

CHAPITRE 3

Architecture de l'ensemble d'instructions ARM

Plan du Chapitre

Introduction

Les opérandes: les registres

Les opérations ARM

- Traitement de données
- Branchements
- Transfert de données
- Multiplication

Exécution conditionnelle

Introduction

ARM Holdings développe des architectures de micro-processeurs et des jeux d'instructions qui sont basé sur l'architecture RISC.

➤ L'architecture la plus utilisée au monde

Micro-processeurs ARM:

- Supportent 32 et 64 bits,
- Implémentation de petite taille qui favorise une consommation réduite d'énergie.

Jeux d'instructions:

- Deux jeux d'instructions sont supportés: Les instructions ont toutes 32 bits sauf si le jeu d'instructions Thumb est actif (*Les instructions du mode Thumb ont 16 bits pour économiser de la mémoire*),
- La plupart des instructions s'exécutent en un seul cycle,
- Exécution conditionnelle possible pour la plupart des instructions,
- Instructions qui combinent décalages et opérations logiques et arithmétiques,
- Les instructions Load et Store ont accès à la mémoire,
- Les opérandes des instructions ARM sont des registres qui se trouvent dans un banc de registre.

Les opérandes: les registres

Les opérandes des instructions ARM ne peuvent pas être des variables quelconques; ils doivent provenir d'un **banc de registre**.

- La taille d'un registre dans l'architecture ARM est de **32 bits**.

L'architecture repose sur 17 registres :

- 16 registres de 32 bits sont disponibles:
 - R0 à R12: usage général
 - R13 à R15: registres "spéciaux"
- Un registre (CPSR) "Current Program Status Register" est utilisé pour mémoriser les résultats des opérations (codé sur 32 bits).

Les opérandes: les registres

Le banc de registres est composé de :

- ✓ 13 registres de données génériques (**r0- r12**),
- ✓ Le registre spécifique **r13** qui est réservé comme pointeur de pile **SP** '*Stack Pointer*',
- ✓ Le registre spécifique **r14** qui est aussi le registre de retour de sous programme **LR** '*Link Register*'. Ce registre évite l'empilage du PC,
- ✓ Le registre spécifique **r15** est le compteur de programme **PC** '*Program Counter*'.
 - Indique où se trouve l'instruction en cours d'exécution,
 - Le changement de son contenu permet de faire un "branchement" dans le programme ou un retour de fonction.
- ✓ Le registre de statut **CPSR** (Current Program Status Register) contient des informations sur l'état du processeur.

User 32

r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)
r15 (pc)

cpsr

Les opérandes: le registre r13 (SP) :

- Par convention, on définit un bloc de mémoire qu'on nomme "pile" (stack).
- La pile est utilisée pour sauvegarder **temporairement** des valeurs de travail.

Exemples: variables locales, paramètres et valeurs de retour des fonctions

- Le Stack Pointer contient une adresse de la pile 'un bloc de mémoire' à laquelle on peut écrire ou lire une valeur.

Est-ce que le changement de la valeur de SP directement peut être dangereux?

- ✓ il ne faut pas perdre la trace de la pile, on procède plutôt en l'augmentant ("push") ou en la diminuant ("pop") à l'aide d'instructions dédiées.

Les opérandes: le registre r14 (LR) :

- Indique à partir d'où un appel d'une fonction a été fait,
- Sa copie dans R15 permet de poursuivre l'exécution du programme après celle d'une fonction.

Dans le cas où le programme assembleur indique un appel de fonction (ex: CALL maFonction)

maFonction:

- INST 1 ; première instruction
- ... ; liste d'instructions
- **RET** ; "return", indique que la fonction est terminée


Il suffirait par modifier PC pour le mettre à l'adresse correspondant aux instructions de "maFonction".

Que faire lorsqu'on parvient à **RET ?**
Où doit-on revenir ?



