Durée 2h Documents autorisés.

Sujet d'examen 1 Architecture des ordinateurs Licence S3

Architecture des ordinateurs

Les seuls documents autorisés sont les fascicules de cours, les énoncés de TD et les notes manuscrites de TD.

1 Microprogrammation

On considère le microprocesseur dont l'architecture est présentée Figure 1 page 2. Cette architecture est commentée dans la section 1.1 et correspond à celle étudiée en travaux dirigés.

Questions : écrire la séquence de signaux (micro-instructions) réalisant les instructions assembleurs suivantes :

1. `SUB R0,R1` ;
2. `INC RX0` (incrémente l'octet en mémoire se trouvant à l'adresse contenu dans la combinaison des 2 registres R0 — poids faible — et R1 — poids fort).

1.1 Présentation de l'architecture utilisée

Le microprocesseur de la Figure 1 page 2 est composé des éléments suivants :

– **un bloc registres 8 bits** : nommés de $R0$ à $R7$, il s'agit de registres généraux 8 bits. Pour manipuler ce bloc, nous disposons d'une commande de sélection $SR0..2$ (*Select Register*) permettant de sélectionner l'un des huit registres comme illustré Tableau 1. Ceci se traduit par l'activation d'un unique CS (Chip Selector) de l'un des huit registres. Le signal Rin provoque le chargement dans le registre sélectionné de l'information disponible sur le bus. Le signal $Rout$ provoque la mise sur le bus de l'information contenue dans le registre sélectionné.

SR2..0	Registre sélectionné	SR2..0	Registre sélectionné
000	R0	100	R4
001	R1	101	R5
010	R2	110	R6
011	R3	111	R7

TAB. 1 – Sélection dans le bloc registres

– **Un bloc ALU** : il est constitué d'une unité arithmétique et logique possédant un certain nombre de commandes $cmd0..N$ dont la taille est à spécifier. Un certain nombre de Flags (à spécifier) sont stockés dans un registre de flags appelé F. Les entrées de l'ALU sont les sorties de deux registres 8 bits X et Y. Ces registres peuvent charger l'information présente sur le bus de données par les signaux Xin et Yin. La sortie de l'ALU est relié au bus de données et est activées à l'aide du signal $ALUout$.
Notons que les registres X et Y sont des registres internes et que le programmeur n'a pas accès à ces registres.

