

Architecture des ordinateurs

Cours 8

Responsable de l'UE : Karine Heydemann

Contact : prenom.nom@lip6.fr

Plan du cours 8

- 1 Rappels sur les appels de fonction
- 2 Exemple de fonctions imbriquées
- 3 Exemple de fonctions récursives
- 4 Paramètres passés par valeur versus par adresse
- 5 Pour aller plus loin

Contrat appelant - appelé

Conventions d'appels

Règles définissant le protocole d'échange de valeurs entre une fonction appelante et une fonction appelée, notamment :

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

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

Conventions d'appel

Utilité et importance

- Les respecter permet à une fonction d'être appelée par tout code qui connaît sa signature : nom + nombre, ordre et type de ses arguments
- 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

Fonctions imbriquées et récursives

- Fonctions imbriquées : il n'y a rien de plus à faire pour appeler une fonction depuis une fonction que de suivre scrupuleusement les conventions d'appel !
- Fonction récursives : c'est une fonction qui s'appelle elle-même mais cela ne change rien, il suffit de suivre les conventions d'appel

Résumé : fonction et appel de fonction en MIPS

Fonctions

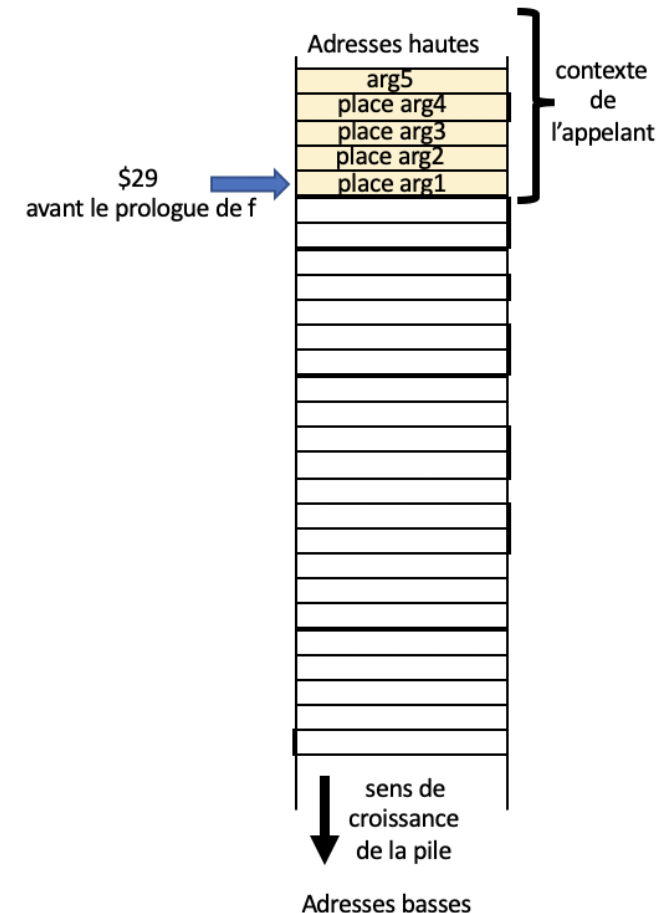
- 1 Alloue son contexte d'exécution sur la pile
- 2 Sauvegarde les registres persistants qu'elle utilise + \$31
- 3 Récupère la valeur des arguments de l'appel dans \$4-\$7 + les suivants sur la pile
- 4 Réalise un traitement à partir des valeurs des arguments
- 5 Place la valeur du résultat dans \$2
- 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 arguments dans \$4-\$7 + les suivants sur la 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

Etat de la pile à l'entrée d'une fonction f

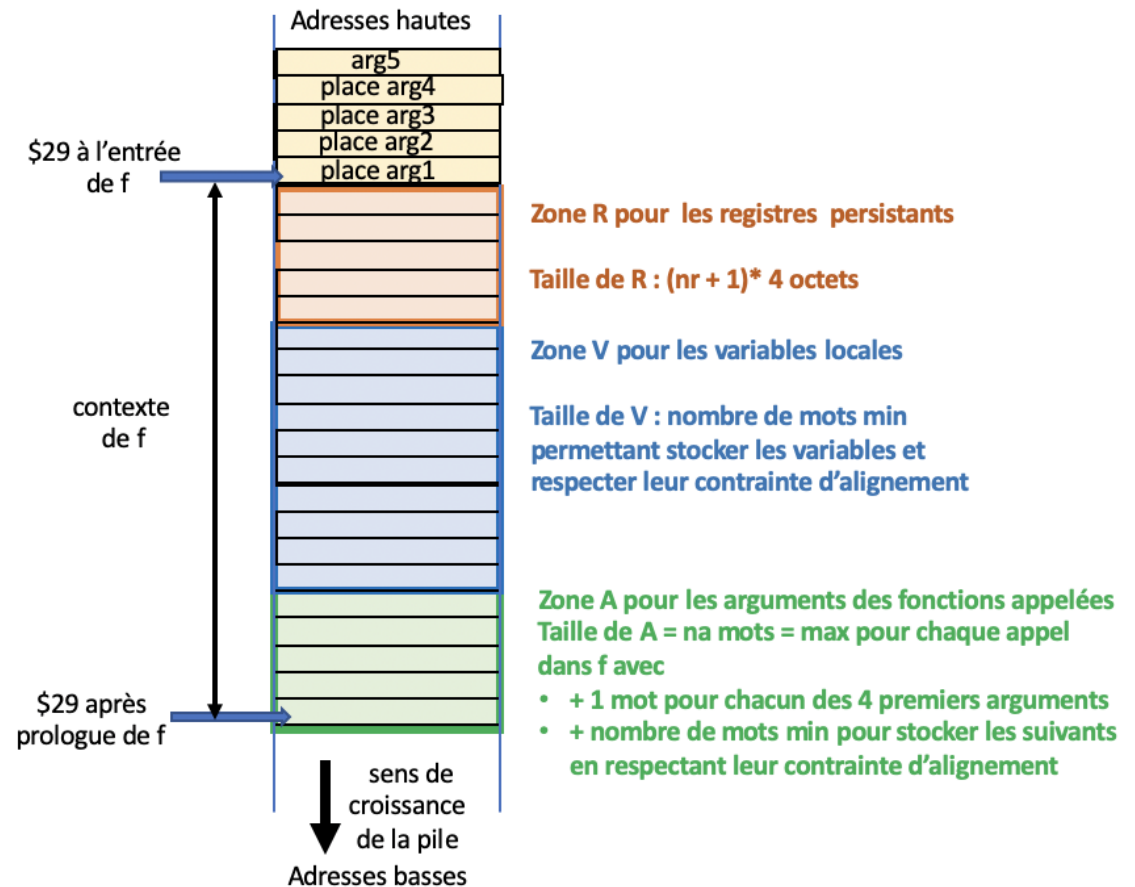
- Le pointeur de pile pointe vers le contexte de la fonction appelante qui contient au sommet des emplacements pour les arguments des fonctions qu'elle appelle
- Le pointeur de pile désigne l'emplacement du premier argument de la fonction f



Prologue et épilogue d'une fonction f

Contexte d'exécution d'une fonction

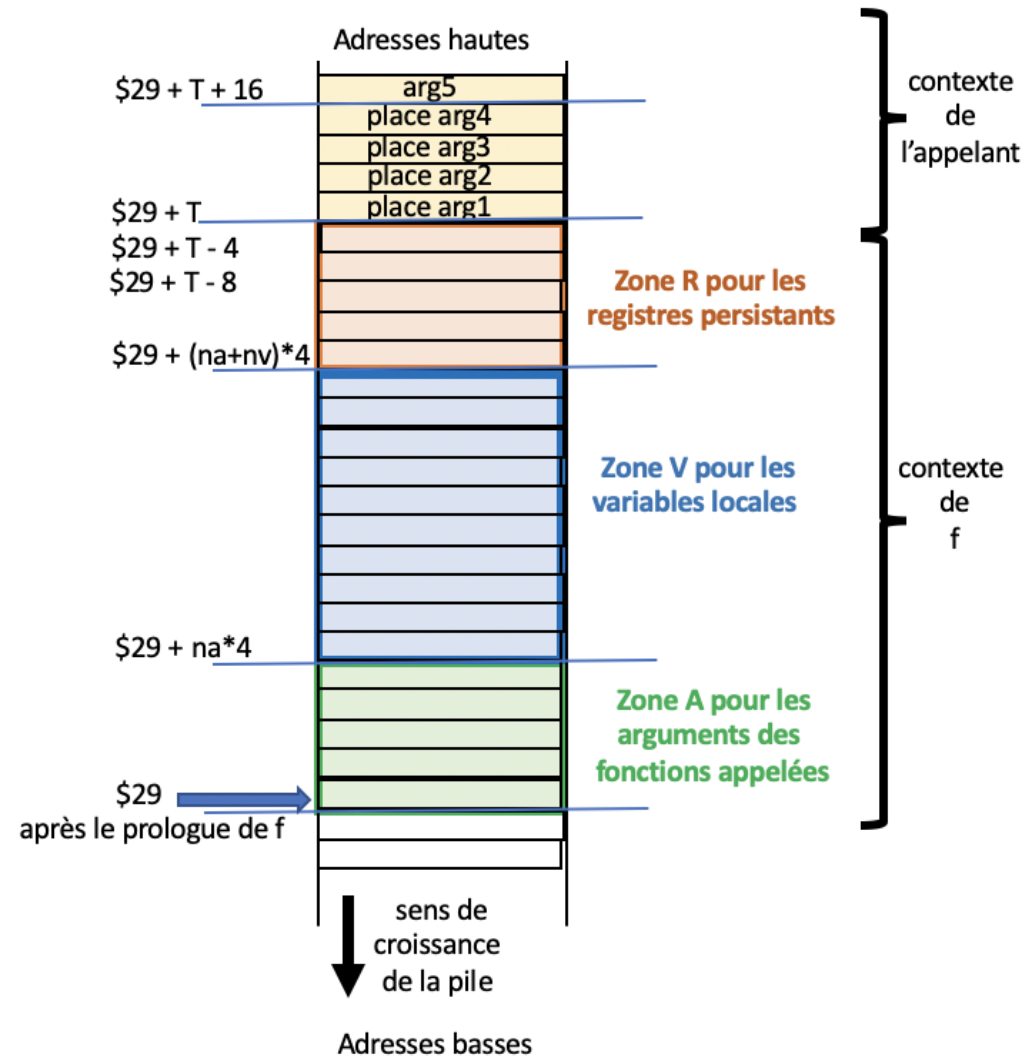
- Alloué/désalloué par la fonction f dans son prologue/épilogue
- Contient 3 zones d'emplacements :
 - 1 Zone R : sauvegarde des registres persistants utilisés dans la fonction $f + \$31$
 - 2 Zone V : variables locales de f
 - 3 Zone A : arguments des fonctions appelées par f
- Le contexte de la fonction appelante est en dessous (du point de vue du sommet de la pile) de celui de la fonction appelée



Contexte d'une fonction f

En partant du sommet de pile, on trouve

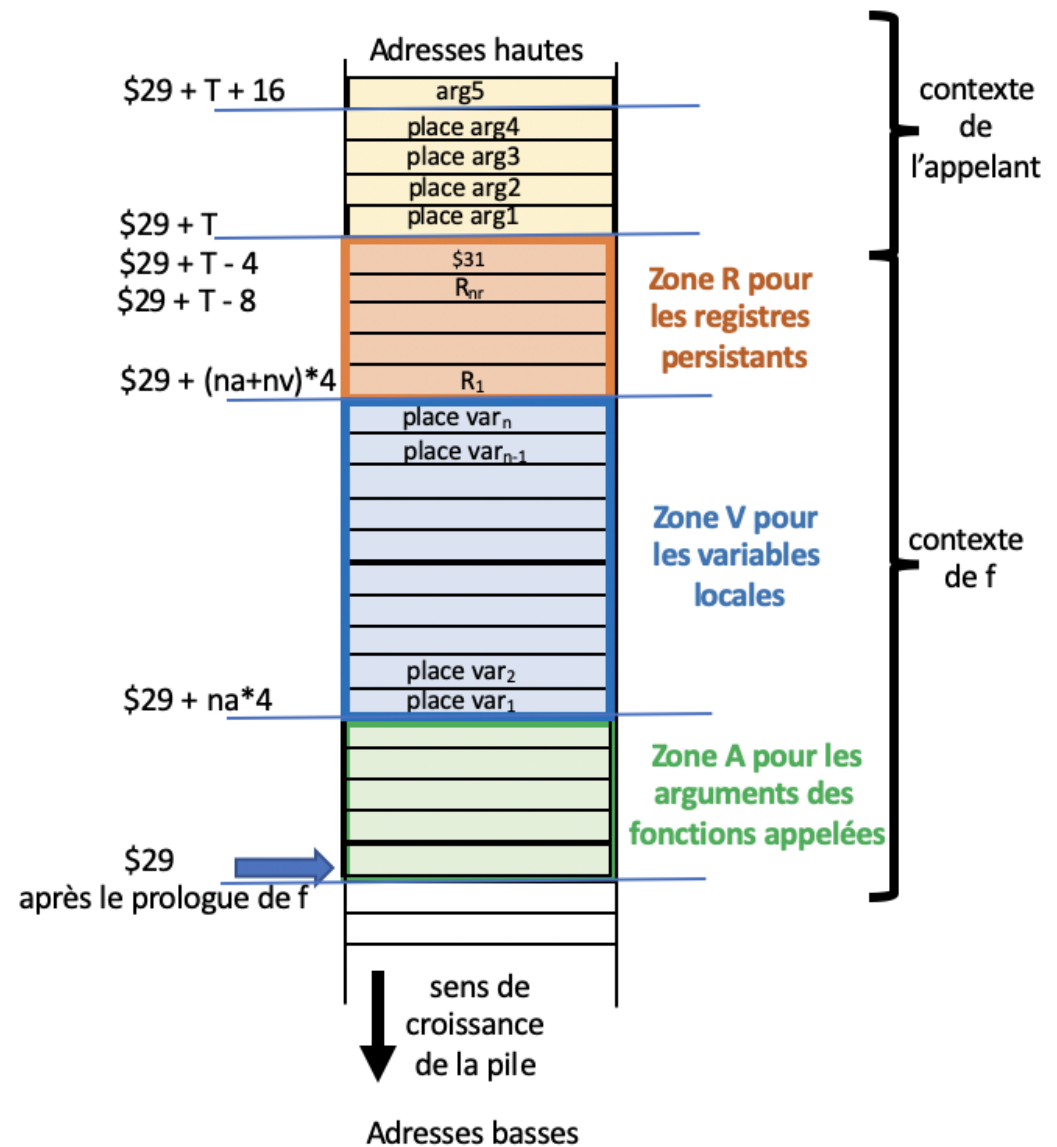
- la zone A pour les arguments des fonctions appelées par f
- la zone V pour les variables locales de f :
la première déclarée a pour adresse $\$29 + na*4$
- la zone R avec les emplacements pour la sauvegarde du contenu des registres persistants utilisés dans la fonction $f + \$31$:
 $\$31$ est rangé le plus profondément à $\$29 + (na + nv + nr + 1) * 4 - 4$ soit **$\$29 + T - 4$**
- Le contexte de la fonction appelante contient les emplacements pour les arguments de f :
l'emplacement du premier se trouve à $\$29 + T$, le 5ème argument est à $\$29 + T + 16$



Contenu du contexte d'une fonction f

En partant du sommet de pile, on trouve

- la zone A pour les arguments des fonctions appelées par f
- la zone V pour les variables locales de f :
la première déclarée a pour adresse $\$29 + na*4$
- la zone R avec les emplacements pour la sauvegarde du contenu des registres persistants utilisés dans la fonction $f + \$31$:
 $\$31$ est rangé le plus profondément à $\$29 + (na + nv + nr + 1) * 4 - 4$ soit $\$29 + T - 4$
- Le contexte de la fonction appelante contient les emplacements pour les arguments de f :
l'emplacement du premier se trouve à $\$29 + T$, le 5ème argument est à $\$29 + T + 16$



Écriture du `main` : recommandations

Détermination de la taille du contexte

- Dans le code source, déterminer n_a le nombre de mots max pour stocker les arguments des fonctions appelées. Pour chaque appel, on compte :
 - 1 mot pour chacun des 4 premiers arguments
 - Pour les arguments suivants, autant de mots que nécessaire en fonction de leur taille et leur contrainte d'alignement

Si au plus 4 arguments pour chacun des appels, alors n_a = nombre d'arguments max.

- À partir du code source, déduire le nombre d'octets puis de mots n_v nécessaires aux variables locales : (voir cours 6)

Écriture du code

- Écrire le prologue : allocation du contexte, si besoin initialisation des variables locales en pile
- Écrire le code du corps du `main`
- Écrire l'épilogue : désallocation du contexte puis terminaison du programme

Maximum de 3 nombres

```
int x = 2, y = 4, z = 5;
void main() {
    int res = max3(x, y, z);
    printf("%d", res);
    exit();
}

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

int max3(int a, int b, int c) {
    int tmp;
    tmp = max2(a, b);
    return max2(tmp, c);
}
```

- Le `main` appelle la fonction `max3`
- La fonction `max3` appelle 2 fois la fonction `max2`

Code exemple : programme principal

```
.data
x:    .word 2
y:    .word 4
z:    .word 5
.text
addiu $29, $29, -16 # nv =1 + na = 3
# tmp = max3(x,y,z)
lui   $8, 0x1001
lw    $4, 0($8) # $4 = 1er arg = x
lw    $5, 4($8) # $5 = 2nd arg = y
lw    $6, 8($8) # $6 = 3eme arg = z
jal   max3
ori   $4, $2, 0 # affichage res
ori   $2, $0, 1 # affichage entier
syscall
# terminaison
addiu $29, $29, +16
ori   $2, $0, 10
syscall
```

```
int x = 2, y = 4, z = 5;
void main() {
    int res = max3(x,y,z);
    printf("%d", res);
    exit();
}
```

- Application stricte des conventions d'appel
- NB : on se passe entièrement de `res` mais son emplacement est alloué sur la pile (et désalloué)

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 :
 - 1ère étape car on ne connaît pas les registres persistants à sauvegarder avant de l'avoir écrit
 - Écrire les lectures et écritures en pile (d'arguments ou variables locales qui ont une adresse relative au pointeur de pile) avec ?? pour l'immédiat des instructions de transfert mémoire correspondantes
 - 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 n_r puis T_R sans oublier \$31
 - À partir du code source de la fonction, déterminer n_a le nombre de mots max pour stocker les arguments des fonctions appelées
 n_a = nombre d'arguments max si tous les appels ont moins de 4 arguments.
 - À partir du code source de la fonction, déduire le nombre de d'octets T_V puis de mots n_v nécessaires aux emplacements des variables locales : (voir cours 6)

Écriture d'une fonction : recommandations

Étapes conseillées

- 4 Écrire le prologue :
 - Allocation des emplacements sur la pile
 - Sauvegarde des registres persistants
 - Sauvegarde des arguments et initialisation des variables locales si besoin
- 5 Dans le corps de la fonction, adapter les déplacements relatifs aux pointeurs de pile quand nécessaire (accès à aux variables locales, ou aux arguments de la fonction).
NB : si besoin dessiner la pile pour déterminer les adresses des variables locales et des arguments.
- 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

Maximum de 3 nombres, rappel de max2

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

int max3(int a, int b, int c) {
    int tmp;
    tmp = max2(a, b);
    return max2(tmp, c);
}
```

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

    # corps
    # test (a < b)
    slt    $16, $4, $5 # $16=1 si a < b
    bne    $16, $0, b_max
    ori    $2, $4, 0    # cas a > b
    j      fin_max2

b_max:
    ori    $2, $5, 0

fin_max2:

    # epilogue
    lw     $31, 4($29)
    lw     $16, 0($29)
    addiu  $29, $29, +8
    jr     $31
```

Code exemple : fonction max3

```
max3: # nv=1 + na=2 + nr=0 + 1
      addiu $29, $29, -16 # 4 mots
      sw     $31, 12($29)
      # sauvegarde 3eme arg (c)
      sw     $6, 24($29)

      # tmp = max2(a,b) -- $4=a, $5=b
      jal    max2
      # $2 vaut max2(a,b) (tmp)
      # max2(tmp,c)
      ori    $4, $2, 0      # 1er arg = tmp
      lw     $5, 24($29)    # 2eme arg = c
      jal    max2
      # $2 vaut max2(tmp,c)

      lw     $31, 12($29)
      addiu  $29, $29, +16
      jr     $31
```

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

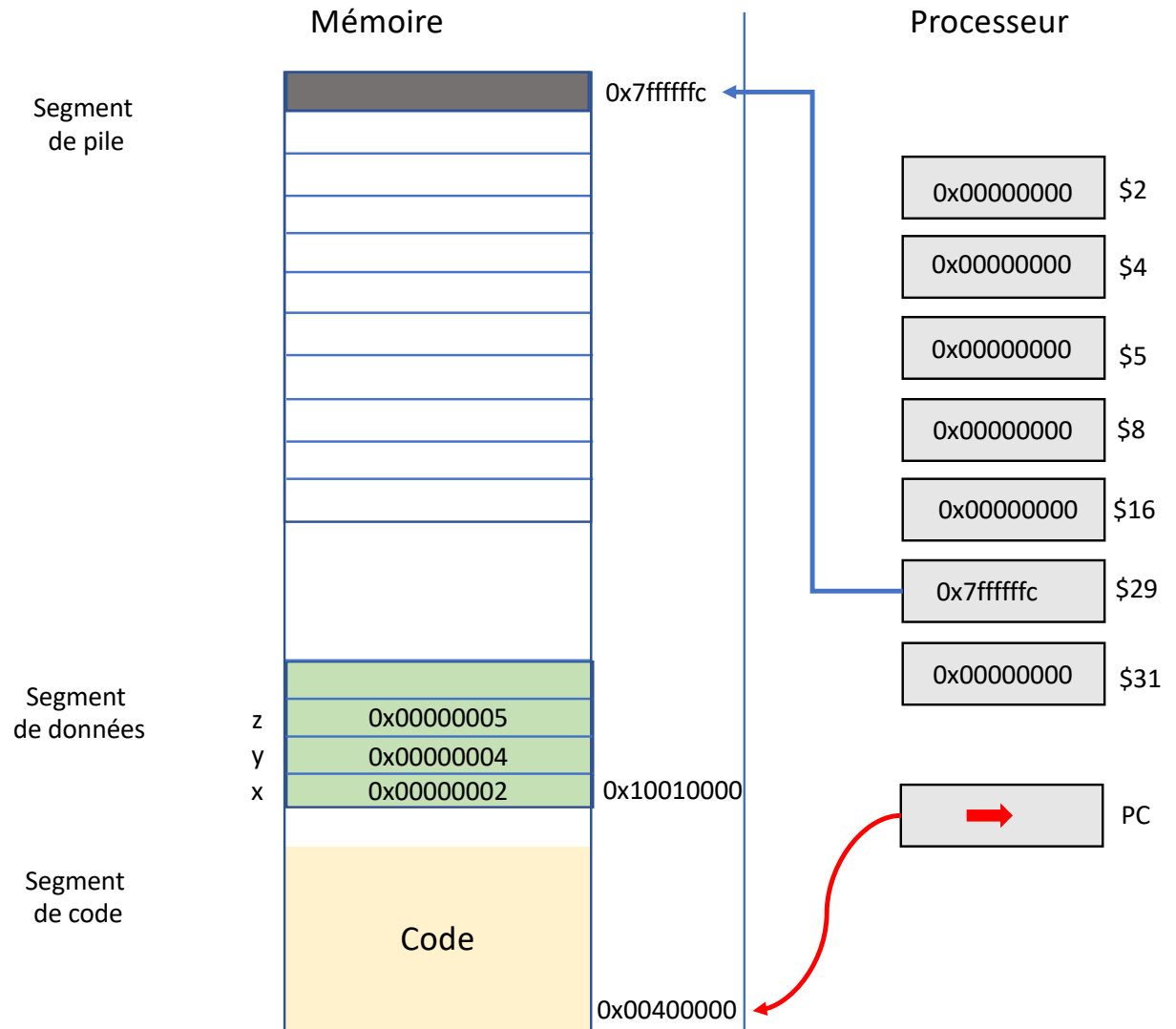
int max3(int a, int b, int c) {
    int tmp;
    tmp = max2(a,b);
    return max2(tmp,c);
}
```

- On doit sauvegarder la valeur de `c` (3ème arg), dans son emplacement situé dans le contexte de la fonction appelante, car sa valeur est utilisée dans le 2ème appel à `max2`
- La valeur de retour de `max3` est la valeur de retour du 2ème appel à `max2`
- On se passe entièrement de `tmp` mais son emplacement est alloué sur la pile (et désalloué)

Exécution et évolution de la pile

```

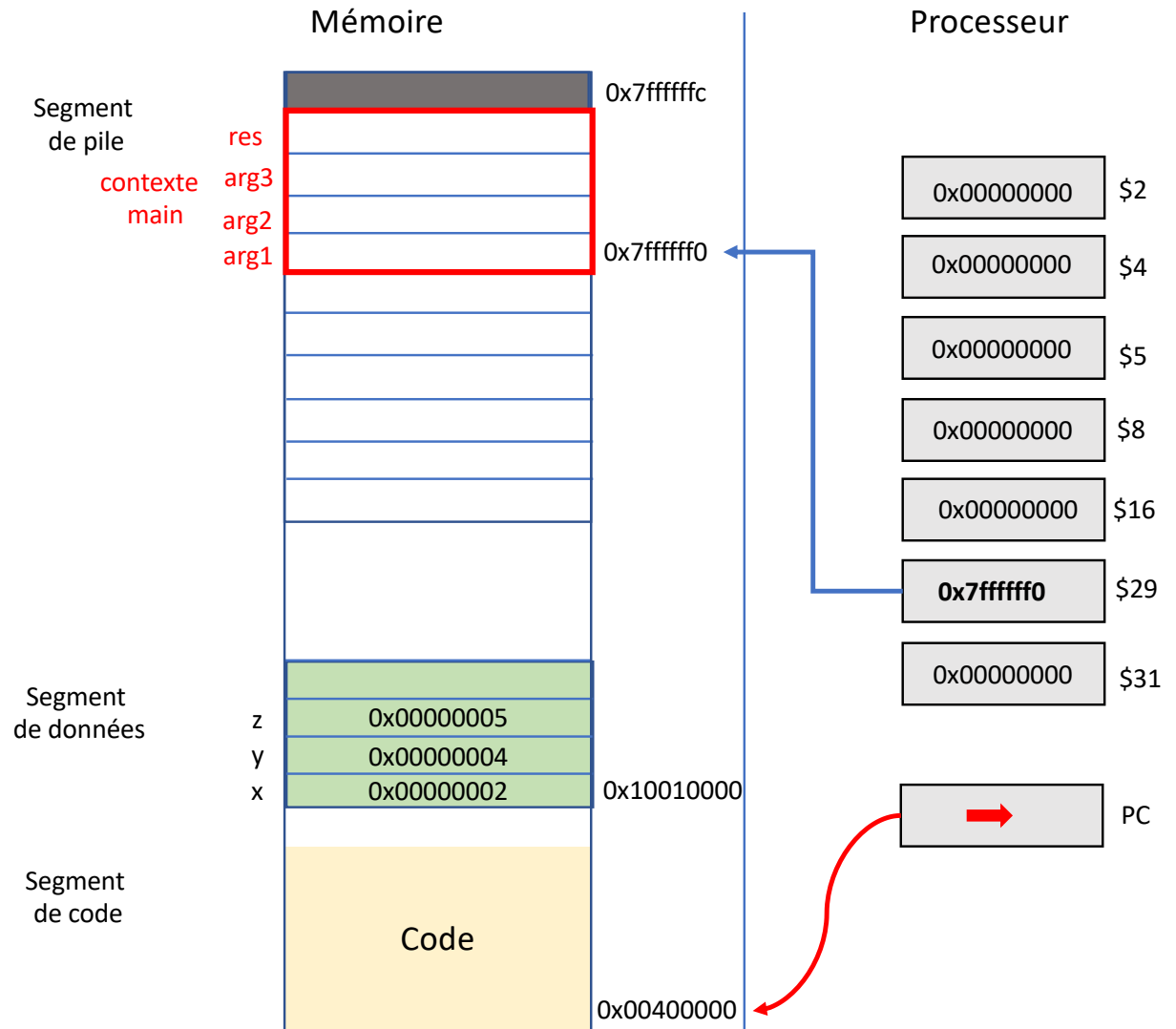
.data
x: .word 2
y: .word 4
z: .word 5
.text
→ addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)            # 1er arg = x
lw $5, 4($8)            # 2ème arg = y
lw $6, 8($8)            # 3ème arg = z
jal max3                # appel à max3
ori $4, $2, 0
ori $2, $0, 1           # affichage
syscall
addiu $29, $29, +16     # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16     # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)
sw $6, 24($29)          # sauvegarde 3eme arg (c)
jal max2
ori $4, $2, 0           # 1er arg=tmp
lw $5, 24($29)          # 2eme arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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         # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