Les opérandes: le registre r14 (LR) :


Un appel de fonction (sous programme) à l'aide d'instructions dédiées entraîne un changement de la valeur du PC (Program Counter),



Un retour de fonction ne peut se faire que si PC reprend sa valeur avant l'appel de la fonction,



Par convention, le registre LR est utilisé pour sauvegarder l'adresse de retour pendant l'exécution d'une fonction,



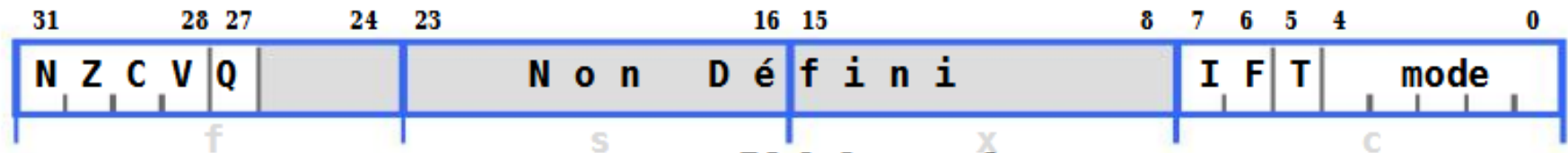
Mettre la valeur de LR dans PC permet de poursuivre l'exécution du programme principale,



Utiliser LR à d'autres fins ne peut que causer des problèmes si on n'a pas le plein contrôle du programme.

Registre de statut CPSR

Un registre de statut CPSR décrit l'état du processeur:



Indicateurs conditionnels

- ☐ N = Résultat Négatif de l'ALU
- ☐ Z = Résultat nul de l'ALU (Zéro)
- ☐ C = Retenue (Carry)
- ☐ V = Débordement (oVerflow)

Q = débordement avec mémoire

- ☐ Architecture 5TE seulement
- ☐ Indique qu'un débordement s'est produit pendant une série d'opérations

Validation des interruptions

- ☐ I = 1 dévalide IRQ.
- ☐ F = 1 dévalide FIQ.

Mode Thumb

- ☐ Architecture xT seulement
- ☐ T = 0, Processeur en mode ARM
- ☐ T = 1, Processeur en mode Thumb

Indicateurs de mode

- ☐ Indiquent le mode actif : système, IRQ, FIQ, utilisateur...

Registre de statut CPSR

- Les bits de poids fort représentent les indicateurs de l'UAL [R\[31-28\]](#):
 - **N** : Détection de signe négatif
N=1 si résultat < 0 (le bit de signe égal à 1); 0 autrement
 - **Z** : Détection de zéro
Z=1 si résultat = 0; 0 autrement
=> *Souvent utilisé pour détecter l'égalité*
 - **C** : Détection de retenue ("carry") génère par l'ALU dans le cas des instructions arithmétiques et de décalage (shifter) dans le cas des instructions logiques.
C=1 si l'opération a impliqué une retenue
 - **V** : Dépassement de capacité dans le cas d'une opération arithmétique signée.
Détection de dépassements (overflow)
V=1 si l'opération a impliqué un dépassement
- Les bits 6 et 7 sont des bits de contrôle des interruptions [R\[7-6\]](#).
- Le bit 5 permet de passer en mode Thumb [R\[5\]](#).
- Les 5 bits de faible poids indiquent le mode opératoires [R\[4-0\]](#).

Registre de statut CPSR - modes opératoires R[4-0]

Un microprocesseur ARM a 7 modes opératoires de base :

mode	Fonction	code
USER	Mode dans lequel la plupart des programmes et applications s'exécutent	1 0000
FIQ	traitement des interruptions prioritaires	1 0001
IRQ	traitement des interruptions ordinaires	1 0010
Supervisor	reset et traitement interruptions logicielles	1 0011
Abort	Saisi sur une exception d'accès à la mémoire	1 0111
Undefined	Saisi lorsqu'une instruction indéfinie est exécutée; erreur décodage instruction	1 1011
System	Mode dans lequel le système d'exploitation s'exécute, partageant la vue du registre avec le mode utilisateur	1 1111

➤ Généralement, le système utilise le jeu de registres du mode Utilisateur '**User**'.

Les opérations ARM

Il est évident que tout processeur ARM doit être capable d'effectuer une opération à la suite d'une décision prise par l'unité de contrôle.

