8080

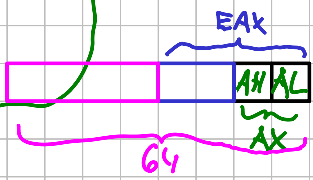
8086 ⇢ i386 → AMD64

16-bit | x86 32-bit | x86_64 64-bit

↳ nás zajímá 32b mód s lineární adresací

(i386)

Registry:



		64b
int	EAX	RAX
	EBX	RBX
	ECX	RCX
	EDX	RDX
	ESI (source index)	RSI
	EDI (dest. index)	RDI
	EBP (base pointer)	RBP
	ESP (stack pointer)	RSP
	EIP (instruction ptr)	RIP
	EFLAGS	RFLAGS
float	ST0–ST7 (float stack)	
	FPUSR (FPU status reg.)	
	FPUCR (FPU control reg.)	
vector	XMM0–7	XMM8–15
	YMM0–7 (256b rozšíření)	YMM 8–15
	MXCSR (control & status reg.)	

R0–7, navíc R8–15

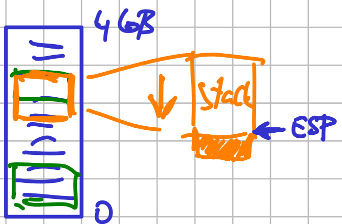
Datové typy:

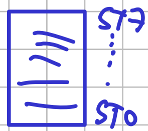
little-endian

- integer 8/16/32/(64) — 64/(128)
- float
  - 32 — 24+8 — float
  - 64 — 53+11 — double
  - 80 — 64+16 — long double
  - (16 — 11+5)
- vektory 128, 256

Paměť: pole bytů

společná pro kód + data





Intel ADD EBX,1

AT&T addl $1, %ebx

q (64b), long (32b), w (16b), b (8b)

pro srovnání:

ADD DWORD PTR [EBX],1

addl $1, (%ebx)

→ operandy:

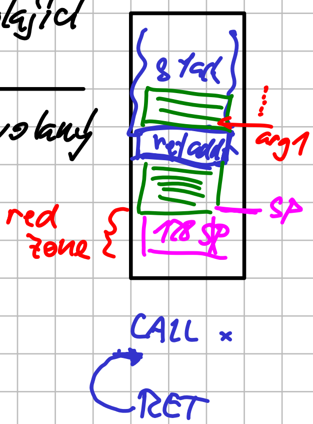
- $ číslo ... literál
- % registr
- číslo ... adresa v paměti
- (%reg) ... adresace registrem
- číslo(%reg) ... adresa = reg + číslo
- číslo(%r1, %r2, mul) — mul: 1,2,4,8 ... adresa = r1 + mul·r2 + číslo
- implicitní

ABI (Application Binary Interface) pro i386 a Linux a C

↳ volací konvence

stack frame

	⋮	
+16	arg3	volající
+12	arg2	
+8	arg1	
EBP+4	návratová adresa	
EBP	původní hodnota EBP	volaný
−4	lokální proměnné	
	pracovní prostor	
ESP →		