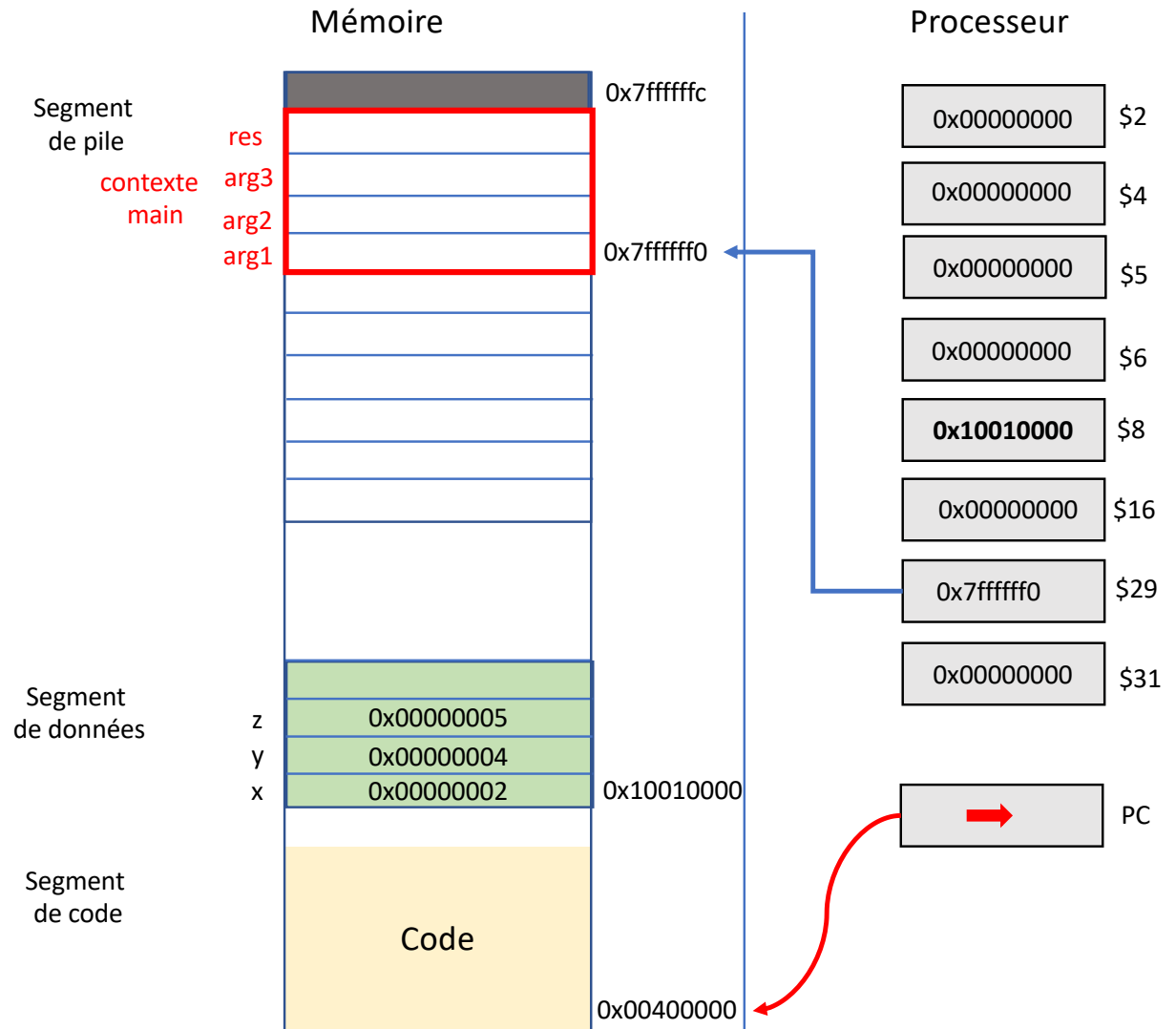
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)
sw $6, 24($29)         # sauvegarde 3eme arg (c)
jal max2
ori $4, $2, 0
lw $5, 24($29)         # 2eme arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

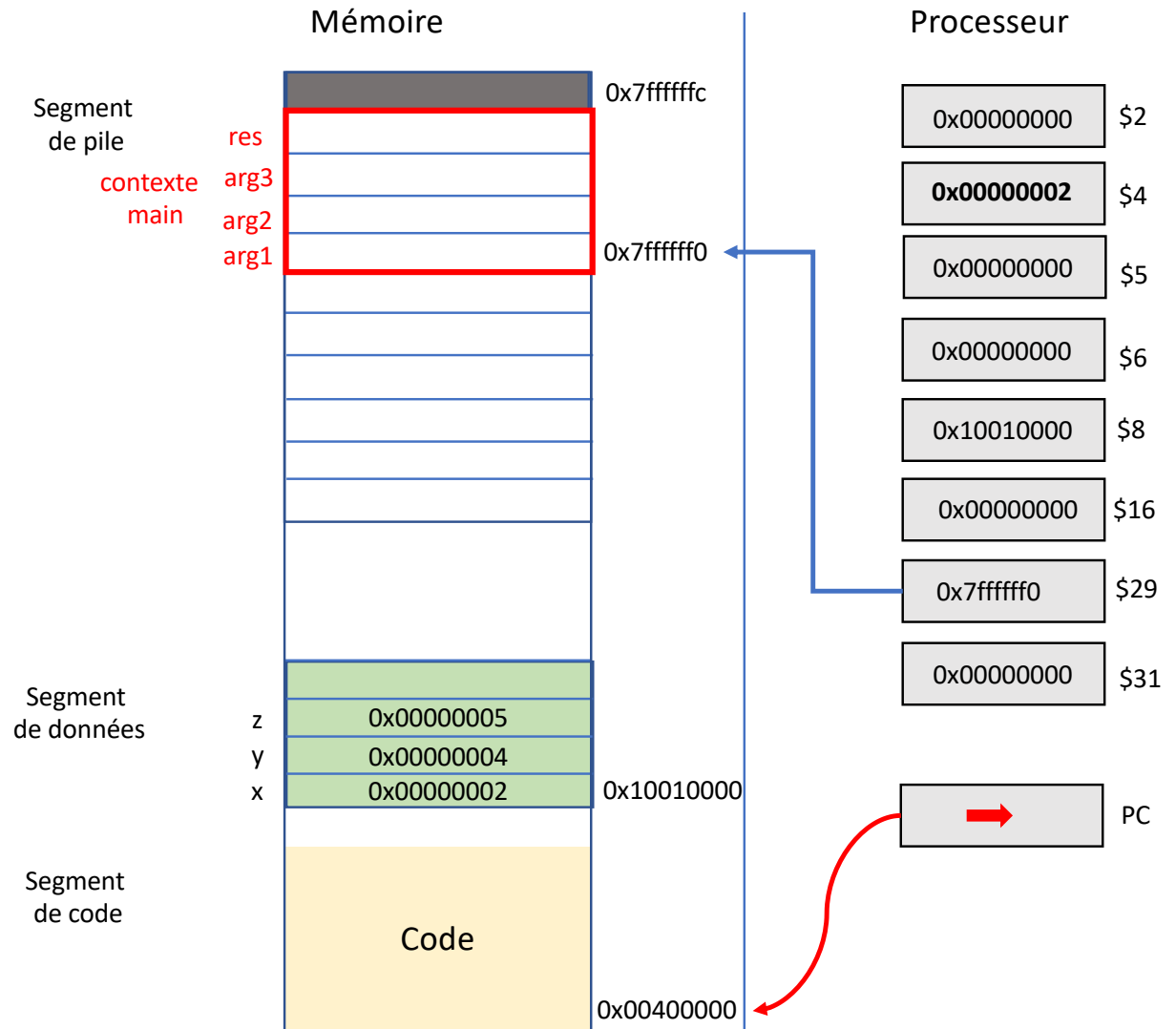
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3ème arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0          # 1er arg=tmp
lw $5, 24($29)         # 2ème arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

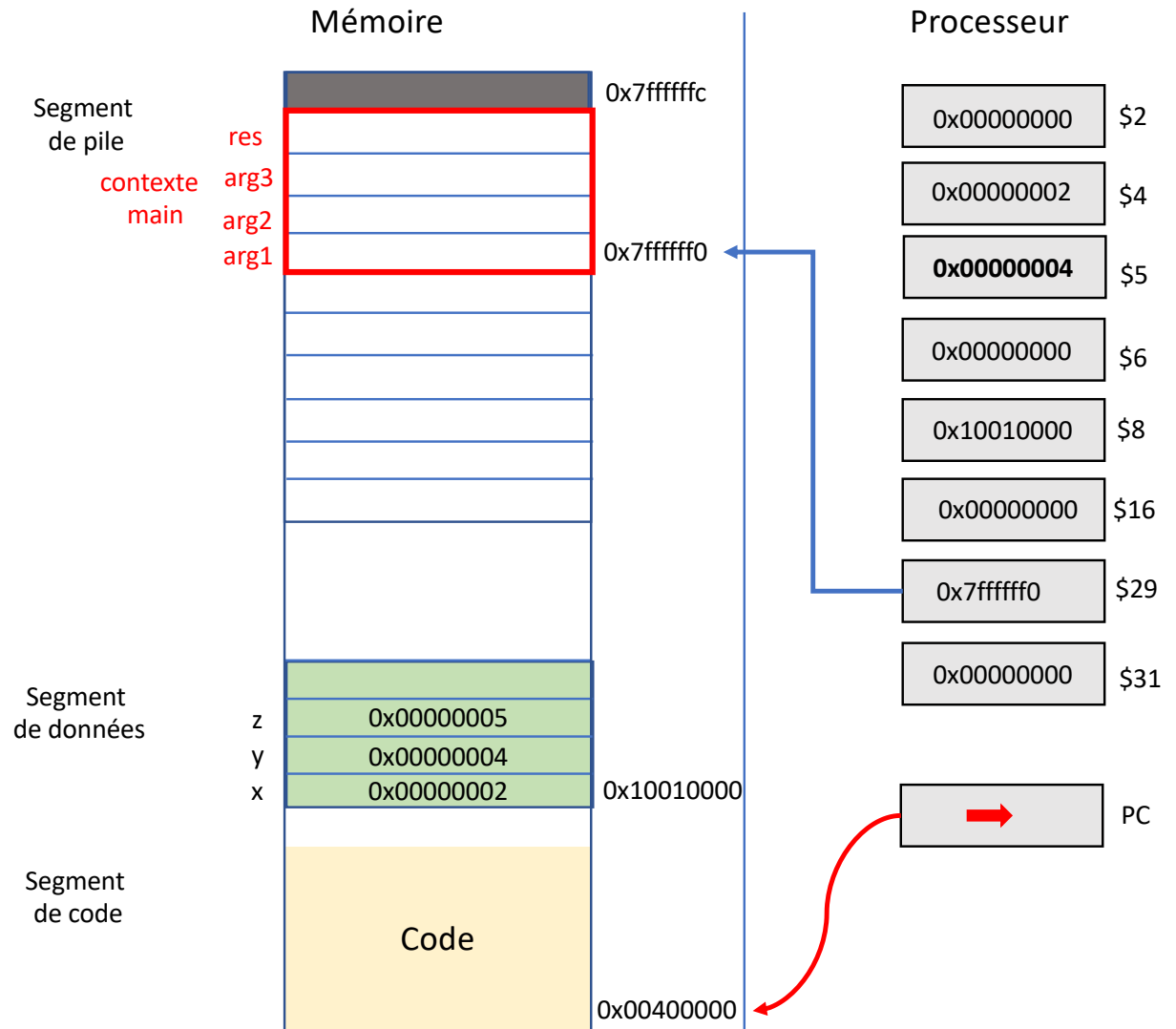
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)
sw $6, 24($29)         # sauvegarde 3eme arg (c)
jal max2
ori $4, $2, 0          # 1er arg=tmp
lw $5, 24($29)         # 2eme arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

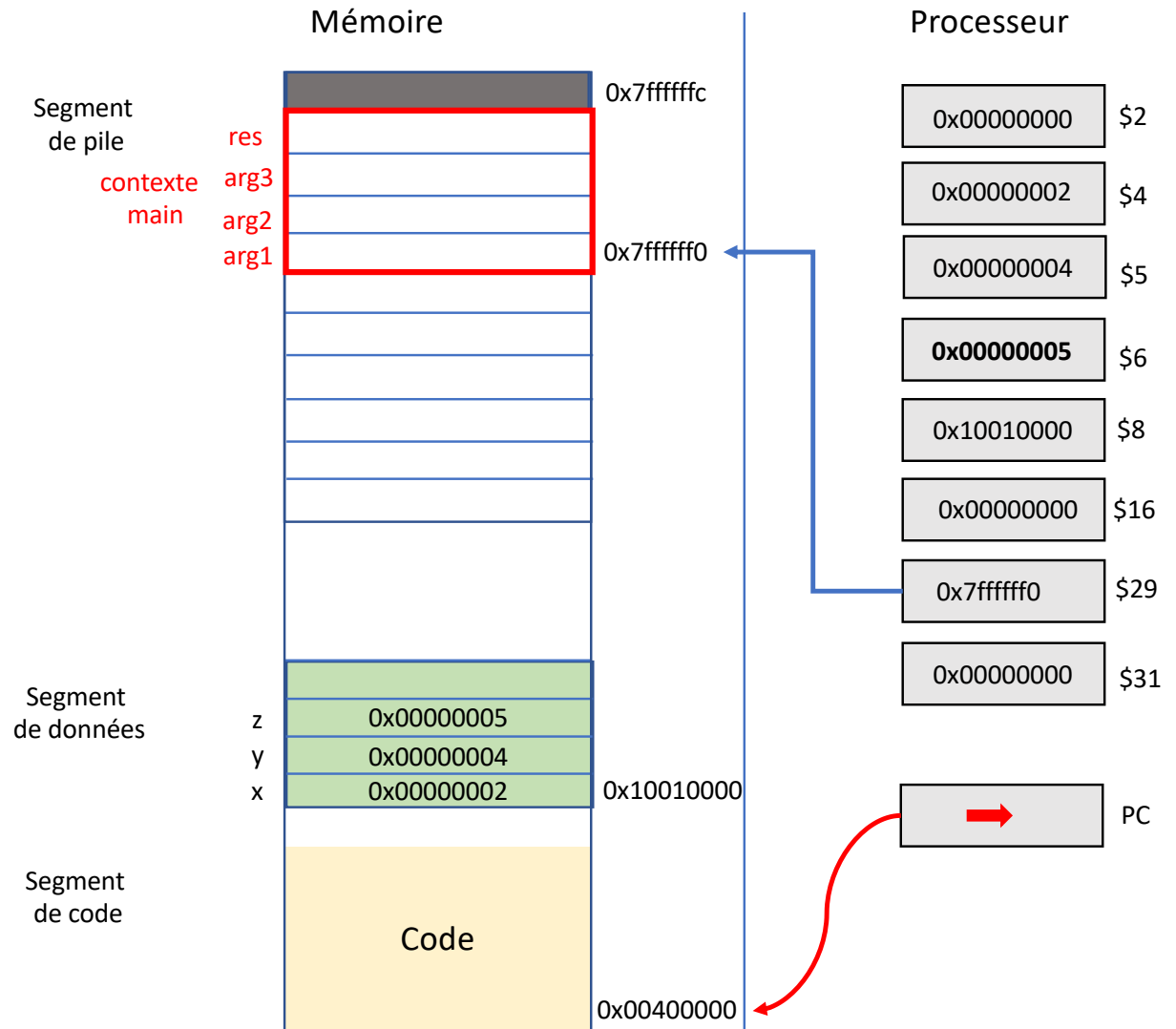
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)
sw $6, 24($29)         # sauvegarde 3ème arg (c)
jal max2
ori $4, $2, 0          # 1er arg=tmp
lw $5, 24($29)         # 2ème arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
→ jal max3             # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)
sw $6, 24($29)         # sauvegarde 3ème arg (c)
jal max2
ori $4, $2, 0          # 1er arg=tmp
lw $5, 24($29)         # 2ème arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```

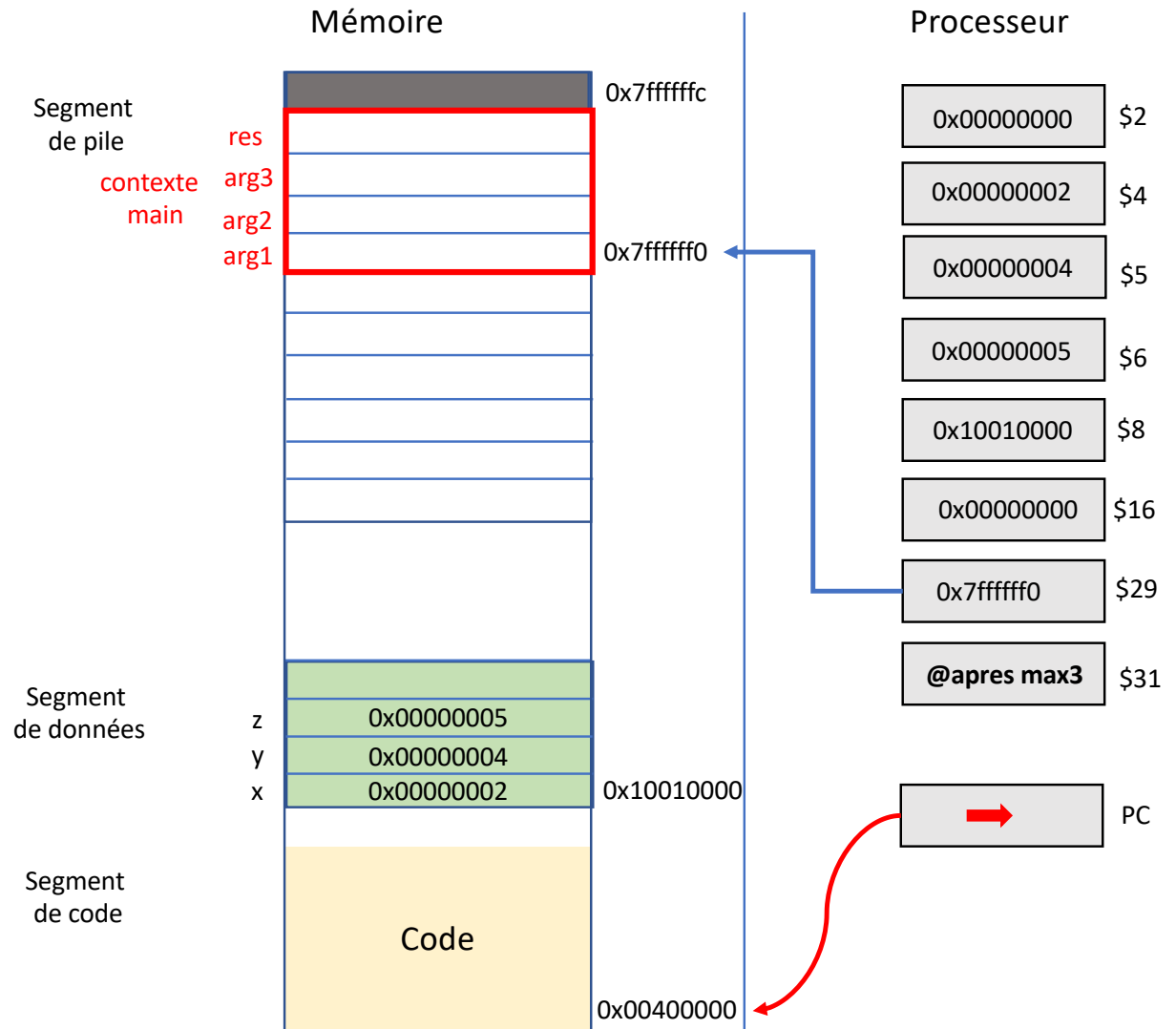


Exécution et évolution de la pile

```

.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3eme arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0          # 1er arg=tmp
lw $5, 24($29)         # 2eme arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31

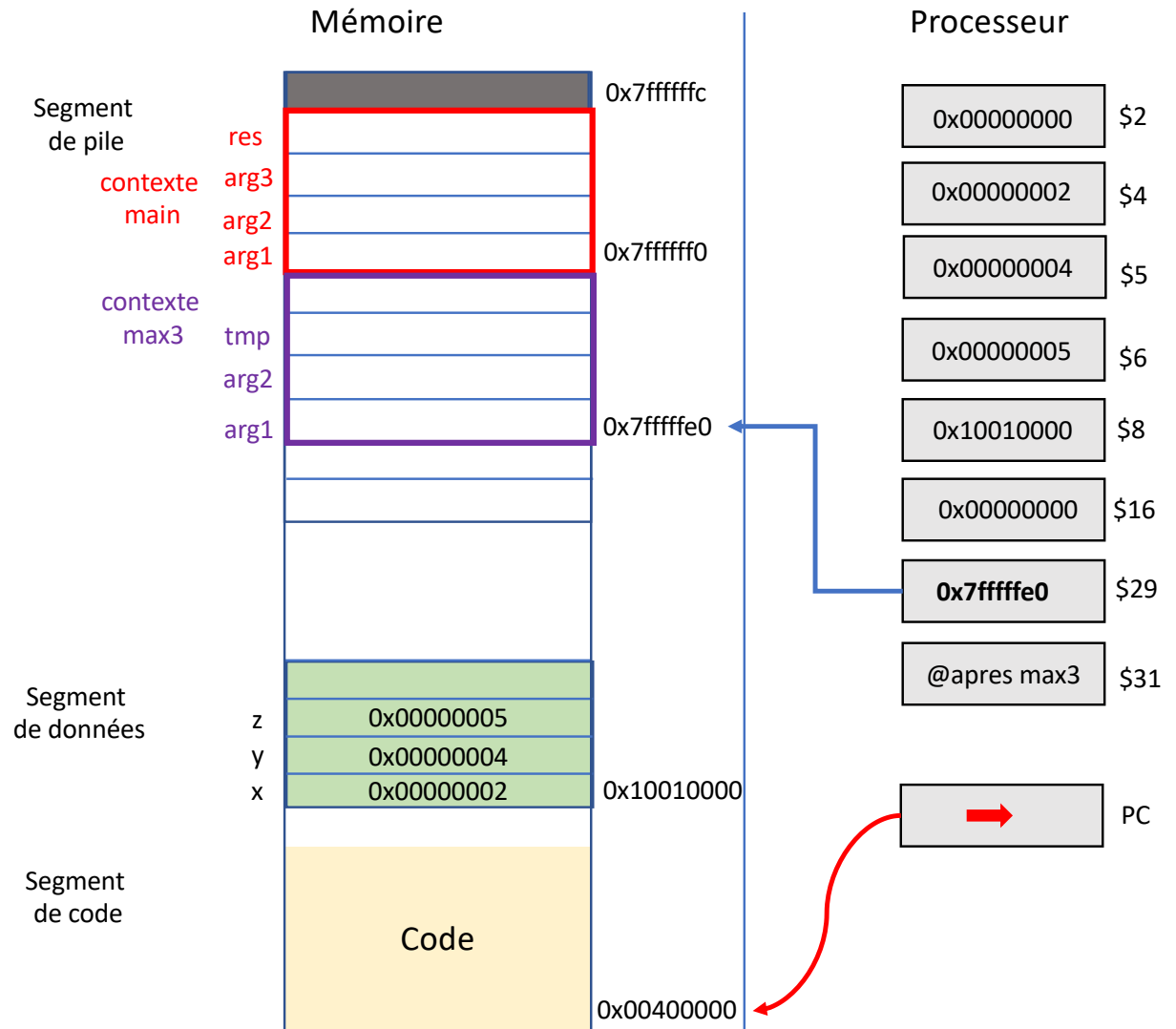
```



