NOTICE OF CHANGES FROM THE FIRST PRINTING

Pages C-36 and C-37 have been changed in this printing to reflect improved clock rate specifications.

How to reach us:
USA / EUROPE: Motorola Literature Distribution;
P.O. Box 20912; Phoenix, Arizona 85036. 1–800–441–2447

MFAX: RMFAX0@email.sps.mot.com – TOUCHTONE (602) 244–6609
INTERNET: http://Design–NET.com

JAPAN: Nippon Motorola Ltd.; Tatsumi–SPD–JLDC, Toshikatsu Otsuki,
6F Seibu–Butsuryu–Center, 3–14–2 Tatsumi Koto-Ku, Tokyo 135, Japan.
03-3521–8315

HONG KONG: Motorola Semiconductors H.K. Ltd.;
8B Tai Ping Industrial Park, 51 Ting Kok Road, Tai Po, N.T., Hong Kong.
852–26629298

Order this document by DSP56300FM/AD

Appendix C BENCHMARK PROGRAMS

C-1 INTRODUCTION

The following benchmarks illustrate the source code syntax and programming techniques for the DSP56300 Core. The assembly language source is organized into 6 columns as shown below.

Label	Opcode	Operands	X Bus Data	Y Bus Data	Comment
FIR	MAC	X0,Y0,A	X:(R0)+,X0	Y:(R4)+,Y0	;Do each tap

The Label column is used for program entry points and end of loop indication. The Opcode column indicates the Data ALU, Address ALU or Program Controller operation to be performed. The Operands column specifies the operands to be used by the opcode. The X Bus Data specifies an optional data transfer over the X Bus and the addressing mode to be used. The Y Bus Data specifies an optional data transfer over the Y Bus and the addressing mode to be used. The Comment column is used for documentation purposes and does not affect the assembled code. The Opcode column must always be included in the source code.

C-2 SET OF BENCHMARKS

C-2.1 Real Multiply

$$c = a \times b$$

Opcode	Operands	X Bus Data	Y Bus Data	Comment	Prog wrds	Clock Cycles
move		x:(r0),x0	y:(r4),y0	;	1	1
mpyr	x0,y0,a			;	1	1
move		a,x:(r1)		;	1	2 i'lock
				Totals	3	4

C-2.2 N Real Multiplies

$$c(i) = a(i) \times b(i) \qquad i = 1, 2, \ldots, N$$

Memory map:

pointer	X mem	Y mem
r0	a(i)	
r4		b(i)
r1	c(i)	

```
                                                                 Prog    Clock
                                                                 wrds    Cycles
        move    #AADDR,r0
        move    #BADDR,r4
        move    #CADDR,r1
        move                x:(r0)+,x0    y:(r4)+,y0    ;        1       1
        mpyr    x0,y0,a     x:(r0)+,x0    y:(r4)+,y0    ;        1       1
        do      #N-1,end                                ;        2       5
        mpyr    x0,y0,a     a,x:(r1)+     y:(r4)+,y0    ;        1       1
        move                x:(r0)+,x0                  ;        1       1
end                                                     ;
        move                a,x:(r1)+                   ;        1       1
                                                        Totals   7       2N+8
```

C-2.3 Real Update

$$d = c + a \times b$$

```
                                                                 Prog    Clock
                                                                 wrds    Cycles
        move    #AADDR,r0
        move    #BADDR,r4
        move    #CADDR,r1
        move    #DADDR,r2
        move                x:(r0),x0     y:(r4),y0     ;        1       1
        move                x:(r1),a                    ;        1       1
        macr    x0,y0,a                                 ;        1       1
        move                a,x:(r2)                    ;        1       2 i'lock
                                                        Totals   4       5
```

C-2.4 N Real Updates

$$d(i) = c(i) + a(i) \times b(i) \qquad i = 1, 2, \ldots, N$$

Memory map:

pointer	X mem	Y mem
r0	a(i)	
r4		b(i)
r1	c(i)	
r5		d(i)

						Prog wrds	Clock Cycles
	move	#AADDR,r0					
	move	#BADDR,r4					
	move	#CADDR,r1					
	move	#DADDR,r5					
	move		x:(r0)+,x0	y:(r4)+,y0	;	1	1
	move		x:(r1)+,a		;	1	1
	move		x:(r1)+,b		;	1	1
	do	#N/2,end			;	2	5
	macr	x0,y0,a	x:(r0)+,x1	y:(r4)+,y1	;	1	1
	macr	x1,y1,b	x:(r0)+,x0	y:(r4)+,y0	;	1	1
	move	x:(r1)+,a	a,y:(r5)+		;	1	1
	move	x:(r1)+,b	b,y:(r5)+		;	1	1
end							
					Totals	9	2N+8

C-2.5 Real Correlation Or Convolution (FIR Filter)

$$c(n) = \sum_{i=0}^{N-1} [a(i) \times b(n-i)]$$

Memory map:

pointer	X mem	Y mem
r0	a(i)	
r4		b(i)

					Prog wrds	Clock Cycles
move	#AADDR,r0					
move	#BADDR,r4			;		
move	#N-1,m4			;		
move	m4,m0			;		
movep	y:input,y:(r4)			;	1	2
clr	a	x:(r0)+,x0	y:(r4)-,y0	;	1	1
rep	#N-1			;	1	5
mac	x0,y0,a	x:(r0)+,x0	y:(r4)-,y0	;	1	1
macr	x0,y0,a		(r4)+	;	1	1
movep	a,y:output			;	1	2 i'lock
				Totals	6	N+14

Memory map:

pointer	X mem	Y mem
r0	a(i)	
r1	b(i)	

					Prog wrds	Clock Cycles
	move	#AADDR,r0				
	move	#BADDR,r1		;		
	move	#N-1,m1		;		
	move	m1,m0		;		
	movep	y:input,x:(r1)		;	1	2
	clr	a	x:(r0)+,x1	;	1	1
	do	#N-1,end		;	2	5
	move		x:(r1)-,x0	;	1	1
	mac	x0,x1,a	x:(r0)+,x1	;	1	1
end				;		
	move		x:(r1)-,x0	;	1	1
	macr	x0,x1,a	(r1)+	;	1	1
	movep	a,y:output		;	1	2 i'lock
				Totals	9	2N+10

C-2.6 Real * Complex Correlation Or Convolution (FIR Filter)

$$cr(n) = jci(n) = \sum_{i=0}^{N-1} [(ar(i) + jai(i)) \times b(n-i)]$$

$$cr(n) = \sum_{i=0}^{N-1} ar(i) \times b(n-i) \qquad ci(n) = \sum_{i=0}^{N-1} ai(i) \times b(n-i)$$

Memory map:

pointer	X mem	Y mem
r0	ar(i)	ai(i)
r4	b(i)	
r1	cr(n)	ci(n)

						Prog wrds	Clock Cycles
	move	#AADDR,r0			;		
	move	#BADDR,r4			;		
	move	#CADDR,r1			;		
	move	#N-1,m4			;		
	move	m4,m0			;		
	movep	y:input,x:(r4)			;	1	2
	clr	a	x:(r0),x0		;	1	1
	clr	b	x:(r4)-,x1	y:(r0)+,y0	;	1	1
	do	#N-1,end			;	2	5
	mac	x0,x1,a	x:(r0),x0		;	1	1
	mac	y0,x1,b	x:(r4)-,x1	y:(r0)+,y0	;	1	1
end							
	macr	x0,x1,a			;	1	1
	macr	y0,x1,b	(r4)+		;	1	1
	move		a,x:(r1)		;	1	1
	move			b,y:(r1)	;	1	1
					Totals	11	2N+11

C-2.7 Complex Multiply

$$cr + jci = (ar + jai) \times (br + jbi)$$

$$cr = ar \times br - ai \times bi \qquad ci = ar \times bi + ai \times br$$

Memory map:

pointer	X mem	Y mem
r0	ar	ai
r4	br	bi
r1	cr	ci

					Prog wrds	Clock Cycles
move	#AADDR,r0					
move	#BADDR,r4					
move	#CADDR,r1					
move		x:(r0),x1	y:(r4),y0	;	1	1
mpy	y0,x1,b	x:(r4),x0	y:(r0),y1	;	1	1
macr	x0,y1,b			;	1	1
mpy	x0,x1,a			;	1	1
macr	-y0,y1,a		b,y:(r1)	;	1	1
move		a,x:(r1)		;	1	2 i'lock
				Totals	6	7

