

# **WPO Week 12**

## **Executiemodel**

### **(virtual machine voor SmiLang)**

# Executie van C programma's

## C Code

```
int add(int a, int b);

int main() {
    int answer;
    answer = add(40,2);
    return 0;
}

int add(int a, int b) {
    int result = a + b;
    return result;
}
```

gcc -m32 -S main.c  
Gecompileerd naar

## Assembly code

```
.globl _add
.align 4, 0x90 _add:

pushl %ebp
movl %esp, %ebp
subl $12, %esp
movl 12(%ebp), %eax
movl 8(%ebp), %ecx
movl %ecx, -4(%ebp)
movl %eax, -8(%ebp)
movl -4(%ebp), %eax
addl -8(%ebp), %eax
movl %eax, -12(%ebp)
movl -12(%ebp), %eax
addl $12, %esp
popl %ebp
retl
## @main
.subsections_via_symbols
```

## Hardware architectuur

Uitgevoerd door



Stack aanpassen,  
registers updaten  
(%ebp, %esp, %eax...)  
Aritmetische operaties

# Executie van SmiLang programma's

## SmiLang Code

```
int one() {  
    return 1 ;  
}  
  
int minus(int arg1,  
          int arg2) {  
    return arg1 - arg2;  
}  
  
int multiTempArgs(int arg1,  
                  int arg2) {  
    int var1, var2, var3;  
    var1 = arg1 + one();  
    var2 = arg2 - arg1;  
    var3 = arg2 - var1 + var2;  
    return minus(var3, arg1 + 2);  
}
```

## Bytecode

(Manueel)  
gecompileerd naar

```
PushInt 1  
ReturnTop 0  
  
PushArg 1  
PushArg 0  
Plus  
ReturnTop 2  
  
DefineTemps 3  
PushArg 1  
Call _one  
StoreTemp 0  
PushArg 2  
PushArg 1  
Minus  
StoreTemp 1  
PushArg 1  
PushTemp 0  
Minus  
PushTemp 1  
Plus  
StoreTemp 2  
PushTemp 2  
PushArg 0  
PushInt 2  
Plus  
Call _minus  
ReturnTop 2
```

## Virtual Machine



**("stack machine")**

Geïnterpreteerd door

Stack aanpassen,  
registers updaten  
(%ebp, %esp, %eax...)  
Aritmetische operaties

Uitgevoerd door



# Executie van SmiLang programma's

Focus vandaag

## SmiLang Code

```
int one() {  
    return 1 ;  
}  
  
int minus(int arg1,  
          int arg2) {  
    return arg1 - arg2;  
}  
  
int multiTempArgs(int arg1,  
                  int arg2) {  
    int var1, var2, var3;  
    var1 = arg1 + one();  
    var2 = arg2 - arg1;  
    var3 = arg2 - var1 + var2;  
    return minus(var3, arg1 + 2);  
}
```

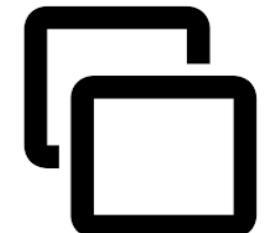
(Manueel)  
gecompileerd  
naar

## Bytecode

```
PushInt 1  
ReturnTop 0  
  
PushArg 1  
PushArg 0  
Plus  
ReturnTop 2  
  
DefineTemps 3  
PushArg 1  
Call _one  
StoreTemp 0  
PushArg 2  
PushArg 1  
Minus  
StoreTemp 1  
PushArg 1  
PushTemp 0  
Minus  
PushTemp 1  
Plus  
StoreTemp 2  
PushTemp 2  
PushArg 0  
PushInt 2  
Plus  
Call _minus  
ReturnTop 2
```

Geïnterpreteerd  
door

Virtual  
Machine



("stack machine")  
Stack aanpassen,  
registers updaten  
(%ebp, %esp, %eax...)  
Aritmetische operaties

# SmiLang

```
int one() {  
    return 1 ;  
}  
  
int minus(int arg1, int arg2) {  
    return arg1 - arg2;  
}  
  
int multiTempArgs(int arg1, int arg2) {  
    int var1, var2, var3;  
    var1 = arg1 + one();  
    var2 = arg2 - arg1;  
    var3 = arg2 - var1 + var2;  
    return minus(var3, arg1 + 2);  
}
```

