aba 8-bit add RegA=RegA+RegB  cpd 16-bit compare RegD with memory
abx unsigned add RegX=RegX+RegB  cpdx 16-bit compare RegX with memory
aby unsigned add RegY=RegY+RegB  cpy 16-bit compare RegY with memory
adca 8-bit add with carry to RegA  daa 8-bit decimal adjust accumulator
adcb 8-bit add with carry to RegB  dbeq decrement and branch if result=0
add 8-bit add to RegA  dbeq Y,loop
addb 8-bit add to RegB  dbeq Y,loop
addd 16-bit add to RegD  dec 8-bit decrement memory
anda 8-bit logical and to RegA  deca 8-bit decrement RegA
andb 8-bit logical and to RegB  decb 8-bit decrement RegB
andcc 8-bit logical and to RegCC  des 16-bit decrement RegSP
asl/lsl 8-bit left shift Memory  dx 16-bit decrement RegX
asl/lsla 8-bit left shift RegA  dey 16-bit decrement RegY
aslb/lslb 8-bit arith left shift RegB  asld/lsld 16-bit left shift RegD
asr 8-bit arith right shift Memory  emaca 16 by 16 signed mult, 32-bit add
asra 8-bit arith right shift to RegA  emaxd 16-bit unsigned maximum in RegD
asrb 8-bit arith right shift to RegB  bcc branch if carry clear
bclr branch clear in memory  bccr PTT,$\#01$
bcs branch if carry set  bcs branch if carry set
beq branch if result is zero (Z=1)  beeq 8-bit logical exclusive or to RegA
bge branch if signed ≥  bge branch if signed ≥
bgt branch if signed >  bgnd enter background debug mode
bhi branch if unsigned >  bgt branch if signed >
bhs branch if unsigned ≥  bhi branch if unsigned >
bis branch if signed ≤  bhi branch if unsigned ≥
blo branch if unsigned <  bhs branch if unsigned ≥
bs branch if unsigned ≤  bps branch if unsigned ≤
bit branch if signed <  bps branch if unsigned ≤
 bmi branch if result is negative (N=1)  bps branch if unsigned ≤
 bne branch if result is nonzero (Z=0)  bidiv 16-bit unsign d, X=D/X, D=rem
bnpl branch if result is positive (N=0)  bidiv 16-bit unsign d, X=D/X, D=rem
bpl branch if result is positive (N=0)  bne branch if unsigned <
br branch always  bb branch if unsigned <
brclr branch if bits are clear  brn branch never
brclr PTT,$\#01$,loop  brset branch if bits are set
brs branch if bits are set  brs set PTT,$\#01$,loop
brset PTT,$\#01$,loop  bset bit set clear in memory
bset bit set clear in memory  bset PTT,$\#004$
bsr branch to subroutine  bsl branch to subroutine
bvc branch if overflow clear  bvc branch to subroutine
bvs branch if overflow set  bvs branch if overflow set
call subroutine in expanded memory  call subroutine in expanded memory
cbea 8-bit compare RegA with RegB  call subroutine in expanded memory
clc clear carry bit, C=0  call subroutine in expanded memory
cli clear I=0, enable interrupts  call subroutine in expanded memory
clra clear RegA  clra clear RegA
clrb RegB clear  clrb RegB clear
clv clear overflow bit, V=0  clv clear overflow bit, V=0
cmpa 8-bit compare RegA with memory  cmpa 8-bit compare RegA with memory
cmpb 8-bit compare RegB with memory  cmpb 8-bit compare RegB with memory
com 8-bit logical complement to memory  com 8-bit logical complement to memory
comaa 8-bit logical complement to RegA  coma 8-bit logical complement to RegA
comaba 8-bit logical complement to RegB  comb 8-bit logical complement to RegB
<table>
<thead>
<tr>
<th>Pseudo op</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>org</td>
<td>Specific absolute address to put subsequent object code</td>
</tr>
<tr>
<td>=</td>
<td>equ</td>
</tr>
<tr>
<td></td>
<td>Define a constant symbol</td>
</tr>
<tr>
<td>set</td>
<td>Define or redefine a constant symbol</td>
</tr>
<tr>
<td>dc.b</td>
<td>db</td>
</tr>
<tr>
<td></td>
<td>fcb .byte</td>
</tr>
<tr>
<td></td>
<td>Allocate byte(s) of storage with initialized values</td>
</tr>
<tr>
<td>fcc</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Create an ASCII string (no termination character)</td>
</tr>
<tr>
<td>dc.w</td>
<td>dw</td>
</tr>
<tr>
<td></td>
<td>fdb .word</td>
</tr>
<tr>
<td></td>
<td>Allocate word(s) of storage with initialized values</td>
</tr>
<tr>
<td>dc.l</td>
<td>dl .long</td>
</tr>
<tr>
<td></td>
<td>Allocate 32-bit long word(s) of storage with initialized values</td>
</tr>
<tr>
<td>ds</td>
<td>ds.b rmb .blkb</td>
</tr>
<tr>
<td></td>
<td>Allocate bytes of storage without initialization</td>
</tr>
<tr>
<td>ds.w</td>
<td>.blkw</td>
</tr>
<tr>
<td></td>
<td>Allocate bytes of storage without initialization</td>
</tr>
<tr>
<td>ds.l</td>
<td>.blkl</td>
</tr>
<tr>
<td></td>
<td>Allocate 32-bit words of storage without initialization</td>
</tr>
</tbody>
</table>