Types d'instructions:

Les instructions ARM sont organisées en catégories :

- ❖ Traitement de données : arithmétiques, logiques, comparaison...
- ❖ Branchements,
- ❖ Transfert de données: Load et Store
- ❖ Multiplication..

Exemple: en notation ARM, l'instruction **add** permet d'additionner deux opérandes (b et c) et mettre le résultat dans un autre opérande (a) suivant la forme suivante:



Les opérations ARM: Traitement de données

Opérations arithmétiques:

Mnemonic	Instruction	Action
SUB Rd,Rn,Op2	Subtract	$Rd \leftarrow Rn - Op2$
RSB Rd,Rn,Op2	Reverse Subtract	$Rd \leftarrow Op2 - Rn$
ADD Rd,Rn,Op2	Add	$Rd \leftarrow Rn + Op2$
ADC Rd,Rn,Op2	Add with carry	$Rd \leftarrow Rn + Op2 + C$
SBC Rd,Rn,Op2	Subtract with Carry	$Rd \leftarrow Rn - Op2 + C - 1$
RSC Rd,Rn,Op2	Reverse Subtract with Carry	$Rd \leftarrow Op2 - Rn + C - 1$

Opérations logiques:

Mnemonic	Instruction	Action
AND Rd,Rn,Op2	AND	$Rd \leftarrow Rn \text{ AND } Op2$
XOR Rd,Rn,Op2	XOR	$Rd \leftarrow Rn \text{ XOR } Op2$
ORR Rd,Rn,Op2	OR	$Rd \leftarrow Rn \text{ OR } Op2$
BIC Rd,Rn,Op2	Bit Clear	$Rd \leftarrow Rn \text{ AND NOT } Op2$
EOR Rd,Rn,Op2	Exclusive OR	$Rd \leftarrow (Rn \text{ AND NOT } Op2) \text{ OR } (Op2 \text{ AND NOT } Rn)$

Les opérations ARM: Traitement de données

Comparaisons:

Mnemonic	Instruction	Action
CMP Rn,Op2	Compare	Positionne les Flags de registre CPSR pour Rn - Op2
CMN Rn,Op2	Compare Negative	Positionne les FLags de registre CPSR pour Rn + Op2

Mouvement de registres:

Mnemonic	Instruction	Action
MVN Rd, Op2	Move negative register	Rd <= Not Op2
MOV Rd, Op2	Move register or constant	Rd <= Op2
MOV Pc, LR	@fin de sous-programme	Pc <= LR

Exemples:

L'instruction MOV permet d'effectuer des transferts à destination de registres :

MOV r3,#4

Le PC peut être utilisé comme destination :

MOV r15 ,r14

ou

MOV PC, LR

Les opérations ARM

Branchements:

Mnemonic	Instruction	Action
B Label	Branch	R15 (PC) <= address label
BL Label	Branch with Link	R14 <= R15, R15 (PC) <= address label
BX Label	Branch and Exchange	R15 (PC) <= Rn, T_bit <= Rn[0] (<i>T c'est le bit Numéro 5 R[5] de registre CPSR</i>)

Multiplication:

Mnemonic	Instruction	Action
MUL Rd,Rm,Rs	Multiply	Rd <= Rm * Rs
MLA Rd,Rm,Rs,Rn	Multiply Accumulate	Rd <= (Rm * Rs) + Rn
MLS Rd,Rm,Rs,Rn	Multiply-Subtract	Rd <= (Rm * Rs) - Rn

Transfert des données:

Mnemonic	Instruction	Action
LDR	Load register from memory	Rd <= (address)
LDM	Load multiple registers	Stack manipulation (Pop)
STR	Store register to memory	<address> <= Rd
STM	Store Multiple	Stack manipulation (Push)

