

| | | | |
|------------|---|-------|--|
| aba | 8-bit add RegA=RegA+RegB | des | 16-bit decrement RegSP |
| abx | unsigned add RegX=RegX+RegB | dex | 16-bit decrement RegX |
| aby | unsigned add RegY=RegY+RegB | dey | 16-bit decrement RegY |
| adca | 8-bit add with carry to RegA | ediv | RegY=(Y:D)/RegX, unsigned divide |
| adcb | 8-bit add with carry to RegB | edivs | RegY=(Y:D)/RegX, signed divide |
| adda | 8-bit add to RegA | emacs | 16 by 16 signed multiply, 32-bit add |
| addb | 8-bit add to RegB | emaxd | 16-bit unsigned maximum in RegD |
| addd | 16-bit add to RegD | emaxm | 16-bit unsigned maximum in memory |
| anda | 8-bit logical and to RegA | emind | 16-bit unsigned minimum in RegD |
| andb | 8-bit logical and to RegB | eminm | 16-bit unsigned minimum in memory |
| andcc | 8-bit logical and to RegCC | emul | RegY:D=RegY*RegD unsigned multiply |
| asl/lsl | 8-bit left shift Memory | emuls | RegY:D=RegY*RegD signed multiply |
| asla/lsla | 8-bit left shift RegA | eora | 8-bit logical exclusive or to RegA |
| aslb/lslb | 8-bit arith left shift RegB | eorb | 8-bit logical exclusive or to RegB |
| asld/lsl d | 16-bit left shift RegD | etbl | 16-bit look up and interpolation |
| asr | 8-bit arith right shift Memory | exg | exchange register contents exg X,Y |
| asra | 8-bit arith right shift to RegA | fdiv | unsigned fract div, X=(65536*D)/X |
| asrb | 8-bit arith right shift to RegB | ibeq | increment and branch if result=0 ibeq Y,loop |
| bcc | branch if carry clear | ibne | increment and branch if result≠0 ibne A,loop |
| bclr | bit clear in memory bclr PTT, #01 | idiv | 16-bit unsigned div, X=D/X, D=remainder |
| bcs | branch if carry set | idivs | 16-bit signed divide, X=D/X, D= remainder |
| beq | branch if result is zero (Z=1) | inc | 8-bit increment memory |
| bge | branch if signed ≥ | inca | 8-bit increment RegA |
| bgnd | enter background debug mode | incb | 8-bit increment RegB |
| bgt | branch if signed > | ins | 16-bit increment RegSP |
| bhi | branch if unsigned > | inx | 16-bit increment RegX |
| bhs | branch if unsigned ≥ | iny | 16-bit increment RegY |
| bita | 8-bit and with RegA, sets CCR | jmp | jump always |
| bitb | 8-bit and with RegB, sets CCR | jsr | jump to subroutine |
| ble | branch if signed ≤ | lbcc | long branch if carry clear |
| blo | branch if unsigned < | lbcS | long branch if carry set |
| bls | branch if unsigned ≤ | lbeq | long branch if result is zero |
| blt | branch if signed < | lbge | long branch if signed ≥ |
| bmi | branch if result is negative (N=1) | lbgt | long branch if signed > |
| bne | branch if result is nonzero (Z=0) | lbhi | long branch if unsigned > |
| bpl | branch if result is positive (N=0) | lbhs | long branch if unsigned ≥ |
| bra | branch always | lbl | long branch if signed ≤ |
| brclr | branch if bits are clear brclr PTT, #01, loop | lblo | long branch if unsigned < |
| brn | branch never | lbls | long branch if unsigned ≤ |
| brset | branch if bits are set brset PTT, #01, loop | lblt | long branch if signed < |
| bset | bit set in memory bset PTT, #04 | lbmi | long branch if result is negative |
| bsr | branch to subroutine | lbne | long branch if result is nonzero |
| bvc | branch if overflow clear | lbp1 | long branch if result is positive |
| bvs | branch if overflow set | lbra | long branch always |
| call | subroutine in expanded memory | lbrn | long branch never |
| cba | 8-bit compare RegA with RegB, RegA-RegB | lbvc | long branch if overflow clear |
| clc | clear carry bit, C=0 | lbvs | long branch if overflow set |
| cli | clear I=0, enable interrupts | ldaa | 8-bit load memory into RegA |
| clr | 8-bit memory clear | ldab | 8-bit load memory into RegB |
| clra | RegA clear | ladd | 16-bit load memory into RegD |
| clrb | RegB clear | lds | 16-bit load memory into RegSP |
| clv | clear overflow bit, V=0 | ldx | 16-bit load memory into RegX |
| cmpa | 8-bit compare RegA with memory | ldy | 16-bit load memory into RegY |
| cmpb | 8-bit compare RegB with memory | leas | 16-bit load effective addr to SP leas 2, sp |
| com | 8-bit logical complement to memory | leax | 16-bit load effective addr to X leax 2, x |
| coma | 8-bit logical complement to RegA | leay | 16-bit load effective addr to Y leay 2, y |
| comb | 8-bit logical complement to RegB | lsr | 8-bit logical right shift memory |
| cpd | 16-bit compare RegD with memory | lsra | 8-bit logical right shift RegA |
| cpX | 16-bit compare RegX with memory | lsrb | 8-bit logical right shift RegB |
| cpy | 16-bit compare RegY with memory | lsrd | 16-bit logical right shift RegD |
| daa | 8-bit decimal adjust accumulator | maxa | 8-bit unsigned maximum in RegA |
| dbeq | decrement and branch if result=0 dbeq Y, loop | maxm | 8-bit unsigned maximum in memory |
| dbne | decrement and branch if result≠0 dbne A, loop | mem | determine the Fuzzy logic membership grade |
| dec | 8-bit decrement memory | mina | 8-bit unsigned minimum in RegA |
| deca | 8-bit decrement RegA | minm | 8-bit unsigned minimum in memory |
| decb | 8-bit decrement RegB | movb | 8-bit move memory to memory movb #100, PTT |

