First:________________ Middle Initial: _____ Last:____________________

EE319K Final Exam

Fall 2004

Jonathan W. Valvano

This is a closed book exam. You must put your answers in these boxes only. You have 3 hours, so allocate your time accordingly. ***Please read the entire exam before starting.***

(5) Question 1. Precision in decimal digits	
(2) Part 2a. Choose A-E	
(2) Part 2b. Choose A-E	
(2) Part 2c. Choose A-E	
(2) Part 2d. Choose A-E	
(2) Part 2e. Choose A-E	
(2) Part 3a. Specify `RegB`	
(2) Part 3b. Specify `0 or 1`	
(1) Part 3c. Specify `0 or 1`	
(2) Part 4a. Give the value for `xxx`	
(3) Part 4b. Give the value for `yyy`	
(5) Question 5. Give the value for `zzz`	
(5) Question 6. Give the value	
(5) Question 7. Give the hexadecimal value	
(5) Question 8. Give the op code	

(5) Question 9. Simplified memory cycles (you may or may not need all 5 entries)

R/W	Addr	Data

(5) Question 10. Show the assembly code

(10) Question 11. Hand assemble this program

Address	Machine code

Source code

```
     org  $3900
aa   rmb  2
bb   rmb  1
TCNT equ  $0044
     org  $4300
     ldx  TCNT
loop ldx  #cc
     ldaa 2,x
     staa bb
     bra  loop
cc   fcb  0,1,2,3,4
```

(15) Question 12. Show the assembly code

(20) Question 13. Show the assembly code

(5) Question 1. The measurement system range is 0 to 19.9 and a resolution of 0.1. What is the precision in decimal digits?

(10) Question 2. Place letter code for the best answer in the answer boxes

(2) Part a) Which direction does data flow on the data bus during a **read cycle**?

A) From 6812 to memory, or from 6812 to output device
B) From memory to 6812, or from input device to 6812
C) From input device to memory
D) From memory to output device
E) None of these answers is correct

(2) Part b) Which term best describes a computer system with a response time to external events that is short and bounded?

A) Real time
B) Dynamic
C) Busy-waiting
D) Nonintrusive
E) None of these answers is correct

(2) Part c) Which data structure has the following features? It can hold a variable number of fixed-size elements. It has two main operations, one to store data into itself, and a second operation to remove data. The data is removed in a "first come first served" order.

A) String
B) Binary tree
C) FIFO queue
D) MACQ
E) None of these answers is correct

(2) Part d) What is the difference between **busy-waiting** and **gadfly** synchronization?

A) Busy-waiting is used for I/O devices and gadfly is used for software events.
B) Busy-waiting allows you to perform other operations (so you can be busy) while waiting for I/O devices and gadfly simply waits for the I/O device to be ready.
C) The two terms describe exactly the same synchronization method.
D) Gadfly involves software loops, while busy-waiting involves hardware interrupts.
E) None of these answers is correct.

(2) Part e) What is **drop out**?

A) Drop out is the error that occurs when the result of a calculation exceeds the range of the number system.
B) Drop out is the error that occurs after a right shift or a divide, and the consequence is that an intermediate result looses its ability to represent all of the values.
C) Drop out is when both the Carry and the Overflow bits are set.
D) Drop out is data is lost when the software does not respond fast to an I/O event.
E) None of these answers is correct.

(5) Question 3. Consider the result of executing the following two 6812 assembly instructions.

```
ldab #170
addb #125
```

(2) Part a) What will be the unsigned decimal value in Register B (0 to 255)?

(2) Part b) What will be the value of the carry (C) bit?

(1) Part c) What will be the value of the overflow (V) bit?

(5) Question 4. Consider the following assembly subroutine that creates two local variables, called **p** and **q**. The variable **p** is 8 bits and initialized to 50. The variable **q** is 16 bits and initialized to 500. The local variable binding is created using the **set** pseudo-ops.

```
p    set   xxx         ; binding
q    set   yyy         ; binding
sub1 pshx              ; save register X
     movb #50,1,-sp    ; allocate and initialize p
     movb #500,2,-sp   ; allocate and initialize q
;... stuff
     ldaa P,sp  ; read from p
     ldx  Q,sp  ; read from q
;... more stuff
     leas 3,s    ; deallocate p,q
     pulx        ; restore register X
     rts         ; return
```

(2) Part a) What value should you use in the **xxx** position to implement the binding of **p**?

