Compiler Construction

Week 7: Code generation I

Sjaak Smetsers Mart Lubbers Jordy Aaldering

2024/2025 KW3

Radboud University

Recap

Abstract machine

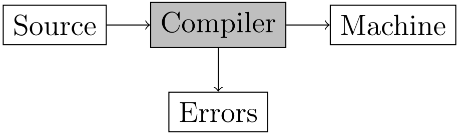
SSM

LLVM

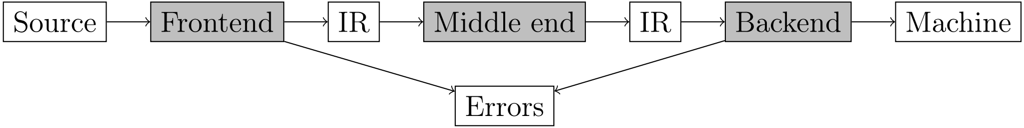
Conclusion

Recap

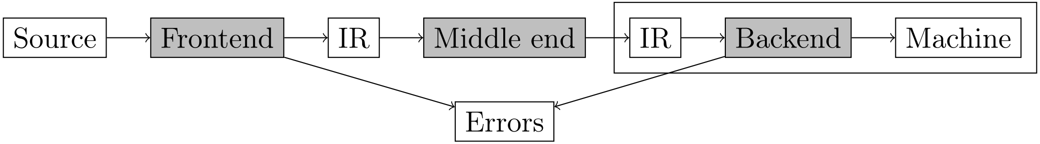
Compiler



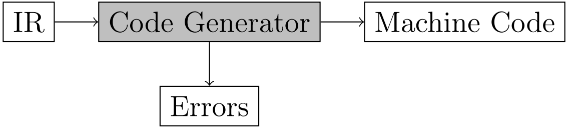
Three pass compiler



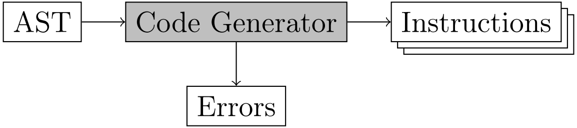
Three pass compiler



Backend

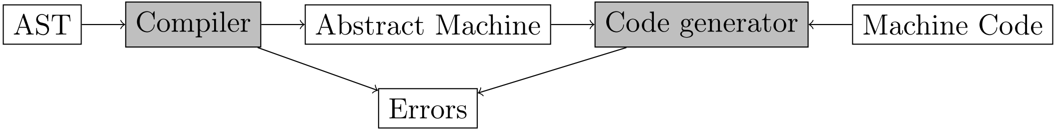


Backend

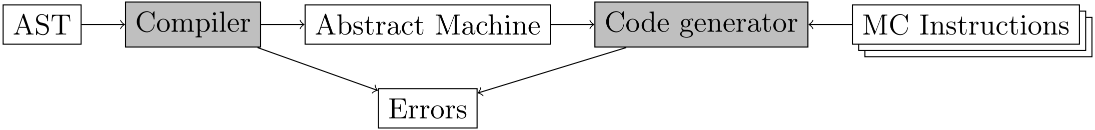


Abstract machine

Abstract machine



Abstract machine



Abstract Machines

- Convenient middle ground

Abstract Machines

- Convenient middle ground
- Examples

Abstract Machines

- Convenient middle ground
- Examples
  - **LLVM**
  - ABC
  - **SSM**
  - C
  - C--

Abstract Machines

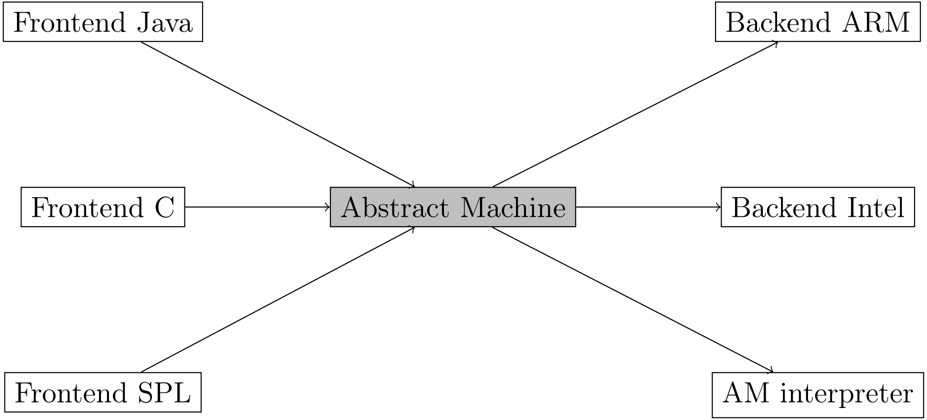
- Convenient middle ground
- Examples
- Similar Architecture but simplified

Abstract Machines

- Convenient middle ground
- Examples
- Similar Architecture but simplified
- Interpreter

Abstract Machines

- Convenient middle ground
- Examples
- Similar Architecture but simplified
- Interpreter

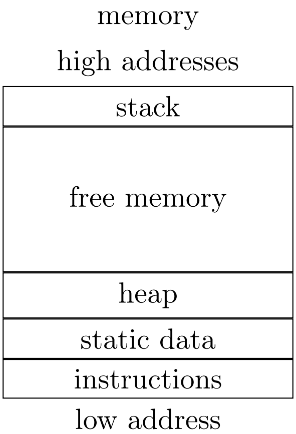


Abstract Machines

- Convenient middle ground
- Examples
- Similar Architecture but simplified
- Interpreter
- Assignment 3: Compiler SPL to SSM or LLVM*

