

# RAPPORT PROJET ASSEMBLEUR – KELBERT PAUL

---

## Table des matières

---

RAPPORT PROJET ASSEMBLEUR – KELBERT PAUL.....	1
Exemple 1 .....	3
But .....	3
Table des symboles .....	3
Arbre.....	3
Code assembleur .....	3
Analyse .....	4
Exemple 2 .....	4
But .....	4
Table des symboles .....	4
Arbre.....	4
Code assembleur .....	5
Analyse .....	5
Exemple 3 .....	6
But .....	6
Table des symboles .....	6
Arbre.....	6
Code assembleur .....	6
Analyse .....	8
Exemple 4 .....	8
But .....	8
Table des symboles .....	8
Arbre.....	9
Code assembleur .....	9
Analyse .....	9
Exemple 5 .....	10
But .....	10
Table des symboles .....	10
Arbre.....	11

Code assembleur .....	11
Analyse .....	12
Exemple 6 .....	13
But .....	13
Table des symboles .....	13
Arbre.....	14
Code assembleur .....	14
Analyse .....	16
Exemple 7 .....	16
But .....	16
Table des symboles .....	16
Arbre.....	17
Code assembleur .....	17
Analyse .....	18
Exemple 8 .....	19
But .....	19
Table des symboles .....	19
Arbre.....	19
Code assembleur .....	19
Analyse .....	21

## Exemple 1

### But

Le but de cet exemple est de montrer le bon fonctionnement de notre compilateur, car il ne présente aucune variable ni fonction. Le résultat attendu est donc une table des symboles très petite, un arbre composé de deux nœuds et un code ASM ne renvoyant rien.

### Table des symboles

```
{nom=main, type=void, cat=fonction}
```

### Arbre

```
PROG  
└─FUNCTION/main
```

### Code assembleur

```
.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:
```

## Analyse

### BSim

REGISTERS															
R0: 00000000	R8: 00000000	R16: 00000000	R24: 00000000												
R1: 00000000	R9: 00000000	R17: 00000000	R25: 00000000												
R2: 00000000	R10: 00000000	R18: 00000000	R26: 00000000												
R3: 00000000	R11: 00000000	R19: 00000000	BP: 00000000												
R4: 00000000	R12: 00000000	R20: 00000000	LP: 80000290												
R5: 00000000	R13: 00000000	R21: 00000000	SP: 00000294												
R6: 00000000	R14: 00000000	R22: 00000000	XP: 00000000												
R7: 00000000	R15: 00000000	R23: 00000000	R31: 00000000												

  

INSTRUCTIONS (SUPERVISOR MODE)															
270: C3BD0000	ADDC(SP, 0, SP)														
274: C7BD0000	return_main: SUBC(SP, 0, SP)														
278: 637DFFFC	LD(SP, -4, BP)														
27C: C3BDFFFC	ADDC(SP, -4, SP)														
280: 639DFFFC	LD(SP, -4, LP)														
284: C3BDFFFC	ADDC(SP, -4, SP)														
288: 6FFC0000	JMP(LP)														
28C: 779FFFF3	debut: BR(main, LP)														
PC → 290: 00000000	HALT()														
294: 80000290	pile: ADD(R0, R0, R0)														
298: 00000000	HALT()														
29C: 00000000	HALT()														
2A0: 00000000	HALT()														
2A4: 00000000	HALT()														
2A8: 00000000	HALT()														
2AC: 00000000	HALT()														

  

STACK															
230: 603DFFFC															
234: C3BDFFFC															
238: 601DFFFC															
23C: C3BDFFFC															
240: 637DFFFC															
244: C3BDFFFC															
248: 639DFFFC															
24C: C3BDFFFC															
250: 6FFC0000															
254: C3BF0294															
258: 77FF000C															
25C: C3BD0004															
260: 679DFFFC															
264: C3BD0004															
268: 677DFFFC															
26C: 837DF800															
270: C3BD0000															
274: C7BD0000															
278: 637DFFFC															
27C: C3BDFFFC															
280: 639DFFFC															
284: C3BDFFFC															
288: 6FFC0000															
28C: 779FFFF3															
290: 00000000															
294: 80000290															

