Tahák na x86(_64) assembler

Datové typy

- Celočíselné typy: **b** (byte, 8bit), **w** (word, 16bit), **l** (long, 32bit), **q** (quadword, 64bit), **dq** (double quadword, 128bit)[1]
- Floating point: half (16bit, z toho 11 mantisa), single (32/24), double (64/53), extended (80/64).
- Referencovat lze i nižší části registrů. Například registru `%rax` je dolních 8, 16 a 32 bitů `%al`, `%ax` a `%eax`. U nových registrů `%r8` až `%r15` pak přidáním suffixů `b`, `w`, `d`pro dolních 8, 16 a 32 bitů, takže například `%r8d` je dolních 32 bitů z `%r8`. Dále lze adresovat horních 8 bitů 16-bitových registrů pomocí suffixu `h`, tedy `%ah` je horní polovina `%ax`.

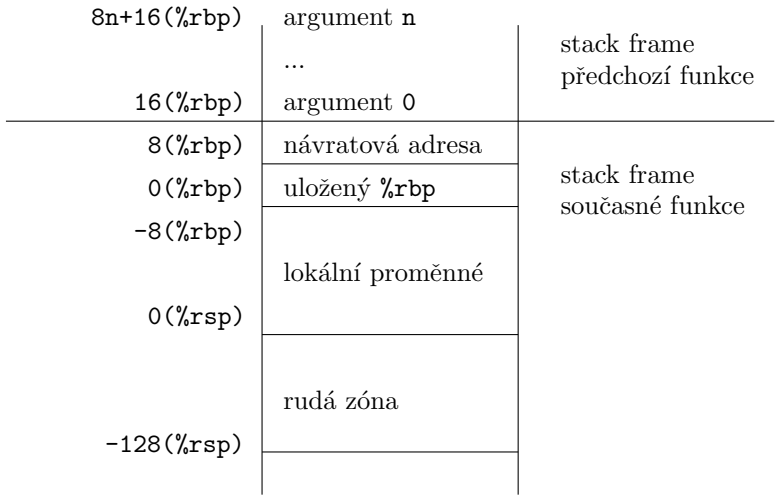
Syntax A&T

```
addq $1, %rbx
```

- Celočíselné instrukce mají suffix z `b w l q` nebo `dq` podle velikosti operadů.
- Cíl operace je vždy poslední argument (add a spol.), nebo se vůbec nepíše (push).
- Typy operandů:
  - `$`*číslo* – literál (konstanta zakódovaná v instrukci)
  - `%`*registr* – obsah registru
  - *adresa* – hodnota v paměti na *adrese*(ta je literál)
  - *offset*`(%`*reg1*`,%`*reg2*`,`*násobitel*`)` – operand se nachází v paměti, na adrese *offset* + `%`*reg1* + *násobitel* * `%`*reg2*
  - implicitní – nepíše se, instrukce typu `push`, registry často `%rsp`, `%rip`, `%rax`

Volací konvence

Stack frame funkce vypadá na AMD64 zhruba následovně:



[1] Délka instrukce na x86 je omezena na 15 bytů, takže neexistují instrukce, které by měly konstantní argumenty, který je delší. Přenos takových konstant tedy musí jít po částech nebo přes paměť.

Rudá zóna existuje pouze na AMD64 a je to místo pro lokální proměnné, které je scratch pro volané funkce.

- Na i386 se všechny argumenty předávají na zásobníku, AMD64 předává část argumentů v registrech.
- Všechny registry, ve kterých se předávají argumenty nebo výsledky jsou scratch.
- Pokud má funkce proměnný počet argumentů (nebo se to o ní neví), předává se v `%al` horní mez na počet použitých xmm registrů.
- Následující tabulka určuje využití registrů. Registry označené (f)arg*i* nesou *i*-tý (floatový) argument, (f/x)res*i* předává (floatový, extended floatový) výsledek.

`%rax`	varg/res1	`%r8`	arg5	`%xmm0`	farg1/fres1	`%xmm8-15`	scratch
`%rbx`	save	`%r9`	arg6	`%xmm1`	farg2/fres2	`%st0`	xres1
`%rcx`	arg4	`%r10`	scratch	`%xmm2`	farg3	`%st1`	xres2
`%rdx`	arg3/res2	`%r11`	scratch	`%xmm3`	farg4	`%sT2-7`	scratch
`%rsi`	arg2	`%r12`	save	`%xmm4`	farg5		
`%rdi`	arg1	`%r13`	save	`%xmm5`	farg6		
`%rbp`	save	`%r14`	save	`%xmm6`	farg7		
`%rsp`	save	`%r15`	save	`%xmm7`	farg8		

objdump a sekce binárky

- `objdump -d` *program* – disassembluje kód programu
- `objdump -s` *program* – vypíše sekce programu hexa (užitečné pro vyčtení řetězců)
- `objdump -h` *program* – zobrazí všechny sekce programu

Některé význačné sekce:

- `.text` – instrukce programu
- `.rodata`, `.data` – statická data jako například řetězcové konstanty
- `.bss` – nezinicializovaná statická data
- `.init`, `.fini` – inicializace a úklid programu; typicky zaplní překladač/linker
- `.ctors`, `.dtors` – konstruktory a destruktory jazyka C (toto není to samé jako u C++ objektů!)
- `.plt` – pařezy pro napojení funkcí z dynamicky linkovaných knihoven

Instrukce jsou typicky následovány velikostí operace (pokud to dává smysl), instrukci mov tedy můžete vidět jako `movb`, `movw` atp.