Typical memory layout

Typical memory layout



memory

high addresses

stack

free memory

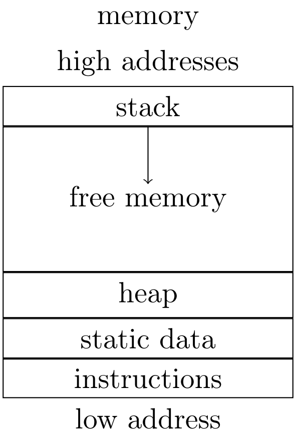
heap

static data

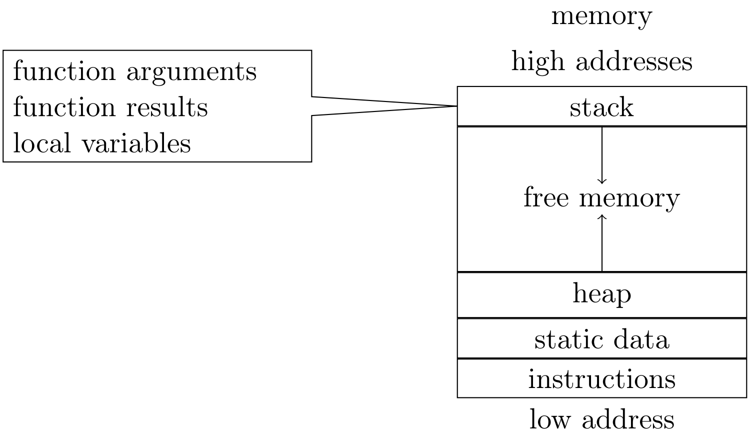
instructions

low address

Typical memory layout



Typical memory layout



memory

high addresses

function arguments
function results
local variables

stack

free memory

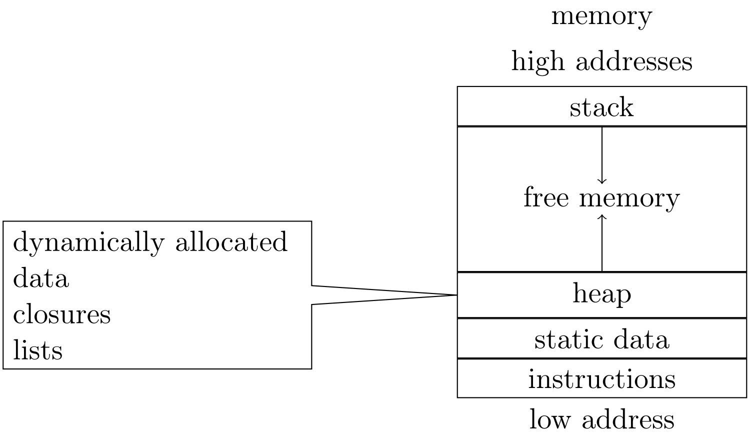
heap

static data

instructions

low address

Typical memory layout



memory

high addresses

stack

free memory

heap

static data

instructions

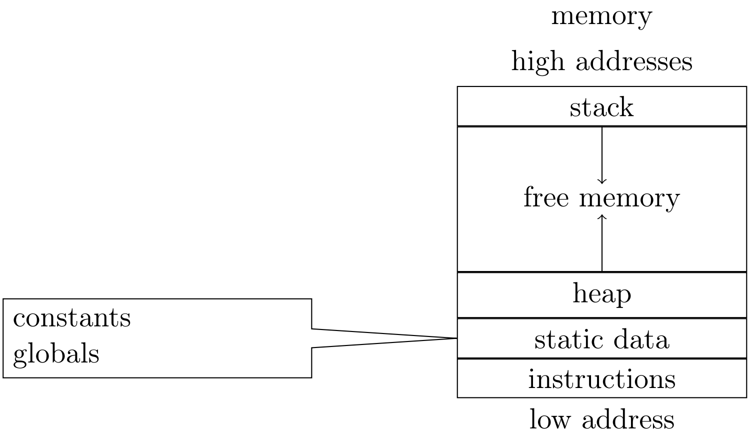
low address

dynamically allocated data

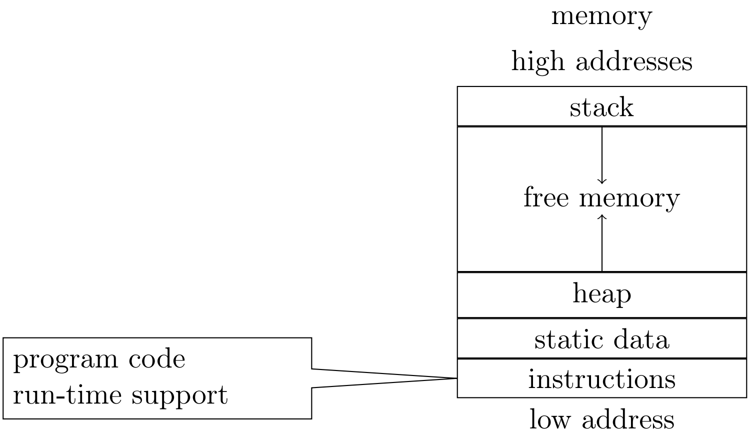
closures

lists

Typical memory layout



Typical memory layout



memory

high addresses

stack

free memory

heap

static data

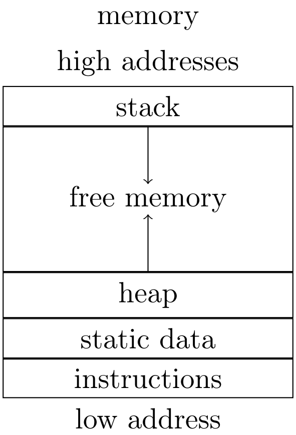
instructions

low address

program code

run-time support

Typical memory layout



memory

high addresses

stack

free memory

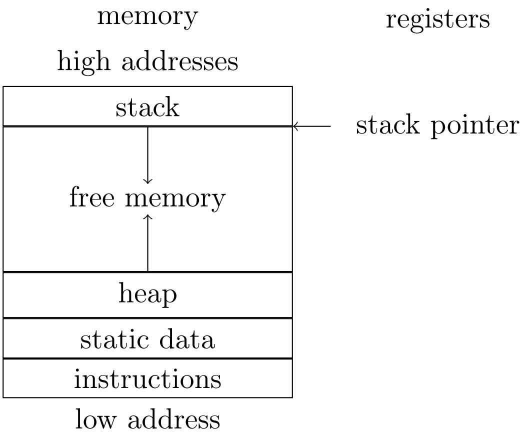
heap

static data

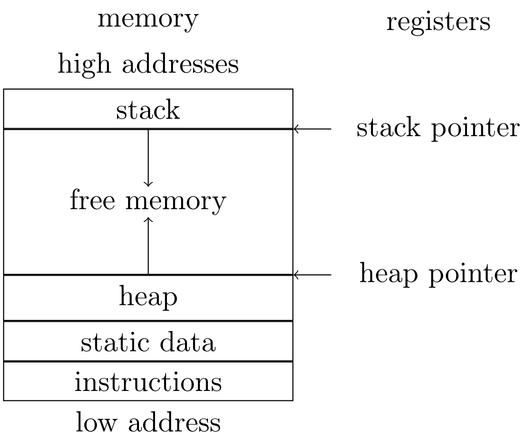
instructions

low address

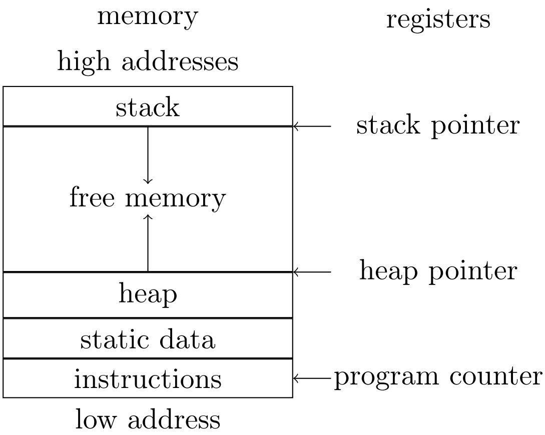
Typical memory layout



Typical memory layout



Typical memory layout

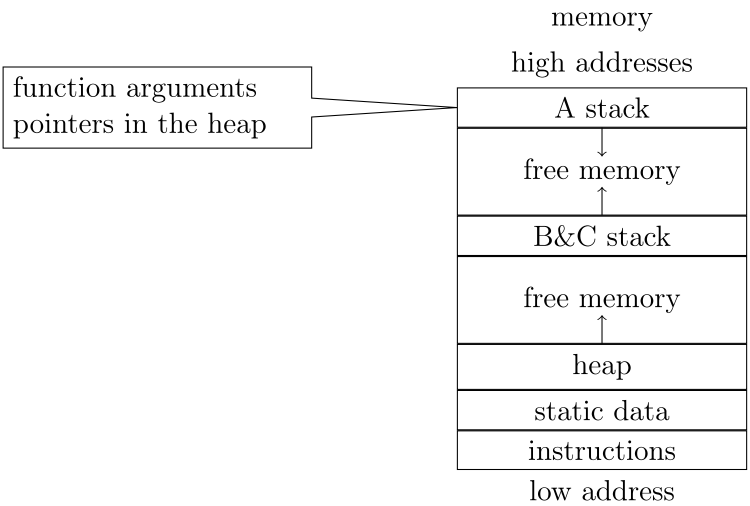


Typical memory layout

ABC memory layout

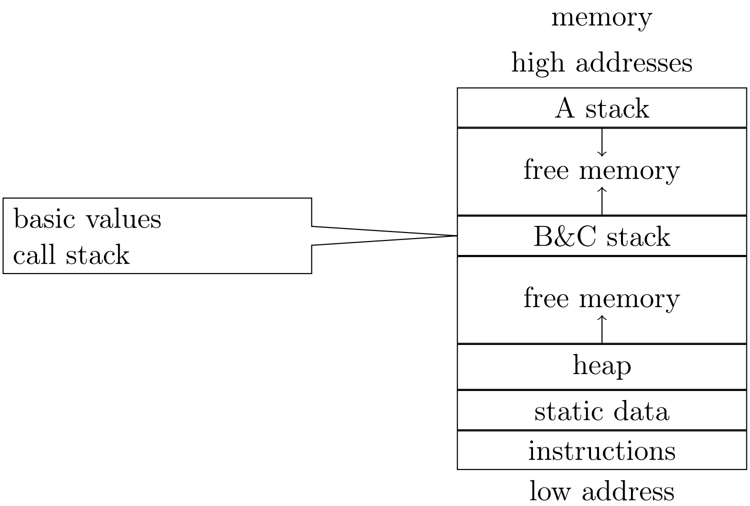
Typical memory layout

ABC memory layout



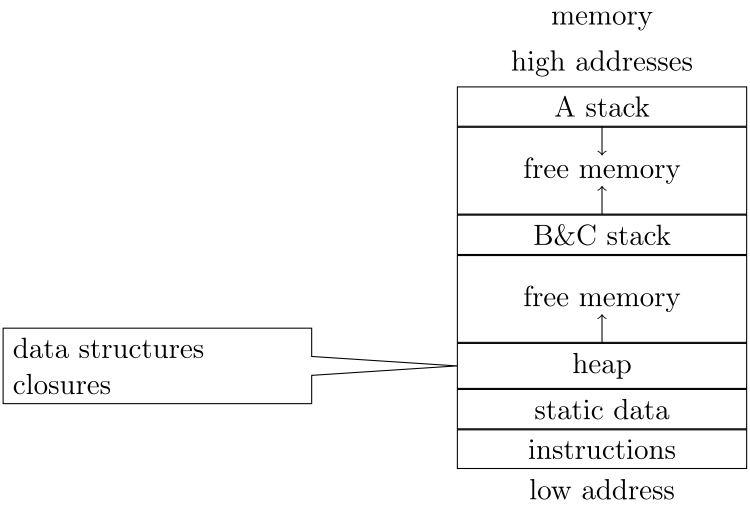
Typical memory layout

ABC memory layout



Typical memory layout

ABC memory layout



memory

high addresses

A stack

free memory

B&C stack

free memory

heap

static data

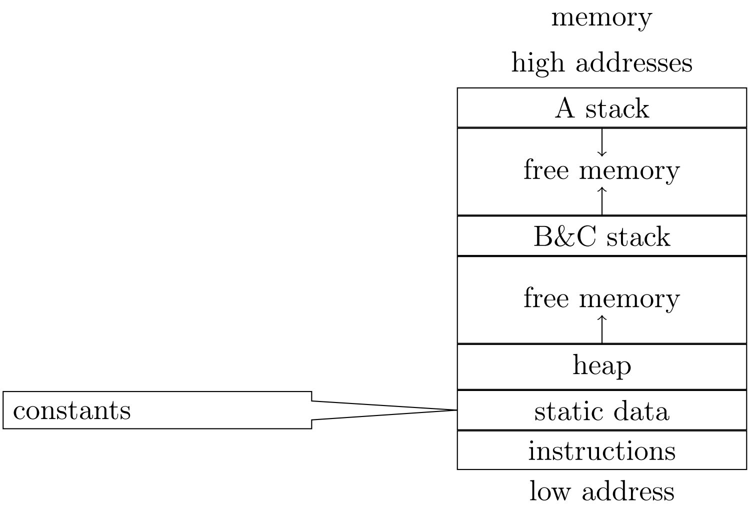
instructions

low address

data structures
closures

Typical memory layout

ABC memory layout



memory

high addresses

A stack

free memory

B&C stack

free memory

heap

static data

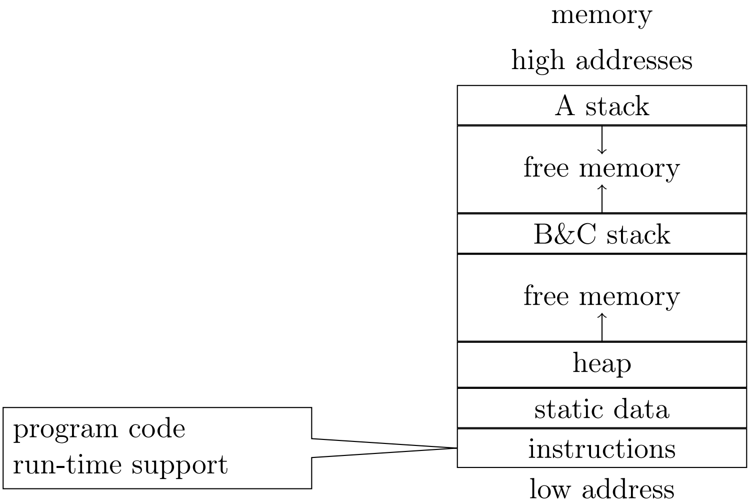
instructions

low address

constants

Typical memory layout

ABC memory layout



memory

high addresses

A stack

free memory

B&C stack

free memory

heap

static data

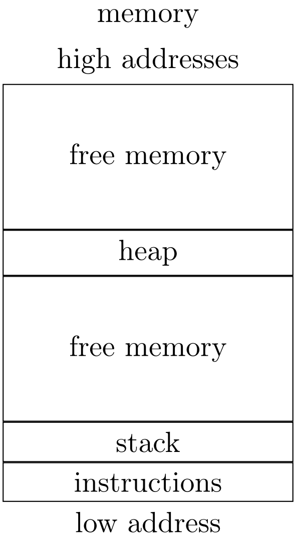
instructions

low address

program code
run-time support

Non-typical memory layout

Simple Stack Machine (SSM)