(3) Part b) What value should you use in the **yyy** position to implement the binding of **q**?

(5) Question 5. Consider the following main program that calls an assembly subroutine using call by reference parameter passing on the stack. An address to an output port is pushed on the stack, and this subroutine will set bit 0 of that port. The subroutine uses Register X stack frame.

```
port set  zzz     ; binding
main lds  #$4000
     ldd  #PTT    ; address to PTT
     pshd         ; pass 16-bit port parameter on stack
     jsr  Set0
     leas 2,sp
     stop
Set0 pshx          ; save register X
     tsx           ; create Register X stack frame
     leas -4,sp    ; allocate locals
     ldy  port,x   ; get port parameter
     bset 0,y,#$01 ; set bit0 of the port
     leas 4,sp     ; deallocate locals
     pulx          ; restore register X
     rts           ; return
```

What value should you use in the **zzz** position to implement the binding of this parameter?

(5) Question 6. A signed 16-bit decimal **fixed-point** number system has a Δ resolution of 1/100. What is the corresponding value of the number if the integer part stored in memory is $C000?

(5) Question 7. Assume RegX is initially $4321, RegD is initially $8765. What is resulting hexadecimal value in RegD after these instructions execute?

```
     psha
     pshb
     pshx
     pula
     leas 2,s
     pulb
```

(5) Question 8. Assume **`PTT`** is a signed 8-bit input and **`PTM`** is an output. The goal of this code is to clear **`PTM`** if **`PPT`** is larger than 100. Which op code should be used in the **`???`** position?

```
    ldaa PTT
    cmpa #100
    ???  skip
    clr  PTM
loop
```

(5) Question 9. Give the simplified memory cycles produced when the following one instruction is executed. Assume the PC contains $4000, Register Y contains $3900, each memory location from $3800 to $3FFF contains a value equal to the least significant byte of its address. I.e., $3800 contains $00, $3801 contains $01, etc. Just show R/W=Read or Write, Address, and Data for each cycle.

```
$4000 EE71   ldx 2,y+
```

(5) Question 10. Consider a table with 25 entries and 4 fields. Each entry has different values, but the same size and format. The assembly code for table entry number 5 is shown below

```
Entry5 fcb 100              8-bit count field
       fdb 1000             16-bit time field
       fcb "happy,",0,0,0  8-byte name field
       fdb -50,300,-200     three 16-bit yaw,pitch,roll fields
```

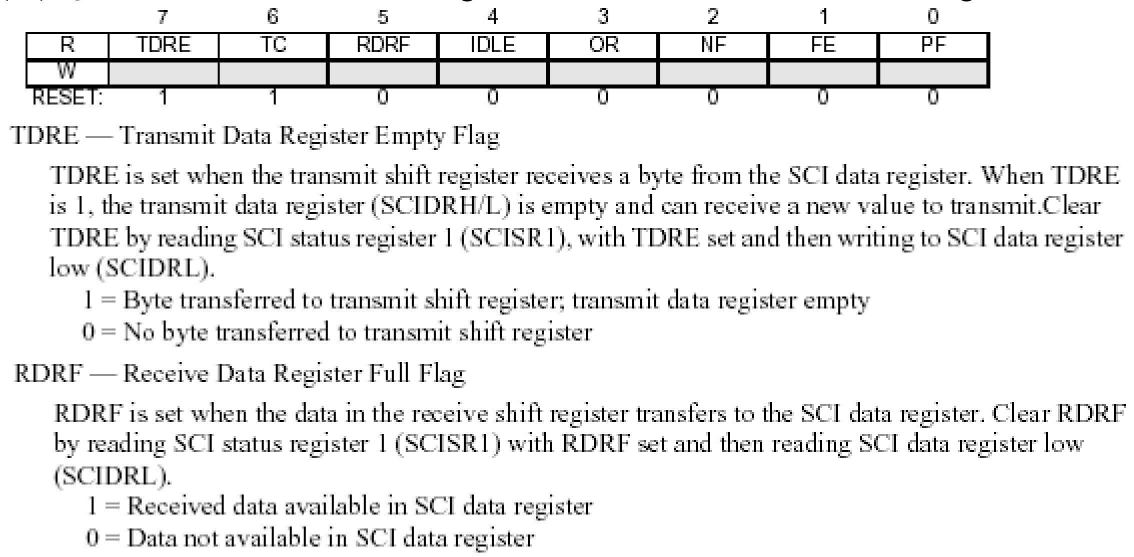
Assume Register X is pointing to this entry

```
       ldx  #Entry5
```

