Architecture des ordinateurs
Cours 7

Responsable de l'UE : Karine Heydemann

Contact : prenom.nom@lip6.fr

Contrôles à venir

Partiel

- Lundi 8 Novembre de 8h30 à 10h
- Le mémento MIPS est le seul document autorisé
- Le programme du partiel : cours 1 à 6, TD-TME séances 1 à 6

TME solo

- Lundi 13 décembre 8h30 à 9h25 ou 9h30 à 10h25
- Le mémento MIPS est le seul document autorisé
- Le programme du TME solo : cours 1 à 8, TD-TME séances 1 à 8

Plan du cours 7

Fonctions et appels de fonction

Fonction

Morceau de programme qui :

- reçoit des arguments
- renvoie un résultat
- peut accéder aux variables globales
- a des variables locales qui lui sont propres

```
type_res fct_name(type_arg1 arg1,
                  ...,
                  type_argN argN)
  /* déclarations */
  type_var1 var1;
  ...
  type_varN varN;
  /* instructions */
  instruction1;
  ...
  instructionN;
  /* retour a l'appelant */
  return val;
```

Appel de fonction

Expression

- Valeur : celle du résultat renvoyé par la fonction
- Effet :
  - aller exécuter le code d'une fonction
  - avec les valeurs spécifiées pour ses arguments
  - en revenir avec la valeur du résultat

```
{
  /* code avant */
  ...
  /* appel de fonction */
  res = fct_name(arg1,arg2,...,argN);

  /* code suite */
  ...
}
```

Fonctions et appels de fonction en assembleur

Fonction

Suite d'instructions assembleur qui :

- a un point d'entrée ≡ @1ère instruction
- reçoit des arguments ≡ ???
- a un point de sortie et un résultat ≡ saut retour avec une valeur ???
- peut accéder aux variables globales
- a des variables locales ≡ emplacement dans son contexte d'exécution

Appel de fonction

Suite d'instructions permettant :

- aller exécuter le code d'une fonction `f` permettant de revenir en séquence ≡ saut à f avec adresse de retour
- avec des valeurs spécifiques pour ses arguments ≡ ???
- récupérer le résultat renvoyé par la fonction ≡ ???

```
type_res fct_name(type_arg1 arg1,
                  ...,
                  type_argN argN)
  /* declarations */
  type_var1 var1;
  ...
  type_varN varN;
  /* instructions */
  instruction1;
  ...
  instructionN;
  /* retour a l'appelant */
  return val;
```

```
{
  /* code avant */
  ...
  /* appel de fonction */
  res = fct_name(arg1,arg2,...,argN);

  /* code suite */
  ...
}
```

Sauvegarde des registres

- Le code assembleur d'une fonction utilise les registres généraux pour réaliser son traitement comme tout autre code assembleur : change le contenu des registres aussi qui sont utilisés par la fonction appelante
- Besoin d'un mécanisme de sauvegarde et restauration du contenu de certains registres afin d'assurer que l'exécution après l'appel de fonction se fera avec un état des registres qui est cohérent au travers de l'appel
- La profondeur d'appel maximale n'est pas déterminable statiquement dans le cas général :
  - la pile est utilisée pour la sauvegarde des registres
  - dans le contexte d'une fonction il y a des emplacements dédiés à la sauvegarde de registres en entrée de la fonction et leur restauration à la sortie

Contrat appelant - appelé

Conventions d'appels

Règles définissant le protocole d'échange de valeurs entre une fonction appelante (celle qui appelle) et appelée (celle qui est appelée), notamment :

- Comment sont passés les paramètres (appelante $\rightarrow$ appelée)
- Comment est passée l'adresse de retour (appelante $\rightarrow$ appelée)
- Comment est passée la valeur de retour (appelée $\rightarrow$ appelante)
- Quels registres sont à sauvegarder, comment et par qui (appelée ou appelant)

Les conventions d'appel sont définies dans l'*Application Binary Interface* d'une architecture

Utilité et importance des conventions d'appel

- Les respecter permet à une fonction d'être appelée par tout code qui connaît sa signature (nom, nombre, ordre et type des paramètres)
- Les respecter permet d'appeler une fonction juste en connaissant sa signature
- Les respecter permet d'écrire du code compatible avec du code qui les respecte aussi : fonctions de bibliothèques, code généré par un autre compilateur, code assembleur écrit par collègue ou une entité tierce

Application Binary Interface

ABI selon Wikipedia

En informatique, une *Application Binary Interface* (ABI, interface binaire-programme), décrit une interface de bas niveau entre les applications et le système d'exploitation, entre une application et une bibliothèque ou bien entre différentes parties d'une application.

[...]

Une ABI définit notamment des conventions d'appel des fonctions pour une architecture donnée.

C'est l'ABI qui définit le rôle précis des registres généraux (paramètres de fonctions, résultats de fonctions, variables temporaires ?) et la responsabilité de leur intégrité (appelant ou appelé).

C'est l'ABI qui définit la structure de la pile, notamment l'organisation des emplacements réservés aux paramètres supplémentaires d'appel d'une fonction, à la sauvegarde de certains registres, à l'allocation de mémoire dynamiquement sur la pile (taille connue à la compilation) selon la portée de l'identifiant.

Dans ce cours, on utilise une version simplifiée de l'ABI
`https://refspecs.linuxfoundation.org/elf/mipsabi.pdf`.

Conventions MIPS pour le passage des paramètres

- Les **4 premiers paramètres sont passés par les registres `$4`, `$5`, `$6`, `$7`.**
- Les **paramètres suivants sont passés dans la pile** : des emplacements dédiés dans le contexte d'exécution de la fonction appelante sont nécessaires pour ce passage + écriture de leur valeur juste avant d'aller exécuter le code de la fonction appelée
- Explications : Le passage par registre des 4 premiers paramètres évite des écritures en pile, souvent moins de 4 paramètres pour les fonctions
- **Une place est TOUJOURS réservée sur la pile pour les paramètres dont la valeur est passée par registre (max 4 premiers)**
- Explications : une fonction peut sauvegarder la valeur des paramètres dans leur emplacement parce que
  - son code n'est pas optimisé, ainsi les paramètres comme les variables locales ne sont pas optimisés en registre (utilisation d'un paramètre $\Rightarrow$ lecture depuis l'emplacement du paramètre)
  - la fonction appelle aussi des fonctions et donc utilise les registres `$4`, `$5`, `$6`, `$7` pour le passage des paramètres lors de ces appels
  - la fonction a besoin de beaucoup de registres pour ses calculs intermédiaires, sauvegarder la valeur des arguments sur la pile permet de libérer les registres contenant la valeur des paramètres

Conventions MIPS pour le passage des paramètres

- Les **4 premiers paramètres sont passés par les registres `$4`, `$5`, `$6`, `$7`.**
- Les **paramètres suivants sont passés dans la pile** : des emplacements dédiés dans le contexte d'exécution de la fonction appelante sont nécessaires pour ce passage + écriture de leur valeur juste avant d'aller exécuter le code de la fonction appelée
- Explications : Le passage par registre des 4 premiers paramètres évite des écritures en pile, souvent moins de 4 paramètres pour les fonctions
- **Une place est TOUJOURS réservée sur la pile pour les paramètres dont la valeur est passée par registre (max 4 premiers)**
- Explications : une fonction peut sauvegarder la valeur des paramètres dans leur emplacement parce que
  - son code n'est pas optimisé, ainsi les paramètres comme les variables locales ne sont pas optimisés en registre (utilisation d'un paramètre $\Rightarrow$ lecture depuis l'emplacement du paramètre)
  - la fonction appelle aussi des fonctions et donc utilise les registres `$4`, `$5`, `$6`, `$7` pour le passage des paramètres lors de ces appels
  - la fonction a besoin de beaucoup de registres pour ses calculs intermédiaires, sauvegarder la valeur des arguments sur la pile permet de libérer les registres contenant la valeur des paramètres

Conventions MIPS pour le passage des paramètres

- Les **4 premiers paramètres sont passés par les registres `$4`, `$5`, `$6`, `$7`.**
- Les **paramètres suivants sont passés dans la pile** : des emplacements dédiés dans le contexte d'exécution de la fonction appelante sont nécessaires pour ce passage + écriture de leur valeur juste avant d'aller exécuter le code de la fonction appelée
- Explications : Le passage par registre des 4 premiers paramètres évite des écritures en pile, souvent moins de 4 paramètres pour les fonctions
- **Une place est TOUJOURS réservée sur la pile pour les paramètres dont la valeur est passée par registre (max 4 premiers)**
- Explications : une fonction peut sauvegarder la valeur des paramètres dans leur emplacement parce que
  - son code n'est pas optimisé, ainsi les paramètres comme les variables locales ne sont pas optimisés en registre (utilisation d'un paramètre $\Rightarrow$ lecture depuis l'emplacement du paramètre)
  - la fonction appelle aussi des fonctions et donc utilise les registres `$4`, `$5`, `$6`, `$7` pour le passage des paramètres lors de ces appels
  - la fonction a besoin de beaucoup de registres pour ses calculs intermédiaires, sauvegarder la valeur des arguments sur la pile permet de libérer les registres contenant la valeur des paramètres

Conventions MIPS pour le passage des paramètres

- Les **4 premiers paramètres sont passés par les registres `$4`, `$5`, `$6`, `$7`.**
- Les **paramètres suivants sont passés dans la pile** : des emplacements dédiés dans le contexte d'exécution de la fonction appelante sont nécessaires pour ce passage + écriture de leur valeur juste avant d'aller exécuter le code de la fonction appelée
- Explications : Le passage par registre des 4 premiers paramètres évite des écritures en pile, souvent moins de 4 paramètres pour les fonctions
- **Une place est TOUJOURS réservée sur la pile pour les paramètres dont la valeur est passée par registre (max 4 premiers)**
- Explications : une fonction peut sauvegarder la valeur des paramètres dans leur emplacement parce que
  - son code n'est pas optimisé, ainsi les paramètres comme les variables locales ne sont pas optimisés en registre (utilisation d'un paramètre $\Rightarrow$ lecture depuis l'emplacement du paramètre)
  - la fonction appelle aussi des fonctions et donc utilise les registres `$4`, `$5`, `$6`, `$7` pour le passage des paramètres lors de ces appels
  - la fonction a besoin de beaucoup de registres pour ses calculs intermédiaires, sauvegarder la valeur des arguments sur la pile permet de libérer les registres contenant la valeur des paramètres

Conventions MIPS pour l'adresse et la valeur de retour

Valeur de retour

- La valeur de retour est passée à la fonction appelante **via le registre `$2`**
  (+ `$3` lorsque sa taille dépasse 32 bits)

Adresse de retour

- L'adresse de retour dépend du site d'appel, elle correspond à l'adresse qui suit le saut vers le code de la fonction appelée
- L'adresse de retour est passée à la fonction appelée via le **registre `$31`**
- Le registre `$31` est écrit avec l'adresse de l'instruction en séquence dans le code par toutes les instructions dont le nom se termine par `al` signifiant *and link* :
  - `jal ma_fonction` met l'adresse de l'instruction qui la suit dans `$31` et met l'adresse spécifiée par l'étiquette `ma_fonction` dans `PC`
  - Dans la fonction appelée, le registre `$31` permet de revenir en séquence dans la fonction appelante via l'instruction `jr $31`
- Lorsqu'une fonction appelle une autre fonction le registre `$31` est modifié :
  $\Rightarrow$ le registre `$31` est TOUJOURS sauvegardé
  cela évite de perdre l'adresse de retour !

Conventions MIPS pour la sauvegarde des registres

- Au début d'une fonction, on pourrait sauvegarder tous les registres que celle-ci va utiliser (sauf `$2` dont la modification par la fonction appelée doit être visible pour la fonction appelante : il contient le résultat !)
- La convention MIPS définit des registres dits persistants : ce sont ceux dont la valeur doit rester intègre au travers les appels de fonction.
  - Registres persistants : `$16-$23`
  - Si une fonction utilise ces registres elle doit
    1) sauvegarder sur la pile leur valeur dans son prologue
    2) restaurer leur valeur à partir la pile dans son épilogue
- Les registres sont sauvegardés par ordre de numéro croissant des plus petites adresses aux plus grandes sur la pile : le registre de plus petit numéro est le plus proche du sommet pile.
- Dans le prologue, il faut allouer de la place sur la pile pour la sauvegarde des registres persistants + `$31` puis y écrire leur contenu
- Dans l'épilogue, il faut restaurer les registres puis désallouer les emplacements
- Si une fonction n'utilise pas ces registres, une seule sauvegarde = celle de `$31`
- Pas de sauvegarde dans le programme principal qui se termine par un `exit` !

Résumé : fonction et appel de fonction en assembleur

Fonctions

1. Alloue son contexte d'exécution sur la pile
2. Sauvegarde les registres persistants qu'elle utilise + `$31`
3. Récupère la valeur des arguments de l'appel dans `$4`-`$7` + pile
4. Réalise un traitement à partir des valeurs des arguments
5. Place la valeur du résultat dans `$2`
6. Restaure les registres persistants + `$31`
7. Désalloue son contexte d'exécution
8. Effectue un saut à l'adresse de retour avec `jr $31`

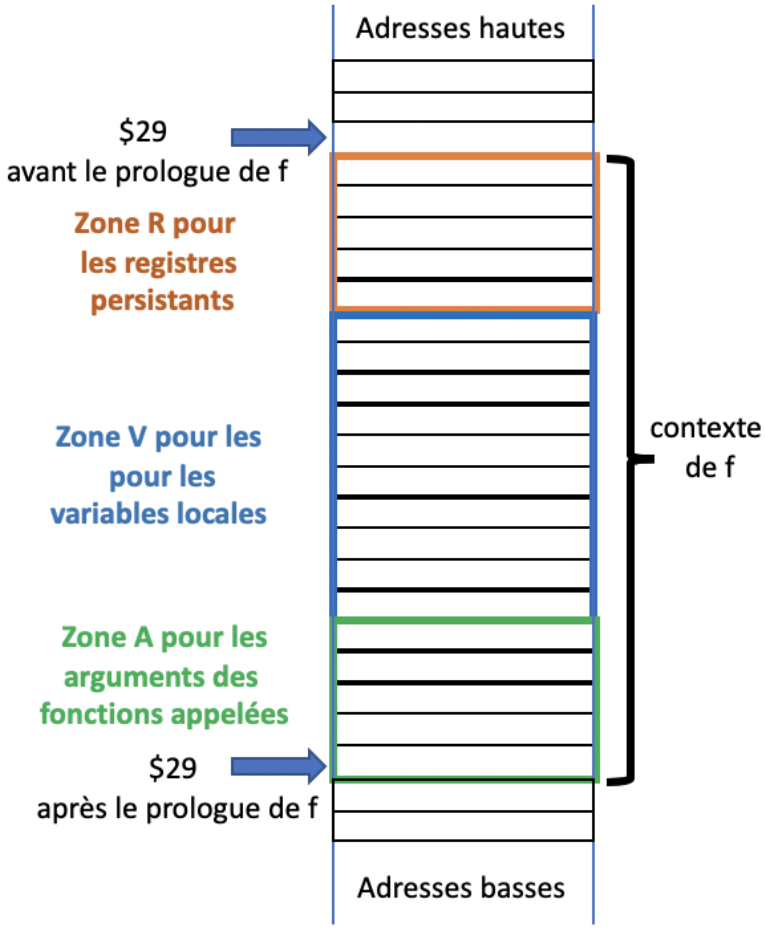
Appel de la fonction `f`

- Passage des paramètres dans `$4`-`$7` + pile
- Saut à la fonction `f` avec passage de l'adresse de retour dans `$31` avec l'instruction `jal f`
- Récupération de la valeur de retour dans `$2`

Prologue et épilogue de fonctions appelantes

Contexte d'exécution d'une fonction `f`

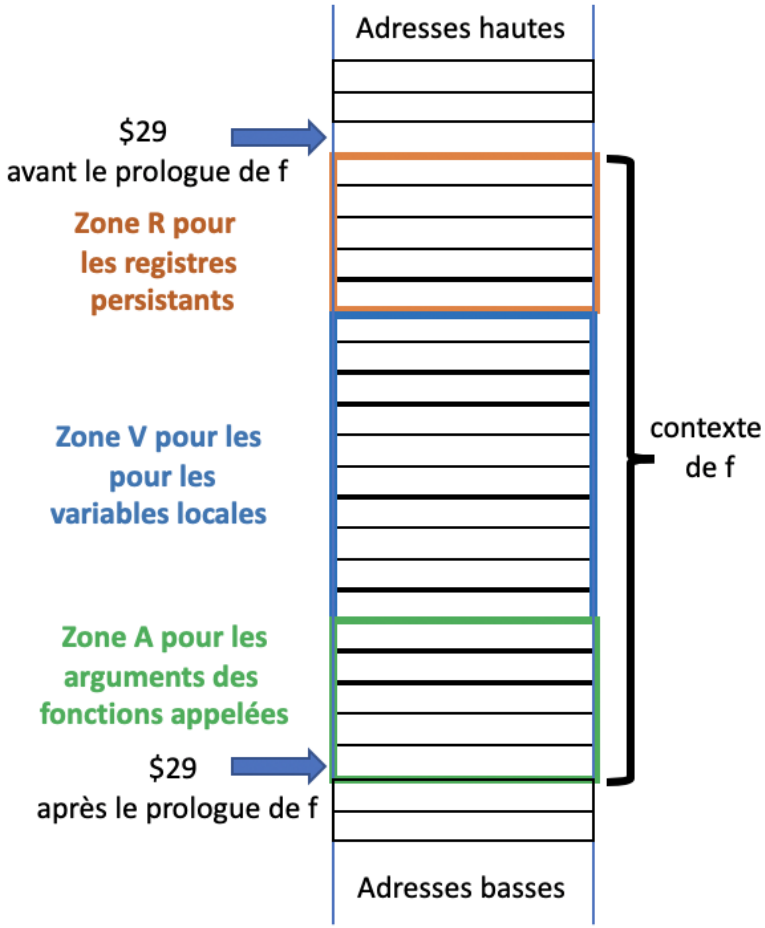
- Alloué/désalloué par la fonction `f` dans son prologue/épilogue
- Contient 3 zones d'emplacements :
  1. Zone `R` : sauvegarde des registres persistants utilisés dans la fonction `f`
  2. Zone `V` : variables locales de `f`
  3. Zone `A` : arguments des fonctions appelées par `f`



Prologue et épilogue de fonctions appelantes

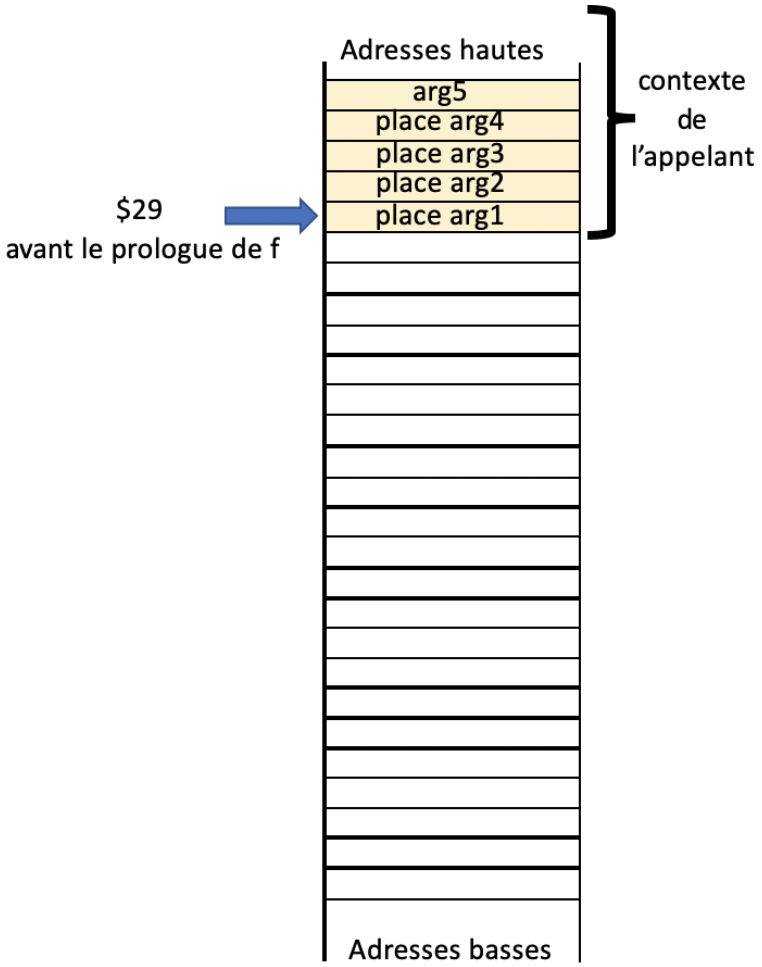
Taille en octets du contexte T = T_R + T_V + T_A

- T_R = taille de `R` vaut `4*(nr+1)`, avec `nr` le nombre de registres persistants utilisés dans le code de `f`
- T_V : taille de `V` dépend de la taille des variables locales et leurs contraintes d'alignement (cours 6)
- T_A = taille de `A` est égale à `4*na` avec `na` le nombre max de mots pour contenir les arguments des fonctions appelées par `f`.
  NB : on suppose que les 4 premiers arguments sont toujours de taille inférieure ou égale à 4.
  4 premiers arguments = 1 emplacement de 4 octets dans la zone `A` chacun.
  Arguments suivants = comme pour les variables locales



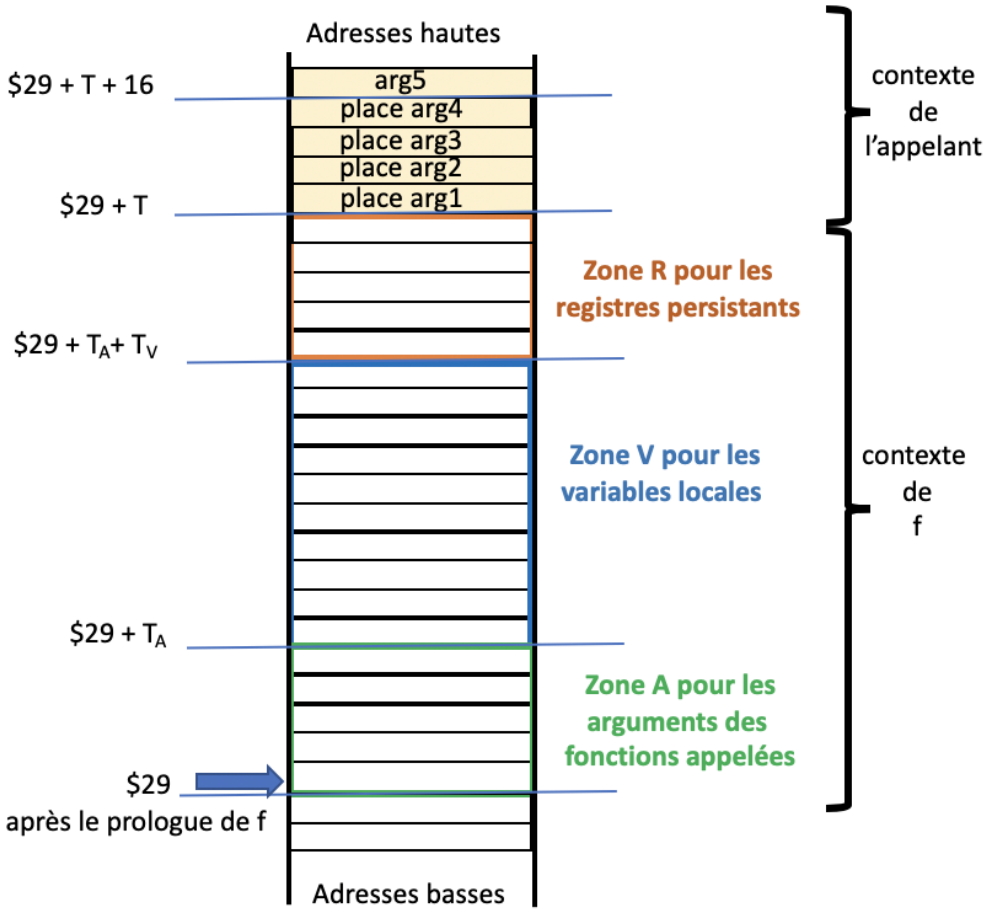
État de la pile à l'entrée d'une fonction

- Lorsqu'on arrive dans une fonction, `$29` pointe vers l'emplacement du 1er argument (contenu indéfini car il est dans `$4`) qui se trouve dans le contexte de la fonction appelante