memory

high addresses

free memory

heap

free memory

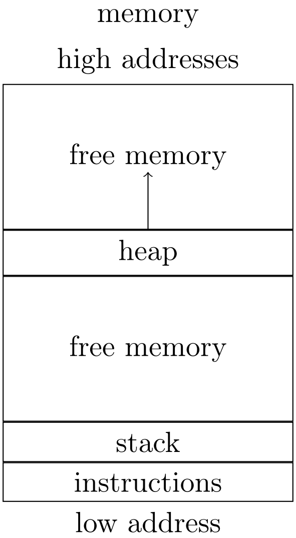
stack

instructions

low address

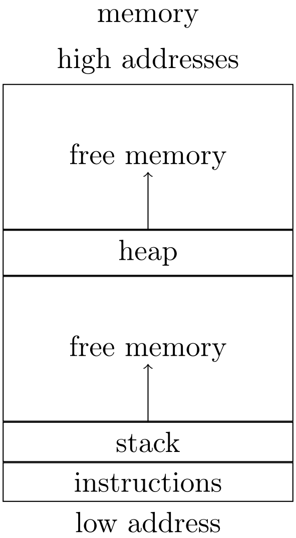
Non-typical memory layout

Simple Stack Machine (SSM)



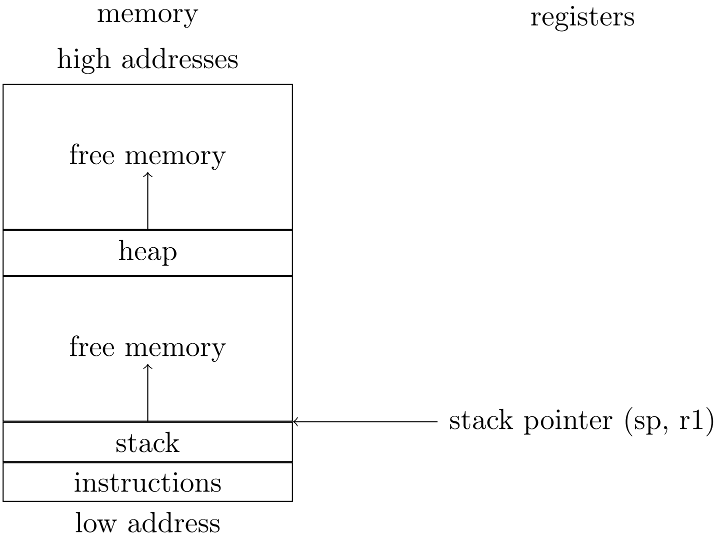
Non-typical memory layout

Simple Stack Machine (SSM)



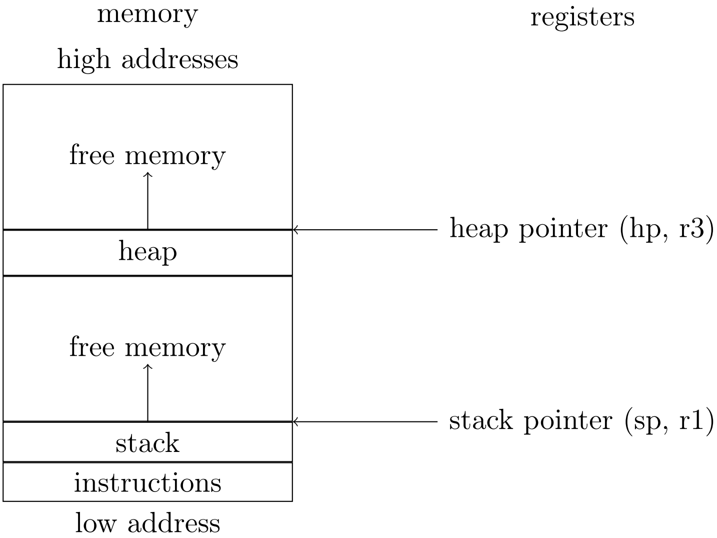
Non-typical memory layout

Simple Stack Machine (SSM)



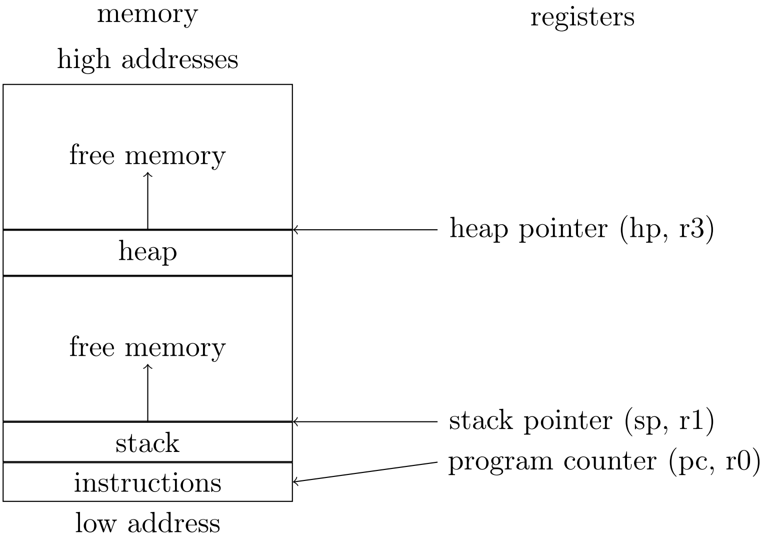
Non-typical memory layout

Simple Stack Machine (SSM)



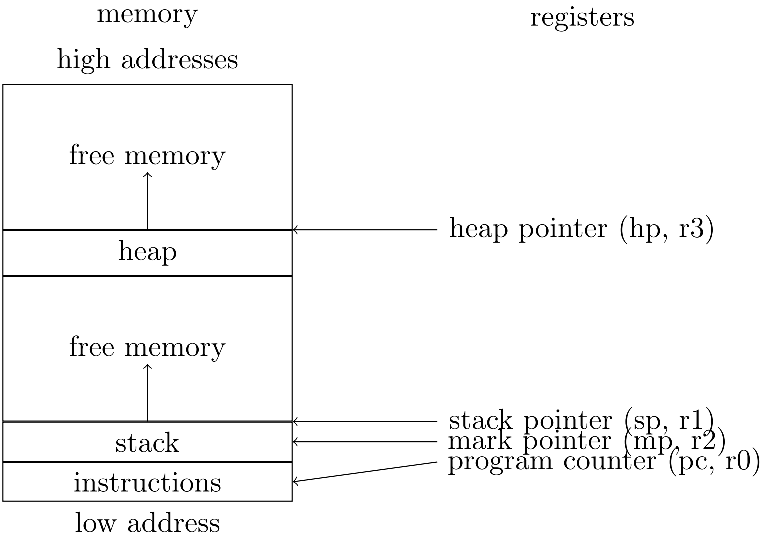
Non-typical memory layout

Simple Stack Machine (SSM)



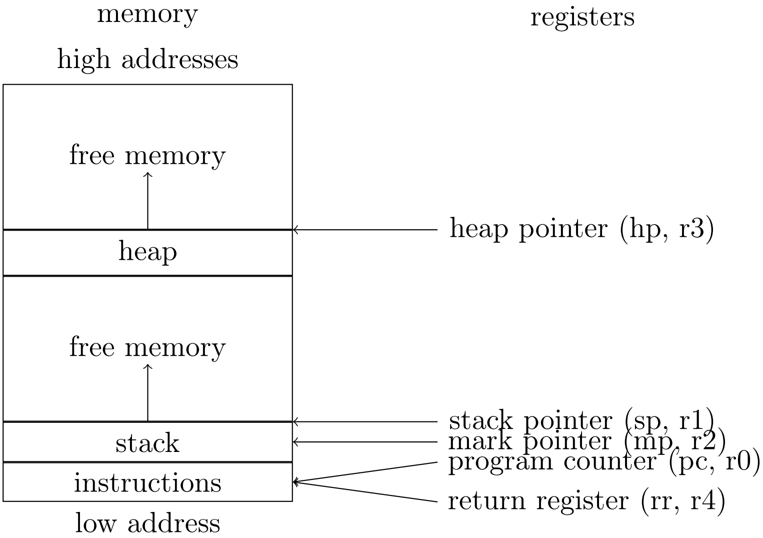
Non-typical memory layout

Simple Stack Machine (SSM)



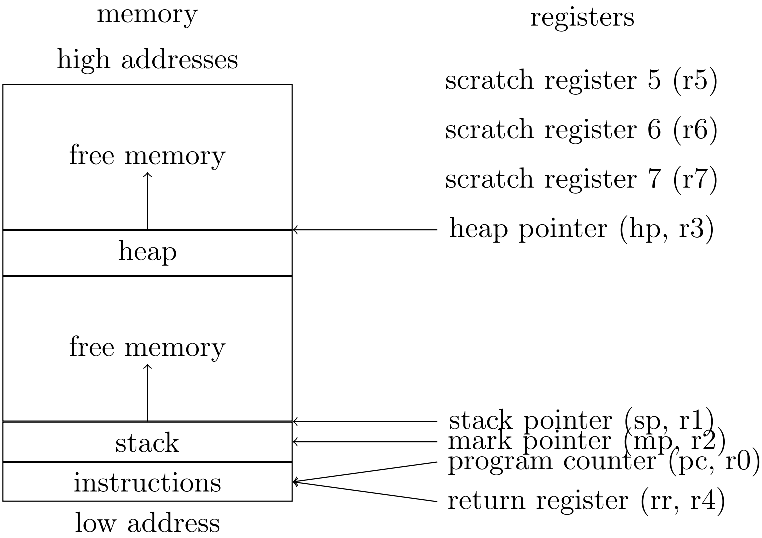
Non-typical memory layout

Simple Stack Machine (SSM)



