ALM2 – Processim
Etude d'un processeur simple en PC/PO

LECHEVALIER Maxime & GATTAZ Rémi

03 Avril 2015

Table des matières

1 Prise en main et observations

1.1 Questions sur le code des exemples :

Que font les instructions du traitant de ré-initialisation (reset) et comment est effectué le branchement au début du programme utilisateur ?

Le traitant reset initialise la pile en mettant la valeur 0 dans le registre sp et met en place dans slr l'adresse à laquelle le programme utilisateur commence (0x80). C'est avec l'instruction rti qui va mettre slr dans pc et autoriser les interruptions que le saut vers le progamme utilisateur va alors se faire.

Ou est effectué l'initialisation du pointeur de pile sp ?

L'initialisation de la pile se fait dans le traitant reset.

Quelle partie du programme exemple_it_irs empile des octets ? Comment y accède-t-on ?

C'est dans le traitant irq que l'on empile des octets. On y accède lorsqu'une intéruption matérielle survient et que l'instruction situé à l'adresse 0x02 est exécuté.

A quelle adresse est stocké le premier octet empilé ?

Avant le premier store, sp a été intialisé à 0x00 puis on lui a soustrait 0x01. SP valait donc 0xFF lors du premier store. C'est donc à cette adresse que la valeur a été stocké.

1.2 Questions sur la PC du processeurs (le microcode) :

Que se passe-t-il au cours de la première microinstruction exécutée après la mise sous tension ou la réinitialisation du processeur. Quelle registres (ou partie de registres) sont initialisées ?

La première micro instruction du microcode initialise pc à 0 et modifie le flag I du registre cpsr. Le but est de bloquer les interruptions et de faire charger le traitant reset qui se trouve dans le programme.

Combien de microactions sont exécutées entre deux passages à fetch ?

- pour une instruction de calcul reg op=reg ? Entre 23 et 26
- pour une instruction de calcul reg op=reg ? Entre 25 et 28
- pour une instruction blcond ? Entre 19 et 23

2 Load et Store

```
!*****************************************************************!
! Gestion des instructions d'accès à la mémoire : load et store     !
! load/store   [reg, reg_ou_#4bits]                                 !
!                                                                   !
!            ir        mk2                                        !
! Format : code_op8  oooo gggg                                      !
!          adresse = reg_gggg + oooo_reg ou #oooo                   !
!                                                                   !
!           76 5 4  3210                                            !
! Code_op8: 01 E #  rval                                            !
!                                                                   !
!  Lecture   oooo  :   déjà fait, dans mb                           !
!  Lecture   reg_gggg  :   mk1  (lbbus)                             !
!  Ecriture  reg_dest  :   MK1  (lcbus)                             !
!*****************************************************************!
            ! dans mb se trouve déja la valeur du déplacement
            ! on veut charger dans ma, base+depl
            ! la valeur base est dans le registre indiqué par les 4 bits de poids
            ! faible de mk2
memoire:    mk1 = mk2
            mk1 = mb + lbbus ! dans mk1 se trouve depl(mb) + base(lbbus)
            0 = 0 + mk1 ema ! Chargement de mk1 dans ma
            ! Test si il s'agit d'un load ou d'un store
            j5 store

! *********** Load ***********
! Lecture puis deplacement de la valeur dans mb dans le bon registre
load:       read
            read
            mk1 = ir
            mk1 = shl (mk1 + mk1)
            mk1 = shl (mk1 + mk1)
            lcbus = mb
            jp fetch
            ! Il y a répétition de code (ranger) mais on gagne 1 micro-instruction

! *********** Store ***********
! mise en place de la valeur à écrire dans mb puis ecriture
store:      mk1 = ir
            0 = 0 + lbbus emb
            write
            write
            write
            jp fetch

branch_autres:  j6  autres
```

Le calcul de l'adresse à laquel se fait le load et le store se fait avec une adresse de base auquel on ajoute un déplacement. Dans le cas ou l'on utilise un registre pour définir le déplcaement, la valeur est en complément à 2 sur 16 bits. En revanche, lorsque l'on utilise une valeur directe, le déplacement est une valeur positive codé sur 4 bits.

Pour pouvoir gérer la valeur immédiate comme une valeur en complément à 2, il faudrait avoir une instructions ou une méthode facile qui permet de faire une copie du bit de signe de la valeur immédiate. Dans le cas présent, cela impliquerait un nombre de micro instructions important, nous avons donc décidé de ne pas supporter ceci.

Le programme de test de ces deux fonctionalités est test mem.s

3 Interruptions

3.1 Interruptions Logicielles

```
!*****************************************************************!
! Gestion des autres instructions                                 !
!                                                                 !
! 7654 3210                                                       !
! 1100          mov  cpsr/spsr, reg                               !
! 1101          mov  reg, cpsr/spsr                               !
! 1111 0        rti                                               !
! 1111 1        swi                                               !
!                                                                 !
! 1110 0        seti                                              !
! 1110 1        clri                                              !
!*****************************************************************!

autres:         j5  instr_it
                ! ce sont les mov sur les psr
                ! on se ramène à un traitement de move
                mb = labus
                jp calcul

instr_it:       j4 it_log
                j3 clear

set:            seti
                jp fetch

clear:          clri
                jp fetch


it_log:  j3 swi

! Retour depuis interruption
! Il y a répétition de code (clear) mais on gagne 1 micro-instruction
rti:            f = ac ! Restoration de cpsr depuis spsr
                pc = a ! Restoration de pc depuis slr
                clri ! autorise à nouveau les interruptions
                jp fetch

! Départ interruption logicielle
swi:            ji fetch ! Si une interruption est déja en cours, on saute
                a = pc ! Sauvegarde de pc dans slr
                ac = f ! Sauvegarde de cpsr dans spsr
                seti ! Bloque la gestion des interruptions
                pc = shl(1 + 1) ! Mise en place du vecteur swi dans pc (0x4)
                jp fetch
```

3.2 Interruptions Matérielles

```
fetch:      ji fetch_normal ! ignore les interruption matérielle si seti
            jq irq_mat ! gère une interruption matérielle

fetch_normal: pc = 0 + pc ema
            read
            ...
            ...
            ...
! Départ interruption matérielle
! Il y a répétition de code avec swi mais on gagne 1 micro-instruction
irq_mat:    a = pc ! Sauvegarde de pc dans slr
            ac = f ! Sauvegarde de cpsr dans spsr
            seti ! Bloque la gestion des interruptions
            pc = 1 + 1 ! Mise en place du vecteur it matérielle dans pc (0x2)
            jp fetch
```

Le programme de test pour les interruptions matérielles et logicielles est test it.s

4 Remarques générales

Il y a plusieurs sections dans lesquelles il est indiqué que le code en question est une copie d'une portion de code qui se trouve ailleurs. Il serait possible de factoriser ces portions de code mais cela impliquerait que le nombre de micro-instructions pour les instructions liés augmenterait. Etant donnée que notre microcode est court et donc ne remplit pas la mémoire dédié du processeur, nous avons donc décidé de privilégier les performances et copier un peu de code.