výsledek: EAX / EDX:EAX

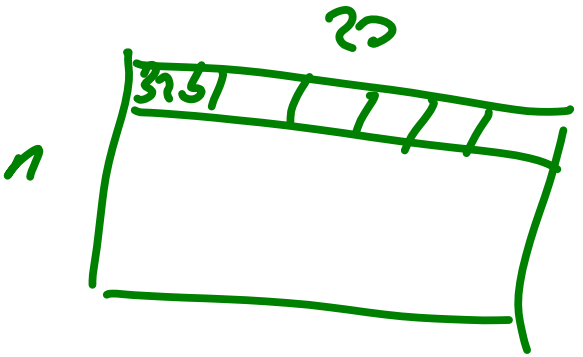
ST0 float

Scratch: EAX, ECX, EDX, ST0–7, část EFLAGS

Zdrojový text

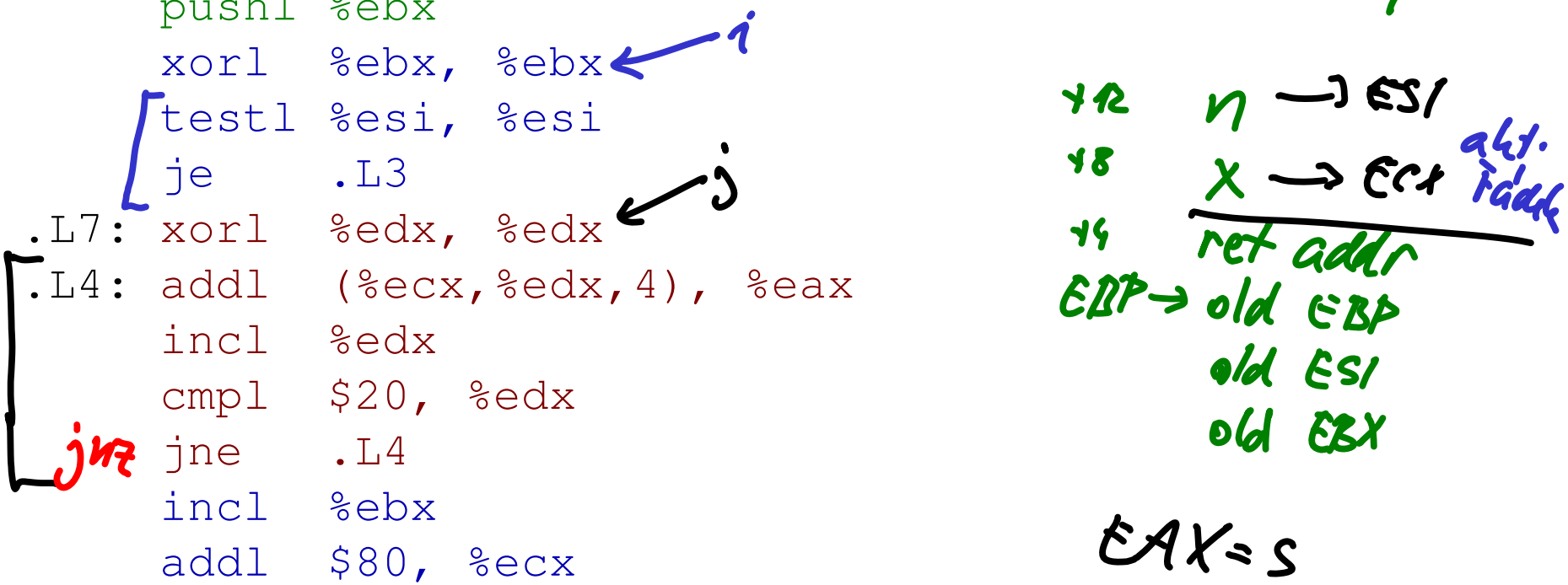
```
typedef unsigned int uint;

uint f(uint x[][20], uint n)
{
  uint s = 0;
  for (uint i=0; i<n; i++)
    for (uint j=0; j<20; j++)
      s += x[i][j];
  return s;
}
```



32-bitový assembler: `-march=athlon -O2`

```
f:    pushl %ebp
      xorl  %eax, %eax
      movl  %esp, %ebp
      pushl %esi
      movl  12(%ebp), %esi
      movl  8(%ebp), %ecx
      pushl %ebx
      xorl  %ebx, %ebx
      testl %esi, %esi
      je    .L3
.L7:  xorl  %edx, %edx
.L4:  addl  (%ecx,%edx,4), %eax
      incl  %edx
      cmpl  $20, %edx
      jne   .L4
      incl  %ebx
      addl  $80, %ecx
      cmpl  %ebx, %esi
      ja    .L7
.L3:  popl  %ebx
      popl  %esi
      leave
      ret
```

