

Name _____ section _____ date _____ grade _____ / 20

Part 1. From the article, "Where the bailout went wrong" by Neil Barofsky. Choose the best answer and put it on the EPITA answer sheet. The negative point rule will apply. ____ / 20 pts.

1. ____ In this phrase, "legislation that bailed out the banks," the expression "to bail out" means:
 - a. to run on schedule
 - b. to make a profit
 - c. to rescue
 - d. to pay back

2. ____ In this phrase, "the positive assessment is warranted," the word "warranted" means:
 - a. deserved
 - b. wasted
 - c. conserved
 - d. guaranteed

3. ____ An "extension of credit" means
 - a. Getting credit for a job well done
 - b. Getting a credit card
 - c. Allowing a bank to lend more money
 - d. Allowing someone to borrow more money

4. ____ The Treasury Dept gave money to the banks without a policy that compelled them to extend credit to borrowers.
 - a. true
 - b. false

5. ____ To understate something means
 - a. To say that something is smaller than it really is
 - b. To say that something is not important
 - c. To say that something is small in a discreet way
 - d. To say that something has been stated

6. ____ The word "incentive" is closest in meaning to:
 - a. encouragement
 - b. compensation
 - c. punishment
 - d. pressure

7. ____ In the sentence "The act's goal of helping homeowners was shelved," "to shelf" here means:

- a. To put something on a shelf
 - b. To keep something for a future purpose
 - c. To keep something safe
 - d. To hide something
8. ____ If you file for foreclosure, that means you
- a. Sell your house
 - b. Go from one bank to another
 - c. Declare that you cannot pay your mortgage
 - d. Declare that your bank does not want your mortgage
9. ____ It was therefore no surprise that lending did not increase but rather continued to decline well into the recovery.
- a. true
 - b. false
10. ____ In the sentence in #9, "recovery" here means
- a. recession
 - b. to get something back that you borrowed
 - c. return to a healthy economy
 - d. return to a market economy

Part 2. Short Answers. Answer the following questions with short clear answers. (5 points for each question for a total of 20 pts) _____ / 20

1. What is the stock exchange and how does it work?

2. What is the principal of microcredit and how does it work?

3. Give some reasons that motivated the E. U. to create a single currency.

4. Define the term “bailout” in the context of the US financial crisis.

Part 3. Essay. Choose **one** of the two following subjects and answer below.

20 pts.

1. The success and omnipresence of Coca-Cola as a product perfectly symbolizes the Americanization of France. Do you agree? Explain.
2. Describe the health insurance issue in the US. In general, how is the current system different from the French system? Why are there so many Americans that don't have health insurance today?

This image shows a single sheet of white paper with horizontal blue or grey ruling lines. The lines are evenly spaced and run across the width of the page. There are approximately 20 lines visible. The paper appears to be a standard notebook page or a sheet of stationery. There is no handwriting or other markings on the page.



Explain what is (or is supposed to be) funny about this cartoon.

____ 10 pts

Bonus question. _____ 10 points max.

_____ 10 points max.

Refer to the songs mentioned in the final exposés on music and say whether you think music can change the world, or people in general.

(Any points given here will be added to the total.)

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Algorithmique

Partiel n° 2

INFO-SPÉ – EPITA

D.S. 312354.45 BW (10 mai 2011 - 9 :00)

Consignes (à lire) :

- ☐ Vous devez répondre sur **les feuilles de réponses prévues à cet effet**.
 - Aucune autre feuille ne sera ramassée (gardez vos brouillons pour vous).
 - Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées** : utilisez des brouillons !
 - Ne séparez pas les feuilles à moins de pouvoir les ré-agrafer pour les rendre.
 - Aucune réponse au crayon de papier ne sera corrigée.
 - ☐ La présentation est notée en moins, c'est à dire que vous êtes noté sur 20 (le barème est sur 30 mais la note sera ramenée à 20) et que les points de présentation (2 au maximum) sont retirés de cette note.
 - ☐ **Les algorithmes :**
 - Tout algorithme doit être écrit dans le langage ALGO (pas de C, CAML ou autre).
 - Tout code ALGO non indenté ne sera pas corrigé.
 - Tout ce dont vous avez besoin (types, routines) est indiqué en **annexe** (dernière page) !
 - ☐ Durée : 3h00
-

Exercice 1 (Gisement épuisant... – 5 points)

Des mineurs veulent sécuriser la circulation entre les différents points d'extraction (représentés par des sommets) reliés par des galeries (figure 1). Tous ces points d'extraction sont accessibles séparément depuis l'extérieur et le réseau est suffisamment complexe pour qu'en général plusieurs itinéraires permettent de passer d'un point d'extraction à un autre. Il n'est donc pas nécessaire que chaque galerie soit sécurisée pour qu'il existe toujours entre deux points d'extraction au moins un itinéraire qui le soit.

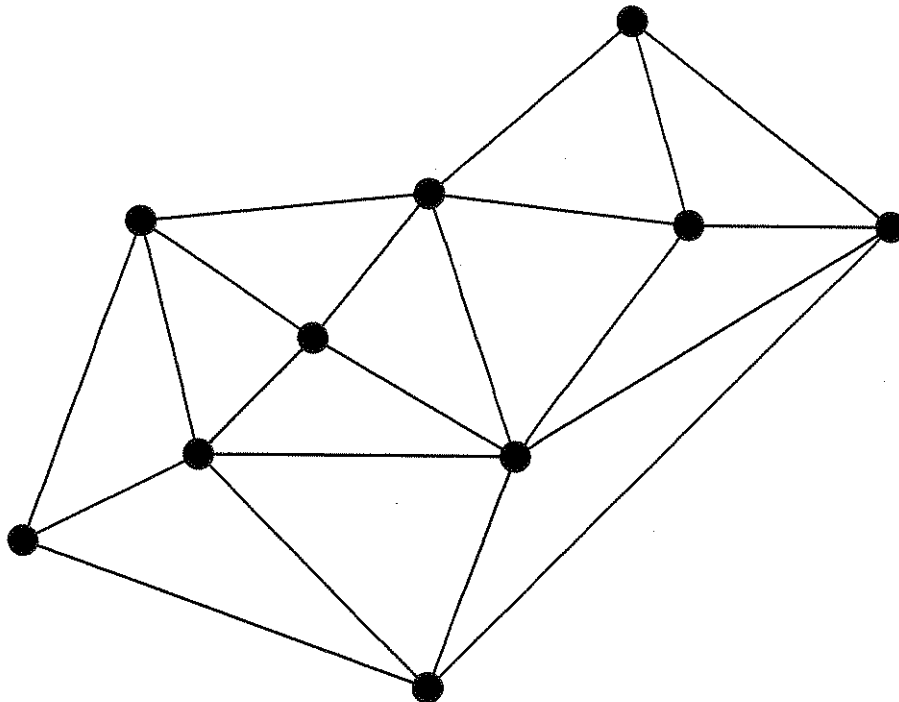


FIGURE 1 – Graphe associé à une mine et son réseau.

1. On veut déterminer le plus petit nombre de galeries à sécuriser : à quoi correspond la solution en termes de graphes ?
2. Dans le cas de la figure 1, combien faut-il sécuriser de galeries ?
3. Proposer une solution graphique (Surligner les galeries que vous vous proposez de sécuriser).
4. En généralisant à un réseau de N points d'extraction, combien faudrait-il sécuriser de galeries ?
5. Justifier cette réponse.

On affine l'analyse du problème : Pour chaque galerie, nous avons évalué le coût des travaux de sécurisation (voir figure 2).

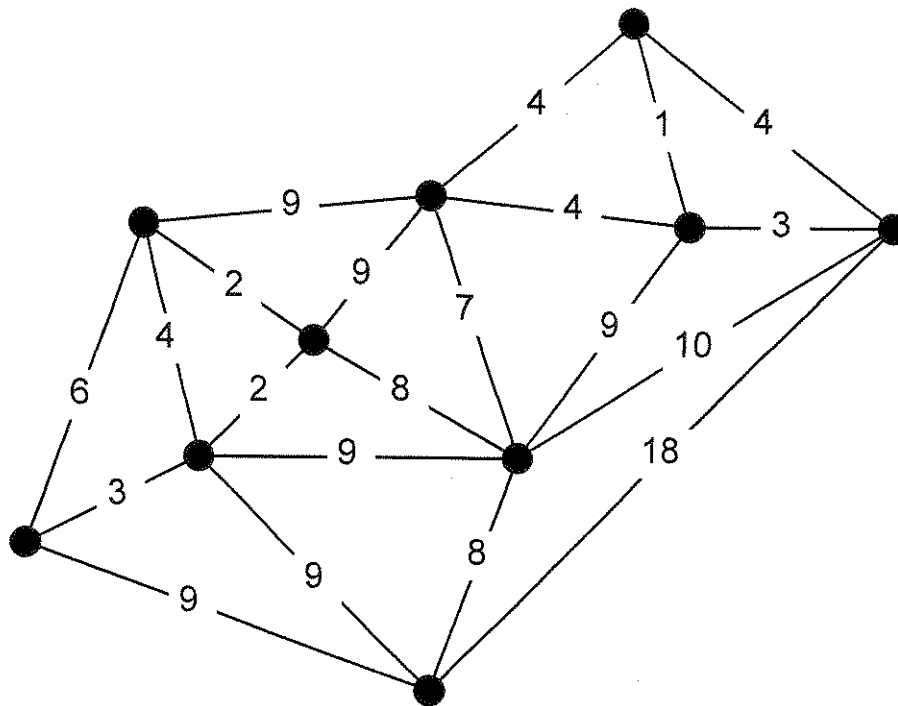


FIGURE 2 – Graphe valué associé à la sécurisation des galeries d'un réseau de grottes.

6. Comment dans ce cas sécuriser l'accès à toutes les grottes au moindre coût ?
7. Proposer une solution graphique (Tracer en gras les galeries qu'il faut sécuriser).
8. Cette solution est-elle unique ?
9. Pourquoi ?

Exercice 2 (Mangez des crêpes – 16 points)

Après avoir creusé, une bonne crêpe pour se remettre ?

Le but ici est de savoir dans combien de temps nous pourrons manger une crêpe à la banane flambée.

Ci-dessous la recette, avec pour chaque tâche sa durée en secondes.

La recette	Durée (en sec.)	Réf.
Mettre la farine dans un saladier,	3	A
ajouter deux œufs,	30	B
ajouter le lait doucement et mélanger.	600	C
Dans une poêle mettre du rhum.	3	D
Couper les bananes en fines lamelles,	300	E
les mélanger au rhum.	30	F
Faire chauffer le mélange,	120	G
faire flamber le mélange.	10	H
Faire cuire une crêpe,	10	I
verser du mélange bananes-rhum sur la crêpe.	10	J

Quelques précisions concernant l'ordre des tâches :

- la préparation de la pâte à crêpe et celle du mélange rhum-banane peuvent se faire en parallèle ;
- ce n'est que lorsque la crêpe sera cuite et que le mélange sera prêt qu'on pourra verser du mélange sur la crêpe et enfin la déguster ;
- les autres étapes se réalisent séquentiellement (on doit mettre la farine avant les œufs, on doit mettre le rhum avant les bananes).

1. Modéliser la recette sous forme de graphe :

- Les sommets sont les tâches.
- Les tâches *debut* et *fin* représentent le début et la fin du projet.
- La représentation du graphe doit être planaire¹ !

La graphe qui représente la recette est sans circuit. De plus, tous les sommets sont atteignables depuis un sommet donné (ici la commande de la crêpe = la tâche *debut*, sommet n° 1 dans la représentation machine). Le graphe sera ici en représentation dynamique. Dans toute la suite de l'exercice, le graphe aura ces spécifications !

2. **Le cuisinier est tout seul en cuisine** : la durée minimum est donc la somme des temps nécessaires à chaque tâche. Par contre, il faut l'aider à trouver l'ordre dans lequel il va pouvoir faire les différentes tâches : la solution est un tri topologique du graphe.

- (a) Une solution de tri topologique évidente est donnée par l'ordre de la recette. Donner une autre solution : compléter celle donnée sur les feuilles de réponses.
- (b) Comment utiliser un parcours en profondeur pour trouver une solution de tri topologique dans un graphe de projet ?
- (c) Écrire l'algorithme correspondant : la solution de tri topologique doit être stockée dans une pile de pointeurs (le graphe est représenté en dynamique).

3. **Le cuisinier a trouvé des aides, les tâches peuvent donc être réalisées en parallèle** :

- (a) La date au plus tôt de la tâche *i* est la date à laquelle la tâche peut commencer au plus tôt. Comment calculer les dates au plus tôt de chaque tâche à partir de ce type de graphe ? Remplir le tableau donnant les dates au plus tôt pour la recette de la crêpe.
- (b) Quel temps faudra-t'il au minimum pour pouvoir déguster notre crêpe ? Comment obtenir cette date (la durée minimale du projet) ?
- (c) La date au plus tard de la tâche *i* est la date au plus tard où la tâche peut commencer sans entraîner de retard sur le projet. Comment calculer les dates au plus tard de chaque tâche ? Remplir le tableau donnant les dates au plus tard pour la recette de la crêpe.
- (d) Comment obtient-on les tâches critiques dans ce type de projet ? Donner les tâches critiques pour le présent projet.
- (e) Écrire l'algorithme qui détermine le temps minimum de réalisation d'un projet représenté par un graphe ayant les mêmes spécifications que ci-dessus. Vous devez obligatoirement utiliser l'algorithme de la question 2.

1. Pour mémoire, cela signifie que les arcs ne doivent pas se couper !

Exercice 3 (Construire un ARPM par suppression – 9 points)

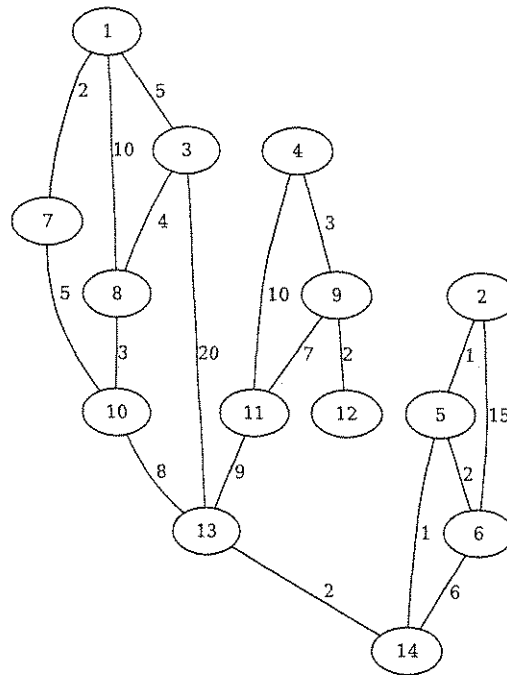


FIGURE 3 – Graphe valué non orienté

Le but ici est de construire un ARPM (Arbre de Recouvrement de Poids Minimum) sur un principe *inverse* de celui de l'algorithme de Kruskal : au lieu d'ajouter des arêtes dans l'arbre en construction en partant d'un arbre vide, on va supprimer les arêtes inutiles en partant du graphe d'origine. Au final, cet algorithme construit un arbre (graphe non-orienté sans cycle et connexe) dont la somme des poids est minimum.