  

MEM[0x298]															
264: C3BD0004															
268: 677DFFFC															
26C: 837DF800															
270: C3BD0000															
274: C7BD0000															
278: 637DFFFC															
27C: C3BDFFFC															
280: 639DFFFC															
284: C3BDFFFC															
288: 6FFC0000															
28C: 779FFFF3															
290: 00000000															
294: 80000290															

### Fonctionnement

Le retour de BSim ne présente aucuns problèmes, ce qui paraît logique au vu du but de l'exemple.

## Exemple 2

### But

Le but de ce programme est le même que l'exemple précédent, mais avec quatre variables.

### Table des symboles

```
{nom=main, type=void, cat=fonction}
{nom=i, type=int, cat=global, val=10}
{nom=j, type=int, cat=global, val=20}
{nom=k, type=int, cat=global}
{nom=l, type=int, cat=global}
```

### Arbre

```
PROG
└─FONCTION/main
```

## Code assembleur

```
.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:
```

## Analyse

### BSim

The screenshot displays the BSIM debugger interface with four main panels:

- REGISTERS:** A table showing the state of 32 registers (R0-R31). R0-R7 are all 00000000. R8-R15 are 00000000. R16-R23 are 00000000. R24-R31 are 00000000. R28 is 800002A0. R29 is 000002A4. R30 is 00000000. R31 is 00000000.
- INSTRUCTIONS (SUPERVISOR MODE):** A list of instructions with their addresses and assembly code. The PC (Program Counter) is at 2A0:00000000, pointing to the instruction `HALT()`. The instructions are:
  - 280: C3BD0000 `ADD(C(SP,0,SP)`
  - 284: C7BD0000 `return_main: SUBC(SP,0,SP)`
  - 288: 637DFFFC `LD(SP,-4,BP)`
  - 28C: C3BDFFFC `ADD(C(SP,-4,SP)`
  - 290: 639DFFFC `LD(SP,-4,LP)`
  - 294: C3BDFFFC `ADD(C(SP,-4,SP)`
  - 298: 6FFC0000 `JMP(LP)`
  - 29C: 779FFFF3 `debut: BR(main,LP)`
  - 2A0: 00000000 `HALT()`
  - 2A4: 800002A0 `pile: ADD(R0,R0,R0)`
  - 2A8: 00000000 `HALT()`
  - 2AC: 00000000 `HALT()`
  - 2B0: 00000000 `HALT()`
  - 2B4: 00000000 `HALT()`
  - 2B8: 00000000 `HALT()`
  - 2BC: 00000000 `HALT()`
- STACK:** A table showing the stack contents. The SP (Stack Pointer) is at 2A4:800002A0. The stack grows downwards from higher addresses to lower addresses. The contents are:
  - 240: 637DFFFC
  - 244: C3BDFFFC
  - 248: 639DFFFC
  - 24C: C3BDFFFC
  - 250: 6FFC0000
  - 254: C3BF02A4
  - 258: 77FF0010
  - 25C: 0000000A
  - 260: 00000014
  - 264: 00000000
  - 268: 00000000
  - 26C: C3BD0004
  - 270: 679DFFFC
  - 274: C3BD0004
  - 278: 677DFFFC
  - 27C: 837DF800
  - 280: C3BD0000
  - 284: C7BD0000
  - 288: 637DFFFC
  - 28C: C3BDFFFC
  - 290: 639DFFFC
  - 294: C3BDFFFC
  - 298: 6FFC0000
  - 29C: 779FFFF3
  - 2A0: 00000000
  - 2A4: 800002A0
- MEM[0x2a8]:** A table showing memory contents starting from address 274. The contents are:
  - 274: C3BD0004
  - 278: 677DFFFC
  - 27C: 837DF800
  - 280: C3BD0000
  - 284: C7BD0000
  - 288: 637DFFFC
  - 28C: C3BDFFFC
  - 290: 639DFFFC
  - 294: C3BDFFFC
  - 298: 6FFC0000
  - 29C: 779FFFF3
  - 2A0: 00000000
  - 2A4: 800002A0

## Fonctionnement

Aucun problème de présent, le code généré est correct.

