# 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	
But	
Table des symboles	
Arbre	

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

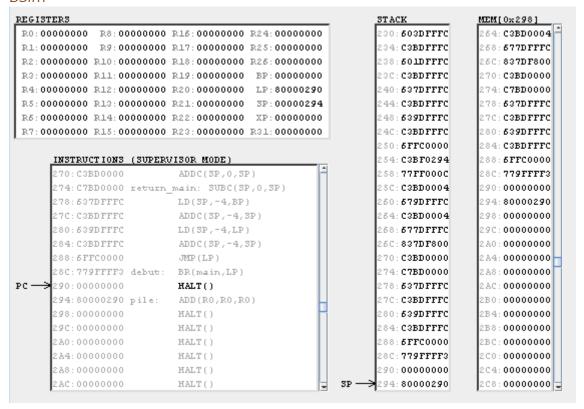
#### PROG

└FONCTION/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:
```

#### **BSim**



#### *Fonctionnement*

Le retour de BSim ne présente aucuns problèmes, ce qui parait 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

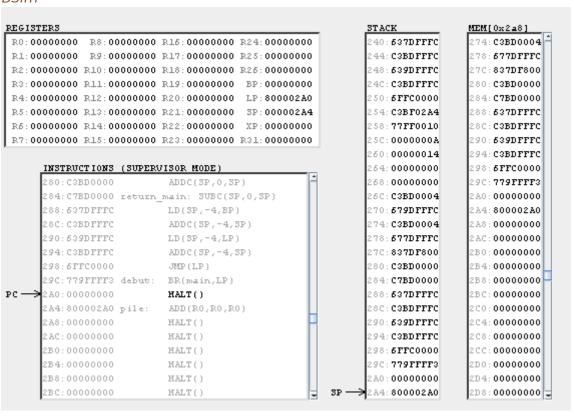
### Code assembleur

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

```
CMOVE(pile,SP)
      BR(debut)
i:
      LONG(10)
j:
      LONG(20)
k:
      LONG(0)
1:
      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**



### **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 I tel quel I = 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

FONCTION/main

AFF

IDF/k

CONST/2

AFF

IDF/l

PLUS

IDF/i

MUL

CONST/3

IDF/j
```

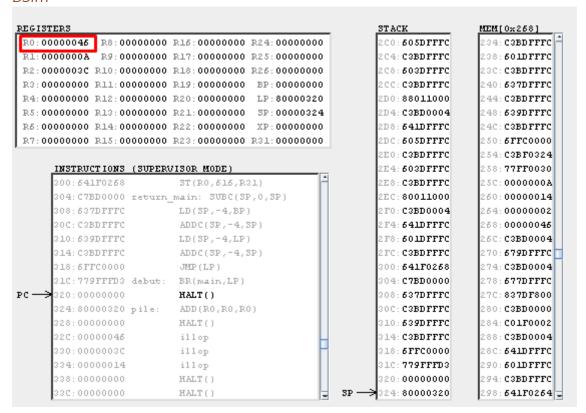
### Code assembleur

```
.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(RO)
      POP(R0)
      ST(RO,k)
      LD(i,R0)
      PUSH(RO)
      CMOVE(3,R0)
      PUSH(RO)
      LD(j,R0)
      PUSH(RO)
      POP(R2)
      POP(R1)
      MUL(R1,R2,R0)
      PUSH(RO)
      POP(R2)
      POP(R1)
      ADD(R1,R2,R0)
      PUSH(RO)
      POP(R0)
      ST(RO,I)
return_main:
      DEALLOCATE(0)
      POP(BP)
      POP(LP)
      RTN()
debut:
      CALL(main)
      HALT()
pile:
```

#### **BSim**



#### *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.

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

