Problème 1. Processeur HoMade (13 Pts)

Vous allez utiliser le processeur HoMade (quelle surprise !!) et son assembleur HasmV5. Un sous ensemble de la grammaire HasmV5 est en annexe.

Question 1 (5 pts)

Construire deux fonctions HasmV5.

- La première RDM_INIT range x"80000000" dans le registre 1 de l'IP registre. (voir les IPs >1 et <1 dans l'annexe)
- La seconde fonction RDM_NEXT produit sur la pile, à chaque appel, le nombre suivant dans la même suite aléatoire que celle utilisée par l'IP rdm du TP : le décalage à droite de la valeur du registre est complétée sur le bit 31 , on obtient r(31:0) <= (((r(0) xnor r(2)) xnor r(3)) xnor r(4)) concaténé à r(31:1). Il faut bien sur conserver cette nouvelle valeur dans le registre 1 pour le prochain appel. (Xnor = Xor puis invert)

Ecrire un programme main qui calcule et affiche le dixième nombre aléatoire de la suite.

Question 2 (3pts)

Ecrire un programme en boucle infinie qui produit sur le sommet de pile le $I^{ième}$ nombre aléatoire, I étant la valeur présente sur les switches de la carte avec I>1. A chaque pression de bouton on relance un nouveau calcul avec la valeur présente sur les switches, si le bouton 0 on utilisera un calcul soft par la fonction de la question 1 (ne pas oublier de réinitialiser le registre 1), si le bouton 1 est appuyé on utilise l'IP hard rdm, pour les autres boutons on utilisera la dernière configuration utilisée soit soit hard.

Question 3 (3pts)

La notion de DYNAMIC, utilisée pour le calcul de Fibo en TP, est également présente sur les esclaves. Vous allez écrire un programme simple qui utilise **4** esclaves en anneau [X=4, Y=1], chacun produit les nombres aléatoires de la même série.

Le programme est une boucle infinie qui à chaque itération, saisit la valeur I depuis les switches, l'envoie (broadcast) à tous les esclaves de l'anneau et génère sur la pile de chaque processeur esclave le $(I + Xnum)^{ième}$ nombre aléatoire en utilisant soit l'IP rdm soit la fonction de la question 1 suivant le bouton appuyé comme pour la question 2. (Xnum est le numéro de l'esclave sur la dimension X). Quand tous les esclaves ont fini, on vide la pile des esclaves et on recommence la boucle. **Vous pouvez réduire la difficulté en exécutant toujours l'IP rdm quel que soit le bouton appuyé (le nombre de point pour cette question sera réduit lui aussi, signalez-le sur la copie !!!)**

Question 4 (2pts)

Montrez que l'on peut recouvrir le calcul de la série aléatoire sur les esclaves et la communication de la prochaine valeur des switches à prendre en compte lors du passage à l'itération suivante de la boucle infinie. Modifiez le programme.

Exercice 2 Mémoire cache (4 Pts)

Soit le cache suivant en direct mapping: taille 1K mots, Bloc de 4 mots, Mots de 32 bits et adresses du CPU sur 16 bits.

1. Donnez le nombre et la taille des champs de l'adresse ?
2. Pour le programme suivant construire la trace chronologique des adresses envoyées par le processeur en lecture et écriture sur le tableau A, on suppose que A est rangé à l'adresse x0000. . La variable I reste dans un registre. Calculez le nombre de défaut de cache pour cette trace sur ce cache. Estimez le gain par rapport à une mémoire sans cache.

```
Int A[512], i ;
for (i = 0; i < 256; i++) {
   A[i+256] = A[i] + i ;
}
```

3. Montrez que l'on peut améliorer les performances du cache en tirant partie du principe de localité spatiale, déroulez la boucle dans cet esprit, vous pouvez utiliser des registres R0, R1, R2 et R3 pour les variables temporaires. Recalculez le nombre de défaut de cache. Quelle est la taille minimale du cache qui permettrait d'obtenir le même nombre de défaut avec cette programmation. Quel gain est relatif au principe de localité temporelle.
4. A taille égale du cache, proposez une autre organisation/mapping matérielle du cache pour améliorer ce taux sans avoir à transformer le programme initial. Comparez le nombre de défaut de cache avec la solution de la question 3.

Exercice 3 Question de cours (3pts)

Les algorithmes de remplacements sont indispensables dans un cache lorsque le mapping est associatif. On utilise ici un cache set associatif de taille 4 blocs. Il faut donc choisir parmi les 4 celui que l'on écrase !!

1. Proposez une implantation matérielle de la méthode LRU, comment choisir celui qu'on écrase, quel sera le surcout pour chaque ensemble en nombre de bit ?
2. Proposez une méthode FIFO, quelle sera le surcout de votre méthode ?
3. Proposez une méthode aléatoire, quel surcout ?