## Exemple 3

### But

Le but de cet exemple est un peu plus complexe que les deux précédents, car les variables sont à présent traitées.

$k = 2 / i = 10 / j = 20$

Nous voulons calculer l tel quel  $l = i + (3*j)$ . Le résultat attendu est donc 70.

## Table des symboles

```
{nom=main, type=void, cat=fonction}
{nom=i, type=int, cat=global, val=10}
{nom=j, type=int, cat=global, val=20}
{nom=k, type=int, cat=global}
{nom=l, type=int, cat=global}
```

## Arbre

```
PROG
└─FUNCTION/main
  └─AFF
    └─IDF/k
      └─CONST/2
    └─AFF
      └─IDF/l
        └─PLUS
          └─IDF/i
            └─MUL
              └─CONST/3
                └─IDF/j
```

## Code assembleur

```
.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:

## Analyse

### BSim

The screenshot displays the BSim debugger interface with four main panels:

- REGISTERS:** A table of 32 registers (R0-R31). R0 is highlighted with a red box and contains the value 00000046. Other registers like R8, R15, R24, R1, R9, R17, R25, R2, R10, R18, R26, R3, R11, R19, BP, R4, R12, R20, LP, R5, R13, R21, SP, R6, R14, R22, XP, R7, R15, R23, and R31 contain various zero or specific values.
- INSTRUCTIONS (SUPERVISOR MODE):** A list of assembly instructions with their addresses. The instruction at address 320 is `HALT()`, and the PC (Program Counter) is pointing to it. Other instructions include `ST(R0,615,R31)`, `return_main: SUBC(SP,0,SP)`, `LD(SP,-4,BP)`, `ADD(C(SP,-4,SP)`, `LD(SP,-4,LP)`, `ADD(C(SP,-4,SP)`, `JMP(LP)`, `BR(main,LP)`, `ADD(R0,R0,R0)`, `HALT()`, `illop`, and `HALT()`.
- STACK:** A list of stack addresses and their corresponding values. The stack grows downwards from higher addresses (2C0) to lower addresses (298). Values include `505DFFFC`, `C3BDFFFC`, `603DFFFC`, `88011000`, `641DFFFC`, `605DFFFC`, `80011000`, `8000000A`, `00000014`, `00000002`, `00000046`, `C3BD0004`, `679DFFFC`, `C3BD0004`, `677DFFFC`, `837DF800`, `C3BD0000`, `C01F0002`, `C3BD0004`, `641DFFFC`, `601DFFFC`, `C3BDFFFC`, and `80000320`.
- MEM[0x258]:** A memory window showing the contents of memory starting at address 0x258. The values are the same as those in the STACK panel.

### Fonctionnement

Comme le montre la valeur encadrée, le résultat est 46. Or, BSim renvoie les valeurs en hexadécimal, ce qui correspond à 70 en décimal. Donc le programme fonctionne.

## Exemple 4

### But

Le but de cet exemple est de tester la fonction de lecture et d'écriture de notre générateur ASM. Ici, nous testerons avec  $i = 15$  (en lecture) et  $j = 20$ . Le programme doit donc écrire 35.

### Table des symboles

```
{nom=main, type=void, cat=fonction}
{nom=i, type=int, cat=global}
{nom=j, type=int, cat=global, val=20}
```



## Arbre

```
PROG
├─FONCTION/main
│   └─AFF
│       └─IDF/i
│           └─LIRE
│               └─ECR
│                   └─PLUS
│                       └─IDF/i
│                           └─IDF/j
```

## Code assembleur

