République Algérienne Démocratique et Populaire
Ministère de l'enseignement Supérieur et de la Recherche Scientifique

Université de Batna

Faculté des Sciences de l'Ingénieur

Département d'électronique

Mémoire

Pour Obtenir le diplôme de

Magister en électronique

Option : *Microélectronique, IC Design*

Présenté et soutenu publiquement par :

Mr BELKHEIRI Ahmed
Ingénieur d'État en Électronique de l'Université de Laghouat

Thème

Etude et application d'un système à base du processeur ARM (*Advanced RISC Machine)*

Soutenu le 18/04/2007 .

Devant le Jury composé de :

Dr. F. HOBAR	Prof. U. Constantine	Président
Dr. N. BOUGUECHAL	Prof. U. Batna	Rapporteur
Dr. Z. DIBI	M.C. U. Batna	Examinateur
Dr. A. BENHAYA	M.C. U. Batna	Examinateur

Promotion 2003

Dédicaces

A la mémoire de mon cher frère ABBA Slimane. Que Dieu le tout-puissant accorde au défunt Sa Sainte Miséricorde et l'accueille en Son Vaste Paradis …

A mes chers parents qui méritent toutes les expressions de remerciement, de considération et de reconnaissance…

A celle qui m'a tout donnée…et qui m'a comblé de tendresse et de bonheur; à ma chère femme M…

A mes gosses Mahdi Salah-Eddine et Mohammed dheaâ-eddine…

A mes très chers frères et sœurs pour leurs aides précieuses et leurs encouragements…

A tous mes ami(e)s du laboratoire de microélectronique, de la cité universitaire de Kechida, et d'Elassafia…

A l'ensemble du personnel de la TDA…

Je dédie ce modeste travail

Ahmed…

Remerciements

En premier lieu louange à dieu de nous voir éclairé sur le chemin droit… de nous avoir accordé la connaissance de la science.

Je tien à remercier très fortement le docteur BOUGUECHAL Nour-Eddine professeur à l'université de BATNA (Institut d'électronique), d'avoir accepter de m'encadrer et diriger ce travail. Qu'il trouve ici ma profonde reconnaissance pour les conseils qu'il n'a pas cessé de me prodiguer.

Mes remerciements vont aux membres de jury d'avoir accepter d'examiner et juger ce travaille :

Le professeur F. HOBAR à l'université de Constantine, pour l'honneur qu'il ma fait en acceptant de présider le jury.

Les docteurs Z. DIBI, A. BENHAYA Maîtres de conférence, à l'université de BATNA, pour avoir accepter d'examiner ce travail.

J'exprime ma vive reconnaissance envers Mr L. SAIDI directeur Adjoint à la PG à l'université de BATNA (Institut d'électronique), et touts les enseignants qui ont collaborés à notre formation sans exception

Mes sincères remerciements vont également à Mr M. BELKHEIRI chargé de cours à l'université de Laghouat, pour leur aide.

Ahmed…

Résumé

Le but de ce travail est de se familiariser et d'étudier en détail l'architecture et la structure interne du processeur ARM, l'un des processeurs très performants (haute vitesse d'exécution, faible consommation, surface de silicium assez réduite), afin d'être en mesure d'utiliser ce processeur notamment dans les systèmes sur chip (SOC) ou la consommation doit être aussi faible que possible afin d'économiser l'énergie électrique.

On a présenté dans le 1er chapitre les concepts CISC, RISC et les techniques de faible consommation. Les chapitres 2 et 3 sont consacrés pour l'étude des processeurs ARM, concernant la structure interne, les types des processeurs ARM et le jeux d'instruction. Le 4eme chapitre est dédié au développement d'un système a base du processeur ARM.

***Mots clés** : CISC, RISC, ARM, pipeline, super-scalaire, MIPS/Watt, thumb, macrcelle soft, CMOS, routage, placement, compilation de silicium, JTAG, bus AMBA.*

The purpose of this work is to get acquainted and to study in detail the architecture and the internal structure of the processor ARM, one of the very successful processors (high speed of execution, low consumption, reduced enough surface of silicon), to be capable of using this processor notably in systems on chip (SOC) or consumption must be as weak as possible to save electrical energy.

We have presented in the 1st chapter the concepts CISC, RISC and techniques of low consumption. Chapters 2 and 3 are dedicated for the study of processors ARM, concerning internal structure, types of processors ARM and instruction set. The 4th chapter is dedicated to the development of a system based on the ARM processor.

***Keywords:** CISC, RISC, ARM, pipeline, super scalar, MIPS / WATT, thumb, macrcelle soft, CMOS. Routing, placement, silicon compilation, JTAG, bus AMBA*

SOMMAIRE

CHAPITRE III : L'ARCHITECTURE DU JEU D'INSTRUCTION DU PROCESSEUR ARM

CHAPITRE IV : SUPPORT ARCHITECTURAL POUR LE DEVELOPPEMENT D'UN SYSTEME A BASE DU PROCESSEUR ARM

INTRODUCTION GENERAL

Le « micro-processeur » ou plus simplement « processeur » est le coeur des ordinateurs. Fruit de plus 50 années de recherche et de développement, le processeur détermine les capacités ainsi que les performances des machines numériques actuelles. À l'origine, outils de calcul scientifique, les ordinateurs – et donc les processeurs – sont devenus des produits « grand public » avec l'explosion des jeux vidéo et des applications multimédia. Bien que cela puisse paraître paradoxal, le modèle de fonctionnement n'a pas changé depuis son invention. Un programme est toujours constitué d'instructions qui agissent sur des données. Ces instructions sont assemblées suivant un schéma séquentiel et leurs exécutions se font au travers du processeur qui les exécute les unes après les autres. Le modèle d'exécution fait aussi appel à des unités annexes telles que la mémoire ou les contrôleurs de périphériques, dont le principe de fonctionnement n'a pas beaucoup changé lui aussi. Ce qui a par contre fondamentalement changé est la méthode d'exécution. À l'origine, les processeurs étaient d'une architecture simple et le procédé de fabrication limitait la densité d'intégration. En 1971, Intel introduit le 4004 comme première solution intégrant toutes les fonctions d'un processeur sur une seule puce. Le 4004 intègre 2300 transistors et fonctionne à 740 KHz. Trente ans plus tard, les processeurs de la dernière génération intègrent 120 millions de transistors et fonctionnent à 3 GHz. [1] Cette amélioration constante des performances est inégalée par aucune autre branche industrielle. On retiendra que « la loi de Moore » qui prédit un doublement du nombre de transistors intégrés tous les 18 mois, s'est avérée valide jusqu'à présent. En un autre terme, on peut considérer que le gain en performance sur 30 ans est de l'ordre de « 36000 » [2]. Cette amélioration constante des performances due à l'évolution des procédés de fabrication, a entraîné, à la fois, une augmentation de la fréquence de fonctionnement et une augmentation de la densité d'intégration. Les projections les plus récentes prévoient une capacité d'intégration proche du milliard de transistors. Ce gain en densité d'intégration entraîne une consommation élevée – proche de 300W– qui représente un frein à l'implémentation de nouvelles fonctions complexes. Il en résulte que l'augmentation des performances d'un système passe à la fois par un accroissement des fonctions intégrées mais aussi par de nouvelles solutions architecturales. En particulier, ces dernières années ont vu la prolifération d'architectures super-scalaires très complexes dont le niveau de performance est assez spectaculaire. Ces architectures fortement spéculatives font appel à

des pipelines très longs qui permettent d'atteindre des fréquences d'utilisation élevées. D'autre part, de nouvelles techniques ont été explorées pour augmenter les performances sans pour autant se cantonner dans des modèles classiques d'exécution. Le résultat net de toute cette activité est le déploiement de nouvelles architectures telles que l'architecture RISC qui a donné naissance à des produits ayant plus ou moins un succès commercial. Parmi ces produit on trouve le processeur ARM notre sujet d'étude, qui a été développé originalement à Acorn Computer Limited de cambridge à l'Angleterre, entre 1983 et 1985. C'était le premier microprocesseur RISC développé pour l'utilisation commerciale et a quelques différences significatives d'architectures suivantes le RISC. Le concept de RISC est le fruit des programmes de recherche de processeur de l'universités de Stanford et Berkeley autour de 1980. En 1990 la société ARM Ltd a été fondée comme une société séparée spécifiquement pour élargir l'exploitation de technologie de ARM, depuis quand le ARM a été donné une licence à beaucoup de fabricants de semi-conducteur dans le monde entier. Il est devenu établi comme un leader du marché pour les applications incorporées de puissance réduite et de coût sensible. Cette mémoire est un travail de synthèse consacrée à l'étude de ce processeur, se présentera en quatre chapitres :

Le premier chapitre sera consacré à la présentation des concepts CISC et RISC, les circuits internes constituant un microprocesseur, et section pour décrire les approches de la puissance de circuits CMOS.

Le deuxième chapitre présentera l'organisation et la structure interne du processeur ARM, quelques types de processeurs ARM entiers, et en détail leARMTDI7.

Le troisième chapitre sera dédié à l'étude de l'architecture de jeu d'instruction ARM standard en détail, et la présentation du jeu d'instruction 16 bits thumb en bref.

Le quatrième chapitre fera l'objet d'un support architectural pour le développement d'un système à base du processeur ARM.

CHAPITRE I

GENERALITES SUR LES CONCEPTS CISC ET RISC

I.1 INTRODUCTION

Les microprocesseurs ont d'abord été développés en version CISC, soit « complex instruction set computer », ordinateur à jeu complexe d'instruction. Tel a été le cas du père de tous les microprocesseurs, le 4004 de Intel, puis des célèbres chefs de file 8008 de Intel et 6800de motorola.

Puis on s'est aperçu que la technologie RISC « *reduced instruction set computer* », ordinateur à jeu réduit d'instructions pouvait s'appliquer aux microprocesseurs également avec les plus grands bénéfices en termes de coût et de performances.

Contrairement à ce nom pourrait laisser penser, le concept RISC ne porte pas seulement sur des questions d'instructions et de programmes, loin s'en faut .car de grandes différences sont également introduites dans les architectures.

En fait, ces différences architectures sont tellement séduisantes que les processeurs CISC se les accaparent à leur tour. Tout semble même indiquer qu'on s'oriente désormais vers une convergence des architecture CISC et RISC, chacune empruntant à l'autre ce qu'elle offre de meilleur.

Nous allons présenter, dans ce chapitre la longue marche vers le RISC. Les différentes architectures utilisées dans la conception des microprocesseurs pour l'amélioration de ces performances (vitesse, consommation, coût de fabrication …). Ainsi que les autres concepts de bas d'un simple microprocesseur (16 bits) MU0, Ainsi quelques remarques pour but d'augmenter la vitesse d'exécution. On va terminer ce chapitre avec un rappelle sur les circuits internes des microprocesseurs,

.

I.2 LES INSTRUCTIONS SIMPLES ET COMPLEXES

Un microprocesseur, c'est un circuit électronique programmable. Cela signifie qu'il peut exécuter de multiples fonctions sous la commande des instructions d'un programme. Les instructions sont rassemblées en un flot linéaire et unique par le programme, même si leur exécution, ensuite, obéit à des règles qui rompent cette belle séquence initiale. Elles sont stockées dans la mémoire centrale afin de pouvoir être exécutées.

Cette exécution est cadencée par une horloge généralement externe au processeur. L'intervalle de temps s'écoulant entre un battement de l'horloge et le suivant définit le **cycle de l'horloge** on parle également d'une **période d'horloge** ou, de façon moins rigoureuse, d'un **temps d'horloge** .le nombre de période d'horloge par seconde est la **fréquence d'horloge.**

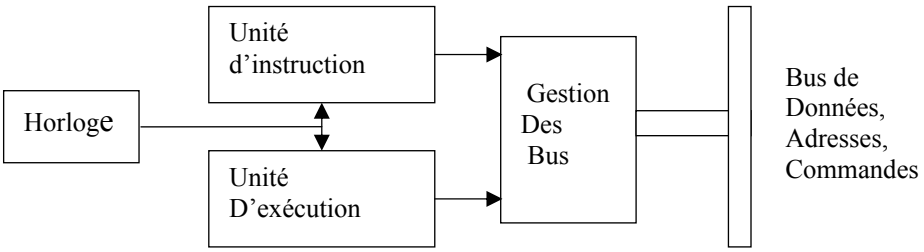


Figure I.1 Synoptique ultra -simplifié d'un microprocesseur élémentaire.

La figure I.1. Présente le schéma synoptique d'un microprocesseur élémentaire comprend :

a. une unité d'instruction
Chargée de lire les instructions l'une après l'autre dans la mémoire centrale, de les décoder, puis de les transmettre l'une après l'autre à l'unité d'exécution. Elle englobe des registres tel que le compteur ordinal, éventuellement rebaptisé d'un nom plus valorisant.

b. une unité d'exécution
Qui exécute ce que les instructions prescrivent.

c. une horloge
Partiellement ou totalement externe, qui cadence le fonctionnement de ces unités.

d. des circuits
Gérant les bus système, ceux qui sortent du microprocesseur : bus de commandes, bus de données et bus d'adresses.

Les instructions sont des codes numériques, binaires, qui doivent être décodés par le processeur pour que celui-ci sache ce qu'il doit faire. Parmi les instructions les plus courantes figurent les instructions arithmétiques, l'addition par exemple, des instructions logiques telles que le OU ou le ET logique, des instructions de rotation des bits, de décalage, etc. un instruction peut être simple ou complexe, par exemple :

- L'instruction commandant d'ajouter 1 au contenu d'un registre interne du microprocesseur est une instruction simple. Elle met en jeu un simple additionneur et tout se passe dans le processeur, en interne. Il suffit généralement d'un seul cycle d'horloge pour exécuter une telle opération.

- En revanche, l'instruction commandant l'incrémentation du contenu d'une adresse mémoire (on ajoute 1 au contenu de cette adresse) est une instruction complexe .d'abord, il faut agir sur un élément externe au processeur, la cellule en mémoire. Une telle opération implique le transfert de son contenu dans le processeur, l'addition de 1, puit l'écriture du résultat à la même adresse. A l'évidence, il va falloir y consacrer plus d'un cycle d'horloge dans le cas le plus général. Il existe des instructions encore bien plus complexes que celle-ci.

I.3 LE MICROCODE DES PROCESSEURS CISC

Une instruction simple peut être directement exécuté par le processeur. Le processeur met à sa disposition les circuits électroniques nécessaires, par exemple un additionneur. Ce dernier constitue la pierre angulaire de toute unité arithmétique et logique, ou UAL en abrégé (et ALU en anglais).

En revanche, l'exécution d'instructions complexes ne peut plus faire appel à des circuits fixes (on dit : câblés) car il faudrait multiplier leur nombre. Or, il existe une limite physique qui est le nombre de composants que l'on peut intégrer dans un microprocesseur. Si l'on voulait exécuter toutes les instructions à l'aide de circuits câblés distincts, le nombre de composants intégrés sur la puce de silicium du processeur, déjà énorme bien que des mesures rigoureuses d'économies soient prises, deviendrait vite inacceptable. Le processeur simplement impossible à fabriquer dans l'état actuel de la technique. Une autre solution a donc été adoptée.

Chaque instruction complexe est fractionnée en certain nombre de séquences minimaliste appelées micro-instructions. En assemblant ensuite ces micro-instructions à la demande d'instruction complexe d'origine, on peut reconstituer chacune de ces instructions complexes.

Ces micro-instructions sont rassemblées et enregistrées dans le silicium du processeur sous forme d'un « microcode ». Autour de ce microcode doivent être installés des circuits de logique chargés d'organiser la composition séquentielle des micro-instructions, en tenant bien évidemment compte des ordres transmis par l'unité d'instruction. Il suffit donc d'un nombre réduit de microcodes pour reconstituer une grande diversité d'instructions plus ou moins complexes. (Le microcode enregistré dans le silicium est appelé *firmware* en anglais).

C'est ce concept qui a été appliqué aux processeurs CISC, sigle provenant de l'expression « *complex instruction set computer* », soit « ordinateur à jeu complexe d'instructions ». Il procure à la fois des avantages et des inconvénients.

Le décodeur d'instruction déchiffre l'instruction qu'il reçoit et l'applique au bloc de micro-codage qui la transforme en opération élémentaires. C'est ce que représente, avec un grand souci de simplification, le schéma de la figure I.2. Cela signifie que ce microcode est enregistré une pour toutes par le fabricant des circuits et que seules, les instructions s'y référant peuvent être prises en compte.

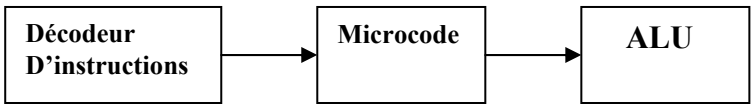


Figure I. 2 Le microcode reçoit les instructions du décodeur et les transforme en opérations élémentaires pour l'unité arithmétique et logique (ALU) du processeur.

I.3.1 Avantages

Le premier avantage, c'est qu'en utilisant des instructions complexes micro- codées, le programme d'application est plus court ; il comporte moins d'instructions puisque certaines d'entre elles sont capables d'en faire d'avantage. Si le programme est plus court, il occupe aussi moins de mémoire.

I.3.2 Inconvénients

Les instructions complexes imposent une logistique contraignante au microprocesseur. Par exemple, les temps d'exécution différent d'un instruction à l'autre ; il faudra donc systématiquement les vérifier. D'autre part, toute la vitesse d'un processeur doit s'aligner sur les temps d'exécution des instructions les plus complexes. Statiquement, on constate d'ailleurs que ces instructions complexes sont beaucoup moins utilisées que les instructions simples. Elles pénalisent ainsi l'ensemble du système. D'autre part, le microcode utilise une surface non négligeable de silicium. Or, en premier approche, le prix de revient d'un processeur s'accroît, plus les défauts de surface d'une puce risquent de se manifester et plus les rendements en production baissent. Si l'on pouvait supprimer ce microcode, on réaliserait une économie substantielle.

I.4 SUPPRESSION DU MICROCODE AVEC LE RISC

Si l'on se résout à se passer des instructions complexes, on pourra du même coup éliminer le microcode et économiser de la surface de silicium. C'est ce qui été fait avec le RISC, ces lettres provenant de l'expression « *Reduced Instruction Set Computer* », soit « ordinateur à jeu réduit d'instructions ».

Par quoi les remplacer si l'on en réellement besoin ? Deux solutions sont concevables :

1. une instruction complexe est remplacée par une série d'instructions simples exécutant les mêmes fonctions. Si l'on développe un programme en langage évolué, il faudra que le compilateur sur lequel repose tout le poids de la traduction du programme soit particulièrement efficace. Le compilateur est chargé de traduire en binaire (en code machine) des instructions rédigées en langage évolué (en C++, en pascal , en fortran , etc..) l'accent est donc mis sur la réalisation d'excellents compilateurs.

2. Si, en dépit de tout, il faut conserver quelques instructions complexes, on ne va plus les microcoder mais les câbler quand même dans le silicium. cela signifie qu'on va ajouter, dans le silicium, les quelques circuits spécifiques organisés à demeure et

capables d'exécuter ces instructions complexes ; il ne s'agit donc plus d'une microprogrammation. Certes, ces circuits occupent de la surface de silicium mais bien moins qu'un microcode. d'autre part, l'exécution d'une instruction câblée est beaucoup plus rapide que l'exécution de la même instruction microcodée.

La suppression du microcode et le câblage de touts les instructions restants offre un autre avantage : chaque instruction peut être exécuter en seul période d'horloge, ce qui constitue l'un des objectifs recherchés.

I.5 LA LONGUE MARCHE VERS LE RISC

Le concept de RISC « *Reduced Instruction Set Computer* », applique ainsi cette idée : « faire les chose simplement et vite ». Les premiers travaux sur le RISC sont dus à IBM et remonte à 1975. Cella se passe au centre de recherches Thomas J.Watson, prés de New York. Ils sont menés sous la direction de John Cocke, considéré aujourd'hui le père du RISC. [3]

John Cocke commence par analyser le fonctionnement d'un gros système IBM de l'époque, un IBM 370. Il étudie la distribution statistique de l'utilisation des instructions et constate qu'en réalité, seul un petit nombre d'instructions simples telles que LOAD, STORE, ADD, SUBSTRACT, BRANCH (en français : charger, ranger, additionner, soustraire, brancher) sont massivement exécutées, les autres instructions, en particulier celles exécutant des fonctions plus complexes, n'interviennent que rarement.

Ces premiers résultats le mettent sur la voie d'une machine simplifiée, n'ayant recours qu'aux instructions plus courantes mais fonctionnant alors plus rapidement, sans surcharge inutile. La contrepartie, c'est que il faut désormais émuler les anciennes instructions complexes par logiciel, ce qui allonge le programme et son code, mais le bilan global fait largement pencher en faveur du jeu réduit d'instructions simples.

A l'époque, le frein était plutôt représenté par le prix à payer pour les mémoires chargées de stocker le code du programme. Depuis, cet obstacle a sauté, ce qui explique en partie pour quoi le RISC a mis si longtemps à s'imposer.

Tout en étudiant des jeux simples d'instructions de bas niveau, John Cocke s'intéresse également à des compilateurs sophistiqués engendrant un code au niveau, du langage processeur,
Du binaire, suffisamment efficace pour éliminer le microcodage. En 1979, il réalise un prototype, l'ordinateur RISC modèle 801. [1]
Développant ces idées, David Paterson, alors étudiant à l'université de Berkeley, Californie, puis professeur, emploie pour la première fois ce terme « RISC ». Il reconnaît que l'instruction d'un microcode dans le silicium du processeur constitue une surcharge ralentissant considérablement la vitesse d'exécution. D'autre part, il constate que les compilateurs souffrent de l'existence d'instructions complexes.
Chez IBM, l'ordinateur 801 fait la preuve qu'il est possible à la fois de réduire le jeu d'instructions et de créer des compilateurs efficaces. Le fonctionnement parallèle en pipeline peut être amélioré et l'objectif d'une instruction par cycle d'horloge devient réalisable.
Parmi les tout premiers microprocesseurs RISC commercialisés, on peut citer les clippers C100 de *Fairchild,* le 88000 de *Motorola,* le Sparc de *sun,* l'AM 29000 de AMD, les séries de *Mips Computer,* etc…

I.6 COMPARAISON ENTRE LES PROCESSEURS RISC ET SISC

La figure I.3. Illustre le schéma synoptique du principe de fonctionnement du processeur RISC et CISC.

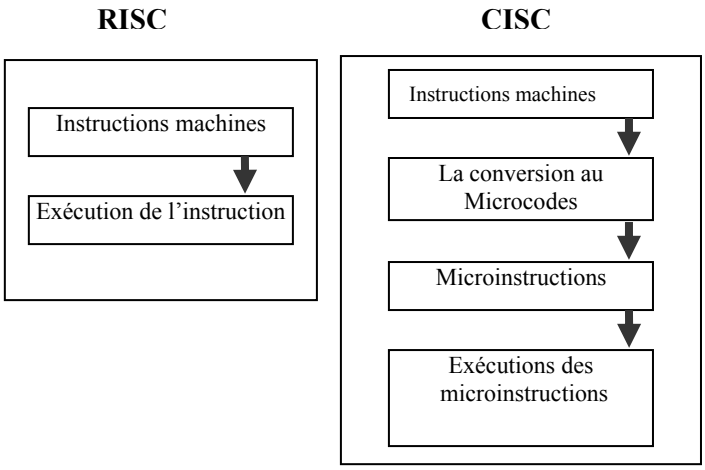


Figure I. 3 Concepts de base du RISC et du CISC.

Au milieu des années 70, deux facteurs sont ébranlés les idées dans les esprits par les décennies précédentes :

- d'une part les mémoires sont devenues plus rapides qu'elles ne l'étaient auparavant.
- d'autre part, des étude réalisées conduites sur des langages de haut niveau montrèrent que :
  - les programmes sont constitués à 85% d'affectations **if** et d'appels de procédures,
  - 80% des affectations sont de la forme **variable = valeur**

Les résultats précédents peuvent se résumer par la phrase suivante : *80% des traitements des langages de haute niveau font appel à 20% des instructions du CPU*. D'où l'idée d'améliorer la vitesse de traitement des instructions les plus souvent utilisées. [3]

Les machines RISC se distinguent des machines CISC par le fait que les instructions sont exécutées en un cycle d'horloge.

Tableau I. 1 Caractéristiques comparées des architectures RISC et CISC

RISC	CISC
Instructions simples ne prenant qu'un seul cycle	Instructions complexes prenant plusieurs cycles
Seul les instructions LAOD et STOR ont accès à la mémoire	Toutes les instructions sont susceptibles d'accéder à la mémoire
Instruction au format fixe	Instruction au format variable
Beaucoup de registres	Peu de registres
Peu de modes d'adressage	Beaucoup de modes d'adressage

Le principe qui participe à la conception des architectures RISC est simple; tout faire pour que le temps de cycle du chemin des données soit le plus court possible. Le temps de cycle du chemin des données est constitué par :

- le transfert des données entre mémoire et registres du CPU
- le transfert des données entre registre du CPU et opérandes de l'UAL
- la réalisation des calculs par l'UAL

En fait le terme réduit de l'appellation RISC est plutôt mal choisi. Si les processeurs ont moins d'instructions que les processeurs CISC c'est une conséquence directe du fait que les modes d'adressages en RISC sont beaucoup moins nombreux que ceux des processeurs CISC : de fait le système de codage des instructions par cycle d'horloge.

I.7 QUELQUES ARCHITECTURES UTILISEES DANS LA CONCEPTION DES µ P

I.7.1 Architecture uniprocesseur

I.7.1.1 Architecture de Von Neumann (1946)

Architecture conventionnelle, la plus utilisée dans le domaine des ordinateurs. Elle repose sur quatre entités principales, figure I.4 :

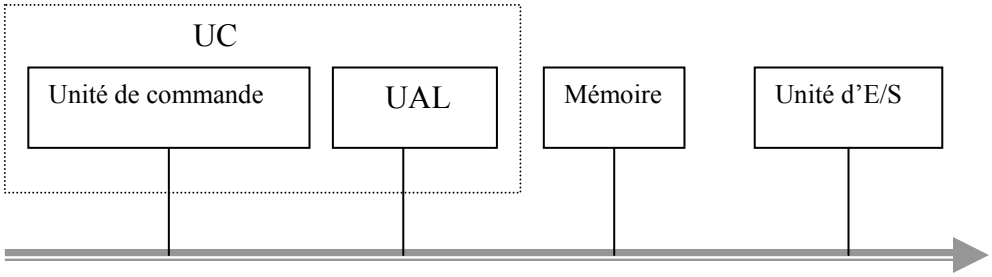


Figure I. 4 Architecture de Von Neumann

Un seul processeur (unité centrale) travaille de manière séquentielle sur des informations en mémoire qui constituent à la fois les données et les programmes : une seule unité de commande traitant une seule séquence d'instruction (*Single Instruction Stream*) et une seule unité d'exécution (UAL) traitant une unique séquence de données (*Single Data Stream*). Cette architecture est donc appelée SISD (*Single Instruction Single Data*). Un bus unique relie les différents modules. Cette architecture à commande unique constitue un frein au traitement parallèle.

I.7.1.2 Architecture de Harvard (1940)

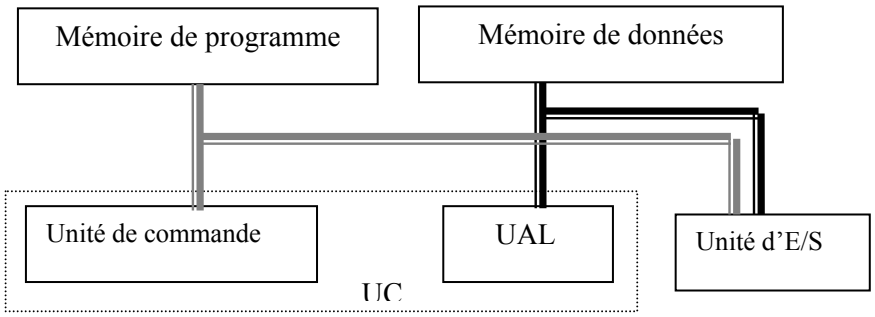


Figure I. 5 Architecture de Harvard

Comme illustre la figure I.5. Cette architecture est caractérisée par la séparation des données et programmes en mémoire. L'accès à chacun s'effectue par un bus séparé, différent et indépendant. Donc un accès simultané aux instructions et aux données est possible, ce qui implique une exécution plus rapide. Cette architecture est abandonnée sur les ordinateurs universels en raison de sa complexité matérielle, mais elle est avantageuse pour les systèmes de traitement numérique des signaux. Le 4004 (premier processeur Intel) relève de cette architecture, de même que la plupart des processeurs DSP. [2]

I.7.2 Architectures parallèles

I.7.2.1 Architectures avec cadencement des données

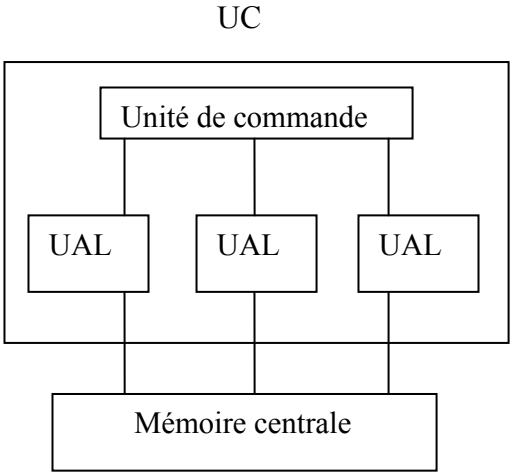


Figure I. 6 Architecture avec cadencement de données

Cette architecture à flot de données s'oppose à une architecture traditionnelle dite à flot d'instructions. Cette configuration est appelée SIMD (*Single Instruction Multiple Data*) et ce type de processeur « processeur vectoriel ». On peut classer dans cette catégorie les « processeurs vectoriels » disposant des instructions vectorielles (exemple d'application :

addition de deux vecteurs en virgule flottante). L'unité de commande envoie une instruction à toutes les UAL qui exécutent l'instruction pas à pas sur des données locales. Le réseau d'interconnexions permet aux résultats d'être envoyés vers une autre UAL qui pourra les utiliser comme opérande dans une instruction suivante. Les processeurs DSP utilisent cette architecture.

I.7.2.2 Architecture multiprocesseurs (MIMD)

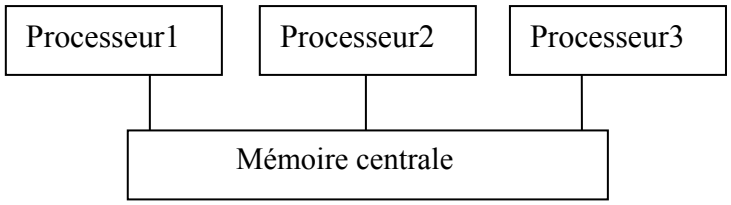


Figure I.7 Architecture MIMD

Plusieurs processeurs partagent la même mémoire. Chaque unité centrale (processeur) dispose de son propre programme indépendant.

I.7.2.3 Processeur multi-unités de traitement

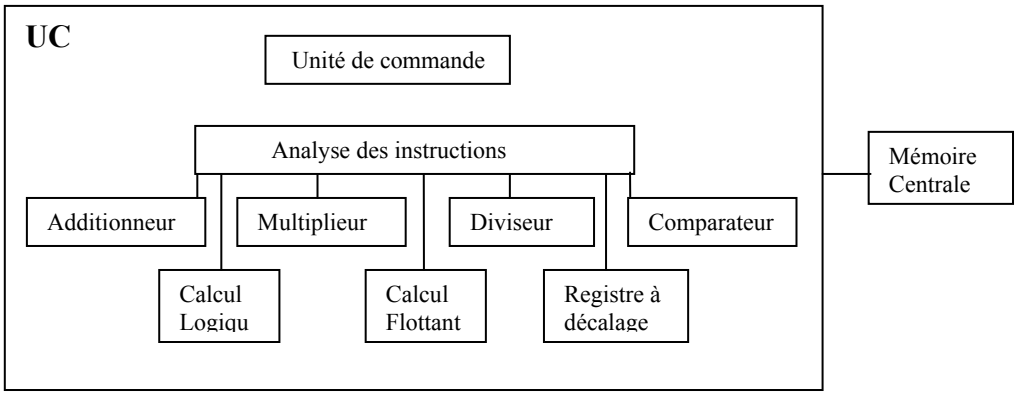


Figure I. 8 Processeur multi-unité.

Cette architecture n'a de sens que si le temps de chargement et de décodage est faible par rapport à la durée de son exécution.

I.8 LE FONCTIONNEMENT EN PIPELINE ET SUPERSCALAIRES

On tente, par tous les moyens, d'accélérer le fonctionnement des outils informatiques. Le plus évident consisterait à accroître la fréquence d'horloge, mais la technique en limite le plafond.
Aussi a-t-il fallu concevoir de nombreuses astuces pour améliorer la vitesse apparente de travail. Deux d'entres elles on reçu le nom de fonctionnement en *pipeline* et fonctionnement *superscalaire.*

I.8.1 Le fonctionnement en pipeline

L'idée est inspirée de l'organisation du travail à la chaîne : cette technique permet d'effectuer davantage de travail par unité de temps. Chaque étape de l'exécution de l'instruction (chargement, décodage, recherche des données) se fait dans une unité séparée

Par exemple, quatre unités chaînées peuvent intervenir :

1- une unité de recherche de l'instruction. (R)
2- Une unité de décodage. (D)
3- Une unité d'exécution. (Ex)
4- Une unité de résultat chargée de l'enregistrement. (Ec)

De ce fait, quatre instructions différentes peuvent se trouver simultanément sur la chaîne à des degrés divers de finition (figure I.9.). En une première approche et au maximum, la vitesse globale de travail est multipliée par quatre.

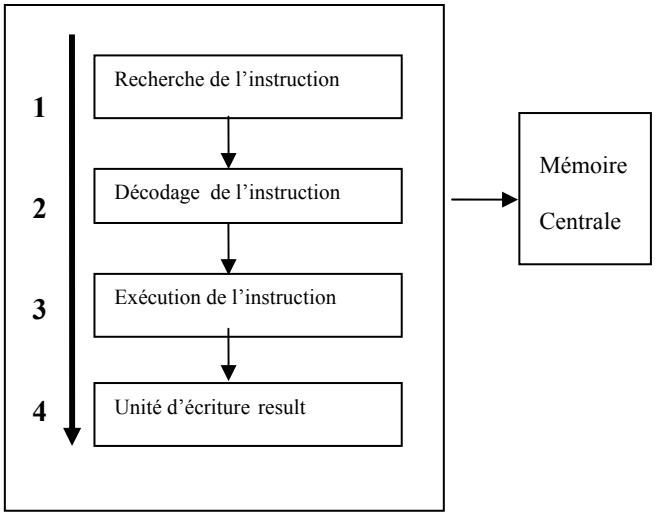


Figure I. 9 Processeur pipeliné

Dans un processeur sans pipeline, les instructions sont exécutées les unes après les autres. C'est à dire qu'il ne rentrera une nouvelle instruction dans le pipeline que lorsque l'instruction précédente est passée par tous les étages et est terminée.

Examinons le mode pipeline (figure I.10.). On fractionne l'exécution d'une instruction en opérations élémentaires effectuées en chaîne. Des unités successives sont chargées de ces tâches. Lorsqu'une unité a terminé son intervention et passe l'instruction au poste suivant, elle peut immédiatement traiter l'instruction suivante, pour ce qui concerne. De ce fait, quatre instructions différentes peuvent se trouver simultanément sur la chaîne à des degrés divers de définition.

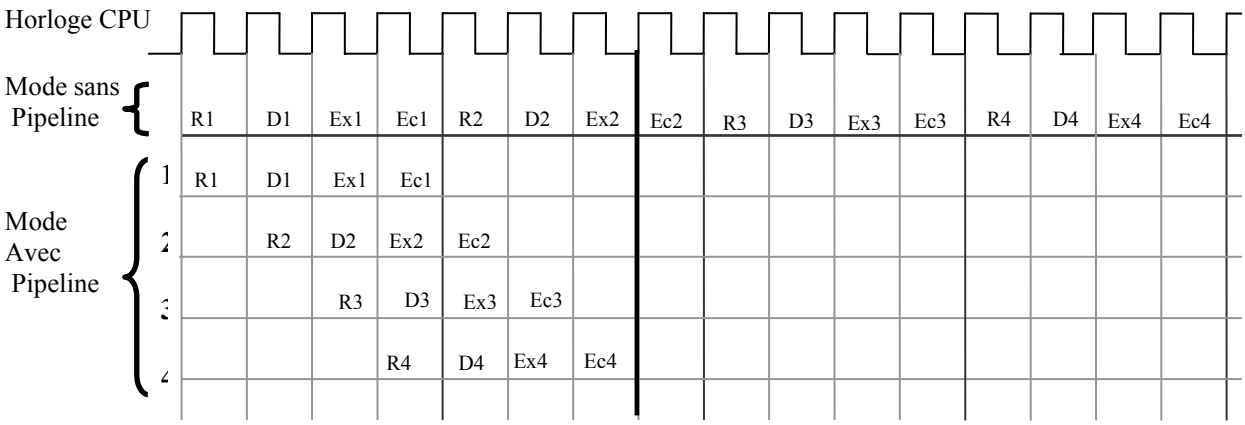


Figure I. 10 Principe du pipeline. Chaque unité exécute ce qu'elle est chargée de faire sur une instruction puis passe le résultat à l'unité suivante.

I.8.2 Fonctionnement superscalaire

Un programme est composé d'une succession unique (un seul « flot ») d'instructions. Elles sont enregistrées dans la mémoire centrale à des adresses croissantes. Puisque le microprocesseur peut recevoir plusieurs instructions dans sa file d'attente, pourquoi ne pas les distribuer immédiatement à des unités d'exécution en parallèle ?c'est ce que fait le mode superscalaire. Le but recherché est d'exécuter plusieurs instructions en parallèle et par conséquence l'augmentation de la vitesse d'exécution du programme. On est ainsi passé à une architecture incluant plusieurs unités d'exécutions spécialisées, par exemple (figure I.11):

- Une unité de calcul sur des nombres entiers.
- Une unité de calcul en virgule flottante.
- Une unité de chargement et de rangement.
- Une unité de gestion système pour les branchements.

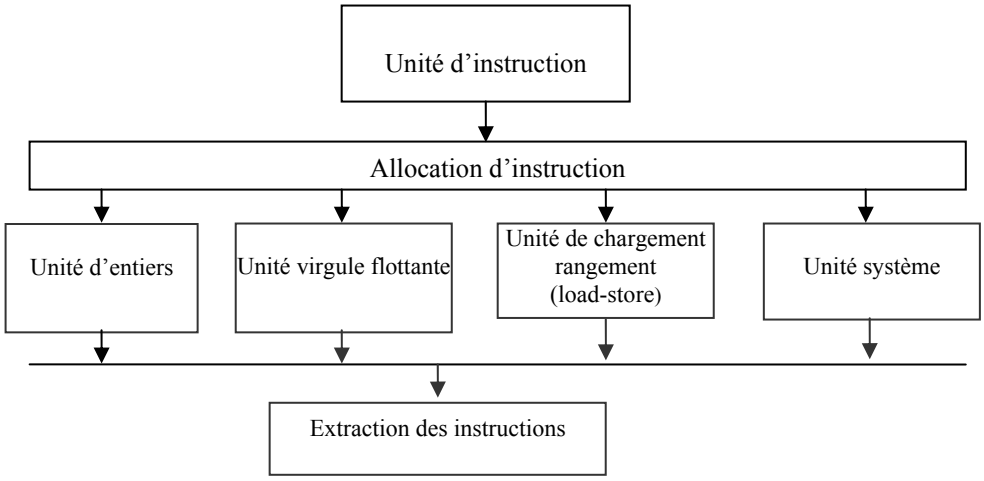


Figure I. 11 Fonctionnement superscalaire avec quatre unités d'exécutions fonctionnant en parallèle.

La figure I.12. Présente le chronogramme d'exécution d'un programme fait appel à quatre unités, de plus chacune d'elle est pilpelinée.

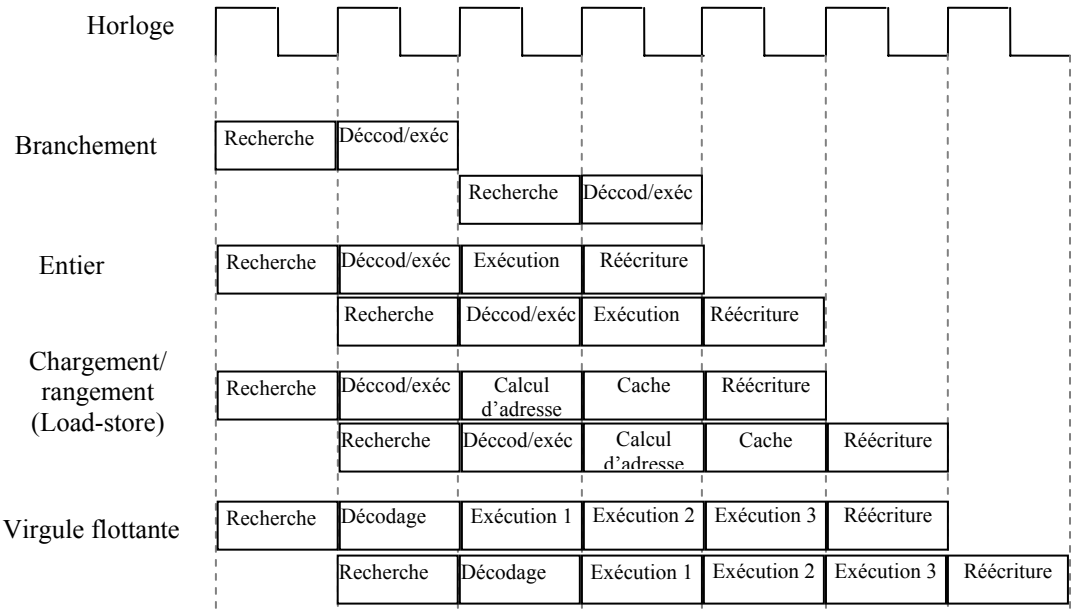


Figure I. 12 Combinaison entre le pipeline et le superscalaire

I.9 CONCEPTION D'UN TRES SIMPLE PROCESSEUR (MU0)

La figure I.13. Présente le schéma synoptique d'un système à microprocesseur (CPU) basé sur le modèle de Von Neumann .Le CPU est le maître unique dans les ordinateurs simples, et c'est lui qui contrôle tout le système .La séquence des instructions (ou programme) et les données sont stockées en mémoire à des emplacements successifs et en code binaire. Les trois parties sont connectées par les trois bus (adresse, données, contrôle) créés par le CPU

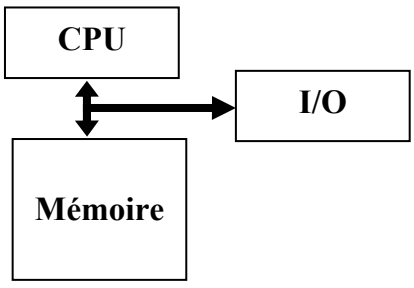


Figure I. 13 Le schéma synoptique d'un système à microprocesseur

Le rôle du CPU se décompose en quatre étapes :

1. aller chercher l'instruction suivante en mémoire cycle de chargement ;
2. la décoder pour déterminer l'action qu'il doit effectuer ;
3. l'exécuter cycle d'exécution ;
4. revenir à l'étape 1.

Le CPU est composé de :

- l'unité arithmétique et logique **UAL**
- l'unité de contrôle.
- Des registres et mémoire rapide localisés dans le CPU.

Laissez nous maintenant concevoir un très simple microprocesseur MU0 avec un longueur d'instruction de 16 bits et ainsi le bus de données, et avec minimum des composants, comme il est présenté dans la figure I.14.

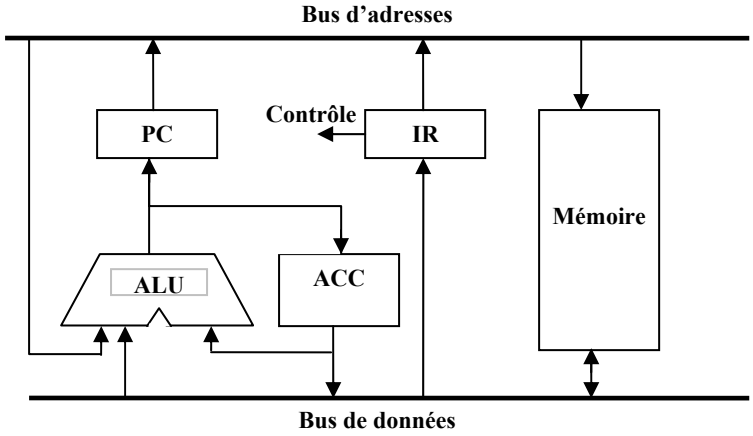


Figure I. 14 Schéma synoptique d'un simple microprocesseur MU0

- Le compteur du programme **PC** (*program counter*) ou bien le compteur ordinal est un registre qui contient l'adresse de la prochaine instruction à exécuter.

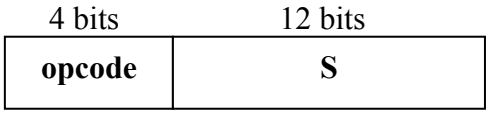
- L'accumulateur **ACC** (*accumulator*) est un registre qui contient la donnée qui est entrain de traiter.

- Le registre d'instruction **IR** (*instruction register*) contient le code de l'instruction qui vient d'être exécuté.

- L'unité arithmétique et logique **ALU** (*arithmetic and logic unit*) est chargée d'exécuter les opérations arithmétiques et logiques sur les données.

Nous allons utiliser seulement 8 instructions, donc nous allons besoin de 4 bits pour chaque instruction ce qu'on appel le code d'opération (*opcode* :*operation code*) ,les autres12 bits sont utilisés pour définir les adresse mémoire pour les données (opérandes) .

Notre code d'instruction de 16 bits (*le code machine*) a le format suivant :



Cette machine peut adresser 2^{12} = 4k mots (words) = 8k octets (bytes) de données.