Exécution et évolution de la pile

```

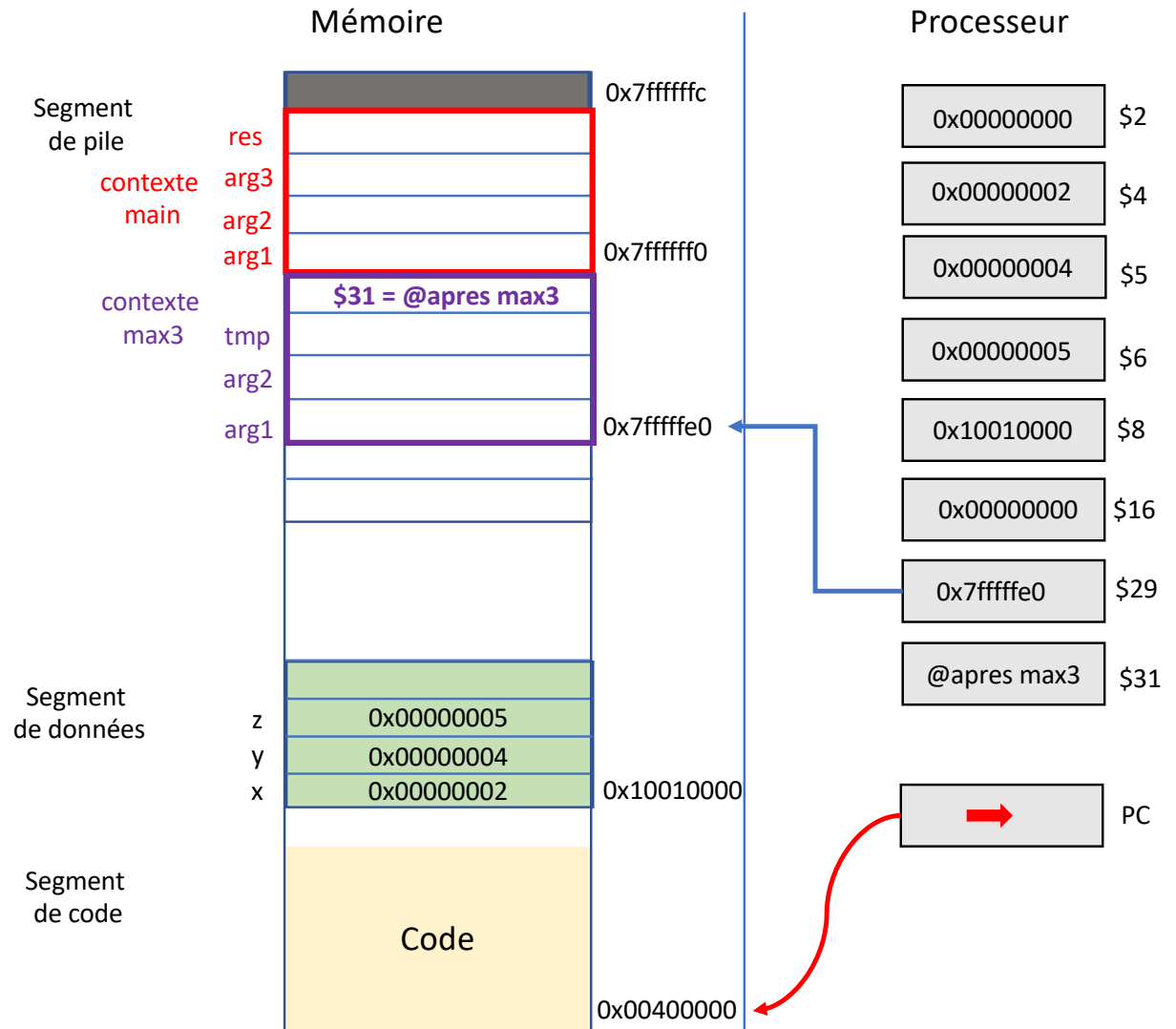
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3ème arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0          # 1er arg=tmp
lw $5, 24($29)         # 2ème arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

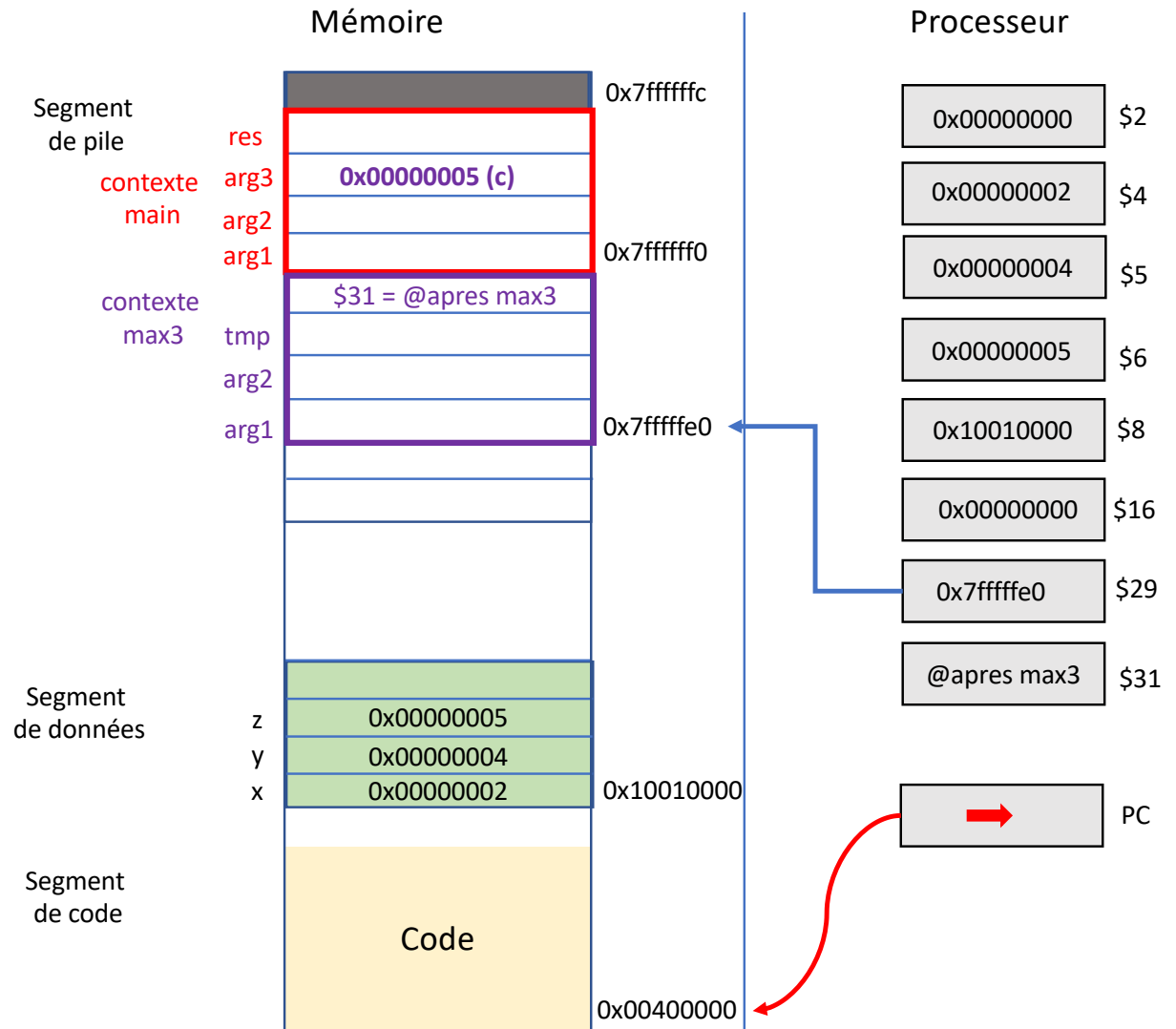
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3ème arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0          # 1er arg=tmp
lw $5, 24($29)         # 2ème arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

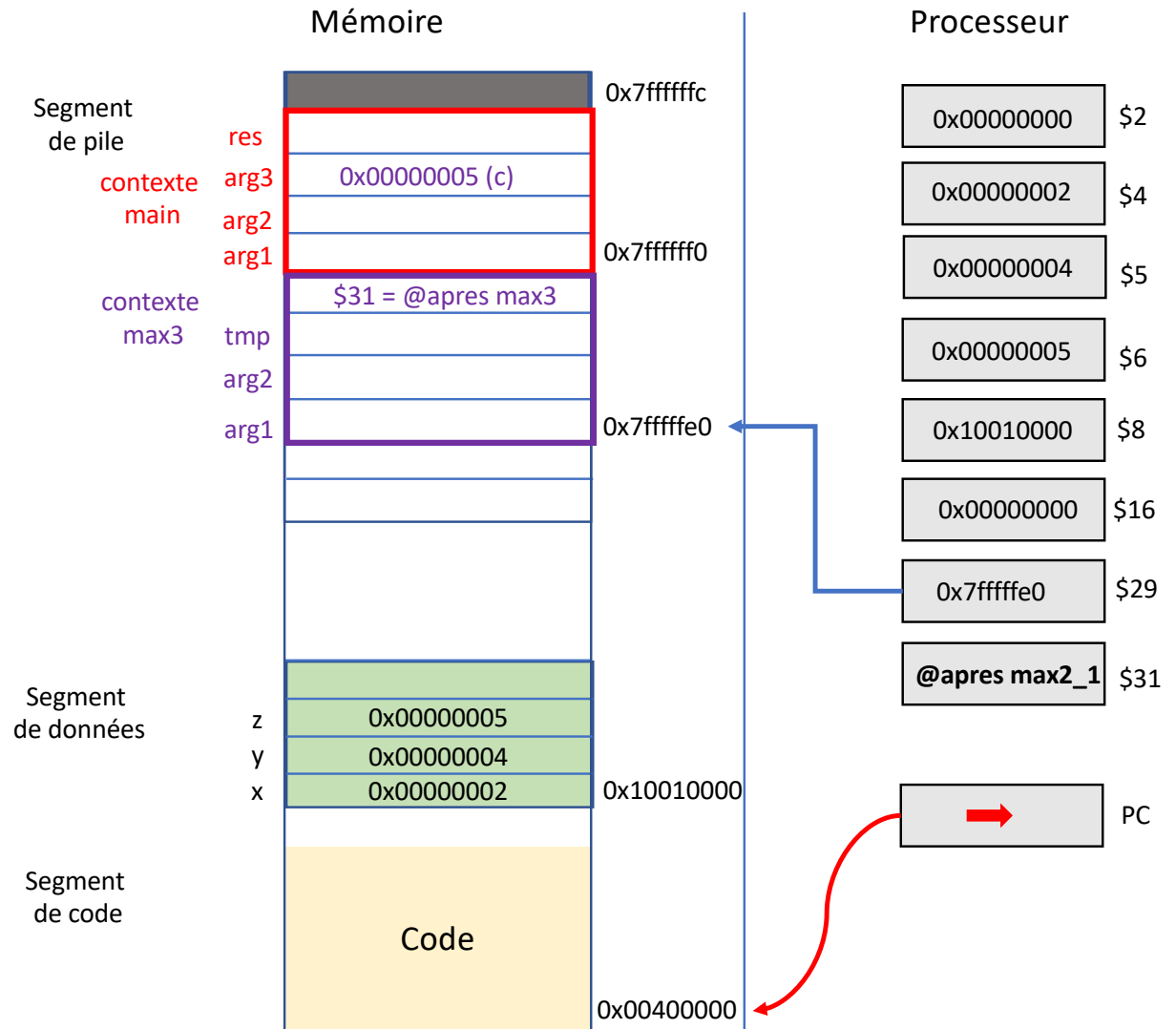
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3ème arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0          # 1er arg=tmp
lw $5, 24($29)         # 2ème arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

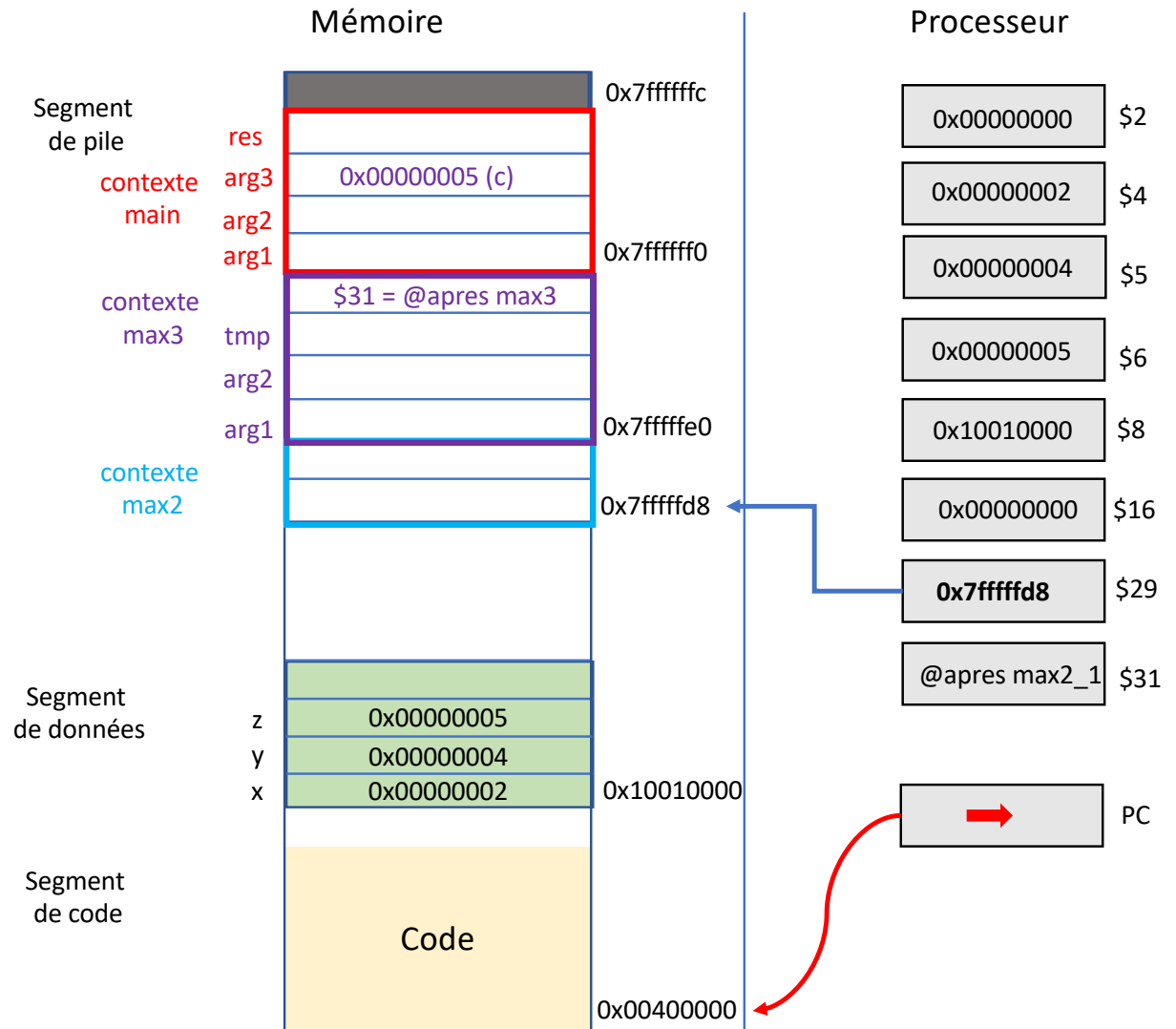
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3ème arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0
lw $5, 24($29)         # 2ème arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

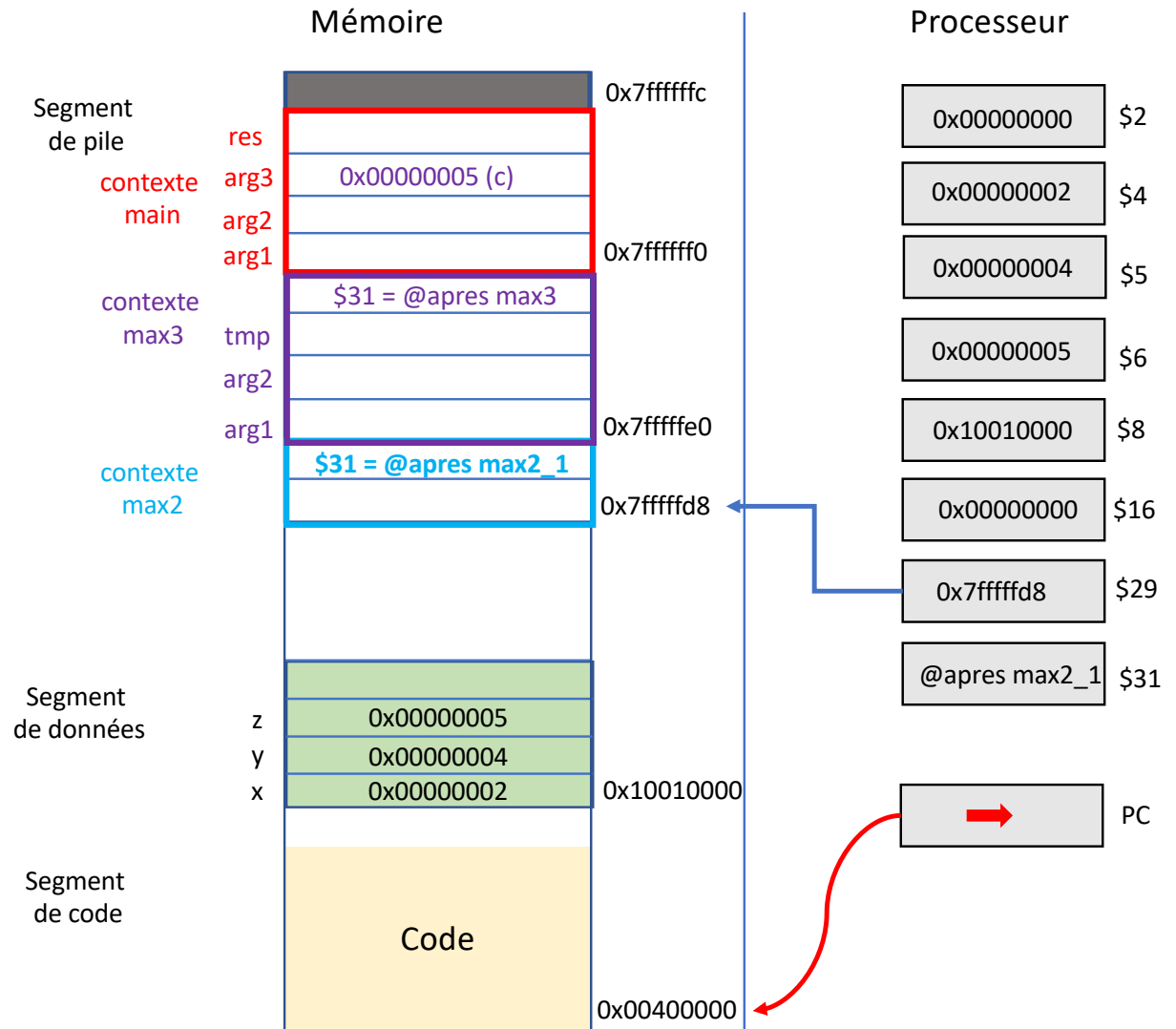
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3eme arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0
lw $5, 24($29)         # 2eme arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

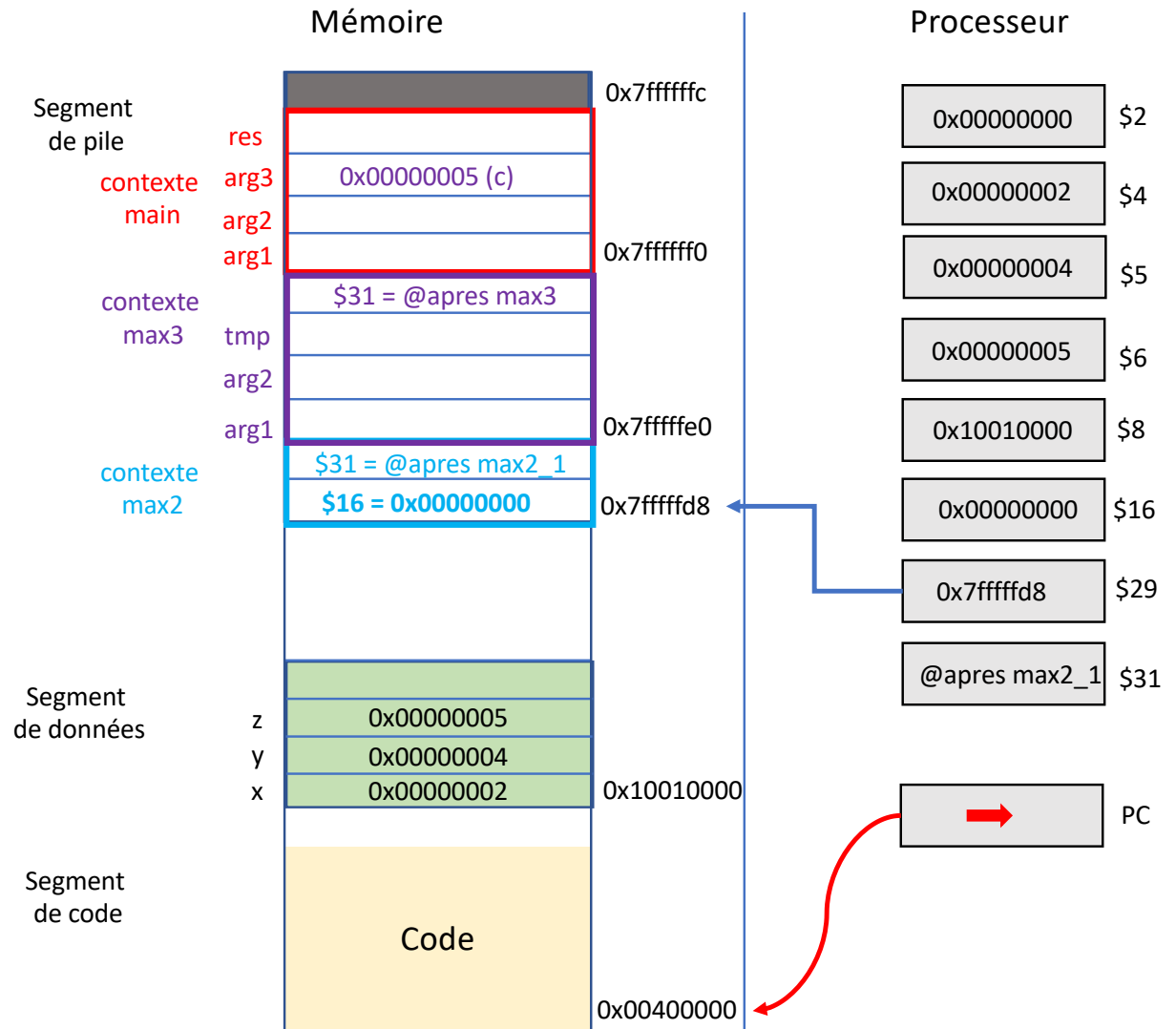
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3eme arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0
lw $5, 24($29)         # 2eme arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

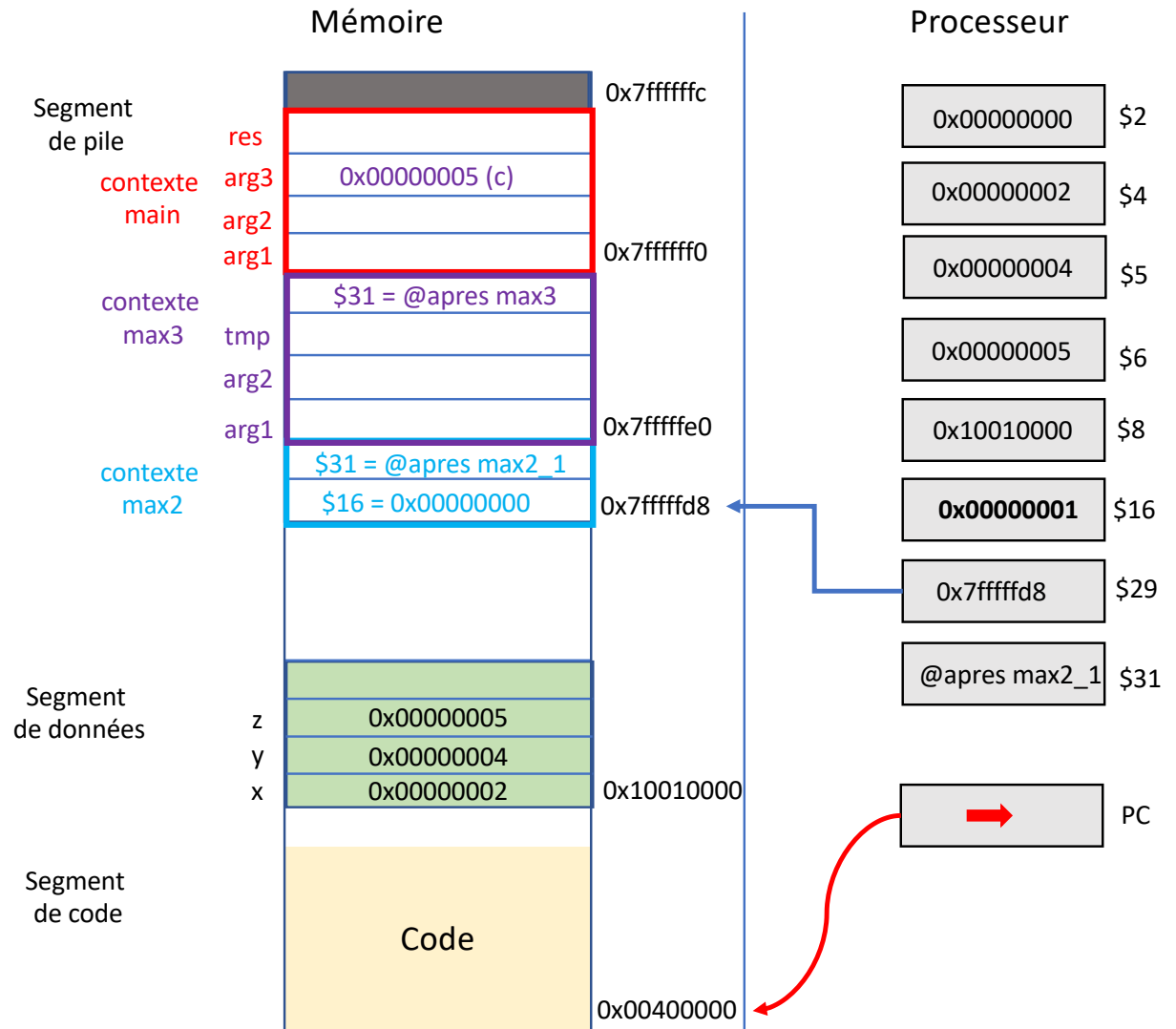
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)
sw $6, 24($29)         # sauvegarde 3eme arg (c)
jal max2
ori $4, $2, 0
lw $5, 24($29)         # 2eme arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

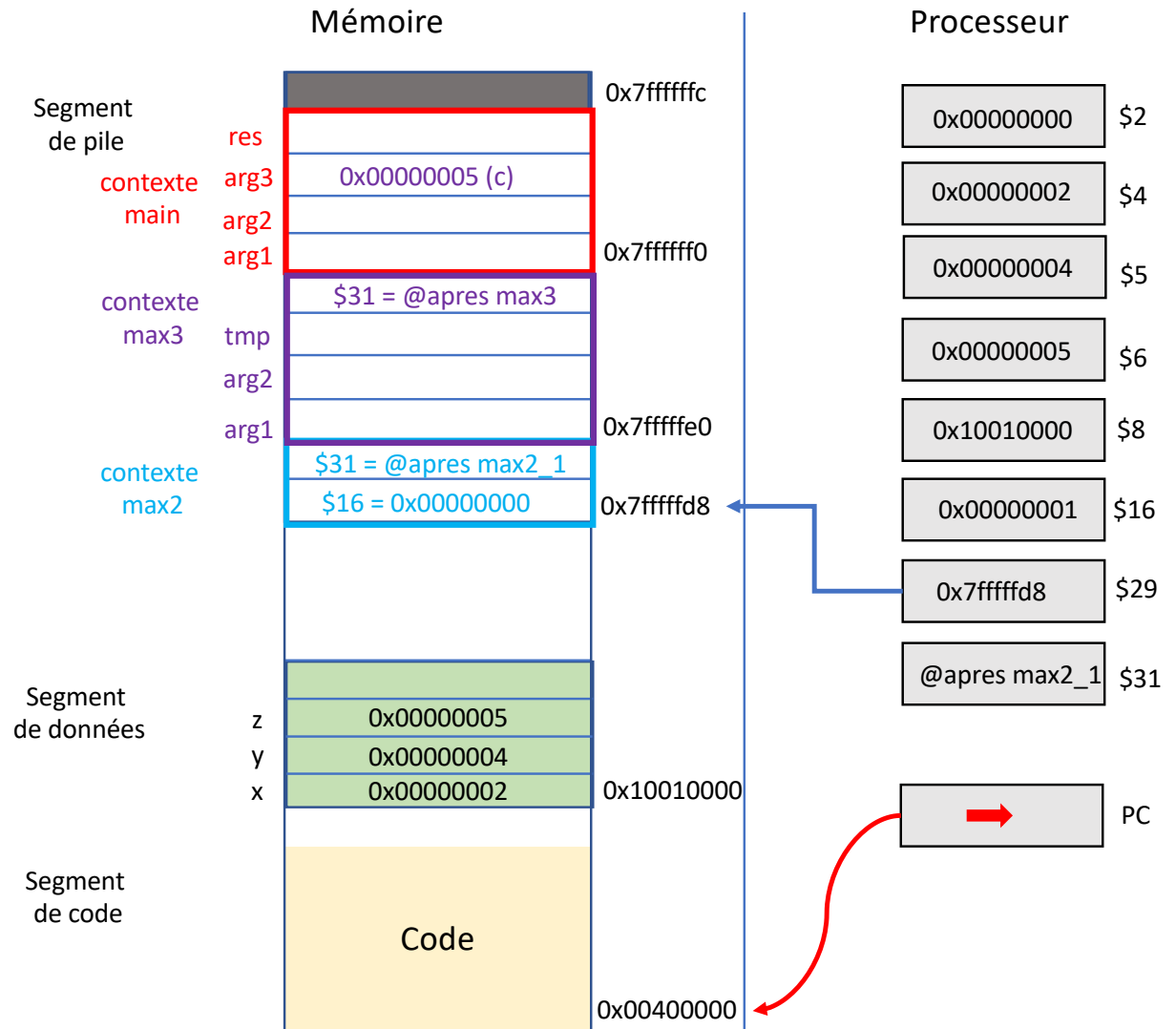
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)
sw $6, 24($29)         # sauvegarde 3eme arg (c)
jal max2
ori $4, $2, 0
lw $5, 24($29)         # 2eme arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $16 vaut 1 si a < b
beq $16, $0, a_max     # valeur de retour dans $2
ori $2, $5, 0
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
    
```



