Université Toulouse III – Paul Sabatier
L2 Informatique

Architecture 2
19 mai 2013
Semestre 4

Projet — Diviseur entier performant

Antoine de Roquemaurel (Groupe 2.2)

Ce fichier à pour but de bien comprendre l'organisation et le développement du projet. Les fichiers sources en assembleur sont présent dans le fichier `./fastdiv.s`.

1 Compilation et tests

Afin de compiler facilement le projet, un fichier *Makefile* à été créé, ainsi la simple commande `make` compilera le projet.
Celle-ci créera ainsi un fichier `./fastdiv.elf` pouvant être exécuté à l'aide d'un environnement d'exécution ARM.

Afin de tester facilement le projet, le plus simple est de lancer le projet avec un débuggeur (insight par exemple), de mettre des points d'arrêts après chaque appel de programme et ainsi vérifier après chaque appel que les registres r0 et r1 contiennent les valeurs prévues.

2 Algorithmes et codes

2.1 Algorithme générale de la division

```
-- Implémentation performante de la division X = Q * Y + R
-- Si Y = 0, branchement sur le sous-programme div_by_0
-- à fournir par l'utilisateur de cette fonction
--  r0 (entree) = X (Dividende)
--  r1 (entree) = Y (Diviseur)
--  r0 (sortie) = R (Reste)
--  r1 (sortie) = Q (Quotient)
fonction fastdiv(entree r0, entree r1, sortie r0, sortie r1)
debut
  if r1 = 0 alors
    div_by_0();
  fin si;

  ---- Gestion du cas ou le diviseur est une puissance de 2
  ---- Cf algo 2.2

  -- Si on arrive ici, ce n'est pas une puissance de 2

  r2 <- 0;
  si r0 != 0 alors
    r1 <- r1 < 16;

    pour r3=1 à 16 faire
      r0 <- r0 < 1;
      r0 <- r0 - r1;
      r2 <- r2 < 1;
      si r0 >= 0 alors
        r2 <- r2 + 1;
      sinon
        r0 <- r0 + r1;
      fin si;
    fin pour;
  fin si;
  r1 <- r0 < 16;
  r0 <- r2;
fin
```

```
@ Implémentation performante de la division X = Q * Y + R
@ Si Y = 0, branchement sur le sous-programme div_by_0
@ à fournir par l'utilisateur de cette fonction
@ r0 (entree) = X (Dividende)
@ r1 (entree) = Y (Diviseur)
@ r0 (sortie) = R (Reste)
@ r1 (sortie) = Q (Quotient)
fastdiv:
      STMFD sp!,{r2-r10}
      CMP   r1, #0
      BEQ   div_by_0



@ Gestion du cas ou le diviseur est une puissance de 2
@ Cf algo 2.2

@ Si on arrive ici, ce n'est pas une puissance de 2

      MOV   r2, #0
si1:    CMP   r0, #0
      BEQ   fsi1
      MOV   r1, r1, LSL #16

      MOV   r3, #1
pour:   CMP   r3, #16
      BHI   fpour
      MOV   r0, r0, LSL #1
      SUB   r0, r0, r1

      MOV   r2, r2, LSL #1
si2:    CMP   r0, #0
      BLT   sinon2
      ADD   r2, r2, #1
      B   fsi2
sinon2:
      ADD   r0, r0, r1
fsi2:

      ADD   r3, r3, #1
      B   pour
fpour:
fsi1:
      MOV   r1, r0, LSR #16
      MOV   r0, r2
finFct:
      LDMFD sp!,{r2-r10}
      MOV   pc,r14
```

