Université Toulouse III - Paul Sabatier
Licence Informatique
UE Archi (KINXIB41)
Automne 2024

Travaux pratiques
Programmation assembleur ARM

TP1 : structures algorithmiques

Exercice 1

Ecrire un programme qui calcule dans `r0` le PGCD (plus grand commun diviseur) 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`). On fera en sorte que nb1 soit supérieur ou égal à nb2.
L'algorithme est le suivant :

```
a = nb1;
b = nb2;
while (b != 0) {
   r = a;
   while (r ≥ b)
      r = r - b;
   a = b;
   b = r;
}
// a est le pgcd
```

Exemples pour tester :
- le PGCD de 45 et 25 est 5
- le PGCD de 48 et 18 est 6
- le PGCD de 24 et 8 est 8

Exercice 2

Ecrire un programme qui compte, dans r0, le nombre de 1 dans la représentation binaire d'un nombre qui se trouve dans r1 (on initialisera r1 à partir du debugger ou par une instruction en début du programme).

Par exemple, si r1=$(130)_{10} = (10000010)_2 = (82)_{16}$, le résultat doit être r0=2.
Si r1=0xDE000000, on doit obtenir r0=6.

Exercice 3

Ecrire un programme qui détermine si le nombre placé dans r1 est un carré, et écrit 1 dans r0 si c'est le cas et 0 sinon.

Par exemple, 81 est un carré (9^2) mais 66 n'en est pas un.

Exercice 4

Ecrire un programme qui copie dans chacun des quatre registres r0, r1, r2 et r3 un octet de l'entier contenu dans r5 (initialisé avec le debugger). L'octet de poids faible ira dans r0, le suivant dans r1, etc.

Par exemple, si r5=0xAABBCCDD, on doit obtenir : r0=0xDD, r1=0xCC, r2=0xBB, r3=0xAA.

TP2 : accès mémoire

Exercice 5

Ecrire un programme qui commence par les déclarations suivantes :

```
a: .byte 22
b: .byte 12
op: .byte '+'  pourrait aussi contenir '-' ou '*'
.align
res: .fill 1,4
```

et qui écrit dans la variable res le résultat de l'opération op sur les variables a et b. Toutes les variables sont des nombres compris entre 0 et 255 et codés sur un octet.

Par exemple, si op contient '+', le programme doit écrire 34 dans res ; si op contient '-', il doit écrire 10 ; si op contient '*', il doit écrire 264.

Verifier que le résultat est correct dans l'onglet *Memory*.

Exercice 6

On veut compacter un tableau d'entiers, c'est-à-dire supprimer tous ses éléments égaux à 0. Par exemple, le tableau suivant :

0	12	5	0	-2	8	9	15	0	-3	0	0	0	4	8	42

est compacté en

12	5	-2	8	9	15	-3	4	8	42

Ecrire ce programme en utilisant un adressage indexé. Le résultat est écrit dans un deuxième tableau. On peut utiliser un index de lecture et un index d'écriture (par exemple, quand l'index de lecture vaut 1, ce qui correspond à l'élément dont la valeur est 12, l'index d'écriture vaut 0 puisque cet élément est le premier du tableau résultat). A la fin du programme, l'index d'écriture représente le nombre d'éléments dans le tableau compacté.

Exercice 7

Ecrire un programme qui transforme tous les caractères d'une chaîne en majuscules (la chaîne initiale peut déjà contenir quelques majuscules ainsi que des caractères qui ne sont pas des lettres – ils doivent être conservés).
Indications :

- Les lettres, minuscules et majuscules, sont placées les unes à la suite des autres, dans l'ordre alphabétique, dans la table des codes ASCII. Un caractère est donc une lettre minuscule si son code ASCII est compris entre 'a' ('a' représente le code ASCII de la lettre a) et 'z'.
- 'A'-'a' = 'B'-'b' = 'C'-'c' ... donc on peut passer du code ASCII d'une lettre minuscule à celui de la lettre majuscule correspondante en lui ajoutant 'A'-'a'.

