heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

heig-vd

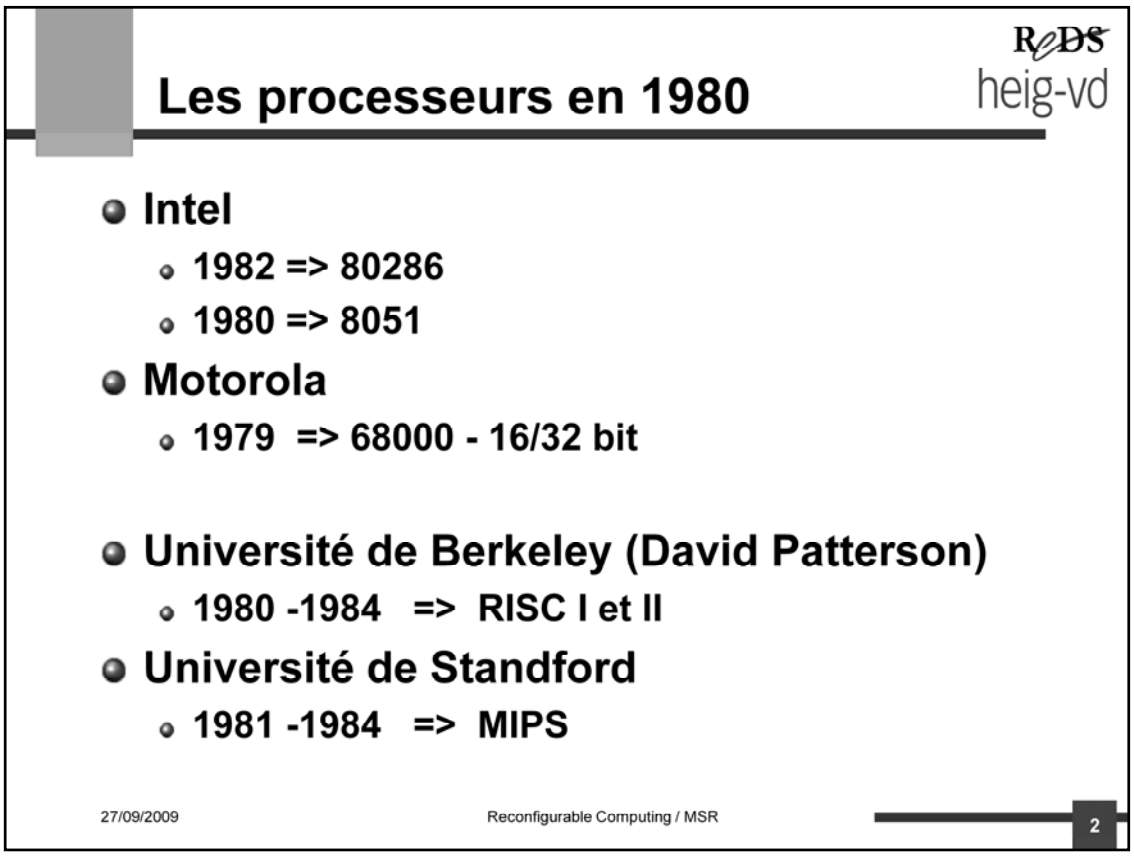
Reconfigurable & Embedded Digital Systems

Master MSE

Module d'aprofondissement ReCO
Reconfigurable Computing

ARCHITECTURES DES PROCESSEURS ARM

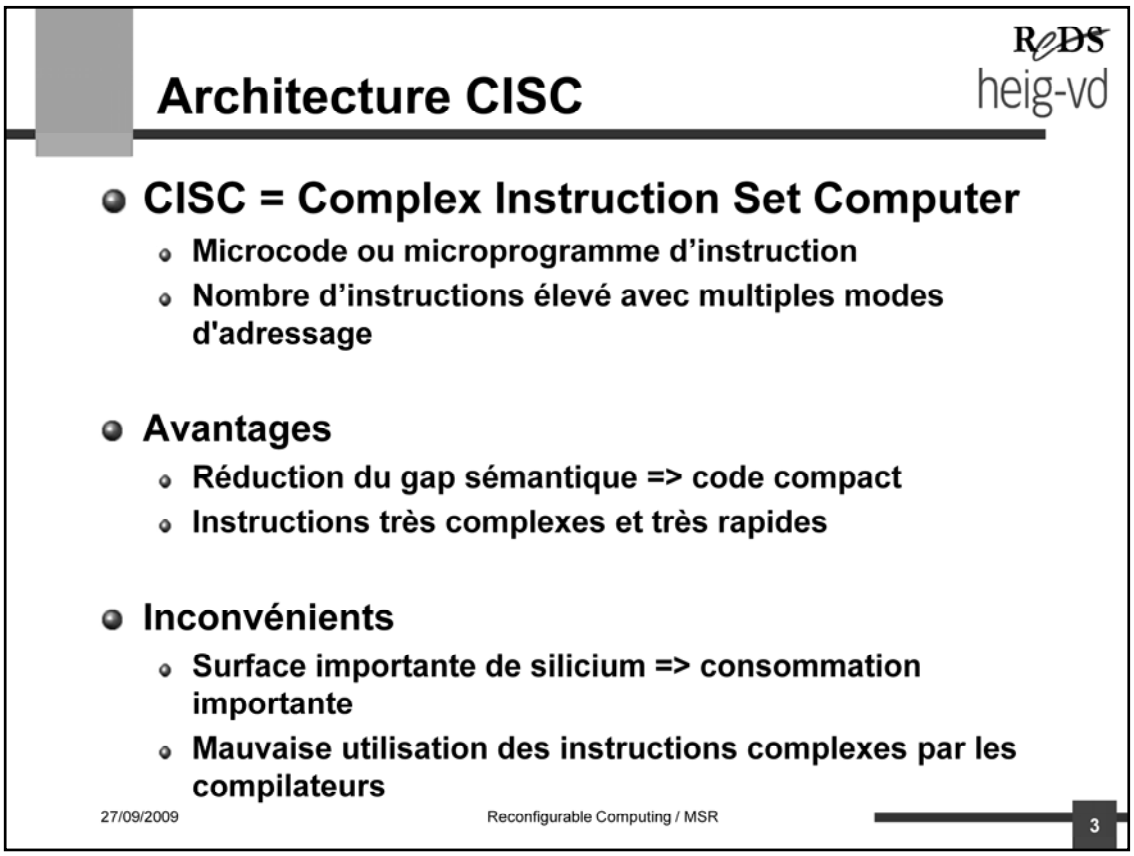
Michel Starkier – V1.0



Dans les années 80, la famille Intel X86 commençait à s'étoffer avec le micro-processeur 80286 introduit sur le marché en 1982 et qui comprenait 134 000 transistors – ce qui était énorme à l'époque. En 1985, le 80386 (275 000 transistors) passait en production.

Un autre micro processeur a eu un grand succès à la même époque pour des applications industrielles, le Motorola 68000, un processeur 16/32 bits développé par Motorola. Ce processeur a initialement équipé les cartes processeurs au standard VME (bus et format de carte introduit par Motorola).

Ces deux types de processeur était considéré (à posteriori) comme des processeurs de type CISC. Un projet financé par l'agence de recherche militaire américaine ARPA été lancé au début des années 80. Il s'agissait d'explorer de nouvelles architectures de processeurs. Dans le cadre de ce projet, deux équipes universitaire ont travaillé dans des directions parallèles. Le premier de ces projets était le projet RISC de l'université de Berkeley (1980 -1984) , dirigé par David Patterson. Son design RISC a été ensuite commercialisé sous le nom de SPARC. Le second de ces projets était le projet MIPS de l'université de Standford (1981-1984), dirigé par John L. Hennessy. Le design MIPS à été commercialisé par la société éponyme MIPS Computer System.



L'appellation CISC = Complex Instruction Set Computer (Ordinateur à jeu d'instructions complexes) a été donnée à certains types de processeur rétroactivement - après l'invention du RISC.

L'architecture des processeurs CISC comprend une mémoire de microcode ou microprogramme d'instruction. Une instruction pour CISC sera traitée en interne par l'exécution d'une séquence de micro-instructions. Le nombre d'instructions est élevé. Certaines instructions sont complexes, comportant plusieurs opérations exécutées par la séquence de micro-instructions.

Les instructions sont de taille variable (nombre d'octets). Le temps d'exécution est aussi variable (nombre de cycles).

De multiples modes d'adressage sont supportés (pour toutes les instructions).

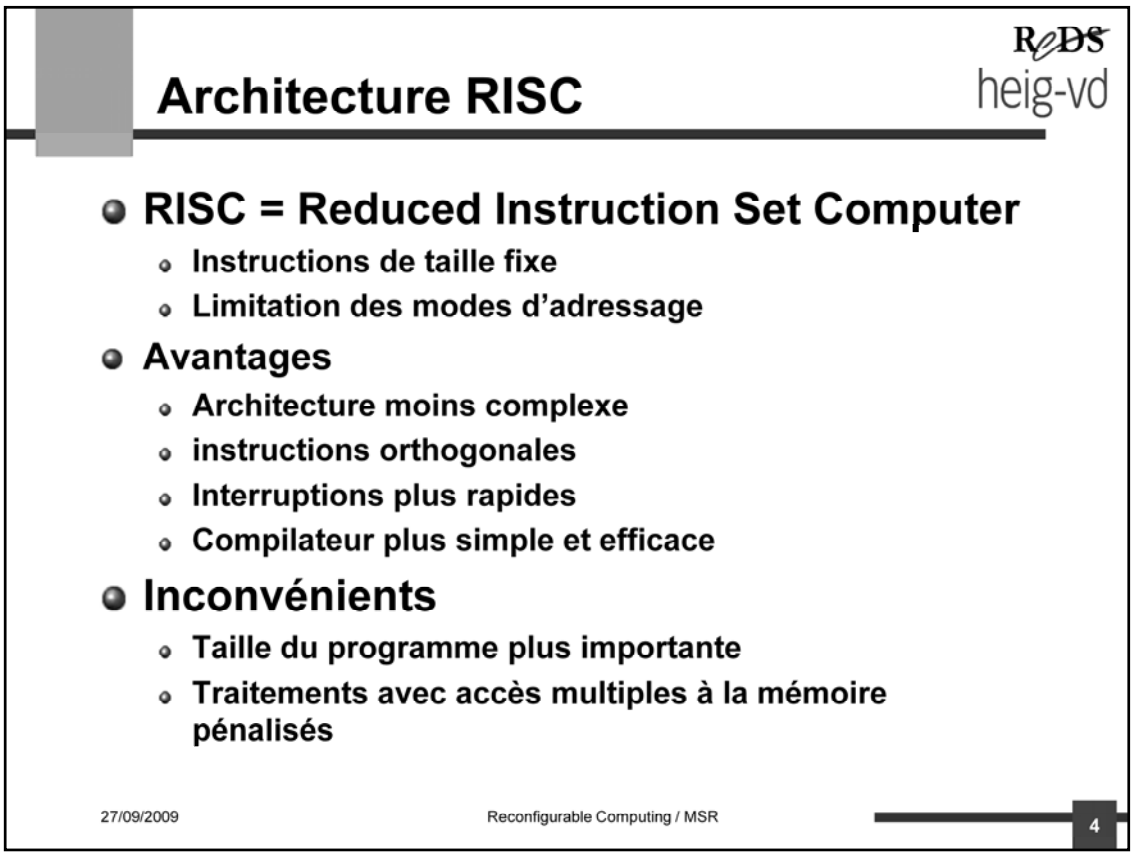
Avantages

L'avantage de l'architecture CISC est la réduction du gap sémantique entre le code de haut niveau et le langage machine. Il faut moins d'instructions pour réaliser une fonction de haut niveau. De ce fait, le code est dense, ce qui permet de réduire les tailles des mémoires d'instruction et des caches. La microprogrammation, permet une correction du jeu d'instructions sans modification de l'architecture du processeur.

Inconvénients

Le design du chip est très complexe et long, nécessite une surface importante de silicium, ce qui augmente la consommation. Les instructions complexes sont difficilement exploitées par les compilateurs.

Note : Semantic Gap :écart sémantique entre langage de haut niveau et langage machine (couvert par le compilateur.
Sémantique : étude du sens / lien entre signifiant et signifié



Les processeurs RISC = Reduced Instruction Set Computer (Ordinateur à jeu d'instruction réduit) ne comportent pas de microprogrammes. Ils font appel à un structure pipelinée la plus simple possible.

Les instructions sont de taille fixe et les modes d'adressage sont limités.

- Adressage par registres pour les opérations arithmétiques et logiques
- Adressage indirect pour les accès mémoire.

Les instructions sont la plupart du temps exécutées en un cycle d'horloge.

Avantages

Le nombre d'instruction est minimum. Le design est plus simple et plus rapide.

Le traitement peut être plus facilement accéléré en se concentrant sur une architecture pipelinée efficace. Le chip est plus petit et consomme moins. La fréquence d'horloge peut plus facilement être augmentée.

Les interruptions sont plus rapides, car le pipeline est plus court et moins complexe que celui des CISC.

Le compilateur est plus simple à implémenter et plus efficace.

Inconvénient

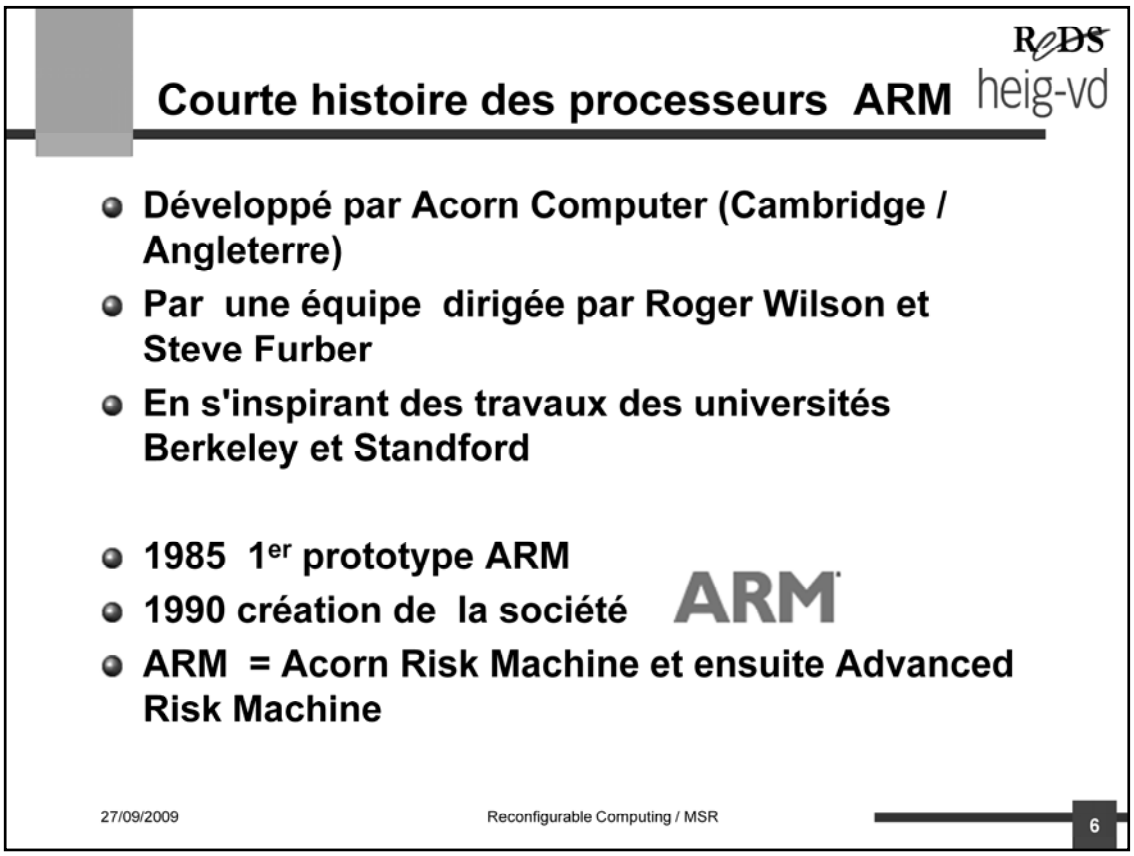
La taille du programme est plus importante.

Les instructions sont de taille fixe, ce qui n'est pas toujours optimisé.

Comme il n'est pas possible d'accéder à la mémoire par une instruction de traitement. Les traitements de données avec accès multiples à la mémoire (chaines, signal , ...) sont pénalisés.

Cours ReCo

VUE D'ENSEMBLE DES PROCESSEURS ARM



Acorn est une société anglaise située à Cambridge..

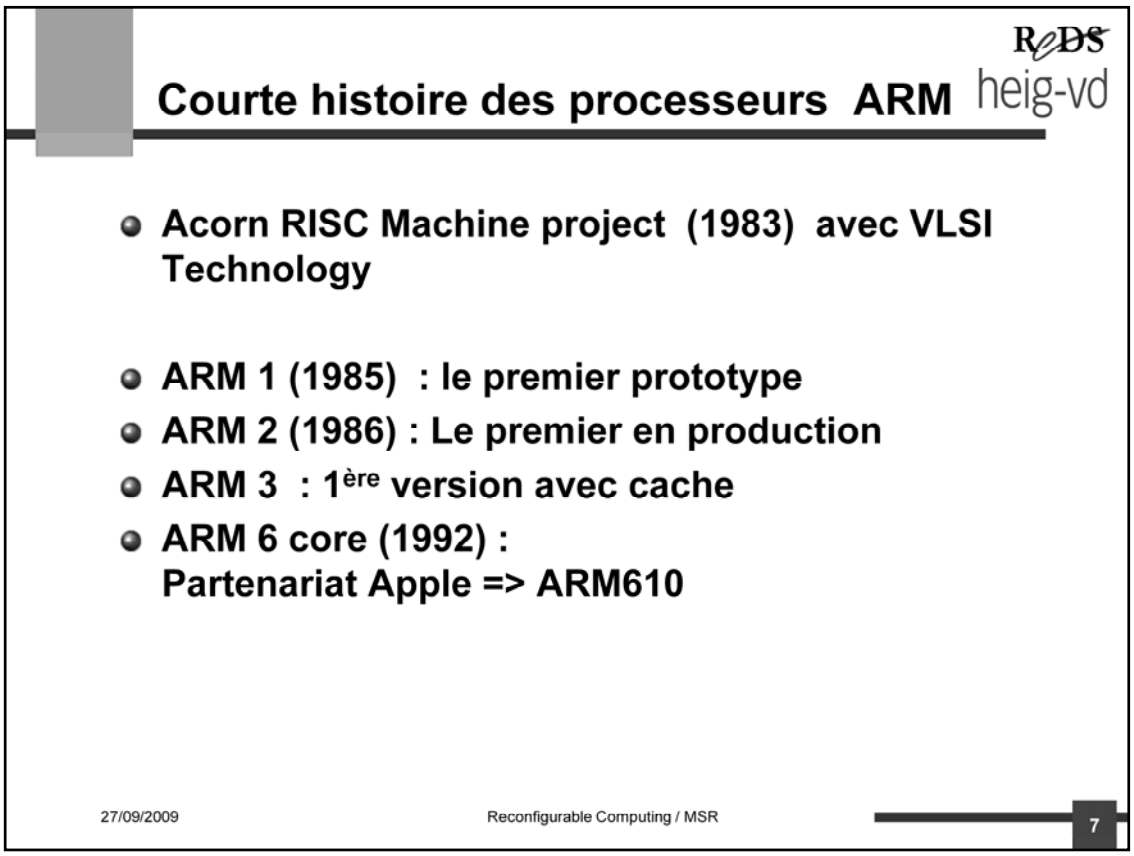
En 1985, Acorn développe le premier prototype du processeur ARM (destiné à un ordinateur personnel. l'Acorn RISC Machine) . L'ARM2 passe en production en 1986. L'ARM 2 ne comprenait que 30 000 transistors .