Pour cet algorithme il nous faut la liste des arêtes triées mais dans l'ordre décroissant (du plus fort au plus faible coût). Ensuite, pour que le résultat reste un arbre, il faut s'assurer que les arêtes supprimées ne sont pas nécessaires à la connexité.

Nous exprimerons le résultat de l'algorithme avec la *liste* des arêtes supprimées représentée (la liste) par une matrice T telle que si $T[i, j] \neq 0$ alors l'arête (i, j) a été supprimée (bien évidemment, comme le graphe est non-orienté, on vérifie que $\forall i, j \ T[i, j] = T[j, i]$).

On supposera que le graphe en entrée est connexe. On supposera également que la liste des arêtes triées est déjà construite. Elle est fournie à notre algorithme sous la forme d'un ensemble correspondant à la spécification suivante :

types

```
arete = ^s_arete
s_arete = enregistrement
  t_listsom      src, dst
  reel           cout
fin enregistrement s_arete
ensemble = /* Type "opaque" */
```

Opérations

```
/* l'ensemble est-il vide */
est_vide(ensemble e) : boolean
/* nombre d'éléments de l'ensemble */
card(ensemble e) : entier
/* Renvoie et supprime l'élément maximum de l'ensemble e */
supprime_max(ensemble e) : arete
```

1. Donner le principe d'un algorithme qui, à l'aide d'un parcours profondeur, détermine si deux sommets sont connectés dans un graphe.
2. Écrire la fonction `lie_rec(dst,ps,T,M)` qui teste (elle renvoie donc un booléen) s'il existe un chemin depuis le sommet pointé par `ps` jusqu'au sommet de numéro `dst`. La fonction utilise le principe de la question précédente. La matrice `T` décrit les arêtes qui ont été supprimées du graphe et le vecteur de booléens `M` sert de vecteur de marques pour le parcours profondeur. Cette fonction sera appelée par la fonction suivante :

```
algorithme fonction lie : booléen
parametres locaux
  t_graphe_d      g
  entier          dst
  t_listsom       ps
  t_mat_entiers   T
variables
  t_vect_booléens M
  entier         i
debut
  pour i ← 1 jusqu'à g.ordre faire
    M[i] ← faux
  fin pour
  retourne (lie_rec(dst, ps, T, M))
fin algorithme fonction lie
```

3. Lorsque l'on supprime les arêtes du graphe, comment sait on que l'on peut arrêter l'algorithme?
4. Écrire la procédure `revdel(g,E,T)` qui applique l'algorithme de construction de l'ARPM par suppression des arêtes dans le graphe `g`, à partir de l'ensemble d'arête `E` et indique dans la matrice `T` les arêtes supprimées.
5. Donner la liste des arêtes supprimées par cet algorithme appliqué au graphe de la figure 3.

Annexes

Représentation dynamique des graphes :

```
types
  t_listsom = ↑ s_som
  t_listadj = ↑ s_ladj
  s_som      = enregistrement
    entier    som
    t_listadj succ
    t_listadj pred
    t_listsom suiv
  fin enregistrement s_som
  s_ladj      = enregistrement
    t_listsom vsom
    reel      cout
    t_listadj suiv
  fin enregistrement s_ladj
  t_graphe_d = enregistrement
    entier  ordre
    booléen orient
    t_listsom lsom
  fin enregistrement t_graphe_d
```

Autres types utiles :

```
constantes
  Max = /* une valeur suffisante ! */
types
  t_vect_entiers = Max entier
  t_vect_reels   = Max reels
  t_vect_booleens = Max booléen
  t_mat_entiers  = Max × Max entier
  t_mat_reels    = Max × Max reel
```

Routines autorisées :

Piles : le type t_pile

Toutes les opérations sur les piles peuvent être utilisées à condition de préciser le type des éléments.

- `pile_vide () : t_pile`
- `est_vide (t_pile p) : booléen`
- `empiler(t_elt_pile elt, t_pile p) : t_pile`
- `depiler (t_pile p) : t_pile`
- `sommet (t_pile p) : t_elt_pile`

Autres

Les fonctions *max*, *min*, *abs*, ainsi que les valeurs ∞ et $-\infty$ sont aussi autorisées. De même pour la fonction *recherche* (*entier s*, *t_graphe_d G*) qui retourne le pointeur sur le sommet *s* dans le graphe *G* (de type *t_listsom*).

Nom	
Prénom	
Groupe	

Note	
------	--

Algorithmique - Info-SPE

Partiel n° 2

D.S. 312354.45 BW (10 mai 2011 - 09 :00)

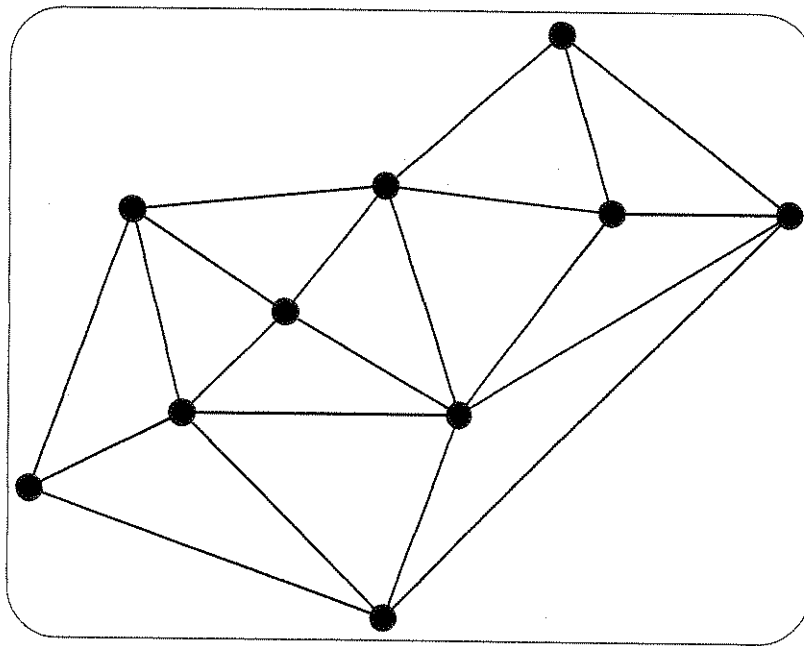
Feuilles de réponses

Réponses 1 (Gisement épuisant... - 5 points)

1. La solution correspond à :

2. Dans le cas de la figure 1, combien faut-il sécuriser de galeries ?

3. Proposer une solution graphique (Surligner les galeries que vous vous proposez de sécuriser).



4. Pour un réseau de N points d'extraction, il faut sécuriser

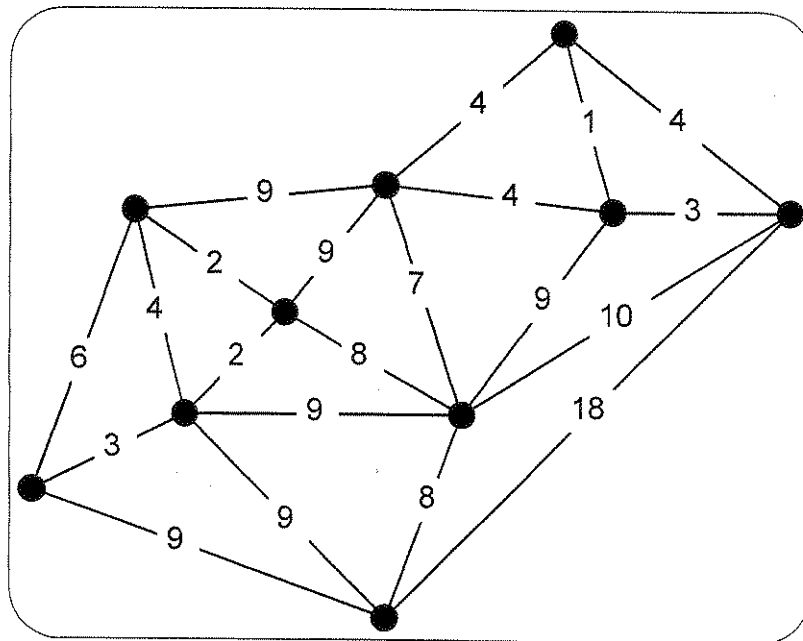
galeries.

5. Justification :

On affine l'analyse du problème : pour chaque galerie, nous avons évalué le coût des travaux de sécurisation (voir figure 2).

6. Comment dans ce cas sécuriser l'accès à toutes les grottes au moindre coût ?

7. Proposer une solution graphique (Tracer en gras les galeries qu'il faut sécuriser).



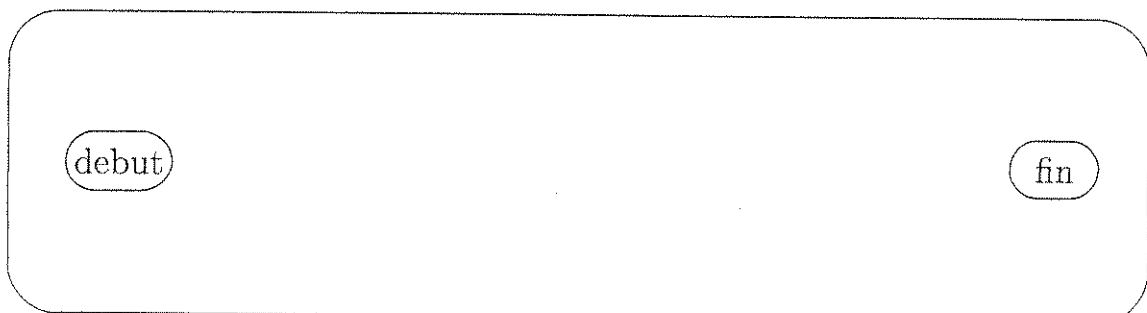
8. La solution est-elle unique ? OUI – NON

9. Justification :

-
-

Réponses 2 (Mangez des crêpes – 16 points)

1. Graphe représentant le projet :



- debut* - D - - - B - - - G - - - *fin*

- 1

- Algorithme (parcours récursif) :

This image shows a full page of blank graph paper. The grid consists of small, equal-sized squares formed by thin black lines. There are 20 columns and 20 rows of squares, creating a total of 400 square units. The lines are evenly spaced and extend across the entire area of the page, leaving no margins or additional markings.

3

debut

[illegible]

3. (a) Comment calculer les dates au plus tôt de chaque tâche à partir de ce type de graphe ?

[illegible][illegible]

(b) Durée minimale avant de pouvoir déguster la crêpe :

Comment obtenir la durée minimale du projet ?

(c) Comment calculer les dates au plus tard de chaque tâche ?

Dates au plus tard pour la recette :

debut	A	B	C	D	E	F	G	H	I	J	fin
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

(d) Comment obtient-on les tâches critiques dans ce type de projet ?

Tâches critiques pour le présent projet : une croix (X) pour chaque tâche critique !

debut	A	B	C	D	E	F	G	H	I	J	fin
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

(e) **Spécifications** : La fonction `duree_minimale (G, source, finale)` calcule la longueur du plus long chemin (la durée minimale du projet) dans le graphe G , entre les tâches *source* et *finale*.

algorithme fonction duree_minimale : reel

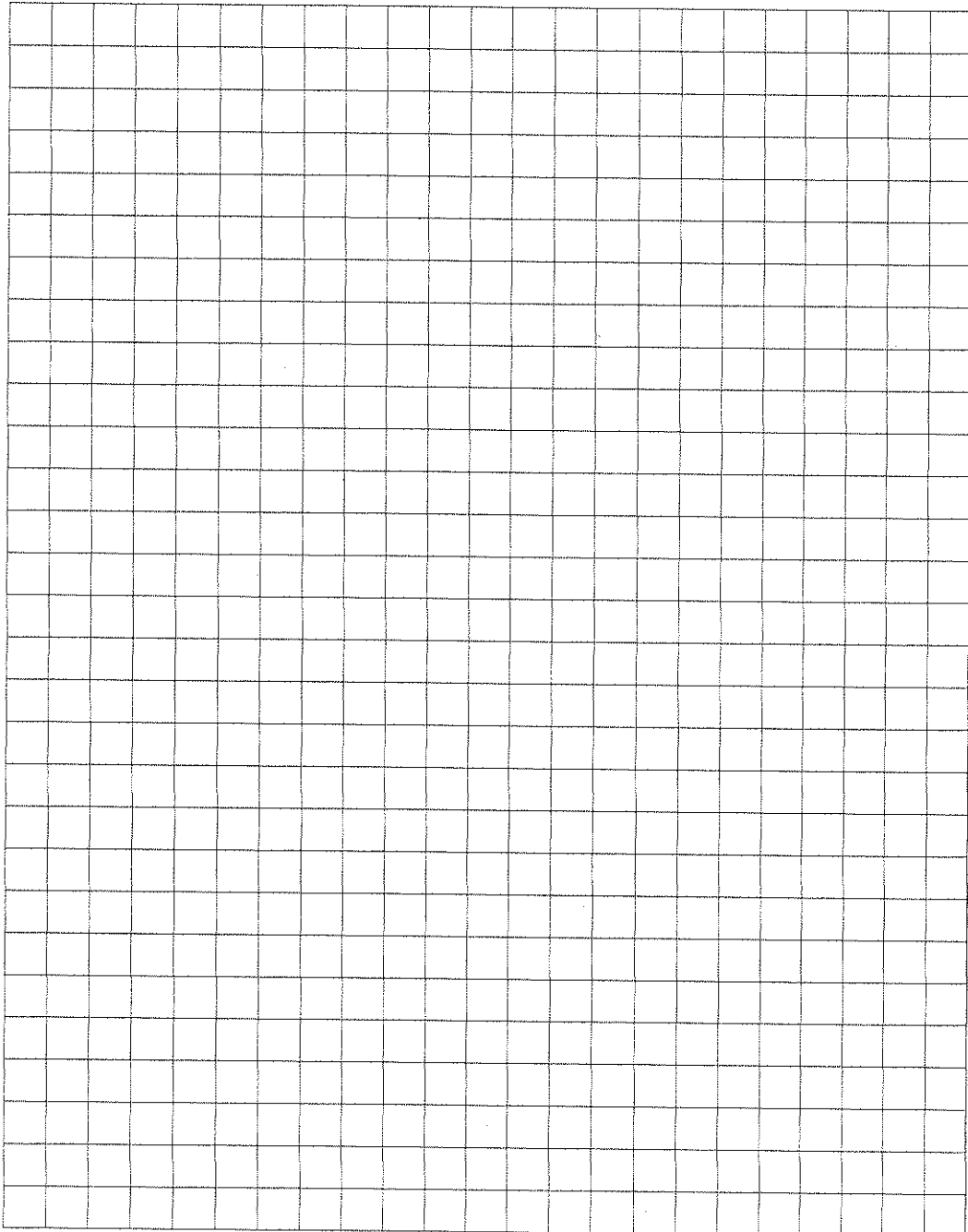
parametres locaux

t_graphe_d G

entier s, f

variables

debut



fin algorithme fonction duree_minimale

1. Principe du test de connexité entre deux sommets par un parcours profondeur :

- ```

algorithme fonction lie_rec : booléen
 paramètres locaux
 entier dst
 t_listsom ps
 t_mat_entiers T
 paramètres globaux
 t_vect_booléens M
 variables

```

A full-page view of a blank sheet of white graph paper. The grid consists of thin, light gray horizontal and vertical lines intersecting at right angles to form small squares. There are approximately 20 columns and 20 rows of squares across the page. The margins are uniform on all sides.

