LU2IN014 : Machine et Représentation

Cours 4 : Mémoire, Variables Globales, Tableaux et Structures de Contrôle

quentin.meunier@lip6.fr

1. Architecture de la mémoire
2. Adresse des variables globales et instructions de transfert mémoire
3. Implantation des tableaux en assembleur
4. Sauts inconditionnels et conditionnels
5. Réalisation des structures de contrôle des langages de haut niveau à l'aide de sauts

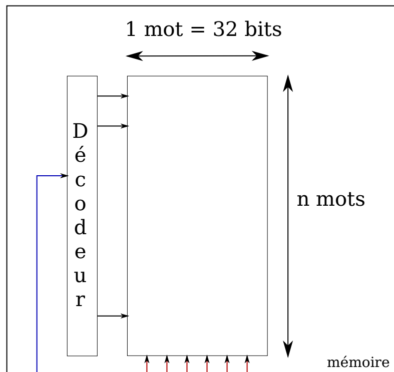
1. **Architecture de la mémoire**
2. Adresse des variables globales et instructions de transfert mémoire
3. Implantation des tableaux en assembleur
4. Sauts inconditionnels et conditionnels
5. Réalisation des structures de contrôle des langages de haut niveau à l'aide de sauts

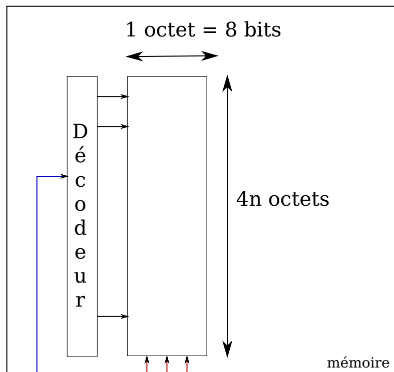
Architecture de la mémoire

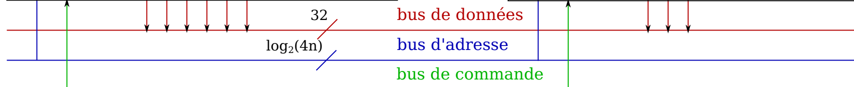
Généralités

- La mémoire peut être vue comme :
  - une suite de n **mots** de 32 bits
  - un tableau de $4n$ octets de 8 bits
- La position d'un octet dans la mémoire définit son **adresse** : adresse du 1er octet = 0, adresse du 2ème octet = 1, ..., adresse du dernier = 4n - 1
- On dit que l'**unité adressable** est l'octet : on peut écrire ou lire au minimum un octet
- En Mips, on peut écrire ou lire au plus 1 mot (4 octets) à la fois ; dans ce cas, l'adresse du mot est l'adresse la plus petite des 4 octets du mot, et cette adresse doit être un multiple de 4

Architecture de la mémoire







Capacité mémoire

Définition

- La **capacité mémoire** correspond au nombre d'octets qu'elle peut stocker, l'unité de la taille d'une mémoire est donc un **nombre d'octets**

Capacité de stockage

- 1 kilo-octet ou 1 Kio $= 2^{10} = 1\ 024$ octets
- 1 mega-octet ou 1 Mio $= 2^{20} = 1\ 048\ 576$ octets
- 1 giga-octet ou 1 Gio $= 2^{30}$ octets
- 1 tera-octet ou 1 Tio $= 2^{40}$ octets

- Lire la page wikipédia "**Préfixes binaires**"

Rangement de mots de plusieurs octets

Rangement de mots de plusieurs octets

- Une donnée de plusieurs octets est rangée à des adresses contigües

Little/Big Endian

- Soit $M = 0xo_3o_2o_1o_0$ un mot de 4 octets rangé à l'adresse A
- Il y a 2 rangements possibles :
  - Big Endian (grand boutien) : l'octet de poids fort est rangée à l'adresse la plus petite
  - Little Endian (petit boutien) : l'octet de poids faible est rangé à l'adresse la plus petite ⇒ **rangement utilisé en Mips**

Adresse	Little Endian	Big Endian
A	o_0	o_3
A + 1	o_1	o_2
A + 2	o_2	o_1
A + 3	o_3	o_0

Transfert entre la mémoire et le processeur

- Le processeur initie les transferts de données entre le processeur et la mémoire et indique/donne à la mémoire :
  - L'adresse du mot à transférer
  - La taille du mot à transférer
  - Le sens du transfert :
    - Si processeur → mémoire alors c'est une **écriture** ou un store,
    - Si mémoire → processeur alors c'est une **lecture** ou un load (chargement)
  - La donnée à écrire si écriture
- Effet d'une écriture : jusqu'à la prochaine écriture à la même adresse (sinon jusqu'à la fin du programme)
- Une lecture n'est pas destructrice

Exemple d'écriture d'un mot

Effet de l'écriture du mot 0xAABBCCDD à l'adresse 0x4

- Vue par octet :

Adresse	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	...
Contenu	0x??	0x??	0x??	0x??	0xDD	0xCC	0xBB	0xAA	0x??	0x??	0x??	0x??	...

- Vue par mots :

Adresse	0x0	0x4	0x8	...
Contenu	0x????????	0xAABBCCDD	0x????????	...

Exemple d'écriture d'un demi-mot

Effet de l'écriture du demi-mot 0x1234 à l'adresse 2

- Vue par octet :

Adresse	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	...
Contenu	0x??	0x??	0x34	0x12	0xDD	0xCC	0xBB	0xAA	0x??	0x??	0x??	0x??	...

