Architecture et Système

Exemple d'une architecture primitive

Ce document explique l'architecture d'un petit ordinateur primitif. Bien évidemment, les processeurs actuels sont beaucoup plus complexes. Autre qu'un manque de certains dispositifs importants (jeu d'instructions plus poussé, pile, interruptions, protection mémoire, ...) l'architecture présentée ici fait quelques simplifications et comporte des faiblesses structurelles qu'on discutera par la suite. Néanmoins, cet exercice permet de se faire une idée comment un ordinateur peut fonctionner.

1 Composants

Sauf mention contraire, tous les mots sont de 8 bits (des octets).

La **mémoire princale** (RAM) comporte $2^8 = 256$ octets.

Le **processeur** (CPU) comporte les registres suivants :

- Le compteur de programme (`PC`) qui pointe vers la prochaine instruction (ou le prochain opérand) à récupérer dans la RAM.
- Le registre d'instruction (`IR`) qui contient l'instruction à exécuter dans le cycle qui suit. Il se sous-divise dans trois parties :
  - `IC`, le code d'opération (4 bits). Il sert comme indice vers la ROM pour récupérer le prochain jeu de signaux de contrôle.
  - `RD`, indiquant un registre de *destination* (2 bits)
  - `RS`, indiquant un registre de *source* (2 bits)
- Quatres registres généraux appelés `A`, `B`, `C` et `D`.
- Deux drapeaux (bits) spéciaux:
  - Carry (`Cy`) qui garde le resultat d'une comparaison.
  - Halt (`Ht`) indique que le calcul est terminé.

La ROM contient les **signaux de contrôle**. Elle est composée de 16 mots à 16 bits.

L'**unité arithmétique-logique** (ALU) stocke le résultat d'une addition dans le registre `Z`.

Tout bit de stockage (verrou/bascule) est muni de deux signaux :

- un *signal de donnée* (D) ;
- un *signal d'activation* (A).

Après un cycle de calcul, avec le prochain tic d'horloge, le bit prend la valeur D si A égale 1 ; sinon, il reste inchangé. Le signal d'activation est partagé entre tous les bits d'un même octet.

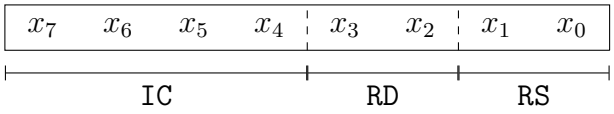

Figure 1: Sous-division du registre `IR`.

