Rapport – Etape #1 Projet Compilation

Guillaume IMPELLIZZERI – N° étudiant : 31804343

Table des matières

1. Exemple 1 3
1.1. Résultat attendu 3
1.2. Table des symboles 3
1.3. Arbre abstrait généré 3
1.4. Code Assembleur « beta » généré 3
1.5. Analyse 4
1.5.1. BSIM 4
1.5.2. Conclusion 4
2. Exemple 2 4
2.1. Résultat attendu 4
2.2. Table des symboles 4
2.3. Arbre abstrait généré 4
2.4. Code Assembleur « beta » généré 4
2.5. Analyse 5
2.5.1. BSIM 5
2.5.2. Conclusion 5
3. Exemple 3 6
3.1. Résultat attendu 6
3.2. Table des symboles 6
3.3. Arbre abstrait généré 6
3.4. Code Assembleur « beta » généré 6
3.5. Analyse 7
3.5.1. BSIM 7
3.5.2. Conclusion 7
4. Exemple 4 8
4.1. Résultat attendu 8
4.2. Table des symboles 8
4.3. Arbre abstrait généré 8
4.4. Code Assembleur « beta » généré 8
4.5. Analyse 9
4.5.1. BSIM 9
4.5.2. Conclusion 9

5. Exemple 5 10
5.1. Résultat attendu 10
5.2. Table des symboles 10
5.3. Arbre abstrait généré 10
5.4. Code Assembleur « beta » généré 10
5.5. Analyse 11
5.5.1. BSIM 11
5.5.2. Conclusion 12
6. Exemple 6 12
6.1. Résultat attendu 12
6.2. Table des symboles 12
6.3. Arbre abstrait généré 13
6.4. Code Assembleur « beta » généré 13
6.5. Analyse 14
6.5.1. BSIM 14
6.5.2. Conclusion 14
7. Exemple 7 15
7.1. Résultat attendu 15
7.2. Table des symboles 15
7.3. Arbre abstrait généré 15
7.4. Code Assembleur « beta » généré 15
7.5. Analyse 17
7.5.1. BSIM 17
7.5.2. Conclusion 17
8. Exemple 8 17
8.1. Résultat attendu 17
8.2. Table des symboles 17
8.3. Arbre abstrait généré 18
8.4. Code Assembleur « beta » généré 18
8.5. Analyse 20
8.5.1. BSIM 20
8.5.2. Conclusion 20

1. Exemple 1

1.1. Résultat attendu

L'exemple 1 consiste à la création d'une fonction main qui est la fonction principale de notre programme. Celle-ci ne renvoie rien, n'a aucun paramètres et aucune variables locales. Le résultat attendu est donc un arbre composé seul d'une racine « PROG » suivi de son fils « FONCTION/main ».

1.2. Table des symboles

```
==========TDS générée==========
{nom= main; type= void; cat= fonction; nbparam= 0; nbloc= 0}
```

1.3. Arbre abstrait généré

```
==========Arbre généré==========
PROG
└─FONCTION/main
```

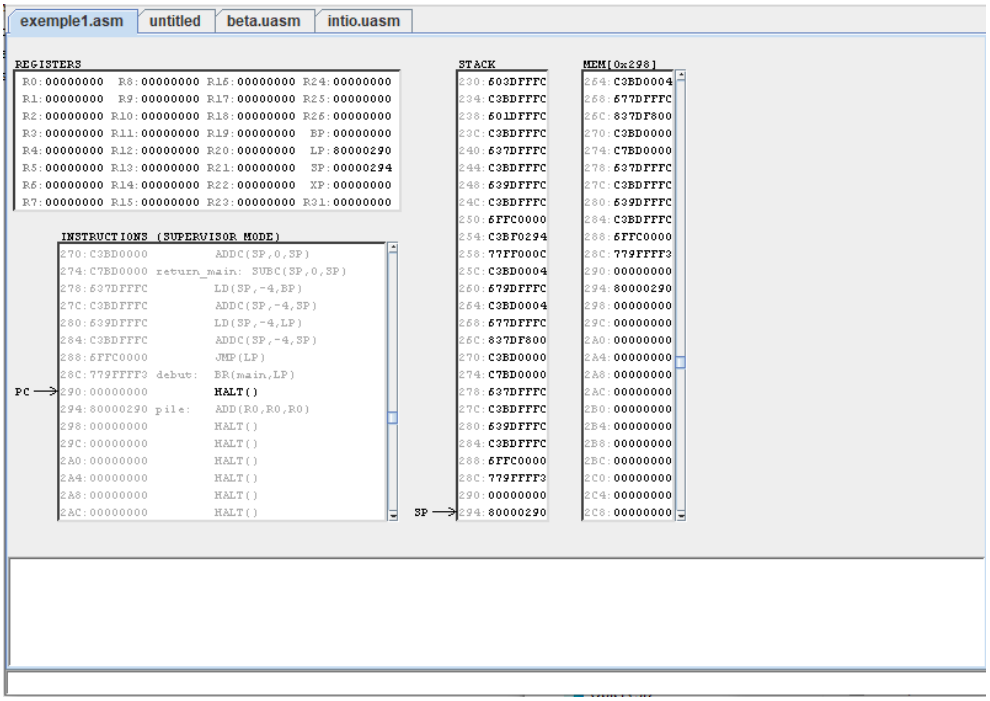
1.4. Code Assembleur « beta » généré

==========Code ASM généré==========

```
.include beta.uasm
.include intio.uasm
.options tty

        CMOVE(pile,SP)
        BR(debut)
main:
        PUSH(LP)
        PUSH(BP)
        MOVE(SP,BP)
        ALLOCATE(0)
return_main:
        DEALLOCATE(0)
        POP(BP)
        POP(LP)
        RTN()
debut:
        CALL(main)
        HALT()
pile:
```

1.5. Analyse

1.5.1. BSIM



1.5.2. Conclusion

Au vu des résultats sur BSIM, il n'y a aucun problème en particulier, du fait principal qu'il y a très peu de choses dans cet arbre abstrait ainsi que dans la TDS.