```
PROG
 └FONCTION/main
   ∟AFF
     └IDF/i
     ∟LIRE
   ∟ECR
     ∟PLUS
      └IDF/i
      └IDF/j
Code assembleur
```

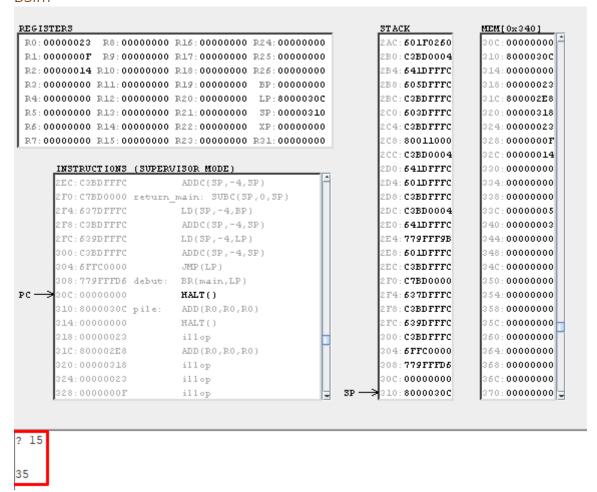
.include beta.uasm .include intio.uasm .options tty

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

HALT()

pile:

#### **BSim**



#### **Fonctionnement**

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

# Exemple 5

### But

lci 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.

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

```
PROG
└FONCTION/main
   ∟AFF
     └IDF/i
     ∟LIRE
   ∟SI/1
     ∟<sub>SUP</sub>
       └IDF/i
       └CONST/10
     ∟BL0C
       ∟ECR
         └CONST/1
     ∟BL0C
       ∟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(RO)
      POP(R0)
      ST(RO,i)
      LD(i,R0)
      PUSH(RO)
      CMOVE(10,R0)
      PUSH(RO)
      POP(R2)
      POP(R1)
      CMPLT(R2,R1,R0)
      PUSH(RO)
      POP(R0)
      BF(RO, sinon1)
      CMOVE(1,R0)
      PUSH(RO)
      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:
```

### **BSim**

```
REGISTERS
                                                             STACK
                                                                              MEM[0x37c]
 R0:00000002 R8:00000000 R15:00000000 R24:00000000
                                                              2EC: C3BDFFFC
                                                                               348:779FFFC5
 R1:00000002 R9:00000000 R17:00000000 R25:00000000
                                                              2F0: C3BD0004
                                                                               34C:00000000
 R2:0000000A R10:00000000 R18:00000000 R25:00000000
                                                              2F4: 641DFFFC
                                                                               350:8000034C
 R3:00000000 R11:00000000 R19:00000000 BP:00000000
                                                              2F8:779FFF96
                                                                               354:00000000
 R4:00000000 R12:00000000 R20:00000000 LP:8000034C
                                                              2FC: 501DFFFC
                                                                               358:00000002
 R5:00000000 R13:00000000 R21:00000000 3P:00000350
                                                              300: C3BDFFFC
                                                                               35C:80000328
                                                              304:77FF000A
                                                                               350:00000358
 R5:00000000 R14:00000000 R22:00000000 XP:00000000
 R7:00000000 R15:00000000 R23:00000000 R31:00000000
                                                              308: CO1F0002
                                                                               354:00000002
                                                                               358:00000002
                                                              30C: C3BD0004
      INSTRUCTIONS (SUPERVISOR MODE)
                                                              310:641DFFFC
                                                                               350:0000000A
                                                                               370:00000000
                                                              314:601DFFFC
      32C:C3BDFFFC
                        ADDC(SP,-4,SP)
                                                              318: C3BDFFFC
                                                                               374:00000000
      330:C7BD0000 fsil: SUBC(SP,0,SP)
       334:637DFFFC
                           LD(SP,-4,BP)
                                                              310: C3BD0004
                                                                               378:00000000
                                                                               37C:00000002
      338:C3BDFFFC
                           ADDC(SP,-4,SP)
                                                              320:641DFFFC
                          LD(SP,-4,LP)
                                                                               380:00000000
      33C:639DFFFC
                                                              324:779FFF8B
      340:C3BDFFFC
                          ADDC(SP,-4,SP)
                                                                               384:00000000
                                                              328:601DFFFC
                                                                              388:00000000
      344: 5FFC0000
                          JMP(LP)
                                                              320: C3BDFFFC
      348:779FFFC5 debut: BR(main,LP)
                                                              330: C7BD0000
                                                                              38C:00000000
                                                              334:637DFFFC
                                                                              390:00000000
PC → 34C:00000000 HALT()
       350:8000034C pile: ADD(R0,R0,R0)
                                                              338: C3BDFFFC
                                                                               394:00000000
       354:000000000
                                                              33C:639DFFFC
                                                                               398:00000000
                           HALT()
      358:000000002
                           WRCHAR()
                                                              340: C3BDFFFC
                                                                               39C:00000000
      350:80000328
                          ADD(R0,R0,R0)
                                                              344: 5FFC0000
                                                                               3A0:00000000
      350:00000358
                                                              348:779FFFC5
                                                                               3A4:00000000
                           illop
      354:000000002
                           WRCHAR()
                                                              34C:00000000
                                                                               3A8:00000000
      358:00000002
                           WRCHAR()
                                                       3P -
                                                            ⇒350:8000034C
                                                                              3AC:000000000 🖶
2 2
```

2



#### **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.

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