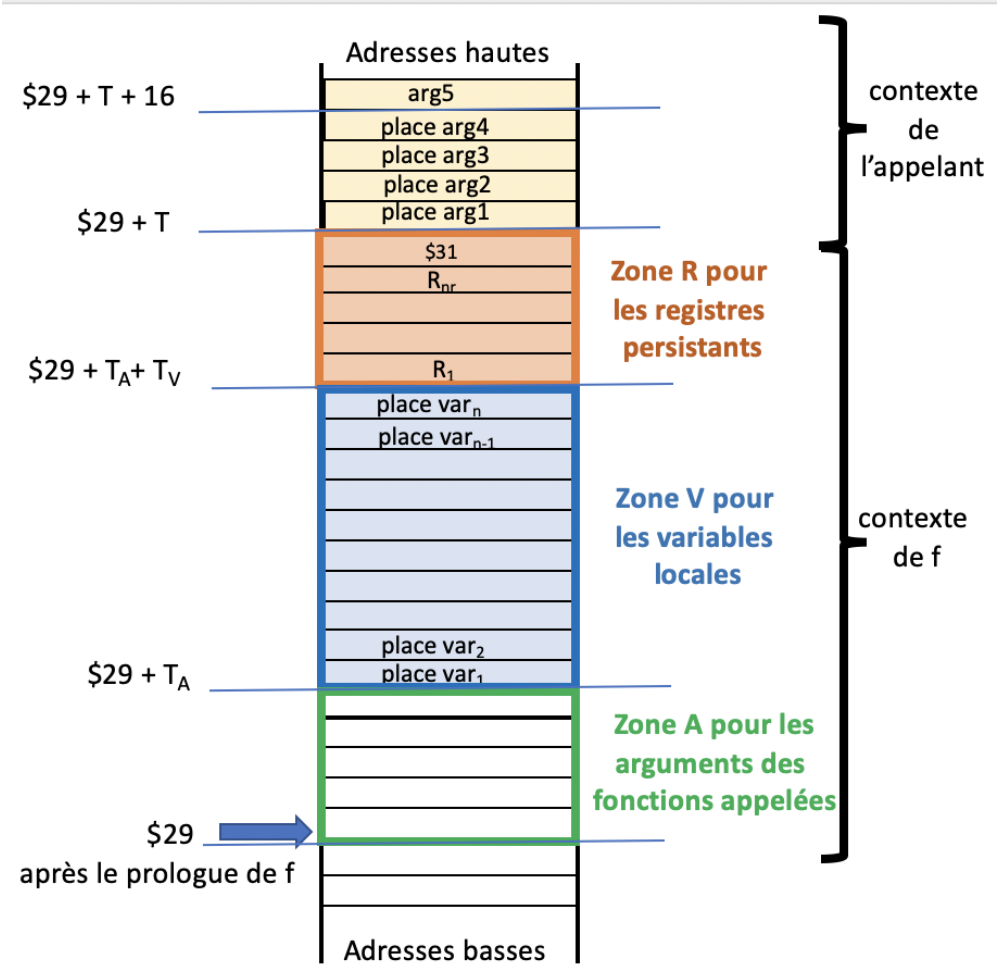
État de la pile au début du prologue d'une fonction

- La fonction alloue son contexte en décrémentant le pointeur de pile de `T` octets



Étapes dans le prologue d'une fonction et état de la pile

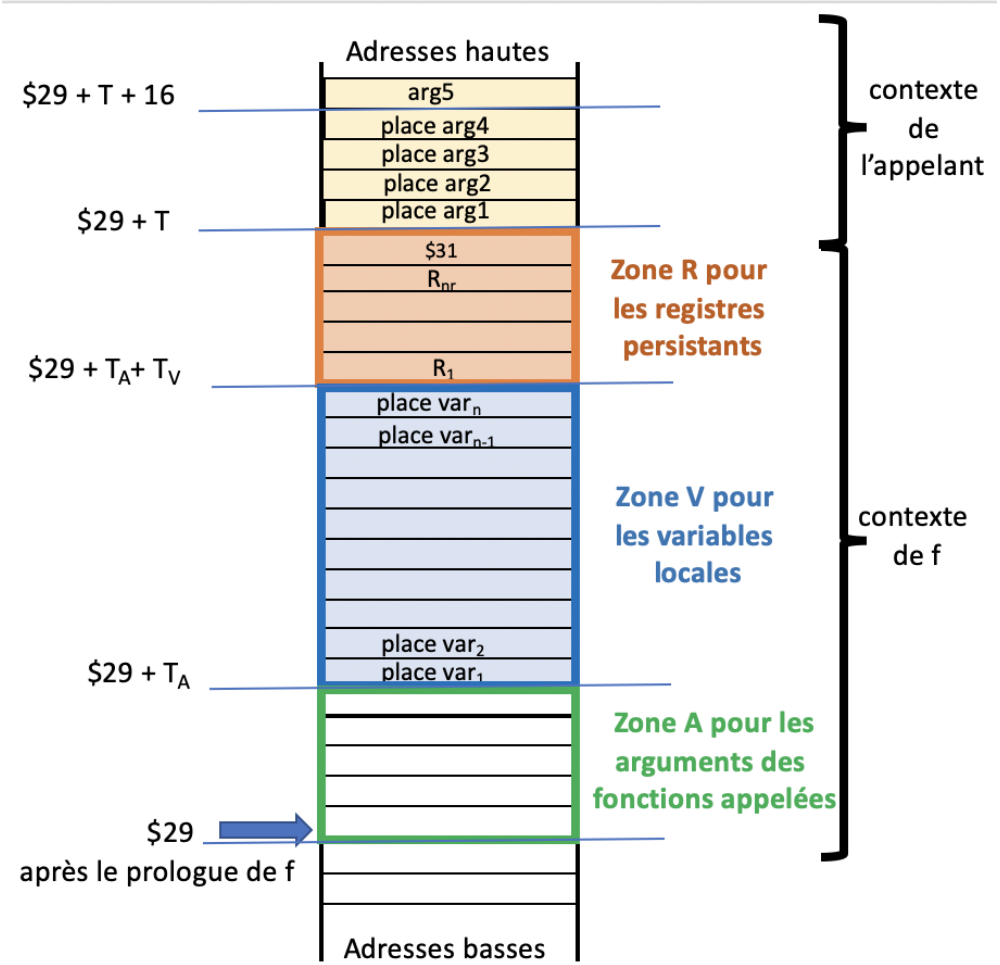
1. La fonction alloue son contexte en décrémentant le pointeur de pile de `T` octets
2. La fonction sauvegarde le contenu des registres persistants + `$31` :
   - R_1 le premier (plus petit numéro) est écrit dans la pile à `$29+T_A+T_V`,
   - R_2 le 2ème à `$29+T_A+T_V+4`,
   - ...
   - R_{nr} le dernier est écrit à `$29+T_A+T_V+T_R-8` soit `$29+T-8`
   - `$31` à `$29+T-4`
3. Si besoin, les paramètres en registre sont sauvegardés :
   - `$4` est écrit dans la pile à `$29+T`,
   - `$5` à `$29+T+4`,
   - `$6` à `$29+T+8` et
   - `$7` à `$29+T+12`



Étapes du prologue d'une fonction et état de la pile

4. Le cas échéant, la fonction lit dans la pile
   - son 5ème argument à l'adresse `$29+T+16`,
   - son 6ème argument à `$29+T+x` avec `x>16` qui dépend de la taille de 5ème argument et des contraintes d'alignement du 6ème.
   - ...
5. Si besoin, la fonction initialise ses variables locales
   - var_1 la première déclarée se trouve à $\$29+T_A$,
   - la deuxième à $\$29+T_A+taille(var)_1+align$,
   - ...

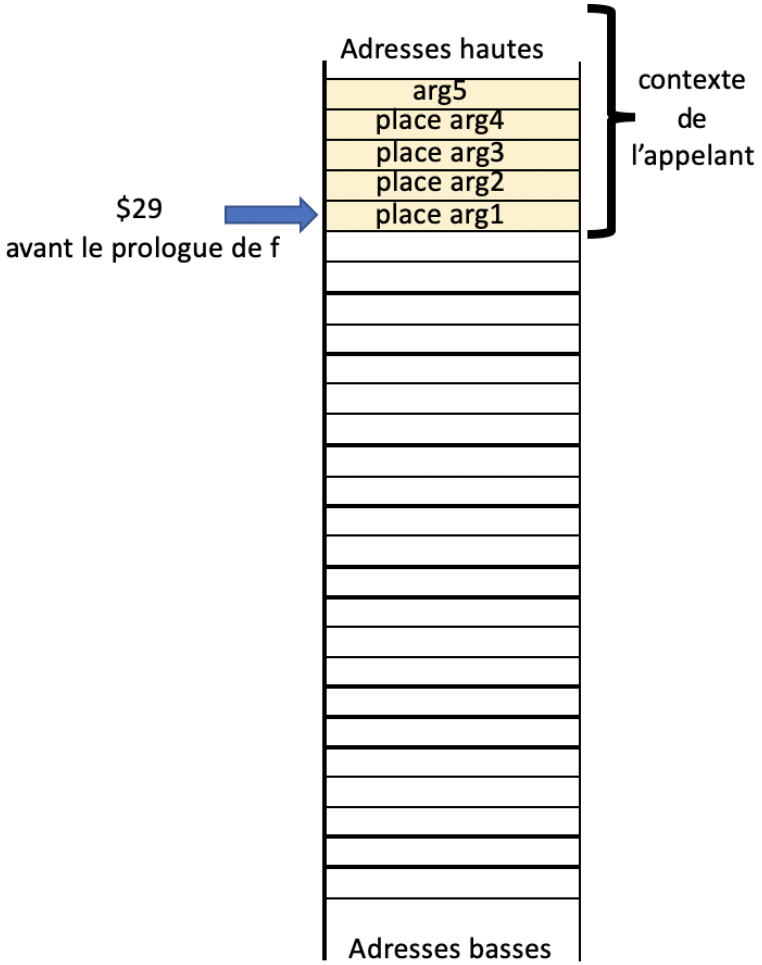
La fonction peut alors effectuer ses traitements



État de la pile à la sortie d'une fonction

- Dans l'épilogue d'une fonction, on restaure les registres persistants puis on incrémente `$29` de `T` octets. Enfin, on retourne à la fonction appelante (avec `jr $31`)
- Ainsi, au retour de la fonction dans la fonction appelante, `$29` pointe de nouveau vers l'emplacement du 1er argument dans le contexte de la fonction appelante :

  Le pointeur de pile et les registres persistants se trouvent dans le même état qu'avant la fonction



Code C exemple

```c
int x = 2, y = 4, z = 5;

void main(){
  int t1, t2;
  t1 = max2(x,y);
  t2 = max2(t1,z);
  printf("%d",t2);
  exit();
}

int max2(int a, int b){
  if (a < b)
    return b;
  else
    return a;
}
```

Code exemple : programme principal (début)

```
.data
  x: .word 2
  y: .word 4
  z: .word 5

.text
  addiu $29, $29, -16 # 4 mots : 2 var locales, 2 paramètres
                      # @t1 = $29 + 8, @t2 = $29 + 12
  # t1 = max2(x,y)
  lui $8, 0x1001          # @x dans $8 non persistant
  lw $4, 0($8)            # 1er paramètre = x dans $4
  lw $5, 4($8)            # 2ème paramètre = y dans $5
  jal max2                # premier appel àmax2
# @1 adresse retour premier appel, $2 contient le resultat
  sw $2, 8($29)           # écriture du resultat dans t1

  # t2 = max2(t1,z)
  lw $4, 8($29)           # 1er paramètre = t1 dans $4
  lui $8, 0x1001          # $8 non persistant => rechargement de @x
  lw $5, 8($8)            # 2ème paramètre = z dans $5
  jal max2                # second appel àmax2
# @2 adresse retour 2eme appel, $2 contient le resultat
  sw $2, 12($29)           # écriture du résultat dans t2
  ...
```

Code exemple : programme principal (fin)

```
# printf("%d",t2)
  lw $4, 12($29)   # lecture t2
  ori $2, $0, 1    # affichage
  syscall

# terminaison
  addiu $29, $29, +16 # désallocation emplacements pile
  ori $2, $0, 10
  syscall
```

Commentaires

- Dans le main, le prologue ne dépend que du code source (nb variables locales et leur type, nombre d'appels de fonction et leur signature)
- Il ne faut pas oublier qu'un registre non persistant (`$8` dans l'exemple) peut avoir été modifié par un appel de fonction : préférable d'utiliser des registres persistants pour les valeurs que l'on souhaite conserver au travers un appel de fonction ou que l'on ne souhaite pas recalculer/charger

Écriture d'une fonction : recommandations

Étapes conseillées

0. Se représenter la pile à l'entrée de la fonction
1. Écrire le code du corps de la fonction :
   - On commence par le corps car on ne connait pas les registres persistants à sauvegarder (prologue en 2ème étape)
   - Ecrire les lectures et écriture en pile (de paramètres ou variables locales) en laissant un ? ? pour l'immédiat de ces instructions de transfert mémoire (adresse relative au pointeur de pile)
   - Choisir les registres qu'on veut associer avec les variables locales (si optimisées en registre)
2. Déterminer la taille du contexte
   - Lister les registres persistants utilisés dans le corps, en déduire `nr` et la taille T_R sans oublier `$31`
   - Dans le code source, déterminer `na` le nombre de mots max pour stocker les arguments des fonctions appelées puis T_A.
     NB : si 4 ou moins d'arguments max, alors `na` = nombre d'arguments max.
   - À partir du code source, déduire le nombre d'octets T_V (ou le nombre de mots `nv`) nécessaires dans la zone des variables locales

Écriture d'une fonction : recommandations

Étapes conseillées

4. Écrire le prologue :
   - allocation des emplacements sur la pile,
   - sauvegarde des registres persistants.

   Si besoin dessiner la pile pour déterminer les adresses des variables locales et des paramètres passés en pile.
   - Si besoin, sauvegarde des 4 premiers paramètres, lecture des suivants.
   - Si besoin, initialisation des variables locales en pile.
5. Dans le corps de la fonction, adapter les déplacements relatifs aux pointeurs de pile quand nécessaire (accès à aux variables locales, ou aux paramètres de la fonction).
6. Écrire le prologue soit dans l'ordre :
   - Restauration des registres (= lecture des valeurs sauvegardées sur la pile),
   - Désallocation des emplacements sur la pile,
   - Retour à l'appelant

Code exemple : corps de max2

```
max2:                #code de la fonction max2
# prologue A ECRIRE


# corps de max2: $4 contient a et $5 contient b
  slt $16, $4, $6   # $16 vaut 1 si a < b
  beq $16, $0, a_max
  ori $2, $5, 0     # valeur de retour dans $2
  j fin
a_max:
  ori $2, $4, 0     # valeur de retour dans $2
fin:
# épilogue A ECRIRE
```

Code exemple : prologue et épilogue de max2

```
max2:
# prologue
  addiu $29, $29,-8 #nr = 1 ($16) + $31, nv = 0, nb_arg_max = 0
  sw $31, 4($29)
  sw $16, 0($29)

# corps de max2: $4 contient a et $5 contient b
  slt $16, $4, $5   # $5 vaut 1 si a < b
  beq $16, $0, a_max
  ori $2, $5, 0     # valeur de retour dans $2
  j fin
a_max:
  ori $2, $4, 0     # valeur de retour dans $2
fin:
# epilogue
  lw $31, 4($29)
  lw $16, 0($29)
  addiu $29, $29,+8
  jr $31
```

RMQ : en utilisant que des registres non persistants pour les registres de travail, la taille du contexte d'une fonction ne dépend que du code source (nb variables locales et leur type, nombre d'appels de fonction et leur signature)

Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

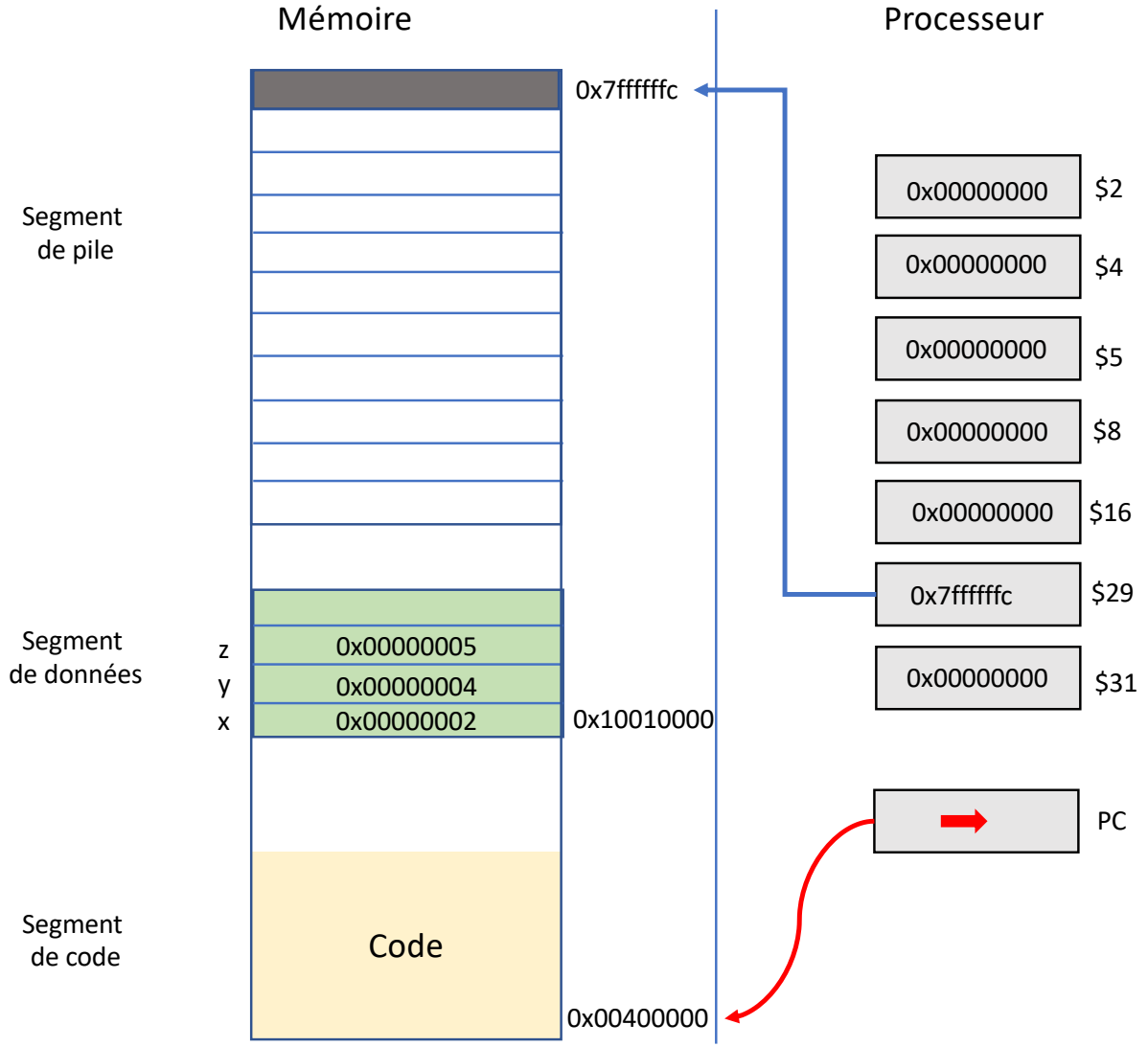


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
 # t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

