

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

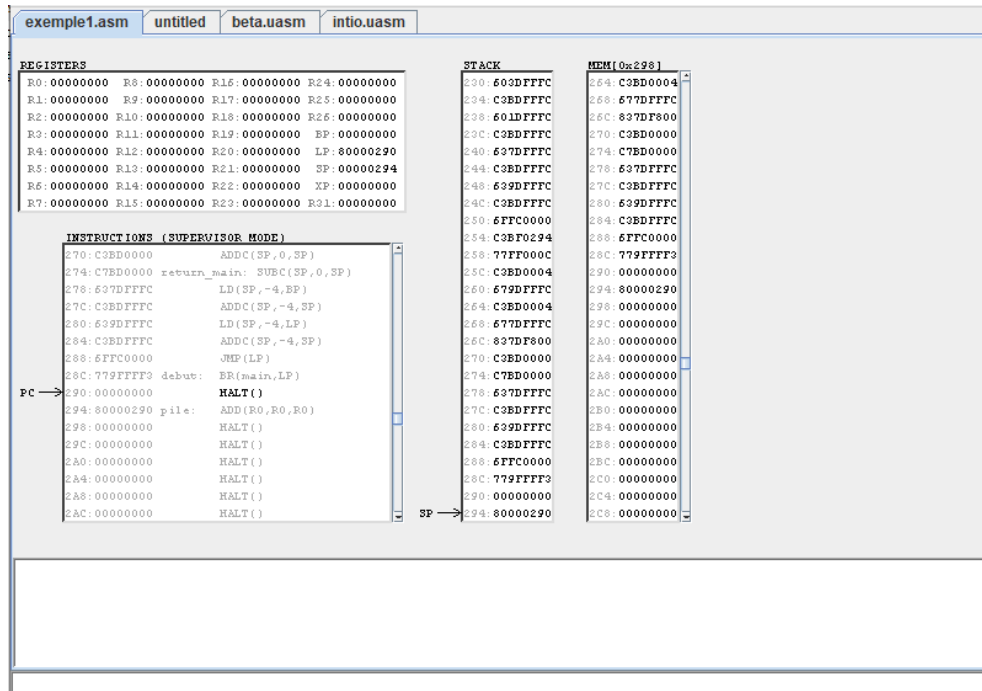
```
=====Code ASM généré=====
```

```
.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

The screenshot displays the BSIM (Beta Simulator) interface with the following components:

- Registers:** A table showing the state of registers R0 through R31. R0-R7 are initialized to 0x00000000. R8-R15 are initialized to 0x00000000. R16-R23 are initialized to 0x00000000. R24-R31 are initialized to 0x00000000. The stack pointer (SP) is at 0x800002A0.
- Instructions (SUPERVISOR MODE):** A list of instructions being executed. The current instruction is `HALT()` at address 0x2A0. The previous instruction was `ADD(R0,R0,R0)` at address 0x2A4.
- Stack:** A table showing the stack contents. The stack grows downwards from 0x2A0 to 0x2C0. The current stack pointer (SP) is at 0x800002A0.
- Memory (0x290):** A table showing the memory contents. The memory is initialized to 0x00000000.

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,I)
return_main:
DEALLOCATE(0)
POP(BP)
POP(LP)
RTN()
debut:
CALL(main)
HALT()
pile:

```

3.5. Analyse

3.5.1. BSIM

The screenshot displays the BSIM interface with the following components:

- Registers:** A table showing the state of 16 registers (R0-R15). R0 contains 00000046, R1 contains 0000000A, and R2 contains 0000000C. Other registers are zero or contain specific values like 80000320 for LP and 00000324 for SP.
- Instructions (SUPERVISOR MODE):** A list of assembly instructions with their addresses. The instruction at address 320 is `HALT()`, which is the current instruction being executed. The instruction at address 324 is `ADD(R0,R0,R0)`.
- Stack:** A table showing the stack contents. The stack pointer (SP) is at address 324, pointing to the value 80000320. The stack grows downwards from higher addresses to lower addresses.
- Memory (0x320):** A table showing the contents of memory starting at address 0x320. The value at address 320 is 00000046, which is highlighted in red.

