ARCHITECTURES DES COMPOSANTS PROGRAMMABLES

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

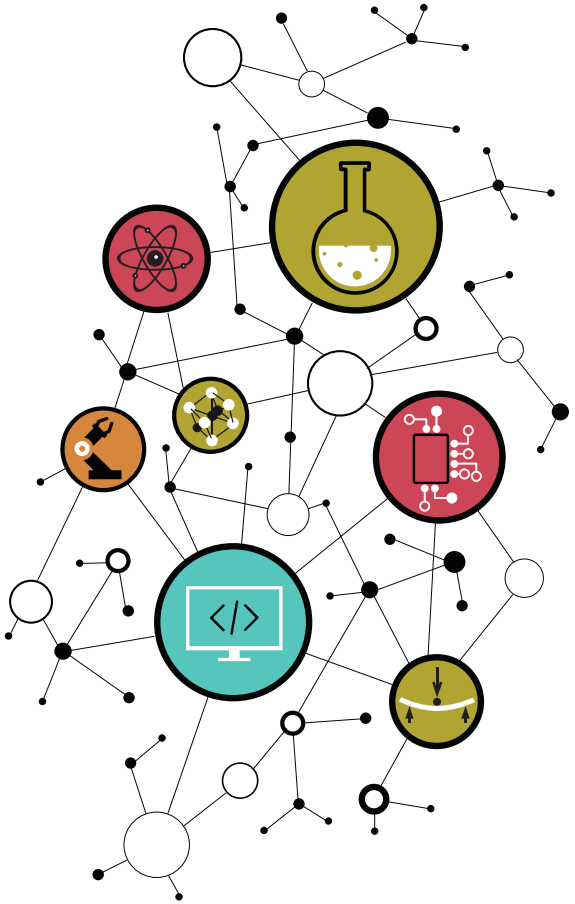
Correction TP

4 – Allocations automatiques et segment de pile

4.4. Appel et paramètres de fonction

2020-2021

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE



TP 4 – Allocations automatiques et segment de pile

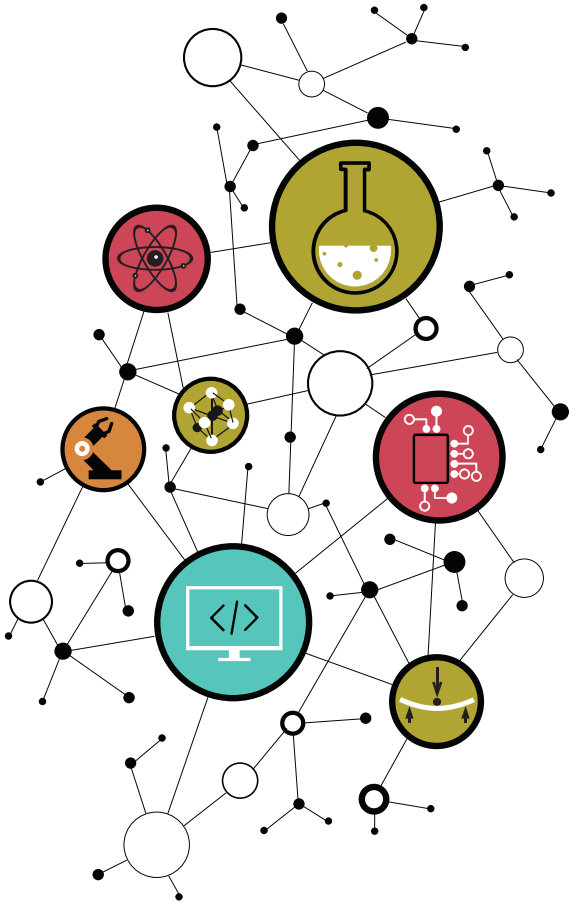
4.4. Appel et paramètres de fonction

Le code présenté est celui issu du fichier `function_parameters.c`, au détail près que le processus de compilation n'a pas été arrêté avant la phase d'assemblage comme initialement demandé.

En effet il s'agit ici du code obtenu après compilation complète, puis désassemblage de l'exécutable final :

```
gcc function_parameters.s -o function_parameters
objdump -S function_parameters
```

Informations sur le corrigé



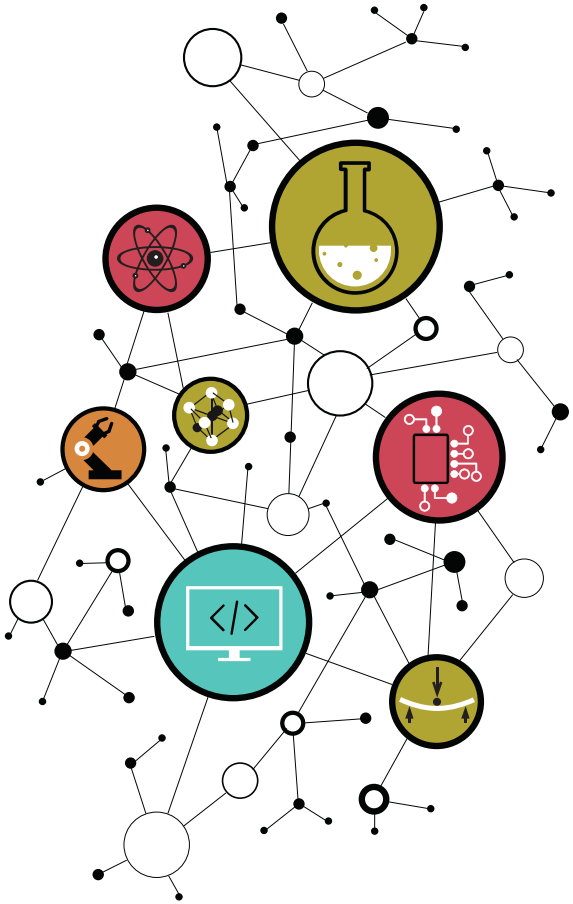
La version désassemblée a pour but de montrer le code assembleur associé aux adresses des instructions (plus pratique pour représenter l'utilité du pointeur d'instructions IP).

D'autre part, le code a été manuellement modifié pour ajouter systématiquement l'information de taille de l'opérande

- par ex : `pushb` , `pushw` , `pushl` , `pushq` , …
- Indique l'instruction `push` suivie de la taille de l'opérande manipulé :
- `b = byte = 1 octet, w = word = 2o, l = long = 4o, q = quad = 8o`

Note : Pour des raisons de clarté pédagogique, la ligne de code surlignée est celle en cours d'exécution (et non la prochaine à exécuter comme le fait classiquement un *debugger*).

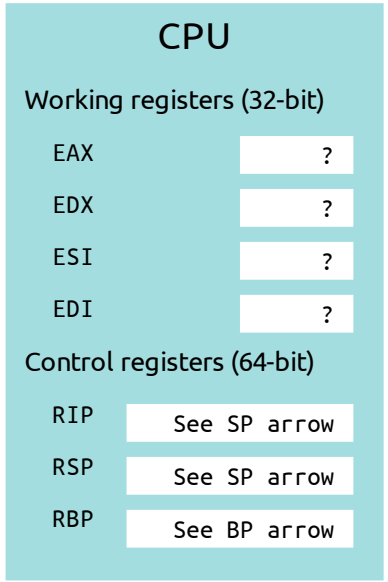
function_parameters.c

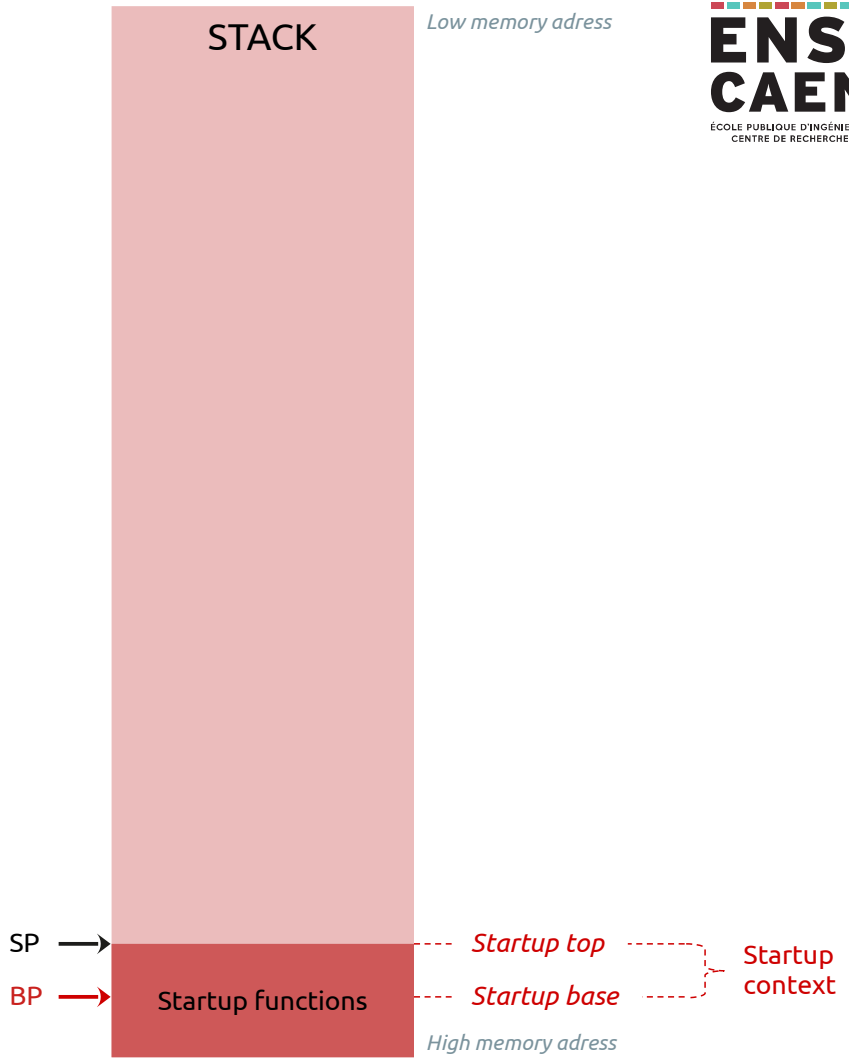


```c
/* ANSI C standard syntax - 1989 */
void function_1 (void) ;
int  function_2 (int a, int b, int c);

int main(void)
{
    function_1();
    return 0;
}

void function_1 (void)
{
    int ret_1;
    ret_1 = function_2 (1, 2, 3);
}

/* K&R C original syntax - 1978 */
int function_2 (a_2, b_2, c_2)
int a_2;
int b_2;
int c_2;
{
    return a_2 + b_2 + c_2;
}
```

Disassembly

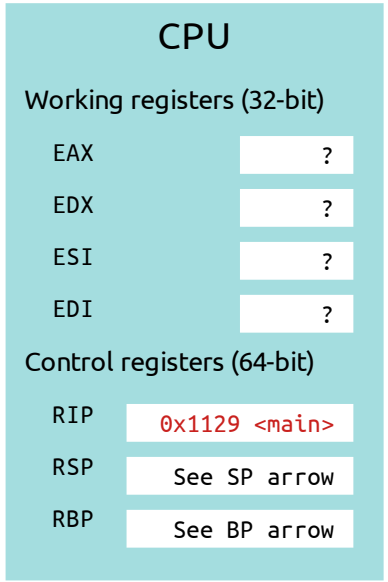
```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```

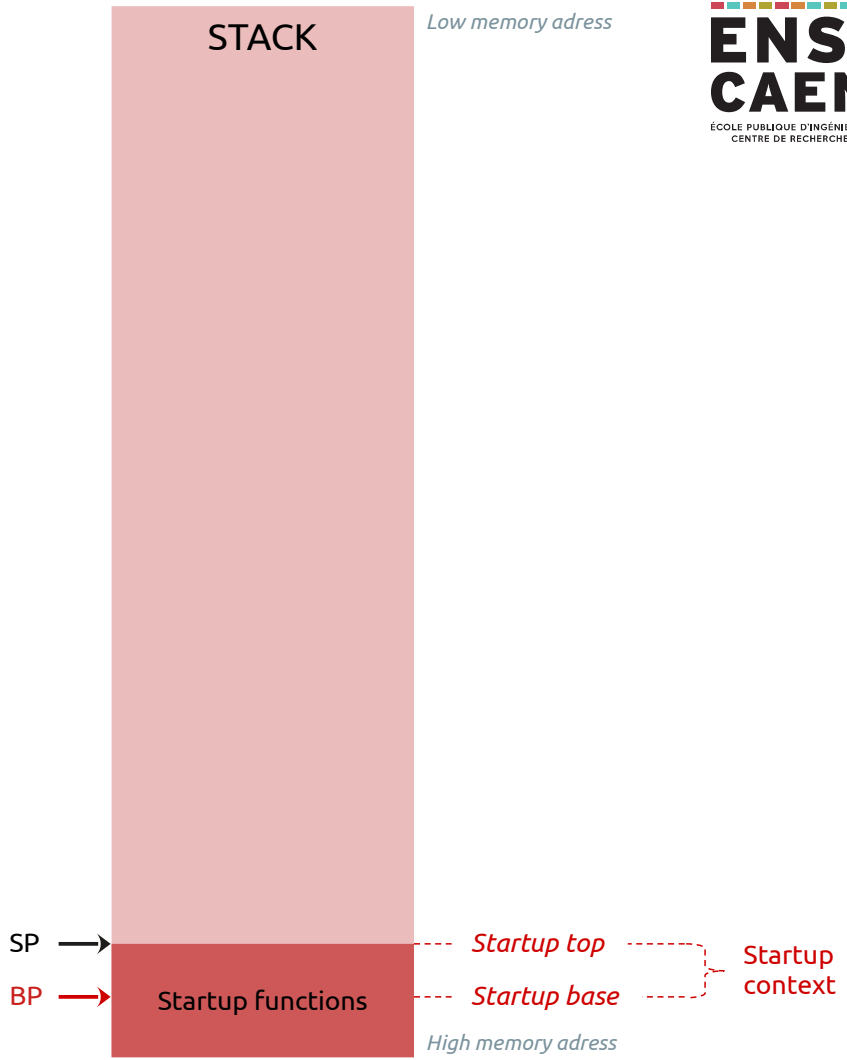




Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```





Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```

CPU

Working registers (32-bit)

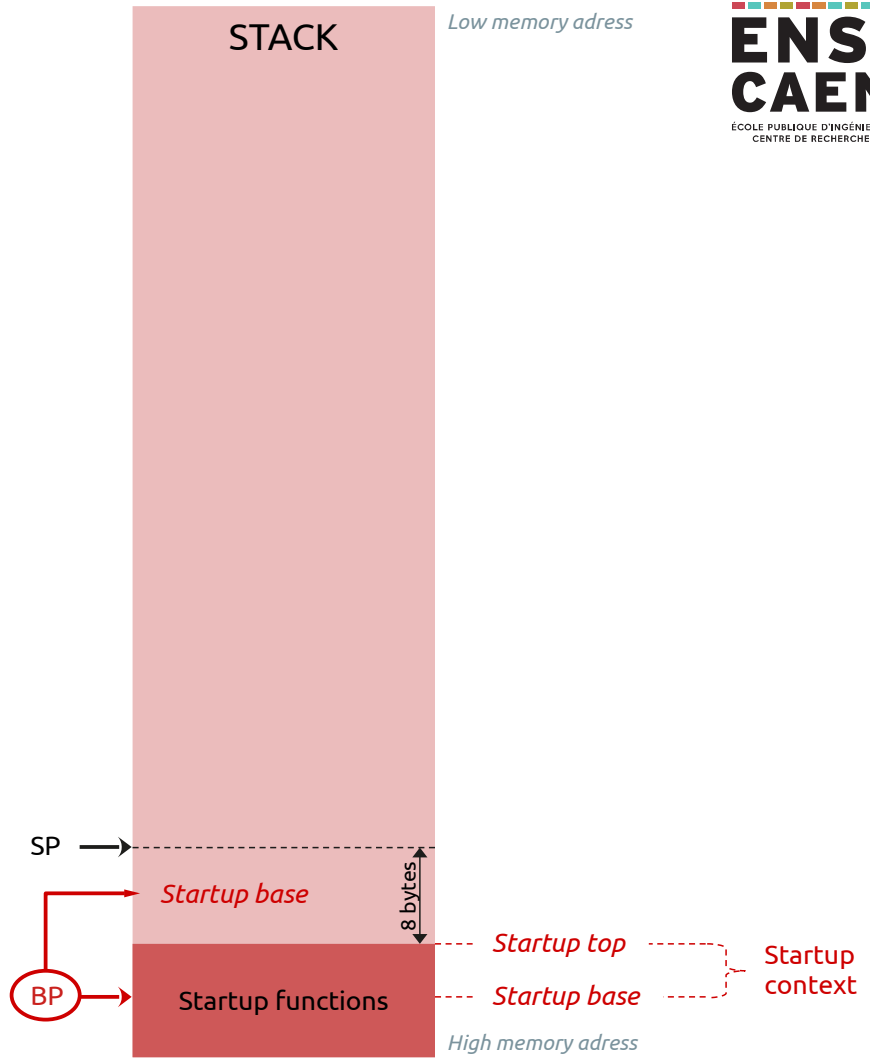
EAX	?
EDX	?
ESI	?
EDI	?

Control registers (64-bit)

RIP	0x112a
RSP	See SP arrow
RBP	See BP arrow

INSTRUCTION EXPLAINED