En 1980, Apple commence à travailler avec Acorn sur de nouvelles versions du processeur.

Apple utilise l'ARM610 pour différents produits dont le PDA Newton.

En 1990 apparait une nouvelle société nommée ARM qui signifie désormais **Advanced RISC Machine.**

The History of The ARM Architecture, *Markus Levy, Convergence Promotions.*



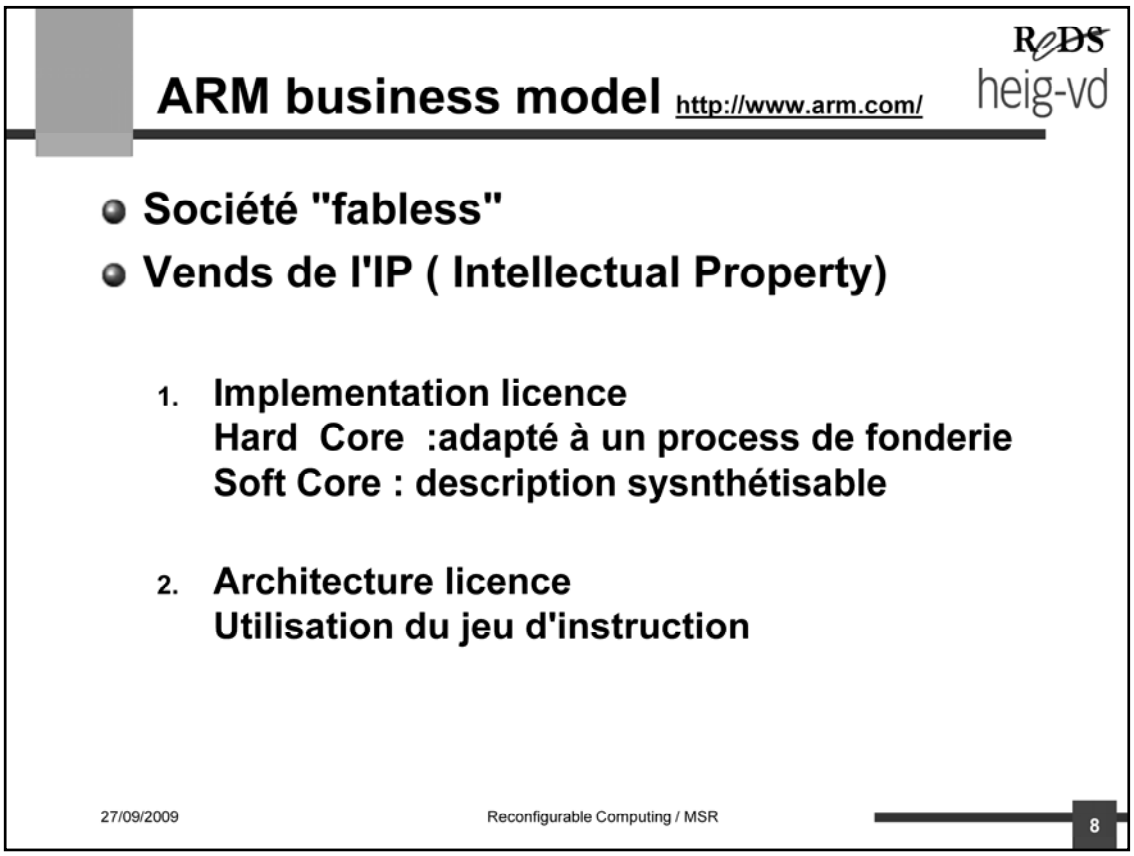
Le projet Acorn Risk Machine démarre en 1993 avec le partenaire VLSI Technology – fondeur des prototypes ARM.

L'ARM1 (1985) était le premier prototype de processeur ARM avec architecture RISC.

L' ARM2 (1986) a été le premier processeur en production. Il ne comprenait que 30 000 transistors . L' ARM2 avait un bus de donnée de 32 bits, et un bus d'adresse de 26 bits (espace d'adressage de 64 Mbyte). Perfomance : 4 MIPSs@8Mhz

L'ARM3 était une version améliorée de l'ARM2 avec une mémoire cache.

ARM 6 (1992) conçu à la demande d'Apple , n'avait que 35 000 transistors. Performance : 17MIPS@20MHz



"ARM is a a 32-bit RISC processor architecture currently being developed by the ARM corporation. The business model behind ARM is based on licensing the ARM architecture to companies that want to manufacture ARM-based CPU's or system-on-a-chip products. The two main types of licenses are the Implementation license and the Architecture license.

The *Implementation license* provides complete information required to design and manufacture integrated circuits containing an ARM processor core. ARM licenses two types of cores: soft cores and hard cores. A hard core is optimised for a specific manufacturing process, while a soft core can be used in any process but is less optimised.

The *Architecture license* enables the licensee to develop their own processors compliant with the ARM ISA.

ARM processors possess a unique combination of features that makes ARM the most popular embedded architecture today.

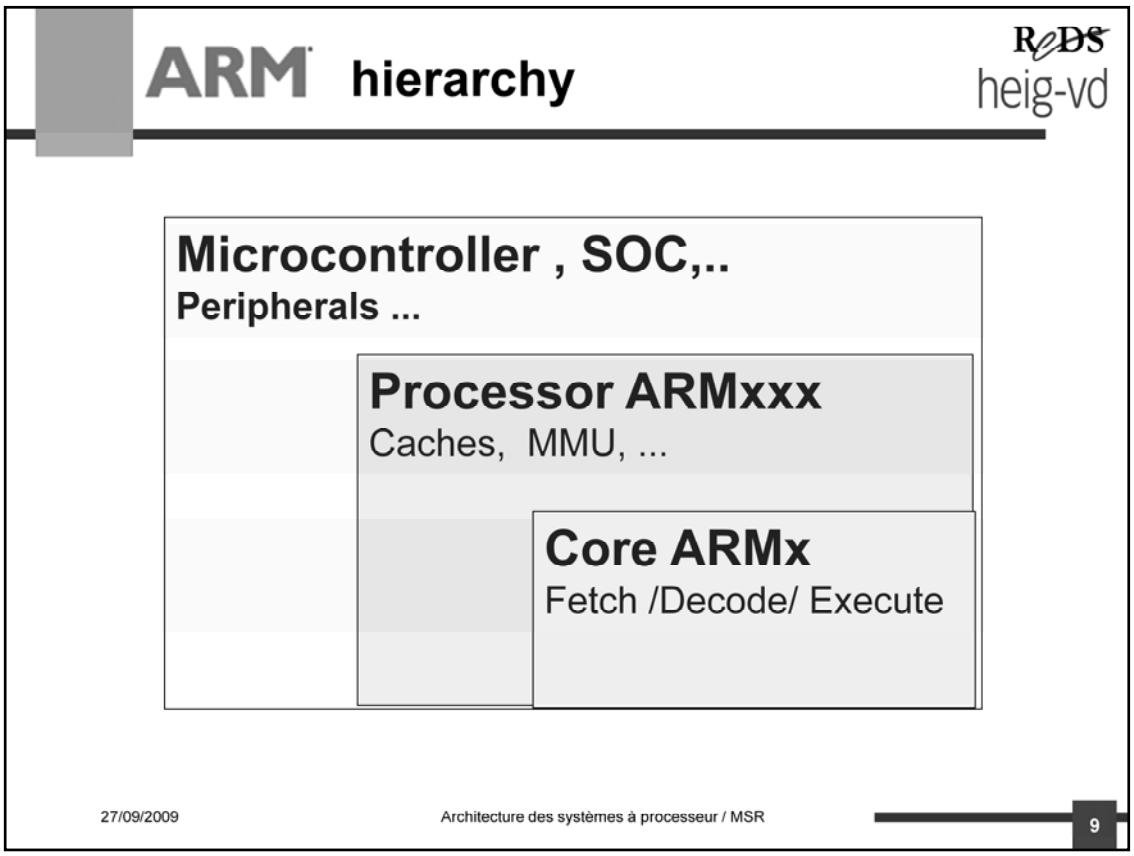
First, ARM cores are very simple compared to most other general-purpose processors, which means that they can be manufactured using a comparatively small number of transistors, leaving plenty of space on the chip for application-specific macrocells. A typical ARM chip can contain several peripheral controllers, a digital signal processor, and some amount of on-chip memory, along with an ARM core.

Second, both ARM ISA and pipeline design are aimed at minimising energy consumption — a critical requirement in mobile embedded systems.

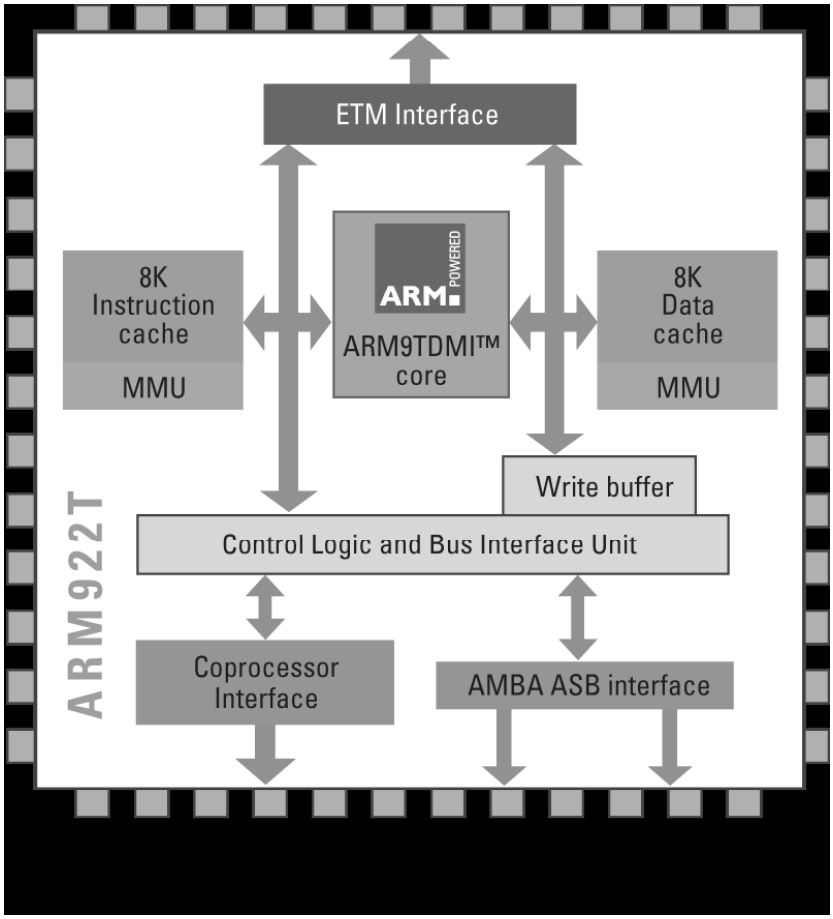
Third, the ARM architecture is highly modular: the only mandatory component of an ARM processor is the integer pipeline; all other components, including caches, MMU, floating point and other co-processors are optional, which gives a lot of flexibility in building application-specific ARM-based processors.

Finally, while being small and low-power, ARM processors provide high performance for embedded applications. For example, the PXA255 XScale processor running at 400MHz provides performance comparable to Pentium 2 at 300MHz, while using fifty times less energy."

The ARM Architecture, Leonid Ryzhyk, 2006

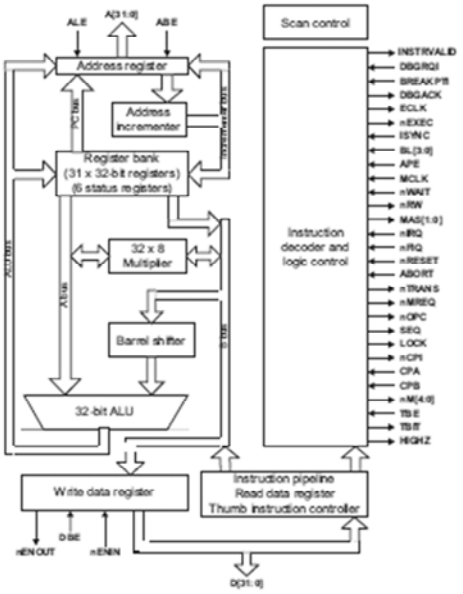


Exemple de l'ARM922T comprenant un coeur ARM9TDMI, des mémoires caches, un coprocesseur, et une interface bus AMBA.



Core ARM7TDMI

- 1993
- Pipeline à 3 niveaux
- Architecture Von Neuman
- 60MIPS@60MHz
- jeu d'instruction : V4T – 16/32 bits
- ARM710T, 720T, 740T,
- Plusieurs descriptions synthétisables



27/09/2009 Reconfigurable Computing / MSR 10

Processeurs de la famille ARM7

ARM7TDMI(-S) 3-stage pipeline 15 MIPS @ 16.8 MHz
ARM710T MMU 36 MIPS @ 40 MHz
ARM720T 8KB unified,MMU 60 MIPS @ 59.8 MHz
ARM740T MPU
ARM7EJ-S Jazelle DBX

Utilisation de l'ARM7 dans des produits industriels

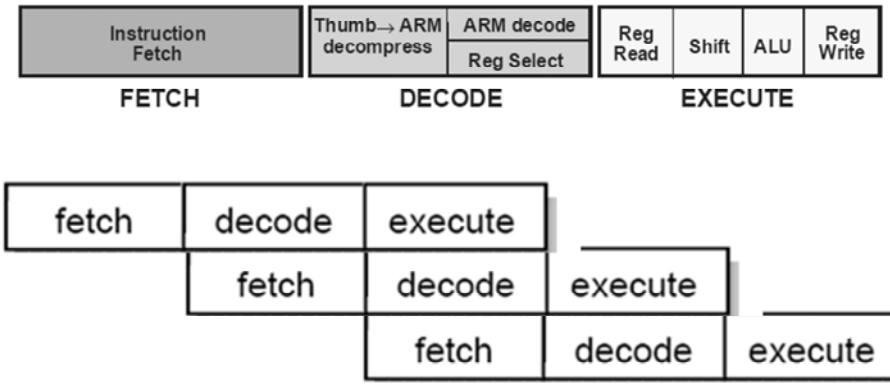
Audio processing in the SEGA Dreamcast
D-Link DSL-604+ Wireless ADSL Router
iPod from Apple
Juice Box
Lego Mindstorms NXT
Most of Nokia's mobile phone range.
Nintendo DS and Game Boy Advance from Nintendo
PocketStation
Roomba 500 series from iRobot
Sirius Satellite Radio receivers

Références ARM et WIkipédia

Pipeline de l'ARM7

R⊘DS heig-vd

- **Toutes les instructions ne peuvent pas s'exécuter en 1 cycles**



27/09/2009 Reconfigurable Computing / MSR 11

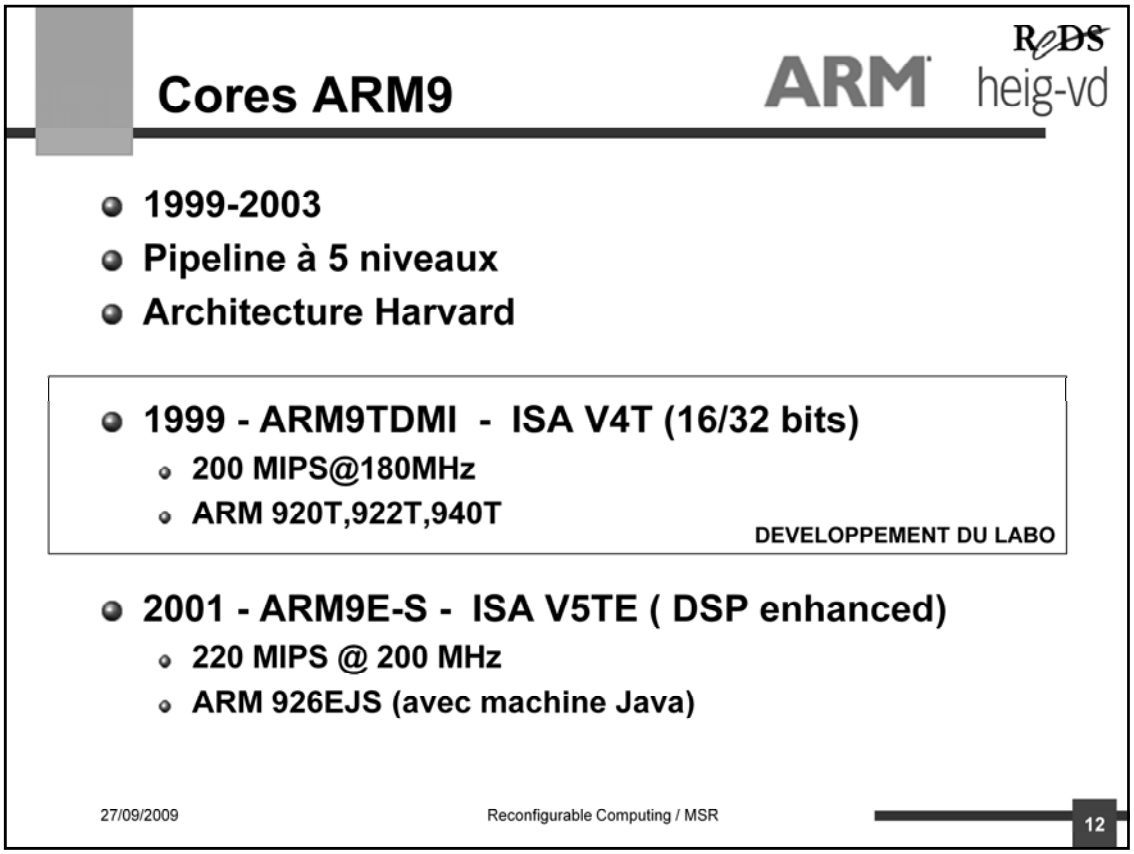
La famille ARM7 utilise un pipeline à 3 étages. La grande majorité des instructions prenant 1 cycle horloge (dans le cas où on ne travaille que sur les registres).

Fetch : L'instruction est lue dans la mémoire.

Decode : Décodage de l'instruction

Execute : Lecture des registres, opérations ALU, décalage, écriture des résultats vers les registres

Les opérations Load et Store (accès mémoire) prennent 3 cycles.