Non-typical memory layout

Simple Stack Machine (SSM)



SSM

SSM Grammar

```
SSMProgram  ::= Line*
Line        ::= ((Label ":")?) (Instruction?) (Comment?)
Label       ::= Identifier
Instruction ::= ("ldc" | ...) Argument*
Argument    ::= Label | "-"? Number
Number      ::= Decimal | HexaDecimal
Decimal     ::= DecDigit*
HexaDecimal ::= "0x" HexDigit*
DecDigit    ::= "0" | .. |  "9"
HexDigit    ::= DecDigit | "a" | .. | "f" | "A" | .. | "F‘‘
Identifier  ::= DecDigit | "a" | .. | "z" | "A" | .. | "F" | "-" | "_"
Comment     ::= (";" | "//") .*
```

SSM Tip (1)

Annotate instructions

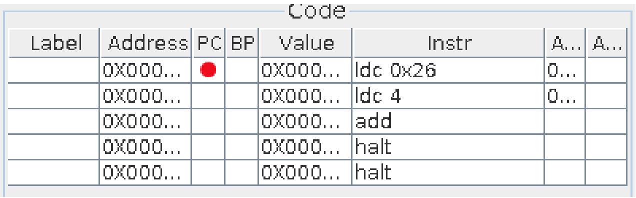
```
annote REG LOWOFFSET HIGHOFFSET COLOR MESSAGE
```

```
ldc 38
ldc 4
add
annote SP 0 0 red "The Answer"
halt
```

SSM Tip (1)

Annotate instructions

```
ldc 38
ldc 4
add
annote SP 0 0 red "The Answer"
halt
```



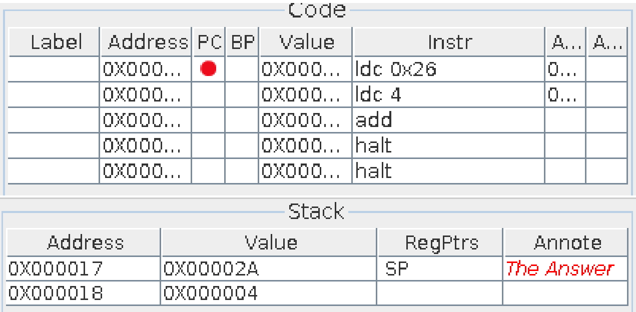
Code

Label	Address	PC	BP	Value	Instr	A...	A...
	0X000...	●		0X000...	ldc 0x26	0...	
	0X000...			0X000...	ldc 4	0...	
	0X000...			0X000...	add		
	0X000...			0X000...	halt		
	0X000...			0X000...	halt		

SSM Tip (1)

Annotate instructions

```
ldc 38
ldc 4
add
annote SP 0 0 red "The Answer"
halt
```



Code

Label	Address	PC	BP	Value	Instr	A...	A...
	0X000...	●		0X000...	ldc 0x26	0...	
	0X000...			0X000...	ldc 4	0...	
	0X000...			0X000...	add		
	0X000...			0X000...	halt		
	0X000...			0X000...	halt		

Stack

Address	Value	RegPtrs	Annote
0X000017	0X00002A	SP	*The Answer*
0X000018	0X000004		

SSM Tips (2)

CLI

```
frobnicator~/projects/ssm$ java -jar ssm.jar --help
Simple Stack Machine Interpreter
Version 2.4.0, May 10, 2022
usage: [--clisteps <steps>] [--cli] [--file <path> OR --stdin]
  --help               : Print this help
  --version            : Print version
  --clisteps <steps> : The amount of steps to run. -1 for infinite(default)
    .
                       Only in cli mode
  --stdin              : Read code from stdin
  --file <path>        : Read code from path
  --cli                : No GUI, runs code and exits on halt
  --haltonerror        : Halt on error. Only in cli mode
  --guidelay           : Amount of time to sleep in milliseconds between
    steps in
                       the GUI. Default: 50
```

SSM Tips (3)

Documentation

SSM Tips (3)

Documentation

Read the documentation

SSM Tips (3)

Documentation

Read the documentation

Read the documentation

SSM Tips (3)

Documentation

Read the documentation

Read the documentation

Read the documentation

SSM Tips (3)

Documentation

Read the documentation

Read the documentation

Read the documentation

Read the documentation

SSM Tips (3)

Documentation

Read the documentation
Read the documentation
Read the documentation
Read the documentation
Read the documentation

SSM Tips (3)

Documentation

Read the documentation

Read the documentation

Read the documentation

Read the documentation

Read the documentation

Read the documentation

SSM Tips (3)

Documentation

Read the documentation

Read the documentation

Read the documentation

Read the documentation

Read the documentation

Read the documentation

Read the documentation

SSM Tips (3)

Documentation

Read the documentation

Read the documentation

Read the documentation

Read the documentation

Read the documentation

Read the documentation

Read the documentation

Read the documentation

SSM Tips (3)

Documentation

Read the documentation

Read the documentation

Read the documentation

Read the documentation

Read the documentation

Read the documentation

Read the documentation

Read the documentation

Read the documentation

SSM Tips (3)

Documentation

Read the documentation
Read the documentation
Read the documentation
Read the documentation
Read the documentation
Read the documentation
Read the documentation
Read the documentation
Read the documentation
Read the documentation

Instruction overview

<table>
<tr><th colspan="5">Copying</th></tr>
<tr><th>Load</th><th>Load Multiple</th><th>Load Address</th><th>Store</th><th>Store multiple</th></tr>
<tr><td>lda/ldh</td><td>ldma/ldmh</td><td>ldaa</td><td>sta</td><td>stma</td></tr>
<tr><td>lds</td><td>ldms</td><td>ldsa</td><td>sts</td><td>stms</td></tr>
<tr><td>ldl</td><td>ldml</td><td>ldml</td><td>stl</td><td>stml</td></tr>
<tr><td>ldc</td><td></td><td></td><td></td><td></td></tr>
<tr><th colspan="5">Register</th></tr>
<tr><td>ldr</td><td>str</td><td>str</td><td></td><td></td></tr>
<tr><td>swp</td><td>swpr</td><td>swprr</td><td></td><td></td></tr>
</table>

Instruction overview

Copying				
Load	Load Multiple	Load Address	Store	Store multiple
lda/ldh	ldma/ldmh	ldaa	sta	stma
lds	ldms	ldsa	sts	stms
ldl	ldml	ldml	stl	stml
ldc				
Register				
ldr	str	str		
swp	swpr	swprr		

ld	load	a/h	address	m	multiple
st	store	s	stack	-a	indirection
		l	local		
		c	constant		
		r	register		

Instruction overview

Copying				
Load	Load Multiple	Load Address	Store	Store multiple
lda/ldh	ldma/ldmh	ldaa	sta	stma
lds	ldms	ldsa	sts	stms
ldl	ldml	ldml	stl	stml
ldc				
Register				
ldr	str	str		
swp	swpr	swprr		

Documentation **lda**

```
SP_post = SP_pre
M_post[SP_post] = M_pre[M_pre[SP_pre] + M_pre[PC_pre+1]]
```

Instruction overview

Copying				
Load	Load Multiple	Load Address	Store	Store multiple
lda/ldh	ldma/ldmh	ldaa	sta	stma
lds	ldms	ldsa	sts	stms
ldl	ldml	ldml	stl	stml
ldc				
Register				
ldr	str	str		
swp	swpr	swprr		

Documentation **lds**

```
SP_post = SP_pre + 1
M_post[SP_post] = M_pre[SP_pre + M_pre[PC_pre+1]
```

Instruction overview

Copying				
Load	Load Multiple	Load Address	Store	Store multiple
lda/ldh	ldma/ldmh	ldaa	sta	stma
lds	ldms	ldsa	sts	stms
ldl	ldml	ldml	stl	stml
ldc				
Register				
ldr	str	str		
swp	swpr	swprr		

Documentation **ldl**

```
SP_post = SP_pre + 1
M_post[SP_post] = M_pre[MP_pre + M_pre[PC_pre+1]
```

Instruction overview

Copying				
Load	Load Multiple	Load Address	Store	Store multiple
lda/ldh	ldma/ldmh	ldaa	sta	stma
lds	ldms	ldsa	sts	stms
ldl	ldml	ldml	stl	stml
ldc				
Register				
ldr	str	str		
swp	swpr	swprr		

Documentation **ldc**

```
SP_post = SP_pre + 1
M_post[SP_post] = M_pre[PC_pre+1]
```

Instruction overview

Copying				
Load	Load Multiple	Load Address	Store	Store multiple
lda/ldh	ldma/ldmh	ldaa	sta	stma
lds	ldms	ldsa	sts	stms
ldl	ldml	ldml	stl	stml
ldc				
Register				
ldr	str	str		
swp	swpr	swprr		

Documentation **ldma**

```
displ = M_pre[PC_pre + 1]
size = M_pre[PC_pre + 2]
SP_post = SP_pre + size - 1
M_post[SP_post - size + 1 .. SP_post]
 = M_pre[M_pre[SP_pre] + displ .. M_pre[SP_pre] + displ + size - 1]
```

Instruction overview

		Copying		
Load	Load Multiple	Load Address	Store	Store multiple
lda/ldh	ldma/ldmh	ldaa	sta	stma
lds	ldms	ldsa	sts	stms
ldl	ldml	ldml	stl	stml
ldc				
		Register		
ldr	str	str		
swp	swpr	swprr		

Documentation **ldla**

```
SP_post = SP_pre + 1
M_post[SP_post] = MP_pre + M_pre[PC_pre+1]
```

Instruction overview

Arithmetics

Integer					
add	sub	mul	div	mod	neg
Comparison					
eq	ne	lt	le	gt	ge
Boolean					
and	or	xor	not		

