Кратък справочник за избрани асемблерни команди за 32-разрядната архитектура x86

Кирил Иванов

Май 2017 г.

Този кратък справочник е предназначен за въведение в практическото програмиране на асемблерен език потребителско (при ограничени привилегии на изпълняваната програма). Представени са ограничен брой команди и, при това, без детайли. Всички команди за целочисления блок на процесора започват с буква, различна от f, и всички команди за блока за работа с кодове с плаваща запетая започват с буква f.

Списък по азбучен ред

adc op1, op2 – записва в op1 сумата на op1, op2 и флага CF

add op1, op2 – записва в op1 сумата на op1 и op2

and op1, op2 – записва в op1 поразрядното И (AND; поразрядната конюнкция) на op1 и op2

bsf op1, op2 – търси *отдясно наляво* единичен разряд в op2 и може да завърши по два начина: Първо, ако op2 е нула, записва единица във флага ZF и op1 е неопределено. Второ, ако в op2 има единичен разряд, номера на най-дясната (най-младшата) единица (броенето започва от нула за младшия разряд) записва в op1 и нулира флага ZF

bsr op1, op2 – търси *отляво надясно* единичен разряд в op2 и може да завърши по два начина: Първо, ако op2 е нула, записва единица във флага ZF и op1 е неопределено. Второ, ако в op2 има единичен разряд, номера на най-лявата (най-старшата) единица (броенето започва от нула за младшия разряд) записва в op1 и нулира флага ZF

bswap op – op трябва да бъде 32-разряден; пренарежда байтовете на op в точно обратен ред (например $12345678_{(16)}$ става $78563412_{(16)}$)

bt op1, op2 – записва във флага CF разряда от op1, който има номер op2 (номерирането започва от 0 за младшия разряд)

btc op1, op2 – записва във флага CF разряда от op1, който има номер op2 (номерирането започва от 0 за младшия разряд), и инвертира записания разряд

btr op1, op2 – записва във флага CF разряда от op1, който има номер op2 (номерирането започва от 0 за младшия разряд), и нулира в op1 записания разряд

bts op1, op2 – записва във флага CF разряда от op1, който има номер op2 (номерирането започва от 0 за младшия разряд), и после записва 1 в същия разряд

call adr – записва в стека адрес за връщане от подпрограма (записва адреса на командата, намираща се точно след call adr) и предава управлението към адреса adr; записваният адрес може да бъде в различни формати, включително пълен логически (обикновено размерът му се определя автоматично от компилатора, но може да бъде явно указан)

cbw – преобразува 8-разрядния допълнителен код в регистъра AL в 16-разряден допълнителен код в регистъра AX

cdq – преобразува допълнителния код от регистъра EAX в 64-разряден допълнителен код с младша половина в регистъра EAX и старша половина в регистъра EDX

clc – нулира флага CF

cld – нулира флага DF

cmc – инвертира флага CF

cmp op1, op2 – изчислява разликата $op1 - op2$ без да съхранява никъде резултата, обаче модифицира флаговете, точно както би ги променила командата sub op1, op2 (обикновено cmp предшествува команди за условен преход по условие някаква релация между op1 и op2)

cmpsb – изчислява разликата $\text{byte ptr DS:[ESI]} - \text{byte ptr ES:[EDI]}$ без да съхранява получения резултат, но като модифицира флаговете, точно както биха ги променили командите cmp или sub със същите умаляемо и умалител, и след това прибавя към ESI и към EDI или $+1$, когато флагът $DF = 0$, или -1, когато флагът $DF = 1$; може да се зацикля чрез префиксите rep, repz, repe, repnz и repne

cmpsd – изчислява разликата $\text{dword ptr DS:[ESI]} - \text{dword ptr ES:[EDI]}$ без да съхранява получения резултат, но като модифицира флаговете, точно както биха ги променили командите cmp или sub със същите умаляемо и умалител, и след това прибавя към ESI и към EDI или $+4$, когато флагът $DF = 0$, или -4, когато флагът $DF = 1$; може да се зацикля чрез префиксите rep, repz, repe, repnz и repne

cmpsw – изчислява разликата $\text{word ptr DS:[ESI]} - \text{word ptr ES:[EDI]}$ без да съхранява получения резултат, но като модифицира флаговете, точно както биха ги променили командите cmp или sub със същите умаляемо и умалител, и след това прибавя към ESI и към EDI или $+2$, когато флагът $DF = 0$, или -2, когато флагът $DF = 1$; може да се зацикля чрез префиксите rep, repz, repe, repnz и repne

cwd – преобразува допълнителния код от регистъра AX в 32-разряден допълнителен код с младша половина в регистъра AX и старша половина в регистъра DX

cwde – преобразува допълнителния код от регистъра AX в 32-разряден допълнителен код в регистъра EAX

dec op – намалява с единица стойността на op; *не променя* флагове

fadd – еквивалентна на ***faddp st(1), st(0)***

fadd op – замества върха st(0) на регистровия стек със сумата st(0)+op, където op е в код в с плаваща запетая в регистровия стек или в паметта (тогава разрядността на op се определя от типа му и може да бъде 4, 8 или 10 байта)

