

Ce TP est à rendre, car il compte pour une note de travaux pratiques. Il faudra donc déposer une feuille de réponses faite pendant la séance sur Moodle.

Le travail est personnel, la note est individuelle. Toute tentative pour copier les réponses de quelqu'un d'autre sera sanctionnée par un zéro (Discord et autres). Et le TP doit être commencé et fini uniquement pendant les séances de TP, pas à la maison. Par contre, vous pouvez communiquer verbalement pour vous aider, pendant les séances. Le deuxième DS portera sur le contenu des TP.

👉 Téléchargez [ReponsesTP2.txt](#) et mettez votre nom/prénom et groupe de TP aux endroits indiqués. Ne modifiez surtout pas la structure de ce fichier.

Le TP2 porte sur l'écriture de traitements conditionnels : comment faire pour exécuter ou non une partie du programme selon une condition, et comment faire pour recommencer une partie du programme tant qu'une condition est vraie ou jusqu'à ce qu'elle soit fausse. C'est ce qu'on appelle des *structures de contrôle* : alternatives et boucles définissent le déroulement du programme.

👉 Ouvrez l'URL <https://perso.univ-rennes1.fr/pierre.nerzic/IntroArchi/CimPU/>.

👉 Mettez CimPU en niveau 3.

1. Techniques de base

Il y a deux points à comprendre, 1) comment code-t-on une alternative ou une boucle et 2) comment code-t-on une condition ? On commence par le dernier point.

1.1. Codage d'un calcul de condition simple

Les conditions les plus simples sont des comparaisons, par exemple : $N1 == 0$, $i < 10$, etc. Il y a deux valeurs et un opérateur de comparaison : $v_1 \text{ op } v_2$. Les valeurs peuvent être des constantes ou des variables. Le but est de savoir si cette condition est vraie ou fausse. On le saura avec les indicateurs VCNZ de l'unité arithmétique.

Ces indicateurs sont affectés par l'instruction `CMP reg, valeur`. On doit fournir un nom de registre et une valeur qui peut être soit une constante, ex: `CMP R0,3`, soit un autre registre, ex: `CMP R0,R1`, soit une cellule mémoire, ex: `CMP R0,[adresse]`. L'instruction `CMP` calcule une soustraction entre ses opérandes. Ça affecte les indicateurs VCNZ. Par exemple, en cas d'égalité des opérandes, Z est mis à 1.

Avec cette instruction, la démarche est simple et toujours la même :

1. Simplifier la condition au maximum, mettre les variables d'un côté, les constantes de l'autre.
2. Écrire les instructions qui placent la première variable de la condition (pas les constantes) dans un registre. Les constantes seront employées directement.
3. Utiliser `CMP` sur les deux valeurs (registres ou constantes).

Exemples :

```
; V1 == 0 ?  
LD R0, [V1]  
CMP R0, 0
```

```
    ; V2 < 10 ?
    LD R0, [V2]
    CMP R0, 10

    ; V1 > V2 ?
    LD R0, [V1]
    CMP R0, [V2]

    ; V1 + V2 < 5 ?
    LD R0, [V1]
    ADD R0, [V2]
    CMP R0, 5

    ; V1 > V2 + V3 ?
    LD R0, [V1]
    LD R1, [V2]
    ADD R1, [V3]
    CMP R0, R1

; variables
V1:   RB 1   ; RB 1 = réserver 1 octet non initialisé
V2:   RB 1   ; RB 1 = réserver 1 octet non initialisé
V3:   RB 1   ; RB 1 = réserver 1 octet non initialisé
```

Ça doit vous sembler bizarre qu'on fasse toujours la même chose quelque soit l'opérateur de comparaison dans la condition. Dans tous les cas, on prépare les choses à comparer dans des registres et ensuite on exécute `CMP`. C'est ce qui se passe ensuite qui change selon l'opérateur.

1.1.1. Exercice

👉 Codez le calcul des conditions suivantes dans ce programme :



```
;; programme

    ; V2 == 9 ?

    ; V1 != V2 ?

    ; V2 > V1 ?

    ; V1 + 1 == 6 ?

    ; V1 + V2 - 1 < 3 ?

    ; V1 + 1 > V2 + 3 ?

    ; 2*V1 - 1 >= V1 + 2 ?
```

```
    ; 12 == 17 ?  
    ; non mais ça va pas ? cette condition toujours fausse n'est pas à coder !  
  
    ; fin du programme  
    HLT  
  
;; variables du programme  
V1:    RB 1    ; RB 1 = réserver 1 octet non initialisé  
V2:    RB 1    ; RB 1 = réserver 1 octet non initialisé
```

☛ Vous devez simplifier chaque condition au maximum : amener toutes les constantes à droite et les variables à gauche, et programmer les calculs. Suivez les exemples précédents.

