PROCESIM : étude d'un processeur simple en PC/PO

Groupe Architectures Logicielles et Matérielles

Document Etudiants

1 Mise en œuvre sur PROCSESIM

Les registres de procesim ont un nom et un numéro (exemple : TIR, numéro 3) et sont moins nombreux que ceux du jeu d'instructions. On trouvera donc :

- Des registres PROCESIM qui correspondent à des registres du jeu d'instructions (pas forcément de même nom).
- Des registres PROCESIM qui ne correspondent pas à des registres du jeu d'intructions (dont constantes, temporaire(s), registre d'instruction).
- Des registres du jeu d'instructions qui ne sont pas simulés par manque de registres dans la PO de PROCESIM : R2, R5, R6, R12, R13, R14, R15.

Procesim		Jeu instr	Autre
Num	Nom	Nom	
0	pc	R0/PC	
1	sp	R1/SP	
2	ir		Rinstr p. fort
3	tir	R3/LR	
4	0	R4/0	
5	1		Cte 1
6	ff		Cte -1
7	A	slr	

Procesim		Jeu instr	Autre
Num	Nom	Nom	
8	B	R8	
9	C	R9	
A	D	R10	
B	E	R11	
C	F	cpsr	
D	mk1		temporaire
E	mk2		Rinstr p. faible
F	ac	spsr	

Les principales autres limitations de mise en œuvre sur PROCESIM sont liées aux indicateurs (pas d'indicateur V) et aux opérations réalisables dans l'UAL, ce qui explique que certaines instructions de calcul ne soient pas gérées.

1.1 Structure interne des registres (figure 1)

Chaque registre R_j est piloté par trois signaux : chargement en fin de cycle de la valeur présente sur le bus c (chR_j), sortie de son contenu via des portes trois états, sur les bus A (R_j^A) et B (R_j^B).

1.2 Sélection des registres

La sélection des registres est réalisée par 3 démultiplexeurs 1 vers 16, dont 3 sorties seulement sont représentées sur la figure 2. Leurs entrées de sélection recoivent (selon la valeur des sorties ba,bb et bc de la PC) un numéro de registre fourni soit par les sorties sa, sb et sc (chacune sur 4 bits) de la pc, soit par quatre bits du contenu du registre mk1 ou mk2. Les registres sont sélectionnés et les indicateurs de l'UAL mémorisés dans les bascules μC,μN et μZ uniquement s'il ne s'agit pas d'une microinstruction de séquencement.

Le microassembleur masm traduit chaque microinstruction symbolique en binaire. Pour la commande du bus a :

- pc = ir + ac sera traduit par : ba=0 (utilisation de sb) et sa=2 (ir)
- pc = labus + ac sera traduit par : ba=1 (utilisation de mk2) et valeur indifférente pour sa.

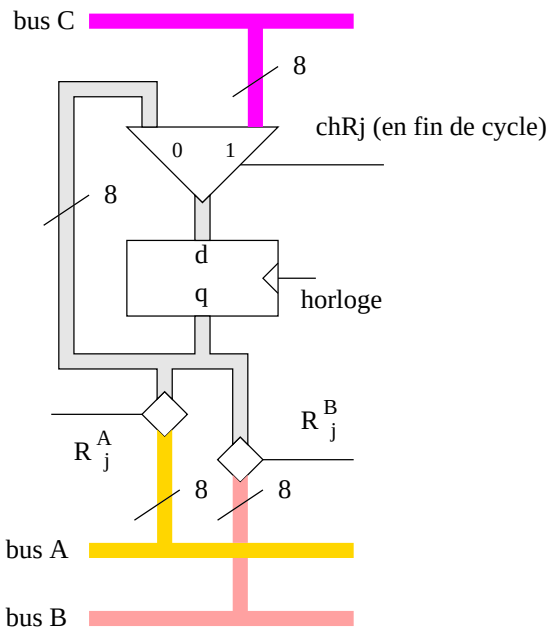

FIG. 1 – Organisation interne d'un registre de la PO procesim

1.3 Commande de l'unité de calcul

De même les deux bits de choix de l'opération réalisée dans l'UAL peuvent être fournis (selon la valeur de la sortie bu de la partie commande) soit par la sortie ual (sur deux bits) de la PC, soit par deux bits du registre mk2.

