

## TD 6 - ASSEMBLEUR ET ENTRÉES/SORTIES

Programmation des interruptions, réalisation contrôleur graphique pour le processeur LC-2.

Page web : <http://www-rocq.inria.fr/~acohen/teach/archi.html>

On propose un script shell pour faciliter le chargement de code en langage machine LC-2 dans les composants SRAM8K de *DigLog* : `~cohen/bin/hex2ram`.

Utilisation :

```
> hex2ram fichier
```

Convertit `fichier.hex` (après assemblage par `lc2asm`) en `fichier_lo.ram` et `fichier_hi.ram`. Ces deux fichiers `.ram` décrivent la mémoire de poids faible (`lo`) et de poids fort (`hi`) à charger dans les composants SRAM8K “appropriés” (en fonction de l’adresse indiquée par `.ORIG` dans le code source) du circuit de la page 6 (`lc2_mem.lgf`).

### Exercice 6.1 - Routine d’interruption

Les entrées/sorties du LC-2 sont gouvernées par les quatre registres KBSR, KBDR, CRTSR et CRTDR, associés respectivement aux adresses `xF400`, `xF401`, `xF3FC`, et `xF3FF`. Toute lecture du caractère (code ASCII) contenu dans le registre KBDR (c’est-à-dire, toute lecture en mémoire à l’adresse `xF400`) provoque la mise à 1 du bit 15 du registre KBSR, alors que la frappe d’une touche au clavier provoque l’annulation de ce même bit. De surcroît, toute écriture d’un caractère dans CRTDR (c’est-à-dire, toute écriture en mémoire à l’adresse `xF3FF`) provoque la mise à 0 du bit 15 du registre CRTSR, alors que l’affichage effectif de ce caractère à l’écran remet automatiquement ce bit à 1. Ainsi, les registres KBSR et CRTSR indiquent qu’un caractère est en attente d’être lu au clavier (par le processeur) ou affiché à l’écran. Enfin, les registres KBDR/CRTDR sont « bloqués » (leur contenu n’est pas modifiable) tant que le bit 15 du registre KBSR/CRTSR correspondant est à 0.

On assemblera les codes proposés à l’aide du compilateur `lc2asm` et on utilisera les 8 circuits *DigLog* `lc2_*.lgf` pour la vérification et la simulation. Si vous avez lancé *DigLog* par la commande `diglog lc2_*.lgf`, la page 6 contient la RAM du LC-2 et la page 7 joue le rôle d’un panneau de commande et d’expérimentation.

La mémoire du LC-2 est pré-chargée à l’adresse `x3000` avec une boucle qui envoie répétitivement à l’écran (CRT) les 8 mots (caractères ASCII sur 8 bits) stockés de l’adresse `xF000` à l’adresse `xF007`, puis commande un retour à la ligne. Cette boucle est exécutée par le processeur.

Le registre KBSR est configuré de telle sorte que la saisie d’un caractère au clavier provoque une interruption de vecteur  $INTV = x10$ . Ce vecteur est lui-même pré-initialisé à l’adresse `x2000`, début de la routine d’interruption. L’effet de cette routine est de placer le caractère en mémoire de telle sorte qu’il apparaisse consécutif au caractère précédent à l’écran.

La routine d’interruption dispose ainsi d’une variable `Curseur` — initialisée à 0 — et effectue les tâches suivantes.

1. Les registres utilisés par la routine sont sauvegardés un par un dans la *pile du système*, en incrémentant à chaque fois `R6` (pour réserver un mot) puis en sauvegardant le registre à « protéger ». Contrairement à la pile des appels de procédure, aucune autre information ne doit y être stockée.
2. Transférer le code ASCII contenu dans le registre KBDR à l’adresse obtenue en ajoutant `xF000` à la valeur d’une variable `Curseur` : ce calcul correspond à l’emplacement du caractère courant sur la ligne.
3. Incrémenter la variable `Curseur` modulo 8.
4. Restaurer les registres sauvegardés dans la pile ; puis retourner au programme s’exécutant avant l’interruption à l’aide de l’instruction `RTI`.

#### Question 6.1.1

On ne s’intéresse pas au cas où plusieurs interruptions superposées doivent être exécutées (frappe rapide de plusieurs touches du clavier).

Implémenter cette routine en assembleur LC-2, puis en langage machine, et chargez-la à l’adresse `x2000`. Elle sera alors appelée automatiquement lors de la frappe d’une touche au clavier.