```
pushq %rbp

is equivalent to:

subq $8, %rsp
movq %rbp, (%rsp)
```



Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```

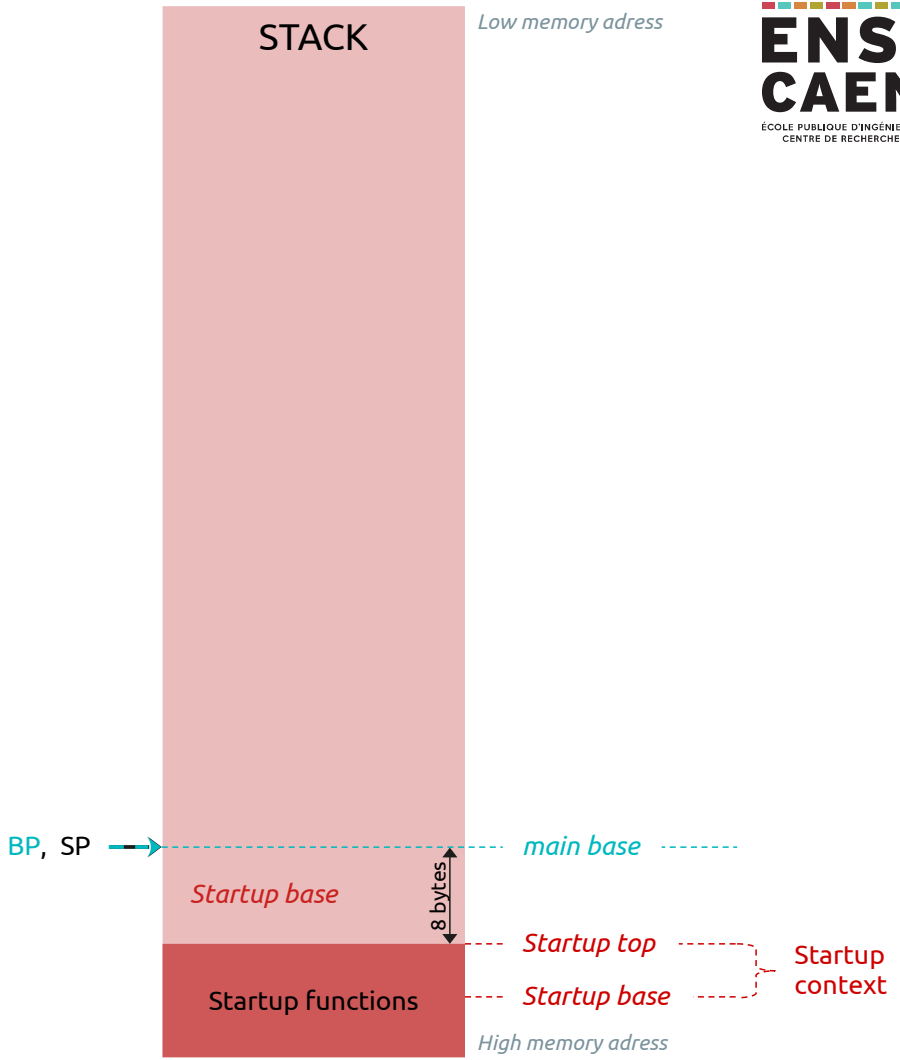
CPU

Working registers (32-bit)

EAX	?
EDX	?
ESI	?
EDI	?

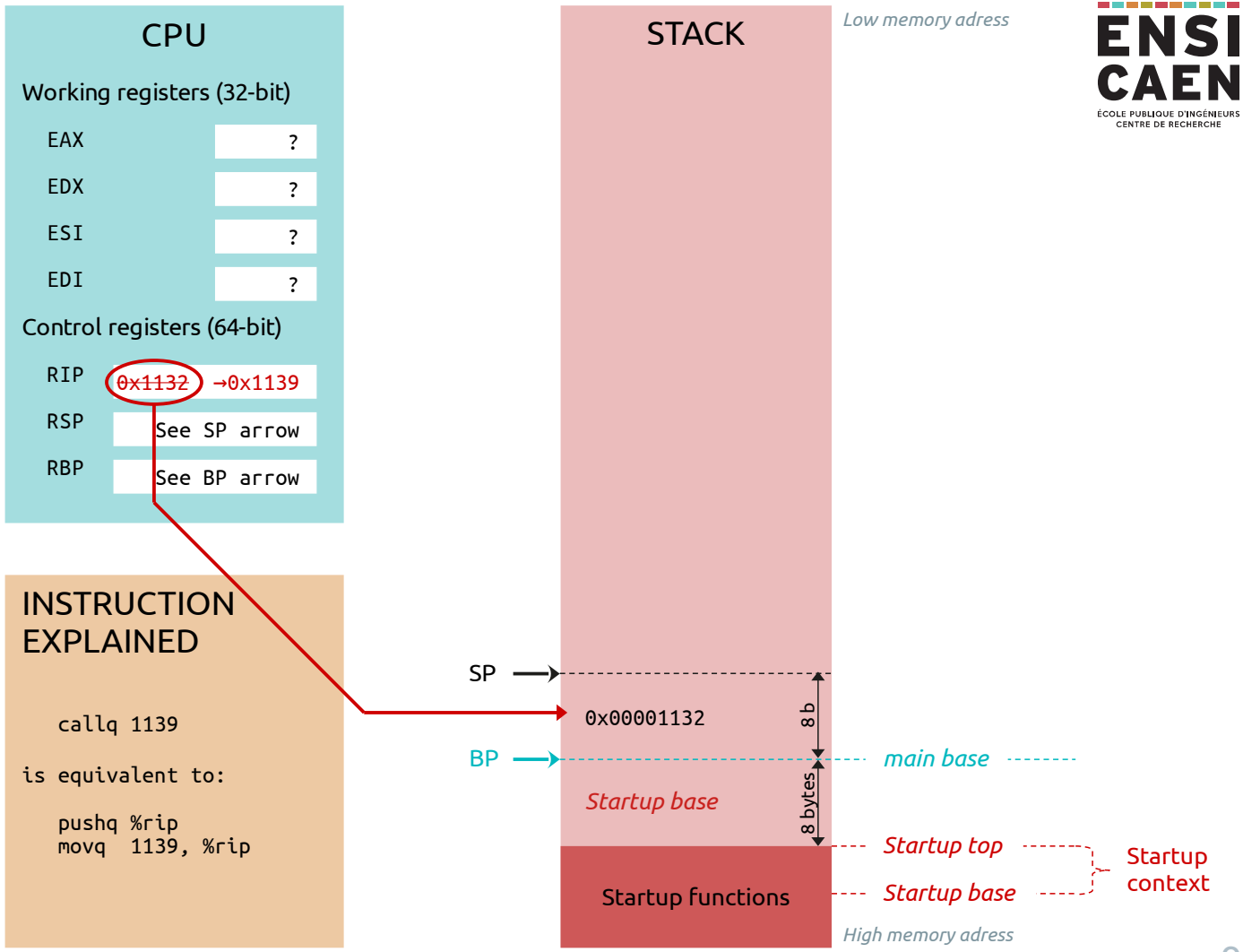
Control registers (64-bit)

RIP	0x112d
RSP	See SP arrow
RBP	See BP arrow



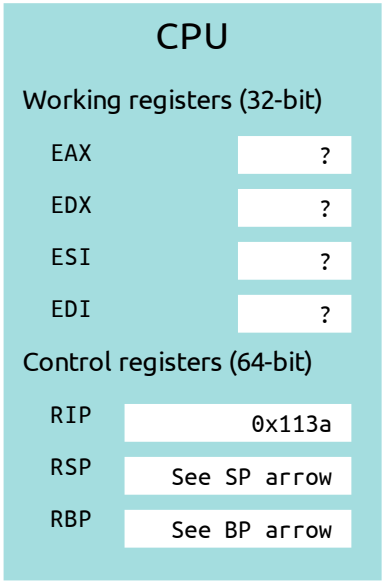
Disassembly

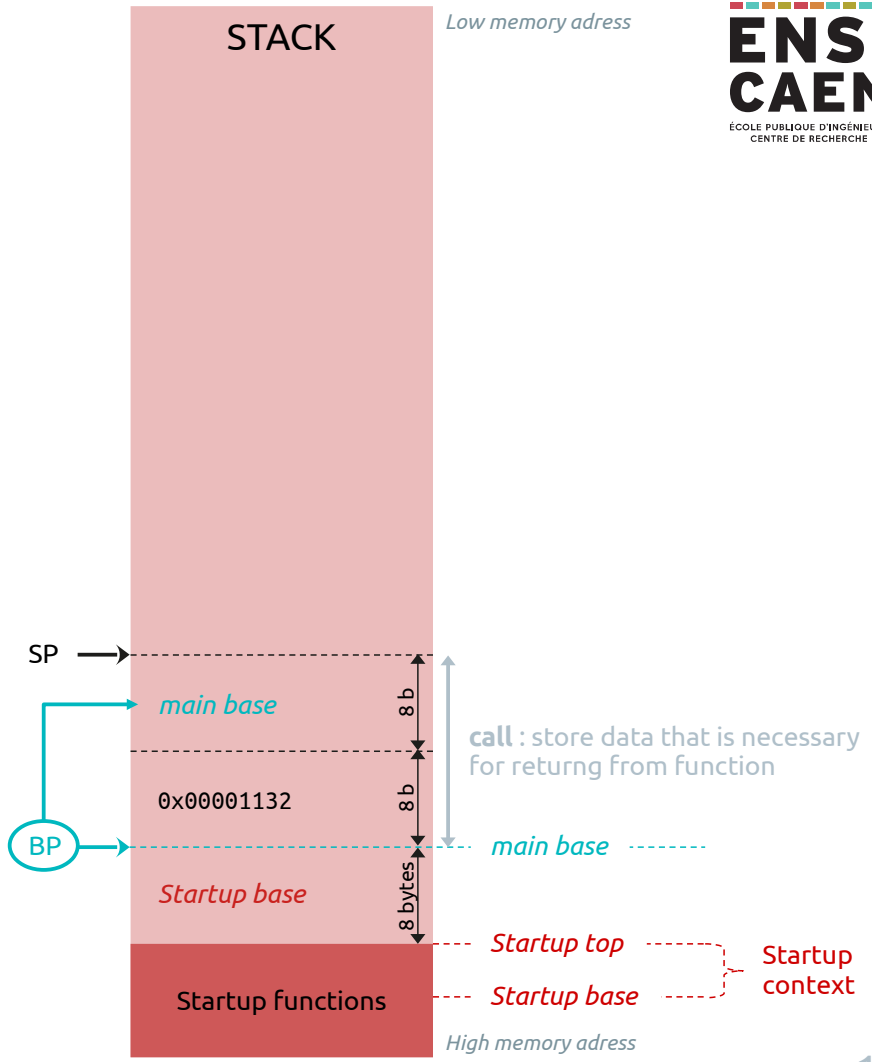
```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```



Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```





Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```

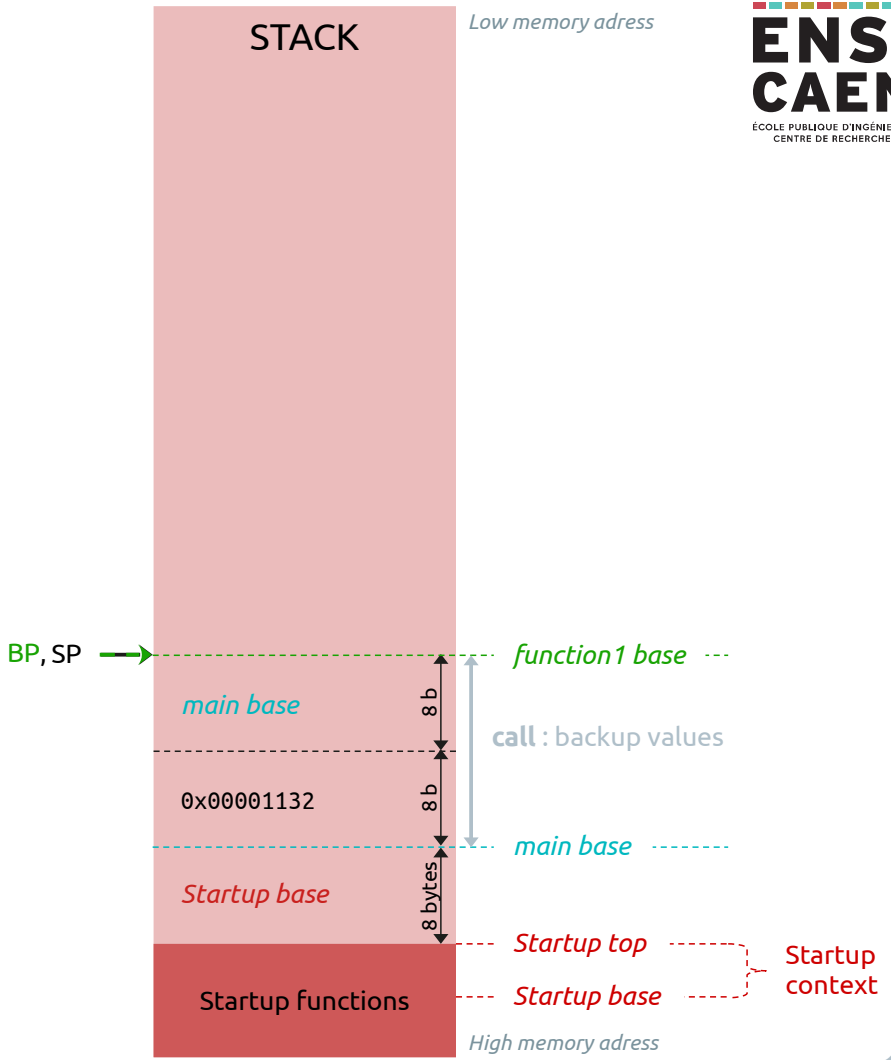
CPU

Working registers (32-bit)

EAX	?
EDX	?
ESI	?
EDI	?