2 Compléments sur le séquencement des microinstructions

La partie commande de la machine procesim est de type microprogrammé (cf cours "partie_opérative/partie_commande"). De l'état numéro x peuvent partir :

1. une transition inconditionnelle vers l'état de numéro $x+1$ (calculé par l'incrémenteur)
2. une transition inconditionnelle vers l'état de numéro $y \neq x+1$
3. deux transitions conditionnelles : vers l'état de numéro $y \neq x+1$ avec la condition ee et vers l'état $x+1$ avec la condition $\overline{ee}$, avec ee parmi IR_0, IR_1, ..., IR_7, uN, uC, uZ, IRQ, I_de_F, constante 1 (condition toujours vraie).

Les deux derniers types de transitions ne peuvent avoir lieu qu'avec une microinstruction de séquencement (format/usaut=1) qui ne modifie aucun registre de la PO. Le numéro de l'état suivant y (μadresse) est alors encodé dans les 8 bits de droite de la microinstruction, les bits bu, bc, bb et ba n'ayant dans ce aucun effet sur les registres de la PO. Le signal choisi comme condition de transition est donné par le champ cond de la microinstruction.

A titre d'exemple, la microinstruction symbolique **j4 suite** est traduite par :

1. μadresse = numéro de l'état associé à la μétiquette suite
2. format/μsaut = 1 (ceci est une microinstruction de séquencement)
3. cond = 1100 (12_{10}) (sélection du signal IR_4 comme condition de μbranchement).

3 Présentation de PROCESIM

PROCESIM est un logiciel d'enseignement pour l'apprentissage du fonctionnement interne des PROCEsseurs par la SIMulation. Il peut être utilisé à deux niveaux :