- Vue par mots :

Adresse	0x0	0x4	0x8	...
Contenu	0x1234????	0xAABBCCDD	0x????????	...

Exemple d'écriture d'un octet

Effet de l'écriture de l'octet 0xFF à l'adresse 1

- Vue par octet :

Adresse	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	...
Contenu	0x??	0xFF	0x34	0x12	0xDD	0xCC	0xBB	0xAA	0x??	0x??	0x??	0x??	...

- Vue par mots :

Adresse	0x0	0x4	0x8	...
Contenu	0x1234FF??	0xAABBCCDD	0x????????	...

1 Architecture de la mémoire

2 Adresse des variables globales et instructions de transfert mémoire
- Adresses des variables globales
- Instructions de transfert mémoire
- Illustration des accès mémoire
  - Exécution complète d'instructions par le processeur
  - Exemple de programme assembleur

3 Implantation des tableaux en assembleur

4 Sauts inconditionnels et conditionnels

5 Réalisation des structures de contrôle des langages de haut niveau à l'aide de sauts

Adresses des variables globales

Comment calculer l'adresse d'une variable globale ?

- **Hypothèse** : la section .data est implantée au début du segment data, à l'adresse `0x10010000`
- L'adresse de la première variable `v1` est l'adresse `0x10010000`
- L'adresse de la deuxième variable `v2` est `@v1 + taille(v1) + alignement`
- L'adresse de la n-ième variable `vn` est `@vn-1 + taille(vn-1) + alignement`
- Contrainte d'alignement : l'adresse d'une variable doit être un multiple de sa taille
  - Par exemple, l'adresse d'un mot doit être un multiple de 4, donc se terminer par 0x0, 0x4, 0x8 ou 0xc
- `alignement` est la plus petite valeur (positive ou nulle) permettant de respecter la contrainte d'alignement

Adresses des variables globales

Exemple

```
.data
n : .word 0x20 # @ =
c0 : .byte 0x31 # @ =
tab : .word 0x0, 0x1 # @ =
str : .asciiz "Entrez un nombre :" # @ =
p : .word 0x10 # @ =
c1 : .byte 0xFF # @ =
c2 : .byte 0xFE # @ =
```

Instructions de transfert mémoire

Lecture et écriture mémoire

- Ces instructions lisent (load) ou écrivent (store) des données en mémoire
- Elles utilisent l'ALU pour le calcul d'adresse
- Elle utilisent les bus d'adresse, de données et de commande pour réaliser le transfert mémoire
- L'adresse accédée est toujours égale au contenu d'un registre + un immédiat
- Taille du mot transféré indiquée dans l'opération : `lw`, `lh`, `lb`, `sw`, `sh`, `sb`
- 2 opérandes sources pour l'adresse mémoire, format OpImm(OpReg)
- 1 opérande registre contenant la valeur à écrire ou lue en mémoire : OpReg
- Syntaxe des instructions d'accès mémoire : **Codop OpReg, Imm(OpReg)**

Instructions de transfert mémoire

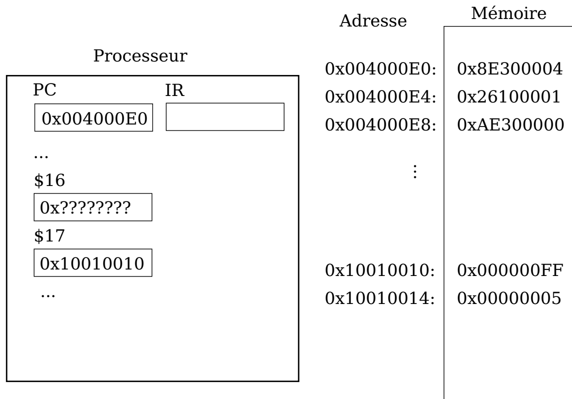
Lecture

- `lw $9, 0($8)` : lire un mot à l'adresse `0 + $8` et le ranger dans $9
- `lh $9, 2($8)` : lire un demi-mot à l'adresse `2 + $8` et le ranger dans $9
- `lb $11, 12($10)` : lire un octet à l'adresse `12 + $10` et le ranger dans $11

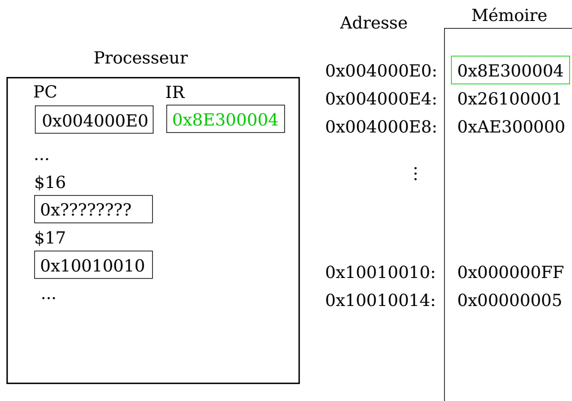
Ecriture

- `sw $16, 0($8)` : écrire le mot contenu dans $16 à l'adresse `0 + $8`
- `sh $17, 4($9)` : écrire le demi-mot contenu dans $17 à l'adresse `4 + $9`
- `sb $18, 13($8)` : écrire l'octet de poids faible de $18 à l'adresse `13 + $8`

