Eléments de solutions

Partiel d'INF401 Mars 2023

Numération, opération en base 2 et en complément à 2

a)

In [1]:
```
! ../_bin/show 16 3202
```

```
          Bit numbers
          1111 11
          5432 1098 7654 3210                            if natural     if signed

:         0000 1100 1000 0010        :  0xc82 -->       3202 or       +3202
```

In [2]:
```
! ../_bin/show 8 -75
```

```
          Bit numbers
          7654 3210                             if natural     if signed

:         1011 0101        :  0xb5 -->        181 or         -75
```

b)

In [3]:
```
! ../_bin/show 12 0x322
```

```
          Bit numbers
          11
          1098 7654 3210                            if natural     if signed

:         0011 0010 0010        :  0x322 -->        802 or        +802
```

In [4]:
```
! ../_bin/show 16 0xFE75
```

```
          Bit numbers
          1111 11
          5432 1098 7654 3210                            if natural     if signed

:         1111 1110 0111 0101        :  0xfe75 -->      65141 or        -395
```

c)

In [5]:
```
! ../_bin/add 8 0b11001011 0b01100011
```

```
          Bit numbers
          7654 3210                          if natural     if signed

          1100 1011 left   :  0xcb -->        203 or         -53
        + 0110 0011 right  :  0x63 -->         99 or         +99
   C=1 == 1000 0110 < c0=0   (in carries)
   V=0  ^ ---- ----
 Z=0 N=0->0010 1110  =     :  0x2e -->         46 or         +46
```

In [6]:
```
! ../_bin/subc2 8 0b11001011 0b01100011
```

```
          Bit numbers
          7654 3210                          if natural     if signed

          1100 1011 left   :  0xcb -->        203 or         -53
        + 1001 1100 right  :  0x9c -->        156 or        -100
   C=1 != 0011 1111 < c0=1   (in carries)
   V=1  ^ ---- ----
 Z=0 N=0->0110 1000  =     :  0x68 -->        104 or        +104
```

Petit dessin ascii

Rappel du squelette de code donné :

In [7]:
```
%%writefile squelette.se
    .data
point:  .byte '.'
diese:  .byte '#'
  .balign 4
tortue: .word 0
lievre: .word 0

  .text
  .global main
main:
  push {lr}

#include "tantQue.se"

  mov r1,#42
  bl EcrCar  @une étoile à la fin
  mov r0,#0  @0 pour ok en sortie
  pop {lr}
  bx lr

LD_point:  .word point
LD_diese:  .word diese
LD_tortue: .word tortue
LD_lievre: .word lievre
```

```
Writing squelette.se
```

Ajout d'une description (Makefile) du code qui suive les questions :

In [8]:
```
%%writefile Makefile
.RECIPEPREFIX = >
all: squelette.se tantQue.se es.s
> arm-linux-gnueabi-cpp -P squelette.se -o dessin.s
> arm-linux-gnueabi-gcc -static dessin.s es.s -o dessin.e
> qemu-arm  dessin.e

tantQue.se: grandeCondition.se aLaLigne.se avancerTortueLievre.se
> echo "@A FAIRE : Boucle TQ et pour, L1-13 avec :" > squeletteTantQue.se
> echo "#include \"grandeCondition.se\"" >> squeletteTantQue.se
> echo "#include \"aLaLigne.se\"" >> squeletteTantQue.se
> echo "#include \"avancerTortueLievre.se\"" >> squeletteTantQue.se
> cp -n squeletteTantQue.se tantQue.se

aLaLigne.se:
> echo "@A FAIRE : aLaLigne, L10" > aLaLigne.se

avancerTortueLievre.se:
> echo "@A FAIRE : avancerTortueLievre, L11-12 avec :" > avancerTortueLievre.se

grandeCondition.se: point.se petiteCondition.se
> echo "@A FAIRE : grande condition, L3-8 avec :" > squeletteGrandeCondition.se
> echo "#include \"point.se\"" >> squeletteGrandeCondition.se
> echo "#include \"petiteCondition.se\"" >> squeletteGrandeCondition.se
> cp -n squeletteGrandeCondition.se grandeCondition.se

petiteCondition.se: point.se
> echo "@A FAIRE : petite condition, L6-7 avec :" > petiteCondition.se

point.se:
> echo "@A FAIRE : point, L4 et 7" > point.se

clean:
>    rm *.se
```

```
Writing Makefile
```

Test :

In [9]:
```
!make -s
```

```
*
```

Pour l'instant, l'exécution donne une étoile. C'est normal. On l'a ajoutée à la fin pour controler les affichages.

d)

Le .balign est nécessaire car les données point et dieze situées avant occupent 2 octets (seulement) et qu'il faut une adresse multiple de 4 pour tortue (et lievre).