2.2 Algorithme si le diviseur est une puissance de 2

2.2.1 Algorithme de remplissage de la table des puissances

```
-- Remplis la table à 256 entréees
-- Donnant pour chaque valeur entre 0 et 255,
-- n si la valeur est une puissance de 2, 32 sinon
-- r0 (entree) : Adresse de début du tableau
fonction fillTab(entree r0)
debut
  r0 <- r0 + 1;
  r1 <- 0;
  r2 <- 1;
  tantque r2 < 512 faire
    Mem32(r1) <- r0;
    r1 <- r1 + 1;
    r0 <- r0 + r2;
    r2 <- r2 << 1;
  fin tantque;
fin
```

```
@ Remplis la table à 256 entréees
@ Donnant pour chaque valeur entre 0 et 255,
@ n si la valeur est une puissance de 2, 32 sinon
@ r0 (entree) : Adresse de début du tableau
fillTab:
      STMFD sp!,{r0-r3}
      ADD   r0, r0, #1
      MOV   r1, #0
      MOV   r2, #1
tq3:    CMP   r2, #512
      BHS   ftq3
      STRB  r1, [r0]
      ADD   r1,r1,#1
      ADD   r0, r0, r2
      MOV   r2, r2, LSL #1
      B   tq3
ftq3:
      LDMFD sp!,{r0-r3}
      MOV   pc, r14
endFillTab:
```

2.2.2 Algorithme de vérification si c'est une puissance de 2

```
r4 <- @tabPow2;
r5 <- r1 < 24; -- r5 contient le 1er octet
r6 <- Mem32(r4+r5);

r5 <- r1 < 8;
r5 <- r5 < 24; -- r5 contient le 2nd octet
r7 <- Mem32(r4+r5);

r5 <- r1 < 16;
r5 <- r5 < 24; -- r5 contient le 3eme octet
r8 <- Mem32(r4+r5);

r5 <- r1 < 24;
r5 <- r5 < 24; -- r5 contient le 4eme octet

r10 <- 0;

si r6 != 32 alors
  r10 <- r6 + 24;
fin si;

si r7 != 32alors
  si r10 = 0 alors
    goto fsitab;
  fin si;
  r10 <- r7 + 16;
fin si;

si r8 != 32 alors
  si r10 = 0 alors
    goto fsitab;
  fin si;
  r10 <- r8 + 8;
fin si;

si r9 != 32 alors
  si r10 = 0 alors
    goto fsitab;
  fin si;
  r10 <- r9;
fin si;

si r10 != 0 alors
  r0 <- r0 < r10;
  r1 <- 0;
 -- Fin de la fonction de division fastdiv
  retourner;
fin si;

fsitab:
```

```
      ADR   r4, tabPow2
      MOV   r5, r1, LSR #24 @r5 contient ↩
          l'octet de poids fort
      LDRB  r6, [r4, r5]
      MOV   r5, r1, LSL #8
      MOV   r5, r5, LSR #24 @r5 contient ↩
          le 2nd octet de poids fort
      LDRB  r7, [r4, r5]
      MOV   r5, r1, LSL #16
      MOV   r5, r5, LSR #24 @r5 contient ↩
          le 3e octet de poids fort
      LDRB  r8, [r4, r5]
      MOV   r5, r1, LSL #24
      MOV   r5, r5, LSR #24 @r5 contient ↩
          l'octet de poids faible
      LDRB  r9, [r4, r5]
      MOV   r10, #0
powsi1:   CMP   r6, #32
      BEQ   fpowsi1
      MOV   r10, r6
      ADD   r10, r10, #24
fpowsi1:
powsi2:   CMP   r7, #32
      BEQ   fpowsi2
      CMP   r10, #1
      BEQ   fsitab
      MOV   r10, r7
      ADD   r10, r10, #16
fpowsi2:
powsi3:   CMP   r8, #32
      BEQ   fpowsi3
      CMP   r10, #1
      BEQ   fsitab
      MOV   r10, r8
      ADD   r10, r10, #8
fpowsi3:
powsi4:   CMP   r9, #32
      BEQ   fpowsi4
      CMP   r10, #1
      BEQ   fsitab
      MOV   r10, r9
fpowsi4:

tabsi4:   CMP   r10, #0
      BEQ   fsitab
      MOV   r0, r0, LSR r10
      MOV   r1, #0
      B   finFct
fsitab:
```