

Examen d'Architecture des Ordinateurs

Majeure 1 – École Polytechnique

2007–2008

- L'examen dure 3 heures.
- Le sujet comporte 8 pages dont des rappels sur le LC-2.
- Tous documents autorisés.
- Le barème est donné à titre indicatif, il sert surtout à évaluer le poids respectif des sections.
- L'examen contient 2 exercices distincts que vous devez rendre sur **2 COPIES SÉPARÉES** (parce qu'ils seront corrigés par 2 correcteurs différents).
- Il est impératif de commenter vos programmes; la plupart des lignes/instructions doivent être suivies d'un commentaire permettant de comprendre leur rôle.

Exercice 1 - Accélérateur matériel (10 points)

Le parallélisme est présenté comme une voie majeure de dimensionnement des architectures, mais il existe en fait une seconde voie, orthogonale : la spécialisation des architectures. Le principe est d'ajouter dans le processeur des composants spécialisés ou semi-spécialisés qui permettent d'exécuter une séquence d'opérations à l'intérieur d'un circuit, plutôt que comme une séquence classique d'instructions. Dans un circuit, une séquence d'opérations correspond en fait au passage à travers plusieurs portes combinatoires, et s'exécute donc beaucoup plus rapidement que lorsqu'il faut passer par les phases de chargement, décodage, exécution et écriture du résultat de chaque instruction. En outre, l'exécution directe dans un circuit consomme moins d'énergie.

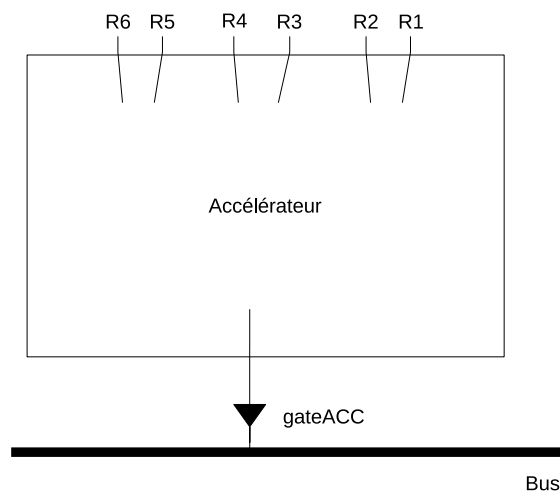


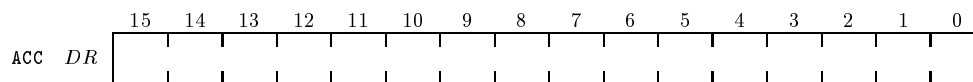
FIG. 1 – Accélérateur matériel dans le LC-2.

On veut étudier l'ajout d'un tel circuit, également appelé accélérateur matériel, dans le LC-2. Un accélérateur peut être conçu pour implémenter une seule séquence d'opérations, ou plusieurs; nous étudierons les deux cas. L'accélérateur étudié comporte 6 entrées, qui correspondent toujours aux registres R_1 à R_6 ; pour ce faire, un troisième port de sortie a été ajouté à tous les registres, et cette troisième sortie des registres R_1 à R_6 est donc directement connectée aux 6 entrées de l'accélérateur, voir Figure 1.

L'accélérateur comporte également une sortie sur 16 bits directement reliée au bus par un tristate appelé *gateACC*, voir Figure 1. Pour construire cet accélérateur, vous pourrez utiliser des additionneurs (2 entrées de 16 bits, une sortie de 16 bits et une retenue), des multiplexeurs (2 entrées de 16 bits, une sortie de 16 bits) ; si vous le jugez nécessaire, vous pourrez également ajouter quelques portes élémentaires (AND, OR, NOT, etc). Pour chaque schéma de circuit, il vous est demandé de préciser la nature des opérateurs utilisés, les connexions entre opérateurs et le nombre de bits portés par chaque fil. Autant que possible, on essaiera de minimiser d'abord le temps d'exécution, et ensuite le coût (nombre d'opérateurs) du circuit ; mais, il ne s'agit pas de trouver un temps et un coût optimaux, simplement de construire des circuits raisonnablement efficaces.

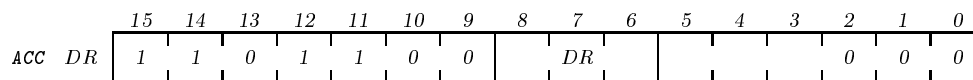
Question 1.1

Pour utiliser cet accélérateur, il nous faut une instruction *ACC* ; en outre, cette instruction doit spécifier un registre de destination *DR*. On rappelle que le circuit de contrôle du LC-2 ne peut tester/décoder que les bits I_{15} à I_{11} d'une instruction et le bit *BEN*. Pour ajouter l'instruction de l'accélérateur, on veut bien modifier le circuit de contrôle, mais pas augmenter le nombre de bits d'entrée du circuit de contrôle. Comment spécifier une nouvelle instruction pour l'accélérateur tout en conservant toutes les instructions existantes ? Plusieurs solutions sont possibles. Indiquer le format précis de votre instruction (valeur ou champ correspondant à chacun des 16 bits de l'instruction).



Réponse

On utilise l'opcode de l'instruction *RET* car les 12 bits de cette instruction ne sont pas utilisés, notamment le bit I_{11} que le circuit de contrôle peut utiliser pour distinguer *ACC* ($I_{11} = 1$) de *RET* ($I_{11} = 0$). On utilise les bits I_8 à I_6 pour *DR*.



Question 1.2

On veut que le circuit soit capable de réaliser l'opération $R_1 + R_2 + R_3 + R_4 + R_5 + R_6$. On supposera que le résultat tient toujours sur 16 bits. Donner le schéma détaillé d'un circuit répondant aux spécifications énoncées jusqu'à présent ; expliquer éventuellement les choix effectués. Le résultat des calculs tenant sur 16 bits, on ne se préoccupera pas de la sortie des additionneurs correspondant à la retenue.