Exécution et évolution de la pile

```

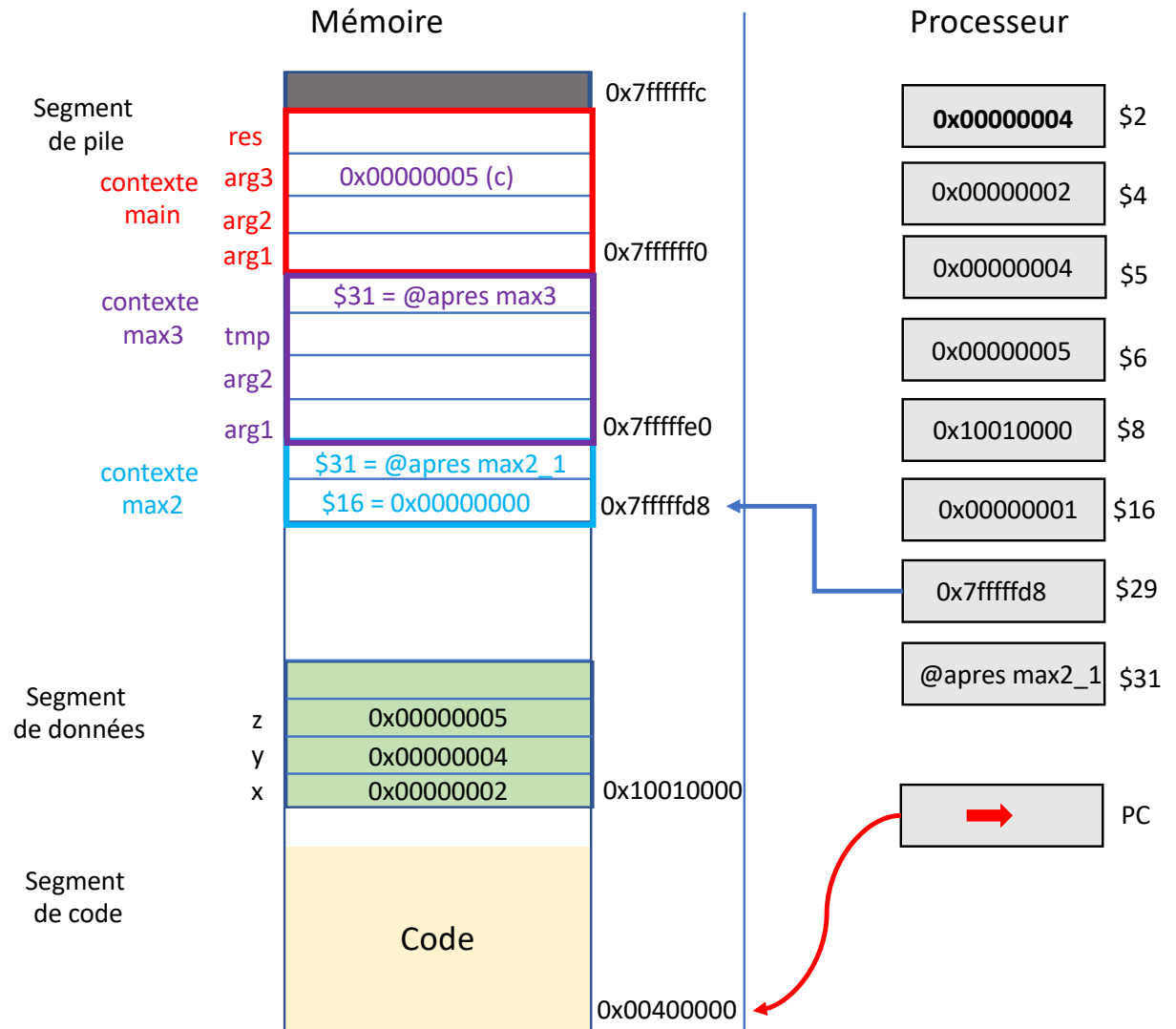
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3eme arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0
lw $5, 24($29)         # 2eme arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

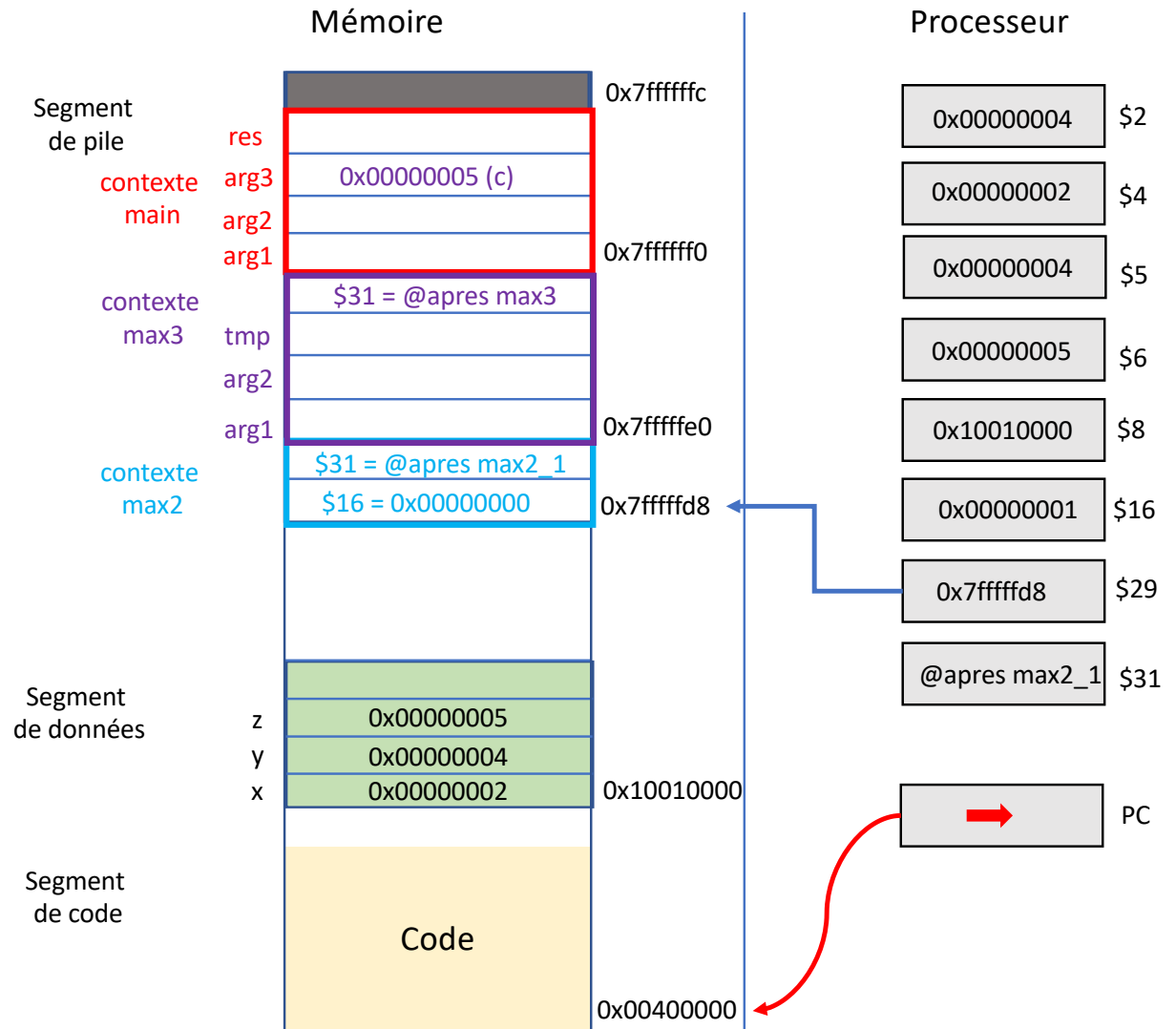
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)
sw $6, 24($29)         # sauvegarde 3ème arg (c)
jal max2
ori $4, $2, 0
lw $5, 24($29)         # 2ème arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

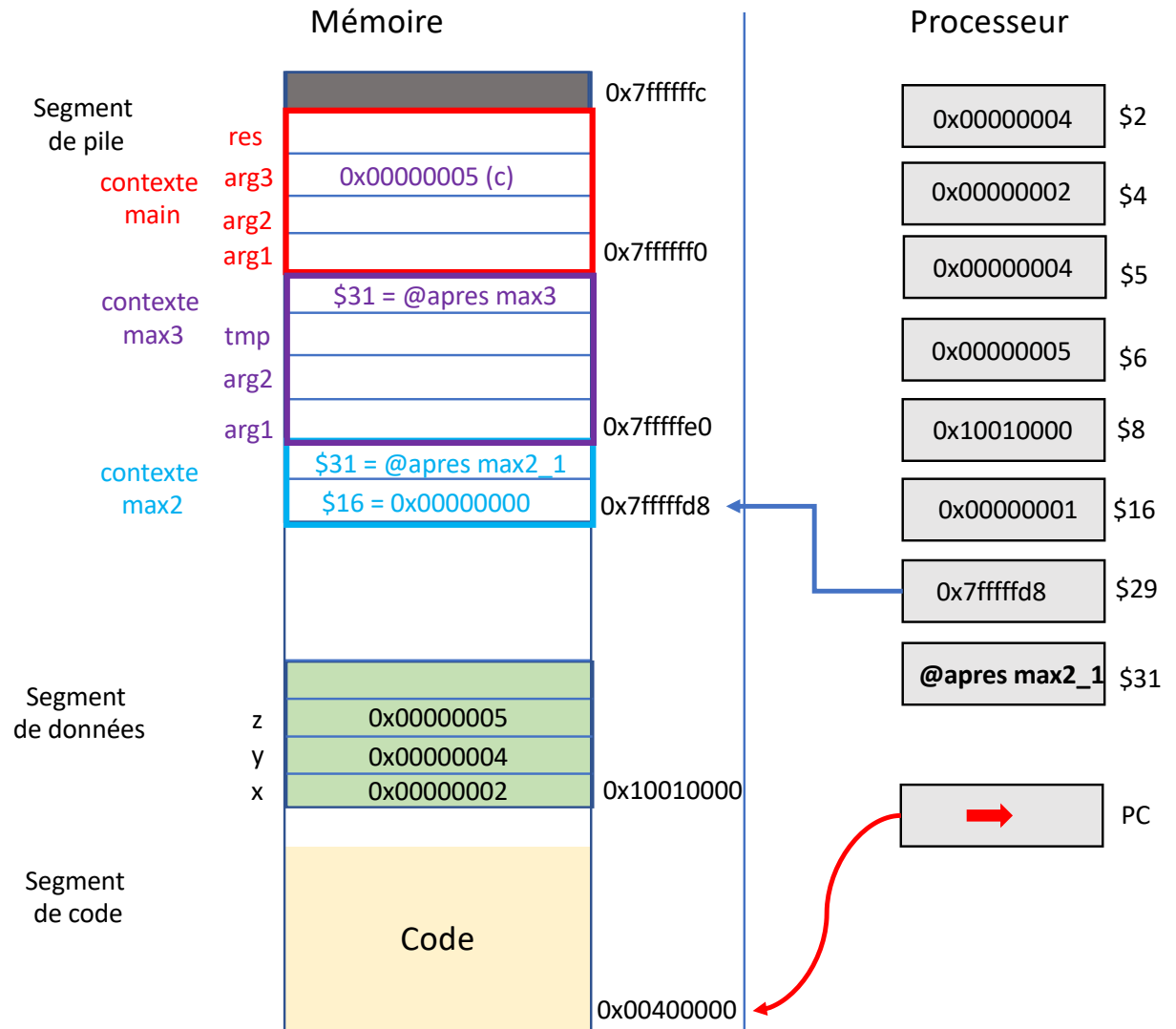
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3eme arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0
lw $5, 24($29)         # 2eme arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

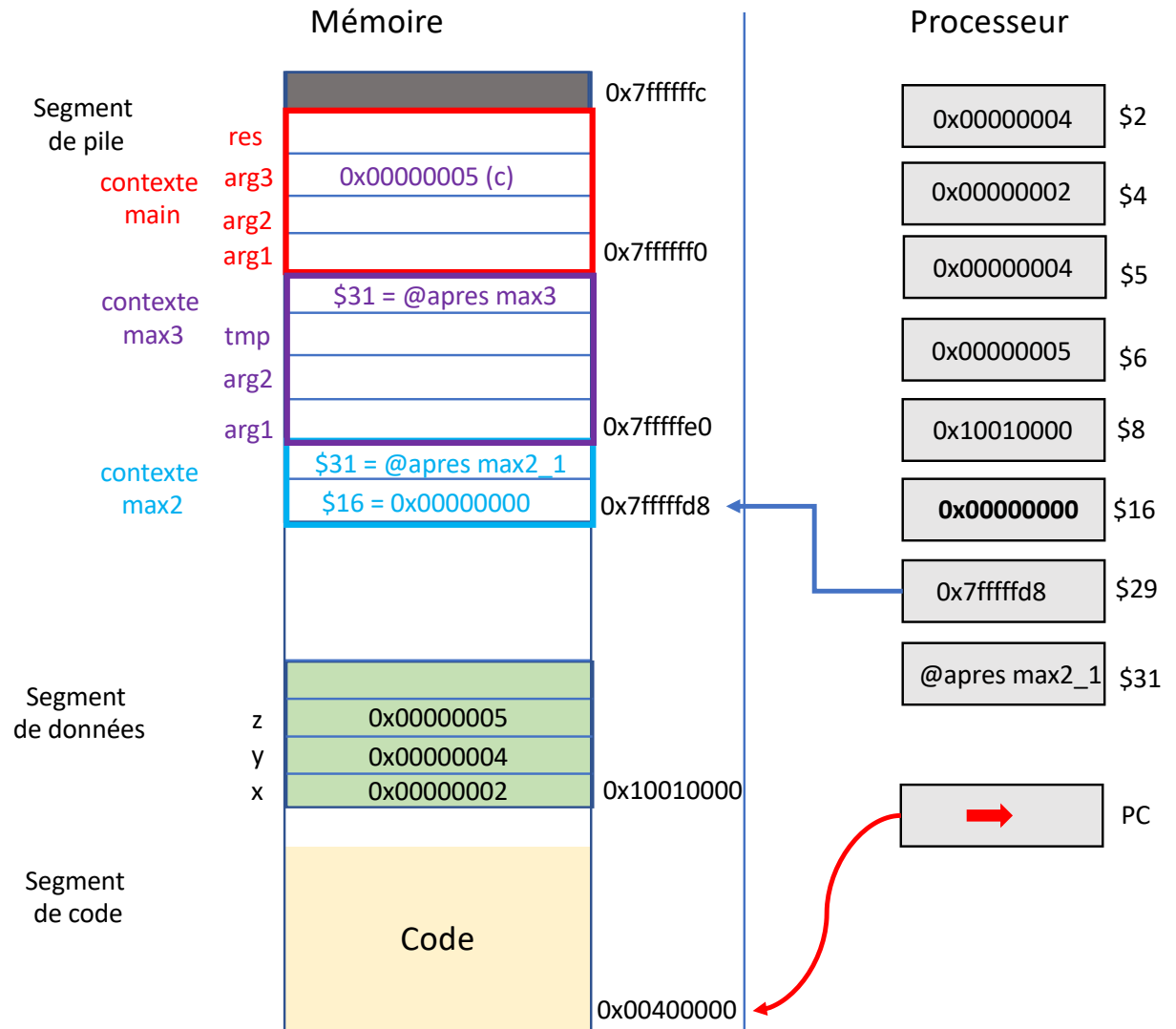
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3eme arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0
lw $5, 24($29)         # 2eme arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3eme arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0          # 1er arg=tmp
lw $5, 24($29)         # 2eme arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

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

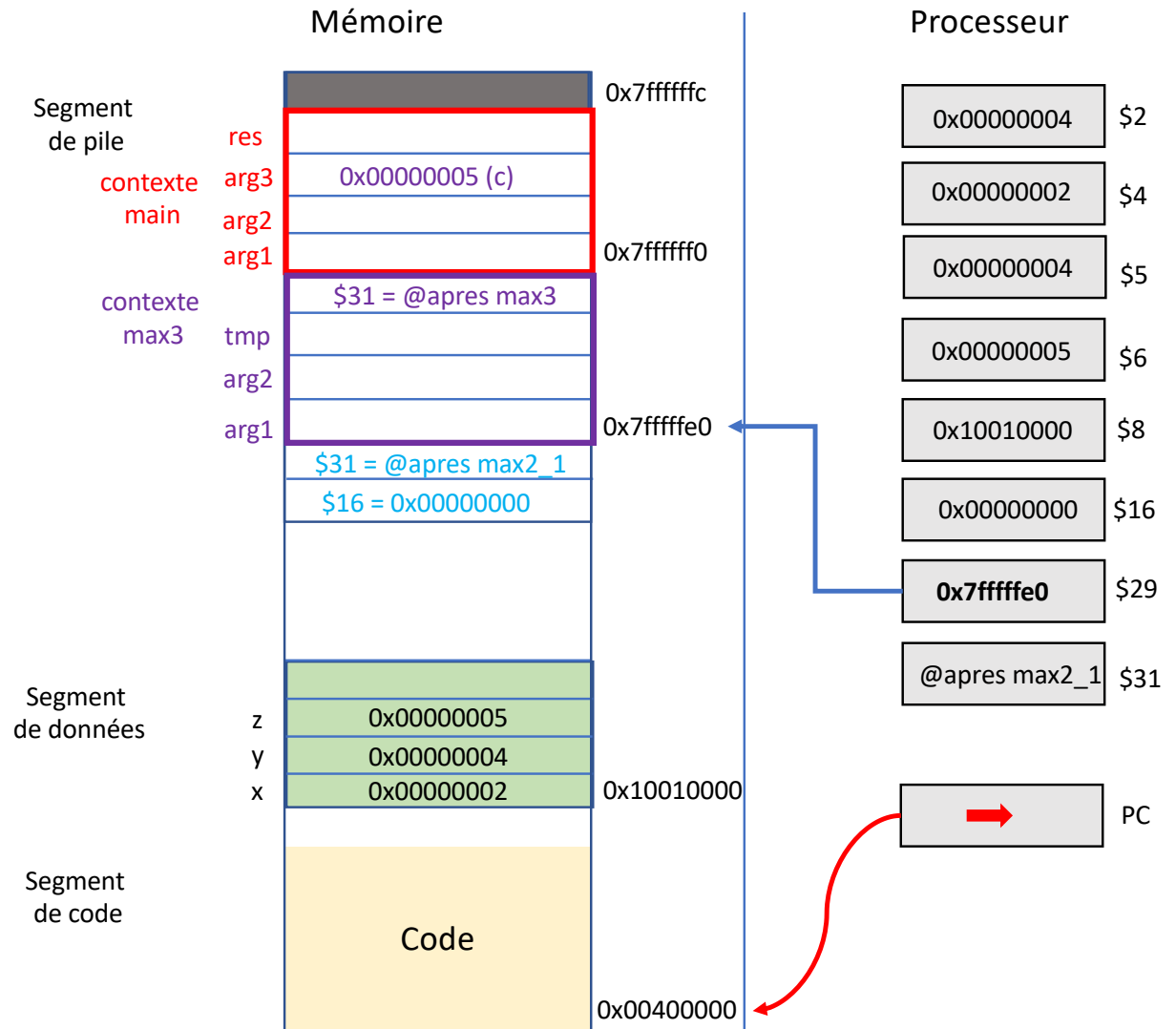
.text
addiu $29, $29, -16      # 1 var locale + 3 param max
lui   $8, 0x1001
lw    $4, 0($8)          # 1er arg = x
lw    $5, 4($8)          # 2ème arg = y
lw    $6, 8($8)          # 3ème arg = z
jal   max3               # appel à max3
ori   $4, $2, 0
ori   $2, $0, 1          # affichage
syscall
addiu $29, $29, +16      # désallocation pile
ori   $2, $0, 10
syscall

max3:  #-----
addiu $29, $29, -16      # nr = 0 + $31, nv = 1, na = 2
sw    $31, 12($29)
sw    $6, 24($29)        # sauvegarde 3eme arg (c)
jal   max2
ori   $4, $2, 0          # 1er arg=tmp
lw    $5, 24($29)        # 2eme arg = c
jal   max2
lw    $31, 12($29)
addiu $29, $29, +16
jr    $31

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        # $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:
lw    $31, 4($29)
lw    $16, 0($29)
addiu $29, $29, +8
jr    $31

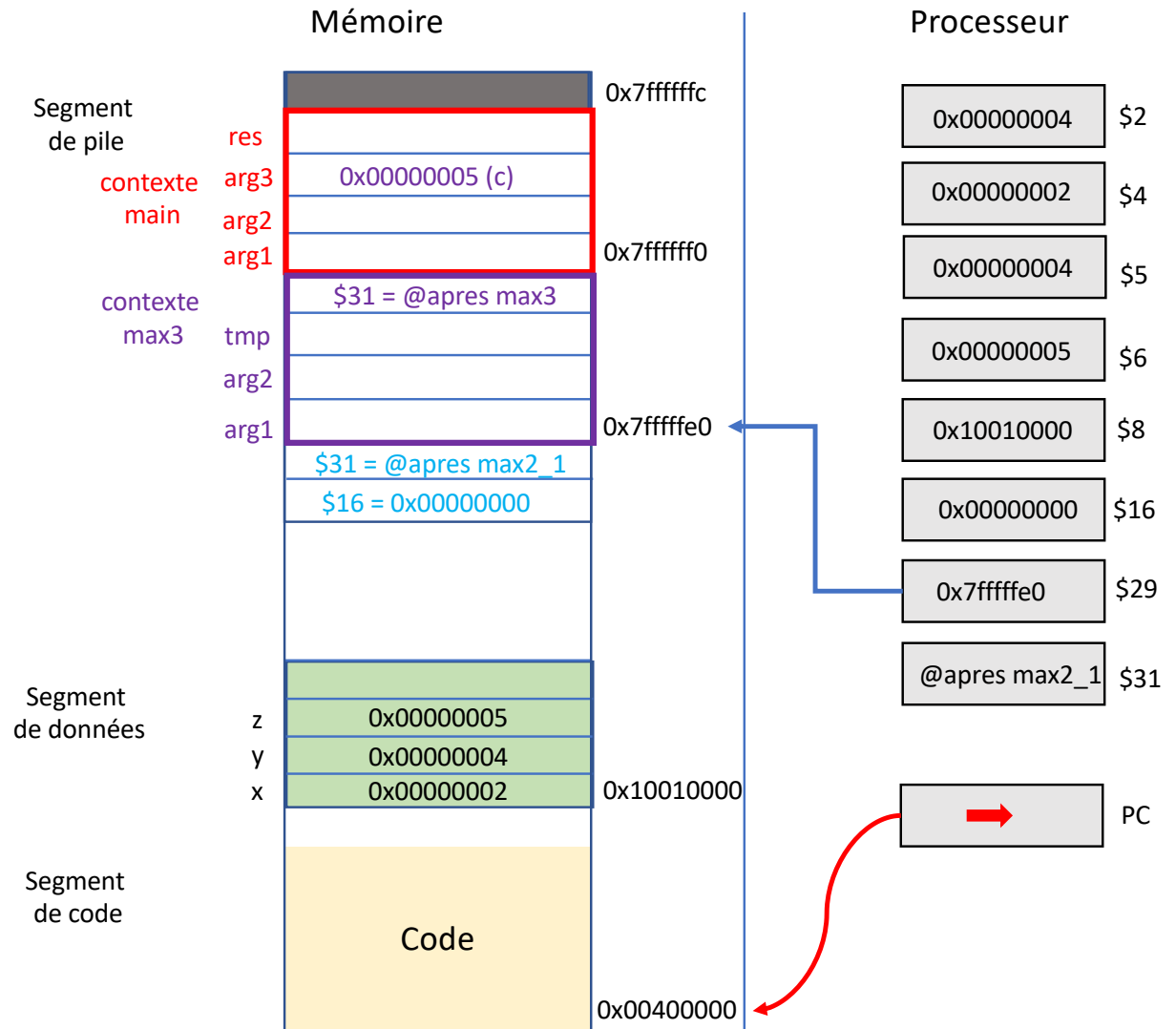
```



Exécution et évolution de la pile

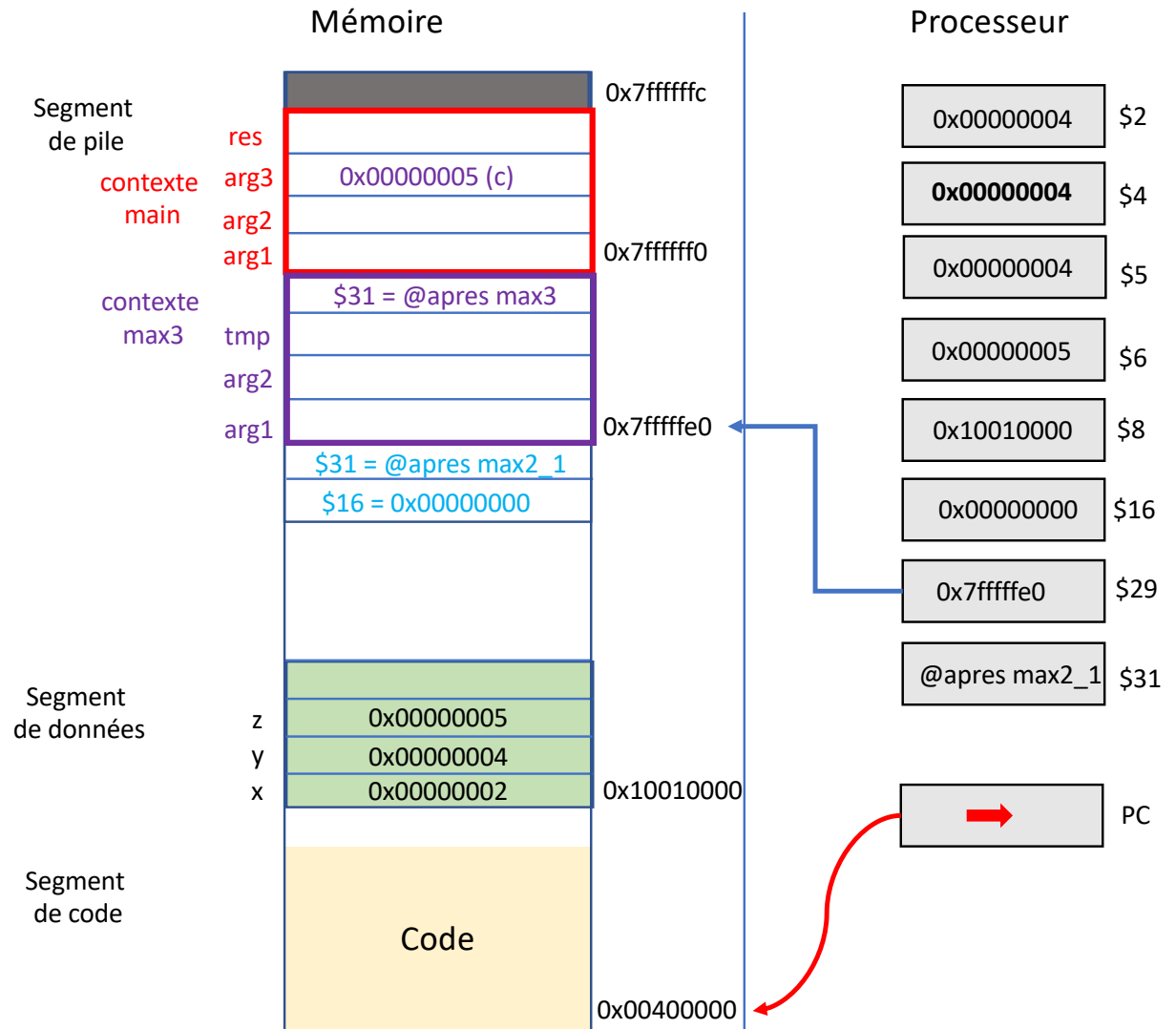
```

.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3ème arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0          # 1er arg=tmp
lw $5, 24($29)         # 2ème arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

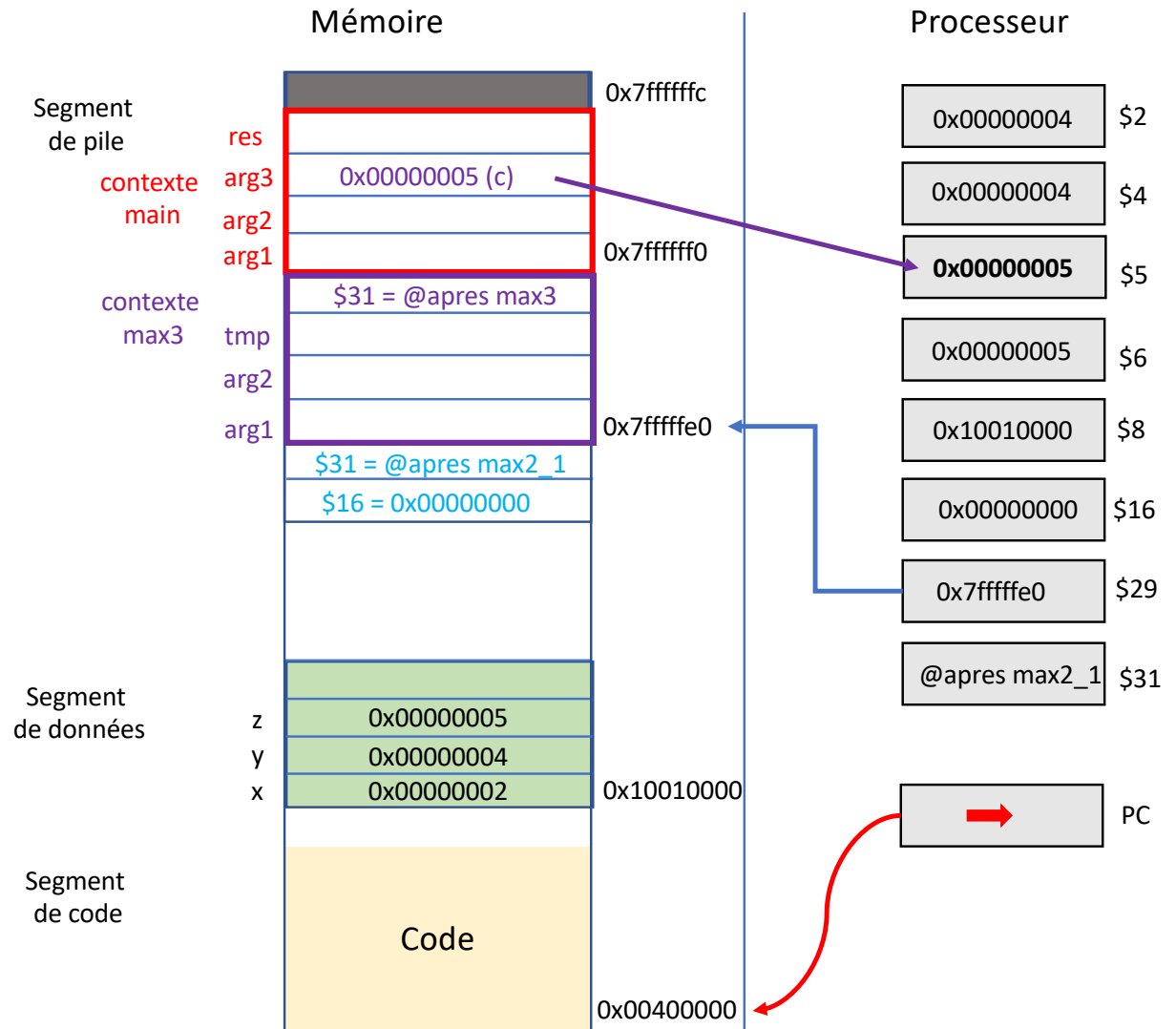
```
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3ème arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0
lw $5, 24($29)         # 2ème arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
```



Exécution et évolution de la pile

