SOUS-PROGRAMME OU FONCTION

1)- Appel d'un sous-programme dans un programme principal assembleur :

1-1)- L'appel :

L'appel s'effectue par les instructions BSR ou JSR

Une sous-routine est un ensemble d'instructions exécutables, à partir d'un appel provenant de n'importe quel point dans un programme. Le programmeur n'écrit qu'une fois le code d'une sous-routine, et ce code s'exécutera à chaque endroit dans le programme où une instruction d'appel est placée. Les avantages au programmeur sont la réduction du nombre d'instructions à coder, la modularisation du code.

Pour faire appel à une sous-routine, le contrôle du programme doit se transférer du programme principal à la sous-routine. Ce transfert provoque un « changement de contexte ». En langage assembleur 68000, l'instruction d'appel sauvegarde la valeur du compteur de programme (PC) sur la pile. Puis, elle charge le PC avec l'adresse de la première instruction de la sous-routine appelée. La valeur sauvegardée sur la pile est l'adresse de la première instruction qui sera exécutée suivant le retour de la sous-routine.

1-2)- Le retour :

Le retour s'effectue par l'instruction RTS

L'instruction RTS force un branchement à l'adresse du PC entreposée sur la pile lors de l'exécution des instructions BSR ou JSR. Le PC retourne à son état précédent l'appel de la sous-routine. C'est-à-dire que le PC pointera l'instruction suivante après BSR ou JSR.

1-3)- Exemple :

```
DIODES      EQU         $E003

            org         $400
debut
            move.b      #$FF,DIODES
            bsr         tempo
            move.b      #$00,DIODES
            jsr         tempo

            trap        #15
            dc.w        0

tempo       move.w      #$FFFF,D0
bcl         subi.w      #1,D0
            bne         bcl
            rts

            end          debut
```

1-4)- Analyse du fonctionnement :

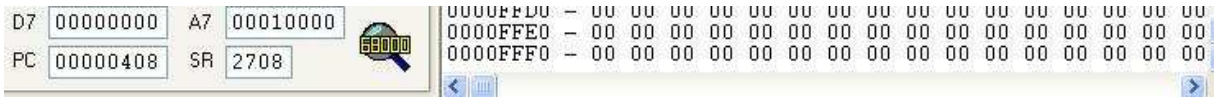
1-4-1)- Observation de la différence de codage entre les instructions BSR et JSR.

```
           0000 E003     1    DIODES      EQU         $E003
                         2
00000400                 3                org         $400
                         4    debut
00000400   13FC 00FF     5                move.b      #$FF,DIODES
00000404   0000 E003
00000408   6100 0014     6                bsr         tempo
0000040C   13FC 0000     7                move.b      #$00,DIODES
00000410   0000 E003
00000414   4EB9 0000     8                jsr         tempo
00000418   041E
                         9
0000041A   4E4F         10                trap        #15
0000041C   0000         11                dc.w        0
                        12
0000041E   303C FFFF    13    tempo       move.w      #$FFFF,D0
00000422   0440 0001    14    bcl         subi.w      #1,D0
00000426   66FA         15                bne         bcl
00000428   4E75         16                rts
                        17
           0000 0400    18                end         debut
```

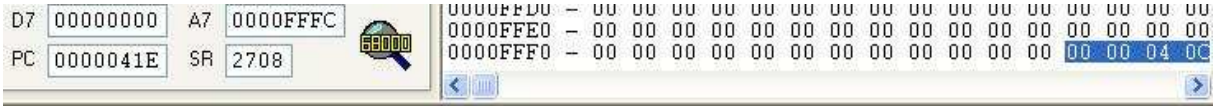
1-4-2)- Observation des mécanismes internes lors de l'exécution des instructions d'appel et de retour de sous-routine.

a)- Appel avec BSR

Contenu des registres et de la pile avant l'exécution de BSR :

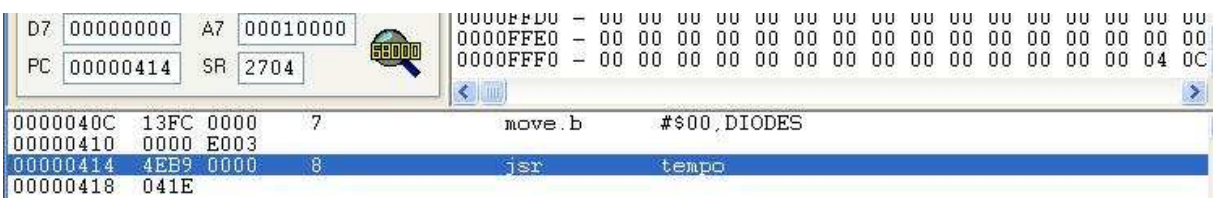


Contenu des registres et de la pile après l'exécution de BSR :

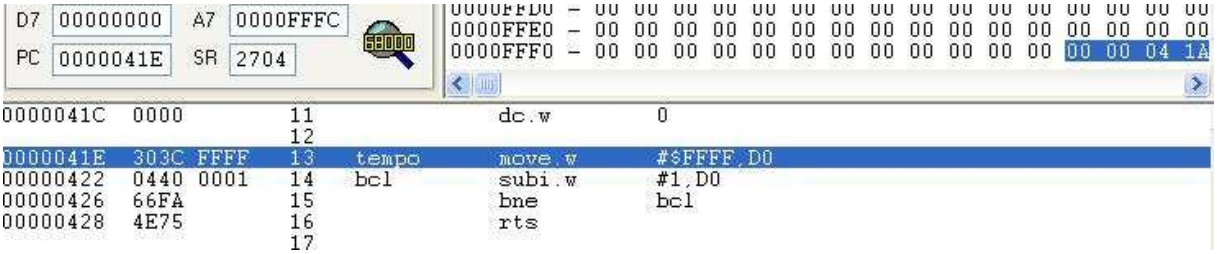


b)- Appel avec JSR

Contenu des registres et de la pile avant l'exécution de JSR :