I.9.1 Le jeu d'instruction du MU0

Nous avons choisi le format des instructions (4 bits pour le opcode ,12 bits pour l'adresse de l'opérande). Et l'espace mémoire à utiliser. La longueur du mot traité est 16 bits cela définit la capacité des registres internes, les bus internes et externes et la catégorie du microprocesseur. Le tableau I.2. Présente le jeu d'instruction de notre microprocesseur :

Tableau I.2. Le jeu d'instruction de MUO

instruction	Opcode	effet
LDA S	**0000**	**mem[S] → ACC**
STO S	**0001**	**ACC → mem[S]**
ADD S	**0010**	**ACC + mem[S] → ACC**
SUB S	**0011**	**ACC - mem[S] → ACC**
JMP S	**0100**	**PC : = S**
JGE S	**0101**	**SI ACC > 0, PC : = S**
JNE S	**0110**	**SI ACC = 0, PC : = S**
STP	**0111**	**STOP**

I.9.2. Exemple d'exécution d'un programme simple

Après l'évolution architecturale que nous venons de présenter, nous allons comprendre, maintenant, comment un programme simple peut être exécuté par notre microprocesseur. Ce qui démystifier son fonctionnement. Notre exemple consiste à additionner deux nombres, qui sont localisé dans les cases mémoire 2E et 2F successivement et nous allons stocker le résultat dans l'adresse 30. Nous sommes besoin de charger l'accumulateur ACC avec la première valeur, additionner l'autre valeur, et stocker le résultat dans la mémoire.

Mnémonique		**Code machine**	**explication**
LDA 02E		**002E**	mettre [2E] dans ACC
ADD 02F	⟶	**202F**	Ajouter [2F] à [ACC]
STO 030		**1030**	mettre [ACC] dans l'adresse 30
STP		**7**	fin du programme

Ces instructions doivent être rangées en mémoire. Chaque cellule mémoire contient deux octets. Sachant qu'on commence par la première ligne, on va lire aux emplacements successifs ce que montre la figure I.15. Il nous reste maintenant à dérouler le film de ce qui va passer dans le microprocesseur. Après cette démonstration, nous aurons assimilé le principe de fonctionnement très général de touts les microprocesseurs, ainsi celle de processeur ARM (notre sujet).

Mémoire

002E
202F
1030
7

Figure I. 15 Présentation du programme dans la mémoire

Etape 1 : au départ

Ce qui intéresse ici, le contenu du compteur ordinal PC, du registre d'instructions et de l'accumulateur par rapport au contenu de la mémoire. Tout juste au lancement de l'exécution de ce programme, la situation est telle que le montre la figure I.16. Le PC contient l'adresse de la première instruction, donc000, alors que l'accumulateur contient un point d'interrogation. Il en va de même du registre d'instructions.

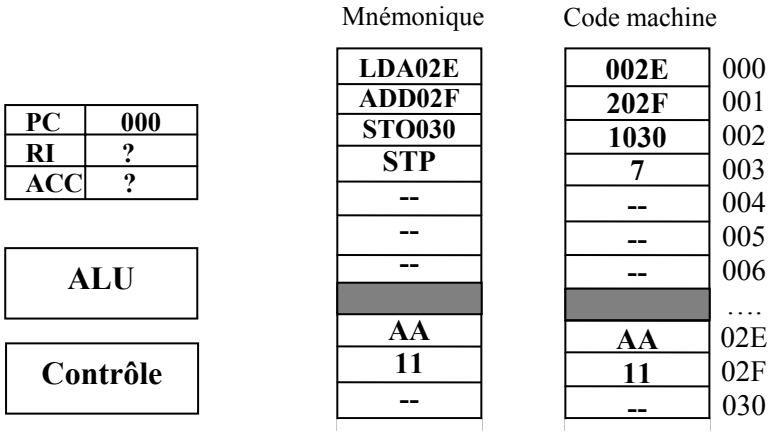


Figure I. 16 La vue de première étape

Etape 2 :

La séquence dont nous déroulons le programme est commandée par les impulsions d'horloge successives. Les circuits de commande et de séquencement commencent par commander la lecture de la mémoire à l'adresse contenue dans le PC. Donc 000.le mot de la cellule 000 est alors lu en mémoire et ramené dans le registre d'instruction, c'est la recherche de l'instruction (*fetch instruction*). Il est simplement dupliqué et n'est pas effacé de la mémoire. Aussitôt, et automatiquement, le compteur ordinal PC s'incrémente et pointe la cellule suivante, donc 001.cela se passe dans le premier cycle. (Figure I.17.)

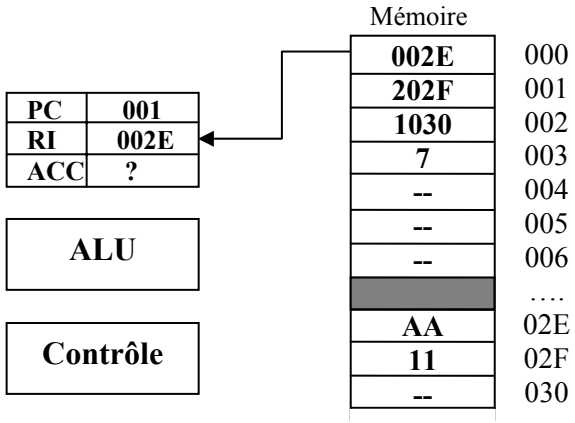


Figure I.17 Le premier cycle de deuxième étape du programme

Dans le deuxième cycle le décodeur génère les signaux nécessaires pour le chargement de l'accumulateur par la donnée stockée dans l'adresse 2E, donc AA, c'est l'exécution de l'instruction décodée. Ce que montre la figure I.18.

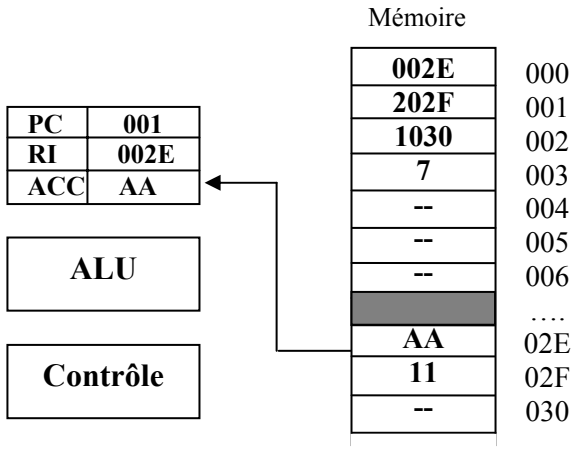


Figure I.18 Deuxième cycle de la deuxième étape.

Etape 3 : on passe à la seconde instruction

Comme pour la première instruction, dans le premier cycle le microprocesseur commence par lire le code de l'opération à exécuter. Il le loge dans le registre d'instructions. Le PC s'incrémente de nouveau. Dans le deuxième cycle l'instruction va décoder. Avec les signaux générés par le décodeur l'ALU va additionner le contenu de l'adresse 2F avec celle de l'ACC et l résultat sera automatiquement dans l'ACC .comme montre la figure I.19.

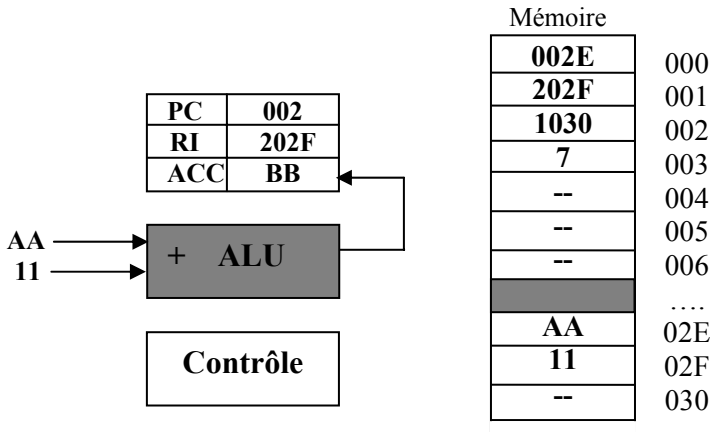


Figure I.19 L'exécution de l'addition

Etape 3 : on passe à la troisième instruction

La même procédure sera réalisé, recherche de l'instruction, l'incrémentation du PC dans le premier cycle. Le décodage et l'exécution de l'opération pour stocker le résultat dans l'adresse 030. (Figure I.20.).

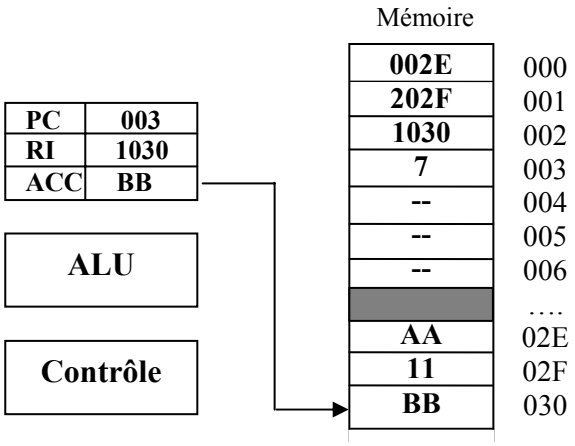


Figure I. 20 Stockage de résultat dans la mémoire

Etape 4 : on passe à la quatrième instruction

Cette instruction va prendre un seul cycle pour l'exécution. C'est la fin de notre programme.

I.9. 3 Remarques

En peut conclure de cet exemple, que le microprocesseur réalise des opérations, suivant un code d'instructions, stockées à la mémoire. Ces instructions contiennent deux parties le opcode et l'opérande. Le nombre de cycle d'horloge, pris pour une instruction du MU0, est le même que le nombre d'accès à la mémoire se fait.

- LDA STO ADD SUB chacune prend 2 cycle d'horloge, un pour rechercher et décoder l'instruction, le second pour rechercher et traiter la donnée.
- JMP, JGE, JNE, STP chacune utilise un seul accès à la mémoire (lecture), par conséquent l'exécution sera réalisée dans un seul cycle d'horloge.

On constate aussi que dans le MU0, la mémoire contient le programme, et les données à la fois, et ces espaces sont séparés.
Notre microprocesseur possède un seul registre interne c'est l' ACC, se qui impose l'accès courant à la mémoire, pour éviter ça il nous faut utiliser des autres registres internes, pour stocker les résultats et les adresses mémoire. Par conséquent le microprocesseur sera plus rapide.

I.10 CIRCUITS ELECTRIQUE INTERNES DES MICROPROCESSEURS

Dans cette section on va expliquer comment les informations circulent à l'intérieur du microprocesseur. Pour cela, on va présenter des structures de multiplexeurs et de démultiplexeur, de codeurs, de points mémoires et de registres, etc.

I.10.1 Multiplexeurs

Le multiplexage consiste à faire passer sur une seule voie des données pouvant provenir de plusieurs voies d'entrées. Il convient d'aiguiller une entrée, et une seule, sur l'unique voie de transfert. Par exemple examinons la figure I.21. Qui présente quatre entrées à multiplexer sur une sortie. Le mot de commande, d'aiguillage, est lui-même sur deux bits afin d'obtenir ici quatre combinaisons à l'aide des inverseurs.

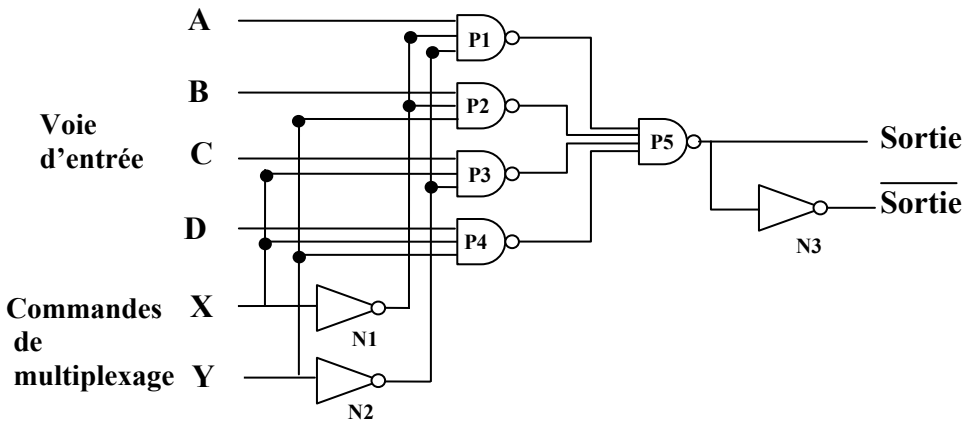


Figure I. 21 Structure d'un multiplexeur 4 voies d'entée, avec une double sortie (directe et inversée).il fait essentiellement appel à des portes NAND notées P et des inverseurs notées N.

Le tableau d'états des lignes de commande X et Y du multiplexeur quatre voie est donnée dans le tableau I.3. Cela demande quelques instants de réflexion.

Tableau I.3 Effet des commandes du multiplexeur quatre voies

X	Y	Voie autorisée
0	0	A
0	1	B
1	0	C
1	1	D

I.10.2 Démultiplexeurs

Le démultiplexage est l'opération inverse. Les informations entrant sur une voie unique sont distribuées sur plusieurs voies de sortie. La figure I.22. Donne l'exemple d'un démultiplexeur 4 voies. Le mot de commande est, de nouveau, sur deux bits et selon son état, c'est l'une ou l'autre des voies qui est activée.

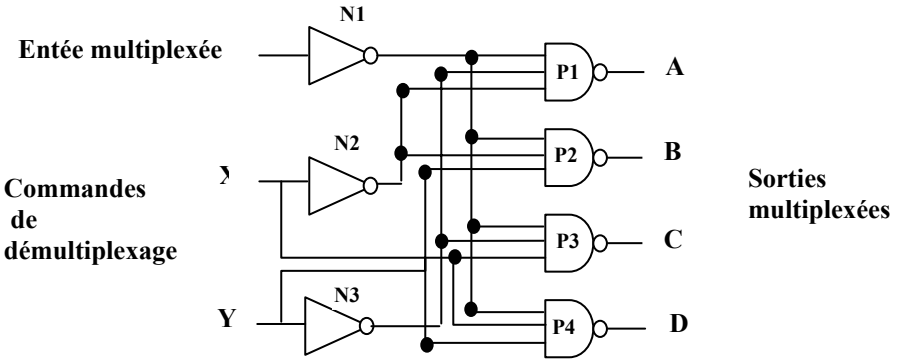


Figure I. 22 Démultiplexeur 4 voies. L'entée se fait sur une voie unique. Les quatre voies de sortie sont A, B, C et D.

I.10.3 Codeurs et transcodeurs

Voici une autre famille de circuits, ceux destinés au codage ou, au-delà, au transcodage. La fonction inverse est le décodage. Le principe en est très simple, la figure I.23. L'illustrant avec le simple décodage de deux bits, X et Y, fournissant 4 combinaisons sur les sorties A, B, C, et D.

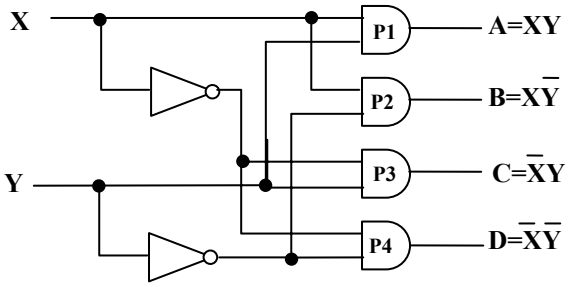


Figure I. 23 Exemple de décodage de deux bits fournissant quatre combinaisons.

En pratique, les applications du décodage peuvent aller très loin. Le nombre de codes étant relativement élevé. Il existe plusieurs types du codeur et décodeurs selon les applications désignée. Par exemple codeur BCD (transposer du décimal en DCB soit décimal codé binaire) comme présente le tableau 1.4. Est un décodeur BCD pour la réalisation de la procédure inverse du codeur BCD.

Tableau 1.4 Codage décimal à DCB compacté

Décimal	Décimal codé binaire (DCB)			
	B3	B2	B1	B0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

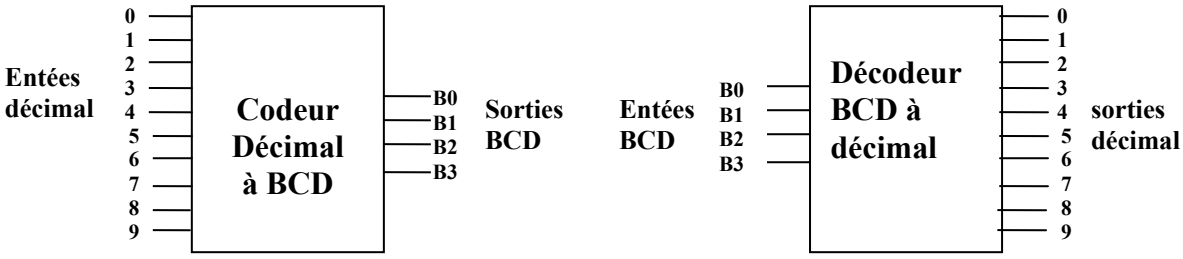


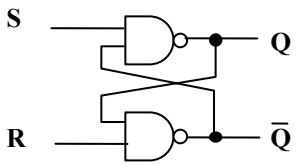
Figure I. 24 Schéma synoptique d'un codeur BCD et décodeur BCD

I.10.4 Bascules

Une bascule est un élément bistable, disposant de deux états d'équilibre. Elle est comparable, en cela, à un interrupteur courant qui possède, lui aussi, deux états stables qu'il maintient tant qu'on n'en modifie pas volontairement un.
En anglais, une bascule est appelée, un **flip-flop**. Par convention, les sorties existent en direct et en inversé et sont appelées Q et $\overline{Q}$.

I.10. 4.1 Bascule RS

La bascule RS peut être réalisée avec des portes OU ou ET, ainsi qu'en témoigne la figure I.25. Les deux entée sont appelées R et S. la table d'états explique le comportement de ce circuit.



Entées		Sorties	
R	S	Q	$\overline{Q}$
0	0	Q_{n-1}	$\overline{Q_{n-1}}$
0	1	1	0
1	0	0	1
1	1	?	?

Figure I. 25 Bascule RS réalisée à l'aide de porte NAND

1. **R=S=0 :** les sorties conservent leur état précédent, noté $\mathbf{Q_{n-1}}$.
2. **R=0 et S=1 :** la sortie Q passe à 1 .on dit qu'elle est mise à un (**set** : S=1).
3. **R=1 et S=0 :** la sortie Q passe à 0. on dit qu'elle est remise à zéro (**Reset** : R =1).
4. **R=S=1 :** on crée une situation de sortie indéterminée, non, prévisible, ce que nous avons marqué par un point d'interrogation.

I.10.4.2 Bascule JK

La bascule RS possède un état indéterminé et donc interdit. Pour tourner cette difficulté, la bascule JK a été inventée. La table d'états est donnée dans le tableau I.5.

Tableau I.5 Etats de la bascule

Entées		Sorties
J	K	Qn
0	0	Q_{n-1} (état précédent)
0	1	0
1	0	1
1	1	$\overline{Qn}$ (inversion de l'état courant)

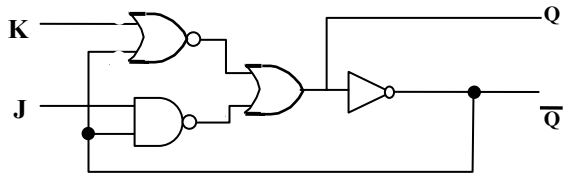
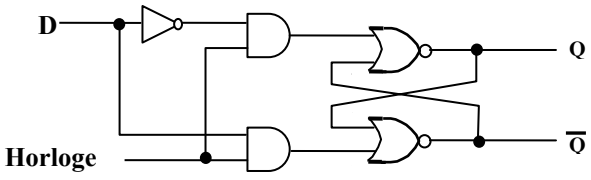


Figure I. 26 Montage simplifié de principe d'une bascule JK

I.10.4. 3 Bascule D

Avec les bascules RS et JK, deux entrées sont disponibles. La bascule D, elle, ne possède qu'une seule entée. Elle est attaquée par un signal via un inverseur (figure I.27.). On s'arrange pour que la sortie « recopie » l'entée, et ce, après un petit délai, d'ou la lettre D qui marque cette organisation. De ce fait :

$$\mathbf{Q = D} \quad \text{et} \quad \mathbf{\overline{Q} = \overline{D}}$$



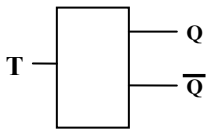
Temps n	Temps n+ 1
D	Q
1	0
0	1

Figure I. 27 Bascule D.

I.10. 4. 4 Bascule T

La bascule de type T change d'état, passe dans la situation opposée, à chaque impulsion d'entrée. Son T provient de sa dénomination anglaise, « Toggle », basculer. Cette bascule est à entée unique :

- Un 0 à l'entrée verrouille la bascule dans son état précédent.
- Un 1 l'entrée provoque l'inversion des sorties.



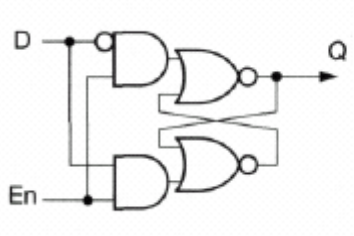
Etats de la bascule T

T	Qn	Qn+1
0	0	0
0	1	1
1	0	1
1	1	0

Figure 1. 28 Bascule type T

Remarque

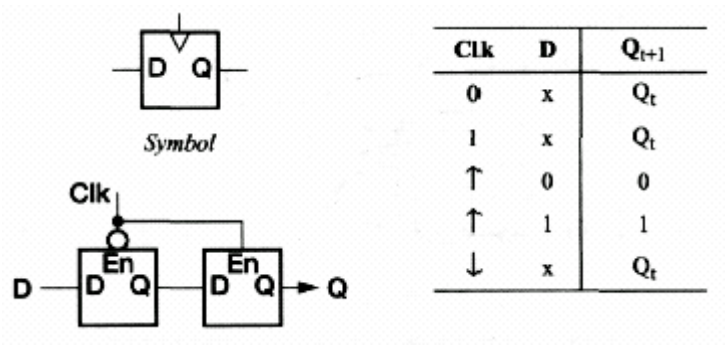
Une latche transparente (ou le D-type) est réalisé avec une bascule *D* a une entrée de données (*D*) et un signal de validation (*En*); la sortie suit l'entrée *D* lorsque *En* est en haut, mais elle conserve sa état précédente si *En* au niveau bas à n'importe quelle valeur appliquée sur D. Cela peut être réalisé avec bascule R-S en générant R et S à partir de *D* et *En* comme indiqué dans la figure I.29.



En	D	Q_{t+1}
0	0	Q_t
0	1	Q_t
1	0	0
1	1	1

Figure I. 29 Une latche transparente D-type

En principe il doit être possible de construire n'importe quel circuit séquentiel employant une latche de D-type qui assure la logique combinatoire entre les étages. L'application d'une impulsion très courte positive à *En* laissera les données d'état suivantes a travers la latche et le conservera ensuite avant que la logique combinatoire n'ait le temps pour répondre aux nouvelles valeurs. Cependant, en pratique cela se tourne d'être une façon très dure de construire un circuit fiable.
Il y a des façons diverses de construire des circuits de *latching* plus fiables, dont la plupart exigent que chaque signal passe par deux latches transparentes par le cycle d'horloge. Dans le plus simple de ceux-ci, la deuxième latche est placée en série avec le premier et fait fonctionner avec la fonction inverse de validation. À tout moment, une latche ou l'autre se conserve et l'autre est transparent. Sur un bord d'horloge, quand la première latche va opaque et le deuxième va transparente, la valeur de données d'entrée se propage vers la sortie. Il est alors tenu par le cycle complet jusqu'au même bord dans le cycle suivant. Donc c'est une latche à barrière déclenchée (*edge-triggered latch)*, on montre son symbole logique, de circuit et la table de vérité dans la Figure I.30. (Dans la table un 'x' dans une colonne d'entrée indique que l'entrée n'a aucun effet sur la sortie.)



Clk	D	Q_{t+1}
0	x	Q_t
1	x	Q_t
↑	0	0
↑	1	1
↓	x	Q_t

Figure I. 30 Une latche à barrière déclenchée (*edge-triggered latch)*

I.10. 5 Les mémoires

I.10. 5.1 Organisation des mémoires

Une mémoire peut être représentée, comme offrant une distribution en surface des cellules mémoire, élémentaire qui ne peut stocker qu'un seul bit, Comme une grille de mots croisés. On parle alors de colonnes et de rangées, Pour adresser une mémoire donnée. Le principe de l'adressage des cellules élémentaires est indiqué dans la figure I.31.

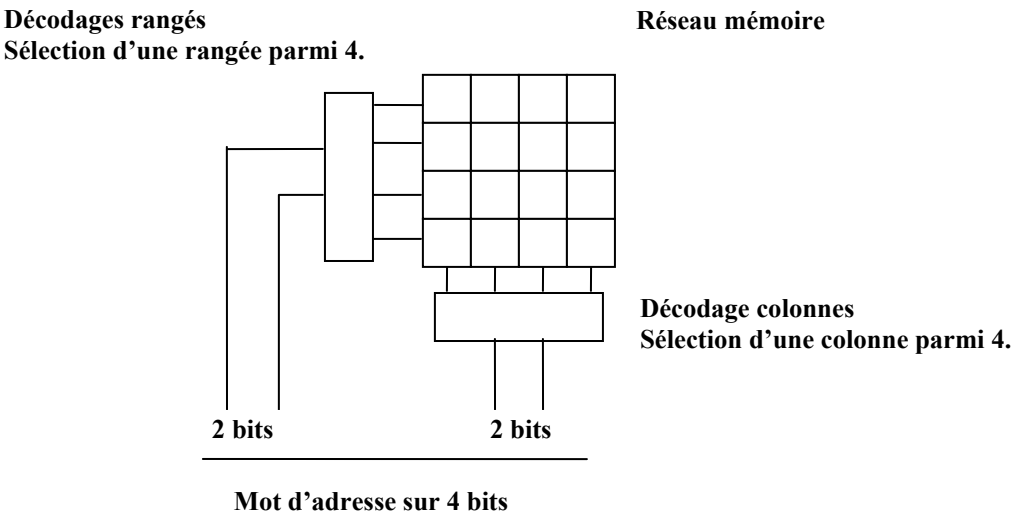


Figure I. 31 Un mot d'adresse est décodé pour viser une seule cellule à partir de sa rangée et de sa colonne.

On réduit le réseau à 16 cellules mémoire, ce qui fait qu'un mot d'adressage sur bits suffit pour designer une cellule élémentaire. Si l'on détaille ce montage, on aboutit à la figure I.32.

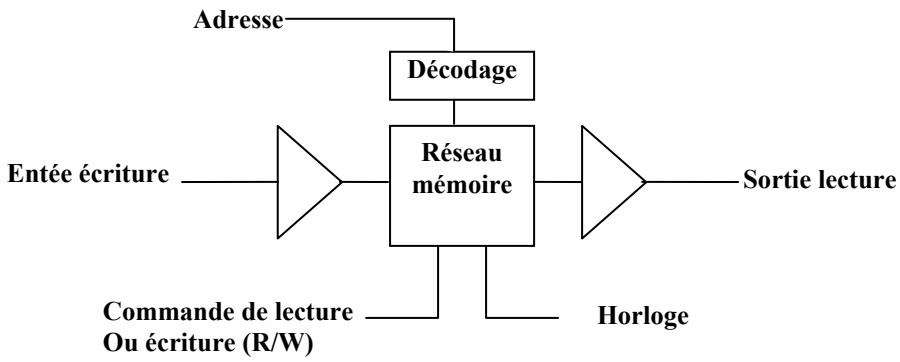


Figure I. 32 Un circuit comprend, outre le réseau des points de la mémoire, un décodeur d'adresse, une logique de commande d'écriture ou de lecture, des amplificateurs tampons d'écriture ou de lecture et une horloge cadencant le *fonctionnement.*

I.10. 5.2 Mémoires mortes ROM

Le nom générique de ce type de mémoire est ROM pour « *Read Only Memory* », soit mémoire à lecture seulement.

I.10. 5. 2.a) Les ROM de base

La ROM de base est un circuit programmé au cours de sa fabrication, par masquage, chaque point mémoire est un fusible, transistor bipolaire… etc.

I.10. 5. 2. b) ROM programmable, ou PROM

Est une ROM qui peut être programmée par l'utilisateur, soi-même, chaque point est équipé d'un transistor généralement bipolaire ou bien plus rarement MOS, en série avec ce transistor se trouve un fusible.(figure I.33.)

1. si le fusible est intact, le transistor conduit et l'on suppose qu'on a enregistré un 1.
2. si l'on fait claquer ce fusible, le courant ne passe plus, et l'on suppose qu'on a enregistré un zéro.

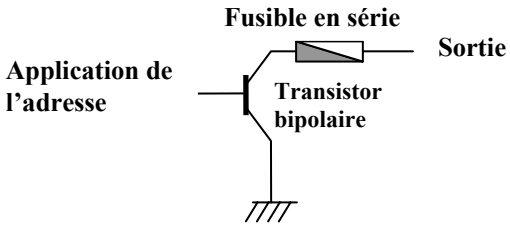


Figure I. 33 Principe du point mémoire d'une ROM

I.10. 5. 2.c) La PROM reprogrammable EPROM

Si l'on veut pouvoir modifier la programmation, il faut passer à une autre technologie, celles des EPROM effaçable électriquement. La technique consiste, avec un MOS, à isoler son électrode de commande «gate », des lors (figure I.34.).

1. on peut charger le gate en lui injectant des charges électriques, à travers l'isolant par effet tunnel.
2. pour supprimer ces charges, les effacer, on bombarde le dispositif avec l'ultraviolet court pendant quelques minutes

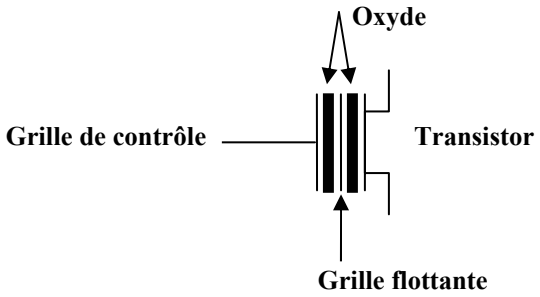


Figure I. 34 Cellule mémoire d'une EPROM

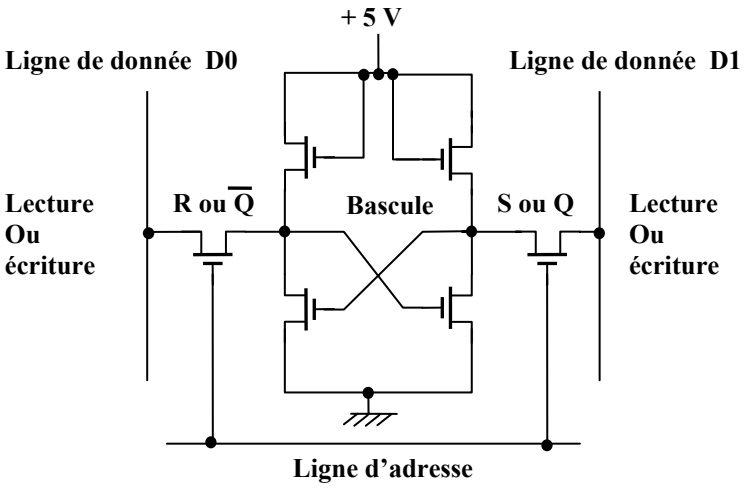
Un autre type des EPROM ont été mis au point c'est EEPROM pour « *Electrically Erasable PROM* », soit une PROM effaçable électriquement, cela va considérablement minimisé le temps d'effacement. En fin, les mémoires « Flache » peuvent aussi être considérée come variante des ROM mais plus récente. Leurs principaux avantages, c'est que ces mémoires flache sont programmables électriquement et qu'elles ne sont pas volatiles, qu'elles conservent des temps d'accès réduits et que leur capacité est devenue importante.

I.10. 5.3 Mémoires vives

Les mémoires vives sont les RAM, ainsi appelées par analogie avec le sigle ROM. Son interprétation est « Random Access Memory », soit mémoire à accès aléatoire. Cela signifie qu'on peut adresser une cellule mémoire directement, via son adresse, mais c'est également le cas des ROM. Les RAM existent en deux grandes familles de base :

I.10. 5. 3.a) RAM statiques SRAM

Chaque point mémoire d'un RAM statique est une simple bascule en transistor MOS (figure I.35.) le fonctionnement est celui des bascules : l'application d'un niveau à une entrée verrouille la bascule dans un état donné ; elle y reste tant qu'on n'applique pas un ordre inverse. On peut vérifier cet état « lire la mémoire » sans le perturber. On notera simplement qu'un point mémoire mobilise une bascule complète, plus d'ailleurs des transistors d'adressage. Elle occupe donc un espace de silicium non négligeable, bien davantage que les RAM dynamiques. Elle consomme aussi davantage. Elle est plus rapide qu'une RAM dynamique. La RAM statique peut travailler sur la fréquence zéro.



Figue I. 35 Structure de principe d'une cellule de mémoire RAM statique.

I.10.5.3.b) RAM dynamiques DRAM

La RAM dynamique n'a besoin que d'un seul transistor par cellule. Ce MOS est chargé par une capacité de structure qui, de toutes façons, existe par construction. On ne peut jamais éviter les condensateurs parasites. Elle est simplement portée à une valeur compatible avec le stockage d'une information binaire (figure I.36.).

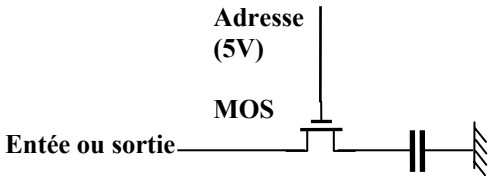


Figure I. 36 Principe d'une cellule d'une mémoire RAM statique. C'est la capacité qui fait office de mémoire, le transistor ne servant qu'à l'adressage.

Le fonctionnement de cette cellule est le suivant :

- l'information binaire est réellement mémorisée par le condensateur.
- Le transistor commande l'adressage. Lorsque le 5 V est appliqué à son électrode de commande, on a accès au condensateur.
- Pour écrire un niveau 1, on adresse la cellule et on applique 5 V à l'entrée. Le condensateur se charge.
- Pour écrire un zéro, on adresse la cellule et on applique 0 V à l'entée. Le condensateur se décharge.
- Pour lire, on adresse la cellule et on mesure à la sortie la tension présente aux bornes du condensateur.

Tout cela est extrêmement simple, malheureusement le condensateur manifeste une propension toute naturelle à se décharger dans le temps, car aucun isolant n'est parfait. Si l'on a stocké un zéro, ce n'est pas grave, mais si l'on a enregistré un 1 (du 5V), on risque de rien retrouver au bout d'un certain temps.

C'est pourquoi, périodiquement, il faut procéder à une double opération :

1. lire toutes les cellules à tour de rôle.
2. si l'on constate qu'une cellule enregistre un1 et que, par conséquent elle est en train de se décharger, il faut refaire le plein immédiatement. Il faut donc réécrire un 1 immédiatement.

Cette opération s'appelle le **rafraîchissement** de la mémoire. Elle s'effectue sous la commande de circuits spécialisés, parfois inclus dans le microprocesseur. Elle s'effectuait via une ligne d'adresse directe en mémoire (DMA) aux origines des PC.

I.10.6 Les registres

Un registre, c'est l'assemblage de plusieurs points mémoire. Il peut y en avoir 8, ou 16, ou 32, ou toute autre valeur. Par convention, un registre est dispositif à accès parallèle : tous les bits entrent et sortent en parallèle. Lorsque tel n'est pas le cas, on le précise en parlant de « registre à décalage », par exemple.

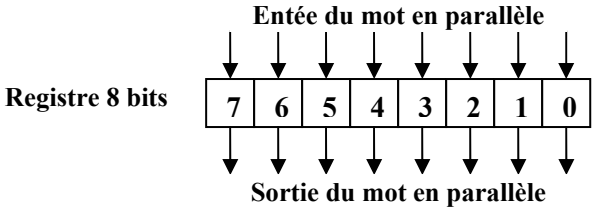


Figure I. 37 Un registre 8 bits.

I.10.6.1 Les registres à décalage

Les registres à décalage servent non seulement à décaler les données pour exécuter des tests ou des opérations arithmétiques, mais encore à exécuter des conversions parallèle série ou série parallèle. Par exemple (la communication entre PC modem puis ligne téléphonique). Selon le processus appliqué, on parle de décalage ou de rotation, ces termes n'étant pas parfaitement normalisés. La rotation c'est faire tourner un mot en rond dans le registre, le bit qui sort d'un côté rentrant dans le registre de l'autre côté. Rappelons, à ce propos, que décaler un mot binaire d'un seul bit revient soit à le multiplier par deux (décalage vers la gauche), soit à le diviser par deux (décalage vers la droite).

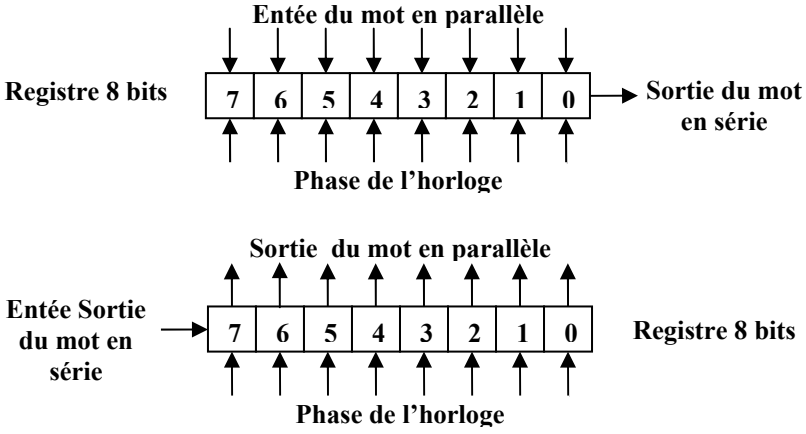


Figure I. 38 Fonctionnement des registres à décalage en deux modes de conversion

La figureI.39. illustre comment relier une ligne (un seul bit) du bus à un registre avec des port de transfert commandée par un ordre provenant des circuits de commande internes du microprocesseur, pour lecture (écriture) d'un bit du (dans) registre.

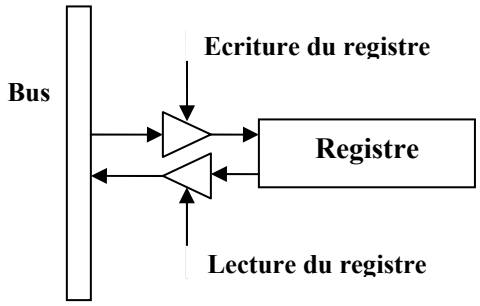


Figure I. 39 Montage d'aiguillage pour une lecture et une écriture

I.11 Conception pour une consommation électrique faible

Le point de départ pour la conception pour une faible puissance c'est savoir où la puissance va disparaître dans les circuits existants. La technologie CMOS est la technologie dominante pour l'électronique digitale moderne très performante et a quelques bonnes propriétés pour la conception pour une puissance réduite, donc nous commençons par regard comment la puissance sera consommer dans un circuit CMOS.

Un circuit CMOS typique est la porte statique NAND, illustrée dans la figure I. 40. Toute l'oscillation de signaux est entre les tensions d'alimentation et la masse, Vdd et Vss, Jusque récemment le 5 volt était la polarisation standard, mais beaucoup de processus de CMOS modernes requiert une tension d'alimentation inférieure d'autour de 3 volts et les dernières technologies fonctionnent avec des alimentation entre 1 et 2 volts et sera réduite plus loin dans l'avenir.

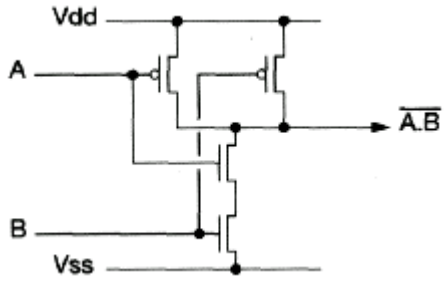


Figure I. 40 Une porte NAND CMOS statique à deux entrées

La porte fonctionne en connectant la sortie à Vdd à travers un transistor de p-type (*pull-up*), ou bien à Vss à travers un transistor de n-type (*pull-down*). Quand les entrées sont presque égaux à l'un de deux Vdd ou Vss, alors un de ces réseaux sera conduit et l'autre évidemment ne sera pas, ainsi il n'y a aucun chemin de Vdd à Vss à travers la porte. En outre, la sortie est normalement connectée aux entrées à travers des portes semblables Et perçoit donc seulement une charge capacitive. Une fois que la sortie est pilotée par Vdd ou Vss, pas de courant qui circule. En plus un temps très court sera pris après la commutation de la porte pour que le

circuit atteint sa condition stable et aucun nouveau courant ne soit pris de l'alimentation. Cette caractéristique de la consommation de puissance seulement à la commutation n'est pas partagée par beaucoup d'autres technologies de logique et a été un facteur principal dans la fabrication de la technologie CMOS et dans le choix des circuits intégrés de haute densité.

I.11.1 Les composants de l'énergie de circuit CMOS

La consommation électrique totale d'un circuit CMOS comprend trois composants :

1. **puissance de commutation**.

   C'est la puissance dissipée par la charge et la décharge de la capacité de sortie de la porte C_L et représente le travail utile exécuté par la porte. L'énergie par la transition de sortie est:

   $$E_t = \frac{1}{2} \times C_L \times Vdd^2 \approx 1 \text{ picojuole} \qquad \text{(I.1)}$$

2. **puissance de court-circuit.** Quand les entrées de la porte sont à un niveau intermédiaire tous les deux réseaux type p et n peut conduire. Cela va créer un chemin transitoire de conduction entre Vdd et Vss. Avec un circuit correctement conçu (que signifie généralement celui qui évite des transitions lentes de signal) la puissance de court-circuit doit être une petite fraction de puissance de commutation.
3. **Courant de fuite.**
   Les réseaux de transistor conduisent un très petit courant quand ils sont dans leur état blocages 'off' ; quoique sur un processus conventionnel ce courant soit très petit (une petite fraction d'un nanoamp par une porte), c'est la seule dissipation dans un circuit qui est actionné, mais inactif et peut drainer une batterie d'alimentation au cours d'une longue période de temps. C'est généralement négligeable dans un circuit actif. [1]
   Dans un circuit actif bien conçu le puissance de commutation domine, avec le puissance de court-circuit de10 % à 20 % au puissance total et le courant de fuite étant significatif seulement quand le circuit est inactif. [1] Cependant, la tendance de l'opération pour baisser la tension mène à un compromis entre la performance et le courant de fuite comme discuté plus loin ci-dessous et la fuite est un souci augmentant pour les conceptions très performantes à puissance réduite.

I.11.2 puissance des circuits CMOS

La puissance totale dissipée, Pc, de circuit CMOS, négligeant les composants de court-circuit et de fuite, est la somme des puissances dissipées de chaque porte *g* dans le circuit :

$$Pc = \frac{1}{2} \cdot f \cdot Vdd^2 \cdot \sum_{g \in C} A_g \cdot C_L^g \qquad \text{(I.2)}$$

Où f est la fréquence d'horloge, A_g est le facteur d'activité de porte (indiquant le fait que pas toutes les portes commutent chaque cycle d'horloge) et C_L^g est la capacité de charge de porte. Notons que avec la somation de ces lignes d'horloge, qui font actionner deux transitions par un cycle d'horloge, ont un facteur d'activité de 2.

I.11.3 Conception de circuit à puissance réduite

La capacité de charge d'une porte typique est en fonction de la technologie de processus, et n'est pas sous le contrôle direct de désigner. Les autres paramètres de l'équation [I.2] déterminent les différentes approches pour la conception pour une puissance réduite. Ceux-ci sont cités ci-dessous :

- Réduire au minimum la tension d'alimentation, Vdd. La contribution de la tension de d'alimentation à la dissipation de puissance fait ça une cible évidente. C'est discuté plus loin ci-dessous.
- Réduire au minimum l'activité de circuit. Les techniques comme l'horloge *clock-gateing* tombent sous ce titre. Chaque fois qu'une fonction de circuit n'est pas nécessaire, l'activité doit être éliminée.
- Réduire au minimum le nombre de portes. Des circuits simples emploient moins de puissance que circuits complexes.
- Réduire au minimum la fréquence d'horloge f . Eviter un hauts taux d'horloge est clairement désirable, mais bien qu'un taux bas d'horloge réduise la consommation électrique il réduit aussi la performance, ayant un effet neutre sur l'efficacité de d'énergie (mesuré, par exemple, dans MIPS - des Millions d'Instructions Par Seconde - par un watt). Si, cependant, une fréquence d'horloge réduite permet le fonctionnement à Vdd réduit, ce sera fortement avantageux à l'efficacité de l'énergie.

I.11.4 Réduction de Vdd

Comme la particularité de la taille sur les processus de CMOS devient plus petite, il y a la pression pour réduire la tension d'alimentation. C'est parce que les matériaux constituants les transistors ne peuvent pas résister un champ électrique de force illimitée et comme les transistors deviennent plus petits la force des champs va s'augmente si la tension d'alimentation est tenue constante. Cependant, avec l'augmentation de l'intérêt de la conception spécifiquement pour une puissance réduite, cela peut être engageant pour une tension de d'alimentation d'être réduit plus rapidement qu'est nécessaire, seulement pour éviter les dégât électrique. Qu'est-ce qu'il empêche l'utilisation de basse tension d'alimentation ? Le problème avec la réduction Vdd va réduire aussi la performance du circuit. On donne le courant de saturation du transistor:

$$I_{sat} \propto (Vdd - V_t)^2 \qquad \text{(I.3)}$$

Où V_t est la tension de seuil du transistor. La charge sur un noeud de circuit est proportionnelle à Vdd, on donne la fréquence maximal de travail est:

$$f_{\max} \propto \frac{(Vdd - V_t)^2}{Vdd} \qquad \text{(I.4)}$$

Alors la réduction du Vdd va réduire la fréquence du travail. La perte de performance sur un processus de sous-micron ne peut pas être aussi sévère comme l'équation I.4 suggère puisque le courant à la haute tension peut être limité par les effets de saturation de vélocité, mais la performance sera perdue dans une certaine mesure. L'équation I.4 suggère qu'une façon

évidente d'améliorer la perte de performance soit de réduire V_t. Cependant le courant de fuite dépend fortement de V_t :

$$I_{leak} \propto \exp\left(-\frac{V_t}{35\ mV}\right) \qquad \text{(I.5)}$$

Même une petite réduction de V_t peut significativement augmenter le courant de fuite, augmentant le drain de batterie à travers un circuit inactif. Il y a donc un accord (*trade-off)* à être affecté entre la maximisation de la performance et la minimisation de la puissance de réserve (*standby power)*, Cette issue doit être considérée soigneusement par les designers de systèmes où toutes les deux caractéristiques sont importantes.