7

3. Condition d'arrêt de l'algorithme de suppression des arêtes :

4. **Spécification :** la procédure `revdel(g,E,T)` applique l'algorithme de construction de l'ARPM par suppression des arêtes dans le graphe  $g$ , à partir de l'ensemble d'arête  $E$  et indique dans la matrice  $T$  les arêtes supprimées.

```

algorithm procedure revdel
 parametres locaux
 t_graphe_d g
 parametres globaux
 ensemble E
 t_mat_entiers T
 variables

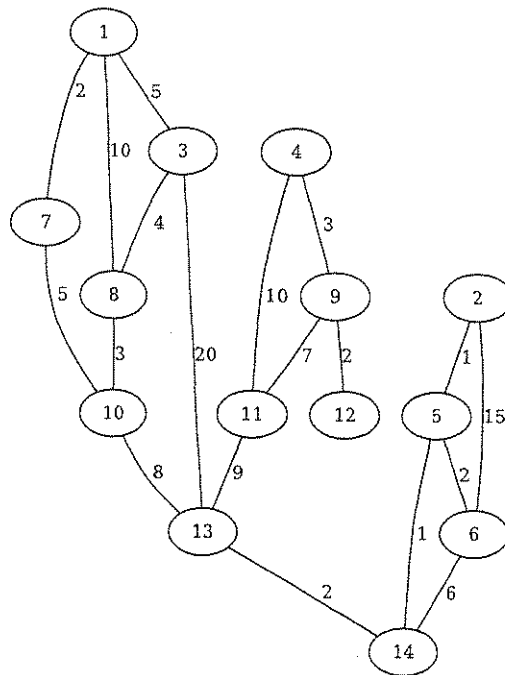
```

debut

This image shows a full page of blank graph paper. The grid consists of small, equal-sized squares formed by thin black lines. There are approximately 20 columns and 20 rows of squares across the entire page. The background is white, and the lines are evenly spaced both horizontally and vertically.

```
fin algorithme procedure revdel
```

5. Surligner les arêtes supprimées par l'algorithme



*Partiel n°2 de Physique*  
*Documents et calculatrice non autorisés*

**Partie cours** (sur 5 points)

Les questions sont indépendantes

- 1- a) Interpréter l'expérience de Davisson et Germer, en précisant les deux principaux résultats de cette expérience.  
b) Retrouver la relation de Bragg donnée par :  $2d \sin(\theta) = n\lambda$
- 2- a) Interpréter le principe d'incertitude de Heisenberg.  
b) Donner un exemple d'expérience qui illustre ce principe.
- 3- Donner la configuration électronique des éléments atomiques suivants, en précisant les différents nombres quantiques et en représentant les cases quantiques.  
Chrome:  $\text{Cr}^{3+}$  ( $Z = 24$ )  
Zirconium Zr ( $Z = 40$ ).

**Exercice 1**      ***Physique atomique***      (sur 5 points)

Le modèle de Bohr a permis d'exprimer l'énergie de tous les niveaux de l'atome d'hydrogène. Cette énergie s'exprime en fonction du nombre quantique principal  $n$

- 1- Rappeler l'expression de l'énergie  $E_n$  en fonction de  $n$
- 2- En déduire l'expression de la longueur d'onde de la transition du niveau supérieur d'énergie  $E_m$ , vers le niveau inférieur d'énergie  $E_n$  et qui s'écrit comme :

$$\frac{1}{\lambda_{mn}} = R_H \left( \frac{1}{n^2} - \frac{1}{m^2} \right)$$

Préciser l'expression de la constante de Rydberg  $R_H$ .

- 3- Donner en fonction de  $R_H$ , les longueurs d'onde des deux premières raies de la série de Lyman.
- 4- Exprimer la fréquence  $\nu$  du photon pouvant porter l'atome du niveau fondamental vers le 3<sup>ème</sup> état excité.
- 5- a) comment se réécrivent les deux expressions établies dans les questions 1 et 2 dans le cas d'un hydrogénoïde de numéro atomique  $Z$ . Justifier votre réponse.  
  
b) En déduire les longueurs d'onde en fonction de  $R_H$ , des deux premières raies de la série K de l'hydrogénoïde  $\text{B}^{4+}$ .

**Exercice 2****Mécanique quantique**

(sur 5 points)

On considère une particule libre à une dimension, dans un puits de potentiel de largeur  $L$ , tel que l'énergie potentielle vérifie :

$$E_p(x) = 0 \quad \text{Pour} \quad 0 < x < L$$

$$E_p(x) \rightarrow \infty \quad \text{Pour} \quad x = 0 \text{ et } x = L$$

1) On montre que la fonction d'onde de cette particule est de la forme :

$$\Psi(x) = A \sin\left(\frac{n\pi}{L}x\right)$$

Utiliser l'équation de Schrödinger :  $H\Psi = E\Psi$  pour montrer que l'énergie de la particule est de la forme :

$$E_n = \frac{\hbar^2 \pi^2}{2mL^2} n^2$$

(On rappelle qu'à une dimension:  $\Delta \psi(x) = \frac{d^2 \psi}{dx^2}$ )

2- Utiliser la condition de normalisation pour exprimer la constante  $A$ .

$$\text{On donne : } \sin^2(\alpha) = \frac{1 - \cos(2\alpha)}{2}$$

**Exercice 3****Thermodynamique**

(sur 7 points) (dont 2 points Bonus)

On considère 1 mole de Gaz parfait dans l'état(1), définit par les conditions ( $P_1 = 10^5 \text{ Pa}$ ,  $V_1 = 24.10^{-3} \text{ m}^3$ ) et une température  $T_1$ .

On fait subir successivement à ce gaz :

- Une compression isotherme, qui le ramène vers l'état (2) définit par les conditions ( $P_2$ ,  $V_2$ ). Tel que  $V_2 = V_1/2$
- Une détente isobare, qui le ramène à son volume initial: état (3) de conditions ( $P_3$ ,  $V_3$ )
- Un refroidissement isochore, qui le ramène à l'état initial ( $P_1$ ,  $V_1$ ).

On donne :  $C_p = (7/2)R$  et  $C_v = (5/2)R$

- 1 - Représenter le cycle des 3 transformations dans le diagramme ( $P$ ,  $V$ )
- 2 - Calculer les températures  $T_1$  et  $T_2$  ainsi que la pression maximale  $P_2$
- 3 - Calculer le travail et la quantité de chaleur échangés par la mole de gaz au cours de chaque transformation. Faire le bilan du cycle.

N.B : Pour faciliter les calculs, on prend :  $R \approx 8 \text{ JK}^{-1}\text{mole}^{-1}$

## *Formulaire*

### 1- 3<sup>ième</sup> postulat de Bohr

$$\Delta E = h.\nu = \frac{h.c}{\lambda}$$

### 2- Opérateur Hamiltonien

$$H = -\frac{\hbar^2}{2m}\Delta + E_p$$

### 3- Probabilité de présence entre 0 et L

$$P = \int_0^L |\psi(x)|^2 dx$$

### 4- Energie interne d'un gaz parfait

$$dU = n.C_v.dT$$

### 5- 1<sup>er</sup> Principe de la thermodynamique

$$\Delta U = W + Q$$

### 6- Quantité de chaleur échangée

A pression constante :  $\delta Q = n.C_p.dT$

A volume constant :  $\delta Q = n.C_v.dT$



# Architecture des ordinateurs

## Partiel 2

Durée : 1h30

**Exercice 1 (9 points)**

Toutes les questions de cet exercice sont indépendantes. À l'exception des registres utilisés pour renvoyer une valeur de sortie, aucun registre ne devra être modifié en sortie de vos sous-programmes. Une chaîne de caractères se termine toujours par un caractère nul (la valeur zéro). On suppose pour tout l'exercice que les chaînes ne sont jamais nulles (elles possèdent au moins un caractère non nul).

1. Réalisez le sous-programme `IsNumber` qui détermine si une chaîne de caractères contient uniquement des chiffres.

Entrée : `AO.L` pointe sur une chaîne non nulle.

Sortie : Si la chaîne contient uniquement des chiffres, `DO.L` renvoie 0.  
Si la chaîne contient d'autres caractères que des chiffres, `DO.L` renvoie 1.

2. Réalisez le sous-programme `GetSum` qui additionne tous les chiffres présents dans une chaîne de caractères.

Entrée : `AO.L` pointe sur une chaîne non nulle contenant uniquement des chiffres.

Sortie : `DO.L` renvoie la somme de tous les chiffres de la chaîne.

Exemple :

|                 |   |     |     |     |     |     |     |     |     |     |   |
|-----------------|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|---|
| <code>AO</code> | → | '7' | '0' | '4' | '8' | '9' | '4' | '2' | '0' | '3' | 0 |
|-----------------|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|---|

`DO` doit renvoyer la valeur 37 ( $37 = 7 + 0 + 4 + 8 + 9 + 4 + 2 + 0 + 3$ ).

Indications :

Réalisez une boucle qui pour chaque caractère de la chaîne :

- Copie le caractère en cours dans le registre `D1.B`.
- Convertit le caractère en une valeur numérique.
- Ajoute la valeur numérique du caractère au registre `DO.L`.

3. À l'aide des sous-programmes `IsNumber` et `GetSum`, réalisez le sous-programme `Checksum` qui renvoie la somme des chiffres d'une chaîne de caractères.

Entrée : `AO.L` pointe sur une chaîne non nulle.

Sortie : Si la chaîne contient uniquement des chiffres :

`DO.L` renvoie 0 et `D1.L` renvoie la somme.

Si la chaîne contient d'autres caractères que des chiffres :

`DO.L` renvoie 1 et `D1.L` renvoie 0.

**Exercice 2 (2 points)**

Codez les instructions suivantes en langage machine 68000, vous détaillerez les différents champs puis vous exprimerez le résultat final sous forme hexadécimale en précisant la taille des mots supplémentaires lorsque le cas se présente.

1. MOVE.B \$19, 29(A3)
2. MOVE.W #\$19, 29(A4, D6.L)

**Exercice 3 (2 points)**

Vous indiquerez après chaque instruction, le nouveau contenu des registres (sauf le PC) et/ou de la mémoire qui viennent d'être modifiés. Vous utiliserez la représentation hexadécimale.

Attention : La mémoire et les registres sont réinitialisés à chaque nouvelle instruction.

Valeurs initiales : D0 = \$FFFFFFFB A0 = \$00005000 PC = \$00006000  
 D1 = \$FFFF0003 A1 = \$00005008  
 D2 = \$FFFFE000 A2 = \$00005010

\$005000 54 AF 18 B9 E7 21 48 C0  
 \$005008 C9 10 11 C8 D4 36 1F 88  
 \$005010 13 79 01 80 42 1A 2D 48

1. MOVE.W #50, -(A1)
2. MOVE.B \$5002(PC), -2(A2, D0.L)

**Exercice 4 (4 points)**

Soit le sous-programme `Select` qui utilise comme registre d'entrée `D1.B` et comme registre de sortie `D0.L`.

```
Select tst.b d1
 bne.s suite1
 move.l #200, d0
 rts
suite1 bmi.s suite3
 cmp.b #$61, d1
 blt.s suite2
 move.l #400, d0
 rts
suite2 move.l #600, d0
 rts
suite3 move.l #800, d0
 rts
```

1. Quelle valeur renvoie `Select` pour une valeur d'entrée égale à 18 ?
2. Quelle valeur renvoie `Select` pour une valeur d'entrée égale à -5 ?
3. Quelle valeur renvoie `Select` pour une valeur d'entrée nulle ?
4. Quelle valeur renvoie `Select` pour une valeur d'entrée égale à 96 ?

**Exercice 5 (3 points)**

En utilisant l'extrait du DataSheet fourni en annexe, déterminez pour le PIC 12F509 :

1. La taille d'une donnée.
2. Le nombre et la taille des mots programme.
3. Le nombre et le(s) type(s) de mémoire. Précisez dans chacun des cas la particularité (volatile ou non), la taille de la mémoire et les informations stockées dans cette mémoire.

## Integer Instructions

**MOVE****Move Data from Source to Destination**  
(M68000 Family)**MOVE****Operation:** Source → Destination**Assembler****Syntax:** MOVE < ea > , < ea >**Attributes:** Size = (Byte, Word, Long)**Description:** Moves the data at the source to the destination location and sets the condition codes according to the data. The size of the operation may be specified as byte, word, or long. Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | * | * | 0 | 0 |

X — Not affected.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Always cleared.

C — Always cleared.

**Instruction Format:**

|    |    |    |    |          |    |             |   |   |   |        |   |   |   |          |   |
|----|----|----|----|----------|----|-------------|---|---|---|--------|---|---|---|----------|---|
| 15 | 14 | 13 | 12 | 11       | 10 | 9           | 8 | 7 | 6 | 5      | 4 | 3 | 2 | 1        | 0 |
| 0  |    | 0  |    | SIZE     |    | DESTINATION |   |   |   | SOURCE |   |   |   |          |   |
|    |    |    |    | REGISTER |    | MODE        |   |   |   | MODE   |   |   |   | REGISTER |   |

**Instruction Fields:**

Size field—Specifies the size of the operand to be moved.

01 — Byte operation

11 — Word operation

10 — Long operation

**MOVE****Move Data from Source to Destination  
(M68000 Family)****MOVE**

Destination Effective Address field—Specifies the destination location. Only data alterable addressing modes can be used as listed in the following tables:

| Addressing Mode         | Mode | Register       |
|-------------------------|------|----------------|
| Dn                      | 000  | reg. number:Dn |
| An                      | —    | —              |
| (An)                    | 010  | reg. number:An |
| (An) +                  | 011  | reg. number:An |
| – (An)                  | 100  | reg. number:An |
| (d <sub>16</sub> ,An)   | 101  | reg. number:An |
| (d <sub>8</sub> ,An,Xn) | 110  | reg. number:An |

| Addressing Mode         | Mode | Register |
|-------------------------|------|----------|
| (xxx).W                 | 111  | 000      |
| (xxx).L                 | 111  | 001      |
| #<data>                 | —    | —        |
|                         |      |          |
|                         |      |          |
| (d <sub>16</sub> ,PC)   | —    | —        |
| (d <sub>8</sub> ,PC,Xn) | —    | —        |

**MC68020, MC68030, and MC68040 only**

|                 |     |                |
|-----------------|-----|----------------|
| (bd,An,Xn)*     | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

|                 |   |   |
|-----------------|---|---|
| (bd,PC,Xn)*     | — | — |
| ([bd,PC,Xn],od) | — | — |
| ([bd,PC],Xn,od) | — | — |

\*Can be used with CPU32.