Instruction overview

Arithmetics

Integer					
add	sub	mul	div	mod	neg
Comparison					
eq	ne	lt	le	gt	ge
Boolean					
and	or	xor	not		

Representation

- False: 0
- True: -1 (actually anything else)
- Integers: two's complement
- Characters: unicode integers

Instruction overview

Branching

<table>
<tr><th colspan="5">Adjust stack pointer</th></tr>
<tr><td>ajs</td><td></td><td></td><td></td><td></td></tr>
<tr><th colspan="5">Branch</th></tr>
<tr><td>bra</td><td>brt</td><td>brf</td><td></td><td></td></tr>
<tr><th colspan="5">Subroutines</th></tr>
<tr><td>bsr</td><td>jsr</td><td>ret</td><td>link</td><td>unlink</td></tr>
</table>

Subroutines

bsr	Branch to subroutine (push pc, jump)
jsr	Jump to subroutine (bsr but use pc from stack)
link	Allocate local variables (push mp, adjust sp)
unlink	Deallocate local variables (adjust sp, pop mp)
ret	Return from subroutine (pop pc, jump)

Instruction overview

Misc

nop
halt
trap

Instruction overview

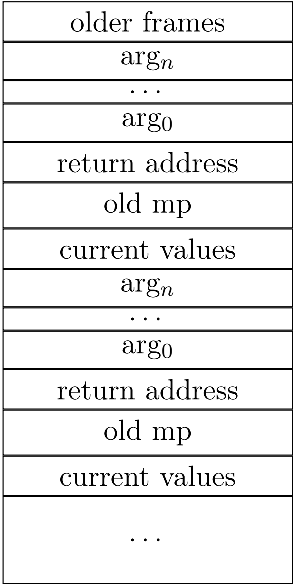
Misc

nop
halt
trap

No	Semantics
0	Print int
1	Print char
2	Print char array
10	Ask integer
11	Ask char
12	Ask char array
20	open file for reading
21	open file for writing
22	read char from file
23	write char to file
24	close file

Stack Frames

- Where to store things



older frames

arg_n

$\ldots$

arg_0

return address

old mp

current values

arg_n

$\ldots$

arg_0

return address

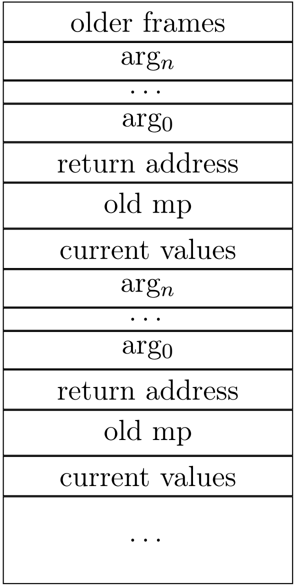
old mp

current values

$\ldots$

Stack Frames

- Where to store things
- Arguments



older frames

arg_n

…

arg_0

return address

old mp

current values

arg_n

…

arg_0

return address

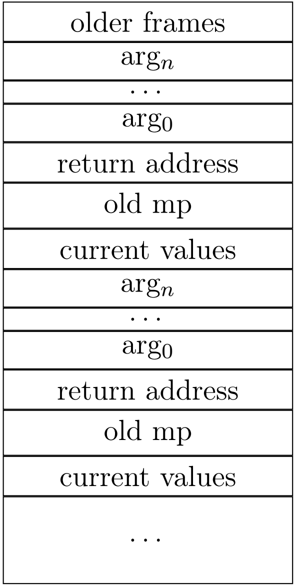
old mp

current values

…

Stack Frames

- Where to store things
- Arguments
- Locals



older frames

arg_n

$\ldots$

arg_0

return address

old mp

current values

arg_n

$\ldots$

arg_0

return address

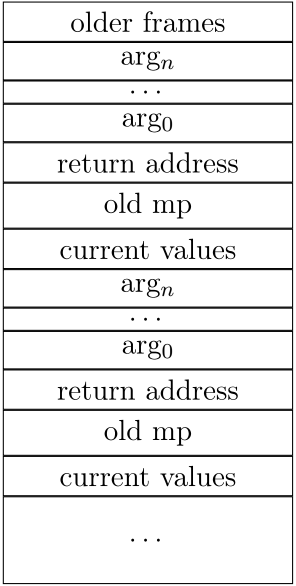
old mp

current values

$\ldots$

Stack Frames

- Where to store things
- Arguments
- Locals
- Globals (later)



older frames

arg_n

$\ldots$

arg_0

return address

old mp

current values

arg_n

$\ldots$

arg_0

return address

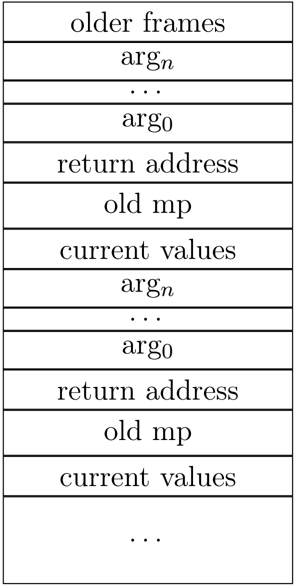
old mp

current values

$\ldots$

Stack Frames

- Where to store things
- Arguments
- Locals
- Globals (later)



older frames

arg_n

$\ldots$

arg_0

return address

old mp

current values

arg_n

$\ldots$

arg_0

return address

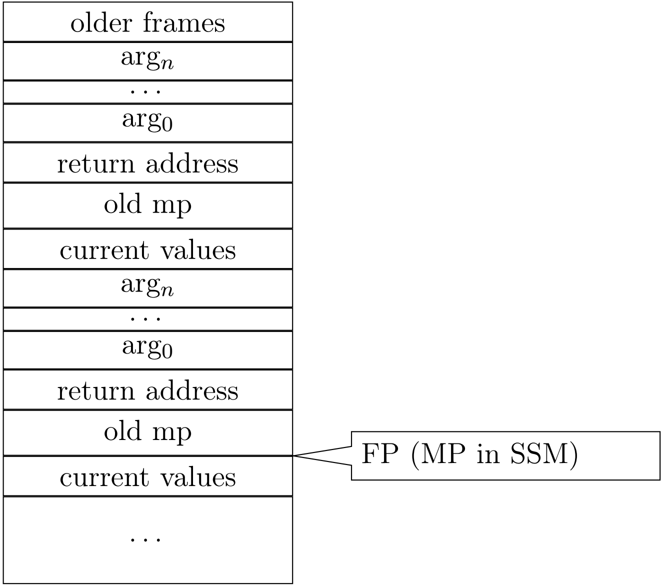
old mp

current values

$\ldots$

Stack Frames

- Where to store things
- Arguments
- Locals
- Globals (later)



older frames

arg_n

...

arg_0

return address

old mp

current values

arg_n

...

arg_0

return address

old mp

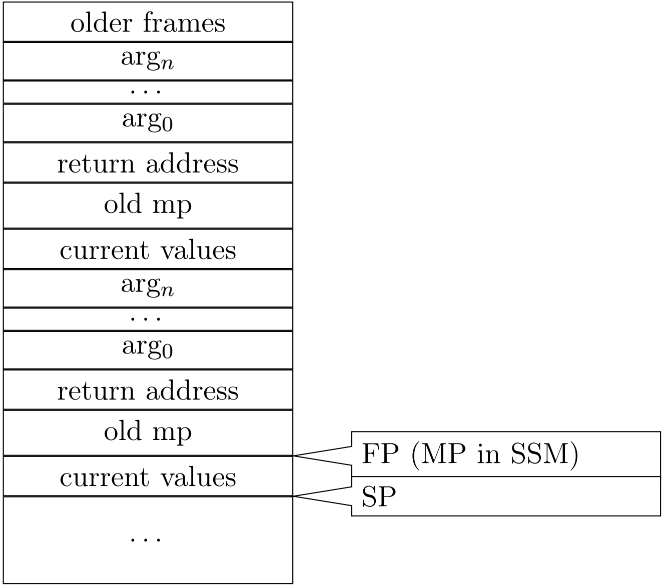
current values

...

FP (MP in SSM)

Stack Frames

- Where to store things
- Arguments
- Locals
- Globals (later)



older frames

arg$_n$

. . .

arg$_0$

return address

old mp

current values

arg$_n$

. . .

arg$_0$

return address

old mp

current values

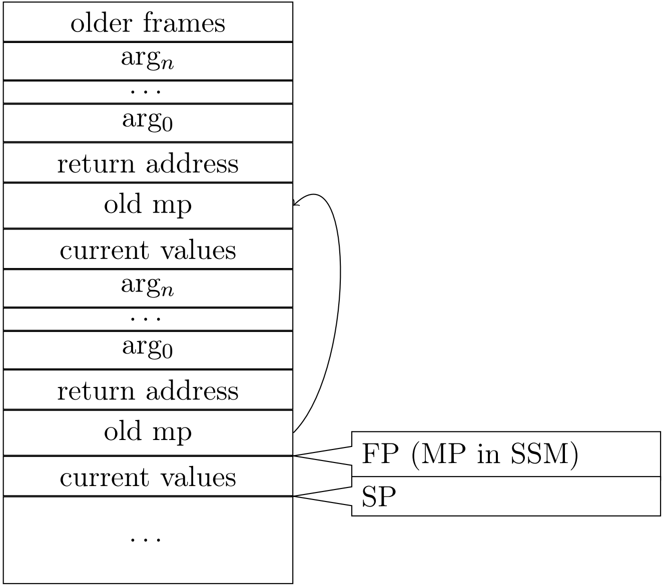
. . .