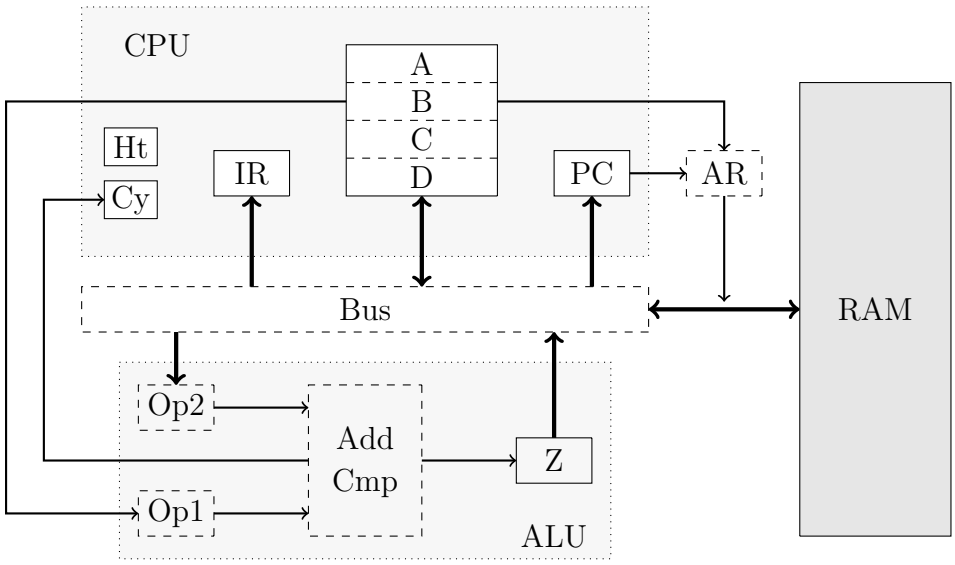

Figure 2: Flux de données dans notre ordinateur. Les boîtes en pointillé représentent des calculs intermédiaires. Pour simplicité, la ROM et les signaux de contrôle ont été omises.

2 Instructions

Notre ordinateur connaît 10 instructions machine expliquées dans le tableau ci-dessous. Certaines instructions sont entièrement codées dans un seul octet, d'autres prennent un argument qui se trouve dans le octet suivant.

Dans le tableau, la colonne 'assembleur' donne une représentation symbolique, et le codage binaire est celui utilisé dans l'ordinateur. Le code d'opération proprement dit est dans les quatre bits premiers et se trouve dans `IC`. Si l'opération prend en compte un registre, on l'appelle *destination*, il sera indiqué par les bits `dd` qui seront dans `RD`, avec `00` pour `A`, `01` pour `B` etc. Si l'opération prend un deuxième registre, on l'appelle *source*, et il correspond aux bits `ss` dans `RS`. Un bit `x` n'aura aucune importance.

assembleur	**codage binaire**	**explication**
`hlt`	`0001 xx xx`	Arrêter l'opération de l'ordinateur, mettre `Ht` à 1.
`mov R,R`	`0010 dd ss`	Mettre le contenu de la source dans la destination.
`mov R,V`	`0011 dd xx` `vvvv vv vv`	Mettre la valeur de l'octet suivant dans la destination.
`add R,R`	`0100 dd ss`	Additionner deux registres et mettre le resultat dans la destination.
`add R,V`	`0101 dd xx` `vvvv vv vv`	Additionner le registre de destination et une constante `V` dans l'octet suivant.
`cmp R,R`	`0110 dd ss`	Comparer deux registres. Si la destination est plus grande, mettre `Cy` à 1 sinon à 0.
`cmp R,V`	`0111 dd xx` `vvvv vv vv`	Analogue, mais avec destination et une constante.
`mov [R],R`	`1000 dd ss`	Écrire la valeur de la source dans la mémoire à l'adresse indiquée par la destination.
`jmp A`	`1001 xx xx` `aaaa aa aa`	Sauter vers l'adresse indiquée dans l'octet suivant.
`jg A`	`1010 xx xx` `aaaa aa aa`	Sauter vers l'adresse indiquée dans l'octet suivant si `Cy` = 1.

Exemples :

- L'instruction `mov B,C` se code `0010 0110` en binaire ou `26` en hexadécimal.
- L'instruction `mov D,42` met la valeur $66 = (42)_{16}$ dans `D` et se code `3C 42` en hexadécimal (en supposant que les `x` sont 0).

3 La ROM et les signaux de contrôle

La ROM consiste de 16 mots à 16 bits. Chaque mot représente l'opération à exécuter dans un cycle, typiquement un transfert de données. Les bits seront utilisés comme *signaux de contrôle*, ils seront fournis à d'autres endroits dans le circuit (multiplexeurs, décodeurs, signaux d'activation).

Pour une instruction machine (voir Section 2), le premier cycle est codé dans le mot dont l'indice correspond à son code d'opération (donc 1 pour `hlt` etc). Si une opération s'effectue dans plusieurs cycles, on indique le prochain cycle dans le champ `next` (voir ci-dessous). Ayant complété une instruction, on utilisera l'indice 0 pour obtenir la prochaine instruction depuis la mémoire.