ATTENTION: ne changez pas la structure de ce programme (commentaire et ligne vide). Mettez vos réponses juste en dessous de chaque commentaire et laissez bien une ligne vide après. Si vous enlevez le commentaire ou la ligne vide, votre réponse ne sera pas prise en compte.

☛ Une fois fini, copiez-collez ce programme dans `ReponsesTP2.txt`. On ne peut pas le tester car il ne calcule rien.

1.2. Saut selon la condition

Le principe général à comprendre quelle que soit la structure de contrôle, c'est :

Quand la condition est fausse, on doit sauter la partie du programme qui serait à exécuter si la condition était vraie.

☛ Relisez bien cette phrase et imprégnez-vous de ce schéma.

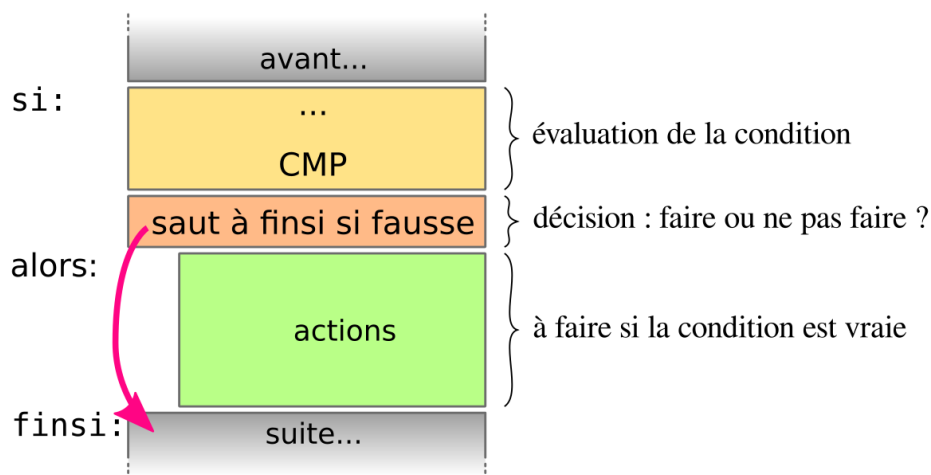


Figure 1: Saut à FINSI si la condition est fausse

Donc, forcément, tout repose sur l'instruction qui fait le saut. C'est elle qui consulte les indicateurs VCNZ affectés par l'instruction CMP.

Alors il y a deux cas possibles : soit vous considérez que les nombres comparés sont sans signe (entiers naturels de 0 à 255), soit les nombres comparés sont en convention $C2^8$ (-128 à +127).

1.2.1. Sauts pour ces comparaisons non signées

op	Saut si	Branchement	Signification
<	≥	BHS	Branch if higher or same
≤	>	BHI	Branch if higher
=	≠	BNE	Branch if not equal
≠	=	BEQ	Branch if equal
≥	<	BLO	Branch if lower
>	≤	BLS	Branch if lower or same

Ce tableau vous donne l'instruction de saut à choisir selon l'opérateur de la condition, pour sauter quand la comparaison entre deux entiers non signés est fausse.

1.2.2. Sauts pour ces comparaisons signées

op	Saut si	Branchement	Signification
<	≥	BGE	Branch if greater or equal
≤	>	BGT	Branch if greater than
=	≠	BNE	Branch if not equal
≠	=	BEQ	Branch if equal
≥	<	BLT	Branch if less than
>	≤	BLE	Branch if less or equal

Ce tableau vous donne l'instruction de saut à choisir selon l'opérateur de la condition, pour sauter quand la comparaison entre deux entiers en convention $C2^8$ est fausse.

1.3. Codage des structures de contrôle

Les [TD7](#) et [TD8](#), ainsi que le [CM3](#) montrent comment coder des alternatives et des boucles.

Voici par exemple comment on code une alternative *si/alors/sinon*. Le principe est de sauter au *sinon* si la condition est fausse ; et aussi, à la fin du *alors*, de sauter au *finsi*.

Langage C

```

-----
if (condition)
{
    action1;
}
else {
    action2;
}
-----
    
```

CimPU

```

-----
si:      ; calcul de la condition
        CMP reg, ...
        Bop sinon ; saut si faux

alors:
        ; action1...
        BRA finsi ; sauter toujours

sinon:
        ; action2...

finsi:
-----
    
```

Vous devez choisir l'instruction Bop en fonction de l'opérateur *op* de la condition.