Exemple d'exécution d'instructions par le processeur

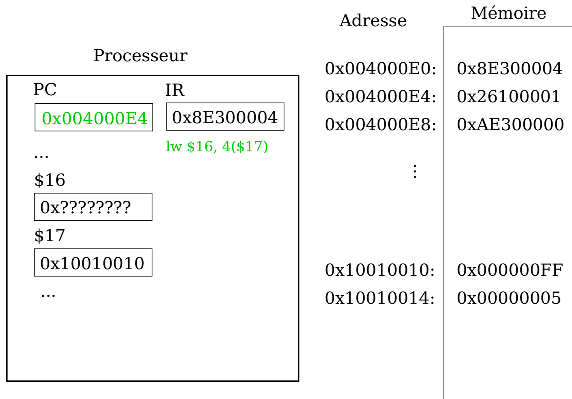


Exemple d'exécution d'instructions par le processeur



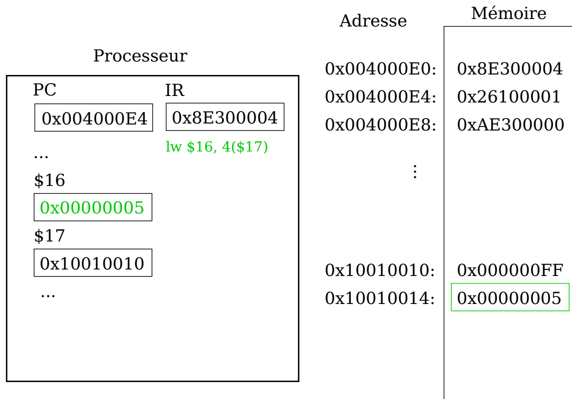
1. Lecture à l'adresse contenue dans le registre PC (0x004000E0)
+ Copie de la valeur lue (0x8E300004) dans le registre IR

Exemple d'exécution d'instructions par le processeur



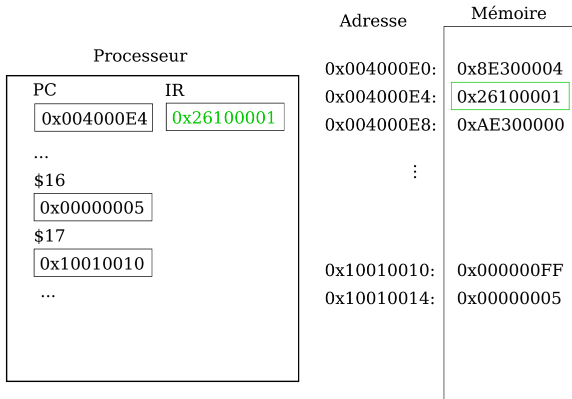
1. Lecture à l'adresse contenue dans le registre PC (0x004000E0)
+ Copie de la valeur lue (0x8E300004) dans le registre IR
2. Décodage de IR : lw $16, 4($17) + Incrémentation de PC

Exemple d'exécution d'instructions par le processeur



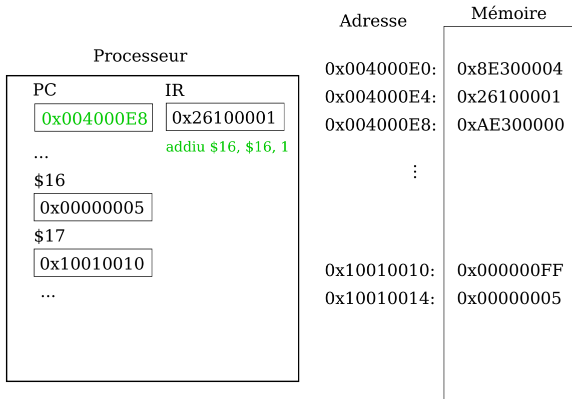
1. Lecture à l'adresse contenue dans le registre PC (0x004000E0)
+ Copie de la valeur lue (0x8E300004) dans le registre IR
2. Décodage de IR : lw $16, 4($17) + Incrémentation de PC
3. Exécution du lw : lecture à l'adresse 0x10010014, écriture du résultat (0x5) dans $16

Exemple d'exécution d'instructions par le processeur



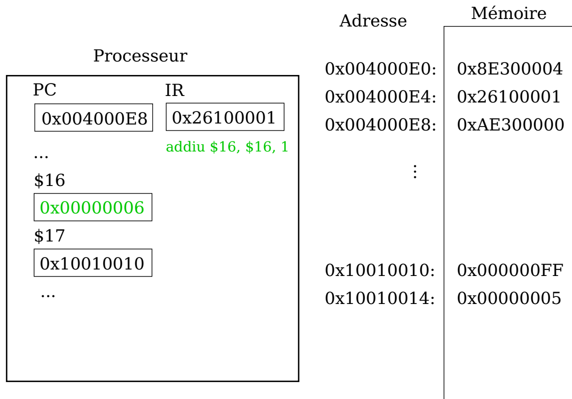
1'. Lecture à l'adresse contenue dans le registre PC (0x004000E4)
+ Copie de la valeur lue dans le registre IR (0x26100001)