Control registers (64-bit)

RIP	0x113d
RSP	See SP arrow
RBP	See BP arrow



Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```

CPU

Working registers (32-bit)

EAX	?
EDX	?
ESI	?
EDI	?

Control registers (64-bit)

RIP	0x1141
RSP	See SP arrow
RBP	See BP arrow

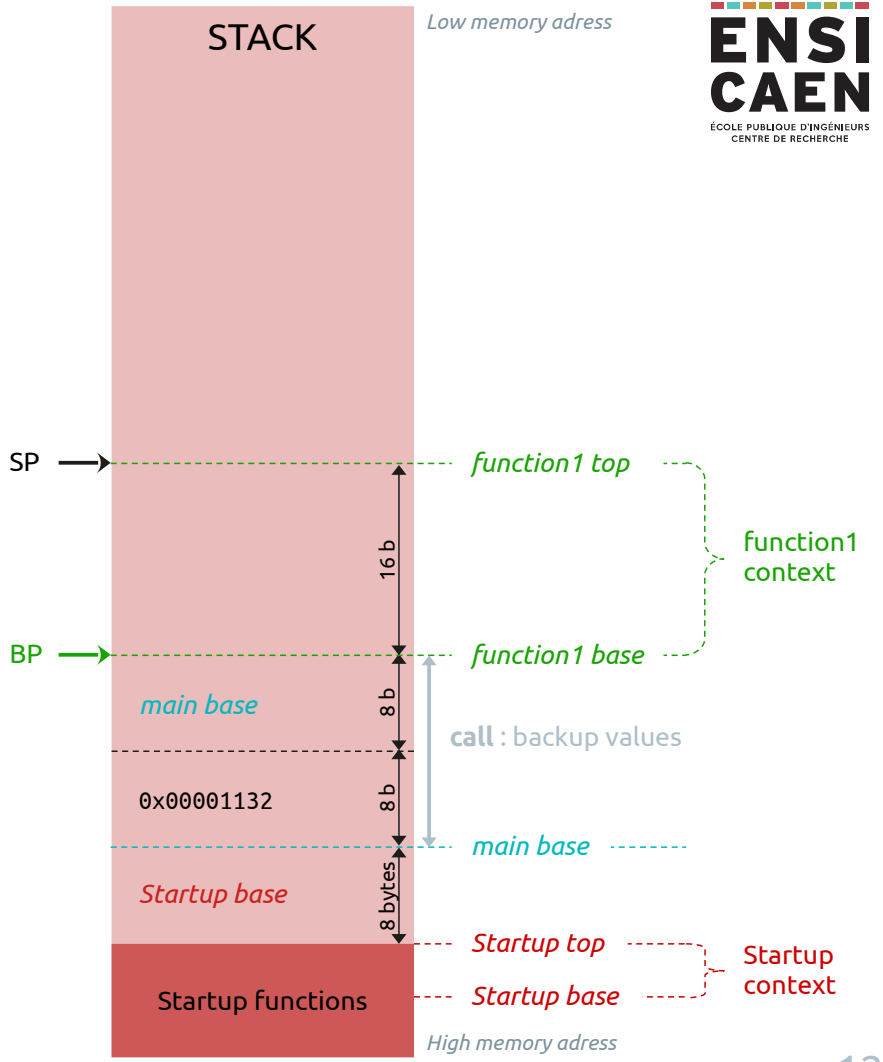
INSTRUCTION EXPLAINED

```
subq $0x10, %rsp
```

uses hexadecimal notation

it is equivalent to:

```
subq $16, %rsp
```



Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```

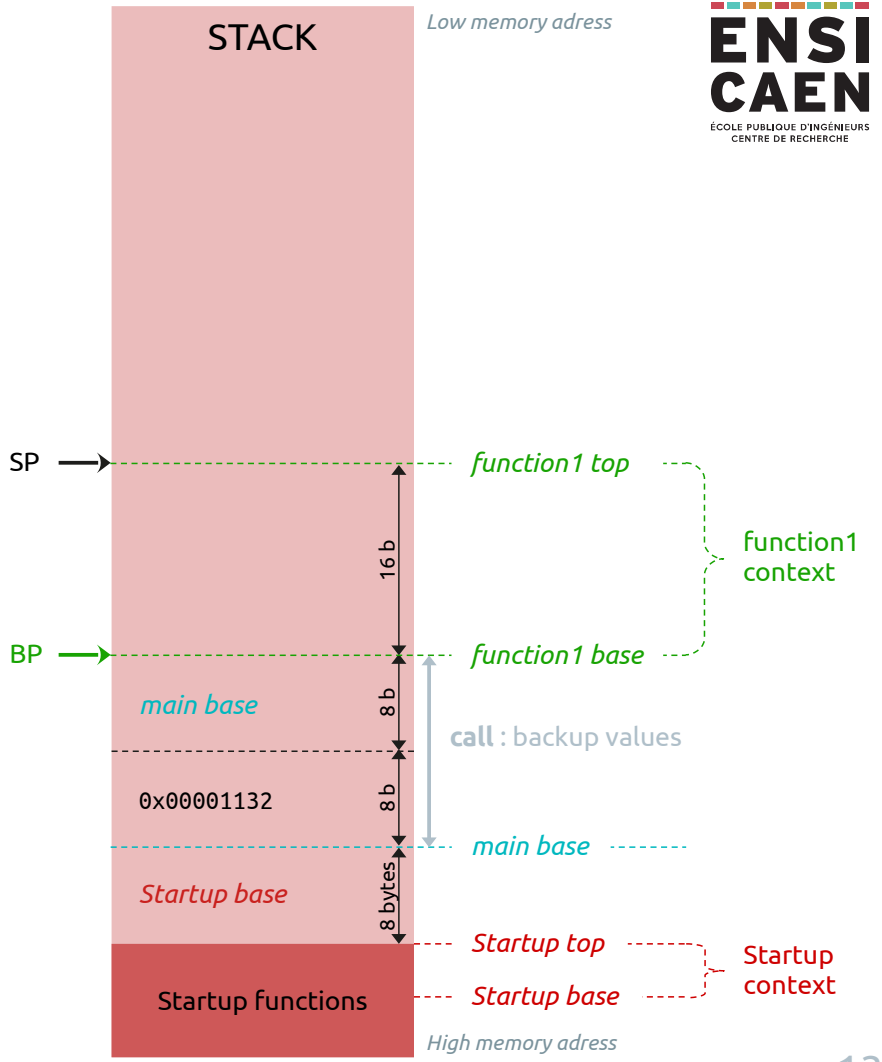
CPU

Working registers (32-bit)

EAX	?
EDX	0x03
ESI	?
EDI	?

Control registers (64-bit)

RIP	0x1146
RSP	See SP arrow
RBP	See BP arrow



Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```

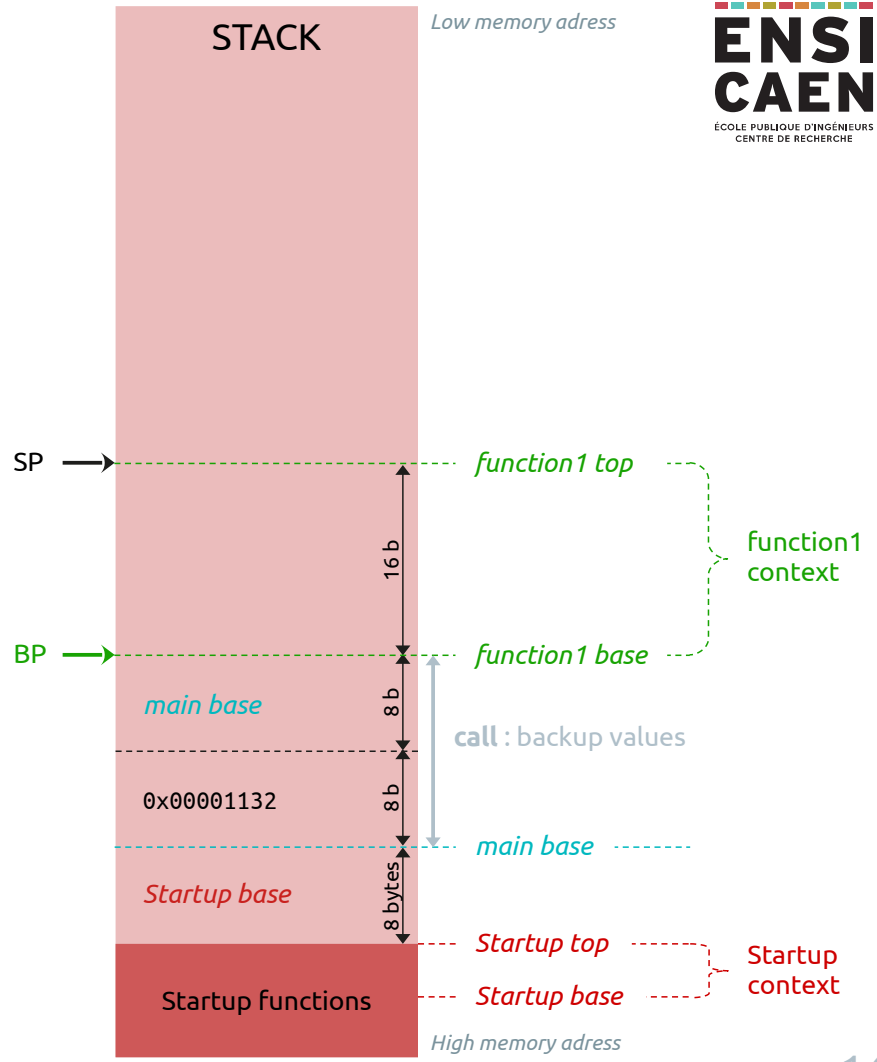
CPU

Working registers (32-bit)

EAX	?
EDX	0x03
ESI	0x02
EDI	?

Control registers (64-bit)

RIP	0x114b
RSP	See SP arrow
RBP	See BP arrow



Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```

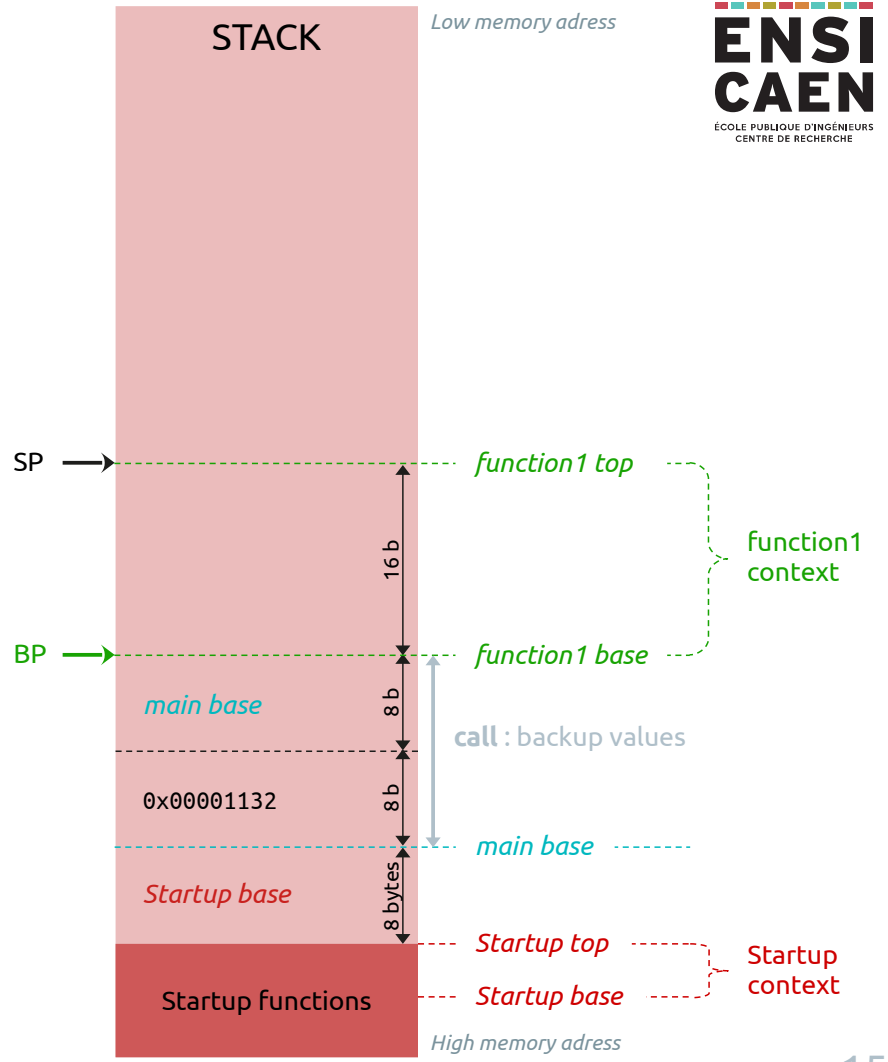
CPU

Working registers (32-bit)

EAX	?
EDX	0x03
ESI	0x02
EDI	0x01

Control registers (64-bit)

RIP	0x1150
RSP	See SP arrow
RBP	See BP arrow



Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```

CPU

Working registers (32-bit)