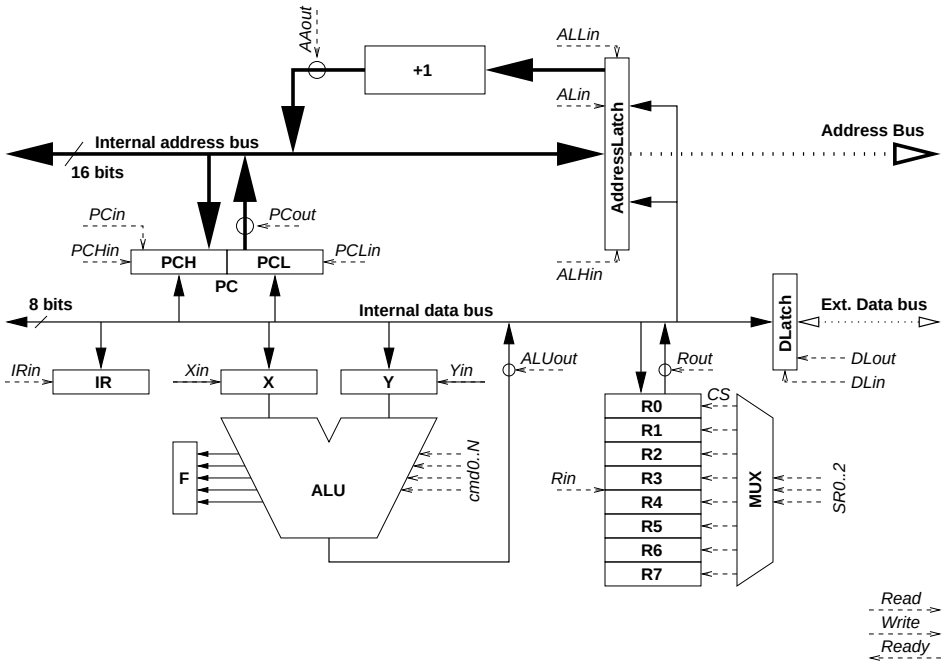

FIG. 1 – Architecture utilisée

– **Un bloc gestion des adresses** : comprenant un bus interne d'adresse 16 bits, un registre *AddressLatch* 16 bits connecté au bus d'adresse externe, un registre PC (Programm Counter) 16 bits et un incrémenteur.
Les signaux associés au registre *AddressLatch* sont au nombre de trois : $ALin$ qui permet de charger dans le registre l'information sur 16 bits présente sur le bus d'adresse interne, $ALLin$ qui permet de charger dans la partie basse du registre l'information sur 8 bits présente sur le bus de données interne et $ALHin$ qui permet de charger dans la partie haute du registre l'information sur 8 bits présente sur le bus de données interne.
Les signaux associés au registre PC sont au nombre de quatre et sont similaires à ceux pour *AddressLatch* : $PCin$ qui permet de charger dans le registre l'information sur 16 bits présente sur le bus d'adresse interne, $PCLin$ qui permet de charger dans la partie basse du registre l'information sur 8 bits présente sur le bus de données interne, $PCHin$ qui permet de charger dans la partie haute du registre l'information sur 8 bits présente sur le bus de données interne et enfin le signal $PCout$ qui met sur le bus d'adresse interne l'information contenue dans PC.
L'incrémenteur prend en entrée le contenu du registre *AddressLatch* et présente en sortie l'adresse suivante (incrémentation de 1). La sortie de l'incrémenteur est mise sur le bus d'adresse interne par le biais du signal $AAout$.
– **un bloc de contrle** : constitué d'un registre d'instruction IR 8 bits et d'une unité de contrôle. L'unité de contrôle est chargée d'engendrer les signaux nécessaire au fonctionnement du microprocesseur en exécutant le cycle :

1. lire l'instruction ;
2. décoder l'instruction ;
3. exécuter l'instruction ;
4. préparer l'instruction suivante.

Dans le reste de l'exercice, on ignorera la phase 4. Le registre IR copie l'information positionnée sur le bus de données interne suivant le signal $IRin$.

– **un bloc gestion de la mémoire** : il comprend un registre de données *DLatch* (pour *Data Latch*) 8 bits qui copie la valeur du bus de données interne sur le signal *DLin*. Sur un signal *Read*, la mémoire met dans le registre *DLatch* la valeur de la case mémoire d'adresse *AddressLatch*. Sur un signal $Write$, la mémoire copie dans la case mémoire d'adresse *AddressLatch* l'information qui est dans *DLatch*. Lorsque la mémoire a terminé l'opération demandée (*Read* ou $Write$), elle positionne en retour le signal *Ready*.

2 Programmation assembleur

On dispose du jeu d'instructions assembleur décrit à la fin de l'exercice.

Le but de l'exercice est d'écrire un programme assembleur effectuant la division euclidienne en utilisant ce jeu d'instructions (remarquez bien que ce dernier ne contient aucune instruction de division).

On impose que :
– les deux opérandes sont strictement positifs et inférieurs à 2^8 ;
– la mémoire contient le diviseur suivit du divisé à une adresse associée à l'étiquette `inputs` ;
– à la fin de l'exécution de votre programme, le registre :
  – `R0` contiendra le numérateur,
  – `R1` contiendra le dénominateur,
  – `R2` contiendra le reste,
  – `R3` contiendra le quotient,

  de la division euclidienne.

Vous êtes libre d'utiliser de l'espace mémoire à condition de le faire correctement (en lui associant une étiquette et en utilisant les instructions assembleur ad-hoc).

Méthode utilisée. Nous allons suivre l'algorithme suivant :

1. initialiser le quotient à 0
2. initialiser i à 8
3. décrémenter i
4. si i est nul alors faire l'étape 12
5. décaler les bits du reste et du quotient de 1 vers les poids fort
6. transférer le bit de poids fort du divisé dans le bit de poids faible du reste
7. décaler les bits du divisé de 1 vers les poids fort
8. si le reste est plus petit que le diviseur alors recommencer l'étape 3
9. soustraire le diviseur au reste
10. incrémenter le bit de poids faible du quotient
11. aller à l'étape 3
12. fin du calcul

Il s'agit de l'algorithme de division classique appliqué à des nombres binaires.

Question.

1. Écrire un bout de code qui étant deux nombres stockés dans les registres `R0` et `R2`, décale les bits de `R2` de 1 vers les poids forts, transfère le bit de poids fort de `R0` dans le bit de poids faible de `R2` et décale `R0` de 1 bit vers les poids forts.