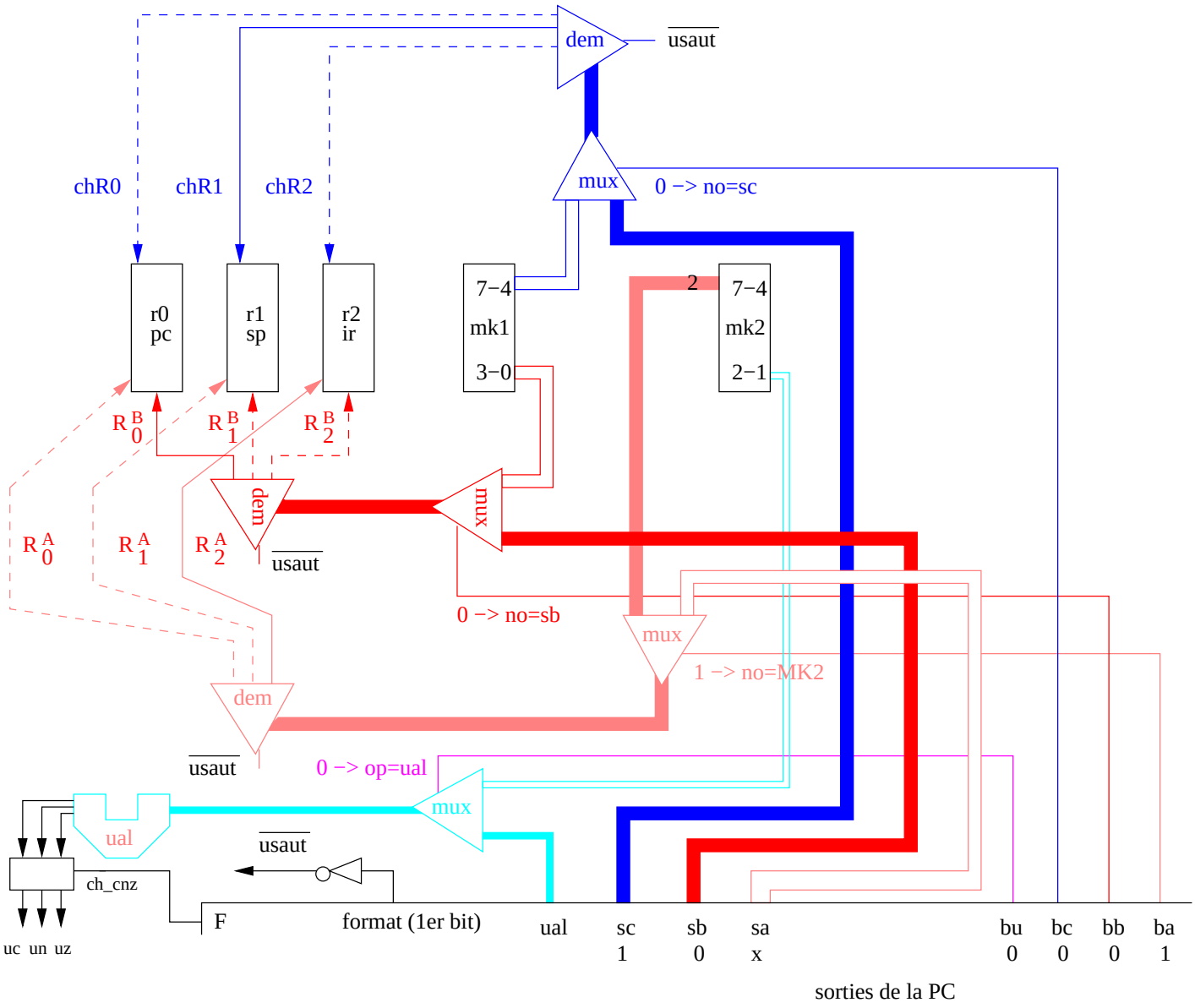

FIG. 2 – Circuit de paramétrisation pour la sélection des registres

- L'observation de machines déjà conçues,
- La création de nouvelles machines.

Il est inspiré d'un outil commercial assez ancien et des améliorations ont été apportées en gardant la simplicité initiale.

Dans les deux cas l'utilisateur dispose d'une architecture matérielle interne figée. Elle est décrite dans la suite. Elle permet de réaliser des machines réelles proches de cette architecture. L'architecture de la partie opérative est organisée autour d'un système à 3 bus, d'un opérateur, de 16 registres d'usage plus ou moins généraux. Des connexions existent qui permettent les liaisons avec une mémoire. Toutes les données et les adresses sont codées sur 8 bits.

La description de la partie contrôle du processeur se fait dans un langage. (microcode) Pour distinguer cette description de la partie contrôle et les instructions du langage machine du processeur on dit que les "actions" élémentaires qui apparaissent dans le graphe de l'automate de contrôle sont des *micro-instructions*. La séquence des micro-instructions constitue un *micro-programme*. Le micro-programme a une forme textuelle et une forme binaire.

L'écran de visualisation montre le contenu des registres, l'état d'utilisation des bus et le contenu d'une partie de la mémoire.

Dans le mode création de nouvelles machines on utilise aussi la visualisation de la mémoire de

