abx	unsigned add RegX+RegB		
aby	unsigned add RegY+RegB	inx	16-bit increment RegX
adca	8-bit add with carry to RegA	iny	16-bit increment RegY
adcb	8-bit add with carry to RegB	jmp	jump always
adda	8-bit add to RegA	jsr	jump to subroutine
addb	8-bit add to RegB	ldaa	8-bit load memory into RegA
addd	16-bit add to RegD	ldab	8-bit load memory into RegB
anda	8-bit logical and to RegA	ldd	16-bit load memory into RegD
andb	8-bit logical and to RegB	lds	16-bit load memory into RegSP
asl/lsl	8-bit left shift Memory	ldx	16-bit load memory into RegX
asla/lsla	8-bit left shift RegA	ldy	16-bit load memory into RegY
aslb/lslb	8-bit arith left shift RegB	lsr	8-bit logical right shift memory
asld/lsld	16-bit left shift RegD	lsra	8-bit logical right shift RegA
asr	8-bit arith right shift Memory	lsrb	8-bit logical right shift RegB
asra	8-bit arith right shift	lsrd	16-bit logical right shift RegD
asrb	8-bit arith right shift to RegB	mul	RegD=RegA*RegB
bcc	branch if carry clear	neg	8-bit 2's complement negate memory
bclr	clear bits in memory	nega	8-bit 2's complement negate RegA
bcs	branch if carry set	negb	8-bit 2's complement negate RegB
beq	branch if result is zero (Z=1)	oraa	8-bit logical or to RegA
bge	branch if signed ≥	orab	8-bit logical or to RegB
bgt	branch if signed >	psha	push 8-bit RegA onto stack
bhi	branch if unsigned >	pshb	push 8-bit RegB onto stack
bhs	branch if unsigned ≥	pshx	push 16-bit RegX onto stack
bita	8-bit and with RegA, sets CCR	pshy	push 16-bit RegY onto stack
bitb	8-bit and with RegB, sets CCR	pula	pop 8 bits off stack into RegA
ble	branch if signed ≤	pulb	pop 8 bits off stack into RegB
blo	branch if unsigned <	pulx	pop 16 bits off stack into RegX
bls	branch if unsigned ≤	puly	pop 16 bits off stack into RegY
blt	branch if signed <	rol	8-bit roll shift left Memory
bmi	branch if result is negative (N=1)	rola	8-bit roll shift left RegA
bne	branch if result is nonzero (Z=0)	rolb	8-bit roll shift left RegB
bpl	branch if result is positive (N=0)	ror	8-bit roll shift right Memory
bra	branch always	rora	8-bit roll shift right RegA
brclr	branch if bits are clear,	rorb	8-bit roll shift right RegB
brn	branch never	rti	return from interrupt
brset	branch if bits are set	rts	return from subroutine
bset	set bits in memory	sba	8-bit subtract RegA-RegB
bsr	branch to subroutine	sbca	8-bit sub with carry from RegA
bvc	branch if overflow clear	sbcb	8-bit sub with carry from RegB
bvs	branch if overflow set	sec	set carry bit, C=1
cba	8-bit compare RegA with RegB	sei	set I=1, disable interrupts
clc	clear carry bit, C=0	sev	set overflow bit, V=1
cli	clear I=0, enable interrupts	staa	8-bit store memory from RegA
clr	8-bit Memory clear	stab	8-bit store memory from RegB
clra	RegA clear	std	16-bit store memory from RegD
clrb	RegB clear	sts	16-bit store memory from SP
clv	clear overflow bit, V=0	stx	16-bit store memory from RegX
cmpa	8-bit compare RegA with memory	sty	16-bit store memory from RegY
cmpb	8-bit compare RegB with memory	suba	8-bit sub from RegA
com	8-bit logical complement to Memory	subb	8-bit sub from RegB
coma	8-bit logical complement to RegA	subd	16-bit sub from RegD
comb	8-bit logical complement to RegB	swi	software interrupt, trap
cpd	16-bit compare RegD with memory	tab	transfer A to B
cpx	16-bit compare RegX with memory	tap	transfer A to CC
cpy	16-bit compare RegY with memory	tba	transfer B to A
daa	8-bit decimal adjust accumulator	tpa	transfer CC to A
dec	8-bit decrement memory	trap	illegal op code, or software trap
deca	8-bit decrement RegA	tst	8-bit compare memory with zero
decb	8-bit decrement RegB	tsta	8-bit compare RegA with zero
des	16-bit decrement RegSP	tstb	8-bit compare RegB with zero
dex	16-bit decrement RegX	tsx	transfer S+1 to X
dey	16-bit decrement RegY	tsy	transfer S+1 to Y
eora	8-bit logical exclusive or to RegA	txs	transfer X-1 to S
eorb	8-bit logical exclusive or to RegB	tys	transfer Y-1 to S
fdiv	16-bit unsigned fractional divide	wai	wait for interrupt
idiv	16-bit unsigned divide	xgdx	exchange RegD with RegX
inc	8-bit increment memory	xgdy	exchange RegD with RegY
inca	8-bit increment RegA		