Tester le programme sur la chaine suivante :
chaine: .asciz "#ABZ]aez}"
Après exécution du programme, la chaine devra contenir : "#ABZ]AEZ}"
NB : il ne s'agit pas de créer une deuxième chaîne mais de modifier la chaîne initiale.

Exercice 8

Ecrire un programme qui :

1. initialise les N octets d'une zone mémoire qui commence à l'adresse Z1 à une valeur VAL
2. recopie, octet par octet, le contenu de la zone Z1 vers une zone Z2 (comme le fait la fonction memcpy). Envisager tous les cas possibles : les zones se chevauchent ou pas, la zone de départ est située avant ou après la zone d'arrivée dans la mémoire.

Tester le programme avec les configurations suivantes (vérifier le bon fonctionnement en visualisant la mémoire) :

- N=32 – Z1=0x1000 – VAL=0xbb – Z2=0x1040
- N=32 – Z1=0x1040 – VAL=0xcc – Z2=0x1000
- N=64 – Z1=0x1000 – VAL=0xdd – Z2=0x1020
- N=64 – Z1=0x1020 – VAL=0xee – Z2=0x1000

TP3 : Sous-programmes

Exercice 9

1. Ecrire un sous-programme ABS qui reçoit en entrée (dans le registre r1) un entier signé et renvoie (dans r0) sa valeur absolue. On utilisera, pour cela, l'instruction rsb.
2. Ecrire un programme qui calcule la somme des valeurs absolues des éléments (entiers signés) d'un tableau. La valeur absolue d'un élément sera calculée en appelant le sous-programme ABS.
   Appliquer ce programme au tableau suivant de 10 éléments:
   tab: .int 9, -4, 27592, 0, -27592, 9, -4, 27592, 0, -27592
   Le résultat attendu est 110394 (0x1AF3A).

Exercice 10

On veut écrire un programme qui reçoit en entrée une date (sous la forme jour - mois - année) et calcule le quantième (c'est-à-dire le numéro du jour dans l'année, compris entre 1 et 366) correspondant.
Exemples :
- le quantième du 3 mars 2001 est 62
- le quantième du 3 mars 1996 est 63
- le quantième du 8 janvier 2004 est 8

Dans un premier temps, on ne tient pas compte du fait que certaines années sont bissextiles.

1. Ecrire le sous-programme DEBUT qui reçoit dans le registre r1 un numéro de mois (valeur entière) et renvoie dans r0 le quantième du jour précédant ce mois. Les valeurs de ces quantièmes sont indiquées ci-dessous. On suppose que le numéro de mois passé en paramètre est toujours correct, c'est-à-dire compris entre 1 et 12.

Mois	1	2	3	4	5	6	7	8	9	10	11	12
Quantième	0	31	59	90	120	151	181	212	243	273	304	334

2. Ecrire le programme principal qui calcule (en appelant le sous-programme DEBUT) le quantième d'une date dont les numéros de jour, de mois et d'année sont rangés en mémoire (3 variables). Le résultat sera rangé en mémoire dans une 4e variable.
   jour: .int 3
   mois: .int 3
   annee: .int 2001
   resultat: .fill 1,4

On veut maintenant modifier ce programme pour gérer correctement les années bissextiles.

1. Ecrire le sous-programme DIVISIBLE qui reçoit deux valeurs entières a et b passées dans r1 et r2 et renvoie dans r0 le résultat 1 si a est multiple de b et 0 sinon.
2. Ecrire le sous-programme BISSEXTILE qui reçoit dans r1 un numéro d'année (valeur entière) et renvoie dans r0 le résultat 1 si l'année est bissextile et 0 sinon. Une année est bissextile si elle est divisible par 4 mais pas par 100 ou si elle est divisble par 400. Ce sous-programme devra appeler le sous-programme DIVISIBLE.
3. Ecrire le programme principal.

