Université Toulouse III - Paul Sabatier
Licence Informatique
UE Archi (KINXPB41)
Printemps 2024

Travaux pratiques
Programmation assembleur ARM

TP1 : structures algorithmiques

Exercice 1

Ecrire un programme qui calcule dans `r0` le PPCM (plus petit commun multiple) de deux nombres (non signés) contenus dans les registres `r1` (nb1) et `r2` (nb2) (qui peuvent être initialisés directement dans le debugger, ou au début du programme avec deux instructions `mov`).
L'algorithme est le suivant :

```
a = nb1;
b = nb2;
while (a != b) {
   if (a > b)
      b = b + nb2;
   else
      if (a < b)
         a = a + nb1;
}
// a est le ppcm
```

Exemples pour tester :
- le PPCM de 9 et 6 est 18
- le PPCM de 9 et 15 est 45
- le PPCM de 10 et 20 est 20

Exercice 2

Question 1.
Ecrire un programme qui dispose d'un entier `N` dans le registre `r1` et calcule dans `r0` la valeur 2^N. On suppose que la valeur dans $r1$ est inférieure ou égale à 31.
Par exemple, si `r1` contient 5, le programme doit mettre 32 ($=2^5$) dans `r0`.
La valeur 2^N peut être obtenue en partant de 1 que l'on décale `N` fois vers la gauche, puisque 2^N est représenté en binaire par un 1 en position `N`. Par exemple, si on décale 1 cinq fois vers la gauche, on obtient $(100000)_2 = (32)_{10}$.
Ecrire ce programme en utilisant une boucle (faire effectivement N décalages de une position).

Question 2.
Ecrire un programme qui dispose d'un entier `N` dans le registre `r1`, tel que N est une puissance de 2, et calcule dans `r0` la valeur $log_2(N)$.
Par exemple, si `r1` contient 32, le programme doit mettre 5 dans `r0`.

Exercice 3

Ecrire un programme qui calcule dans `r0` la racine carrée entière d'un nombre N présent dans `r1` selon l'algorithme suivant :

```
racine = 0;
while ( (racine+1)^2 ≤ N )
   racine = racine + 1;
```

Exemples pour tester :
- la racine entière de 16 est 4
- la racine entière de 19 est 4
- la racine entière de 79 est 8

Exercice 4

Ecrire un programme qui extrait un "champ de bits"du contenu de `r1` et le place dans les bits de poids faible de `r0`. La position du 1er bit (celui de poids le plus fort) et celle du dernier bit (celui de poids le plus faible) à extraire sont données respectivement dans `r2` et `r3`.

Exemples :
- si `r1=0b111000101011`, `r2=7`, `r3=2`, on veut placer dans `r0` (alignés sur la droite, c'est-à-dire à partir du bit `0`) les bits `7-2` du registre `r1`, c'est-à-dire les bits colorés en rouge ici : `r1=0b111000101010`. On doit donc obtenir `r0=0b001010`, soit `r0=0x0a`.
- si `r1=0b111000101011`, `r2=3`, `r3=0`, on doit obtenir `r0=0b1011`, soit `r0=0x0b`.

TP2 : accès mémoire

Exercice 5

Ecrire un programme qui commence par les déclarations suivantes :

```
a: .byte 8
b: .byte 12
somme: .fill 1,1
```

et qui écrit dans la variable somme la somme des variables a et b. Toutes les variables sont des nombres compris entre 0 et 255 et codés sur un octet.
Verifier que le résultat est correct dans l'onglet *Memory*.

