TELECOM SudParis

IP PARIS

L’assembleur MIPS

CSC 4536

Compilation, de l’algorithme à la porte logique

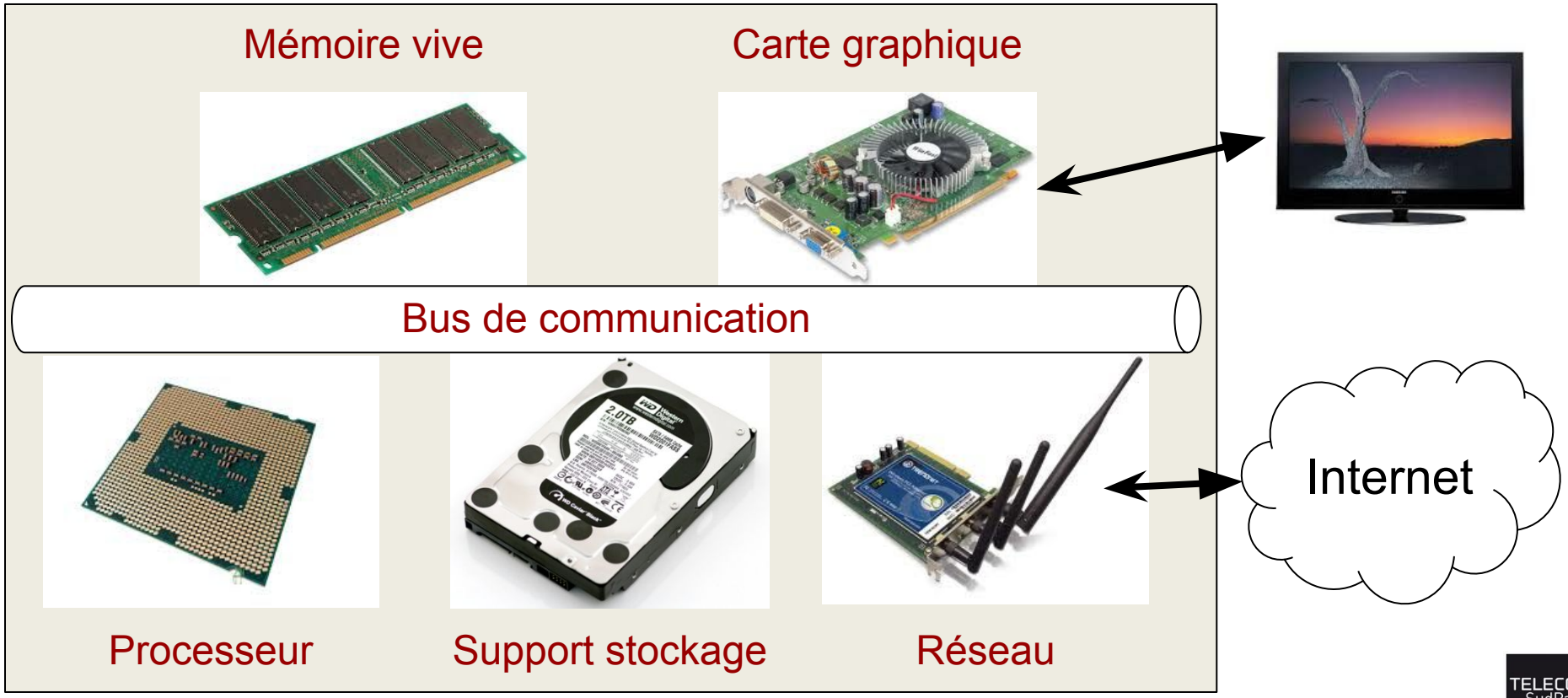
Gaël Thomas

Objectifs

- Comprendre comment fonctionne un processeur
- Être capable de programmer en assembleur MIPS

Rappel : un ordinateur

Ordinateur = un processeur, de la mémoire vive et des périphériques interconnectés par un bus de communication



Rappel : un ordinateur

- Processeur : unité capable d'effectuer des calculs, des accès à la mémoire vive et des accès aux périphériques

- Mémoire vive : stocke les données de travail du processeur
  - Accès rapide, données perdues en cas de perte de courant
  - Exemple : SDRAM (Synchronous Dynamic Random Access
  - Memory)

- Périphérique : fournit ou stocke des données secondaires
  - Clavier, souris, carte réseau, carte graphique, carte son…

- Bus de communication : interconnecte le processeur, la mémoire et les périphériques

