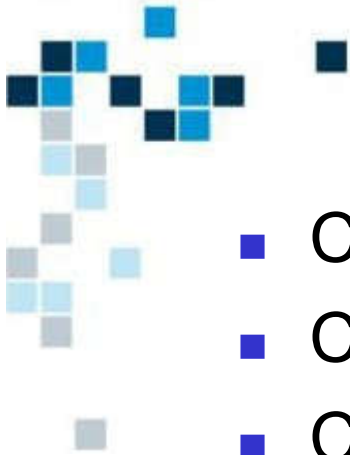


Systemes à Microprocesseurs

Cycle Ingénieur Troisième Année

Sébastien Bilavarn



Plan

- Ch1 – Représentation de l'information
- Ch2 – ARM Instruction Set Architecture
- Ch3 – Accès aux données
- Ch4 – Programmation structurée
- Ch5 – Cycle d'exécution
- **Ch6 – Codage binaire**
- Ch7 – Microcontrôleur ARM Cortex-M

La représentation en mémoire

- Codage binaire des instructions
 - Programme
- Exécution séquentielle et rupture de séquence
- Instructions de branchement relatif



Codage binaire d'une instruction

- Une instruction représente une opération élémentaire du processeur
 - L'ensemble des instructions disponibles s'appelle le jeu d'instruction
 - Chaque instruction définit un traitement précis opéré par la machine
- En assembleur, l'instruction s'écrit sous la forme d'un mot-clé suivi de ses opérandes.
 - On utilise des mnémoniques pour des raisons de commodité (faciliter l'écriture de programmes assembleur)
 - Le processus d'assemblage transforme ensuite la séquence de mnémoniques instructions en séquence de codes binaires interprétable par la machine.
- Pour le processeur, une instruction est codée par un mot de 32 bits (codage binaire).
 - Chaque bit de l'instruction 32 bit a un rôle précis



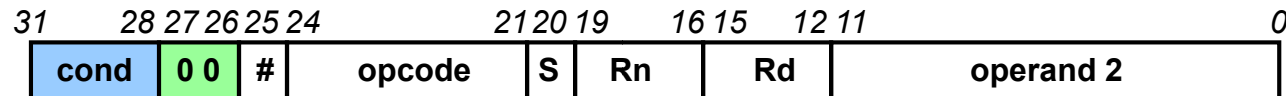
Codage binaire d'une instruction

- Une instruction de traitement est composée des champs
 - Registre résultat
 - Rd: R0...R15
 - Premier opérande (toujours un registre)
 - Rn: R0...R15
 - Deuxième opérande
 - Rm: R0...R15
 - Rm: R0...r15 avec décalage (log/arith/rot, quantité)
 - Une valeur immédiate (constante)
 - Opération à réaliser par l'UAL
 - Opcode
 - Affecte le registre d'état CPSR
 - Suffixe S
 - Condition
 - Exécution conditionnelle (EQ, LE, etc.)



Codage binaire

- Instructions de traitement (1)

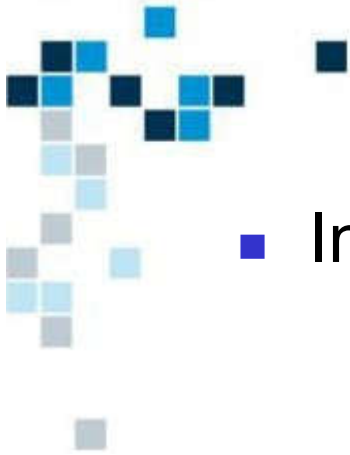


- *Cond*: l'instruction est exécutée si le registre d'état CPSR vérifie la condition spécifiée

Asm	Cond
EQ	0000
NE	0001
CS/HS	0010
CC/LO	0011
MI	0100
PL	0101

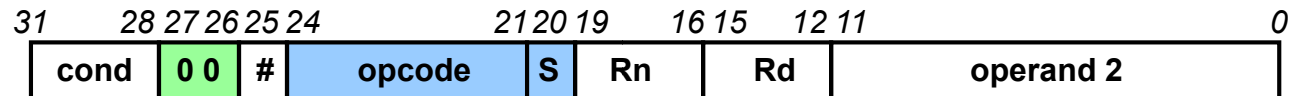
Asm	Cond
VS	0110
VC	0111
HI	1000
LS	1001
GE	1010
LT	1011

Asm	Cond
GT	1100
LE	1101
AL	1110
NV	1111



Codage binaire

- Instructions de traitement (2)