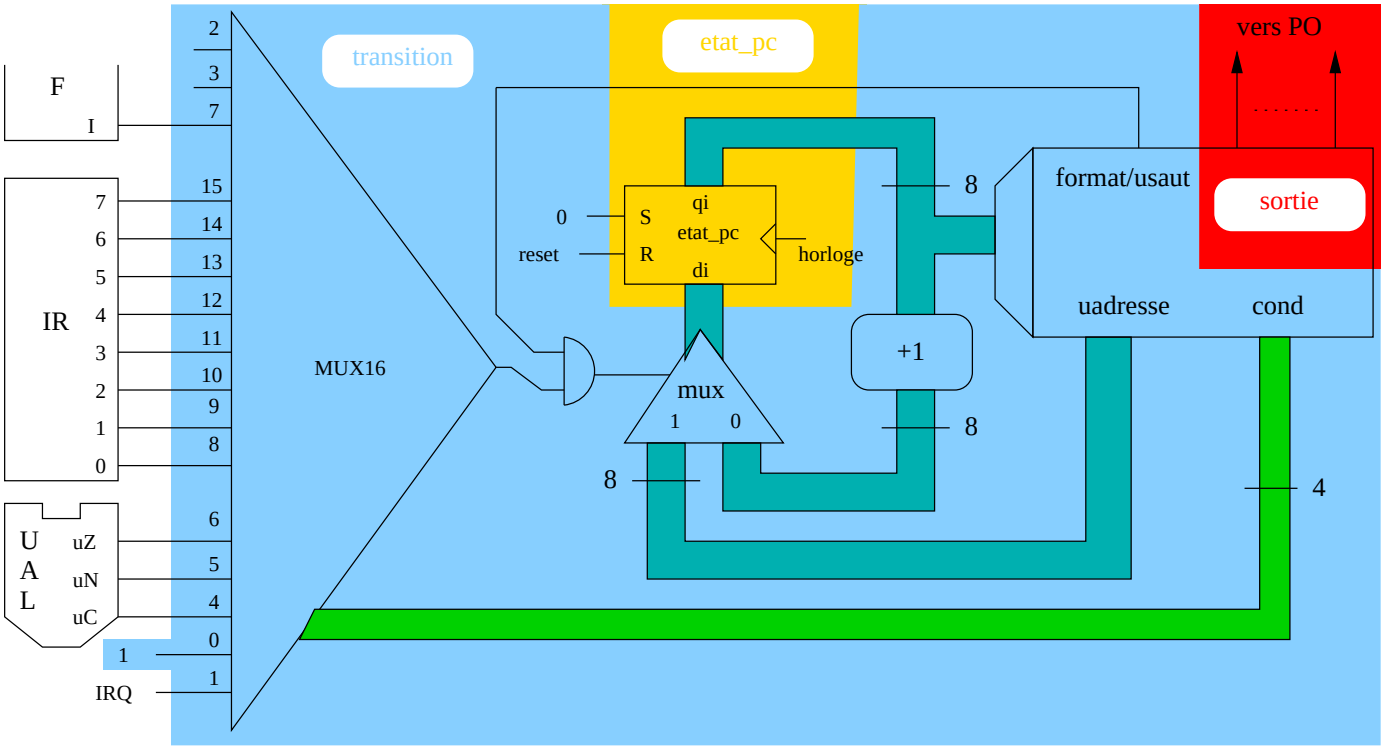

FIG. 3 – L'automate microprogrammé partie commande

micro-programme réalisant la partie contrôle. En mode d'observation cette mémoire est présente et l'utilisateur peut en partie l'ignorer.

3.1 Mode Observation

En mode d'observation il faut
- écrire un programme en langage machine de la machine étudiée et des données en hexadécimal dans la mémoire. (Ce programme doit être sauvegardé dans un fichier textuel puis, depuis le simulateur il est lu à partir du fichier)
- charger l'image binaire d'un microprogramme. (Cette image peut être lue depuis un fichier)
- démarrer l'exécution. (en faisant avancer l'horloge clock)
- si des requêtes d'interruptions font partie de l'étude, le bouton IRQ simule l'occurence d'une telle requête.

Le fichier contenant le programme en langage machine est un fichier de 256 lignes. Il comporte sur chaque ligne une valeur écrite en hexadécimal, sans avoir besoin de préciser 0x.. . Cette valeur peut être suivie d'un commentaire qui n'est pas pris en considértation. C'est par exemple le mnémonique de l'instruction, plus facile à lire que l'hexa. Le fichier doit être suffixé par ".ob" : `nomdefichier.ob`, ...

Mode d'emploi succint de PROCESIM

Lancement du simulateur :

`procesim`

Dans l'environnement de simulation :

• Icône disquette RAM : (fichier suffixé .ob) chargement en Ram du programme objet assemblé s'il existe. Une fenêtre permet de visualiser le contenu de la RAM.

• Icône disquette ROM (fichier suffixé .mob) chargement en ROM du microprogramme. Une fenêtre pemet de visualiser le contenu de la ROM et de suivre l'évolution du microprogramme.

• Pas à pas : le bouton clock permet d'exécuter la prochaine micro-instruction.

• Instruction : Le bouton Instr permet d'exécuter une suite de micro-instructions jusqu'à l'adresse en ROM spécifiée. Attention il faut sortir de ce mode par le bouton cancel pour pouvoir utiliser d'autres boutons.

• Reset : le bouton reset permet une réinitialisation 'aléatoire' de tous les registres du processeur (comme le ferait une broche reset sur un vrai circuit)

• Le bouton traces permet d'obtenir une liste des transferts de registres.

3.2 Mode Création de nouvelles machines

3.2.1 Architecture de la partie contrôle

Les micro-instructions sont stockées dans une ROM. La boîte appelée `"logique saut ?` contient le circuit logique combinatoire qui permet, à partir du champ `condi` de la micro-instruction, des micro-Flags et du registre instruction, de déterminer s'il faut, ou non, sauter à la micro-instruction dont l'adresse est dans le champ `adresse suivante`.