Exercice 6

On veut trier un tableau de N entiers signés dans l'ordre croissant, selon l'algorithme du tri par sélection.
Ce tri consiste à chercher le plus petit élément du tableau et à l'échanger avec le premier élément, puis à recommencer à partir de l'élément suivant, et ce jusqu'à l'avant-dernier élément. L'algorithme est donné ci-dessous :

```
for (i=0; i<N-1 ; i++) {
     minindex = i;
     for (j=i+1; j<N ; j++)
        if (tab[j] < tab[minindex])
           minindex = j;
     if (minindex != i){
        // échanger tab[i] et t[minindex]
        tmp = tab[i];
        tab[i] = t[minindex];
        tab[minindex] = tmp;
     }
  }
```

Ecrire ce programme.
On pourra le tester avec le tableau suivant (10 éléments):

```
tab: .word 22, -12, 0, 9, 5, -1, 5, 43, 10, -10
```

Exercice 7

Le code de César permet de chiffrer un message (texte) en remplaçant chaque lettre par la lettre qui se trouve n positions plus loin dans l'alphabet (circulairement : après Z, on revient à A).
Ecrire un programme qui déchiffre le message qui se trouve à l'adresse **`message`** qui a été codé avec un décalage (n) indiqué par la variable **`dec`**, et qui ne comporte que des lettres majuscules. Le message décodé sera écrit à la place du message codé.
On rappelle que les lettres majuscules sont placées les unes à la suite des autres, dans l'ordre alphabétique, dans la table des codes ASCII.

Utiliser ce programme pour déchiffrer le message suivant qui a été produit avec un décalage de 2 (la lettre A a été transformée en C) :

```
message: .asciz "DTCXQXQWURQWXGBRCUUGTCNGZGTEKEGUWKXCPV"
dec: .word 2
```

Exercice 8

Ecrire un programme qui recopie la chaîne de caractères **`chaine1`** à l'adresse **`chaine2`** en supprimant tous les caractères qui représentent des chiffres.

```
chaine1: .asciz "JLkd2nj345bnzApdd0j9"
chaine2: .fill 255,1
```

Le résultat attendu est "JLkdnjbnzApddj".

TP3 : Sous-programmes

Exercice 9

Dans cet exercice, on s'intéresse à des octets dont la représentation **en binaire** est un palindrome. Par exemple, $(11011011)_2$ est un palindrome.

Question 1. Ecrire un sous-programme qui reçoit un octet en paramètre, dans `r1`, et renvoie, dans `r0`, un 1 si l'octet est un palindrome et un `0` sinon.

Question 2. Ecrire un programme qui utilise le sous-programme précédent pour compter le nombre de palindromes dans un tableau de 10 octets.

Par exemple, le tableau suivant contient 4 palindromes :

```
tab: .byte 0xc2, 0b11000011, 9, 252, 0xFF, 0x81, 0b10, 63, 0b11000101, 219
```

Exercice 10

On veut implémenter le jeu de la vie sur une grille 16x16.
Chaque case (*cellule*) peut être vivante (représentée par le caractère 'o') ou morte (représentée par le caractère ' ' – espace). Une cellule possède 8 voisines.

A chaque étape d'évolution (*génération*), les cellules évoluent de la manière suivante :

- une cellule **vivante** ne le reste que si **exactement 2 ou 3 de ses voisines sont vivantes**. Si aucune ou seulement une des ses voisines est vivante, la cellule meurt par sous-population. Si plus de 3 de ses voisines sont vivantes, elle meurt par sur-population.
- une cellule **morte** (re-)naît si exactement 3 de ses voisines sont vivantes.
- les cellules des bordures n'évoluent pas.

Vous trouverez un fichier définissant plusieurs grilles de départ sur Moodle.

Question 1. Ecrire un sous-programme `voisines` qui reçoit en paramètre un pointeur sur une cellule (dans `r1`) et renvoie, dans `r0`, le nombre de ses voisines vivantes. On suppose que ce sous-programme ne sera jamais appelé pour traiter une cellule du bord.

Question 2. Ecrire un sous-programme `calcul` qui reçoit en paramètre dans `r1` un pointeur sur une grille et, en utilisant le sous-programme `voisines`, écrit dans un tableau `population` d'octets le nombre de voisines vivantes de chaque cellule de la grille.

```
population: .fill 16*16,1,0
```

Question 3. Ecrire un sous-programme `generation` qui reçoit en paramètre dans `r1` un pointeur sur une grille et fait évoluer les cellules de la grille en utilisant le tableau `population`.

Question 4. Ecrire le programme principal qui appelle les sous-programmes précédents pour calculer, dans une boucle infinie, les générations successives à partir d'une grille de départ.