Réponse

Voir Figure 2.

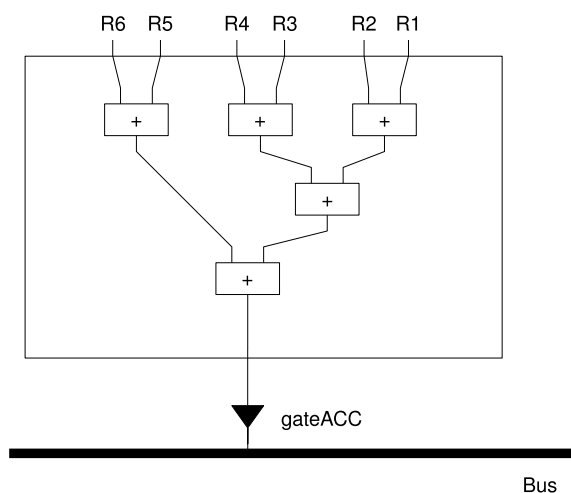


FIG. 2 – Accélérateur matériel pour la somme.

Question 1.3

En supposant que la durée d'un cycle d'horloge est très grand par rapport au passage à travers n'importe quel circuit combinatoire et à travers la mémoire, indiquer les différentes étapes effectuées par le circuit de contrôle pour l'exécution de votre instruction. Préciser le nombre de cycles nécessaires pour l'exécution de cette instruction. Pour le format de présentation de votre réponse, vous pouvez vous inspirer de l'exemple de l'instruction ADD ci-dessous :

1. $MAR \leftarrow PC$ ($LD.MAR=1$, $GatePC=1$), $PC \leftarrow PC+1$ ($LD.PC=1$, $PCMX=00$)
2. $mémoire \leftarrow MAR$ ($RW=0$)
3. $MDR \leftarrow mémoire$ ($LD.MDR=1$, $MIO.EN=1$)
4. $IR \leftarrow MDR$ ($GateMDR=1$, $LD.IR=1$)
5. $BEN \leftarrow NZP$ ($LD.BEN=1$)
6. $DR \leftarrow ALU$ ($SR1MX=01$, $DRMX=00$, $ALUK=00$, $GateALU=1$, $LD.REG=1$), $NZP \leftarrow ALU$ ($GateALU=1$, $LD.CC=1$)

Réponse

1. $MAR \leftarrow PC$ ($LD.MAR=1$, $GatePC=1$), $PC \leftarrow PC+1$ ($LD.PC=1$, $PCMX=00$)
2. $mémoire \leftarrow MAR$ ($RW=0$)
3. $MDR \leftarrow mémoire$ ($LD.MDR=1$, $MIOEN=1$)
4. $IR \leftarrow MDR$ ($GateMDR=1$, $LD.IR=1$)
5. $BEN \leftarrow NZP$ ($LD.BEN=1$)
6. $DR \leftarrow ACC$ ($DRMX=[8 :6]$, $GateACC=1$, $LD.REG=1$), $NZP \leftarrow ALU$ ($GateALU=1$, $LD.CC=1$)

Question 1.4

1. Écrire un programme assembleur qui calcule $R_1 + R_2 + R_3 + R_4 + R_5 + R_6$ (les registres sont supposés déjà contenir les opérandes du calcul), et stocke le résultat dans R_2 (on suppose toujours que le résultat tient sur 16 bits), sans utiliser l'accélérateur. On ne se souciera pas de préserver les valeurs des registres à la fin du calcul.
2. Réécrire ensuite ce programme en utilisant l'accélérateur.
3. Donner les temps d'exécution des deux programmes.

Réponse

1.

```
ADD R1, R1, R2
ADD R3, R3, R4
ADD R6, R5, R6
ADD R3, R1, R3
ADD R2, R6, R3
```
2. On peut écrire directement dans R_2 puisqu'on peut lire et écrire un registre au même cycle.

```
ACC R2
```
3. 30 cycles et 6 cycles respectivement.

Question 1.5

On veut maintenant que notre accélérateur puisse sortir non pas un mais deux résultats à la fois : $R_1 + R_2 + R_3 + R_4$ et $R_1 + R_2 + R_3 + R_4 + R_5 + R_6$. Cependant, on veut ne garder qu'une seule sortie sur le bus, on accepte donc que ces résultats seront écrits en séquence, à deux cycles différents dans le banc de registres (d'abord $R_1 + R_2 + R_3 + R_4$, puis $R_1 + R_2 + R_3 + R_4 + R_5 + R_6$). En outre, seul le second résultat peut faire l'objet d'un test.

1. Comment modifier la structure interne du circuit ? Expliquer clairement votre raisonnement, puis redessiner le schéma du circuit, et indiquer quels opérateurs vous ajoutez ou modifiez éventuellement.
2. Donner le nouveau format de votre instruction pour qu'elle puisse spécifier deux registres de destination dorénavant : $DR1$ et $DR2$. Si le choix effectué à la question 1.1 ne convient plus, vous pouvez changer complètement d'instruction.
3. Indiquer à nouveau les différentes étapes effectuées par le circuit de contrôle pour l'exécution de votre instruction.
4. Quelle est la durée, en cycles, de l'exécution de l'instruction ?

5. Ecrire un programme assembleur qui calcule $R_1 + R_2 + R_3 + R_4$ et $R_1 + R_2 + R_3 + R_4 + R_5 + R_6$ à l'aide de l'accélérateur (les registres sont supposés déjà contenir les opérandes du calcul) et stocke les résultats dans R_1 et R_2 respectivement.

