URCA - UFR Sciences
Département MMI

Licence INFO - Info0204
Eléments d'architecture des ordinateurs

Travaux dirigés n° 4

Langage d'assemblage

Exercice 1 (Interprétation de code, exercice type)
On considère le code machine suivant :

```
01 0C 30 0A 40 05 00 00 60 06 00 01 11 12 70 00
```

1°) Traduisez ce code machine en langage assembleur (utilisation des mnémoniques). Quelles sont les cases mémoire éventuellement utilisées par ce programme, en consultation et/ou en modification ?

2°) On suppose que, dans la mémoire de données, la case 12 contient la valeur 22 et que les cases 0 et 1 contiennent respectivement les valeurs 7 et 13.
Suivez pas à pas l'exécution du code (vous présenterez les effets sur les différents registres et sur la mémoire de données, sous forme d'un tableau).

3°) Qu'en est-il si la case 12 de la mémoire de données contenait en fait la valeur 11 ? la valeur 10 ?

4°) Que fait ce programme dans le cas général ? Vous représenterez les différentes actions sous forme d'un organigramme, puis d'un algorithme.

Mémoire de données et accumulateur

Exercice 2
1°) On suppose que les 8 premiers octets de la mémoire de données contiennent, en hexadecimal, `0123456789ABCDEF`.
Donnez le contenu de l'accumulateur après chacune des instructions suivantes :

```
LOAD 14        LOAD #14        LOAD Mem[7]        LOAD Mem[4]
```

2°) Quel serait le contenu de l'accumulateur après l'exécution des deux instructions (successives) `LOAD 33` et `SUB #33` ?

3°) Ecrivez un programme assembleur permettant d'échanger les contenus des cases mémoire 1 et 2.

Mémoire d'instructions

Exercice 3
On considère le code assembleur suivant :

```
0 : In
1 : MUL 3
2 : CMP #15
3 : JC 5
4 : SUB 2
5 : Out
6 : END
```

1°) Traduisez ce code en langage machine, en hexadecimal puis en binaire.

2°) Peut-on l'interpréter comme une suite d'octets ? de caractères ASCII ? d'entiers signés représentés en complément à 2 sur un octet ? de flottants simple précision (selon la norme IEEE 754) ?

3°) Quel est le résultat de son exécution si la valeur transmise par l'utilisateur est 20 (resp. 3, 7, 100 ou 90) ?

4°) Représentez l'effet général de ce code sous la forme d'un organigramme.

Unité Arithmétique et Logique - Accumulateur

Exercice 4
Donnez des codes assembleur permettant :

1. d'additionner les valeurs 3 et 7, et de stocker le résultat à l'adresse mémoire 13 ;
2. de multiplier par 5 le contenu de la case mémoire n° 2 ;
3. de retrancher de la valeur stockée dans la case n° 2 de la mémoire de données celle stockée dans la case n° 1 ;
4. d'échanger les contenus des cases n° 1 et 2 de la mémoire, sans utiliser d'autre case mémoire : proposez une solution différente de celle de la dernière question de l'exercice 2 ; cette nouvelle solution est-elle toujours satisfaisante ?

Exercice 5
On suppose disposer de valeurs a, b, c, d, … z, rangées respectivement dans les cases 1 à 26 de la mémoire de données (vous pouvez utiliser les autres cases, selon les besoins).

1°) Ecrivez un programme en langage assembleur permettant d'évaluer la valeur de $a+b$, et de ranger le résultat dans la case n° 0 de la mémoire de données.

2°) Même chose
— pour l'évaluation de l'expression $(a+b)\times(c+d)$;
— pour l'expression $(a+b)^2\times(a+c)^3-(a+d)\times(a+e)$.

3°) Quel serait le résultat de l'évaluation de l'expression
$a\times(a+1)\times\ldots\times(a+255)$?

Saut en avant

Exercice 6

1°) On dispose du code assembleur suivant, qui agit sur les contenus de la case n° 0 de la mémoire de donnée :

```
0 : LOAD Mem[0]        3 : SUB 3              6 : STORE Mem[0]
1 : CMP 5              4 : JMP 6              7 : END
2 : JZ 5               5 : ADD 3
```

Qu'effectue ce code si la case n° 0 de la mémoire de donnée contient initialement la valeur 7 ? la valeur 5 ? la valeur 0 ?
Exprimez le résultat de son exécution dans le cas général.

2°) Même question avec le code suivant :

```
0 : LOAD Mem[0]        4 : ADD #80            8 : STORE Mem[0]
1 : MUL 2              5 : JMP 8              9 : END
2 : JC 6               6 : LOAD Mem[0]
3 : LOAD Mem[0]        7 : SUB #80
```

Comparez l'effet de ce code à celui obtenu en ne conservant que les lignes 0, 4, 8 et 9.

3°) Ecrivez un code assembleur premettant d'arrondir un entier au multiple de 16 le plus proche : vous pouvez comparer à 8 le reste de la division par 16 (essayez sur des exemples).

Exercice 7 (Suite de Syracuse)

Ecrivez un code assembleur permettant, à partir du contenu de la case mémoire n° 0 :
- de le diviser par 2 s'il est pair ;
- de le tripler puis ajouter 1 dans le cas contraire.

Exercice 8

On considère le contenu de la case 0 de la mémoire de données comme représentant un caractère ASCII (on suppose que la valeur est bien entre 0 et 127).
Ecrivez un code assembleur permettant de transformer cette lettre en majuscule (passage en majuscule s'il s'agit d'une minuscule ; pas de changement dans le cas contraire).

Saut en arrière

Exercice 9

Dressez le tableau d'exécution du code assembleur suivant :

```
0 : LOAD 1             3 : MUL 2              6 : END
1 : CMP 1CO            4 : JMP 1
2 : JC 6               5 : STORE Mem[0]
```

1°) Que fait ce code ?

2°) Transformez le pour calculer, après que l'utilisateur ait saisi une valeur, la première puissance de 2 supérieure à la quantité saisie

3°) ... puis la dernière puissance de 2 juste inférieure à la quantité (proposez deux solutions).

Exercice 10

Ecrivez un code assembleur permettant de calculer la somme des n premiers entiers : la valeur de n est stockée dans la case n° 0 de la mémoire de donnée (ou lue au clavier) ; le résultat du calcul sera rangé dans la case n° 1 (ou affiché à l'écran).

Exercice 11

1°) Ecrivez un code assembleur permettant de multiplier par 40 le contenu de la case mémoire n° 0.

2°) Même chose, mais cette fois sans utiliser le mnémonique `MUL`.

3°) Ecrivez un code assembleur permettant d'élever à la puissance 40 le contenu de la case mémoire n° 0.

Exercice 12 (Calcul du PGCD par les différences)

On souhaite, par exemple, calculer le PGCD d de 34 et 16 :
- $d = pgcd(34, 16) = pgcd(34 - 16, 16) = pgcd(18, 16)$;
- en poursuivant de même on a $d = pgcd(18, 16) = pgcd(2, 16) = pgcd(2, 14) = pgcd(2, 12) = \ldots = pgcd(2, 2) = 2$.
- Notez qu'on arrête les simplifications dès que les deux valeurs deviennent égales.

On peut utiliser cette méthode pour calculer le PGCD de deux entiers strictement positifs quelconques.

On suppose disposer, dans les deux premières cases de la mémoire de données, de deux entiers strictement positifs.
Écrivez un code assembleur permettant de calculer leur PGCD par la méthode des différences, et de le ranger dans la case mémoire n° 0 (on ne se soucie pas de conserver les valeurs initiales).