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)
```


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

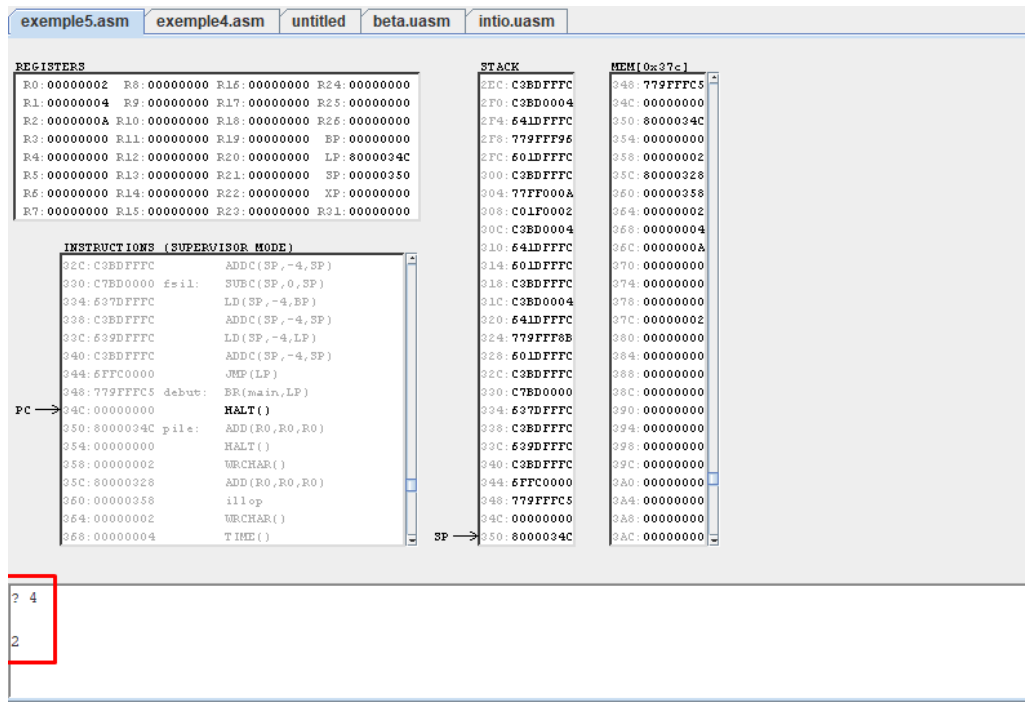
```
=====Arbre généré=====
PROG
├─ FONCTION/main
│   └─ AFF
│       ├── IDF/i
│       ├── LIRE
│       └─ SI/1
│           ├── SUP
│           │   ├── IDF/i
│           │   └─ CONST/10
│           └─ BLOC
│               ├── ECR
│               │   └─ CONST/1
│               └─ BLOC
│               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

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

The screenshot displays the BSIM (Basic Simulator) interface with the following components:

- Registers:** A table showing the state of 32 registers (R0-R31). R0-R7 contain zeros, while R8-R31 contain various hexadecimal values.
- Instructions (SUPERVISOR MODE):** A list of instructions with their addresses and comments. The PC (Program Counter) is currently at address 35C:00000000, pointing to the `HALT()` instruction.
- Stack:** A table showing the stack contents, with addresses ranging from 2FC:601F025C to 360:8000035C. The SP (Stack Pointer) is currently at address 360:8000035C.
- Memory (0x368):** A table showing memory contents, with addresses ranging from 334:C3EDFFFC to 398:00000000.
- PC (Program Counter):** A red box highlights the PC value 0, indicating the current instruction address.

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

The screenshot shows the BSIM debugger interface with the following components:

- Registers:** A table showing the state of 32 registers (R0-R31). R0-R7 are 00000000, R8-R15 are 00000000, R16-R23 are 00000000, R24-R31 are 00000000. R25-R31 are 00000000.
- Instructions (SUPERVISOR MODE):** A list of instructions with their addresses. The PC (Program Counter) is at 384:00000000, pointing to the instruction `HALT()`. The instructions are:
 - 364: C3BDFFFC `ADD(C, SP, -4, SP)`
 - 368: C7BD0000 `return_main: SUBC(SP, 0, SP)`
 - 36C: 637DFFFC `LD(SP, -4, BP)`
 - 370: C3BDFFFC `ADD(C, SP, -4, SP)`
 - 374: 639DFFFC `LD(SP, -4, LP)`
 - 378: C3BDFFFC `ADD(C, SP, -4, SP)`
 - 37C: 6FFC0000 `JMP(LP)`
 - 380: 779FFFE9 `debut: BR(main, LP)`
 - 384: 00000000 `HALT()`
 - 388: 80000384 `pile: ADD(R0, R0, R0)`
 - 38C: 00000000 `HALT()`
 - 390: 0000000A `illop`
 - 394: 80000360 `ADD(R0, R0, R0)`
 - 398: 00000390 `illop`
 - 39C: 0000000A `illop`
 - 3A0: 00000000 `HALT()`
- Stack:** A table showing the stack contents. The SP (Stack Pointer) is at 388:80000384. The stack contains:
 - 324: 6FFC0000
 - 328: C3BD0004
 - 32C: 679DFFFC
 - 330: C3BD0004
 - 334: 677DFFFC
 - 338: 837DF800
 - 33C: C3BD0000
 - 340: 601F025C
 - 344: C3BD0004
 - 348: 641DFFFC
 - 34C: 601DFFFC
 - 350: C3BDFFFC
 - 354: C3BD0004
 - 358: 641DFFFC
 - 35C: 779FFF7D
 - 360: 601DFFFC
 - 364: C3BDFFFC
 - 368: C7BD0000
 - 36C: 637DFFFC
 - 370: C3BDFFFC
 - 374: 639DFFFC
 - 378: C3BDFFFC
 - 37C: 6FFC0000
 - 380: 779FFFE9
 - 384: 00000000
 - 388: 80000384
- MEM[0x3b8]:** A table showing memory contents. The memory contains:
 - 384: 00000000
 - 388: 80000384
 - 38C: 00000000
 - 390: 0000000A
 - 394: 80000360
 - 398: 00000390
 - 39C: 0000000A
 - 3A0: 00000000
 - 3A4: 00000000
 - 3A8: 00000000
 - 3AC: 00000000
 - 3B0: 00000000
 - 3B4: 00000000
 - 3B8: 00000001
 - 3BC: 00000000
 - 3C0: 00000000
 - 3C4: 00000000
 - 3C8: 00000000
 - 3CC: 00000000
 - 3D0: 00000000
 - 3D4: 00000000
 - 3D8: 00000000
 - 3DC: 00000000
 - 3E0: 00000000
 - 3E4: 00000000
 - 3E8: 00000000

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

The screenshot shows the BSIM window with the following sections:

- Registers:** A table showing the state of 16 registers (R0-R15). R0-R7 are all 00000000. R8-R15 contain various values, including 00000000, 0000000A, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000.
- Instructions (Supervisor Mode):** A list of assembly instructions with their addresses. The PC (Program Counter) is at 3A0:00000000, pointing to the instruction `HALT()`. The instructions include `ADDC(SP,-4,SP)`, `SUBC(SP,0,SP)`, `LD(SP,-4,BP)`, `ADDC(SP,-4,SP)`, `LD(SP,-4,LP)`, `ADDC(SP,-4,SP)`, `JMP(LP)`, `BR(main,LP)`, `HALT()`, `ADD(R0,R0,R0)`, `HALT()`, `illop`, `ADD(R0,R0,R0)`, `illop`, `illop`, and `CYCLE()`.
- Stack:** A list of stack addresses and their contents. The stack grows downwards from 340:00000004 to 3A4:800003A0. The contents include `C3BD0004`, `641DFFFC`, `779FFFC5`, `C7ED0008`, `601DFFFC`, `C3BDFFFC`, `641F025C`, `601F025C`, `C3BD0004`, `641DFFFC`, `601DFFFC`, `C3BDFFFC`, `C3BD0004`, `641DFFFC`, `601DFFFC`, `779FFFC5`, `601DFFFC`, `6FFC0000`, `779FFFC5`, `00000000`, and `800003A0`.
- Memory (0x3A4):** A list of memory addresses and their contents. The memory starts at 3A0:00000000 and ends at 3FC:00000000. The contents are mostly 00000000, with some non-zero values at 3A4:800003A0, 3A8:00000000, 3AC:00000000, 3B0:8000037C, 3B4:00000000, 3B8:00000000, and 3BC:00000003.

The PC (Program Counter) is highlighted in red, indicating the current instruction being executed.

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.