Réponse

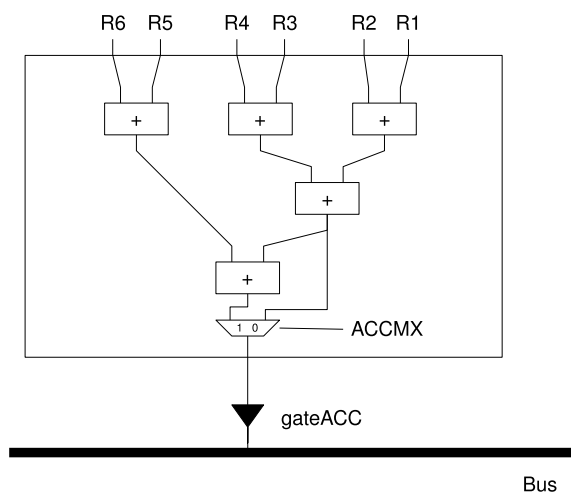


FIG. 3 – Accélérateur matériel produisant deux résultats.

1. Voir Figure 3

2. $ACC \ DR1, DR2$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	1	0	0	DR1			DR2			0	0	0

3. (a) $MAR \leftarrow PC$ ($LD.MAR=1, GatePC=1$), $PC \leftarrow PC+1$ ($LD.PC=1, PCMX=00$)
 (b) $mémoire \leftarrow MAR$ ($RW=0$)
 (c) $MDR \leftarrow mémoire$ ($LD.MDR=1, MIOEN=1$)
 (d) $IR \leftarrow MDR$ ($GateMDR=1, LD.IR=1$)
 (e) $BEN \leftarrow NZP$ ($LD.BEN=1$)
 (f) $DR1 \leftarrow ACC$ ($ACCMX=0, DRMX=[8:6], GateACC=1, LD.REG=1$)
 (g) $DR2 \leftarrow ACC$ ($ACCMX=1, DRMX=[5:3], GateACC=1, LD.REG=1$), $NZP \leftarrow ALU$ ($GateALU=1, LD.CC=1$)
4. 7 cycles.
5. Cette fois-ci, on ne peut utiliser $DR1 = R_1$ sinon on modifie R_1 avant de récupérer le résultat $R_1 + R_2 + R_3 + R_4 + R_5 + R_6$. On doit donc utiliser temporairement un autre registre, e.g., R_0 , ou une zone mémoire. Par contre, on peut écrire directement dans R_2 sans affecter le résultat.
- $ACC \ R0, R2 \ ADD \ R1, R0, \#0$

Question 1.6

On veut maintenant que notre circuit soit un peu plus flexible et puisse réaliser deux ensembles de calculs différents : soit $R_1 + R_2 + R_3 + R_4$ et $R_1 + R_2 + R_3 + R_4 + R_5 + R_6$ (comme précédemment), soit $3 \times R_1 + 5 \times R_2$ et $5 \times R_1 + 8 \times R_2$. Pour le second ensemble de calculs, on n'utilisera donc que deux des six entrées (R_1 et R_2). Pour décider quel ensemble de calculs on veut effectuer, on dispose d'une nouvelle entrée dans le circuit : un bit de configuration $cnfg$. Ce bit de configuration doit être spécifié par votre instruction.

1. Indiquer comment votre instruction peut spécifier ce bit de configuration, sans pour autant ajouter des entrées au circuit de contrôle du LC-2. A nouveau, si le choix effectué précédemment pour l'instruction ne convient plus, vous pouvez changer complètement d'instruction.
2. Comment modifier la structure interne du circuit pour qu'il puisse, selon la configuration, effectuer l'un ou l'autre des ensembles de calculs ci-dessus ? Expliquer clairement votre raisonnement, puis redessiner le schéma du circuit, et indiquer quels opérateurs vous ajoutez ou modifiez éventuellement.

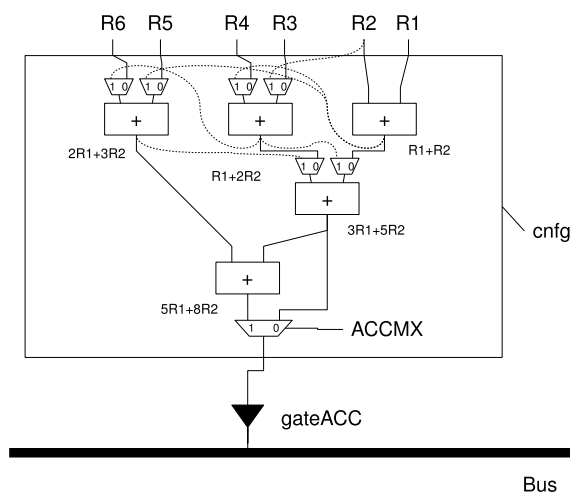


FIG. 4 – Accélérateur matériel pouvant effectuer deux ensembles de calculs différents.

Réponse

1. ACC DR1, DR2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	1	cnfg	0		DR1		DR2			0	0	0
2. Voir Figure 4

Question 1.7

On veut écrire un programme assembleur qui calcule le n^{ieme} nombre de la suite de Fibonacci. A priori, le programme source correspondant pourrait ressembler à (le premier élément a pour index $n = 0$) :

```
fn_2 = 0;
fn_1 = 1;
for (i=2; i <= n; i++) {
    fn = fn_1 + fn_2;
    fn_2 = fn_1;
    fn_1 = fn;
}
```

Mais on sait que $n = 5 \times k + 1$ ($k > 0$), et on veut calculer 5 nombres de Fibonacci (et non un seul) à chaque itération. Expliquer comment modifier le programme source ci-dessus (donner la version modifiée), puis écrire le programme assembleur correspondant. On supposera que R_0 contient n au début du programme, et le résultat sera également placé dans R_0 . On n'utilisera pas l'accélérateur dans cette question.

