GIF-1001 Ordinateurs: Structure et Applications
Hiver 2015
Examen final
28 avril 2015
Durée: 110 minutes

Cet examen comporte 8 questions sur 13 pages (incluant celle-ci), comptabilisées sur un total de 100 points. L'examen compte pour 40% de la note totale pour la session. Assurez-vous d'avoir toutes les pages. Lisez attentivement les notes suivantes:

- Vous avez droit à une feuille aide-mémoire 8.5×11 recto-verso, écrite à la main, ainsi qu'une calculatrice acceptée.
- Écrivez vos réponses dans le cahier bleu qui vous a été remis;
- L'annexe A contient un rappel sur les unités en binaire, ainsi que sur le calcul des logarithmes en base 2.
- L'annexe B contient une liste d'instructions ARM.

La table ci-dessous indique la distribution des points pour chaque question.

Question:	1	2	3	4	5	6	7	8	Total
Points:	10	10	15	15	15	10	15	10	100

Bonne chance, et bon été!

1. (10 points) Répondez aux questions suivantes portant sur les ports série et USB.

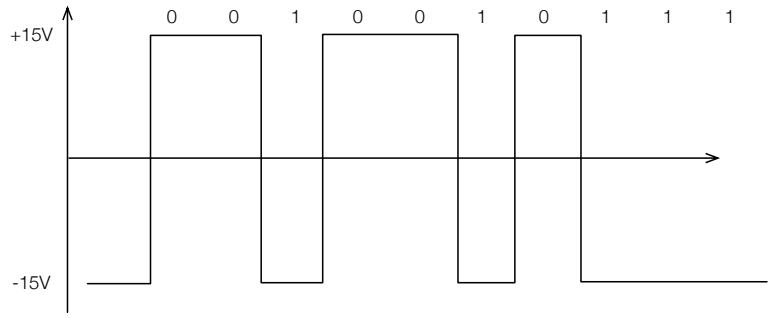
   (a) (5 points) Dans le protocole RS-232, un signal entre +3V et +15V est interprété comme un 0 logique, tandis qu'un signal entre -3V et -15V l'est comme un 1 logique. Dessinez l'évolution du voltage sur la ligne de transmission si l'on envoie l'octet `0xD2`, et que l'on emploie la configuration suivante:

   - 8 bits par octet
   - parité impaire
   - 1 bit d'arrêt

   Solution: Les bits sont envoyés du LSB au MSB. La chaîne de bits correspondante à `0xD2` est: 11010010, donc on envoie:

   0 (start bit) - 0 - 1 - 0 - 0 - 1 - 0 - 1 - 1 - 1 (parité impaire) - 1 (stop bit)

   Donc, l'évolution du voltage est:

   (b) (5 points) Contrairement au RS-232, le bus USB utilise deux lignes (D+ et D-) pour envoyer des données. Décrivez le fonctionnement de cette approche, et expliquez l'avantage de procéder de cette manière.

   Solution: Cette approche est nommée "transfert différentiel". Le signal est envoyé sur D+, et son inverse sur D-. Au bout de la ligne, le signal est reconstruit en calculant la différence (D+ - D-). L'avantage de cette approche est la résistance au bruit (additif) présent sur la ligne.

2. (10 points) Ordonnancement de processus. La liste suivante indique des processus en attente d'exécution (dans l'ordre de leur arrivée), ainsi que leur durée.

   1. P1, 2 quanta
   2. P2, 4 quanta
   3. P3, 3 quanta
   4. P4, 1 quanta

   Pour les trois questions suivantes, indiquez quel processus sera exécuté à chaque quantum de temps si les algorithmes suivants sont utilisés:

   (a) (3 points) Premier arrivé, premier servi.

Solution: P1 - P1 - P2 - P2 - P2 - P2 - P3 - P3 - P3 - P4

(b) (3 points) Plus court d'abord.

Solution: P4 - P1 - P1 - P3 - P3 - P3 - P2 - P2 - P2 - P2

(c) (3 points) Tourniquet.

Solution: P1 - P2 - P3 - P4 - P1 - P2 - P3 - P2 - P3 - P2

(d) (1 point) Lequel de ces trois algorithmes est le plus équitable, et pourquoi?

Solution: Le "tourniquet", car c'est celui qui distribue le temps du microprocesseur à tous les processus de façon égale.

