**Logické instrukce**

Při studiu této kapitoly předpokládáme znalost kapitol přesuny a aritmetické instrukce.

Logické instrukce realizují nám již známé logické funkce AND, OR, XOR.

Problematiku si ukážeme na instrukci and , ostatní se dělají analogicky. Adresové módy jsou stejné jako u součtu – add , a mov , takže při výkladu se omezíme jenom na použití registrů. Na straně 330 datasheetu se pak podívejte na další možnosti instrukce AND.

mov #0xbc3a , W0

mov #0xe038 , W1

and W0, W1, W2

Instrukce udělá logický součin registrů W0 a W1 a výsledek uloží do registru W2. Jediná důležitá věc, kterou si musíme pamatovat, je tato**: u logických instrukcí se dělá každý bit zvlášť**

**Logický součin - AND**



Výsledek horní operace je samozřejmě 0xA038

**Logický součet- OR**

mov #0xbc3a , W0

mov #0xe038 , W1

ior W0, W1, W2 ; logicky soucet se nazyva v nasem assembleru ior



Výsledek je samozřejmě 0xFC3A

**Výhradně-nebo Exclusive-OR**

mov #0xbc3a , W0

mov #0xe038 , W1

xor W0, W1, W2



Výsledek je 0x5f23

**Maskování**

Logické operace se používají k tzv. maskování. To užíváme v případě, kdy potřebujeme vnutit do registru na některé bity určitou hodnotu. Typickým příkladem je čtení dat z portu. Např. port A má pouze 5 bitů. RA0 – RA4 . Při čtení pomocí instrukce

mov PORTA, W6

ale do registru W6 uložíme všech 16 bitů. U bitů 15 – 5 nikdo neví, jakou mají hodnotu. Musíme proto do registru na pozici 15 – 5 bit vnutit tu hodnotu, kterou si my přejeme. To provedeme pomocí maskování

při použití instrukce and vnutíme do určitých pozic 0 ( cokoli AND 0 = 0 )

při použití instrukce ior vnutíme do určitých pozic 1 ( cokoli OR 1 = 1 )

vidíme to hezky ve file log01.s

instrukce xor zajistí negaci těch bitů, u kterých má druhý operand hodnotu 1

cokoli XOR 0 = cokoli

cokoli XOR 1 = negace cokoli

opět se podívejte do log.s

**Posuvy**

Instrukce posuvů posouvají bity doleva nebo doprava

**Posuv doleva**

SL

Instrukce je v datasheetu na straně 335 , číslo instrukce je 73. Má opět několik variant, vysvětlíme si jenom základní variantu.

**SL zdroj , cíl**

Vezme bity ze zdroj, posune o jeden bit doleva, výsledek uloží do cíl . Je možno použít veškeré adresovací módy. Do 0. bitu se uloží 0 . Hodnota, která byla v 15. bitu, je ztracena .



Posuv doleva zdvojnásobí číslo v registru

**Posuv doprava**

má dvě varianty – logický a aritmetický posuv

**Logický posuv doprava LSR**

**LSR zdroj, cil**



Posune bity v registru doprava. Hodnota, která byla v 0. bitu, mizí. Do 15. bitu se vloží 0

Vidíme, že číslo v registru se vydělí 2

**Aritmetický posuv doprava ASR**

**ASR zdroj , cil**

Tento posuv zachovává při posuvu znaménko.

Nejlépe vše vysvětlíme na příkladu:

číslo 0xc000 je desítkově -16384 ( minus, záporné. Převeďte si to do binární soustavy, udělejte druhý doplněk, zkuste si to sami)

Pokud uděláme LSR , tedy logický shift, dostaneme 0x6000 . Tohle číslo je ale kladné, a přitom by to mělo být ( - 16384) / 2 = -8192 . Toto číslo je ale hexadecimálně 0xe000 . ( zkuste sami !). Vidíme, že při posuvu je nutno „zdublovat“ - zdvojit - znaménkový bit, abychom zachovali znaménko čísla. A to dělá aritmetický posuv doprava .



**Rotace**

Kromě posuvů existují ještě rotace - RLC, RLNC, RRC, RRNC . Tyto instrukce se chovají podobně jako posuny, ale navíc bit, který u posunů „vypadl“ , vrací na druhou stranu registru. Doporučuji vaší pozornosti file assembler.pdf , kde jsou jednotlivé instrukce detaelně popsány. RLC je na straně 373 , u instrukce je vždy popis s příklady.