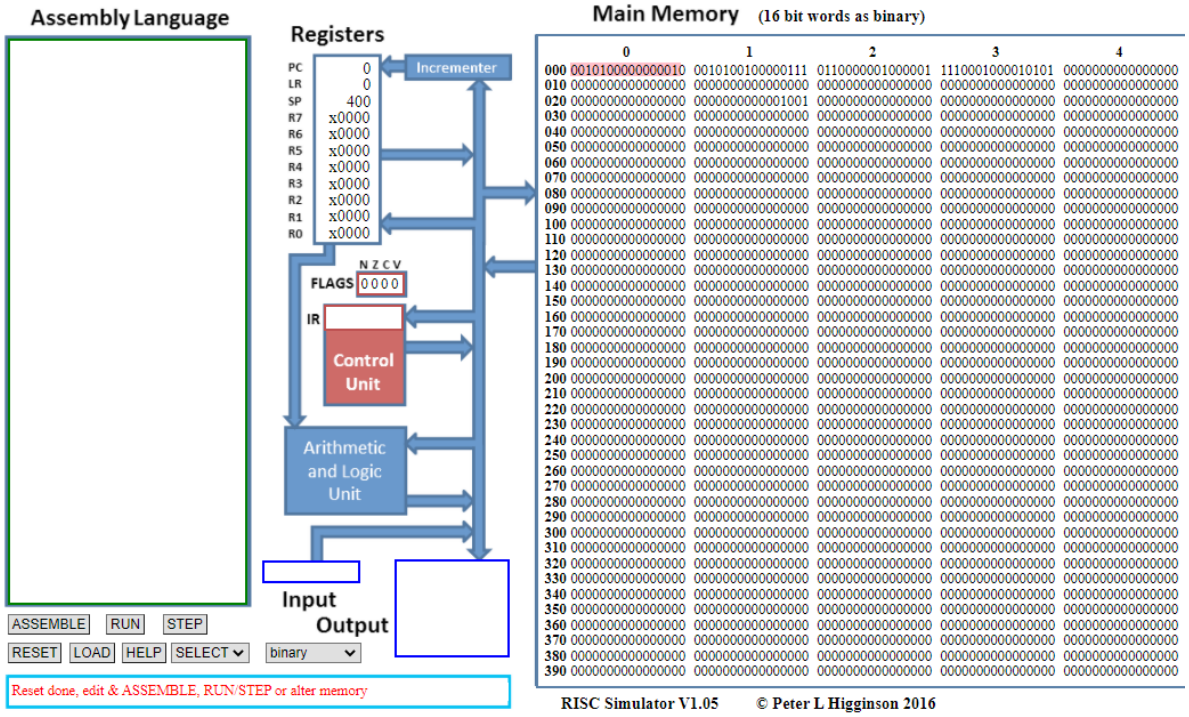
Architectures matérielles et système d'exploitation	Chap 02	**Modèle d'architecture d'un ordinateur**

EXERCICES

Exercice 1 : comprendre l'exécution d'un programme

Ci-dessous, une capture d'écran du simulateur RISC développé par Peter Higginson (https://peterhigginso
co.uk/RISC/)



1. Identifier, en les entourant, les quatre parties de l'architecture de von Neumann sur ce simulateur. Entourer également le CPU, puis localiser les registres `PC` et `IR`.
2. Traduire par une phrase chacune des instructions du programme assembleur suivant (s'aider du tableau du cours) :

```
MOV R0,#34
STR R0,33
HLT
```

3. **(Dans le simulateur)** Recopier ce programme dans la zone « Assembly Language », puis valider avec le bouton « Submit ». Le programme est traduit en langage machine et stocké en mémoire à partir de l'adresse 0. Repérer et recopier les mots binaires obtenus.
4. **(Dans le simulateur)** Exécuter le programme pas à pas avec le bouton « STEP ». Pour chaque instruction, décrire en détail ce qui se passe, en faisant le lien avec le cours.

Exercice 2 : comprendre l'exécution d'un programme (suite)

1. Traduire par une phrase chacune des instructions du programme suivant :

```
MOV R0,#34
MOV R1,#5
SUB R0,#30
ADD R0,R0,R1
STR R0,12
HLT
```

2. Quels sont les états des registres à la fin du programme, et quelle est la valeur stockée dans la case mémoire 12 ?
3. **(Dans le simulateur)** Recopier le programme, lancer l'exécution et observer ce qui se passe à chaque étape, en faisant le lien avec le cours.

Exercice 3 : instructions conditionnelles

Il existe d'autres instructions que celles vues en cours. En voici quelques-unes, importantes, concernant les **comparaisons** et les **sauts** (ruptures de séquence), qui permettent de réaliser des tests.

Exemple 1 — Instructions de comparaison et de saut

Instruction	Signification
`BRA 42`	Saut inconditionnel (**branch**) : la prochaine instruction à exécuter se situe à l'adresse 42.
`CMP R0,#23`	Compare la valeur de `R0` au nombre 23. Doit précéder un saut conditionnel (`BEQ`, `BNE`, `BGT`, `BLT`).
`CMP R0,R1`	Compare les valeurs de `R0` et `R1`.
`BEQ 78`	Après un `CMP` : saute à l'adresse 78 si les deux valeurs comparées sont égales (**Branch if EQual**).
`BNE 78`	Saute à l'adresse 78 si les deux valeurs sont différentes (**Branch if Not Equal**).
`BGT 78`	Saute à l'adresse 78 si la première valeur est strictement supérieure à la seconde (**Branch if Greater Than**).
`BLT 78`	Saute à l'adresse 78 si la première valeur est strictement inférieure à la seconde (**Branch if Less Than**).

