OS 2018 v. 02

MIT ;)

https://pdos.csail.mit.edu/6.828/2018

0. POCITAC x86

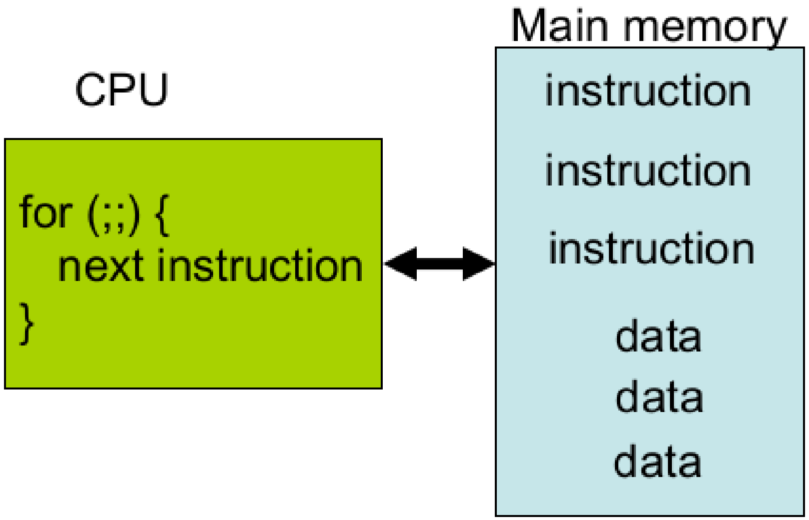
Schema

- Von Neumann schema PC

- CPU
- Memory
- I/0
- zbernica

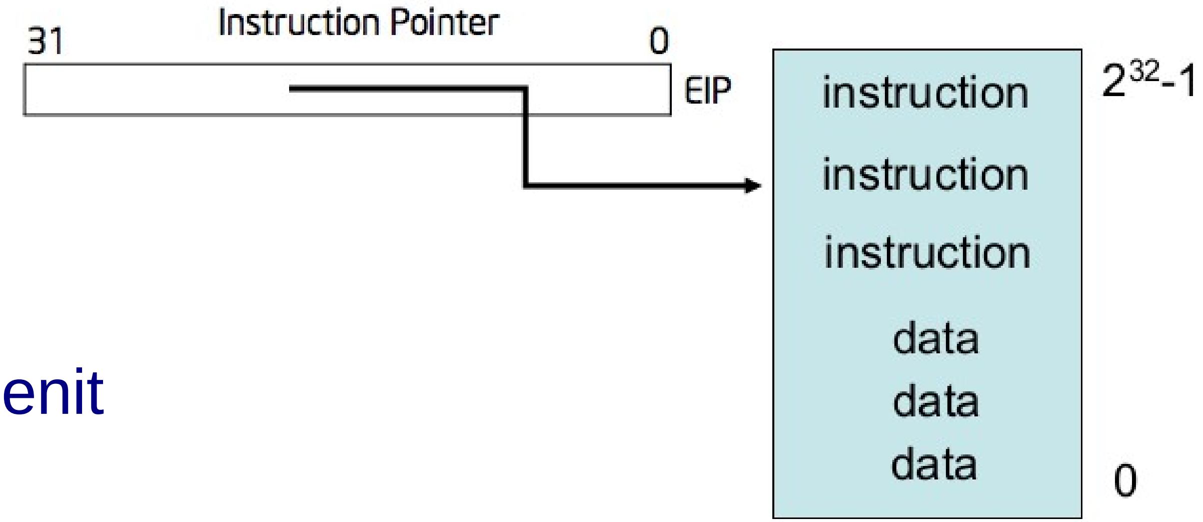
Program

- Pamat
  - Instrukcie
  - Udaje
- CPU
  - Interpretacia
  - Manipulacia



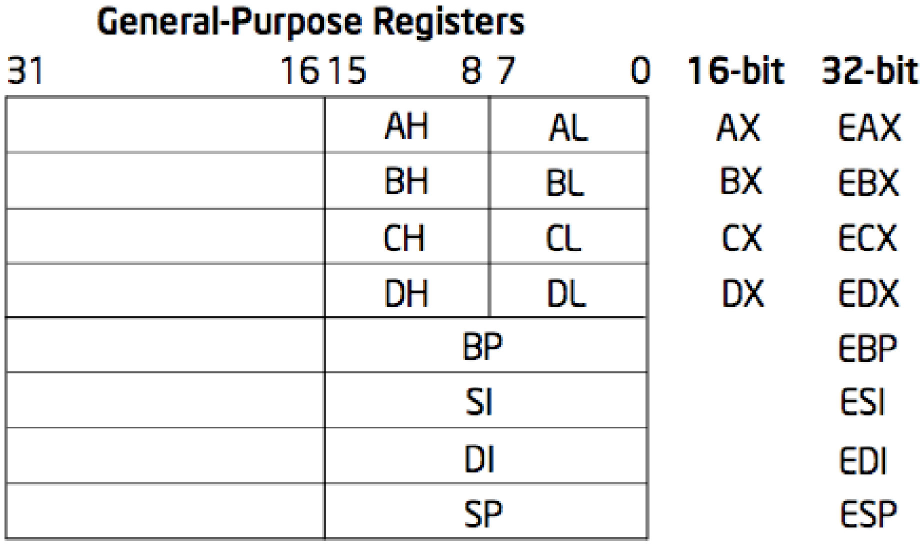
EIP

- Zvyseny po kazdej instrukcii
- Rozlicna dlzka instrukcii
- Automaticky modifikovany
  - Call
  - Ret
  - Jmp
- Neda sa priamo menit



