Exercice 1

Application directe de ce qui a été vu en cours (S2, S3) et en TD.

$x = (-1)^s (1+f) 2^e$ avec $0 \leqslant f < 1$

$1 \leqslant (1+f) < 2$

$2^e \leqslant (1+f) 2^e < 2^{e+1}$

$2^e \leqslant |x| < 2^{e+1}$

Ici $x = \sqrt{15}$ donc $|x| \approx 3,87$ donc $2 \leqslant 3,87 < 4$ donc $e = 1$

Donc $e = E - 127 = 1$ donc $E = 128 = 128 + 0 = (1000\,0000)_2$

Par ailleurs $x > 0$ donc $s = 0$

$|x| = m\, 2^e$ donc $m = \frac{x}{2^e} = \frac{\sqrt{15}}{2} \approx 1,93$ donc $f = m - 1 = 0,93649$

$16 \times f = 14,98 \to 14 = (1110)_2$ et $f_2 \approx 0,98$

$16 \times f_2 = 15,74 \to 15 = (1111)_2$ et $f_3 \approx 0,74$

$16 \times f_3 = 11,88 \to 11 = (1011)_2$ et $f_4 = 0,86$

$16 \times f_4 = 13,91 \to 13 = (1101)_2$ et $f_5 \approx 0,91$

$16 \times f_5 = 14,69 \to 14 = (1110)_2$ et $f_6 \approx 0,69$

$16 \times f_6 = 11,08 \to 11 = (1011)_2$ et $f_7 \approx 0,08$

$16 \times f_7 \approx 1,309 \to 1 = (0001)_2$ et $f_8 = 0,309$

~~$16 \times f_8 = 4,95 \approx 5 \to 5 = (0101)_2$~~

$8 \times f_8 = 2,47 \approx 2 \to 2 = (010)_2$

$4 \times 7 + 3 = 31$ donc on obtient les 31 bits en faisant 7 multiplications par $16 = 2^4$ et 1 multiplication par $8 = 2^3$

Donc

0	1000	0000	1110	1111	1011	1110	1110	1010	0001	1010
↑	4	4	0	7	7	D	E	F	5	8	A

$x > 0$ $E = 128$ $f = 0,93$

A 10
B 11
C 12
D 13
E 14
F 15

En hexadécimal, ce code s'écrit 4077DEF58A et nécessite 10 chiffres ($10 \times 4 = 40$ bits)

Il n'était pas nécessaire d'avoir correctement répondu aux premières questions pour donner la réponse à cette dernière question : 40 bits, cela fait 10 blocs de 4 bits.

exercice 2

C'est une règle de trois

$N = aT + b$

a) 0 °C → 0

0,0625 °C → 1

$(N=0, T=0°C) \Rightarrow b = 0$

$N=1$ $T=0{,}0625 \Rightarrow a = \frac{1}{0{,}0625} = 16\ °C^{-1}$

$1°C = 16 \times 0{,}0625 \rightarrow 16 \times 1 = 16$

donc $N_1 = 16$ donc pour une température de $T = T \times 1$,

on obtient un entier $N \approx T \times N_1$

b) Le microcontrôleur ne fait des calculs que sur des nombres entiers : sa unité arithmétique et logique ne ~~sait~~ permet de faire que des opérations arithmétiques sur des nombres entiers. Donc pour afficher T avec un chiffre après la virgule, on calcule 10T et on écrira un séparateur décimal (une virgule) entre le chiffre des dizaines et le chiffre des unités. (vu en TD)

~~VRAI~~ $T \approx N/N_1$ donc $10\,T \approx 10\,N/N_1 \approx 10\,N/16$

$\approx \frac{10}{16} N \approx \frac{8+2}{16} N \approx \left(\frac{1}{2} + \frac{1}{8}\right) N$