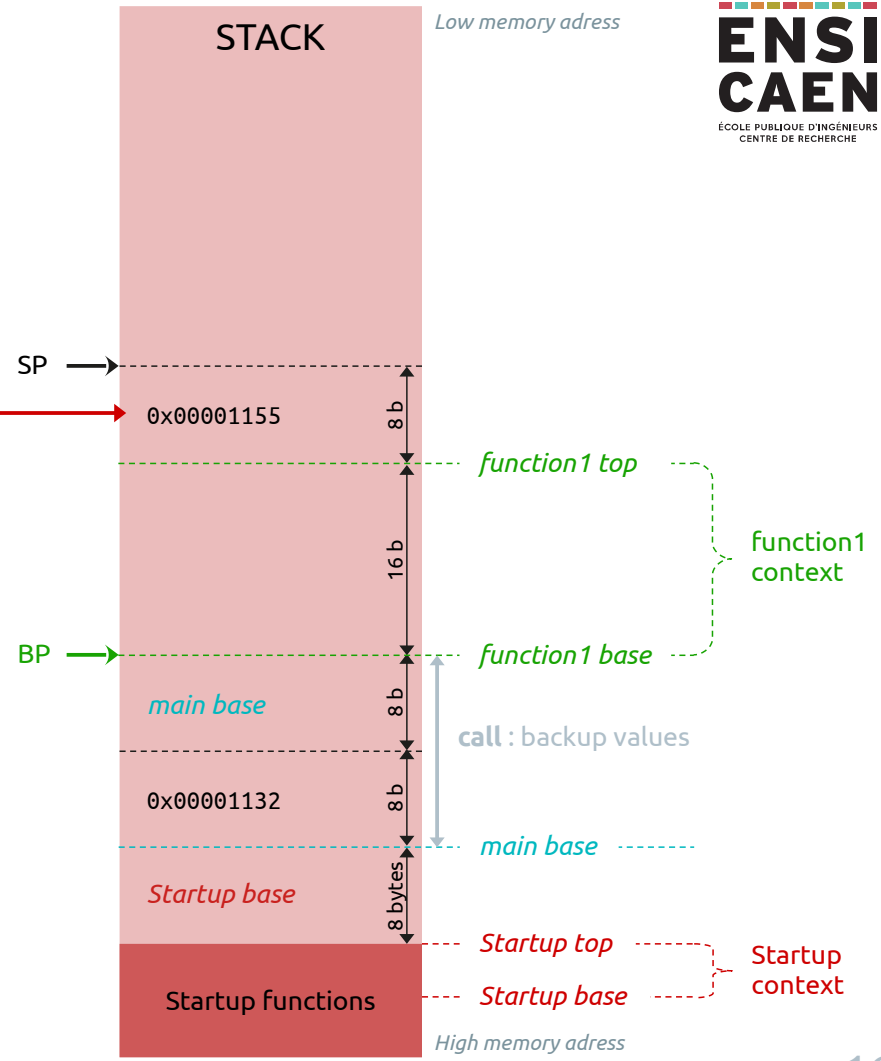
EAX	?
EDX	0x03
ESI	0x02
EDI	0x01

Control registers (64-bit)

RIP	~~0x1155~~ →0x115b
RSP	See SP arrow
RBP	See BP arrow

INSTRUCTION EXPLAINED

```
callq 115b

is equivalent to:

pushq %rip
movq  115b, %rip
```



Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```

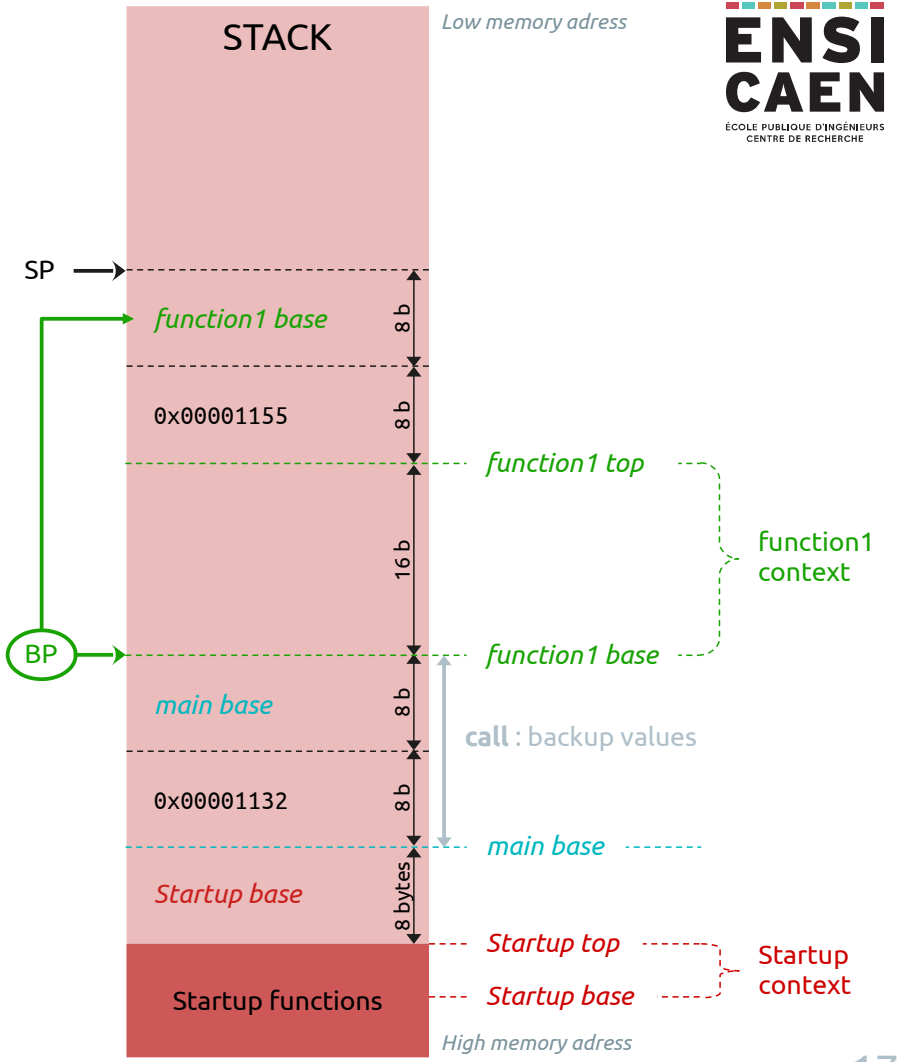
CPU

Working registers (32-bit)

EAX	?
EDX	0x03
ESI	0x02
EDI	0x01

Control registers (64-bit)

RIP	0x115c
RSP	See SP arrow
RBP	See BP arrow



Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```

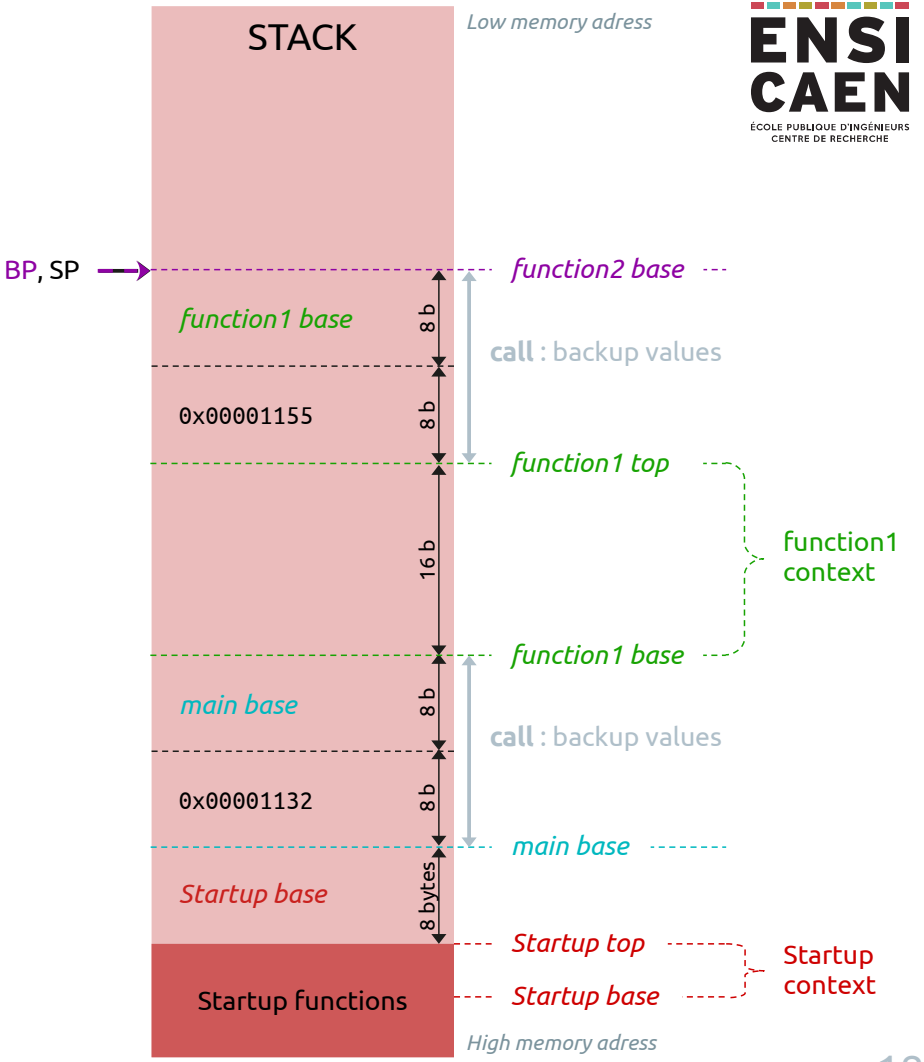
CPU

Working registers (32-bit)

EAX	?
EDX	0x03
ESI	0x02
EDI	0x01

Control registers (64-bit)

RIP	0x115f
RSP	See SP arrow
RBP	See BP arrow



Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```

CPU

Working registers (32-bit)

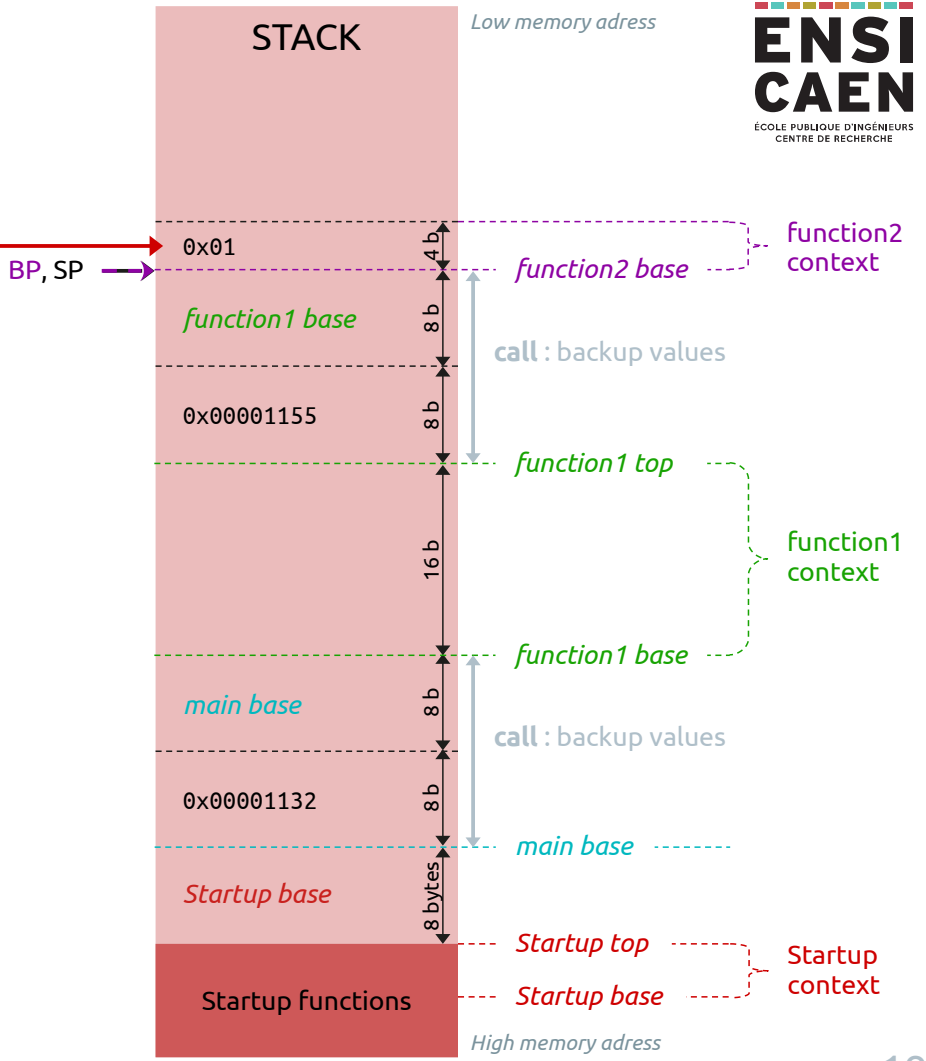
EAX	?
EDX	0x03
ESI	0x02
EDI	0x01

Control registers (64-bit)

RIP	0x1162
RSP	See SP arrow
RBP	See BP arrow

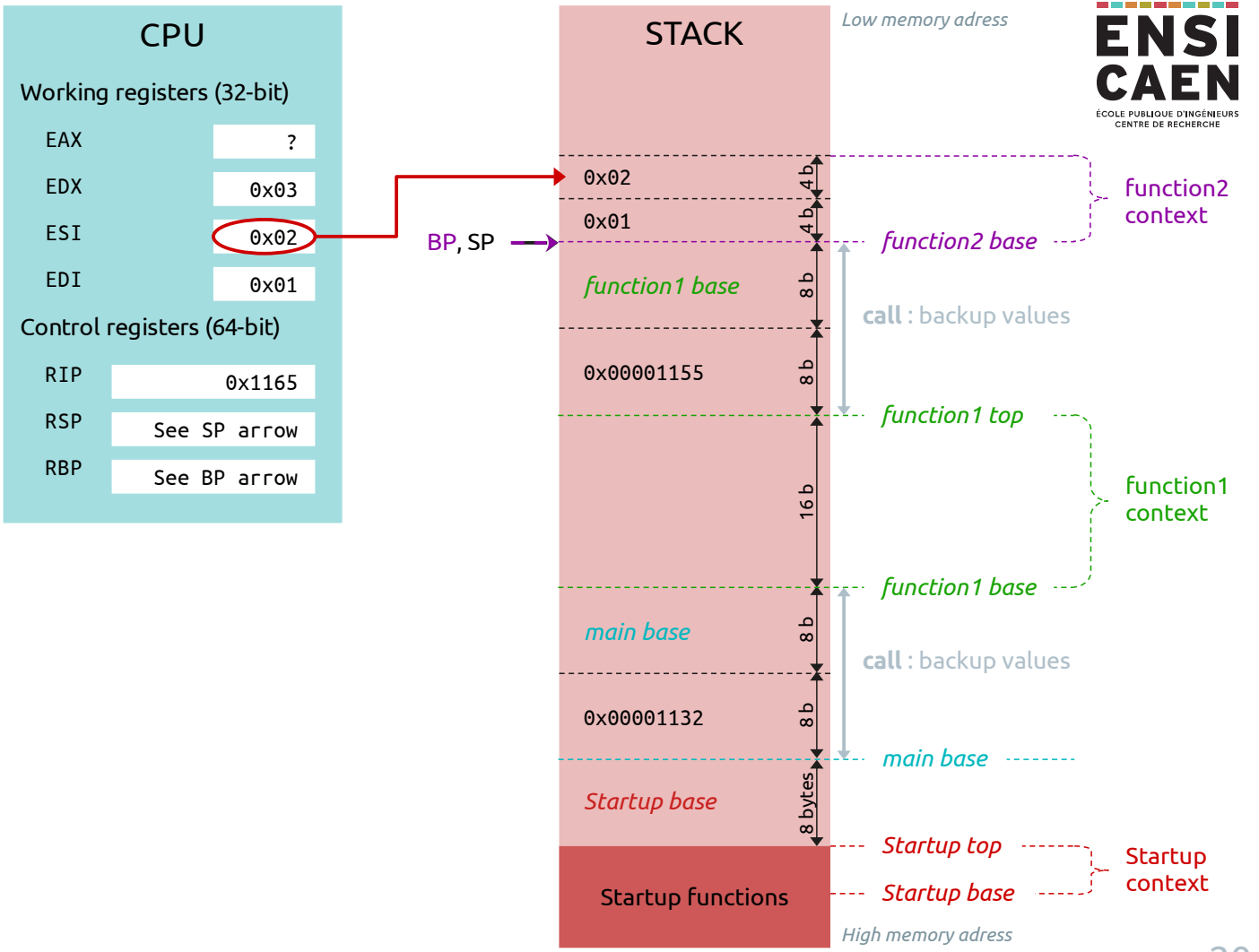
INSTRUCTION EXPLAINED

```
    movl %edi, %rbp
= Place it into RBP

    movl %edi, (%rbp)
= Place it at the address
pointed by RBP

    movl %edi, -0x4(%rbp)
= Place it 4 bytes before
the address pointed by RBP
```