- $S = 1$: affecte CPSR
- *Opcode* : code de l'opération à effectuer

Asm	Opcode
AND	0000
EOR	0001
SUB	0010
RSB	0011
ADD	0100
ADC	0101

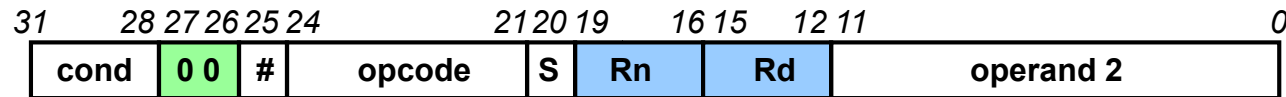
Asm	Opcode
SBC	0110
RSC	0111
TST	1000
TEQ	1001
CMP	1010
CMN	1011

Asm	Opcode
ORR	1100
MOV	1101
BIC	1110
MVN	1111



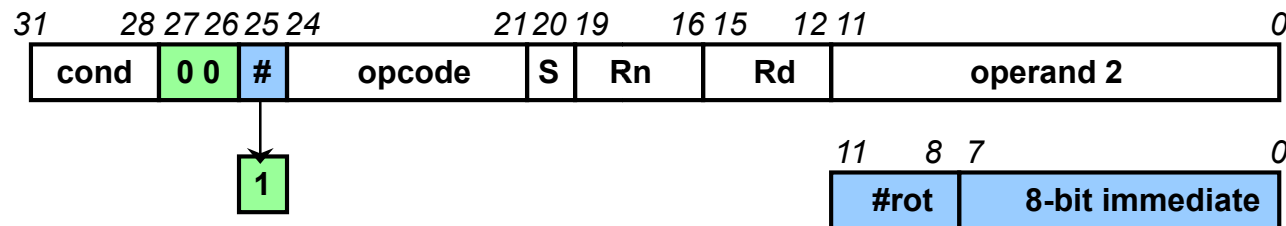
Codage binaire

■ Instructions de traitement (3)



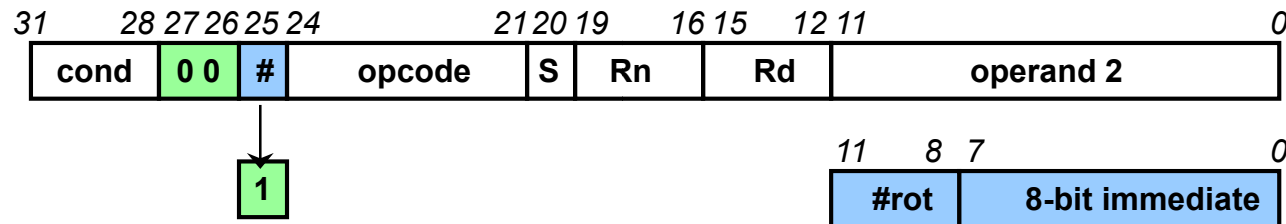
- $Rd \rightarrow$ numéro du registre de destinations
 - 4 bits pour sélection parmi les 16 registres possibles
- $Rn \rightarrow$ numéro du registre qui sert de premier opérande
 - 4 bits pour sélection parmi les 16 registres possibles
- $operand2 \rightarrow$ relié à l'entrée B
 - possibilité de rotation/décalage

■ Instructions de traitement (4)

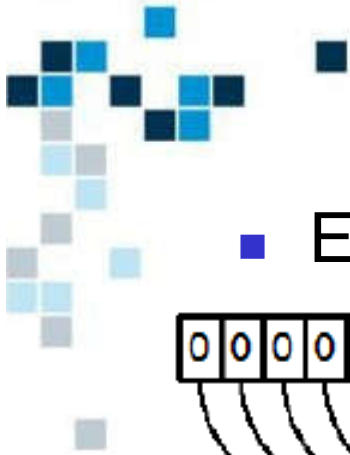


- Si le bit # est à 1, *operand2* est une valeur littérale sur 32 bits.
 - Une instruction est codée sur 32 bits, il n'est donc pas possible de coder n'importe quelle valeur littérale 32 bits.
 - Cette valeur est construite par rotation vers la droite d'une valeur sur 8 bits.