Registre CPU

- 8, 16, 32 bit
- Specialny ucel pre niektore
- Spatna kompatibilita

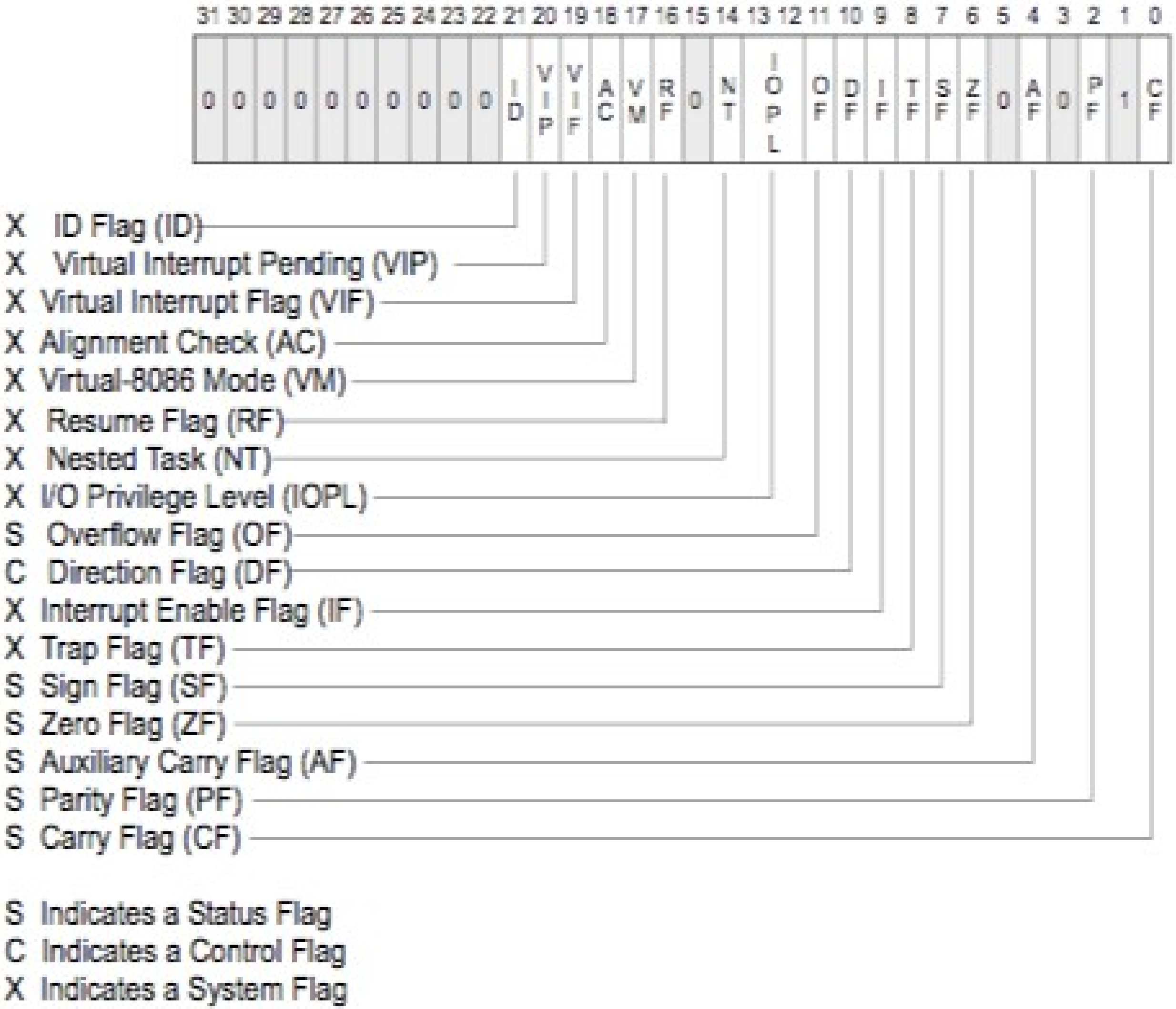


General-Purpose Registers

31 16	15 8	7 0	16-bit	32-bit
	AH	AL	AX	EAX
	BH	BL	BX	EBX
	CH	CL	CX	ECX
	DH	DL	DX	EDX
	BP			EBP
	SI			ESI
	DI			EDI
	SP			ESP

EFLAGS

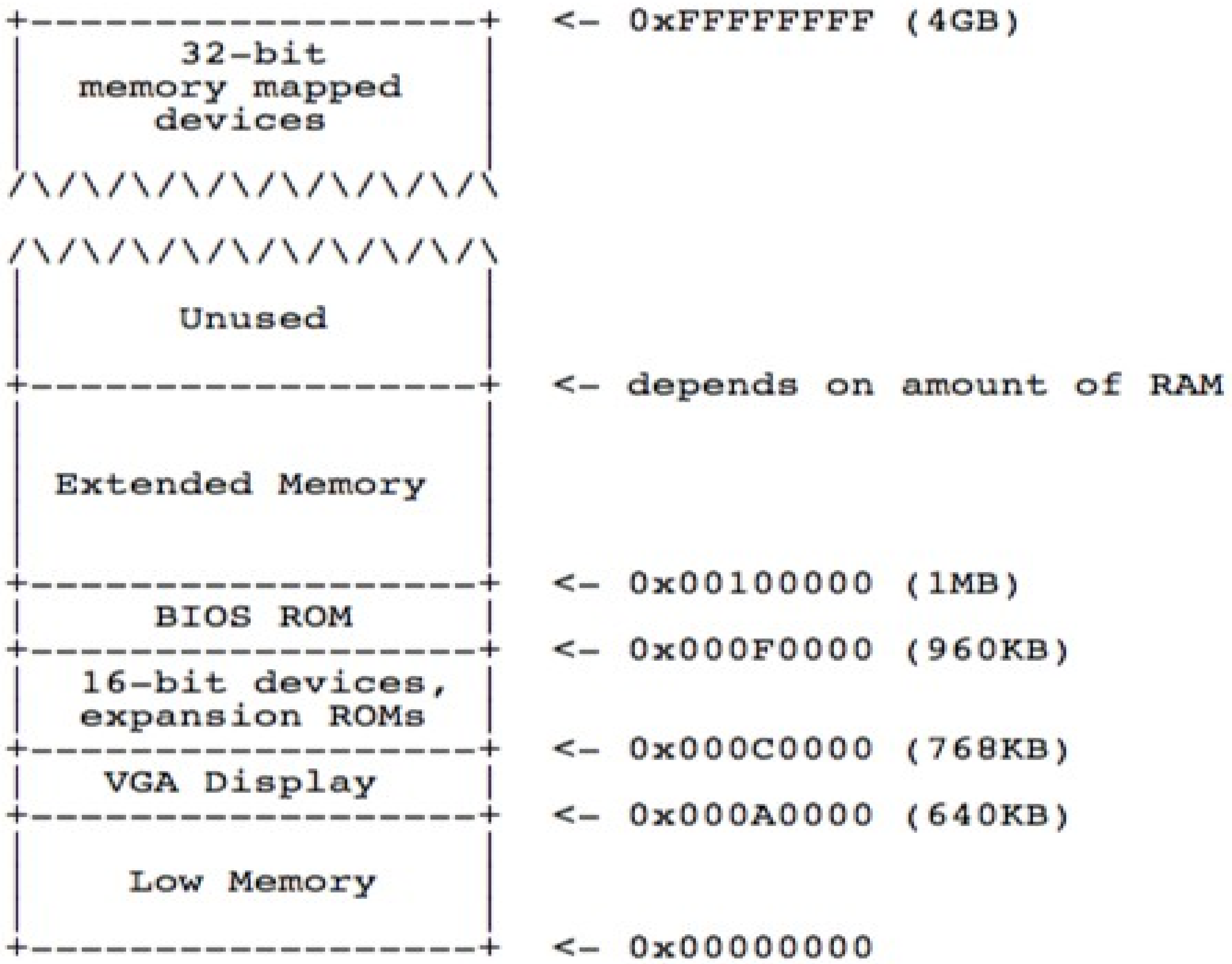
- TEST
- CMP
- Conditional JMP



Zasobnikove instrukcie

- Push
- Pop
- Call
- Ret

Fyzicka pamat



```
+------------------+  <- 0xFFFFFFFF (4GB)
|      32-bit      |
|  memory mapped   |
|     devices      |
|                  |
/\/\/\/\/\/\/\/\/\/\

/\/\/\/\/\/\/\/\/\/\
|                  |
|      Unused      |
|                  |
+------------------+  <- depends on amount of RAM
|                  |
|                  |
| Extended Memory  |
|                  |
|                  |
+------------------+  <- 0x00100000 (1MB)
|     BIOS ROM     |
+------------------+  <- 0x000F0000 (960KB)
|  16-bit devices, |
|  expansion ROMs  |
+------------------+  <- 0x000C0000 (768KB)
|   VGA Display    |
+------------------+  <- 0x000A0000 (640KB)
|                  |
|    Low Memory    |
|                  |
+------------------+  <- 0x00000000
```

Instrukcna sada x86

- Prenos udajov (mov, push, pop, ...)
- Aritmetika (test, shl, add, sub, …)
- I/O (in, out, ...)
- Riadiace (jmp, jz, jnz, call, ret, …)
- Retazcove (rep, movsb, scasw, …)
- Systemove (iret, int, ...)