Les opérations ARM: Transfert de données

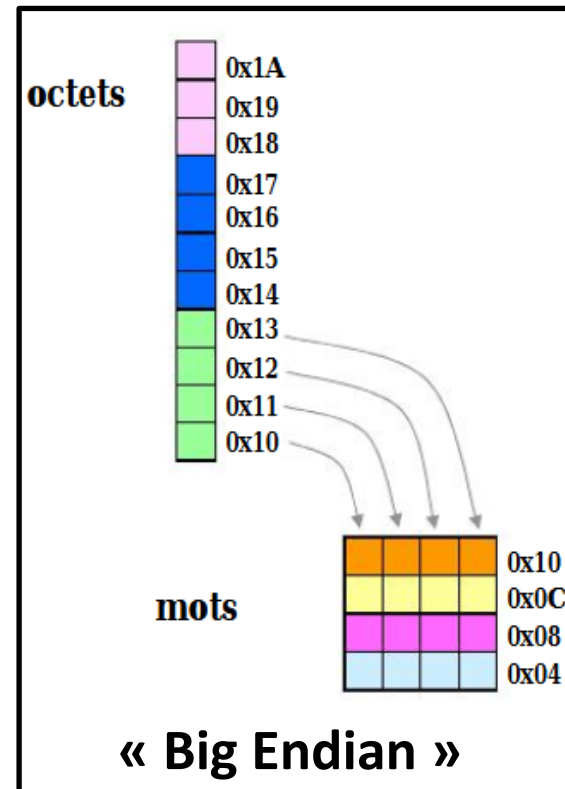
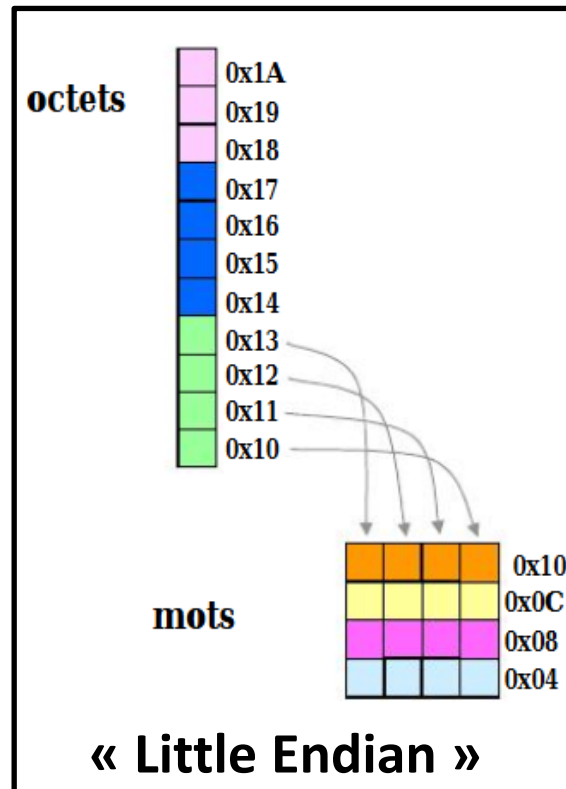
Il existe 2 instructions d'accès à la mémoire: LOAD (**LDR**) et STORE (**STR**)

- L'adressage mémoire se fait sur 32 bits.
- Le type des données peut être:
 - **Word** (mots) signifie 32 bits = quatre octets (**LDR, STR**)
 - **Halfword** (demi-mots) signifie 16 bits = deux octets (**LDRH, LDRSH, STRH**)
 - **Byte** signifie 8 bits = un octet (**LDRB, LDRSB, STRB**)
- ✓ **LDRB (Load Byte)** : charge un octet (8 bits) depuis la mémoire dans les bits de poids faible d'un registre de destination.
- ✓ **LDRH (Load Halfword)** : charge un demi-mot (16 bits) depuis la mémoire dans un registre de destination.
- Par défaut, les bits restants sont mis à zéro pour obtenir une valeur de 32 bits.
- ✓ **LDRSB (Load Signed Byte)** : étend le signe d'un octet chargé à 32 bits.
- ✓ **LDRSH (Load Signed Halfword)** : étend le signe d'un demi-mot chargé à 32 bits.
- ✓ **STRB (Store Byte)** : sert à stocker un octet (8 bits) provenant d'un registre dans une adresse mémoire.
- ✓ **STRH (Store Halfword)** : est utilisée pour stocker un demi-mot (16 bits inférieurs) d'un registre source vers une adresse mémoire (=contenu d'un registre de destination).