NB: quand il y a plusieurs structures de contrôle, chacune possédant ses labels, il faut numéroter les labels. Par exemple *si1*, *alors1*, *si2*, *alors2*...

Une boucle TantQue ressemble structurellement (quand on la code avec des instructions) à une conditionnelle. La seule différence, c'est l'ajout d'un BRA à la fin des actions à faire quand la condition est vraie, pour revenir au test de la condition.

Langage C


```
-----  
while (condition)  
{  
    action1;  
}  
-----
```

CimPU

```
-----  
tantque: ; calcul de la condition  
        CMP reg, ...  
        Bop fintantque ; saut si faux  
faire:  
        ; action1...  
        BRA tantque ; sauter toujours  
fintantque:  
-----
```

Vous devez comprendre la logique du codage des structures de contrôle.

2. Exercice : chiffre hexadécimal

☛ Complétez ce programme. Il y a trois alternatives successives, donc attention à numéroter les labels. 

```
;; programme  
    ; lire un nombre sur le port 10 et le mettre dans N1  
    IN R0, 10  
    ST R0, [N1]  
  
    ; partie à programmer  
    ; if (N1 < 16) {  
    ;     N2 = N1 + 55 ; 55='A'-10  
    ; }  
    ; if (N1 <= 9) {  
    ;     N2 = N1 + '0'  
    ; }  
    ; if (N1 > 15) {  
    ;     N2 = '?'  
    ; }  
  
    ; afficher N2 sur le port 11  
    LD R0, 11 ; passer le DSKY en mode 11  
    OUT R0, 255  
    LD R0, [N2] ; R0 = valeur de N2  
    OUT R0, 11 ; R0 = code ascii => affichage du caractère  
    ; fin du programme
```

HLT

```
;; variables du programme
N1:   RB 1   ; RB 1 = réserver 1 octet non initialisé
N2:   DB '@'
```

Le principe est de construire le code ASCII du chiffre : quand il est entre 0 et 9, c'est le code de 0+le chiffre, quand il est entre 10 et 15, c'est le code de A+ le chiffre -10. Les conditionnelles sont dans un certain ordre à cause de l'absence de *sinon* (c'est voulu).

NB: les variables sont des entiers non signés (mais ça ne changerait rien s'ils l'étaient).

☛ Terminez-le, assemblez-le et essayez-le avec différents nombres assez petits, entre 0 et 20 et regardez ce qu'il affiche. Il met ? si le nombre n'est pas un chiffre hexadécimal, @ s'il y a un souci avec votre programmation, et si votre nombre est entre 0 et 15, il affiche le chiffre hexadécimal correspondant.

☛ Une fois qu'il marche, copiez-collez ce programme dans `ReponsesTP2.txt`.

Vérifiez maintenant si votre `ReponsesTP2.txt` est bien enregistré. Si vos quotas disque sont épuisés, les nouveaux fichiers sont vides ! Utilisez du `-hs ~` pour savoir combien de place vous occupez et `rm -fr .cache` pour récupérer un peu de place.

3. Exercice : division euclidienne

☛ Complétez ce programme :



```
;; division euclidienne d'un entier par un autre
; lire un nombre sur le port 10 et le mettre dans N1
IN  R0, 10
ST  R0, [N1]
; lire un autre nombre sur le port 10 et le mettre dans N2
IN  R0, 10
ST  R0, [N2]

; partie à programmer
; if (N2 > 0) {
;     Q = 0
;     while (N1 >= N2) {
;         N1 = N1 - N2
;         Q = Q + 1
;     }
; } else {
;     Q = 255
;     N1 = 255
; }

; afficher Q puis N1 sur le port 10
LD  R0, [Q]
OUT R0, 10
```

```
LD R0, [N1]
OUT R0, 10
; fin du programme
HLT

;; variables du programme
N1:   RB 1
N2:   RB 1
Q:    RB 1
```

NB: il y a une mise à zéro de Q avant la boucle.

☛ Terminez-le, assemblez-le et essayez-le avec différents nombres. Ce programme affiche le quotient et le reste de la division du premier nombre par le second.

☛ Une fois qu'il marche, copiez-collez ce programme dans ReponsesTP2.txt.

4. Exercice : Plus Grand Diviseur Commun

Soient deux entiers, N1 et N2. On va calculer leur PGCD (ou PGDC). C'est le nombre le plus grand qui divise de manière entière les deux nombres. Ça sert entre autres, à réduire une fraction. Par exemple $\frac{91}{52} = \frac{7}{4}$ parce que $91 = 7 \times 13$ et $52 = 4 \times 13$, 13 est le PGCD de 91 et 52.