1. cast: ZAVEDENIE JADRA

BIOS

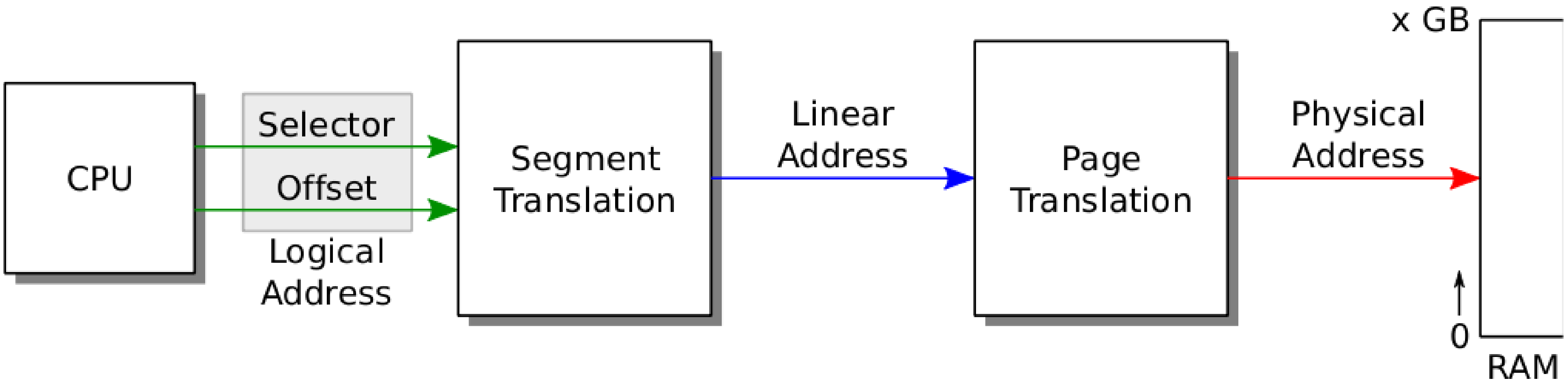
- Pripravit hw
- Najst bootovatelne medium
- Nacitat zavadzac z media na adresu 0x7c00
- Dat mu riadenia

Boot loader

- Prepnutie CPU do modernejsieho modu adresovania
- Nacitanie jadra do pamati
- Dat riadenie jadru

Boot loader - asm

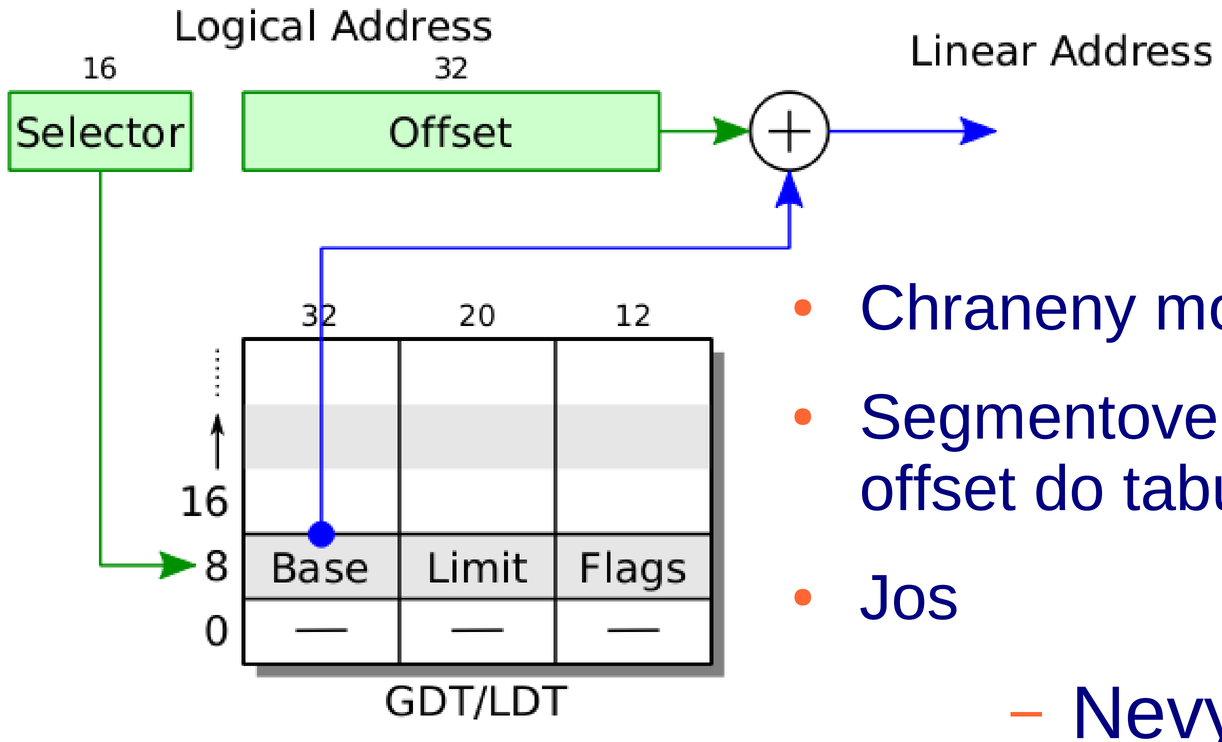
- Cli – prerusenia hw, Cld – praca s retazcami...
- Realny mod CPU
  - 20 bitove adresy z 2 16 bitovych reg
  - CS, DS, ES, SS
  - Logicka, linearna a fyzicka adresa
  - Virtualne adresy



Boot loader - asm

- Obmedzenie realneho modu adresacie
  - 64kB per program
  - 1MB per fyzicka pamat
- Trik A20 pre zapnutie ostatnych adresnych bitov CPU!
  - BIOS
  - Klavesnicovy radic (JOS)
  - Tzv. “rychle” zapnutie cez port

Boot loader - asm



- Chraneny mod CPU
- Segmentove registre (selector) – offset do tabulky (base, limit, flags)
- Jos
  - Nevyuziva preklad cez tabulku
  - 3 polozky ma tabulka

Boot loader – asm a C

- Nastavenie zasobnika
- Spustenie bootmain()
  - Nacitat a spustit jadro
  - Co v pripade, ze 1. sektor neobsahuje kod?
  - Jadro: elf format, od 2. sektora na disku
  - Nacitanie sekcii do pamate
  - Spustenie jadra

Jadro

- Jadro je v pamati od 0x100000, nie od 0xf0100000
- Musi sa zapnut preklad adries (strankovanie)
- Mapuju sa dva regiony!!! Preco?