Processeurs de la famille ARM9

ARM920T with 16KB each of I/D cache and an MMU

ARM922T with 8KB each of I/D cache and an MMU

ARM940T with cache and MPU

ARM926EJ-S (includes ARM Jazelle technology which enables the direct execution of 8-bit Java bytecode in hardware), and an MMU

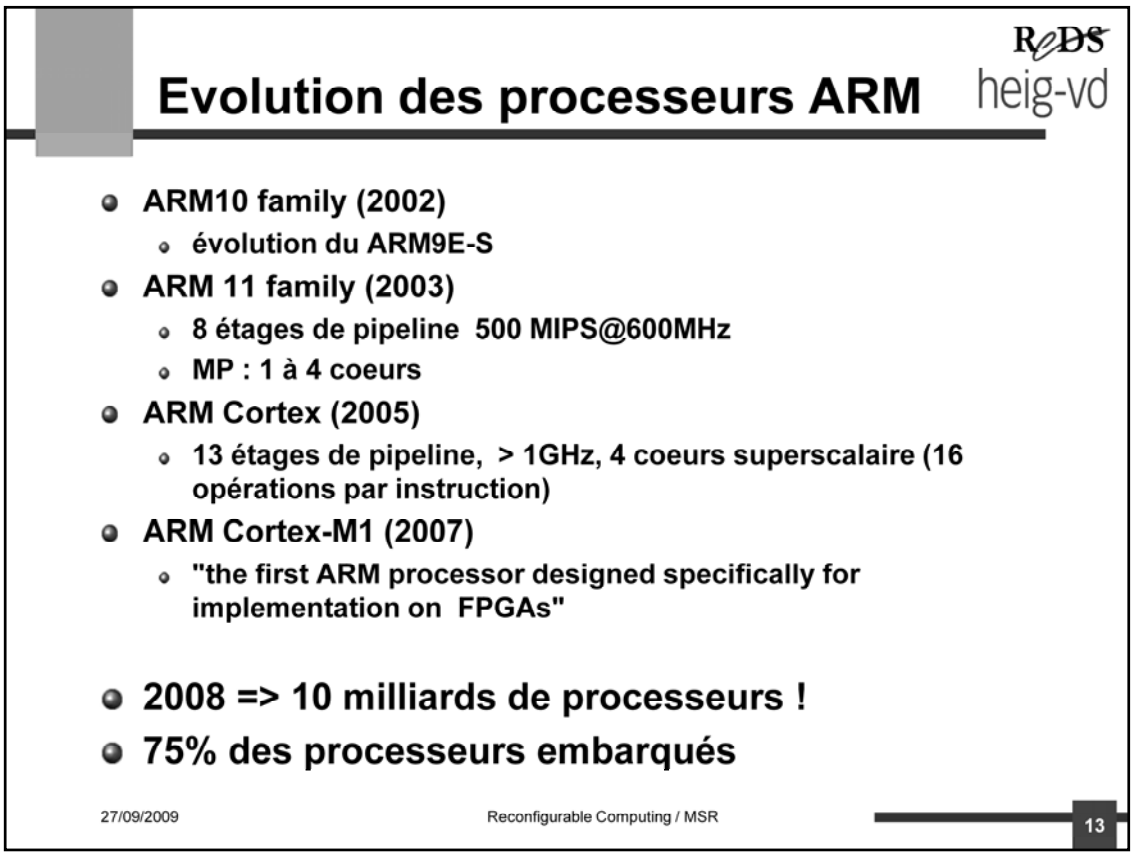
ARM920T, 922T, 940T

Armadillo, Atmel , Tapwave Zodiac, Hewlett Packard, Sun SPOT, Cirrus Logic, Samsung, NXP Philips

ARM926EJS

Sony ,Siemens,Texas Instruments ,Qualcomm ,Freescale i.MX21, i.MX27, Atmel AT91SAM9, NXP Semiconductors LPC3000, Marvell, NEC

Références ARM et WIkipédia



ARM1136J(F)-S 8-stage pipeline, SIMD, Thumb, Jazelle DBX, (VFP), Enhanced DSP instructions variable, MMU 740 @ 532-665

Texas Instruments, Nokia, Freescale, HTC , Motorola

ARM1156T2(F)-S, ARM1176JZ(F)-S 9-stage pipeline, SIMD, Thumb-2, (VFP), Enhanced DSP instructions variable, MPU

Apple iPhone, Motorola, NVIDIA

ARM11 MPCore 1-4 core SMP variable, MMU

Cortex-A8 13-stage superscalar pipeline 2.0 DMIPS/MHz

Texas Instruments , Apple iPod touch, FreeScale i.MX51-SOC, Apple iPhone 3GS, Palm Pre, Samsung , Sony Ericsson , Nokia

Cortex-R4(F) 600 DMIPS @ 475MHz real time

Cortex-M3 Microcontroller profile,

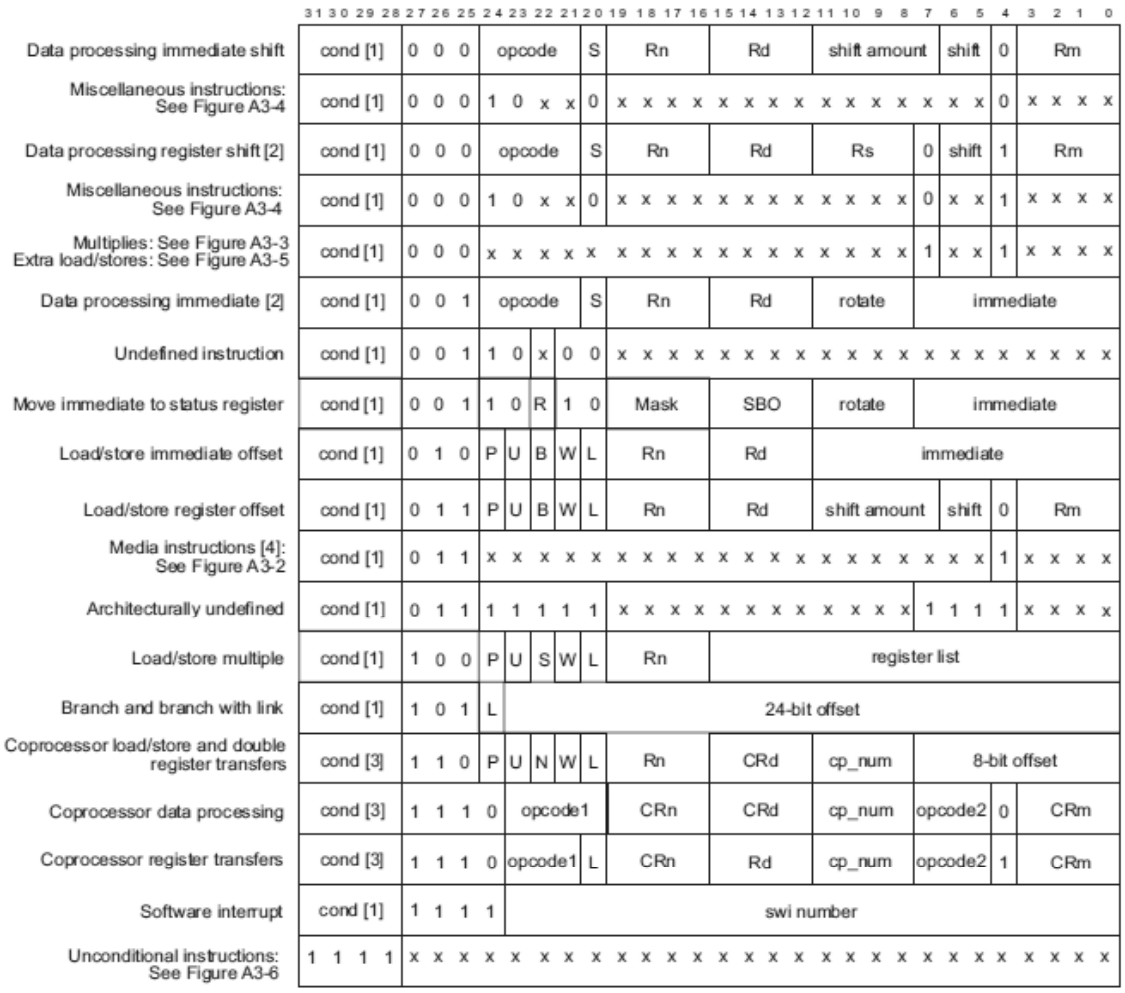
Texas Instruments, ST Microelectronics STM32, NXP Semiconductors, Toshiba , Atmel

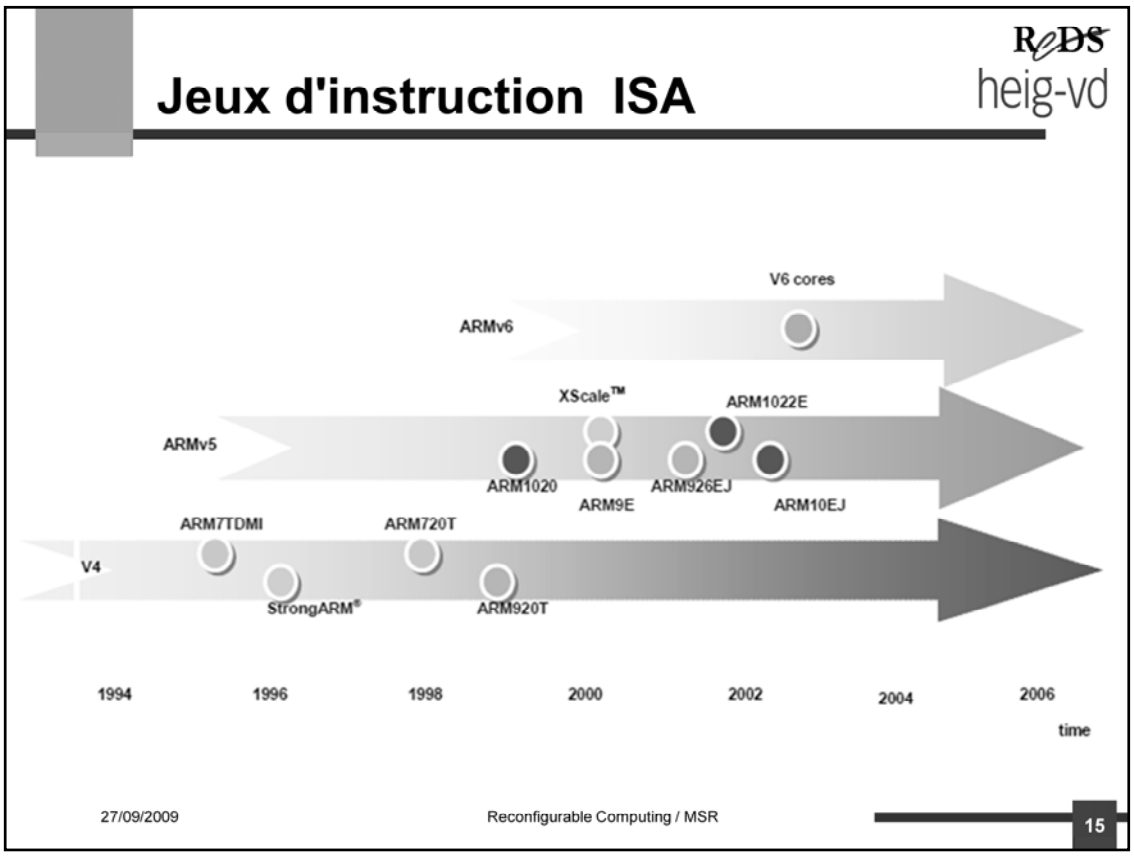
Cortex-M1 FPGA targeted, Microcontroller profile, 136 DMIPS @ 170 FPGA-dependent Actel, Altera Cyclone III

Références ARM et WIkipédia

Jeux d'instruction ISA

- **ARMv4T**
  Introduction du jeu d'instruction Thumb 16 bits
- **ARMv5TE**
  Introduction des instructions de type DSP (enhanced)
- **ARMv6**
  SIMD (Single Instruction Multiple Data) et les instructions spécifiques multi-coeurs
- **ARMv7**
  Instructions spécifiques de la famille Cortex

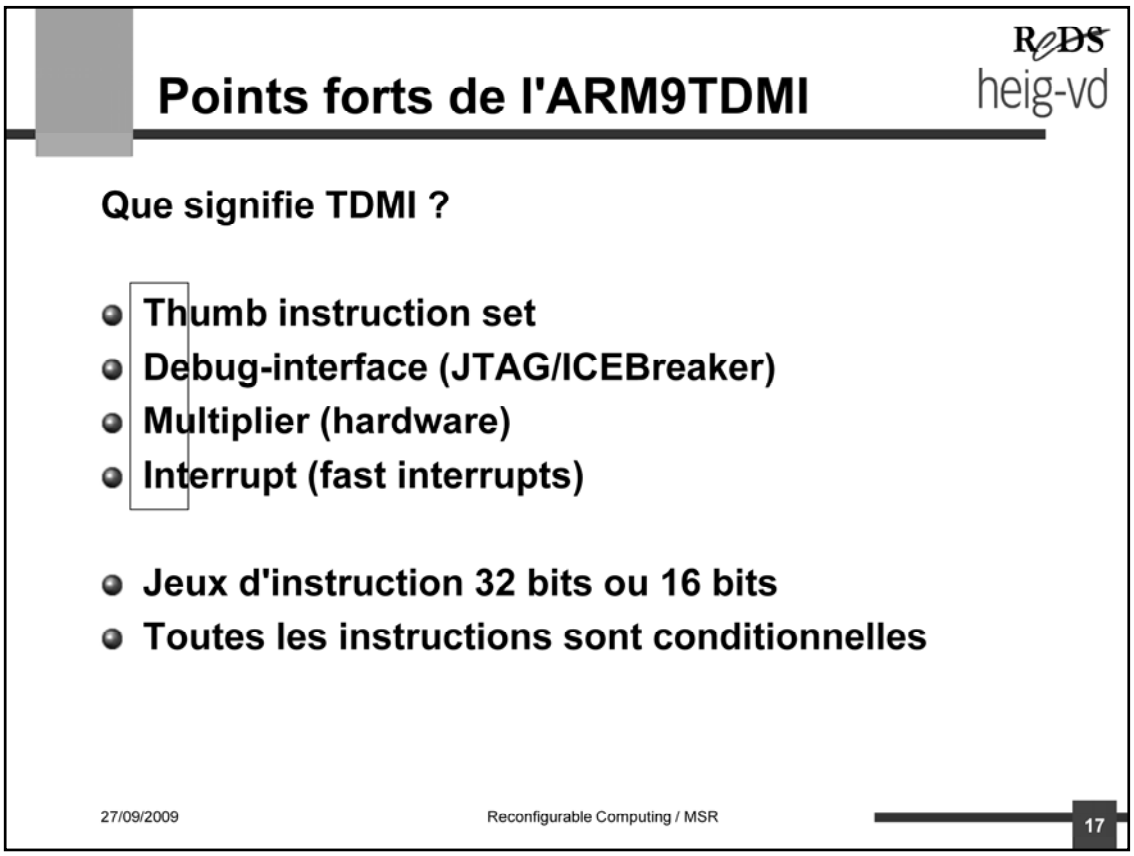




La figure ci-dessus montre l'évolution des jeux d'instructions ARM de 1994 à 2006 en fonction des mises sur le marché de nouvelles générations de cores et processeurs.

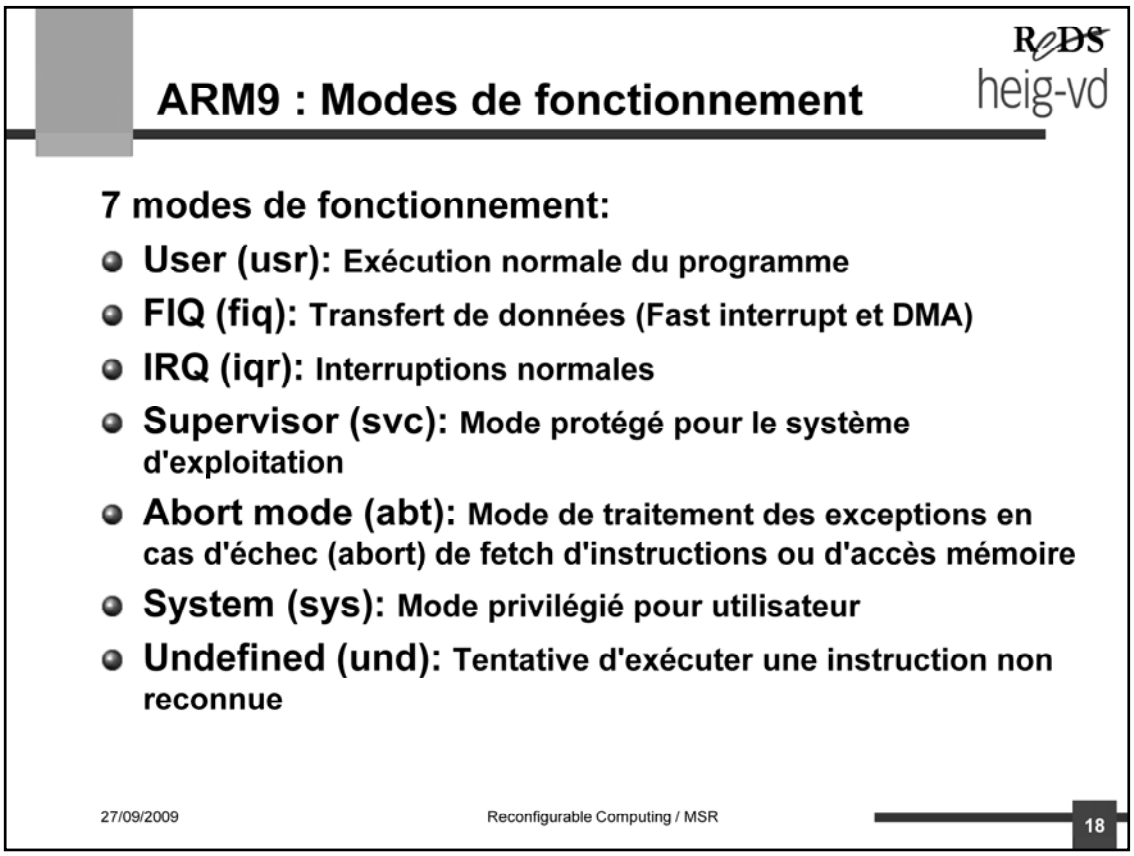
Cours ReCo

PROCESSEUR ARM9TDMI



Famille ARM9TDMI

Applications:
Hand-held products such as smart phones, communicators & PDA's
3G baseband and applications processor
Digital still camera
Consumer audio & video products
Automotive infotainment
Set-top box
Features:
32/16-bit RISC architecture (ARMv4T)
32-bit ARM instruction set for maximum performance and flexibility
16-bit Thumb instruction set for increased code density
MMU which supports operating systems including Symbian OS, Windows CE, Linux & Palm OS
Instruction and data caches: 8K/8K
Industry standard 32-bit AMBA bus interface
5-stage integer pipeline achieves 1.1 MIPS/MHz
Excellent debug support for SoC designers, including ETM interface
Portable to latest 0.18µm, 0.15µm, 0.13µm silicon processes.
Benefits:
Runs all major OSs and existing middleware.
Single development toolkit for reduced development costs and shorter development cycle time
Multiple sourcing from industry-leading silicon vendors
Upward migration path to Cortex family
Excellent debug support for SoC designers
Instruction set can be extended by the use of coprocessors



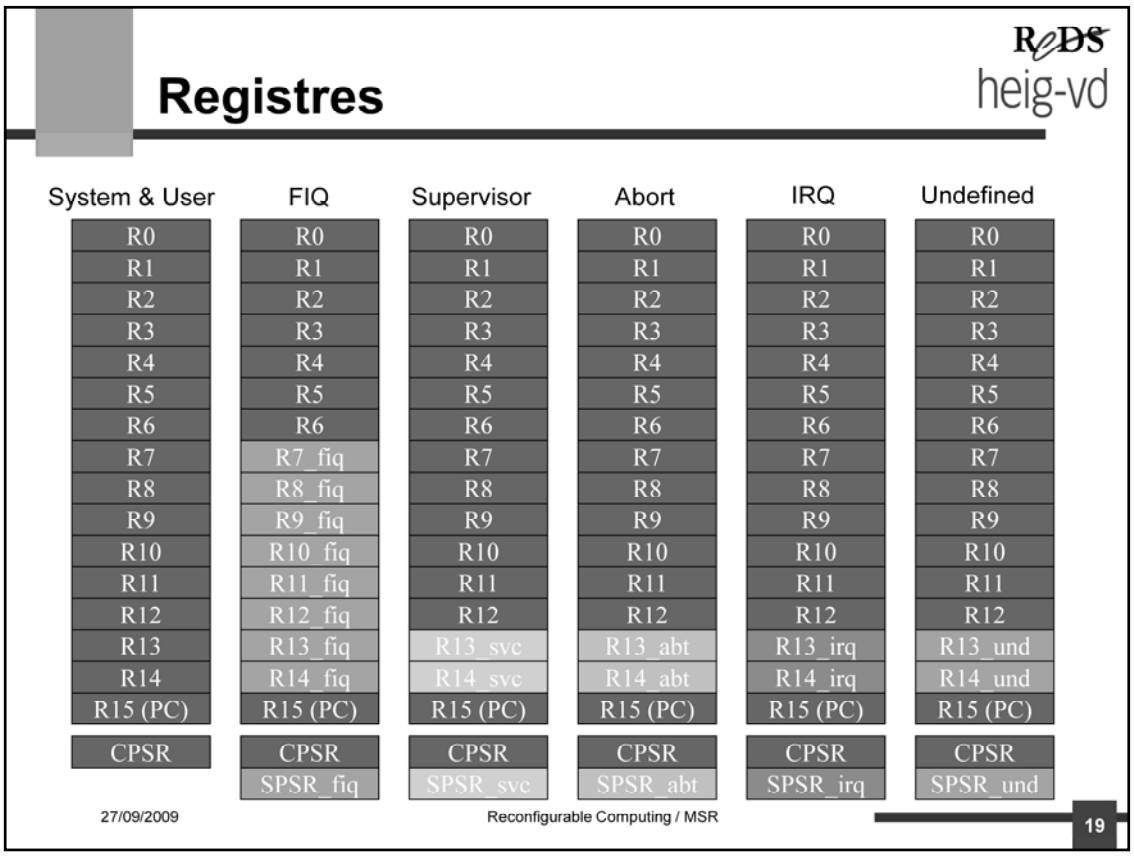
Les changements de mode peuvent être effectués sous contrôle logiciel, ou peuvent être causés par les interruptions externes ou le traitement des exceptions.