Format des micro-instructions : Les champs des micro-instructions peuvent être classés suivant deux fonctionnalités. Certains champs contiennent les valeurs des commandes de la partie opérative dans l'état courant de l'automate de contrôle. Dautres champs concernent l'enchaînement entre les états de l'automate : si la condition de saut est réalisée (champ condi), on passe à la micro-instruction dont l'adresse se trouve dans un champs de la micro-instruction, sinon on passe à la micro-instruction suivante.

Mode d'emploi succint de Proce-sim

Ecrire un microprogramme dans un fichier suffixé `.muc` .
Micro-assemblage du fichier nomdefichier.muc contenant le microcode

`masm nomdefichier.muc`

produit le microcode dans le fichier `sortie.mob` qu'il est conseillé de renommer, par exemple du même nom que le .muc dont il est issu !

Puis on se ramène à une observation.

3.2.2 Architecture du processeur noyau dans PROCESIM

Dans PROCESIM il existe un noyau de processeur, c'est à dire un ensemble de possibilités de processeur. On peut le personnaliser en définissant son jeu d'instructions (son architecture logicielle) et la façon dont ces instructions sont définies pour ce noyau (architecture de l'exécution matérielle)

Le noyau comporte une partie opérative et un séquenceur ou partie contrôle.

3.2.3 Partie opérative

Elle comprend 18 registres.
MA est le registre d'adresse vers la mémoire.
MB est le registre de données.
Les 16 autres registres sont
`PC, SP, IR, TIR, 0, 1, ff, A, B, C, D, E, F, MK1, MK2, AC`
Ils numérotés de 0 à 15 dans l'ordre des noms ci dessus. (Ainsi par exemple A correspond au numéro 7).
Le registre 0 contient la constante 0, 1 contient 1 et ff contient -1.
On fera attention à ne pas confondre le nom et le numéro. 0 est le registre numéro 5.
PC est le compteur programme.(Program Counter)

SP est le pointeur de pile. (Stack Pointer)
IR est le registre instruction. (Instruction Register)
TIR est l'extension du registre instruction dans certaines machines.
F est le registre de Codes de conditions arithmétiques. (Flag)
MK1 et MK2 ne sont utilisés que dans le mode de création de machines.

La partie opérative comporte un opérateur arithmétique et logique (UAL) et un opérateur permettant de faire des décalages de 1 bit à droite ou à gauche. Toute sortie de l'opérateur de calcul passe dans l'opérateur de décalage.

La partie opérative comporte 3 bus : les bus nommés A, B et C.
A et B permettent d'acheminer le contenu des 16 registres vers les entrées de l'UAL. (A vers l'entrée gauche et B vers l'entrée droite). C permet d'acheminer les sorties de l'opérateur de décalage vers les 16 registres.

L'information passant sur le bus B peut être mémorisée dans le registre MA. La sortie du décaleur peut être mémorisée dans le registre MB. L'information présente dnas le registre MB peut être transmise vers l'entrée gauche de l'UAL.

3.2.4 Primitives de base pour l'écriture des micro-programmes

Le langage d'écriture des microprogrammes (microcode) permet de décrire les graphes de contrôle. Les primitives de base pour l'écriture de micro-programmes s'appellent des micro-instructions.

Il y a trois sortes de micro-instructions :
les micro-instructions de calcul,
les micro-instructions de séquencement.
les micro-instructions permettant l'accès à la mémoire.

3.2.5 Micro-instructions de calcul

Les micro-instructions de calcul expriment les actions qui ont lieu dans la partie opérative. Une action type est une action de calcul.
Dans une ligne de micro-instruction de calcul en langage de microcode on précisera

- Le registre `sourceA` (transmis par le bus A). C'est un des 16 registres. L'entrée gauche de l'UAL est MB ou ce registre source.
- Le registre `sourceB` (transmis par le bus B). C'est un des 16 registres. L'entrée gauche de l'UAL est ce registre source.
- L'opération UAL. Cela peut être une des 4 opérations `+, AND, XOR, entrée_gauche_inchangée`. Le AND et le XOR sont bit à bit.
- L'opération décaleur. Cela peut être une des 3 opérations `shl, shr, rien`. (décalages gauche ou droite de 1 position) ou pas de décalage.
- Le(s) registre(s) destinataire(s). Les destinataires possibles sont les 16 registres (y compris 0,1,ff mais ils restent inchangés) et MB. En plus le registre MA peut être destinataire de la valeur circulant sur le bus B.