2. Exemple 2

2.1. Résultat attendu

Le résultat attendu est le même que l'exemple précédent mais génère en plus 4 variables avec i initialisé à 10 et j initialisé à 20.

2.2. Table des symboles

```
==========TDS générée==========
{nom= main; type= void; cat= fonction; nbparam= 0; nbloc= 0}
{nom= main; type= void; cat= fonction; nbparam= 0; nbloc= 0}
{nom= i; type= int; cat= global; val= 10; rang= 0; scope= null}
{nom= j; type= int; cat= global; val= 20; rang= 0; scope= null}
{nom= k; type= int; cat= global; val= 0; rang= 0; scope= null}
{nom= l; type= int; cat= global; val= 0; rang= 0; scope= null}
```

2.3. Arbre abstrait généré

```
==========Arbre généré==========
PROG
└─FONCTION/main
```

2.4. Code Assembleur « beta » généré

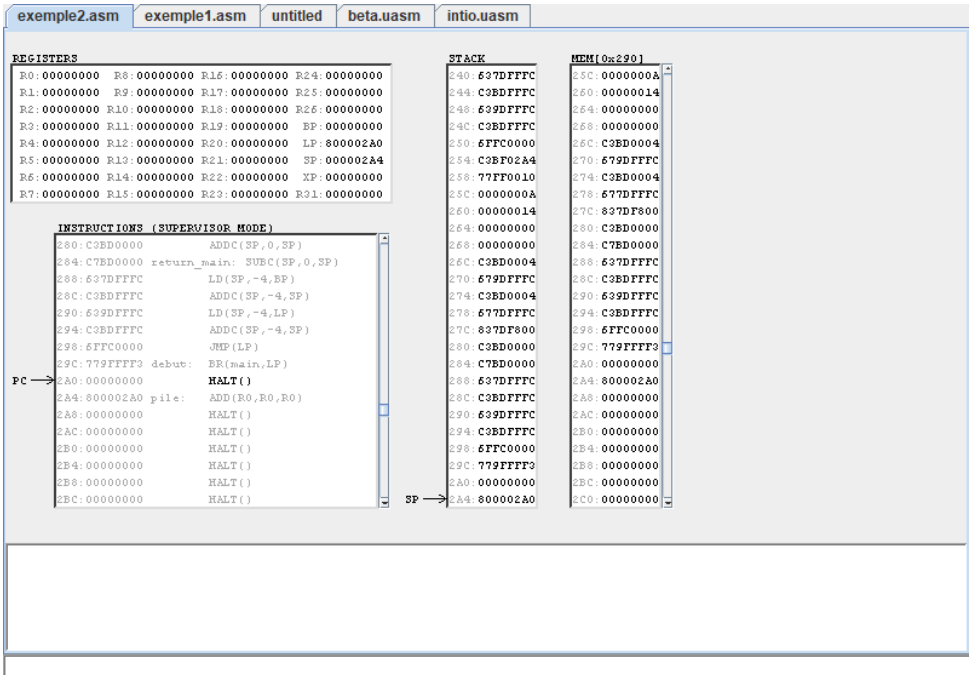
==========Code ASM généré==========

```
.include beta.uasm
.include intio.uasm
.options tty
```

```
        CMOVE(pile,SP)
        BR(debut)
i:      LONG(10)
j:      LONG(20)
k:      LONG(0)
l:      LONG(0)
main:
        PUSH(LP)
        PUSH(BP)
        MOVE(SP,BP)
        ALLOCATE(0)
return_main:
        DEALLOCATE(0)
        POP(BP)
        POP(LP)
        RTN()
debut:
        CALL(main)
        HALT()
pile:
```

2.5. Analyse

2.5.1. BSIM



2.5.2. Conclusion

Comme on peut le voir les variables i,j,k,l ont bien été créées.

3. Exemple 3

3.1. Résultat attendu

K étant initialisé à 2. On calcule l tel que l = i+(3*j) (sachant que i=10 et j=20) ➔ l=70

3.2. Table des symboles

```
==========TDS générée==========
{nom= main; type= void; cat= fonction; nbparam= 0; nbloc= 0}
{nom= i; type= int; cat= global; val= 10; rang= 0; scope= null}
{nom= j; type= int; cat= global; val= 20; rang= 0; scope= null}
{nom= k; type= int; cat= global; val= 0; rang= 0; scope= null}
{nom= l; type= int; cat= global; val= 0; rang= 0; scope= null}
```

3.3. Arbre abstrait généré

```
==========Arbre généré==========
PROG
└FONCTION/main
  └AFF
    └IDF/k
    └CONST/2
  └AFF
    └IDF/l
    └PLUS
      └IDF/i
      └MUL
        └CONST/3
        └IDF/j
```