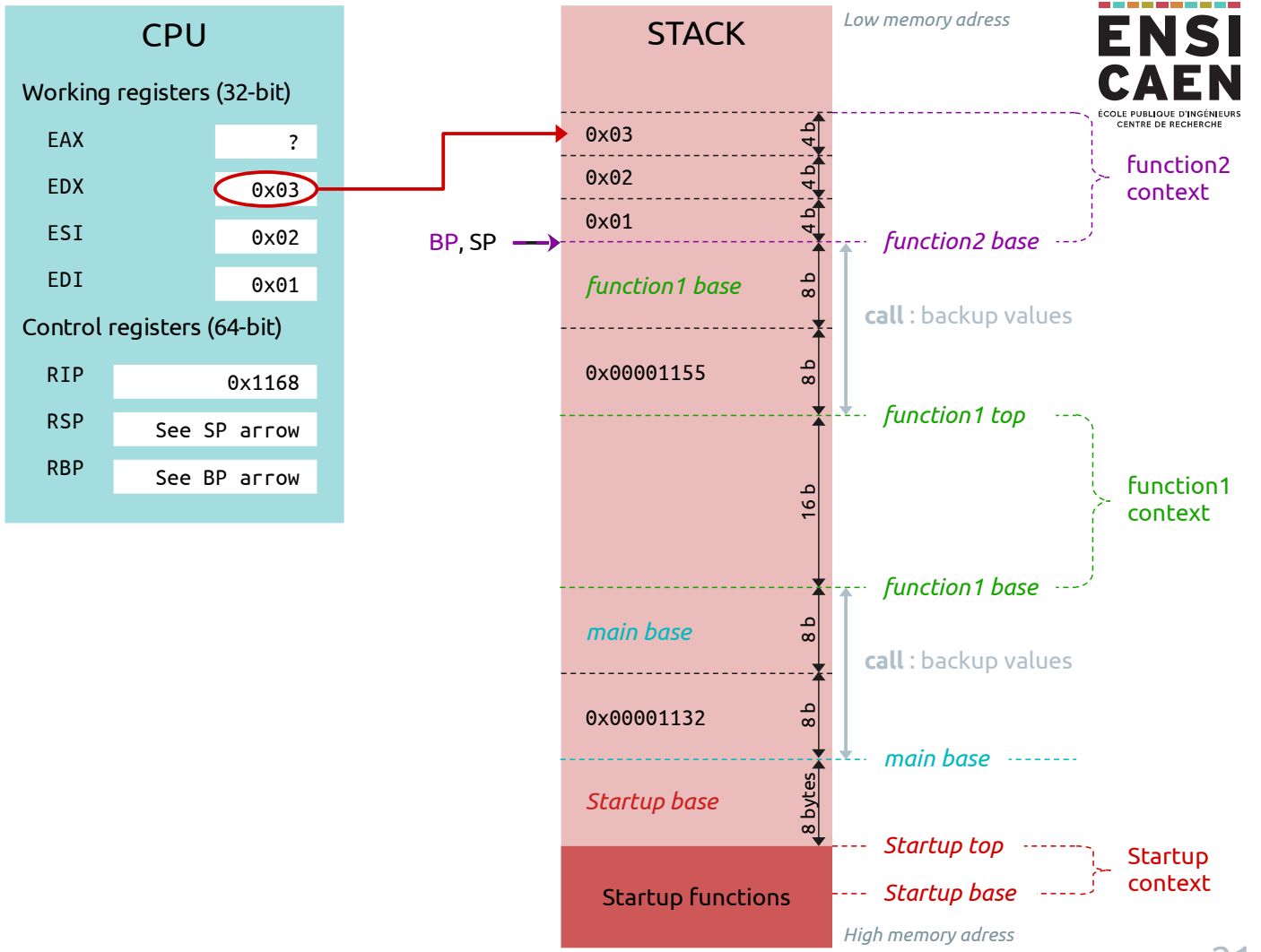
Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```



Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```



Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```

CPU

Working registers (32-bit)

EAX	?
EDX	0x01
ESI	0x02
EDI	0x01

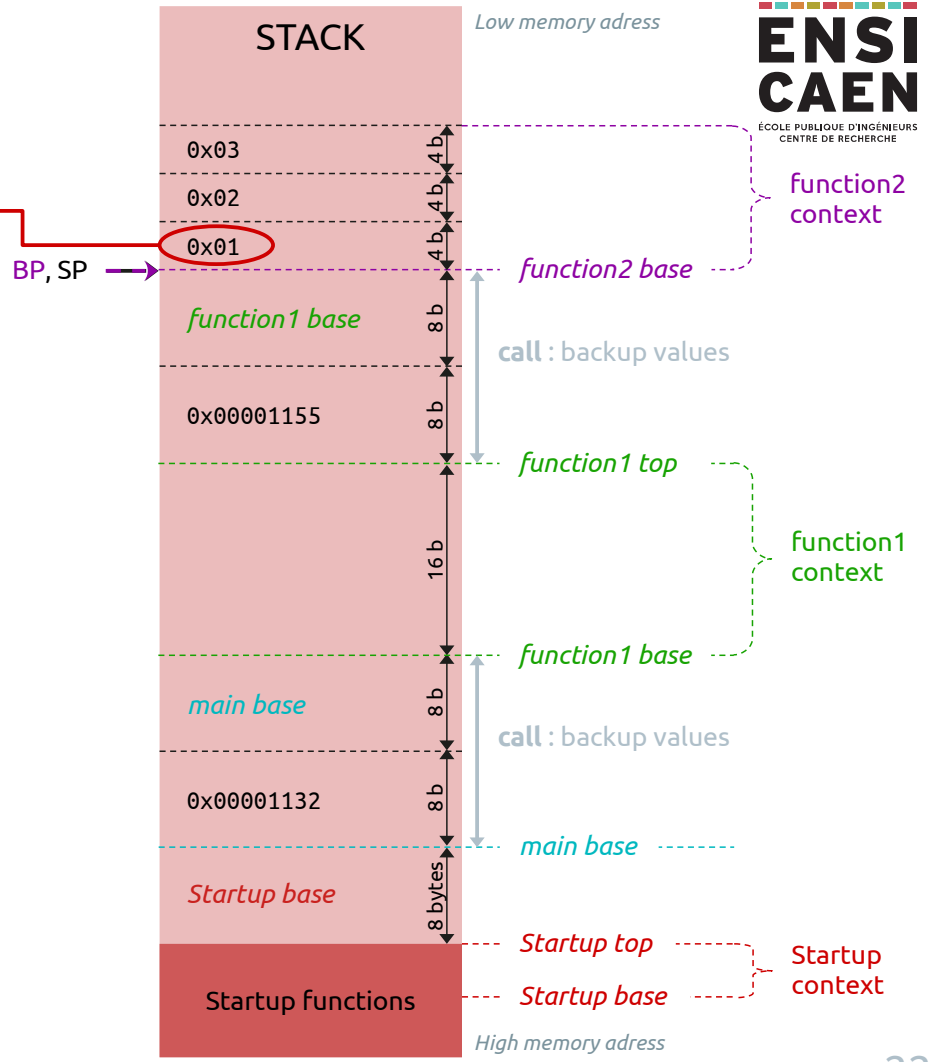
Control registers (64-bit)

RIP	0x116b
RSP	See SP arrow
RBP	See BP arrow

INSTRUCTION EXPLAINED

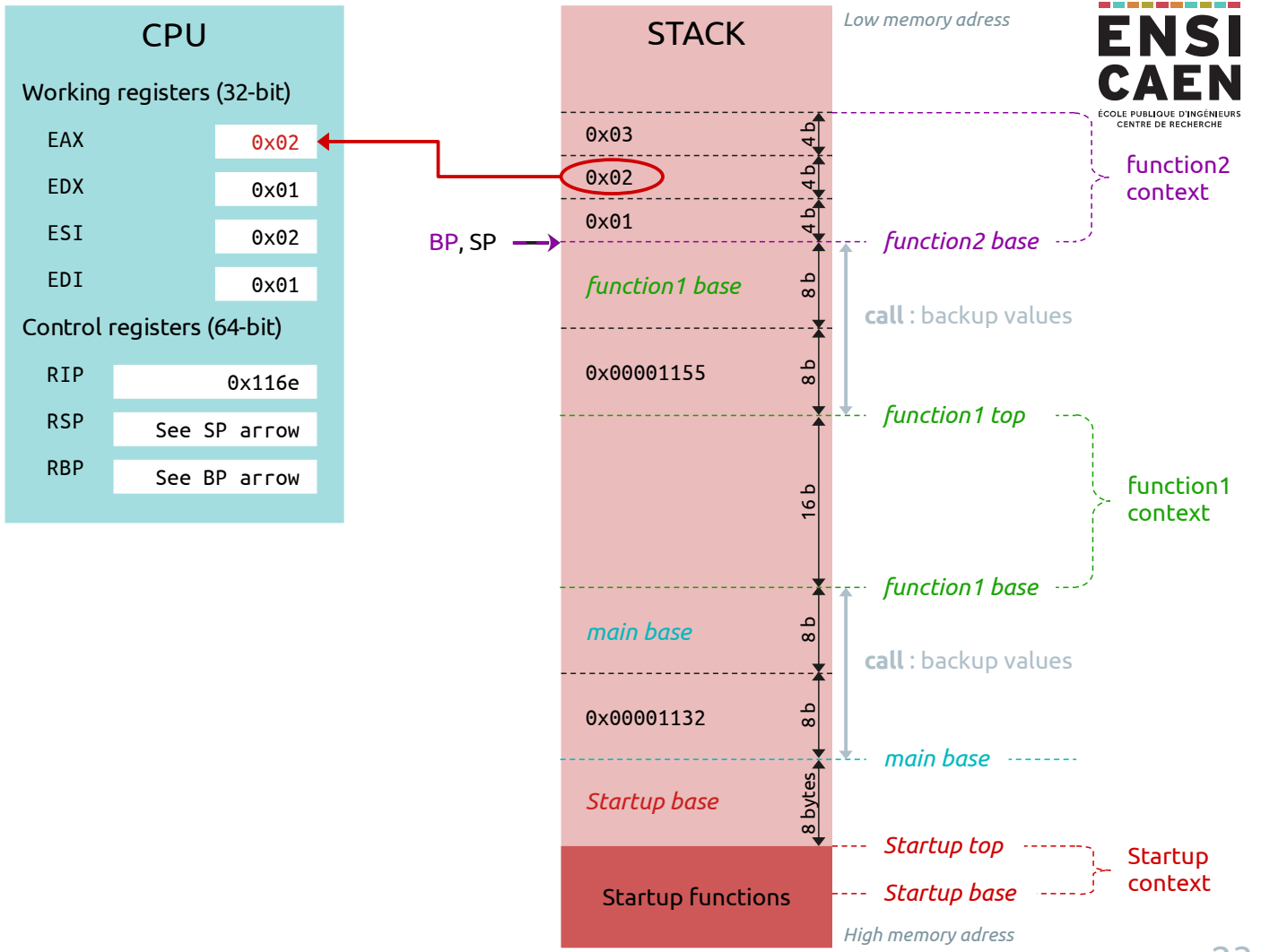
```
movl -0x4(%rbp), %edx
```

= Take the value that is 4 bytes before the address pointed by RBP, and copy it into EDX



Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```



Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```

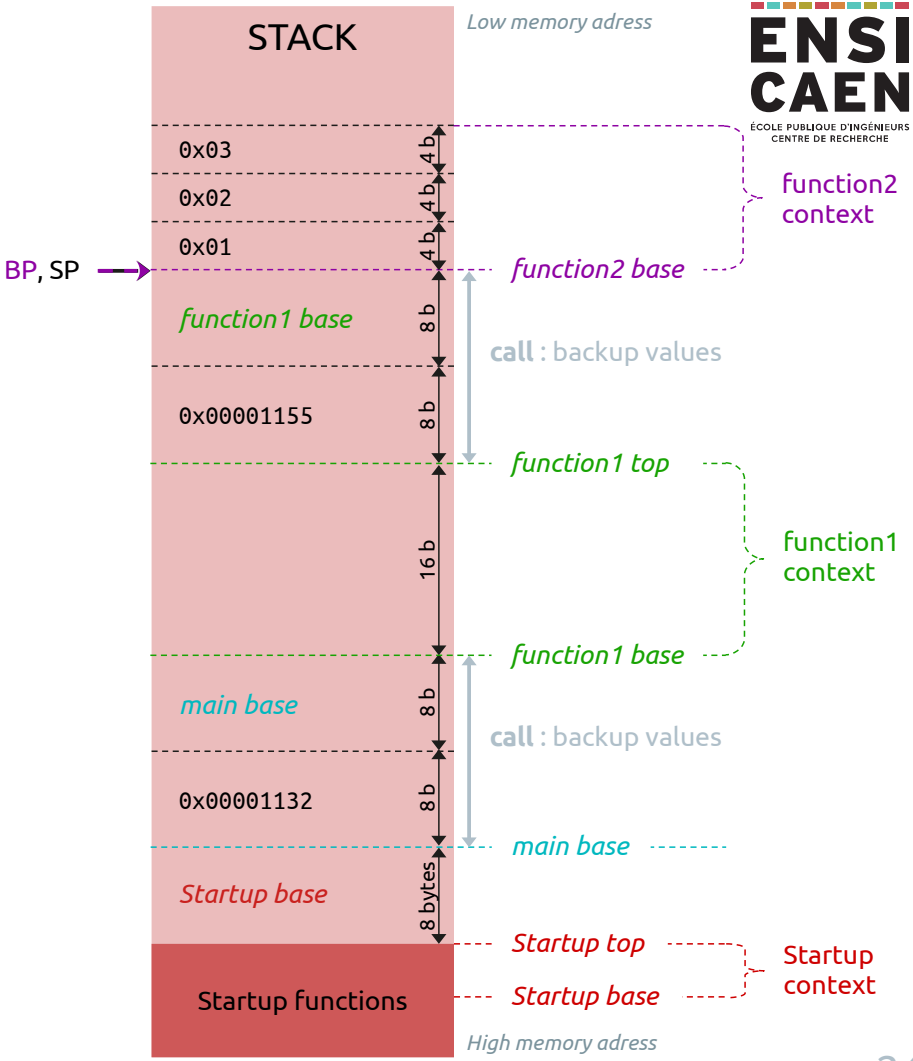
CPU

Working registers (32-bit)