C-2.8 N Complex Multiplies

$$cr(i) + jci(i) = (ar(i) + jai(i)) \times (br(i) + jbi(i)) \qquad i = 1, 2, \ldots, N$$

$$cr(i) = ar(i) \times br(i) - ai(i) \times bi(i)$$

$$ci(i) = ar(i) \times bi(i) + ai(i) \times br(i)$$

Memory map:

pointer	X mem	Y mem
r0	ar(i)	ai(i)
r4	br(i)	bi(i)
r5	cr(i)	ci(i)

					Prog wrds	Clock Cycles
move	#AADDR,r0			;		
move	#BADDR,r4			;		
move	#CADDR-1,r5			;		
move		x:(r0),x1	y:(r4),y0	;	1	1
move		x:(r5),a		;	1	1
do	#N,end			;	2	5
mpy	y0,x1,b	x:(r4)+,x0	y:(r0)+,y1	;	1	1
macr	x0,y1,b	a,x:(r5)+		;	1	1
mpy	-y0,y1,a		y:(r4),y0	;	1	1
macr	x0,x1,a	x:(r0),x1	b,y:(r5)	;	1	1
end						
move	a,x:(r5)			;	1	2 i'lock
				Totals	9	4N+9

C-2.9 Complex Update

$$dr + jdi = (cr + jci) + (ar + jai) \times (br + jbi)$$

$$dr = cr + ar \times br - ai \times bi \qquad di = ci + ar \times bi + ai \times br$$

Memory map:

pointer	X mem	Y mem
r0	ar	ai
r4	br	bi
r1	cr	ci
r2	dr	di

					Prog wrds	Clock Cycles
move	#AADDR,r0					
move	#BADDR,r4					
move	#CADDR,r1					
move	#DADDR,r2					
move			y:(r1),b	;	1	1
move		x:(r0),x1	y:(r4),y0	;	1	1
mac	y0,x1,b	x:(r4),x0	y:(r0),y1	;	1	1
macr	x0,y1,b	x:(r1),a		;	1	1
mac	x0,x1,a			;	1	1
macr	-y0,y1,a		b,y:(r2)	;	1	1
move		a,x:(r2)		;	1	2 i'lock
				Totals	7	8

C-2.10 N Complex Updates

$$dr(i) + jdi(i) = (cr(i) + jci(i)) + (ar(i) + jai(i)) \times (br(i) + jbi(i))$$
$$dr(i) = cr(i) + ar(i) \times br(i) - ai(i) \times bi(i)$$
$$di(i) = ci(i) + ar(i) \times bi(i) + ai(i) \times br(i)$$
$$i = 1, 2, \ldots, N$$

Memory map:

pointer	X mem	Y mem
r0	ar(i) ; ai(i)	
r4		br(i) ; bi(i)
r1	cr(i) ; ci(i)	
r5		dr(i) ; di(i)

```
                                                          Prog   Clock
                                                          wrds   Cycles
        move  #AADDR,r0                          ;
        move  #BADDR,r4                          ;
        move  #CADDR,r1                          ;
        move  #DADDR-1,r5                        ;
        move             x:(r0)+,x1  y:(r4)+,y0  ;        1      1
        move             x:(r1)+,b   y:(r5),a    ;        1      1
        do    #N,end     ;25                     ;        2      5
        mac   y0,x1,b    x:(r0)+,x0  y:(r4)+,y1  ;        1      1
        macr  -x0,y1,b   x:(r1)+,a   a,y:(r5)+   ;        1      1
        mac   x0,y0,a    x:(r1)+,b   b,y:(r5)+   ;        1      2 i'lock
        macr  x1,y1,a    x:(r0)+,x1  y:(r4)+,y0  ;        1      1
end
        move                         a,y:(r5)+   ;        1      2 i'lock
                                                 Totals   9      5N+9
```

Memory map:

pointer	X mem	Y mem
r0	ar(i)	ai(i)
r4	br(i)	bi(i)
r1	cr(i)	ci(i)
r5	dr(i)	di(i)

```
                                                              Prog   Clock
                                                              wrds   Cycles
        move    #AADDR,r0                           ;
        move    #BADDR,r4                           ;
        move    #CADDR,r1                           ;
        move    #DADDR-1,r5                         ;
        move                x:(r5),a                ;         1      1
        move                x:(r0),x1    y:(r4),y0  ;         1      1
        move                x:(r4)+,x0   y:(r1),b   ;         1      1
        do      #N,end                              ;         2      5
        mac     y0,x1,b     a,x:(r5)+    y:(r0)+,y1 ;         1      1
        macr    x0,y1,b     x:(r1)+,a               ;         1      1
        mac     -y0,y1,a    y:(r4),y0               ;         1      1
        macr    x0,x1,a     x:(r0),x1    b,y:(r5)   ;         1      1
        move                x:(r4)+,x0   y:(r1),b   ;         1      1
end
        move                a,x:(r5)                ;         1      1
                                                    Totals    11     5N+9
```

C-2.11 Complex Correlation Or Convolution (FIR Filter)

$$cr(n) + jci(n) = \sum_{i=0}^{N-1} [(ar(i) + jai(i)) \times (br(n-i) + jbi(n-i))]$$

$$cr(n) = \sum_{i=0}^{N-1} [ar(i) \times br(n-i) - ai(i) \times bi(n-i)]$$

$$ci(n) = \sum_{i=0}^{N-1} [ar(i) \times bi(n-i) + ai(i) \times br(n-i)]$$

Memory map:

pointer	X mem	Y mem
r0	ar(i)	ai(i)
r4	br(i)	bi(i)
r1	cr(i)	ci(i)

```
                                                              Prog    Clock
                                                              wrds    Cycles
        move   #AADDR,r0                            ;
        move   #BADDR,r4                            ;
        move   #CADDR,r1
        move   #N-1,m4
        move   #m4,m0
        movep  y:input,x:(r4)                                 1       2
        movep  y:input,y:(r4)                                 1       2
        clr    a                                    ;         1       1
        clr    b          x:(r0),x1    y:(r4),y0    ;         1       1
        do     #N-1,end                             ;         2       5
        mac    y0,x1,b    x:(r4)-,x0   y:(r0)+,y1   ;         1       1
        mac    x0,y1,b                              ;         1       1
        mac    x0,x1,a                              ;         1       1
        mac    -y0,y1,a   x:(r0),x1    y:(r4),y0    ;         1       1
end
        mac    y0,x1,b    x:(r4),x0    y:(r0)+,y1   ;         1       1
        macr   x0,y1,b                              ;         1       1
        mac    x0,x1,a                              ;         1       1
        macr   -y0,y1,a                             ;         1       1
        move                           b,y:(r1)     ;         1       1
        move              a,x:(r1)                  ;         1       1
                                                    Totals    16      4N+13
```

C-2.12 Nth Order Power Series (Real)

$$c = \sum_{i=0}^{N-1} [a(i) \times b^i]$$

Memory map:

pointer	X mem	Y mem
r0	a(i)	
r4		b
r1	c	

```
                                                             Prog    Clock
                                                             wrds    Cycles
      move    #AADDR,r0                             ;
      move    #BADDR,r4
      move    #CADDR,r1
      move               x:(r0)+,a                  ;        1       1
      move                          y:(r4),x0                1       1
      mpyr    x0,x0,b    x:(r0)+,y0                 ;        1       1
      move                          b,y1            ;        1       2 i'lock
      do      #N-1,end                              ;        2       5
      mac     y0,x0,a    x:(r0)+,y0                 ;        1       1
      mpyr    x0,y1,b    b,x0                       ;        1       1
end
      macr    y0,x0,a                               ;        1       1
      move               a,x:(r1)                   ;        1       2 i'lock
                                                    Totals   10      2N+11
```