TP4 : Programmation des entrées/sorties

Exercice 11

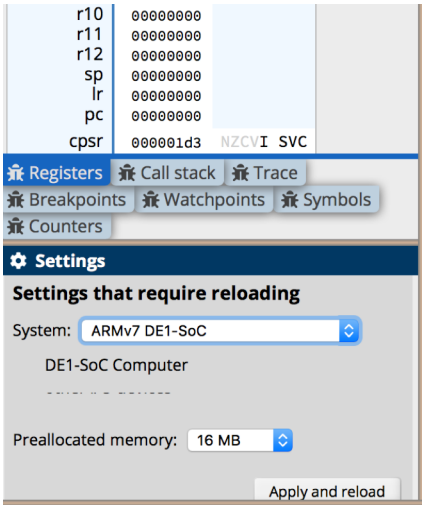
Avec cet exercice, on veut simuler le fonctionnement d'une machine à laver.
Le fonctionnement attendu est le suivant :

- un bouton ON/OFF permet d'allumer/éteindre la machine (selon qu'il est enfoncé ou non).
- 3 boutons permettent de sélectionner 3 programmes différents (voir tableau ci-dessous). Dès qu'un programme est sélectionné, le cycle commence. A la fin du cycle, la machine se met en attente du lancement d'un nouveau programme.
  Lorsque le programme inclut un lavage, la machine doit, au préalable, remplir la cuve et porter l'eau à la température souhaitée.
  Le lavage et l'essorage nécessitent un certain nombre de rotations du tambour, vers la droite ou vers la gauche, à une vitesse lente ou rapide.
- en cas d'arrêt de la machine au cours d'un cycle, la cuve est vidée si nécessaire. La position du bouton est vérifiée à chaque étape d'un cycle.

prog.	**description**	**lavage**	**essorage**
0	lavage rapide à 30°	2 × (1×RD puis 1×RG) - lent	2 × (3×RD puis 3×RG) - rapide
1	lavage lent à 60°	3 × (2×RD puis 2×RG) - lent	2 × (3×RD puis 3×RG) - rapide
2	essorage seul	-	4 × (2×RD puis 2×RG) - rapide

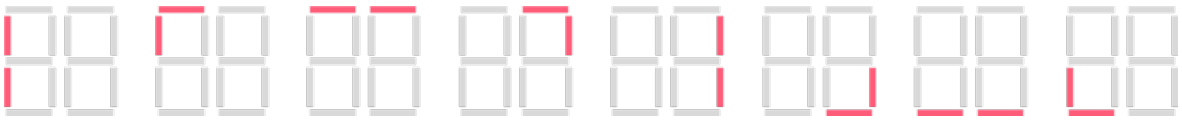
Description des 3 programmes disponibles.
RD = rotation à droite, RG = rotation à gauche

Sur CPULATOR, on peut choisir une carte cible, la carte ARMv7 DE1-SoC (la sélectionner dans le menu déroulant puis cliquer sur le bouton *Apply and Reload* qui se trouve en dessous) :



On va utiliser les périphériques de cette carte pour simuler le comportement de la machine à laver :

- le switch le plus à droite matérialisera le bouton ON/OFF.
- les trois boutons de droite permettront de sélectionner un programme
- les LEDs seront utilisées pour visualiser le remplissage de la cuve (LEDs éteintes = cuve vide / LEDs toutes allumées = cuve pleine)
- les deux afficheurs 7-segments de gauche montreront la température de l'eau quand la cuve est pleine
- les deux afficheurs 7-segments de droite serviront à visualiser la rotation du tambour selon le cycle suivant :



Question 1. Ecrire le sous-programme **`wait1`** qui réalise une attente d'environ un dixième de seconde. Il s'agit de compter de 0 jusqu'à une valeur prédéfinie (ou de décompter depuis cette valeur jusqu'à 0).
Exemple de valeur : 0x1FFFF.

Question 2. Ecrire le sous-programme **`waitn`** qui réalise une attente de *n* dixièmes de seconde. La valeur de *n* est passée en paramètre dans `r8`. Ce sous-programme doit appeler le sous-programme `wait1`.