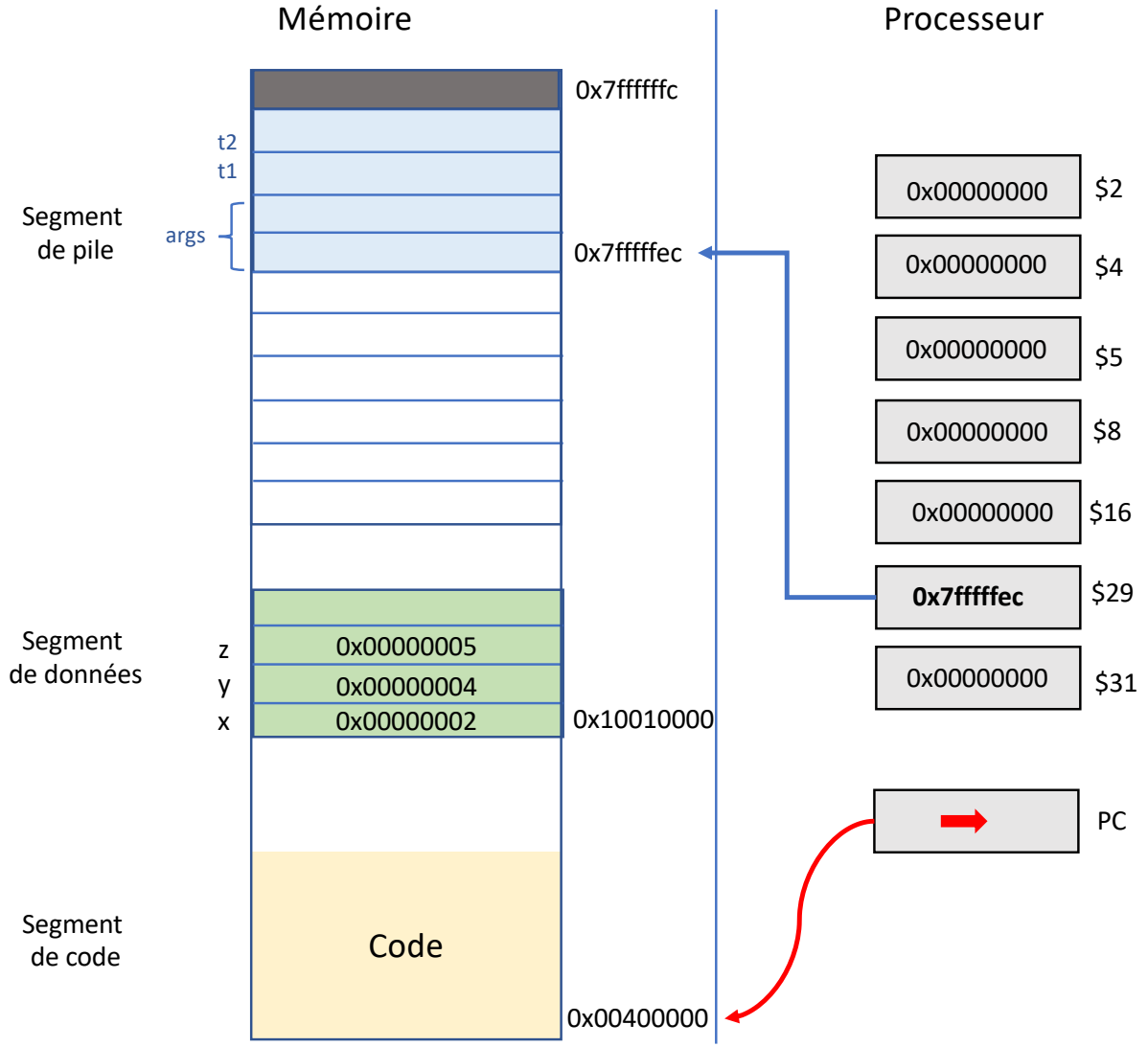


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

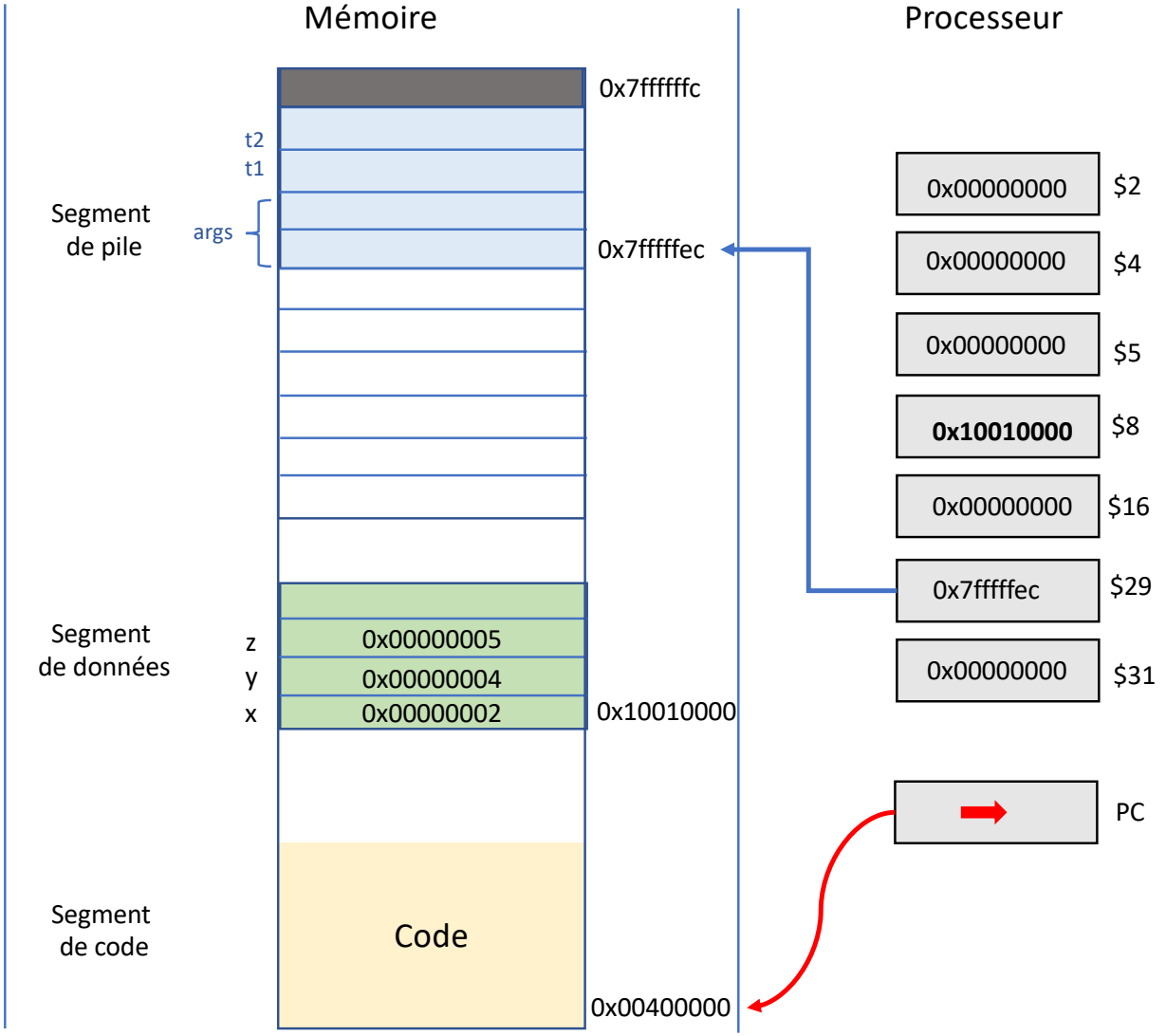


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

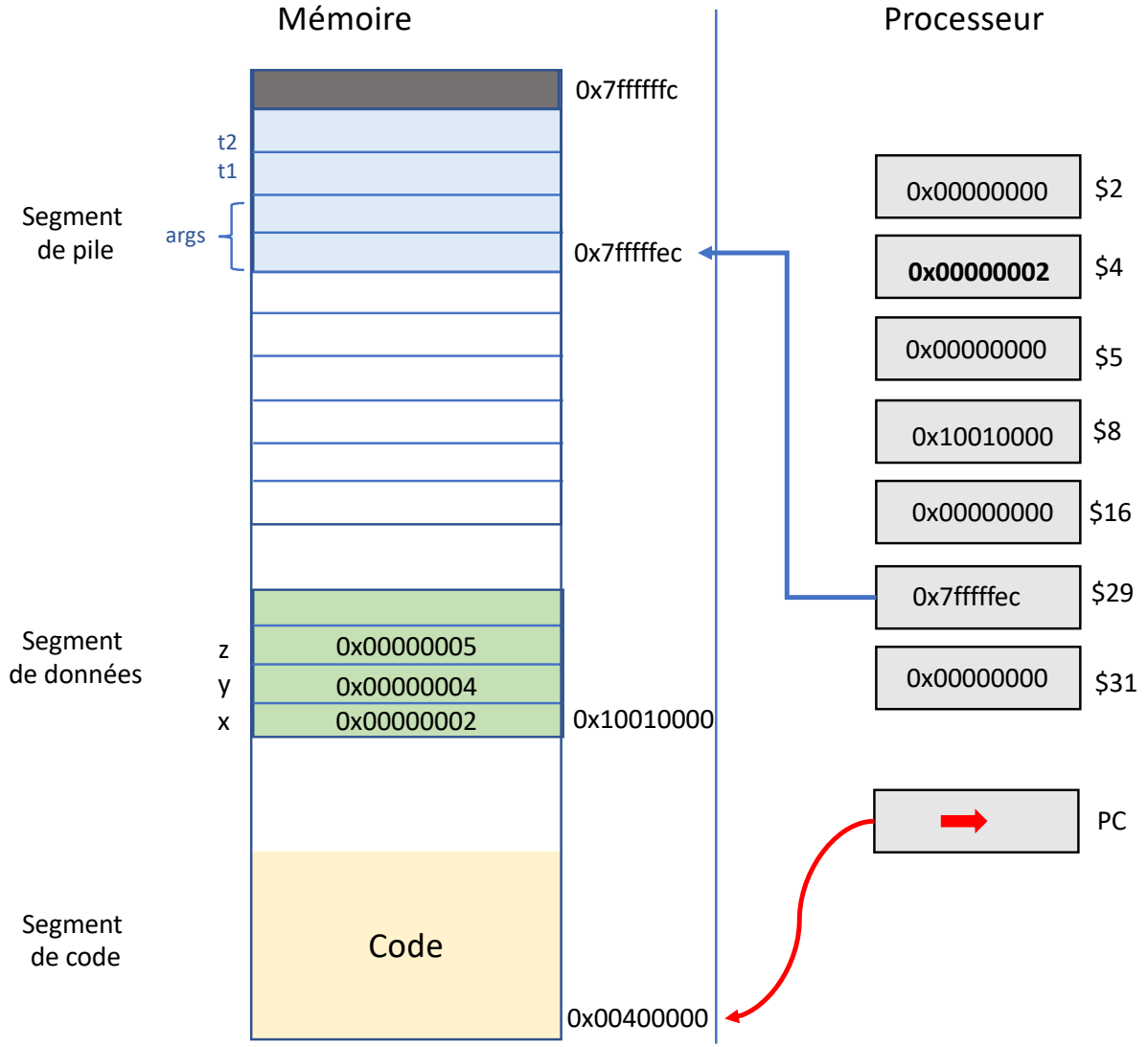


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

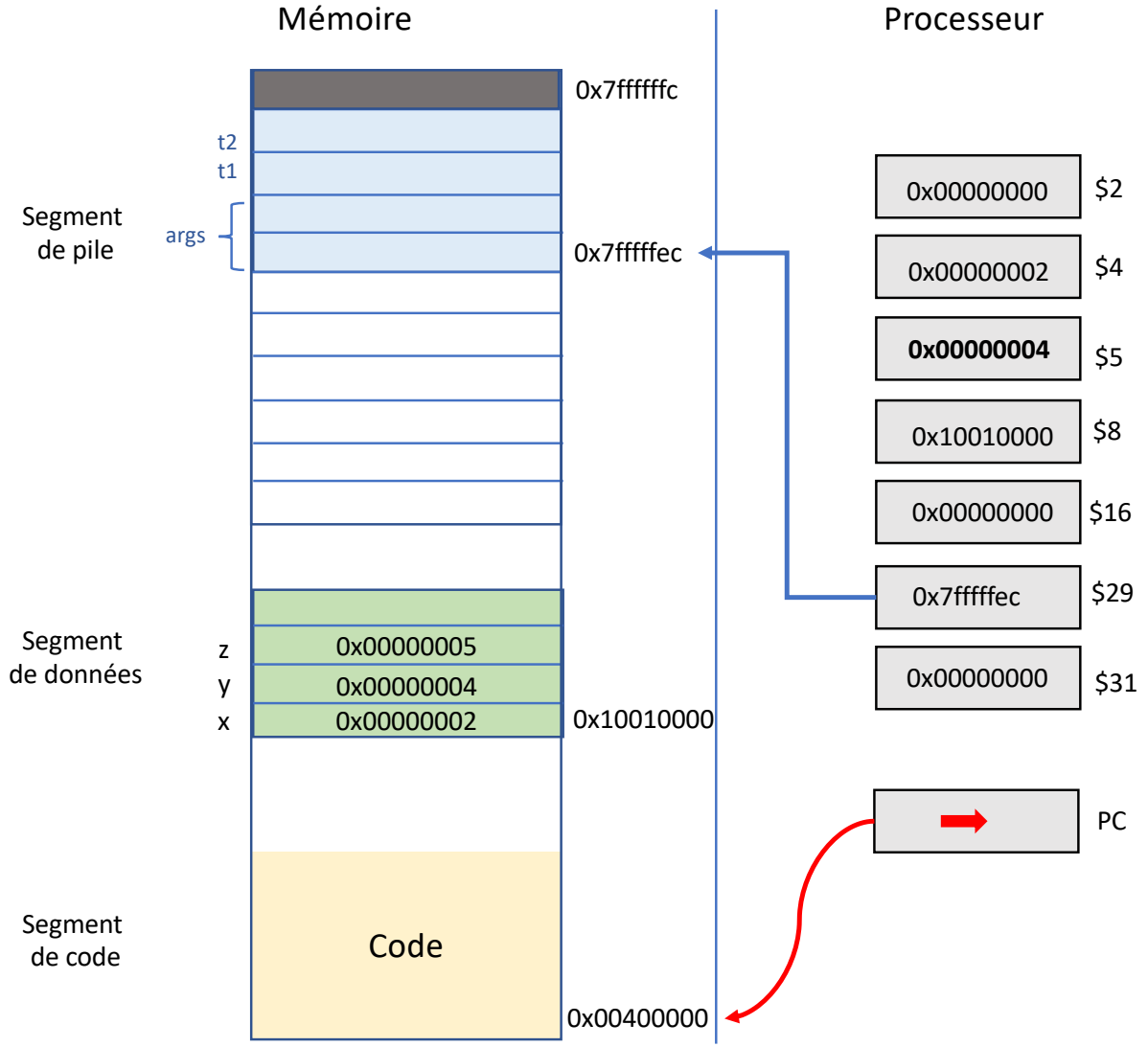


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8         # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5       #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0         # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0         # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

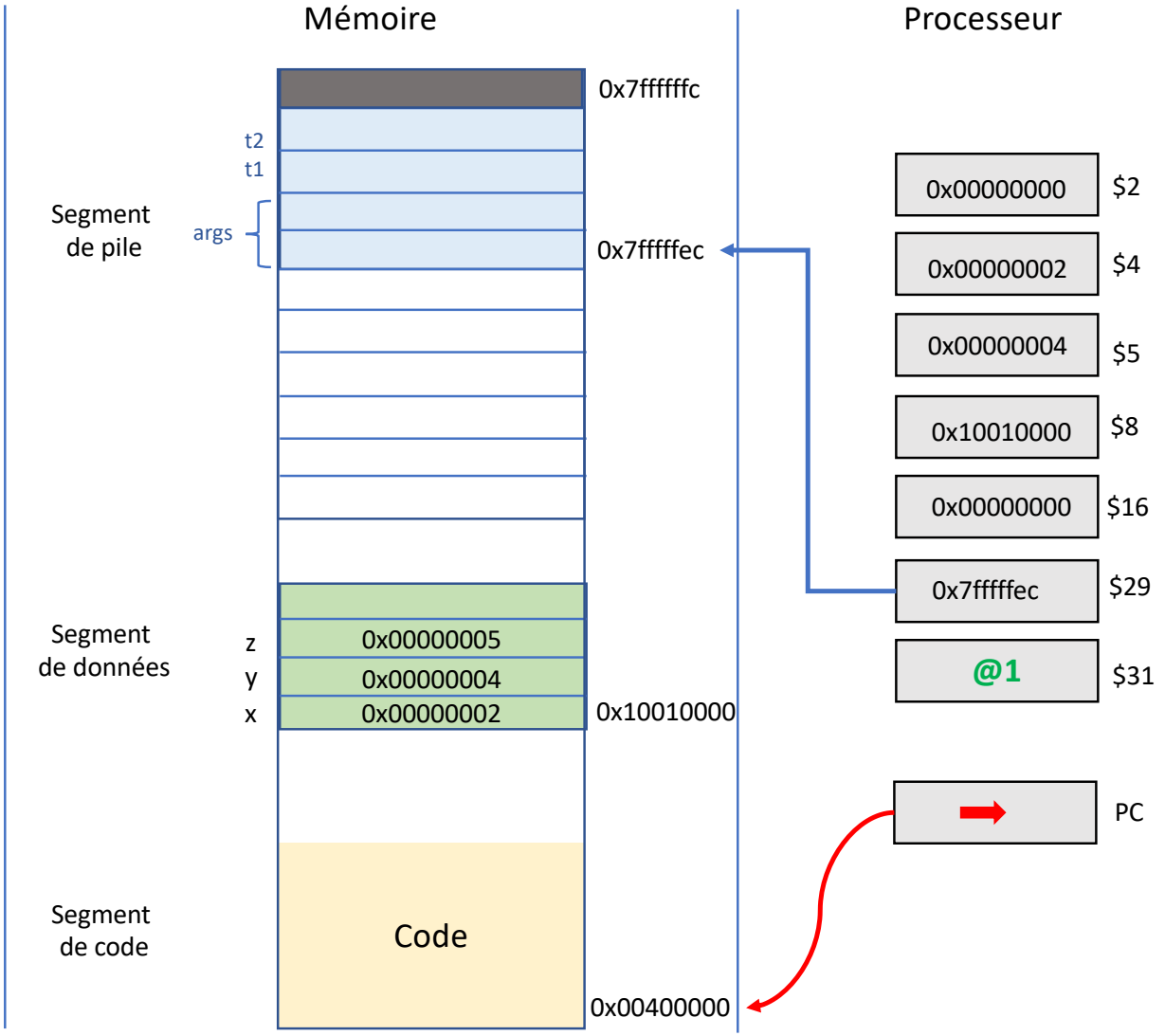


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw     $31, 4($29)
 sw     $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq    $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw     $31, 4($29)
 lw     $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