Tester le fonctionnement « simultané » de la boucle et de la saisie d’un caractère au clavier.

#### Réponse

Dans la solution suivante, on commence par incrémenter le curseur avant de lire le registre KBDR, et donc avant d’autoriser la frappe d’un nouveau caractère. L’utilisation de la pile permet de plus de superposer plusieurs interruptions.

```
.ORIG x2000 ; début de la routine à l’adresse x2000
ADD R6, R6, #1 ; réserver un mot dans la pile
STR R0, R6, #0 ; sauver R0
ADD R6, R6, #1 ; réserver un mot dans la pile
STR R1, R6, #0 ; sauver R1
LD R1, Curseur ; charger la variable Curseur dans R1
LD R0, Video ; R0 reçoit xF000
ADD R0, R0, R1 ; ajouter xF000 pour obtenir l’adresse du curseur
```

```

ADD    R1, R1, #1 ; incrémenter R1
AND    R1, R1, x7 ; R1 reçoit R1 modulo 8
ST     R1, Curseur ; ranger R1 en mémoire
LDI    R1, KBDR ; transférer le code ASCII de la touche dans R1
STR    R1, R0, #0 ; écrire le code ASCII dans le tampon
LDR    R1, R6, #0 ; restaurer la valeur d'origine de R1
ADD    R6, R6, #-1 ; libérer le mot correspondant de la pile
LDR    R0, R6, #0 ; restaurer la valeur d'origine de R0
ADD    R6, R6, #-1 ; libérer le mot correspondant de la pile
RTI    ; retour au code executé avant l'interruption
Curseur .FILL #0 ; variable Curseur initialisé à 0
Video .FILL xF000 ; adresse de base du tampon mémoire video
KBDR .FILL xF401 ; adresse du registre KBDR (memory-mapped I/O)
.END ; fin de la routine

```

### Question 6.1.2

Analyser les différents événements qui peuvent survenir lors de la saisie rapide de plusieurs caractères au clavier. On tentera d'améliorer la routine d'interruption de la question précédente afin de corriger les erreurs éventuelles, tout en garantissant un fonctionnement « intuitif » pour l'utilisateur du clavier : en particulier, l'ordre des caractères saisis doit être préservé.

### Réponse

Ces problèmes ont été étudiés et résolus dès la question précédente.

## Exercice 6.2 (facultatif) - Contrôleur graphique

Cet exercice est une ouverture vers des compléments en architecture, il est proposé sans correction.

En complément de l'exercice 6.1, on propose de laisser un *contrôleur graphique* se charger de la copie du tampon vers l'écran, afin de libérer le processeur. Le schéma du contrôleur graphique et sa place dans le système sont décrits sur la figure 15. On notera que le contrôleur graphique accède directement à la mémoire (sans utiliser le bus du LC-2 ni les registres MAR et MDR) et que chaque mot est écrit directement dans le registre CRTDR (sans utiliser le bus du LC-2).

En permanence, le contrôleur graphique copie mot par mot le contenu de la *mémoire vidéo* — entre  $xF000$  et  $xF007$  — dans le registre CRTDR.

L'affichage des caractères s'effectue en respectant le protocole suivant :

- attendre tant que  $CRTSR[15]$  est nul ; le système de contrôle de l'écran positionne  $CRTSR[15]$  à 1 lorsque le caractère contenu dans CRTDR a été affiché ;
- dès que  $CRTSR[15]$  vaut 1, charger CRTDR avec le caractère à afficher ; le système de contrôle de l'écran réagit en annulant  $CRTSR[15]$  ;