La plupart des programmes d'application s'exécutent en mode utilisateur. Lorsque le processeur est en mode utilisateur, le programme en cours d'exécution ne peut accéder à certaines ressources système protégé ou ne peut changer de mode, autrement que par une exception . Cela permet à un système d'exploitation convenablement écrit de contrôler l'utilisation des ressources système.

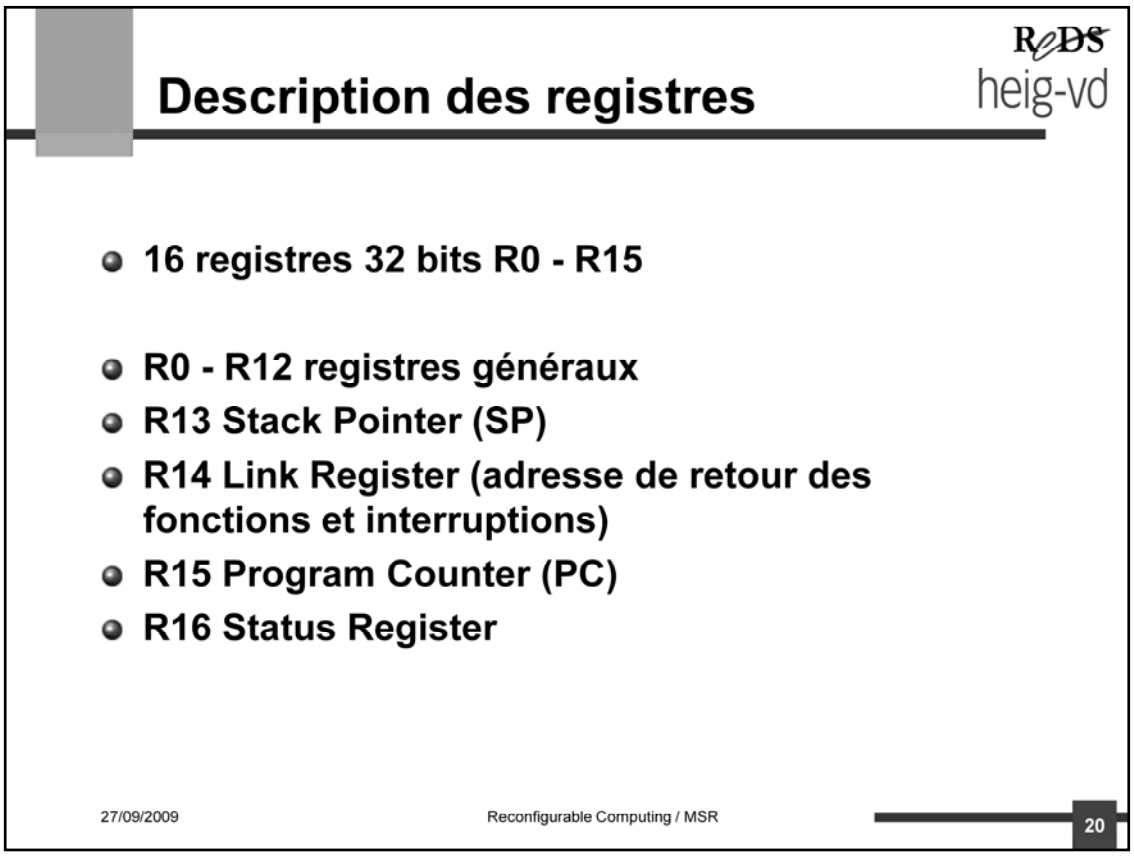
Les modes autres que le mode utilisateur sont appelés modes privilégiés. Ils ont un accès complet aux ressources système et peuvent modifier librement le mode. Cinq d'entre eux sont appelés modes d'exception: • FIQ • IRQ • Abort • non-défini. Ils sont activés lorsque surviennent des exceptions spécifiques. Chacun de ces modes a certains registres supplémentaires pour éviter de modifier des registres du mode user lors de l'exception.

Le dernier mode est le mode système, qui n'est pas commuté par une exception et qui a exactement les registres disponibles que ceux du mode utilisateur. Toutefois, il est un mode privilégié et n'est donc pas soumis aux restrictions du mode utilisateur. Il est destiné à être utilisé par des tâches de système d'exploitation qui doivent accéder aux ressources système, mais en évitant d'utiliser les registres supplémentaires associées avec les modes d'exception.

Arm Architecture Reference Manual



Notez que chaque mode d'exception comporte des registres supplémentaires (de même nom que ceux du mode user), 8 pour le FIQ et 3 pour les quatre autres.

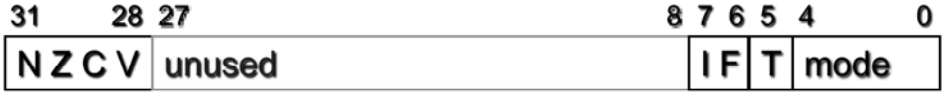


Notez que l'ARM9TDMI comporte 37 registres, 16 sont accessibles dans chaque mode.

ARM Registres d'état

R&DS heig-vd

- CPSR : Current Program Status Register
  - Bits de condition (N Z C V)
  - Bits de masquage des interruptions (I F)
  - Jeu d'instructions (T)
  - Mode du processeur (mode)



Status bits

N: résultat Négatif
Z: résultat = Zéro
C: Carry (retenue)
V: oVerflow (débordement)

Mode bits

M[4:0]	Mode	Accessible registers
0b10000	User	PC, R14 to R0, CPSR
0b10001	FIQ	PC, R14_fiq to R8_fiq, R7 to R0, CPSR, SPSR_fiq
0b10010	IRQ	PC, R14_irq, R13_irq, R12 to R0, CPSR, SPSR_irq
0b10011	Supervisor	PC, R14_svc, R13_svc, R12 to R0, CPSR, SPSR_svc
0b10111	Abort	PC, R14_abt, R13_abt, R12 to R0, CPSR, SPSR_abt
0b11011	Undefined	PC, R14_und, R13_und, R12 to R0, CPSR, SPSR_und
0b11111	System	PC, R14 to R0, CPSR (ARMv4 and above)

Conditions

- **Condition AL, always => instruction toujours exécutée**

Code	Suffix	Flags	Meaning
0000	EQ	Z set	equal
0001	NE	Z clear	not equal
0010	CS	C set	unsigned higher or same
0011	CC	C clear	unsigned lower
0100	MI	N set	negative
0101	PL	N clear	positive or zero
0110	VS	V set	overflow
0111	VC	V clear	no overflow
1000	HI	C set and Z clear	unsigned higher
1001	LS	C clear or Z set	unsigned lower or same
1010	GE	N equals V	greater or equal
1011	LT	N not equal to V	less than
1100	GT	Z clear AND (N equals V)	greater than
1101	LE	Z set OR (N not equal to V)	less than or equal
1110	AL	(ignored)	always

Opcode [31:28]	Mnemonic extension	Meaning	Condition flag state
0000	EQ	Equal	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 overflow	V clear
1000	HI	Unsigned higher	C set and Z clear
1001	LS	Unsigned lower or same	C clear or Z set
1010	GE	Signed greater than or equal	N set and V set, or N clear and V clear (N == V)
1011	LT	Signed less than	N set and V clear, or N clear and V set (N != V)
1100	GT	Signed greater than	Z clear, and either N set and V set, or N clear and V clear (Z == 0,N == V)
1101	LE	Signed less than or equal	Z set, or N set and V clear, or N clear and V set (Z == 1 or N != V)
1110	AL	Always (unconditional)	-
1111	-	See *Condition code 0b1111*	-

Cours ReCo

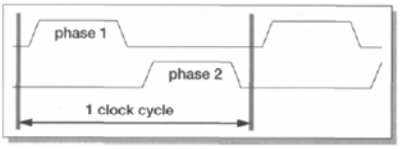
PIPELINE DE L'ARM9TDMI

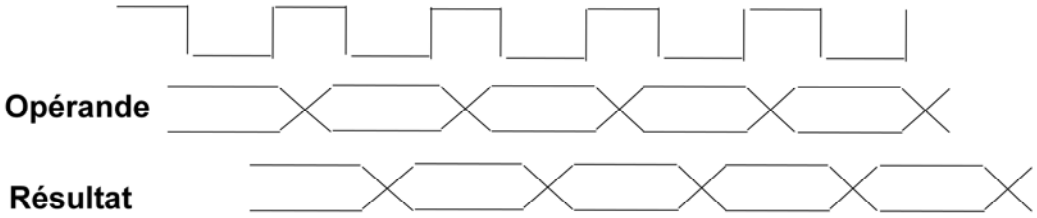
ARM9 mécanisme de latches

R&DS heig-vd

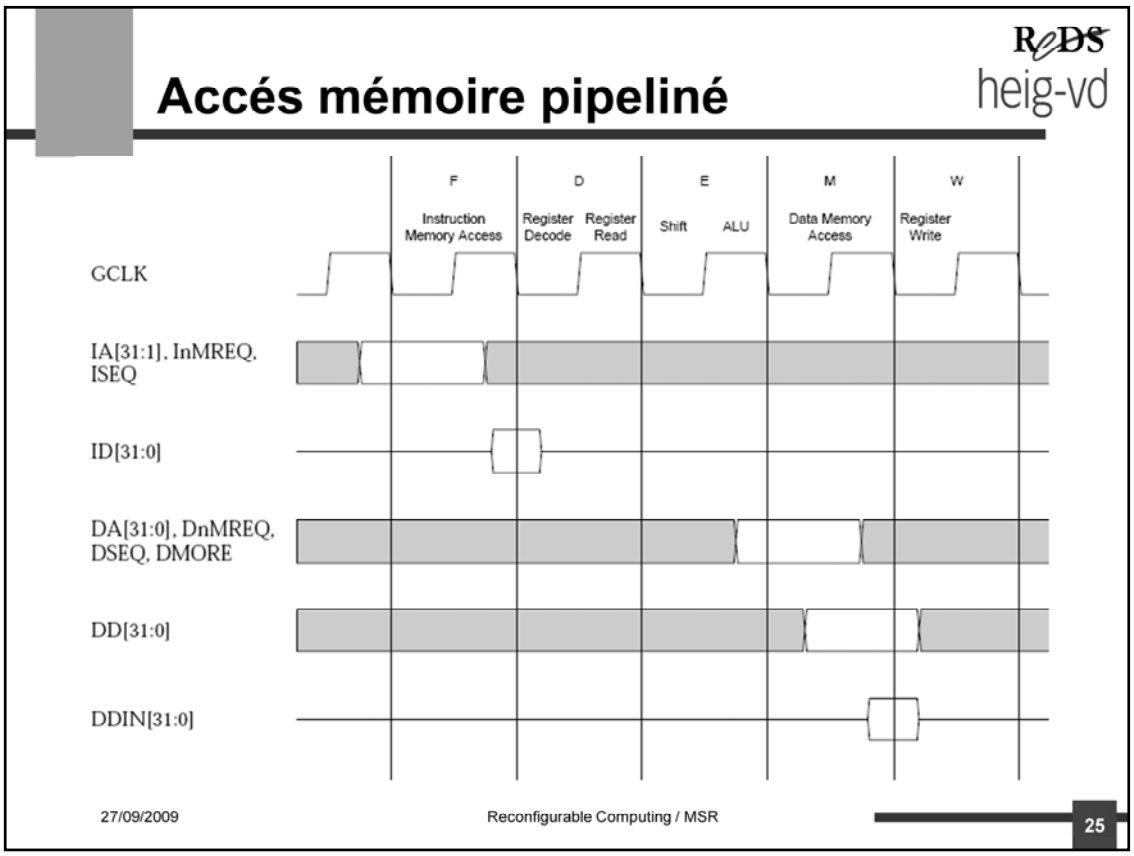
- **Pas de bascules clockées , mais des latches sur 2 phases différentes de l'horloge**

1. **phase 1 ouvert (opérande)**
2. **traitement**
3. **phase 2 ouvert (résultat)**





Voir une explication plus détaillée dans le livre de Furber, ARM Soc Architecture § 4.4.



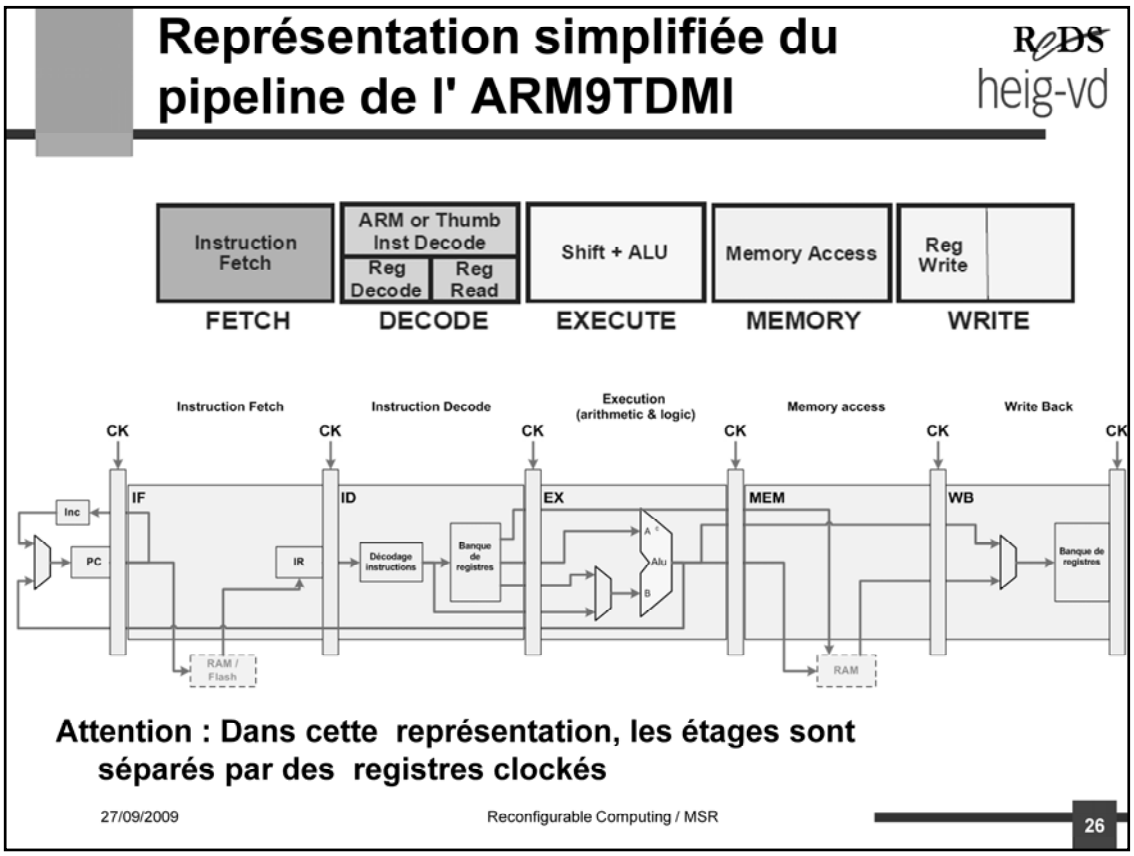
L'horloge du processeur est représentée sur la première ligne du chronogramme qui est divisée en 5 cycles : les cycles de traitement du processeur – Instruction , Decode, Execute, Memory access et Write back. Certains cycles sont séparées en deux parties comme le cycle Decode – Register Decode et Register read. (Voir la slide "latch")

Les deux lignes suivantes du chronogramme représentent l'accès à la mémoire d'instruction – IA le bus d'adresse avec les signaux de contrôle et ID le bus de données (accès aux mots d'instruction 32 bits).

Notons que les adresses sont présentes un cycle d'horloge avant les données. Les accès mémoire des cœurs ARM9 sont toujours pipelinés. Ceci permet un accès mémoire synchrone. Les adresses et les données lues sont valides après par le front montant de l'horloge, et les datas sont latchées au front descendant de l'horloge.

Il en est de même pour les 3 lignes suivantes qui représentent l'accès à la mémoire de données – DA le bus d'adresse avec les signaux de contrôle , DD le bus des données écrites dans la mémoire et DDIN le bus de données lues dans la mémoire. Les adresses sont valides après le front montant de l'horloge dans le cycle Execute (un cycle avant les données) , les données lues sont latchées au front descendant terminant le cycle Memory access, les données écrites sont présentes après le front montant d'horloge du cycle Memory access et doivent être latchées par la mémoire sur le front montant d'horloge en fin du cycle M.

Un signal en sortie indique à la mémoire si l'accès est séquentiel (adresse incrémentée de 1, 2 ou 4). La mémoire ou un périphérique peut introduire des cycles d'attente (wait states).