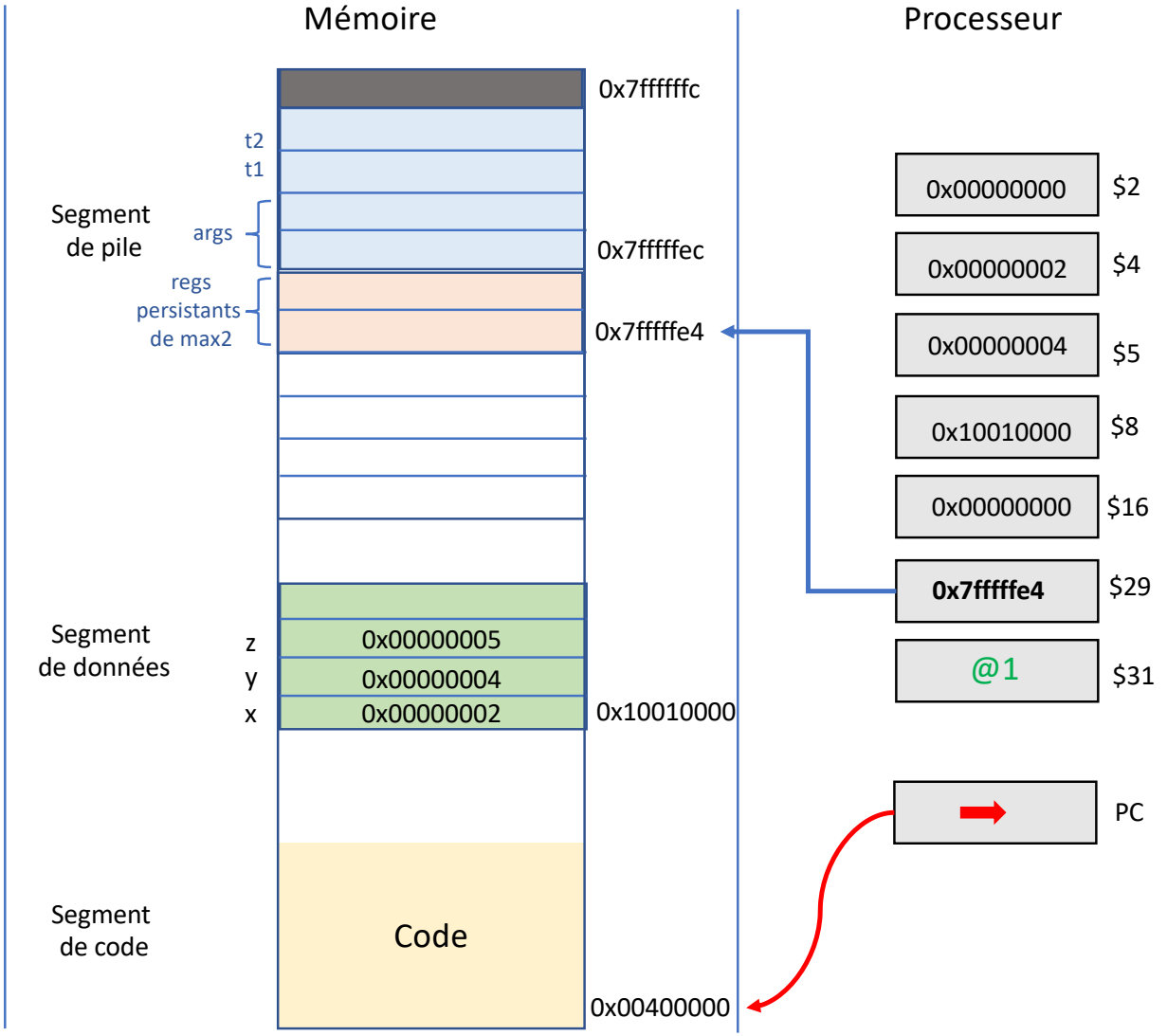


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

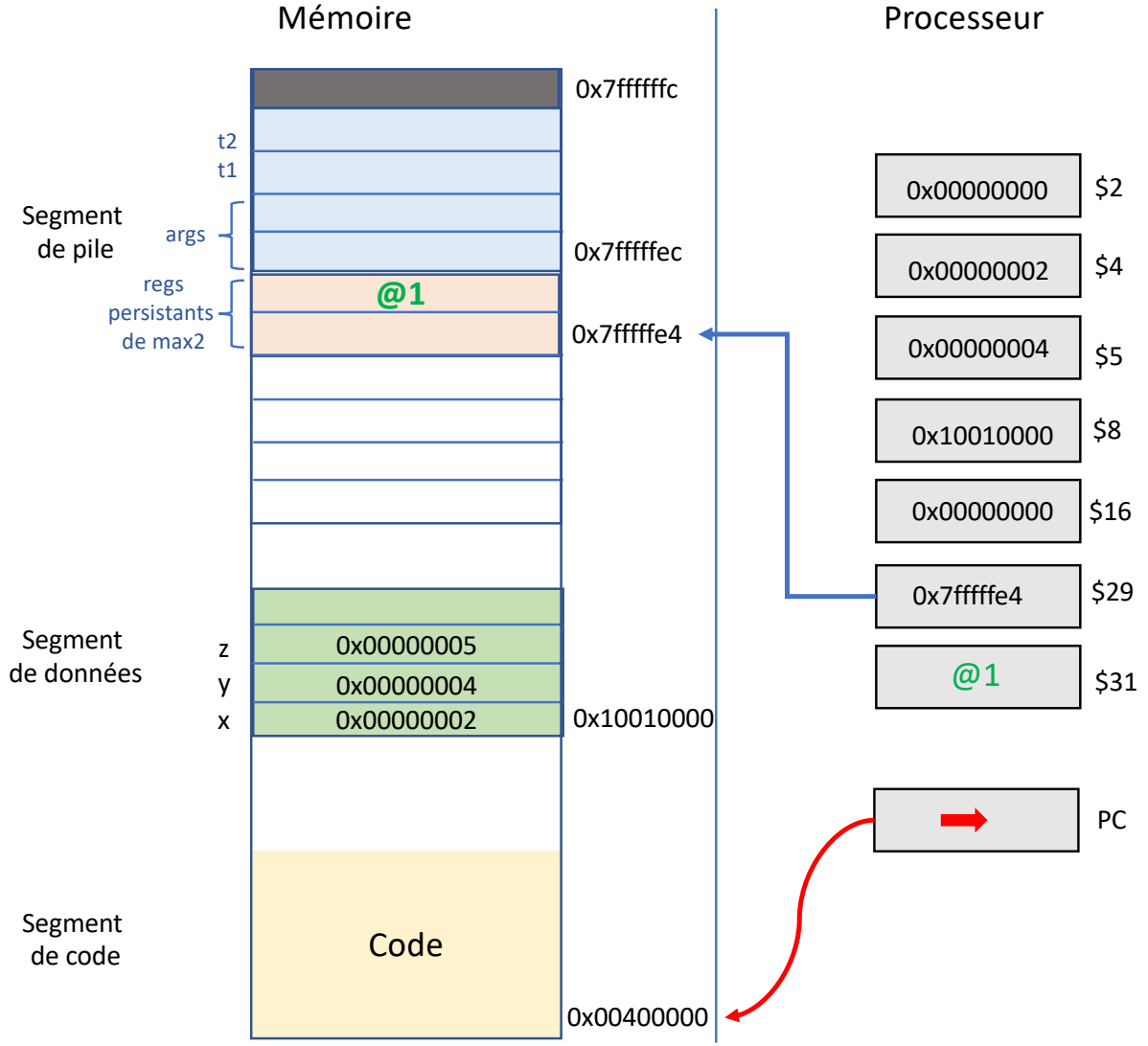


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

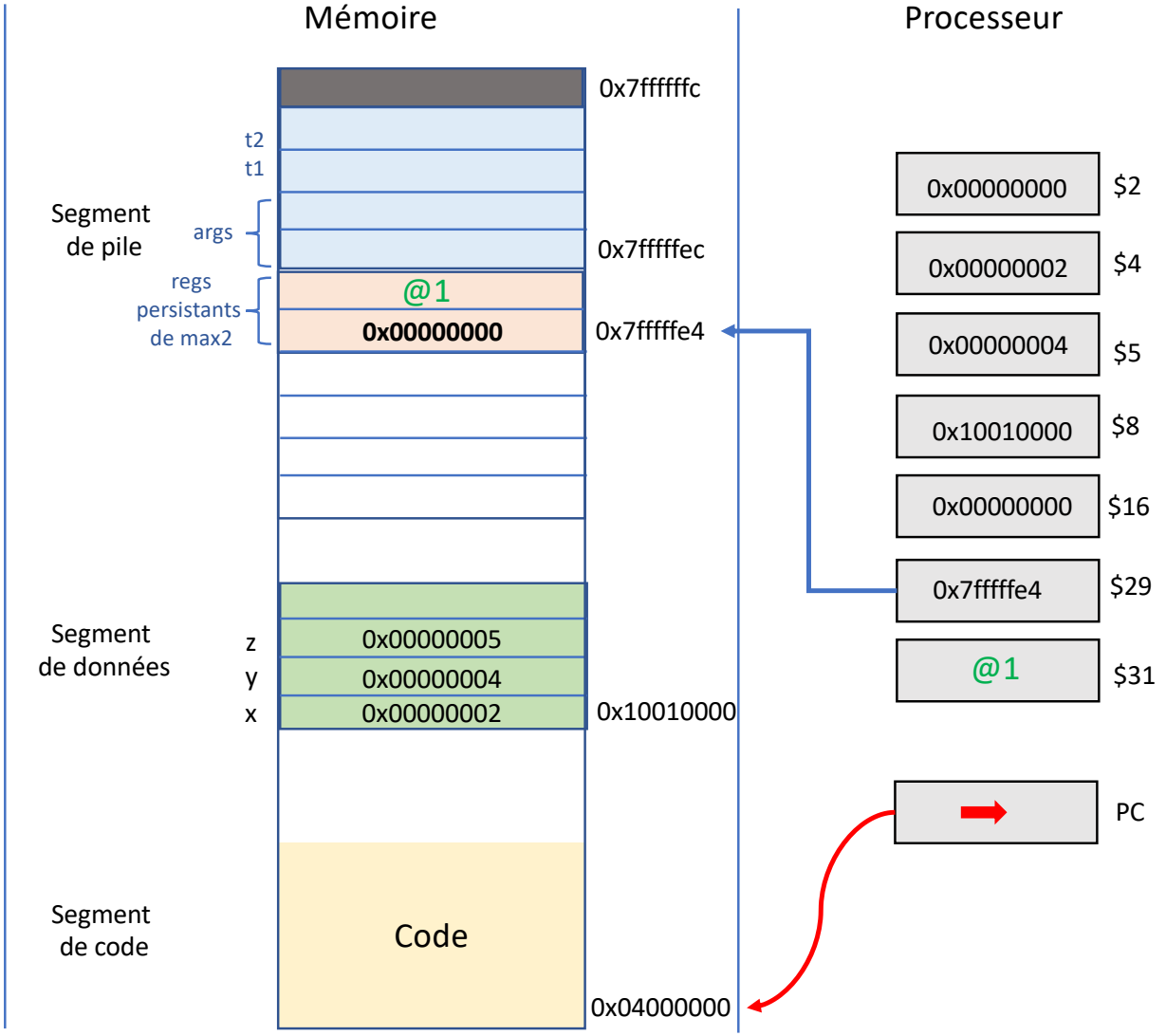


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

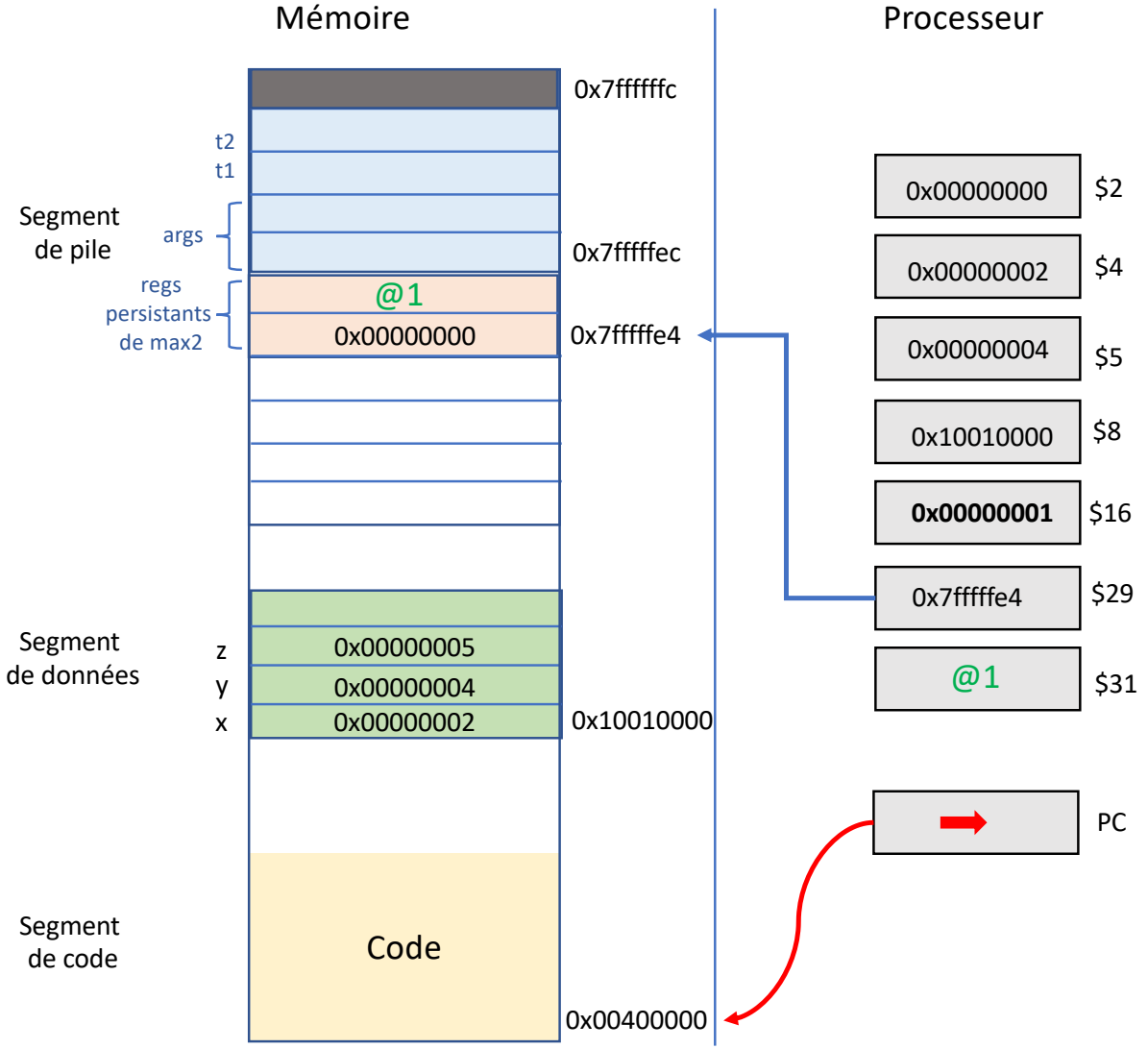


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

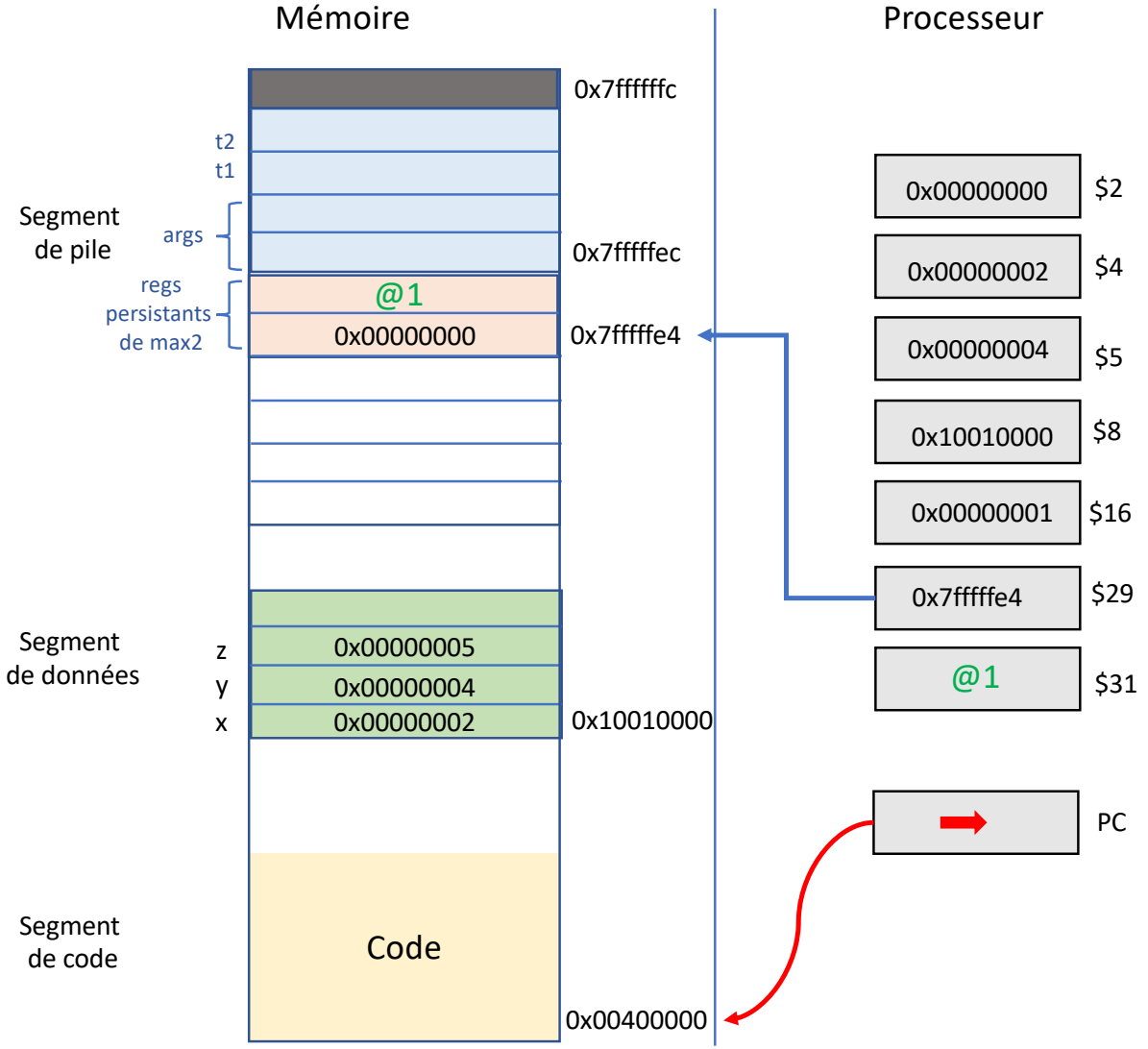


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

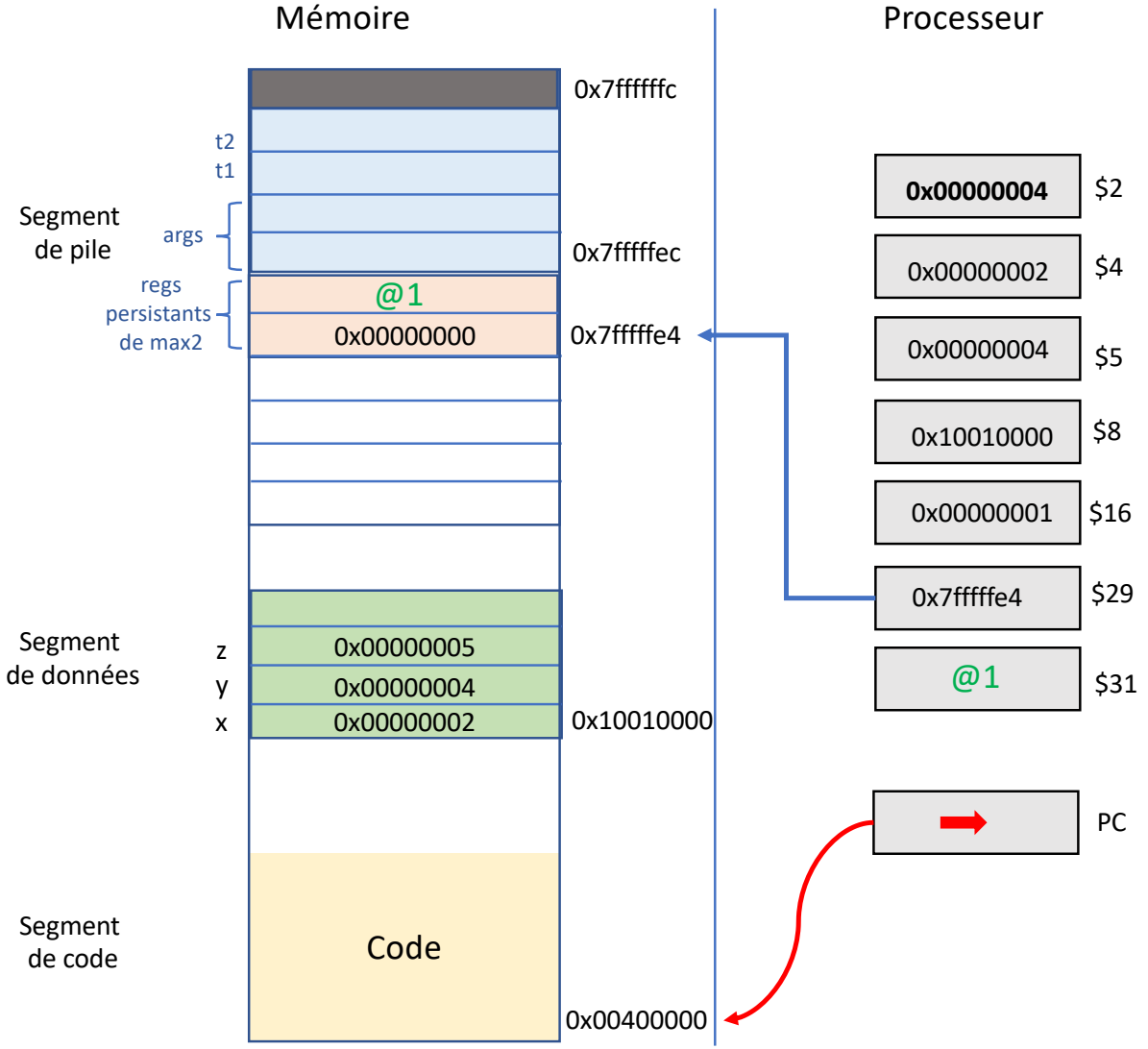


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

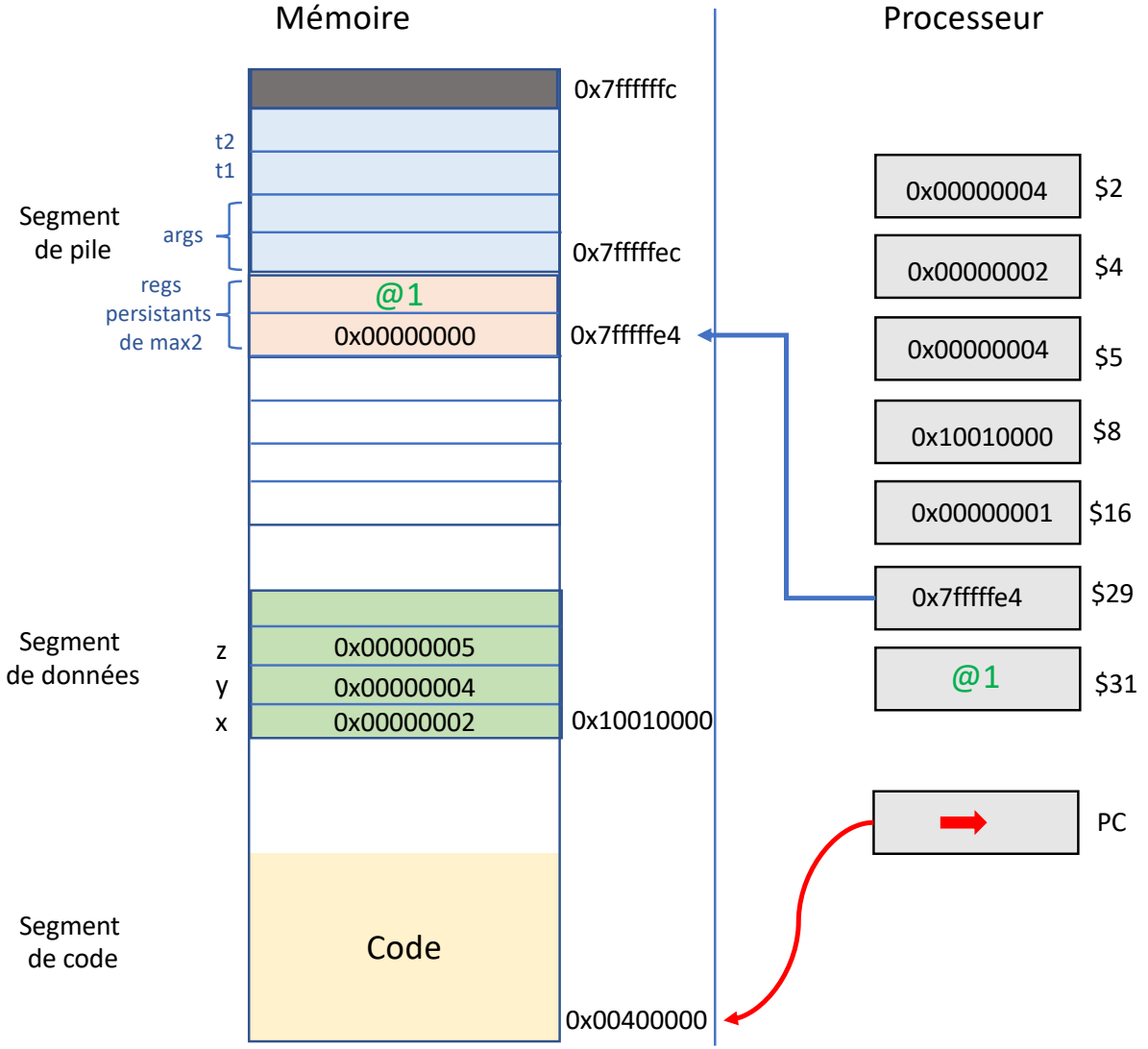


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw     $2, 8($29)         # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori    $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8         # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw     $31, 4($29)
 sw     $16, 0($29)
 slt    $16, $4, $5        #$5 vaut 1 si a < b
 beq    $16, $0, a_max
 ori    $2, $5, 0          # valeur de retour dans $2
 j      fin
a_max:
 ori    $2, $4, 0          # valeur de retour dans $2
fin:
 lw     $31, 4($29)
 lw     $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

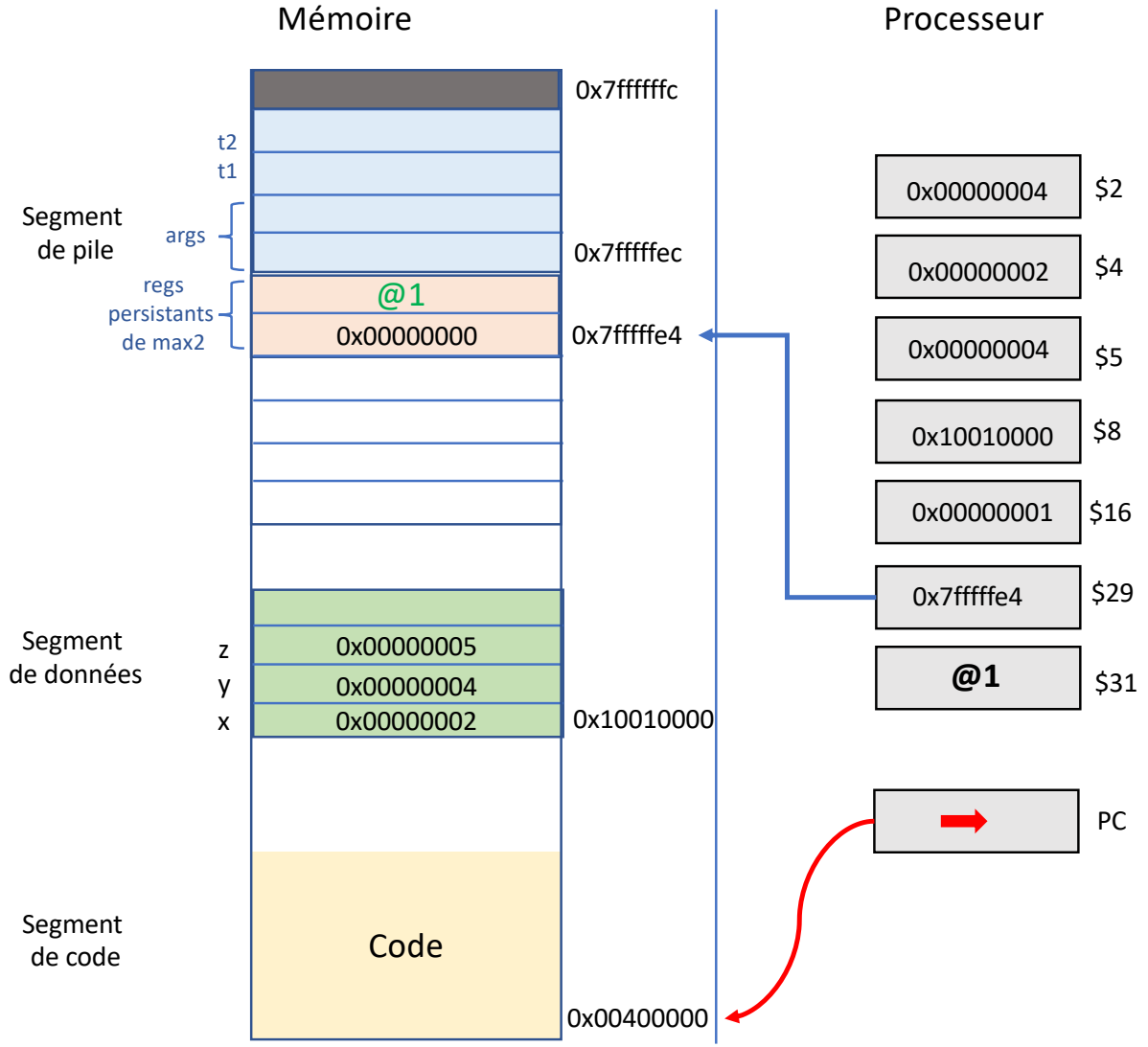


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

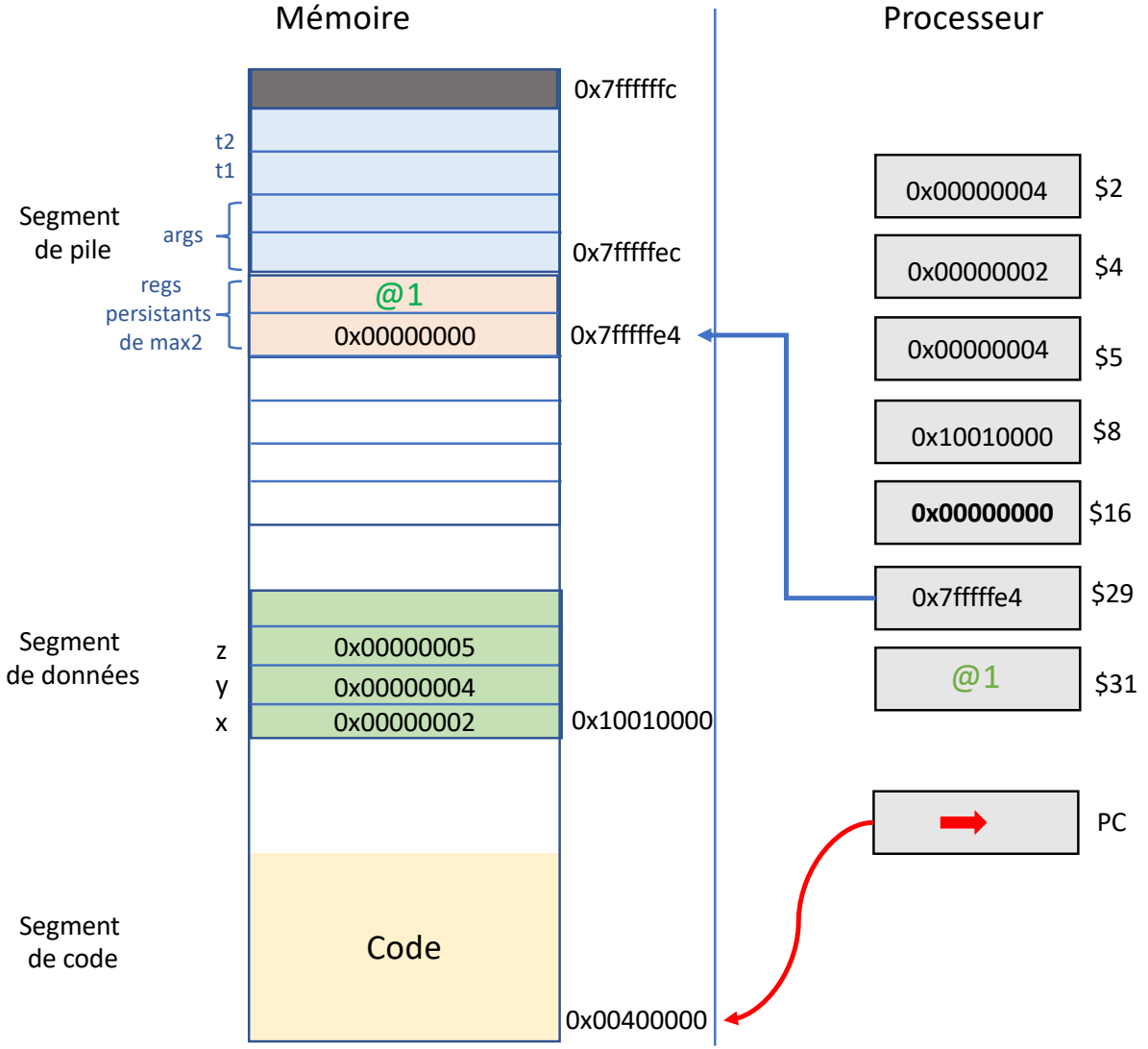


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

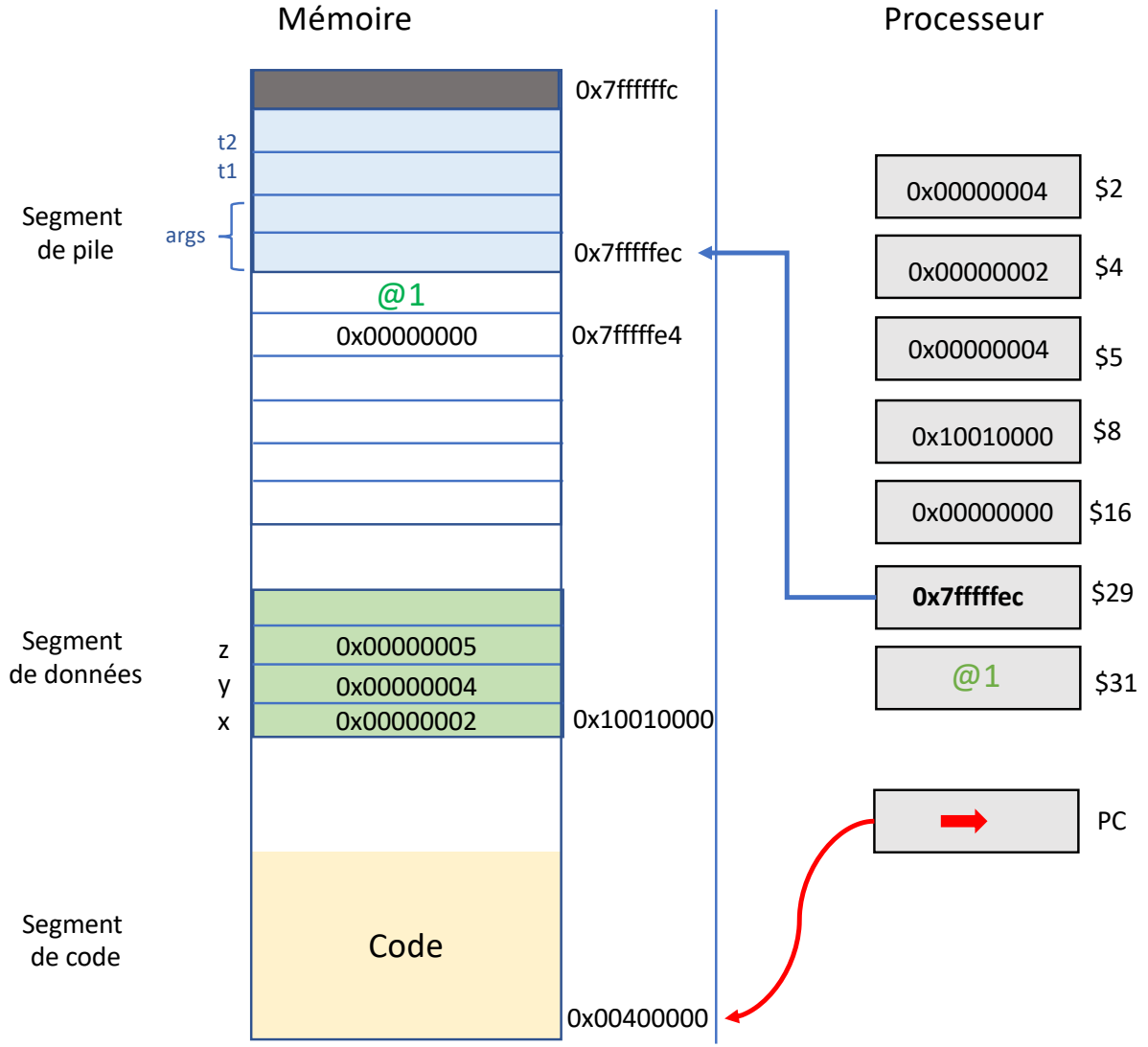


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw     $2, 8($29)         # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori    $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8         # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw     $31, 4($29)
 sw     $16, 0($29)
 slt    $16, $4, $5        #$5 vaut 1 si a < b
 beq    $16, $0, a_max
 ori    $2, $5, 0          # valeur de retour dans $2
 j      fin
a_max:
 ori    $2, $4, 0          # valeur de retour dans $2
fin:
 lw     $31, 4($29)
 lw     $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