C-2.13 2nd Order Real Biquad IIR Filter

$$w(n)/2 = x(n)/2 - (a1)/2 \times w(n-1) - (a2)/2 \times w(n-2)$$
$$y(n)/2 = w(n)/2 + (b1)/2 \times w(n-1) + (b2)/2 \times w(n-2)$$

Memory map:

pointer	X mem	Y mem
r0	w(n-2), w(n-1)	
r4		a2/2, a1/2, b2/2, b1/2

					Prog wrds	Clock Cycles
ori	#$08,mr			;		
move	#AADDR,r0			;		
move	#BADDR,r4			;		
move	#1,m0					
move	#3,m4					
movep	y:input,a			;	1	1
rnd	a	x:(r0)+,x0	y:(r4)+,y0	;	1	1
mac	-y0,x0,a	x:(r0)-,x1	y:(r4)+,y0	;	1	1
mac	-y0,x1,a	x1,x:(r0)+	y:(r4)+,y0	;	1	1
mac	y0,x0,a	a,x:(r0)	y:(r4),y0	;	1	2 i'lock
macr	y0,x1,a			;	1	1
movep	a,y:output			;	1	2 i'lock
				Totals	7	9

C-2.14 N Cascaded Real Biquad IIR Filter

$$w(n)/2 = x(n)/2 - (a1)/2 \times w(n-1) - (a2)/2 \times w(n-2)$$

$$y(n)/2 = w(n)/2 + (b1)/2 \times w(n-1) + (b2)/2 \times w(n-2)$$

Memory map:

pointer	X mem	Y mem
r0	w(n-2)1, w(n-1)1, w(n-2)2, ...	
r4		(a2/2)1, (a1/2)1, (b2/2)1, (b1/2)1, (a2/2)2, ...

```
                                                         Prog   Clock
                                                         wrds   Cycles
        ori     #$08,mr                             ;
        move    #AADDR,r0                           ;
        move    #BADDR,r4                           ;
        move    #(2N-1),m0                          ;
        move    #(4N-1),m4                          ;
        move                x:(r0)+,x0  y:(r4)+,y0  ;    1      1
        movep   y:input,a                           ;    1      1
        do      #N,end                              ;    2      5
        mac     -y0,x0,a    x:(r0)-,x1  y:(r4)+,y0  ;    1      1
        mac     -y0,x1,a    x1,x:(r0)+  y:(r4)+,y0  ;    1      1
        mac     y0,x0,a     a,x:(r0)+   y:(r4)+,y0  ;    1      2 i'lock
        mac     y0,x1,a     x:(r0)+,x0  y:(r4)+,y0  ;    1      1
end
        rnd     a                                   ;    1      1
        movep   a,y:output                          ;    1      2 i'lock
                                              Totals     10     5N+10
```

C-2.15 N Radix-2 FFT Butterflies (DIT, in-place algorithm)

$$ar' = ar + cr \times br - ci \times bi \qquad br' = ar - cr \times br + ci \times bi = 2 \times ar - ar'$$

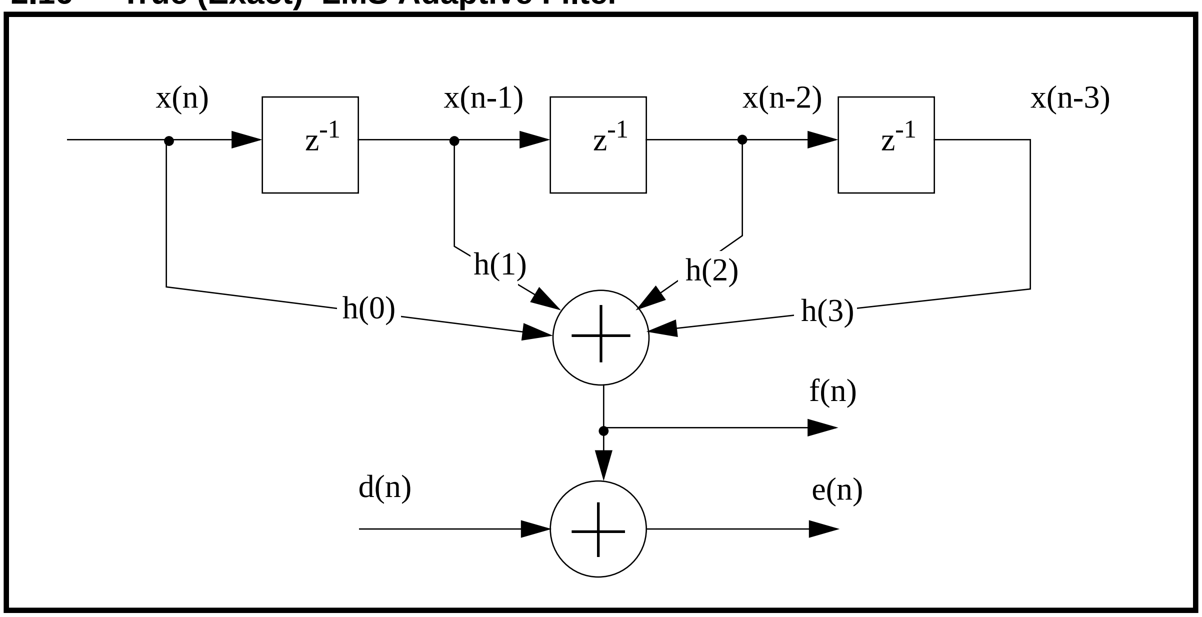
$$ai' = ai + ci \times br + cr \times bi \qquad bi' = ai - ci \times br - cr \times bi = 2 \times ai - ai'$$

Memory map:

pointer	X mem	Y mem
r0	ar(i)	ai(i)
r1	br(i)	bi(i)
r6	cr(i)	ci(i)
r4	ar’(i)	ai’(i)
r5	br’(i)	bi’(i)

					Prog wrds	Clock Cycles
move	#AADDR,r0			;		
move	#BADDR,r1			;		
move	#CADDR,r6			;		
move	#ATADDR,r4			;		
move	#BTADDR-1,r5			;		
move		x:(r1),x1	y:(r6),y0	;	1	1
move		x:(r5),a	y:(r0),b		1	1
do	#N,end			;	2	5
mac	y0,x1,b	x:(r6)+n,x0	y:(r1)+,y1	;	1	1
macr	x0,y1,b	a,x:(r5)+	y:(r0),a	;	1	1
subl	b,a			;	1	1
move		x:(r0),b	b,y:(r4)	;	1	1
mac	x0,x1,b	x:(r0)+,a	a,y:(r5)	;	1	1
macr	-y0,y1,b	x:(r1),x1	y:(r6),y0	;	1	1
subl	b,a	b,x:(r4)+	y:(r0),b	;	1	2 i'lock
end						
move		a,x:(r5)+		;	1	2 i'lock
				Totals	12	8N+9

C-2.16 True (Exact) LMS Adaptive Filter



Notation and symbols:

x(n)	-	Input sample at time n.
d(n)	-	Desired signal at time n.
f(n)	-	FIR filter output at time n.
H(n)	-	Filter coefficient vector at time n. H={h0,h1,h2,h3}
X(n)	-	Filter state variable vector at time N, X={x(n),x(n-1),x(n-2),x(n-3)}.
u	-	Adaptation gain.
NTAPS	-	Number of coefficient taps in the filter. For this example, ntaps=4.