Language features:

- Integer constanten
- Lokale variabelen & parameters
- + and -
- Functies

# Register vs. Stack Machine

## Register Machine (de meeste CPUs)

Plaats tijdelijke waarden  
in general-purpose registers  
(indien mogelijk)

```
movl $1, %eax  
movl $2, %ebx  
add %eax, %ebx  
movl $3, %ebx  
sub %eax, %ebx
```

1 + 2 - 3

## Stack Machine (sommige VMs)

Plaats tijdelijke waarden  
op de stack

```
PushInt 1  
PushInt 2  
Plus  
PushInt 3  
Minus
```



Stack

# Register vs. Stack Machine

## Register Machine (de meeste CPUs)

Plaats tijdelijke waarden  
in general-purpose registers  
(indien mogelijk)

```
movl $1, %eax  
movl $2, %ebx  
add %eax, %ebx  
movl $3, %ebx  
sub %eax, %ebx
```

1 + 2 - 3



Stack

## Stack Machine (sommige VMs)

Plaats tijdelijke waarden  
op de stack

```
PushInt 1  
PushInt 2  
Plus  
PushInt 3  
Minus
```

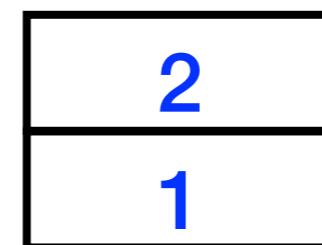
# Register vs. Stack Machine

## Register Machine (de meeste CPUs)

Plaats tijdelijke waarden  
in general-purpose registers  
(indien mogelijk)

```
movl $1, %eax  
movl $2, %ebx  
add %eax, %ebx  
movl $3, %ebx  
sub %eax, %ebx
```

1 + 2 - 3



Stack

## Stack Machine (sommige VMs)

Plaats tijdelijke waarden  
op de stack

```
PushInt 1  
PushInt 2  
Plus  
PushInt 3  
Minus
```

# Register vs. Stack Machine

## Register Machine (de meeste CPUs)

Plaats tijdelijke waarden  
in general-purpose registers  
(indien mogelijk)

```
movl $1, %eax  
movl $2, %ebx  
add %eax, %ebx  
movl $3, %ebx  
sub %eax, %ebx
```

1 + 2 - 3



Stack

## Stack Machine (sommige VMs)

Plaats tijdelijke waarden  
op de stack

```
PushInt 1  
PushInt 2  
Plus  
PushInt 3  
Minus
```

# Register vs. Stack Machine

## Register Machine (de meeste CPUs)

Plaats tijdelijke waarden  
in general-purpose registers  
(indien mogelijk)

```
movl $1, %eax  
movl $2, %ebx  
add %eax, %ebx  
movl $3, %ebx  
sub %eax, %ebx
```

1 + 2 - 3



Stack

## Stack Machine (sommige VMs)

Plaats tijdelijke waarden  
op de stack

```
PushInt 1  
PushInt 2  
Plus  
PushInt 3  
Minus
```

# Register vs. Stack Machine

## Register Machine (de meeste CPUs)

Plaats tijdelijke waarden  
in general-purpose registers  
(indien mogelijk)

```
movl $1, %eax  
movl $2, %ebx  
add %eax, %ebx  
movl $3, %ebx  
sub %eax, %ebx
```