fadd st(i), st(j) – замества st(i) със сумата st(i)+st(j); операндите са регистри за данни в блока за изчисления с плаваща запетая; единият от индексите i и j трябва да бъде 0

faddp st(i), st(0) – замества st(i) със сумата st(i)+st(0) и премахва върха на регистровия стек в устройството за изчисления с плаваща запетая

fcom – изчислява разликата на st(0) и st(1) без да я съхранява, но модифицира флагове в думата на състоянието

fcom op – изчислява разликата на st(0) и op без да я съхранява, но модифицира флагове в думата на състоянието; op е в код с плаваща запетая и или е регистър st(i), или е в паметта (с разрядност 4, 8 или 10, подразбирана от типа му)

fcomp – изчислява разликата на st(0) и st(1) без да я съхранява, но модифицира флагове в думата на състоянието, а после премахва върха st(0) на регистровия стек

fcomp op – изчислява разликата на st(0) и op без да я съхранява, но модифицира флагове в думата на състоянието, а после премахва върха st(0) на регистровия стек; op е в код с плаваща запетая и или е регистър st(i), или е в паметта (с разрядност 4, 8 или 10, подразбирана от типа му)

fadd st(i), st(j) – замества st(i) със сумата st(i)+st(j); операндите са регистри от регистровия стек в блока за изчисления с плаваща запетая; единият от индексите i и j трябва да бъде 0

fdecstp – премахва върха st(0) на регистровия стек в блока за изчисления с плаваща запетая

fiadd op – замества върха st(0) на регистровия стек със сумата st(0)+op, където op е в допълнителен код в паметта (разрядността на op се определя от типа му и може да бъде 2, 4 или 8 байта)

ficom op – изчислява разликата на st(0) и op без да я съхранява, но модифицира флагове в думата на състоянието; op е в допълнителен код в паметта (с разрядност 2, 4 или 8, подразбирана от типа му)

ficomp op – изчислява разликата на st(0) и op без да я съхранява, но модифицира флагове в думата на състоянието, а после премахва върха st(0) на стека; op е в допълнителен код в паметта (с разрядност 2, 4 или 8, подразбирана от типа му)

fild op – прочита допълнителен код с плаваща запетая op, преобразува го и го включва като нов връх на регистровия стек в блока за изчисления с плаваща запетая (разрядността на op се определя от типа му и може да бъде 2, 4 или 8 байта)

fist op – записва в op във вид на допълнителен код върха на регистровия стек в блока за изчисления с плаваща запетая (разрядността на op се определя от типа му и може да бъде 2, 4 или 8 байта)

fistp op – записва същото, както fist op, а после премахва върха на регистровия стек

fld op – прочита код с плаваща запетая op, преобразува го и го включва като нов връх на регистровия стек в блока за изчисления с плаваща запетая (разрядността на op се определя от типа му и може да бъде 4, 8 или 10 байта)

fsqrt – замества върха на регистровия стек в блока за изчисления с плаваща запетая с квадратен корен от него

fstsw ax – записва в регистъра AX думата на състоянието на блока за изчисления с плаваща запетая

frndint – замества върха на регистровия стек в блока за изчисления с плаваща запетая с най-близкото до него цяло число

fst op – записва във вид на код с плаваща запетая в op върха на регистровия стек в блока за изчисления с плаваща запетая (разрядността на op се определя от типа му и може да бъде 4, 8 или 10 байта)

fstp op – записва същото, както fst op, а после премахва върха на регистровия стек

imul op1, op2 – записва в op1 произведението на op1 и op2, които са в допълнителен код

imul reg, op, num – записва в регистъра reg произведението на op и числото num; операндите и резултатът са в допълнителен код

inc op – увеличава с единица стойността на op; *не променя* флагове

j... adr – условен преход към адреса adr, като условието е назовано с буквите, заместващи многоточието. Вариантите на условния преход според стойностите на флаговете са следните:

Команди за условен преход според флагове

j... adr – условен преход към адреса adr, като условието е назовано с буквите, заместващи многоточието

Когато в названието на командата присъствува буквата ***n***, тя винаги означава отрицание на условието, назовано с останалите букви.

За назоваването на условието се използват три вида буквосъчетания:

Първо,

за преходи **според стойност на един флаг** (възможен е само за флагове ZF, CF, OF, SF, PF):

jz – преход при ZF=1;

jnz – преход при ZF=0;

jc – преход при CF=1;

jnc – преход при CF=0;

jo – преход при OF=1;

jno – преход при OF=0;

js – преход при SF=1;

jns – преход при SF=0;

jp – преход при PF=1;

jnp – преход при PF=0.