■ Instructions de traitement (5)

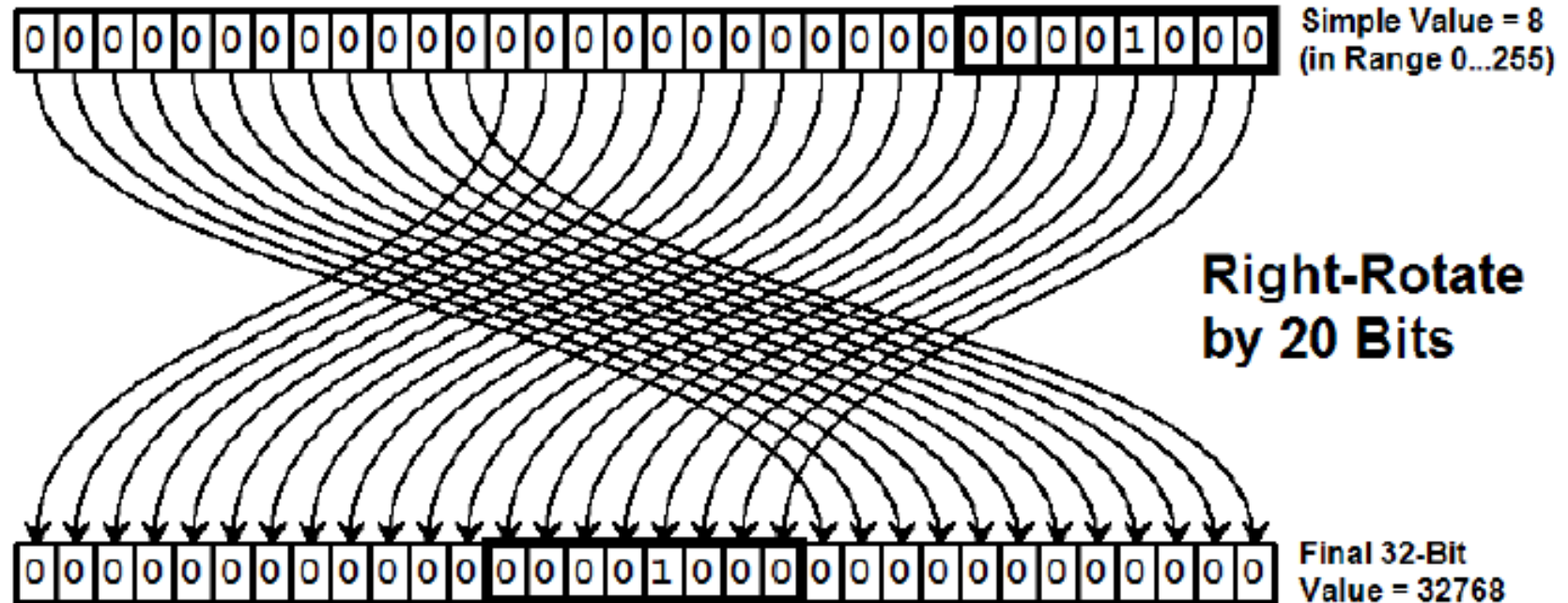


- Cette valeur est construite par rotation vers la droite d'une valeur sur 8 bits.
- Le nombre de rotations (*#rot*) doit être pair: on ne code pas le bit de poids faible.
- Exemple 1: ADD r5, r4, #0x5C0
 - $(5C0)_{\text{hex}} = (0\dots0\ 0\mathbf{101\ 1100\ 0000})_{\text{bin}}$
 - $8\text{-bit immediate} = (17)_{\text{hex}} = (000\mathbf{1\ 0111})_{\text{bin}}$
 - $(\#rot) = (26)_{\text{dec}} = (1101\mathbf{0})_{\text{bin}}$