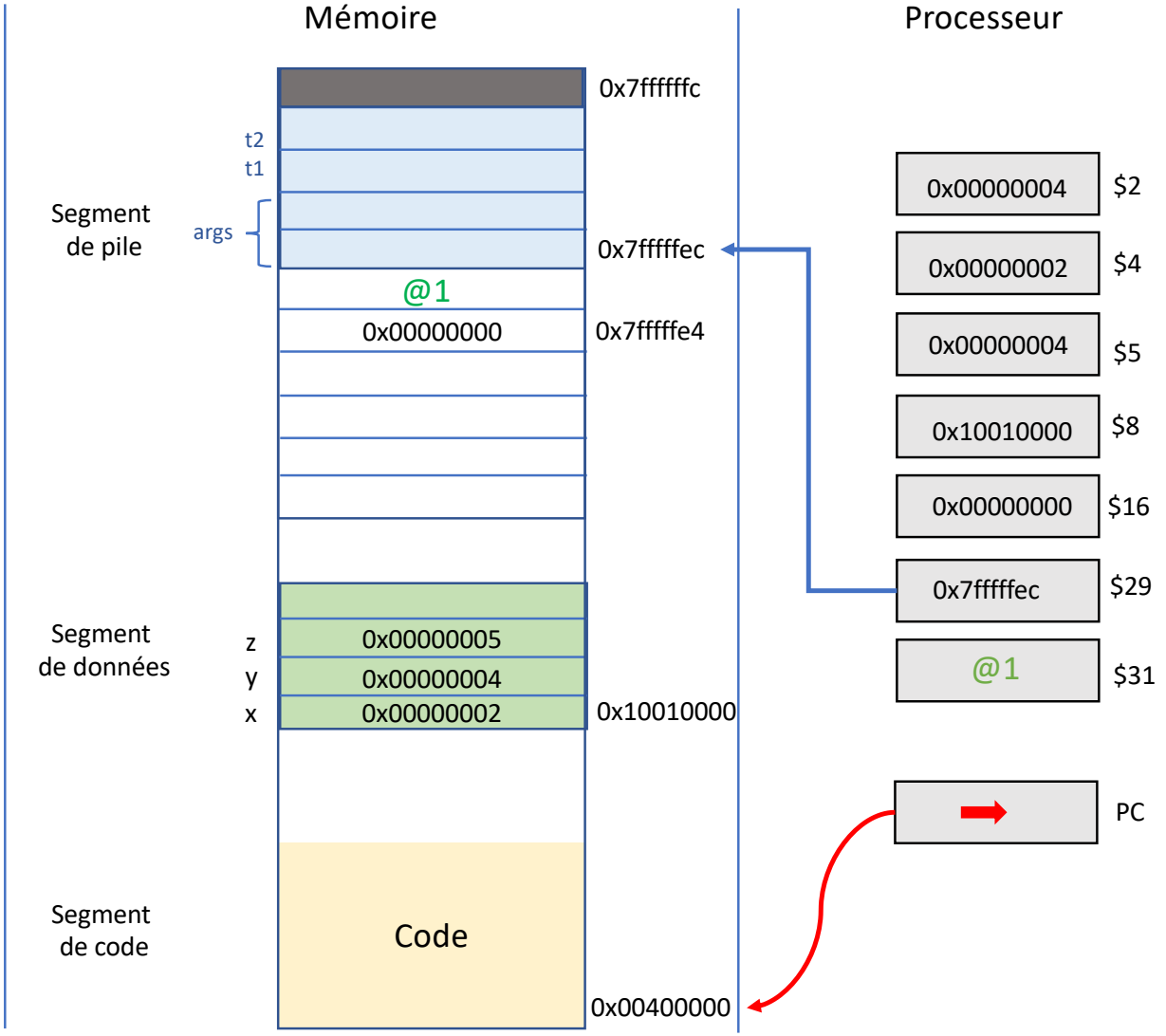


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

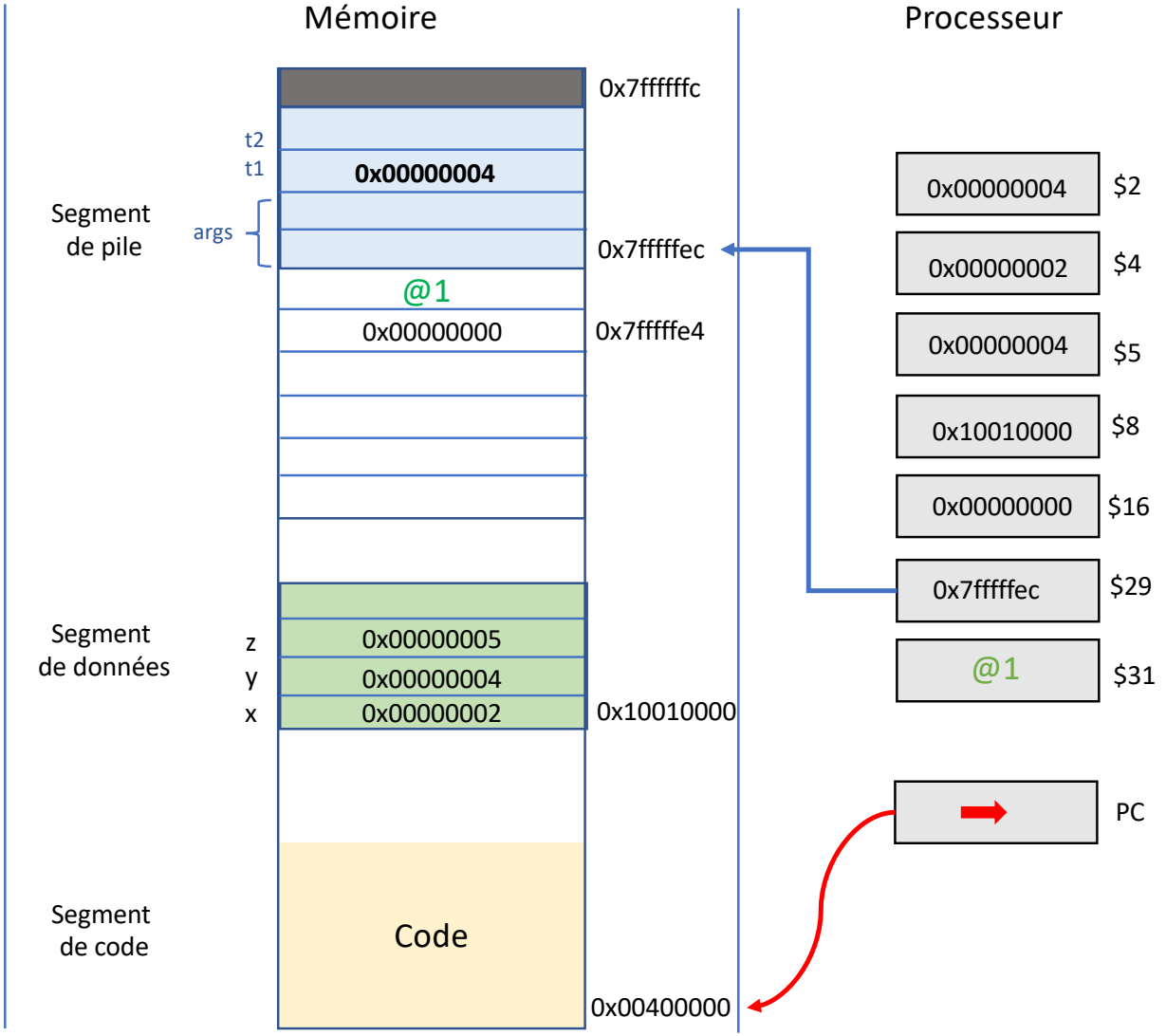


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

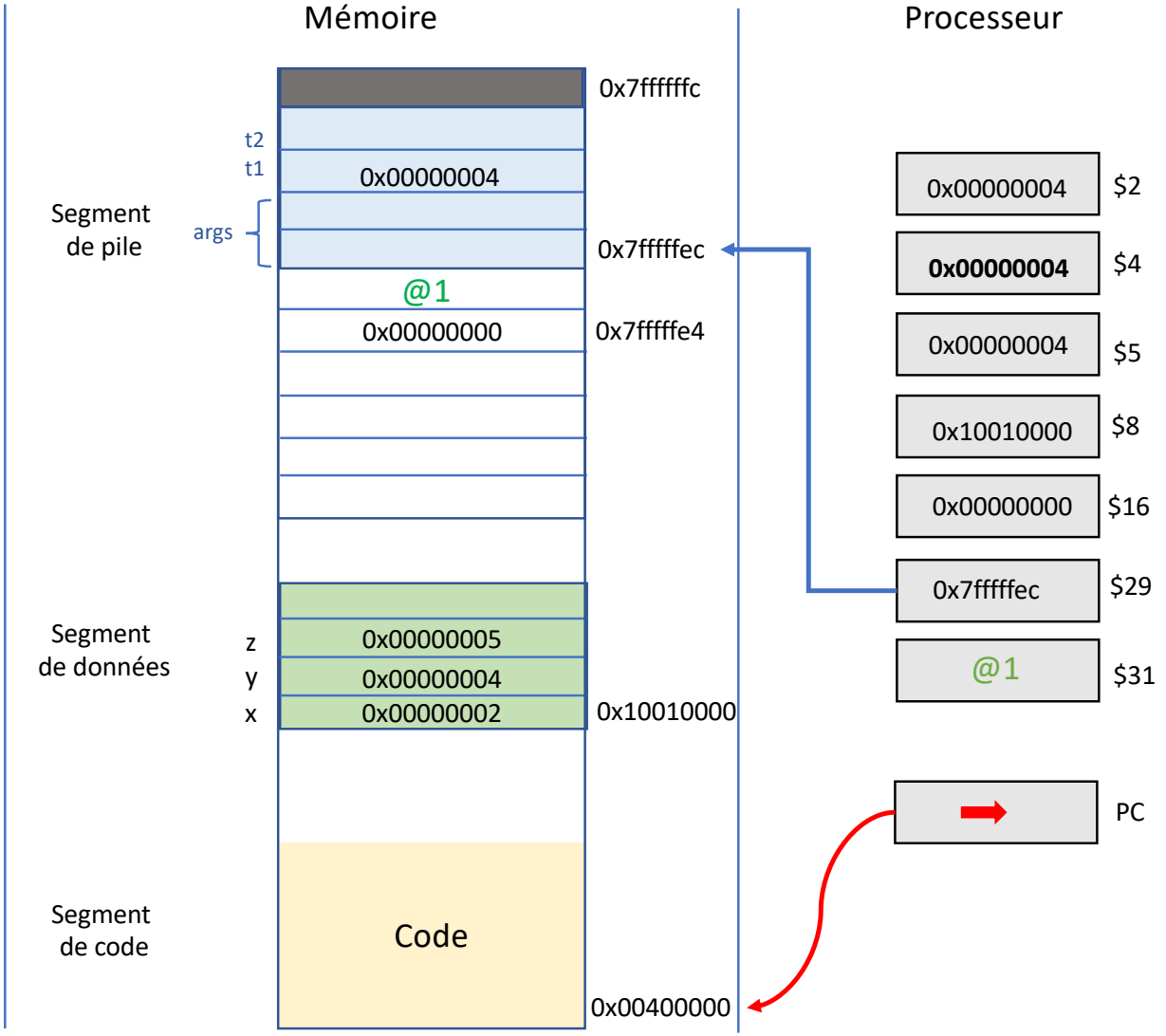


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

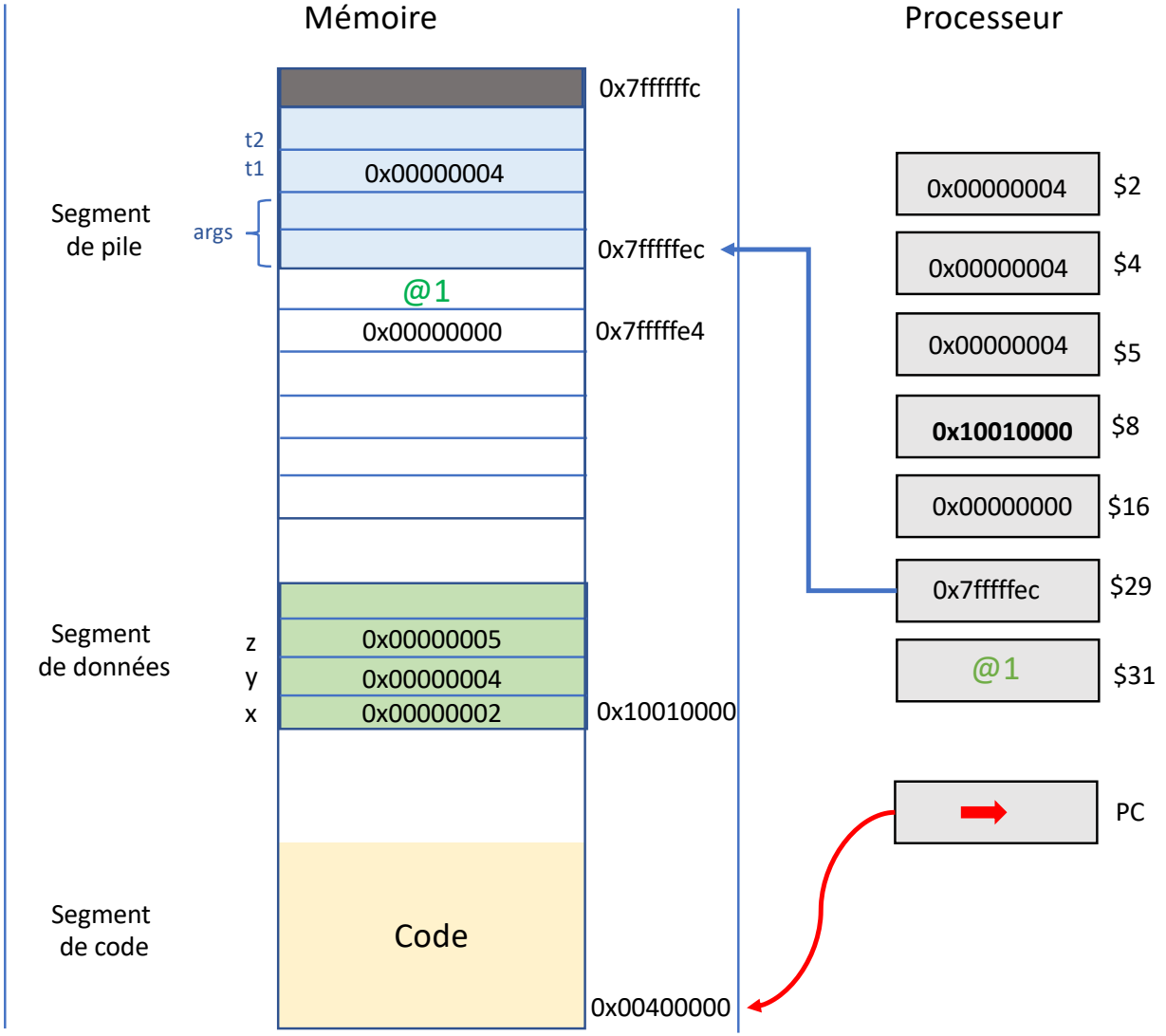


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

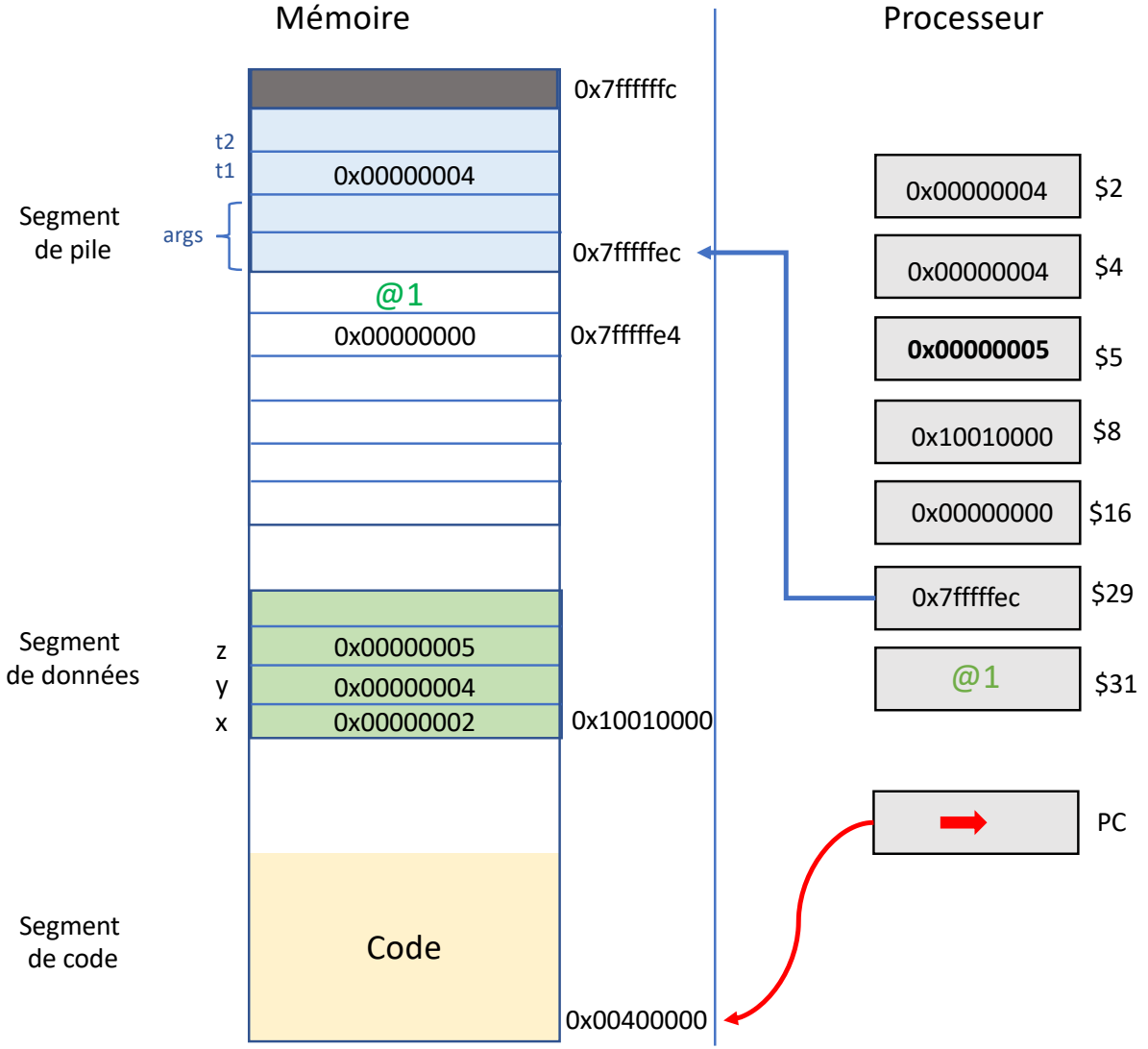


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

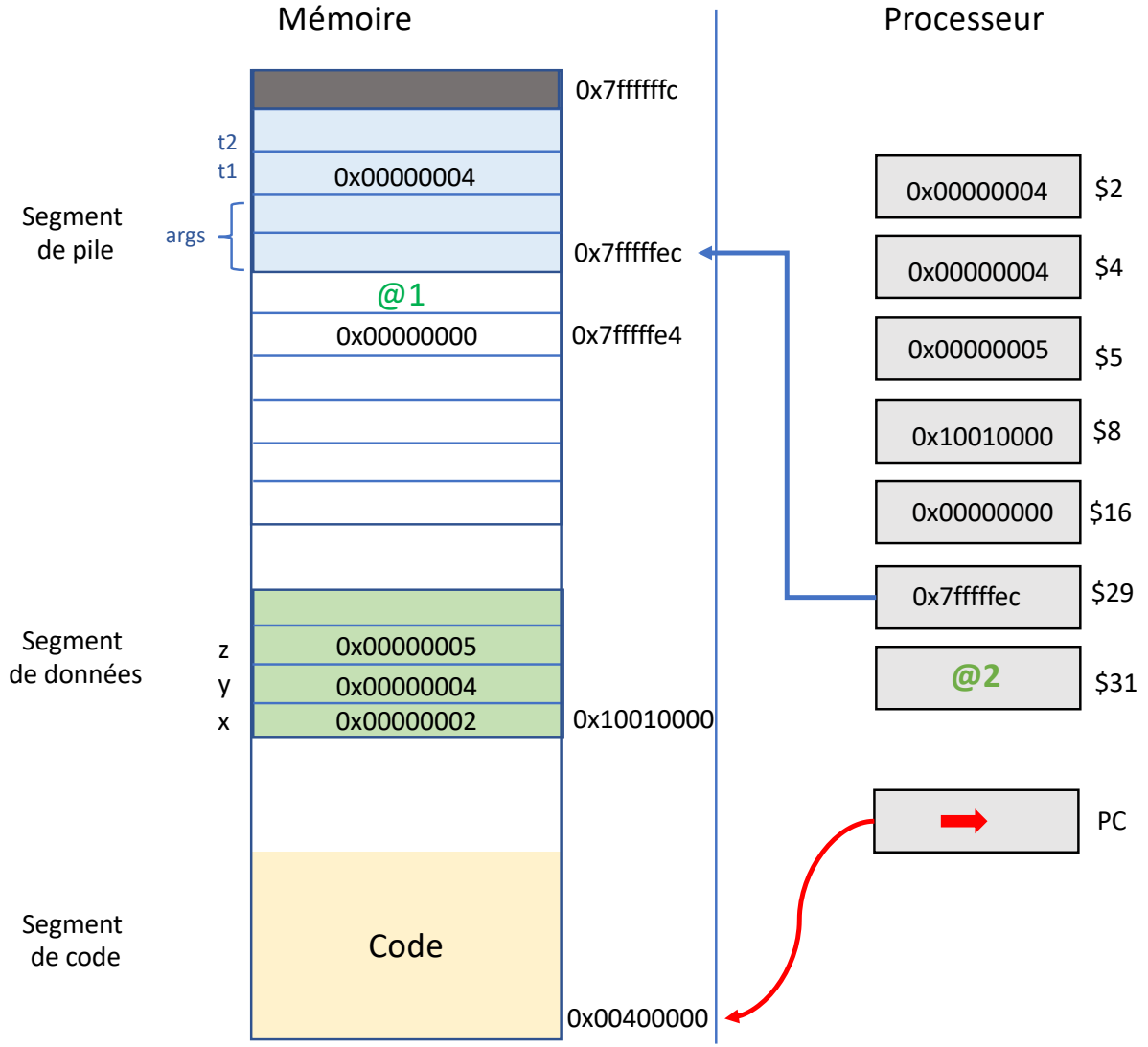


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw     $31, 4($29)
 sw     $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq    $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw     $31, 4($29)
 lw     $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