<program HoMade> ::=
[IPdeclare_list]
(SLAVE
[PCdeclare_list][VCdeclare_list]
[WORDdeclare_list]
[STATIC_PC_VC_list]
START
[WORD_list]
MASTER) ! PROGRAM -- PROGRAM est utilisé pour la programmation d'un seul processeur le code SLAVE est vide
[VCdeclare_list]
[WORDdeclare_list]
[STATIC_VC_list]
START
[WORD_list]
ENDPROGRAM
<IPdeclare> ::=
IP ! 3IP

```
<IP> ::= -- pas utile ici
<Hexa> ::=
 0 ! 1 ! 2 ! 3 ! 4 ! 5 ! 6 ! 7 ! 8 ! 9 ! A ! B ! C ! D ! E ! F ! a ! b ! c ! d ! e ! f
<digit> ::=
0 ! 1 ! 2 ! 3 ! 4 ! 5 ! 6 ! 7 ! 8 ! 9
<bit> : :=
0 ! 1
<3IP> : := -- déclaration d'un bloc de 3 IPs, utilisé pour le STATIC et DYNAMIC
:3IP 3IP_name
   { IP_available }3 !
   [ IP_available ]2 ( RETURN ! HLT ) !
   IP_available ( RETURN ! HLT ) NULL ! ( RETURN ! HLT ) NULL NULL  -- RETURN pour VC et HLT pour PC
;
<IP_available> ::=
IP_name ! IP_name_predefined
<PCdeclare> ::= -- déclaration du Parallel Component , permet de déclarer du calcul SPMD sur les esclaves
PC PC_name
<VCdeclare> ::= -- déclaration d'un Virtual Component, utiliser pour la programmation réflexive
VC VC_name
<WORDdeclare> ::=
: WORD_name
[ Homade_code ]
;
<STATIC_PC_VC> ::= -- affectation à la compilation d'un IP soft ou au plus 3 IP hard à un VC ou un PC
' ( WORD_name ! 3IP_name ) (PC_name ! VC_name ) STATIC
<STATIC_VC> ::=
' ( WORD_name ! 3IP_name ) VC_name  STATIC
<WORD_list> : :=
{ WORD_name }*
<Homade_code> ::=
Literal !
Dynamic_spec !
IP_name !
WORD_name
VC_call !
PC_call !          -- Sur le maître exclusivement
WAIT !             -- Sur le maître exclusivement  !!!!!!  //x wait  équivaut //w
Contition !
Boucle !
<Literal> ::=
$  hexa { hexa }<8 !      -- tout est en hexa , en complément à 2  ex : $ FE12A
$ & [ - ] digit { digit }<10 !  -- codé en décimal signé , mot de 32 bits max  ex $ &12545 ou $ &-1254
$ % bit { bit } <32   -- codé en binaire , les poids forts non spécifiés sont égaux à 0 ex $ %0111001$
<Dynamic_spec> ::= -- affectation dynamique d'un IP soft ou Hard
['] ( WORD_name ! 3IP_name ) (PC_name ! VC_name ) DYNAMIC   -- PC_name seulement dans  les WORD des
esclaves, le WORD_name et VC_name doivent être déclarés dans la même partie
 <VC_call> ;:=
VC_name /x
<PC_call> ::=
PC_name //x  ! -- exécution en parallèle sur les actifs
PC_name //w   -- exécution en parallèle sur les actifs avec barrière de synchronisation  en fin d'exécution
<Condition> ::=
IF  -- test sommet de pile, IF consomme le sommet de pile
[ Homade_code ]
[ ELSE
```

[Homade_code]
]
ENDIF
<Boucle> ::=
(
BEGIN
[Homade_code]
(AGAIN ! UNTIL) -- UNTIL consomme sommet de pile, répète le code si sommet est VRAI, AGAIN boucle infinie
) ! (
FOR -- sommet de pile nombre d'itération + (ex 4 FOR exécute 4 3 2 1 0 , dépile de 1)
[Homade_code]
NEXT)
<IP_name_predefined> ::=
+ ! -- a b +
- ! -- a b –
= ! < ! > ! 0= ! 0< ! <= ! >= ! <> !
1+ ! 1- ! 2/ !
negate ! -- comp2
led !
leddup !
switch !
pop1 ! -- idem que drop : dépile
pop2 ! pop3 ! – dépile 2 ou 3 d'un coup
7seg ! 7segdup !
nop !
btn ! btnpush !
rdm ! -- appel IP rdm hard
dup ! -- (a -- a a) : duplique le sommet de la pile
swap ! -- (a b -- b a) : permutation du sommet et du sous_sommet
drop !
tuck ! -- (a b -- b a b) : recopie et insère le sommet sous le sous-sommet
over ! -- (a b -- a b a) : recopie et empile le sous-sommet
rot ! -- (a b c -- b c a) : Rotation des trois valeurs en tête de pile dans le sens des aiguilles d'une montre
-rot ! -- (a b c -- c a b) : Rotation des trois valeurs en tête de pile dans le sens inverse des aiguilles d'une montre
nip ! -- (a b -- b) : supprime le sous-sommet
nip3 ! -- (a b c - b c) supprime le sous sous sommet
nip23 ! -- (a b c - c) supprime les deux sous et sous sous sommets
dup2 ! -- (b c - b c b c) duplique les deux sommets
dup3 ! -- (a b c - a b c a b c) duplique les trois sommets
and !
or !
xor ! -- Xor (a b - a xor b) 32 bits
-> ! rshift ! -- decalage : (a b - SHR(a, b)) rigth shift a de b bits (0..31)
<- ! lshift !
M2S ! -- top de la pile du master vers le buffer du premier slave
S2M ! -- et l'inverse
bcomx ! b>x ! -- communication sur l'axe des x, les buffers des esclaves avec x= 0 ne changent pas leur valeur
bcomy ! b>y !
comx ! >x ! -- communication en anneau sur l'axe des x des buffers de chaque esclave
comy ! >y !
get ! put ! – transfert entre esclave et buffer de communication
xnum ! – numéro de l'esclave sur la dimension X
0 ! >1 ! >2 ! >3 ! >4 ! >5 ! >6 ! >7 ! -- rangement du sommet de pile sur registre i
0> ! 1> ! 2> ! 3> ! 4> ! 5> ! 6> ! 7> ! -- copie registre i sur la pile