

Operační systémy

Instrukční sady dalších procesorů

Petr Krajča



Katedra informatiky
Univerzita Palackého v Olomouci

- kód v assembleru jde zapsat více způsoby
- dosud používaná syntaxe assembleru se označuje jako **Intel**
- často se používá alternativní syntaxe **AT&T**
- operace zapisované ve tvaru
`<jmeno><velikost> zdroj, cil`
- `<jmeno>` označuje název operace `mov`, `add`, `cmp`
- `<velikost>` je písmeno `b`, `w`, `l` nebo `q` udává velikost operandů (1, 2, 4 nebo 8 B)
- registry se zapisují ve tvaru `%reg` (např. `%eax`)
- konstanty začínají znakem `$` (např. `$100`)
- adresace paměti ve tvaru `disp(base,index,scale)` (např. `-10(%ecx,%ebx,2)` \rightsquigarrow `[ecx + 2 * ebx - 10]`)

AT&T

```
pushw %ax
```

```
movl $100, %eax
```

```
addl %ebx, %eax
```

```
subl (%eax), %ecx
```

```
subl (%eax,%ebx), %ecx
```

```
subl -10(%eax), %ecx
```

```
subl -10(%eax,%ebx,2), %ecx
```

```
andw $42, -16(%eax)
```



Intel

```
push ax
```

```
mov eax, 100
```

```
add eax, ebx
```

```
sub ecx, [eax]
```

```
sub ecx, [eax + ebx]
```

```
sub ecx, [eax - 10]
```

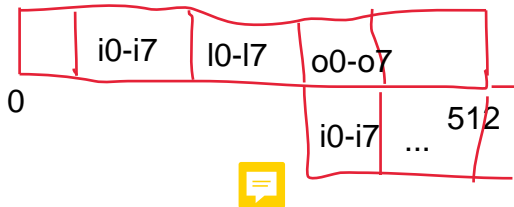
```
sub ecx, [eax + ebx * 2 - 10]
```

```
and word ptr [eax - 16], 42
```




```
pushl    %ebp
movl     %esp, %ebp
andl     $-16, %esp
subl     $16, %esp
movl     $121, 4(%esp)
movl     $.LC0, (%esp)
call     printf
xorl     %eax, %eax
leave
ret
```

- rodina procesorů (některé dostupné pod GPL)
- každá instrukce zabírá v paměti 4B
- snaha eliminovat množství instrukcí
- operace běžně se třemi operandy
- velké množství registrů (řádově stovky), běžně dostupných 32 registrů
- globální registry g0 – g7 (g0 je vždy nula)
- registrové okno – 24 registrů
 - i0 – i7 – argumenty předané funkci
 - l0 – l7 – lokální proměnné
 - o0 – o7 – argumenty předávané další funkci
- speciální využití některých registrů
 - fp – frame pointer (i6)
 - sp – stack pointer (o6)
 - návratová adresa – i7/o7



■ příklady operací




```
add    %i0, 1, %l1      # l1 := i0 + 1
subcc  %i1, %i2, %i3     # i3 := i1 - i2
subcc  %i1, %i2, %g0     # g0 := i1 - i2    (cmp)
or     %g0, 123, %l1     # l1 := g0 / 123   (mov)
```

■ malá velikost instrukce

- operace neumožňují adresovat paměť \Rightarrow specializované operace ld, st
- \Rightarrow load/store architektura
- interně se pracuje s celými registry
- jako konstanty jde běžně používat pouze hodnoty -4096 – 4095
- přiřazení velkých čísel ve dvou krocích

```
sethi  0x226AF3, %l1    # nastaví horní bity
or     %l1, 0x1EF, %l1  # nastaví dolní bity
```

- jednoduché instrukce
- potenciálně rychlejší zpracování
- skoky (podmíněné i nepodmíněné) se neprovádí okamžitě
- k optimálnímu využití pipeline se přidává *delay slot*
- ještě je zpracována následující instrukce za operací skoku 
- možnost nastavit *annul bit*, operace v delay slotu se provede právě tehdy, pokud se provede i skok





```
cmp    %l1, %l2  
bl,a   addr  
mov    %g0, %l3
```

- rodina 32- a 64bitových procesorů typicky využívaná v embedded a přenosných zařízeních
- optimalizace na nízkou spotřebu el. energie a paměti
- není jeden výrobce, licence dalším výrobcům
- základní jádro je licencováno výrobcům k výrobě SoC (Qualcomm Snapdragon, nVidia Tegra, Apple A4-A14, M1, Samsung Exynos...)
- několik variant instrukční sady—v současné době používané ještě ARMv5, ARMv6, ale především ARMv7 (32 bitů) a ARMv8 (64 bitů)
- děleny ještě podle určení A (aplikační), R (real-time), M (mikrokontrolery)
- architektura *big.LITTLE* – kombinace pomalejších a úspornějších jader (LITTLE) s výkonnými (big), která jsou využívána podle aktuálního zatížení systému



- podpora několika různých typů instrukčních sad (+ rozšíření dle modelu, např. specializované instrukce pro kryptografii)
- přímá podpora až 16 koprocetorů
- load/store architektura