Using Register X, write assembly code that reads the **`pitch`** value of **`Entry5`** into a register

(10) Question 11. Hand assemble the program shown on the answer pages

(15) Question 12. The **`SCISR1`** register contains the **`TDRE`** and **`RDRF`** flags



	7	6	5	4	3	2	1	0
R	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
W								
RESET:	1	1	0	0	0	0	0	0

TDRE — Transmit Data Register Empty Flag

TDRE is set when the transmit shift register receives a byte from the SCI data register. When TDRE is 1, the transmit data register (SCIDRH/L) is empty and can receive a new value to transmit.Clear TDRE by reading SCI status register 1 (SCISR1), with TDRE set and then writing to SCI data register low (SCIDRL).

1 = Byte transferred to transmit shift register; transmit data register empty
0 = No byte transferred to transmit shift register

RDRF — Receive Data Register Full Flag

RDRF is set when the data in the receive shift register transfers to the SCI data register. Clear RDRF by reading SCI status register 1 (SCISR1) with RDRF set and then reading SCI data register low (SCIDRL).

1 = Received data available in SCI data register
0 = Data not available in SCI data register

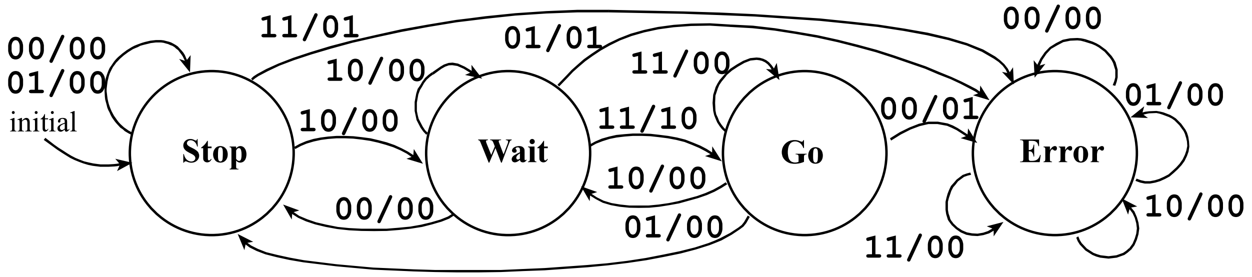
The **`SCIDRL`** register serial input/output data. Write a subroutine that inputs one ASCII character from the serial port. Your subroutine should wait for new input using gadfly synchronization. The subroutine uses return by value parameter passing with RegB. You don't need to write the initialization ritual. COMMENTS WILL BE GRADED.

```
SCISR1   equ $00CC  ; SCI Status Register 1
SCIDRL   equ $00CF  ; SCI Data Register Low
```

(20) Question 13. The objective of this problem is to implement the following two-input two-output Mealy finite state machine. The state graph and ROM-based data structure are given. Your job is to write the entire main program, including reset vector, to run this machine. The FSM repeats over and over the following sequence

1) Input from PTT;
2) Output to PTM (depends on input and state);
3) Change states (depends on input and state).

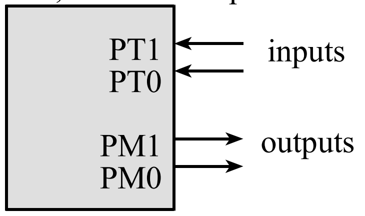
The initial state is **Stop**.



You will use the following linked data structure (without copying it to the answer sheet)

```
      org  $4000                   Put in ROM
Stop  fcb  0,0,0,1                 Outputs
      fdb  Stop,Stop,Wait,Error    Next states
Wait  fcb  0,1,0,2
      fdb  Stop,Error,Wait,Go
Go    fcb  1,0,0,0
      fdb  Error,Stop,Wait,Go
Error fcb  0,0,0,0
      fdb  Error,Error,Error,Error
```

The inputs are connected to PTT bits 1,0 and the outputs are connected to PTM bits 1,0.