Réponse

Code source avec calcul par pas de 5 (en supposant $n = 5 \times k + 1$ ($k > 0$)) :

```
fn_2 = 0;
fn_1 = 1;
for (i=2; i <= n; i+=5) {
    fn = fn_1 + fn_2;
    fnp1 = fn + fn_1;
    fnp2 = fnp1 + fn;
    fnp3 = fnp2 + fnp1;
    fnp4 = fnp3 + fnp2;
    fn_2 = fnp3;
    fn_1 = fnp4;
}
```

Code assembleur :

```
NOT R0, R0;
ADD R0, R0, #1; R0 = -n
```

```

AND R1, R1, #0; fn_2 = 0
ADD R2, R1, #1; fn_1 = 0
ADD R6, R1, #2; i = 2
LOOP ADD R3, R2, R1; fn = fn_1 + fn_2
ADD R4, R3, R2; fnp1 = fn + fn_1
ADD R5, R4, R3; fnp2 = fnp1 + fn
ADD R1, R5, R4; fnp3 = fnp2 + fnp1, fn_2 = fnp3
ADD R2, R1, R5; fnp4 = fnp3 + fnp2, fn_1 = fnp4
ADD R6, R6, #5; i = i + 5
ADD R4, R0, R6; i-n
BRnz LOOP; <= 0?
ADD R0, R2, #0

```

Question 1.8

Modifier maintenant votre programme assembleur pour qu'il utilise l'accélérateur.

Réponse

Comme précédemment, on ne peut directement écrire $3 \times R_1 + 5 \times R_2$ dans R_1 sans altérer le calcul de $5 \times R_1 + 8 \times R_2$, donc on utilise R_3 comme registre temporaire.

```

NOT R0, R0;
ADD R0, R0, #1; R0 = -n
AND R1, R1, #0; fn_2 = 0
ADD R2, R1, #1; fn_1 = 0
ADD R6, R1, #2; i = 2
LOOP ACC R3, R2; R3 = 3R1+5R2, R2 = 5R1 + 8R2
ADD R1, R3, #0; R3 = R1
ADD R6, R6, #5; i = i + 5
ADD R4, R0, R6; i-n
BRnz LOOP; <= 0?
ADD R0, R2, #0

```

Question 1.9

Donner le nombre de cycles d'exécution des deux versions du programme (avec et sans accélérateur) pour $n = 11$, en détaillant les calculs.

Pourquoi est-il préférable de disposer d'un accélérateur pour un programme tel que la suite de Fibonacci, plutôt que d'un processeur multi-coeur dont le nombre de coeurs serait égal ou supérieur au nombre d'additionneurs que vous avez utilisés ?

Réponse

Sans accélérateur : $2 \times (BR + 7 \times ALU) + 6 \times ALU = 2 \times (6 + 6 \times 6) + 6 \times 6 = 120$ cycles.

Avec accélérateur : $2 \times (BR + 3 \times ALU + ACC) + 6 \times ALU = 2 \times (6 + 3 \times 6 + 7) + 6 \times 6 = 98$ cycles.

Le programme pour calculer la suite de Fibonacci peut difficilement s'exécuter en parallèle puisque chaque itération dépend du résultat de la précédente. Dans ce cas-là, la seule façon d'accélérer le calcul est d'accélérer l'exécution de chacune des étapes du calcul, par exemple en remplaçant des sous-séquences d'instructions de calcul par un circuit, comme le permet l'accélérateur. En outre, plus le circuit est conçu pour effectuer un grand nombre de calculs différents, i.e., plus il est flexible, plus la probabilité qu'un programme quelconque puisse en bénéficier devient grande.

Question 1.10

On repart de l'accélérateur de la question 1.5. On veut maintenant utiliser l'accélérateur pour effectuer une addition non signée de $R_1 + R_2 + R_3 + R_4 + R_5 + R_6$, mais dont le résultat tient sur 32 bits au lieu de 16.

Comment modifier la structure interne du circuit pour que l'accélérateur puisse effectuer ce calcul (et seulement celui-ci) ? Expliquer clairement votre raisonnement, puis redessiner le schéma du circuit, et indiquer quels opérateurs vous ajoutez ou modifiez éventuellement ; ne pas oublier de préciser le nombre de bits portés par chaque fil.

Réponse

Voir Figure 5

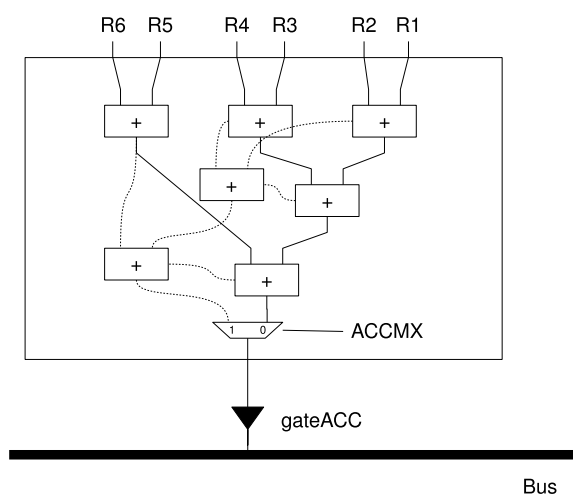
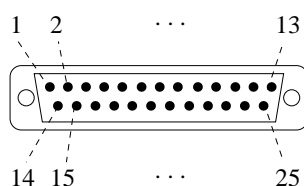


FIG. 5 – Accélérateur matériel pour la somme sur 32 bits.

Exercice 2 - Impression *via* le port parallèle (10 points)

Le port parallèle, ou port LPT pour *Line Printing Terminal*, est un connecteur situé à l'arrière des ordinateurs compatibles PC reposant sur la communication parallèle. Il a été conçu pour des imprimantes texte employant un jeu de caractères de 8 bits, l'ASCII, mais est utilisé pour un grand nombre de périphériques variés. Le but de cet exercice est de mettre en place aussi bien matériellement que logiciellement le support du port parallèle pour le LC-2 pour effectuer des impressions.