3.4. Code Assembleur « beta » généré

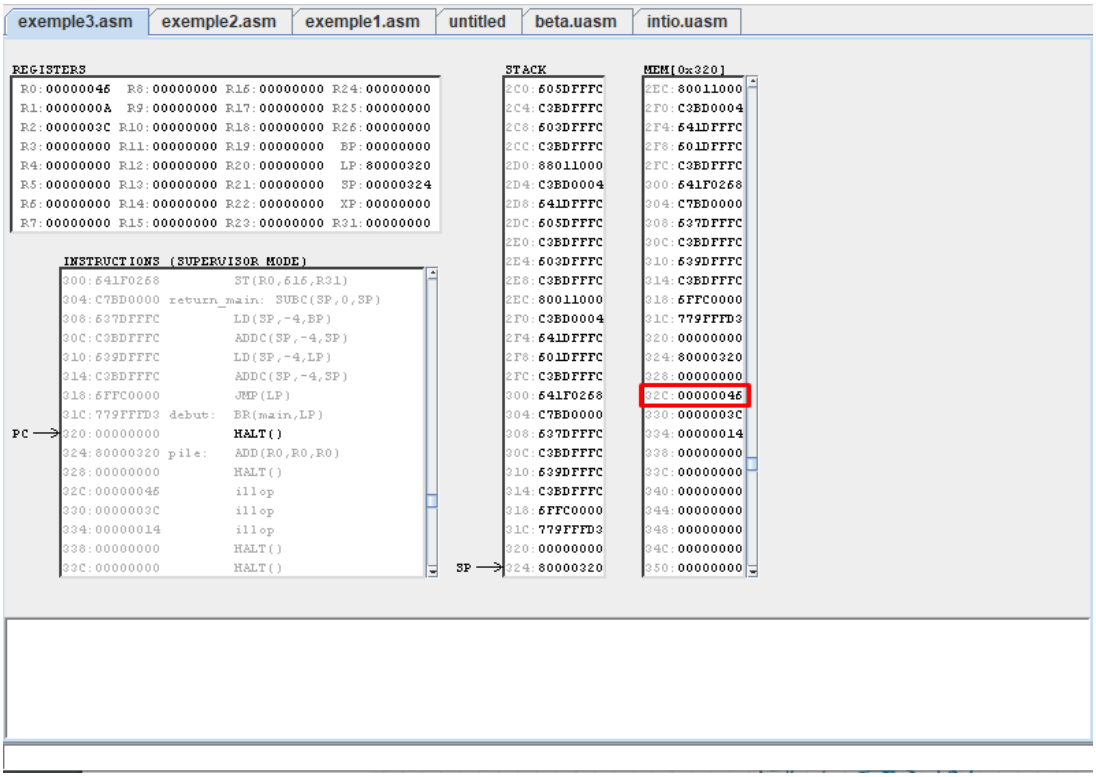
==========Code ASM généré==========

```
.include beta.uasm
.include intio.uasm
.options tty

        CMOVE(pile,SP)
        BR(debut)
i:      LONG(10)
j:      LONG(20)
k:      LONG(0)
l:      LONG(0)

main:
        PUSH(LP)
        PUSH(BP)
        MOVE(SP,BP)
        ALLOCATE(0)
        CMOVE(2,R0)
        PUSH(R0)
        POP(R0)
        ST(R0,k)
        LD(i,R0)
        PUSH(R0)
        CMOVE(3,R0)
```

```
        PUSH(R0)
        LD(j,R0)
        PUSH(R0)
        POP(R2)
        POP(R1)
        MUL(R1,R2,R0)
        PUSH(R0)
        POP(R2)
        POP(R1)
        ADD(R1,R2,R0)
        PUSH(R0)
        POP(R0)
        ST(R0,l)
return_main:
        DEALLOCATE(0)
        POP(BP)
        POP(LP)
        RTN()
debut:
        CALL(main)
        HALT()
pile:
```

3.5. Analyse

3.5.1. BSIM



3.5.2. Conclusion

Comme on peut le voir, on obtient bien à la fin, le résultat : 70 (qui était le résultat attendu)

4. Exemple 4

4.1. Résultat attendu

L'objectif de cet exemple est de prendre en entrée un entier venu de l'utilisateur (dans notre cas on prend en entrée i=10). Puis, on fait la somme entre i et j, ce qui doit donner un résultat de 30.

4.2. Table des symboles

```
==========TDS générée==========
{nom= main; type= void; cat= fonction; nbparam= 0; nbloc= 0}
{nom= i; type= int; cat= global; val= 0; rang= 0; scope= null}
{nom= j; type= int; cat= global; val= 20; rang= 0; scope= null}
```

4.3. Arbre abstrait généré

```
==========Arbre généré==========
PROG
└FONCTION/main
  └AFF
    └IDF/i
    └LIRE
  └ECR
    └PLUS
      └IDF/i
      └IDF/j
```