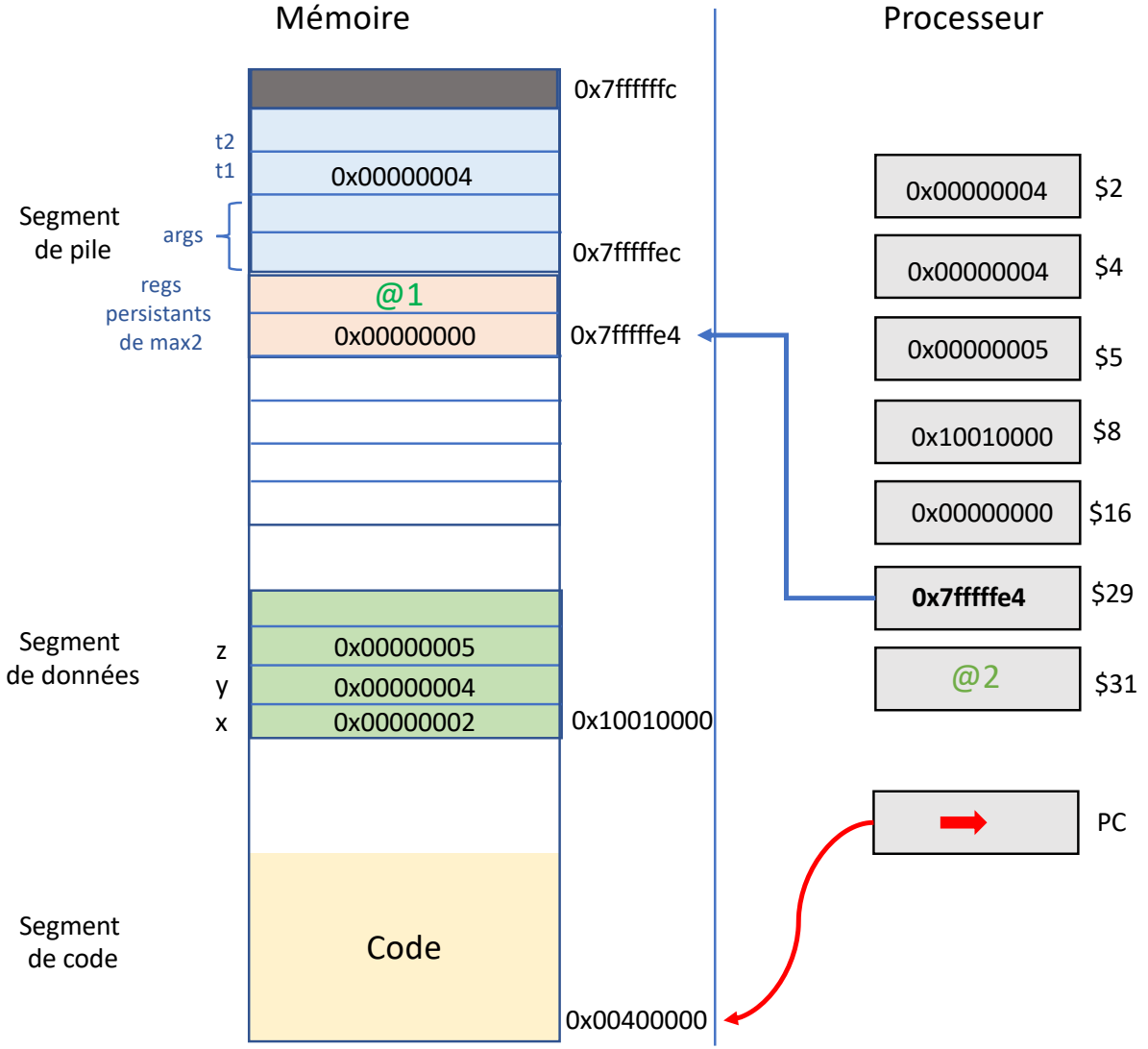


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw     $2, 8($29)         # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori    $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8         # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw     $31, 4($29)
 sw     $16, 0($29)
 slt    $16, $4, $5        #$5 vaut 1 si a < b
 beq    $16, $0, a_max
 ori    $2, $5, 0          # valeur de retour dans $2
 j      fin
a_max:
 ori    $2, $4, 0          # valeur de retour dans $2
fin:
 lw     $31, 4($29)
 lw     $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

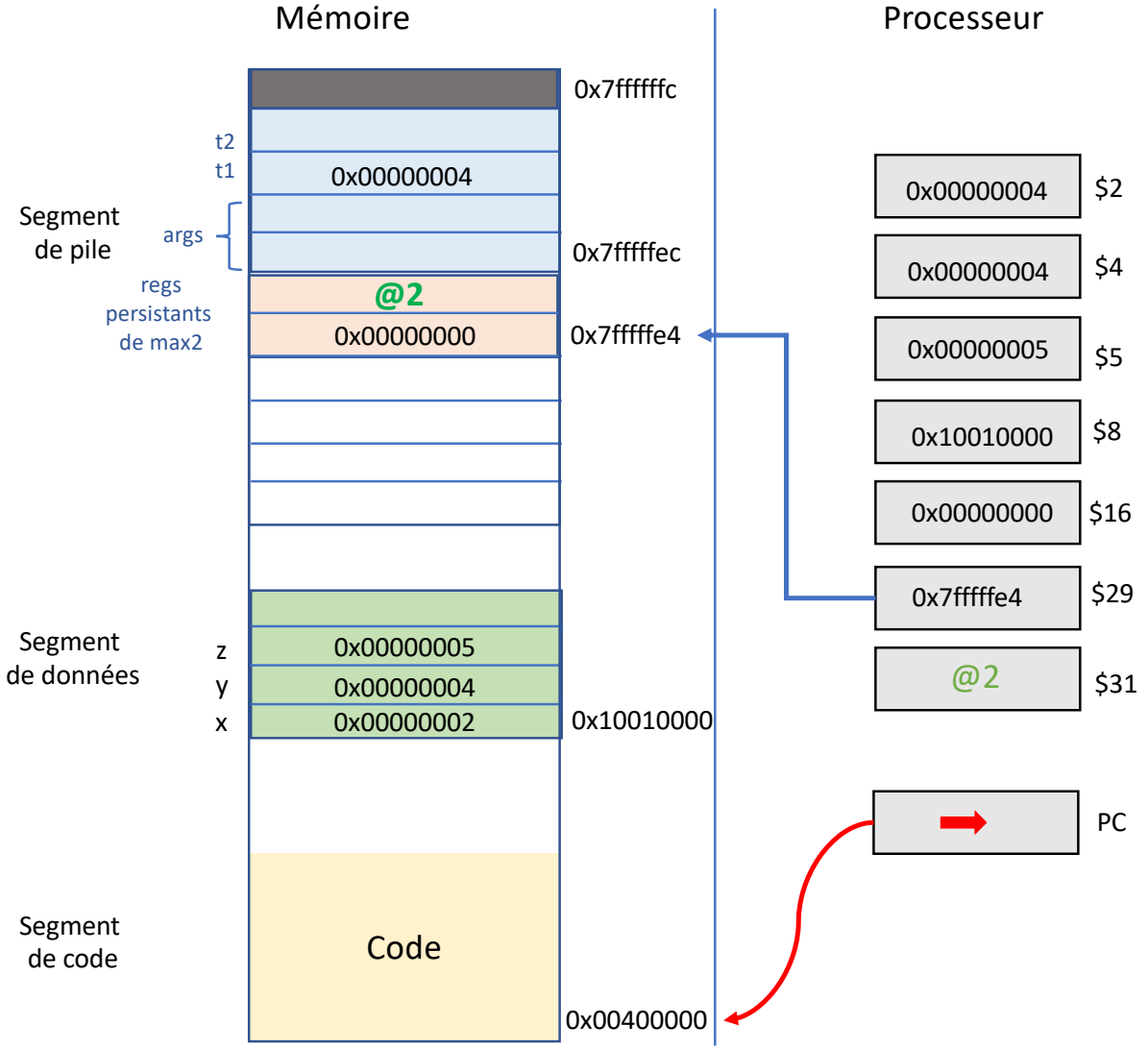


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw     $2, 8($29)         # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori    $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8         # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw     $31, 4($29)
 sw     $16, 0($29)
 slt    $16, $4, $5        #$5 vaut 1 si a < b
 beq    $16, $0, a_max
 ori    $2, $5, 0          # valeur de retour dans $2
 j      fin
a_max:
 ori    $2, $4, 0          # valeur de retour dans $2
fin:
 lw     $31, 4($29)
 lw     $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