```
.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:
```

## Analyse

### BSim

The screenshot displays the BSim debugger interface with four main panels:

- REGISTERS:** A table showing the state of 32 registers (R0-R31). R0 contains 00000023, R1 contains 0000000F, and R2 contains 00000014. Other registers like R10, R11, R12, R13, R14, R15, R16, R17, R18, R19, R20, R21, R22, R23, R24, R25, R26, R27, R28, R29, R30, R31 are all zero.
- INSTRUCTIONS (SUPERVISOR MODE):** A list of instructions with their addresses and assembly code. The instruction at address 30C is highlighted, showing `HALT()`. The PC (Program Counter) is pointing to address 30C.
- STACK:** A table showing the stack contents. The stack pointer (SP) is pointing to address 310, which contains the value 8000030C.
- MEM[0x340]:** A table showing memory contents starting from address 30C. The value at address 310 is 8000030C.

Below the main interface, there is a small window with a red border containing the text "? 15" and "35".

### Fonctionnement

Nous pouvons voir que la valeur 35 est bien affichée. Donc cet exemple est correct !

## Exemple 5

### But

Ici nous voulons tester l'écriture de conditions. C'est pour cela que l'utilisateur va devoir entrer une valeur, et que le programme va écrire 1 si cette valeur est supérieure à 10, ou alors 2 si elle est inférieure.

### Table des symboles

```
{nom=main, type=void, cat=fonction}  
{nom=i, type=int, cat=global}
```

## Arbre

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

## Code assembleur

```
.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(fsil)
sinon1:
```

```

        CMOVE(2,R0)
        PUSH(R0)
        POP(R0)
        WRINT()

fsil:
return_main:
        DEALLOCATE(0)
        POP(BP)
        POP(LP)
        RTN()

debut:
        CALL(main)
        HALT()

