# Reverzne inzinierstvo

Bezpecnost informacnych systemov z pohladu praxe

Peter Svec

0x01

# >shellcode vysledky

## All Time Rankings

| Rank | Hacker | Score |
|---|---|---|
| #1 | ALŠEKU | 11 |
| #2 | Power_Puffers | 11 |
| #3 | Slava_Ukrajine | 11 |
| #4 | FIIT_Dropouts | 11 |
| #5 | tichoslapos_divocakos | 11 |

# >motivacia

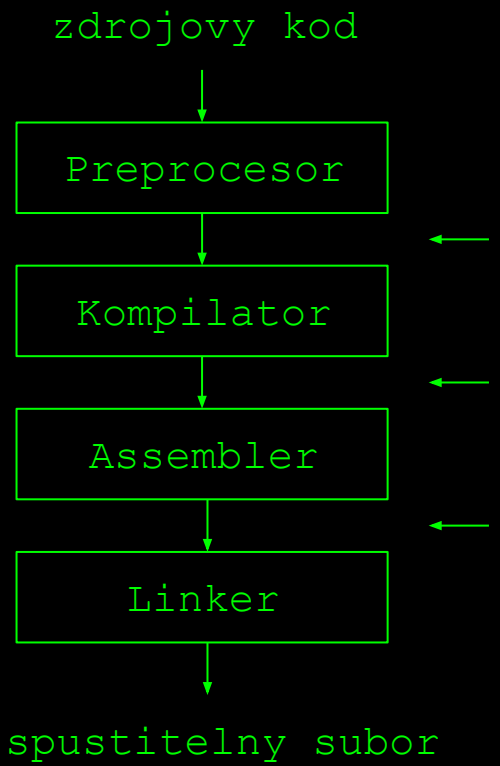
>schopnost pochopit program aj bez zdrojoveho kodu
>vyvoj exploitov
>analyza malveru
>cracking, patching

# >standardny proces

```
#include <stdio.h>
int main()
{
    printf("");
    return 0;
}
```

↓

```
00 0f ff 48 22 01 55
42 12 69 12 00 48 12
13 22 f5 a5 …
```




zdrojovy kod (makra, komentare, hlavickove subory)

zdrojovy kod (jazyk sym. instrukcii)

objektovy kod

# >zostavovaci process

>Pocas zostavovania programu stracame mnozstvo informacii:
- >nazvy premennych
- >nazvy funkcii, tried,...
- >komentare
- >struktury
- >optimalizacie prekladaca (inlining, loop unrolling,...)

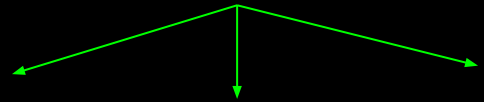
# >reverzne inzinierstvo

>opacny proces

spustitelny subor



analyticky proces



disassembler

dekompilator

ladenie



pochopenie functionality

0x06

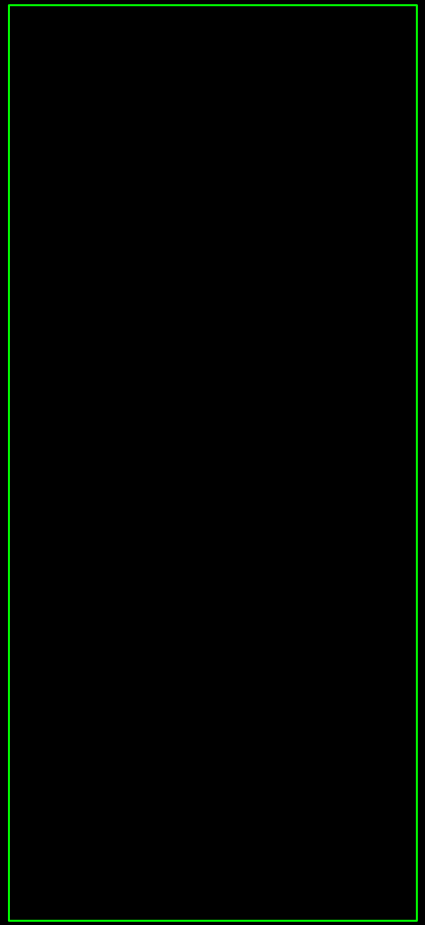
# >zasobnik

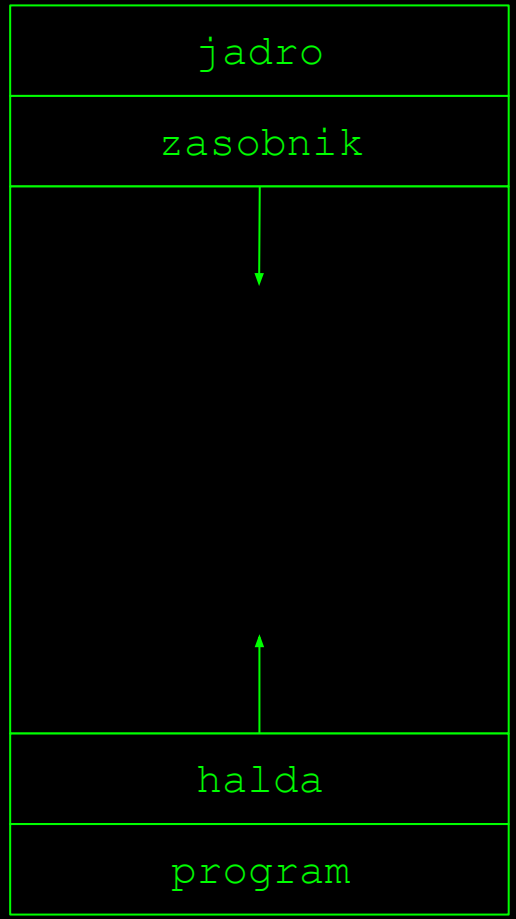
>LIFO datova struktura pouzivana pri volani funkcii (call stack)
>Na zasobnik sa ukladaju:
    >lokalne premenne
    >navratova adresa 
    >zasobnikovy ramec z predchadzajuceho volania
    >pri vysokom mnozstve argumentov aj argumenty

**ZASOBNIK RASTIE OPACNYM SMEROM**
**OD VYSOKYCH ADRIES(0xFFFF…) PO NIZSIE (0x000…)**

# >zasobnik

```
int foo()
{
    int a,b,c;
    b=1;c=2;
    a=b+c;
    return a;
}

int main()
{
    foo();
    return 0;
}
```