- Nastavenie registra ebp pre spravny “stack trace”
- Nastavenie zasobnika (registra esp)
- Spustenie kodu v C - funkcie i386_init()

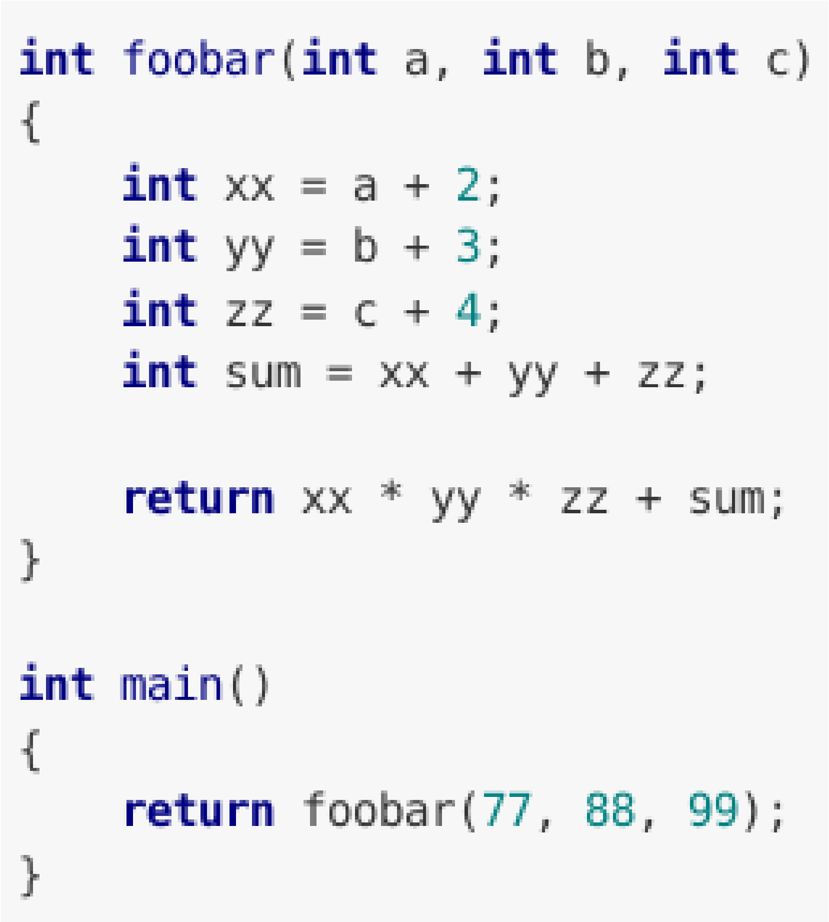
2. cast: ZASOBNIK na x86

Konvencia volani - cdecl

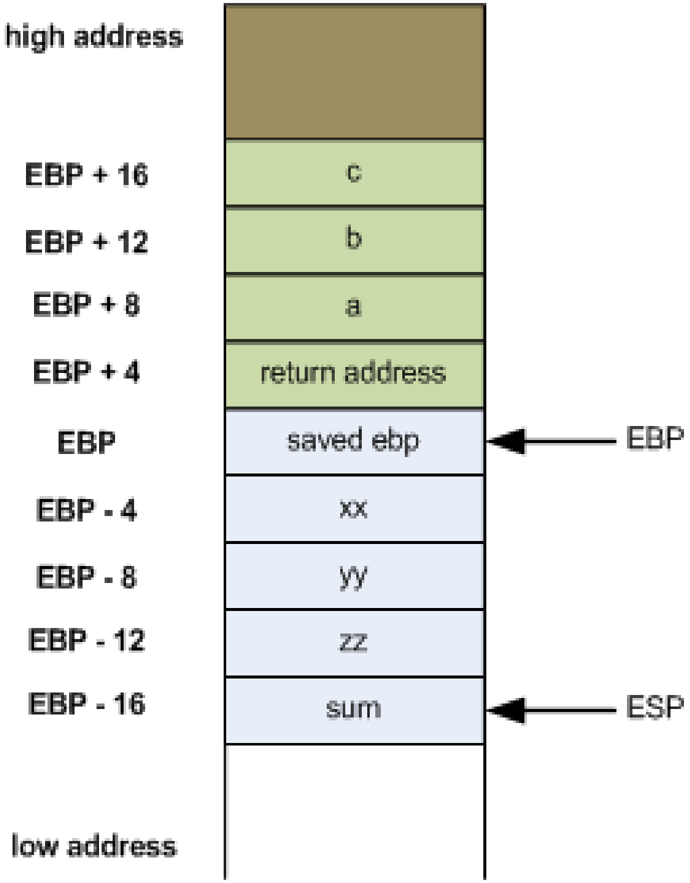
```
int volany(int, int, int);

int volajuci(void) {

    int ret;

    ret = volany(1, 2, 3);

    ret += 5;

    return ret;

}
```

```
volajuci:
    ; novy stack frame
    push ebp
    mov ebp, esp
    ; uloz argumenty
    push 3
    push 2
    push 1
    ; zavolaj 'volaneho'
    call volany
    ; zrus argumenty
    add esp, 12
    ; pripocitaj 5
    add eax, 5
    ; obnov povodny stack frame
    pop ebp
    ; navrat
    ret
```

