.title "cascade.sa"

.def \_cascadeSection

.def \_fixedMpy

.def \_cascade

.def \_fixedRound

.def \_float2Fixed

**.sect** ".fixed"

**.global** \_input\_sample

**.global** \_output\_sample

**.global** \_cascade

**.global** \_fixedMpy

**.global** \_fixedRound

**.global** \_dOffset

**.global** \_gain

**.global** \_filterSections

**.global** \_float2Fixed

\_filterSections .usect "arraySect", 160, 4 ;allocate int filterSections(5\*8)

**.global** \_dBuffer

\_dBuffer .usect "array2Sect", 80, 4 ;allocate int dBuffer(5\*3);

\_BITS **.set** 4 ;total bits including sign

\_sections **.set** 5

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Interrupt to perform a cascaded filter

\* using fixed point arithmetic

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**\_cascade:** .cproc

.reg x, filter, dBuff, dOffset, p\_dOffset, sections, p\_filter, p\_dBuff, i

.reg count, product, result, gain, mask, inputLR, shift, temp

mvkl \_dBuffer, dBuff

mvkh \_dBuffer, dBuff

mvkl \_filterSections, filter

mvkh \_filterSections, filter

mvkl \_dOffset, p\_dOffset

mvkh \_dOffset, p\_dOffset

mvkl \_sections, sections

mvkh \_sections, sections

mvkl \_gain, gain

mvkh \_gain, gain

ldw \*p\_dOffset, dOffset

ldw \*gain, gain

.call inputLR = \_input\_sample() ;obtain 2 int\_16

mvkl 0x0000FFFF, mask ;mask lower 16 bits

mvkh 0x0000FFFF, mask

**AND** mask, inputLR, x ;right channel (chan2)

mvk 15, shift

shl x, shift, x ;shift by 15 to preserve pos sign

.call x = \_fixedRound(x)

.call x = \_fixedMpy(x,gain) ;input gain

mvk 0, i

;loop over every section for cascade

**loop:**

mpy i, 4, count ; dBuff[i][4], i\*4

addaw dBuff, count, p\_dBuff ;&dBuff[i][0]

mpy i,8,count ;filterSections[i][8]

addaw filter, count, p\_filter ;&filterSections[i][0]

.call x = \_cascadeSection(x,p\_dBuff,dOffset,p\_filter) ;get output of single section

**add** i, 1, i

cmplt i, sections, count

[count] b loop

**add** dOffset, 1, dOffset

mvk 0x3, mask

**AND** dOffset, mask, dOffset ;(dOffset+1)%4

stw dOffset, \*p\_dOffset

shru x, shift, x ;shift into lower 16 bits

.call \_output\_sample(x) ;output sample

mvk 0x1, mask ;enable interrupt

mvc CSR, temp

**OR** temp, mask, temp

mvc temp, CSR

.return

.endproc

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Fixed point computation of a

\* cascaded section

\* @param x: input fixed point value

\* @param B4: pointer to dBuff array, used for circular buffering

\* @param dOffset: starting location for dBuff

\* @param filter: fixed point filter with format:

\* a2k, b2k, a1k, b1k, a0k, b0k, scalarShift(int)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**\_cascadeSection:** .cproc x, B4, dOffset, filter ;cascadeSection(x(n),\*dBuff(n=0),dOffset,filterCoef)

.reg p\_a, p\_b, a, b, d, product, dresult, yresult, count, scalarShift, oldAMR, sign, mask

.circ dBuff/B4

mvc AMR, oldAMR

mvkl 0x3<<16|0x1<<8, count

mvkh 0x3<<16|0x1<<8, count

mvc count,AMR ;B4 circular buffer size 4\*4

mvk 0,yresult

addaw dBuff, dOffset, dBuff ;shift dBuff to d(n)

ldw \*+filter[6], scalarShift ;load n value to shift by

subaw dBuff, 2, dBuff ;d[n-2]

mvk 0, dresult

mvk 2, count

**dLoop:** .trip 2 ;i=2;i>0;i--

lddw \*filter++, **b:**a

ldw \*dBuff++, d

.call product = \_fixedMpy(d,a)