FP (MP in SSM)

SP

Stack Frames

- Where to store things
- Arguments
- Locals
- Globals (later)



older frames

arg_n

...

arg_0

return address

old mp

current values

arg_n

...

arg_0

return address

old mp

current values

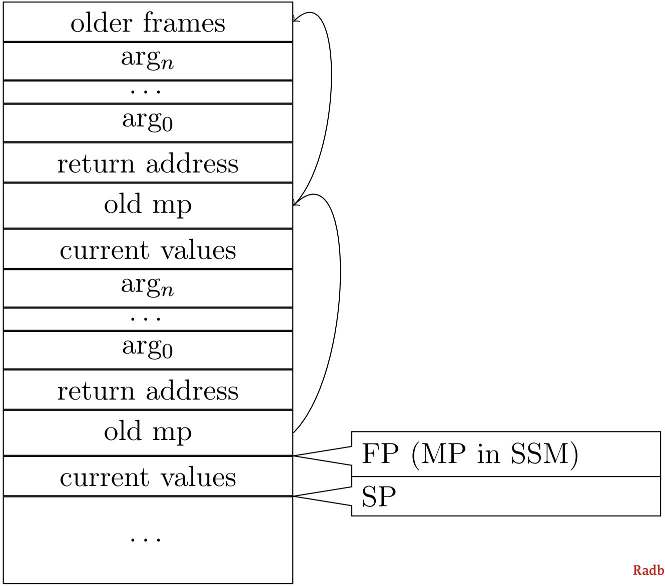
...

FP (MP in SSM)

SP

Stack Frames

- Where to store things
- Arguments
- Locals
- Globals (later)



older frames

arg_n

...

arg_0

return address

old mp

current values

arg_n

...

arg_0

return address

old mp

current values

...

FP (MP in SSM)

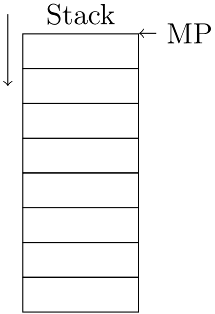
SP

Stack Frame Example

```
f (x, y) {
  var z = 4;
  return x+y+z;
}

main () {
  f(30, 8);
}
```

```
 0          bra main ← pc
 2  f:      link 1
 4          ldc 4
 6          stl 1
 8          ldl -3
10          ldl -2
12          add
13          ldl 1
15          add
16          str RR
18          unlink
19          ret
20  main:   ldc 30
22          ldc 8
24          bsr f
26          ajs -2
28          ldr RR
30          trap 0
32          halt
```

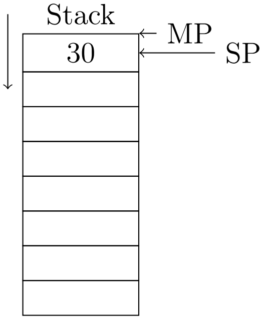


Stack Frame Example

```
f (x, y) {
  var z = 4;
  return x+y+z;
}

main () {
  f(30, 8);
}
```

```
0            bra main
2   f:       link 1
4            ldc 4
6            stl 1
8            ldl -3
10           ldl -2
12           add
13           ldl 1
15           add
16           str RR
18           unlink
19           ret
20  main:    ldc 30 ⟵ pc
22           ldc 8
24           bsr f
26           ajs -2
28           ldr RR
30           trap 0
32           halt
```

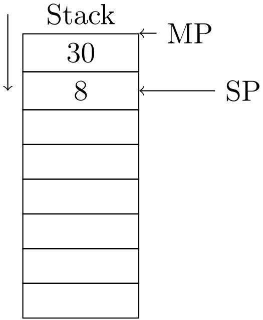


Stack Frame Example

```
f (x, y) {
  var z = 4;
  return x+y+z;
}

main () {
  f(30, 8);
}
```

```
0          bra main
2    f:    link 1
4          ldc 4
6          stl 1
8          ldl -3
10         ldl -2
12         add
13         ldl 1
15         add
16         str RR
18         unlink
19         ret
20   main: ldc 30
22         ldc 8 ← pc
24         bsr f
26         ajs -2
28         ldr RR
30         trap 0
32         halt
```

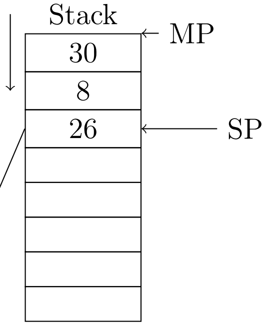


Stack Frame Example

```
f (x, y) {
  var z = 4;
  return x+y+z;
}

main () {
  f(30, 8);
}
```

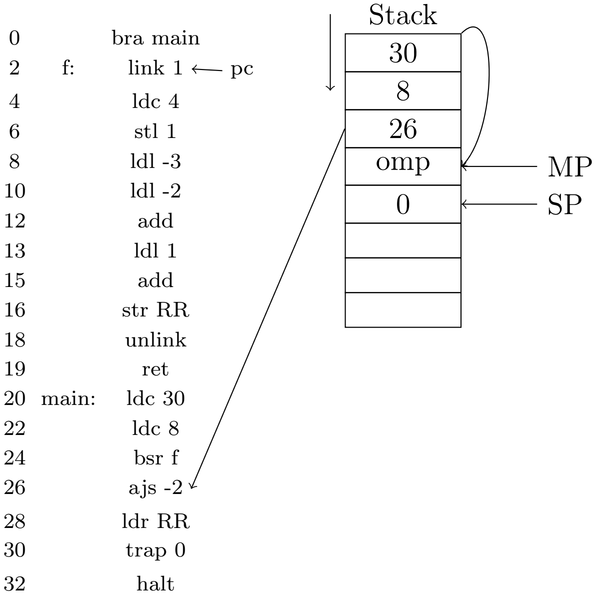
```
0          bra main
2   f:     link 1
4          ldc 4
6          stl 1
8          ldl -3
10         ldl -2
12         add
13         ldl 1
15         add
16         str RR
18         unlink
19         ret
20  main:  ldc 30
22         ldc 8
24         bsr f  ← pc
26         ajs -2
28         ldr RR
30         trap 0
32         halt
```



Stack Frame Example

```
f (x, y) {
  var z = 4;
  return x+y+z;
}

main () {
  f(30, 8);
}
```



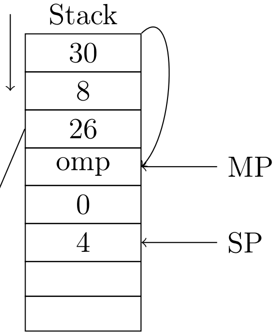
```
0           bra main
2   f:      link 1 ← pc
4           ldc 4
6           stl 1
8           ldl -3
10          ldl -2
12          add
13          ldl 1
15          add
16          str RR
18          unlink
19          ret
20  main:   ldc 30
22          ldc 8
24          bsr f
26          ajs -2
28          ldr RR
30          trap 0
32          halt
```

Stack Frame Example

```
f (x, y) {
  var z = 4;
  return x+y+z;
}

main () {
  f(30, 8);
}
```

```
0            bra main
2    f:      link 1
4            ldc 4 ←— pc
6            stl 1
8            ldl -3
10           ldl -2
12           add
13           ldl 1
15           add
16           str RR
18           unlink
19           ret
20   main:   ldc 30
22           ldc 8
24           bsr f
26           ajs -2
28           ldr RR
30           trap 0
32           halt
```

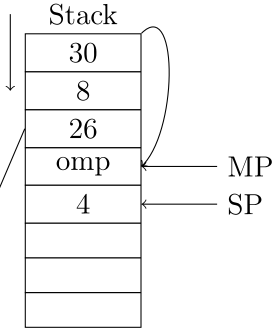


Stack Frame Example

```
f (x, y) {
  var z = 4;
  return x+y+z;
}

main () {
  f(30, 8);
}
```

```
0              bra main
2    f:        link 1
4              ldc 4
6              stl 1  ← pc
8              ldl -3
10             ldl -2
12             add
13             ldl 1
15             add
16             str RR
18             unlink
19             ret
20   main:     ldc 30
22             ldc 8
24             bsr f
26             ajs -2
28             ldr RR
30             trap 0
32             halt
```

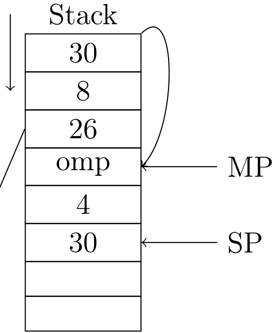


Stack Frame Example

```
f (x, y) {
  var z = 4;
  return x+y+z;
}

main () {
  f(30, 8);
}
```

```
0           bra main
2   f:      link 1
4           ldc 4
6           stl 1
8           ldl -3  ← pc
10          ldl -2
12          add
13          ldl 1
15          add
16          str RR
18          unlink
19          ret
20  main:   ldc 30
22          ldc 8
24          bsr f
26          ajs -2
28          ldr RR
30          trap 0
32          halt
```

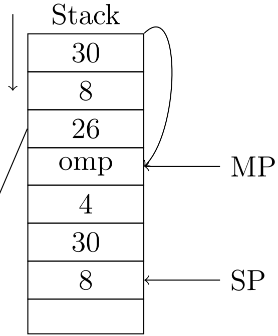


Stack Frame Example

```
f (x, y) {
  var z = 4;
  return x+y+z;
}

main () {
  f(30, 8);
}
```

```
0           bra main
2    f:     link 1
4           ldc 4
6           stl 1
8           ldl -3
10          ldl -2  ← pc
12          add
13          ldl 1
15          add
16          str RR
18          unlink
19          ret
20   main:  ldc 30
22          ldc 8
24          bsr f
26          ajs -2
28          ldr RR
30          trap 0
32          halt
```



Stack

30

8

26

omp ← MP

4

30

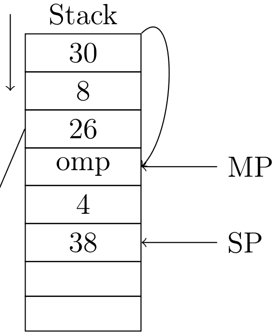
8 ← SP

Stack Frame Example

```
f (x, y) {
  var z = 4;
  return x+y+z;
}

main () {
  f(30, 8);
}
```

```
0           bra main
2   f:      link 1
4           ldc 4
6           stl 1
8           ldl -3
10          ldl -2
12          add  ← pc
13          ldl 1
15          add
16          str RR
18          unlink
19          ret
20  main:   ldc 30
22          ldc 8
24          bsr f
26          ajs -2
28          ldr RR
30          trap 0
32          halt
```