$\approx \frac{N}{2} + \frac{N}{8}$

donc pour calculer 10T, il suffit de calculer
- $\frac{N}{2}$ (décalage à droite de 1 chiffre)
- $\frac{N}{8}$ (décalage à droite de 3 chiffres)
- de les ajouter

exemple $T = 23{,}6°C \Rightarrow N = 378 \Rightarrow \frac{N}{2} = 189$ et $\frac{N}{8} = 47 \Rightarrow 189 + 47 = 236$

$\Rightarrow 236/10 = 23$

$\Rightarrow 236 \% 10 = 6$

c) Affichage Température : ← sous-programme, comme demandé dans l'énoncé.

```
DixT = (SorteCapteur >> 1) + (SorteCapteur >> 3)
debug dec DixT/10, ",", dec DixT%10, CR
return
```

L'opérateur de décalage à droite a été vu en cours et en TP

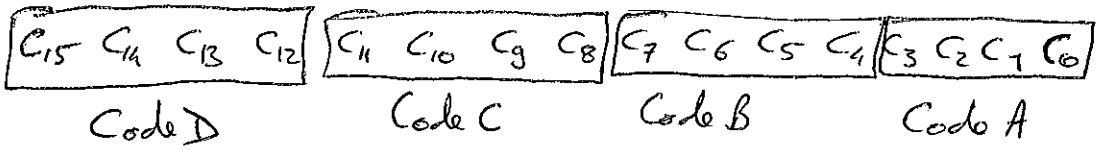
Avec DixT une variable de type word déclarée ~~avant~~ en début de programme

(Si la température est négative, il faut faire des décalages arithmétiques)

exercice 3

Un numéro de carte bleue compote 16 chiffres décimaux

a) Donc pour le stocker en BCD, il faut 16 quartets, soit
8 octets
4 mots de 16 bits



b) C_i et V_i sont des chiffres décimaux, donc le type de variable le plus petit qui peut contenir un ~~octet~~ chiffre décimal est un quartet (nib).

c) Il y a 16 chiffres décimaux C_i. Donc l'indice i va de 0 à 15
Le type de variable le plus petit qui peut contenir un nombre entier 0 et 15 est un quartet (nib). Pour savoir si i est pair ou impair, il suffit de regarder son bit de poids faible (bit de parité), en faisant i.bit0 ou $i \,//\, 2$. (vu en cours)

d) Lorsque i est pair $V_i = C_i$. Lorsque i est impair

C_i	V_i	$2C_i$	$2C_i - 9$
0	0	0	-9
1	2	2	-7
2	4	4	-5
3	6	6	-3
4	8	8	-1
5	1	10	1
6	3	12	3
7	5	14	5
8	7	16	7
9	9	18	9

$0 \leqslant C_i \leqslant 4 \Rightarrow V_i = 2C_i$

$5 \times 2 = 10 \Rightarrow 1 + 0 = 1$

$5 \leqslant C_i \leqslant 9 \Rightarrow V_i = 2C_i - 9$

$9 \times 2 = 18 \rightarrow 1 + 8 = 9$

Donc pour déterminer V_i, il suffit de comparer C_i à 4: ~~[illegible]~~

si $C_i \leqslant 4$ alors $V_i = 2C_i$
sinon $V_i = 2C_i - 9$

$= (DU)_{10}$ D dizaines, U unités

$2.C_i = 10D + U$

Si $D = 0$ $\quad D + U = U = 2C_i$

Si $D = 1$ $\quad 2C_i = 10 + U$
$2C_i = 9 + 1 + U$
donc $D + U = 1 + U = 2C_i - 9$!

e) Donc

Si i est pair alors $V_i = C_i$

sinon si $C_i \leq 4$ alors $V_i = 2C_i$

sinon $V_i = 2C_i - 9$

```
CalculVi :
  if c.bit0 = 0 then
    Vi = Ci
  elsif (Ci <= 4) then
    Vi = 2 * Ci
  else
    Vi = 2 * Ci - 9
  endif
  return
```