I.12 CONCLUSION

Toutefois, sur ce concept brut de RISC viennent se greffer de nouvelles idées telles que :

- Les instructions doivent rester d'une longueur fixe identique.
- les modes d'adressage doivent se simplifier via une architecture de registre à registre et des accès mémoire obtenus avec des instructions *laod* et *store* exclusivement, c'est-à-dire de chargement et de rangement, sans aucune autre compilation.
- il faut multiplier le nombre des registres internes dans le microprocesseur, des registres spécialisés mais aussi beaucoup de registres généraux pour limiter les aller et retour en mémoire.
- le format des instructions doit être sur trois opérandes. cela permet, en une seule instruction, de désigner à la fois la source et la cible, par exemple.
- Il convient d'utiliser intensivement des mémoires tampons, ou « cache », pour accélérer les opérations.

Comme nous constatons, le concept RISC n'est pas réduit à l'usage d'un jeu d'instructions simplifié mais va bien au-delà, de telle sort qu'il n'existe pas de définition fondamentale, courte et précise du RISC.
Il en résulte des architectures plus dépouillées, demandant moins de transistors et partant, exigeant moins de surface de silicium, ce qui est favorable en cascade aux bons rendements en production et à la baisse des prix .la simplification des puces a pour autre résultat la possibilité d'accroître les fréquences d'horloge, et réduire la consommation.

Pendant de nombreuses années, ces concepts suscitent d'innombrables discussions portant en particulier, sur l'architecture « superscalaire »des processeurs. Toutefois, les réalisations vont se multiplier.

CHAPITRE II

STRUCTURE ET ORGANISATION INTERNE DU PROCESSEUR ARM

II.1 INTRODUCTION

L'organisation interne du processeur ARM n'a pas connu une grande changement commencent par premiers dispositifs à 3 micron développés à *Acorn computer* entre 1983 et 1985 jusqu'à le ARM6 et ARM7 développé par ARM Ltd (*ARM limited*) entre 1990 et 1995. Ces processeurs sont conçus avec pipeline à 3 étages. Au cours de cette période l'exploitation de la technologie CMOS et particulièrement les processus des tailles réduites a amélioré effectivement les performances de ces cœurs, mais les principes de base de fonctionnement est restée en grande partie le même.

Depuis 1995 plusieurs nouveaux coeurs du ARM ont été conçus avec des performances significativement plus hautes à l'aide du pipeline à 5 étages et avec une mémoire dont les espaces des instructions et des données sont séparées. [4]

Ce chapitre inclut les descriptions des structures internes de ces deux styles de base du cœur ARM et couvre les principes généraux d'opération des pipelines à 3 étages et à 5 étages et quelques détails de l'implémentation. Et ainsi une étude détaillée du ARM 7 et ARM8 et présentation des quelques types du processeur ARM existés.

II.2. L'ORGANISATION INTERNE DU PROCESSEUR ARM

II.2.1 Pipeline à 3 étages

L'organisation de pipeline à 3 étages du ARM est illustrée dans la figure II.1. Les composants principaux sont :

- **La banque de registre** (*registre bank*)**:** Qui maintient l'état du processeur. Il a deux ports de lecture et un pour l'écriture. Facilitant l'accès à n'importe quel registre, plus un port complémentaire de lecture et un port complémentaire d'écriture qui donne l'accès spécial à r 15, (le compteur du programme). (Le port complémentaire de l'écriture sur r 15 le permet de s'incrémenter pendant la recherche de l'adresse, et le port de l'écriture permet à l'instruction de recherche de reprendre après qu'une adresse de données a été tirée.)
- **Baril à décalage :** (*barrel shifter*) : peut décale ou faire tourner un opérande par n'importe quel nombre de bits.
- **L'ALU :** qui exécute les fonctions arithmétiques et logiques exigées par le jeu d'instruction.
- **Le registre d'adresse et l'incrémenter :** choisit et maintiennent toutes les adresses de la mémoire et produisent des adresses séquentielles si nécessaire.
- **Les registres de donnée :** contient les données passant à et de la mémoire.
- **Le décodeur d'instruction :**(et l'unité de contrôle associée).

Dans un seul cycle d'instruction de traitement de donnée, deux registres d'opérande ont eus accès, La valeur sur le bus B est décalée et combinée avec la valeur sur le bus A de l'ALU, alors le résultat est écrit différemment sur la banque du registre. La valeur sur le compteur du programme est écrit dans le registre d'adresse, d'où il va exciter l'incrémenter, alors la valeur incrémentée est copiée dans rl5 dans la banque de registre et aussi dans le registre d'adresse pour être employé comme l'adresse de l'instruction à rechercher suivante.

II.2.1.a) Les 3 étages de pipeline :

Les processeurs ARM jusqu'à l'ARM7 emploient une structure simple de pipeline à 3 étages:[1]

- **1. recherche de l'instruction** *(fetch)* **:**
  L'instruction est recherchée de la mémoire et placée dans le pipeline d'instruction.
- **2. Décodage de l'instruction** *(decode)* **:**
  L'instruction est décodée et les signaux de contrôle de chemin de données sont préparés pour le cycle suivant.
- **3. Exécution de l'instruction** *(execute)* **:**
  L'instruction possède un chemin de donnée ; la banque de registre lit les opérandes, un opérande est décalé selon l'opération, le résultat de l'ALU produit et écrit différemment (written back) dans un registre de destination.

Notons que, trois instructions différentes peuvent occuper chacune de ces étages, donc le dispositif de chaque étage doit être capable d'accomplir des opérations indépendamment.

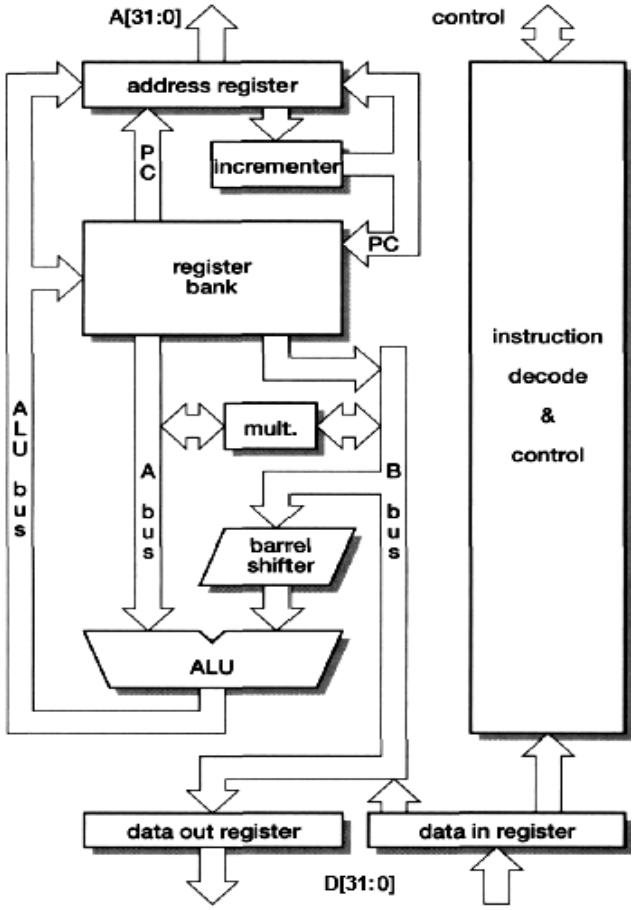


Figure II.1 Organisation de pipeline à 3 étages du ARM.

Quand le processeur exécute des instructions de traitement de donnée simple le pipeline permet à une seule instruction d'être s'achever à chaque cycle d'horloge. Une instruction individuelle prend trois cycles d'horloge pour achever, donc il y a une **latence** à trois cycles, mais la sortie est une instruction par cycle. La figure II.2 montre une opération d'un pipeline à 3 étages pour une instruction d'un seul cycle.

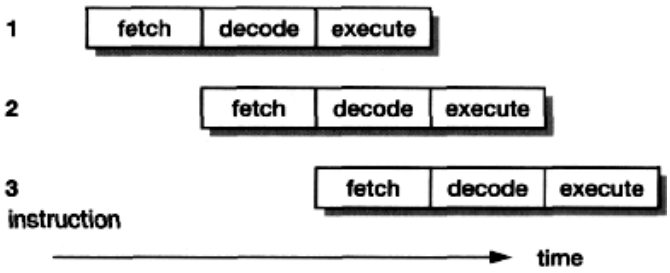


Figure II.2 opération d'un pipeline à 3 étages d'une instruction à un seul cycle du ARM

Quand une instruction à multi-cycle est exécutée le flux est moins régulier comme il est illustré dans la figure II.3. Cela montre l'apparition d'une séquence de l'instruction ADD avec une instruction d'arrangement de données, STR, d'un seul cycle arrivant après le

premier ADD. On montre en gris les cycles qui ont accès à la mémoire principale donc on peut voir que la mémoire est employée dans chaque cycle. Le chemin de données est de même employé dans chaque cycle, étant impliqué dans tous les cycles d'exécution, le calcul d'adresse et le transfert de données. L'unité de décodage produit toujours les signaux de contrôle pour le chemin de données afin de l'employer dans le cycle suivant, donc en plus des cycles explicites de décodage il produit aussi le contrôle pour le transfert de données pendant le cycle de calcul d'adresse du STR.

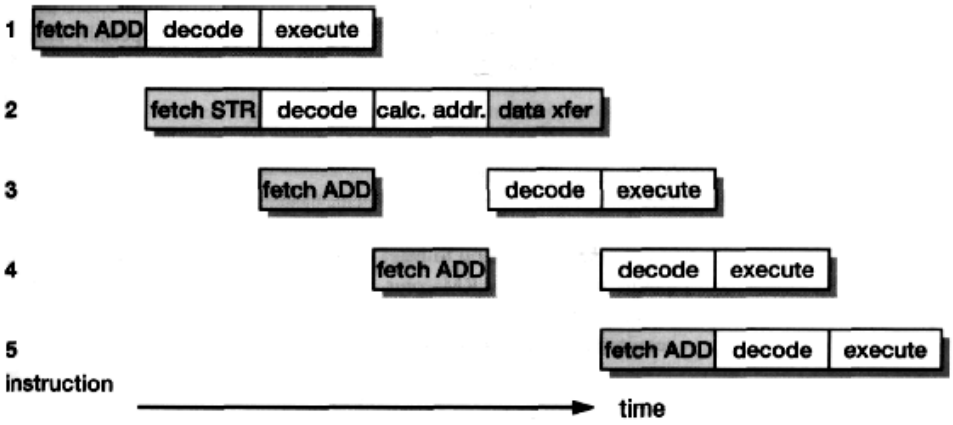


Figure II.3 Opération d'un pipeline à 3 étages d'une instruction à multi-cycle.

Ainsi, dans ce déroulement de l'instruction, tous les composants du processeur sont actifs dans chaque cycle et le facteur limitant c'est la mémoire, Définissant le nombre de cycles que la séquence doit prendre.

II.2.1.b) Comportement du PC

Une conséquence du modèle d'exécution de pipeline employé par le processeur ARM est que le compteur du programme, qui est visible pour l'utilisateur comme rl5, doit incrémenter en avance de l'instruction actuelle. Si, comme noté ci-dessus, les instructions vont chercher l'instruction suivante, mais pendant leur premier cycle, cela suggère que le PC doit indiquer huit octets (deux instructions) en avance de l'instruction actuelle.
C'est, en effet, ce qui arrive et le programmeur qui essaye d'avoir accès au PC directement par rl5 doit tenir compte de l'exposition du pipeline ici. Cependant, pour les buts les plus normaux l'assembleur ou le compilateur manipule tous les détails.

Le comportement Même plus complexe est exposé si rl5 est employé plus tard que le premier cycle d'une instruction, puisque l'instruction incrémentera le PC pendant son premier cycle. Une telle utilisation du PC n'est pas souvent avantageuse donc la définition d'architecture du ARM spécifie le résultat comme 'imprévisible' et il doit être évité, d'autant plus que des ARM nouveaux n'ont pas le même comportement dans ces cas.

II.2.2 Pipeline à 5 étages

Tous les processeurs doivent se développer pour satisfaire la demande d'une plus haute performance. Le pipeline à 3 étages employé dans les ARMs jusqu'à le ARM7 est très avantageux, mais il exige que l'organisation de processeur soit repensée. On donne le temps, T_{prog} , exigé pour exécuter un programme donné:

$$T_{prog} = \frac{N_{inst} \times CPI}{f_{clk}} \qquad \text{(II.1)}$$

Où N_{inst} est le nombre des instructions exécutées du ARM au cours du programme, le CPI est le nombre moyen du cycle d'horloge par instruction et f_{clk} est la fréquence d'horloge du processeur. Tandis que N_{inst} est constant pour un programme donné (compilé Avec un compilateur donné employant un jeu donné d'optimisations, etc.) il y a seulement deux façons d'augmenter la performance :

- Augmentant le taux d'horloge, f_{clk} Cela exige la logique dans chaque étage de pipeline à être simplifié et, donc, le nombre d'étages de pipeline à être augmenté.
- Réduisant le nombre moyen de cycles d'horloge par instruction (CPI).

Un pipeline typique du processeur à 5 étages est employé dans l'ARM9TDMI. L'organisation de l'ARM9TDMI est illustrée dans la Figure II.4.

Un pipeline à 5 étages du ARM a la structure suivante :

- **1. recherche de l'instruction** (*fetch*)
  L'instruction est cherchée de la mémoire et placée dans le pipeline d'instruction.
- **2. Décodage** (*decode*)
  L'instruction est décodée et des opérandes sont lus du fichier de registre. Il y a trois ports de lecture de l'opérande dans le fichier de registre, donc la plupart des instructions ARM ont reçus la source de tous leurs opérandes dans un cycle.
- **3. Exécution** *(execute)*
  Un opérande est changé et le résultat d'ALU produit. Si l'instruction est une charge ou rangement l'adresse de mémoire est calculée dans l'ALU.
- **4. Tampon de données** *(Buffer data)*
  La mémoire de données aura eue accès si nécessaire. Autrement le résultat de l'ALU est simplement amortit pour un cycle d'horloge pour donner le même flux de pipeline pour toutes les instructions.
- **5. Ecriture différée** *(write back)*
  Les résultats produits par l'instruction sont écrits dans le champ des registres, incluant n'importe quelles données chargées de la mémoire.

Ce pipeline à 5 étages a été employé pour beaucoup de processeurs RISC et est considéré pour être la façon 'classique' de concevoir un tel processeur.

II.2.2. a) Expédition des données (*data forwarding*)

Une source principale de complexité dans le pipeline à 5 étages (comparé au pipeline à 3 étages) est que l'exécution d'instruction est étendue à travers trois étages de pipeline, la seule façon de résoudre le problème des dépendances de données sans caler le pipeline est d'introduire des chemins d'expédition

Les dépendances de données surgissent quand une instruction doit employer le résultat d'un de ses prédécesseurs avant que ce résultat ne soit renvoyé dans le champ des registres. L'expédition de chemins permet aux résultats d'être passé entre des étages immédiatement qu'ils sont disponibles et le pipeline à 5 étages du ARM requiert que chacun des trois opérandes source à être expédié de chacun de trois registres de résultat intermédiaires comme indiqué dans la Figure II.4.

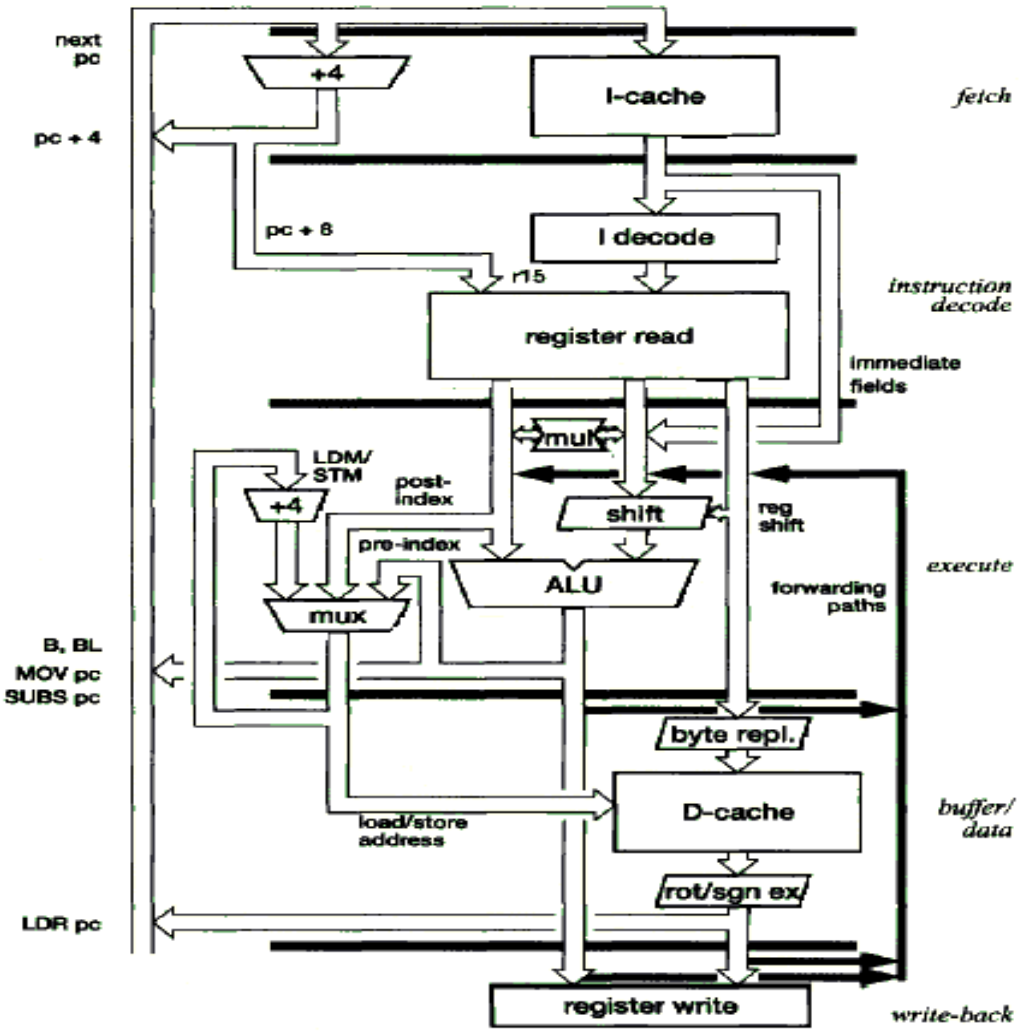


Figure II.4 Organisation du pipeline à 5 étages du ARM9TDMI

II.2.2. b) Comportement du PC

Le comportement de rl5, comme vu par le programmeur et décrit dans ' le comportement de PC ' est basé sur les caractéristiques opérationnelles du pipeline à 3 étages du ARM. Le pipeline à 5 étages va lire les opérandes d'instruction à un étage plus tôt dans le pipeline et acquerrait évidemment une valeur différente (PC+4 plutôt que PC+8). Cela veut dire des incompatibilités de code inacceptables, cependant, le pipeline à 5 étages de ARM tout émule le comportement des conceptions plus ancien de 3 étages référant à la figure II.4. La valeur de PC incrémentée de l'étage de la recherche de l'instruction est inscrite directement sur le champ des registres dans l'étage de décodage, by-passant le registre de pipeline entre les deux étages. PC+4 pour l'instruction suivante est égal à PC+8 pour l'instruction actuelle, alors la valeur adéquate du rl5 est obtenue sans matériel complémentaire.

II.3 EXECUTION DES INSTRUCTIONS DU ARM

L'exécution d'une instruction du ARM peut le mieux être comprise en ce qui concerne l'organisation des chemins de donnée comme il est présenté dans la figure II.1. Nous emploierons une version annotée de ce diagramme, omettant la section de logique de contrôle et mettant en évidence les bus actifs pour montrer le mouvement d'opérandes autour des unités diverses dans le processeur. Nous commençons par une instruction de traitement de données simple.

II.3.1 Instruction de traitement de données

Une instruction de traitement de données requiert deux opérandes, dont l'un est toujours un registre et l'autre est un deuxième registre ou bien une valeur immédiate. On passe le deuxième opérande par le baril à décalage où c'est soumis à une opération de décalage générale, alors il est combiné avec le premier opérande dans l'ALU. Finalement, le résultat de l'ALU est écrit dans le registre de destination.

Toutes ces opérations s'exécutent dans un seul cycle d'horloge comme indiqué dans la figure II.5. Notons aussi comment la valeur du PC dans le registre d'adresse est incrémentée et copiée différemment, au même temps dans le registre d'adresse rl5 et dans le banque de registres. L'instruction suivante est seulement chargée dans le fond du pipeline d'instruction (i. Pipe). La valeur immédiate, quand elle est requit, est extraite de l'instruction actuelle au sommet du pipeline d'instruction. Pour des instructions de traitement de données seulement le fond de huit bits (bits [7:0]) de l'instruction sont employées dans la valeur immédiate.

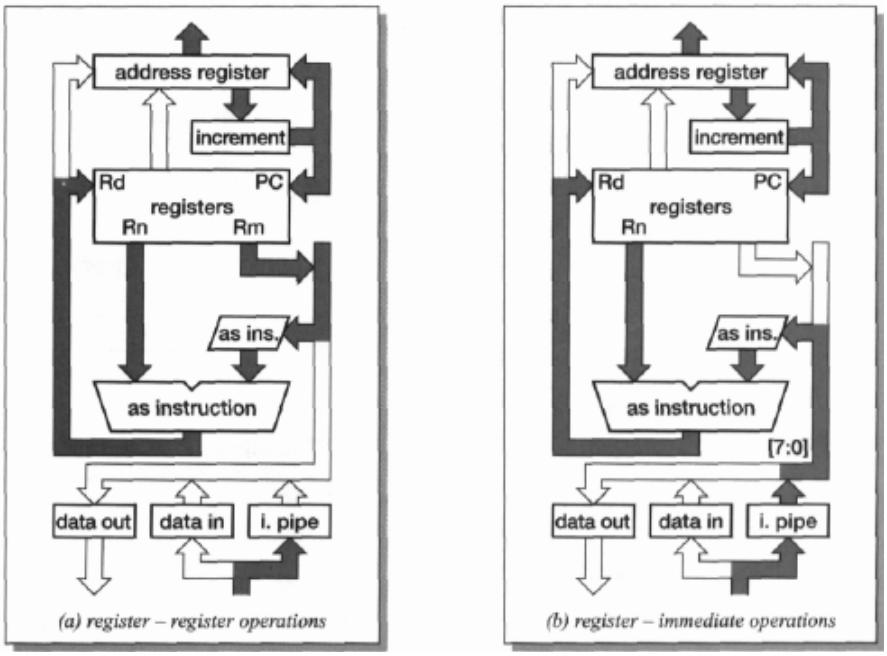


Figure II.5 Comportement de chemin de données pour une instruction de traitement de donnée.

II.3.2 Instruction du transfert de données

Les instruction du transfert de données (chargement ou rangement) calcule une adresse du mémoire d'une même manièrequ'une instruction de traitement de données calcule son résultat. Un registre est employé comme l'adresse de base, à laquelle est ajouté (ou soustrait) une donnée qui peut de nouveau être un autre registre ou une valeur immédiate. Cependant, une valeur à 12 bits immédiate est employée sans une opération de décalage plutôt qu'une valeur décalée à 8 bits. L'adresse est envoyée au registre d'adresse et dans un deuxième cycle le transfert de données a lieu. Le chemin de donnée est en grand parti inoccupé pendant le cycle de transfert de données, l'ALU contient les composants d'adresse du premier cycle et est disponible pour calculer une modification d'auto-indexation au registre de base si c'est nécessaire. (Si l'auto-indexation n'est pas nécessaire la valeur calculée n'est pas écrite déferrement au registre de base dans le deuxième cycle.)

On montre l'agissement de chemin de données pour les deux cycles d'une instruction de rangement de données (STR) avec une offset immédiate dans la Figure II.6. Notons comment la valeur de PC incrémentée est stockée dans le banque de registre à la fin du premier cycle pour que le registre d'adresse soit libre d'accepter une adresse de transfert de données pour le deuxième cycle, ensuite à la fin du deuxième cycle la valeur de PC est inscrite différemment dans le registre d'adresse pour permettre à l'instruction pré-recherche (*prefetching*) de poursuivre.

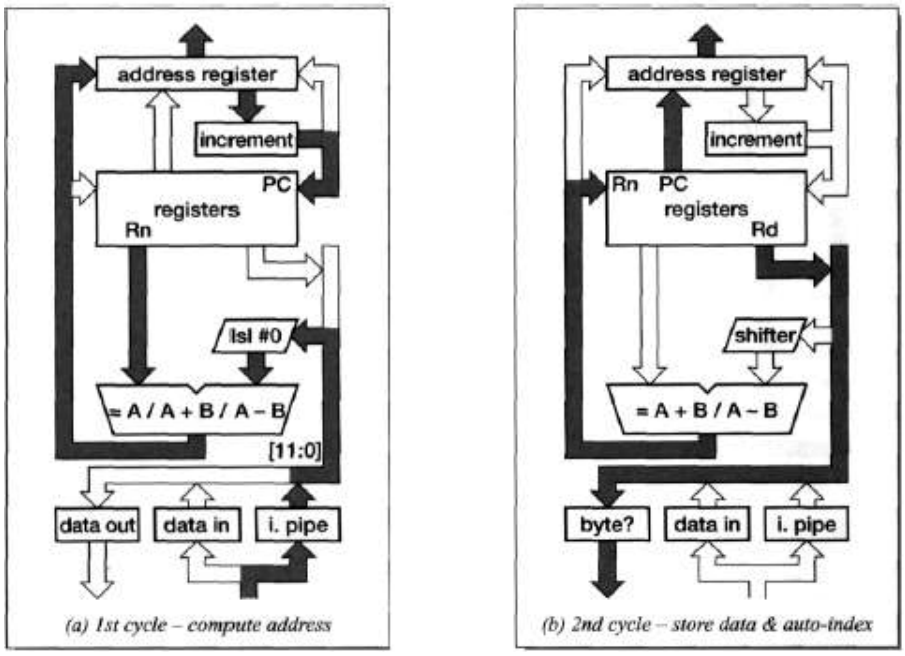


Figure II.6 Comportement de chemins de données pour une instruction de transfert de données

Il doit être noté à cet étage que la valeur envoyée au registre d'adresse dans un cycle est la valeur employée pour l'accès du mémoire dans le cycle suivant.
Le registre de l'adresse est, en effet, un registre de pipeline entre les chemins de donnée du processeur et la mémoire externe.

Quand l'instruction spécifie un rangement d'un type de donnée d'un octet, ‘ les données sortant ' le bloc extraient l'octet de fond de registre et le reproduisent quatre fois à travers le bus de données de 32 bits. La logique de contrôle du mémoire externe peut alors employer les deux bits du poids faible de bus d'adresse pour activer l'octet approprié dans le système de mémoire.
Les instructions de chargement suivent un modèle semblable sauf que les données de la mémoire seulement arrivent au même temps que ' les données entrant ' le registre sur le deuxième cycle et un troisième cycle est nécessaire pour transférer les données de là mémoire au registre de destination.

II.3.3 Instruction du branchement

Les instructions de branchement calculent l'adresse cible dans le premier cycle comme indiqué dans la Figure II.7. Un champ à 24 bits immédiats est extrait de l'instruction et ensuite décalé à gauche de deux positions de bit pour donner une offset alignée de mot qui va ajouter au PC. Le résultat est utilisé comme une adresse de recherche d'instruction et tandis que le pipeline d'instruction est occupé, à nouveau l'adresse de retour est copié dans le registre de lien (rl4) si c'est requis (c'est-à-dire si l'instruction est ' un branchement avec lien (*branch with link)).*

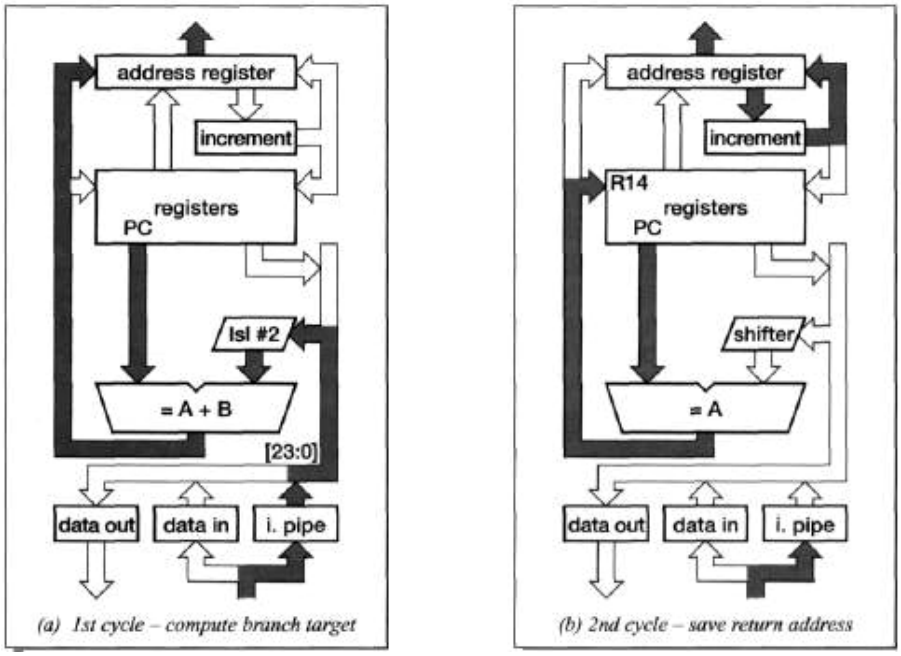


Figure II.7 Comportement des chemins de données pour une instruction du branchement

Le troisième cycle, qu'est requis pour achever le remplissage de pipeline, à nouveau, est aussi employé pour faire une petite correction à la valeur stockée dans le registre de lien (*link register*) afin qu'il indique directement à l'instruction qui suit la branche. C'est nécessaire parce que rl5 contient le PC + 8 tandis que l'adresse de l'instruction suivante est le PC + 4.

II.4 MISE EN OEUVRE DU ARM

La mise en oeuvre du ARM y suit une approche semblable décrit dans le Chapitre 1 pour MUO; la conception est divisée dans une section de chemin de données qui est décrite dans le niveau de transfert de registre (RTL *register transfer level)* et une section de contrôle qui est vue comme une machine finie d'état (FSM *finite state machine*)

II.4.1 Arrangement de chronométrage (*Clocking scheme*)

La plupart des ARMs ne fonctionnent pas avec un bord des registres sensibles (*edge-sensitive register*); au lieu de cela la conception est basée autour des horloges non-chevauché à 2 phases, comme indiqué dans la Figure II.8, qui est produit intérieurement à partir d'un signal d'horloge d'entrée simple. Cet arrangement permet l'utilisation de latches transparentes sensibles à niveau. Le mouvement de données est contrôlé en passant les données alternativement par les latches qui sont ouverts pendant la phase 1 et les latches qui sont ouverts pendant la phase 2. La propriété de non-chevauchement de la phase 1 et la phase 2 de l'horloge assurent qu'il n'y a aucune condition de parcours dans le circuit.

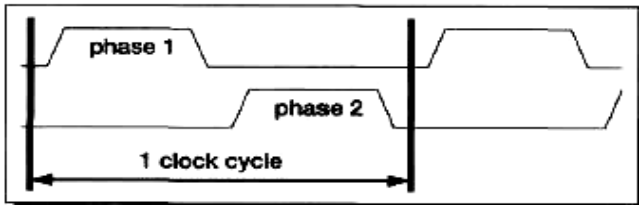


Figure II.8 Signale de l'horloge de non-chevauchement à 2 phases.

II.4.2 Timing de chemin de données (*datapath timing*)

Le timing normal des composants de chemin de données dans un pipeline à 3 étages est illustré dans la Figure II.9. Les bus de lecture du registre sont dynamiques et sont préchargé pendant la phase 2 (ici 'dynamiques' signifie qu'ils n'ont conduit pas parfois et conservent leurs valeurs de logique comme une charge électrique; les circuits de conservation de charge sont employés pour donner le comportement pseudo-statique pour que les données ne soient pas perdues si l'horloge est arrêtée à n'importe quelle point de son cycle). Quand la phase 1 au niveau haut, les registres sélectionnées déchargent les bus de lecture qui deviennent valides ensuite dans la phase 1. Un opérande passe à travers le baril à décalage (*barrel shifter*), qui emploie aussi des techniques dynamiques et la sortie de baril à décalage devient valide un peu plus tard dans la phase 1. L'ALU a des latches d'entrée qui sont ouverts pendant la phase 1, permettant aux opérandes de commencer à se combiner dans l'ALU instantanément qu'ils sont valides, mais ils se ferment à la fin de la phase 1. L'ALU continue alors à traiter les opérandes pendant la phase 2, produisant une sortie valide vers la fin de la phase laquelle sera connecté à travers latche au registre de destination à la fin de la phase 2.
Notons comment, les données passent par les latches d'entrée de l'ALU, ceux-ci n'affectent pas le timing du chemin des donnés puisqu'ils sont ouverts quand des données valides arrivent. Cette propriété de latches transparentes est exploitée en beaucoup de places dans la conception du ARM pour assurer que les horloges ne ralentissent pas les signaux critiques.
La durée minimal d'un cycle du chemin de données est donc la somme de : [4], [5]

- le temps de lecture du registre;
- Le retard du décalage;

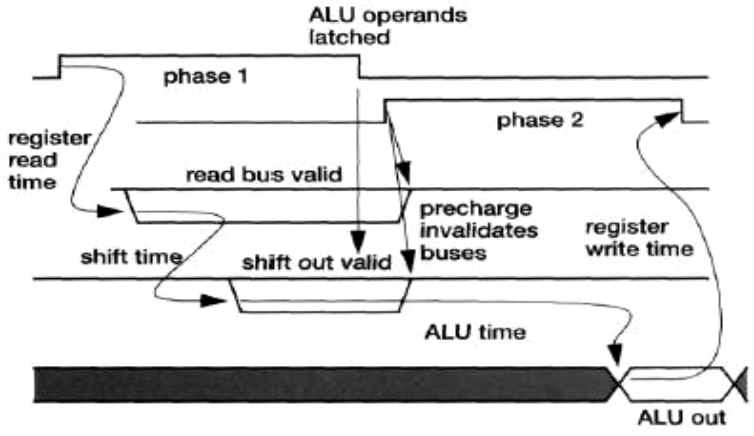


Figure II. 9 Timing de chemin de données

- Le retard de l'ALU;
- le temps de l'écriture du registre;
- le temps de non-chevauchement entre la phase 2 et la phase 1.

De ceux-ci, le retard de l'ALU domine. Le retard de l'ALU est fortement variable, selon l'opération qu'il exécute. Des opérations logiques sont relativement rapide, puisqu'ils n'impliquent pas la propagation de retenu. Des opérations arithmétiques (l'addition, la soustraction et des comparaisons) impliquent de plus longs chemins de logique comme le retenu peut se propager à travers la largeur du mot.

II.4.3 Conception de l'additionneur

Le premier prototype de processeur ARM a employé un simple (*ripple-carry adder*) additionneur à carry d'ondulation comme indiqué dans la Figure II.10. L'inconvénient de ce concept est dans le cas de 32 bits ou le chemin du cary sera plus long. [4]

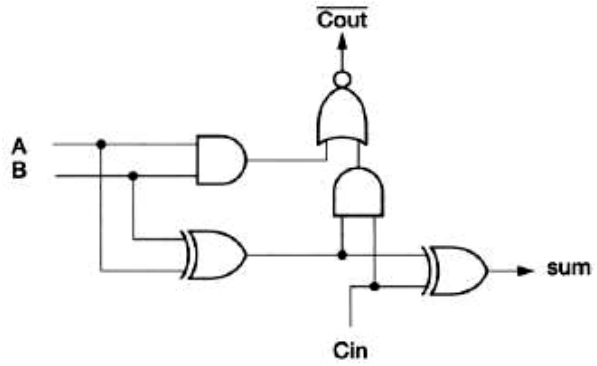


Figure II.10 Additionneur à carry d'ondulation du ARM1

Pour éviter ce problème et permettre un taux d'horloge plus haut, ARM2 utilise 4 bits pour le carry (*4 bits carry look-ahead)*) pour réduire la longueur de chemin de retenu. On montre le circuit dans la Figure II.11. La logique génère les signaux du carry généré (G) et propagé (P) qui contrôlent les 4 bits de carry de sortie. La longueur du chemin de propagation du carry est réduite à huit retards de la porte, employant de nouveau des portes auxiliaires "et – ou - inverser" et la logique alternant ET-OU.

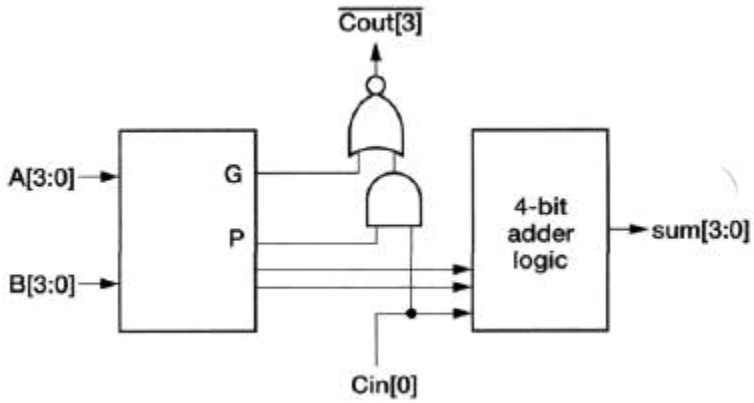


Figure II.11 Schéma d'additionneur de carry à Regard en avant

Une nouvelle amélioration temps a été introduite sur l'ARM6 employant un additionneur de carry sélectif (*carry-select adder*). Cette forme d'additionneur calcule les sommes de différents champs du mot pour un carry entrant, de zéro et un ensemble et ensuite le résultat final est choisi en employant la valeur correcte de carry entrant pour contrôler le multiplexeur. Le schéma complet est illustré dans la Figure II.12.

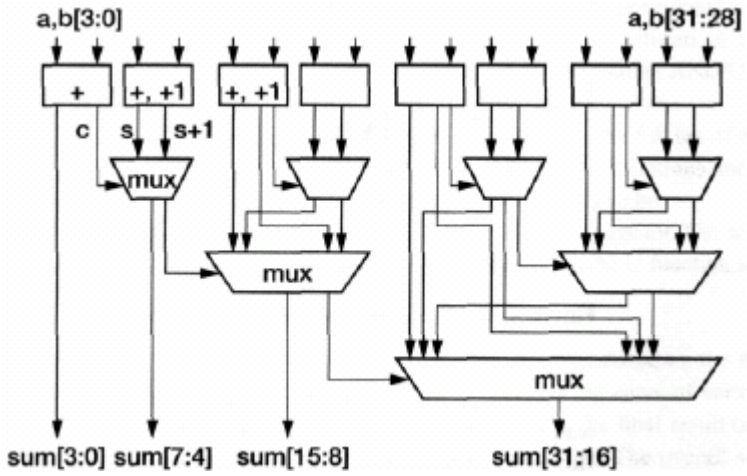


Figure II.12 Schéma d'additionneur sélectif utilisé dans l'ARM6

II.4.4 ALU

L'ALU n'additionne pas seulement ses deux entrées. Il doit exécuter un jeu d'opérations de données définies par le jeu d'instruction, incluant des calculs d'adresse pour des transferts de mémoire, des calculs de branche, des fonctions logiques, etc.
Un exemple d'un logique du ALU du ARM2 est illustré dans la Figure II.13. Le jeu de fonctions produites par cet ALU et les valeurs associées de sélection de fonction de l'ALU est inscrit dans le Tableau II.1.

Tableau II.1 codes des fonctions de l'ALU du ARM2.

Fs5	fs4	fs3	fs2	fs1	fs0	sorties de ALU
0	0	0	1	0	0	A and B
0	0	1	0	0	0	A and not B
0	0	1	0	0	1	A xor B
0	1	1	0	0	1	A + not B + carry
0	1	0	1	1	0	A + B + carry
1	1	0	1	1	0	not A + B + carry
0	0	0	0	0	0	A
0	0	0	0	0	1	A or B
0	0	0	1	0	1	B
0	0	1	0	1	0	not B

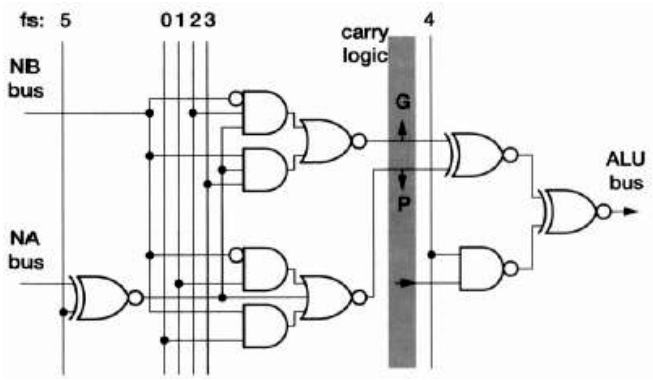


Figure II.13 Logique de ALU du ARM2 pour un résultat

Un autre exemple de la Figure II.14 présente l'organisation de l'ALU du ARM6. Les opérandes d'entrée sont chacun sélectivement inversés, ensuite additionnés et combinés dans l'unité logique et finalement le résultat sollicité est choisi et écrit sur le bus de résultat de l'ALU.

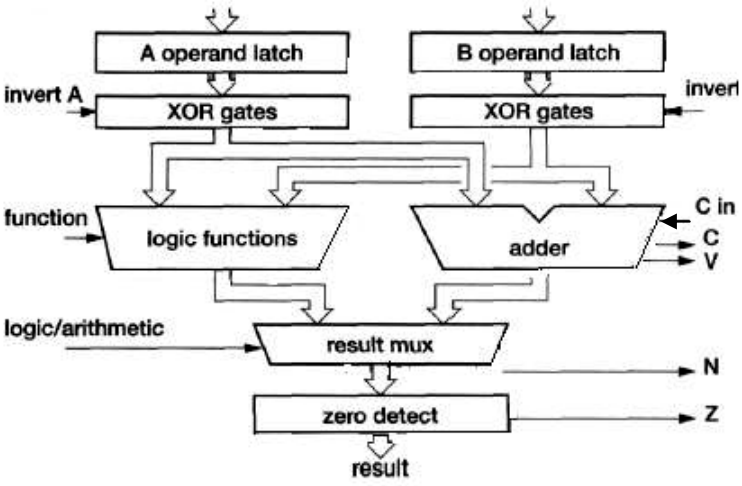


Figure II.14 Organisation de l'ALU du ARM6

les drapeaux C et V sont générés dans l'additionneur (ils n'ont aucune signification pour les opérations logiques), le drapeau N est copié du bit 31 du résultat et le drapeau Z est évalué du bus de résultat entier. Notons que la production du drapeau Z exige une porte NOR à 32 entrées et cela peut facilement devenir un signal de chemin critique.

II.4.5 Le baril à décalage (*barrel shifter*)

L'architecture du ARM supporte les instructions qui exécutent une opération de décalage en série avec une opération de l'ALU, suivant à l'organisation montrée dans la Figure II.1. La performance du baril est donc critique puisque le temps de décalage contribue directement au temps du cycle du chemin de données comme indiqué dans le diagramme de la FigureII.9. Pour réduire au minimum le retard par le baril, Une matrice de commutateur de barre (*cross-bar switch matrix*) est employée pour diriger chaque entrée à la sortie appropriée. Le principe du commutateur de barre est illustré dans la Figure II.15, où une matrice 4 x 4 est montrée. (Les processeurs ARM utilisent une matrice de 32 x 32.) Chaque entrée est connectée à chaque sortie par un commutateur. Chaque commutateur peut être implémenté comme un transistor NMOS simple.

Les fonctions de décalage sont réalisées en reliant des commutateurs le long des diagonales à une entrée de contrôle commune :

- Pour une fonction de décalage à gauche ou à droite, une diagonale est activée. Cela connecte tous les bits d'entrées à leurs sorties respectives où ils sont employés. le baril à décalage fonctionne en logique où '1' est représenté comme une masse et '0' par un potentiel d'alimentation.
- Pour une fonction de rotation à droite, on permet la diagonale à décalage à droite au même temps d'activée avec le diagonale à décalage à gauche complémentaire
- Le décalage à droite arithmétique utilise une extension de signe. une logique séparée est employée pour décoder le nombre de décalage et activer la sortie appropriée.

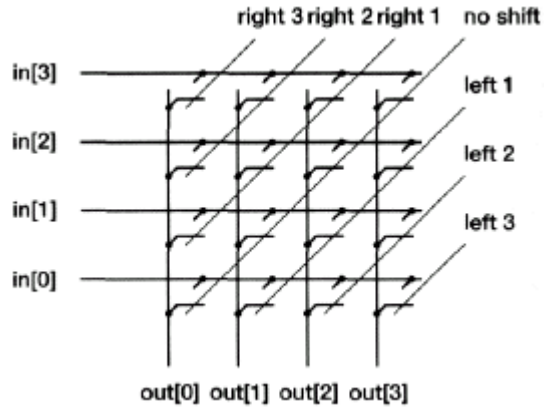


Figure II.15 Principe du commutateur de barre de baril à décalage

II.4.6 La conception de multiplicateur (*multiplier design*)

Tous les processeurs ARM sauf le premier prototype ont inclus un support matériel pour la multiplication d'entier. Deux types de multiplicateur ont été employés :

- les plus ancien coeurs ARM incluent un support matériel de multiplication moins coûteux qui supporte seulement le résultat de multiplication de 32 bits et des instructions de multiplication-accumulation (*multiply-accumulate*).
- Des coeurs ARM récents ont un matériel de multiplication très performant et supporte un résultat de multiplication de 64 bits et des instructions multiplication-accumulation (*multiply-accumulate*).