1 + 2 - 3

0

Stack

## Stack Machine (sommige VMs)

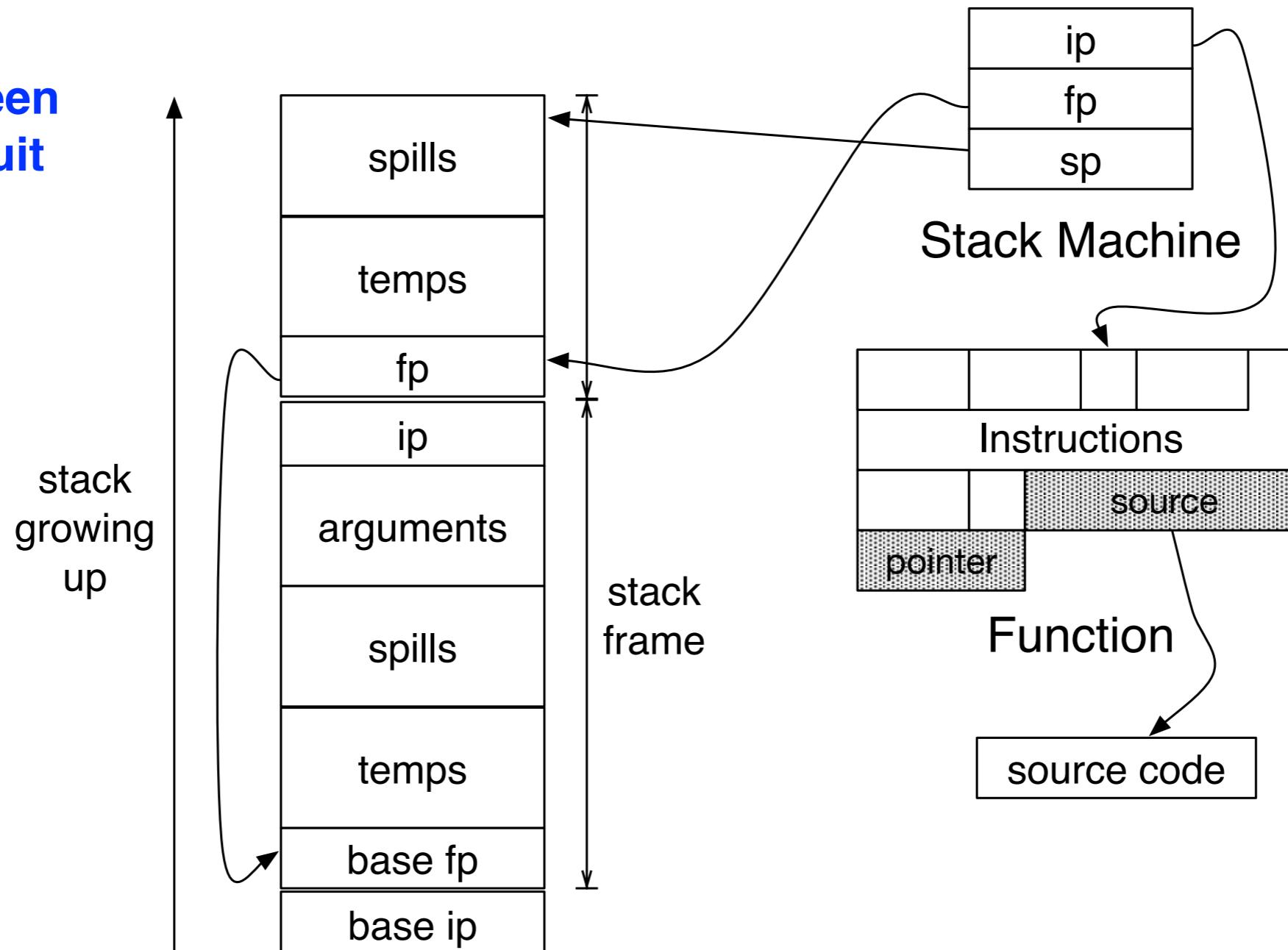
Plaats tijdelijke waarden  
op de stack

```
PushInt 1  
PushInt 2  
Plus  
PushInt 3  
Minus
```