You may use the following I/O port definitions (without copying them to the answer sheet)

```
DDRM equ $0252  ; Port M Direction
DDRT equ $0242  ; Port T Direction
PTM  equ $0250  ; Port M I/O Register
PTT  equ $0240  ; Port T I/O Register
```

postbyte,xb	syntax	mode	explanations
`rr000000`	`,r`	`IDX`	5-bit constant, n=0
`rr00nnnn`	`n,r`	`IDX`	5-bit constant, n=0 to +15
`rr01nnnn`	`-n,r`	`IDX`	5-bit constant, n=-16 to -1
`rr100nnn`	`n,+r`	`IDX`	pre-increment, n=1 to 8
`rr101nnn`	`n,-r`	`IDX`	pre-decrement, n=1 to 8
`rr110nnn`	`n,r+`	`IDX`	post-increment, n=1 to 8
`rr111nnn`	`n,r-`	`IDX`	post-decrement, n=1 to 8
`111rr100`	`A,r`	`IDX`	Reg A accumulator offset
`111rr101`	`B,r`	`IDX`	Reg B accumulator offset
`111rr110`	`D,r`	`IDX`	Reg D accumulator offset
`111rr000 ff`	`n,r`	`IDX1`	9-bit cons, n 16 to 255
`111rr001 ff`	`-n,r`	`IDX1`	9-bit const, n -256 to -16
`111rr010 eeff`	`n,r`	`IDX2`	16-bit const, any 16-bit n
`111rr111`	`[D,r]`	`[D,IDX]`	Reg D offset, indirect
`111rr011 eeff`	`[n,r]`	`[IDX2]`	16-bit constant, indirect

rr	register
00	X
01	Y
10	SP
11	PC

BRA

Operation: (PC) + $0002 + Rel ⇒ PC

Unconditional branch to an address calculated as shown in the expression. Rel is a relative offset stored as a two's complement number in the second byte of the branch instruction.

Source Form	Address Mode	Object Code
BRA *rel8*	REL	`20 rr`

LDX

Operation: (M : M+1) ⇒ X

Loads the most significant byte of index register X with the content of memory location M, and loads the least significant byte of X with the content of the next byte of memory at M+1.

Source Form	Address Mode	Object Code
LDX *#opr16i*	IMM	`CE jj kk`
LDX *opr8a*	DIR	`DE dd`
LDX *opr16a*	EXT	`FE hh ll`
LDX *oprx0_xysp*	IDX	`EE xb`
LDX *oprx9,xysp*	IDX1	`EE xb ff`
LDX *oprx16,xysp*	IDX2	`EE xb ee ff`
LDX [D,*xysp*]	[D,IDX]	`EE xb`
LDX [*oprx16,xysp*]	[IDX2]	`EE xb ee ff`

LDAA

Operation: (M) ⇒ A

Loads the content of memory location M into accumulator A. The condition codes are set according to the data.

Source Form	Address Mode	Object Code
LDAA *#opr8i*	IMM	`86 ii`
LDAA *opr8a*	DIR	`96 dd`
LDAA *opr16a*	EXT	`B6 hh ll`
LDAA *oprx0_xysp*	IDX	`A6 xb`
LDAA *oprx9,xysp*	IDX1	`A6 xb ff`
LDAA *oprx16,xysp*	IDX2	`A6 xb ee ff`
LDAA [D,*xysp*]	[D,IDX]	`A6 xb`
LDAA [*oprx16,xysp*]	[IDX2]	`A6 xb ee ff`

STAA

Operation: (A) ⇒ M

Stores the content of accumulator A in memory location M. The content of A is unchanged.