```
PROG
└FONCTION/main
  ∟<sub>AFF</sub>
    └IDF/i
    └const/0
  LTQ/1
    ∟<sub>SUP</sub>
       └IDF/i
       └IDF/i
    ∟BL0C
       ∟ECR
         └IDF/i
       ∟AFF
         └IDF/i
         ∟PLUS
           └IDF/i
           └CONST/1
```

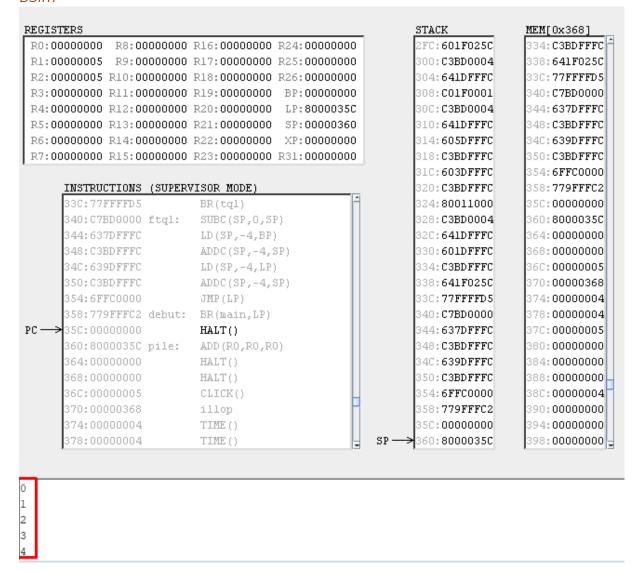
### Code assembleur

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

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

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

#### **BSim**



#### **Fonctionnement**

Tout est bien affiché, cela fonctionne.

# Exemple 7

#### But

Le résultat souhaité est 5.

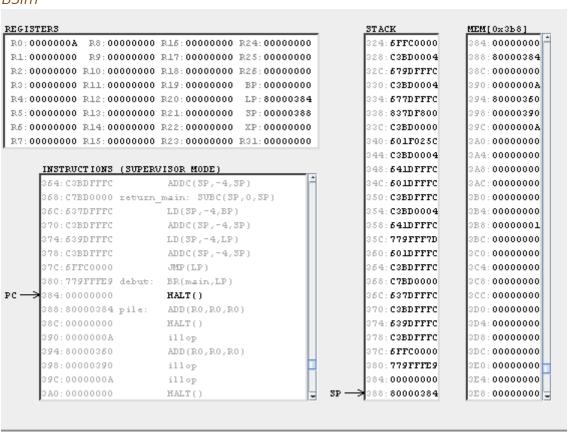
```
{nom=main, type=void, cat=fonction}
{nom=a, type=int, cat=global, val=10}
{nom=f, type=void, cat=fonction, nbParam=1, nbBloc=2}
{nom=i, type=int, cat=param, rang=0, scope=f}
{nom=x, type=int, cat=local, rang=0, scope=f}
{nom=y, type=int, cat=local, rang=1, scope=f}
```

main:

```
{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)
      LONG(10)
a:
f:
      PUSH(LP)
      PUSH(BP)
      MOVE(SP,BP)
      ALLOCATE(2)
      CMOVE(0,R0)
      PUSH(RO)
      POP(R0)
      PUTFRAME(RO,-12)
      CMOVE(1,R0)
      PUSH(RO)
      POP(R0)
      PUTFRAME(R0,4)
      GETFRAME(-12,R0)
      PUSH(RO)
      GETFRAME(0,R0)
      PUSH(RO)
      GETFRAME(4,R0)
      PUSH(RO)
      POP(R2)
      POP(R1)
      ADD(R1,R2,R0)
      PUSH(RO)
      POP(R2)
      POP(R1)
      ADD(R1,R2,R0)
      PUSH(RO)
      POP(R0)
      ST(R0,a)
return_f:
      DEALLOCATE(2)
      POP(BP)
      POP(LP)
      RTN()
```

```
PUSH(LP)
      PUSH(BP)
      MOVE(SP,BP)
      ALLOCATE(0)
      LD(a,R0)
      PUSH(RO)
      POP(R0)
      WRINT()
return main:
      DEALLOCATE(0)
      POP(BP)
      POP(LP)
      RTN()
debut:
      CALL(main)
      HALT()
pile:
```

### **BSim**



10

#### *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
└FONCTION/f
  ∟AFF.
    ∟IDF/x
    ∟PLUS
      └IDF/i
      └IDF/j
  ∟RET/f
    ∟PLUS
      └IDF/x
      └CONST/10
└FONCTION/main
  ∟AFF
    └IDF/a
    LAPPEL/f
      └CONST/1
      └CONST/2
  ∟ECR
```

### Code assembleur

└IDF/a

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

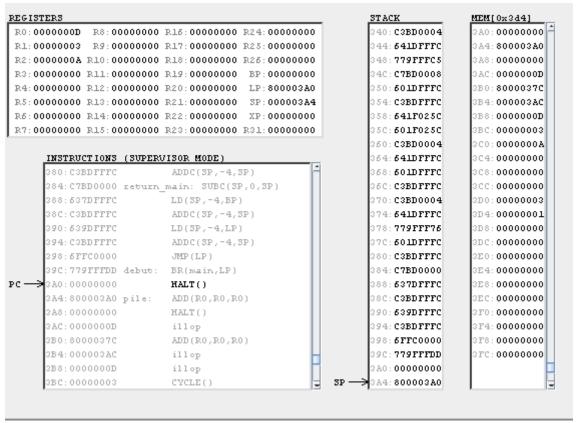
```
CMOVE(pile,SP)
BR(debut)
a: LONG(0)
f:
PUSH(LP)
PUSH(BP)
MOVE(SP,BP)
```

```
ALLOCATE(1)
      GETFRAME(-16,R0)
      PUSH(RO)
      GETFRAME(-12,R0)
      PUSH(RO)
      POP(R2)
      POP(R1)
      ADD(R1,R2,R0)
      PUSH(RO)
      POP(R0)
      PUTFRAME(R0,0)
      GETFRAME(0,R0)
      PUSH(RO)
      CMOVE(10,R0)
      PUSH(RO)
      POP(R2)
      POP(R1)
      ADD(R1,R2,R0)
      PUSH(RO)
      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(RO)
      CMOVE(2,R0)
      PUSH(RO)
      CALL(f)
      DEALLOCATE(2)
      POP(R0)
      ST(R0,a)
      LD(a,R0)
      PUSH(RO)
      POP(R0)
      WRINT()
return_main:
      DEALLOCATE(0)
      POP(BP)
      POP(LP)
      RTN()
debut:
      CALL(main)
      HALT()
```

### pile:

# Analyse

### **BSim**



13

### **Fonctionnement**

Le programme fonctionne parfaitement.