## Integer Instructions

**MOVE**

**Move Data from Source to Destination**  
(M68000 Family)

**MOVE**

Source Effective Address field—Specifies the source operand. All addressing modes can be used as listed in the following tables:

| Addressing Mode         | Mode | Register       |
|-------------------------|------|----------------|
| Dn                      | 000  | reg. number:Dn |
| An                      | 001  | reg. number:An |
| (An)                    | 010  | reg. number:An |
| (An) +                  | 011  | reg. number:An |
| – (An)                  | 100  | reg. number:An |
| (d <sub>16</sub> ,An)   | 101  | reg. number:An |
| (d <sub>8</sub> ,An,Xn) | 110  | reg. number:An |

| Addressing Mode         | Mode | Register |
|-------------------------|------|----------|
| (xxx).W                 | 111  | 000      |
| (xxx).L                 | 111  | 001      |
| #<data>                 | 111  | 100      |
|                         |      |          |
|                         |      |          |
| (d <sub>16</sub> ,PC)   | 111  | 010      |
| (d <sub>8</sub> ,PC,Xn) | 111  | 011      |

**MC68020, MC68030, and MC68040 only**

|                 |     |                |
|-----------------|-----|----------------|
| (bd,An,Xn)**    | 110 | reg. number:An |
| ([bd,An,Xn],od) | 110 | reg. number:An |
| ([bd,An],Xn,od) | 110 | reg. number:An |

|                 |     |     |
|-----------------|-----|-----|
| (bd,PC,Xn)**    | 111 | 011 |
| ([bd,PC,Xn],od) | 111 | 011 |
| ([bd,PC],Xn,od) | 111 | 011 |

\*For byte size operation, address register direct is not allowed.

\*\*Can be used with CPU32.

**NOTE**

Most assemblers use MOVEA when the destination is an address register.

MOVEQ can be used to move an immediate 8-bit value to a data register.

## 2.4 BRIEF EXTENSION WORD FORMAT COMPATIBILITY

Programs can be easily transported from one member of the M68000 family to another in an upward-compatible fashion. The user object code of each early member of the family, which is upward compatible with newer members, can be executed on the newer microprocessor without change. Brief extension word formats are encoded with information that allows the CPU32, MC68020, MC68030, and MC68040 to distinguish the basic M68000 family architecture's new address extensions. Figure 2-3 illustrates these brief extension word formats. The encoding for SCALE used by the CPU32, MC68020, MC68030, and MC68040 is a compatible extension of the M68000 family architecture. A value of zero for SCALE is the same encoding for both extension words. Software that uses this encoding is compatible with all processors in the M68000 family. Both brief extension word formats do not contain the other values of SCALE. Software can be easily migrated in an upward-compatible direction, with downward support only for nonscaled addressing. If the MC68000 were to execute an instruction that encoded a scaling factor, the scaling factor would be ignored and would not access the desired memory address. The earlier microprocessors do not recognize the brief extension word formats implemented by newer processors. Although they can detect illegal instructions, they do not decode invalid encodings of the brief extension word formats as exceptions.

|     |          |    |    |    |     |   |   |   |                      |   |   |   |   |   |   |
|-----|----------|----|----|----|-----|---|---|---|----------------------|---|---|---|---|---|---|
| 15  | 14       | 13 | 12 | 11 | 10  | 9 | 8 | 7 | 6                    | 5 | 4 | 3 | 2 | 1 | 0 |
| D/A | REGISTER |    |    |    | W/L | 0 | 0 | 0 | DISPLACEMENT INTEGER |   |   |   |   |   |   |

(a) MC68000, MC68008, and MC68010

|     |          |    |    |    |     |       |   |   |                      |   |   |   |   |   |   |
|-----|----------|----|----|----|-----|-------|---|---|----------------------|---|---|---|---|---|---|
| 15  | 14       | 13 | 12 | 11 | 10  | 9     | 8 | 7 | 6                    | 5 | 4 | 3 | 2 | 1 | 0 |
| D/A | REGISTER |    |    |    | W/L | SCALE |   | 0 | DISPLACEMENT INTEGER |   |   |   |   |   |   |

(b) CPU32, MC68020, MC68030, and MC68040

Figure 2-3. M68000 Family Brief Extension Word Formats

Table 2-1. Instruction Word Format Field Definitions

| Field              | Definition                                                                                                            |
|--------------------|-----------------------------------------------------------------------------------------------------------------------|
| <b>Instruction</b> |                                                                                                                       |
| Mode               | Addressing Mode                                                                                                       |
| Register           | General Register Number                                                                                               |
| <b>Extensions</b>  |                                                                                                                       |
| D/A                | Index Register Type<br>0 = Dn<br>1 = An                                                                               |
| W/L                | Word/Long-Word Index Size<br>0 = Sign-Extended Word<br>1 = Long Word                                                  |
| Scale              | Scale Factor<br>00 = 1<br>01 = 2<br>10 = 4<br>11 = 8                                                                  |
| BS                 | Base Register Suppress<br>0 = Base Register Added<br>1 = Base Register Suppressed                                     |
| IS                 | Index Suppress<br>0 = Evaluate and Add Index Operand<br>1 = Suppress Index Operand                                    |
| BD SIZE            | Base Displacement Size<br>00 = Reserved<br>01 = Null Displacement<br>10 = Word Displacement<br>11 = Long Displacement |
| I/IS               | Index/Indirect Selection<br>Indirect and Indexing Operand Determined in Conjunction with Bit 6, Index Suppress        |

For effective addresses that use a full extension word format, the index suppress (IS) bit and the index/indirect selection (I/IS) field determine the type of indexing and indirect action. Table 2-2 lists the index and indirect operations corresponding to all combinations of IS and I/IS values.





# PIC12F508/509/16F505

## 8/14-Pin, 8-Bit Flash Microcontrollers

### Devices Included In This Data Sheet:

- PIC12F508 • PIC12F509 • PIC16F505

### High-Performance RISC CPU:

- Only 33 Single-Word Instructions to Learn
- All Single-Cycle Instructions Except for Program Branches, which are Two-Cycle
- 12-Bit Wide Instructions
- 2-Level Deep Hardware Stack
- Direct, Indirect and Relative Addressing modes for Data and Instructions
- 8-Bit Wide Data Path
- 8 Special Function Hardware Registers
- Operating Speed:
  - DC – 20 MHz clock input (PIC16F505 only)
  - DC – 200 ns instruction cycle (PIC16F505 only)
  - DC – 4 MHz clock input
  - DC – 1000 ns instruction cycle

### Special Microcontroller Features:

- 4 MHz Precision Internal Oscillator:
  - Factory calibrated to  $\pm 1\%$
- In-Circuit Serial Programming™ (ICSP™)
- In-Circuit Debugging (ICD) Support
- Power-On Reset (POR)
- Device Reset Timer (DRT)
- Watchdog Timer (WDT) with Dedicated On-Chip RC Oscillator for Reliable Operation
- Programmable Code Protection
- Multiplexed MCLR Input Pin
- Internal Weak Pull-Ups on I/O Pins
- Power-Saving Sleep mode
- Wake-Up from Sleep on Pin Change
- Selectable Oscillator Options:
  - INTRC: 4 MHz precision Internal oscillator
  - EXTRC: External low-cost RC oscillator
  - XT: Standard crystal/resonator
  - HS: High-speed crystal/resonator (PIC16F505 only)
  - LP: Power-saving, low-frequency crystal
  - EC: High-speed external clock input (PIC16F505 only)

### Low-Power Features/CMOS Technology:

- Operating Current:
  - < 175  $\mu\text{A}$  @ 2V, 4 MHz, typical
- Standby Current:
  - 100 nA @ 2V, typical
- Low-Power, High-Speed Flash Technology:
  - 100,000 Flash endurance
  - > 40 year retention
- Fully Static Design
- Wide Operating Voltage Range: 2.0V to 5.5V
- Wide Temperature Range:
  - Industrial: -40°C to +85°C
  - Extended: -40°C to +125°C

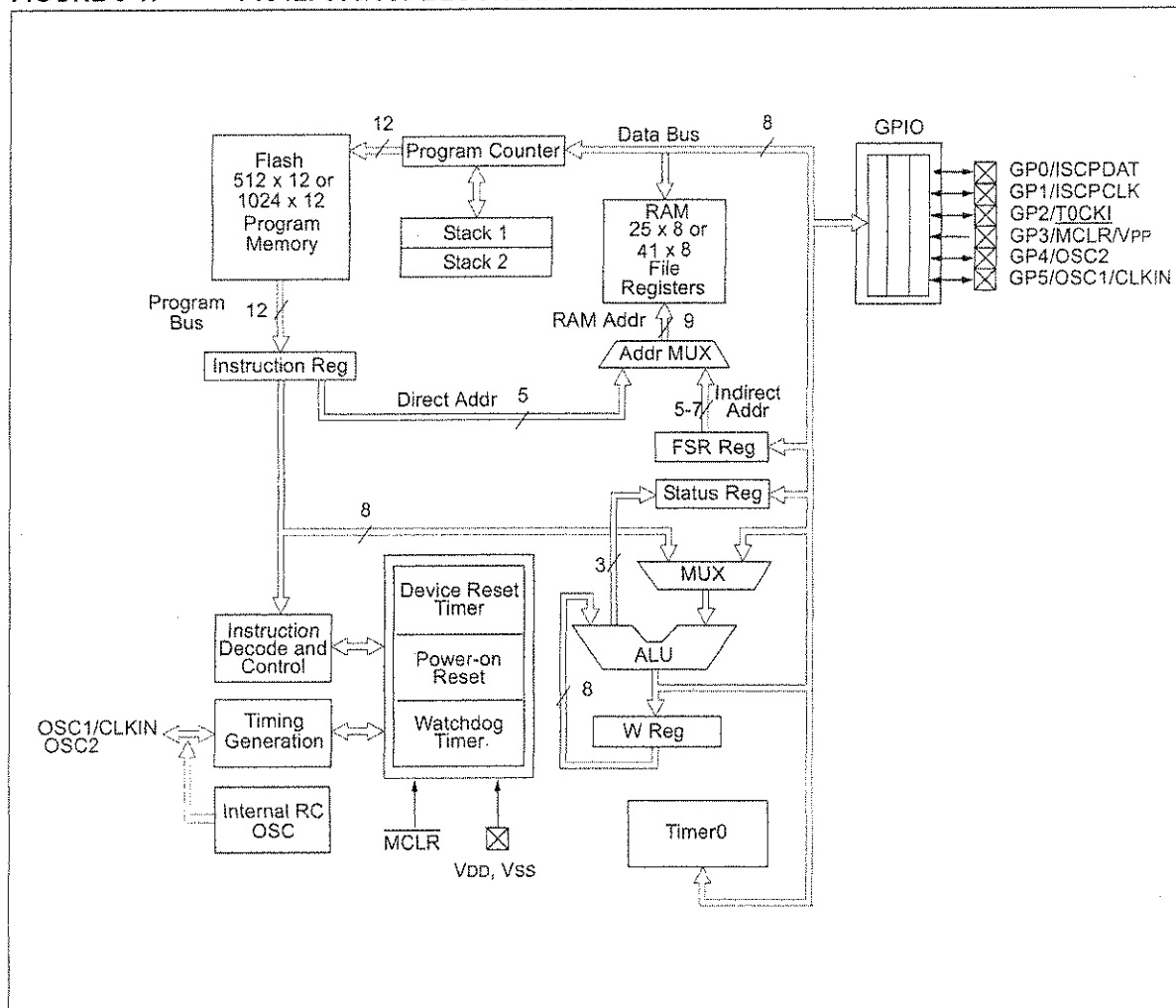
### Peripheral Features (PIC12F508/509):

- 6 I/O Pins:
  - 5 I/O pins with individual direction control
  - 1 input only pin
  - High current sink/source for direct LED drive
  - Wake-on-change
  - Weak pull-ups
- 8-Bit Real-Time Clock/Counter (TMR0) with 8-Bit Programmable Prescaler

### Peripheral Features (PIC16F505):

- 12 I/O Pins:
  - 11 I/O pins with individual direction control
  - 1 input only pin
  - High current sink/source for direct LED drive
  - Wake-on-change
  - Weak pull-ups
- 8-Bit Real-Time Clock/Counter (TMR0) with 8-Bit Programmable Prescaler

FIGURE 3-1: PIC12F508/509 BLOCK DIAGRAM



| Device    | Program Memory | Data Memory  | I/O | Timers<br>8-bit |
|-----------|----------------|--------------|-----|-----------------|
|           | Flash (words)  | SRAM (bytes) |     |                 |
| PIC12F508 | 512            | 25           | 6   | 1               |
| PIC12F509 | 1024           | 41           | 6   | 1               |
| PIC16F505 | 1024           | 72           | 12  | 1               |

