SOLUTION EXERCICE 3-1 PIC32MX

Voici tout d'abord le programme en C.

```
void main() {

   uint8_t i;
   bool bNegatif;
   char c;
   uint8_t code;

   int8_t  a = 7;
   int16_t b = 321;
   int32_t  d = 0x12345678;

   // addition 8 bits
   a = a + 0x31;

   // addition 16 bits
   b = b + 0x123;

   // addition 32 bits
  d = d + 0x10000;

   // If then else
   if (a >= 0) {
      bNegatif = 0;
   } else {
      bNegatif = 1;
   }

   // boucle for et switch
   for (i=0; i<6 ; i++) {

      switch (i) {
         case 1 : code = 'A' ; break;
         case 2 : code = 'B' ; break;
         case 3 : code = 'C' ; break;
         default : code = 'Z' ; break;
      }
   }

   // boucle while
   i = 10;
   while (i > 0 ) {
       i--;
   }
}
```

Voici le listing assembleur obtenu avec des commentaires permettant de comprendre les actions réalisées.

☝Attention : les adresses mémoire ne sont pas forcément dans l'ordre !

```
Disassembly Listing for Ex3_1
Generated From:
C:/Users/zfpchr/Documents/ETML_ES/EtCoursHW/SL229_MINF/CoursPI
C32/ProjExercices/Ex3_1.X/dist/default/production/Ex3_1.X.prod
uction.elf
15 nov. 2016 15:19:42

c:/users/zfpchr/documents/etml_es/etcourshw/sl229_minf/courspi
c32/projexercices/ex3_1.x/main_ex3_1.c
---------------------------
1:          #include <stddef.h>          // Defines NULL
2:          #include <stdbool.h>         // Defines true
3:          #include <stdlib.h>          // Defines EXIT_FAILURE
4:          #include <stdint.h>          // Defines EXIT_FAILURE
5:
6:                      void main() {
9D000000  27BDFFE8    ADDIU SP, SP, -24
9D000004  AFBE0014    SW S8, 20(SP)
9D000008  03A0F021    ADDU S8, SP, ZERO
                                        // S8 correspond à FP
7:
8:                         uint8_t i;
9:                         bool bNegatif;
10:                        char c;
11:                        uint8_t code;
12:
13:                        int8_t  a = 7;
9D00000C  24020007    ADDIU V0, ZERO, 7 // reg V0 = 7
9D000010  A3C20001    SB V0, 1(S8)      // 8 bits => stack
14:                        int16_t b = 321;
9D000014  24020141    ADDIU V0, ZERO, 321    // reg V0 = 321
9D000018  A7C20002    SH V0, 2(S8)      // 16 bits => stack
15:                        int32_t  d = 0x12345678;
9D00001C  3C021234    LUI V0, 4660
// V0 = 4660 << 16 => 0x1234'0000
9D000020  34425678    ORI V0, V0, 22136
// V0 = 0x1234'0000 | 0x0000'5678 => 0x1234'5678
9D000024  AFC20004    SW V0, 4(S8)      // 32 bits => stack
16:
17:                        // addition 8 bits
18:                        a = a + 0x31;
9D000028  93C20001    LBU V0, 1(S8)    // a (stack) 8 bits => V0
9D00002C  24420031    ADDIU V0, V0, 49  // V0 = V0 + 49
9D000030  304200FF    ANDI V0, V0, 255  // V0 = V0 & 0xFF
// traitement au cas ou le résultat déborde les 8bits
9D000034  A3C20001    SB V0, 1(S8)    // VO => a (stack) 8 bits
19:
```

```
20:                     // addition 16 bits
21:                     b = b + 0x123;
9D000038  97C20002    LHU V0, 2(S8)  // b (stack) 16 bits => V0
9D00003C  24420123    ADDIU V0, V0, 291       // V0 = V0 + 291
9D000040  3042FFFF    ANDI V0, V0, -1    // V0 = V0 & 0xFFFF
// avec le ANDI la constante est 16 bits. Donc -1 => 0xFFFF
9D000044  A7C20002    SH V0, 2(S8) // VO => b (stack) 16 bits
22:
23:                     // addition 32 bits
24:                   d = d + 0x10000;
9D000048  8FC30004    LW V1, 4(S8)  // V1 = d (stack) 32 bits
9D00004C  3C020001    LUI V0, 1  // V0 = 1 << 16 => 0x0001'0000
9D000050  00621021    ADDU V0, V1, V0  // V0 = V0 + V1 {V1 = d}
9D000054  AFC20004    SW V0, 4(S8)   // VO => d (stack) 32 bits
25:
26:                     // If then else
27:                     if (a >= 0) {
9D000058  83C20001    LB V0, 1(S8)  // V0 = a (stack) 8 bits
9D00005C  04400004    BLTZ V0, 0x9D000070     //Si V0 < 0 saut
9D000060  00000000    NOP
28:                         bNegatif = 0;
9D000064  A3C00008    SB ZERO, 8(S8)     // 0 => bNegatif
9D000068  0B40001E    J 0x9D000078       // saut en 0x9D000078
9D00006C  00000000    NOP
29:                     } else {
30:                       bNegatif = 1;
9D000070  24020001    ADDIU V0, ZERO, 1       // V0 = 1
9D000074  A3C20008    SB V0, 8(S8)            // 1 => bNegatif
31:                     }
32:
33:                     // boucle for et switch
34:                     for (i=0; i<6 ; i++) {
9D000078  A3C00000    SB ZERO, 0(S8)     // i = 0 (8 bits)
9D00007C  0B40003C    J 0x9D0000F0       // saut en 0x9D0000F0
9D000080  00000000    NOP

// On vient ICI lorsque l'on sort des cases:
// Correspond à l'action i++
// Si V0 NE 0 saut en 0x9D000084 (début switch)
9D0000F8  1440FFE2    BNE V0, ZERO, 0x9D000084
9D0000FC  00000000    NOP

9D0000E8  93C20000    LBU V0, 0(S8)      // V0 = i
9D0000EC  24420001    ADDIU V0, V0, 1    // V0 = VO + 1
9D0000F0  A3C20000    SB V0, 0(S8)       // i = V0

9D0000F4  93C20000    LBU V0, 0(S8)      // V0 = i
```