movw 16-bit move memory to memory movw #13,SCIBD
mul unsigned RegD=RegA*RegB
neg 8-bit 2's complement negate memory
nega 8-bit 2's complement negate RegA
negb 8-bit 2's complement negate RegB
oraa 8-bit logical or to RegA
orab 8-bit logical or to RegB
orcc 8-bit logical or to RegCC
psha push 8-bit RegA onto stack
pshb push 8-bit RegB onto stack
pshc push 8-bit RegCC onto stack
pshd push 16-bit RegD onto stack
pshx push 16-bit RegX onto stack
pshy push 16-bit RegY onto stack
pula pop 8 bits off stack into RegA
pulb pop 8 bits off stack into RegB
pulc pop 8 bits off stack into RegCC
puld pop 16 bits off stack into RegD
pulx pop 16 bits off stack into RegX
puly pop 16 bits off stack into RegY
rev Fuzzy logic rule evaluation
revw weighted Fuzzy rule evaluation
rol 8-bit roll shift left Memory
rola 8-bit roll shift left RegA
rolb 8-bit roll shift left RegB
ror 8-bit roll shift right Memory
rora 8-bit roll shift right RegA
rorb 8-bit roll shift right RegB
rtc return sub in expanded memory
rti return from interrupt
rts return from subroutine
sba 8-bit subtract RegA=RegA-RegB
sbca 8-bit sub with carry from RegA
sbc b 8-bit sub with carry from RegB
sec set carry bit, C=1
sei set I=1, disable interrupts
sev set overflow bit, V=1
sex sign extend 8-bit to 16-bit reg sex B,D
staa 8-bit store memory from RegA
stab 8-bit store memory from RegB
std 16-bit store memory from RegD
sts 16-bit store memory from SP
stx 16-bit store memory from RegX
sty 16-bit store memory from RegY
suba 8-bit sub from RegA
subb 8-bit sub from RegB
subd 16-bit sub from RegD
swi software interrupt, trap
tab transfer A to B
tap transfer A to CC
tba transfer B to A
tbeq test and branch if result=0 tbeq Y,loop
tbl 8-bit look up and interpolation
tbne test and branch if result!=0 tbne A,loop
tfr transfer register to register tfr X,Y
tpa transfer CC to A
trap illegal instruction interrupt
trap illegal op code, or software trap
tst 8-bit compare memory with zero
tsta 8-bit compare RegA with zero
tstb 8-bit compare RegB with zero
tsx transfer S to X
tsy transfer S to Y
txs transfer X to S
tys transfer Y to S
wai wait for interrupt
wav weighted Fuzzy logic average

xgdx exchange RegD with RegX
xgdy exchange RegD with RegY

| Example | Mode | Effective Address |
|------------|--------------|------------------------|
| ldaa #u | immediate | No EA |
| ldaa u | direct | EA is 8-bit address |
| ldaa U | extended | EA is a 16-bit address |
| ldaa m,r | 5-bit index | EA=r+m (-16 to 15) |
| ldaa v,+r | pre-incr | r=r+v, EA=r (1 to 8) |
| ldaa v,-r | pre-dec | r=r-v, EA=r (1 to 8) |
| ldaa v,r+ | post-inc | EA=r, r=r+v (1 to 8) |
| ldaa v,r- | post-dec | EA=r, r=r-v (1 to 8) |
| ldaa A,r | Reg A offset | EA=r+A, zero padded |
| ldaa B,r | Reg B offset | EA=r+B, zero padded |
| ldaa D,r | Reg D offset | EA=r+D |
| ldaa q,r | 9-bit index | EA=r+q |
| ldaa W,r | 16-bit index | EA=r+W |
| ldaa [D,r] | D indirect | EA={r+D} |
| ldaa [W,r] | indirect | EA={r+W} |