Source Form	Address Mode	Object Code
STAA *opr8a*	DIR	`5A dd`
STAA *opr16a*	EXT	`7A hh ll`
STAA *oprx0_xysp*	IDX	`6A xb`
STAA *oprx9,xysp*	IDX1	`6A xb ff`
STAA *oprx16,xysp*	IDX2	`6A xb ee ff`
STAA [D,*xysp*]	[D,IDX]	`6A xb`
STAA [*oprx16,xysp*]	[IDX2]	`6A xb ee ff`

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

example	addressing mode	Effective Address
ldaa #u	immediate	none
ldaa u	direct	EA is 8-bit address (0 to 255)
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-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

Pseudo op	meaning
org org	Specific absolute address to put subsequent object code
= equ	Define a constant symbol
set	Define or redefine a constant symbol
dc.b db fcb byte	Allocate byte(s) of storage with initialized values
fcc	Create an ASCII string (no termination character)
dc.w dw fdb .word	Allocate word(s) of storage with initialized values
dc.l dl .long	Allocate 32-bit long word(s) of storage with initialized values
ds ds.b rmb .blkb	Allocate bytes of storage without initialization
ds.w .blkw	Allocate bytes of storage without initialization
ds.l .blkl	Allocate 32-bit words of storage without initialization

00 0,X 5b const	10 −16,X 5b const	20 1,+X pre-inc	30 1,X+ post-inc	40 0,Y 5b const	50 −16,Y 5b const	60 1,+Y pre-inc	70 1,Y+ post-inc	80 0,SP 5b const	90 −16,SP 5b const	A0 1,+SP pre-inc	B0 1,SP+ post-inc	C0 0,PC 5b const	D0 −16,PC 5b const	E0 n,X 9b const	F0 n,SP 9b const
01 1,X 5b const	11 −15,X 5b const	21 2,+X pre-inc	31 2,X+ post-inc	41 1,Y 5b const	51 −15,Y 5b const	61 2,+Y pre-inc	71 2,Y+ post-inc	81 1,SP 5b const	91 −15,SP 5b const	A1 2,+SP pre-inc	B1 2,SP+ post-inc	C1 1,PC 5b const	D1 −15,PC 5b const	E1 −n,X 9b const	F1 −n,SP 9b const
02 2,X 5b const	12 −14,X 5b const	22 3,+X pre-inc	32 3,X+ post-inc	42 2,Y 5b const	52 −14,Y 5b const	62 3,+Y pre-inc	72 3,Y+ post-inc	82 2,SP 5b const	92 −14,SP 5b const	A2 3,+SP pre-inc	B2 3,SP+ post-inc	C2 2,PC 5b const	D2 −14,PC 5b const	E2 n,X 16b const	F2 n,SP 16b const
03 3,X 5b const	13 −13,X 5b const	23 4,+X pre-inc	33 4,X+ post-inc	43 3,Y 5b const	53 −13,Y 5b const	63 4,+Y pre-inc	73 4,Y+ post-inc	83 3,SP 5b const	93 −13,SP 5b const	A3 4,+SP pre-inc	B3 4,SP+ post-inc	C3 3,PC 5b const	D3 −13,PC 5b const	E3 [n,X] 16b indr	F3 [n,SP] 16b indr
04 4,X 5b const	14 −12,X 5b const	24 5,+X pre-inc	34 5,X+ post-inc	44 4,Y 5b const	54 −12,Y 5b const	64 5,+Y pre-inc	74 5,Y+ post-inc	84 4,SP 5b const	94 −12,SP 5b const	A4 5,+SP pre-inc	B4 5,SP+ post-inc	C4 4,PC 5b const	D4 −12,PC 5b const	E4 A,X A offset	F4 A,SP A offset
05 5,X 5b const	15 −11,X 5b const	25 6,+X pre-inc	35 6,X+ post-inc	45 5,Y 5b const	55 −11,Y 5b const	65 6,+Y pre-inc	75 6,Y+ post-inc	85 5,SP 5b const	95 −11,SP 5b const	A5 6,+SP pre-inc	B5 6,SP+ post-inc	C5 5,PC 5b const	D5 −11,PC 5b const	E5 B,X B offset	F5 B,SP B offset