Exercice 11

On veut traiter des images de 16 x 8 pixels. Chaque pixel est représenté par un octet qui peut prendre une des deux valeurs suivantes : 0 ou 35 (caractère '#'). Par exemple, le fichier image.s définit l'image suivante :

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

Lorsque l'on calcule l'érosion d'une image, tout pixel (hors bordure) qui possède au moins un voisin blanc (code ASCII 0) devient blanc. Les voisins d'un pixel sont les 8 pixels qui l'entourent, ce qui inclut les pixels en diagonale. Les bordures de l'image ne sont pas modifiées.

1. Ecrire un sous-programme qui calcule le résultat de l'érosion sur un pixel. Ce sous-programme aura les paramètres suivants :
   - en entrée : adresse du pixel dans l'image d'origine (passée dans r1)
   - en sortie : valeur du pixel après érosion (dans r0)
2. Ecrire un programme qui calcule l'érosion d'une image.

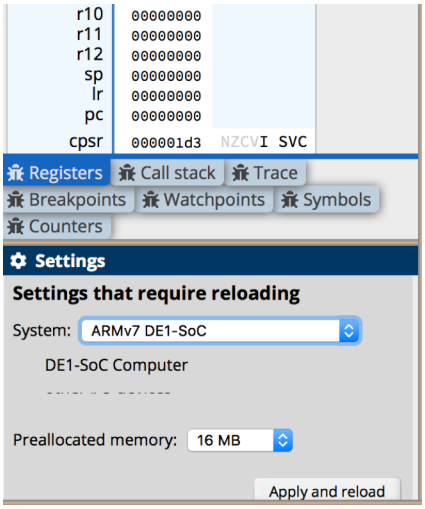
Dans le cas de l'exemple proposé ci-dessus, le résultat doit être :

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

TP4-5 : Programmation des entrées/sorties

Les systèmes informatiques sont généralement constitués, en plus du processeur et de la mémoire, de dispositifs périphériques permettant d'interagir avec l'utilisateur.

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



Cette carte comporte un certain nombre de périphériques, parmi lesquels :

- des LEDs
- des boutons poussoir (push buttons)
- des afficheurs 7-segments

On peut contrôler ces dispositifs par programme, en lisant/modifiant des registres d'entrée/sortie (E/S) qui se trouvent dans leur interface.

Ces registres d'E/S ne sont pas connus du processeur, et ne peuvent donc pas être désignés directement (par leur nom) dans un programme. Par contre, ils sont associés à une adresse qui n'est pas utilisée par la mémoire (si les adresses sont sur 32 bits, elles peuvent désigner 4 GO - si on a moins de 4 GO de mémoire, on peut utiliser certaines adresses pour désigner autre chose q'un octet en mémoire). L'accès à ces registres d'E/S se fait donc par des instructions ldr/str aux adresses qui leur correspondent.

Contrôle des LEDs Un registre associé à l'adresse 0xff200000 contient l'état des LEDs. Quand le bit i de ce registre est à 1, la LED correspondante est allumée ; quand il est à 0, la LED est éteinte.

Pour allumer la LED 0, on peut donc écrire :

```
ldr r8,=0xff200000 @ r8 = 0xff200000
                   @ on ne peut pas écrire mov r8,#0xff200000
mov r0,#1 @ un 1 en position 0 pour allumer la LED 0
str r0,[r8] @ on écrit les valeurs souhaitées pour les LEDs dans le registre
          @ de contrôle des LEDs
```

On obtient alors le résultat suivant :