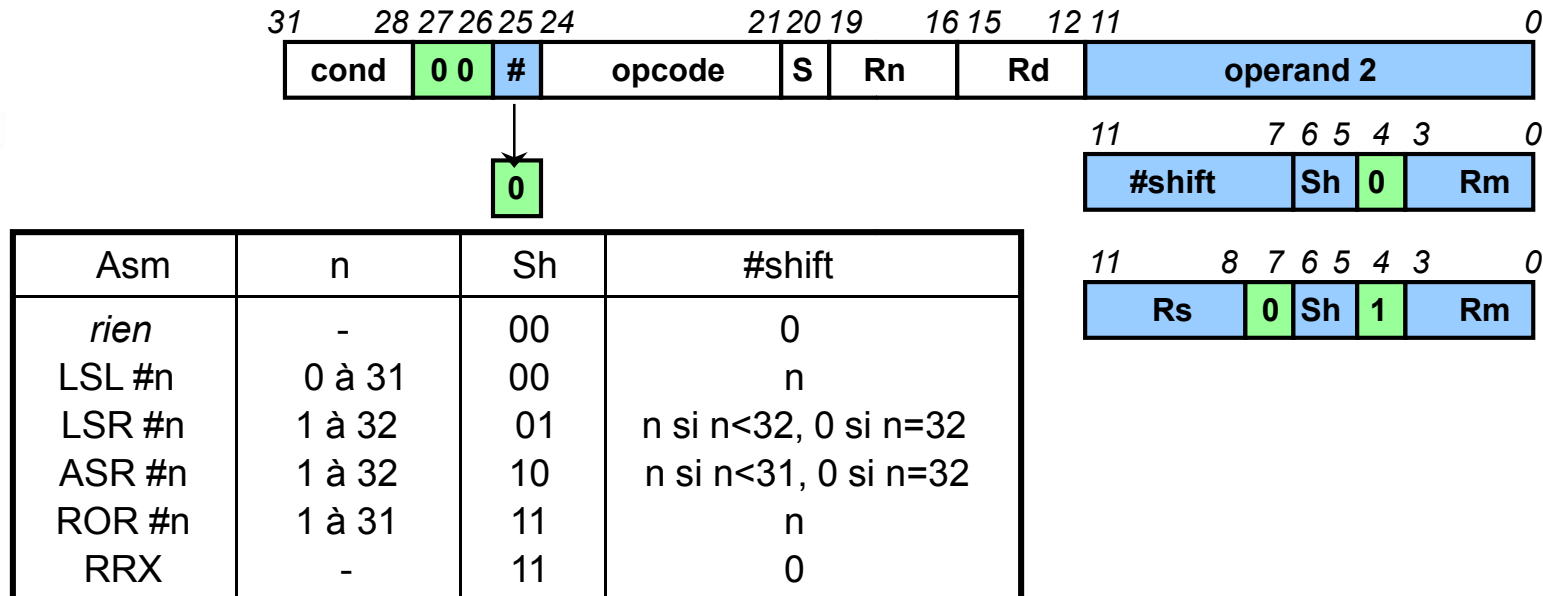
Codage binaire

■ Exemple 2: codage de la valeur 32768

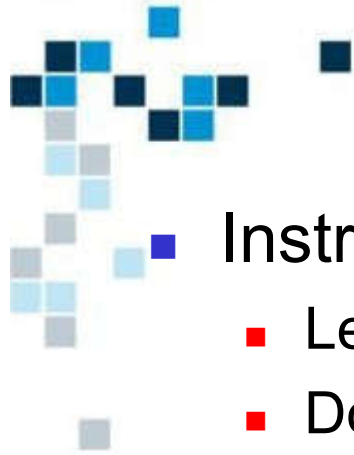


- $(32768)_{\text{dec}} = (0\dots0 \text{ 0000 1000 } 0000 \ 0000 \ 0000)_{\text{bin}}$
- 8-bit immediate = $(8)_{\text{dec}} = (\text{0000 1000})_{\text{bin}}$
- $(\#rot) = (20)_{\text{dec}} = (1010\text{X})_{\text{bin}}$
- $(32768)_{\text{dec}} = (0\dots0 \ 1010 \text{ 0000 1000})_{\text{cod}}$

■ Instructions de traitement (6)



- Si le bit **#** est à 0, *operand2* est un registre (*Rm*) sur lequel on applique (ou non) une rotation/décalage
 - Le nombre de bits de décalage est soit
 - Un littéral **#shift** (5 bits 32 valeurs)
 - Le contenu d'un registre **Rs** (4 bits pour sélection parmi R0...R15)

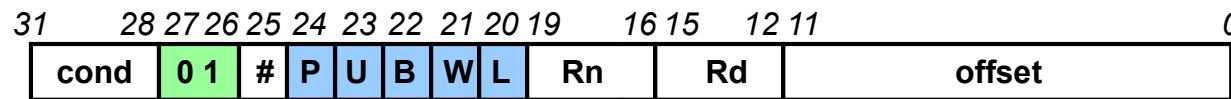


Codage binaire d'une instruction

- Instructions de transfert
 - Lecture/écriture
 - Données accédées
 - Mots, demi-mots, octets
 - Signées/non signées
 - Mode d'accès (pré/post indexé, +/- offset, write back)
 - Transfert simple/multiple
 - Registre source/destination, liste de registres
 - Registre de base
 - Offset
 - Condition
 - Exécution conditionnelle d'un transfert