System equations:

True LMS Algorithm	Delayed LMS Algorithm
e(n)=d(n)-H(n)X(n)	e(n)=d(n)-H(n)X(n)
H(n+1)=H(n)+uX(n)e(n)	H(n+1)=H(n)+uX(n-1)e(n-1)

LMS Algorithm:

True LMS Algorithm	Delayed LMS Algorithm
Get input sample	Get input sample
Save input sample	Save input sample
Do FIR	Do FIR
Get d(n), find e(n)	Update coefficients
Update coefficients	Get d(n), find e(n)
Output f(n)	Output f(n)
Shift vector X	Shift vector X

Memory map:

pointer	X mem	Y mem
r0	x(n), x(n-1), x(n-2), x(n-3)	
r4, r5		h(0), h(1), h(2), h(3)

```
                                                              Prog    Clock
                                                              wrds    Cycles
        move    #-2,n0                                  ;
        move    n0,n4
        move    #NTAPS-1,m0                             ;
        move    m0,m4                                   ;
        move    m0,m5                                   ;
        move    #AADDR+NTAPS-1,r0                       ;
        move    #BADDR,r4                               ;
        move    r4,r5                                   ;
_getsmp
        movep   y:input,x0                 ;get input sample  1       1
        clr     a           x0,x:(r0)+     y:(r4)+,y0   ;save 1       1
                                                        ;X(n), get h0
        rep     #NTAPS-1                                ;do fir 1     5
                                                        ;do taps
        mac     x0,y0,b     x:(r0)+,x0     y:(r4)+,y0   ;     1       1
                                                        ;last tap
        macr    x0,y0,b                                 ;     1       1
;Get d(n), subtract fir output, multiply by "u",
;put the result in y1.
;This section is application dependent.
        move    x:(r0)+,x0  y:(r4)+,a                         1       1
        movep   b,y:output  ;output fir if desired            1       1
        move                y:(r4)+,b                         1       1
        do      #NTAPS/2,cup                            ;     2       5
        macr    x0,x1,a     x:(r0)+,x0     y:(r4)+,y0   ;     1       1
        macr    x0,x1,b     x:(r0)+,x0     y:(r4)+,y1   ;     1       1
        tfr     y0,a                       a,y:(r5)+          1       1
        tfr     y0,b                       b,y:(r5)+          1       1
cup
        move                x:(r0)+n0,x0   y:(r4)+n4,y0 ;     1       1
;continue looping (jmp _getsmp)
                                                        Total 15      3N+16
```

C-2.17 Delayed LMS Adaptive Filter

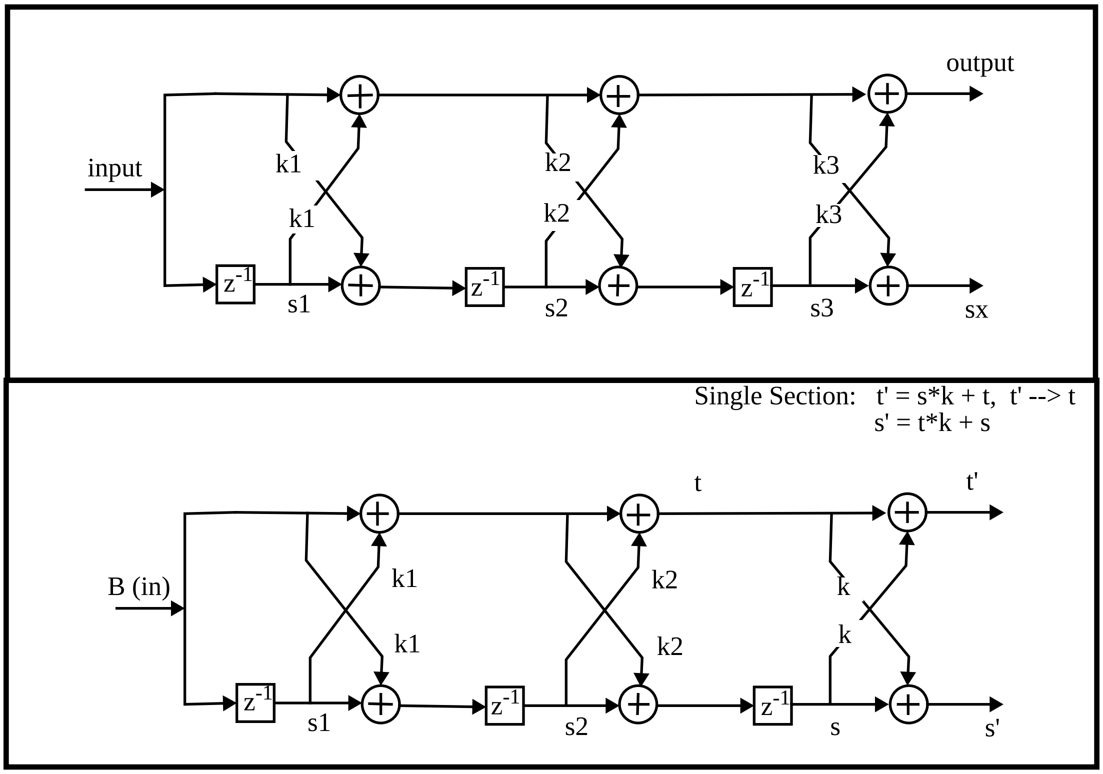
- error signal is in y1
- FIR sum in a = a + h(k)old*x(n-k)
- h(k)new in b = h(k)old + error*x(n-k-1)

Memory map:

pointer	X mem	Y mem
r0	x(n), x(n-1), x(n-2), x(n-3), x(n-4)	
r5, r4		dummy, h(0), h(1), h(2), h(3)

```
                                                                Prog   Clock
                                                                wrds   Cycles
      move    #STATE,r0                    ;start of X
      move    #2,n0                        ;used for pointer update
      move    #NTAPS,m0                    ;number of filter taps
      move    #COEF+1,r4                   ;start of H
      move    m0,m4                        ;number of filter taps
      move    #COEF,r5                     ;start of H-1
      move    m4,m5                        ;number of filter taps
      movep   y:input,a                    ;get input sample     1      1
      move    a,x:(r0)                     ;save input sample    1      1
      clr     a            x:(r0)+,x0      ;x0<-x(n)             1      1
      move                 x:(r0)+,x1      y:(r4)+,y0            1      1
                                           ;x1<-x(n-1); y0<-h(0)
      do      #TAPS/2,lms                            ;           2      5
      ;a<-h(0)*x(n) b<-h(0) Y<-dummy
      mac     x0,y0,a      y0,b            b,y:(r5)+             1      2 i'lock
      ;b<-H(0)=h(0)+e*x(n-1), x0<-x(n-2), y0<-h(1)
      macr    x1,y1,b      x:(r0)+,x0      y:(r4)+,y0   ;        1      1
      ;a<-a+h(1)*x(n-1); b<-h(1); Y(0)<-H(0)
      mac     x1,y0,a      y0,b            b,y:(r5)+    ;        1      2 i'lock
      ;b<-H(1)=h(1)+e*x(n-2); x1<-x(n-3); y0<-h(2)
      macr    x0,y1,b      x:(r0)+,x1      y:(r4)+,y0   ;        1      1
lms
      movep   a,y:output                                         1      1
      move    b,y:(r5)+                    ;Y<-last coef         1      1
      move    (r0)-n0                      ;update pointer       1      1
                                                  Totals        13      3N+12
```

C-2.18 FIR Lattice Filter

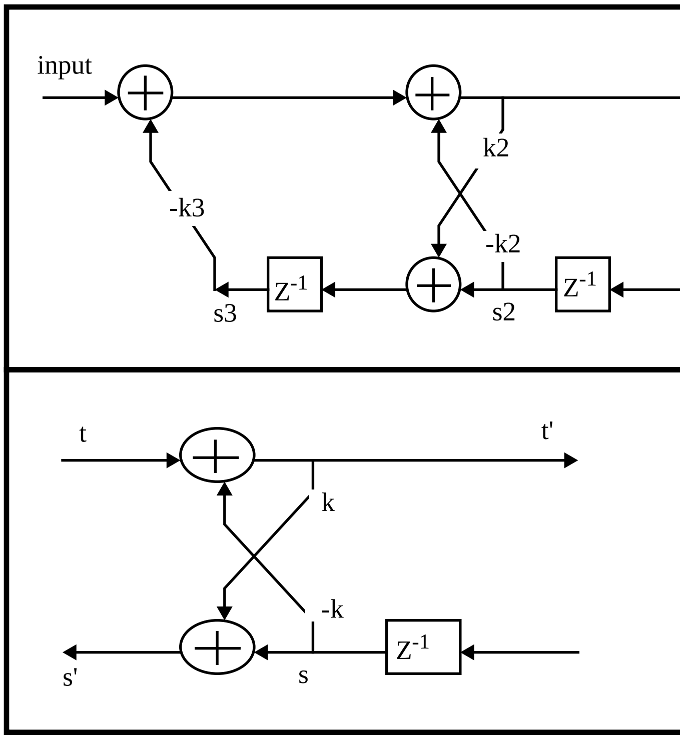


Memory map:

pointer	X mem	Y mem
r0	s1, s2, s3, sx	
r4		k1, k2, k3

```
                                                                     Prog   Clock
                                                                     wrds   cycles
        move    #S,r0                       ;point to s
        move    #N,m0                       ;N=number of k coefficients
        move    #K,r4                       ;point to k coefficients
        move     #N-1,m4                    ;mod for k's
        movep   y:datin,b                   ;get input                  1      1
        move     b,a                        ;save first state           1      1
        move                x:(r0),x0   y:(r4)+,y0  ;get s, get k        1      1
        do      #N,_elat                            ;                    2      5
        macr    x0,y0,b                 b,y1        ;s*k+t,copy t for mul 1     1
        tfr     x0,a        a,x:(r0)+               ;save s', copy next s 1     1
        macr    y1,y0,a     x:(r0),x0   y:(r4)+,y0  ;t*k+s, get s, get k 1      1
_elat
        move                a,x:(r0)+    y:(r4)-,y0 ;adj r4,dummy load   1      1
        movep   b,y:datout                          ;output sample       1      1
                                                    Totals               10     3N+10
```

C-2.19 All Pole IIR Lattice Filter



Single section:

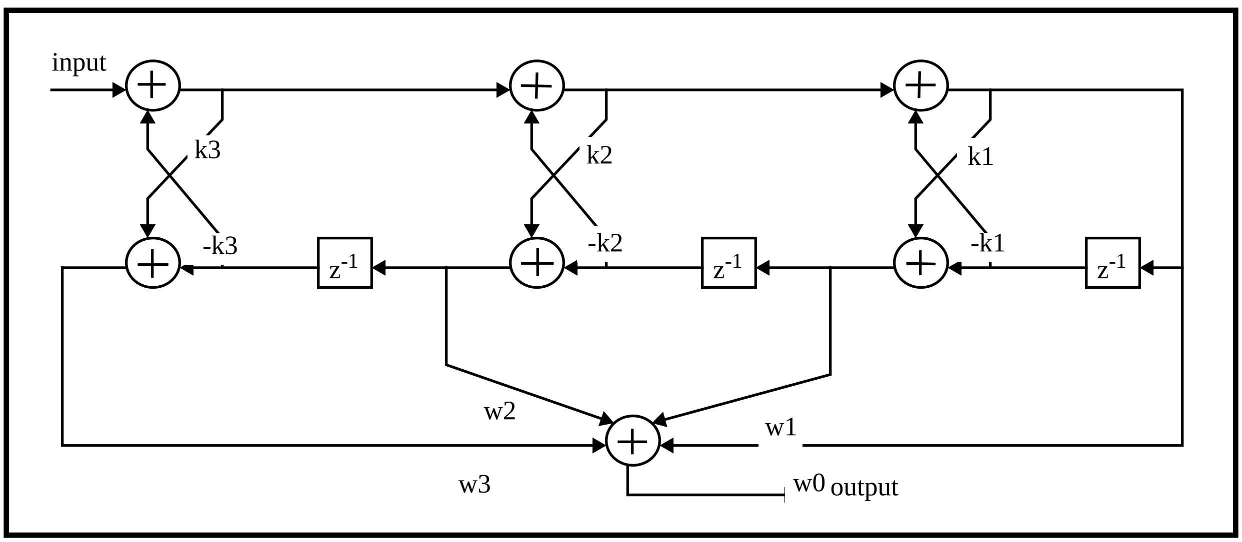
```
t' = t - k*s
s' = s + k*t'
t' --> t
```

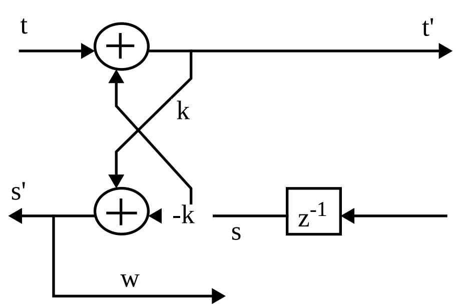
Memory map:

pointer	X mem	Y mem
r0	k3, k2, k1	
r4		s3, s2, s1

```
                                                                   Prog   Clock
                                                                   wrds   Cycles
        move    #k+N-1,r0                        ;point to k
        move    #N-1,m0                          ;number of k's-1
        move    #STATE,r4              ;point to filter states
        move    m0,m4                            ;mod for states
        move    #1,n4                            ;
        movep   y:datin,a              y:(r4)+,b ;get input         1      1
        move              x:(r0)-,x0   y:(r4)+,y0 ;get s, get k     1      1
        macr    -x0,y0,a  x:(r0)-,x0   y:(r4),y0 ;s*k+t             1      1
        do      #N-1,_endlat                     ;do sections       2      5
        macr    -x0,y0,a               y:(r4)+,y1 ;                 1      1
        tfr     y1,b      a,x1         b,y:(r4)  ;                  1      2 i'lock
        macr    x1,x0,b   x:(r0)-,x0   y:(r4),y0                    1      1
_endlat
        movep   a,y:datout                                          1      1
        move              x:(r0)+,x0   y:(r4)+,r0 ;output sample    1      1
        move    b,y:(r4)+                        ;save s'           1      1
;save last s', update r4
        move              a,y:(r4)                                  1      1
                                                 Totals             12     4N+8
```

C-2.20 General Lattice Filter





Single section:

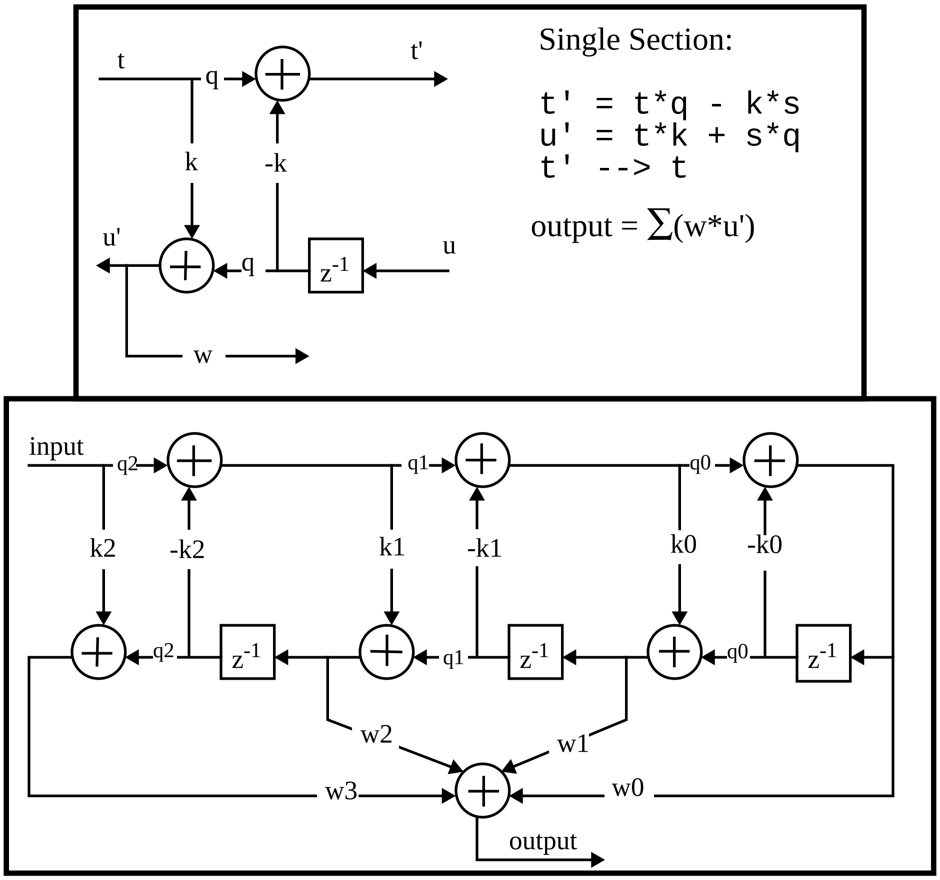
t' = t - k*s
s' = s + k*t'
t' --> t
output = $\sum$(w*s')

