Liste 2 : saisies, affichages et alternatives pour flottants à double précision (1 séance)

Notions à considérer pour pouvoir faire les exercices

Instructions utiles :

- *COMISD opd, ops* compare *opd* avec *ops* qui sont 2 flottants à double précision. L'opérande *opd* n'est pas modifié, car le but de la comparaison est uniquement de modifier certains indicateurs. Les opérandes acceptées par COMISD sont les mêmes que celles acceptées par ADDSD, SUBSD, etc.

- Après l'exécution de *COMISD opd, ops*, voici les instructions de branchement conditionnel utilisables (ces instructions de branchement ne sont pas toutes les mêmes que celles utilisées avec les entiers signés) :

Condition	*Instruction*	*Branchement si*	*Signification*
opd = ops	JE	ZF = 1	Jump if Equal
opd <> ops	JNE	ZF = 0	Jmp if Not Equal
opd > ops	JA ou JNBE	CF = 0 et ZF = 0	Jump if Above ou Jump if Not Below nor Equal
opd < ops	JB ou JNAE	CF = 1	Jump if Below ou Jump if Not Above nor Equal
opd >= ops	JAE ou JNB	CF = 0	Jump if Above or Equal ou Jump if Not Below
opd <= ops	JBE ou JNA	CF = 1 ou ZF = 1	Jump if Below or Equal ou Jump if Not Above

Construction en assembleur d'une alternative if... else avec des doubles :

Exemple :

```
double d = 8.0;

if (d <= 4)
      printf("contenu du if");
else
      printf("contenu du else");
```

Etapes de conversion vers les instructions en assembleur :

1. Les 2 éléments d et 4 de l'expression booléenne (d <= 4) vont devenir les 2 opérandes de l'instruction COMISD :

```
      movsd  xmm0, d
      comisd xmm0, dtmp
```

   dtmp est une constante du type double contenant la valeur 4 qui est déclarée ainsi : `const double dtmp = 4.0;`

2. Après COMISD, on ajoute l'instruction JCC qui correspond à l'opérateur de comparaison utilisé en C qui est, dans cet exemple, l'opérateur <=. Ceci donne JBE contenuif dont le but est d'effectuer un branchement vers l'intérieur du if uniquement quand l'expression booléenne (d <= 4) est vraie.

```
      movsd  xmm0, d
      comisd xmm0, dtmp
      jbe    contenuif

contenuif:
      // contenu du if
```

3. Après JBE contenuif, on ajoute l'instruction JCC opposée à JBE qui est JNBE contenuelse, de sorte à effectuer un branchement vers l'étiquette contenuelse uniquement quand l'expression booléenne (d <= 4) est fausse.

```
      movsd  xmm0, d
      comisd xmm0, dtmp
      jbe    contenuif
      jnbe   contenuelse

contenuif:
      // contenu du if

contenuelse:
      // contenu du else
```

4. Pour éviter d'entrer automatiquement dans le contenu du else lorsque se termine l'exécution du contenu du if, on ajoute à la fin de celui-ci l'instruction JMP finif.

```
      movsd  xmm0, d
      comisd xmm0, dtmp
      jbe    contenuif
      jnbe   contenuelse

contenuif:
      // contenu du if

      jmp    finif

contenuelse:
      // contenu du else

finif:
```

5. On ajoute les instructions à l'intérieur du contenu du if et du contenu du else. Le résultat final est le suivant :

```
#include <stdio.h>

double d = 8.0;
const double dtmp = 4.0;
const char msgif[] = "contenu du if";
const char msgelse[] = "contenu du else";

void main()
{
	_asm
	{
		movsd  xmm0, d
		comisd xmm0, dtmp
		jbe    contenuif
		jnbe   contenuelse

	contenuif:
		push   offset msgif
		call   dword ptr printf
		add    esp, 4

		jmp    finif

	contenuelse:
		push   offset msgelse
		call   dword ptr printf
		add    esp, 4

	finif:
	}
}
```

6. On peut optimiser le programme ainsi :

```
#include <stdio.h>

double d = 8.0;
const double dtmp = 4.0;
const char msgif[] = "contenu du if";
const char msgelse[] = "contenu du else";

void main()
{
	_asm
	{
		movsd  xmm0, d
		comisd xmm0, dtmp
		jnbe   contenuelse

	contenuif:
		push   offset msgif
		call   dword ptr printf
		add    esp, 4

		jmp    finif
```

```
        contenuelse:
                push    offset msgelse
                call    dword ptr printf
                add     esp, 4

        finif:
        }
}
```

On a enlevé l'instruction JBE contenuif, car elle n'est pas indispensable. Ainsi, soit l'instruction JNBE contenuelse effectue un branchement vers l'étiquette contenuelse, soit ce branchement n'a pas lieu et l'exécution du contenu du if commence alors.