Codage binaire

- Instructions de transfert de mots ou d'octets non-signés (LDR, STR, LDRB, STRB)

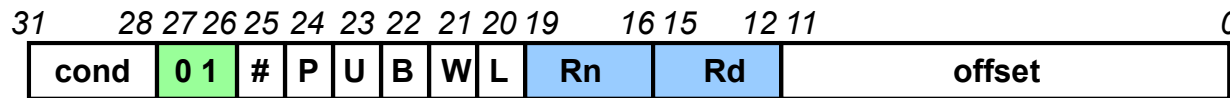


- P : pre/post index
 - 1 pré-indexé, 0 post-indexé
- U : up/down
 - 1 + offset, 0 -offset
- B : byte/word
 - M1 accès 8 bits, 0 accès 32 bits
- W : write-back
 - Si $P=1$, $W=1$ adressage pré-indexé automatique
- L : load/store



Codage binaire

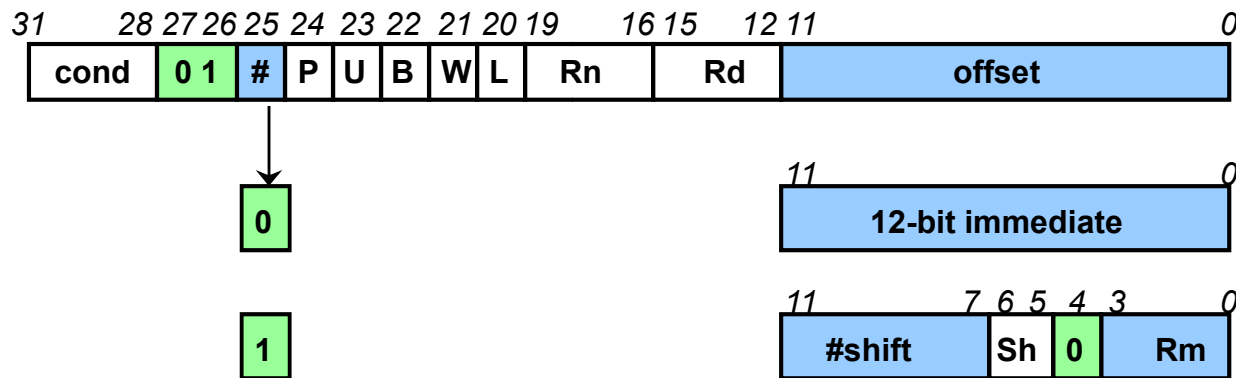
- Instructions de transfert de mots ou d'octets non-signés (LDR, STR, LDRB, STRB)



- $Rd \rightarrow$ registre source (si $L=0$, store) ou destination (si $L=1$, load)
- $Rn \rightarrow$ registre de base

Codage binaire

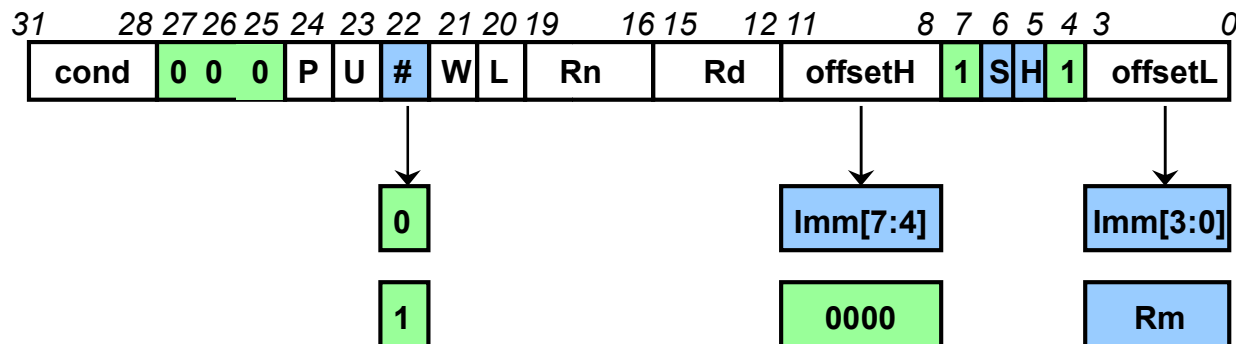
- Instructions de transfert de mots ou d'octets non-signés (LDR, STR, LDRB, STRB)