Le support d'un coût réduit emploie le chemin de donnée principal itérativement, employant le baril à décalage et l'ALU pour produire un produit à 2 bits dans chaque cycle d'horloge. Le logique de terminaison va arrêter les itérations quand il n'y a rien dans le registre de multiplication. Lorsque la performance de multiplication est très importante, plus de ressource de matériel doivent y être employée. Dans quelques systèmes embarqués le coeur ARM est employé pour accomplir le traitement numérique du signal en temps réel (DSP) en plus des fonctions de contrôle général. Les opérations du DSP utilisent typiquement une multiplication intensive. Alors la performance du matériel de multiplication peut être critique à cause des contraintes en temps réel.

La multiplication très performante employée dans quelques coeurs de ARM emploie une représentation largement employée superflue binaire pour éviter les retards dus à la propagation du carry (*carry-propagate*) associés à l'addition partielle de produits. Les résultats intermédiaires sont tenus comme des sommes partielles et carry partiel où le vrai résultat binaire est obtenu en ajoutant ces deux ensembles dans des additionneurs à carry propagé comme dans ALU principal, mais ce la se fait une fois à la fin de la multiplication. Pendant la multiplication les sommes et les carrys partiels sont combinés dans des additionneurs à carry préservé (*carry-save)* où le carry se propage à travers un bit par étage additionneur. Cela donne un chemin logique beaucoup plus au additionneur à carry préservé que celle du additionneur à carry propagé.

Il y a plusieurs façons de construire un additionneur à carry préservé, mais le plus simple est la forme de 3 entrées à 2 sorties. Cela accepte comme des entrées une somme partielle, un carry partiel et un produit partiel, les sorties sont une nouvelle somme partielle et un nouveau carry partiel où le carry a deux fois le poids binaire de la somme. La figure II.16 illustre les deux structures. L'additionneurs à carry propagé prend deux nombres conventionnels (non redondants) binaires comme des entrées et produit une somme binaire; l'additionneurs à carry préservé prend un bit binaire et un redondant (somme partielle et report partiel) à l'entrée et produit une somme binaire redondant.

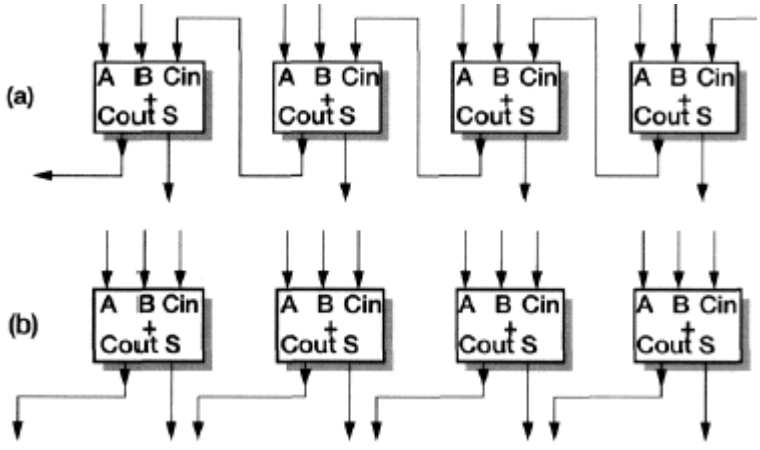


Figure II.16 Structures de l'additionneur à (a) carry propagé (b) carry

Pendant les étapes de multiplication itératives, la somme est injectée et combinée avec un nouveau produit partiel dans chaque itération. Quand tous les produits partiels ont été ajoutés, la représentation redondante est convertie à nombre binaire conventionnel en ajoutant la somme partielle et le carry partiel dans L'additionneur à carry propagé de l'ALU.

Des multiplicateurs ultra-rapides ont plusieurs couches d'additionneur à carry préservé en série, chacun a un produit partiel. On montre la structure complète d'un multiplicateur très performant employé par quelques coeurs du ARM dans la Figure II.17. La matrice du carry préservé a quatre couches d'additionneur, chacun contient deux bits de multiplicateur, donc La matrice peut multiplier huit bits par un cycle d'horloge. Les registres de la somme partielle et du carry sont vides au début de l'instruction, ou le registre de somme partiel peut être initialisé par la valeur contenu dans l'accumulateur. Quand le multiplicateur est décalé à droite de huit bits par un cycle dans le registre ' Rs', la somme partielle et le carry sont fait une rotation à droite de huit bits par un cycle.

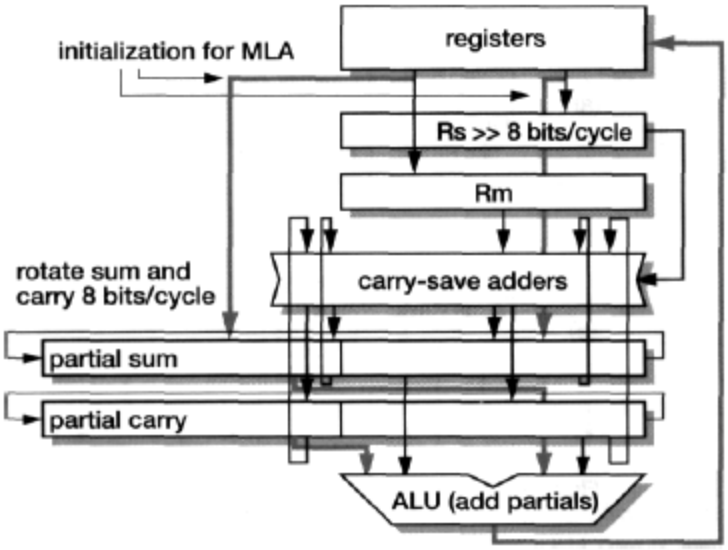


Figure II.17 Multiplicateur à haute vitesse du ARM

II.4.7 La banque de registres (*bank register*) :

Le dernier bloc principal du processeur ARM est la banque de registres. C'est où tout l'état visible d'utilisateur (*user-visible*) est stocké dans 31 registres généraux à 32 bits, peuvent stocker autour de 1 K Bit de données en tout. Puisque la cellule de registre à 1 bit de base est répétée plusieurs fois dans la conception, il vaut la peine de mettre un effort considérable dans la minimisation de sa taille. La figure II.18 montre le circuit de transistor de la cellule de registre employée dans des coeurs du ARM jusqu'à l'ARM6. La cellule de stockage est une paire asymétrique mutuelle-couplée des inverseurs CMOS qui sont surpiloté (*overdriven*) par un signal fort du bus ALU quand le contenu de registre est changé. L'inverseur de réactions est conçu d'une manière de réduire au minimum la résistance de la cellule à la nouvelle valeur. Les bus A et B de lecture sont pré chargé à Vdd pendant la phase 2 du cycle d'horloge, donc la cellule de registre a besoin seulement de la décharge des bus de lecture, qu'il fait par des transistors de type n de passage quand la ligne de lecture est valide.

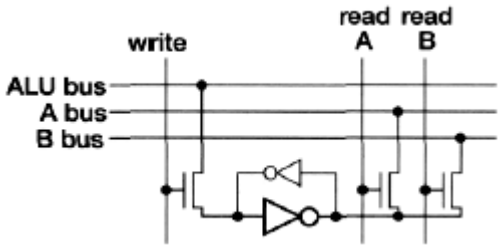


Figure II.18 Circuit de cellule de registre d'un ARM6

Ces cellules de registre travaillent avec 5 volts, et sont arrangées dans des colonnes pour former un registre à 32 bits et les colonnes sont empaquetées pour former une banque de registre complète. Les décodeurs des lignes de permutation pour la lecture et l'écriture sont alors empaqueté au-dessus des colonnes comme indiquées dans la Figure II.19, donc les lignes de permutation sont dirigés verticalement et les bus de données horizontalement à travers la matrice de cellules de registre. Le registre du compteur de programme (PC) du ARM fait physiquement partie du banque de registre dans les coeurs plus simples, mais il a deux ports d'écriture et trois ports de lecture, tandis que les autres registres ont un seul port d'écriture et deux ports de lecture. La symétrie de la matrice de registre est préservée en mettant le PC à la fin pour être accessible aux ports complémentaires.

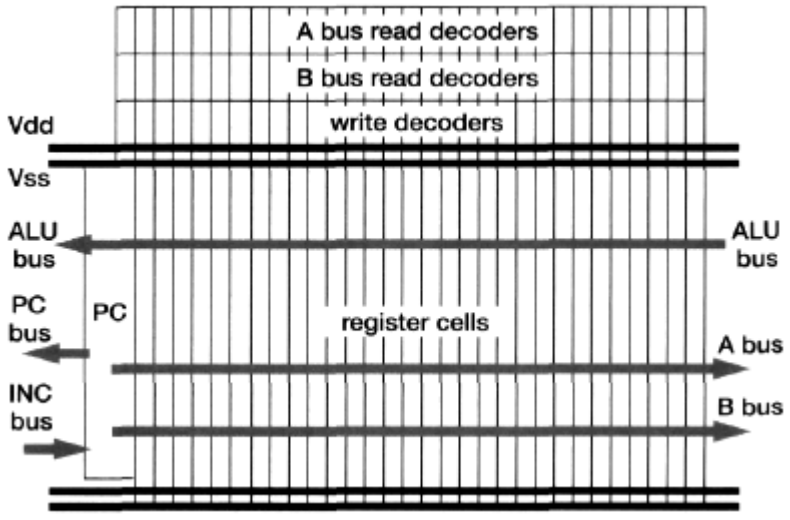


Figure II.19 Diagramme d'arrangement de la banque de registres.

II. 4. 8 Structures de contrôle

La logique de contrôle sur les coeurs du ARM plus simples est constituée de trois composants structurels, comme il est indiqué dans la Figure II.20.

1. Un décodeur d'instruction PLA (*programmable logic array*). Cette unité emploie quelques bits d'instruction et un compteur interne de cycle pour définir la classe d'opération qui doit être exécuté sur le chemin de données dans le cycle suivant.
2. Contrôle Distribué secondaire associé à chaque bloc principal de fonction du chemin de données. Cette logique emploie l'information de classe du décodeur principal PLA pour sélectionner d'autres bits d'instruction et/ou l'information de l'état du processeur pour contrôler le chemin de données.
3. Des unités de commande décentralisées pour les instructions spécifiques qui prennent un nombre variable des cycles pour achever (chargement et arrangement multiple, multiplication et les opérations du coprocesseur). Ici le décodeur principal PLA est verrouillé dans un état fixé jusqu'à l'unité de contrôle indique l'achèvement.

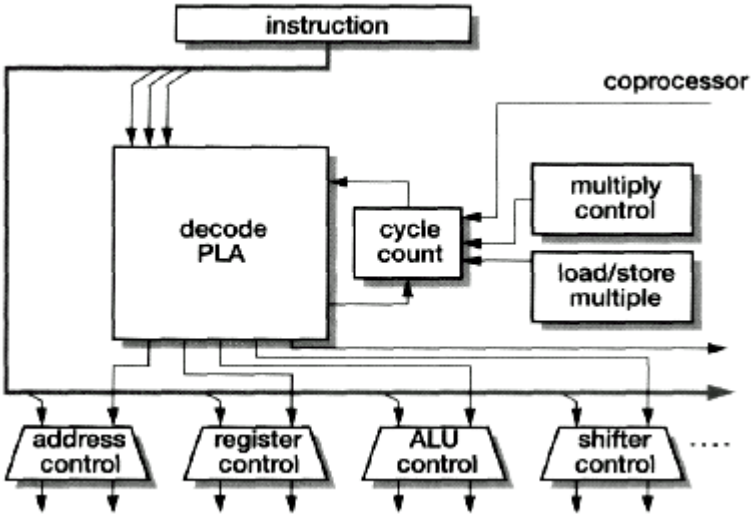


Figure II.20 Structure logique de contrôle du ARM.

II.4.9 Conception physique (*physical design*)

Jusqu'ici nous avons été concernés principalement par la conception de logique d'un coeur du ARM et nous n'avons pas parlé à son implémentation physique sur un processus de CMOS spécifique. Il y a deux mécanismes principaux pour implémenter un coeur de processeur ARM (ou un autre coeur) sur un processus spécifique :

- Une macrocellule matérielle (*hard macrocell*) est délivrée comme la disposition physique prête à être incorporé dans la conception finale;
- Une macrocellule logicielle (soft *macrocell*) est délivrée comme une conception synthétisable exprimée dans un langage de description du matériel comme le VHDL.

Une macrocellule matérielle peut être entièrement caractérisée sur le processus cible et peut exploiter les avantages du layout complètement personnalisé accordée à main (*full custom hand-tuned layout*), mais il peut être employé seulement sur le processus spécifique pour lequel il a été conçu. Le layout doit être modifiée et recaractérisée pour chaque nouveau processus. Une macrocellule logicielle peut facilement être transportée à une nouvelle technologie de processus, mais après chaque changement de processus il doit toujours être recaractérisé.

Les premiers coeurs du ARM ont été produits seulement comme des macrocellules matérielles. Leur conception était basée sur le chemin de données d'un conception complètement personnalisé (*full custom*), avec une logique de contrôle conçue au niveau schématique et converti au layout employant des outils de placement automatique et du routage et ainsi une bibliothèque de cellule standard. Pour soulager la portabilité du processus les coeurs ont été conçus employant des règles de conception génériques (pour la bibliothèque des cellules et la personnalisation complète du chemin de données) qui a permis aux transformations géométriques de même layout physique d'être employé pour dresser la carte de la même disposition physique sur une gamme du processus individuels avec des règles de conception semblable, mais pas identiques. Des coeurs du ARM récents ont été disponibles dans des formes matérielle et logicielle. Mac-rocells matérielle emploie de plus en plus la synthèse pour leur logique de contrôle en conservant le chemin de données complètement personnalisées et dessinées à main. Les macrocellules logicielles sont entièrement synthétisables par la description au niveau de transfert de registre (RTL). Le choix entre les macrocellules matérielles et logicielles. (Ou le netlists de niveau de porte) est une décision complexe. Des macrocellules matérielles peuvent clairement donner le meilleur surface, performance et l'efficacité d'énergie sur un processus, mais il prend un temps significatif, effort et coût. Des macrocellules logicielle et des netlists portatifs sont des outils plus flexibles et automatisés, sont maintenant d'une qualité qui signifie qu'ils viennent près de la layout concernant la performance.

II.5 LES TYPES DU COEUR ARM

Un coeur de processeur ARM dans un système est la machine qui va chercher des instructions de la mémoire et les exécute. Les coeurs du ARM sont très petits, occupant typiquement juste quelques millimètres carrés d'espace de puce. La technologie moderne VLSI permet à un grand nombre d'autres composants de système d'être incorporé sur la même puce. Ceux-ci peuvent être étroitement assemblés au coeur de processeur, comme la mémoire cache et le matériel de gestion de la mémoire, Ou ils peuvent être sans rapport aux composants de système comme le matériel de traitement de signal. Ils peuvent même inclure des nouveaux coeurs de processeur ARM. Parmi tous ces composants le coeur ARM est le composant le plus complexe, évidement nécessite un outil logiciel de développement et de mise au point.

Le bon choix d'un coeur ARM est une des décisions très critiques dans la conception d'un nouveau système. Dans les paragraphes suivantes on va décrire le cœur ARM7TDMI en détail, et en bref les autres principaux cœurs actuels.

5.1 ARM7TDMI

L'ARM7TDMI est le coeur ARM le plus largement utilisé à travers une gamme d'applications, notamment dans beaucoup de téléphones portables digitaux.

Il s'est développé du premier cœur ARM pour mettre en œuvre le modèle de la programmation d'un espace d'adresse du 32 bits, c'est l'ARM6. L'ARM6 a employé les techniques de circuit qui l'ont empêché de faire fonctionner sûrement avec une alimentation d'énergie de moins de 5 volts. L'ARM7 a corrigé ce manque et ensuite des instructions de multiplication de 64 bits, un support de mise au point sur chip, un jeu d'instruction compressé (*thumb instruction*) et un matériel du point de contrôle (*watchpoint*) de l'EmbeddedlCE était sont touts ajoutés pour donner la naissance du ARM7TDMI. L'origines du nom est comme suit :

- **ARM7**, c'est le cœur entier ARM6 à 32 bits, mais compatible de s'alimenter avec 3 v.
- **T** (**T***humb instruction*), un jeu d'instruction compressé à 16 bits
- **D** (on chip **D***ebug support*), un support de déboguage sur chip, permettant au processeur de faire halte en réponse à une demande de mise au point;
- **M** (***M****ultiplier*), Un Multiplicateur, avec plus haute performance que ses prédécesseurs et accomplirant d'un excessif résultat à 64 bits.
- **I** (*Embedded**I**CE*) matériel pour donner point de contrôle et vérification sur chip.

II.5.1.1 Organisation du ARM7TDMI

L'organisation du ARM7TDMI est illustrée dans la Figure II.21. Le ARM7TDMI est un coeur entier de base utilisant un pipeline à 3 étages avec quelques particularités importantes et extensions:

- Supporte le résultat à 64 bits de multiplication, chargement et rangement (load store) des octet signé et de demi mots, ainsi le jeu d'instruction compressée (thumb instruction set).
- inclut le module *EmbeddedlCE* (décrit dans le chapitre IV) pour supporter la mise au point du système incorporée.
- un port d'accès d'essai de JTAG, pour supporter l'accès du matériel de mise au point du système.

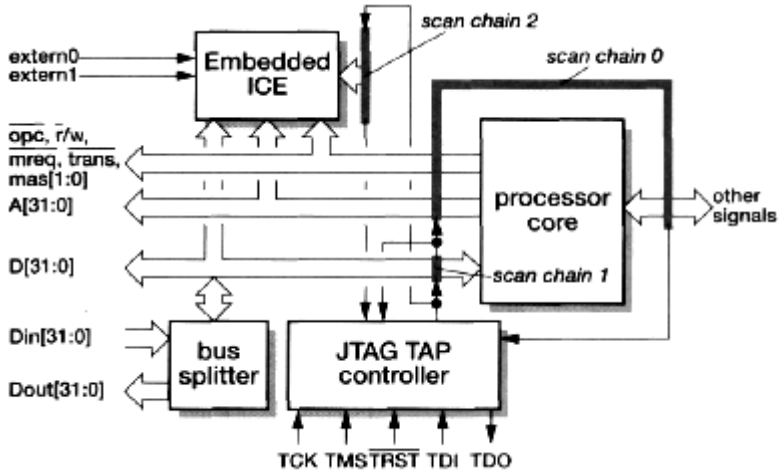


Figure II.21. Organisation du ARM7TDMI

Dans la figure II.22 on montre les signaux d'interface groupés par la fonction et le rôle de chaque groupe est décrit ci-dessous avec, quand c'est nécessaire, l'information sur les signaux particuliers et le chronométrage d'interface

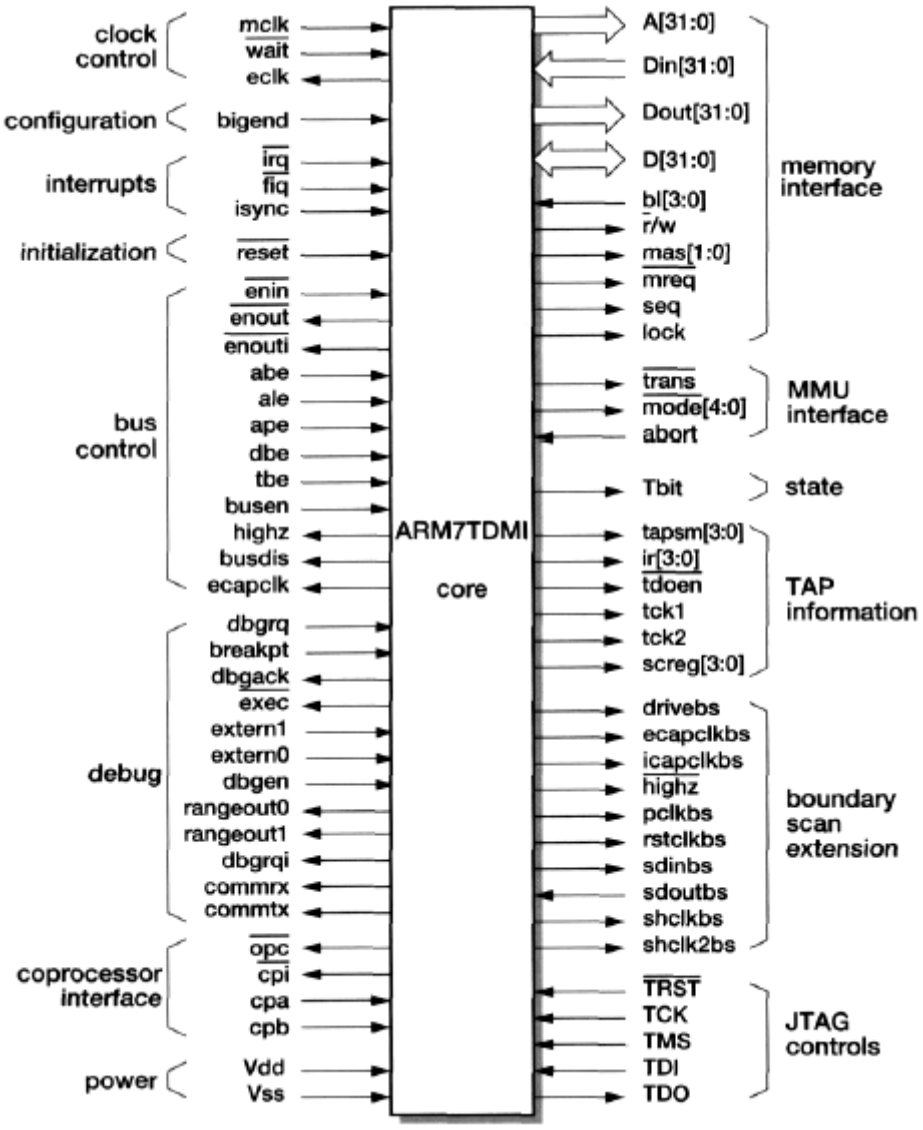


Figure II.22 Les signaux d'interface du cœur ARM7TDMI

- **Contrôle d'horloge :**

Tous les changements d'état dans le processeur sont contrôlés par mclk, l'horloge de la mémoire. Bien que cette horloge puisse être manipulée extérieurement pour causer l'attente de processeur pour un accès lent, il est souvent plus simple de fournir une horloge libre (*free-running clock*) et employer $\overline{wait}$ pour sauter des cycles d'horloge. L'horloge interne est fictivement un ET logique de mclk et $\overline{wait}$ donc $\overline{wait}$ doit changer seulement quand mclk est en bas.

- **Interface de la mémoire :**

L'interface de la mémoire comprend le bus d'adresse de 32 bits (A [31:O]), un bus bidirectionnel de données (D [31:O]), des bus séparés des données sortants (Dout [31:O]) et des données entrants (D*in* [31:O]). Avec dix signaux de contrôle.

· $\overline{mrea}$ indique un cycle de processeur qui requiert un accès de la mémoire.

· seq indique que l'adresse de la mémoire sera e séquentielle (à ou probablement la même) que dans le cycle précédent.

· *lock* indique que le processeur doit forcer le bus pour assurer l'atomicité des phases de la lecture et l'écriture d'une instruction d'échange(SWAP).

· $\overline{r/w}$ indique si le processeur exécute un cycle d'écriture ou lecture

· *mas* [1:0] évaluent la taille d'accès de la mémoire, indiquant si l'accès est un quantité d'octet, demi mot ou un mot.

· *bl* [3:0] sont extérieurement contrôlé permettant l'ouverture des latches sur chacun des quatre octets sur le bus d'entrée de données.

- **L'interface du MMU** (*memory management unit*) **:**

Les signaux d'interface du MMU fournissent l'information qui est employée pour contrôler l'accès à l'espace de la mémoire. Le signal $\overline{trans}$(translation control) indique si le processeur est dans le mode d'utilisateur ($\overline{trans}$ = 0) ou dans le mode privilégié ($\overline{trans}$ = 1). Une information plus détaillée sur le mode d'exploitation est exigée, le signal $\overline{mode}$ [4:0] reflète le fond du cinq bits du CPSR (inversé), bien que la gestion de la mémoire à ce niveau soit rarement employée.

.

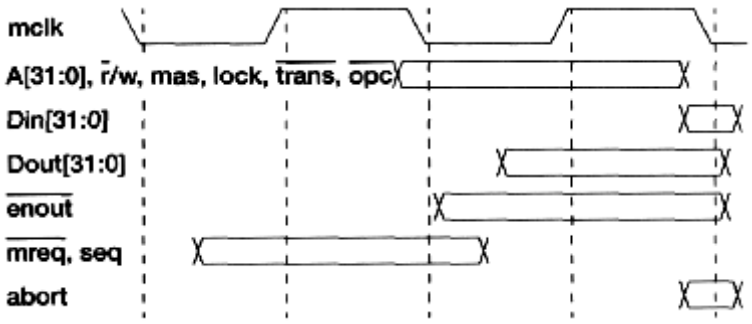


Figure II.23 Timing du MMU la mémoire du cœur ARM7TDMI

Quand un accès est rejeté, cela est signalé sur l'entrée d'arrêt *abord*. Le timing de l'arrêt *abord* doit être valide à la fin du cycle, avec les données. Cela est illustré dans figure II.23.

- **L'état :** le signal de sortie *Tbit* informe l'environnement si le processeur exécute actuellement une instructions ARM ou une instruction *thumb*.

- **Configuration :** *bigend* commute l'ordre d'octet entre *little* et *big endian* (voir l'organisation de la Mémoire au chapitre IV). Cette entrée configure comment le processeur va fonctionner.

- **Interruption :** Deux entrées d'interruption peuvent être asynchrone à l'horloge de processeur puisqu'ils passent par les latches de synchronisation avant d'entrée à la logique de contrôle du processeur. la demande de l'interruption rapide $\overline{fiq}$ u la priorité que la demande normal d'interruption $\overline{irq}$ L'entrée *isync* permet aux synchroniseurs d'interruption d'être by-passé quand l'environnement fournit les entrées qui sont déjà synchrones à *mclk* , cela enlève le retard de synchronisation de la latence s'interruption.

- **L'initialisation :** $\overline{reset}$ démarre le processeur d'un état connu, a partir de l'adresse 00000000_{16}.
- **Contrôle de bus :** Le coeur incorpore une latche transparente contrôlée par *ape*. Le coeur ARM7TDMI indique quand il exécute un cycle d'écriture en signalant sur $\overline{enout}$ O le bus de données externe est bidirectionnel, alors $\overline{enout}$ st employé pour valider *dout*[31:0] sur ceci. le signal *dbe* (*data bus enable signal*), peut être employé pour assurer que $\overline{enout}$ reste inactifs dans de telles circonstances. Les autres signaux de contrôle de bus $\overline{enin}$ $\overline{enouti}$ *be, ale, tbe , busen, highz, busdis* et *ecapclk*, réalisent autres fonctions divers.
- **Support de déboguage :** L'architecture de mise au point du processeur ARM est décrite dans le chapitre IV. Le module *EmbeddedlCE* contient des registres de *watchpoint* et *breakpoint* pour l'observation et le contrôle qui permet au code d'être arrêté pour la mise au point. Ces registres sont contrôlés par le port d'essai JTAG employant *scan chain* 2 (voir la Figure II.21).Quand on rencontre un *breakpoint* ou *watchpoint*, le processeur s'arrête et entre à l'état de mise au point
- **Interface de déboguage :** L'interface de mise au point étend la fonctionnalité fournie par le macrocellule intégré *EmbeddedlCE* en permettant à un matériel externe de valider le support de mise au point (via *dbgen*).Et fait une demande de mise au point asynchrone (sur *dbgrq*) ou une demande d'une instruction synchrone (sur *breakpf*). Le matériel externe est informé quand le coeur est dans le mode de mise au point via *dbgack*. Le signal de demande de mise au point interne est exporté sur *dbgrqi*. Des événements externes peuvent contribuer au déclenchement de *watchpoints* via *extern0* et *externl* et *rangeout0* et *rangeoutl* sont utilisés pour signaler la présence des *watchpoints*. Un tampon de transmission de communication si est vide est signalé sur *commtx* et un tampon de réception si est vide est signalé sur *commrx*. Le processeur indique si vraiment l'instruction actuelle dans l'étage d'exécutio $\overline{exec}$ exécutée sur le signal.
- **Interface de coprocesseur :** Quand une instruction de coprocesseur commence l'exécution il y a un mécanisme appelé en anglais (*hand shake)* entre le ARM et le coprocesseur pour confirmer qu'ils sont prêts à l'exécution. il emploie trois signaux :

1- $\overline{cpi}$ (de l'ARM à tous les coprocesseurs). Ce signal indique que le ARM a identifié une instruction de coprocesseur et veut l'exécuter.
2- *Cpa* (des coprocesseurs au ARM). Ce signal indique au ARM qu'il n'y a aucune présence de coprocesseur qui est capable d'exécuter l'instruction actuelle.
3- *Cpb* (des coprocesseurs au ARM). Ce signal indique au ARM que le coprocesseur est occupé et ne peut pas encore commencer à exécuter l'instruction.

Un signal complémentaire fourni aux coprocesseurs est *ope*, qui indique si un accès de mémoire doit aller chercher une instruction ou une données.

- **Alimentation :** Le coeur ARM7TDMI est conçu pour fonctionner avec une alimentation nominale de 5 volts ou voisine de 3 volts, bien que cela dépende aux capacités de la technologie de processus aussi bien que le style employé de conception de circuit dans le coeur.
- **Interface JTAG :** Les signaux de contrôle de JTAG comme auront prescrits par la norme décrite dans le chapitre IV et sont connectés au contrôleur de test via des pines spécifiques.
- **Information de TAP :** (TAP information) : Ces signaux sont employés pour supporter des nouvelles chaînes de scannage (scan chains) au système JTAG, avec les signaux d'extension (boundary scan extension) détaillés ci-dessous. Tapsm [3:0] indique l'état du contrôleur TAP; ir [3:0] donne le contenu du registre d'instruction de contrôleur TAP; screg [3:0] est l'adresse du registre de scannage actuellement choisi par le contrôleur de TAP; tckl et tck2 forment une paire d'horloge non-chevauchant pour contrôler des chaînes $\overline{tdoen}$ nage d'extension et indique quand des données périodiques sont éjectées sur tdo.
- **Extension boundary scan:** (boundary scan extension) La cellule ARM7TDMI contient contrôleur TAP de JTAG pour supporter la fonctionnalité de EmbeddedlCE et ce contrôleur est capable de supporter autres équipements de scannage sur chip qui sont eus accès par le port JTAG. Les des signaux d'interface drivebs, ecapclkbs, icapclkbs,

$\overline{hiahz}$, pclkbs, rstclkbs, sdinbs, sdoutbs, shclkbs et shclk2bs sont donc disponible pour permettre aux chemins de scannage arbitraires complémentaires d'être ajoutés au système.

5.1.2 Les caractéristiques du ARM7TDMI

Les caractéristiques du cœur sont résumées dans le Tableau 2.2.

Tableau II.2 Les caractéristiques du ARM7TDMI. [1], [13]

Processus 0.35 µm	Transistors 74209	MIPS 60
Couches du métal 3	surface du coeur 2.1 mm2	Power 87 mW
Vdd 3.3V	Horloge 0-66 MHz	MIPSAV 690

II.5.1.3 ARM7TDMI-S synthétisable

Le coeur du processeur ARM7TDMI standard est une macrocellule matérielle, c'est à dire qu'il est produit comme un morceau de disposition physique, personnalisé à la technologie de processus appropriée. L'ARM7TDMI-S est une version synthétisable de l'ARM7TDMI, produite comme un module de langage évolué qui peut être synthétisé employant n'importe quelle bibliothèque de cellule appropriée dans la technologie cible. C'est donc plus facile de changer à une nouvelle technologie de processus qu'une macrocellule matérielle. Le processus de synthèse supporte quelques variations facultatives sur la fonctionnalité du coeur de processeur. Ceux-ci incluent :

- Omission de la cellule EmbeddedlCE.
- le remplacement du multiplicateur à 64 bits de résultat avec un multiplicateur plus petit et plus simple qui supporte seulement les instructions multiplication du ARM qui produisent un résultat 32 bits.

II.5.2 ARM 8

Le coeur ARM8 a été développé au ARM ltd de 1993 à 1996 pour fournir la demande d'un coeur ARM avec une plus haute performance qu'était réalisé par ARM7 à Pipeline à 3 étages. Il a maintenant été remplacé par l'ARM9TDMI et ARM10TDMI, mais sa conception lève quelques points d'intérêt. Comme a été discuté précédemment, la performance d'un coeur de processeur peut être améliorée par :

- Augmentation du taux d'horloge. Cela exige la logique dans chaque étage de pipeline à être simplifié et, donc, le nombre d'étages de pipeline à être augmenté.
- Réduction du CPI (cycles d'horloge par instruction). Cela exige ou bien que les instructions qui occupent plus qu'une fente (slot) de pipeline dans un ARM7 sont ré-implémenter pour occuper moins des fentes (slot), ou bien ces cabines de pipeline causées par des dépendances entre des instructions sont réduites, ou une combinaison d'entre toute les deux..

II.5.2.1 Organisation interne du ARM8

Le coeur ARM8 contient une unité de pré recherche (prefetch), une unité entiere, et une mémoire à double largeur de bande (double bandwidth). L'organisation complète du cœur est illustrée dans la figure II.24.

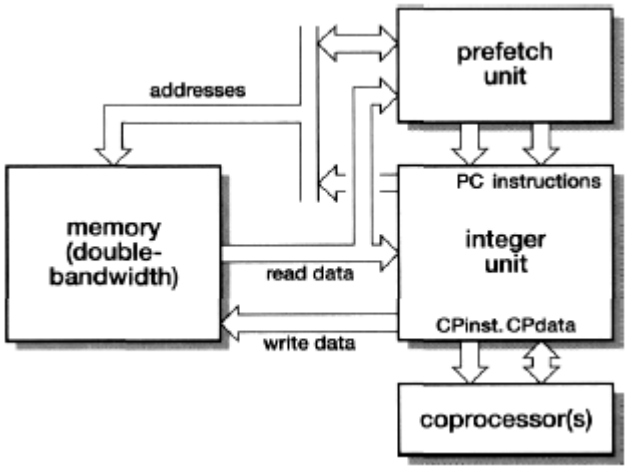


Figure II.24 Organisation du cœur ARM8

II.5.2.1. a) Mémoire à double largeur de bande

ARME8 est attaché à une mémoire unifiée (ou bien une mémoire cache ou bien une RAM sur chip) mais exploite la nature séquentielle de la plupart des accès de mémoire pour achever la largeur de bande double à partir d'une seule mémoire. Il suppose que la mémoire, à laquelle il est connecté peut produire un mot dans un cycle d'horloge et produire le mot séquentiel suivant la moitié d'un cycle plus tard concurremment avec le départ de l'accès suivant.

II.5.2.1. b) Unité de prérecherche

L'unité pré recherche est responsable de ramener les instructions de la mémoire et les maintenir temporairement (pour exploiter la mémoire à largeur de bande double). Ensuite Il les fournit à l'unité d'entier jusqu'à une instruction par un cycle d'horloge, avec sa valeur de P.

II.5.2.1. c) Organisation de l'unité entier

L'organisation de l'unité d'entier du ARME8 est illustrée dans la Figure II.25. Le flot d'instruction et des valeurs de PC associées est fourni par l'unité de prérecherche à travers l'interface montrée au sommet de la figure II.25. Le coprocesseur de contrôle de système connecte par des bus de données et activé par l'instruction de coprocesseur adéquate à gauche de la figure; l'interface de la mémoire de données est à la droite de la figure et comprend un bus d'adresse, un bus d'écriture de données et un bus de lecture de données.

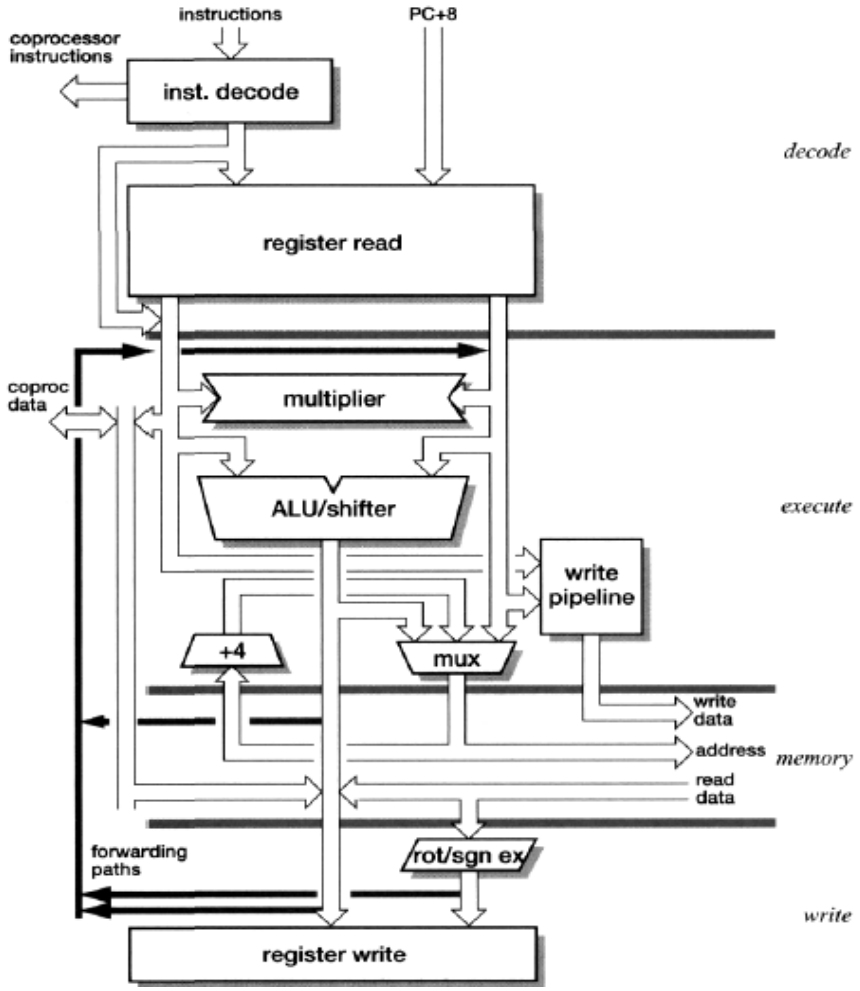


Figure II.25 Organisation de l'unité entière du ARM8

Le processeur ARM8 utilise un pipeline à 5 étages avec l'unité prérecherche s'occupant de la premier étage et l'unité entière employant les quatre étages :

1. prérecherche de l'instruction.
2. décodage de l'instruction et la lecture du registre.
3. Exécution.
4. Accès de la mémoire de données.
5. l'écriture déféré des résultats.

Le coeur du ARM8 contient 124 554 transistors et fait fonctionner aux vitesses jusqu'à 72 MHZ, a été conçu sur le processus CMOS de 0.5 µm avec trois couches de métal. [1], [13]

II.5.2 ARM9TDMI

Le coeur ARM9TDMI prend la fonctionnalité de l'ARM7TDMI jusqu'à un niveau de performance significativement plus haut. Comme l'ARM7TDMI (et différemment au ARME8) il inclut un support pour le jeu d'instruction thumb et un module EmbeddedlCE pour la vérification sur chip. L'amélioration du performance est achevée en adoptant un pipeline à 5 étages pour augmenter le taux d'horloge maximal et en employant des ports de mémoire des instruction et de données séparée pour permettre un CPI amélioré (des combien de travail un processeur fait dans un cycle d'horloge).

II.5.2.1 Fonctionnement du pipeline

L'opération du pipeline à 5 étages du ARM9TDMI est illustrée dans la Figure II.26, où il est comparé avec le pipeline à 3 étapes ARM7TDMI. La figure montre comment les fonctions de traitement principales du processeur sont redistribuées à travers les étages de pipeline complémentaires pour permettre à la fréquence d'horloge d'être doublées (approximativement) sur la même technologie de processus. La redistribution que les fonctions d'exécution (lecture de registre, décalage, ALU, écriture dans un registre) n'est pas tout ce que l'on exige pour réaliser ce taux d'horloge plus haut. Le processeur doit aussi être capable d'avoir accès à la mémoire d'instruction dans la moitié du temps que l'ARM7TDMI prend et la logique de décodage de l'instruction doit aussi être restructuré pour permettre au registre de lecture d'avoir lieu concurremment avec une partie substantielle du décodage. Comme on a vu le ARM7TDMI implémente le jeux d'instruction de pouce avec la décompression de celle-ci a un jeu d'instruction ARM ordinaire utilisation une période lâche dans le pipeline de ARM7. Le cœur ARM9TDMI contient un block matériel dédié pour l'implémentation du jeu d'instruction de pouce.

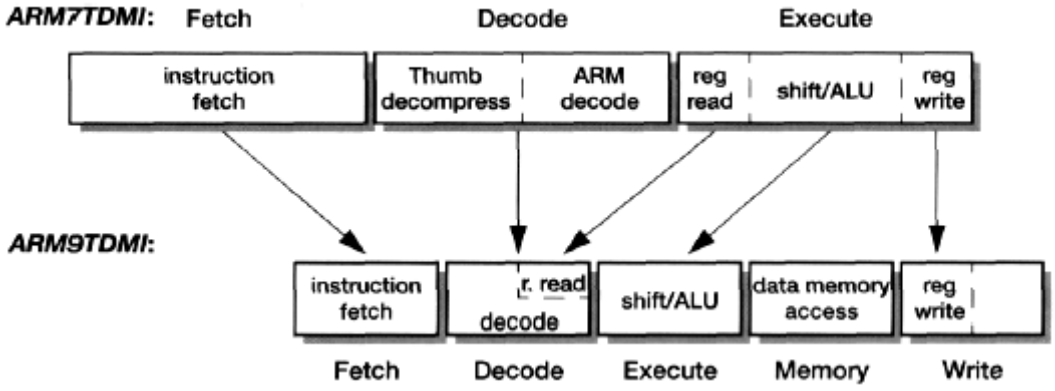


Figure II.26 comparaison du pipeline du ARM7TDMI et ARM9TDMI.

II.5.2.2 Caractéristiques physique du ARM9TDMI

Bien que le premier coeur ARM9TDMI ait été mis en oeuvre sur une technologie de 0.35 µm et 3.3 V, la conception a été portée sur des processus de 0.25 µm et 0.18 µm employant des alimentation de en bas de 1.2 V
Les caractéristiques du coeur ARM9TDM sur le processus de 0.25 µm sont résumées dans le Tableau II.3.

Tableau II.3 Les caractéristiques du ARM9TDMI. [4], [6]

Processus 0.25 urn	Transistors 111.000	MIPS 220
Couches du métal 3	surface du coeur 2.1 mm^2	Power 150mW
Vdd 2.5V	Horloge 0-200 MHz	MIPSAV 1500

II.5.2.3 ARM9E-S

L'ARM9E-S est une version synthétisable du coeur ARM9TDMI. Il exécute une version étendue du jeu d'instruction du ARM comparé avec le coeur matériel, L'ARM9E-S incluant des extensions de jeu d'instruction de traitement de signal. L'ARM9E-S est 30 % plus grand que l'ARM9TDMI sur le même processus. Il occupe 2.7 mm2 sur 0.25 µm le processus de CMOS. [1], [13]

II.5.3 ARM10TDMI

L'ARM10TDMI est le coeur du processeur ARM qu'a une haute qualité. De même que l'ARM9TDMI livre approximativement deux fois la performance de l'ARM7TDMI sur le même processus, l'ARM10TDMI est placée pour faire fonctionner à deux fois performance de l'ARM9TDMI. Il est destiné de livrer 400 dhrystone 2.1 MIPS à 300 MHZ sur la technologie CMOS de 0.25 µm, Pour réaliser ce niveau performance, a partir du ARM9TDMI, deux approches ont été combinée: [4], [6]

1. Le taux d'horloge maximal a été augmenté.
2. Le CPI (le nombre moyen d'horloge par instruction) a été réduit.

L'approche d'ARM10TDMI est de conserver un pipeline très semblable à l'ARM9TDMI, mais supporter un taux d'horloge plus haut en optimisant chaque étage d'une façon particulière (Figure II.27) :
Les étages de la recherche et de la mémoire sont efficacement augmentés d'un cycle d'horloge à un et demi en fournissant l'adresse pour le cycle suivant. Pour achever ça dans l'étage de la mémoire, les adresses de la mémoire sont calculées dans un additionneur séparé qui peut produire son résultat plus rapidement que l'ALU principal.
L'étage de l'exécution emploie une combinaison des techniques des circuits améliorées et restructurant pour réduire son chemin critique. Par exemple, le multiplicateur n'utilise pas la ALU principal pour résoudre sa somme partielle et des bornes de produit; au lieu de cela il a son propre additionneur dans l'étage de mémoire (les multiplications n'ont jamais accès à la mémoire, donc cette étage est libre.)

L'étage de décodage de l'instruction est la seule partie de la logique de processeur qui ne pouvait pas être simplifiée suffisamment pour supporter un taux d'horloge plus haut, donc ici un étage de pipeline complémentaire a été insérée.
Le résultat est un pipeline à 6 étages qui peut fonctionner plus rapidement que le pipeline à 5 étages du ARM9TDMI, mais exige que ses mémoires de travail soient un peu plus rapidement que les mémoires ARM9TDMI'S.

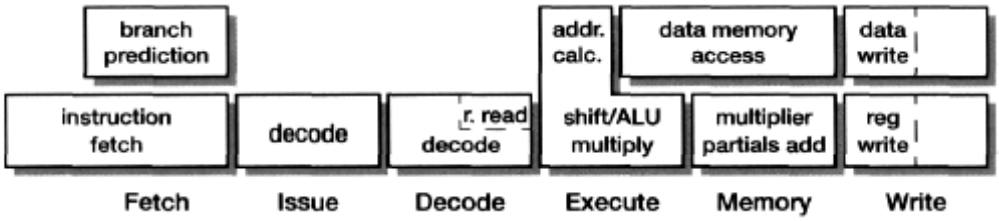


Figure II.27 Pipeline du ARM10TDMI

II.6 Conclusion

Tous les premiers coeurs de processeur ARM, jusqu'à le cœur ARM7TDMI, étaient basés sur un simple pipeline à 3 étapes (décodage, recherche, exécution). De premier ARM1, développé a Acorn computers au début des années 1980, passant par les coeurs ARM7TDMI qui sont utilisé dans la plupart de combinés de téléphone portable d'aujourd'hui, les principes de base d'opération ont à peine changé. Les actions de développement effectué en première décade du ARM se sont concentrées sur les aspects suivants de la conception :

- L'amélioration du performance par l'optimisation de chemin critique et font rétrécir le processus de conception;
- Des applications à basse puissance par la logique de CMOS statique, tension d'alimentation réduite et compression de code (le jeu d'instruction de Pouce);
- supportent pour le développement de système, des équipements additionnels sur chip de mise au point (vérification), des bus et des outils de logiciel.