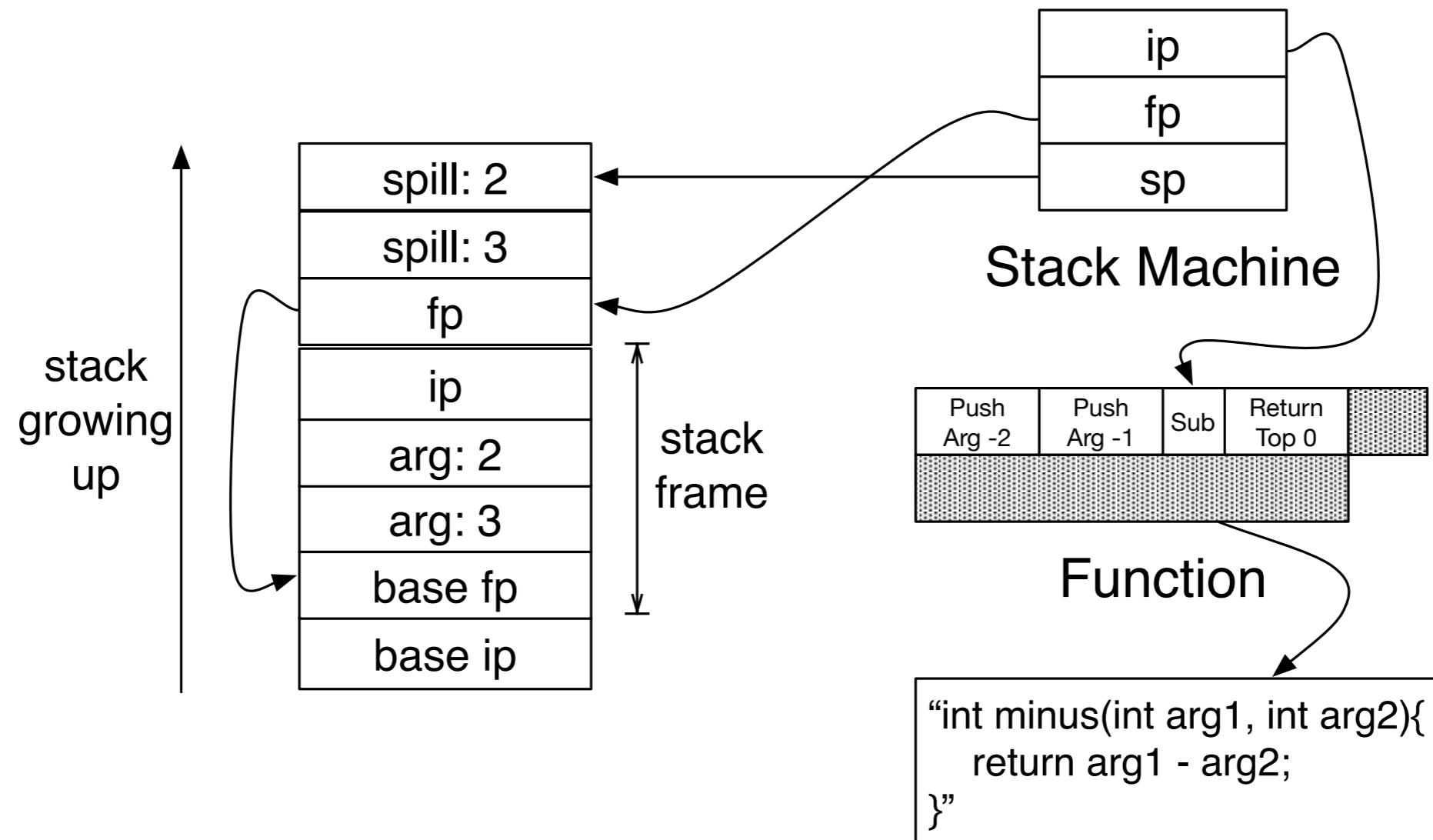
Stack machine bevat enkel de registers  
%sp, %bp en %ip

# SmiLang Stack Layout

fp komt overeen  
met de %bp uit  
het HOC



# SmiLang Stack Layout in Werking



# SmiLang Bytecode Instructies

- PushInt XXXX
- Plus
- Minus
- Pop
- PushTemp X
- StoreTemp X
- DefineTemp X
- Call XXXX
- PushArg X
- ReturnTop X

Doel vandaag: implementeer de SmiLang bytecode instructies

# Praktische Informatie

Oefeningen maken in **Dodona** of **lokaal** op je eigen computer

**Dodona**: implementeer gevraagde functies

**Lokaal**: download WPO\_week12.zip +  
implementeer gevraagde functies in bestand +  
compileer via makefile +  
run in terminal: `./stackmachine`

# Praktische Informatie (Lokaal)

WPO\_week12.zip

- encoder.c
- encoder.h
- examples.smi.c**
- examples.smi.h
- makefile
- stackmachine.c**