- Offset: soit un littéral non signé sur 12 bits, soit un registre d'index (Rm) éventuellement décalé sur un nombre constant de bits ($\#shift$)

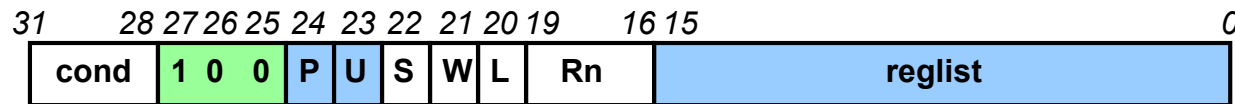
Codage binaire

- Instructions de transfert de demi-mots ou d'octets signés (LDRH, STRH, LDRSH, STRSH, LDRSB, STRSB)



- S : signed
 - 1 nombre signé, 0 non signé
- H : half-word/byte
 - 1 accès 16 bits, 0 accès 8 bits
- Offset : soit un littéral sur 8 bits (*Imm*), soit un registre d'index (*Rm*) non décalé

- Instructions de transfert multiple (LDMmode, STMmode)



mode	U	P
DA	0	0
DB	0	1
IA	1	0
IB	1	1

- Chaque bit de reglist contrôle le transfert d'un registre: si $\text{reglist}[i] = 1$, alors le registre r_i sera transféré



Programme

- Pour nous, un programme est une séquence d'instructions disposées l'une après l'autre dans un fichier texte et repérées (éventuellement) par des labels
- Pour le processeur, un programme est
 - Une succession de mots de 32 bits
 - ... conformes à la convention de codage des instructions pour le processeur considéré
 - ... rangés en mémoire à des adresses contigües
 - Une instruction est repérée par son adresse mémoire



Programme

- Exemple: programme assembleur ARM implanté à l'adresse 0x8000 pour une organisation little-endian

		Adresses croissantes →			
MOV r4, #1	0x8000	01	40	A0	E3
STR r4, [r6], #4	0x8004	04	40	86	E4
STR r4, [r6]	0x8008	00	40	86	E5
STR r4, [r6, #4]	0x800C	04	40	86	E5
MOV r7, #1	0x8010	01	70	A0	E3
MOV r9, #5	0x8014	05	90	A0	E3
MOV r8, r7	0x8018	07	80	A0	E1
LDR r4, [r6]	0x801C	00	40	96	E5
LDR r5, [r6, #4]!	0x8020	04	50	B6	E5
ADD r4, r4, r5	0x8024	05	40	84	E0
		Poids faible		Poids fort	



Programme

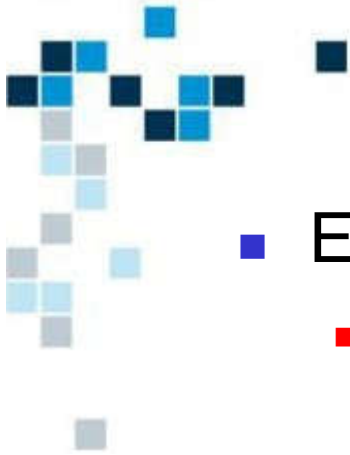
- Le "compteur programme"
 - Registre r15 également appelé PC (Program Counter)
 - Contient l'adresse de la prochaine instruction à lire en mémoire
 - Évolue de 4 en 4 au fil de l'exécution
 - En avance de deux instructions sur l'instruction suivante

MOV r4, #1	0x8000	01	40	A0	E3
STR r4, [r6], #4	0x8004	04	40	86	E4
STR r4, [r6]	0x8008	00	40	86	E5
STR r4, [r6, #4]	0x800C	04	40	86	E5
MOV r7, #1	0x8010	01	70	A0	E3
MOV r9, #5	0x8014	05	90	A0	E3
MOV r8, r7	0x8018	07	80	A0	E1
LDR r4, [r6]	0x801C	00	40	96	E5
LDR r5, [r6, #4]!	0x8020	04	50	B6	E5
ADD r4, r4, r5	0x8024	05	40	84	E0

Instruction en cours d'exécution

PC (r15)
0x8018

Instruction lue en mémoire



Programme

- Exécution séquentielle et rupture de séquence
 - L'incrémentation automatique du registre PC permet d'exécuter les instructions séquentiellement (l'une après l'autre)
 - Nécessité d'instructions de "rupture de séquence"
 - Pour exécuter conditionnellement des blocs d'instruction (traduction des structures *if* et *switch* en langage C)
 - Pour exécuter un bloc d'instructions en boucle
 - Méthode brutale: agir directement sur le registre PC
 - MOV pc, ...
 - ADD/SUB pc, ...