Motorola 6811 assembly instructions

andcc	8-bit logical and to RegCC	lbne	long branch if result is nonzero
bgnd	enter background debug mode	lbpl	long branch if result is positive
call	subroutine in expanded memory	lbra	long branch always
dbeq	decrement and branch if result=0	lbrn	long branch never
dbne	decrement and branch if result≠0	lbvc	long branch if overflow clear
ediv	RegY=(Y:D)/RegX, unsigned divide	lbvs	long branch if overflow set
edivs	RegY=(Y:D)/RegX, signed divide	leas	16-bit load effective addr to SP
emacs	16 by 16 signed mult, 32-bit add	leax	16-bit load effective addr to X
emaxd	16-bit unsigned maximum in RegD	leay	16-bit load effective addr to Y
emaxm	16-bit unsigned maximum in memory	maxa	8-bit unsigned maximum in RegA
emind	16-bit unsigned minimum in RegD	maxm	8-bit unsigned maximum in memory
eminm	16-bit unsigned minimum in memory	mem	determine the membership grade
emul	RegY:D=RegY*RegD unsigned mult	mina	8-bit unsigned minimum in RegA
emuls	RegY:D=RegY*RegD signed mult	minm	8-bit unsigned minimum in memory
etbl	16-bit look up and interpolation	movb	8-bit move memory to memory
exg	exchange register contents	movw	16-bit move memory to memory
ibeq	increment and branch if result=0	orcc	8-bit logical or to RegCC
ibne	increment and branch if result≠0	pshc	push 8-bit RegCC onto stack
idivs	16-bit by 16-bit signed divide	pshd	push 16-bit RegD onto stack
lbcc	long branch if carry clear	pulc	pop 8 bits off stack into RegCC
lbcs	long branch if carry set	puld	pop 16 bits off stack into RegD
lbeq	long branch if result is zero	rev	Fuzzy logic rule evaluation
lbge	long branch if signed ≥	revw	weighted Fuzzy rule evaluation
lbgt	long branch if signed >	rtc	return sub in expanded memory
lbhi	long branch if unsigned >	sex	sign extend 8-bit to 16-bit reg
lbhs	long branch if unsigned ≥	tbeq	test and branch if result=0
lble	long branch if signed ≤	tbl	8-bit look up and interpolation
lblo	long branch if unsigned <	tbne	test and branch if result≠0
lbls	long branch if unsigned ≤	tfr	transfer register to register
lblt	long branch if signed <	trap	illegal instruction interrupt
lbmi	long branch if result is negative	wav	weighted Fuzzy logic average

Motorola 6812 assembly instructions (in addition to the 6811)

example	addressing mode	Effective Address
ldaa #u	immediate	EA is 8-bit address (0 to 255)
ldaa u	direct	EA is 8-bit address (0 to 255)
ldaa U	extended	EA is a 16-bit address
ldaa m,r	8-bit index	EA=r+m (0 to 255)

Motorola 6811 addressing modes

example	addressing mode	Effective Address
ldaa m,r	5-bit index	EA=r+m (-16 to 15)
ldaa v,+r	pre-increment	r=r+v, EA=r (1 to 8)
ldaa v,-r	pre-decrement	r=r-v, EA=r (1 to 8)
ldaa v,r+	post-increment	EA=r, r=r+v (1 to 8)
ldaa v,r-	post-decrement	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 (-256 to 255)
ldaa W,r	16-bit index	EA=r+W (-32768 to 65535)
ldaa [D,r]	D indirect	EA={r+D}
ldaa [W,r]	indirect	EA={r+W} (-32768 to 65535)

Motorola 6812 addressing modes (in addition to the 6811)