De gauche à droite :

1. Incrémentation du PC ou sélection d'une adresse provenant de l'ALU (branchement)

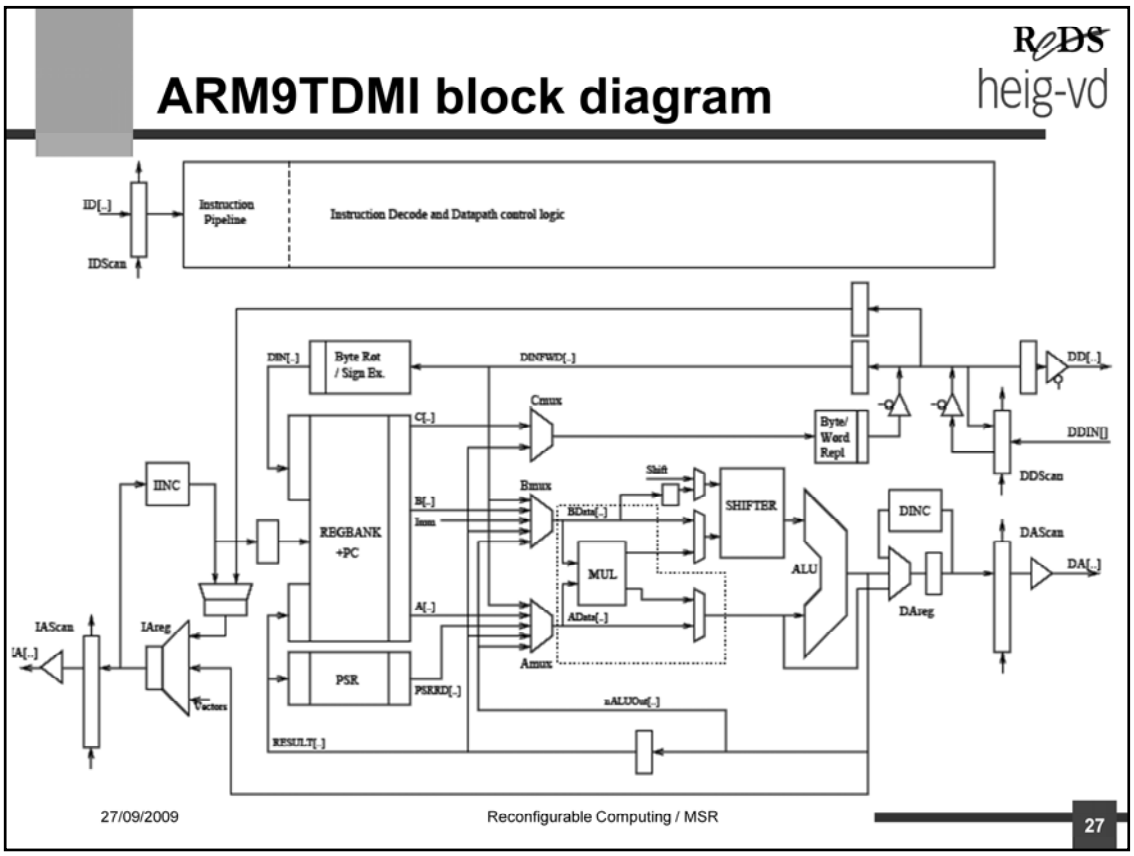
2. Fetch, lecture de l'instruction

3. Décodage, sélection de 3 opérandes dans la banque de registre

4. Exécution de l'opération avec pour un opérande sélection entre une valeur immédiate et la sortie d'un registre

5. Ecriture de la mémoire avec le contenu d'un registre ou lecture de la mémoire, avec la sortie de l'ALU fournissant l'adresse mémoire.

6. Ecriture dans la banque de registre (destination) avec en entrée sélection d'un registre ou de la doonée lue en mémoire

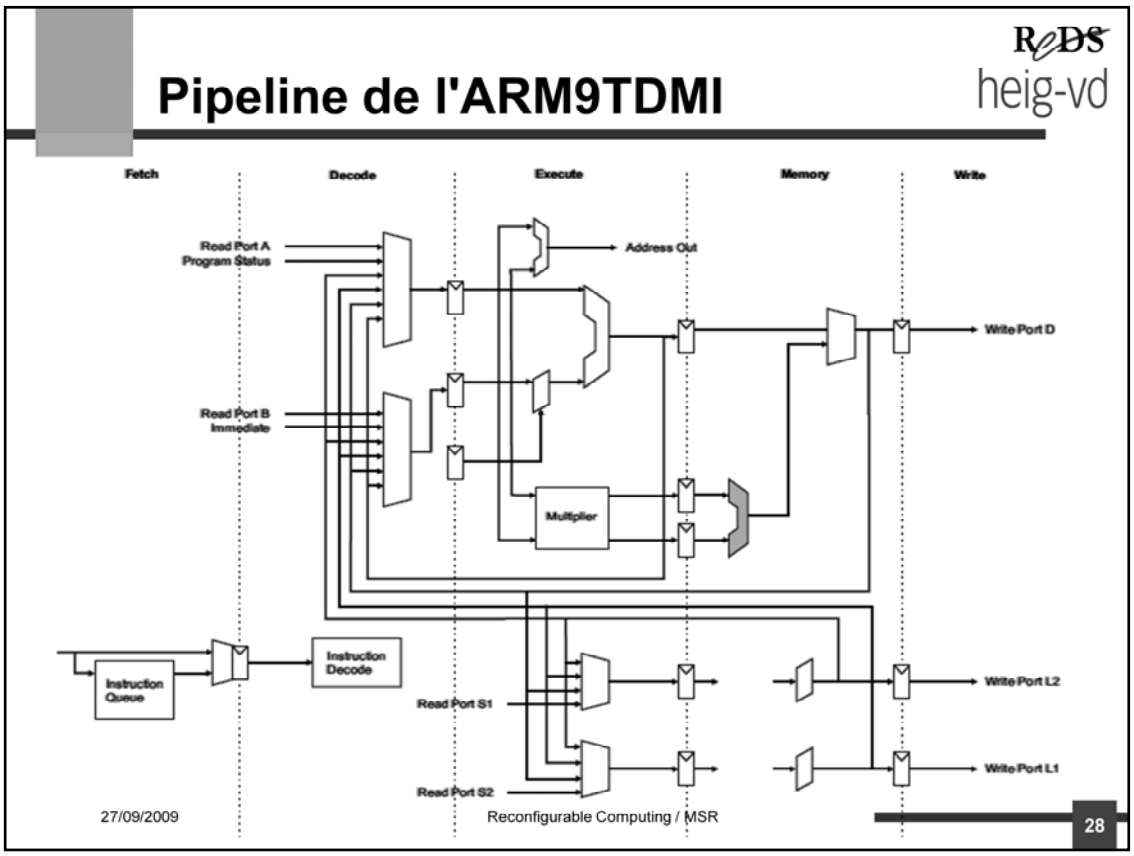


A l'entrée de l'ALU, deux multiplexeurs permettent de choisir comme opérandes les registres (A et/ou B), une valeur immédiate, ou la sortie de l'ALU .

Une troisiéme sortie de la banque de registre peut être utilisée en parallèle pour une écriture en mémoire (sortie DD). Pour une lecture, la donnée en provenance de la mémoire (DDIN) est acheminée à l'entrée de la banque de registre .

La donnée en sortie de l'ALU peut être soit écrite dans la banque de registre, soit utilisée comme adresse de la mémoire de donnée (DA) ou encore comme adresse de la mémoire d'instruction (IA).

Pour les opérations séquentielles, l'adresse d'instruction est incrémentée par le bloc INC. Les instructions sont lues sur le bus ID.



Ce bloc diagramme de l'ARM9TDMI est sans doute le seul dans la littérature donnant une explication du mécanisme hardware de pipeline. Comparez ce diagramme à celui de la slide précédente. On note un certain nombre de différence , dont l'apparition d'une deuxième ALU dans la partie Memory !

Ce bloc diagramme est extrait du document ARM Product Overview ARM920T.

Cours ReCo

JEU D'INSTRUCTION V4T

ARM9TDMI jeu d'instructions V4T

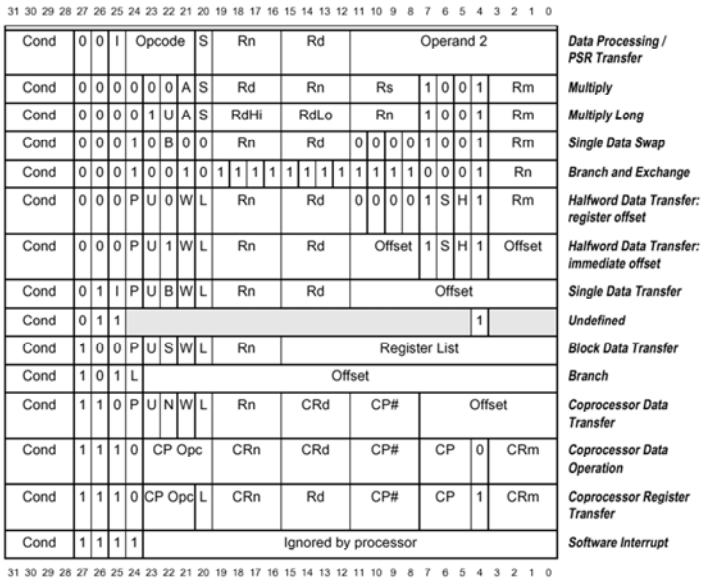
R⁄DS heig-vd

- Instructions de traitement
  - Arithmétique, logique et shift
- Instructions de transfert
  - mémoire <=> registre ou registre <=> registre
- Instructions de contrôle
  - branchements conditionnels et sauts
- Format instructions 32 bits
  - Sauf jeu compressé 16 bits Thumb

The ARMv4T ARM instruction set

Mnemonic	Operation	Mnemonic	Operation
MOV	Move	MVN	Move Not
ADD	Add	ADC	Add with Carry
SUB	Subtract	SBC	Subtract with Carry
RSB	Reverse Subtract	RSC	Reverse Subtract with Carry
CMP	Compare	CMN	Compare Negated
TST	Test	TEQ	Test Equivalence
AND	Logical AND	BIC	Bit Clear
EOR	Logical Exclusive OR	ORR	Logical (inclusive) OR
MUL	Multiply	MLA	Multiply Accumulate
SMULL	Sign Long Multiply	SMLAL	Signed Long Multiply Accumulate
UMULL	Unsigned Long Multiply	UMLAL	Unsigned Long Multiply Accumulate
MRS	Move From Status Register	MSR	Move to Status Register
B	Branch		
BL	Branch and Link		
BX	Branch and Exchange	SWI	Software Interrupt
LDR	Load Word	STR	Store Word
LDRH	Load Halfword	STRH	Store Halfword
LDRB	Load Byte	STRB	Store Byte
LDRSH	Load Signed Halfword	LDRSB	Load Signed Byte
LDMIA	Load Multiple	STMIA	Store Multiple
SWP	Swap Word	SWPB	Swap Byte
CDP	Coprocessor Data Processing		
MRC	Move From Coprocessor	MCR	Move to Coprocessor
LDC	Load To Coprocessor	STC	Store From Coprocessor

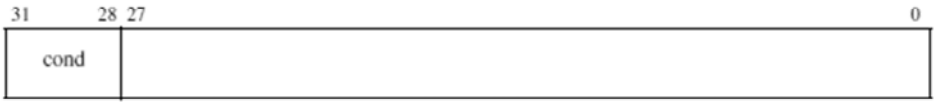
ISA V4T


31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Cond 0 0 I Opcode S Rn Rd Operand 2 Data Processing / PSR Transfer
Cond 0 0 0 0 0 0 A S Rd Rn Rs 1 0 0 1 Rm Multiply
Cond 0 0 0 0 1 U A S RdHi RdLo Rn 1 0 0 1 Rm Multiply Long
Cond 0 0 0 1 0 B 0 0 Rn Rd 0 0 0 0 1 0 0 1 Rm Single Data Swap
Cond 0 0 0 1 0 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 Rn Branch and Exchange
Cond 0 0 0 P U 0 W L Rn Rd 0 0 0 0 1 S H 1 Rm Halfword Data Transfer: register offset
Cond 0 0 0 P U 1 W L Rn Rd Offset 1 S H 1 Offset Halfword Data Transfer: immediate offset
Cond 0 1 I P U B W L Rn Rd Offset Single Data Transfer
Cond 0 1 1 1 Undefined
Cond 1 0 0 P U S W L Rn Register List Block Data Transfer
Cond 1 0 1 L Offset Branch
Cond 1 1 0 P U N W L Rn CRd CP# Offset Coprocessor Data Transfer
Cond 1 1 1 0 CP Opc CRn CRd CP# CP 0 CRm Coprocessor Data Operation
Cond 1 1 1 0 CP Opc L CRn Rd CP# CP 1 CRm Coprocessor Register Transfer
Cond 1 1 1 1 Ignored by processor Software Interrupt
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Instructions conditionnelles

R&DS heig-vd

- Toutes les instructions sont conditionnelles => test du status register
- Champ de condition de 4 bits => 14 conditons



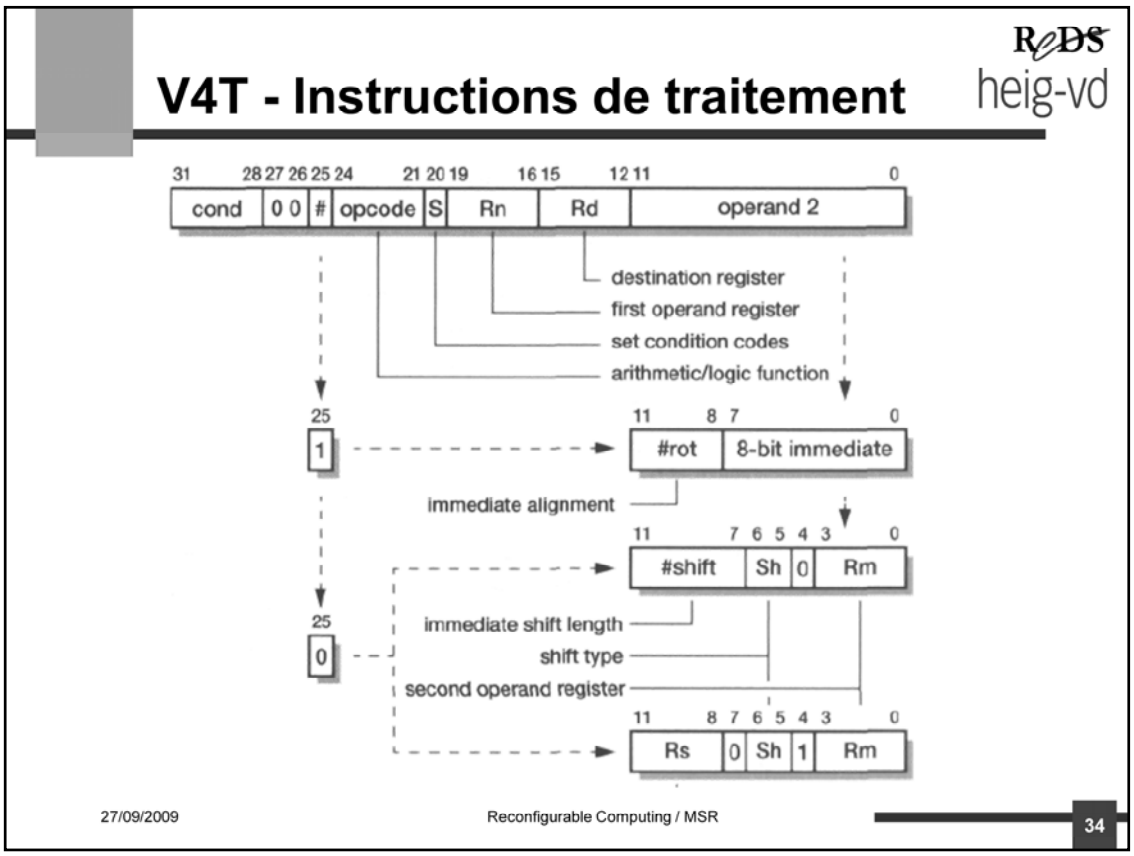
- Condition AL (always) => pas de prise en compte de condition

Opcode [31:28]	Mnemonic extension	Meaning	Condition flag state
0000	EQ	Equal	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 overflow	V clear
1000	HI	Unsigned higher	C set and Z clear
1001	LS	Unsigned lower or same	C clear or Z set
1010	GE	Signed greater than or equal	N set and V set, or N clear and V clear (N == V)
1011	LT	Signed less than	N set and V clear, or N clear and V set (N != V)
1100	GT	Signed greater than	Z clear, and either N set and V set, or N clear and V clear (Z == 0,N == V)
1101	LE	Signed less than or equal	Z set, or N set and V clear, or N clear and V set (Z == 1 or N != V)
1110	AL	Always (unconditional)	-

Instructions Thumb

- Jeu d'instruction limité 16 bits
- But : économiser la taille mémoire (35%)
- Nombre de registres accessibles limité 16 => 8
- Seuls les branchements sont conditionnels

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Shift by immediate	0	0	0	opcode [1]		immediate					Rm			Rd		
Add/subtract register	0	0	0	1	1	0	opc	Rm			Rn			Rd		
Add/subtract immediate	0	0	0	1	1	1	opc	immediate			Rn			Rd		
Add/subtract/compare/move immediate	0	0	1	opcode		Rd / Rn		immediate								
Data-processing register	0	1	0	0	0	0	opcode				Rm / Rs			Rd / Rn		
Special data processing	0	1	0	0	0	1	opcode [1]		H1	H2	Rm			Rd / Rn		
Branch/exchange instruction set [3]	0	1	0	0	0	1	1	1	L	H2	Rm			SBZ		
Load from literal pool	0	1	0	0	1	Rd			PC-relative offset							
Load/store register offset	0	1	0	1	opcode			Rm			Rn			Rd		
Load/store word/byte immediate offset	0	1	1	B	L	offset					Rn			Rd		
Load/store halfword immediate offset	1	0	0	0	L	offset					Rn			Rd		
Load/store to/from stack	1	0	0	1	L	Rd			SP-relative offset							
Add to SP or PC	1	0	1	0	SP	Rd			immediate							
Miscellaneous: See Figure 6-2	1	0	1	1	x	x	x	x	x	x	x	x	x	x	x	x
Load/store multiple	1	1	0	0	L	Rn			register list							
Conditional branch	1	1	0	1	cond [2]				offset							
Undefined instruction	1	1	0	1	1	1	1	0	x	x	x	x	x	x	x	x
Software interrupt	1	1	0	1	1	1	1	1	immediate							
Unconditional branch	1	1	1	0	0	offset										
BLX suffix [4]	1	1	1	0	1	offset										0
Undefined instruction	1	1	1	0	1	x	x	x	x	x	x	x	x	x	x	1
BL/BLX prefix	1	1	1	1	0	offset										
BL suffix	1	1	1	1	1	offset										

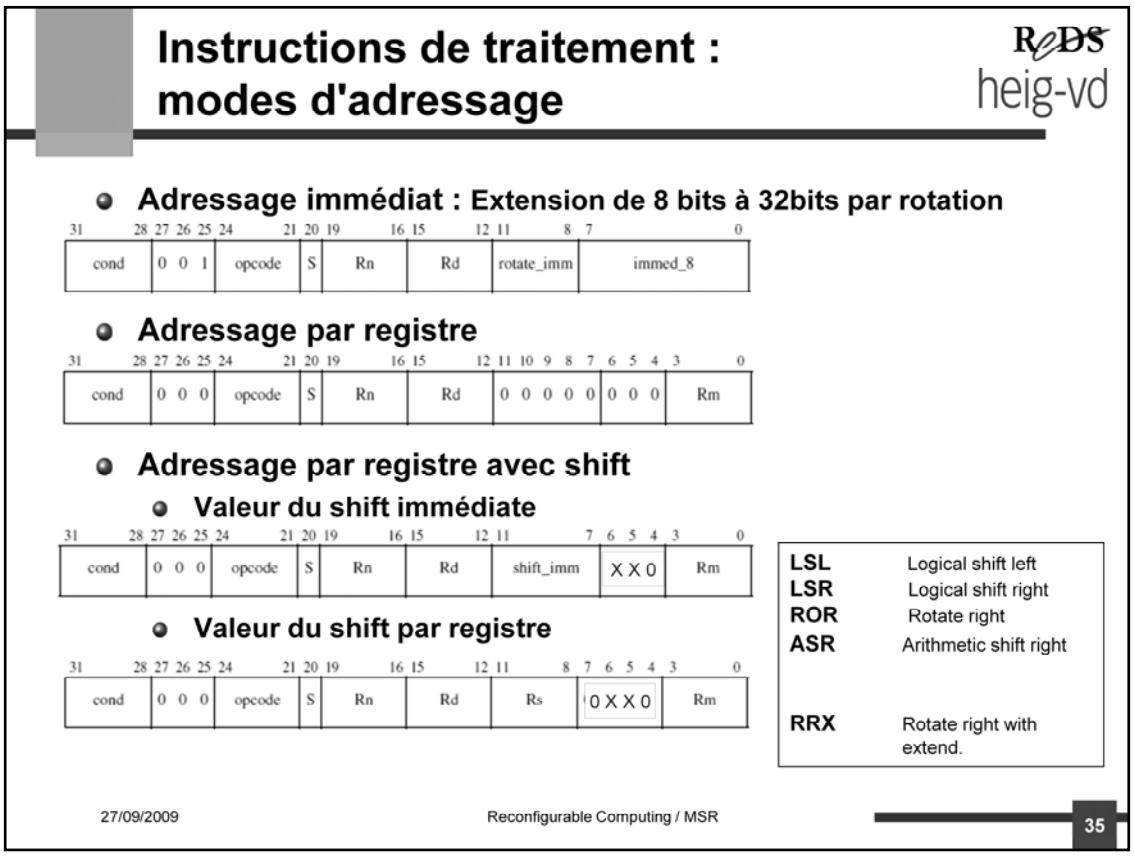


Instructions effectuées en 1 cycles sauf accès à R15.

