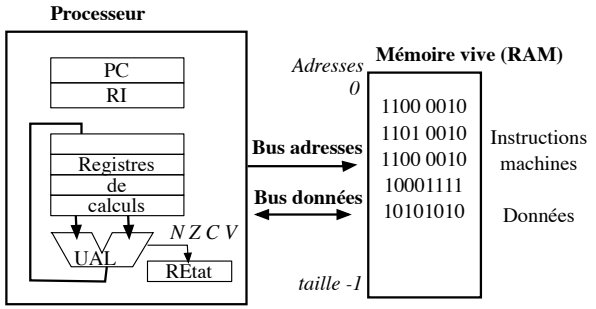
Traduction des structures algorithmiques en langage d'assemblage

- Introduction
- Comment traduire un programme en langage de haut niveau possédant des structures de contrôle : "si alors sinon", "tant que faire" ... en langage d'assemblage ?
- Travail réalisé par le compilateur
- Exemple en C:
  - if (a>b) {a=a+b;} else {a=a+1;}

Rappel

- Le processeur exécute des Instructions machines de calcul (ADD, SUB…) et de chargement/stockage en mémoire (Ld, Str)
- Les instructions sont exécutées dans l'ordre d'écriture et de stockage en mémoire lors de l'exécution.



Exemple

```
MOV r0, #3
LD r1, [r0]
MOV r2, #3
ADD r1, r2, #12
STR r0, [r4]
....
```

- On sait traduire des affectations et des calculs
- Exécution des instructions toujours en séquence
- Comment traduire par exemple :
  - *si r1 > r2 alors r1 = 3 sinon r1 = 4*
- Calcul de la condition r1 > r2 ?
- Puis enchainement particulier des instructions du ***alors*** et du ***sinon ?***

Calcul de la condition

- Comment calculer la condition de la structure conditionnelle ?
- Exécuter un calcul; en général une soustraction
- Evaluer le résultat de la condition à l'aide des valeurs des flags du processeur : N, Z, V et C
- Exemple en ARM
  - **SUBS r1, r2, r3** @r1=r2-r3 et mise à jour des flags
  - Le résultat de la soustraction peut ne pas avoir d'importance
  - Soustraction sans stockage du résultat mais avec mise à jour des flags : **CMP**
- Il faut ensuite inventer une nouvelle instruction qui soit capable de changer la ***séquentialité*** des instructions en fonction de la valeur des flags

L'instruction de branchement

- Appelé aussi "**rupture de séquence**"
- L'instruction qui suit un branchement n'est pas forcément celle qui suit en mémoire. Suivant la valeur des flags, on peut « se brancher » à une instruction ailleurs en mémoire
- Il faut donc spécifier cette adresse de branchement dans l'instruction

L'adresse de ce branchement est :

- **soit donnée dans l'instruction (processeur CISC)**
- **soit c'est un déplacement qui est donné dans l'instruction. Le branchement consiste à ajouter ce déplacement à PC**

L'instruction de branchement

- Les conditions possibles dépendent des 4 flags Z, N,C et V
- La valeur de la condition dépend de la valeur de ces flags et donc de **la dernière instruction de calcul effectuée** par le processeur et modifiant ces flags (en général un CMP)
- Exemple : la condition "égale" est traduite par Z=1 après un CMP
  - En effet op1 = op2 si op1-op 2= 0 (Z=1)

Instruction de branchement

- Exemple en ARM:
- On regarde la mini documentation ARM, On trouve page 5 :

B{cond} deplacement

- Le déplacement est codé en complément à 2 sur 24 bits. On peut se brancher avant ou après. Le calcul effectué par le branchement est PC= PC + 4* depl
- Amplitude de branchement -2^{23} à $2^{23}-1$ instructions
- On ne va pas calculer les déplacements, c'est l'assembleur qui le fera pour nous , on utilise des étiquettes

Les codes conditions

Voir dans la documentation ARM figure 2 page 3

- On retrouve des conditions arithmétiques et leurs traductions sur les flags
- Exemple :
  - **etiq : cmp r0, #22**
    bhi etiq
- Le condition **hi** : > en non signé, donc le branchement à **etiq** aura lieu si r0 > 22 en considérant la valeur de r0 en base 2
- Si l'on veut considérer la valeur de r0 en complément à 2 on utilise **la condition gt**