4.4. Code Assembleur « beta » généré

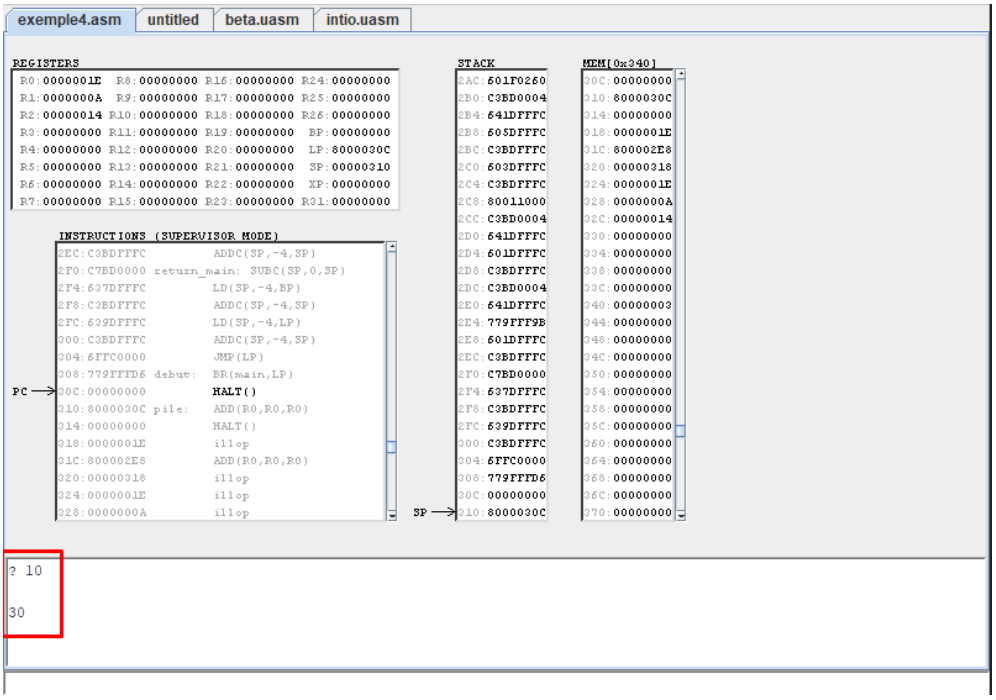
==========Code ASM généré==========

```
.include beta.uasm
.include intio.uasm
.options tty

    CMOVE(pile,SP)
    BR(debut)
i:   LONG(0)
j:   LONG(20)
main:
    PUSH(LP)
    PUSH(BP)
    MOVE(SP,BP)
    ALLOCATE(0)
    RDINT()
    PUSH(R0)
    POP(R0)
    ST(R0,i)
    LD(i,R0)
    PUSH(R0)
    LD(j,R0)
    PUSH(R0)
    POP(R2)
    POP(R1)
```

```
    ADD(R1,R2,R0)
    PUSH(R0)
    POP(R0)
    WRINT()
return_main:
    DEALLOCATE(0)
    POP(BP)
    POP(LP)
    RTN()
debut:
    CALL(main)
    HALT()
pile:
```

4.5. Analyse

4.5.1. BSIM



4.5.2. Conclusion

Comme on peut le voir on obtient bien le résultat attendu quand on est en prend en entrée i=10.

5. Exemple 5

5.1. Résultat attendu

En fonction de ce que l'on va mettre en entrée sur la variable i, cela va afficher 1 si c'est supérieur à 10 et 2 si c'est inférieur.

5.2. Table des symboles

```
==========TDS générée==========
{nom= main; type= void; cat= fonction; nbparam= 0; nbloc= 0}
{nom= i; type= int; cat= global; val= 0; rang= 0; scope= null}
```

5.3. Arbre abstrait généré

```
=========Arbre généré==========
PROG
└FONCTION/main
  └AFF
    └IDF/i
    └LIRE
  └SI/1
    └SUP
      └IDF/i
      └CONST/10
    └BLOC
      └ECR
        └CONST/1
    └BLOC
      └ECR
        └CONST/2
```