**(Lokaal) Je moet  
enkel code schrijven  
voor stackmachine.c**

Documenten:

- Opgaven.pdf  
(bevat o.a. extra oefeningen)
- SmiLang.pdf  
(overzicht van de taal + de stackmachine)

# examples.smi.c

WPO\_week12.zip

- encoder.c
- encoder.h
- examples.smi.c**
- examples.smi.h
- makefile
- stackmachine.c**

**(Lokaal) Je moet  
enkel code schrijven  
voor stackmachine.c**

## examples.smi.c:

```
char * example01ReturnSource = "int example01Return(){return 1;};\n\nFunction * createFunctionExample01Return(){\n    FunctionBuffer *buff = createFunctionBuffer(); // ignore this\n    pushInt(buff,1);\n    returnTop0(buff);\n    sourcePointer(buff, example01ReturnSource); // ignore this\n    return assemble(buff); // ignore this\n}
```

Compileert "manueel" de functie "example01Return"

```
int example01Return() {\n    return 1 ;\n}
```

naar de instructies:

```
PushInt 1\nReturnTop 0
```

# examples.smi.c

WPO\_week12.zip

- encoder.c
- encoder.h
- examples.smi.c**
- examples.smi.h
- makefile
- stackmachine.c**

**(Lokaal) Je moet  
enkel code schrijven  
voor stackmachine.c**

## examples.smi.c:

```
char * example01ReturnSource = "int example01Return(){return 1;}";
Function * createFunctionExample01Return();

char * example02AddSource = "int example02Add(){return 1 + 2;}";
Function * createFunctionExample02Add();

...
char * multiTempArgsSource = "int multiTempArgs(int arg1, int arg2){\n\t\
    int temp1, temp2, temp3;\n\t\
    temp1 = arg1 + one();\n\t\
    temp2 = arg2 - arg1;\n\t\
    temp3 = arg2 - temp1 + temp2;\n\t\
    return minus(temp3, arg1 + 2);\n}";
char * example11MultiTempArgsSource = "int example11MultiTempArgs()\n\t\
    return multiTempArgs(5,7);";
Function * createFunctionExample11MultiTempArgs();
```

11 “SmiLang programma's” die progressief complexer worden

# stackmachine.c

WPO\_week12.zip

- encoder.c
- encoder.h
- examples.smi.c**
- examples.smi.h
- makefile
- stackmachine.c**