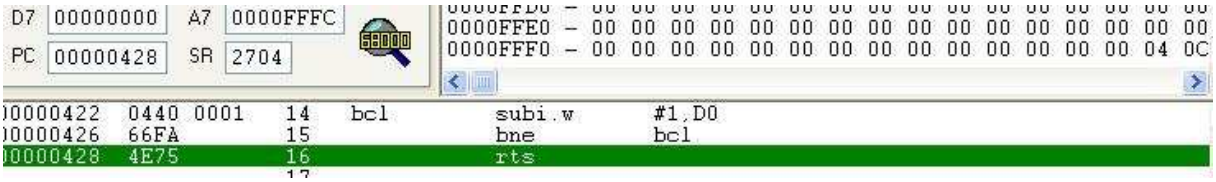


Contenu des registres et de la pile après l'exécution de JSR :

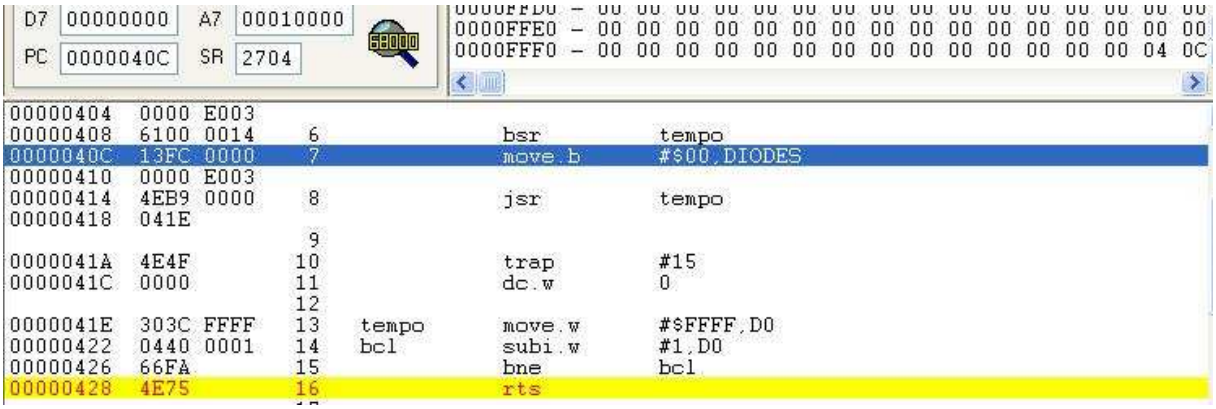


c)- Retour avec RTS

Contenu des registres et de la pile avant l'exécution de RTS :



Contenu des registres et de la pile après l'exécution de RTS :



1-5)- Sauvegarde et restauration du contexte :

Comme indiqué précédemment, l'appel de sous-routine provoque le changement de contexte. Hors le contexte n'est pas sauvegardé. Nous pouvons alors sauvegarder le contexte avant l'appel du sous-programme, et, le restaurer avant le retour au programme principal.Pour ce faire nous utilisons l'instruction MOVEM.f

Sauvegarde de tous les registres de données et d'adresses :

MOVEM.l D0-D7/A0-A7,-(SP)

Restauration de tous les registres de données et d'adresses :

MOVEM.l (SP)+,D0-D7/A0-A7

2)- Appel d'un sous-programme dans un programme principal en C : (appel de fonction)

Dans ce cas là, il nous faut créer deux fichiers. Le fichier « main.c » et le fichier pour le sous programme assembleur.

2-1)- Appel de fonction sans passage de paramètres :

a)- Fichier « main.c »

```
main()
{
      tempo();
}
```

b)- Fichier « tempo.asm »

```
_tempo      move.w      #$FFFF,D0
bcl         subi.w      #1,D0
            bne         bcl
            rts
```

2-2)- Appel de fonction sans passage de paramètres :

a)- Fichier « main.c »

```
main()
{
      tempo(2);         //passage du paramètre 2
}
```

b)- Fichier « tempo.asm »

```
_tempo      move.l      4(A7),D1     ;récupération du paramètre dans la pile
enc         move.w      #$FFFF,D0
bcl         subi.w      #1,D0
            bne         bcl
            subi.b      #1,D1
            bne         enc
            rts
```

3)- Appel d'un sous-programme (fonction) en C dans un programme principal en C :

3-1)- Appel de fonction sans passage de paramètres :

a)- Fichier « main.c »

```
main()
{
tempo();
}

void tempo()
{
unsigned int j;

for(j=0;j<0x1;j++);

}
```

3-2)- Appel de fonction avec passage de paramètres :

a)- Fichier « main.c »

```
main()
{
tempo(2);
}

void tempo(char a)
{
unsigned int j;
while(a>0)
    {
    for(j=0;j<0x1;j++);
    a--;
    }
}
```