Exécution séquentielle et rupture de séquence

■ Exemple: traduction d'une boucle *for*

0x8000	MOV	r4, #0	@ tmp=0
0x8004	MOV	r5, #0	@ i=0
0x8008	ADD	r4, r4, r5	@ tmp+=i
0x800C	ADD	r5, r5, #1	@ i++
0x8010	CMP	r5, #5	
0x8014	MOVL	T pc, 0x8008	@ i<5 : réitérer

■ Branchement absolu: on connaît l'adresse de l'instruction destination

■ Inconvénients:

- Le programme doit être placé à une adresse fixe connue
- Programme difficile à modifier (insertion d'instructions)

Exécution séquentielle et rupture de séquence

■ Exemple: traduction d'une boucle *for*

```
0x8000      MOV      r4, #0          @ tmp=0
0x8004      MOV      r5, #0          @ i=0
0x8008      ADD      r4, r4, r5      @ tmp+=i
0x800C      ADD      r5, r5, #1      @ i++
0x8010      CMP      r5, #5
0x8014      MOVLT    pc, 0x8008   @ i<5 : réitérer
```

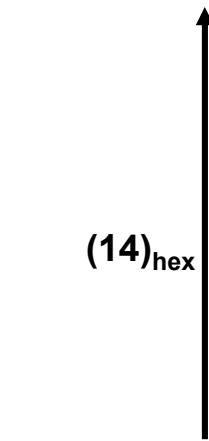

■ Branchement absolu: on connaît l'adresse de l'instruction destination

■ Inconvénients:

- Le programme doit être placé à une adresse fixe connue
- Programme difficile à modifier (insertion d'instructions)
- Limitation sur les valeurs littérales dans l'instruction MOV

Exécution séquentielle et rupture de séquence

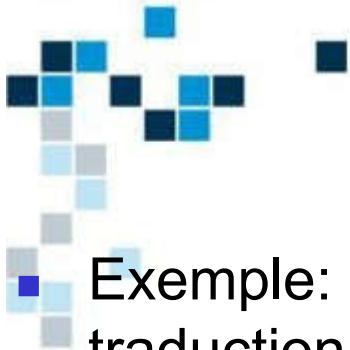
- Exemple: traduction d'une boucle *for*



(14)_{hex} ↑

0x8000	MOV	r4, #0	@ tmp=0
0x8004	MOV	r5, #0	@ i=0
0x8008	ADD	r4, r4, r5	@ tmp+=i
0x800C	ADD	r5, r5, #1	@ i++
0x8010	CMP	r5, #5	
0x8014	SUBLT	pc, pc, #0x14	@ i<5 : réitérer
0x8018	...		
0x801C	...		

- Branchement relatif: on connaît la distance entre la valeur courante de PC et l'instruction destination
- Inconvénient: insertion d'instructions



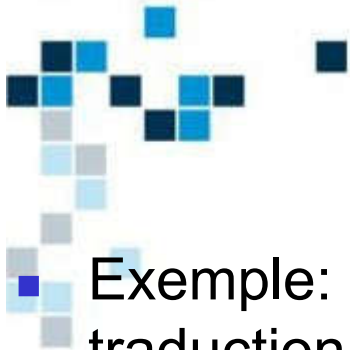
■ Exemple:
traduction
d'un appel
de sous-
programme

Sans
instruction BL:
comment
revenir d'un
appel de sous-
programme?

Cas de l'instruction BL

```
@ Fonction somme: n dans r1, résultat dans r0
0x8000  MOV      r4, #0           @ tmp=0
0x8004  MOV      r5, #0           @ i=0
0x8008  ADD      r4, r4, r5       @ tmp+=i
0x800C  ADD      r5, r5, #1       @ i++
0x8010  CMP      r5, r1
0x8014  SUBLT   pc, pc, #0x14   @ i<n : réitérer
0x8018  MOV      r0, r4           @ résultat = tmp
0x801C  ???      @ return

@ programme principal
0x8020  MOV      r1, #5           @ n = 5
0x8024  MOV     pc, 0x8000       @ appel à somme
0x8028  MOV      r6, r0           @ r6 = résultat
0x802C  MOV      r1, #10          @ n = 10
0x8030  MOV     pc, 0x8000       @ appel à somme
0x8034  MOV      r7, r0           @ r7 = résultat
```



Exemple:
traduction
d'un appel
de sous-
programme

Sans
instruction BL:
il faut
sauvegarder
explicitement
l'adresse de
retour.