**(Lokaal) Je moet  
enkel code schrijven  
voor stackmachine.c**

Doel vandaag:  
implementeer alle SmiLang  
instructies, behalve ReturnTop

## stackmachine.c:

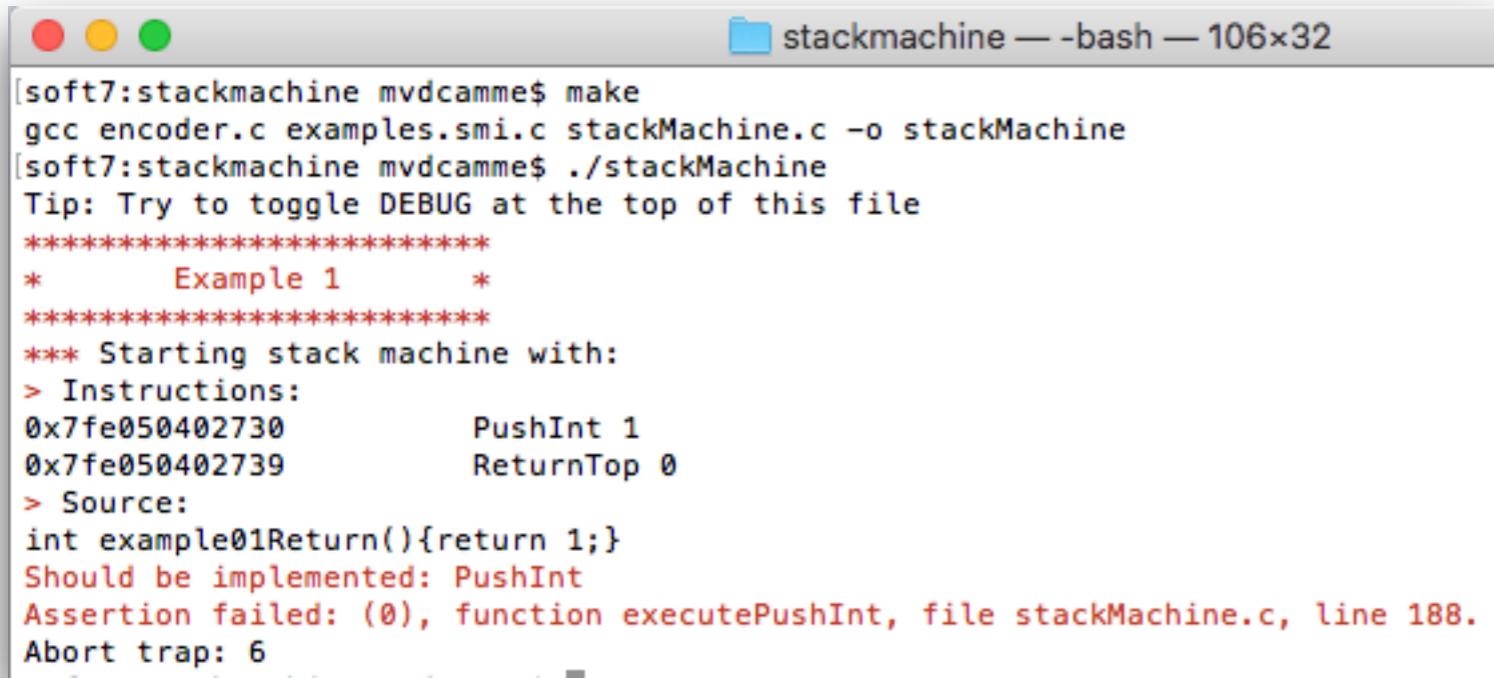
Bevat code voor:

- Interne werking van de Stack machine
- Een paar debugging functies
- Code om alle 11 voorbeeldprogramma's automatisch te draaien
- Uitvoeren van de SmiLang instructies**

```
130  ****
131  * START IMPLEMENTING HERE
132  ****
133
134 void executePop() {
135     printf("Should be implemented: POP\n");
136     assert(0);
137 }
138
139 void executePlus(){
140     printf("Should be implemented: PLUS\n");
141     assert(0);
142 }
143
144 void executeSub(){
145     printf("Should be implemented: SUB\n");
146     assert(0);
147 }
```

# De Setup Uitvoeren (Lokaal)

```
make  
./stackmachine
```



A terminal window titled "stackmachine — -bash — 106x32". The window shows the command "make" being run, followed by the output of the program. The program starts with an example, then shows the stack machine configuration, and finally runs a test function "example01Return()". It prints "Should be implemented: PushInt" and fails an assertion at line 188 of stackMachine.c.

```
[soft7:stackmachine mvdcamme$ make  
gcc encoder.c examples.smi.c stackMachine.c -o stackMachine  
[soft7:stackmachine mvdcamme$ ./stackMachine  
Tip: Try to toggle DEBUG at the top of this file  
*****  
* Example 1 *  
*****  
*** Starting stack machine with:  
> Instructions:  
0x7fe050402730 PushInt 1  
0x7fe050402739 ReturnTop 0  
> Source:  
int example01Return(){return 1;}  
Should be implemented: PushInt  
Assertion failed: (0), function executePushInt, file stackMachine.c, line 188.  
Abort trap: 6
```