```

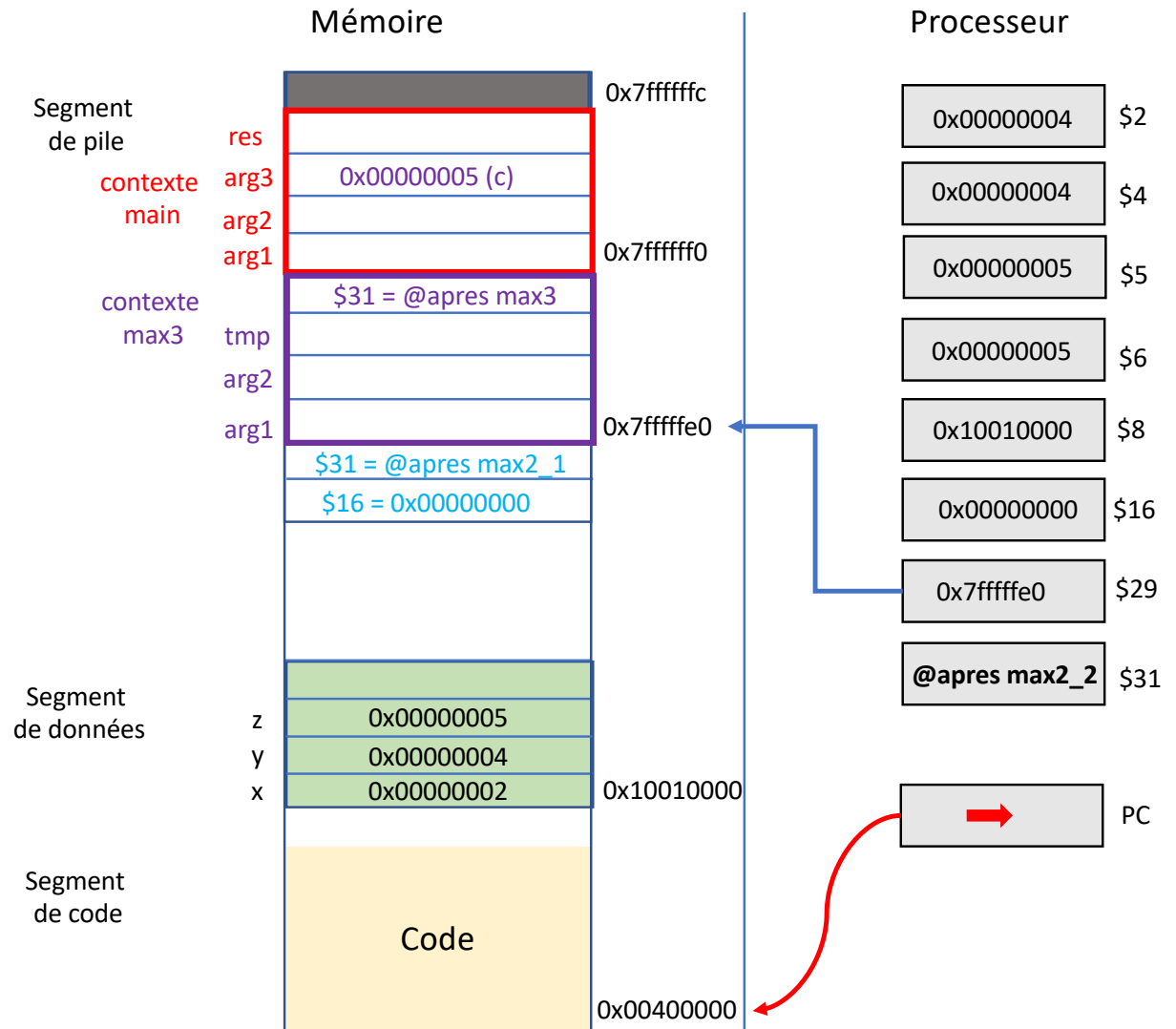
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3ème arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0
lw $5, 24($29)         # 2ème arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

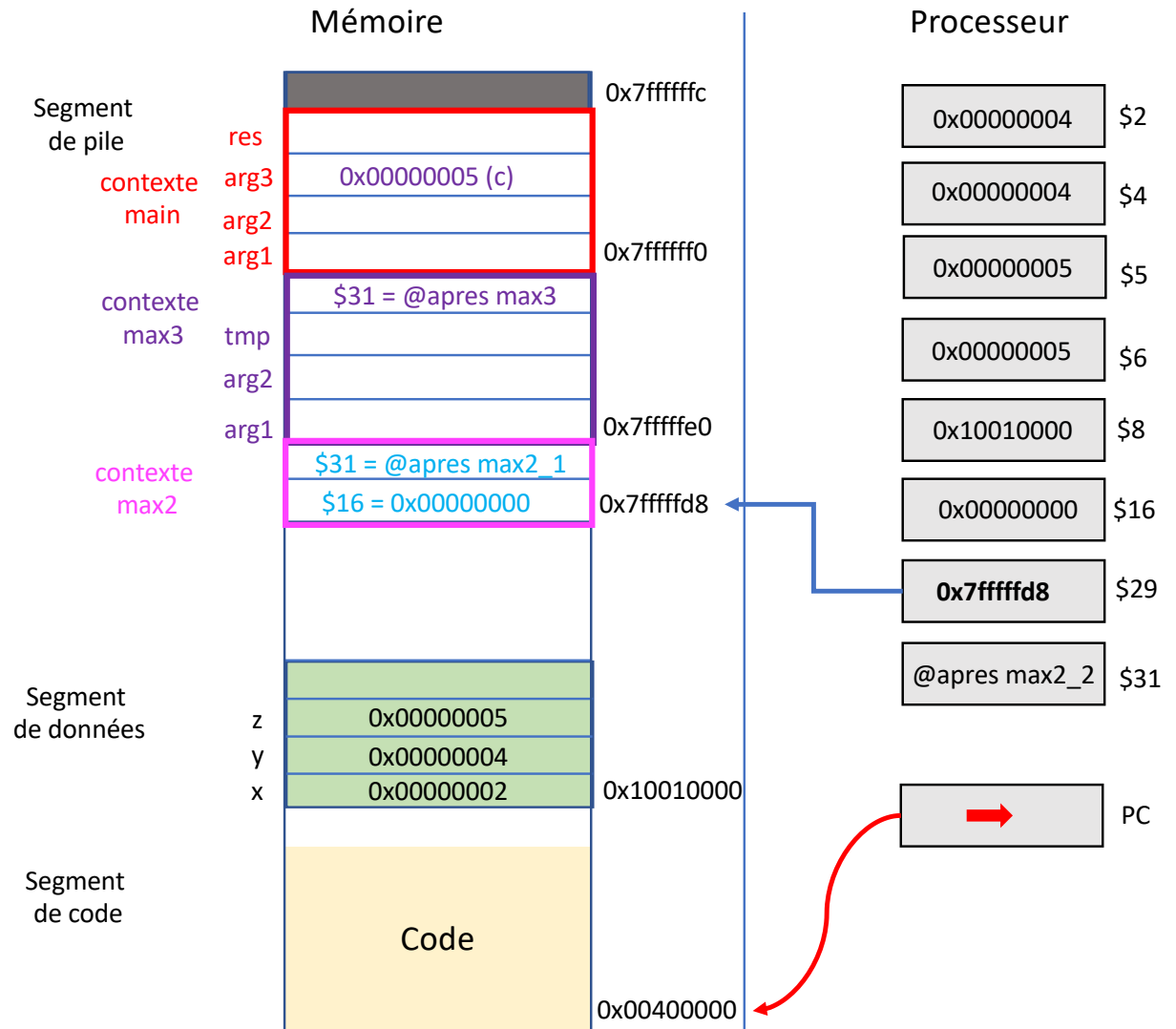
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3ème arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0
lw $5, 24($29)         # 2ème arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

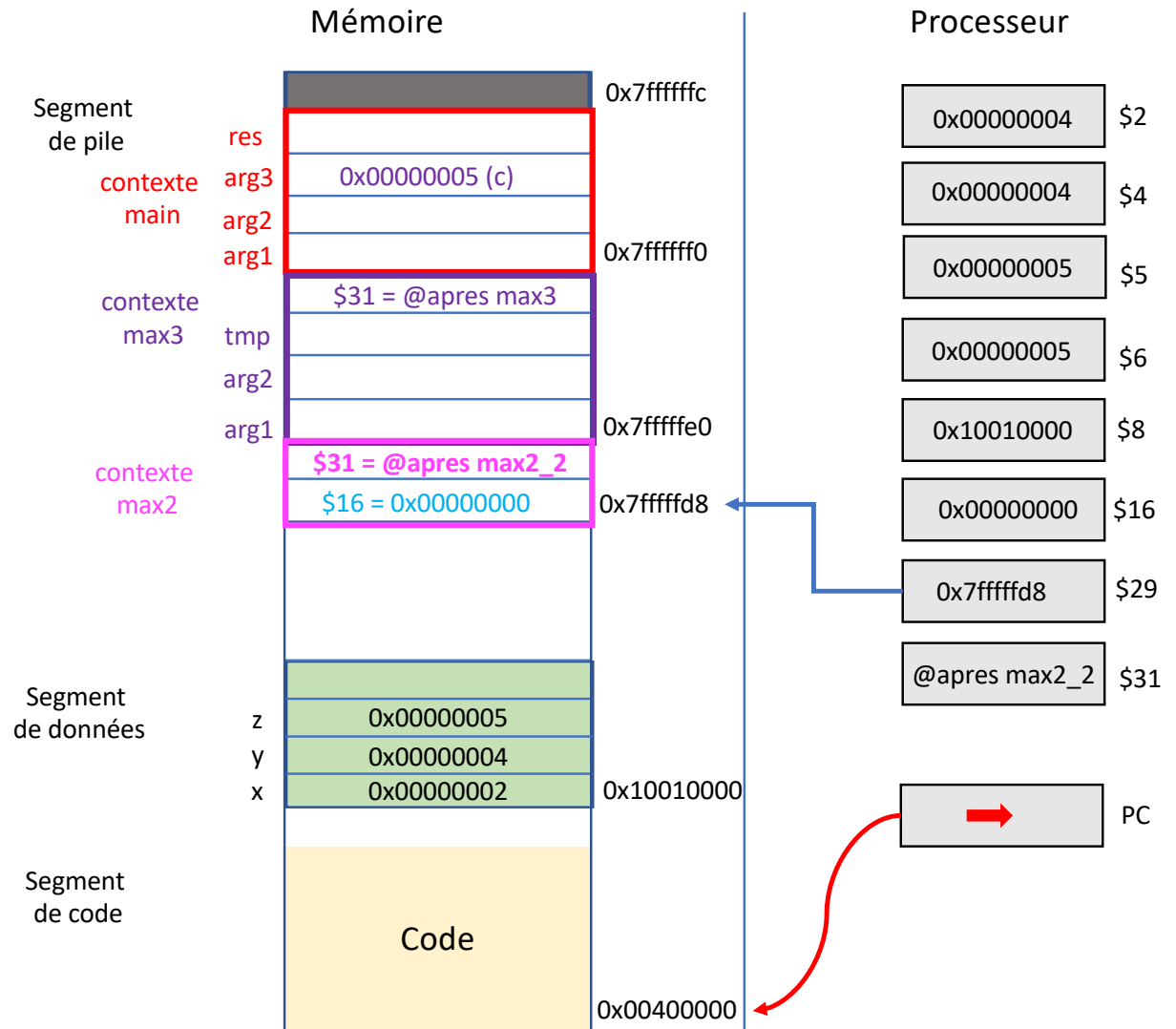
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3eme arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0
lw $5, 24($29)         # 2eme arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

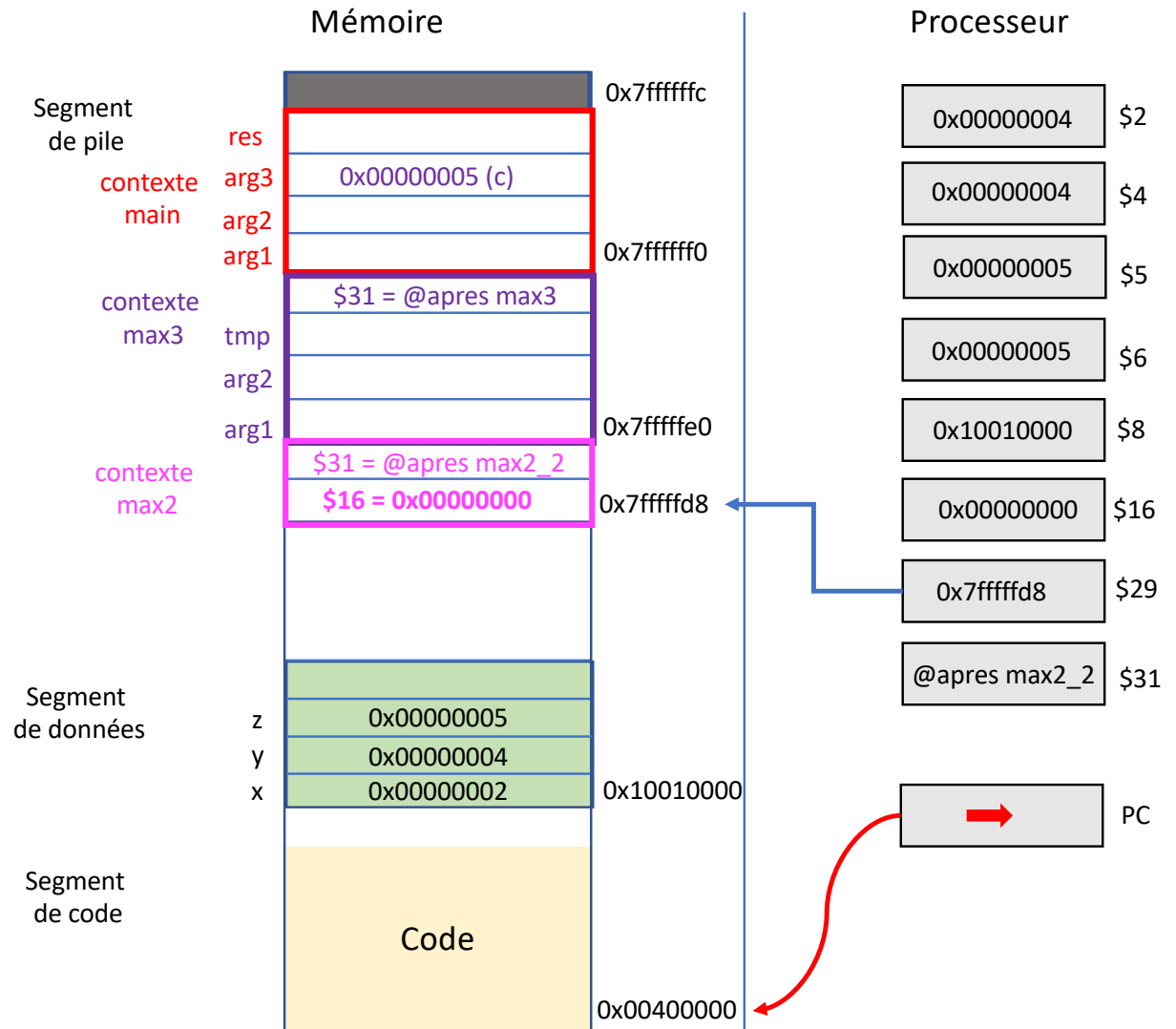
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3ème arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0
lw $5, 24($29)         # 2ème arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
max2: #-----
addiu $29, $29, -8     # nr = 1 ($16) + $31, nv = 0, max_arg = 0
sw $31, 4($29)
sw $16, 0($29)         # $16 vaut 1 si a < b
slt $16, $4, $5
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
    
```



Exécution et évolution de la pile

```

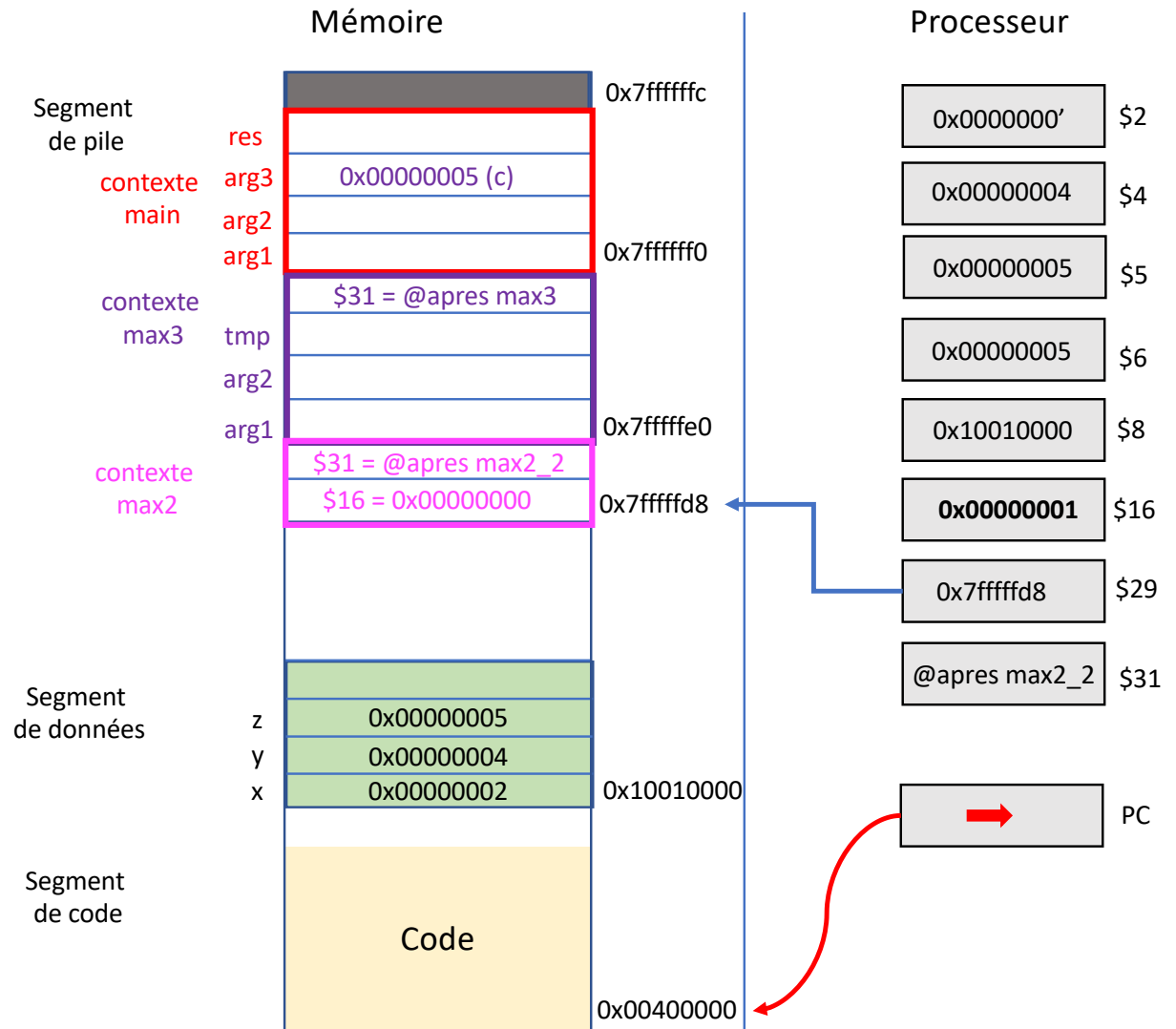
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3eme arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0
lw $5, 24($29)         # 2eme arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

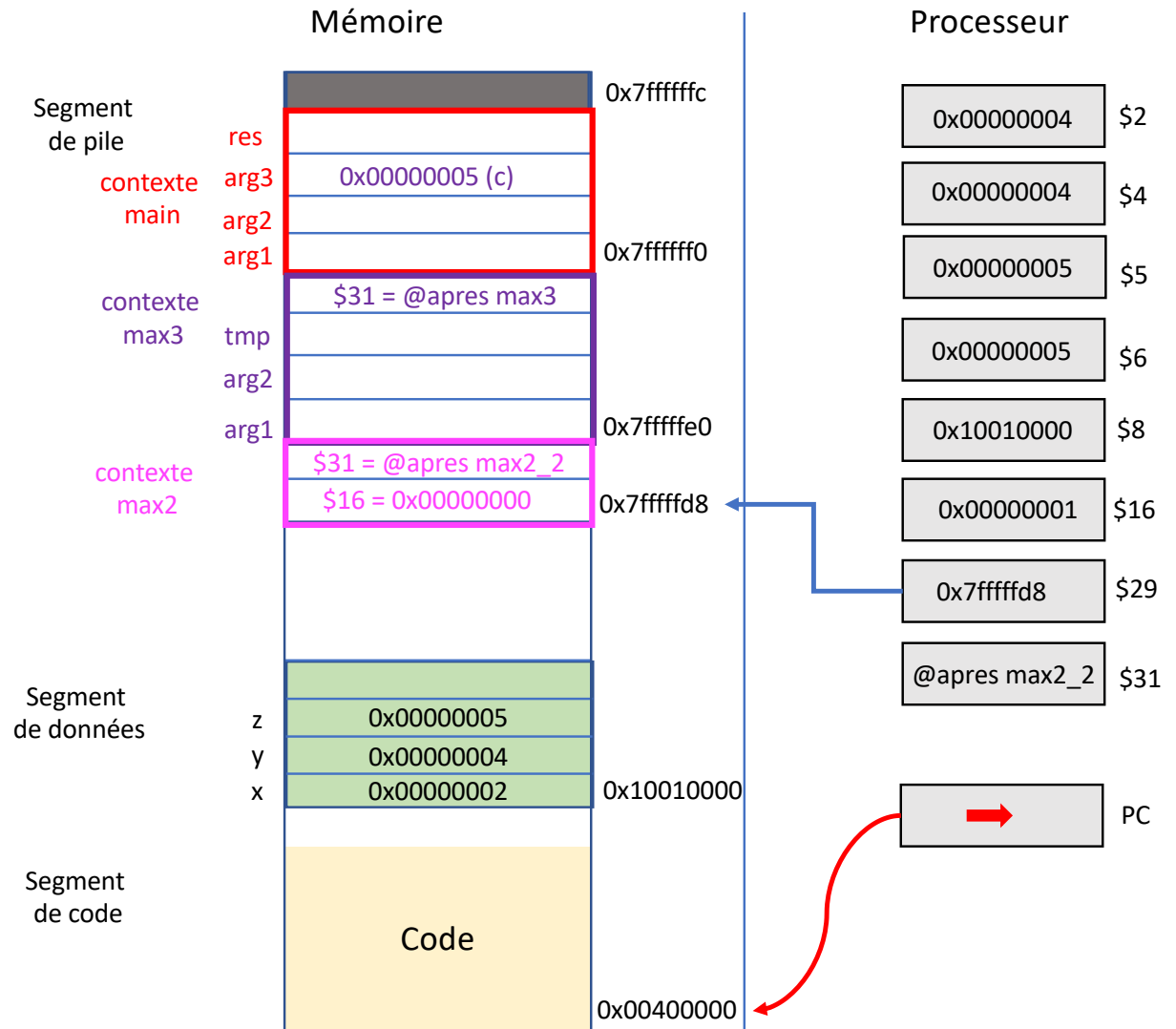
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3ème arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0          # 1er arg=tmp
lw $5, 24($29)         # 2ème arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $16 vaut 1 si a < b
beq $16, $0, a_max     # valeur de retour dans $2
ori $2, $5, 0
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
    
```



Exécution et évolution de la pile

```

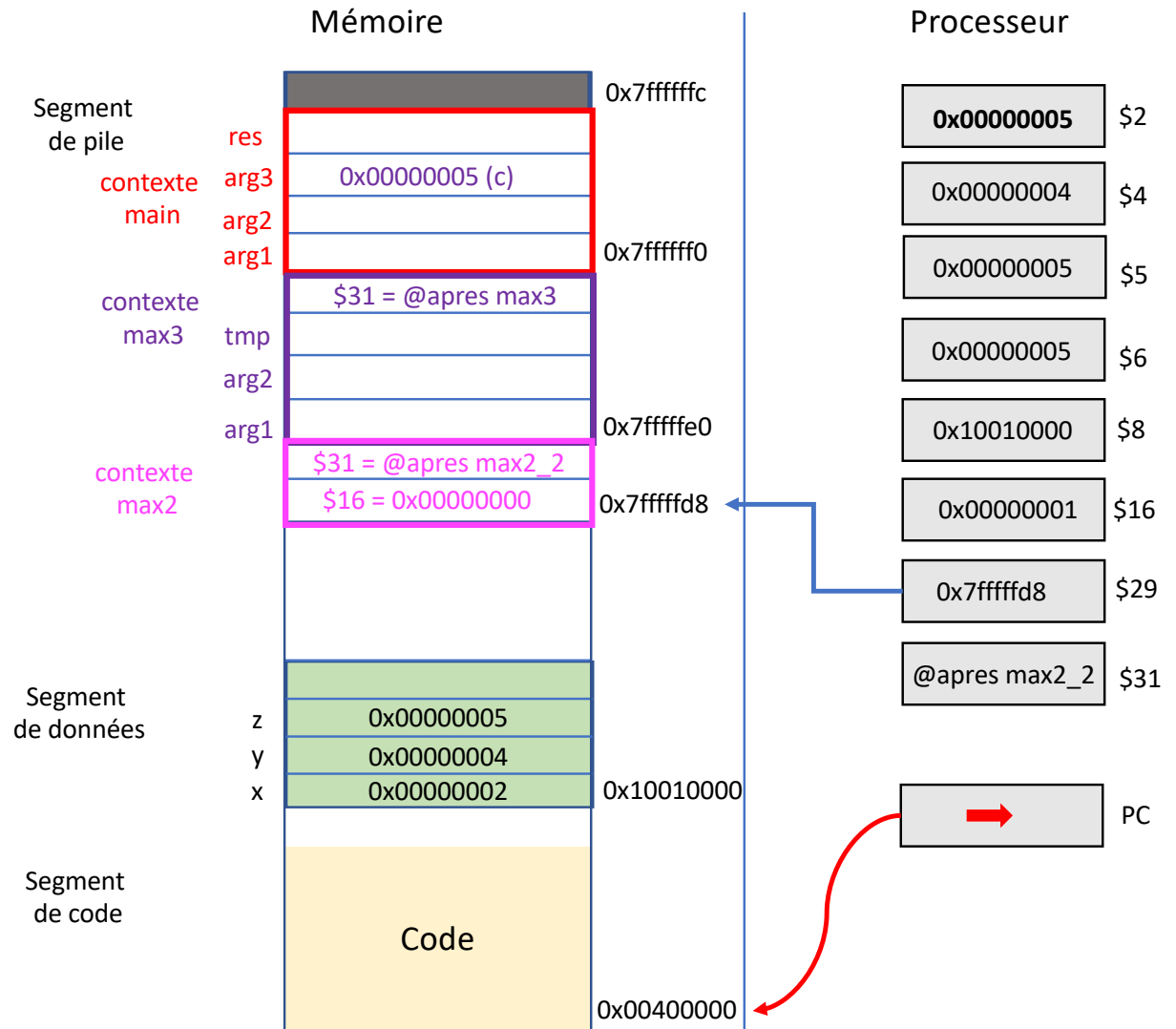
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3eme arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0
lw $5, 24($29)         # 2eme arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

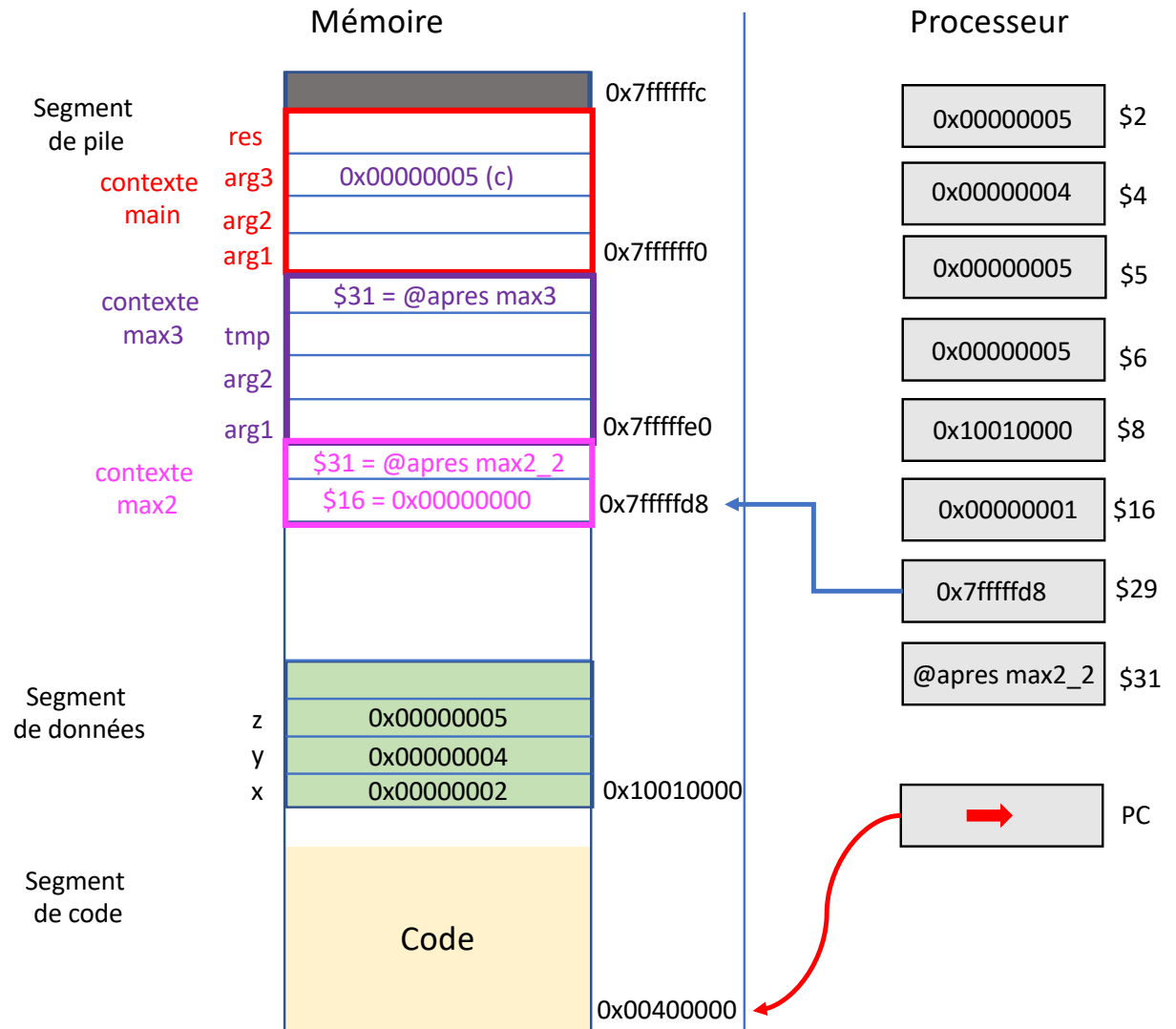
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3eme arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0          # 1er arg=tmp
lw $5, 24($29)         # 2eme arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

