# info201 : Système d'exploitation
# Examen

Pierre Hyvernat
Sébastien Monnet
UFR Sciences et Montagne, université Savoie Mont Blanc
Pierre.Hyvernat@univ-smb.fr
Sebastien.Monnet@univ-smb.fr

Documents et calculatrices interdits.

Durée : 1h30.

Un barème provisoire est donné dans la marge, un point négatif est réservé pour la présentation.

## Partie 1 : architecture, cours

(2) *Question 1.* À quoi correspondent les initiales UAL (*ALU* en anglais) et UC (*CU* en anglais).

(2) *Question 2.* Décrivez, en quelques lignes, les étapes effectuées pour exécuter un programme dont le code (en langage machine) a été chargé en RAM.

## Partie 2 : architecture, langage d'assemblage

Des opérations arithmétiques du langage d'assemblage MMIX sont :
- `ADD $x, $y, $z` pour additionner les registres `$y` et `$z` et mettre le résultat dans `$x`,
- `MUL $x, $y, $z` pour multiplier les registres `$y` et `$z` et mettre le résultat dans `$x`,
- `DIV $x, $y, $z` pour multiplier le registre `$y` par `$z` et mettre le résultat dans `$x`,
- `JMP n` pour aller directement à l'instruction "`+n`". Lorsque `n` est positif, cela "saute" `n-1` instructions.
- `BP $x, n` pour aller directement à l'instruction "`+n`", mais *seulement lorsque* `$x` *est strictement positif.* (Sinon, passe à l'instruction suivante.)

Les opérations `ADD`, `MUL` et `DIV` ont une variante où le registre `$z` est remplacé par une constante entière $n$ entre `0` et `255`.

(2) *Question 1.* En supposant que $x$ et $y$ sont deux registres contenant des entiers, comment peut-on effectuer l'opération "$y = (x + 2y + 3)^4 + 5$", sans modifier le registre $x$ ?

(3) *Question 2.* Comment pourrait-on coder la séquence Python suivante en MMIX *en utilisant uniquement les instructions données plus haut* ?

```
n = 0
while x > 0:
    n = n + 1
    x = x // 2
```

Vous supposerez que `x` est un registre contiennant un entier positif ou nul et que `n` est déjà initialisé à `0`.

*Attention,* 1 point est réservé au fait que votre code marche lorsque `x` vaut `0`.

## Partie 3 : shell et redirections

*Rappels :* on peut enchainer les commandes `CMD1 | CMD2 | CMD3` pour :
- lancer la commande `CMD1`,
- faire agir `CMD2` sur la sortie de `CMD1`,
- faire agir `CMD3` sur la sortie de `CMD2`.

(2) *Question 1. Rappels*

- La commande `head -n100 FICHIER1 FICHIER2 ... FICHIERn` affiche les 100 premières lignes de chaque fichier donné en argument. Si aucun fichier n'est donné, la commande affiche les 100 premières ligne de l'entrée standard.
- La commande `cat FICHIER1 FICHIER2 ... FICHIERn` affiche toutes les lignes des fichiers, dans l'ordre où ils ont été donnés. Si aucun fichier n'est donné, la commande affiche toutes les lignes de l'entrée standard.

Expliquez précisemment ce que font les 2 lignes de commandes suivantes :

```
$ head -n100 MISPI*.txt | cat
```

et

```
$ cat MISPI*.txt | head -n100
```

(2) *Question 2.*

La commande `factor NOMBRE1 NOMBRE2 ... NOMBREn` affiche la factorisation des nombres donnés en arguments :

```
$ factor 9 11
9: 3 3
11: 11
```

Si aucun nombre n'est donné, la commande agit sur tous les nombres de l'entrée standard.

La commande `grep " .* " FICHIER1 FICHIER2 ... FICHIERn` affiche les lignes contenant *au moins 2 espaces* parmi les fichiers donnés. Si aucun fichier n'est donné, la commande agit sur l'entrée standard. Si on ajoute l'argument `-v`, la commande est inversée et affiche les lignes contenant *au plus un espace*.

La commande `seq DEBUT FIN` affiche les entiers entre `DEBUT` et fin `FIN` sur la sortie standard.

Donnez une ligne de commandes qui affiche tous les nombres premiers plus petits que 1000. (Chaque ligne du résultat devra commencer par un des nombres premier mais pourra contenir d'autres choses.) N'oubliez pas de fournir des détails pour expliquer comment fonctionne votre ligne de commandes.

### Partie 4 : fichiers

(2) *Question 1.* Le *déplacement* d'un fichier

```
$ mv GROS_FICHIER GROS_FICHIER.backup
```

est en général très rapide, alors que la *copie*

```
$ cp GROS_FICHIER GROS_FICHIER.backup
```

est beaucoup plus lente.

Expliquez la différence en détaillant les opérations effectuées.

(1) *Question 2.* Parfois, le déplacement prend autant de temps qu'une copie

```
$ mv /home/hyvernat/GROS_FICHIER /media/usb/GROS_FICHIER.backup
```

Pourquoi ?

(2) *Question 3.* Décrivez certains des avantages et inconvénients à utiliser une allocation par blocs pour le stockage des fichiers sur leur support de stockage.

### Partie 5 : processus et ordonnancement

(2) *Question 1.* Décrivez l'ordonnancement des processus suivants

| processus | création (ms) | durée (ms) |
|---|---|---|
| $P_1$ | 0 | 400 |
| $P_2$ | 298 | 300 |
| $P_3$ | 299 | 200 |
| $P_4$ | 499 | 100 |

lorsque l'ordonnanceur est de type "tourniquet" avec un quantum de temps de 100ms.