Memory map:

pointer	X mem	Y mem
r0	k3, k2, k1, w3, w2, w1, w0	
r4		s4, s3, s2, s1

```
                                                                      Prog  Clock
                                                                      wrds  Cycles
        move     #K,r0                     ;point to coefficients
        move    #2*N,m0                    ;mod 2*(# of k's)+1
        move    #STATE,r4                  ;point to filter states
        move    #-2,n4
        move    #N,m4                      ;mod on filter states
        movep   y:datin,a                  ;get input                  1     1
        move                x:(r0)+,x0  y:(r4)-,y0                     1     1
        do      #N,_endlat                                             2     5
        macr     -x0,y0,a                             ;                1     1
        tfr     y0,b        a,x1        b,y:(r4)+n4   ;                1     2 i'lock
        macr    x1,x0,b     x:(r0)+,x0  y:(r4)-,y0    ;                1     1
_endlat
        move                            b,y:(r4)+     ;save s'         1     2 i'lock
        clr     a                       a,y:(r4)+     ;save last s', update r4  1  1
        move                            y:(r4)+,y0                     1     1
        rep     #N                                    ;                1     5
        mac     x0,y0,a     x:(r0)+,x0  y:(r4)+,y0    ;s*w+out, get s, get w  1  1
        macr    x0,y0,a                               ;last mac        1     1
        movep   a,y:datout                            ;output sample   1     2 i'lock
                                                      Totals           14    5N+19
```

C-2.21 Normalized Lattice Filter



Memory map:

pointer	X mem	Y mem
r0	q2, k2, q1, k1, q0, k0, w3, w2, w1, w0	
r4		sx, s2, s1, s0

					Prog wrds	Clock Cycles
	move	#COEF,r0		;point to coefficients		
	move	#3*N,m0		;mod on coefficients		
	move	#STATE+1,r4		;point to state variables		
	move	#N,m4		;mod on filter states		
	movep	y:datin,y0		;get input sample	1	1
	move		x:(r0)+,x1	;get q in the table	1	1
	do	#N,_elat			2	5
	mpy	x1,y0,a	x:(r0)+,x0 y:(r4),y1	;q*t,get k,get s	1	1
	macr	-x0,y1,a	b,y:(r4)+	;q*t-k*s,save new s	1	1
	mpy	x0,y0,b		;k*t	1	1
	macr	x1,y1,b	x:(r0)+,x1 a,y0	;k*t+q*s,get next q,set t'	1	1
_elat						
	move	b,y:(r4)+		;save second last state	1	2 i'lock
	move	a,y:(r4)+		;save last state	1	1
	clr	a	y:(r4)+,y0	;clear a, get first state	1	1
	rep	#N			1	5
	mac	x1,y0,a	x:(r0)+,x1 y:(r4)+,y0	;fir taps	1	1
	macr	x1,y0,a	(r4)+	; round, adj pointer	1	1
	movep	a,y:datout		;output sample	1	2 i'lock
				Total	15	5N+19

C-2.22 [1x3][3x3] Matrix Multiplication

					Prog wrds	Clock Cycles
_init						
	move	#MAT_A,r0		;point to A matrix		
	move	#MAT_B,r4		;point to B matrix		
	move	#MAT_X,r1		;output X matrix		
	move	#2,m0		;mod 3		
	move	#8,m4		;mod 9		
	move	m0,m1		;mod 3		
_start						
	move	x:(r0)+,x0	y:(r4)+,y0		1	1
	mpy	x0,y0,a	x:(r0)+,x0 y:(r4)+,y0		1	1
	mac	x0,y0,a	x:(r0)+,x0 y:(r4)+,y0		1	1
	macr	x0,y0,a	x:(r0)+,x0 y:(r4)+,y0		1	1
	mpy	x0,y0,b	x:(r0)+,x0 y:(r4)+,y0		1	1
	move		a,y:(r1)+		1	1

	mac	x0,y0,b	x:(r0)+,x0	y:(r4)+,y0		1	1
	macr	x0,y0,b	x:(r0)+,x0	y:(r4)+,y0		1	1
	mpy	x0,y0,a	x:(r0)+,x0	y:(r4)+,y0		1	1
	move			b,y:(r1)+		1	1
	mac	x0,y0,a	x:(r0)+,x0	y:(r4)+,y0		1	1
	macr	x0,y0,a				1	1
	move			a,y:(r1)+		1	2 i’lock
_end							
					Totals	13	14

C-2.23 N Point 3x3 2-D FIR Convolution

The two dimensional FIR uses a [3x3] coefficient mask:

c(1,1) c(1,2) c(1,3)

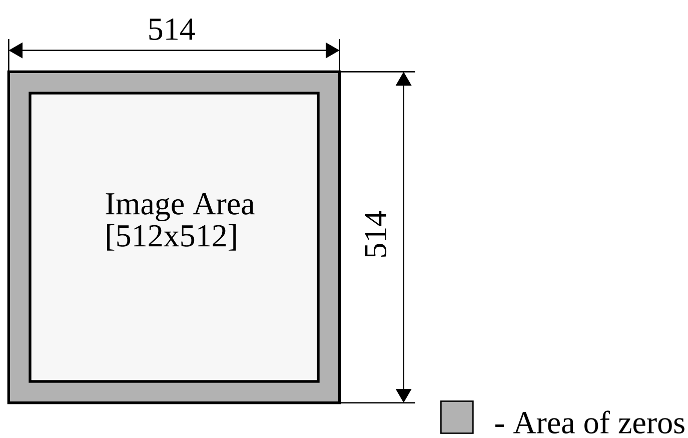
c(2,1) c(2,2) c(2,3)

c(3,1) c(3,2) c(3,3)

stored in Y memory in the order:

c(1,1), c(1,2), c(1,3), c(2,1), c(2,2), c(2,3), c(3,1), c(3,2), c(3,3).

The image is an array of 512x512 pixels. To provide boundary conditions for the FIR filtering, the image is surrounded by a set of zeros such that the image is actually stored as a 514x514 array. i.e.



The image (with boundary) is stored in row major storage. The first element of the array image(,) is image(1,1) followed by image(1,2). The last element of the first row is image(1,514) followed by the beginning of the next column image(2,1). These are stored sequentially in the array "im" in X memory:

Image(1,1) maps to index 0, image(1,514) maps to index 513;

Image(2,1) maps to index 514 (row major storage).

Although many other implementations are possible, this is a realistic type of image environment where the actual size of the image may not be an exact power of 2. Other possibilities include storing a 512x512 image but computing only a 511x511 result, computing a 512x512 result without boundary conditions but throwing away the pixels on the border, etc.

Memory map:

r0	-->	image(n,m) image(n,m+1) image(n,m+2)
r1	-->	image(n+514,m) image(n+514,m+1) image(n+514,m+2)
r2	-->	image(n+2*514,m) image(n+2*514,m+2) image(n+2*514,m+3)
r4	-->	FIR coefficients
r5	-->	output image

```
                                                                Prog   Clock
                                                                wrds   Cycles
    move    #MASK,r4                               ;point to coeffi-
                                                   cients
    move     #8,m4                                 ;mod 9
    move    #IMAGE,r0                              ;top boundary
    move    #IMAGE+514,r1                          ;left of first pixel
;left of first pixel 2nd row
    move    #IMAGE+2*514,r2                        ;
;adjust. for end of row
    move    #2,n1                                  ;
    move     n1,n2                                 ;
    move     #IMAGEOUT,r5                          ;output image
;first element, c(1,1)
    move                  x:(r0)+,x0  y:(r4)+,y0   ;               1      1
    do       #512,row                              ;               2      5
    do      #512,col                               ;               2      5
    mpy      x0,y0,a      x:(r0)+,x0  y:(r4)+,y0   ;c(1,2)         1      1
```