On écrira ainsi une micro-instruction de calcul en langage de microcode :

```
reg.dest = fct_décaleur (sourceA opérationUAL sourceB)
```

ou

```
reg.dest = sourceA opérationUAL sourceB
```

ou

```
reg.dest = sourceA opérationUAL sourceB (majflag ou ema)
```

Les micro-instructions de calcul modifient les micro-flags μZ, μN, μC selon les résultats du calcul dans l'UAL. Ils sont mis à jour dans les flags Z N C connus du programmeur (dans le

registre F) grâce à la commande
`majflag` qui peut être ajoutée à une ligne de micro-instruction en langage microcode.

`ema` peut être ajouté à toute micro-instruction. Il y a alors chargement de MA par ce qui est sur le bus B.

Exemples de microinstructions valides (autres que celles du micro-programme)

destinataire	décaleur	sourceA	opérationUAL	sourceB	
pc =		mb	+	1	majflag
tir =	shl(	ir	xor	ff)	
mb =		ac	*commentaire*	*opération =*	*entrée gauche*
a =	shr(	b	+	ac)	
0 =		ir	and	1	majflag
ac =		mb			
ac =		ff	+	ac	ema

3.2.6 Micro-instructions de séquencement

Ce sont des sauts d'une micro-instruction à une autre.
j0,j1,...j7 : saut si (0, 1er, .., 7ème) bit de IR est à 1
jp : saut inconditionnel,
jz : saut si microZ = 1,
jn : saut si microN = 1,
jc : saut si microC = 1
ji : saut si bit I du registre F est à 1
jq : saut si interruption externe

syntaxe :
j* etiquette
.....
suite du microprogramme
.....
etiquette : suite du microprogramme

3.2.7 Accés à la mémoire

read et write permettent la lecture et l'écriture d'un mot en mémoire via les registres MB (donnée) et MA (adresse). Le temps d'accès de la mémoire est de deux périodes d'horloge du processeur (2 read) pour la lecture et de 3 cycles pour l'écriture (3 write).

3.2.8 Codage des micro-instructions par des commandes

Dans la réalisation de la partie contrôle, les micro-instructions sont stockées séquentiellement dans une mémoire morte de 512 mots de 38 bits. Le codage d'une ligne de micro-instruction se fait sur 38 bits. Pour décrire les 38 bits et leurs effets, on peut les grouper par "champs". Le premier bit code les deux types de micro-instructions.

3.2.9 Micro-instructions de calcul

Premier bit = **0** : C'est le 1er format possible, celui des micro-instructions de calcul. Le tableau donne les champs et nombre de bits. (Les deux champs ϕ sont inutilisés)

0	ϕ	dec	mx	mb	rd	wr	ma	si	ci	f	ual	c	sc	sb	sa	ϕ	BU	BC	BB	BA
1	4	2	1	1	1	1	1	1	1	1	2	1	4	4	4	4	1	1	1	1

Explication des commandes par catégories.
Commandes de sélection :
sc : Bus C (4 bits) : si commande non paramétrée, numéro du registre qui sera chargé. (cela peut être le numéro d'un registre constant, mais alors chargement non effectif)
sa : Bus A (4 bits) : en mode simple, numéro du registre qui est transmis sur le bus A.
sb : Bus B (4 bits) : en mode simple, numéro du registre qui est transmis sur le bus B.
mx : commande du Mux sur l'entrée g de l'UAL (1 bit) : 1 : transmet bus A ; 0 : transmet MB

Commandes de chargement :
ma : chargement de MA (1 bit) : 1 : MA <- ce qui est sur le B-bus ; 0 : MA inchangé
mb : chargement de MB (1 bit) : 1 : MB <- sortie du décaleur ; 0 : MB inchangé
C (1 bit) : Autorise/interdit le chargement de ce qui est sur le bus C dans le registre spécifié par le champ Bus C

Commandes d'échange avec la mémoire : (Les deux bits ne peuvent être simultanément à 1)
rd (1 bit) Read : 1 : lecture. MB <- Mem[MA]
wr (1 bit) Write : 1 : écriture. Mem[MA] <- MB

Commandes de calcul : ual (2 bits) :
00 : somme arithmétique des 2 entrées de l'UAL
01 : entrée de gauche
11 : XOR bit à bit des 2 entrées de l'UAL
10 : ET bit à bit des 2 entrées de l'UAL

dec : Décalage (2 bits) :
00 : pas de décalage
01 : sortie ual décalée à droite
10 : sortie ual décalée à gauche
11 : non utilisée

F (1 bit) : 1 : mettre à jour les flags du registre F (Z, N et C) par la valeur actuelle des microflags μC, μN, μZ.