## 68000 Quick Reference

| Opcode            | Size            | Operand              | CCR   | Effective Address |    |      |       |       |        |           |       |       |        |           |    |                                                                                                                                             | Operation                                                                                | Description |
|-------------------|-----------------|----------------------|-------|-------------------|----|------|-------|-------|--------|-----------|-------|-------|--------|-----------|----|---------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|-------------|
|                   | BWL             | sr,ds                | XNZVC | Dn                | An | (An) | (An)+ | -(An) | (d,An) | (d,An,Rn) | abs,W | abs,L | (d,PC) | (d,PC,Rn) | #n |                                                                                                                                             |                                                                                          |             |
| ABCD              | B               | Dy,Dx<br>-(Ay),-(Ax) | *U*U* | -                 | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -  | $Dy_{10} + Dx_{10} + X \rightarrow Dx_{10}$<br>$-(Ay)_{10} + -(Ax)_{10} + X \rightarrow -(Ax)_{10}$                                         | Add BCD with Extend                                                                      |             |
| ADD               | BWL             | sr,Dn<br>Dn,ds       | ***** | sr                | sr | sr   | sr    | sr    | sr     | sr        | sr    | sr    | sr     | sr        | sr | $sr + Dn \rightarrow Dn$<br>$Dn + ds \rightarrow ds$                                                                                        | Add binary<br>(ADDI or ADDQ is used when source is #n)                                   |             |
| ADDA <sup>2</sup> | WL              | sr,An                | ----- | sr                | sr | sr   | sr    | sr    | sr     | sr        | sr    | sr    | sr     | sr        | sr | $sr + An \rightarrow An$                                                                                                                    | Add address (W sign-extended to L)                                                       |             |
| ADDI <sup>2</sup> | BWL             | #n,ds                | ***** | ds                | -  | ds   | ds    | ds    | ds     | ds        | ds    | ds    | -      | -         | -  | $\#n + ds \rightarrow ds$                                                                                                                   | Add immediate                                                                            |             |
| ADDQ <sup>2</sup> | BWL             | #n,ds                | ***** | ds                | ds | ds   | ds    | ds    | ds     | ds        | ds    | ds    | -      | -         | -  | $\#n + ds \rightarrow ds$                                                                                                                   | Add quick immediate (#n range: 1 to 8)                                                   |             |
| ADDX              | BWL             | Dy,Dx<br>-(Ay),-(Ax) | ***** | -                 | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -  | $Dy + Dx + X \rightarrow Dx$<br>$-(Ay) + -(Ax) + X \rightarrow -(Ax)$                                                                       | Add with Extend                                                                          |             |
| AND               | BWL             | sr,Dn<br>Dn,ds       | -**00 | sr                | -  | sr   | sr    | sr    | sr     | sr        | sr    | sr    | sr     | sr        | sr | $sr \& Dn \rightarrow Dn$<br>$Dn \& ds \rightarrow ds$                                                                                      | Logical AND<br>(ANDI is used when source is #n)                                          |             |
| ANDI <sup>2</sup> | BWL             | #n,ds                | -**00 | ds                | -  | ds   | ds    | ds    | ds     | ds        | ds    | ds    | -      | -         | -  | $\#n \& ds \rightarrow ds$                                                                                                                  | Logical AND immediate                                                                    |             |
| ANDI <sup>2</sup> | B               | #n,CCR               | ***** | -                 | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -  | $\#n \& CCR \rightarrow CCR$                                                                                                                | Logical AND immediate to CCR                                                             |             |
| ANDI <sup>2</sup> | W               | #n,SR                | ***** | -                 | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -  | $\#n \& SR \rightarrow SR$                                                                                                                  | Logical AND immediate to SR (Privileged)                                                 |             |
| ASL               | BWL             | Dx,Dy                | ***** | -                 | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -  |                                                                                                                                             | Arithmetic shift Dy Dx bits left/right                                                   |             |
| ASR               | BWL             | #n,Dy                | ***** | -                 | -  | ds   | ds    | ds    | ds     | ds        | ds    | ds    | -      | -         | -  |                                                                                                                                             | Arithmetic shift Dy #n bits L/R (#n: 1 to 8)                                             |             |
|                   | W               | ds                   | ***** | -                 | -  | ds   | ds    | ds    | ds     | ds        | ds    | ds    | -      | -         | -  |                                                                                                                                             | Arithmetic shift ds 1 bit left/right (W only)                                            |             |
| Bcc               | BW <sup>d</sup> | label                | ----- | -                 | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -  | if cc true then<br>$PC + d \rightarrow PC$                                                                                                  | Branch conditionally (cc: See Table next pg)<br>(d: 8/16-bit signed integer)             |             |
| BCHG              | B L             | Dn,ds<br>#n,ds       | ----- | ds                | -  | ds   | ds    | ds    | ds     | ds        | ds    | ds    | -      | -         | -  | $\text{NOT}(\text{bit number of } ds) \rightarrow Z$<br>$\text{NOT}(\text{bit } n \text{ of } ds) \rightarrow \text{bit } n \text{ of } ds$ | Set Z with state of specified bit in ds then<br>invert the bit in ds                     |             |
| BCLR              | B L             | Dn,ds<br>#n,ds       | ----- | ds                | -  | ds   | ds    | ds    | ds     | ds        | ds    | ds    | -      | -         | -  | $\text{NOT}(\text{bit number of } ds) \rightarrow Z$<br>$0 \rightarrow \text{bit number of } ds$                                            | Set Z with state of specified bit in ds then<br>clear the bit in ds                      |             |
| BRA               | BW <sup>d</sup> | label                | ----- | -                 | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -  | $PC + d \rightarrow PC$                                                                                                                     | Branch always (d: 8/16-bit signed integer)                                               |             |
| BSET              | B L             | Dn,ds<br>#n,ds       | ----- | ds                | -  | ds   | ds    | ds    | ds     | ds        | ds    | ds    | -      | -         | -  | $\text{NOT}(\text{bit } n \text{ of } ds) \rightarrow Z$<br>$1 \rightarrow \text{bit } n \text{ of } ds$                                    | Set Z with state of specified bit in ds then<br>set the bit in ds                        |             |
| BSR               | BW <sup>d</sup> | label                | ----- | -                 | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -  | $PC \rightarrow -(SP); PC + d \rightarrow PC$                                                                                               | Branch to subroutine (d: 8/16-bit sign-int)                                              |             |
| BTSY              | B L             | Dn,ds<br>#n,ds       | ----- | ds                | -  | ds   | ds    | ds    | ds     | ds        | ds    | ds    | ds     | ds        | ds | $\text{NOT}(\text{bit } Dn \text{ of } ds) \rightarrow Z$<br>$\text{NOT}(\text{bit } \#n \text{ of } ds) \rightarrow Z$                     | Set Z with state of specified bit in ds<br>Leave the bit in ds unchanged                 |             |
| CHK               | W               | sr,Dn                | -*UUU | sr                | -  | sr   | sr    | sr    | sr     | sr        | sr    | sr    | sr     | sr        | sr | if $Dn < 0$ or $Dn > sr$ then TRAP                                                                                                          | Compare Dn with 0 and upper bound [sr]                                                   |             |
| CLR               | BWL             | ds                   | -0100 | ds                | -  | ds   | ds    | ds    | ds     | ds        | ds    | ds    | -      | -         | -  | $0 \rightarrow ds$                                                                                                                          | Clear destination to zero                                                                |             |
| CMP               | BWL             | sr,Dn                | ***** | sr                | sr | sr   | sr    | sr    | sr     | sr        | sr    | sr    | sr     | sr        | sr | set CCR with $Dn - sr$                                                                                                                      | Compare Dn to source                                                                     |             |
| CMPI <sup>2</sup> | WL              | sr,An                | ***** | sr                | sr | sr   | sr    | sr    | sr     | sr        | sr    | sr    | sr     | sr        | sr | set CCR with $An - sr$                                                                                                                      | Compare An to source                                                                     |             |
| CMPI <sup>2</sup> | BWL             | #n,ds                | ***** | ds                | -  | ds   | ds    | ds    | ds     | ds        | ds    | ds    | -      | -         | -  | set CCR with $ds - \#n$                                                                                                                     | Compare destination to #n                                                                |             |
| CMPW <sup>2</sup> | BWL             | (Ay)+,(Ax)+          | ***** | -                 | -  | -    | ee    | -     | -      | -         | -     | -     | -      | -         | -  | set CCR with $(Ax) - (Ay)$                                                                                                                  | Compare (Ax) to (Ay); Increment Ax & Ay                                                  |             |
| DBcc              | W               | Dn,label             | ----- | -                 | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -  | if cc false then { $Dn-1 \rightarrow Dn$<br>if $Dn < -1$ then $PC+d \rightarrow PC$ }                                                       | Test condition, decrement & branch<br>(d: 16-bit signed integer)                         |             |
| DIVS              | W               | sr,Dn                | -***0 | sr                | -  | sr   | sr    | sr    | sr     | sr        | sr    | sr    | sr     | sr        | sr | $\pm 32\text{bit } Dn / \pm 16\text{bit } sr \rightarrow \pm Dn$                                                                            | $Dn = [16\text{-bit remainder}, 16\text{-bit quotient}]$                                 |             |
| DIVU              | W               | sr,Dn                | -***0 | sr                | -  | sr   | sr    | sr    | sr     | sr        | sr    | sr    | sr     | sr        | sr | $32\text{bit } Dn / 16\text{bit } sr \rightarrow Dn$                                                                                        | $Dn = [16\text{-bit remainder}, 16\text{-bit quotient}]$                                 |             |
| EOR               | BWL             | Dn,ds                | -*000 | ds                | -  | ds   | ds    | ds    | ds     | ds        | ds    | ds    | -      | -         | -  | $Dn \text{ XOR } ds \rightarrow ds$                                                                                                         | Logical exclusive OR Dn to ds                                                            |             |
| EORI <sup>2</sup> | BWL             | #n,ds                | -*000 | ds                | -  | ds   | ds    | ds    | ds     | ds        | ds    | ds    | -      | -         | -  | $\#n \text{ XOR } ds \rightarrow ds$                                                                                                        | Logical exclusive OR #n to ds                                                            |             |
| EORI <sup>2</sup> | B               | #n,CCR               | ***** | -                 | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -  | $\#n \text{ XOR } CCR \rightarrow CCR$                                                                                                      | Logical exclusive OR #n to CCR                                                           |             |
| EORI <sup>2</sup> | W               | #n,SR                | ***** | -                 | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -  | $\#n \text{ XOR } SR \rightarrow SR$                                                                                                        | Logical exclusive OR #n to SR (Privileged)                                               |             |
| EXG               | L               | Rx,Ry                | ----- | ee                | ee | -    | -     | -     | -      | -         | -     | -     | -      | -         | -  | register $\leftrightarrow$ register                                                                                                         | Exchange registers (32-bit only)                                                         |             |
| EXT               | WL              | Dn                   | -**00 | -                 | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -  | $Dn.B \rightarrow Dn.W \mid Dn.W \rightarrow Dn.L$                                                                                          | Sign extend (change B to W or W to L)                                                    |             |
| ILLEGAL           |                 |                      | ----- | -                 | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -  | $PC \rightarrow -(SSP); SR \rightarrow -(SSP)$                                                                                              | Generate illegal instruction exception                                                   |             |
| JMP               |                 | ds                   | ----- | -                 | -  | ds   | -     | -     | ds     | ds        | ds    | ds    | ds     | ds        | ds | $ds \rightarrow PC$                                                                                                                         | Jump to address specified by ds                                                          |             |
| JSR               |                 | ds                   | ----- | -                 | -  | ds   | -     | -     | ds     | ds        | ds    | ds    | ds     | ds        | ds | $PC \rightarrow -(SP); ds \rightarrow PC$                                                                                                   | push PC, jump to subroutine at address ds                                                |             |
| LEA               | L               | sr,An                | ----- | -                 | -  | sr   | -     | -     | sr     | sr        | sr    | sr    | sr     | sr        | sr | $sr \rightarrow An$                                                                                                                         | Load effective address of sr to An                                                       |             |
| LINK              |                 | An,#n                | ----- | -                 | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -  | $An \rightarrow -(SP); SP \rightarrow An$<br>$SP + \#n \rightarrow SP$                                                                      | Create local workspace on stack<br>(n must be negative to allocate)                      |             |
| LSL               | BWL             | Dx,Dy                | ***0* | -                 | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -  |                                                                                                                                             | Logical shift Dy, Dx bits left/right                                                     |             |
| LSR               | BWL             | #n,Dy                | ***0* | -                 | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -  |                                                                                                                                             | Logical shift Dy, #n bits L/R (#n: 1 to 8)                                               |             |
|                   | W               | ds                   | ***0* | -                 | -  | ds   | ds    | ds    | ds     | ds        | ds    | ds    | -      | -         | -  |                                                                                                                                             | Logical shift ds 1 bit left/right (W only)                                               |             |
| MOVE              | BWL             | ea,ea                | -**00 | ea                | sr | ea   | ea    | ea    | ea     | ea        | ea    | ea    | sr     | sr        | sr | $sr \rightarrow ds$                                                                                                                         | Move data from source to destination                                                     |             |
| MOVE              | W               | sr,CCR               | ***** | sr                | -  | sr   | sr    | sr    | sr     | sr        | sr    | sr    | sr     | sr        | sr | $sr \rightarrow CCR$                                                                                                                        | Move source to Condition Code Register                                                   |             |
| MOVE              | W               | sr,SR                | ***** | sr                | -  | sr   | sr    | sr    | sr     | sr        | sr    | sr    | sr     | sr        | sr | $sr \rightarrow SR$                                                                                                                         | Move source to Status Register (Privileged)                                              |             |
| MOVE              | W               | SR,ds                | ----- | ds                | -  | ds   | ds    | ds    | ds     | ds        | ds    | ds    | -      | -         | -  | $SR \rightarrow ds$                                                                                                                         | Move Status Register to destination                                                      |             |
| MOVE              | L               | USP,An<br>An,USP     | ----- | -                 | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -  | $USP \rightarrow An$<br>$An \rightarrow USP$                                                                                                | Move User Stack Pointer to An (Privileged)<br>Move An to User Stack Pointer (Privileged) |             |