Les codes conditions du processeur ARM

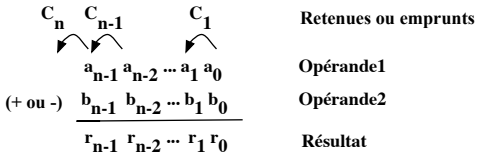
•Cas particulier: **AL : toujours**

•**Conditions**		**Expression en fonction des flags**
Non signé (base 2)		
•CS/HS	(≥)	C
•CC/LO	(<)	$\overline{C}$
•HI	(>)	C.$\overline{Z}$
•LS	(≤)	$\overline{C}$+Z
Signé (complément à 2)		
•GE	(≥)	N.V+$\overline{N}$.$\overline{V}$
•LT	(<)	N.$\overline{V}$+$\overline{N}$.V
•GT	(>)	$\overline{Z}$.(N.V+$\overline{N}$.$\overline{V}$)
•LE	(≤)	Z+(N.$\overline{V}$+$\overline{N}$.V)
Signé et non signé		
•EQ	(=)	$\overline{Z}$
•NEQ	(≠)	Z

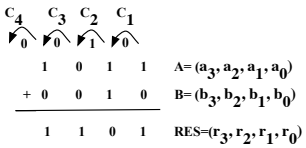
Rappel sur les flags

• Mise à jour en fonction du dernier calcul effectué dans le processeur (par une instruction modifiant ces flags dans le ARM: CMP, ADDS, SUBS,...)

•Soit le calcul sur deux opérandes suivants:



• Exemple



Flag Z

• **Z**: =1 si le résultat est nul; 0 sinon

-Z= $\overline{r_{n-1}}$.$\overline{r_{n-2}}$ ... .$\overline{r_1}$.$\overline{r_0}$

-Après une soustraction OP1-OP2, Z=1 si OP1 = OP2

-Valable pour les deux conventions de stockage possibles:

- Base 2 (entiers naturels : «non signés»)
- Complément à 2 (entiers relatifs : «signés»)

Le flag C

-Significatif seulement si la convention de codage des opérandes est la base 2

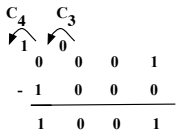
-En cas d'une addition : C= dernière retenue sortante (C_n)

-Cas particulier du processeur ARM: C= complément du dernier emprunt de la soustraction (Complément de C_n)

-Donc convention inverse: après une soustraction OP1-OP2 en supposant les entiers en base 2 (non signé) C=0 indique que OP1 < OP2

-Permet aussi d'indiquer que le résultat est juste sur n bits (après une addition ou une soustraction)

-Exemple: 0001-1000 = 1001 ; le dernier emprunt est égal à 1 donc C=0 , on a bien 1 < 8, mais le résultat est faux (en base 2)



Flag N

- N = bit de poids fort du résultat de l'opération
- Significatif seulement en supposant les entiers codés en complément à 2
- N =1 si le résultat est négatif
- Donc après OP1-OP2 , N=1 si OP1<OP2
- Exemple : 0001 - 0010 = 1111 N=1 on a bien 1<2
- En fait valable seulement si le résultat est juste
  - Exemple : 1000 - 0001 = 0111 ; N= 0 alors que (-8) < 1
- Ne suffit pas à calculer des conditions sur les opérandes lorsque le résultat est faux (voir flag V)

Flag V

- oVerflow; V=1 si le résultat est faux sur n bits (valable pour une addition ou une soustraction)
- Significatif seulement en supposant les entiers codés en Complément à 2
- Calcul à l'aide des deux dernières retenues de l'addition (ou emprunts pour la soustraction)

$$V = C_n \oplus C_{n-1}$$

- Calcul à l'aide des bits de poids fort des opérandes

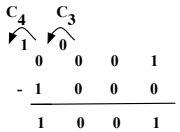
$$V = \overline{a_{n-1}}.\overline{b_{n-1}}.C_{n-1} + a_{n-1}.b_{n-1}.\overline{C_{n-1}}$$

A titre d'exercice retrouver cette formule en partant de la formule de C_n en fonction de a_{n-1}, b_{n-1} et C_{n-1}