0xffff

0x0000

0x08

# >zasobnik

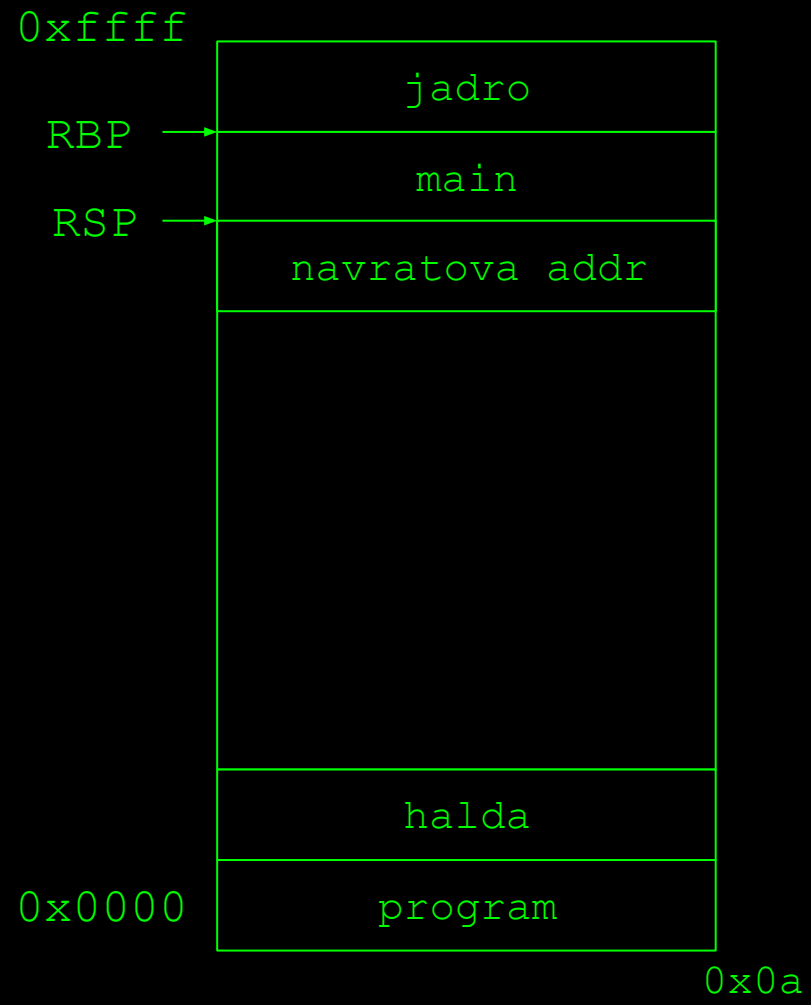
```
int foo()
{
    int a,b,c;
    b=1;c=2;
    a=b+c;
    return a;
}

int main()
{
    foo();
    return 0;
}
```

```
call foo
```

# >zasobnik

```
int foo()
{
    int a,b,c;
    b=1;c=2;
    a=b+c;
    return a;
}

int main()
{
    foo();
    return 0;
}
```

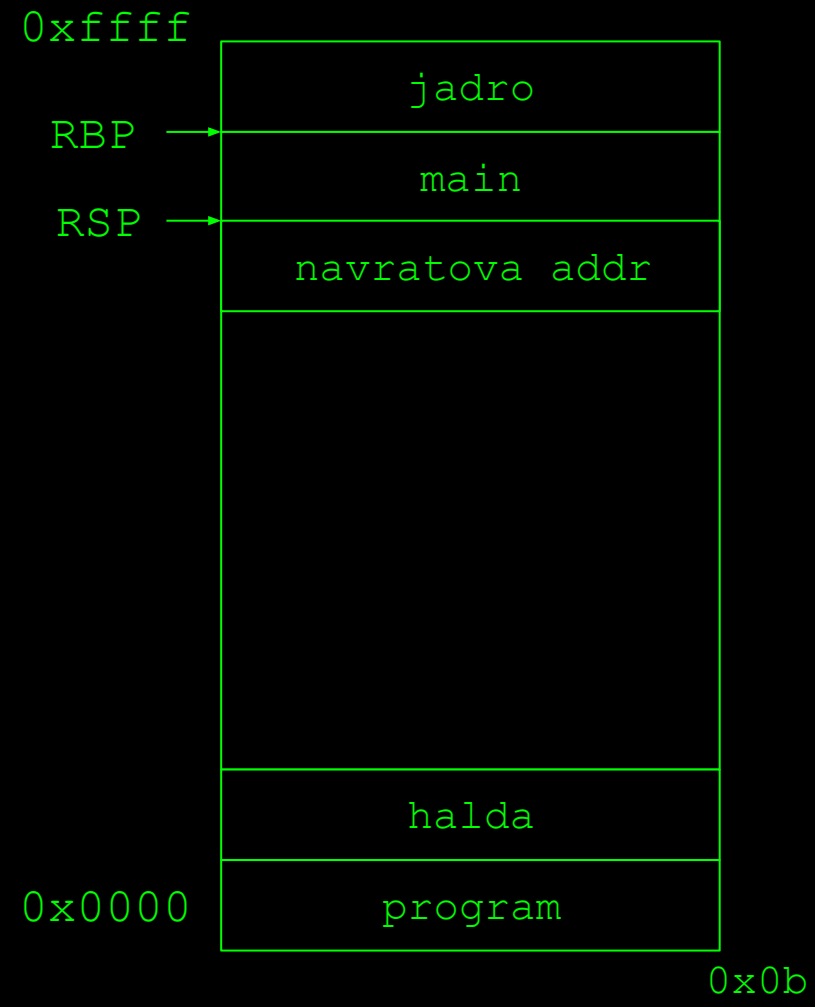
```
call foo
```



# >zasobnik

```
int foo()
{
    int a,b,c;
    b=1;c=2;
    a=b+c;
    return a;
}

int main()
{
    foo();
    return 0;
}
```