Les opérations ARM: Transfert de données

Organisation de la mémoire

- La taille d'une donnée stockée en mémoire est 1 seul octet = 8 bits.
- Les données ayant plusieurs octets (2-4 octets) peuvent être stockées de 2 façons:



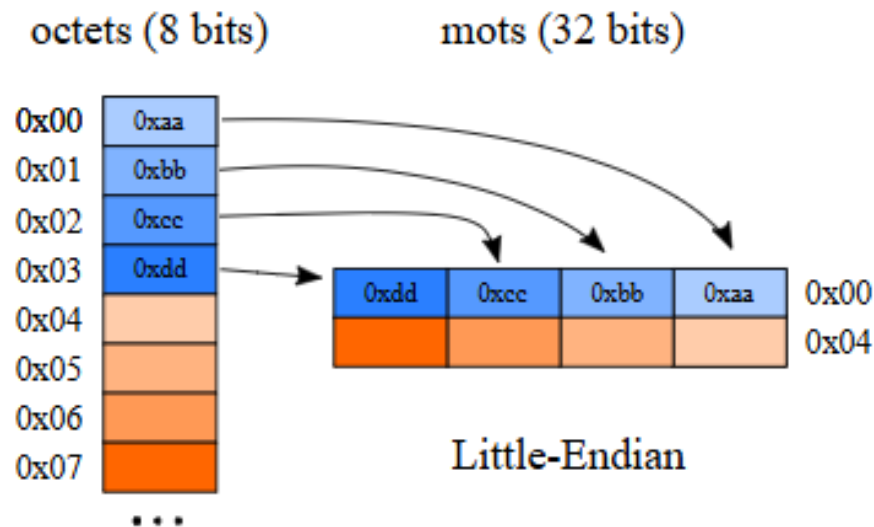
“Little endian”: l’octet le moins significatif est placé à la plus petite adresse dans la mémoire.
“Big endian”: l’octet le moins significatif est placé à la plus haute adresse dans la mémoire.

Les opérations ARM: Transfert de données

Organisation de la mémoire

La majorité des processeurs ARM sont little-endian (*octet de poids faible d'un mot de 32 bits est placé à l'adresse la plus basse*).

Exemple:



Notez aussi qu'il y a une contrainte d'alignement dans **le banc de registre** et que les adresses des mots de 32 bits sont multiples de 4.

- le premier mot se trouve à l'adresse 0,
- le deuxième à l'adresse 4,
- le troisième à l'adresse 8,
- le quatrième à l'adresse 12 (0xc) et ainsi de suite.

Les opérations ARM: Transfert de données

Pre/Post-indexed

■ Avec LOAD (**LDR**)

Pre-indexed	LDR R0, [R1, R2]	@R0 = mem[R1+R2] R1 unchanged
Auto-indexing	LDR R0, [R1, R2] !	@R0 = mem[R1+R2] R1 = R1+R2
Post-indexed	LDR R0, [R1], R2	@R0 = mem[R1] R1 = R1+R2

■ Avec STORE (**STR**)

Pre-indexed	STR R0, [R1, R2]	mem[R1+R2] = R0 R1 unchanged
Auto-indexing	STR R0, [R1, R2] !	mem[R1+R2] = R0 R1 = R1+R2
Post-indexed	STR R0, [R1], R2	mem[R1] = R0 R1 = R1+R2

Dans les instructions de transfert, le registre entre crochet [**Rx**] indique qu'il s'agit d'un registre d'adresse servant à pointer une donnée en mémoire.