3.2.10 Micro-instructions de séquencement

Premier bit = **1** : C'est le 2ème format possible, celui des micro-instructions de séquencement. Le tableau donne les champs et nombre de bits. (Le champs ϕ est inutilisé)

1	cond	ϕ	localisation de la micro-instruction suivante dans la mémoire morte de microprogramme.
1	4	25	8

codage sur 4 bits	signification de la condition
1000	Saut si bit 0 du registre IR =1
1001	Saut si bit 1 du registre IR =1
1111	Saut si bit 7 du registre IR =1
0000	Saut inconditionnel
0110	Saut si μZ=1
0101	Saut si μN=1
0100	Saut si μC=1
0111	Saut si bit I du registre F = 1
0001	Saut si requète d'interruption externe

Adresse suivante (8 bits) : adresse de la prochaine micro-instruction (si la condition de saut est vérifiée). Si la condition n'est pas vérifiée il y a passage à la micro-instruction suivante.

Remarque : Dans le cas d'un saut aucune opération et chargement de registre ne sont réalisés dans la PO (toutes les commandes de chargement de registre sont inhibées). Dans une vraie machine ce serait tout à fait possible de le faire (ex : `jn etiquette et pc = pc + 1`) mais ce n'est pas prévu dans le simulateur et le micro-assembleur de PROCESIM.

3.2.11 Primitives avancées pour l'écriture de micro-programmes

Ces primitives ne sont utilisés que en mode description de nouvelle machine.

labus désigne le registre dont le numéro est dans les 4 bits de poids forts de MK2
lbbus désigne le registre dont le numéro est dans les 4 bits de poids faibles de MK1
lcbus désigne le registre dont le numéro est dans les 4 bits de poids forts de MK1
lual désigne lopération à effectuer dans lUAL dont le code est dans les bits 1 et 2 de MK2.
Spéciales : seti et clri : mise a 1 et à 0 du bit I de F (sert au traitement des interruptions)
ji : saut si bit I du registre F est à 1
jq : saut si interruption externe

Commandes : Manipulation du bit d'autorisation de prise en compte des interruptions :
Si (1bit) : 1 : mettre à 1 le bit I du registre F
Ci (1bit) : 1 : mettre à 0 le bit I du registre F

microcode.muc

```
! Variante avec prise en compte des interruptions

!*********************************************************************!
!  init                                                               !
!  PC <- 0  // I <- 1                                                 !
!*********************************************************************!

init:       pc = 0  seti

!*********************************************************************!
!  fetch                                                              !
!  Rinstr_fort (ir)            <- Mem[pc] // pc ++                    !
!  Rinstr_faible (mb // mlk2)  <- Mem[pc] // pc ++                    !
!                                                                     !
! 1er  octet (poids fort : désigne opération) dans ir                 !
! 2ème octet oooo dddd dans mb et mk2                                 !
!                                                                     !
!*********************************************************************!

fetch:     pc = 0 +pc ema
           read
           read  pc = pc +1
           ir = mb

           pc = 0 +pc ema
           read
           read  pc = pc +1
           mk2 = mb

           j7   branch_autres      ! 1x : branchement ou autres instr

!*********************************************************************!
! Section commune à mémoire et calcul                                 !
!                                                                     !
! Mettre l'opérande reg_oooo ou #oooo dans mb                         !
!*********************************************************************!
            j4 oooo_imm

            ! reg_oooo dans mk2 poids forts : lire reg[MK2]
oooo_reg:   mb = labus
            jp  oooo_lu

            ! #oooo dans mk2 poids forts    : prendre mb = mk2>> 4
oooo_imm:   mk1 = shr(mk2)
            mk1 = shr(mk1)
            mk1 = shr(mk1)
            mb  = shr(mk1)

oooo_lu:   j6   memoire            ! 01 : load ou store
```