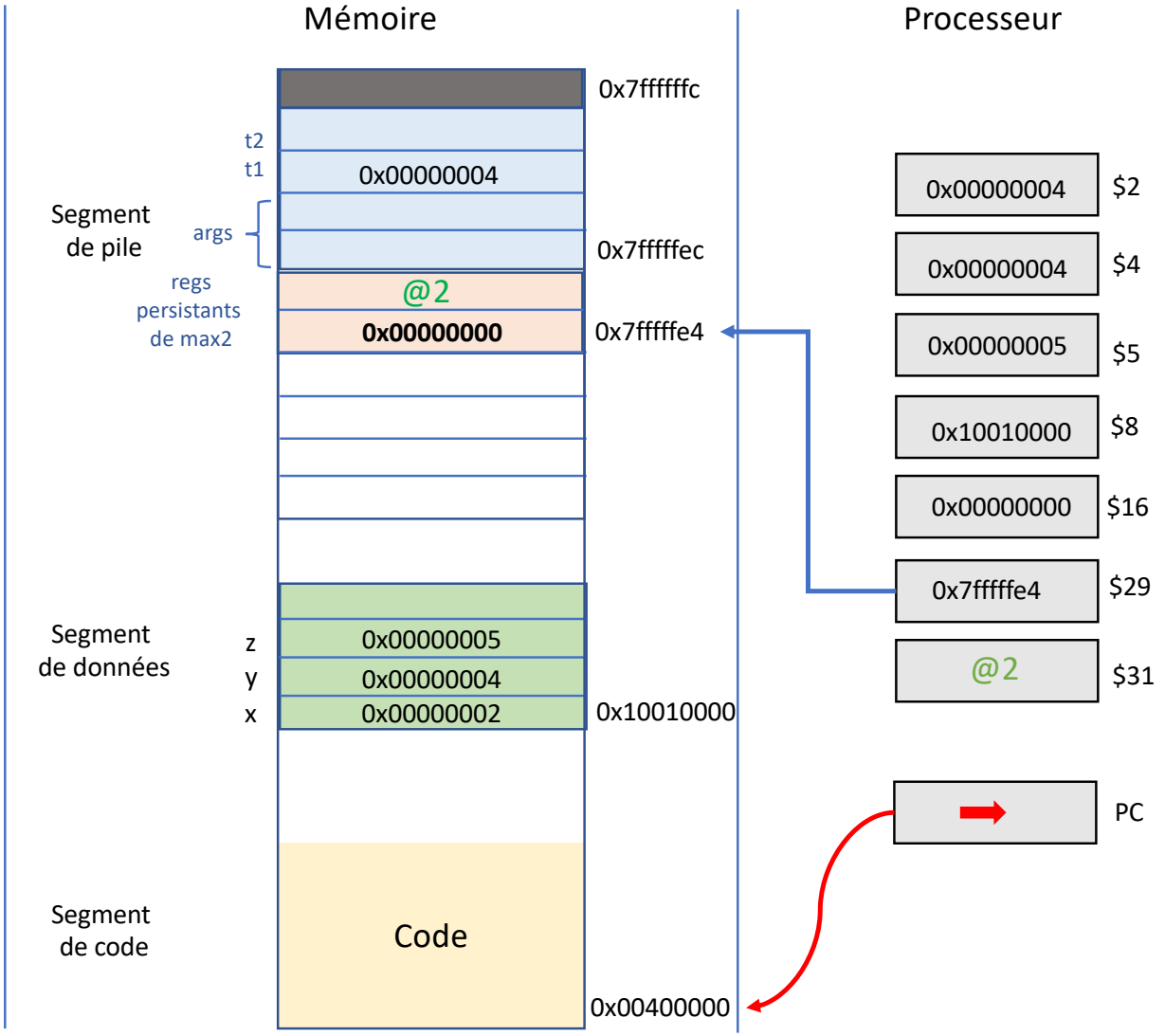


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

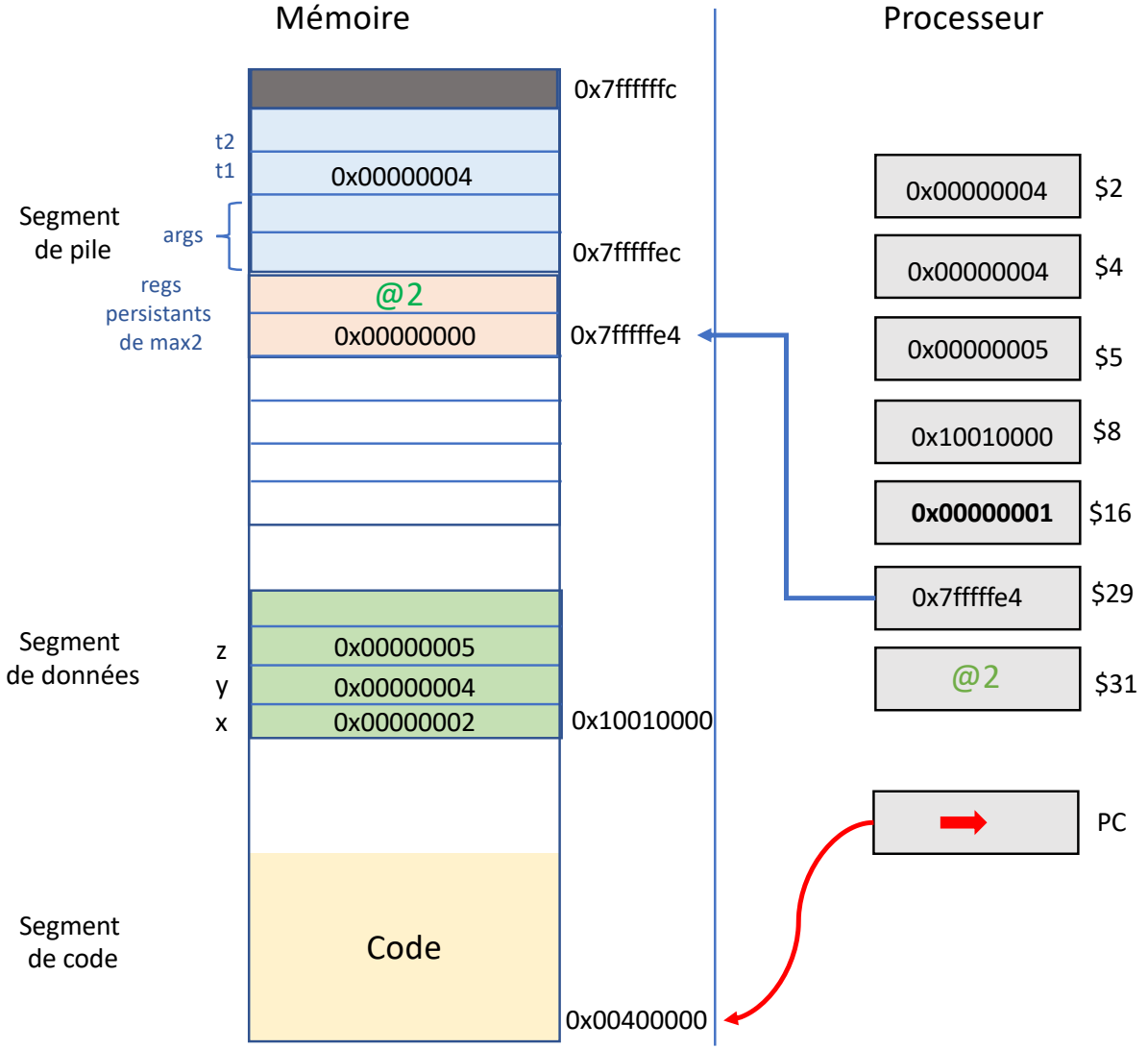


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

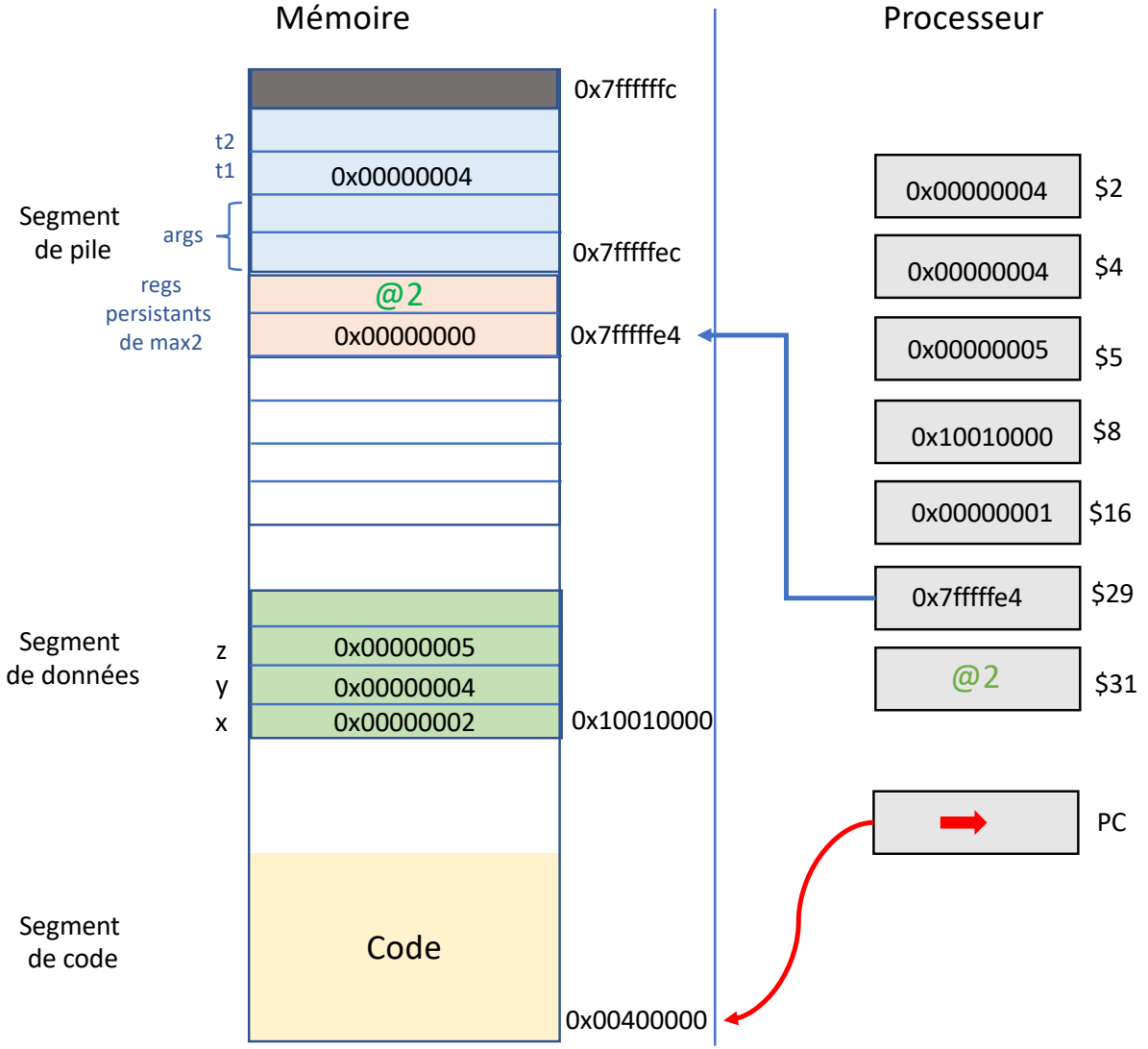


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

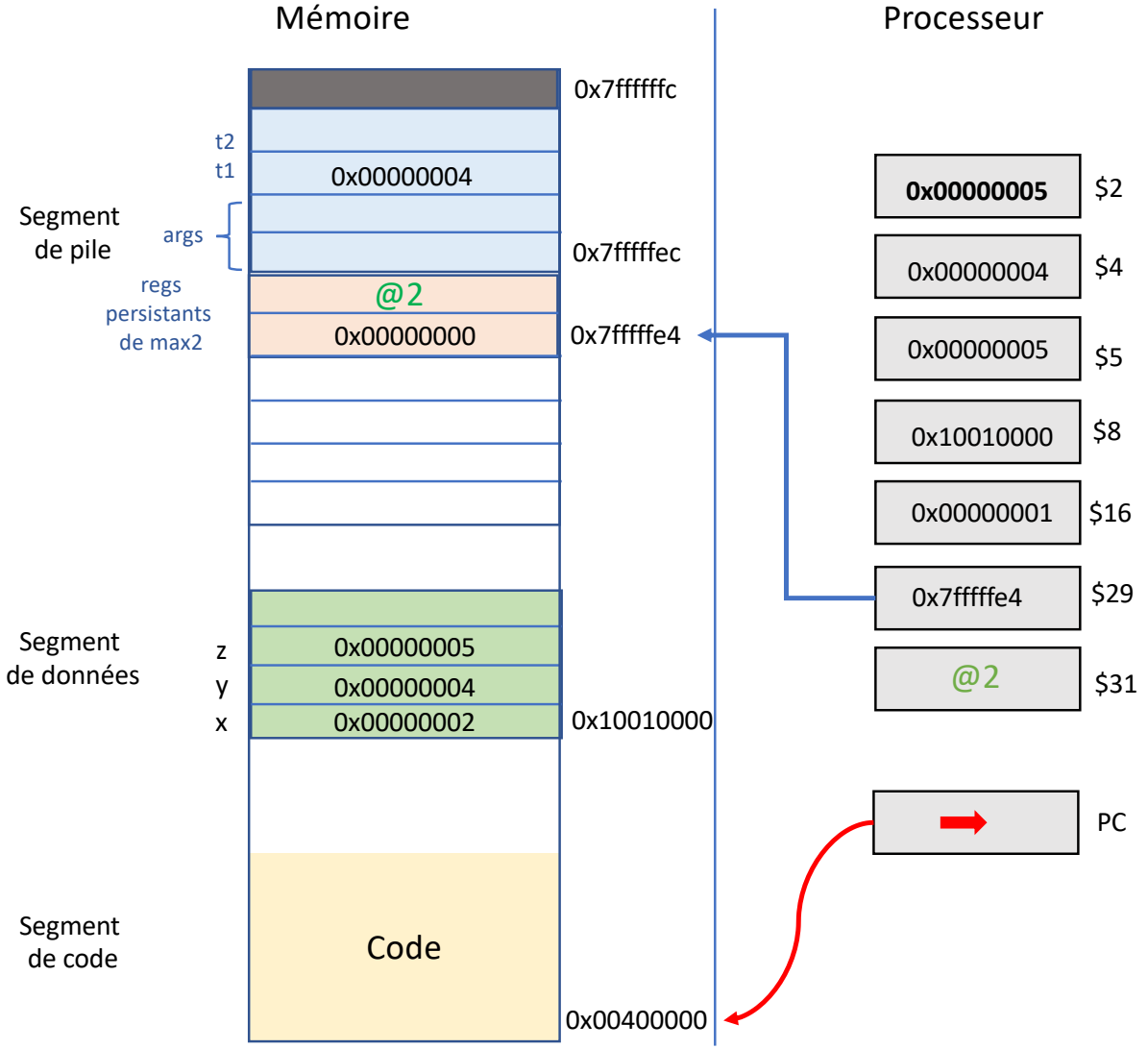


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

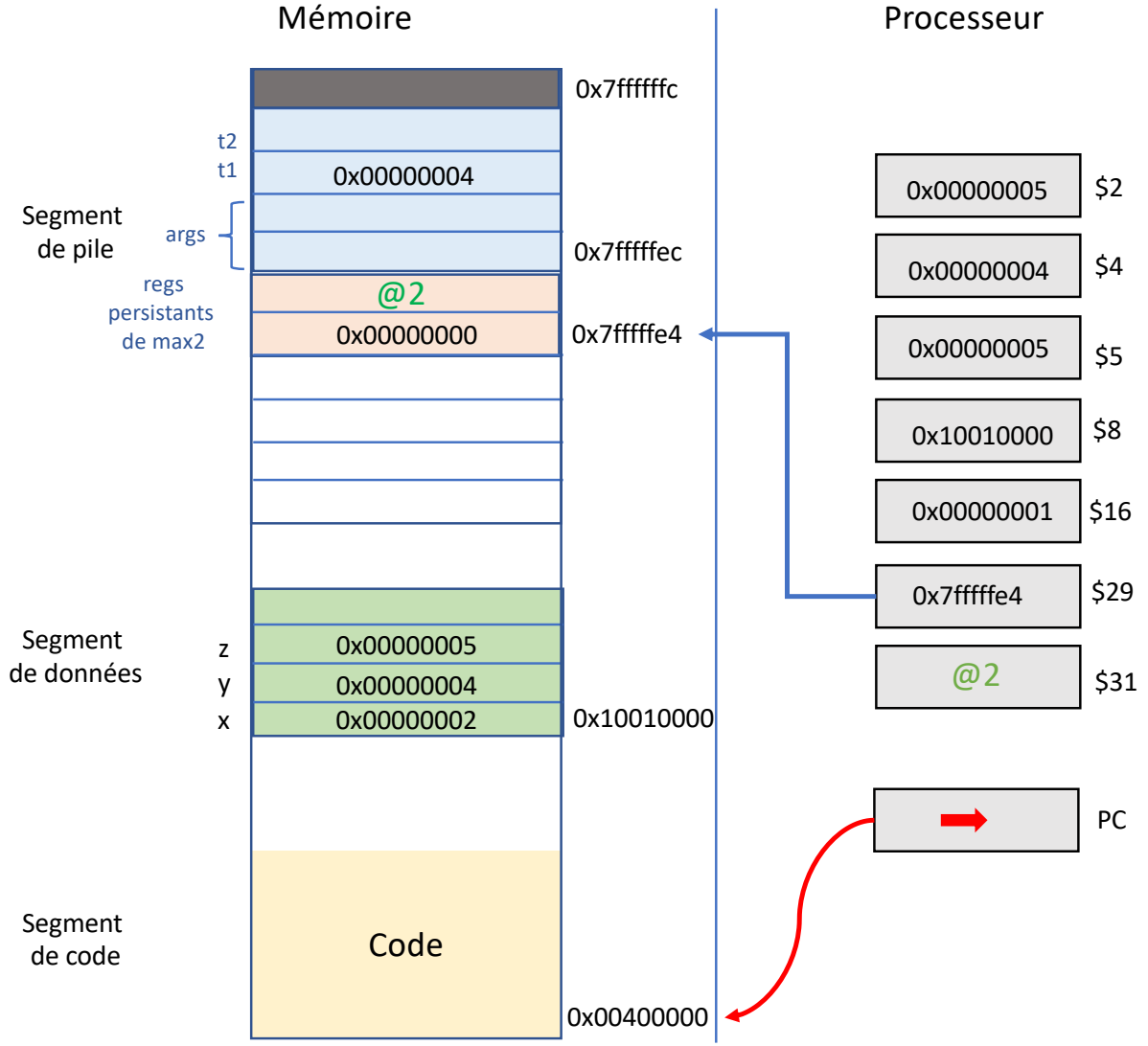


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

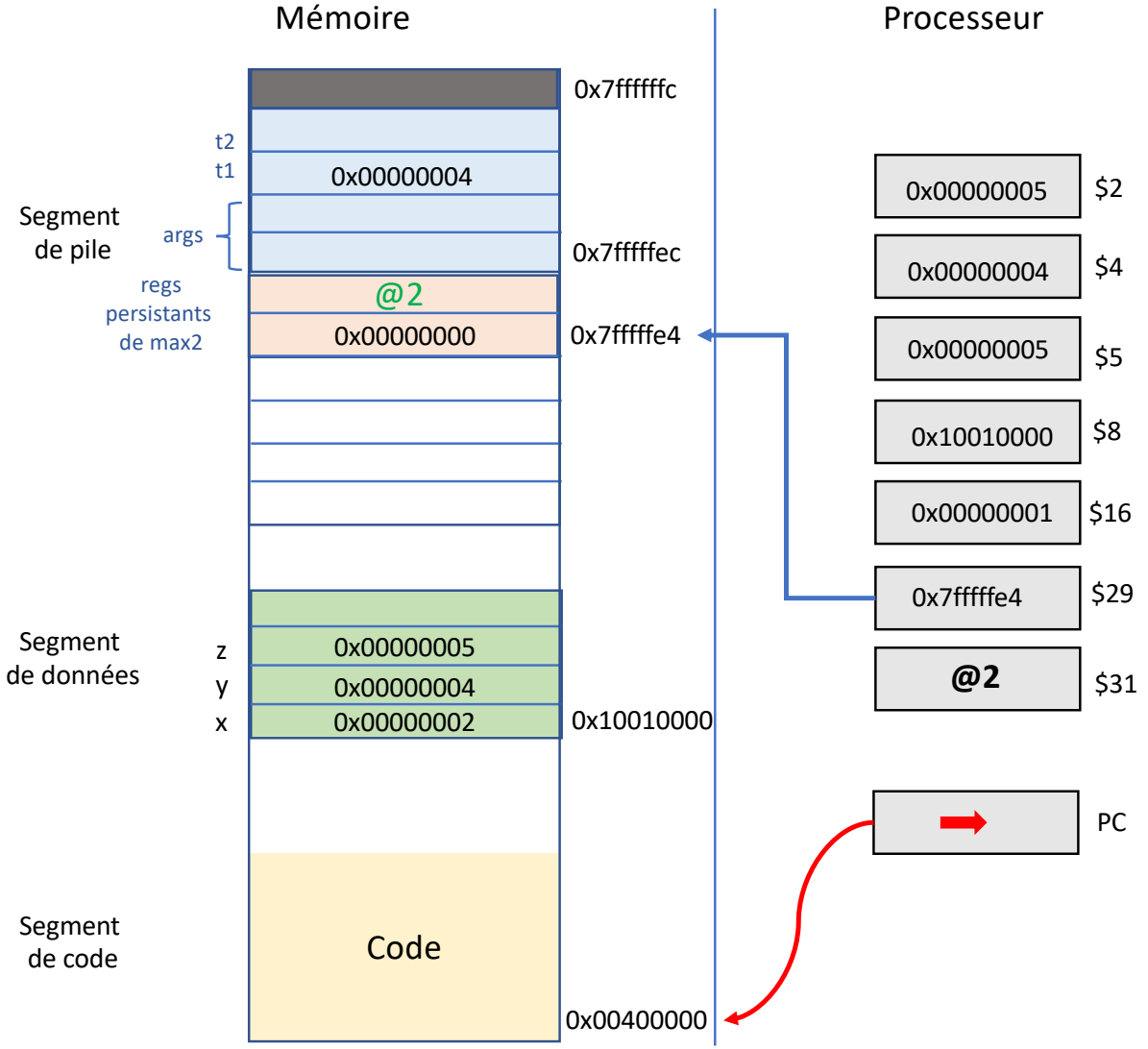


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

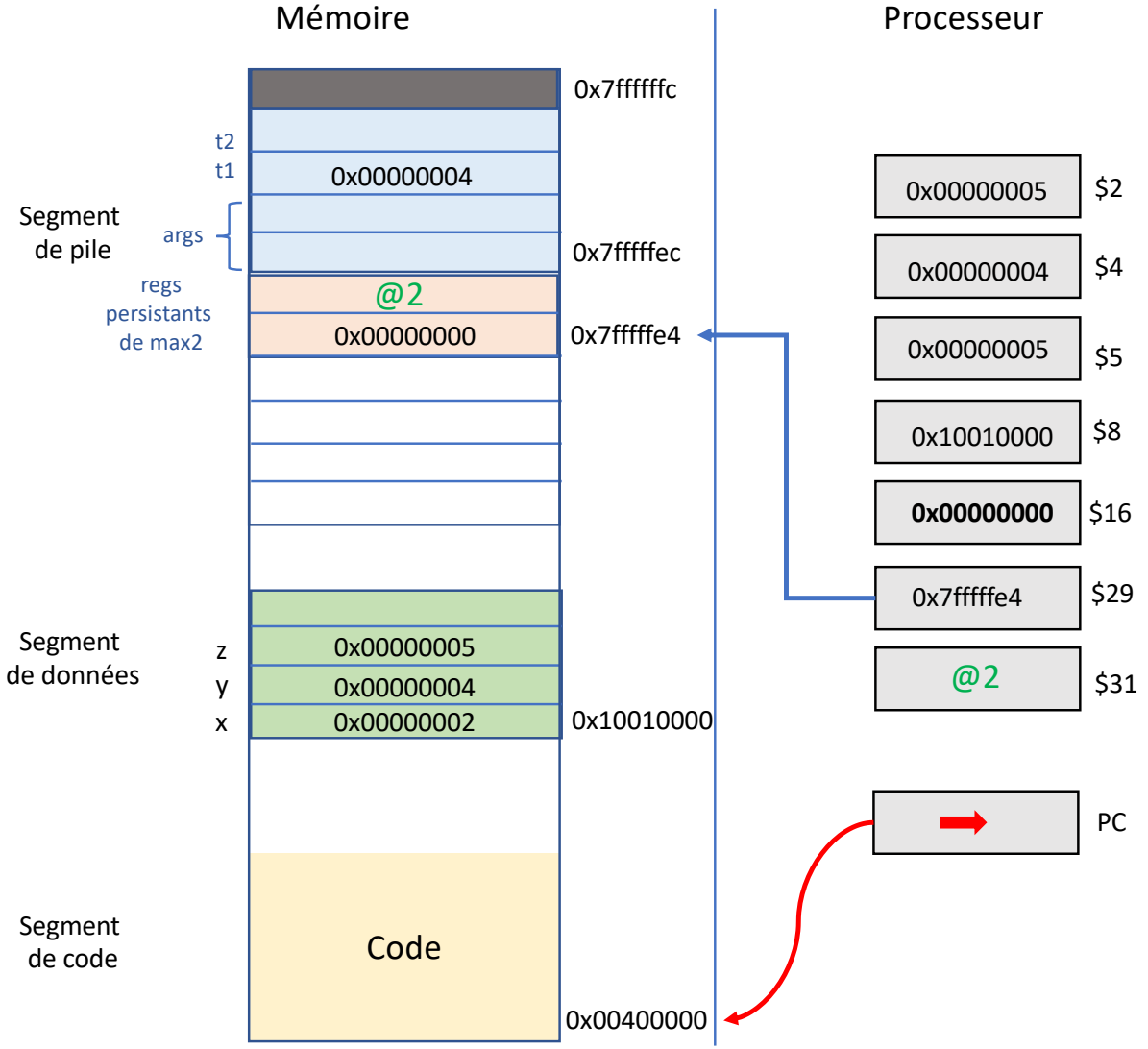


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

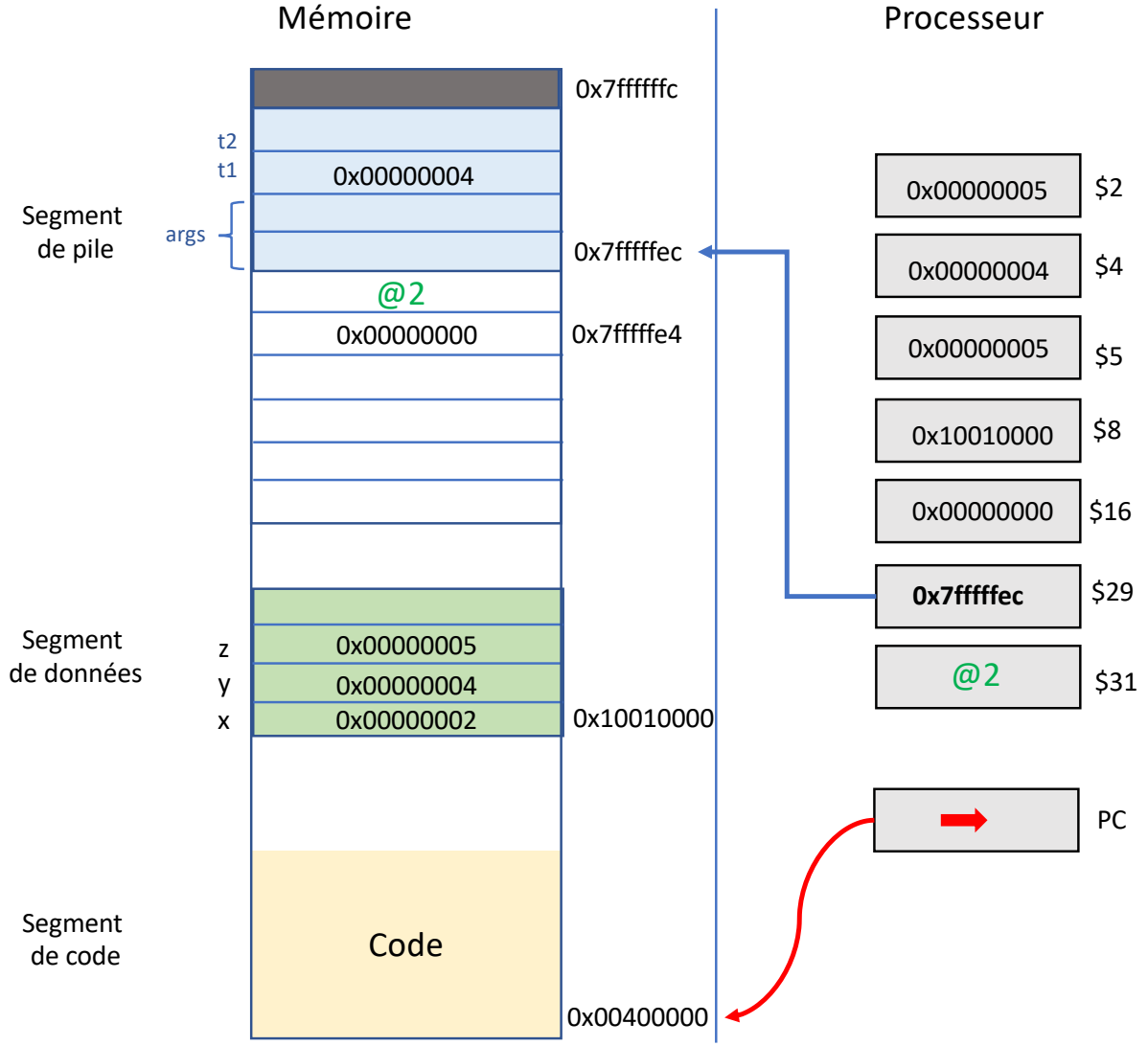


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

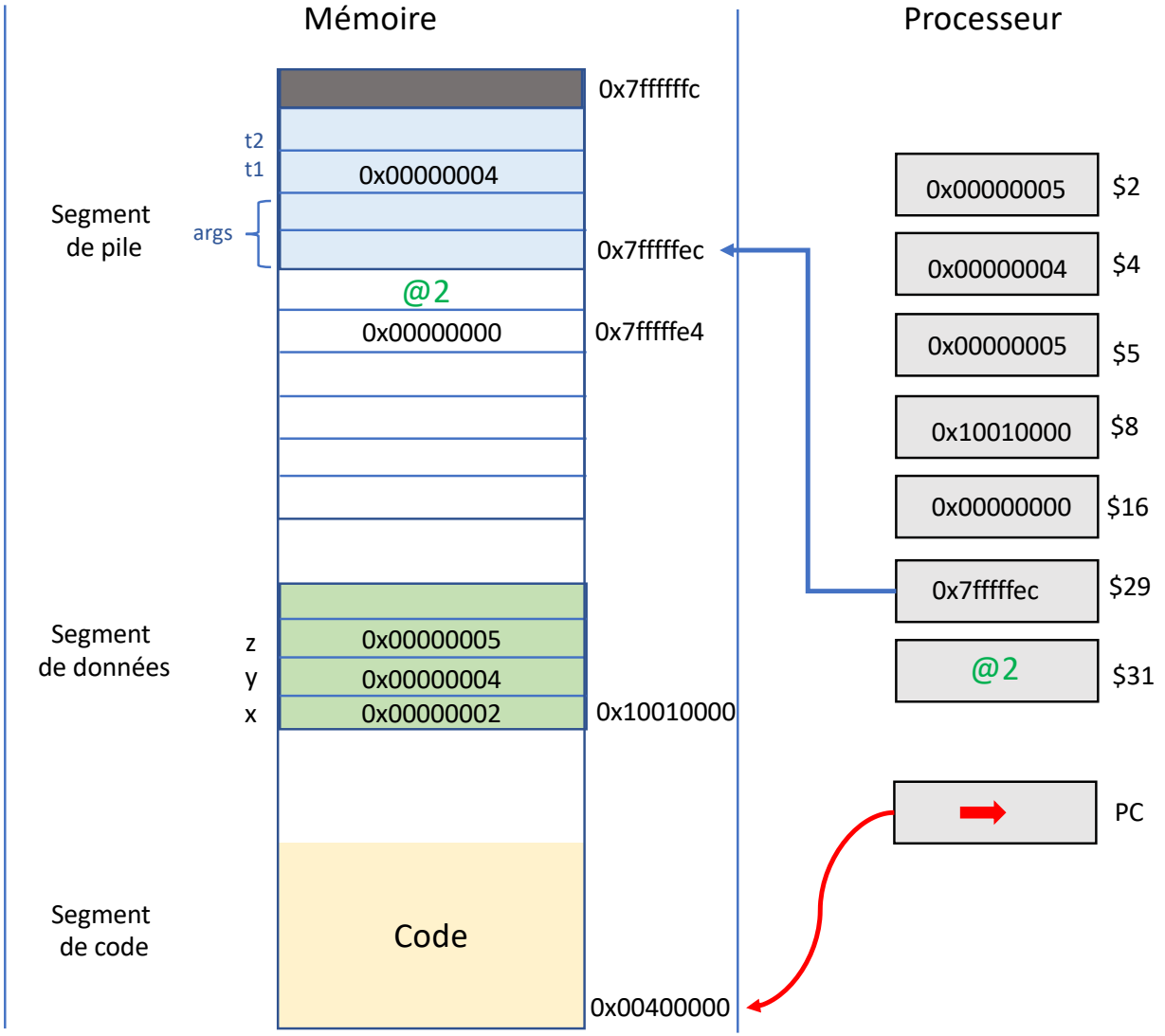


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

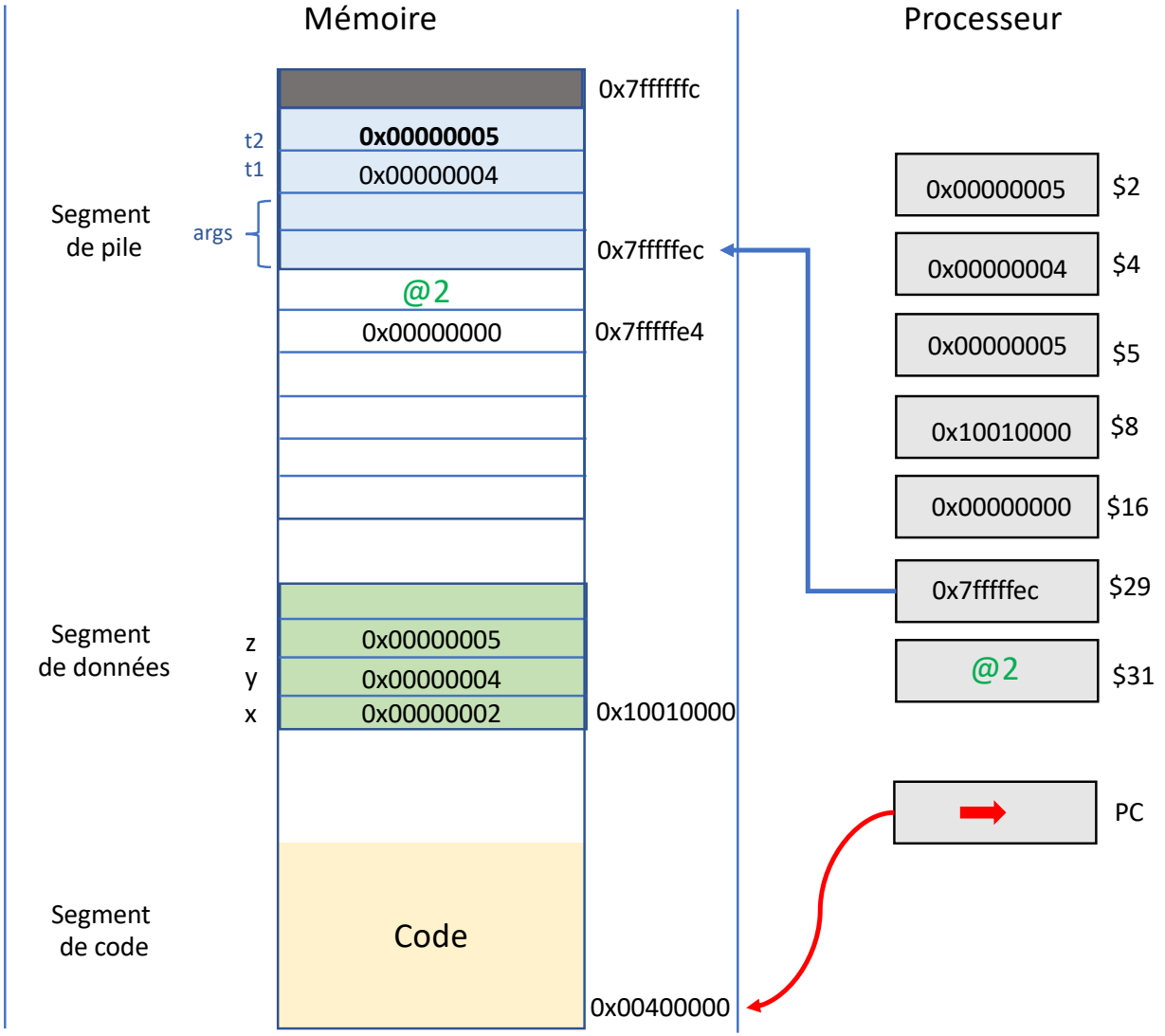


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

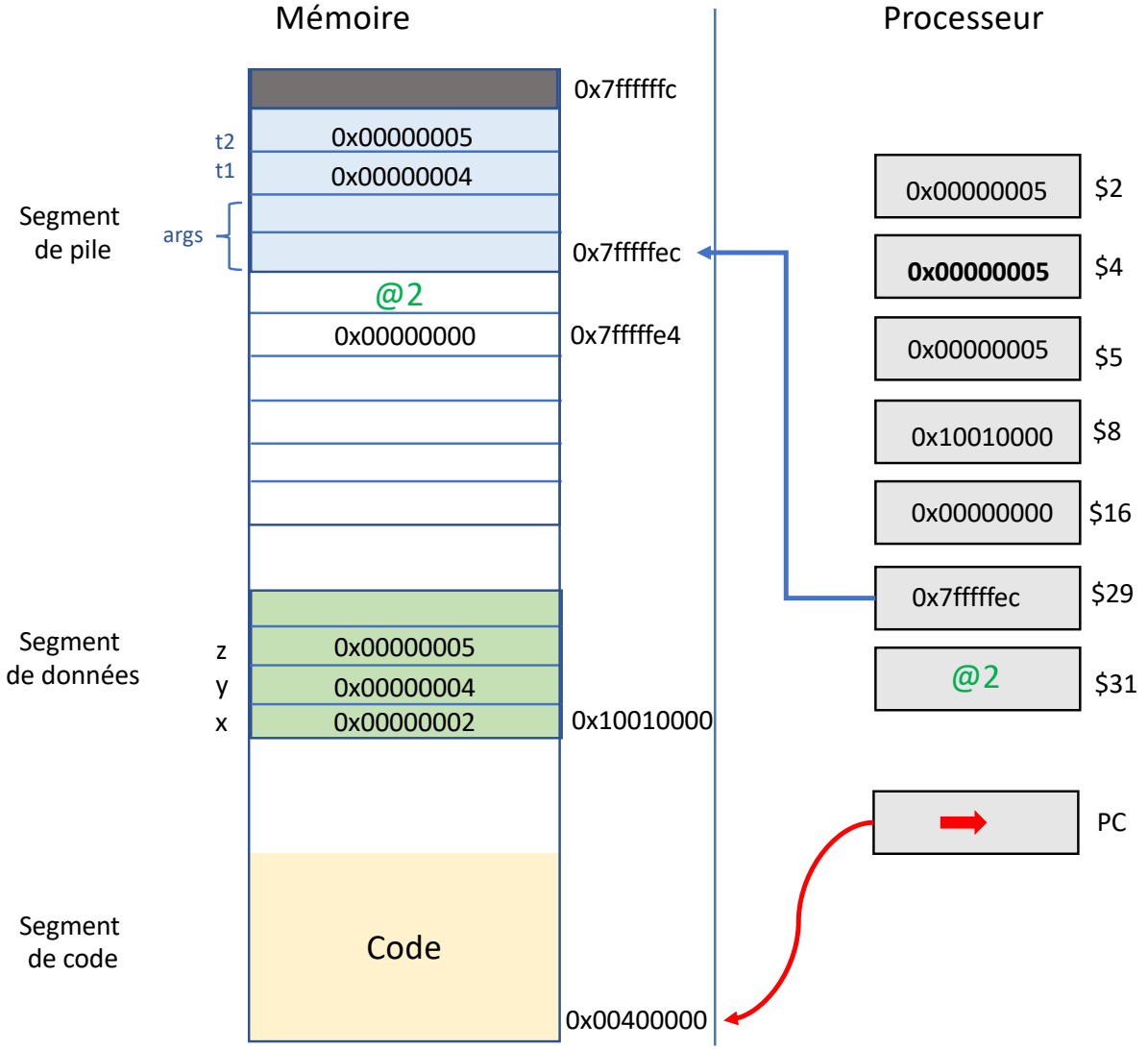


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

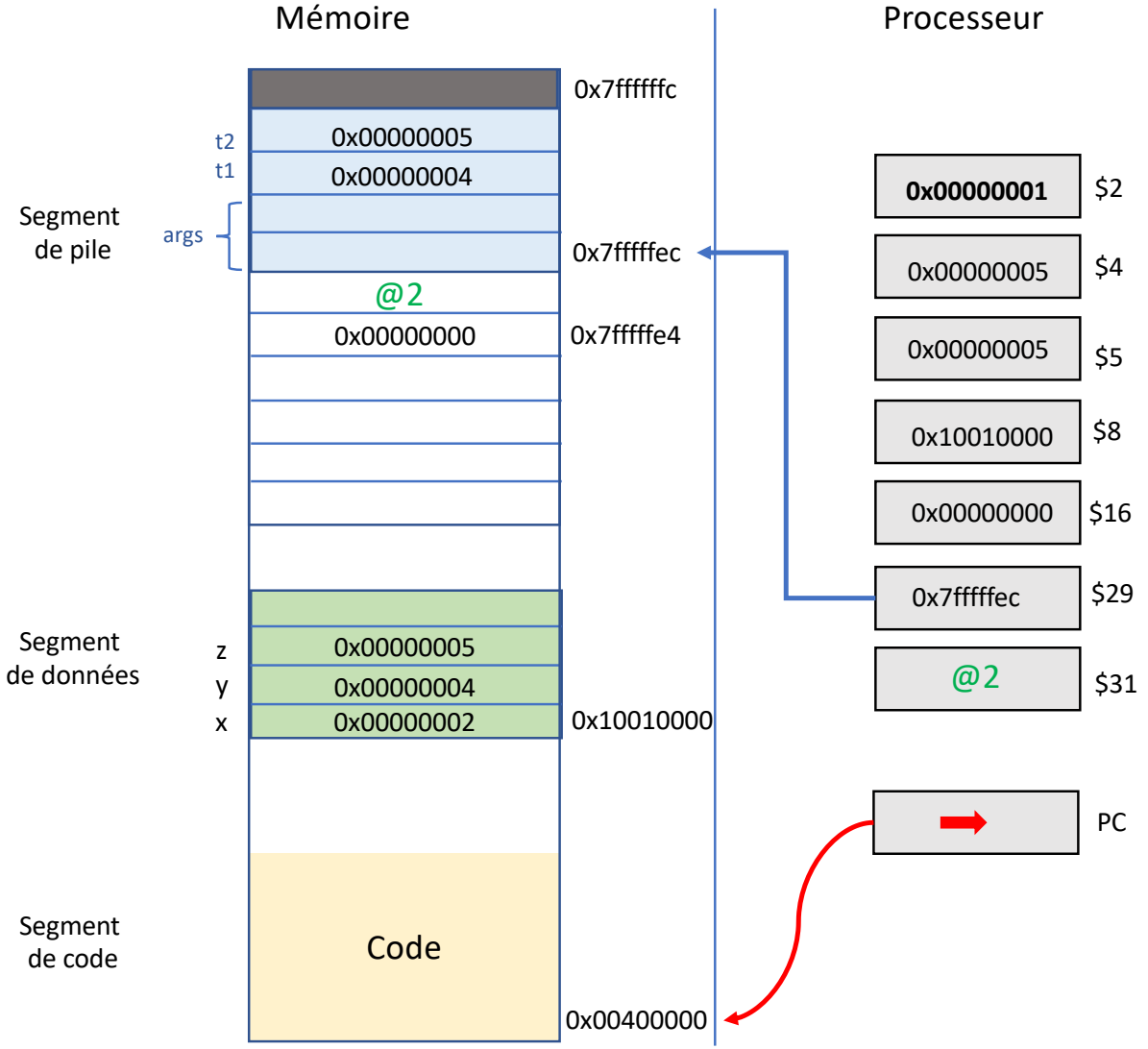


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

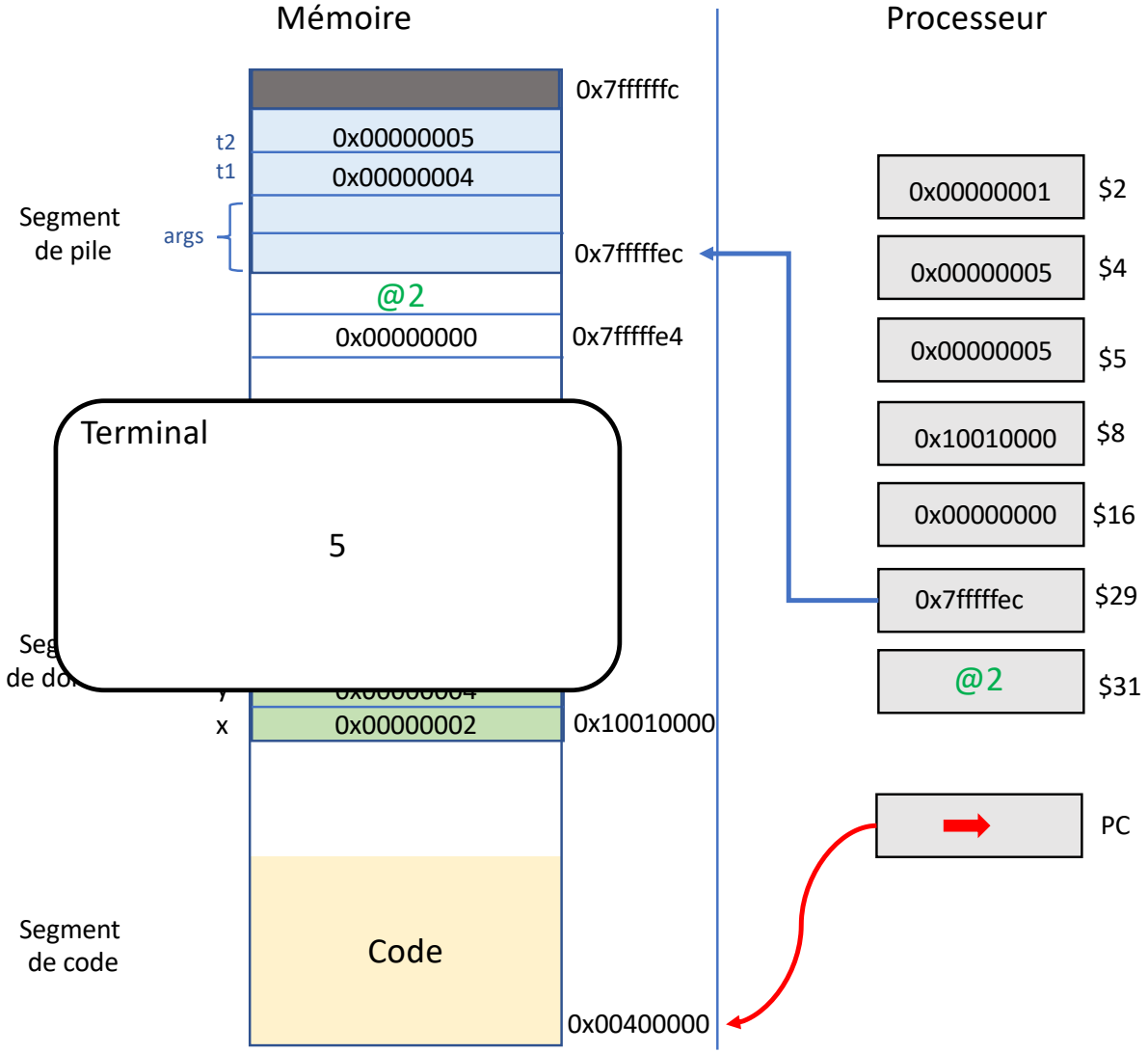


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

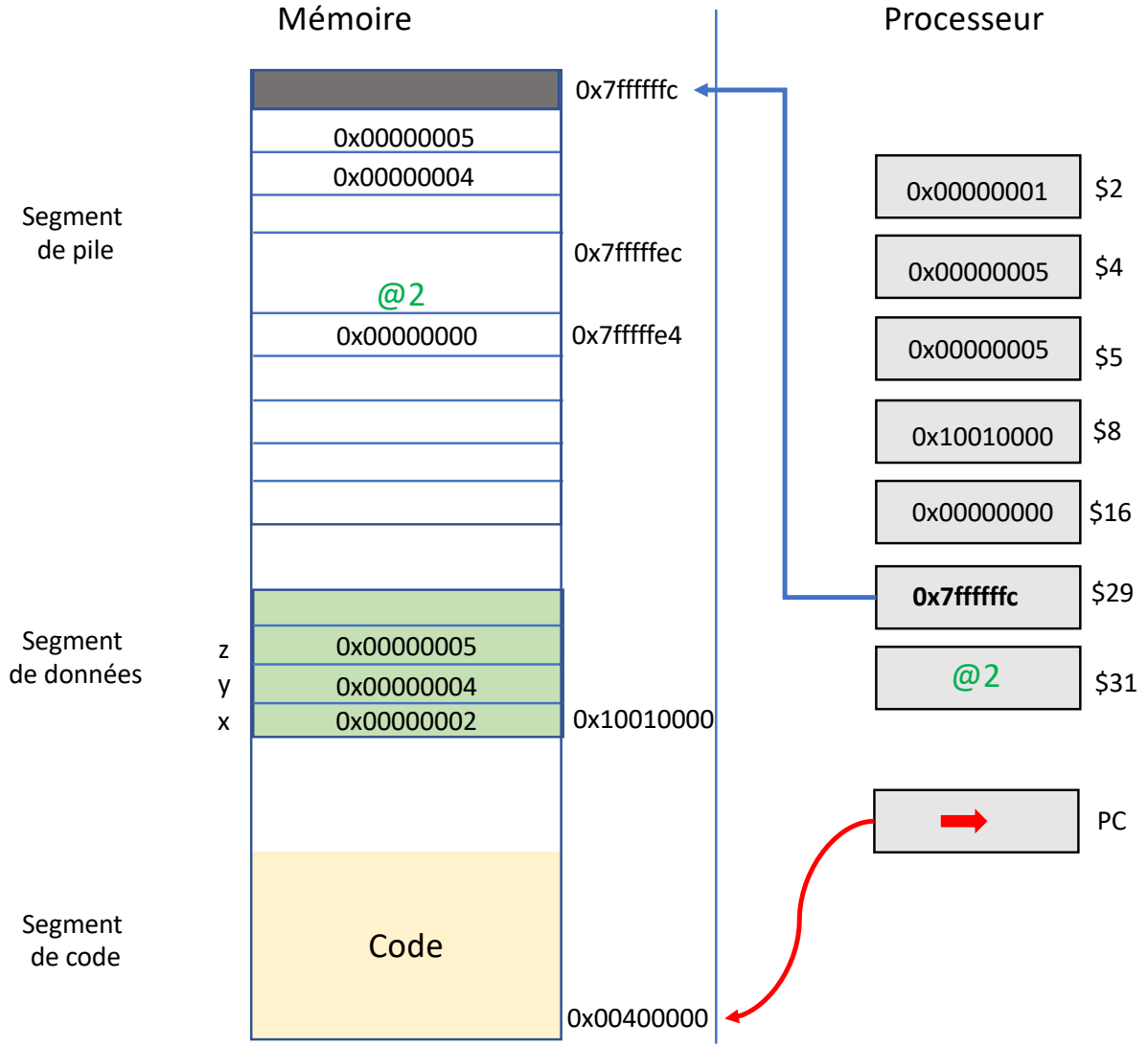


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```

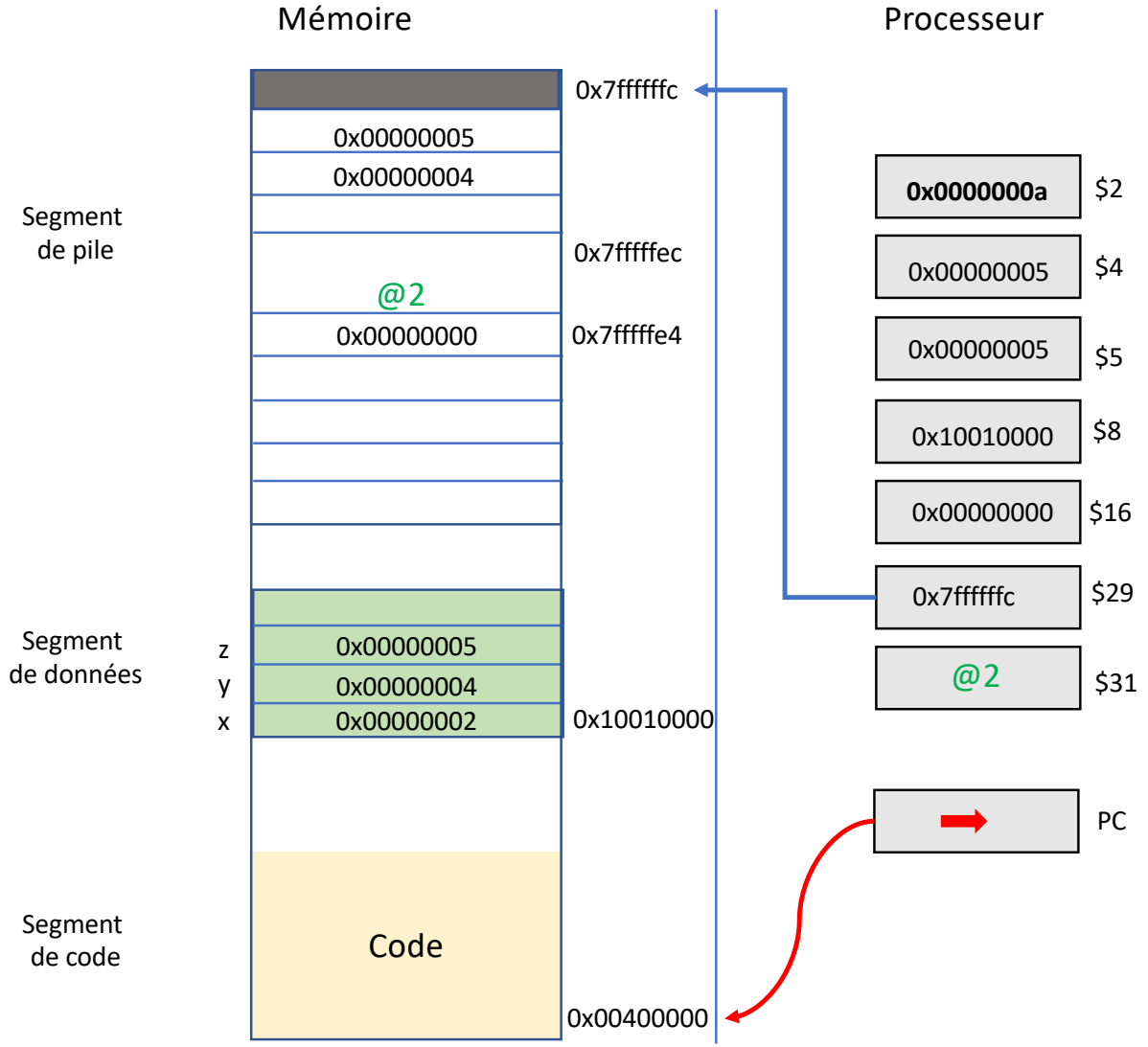
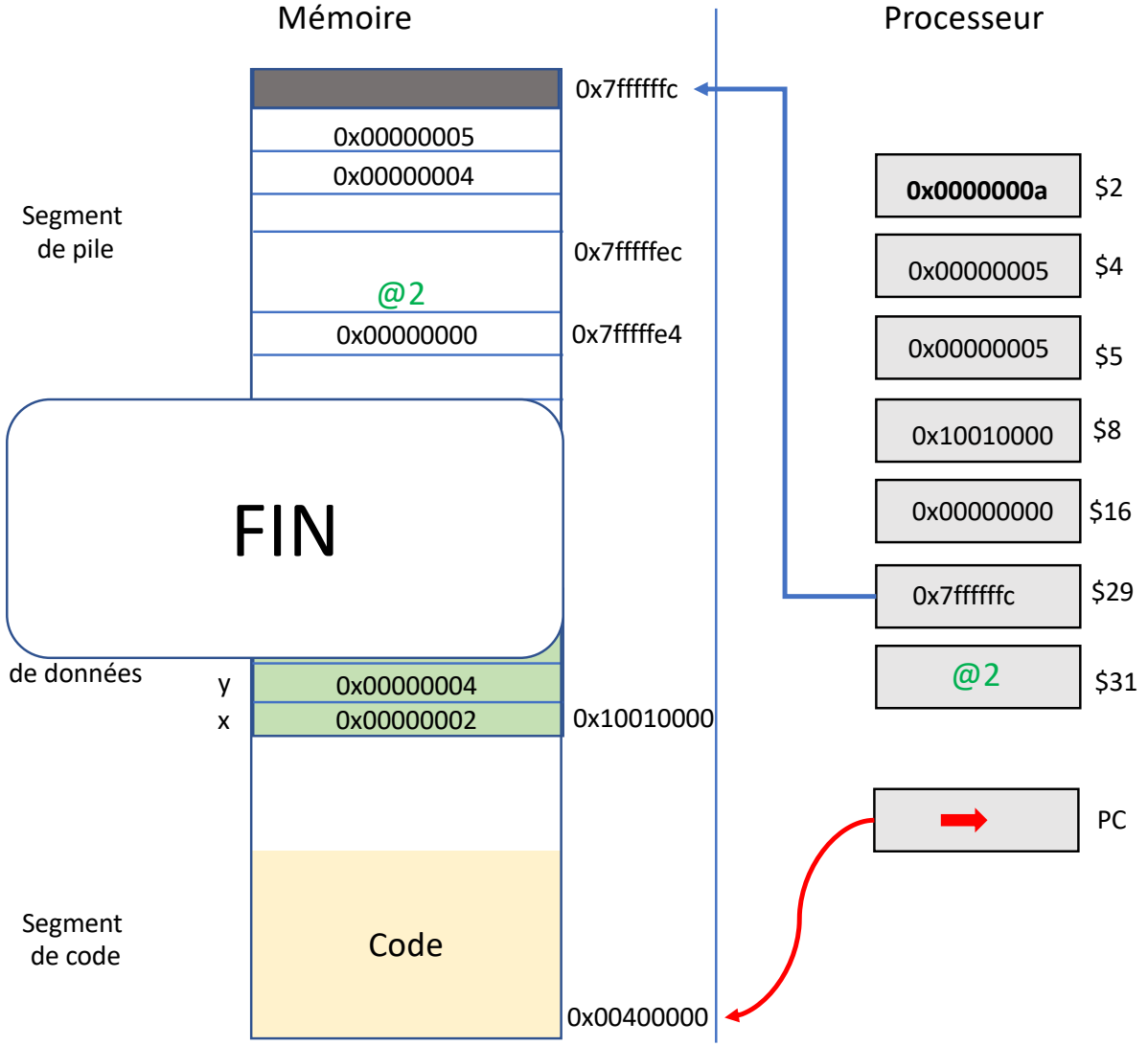


Illustration de l'exécution

```
.data
 x:  .word 2
 y:  .word 4
 z:  .word 5
.text
 addiu $29, $29, -16       # 2 var locales + 2 param max
# t1 = max2(x,y)
 lui    $8, 0x1001         # @x dans $8 non persistant
 lw     $4, 0($8)          # 1er paramètre = x dans $4
 lw     $5, 4($8)          # 2ème paramètre = y dans $5
 jal    max2               # premier appel à max2
# @1 adresse retour premier appel, $2 contient le résultat
 sw    $2, 8($29)          # écriture du résultat dans t1
 # t2 = max2(t1,z)
 lw     $4, 8($29)         # 1er paramètre = t1 dans $4
 lui    $8, 0x1001         # $8 non persistant => rechargement de @x
 lw     $5, 8($8)          # 2ème paramètre = z dans $5
 jal    max2               # second appel à max2
 # @2  adresse retour 2eme appel, $2 contient le résultat
 sw     $2, 12($29)        # écriture du résultat dans t2
 # printf("%d",t2)
 lw     $4, 12($29)        # lecture t2
 ori    $2, $0, 1          # affichage
 syscall
# terminaison
 addiu $29, $29, +16       # désallocation emplacements pile
 ori     $2, $0, 10
 syscall
max2:
 addiu $29, $29,-8          # nr = 1 ($16 ) + $31, nv = 0, max_arg = 0
 sw      $31, 4($29)
 sw      $16, 0($29)
 slt     $16, $4, $5        #$5 vaut 1 si a < b
 beq     $16, $0, a_max
 ori     $2, $5, 0          # valeur de retour dans $2
 j       fin
a_max:
 ori     $2, $4, 0          # valeur de retour dans $2
fin:
 lw      $31, 4($29)
 lw      $16, 0($29)
 addiu $29, $29,+8
 jr $31
```



Paramètres de type tableau

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

void main(){
  str2cap(str);
  printf("%s", str);
  exit();
}

void str2cap(char * str){
  int i = 0;
  while (str[i] != '\0') {
     str[i] = str[i] - 0x20;
     i = i + 1;
  }
  return;
}
```

- La fonction `str2cap` a 1 paramètre qui est une chaîne de caractères et ne renvoie rien : c'est la chaîne qui est transformée.
- La fonction transforme en majuscule une chaine contenant des minuscules.

Paramètres de type tableau : exemple d'appel

```
.data
str: .asciiz "helloworld"

.text
 # prologue du main
 addiu $29, $29, -4   # 0 variable locale + 1 paramètre
 # str2cap(str)
 lui  $4, 0x1001      # $4 1er arg = 0x10010000 = @str
 jal str2cap
 # printf("%s", str);
 lui  $4, 0x1001      # $4 1er arg = 0x10010000 = @str
 ori  $2, $0, 4       # $5 = 5 affichage chaine
 syscall
 # epilogue du main
 addiu $29, $29, 4    # Désallocation dans la pile
 ori  $2, $0, 10
 syscall              # exit
```

- L'adresse de `str` est passée en paramètre (dans `$4`)
- `$4` n'est pas persistant : relecture de l'adresse de `str` pour l'affichage de la chaine après l'appel

Paramètres de type tableau : exemple de fonction

```
str2cap:
  # prologue A ECRIRE
  addiu $29, $29, -X?  # nv = 1 (i) + nr = ? + na = 0
  # sauvegarde registres persistants àécrire + paramètres si besoin

  ori  $8, $0, 0          # i <- 0, $8 <- valeur de i