EAX	0x02
EDX	0x03
ESI	0x02
EDI	0x01

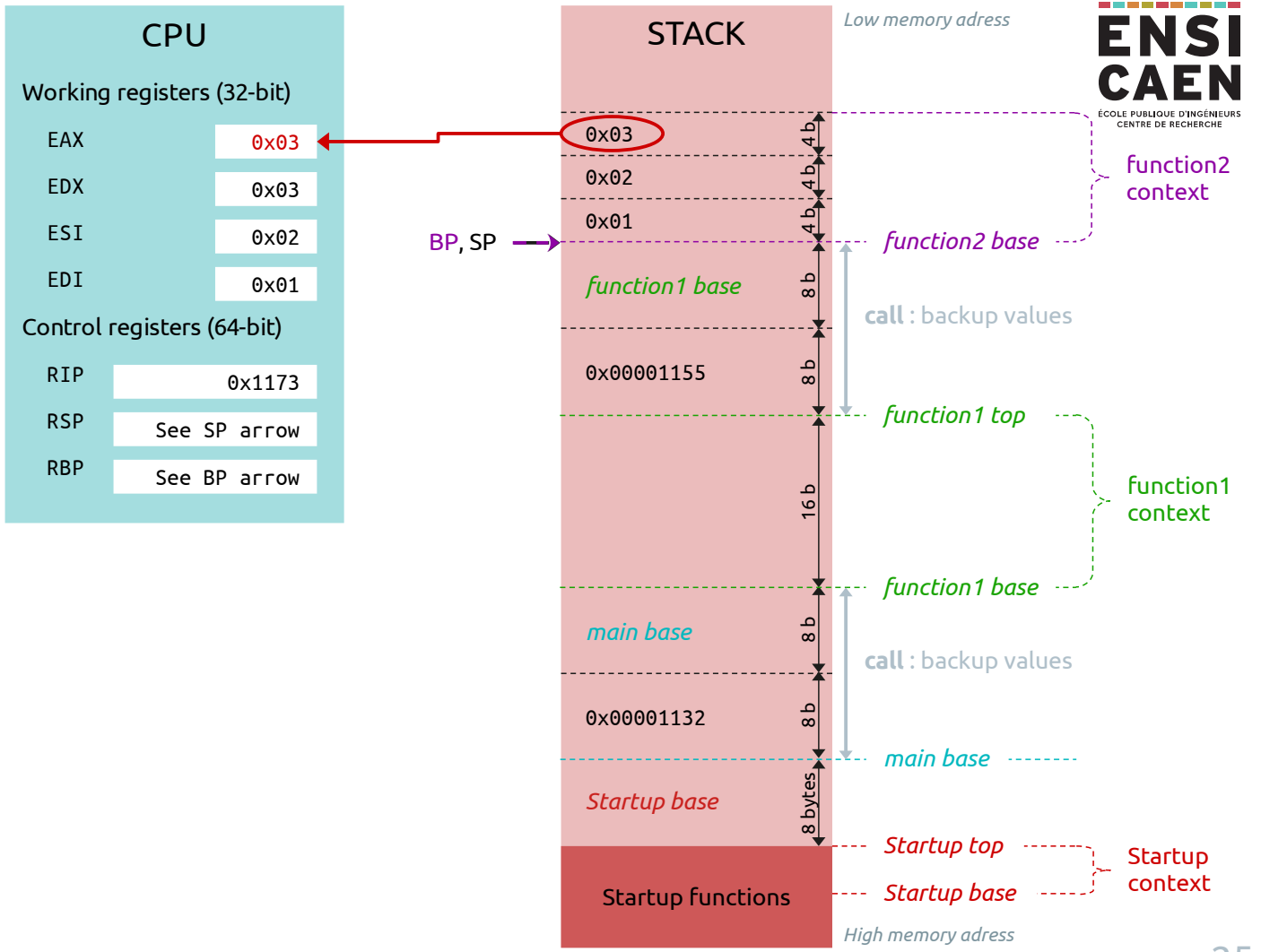
Control registers (64-bit)

RIP	0x1170
RSP	See SP arrow
RBP	See BP arrow



Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```



Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```

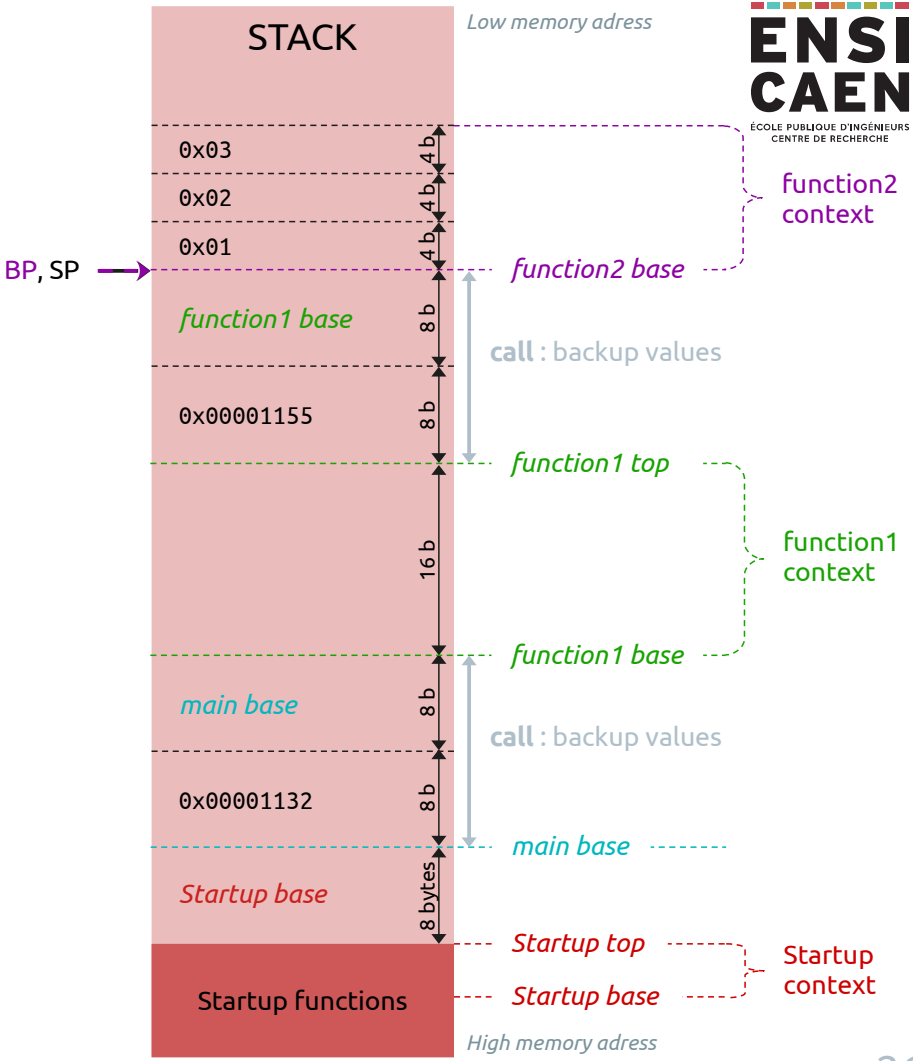
CPU

Working registers (32-bit)

EAX	0x06
EDX	0x03
ESI	0x02
EDI	0x01

Control registers (64-bit)

RIP	0x1175
RSP	See SP arrow
RBP	See BP arrow



Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```

CPU

Working registers (32-bit)

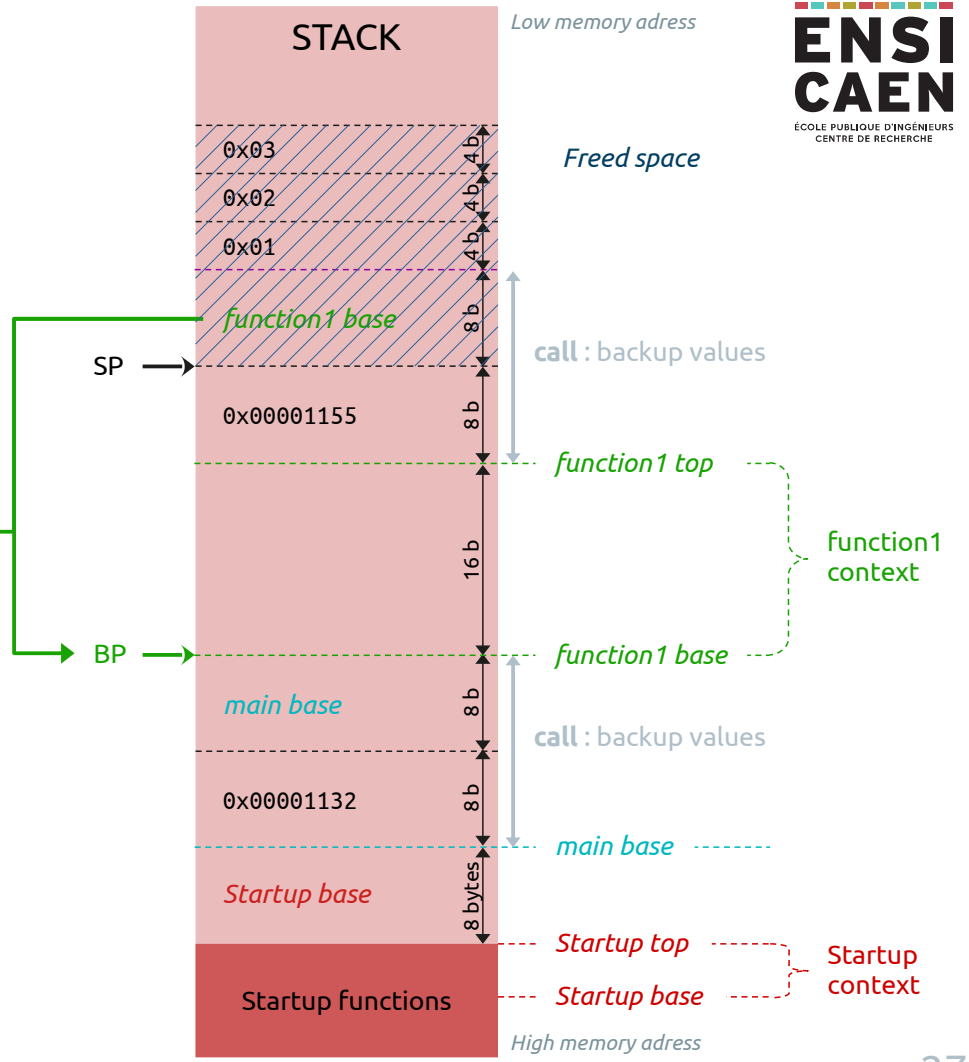
EAX	0x06
EDX	0x03
ESI	0x02
EDI	0x01

Control registers (64-bit)

RIP	0x1176
RSP	See SP arrow
RBP	See BP arrow

INSTRUCTION EXPLAINED

```
popq %rbp

is equivalent to:

movq (%rsp), %rbp
addq $8, %rsp
```



Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```

CPU

Working registers (32-bit)

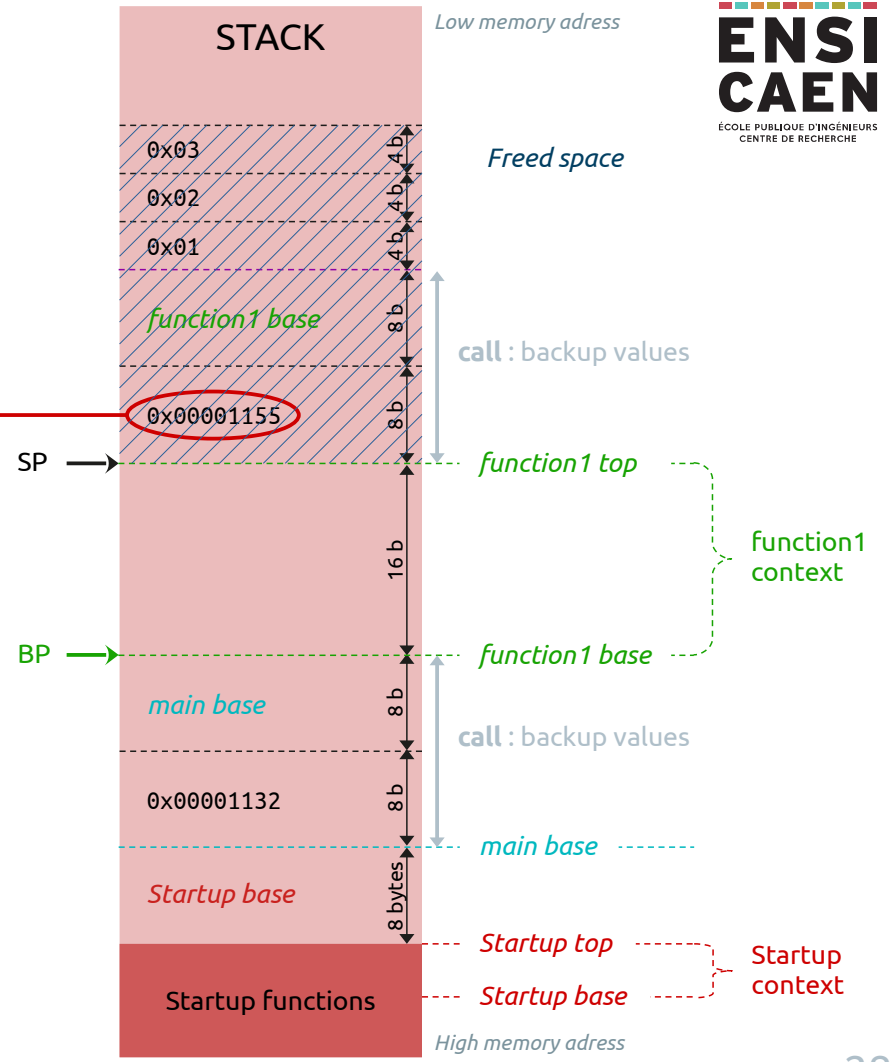
EAX	0x06
EDX	0x03
ESI	0x02
EDI	0x01

Control registers (64-bit)

RIP	0x1155
RSP	See SP arrow
RBP	See BP arrow

INSTRUCTION EXPLAINED

```
retq

is equivalent to:

popq %rip
```



Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```

CPU

Working registers (32-bit)