3. (15 points) Dans un système avec mémoire paginée où:

   - les pages ont une taille de 4Ko;
   - le nombre maximal de pages est de 65 536;
   - la taille de la mémoire physique est de 64Mo;
   - 2 bits d'information supplémentaire sont stockés à chaque entrée dans la table des pages;
   - un extrait de la table des pages est donné par:

Page	Trame ("frame")	Information
0x00	0x02	...
0x01	0xB2	...
0x02	0x1	...
⋮	⋮	⋮
0x10	0x1E	...
0x11	0x1A4	...
0x12	0xF0	...

   (a) (3 points) Quelle est la quantité maximale de mémoire pouvant être utilisée par un programme?

   Solution: Le nombre maximal de pages est 65536, et chaque page a une taille de 4Ko. La taille totale d'un programme est donc de $65536 \times 4\text{Ko} = 256\text{Mo}$

   (b) (3 points) Quelle est la taille de la table des pages?

   Solution: Il y a 65 536 pages au total, et 64Mo/4Ko = 16384 trames. Nous avons donc besoin de $\log_2(16384) = 14$ bits pour stocker l'index de la trame à chaque entrée dans la table des pages. La taille totale de la table des pages sera de $65536 \times (14 + 2) = 128\text{Ko}$

   (c) (4 points) À quoi sert la table des pages? Donnez un exemple de son utilisation en expliquant comment un MMU traiterait l'adresse logique `0x0011A3C` dans le contexte de la table des pages ci-haut.

   Solution: La table des pages sert à traduire une adresse virtuelle en une adresse physique. L'adresse logique `0x0011A3C` est séparée en deux: 1) 16 bits pour le numéro de page `0x0011`, et 2) 12 bits pour l'offset `0xA3C`. Le numéro de trame correspondant à la page est `0x1A4`, donc l'adresse résultante serait `0x1A4A3C`.

   (d) (5 points) Comment est-il possible que la taille maximale d'un programme puisse être plus grande que la taille maximale de la mémoire physique? Qu'arrive-t-il lorsque la mémoire physique est pleine, et que l'on veuille charger un nouveau programme?

   Solution: La mémoire virtuelle peut effectivement simuler la disponibilité d'une plus grande quantité de mémoire qu'il en est réellement disponible. Toutes les pages d'un même programme n'ont pas à être chargées en mémoire en même temps: seules celles qui sont nécessaires à l'exécution le sont. Cela permet donc de charger plusieurs morceaux de plusieurs programmes en même temps.

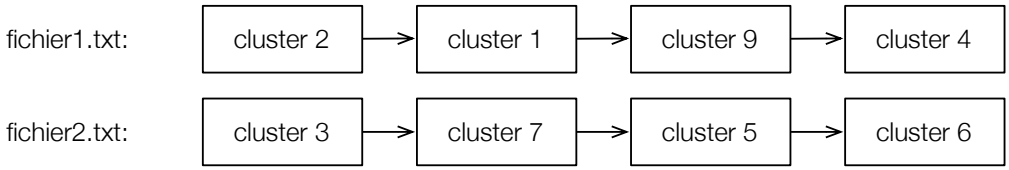
Lorsque la mémoire est pleine et que l'on veut charger un nouveau programme, on a une faute de page. Il faut trouver une (ou plusieurs) page en mémoire à remplacer. La page à remplacer (par exemple la moins utilisée) est transférée sur le disque dur pour laisser place à la nouvelle page.

4. (15 points) Tout comme la mémoire RAM doit être allouée à des processus, la mémoire disponible sur le disque dur doit aussi être allouée pour des fichiers. La mémoire d'un disque dur est divisée en blocs nommés "clusters".

   (a) (2 points) Dans l'allocation chaînée, chaque cluster possède un pointeur vers le cluster suivant dans le fichier. Quel est le désavantage de cette technique, comparativement à l'allocation chaînée avec table d'allocation (FAT)?

   Solution: L'accès aléatoire est très long: pour accéder à un cluster au milieu d'un fichier, par exemple, il faut parcourir tous les clusters, un à un, jusqu'à ce qu'on parvienne au cluster désiré. Comme les clusters sont sur le disque, il faut faire plusieurs accès disque pour y parvenir, ce qui est très long.

   (b) (7 points) Quelles seront la table de répertoire et la table d'allocation correspondant aux deux fichiers suivants, représentés ici en allocation chaînée? Note: vous n'avez pas à inscrire les informations supplémentaires de la FAT habituellement présentes dans la table de répertoire, telles que le temps de modification, le dernier accès, etc.