Второ,

за преходи **след сравняване на цели числа в код без знак**. Предполага се, че флаговете са определени от команда ***cmp op1, op2*** или ***sub op1, op2***. Тогава вариантите са:

je – преход при $op1 = op2$;

jne – преход при $op1 \neq op2$;

jb или ***jnae*** – преход при $op1 < op2$;

jbe или ***jna*** – преход при $op1 \leq op2$;

ja или ***jnbe*** – преход при $op1 > op2$;

jae или ***jnb*** – преход при $op1 \geq op2$.

Трето,

за преходи **след сравняване на цели числа в допълнителен код** (т. е. със знак). Предполага се, че флаговете са определени от команда ***cmp op1, op2*** или ***sub op1, op2***. Тогава условните преходи са:

je – преход при $op1 = op2$;

jne – преход при $op1 \neq op2$;

jl или ***jnge*** – преход при $op1 < op2$;

jle или ***jna*** – преход при $op1 \leq op2$;

jg или ***jnle*** – преход при $op1 > op2$;

jge или ***jnl*** – преход при $op1 \geq op2$.

jcxz adr – ако регистъра cx е нула, предава управлението към адреса adr, а иначе не прави преход

jecxz adr – ако регистъра ecx е нула, предава управлението към адреса adr, а иначе не прави преход

jmp adr – безусловно предава управлението към адреса adr (обикновено adr е етикет)

lahf – записва в регистъра AH младшия байт на регистъра с флагове EFLAGS

lea op1, op2 – записва в op1 адреса на op2 (записва в op1 само отместването от пълния логически адрес на op2, след което op1 може да се използва като базов регистър за адресиране на операнда op2 или на съседни с него данни); op1 е 32-разряден регистър; op2 е в оперативната памет

lodsb – записва в регистъра AL един байт от адрес DS:[ESI] и след това прибавя към ESI или $+1$, когато флагът $DF=0$, или -1, когато флагът $DF=1$; може да се зацикля чрез префиксите rep, repz, repe, repnz и repne

lodsd – записва в регистъра EAX четворката байтове, взета от адрес DS:[ESI], и след това прибавя към ESI или $+4$, когато флагът $DF=0$, или -4, когато флагът $DF=1$; може да се зацикля чрез префиксите rep, repz, repe, repnz и repne

lodsw – записва в регистъра AX двойката байтове, взета от адрес DS:[ESI], и след това прибавя към ESI или $+2$, когато флагът $DF=0$, или -2, когато флагът $DF=1$; може да се зацикля чрез префиксите rep, repz, repe, repnz и repne

loop adr – намалява регистъра ECX с 1 да променя флагове и прави преход към адреса adr, точно когато след намаляването $ECX \neq 0$

loope adr – намалява регистъра ECX с 1 без да променя флагове и прави преход към адреса adr, точно когато флагът ZF=1 и след намаляването $ECX \neq 0$

loopne adr – намалява регистъра ECX с 1 без да променя флагове и прави преход към адреса adr, точно когато флагът ZF=0 и след намаляването $ECX \neq 0$

loopz *е еквивалентна на* ***loope***

loopnz *е еквивалентна на* ***loopne***

mov op1, op2 – записва в op1 операнда op2

movsb – копира един байт от адрес DS:[ESI] на адрес ES:[EDI] и след това прибавя към ESI и към EDI или $+1$, когато флагът $DF=0$, или -1, когато флагът $DF=1$; може да се зацикля чрез префиксите rep, repz, repe, repnz и repne

movsd – копира четворка байтове от адрес DS:[ESI] на адрес ES:[EDI] и след това прибавя към ESI и към EDI или $+4$, когато флагът $DF=0$, или -4, когато флагът $DF=1$; може да се зацикля чрез префиксите rep, repz, repe, repnz и repne

movsw – копира двойка байтове от адрес DS:[ESI] на адрес ES:[EDI] и след това прибавя към ESI и към EDI или $+2$, когато флагът $DF=0$, или -2, когато флагът $DF=1$; може да се зацикля чрез префиксите rep, repz, repe, repnz и repne

movsx op1, op2 – записва в op1 операнда op2 и едновременно разширява по-малката разрядност на op2 до по-голямата разрядност на op1; операндите (трябва да) са в допълнителен код

xchg op1, op2 – разменя стойностите на двата операнда

neg op – променя (обръща) знака на op; op е в допълнителен код

nop – не променя нищо, но заема един машинен такт (използва се главно за предизвикване на паузи)

not op – записва в op поразрядното отрицание на op

or op1, op2 – записва в op1 поразрядното Или (OR; поразрядната дизюнкция) на op1 и op2

pop op – премахва върха на хардуерно поддържания стек и го записва в op

popa – възстановява от хардуерно поддържания стек регистрите ax, bx, cx, dx, si, di, bp, sp като премахва от стека съответните стойности

popad – възстановява от хардуерно поддържания стек регистрите eax, ebx, ecx, edx, esi, edi, ebp, esp като премахва от стека съответните стойности

popf – премахва 16-разрядна данна от върха на хардуерно поддържания стек и записва премахнатото в младшите 16 разряда на флаговия регистър EFLAGS (т. е. записва в регистъра FLAGS)