La mémoire vive

- Mémoire : ensemble de cases numérotées stockant des octets
- Octet : regroupement de 8 bits (bytes en anglais)
- Bit : valeur valant 0 ou 1
  - 0 : non actif (“courant ne passe pas”)
  - 1 : actif (“courant passe”)
- Un octet permet de représenter $2^8 = 256$ valeurs
  - Soit un nombre entre -128 et 127
  - Soit un nombre entre 0 et 255
  - Soit un caractère (par exemple 97 == ‘a’)
  - Soit toute valeur énumérable comprise en 0 et 255

Représentation d'un octet

- Sous forme binaire : 0100 1010b
- Sous forme décimale non signée : 74 (= $2^6 + 2^3 + 2^1$)
- Souvent sous forme hexadécimale
  - 4 bits = 1 digit entre 0 et F
  - 8 bits = 2 digit entre 0 et F
  - Exemple : 0100 1010b = 0x4A

0000	0
0001	1
0010	2
0011	3

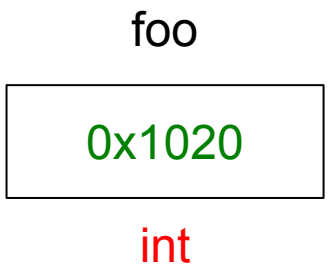
0100	4
0101	5
0110	6
0111	7

1000	8
1001	9
1010	A
1011	B

1100	C
1101	D
1110	E
1111	F

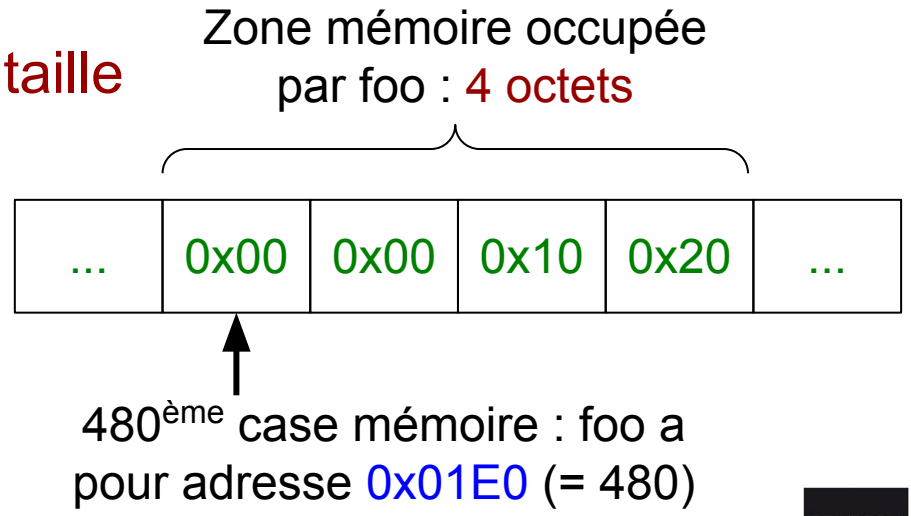
Variable et adresse mémoire

- Dans un langage de haut niveau, une variable
  - est un emplacement mémoire
  - qui possède un nom, un type et une valeur



- À l'exécution, une variable
  - est un emplacement mémoire
  - qui possède une valeur et une taille

- Le numéro de la case mémoire contenant la variable s'appelle son adresse



Le processeur

- Circuit électronique qui contient principalement
  - Des registres : sortes de variables internes du processeur
    - Souvent numéroté $0, $1 etc.
  - Des unités de calcul capable d'effectuer des calculs sur des registres
  - Une unité pour accéder à la mémoire
    - Lecture d'un ensemble d'octets de la mémoire vers un registre
    - Écriture d'un ensemble d'octets d'un registre vers la mémoire
  - Une unité pour accéder aux périphériques

- Un processeur n bits est un processeur avec des registres de calcul entier sur n bits
  - Aujourd'hui, souvent processeurs 64 bits sauf dans l'embarqué