Physiquement, le port parallèle se présente sous la forme d'un connecteur DB25 à 25 broches, voir Figure 6(a), dont le brochage est précisé en Figure 6(b).



Broche	Signal	Sortie	Entrée
1	nSTROBE	✓	
2	D0	✓	✓
3	D1	✓	✓
4	D2	✓	✓
5	D3	✓	✓
6	D4	✓	✓
7	D5	✓	✓
8	D6	✓	✓
9	D7	✓	✓
10	nACK		✓
11	BUSY		✓
12	PE		✓
13	SELECT		✓
14	nAUTOFEED	✓	
15	nERROR		✓
16	nINIT	✓	
17	nSELECTIN	✓	
18-25	masse		

(a) Connecteur DB25

(b) Brochage du DB25 pour la communication parallèle

FIG. 6 – Caractéristiques du port parallèle

Chaque broche (celles reliées à la masse mises à part) est porteuse d'un signal à destination de l'ordinateur (entrée cochée en Figure 6(b)), à destination du périphérique (sortie cochée en Figure 6(b)),

ou les deux. Le détail des signaux est le suivant :

- **nSTROBE** : une transition de 1 à 0 de ce signal informe l'imprimante que de nouvelles données sont présentes sur les lignes D0 à D7 et qu'il faut les prendre en compte.
- **D0 à D7** : c'est la valeur ASCII du caractère à imprimer.
- **nACK** : l'imprimante met à 0 ce signal pour indiquer à l'ordinateur qu'elle a bien reçu le caractère transmis et qu'il peut continuer la transmission.
- **BUSY** : ce signal est mis à 1 par l'imprimante lorsque son buffer de réception est plein. L'ordinateur est ainsi averti que celle-ci ne peut plus recevoir de données. Il doit attendre que ce signal revienne à 1 pour recommencer à émettre.
- **PE** : signifie « paper error ». L'imprimante indique à l'ordinateur par un 1 que l'alimentation en papier a été interrompue.
- **SELECT** : cette ligne indique à l'ordinateur si l'imprimante est « on line » (0) ou « off line » (1).
- **nAUTOFEED** : lorsque ce signal est à 0, l'imprimante doit effectuer un saut de ligne à chaque caractère « return » reçu. En effet, certaines imprimantes se contentent d'effectuer un simple retour chariot en présence de ce caractère.
- **nERROR** : lorsque ce signal est à 0, il indique à l'ordinateur que l'imprimante a détecté une erreur et qu'il faut renvoyer l'information précédente.
- **nINIT** : lorsque ce signal est à 0, l'ordinateur demande une initialisation de l'imprimante.
- **nSELECTIN** : l'ordinateur met l'imprimante hors ligne en maintenant ce signal à 0.

Les périphériques du LC-2 contiennent des registres physiques, tous de taille 16 bits, auxquels le LC-2 peut accéder en lisant ou écrivant à des adresses spécifiques, une par registre. Ces adresses sont captées par le circuit de « contrôle des adresses », voir Figure 7. En cas de requête d'écriture à une adresse correspondant à un registre de périphérique, le circuit active en fait l'écriture dans le registre physique correspondant, et la requête n'est pas envoyée à la mémoire; la mémoire est inhibée en positionnant le signal **MEM.EN** à 0. De même, en cas de lecture à une adresse correspondant à un registre de périphérique, le circuit positionne le signal **INMUX** de façon à sélectionner la valeur provenant du registre, et **MEM.EN** à 0.

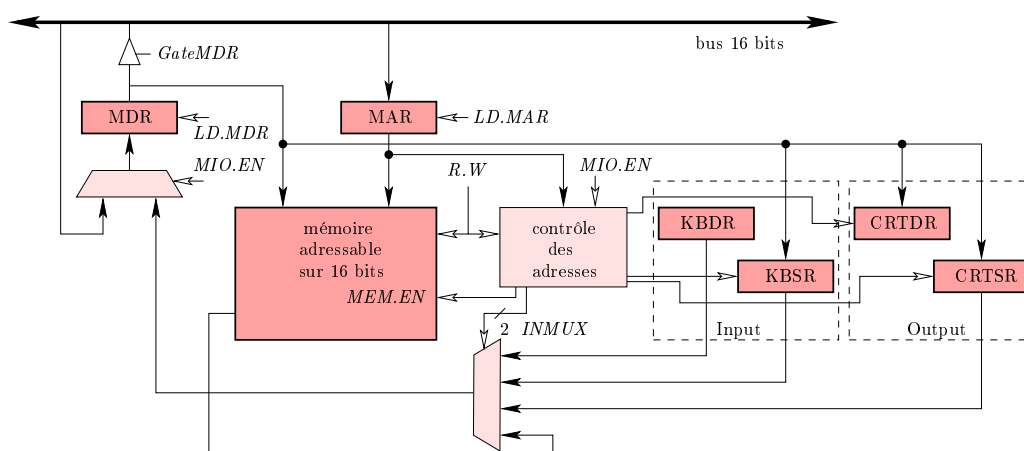


FIG. 7 – Système mémoire du LC-2

Les registres KBDR (pour Keyboard Data Register) et KBSR (pour Keyboard Status Register) en Figure 7 sont par exemple les registres physiques du clavier (CRTDR et CRTSR sont quant à eux ceux de l'écran). Leurs adresses (respectivement **xF401** et **xF400**) sont filtrées par le contrôleur d'adresses qui détermine si on tente d'accéder à la mémoire ou à un registre de périphérique. Ainsi, par exemple, lors d'une écriture sur le registre KBSR (**xF401** dans le registre MAR, signaux **MIO.EN** et **R.W** à 1 pour un accès au système mémoire en écriture), le contrôleur enverra le signal d'écriture au registre KBSR et non à la mémoire. Lors d'une lecture du registre KBSR (**xF400** dans le registre MAR, signaux **MIO.EN** à 1 et **R.W** à 0 pour un accès au système mémoire en lecture), le contrôleur envoie le signal **INMUX** adapté pour laisser passer vers le registre MDR l'information contenue dans KBSR.