L'ARM7TDMI représente le pinacle de ce processus de développement et son succès commercial démontre la viabilité du pipeline original très simple à 3 étages dans un monde dominé par des PC avec toujours de plus en plus complexe superscalaire, superpipeline, très performant des microprocesseurs. La deuxième décade de développement des ARM a vu une diversification prudente de l'organisation des ARM demandant un niveau plus haut de performance:

Le premier pas à un pipeline à 5 étages rapporte un doublement du performance (tous les autres facteurs étant égaux) aux dépens de quelque logique d'expédition dans le coeur et ou bien mémoire de largeur de bande double (comme dans le ARME8) ou bien des mémoires à des instructions et données séparées (comme dans l'ARM9TDMI).

Le doublement suivant du performance, réalisée dans le ARM1OTDMI, est plutôt le plus durement gagné. Le pipeline à 6 étages est tout à fait semblable au pipeline à 5 étages employé auparavant, mais les fentes de temps allouées à l'accès de la mémoire ont été étendues pour permettre aux mémoires de soutenir des taux d'horloge plus hauts sans brûlure excessif de la puissance. Le coeur de processeur incorpore aussi plus de découplage dans l'unité pré-recherche pour permettre des branches à être prévues et enlevé du flot d'instruction et dans l'interface de mémoire de données pour permettre au processeur de continuer l'exécution quand un accès de données prend quelque temps pour réaliser (par exemple, en raison d'une cache échoué (cache miss)).

L'amélioration de la performance est réalisée par une combinaison de taux d'horloge augmenté et de CPI réduit - le nombre moyen d'horloge par instruction. Le taux d'horloge accru exigera d'habitude un pipeline plus profond qui aura tendance à augmenter le CPI, donc on exige que des mesures de redressement récupèrent la perte de CPI et l'améliorer ensuite plus loin.

Jusqu'à présent, tous les processeurs ARM ont été basés sur des organisations qui traitent au maximum une instruction par un cycle d'horloge et toujours dans l'ordre de programme. L'ARM10TDMI et le processeur EMULET3 sont capable de tenir des instructions coulant pendant un accès de données lent et tous les deux de ces processeurs incluent aussi la logique de prédiction de branchement pour réduire le progressif d'occuper à nouveau leurs pipelines sur des instructions de branchement. AMULET3 supprime l'installation des instructions de branchement prévues, mais les exécute toujours ; ARM10TDMI va chercher des instructions de branchement, mais supprime leur exécution. Mais par les standards de PC d'aujourd'hui de haute qualité et des processeurs de poste de travail, ceux-ci sont des machines toujours très simples. Cette simplicité a des bénéfices directs dans des applications de système sur chip dans lesquelles des processeurs simples exigent des transistors de moins que des processeurs complexes et occupent donc moins espace de silicium et consomme moins de puissance

CHAPITRE III

L'ARCHITECTURE DU JEU D'INSTRUCTION DU PROCESSEUR ARM

III.1 INTRODUCTION

La programmation du processeur ARM est très facile au niveau d'assembleur, pour la plupart des applications il soit plus approprié au programme en langage de haut niveau comme le C ou le C ++. La programmation en assembleur exige que le programmeur pense au niveau de l'instruction de machine. Une instruction de processeur ARM a un longueur de 32 bits, ainsi il y a autour de 4 milliards d'instructions machine différentes binaires. Heureusement il y a la structure considérable dans l'espace d'instruction, donc le programmeur ne doit pas être familier avec tous les 4 milliards code binaires. Cependant, il y a une quantité considérable de détail à être obtenu directement dans chaque instruction. L'assembleur est un programme informatique qui manipule la plupart de ce détail pour le programmeur.

Dans ce chapitre nous allons étudier le jeu d'instruction du processeur ARM, pour voir presque la gamme complète des instructions qui sont disponibles dans le jeu d'instruction du processeur ARM.

Quelques coeurs du ARM exécutent encore une forme compressée du jeu d'instruction où un sous-ensemble du plein jeu d'instruction du ARM qui est codé sur 16 bits. Ces instructions sont appelés jeu d'instruction thumb et sont discutées en bref à la fin du Chapitre.

.

III.2 GENERALITES

III.2.1 Le modèle du programmeur du ARM

Le jeu d'instruction d'un processeur définit les opérations que le programmeur peut utiliser pour changer l'état du système incorporant le processeur. Cet état comprend évidemment les données dans les registres visibles du processeur et la mémoire du système. Chaque instruction peut être vue comme l'exécution d'une transformation désignée de l'état avant que l'instruction ne soit exécutée à l'état après qu'il a achevé. Notons que bien qu'un processeur ait typiquement beaucoup des registres invisibles inclus dans l'exécution d'une instruction, les valeurs de ces registres ne sont pas significatives avant et après l'exécution de l'instruction ; seulement les valeurs dans les registres visibles ont quelques significations. La figure III.1 illustre les registres visibles dans un processeur ARM.
Lorsque on écrit un programme de niveau d'utilisateur, seulement les 15 registres généraux de 32 bits (r0 à r14), le compteur du programme (r15) et le registre d'état de programme actuel (CPSR) doit être considéré. Les autres registres sont employés seulement pour la programmation de niveau de système et pour le traitement d'exceptions (par exemple, interruption).

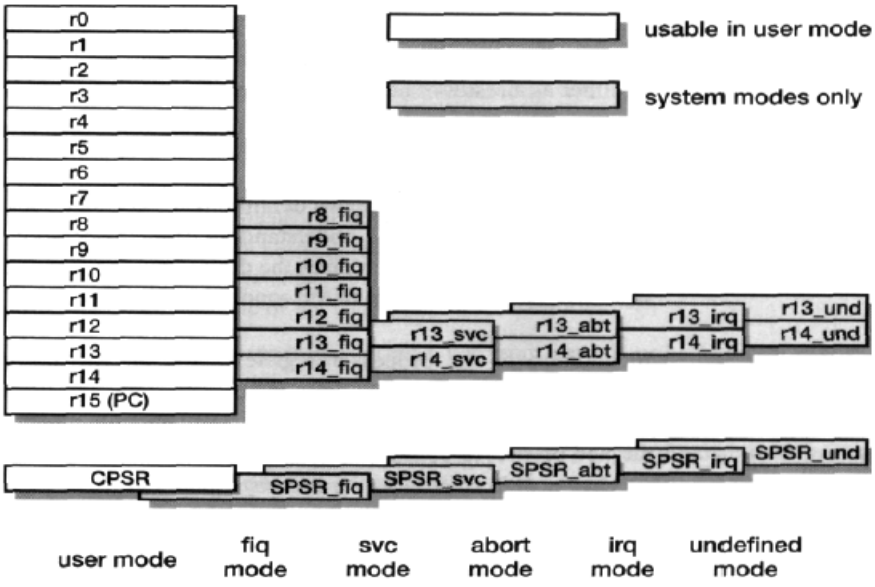


Figure III.1 Les registres visibles du ARM

III.2.2 Mode privilégié

La plupart des programmes fonctionnent dans le mode d'utilisateur comme décrit précédemment. Cependant, le processeur ARM a d'autres modes privilégiés d'exploitation qui sont employés pour réaliser des exceptions et des appels de superviseur (*supervisor calls*) (qui est parfois appelé une interruption logiciel). Où le jeu de registre n'est pas les registres d'utilisateur, les registres appropriés gris montrés dans la Figure III.1 remplacent les registres d'utilisateur correspondants et le SPSR (*Saved Program Status Register*) devient aussi accessible.

Quelques processeurs ARM ne supportent pas tous les modes d'exploitation. Les modes privilégiés peuvent seulement être entrés par des mécanismes contrôlés; Avec la protection de mémoire appropriée ils permettent à un système entièrement protégé d'exploitation d'être construits. La plupart des ARMs sont employés dans des systèmes incorporés où une telle protection est inconvenante, mais les modes privilégiés peuvent toujours être employés pour donner un niveau plus faible de protection qui est utile pour la prise au piège du logiciel dévoyé.
Le mode superviseur est un programme qui fonctionne à un niveau privilégié, ce qui signifie qu'il peut faire des choses qu'un programme de niveau d'utilisateur ne peut pas le faire directement.
Le processeur ARM supporte un mode de superviseur protégé. Le mécanisme de protection assure que le code d'utilisateur ne peut gagner des privilèges de superviseur sans contrôles appropriés, étant effectués pour assurer que le code ne risque pas d'opérations illégales. Le résultat de cela pour le programmeur de niveau d'utilisateur est que les fonctions au niveau système peuvent seulement avoir accès que par des appels spécifiés de superviseur. Ces fonctions incluent généralement n'importe quels accès aux registres de périphérique de matériel et aux opérations largement employées comme l'entrée et la sortie de caractère. Les programmeurs de niveau d'utilisateur sont principalement concernés par l'invention d'algorithmes pour fonctionner sur les données 'en possession' selon leurs programmes et comptent sur le système d'exploitation pour manipuler toutes les transactions avec le monde extérieur de leurs programmes. Chaque mode privilégié (sauf le mode de système) y a associé un Registre de Statut de Programme Sauvé, ou SPSR. Ce registre est utilisé pour maintenir l'état du CPSR (le Registre de Statut de Programme Actuel) quand le mode privilégié est entré, afin que l'état d'utilisateur puisse être entièrement rétabli quand le processus d'utilisateur est repris.

III.2.3 Le CPSR (*Current Program Status Register*)

Le CPSR est utilisé dans des programmes de niveau d'utilisateur pour stocker les bits de code de condition. Ces bits sont utilisés, par exemple, pour enregistrer le résultat d'une opération de comparaison et contrôler si vraiment un branchement conditionnel est pris. Le programmeur de niveau d'utilisateur n'a pas besoin d'être concerné par comment ce registre est configuré, le registre est illustré dans la figure III.2. Les bits du poids faible du registre contrôlent le mode de processeur, les bits 'T', (pour sélectionner le jeu d'instruction 'thumb') et 'I F' (pour valider l'interruption) sont protégé du changement selon le programme de niveau d'utilisateur.

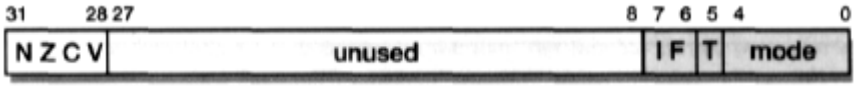


Figure III.2 Format du CPSR du ARM

Les drapeaux de code de condition sont dans les quatre bits du pois fort du registre et ont les significations suivantes :

- N : Négatif; la dernière opération ALU qui a changé les drapeaux a produit un résultat négatif (N=1).

- Z : Zéro; la dernière opération de l'ALU qui a changé les drapeaux a produit un résultat zéro (chaque bit du résultat de 32 bits était zéro).
- C : Carry; la dernière opération ALU qui a changé les drapeaux a produit un carry, ou bien suite à une opération arithmétique dans l'ALU ou bien du baril à décalage.
- V : débordement; la dernière opération arithmétique de ALU qui a changé les drapeaux produits un débordement dans le bit de signe.

Le mode courant d'exploitation utilisé est défini par les cinq bits du pois faible du CPSR comme il est illustré dans la Figure III.2. L'interprétation de ces bits est résumée dans le tableau III.1.

Tableau III.1 Modes d'exploitation et utilisation de registre. [13]

CPSR[4:0]	Mode	Utilisation	Registres
10000	utilisateur	Code utilisateur normal	utilisateur
10001	FIQ	Traitement rapide d'interruption	_fiq
10010	IRQ	Traitement d'interruption standard	_irq
10011	SVC	Traitement d'interruption logiciel SWIs	_svc
10111	Abort	Traitement de fautes de mémoire	_abt
11011	Undef	Traitement de pièges d'instruction non définis	_und
11111	System	Réaliser des tâches privilégiées de système d'exploitation	user

III.2.4 Les types de donnés

Les processeurs ARM supportent six types de données :

- Octets de 8 bits signés et non signés.
- Demi mots de 16 bits signées et non signées; sont alignés sur des lignes à 2 octets.
- Mots de 32 bits signés et non signés; sont alignés sur des lignes à 4 octets.

Les instructions du ARM sont tous des mots de 32 bits et doivent être des mots alignées. Les instructions thumb sont des demi mots et doivent être alignées sur des lignes à 2 octets. Intérieurement toutes les opérations du ARM sont sur des opérandes de 32 bits; les types de données plus courts sont seulement soutenus par des instructions de transfert de données. Quand un octet est chargé de la mémoire c'est zéro - ou prolongé de signe à 32 bits et a ensuite traité comme une valeur 32 bits pour le traitement interne. Les coprocesseurs du ARM peuvent supporter d'autres types de données et en particulier il y a un jeu défini de types pour représenter des valeurs de virgule flottante. Il n'y a aucun support explicite pour ces types dans le coeur du ARM, cependant et en absence d'un coprocesseur de virgule flottante ces types sont interprétés par le logiciel qui emploie les types standard mentionnés ci-dessus.

III.2.5 Organisation de la mémoire

En plus de l'état de registre du processeur, un système à processeur ARM a l'état de la mémoire. La mémoire peut être vue comme un tableau linéaire d'octets numérotés du zéro jusqu'à 2^{32}-l. Les données peuvent être des octets de 8 bits, des demi mots de16 bits ou des mots de 32 bits. Les mots sont toujours alignés sur des lignes de 4 octets et les demi mots sont alignés sur des mêmes lignes d'octet. L'organisation de la mémoire est illustrée dans la Figure III.3. Cela montre un petit secteur de mémoire où chaque emplacement d'octet a un numéro unique. Un octet peut occuper chacun de ces emplacements et on montre quelques exemples dans la figureIII.3. Une taille d'une donnée de mot doit occuper un groupe d'emplacements de quatre octets commençant à une adresse d'octet qui est un multiple de quatre et de nouveau la figure contient un couple d'exemples. Les demi mots occupent emplacements de deux octets commençant à une même adresse d'octet.

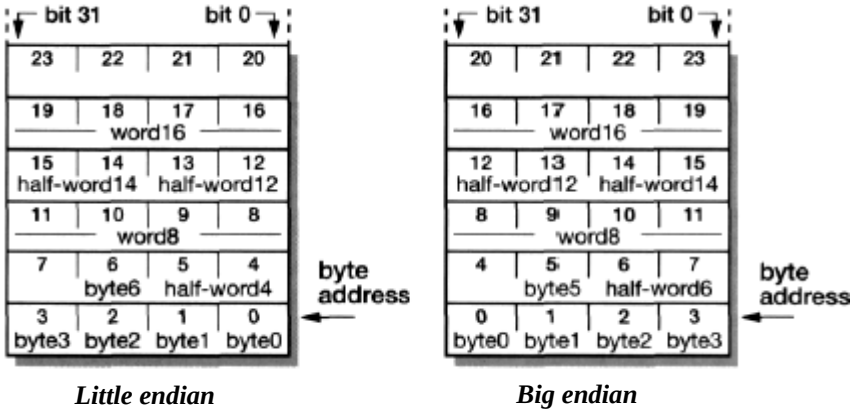


Figure III.3 Les deux organisations de la mémoire

Il y a deux façons de stocker les mots dans une mémoire adressée avec des octet, selon si l'octet du pois faible est stocké à une adresse inférieure ou plus supérieure que l'octet du pois fort suivant. Puisqu'il n'y a pas une raison du choix d'une approche sur l'autre. Les deux arrangements sont illustrés dans la Figure III.3, qui montre comment un groupement de types de données serait stocké sous les deux arrangements. Le demi mot (half-word 12) est trouvé à l'adresse 12, et cetera.) Les terminologies' ***big endian***' et '***little endian***' sont employées pour dénoter les deux approches.

III.2.6 Architecture du chargement rangement (*load-store architecture*)

En commun avec la plupart des processeurs RISC, ARM utilise une architecture du chargement rangement. Cela signifie que le jeu d'instruction traitera seulement (addition, soustraction, et cetera) les valeurs qui sont dans les registres (ou indiquées directement dans l'instruction lui-même) et placeront toujours les résultats d'une telle opération dans un registre. Les seules opérations qui s'appliquent à l'état de la mémoire sont celles qui copient des valeurs de la mémoire dans les registres (Instructions de chargement) ou copient des valeurs de registre dans la mémoire (Instructions de rangements). Les processeurs CISC permettent typiquement à une valeur de la mémoire d'être ajouté à une valeur dans un registre et permettent parfois à une valeur dans un registre d'être ajouté à une valeur dans la mémoire.

Le processeur ARM ne supporte pas telles opérations 'de mémoire-mémoire'. Donc toutes les instructions du ARM tombent dans un des trois catégories suivantes :

1. Instructions de traitement de données. Ceci utilise et traite seulement des valeurs de registre. Par exemple, une instruction peut ajouter deux registres et placer le résultat dans un registre.

2. Les instructions de transfert de données. Ceci copie des valeurs de la mémoire dans des registres (une instruction de chargement) ou copie des valeurs de registre dans la mémoire (des instructions de rangement).

3. Instructions de flux de contrôle. L'exécution d'instruction normale emploie des instructions stockées aux adresses de mémoire consécutives. Les instructions de flux de contrôle causent une commutation à une adresse différente, de manière permanente (des instructions de branchement) ou bien maintenir d'une adresse de retour pour reprendre l'ordre original (instructions de branchement et de liaison) ou la prise au piège dans le code de système (des appels de superviseur).

III.3 LE JEU D'INSTRUCTION DU PROCESSEUR ARM

III.3.1 Exceptions

Les exceptions sont d'habitude employées pour traiter les événements inattendus qui surgissent pendant l'exécution d'un programme, comme une interruption ou des fautes de la mémoire. Dans l'architecture du ARM le terme est aussi employé pour couvrir une interruption logiciel et des pièges d'instruction non définis (qui ne sont pas vraiment 'inattendu') et la fonction remise (reset) de système qui surgit logiquement avant plutôt que pendant l'exécution d'un programme. Ces événements sont tous groupés dans le titre 'd'exception' parce qu'ils tous emploient le même mécanisme de base dans le processeur. Les exceptions du ARM peuvent être considérées dans trois groupes :

1. Exceptions produites comme l'effet direct d'exécuter une instruction. Interruption logicielle, des instructions non définies (incluant des instructions de coprocesseur où le coprocesseur demandé est absent) et des arrêts de prérecherche (les instructions qui sont invalide dus à une faute de la mémoire arrivant pendant la recherche).

2. Exceptions produites comme un effet secondaire d'une instruction. Les arrêts de données (une faute de mémoire pendant l'accès de chargement ou rangement de données).

3. Exceptions produites extérieurement, sans rapport au flux d'instruction. Remis (reset), interruption normale IRQ et interruption rapide FIQ.

Le processeur va entrer aux modes privilégiés d'exploitation qui sont employés pour manipuler les exceptions appropriées. Également quelques nouveaux registres sont

automatiquement activés (figure III.1). Par exemple un événement externe (comme mouvement de la souris) arrive va produire une interruption rapide (sur le pin FIQ).
La même vue de r0-r7 qu'auparavant, aussi une nouvelle vue du jeu de r8-r14 et de plus, le SPSR maintient la valeur de CPSR. En échangeant à quelques registres, il le fait plus facile pour le programmeur préserver l'état du processeur. Par exemple, pendant le mode FIQ, r8-r14 peut être employé librement. En rendant en arrière le mode d'utilisateur, les valeurs originales de r8-r14 seront automatiquement rétablies.

III.3.1.a) Que passe-t-il lorsque une exception survient ?

- Le ARM termine l'instruction actuelle comme de mieux qu'il peut.
- Il part de la séquence de l'instruction actuelle pour manipuler l'exception en exécutant les étapes suivantes :
    1. Il change le mode d'exploitation correspondant à l'exception spécifique.
    2. Il maintien l'adresse de l'instruction (PC) qui suit l'instruction d'exception dans le r14 correspondant au nouveau mode. Par exemple, si FIQ arrive, la valeur de PC est stockée dans R14 (FIQ).
    3. Il maintien l'ancienne valeur de CPSR dans le SPSR du nouveau mode.
    4. Il met hors de service les exceptions de priorité inférieure (pour être considéré Plus tard).
    5. Il force le PC à une nouvelle valeur correspondant à l'exception. L'adresse de vecteur approprié est mentionnée dans le tableau III.2

Tableau III.2 Adresses de vecteur d'exception. [6]

Exception	Mode	Adresses de vecteur
Remise en place	SVC	0x00000000
Instruction indéfinie	UND	0x00000004
Interruption logiciel	SVC	0x00000008
Arrêt de précherche (faut recherche de la mémoire)	Abort	0x0000000C
Arrêt de donnée (faut d'accès de donnée à la mémoire)	Abort	0x000000010
IRQ (interruption normal)	IRQ	0x000000018
FIQ (interruption rapide)	FIQ	0x00000001C

III.3.1.b) Retour d'exception

- Une fois que l'exception a été traitée (par l'instructeur d'exception), la tâche d'utilisateur est reprise.
- Le programme d'instructeur (ou la Routine de Service d'interruption) doit rétablir l'état d'utilisateur exactement comme il était avant que l'exception ne soit arrivée :

    1. N'importe quels registres d'utilisateur modifiés doivent être rétablis de la pile de l'instructeur
    2. Le CPSR doit être rétabli de SPSR approprié
    3. Le PC doit être changé en arrière à l'adresse d'instruction dans le flot d'instruction d'utilisateur.

- Pas 1 et 3 sont fait par l'utilisateur, le pas 2 par le processeur
- Les registres rétablissement de la pile seraient le même comme dans le cas de sous-programmes
- le rétablissement de la valeur de PC est plus compliqué. La façon exacte de le faire dépend sur lequel l'exception sera retournée.

III.3.1.c) Priorités d'exception

Puisque les exceptions peuvent surgir en même temps, un ordre de priorité doit être clairement défini. Pour le processeur du ARM c'est :

- Remis en place (priorité la plus haute)
- Arrêt de données (c'est-à-dire. La faute d'écriture ou lecture de données de la mémoire)
- La demande d'interruption rapide (FIQ)
- La demande d'interruption normale (IRQ)
- Arrêt de prérecherche
- Interruption logicielle (SWI), instruction non définie

III.3.2 Exécution conditionnelle

Une particularité commune du jeu d'instruction du ARM est que chaque instruction (à l'exception des certaines instructions v5T) est conditionnellement exécutée. Des branchements conditionnels sont une particularité standard de la plupart des jeux d'instruction, mais le ARM étend l'exécution conditionnelle à toutes ses instructions, incluant des appels de superviseur et des instructions de coprocesseur. Le champ de condition occupe les quatre bits de sommet du champ d'instruction à 32 bits :

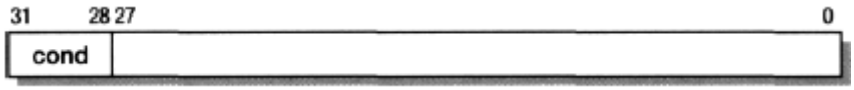


Figure III.4 Le champ du code de condition du ARM

Chacune de ces 16 valeurs du champ de condition cause que l'instruction est exécutée ou sauté selon les valeurs des drapeaux N, Z, C et V dans le CPSR. On donne les conditions dans le tableau III.3. Chaque mnémonique d'instruction du ARM peut être étendue en ajoutant les deux lettres définies dans ce tableau, bien que ' la condition toujours 'Always' (AL) puisse être négligée puisque c'est la condition de défaut qui est assumée si aucune autre condition n'est spécifiée. La condition jamais 'Never' (NV) ne doit pas être employée, La raison d'éviter la condition 'Never' est que ARMltd a indiqué qu'ils peuvent employer ce secteur de l'espace d'instruction pour d'autres buts dans l'avenir.

Tableau : III.3 les codes de condition du ARM. [13]

Opcode [31 :28]	**extention de mnémonique**	**explication**	**drapeaux d'état pour l'exécution**
0000	EQ	Equal/Equals zero	Z set
0001	NE	Not equal	Z clear
0010	CS/HS	Carry set/ unsigned higher or same	C set
0011	CC/LO	Carry clear / unsigned lower	C clear
0100	MI	Minus/ negative	N set
0101	PL	Plus/ positive or zero	N clear
0110	VS	Overflow	V set
0111	VC	No overflows	V clear
1000	HI	unsigned higher	C set and Z clear
1001	LS	unsigned lower or same	C clear and Z set
1010	GE	signed greater than or equal	N equal V
1011	LT	signed less than	N is not equal to V
1100	GT	signed greater than	Z clear and N equals V
1101	LE	signed less than or equal	Z set or N is not equal to V
1110	AL	Always	any
1111	NV	never (do not use)	none

III.3.3 Instructions de traitement de données

III.3.3.1 Formats d'assembleur

Les instructions de traitement de données du ARM permettent au programmeur d'exécuter des opérations arithmétiques et logiques sur des données stockées dans les registres. Toutes les autres instructions déplacent juste des données autour et contrôlent l'ordre d'exécution de programme, donc les instructions de traitement de données sont les seules instructions qui modifient les valeurs de données dans les registres du ARM.
Voici quelques règles qui s'appliquent aux instructions de traitement de données du ARM :

- Tous les opérandes sont de 32 bits de largeur et arrivent des registres ou sont spécifié avec des littéraux dans l'instruction lui-même.
- Le résultat, s'il y a un, est de 32 bits de largeur et est placé dans un registre. (Il y a une exception pour un résultat à 64 bits; voir plus loin.).
- Chacun des registres d'opérande et le registre de résultat sont indépendamment spécifié dans l'instruction. C'est-à-dire le ARM utilise un format de 3 adresses pour ces instructions.

III.3.3.1.a) Opérandes d'un registre simples

Les différentes instructions disponibles dans cette forme sont citées ci-dessous selon leurs classes :

- Opération arithmétique :

```
ADD   r0,  r1,  r2          ; r0 := r1   + r2              (addition simple)
ADC   r0,  r1,  r2          ; r0 := r1   + r2  + C         ( addition avec carry)
SUB   r0,  r1,  r2          ; r0 := r1   -  r2             (soustraction)
SBC   r0,  r1,  r2          ; r0 := r1   -  r2  + C - 1    (soustraction avec carry)
```

```
RSB     r0,   r1,   r2          ;  r0 := r2    -  r1            (soustraction inverse)
RSB     r0,   r1,   r2          ;  r0 :=  r2   -  r1   + C - 1  (soustraction inverse avec carry)
```

- Opérations logiques bit par bit (*bit wise logical operations*) :

Ces instructions exécutent l'opération logique Booléenne spécifié sur chaque paire de bit des opérandes d'entrée, si dans le premier cas r0 [i] : = r1 [i] and r2 [i] pour chaque valeur de i de 0 à 31 y compris, où r0 [i] est le i^{eme} bit de r0.

```
AND     r0,   r1,   r2          ; r0 := r1    and  r2
ORR     r0,   r1,   r2          ; r0 := r1    or  r2
EOR     r0,   r1,   r2          ; r0 := r1    xor  r2
BIC     r0,   r1,   r2          ; r0 := r1    and not  r2
```

- Opération de mouvement de registre :

```
MOV     r0,   r2          ; r0 :=  r2          (Déplacer le deuxième opérande)
MVN     r0,   r2          ; r0 :=  not r2       (move negated)
```

- Opération de comparaison:

Ces instructions ne produisent pas de résultat (qui est donc omis du format d'assembleur) mais mettre en jeu juste les bits de code de condition (N, Z, C et V) dans le CPSR selon l'opération choisie.

```
CMP     r1,   r2          ;   (mis en place le jeu de code de condition pour    r1  -  r2 )
CMN     r1,   r2          ;   (mis en place le code de condition pour    r1  +  r2 )
TST     r1,   r2          ;   (mis en place le code de condition pour    r1  and  r2 )
TEQ     r1,   r2          ;   (mis en place le code de condition pour    r1  xor  r2 )
```

III.3.3.1.b) Opérandes immédiate

Si, au lieu de l'addition de deux registres, nous voulons simplement ajouter un constant à un registre nous pouvons remplacer le deuxième opérande source d'une valeur immédiate, qui est un littéral constant, a précédé par ' #':

```
ADD     r3,  r3,   #1       ; r3 :=  r3 + 1
AND     r8,  r7,   #&ff     ; r8 :=  r7[7:0]
```

Le deuxième exemple montre que la valeur immédiate peut être spécifiée en hexadécimal (base 16) en mettant '&' après le ' #'.

III.3.3.1.c) Opérandes d'un registre à décalage

Une troisième façon de spécifier une opération de données est semblable a la première, mais permet au deuxième opérande de registre d'être soumis à une opération de décalage avant qu'il ne soit combiné avec le premier opérande. Par exemple :

```
ADD   r3,  r2,  r1,  LSL  # 3                          ; r3 := r2 + 8 x  r1
```

Notons que c'est toujours une seule instruction du ARM, exécutée dans un seul cycle d'horloge. La plupart des processeurs réalisent les opérations de décalage comme des instructions séparées, mais le processeur ARM les combine avec une opération générale ALU dans une seule instruction. Ici 'LSL' indique ' le décalage logique à gauche par un nombre spécifié de bit, qui dans cet exemple est 3. Comme précédemment, ' # ' indique la quantité immédiate. Il est aussi possible d'utiliser une valeur de registre pour spécifier le nombre de bit le deuxième opérande qui doit être décalé avec :

```
ADD   r5,  r5,  r3,  LSL r2              ; r5 : = r5 + r3 x 2 r 2
```

Les opérations possibles de décalage sont illustrées dans la figure III.5.

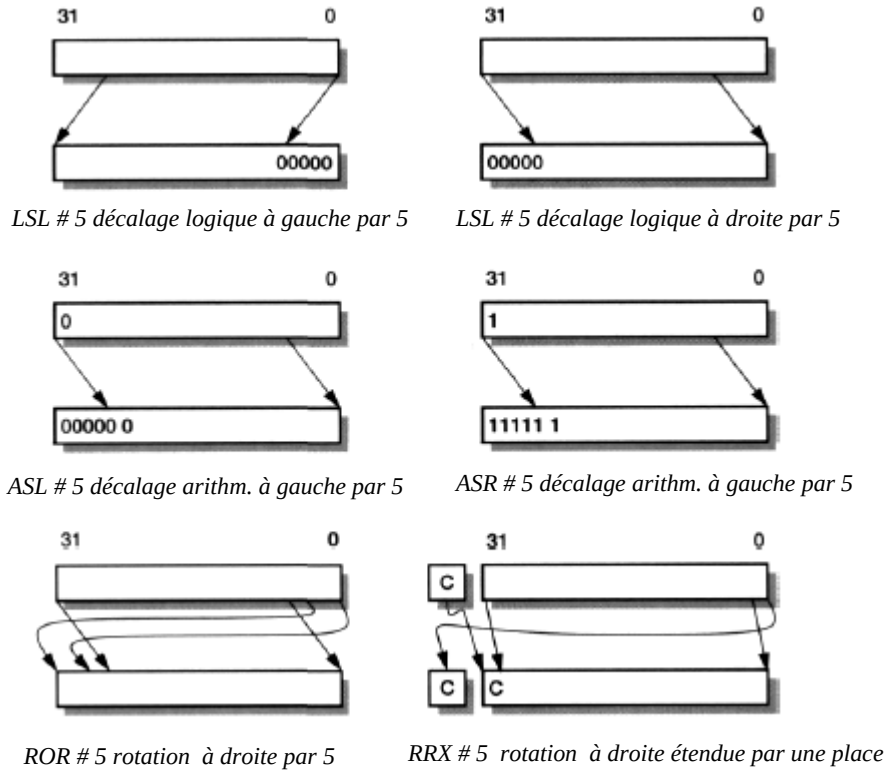


Figure III.5 opérations de décalage du ARM

III.3.3.2 Formats binaire

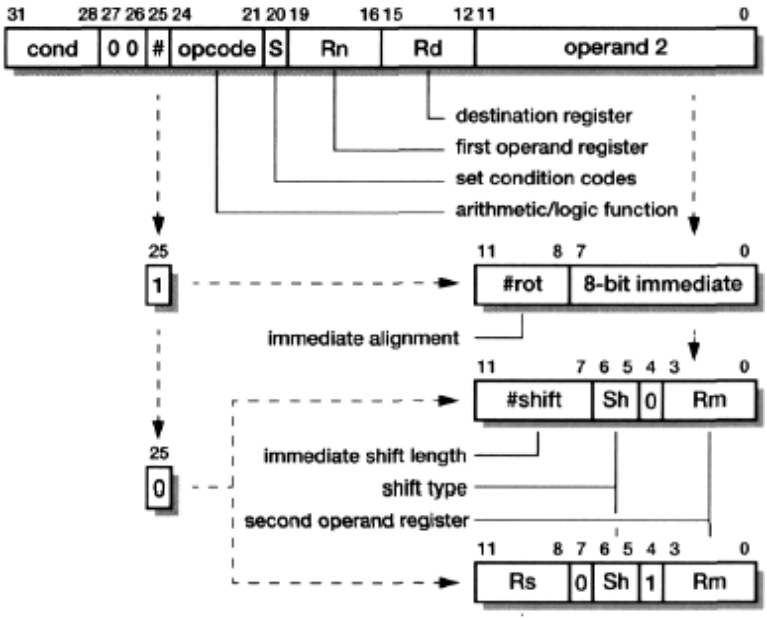


Figure III. 6 Codage binaire d'instruction de traitement de données.

- **description**

Ces types d'instructions utilisent 3 format d'adresse : le premier opérande toujours est un registre, le deuxième opérande peut être un registre, un registre à décaler ou une valeur immédiate, le résultat toujours est un registre. Pour le deuxième opérande de registre, le décalage peut être logique, arithmétique ou rotation, C'est spécifié dans « le type de décalage ». Les opérations qui peuvent être réalisés sont inscrites dans le tableau III.4. Quand l'instruction n'exige pas tous les opérandes disponibles (par exemple MOV ignore Rn et CMP ignore Rd) le champ de registre inutilisé doit être mis à zéro. L'assembleur fait ça automatiquement. Ces instructions permettent le contrôle direct si vraiment les codes de condition du processeur sont affectés par leur exécution par le bit S (le bit 20). Quand l'on désactivé, les codes de condition seront inchangés; et quand l'on activé (set) (et Rd n'est pas r15; voir ci-dessous) :

- N le drapeau – mis en place si le résultat est négatif (N égale le bit 31 de résultat)
- Z le drapeau - mis en place si le résultat est zéro
- C le drapeau - mis en place s'il y a un carry d'ALU pendant des opérations arithmétiques, ou mis par le baril.
- Le drapeau V est préservé dans des opérations non-arithmétiques. Il est survenu dans l'opération arithmétique s'il y a un débordement du bit 30 au bit 31 et remis à zéro s'il n'y a pas un débordement. Il est significatif seulement quand les opérandes sont vus comme le deuxième complément des valeurs signées.

Tableau III.4 instruction de traitement de données. [13]

Opcode [24 :21]	Mnémonique	Explication	Effet
0000	AND	ET logique bit par bit	Rd :=Rn AND Op2
0001	EOR	OU exclusive logique bit par bit	Rd :=Rn EOR Op2
0010	SUB	Soustraction	Rd :=Rn – Op2
0011	RSB	Soustraction inversée	Rd :=Op2 – Rn
0100	ADD	Addition	Rd :=Rn + Op2
0101	ADC	Addition avec carry	Rd:= Rn + Op2 + C
0110	SBC	Soustraction avec carry	Rd:= Rn –Op2 +C – 1
0111	RSC	Reverse subtract with carry	Rd:= Op2 – Rn + C – 1
1000	TST	Test	Sec on Rn AND Op2
1001	TEQ	Test equivalence	Sec on Rn EOR Op2
1010	CMP	Compare	Sec On Rn – Op2
1011	CMN	Compare negated	Sec on Rn + Op2
1100	ORR	Logical bit-wise OR	Rd:= Rn OR Op2
1101	MOV	Move	Rd:= Op2
1110	BIC	Bit clear	Rd:= Rn AND NOT Op2
1111	MVN	Move negated	Rd:= NOT Op2

III.3.3.3 Multiplications par constant

Ces instructions peuvent être employées, pour multiplier un registre par un petit constant plus efficacement que d'être réalisé en employant les instructions de multiplication spéciale du processeur ARM décrites dans la section suivante. Quand on multipliant par une valeur constante, il est possible de remplacer la multiplication par une séquence fixée d'addition et de soustraction ayons le même effet.
Par exemple, multiplications par 5 pourraient être réalisé employant une simple instruction :

```
ADD  Rd,  Rm,  Rm,  LSL #2     ; Rd = Rm + (Rm * 4) = Rm * 5
```

III.3.3.4 Instruction de multiplication

L'instruction de multiplication du ARM est un produit de deux nombres de 32 bits binaires tenus dans les registres. Le résultat de multiplication est un produit de 64 bits. Cette instruction, disponible seulement sur certaines versions du processeur, le résultat de 64 bits est stocké dans deux registres indépendamment spécifiés. D'autre processeur ARM stocke seulement les 32 bits de poids faible du résultat Dans tous les cas il y a une instruction de multiplication-accumulation qui ajoute le produit à un total accumulé et des opérandes signés et non signés peuvent être employés.

III.3.3.4.a) format de l'assembleur

Les instructions qui produisent les 32 bits de poids faible du produit :

```
MUL{<cond>}{S} Rd, Rm, Rs
MLA{<cond>}{S} Rd, Rm, Rs, Rn
```

Les instructions suivantes produisent un résultat complet de 64 bits :

<mul>{<cond>}{S}RdHi, RdLo, Rm, Rs where <mul>
Où <mul> est un des types de multiplication de 64 bits (UMULL, UMLAL, SMULL, SMLAL).

III.3.3.4.b) format binaire

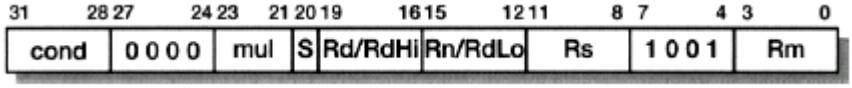


Figure III.7 Code binaire de l'instruction de multiplication

- **Description**

Les différentes formes des fonctions de multiplication sont mentionnées dans le tableau III.5.

Tableau III.5 instructions de multiplication. [13]

Opcode [23:21]	Mném-onique	Explication	Effet
000	MUL	Multiplication de deux registre (résultat de 32-bits)	Rd:=(Rm*Rs)[31:0]
001	MLA	Multiplication avec addition (résultat de 32-bits)	Rd:=(Rm*Rs + Rn)[31:0]
100	UMULL	Multiplication longue non signé (résultat de 64-bits)	RdHirRdLo := Rm * Rs
101	UMLAL	MUL longue non signé avec addition du contenu des--registres de destination	RdHi: RdLo += Rm * Rs
110	SMULL	Multiplication longue sur 64 bits signée	RdHi:RdLo := Rm * Rs
111	SMLAL	Multiplication longue avec addition sur 64 bits signée	RdHi:RdLo+=Rm*Rs

La notation employée dans le tableau est comme suit :

- 'RdHi:RdLo' est le nombre à 64 bits formé en attachant RdHi (les 32 bits de poids fort) et RdLo (les 32 bits de poids faible).
- La nomination simple est dénoté par ' : = '.
- L'addition (l'addition du côté droit à gauche) est dénoté par ' + = '.

Le bit S contrôle la mise en place des codes de condition comme avec autres instructions de traitement de données. Quand il survient l'instruction :

- le drapeau N est mis en place selon la valeur de bit 31 de Rd pour les formes qui produisent le résultat de 32 bits et le bit 31 de RdHi pour les longues formes.
- Le drapeau Z est mis en place si Rd ou RdHi et RdLo sont zéro.
- Le drapeau C est mis en place à une valeur sans signification.
- Le drapeau V est inchangé.

III.3.4 Instructions de transfert de données

Les instructions de transfert de données permettent de déplacer les données entre les registres du processeur ARM et la mémoire. Les instructions de transfert de données du ARM sont toutes basées autour de l'adressage indirect de registre, avec les modes qui incluent l'adressage de base plus index et la base plus offset. L'adressage indirect de registre utilise une valeur dans un registre (le registre de base) comme une adresse de mémoire et ou bien charge la valeur de cette adresse dans un autre registre ou bien stocke la valeur d'un autre registre dans cette adresse de mémoire. Ces instructions sont écrites avec l'assembleur comme suit :

```
LDR          r0, [r1]          ; r0 := mem32[r1]
STR          r0, [r1]          ; mem32[r1] := r0
```

Pour charger ou stocker d'un ou dans un emplacement particulier de la mémoire, un registre de ARM doit être initialisé pour contenir l'adresse de cet emplacement, ou, dans le cas d'instructions de transfert de registre simples. Pour cela les assembleurs du ARM ont une pseudo instruction ADR, qui fait ça facile. Une instruction pseudo ressemble une instruction normale dans le code source d'assembleur mais ne correspond pas directement à une instruction particulière du ARM. Comme un exemple, considérons un programme qui doit copier des données de TABLE1 dans TABLE2 :

```
Copy    ADR    r1,   TABLE1      ; r1 Pointe au TABLE1
        ADR    r2,   TABLE2      ; r2 pointe au TABLE2
        . .
TABLE1                           ; < la source de donnée >
        . .
TABLE2                           ; < destination >
```

Si le registre de base ne contient pas exactement l'adresse juste, un offset jusqu'à 4 KO peut être ajoutée (ou soustraite) à (de) la base pour calculer l'adresse de transfert :

```
LDR          r0, [r1,#4 ]          ; r0 := men32[r1+ 4]
```

C'est le mode d'adressage pré indexé. Il permet à un registre de base d'être employé pour avoir accès à quelques emplacements de mémoire qui sont dans le même secteur. Parfois il est utile de modifier le registre de base pour indiquer l'adresse de transfert. Cela peut être réalisé en employant l'adressage pré indexé avec l'auto-indexation et permet au programme de marcher par une table de valeurs :

```
LDR          r0, [r1,#4]!          ; r0 := mem32[r1+ 4]
                                   ; r1 := r1+ 4
```

Le point d'exclamation indique que l'instruction doit mettre à jour le registre de base après l'introduction du transfert de données. Une autre forme, appelée l'adressage indexé de poste, permet à la base d'être employée sans un offset comme l'adresse de transfert, après qu'il soit auto indexée :

```
LDR          r0, [r1], #4          ;r0 := mem32 [r1]
                                   ;r1 := r1 + 4
```

Ici le point d'exclamation n'est pas nécessaire, puisque la seule utilisation de l'offset immédiate va modifier le registre de base. De nouveau, cette forme de l'instruction est exactement équivalente d'un chargement simple indirect de registre suivi par une instruction de traitement de données, mais c'est plus rapide et occupe moins d'espace de code.

Remarque

La taille de donnée qui est transféré peut être un seul octet non signé de 8 bits au lieu d'un mot de 32 bits. Cette option est choisie en ajoutant une lettre B sur l'opcode :

```
LDRB        r0,  [r1]          ; r0 := mem8 [r1]
```

L'octet chargé est placé dans l'octet de fond de r0 et les octets restants dans r0 sont remplis de zéro.

III.3.4.1 Instructions de Chargement – Rangement *(load-store)* d'un seul registre

Ces instructions sont la façon la plus flexible de transférer des octets simples ou des mots entre les registres du ARM et la mémoire.

III.3.4.1.a) format de l'assembleur

La forme pré indexée de l'instruction :

LDR/STR {< cond >} {B} Rd [, Rn, < offset>] {!}

La forme indexée de poste :

LDR/STR {< cond >} {B} {T} Rd, [Rn], <offset > A

La forme utile relative de PC qui laisse l'assembleur pour faire tout le travail :

LDR/STR {< COnd >} {B} Rd, ÉTIQUETTE

LDR est ' *load register*', STR est '*store register* '; 'B' pour le choit facultatif d'un transfert d'octet non signé, par défaut est le mot; < offset> peut être # + /-< l2-bit immédiat > ou + /- Rm {, le décalage} où le spécificateur de décalage est le même qu'une instruction de traitement de données sauf que le registre de valeurs de décalage indiquées n'est pas disponible; '!' utilisé pour une écriture déféré (auto-indexant) dans la forme pré indexée.
Le drapeau T choisit la vue d'utilisateur de la traduction de mémoire et le système de protection et si seulement être employé dans des modes de non-utilisateur. L'utilisateur doit entièrement comprendre l'environnement de gestion de la mémoire dans lequel le processeur est employé, donc c'est vraiment une facilité pour des experts de système d'exploitation.

III.3.4.1.b) format binaire

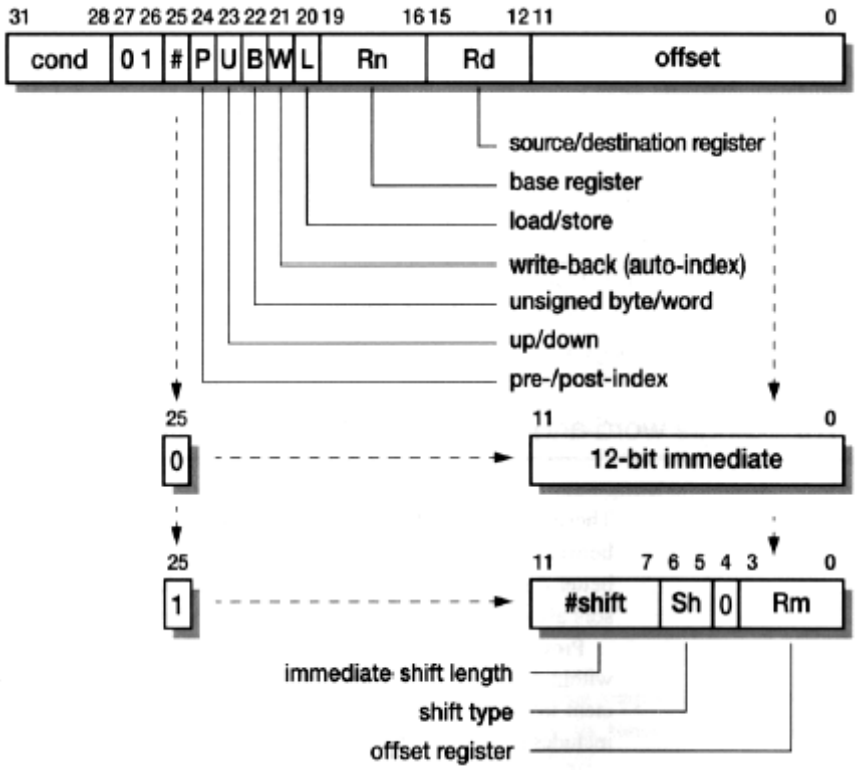


Figure III.8 Format binaire d'une instruction de transfert de donnée d'un seul registre et d'un octet non signé.