e)

In [10]:
```
%%writefile point.se
@l4 : affichage point
  ldr r1, LD_point
  ldrb r1, [r1]
  bl EcrCar
```

```
Overwriting point.se
```

Test :

In [11]:
```
!make -s
```

```
.*
```

Cela affiche le point et l'étoile de fin. C'est normal, c'est ce qui est attendu.

f)

In [12]:
```
%%writefile aLaLigne.se
@l10 : aLaLigne
  bl ALaLigne
```

```
Overwriting aLaLigne.se
```

Test :

In [13]:
```
!make -s
```

```
.
*
```

Il y a un retour (à la ligne) après le point. C'est ce qui est attendu.

g)

In [14]:
```
%%writefile avancerTortueLievre.se
@l11 et 12 : incrémentation tortue et lievre
  ldr r0, LD_tortue
  ldr r1, [r0]
  add r1, r1, #1
  str r1, [r0]
  ldr r0, LD_lievre
  ldr r1, [r0]
  add r1, r1, #4
  str r1, [r0]
```

```
Overwriting avancerTortueLievre.se
```

Pas de test d'exécution, cela ne change pas l'affichage.

h)

In [15]:
```
%%writefile petiteCondition.se
ldr r0, LD_lievre
    ldr r1, [r0]
    cmp r5, r1
    bge sinon
alors:
    ldr r1, LD_diese
    ldrb r1, [r1]
    bl EcrCar
    b fsi
sinon:
    ldr r1, LD_point
    ldrb r1, [r1]
    bl EcrCar
fsi:
```

```
Overwriting petiteCondition.se
```

In [16]:
```
!make -s
```

```
..
*
```

Il y a 1 point normalement (le point produit précédement), et un 2nd point ou un diese selon les valeurs respectives de r5 et lievre.

i)

In [17]:
```
%%writefile grandeCondition.se
    ldr r2, LD_tortue
    ldr r3, [r2]
    cmp r5, r3
    bge sinonG
alorsG:
    ldr r1, LD_point
    ldrb r1, [r1]
    bl EcrCar
    b fsiG
sinonG:
#include "petiteCondition.se"
fsiG:
```

```
Overwriting grandeCondition.se
```

In [18]:
```
!make -s
```

```
.
*
```

Il n'y a plus que 1 caractères parce car les 2 conditions sont maintenant bien codées. Il ne manque plus que les boucles pour avoir plus de caractères et le dessin.

j)

In [19]:
```
%%writefile tantQue.se
tq: ldr r2, LD_tortue
    ldr r3, [r2]
    cmp  r3, #10
    bge finTQ
    mov r5, #0
pour: cmp r5, #20
   bgt finPour
#include "grandeCondition.se"
add r5, r5, #1
   b pour
finPour:
#include "aLaLigne.se"
#include "avancerTortueLievre.se"
    b tq
finTQ:
```

```
Overwriting tantQue.se
```

In [20]:
```
!make -s
```

```
....................
.###................
..######............
...#########........
....############.....
.....###############.
......################
.......###############
........#############
.........############
*
```

Voila enfin le dessin !

P.S. : initialement, le sujet était très légèrement différent et produisait le dessin suivant

```
#....................
.####................
..#######............
...##########........
....#############....
.....################
......################
.......###############
........#############
.........############
```

Pour simplifier le sujet, un détail a été modifié. Pouvez-vous retrouver ce détail et le changer pour obtenir ce dessin ?

Détection et correction d'erreur

k)

$0b01101101$ et $0b10001001$ sont invalides.
$0b11010010$ est valide est transmet l'information $0b1010010$.

In [21]:
```
! ../_bin/show 7 0b1010010
```

```
          Bit numbers
          654 3210                          if natural     if signed

:         101 0010        :  0x52 -->       82 or         -46
```

l)

In [22]:
```
! ../_bin/show 7 22
```

```
          Bit numbers
          654 3210                          if natural     if signed

:         001 0110        :  0x16 -->       22 or         +22
```

22 est codé en C2 sur 7 bits par $0b0010110$ donc, l'octet valide est $0b10010110$.

Pour information :

In [23]:
```
! ../_bin/show 8 0b10010110
```

```
          Bit numbers
          7654 3210                          if natural     if signed

:         1001 0110        :  0x96 -->       150 or        -106
```

m)

$0b0011001-$ est valide et transmet l'information $0b0010$ (2 en décimal).
$0b1111100-$ est invalide, le bit n_0 est incorrect, après correction l'octet devient
$0b1110100-$ qu iest un octet valide et transmet l'information $0b1110$ (14 si $\mathbb{N}$, -2 si $\mathbb{Z}$).

n)

6 ($0b0110$) est codé par l'octet valide $0b0110011-$.

<fin>