Exemple d'exécution d'instructions par le processeur



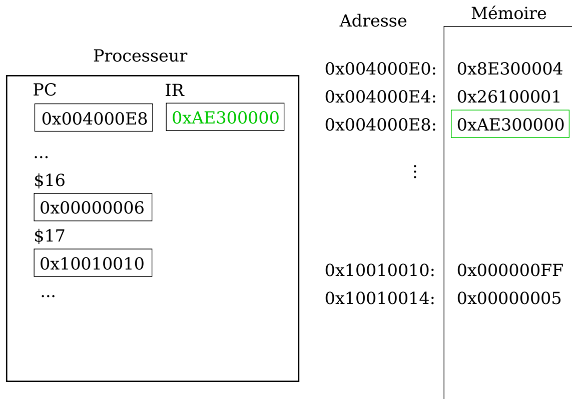
1'. Lecture à l'adresse contenue dans le registre PC (0x004000E4)
+ Copie de la valeur lue dans le registre IR (0x26100001)
2'. Décodage de IR : addiu $16, $16, 1 + Incrémentation de PC

Exemple d'exécution d'instructions par le processeur



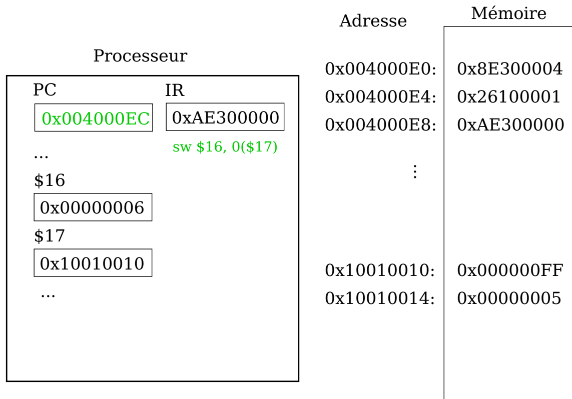
1'. Lecture à l'adresse contenue dans le registre PC (0x004000E4)
+ Copie de la valeur lue dans le registre IR (0x26100001)
2'. Décodage de IR : addiu $16, $16, 1 + Incrémentation de PC
3'. Exécution de l'instruction

Exemple d'exécution d'instructions par le processeur



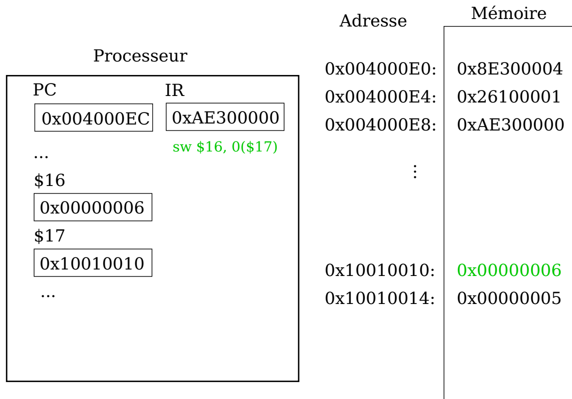
1". Lecture à l'adresse contenue dans le registre PC (0x004000E8)
+ Copie de la valeur lue dans le registre IR (0xAE300000)

Exemple d'exécution d'instructions par le processeur



1". Lecture à l'adresse contenue dans le registre PC (0x004000E8)
+ Copie de la valeur lue dans le registre IR (0xAE300000)
2". Décodage de IR : sw $16, 0($17) + Incrémentation de PC

Exemple d'exécution d'instructions par le processeur



1". Lecture à l'adresse contenue dans le registre PC (0x004000E8)
+ Copie de la valeur lue dans le registre IR (0xAE300000)
2". Décodage de IR : sw $16, 0($17) + Incrémentation de PC
3". Exécution de l'instruction

Exemple assembleur

Programme C	ASM	Binaire
	.data	
	a: .word 5	
int a = 5;	b: .word 9	
int b = 9;	c: .space 4	
int c;	.text	
	lui $8, 0x1001	0x3c081001
void main() {	lw $9, 0($8)	0x8d090000
c = a + b;	lw $10, 4($8)	0x8d0a0004
exit();	addu $9, $9, $10	0x012a4821
}	sw $9, 8($8)	0xad090008
	ori $2, $0, 10	0x3402000a
	syscall	0x0000000c

Remarque sur le code assembleur

- On retrouve les 3 phases : 1) Lecture mémoire → registre 2) Opération registre → registre 3) Écriture registre → mémoire

1 Architecture de la mémoire

2 Adresse des variables globales et instructions de transfert mémoire

3 Implantation des tableaux en assembleur
- Définition d'un tableau à une dimension
- Exemples

4 Sauts inconditionnels et conditionnels

5 Réalisation des structures de contrôle des langages de haut niveau à l'aide de sauts

Structure de données Tableau

Tableau

- Ensemble d'éléments du même type, de nombre fixé
- Ensemble ordonné : un indice donne la position d'un élément dans le tableau

Accès à un élément

- `T[i]` désigne l'élément `i` du tableau `T`

Exemple en C

- `int tab[5] = {1, 2, 3, 4, 5};`
- `char une_chaine[3] = {'a', 'b', 'c'};`
- `char autre_chaine[] = "efgh";`
- Opération d'indexation : `T[2]` désigne l'élément d'indice 2

Implantation d'un tableau à une dimension

Implantation mémoire

- Un tableau est implanté de manière contigüe en mémoire : éléments rangés les uns à la suite des autres, de manière ordonnée