popfd – премахва 32-разрядна данна от върха на хардуерно поддържания стек и записва премахнатото във флаговия регистър EFLAGS

pushf – добавя (съдържанието на) регистъра FLAGS (младшите 16 разряда на флаговия регистър EFLAGS) като нов връх на хардуерно поддържания стек

pushfd – добавя (съдържанието на) регистъра EFLAGS като нов връх на хардуерно поддържания стек

push op – записва op в хардуерно поддържания стек

pusha – записва в хардуерно поддържания стек регистрите ax, bx, cx, dx, si, di, bp, sp

pushad – записва в хардуерно поддържания стек регистрите eax, ebx, ecx, edx, esi, edi, ebp, esp

rep – префикс (не е самостоятелна команда), който може да стои пред така наричаните „низови“ команди и предизвиква тяхното зацикляне: указва, че след изпълнението на командата регистърът ECX се намалява с единица и, когато в него остане нещо различно от нула, командата трябва да се изпълни наново (по подразбиране „низовите“ команди lodsb, lodsw, stosd, movsw и т. н. имат операнди DS:[ESI] и/или ES:[EDI] и модифицират регистрите ESI и EDI в зависимост от флага DF, което позволява да обработват различни данни при поредните изпълнения на командата)

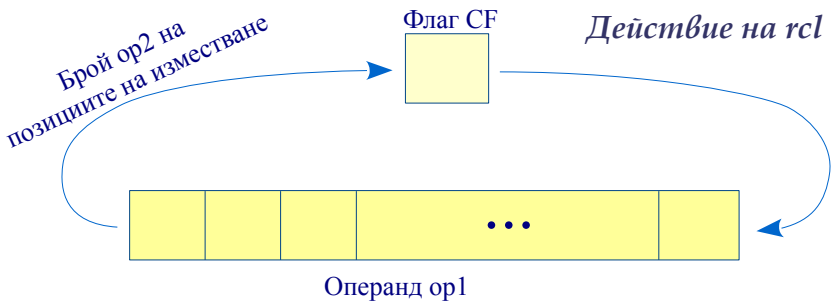
repe – префикс (не е самостоятелна команда), който може да стои пред така наричаните „низови“ команди и предизвиква тяхното зацикляне: указва, че след изпълнението на командата регистърът ECX се намалява с единица и, когато след намаляването $ECX \neq 0$ и едновременно с това флагът $ZF = 1$, командата трябва да се изпълни наново

repne – префикс (не е самостоятелна команда), който може да стои пред така наричаните „низови“ команди и предизвиква тяхното зацикляне: указва, че след изпълнението на командата регистърът ECX се намалява с единица и, когато след намаляването $ECX \neq 0$ и едновременно с това флагът $ZF = 0$, командата трябва да се изпълни наново

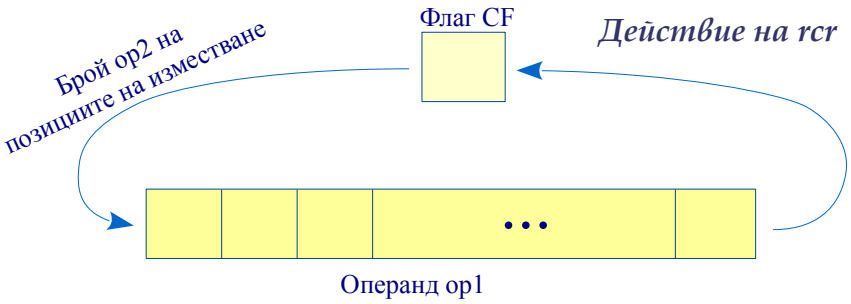
repnz – префикс (не е самостоятелна команда), еквивалентен на ***repne***

repz – префикс (не е самостоятелна команда), еквивалентен на ***repe***

rcl op1, op2 – циклично измества разрядите на op1 наляво през флага CF на op2 позиции; op2 трябва да бъде 8-разряден и е или число, или регистъра CL; rcl работи по схемата:

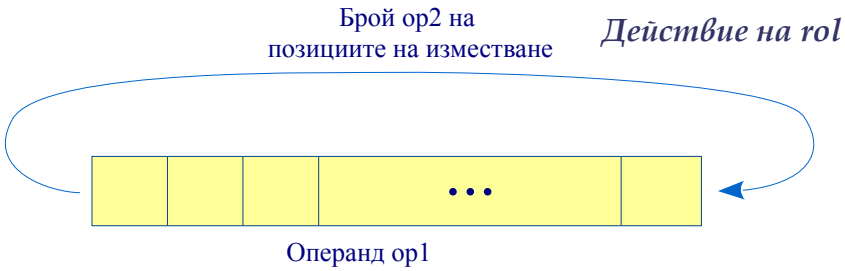


rcr op1, op2 – циклично измества разрядите на op1 надясно през флага CF на op2 позиции; op2 трябва да бъде 8-разряден и е или число, или регистъра CL; rcr работи по схемата:

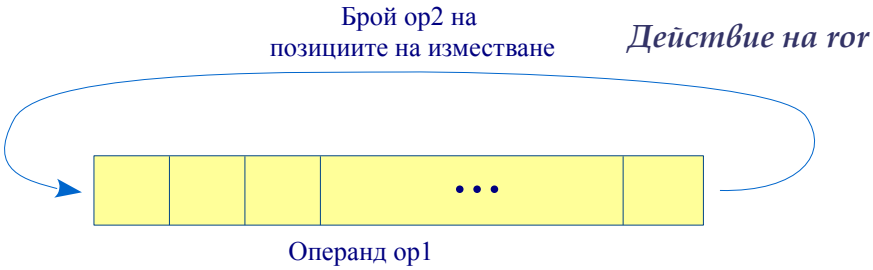


ret num – извлича (и премахва) от стека адрес за връщане от подпрограма и прави преход към него; когато има операнд (което не е задължително) премахва num байта от върха на хардуерно поддържания стек след извличането на адреса

rol op1, op2 – циклично измества разрядите на op1 наляво на op2 позиции; op2 трябва да бъде 8-разряден и е или число, или регистъра CL; rol работи по схемата:



ror op1, op2 – циклично измества разрядите на op1 надясно на op2 позиции; op2 трябва да бъде

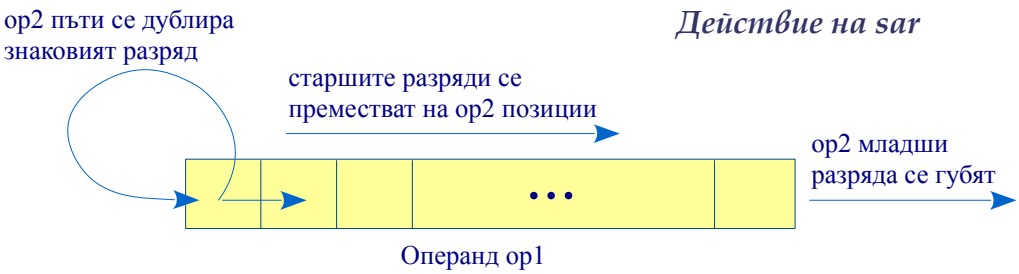


8-разряден и е или число, или регистъра CL; ror работи по схемата:

sahf – записва младшия байт на регистъра с флагове EFLAGS в регистъра AH

sal op1, op2 – еквивалентно на ***shl*** (аритметично изместване наляво; еквивалентно на умножение с 2^{op2} и за код без знак, и за допълнителен код)

sar op1, op2 – аритметично изместване надясно; еквивалентно е на делене с 2^{op2} със закръгляне на резултата надолу до най-близкото цяло число, както за код без знак, така и за допълнителен код; sar се различава от shr по това, че знаковият разряд се дублира (запазва си стойността, а при shr отляво се дописват нули); op2 е 8-разряден и е или число, или регистъра CL; sar работи така:



sbb op1, op2 – записва в op1 разлика с умаляемо op1 и умалител сумата на op2 и флага CF (т. е. записва $op1 - op2 - CF$)

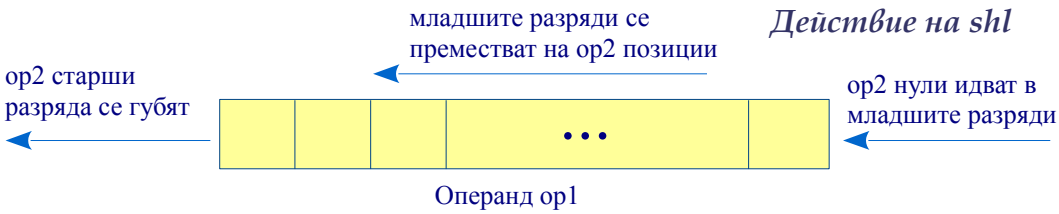
scasb – изчислява разликата AL – byte ptr ES:[EDI] без да съхранява получения резултат, но като модифицира флаговете, точно както биха ги променили командите cmp или sub със същите умаляемо и умалител, и след това прибавя към EDI или $+1$, когато флагът $DF=0$, или -1, когато флагът $DF=1$; може да се зацикля чрез префиксите rep, repz, repe, repnz и repne

scasd – изчислява разликата EAX – dword ptr ES:[EDI] без да съхранява получения резултат, но като модифицира флаговете, точно както биха ги променили командите cmp или sub със същите умаляемо и умалител, и след това прибавя към EDI или $+4$, когато флагът $DF=0$, или -4, когато флагът $DF=1$; може да се зацикля чрез префиксите rep, repz, repe, repnz и repne