- **Description**

Ces instructions produisent une adresse commençant d'un registre de base (Rn), ajoutant ensuite (U = 1) ou soustrayant (U = 0) une valeur immédiat non signé ou un offset de registre. La base ou l'adresse calculée sont employées pour charger (L = 1) ou stocker (L = 0) une quantité d'un octet non signé (B = 1) ou d'un mot (B = 0) de ou dans un registre (Rd), de, ou, à la mémoire. Quand un octet est chargé dans un registre c'est zéro étendu à 32 bits. Quand un octet est stocké dans la mémoire, les huit bits de fond du registre sont stockés dans l'emplacement adressé. Un pré indexé (P = 1) l'adressage du mode emploie l'adresse calculée pour la charge ou l'opération de rangement et ensuite, quand une écriture déférée est demandée (W =1), met à jour le registre de base à la valeur calculée.
Le mode d'adressage indexé de poste (P = 0) utilise le registre de base non modifié pour le transfert et met ensuite à jour le registre de base à l'adresse calculée sans tenir compte du bit W (puisque l'offset n'a aucune signification d'autre que comme un modificateur de registre de base et peut toujours être mise au zéro immédiat si on ne désire aucun changement).

III.3.4.2 Instructions de transfert de multiples registres

III.3.4.2.a) format de l'assembleur

Quand une quantité considérable de données doit être transférée, il est préférable de déplacer plusieurs registres à la fois. Ces instructions permettent à n'importe quel sous-ensemble (ou tous) des 16 registres d'être transférés avec une instruction simple. Le compromis est que les modes d'adressage disponibles sont plus limités qu'avec une instruction de transfert de registre simple. Un exemple simple de cette classe d'instruction est :

```
LDMIA    r1, {r0, r2, r4}    ; r0 := mem32[r1]
                             ; r2 := mem32[r1+4]
                             ; r4 := mem32[r1+8]
```

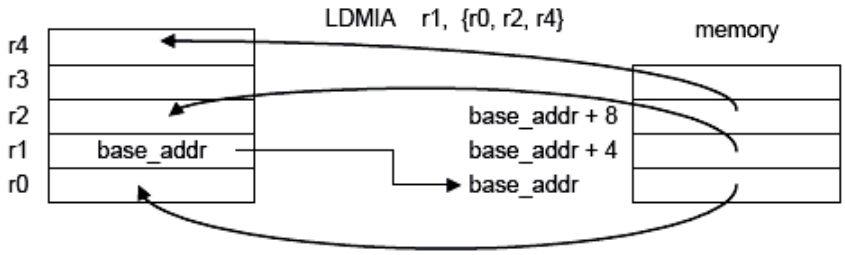


Figure III.9 Effet de l'instruction LDMIA

L'instruction complémentaire de LDMIA est l'instruction STMIA :

```
STMIA    r1, {r0, r2, r4}    ; mem32[r1] := r0
                             ; mem32[r1 + 4] := r2
                             ; mem32[r1 + 8] := r4
```

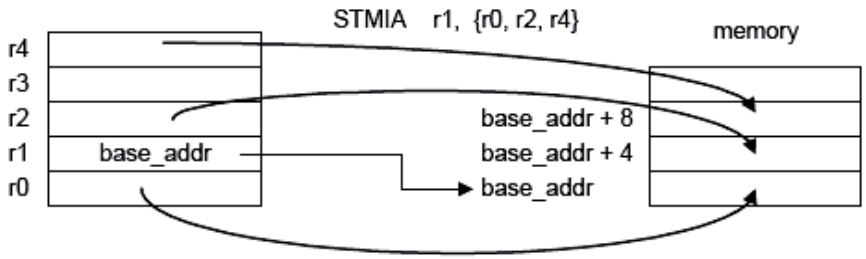


Figure III.10 Effet de l'instruction STMIA

Jusqu'à ici, r1, le registre d'adresse de base, n'a pas été changé. Nous pouvons mettre à jour ce registre d'indicateur en ajoutant « ! » après r1 comme il est présenté dans la figure III.10.

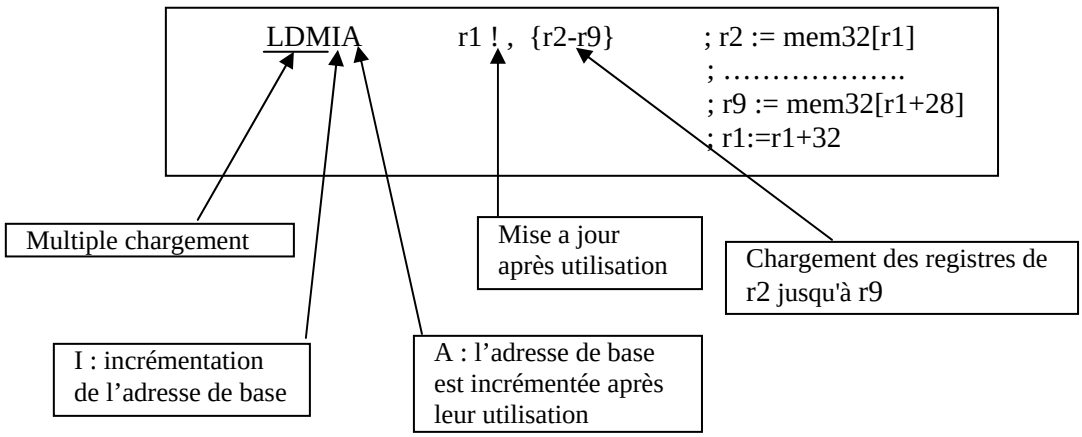


Figure III.11 explication de l'instruction LDMIA

En fait, on pourrait employer 4 formes différentes de chargement/Rangement de multiples registres :

- Increment - After LDMIA et STMIA
- Increment - Before LDMIB et STMIB
- Decrement - After LDMDA et STMDA
- Decrement - Before LDMDB et STMDB

La figure III.12 illustre les 4 modes de l'instruction de rangement de multiples registres

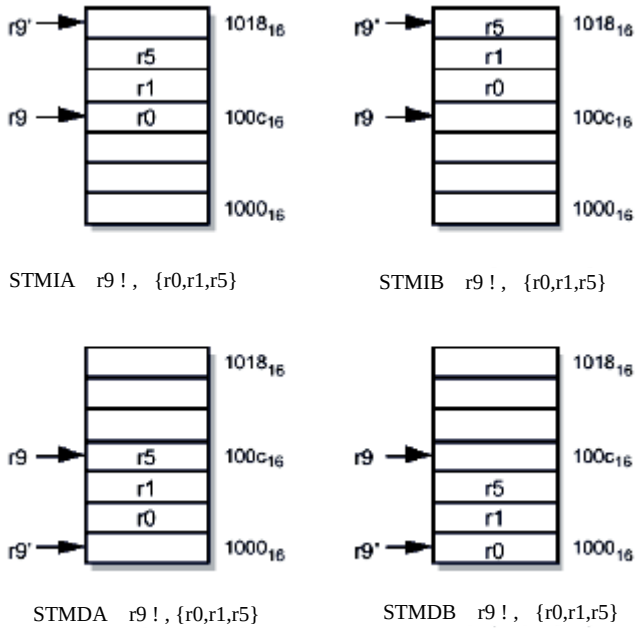


Figure III.12 les 4 modes de l'instruction chargement rangement de multiples registres

III.3.4.2.b) format binaire

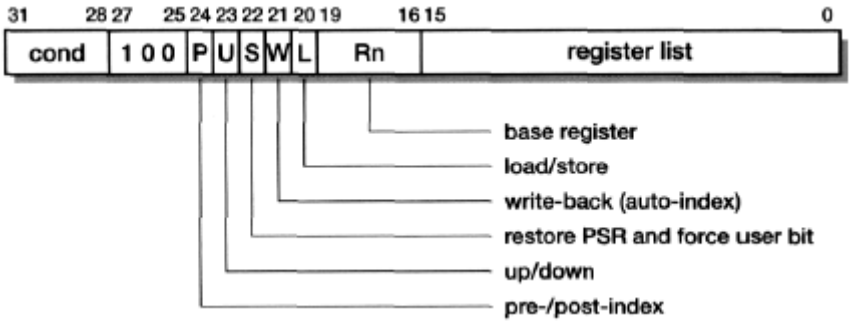


Figure III.13 Code binaire de l'instruction de transfert de données de multiples

- **Description**

La liste de registre dans le fond de 16 bits de l'instruction inclut un bit pour chaque registre visible, avec le bit 0 on contrôle si vraiment ou non r0 est transféré, le bit 1 contrôle r1, et cetera jusqu'au bit15 qui contrôle le transfert du PC. Les registres qui sont chargés ou stockés de, ou à, un bloc contigu de mots de mémoire sont définis par le registre de base et le mode d'adressage. L'adresse de base sera incrémentée (U = 1) ou décrémentée (U = 0) avant (P = 1) ou après (P = 0) chaque transfert de mot. L'auto-indexation est soutenue; si W = 1 le registre de base sera augmenté (U = 1) ou diminué (U = 0) par le nombre d'octets transférés quand l'instruction est achevée. Les formes spéciales de l'instruction permettent d'être employées pour rétablir le CPSR : si le PC est dans la liste de registre d'une charge multiple et le bit S est mis en place, le SPSR du mode actuel sera copié dans le CPSR, donnant un retour atomique et rétablir l'instruction d'état. Cette forme ne doit pas être employée dans le code de mode d'utilisateur puisque il n'y a aucun SPSR dans le mode d'utilisateur. Si le PC n'est pas dans la liste de registre et le bit S est mise en place et charge et stocker des instructions multiples exécutées dans des modes de non-utilisateur transféreront les registres de mode d'utilisateur (en employant le registre de base de mode actuel). Cela permet à un système d'exploitation de sauver et rétablir l'état de processus d'utilisateur.

III.3.5 Instructions d'échange entre mémoire et registre (SWP)

Les instructions d'échange combinent une chargement et rangement d'un mot ou un octet non signé dans une seule instruction.

3.3.5.a) format binaire

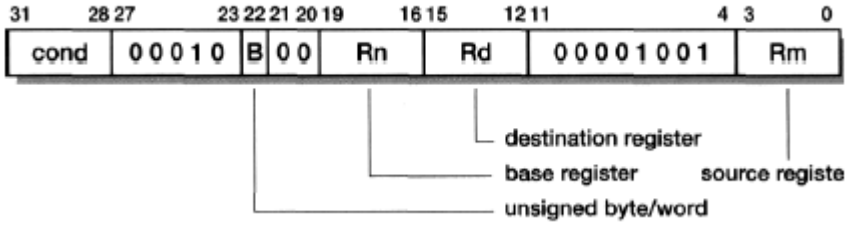


Figure III.14 Code binaire de l'instruction d'échange entre registre et mémoire.

- **Description**

L'instruction de chargement de mot (B = 0) ou l'octet non signé (B = 1) dans l'emplacement de mémoire adressé par Rn dans Rd et stocke le même type de données de Rm dans le même emplacement de mémoire. Rd et Rm peut être le même registre (mais sont distincts de Rn), dans le cas où le registre et des valeurs de mémoire est échangé. Le ARM exécute la lecture séparée de mémoire et ensuite les cycles de l'écriture de la mémoire, mais affirme un signal de verrouillage pour indiquer au système de mémoire que les deux cycles ne doivent pas être séparés.

III.3.5.b) Format de l'assembleur

SWP{<cond>}{B} Rd, Rm,[Rn]

Exemple

```
ADR     r0, SEMAPHORE
SWPB    r1, r1, [r0]            ; exchange byte
```

III.3.6 Instructions de transfert de registre d'état au registre général

Quand il est nécessaire de sauvegarder ou modifier le contenu du CPSR ou le SPSR du mode actuel, ce contenu doit d'abord être transféré dans un registre général, les bits choisis modifiées et ensuite la valeur rendue au registre d'état. Ces instructions accomplissent la première étape.

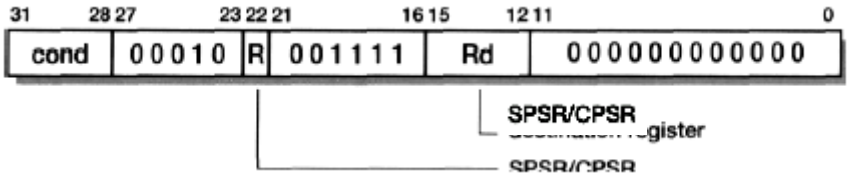


Figure III.15 Code binaire de l'instruction de transfert de registre d'état au registre général

Le CPSR (R = 0) ou le mode actuel SPSR (R = 1) est copié dans le registre de destination (Rd). Tous les 32 bits sont copiés. La forme d'assembleur de cette instruction est :

MRS {<cond>} Rd, CPSRISPSR

Exemple

```
MRS   r0,CPSR                 ; placement de CPSR dans  r0

MRS   r3,SPSR                 ; placement de  SPSR dans r3
```

III.3.7 Instructions de transfert de registre général au registre d'état

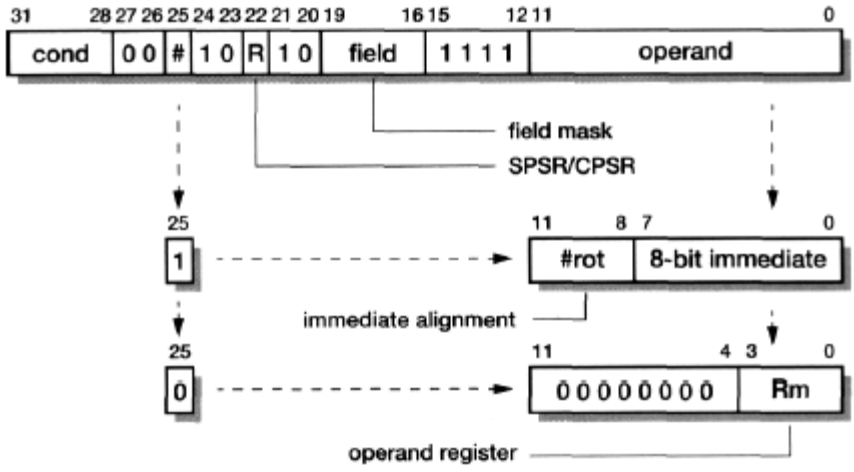


Figure III.16 Code binaire de l'Instructions de transfert de registre général au registre d'état

L'opérande, qui peut être un registre (Rm) ou un immédiat tourné à 8 bits (indiqué de la même manière comme la forme immédiate d'opérande dans les instructions de traitement de données), est déplacé sous un masque des champs au CPSR (R = 0) ou le mode actuel SPSR (R = 1). Le masque des champs contrôle la mise à jour des champs de quatre octets dans le registre de PSR. Les 16 bits d'instruction détermine si PSR [7:0] est mis à jour, le bit 17 contrôle PSR [15:8], le bit 18 contrôle PSR [23:16] et le bit 19 contrôle PSR [31:24]. Quand un opérande immédiat est employé seulement les drapeaux (PSR [31:24]) peuvent être choisis pour la mise à jour. (Ceux-ci sont les seuls bits qui peuvent être mises à jour par le code de mode d'utilisateur.)
La forme d'assembleur de cette instruction est :

MSR{<cond>} CPSR_fISPSR_f, #<32-bit immediate>
MSR{<cond>} CPSR_<field>ISPSR_<field>, Rm

Où < f ield > est un de :

- c - le champ de contrôle - PSR [7:0].
- x - le champ d'extension - PSR [15:8] (inutilisé sur ARMs actuels).
- s - le champ de statut - PSR [23:16] (inutilisé sur ARMs actuels).
- f- le champ de drapeaux -PSR [31:24].

III.3.8 Instructions de branchement et d'interruptions

III.3.8.1 Instructions de branchement et branchement avec lien (B,BL)

Les instructions de branchement et Branchement avec un lien sont la façon standard de changer l'ordre d'exécution des instructions.

3.3.8.1.a) format binaire

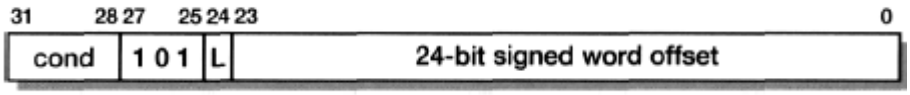


Figure III.17 Code binaire de l'instruction de branchement.

- **Description**

Ces instructions causent que le processeur commence à exécuter des instructions d'une adresse calculée par l'offset à 24 bits indiquée dans l'instruction, en le décale à droite par deux places pour former une offset de mot, on l'ajoute ensuite au compteur de programme qui contient l'adresse de l'instruction de branchement plus huit octets. L'assembleur va calculer l'offset correcte dans des circonstances normales. La gamme de l'instruction de branche est + /-32 Mo. Le branchement avec lien, qui a le bit L (le bit24), déplace aussi l'adresse de l'instruction après le branchement dans le registre de lien (r14) du mode de processeur actuel. C'est normalement employé pour exécuter un appel de sous-programme, avec le retour étant causé en copiant le registre de lien déferrement dans le PC. Toutes les deux formes de l'instruction peuvent être exécutées conditionnellement ou inconditionnellement.

III.3.8.1.a) format de l'assembleur

B{L}{<cond>} <target address>

'L'indique le lien et le branchement; si 'L' n'est pas inclus une branchement sans lien est produite. ' <Cond> ' doit être une des extensions mnémoniques données dans le tableau 3.3, ou, si est omis, 'AL-' est assumé. ' < L'adresse cible > ' est normalement une étiquette dans le code d'assembleur; l'assembleur produira le offset (qui sera la différence entre l'adresse de la cible et l'adresse de l'instruction de branchement plus 8).

Exemple

Un saut inconditionnel

```
                B           LABEL     ;  Saut inconditionnel ..
                ..
LABEL           ..                    ; .. Ici
```

Appeler un sous-programme

```
                ..
                BL      SUB       ; branchement et relier  au sous-programme  SUB
                ..                ; Retour ici
                ..
SUB             ..                ; Le point d'entrée de sous-programme
                MOV   PC,  r14    ; Retour
```

Remarque

Il existe d'autres types de ces instruction BX et BXL pour les ARMs qui supportent le jeu d'instruction compressé thumb (16 bits) et sont un mécanisme pour commuter le processeur pour exécuter des instructions thumb ou pour le retour symétriquement au appel d'un sous programme ARM et thumb. Une instruction thumb semblable cause que le processeur rentre dans des instructions ARM de 32 bits.

III.3.8.2 Interruption logiciel (SWI)

L'instruction d'interruption logicielle est employé pour des appels au système d'exploitation et est souvent appelé ' un appel de superviseur'. Il met le processeur dans le mode de superviseur et commence à exécuter des instructions de l'adresse 0x08.

III.3.8.2.a) format binaire

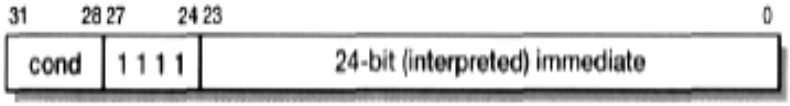


Figure III.18 Code binaire de l'instruction SWI

- **Description**

Le champ de la valeur immédiat de 24 bits n'influence pas sur l'opération de l'instruction, mais peut être interprété par le code de système. Si la condition faite passe l'instruction pour entrer dans le mode de superviseur employant une exception standard du ARM. En détail, les actions de processeur sont :

1. Sauve l'adresse de l'instruction après le SWI dans r14_svc.
2. Sauve le CPSR dans SPSR_SVC.
3. Entre en mode de superviseur et mettre hors de service IRQS (mais pas FIQs) en mettant CPSR [4 :0] à 100112 et CPSR [7] à l.
4. Mettre le PC à 0816 et commence à exécuter les instructions de cette adresse.

Pour retourner à l'instruction après le SWI la routine de système ne doit pas seulement copier r14_svc en arrière dans le PC, mais il doit aussi rétablir le CPSR de SPSR_SVC. Cela exige l'utilisation d'une des formes spéciales de l'instruction de traitement de données décrite précédemment.

III.3.8.2.b) format de l'assembleur

SWI {<cond>} <valeur immédiate de 24 bits>

Exemple

Pour visualiser le caractère A

```
MOV     r0, # 'A'          ; mettre 'A' dans r0..
SWI     SWI_WriteC         ; et imprime le
```

Un sous-programme de visualisation d'une chaîne de caractères après l'appel :

```
        ..
        BL      STROUT          ; visualiser le message suivant
                =       « HELLO WORLD », &0a, &0d, 0
        ..                      ;revient ici
                                        ..
STROU   LDRB    r0,   [r14],   #1       ; mettre les caractères
        CMP     r0,   #0                ; Vérifiez pendant la fin le marqueur
        SWINE   SWI_WriteC              ; si n'est pas la fin, imprime
        BNE     STROUT                  ; . .   et boucler
        ADD     r14,  #3                ; Alignez au mot suivant
        BIC     r14,  #3
        MOV     PC,   r14               ;Retour
```

Pour mettre fin à l'exécuter de programme d'utilisateur et retour au programme de moniteur

```
SWI     SWI_Exit        ;retour au moniteur
```

III.3.9 Instructions de coprocesseur

L'architecture du ARM supporte un mécanisme général pour étendre le jeu d'instruction par le complément de coprocesseurs. L'utilisation la plus commune d'un coprocesseur est le coprocesseur de système contrôler des fonctions sur chip comme la mémoire cache et l'unité de gestion de mémoire sur l'ARM720. Le coprocesseur du ARM de la virgule flottante a aussi été développé et des coprocesseurs d'application spécifiques est une possibilité.

Les coprocesseurs ARM ont leurs propres jeux de registre détectives et leur état est contrôlé par les instructions du ARM. Le ARM a la responsabilité unique du flux de contrôle, donc les instructions de coprocesseur sont concernées par le transfert de données et le traitement de données.

III.3.9.1 Opérations de données de coprocesseur

Ces instructions sont employées pour contrôler des opérations internes sur des données dans les registres de coprocesseur. Le format standard suit la forme de 3-address des instructions de traitement de données entier du ARM, mais d'autres interprétations de tous les champs de coprocesseur sont possibles.

III.3.9.1.a) format binaire

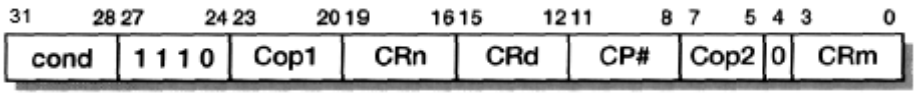


Figure III.19 Code binaire d'opérations de données de coprocesseur.

- **Description**

Le coprocesseur identifié avec le nombre de coprocesseur CP# qui va accepter l'instruction et exécuter l'opération défini par les champs Copl et Cop2, employant CRn et CRM comme des source des opérandes et plaçant le résultat dans CRd.

III.3.9.1.b) format de l'assembleur

CDP{<cond>} <CP#>, <Copl>, CRd, CRn, CRm{, <Cop2>}

III.3.9.2 Transferts de données de coprocesseur

Les instructions de transfert de données de coprocesseur sont semblables aux formes de offset immédiates du mot et des instructions de transfert de données d'octet non signées décrites plus tôt, mais avec l'offset limitée à huit bits plutôt que 12. Les formes Auto-indexées sont disponibles, avec l'adressage pré et indexé de poste.

III.3.9.a) format binaire

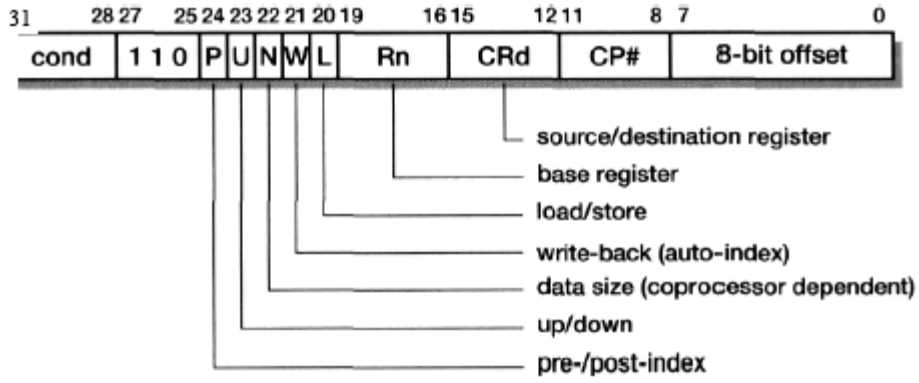


Figure III.20 Code binaire de transferts de données de coprocesseur

- **Description**

Le calcul d'adresse a lieu dans le ARM, employant un registre de base de ARM (Rn) et une offset immédiate à 8 bits. Le mode d'adressage et l'auto-indexation sont contrôlés de la même manière comme les instructions de transfert de mot et des d'octet non signées du ARM. Les données sont fournies par ou reçues dans un registre de coprocesseur (CRd), avec le nombre de mots transférés étant contrôlées par le coprocesseur et le bit N choisissant une des deux longueurs possibles.

III.3.9.2.b) format de l'assembleur

La forme pré indexée :

LDCISTC{<cond>}{L} <CP#>, CRd, [Rn, <offset>]

La forme post indexée { ! } :

LDCISTC{<cond>}{L} <CP#>, CRd, [Rn], <offset>

Dans les deux cas LDC choisit un chargement de la mémoire dans le registre de coprocesseur, STC choisit un rangement du registre de coprocesseur dans la mémoire. Le drapeau L, s'il est mis en place, choisit le long type de données (N = 1). <Offset> est # immédiate + /-<8-bit;-.

III.3.9.3 Transferts de registre de coprocesseur

Ces instructions permettent à un entier produit dans un coprocesseur d'être transférées directement dans un registre du ARM ou les drapeaux de code de condition du ARM. Des utilisations typiques sont :

- Une opération de virgule flottante FIX qui rend l'entier à un registre du ARM;
- Une comparaison de virgule flottante que rend le résultat de la comparaison directement au ARM, conditionne des drapeaux de code où il peut déterminer le flux de contrôle;
- une opération de FLOAT qui prend une valeur d'entier d'un registre du ARM et l'envoie au coprocesseur où il est converti à la représentation de virgule flottante et placé dans un registre de coprocesseur.

III.3.9.3.a) format binaire

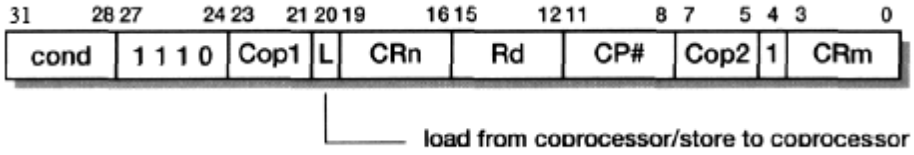


Figure III.21 Code binaire de Transferts de registre de coprocesseur.

- **Description**

Si un coprocesseur accepte une instruction de chargement de coprocesseur, il va exécuter normalement une opération définie par Copl et Cop2 sur des opérandes source CRn et CRm et va rendre un résultat d'entier de 32 bits au ARM qui le placera dans Rd. Si un coprocesseur accepte une instruction de rangement à un coprocesseur, il va accepter un entier de 32 bits de registre Rd du ARM. Si le PC est spécifié comme le registre de destination Rd dans une

instruction de chargement de coprocesseur, les quatre bits de sommet de l'entier à 32 bits produits par le coprocesseur sont placées dans le N, Z, C et des drapeaux V dans le CPSR.

III.3.9.3.b) format de l'assembleur

Déplacement du coprocesseur au registre de ARM

MRC{<cond>} <CP#>, <Copl>, Rd, CRn, CRm{,<Cop2>}

Déplacement du registre de ARM au coprocesseur

MCR{<cond>} <CP#>, <Copl>, Rd, CRn, CRm{,<Cop2>}

III.4 LE JEU D'INSTRUCTION COMPRESSEE THUMB

Le jeu d'instruction thumb s'adresse à la densité de code. Il peut être vu comme une forme compressée d'un sous-ensemble du jeu d'instruction du ARM. L'implémentation de thumb emploie la décompression dynamique dans un pipeline d'instruction du ARM et ensuite les instructions seront exécutées comme des instructions standard du ARM dans le processeur. Le jeu d'instruction thumb est entièrement supporté par des outils de développement de ARM et une application peut mélanger des sous-programmes de jeu d'instruction ARM standard et de thumb avec tout souplesse pour optimiser les performances.

III.4.1 Le bit T de CPSR

Les processeurs ARMs qui supportent le jeu d'instruction thumb peuvent aussi exécuter le jeu d'instruction ARM standard de 32 bits et l'interprétation du flot d'instruction à n'importe quel temps particulier est déterminée par le bit 5 du CPSR, Si T est mis en place le processeur interprète le flot d'instruction comme des instructions thumb de 16 bits, si non, il l'interprète comme des instructions ARM standard. Pas tous les processeurs ARMs ne sont capables d'exécuter des instructions thumb; seulement qui ont une lettre T dans leur nom, comme l'ARM7**T**DMI décrit dans le chapitre II. Pour commuter à une instruction thumb on utilise l'instruction de branchement et changement **BX**.

III.4.2 Le model de programmateur de *thumb*

Le jeu d'instruction thumb est un sous-ensemble du jeu d'instruction de ARM ou les instructions utilisent un nombre limité de registres. Le modèle du programmeur est illustré dans la figure III.21. Le jeu d'instruction donne le plein accès au huit registres généraux ' Lo' r0 à r7 et fait l'utilisation étendu de r13 à r15 pour des buts spéciaux :

- r13 est employé comme un indicateur de pile.
- r14 est employé comme le registre de liaison.
- r15 est le compteur de programme (le PC).

Les registres restants (r8 à r12 et le CPSR) ont seulement l'accès si :

- Quelques instructions permettent aux ' Hi' registres (r8 à r15) d'être spécifié.
- Les drapeaux de code de condition de CPSR sont mis en place par des opérations arithmétiques et logiques et contrôlent le branchement conditionnel.

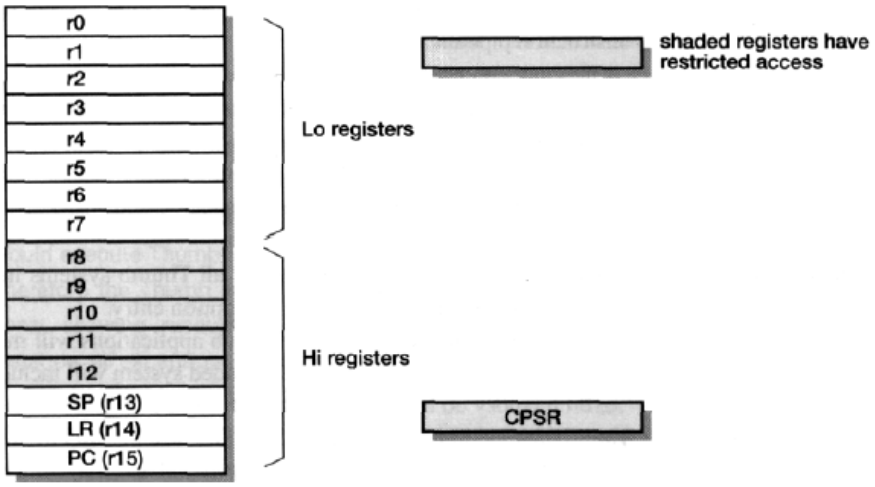


Figure III. 22 Les registres accessibles du jeu d'instruction thumb.

Toutes les instructions thumb sont de 16 bits de largeur. Ils s'exécutent avec même façon des instructions ARM donc ils héritent beaucoup de propriétés du jeu d'instruction ARM standard :

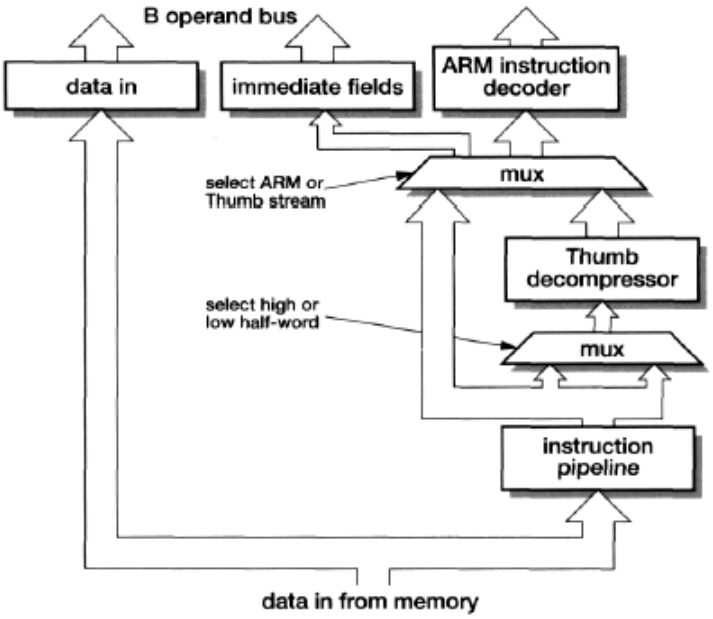
- L'architecture de chargement/rangement avec le traitement de données, le transfert de données et le contrôle de flux des instructions.
- support le type de donnée d'un octet de 8 bits, de demi mot de16 bits et de mot de 32 bits.
- une mémoire de 32 bits non segmentée.

Cependant, pour achever une longueur d'instruction de 16 bits quelques particularités caractéristiques du jeu d'instruction ARM ont été abandonnées :

- la Plupart des instructions thumb sont exécuté inconditionnellement. (Par contre toutes les instructions ARM sont exécutées conditionnellement.)
- Beaucoup d'instructions de traitement de données thumb utilisent un format de 2 adresse (le registre de destination est le même qu'un des registres source). (Les instructions de traitement de données ARM, à l'exception de la multiplication de 64 bits, utilisent un format de 3 adresses.)
- des formats d'instruction thumb sont moins réguliers que des formats d'instruction ARM, suite au codage dense.

III.4.3 IMPLEMENTATION DES INSTRUCTIONS THUMB

Le jeu d'instruction thumb peut être incorporé dans une même macrocellule de processeur ARM à un pipeline de 3 étages avec des changements relativement secondaires à la plupart de logique de processeur. Le plus grand ajout est le décompresseur d'instruction thumb dans le pipeline d'instruction; cette logique traduit une instruction thumb à son instruction ARM équivalente. On montre l'organisation de cette logique dans la Figure III.22. L'ajout de la logique de décompresseur en série avec le décodeur d'instruction peut augmenter la latence de décodage.



FigureIII.23 Organisation de décompresseur des instructions thumb.

III.4.3 APPLICATION DES INSTRUCTIONS THUMB

Pour voir les avantages que nous offrent les instructions thumb nous devons citer en détail ses propriétés. Les instructions thumb sont de longueur de 16 bits et codent la fonctionnalité d'une instruction ARM avec la moitié de nombre de bit, mais puisqu'une instruction thumb a typiquement moins contenu sémantique qu'une instruction ARM, un programme particulier exigera plus d'instructions thumb que cela ait eu besoin d'instructions ARM. La proportion variera de programme au programme, mais dans un code thumb d'exemple typique peut exiger 70 % de l'espace de code ARM. Donc si nous comparons la solution thumb avec la solution de code ARM pure, les caractéristiques suivantes apparaissent : [4]

- Le code thumb exige 70 % de l'espace du code ARM.
- le code de Pouce utilise 40 % plus d'instructions que le code ARM.
- Avec une mémoire de 32 bits, le code ARM est 40 % plus rapide que le code thumb.
- Avec une mémoire de 16 bits, le code thumb est 45 % plus rapide que le code ARM.
- le code de Pouce utilise 30 % de puissance mémoire externe moins que le code de ARM.

Ainsi où les performances sont toutes importantes, un système doit employer la mémoire de 32 bits et exécuter le code ARM. Où le coût et la consommation électrique sont plus importants, un système de mémoire de 16 bits et le code thumb peuvent être un meilleur choix. Cependant, il y a les positions intermédiaires qui peuvent donner le mieux des deux :

- un système de ARM de 32 bits de haute qualité peut employer le code thumb pour certaines routines non-critiques pour économiser des nécessités de mémoire ou de puissance.

- un système de 16 bits de basse gamme peut avoir une petite portion de la RAM de 32 bits sur chip pour des routines critiques exécutant le code ARM, mais employer le code thumb hors chip pour toutes les routines non critiques.

III.5 CONCLUSION

Toutes les instructions ARM ont une largeur de 32 bits (sauf les instructions thumb de 16 bits compressées) et sont alignées sur des lignes à 4 octets dans la mémoire. Les particularités les plus notables du jeu d'instruction ARM sont :

- l'architecture de chargement rangement (load-store)
- des instructions de traitement de données à 3 adresses (c'est-à-dire les deux registres d'opérande source et le registre de résultat sont tout indépendamment spécifiés);
- exécution conditionnelle de chaque instruction;
- l'inclusion des instructions très puissante de chargement et rangement de multiples registres.
- La possibilité d'exécuter une opération de décalage générale et une opération générale de ALU dans une seule instruction qui s'exécute dans un cycle d'horloge simple;
- l'extension ouverte de jeu d'instruction par le jeu d'instruction de coprocesseur, incluant l'addition de nouveaux registres et des types de données au modèle du programmeur;
- une représentation compressée très dense de 16 bits de jeu d'instruction dans l'architecture de thumb.

CHAPITRE IV

SUPPORT ARCHITECTURAL POUR LE DEVELOPPEMENT D'UN SYSTEME A BASE DU PROCESSEUR ARM.

IV.1 INTRODUCTION

La conception d'un système d'ordinateur est une tâche complexe, Concevoir un système incorporé sur puce (SoC) est encore plus complexe. Le développement a souvent lieu intégralement dans un environnement de CAO, et le premier silicium ne doit pas seulement fonctionner correctement mais doit offrir les performances nécessaires et aussi être réalisable. La seule portée pour la fixation de défauts de conception est dans le logiciel, puisque la modification dans la puce pour corriger des erreurs prend un temps et ainsi augmente le coût de la fabrication.

Pendant les deux dernières décennies l'approche principale au développement de système de microprocesseur a été basée sur l'émulateur sur circuit ICE (*In Circuit Emulator*). Le système lui-même était un circuit imprimé incorporant un microprocesseur, des mémoires, et des autres périphériques.

Pour employer l'émulateur ICE, le microprocesseur doit être enlevé et connecter convenablement à l'équipement de l'émulateur. Le ICE émule le fonctionnement du microprocesseur et donne une vue intérieure à l'utilisateur sur l'état interne du système, donnant l'accès pour lire et modifier le contenu des mémoires et des registres du processeur et mettre des points de contrôle, etc.….

Actuellement le microprocesseur lui-même est devenu juste une cellule sur une grande puce, ces approches entières qui se sont effondrées. Il n'est pas possible de déconnecter une partie d'une puce il n'y a, encore, aucune approche qui a remplacé l'émulateur ICE dans tous ses rôles, mais il y a plusieurs techniques qui nous aide, quelques une exigent un support explicite dans l'architecture du processeur. Ce chapitre couvre les techniques disponibles pour supporter le développement de système sur chip basées sur des coeurs du ARM et les particularités architecturales assemblés dans les coeurs pour aider dans ce processus. À la fin de ce chapitre on va citer quelque type des unités centrales sur chip à base du processeur ARM entier couplé à une cache et unité de gestion de mémoire MMU par exemple. Selon l'exigence de l'application.

IV.2 HIERARCHIE DE LA MEMOIRE

Un microprocesseur moderne peut exécuter des instructions à un très haut taux. Pour exploiter cette performance entièrement potentielle le processeur doit être connecté à un système de mémoire qui est très grand et très rapide. Si la mémoire est trop petite, il ne sera pas capable de tenir assez de programmes. Si c'est aussi lent, la mémoire ne sera pas capable de fournir rapidement les instructions que le processeur peut les exécuter. Malheureusement, une grande capacité de la mémoire implique qu'elle est très lente. Il n'est pas donc possible de concevoir une seule mémoire qui est assez grande et assez rapide pour supporter un processeur très performant. Il est, cependant, possible de construire un système de mémoire composé qui se combine une petite, mémoire rapide et une grande mémoire principale mais relativement lente, pour présenter un comportement externe avec la statistique d'un programme typique, ce la se semble comme une grande mémoire. La petite mémoire rapide est la mémoire cache, qui maintient automatiquement les copies des instructions et des données que le processeur les emploie le plus fréquemment. L'efficacité de la cache dépend des propriétés de la localité spatiale et de localité temporelles du programme. Ce principe de mémoire à deux niveaux peut être étendu dans une hiérarchie de mémoire à plusieurs niveaux et le disque dur de l'ordinateur peut être vu comme une partie de cette hiérarchie. Avec l'appui de gestion de mémoire approprié, la taille d'un programme est limitée pas par la mémoire principale de l'ordinateur, mais par la taille du disque dur, qui peut être beaucoup plus grand que la mémoire principale.

IV.2.1 La vitesse et la taille de la mémoire

Une hiérarchie de mémoire d'ordinateur typique comprend plusieurs niveaux, chaque niveau ayant une taille et une vitesse caractéristique. [4]

- Les registres de processeur peuvent être vu comme le sommet de la hiérarchie de mémoire. Un processeur RISC aura typiquement autour de trente deux registres de 32 bits faisant un total de 128 octets, avec un temps d'accès de quelques nanosecondes.
- La mémoire cache sur chip d'une capacité de huit à 32 KO et un temps d'accès autour de dix nanosecondes.
- Des systèmes très performants de bureau peuvent avoir un deuxième niveau de cache secondaire hors chips avec une capacité de quelques cent KO et un temps d'accès de quelques dizaines de nanosecondes.
- La mémoire principale sera des méga-octets aux dizaines des méga-octets de mémoire dynamique avec un temps d'accès autour de 100 nanosecondes.
- Le magasin de secours (*backup store*), d'habitude sur un disque dur, sera des centaines de Mo jusqu'à quelques Go avec un temps d'accès de quelques dizaines de millisecondes.

Notons que la différence des performances entre la mémoire principale et le magasin de secours est beaucoup plus grand que la différence entre autres niveaux adjacents, même quand il n'y a aucune cache secondaire dans le système. Les données qui sont tenues dans les registres sont dans le contrôle direct du compilateur ou le programmeur d'assembleur, mais le contenu des niveaux restants de la hiérarchie est d'habitude géré automatiquement. Les caches sont efficacement invisibles à l'application, avec des blocs ou ' les pages d'instructions et des données migrant en haut et en bas de la hiérarchie dans le contrôle de matériel. La pagination entre la mémoire principale et le magasin de secours est contrôlée par le système d'exploitation et reste transparente à l'application. Puisque la différence des performances

entre la mémoire principale et le magasin de secours est si grand, beaucoup plus d'algorithmes sophistiqués sont exigés ici pour déterminer quand migrer des données entre les niveaux.
Un système incorporé n'aura pas d'habitude un support de stocke et n'exploitera pas donc la pagination. Cependant, beaucoup de systèmes incorporés incorporent des caches et des chips de CPU de ARM emploient une gamme d'organisations de cache. Nous regarderons donc l'organisation de la cache dans quelque détail. Une mémoire rapide est plus chère par bit que la mémoire lente, donc une hiérarchie de mémoire aspire aussi à donner une performance près de la mémoire la plus rapide avec un coût moyen par bit s'approchant de celui de la mémoire la plus lente.

IV.2.2 Mémoire sur chip

Quelque forme de mémoire sur chip est nécessaire si un microprocesseur doit livrer sa meilleure performance. Avec des taux d'horloge d'aujourd'hui, seulement la mémoire sur chip peut supporter l'état zéro d'attente des vitesses d'accès et il donnera aussi la meilleure efficacité de puissance et a réduit l'interférence électromagnétique que la mémoire hors chips. Dans beaucoup de systèmes incorporent une simple RAM sur chip est préférée qu'une cache pour un bénéfice de nombreuses raisons : [7]

- C'est plus simple, et emploie moins de puissance.
- a un comportement plus déterminé.

L'inconvénient avec une RAM sur chip vis a vis de la cache est qu'il exige la gestion explicite par le programmeur, tandis qu'une cache est d'habitude transparente au programmeur. Où le mélange de programme est bien défini et dans le contrôle du programmeur, la RAM sur chip peut efficacement être employée comme une cache contrôlée de logiciel. Où le mélange d'application ne peut pas être prévu que cette tâche de contrôle devient très difficile. De là une cache est d'habitude préférée dans n'importe quel système universel où le mélange d'application est inconnu. Un avantage important de la RAM sur chip est qu'il permet au programmeur d'y répartir l'espace employant la connaissance du chargement de prochain traitement. Une cache laisse à ses propres dispositifs de la connaissance seulement de comportement de programme passé et il ne peut jamais donc se préparer d'avance pour des tâches prochaines critiques. De nouveau, c'est une différence qui est la plus probable pour être significative quand des tâches critiques doivent rencontrer des contraintes strictes en temps réel. Le designer de système doit décider qui est l'approche juste pour un système particulier, prenant tous ces facteurs en considération. Bien que la forme de mémoire sur chip soit choisie, il doit être spécifié avec grand soin. Il doit être assez rapide de poursuivre le processeur occupé et assez grand pour contenir des routines critiques, mais ni trop vite (ou il consommera trop de puissance) ni trop grand (ou il occupera trop de secteur de chip).

IV.2.3 Mémoire cache

Une mémoire cache est une petite, mémoire très rapide qui maintient les copies de valeurs de mémoire récemment employées. Il fonctionne d'une manière transparente au programmeur. Ces jours c'est d'habitude mis en oeuvre sur le même chip que le processeur. Les caches travaillent parce que les programmes présentent normalement la propriété de localité, ce qui signifie qu'à n'importe quel temps particulier ils ont tendance à exécuter les mêmes instructions plusieurs fois (par exemple dans une boucle) sur les mêmes secteurs de

données (par exemple une pile). Les caches peuvent être construites de plusieurs façons. Au niveau le plus haut un processeur peut avoir une des deux organisations suivantes :

- Une cache unifiée : C'est une cache simple, des instructions et des données stockés ensemble, comme illustré dans la Figure IV.1.a.
- Une cache des instructions et de données Séparée.
  Cette organisation est parfois appelée une architecture de Harvard modifiée comme indiqué dans la Figure IV.1.b.