Le principe est de soustraire le plus petit du plus grand nombre, tant qu'ils sont différents.

☛ Complétez ce programme :



```
;; calcul du PGCD de deux entiers (non signés)
; lire un nombre sur le port 10 et le mettre dans N1
IN R0, 10
ST R0, [N1]
; lire un autre nombre sur le port 10 et le mettre dans N2
IN R0, 10
ST R0, [N2]

; partie à programmer
; while (N1 != N2) {
;     if (N1 > N2) {
;         N1 = N1 - N2
;     } else {
;         N2 = N2 - N1
;     }
; }

; afficher N1 sur le port 10
LD R0, [N1]
OUT R0, 10
; fin du programme
HLT
```

```
;; variables du programme
N1:    RB 1
N2:    RB 1
```

☛ Terminez-le, assemblez-le et essayez-le avec différents nombres: 91 et 52, 52 et 91, 17 et 37 (ils sont premiers entre eux)...

☛ Une fois qu'il marche, copiez-collez ce programme dans ReponsesTP2.txt.

5. Exercice : méthode de Héron

Voici un dernier algorithme mathématique. Il permet de calculer la racine carrée d'un nombre N par approximations successives. L'idée de Héron (1^{er} siècle avant JC) est de trouver le nombre n tel que $n * n = N$ en construisant des rectangles dont la surface reste égale à N et qui sont « de plus en plus carrés », c'est à dire dont les côtés sont de plus en plus égaux. Au début, le rectangle est allongé et à la fin, le rectangle est carré (largeur = hauteur). À chaque itération, la surface reste égale à N , donc à la fin, le côté du rectangle est la racine carrée de N .

La question est : comment construit-on un rectangle ayant un côté de longueur n tel que sa surface soit N ? Ce que propose Héron, c'est de considérer que l'autre côté est N/n , comme ça la surface du rectangle est $n \times (N/n) = N$. Alors quand, en plus, on a l'égalité des deux côtés, c'est à dire $n = N/n$, c'est que n est la racine carrée de N . Mais comment arriver à ça? Tout simplement, dit Héron, en calculant la moyenne entre n et N/n et en la remettant dans n . Ainsi, à chaque itération, n évolue vers une valeur qui est de plus en plus proche de N/n .

Donc, ce qu'il faut, c'est initialiser n à une valeur quelconque comprise entre 1 et N , et réaffecter n avec $(n + (N/n))/2$ jusqu'à ce que n ne change plus, ou qu'on ait fait un certain nombre de tours.

Voici un programme à compléter. Le calcul compliqué vous est fourni :



```
;; calcul de la racine carrée d'un entier par la méthode de Héron
; lire un nombre sur le port 10 et le mettre dans N
; NB: ce nombre est nécessairement non signé : 0..255
IN  R0, 10
ST  R0, [N]
LSR R0          ; n = N/2 pour le point de départ
ST  R0, [n]

; partie à programmer
; i=8
; répéter {

;     calculer n = (n + N/n) / 2
LD  R0, [N]    ; R0 = N
DIV R0, [n]    ; R0 = N/n
ADD R0, [n]    ; R0 = (N/n) + N
LSR R0        ; R0 = ((N/n) + N) / 2
ST  R0, [n]    ; n = ((N/n) + N) / 2

;     i = i - 1;
```



```
    ; } jusqu'à (i==0);

    ; afficher n sur le port 10
    LD R0, [n]
    OUT R0, 10
    ; fin du programme
    HLT

;; variables du programme
N:    RB 1
n:    RB 1
i:    RB 1
```

☛ Mettez CimPU en niveau 4. L'instruction de division DIV n'est pas disponible en dessous de ce niveau.

☛ Terminez-le, assemblez-le et essayez-le avec différents nombres, en vérifiant à la calculatrice. Comment ça, on n'a que des entiers (en plus, on ne peut pas calculer racine de 255) ? Voilà pourquoi on cherche à avoir des processeurs dont les nombres sont plus grands qu'un octet (processeurs 64 bits par exemple).

☛ Une fois qu'il marche, copiez-collez ce programme dans `ReponsesTP2.txt`.

Au passage, vous avez vu comment on calcule une expression numérique assez complexe. Ça peut être très compact et très efficace sur un processeur.

NB: vous voyez que la variable *i* n'est pas vraiment nécessaire, car on peut utiliser R1 à sa place.

Pensez à déposer votre `ReponsesTP2.txt` dans la zone de dépôt sur Moodle. Rappel : c'est un travail personnel, individuel qui est soumis à une notation. Les tricheries seront sanctionnées.