1. À l'aide du tableau ci-dessus, traduire chacune des instructions suivantes :

```
CMP R2,#100
BNE 36
CMP R1,R2
BGT 36
```

2. Donner l'instruction en assembleur correspondant à la phrase suivante : « Si la valeur du registre `R3` est strictement inférieure à 5, la prochaine instruction à exécuter se situe à l'adresse mémoire 40. »

Utilisation d'étiquettes

En pratique, on utilise plutôt des **étiquettes** (**labels**) avec `BRA`, `BEQ`, `BNE`, `BGT`, `BLT` : on remplace l'adresse mémoire par le nom d'une étiquette. C'est l'assembleur qui se charge de convertir chaque étiquette en adresse mémoire.

Par exemple, voici un programme utilisant une étiquette nommée `Sinon` (chaque ligne est commentée) :

```
MOV R0,#5           // stocke 5 dans R0
MOV R2,#6           // stocke 6 dans R2
CMP R0,R2           // compare R0 et R2
BNE Sinon           // si R0 != R2, saute a l'etiquette Sinon
ADD R0,R0,R2        // R0 <- R0 + R2
STR R0,42           // stocke R0 a l'adresse memoire 42
HLT                 // arrete le programme
Sinon SUB R0,R0,R2  // R0 <- R0 - R2
      STR R0,42     // stocke R0 a l'adresse memoire 42
      HLT           // arrete le programme
```

3. Proposer un programme Python pouvant correspondre à ce programme assembleur. On nommera `a` et `b` les variables associées aux registres `R0` et `R2` (indication : il y a une instruction conditionnelle à formaliser).

4. Traduire en assembleur le programme Python suivant :

```
a = 3
b = 2
if a <= 5:
    b = a + b
else:
    a = a + 3
```

Exercice 4 : décodage du code machine

Le jeu d'instructions du simulateur donne le code binaire de chaque opération. En voici une sélection.

Exemple 2 — Codes d'opération

Code	Assembleur	Description
`0000 0`	`HLT`	Arrête le programme.
`0001 0`	`ADD Rd,#nb`	Ajoute `nb` à `Rd`, résultat dans `Rd`.
`0001 1`	`SUB Rd,#nb`	Soustrait `nb` à `Rd`, résultat dans `Rd`.
`0010 0`	`CMP Rb,#nb`	Compare `nb` à `Rb`.
`0010 1`	`MOV Rd,#nb`	Place `nb` dans `Rd`.
`0110 000`	`ADD Rd,Rs,Rb`	Range `Rs + Rb` dans `Rd`.
`0110 001`	`SUB Rd,Rs,Rb`	Range `Rs - Rb` dans `Rd`.
`100...`	`BRA / B<cond>`	Instructions de saut (voir tableau ci-dessous).
`1110`	`STR Rd,adr`	Stocke `Rd` à l'adresse `adr`.
`1111`	`LDR Rd,adr`	Charge dans `Rd` la valeur de l'adresse `adr`.
`0111 0110 10`	`CMP Rd,Rs`	Compare `Rs` à `Rd`.

Exemple 3 — Codes des instructions de saut

Code	Assembleur
`1000 000`	`BRA`
`1000 001`	`BEQ`
`1000 010`	`BNE`
`1001 100`	`BGT`
`1001 101`	`BLT`

Règles de codage

Dans le simulateur RISC :

- il y a 8 registres, de `R0` à `R7`, chacun codé sur **3 bits** (`000` pour `R0`, `001` pour `R1`, …, `111` pour `R7`) ;
- les adresses mémoires sont codées par leur numéro en binaire ;
- chaque instruction est codée sur **16 bits**.

Les instructions de saut ont la forme `100x xxxa aaaa aaaa`, où `xxxx` indique le type de comparaison (4 bits) et `aaaaaaaaa` l'adresse mémoire visée (9 bits).

1. Pour chaque instruction machine ci-dessous, identifier le **code d'opération** et la valeur des **opérandes**.

Instruction machine	Assembleur
0010101000001011	`MOV R2,#11`
0110000101100001	`ADD R5,R4,R1`
1000010000110100	`BNE 50`

2. Traduire en langage machine l'instruction `CMP R3,#13`, puis l'instruction `BLT 20`.
3. En jouant le rôle de l'unité de contrôle lors de la phase de **décodage**, retrouver les instructions assembleur correspondant au code machine suivant :

```
0010100000000101
1111001000001100
0110000001001000
1110001000001101
0000000000000000
```

Conversions en Python

Pour convertir un entier en binaire :

```
>>> bin(27)
'0b11011'
```

La valeur binaire de 27 est donc `11011` (on ne garde que ce qui suit `0b`). On peut ajouter autant de `0` que nécessaire à gauche sans changer la valeur.

Pour convertir un nombre binaire en entier :

```
>>> int('11011', 2)
27
```

Le second argument `2` indique que le nombre est donné en base 2.

Exercice 5 (bonus)

Écrire en assembleur les instructions correspondant à l'algorithme suivant :

```
s <- 0
Lire x
Tant que x >= 0 faire
    s <- s + x
    Lire x
fin Tant que
Afficher s
```

On supposera que les variables `s` et `x` sont stockées respectivement dans les registres `R0` et `R1`.

Aide

Utiliser à bon escient la documentation du simulateur RISC peterhigginson.co.uk/RISC/instruction_set.pdf. Par exemple, l'instruction « Lire `x` » s'écrit `INP R1,2`.