Le status register est mis à jour seulement si le flag S est activé

Exemples

0000	AND	Logical bit-wise	Rd:=Rn AND Op2
0001	EOR	Logical bit-wise exclusive	Rd := Rn EOR Op2
0010	SUB	Subtract	Rd := Rn - Op2
0011	RSB	Reverse subtract	Rd := Op2 - Rn
0100	ADD	Add Rd :=	Rn + Op2
0101	ADC	Add with carry	Rd := Rn + Op2 + C
0110	SBC	Subtract with carry R	d := Rn - Op2 + C - 1
0111	RSC	Reverse subtract with carry	Rd := Op2 - Rn + C - 1
1000	TST	Test	Rn AND Op2
1001	TEQ	Test equivalence	Rn EOR Op2
1010	CMP	Compare Sec on	Rn - Op2
1011	CMN	Compare negated	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



Les instructions de traitement de données comportent en général 2 ou 3 opérandes . Le mode d'adressage d'un des opérandes est sélectionnable, l e ou les opérandes restants sont toujours des registres.

Adressage immédiat

La valeur de l'opérande immédiat (sur 32 bits) est formée par la rotation - de 0 à 15 bits - d'une constante de 8 bit

Valide : 0xFF,0x104,0xFF0,0xFF00,0xFF000,0xFF000000,0xF000000F

Invalide : 0x101,0x102,0xFF1,0xFF04,0xFF003,0xFFFFFFFF,0xF000001F

```
Exemples :      ADD R3, R3, #1      ; Add one to the value of register 3
                CMP R7, #1000       ; Compare value of R7 with 1000
```

Adressage par registre

Tous les opérandes sont des registres .

```
Exemples :      MOV R2, R0          ; Move the value of R0 to R2
                ADD R4, R3, R2      ; Add R2 to R3, store result in R4
```

Adressage par registre avec shift ou rotation de l'opérande

5 types de shifts/rotates

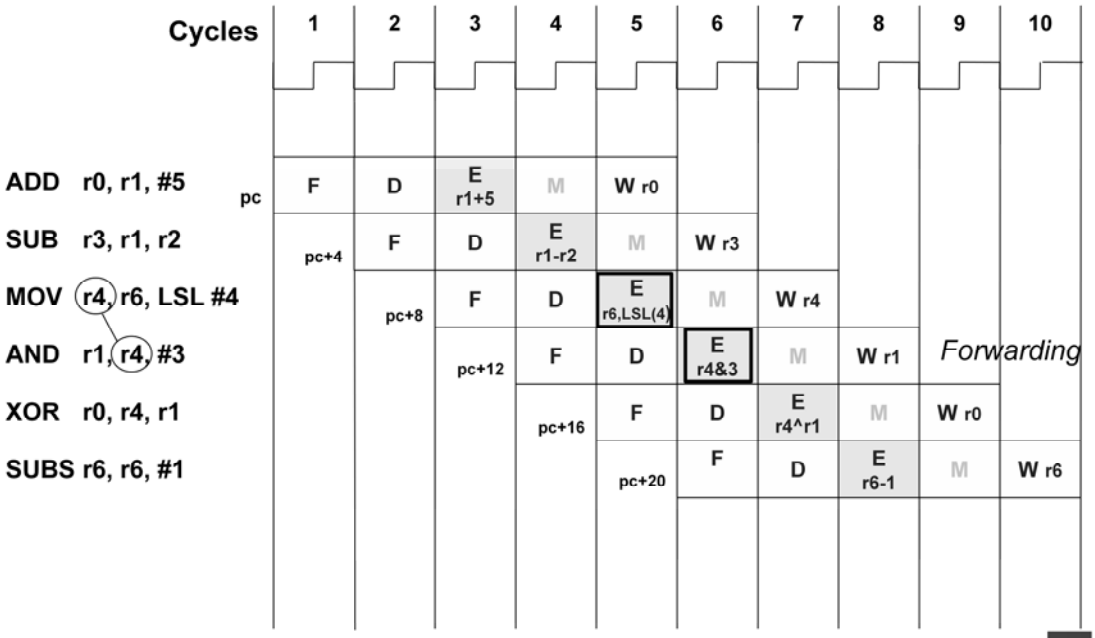
10	ASR	Arithmetic shift right (bit de signe étendu à gauche)
00	LSL	Logical shift left
01	LSR	Logical shift right
11	ROR	Rotate right
	RRX	Rotate right with extend.

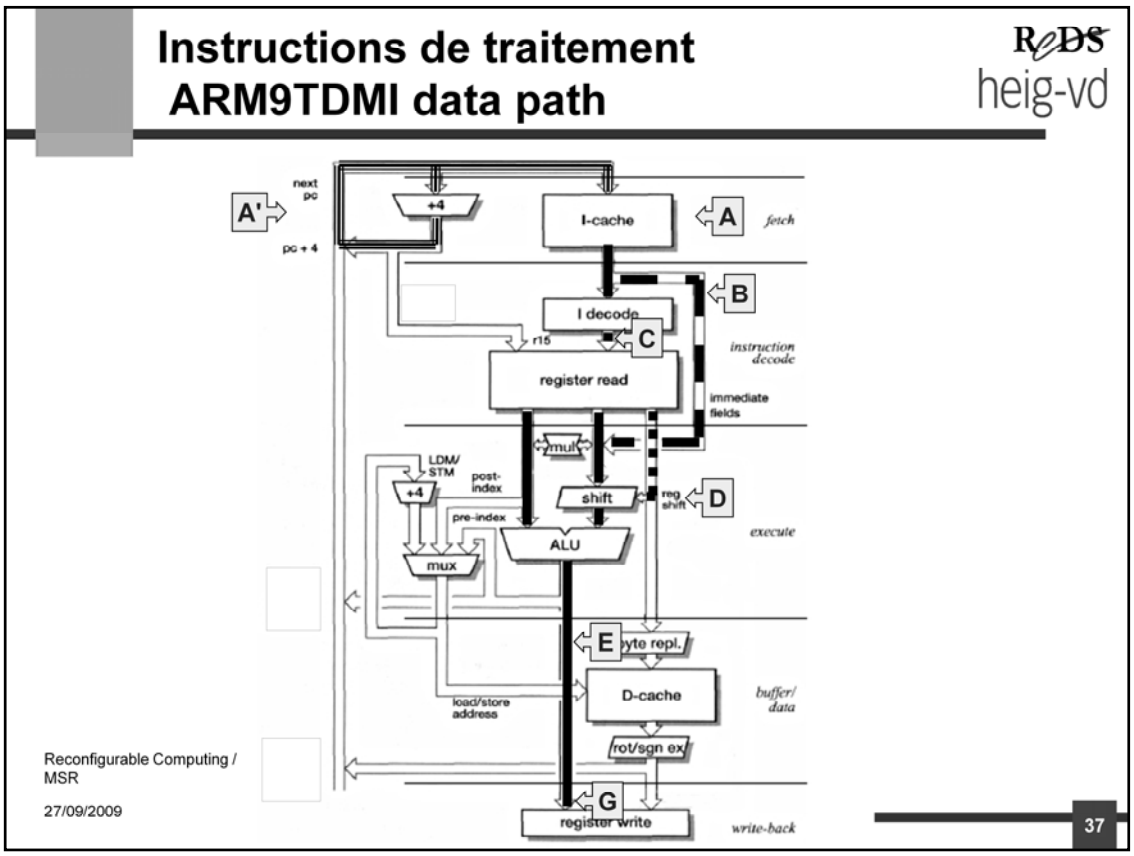
(rotation d'un bit avec utilisation du cary comme 33ème bit, permet les shifts et rotations avec des données de taille > 32)

Le nombre de bit du shift ou rotation peut être spécifié par une valeur immédiate ou par un registre:

```
Exemples :      MOV R2, R0, LSL #2          ; Shift R0 left by 2, write to R2, (R2=R0x4)
                ADD R9, R5, R5, LSL #3      ; R9 = R5 + R5 x 8 or R9 = R5 x 9
                MOV R12, R4, ROR R3         ; R12 = R4 rotated right by value of R3
```

Instructions de traitement pipeline





Chemin de données durant les 5 cycles d'une l'instruction de traitement

Note : pc représente l'adresse de l'instruction

A- Durant le cycle 1 / Fetch , l'instruction à l'adresse pc est lue dans la mémoire d'instruction

A'- Durant le même cycle le PC est incrémenté de 4 pour permettre la lecture de l'instruction suivante à l'adresse pc+4 au cycle suivant.

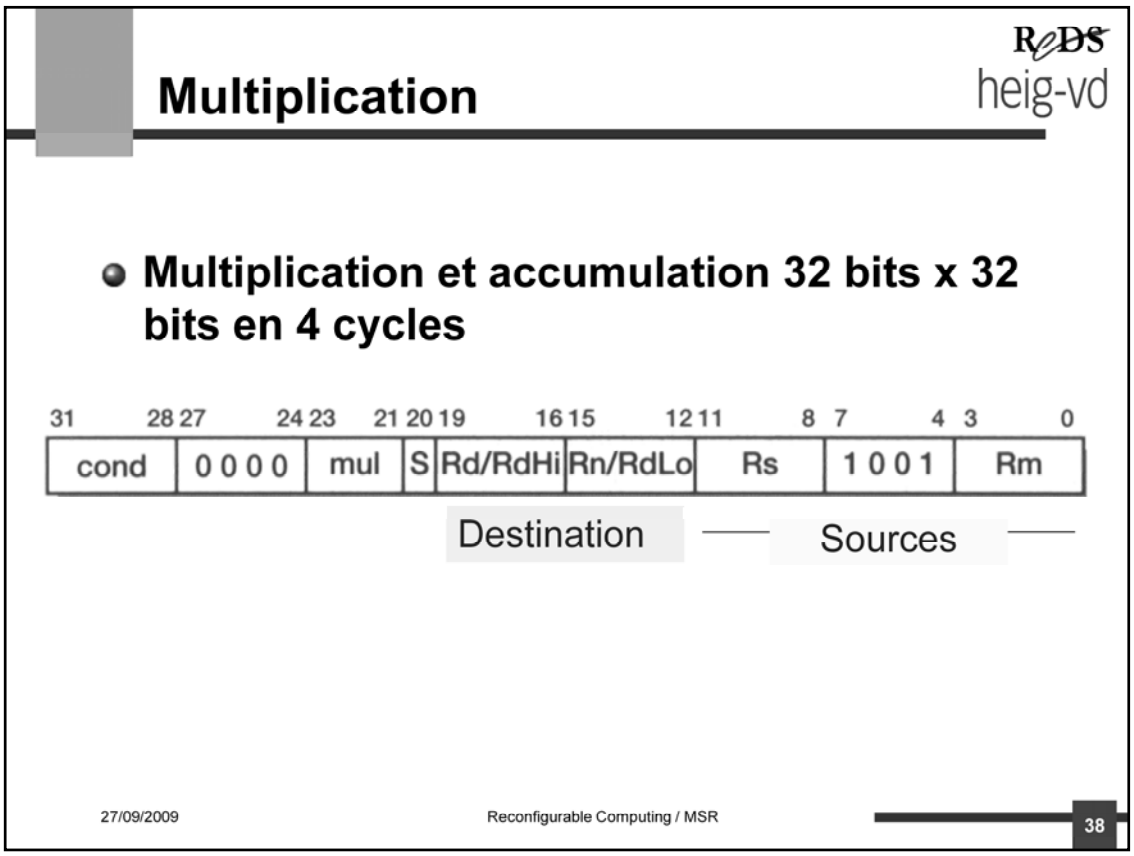
B- Durant le cycle 2 / Decode , l'instruction d'adresse pc est décodée. Si l'instruction comporte un champ d'adressage immédiat, la donnée de ce champ est acheminée à l'entrée de l'alu (après passage par un registre)

C- Durant le même cycle, l'adresse des registres opérandes est décodée . Les données contenues dans ces registres seront présentes à l'entrée de l'alu au prochain cycle.

D- Durant le cycle 3- Execute , l'opération spécifiée par l'instruction est effectuée.

E- Durant le cycle 4 / Memory Access, la donnée résultante de l'opération traverse l'étage mémoire sans être modifiée

D- Durant le cycle 5 / Write Back, le résultat est écrit dans le registre destination.

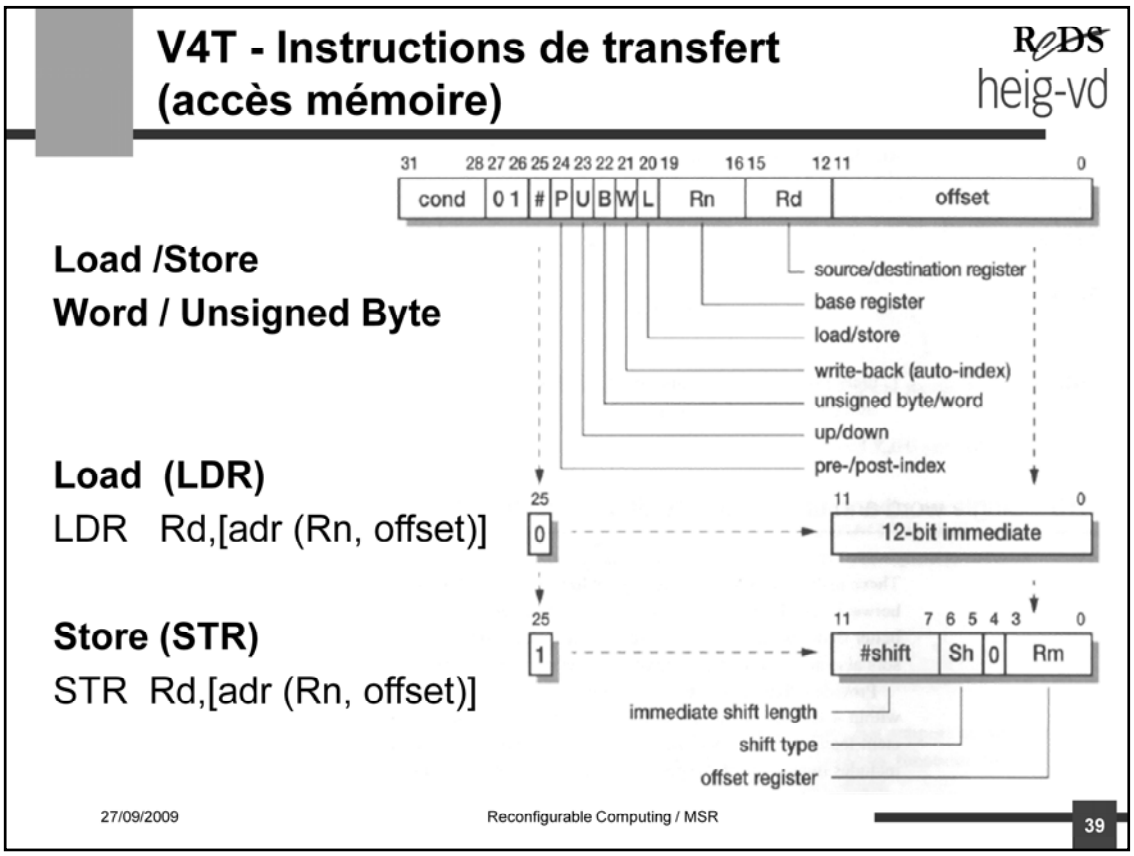


Les instructions de multiplication peuvent comporter 2 registres de destination (RdHi et RdLo) pour stocker un résultat sur plus de 32 bits. A noter que l'instruction MLA comporte 1 registre de destination (Rd) et 3 registres de source (Rn,Rs,Rm).

000	MUL	Multiply (32-bit result)	Rd:=(Rm*Rs)[31:0]
001	MLA	Multiply-accumulate (32-bit result)	Rd:=(Rm*Rs + Rn)[31:0]
100	UMULL	Unsigned multiply long	RdHir:RdLo := Rm * Rs
101	UMLAL	Unsigned multiply-accumulate long	RdHi: RdLo += Rm * Rs
110	SMULL	Signed multiply long	RdHi:RdLo := Rm * Rs
111	SMLAL	Signed multiply-accumulate long	RdHi:RdLo +=Rm*Rs

Types de multiplications :

Normal	32-bit x 32-bit, bottom 32-bit result
Long	32-bit x 32-bit, 64-bit result
Halfword	16-bit x 16-bit, 32-bit result
Word ∞ halfword	32-bit x 16-bit, top 32-bit result
Most significant word	32-bit x 32-bit, top 32-bit result
Dual halfword	16-bit x 16-bit, 32-bit result.



Les instructions de transfert sont de type Load (LDR) ou Store (STR). Les formats de données supportés sont les mots (signés ou non signés) de 32 bits et les octets non-signés.

Load (lecture mémoire)

Chargement d'un registre (de destination) avec une donnée en mémoire .

Rd est le registre destination.

L'adresse mémoire est calculée à partir de la valeur du registre Rn et d'un offset.

LDR Rd, [adr(Rn, offset)] Rd → Mémoire [adr]

Store (écriture mémoire)

Ecriture du contenu d'un registre (de source) à une adresse mémoire.

Rd est le registre source.