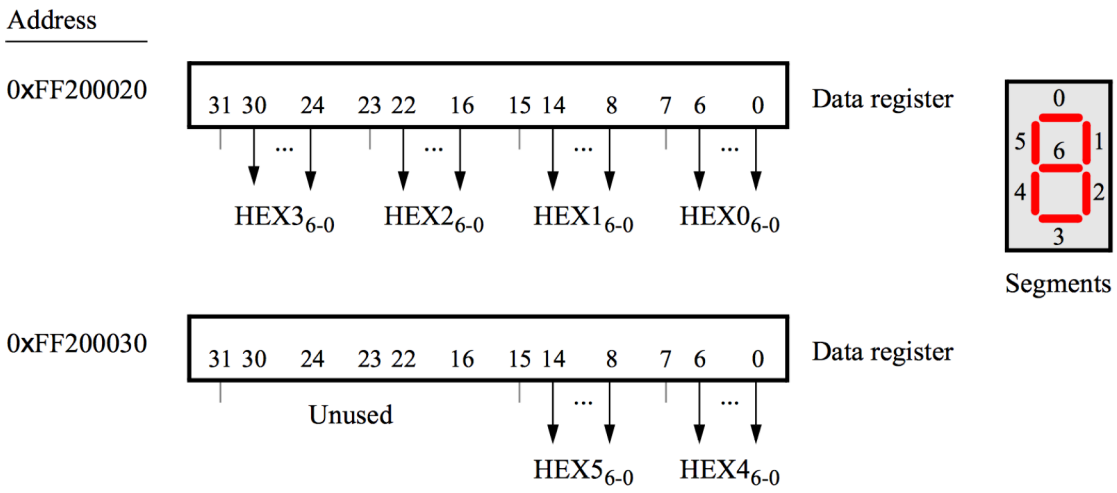
Autre exemple : pour allumer les LEDs 1 et 2, on doit écrire un 1 en position 1 et en position 2, soit la valeur 0b110 (ou 0b0000000110) dans le registre de contrôle des LEDs.

Contrôle des boutons poussoirs Pour les 4 boutons poussoir, un registre similaire se trouve à l'adresse 0xff20005c. Le bit i de ce registre indique si le bouton i a été pressé (dans ce cas, le bit est à 1). Par exemple, si on a pressé les boutons 1 et 3, le registre contient 0b0000...0001010.

Dans le simulateur, pour presser un bouton, il faut cliquer deux fois : la première fois, le carré devient bleu ; la deuxième fois, il redevient blanc. Le bit correspondant passe alors à 1 et y reste tant que le programme ne le remet pas à 0. Pour mettre le bit i à 0, il faut écrire un 1 en position i dans le registre.

Contrôle des afficheurs 7-segments Il y en a 6 et ils sont contrôlés par deux registres, l'un à l'adresse 0xff200020, qui contrôle les afficheurs 0 à 3, et l'autre à l'adresse 0xff200030, qui contrôle les afficheurs 4 et 5.

Pour chaque afficheur, il y a 7 bits à positionner comme représenté sur le schéma suivant :



Par exemple, pour afficher 0 sur l'afficheur 0, c'est-à-dire allumer les segments 0 à 5 de cet afficheur, on peut écrire :

```
ldr r8,=0xff200020
mov r0,#0b0111111 @ segment 6 éteint, segments 5 à 0 allumés
str r0,[r8]
```

Valeurs à écrire pour afficher les chiffres de 0 à 3 :

Chiffre	0	1	2	3
Valeur	0b0111111	0b0000110	0b1011011	0b1001111

Exercice 12

Fonctions d'attente

- Ecrire un sous-programme **wait1** qui permet d'attendre un laps de temps de l'ordre de une seconde. Ce sous-programme devra simplement compter de 0 à une valeur prédéfinie (ou décompter de cette valeur jusqu'à 0). Ce sous-programme n'a pas de paramètre.