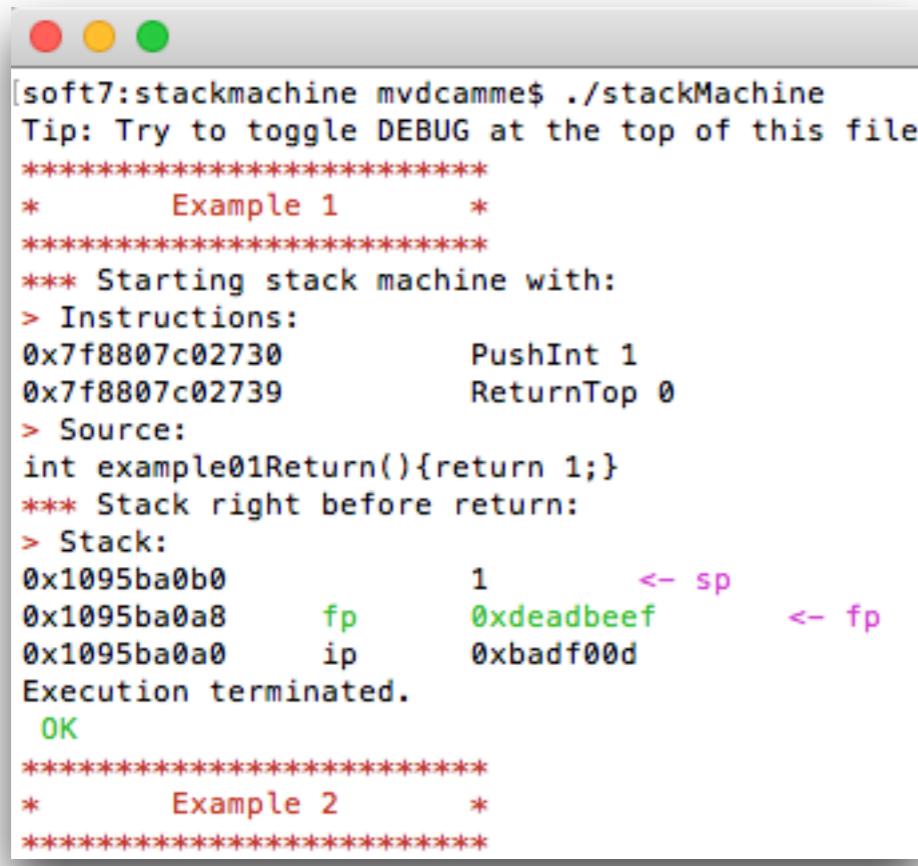
Vóór het implementeren  
van **PushInt**

```
void executePushInt(IntType cst){  
    printf("\033[0;31mShould be implemented: PushInt\033[0m\n");  
    assert(0);  
}
```

Beschrijving van de operaties: zie SmiLang.pdf, hoofdstuk 3

# De Setup Uitvoeren (Lokaal)

```
make  
./stackmachine
```



The screenshot shows a terminal window with a gray header bar containing red, yellow, and green window control buttons. The main area displays the output of a command-line application. The text is color-coded: blue for file paths and command names, red for section headers, and green for status messages like 'OK'. The application provides a tip about DEBUG mode and then executes Example 1. It shows instructions (PushInt 1, ReturnTop 0), source code (a simple return statement), and the state of the stack (sp at 0x1095ba0b0 pointing to 1, fp at 0x1095ba0a8 pointing to 0xdeadbeef). After executing Example 1, it prints 'Execution terminated.' followed by 'OK'. It then starts Example 2.

```
[soft7:stackmachine mvdcamme$ ./stackMachine
Tip: Try to toggle DEBUG at the top of this file
*****
*      Example 1      *
*****
*** Starting stack machine with:
> Instructions:
0x7f8807c02730      PushInt 1
0x7f8807c02739      ReturnTop 0
> Source:
int example01Return(){return 1;}
*** Stack right before return:
> Stack:
0x1095ba0b0          1      <- sp
0x1095ba0a8          fp     0xdeadbeef    <- fp
0x1095ba0a0          ip     0xbadf00d
Execution terminated.
OK
*****
*      Example 2      *
*****
```

Ná het implementeren van **PushInt**

```
void executePushInt(IntType cst){
// printf("\033[0;31mShould be implemented: PushInt\033[0m\n");
// assert(0); Verwijder bestaande printf en assert
// ...
}      . . .      en implementeer oplossing
```

# Oefening 1

Opgaven: zie sectie 3 in Smilang.pdf op Canvas

```
int example01Return() {                                PushInt 1
    return 1;                                         ReturnTop 0
}

void executePushInt(IntType cst){                      // TODO Implementeren in oef 1
}

void executeReturnTop(unsigned char numArgs){
    if (numArgs != 0) {
        printf("\033[0;31mShould be implemented: ReturnTop for functions with non 0 arguments\033[0m\n");
        assert(0);
    }
    IntType retValue = *sp;
    ip = *((unsigned char **)(fp - 1));
    sp = fp - 2; //before ip
    fp = *((IntType **)fp);
    *(++sp) = retValue;
}
```