Saisies, affichages et alternative avec des flottants à double précision :

Exemple :

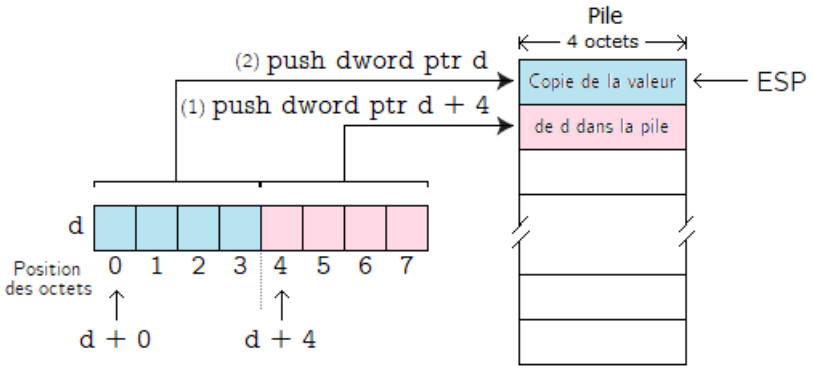
```
#include<stdio.h>

const char formatSaisie[] = "%lf";
const char txtSaisie[] = "Entrer le 1er nombre : ";
const char txtSaisie1[] = "Entrer le 2e nombre : ";
const char formatAffichage[] = "Le nombre %.2lf est plus grand que le nombre %.2lf";
const char formatAffichage1[] = "Le nombre %.2lf est plus grand ou egal au nombre %.2lf";
double d, d1;

void main()
{
        _asm
        {
                push offset txtSaisie
                call dword ptr printf
                add  esp, 4

                push offset d
                push offset formatSaisie
                call dword ptr scanf
                add  esp, 8

                push offset txtSaisie1
                call dword ptr printf
                add  esp, 4

                push offset d1
                push offset formatSaisie
                call dword ptr scanf
                add  esp, 8

                /* if (d < d1)
                        printf("Le nombre %.2lf est plus grand que le nombre %.2lf", d1, d);
                   else
                        printf("Le nombre %.2lf est plus grand ou egal au nombre %.2lf", d, d1);
                */

                movsd  xmm0, d
                comisd xmm0, d1
                jb  contenuif                   // cette instruction est non nécessaire
                jnb contenuelse

        contenuif:
                push dword ptr d + 4
```

```
                push dword ptr d
                push dword ptr d1 + 4
                push dword ptr d1
                push offset formatAffichage
                call dword ptr printf
                add  esp, 20

                jmp  fin

        contenuelse:
                push dword ptr d1 + 4
                push dword ptr d1
                push dword ptr d + 4
                push dword ptr d
                push offset formatAffichage1
                call dword ptr printf
                add  esp, 20

        fin:
        }
}
```

Quelques commentaires :

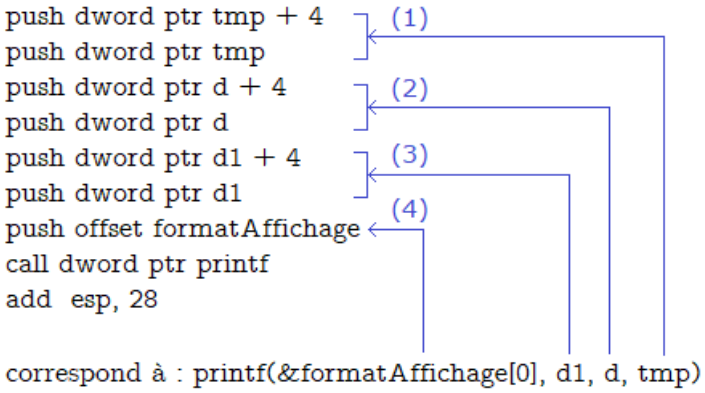
- Pour empiler la valeur d'une variable ou d'une constante du type double, il faut utiliser à 2 reprises l'instruction push, car un double est codé sur 8 octets et l'instruction push ne permet que d'empiler 4 octets à la fois.

  Par exemple, voici comment empiler la valeur de la variable d :



  On empile tout d'abord les 4 octets de la variable d les plus éloignés en mémoire avec push dword ptr d + 4, puis on empile les 4 octets situé au début de la variable d avec push dword ptr d.

- Le passage des paramètres en assembleur à une fonction se fait dans l'ordre contraire à l'ordre en C d'écriture des paramètres entre les parenthèses.



<u>L'instruction de sélection switch :</u>

L'instruction switch permet de tester le contenu d'une variable de type **entier** (char, short, ou int) avec plusieurs valeurs connues à l'avance.

