Numéro de copie :

LIF6 – CC-TD-F printemps 2011

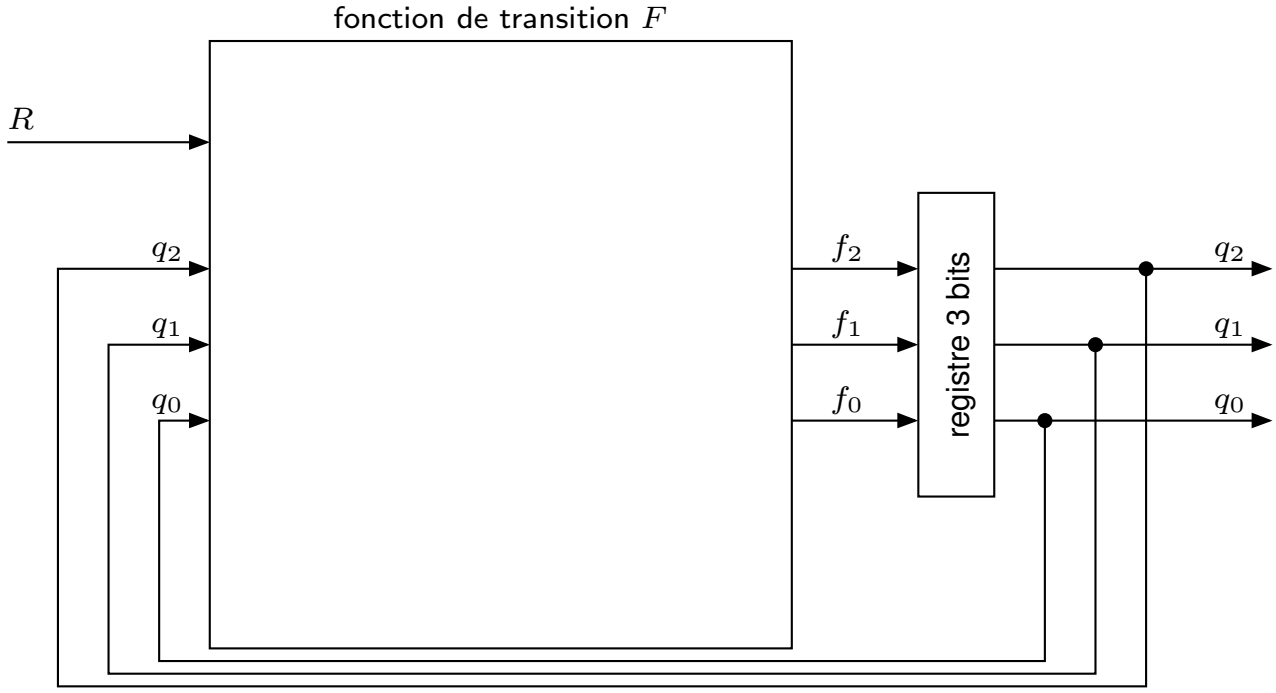
20 juin 2011, 13h en Thémis 70, durée 1h30 – Aucun document autorisé, calculatrices, téléphones et autres sont interdits – Le sujet est à rendre complété : n'oubliez pas d'y inscrire votre numéro de copie.

I Compteur séquentiel 3 bits (10 pts)

On veut mettre au point un compteur séquentiel 3 bits : on va modéliser ce compteur par un automate fini séquentiel, puis mettre au point un circuit séquentiel pour l'implanter. Le compteur présente :

- trois sorties binaire q_2, q_1 et q_0, qui permettent d'obtenir la valeur courante $(q_2, q_1, q_0)_2$ du compteur ;
- une entrée R (comme reset), pour remettre à 0 la valeur du compteur.

A chaque top de l'horloge, soit $R = 0$ et la valeur du compteur est incrémentée, soit $R = 1$ et la valeur du compteur repasse à 0. L'état courant du compteur est (q_2, q_1, q_0), et l'état suivant (f_2, f_1, f_0) est déterminé par une fonction de transition F telle que $(f_2, f_1, f_0) = F(q_2, q_1, q_0, R)$. L'état initial du compteur est $(0, 0, 0)$.



Q.I.1) - (3 pts) Donnez une représentation graphique de l'automate fini séquentiel correspondant à notre compteur : chaque état (q_2, q_1, q_0) est représenté par un cercle qui contient le triplet (q_2, q_1, q_0) ; chacune transition possible est représentée par une fléche, portant soit l'étiquette $R = 0$ soit l'étiquette $R = 1$.

Q.I.2) - (2 pts) Complétez le tableau suivant, de manière à ce qu'il représente la fonction de transition F :

			$R=0$			$R=1$		
q_2	q_1	q_0	f_2	f_1	f_0	f_2	f_1	f_0
0	0	0						
0	0	1						
0	1	0						
0	1	1						
1	0	0						
1	0	1						
1	1	0						
1	1	1						

Q.I.3) - (3 pts) On conserve la notation $(f_2, f_1, f_0) = F(q_2, q_1, q_0, R)$ pour exprimer le fait de passer de l'état courant (q_2, q_1, q_0) à l'état suivant (f_2, f_1, f_0). Montrez proprement que l'on a les expressions suivantes, qui expriment la fonction transition F par des formules booléennes (rappelons que $a \oplus b = \overline{a}\, b + a\, \overline{b}$) :

$$f_0 = \overline{q_0}\, \overline{R}, \quad f_1 = (q_1 \oplus q_0)\, \overline{R}, \quad f_2 = \big(q_2 \oplus (q_1\, q_0)\big)\, \overline{R}.$$

Q.I.4) - (2 pts) Dans le schéma du circuit séquentiel, complétez la fonction de transition F, de manière à obtenir le compteur 3 bits demandé.