microcode.muc

```
!*********************************************************************!
! Gestion des instructions de calcul                                  !
!                                                                     !
! Format : code_op8  oooo rrrr                                        !
!                    o : opérande droit = ual_gauche (reg ou #4b)     !
!                    r : opérande gauche / résultat  ual droit        !
!                                                                     !
!          76 5 4       3  2  10                                      !
! Code_op8: 00 S # sansbar /+1 ual                                    !
!                        0  0  00  (+)     /o + d + 1  sub            !
!                        0  0  01  (g)     /o + 1      ---            !
!                        0  0  10  (&)     /o & d + 1  ---            !
!                        0  0  11  (^)     /o ^ d + 1  ---            !
!                        0  1  00  (+)     /o + d      ---            !
!                        0  1  01  (g)     /o          mvn            !
!                        0  1  10  (&)     /o & d      bic            !
!                        0  1  11  (^)     /o ^ d      xnor           !
!                        1  0  00  (+)      o + d + 1  ---            !
!                        1  0  01  (g)      o + 1      ---            !
!                        1  0  10  (&)      o & d + 1  ---            !
!                        1  0  11  (^)      o ^ d + 1  ---            !
!                        1  1  00  (+)      o + d      add            !
!                        1  1  01  (g)      o          mov            !
!                        1  1  10  (&)      o & d      and            !
!                        1  1  11  (^)      o ^ d      xor            !
!                                                                     !
!  Sélection opération :   mk2  (lual)                                !
!  Lecture   reg_oooo  :   MK2  (labus)                               !
!  Lecture   reg_dddd  :   mk1  (lbbus)                               !
!  Ecriture  reg_dddd  :   MK1  (lcbus)                               !
!*********************************************************************!


!1) Mettre xxxx dddd dans mk1  (pour mb = mb oper lbbus)
calcul:     mk1 = mk2

!2)  ne pas inverser si ir3 == 1
testbar:    j3  testplusun
            mb = mb xor ff

!3)  ne pas ajouter 1 si ir2 == 1
testplusun: j2 calculer
            mb = mb + 1

!4)  calculer mb = mb lual lbbus avec ou sans les flags
calculer:   mk2 = ir
            j5  avecs

sanss:      mb = mb lual lbbus
            jp ranger

avecs:      mb = mb lual lbbus majflag

!5)  lcbus = mb   : mettre dddd dans mk1 poids forts (<<4)

ranger:     mk1 = shl (mk1 + mk1)
            mk1 = shl (mk1 + mk1)
            lcbus = mb
            jp fetch
```

microcode.muc

```
!****************************************************************!
! 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: 10 E #  dest                                         !
!                                                                !
!  Lecture   oooo  :   déjà fait, dans mb                        !
!  Lecture   reg_gggg  :   mk1  (lbbus)                          !
!  Ecriture  reg_dest  :   MK1  (lcbus)                          !
!****************************************************************!
memoire:

            !  a completer ....
            !
            jp fetch



branch_autres: j6  autres

!****************************************************************!
! Gestion de b/bl cond                                           !
!                        cond parmi eq (Z==1)                    !
!                                  hs (C==1)                     !
!                                  __ (toujours)                 !
!                                                                !
! Format : code_op8  deplacement_8_en_octets                     !
!                                                                !
!           76 5 4  3210                                         !
! Code_op8: 00 /l z C xxx                                        !
!              0 0 0 xxx  bl                                     !
!              1 0 0 xxx  b                                      !
!              0 1 x xxx  bleq                                   !
!              1 1 x xxx  beq                                    !
!              0 0 1 xxx  blcs                                   !
!              1 0 1 xxx  bcs                                    !
!****************************************************************!

!1) Calculer adresse de destination dans mb (deplacement dans mk2)
             mb = pc + mk2

!2) Copier adresse de retour dans mk1
             mk1 = pc

!3) tester condition de branchement

            j3   beq
            j2   bcs
            jp   brancher        ! bal : inconditionnel !

            ! amener flag à tester en bit 0
            ! C : bit 2
bcs:        mk2 = shr (f)
            mk2 = shr (mk2)
            jp test_flag
```

microcode.muc

```
            ! Z : bit 0
beq:        mk2 = f

test_flag:  mk2 = mk2 and 1
            jz   fetch

brancher:   pc = mb
            j5   fetch           ! pas bl : terminé     !
            tir = mk1            ! bl : sauver --> lr/tir !
            jp   fetch

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

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:    j3  set

clear:       clri
             jp fetch

set:         seti
             jp fetch
```