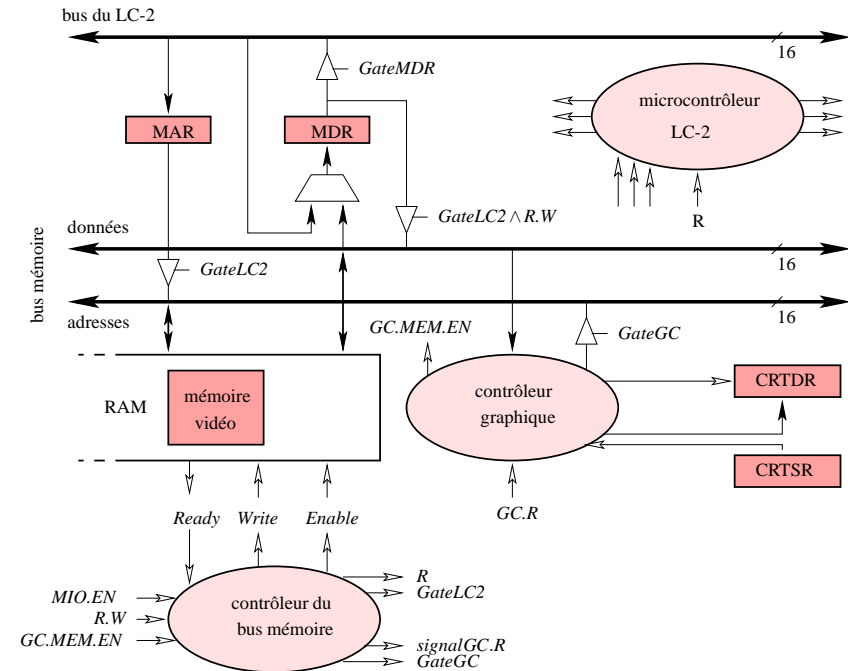


FIG. 15 – Description schématique du DMA

- passer au caractère suivant et recommencer à l'étape 1.

Le schéma de la figure 15 montre un *bus mémoire* partagé entre le processeur et le contrôleur. Ce bus est distinct du bus de données interne au processeur. Sur le circuit `lc2_mem.lgf`, il correspond aux signaux  $MEMDATA[0]$  à  $MEMDATA[15]$  pour les lignes de données et aux signaux  $MEMADDR[0]$  à  $MEMADDR[15]$  pour les lignes d'adresses. Outre les lignes de données et d'adresses, le bus comporte deux lignes de contrôle :  $GateLC2$  et  $GateGC$  permettent d'arbitrer les accès mémoire entre le processeur et le contrôleur graphique, respectivement.

Le processeur et le contrôleur graphique indiquent leurs requêtes mémoire à l'aide des signaux  $MIO.EN$  et  $GC.MEM.EN$  ; ceux-ci sont activés lorsque, respectivement, le processeur ou le contrôleur graphique veut effectuer un accès. Un troisième signal nommé  $R.W$  indique que le processeur veut lire (0) ou écrire (1).

La mémoire elle-même est contrôlée par un signal  $Enable$  qui autorise l'accès et un signal  $Write$  qui indique que l'accès s'effectue en lecture (0) ou en écriture (1). Lorsque l'accès est terminé, la mémoire envoie le signal  $Ready$ . Par ailleurs, la latence d'accès à la mémoire est réglable : de 1 cycle (par défaut) à 15 cycles (en haut à droite du panneau de contrôle), la latence 0 étant interdite.

Le contrôleur de bus répond au processeur (respectivement au contrôleur graphique) que son accès mémoire est effectué, par l'intermédiaire du signal *R* (respectivement *GC.R*). Le processeur et le contrôleur sont bloqués en attente de cette réponse, dès lors qu'ils ont initié un accès mémoire.

**Question 6.2.1**

Écrire le diagramme des blocs et l'automate de contrôle du contrôleur graphique, puis réaliser le circuit en *DigLog*. Les tests seront effectués en isolation du LC-2 (pas d'accès concurrents à la mémoire); il suffit pour cela de déconnecter les signaux *MIO.EN* et *Enable*. La mémoire vidéo sera initialisée avec une chaîne de caractères au choix.

Réponse

**Question 6.2.2**

Proposer un protocole de contrôle du bus mémoire qui permet d'arbitrer les conflits d'accès mémoire entre le processeur et le contrôleur graphique, en donnant la priorité au processeur. Écrire ce protocole à l'aide d'un automate, puis en déduire le circuit séquentiel correspondant (contrôle câblé); ce circuit est le *contrôleur du bus mémoire*. On pourra munir ce circuit de deux signaux *SwitchLC2* et *SwitchGC* autorisant les accès mémoire du LC-2 et/ou du contrôleur graphique, respectivement.

Réponse

**Question 6.2.3**

Écrire une boucle infinie qui attend la frappe de touches du clavier et met à jour la mémoire vidéo de la même manière que l'interruption de l'exercice précédent. Charger cette boucle à l'adresse *x3000* et tester l'exécution simultanée de celle-ci avec le contrôleur graphique.