**sub** dresult,product,dresult ;d-=d(n-i)\*a(i)

.call product = \_fixedMpy(d,b)

**add** product,yresult,yresult ;y+=d(n-i)\*b(i)

**sub** count, 1, count

[count] b dLoop

lddw \*filter++, **b:**a

.call product = \_fixedMpy(x, a)

**add** dresult, product, dresult ;d[n]=a(0)\*x(n)

shl dresult, scalarShift, dresult ;undo scaling

stw dresult, \*dBuff ;store d[0]

.call product = \_fixedMpy(dresult,b) ;d[0]\*b[0]

**add** product, yresult,yresult ;y+=d[0]\*b0

mvc oldAMR, AMR

.return yresult

.endproc

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Computes the fixed point multiplication.

\* Rounds the result to \_BITS

\* @param a: first fixed point argument

\* @param b: second fixed point argument

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\_fixedMpy .cproc a,b

.reg product

smpyh a,b,product ;do fixed multiply .call product = \_fixedRound(product)

.return product

.endproc

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Rounds the given value to a fixed \_BITS

\* @param value: fixed value to be rounded

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\_fixedRound .cproc value

.reg val, mask, product,bits

mvkl 0x80000000, val ;1 in the MSB

mvkh 0x80000000, val

mvk \_BITS, bits

shru val, \_BITS, val ;shift 1 to the bits+1 place

**add** value, val, product ;add 0b[0...0]1 where the 1 is at bits+1

mvkl 0xFFFFFFFF, mask

mvkh 0xFFFFFFFF, mask

mvk 32, val

**sub** val,bits,bits

shl mask, bits, mask ;mask upper b bits

**AND** product, mask, product

.return product

.endproc

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* converts a floating point number to a

\* fixed point signed number

\* @param floatNum: floating point number to convert

\* @param bits: number of bits for the desired fixed point number

\* (bits includes the sign bit)

\* @param maxval: max value used to scale the output

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\_float2Fixed .cproc floatNum, bits, maxVal

.reg floatTemp, float1, n, boolVal, div2, npow, temp, mask

mvkl 0x3f800000, float1 ;float 1.0f

mvkh 0x3f800000, float1

mvk 31, n ;number to shift by (2^n)

mvk 2, div2

intsp div2, div2

rcpsp div2, div2 ;float 1.0f/2

;find the 2^-n value to scale below 1

**GTOneLoop:**

cmpgtsp maxVal, float1, boolVal ;divides float by 2 until less than 1

[boolVal] sub n, 1, n ;decrement n until max<1.0f

[boolVal] mpysp maxVal, div2, maxVal

[boolVal] b GTOneLoop

;multiply by 2^(31-n) where n causes scaling

**sub** n, 1, n ;exponent format E=0b1001 1110 for 2^31

mvk 0x80, temp ;0b1000 0000

**OR** n, temp, n ;get into required exponent format

shl n, 23, npow ;shift n into exponent of float

mpysp floatNum, npow, floatNum ;mpy 2^(31-n) where n makes result < 1

;add 2^(31-b)

mvk 0x9E, n ;exponent format 0b0 1001 1110 (float 2^31)

**sub** n, bits, n ;31-b

shl n, 23, n ;shift n into exponent of float

addsp floatNum, n, floatNum ;add 2^(31-b)

spint floatNum, floatNum ;convert to int

;mask the (32-b) LSBs to 0

mvkl 0xFFFFFFFF, mask

mvkh 0xFFFFFFFF, mask

mvk 32, n

**sub** n,bits,bits

shl mask, bits, mask ;mask has b+1(signed) zeros in LSB

**AND** floatNum, mask, floatNum ;set lower b+1 bits to 0

.return floatNum

.endproc