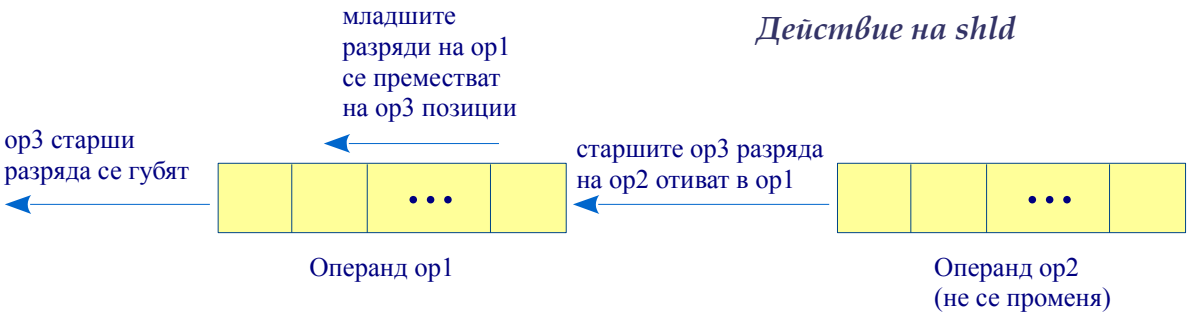
scasw – изчислява разликата AX – word ptr ES:[EDI] без да съхранява получения резултат, но като модифицира флаговете, точно както биха ги променили командите cmp или sub със същите умаляемо и умалител, и след това прибавя към EDI или $+2$, когато флагът $DF=0$, или -2, когато флагът $DF=1$; може да се зацикля чрез префиксите rep, repz, repe, repnz и repne

set... op – записва в op, който трябва да бъде 8-разряден, или 1, или 0. Записваната стойност се интерпретира съответно като true или folse за изпълнението на логическото условие, назовано с буквите, заместващи многоточието. То може да се замества със същите букви, които може да се пишат в командата за условен преход **j...** след буквата **j** (описани са в края на файла).

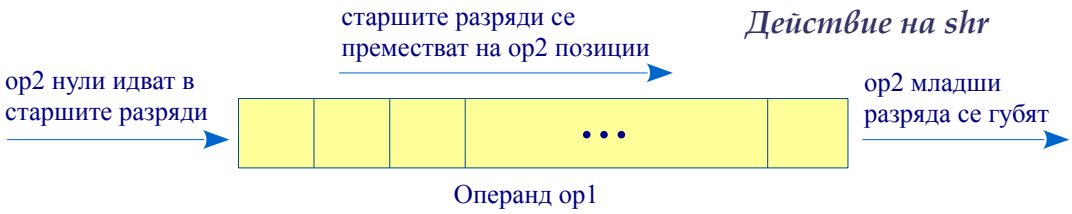
shl op1, op2 – логическо изместване на op1 наляво на op2 позиции (старшите op2 разряда се губят, а отдясно се появява op2 нули); op2 е 8-разряден и е или число, или регистъра CL; работи, както е показано на схемата (еквивалентно с умножение по 2^{op2}):



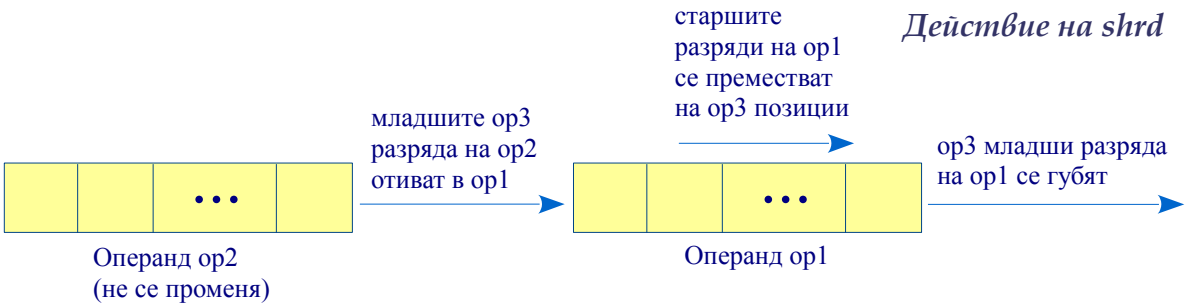
shld op1, op2, op3 – изместване с двойна точност на op1 наляво на op3 позиции като в op1 отдясно идват старшите разряди на op2; op2 не се променя; op3 е 8-разряден и е или число, или регистъра CL; работи, както е показано на схемата:



shr op1, op2 – логическо изместване на op1 наляво на op2 позиции (старшите op2 разряда се губят, а отдясно се появява op2 нули); еквивалентно е на делене с 2^{op2} само за кодове на положителни числа; op2 е 8-разряден и е или число, или регистъра CL; shr работи по схемата:



shrd op1, op2, op3 – изместване с двойна точност на op1 надясно на op3 позиции като в op1 отляво идват младшите разряди на op2; op2 не се променя; op3 е 8-разряден и е или число, или регистъра CL; работи, както е показано на схемата:



stc – записва 1 във флага CF

std – записва 1 във флага DF

stosb – записва на адрес ES:[EDI] съдържанието на регистъра AL и след това прибавя към EDI или $+1$, когато флагът $DF=0$, или -1, когато флагът $DF=1$; може да се зацикля чрез префиксите rep, repz, repe, repnz и repne