Freescale 6812 addressing modes r is X, Y, SP, or PC

| Pseudo op | Meaning |
|--------------------------|-------------------------------|
| org | Where to put subsequent code |
| = equ set | Define a constant symbol |
| dc.b db fcb .byte | Allocate byte(s) with values |
| fcc | Create an ASCII string |
| dc.w dw fdb .word | Allocate word(s) with values |
| dc.l dl .long | Allocate 32-bit with values |
| ds ds.b rmb .blkb | Allocate bytes without init |
| ds.w .blkw | Allocate word(s) without init |

n is Metrowerks number

| Vector | n | Interrupt Source | Arm |
|--------|----|----------------------------|------------------|
| \$FFFE | | Reset | None |
| \$FFF8 | 3 | Trap | None |
| \$FFF6 | 4 | SWI | None |
| \$FFF0 | 7 | Real time interrupt | CRGINT.RTIE |
| \$FFEE | 8 | Timer channel 0 | TIE.C0I |
| \$FFEC | 9 | Timer channel 1 | TIE.C1I |
| \$FFEA | 10 | Timer channel 2 | TIE.C2I |
| \$FFE8 | 11 | Timer channel 3 | TIE.C3I |
| \$FFE6 | 12 | Timer channel 4 | TIE.C4I |
| \$FFE4 | 13 | Timer channel 5 | TIE.C5I |
| \$FFE2 | 14 | Timer channel 6 | TIE.C6I |
| \$FFE0 | 15 | Timer channel 7 | TIE.C7I |
| \$FFDE | 16 | Timer overflow | TSCR2.TOI |
| \$FFD6 | 20 | SCIO TDRE, RDRF | SCIOCR2.TIE,RIE |
| \$FFD4 | 21 | SCII TDRE, RDRF | SCII CR2.TIE,RIE |
| \$FFCE | 24 | Key Wakeup J | PIEJ.[7,6,1,0] |
| \$FFCC | 25 | Key Wakeup H | PIEH.[7:0] |
| \$FF8E | 56 | Key Wakeup P | PIEP.[7:0] |

Interrupt Vectors and interrupt number.