5.4. Code Assembleur « beta » généré

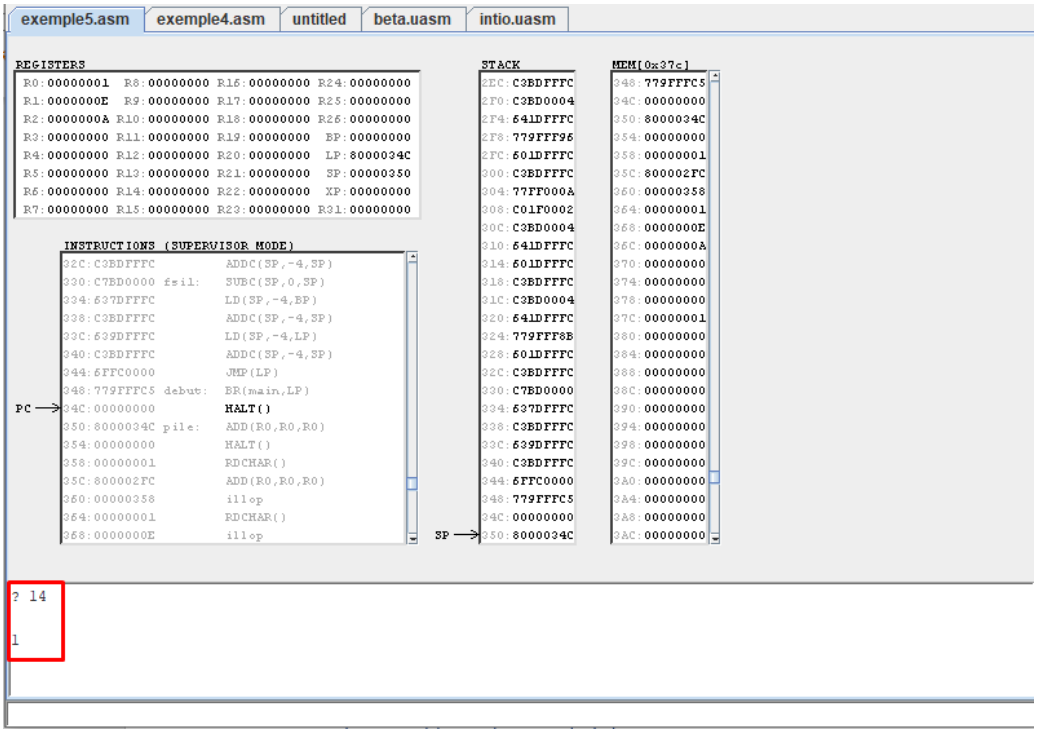
==========Code ASM généré==========

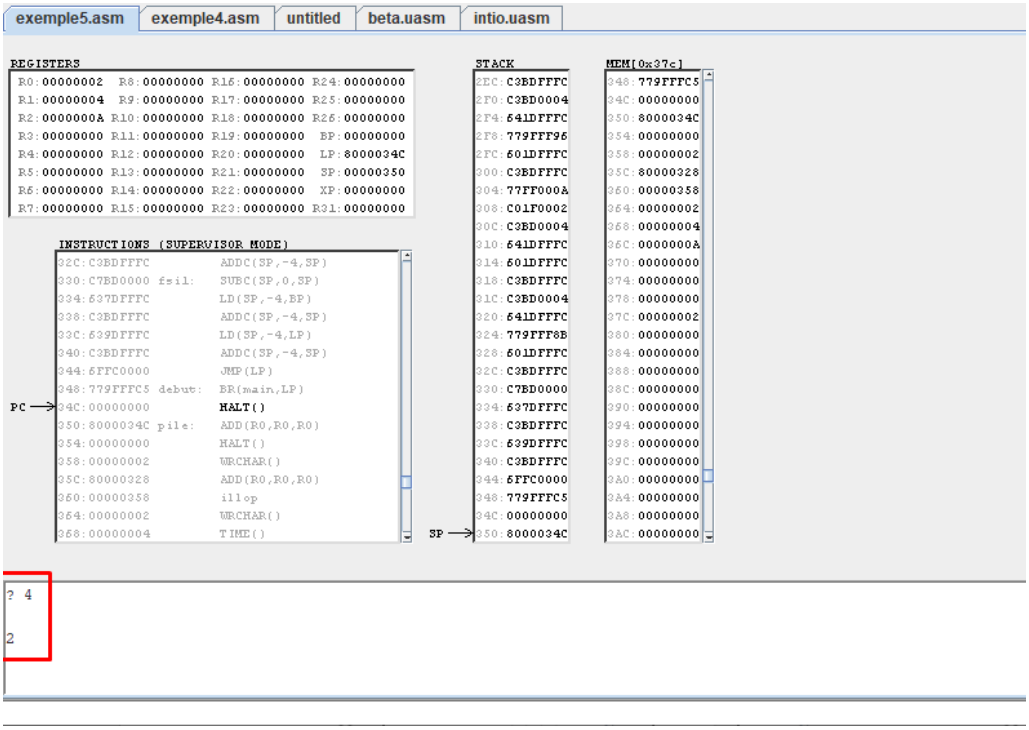
```
.include beta.uasm
.include intio.uasm
.options tty

    CMOVE(pile,SP)
    BR(debut)
i:   LONG(0)
main:
    PUSH(LP)
    PUSH(BP)
    MOVE(SP,BP)
    ALLOCATE(0)
    RDINT()
    PUSH(R0)
    POP(R0)
    ST(R0,i)
    LD(i,R0)
    PUSH(R0)
    CMOVE(10,R0)
    PUSH(R0)
    POP(R2)
    POP(R1)
```

```
    CMPLT(R2,R1,R0)
    PUSH(R0)
    POP(R0)
    BF(R0,sinon1)
    CMOVE(1,R0)
    PUSH(R0)
    POP(R0)
    WRINT()
    BR(fsi1)
sinon1:
    CMOVE(2,R0)
    PUSH(R0)
    POP(R0)
    WRINT()
fsi1:
return_main:
    DEALLOCATE(0)
    POP(BP)
    POP(LP)
    RTN()
debut:
    CALL(main)
    HALT()
pile:
```

5.5. Analyse

5.5.1. BSIM





5.5.2. Conclusion

Comme le montre les deux captures d'écran ci-dessus, le code assembleur généré fonctionne parfaitement.

6. Exemple 6

6.1. Résultat attendu

Cet exemple a pour but de faire tester la boucle « tant que ». Le résultat attendu est donc une suite d'entiers consécutifs de 0 à 4 inclus.