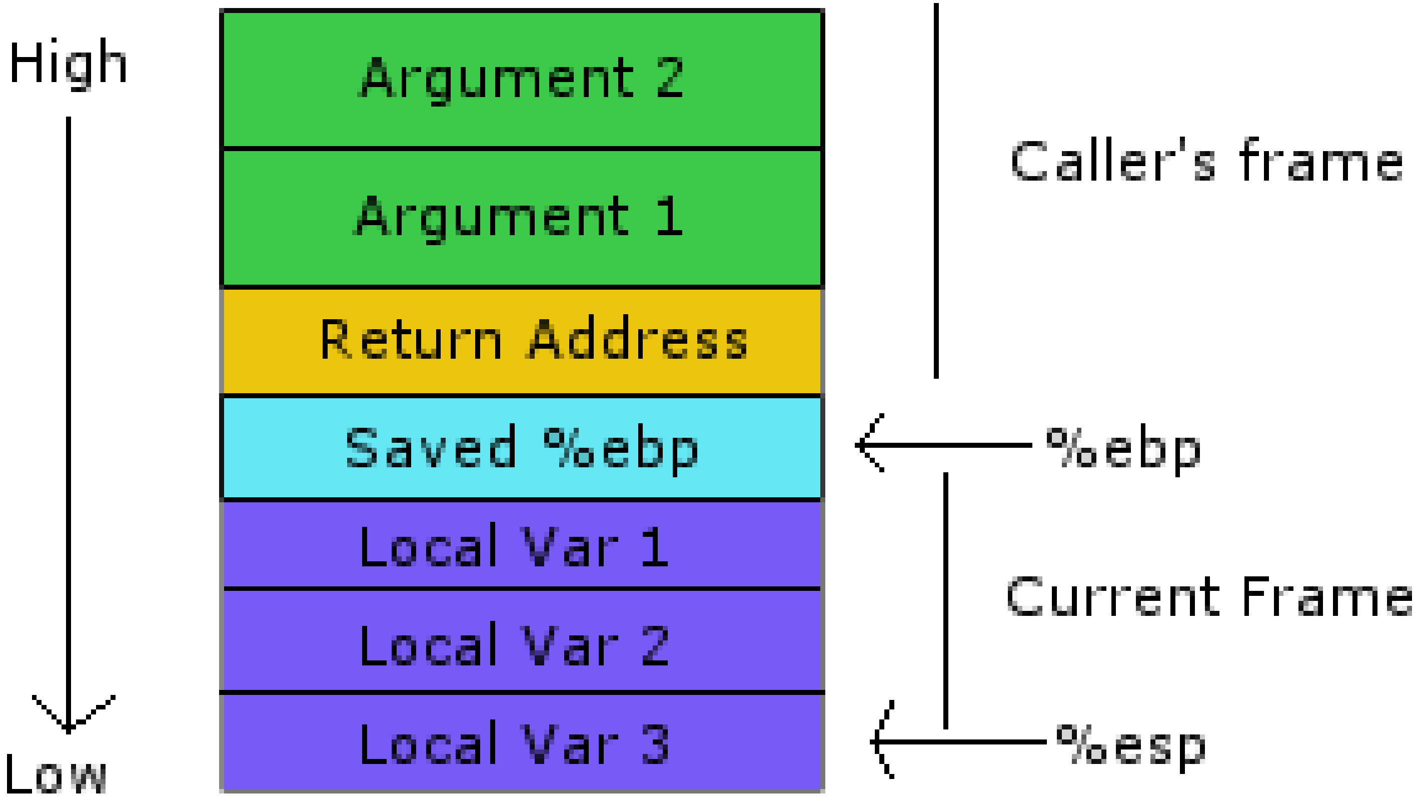
Zasobnik



```c
int foobar(int a, int b, int c)
{
    int xx = a + 2;
    int yy = b + 3;
    int zz = c + 4;
    int sum = xx + yy + zz;

    return xx * yy * zz + sum;
}

int main()
{
    return foobar(77, 88, 99);
}
```



Zasobnik a volanie funkcie



High

Low

Argument 2

Argument 1

Return Address

Saved %ebp

Local Var 1

Local Var 2

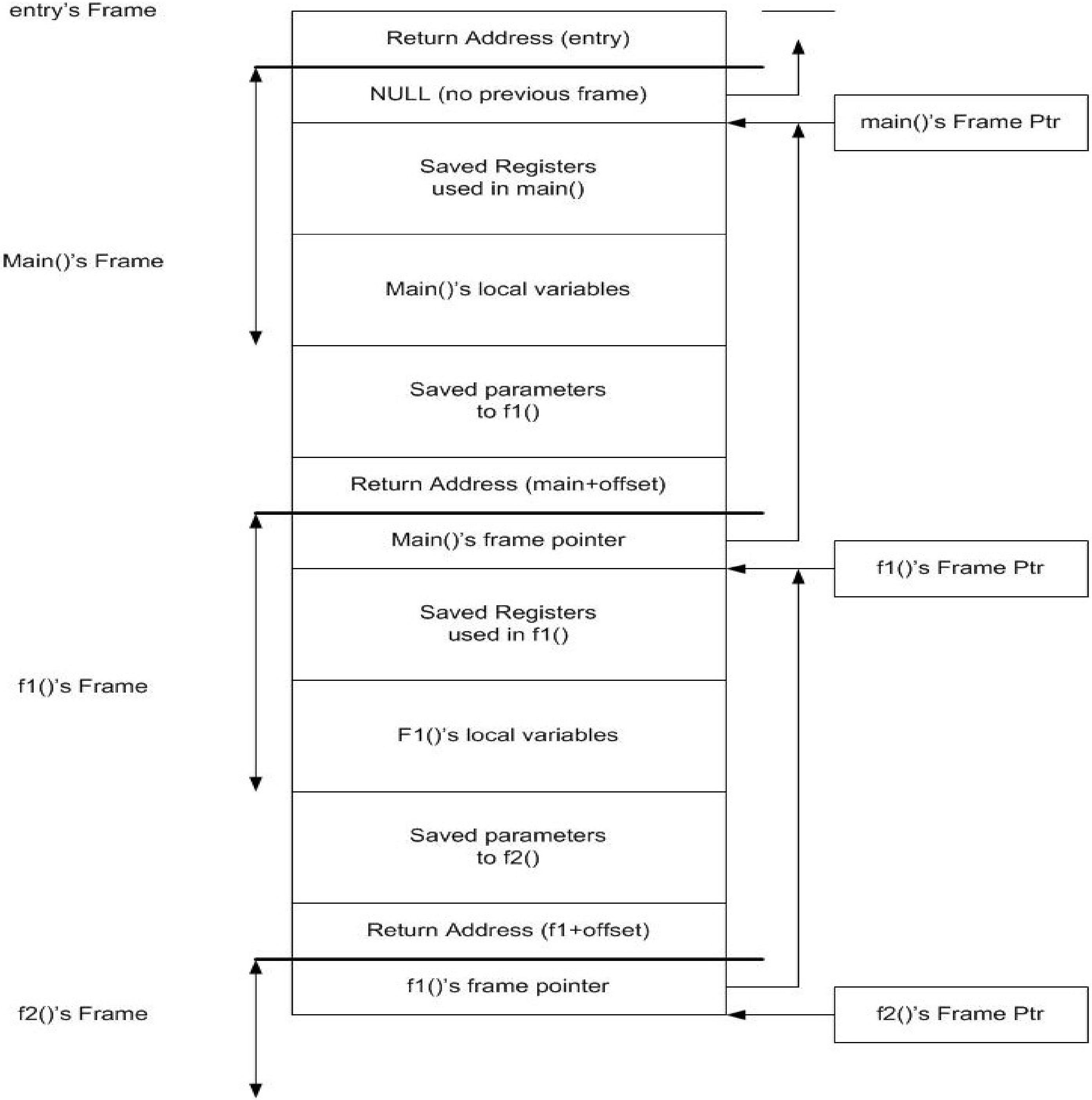
Local Var 3

Caller's frame

%ebp

Current Frame

%esp



entry's Frame

Return Address (entry)

NULL (no previous frame)

main()'s Frame Ptr

Saved Registers
used in main()

Main()'s Frame

Main()'s local variables

Saved parameters
to f1()

Return Address (main+offset)

Main()'s frame pointer

f1()'s Frame Ptr

Saved Registers
used in f1()

f1()'s Frame

F1()'s local variables

Saved parameters
to f2()