Toutes ces deux organisations ont leurs mérites. La cache unifiée s'ajuste automatiquement la proportion de la mémoire cache employée par des instructions selon les exigences de programme actuelles, donnant une meilleure performance qu'une partition fixée. D'autre part les caches séparées permettent aux instructions de rangement et chargement d'exécuter dans un seul cycle d'horloge.

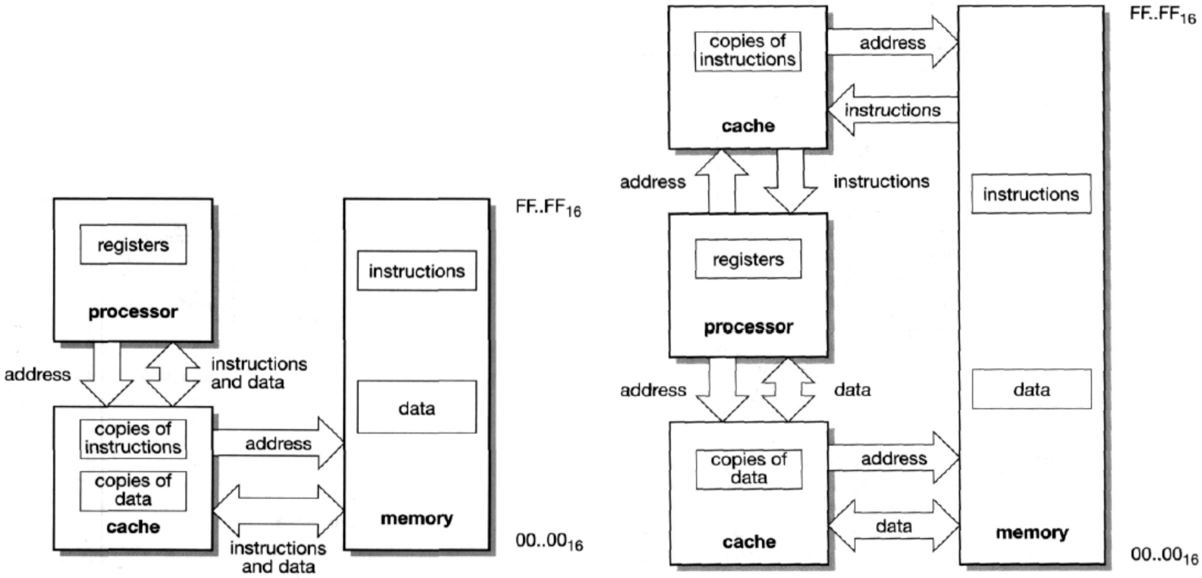


Figure IV.1 Organisation de mémoire cache (a) cache avec des instructions et de données unifié. (b) cache avec des instructions et de données séparés

Puisque le processeur peut fonctionner à son haut taux d'horloge seulement quand les éléments de mémoire qu'il requête sont tenus dans la cache, la performance complète de système dépend fortement de la proportion des accès de mémoire qui ne peuvent pas être satisfaits par la cache. Un accès à un élément qui est dans la cache est appelé un ***hit*** et un accès à un élément qui n'est pas dans la cache est une ***miss***. La proportion de tous les accès de mé **a)** cache avec des instructions et de est ***le tau*** **b)** cache avec des instructions et de un
pou données unifié. satisfait es données séparés

IV.2.3.1 Organisation du cache

Puisqu'une cache stocke un choix dynamiquement variable des éléments de la mémoire principale, il doit avoir stocker les données et les adresse à laquelle les données sont stockées dans la mémoire principale.

IV.2.3.1.a) Cache mappée directement (*the direct mapped cache*) **:**

L'organisation la plus simple de ces composants est la cache mappée directement qui est illustrée dans la figure IV.2. Une ligne (**line**) de données est stockée avec une étiquette (**tag**) d'adresse dans une mémoire qui est adressée par quelque portion de l'adresse de mémoire (**index**). Pour vérifier si vraiment un élément de mémoire particulier est stocké dans la cache, les bits d'adresse d'index sont employés pour avoir accès à l'entrée de cache. Les bits de l'adresse supérieure sont alors comparés aux étiquettes stockées, si sont égaux l'élément est dans la cache.

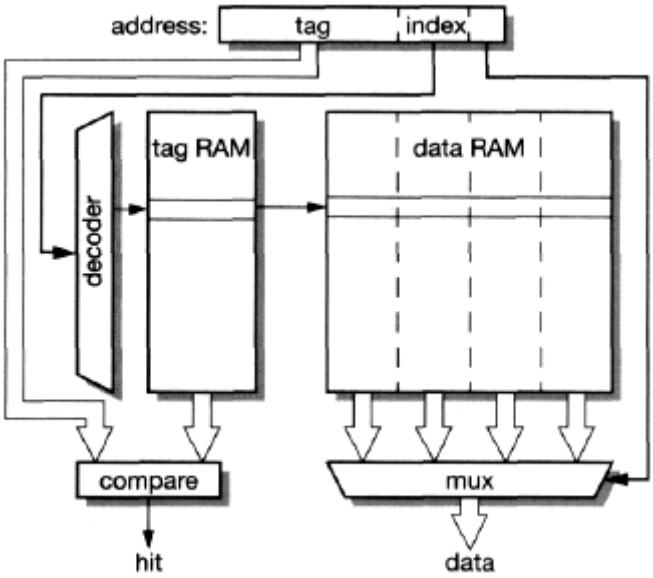


Figure IV. 2 Organisation de cache mappée directement.

IV.2.3.1.b) jeu de deux caches associées (*the set associative cache*)

Une cache de 2 voies associées est illustrée dans la figure IV.3. Cette forme de cache est effectivement deux caches directement mappées fonctionnant en parallèle. Une adresse présentée dans la cache peut trouver ses données dans l'une ou l'autre moitié, donc chaque adresse de mémoire peut être stockée en chacune de deux places. Chacun des deux éléments proposés pour un seul emplacement dans la cache mappée directement peut maintenant occuper une de ces places, permettant à la cache d'envoyer les deux.

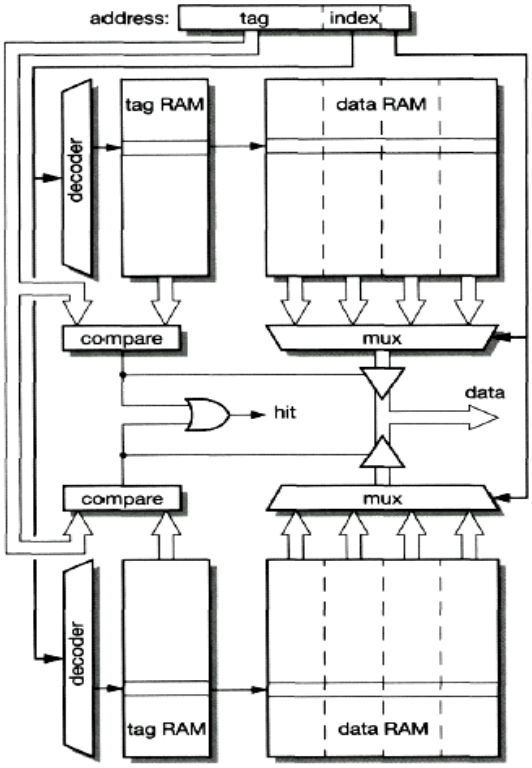


Figure IV. 3 Organisation de deux caches associées.

IV.2.3.1.b) Cache entièrement associative (*the fully associative cache)*

À un autre extrême d'associativité, il est possible de concevoir une cache entièrement associative dans la technologie VLSI. Plutôt que la continuation de diviser la cache directe mappée à des composants plus petits, le dépôt d'étiquette est conçus différemment employant le contenu de la mémoire adressée CAM (*content adrdressed memory*). La cellule de CAM est une cellule de RAM avec un comparateur, donc un CAM basé sur dépôt d'étiquette peut exécuter une recherche parallèle pour localiser une adresse dans n'importe quel emplacement. L'organisation d'une cache entièrement associative est illustrée dans la Figure IV.4.

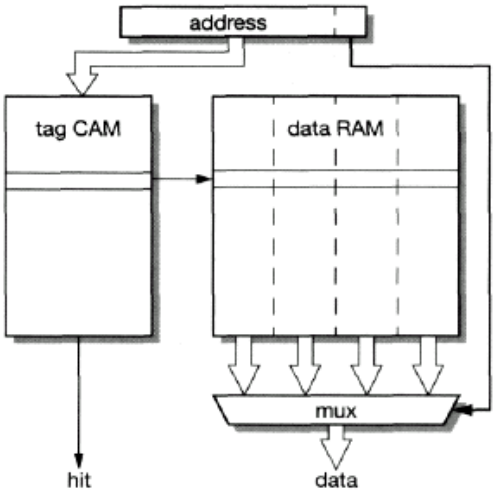


Figure IV. 4 Organisation d'une cache entièrement associative

IV.2.4 Gestion de la mémoire

Des systèmes d'ordinateur modernes ont typiquement beaucoup de programmes actifs en même temps. Un processeur simple peut, bien sûr, seulement exécuter des instructions d'un seul programme à n'importe quel instant, mais avec la commutation rapide entre les programmes actifs va sembler qu'ils vont s'exécuter au même temps. La commutation rapide est gérée par le système d'exploitation, donc le programmeur d'application peut écrire son programme comme s'il possède la machine entière. Le mécanisme a eu l'habitude de soutenir cette illusion est décrit par l'unité de gestion de mémoire **MMU** (**M**emory **M**anagement **U**nit). Il y a deux approches principales de la gestion de la mémoire, appelée la segmentation et la pagination.

IV.2.4 .1 Segmentation

La forme la plus simple de gestion de la mémoire permet à une application de voir sa mémoire comme un jeu de segments, où chaque segment contient une collection particulière d'information. Par exemple, un programme peut avoir un segment de code contenant toutes ses instructions, un segment de données et un segment de pile. Chaque accès de mémoire fournit un sélecteur de segment et une adresse logique au MMU. Chaque segment y a une adresse de base et une limite associée. L'adresse logique est une offset de l'adresse. Le mécanisme d'accès pour MMU segmenté est illustré dans la figure IV.5.

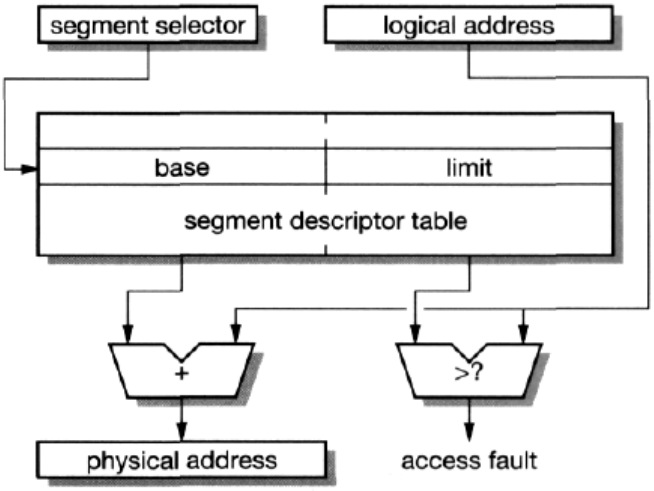


Figure IV.5 Mécanisme de gestion de mémoire segmenté

La segmentation permet à un programme d'avoir sa propre vue de mémoire et coexister d'une manière transparente avec d'autres programmes dans le même espace de mémoire. Il s'exécute avec difficulté, quand les programmes de coexistence varient et la mémoire disponible est limitée. Puisque les segments ont de taille variable, la mémoire libre devient fragmentée dans certain temps et un nouveau programme peut être incapable de commencer, pas parce que la mémoire libre est insuffisante, mais parce que dans la mémoire libre aucune pièce n'est assez grand pour tenir le segment de la taille exigée par le nouveau programme.
La situation peut être soulagée par le système d'exploitation déplaçant des segments autour dans la mémoire pour unir la mémoire libre dans un grand morceau, mais c'est inefficace et la plupart des processeurs incorporent maintenant un arrangement de mappe mémoire basé sur

les gros morceaux de taille fixé de mémoire appelée des pages. Quelques architectures incluent la segmentation et la pagination, mais beaucoup, incluant le ARM, supportent juste la pagination sans segmentation.

IV.2.4.2 Pagination

Dans un arrangement de gestion de mémoire de pagination les deux espaces des adresses logiques et des adresses physiques sont divisés dans des composants de taille fixée appelés des pages. Une page est d'habitude quelques kilo-octets dans la taille, mais des architectures différentes emploient des tailles de page différentes. Le rapport entre les pages logiques et physiques est stocké dans des tableaux de page, qui sont tenus dans la mémoire principale. Une seule somme montre que le stockage de la traduction dans un seul tableau exige une très grand tableau : si une page est 4 KO, 20 bits d'une adresse 32 bits doivent être traduites, qui exige 2^{20} x 20 bits de données dans le tableau, ou un tableau d'au moins 2.5 Mo. C'est un peu raisonnable au-dessus, pour imposer à un petit système. Au lieu de cela, la plupart des systèmes de pagination emploient deux, ou plus, de niveaux de table de page. Par exemple, le sommet de dix bits de l'adresse peuvent être employés pour identifier le tableau de page appropriée de deuxième niveau dans la liste d'adresses de tableau de page du premier niveau et la seconde dix bits de l'adresse identifie alors l'entrée de table de page qui contient le numéro de page physique. Cet arrangement de traduction est illustré dans la figure IV.6.

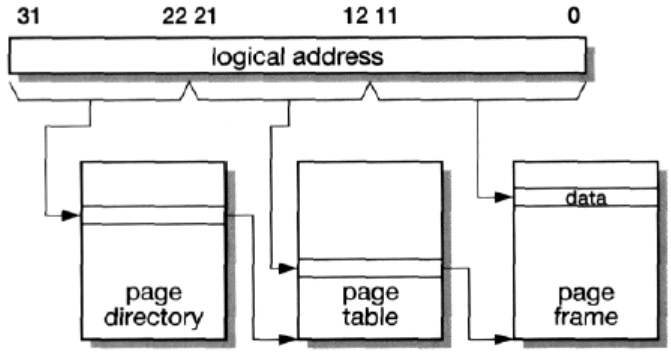


Figure IV. 6 Arrangement de gestion de mémoire de pagination

IV.3 L'INTERFACE DE LA MEMOIRE DU PROCESSEUR ARM

Dans cette section nous allons voir les principes généraux impliqués dans la connexion d'un processeur ARM à un système de mémoire construit a partir des portions de mémoire standard. L'efficacité de l'interface mémoire est un déterminant important de la performance du système, donc ces principes doivent être bien compris par le concepteur qui veut développer un système très performant. Les cœurs ARM plus récents utilise l'interface directe AMBA (voir plus loin).

IV.3.1 Les signaux de bus du processeur ARM

Les processeurs ARMs varient dans les détails de leur interface de bus, mais ils sont généralement semblables dans la nature. Les signaux de bus d'interface de la mémoire incluent :

- Le bus d'adresse, A [31 :0].
- Le bus bidirectionnel de données, D [31.0], le long lequel les données sont transférées.
- Les signaux qui spécifient si la mémoire est indispensable ($\overline{mreq}$) et si l'adresse est séquentiel (sep).
- Les signaux qui spécifient la direction ($\overline{r}/w$) et la taille (($\overline{b}/w$) pour les premiers processeurs, *mas [1:0]* pour les derniers processeurs) du transfert.
- Bus de contrôle (*abe, ale, ape, dbe, lock, bl [3:0]*).

IV.3.2 Simple Interface de mémoire

Le plus simple interface de la mémoire est approprié pour fonctionner avec la ROM et la RAM statique (SRAM). ces dispositifs exigent que l'adresse soit fixe jusqu'à la fin du cycle, qui peut être réalisé en mettant hors de service du pipeline d'adresse (*ape* en état bas) pour les premiers processeurs ou bien le re-timing le bus d'adresse (connectant *ale* à $\overline{mclk}$) pour les derniers processeurs . Les bus d'adresse et de donnée peuvent alors être connectés directement aux portions de mémoire comme il est indiqué dans la figure IV.7.qui montre aussi les signaux $\overline{RAMoe}$ et $\overline{ROMoe}$ (*RAM et ROM enable output*) et $\overline{RAMwe}$ (*RAM write enable).*

Cette figure illustre la connexion des portions de mémoire à 8 bits, qui est une configuration standard pour les SRAMs et les ROMs, quatre portions de chaque type sont nécessaires pour construire une mémoire à 32 bits. La notation sur la figure montre la numérotation des bus à l'intérieur du dispositif et les fils de bus auxquels sera connectés à l'extérieur du dispositif, si, par exemple, les pins D [7:0] de la SRAM qui est la plus proche du ARM sont connectés aux bus D [31:24] qui est connecté aux pins D (31:24) du ARM. Les deux lignes d'adresse de pois faible, A[1:0], sont employées pour la sélection d'octet, ils sont employés par la logique de contrôle et n'ont pas connectés à la mémoire.

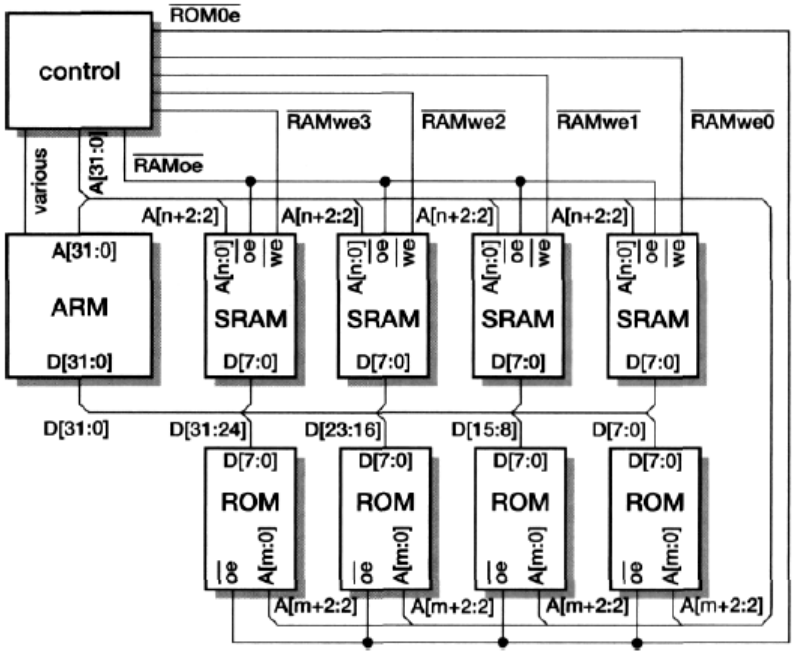


Figure IV. 7 Système de mémoire de base du ARM

Certainement le processeur ARM va lire telle donnée sous forme d'un octet ou d'un mot, le système de mémoire peut ignorer la différence. Le processeur ARM va extraire l'octet adressé et ignorera le reste du mot. Donc les ROMs n'ont pas besoin d'un bit de validation individuelle et l'utilisation d'un dispositif à 16-bit ne cause aucun problème. L'octet écrit, cependant, exige un octet de validation individuelle, donc la logique de contrôle doit produire quatre octets de commande de validation d'écriture. Cela fait l'utilisation difficile de plus grandes RAMs (et inefficace) à moins qu'ils joignent l'octet de validation séparément.

IV.3.3 La logique de contrôle

La logique de contrôle accomplit les taches suivantes :

- décide quand activer la RAM et quand activer la ROM. Cette logique détermine la carte mémoire de système. Le processeur commence a partir du position zéro après la remis (reset), donc il doit trouver au premier lieu la ROM puisque la RAM n'est pas initialisés. La carte mémoire la plus simples valide donc la ROM si A [31] est au niveau bas et la RAM est au niveau haut.
- Il contrôle l'octet de validation d'écriture pendant l'opération d'écriture.
  Pendant l'écriture d'un mot tout les octet de validation doit être activées, pendant l'écriture d'un seul octet l'octet adressé doit être activé.
- Il assure que les données sont prêtes avant que le processeur continue. La solution la plus simple est d'obliger *mclk* d'être assez lente pour assurer que tous les dispositifs de mémoire peuvent avoir accès dans un simple cycle d'horloge. Des systèmes plus sophistiqués peuvent avoir un jeu l'horloge pour une suite d'accès à la RAM et utilisent des états d'attendre (typiquement plus lente) pour la ROM et les accès aux périphériques.

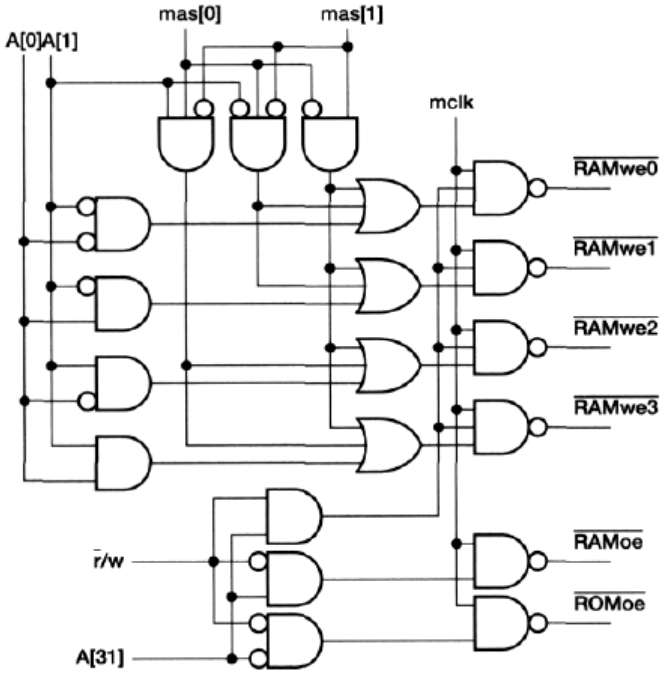


Figure IV. 8 la logique de contrôle d'un système de mémoire simple du ARM

La logique nécessaire pour accomplir les précédentes taches citées est tout à fait directement illustrée dans la figure IV.8. (Toute cette logique peut être implémentée employant un seul circuit logique programmable PLD

IV.3.4 États d'Attente

Si nous essayons d'accélérer l'horloge dans ce système il n'y a pas un problème concernant la RAM, par contre la ROM ne peut pas travailler avec la même vitesse, des états d'attente sont introduits pour permettre plus de temps d'accès à la ROM.
La logique de contrôle de la mémoire doit incorporer une simple machine d'état pour contrôler l'accès de la ROM. On montre un diagramme de transition approprié d'état pour quatre cycles d'horloge dans la figure IV.9.

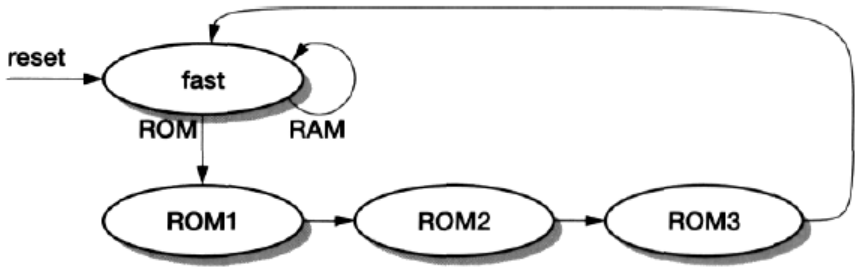


Figure IV. 9 Diagramme de transition d'état de contrôle d'attente du ROM

Trois états de ROM sont employés pour étaler le temps d'accès du ROM à quatre cycles affirmant l'entrée $\overline{wait}$ du processeur ARM.

IV.4 L'ARCHITECTURE DE BUS AMBA

Les bus d'interface du processeur ARM sont optimisés pour l'interfaçage d'une cache très rapide. Quand un coeur ARM utilisé, avec ou sans cache, comme un composant dans un système complet sur puce, quelques interfaces sont indispensables pour permettre au ARM de communiquer avec autre macrocelles.

L'Architecture de bus de Microcontrôleur avancée, AMBA (*Advanced Microcontroller Bus Architecture*), est spécifié pour standardiser la connexion sur chip des différentes macrocellules, et par conséquence permettre la réutilisation des macrocellules déjà conçues. Financement conception d'un système complexe sur un chip basé sur une nouvelle combinaison de macrocellules existantes pourrait devenir une tâche directe et facile.

Puisque *ARM ltd* est intéressé par la vente de la propriété intellectuelle IP qui inclut les microprocesseurs et d'autres périphériques, il est devenu utile de faire propager une structure de bus standard qui est ouverte pour aux ingénieurs pour employer dans la conception avec IP de ARM. Les issues comme la réutilisation de conception et la conception de système modulaire sont importantes pour *ARM ltd*. AMBA permet même plus de IP qui travaille ensemble avec ARM IP et fournit un jeu commun de protocoles pour l'infrastructure de bus qui est devenue de facto norme employée dans la conception des SoC aujourd'hui. AMBA est aussi utilisé dans des conceptions basées sur des microprocesseurs autres que ARM.

IV.4.1 Les bus AMBA

Trois bus sont définis dans la spécification AMBA :

- **AHB** (*advance high-performance Bus*) **:** Le bus avancé de grande performance est employé pour connecter des modules de système très performants. Le bus AHB est conçu pour remplacer l'ASB dans les systèmes de très hautes performances.
- **ASB** (*advanced system Bus*) **:** Le bus de système avancé est employé pour connecter des modules de système très performants. Il soutient le mode de transferts de données.
- **APB** *(Advanced Peripheral Bus)* **:** Le bus avancé de périphérique offre une interface des modules de système moins performants.

Un microcontrôleur typique basé sur un bus AMBA incorpore un AHB ou bien un ASB assemblé avec un APB comme illustré dans la figure IV.10. L'ASB est le bus le plus ancien de système, le bus AHB étant conçu plus tard pour améliorer les performances du système.
Le APB est généralement employé comme un bus local secondaire qui apparaît comme un simple module d'esclave pour le AHB ou l'ASB.

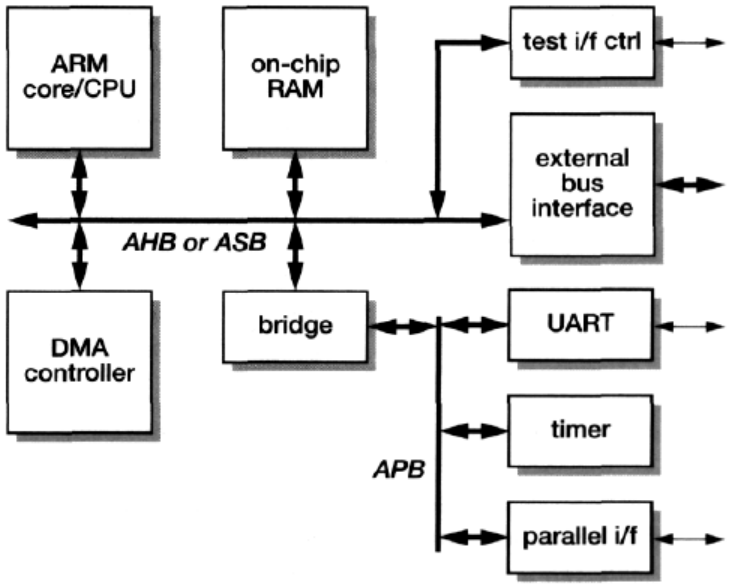


Figure IV. 10 Système typique à base de AMBA

Un arrangement des bus est introduit par le bus maître qui demande un accès d'un arbitre central. L'arbitre décide la priorité, quand il y a un conflit dans l'accès. Et sa conception est une issue spécifique de système. L'ASB seulement spécifie le protocole qui doit être suivi :

- Le maître, x, délivre une demande (AREQx) à l'arbitre central.
- Quand le bus est libre; l'arbitre donne une autorisation (AGNTx) au maître.

IV.5 OUTILS DE DEVELOPPEMENT DU PROCESSEUR ARM

Le développement de logiciel pour le processeur ARM est soutenu par une gamme logique d'outils développés par ARM Ltd, et il y a aussi beaucoup d'outils de domaine publics disponibles, comme le compilateur C gcc pour le ARM. Puisque le ARM est largement employé comme un contrôleur incorporé où le matériel cible n'est pas un bon environnement pour le développement de logiciel, les outils sont destinés pour le développement simultané (c'est-à-dire ils courent sur une architecture différente un pour lequel ils produisent le code) d'une plate-forme comme un PC travaille sous Windows. On montre la structure complète du toolkit de développement simultané de ARM dans la figure IV.11. Des fichiers de source C ou d'assembleur sont compilé ou assemblé dans des fichiers de format d'objet de ARM (*.aof*), qui sont alors liés dans des fichiers de format d'image de ARM (*.aif*). Les fichiers de format d'image peuvent être construits pour inclure les tables de débugue (debug) exigées par le débogueur symbolique de ARM (**ARMsd** qui peut charger, exécuter et contrôler des programmes sur le matériel '*hardware*' comme une carte de développement de ARM ou bien l'utilisation d'une émulation de logiciel du ARM (**ARMulator**). L'ARMulator a été conçu

pour permettre l'extension facile du modèle de logiciel d'inclure des particularités de système comme des caches, les caractéristiques particuliers de timing de la mémoire, et cetera.

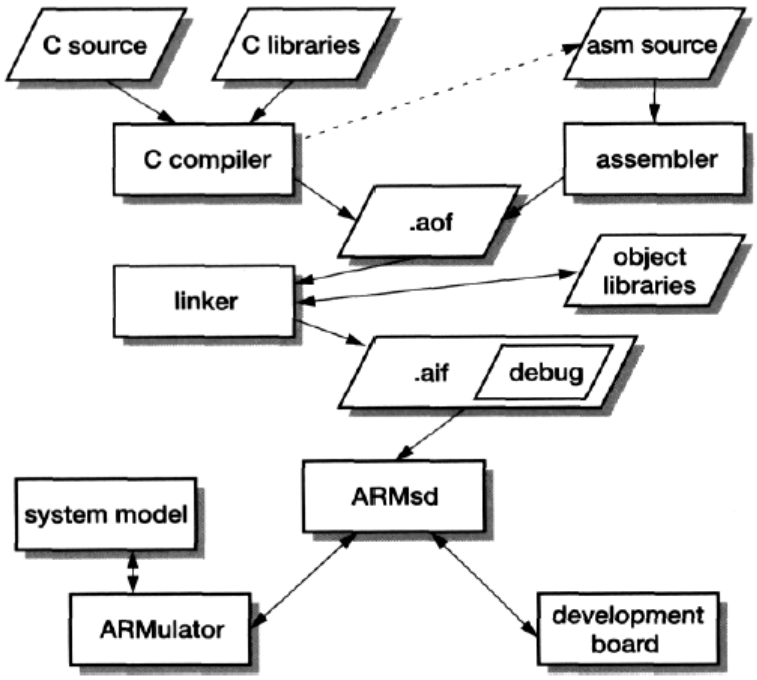


Figure IV.11 Structure d'une compilation (toolkit) de développement du ARM

Alors un toolkit de développement logiciel complet pour concevoir des applications pour les architectures ARM inclut :

- **Le compilateur C du ARM**

Le compilateur C du ARM est adapté avec la norme de l'ANSI (American National Standars Institute) pour C et est soutenu par la bibliothèque appropriée de fonctions standard. Il emploie la norme d'appel de procédure du ARM pour toutes les fonctions extérieurement disponibles. On peut dire au lieu du format d'objet de ARM, on peut produire une sortie de source d'assembleur qui donc peut être inspectée, ou même optimisée et assemblé par la suite. Le compilateur peut aussi produire le code thumb.

- **L'assembleur du ARM**

L'assembleur du ARM est un macro assembleur excessif qui produit la sortie de format d'objet de ARM qui peut être liée avec la sortie du compilateur C. Le langage de source d'assembleur est près du niveau de machine, avec la plupart des instructions d'assembleur traduisant en des instructions simple de ARM (ou Thumb).

- **Le linker**

Le linker prend un ou plusieurs fichiers objet et les combine dans un programme exécutable. Il résout les références symboliques entre les fichiers d'objet et les modules d'objet extraits de la bibliothèque nécessaire selon le programme. Il peut assembler les

composants divers du programme des façons différentes, selon si le code doit exécuter dans la RAM (Random Access Memory) ou la ROM (Read Only memory). Normalement le linker inclut des tables de débugue (mise au point) dans le fichier de sortie. Le linker peut aussi produire des modules de bibliothèque d'objet qui ne sont pas exécutables, mais sont prêts pour la jonction efficace avec des fichiers d'objet de future.

- **ARMsd**

Le débogueur symbolique du ARM est une interface (*front-end)* pour aider dans le déboguage de programmes exécutés sous une émulation (par exemple ARMultor) ou bien à distance sur un système cible comme un kit de développement de ARM. Le système doit soutenir les protocoles de déboguage appropriés à distance ou bien via une ligne série ou bien par une interface d'essai de JTAG (voir plus loin). Un ARMsd de base, permet à un programme exécutable d'être chargé dans l'ARMulator ou un kit de développement ensuite s'exécuté. Il permet la configuration de points de contrôle *breakpoints*, qui ont des adresses dans le code, s'ils sont exécutés, ils vont causer une halte pour que l'état de processeur puisse être examiné. Dans l'ARMulator, ou bien si en fonctionne sur un matériel avec le support approprié, il permet aussi la mise de point de contrôle *watchpoints*.

IV.5.1 ARMulator

L'ARMULATOR (l'émulateur de ARM) est une suite de programmes qui modèle le comportement de divers coeurs de processeur ARM sur un système hôte. Il peut fonctionner aux différents niveaux de précision :

- La précision des instructions : c'est la modélisation de comportement exact de l'état de système sans respect des caractéristiques de timing de précision du processeur.
- La précision de cycle : c'est la modélisation de comportement exact du processeur sur une base de cycle par cycle, permettant le nombre exact de cycles d'horloge qu'un programme exige pour être établi.
- La précision de timing : c'est la modélisation des signaux présents aux temps correct dans un cycle, permettant aux retards de logique d'être représenté.

Toutes ces approches dirigées sont considérablement plus lentes que dans le matériel réel. L'ARMultor a un rôle dans la conception de système incorporée. Il soutient le prototypage de haut niveau des différentes parties du système pour soutenir le développement de logiciel et l'évaluation d'alternatives architecturales. Il est composé de quatre composants :

- Le modèle de coeur de processeur, qui peut imiter n'importe quel coeur de ARM récent, incluant le jeu d'instruction thumb.
- Une interface de mémoire qui permet aux caractéristiques du système de mémoire cible d'être modelée. Des modèles divers sont fournis pour soutenir prototypage rapide, mais l'interface est entièrement personnalisable pour incorporer le niveau de détail exigé.
- Une interface de coprocesseur qui soutient des modèles de coprocesseur personnalisés.
- une interface de système d'exploitation qui permet aux appels de système d'être traités par l'hôte ou émulée sur le modèle de ARM.

L'ARMulator peut aussi être employé comme le coeur modèle comportemental d'un ARM de précision de timing dans un environnement de simulation de matériel basé autour d'un langage comme VHDL. Un ' **VHDL wrapper**' doit être produit à l'interface de code C de l'ARMulator à l'environnement VHDL.

IV.5.2 Les outils de prototype de système matériel

La tâche faisant face au designer de système sur chip d'aujourd'hui intimide. Le nombre de portes sur un chip tend à grandir à un taux exponentiel et est déjà dans les millions. Avec les meilleurs outils de conception de logiciel disponibles sur le marché, les designers ne peuvent produire des systèmes entièrement évalués de cette complexité dans les contraintes de temps. Le premier pas vers l'adressage de ce problème, comme indiqué, est une proportion significative de la conception sur des composants de conception préexistants. La réutilisation de conception peut réduire la quantité de nouveau travail de conception à une petite fraction du nombre total de portes sur le chip. Une approche systématique à la connexion sur chip à l'aide d'un bus comme AMBA en plus réduit la tâche de conception. Cependant, il y a des problèmes toujours difficiles à résoudre, comme :

- Comment le designer peut être sûr que tous les blocs réutilisables choisis, de diverses sources, vont travailler correctement ensemble ?
- Comment le designer peut être sûr que le système va avoir les performances exigées.
- Comment les designers de logiciel peuvent-ils retoucher leur travail avant que le chip soit disponible ?

La simulation de système employant des outils logiciel aboutit d'habitude à une performance de plusieurs ordres d'ampleur plus bas que celui du système final, rendant peu pratique le développement de logiciel et la vérification complète de système. Une solution qui va une voie considérable vers l'adressage de toutes ces issues est l'utilisation de matériel prototype : c'est un système de matériel qui combine tous les composants exigés dans une forme qui ne fait aucune tentative de rencontrer la puissance et les contraintes de taille du système final, mais fournit une plate-forme pour la vérification de système et le développement de logiciel. Le kit de développement de ARM est un ensemble de circuit incorporant une gamme de composants et des interfaces pour soutenir le développement de systèmes basé sur un processeur ARM. Il inclut un cœur ARM (par exemple, un ARM7TDMI), les composants de mémoire qui peuvent être configurés pour correspondre à la performance et la largeur de bus de la mémoire dans le système cible et les dispositifs programmables électriquement qui peuvent être configurés pour émuler des périphériques d'application spécifiques. Il peut soutenir, le développement matériel et logiciel avant que le matériel final d'application spécifique ne soit disponible.
Parmi ces systèmes de développement on trouve (*Rapid Silicon prototyping*) présenté par VLSI Technologie. La base de ce système est d'employer des chips de référence particulièrement développées, chacun est constitué d'un jeu spécifique de composants sur chip et supporte des extensions hors chips, qui peuvent être employées pour modeler une conception de système sur chip. Le système cible est modelé dans deux étapes :

1. Le chip de référence choisi est 'déconfiguré' pour rendre ces blocs qui ne sont pas exigés dans le système cible sur chip inactif.

2. Les blocs exigés dans le système cible qui n'est pas disponible sur le chip de référence est implémenté (mis en œuvre) comme des extensions hors chips. Ceux-ci peuvent être des circuits intégrés déjà existés avec la fonctionnalité nécessaire ou bien des FPGAS configuré à la fonction nécessaire utilisant un langage de haut niveau comme le VHDL pour la synthèse. L'utilisation des blocs préexistants, interconnectés utilisant le bus standard comme AMBA, va réduire au minimum le risque technique dans la production du chip final.

Tous les blocs dans le chip de référence existent comme des composants synthétisables. Les descriptions de langage de haut niveau des fonctions exigées sont prises ensemble avec la source de langage de haut niveau pour configurer le FPGAS, les fonctions déconfigurées sont lâchées et le résultat sera résynthétisé pour donner le chip cible. Ce processus est illustré dans la figure IV.12. Il est clair que cette approche dépend du chip de référence contenant les composants clefs appropriés, comme le coeur de µp et probablement un système de traitement de signal (où c'est exigé par l'application cible). Bien qu'en principe il puisse être possible de construire un chip de référence simple qui contient chaque coeur de processeur de ARM différent (incluant toute différente cache et des configurations de MMU), en pratique un tel chip ne serait pas économique même pour des buts prototypage. Le succès de l'approche dépend donc du bon choix des composants sur le chip de référence pour assurer qu'il peut couvrir une large distribution de systèmes, et différents chips de référence avec différente µp et d'autres coeurs clefs étant construits pour des domaines de différentes applications .

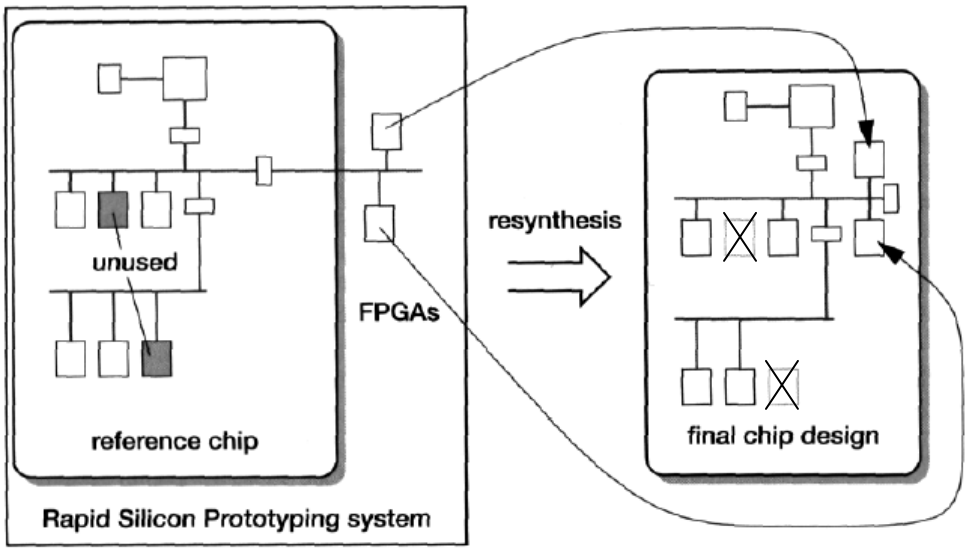


Figure IV. 12 Principe de prototype de Rapid Silicon

IV.5.3 Le processus de Conception de Système Incorporé

Pour lequel un processus de conception de système incorporée commence généralement par un jeu d'exigences que le produit doit faire. La disposition suivante est une liste des étapes dans le processus et un résumé court de ce qu'arrive à chaque état de la conception. On montre les marches dans la figure IV. 13.

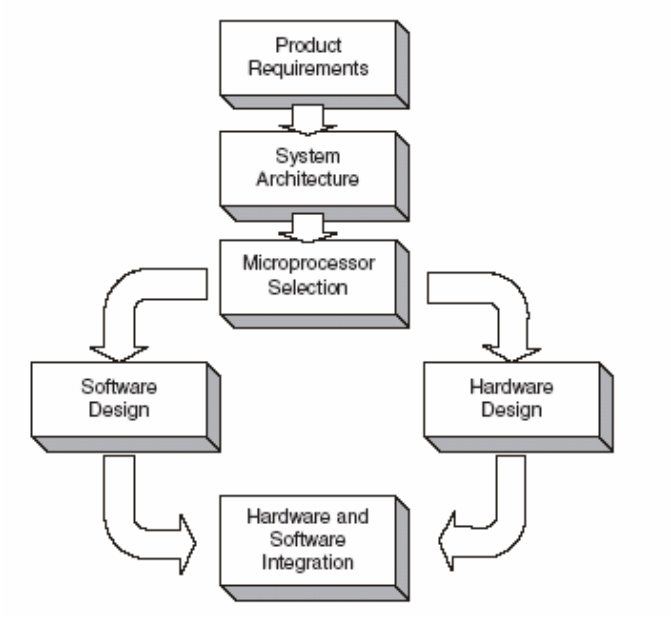


Figure IV. 12 Processus de conception d'un système incorporé.

- **Les spécifications de produit**

L'exigence des documents de phase des spécifications de produit ainsi que la définition des particularités sont exigées par la fonctionnalité du produit. Le marketing, l'ingénierie, ou autres experts du domaine doivent interpréter l'attente du client pour résoudre un problème spécifique, ces facteurs documentent les exigences du produit. La conquête des exigences correctes réussit le projet d'un bon début, réduit au minimum la probabilité de modifications de produit futur et assure qu'il y a un marché pour le produit. De bons produits résolvent des besoins réels, sont faciles à utiliser, et réalisent des bénéfices tangibles.

- **L'architecture de Système**

L'architecture de système définit les blocs principaux et les fonctions du système. Les interfaces, la structure des bus, la fonctionnalité du matériel et du logiciel sont déterminées. Les designers de système emploient des outils de simulation, des modèles de logiciel et des tableaux pour déterminer l'architecture qui le mieux s'adapte aux exigences de système.

- **Le choix de Microprocesseur**

Un des pas les plus difficiles dans la conception de système incorporé peut être le choix du microprocesseur. Il y a un nombre infini de manières, techniques et non technique, de comparer les microprocesseurs. Des facteurs importants incluent la performance, le coût, la puissance, les outils de développement de logiciel, choix de RTOS (*reel time operating system*) et les modèles de simulations disponibles.

- **La conception de matériel**

Une fois l'architecture mise en place et le processeur (s) choisi, le pas suivant est la conception de matériel, le choix des composants, le code de Verilog et VHDL, la synthèse, l'analyse de timing et la conception physique de chip et des kits de développement. L'équipe de conception de matériel produira quelques données importantes pour l'équipe de logiciel comme le plan (s) d'adresse de CPU et les définitions pour tout registres programmables de logiciel. L'exactitude de cette information est cruciale au succès du projet entier.

- **La conception de Logiciel**

Une fois le mappe de mémoire définie et les registres de matériel documentés, on engage le processus de développement du logiciel. des exemples incluent le chargement de code (*boot code*) pour mettre en marche le CPU et initialiser le système, le diagnostic de matériel, le système en temps réel d'exploitation (RTOS), des pilotes de dispositif et le logiciel d'application. Pendant cette phase, outils de compilation et la mise au point sont choisis et le codage est réalisé.

- **L'intégration de matériel et de Logiciel**

Le pas le plus crucial dans la conception de système incorporé est l'intégration de matériel et de logiciel. Quelque part pendant le projet le logiciel nouvellement codé rencontre le matériel nouvellement conçu.

- **La vérification et la Validation**

Afin d'intégrer le matériel et le logiciel, deux concepts sont importants la vérification et la validation. Ceux-ci sont les pas finals pour assurer qu'un système de travail rencontre les exigences de conception.

IV.6 L'INTERFACE STANDARD DE TEST JTAG

Le test de système embarqué sur chip pour une application spécifique est une opération, de longue durée, complexe, si on fait appel à la technique traditionnelle des sondes de mesure. Cette approche interdit d'ailleurs la mesure de signaux internes non transmis vers l'extérieur. Cet état de fait explique qu'un certain nombre de sociétés se soient concertées pour trouver une solution standard à ce problème. Plusieurs fabricants trouvèrent la situation suffisamment critique pour fonder une association, le comité JTAG (Joint Test Action Group). Le résultat tangible de cette collaboration est une interface spéciale dont nous allons voir le détail dans le cadre de cette section. Cette interface est nommée l'interface standard de test **JTAG** (**JTAG** *boundary scan testinterface*) ou bien IEEE 1149. Ce protocole, (Interface JTAG) développé par les membres du comité peut se targuer d'un certain nombre de caractéristiques marquantes. La caractéristique la plus remarquable est que cette interface travaille, depuis le composant, de façon totalement transparente et asynchrone. Cela signifie qu'il est possible de tester le composant concerné en cours de fonctionnement sans que cela n'ait la moindre influence néfaste sur son fonctionnement. Outre les caractéristiques et fonctions prévues par le standard, rien n'interdit au fabricant de circuit intégré d'ajouter ses

propres fonctionnalités à l'interface JTAG. C'est le cas de système à base de processeur ARM. La figure IV.13 représente la synoptique de l'interface JTAG. On reconnaît aisément les 4 signaux servant au pilotage du circuit JTAC.