stosd – записва на адрес ES:[EDI] съдържанието на регистъра EAX и след това прибавя към EDI или $+4$, когато флагът $DF=0$, или -4, когато флагът $DF=1$; може да се зацикля чрез префиксите rep, repz, repe, repnz и repne

stosw – записва на адрес ES:[EDI] съдържанието на регистъра AX и след това прибавя към EDI или $+2$, когато флагът $DF=0$, или -2, когато флагът $DF=1$; може да се зацикля чрез префиксите rep, repz, repe, repnz и repne

sub op1, op2 – записва в op1 разлика с умаляемо op1 и умалител op2 (т. е. записва $op1-op2$)

test op1, op2 – изчислява поразрядното И (AND; поразрядната конюнкция) на op1 и op2 без да го съхранява, но модифицира флаговете, точно както би ги променила командата and op1, op2

xadd op1, op2 – записва в op1 сумата на op1 и op2, а в op2 записва началната (преди сумирането) стойност на op1

xchg op1, op2 – разменя местата (стойностите) на op1 и op2

xor op1, op2 – записва в op1 поразрядното Разделително Или (XOR) на op1 и op2

Списък по видове обработки (по групи команди)

Операции за управление на блока за работа с кодове с плаваща запетая

F... *(многоточието назовава действието).*

Аритметични операции за кодове с двоично кодиране на стойността (допълнителен код и код без знак)

ADC (събиране заедно с флага CF);
ADD (събиране);
CMP (изваждане с цел сравняване);
CMPXCHG (изваждане с цел сравняване и обмен според равенството);
DEC (намаляване с 1);
DIV (делене на цели без знак);
IDIV (делене на цели със знак);
IMUL (умножение на цели със знак);
INC (увеличаване с 1);
MUL (умножение на цели без знак);
NEG (смяна на знака);
SBB (изваждане със заем от флага CF);
SUB (изваждане);
XADD (размяна и събиране).

Команди за поддръжка на двоично-десетична аритметика от целочисления блок

а) в непакетиран формат:
AAA (след събиране);
AAD (преди деление);
AAM (след умножение);
AAS (след изваждане).
б) в пакетиран формат:
DAA (след събиране);
DAS (след изваждане).

Логически операции (всички са поразрядни)

AND (логическо и);
NOT (отрицание);
OR (логическо или);
TEST (сравняване чрез логическо и);
XOR (логическо изключващо или).

Измествания

RCL (ротация наляво през CF);
RCR (ротация надясно през CF);
ROL (ротация наляво);
ROR (ротация надясно);
SAL (аритметично наляво);
SAR (аритметично надясно);
SHL (логическо наляво);
SHLD (“двойно” изместване наляво, т. е. с извличане от друга данна);
SHR (логическо надясно);
SHRD (“двойно” изместване надясно, т. е. с извличане от друга данна).

Побитови обработки

BSF (търсене на единичен бит напред);
BSR (търсене на единичен бит назад);
BT (извличане);
BTC (извличане и инвертиране);

BTR (извличане и нулиране);
BTS (извличане и записване на единица);
SET# (записва в байт стойността на логическо условие като еднобайтова единица или нула;
замества различните мнемонични съкращения от таблицата за условни преходи).

Празна операция

NOP (само изразходва 1 такт).

Работа с флагове

CLC (нулиране на CF);
CLD (нулиране на DF);
CLI (нулиране на IF; привилегирована);
CMC (инвертиране на CF);
LAHF (младшият байт на FLAGS в AH);
SAHF (AH в младшия байт на FLAGS);
STC (записване на 1 във флага CF);
STD (записване на 1 във флага DF) ;
STI (записване на 1 във флага IF; привилегирована).
Също и POPF, POPFD, PUSHF, PUSHFD.

Предаване на управлението (преходи)

CALL (обръщение към подпрограма);
J# (условен преход; # замества различните мнемонични съкращения);
JCXZ (преход при CX=0);
JECXZ (преход при ECX=0);
JMP (безусловен преход);
LOOP (за цикъл по ECX);
LOOPE (за цикъл по ECX и ZF);
LOOPNE (за цикъл по ECX и не ZF);
LOOPNZ (=LOOPNE);
LOOPZ (=LOOPE);
RET (връщане от подпрограма).

Преобразувания и прехвърляния

BSWAP (размяна на байтовете);
CBW (байт в дума);
CDQ (двойна в четворна дума);
CWD (дума в двойна дума с участие на DX);
CWDE (дума в двойна дума само в EAX);
LDS (зареждане на пълен логически адрес, включително в DS);
LEA (зареждане на ефективен адрес);
LES (зареждане на пълен логически адрес, включително в ES);
LFS (зареждане на пълен логически адрес, включително в FS);
LGS (зареждане на пълен логически адрес, включително в GS);
LSL (зареждане на граница на сегмент);
LSS (зареждане на пълен логически адрес, включително в SS);
MOV (записване на избрано място);
MOVSX (прехвърляне със знаково разширяване на разрядността);
MOVZX (прехвърляне с беззнаково разширяване на разрядността);
XCHG (размяна на местата на операндите);
XLAT (извлича байт от таблица от до 256 байта) ;
XLATB (като XLAT, но само с регистъра DS).

