# TP8 : étude d'un mini processeur

Le jeu d'instructions et le processeur NanoP utilisés dans ce TP ont été conçus par Jean-Luc Béchennec.

## 1 À propos de ce TP

Ce TP sera noté. Vous devrez donc rendre (par groupes de 2) un rapport répondant aux questions posées dans le sujet. Notez bien que le plus important n'est pas la réponse à la question (bien souvent celle-ci vous sera donnée par les enseignants pendant le TP) mais la justification de cette réponse.

Les rapports sont à rendre au format pdf, sur Madoc, pour le dimanche 10 janvier à 23h55 au plus tard. Soyez à la fois précis et concis dans vos réponses.

## 2 Le processeur NanoP

NanoP est un mini processeur, construit avec Logisim. Vous pouvez récupérer le fichier décrivant celui-ci sur Madoc. NanoP fonctionne avec trois registres : A, PC et CCR.

### 2.1 Registre A

Le registre A est un accumulateur sur 8 bits. Il sert à stocker, pour chaque calcul, le premier opérande et le résultat. Tous les calculs arithmétiques et logiques ainsi que les instructions permettant de copier le contenu de A en mémoire ou le contenu d'une case mémoire dans A opèrent implicitement sur A.

1. On considère que les entiers sont représentés en complément à deux. Quel est le plus grand entier qui peut être stocké dans A ?
2. Repérez le registre A avec Logisim.

### 2.2 Registre PC

Le registre PC est un compteur ordinal sur 8 bits. Il sert à stocker l'adresse de l'instruction en cours d'exécution.

3. Que pouvez-vous déduire sur le nombre maximum d'opérations pouvant être accueillies dans la mémoire programme ?
4. Repérez le registre PC avec Logisim.
5. Qu'appelle-t-on la mémoire de programme ?
6. Repérez la mémoire de programme (les adresses contenues dans PC indiquent des emplacements dans celle-ci).

### 2.3 Registre CCR

Le registre CCR est un registre de codes de conditions (*condition code register*) sur 2 bits (N et Z). Il sert à stocker des bits donnant des informations sur le résultat de la dernière instruction arithmétique. Le bit N indique si le résultat était négatif et le bit Z indique si le résultat était nul. Ces bits sont généralement appelés des *flags*.

7. Voyez-vous d'autres informations qu'il pourrait être utile de retenir sur le résultat d'une instruction arithmétique ?
8. Repérez le registre CCR avec Logisim.

## 2.4 Mémoire de données

On considère une mémoire de données de taille très réduite : seulement 16 octets.

9. Qu'appelle-t-on la mémoire de données ?
10. Repérez cette mémoire avec Logisim.
11. Quelle taille font les adresses mémoire ?

## 2.5 Organisation du processeur

On notera que NanoP est séparé en deux parties : une partie opérative (en haut) qui se charge d'effectuer les opérations (addition, soustraction, stockages et chargements, etc) et une partie contrôle (en bas) qui se charge de sélectionner les opérations à effectuer. Cette sélection se fait en fonction des instructions stockées dans la mémoire de programme. L'ensemble du jeu d'instructions de NanoP est décrit dans la partie suivante.

12. Repérez les deux parties de NanoP sur Logisim.

# 3 Le jeu d'instructions de NanoP

On considère 13 instructions, faisant chacune un octet : 3 instructions arithmétiques, 2 comparaisons, 3 instructions de chargement et de stockage, 5 instructions de branchement.

## 3.1 Les instructions arithmétiques

### Addition à $A$ d'un opérande situé en mémoire

| Mnémonique | Sémantique | Code |
|---|---|---|
| add *addr* | $A \leftarrow A + mem[addr]$, $CCR$ modifié | 0000aaaa |

*addr* est une adresse de la mémoire de données sur 4 bits. Dans le code de l'instruction, cette adresse est stockée dans les 4 bits de poids faible : aaaa.

Le $CCR$ est modifié en fonction du résultat de l'addition.