Cela ne correspond pas à ce qui est demandé, mais il faut reconnaître que l'on peut aussi écrire

```
CalculVi :
  Vi = ((2*Ci)/10)+((2*Ci)//10)
  return
```

c'est une description comportementale, dont la mise en œuvre fait appel au quotient et au reste de la division par 10 de $2C_i$ → très coûteux en temps de calcul !

f) La valeur maximale de V_i est 9. Donc si tous les V_i valent 9, leur somme est égale à $9 \times 16 = 144$. Pour stocker ce nombre, le type de variable le plus petit est l'octet (byte)

g)

```
CodeA   var  word
CodeB   var  word
CodeC   var  word
CodeD   var  word
Somme   var  byte
Ci      var  nib
Vi      var  nib
i       var  nib
```

CodeD = $1234
CodeC = $5678
CodeB = $9123
CodeA = $4567

~~Somme~~ i=15

C_i = CodeD.nib3 : gosub CalculVi : Somme = Vi

C_i = CodeD.nib2 : i=i-1 : gosub CalculVi ; Somme = Somme + Vi

C_i = CodeD.nib1 : i=i-1 : gosub CalculVi : Somme = Somme + Vi

C_i = CodeD.nib0 : i=i-1 : gosub CalculVi : Somme = Somme + Vi

idem pour CodeC, CodeB, CodeA ...

```
if Somme //10 = 0 then
  debug "numero valide", cr
else
  debug "numero invalide", cr
endif
end
```

Université de Nantes - IUT de Saint-Nazaire

Département de Mesures Physiques

Devoir surveillé d'Informatique d'Instrumentation II

19 novembre 2009. Durée : 1 **heure** 45.

Les trois questions sont indépendantes. Il est suggéré de lire l'énoncé *en entier* avant de répondre aux questions.

1. **(4 points)** Le circuit intégré `DIOPSIS AT572D940HF` d'ATMEL[1] est un biprocesseur (*dual core*) constitué d'un processeur 32 bits `ARM926` et d'un microcontrôleur dont l'architecture a été optimisée pour des applications de traitement numérique des signaux, de la famille `Magic DSP`. Cadencé à 100 MHz, cet ensemble a une capacité de traitement de 1,6 milliard d'opérations arithmétiques en virgule flottante par seconde (1,6 GFLOPS), car il est capable d'effectuer simultanément 16 opérations arithmétiques. Dans ce composant, les nombres réels sont codés en utilisant un format sur 40 bits, constitué d'un bit de signe, de 8 bits de codage de l'exposant et de 31 bits de codage de la mantisse (voir figure 1). Ce format est construit sur le même principe que le format IEEE 754 sur 32 bits, avec une mantisse comprenant 8 bits supplémentaires, afin d'augmenter l'exactitude des calculs.

s	e_7	e_6	e_5	e_4	e_3	e_2	e_1	e_0	d_1	d_2	d_3	...	d_{29}	d_{30}	d_{31}

Figure 1: Format de codage sur 40 bits des nombres réels en virgule flottante, utilisé par l'`AT572D940HF`. Lorsque $E = (e_7e_6e_5e_4e_3e_2e_1e_0)_2$ est compris entre 1 et 254, le nombre réel codé sous ce format est égal à $x = (-1)^s\, 2^{E-127}\, m$, avec $m = 1 + \sum_{i=1}^{31} d_i\, 2^{-i}$.

3 — 0,5

Avec ce format, quelle sera la représentation du nombre $\sqrt{15}$? Écrire ensuite ce résultat en hexadécimal.

0,5 Dans cette seconde écriture, combien de chiffres sont nécessaires ?

2. **(6 points)** Le circuit intégré `TMP121` de *Texas Instruments Inc.*[2] est un capteur linéaire de température, qui délivre un résultat de mesure codé en notation en complément à 2 sur 13 bits. Le pas de quantification de ce capteur numérique est égal à $q = 0.0625$°C. Ainsi, une température de 0°C fournira le nombre 0, tandis qu'une température de 0.0625°C fournira le nombre 1.