| Opcode             | Size | Operand              | CCR    | Effective Address |    |      |       |       |       |          |       |       |        |           |    | Operation                                                                                                                         | Description                                                                                                                    |
|--------------------|------|----------------------|--------|-------------------|----|------|-------|-------|-------|----------|-------|-------|--------|-----------|----|-----------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
|                    |      |                      |        | Dn                | An | (An) | (An)+ | -(An) | (dAn) | (dAn,Rn) | abs.W | abs.L | (d,PC) | (d,PC,Rn) | #n |                                                                                                                                   |                                                                                                                                |
| MOVEA <sup>2</sup> | WL   | sr,An                | -----  | sr                | sr | sr   | sr    | sr    | sr    | sr       | sr    | sr    | sr     | sr        | sr | sr → An                                                                                                                           | Move source to An (MOVE sr,An use MOVEA)                                                                                       |
| MOVEN <sup>2</sup> | WL   | Rn-Rn,ds<br>sr,Rn-Rn | -----  | -                 | -  | ds   | -     | ds    | ds    | ds       | ds    | ds    | -      | -         | -  | Registers → ds<br>sr → Registers                                                                                                  | Move specified registers to/from memory<br>(W source is sign-extended to L for Rn)                                             |
| MOVEP              | WL   | Dn,d(An)<br>d(An),Dn | -----  | -                 | -  | -    | -     | -     | -     | -        | -     | -     | -      | -         | -  | Dn → d(An)...d+2(An)...d+4(A)<br>d(An) → Dn; d+2(An)...d+4(A)                                                                     | Move Dn to/from alternate memory bytes<br>(Access only even or odd addresses)                                                  |
| MOVEQ <sup>2</sup> | L    | #n,Dn                | ---*00 | -                 | -  | -    | -     | -     | -     | -        | -     | -     | -      | -         | -  | #n → Dn                                                                                                                           | Move sign extended 8-bit #n to Dn                                                                                              |
| MULS               | W    | sr,Dn                | ---*00 | sr                | -  | sr   | sr    | sr    | sr    | sr       | sr    | sr    | sr     | sr        | sr | ±16bit sr * ±16bit Dn → ±Dn                                                                                                       | Multiply signed 16-bit; result: signed 32-bit                                                                                  |
| MULU               | W    | sr,Dn                | ---*00 | sr                | -  | sr   | sr    | sr    | sr    | sr       | sr    | sr    | sr     | sr        | sr | 16bit sr * 16bit Dn → Dn                                                                                                          | Multiply unsg'd 16-bit; result: unsg'd 32-bit                                                                                  |
| NBCD               | B    | ds                   | *U*U*  | ds                | -  | ds   | ds    | ds    | ds    | ds       | ds    | ds    | -      | -         | -  | 0 - ds <sub>10</sub> - X → ds                                                                                                     | Negate BCD with Extend                                                                                                         |
| NEG                | BWL  | ds                   | *****  | ds                | -  | ds   | ds    | ds    | ds    | ds       | ds    | ds    | -      | -         | -  | 0 - ds → ds                                                                                                                       | Negate ds                                                                                                                      |
| NEGX               | BWL  | ds                   | *****  | ds                | -  | ds   | ds    | ds    | ds    | ds       | ds    | ds    | -      | -         | -  | 0 - ds - X → ds                                                                                                                   | Negate ds with Extend                                                                                                          |
| NOP                |      |                      | -----  | -                 | -  | -    | -     | -     | -     | -        | -     | -     | -      | -         | -  | None                                                                                                                              | No operation occurs                                                                                                            |
| NOT                | BWL  | ds                   | ---*00 | ds                | -  | ds   | ds    | ds    | ds    | ds       | ds    | ds    | -      | -         | -  | NOT( ds ) → ds                                                                                                                    | Logical NOT (ones complement of ds)                                                                                            |
| OR                 | BWL  | sr,Dn<br>Dn,ds       | ---*00 | sr                | -  | sr   | sr    | sr    | sr    | sr       | sr    | sr    | sr     | sr        | sr | sr OR Dn → Dn<br>Dn OR ds → ds                                                                                                    | Logical OR<br>(OR) is used when source is #n)                                                                                  |
| ORI <sup>2</sup>   | BWL  | #n,ds                | ---*00 | ds                | -  | ds   | ds    | ds    | ds    | ds       | ds    | ds    | -      | -         | -  | #n OR ds → ds                                                                                                                     | Logical OR #n to ds                                                                                                            |
| ORI <sup>2</sup>   | B    | #n,CCR               | *****  | -                 | -  | -    | -     | -     | -     | -        | -     | -     | -      | -         | -  | #n OR CCR → CCR                                                                                                                   | Logical OR #n to CCR                                                                                                           |
| ORI <sup>2</sup>   | W    | #n,SR                | *****  | -                 | -  | -    | -     | -     | -     | -        | -     | -     | -      | -         | -  | #n OR SR → SR                                                                                                                     | Logical OR #n to SR (Privileged)                                                                                               |
| PEA                | L    | ds                   | -----  | -                 | -  | ds   | -     | -     | ds    | ds       | ds    | ds    | ds     | ds        | -  | ds → -(SP)                                                                                                                        | Push effective address of ds onto stack                                                                                        |
| RESET              |      |                      | -----  | -                 | -  | -    | -     | -     | -     | -        | -     | -     | -      | -         | -  | Assert RESET Line                                                                                                                 | Issue a hardware RESET (Privileged)                                                                                            |
| ROL                | BWL  | Dx,Dy                | ---*0* | -                 | -  | -    | -     | -     | -     | -        | -     | -     | -      | -         | -  |                                                 | Rotate Dy, Dx bits left/right (without X)<br>Rotate Dy, #n bits left/right (#n: 1 to 8)<br>Rotate ds 1-bit left/right (W only) |
| ROR                | W    | ds                   |        | -                 | -  | ds   | ds    | ds    | ds    | ds       | ds    | ds    | -      | -         | -  |                                                                                                                                   |                                                                                                                                |
| ROXL               | BWL  | Dx,Dy                | ***0*  | -                 | -  | -    | -     | -     | -     | -        | -     | -     | -      | -         | -  |                                                | Rotate Dy, Dx bits L/R (X used then updated) Rotate Dy, #n bits left/right (#n: 1 to 8)<br>Rotate ds 1-bit left/right (W only) |
| ROXR               | W    | ds                   |        | -                 | -  | ds   | ds    | ds    | ds    | ds       | ds    | ds    | -      | -         | -  |                                                                                                                                   |                                                                                                                                |
| RTE                |      |                      | *****  | -                 | -  | -    | -     | -     | -     | -        | -     | -     | -      | -         | -  | (SP)+ → SR; (SP)+ → PC                                                                                                            | Return from exception (Privileged)                                                                                             |
| RTR                |      |                      | *****  | -                 | -  | -    | -     | -     | -     | -        | -     | -     | -      | -         | -  | (SP)+ → CCR; (SP)+ → PC                                                                                                           | Return from subroutine and restore CCR                                                                                         |
| RTS                |      |                      | -----  | -                 | -  | -    | -     | -     | -     | -        | -     | -     | -      | -         | -  | (SP)+ → PC                                                                                                                        | Return from subroutine                                                                                                         |
| SBCD               | B    | Dy,Dx<br>-(Ay),-(Ax) | *U*U*  | -                 | -  | -    | -     | -     | -     | -        | -     | -     | -      | -         | -  | Dx <sub>10</sub> - Dy <sub>10</sub> - X → Dx <sub>10</sub><br>-(Ax) <sub>10</sub> - -(Ay) <sub>10</sub> - X → -(Ax) <sub>10</sub> | Subtract BCD with Extend                                                                                                       |
| SCC                | B    | ds                   | -----  | ds                | -  | ds   | ds    | ds    | ds    | ds       | ds    | ds    | -      | -         | -  | If cc is true then 1's → ds<br>else 0's → ds                                                                                      | If cc true then ds.B = 11111111<br>else ds.B = 00000000                                                                        |
| STOP               |      | #n                   | *****  | -                 | -  | -    | -     | -     | -     | -        | -     | -     | -      | -         | -  | #n → SR; STOP                                                                                                                     | Move #n to SR, stop processor (Privileged)                                                                                     |
| SUB                | BWL  | sr,Dn<br>Dn,ds       | *****  | sr                | sr | sr   | sr    | sr    | sr    | sr       | sr    | sr    | sr     | sr        | sr | Dn - sr → Dn<br>ds - Dn → ds                                                                                                      | Subtract binary<br>(SUBI or SUBD is used when source is #n)                                                                    |
| SUBA <sup>2</sup>  | WL   | sr,An                | -----  | sr                | sr | sr   | sr    | sr    | sr    | sr       | sr    | sr    | sr     | sr        | sr | An - sr → An                                                                                                                      | Subtract address (W sign-extended to L)                                                                                        |
| SUBI <sup>2</sup>  | BWL  | #n,ds                | *****  | ds                | -  | ds   | ds    | ds    | ds    | ds       | ds    | ds    | -      | -         | -  | ds - #n → ds                                                                                                                      | Subtract immediate                                                                                                             |
| SUBQ <sup>2</sup>  | BWL  | #n,ds                | *****  | ds                | ds | ds   | ds    | ds    | ds    | ds       | ds    | ds    | -      | -         | -  | ds - #n → ds                                                                                                                      | Subtract quick immediate (#n range: 1 to 8)                                                                                    |
| SUBX               | BWL  | Dy,Dx<br>-(Ay),-(Ax) | *****  | -                 | -  | -    | -     | -     | -     | -        | -     | -     | -      | -         | -  | Dx - Dy - X → Dx<br>-(Ax) - -(Ay) - X → -(Ax)                                                                                     | Subtract with Extend                                                                                                           |
| SWAP               | W    | Dn                   | ---*00 | -                 | -  | -    | -     | -     | -     | -        | -     | -     | -      | -         | -  | bits[31:16] ↔ bits[15:0]                                                                                                          | Exchange the 16-bit halves of Dn                                                                                               |
| TAS                | B    | ds                   | ---*00 | ds                | -  | ds   | ds    | ds    | ds    | ds       | ds    | ds    | -      | -         | -  | test ds → CCR; 1 → bit7 of ds                                                                                                     | N and Z set to reflect ds, bit7 of ds set to 1                                                                                 |
| TRAP               |      | #n                   | -----  | -                 | -  | -    | -     | -     | -     | -        | -     | -     | -      | -         | -  | PC → -(SSP); SR → -(SSP);<br>(vector table entry) → PC                                                                            | Push PC and SR, PC set by vector table #n<br>(#n range: 0 to 15)                                                               |
| TRAPV              |      |                      | -----  | -                 | -  | -    | -     | -     | -     | -        | -     | -     | -      | -         | -  | If V then TRAP #7                                                                                                                 | If overflow, execute an Overflow TRAP                                                                                          |
| TST                | BWL  | ds                   | ---*00 | ds                | -  | ds   | ds    | ds    | ds    | ds       | ds    | ds    | -      | -         | -  | test ds → CCR                                                                                                                     | N and Z set to reflect ds                                                                                                      |
| UNLK               |      | An                   | -----  | -                 | -  | -    | -     | -     | -     | -        | -     | -     | -      | -         | -  | An → SP; (SP)+ → An                                                                                                               | Remove local workspace from stack                                                                                              |

| Condition Tests ( & logical AND, + logical OR, ! logical NOT, * Unsigned ) |             |         |    |                  |                           |
|----------------------------------------------------------------------------|-------------|---------|----|------------------|---------------------------|
| cc                                                                         | Condition   | Test    | cc | Condition        | Test                      |
| T                                                                          | true        | 1       | VC | overflow clear   | !V                        |
| F                                                                          | false       | 0       | VS | overflow set     | V                         |
| HI*                                                                        | high        | IC & IZ | PL | plus             | !N                        |
| LS*                                                                        | low or same | C + Z   | MI | minus            | N                         |
| CC, HS*                                                                    | carry clear | !C      | GE | greater or equal | N & V + !N & !V           |
| CS, LO*                                                                    | carry set   | C       | LT | less than        | N & !V + !N & V           |
| NE                                                                         | not equal   | !Z      | GT | greater than     | N & V & !Z + !N & !V & !Z |
| EQ                                                                         | equal       | Z       | LE | less or equal    | Z + N & !V + !N & V       |

An Address register (16/32-bit, n=0-7)  
Dn Data register (8/16/32-bit, n=0-7)  
Rn any data or address register  
PC Program Counter (24-bit)  
sr Source ds Destination  
#n Immediate data d Displacement  
ea Effective Address (source or destination)  
BCD Binary Coded Decimal

SSP Supervisor Stack Pointer (32-bit)  
USP User Stack Pointer (32-bit)  
SP Active Stack Pointer (same as A7)  
label Destination of Branch (Assembler calculates displacement value)  
SR Status Register (16-bit)  
CCR Condition Code Register (lower 8-bits of SR)  
N negative, Z zero, V overflow, C carry, X extend  
\* set according to result of operation  
- not affected, 0 cleared, 1 set, U undefined

- Long only; all others are byte only
- Assembler selects appropriate opcode
- Branch sizes: .B or .S -128 to +127 bytes, .W or .L -32768 to +32767 bytes

# Electronique

## Partiel 2 – Mai 2011

*Les calculatrices et les documents ne sont pas autorisés. Le barème est donné à titre indicatif.  
Réponses exclusivement sur le sujet*

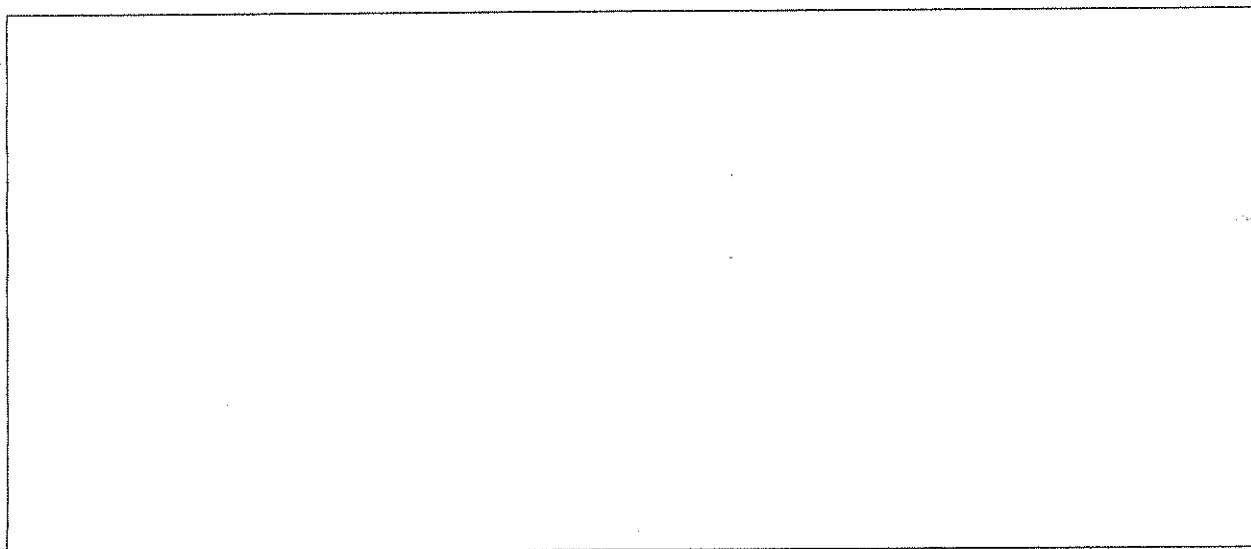
Durée 1h30

Nom : ..... Classe : .....

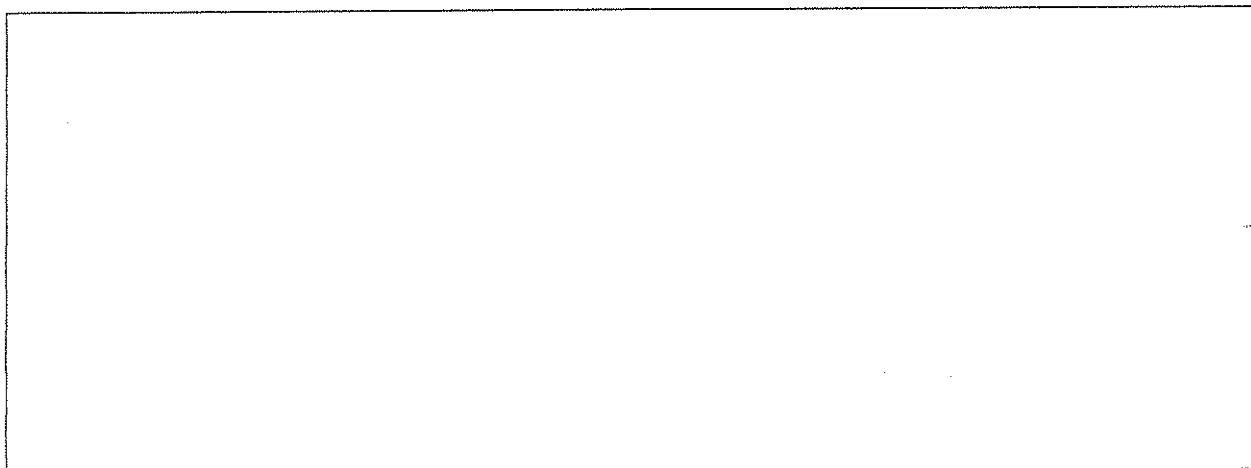
Prénom : .....

### Exercice 1. Questions de cours (3 points)

- A. Donnez le schéma équivalent petits signaux d'un JFET Canal N, en précisant bien l'emplacement de chacune des bornes.