  # Corps
loop:
  # Condition de la boucle while
  addu $10, $8, $4       # $10 <- &str[i] = @str + i
  lb   $11, 0($10)       # $11 <- str[i];
  beq  $11, $0, finloop  # test str[i] == 0
  # Corps du while
  addiu $11, $11, -0x20 # $11 <- str[i]-0x20 ($11 contient déjà str[i])
  sb   $11, 0($10)       # écriture str[i] ($10 contient déjà &str[i])
  addiu $8, $8, 1        # incrémentation valeur de i
  # Retour au début de la boucle
  j    loop

finloop:
  # épilogue A ECRIRE
  # restauration registres persistants àécrire
  addiu $29, $29, +X?
  jr $31
```

Paramètres de type tableau : exemple de fonction

```
str2cap:
  # prologue
  addiu $29, $29, -8  # nv = 1 + na = 0 + nr = 0 + $31
  # sauvegarde registres
  sw $31, 4($29)

  ori  $8, $0, 0          # i <- 0, $8 <- valeur de i
  # Corps
loop:
  # Condition de la boucle while
  addu $10, $8, $4       # $10 <- &str[i] = @str + i
  lb   $11, 0($10)       # $11 <- str[i]!;
  beq  $11, $0, finloop  # test str[i] == 0
  # Corps du while
  addiu $11, $11, -0x20 # $11 <- str[i]-0x20 ($11 contient déjà str[i])
  sb   $11, 0($10)       # écriture str[i] ($10 contient déjà &str[i])
  addiu $8, $8, 1        # incrémentation valeur de i
  # Retour au début de la boucle
  j    loop

finloop:
  # épilogue
  lw $31, 4($29)
  addiu $29, $29, +8
  jr $31
```

Paramètres passés par valeur versus par adresse

```
int x = 2, y = 4, z = 5, tmp;

void main(){
  max2bis(x,y,&tmp);
  max2bis(tmp,z,&tmp);
  printf("%d",tmp);
  exit();
}

void max2bis(int a, int b, int* res){
  if (a < b)
    *res = b;
  else
    *res = a;
  return;
}
```

- La fonction `max2bis` a 3 paramètres et ne renvoie pas de valeur
- Le 3ème paramètre est une adresse
- Le deuxième appel a pour 1er argument la valeur de `tmp` et pour 3ème argument l'adresse de la variable globale `tmp`.

Code exemple : programme principal (début)

```
.data
 x: .word 2
 y: .word 4
 z: .word 5
tmp: .space

.text
 addiu $29, $29, -12 # 3 mots : 0 var locale, 3 paramètres max

 # max2bis(x,y,&tmp)
 lui $8, 0x1001          # @x dans $8 non persistant
 lw $4, 0($8)            # 1er paramètre = x dans $4
 lw $5, 4($8)            # 2ème paramètre = y dans $5
 ori $6, $8, 12          # 3ème paramètre = &tmp dans $6
 jal max2bis             # premier appel àmax2bis

 # max2bis(tmp,z,&tmp)
 lui $8, 0x1001        # $8 non persistant => rechargement de @x
 lw $4, 12($8)         # 1er paramètre = tmp dans $4
 lw $5, 8($8)          # 2ème paramètre = z dans $5
 ori $6, $8, 12        # 3ème paramètre = &tmp dans $6
 jal max2bis           # second appel àmax2
 ...
```

Code exemple : programme principal (fin)

```
# printf("%d",tmp)
  lui $8, 0x1001  # $8 non persistant => rechargement de @x
  lw $4, 12($8)    # lecture tmp
  ori $2, $0, 1   # affichage
  syscall

# terminaison
  addiu $29, $29, +12 # désallocation emplacements pile
  ori $2, $0, 10
  syscall
```

- C'est l'adresse de `tmp` qui est passée en 3ème paramètre, non sa valeur
- Pour le 2ème appel, la valeur de `tmp` est passée en 1er paramètre
- Au retour de chaque appel, il faut lire la valeur de `tmp`, modifiée par l'appel à `max2bis` (et recharger le contenu de `$8` avant) :
  après le premier pour la passer en paramètre au 2ème appel,
  après le 2ème appel pour l'afficher.
- On aurait pu utiliser `$16` au lieu de `$8` et éviter le rechargement car `$16` est persistant !

Ce qu'on a vu

Fonctions et appels de fonction en assembleur

- Notion de conventions d'appel : toutes les architectures en ont, reposant sur une pile, souvent avec une instruction dédiée pour l'appel avec un registre dédié au stockage de l'adresse de retour
- Les conventions sont fixées par l'ABI
- La version simplifiée utilisée est décrite dans ce cours et dans la documentation disponible sur le site de l'UE `https://www-licence.ufr-info-p6.jussieu.fr/lmd/licence/2021/ue/LU3IN029-2021oct/doc_MIPS32.pdf` et sur Moodle.
- La notion de pile est essentielle pour le bon fonctionnement du code en présence d'appels de fonction
- Ensemble d'exemples avec des fonctions et des appels de fonction, paramètres de type élémentaire (valeur simple ou adresse)
- La prochaine semaine : fonctions imbriquées, récursives et le passage de variables locales par adresse en paramètre.