1 (a) Quel nombre N_1 sera présent à la sortie du capteur si la température est égale à 1°C ? En déduire la relation générale entre la température T (exprimée en degrés Celsius), le nombre N présent à la sortie du capteur correspondant à cette température et le nombre N_1 présent à la sortie du capteur lorsque $T = 1$°C .

1 (b) On veut utiliser ce composant pour réaliser, à l'aide d'un microcontrôleur, un thermomètre affichant la température avec un chiffre décimal après la virgule[3]. Expliquer pourquoi cela nécessite de calculer $10T$. Montrer que $10T$ peut être déduit de N à l'aide d'une division par 2 et une division par 8.

(c) Écrire alors un sous-programme qui, à partir de la variable `SortieCapteur` (de type `word`) correspondant au nombre délivré par le capteur, affiche (à l'écran du PC) la température exprimée en degrés Celsius, avec un chiffre après la virgule. Pour effectuer des divisions par des puissances de 2, on utilisera l'opérateur logique >> de décalage à droite (voir page 36 du polycopié).

2

[1] Voir `http://www.atmel.com/products/diopsis/default.asp`

[2] Voir `http://focus.ti.com/docs/prod/folders/print/tmp121.html`

[3] Voir J. Dimitrov, "Microcontroller converts digital temperature-sensor readings without floating point arithmetic", Design Ideas, EDN, march 5, 2009, p 43.

3. (10 **points**) Un numéro de carte bleue est constitué de 16 chiffres décimaux $(C_{15}\,C_{14}\,\ldots\,C_1\,C_0)_{10}$. La vérification de la validité de ce numéro et la détection d'une éventuelle erreur de saisie, par exemple au moment d'une transaction bancaire sur internet, peut être effectuée à l'aide de l'*algorithme de Luhn*[4]. Cet algorithme parcourt tous les chiffres C_i de droite à gauche. Pour chaque chiffre C_i, on calcule un *chiffre de vérification* V_i de la manière suivante :

   - si i est pair, $V_i = C_i$;
   - si i est impair, V_i est égal à la somme des chiffres de la représentation décimale de $2\,C_i$. Par exemple, V_i est égal à 2 quand $C_i = 1$ (car $2\,C_i = 2$) et est égal à 3 quand $C_i = 6$ (car $2\,C_i = 12$ et $1 + 2 = 3$).

   Tous les V_i sont ajoutés les uns aux autres. Si le résultat est un multiple de 10, alors le nombre peut être considéré comme valide.

   (a) Combien d'octets sont nécessaires pour coder un numéro de carte bleue en DCB ? Combien de variables de type `word` vont donc être utilisées pour stocker ce nombre ? On appellera `CodeA`, `CodeB`, `CodeC`, ... ces variables. `CodeA` correspondra aux chiffres les plus à droite, `CodeB` aux chiffres un peu plus à gauche, etc ...

   (b) Quel est le type de variable le plus adapté pour stocker les chiffres décimaux C_i et V_i ?

   (c) Quel est le type de variable le plus adapté pour stocker l'indice i ? On appellera `i` la variable contenant l'indice i. Comment peut-on savoir facilement si la variable `i` contient un nombre pair ou impair ?

   (d) À l'aide d'un tableau examinant toutes les valeurs possibles de C_i, montrer que lorsque i est impair, V_i est égal soit à $2\,C_i$ soit à $2\,C_i - 9$. Comment peut-on distinguer ces deux cas ?

   (e) Écrire alors un sous-programme `CalculVi` qui, à partir des deux variables d'entrée `i` et `Ci`, calcule la variable de sortie `Vi`.

   (f) Quelle est la valeur maximale du nombre égal à la somme des V_i ? En déduire le type de variable le plus adapté pour stocker ce nombre. On appellera `Somme` la variable utilisée pour faire ce calcul.