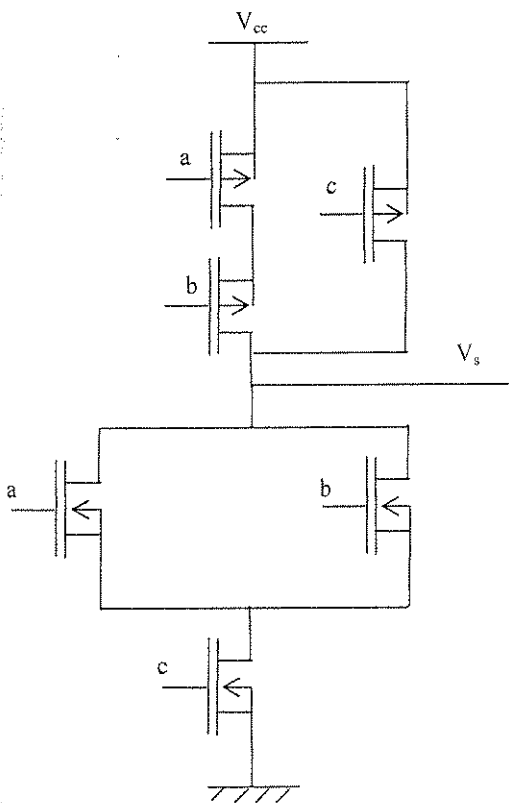
- B. Quel est le principe de fonctionnement du Convertisseur Flash ? De quel type de convertisseur s'agit-il ?



## Exercice 2. Portes logiques et électronique (4 points)

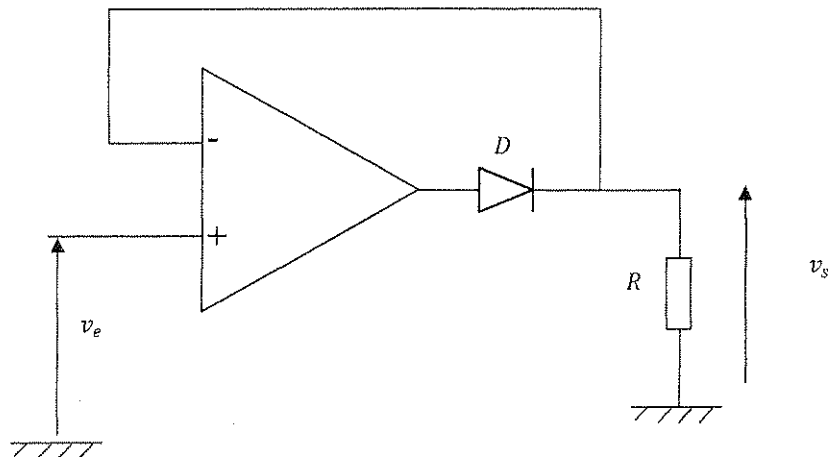
1. Quelles sont les différences entre les technologies TTL et CMOS ?

2. Soit le montage suivant : De quelle fonction logique s'agit-il ? Vous donnerez votre réponse sous la forme d'une équation en justifiant votre réponse.



### Exercice 3. Amplificateur opérationnel (4 points)

On considère le montage suivant :



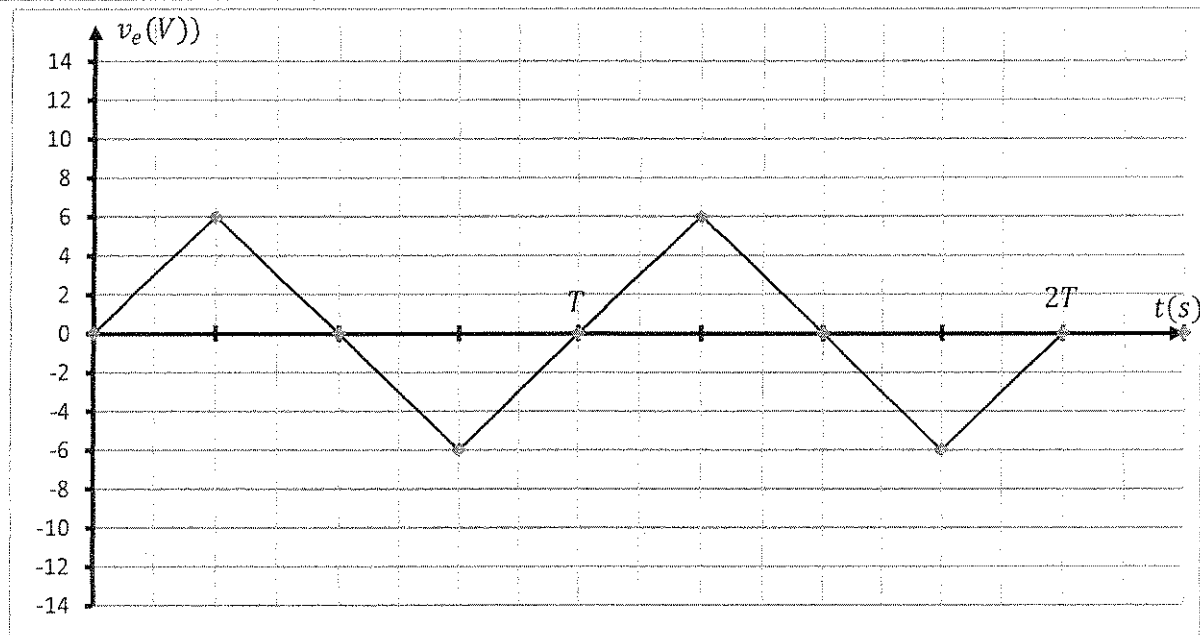
L'amplificateur opérationnel est supposé idéal et la tension sortie de l'AOP est limitée par la saturation aux valeurs extrêmes  $-V_{sat}$  et  $+V_{sat}$ .  
La diode est supposée idéale (ddp nulle en sens direct).

1) Que vaut  $v_s$  si la diode est passante?

2) Que vaut  $v_s$  si la diode est bloquée?

3) A quelle condition (sur  $v_e$ ) la diode est-elle bloquée?

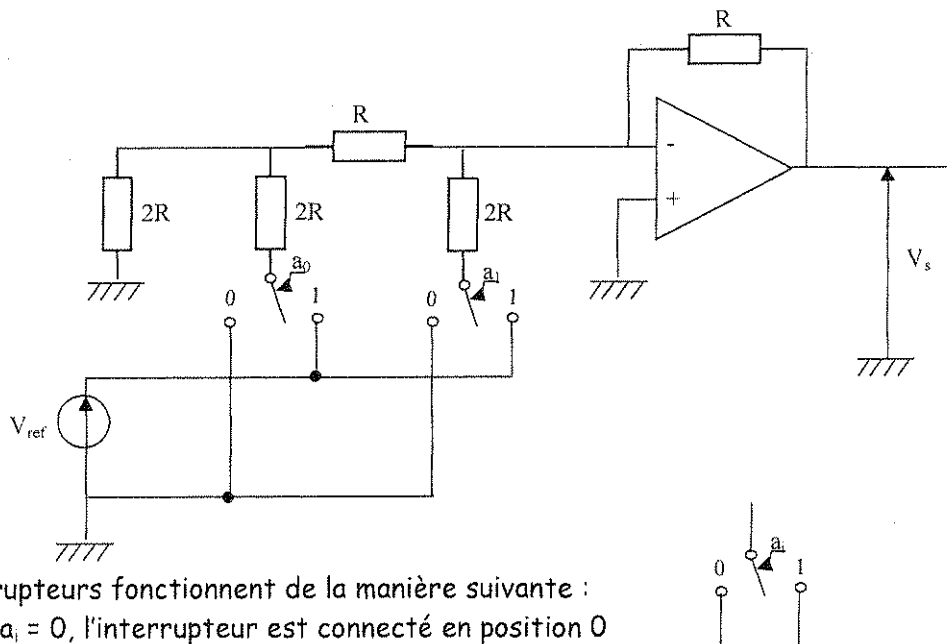
4) La tension  $v_e$  est un signal triangulaire symétrique de période  $T$  et d'amplitude  $6V$ .  
Tracer  $v_s = f(t)$  pour  $0 \leq t \leq 2T$  sur le graphe ci-dessous représentant la tension  $v_e(t)$ .





#### Exercice 4. Conversion (4 points)

On considère le montage suivant :

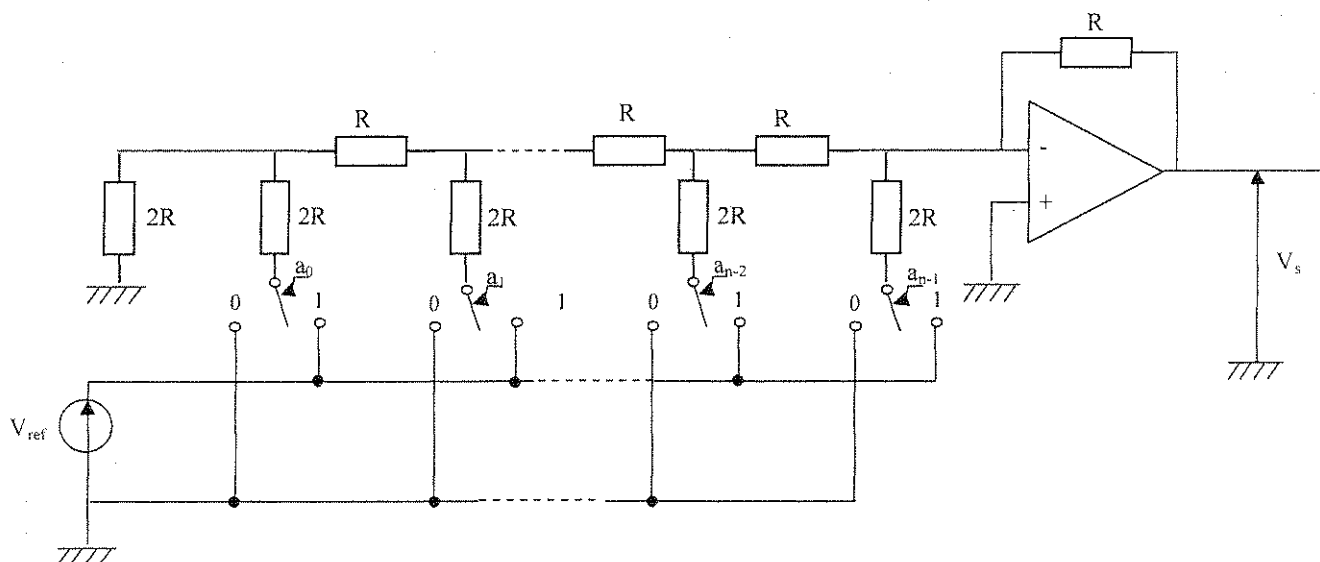


Les interrupteurs fonctionnent de la manière suivante :

- Si  $a_i = 0$ , l'interrupteur est connecté en position 0
- Si  $a_i = 1$ , l'interrupteur est connecté en position 1.

1) Donnez l'expression de  $V_s$  en fonction de  $a_0$ ,  $a_1$  et  $V_{ref}$ .

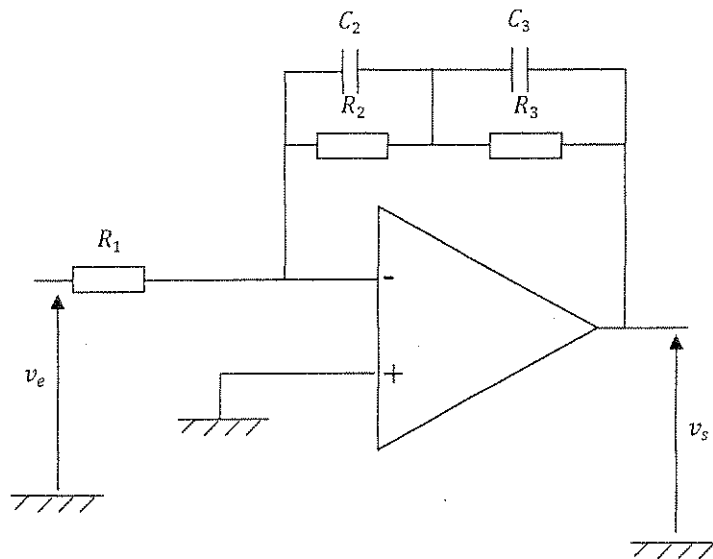
- 2) En généralisant l'expression obtenue précédemment, exprimer  $V_s$  en fonction de  $V_{ref}$  et des  $a_i$  dans le cas du montage ci-dessous. Comment appelle-t-on ce type de montage ?



### Exercice 5. Filtres actifs (4 points)

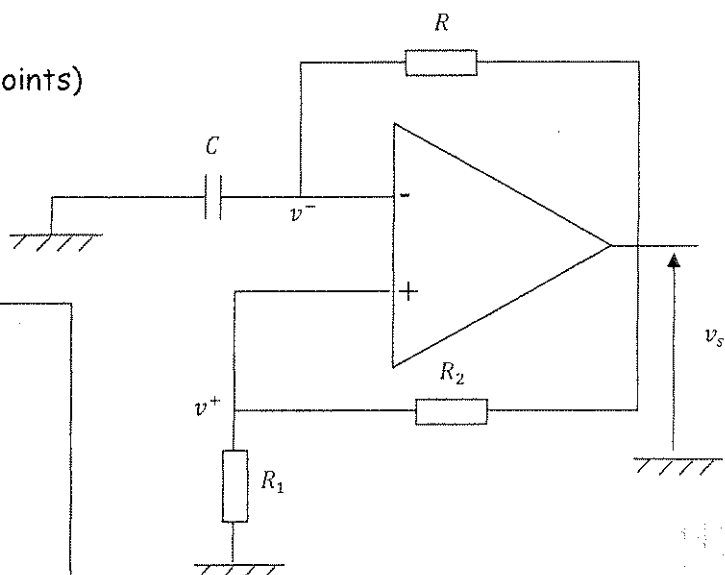
A l'enregistrement d'un disque, les sons graves sont atténués, et les sons aigus sont renforcés, pour une meilleure qualité de l'enregistrement. Par conséquent, à la reproduction, il faut accentuer les sons graves, et atténuer les aigus : c'est le rôle du filtre RIAA, dont on se propose d'étudier ici une réalisation. L'amplificateur opérationnel est supposé idéal.

Déterminez la fonction de transfert du filtre.