L'adresse mémoire est calculée à partir de la valeur du registre Rn et d'un offset

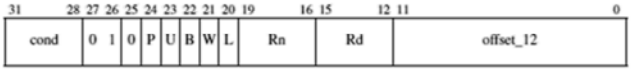
STR Rd, [adr(Rn, offset)] Mémoire [adr] → Rd

Note: L'adressage de la mémoire est un adressage indirect relatif

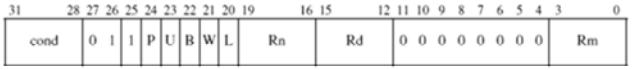
Instructions de transfert : modes d'adressage

R&DS heig-vd

- Adresse mémoire = adresse de base + offset
- Adresse de base toujours dans un registre (base register)
- 3 modes d'adressage pour l'offset :
  - Immédiat



  - Par registre



  - Par registre avec décalage (scaled)

27/09/2009 Reconfigurable Computing / MSR 40

L'adresse mémoire est calculée à partir de la valeur d'un registre (adresse de base) et un offset, qui sont additionnés (ou soustraits) pour former l'adresse mémoire.

Le registre de base peut être l'un des registres généraux (y compris le PC, ce qui permet un adressage relatif au PC (adresse calculée indépendante de la localisation du code dans la mémoire).

L'offset peut-être adressé selon trois différents modes :

Immédiat :

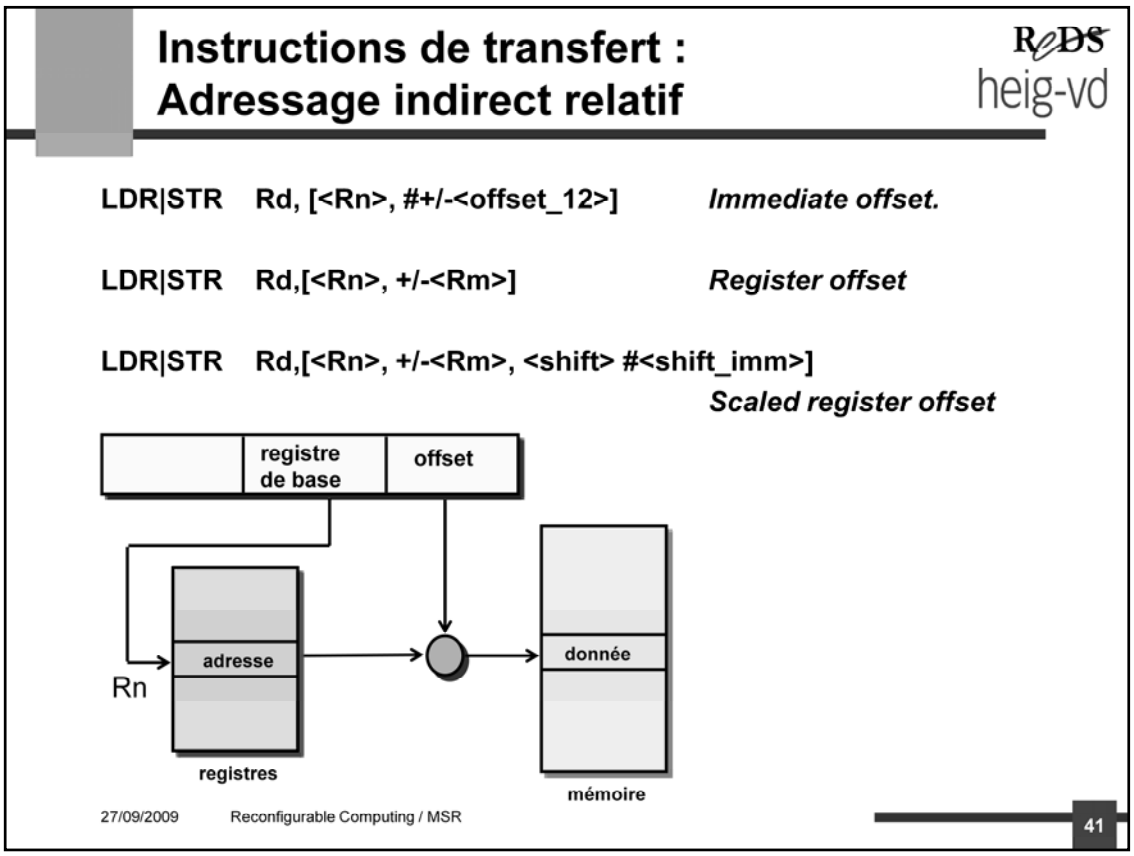
L'offset est un nombre non signé qui peut être ajouté ou soustrait à la valeur du base register . L'adressage immédiat de l'offset e est utile pour accéder aux éléments de données qui sont une adresse fixe par rapport au début d'une structure de donnnées

Direct par registre :

L'offset est la valeur d'un registre généraliste (pas le PC), qui peut être ajoutée ou soustraite à la valeur du registre de base. L'adressage relatif par registre est utile pour accéder à des tableaux ou des blocs de données.

Direct par registre avec décalage :

L'offset est la valeur d'un registre général (pas le PC) décalé par une valeur immédiate, puis ajouté ou soustrait à la valeur du registre de base. Toutes les opérations de décalage ou rotation peuvent être utilisées (Logical Shift Left, Logical Shift Right, Arithmetic Shift Right and Rotate Right), mais Logical Shift Left est la plus utile, permettant une mise à l'échelle de l'index d'un tableau en fonction de la taille de chaque élément du tableau (16bits => shitf 1 / x2 , 32bits shift 2 / x4 .....).



LDR|STR Rd, [<Rn>, #+/-<offset_12>] ***Immediate offset.***

L'adresse est calculée en ajoutant ou en soustrayant la valeur d'un offset (immédiat) à la valeur du registre Rn :

Rd ↔ mémoire [Rn +/- offset]

LDR|STR Rd,[<Rn>, +/-<Rm>] ***Register offset***

L'adresse est calculée en ajoutant ou en soustrayant la valeur du registre Rm à la valeur du registre Rn :

Rd ↔ mémoire [Rn +/- Rm]

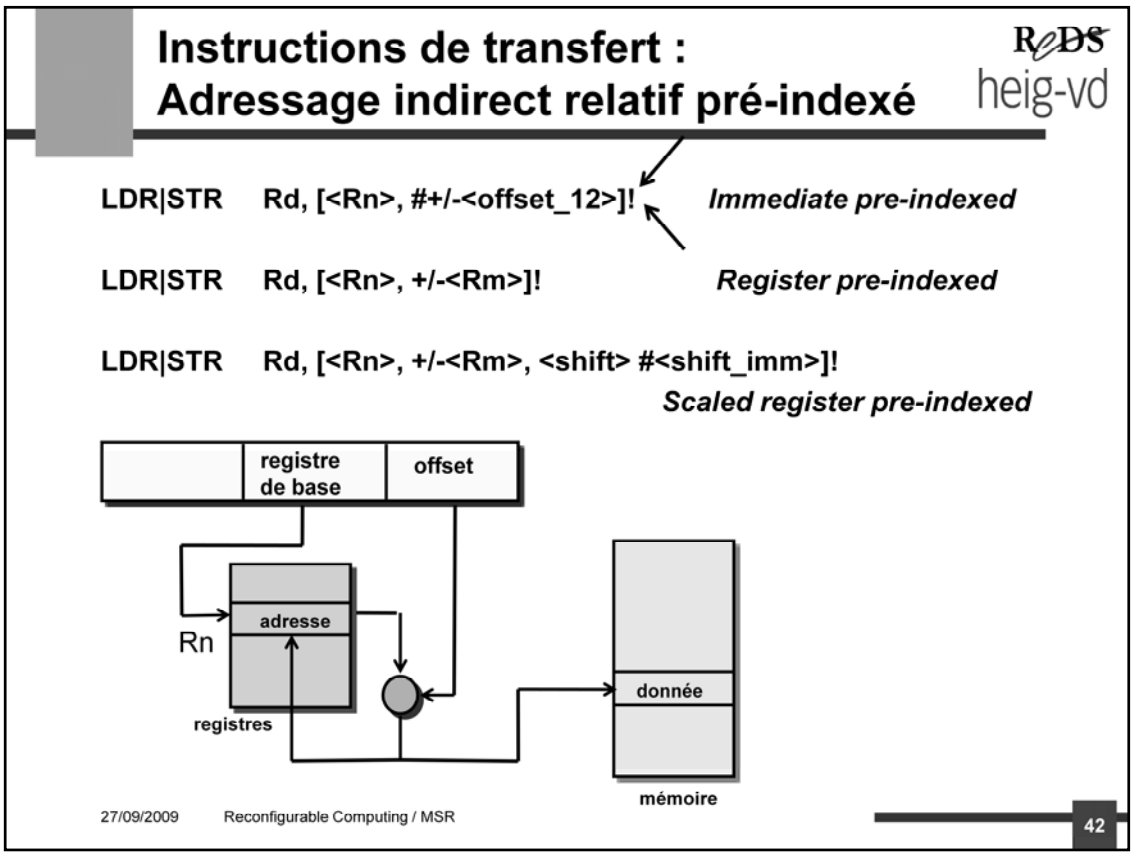
LDR|STR Rd,[<Rn>, +/-<Rm>, <shift> #<shift_imm>]

Scaled register offset

L'adresse est calculée en ajoutant ou en soustrayant la valeur du registre Rm shiftée de shift_imm à la valeur du registre Rn :

Rd ↔ mémoire [Rn +/- shift(Rm,shift_imm)]

Note : le flag W (write back) est à 0. La valeur du registre de base reste inchangée.



LDR|STR Rd, [<Rn>, #+/-<offset_12>]! ***Immediate offset pre-indexed***

L'adresse est calculée en ajoutant ou en soustrayant la valeur d'un offset (immédiat) à la valeur du registre Rn et en écrivant le résultat dans le registre Rn :

Rn ← Rn +/- offset

Rd ↔ mémoire [Rn]

LDR|STR Rd,[<Rn>, +/-<Rm>]! ***Register offset pre-indexed***

L'adresse est calculée en ajoutant ou en soustrayant la valeur du registre Rm à la valeur du registre Rn et en écrivant le résultat dans le registre Rn :

Rd ↔ Rn +/- Rm

Rd ↔ mémoire [Rn]

LDR|STR Rd,[<Rn>, +/-<Rm>, <shift> #<shift_imm>]!

Scaled register offset pre-indexed

L'adresse est calculée en ajoutant ou en soustrayant la valeur du registre Rm shiftée de shift_imm à la valeur du registre Rn et en écrivant le résultat dans le registre Rn :

Rd ↔ Rn +/- shift(Rm,shift_imm)

Rd ↔ mémoire [Rn]

Note : le flag W (write back) est à 1. La valeur du registre de base est changée.

Le flag P (pre-/post index) est à 1.

Instructions de transfert : Adressage indirect relatif post-indexé

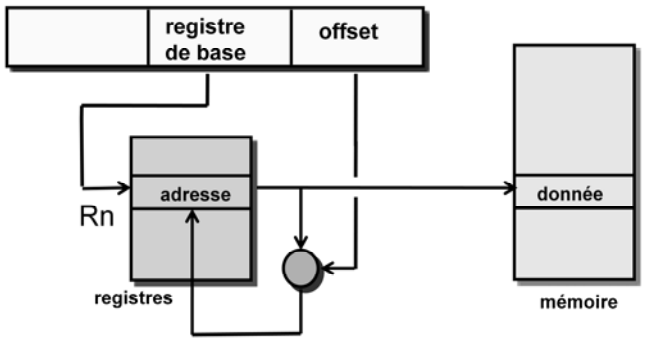
R&DS heig-vd

LDR|STR Rd, [<Rn>], #+/-<offset_12> ***Immediate post-indexed***

LDR|STR Rd,[<Rn>], +/-<Rm> ***Register post-indexed***

LDR|STR Rd, [<Rn>], +/-<Rm>, <shift> #<shift_imm>

Scaled register post-indexed



27/09/2009 Reconfigurable Computing / MSR 43

LDR|STR Rd, [<Rn>], #+/-<offset_12> ***Immediate offset post-indexed***

L'adresse mémoire utilisée est la valeur du registre Rn. Le contenu du registre Rn est ensuite modifié en ajoutant ou en soustrayant la valeur d'un offset (immédiat) à la valeur initiale du registre Rn:

Rd ↔ mémoire [Rn]

Rn ← Rn +/- offset

LDR|STR Rd,[<Rn>], +/-<Rm> ***Register offset post-indexed***

L'adresse mémoire utilisée est la valeur du registre Rn. Le contenu du registre Rn est ensuite modifié en ajoutant ou en soustrayant la valeur du registre Rm à la valeur du registre Rn :

Rd ↔ mémoire [Rn]

Rd ↔ Rn +/- Rm

LDR|STR Rd,[<Rn>], +/-<Rm>, <shift> #<shift_imm>

Scaled register offset post-indexed

L'adresse mémoire utilisée est la valeur du registre Rn. Le contenu du registre Rn est ensuite modifié ajoutant ou en soustrayant la valeur du registre Rm shiftée de shift_imm à la valeur du registre Rn :

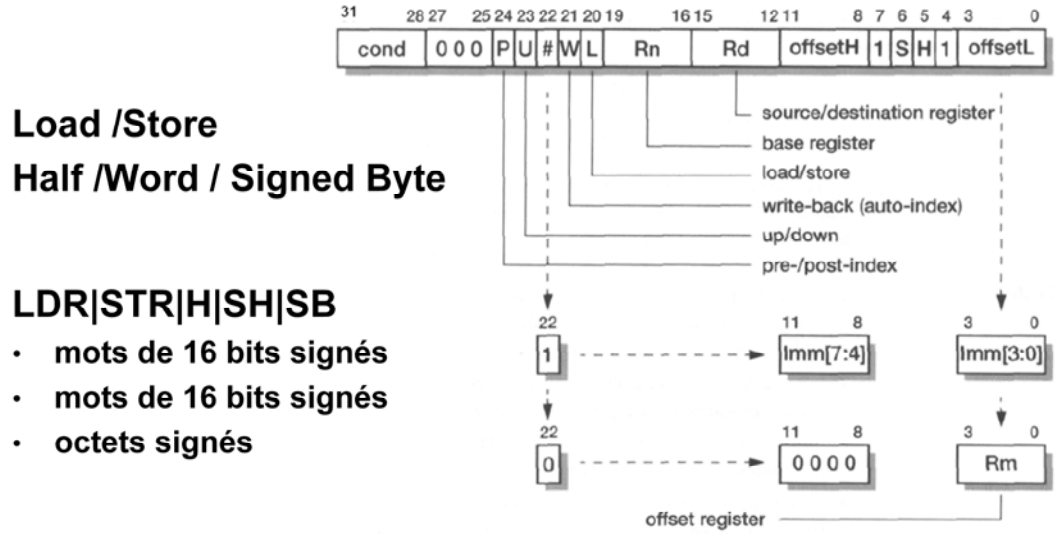
Rd ↔ mémoire [Rn]

Rd ↔ Rn +/- shift(Rm,shift_imm)

Note : le flag W (write back) est à 1. La valeur du registre de base est changée.

Le flag P (pre-/post index) est à 0.

V4T - Instructions de transfert autres formats de données



Load /Store
Half /Word / Signed Byte

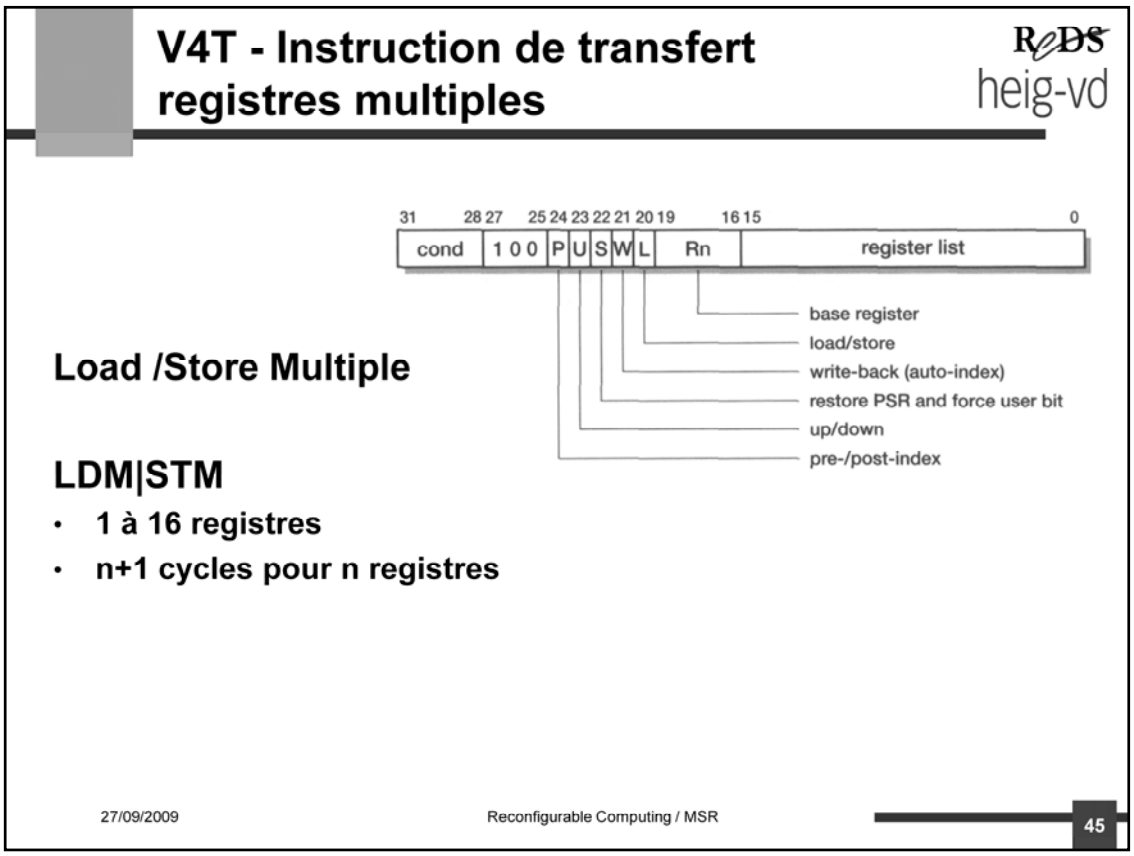
LDR|STR|H|SH|SB

- **mots de 16 bits signés**
- **mots de 16 bits signés**
- **octets signés**

Les instructions de transfert de type Load (LDR|H|SH|SB) ou Store (STR|H|SH|SB). supportent les transferts de mots de 16 bits (signés ou non signés) et d'octets signés.

H	unsigned half-word
SH	signed half-word
SB	signed bytes

Les instructions sont similaires aux instructions de transfert de mots de 32 bits décrites précédemment et ne seront pas détaillées ici.