Question 3. Ecrire le sous-programme **`remplir`** qui simule le remplissage de la cuve : il allume les LEDs une par une en partant de la droite, en respectant une attente de 5 dixièmes de secondes entre chaque LEDs. A la fin, les 10 LEDs sont allumées. Par ailleurs le sous-programme positionne la variable `cuve_pleine` définie ci-dessous à 1.

```
cuve_pleine: .byte 0
```

Question 4. Ecrire le sous-programme **`vider`** qui simule la vidange de la cuve : il éteint les LEDs une par une en partant de la gauche, en respectant une attente de 5 dixièmes de secondes entre chaque LEDs. A la fin, les 10 LEDs sont éteintes. Par ailleurs, le sous-programme remet la variable `cuve_pleine` à 0.

Question 5. Ecrire le sous-programme **`afficher_temperature`** qui affiche sur les deux afficheurs 7-segments de gauche une température passée en paramètre dans `r1` et exprimée en dizaines de degrés comprise entre 0 et 6 inclus. Par exemple, si le paramètre vaut 2, le sous-programme doit afficher 20.
Ecrire le sous-programme **`chauffer`** qui reçoit en paramètre dans `r1` une température cible exprimée en dizaines de degrés et augmente (et affiche) la température depuis 0° jusqu'à la température cible, au rythme de 10° tous les 5 dixièmes de seconde.

Question 6. Ecrire le sous-programme **`tourner_tambour`** qui simule la rotation du tambour sur les deux afficheurs 7-segments de droite selon le schéma présenté un peu plus haut. Les paramètres de ce sous-programme sont :

- dans `r1`, le sens de rotation : vers la droite (0) ou vers la gauche (1),
- dans `r2`, la vitesse de rotation : lente (0) ou rapide (1),
- dans `r3`, le nombre de rotations complètes.

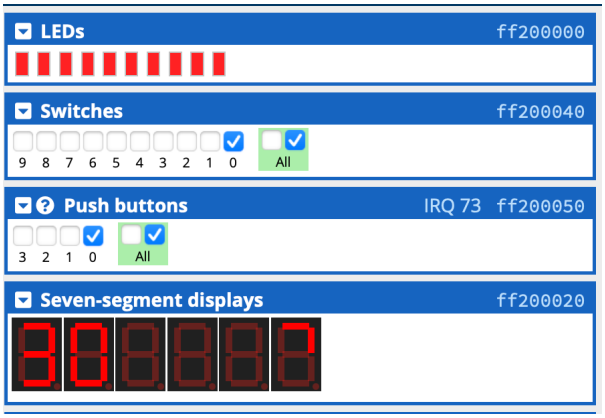
Les positions du tambour sont représentées dans le tableau suivant :

```
positions_tambour: .word 0x3000, 0x2100, 0x0101, 0x0003, 0x0006, 0x000c, 0x0808, 0x1800, 0x3000
```

NB : la première position est répétée à la fin du tableau comme point de départ pour les rotations vers la gauche. La vitesse lente correspond à 5 dixièmes de seconde entre l'affichage de deux positions successives. Pour la vitesse rapide, l'intervalle sera de 1 dixème de seconde.

Question 7. Ecrire le programme principal qui, dans une boucle infinie :

1. initialise la machine : extinction des segments repréésentant le tambour, affichage de la température 0, extinction des LEDs représentant le remplissage de la cuve.
2. attend que la machine soit allumée (appui sur le bouton ON/OFF matérialisé par le positionnement du switch 0)
3. attend qu'un programme (0, 1 ou 2) soit sélectionné avec les boutons poussoirs tout en vérifiant qu'on n'éteint pas la machine entre temps
4. lance le cycle de lavage/essorage. Après chaque étape (remplissage de la cuve, chauffage, rotations complètes du tambour, il faut vérifier si la machine n'a pas été éteinte entre temps. Si c'est le cas, il faut vidanger la cuve puis se remettre dans l'état initial.



Vue pendant le programme 0 : cuve remplie, 30°, tambour en rotation.