   (g) En déduire enfin un programme principal qui appelle le sous-programme `CalculVi` pour chacun des chiffres d'un numéro de carte bleue, calcule la somme des V_i et détermine si ce numéro est valide.

[4] Cet algorithme a été conçu en 1958 par un ingénieur informaticien de la société IBM, Hans Peter Luhn (1896-1964). Voir `http://fr.wikipedia.org/wiki/Formule_de_Luhn`

```
' {$STAMP BS2p}
' {$PBASIC 2.5}

' verification de la validite d'un numero de carte bleue
' par l'algorithme de Luhn
' F. Auger, novembre 2009

'-----------------------------------------------------------------
' declaration des variables

CodeA VAR Word : CodeB VAR Word : CodeC VAR Word : CodeD VAR Word
Ci    VAR Nib  : Vi    VAR Nib  : i     VAR Nib
Somme VAR Byte

'-----------------------------------------------------------------
' programme principal

Main:
 CodeD=$1234 :  CodeC=$5678 : CodeD=$0912 : CodeD=$3459
 GOSUB VerificationValiditeCB
 END

'-----------------------------------------------------------------
' sous-programme de verification d'un numero de carte

VerificationValiditeCB:
 ' analyse de C15, C14, C13, C12
 Ci=CodeD.NIB3 : i=15  : GOSUB CalculVi : Somme = Vi
 Ci=CodeD.NIB2 : i=i-1 : GOSUB CalculVi : Somme = Somme + Vi
 Ci=CodeD.NIB1 : i=i-1 : GOSUB CalculVi : Somme = Somme + Vi
 Ci=CodeD.NIB0 : i=i-1 : GOSUB CalculVi : Somme = Somme + Vi

 ' analyse de C11, C10, C9, C8
 Ci=CodeC.NIB3 : i=i-1 : GOSUB CalculVi : Somme = Somme + Vi
 Ci=CodeC.NIB2 : i=i-1 : GOSUB CalculVi : Somme = Somme + Vi
 Ci=CodeC.NIB1 : i=i-1 : GOSUB CalculVi : Somme = Somme + Vi
 Ci=CodeC.NIB0 : i=i-1 : GOSUB CalculVi : Somme = Somme + Vi

 ' analyse de C7, C6, C5, C4
 Ci=CodeB.NIB3 : i=i-1 : GOSUB CalculVi : Somme = Somme + Vi
 Ci=CodeB.NIB2 : i=i-1 : GOSUB CalculVi : Somme = Somme + Vi
 Ci=CodeB.NIB1 : i=i-1 : GOSUB CalculVi : Somme = Somme + Vi
 Ci=CodeB.NIB0 : i=i-1 : GOSUB CalculVi : Somme = Somme + Vi

 ' analyse de C3, C2, C1, C0
 Ci=CodeA.NIB3 : i=i-1 : GOSUB CalculVi : Somme = Somme + Vi
 Ci=CodeA.NIB2 : i=i-1 : GOSUB CalculVi : Somme = Somme + Vi
 Ci=CodeA.NIB1 : i=i-1 : GOSUB CalculVi : Somme = Somme + Vi
 Ci=CodeA.NIB0 : i=i-1 : GOSUB CalculVi : Somme = Somme + Vi

 IF Somme//10=0 THEN
  DEBUG "numero valide", CR
 ELSE
  DEBUG "numero invalide, Somme//10= ", DEC Somme//10, CR
 ENDIF
 RETURN

'-----------------------------------------------------------------
' sous-programme de calcul de Vi a partir de i et de Ci

CalculVi:
 IF i.BIT0=0 THEN        ' si i est pair
  Vi=Ci                  '  Vi=Ci
 ELSEIF Ci <= 4 THEN     ' sinon si Ci <= 4
  Vi=2*Ci                ' alors Vi=2*Ci
 ELSE                    ' sinon
  Vi=2*Ci-9              '  Vi=2*Ci-9
 ENDIF                   ' finsi
 RETURN                  ' fin de sous-programme
```