SLTIU RD, RS, CONST16	RD = (RS∅ < CONST16∅) ? 1 : 0

```
// Si V0 < 6  V0 = 1 autrement = 0
9D0000F8  2C420006   SLTIU V0, V0, 6
9D0000FC  1440FFE1   BNE V0, ZERO, 0x9D000084
9D000100  00000000   NOP
35:
36:                        switch (i) {
9D000084  93C20000   LBU V0, 0(S8)          // V0 = i
9D000088  24030002   ADDIU V1, ZERO, 2      // V1 = 2
// Si V0 = V1 = 2 saut en 0x9D0000BC
9D00008C  1043000B   BEQ V0, V1, 0x9D0000BC
9D000090  00000000   NOP
9D000094  24030003   ADDIU V1, ZERO, 3
// Si V0 = V1 = 3 saut en 0x9D0000CC
9D000098  1043000C   BEQ V0, V1, 0x9D0000CC
9D00009C  00000000   NOP
9D0000A0  24030001   ADDIU V1, ZERO, 1
// Si V0 != 1 saut en 0x9D0000DC (default)
9D0000A4  1443000D   BNE V0, V1, 0x9D0000DC
9D0000A8  00000000   NOP
37:                            case 1 : code = 'A' ; break;
9D0000AC  24020041   ADDIU V0, ZERO, 65     // 65 = 'A'
9D0000B0  A3C20009   SB V0, 9(S8)           // code = 'A'
9D0000B4  0B400039   J 0x9D0000E4           // saut à i++
9D0000B8  00000000   NOP
38:                            case 2 : code = 'B' ; break;
9D0000BC  24020042   ADDIU V0, ZERO, 66     // 66 = 'B'
9D0000C0  A3C20009   SB V0, 9(S8)           // code = 'B'
9D0000C4  0B400039   J 0x9D0000E4           // saut à i++
9D0000C8  00000000   NOP
39:                            case 3 : code = 'C' ; break;
9D0000CC  24020043   ADDIU V0, ZERO, 67     // 67 = 'C'
9D0000D0  A3C20009   SB V0, 9(S8)           // code = 'C'
9D0000D4  0B400039   J 0x9D0000E4           // saut à i++
9D0000D8  00000000   NOP
40:                            default : code = 'Z'; ; break;
9D0000DC  2402005A   ADDIU V0, ZERO, 90     // 90 = 'Z'
9D0000E0  A3C20009   SB V0, 9(S8)           // code = 'Z'
9D0000E4  00000000   NOP

41:                          }
42:                       }
43:
```

```
44:                        // boucle while
45:                        i=10;
9D000104  2402000A    ADDIU V0, ZERO, 10
9D000108  A3C20000    SB V0, 0(S8)
46:                        while (i > 0 ) {
9D00010C  0B400048    J 0x9D000120        // saut en 0x9D000120
9D00010C  00000000    NOP

9D000120  93C20000    LBU V0, 0(S8)       // i => V0
// Si V0 != 0 saut en 0x9D000114
9D000124  1440FFFB    BNE V0, ZERO, 0x9D000114
9D000128  00000000    NOP
// attention l'adresse suivante est 9D00012C (Retour du main)


47:                             i--;
9D000114  93C20000    LBU V0, 0(S8)       // i => V0
9D000118  2442FFFF    ADDIU V0, V0, -1    // V0 = V0 -1
9D00011C  A3C20000    SB V0, 0(S8)        // V0 => i
// attention l'adresse suivante est 9D000120 (test i > 0)

48:                        }
49:                    }

// Retour du main
9D00012C  03C0E821    ADDU SP, S8, ZERO
9D000130  8FBE0014    LW S8, 20(SP)
9D000134  27BD0018    ADDIU SP, SP, 24
9D000138  03E00008    JR RA
9D00013C  00000000    NOP
```