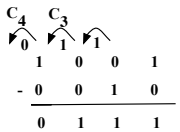
Flag V

Exemples:

- 1- (-8) = -7 ; V=1 donc résultat faux
- Condition ≥ : $N.V + \overline{N}.\overline{V}$ on a bien 1≥-8



- (-7) - 2 = 7 ; V=1 donc résultat faux
- Condition < : $N.\overline{V} + \overline{N}.V$ on a bien -7<2



Exercices

- Pour les opérations suivantes
  - Calculer les quatres flags Z, N, C, V
  - Retrouver quelles conditions sont justes après ces calculs (parmi les conditions des instructions de branchement du ARM)
    - 0111 - 0111
    - 0111 - 1000
    - 0111 + 1000
    - 1111 - 0001
    - 1000 - 0001
    - 1000 - 1111

Traduction des “si alors sinon”

• On oublie le chargement des variables en mémoire dans des registres…

Exemple : si r1 > r2 alors r1 = 3 sinon r1 = 4

• Pour savoir quelle instruction de branchement utiliser, il faut savoir quel codage est utilisé : base 2 ou Complément à 2

• Ensuite on peut lire dans le tableau de la documentation:

-En base 2 : > HI

-En complément à 2 : > GT

Traduction en Goto

• On peut traduire le “si alors sinon” sous forme de goto

• Exemple:

• Si r1>r2 goto Alors

goto sinon

Alors : instructions_du_alors

goto finsi

Sinon : instructions_du_sinon

Finsi :

• On obtient la “forme” du programme assembleur

Traduction du si alors sinon Version 1

- si r1 > r2 alors r1 = 3 sinon r1 = 5
- Hypothèse: r1 et r2 **supposé en Complément à 2** :

```
        cmp r1 , r2
        bgt etiq_alors      @ branchement si r1 > r2
        bal etiq_sinon
etiq_alors : mov r1, #3          @ c'est le alors
        bal etiq_fin_si     @branchement toujours à la fin
etiq_sinon: mov r1, #5      @ c'est le sinon
etiq_fin_si :
```


Traduction du si alors sinon Version 2

```
    Si r1>r2 goto Alors
        Instruction_du_sinon
        goto finsi
Alors :  instructions_du_alors
Finsi :
```

Assembleur ARM :

```
        cmp r1 , r2
        bgt etiq_alors      @ branchement si r1 > r2
        mov r1, #5          @ c'est le sinon
        bal etiq_fin_si     @ branchement toujours
etiq_alors : mov r1, #3     @ c'est le alors
etiq_fin_si :
```

Traduction du si alors sinon Version 3

On peut tester la condition contraire et inverser les deux blocs « alors » et « sinon » :

```
            cmp r1 , r2
            ble etiq_sinon        @ branchement si r1 ≤ r2
            mov r1,#3             @ c'est le alors
            bal etiq_fin_si
etiq_sinon : Mov r1, #5           @ c'est le sinon
etiq_fin_si :
```


Une particularité du processeur ARM

- Instructions conditionnelles
- Permettent d'optimiser le code : gain d'instructions
- Champ cond des instructions arithmétiques : même signification que pour les branchements
- L'instruction est effectuée seulement si la condition est vrai

Exemple :

Addeq r1, r2, r3 ne sera exécutée que si le flag z=1 (condition eq)

La modification du flag a été faite par une instruction précédente

Utilisation des instructions conditionnelles

- On peut traduire le si alors sinon de façon plus simple :

```
cmp r1,r2
movgt r1,#3    @ si r1 > r2
movle r1,#5    @ si r1 ≤ r2
```

Intéressant si le "alors" et le "sinon" ne sont pas trop « compliqués »

Traduction des "tant que faire"

- Même idée que pour le «***si alors***»
- On calcule la condition et on se branche ou non au corps de la boucle
- **Exemple :** while (r1 ≥ r2) r2= r2+1; (Supposé en complément à 2)

Solution1 :

```
tantque :   cmp r1,r2
            bge boucle          @ branchement si r1 ≥ r2 au corps du tant que
            bal fin_tantque     @ retour toujours test de la boucle
boucle :    add r2, r2, #1      @ corps de la boucle
            bal tantque         @ retour toujours test de la boucle
fin_tantque :
```