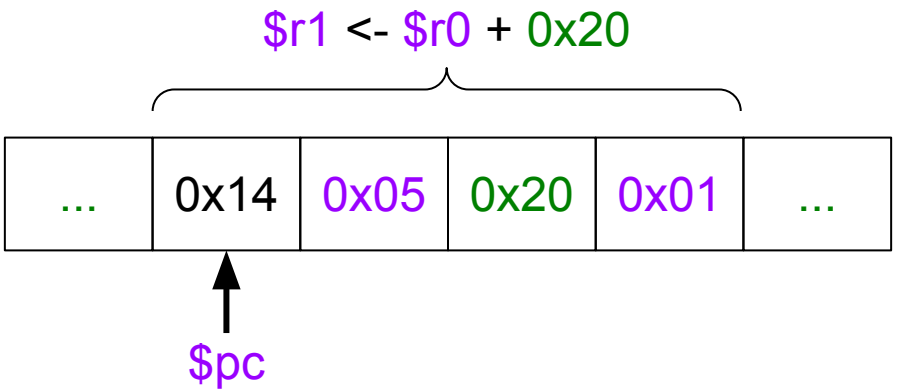
Les instructions

- Un processeur offre en général 5 types d'instructions
  - Instructions d'accès à la mémoire (lecture/écriture)
  - Instructions de calcul
  - Instructions de saut (if, appel de fonction)
  - Instructions d'accès aux périphériques
  - Instructions système (configuration de l'état du processeur)

- Une instruction est codée par une séquence d'octets
  - Exemples
    - 10 88 3 : lit les 8 octets à l'adresse 0x88 dans le registre $3
    - 14 5 20 1 : ajoute 20 au registre $5 et stocke le résultat dans $1

Le registre $pc

- Les instructions sont stockées dans la mémoire
- Le registre spécial $pc du processeur indique l'adresse de la prochaine instruction à exécuter



- Remarque : une instruction de saut est simplement une instruction qui modifie $pc (exemple $pc <- $pc + 0x12)

Une boucle simple du processeur

- Boucle infinie de 3 étapes :
  - Charge l'instruction se trouvant en $pc dans un registre interne (caché au programmeur) nommé $insn
  - Avance $pc sur l'instruction suivante
  - Décode et exécute l'instruction $insn

Le processeur MIPS

- Un ensemble de registres accessibles par le logiciel
  - Nommés $0 à $31 pour les calculs entiers
    - La plupart des registres ont un autre nom symbolique
  - Le registre $pc indique où est la prochaine instruction
  - Deux registres $hi et $lo
    - Multiplication stocke la partie haute dans $hi et basse dans $lo
    - Division stocke le quotient dans $lo et le reste dans $hi

- Un jeu d'instruction très simple pour effectuer
  - Des calculs (add, sub, …)
  - Des accès mémoire (sw, lw, ...)
  - Des chargements de constantes (li, la...)
  - Des sauts (b, beq, jal…)

L'émulateur mars

- mars = émulateur de processeur MIPS

- Intègre un petit système d'exploitation
  - Les fonctions du systèmes peuvent être appelée avec l'instruction syscall
  - Numéro de fonction systèmes appelée dans $v0 ($2)
  - Paramètre de la fonction dans $a0/$a1 ($4/$5)

- Exemple

```
li $v0, 10 # charge 10 dans le registre $v0
syscall    # l'appel système 10 correspond à "exit"
```

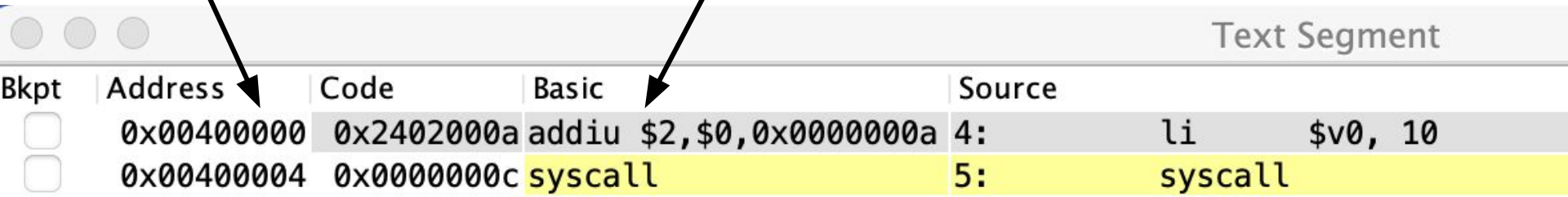
Assemblage et macro-instruction

- Un texte MIPS est traduit directement en instruction MIPS, on appelle cette transformation la phase d'assemblage

- La phase d'assemblage s'occupe de
  - Calculer les adresses des points de saut et des variables
  - Traduire certaines macro-instructions en instructions plus simple
    - Par exemple li $v0, 10 et traduit en addiu $v0, $0, 10