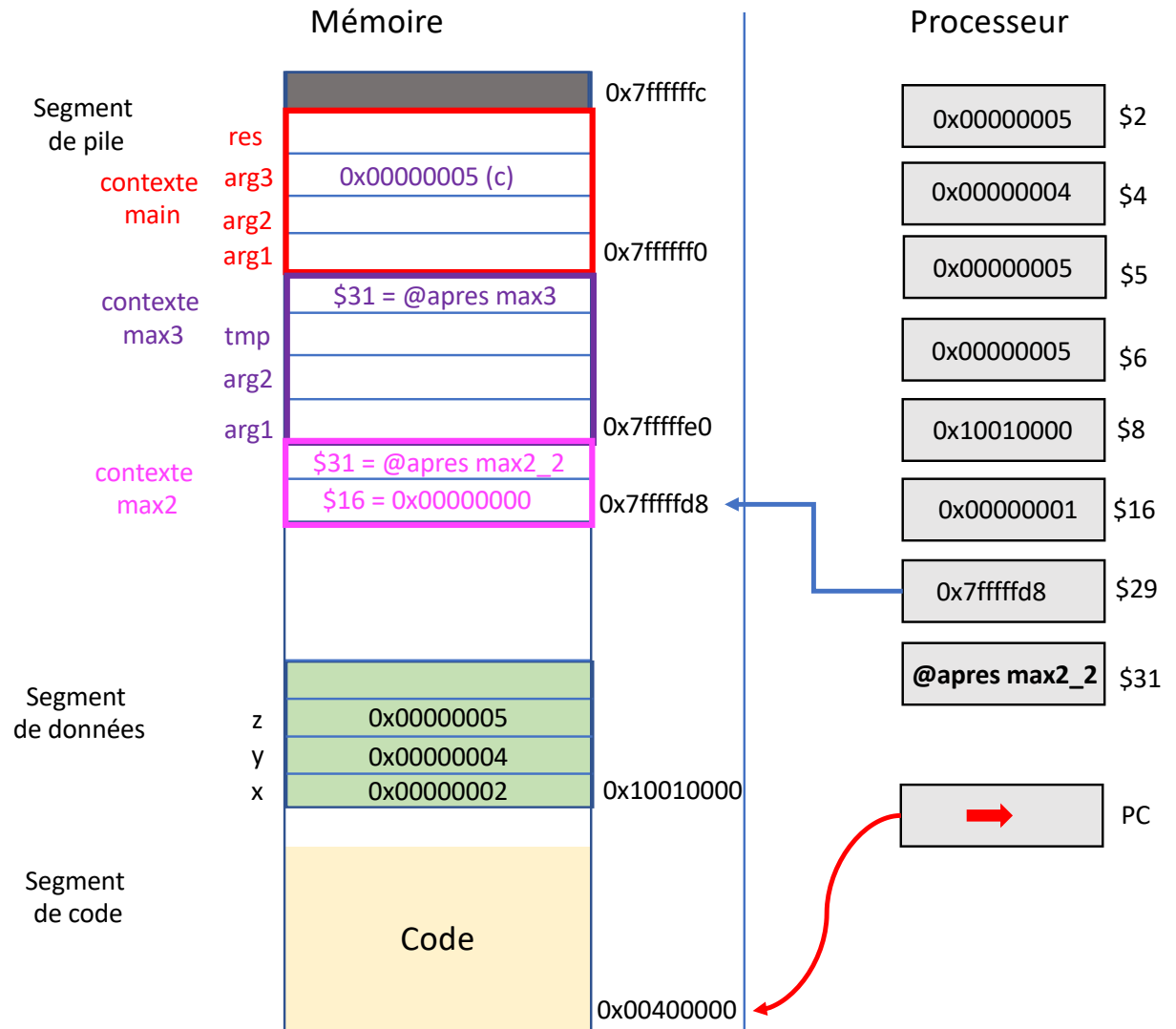
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3ème arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0
lw $5, 24($29)         # 2ème arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

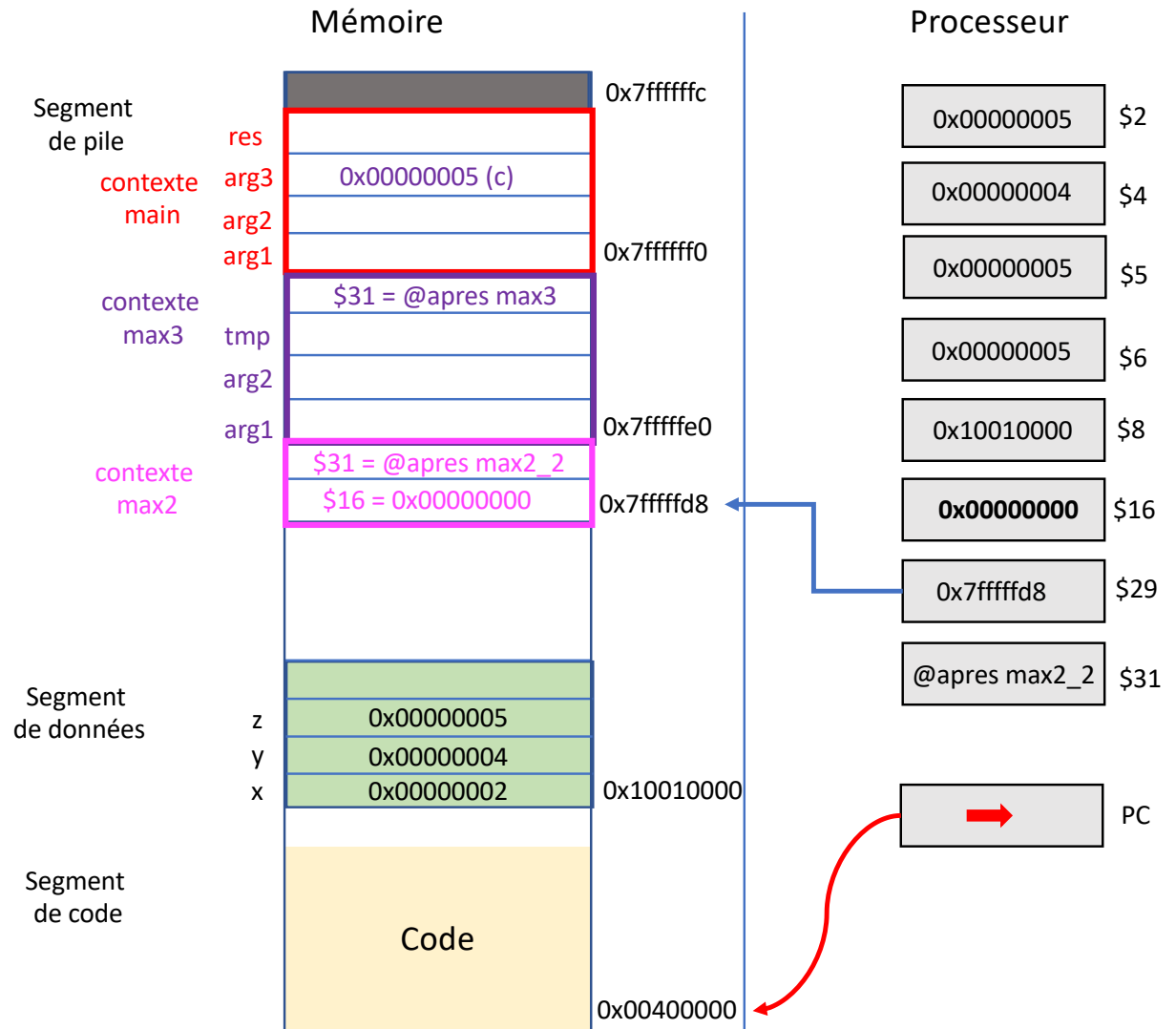
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3eme arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0
lw $5, 24($29)         # 2eme arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3eme arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0          # 1er arg=tmp
lw $5, 24($29)         # 2eme arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

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

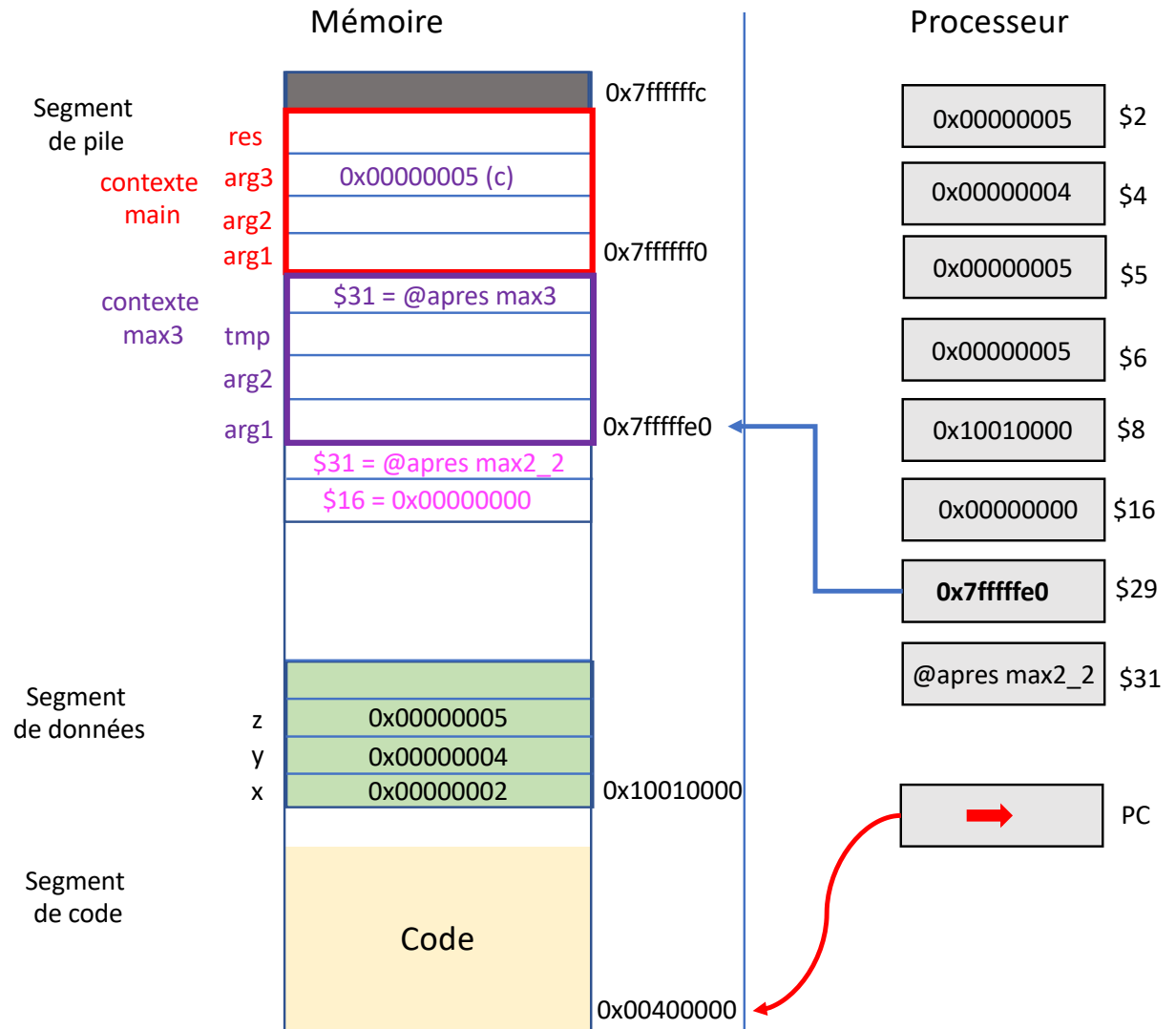
.text
addiu $29, $29, -16      # 1 var locale + 3 param max
lui    $8, 0x1001
lw     $4, 0($8)          # 1er arg = x
lw     $5, 4($8)          # 2ème arg = y
lw     $6, 8($8)          # 3ème arg = z
jal     max3              # appel à max3
ori     $4, $2, 0
ori     $2, $0, 1         # affichage
syscall
addiu   $29, $29, +16     # désallocation pile
ori     $2, $0, 10
syscall

max3:    #-----
addiu   $29, $29, -16      # nr = 0 + $31, nv = 1, na = 2
sw      $31, 12($29)
sw      $6, 24($29)        # sauvegarde 3eme arg (c)
jal     max2
ori     $4, $2, 0          # 1er arg=tmp
lw      $5, 24($29)        # 2eme arg = c
jal     max2
lw      $31, 12($29)
addiu   $29, $29, +16
jr      $31

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        # $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:
lw      $31, 4($29)
lw      $16, 0($29)
addiu   $29, $29, +8
jr      $31

```

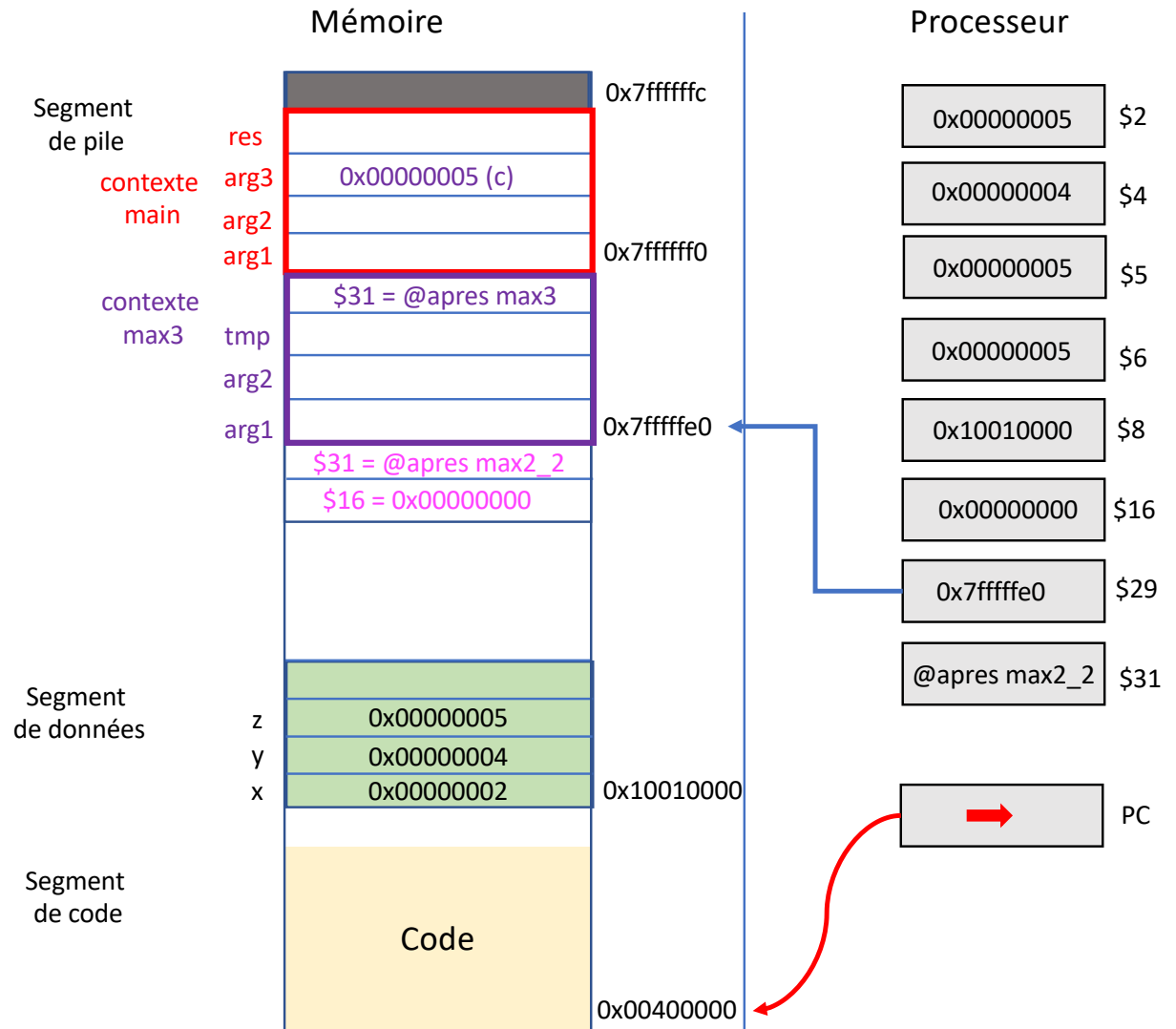


Exécution et évolution de la pile

```

.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3ème arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0
lw $5, 24($29)         # 2ème arg = c
→ jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31

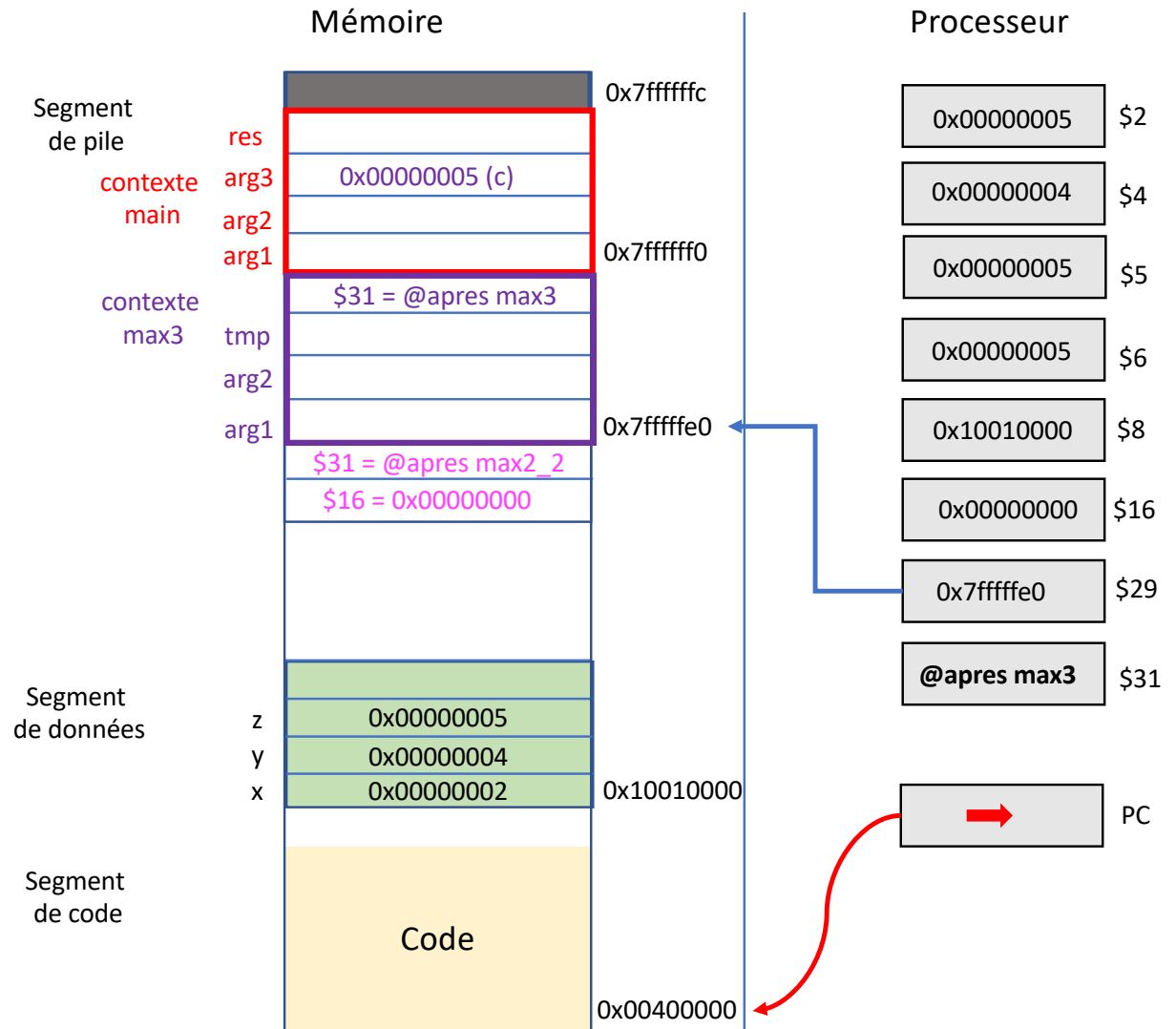
```



Exécution et évolution de la pile

```

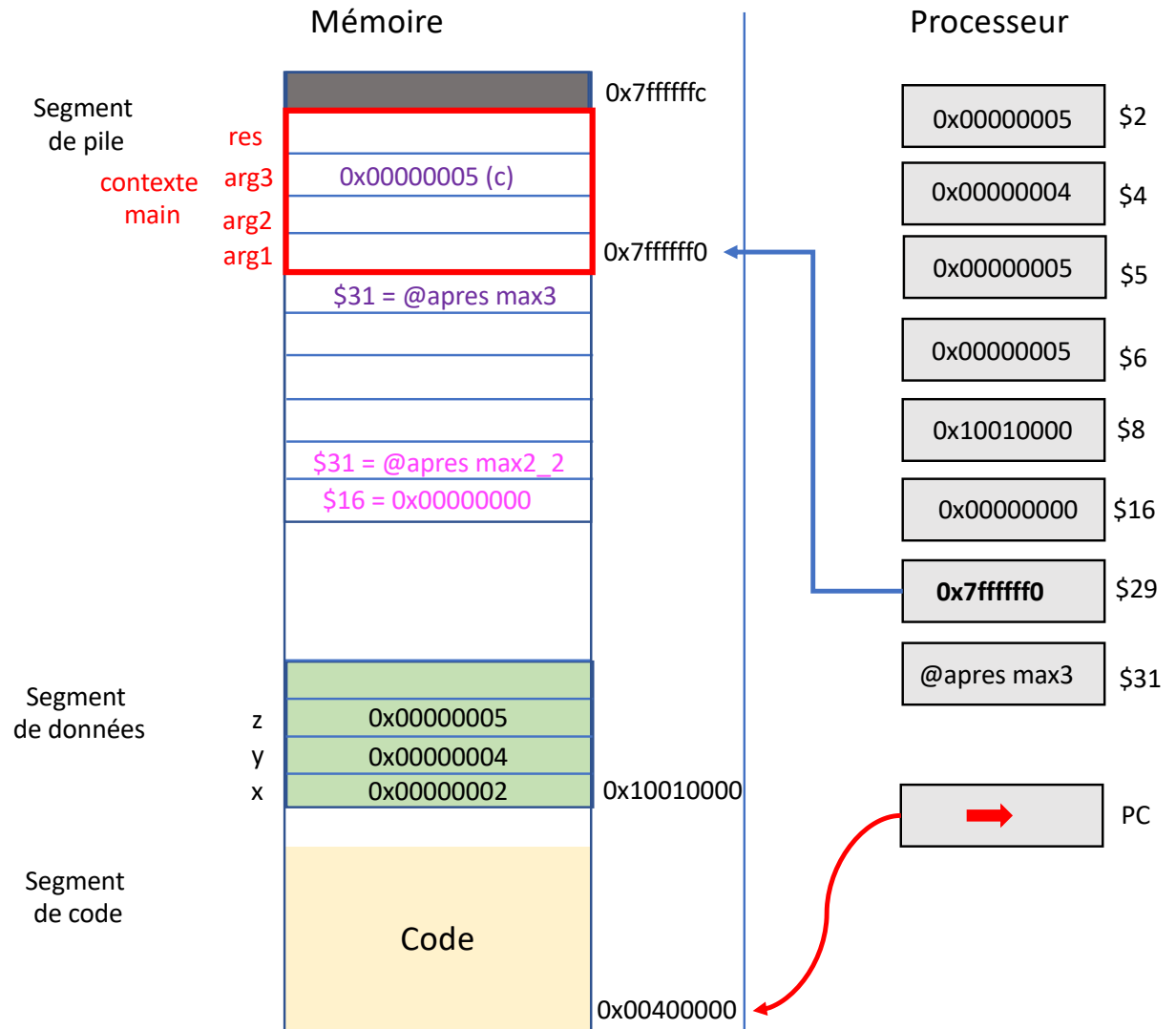
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3ème arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0
lw $5, 24($29)         # 2ème arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16    ←
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

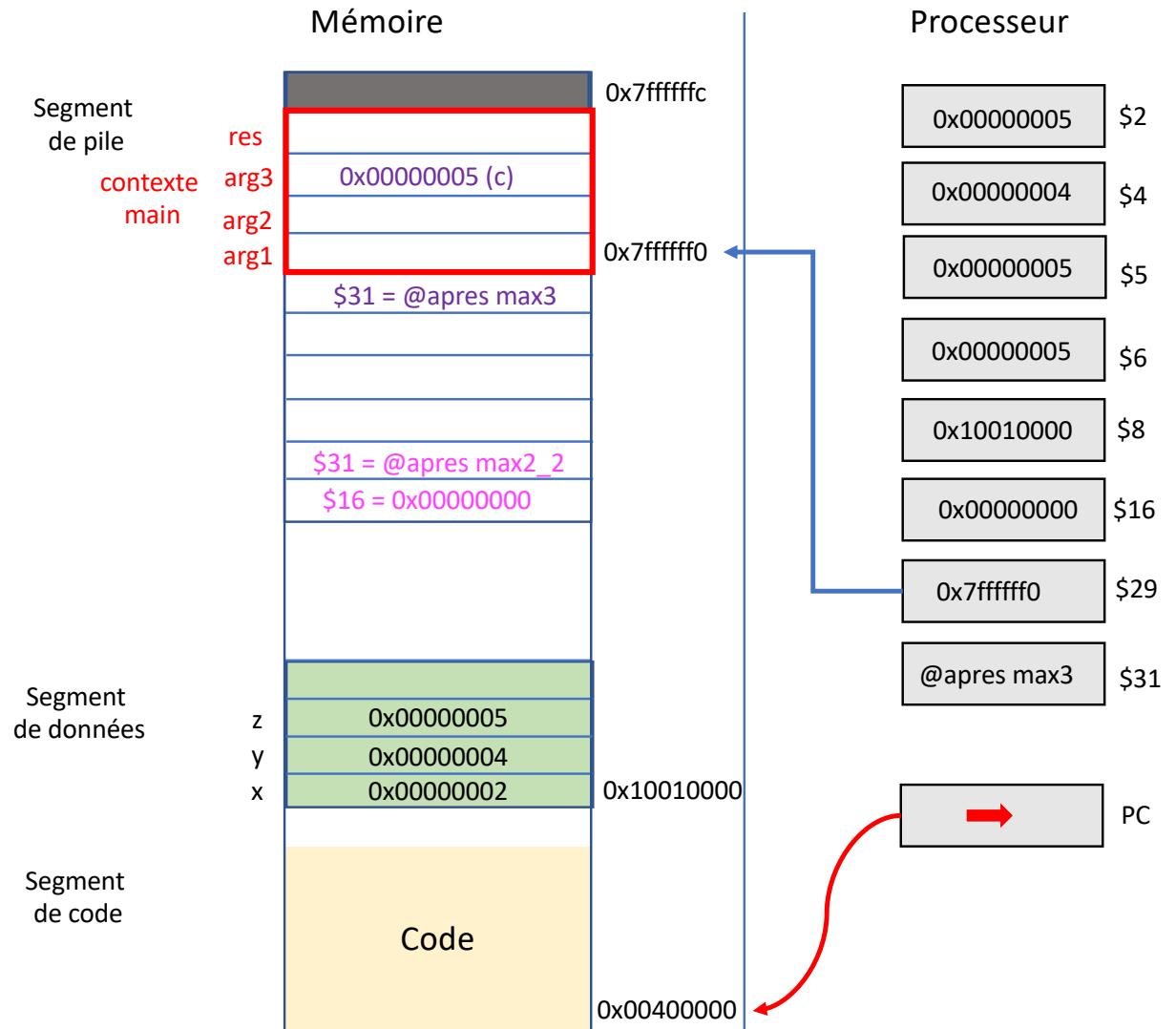
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3eme arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0          # 1er arg=tmp
lw $5, 24($29)         # 2eme arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
→ jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