Return Address (f1+offset)

f1()'s frame pointer

f2()'s Frame

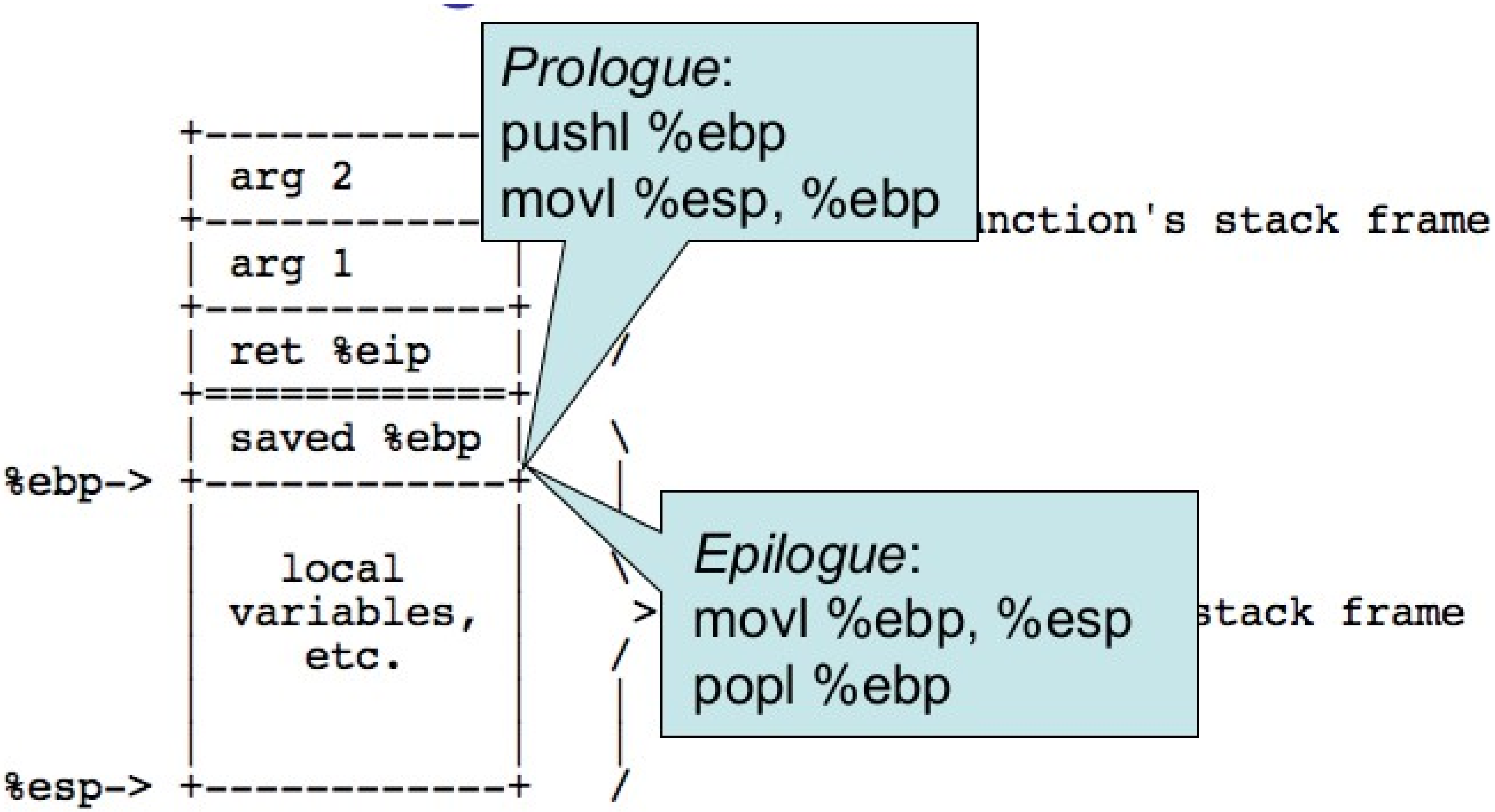
f2()'s Frame Ptr

Prolog a Epilog funkcie

```
int volany(int, int, int);

int volajuci(void) {

    int ret;

    ret = volany(1, 2, 3);

    ret += 5;

    return ret;

}
```

```
volajuci:
    ; novy stack frame
    push ebp
    mov ebp, esp
    ; uloz argumenty
    push 3
    push 2
    push 1
    ; zavolaj 'volaneho'
    call volany
    ; zrus argumenty
    add esp, 12
    ; pripocitaj 5
    add eax, 5
    ; obnov povodny stack frame
    pop ebp
    ; navrat
    ret
```