Réponse

Le circuit du LC-2 avec contrôleur graphique est contenu dans l'archive *dma.tar.gz*. Il est préchargé avec le programme suivant (*kbpoll.asm*, disponible dans l'archive), conçu pour tester l'exécution concurrente du LC-2 et du contrôleur graphique.

```

.ORIG    x3000
LD      R0,Video
AND    R1,R1,#0    ; curseur
LD      R2,KBSR
Polling LDR    R3,R2,#0    ; frappe d'un caractère ?
BRn    Polling    ; boucler si ce n'est pas le cas
ADD    R3,R0,R1
ADD    R1,R1,#1    ; incrémenter le curseur
AND    R1,R1,x7
LDR    R4,R2,#1    ; lire le code du caractère
STR    R4,R3,#0
BRnzp  Polling
Video  .FILL    xF000
    
```

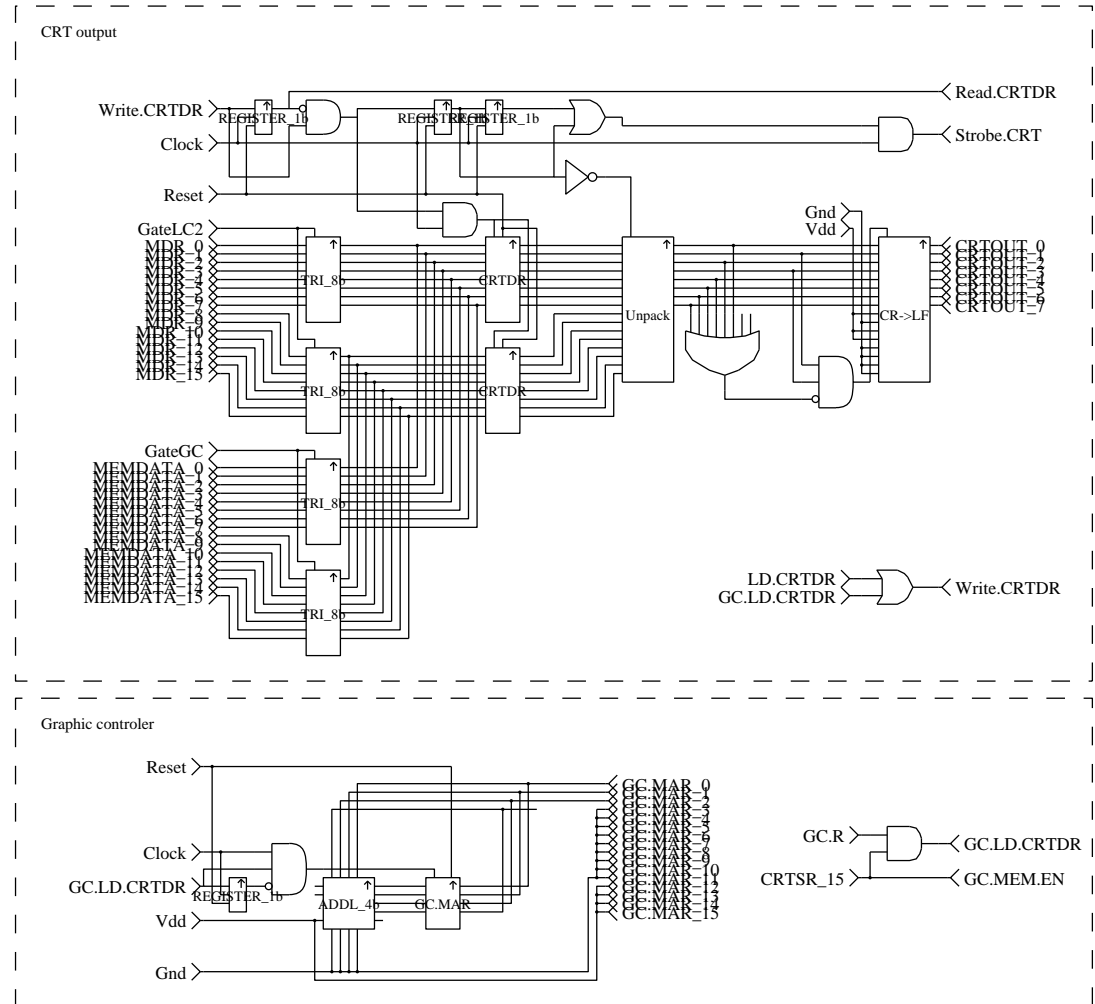


FIG. 16 – Contrôleur graphique

```

KBSR    .FILL    xF400
        .END
    
```