Exercice 6. Montage astable (1 points)

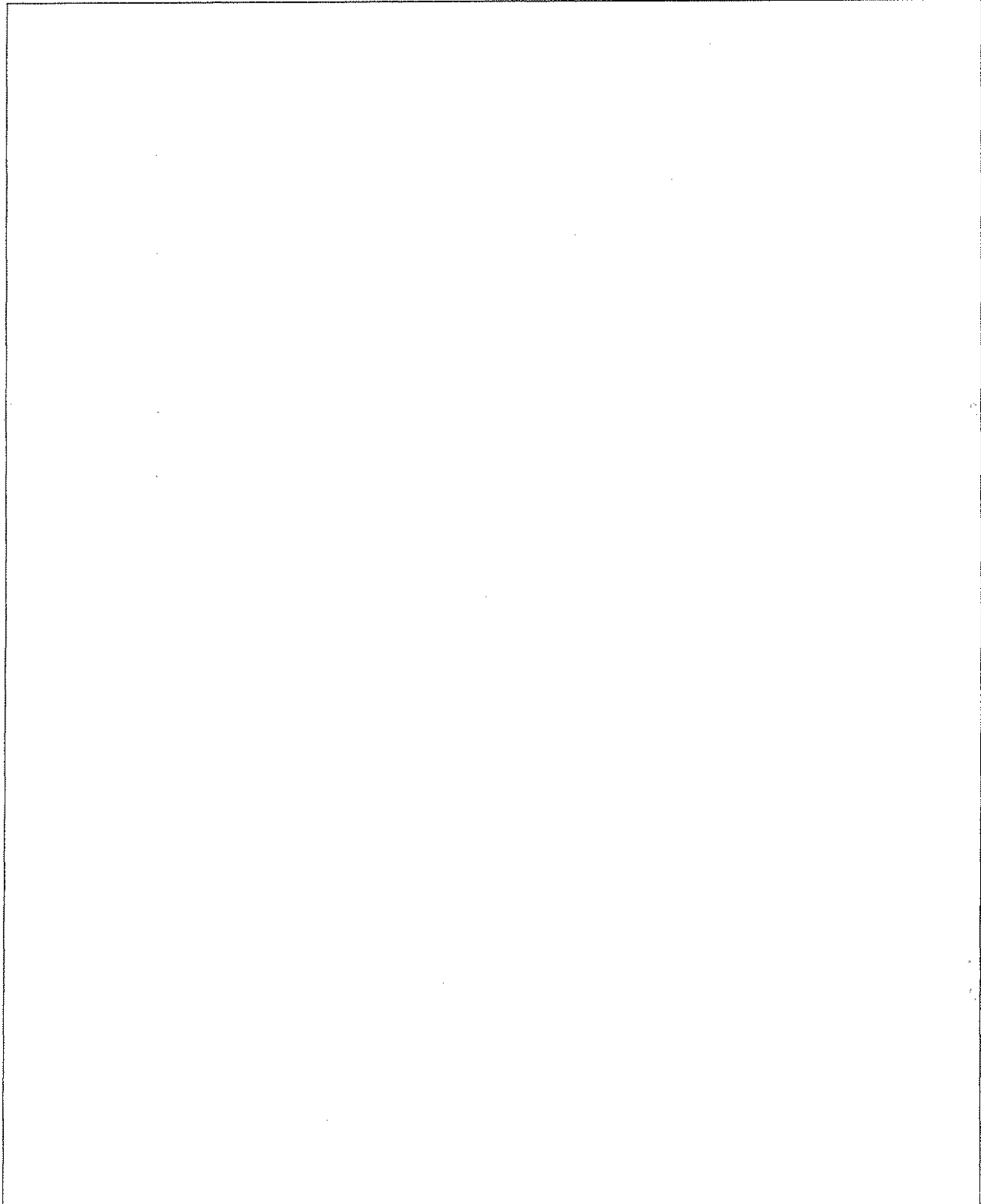
On considère le montage ci-contre :  
Déterminer l'expression de  $v^+$  en fonction  
de  $v_s$  et montrer que le potentiel  $v^-$  est  
solution d'une équation différentielle.



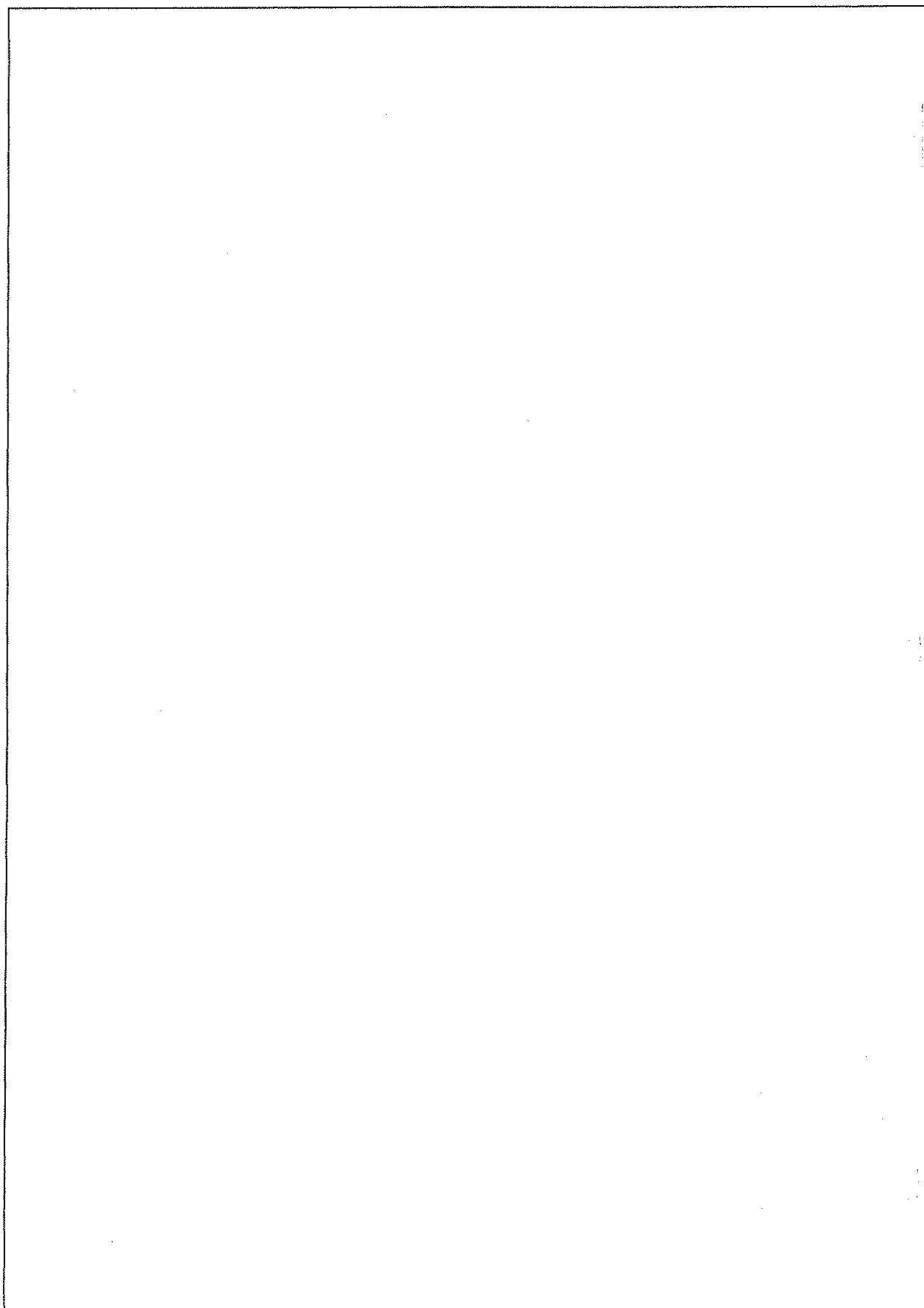
### Question Bonus (2 points)

On suppose qu'à l'instant  $t = 0$ , le condensateur est déchargé et que  $v_s = +V_{sat}$ . Déterminer et tracer en fonction du temps les variations de  $v^-$  jusqu'au point de basculement du comparateur. Le comparateur ayant basculé, déterminer et tracer les nouvelles variations de  $v^-$ .

Montrer que le comparateur basculera de nouveau et que ce processus instable se répète indéfiniment.



Si vous manquez de place, vous pouvez utiliser le cadre ci-dessous.

A large, empty rectangular box with a thin black border, occupying the majority of the page below the instruction. It is intended for the user to provide additional information or examples if space is needed.

## Partiel 2

Durée : quatre heures

Documents et calculatrices non autorisées

Nom :

Prénom :

Groupe :

### Exercice 1 (4 points)

Soit  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$  définie pour tout  $(x, y) \in \mathbb{R}^2$  par  $f(x, y) = 2x^3 + 6xy - 3y^2 + 2$ .

1. Déterminer les points critiques de  $f$ .

2. Pour chacun des points critiques, préciser s'il s'agit d'un maximum local, d'un minimum local ou d'un point-col.

[suite du cadre page suivante]

### Exercice 2 (5 points)

On considère la fonction  $f$ ,  $2\pi$ -périodique définie pour tout  $x \in [-\pi, \pi]$  par  $f(x) = |x|$ .

1. Déterminer les coefficients de Fourier  $a_0$ ,  $a_n$  et  $b_n$  associés à  $f$ .



2. Déterminer  $\sum_{n=0}^{+\infty} \frac{1}{(2n+1)^2}$

3. Déterminer  $\sum_{n=1}^{+\infty} \frac{1}{n^2}$

4. Déterminer  $\sum_{n=0}^{+\infty} \frac{1}{(2n+1)^4}$

5. Déterminer  $\sum_{n=1}^{+\infty} \frac{1}{n^4}$

### Exercice 3 (4 points)

Soient  $E = \mathbb{R}_3[X]$  muni du produit scalaire  $\langle P, Q \rangle = \int_{-1}^1 P(x)Q(x) dx$

Pour éviter des calculs inutiles, pensez à utiliser la parité car on intègre sur un intervalle centré en 0.

1. Via l'algorithme de Gram-Schmidt, orthogonaliser la famille  $(1, X, X^2)$ .

[suite du cadre page suivante]

2. Déterminer le projeté orthogonal de  $X^3$  sur  $\mathbb{R}_2[X] = \text{Vect}(1, X, X^2)$

[suite du cadre page suivante]

3. Déterminer  $\min_{(a,b,c) \in \mathbb{R}^3} \int_{-1}^1 (x^3 - ax^2 - bx - c)^2 dx$

[suite du cadre page suivante]

### Exercice 4 (5 points)

Soit  $(f_n)$  la suite de fonctions définie sur  $\mathbb{R}$  par  $f_n(x) = \frac{(-1)^n}{n} e^{-x\sqrt{n}}$

1. Etudier la convergence simple de  $(|f_n|)$  sur  $\mathbb{R}^+$ .

2. Etudier la convergence uniforme de  $(|f_n|)$  sur  $\mathbb{R}^+$ .

3. Etudier la convergence absolue de  $\sum f_n$  sur  $\mathbb{R}^+$ .

4. Etudier la convergence simple de  $\sum f_n$  sur  $\mathbb{R}^+$ .

5. Etudier la convergence normale de  $\sum f_n$  sur  $\mathbb{R}^+$ .

6. Etudier la convergence uniforme de  $\sum f_n$  sur  $\mathbb{R}^+$ .

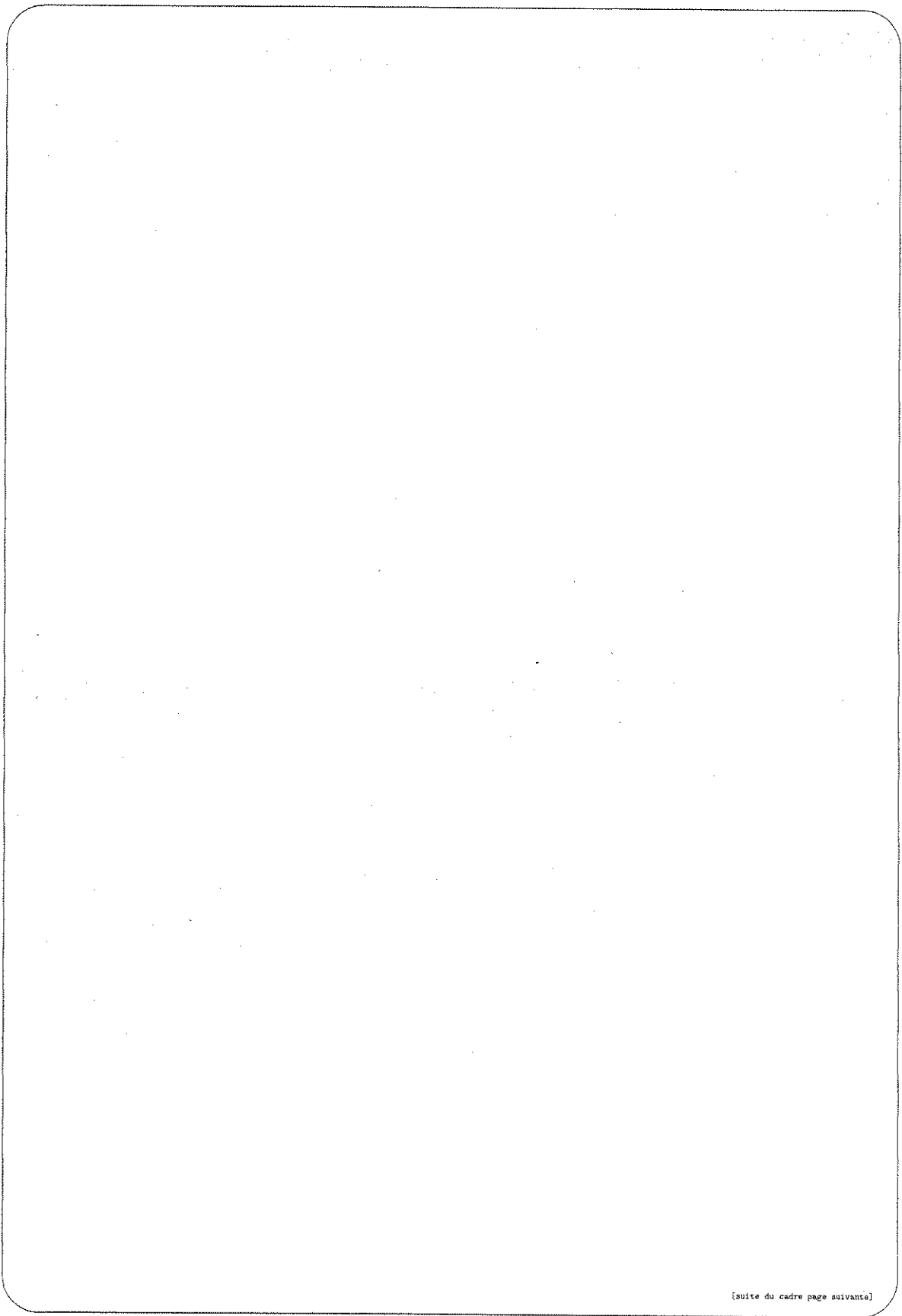
### Exercice 5 (3 points)

Soit  $a \in \mathbb{R} - \mathbb{Z}$ . On considère la fonction  $f$ ,  $2\pi$ -périodique définie pour tout  $x \in [-\pi, \pi]$  par  $f(x) = \cos(ax)$ .

1. Déterminer les coefficients de Fourier de  $f$ .

N.B. : pour tout  $n \in \mathbb{N}^*$ , vous exprimerez *impérativement*  $a_n(f)$  sous la forme  $a_n(f) = k_n \sin(a\pi)$  où  $(k_n)$  est une suite réelle (dépendant de  $a$ ) à déterminer.

[suite du cadre page suivante]



[suite du cadre page suivante]

2. En déduire  $\sum_{n=1}^{+\infty} \frac{(-1)^n}{a^2 - n^2}$

3. En déduire  $\sum_{n=1}^{+\infty} \frac{1}{a^2 - n^2}$