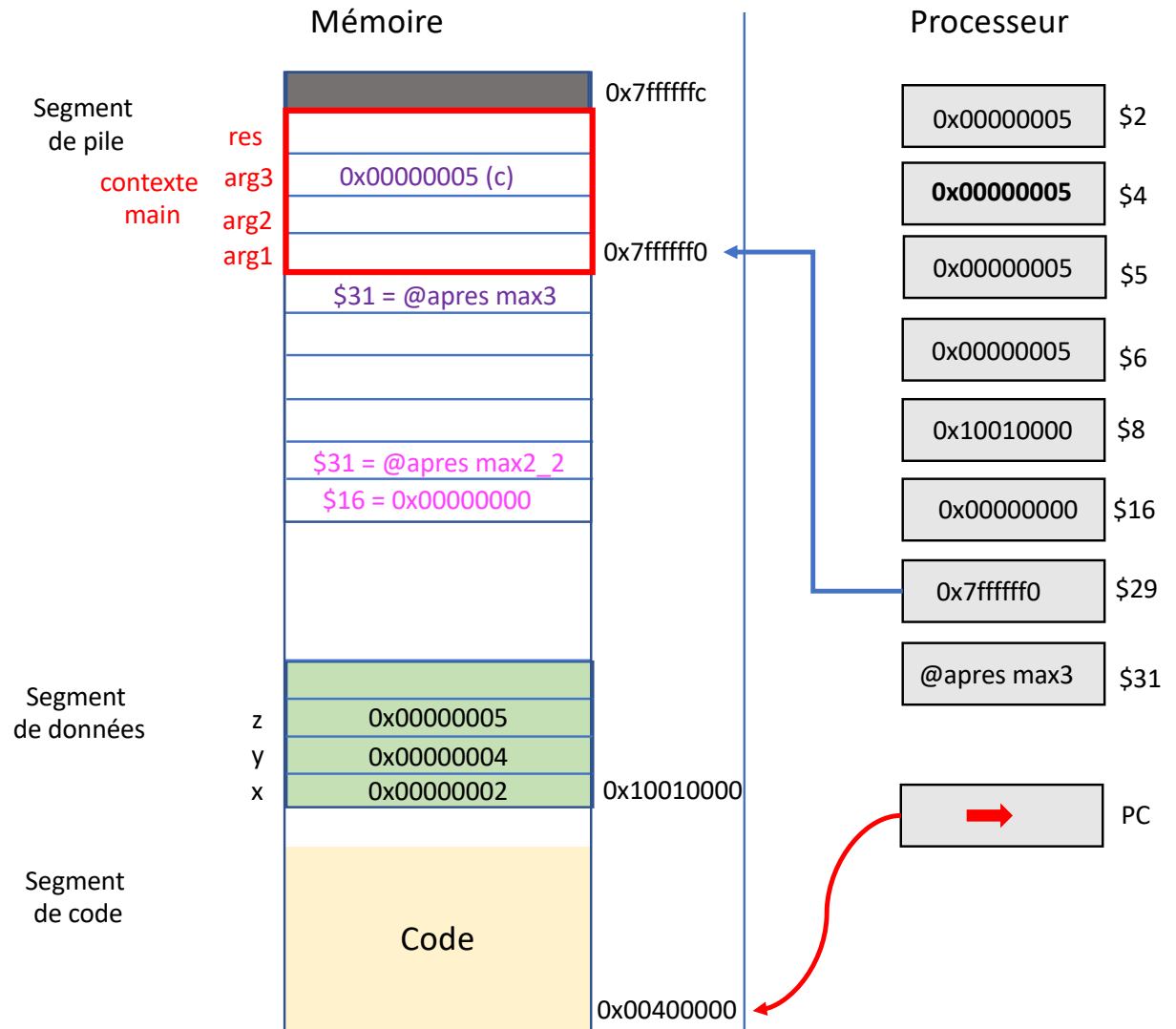
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3eme arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0          # 1er arg=tmp
lw $5, 24($29)         # 2eme arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile

```

.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3eme arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0          # 1er arg=tmp
lw $5, 24($29)         # 2eme arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```

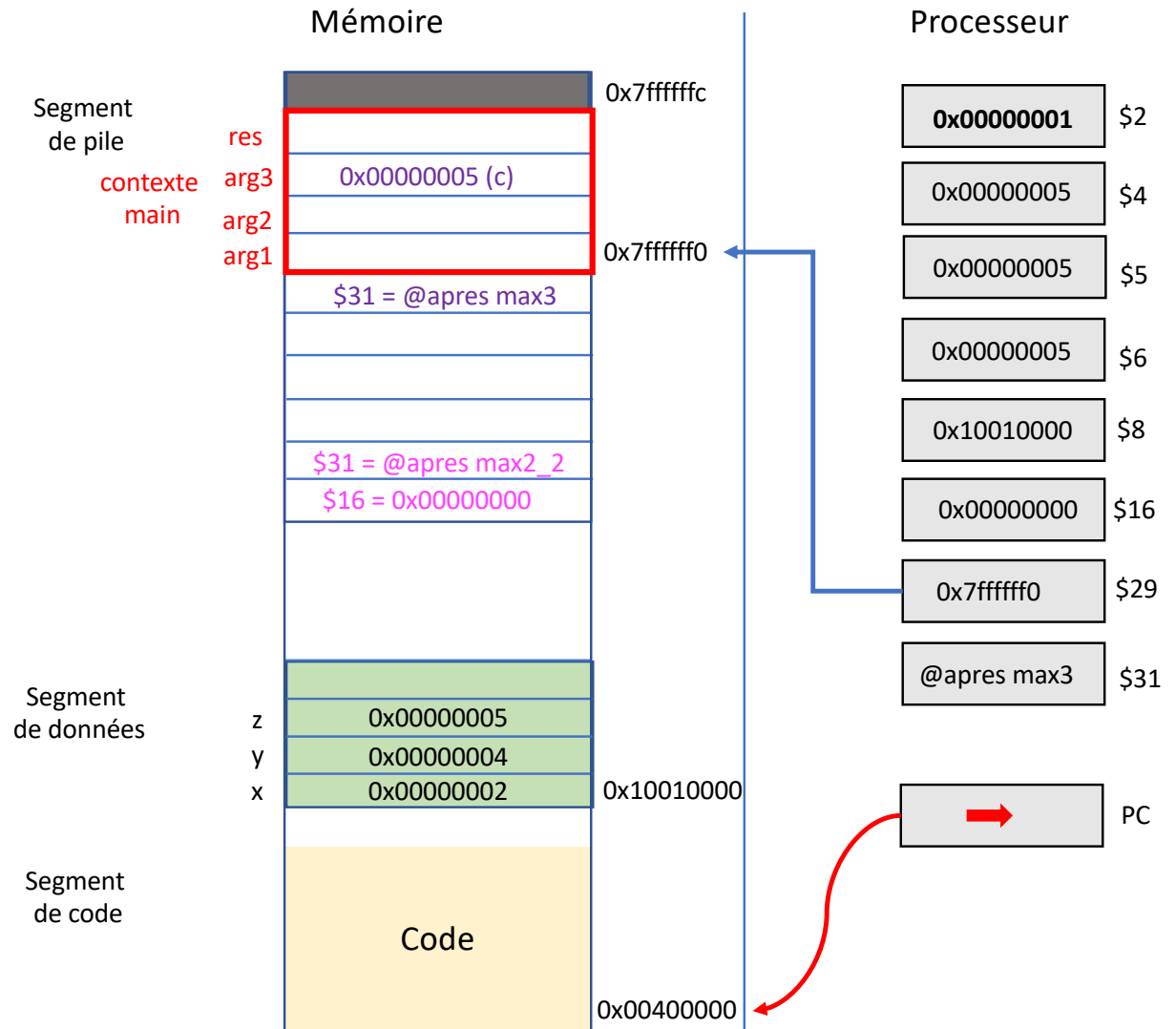


Exécution et évolution de la pile

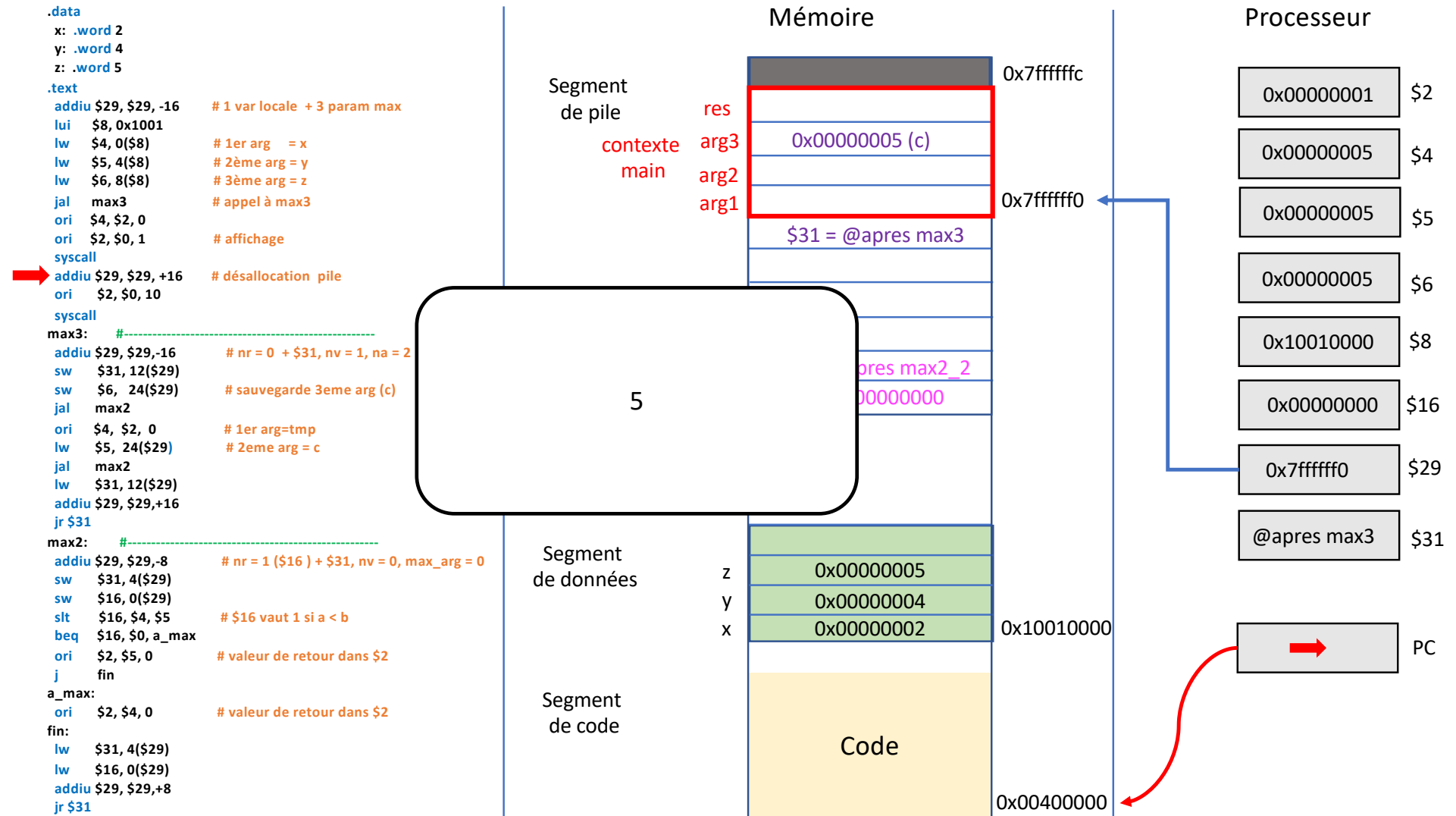
```

.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
→ syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3ème arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0          # 1er arg=tmp
lw $5, 24($29)         # 2ème arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31

```



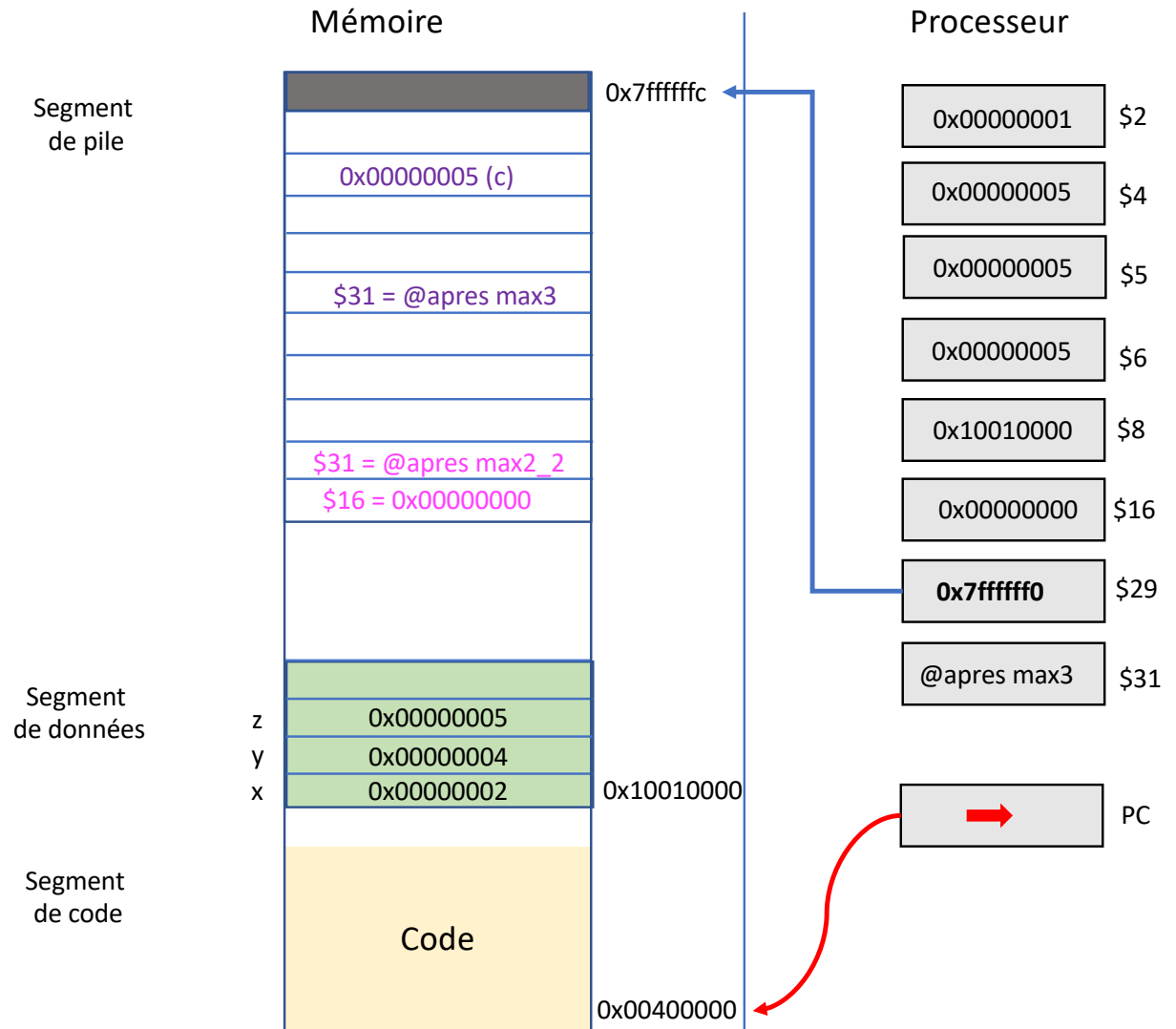
Exécution et évolution de la pile



Exécution et évolution de la pile

```

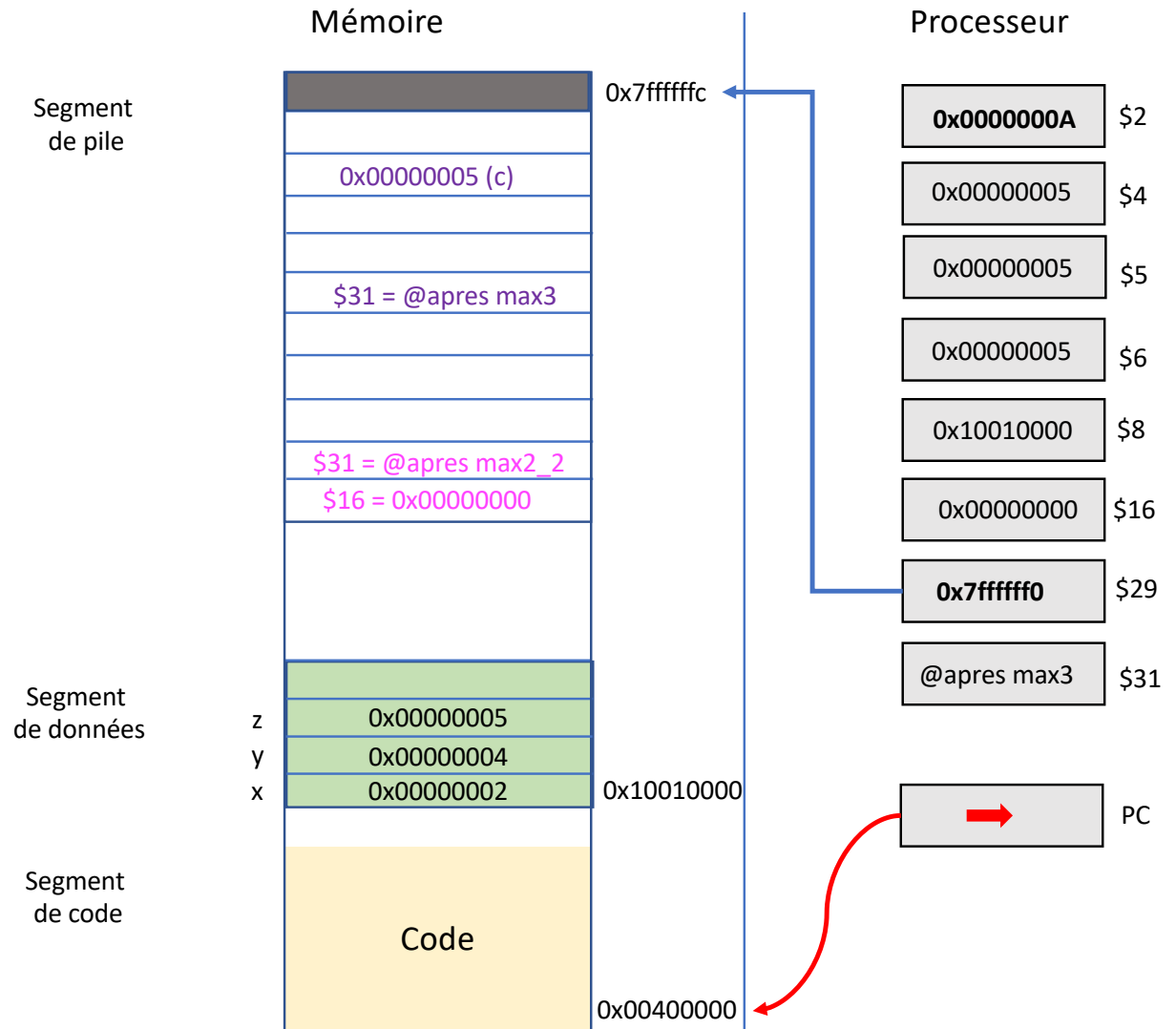
.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
→ ori $2, $0, 10
syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)
sw $6, 24($29)         # sauvegarde 3eme arg (c)
jal max2
ori $4, $2, 0          # 1er arg=tmp
lw $5, 24($29)         # 2eme arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



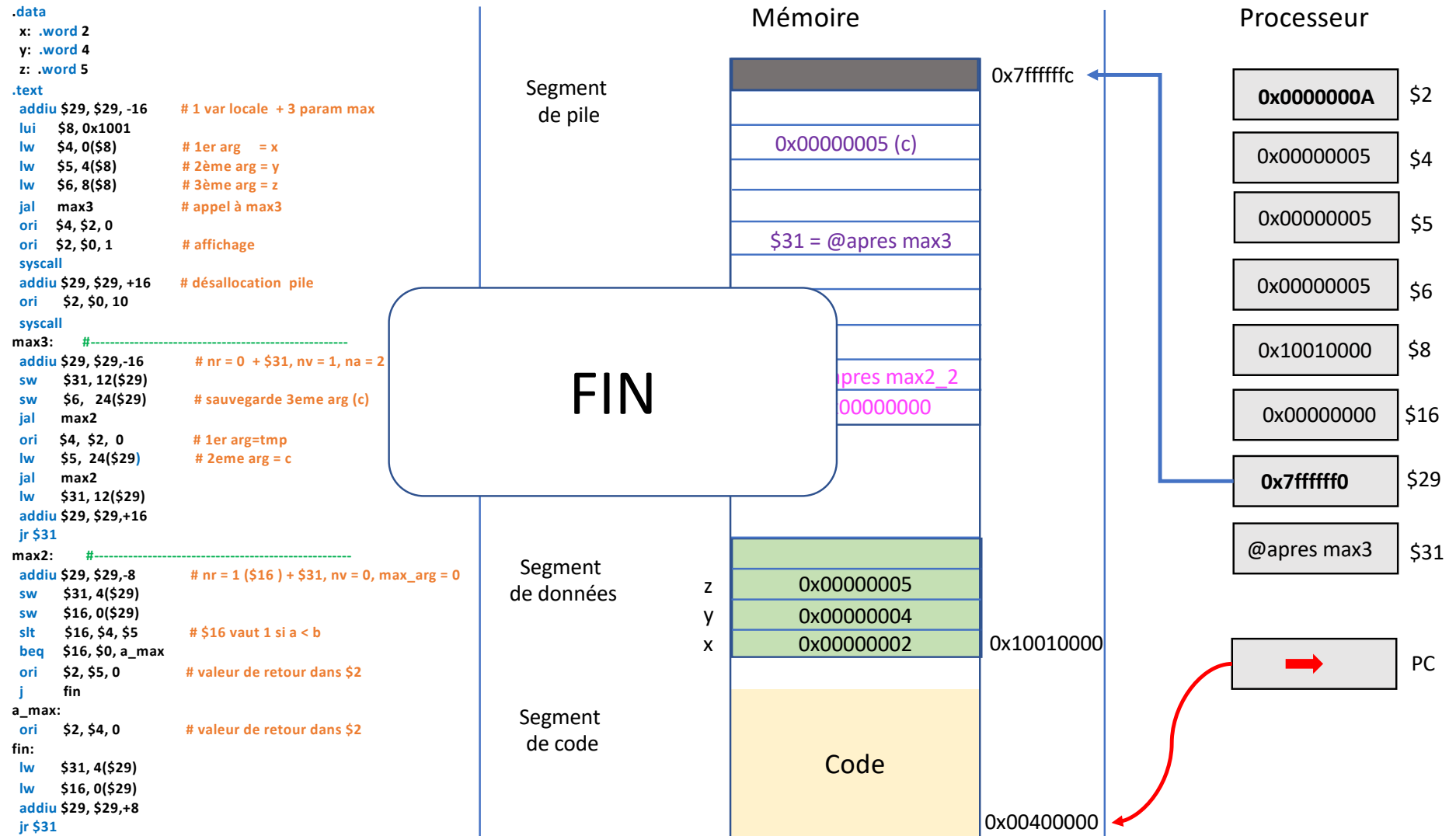
Exécution et évolution de la pile

```

.data
x: .word 2
y: .word 4
z: .word 5
.text
addiu $29, $29, -16    # 1 var locale + 3 param max
lui $8, 0x1001
lw $4, 0($8)           # 1er arg = x
lw $5, 4($8)           # 2ème arg = y
lw $6, 8($8)           # 3ème arg = z
jal max3               # appel à max3
ori $4, $2, 0
ori $2, $0, 1          # affichage
syscall
addiu $29, $29, +16    # désallocation pile
ori $2, $0, 10
→ syscall
max3: #-----
addiu $29, $29, -16    # nr = 0 + $31, nv = 1, na = 2
sw $31, 12($29)        # sauvegarde 3ème arg (c)
sw $6, 24($29)
jal max2
ori $4, $2, 0          # 1er arg=tmp
lw $5, 24($29)         # 2ème arg = c
jal max2
lw $31, 12($29)
addiu $29, $29, +16
jr $31
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        # $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:
lw $31, 4($29)
lw $16, 0($29)
addiu $29, $29, +8
jr $31
    
```



Exécution et évolution de la pile



Élément d'une suite réursive

```
int n = 3;
int u_n(int n) {
    if (n == 0)
        return 42;
    else
        return 3 * u_n(n-1) + n;
}

void main() {
    int tmp = u_n(n);
    printf("%d", tmp);
    exit();
}
```

- La fonction `u_n` s'appelle elle-même
- C'est un appel comme un autre, le problème de la terminaison est algorithmique et non au niveau assembleur
- Ici la récursion est non terminale

Code exemple : programme `main`

```
.data
n: .word 3

.text # main : 1 var local, 1 arg max
    addiu $29, $29, -8

    # elem = u_n(n)
    lui   $8, 0x1001
    lw    $4, 0($8)
    jal   u_n
    # printf("%d", tmp)
    ori   $4, $2, 1
    ori   $2, $0, 1
    syscall

    addiu $29, $29, 8
    ori   $2, $0, 10
    syscall
```

Code exemple : fonction `u_n`

```
u_n: # u_n(n) = 42 si n=0 sinon 3*u_n(n-1)+n
    addiu $29, $29, -8 # nr=0 +$31, nv=0, na=1
    sw     $31, 4($29)
    sw     $4, 8($29) # sauvegarde de n
    beq    $4, $0, fin_u0
    addiu  $4, $4, -1 # 1er arg = n - 1
    jal    u_n
    ori    $8, $0, 3
    mult   $2, $8      # u_n(n-1)*3
    mflo   $2
    lw     $8, 8($29) # relecture n
    addiu  $2, $2, $8 # u_n(n-1)*3+n
    j      fin_u
fin_u0:
    ori    $2, $0, 42 # cas n = 0
fin_u:
    lw     $31, 4($29)
    addiu  $29, $29, 8
    jr     $31
```

```
int n = 3;
int u_n(int n) {
    if (n == 0)
        return 42;
    else
        return 3*u_n(n-1) +
                n;
}
```

- Appel de la fonction `u_n` dans `u_n` : respect des conventions d'appel, pas plus !
- Sauvegarde de l'argument dans son emplacement car `$4` est écrasé lors de l'appel récursif.

Arbre d'appels

```
n = 3
```

```
main -> u_n(3)
```

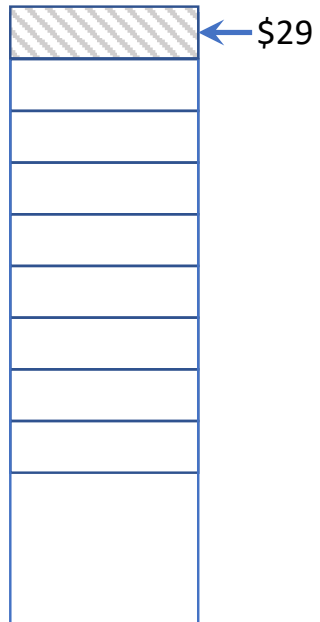
```
        -> u_n(2)
```

```
            -> u_n(1)
```

```
                -> u_n(0)
```

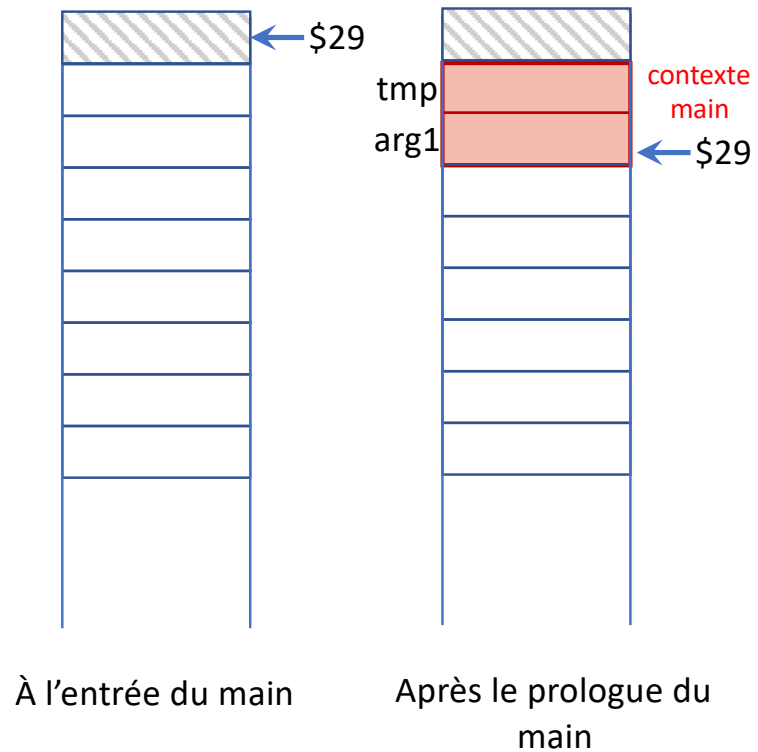
- Quand n vaut 3, il y a 3 appels récursifs.
- Chaque appel à u_n alloue un contexte et sauvegarde son argument dans son emplacement qui se trouve dans le contexte de la fonction appelante.
- La pile contient tous les contextes lorsque l'on est dans l'appel terminal ($u_n(0)$).