pile:

```

## Analyse

### BSim

REGISTERS			
R0: 00000002	R8: 00000000	R16: 00000000	R24: 00000000
R1: 00000002	R9: 00000000	R17: 00000000	R25: 00000000
R2: 0000000A	R10: 00000000	R18: 00000000	R26: 00000000
R3: 00000000	R11: 00000000	R19: 00000000	BP: 00000000
R4: 00000000	R12: 00000000	R20: 00000000	LP: 8000034C
R5: 00000000	R13: 00000000	R21: 00000000	SP: 00000350
R6: 00000000	R14: 00000000	R22: 00000000	XP: 00000000
R7: 00000000	R15: 00000000	R23: 00000000	R31: 00000000

  

INSTRUCTIONS (SUPERVISOR MODE)	
32C: C3BDFEFC	ADDC(SP, -4, SP)
330: C7BD0000	fsil: SUBC(SP, 0, SP)
334: 637DFEFC	LD(SP, -4, BP)
338: C3BDFEFC	ADDC(SP, -4, SP)
33C: 639DFEFC	LD(SP, -4, LP)
340: C3BDFEFC	ADDC(SP, -4, SP)
344: 6FFC0000	JMP(LP)
348: 779FFFC5	debut: BR(main, LP)
PC → 34C: 00000000	HALT()
350: 8000034C	pile: ADD(R0, R0, R0)
354: 00000000	HALT()
358: 00000002	WRCHAR()
35C: 80000328	ADD(R0, R0, R0)
360: 00000358	illop
364: 00000002	WRCHAR()
368: 00000002	WRCHAR()

  

STACK	MEM[0x37c]
2EC: C3BDFEFC	348: 779FFFC5
2F0: C3BD0004	34C: 00000000
2F4: 641DFEFC	350: 8000034C
2F8: 779FFFC5	354: 00000000
2FC: 601DFEFC	358: 00000002
300: C3BDFEFC	35C: 80000328
304: 77FF000A	360: 00000358
308: C01F0002	364: 00000002
30C: C3BD0004	368: 00000002
310: 641DFEFC	36C: 0000000A
314: 601DFEFC	370: 00000000
318: C3BDFEFC	374: 00000000
31C: C3BD0004	378: 00000000
320: 641DFEFC	37C: 00000002
324: 779FFFC5	380: 00000000
328: 601DFEFC	384: 00000000
32C: C3BDFEFC	388: 00000000
330: C7BD0000	38C: 00000000
334: 637DFEFC	390: 00000000
338: C3BDFEFC	394: 00000000
33C: 639DFEFC	398: 00000000
340: C3BDFEFC	39C: 00000000
344: 6FFC0000	3A0: 00000000
348: 779FFFC5	3A4: 00000000
34C: 00000000	3A8: 00000000
SP → 350: 8000034C	3AC: 00000000

  

? 2  
2

### REGISTERS

R0: 00000001	R8: 00000000	R16: 00000000	R24: 00000000
R1: 0000000C	R9: 00000000	R17: 00000000	R25: 00000000
R2: 0000000A	R10: 00000000	R18: 00000000	R26: 00000000
R3: 00000000	R11: 00000000	R19: 00000000	BP: 00000000
R4: 00000000	R12: 00000000	R20: 00000000	LP: 8000034C
R5: 00000000	R13: 00000000	R21: 00000000	SP: 00000350
R6: 00000000	R14: 00000000	R22: 00000000	XP: 00000000
R7: 00000000	R15: 00000000	R23: 00000000	R31: 00000000

### INSTRUCTIONS (SUPERVISOR MODE)

32C: C3BDFEFC	ADDC(SP, -4, SP)
330: C7BD0000	fsil: SUBC(SP, 0, SP)
334: 637DFFFC	LD(SP, -4, BP)
338: C3BDFEFC	ADDC(SP, -4, SP)
33C: 639DFFFC	LD(SP, -4, LP)
340: C3BDFEFC	ADDC(SP, -4, SP)
344: 6FFC0000	JMP(LP)
348: 779FFFC5	debut: BR(main, LP)
34C: 00000000	PC → HALT()
350: 8000034C	pile: ADD(R0, R0, R0)
354: 00000000	HALT()
358: 00000001	RDCHAR()
35C: 800002FC	ADD(R0, R0, R0)
360: 00000358	ill op
364: 00000001	RDCHAR()
368: 0000000C	ill op

### STACK

2EC: C3BDFEFC
2F0: C3BD0004
2F4: 641DFFFC
2F8: 779FFFC5
2FC: 601DFFFC
300: C3BDFEFC
304: 77FF000A
308: C01F0002
30C: C3BD0004
310: 641DFFFC
314: 601DFFFC
318: C3BDFEFC
31C: C3BD0004
320: 641DFFFC
324: 779FFFC5
328: 601DFFFC
32C: C3BDFEFC
330: C7BD0000
334: 637DFFFC
338: C3BDFEFC
33C: 639DFFFC
340: C3BDFEFC
344: 6FFC0000
348: 779FFFC5
34C: 00000000
350: 8000034C

### MEM[0x37c]

348: 779FFFC5
34C: 00000000
350: 8000034C
354: 00000000
358: 00000001
35C: 800002FC
360: 00000358
364: 00000001
368: 0000000C
36C: 0000000A
370: 00000000
374: 00000000
378: 00000000
37C: 00000001
380: 00000000
384: 00000000
388: 00000000
38C: 00000000
390: 00000000
394: 00000000
398: 00000000
39C: 00000000
3A0: 00000000
3A4: 00000000
3A8: 00000000
3AC: 00000000

?

12

1

## Fonctionnement

Comme nous pouvons le voir sur les deux captures précédentes, le programme fonctionne.

## Exemple 6

But

Le but de cet exemple est de tester notre générateur de Tant Que. Le programme doit donc afficher 0 1 2 3 4 5.

## Table des symboles

```
{nom=main, type=void, cat=fonction}
{nom=i, type=int, cat=global}
{nom=n, type=int, cat=global, val=5}
```

## Arbre

```
PROG
└─FONCTION/main
  └─AFF
    └─IDF/i
      └─CONST/0
    └─TQ/1
      └─SUP
        └─IDF/i
          └─IDF/i
        └─BLOC
          └─ECR
            └─IDF/i
          └─AFF
            └─IDF/i
              └─PLUS
                └─IDF/i
                  └─CONST/1
```

## Code assembleur

```
.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(ftq1)
```

ftq1:

```
return_main:
    DEALLOCATE(0)
    POP(BP)
    POP(LP)
    RTN()
```

```
debut:
    CALL(main)
    HALT()
```

pile:

## Analyse

### BSim

The screenshot displays the BSim debugger interface with four main panels:

- REGISTERS:** A table showing the state of 32 registers (R0-R31). R0-R7 are all 00000000. R8-R15 are 00000000. R16-R23 are 00000000. R24-R31 are 00000000.
- INSTRUCTIONS (SUPERVISOR MODE):** A list of instructions with their addresses and assembly code. The PC (Program Counter) is pointing to address 35C:00000000, which contains the instruction `HALT()`.
- STACK:** A list of stack frames. The SP (Stack Pointer) is pointing to address 360:8000035C.
- MEM[0x368]:** A list of memory addresses and their values. The values are mostly 00000000, with some non-zero values at higher addresses.

Below the main panels, there is a small table with 5 rows and 1 column, containing the numbers 0, 1, 2, 3, and 4. The number 4 is highlighted with a red border.

### Fonctionnement

Tout est bien affiché, cela fonctionne.

## Exemple 7

### But

Le résultat souhaité est 5.

### Table des symboles

```
{nom=main, type=void, cat=fonction}
{nom=a, type=int, cat=global, val=10}
{nom=f, type=void, cat=fonction, nbParam=1, nbBloc=2}
{nom=i, type=int, cat=param, rang=0, scope=f}
{nom=x, type=int, cat=local, rang=0, scope=f}
{nom=y, type=int, cat=local, rang=1, scope=f}
```



## Arbre

```
{nom=main, type=void, cat=fonction}  
{nom=a, type=int, cat=global, val=10}  
{nom=f, type=void, cat=fonction, nbParam=1, nbBloc=2}  
{nom=i, type=int, cat=param, rang=0, scope=f}  
{nom=x, type=int, cat=local, rang=0, scope=f}  
{nom=y, type=int, cat=local, rang=1, scope=f}
```

## Code assembleur

```
.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(0,R0)  
        PUSH(R0)  
        POP(R0)  
        PUTFRAME(R0,-12)  
        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:
```

pile:

## Analyse

*BSim*

REGISTERS				STACK	MEM[0x3b8]
R0: 00000000A	R8: 00000000	R16: 00000000	R24: 00000000	324: 6FFC0000	384: 00000000
R1: 00000000	R9: 00000000	R17: 00000000	R25: 00000000	328: C3BD0004	388: 80000384
R2: 00000000	R10: 00000000	R18: 00000000	R26: 00000000	32C: 679DFFFC	38C: 00000000
R3: 00000000	R11: 00000000	R19: 00000000	BP: 00000000	330: C3BD0004	390: 0000000A
R4: 00000000	R12: 00000000	R20: 00000000	LP: 80000384	334: 677DFFFC	394: 80000360
R5: 00000000	R13: 00000000	R21: 00000000	SP: 00000388	338: 837DF800	398: 00000390
R6: 00000000	R14: 00000000	R22: 00000000	XP: 00000000	33C: C3BD0000	39C: 0000000A
R7: 00000000	R15: 00000000	R23: 00000000	R31: 00000000	340: 601F025C	3A0: 00000000
				344: C3BD0004	3A4: 00000000
				348: 641DFFFC	3A8: 00000000
				34C: 601DFFFC	3AC: 00000000
				350: C3BDFFFC	3B0: 00000000
				354: C3BD0004	3B4: 00000000
				358: 641DFFFC	3B8: 00000001
				35C: 779FFF7D	3BC: 00000000
				360: 601DFFFC	3C0: 00000000
				364: C3BDFFFC	3C4: 00000000
				368: C7BD0000	3C8: 00000000
				36C: 637DFFFC	3CC: 00000000
				370: C3BDFFFC	3D0: 00000000
				374: 639DFFFC	3D4: 00000000
				378: C3BDFFFC	3D8: 00000000
				37C: 6FFC0000	3DC: 00000000
				380: 779FFFE9	3E0: 00000000
				384: 00000000	3E4: 00000000
				388: 80000384	3E8: 00000000

## Fonctionnement

Comme nous pouvons le constater, le programme ne renvoie pas la bonne valeur. Cela est peut être dû au fait que la fonction créée ne renvoie rien (void), donc la valeur affichée reste la même (a = 10).

## Exemple 8

### But

Le résultat attendu de la part de cet exemple est 13.

### Table des symboles