Cas de l'instruction BL

```
@ Fonction somme: n dans r1, résultat dans r0
0x8000 MOV      r4, #0          @ tmp=0
0x8004 MOV      r5, #0          @ i=0
0x8008 ADD      r4, r4, r5      @ tmp+=i
0x800C ADD      r5, r5, #1      @ i++
0x8010 CMP      r5, r1
0x8014 SUBLT    pc, pc, #0x14 @ i<n : réitérer
0x8018 MOV      r0, r4          @ résultat = tmp
0x801C MOV      pc, r14        @ return

@ programme principal
0x8020 MOV      r1, #5          @ n = 5
0x8024 MOV      r14, pc
0x8028 MOV      pc, 0x8000      @ appel à somme
0x802C MOV      r6, r0          @ r6 = résultat
0x8030 MOV      r1, #10        @ n = 10
0x8034 MOV      r14, pc
0x8038 MOV      pc, 0x8000      @ appel à somme
0x803C MOV      r7, r0          @ r6 = résultat
```

Exécution séquentielle et rupture de séquence

■ Conclusion

- Agir directement sur le PC est complexe et source d'erreur.
- Rend un programme assembleur difficile à modifier.
- Rend difficile l'appel à des sous programmes.
- On utilise des instructions de branchement, associées à des étiquettes.
- Les étiquettes sont des adresses correspondant à une instruction, ou à une donnée.
- Exemple:
 - VALEUR: .word 123
 - LOOP: MOV R1, R2
- Un branchement vers une étiquette est codé dans l'instruction par un saut relatif (ou offset, ou déplacement signé).
- C'est l'étape d'assemblage qui calcule l'offset.



Instructions de branchement relatif

- Branch (B)
- Branch and Link (BL)



- Offset : déplacement signé sur 24 bits
- L : Link (0 branch, 1 branch and link)
- Effets:
 - B: $PC \leftarrow PC + \text{offset}$
 - BL: $r14 \leftarrow PC - 4$; $PC = PC + \text{offset}$
 - r14 : Link Register (contient l'adresse de retour)

Instructions de branchement relatif

- Exemple traduction d'une boucle *for*
- Utilisation de labels en langage d'assemblage: le déplacement est calculé automatiquement par l'assembleur

```
MOV      r4, #0           @ tmp=0
MOV      r5, #0           @ i=0
loop:    ADD      r4, r4, r5    @ tmp+=i
         ADD      r5, r5, #1    @ i++
         CMP      r5, #5
         BLT      loop         @ i<5 : réitérer
         ...
```



Représentation binaire: **BAFFFFFFB**

Instructions de branchement relatif

- Exemple traduction d'une boucle *for*
- Utilisation d'adresses en langage d'assemblage: le déplacement est calculé automatiquement par l'assembleur

(Loop:)

0x8000	MOV	r4, #0	@ tmp=0
0x8004	MOV	r5, #0	@ i=0
0x8008	ADD	r4, r4, r5	@ tmp+=i
0x800C	ADD	r5, r5, #1	@ i++
0x8010	CMP	r5, #5	
0x8014	BLT	0x8008 (Loop)	@ i<5 : réitérer
0x8018	...		



Instructions de branchement relatif

■ Exemple:
traduction
d'un appel
de sous-
programme

■ L'instruction
BL simplifie
grandement
l'écriture du
programme
assembleur.

```
@ Fonction somme: n dans r1, résultat dans r0

somme:      MOV      r4, #0           @ tmp=0
            MOV      r5, #0           @ i=0
loop        ADD      r4, r4, r5       @ tmp+=i
            ADD      r5, r5, #1       @ i++
            CMP      r5, r1
            BLT      loop            @ i<5 : réitérer
            MOV      r0, r4           @ résultat = tmp
            MOV      pc, lr          @ return

@ programme principal
main:       MOV      r1, #5           @ n = 5
            BL       somme            @ appel à somme
            MOV      r6, r0           @ s1 = résultat
            MOV      r1, #10          @ n = 10
            BL       somme            @ appel à somme
            MOV      r7, r0           @ s2 = résultat
```