Licence Maîtrise d'Informatique de Lille

Licence d'informatique – 2001/2002

USTL

Architecture des Ordinateurs: Fiche de TD 3

Microprocesseurs

novembre 2001

1 Architecture mono-bus

On considère le microprocesseur dont l'architecture est présentée Figure 1. Ce microprocesseur est composé des éléments suivants :

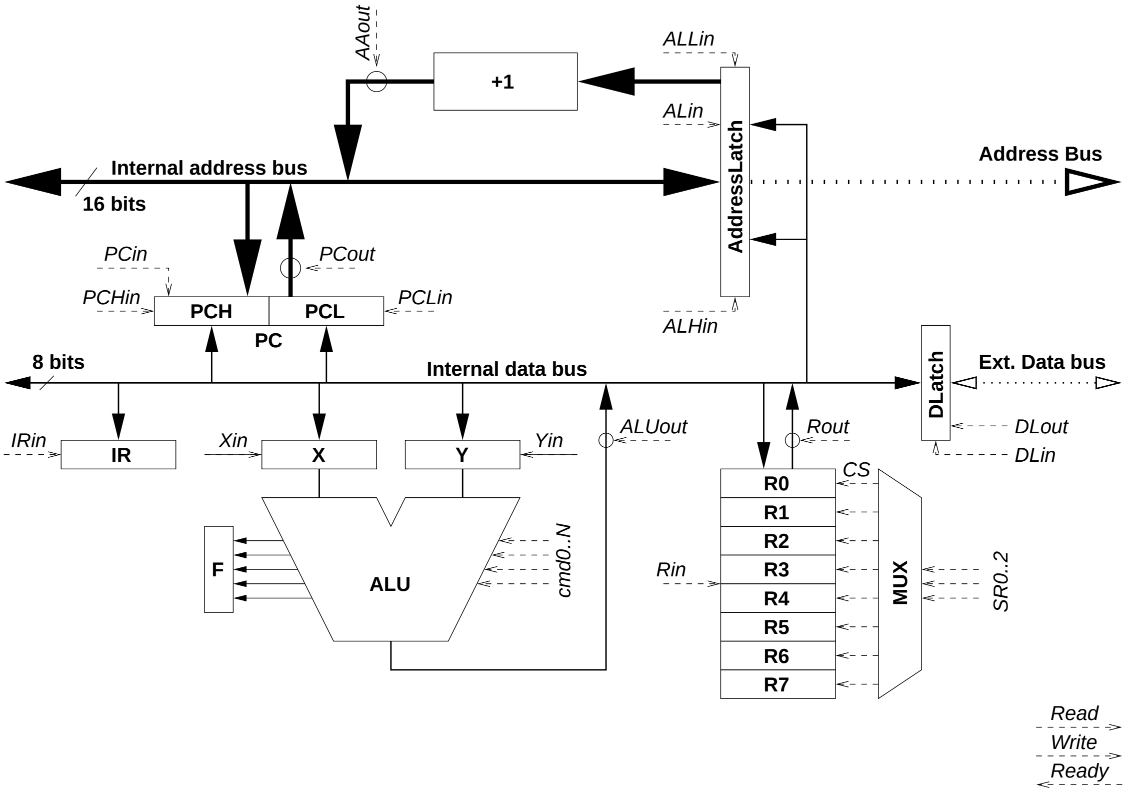

Figure 1: Architecture mono-bus

- **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
000	R0
001	R1
010	R2
011	R3
100	R4
101	R5
110	R6
111	R7

Table 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 déterminer. Un certain nombre de Flags (à déterminer) 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.

- **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 contrôle** : constitué d'un registre d'instruction IR 8 bits et d'une unité de contrôle. L'unité de contrôle est chargée de générer 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 Jeu d'instructions

On souhaite que notre microprocesseur dispose des instructions suivantes :

- `ST Rn, HHLL` — (*pour STore*) qui stocke la valeur du registre `Rn` en mémoire à l'adresse (sur 16 bits) `HHLL` ;
- `LD Rn, HHLL` — (*pour LoaD*) qui charge dans le registre `Rn` la valeur stockée en mémoire à l'adresse (sur 16 bits) `HHLL` ;
- `MV Rn, arg` — (*pour MoVe*) qui charge dans le registre `Rn` la valeur de l'argument "`arg`". L'argument est soit une valeur immédiate, soit un registre ;
- `JMP HHLL` — (*pour JuMP*) qui effectue un branchement à l'adresse (sur 16 bits) `HHLL` ;
- `ADD Rn, Rm` — (*pour ADD :-)*) qui additionne la valeur du registre `Rn` avec la valeur du registre `Rm` et stocke le résultat dans le registre `Rn` ;
- `SUB Rn, Rm` — (*pour SUBstract*) qui soustrait la valeur du registre `Rm` à la valeur du registre `Rn` et stocke le résultat dans le registre `Rn` ;
- `AND Rn, Rm` — 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` ;
- `NOT Rn` — qui inverse bit à bit le registre `Rn` et qui y stocke le résultat ;
- `JZ HHLL` — (*pour Jump if Zero*) qui effectue un saut à l'adresse 16 bits `HHLL` si l'opération précédente à donné un résultat nul ;
- `JC HHLL` — (*pour Jump if Carry*) qui effectue un saut à l'adresse 16 bits `HHLL` si l'opération précédente à engendré une retenue ;
- `NOP` — (*pour No Operation*) qui est une instruction qui ne fait rien...

Question 2.1 : *Proposez un format hexadécimal pour ces instructions en minimisant la taille des instructions (le minimum d'octets possibles). Toutes les opérations devront être de la forme* `CodeOpération Opérandes`.

Par exemple, l'instruction `JMP HHLL` *peut être codé sur trois octets :*

`FF LL HH`

Le code opération est l'octet `FF` *(255 en décimal) suivi de deux octets qui indiquent l'adresse où l'on désire faire le branchement.*

<u>*NB.*</u> *Ici on choisit de mettre d'abord l'octet de poids faible. Ceci correspond à la notation* "big endian" *(grand-boutiste en français). La notation qui consiste à mettre d'abord l'octet de poids fort est la notation* "little endian" *(petit-boutiste).*

Question 2.2 : *Écrire le programme assembleur effectuant une copie de la zone mémoire commençant à l'adresse stockée en mémoire à l'adresse 0100h dans la zone mémoire dont l'adresse est stockée en 0102h. La copie s'arrête dès que l'on a copié la valeur 00h. On considérera que les zones mémoires sont distinctes.*

L'écrire en héxadécimal en supposant que le programme commence à l'adresse 0200h.

3 Micro-programmation

Question 3.1 : *Écrire La séquence de signaux (micro-instruction) réalisant la phase 1. On écrira sur une même ligne tous les signaux pouvant être envoyés simultanément. On utilisera un "signal" particulier : $WaitMemory$ qui signale à l'unité de contrôle d'attendre le signal Ready de la mémoire.*

Question 3.2 : *Écrire la phase 3 pour les instructions dont le code d'opération est : 42h, 49h, 53h, 78h, 7Fh, 81h, F5h, 70h, 73h*