| Address | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 | Name |
|----------|--------|---------|-------|---------|--------|-------|-------|-------|-----------|
| \$0040 | IOS7 | IOS6 | IOS5 | IOS4 | IOS3 | IOS2 | IOS1 | IOS0 | TIOS |
| \$0044-5 | Bit 15 | 14 | 13 | 12 | 11 | 10 | | Bit 0 | TCNT |
| \$0046 | TEN | TSWAI | TSFRZ | TFFCA | 0 | 0 | 0 | 0 | TSCR1 |
| \$004C | C7I | C6I | C5I | C4I | C3I | C2I | C1I | C0I | TIE |
| \$004D | TOI | 0 | PUPT | RDPT | TCRE | PR2 | PR1 | PR0 | TSCR2 |
| \$004E | C7F | C6F | C5F | C4F | C3F | C2F | C1F | C0F | TFLG1 |
| \$004F | TOF | 0 | 0 | 0 | 0 | 0 | 0 | 0 | TFLG2 |
| \$0050-1 | Bit 15 | 14 | 13 | 12 | 11 | 10 | | Bit 0 | TC0 |
| \$0052-3 | Bit 15 | 14 | 13 | 12 | 11 | 10 | | Bit 0 | TC1 |
| \$0054-5 | Bit 15 | 14 | 13 | 12 | 11 | 10 | | Bit 0 | TC2 |
| \$0056-7 | Bit 15 | 14 | 13 | 12 | 11 | 10 | | Bit 0 | TC3 |
| \$0058-9 | Bit 15 | 14 | 13 | 12 | 11 | 10 | | Bit 0 | TC4 |
| \$005A-B | Bit 15 | 14 | 13 | 12 | 11 | 10 | | Bit 0 | TC5 |
| \$005C-D | Bit 15 | 14 | 13 | 12 | 11 | 10 | | Bit 0 | TC6 |
| \$005E-F | Bit 15 | 14 | 13 | 12 | 11 | 10 | | Bit 0 | TC7 |
| \$0082 | ADPU | AFFC | ASWAI | ETRIGLE | ETRIGP | ETRIG | ASCIE | ASCIF | ATD0CTL2 |
| \$0083 | 0 | S8C | S4C | S2C | S1C | FIFO | FRZ1 | FRZ0 | ATD0CTL3 |
| \$0084 | SRES8 | SMP1 | SMP0 | PRS4 | PRS3 | PRS2 | PRS1 | PRS0 | ATD0CTL4 |
| \$0085 | DJM | DSGN | SCAN | MULT | 0 | CC | CB | CA | ATD0CTL5 |
| \$0086 | SCF | 0 | ETORF | FIFOR | 0 | CC2 | CC1 | CC0 | ATD0STAT0 |
| \$008B | CCF7 | CCF6 | CCF5 | CCF4 | CCF3 | CCF2 | CCF1 | CCF0 | ATD0STAT1 |
| \$008D | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 | ATD0DIEN |
| \$008F | PAD07 | PAD06 | PAD05 | PAD04 | PAD03 | PAD02 | PAD01 | PAD00 | PORTAD0 |
| \$0090-1 | Bit 15 | 14 | 13 | 12 | 11 | 10 | | Bit 0 | ATD0DR0 |
| \$0092-3 | Bit 15 | 14 | 13 | 12 | 11 | 10 | | Bit 0 | ATD0DR1 |
| \$0094-5 | Bit 15 | 14 | 13 | 12 | 11 | 10 | | Bit 0 | ATD0DR2 |
| \$0096-7 | Bit 15 | 14 | 13 | 12 | 11 | 10 | | Bit 0 | ATD0DR3 |
| \$0098-9 | Bit 15 | 14 | 13 | 12 | 11 | 10 | | Bit 0 | ATD0DR4 |
| \$009A-B | Bit 15 | 14 | 13 | 12 | 11 | 10 | | Bit 0 | ATD0DR5 |
| \$009C-D | Bit 15 | 14 | 13 | 12 | 11 | 10 | | Bit 0 | ATD0DR6 |
| \$009E-F | Bit 15 | 14 | 13 | 12 | 11 | 10 | | Bit 0 | ATD0DR7 |
| \$00C9 | 0 | 0 | 0 | SBR12 | SBR11 | SBR10 | | SBR0 | SCIOBD |
| \$00CA | LOOPS | SCISWAI | RSRC | M | WAKE | ILT | PE | PT | SCIOCR1 |
| \$00CB | TIE | TCIE | RIE | ILIE | TE | RE | RWU | SBK | SCIOCR2 |
| \$00CC | TDRE | TC | RDRF | IDLE | OR | NF | FE | PF | SCIOSR1 |
| \$00CD | 0 | 0 | 0 | 0 | 0 | BRK13 | TXDIR | RAF | SCIOSR2 |
| \$00CF | R7/T7 | R6/T6 | R5/T5 | R4/T4 | R3/T3 | R2/T2 | R1/T1 | R0/T0 | SCIODRL |
| \$00D0-1 | 0 | 0 | 0 | SBR12 | SBR11 | SBR10 | | SBR0 | SCII1BD |
| \$00D2 | LOOPS | SCISWAI | RSRC | M | WAKE | ILT | PE | PT | SCII1CR1 |
| \$00D3 | TIE | TCIE | RIE | ILIE | TE | RE | RWU | SBK | SCII1CR2 |
| \$00D4 | TDRE | TC | RDRF | IDLE | OR | NF | FE | PF | SCII1SR1 |
| \$00D5 | 0 | 0 | 0 | 0 | 0 | BRK13 | TXDIR | RAF | SCII1SR2 |
| \$00D7 | R7/T7 | R6/T6 | R5/T5 | R4/T4 | R3/T3 | R2/T2 | R1/T1 | R0/T0 | SCII1DRL |
| \$0240 | PT7 | PT6 | PT5 | PT4 | PT3 | PT2 | PT1 | PT0 | PTT |
| \$0242 | DDRT7 | DDRT6 | DDRT5 | DDRT4 | DDRT3 | DDRT2 | DDRT1 | DDRT0 | DDRT |
| \$0248 | PS7 | PS6 | PS5 | PS4 | PS3 | PS2 | PS1 | PS0 | PTS |
| \$024A | DDRS7 | DDRS6 | DDRS5 | DDRS4 | DDRS3 | DDRS2 | DDRS1 | DDRS0 | DDRS |
| \$0250 | PM7 | PM6 | PM5 | PM4 | PM3 | PM2 | PM1 | PM0 | PTM |
| \$0252 | DDRM7 | DDRM6 | DDRM5 | DDRM4 | DDRM3 | DDRM2 | DDRM1 | DDRM0 | DDRM |
| \$0258 | PP7 | PP6 | PP5 | PP4 | PP3 | PP2 | PP1 | PP0 | PTP |
| \$025A | DDRP7 | DDRP6 | DDRP5 | DDRP4 | DDRP3 | DDRP2 | DDRP1 | DDRP0 | DDRP |
| \$0260 | PH7 | PH6 | PH5 | PH4 | PH3 | PH2 | PH1 | PH0 | PTH |
| \$0262 | DDRH7 | DDRH6 | DDRH5 | DDRH4 | DDRH3 | DDRH2 | DDRH1 | DDRH0 | DDRH |
| \$0268 | PJ7 | PJ6 | 0 | 0 | 0 | 0 | PJ1 | PJ0 | PTJ |
| \$026A | DDRJ7 | DDRJ6 | 0 | 0 | 0 | 0 | DDRJ1 | DDRJ0 | DDRJ |