Définition de la zone mémoire associée à un tableau

- `adresse_debut` : l'adresse de base du tableau, i.e. l'adresse du premier élément
- `taille_elem` : la taille (en octets) d'un élément
- `nb_elem` : le nombre d'éléments du tableau

Implantation d'un tableau à une dimension

Accès à un élément

- On peut calculer l'adresse de l'élement `T[i]` à partir de l'adresse du premier élément du tableau (`T[0]`), de la taille d'un élément du tableau, et de l'indice `i` :
  - `T[i]` = contenu de l'élément `i`
  - `@T[i]` = `&T[i]` = `adresse_debut + i * taille_elem`
- Accéder à `T[i]`, c'est donc déréférencer la case d'adresse `&T[i]`, soit d'adresse `&T[0] + i * taille_elem`

Exemples de tableaux

Tableau d'entiers

- Soit la variable globale `tab` :

```
int tab[5] = { 2, -1, 3, 8, 7 };
```

- **Hypothèse** : l'adresse de début de `tab` est 0x10010008
  - **Remarque** : cette adresse doit être un multiple de 4
- Représentation de la mémoire :

0x10010008	0x1001000c	0x10010010	0x10010014	0x10010018
2	-1	3	8	7

- Directive assembleur pour déclarer le tableau tab dans la section data :

```
tab : .word 2, -1, 3, 8, 7
```

- `&tab[2] = &tab[0] + 2 * 4 = 0x10010008 + 8 = 0x10010010`
- `tab[2]` = contenu de la case d'index 2 = 3

Exemples de tableaux

Tableau de caractères

- Soit la variable globale str :

```
char str[] = "Hello";
```

- Hypothèse : l'adresse de début du tableau est 0x10010010
- Représentation de la mémoire :

0x10010010	0x10010011	0x10010012	0x10010013	0x10010014	0x10010015
'H' (0x48)	'e' (0x65)	'l' (0x6c)	'l' (0x6c)	'o' (0x6f)	'\0' (0x00)

- `str[3]` = contenu du mot d'adresse `&str[3]`
- `&str[3] = &str[0] + 3 * 1 = 0x10010010 + 3 = 0x10010013`
- `str[3] = 0x6c`

Exemple de manipulation d'un tableau

Code C

```
int i = 2;
char message[] = "bonjour";

int main() {
   message[i] = message[i] - 0x20;
   i = i + 1;
   message[i] = message[i] - 0x20;

   printf("%s", message);
   return 0;
}
```

Exemple de manipulation d'un tableau

Code Mips (début)

```
.data
i: .word 0x2
message: .asciiz "bonjour"

.text
  lui $8, 0x1001        # $8 = @i = 0x10010000
  lui $10, 0x1001       # $10 = 0x10010000
  ori $10, $10, 4       # $10 = @message = 0x10010004
```

Exemple de manipulation d'un tableau

Code Mips (suite)

```
# message[i] = message[i] - 0x20
lw    $9, 0($8)          # lecture i
addu  $11, $10, $9       # @message[i] = @message + i*1
lb    $12, 0($11)        # lecture message[i]
addiu $12, $12, -0x20    # calcul de la nouvelle valeur
lw    $9, 0($8)          # lecture i
addu  $11, $10, $9       # @message[i] = @message + i*1
sb    $12, 0($11)        # écriture message[i]

# i = i + 1
lw    $9, 0($8)          # lecture i
addiu $9, $9, 1          # calcul i + 1
sw    $9, 0($8)          # écriture i
```

Exemple de manipulation d'un tableau

Code Mips (fin)

```
# message[i] = message[i] - 0x20
lw     $9, 0($8)         # lecture i
addu   $11, $10, $9      # @message[i] = @message + i*1
lb     $12, 0($11)       # lecture message[i]
addiu  $12, $12, -0x20   # calcul de la nouvelle valeur
lw     $9, 0($8)         # lecture i
addu   $11, $10, $9      # @message[i] = @message + i*1
sb     $12, 0($11)       # écriture message[i]

# printf("%s", message)
ori $2, $0, 4  # Appel système 4 (affichage chaine)
ori $4, $10, 0 # $4 <- @message[0]
syscall

# exit
ori $2, $0, 10
syscall
```

1. Architecture de la mémoire

2. Adresse des variables globales et instructions de transfert mémoire

3. Implantation des tableaux en assembleur

4. Sauts inconditionnels et conditionnels
   - Nécessité de la rupture de séquence
     - Exemple de programme sans rupture de séquence
     - Exemple de programme avec rupture de séquence
   - Instructions MIPS
   - Exemple
   - Détermination de l'adresse de saut

5. Réalisation des structures de contrôle des langages de haut niveau à l'aide de sauts

Exemple de programme sans rupture de séquence

Ajout des N premiers entiers

Adresse d'implantation	Programme Assembleur
	.data
	.text
0x00400000	xor $8, $8, $8 # $8 =
0x00400004	xor $9, $9, $9 # $9 =
0x00400008	addiu $8, $8, 1 # $8 =
0x0040000c	addu $9, $9, $8 # $9 =
0x00400010	addiu $8, $8, 1 # $8 =
0x00400014	addu $9, $9, $8 # $9 =
0x00400018	addiu $8, $8, 1 # $8 =
0x0040001c	addu $9, $9, $8 # $9 =
0x00400020	ori $2, $0, 10 # $2 =
0x00400024	syscall