Évolution de la pile

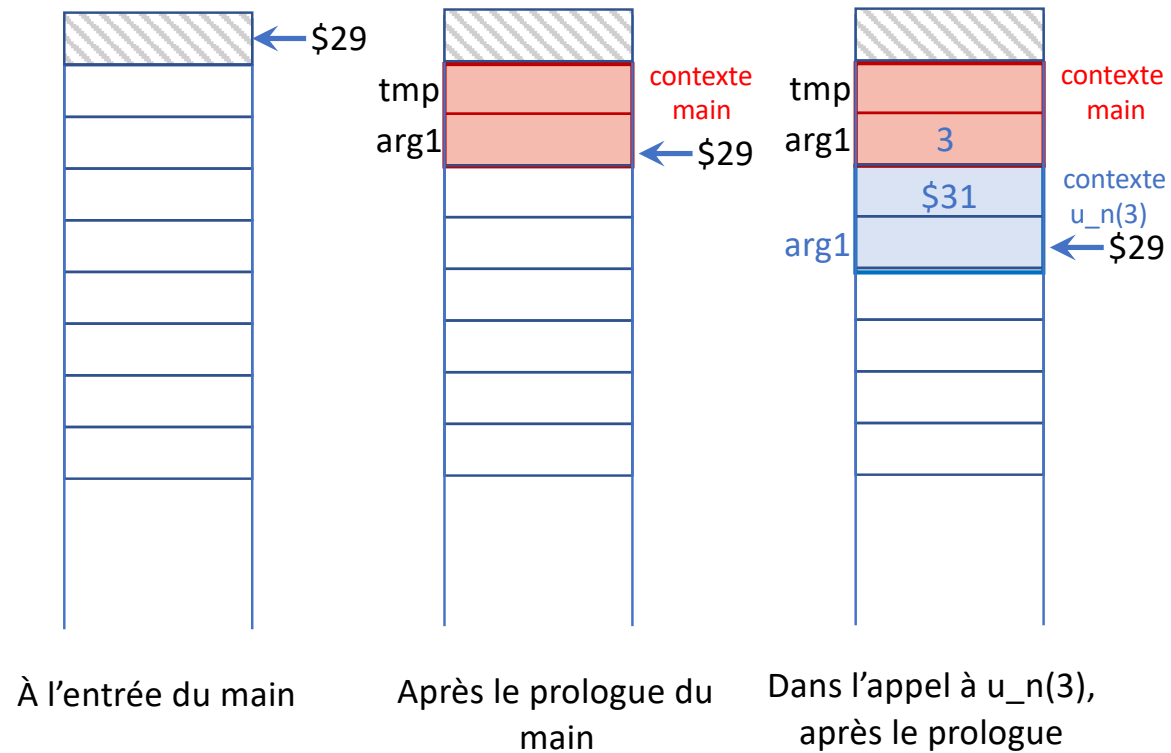


À l'entrée du main

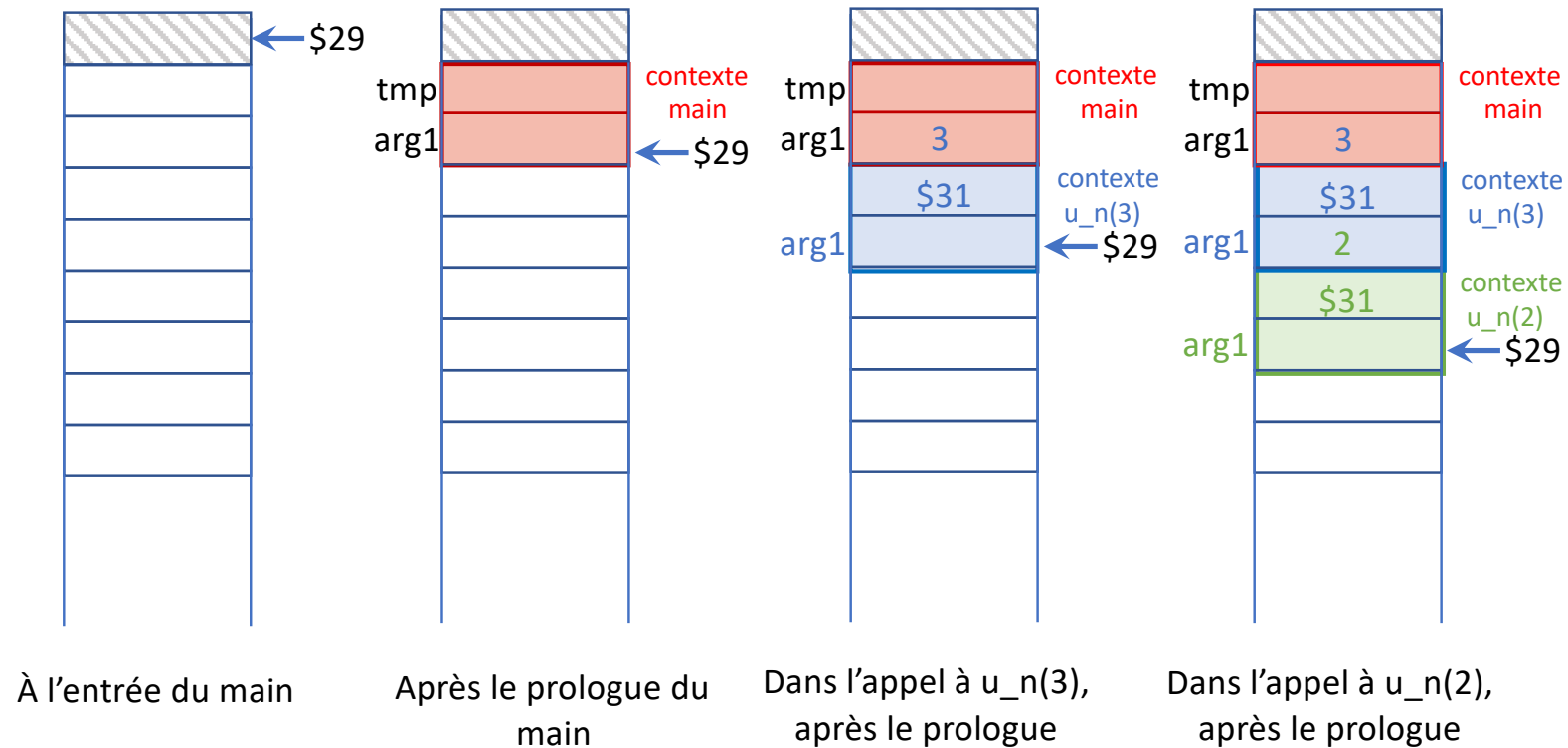
Évolution de la pile



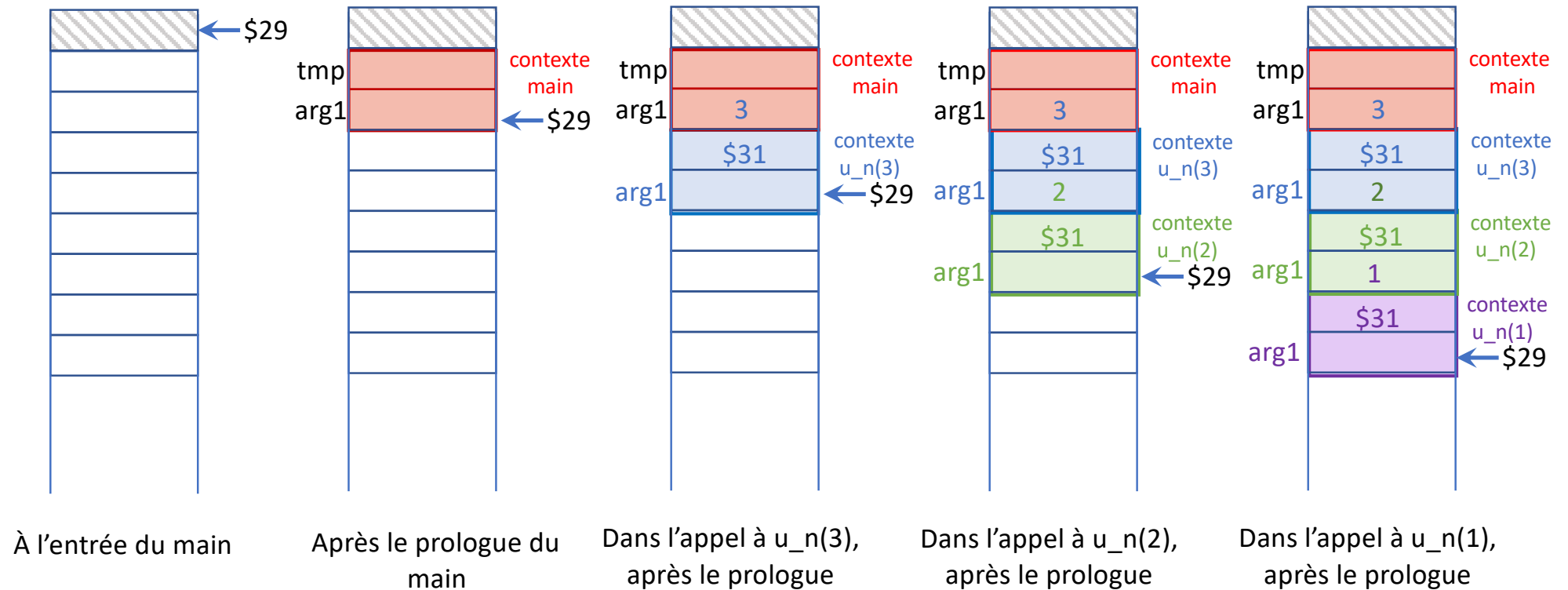
Évolution de la pile



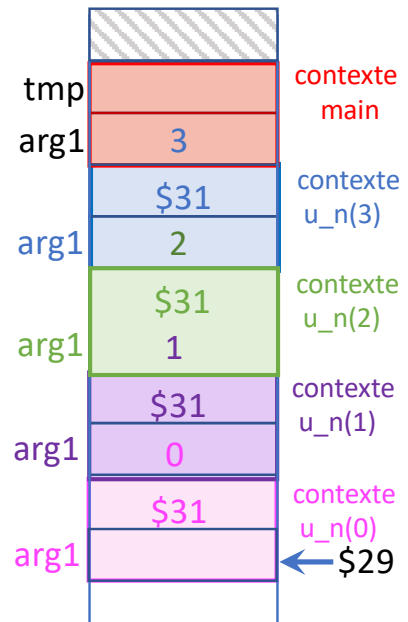
Évolution de la pile



Évolution de la pile

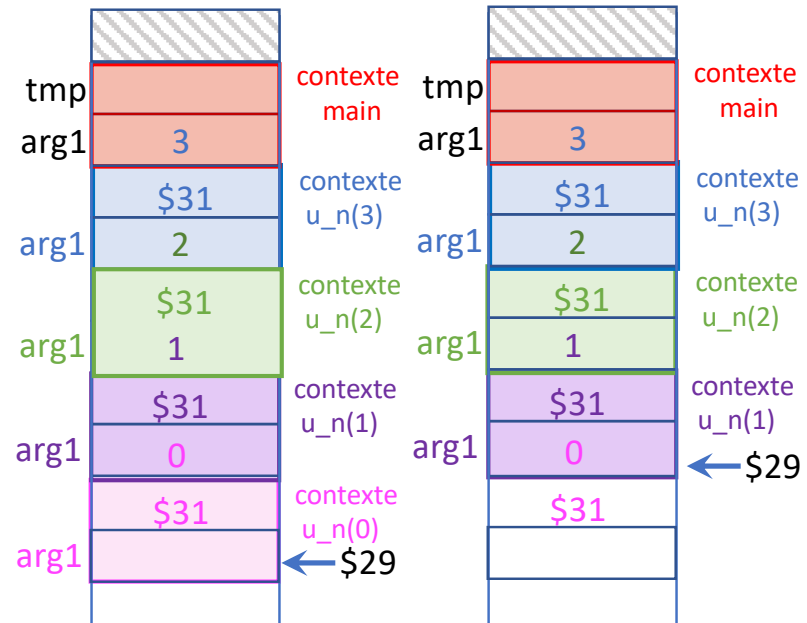


Évolution de la pile



Dans l'appel à `u_n(0)`,
après le prologue

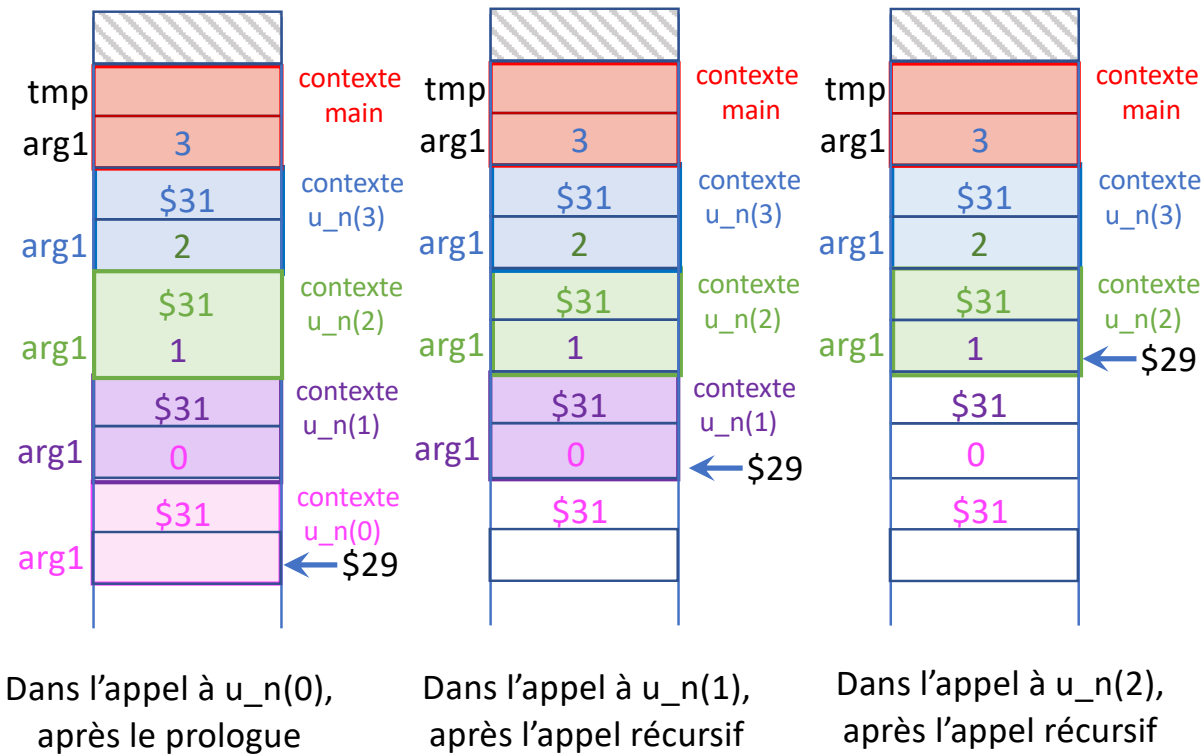
Évolution de la pile



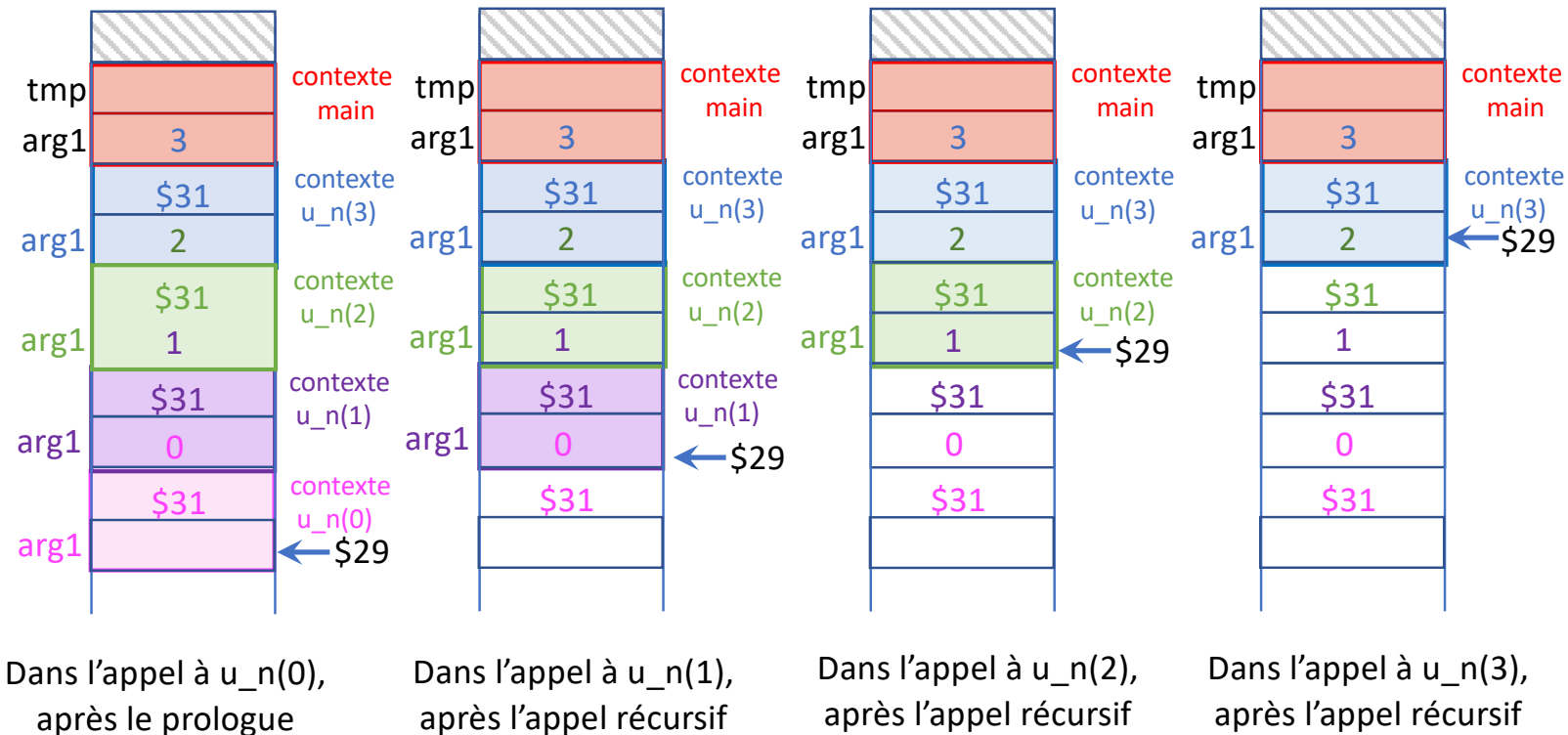
Dans l'appel à u_n(0),
après le prologue

Dans l'appel à u_n(1),
après l'appel récursif

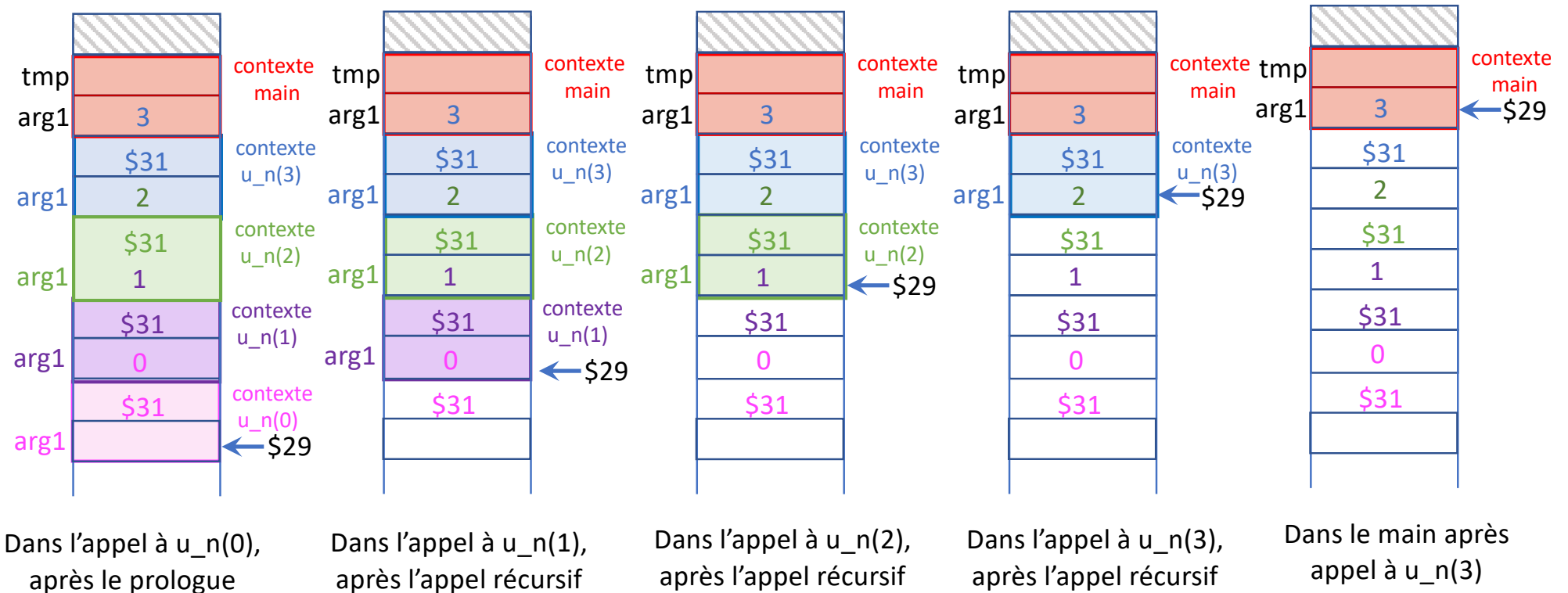
Évolution de la pile



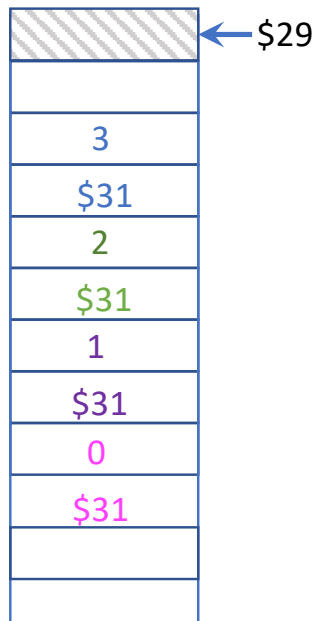
Évolution de la pile



Évolution de la pile



Évolution de la pile



À la fin du main
avant le exit()

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 arg 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 à max2bis

# 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 à max2
...
```

Code exemple : programme principal (fin)

```
# printf("%d",tmp)
lui    $8, 0x1001    # $8 non persistant => rechargement de @x
lw     $4, 12($8)    # lecture tmp en mémoire
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 !

Code exemple : max2bis

```
max2bis:
# prologue
addiu $29, $29, -8 # nr = 1 ($16) + $31, nv = 0, arg_max = 0
sw     $31, 4($29)
sw     $16, 0($29)
# corps de max2bis: $4 contient a, $5 contient b
# $6 contient res qui désigne une adresse
slt    $16, $4, $5 # $16 vaut 1 si a < b
beq    $16, $0, a_max2
sw     $5, 0($6)    # *res = b
j      fin
a_max2:
sw     $4, 0($6)    # *res = a
fin:
# epilogue
lw     $31, 4($29)
lw     $16, 0($29)
addiu  $29, $29, +8
jr     $31
```

- Notez les instructions d'écriture du résultat dans `*res`.
- Le 3ème argument, dans `$6`, contient une adresse, celle de la variable devant contenir le résultat

Variable locale paramètre par adresse

```
void main() {  
    int x = 2, y = 4, z = 5, tmp;  
    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;  
}
```

- Le 3ème argument des 2 appels est l'adresse de la variable locale `tmp`.
- Le deuxième appel a pour 1er argument la valeur de `tmp`.

Code exemple : programme principal (début)

```
.data
.text
    addiu $29, $29, -28  # 7 mots : 4 var locales + 3 arg max
    # x optimisée dans $16, y dans $17, z dans $18
    # tmp se trouve a $29+24
    ori    $16, $0, 2
    ori    $17, $0, 4
    ori    $18, $0, 5

    # max2bis(x,y,&tmp)
    ori    $4, $16, 0      # 1er paramètre = x dans $4
    ori    $5, $17, 0      # 2ème paramètre = y dans $5
    addiu  $6, $29, 24      # 3ème paramètre = &tmp dans $6
    jal    max2bis          # premier appel à max2bis

    # max2bis(tmp,z,&tmp)
    lw     $4, 24($29)      # 1er paramètre = tmp dans $4
    ori    $5, $18, 0      # 2ème paramètre = z dans $5
    addiu  $6, $29, 24      # 3ème paramètre = &tmp dans $6
    jal    max2bis          # second appel à max2
    ...
```

Code exemple : programme principal (fin)

```
# printf("%d",tmp)
lw      $4, 24($29)      # lecture tmp
ori     $2, $0, 1        # affichage
syscall

# terminaison
addiu   $29, $29, +28    # désallocation emplacements pile?;
ori     $2, $0, 10
syscall
```

- L'adresse de `tmp` est relative au pointeur de pile `$29`.
- Le code est très proche de la version précédente, il faut juste adapter l'adresse de la variable.
- Bien sur si `tmp` était lue avant d'être écrite, il aurait fallu écrire sa valeur dans son emplacement sur la pile : l'optimisation en registre s'arrête lors de manipulation de valeur des variables locales via la pile.

Pour aller plus loin...

```
int count_char(char c) {
    int cpt = 0;
    int i = 0;
    char ch[20];
    scanf("%s", ch);
    while (ch[i] != 0) {
        if (ch[i] == c)
            cpt++;
    }
    return cpt;
}

void main() {
    printf("%d", count_char('a'));
    exit();
}
```

- La fonction `count_char` lit une chaîne au clavier qu'elle stocke dans sa variable locale.
- Que se passe-t-il si la chaîne entrée fait plus de 20 caractères ?

Récursion terminale

```
int n = 3;
int u_n(int n) {
    if (n == 0)
        return 42;
    else
        return 3*u_n(n-1) + 4;
}
int u_term(int n) {
    return u_nterm(n, 42);
}
int u_nterm(int n, int u) {
    if (n == 0)
        return u;
    else
        return u_term(n-1, 3*u + 4);
}
void main() {
    printf("%d", u_n(n));
    printf("%d", u_term(n));
    exit();
}
```

- La suite u_n est : u_0 vaut 42 et $u_{n+1} = 3 * u_n + 4$.
- Version recursive terminale possible via la fonction `u_term` qui appelle `u_nterm` avec la valeur de l'élément au rang 0
- La fonction `u_nterm` s'appelle elle-même, mais son résultat est celui de l'appel récursif (cas récursif), la récursion est récursive terminale.
- Ca change quoi en assembleur ?

Fonction `u_nterm`

```
u_nterm: # u_nterm(n,t) = t si n = 0 sinon u_nterm(n-1, 3*t+4)
    addiu $29, $29, -12 # na = 2 + nv = 0 + nr = 0 + $31
    sw     $31, 8($29)
    beq    $4, $0, fin_u_nterm0
    addiu  $4, $4, -1 # n - 1
    ori    $6, $0, 3
    mult   $5, $6      # 3*t
    mflo   $5
    addiu  $5, $5, 4    # 3*t+ 4
    jal    u_nterm      # u_nterm(n-1, 3*t+4)
    j      fin_u_nterm
fin_u_nterm0: # cas n = 0
    or     $2, $0, $5
fin_u_nterm:
    lw     $31, 8($29)
    addiu  $29, $29, 12
    jr     $31
```

- La fonction `u_nterm` doit bien allouer son contexte et attendre la fin de l'appel récursif pour le désallouer et retourner à la fonction appelante... Ca ne change rien ici (le contexte est même plus gros!).
- Il faut "optimiser" si on veut gagner l'empilement des contextes : la fonction `u_nterm` n'a plus besoin de son contexte, et la fonction appelée (la même) a besoin exactement des mêmes emplacements, on peut le réutiliser !

Fonction `u_nterm` optimisée

```
u_nterm_opt : # u_nterm(n,t) = t si n = 0,  
              # sinon vaut u_nterm(n-1, 3*t+4)  
    beq    $4, $0, fin_u_nterm_opt  
    ori    $6, $0, 3  
    mult   $5, $6  
    mflo   $5  
    addiu  $5, $5, 4    # 3*t + 4  
    addiu  $4, $4, -1   # n - 1  
    j      u_nterm_opt # saut à u_nterm_opt avec $4 = n-1 et $5 = 3*t+4  
fin_u_nterm_opt:  
    or     $2, $0, $5   # $2 <- t  
    jr     $31
```

- La fonction `u_nterm_opt` ne fait plus d'appel via `jal`, mais saute au début de la fonction en ayant changé les paramètres... cela devient une boucle !
- Ici plus de contexte du tout (pas de variable locale, ni sauvegarde de registre autre que `$31`), c'est possible avec un contexte en "bouclant" après le prologue et en le désallouant lorsque `n` vaut/atteint 0.
- Cette optimisation est réalisée par les compilateurs (elle s'appelle *tail call optimisation* – qui optimise aussi les fonctions "feuilles").

Ce qu'on a vu

Fonctions et appels de fonction en assembleur

- Rappel des conventions d'appel
- Ensemble d'exemples avec des fonctions imbriquées et des fonctions récursives
- Passage de paramètre par adresse
- La semaine prochaine : 3ème partie du cours avec Franck Wajsbürt !