```
II) Si alors sinon

Register int x, y;

if (x==y)
  {
  x = x + 2;
  y = y - 1;
  }
else
  {
  y = y - 3;
  x = x + 5;
  }
x = x + y;

@ Convention de stockage
@ r1 : x
@ r2 : y

Traduction ARM avec branche alors en tÃªte

compar:       cmp    r1, r2         @ if (x != y) goto sinon
              bne    sinon          @ brancher si Z=0 (resultat != 0)
alors:        add    r1, r1, #2
              sub    r2, r2, #1
              bal    fin_si         @ branch always (ou  b fin)
sinon:        sub    r2, r2, #3
              add    r1, r1, #5
fin_si:       add    r1, r1, r2


Traduction ARM avec la branche sinon en tÃªte

compar:       cmp    r1, r2         @ if ((x-y) == 0) goto alors
              beq    alors          @
sinon:        sub    r2, r2, #3
              add    r1, r1, #5
              b      fin_si         @ goto finsi
alors:        add    r1, r1, #2
              sub    r2, r2, #1
fin_si:       add    r1, r1, r2


Traduction d'un si sans alors

if (x==y)
  {
  x = x + 2;
  y = y - 1;
  }
x = x + y;


compar:       cmp    r1, r2         @ if (x != y) goto finsi
              bne    finsi
alors:        add    r1, r1, #2
              sub    r2, r2, #1
fin_si:       add    r1, r1, r2
```



```
III) Do ... while

do
  {
  x = x + 2;
  y = y + 1;
  }
while (x < y);
x = x + y;

Traduction de la forme if goto en langage d'assemblage

corps:        add    r1, r1, #2     @ corps:  x = x + 2
              add    r2, r2, #1     @         y = y + 1
compar:       cmp    r1, r2         @ if (x < y) goto corps
              blt    corps
fin_do:       add    r1, r1, r2     @ fin:    x = x + y


Si x et y de type entier naturel (unsigned int) : utiliser blo ou son
synonyme bcc au lieu de blt.

IV) While

Avec test aprÃ¨s le corps

while (x < y)                 @          goto test
  {
  x = x + 2;                  @ corps:   x = x + 2
  y = y + 1;                  @          y = y - 1
  }                           @ test:    if (x < y) goto corps
x = x + y;                    @ fin:     x = x + y

@ idem do _while, avec un saut au test devant le corps
              bal      test         @ goto test
corps:        add    r1, r1, #2     @ corps:  x = x + 2
              add    r2, r2, #1     @         y = y + 1
test:         cmp    r1, r2         @ if (x < y) goto corps
              blt    corps
fin_do:       add    r1, r1, r2     @ fin_do: x = x + y



Avec test avant le corps

while (x < y)                 @ test:    if (x >= y) goto fin
  {
  x = x + 2;                  @ corps:   x = x + 2
  y = y + 1;                  @          y = y - 1
  }                           @          goto test
x = x + y;                    @ fin:     x = x + y


test:         cmp    r1, r2         @ if (x >= y) goto fin
              bge    fin
corps:        add    r1, r1, #2     @ corps:  x = x + 2
              add    r2, r2, #1     @         y = y + 1
              b      test           @         goto test
fin_do:       add    r1, r1, r2     @ fin:    x = x + y

@ Si x et y de type entier naturel (unsigned int) , utiliser bhs ou son
synonyme bcs au lieu de bge.
```

```
V) Parcourant

           for (indice = 1; indice <= n; indice ++)
           {
           facto = facto * i;
           }

==> transformation en tant que

        indice = 1;        /* initialisation */
        while (indice <= n)
           {
           facto = facto*i;  /* corps du for */
           indice ++;        /* mise a jour  */
           }

Traduction en langage d'assemblage, test aprÃ¨s le corps.

@ convention de stockage
@ indice r0
@ facto  r1
@ n      r3
@ temp   r2   stockage des poids forts du rÃ©sultat de la multiplication
@             smull retourne un resultat de 64 bits --> dans deux registres

init:    mov    r0, #0           @ indice = 0
         b      test             @ goto test
corps:   smull  r2, r1, r1, r0   @ facto *= indice
         add    r0, r0, #1       @ indice ++
test:    cmp    r0, r3           @ if (indice <= n) goto corps
         ble    corps            @ utiliser bls si indice et n unsigned
suite:                           @ instructions suivant la boucle for

VI) Conditions composÃ©es

if ((a==b) || (c > 1))
  x = y;
else
  y = x;
x++;

Traduction en langage d'assemblage

@ Convention de stockage
@ a r0
@ b r1
@ c r2
@ x r5
@ y r6

         cmp  r0, r1       @          if (a==b) goto alors
         beq  alors
         cmp  r2, #1       @          if (c <= 1) goto sinon
         ble  sinon

alors:   mov  r5, r6       @  alors:  x = y
         b    finsi        @          goto  finsi
sinon:   mov r6, r5        @  sinon:  y = x
finsi:   add r5, r5, #1    @  finsi:  x++
```



```
VII) Boucles imbriquÃ©es

register int i,j,r;

for (i=4; i != 0; i--)
  {
  r ++;
  for (j=0; j != 4; j++)
    {
    r = r*8;
    r = r+j;
    }
  r = r-2;
  }


Tranformation en boucles while

   i = 4:
   while (i != 0)
     {
     r++;             // debut corps for i
     j = 0;
     while (j != 4)
       {
       r = r*8;         // debut coprs for j
       r = r+j;         // fin   corps for j

       j++;             // maj j
       }
     r = r-2;         // fin corps for i

     i--;             // maj i
     }

Tranformation en if .. goto de la boucle for i

          i = 4;
          goto test_i;
corps_i:     r++;
             // inserer ici la traduction de la boucle for j
             r = r-2;
             i --;
test_i:      if (i != 0) goto corps_i;


Transformation en if .. goto de la boucle for j

             j = 0;
             goto test_j;
corps_j:        r = r*8;
                r = r+j;
                j++;
test_j:      if (j!= 4) goto corps_j;


Les deux ensemble :

          @ i : r0, j : r1, r : r2

          .text
```

```
          mov r0,#4             @ i = 4;
          b  test_i             @ goto test_i
corps_i:  add r2,r2,#1          @   r++
          mov r1, #0            @   j = 0
          b  test_j             @   goto test_j
corps_j:  mov r2, r2, LSL #3    @      r = r*8
          add r2, r2, r1        @      r = r+j
          add r1, r1, #1        @      j++
test_j:   cmp r1, #4            @      if (j != 4) goto corps_j
          bne  corps_j
          sub r2, r2, #2        @   r = r -2
          sub r0,r0, #1         @   i--
test_i:   cmp r0,#0             @   if (i !=0 ) goto corps_i
          bne corps_i


Variante optimisÃ©e :
         mov r0, #5
         b test_i
         ...
test_i:  subS r0, r0, #1
         bne corps_i
```