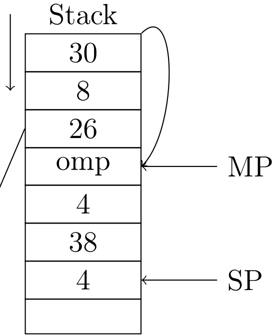


Stack Frame Example

```
f (x, y) {
  var z = 4;
  return x+y+z;
}

main () {
  f(30, 8);
}
```

```
0           bra main
2   f:      link 1
4           ldc 4
6           stl 1
8           ldl -3
10          ldl -2
12          add
13          ldl 1  ← pc
15          add
16          str RR
18          unlink
19          ret
20  main:   ldc 30
22          ldc 8
24          bsr f
26          ajs -2
28          ldr RR
30          trap 0
32          halt
```

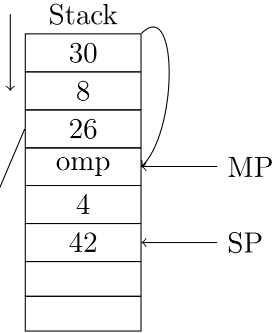


Stack Frame Example

```
f (x, y) {
  var z = 4;
  return x+y+z;
}

main () {
  f(30, 8);
}
```

```
0           bra main
2   f:      link 1
4           ldc 4
6           stl 1
8           ldl -3
10          ldl -2
12          add
13          ldl 1
15          add   <- pc
16          str RR
18          unlink
19          ret
20  main:   ldc 30
22          ldc 8
24          bsr f
26          ajs -2
28          ldr RR
30          trap 0
32          halt
```

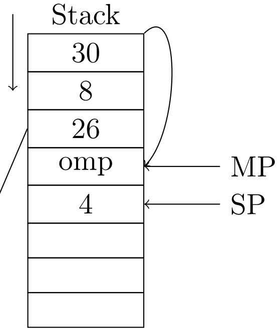


Stack Frame Example

```
f (x, y) {
  var z = 4;
  return x+y+z;
}

main () {
  f(30, 8);
}
```

```
 0           bra main
 2   f:      link 1
 4           ldc 4
 6           stl 1
 8           ldl -3
10           ldl -2
12           add
13           ldl 1
15           add
16           str RR ← pc
18           unlink
19           ret
20   main:   ldc 30
22           ldc 8
24           bsr f
26           ajs -2
28           ldr RR
30           trap 0
32           halt
```



Stack

30

8

26

omp

4

MP

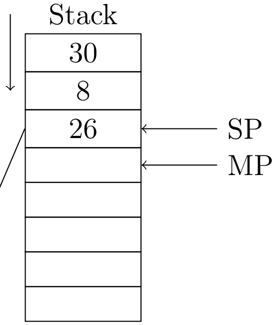
SP

Stack Frame Example

```
f (x, y) {
  var z = 4;
  return x+y+z;
}

main () {
  f(30, 8);
}
```

```
0           bra main
2   f:      link 1
4           ldc 4
6           stl 1
8           ldl -3
10          ldl -2
12          add
13          ldl 1
15          add
16          str RR
18          unlink ← pc
19          ret
20  main:   ldc 30
22          ldc 8
24          bsr f
26          ajs -2
28          ldr RR
30          trap 0
32          halt
```

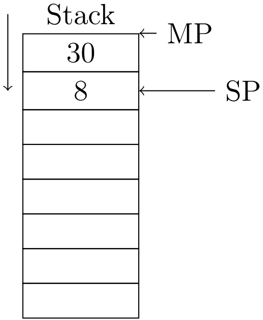


Stack Frame Example

```
f (x, y) {
  var z = 4;
  return x+y+z;
}

main () {
  f(30, 8);
}
```

```
0             bra main
2    f:       link 1
4             ldc 4
6             stl 1
8             ldl -3
10            ldl -2
12            add
13            ldl 1
15            add
16            str RR
18            unlink
19            ret ← pc
20   main:    ldc 30
22            ldc 8
24            bsr f
26            ajs -2
28            ldr RR
30            trap 0
32            halt
```

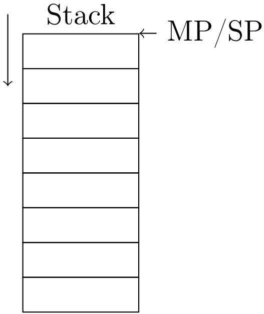


Stack Frame Example

```
f (x, y) {
  var z = 4;
  return x+y+z;
}

main () {
  f(30, 8);
}
```

```
0          bra main
2   f:     link 1
4          ldc 4
6          stl 1
8          ldl -3
10         ldl -2
12         add
13         ldl 1
15         add
16         str RR
18         unlink
19         ret
20  main:  ldc 30
22         ldc 8
24         bsr f
26         ajs -2  ⟵ pc
28         ldr RR
30         trap 0
32         halt
```

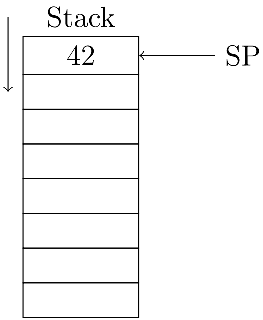


Stack Frame Example

```
f (x, y) {
  var z = 4;
  return x+y+z;
}

main () {
  f(30, 8);
}
```

```
0           bra main
2   f:      link 1
4           ldc 4
6           stl 1
8           ldl -3
10          ldl -2
12          add
13          ldl 1
15          add
16          str RR
18          unlink
19          ret
20  main:   ldc 30
22          ldc 8
24          bsr f
26          ajs -2
28          ldr RR ← pc
30          trap 0
32          halt
```

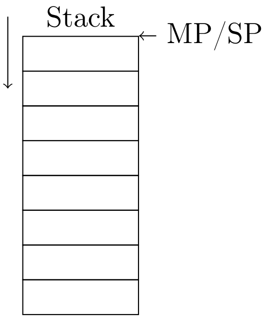


Stack Frame Example

```
f (x, y) {
  var z = 4;
  return x+y+z;
}

main () {
  f(30, 8);
}
```

```
0           bra main
2   f:      link 1
4           ldc 4
6           stl 1
8           ldl -3
10          ldl -2
12          add
13          ldl 1
15          add
16          str RR
18          unlink
19          ret
20  main:   ldc 30
22          ldc 8
24          bsr f
26          ajs -2
28          ldr RR
30          trap 0 ⟵ pc
32          halt
```

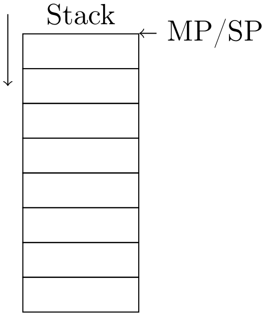


Stack Frame Example

```
f (x, y) {
  var z = 4;
  return x+y+z;
}

main () {
  f(30, 8);
}
```

```
0            bra main
2    f:      link 1
4            ldc 4
6            stl 1
8            ldl -3
10           ldl -2
12           add
13           ldl 1
15           add
16           str RR
18           unlink
19           ret
20   main:   ldc 30
22           ldc 8
24           bsr f
26           ajs -2
28           ldr RR
30           trap 0
32           halt ← pc
```



SSM Tips (4)

Globals

- When to calculate

SSM Tips (4)

Globals

- When to calculate
- How to retrieve

SSM Tips (4)

Globals

- When to calculate
- How to retrieve
- Addresses in spare register

LLVM

LLVM

LLVM

- Optimiser and code generator

LLVM

- Optimiser and code generator and JIT compiler, dynamic linker, debugger, …
- Low level virtual machine

LLVM

- Optimiser and code generator and JIT compiler, dynamic linker, debugger, ...
- ~~Low level virtual machine~~ orphan acronym

LLVM

- Optimiser and code generator and JIT compiler, dynamic linker, debugger, . . .
- ~~Low level virtual machine~~ orphan acronym

LLVM

- Optimiser and code generator and JIT compiler, dynamic linker, debugger, . . .
- ~~Low level virtual machine~~ orphan acronym
- Famous C compiler: clang

LLVM

- Optimiser and code generator and JIT compiler, dynamic linker, debugger, . . .
- ~~Low level virtual machine~~ orphan acronym
- Famous C compiler: clang
- LLVM-IR:high-level assembly
  - Typed

LLVM

- Optimiser and code generator and JIT compiler, dynamic linker, debugger, ...
- ~~Low level virtual machine~~ orphan acronym
- Famous C compiler: clang
- LLVM-IR:high-level assembly
  - Typed
  - Rich data types (structs)

LLVM

- Optimiser and code generator and JIT compiler, dynamic linker, debugger, . . .
- ~~Low level virtual machine~~ orphan acronym
- Famous C compiler: clang
- LLVM-IR:high-level assembly
  - Typed
  - Rich data types (structs)
  - Rich instruction set

LLVM

- Optimiser and code generator and JIT compiler, dynamic linker, debugger, . . .
- ~~Low level virtual machine~~ orphan acronym
- Famous C compiler: clang
- LLVM-IR:high-level assembly
  - Typed
  - Rich data types (structs)
  - Rich instruction set
  - Functions

LLVM

- Optimiser and code generator and JIT compiler, dynamic linker, debugger, . . .
- ~~Low level virtual machine~~ orphan acronym
- Famous C compiler: clang
- LLVM-IR:high-level assembly
  - Typed
  - Rich data types (structs)
  - Rich instruction set
  - Functions
  - FFI with C/C++

LLVM

- Optimiser and code generator and JIT compiler, dynamic linker, debugger, . . .
- ~~Low level virtual machine~~ orphan acronym
- Famous C compiler: clang
- LLVM-IR:high-level assembly
    - Typed
    - Rich data types (structs)
    - Rich instruction set
    - Functions
    - FFI with C/C++
    - But no loops

How to use LLVM-IR

How to use LLVM-IR

Bitcode

- Binary encoding
- Fast parsing
- More difficult to output
- Not stable*

How to use LLVM-IR

Bitcode

- Binary encoding
- Fast parsing
- More difficult to output
- Not stable*

Text

- Pretty text encoding
- Slow parsing
- Easy to generate
- Not stable*

How to use LLVM-IR

Bitcode

- Binary encoding
- Fast parsing
- More difficult to output
- Not stable*

Text

- Pretty text encoding
- Slow parsing
- Easy to generate
- Not stable*

C++ classes

- Direct in-memory encoding of the IR
- Requires an FFI to the C++ library
- Fairly convoluted
- But stable

Example

Hello World!