II Programmation en assembleur du LC3

Pour les deux exercices suivants, il ne sera porté qu'une attention particulièrement faible au code mal commenté, et les commentaires erronés seront pénalisés.

II.1 Somme des éléments d'un tableau (5 pts)

Le but est d'écrire en langage d'assemblage du LC3 une routine permettant de sommer les éléments d'un tableau d'entiers. La routine `somme` doit permettre de sommer le contenu des cases dont les adresses sont comprises entre une adresse de début de tableau placée dans R1, et une adresse de fin de tableau placée dans R2. Le résultat de l'accumulation sera retourné dans R0.

Le programme principal effectue un appel à `somme` pour sommer les entiers entre les adresses `debut` et `fin`, puis stocke le résultat à l'adresse `resultat`.

Q.II.1) - Complétez le code de la routine `somme` :

```
            .ORIG x3000     ; adresse de début de programme
; partie dédiée au code
            LEA R1,debut    ; charge l'adresse du premier élément dans R1
            LEA R2,fin      ; charge l'adresse du dernier élément dans R2
            JSR somme       ; appel la routine somme
            ST R0,resultat  ; stocke le résultat
            HALT            ; termine le programme
; partie dédiée au résultat et aux données
resultat:   .BLKW #1        ; espace pour stocker le résultat
debut:      .FILL #12
            .FILL #45
            .FILL #16
            .FILL #06
fin:        .FILL #78
; Routine somme. On suppose que R1 <= R2.
; paramétres d'entrée : R1 et R2 (peuvent être modifiés par l'appel)
; paramétre de sortie : R0       (la somme calculée)
; registre temporaire : R3       (pour des calculs intermédiaires)
somme:




            .END
```

Q.II.2) - Après l'exécution du programme, quelle sera la valeur stockée à l'adresse `resultat` ? Donnez votre résultat en hexadécimal.

II.2 Division entière (5 pts)

Le but est d'écrire en langage d'assemblage du LC3 une routine permettant de calculer la division euclidienne d'un entier naturel a par un entier naturel $b > 0$. La routine `division` doit permettre de calculer deux entiers naturels q et r tels que $a = bq + r$, avec $0 \leq r < b$. Inspirez-vous de la fonction C++ suivante, qui permet d'implanter les choses simplement :

```
void division(unsigned int &q, unsigned int &r, unsigned int a, unsigned int b) {
  q=0; r=a;
  do {
    q=q+1;
    r=r-b;
  } while(r>0);
  if(r<0) {
    q=q-1;
    r=r+b;
  }
}
```

Q.II.3) - Complétez le code de la routine `division`.

```
        .ORIG x2000
    ; programme principal
              LD R2,a         ; on charge le dividende dans R2
              LD R3,b         ; on charge le diviseur dans R3
              JSR division    ; appel à la routine de division
              ST R0,q         ; on stocke le quotient
              ST R1,r         ; on stocke le reste
              HALT            ; fin du programme (TRAP x25)
    ; Données
    a:        .FILL #157      ; dividende a
    b:        .FILL #2        ; diviseur  b
    q:        .FILL #0        ; quotient q
    r:        .FILL #0        ; reste r
    ; Fonction de division.
    ; paramétres d'entrée  : a(R2) le dividende, b(R3) le diviseur
    ;                        on suppose a >=0, b > 0
    ; paramétres de sortie : q(R0) le quotient,  r(R1) le reste
    ; registre temporaire  : R4
    division:
```

```
              .END
```

Q.II.4) - Après l'exécution du programme, quelles seront les valeurs stockées aux adresses `q` et `r` ? Donnez votre résultat en hexadécimal.

Récapitulatif des instructions du LC3 :

syntaxe	**action**	**nzp**
NOT DR,SR	DR <- not SR	*
ADD DR,SR1,SR2	DR <- SR1 + SR2	*
ADD DR,SR1,Imm5	DR <- SR1 + SEXT(Imm5)	*
AND DR,SR1,SR2	DR <- SR1 and SR2	*
AND DR,SR1,Imm5	DR <- SR1 and SEXT(Imm5)	*
LEA DR,label	DR <- PC + SEXT(PCoffset9)	*
LD DR,label	DR <- mem[PC + SEXT(PCoffset9)]	*
ST SR,label	mem[PC + SEXT(PCoffset9)] <- SR	
LDR DR,BaseR,Offset6	DR <- mem[BaseR + SEXT(Offset6)]	*
STR SR,BaseR,Offset6	mem[BaseR + SEXT(Offset6)] <- SR	
BR[n][z][p] label	Si (cond) PC <- PC + SEXT(PCoffset9)	
RET (JMP R7)	PC <- R7	
JSR label	R7 <- PC ; PC <- PC + SEXT(PCoffset11)	