EAX	0x06
EDX	0x03
ESI	0x02
EDI	0x01

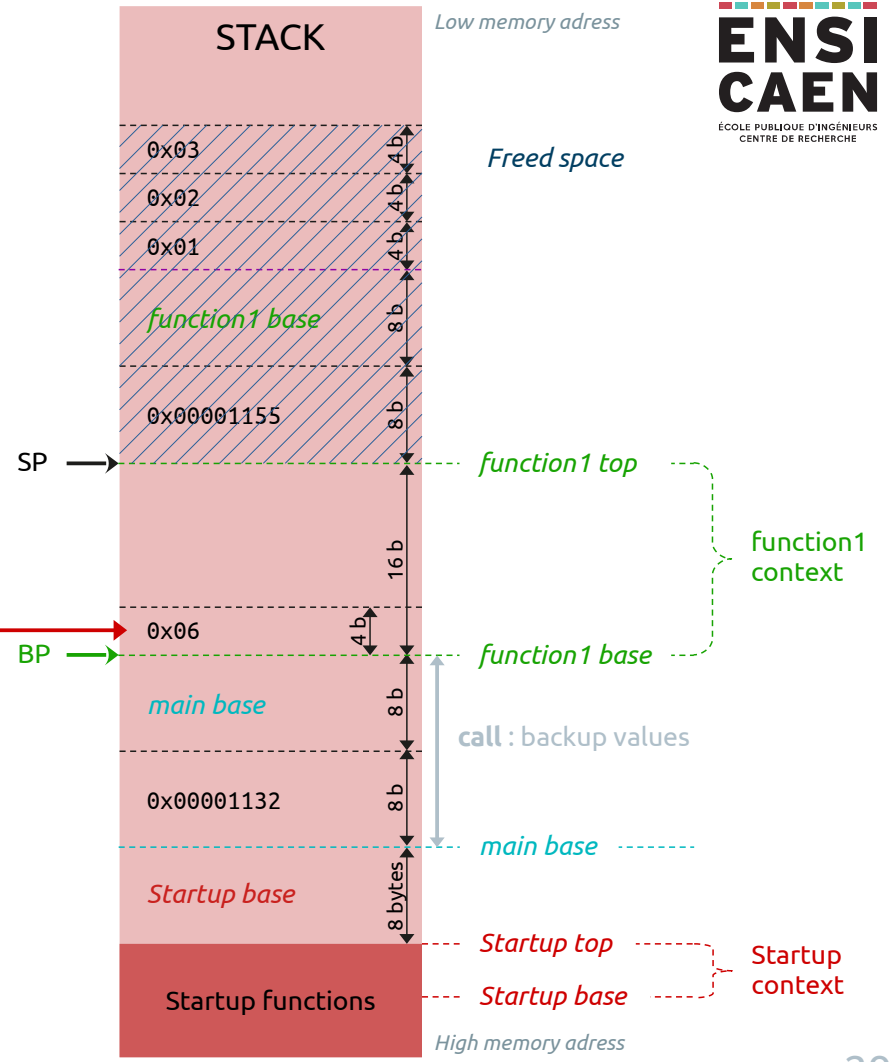
Control registers (64-bit)

RIP	0x1158
RSP	See SP arrow
RBP	See BP arrow

INSTRUCTION EXPLAINED

```
movl %eax, -0x4(%rbp)
```

When placed just after a *call* instruction, this *mov* instruction retrieves the « return » value from the EAX register and places it into the function context



Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```

CPU

Working registers (32-bit)

EAX	0x06
EDX	0x03
ESI	0x02
EDI	0x01

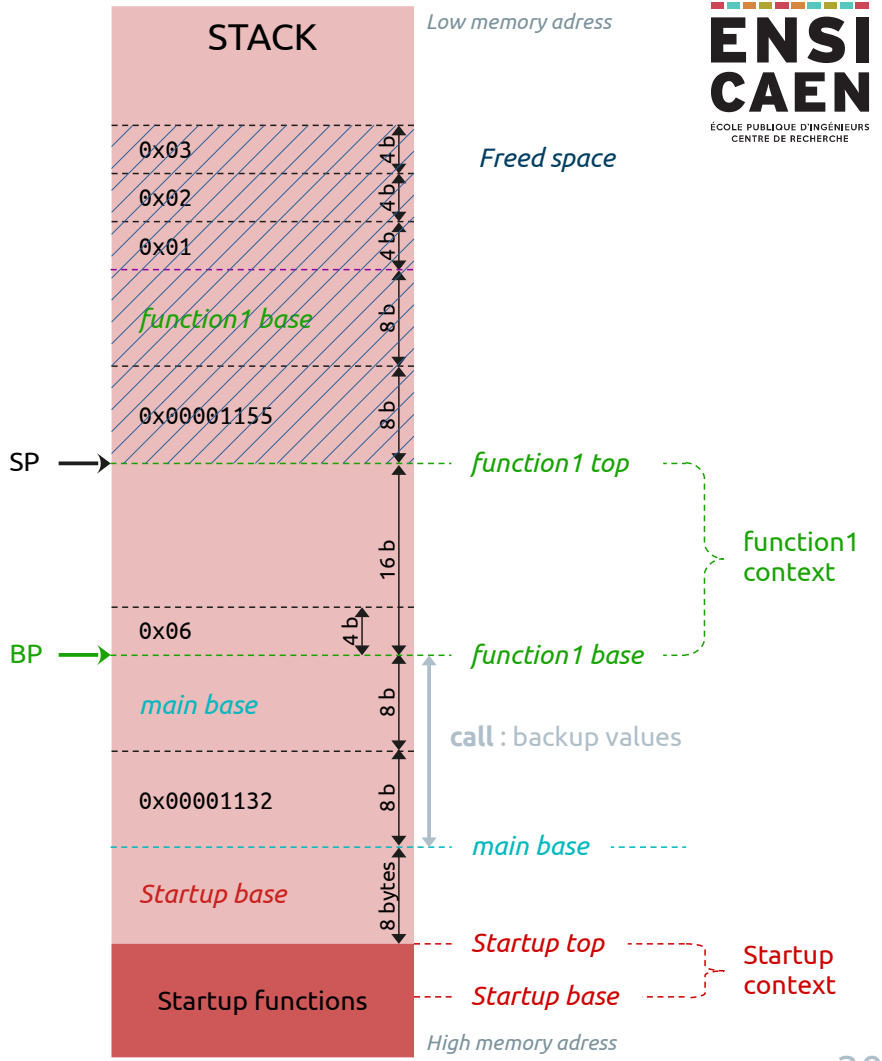
Control registers (64-bit)

RIP	0x1159
RSP	See SP arrow
RBP	See BP arrow

INSTRUCTION EXPLAINED

```
nop
```

No operation,
does nothing



Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```

CPU

Working registers (32-bit)

EAX	0x06
EDX	0x03
ESI	0x02
EDI	0x01

Control registers (64-bit)

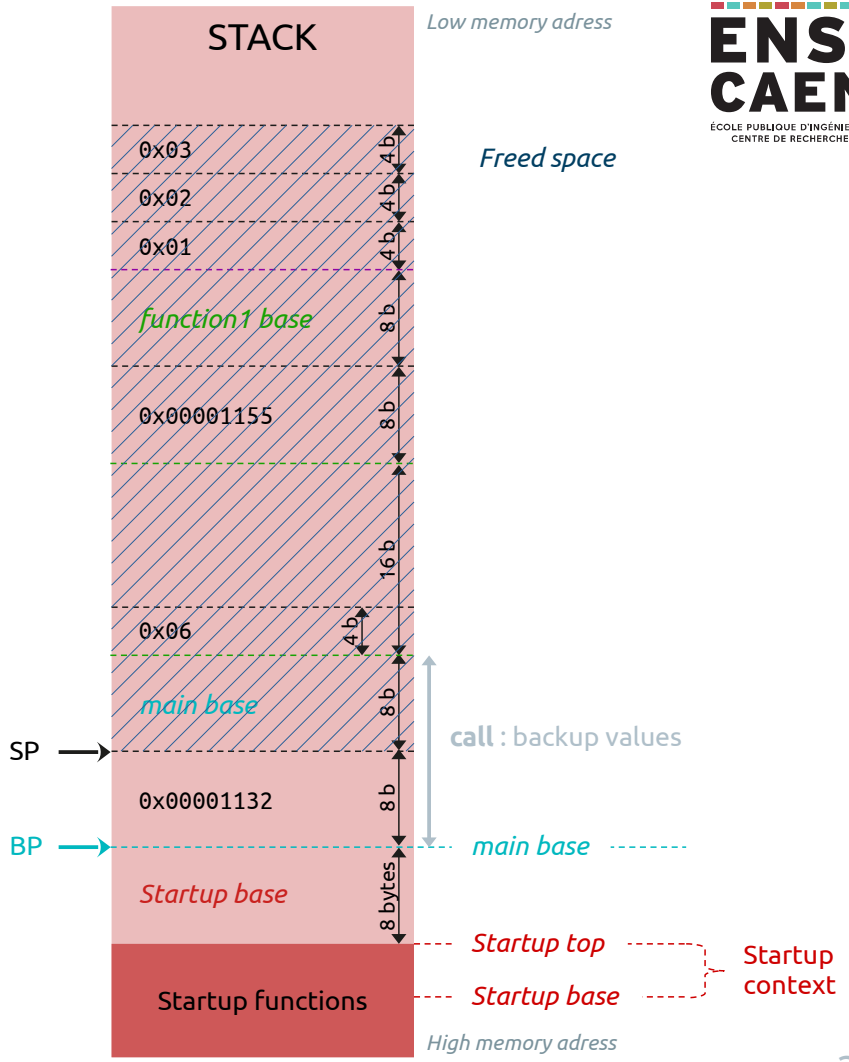
RIP	0x115a
RSP	See SP arrow
RBP	See BP arrow

INSTRUCTION EXPLAINED

```
leaveq
```

Prepares to leave this function, equivalent to:

```
movq %rbp, %rsp
popq %rbp
```



Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```

CPU

Working registers (32-bit)

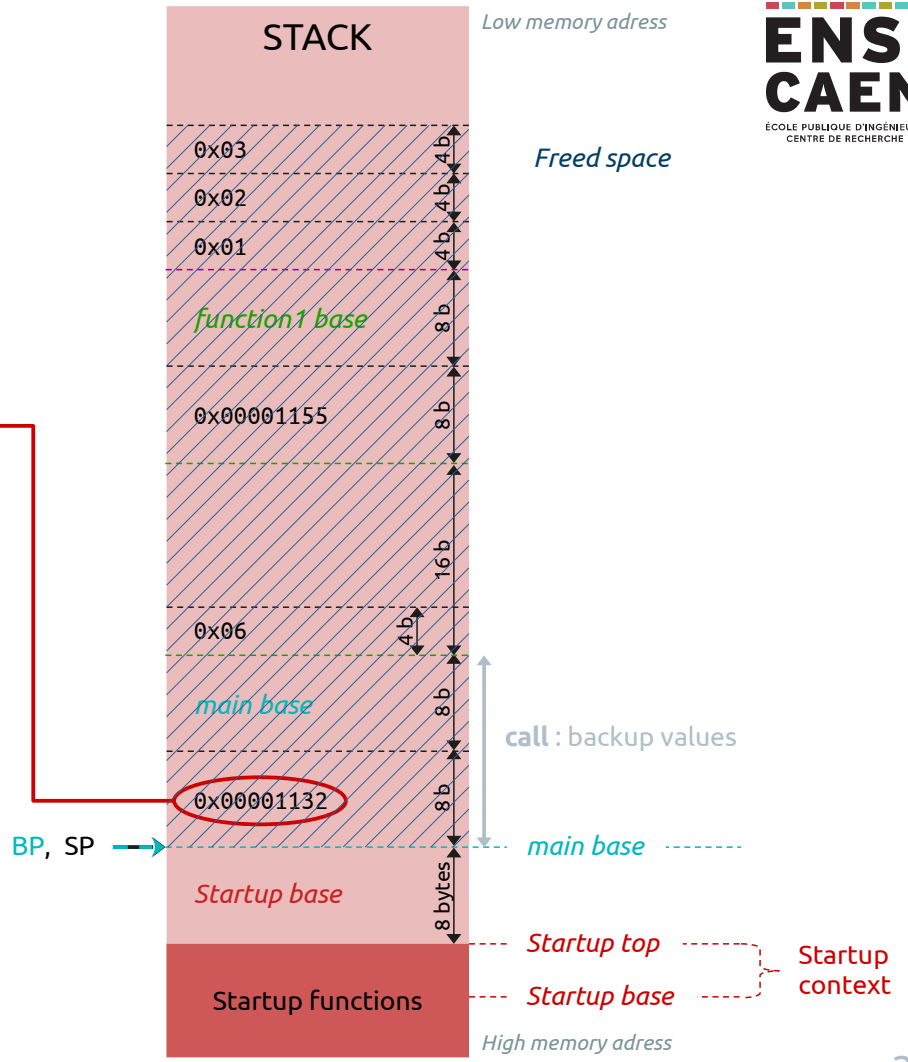
EAX	0x06
EDX	0x03
ESI	0x02
EDI	0x01

Control registers (64-bit)

RIP	0x1132
RSP	See SP arrow
RBP	See BP arrow

INSTRUCTION EXPLAINED

```
retq

is equivalent to:

popq %rip
```



Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```

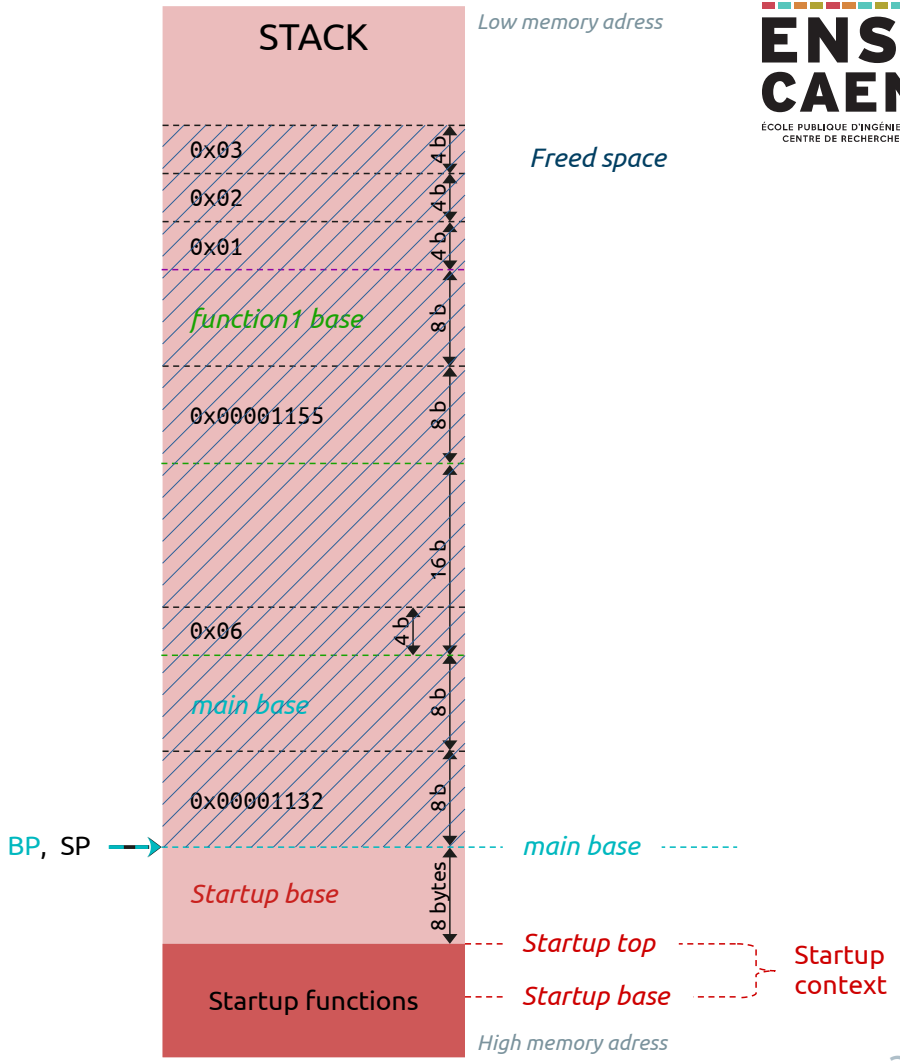
CPU

Working registers (32-bit)

EAX	0x00
EDX	0x03
ESI	0x02
EDI	0x01

Control registers (64-bit)