Exemple de programme sans rupture de séquence

Ajout des N premiers entiers

Adresse d'implantation	Programme Assembleur
	.data
	.text
0x00400000	xor $8, $8, $8 # $8 = 0
0x00400004	xor $9, $9, $9 # $9 = 0
0x00400008	addiu $8, $8, 1 # $8 = 1
0x0040000c	addu $9, $9, $8 # $9 = 1
0x00400010	addiu $8, $8, 1 # $8 = 2
0x00400014	addu $9, $9, $8 # $9 = 1 + 2
0x00400018	addiu $8, $8, 1 # $8 = 3
0x0040001c	addu $9, $9, $8 # $9 = 1 + 2 + 3
0x00400020	ori $2, $0, 10 # $2 = 10
0x00400024	syscall

- On peut factoriser le code identique : utiliser un saut pour ré-exécuter les instructions

Exemple de programme avec rupture de séquence

Ajout des N premiers entiers

```
                              .data
                              .text
0x00400000                    xor   $8, $8, $8 # $8 = 0
0x00400004                    xor   $9, $9, $9 # $9 = 0
                      debut:
0x00400008                    addiu $8, $8, 1  # $8 = 1,2,3,4...
0x0040000c                    addu  $9, $9, $8 # $9 = 1,3,6,10...
0x00400010                    j     debut
0x00400014                    ori   $2, $0, 10 # $2 =
0x00400018                    syscall
```

- L'étiquette `debut` représente l'adresse d'implantation de l'instruction `addiu $8, $8, 1` (soit l'adresse 0x00400008)

Nécessité de la rupture de séquence

Rupture de séquence

- Quitte le modèle d'exécution implicite en attribuant à PC une valeur autre que PC+4
  - Nécessaire lorsque code différent à exécuter selon une condition
  - Nécessaire si exécution d'une même séquence d'instructions `n` fois, `n` grand ou variable (recopie de la séquence `n` fois non possible) ⇒ Pliage du code
- Exemple précédent : saut inconditionnel ⇒ boucle infinie
- Pour sortir d'une boucle après 3 ou n itérations ⇒ saut conditionnel
  - Le saut est réalisé si une condition est vraie, sinon l'exécution continue en séquence
  - Utiliser un registre initialisé à 0, lui ajouter 1 avant chaque branchement conditionnel, se brancher en début de boucle si le registre compteur n'a pas atteint la valeur 3 (ou `n`).
  - Utiliser un registre initialisé à 3 (ou `n`), lui soustraire 1 avant chaque branchement conditionnel, se brancher en début de boucle si le registre compteur n'a pas atteint la valeur 0.

Instructions de saut MIPS

Sauts inconditionnels : “Jump”

- Toujours pris, la valeur de PC est toujours modifiée
- Deux formes :
  - `j label` : met dans PC la valeur associée à l’étiquette `label`
  - `jr $i` : met dans PC la valeur contenue dans le registre `$i`

Sauts conditionnels : “Branchements”

- Pris si la condition est vraie, sinon PC est incrémenté normalement
- Condition exprimée dans l’instruction
- La condition peut être :
  - L’égalité ou l’inégalité du contenu de deux registres
  - La comparaison du contenu d’un registre à 0 ($< 0, \leq 0, > 0, \geq 0$)

Instructions de saut MIPS

Saut conditionnels avec condition d'égalité/inégalité du contenu de deux registres

- `beq $10, $8, label # beq = branch if equal`
  - Si $10 = $8 alors branchement à l'étiquette label, sinon exécution séquentielle
- `bne $10, $8, label # bne = branch if not equal`
  - Si $10 $\neq$ $8 alors branchement à l'étiquette label, sinon exécution séquentielle

Instructions de saut MIPS

Saut conditionnels avec comparaison à 0 du contenu d'un registre

- `bgez $8, label # branch if greater or equal than zero`
  - Si \$8 $\geq$ 0 alors branchement à l'étiquette label, sinon exécution séquentielle
- `bgtz $8, label # branch if greater than zero`
  - Si \$8 $>$ 0 alors branchement à l'étiquette label, sinon exécution séquentielle
- `blez $8, label # branch if less or equal than zero`
  - Si \$8 $\leq$ 0 alors branchement à l'étiquette label, sinon exécution séquentielle
- `bltz $8, label # branch if less than zero`
  - Si \$8 $<$ 0 alors branchement à l'étiquette label, sinon exécution séquentielle

Instructions de saut MIPS

Comment réaliser un saut conditionnel avec comparaison de deux valeurs ?

- Comparaison des deux valeurs avec une **instruction de comparaison**
- Le résultat de la comparaison vaut 1 (vrai) ou 0 (faux)
- Branchement conditionnel avec le résultat

Comparaison registre-registre

- `slt $10, $8, $9 # set if less than`
- $10 = 1 si $8 < $9, $10 = 0 sinon

Instructions de saut MIPS

Comparaison registre-immédiat

- `slti $10, $8, 10 # set if less than immediate`
- `$10 = 1` si `$8 < 10`, `$10 = 0` sinon
- Avec un immédiat non signé (ex : 0x00008000) : `sltiu`

Branchement conditionnel avec le résultat d'une comparaison

- `beq $10, $0, label`
  - Saut à `label` si la comparaison est fausse
- `bne $10, $0, label`
  - Saut à `label` si la comparaison est vraie