Exemple :

```
int i;
```

Langage C	*Assembleur*

`switch (i)`
`{`
`  case 1:  // cas 1`
 	
`  break;`
 	
`  case 2:  // cas 2`
 	
`  break;`
 	
`  default: // cas par défaut`
`}`
	`  cmp i, 1`
	`  je  case1`
	
	`  cmp i, 2`
	`  je  case2`
	
	`  jmp casedefault`
	
	`case1:`
	`  // cas 1`
	
	`  jmp finswitch`
	
	`case2:`
	`  // cas 2`
	
	`  jmp finswitch`
	
	`casedefault:`
	`  // cas par défaut`
	
	`finswitch:`

Les opérateurs conditionnels :

Dans ce tableau, on voit où se situe la priorité des opérateurs conditionnels && et || :

Classes de priorité	*Opérateurs en C*
++++++++++++++	`()`
++++++++++++	`~, -, !, *, &, (casting)`
++++++++++	`*, /, %`
+++++++++	`+, -`
+++++++++	`<<, >>`
++++++++	`<, >, <=, >=`
+++++++	`==, !=`
++++++	`&`
+++++	`^`
++++	`|`
+++	**`&&`**
++	**`||`**
+	`=`

Utilisé dans une expression booléenne dont la forme est *expr1 && expr2*, l'**opérateur conditionnel et** fonctionne ainsi :

- Quand la valeur de vérité de l'expression *expr1* est fausse alors l'expression *expr2* n'est pas évaluée, car on sait déjà à ce moment que l'expression e*xpr1 && expr2* est fausse.

- Quand l'expression *expr1* est vraie alors l'expression *expr2* est évaluée et sa valeur de vérité donne la valeur de vérité de l'expression *expr1 && expr2*.

L'opérateur conditionnel et est pratique notamment quand on souhaite tester si une valeur fait partie d'une plage de valeurs.

Par exemple, on veut effectuer un traitement uniquement quand la valeur de i fait partie de l'intervalle [2, ..., 15] :

```
int i;
```

Langage C	*Assembleur*

`if (i >= 2 && i <= 15)`
`{`
`  // contenu du if`
`}`
	`cmp  i, 2`
	`jnge finif`
	
	`cmp  i, 15`
	`jnle finif`
	
	`// contenu du if`
	
	`finif:`

Utilisé dans une expression booléenne dont la forme est *expr1* || *expr2*, l'**opérateur conditionnel ou** fonctionne ainsi :

- Quand la valeur de vérité de l'expression *expr1* est vraie alors l'expression *expr2* n'est pas évaluée, car on sait déjà à ce moment que l'expression *expr1* || *expr2* est vraie.

- Quand l'expression *expr1* est fausse alors l'expression *expr2* est évaluée et sa valeur de vérité donne la valeur de vérité de l'expression *expr1* || *expr2*.

L'opérateur conditionnel ou est pratique notamment quand on souhaite tester si une valeur ne fait pas partie d'une plage de valeurs.

Par exemple, on veut effectuer un traitement uniquement quand la valeur de d ne fait pas partie de l'intervalle [2, ..., 15] :

```
double d;
const double d2 = 2.0, d15 = 15.0;
```

Langage C	*Assembleur*

`if (d < 2 || d > 15)`
`{`
`  // contenu du if`
`}`
	`  movsd  xmm0, d`
	`  comisd xmm0, d2`
	`  jb  contenuif`
	
	`  comisd xmm0, d15`
	`  jna  finif`
	
	`contenuif:`
	`  // contenu du if`
	
	`finif:`

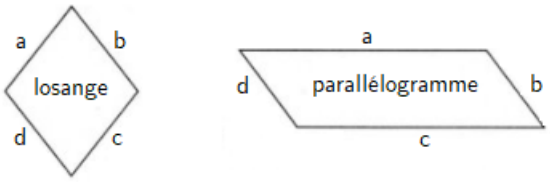
Exercices

1. Saisir les valeurs pour R et I et afficher la valeur pour U. La loi d'Ohm est : la tension U (en volts) aux bornes d'une résistance R (en ohms) est proportionnelle à l'intensité du courant électrique I (en ampères) qui la traverse.

2. Saisir le rayon R d'un cercle et afficher le périmètre de ce cercle. Formule du périmètre : P = 2 π R, où π correspond à 3.14 (nombre pi).

3. Entrer 2 valeurs et saisir l'opération à réaliser sur ces 2 valeurs parmi : 1 pour +, 2 pour -, 3 pour * et 4 pour /, puis afficher le résultat de l'opération.

4. Entrer la taille pour les côtés a, b, c et d d'un quadrilatère. Afficher si ce quadrilatère est un losange (a = b = c = d), un parallélogramme (a = c et b = d) ou une autre forme.



Toutes les valeurs saisies, manipulées et affichées sont des doubles, à l'exception de la valeur de l'opération à réaliser dans l'exercice 3 qui est un entier.