Les 16 bits d'un mot seront utilisés comme suit, en commençant par le bit le plus significatif :

- Deux bits non-utilisés avec valeur 0.
- Deux bits `bi` indiquant qui doit écrire sur le bus :
  - `00` ≙ le registre source, `01` ≙ l'ALU, `1x` ≙ la mémoire
- Trois bits `bo` indiquant la destination du bus :
  - `000` ≙ `PC`, `001` ≙ `IR`, `010` ≙ le registre de destination, `011` ≙ la mémoire, `1xx` ≙ nul part

  L'ALU n'est pas parmi les destinataires. Elle effectuera un calcul dans chaque cycle, même inutile, en utilisant la valeur du bus comme deuxième opérand.
- Le bit `ar` choisit l'adresse qui sera recopiée en `AR` (voir Figure 2) et sert à choisir une case mémoire (où en lecture où en écriture) :
  - `0` ≙ `AR` égale `PC`, `1` ≙ `AR` égale le registre destination
- Le bit `op` indique l'operation à effectuer dans l'ALU, avec 0 ≙ addition et 1 ≙ comparaison.
- Le bit `pc` autorise `PC` à changer.
- Le bit `cj` indique un saut conditionnel (si `Cy` = 1).
- Le bit `hlt` indique l'arrêt du machine.
- `next`, quatre bits donnant l'adresse des signaux du prochain cycle dans la ROM.

4 Contenu de ROM

Voici (en partie) le contenu du ROM. Le tableau manque quelques lignes, on va les remplir plus tard (voir Section 6).

adresse	zéros	bi	bo	ar	op	pc	cj	hlt	next	commentaire
0	00	10	001	0	0	1	0	0	0000	fetch
1	00	00	100	0	0	0	0	1	0001	`hlt`
2										`mov RD,RS`
3	00	10	010	0	0	1	0	0	0000	`mov RD,V`
4	00	00	100	0	0	0	0	0	1011	`Z := RD+RS`
5	00	10	100	0	0	1	0	0	1011	`Z := RD+V`
6										`cmp RD,RS`
7										`cmp RD,V`
8										`mov [RD],RS`
9										`jmp A`
a										`jg A`
b	00	01	010	0	0	0	0	0	0000	`RD := Z`
c	00	00	100	0	0	0	0	0	0000	non utilisé
d	00	00	100	0	0	0	0	0	0000	non utilisé
e	00	00	100	0	0	0	0	0	0000	non utilisé
f	00	00	100	0	0	0	0	0	0000	non utilisé

L'opération 0 (fetch) travaille sous l'hypothèse que la prochaine instruction est celle qui est pointée par `PC`. Du coup, elle récupère cette case de mémoire, la met dans `IR` et avance `PC` en même temps (on va supposer qu'on possède un circuit dedié pour augmenter `PC` sans passer par l'ALU). Grâce à l'augmentation de `PC` par fetch, les opérations prenant un argument supplémentaire pourront le récupérer dans le cycle suivant.

Mis à part le cycle 'fetch', les instructions d'addition nécessitent deux cycles (dont ils partagent le deuxième, stocké à b) tandis que les autres instructions se réalisent dans un seul cycle et branchent directement vers 0.

5 Interaction des composants

Pour un vecteur x de longueur n, notons x_i le composant avec indice i, avec $0 \leq i < n$. Pour un mot s de longueur k et un vecteur x (composé de mots ou de bits) de longueur 2^k, notons $Mux(s, x)$ le résultat du multiplexeur correspondant et $Dec(s)$ le résultat du décodeur correspondant (à savoir, un mot composé de 2^k bits).

On note $Reg := \langle \mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D} \rangle$ et RAM le contenu du RAM. Du coup $Md := Mux(\mathtt{RD}, Reg)$ et $Ms := Mux(\mathtt{RS}, Reg)$ sélectionnent le contenu des registres destination et source.