Calcul des adresses

Traduction macro-instruction



Un exemple complet

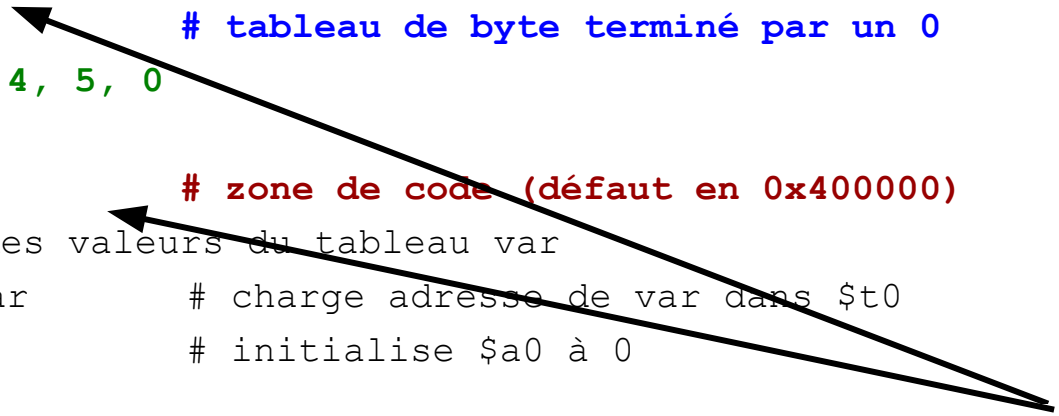
```
.data 0x10010000            # zone de donnée forcée en 0x10010000
var:                        # tableau de byte terminé par un 0
    .byte 1, 2, 3, 4, 5, 0

.text                       # zone de code (défaut en 0x400000)
# calcul la somme des valeurs du tableau var
    la      $t0, var        # charge adresse de var dans $t0
    li      $a0, 0          # initialise $a0 à 0
loop:
    lbu     $t1, ($t0)      # charge l'octet à l'adresse $t0 dans $a0
    beqz    $t1, end        # si cet octet est égal à 0, saute à la fin
    addu    $a0, $a0,  $t1  # ajoute $t1 à $a0
    addiu   $t0, $t0, 1     # passe au caractère suivant du tableau
    b       loop            # recommence à loop
end:
    li      $v0, 1          # appel système 'affiche entier en a0'
    syscall
    li   $v0, 10            # appel system 'exit'
    syscall
```

Un exemple complet

```
.data 0x10010000            # zone de donnée forcée en 0x10010000
var:                        # tableau de byte terminé par un 0
    .byte 1, 2, 3, 4, 5, 0

.text                       # zone de code (défaut en 0x400000)
# calcul la somme des valeurs du tableau var
    la      $t0, var        # charge adresse de var dans $t0
    li      $a0, 0          # initialise $a0 à 0
loop:
    lbu     $t1, ($t0)      # charge l'octet à l'adresse
    beqz    $t1, end        # si cet octet est égal à 0,
    addu    $a0, $a0,  $t1  # ajoute $t1 à $a0
    addiu   $t0, $t0, 1     # passe au caractère suivant
    b       loop            # recommence à loop
end:
    li      $v0, 1          # appel système 'affiche entier en a0'
    syscall
    li   $v0, 10            # appel system 'exit'
    syscall
```