L'exécution simultanée ne pose pas de problèmes particuliers dans ce cas, si les circuits du

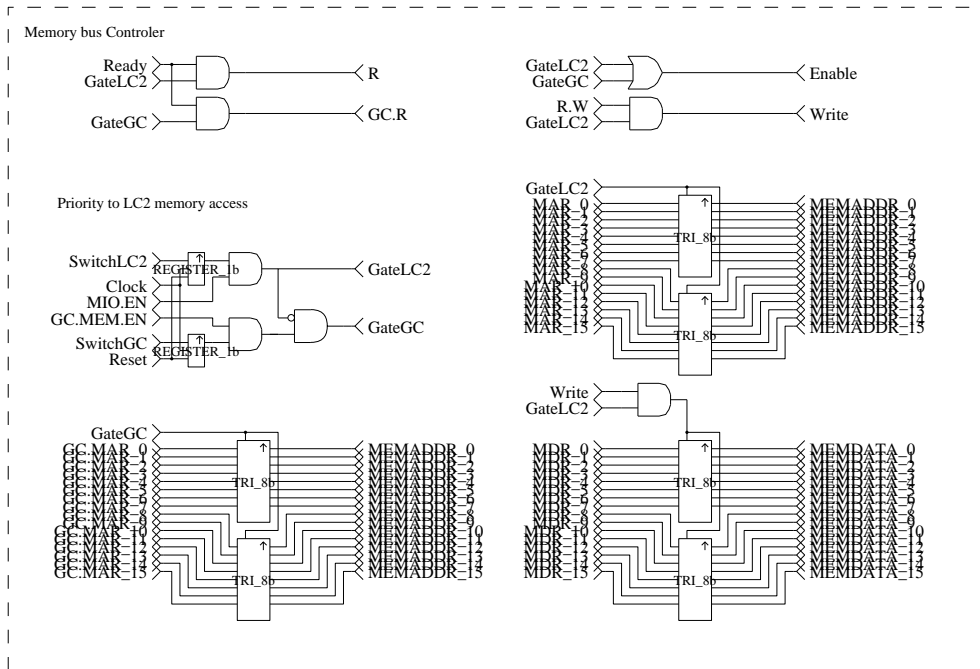


FIG. 17 – Contrôleur de bus

contrôleur graphique et du contrôleur de bus ont été réalisés correctement.

En revanche, si le programme effectue des accès concurrents au registre `CRTDR`, ça se complique. Le protocole de communication avec l'écran (avec `CRTSR`) n'est pas adapté à une exécution concurrente avec le contrôleur graphique : en fonction de la latence de la mémoire, l'un peut voler des cycles à l'autre, en lui faisant croire que l'écran est prêt à recevoir un caractère. Un système de synchronisation (sémaphore ou test-and-set) serait nécessaire. Un programme pour expérimenter ces comportements complexes est fourni avec l'archive de la correction : `concurrent.asm`, à charger à l'adresse `x3000`.

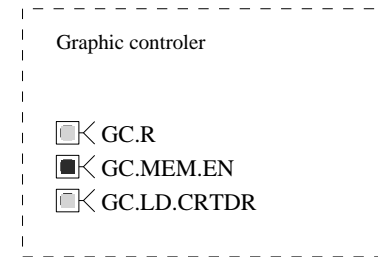
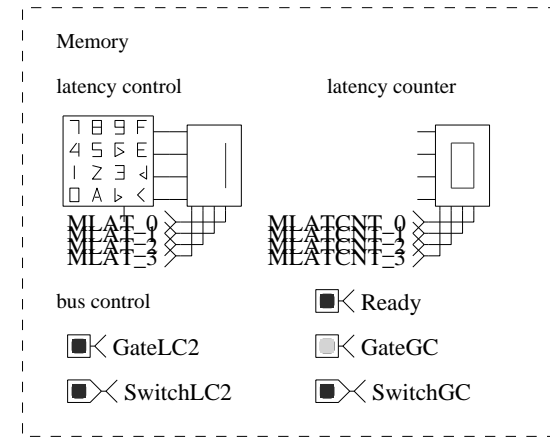


FIG. 18 – Panneau de contrôle du bus et du contrôleur graphique