Question 2.1

Sachant qu'il est impératif de correctement différencier les entrées et les sorties (en effet, écrire sur une ligne destinée à un signal d'entrée ou lire une ligne destinée à un signal de sortie peut endommager un périphérique ou un ordinateur), combien de registres 16 bits faut-il au minimum pour implémenter

le port parallèle sur le LC-2? Vous détaillerez clairement le format de chacun de ces registres. Vous utiliserez ce nombre de registres dans la suite du problème.

Réponse

Il faudra trois registres pour les dialogues avec le port parallèle : un correspondant aux données, en lecture et écriture qu'on appellera DATA, un correspondant aux commandes, en écriture seule qu'on appellera COMMAND et enfin un dernier correspondant à l'état, en lecture seule qu'on appellera STATUS.

Les registres sont sur 16 bits, on doit donc répartir les signaux. Le standard du port parallèle suit le schéma suivant :

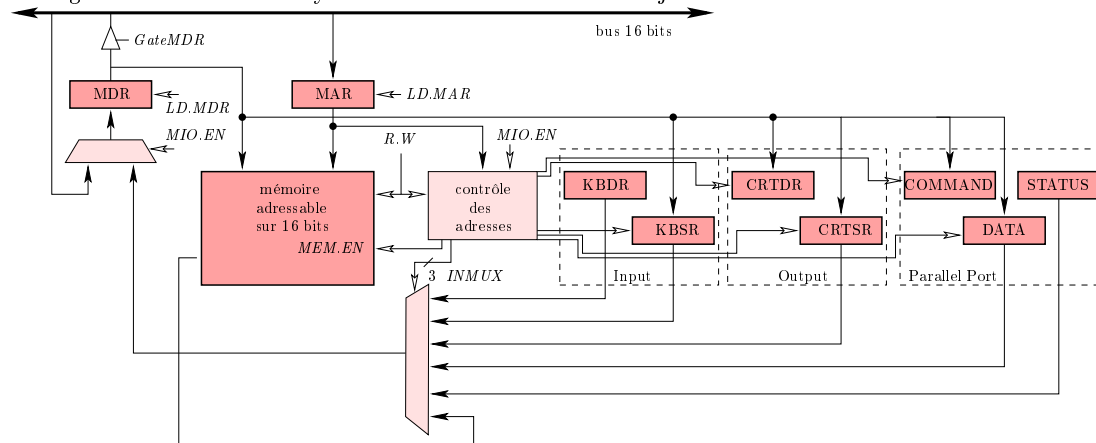
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA									D7	D6	D5	D4	D3	D2	D1	D0
COMMAND													nSELECTIN	nINIT	nAUTOFEED	nSTROBE
STATUS									BUSY	nACK	PE	SELECT	nERROR			

Question 2.2

Modifiez le diagramme des blocs du système mémoire du LC-2 rappelé en Figure 7 pour la prise en charge du port parallèle : placez chacun des registres d'entrée/sortie et indiquez clairement comment ils sont reliés au reste du circuit.

Réponse

Le diagramme des blocs du système mémoire du LC-2 mis à jour est le suivant :



Question 2.3

On suppose que l'adresse correspondant au premier registre utilisé pour le support du port parallèle est `xFF00`. Les autres registres utilisent les adresses suivantes. Faites le schéma combinatoire de la partie "contrôle des adresses" correspondant à la prise en charge du port parallèle. La gestion du signal `INMUX` n'est pas demandée.

Question 2.4

On souhaite faire en sorte que l'instruction `TRAP x32` lance l'impression d'une série de caractères. Le code de la routine de service (qui n'est pas demandé dans cette question) commence à l'adresse `x1000`. Que doit-on faire pour mettre ce mécanisme en place?

Réponse

Dans la table des vecteurs d'interruption, faire correspondre `x32` avec l'adresse `x1000`. Dans le LC-2, la table est stockée sur les 256 premiers mots de la mémoire : à l'adresse `x0032` on mettra `x1000`.

Question 2.5

Écrivez, en assembleur LC-2, la routine de service commençant à l'adresse `x1000` et qui donne l'ordre à une imprimante connectée au port parallèle d'imprimer une suite de caractères ASCII rangée en mémoire à partir de l'adresse contenue dans `R0`. La suite de caractères prend fin au premier mot mémoire contenant le code ASCII 3 (en décimal) qui signifie *end of text*. Chaque mot de 16 bits ne contient qu'un seul caractère. Vous vous aiderez de la spécification des signaux pour respecter les échanges avec l'imprimante. On supposera, pour simplifier, que l'imprimante est déjà initialisée et qu'elle est et restera en ligne (c'est à dire, on ne s'occupera pas des signaux `SELECT`, `nSELECTIN` et `nINIT`).