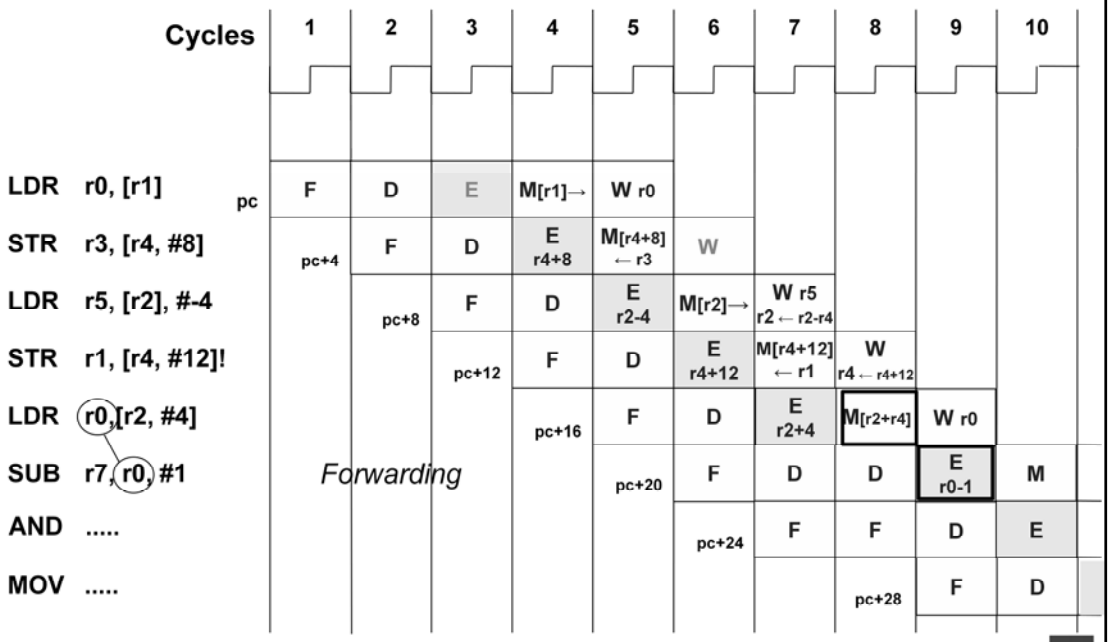
Les instructions de transfert de type Load multiple (LDM) et Store multiple (STM) supportent les transferts en une instruction pour plusieurs registres (1 à 16). Ces instructions permettent uniquement un adressage indirect par registre (Rn) avec pré ou post-indexage.

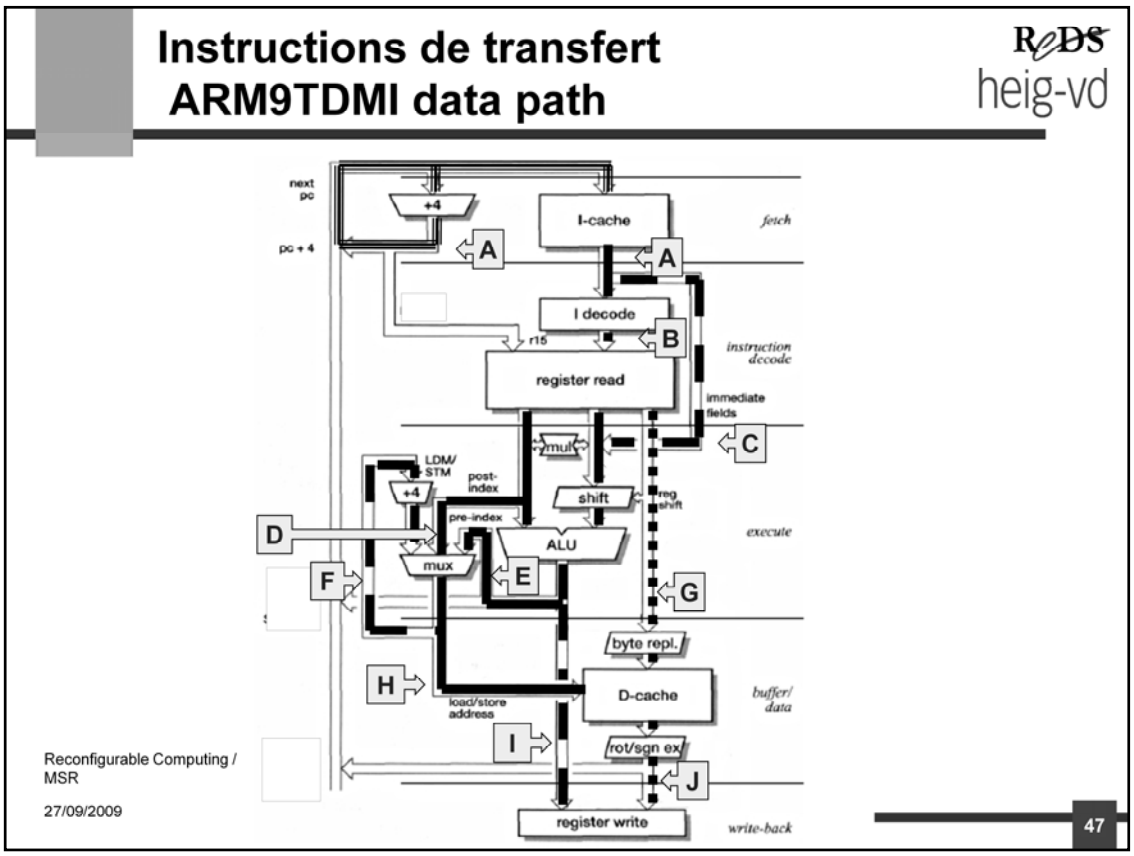
Les contenus des registres sont écrits ou lus à des adresses contigües dans un bloc de la mémoire. L'adresse mémoire est auto-incrémentée ou décrémentée de 4 à chaque transfert de registre.

1. IA Increment After
2. IB Increment Before
3. DA Decrement After
4. DB Decrement Before

⚠ Une forme spécifique de l'instruction permet la restauration du status register après un changement de mode.

Instructions de transfert pipeline





Chemin de données durant les 5 cycles d'une l'instruction de transfert

Note : pc représente l'adresse de l'instruction

A- Durant le cycle 1 / Fetch , l'instruction à l'adresse pc est lue dans la mémoire d'instruction

A'- Durant le même cycle le PC est incrémenté de 4 pour permettre la lecture de l'instruction suivante à l'adresse pc+4 au cycle suivant.

B- Durant le cycle 2 / Decode , l'instruction d'adresse pc est décodée. Si l'instruction comporte un offset d'adresse immédiat, la donnée de ce champ est acheminée à l'entrée de l'alu (après passage par un registre)

C- Durant le même cycle, l'adresse des registres opérandes est décodée . Les données contenues dans le registre de base et le registre d'offset seront présentes à l'entrée de l'alu au prochain cycle.

D- Durant le cycle 3- Execute , si l'adresse est préindexée, l'adresse contenue dans le registre de base est directement utilisée comme adresse de donnée au prochain cycle. Durant ce même cycle, l'offset est ajouté ou soustrait à cette adresse par l'alu.

E- Durant le cycle 3- Execute, si l'adresse est post indexée) l'offset est ajouté ou soustrait à l'adresse contenue dans le registre de base (dans l'alu). le résultat est directement utilisée comme adresse de donnée au prochain cycle

F- Durant le même cycle, l'adresse peut être auto incrémentée ou décrémentée (load et store multiple)

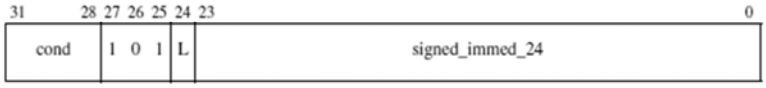
G- S'il s'agit d'un store, la donnée est acheminée à l'entrée de la mémoire de donnée

F- Durant le cycle 4 / Memory Access, l'adresse est présente à l'entrée de la mémoire de donnée et la donnée est lue (load) ou écrite (store)

I - Durant le cycle 5 / Write Back, suivant le type d'instruction, le registre de base peut être mis à jour avec le résultat du calcul dans l'alu

J- Durant ce même cycle, la donnée lue est chargée dans le registre de destination si l'instruction est de type load..

V4T - Instructions de branchement



Branch

B <target address>

Branch and Link

Sauvegarde du PC dans le link register (r14)

BL <target address>

Move to PC register

MOV PC, Rx

Si le flag L est à 1, le contenu du PC est chargé dans le link register R14 qui contient alors l'adresse de retour en cas d'appel de fonction. L'instruction MOV pc, r14 permet alors, en fin de fonction , le retour à l'adresse qui suit l'instruction de branchement.

Le champ <cond>, commun à toute les instructions, autorise les branchements conditionnels (14 possibilitées). Voir la slide "Instructions conditionnelles".

Calcul de l'adresse de branchement

Cette addresse est calculée à partir du champ 24 bit de l'instruction et de l'adresse contenue dans le PC :

1. Le contenu du champ (valeur signée) signed_immed_24 est étendu à 30 bits et décalé à gauche de 2 pour former une valeur sur 32 bits.
2. Le résultat est additionné au PC qui contient l'adresse de l'instruction de branchement plus 8.

Le branchement s'effectue à l'adresse PC + 8 + signed_immed_24 x 4

Le compilateur calcule le champ signed_immed_24 de la façon suivante :

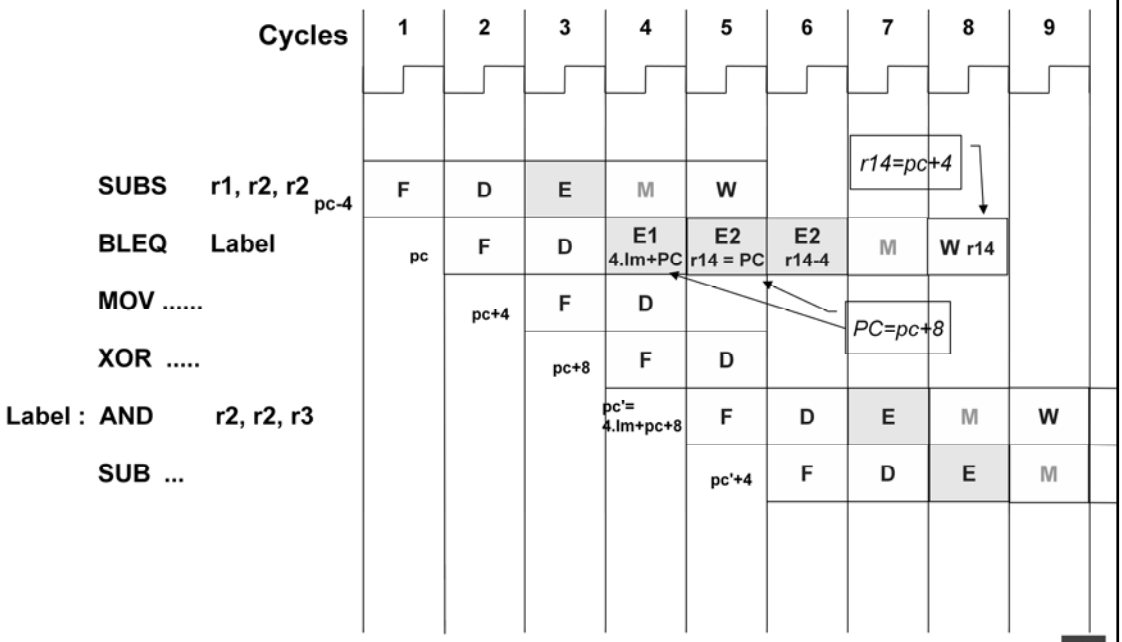
signed_immed_24 = (branch_address – instruction_address – 8) /4

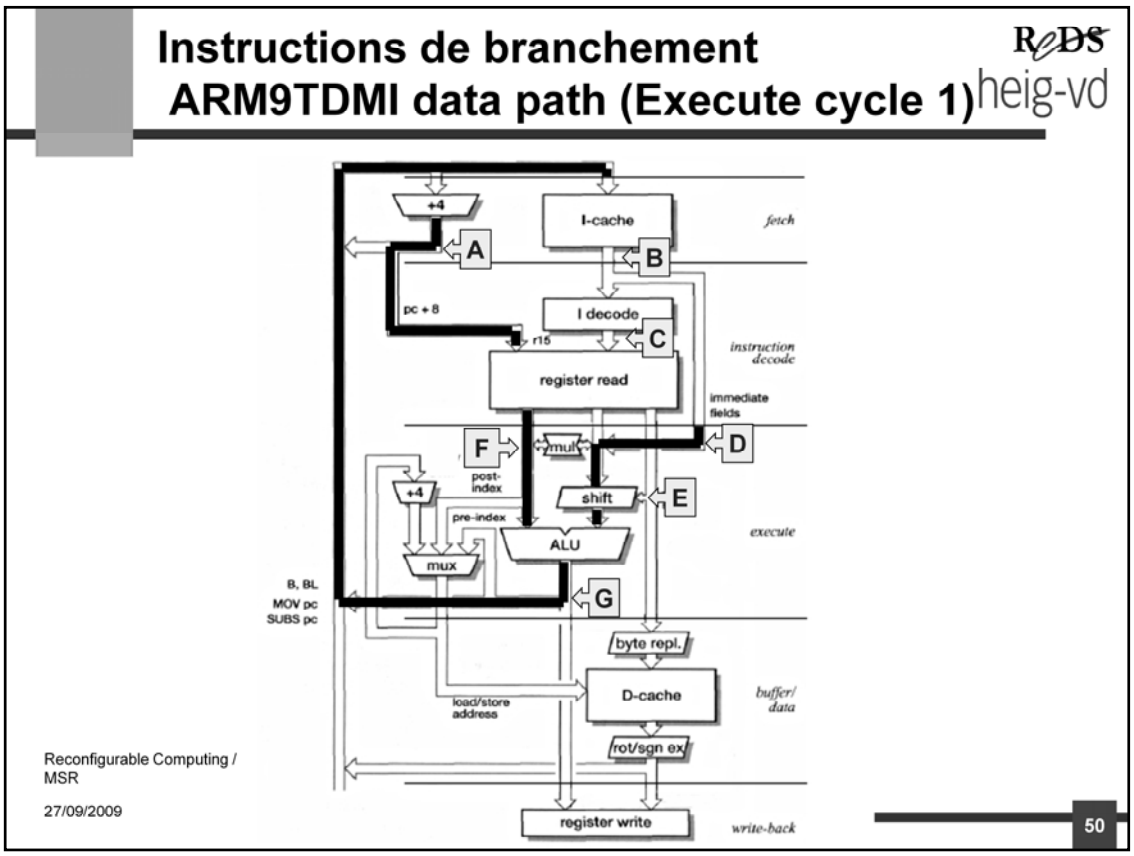
Ceci permet des sauts de ±32Mo (approximativement) par rapport à l'adresse de l'instruction de branchement. Si cette longueur de saut est insuffisante, il est possible d'utiliser l'instruction MOV PC,Rx qui autorise des sauts dans tout l'espace mémoire adressable (théoriquement 4Go).

Exemples :

```
B label          ; branch unconditionally to label
BCC label        ; branch to label if carry flag is clear
BEQ label        ; branch to label if zero flag is set
MOV PC, #0       ;branch to location zero
BL func          ; subroutine call to function
MOV PC, r14      ;return from subroutine
```

Instructions de branchement pipeline





Chemin de données durant le 1er cycle Execute de l'instruction de branchement

Note : pc représente l'adresse de l'intruction de branchement

A - Le PC contient pc+8

B- L'instruction traitée dans le cycle Fetch (en sortie de la mémoire d'instruction est l'instruction d'adresse pc+8

C- L'instruction traitée dans le cycle Decode est l'instruction d'adresse pc+4

D- L'offset d'adresse (signed_immed_24) de l'instruction de branchement est présent à l'entrée de l'alu

E- L'offset d'adresse (signed_immed_24) est shifté à gauche de 2 c'est à dire multiplié par 4 (adressage 32 bits)

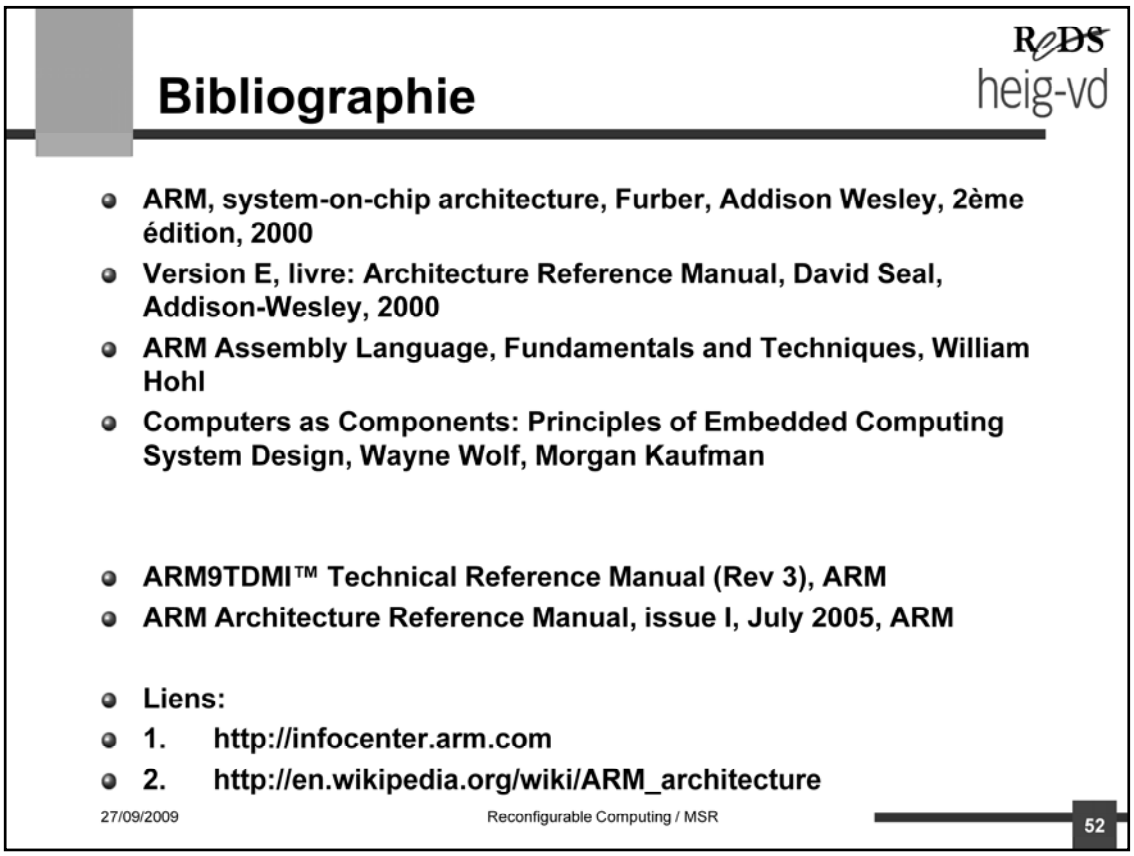
F- la valeur pc+8 est présente à l'entrée de l'alu

G- L'addition est effectuée. La sortie de l'alu a la valeur suivante :

pc+8+4 x signed_immed_24. Au prochain coup d'horloge, le PC prendra cette valeur .

Branchements condiftionnels

Branch	Interpretation	Normal uses
B BAL	Unconditional Always	Always take this branch Always take this branch
BEQ	Equal	Comparison equal or zero result
BNE	Not equal	Comparison not equal or non-zero result
BPL	Plus	Result positive or zero
BMI	Minus	Result minus or negative
BCC BLO	Carry clear Lower	Arithmetic operation did not give carry-out Unsigned comparison gave lower
BCS BHS	Carry set Higher or same	Arithmetic operation gave carry-out Unsigned comparison gave higher or same
BVC	Overflow clear	Signed integer operation; no overflow occurred
BVS	Overflow set	Signed integer operation; overflow occurred
BGT	Greater than	Signed integer comparison gave greater than
BGE	Greater or equal	Signed integer comparison gave greater or equal
BLT	Less than	Signed integer comparison gave less than
BLE	Less or equal	Signed integer comparison gave less than or equal
BHI	Higher	Unsigned comparison gave higher
BLS	Lower or same	Unsigned comparison gave lower or same



Articles :

The History of The ARM Architecture: From Inception to IPO
By Markus Levy, Convergence Promotions

ARM : Performance of the ARM9TDMI™ and ARM9E-S™ cores compared to the ARM7TDMI™ core