6.2. Table des symboles

```
==========TDS générée==========
{nom= main; type= void; cat= fonction; nbparam= 0; nbloc= 0}
{nom= i; type= int; cat= global; val= 0; rang= 0; scope= null}
{nom= n; type= int; cat= global; val= 5; rang= 0; scope= null}
```

6.3. Arbre abstrait généré

```
==========Arbre généré==========
PROG
└FONCTION/main
  └AFF
    └IDF/i
    └CONST/0
  └TQ/1
    └INF
      └IDF/i
      └IDF/n
    └BLOC
      └ECR
        └IDF/i
      └AFF
        └IDF/i
        └PLUS
          └IDF/i
          └CONST/1
```

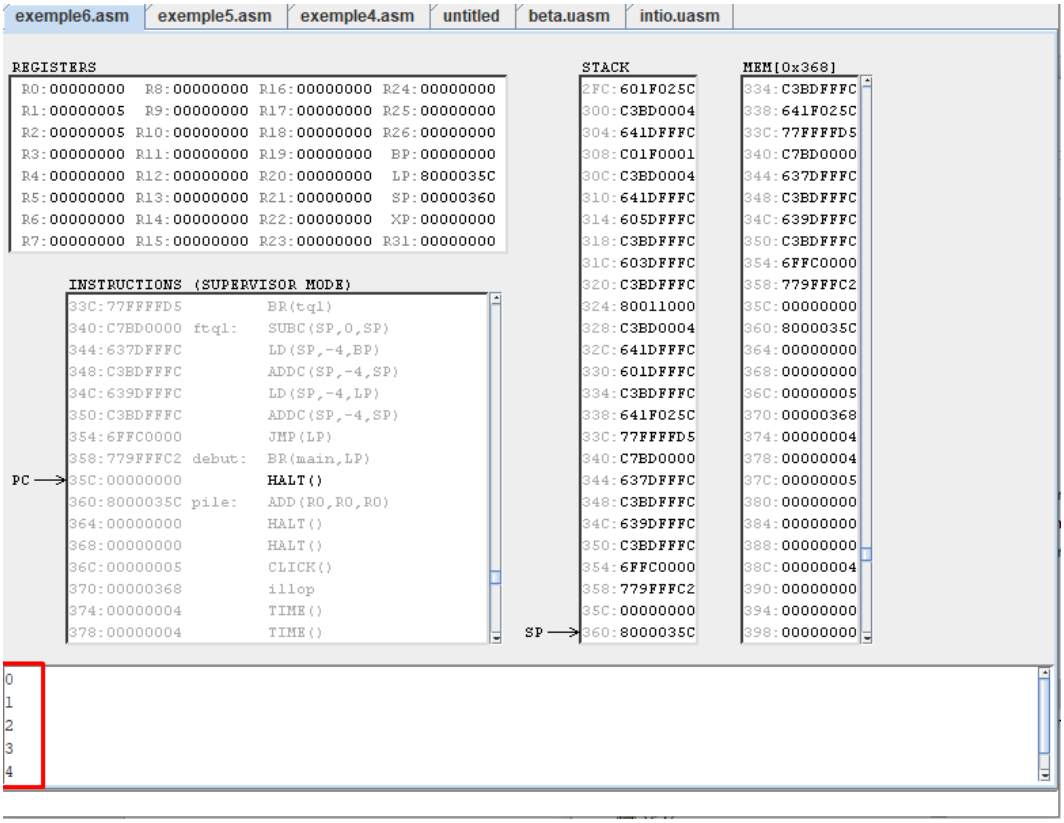
6.4. Code Assembleur « beta » généré

```
==========Code ASM généré==========
.include beta.uasm
.include intio.uasm
.options tty

      CMOVE(pile,SP)
      BR(debut)
i:    LONG(0)
n:    LONG(5)
main:
      PUSH(LP)
      PUSH(BP)
      MOVE(SP,BP)
      ALLOCATE(0)
      CMOVE(0,R0)
      PUSH(R0)
      POP(R0)
      ST(R0,i)
tq1:
      LD(i,R0)
      PUSH(R0)
      LD(n,R0)
      PUSH(R0)
      POP(R2)
      POP(R1)
      CMPLT(R1,R2,R0)
      PUSH(R0)
      POP(R0)
      BF(R0,ftq1)
      LD(i,R0)
      PUSH(R0)
      POP(R0)
      WRINT()
```

```
    LD(i,R0)
    PUSH(R0)
    CMOVE(1,R0)
    PUSH(R0)
    POP(R2)
    POP(R1)
    ADD(R1,R2,R0)
    PUSH(R0)
    POP(R0)
    ST(R0,i)
    BR(tq1)
ftq1:
return_main:
    DEALLOCATE(0)
    POP(BP)
    POP(LP)
    RTN()
debut:
    CALL(main)
    HALT()
pile:
```

6.5. Analyse

6.5.1. BSIM



6.5.2. Conclusion

Comme on peut le voir, la suite de nombres entiers de 0 à 4 inclus est bien générée.

7. Exemple 7

7.1. Résultat attendu

Le résultat attendu de cet exemple 7 est normalement : 5

7.2. Table des symboles

```
==========TDS générée==========
{nom= main; type= void; cat= fonction; nbparam= 0; nbloc= 0}
{nom= a; type= int; cat= global; val= 10; rang= 0; scope= null}
{nom= f; type= void; cat= fonction; nbparam= 1; nbloc= 2}
{nom= i; type= int; cat= param; val= 0; rang= 0; scope= f}
{nom= x; type= int; cat= local; val= 0; rang= 0; scope= f}
{nom= y; type= int; cat= local; val= 0; rang= 1; scope= f}
```

7.3. Arbre abstrait généré

```
==========Arbre généré==========
PROG
└FONCTION/f
  └AFF
    └IDF/x
    └CONST/1
  └AFF
    └IDF/y
    └CONST/1
  └AFF
    └IDF/a
    └PLUS
      └IDF/i
      └PLUS
        └IDF/x
        └IDF/y
└FONCTION/main
  └APPEL/f
    └CONST/3
  └ECR
    └IDF/a
```