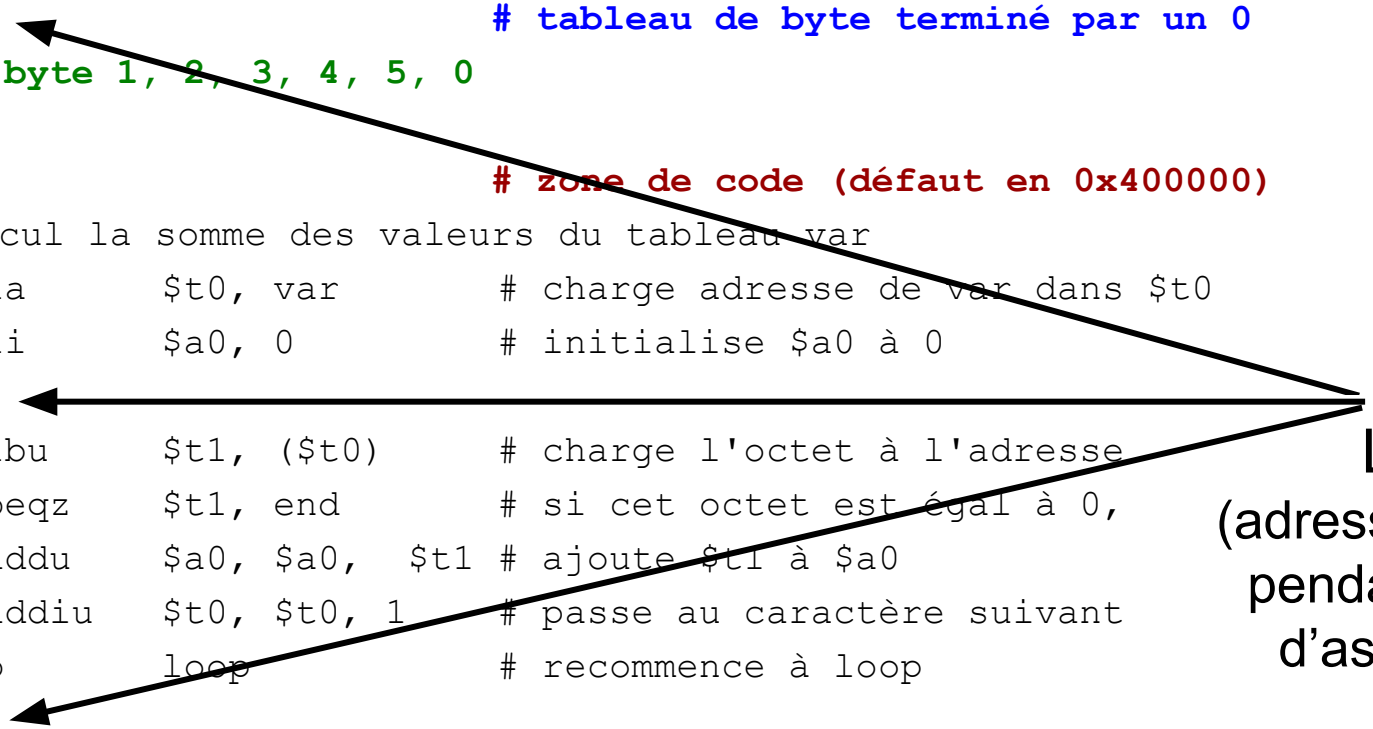


zone de données
et
zone de code

Un exemple complet

```
.data 0x10010000            # zone de donnée forcée en 0x10010000
var:                        # tableau de byte terminé par un 0
    .byte 1, 2, 3, 4, 5, 0

.text                       # zone de code (défaut en 0x400000)
# calcul la somme des valeurs du tableau var
    la      $t0, var        # charge adresse de var dans $t0
    li      $a0, 0          # initialise $a0 à 0
loop:
    lbu     $t1, ($t0)      # charge l'octet à l'adresse
    beqz    $t1, end        # si cet octet est égal à 0,
    addu    $a0, $a0,  $t1  # ajoute $t1 à $a0
    addiu   $t0, $t0, 1     # passe au caractère suivant
    b       loop            # recommence à loop
end:
    li      $v0, 1          # appel système 'affiche entier en a0'
    syscall
    li   $v0, 10            # appel system 'exit'
    syscall
```



Labels
(adresses calculées pendant la phase d'assemblage)

Un exemple complet

```
.data 0x10010000            # zone de donnée forcée en 0x10010000
var:                        # tableau de byte terminé par un 0
     .byte 1, 2, 3, 4, 5, 0

.text                       # zone de code (défaut en 0x400000)
# calcul la somme des valeurs du tableau var
     la      $t0, var       # charge adresse de var dans $t0
     li      $a0, 0         # initialise $a0 à 0
loop:
     lbu     $t1, ($t0)     # charge l'octet à l'adresse
     beqz    $t1, end       # si cet octet est égal à 0,
     addu    $a0, $a0,  $t1 # ajoute $t1 à $a0
     addiu   $t0, $t0, 1    # passe au caractère suivant
     b       loop           # recommence à loop
end:
     li      $v0, 1         # appel système 'affiche enti
     syscall
     li   $v0, 10           # appel system 'exit'
     syscall
```

```
for($t0 = var;
    ($t0) != 0;
    $t0++) {
  $a0 += $t0
}
print $a0
exit
```

Exécution d'un programme avec mars

- Étape 1 : assemblage
  - Transforme le texte en séquence d'instructions
  - Menu (Run) -> (Assembler) dans mars (F3)

- Étape 2 : chargement en mémoire
  - Automatique après la phase d'assemblage dans mars
  - Par défaut le code est chargé en 0x400000 et les données en 0x10010000
    - Dans notre exemple, on force le chargement des données en 0x10010000 pour illustrer

- Étape 3 : exécution
  - Dans mars, on peut exécuter pas à pas, avancer, reculer etc...

Et maintenant

À vous de jouer !