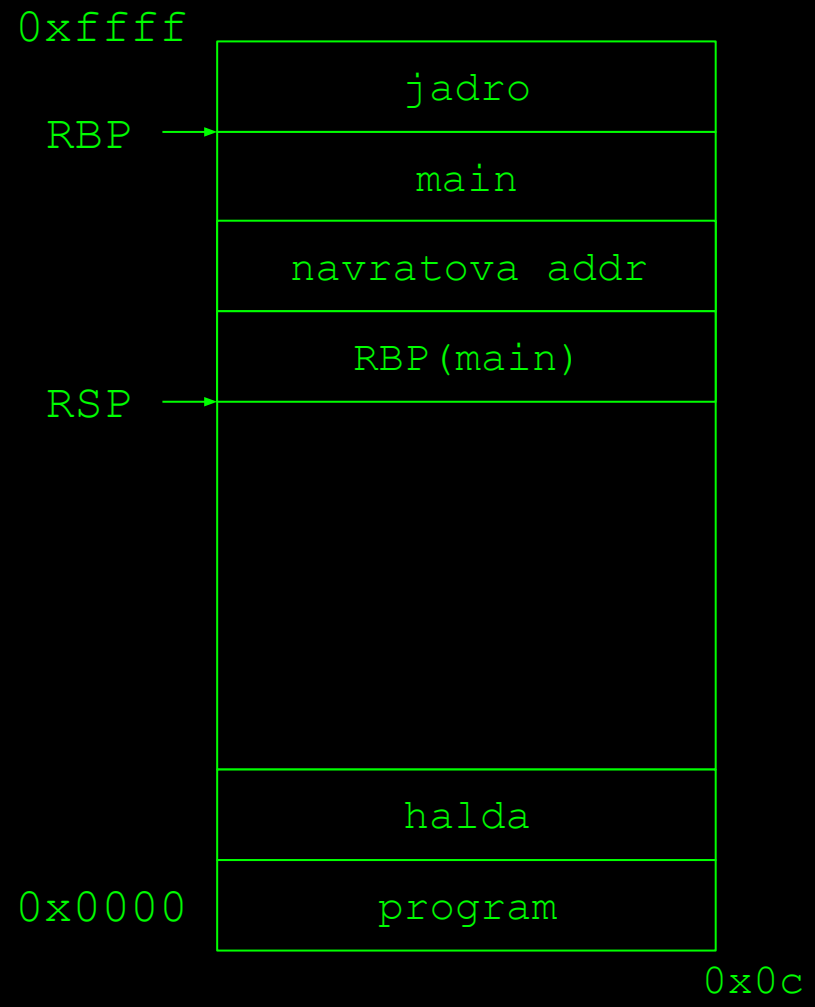
```
call foo

push rbp
```



# >zasobnik

```
int foo()
{
    int a,b,c;
    b=1;c=2;
    a=b+c;
    return a;
}

int main()
{
    foo();
    return 0;
}
```

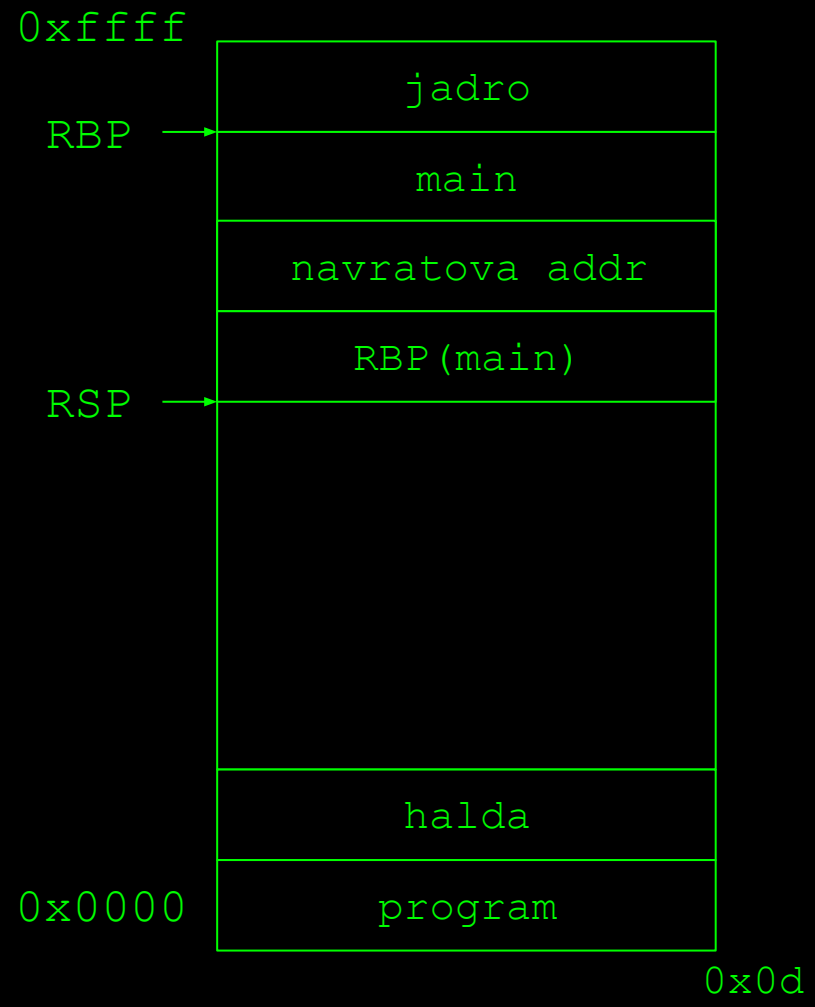
```
call foo

push rbp
```



# >zasobnik

```
int foo()
{
    int a,b,c;
    b=1;c=2;
    a=b+c;
    return a;
}

int main()
{
    foo();
    return 0;
}
```

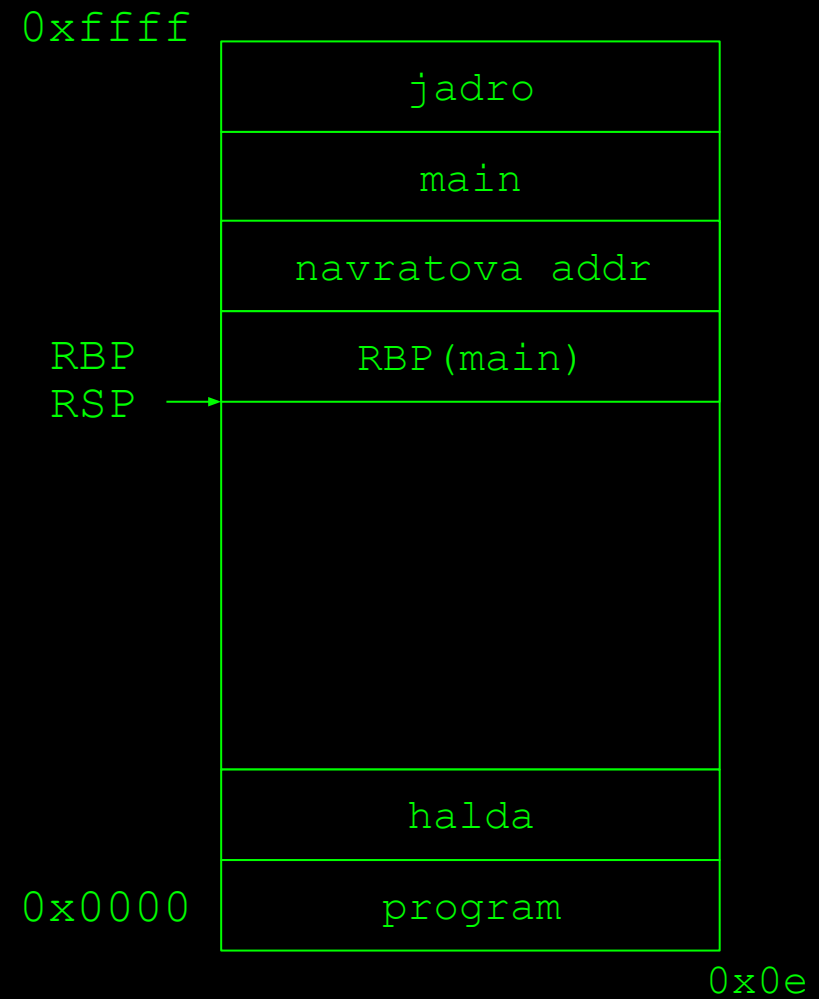
```
call foo

push rbp
mov rbp, rsp
```



# >zasobnik

```
int foo()
{
    int a,b,c;
    b=1;c=2;
    a=b+c;
    return a;
}

int main()
{
    foo();
    return 0;
}
```