Solution:

table des répertoires

fichier	1er cluster
fichier1.txt	2
fichier2.txt	3

table des pages

cluster ID	cluster suivant
1	9
2	1
3	7
4	~
5	6
6	~
7	5
8	
9	4
10	

   (c) (3 points) Si la taille du disque dur est de 32Go, et que chaque cluster occupe 2Ko, quelle sera la taille de la table d'allocation (FAT)?

Solution: Il y a 32Go/2Ko = 16 777 216 clusters sur le disque. Nous aurons besoin de $\log_2(16\,777\,216) = 24\text{bits} = 3\text{octets}$ pour représenter un cluster. La taille de la FAT sera de $16\,777\,216 \times 3\text{o} = 48\text{Mo}$.

Si on assume que seuls FAT16 et FAT32 existent (comme c'est le cas en pratique), alors il nous faudra 4 octets pour représenter un cluster. La taille de la FAT sera alors de $16\,777\,216 \times 4\text{o} = 64\text{Mo}$.

(d) (3 points) Les systèmes d'exploitation modernes emploient des stratégies d'allocation basées sur l'allocation indexée. Expliquez pourquoi l'allocation indexée est préférée à l'allocation chaînée avec table d'allocation.

Solution: L'allocation indexée est similaire à l'allocation chaînée, sauf qu'on stocke une table par fichier, au lieu d'une table pour tous les fichiers. L'avantage est que l'on doit charger en mémoire seulement les tables pour les fichiers nécessaires, donc la mémoire est utilisée plus efficacement.

5. (15 points) Nous voudrions programmer un transfert de données de la mémoire vers le disque dur. Un programmeur écrit le code suivant pour effectuer le transfert d'une quantité de mots de 32 bits indiqués par la variable `NombreDeMotsACopier`.

```
; Requête au disque dur pour la réception de données
LDR R0, =RequeteTransfertDisqueDur
MOV R1, #1
STR R1, [R0]              ; Effectue la requête en envoyant un '1' au disque dur

; Attente que le disque dur soit prêt à recevoir les données
LDR R0, =EtatDuDisqueDur
TestSiDisqueDurPret:
LDR R1, [R0]              ; Lire l'état du disque dur
TST R1, #1                ; Le bit 1 de l'état du disque dur indique s'il est prêt
BNE TestSiDisqueDurPret   ; Si le disque dur n'est pas prêt

; Transfert de données
LDR R0, =AdresseDebutACopier          ; Adresse mémoire du premier mot à copier
LDR R3, =AdresseDestinationDisqueDur  ; Adresse mémoire du disque dur où copier les données
LDR R1, NombreDeMotsACopier           ; Variable contenant le nombre de mots à copier
MUL R1, R1, #4                        ; Calcul du nombre d'octets à copier

MOV R2, #0                ; Compteur de boucle

CopieMemoireDisqueDur:
LDR R4, [R0, R2]          ; Charger le mot à copier dans R2
STR R4, [R3, R2]          ; Écrire le mot à copier
ADD R2, R2, #4
CMP R2, R1
BNE CopieMemoireDisqueDur ; Est-ce que le transfert est terminé?

; et le programme continue...
```

(a) (5 points) Décrivez une alternative à la stratégie employée ci-haut pour transférer des données de la mémoire vers le disque dur.

> **Solution:** Entrées-sorties par interruption. Dans ce cas, le disque dur émet une interruption lorsqu'il est prêt. Le microprocesseur n'a donc pas à effectuer la boucle `TestSiDisqueDurPret`. On remplace par une ISR, qui sera appelée lorsque l'interruption sera levée. Cette ISR effectuera l'opération de charger le mot à copier dans R2, et de l'écrire à l'adresse correspondante. L'avantage est qu'on n'a pas à attendre après le disque dur.

(b) (5 points) Décrivez une seconde alternative à la stratégie employée ci-haut transférer des données de la mémoire vers le disque dur.

> **Solution:** Entrées-sorties par DMA. Dans ce cas, c'est le contrôleur de DMA qui se charge de transférer les données, libérant complètement le microprocesseur. Le microprocesseur initie le transfert en indiquant la plage de données à copier ainsi que l'adresse de destination au contrôleur de DMA.

(c) (5 points) Donnez un avantage et un inconvénient pour chacune des alternatives que vous avez écrites.

Solution: Avantage interruptions: le microprocesseur n'a plus besoin d'attendre après le disque dur, il peut faire autre chose!

Inconvénient interruptions: le microprocesseur doit quand même gérer le transfert.