mac	x0,y0,a	x:(r0)-,x0	y:(r4)+,y0	;c(1,3)	1	1
mac	x0,y0,a	x:(r1)+,x0	y:(r4)+,y0	;c(2,1)	1	1
mac	x0,y0,a	x:(r1)+,x0	y:(r4)+,y0	;c(2,2)	1	1
mac	x0,y0,a	x:(r1)-,x0	y:(r4)+,y0	;c(2,3)	1	1
mac	x0,y0,a	x:(r2)+,x0	y:(r4)+,y0	;c(3,1)	1	1
mac	x0,y0,a	x:(r2)+,x0	y:(r4)+,y0	;c(3,2)	1	1
mac	x0,y0,a	x:(r2)-,x0	y:(r4)+,y0	;c(3,3)	1	1
;preload, get c(1,1)						
macr	x0,y0,a	x:(r0)+,x0	y:(r4)+,y0	;	1	1
;output image sample						
move			a,y:(r5)+	;	1	2 i'lock
col						
; adjust pointers for frame boundary						
;adj r0,r5 w/dummy loads						
move		x:(r0)+,x0	y:(r5)+,y1	;	1	1
;adj r1,r5 w/dummy loads						
move		x:(r1)+n1,x0	y:(r5)+,y1	;	1	1
;adj r2 (dummy load y1), preload x0 for next pass						
move		x:(r0)+,x0		;	1	1
move			y:(r2)+n2,y1	;	1	1
row						
			Total	19 (prog. words)	$11N^2+8N+7$ (clock cycles)	

C-2.24 Parsing data stream

This routine implements parsing of data stream for MPEG audio.
The data stream, composed by concatenated words of variable length, is allocated in consecutive memory words. The words lengths reside in another memory buffer.
The routine extracts words from data stream according to their length.
Two consecutive words are read from the stream buffer and are concatenated in the accumulator. Using bit offset and the specified length, a field of variable length can be extracted. The decision whether to load a new memory word into the accumulator from the stream is determined when bit offset overflow to the LSP of the accumulator.

The following describes the pointers and registers used by the routine:

- r0 - pointer to the buffer in X memory containing the variable length stream.
- r5 - pointer to buffer in Y memory where the length of each field is stored.
- r4 - pointer to a location that stores the "bits offset", number of bits left to be consumed. 48 initially.
- r3 - pointer to a location storing the constant 24.
- r1 - used as temporary storage (no need to initialize).

- y1 - stores the length of the field to be extracted.
- x0 - stores 24.

Memory map:

pointer	X mem	Y mem
r0	stream buffer	
r5		length buffer
r4		"bits offset"
r3	‘24‘	

```
init_                  ;this is the initialization code
          move    #stream_buffer,r0
          move    #length_buffer,r5
          move    #bits_offset,r4
          move    #boundary,r3
          move    #>48,b
          move    #>24,x0
          move                  x0,x:(r3)      b,y:(r4)

                                                                Prog    Clock
                                                                wrds    Cycles
Get_bits
                  ;bring length of next field and ‘24‘
          move                  x:(r3),x0      y:(r5)+,y1       1       1
                  ;bring word for parsing and "bits offset"
          move                  x:(r0)+,a      y:(r4),b         1       1
                  ;bring next word for parsing, point back to first word
          move                  x:(r0)-,a0                      1       1
                  ;calculate new "bits offset", r1 points to current word
          sub     y1,b          r0,r1                           1       1
                  ;save "bits offset" in x1
          move    b,x1                                          1       2
                  ;merge width and offset
          merge   y1,b                                          1       1
                  ;extract the field according to b, place it in a
          extract b1,a,a                                        1       1
                  ;restore "bits offset", r0 points to next word
          tfr     x1,b          (r0)+                           1       1
                  ;compare "bits offset" to 24, extracted word to a1
```

cmp	x0,b	a0,a		1	1
	;if "bits offset" is less or equal 24 another word is needed - update "bits offset" and point to next word				
add	x0,b	ifle		1	1
tgt		r1,r0		1	1
	;save "bits field" in memory				
move		b1,y:(r4)		1	1
			Totals	12	13

C-2.25 Creating data stream

This routine implements creation of data stream for MPEG audio.
Words of variable length are concatenated and stored in consecutive memory words.
The words for generating the stream are allocated in a memory buffer, and are aligned to the right. The words lengths reside in another memory buffer.
The word and its length are loaded for insertion. A word is read from the stream buffer into the accumulator. Using a bit offset and the specified length, a field of variable length is inserted into the accumulator. The accumulator is stored back containing the new concatenated field. The decision whether to read a new word from the stream is determined when bit offset overflow to the LSP of the accumulator.

The following describes the pointers and registers used by the routine:

- r0 - pointer to a buffer in X memory, containing the variable length codes. The code is right aligned at each location.
- r2 - pointer to a buffer in X memory containing the stream generated.
- r4 - pointer to a buffer in Y memory where the actual length of each field is stored.
- r3 - pointer to a location that stores the "bits offset", number of bits left to be consumed. 48 initially.
- r5 - pointer to a location storing the constant 24.
- r1 - used as temporary storage (no need to initialize).
- x0 - stores the current word to be inserted
- y1 - stores the length of the code brought in x0.
- y0 - stores 24.

Memory map:

pointer	X mem	Y mem
r0	data buffer	
r2	stream buffer	
r4		length buffer
r3		"bits offset"
r5		24

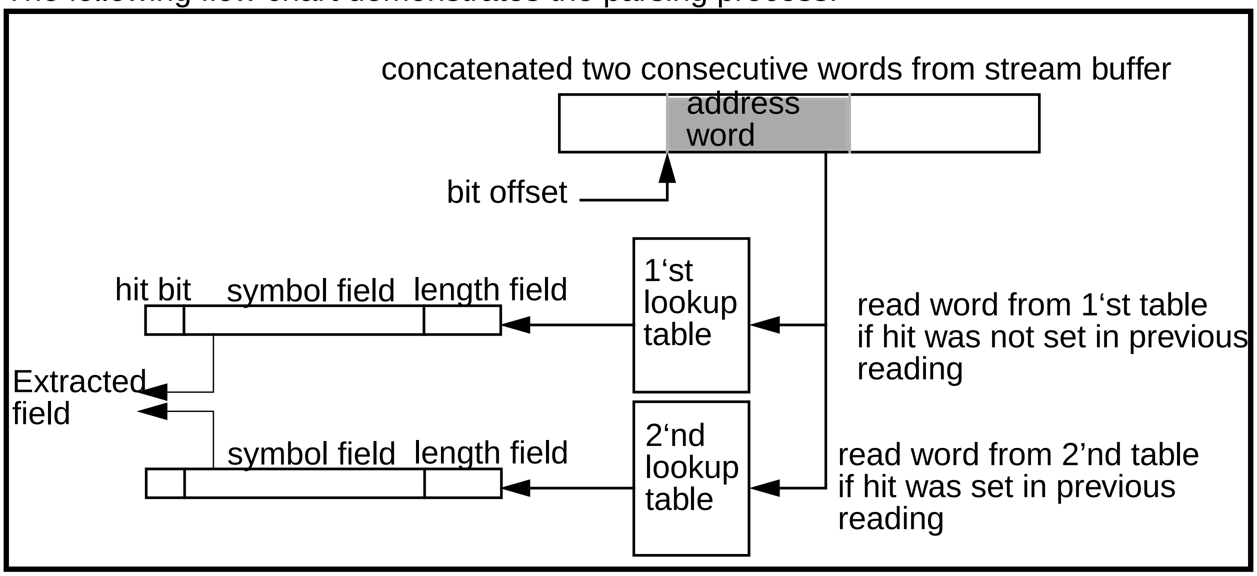
```
init_                ;this is the initialization code
        move    #data_buffer,r0
        move    #stream_buffer,r2
        move    #length_buffer,r4
        move    #bits_offset,r3
        move    #boundary,r5
        move    #>48,b
        move    #>24,y0
        move                 b,x:(r3)      y0,y:(r5)


                                                         Prog    Clock
                                                         wrds    Cycles
Put_bits
                ;bring code and its length
        move                 x:(r0)+,x0    y:(r4)+,y1    1       1
                ;bring "bits offset" and ‘24‘
        move                 x:(r3),b      y:(r5),y0     1       1
                ;calculate new "bits offset", bring current word from
                stream buffer
        sub     y1,b         x:(r2),a                    1       1
                ;save "bits offset" in x1
        move    b,x1                                     1       2
                ;merge width and offset
        merge   y1,b                                     1       1
                ;insert the field according to b, place it in a
        insert  b1,x0,a                                  1       1
                ;restore "bits offset", r1 points to current word
        tfr     x1,b         r2,r1                       1       1
                ;compare "bits offset " to 24, send new word to
                stream buffer
```

cmp	y0,b	a1,x:(r2)+		1	1
	;send a0 to next location in stream buffer in case of crossing boundary				
move		a0,x:(r2)		1	2
	;if "bits offset" is less or equal 24 then update "bits offset " and point to the next word in stream buffer				
add	y0,b	ifle		1	1
tgt		r1,r2		1	1
	;save "bits offset" in memory				
move		b1,y:(r4)		1	1
			Totals	12	14