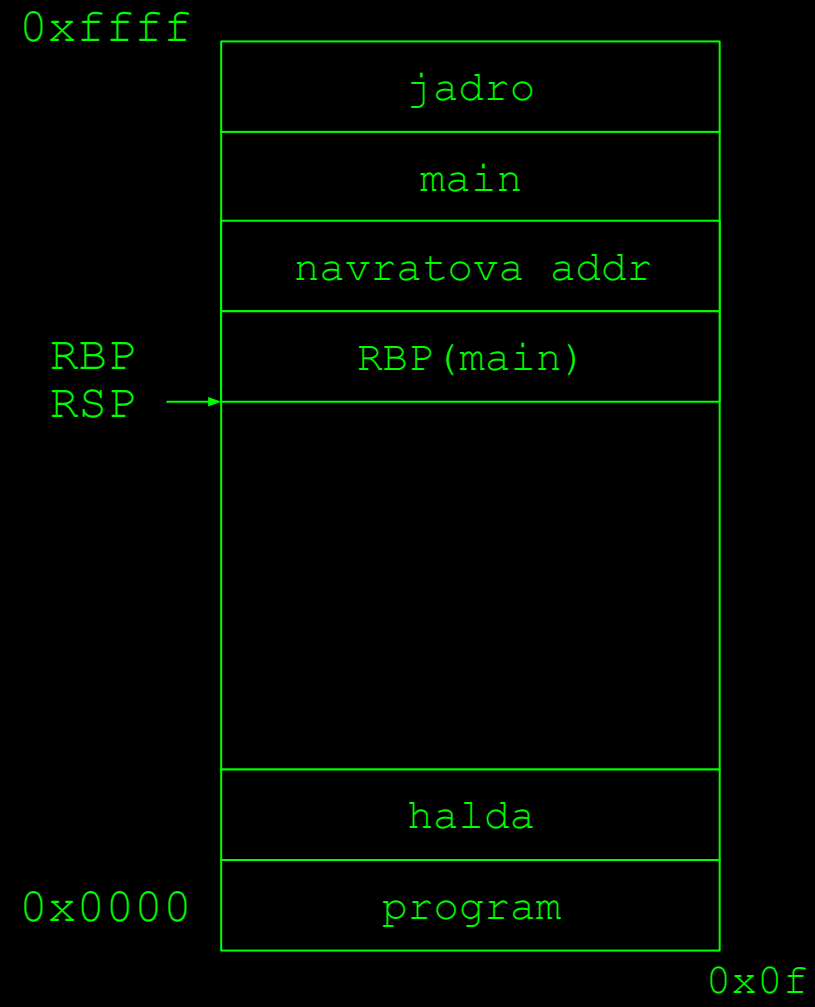
```
call foo

push rbp
mov rbp, rsp
```



# >zasobnik

```
int foo()
{
    int a,b,c;
    b=1;c=2;
    a=b+c;
    return a;
}

int main()
{
    foo();
    return 0;
}
```

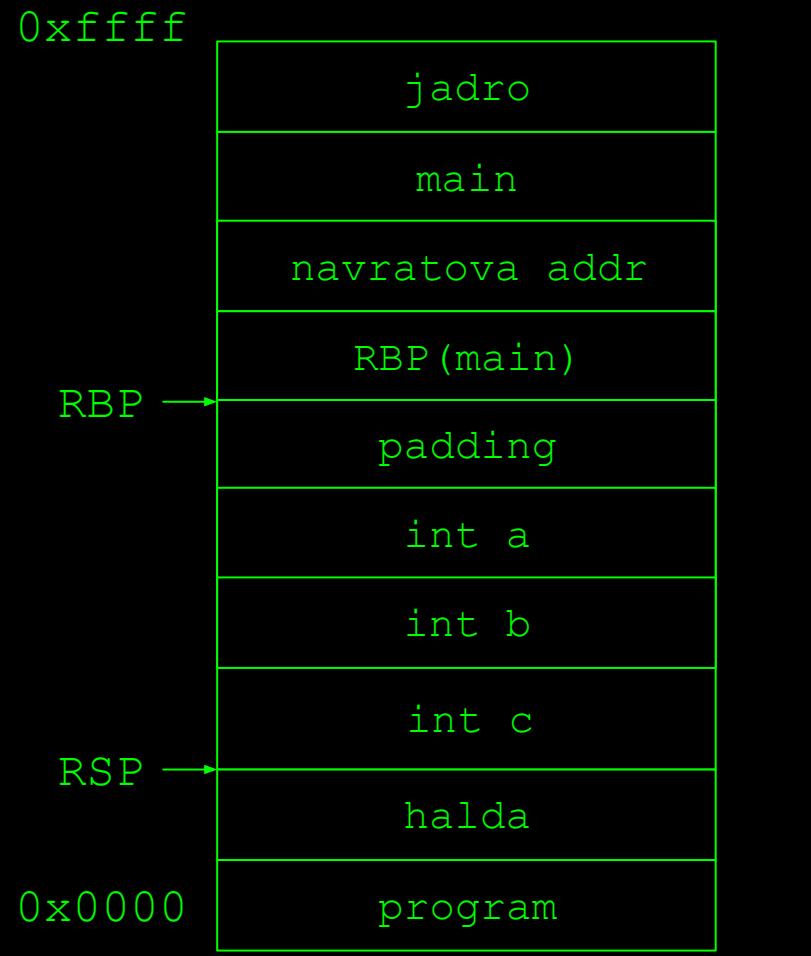
```
call foo

push rbp
mov rbp, rsp
sub rsp,0x10
```



# >zasobnik

```
int foo()
{
    int a,b,c;
    b=1;c=2;
    a=b+c;
    return a;
}

int main()
{
    foo();
    return 0;
}
```