### Addition à $A$ d'une constante

| Mnémonique | Sémantique | Code |
|---|---|---|
| addi *cst* | $A \leftarrow A + cst$, $CCR$ modifié | 1000cccc |

addi pour **add i**mmédiate. *cst* est une constante littérale stockée dans l'instruction dans les 4 bits de poids faible : cccc. On appelle aussi ce type de constante un *immédiat* car elle est immédiatement accessible en même temps que l'instruction. Cette constante est *signée* et représentée en *complément à 2*. Avant de l'additionner, une extension de signe de la constante est réalisée afin de porter le nombre de bits à 8. L'extension de signe consiste à répliquer le bit de poids fort dans les 4 bits ajoutés à gauche.

Le $CCR$ est modifié en fonction du résultat de l'addition.

13. Montrer que l'extension de signe préserve la valeur en complément à deux.
14. Quel est l'intérêt de cette extension de signe ?

### Soustraction d'un opérande situé en mémoire de $A$

| Mnémonique | Sémantique | Code |
|---|---|---|
| sub *addr* | $A \leftarrow A - mem[addr]$, $CCR$ modifié | 0001aaaa |

sub pour **sub**stract. Le $CCR$ est modifié en fonction du résultat de la soustraction.

## 3.2 Les instructions de comparaison

### Comparaison de $A$ avec le contenu d'une case mémoire

| Mnémonique | Sémantique | Code |
|---|---|---|
| cmp *addr* | $A - mem[addr]$, $CCR$ modifié | 0011aaaa |

cmp pour **cmp**are. Le contenu de la case mémoire est soustrait de $A$ et le $CCR$ est modifié en conséquence. Le résultat de la soustraction n'est pas stocké.

### Comparaison de $A$ avec une constante

| Mnémonique | Sémantique | Code |
|---|---|---|
| cmpi *cst* | $A - cst$, $CCR$ modifié | 1011cccc |

La constante est une constante signée. Une extension de signe est réalisée sur cette constante puis elle est soustraite de $A$ et le $CCR$ est modifié en conséquence. Le résultat de la soustraction n'est pas stocké.

## 3.3 Les instructions de chargement et rangement

### Chargement de $A$ avec un opérande situé en mémoire

| Mnémonique | Sémantique | Code |
|---|---|---|
| ld *addr* | $A \leftarrow mem[addr]$ | 0100aaaa |

ld pour **l**oa**d**. Le contenu de la mémoire à l'adresse *addr* est copié dans $A$.

### Chargement de $A$ avec une constante

| Mnémonique | Sémantique | Code |
|---|---|---|
| ldi *cst* | $A \leftarrow cst$ | 1100cccc |

Une extension de signe est réalisée sur cette constante puis elle est copiée dans $A$.

### Stockage du contenu de $A$ en mémoire

| Mnémonique | Sémantique | Code |
|---|---|---|
| st *addr* | $mem[addr] \leftarrow A$ | 0110aaaa |

st pour **st**ore. Le contenu de $A$ est copié en mémoire à l'adresse *addr*.

## 3.4 Les instructions de branchement

L'opérande des instructions de branchement est un déplacement, compté en nombre d'instructions, signé, en complément à 2. Si le branchement est pris le déplacement subit une extension de signe et est ajouté au $PC$.

### Branchement inconditionnel

| Mnémonique | Sémantique | Code |
|---|---|---|
| ba *dep* | $PC \leftarrow PC + dep$ | 1111dddd |

### Branchement si plus petit que

| Mnémonique | Sémantique | Code |
|---|---|---|
| blt *dep* | si $(N == 1)$ $PC \leftarrow PC + dep$ | 1001dddd |

blt pour **b**ranch if **l**ower **t**han. Plus petit que se réfère à l'instruction cmp précédente si elle existe. Comme cette instruction a comparé le contenu de $A$ à l'opérande, $N$ sera à 1 si $A$ contenait une valeur strictement inférieure à l'opérande.

#### Branchement si plus grand ou égal à

| Mnémonique | Sémantique | Code |
|---|---|---|
| bge *dep* | si $(N == 0)$ $PC \leftarrow PC + dep$ | 1010dddd |

bge pour **b**ranch if **g**reater or **e**qual. Plus grand ou égal se réfère à l'instruction cmp précédente si elle existe. Comme cette instruction a comparé le contenu de $A$ à l'opérande, $N$ sera à 0 si $A$ contenait une valeur supérieure ou égale à l'opérande.