Exemple

Somme des 3 premiers entiers

```
.data
.text
   xor    $8, $8, $8   # $8 = 0
   xor    $9, $9, $9   # $9 = 0
   ori    $16, $0, 3   # $16 = 3
debut:
   addiu  $8, $8, 1    # $8 = 1, 2, 3
   addu   $9, $9, $8   # $9 = 1, 3, 6
   bne    $8, $16, debut
   ori    $2, $0, 10
   syscall
```

- $16 représente la borne du nombre d'itérations à effectuer, initialisé à 3
- $8 représente le numéro de l'itération courante, initialisé à 0 ; il est incrémenté à chaque itération
- Sortie de boucle lorsque $8 = $16

Exemple

Somme des 3 premiers entiers – 2e version

```
.data
.text
   ori   $8, $0, 3  # $8 = 3
   xor   $9, $9, $9 # $9 = 0
debut:
   addu  $9, $9, $8 # $9 = 3, 5, 6
   addiu $8, $8, -1 # $8 = 2, 1, 0
   bgtz  $8, debut
   ori   $2, $0, 10
   syscall
```

- \$8 représente le nombre d'itérations restant à exécuter, initialisé à 3 et décrémenté à chaque itération
- Exécution séquentielle lorsque \$8 atteint 0
- Permet de gagner un registre

Autres cas de comparaison de deux valeurs

Cas et suites d'instructions correspondantes

- Saut à l'adresse `label` si $10 < $8

```
slt $9, $10, $8 # $9 vaut 1 si $10 < $8
bne $9, $0, label
```

- Saut à l'adresse `label` si $10 > $8

```
slt $9, $8, $10 # $9 vaut 1 si $8 < $10
bne $9, $0, label
```

- Saut à l'adresse `label` si $10 ≤ $8

```
slt $9, $8, $10 # $9 vaut 1 si $8 < $10
beq $9, $0, label
```

- Saut à l'adresse `label` si $10 ≥ $8

```
slt $9, $10, $8 # $9 vaut 1 si $10 < $8
beq $9, $0, label
```

Détermination de l'adresse de saut : deux formes d'adressage

Adressage absolu

- Concerne uniquement les instructions de format J
- Le label (sur 26 bits) est une partie de l'adresse à laquelle il faut se brancher
- Ex : `j label` donne `PC := PC[31:28] | I * 4`
- `I` = adresse de la cible du saut privée des bits `31:28` et avec `1:0` nuls

000010	Immédiat sur 26 bits

Détermination de l'adresse de saut : deux formes d'adressage

Adressage relatif

- Concerne les instructions `Bxx`
- Le déplacement est relatif à la valeur actuelle de PC
- Ex : `bne $9, $8, label` donne `PC := PC + 4 + (I * 4)`
- `I` = nombre d'instructions entre celle suivant le saut et la destination du saut

000101	Rx	Ry	Immédiat sur 16 bits

Détermination de l'adresse de saut

Conséquences

- Nécessité de connaitre les adresses d'implantation des instructions pour pouvoir déterminer la valeur du champ Immédiat dans les formats `J` et `I` pour les sauts
- $\Rightarrow$ L'assemblage nécessite deux passes

Passe 1 : implantation de toutes les instructions

- Assignation des adresses des instructions
- Codage des instruction (sur 32 bits) en laissant les champs `I` des instructions de saut à une valeur qui n'est pas définitive (nulle)

Passe 2 : détermination des champs laissés vides

- Les adressages absolus sont résolus : il suffit de placer la partie de l'adresse de l'instruction vers laquelle on saute dans le champ `I` de l'instruction de saut
- Les adressages relatifs sont calculés en fonction de l'adresse de l'instruction de saut et de l'adresse de l'instruction cible du saut : nombre d'instructions entre les deux

Détermination de l'adresse de saut

Première passe

Langage d'assemblage	Adresse d'implantation	Binaire
`.data`		
`.text`		
`xor $8, $8, $8`	0x00400000	0x01084026
`xor $9, $9, $9`	0x00400004	0x01294826
`ori $16, $0, 3`	0x00400008	0x34100003
`deb:`		
`addiu $8, $8, 1`	0x0040000c	0x25080001
`addu $9, $9, $8`	0x00400010	0x01284821
`beq $8, $16, fin`	0x00400014	0x1110XXXX
`j deb`	0x00400018	0b000010BBB...BBB
`fin:`		
`ori $2, $0, 10`	0x0040001c	0x3402000a
`syscall`	0x00400020	0x0000000c

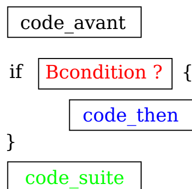
Détermination de l'adresse de saut

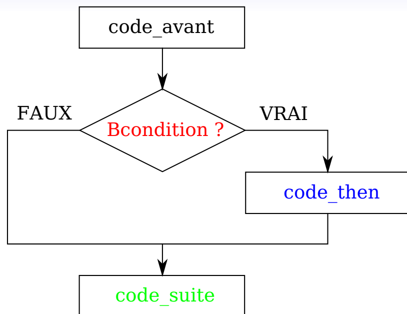
Seconde passe

Langage d'assemblage	Adresse d'implantation	Binaire
.data		
.text		
xor $8, $8, $8	0x00400000	0x01084026
xor $9, $9, $9	0x00400004	0x01294826
ori $16, $0, 3	0x00400008	0x34100003
deb:		
addiu $8, $8, 1	0x0040000c	0x25080001
addu $9, $9, $8	0x00400010	0x01284821
beq $8, $16, fin	0x00400014	0x11100001
j deb	0x00400018	0x08100003
fin:		
ori $2, $0, 10	0x0040001c	0x3402000a
syscall	0x00400020	0x0000000c