   Indications : à quelle opération arithmétique correspond un tel décalage ? Comment repérer la non-nullité du bit de poids fort d'un registre avec une opération arithmétique et un saut ?
2. Écrire le code complet de la division

Jeu d'instructions. On dispose de registres 8 bits généraux nommés de $R0$ à $R7$. On dénomme `RX0` le registre 16 bits obtenu par la concaténation de `R1` et `R0` (`R1` octet de poids fort, `RX1` par `R2` et `R3`, `RX2` par `R4` et `R5` et `RX3` par `R6` et `R7`.

Notre microprocesseur dispose des instructions suivantes :

- `ADD Rm, Rn` — (*pour ADD :-)*) qui additionne la valeur du registre `Rn` avec la valeur du registre `Rm` et stocke le résultat dans le registre `Rn` ;
- `AND Rm, Rn` — qui fait la conjonction bit à bit des deux registres et range le résultat dans `Rn` ;
- `DEC Rn` — (*pour DECrement*) qui soustrait 1 à la valeur de `Rn` et stocke le résultat dans `Rn` ;
- `INC Rn` — (*pour INCrement*) qui ajoute 1 à la valeur de `Rn` et stocke le résultat dans `Rn` ;
- `JC label` — (*pour Jump if Carry*) qui effectue un saut à l'adresse 16 bits spécifiée par l'étiquette `label` si l'opération précédente a engendrée une retenue (ou si le résultat est négatif) ;
- `JMP label` — (*pour JuMP*) qui effectue un branchement à l'adresse (sur 16 bits) spécifiée par l'étiquette `label` ;
- `JZ label` — (*pour Jump if Zero*) qui effectue un saut à l'adresse 16 bits spécifiée par l'étiquette `label` si l'opération précédente a donné un résultat nul ;
- `LD RXn, R0` — (*pour LoaD*) qui charge dans le registre `R0` la valeur stockée en mémoire à l'adresse (sur 16 bits) contenue dans le registre `RXn` ;
- `MV arg,Rn` — (*pour MoVe*) qui charge dans le registre `Rn` la valeur de l'argument "`arg`". L'argument est soit une valeur immédiate, soit un registre ;
- `NOT Rn` — qui inverse bit à bit le registre `Rn` et qui y stocke le résultat ;
- `ST R0, RXn` — (*pour STore*) qui stocke la valeur du registre `R0` en mémoire à l'adresse (sur 16 bits) contenue dans le registre `RXn` ;
- `SUB Rm, Rn` — (*pour SUBstract*) qui soustrait la valeur du registre `Rm` à la valeur du registre `Rn` et stocke le résultat dans le registre `Rn` ;
- `SWP Rn, Rm` — (*pour SWaP*) qui permute les registres `Rn` et `Rm`.

3 Circuits logiques et séquentiels

3.1 Formes normales

On se donne la table de vérité suivante :

a	b	c	res
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

1. Donnez une forme normale associée à cette table.
2. Dessinez un circuit logique associé à cette table.

3.2 Circuit

On se propose de décrire une unité à 4 bits d'entrée et $4 + 2$ bits de sortie pouvant réalisant 2 opérations f et g implantées par 2 circuits de même nom que l'on ne demande pas de décrire.

L'action de cette unité est controlée par une ligne de commande c qui permet d'indiquer ce que l'unité doit retourner (la sortie du circuit f ou celle du circuit g). En plus d'un résultat, cette unité produit en sortie 2 lignes correspondant à un drapeaux $d_1 d_0$.

L'exercice consiste à compléter le schéma de l'annexe 1 qui n'indique que les entrées et les sorties de l'unité. On cherche donc à donner le circuit qui relie les entrées aux sorties (en utilisant des bascules et les circuits décrits dans l'annexe 1).

Notre unité fonctionne de la manière suivante :

- si la ligne c est à 0, l'unité produit le résultat de la fonction g. La ligne d_0 (resp. d_1) est mise à 1 (resp. 0) si le résultat de g est 0 et à 0 (resp. 1) sinon ;
- sinon l'unité produit le résultat de la fonction f et le drapeaux $d_1 d_0$ conserve sa valeur précédente.

Question. Donnez le schéma de l'unité arithmétique décrite dans cet exercice (vous pouvez utiliser des portes logiques élémentaires, des bascules vues en cours et en travaux dirigés et un décodeur ; les fonctions f et g seront décrites par des boitiers).

Annexe 1**.**