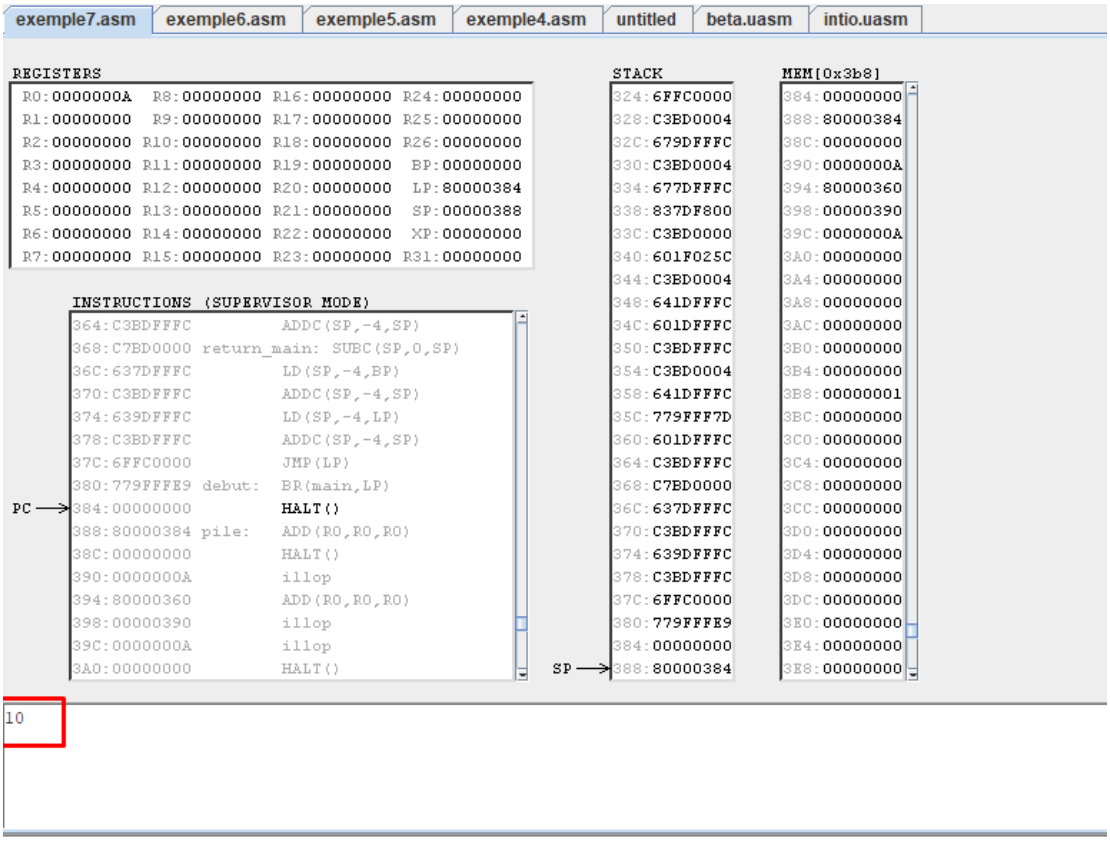
7.4. Code Assembleur « beta » généré

```
==========Code ASM généré==========
.include beta.uasm
.include intio.uasm
.options tty

    CMOVE(pile,SP)
    BR(debut)
a:  LONG(10)
f:
    PUSH(LP)
    PUSH(BP)
    MOVE(SP,BP)
    ALLOCATE(2)
    CMOVE(1,R0)
    PUSH(R0)
    POP(R0)
    PUTFRAME(R0,0)
    CMOVE(1,R0)
```

```
        PUSH(R0)
        POP(R0)
        PUTFRAME(R0,4)
        GETFRAME(-12,R0)
        PUSH(R0)
        GETFRAME(0,R0)
        PUSH(R0)
        GETFRAME(4,R0)
        PUSH(R0)
        POP(R2)
        POP(R1)
        ADD(R1,R2,R0)
        PUSH(R0)
        POP(R2)
        POP(R1)
        ADD(R1,R2,R0)
        PUSH(R0)
        POP(R0)
        ST(R0,a)
return_f:
        DEALLOCATE(2)
        POP(BP)
        POP(LP)
        RTN()
main:
        PUSH(LP)
        PUSH(BP)
        MOVE(SP,BP)
        ALLOCATE(0)
        LD(a,R0)
        PUSH(R0)
        POP(R0)
        WRINT()
return_main:
        DEALLOCATE(0)
        POP(BP)
        POP(LP)
        RTN()
debut:
        CALL(main)
        HALT()
pile:
```

7.5. Analyse

7.5.1. BSIM



7.5.2. Conclusion

Comme on peut le voir sur la capture d'écran ci-dessus, le résultat affiché n'est pas celui attendu. Effectivement, cela peut être dû au fait que la fonction « f » ne retourne pas de valeur et donc l'information que « a » a changé n'est pas « arrivé » au « main » et donc reste à 10, tel que « a » est initialisé dès le début.

8. Exemple 8

8.1. Résultat attendu

Le résultat attendu de l'exemple 8 est : 13. Effectivement, nous appelons la fonction f qui va, dans un premier temps, additionner les 2 entiers passés en paramètre (1 et 2). Cela va donner x=3. La fonction f retourne x+10 donc 13.
Ensuite, dans le main, on initialise a=f(1,2), donc a=13 puis on l'affiche.