Example

Hello World!

```
; Define the target
target triple = "x86_64-pc-linux-gnu"

; Declare the string constant as a global constant.
@.str = private unnamed_addr constant [14 x i8] c"Hello World!\0A\00"

; External declaration of the puts function
declare i32 @puts(ptr) nounwind

; Definition of main function
define i32 @main() {
  ; Call puts function to write out the string to stdout.
  call i32 @puts(ptr @.str)
  ret i32 0
}
```

Example

Hello World!

```
; Definition of main function
define i32 @main() {
  ; Call puts function to write out the string to stdout.
  call i32 @puts(ptr @.str)
  ret i32 0
}
```

Example

Hello World!

```
; Definition of main function
define i32 @main() {
  ; Call puts function to write out the string to stdout.
  call i32 @puts(ptr @.str)
  ret i32 0
}
```

How to run (on Linux at least)

```
$ clang hello.ll -o hello
$ ./hello
Hello World!
```

Example

Use printf

Example

Use printf

```
; Declare the string constant as a global constant.
@.str = private unnamed_addr constant [4 x i8] c"%d\0A\00"

; External declaration of the puts function
declare i32 @printf(ptr, ...) nounwind

; Definition of main function
define i32 @main() {
  %1 = add i32 38, 4
  %2 = add i32 %1, %1
  call i32 @printf(ptr @.str, i32 %2)
  ret i32 0
}
```

Example

Use printf

```
; External declaration of the puts function
declare i32 @printf(ptr, ...) nounwind

; Definition of main function
define i32 @main() {
  %1 = add i32 38, 4
  %2 = add i32 %1, %1
  call i32 @printf(ptr @.str, i32 %2)
  ret i32 0
}
```

Example

Use printf

```
; External declaration of the puts function
declare i32 @printf(ptr, ...) nounwind

; Definition of main function
define i32 @main() {
  %1 = add i32 38, 4
  %2 = add i32 %1, %1
  call i32 @printf(ptr @.str, i32 %2)
  ret i32 0
}
```

Output

```
$ clang hello2.ll -o hello2
$ ./hello2
42
```

Example

Static Single assignment

Example

Static Single assignment

```
; Definition of main function
define i32 @main() {
  %1 = add i32 38, 4
  %2 = add i32 %1, %1
  %2 = add i32 %1, 84
  call i32 @printf(ptr @.str, i32 %2)
  ret i32 0
}
```

Example

Static Single assignment

```
; Definition of main function
define i32 @main() {
  %1 = add i32 38, 4
  %2 = add i32 %1, %1
  %2 = add i32 %1, 84
  call i32 @printf(ptr @.str, i32 %2)
  ret i32 0
}
```

Output

```
$ clang ssa.ll -o ssa
ssa.ll:15:2: error: instruction expected to be numbered '%3' or greater
   15 |        %2 = add i32 %1, 84
      |        ^
1 error generated.
```

Static Single assignment

Pragmatic overview

- You can assign a register only once
- i.e. pick a fresh one for each intermediate value
- So `f (x) { return x + (1 + 2) * (3 - 4);}` translates to:

```
define i32 @f(i32 %x) {
  %1 = add i32 1, 2
  %2 = sub i32 3, 4
  %3 = mul i32 %1, %2
  %4 = add i32 %x, %3
  ret i32 %4
}
```

Mapping high-level constructs on LLVM-IR

- Source: `https://llvm.org/docs/LangRef.html`

Mapping high-level constructs on LLVM-IR

- Source: `https://llvm.org/docs/LangRef.html`

Mapping high-level constructs on LLVM-IR

- Source: `https://llvm.org/docs/LangRef.html`

  ...

Mapping high-level constructs on LLVM-IR

- Source: `https://llvm.org/docs/LangRef.html`

  ...

  ±400 pages...

Mapping high-level constructs on LLVM-IR

- Source: `https://llvm.org/docs/LangRef.html`

  . . .

  ±400 pages. . . Useful if you know what you are looking for.
- Source2:

  `https://mapping-high-level-constructs-to-llvm-ir.readthedocs.io`

Mapping high-level constructs on LLVM-IR

- Source: `https://llvm.org/docs/LangRef.html`

  ...

  ±400 pages... Useful if you know what you are looking for.
- Source2:

  `https://mapping-high-level-constructs-to-llvm-ir.readthedocs.io`

Mapping high-level constructs on LLVM-IR

- Source: `https://llvm.org/docs/LangRef.html`

  . . .

  ±400 pages. . . Useful if you know what you are looking for.
- Source2:

  `https://mapping-high-level-constructs-to-llvm-ir.readthedocs.io`

  Maps all needed constructs, e.g.
- Local variables

Mapping high-level constructs on LLVM-IR

- Source: `https://llvm.org/docs/LangRef.html`
  ...
  ±400 pages... Useful if you know what you are looking for.
- Source2:
  `https://mapping-high-level-constructs-to-llvm-ir.readthedocs.io`
  Maps all needed constructs, e.g.
- Local variables

Mapping high-level constructs on LLVM-IR

- Source: `https://llvm.org/docs/LangRef.html`
  . . .
  ±400 pages. . . Useful if you know what you are looking for.
- Source2:
  `https://mapping-high-level-constructs-to-llvm-ir.readthedocs.io`
  Maps all needed constructs, e.g.
- Local variables
  use **alloca** to allocate on the stack
- If statements

Mapping high-level constructs on LLVM-IR

- Source: `https://llvm.org/docs/LangRef.html`
  ...
  ±400 pages... Useful if you know what you are looking for.
- Source2:
  `https://mapping-high-level-constructs-to-llvm-ir.readthedocs.io`
  Maps all needed constructs, e.g.
- Local variables
  use **alloca** to allocate on the stack
- If statements

Mapping high-level constructs on LLVM-IR

- Source: `https://llvm.org/docs/LangRef.html`
  ...
  ±400 pages... Useful if you know what you are looking for.
- Source2:
  `https://mapping-high-level-constructs-to-llvm-ir.readthedocs.io`
  Maps all needed constructs, e.g.
- Local variables
  use **alloca** to allocate on the stack
- If statements
  use labels (`lbl:`) and branching (**br**) but be aware of SSA!
- Etcetera...

Conclusion

Conclusion

Assignment

- Start implementing your code generator

Conclusion

Assignment

- Start implementing your code generator
- SSM:

Conclusion

Assignment

- Start implementing your code generator
- SSM:
  - Reference and interpreter:
    `https://gitlab.science.ru.nl/compilerconstruction/ssm`

Conclusion

Assignment

- Start implementing your code generator
- SSM:
  - Reference and interpreter: `https://gitlab.science.ru.nl/compilerconstruction/ssm`
  - Read the instruction documentation from front to back

Conclusion

Assignment

- Start implementing your code generator
- SSM:
  - Reference and interpreter: `https://gitlab.science.ru.nl/compilerconstruction/ssm`
  - Read the instruction documentation from front to back
  - Read about the registers

Conclusion

Assignment

- Start implementing your code generator
- SSM:
  - Reference and interpreter:
    `https://gitlab.science.ru.nl/compilerconstruction/ssm`
  - Read the instruction documentation from front to back
  - Read about the registers
  - Read the examples

Conclusion

Assignment

- Start implementing your code generator
- SSM:
  - Reference and interpreter: `https://gitlab.science.ru.nl/compilerconstruction/ssm`
  - Read the instruction documentation from front to back
  - Read about the registers
  - Read the examples
- LLVM:

Conclusion

Assignment

- Start implementing your code generator
- SSM:
  - Reference and interpreter: `https://gitlab.science.ru.nl/compilerconstruction/ssm`
  - Read the instruction documentation from front to back
  - Read about the registers
  - Read the examples
- LLVM:
  - Reference: `https://llvm.org/docs/LangRef.html`

Conclusion

Assignment

- Start implementing your code generator
- SSM:
  - Reference and interpreter: `https://gitlab.science.ru.nl/compilerconstruction/ssm`
  - Read the instruction documentation from front to back
  - Read about the registers
  - Read the examples
- LLVM:
  - Reference: `https://llvm.org/docs/LangRef.html`
  - Pragmatic: `https://mapping-high-level...readthedocs.io`

Conclusion

Assignment

- Start implementing your code generator
- SSM:
  - Reference and interpreter: `https://gitlab.science.ru.nl/compilerconstruction/ssm`
  - Read the instruction documentation from front to back
  - Read about the registers
  - Read the examples
- LLVM:
  - Reference: `https://llvm.org/docs/LangRef.html`
  - Pragmatic: `https://mapping-high-level....readthedocs.io`
- Next week: extension ideas

Conclusion

Assignment

- Start implementing your code generator
- SSM:
  - Reference and interpreter: `https://gitlab.science.ru.nl/compilerconstruction/ssm`
  - Read the instruction documentation from front to back
  - Read about the registers
  - Read the examples
- LLVM:
  - Reference: `https://llvm.org/docs/LangRef.html`
  - Pragmatic: `https://mapping-high-level....readthedocs.io`
- Next week: extension ideas
- ...

Conclusion

Assignment

- Start implementing your code generator
- SSM:
  - Reference and interpreter: `https://gitlab.science.ru.nl/compilerconstruction/ssm`
  - Read the instruction documentation from front to back
  - Read about the registers
  - Read the examples
- LLVM:
  - Reference: `https://llvm.org/docs/LangRef.html`
  - Pragmatic: `https://mapping-high-level....readthedocs.io`
- Next week: extension ideas
- ...
- Think about an extension

Conclusion

Assignment

- Start implementing your code generator
- SSM:
  - Reference and interpreter: `https://gitlab.science.ru.nl/compilerconstruction/ssm`
  - Read the instruction documentation from front to back
  - Read about the registers
  - Read the examples
- LLVM:
  - Reference: `https://llvm.org/docs/LangRef.html`
  - Pragmatic: `https://mapping-high-level....readthedocs.io`
- Next week: extension ideas
- ...
- Think about an extension
- Deadline: day after next lecture.