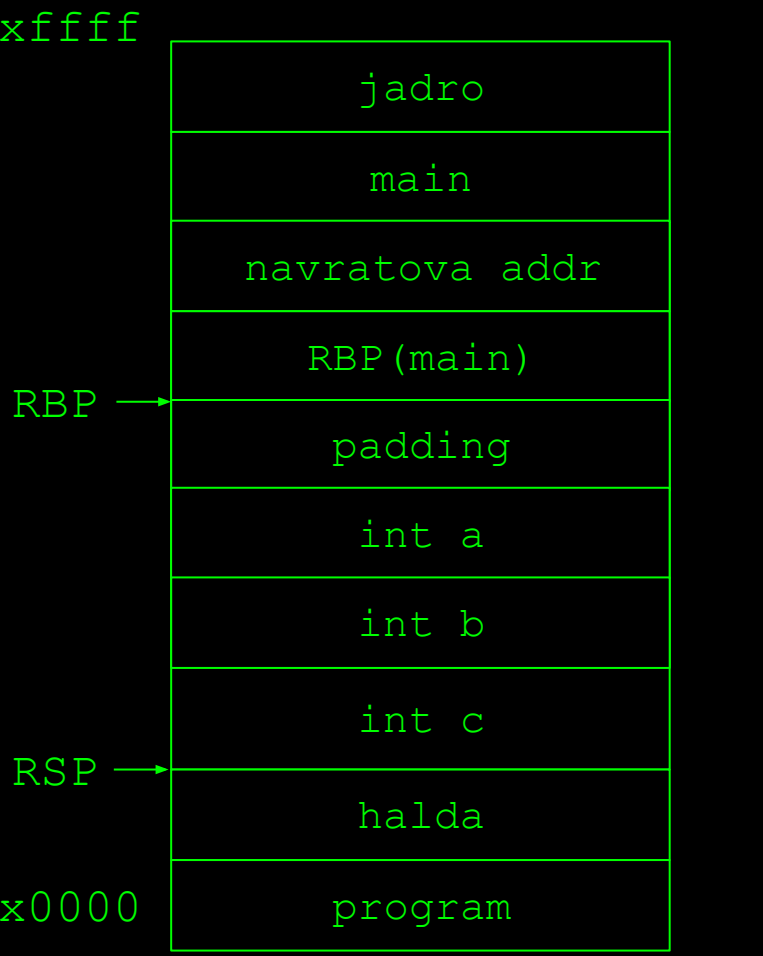
```
call foo

push rbp
mov rbp, rsp
sub rsp,0x10
```



# >zasobnik

```
int foo()
{
    int a,b,c;
    b=1;c=2;
    a=b+c;
    return a;
}

int main()
{
    foo();
    return 0;
}
```

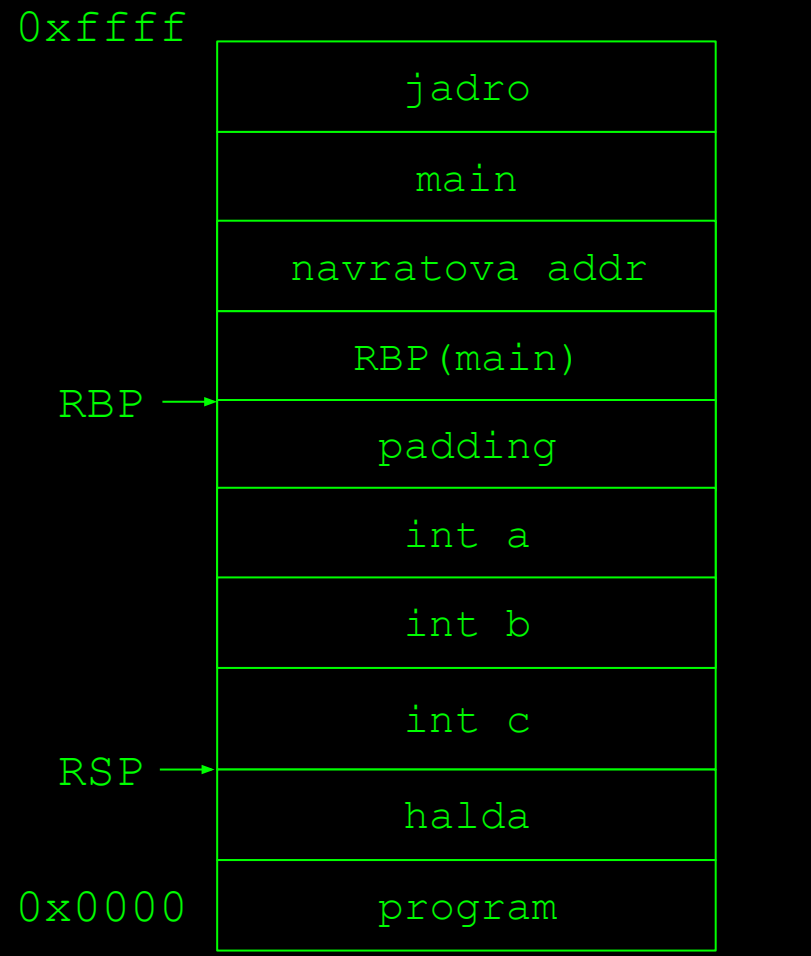
```
call foo

push rbp
mov rbp, rsp
sub rsp,0x10
mov[rbp-8],1
mov[rbp-12],2
```



# >zasobnik

```
int foo()
{
    int a,b,c;
    b=1;c=2;
    a=b+c;
    return a;
}

int main()
{
    foo();
    return 0;
}
```