#### Branchement si égal

| Mnémonique | Sémantique | Code |
|---|---|---|
| beq *dep* | si $(Z == 1)$ $PC \leftarrow PC + dep$ | 1101dddd |

beq pour **b**ranch if **eq**ual. Égal se réfère à l'instruction cmp précédente si elle existe. Comme cette instruction a comparé le contenu de $A$ à l'opérande, $Z$ sera à 1 si $A$ contenait une valeur égale à l'opérande.

#### Branchement si différent

| Mnémonique | Sémantique | Code |
|---|---|---|
| bne *dep* | si $(Z == 0)$ $PC \leftarrow PC + dep$ | 1110dddd |

bne pour **b**ranch if **n**ot **e**qual. Différent se réfère à l'instruction cmp précédente si elle existe. Comme cette instruction a comparé le contenu de $A$ à l'opérande, $Z$ sera à 0 si $A$ contenait une valeur différente de l'opérande.

## 4 Un premier programme

```
0: ldi 0
1: st 0
2: addi 1
3: cmpi 5
4: blt -2
```

15. Quelle valeur est contenue dans A à la fin de ce programme ?
16. Donnez le code machine correspondant.
17. Rentrez ce programme dans la mémoire de programme de NanoP. Que se passe-t-il ?
18. Un élément important pour le fonctionnement d'un processeur est la présence d'une horloge. Pouvez-vous repérer celle de NanoP ? Comment fonctionne-t-elle ? Quel est son rôle ?
19. En cliquant sur l'horloge, exécutez votre programme.

## 5 Fonctionnement de NanoP

NanoP utilise des circuits logiques appelés multiplexeurs (MUX sur le schéma).

20. Comment fonctionnent ces circuits ?
21. Quelle est leur utilité pour NanoP ?

En plus de la mémoire de programme et de la mémoire de données, NanoP utilise deux autres mémoires (en l'occurrence des ROMs).

22. Repérez ces mémoires avec Logisim.
23. Quelle est leur utilité ?
24. Exécutez à nouveau le programme de la partie précédente en vous arrêtant à chaque instruction afin de comprendre comment elle est décodée puis exécutée.

## 6 Un programme un peu plus compliqué

```
 0: ldi 1
 1: st 1
 2: ld 0
 3: st 2
 4: ld 1
 5: add 1
 6: st 1
 7: ba 2
 8: ba -5
 9: ld 2
10: addi -1
11: beq -3
12: ld 0
13: addi -1
14: st 0
15: bne -7
16: ld 1
```

25. Que fait ce programme ? (on s'intéresse à la valeur ultimement contenue dans A, et on considère qu'un argument a été stocké préalablement dans le premier bloc de la mémoire de données)
26. Donnez le code machine correspondant.
27. Rentrez ce programme dans la mémoire de programme de NanoP et exécutez-le.
28. Quelle est l'utilité des branchements inconditionnels ?
29. Pouvez-vous imaginer une manière d'éviter ces branchements (au moins dans le code assembleur) ?

## 7 ≪ Compilation ≫

```c
int main(){
  int n = 5;
  int i = -1;
  while (n >= 0) {
    i = i + 2;
    n = n - i;
  }
  i = i - 1;
  return i;
}
```

30. Que retourne ce programme C ? (selon $n$)
31. Réécrivez-le en utilisant le jeu d'instructions présenté plus haut (ultimement la valeur retournée par le programme C sera contenue dans A, la valeur de $n$ est considérée initialement contenue dans le premier emplacement de la mémoire de données).
32. Commentez les limitations de votre programme.

## 8 Aller plus loin

Vous aurez remarqué qu'il y a trois ≪ trous ≫ dans le jeu d'instructions utilisé dans ce TD. Pouvez-vous suggérer des instructions utiles à ajouter ? (Par exemple, afin de pouvoir utiliser des tableaux de données.)