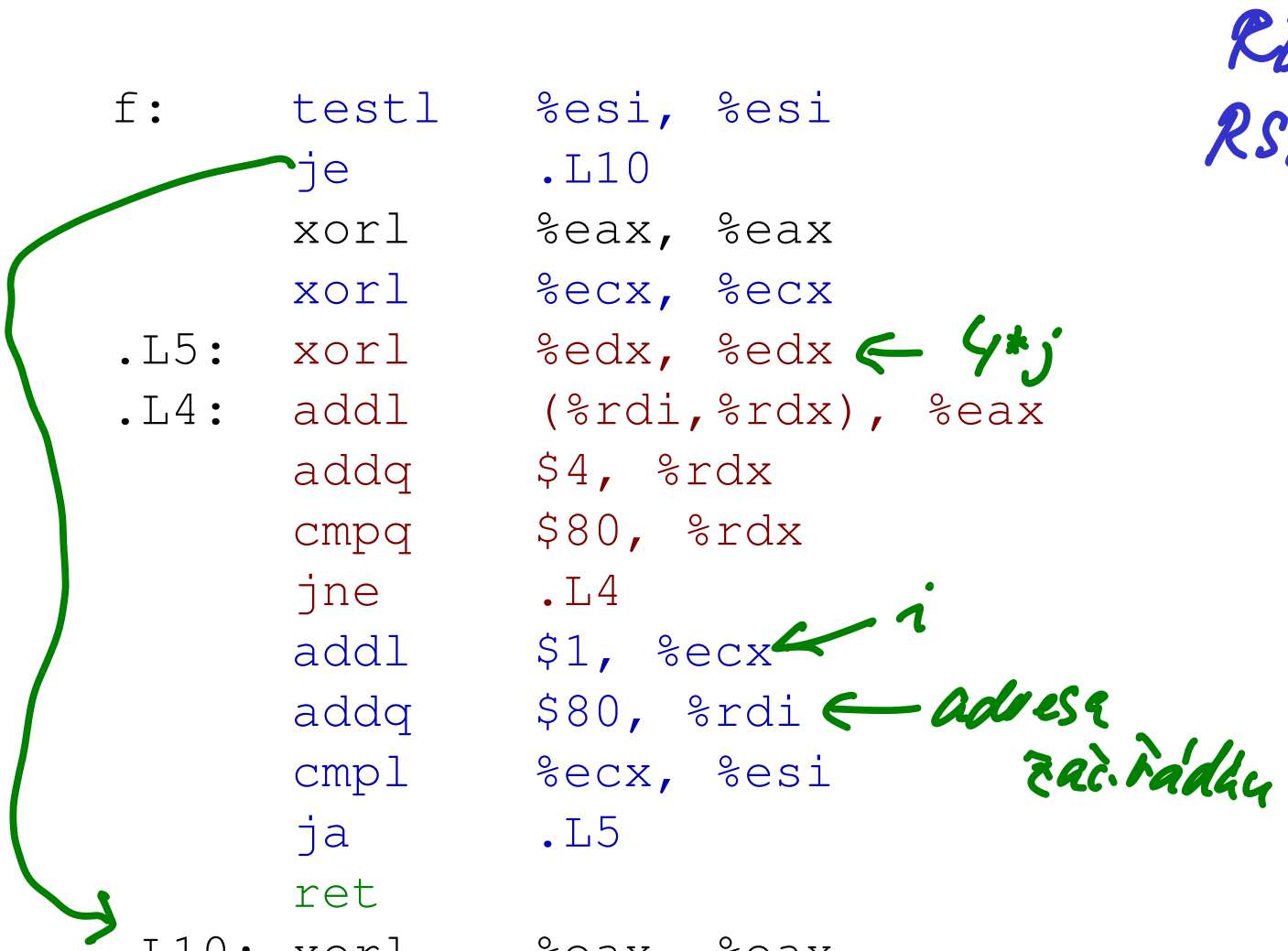


32-bitový assembler: `-fomit-frame-pointer`

nepoužívej EBP, nemusíš ho ukládat

```
f:    pushl %esi
      xorl  %eax, %eax
      pushl %ebx
      movl  16(%esp), %esi
      xorl  %ebx, %ebx
      movl  12(%esp), %ecx
      testl %esi, %esi
      je    .L3
.L7:  xorl  %edx, %edx
.L4:  addl  (%ecx,%edx,4), %eax
      incl  %edx
      cmpl  $20, %edx
      jne   .L4
      incl  %ebx
      addl  $80, %ecx
      cmpl  %ebx, %esi
      ja    .L7
.L3:  popl  %ebx
      popl  %esi
      ret
```

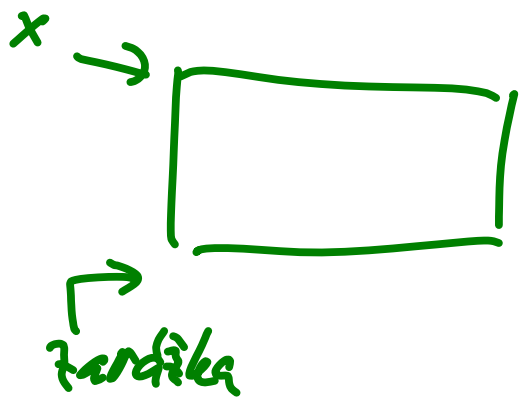
64-bitový assembler: `-march=k8 -m64`




```
f:      testl   %esi, %esi
        je      .L10
        xorl    %eax, %eax
        xorl    %ecx, %ecx
.L5:    xorl    %edx, %edx
.L4:    addl    (%rdi,%rdx), %eax
        addq    $4, %rdx
        cmpq    $80, %rdx
        jne     .L4
        addl    $1, %ecx
        addq    $80, %rdi
        cmpl    %ecx, %esi
        ja      .L5
        ret
.L10:   xorl    %eax, %eax
        ret
```

AMD64, novější GCC: `-march=core2 -m64`

```
f:    testl   %esi, %esi
      je      .L5
      leal    -1(%rsi), %eax
      leaq    (%rax,%rax,4), %rax
      salq    $4, %rax
      leaq    80(%rdi,%rax), %rcx
      xorl    %eax, %eax
.L3:  xorl    %edx, %edx
.L4:  addl    (%rdi,%rdx), %eax
      addq    $4, %rdx
      cmpq    $80, %rdx
      jne     .L4
      addq    $80, %rdi
      cmpq    %rcx, %rdi
      jne     .L3
      ret
.L5:  xorl    %eax, %eax
      ret
```



Rozbalená vnitřní smyčka: `-funroll-loops`

```
.L5:  addl   (%rdi), %eax
      addl   4(%rdi), %eax
      movl   $8, %edx
      addl   (%rdi,%rdx), %eax
      addq   $4, %rdx
      addl   (%rdi,%rdx), %eax
      addq   $4, %rdx
.L4:  addl   (%rdi,%rdx), %eax
      addl   4(%rdi,%rdx), %eax
      addl   8(%rdi,%rdx), %eax
      addl   12(%rdi,%rdx), %eax
      addl   16(%rdi,%rdx), %eax
      addl   20(%rdi,%rdx), %eax
      addl   24(%rdi,%rdx), %eax
      addl   28(%rdi,%rdx), %eax
      addq   $32, %rdx
      cmpq   $80, %rdx
      jne    .L4
```

4 iterace

8 iterací najednou