Přesun dat

- `mov` *src*, *dst* – přesune data stejné velikosti
- `movdqa/movdqu` *src, dst – přesune double-quadword data (zarovnaná, nezarovnaná)*
- `movs`*xy* *src*, *dst* – přesune do většího cíle a rozšíří znaménkově[2]
- `movz`*xy* *src*, *dst* – přesune do většího cíle a rozšíří nulami[2]
- `push` *src* – uloží zdroj na zásobník a sníží stack pointer
- `pop` *dst* – vybere ze zásobníku a zvýší stack pointer
- `xchg` *dst*, *dst* – prohodí obsah svých argumentů
- `cmov`*cc* *src*, *dst* – přesune data, pokud je splněna podmínka *cc*(stejný kód jako podmíněné skoky dále)
- `set`*cc* *dst* – nastaví *dst*na 1 nebo 0, podle podmínky *cc*

Aritmetika

U aritmetických operací se prefixem *i* před instrukcí značí *signed* varianta. Notací `%rdx:%rax` značíme konkatenaci obou registrů.

`add` *src*, *dst*	*dst* `+=` *src*
`sub` *src*, *dst*	*dst* `-=` *src*
`inc` *dst* , `dec` *dst*	dst `+=` *1*, *dst* `-=` *1*
`neg` *dst* , `not` *dst*	aritmetická (bitová) negace
`and/or/xor` *src*, *dst*	logický and/or/xor
`sal/sar` *src*, *dst*	*dst* <<= *src*, *dst* >>= *src* (aritmeticky)
`shl/shr` *src*, *dst*	*dst* <<= *src*, *dst* >>= *src* (logicky)
`mul, imul` *src*, *dst*	*dst*`*=`*src*
`lea` *src*, *dst*	do *dst* nahraje adresu, kterou vyjadřuje *src*
`divb, idivb` *dělitel*	(`%al`, `%ah`) := (`%ax`/*dělitel*, *zbytek*)
`divw, idivw` *dělitel*	(`%ax`, `%dx`) := (`%dx:%ax`/*dělitel*, *zbytek*)
`divl, idivl` *dělitel*	(`%eax`, `%edx`) := (`%edx:%eax`/*dělitel*, *zbytek*)
`divq, idivq` *dělitel*	(`%rax`, `%rdx`) := (`%rdx:%rax`/*dělitel*, *zbytek*)

Porovnání

- `cmp` *src*, *vdst* – nastaví FLAGS podle výsledku operace *vdst* - *src*, do vdst ale nezapíše!
- `test` *src*, *vdst* – nastaví FLAGS podle výsledku operace *vdst* & *src*, do vdst ale nezapíše!

Skoky

Až na druhý zmíněný jmp jsou všechny instrukce skoku povoleny pouze na předem danou adresu adresu, zde značíme jako *label*.

- `jmp` *label* – skoč na návěstí
- `jmp` *op* – skoč na adresu udanou operandem (zde lze použít normální adresování)

[2]Například `movsbl` (`movzbl`) lze číst jako *„move sign(zero)-extended byte to long“*

Dále následuje tabulka podmíněných skoků a jejich významu. Každá instrukce bere právě jeden argument a to cíl skoku jako pevnou adresu. ZF, SF, OF a CF jsou zero, sign, overflow a carry flag.

				Jump if	`cmp` *src*, *vdst*	**EFLAGS**
	`jz`	`je`		zero/equal	*src* = *vdst*	`ZF`
	`jnz`	`jne`		not zero/not equal	*src* ≠ *vdst*	`!ZF`
	`js`			signed		`SF`
	`jns`			not signed		`!SF`
signed	`jg`	`jnle`		greater/not less or equal	*src* < *vdst*	
signed	`jge`	`jnl`		greater or equal/not less	*src* ≤ *vdst*	
signed	`jl`	`jnge`		less/not greater or equal	*src* > *vdst*	
signed	`jle`	`jng`		less or equal/not greater	*src* ≥ *vdst*	
unsigned	`ja`	`jnbe`		above/not below or equal	*src* < *vdst*	`!CF & !ZF`
unsigned	`jae`	`jnb`	`jnc`	above or equal/not below	*src* ≤ *vdst*	`!CF`
unsigned	`jb`	`jnae`	`jc`	below/not above or equal	*src* > *vdst*	`CF`
unsigned	`jbe`	`jna`		below or equal/not above	*src* ≥ *vdst*	`CF|ZF`

Volání funkcí

- `call` *label* – zavolá funkci a uloží adresu následující instrukce na zásobník
- `ret` – opak `call`, odebere adresu ze zásobníku a skočí na ni
- `leave` – „epilog“ – ekvivalent `movq %rbp, %rsp; popq %rbp`

Standardní prolog	**Standardní epilog**
`pushq %rbp`	`movq %rbp, %rsp`
`movq %rsp, %rbp`	`popq %rbp`
`subq N,%rsp`	`ret`

Literatura

AMD, *AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions & Volume 4: 128-bit and 256-bit Media Insructions.*
`http://developer.amd.com/documentation/guides/`

SCO, *System V Application Binary Interface*, verze 4.1.
`http://www.sco.com/developers/devspecs/gabi41.ps`.

Matz, Michael, Hubička, Jan, Jaeger, Andreas, Mitchell, Mark. *System V Application Binary Interface, AMD64 Architecture Processor Supplement.*
`http://www.x86-64.org/documentation/abi-0.99.pdf`

Mareš, Martin. *Programování s ohledem na hardware.*
`http://mj.ucw.cz/papers/hwopt.pdf`