# PushInt

PushInt cst

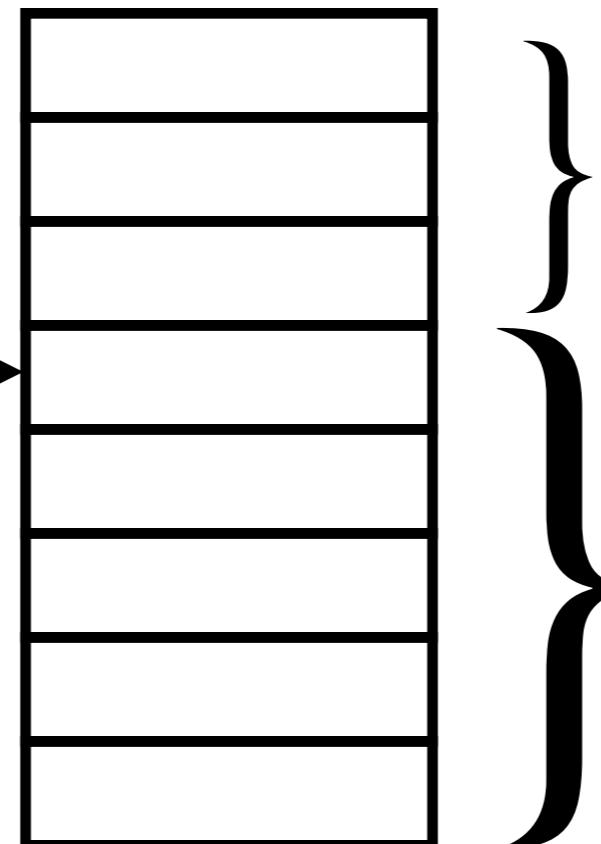
```
void executePushInt(IntType cst){  
    // TODO Implementeren in oef 1  
}  
  
IntType *fp; // Frame pointer  
IntType *sp; // Stack pointer  
unsigned char *ip; // Instruction pointer  
  
IntType baseStack[StackSize]; // Stack
```

Voor het uitvoeren van PushInt cst

Push int op de stack

sp ----->

sp wijst naar de  
top van de stack



Nutteloze waarden  
(garbage)

Nuttige waarden  
(stack frames)

# PushInt

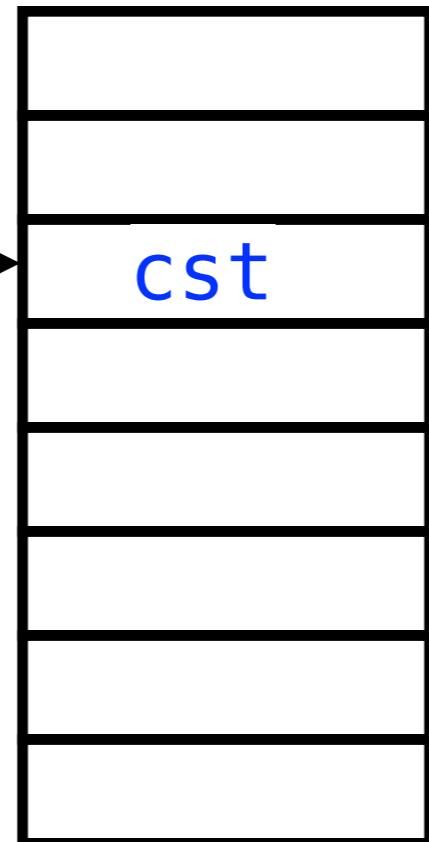
PushInt cst

```
void executePushInt(IntType cst){  
    // TODO Implementeren in oef 1  
}  
  
IntType *fp; // Frame pointer  
IntType *sp; // Stack pointer  
unsigned char *ip; // Instruction pointer  
  
IntType baseStack[StackSize]; // Stack
```

Push int op de stack

sp .....→

sp wijst naar de  
top van de stack



Na het uitvoeren van PushInt cst

} Nutteloze waarden  
(garbage)

} Nuttige waarden  
(stack frames)