Работа със стека

ENTER (формиране на стеков кадър);
LAHF (запис на младшия байт от флагове в регистъра AH);
LEAVE (премахване на стеков кадър);
POP (извличане на една данна);

POPA (извличане на 16-разрядните регистри с общо предназначение);
POPAD (извличане на 32-разрядните регистри с общо предназначение);
POPF (извличане на 16-разрядния флагов регистър);
POPFD (извличане на 32-разрядния флагов регистър; при привилегии 1, 2 и 3 не се променят флаговете VM и RF);
PUSH (записване на единична данна);
PUSHA (записване на 16-разрядните регистри с общо предназначение);
PUSHAD (записване на 32-разрядните регистри с общо предназначение);
PUSHF (записване на 16-разрядния флагов регистър);
PUSHFD (записване на 32-разрядния флагов регистър).

Проверка на индекс или памет

BOUND (дали индекс е в граници);
VERR (дали сегмент е достъпен за четене);
VERW (дали сегмент е достъпен за запис).

За управление на работата с паметта

LOCK (за сигнал LOOK# за монополно владеене на паметта през следващата команда);
WBINVD (обратен запис и недостоверност на кеш-паметта).

Работа с низове (вектори от байтове, думи или двойни думи)

CMPS (изваждане на елементи от низове с цел сравняване);
CMPSB (изваждане за сравняване на байтове от низове);
CMPSD (изваждане за сравняване на двойни думи от низове);
CMPSW (изваждане за сравняване на думи от низове);
INS (въвеждане от порт; влияе се от привилегиите);
INSB (въвеждане от порт на байт; влияе се от привилегиите);
INSD (въвеждане от порт на двойна дума; влияе се от привилегиите);
INSW (въвеждане от порт на дума; влияе се от привилегиите);
LODS (за запис на елемент в акумулатора);
LODSB (за запис на байт в акумулатора AL);
LODSD (за запис на двойна дума в акумулатора EAX);
LODSW (за запис на дума в акумулатора AX);
MOVS (прехвърляне от един към друг низ);
MOVSB (прехвърляне на байт от един към друг низ);
MOVSD (прехвърляне на двойна дума от един към друг низ);
MOVSW (прехвърляне на дума от един към друг низ);
OUTS (записване на един елемент в порт; влияе се от привилегиите);
OUTSB (записване на байт в порт; влияе се от привилегиите);
OUTSD (записване на двойна дума в порт; влияе се от привилегиите);
OUTSW (записване на дума в порт; влияе се от привилегиите);
REP (повторение ECX пъти);
REPE (повторение ECX пъти при ZF);
REPNE (повторение ECX пъти при не ZF);
REPNZ (=REPNE);
REPZ (=REPE);
SCAS (сравнява чрез изваждане акумулатора с елемент на низ);
SCASB (сравнява чрез изваждане на акумулатора AL с байт от низ);
SCASD (сравнява чрез изваждане на акумулатора EAX с двойна дума от низ);
SCASW (сравнява чрез изваждане на акумулатора AX с дума от низ);
STOS (записва акумулатора в елемент на низ) ;
STOSB (записва AL в елемент на низ от байтове) ;
STOSD (записва EAX в елемент на низ от двойни думи);
STOSW (записва AX в елемент на низ от думи).

Команди за системно програмиране (не са описани подробно в този справочник)

ARPL (промяна привилегия);
CLTS (нулиране на флага TS в регистъра CR0);
FWAIT (преди да продължи процесорът проверява за някои особени случаи);
HLT (спиране на процесора до възникване на прекъсване);
IN (въвеждане от порт);
INT (предизвикване на избрано прекъсване);
INTO (предизвикване на прекъсване 4);
INVD (недостоверност на кеш-паметта);
INVLPG (недостоверност на елемент от TLB);
IRET (връщане от прекъсване);
IRETD (=IRET);
LAR (зареждане на права за достъп);
LGDT (зареждане на регистъра GDTR);
LIDT (зареждане на регистъра IDTR);
LLDT (зареждане на регистъра LDTR);
LMSW (зареждане на думата на състоянието);
LTR (зареждане регистъра на задачата);
OUT (извеждане в порт);
SGDT (извличане на съдържанието на регистъра GDTR);
SIDT (извличане на съдържанието на регистъра IDTR);
SLDT (извличане на съдържанието на регистъра LDTR);
SMSW (извличане на думата на състоянието, т. е. на младшата половина на регистъра CR0);
STR (съхраняване на регистъра TR на задачата);
WAIT (преди да продължи процесорът проверява за някои особени случаи).

Също така от привилегиите зависят и командите:

CLI; CLTS; INS; INSB; INSD; INSW; IRET; IRETD; LSL; OUTS; OUTSB; OUTSD; OUTSW; POPFD; STI.