Réponse

```

.ORIG x1000

; Fonction IMPRIME1 pour l'impression d'une serie de caracteres
; commençant a l'adresse contenue dans R0.
;
IMPRIME1      ADD      R4,R0,#0          ; Copie de l'adresse
LECTURE      LDR      R0,R4,#0          ; Lecture d'un caractere
              ADD      R1,R0,#0          ; Copie du caractere
              AND      R2,R2,#0          ; Est-ce la fin (caractere 3)?
              ADD      R2,R2,#-3
              ADD      R1,R1,R2
              BRz      FIN
              JSR      IMPRIME1          ; Impression du caractere
              ADD      R4,R4,#1          ; Passage au caractere suivant
              JMP      LECTURE
FIN           RET

; Fonction IMPRIME1 pour l'impression d'un caractere se trouvant dans R0
;
IMPRIME1      ST       R1,BACKUP_R1
              ST       R2,BACKUP_R2
              ST       R3,BACKUP_R3
ETAT          LD       R1,STATUS          ; Lecture de l'etat dans R1
              ADD      R2,R1,#0          ; Copie de l'etat dans R2
              LD       R3,MASK_BUSY      ; Si BUSY, on recommence
              AND      R1,R1,R3
              BRnp     ETAT
              LD       R3,MASK_PE        ; Si PE, on recommence
              AND      R2,R2,R3
              BRnp     ETAT
              ST       R0,DATA           ; On ecrit le caractere
              LD       R1,ENVOI         ; Indication caractere disponible
              ST       R1,COMMAND
ATTENTE      LD       R1,STATUS          ; Lecture de l'etat dans R1
              ADD      R2,R1,#0          ; Copie de l'etat dans R2
              LD       R3,MASK_nERROR    ; Si !nERROR, on recommence
              AND      R1,R1,R3
              BRz      ETAT
              LD       R3,MASK_nACK      ; Attente de nACK
              AND      R2,R2,R3
              BRnp     ATTENTE
              LD       R1,REPOS          ; Indication plus de caractere a lire
              ST       R1,COMMAND
              LD       R1,BACKUP_R1
              LD       R2,BACKUP_R2
              LD       R3,BACKUP_R3
              RET
BACKUP_R1     .FILL    #0
BACKUP_R2     .FILL    #0
BACKUP_R3     .FILL    #0
DATA          .FILL    xFFE0
COMMAND       .FILL    xFFE1
STATUS        .FILL    xFFE2
MASK_BUSY     .FILL    x0080
MASK_nACK     .FILL    x0040
MASK_PE       .FILL    x0020
MASK_nERROR   .FILL    x0008
ENVOI         .FILL    x000C          ; Commande quand caractere envoye
REPOS         .FILL    x000D          ; Commande au repos

```

Question 2.6

Dans le cas du LC-2, on ne manipule généralement pas des caractères ASCII mais des champs de bits, des adresses ou des nombres signés sur 16 bits. Ces différentes données ne sont pas directement

exprimées sous forme de caractères ASCII, il faut donc les traduire en caractères ASCII pour pouvoir ensuite les imprimer.

Dans cette question, on va s'occuper de la traduction en caractères ASCII d'adresses hexadécimales. On veut écrire une routine (qu'on pourra appeler par JSR) qui va décomposer en caractères ASCII l'adresse contenue dans le registre R0. Dans cette question, le résultat est écrit en mémoire (et non pas envoyé à l'imprimante) à partir de l'adresse indiquée par le registre R1, à raison d'un caractère ASCII par case mémoire. La Figure 8 rappelle les principaux codes ASCII.

Code (décimal)	Caractère	Code (décimal)	Caractère
48	0	65	A
49	1	66	B
50	2	67	C
51	3	68	D
52	4	69	E
53	5	70	F
54	6	45	-
55	7	32	<i>espace</i>
56	8	13	<i>carriage return</i>
57	9	3	<i>end of text</i>

FIG. 8 – Table ASCII partielle (correspondance codes ASCII décimaux et caractères)

Réponse

```

; Fonction imprimant l'adresse contenue dans R0 a partir de l'adresse contenue dans R1
;
PRINT_ADDR    ST      R1, PA_BACKUP_R1 ; Sauvegarde des registres utilises
              ST      R2, PA_BACKUP_R2
              ST      R5, PA_BACKUP_R5
              ST      R6, PA_BACKUP_R6
              ST      R7, PA_BACKUP_R7
              AND     R5, R5, #0      ; On sauve l'adresse a traduire dans R5
              ADD     R5, R0, #0
              AND     R6, R6, #0      ; On sauve l'adresse ou ecrire dans R6
              ADD     R6, R1, #0
              AND     R1, R1, #0      ; R1 = de combien on va decaler a droite
              ADD     R1, R1, #12
PA_CODE       JSR     DECAL_DROITE
              LD      R2, PA_MASK_HEX ; On applique le masque x000F
              AND     R0, R0, R2
              ADD     R2, R0, #-10    ; On traduit en ASCII
BRzp          PA_LETTRE
              LD      R2, PA_ASCII_0 ; Cas des chiffres
              JMP     PA_FIN_TRAD
PA_LETTRE     LD      R2, PA_ASCII_A ; Cas des lettres
PA_FIN_TRAD   ADD     R0, R0, R2
              STR     R0, R6, #0      ; On ecrit le code ASCII
              ADD     R6, R6, #1      ; On incremente l'adresse ou ecrire
              ADD     R0, R5, #0      ; On restaure le nombre original
              ADD     R1, R1, #-4     ; On decalera de 4 de moins
BRzp          PA_CODE
              LD      R2, PA_FIN_CHAINE ; Caractere de fin de chaine
              STR     R6, R1, #0
              LD      R1, PA_BACKUP_R1 ; Restauration des registres utilises
              LD      R2, PA_BACKUP_R2
              LD      R5, PA_BACKUP_R5
              LD      R6, PA_BACKUP_R6
              LD      R7, PA_BACKUP_R7
              RET

PA_BACKUP_R1  .BLKW   1
PA_BACKUP_R2  .BLKW   1