C-2.26 Parsing Hoffman code data stream

This routine implements the parsing of Hoffman code data stream.
The routine extracts a bit field from the stream. Two consecutive words are brought to the accumulator from the stream buffer. An address word is extracted using a bit offset and a field length. The field length is determined by the number of bits needed by the address of the two Hoffman code lookup tables. A word is loaded from the first lookup table. If the hit bit in the word is not set then a field of variable length is extracted. The length of the extracted field is specified in the length field in the word. The bit offset is updated according to the length of the extracted word.
If the hit bit in the word is set then a new address word is read from the stream. A word is brought from the second lookup table. The bit field is extracted according to the same guidelines.
The following flow chart demonstrates the parsing process:



Thek following describes the pointers and registers used by the routine:

- r0 - pointer to the buffer in X memory containing the stream.
- r1 - used as temporary storage (no need to initialize).

- r3 - pointer to buffer in Y memory where the extracted fields are stored.
- r5 - pointer to a location that stores the "bits offset", number of bits left to be consumed. 48 initially.
- r2 - pointer to the right table.
- r6 - pointer to the first lookup table.
- r7 - pointer to the second lookup table.
- r4 - pointer to constants.

Memory map:

pointer	X mem	Y mem
r0	stream buffer	
r3	extracted data buffer	
r5		"bits offset"
r4		#no.1 address bus length
		#no.2 mask word for length field
		#no.3 merged width and offset
		‘24‘
r6	first lookup table	
r7	second lookup table	

```
init_                   ;this is the initialization code
            move    #stream_buffer,r0
            move    #data_buffer,r3
            move    #bits_offset,r5
            move    #constants,r4
            move    #first_table,r2
            move    #first_table,r6
            move    #second_table,r7
                    ;move constants to memory
            move    #>48,b
            move    b,y:(r5)
            move    #>3,n4
            move    #n0_1,y1
            move    y1,y:(r4)+
            move    #n0_2,y1
            move    y1,y:(r4)+
            move    #n0_3,y1
```

```
        move    y1,y:(r4)+
        move    #>24,y1
        move    y1,y:(r4)-n4


                                                                        Prog    Clock
                                                                        wrds    Cycles
Get_bits
                ;bring word from stream, and "bits-offset"
        move                    x:(r0)+,a       y:(r5)+,b               1       1
                ;bring next word from stream, and address length
        move                                    y:(r4)+,y0              1       1
        move                    x:(r0)-,a0                              1       1
                ;calculate new "bits offset", and save old one in x1
        sub     y0,b            b,x1                                    1       1
                ;merge width and offset
        merge   y0,b                                                    1       1
                ;extract the field according to b, place it in a
        extract b1,a,a                                                  1       1
                ;move address to n2
        move                    a0,n2                                   1       1
                ;bring mask for length field in tookup table words
        move                    y:(r4)+,y1                              1       1
                ;bring the merged offset and length for extactionf
        move                    y:(r4)+,x0                              1       1
                ;r1 points to current address for extracted field
        move                    r3,r1                                   1       1
                ;bring word from lookup table
        move                    x:(r2+n2),a                             1       1
                ;extract the field according to x0, place it in b
        extract x0,a,b                                                  1       1
                ;test if hit bit is set, r2 points s first lookup table
        tst     a               r6,r2                                   1       1
                ; if hit bit is set, r2 points second lookup table, a holds
                address length
        tmi     y0,a            r7,r2                                   1       1
                ;restore "bit offset" , send extracted field to memory
        tfr     x1,b            b0,x:(r3)+                              1       1
                ; if hit bit is set, restore r3
        tmi                     r1,r3                                   1       1
                ;mask length field , save pointer to current stream
                word
        and     y1,a            r0,r1                                   1       1
```

	;calculate new "bits offset", y1 holds ’24’				
sub	a,b	y:(r4)-n4,y1		1	1
	;compare "bits offset " to 24, update steam pointer				
cmp	y1,b	(r0)+		1	1
	;if "bits offset" is less or equal 24 another word is needed - update "bits offset " and point to next word				
add	y1,b	ifle		1	1
tgt		r1,r0		1	1
	;save "bits field" in memory				
move		b1,y:(r5)		1	1
			Totals	22	22

C-3 BENCHMARK OVERVIEW

Benchmark	Program Length in Words	Program Length in Clock Cycles	Sample Rate or Execution Time for 66MHz Clock Cycle	Sample Rate or Execution Time for 80MHz Clock Cycle
Real Multiply on page 3	3	4	61 ns	50 ns
N Real Multiplies on page 3	7	2N+8	30N+122 ns	25N+100 ns
Real Update on page 4	4	5	76 ns	62.5 ns
N Real Updates on page 4	9	2N+8	30N+122 ns	25N+100 ns
Real Correlation Or Convolution (FIR Filter) on page 5	6	N+14	66/(N+14) MHz	80/(N+14) MHz
Real * Complex Correlation Or Convolution (FIR Filter) on page 7	9	2N+10	33/(N+5) MHz	40/(N+5) MHz
Complex Multiply on page 7	6	7	106 ns	87.5 ns
N Complex Multiplies on page 8	9	5N+9	76N+137 ns	62.5N+113 ns
Complex Update on page 9	7	8	122 ns	100 ns
N Complex Updates on page 10	9	4N+9	61N+137 ns	50N+113 ns
Complex Correlation Or Convolution (FIR Filter) on page 11	16	4N+13	66/(4N+13) MHz	80/(4N+13) MHz
Nth Order Power Series (Real) on page 12	10	2N+11	30N+167 ns	25N+137ns
2nd Order Real Biquad IIR Filter on page 13	7	9	137 ns	113 ns
N Cascaded Real Biquad IIR Filter on page 14	10	5N+10	66/(5N+10) MHz	16/(N+2) MHz
N Radix-2 FFT Butterflies (DIT, in-place algorithm) on page 15	12	8N+9	122N+137 ns	100N+113 ns

Benchmark	Program Length in Words	Program Length in Clock Cycles	Sample Rate or Execution Time for 66MHz Clock Cycle	Sample Rate or Execution Time for 80MHz Clock Cycle
True (Exact) LMS Adaptive Filter on page 16	15	3N+16	66/(3N+16) MHz	80/(3N+16) MHz
Delayed LMS Adaptive Filter on page 19	13	3N+12	66/(3N+12) MHz	80/(3N+12) MHz
FIR Lattice Filter on page 20	10	3N+10	66/(3N+10) MHz	80/(3N+10) MHz
All Pole IIR Lattice Filter on page 21	12	4N+8	33/(2N+4) MHz	20/(N+2) MHz
General Lattice Filter on page 22	14	5N+19	66/(5N+19) MHz	80/(5N+19) MHz
Normalized Lattice Filter on page 24	15	5N+19	66/(5N+19) MHz	80/(5N+19) MHz
[1x3][3x3] Matrix Multiplication on page 25	13	14	213 ns	175 ns
N Point 3x3 2-D FIR Convolution on page 26	19	$11N^2+8N+7$	$66/(11N^2+8N+7)$ MHz	$80/(11N^2+8N+7)$ MHz