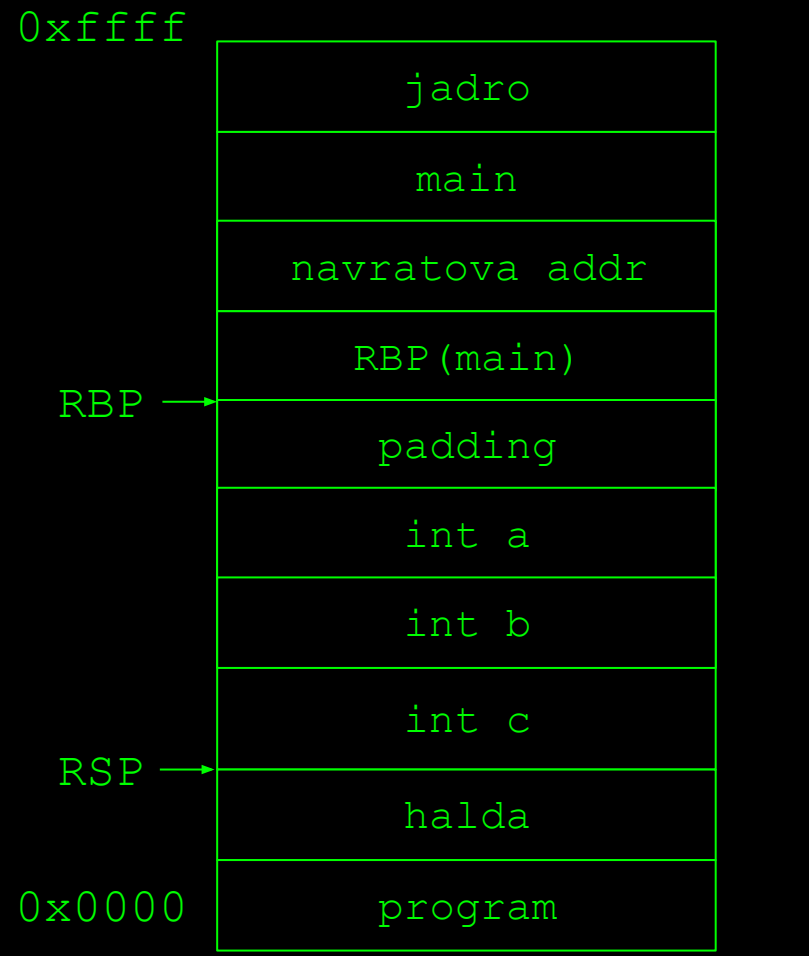
```
call foo

push rbp
mov rbp, rsp
sub rsp,0x10
mov[rbp-8],1
mov[rbp-12],2

vypocty…
```



# >zasobnik

```
int foo()
{
    int a,b,c;
    b=1;c=2;
    a=b+c;
    return a;
}

int main()
{
    foo();
    return 0;
}
```

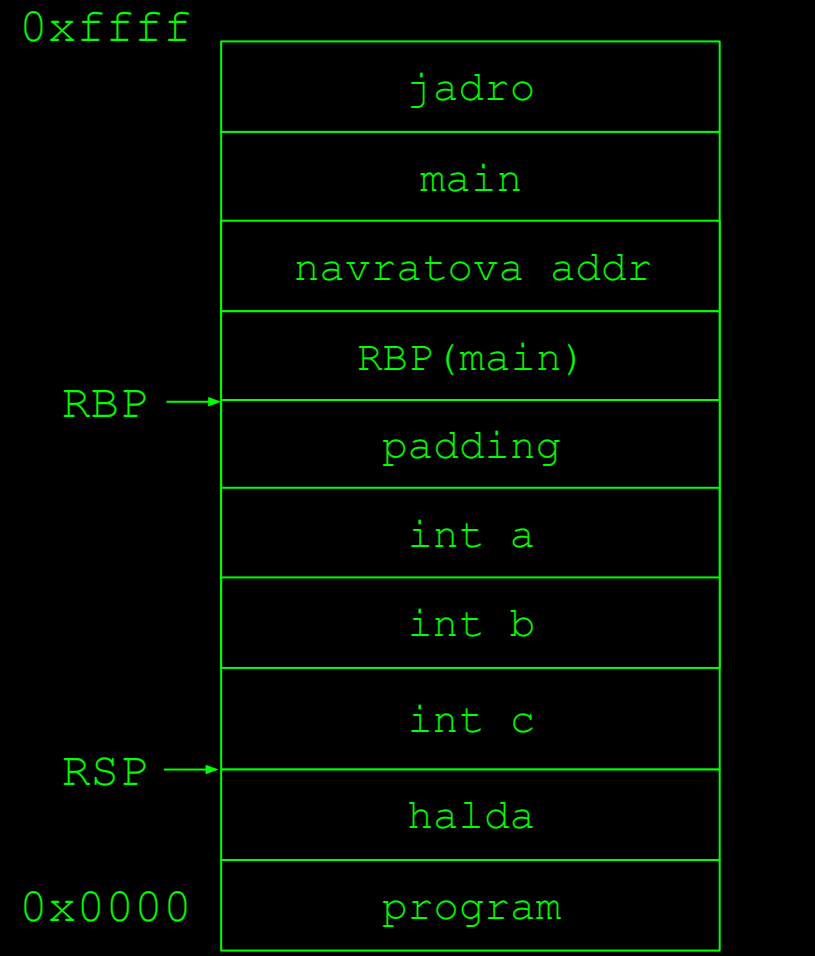
```
call foo

push rbp
mov rbp, rsp
sub rsp,0x10
mov[rbp-8],1
mov[rbp-12],2

vypocty…

mov rax,[rbp-4]
```



# >zasobnik

```
int foo()
{
    int a,b,c;
    b=1;c=2;
    a=b+c;
    return a;
}

int main()
{
    foo();
    return 0;
}
```

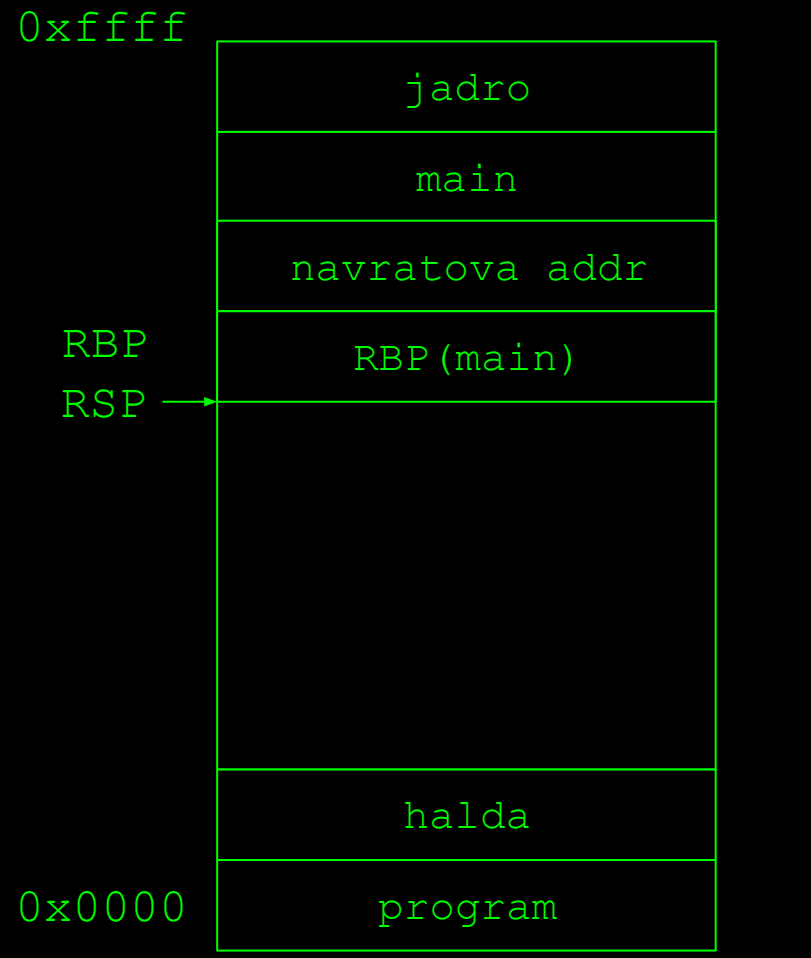
```
call foo

push rbp
mov rbp, rsp
sub rsp,0x10
mov[rbp-8],1
mov[rbp-12],2

vypocty…

mov rax,[rbp-4]
mov rsp, rbp
```