Exemple: instruction LDR

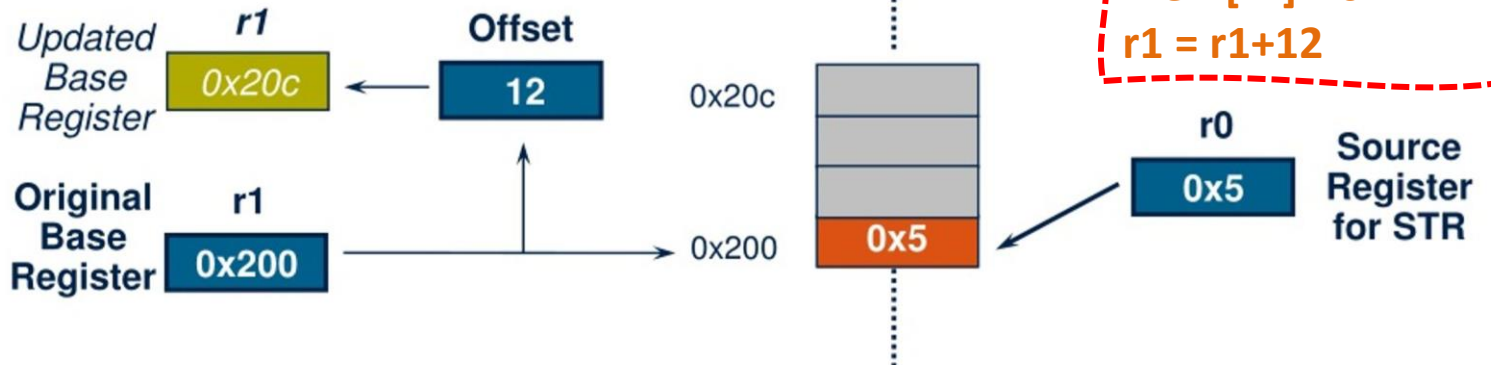
LDR	R0, [R1, #4]	@R0=[R1+4]
LDR	R0, [R1, #4] !	@R0=[R1+4] puis R1=R1+4
LDR	R0, [R1], #4	@R0=[R1] puis R1=R1+4

Exemple: instruction STR

- Pre-indexed: **STR r0, [r1, #12]**



- Post-indexed: **STR r0, [r1], #12**



Application :

Donner l'état de chaque registre et les cases mémoire après l'exécution de chaque instruction :

MOV **r0**, #0x00000002

MOV **r1**, #0x00000001

LDR **r3**, [**r0**]

LDRB **r5**, [**r1**], #2

LDRSB **r6**, [**r0**], #2

LDRH **r8**, [**r1**, **r0**]

LDRSH **r7**, [**r1**, **r0**]!

LDRSB **r4**, [**r1**], #6

STR **r4**, [**r2**]

STRB **r6**, [**r0**], **r2**

STRB **r7**, [**r0**, **r0**]

STRH **r3**, [**r0**, #2]

STRH **r5**, [**r0**, **r2**]!

STRB **r1**, [**r2**, #9]!

Mémoire de données

adresse	contenu
0x00000009	0x8F
0x00000008	0x04
0x00000007	0xB4
0x00000006	0x7D
0x00000005	0x01
0x00000004	0xC2
0x00000003	0x00
0x00000002	0xA1
0x00000001	0x12
0x00000000	0x06

Banc de registres

adresse	contenu
r0	0x00 00 00 00
r1	0x00 00 00 00
r2	0x00 00 00 00
r3	0x00 00 00 00
r4	0x00 00 00 00
r5	0x00 00 00 00
r6	0x00 00 00 00
r7	0x00 00 00 00
r8	0x00 00 00 00

Correction:

Mémoire de données

0x00000009	0x8F
0x00000008	0x04
0x00000007	0xB4
0x00000006	0x7D
0x00000005	0x01
0x00000004	0xC2
0x00000003	0x00
0x00000002	0xA1
0x00000001	0x12
0x00000000	0x06

Banc de registres

r0	0x00 00 00 02 0x00 00 00 04
r1	0x00 00 00 01 0x00 00 00 03 0x00 00 00 07 0x00 00 00 0D
r2	0x00 00 00 00
r3	0x01 C2 00 A1
r4	0xFF FF FF B4
r5	0x00 00 00 12
r6	0xFF FF FF A1
r7	0x00 00 04 B4
r8	0x00 00 04 B4