8.2. Table des symboles

```
==========TDS générée==========
{nom= main; type= void; cat= fonction; nbparam= 0; nbloc= 0}
{nom= a; type= int; cat= global; val= 0; rang= 0; scope= null}
{nom= f; type= int; cat= fonction; nbparam= 2; nbloc= 1}
{nom= x; type= int; cat= local; val= 0; rang= 0; scope= f}
{nom= i; type= int; cat= param; val= 0; rang= 0; scope= f}
{nom= j; type= int; cat= param; val= 0; rang= 1; scope= f}
```

8.3. Arbre abstrait généré

```
==========Arbre généré==========
PROG
└FONCTION/f
  └AFF
    └IDF/x
    └PLUS
      └IDF/i
      └IDF/j
  └RET/f
    └PLUS
      └IDF/x
      └CONST/10
└FONCTION/main
  └AFF
    └IDF/a
    └APPEL/f
      └CONST/1
      └CONST/2
  └ECR
    └IDF/a
```

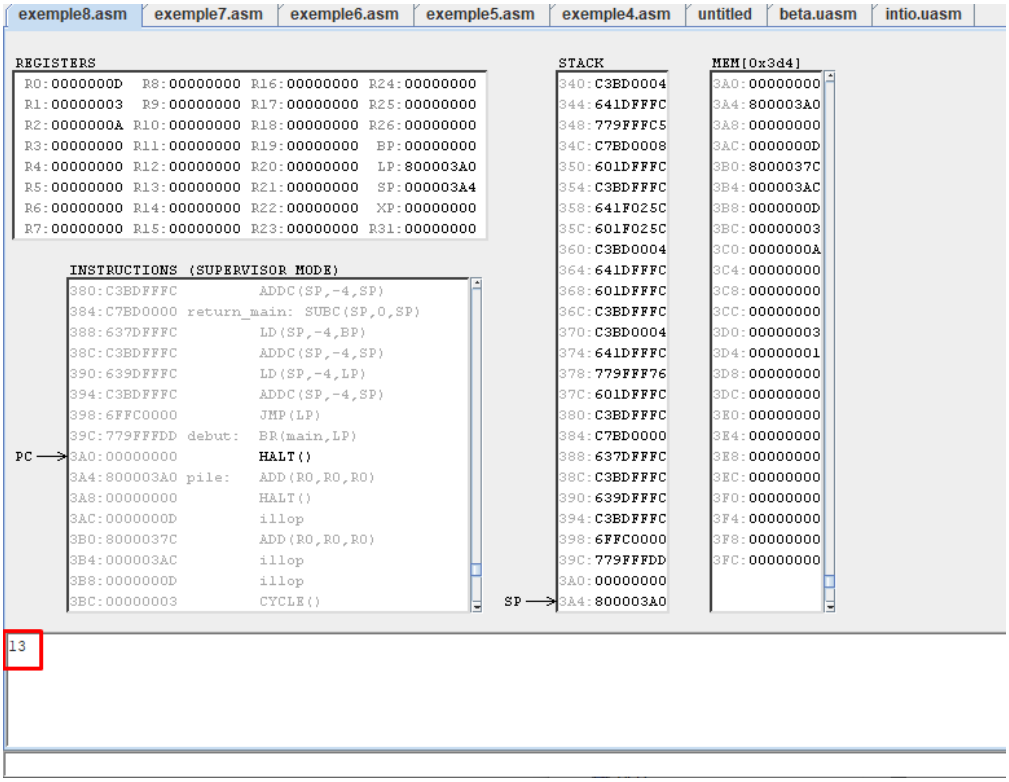
8.4. Code Assembleur « beta » généré

```
==========Code ASM généré==========
.include beta.uasm
.include intio.uasm
.options tty

    CMOVE(pile,SP)
    BR(debut)
a:  LONG(0)
f:
    PUSH(LP)
    PUSH(BP)
    MOVE(SP,BP)
    ALLOCATE(1)
    GETFRAME(-16,R0)
    PUSH(R0)
    GETFRAME(-12,R0)
    PUSH(R0)
    POP(R2)
    POP(R1)
    ADD(R1,R2,R0)
    PUSH(R0)
    POP(R0)
    PUTFRAME(R0,0)
    GETFRAME(0,R0)
    PUSH(R0)
    CMOVE(10,R0)
    PUSH(R0)
    POP(R2)
    POP(R1)
    ADD(R1,R2,R0)
    PUSH(R0)
```

```
        POP(R0)
        PUTFRAME(R0,-20)
        BR(return_f)
return_f:
        DEALLOCATE(1)
        POP(BP)
        POP(LP)
        RTN()
main:
        PUSH(LP)
        PUSH(BP)
        MOVE(SP,BP)
        ALLOCATE(0)
        ALLOCATE(1)
        CMOVE(1,R0)
        PUSH(R0)
        CMOVE(2,R0)
        PUSH(R0)
        CALL(f)
        DEALLOCATE(2)
        POP(R0)
        ST(R0,a)
        LD(a,R0)
        PUSH(R0)
        POP(R0)
        WRINT()
return_main:
        DEALLOCATE(0)
        POP(BP)
        POP(LP)
        RTN()
debut:
        CALL(main)
        HALT()
pile:
```

8.5. Analyse

8.5.1. BSIM



8.5.2. Conclusion

Comme on peut le voir, le résultat affiché sur BSIM est bien celui attendu. Il n'y a donc pas de problèmes au niveau de la génération.