# >zasobnik

```
int foo()
{
    int a,b,c;
    b=1;c=2;
    a=b+c;
    return a;
}

int main()
{
    foo();
    return 0;
}
```

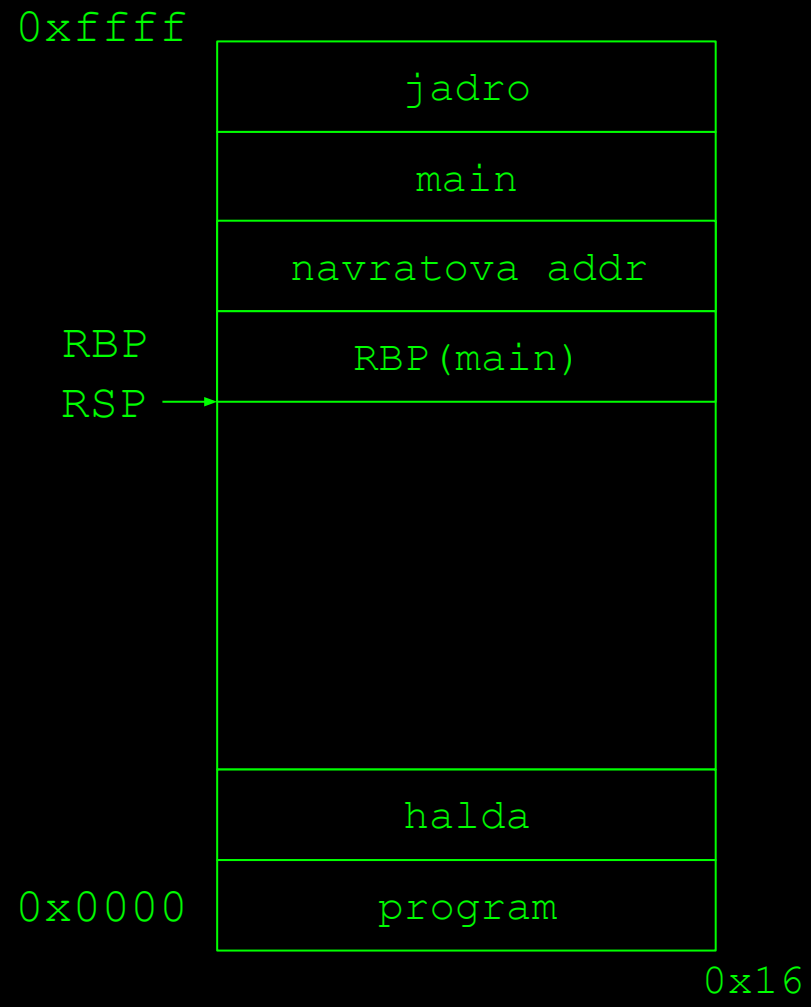
```
call foo

push rbp
mov rbp, rsp
sub rsp,0x10
mov[rbp-8],1
mov[rbp-12],2

vypocty…

mov rax,[rbp-4]
mov rsp, rbp
```



# >zasobnik

```
int foo()
{
    int a,b,c;
    b=1;c=2;
    a=b+c;
    return a;
}

int main()
{
    foo();
    return 0;
}
```

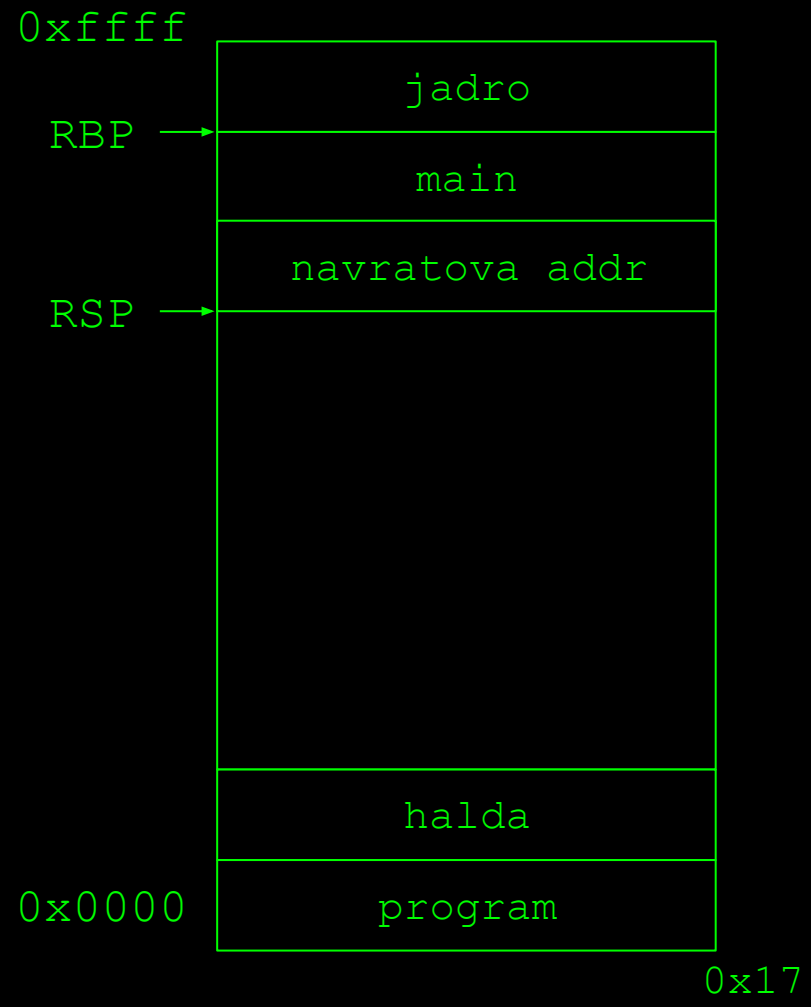
```
call foo

push rbp
mov rbp, rsp
sub rsp,0x10
mov[rbp-8],1
mov[rbp-12],2

vypocty…

mov rax,[rbp-4]
mov rsp, rbp
pop rbp
```



# >zasobnik

```
int foo()
{
    int a,b,c;
    b=1;c=2;
    a=b+c;
    return a;
}

int main()
{
    foo();
    return 0;
}
```