Instruction	RTL
MOV r0, #2	r0 = 0x00000002
MOV r1, #1	r1 = 0x00000001
LDR r3, [r0]	@r3 = mem[r0] = mem[2] r3 = 0x01 C2 00 A1
LDRB r5, [r1], #2	@r5 = mem[r1] = mem[1] r5 = 0x00 00 00 12 r1 = r1+2 r1 = 0x00 00 00 03
LDRSB r6, [r0], #2	@r6 = mem[r0] = mem[2] r6 = 0xFF FF FF A1 r0 = r0+2 r0 = 0x00 00 00 04
LDRH r8, [r1, r0]	@r8 = mem[r1+r0] = mem[7] r8 = 0x00 00 04 B4
LDRSH r7, [r1, r0]!	@r7 = mem[r1+r0] = mem[7] r7 = 0x00 00 04 B4 r1 = r1+r0 r1 = 0x00 00 00 07
LDRSB r4, [r1], #6	@r4 = mem[r1] = mem[7] r4 = 0xFF FF FF B4 r1 = r1+6 {r1=(13) ₁₀ =(D) ₁₆ } r1 = 0x00 00 00 0D

Correction:

Mémoire de données

0x00000009	0x0D
0x00000008	0xB4
0x00000007	0x00
0x00000006	0xA1
0x00000005	0x00
0x00000004	0xA1 0x12
0x00000003	0xFF
0x00000002	0xFF
0x00000001	0xFF
0x00000000	0xB4

Banc de registres

r0	0x00 00 00 04
r1	0x00 00 00 0D
r2	0x00 00 00 00 0x00 00 00 09
r3	0x01 C2 00 A1
r4	0xFF FF FF B4
r5	0x00 00 00 12
r6	0xFF FF FF A1
r7	0x00 00 04 B4
r8	0x00 00 04 B4

STR r4,[r2]

mem[r2] = r4
mem[0] = 0xB4
mem[1] = 0xFF
mem[2] = 0xFF
mem[3] = 0xFF

STRB r6,[r0], r2

mem[r0] = r6
mem[4] = 0xA1
r0 = r0+r2
r0 = 0x00 00 00 04

STRB r7,[r0, r0]

mem[r0 + r0] = r7
mem[8] = 0xB4

STRH r3,[r0, #2]

mem[r0+2] = r3
mem[6] = 0xA1
mem[7] = 0x00

STRH r5,[r0, r2]!

mem[r0+r2] = r5
mem[4] = 0x12
mem[5] = 0x00
r0 = r0+r2
r0 = 0x00 00 00 04

STRB r1,[r2, #9]!

mem[r2+9] = r1
mem[9] = 0x0D
r2 = r2+9
r2 = 0x00 00 00 09

Application 1:

1. Expliquer brièvement les instructions suivantes,
2. Donner le niveau RTL des instructions,
3. Donner l'état de chaque registre y compris le registre d'état CPSR après l'exécution de chaque instruction séparée.

Adresse	Instruction	RTL
0x00000000	MOV r1, #3	
0x00000004	MOV r2, #1	
0x00000008	ADD r3, r1, r2	
0x0000000c	B Prog	
0x00000010	Prog2	
0x00000014	MOV r10, r15	
0x00000018	SUB r6, r2, r1	
0x0000001c	MOV r7, #10	
0x00000020	CMP r6, r7	
0x00000024	BNE Fin	
0x00000028	Prog	
0x0000002c	MUL r4, r2, r3	
0x00000030	ORR r6, r2, #1	
0x00000034	MLA r8, r3, r4, r4	
0x00000038	BL Prog2	
0x0000003c	Fin	

Exécution conditionnelle

L'exécution conditionnelle des instructions ARM est une fonctionnalité clé des processeurs ARM, permettant une plus grande efficacité dans l'exécution du code.

- L'exécution de, presque, toutes les instructions peut être conditionnée par la valeur des flags (N,Z,C,V) de registre CPSR.
- Les branchements peuvent être conditionnés ce qui est très commun.

Exemple:

*Le branchement (**B** en langage assembleur) devient **BEQ** pour «Branch if Equal», ce qui signifie que le branchement ne sera pris que si l'indicateur **Z** est défini.*

- La plupart des instructions peuvent préciser leur action sur les drapeaux (*suffixe S*), ce qui permettra la mise à jour des flags de registre CPSR à la suite de l'exécution de l'instruction.