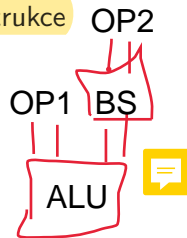
Registry

- 32 obecně použitelných registrů 
- z toho jen 16 je v daný okamžik použitelných (R0 – R15)  
- R13 (SP) – Stack Pointer, R14 (LR) – Link Register, R15 (PC) – Program Counter
- registry R13 a R14 přepínány podle aktuálního režimu procesoru (jaderný režim, obsluha přerušení, atp.), v případě rychlých přerušení přepnuty R8 až R14
- stavový registr APSR pro případné uložení příznaků proběhlé operace 

- všechny instrukce o velikosti 32 bitů
- obvykle 2-3 operandy, příznaky nastavují jen programátorem určené instrukce
- možnost podmíněného vykonávání instrukcí

```
CMP    R0, 0      ; porovnej R0 s 0
RSBLT  R0, R0, 0   ; pokud R0 < 0, pak R0 := 0 - R0
```

- barrel shifter umístěný před ALU a druhý argument
- umožňuje kombinovat operaci s operací bitových posunů a rotací
- v případě přímých hodnot se používá kombinace 8 bitů pro konstantu a 4 bity pro operaci ROR
- 32bitové konstanty přiřazovány ve dvou operacích (horních/dolních šestnáct bitů)
- operace load/store umožňují měnit registr s indexem
- při volání je návratová adresa uložena do registru R14 (LR)
- argumenty jsou předávány přes registry (první čtyři) a zásobník



Příklad instrukce v sadě ARM



Operace typu ADD Rd, Rn, imm

podmienka				typ inš.		použ. oper.				zvyšok sú zakódované operandy																					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
cond				00		1	OpCode				S	Rn				Rd				rotate				imm8							

- cond – podmínka (AL, EQ, NE, LT, GT, ...)
- 00 – typ instrukce
- 1 – použije se konstanta
- OpCode – použitá operace ADD, SUB, RSB, MOV, CMP, ...
- Rn, Rd – registry
- imm8 – přímá hodnota
- rotate – aplikuje ROR(imm8, $2 \times \text{rotate}$)



Operace typu ADD Rd, Rn, Rm

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
cond				00		0	OpCode				S	Rn				Rd				shift								Rm			

- shift – dále zakódováno posunutí Rm o pevný počet bitů nebo o hodnotu danou registrem
- podporovány logické i aritmetické posuny, rotace vpravo

00000000 <fact>:

```
0:      e3500000      cmp      r0, #0
4:      0a000005      beq      20 <fact+0x20>
8:      e92d4010      push     {r4, lr}
c:      e1a04000      mov      r4, r0
10:     e2400001      sub      r0, r0, #1
14:     ebfffffe      bl       0 <fact>
18:     e0000094      mul      r0, r4, r0
1c:     e8bd8010      pop      {r4, pc}
20:     e3a00001      mov      r0, #1
24:     e12fff1e      bx       lr
```


Kódování Thumb

- alternativní způsob kódování instrukcí
- zahuštění kódu (použití hlavně v mikrokontrolerech)
- velikost instrukce 16 bitů nebo 32 bitů
- 16 bitová varianta
 - menší počet operandů (podobná ISA x86)
 - možnost přistupovat pouze k části registrů
 - bez možnosti podmíněného provádění instrukcí jednotlivých instrukcí
 - instrukce IT (if-then) supluje předchozí omezení
- 32 bitové instrukce umožňují přístup k dalším vlastnostem ISA
- přepínání mezi kódováním Thumb a ARM přes instrukce (bxx, blxx)

Další instrukční sady

- Jazelle – spouštění Java bytecode



- 64bitový nástupce architektury ARMv7 
- instrukce velikosti 32 bitů
- módy pro zpětnou kompatibilitu
- 31 64bitových registrů (X0 až X30), jde použít i pouze spodní 32bitové poloviny (W0 až W30)
- registr X31/W31 zero registr (podobně jako u SPARC)
- X30 odpovídá LR, samostatné registry SP a PC, PSTATE (stavový registr)
- v 64bitovém režimu některé vlastnosti zrušeny (podmíněné provádění instrukcí) nebo upraveny (bitové posuny u konstant)
- argumenty předávány přes registry (X0 až X7) a další přes zásobník

RISC: Reduced Instruction Set Computer

- zjednodušený návrh a implementace CPU
- rychlejší běh, určitá omezení

CISC: Complete (Complex) Instruction Set Computer

- poskytují operace velice blízké vyšším progr. jazykům
- snadné pro ruční programování
- náročné na implementaci CPU (+ již nepoužívané instrukce v ISA)

Reálně...

- procesory typu CISC provádí rozklad operací na mikrooperace \Rightarrow vnitřně RISC
- další úroveň abstrakce
- vnitřně dochází ještě k dalším úpravám kódu, např. přejmenovávání registrů
- out-of-order execution \Rightarrow rozdělení (mikro)operací jednotlivým jednotkám \Rightarrow paralelismus

- plánování OoO komplikuje návrh CPU



VLIW: very large instruction word



- snaha využít několik funkčních jednotek
- jedna instrukce může obsahovat několik operací
- \Rightarrow souběžné zpracování
- spolupráce s překladačem \Rightarrow „CPU nemusí hádat, jak poběží program“
- složitější návrh dekódovací jednotky
- Intel Itanium, Digital Signal Processors (DSP)
- v případě AMD64 rozšíření FMA (fused multiply-add)