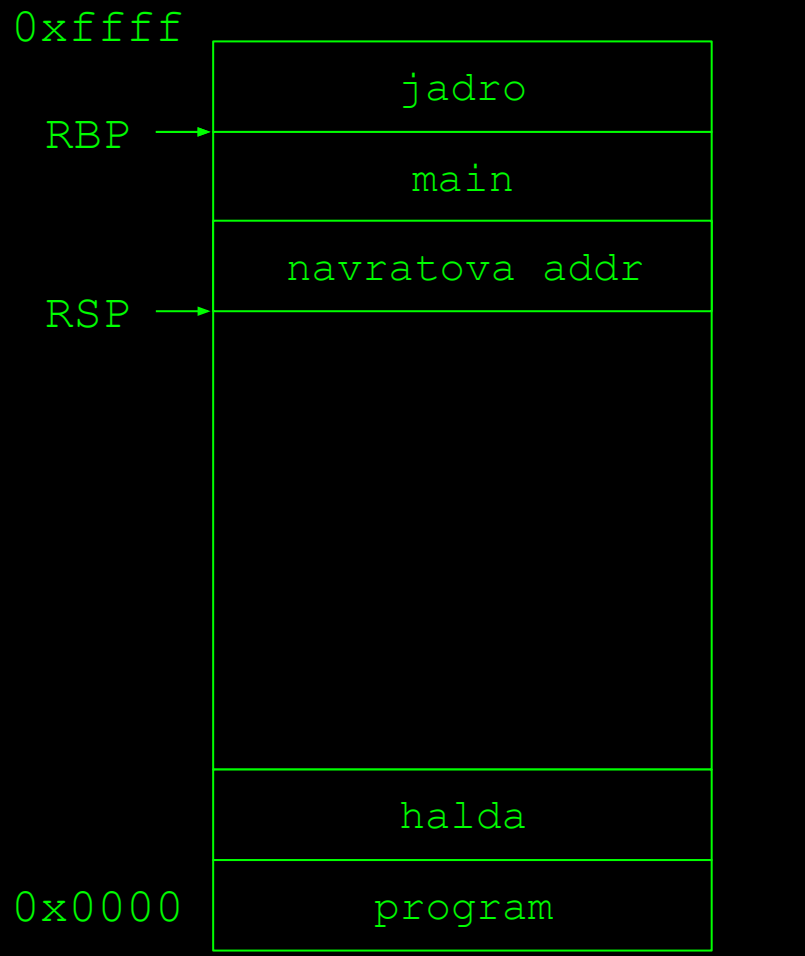
```
call foo

push rbp
mov rbp, rsp
sub rsp,0x10
mov[rbp-8],1
mov[rbp-12],2

vypocty…

mov rax,[rbp-4]
mov rsp, rbp
pop rbp
```



# >zasobnik

```
int foo()
{
    int a,b,c;
    b=1;c=2;
    a=b+c;
    return a;
}

int main()
{
    foo();
    return 0;
}
```

```
call foo

push rbp
mov rbp, rsp
sub rsp,0x10
mov[rbp-8],1
mov[rbp-12],2

vypocty…

mov rax,[rbp-4]
mov rsp, rbp
pop rbp
ret
```

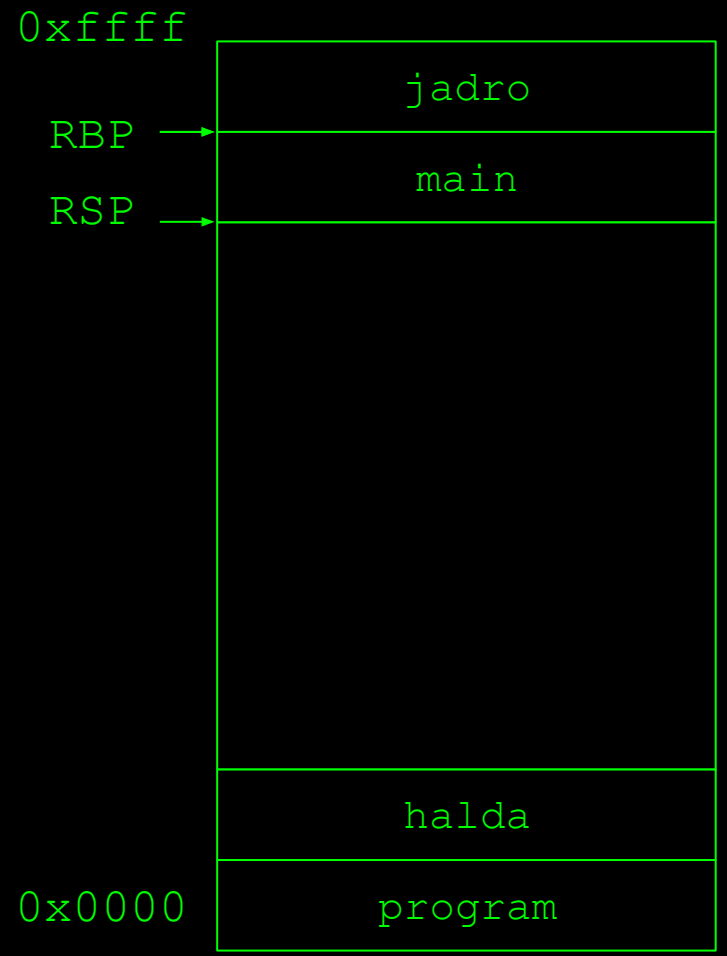



ret = nacita do RIP vrch zasobnika

# >zasobnik

```
int foo()
{
    int a,b,c;
    b=1;c=2;
    a=b+c;
    return a;
}

int main()
{
    foo();
    return 0;
}
```

```
call foo

push rbp
mov rbp, rsp
sub rsp,0x10
mov[rbp-8],1
mov[rbp-12],2

vypocty…

mov rax,[rbp-4]
mov rsp, rbp
pop rbp
ret
```




ret = nacita do RIP vrch zasobnika

# >nastroje

>staticka analyza (bez spustenia)
- >Binary Ninja[1] (cloud verzia)
- >IDA Free 7.6
- >Ghidra
- >Radare2

>dynamicka analyza (so spustenim)
- >strace
- >GDB

[1]https://cloud.binary.ninja/

# >ulohy

>cielom uloh bude zreverzovat binarky a najst licencny kluc
    >najst komunikacny kanal
    >zreverzovat trasnformacie a format
>binarky su dostupne aj na githube[2] (mozete dat hviezdicku)
>analyticke ulohy
    >ziadne programovnie, t.j. odovzdat **kratku** dokumentaciu

[2] https://github.com/petersvec/feictf-2022/tree/master/module_2

# >vela stastia


0x1c