Exécution conditionnelle

- Chaque instruction peut inclure une condition de son exécution. Si cette condition n'est pas remplie, l'instruction est ignorée.
 - Cela permet de réduire le nombre de branchements et d'améliorer l'efficacité du programme en limitant les sauts conditionnels et en utilisant des instructions conditionnelles à la place.

Exemple :

```
CMP      r0,#8    ; r0=8?
```

```
BEQ fin    ; si oui (Z=1) PC=fin
```

```
ADD      r1,r1,#4 ;
```

```
fin
```

Équivalent à

```
CMP      r0,#8    ; r0=8?
```

```
ADDNE    r1,r1,#4  ; si non (Z=0)
```

```
fin
```

} **+ petit et
+ rapide**

Exécution conditionnelle

Il existe 16 conditions d'exécution des instructions en utilisant les flags de registre CPSR :

Condition	Signification	Flags CPSR
EQ	Equal	Z=1
NE	Not Equal	Z=0
HS/CS	Higher or Same/Carry Set (Supérieur ou égal) { <i>unsigned</i> }	C=1
LO/CC	Lower/Carry Clear (strictement inférieure) { <i>unsigned</i> }	C=0
MI	Minus « - »	N=1
PL	Plus « + »	N=0
VS	Overflow Set	V=1
VC	Overflow Clear	V=0

- Lors d'une soustraction **sans emprunt**, le drapeau (Flag) **C** est mis à 1.
- Lors d'une soustraction **avec emprunt**, le drapeau (Flag) **C** est mis à 0. ²⁷

Exécution conditionnelle

Condition	Signification	Flags CPSR
HI	Higher (Strictement supérieur) { <i>unsigned</i> }	C=1 et Z=0
LS	Lower or Same (Inférieur ou égal) { <i>unsigned</i> }	C=0 ou Z=1
GE	Greater or Equal (Supérieur ou égal)	N=V
LT	Less Than (strictement inférieur)	N≠V
GT	Greater Than (Strictement supérieur)	N=V et Z=0
LE	Less or Equal (Inférieur ou égal)	N≠V ou Z=1
AL	Always (toujours)	---
NV	Never (réservé)	---

- **AL** signifie que l'instruction sera toujours exécutée, quelle que soit les valeurs des drapeaux dans le registre d'état CPSR.
- **NV** indique que l'instruction ne doit jamais être exécutée. En pratique, il n'est pas utilisé dans les instructions courantes.

Exécution conditionnelle

Exemple 1:

Il est possible de conditionner une instruction **MOV** pour quelle ne soit exécutée que si le flag **Z** est égal à 1, précisément, si l'instruction précédente a produit un résultat égal à 0 :

MOV r3,#4

Devient dans ce cas : **MOVEQ r3,#4**

Exemple 2:

CMP r1, r2 ; calcule $r1 - r2$, change les drapeaux de reg CPSR

MOVGT r0, r3 ; si $r1$ est supérieur à $r2$, $r0 \leq r3$

MOVLE r0, r4 ; si $r1$ est inférieur ou égal à $r2$, $r0 \leq r4$

BLE label1 ; si $r1$ est inférieur ou égal à $r2$, branche vers label1

BGE label2 ; si $r1$ est supérieur ou égal à $r2$, branche vers label2

Application 2:

Écrivez un programme qui:

- Charge les valeurs 1 et 2 dans les registres R0 et R1, respectivement.
- Place le résultat de $R0 + R1$ dans R2, si $R0 = R1$;
- Place le résultat de $R0 - R1$ dans R2, si $R0 \neq R1$;
- Place le résultat de $R0 * R1$ dans R2, si $R0 \geq R1$;

Application 2:

MOV R0, #1

MOV R1, #2

CMP R0, R1

; comparaison -- mets à jour les drapeaux de CPSR

; saute au bon endroit en fonction du résultat de la comparaison

BEQ egal

BNE pasegal

BGE GreaterorEqual

egal

ADD R2, R0, R1

B fin

pasegal

SUB R2, R0, R1

B fin

GreaterorEqual

MUL R2, R0, R1

B fin

fin