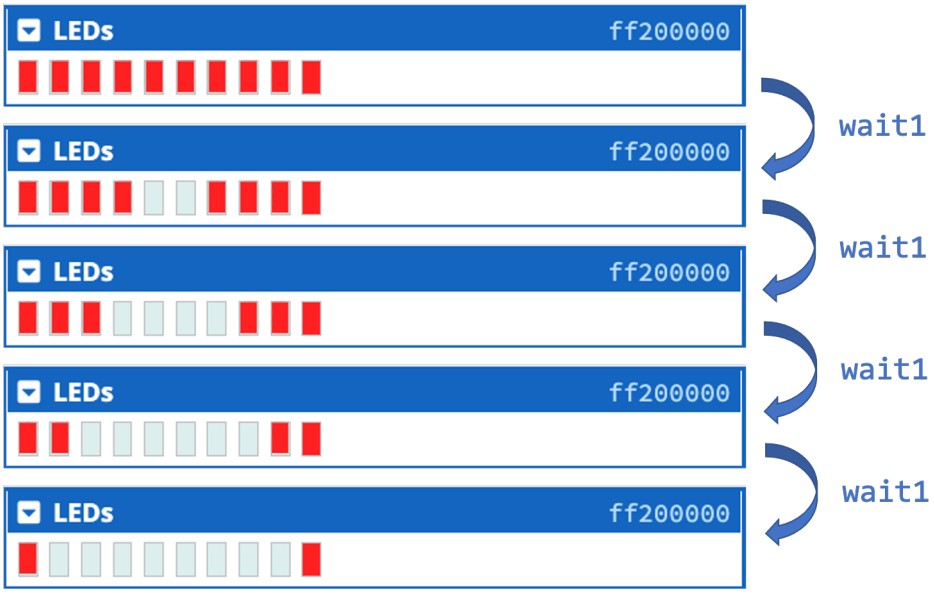
  Valeur recommandée : 0x100000 (vous pouvez essayer d'autres valeurs si ça ne convient pas)

- Ecrire un sous-programme **wait** qui permet d'attendre un laps de temps multiple de celui de wait1. Ce sous-programme reçoit en paramètre, dans r0, le nombre de fois où il doit appeler wait1.

Fonctions de simulation de la porte de l'ascenseur (LEDs)

La position de la porte de l'ascenseur sera visualisée par les LEDS : toutes les LEDs allumées quand la porte est fermée et les 8 LEDs centrales éteintes quand la porte est ouverte.

- Ecrire un sous-programme **ouvrir_porte** qui contrôle le passage de la position fermée à la position ouverte selon le schéma ci-dessous. Ce sous-programme n'a pas de paramètre. Utiliser le sous-programme wait1 pour imposer un délai entre deux positions consécutives.



- De manière similaire, écrire un sous-programme **fermer_porte** qui contrôle le passage de la position ouverte à la position fermée.

Fonctions d'affichage sur les afficheurs 7-segments

L'afficheur 0 (celui qui se trouve le plus à droite) servira à afficher l'étage auquel se trouve la cabine de l'ascenseur.

- Ecrire un sous-programme **affiche_étage** qui reçoit en paramètre, dans r0, le numéro d'un étage, et l'affiche.

L'afficheur 5 (celui qui se trouve le plus à gauche) servira à visualiser le mouvement de la cabine. Il sera éteint lorsque la cabine est arrêtée à un étage, et affichera deux symboles différents quand elle sera en train de monter ou descendre (par exemple, 'u' pour up et 'd' pour down).

- Ecrire un sous-programme **affiche_mouv** qui reçoit en paramètre, dans r0, le mouvement (par exemple 0 pour l'arrêt, 1 pour la montée, 2 pour la descente), et l'affiche.

Fonctions de lecture des boutons poussoirs

- Ecrire un sous-programme **lit_boutons** qui lit le registre des boutons poussoirs, pour déterminer sur lesquels on a appuyé, et réinitialise ce registre à 0 en écrivant des 1 dans les 4 bits. On veut que ce sous-programme soit non bloquant : si aucun bouton n'a été pressé depuis la dernière réinitialisation, il renvoie 0.

Programme de simulation de l'ascenseur

- En utilisant les sous-programmes écrits dans les questions précédentes, écrire un programme qui simule le fonctionnement d'un ascenseur qui dessert 4 étages (de 0 à 3). Son fonctionnement est le suivant :
  - au départ, la cabine est arrêtée à l'étage 0 et la porte est fermée
  - à tout moment, le mouvement de la cabine (arrêt, montée, descente) est indiqué sur l'afficheur 7-segments n°5 (le plus à gauche). Si la cabine est arrêtée, l'étage correspondant est indiqué sur l'afficheur 7-segments n°0 (le plus à droite)
  - pour demander le déplacement de la cabine vers un étage donné, on presse (2 clics) le bouton poussoir correspondant
  - l'ascenseur ne change de direction que lorsqu'il n'y a plus d'étage demandé dans la direction courante (et si un étage est demandé dans l'autre direction).