécessitant un traitement associé.
53. **Finite state machine:** machine d'états finis utilisés pour représenter les systèmes séquentiels.
54. **Flush:** s'applique à l'opération d'élimination d'instructions dans un pipeline lorsque celles ne sont pas utiles cas des instructions suivant un branchemet conditionnel pris.
55. **Forwarding:** méthode de résolution des aléas de données par transmission directe des données entre étages du pipeline sans passer par le banc de registres.
56. **Fully associative cache:** une organisation de cache dans laquelle un bloc peut se placer dans n'importe quel emplacement.
57. **General purpose register:** registre pouvant être utilisé pour des données ou des adresses par pratiquement n'importe quelle instruction.
58. **Hit rate:** taux de réussite d'accès aux données d'un cache. Notion dépendante d'un programme et des données qu'il manipule.
59. **Hit time:** temps d'accès à une donnée à un niveau d'une hiérarchie mémoire temps incluant la vérification que la donnée est présente.
60. **Hold time:** cf. Rappel sur la logique.
61. **Immediate addressing:** mode d'adressage immédiat dans lequel la donnée se trouve dans un champ de l'adresse.
62. **Implementation:** instance matérielle d'une architecture abstraite.
63. **In-order commit:** validation des instructions dans l'ordre dans lequel elles apparaissent dans le flot des instructions.
64. **In-order execution:** exécution des instructions dans l'ordre d'apparition des instructions.
65. **Instruction format:** format d'une instruction (codage, découpage et signification des champs)

66. **Instruction latency:** temps d'exécution d'une instruction.
67. **Instruction mix:** mesure de la fréquence dynamique des instructions sur un ou des programmes.
68. **Instruction set:** jeu d'instructions. Ensemble des instructions comprises par une architecture donnée.
69. **Instruction set architecture:** aussi appellé architecture. Interface abstraite entre le matériel et le logiciel qui contiennent toutes les informations nécessaires pour garantir la bonne exécution d'un programme.
70. **Interrupt:** exception.
71. **Interrupt handler:** programme de traitement associé à une interruption.
72. **Jump address table:** table d'adresses ou de séquences d'instructions alternatives.
73. **Kernel benchmark:** ensemble de parties de programmes utilisés pour l'évaluation de performance.
74. **Kernel mode:** mode indiquant qu'un processus en cours d'exécution est un processus du système opératoire.
75. **Latency (pipeline):** le nombre d'étages dans un pipeline ou le nombre d'étages entre deux instructions en cours d'exécution.
76. **Least recently used:** LRU politique de remplacement utilisé dans les mémoires caches pour sélectionner une donnée à éliminer.
77. **Least significant bit:** LSB bit de plus faible poids.
78. **Level-sensitive clocking:** méthodologie de séquencement basé sur le niveau du signal de l'horloge plutôt que sur un front.
79. **Load-store machine:** architecture de processeur dans laquelle toutes les opérations de lecture et d'écriture mémoire s'effectue entre la mémoire et les registres du processeur et uniquement par des instructions de lecture ou d'écriture mémoire.
80. **Local miss rate:** taux de défaut de cache local dans une hiérarchie mémoire multiniveaux.
81. **Loop unrolling:** dépliage de boucles. Copie multiple du corps de la boucle en vue de d'exposer plus d'instructions à l'ordonnancement.
82. **Machine language:** représentation binaire du jeu d'instruction.
83. **Main memory:** mémoire principale ou centrale.
84. **Memory hierarchy:** une structure qui utilise plusieurs niveaux de mémoires de telle sorte que les mémoires soient de taille croissante et de temps d'accès croissant au fur et à mesure de l'accroissement de la distance par rapport au processeur.
85. **Miss penalty:** temps nécessaire pour lire une bloc mémoire d'un niveau de la hiérarchie mémoire au niveau précédent et qui inclut le temps d'accès, temps de transmission et temps d'insertion dans le niveau précédent.
86. **Miss rate:** taux de défaut de cache.
87. **Most significant bit:** MSB bit de fort poids.
88. **Multicycle implementation:** implémentation processeur basé sur un nombre de cycles multiples pour l'exécution des instructions.
89. **Multilevel cache:** organisation mémoire avec de multiples niveaux de caches (niveau 1 (L1), niveau 2 (L2), ...)
90. **Multiple-instruction issue:** une technique grâce à laquelle l'unité de lecture des instructions (fetch unit) peut envoyer plusieurs instructions au prochain étage du pipeline en un seul cycle d'horloge.
91. **Multiprocessor:** architecture d'ordinateur comprenant plusieurs processeurs ayant un espace mémoire partagée.
92. **Nop:** instruction n'effectuant aucun changement sur l'état d'un processeur.
93. **Object program:** programme objet
94. **Opcode:** code opération d'une instruction.
95. **Out-of-order commit:** validation de l'exécution des instructions dans un ordre qui n'est pas nécessairement l'ordre dans lequel les instructions ont été lues.