1. Architecture de la mémoire
2. Adresse des variables globales et instructions de transfert mémoire
3. Implantation des tableaux en assembleur
4. Sauts inconditionnels et conditionnels
5. **Réalisation des structures de contrôle des langages de haut niveau à l'aide de sauts**

If-Then






```
     code_avant
     Elaboration de Bcondition
     Bcondition_vraie LV
     J LF
LV:  code_then
LF:  code_suite
```

```
     code_avant
     Elaboration de Bcondition
     Bcondition_fausse LF
     code_then
LF:  code_suite
```

If-Then-Else

code_avant

if Bcondition ? {

code_then

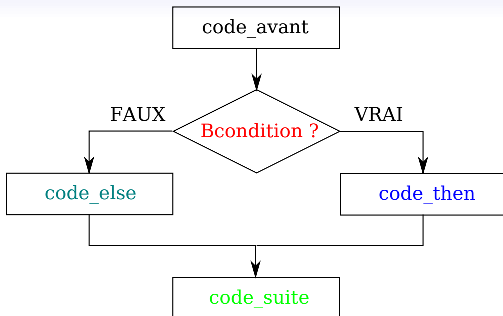
}

else {

code_else

}

code_suite




```
           code_avant
           Elaboration de Bcondition
           Bcondition_fausse label_else
           code_then
           J label_fin
label_else: code_else
label_fin:  code_suite
```

Boucle While

code_avant

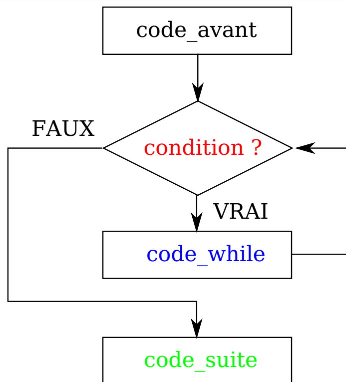
while condition {

code_while

}

code_suite

```
          code_avant
label_deb: Elaboration de condition
          Bcondition_fausse label_fin
          code_while
          J label_deb
label_fin: code_suite
```



Boucle For

code_avant

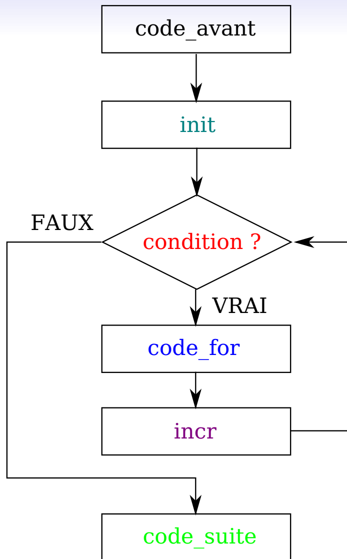
for (init ; condition ; incr) {

code_for

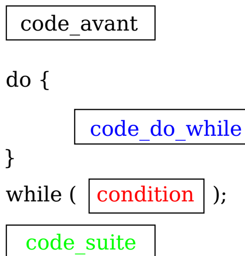
}

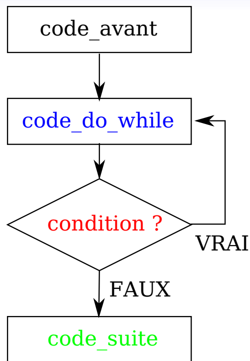
code_suite

```
           code_avant
           init
label_deb: Elaboration de condition
           Bcondition_fausse label_fin
           code_for
           incr
           J label_deb
label_fin: code_suite
```



Boucle Do-While






```
           code_avant
label_deb: code_do_while
           Elaboration de condition
           Bcondition_vraie label_deb
           code_suite
```

Exemple : code C

```
int i = 0;
char str[] = "une chaine quelconque\n";

int main() {
    while (str[i] != 0) {
        printf("%d", str[i]);
        i += 1;
    }
    return 0;
}
```

Exemple : code assembleur

Code assembleur

```
.data
   i: .word 0x0
   str: .asciiz "une chaine quelconque\n"

.text
   lui $8, 0x1001   # $8 = 0x10010000 = @i
   lui $9, 0x1001
   ori $9, $9, 4    # $9 = 0x10010004 = @str
```

Exemple : code assembleur

Code assembleur (suite)

```
debut:
   # Calcul de la condition et branchement
   lw     $10, 0($8)     # lecture i
   addu   $11, $9, $10   # @str[i] = @str + i*1
   lb     $12, 0($11)    # lecture de str[i]
   beq    $12, $0, fin

   # Corps de la boucle
   ori    $2, $0, 1      # Appel système affichage d'un entier
   lb     $4, 0($11)     # Paramètre de l'appel système
   syscall
   lw     $10, 0($8)     # lecture i
   addiu  $10, $10, 1    # i + 1
   sw     $10, 0($8)     # écriture i

   # Saut au début de la boucle
   j      debut
```

Exemple : code assembleur

Code assembleur (fin)

```
fin:
    # Exit
    ori   $2, $0, 10     # Numéro appel système exit
    syscall
```