```
{nom=main, type=void, cat=fonction}  
{nom=a, type=int, cat=global}  
{nom=f, type=int, cat=fonction, nbParam=2, nbBloc=1}  
{nom=x, type=int, cat=local, rang=0, scope=f}  
{nom=i, type=int, cat=param, rang=0, scope=f}  
{nom=j, type=int, cat=param, rang=1, scope=f}
```

### Arbre

```
PROG  
└─FUNCTION/f  
  └─AFF  
    └─IDF/x  
      └─PLUS  
        └─IDF/i  
          └─IDF/j  
            └─RET/f  
              └─PLUS  
                └─IDF/x  
                  └─CONST/10  
└─FUNCTION/main  
  └─AFF  
    └─IDF/a  
      └─APPEL/f  
        └─CONST/1  
          └─CONST/2  
└─ECR  
  └─IDF/a
```

### Code assembleur

```
.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:

## Analyse

BSim

The screenshot displays the BSim debugger interface with four main panels:

- REGISTERS:** A table showing the state of 32 registers (R0-R31). R0 is 0000000D, R8 is 00000000, R15 is 00000000, R24 is 00000000, R1 is 00000003, R9 is 00000000, R17 is 00000000, R25 is 00000000, R2 is 0000000A, R10 is 00000000, R18 is 00000000, R26 is 00000000, R3 is 00000000, R11 is 00000000, R19 is 00000000, BP is 00000000, R4 is 00000000, R12 is 00000000, R20 is 00000000, LP is 800003A0, R5 is 00000000, R13 is 00000000, R21 is 00000000, SP is 000003A4, R6 is 00000000, R14 is 00000000, R22 is 00000000, XP is 00000000, R7 is 00000000, R16 is 00000000, R23 is 00000000, R31 is 00000000.
- INSTRUCTIONS (SUPERVISOR MODE):** A list of instructions with their addresses and assembly code. The PC (Program Counter) is at 3A0:00000000, pointing to the instruction `HALT()`. Other instructions include `ADDC(SP,-4,SP)`, `SUBC(SP,0,SP)`, `LD(SP,-4,BP)`, `ADDC(SP,-4,SP)`, `LD(SP,-4,LP)`, `JMP(LP)`, `BR(main,LP)`, `ADD(R0,R0,R0)`, `ill op`, and `CYCLE()`.
- STACK:** A list of stack addresses and their corresponding values. The SP (Stack Pointer) is at 3A4:800003A0. The stack contains values like `C3BD0004`, `641DFFFC`, `779FFFC5`, `C7BD0008`, `601DFFFC`, `C3BDFFFC`, `641F025C`, `601F025C`, `C3BD0004`, `641DFFFC`, `601DFFFC`, `C3BDFFFC`, `641DFFFC`, `779FFF76`, `601DFFFC`, `C3BDFFFC`, `C7BD0000`, `637DFFFC`, `C3BDFFFC`, `639DFFFC`, `C3BDFFFC`, `6FFC0000`, `779FFFD`, `601DFFFC`, and `800003A0`.
- MEM[0x3d4]:** A list of memory addresses and their corresponding values. The memory contains values like `00000000`, `800003A0`, `00000000`, `0000000D`, `8000037C`, `000003AC`, `0000000D`, `00000003`, `0000000A`, `00000000`, `00000000`, `00000003`, `00000001`, `00000000`, `00000000`, `00000000`, `00000000`, `00000000`, `00000000`, `00000000`, `00000000`, `00000000`, `00000000`, and `00000000`.

13

## Fonctionnement

Le programme fonctionne parfaitement.