96. **Out-of-order execution:** exécution d'instruction dans un ordre différent de l'ordre dans lequel elles ont été lues.
97. **Page fault:** défaut de page (système opératoire)
98. **Page mode:** mode page (système opératoire)
99. **Page table:** table de pages (système opératoire)
100. **Physical address:** adresses physique (réelle)
101. **Physically addressed cache:** mémoire cache dont l'accès se fait avec les adresses physiques et non virtuelles.
102. **Pipeline data hazard:** aléas de données dans un pipeline.
103. **Pipeline stall:** suspension de l'exécution d'un pipeline.
104. **Pipelining:** mode d'exécution des instructions par recouvrement.
105. **Pipelining stage:** étage de pipeline.
106. **Precise interrupt:** interruption associée avec l'instruction concernée.

107. **prefetching:** technique avec laquelle les données utilisées dans un futur proche en cours d'exécution sont chargées dans le cache suffisamment à l'avance pour éviter un défaut de cache.
108. **primary memory:** mémoire principale ou centrale.
109. **program counter:** registre contenant l'adresse de l'instruction en cours d'exécution.
110. **propagation time:** temps de propagation.
111. **protection:** mécanisme de protection mémoire (système opératoire)
112. **read only memory:** Read Only Memory (ROM) mémoire en lecture uniquement
113. **reference bit:** bit utilisé pour mentionner qu'une page est utilisée et pour les gestion de ces pages par des algorithmes de type LRU.
114. **register addressing:** mode d'adressage par registre. L'opérande se trouve dans un registre.
115. **register file:** banc de registres. L'ensemble des registres situés dans un microprocesseur.
116. **rename buffer:** registre supplémentaire utilisé pour conserver des résultats en attente de validation vers des registres du banc de registres.
117. **reorder buffer:** registre conservant des instructions dans une machine avec pipeline dynamique et dont les résultats n'ont pas encore été validées. Les machines avec une exécution out-of-order et une validation dans l'ordre retireront une instruction du reorder buffer que si l'instruction a fini son exécution et que toutes les instructions précédentes aussi.
118. **return address:** adresse de retour utilisée lors d'une appel de fonction pour retourner au code principal.
119. **segmentation:** technique de découpage de la mémoire en parties variables par opposition au système par pages (système opératoire)
120. **set-associative cache:** organisation de cache .
121. **sign-extend:** extension de signe utilisée lors de l'élargissement d'une donnée signée par duplication du bit de signe sur la partie rajoutée en fort poids.
122. **SIMD:** Single Instruction Multiple Data – Instruction unique données multiples technique d'exécution parallèle consistant à appliquer la même instruction à plusieurs données en même temps.
123. **single-cycle implementation:** implémentation où toutes les instructions s'exécutent en un cycle d'horloge.
124. **spatial locality:** localité spatiale des données décrivant la situation où l'accès à des données créée une forte probabilité d'accès à des données proches spatialement au sens de l'espace d'adressage.
125. **SPEC benchmark:** ensemble de programmes utilisés pour l'évaluation de performances. www.spec.org
126. **speculative execution:** exécution spéculative – technique pipeline combinant la prédition de branchement avec l'ordonnancement dynamique.
127. **split cache:** caches séparés pour les données et le code.
128. **SRAM:** Static Random Access Memory – Mémoire à accès aléatoire statique.
129. **structural hazard:** aléas de structure – conflit d'accès à une ressource matérielle du processeur entre instructions lors de l'exécution et empêchant l'exécution de certaines d'entre elles.
130. **superscalar:** une technique avancée de pipeline qui permet d'exécuter plus d'une instruction par cycle.
131. **synchronization:** technique de coordination de plusieurs processus.
132. **synchronous bus:** bus synchrone
133. **synchronous system:** système synchrone.
134. **system call:** appel système – instruction spécialisée du jeu d'instructions permettant le passage du mode utilisateur au mode superviseur pour l'accès aux ressources systèmes.
135. **tag:** identificateur de données.
136. **temporal locality:** localité temporelle des données spécifiant que l'accès à une donnée créée un forte probabilité d'un accès ultérieur à ces données.
137. **transfer time:** temps de transfert d'une donnée.
138. **translation lookaside buffer:** TLB – cache spécialisé permettant de conserver les adresses virtuelles et physiques associées les plus récentes lors d'une traduction d'adresses.
139. **valid bit:** bit validant un bloc de cache.
140. **virtual address:** adresse virtuelle de l'espace d'adressage du processus (système opératoire)
141. **virtual memory:** mémoire virtuelle (système opératoire)
142. **workload:** ensemble de programmes représentatifs des applications s'exécutant sur un ordinateur particulier.
143. **write-back:** technique de mise à jour des données des caches vers le niveau de la hiérarchie suivant.
144. **write buffer:** file d'attente conservant les données en attente d'écriture vers la mémoire.
145. **write through:** technique de mise à jour des données des caches vers le niveau de la hiérarchie suivant.