Avantage DMA: le microprocesseur est complètement libéré. Il peut faire autre chose durant ce transfert.

Inconvénient DMA: l'architecture physique est plus compliquée: il faut rajouter le contrôleur de DMA. Les bus ne peuvent être partagés, donc le DMA doit attendre après le microprocesseur si ce dernier veut accéder aux bus.

6. (10 points) Répondez aux questions suivantes sur les caches.
   (a) (5 points) Sachant que:
      1. le temps d'accès à un mot en mémoire RAM est de 10ns;
      2. le temps d'accès à un mot en cache est de 1ns;
      3. les blocs en cache contiennent 4 mots,

      quel est le gain maximal apporté par l'emploi d'une cache, comparativement à l'accès aux mots en mémoire principale directement? Assumez que le temps de transfert d'un bloc entier entre la RAM et la cache est de 10ns, et qu'un bloc libre est disponible.

      Solution: Le transfert de données est $(10+10+10+10)/(10+1+1+1+1) = 2,85$ fois plus rapide avec une cache. Le 10 au dénominateur dénote le temps de transfert du bloc manquant entre la RAM et la cache.

   (b) (5 points) Expliquez ce qu'il faut faire lorsque l'on veut lire une adresse qui n'est pas présente en cache, et que la stratégie employée est le "write-back". Vous pouvez assumer un seul niveau de cache.

      Solution: Il faut:

      1. Trouver un bloc de libre. Pour cela, on peut par exemple choisir le bloc ayant été accédé il y a le plus longtemps;
      2. Déterminer si ce bloc est sale. S'il est sale, il faut écrire son contenu en mémoire.
      3. On peut ensuite copier le contenu du nouveau bloc de la mémoire vers la cache.
      4. Finalement, on lit la donnée en cache et on la retourne au microprocesseur.

7. (15 points) La liste suivante indique des processus en attente d'allocation mémoire (dans l'ordre de leur arrivée), ainsi que leur taille. La taille totale de la mémoire du système est de 64Mo.
   1. P1, 7Mo
   2. P2, 2Mo
   3. P3, 10Mo
   4. P4, 8Mo
   5. P5, 5Mo

Commençons tout d'abord par analyser le cas d'une architecture utilisant la stratégie d'allocation par mémoire contigüe avec partitions de taille fixe. Les partitions ont une taille de 8Mo. Pour cet exemple, le système ne peut pas allouer de l'espace pour un processus si sa taille dépasse celle d'une partition.

(a) (1 point) Est-ce que le système parvient à allouer de l'espace à tous les processus? Sinon, quel processus ne peut être chargé?

Solution: Non, P3 est trop gros et ne peux être alloué.

(b) (4 points) Quelle sera la fragmentation interne totale lorsque tous les processus seront en mémoire? La fragmentation externe?

Solution: La fragmentation interne totale sera de $(8-7)+(8-2)+(8-8)+(8-5) = 1+6+3 = 10$Mo. La fragmentation externe sera de 0.

Analysons maintenant le cas d'un autre système, toujours à mémoire contigüe, mais qui utilise cette fois des partitions de taille variable.

(c) (2 points) Quelle sera la fragmentation interne totale lorsque tous les processus seront en mémoire? La fragmentation externe?

Solution: La fragmentation (interne et externe) sera de 0.

Toujours en utilisant le système à mémoire contigüe avec partitions de taille variable, supposez maintenant que les processus P1 et P3 résidents en mémoire soient terminés. Par la suite, les nouveaux processus suivants surviennent, dans l'ordre:

1. P6, 8Mo
2. P7, 2Mo

(d) (4 points) Écrivez le contenu de la mémoire si l'algorithme suivant est utilisé pour allouer de l'espace aux nouveaux processus: meilleure allocation ("best fit")?

Solution:

Adresses	Processus
7–9	P2
9–17	P6
17–19	P7
19–27	P4
27–32	P5

(e) (4 points) Écrivez le contenu de la mémoire si l'algorithme suivant est utilisé pour allouer de l'espace aux nouveaux processus: prochaine allocation ("next fit")?

Solution:

Adresses	Processus
7–9	P2
19–27	P4
27–32	P5
32–40	P6
40–42	P7

8. (10 points) Vrai ou faux? (1 point par bonne réponse, -0.5 point par mauvaise réponse, 0 pour pas de réponse, score minimum de 0.)