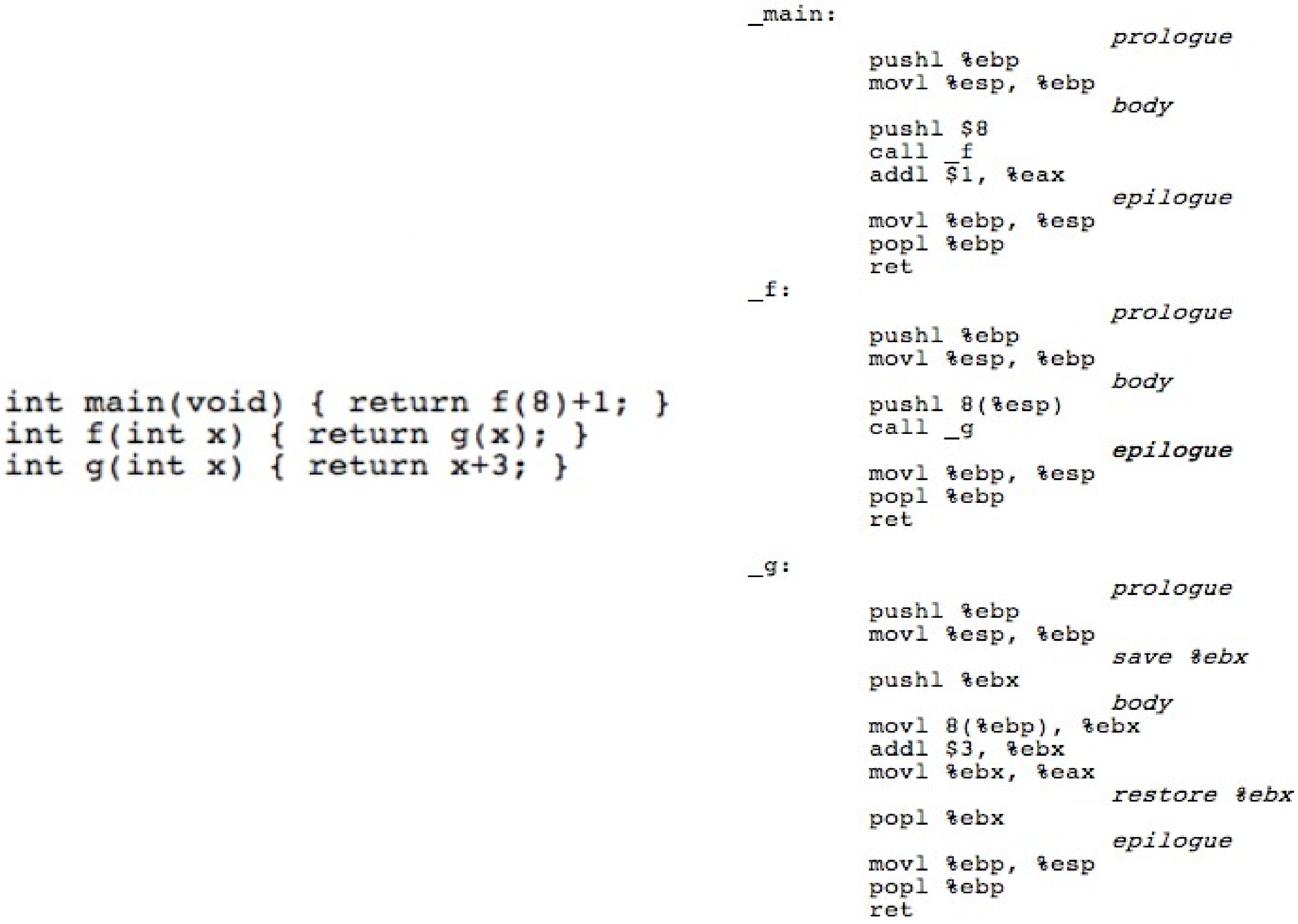
Dalsi obrazok



Prologue:
pushl %ebp
movl %esp, %ebp

Epilogue:
movl %ebp, %esp
popl %ebp

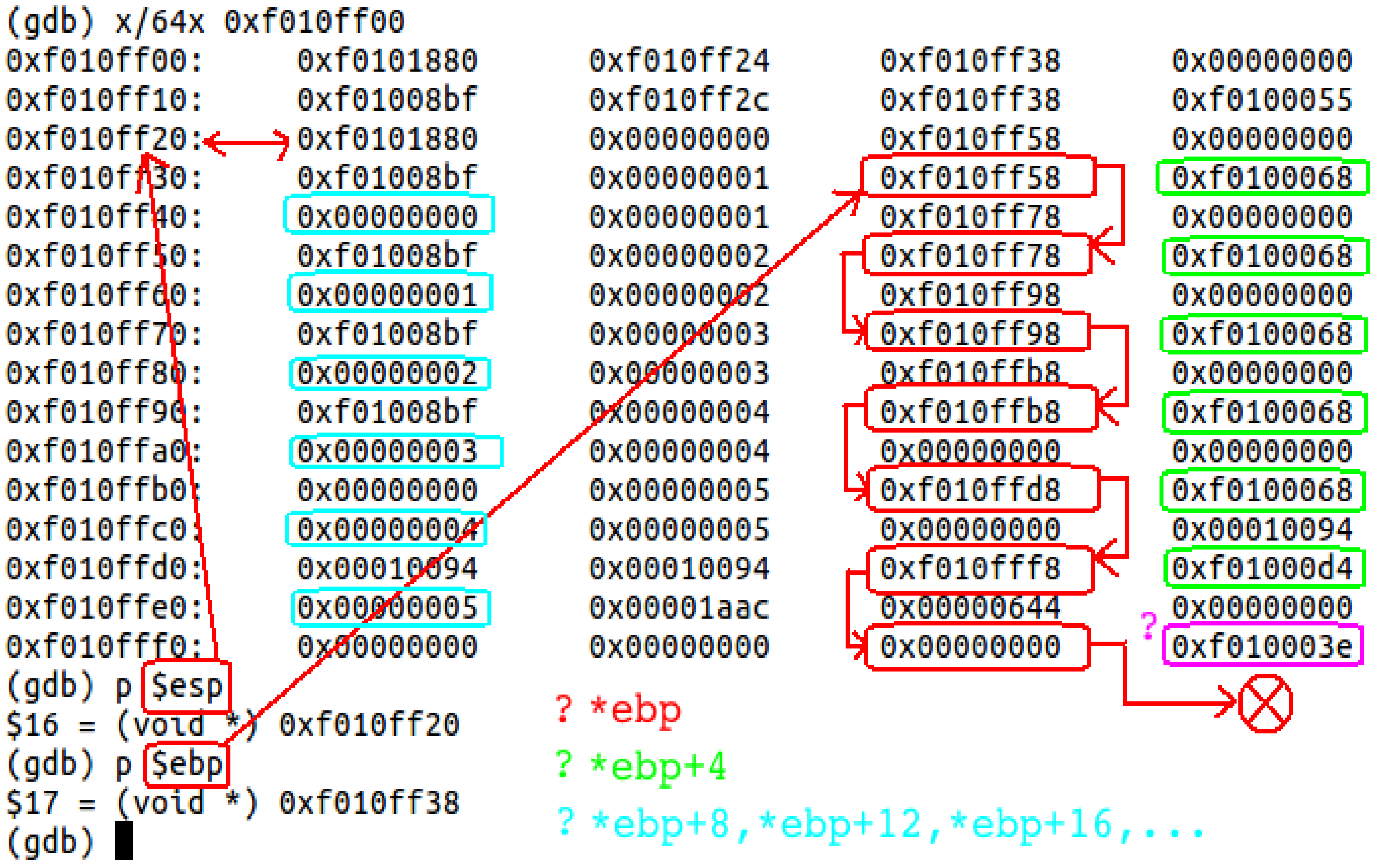
Priklad



```
int main(void) { return f(8)+1; }
int f(int x) { return g(x); }
int g(int x) { return x+3; }
```

```
_main:
                                prologue
                pushl %ebp
                movl %esp, %ebp
                                body
                pushl $8
                call _f
                addl $1, %eax
                                epilogue
                movl %ebp, %esp
                popl %ebp
                ret
_f:
                                prologue
                pushl %ebp
                movl %esp, %ebp
                                body
                pushl 8(%esp)
                call _g
                                epilogue
                movl %ebp, %esp
                popl %ebp
                ret

_g:
                                prologue
                pushl %ebp
                movl %esp, %ebp
                                save %ebx
                pushl %ebx
                                body
                movl 8(%ebp), %ebx
                addl $3, %ebx
                movl %ebx, %eax
                                restore %ebx
                popl %ebx
                                epilogue
                movl %ebp, %esp
                popl %ebp
                ret
```