- **$\overline{\text{TRST}}$** est une entrée de remise à zéro qui initialise l'interface d'essai.
- **TCK** représente le signal d'horloge nécessaire à la logique JTAG. Ce signal est parfaitement indépendant du signal d'horloge propre au système, ce qui explique que ces deux éléments soient totalement asynchrones l'un par rapport à l'autre. Tous les autres signaux de l'interface JTAG sont en synchronisme avec cette entrée d'horloge.
- **TMS** est l'élite de mode d'essai qui contrôle l'opération d'interface de la machine d'état.
- **TDI** est la ligne d'entrée de données d'essai qui fournit les données au JTAG ou à des registres d'instruction.

- **TDO** permet la lecture du contenu d'un registre.

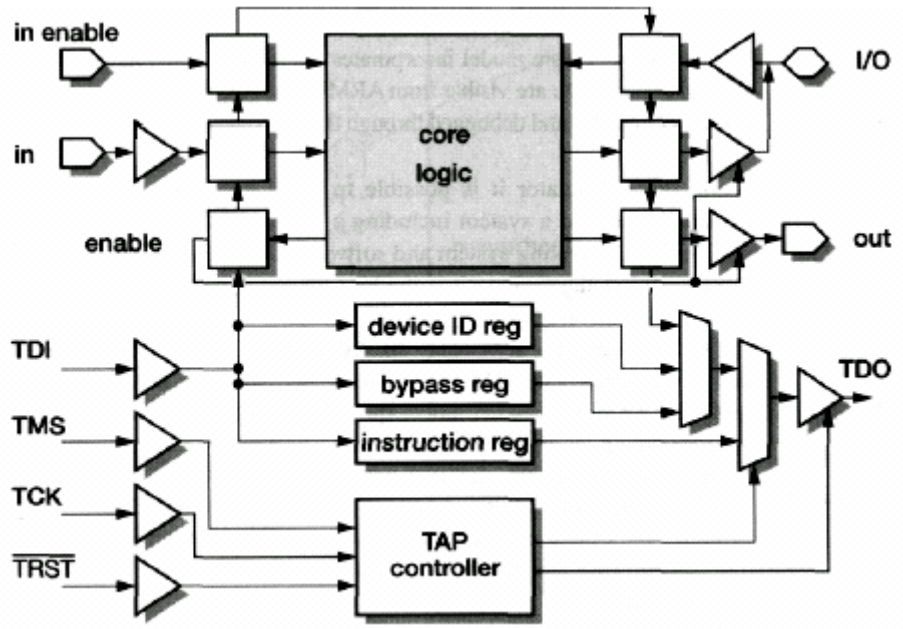


Figure IV.13 Schéma synoptique simplifié d'une interface JTAG.

IV.6.1 registres de données

Le comportement d'un chip particulier est déterminé par le contenu du registre d'instruction d'essai, qui peut choisir entre des différents registres de données :

- le registre d'identification (**device ID regiter**) lit à la sortie le numéro d'identification du composant qui est câblé dans le chip.
- Le registre **By-pass** connecte TDI À TDO avec un retard d'un cycle d'horloge pour donner l'accès rapide à l'appareil de contrôle d'un autre dispositif dans la boucle d'essai sur le même kit.
- Le registre **Boundary scan** intercepte tous les signaux entre la logique fondamentale et les pines et comprend les bits de registre individuel que l'on montre comme des rectangles connectés autour de la logique fondamentale dans la Figure IV.13.

Le registre BoundaryScan est le registre le plus grand et le plus important de l'interface JTAG. Ce registre peut représenter des entrées et des sorties du composant, mais peut également y placer des données en vue, ultérieurement, de les appliquer à l'entrée de la puce, ce qui se traduit par l'application aux sorties d'un signal que l'on aura défini soi-même. Le registre BoundaryScan est constitué d'un ensemble de cellules de mémoire indépendantes associé à une logique servant au pilotage des dites cellules et de la sortie.

IV.6.2 Registre d'instruction

Le registre d'instruction permet à l'utilisateur de sélecter l'une des différentes possibilités de l'interface JTAG. Outre ses fonctions standard, ce registre doit également permettre de sélecter des possibilités spécifiques au fabricant. Les spécifications JTAG ne mentionnent malheureusement pas quels sont les codes binaires à charger dans le registre d'instruction pour la sélection de l'une ou l'autre fonction. Une même fonction peut, de ce fait, avoir un code opérationnel (opcode) différent en fonction du composant concerné. En règle générale, ces informations sont données dans la fiche de caractéristiques du composant en question. Le tableau IV .1 récapitule toutes les fonctions définies par les spécifications JTAG. Seules les 3 premières instructions sont impérativement requises pour que le composant soit compatible au JTAG, sachant que l'on pourra implémenter, ou non, le reste des fonctions. Le fabricant a liberté, outre les fonctions énumérées dans le tableau 1, d'ajouter d'autres fonctions non décrites par les spécifications JTAG, ce qui ne fait qu'accroître la flexibilité de l'interface JTAG.

Tableau IV .1. Instructions de l'interface JTAG [12]

Instruction	Chip-mode	Sélecte le registre	Application
- EXTEST (impératif)	TEST	Registre BoundaryScan	Mesure et génération de signaux externes.
-SAMPLE/ PRELOAD (impératif)	NORMAL	Registre BoundaryScan	Lecture et/ou écriture du registre BoundaryScan.
-BYPASS (impératif)	NORMAL	Registre Bypass	Réduction de la longueur totale de scan.
-INTEST (impératif)	TEST	Registre BoundaryScan	Test interne du composant. Le registre BSc est relié directement à toutes les entrées et sorties du vrai composant.
-RUNBIST (optionnel)	TEST	–	Exécution d'un test autonome intégré (Build In Autotest) du vrai Composant.
-IDCODE (optionnel)	NORMAL	Device-ID	Lecture du code d'identification (ID-code) du composant.
-USERCODE (optionnel)	NORMAL	Registre UserCode	Programmation de PLD, DeviceID étendu.
-CLAMP (optionnel)	TEST	Registre Bypass	Placer des niveaux de sortie définis à l'entrée réduire la longueur totale de scan.
-HIGH Z (optionnel)	TEST	Registre Bypass	Toutes les sorties du composant sont mises à l'état de haute impédance Le composant se trouve, électriquement, découplé de la platine.

- **EXTEST**

En mode EXTEST toutes les broches du composant sont reliées au BSC. Il est possible à l'utilisateur, dans ce mode, de générer ses propres signaux et de les lire en vue, par exemple, tester les interconnexions entre circuits intégrés lorsque l'on se trouve en présence de plusieurs composants faisant partie d'une chaîne JTAG. Il est également possible de commander directement, le cas échéant, certains circuits intégrés périphériques. En règle générale, ce mode est utilisé par les fabricants de circuits imprimés pour vérifier l'absence, sur leurs produits finis, de micro-coupures ou de court-circuit.

- **SAMPLE/PRELOAD**

Ce mode a pour fonction de permettre une lecture du registre BoundaryScan ou de charger, dès le départ, un patron de test avant d'exécuter la fonction EXTEST.

- **BYPASS**

Il est judicieux, lorsque l'on se trouve en présence d'une série de circuits intégrés montés en cascade (chaînes JTAG) mais que l'on n'a besoin, à un moment donné, que des fonctions JTAG d'un unique composant, de donner aux autres composants l'instruction BYPASS. Dans ce mode, la puce fonctionne normalement, mais l'interface JTAG sélecte alors le registre BYPASS, registre dont la taille est limitée à 1 bit et qui ne pilote pas d'autre fonction. Cette approche permet de réduire la longueur de l'ensemble du registre à décalage. Les registres BoundaryScan peuvent comporter un nombre infini de bits (en pratique, ce type de registre comporte souvent plusieurs centaines de bits). Par la sélection du registre BYPASS, l'interface JTAG ne devra envoyer qu'un nombre sensiblement moindre de bits par instruction, ce qui se ressent au niveau de la vitesse de test.

- **INTEST**

Lors d'une opération INTEST, la puce proprement dite est reliée directement aux BSC. Ceci permet de vérifier le fonctionnement, on parle de fonctionnalité, du composant. Il est possible de paramétrer le registre BoundaryScan en y introduisant des états d'entrée définis pour ensuite procéder à une lecture des sorties de la puce. Cette option s'avère tout particulièrement précieuse lors du test de prototypes à base de logique programmable mais est également très intéressante lors d'opérations de remises en état.

- **RUNBIST**

Cette fonction permet de procéder à un test interne de la puce elle-même. Le résultat de cette opération (chip OK/chip not OK) se trouve, une fois cette instruction exécutée, sur la sortie TDO. Le domaine d'utilisation primaire de la fonction RUNBIST est celui des modules de mémoire. Si l'on voulait tester ces mémoires par l'approche standard au travers de l'interface JTAG il faudrait commencer par mettre une valeur donnée dans chacun des emplacements de mémoire, lire cette valeur, remettre dans chaque emplacement de mémoire l'inverse de cette valeur et relire le résultat de cette opération. Ces manipulations doivent se faire pour chaque emplacement et prennent énormément de temps (il ne faut pas oublier que tout cela se passe en série). Une instruction BIST constitue une option sensiblement plus rapide. DEVICE ID Il est possible, par le biais de ce mode, de lire, au travers de l'interface JTAG, le code d'identification (DEVICE-ID) de chaque composant. Ceci permet de vérifier que le composant concerné est bien celui que l'on voulait et d'en lire la version. Le principal « utilisateur » de cette fonction est le logiciel de programmation car il est possible de s'assurer que l'utilisateur a bien sélecté le composant correct.

- **USERCODE**

Ce mode sélecte un registre spécial que le fabricant pourra utiliser à son gré. On utilise le plus souvent ce registre pour y placer des informations de DEVICE-ID additionnelles.
Certains fabricants l'utilisent pour la programmation de composants PLD.

- **CLAMP**

CLAMP ressemble beaucoup à la fonction BYPASS, à la différence que le pilotage des connexions du composant se fait par le biais des BSC. Cela sous-entend que pendant la sélection du registre BYPASS les sorties du composant sont forcées à un niveau fixe prédéfini.

- **HIGH Z**

Cette option a pour fonction de permettre d'isoler électriquement le composant de l'extérieur. Toutes les sorties se trouvant alors en mode HIGH-Z (à haute impédance), de sorte que les mesures effectuées sur le reste des composants ne soit pas influencé par ce composant précis. D'autres fonctions décrites plus haut sont parfaitement définies dans les spécifications JTAG. Comme nous le disions en début d'article, les fabricants ont la liberté d'ajouter de nouvelles fonctions en vue d'augmenter, à leur gré, le potentiel de l'interface JTAG. Ce type d'extensions permet, par exemple, sur certains DSP, de placer des points d'arrêt matériel, de lire et/ou modifier des registres ou des emplacements de mémoire, voire encore de demander l'exécution d'instructions.

IV.6.3 Le contrôleur TAP

Le coeur opérationnel de l'interface JTAG est ce que l'on appelle le contrôleur TAP. (**T**est **A**ccess **P**ort) En collaboration avec le registre d'instruction et le décodeur d'instruction correspondant, le contrôleur TAP se charge du pilotage de l'ensemble de l'interface JTAG. Ce contrôleur est ce que l'on nomme une machine d'état (*state machine*) ne comportant qu'une unique entrée, à savoir l'entrée TMS. Le signal TMS détermine le nouvel état à prendre par le contrôleur. Ce dernier change d'état (state) sur le franc montant du signal TCK. On a besoin de savoir, pour pouvoir commander le contrôleur TAP, dans lequel des 16 états ce dernier se trouve. Il n'est malheureusement pas possible de lire directement cet état mais une petite astuce permet de mettre le contrôleur TAP à l'état de réinitialisation (reset) et ce quelque soit l'état dans lequel il se trouvait. La ligne TMS n'atteint le niveau logique haut dans La structure du diagramme d'état qu'après application de 5 impulsions d'horloge alors, le contrôleur se retrouve dans tous les cas de figure dans l'état « Test Logic Reset ». Un examen de la figure IV.14 permet de le constater aisément : choisissez un état quelconque. À partir de ce fait aller, par le biais de la flèche marquée d'un « 1 », jusqu'à l'état suivant. Reprendre ce processus encore 4 fois. Quel que soit l'état de départ, l'état final sera un « Test Logic Reset ». Certains composants dotés d'un contrôleur JTAG possèdent une broche additionnelle, TRST. Dès que l'on force cette ligne au niveau bas le contrôleur TAP passera automatiquement à l'état d'initialisation (« Test Logic Reset »).

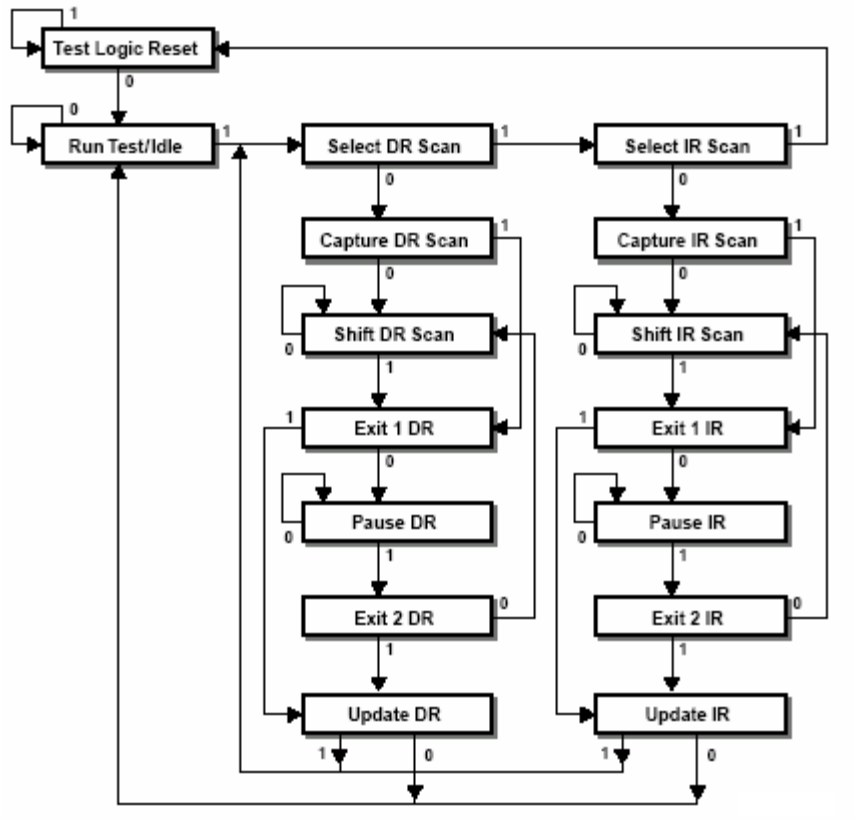


Figure IV.14 Diagramme de transition d'état du contrôleur TAP

Comme nous l'avons exposé plus haut, c'est le contenu du registre d'instruction qui détermine la fonction à remplir par l'interface JTAG. Cela implique de charger un code opératoire (op-code) donné dans le registre d'instruction, le dit code doit correspondre à la fonction que nous désirerions que remplisse l'interface JTAG. Le chargement du registre d'instruction requiert la mise à l'état « Shift-IR »du contrôleur TAP. Le trajet vers cet état passe inévitablement par l'état « Capture-IR ». Dans ce dernier état une valeur donnée définie par le composant lui-même est mise dans le registre d'instruction. En principe, la valeur n'a pas d'importance. Dès que le contrôleur TAP se trouve à l'état « Shift-IR », la sortie TDO s'active. Elle présentera alors le bit de poids faible (LSB) du registre d'instruction. La donnée présente sur l'entrée TDI se décale dans le registre d'instruction à chaque nouveau flanc montant, le contenu du registre d'instruction est décalé d'un cran vers la sortie lors de chaque nouveau flanc descendant. Après le remplissage du registre d'instruction il nous faut indiquer, sans ambiguïté, à l'interface JTAG que ce nouveau contenu doit être activé, ce qui est obtenu dans le cadre d'un état « Update-IR ». Dans cet état le contenu du registre d'instruction est reconnu comme étant l'instruction actuelle. À partir de cette instruction on détermine également lequel des autres registres sera sélectionné, en fonction du tableau 2, en tant que registre de données. La lecture et l'écriture d'un registre de données se fait de la même façon que dans le cas d'un registre d'instruction à la différence près que nous allons utiliser alors l'état « Capture- DR » pour doter le registre de données d'un contenu tenant la route (valide). On utilise alors « Shift-IR » pour la lecture et l'écriture de données et « Update-DR » pour actualiser les données que l'on vient tout juste d'écrire.

IV.6.4 Fichiers BSDL

On trouve, dans le cadre des spécifications JTAG, ce que l'on appelle des fichiers BSDL. Il s'agit de fichiers comportant toutes les informations importantes dans le cas de composants disposant d'une interface JTAG. On y trouve, par exemple, les codes opératoires des instructions JTAG à l'intention d'un type de composant donné. On y trouve également mentionnées les dénominations des broches d'un circuit intégré et l'ordre dans lequel celles-ci sont mentionnées dans le registre BoundaryScan, sans oublier bien entendu le code DEVICE-ID du composant correspondant. La structure des fichiers BSDL est similaire à celle des fichiers VHDL. Il est, de ce fait, facile de les réutiliser dans du VHDL l'information sera fournie par des fichiers BSDL. La plupart des programmes destinés à travailler avec l'interface JTAG sont en mesure d'interpréter les données contenues par un fichier BSDL. Vu qu'il s'agit de fichiers de type ASCII, n'importe quel programme de traitement de texte permet leur examen. Les instructions, le DEVICEID, les dénominations des broches, et le reste, sont clairement répartis dans des blocs distincts.

IV.7 LES UNITES CENTRALES INTEGRES DU PROCESSEUR ARM

Les CPU à base du ARM intègrent un coeur entier de processeur ARM, une cache (s), une unité MMU (s) et évidemment une interface AMBA dans une seule macrocellule. Dans cette section on va présenter en détail l'unité CPU ARM710T et on va citer quelques types de ces unités.

IV.7.1 ARM710T

L'ARM710T, ARM720T et ARM740T sont basés sur le coeur de processeur ARM7TDMI, auquel une cache mixte d'instruction et de données de 8 KO a été ajoutée. La mémoire et les périphériques externes ont accès via une unité maître de bus AMBA. Une mémoire tampon d'écriture, une unité de gestion de mémoire pour les unités (ARM71OT et 720T) ou une unité de protection de mémoire CP15 pour l'unité (ARM740T) sont incorporées. L'organisation de l'ARM710T est illustrée dans la figure IV.18.

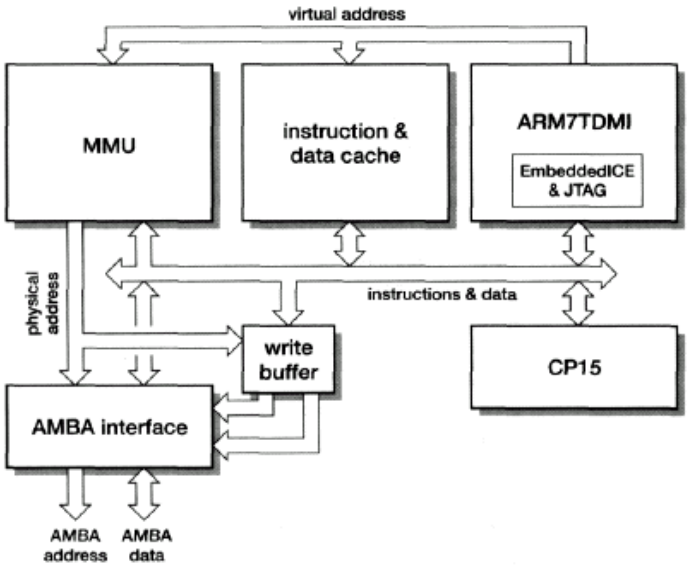


Figure IV.18 Organisation interne du CPU ARM710T

Dans le tableau IV.2. [1] sont mentionnés Les caractéristiques de ARM710T implémenté avec le processus CMOS 0.35 µm.

Tableau IV.2. Les caractéristiques du ARM710T [4], [6]

Processus	**0.35 µm**	**Transistors**	**N/A**	**MIPS**	**53**
Couche de métal	**3**	**surface**	**11.7mm^2**	**puissance**	**240 mW**
Vdd	**3.3V**	**Horloge**	**0-59 MHz**	**MIPS/W**	**220**

IV.7.2 ARM810

ARM810 est unité de CPU comprend le cœur entier ARM8 , une unité de gestion de la mémoire MMU,une cache de 8 KO et une mémoire tampon pour permettre au processeur de continuer son travail pendant une écriture d'un mémoire externe.
Les caractéristiques de ARM710T, implémenté sur le processus CMOS 0.35 µm, sont inscrites sur le tableau IV.3. [1]

Tableau IV.3. Les caractéristiques du ARM810. [4], [6]

Processus	**0.5 µm**	**Transistors**	**836022**	**MIPS**	**86**
Couche de métal	**3**	**surface**	**76mm^2**	**puissance**	**500 mW**
Vdd	**3.3V**	**Horloge**	**0-72 MHz**	**MIPS/W**	**172**

On trouve aussi parmi ces types de CPU à base d'un processeur ARM entier :

- **StrongARM SA-110** à 5 étages de pipeline.
- **ARM920T et ARM940T** qui sont à base du cœur entier ARM9TDMI plus une cache de données et des instructions, MMU, (ARM920T), une unité de protection (ARM940T) et un interface AMBA
- **ARM946E - S et ARM966E - S** sont des unités CPU synthétisable à base de cœur entier synthétisable ARM9E-S, les unité ont un interface AHB de AMBA.

IV.8 Application

Notre application est un modèle VHDL très simplifié mis en œuvre d'un processeur « open source » de 32 bits conformément à l'architecture ARM à 5 étages de pipeline. Il est conçu pour des applications incorporées sur chip avec les particularités suivantes : cache d'instructions et de données séparées, RAM locale, interface PCI, port I/O de 16 bits, contrôleur de mémoire, la possibilité d'ajouter de nouveaux modules utilisant les bus AMBA APB sur chip. Ce modèle est synthétisable et peut être mis œuvre sur FPGAS ou ASICS selon l'outil de synthèse, la simulation peut être faite avec Un simple simulateur VHDL. La figure IV.19 présente la structure interne de ce modèle.

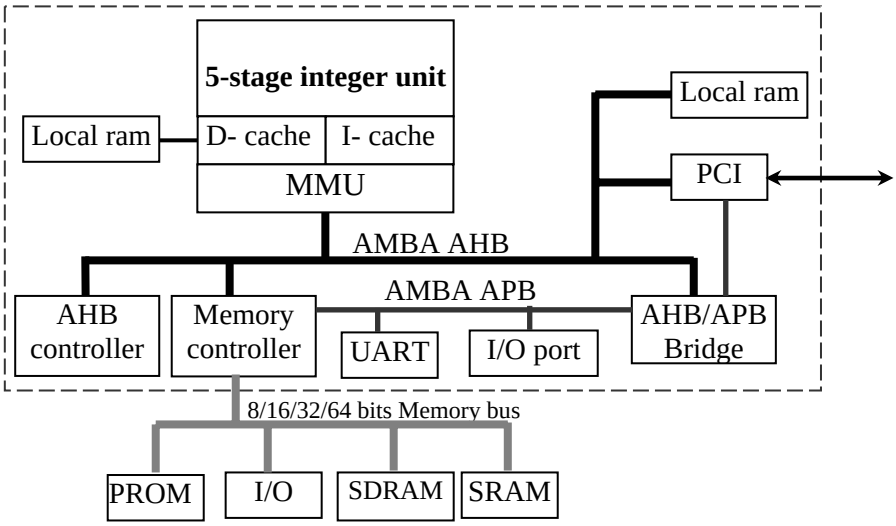


Figure IV.19 Organisation interne du modèle.

- **L'unité entière** l'unité entière est un modèle à 5 étages de pipeline, dont le nombre de registres est configurable de 8 à 32 registres selon les besoins d'application, (8 registres par défaut), une cache d'instructions et données séparées.

- **L'interface de mémoire** L'interface flexible de mémoire fournit une interface directe de la PROM, de dispositifs d'entrée-sortie d'une carte mémoire, de la RAM statique (SRAM) et de la mémoire dynamique synchrone (SDRAM).

- **Le port d'entrée-sortie parallèle :** On fournit un port parallèle d'entrée-sortie de 32 bits.

- **Les bus AMBA sur chip** Le processeur a une pleine mise en oeuvre des bus AMBA AHB et APB sur-chip. Un arrangement de configuration flexible ce qui en fait simple d'ajouter des nouveaux coeurs IP (*Intellectual Property*).

- **L'interface PCI :** l'interface PCI supportant la configuration de cible-uniquement (Target-only) a une basse complexité, peut être employée pour la mise au point de notre model sur le bus PCI.

- **La ram sur chip :** une petite ram sur chip peut être attaché au bus AHB, fournissant une mémoire locale rapide de taille configurable de 1 - 64 KO.

- **L'adressage du système:**

Champs des adresses	maximum taille	périphérique
0×00000000-0×1FFFFFFF 0×20000000-0×3FFFFFFF 0×40000000-0×7FFFFFFF	512 M 512 M 1G	PROM I/O de bus mémoire SRAM et/ou SDRAM
0×80000000-0×8FFFFFFF	256M	Registres sur chip
0×90000000-0×9FFFFFFF	256M	Unité de déboguage

IV.8.2 la configuration du modèle

Le modèle est configurable pour permettre des différentes tailles du cache, la performance de multiplicateur, la génération d'horloge et des technologies cibles. Le modèle est configuré par quelques records de constant, chacun configure un module / fonction spécifique. La déclaration des types de record de configuration est dans le paquetage TARGET.vhd, tandis que la spécification des records de configuration est définie dans le paquetage de DEVICE.vhd.

IV.8.2.a La configuration pour la synthèse :

Le sous-record de configuration de synthèse est employé pour adapter le modèle aux différents outils de synthèse et des bibliothèques cibles :

```
type targettechs is
( virtex, virtex2, atc35, atc25, atc18);
-- synthesis configuration
type syn_config_type is record
targettech: targettechs;
infer_ram : boolean;-- infer cache and dsu ram automatically
infer_regf : boolean;-- infer regfile automatically
infer_rom: boolean;-- infer boot prom automatically
infer_pads: boolean;-- infer pads automatically
infer_mult: boolean;-- infer multiplier automatically
rftype : integer;-- regfile implementation option
targetclk : targettechs; -- use technology specific clock generation
clk_mul : integer;-- PLL clock multiply factor
clk_div : integer;-- PLL clock divide factor
pci_dll : boolean;-- Re-synchronize PCI clock using a DLL
end record;type targettechs is (virtex, virtex2, atc35, atc25, atc18);
```

Selon l'outil de synthèse et la technologie cible, les méga-cellules (RAM, ROM, Pads) dépendant à la technologie peuvent être automatiquement déduits ou directement déclarés.

IV.8.2.b La configuration de l'unité entière

Le record de configuration d'unité entière définit la mise en oeuvre de l'unité entière :

```
-- processor configuration --
type multypes is (none, iterative, m32x8, m16x16, m32x16, m32x32);
type iu_config_type is record
nwindows: integer;-- # register windows (2 - 32)
multiplier: multypes;-- multiplier type
mulpipe: boolean; -- multiplier pipeline registers
mac : boolean; -- multiply/accumulate
fpuen: integer range 0 to 1;-- FPU enable
cpen: boolean; -- co-processor enable
fastjump : boolean;-- enable fast jump address generation
icchold : boolean;-- enable fast branch logic
lddelay: integer range 1 to 2; -- # load delay cycles (1-2)
fastdecode : boolean;-- optimise instruction decoding (FPGA only)
rflowpow : boolean;-- disable regfile when not accessed
watchpoints: integer range 0 to 4; -- # hardware watchpoints (0-4)
end record;
```

IV.8.2.c La configuration du cache

La cache est configurée par le record de configuration de cache :

```
type cache_config_type is record
isets : integer range 1 to MAXSETS; -- # of sets in icache
isetsize: integer;-- I-cache size per set in Kbytes
ilinesize: integer;-- # words per I-cache line
ilock : integer;-- icache locking
ilram : boolean; -- local inst ram enable
ilramsize : integer;-- local inst ram size in kbytes
ilramaddr : integer;-- local inst ram start address (8 msb)
dsets : integer range 1 to MAXSETS; -- # of sets in dcache
dsetsize: integer;-- D-cache size per set in Kbytes
dlinesize: integer;-- # words per D-cache line
dreplace : cache_replace_type; -- icache replacement algorithm
dlock : integer;-- dcache locking
dsnoop : dsnoop_type;-- data-cache snooping
drfast : boolean;-- data-cache fast read-data generation
dwfast : boolean;-- data-cache fast write-data generation
dlram : boolean;-- local data ram enable
dlramsize : integer;-- local data ram size in kbytes
dlramaddr : integer;-- local data ram start address (8 msb)
end record;
```

IV.8.2.d La configuration de contrôleur de mémoire

```
type mctrl_config_type is record
bus8en : boolean; -- enable 8-bit bus operation
bus16en : boolean;-- enable 16-bit bus operation
wendfb : boolean; -- enable wen feed-back to data bus drivers
ramsel5 : boolean; -- enable 5th ram select
sdramen : boolean;-- enable sdram controller
sdinvclk : boolean;-- invert sdram clock
sdsepbus : boolean;-- enable separate SDRAM buses
end record;
```

IV.8.2.e La configuration pour le démarrage (BOOT) de modèle

Plus la procédure de BOOT standard de démarrage de l'adresse 0 dans la mémoire externe, le modèle peut être configuré pour démarrer d'une PROM interne ou de l'unité de déboguage, ceci est après l'implémentation sur une FPGA ou après la production du système comme un circuit intégré ASIC. Les options de BOOT sont définies dans le record de configuration de BOOT comme défini dans le paquetage TARGET :

```
 type boottype is (memory, prom, dual);
type boot_config_type is record
boot : boottype; -- select boot source
ramrws : integer range 0 to 3;-- ram read waitstates
ramwws : integer range 0 to 3;-- ram write waitstates
sysclk : integer;-- cpu clock
baud : positive;-- UART baud rate
extbaud : boolean;-- use external baud rate setting
pabits : positive;-- internal boot-prom address bits
end record;
```

IV.8 Simulation

Après la compilation du modèle VHDL, et pour vérifier la fonctionnalité du système on fournit un banc générique d'essai « tbench/tbgen.vhd ». Ce banc d'essai permet de produire un modèle d'un système avec différentes tailles et différents types de mémoire. Le fichier tbench/tbleon.vhd contient quelque configuration alternative employant le banc générique d'essai :

- **TB_FUNC8, TB_FUNC16, TB_FUNC32, TB_FUNC_SDRAM :** tests fonctionnels réalisants une vérification rapide de la plupart des fonctions sur-chip employant une RAM statique externe de 8-, 16-ou 32 bits, ou SDRAM de 32 bits.
- **TB_MEM :** Mise à l'épreuve de toute la mémoire sur-chip avec les modèles de 0x55 et 0xAA.

Une fois que le modèle est compilé, on utilise le banc d'essai de TB_FUNC32 pour vérifier le comportement du modèle. La simulation doit être commencée dans le répertoire d'adresse supérieure **«** top directory ». Le résultat de la simulation en premier lieu ou le module de testmod.vhd est connecté au secteur d'entrée-sortie et produit des messages de mise au point et des exceptions de bus pour le banc d'essai. Doit être semblable à :

```
# ARM generic testbench
#
# Testbench configuration:
# 32 kbyte 32-bit rom, 0-ws
# 2x128 kbyte 32-bit ram, 2x64 Mbyte SDRAM
#
# * starting ARM model system test*
# Register file
# Multiplier (SMUL/UMUL/MULSCC)
# Divider (SDIV/UDIV)
# Watchpoint registers
# Cache controllers
# Interrupt controller
# UARTs
# Timers, watchdog and power-down
# Parallel I/O port
# Test completed OK, halting with failure
# ** Failure: TEST COMPLETED OK, ending with FAILURE
```

La simulation sera interrompue en produisant un échec dans le code VHDL de module testmod.

IV.8.1 démarrage (Boot) du modèle

Notre système va démarrer à partir d'une PROM configurée de fonctionner de 8 bits, elle nécessite 4 cycles de lecture de 8 bits pour la lecture de 32 bits complet. La figure IV.20 présente l'interface avec une PROM et SRAM de 8 bits.

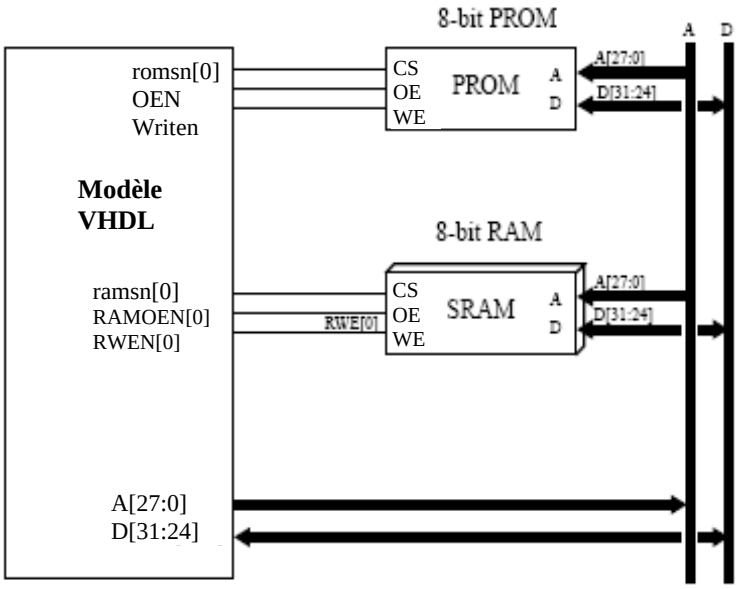
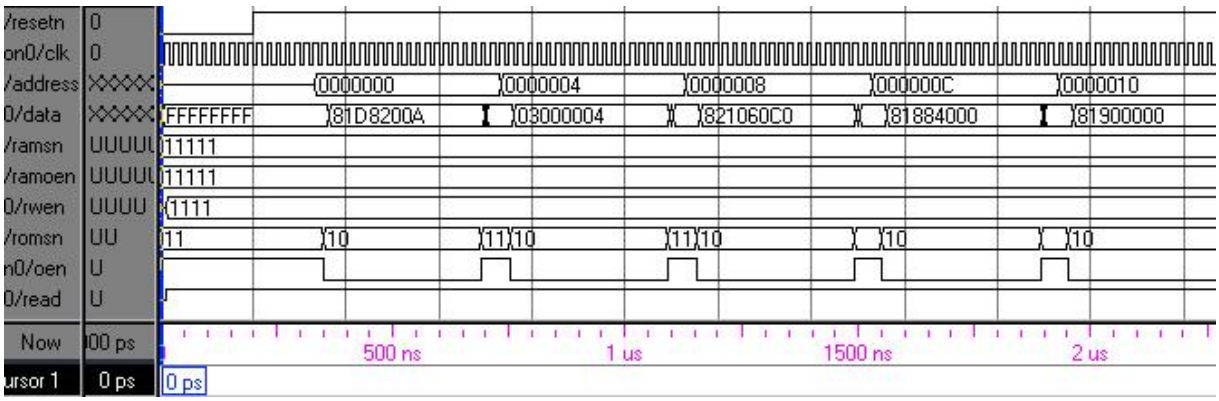


Figure IV.20 interface de mémoire de 8 bits.

Après le lancement de la simulation on visualise les différents signaux d'interface de mémoire d'une période de 250ns dans le chronogramme ci- dessous



Le contenu du PROM est inscrit dans le tableau si déçu :

```
0000 81d8200a 03000004 821060c0 81884000
0010 81900000 81980000 81800000 01000000
0020 0f200000 03002040 8210600f c221e014
0030 c201e008 82086100 85286001 82104002
0040 050003c0 82104002 c221e008 c201e000
0050 820863ff c221e000 82086300 c401e0a0
0060 8730a004 8608e003 82104003 07010200
0070 8210c001 c221e000 8728a004 8208e070
0080 8608a0c0 8730e004 8210c001 8730e002
0090 8210c001 86102800 8210c001 8088a003
00a0 12800003 01000000 82006200 07354e10
```

Dans le chronogramme précèdent on constate les signaux suivants :

- ✓ resetn : entré de mis en marche du système activé en bas après 10 période d'horloge.
- ✓ Clk : horloge du système.
- ✓ A[27:0] : adresse mémoire (sortie)
- ✓ D[31:24] : 8 bits de données de mémoire (bidirectionnel)
- ✓ ramsn[0] : sortie de sélection de SRAM.
- ✓ ramoen[0] : sortie de validation de la sortie de SRAM.
- ✓ rwen[0] : sortie de validation de l'écriture dans la SRAM.
- ✓ romsn[0] : sortie de sélection de PROM.
- ✓ oen : validation de sortie.

IV.9 CONCLUSION

La tendance dans la conception de système incorporée doit intégrer toutes les fonctions principales du système, sauf quelques composants de mémoire, dans un seul chip. Les bénéfices en termes de dépenses de composantes, la fiabilité et l'efficacité de la puissance sont considérables. Le développement technologique de processus de semi-conducteur permet, actuellement, d'implanter des millions de transistors sur un seul chip à moindre coût, des recherches en cours permettront, dans un proche avenir, d'intégrer des dizaines de millions de transistors sur un seul chip, le support architectural étudié dans ce chapitre nous permettre de concevoir un système à base du processeur ARM, en disposant d'outils de développement matériel et logiciel adéquats.

CONCLUSION GENERALE

La conception de système-sur-chip (SoC) est actionnée par unes unités de microprocesseur, des processeurs de traitement de signal numérique (DSPs), des coprocesseurs de fonction fixée, et d'autre module nécessaires pour achever l'application adéquate. Donc, une compréhension d'architecture de processeur fournit un contexte pour choisir le processeur (s) juste pour un SoC, en termes de performance, coût, puissance et d'autres contraintes.

La société ARM est couronnée de succès parce qu'il offre une large variété des coeurs de processeur qui rencontrent les exigences de beaucoup applications de SoC. Le processeur ARM a à l'origine construite sa célébrité sur la basse puissance comme une première fixation de pionnier des microprocesseurs de 32 bits dans des chips conçues pour l'électronique grand public. Certainement, comme l'architecture de ARM a grandi et des conceptions plus complexes ont été achevées pour fournir de performance plus haute, l'effort doit aussi augmenter pour rencontrer des niveaux de performance. Bien qu'il y ait probablement d'autres conceptions de microprocesseur avec une puissance réduite ou performance élevée comparée au processeur ARM, la société a continué à dominer le marché en fournissant un jeu plus large de conceptions, des outils et des alliances pour fournir beaucoup de façons pour les projets de conception qui se servir à la technologie de ARM et supportant les outils fournis par les associés de ARM. Ces associés emploient la propriété intellectuelle de ARM (IP) pour développer des chips et des produits pour le marché d'électronique. Le revenu de ARM vient de trois sources primaires :

1. Montants de la licence pour microprocesseurs et d'autres blocs IP.
2. Droits d'auteur pour l'utilisation des blocs IP du ARM.
3. Outils et kit pour supporter le développement et de déboguage des applications de ARM.

Les particularités principales de l'architecture de ARM sont :

- un grand jeu de registres, pour but d'éviter l'accès à la mémoire au maximum, et par conséquence augmenter la vitesse d'exécution.
- une architecture de chargement, rangement (load-store), pour but de simplifier le code de programme.
- 3-address des instructions (c'est-à-dire les deux registres d'opérande source et le registre résultat sont tout indépendamment spécifiés).
- exécution conditionnelle de chaque instruction.

- l'inclusion des instructions très puissante de chargement et rangement de multiples registres;
- La possibilité d'exécution d'une opération de décalage générale et une opération générale de l'ALU dans une seule instruction qui sera réalisée dans un seul cycle d'horloge.
- Une extension de jeu d'instruction ouverte pour le jeu d'instruction de coprocesseur, incluant l'addition de nouveaux registres et des types de données au modèle du programmeur.
- une représentation très dense de 16 bits compressé de jeu d'instruction l'architecture thumb. S'implique un code de programme moins spacieux.

En plus 'architecture ARM offre une support architectural, qui permettre de incorporer dans une seule chips un coeur ARM, un système de mémoire, des autres module personnalisés compatibles avec le processeur ARM utilisant une interconnexion standard de bus AMBA , fournis pour ce but.

Une autre particularité des coeurs de ARM que sont disponibles dans deux formes macrocellule dur *macro hard* et macrocellule logiciel *macro soft*.

- Une macrocellule dure est une conception qui a été prise entièrement à la mise en oeuvre sur le silicium. On le fournit à l'utilisateur comme un objet de disposition à être inclus avec le reste de la conception à l'étape de conception physique. Les coeurs durs sont entièrement optimisés par le constructeur. L'inconvénient d'un coeur dur est la portabilité entre les différents processus de silicium. Puisque le coeur dur est déjà remis à un processus de semi-conducteur spécifique, donc l'utilisateur est forcé à ce processus.
- Des coeurs soft sont livrés sou format RTL de Verilog . L'utilisateur est responsable de la synthèse et l'implémentation physique du cœur. Ces coeurs offrent plus de flexibilité pour l'utilisateur, lié au processus de silicium, mais ne peuvent pas fournir de performance comme un coeur dur. comme il existe aujourd'hui des outils EDA (*electronic design automation*) de langage mixte Verilog et VHDL capable de concevoir avec des processus de semi-conducteur avancés. Qui aide les designers de concevoir, et de vérifier ces propres applications à base de processeur ARM, ou bien de concevoir des modules compatibles avec le processeur ARM, c'est notre tendance vers le prochain travail.

REFERENCES BIBLIOGRAPHIQUES

[1] P. Nicholas. CARTER, « Architecture de l'ordinateur », EdiScience, 2002.

[2] C.Patrick, C.Armand , « Le microprocesseur PENTIUM, architecture et programmation », DUNOD, Pari, 1994

[3] P.Zanella, Y.ligier, « Architecture et technologie des ordinateurs », 3e édition, DUNOD, Pari, 1998.

[4] S. Furber, « ARM system-on-chip architecture, second edition Addison-Wesley, 2000.

[5] L.Henri, « Microprocesseurs », DUNOD, Paris, 1995.

[6] Datasheets des processeur ARM, http://www.arm.com

[7] J. Hennessey and D. Patterson, Computer Architecture And Quantitative Approach, 2nd ed., Morgan Kaufmann, San Mateo, Calif., 1996.

[8] L.Henri , René-V. Honorat, «Power PC, initiation pratique», DUNOD, Paris, 1995.

.

[9] A. Alfred, U.Jeffrey, « Concepts fondamentaux de l'informatique », DUNOD, Paris, 1993.

[10] I. Phillips and T. Dent, "The Development and Debug of Microsystem Controllers," http://www.isdmag.com/ic_uP/ Development.html.

[11] D. Bursky, "Combo RISC CPU and DRAM Solves Data Bandwidth Issues," Electronic Design, Mar. 4, 1996.

[12] ARM, "AMBA 2.0 specification", http://www.arm.com

[13] C. Tavernier, « Circuits logiques programmables », DUNOD, Paris, 1996.

[14] D. Patterson [1985]. "Reduced Instruction Set Computers", Communications of the ACM, vol. 28, no. 1, Jan. 1985.

[15] [7] Flynn, D., "AMBA: Enabling Reuseable On-Chip Designs", IEEE Micro, 17 (4), July/August 1997, pp 20-27

[16] Jagger, D., "ARM Architecture Reference Manual", Prentice Hall, 1996. ISBN 0-13-736299-4

[17] M. Katevenis [1983]. Reduced Instruction Set Computer Architectures for VLSI, Ph.D. dissertation, Computer Science Div., Univ. of California, Berkeley, 1983. Also published by M.I.T. Press, Cambridge, MA, 1985

[18] Segars, S., Clarke, K. and Goudge, L., "Embedded Control Problems, Thumb and the ARM7TDMI", IEEE Micro, 15

[19] D. Seal, ARM Architecture Reference Manual, 2nd Edition, Addison-Wesley, 2000.

[20] David C. Black and Jack Donovan, " SYSTEMC: FROM THE GROUND UP" Eklectic Ally, Inc. ©2004 Kluwer Academic Publishers New York, Boston, Dordrecht, London, Moscow

[21] R. Jacob Baker, Harry W.li, Davide E.Boyce, IEEE Series on Microelectronic Systems, Stuar K. Tewksbury, Serie Editor.

[22] Ron Mancini, "Application Report Understanding Basic Analog – Active Devices" SLOA026A, Texas instrument, April 2000

[23] Behzad Razavi, "Design of CMOS Analog Circuits", Prentice Hall, 1998.

[24] R.Gray, G. Meyer, "Analysis and design of analog integrated circuits", fourth Edition, John Wiey & Sons, 2000.

[25] Karen Parnell & Nick Mehta, "Programmable Logic Design, Quick Start Hand Book" Second Edition, © Xilinx, January 2002.

[26] J.Pekka Soininen, "Architecture design methods for application domain-specific integrated computer systems", university of oulu, linnanma, 2004.

[27] Sachin Tandon, "A Programmable Architecture for Real-time Derivative Trading", Master of Science, Computer Science School of Informatics, University of Edinburgh 2003

[28] Filip Sebek, “Cache Memories and Real-Time Systems”, MRTC Technical Report 01/37, Department of Computer Engineering, Mälardalen University, Västerås, Sweden, Presented 2001-09-28, revision III

[29] S.Gadd & T.Lenart, “A Hardware Accelerated MP3 Decoder with Bluetooth Streaming Capabilities”, Master of Science Thesis, in cooperation with, C Technologies AB, November 2001

[30] F.Vahid & T.Givargis “ Embedded system design a unified hardware/software introduction”,John Wiley & Sons, inc, 2002.