Traduction des “tant que faire” solution 2

```
tantque :   cmp r1,r2
            blt fin_tantque     @ branchement si r1 < r2 : fin de la boucle
            add r2, r2, #1      @ corps de la boucle
            bal tantque         @ retour toujours test de la boucle
fin_tantque :
```

On gagne une instruction, mais même nombre d’instructions dans chaque tour de boucle

Traduction des “tant que faire” solution 3

```
            bal test_boucle
tant_que :  add r2, r2,#1   @ corps de la boucle
test_boucle: cmp r1, r2
            bge tant_que    @ branchement si r1 ≥ r2 : corps du tantque
fin_tantque :
```

Même nombre d’instruction que la solution 2 mais on gagne l’exécution d’une instruction à chaque tour de boucle (un branchement)

Exercices

- Utiliser les instructions de calcul conditionnelles sur cette exemple
- while (r1 ≥ r2) r2= r2+1
- Et celui là:
- while (r1 == r2) {r2= r2+1; r1=r1+r2}

- **Remarque :** la traduction du ***sialors*** et ***tantque*** peut s’automatiser facilement suivant un des schémas donnés précédemment

Traduction de conditions complexes

Comment traduire des conditions plus compliquées comme:

(r1>r2) ou ((r1≤r3) et (r4=r5))

En C: (r1>r2) || ((r1≤r3) && (r4==r5))

Traduction du OU booléen

• Exemple sur un sialorssinon :

Si (r1>r2) ou (r2<r3) alors A1 sinon A2

(En Complément à 2)

• Pour un OU, on remarque que si la première condition est vrai ce n'est pas la peine d'évaluer la seconde (c'est le **oualors** algorithmique)

Traduction du OU booléen

• Exemple sur un sialorssinon :

Si (r1>r2) ou (r2<r3) alors A1 sinon A2

```
        cmp r1, r2
        bgt alors    @branchement à alors si cond1 est vrai
        cmp r2, r3
        blt alors    @branchement à alors si cond2 est vrai
sinon : A2           @instructions du sinon
        Bal fin      @branchement à la fin du sialorssinon
alors :  A1          @instructions du alors
fin :
```

• Généralisable à n conditions

Traduction du ET booléen

• Exemple sur un sialorssinon :

Si (r1>r2) et (r2<r3) alors A1 sinon A2

```
        cmp r1, r2
        ble sinon    @branchement à sinon si cond1 est fausse
        cmp r2, r3
        bge sinon    @branchement à sinon si cond2 est fausse
alors :  A1          @instructions du alors
        Bal fin      @branchement à la fin du sialorssinon
sinon : A2           @instructions du sinon
fin :
```

• Généralisable à n conditions

Traduction du ET booléen

• On peut aussi utiliser De Morgan et se ramener à un OU

• C1 et C2 = not (not C1 ou not C2)

Si (r1>r2) et (r2<r3) alors A1 sinon A2

• Equivalent à si (r1≤r2) ou (r2≥r3) alors A2 sinon A1

On retrouve la solution précédente:

```
        cmp r1, r2
        ble alors
        cmp r2,r3
        bge alors
sinon : A1
        Bal fin
alors :  A2
fin :
```

Utilisation des instructions conditionnelles

• Si les conditions sont toutes du même type (égal, supérieur…), on peut « gagner » les branchement au **alors** intermédiaires

• Exemple :

Si (r1=2 ou r3=4 ou r5 = r6) alors A1 sinon A2

```
        Cmp r1,#2
        Cmpne r3 , #4  @comparaison effectuée si la 1ere cond est fausse
        Cmpne r5,r6    @comparaison effectuée si la 1er et la 2eme cond sont fausses
        Beq alors      @branchement au alors si cond1 ou (not cond1 et cond2) ou (not cond1 et not cond2 et cond3)
```

Sinon : A2

```
        Bal finsi
```

Alors : A1

Finsi :

Exercices

- Traduire un ***switch*** en assembleur ARM:
- Exemple:

```
int a;
switch (a)
        {
        case 0: A0; break;
        case 1:  A1;break;
        default: A3; break
        }
```

- Traduire un ***for*** en assembleur ARM:
- Exemple:

```
int a ;
for (a=0; a <= 30; a++) {A;}
```