test_backtrace() v JOS

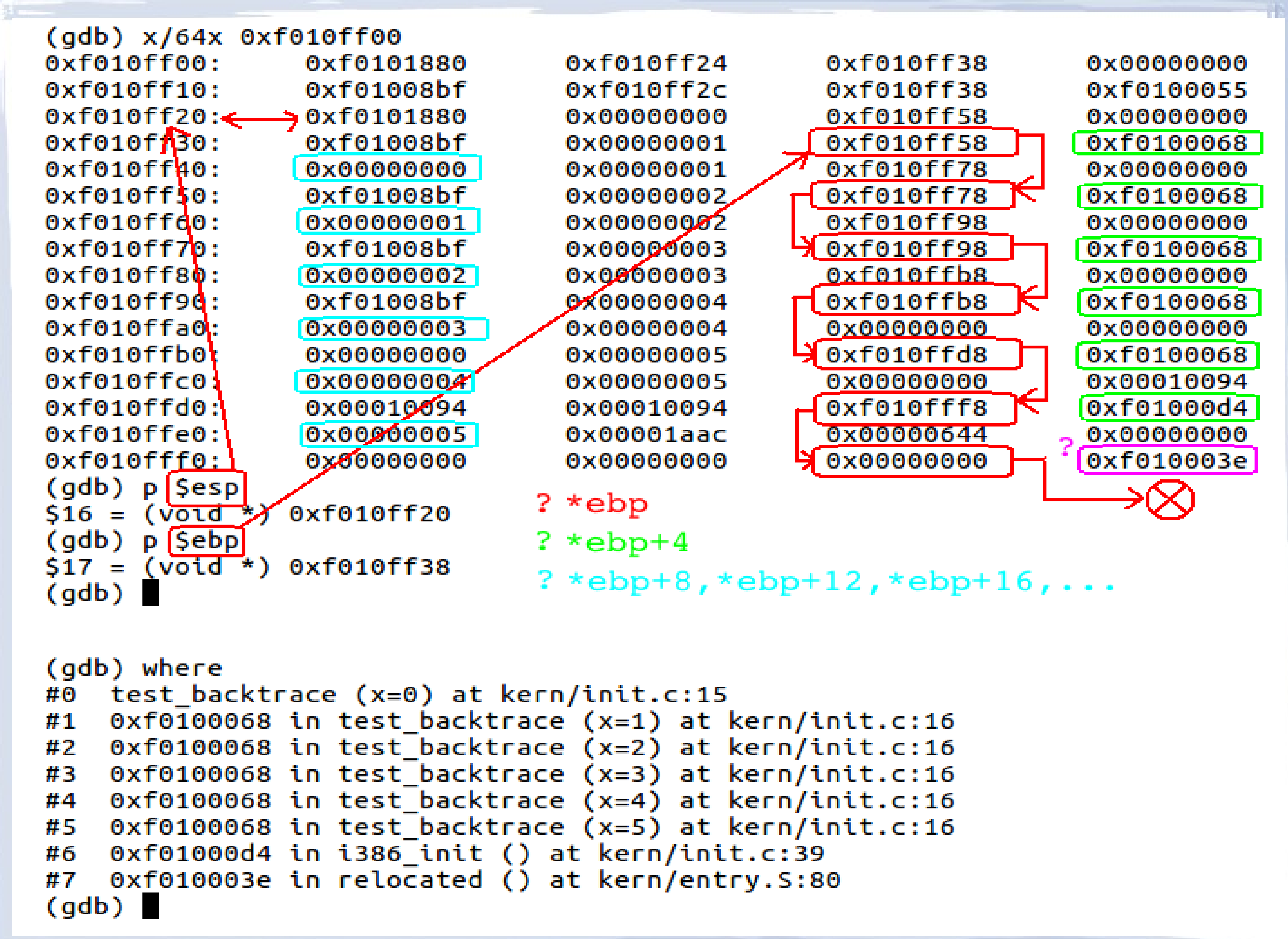


```
(gdb) x/64x 0xf010ff00
0xf010ff00:     0xf0101880      0xf010ff24      0xf010ff38      0x00000000
0xf010ff10:     0xf01008bf      0xf010ff2c      0xf010ff38      0xf0100055
0xf010ff20:     0xf0101880      0x00000000      0xf010ff58      0x00000000
0xf010ff30:     0xf01008bf      0x00000001      0xf010ff58      0xf0100068
0xf010ff40:     0x00000000      0x00000001      0xf010ff78      0x00000000
0xf010ff50:     0xf01008bf      0x00000002      0xf010ff78      0xf0100068
0xf010ff60:     0x00000001      0x00000002      0xf010ff98      0x00000000
0xf010ff70:     0xf01008bf      0x00000003      0xf010ff98      0xf0100068
0xf010ff80:     0x00000002      0x00000003      0xf010ffb8      0x00000000
0xf010ff90:     0xf01008bf      0x00000004      0xf010ffb8      0xf0100068
0xf010ffa0:     0x00000003      0x00000004      0x00000000      0x00000000
0xf010ffb0:     0x00000000      0x00000005      0xf010ffd8      0xf0100068
0xf010ffc0:     0x00000004      0x00000005      0x00000000      0x00010094
0xf010ffd0:     0x00010094      0x00010094      0xf010fff8      0xf01000d4
0xf010ffe0:     0x00000005      0x00001aac      0x00000644      0x00000000
0xf010fff0:     0x00000000      0x00000000      0x00000000      0xf010003e
(gdb) p $esp
$16 = (void *) 0xf010ff20
(gdb) p $ebp
$17 = (void *) 0xf010ff38
(gdb) 
```

? *ebp

? *ebp+4

? *ebp+8,*ebp+12,*ebp+16,...



```
(gdb) x/64x 0xf010ff00
0xf010ff00:     0xf0101880      0xf010ff24      0xf010ff38      0x00000000
0xf010ff10:     0xf01008bf      0xf010ff2c      0xf010ff38      0xf0100055
0xf010ff20:     0xf0101880      0x00000000      0xf010ff58      0x00000000
0xf010ff30:     0xf01008bf      0x00000001      0xf010ff58      0xf0100068
0xf010ff40:     0x00000000      0x00000001      0xf010ff78      0x00000000
0xf010ff50:     0xf01008bf      0x00000002      0xf010ff78      0xf0100068
0xf010ff60:     0x00000001      0x00000002      0xf010ff98      0x00000000
0xf010ff70:     0xf01008bf      0x00000003      0xf010ff98      0xf0100068
0xf010ff80:     0x00000002      0x00000003      0xf010ffb8      0x00000000
0xf010ff90:     0xf01008bf      0x00000004      0xf010ffb8      0xf0100068
0xf010ffa0:     0x00000003      0x00000004      0x00000000      0x00000000
0xf010ffb0:     0x00000000      0x00000005      0xf010ffd8      0xf0100068
0xf010ffc0:     0x00000004      0x00000005      0x00000000      0x00010094
0xf010ffd0:     0x00010094      0x00010094      0xf010fff8      0xf01000d4
0xf010ffe0:     0x00000005      0x00001aac      0x00000644      0x00000000
0xf010fff0:     0x00000000      0x00000000      0x00000000      0xf010003e
(gdb) p $esp
$16 = (void *) 0xf010ff20
(gdb) p $ebp
$17 = (void *) 0xf010ff38
(gdb) 
```

```
(gdb) where
#0  test_backtrace (x=0) at kern/init.c:15
#1  0xf0100068 in test_backtrace (x=1) at kern/init.c:16
#2  0xf0100068 in test_backtrace (x=2) at kern/init.c:16
#3  0xf0100068 in test_backtrace (x=3) at kern/init.c:16
#4  0xf0100068 in test_backtrace (x=4) at kern/init.c:16
#5  0xf0100068 in test_backtrace (x=5) at kern/init.c:16
#6  0xf01000d4 in i386_init () at kern/init.c:39
#7  0xf010003e in relocated () at kern/entry.S:80
(gdb) 
```