   (a) (1 point) Dans une architecture avec 2 niveaux de cache, le microprocesseur peut aussi accéder à la mémoire principale directement, sans passer par les caches.

   Solution: Faux.

   (b) (1 point) Le bus USB 2.0 permet une communication bidirectionnelle (full-duplex).

   Solution: Faux.

   (c) (1 point) Les bus modernes adoptent, pour la plupart, un protocole de communication série.

   Solution: Vrai.

   (d) (1 point) Le bus USB 2.0 possède une ligne réservée pour les interruptions.

   Solution: Faux.

   (e) (1 point) C'est le système d'exploitation qui gère la table des vecteurs d'interruption dans un ordinateur.

   Solution: Vrai.

   (f) (1 point) Ce qui distingue une architecture "super-scalaire" d'une architecture "scalaire" est l'emploi de plusieurs niveaux de cache.

   Solution: Faux.

   (g) (1 point) Le DOS est un système d'exploitation non-préemptif, c'est-à-dire qu'il ne peut pas interrompre le programme en exécution.

   Solution: Vrai.

   (h) (1 point) L'un des rôles du BIOS est de tester les composantes internes de l'ordinateur.

   Solution: Vrai.

   (i) (1 point) Afin d'accélérer les performances, l'architecture de Harvard sépare l'architecture bus-mémoire en deux: une portion pour les entrées, l'autre pour les sorties.

   Solution: Faux.

   (j) (1 point) Le parallélisme peut être atteint au niveau des instructions ("Instruction Level Parallelism" ou ILP), des processus ("Thread Level Parallelism" ou TLP), ainsi que par une combinaison des deux.

Solution: Vrai.

A Annexe: Unités et logarithme

A.1 Unités

Petit rappel sur les unités:

$$\begin{array}{rlrr} 1\text{Ko} & = 2^{10} & = & 1\ 024 \text{ octets} \\ 1\text{Mo} & = 2^{20} = 1\ 024\text{Ko} & = & 1\ 048\ 576 \text{ octets} \\ 1\text{Go} & = 2^{30} = 1\ 024\text{Mo} & = & 1\ 073\ 741\ 824 \text{ octets} \end{array}$$

A.2 Logarithme en base 2

Il est facile de calculer des logarithmes en base 2 à partir de logarithmes dans une autre base N (ex: 10). Pour ce faire, appliquez l'équation suivante:

$$\log_2 x = \frac{\log_N x}{\log_N 2}$$

B Annexe: Instructions ARM et codes de conditions

Dans la table suivante, `Op1` dénote une opérande de type 1, et `Op2` une opérande de type 2.

Mnémonique	Description
`ADD Rd, Rs, Op1`	`Rd = Rs + Op1`
`ADC Rd, Rs, Op1`	`Rd = Rs + Op1 + Carry`
`AND Rd, Rs, Op1`	`Rd = Rs AND Op1`
`ASR Rd, Rs, #imm`	`Rd = Rs / 2`$^{\texttt{imm}}$
`Bcc Offset`	`PC = PC + Offset`, si `cc` est rencontré
`BLcc Offset`	Comme `B`, `LR` = Adr. de l'instr. suivante
`CMP Rs, Op1`	Change les drapeaux comme `Rs-Op1`
`LDR Rd, [Rs, Op2]`	`Rd = Mem[Rs + Op2]`
`LDR Rd, [Rs], Op2`	`Rd = Mem[Rs], Rs = Rs + Op2`
`LDR Rd, [Rs, Op2]!`	`Rs = Rs + Op2, Rd = Mem[Rs]`
`LSL Rd, Rs, #imm`	`Rd = Rs x 2`$^{\texttt{imm}}$
`MUL Rd, Rs, Op1`	`Rd = Rs x Op1`
`MVN Rd, Op1`	`Rd = !Op1` (inverse les bits)
`POP {Reg List}`	Met la liste de registres sur la pile
`PUSH {Reg List}`	Met la liste de registres sur la pile
`SBC Rd, Rs, Op1`	`Rd = Rs - Op1 - C`
`STR Rd, [Rs, Op2]`	`Mem[Rs + Op2] = Rd`
`STR Rd, [Rs], Op2`	`Mem[Rs] = Rd, Rs = Rs + Op2`
`STR Rd, [Rs, Op2]!`	`Rs = Rs + Op2, Mem[Rs] = Rd`
`SUB Rd, Rs, Op1`	`Rd = Rs - Op1`
`TST Rs, Op1`	Change les drapeaux comme `AND Rd, Rd, Op1`