```

```

PA_BACKUP_R5  .BLKW      1
PA_BACKUP_R6  .BLKW      1
PA_BACKUP_R7  .BLKW      1
PA_MASK_HEX   .FILL      x000F
PA_ASCII_A    .FILL      #55
PA_ASCII_0    .FILL      #48
PA_FIN_CHAINE .FILL      #3

; Fonction decalant un champ de bits contenu dans R0
; vers la droite d'un nombre de bits contenu dans R1
; Le resultat est place dans R0
;
DECAL_DROITE  ST          R1, DD_BACKUP_R1 ; Sauvegarde des registres utilises
              ST          R5, DD_BACKUP_R5
              ST          R6, DD_BACKUP_R6
              AND         R5, R5, #0      ; R5 = compteur nombres de bits a copier (-16+decalage)
              ADD         R5, R5, #-16
              ADD         R5, R5, R1
              AND         R6, R6, #0      ; R6 = nombre decale a droite (initialise a 0)
DD_COPIE     ADD         R0, R0, #0
              BRzp        DD_ZERO        ; Cas bit de poids fort de R0 = 0
              ADD         R6, R6, #1      ; Cas bit de poids fort de R0 = 1
DD_ZERO      ADD         R5, R5, #1      ; Incrementation du compteur
              BRzp        DD_FIN
              ADD         R0, R0, R0      ; On decale R0 a gauche
              ADD         R6, R6, R6      ; On decale R6 a gauche
              JMP         DD_COPIE        ; On va copier le bit suivant
DD_FIN       ADD         R0, R6, #0      ; On met le resultat dans R0
              LD          R1, DD_BACKUP_R1 ; Restauration des registres utilises
              LD          R5, DD_BACKUP_R5
              LD          R6, DD_BACKUP_R6
              RET

DD_BACKUP_R1  .BLKW      1
DD_BACKUP_R5  .BLKW      1
DD_BACKUP_R6  .BLKW      1

```

Question 2.7

Le premier standard de port parallèle (le Standard Parallel Port, ou SPP) utilisait les mêmes signaux que ceux décrits dans la Figure 6, à la différence que les signaux de données D0 à D7 étaient unidirectionnels de l'ordinateur vers le périphérique (seule la sortie aurait été cochée en Figure 6(b)). Les systèmes SPP Hewlett Packard Bi-tronics arrivaient cependant à transférer des données 8-bits du périphérique vers l'ordinateur. Proposez une solution pour réaliser des communications 8 bits bidirectionnelles avec les contraintes des caractéristiques SPP.

Réponse

Il reste cinq signaux du périphérique vers l'ordinateur dans la spécification SPP. Les systèmes Hewlett Packard Bi-tronics utilisaient le signal nACK comme bit de contrôle et transféraient quatre bits de données avec les autres signaux. Les informations 8 bits étaient envoyées en deux temps.

Question 2.8

Dans le cas où la communication est bidirectionnelle, que doit-on modifier dans le LC-2 pour permettre au périphérique d'avoir l'initiative de la communication ?

Réponse

Il faut prendre en charge une nouvelle interruption, en ajoutant le microcode, la routine de service, et l'entrée dans la table des vecteurs d'interruption adéquats.

- $SR1MX_2$ sélectionne l'une des quatre entrées du multiplexeur : de *haut* en *bas*, 00 et 01 (10 et 11 ne sont pas utilisés).
- $DRMX_2$ sélectionne le registre destination (signal DR) : $IR[11:9]$ pour 00, $IR[8:6]$ pour 01 (10 et 11 ne sont pas utilisés).
- $SR2MX_1$ est à part : ce signal détermine si la deuxième opérande vient du banc des registres ou du champ immédiat, il est par construction égal au bit 5 du registre d'instruction IR et n'intervient pas dans la microprogrammation.

On rappelle également la liste des instructions du LC-2 (dans sa version simplifiée) en Figure 10.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
ADD DR, SR1, SR2 (DR ← SR1 + SR2)	0	0	0	1	DR			SR1			0	0	0	SR2			
ADD DR, SR1, imm5 (DR ← SR1 + SEXT(imm5))	0	0	0	1	DR			SR1			1	imm5 : immédiat 5-bits signé					
AND DR, SR1, SR2 (DR ← AND(SR1, SR2))	0	1	0	1	DR			SR1			0	0	0	SR2			
AND DR, SR1, imm5 (DR ← AND(SR1, SEXT(imm5)))	0	1	0	1	DR			SR1			1	imm5 : immédiat 5-bits signé					
NOT DR, SR (DR ← NOT(SR))	1	0	0	1	DR			SR			1	1	1	1	1	1	1
BRnzp label (PC = PC[15:9]@offset9 if n.N+z.Z+p.P)	0	0	0	0	n	z	p	offset 9-bits non signé dans la page courante									
JMP label (PC = PC[15:9]@offset9)	0	1	0	0	0	0	0	offset 9-bits non signé dans la page courante									
JSR label (R7 ← PC and PC = PC[15:9]@offset9)	0	1	0	0	1	0	0	offset 9-bits non signé dans la page courante									
JMPR indexed address (PC = BaseR + ZEXT(offset6))	1	1	0	0	0	0	0	BaseR			index 6-bits non signé						
JSR indexed address (R7 ← PC and PC = BaseR + ZEXT(offset6))	1	1	0	0	1	0	0	BaseR			index 6-bits non signé						
LEA DR, label (DR ← PC[15:9]@offset9)	1	1	1	0	DR			offset 9-bits non signé dans la page courante									
LD DR, label (DR ← MEM(PC[15:9]@offset9))	0	0	1	0	DR			offset 9-bits non signé dans la page courante									
LDR DR, indexed address (DR ← MEM(BaseR + ZEXT(offset6)))	0	1	1	0	DR			BaseR			index 6-bits non signé						
ST SR, label (SR → MEM(PC[15:9]@offset9))	0	0	1	1	SR			offset 9-bits non signé dans la page courante									
STR SR, indexed address (SR → MEM(BaseR + ZEXT(offset6)))	0	1	1	1	SR			BaseR			index 6-bits non signé						
RET (PC ← R7)	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	

FIG. 10 – Format des instructions du LC-2