TSCR1 is the first 8-bit timer control register

bit 7 **TEN**, 1 allows the timer to function normally, 0 means disable timer including **TCNT**

TIOS is the 8-bit output compare select register, one bit for each channel (1 = output compare, 0 = input capture)

TIE is the 8-bit output compare arm register, one bit for each channel (1 = armed, 0 = disarmed)

TSCR2 is the second 8-bit timer control register

bits 2,1,0 are **PR2**, **PR1**, **PR0**, which select the rate, let **n** be the 3-bit number formed by **PR2**, **PR1**, **PR0** without PLL **TCNT** is $8\text{MHz}/2^n$, with PLL **TCNT** is $24\text{MHz}/2^n$, **n** ranges from 0 to 7

| PR2 | PR1 | PR0 | Divide by | E = 8 MHz | | E = 24 MHz | |
|-----|-----|-----|--------------|------------------|-------------------|--------------------|-------------------|
| | | | | TCNT period | TCNT frequency | TCNT period | TCNT frequency |
| 0 | 0 | 0 | 1 | 125 ns | 8 MHz | 41.7 ns | 24 MHz |
| 0 | 0 | 1 | 2 | 250 ns | 4 MHz | 83.3 ns | 12 MHz |
| 0 | 1 | 0 | 4 | 500 ns | 2 MHz | 167 ns | 6 MHz |
| 0 | 1 | 1 | 8 | 1 μs | 1 MHz | 333 ns | 3 MHz |
| 1 | 0 | 0 | 16 | 2 μs | 500 kHz | 667 ns | 1.5 MHz |
| 1 | 0 | 1 | 32 | 4 μs | 250 kHz | 1.33 μs | 667 kHz |
| 1 | 1 | 0 | 64 | 8 μs | 125 kHz | 2.67 μs | 333 kHz |
| 1 | 1 | 1 | 128 | 16 μs | 62.5 kHz | 5.33 μs | 167 kHz |

SCI0DRL 8-bit SCI data register

SCI0BD is 16-bit SCI0 baud rate register, let **n** be the 13-bit number Baud rate is $\text{EClk}/n/16$

SCI0CR1 is 8-bit SCI0 control register

bit 4 M, Mode, 0 = One start, eight data, one stop bit, 1 = One start, eight data, ninth data, one stop bit

SCI0CR2 is 8-bit SCI0 control register

bit 7 TIE, Transmit Interrupt Enable, 0 = TDRE interrupts disabled, 1 = interrupt whenever TDRE set

bit 5 RIE, Receiver Interrupt Enable, 0 = RDRF interrupts disabled, 1 = interrupt whenever RDRF set

bit 3 TE, Transmitter Enable, 0 = Transmitter disabled, 1 = SCI transmit logic is enabled

bit 2 RE, Receiver Enable, 0 = Receiver disabled, 1 = Enables the SCI receive circuitry.

SCI0SR1 is 8-bit SCI0 status register

bit 7 TDRE, Transmit Data Register Empty Flag

Set if transmit data can be written to **SCI0DRL**

Cleared by **SCI0SR1** read with TDRE set followed by **SCI0DRL** write.

bit 5 RDRF, Receive Data Register Full

set if a received character is ready to be read from **SCI0DRL**

Clear the RDRF flag by reading **SCI0SR1** with RDRF set and then reading **SCI0DRL**.