06 6,X 5b const	16 −10,X 5b const	26 7,+X pre-inc	36 7,X+ post-inc	46 6,Y 5b const	56 −10,Y 5b const	66 7,+Y pre-inc	76 7,Y+ post-inc	86 6,SP 5b const	96 −10,SP 5b const	A6 7,+SP pre-inc	B6 7,SP+ post-inc	C6 6,PC 5b const	D6 −10,PC 5b const	E6 D,X D offset	F6 D,SP D offset
07 7,X 5b const	17 −9,X 5b const	27 8,+X pre-inc	37 8,X+ post-inc	47 7,Y 5b const	57 −9,Y 5b const	67 8,+Y pre-inc	77 8,Y+ post-inc	87 7,SP 5b const	97 −9,SP 5b const	A7 8,+SP pre-inc	B7 8,SP+ post-inc	C7 7,PC 5b const	D7 −9,PC 5b const	E7 [D,X] D indirect	F7 [D,SP] D indirect
08 8,X 5b const	18 −8,X 5b const	28 8,−X pre-dec	38 8,X− post-dec	48 8,Y 5b const	58 −8,Y 5b const	68 8,−Y pre-dec	78 8,Y− post-dec	88 8,SP 5b const	98 −8,SP 5b const	A8 8,−SP pre-dec	B8 8,SP− post-dec	C8 8,PC 5b const	D8 −8,PC 5b const	E8 n,Y 9b const	F8 n,PC 9b const
09 9,X 5b const	19 −7,X 5b const	29 7,−X pre-dec	39 7,X− post-dec	49 9,Y 5b const	59 −7,Y 5b const	69 7,−Y pre-dec	79 7,Y− post-dec	89 9,SP 5b const	99 −7,SP 5b const	A9 7,−SP pre-dec	B9 7,SP− post-dec	C9 9,PC 5b const	D9 −7,PC 5b const	E9 −n,Y 9b const	F9 −n,PC 9b const
0A 10,X 5b const	1A −6,X 5b const	2A 6,−X pre-dec	3A 6,X− post-dec	4A 10,Y 5b const	5A −6,Y 5b const	6A 6,−Y pre-dec	7A 6,Y− post-dec	8A 10,SP 5b const	9A −6,SP 5b const	AA 6,−SP pre-dec	BA 6,SP− post-dec	CA 10,PC 5b const	DA −6,PC 5b const	EA n,Y 16b const	FA n,PC 16b const
0B 11,X 5b const	1B −5,X 5b const	2B 5,−X pre-dec	3B 5,X− post-dec	4B 11,Y 5b const	5B −5,Y 5b const	6B 5,−Y pre-dec	7B 5,Y− post-dec	8B 11,SP 5b const	9B −5,SP 5b const	AB 5,−SP pre-dec	BB 5,SP− post-dec	CB 11,PC 5b const	DB −5,PC 5b const	EB [n,Y] 16b indr	FB [n,PC] 16b indr
0C 12,X 5b const	1C −4,X 5b const	2C 4,−X pre-dec	3C 4,X− post-dec	4C 12,Y 5b const	5C −4,Y 5b const	6C 4,−Y pre-dec	7C 4,Y− post-dec	8C 12,SP 5b const	9C −4,SP 5b const	AC 4,−SP pre-dec	BC 4,SP− post-dec	CC 12,PC 5b const	DC −4,PC 5b const	EC A,Y A offset	FC A,PC A offset
0D 13,X 5b const	1D −3,X 5b const	2D 3,−X pre-dec	3D 3,X− post-dec	4D 13,Y 5b const	5D −3,Y 5b const	6D 3,−Y pre-dec	7D 3,Y− post-dec	8D 13,SP 5b const	9D −3,SP 5b const	AD 3,−SP pre-dec	BD 3,SP− post-dec	CD 13,PC 5b const	DD −3,PC 5b const	ED B,Y B offset	FD B,PC B offset
0E 14,X 5b const	1E −2,X 5b const	2E 2,−X pre-dec	3E 2,X− post-dec	4E 14,Y 5b const	5E −2,Y 5b const	6E 2,−Y pre-dec	7E 2,Y− post-dec	8E 14,SP 5b const	9E −2,SP 5b const	AE 2,−SP pre-dec	BE 2,SP− post-dec	CE 14,PC 5b const	DE −2,PC 5b const	EE D,Y D offset	FE D,PC D offset
0F 15,X 5b const	1F −1,X 5b const	2F 1,−X pre-dec	3F 1,X− post-dec	4F 15,Y 5b const	5F −1,Y 5b const	6F 1,−Y pre-dec	7F 1,Y− post-dec	8F 15,SP 5b const	9F −1,SP 5b const	AF 1,−SP pre-dec	BF 1,SP− post-dec	CF 15,PC 5b const	DF −1,PC 5b const	EF [D,Y] D indirect	FF [D,PC] D indirect