Bus. On note $AR := Mux(\mathtt{ar}, \langle \mathtt{PC}, Md \rangle)$ et $ram := Mux(AR, RAM)$. Et si on décompose `bi` en deux bits b_1, b_0, alors la valeur sur le bus sera

$$Bus := Mux(b_1, \langle Mux(b_0, \langle Ms, \mathtt{Z} \rangle), ram \rangle).$$

On note par ailleurs $Bus_u := \langle Bus_7 \cdots Bus_4 \rangle$ et $Bus_\ell := \langle Bus_3 \cdots Bus_0 \rangle$.

ALU. Soit $Op_1 := Md$ et $Op_2 := Mux(\mathtt{op}, \langle Bus, \neg Bus \rangle)$, où $\neq$ signifie la négation bit par bit, et soient *Alu_carry*, *Alu_sum* le sorties de l'additionneur entre Op_1 et Op_2. Du coup, si $\mathtt{op} = 0$ alors *Alu_sum* contient le résultat de l'addition (modulo 2^8), et si $\mathtt{op} = 1$ alors $Alu\_carry = 1$ ssi $Op_1 > Op_2$.

Destination. La valeur *Bus* sera transmise vers une parmi plusieurs destinations potentielles, en fonction de l'instruction machine choisie.

D'abord on sélectionne l'unité (`PC`, `IR`, *Reg* ou *RAM*) recevant la donnée en utilisant $u := Dec(\mathtt{bo})$. En effet, certaines sorties de ce circuit (u_4 à u_7) représentent 'nul part' et peuvent être omises ; dans le cas $\mathtt{bo} = \mathtt{1xx}$, on a simplement $u_1 = u_2 = u_3 = u_4 = 0$. Par ailleurs notons $r := Dec(\mathtt{RD})$ et $m := Dec(\mathtt{AR})$. Nous pouvons désormais désigner les signaux de données et activation pour toute destination :

Destination	**Donnée**	**Activation**
`Cy`	*Alu_carry*	`op`
`Ht`	`hlt`	`hlt`
`PC`		
`IR`	$Mux(u_1, \langle \mathtt{next}, Bus_u \rangle)$	1
`RD` / `RS`	Bus_ℓ	u_1
`Z`	*Alu_sum*	1
Reg_i	*Bus*	$r_i \wedge u_2$
RAM_i	*Bus*	$m_i \wedge u_3$

6 Questions

Pour les Questions 1 et 2, vous pouvez supposer que, si `cj` $= 0$ et `bo` $\neq 000$, alors `PC` augmente de 1 si `pc` $= 1$ et reste inchangé sinon.

1. Dans le tableau de Section 4, remplissez les lignes 2 et 8 (pour les instructions `mov R,R` et `mov [R],R`).

2. Dans le tableau de Section 4, remplissez les lignes 6 et 7 (pour les opérations de comparaison).

3. Remplissez les lignes 9 et a dans le tableau ainsi que les formules activation et donnée pour `PC` dans la Section 5. Vos réponses doivent prendre en compte les consignes suivantes :

   - Aprés un cycle, le `PC` peut, en fonction de l'opération en cours, (i) rester inchangé ; (ii) prendre la valeur $pc' :=$ `PC` $+ 1$ (on suppose qu'un circuit dedié pour calculer pc' est disponible) ; (iii) prendre la valeur *Bus* (saut inconditionnel) ; (iv) choisir entre pc' et *Bus* en fonction de `Cy` (saut conditionnel).
   - Pour être compatible avec Questions 1 et 2, respecter la condition en début de cette section.

4. Les instructions suivantes représentent un programme qui calcule les carrées de 1 à 100. Traduisez-les en code hexadécimal en supposant que le programme commence à l'adresse 0 et que les carrées seront stockés à partir de l'adresse `L1` $:= (30)_{16}$.

```
     mov A,1
     mov B,1
     mov C,3
     mov D,L1
L3:  cmp A,10
     jg L2
     mov [D],B
     add B,C
     add C,2
     add A,1
     add D,1
     jmp L3
L2:  hlt
```