RIP	0x1137
RSP	See SP arrow
RBP	See BP arrow



Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```

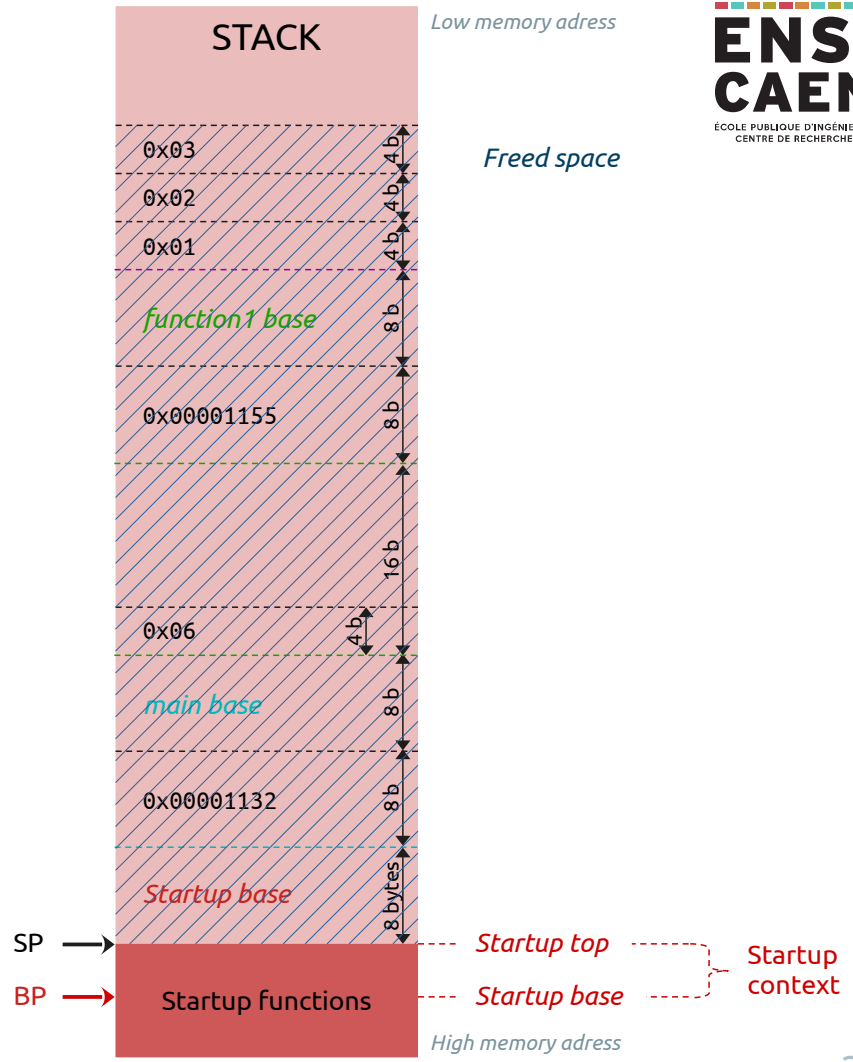
CPU

Working registers (32-bit)

EAX	0x00
EDX	0x03
ESI	0x02
EDI	0x01

Control registers (64-bit)

RIP	0x1138
RSP	See SP arrow
RBP	See BP arrow



Disassembly

```
0000000000001129 <main>:
    1129:     pushq  %rbp
    112a:     movq   %rsp, %rbp
    112d:     callq  1139 <function_1>
    1132:     movl   $0x0, %eax
    1137:     popq   %rbp
    1138:     retq

0000000000001139 <function_1>:
    1139:     pushq  %rbp
    113a:     movq   %rsp,  %rbp
    113d:     subq   $0x10, %rsp
    1141:     movl   $0x3,  %edx
    1146:     movl   $0x2,  %esi
    114b:     movl   $0x1,  %edi
    1150:     callq  115b <function_2>
    1155:     movl   %eax,  -0x4(%rbp)
    1158:     nop
    1159:     leaveq
    115a:     retq

000000000000115b <function_2>:
    115b:     pushq  %rbp
    115c:     movq   %rsp,  %rbp
    115f:     movl   %edi,  -0x4(%rbp)
    1162:     movl   %esi,  -0x8(%rbp)
    1165:     movl   %edx,  -0xc(%rbp)
    1168:     movl   -0x4(%rbp), %edx
    116b:     movl   -0x8(%rbp), %eax
    116e:     addl   %eax,  %edx
    1170:     movl   -0xc(%rbp), %eax
    1173:     addl   %edx,  %eax
    1175:     popq   %rbp
    1176:     retq
```

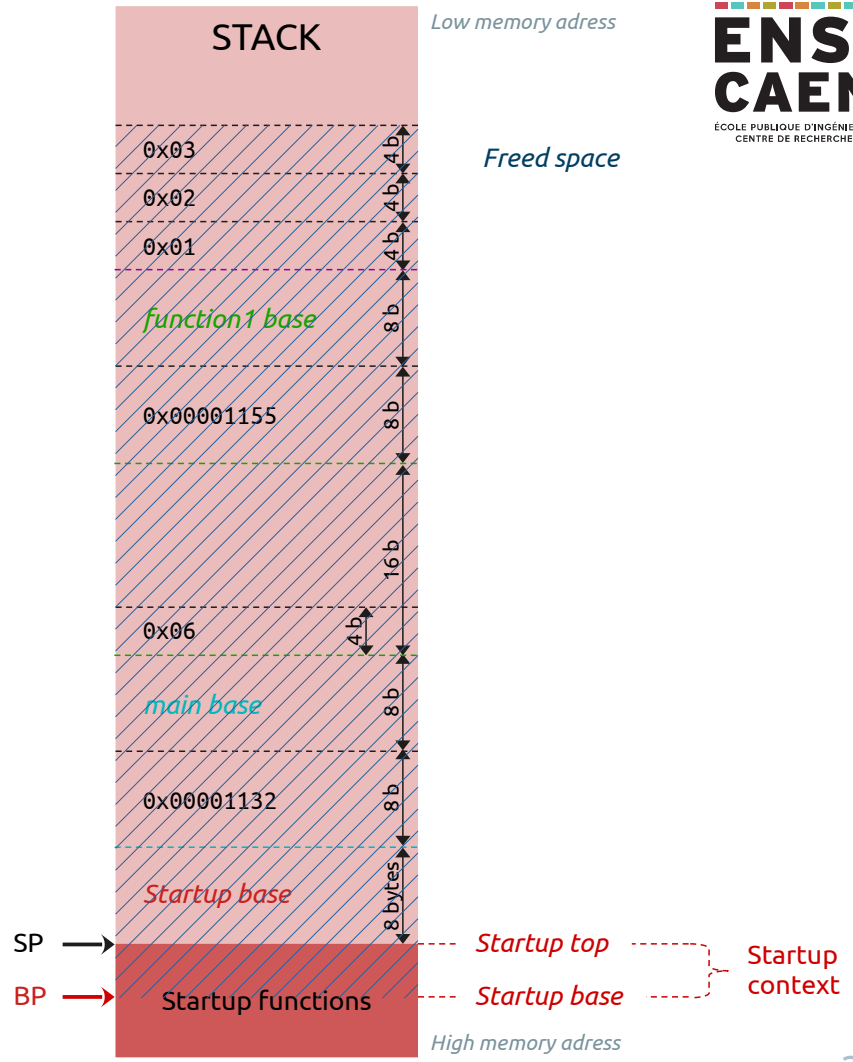
CPU

Working registers (32-bit)

EAX	0x00
EDX	0x03
ESI	0x02
EDI	0x01

Control registers (64-bit)

RIP	0x????
RSP	See SP arrow
RBP	See BP arrow



Informations à retenir

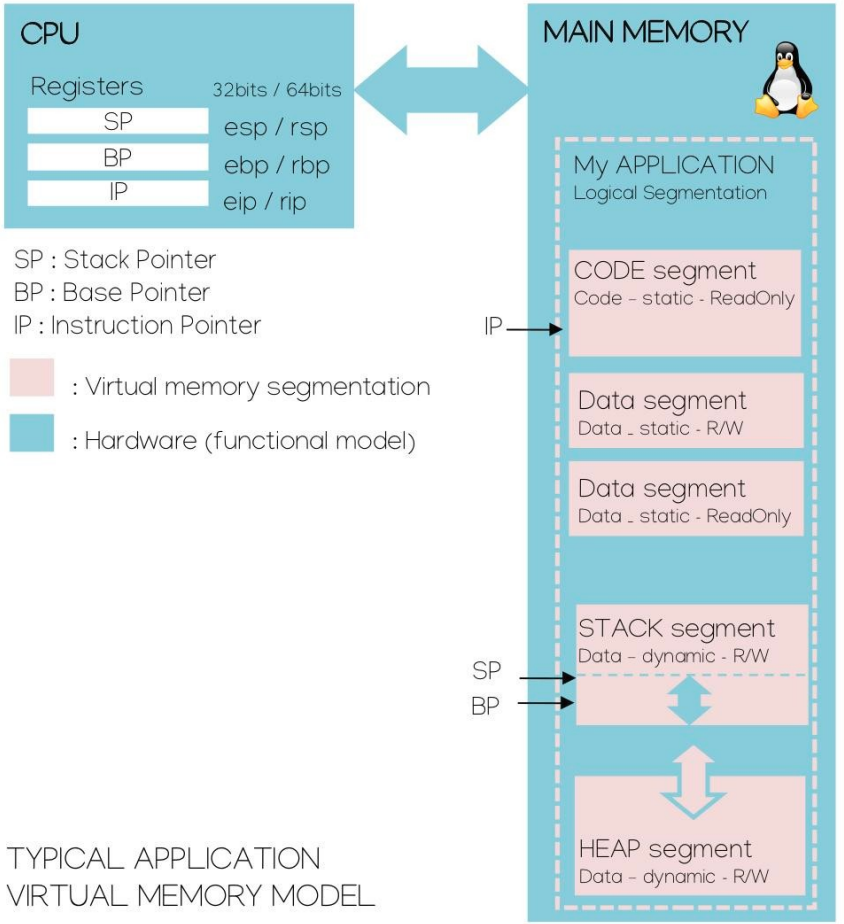
Lors de l'exécution d'un programme, la mémoire dont il dispose est répartie en plusieurs segments.

Les instructions et quelques autres données sont contenues dans l'exécutable (ce qu'on appelle allocation statique).

Les variables locales (et quelques autres informations utiles) sont gérées via la pile/*stack*. C'est le processus d'allocation automatique.

L'allocation dynamique de grands espaces mémoires est traité par le tas/*heap*.

Les allocations statique et dynamique seront abordées dans les prochains sujets de TP.



Le ***Base Pointer*** **BP** (ou *Frame Pointer* FP) se déplace au grès des appels et retours de fonction.

Son rôle est d'indiquer le début de la zone mémoire (plus précisément le début de la zone de la pile) allouée à la fonction en cours d'exécution.

Quand on entre dans une fonction :

- la première instruction (`push %rbp`) consiste à sauvegarde le BP de la fonction appelante

- la seconde instruction (`mov %rsp, %rbp`) consiste à définir une nouvelle valeur au BP (au sommet de la pile) pour indiquer le début de la zone de la *stack* associée à cette nouvelle fonction.

Quand on quitte une fonction :

- l'avant-dernière instruction (`pop %rbp`) permet de redonner à BP la valeur indiquant le début de la zone de la *stack* associée à la fonction appelante. Cette instruction est parfois précédée d'un `leave` (notamment pour faire redescendre SP et ainsi annuler l'allocation mémoire de la fonction qu'on s'apprête à quitter, voir point suivant).

Le ***Stack Pointer* SP** évolue très fréquemment.

Son rôle est d'indiquer où se trouve la dernière donnée placée dans la pile (et donc le sommet de la pile).

Dès lors qu'une donnée est poussée dans la pile (instruction `push`), la valeur de SP change (d'après notre représentation, le pointeur monte).

Quand SP redescend (avec un `pop` ou un `leave` par exemple), les données au dessus du sommet de pile ne sont pas effacées, mais simplement oubliées (elles seront certainement écrasées à l'avenir).

Si l'instruction `sub` est appliquée sur SP, il y a de fortes chances que l'espace mémoire désormais compris entre SP et BP soit dédié au stockage des variables locales à la fonction courante.

Les pointeurs SP et BP gèrent donc l'espace mémoire alloué pour chaque fonction, et ceci sans intervention explicite du développeur : il s'agit d'un mécanisme d'allocation automatique de la mémoire, géré par la chaîne de compilation toute seule !

Le rôle du **pointeur d'instruction IP** (ou *Program Counter* PC) est de contenir l'adresse de la prochaine instruction à exécuter.

La plupart du temps, IP s'incrémente **automatiquement** pour contenir l'adresse de l'instruction suivante. C'est le fonctionnement « normal » d'un programme séquentiel dans lequel toutes les instructions se suivent.

Lors d'un appel de fonction (instruction `call`), il faut casser cette démarche séquentielle et « sauter » dans la fonction appelée. C'est pourquoi **l'instruction `call` modifie directement la valeur de IP** pour contenir l'adresse de la première instruction de la fonction appelée. En même temps, l'instruction **`call` se charge de sauvegarder la valeur originale de IP** (donc l'adresse de l'instruction après le `call`) dans la pile. Mais pourquoi ?

Et bien lorsque la fonction appelée se termine, il faut pouvoir revenir dans la fonction appelante pour exécuter l'instruction d'après le `call` et reprendre le flot séquentiel d'instructions. Pour cela, il faut avoir préalablement mémorisé l'adresse de cette instruction. C'est pour cette raison que, lors du `call`, la valeur de IP (contenant l'adresse de l'instruction qui aurait normalement due être exécutée après) est stockée dans la pile. Au moment de quitter la fonction appelée, c'est **l'instruction `ret` qui se charge de récupérer la valeur mémorisée dans la pile pour la restituer dans le pointeur d'instruction IP**.

Au final, en observant la pile, on remarque une alternance zones bien définies :

- les contextes de fonction (zone mémoire dans laquelle sont stockées les variables locales d'une fonction)
- et les zones de sauvegarde de registres (IP et BP).

Les contextes permettent à une fonction de stocker ses variables locales. Dès lors que la fonction est terminée (instruction `return`), les données stockées dans son contexte sont perdues.

Les zones de sauvegardes de registres permettent de mémoriser puis récupérer les valeurs utiles pour que la fonction appelante puisse reprendre son cours après la fin de la fonction appelée (en récupérant l'adresse de l'instruction à exécuter ainsi que le début de sa zone mémoire dans la pile).

Il reste un mystère à élucider : si les variables locales de chaque fonction sont rangées dans leur contexte respectif, **où se situent les données échangées entre deux fonctions ?**

Lorsqu'une fonction en appelle une autre, la première **transmet les arguments par copie dans des registres** définis par la chaîne de compilation (voir pages 13-15 : les paramètres sont copiés dans les registres EDX, ESI, EDI). La fonction appelée n'a plus qu'à récupérer les valeurs contenues dans ces registres pour **les recopier dans son propre contexte** (pages 19-21).

Quand la fonction appelée a terminé son travail, elle place sa valeur de retour (de l'instruction C `return`) dans le registre EAX (voir pages 26 et 33).

L'échange d'informations entre fonctions se fait donc (généralement) par registres, le tout étant défini par la chaîne de compilation (GCC chez nous) et documenté officiellement.