Introduction

The DREAsm assembler is a command-line program which takes in a text file listing of an assembly program written for the DRE and parses it, does error checking, and outputs an object file suitable for inclusion for a microcontroller download into the chip. This manual gives a description of the various formats of parameters and statements recognized by the DREAsm assembler.

Table of Contents

Invoking the Assembler

To invoke the assembler from the command line, type:

DREAsm filename

where **filename** is the name of the DRE Assembly file you wish to assemble. The assembler will expand that name to "**filename.asm**". The hex output file's name will be "**filename.obj**".

General Information

DREAsm is not case-sensitive, ignores blank lines, and treats multiple tabs or spaces as a single space. Each instruction takes up one line.

Address Format

Addresses may be entered in hexadecimal or decimal format. To specify a hexadecimal address, prefix the number with a dollar sign (**$**) or "**0x**" or suffix it with an "**H**".

For example, the following instructions all specify the same address:

```
RZP   $4      K=0.5    ; Read address 4 times 0.5
RZP   004H    K=0.5    ; Read address 4 times 0.5
RZP   4       K=0.5    ; Read address 4 times 0.5
RZP   0x04    K=0.5    ; Read address 4 times 0.5
```

Coefficient Format

Coefficients may be entered in a variety of numerical formats, including binary, octal, hexadecimal, and real decimal. To specify a binary number, either use the percentage symbol prefix (**%**) or a "**B**" suffix. To specify an octal number, suffix the value with a "**Q**". To specify a hexadecimal number, either prefix the number with a dollar sign (**$**), prefix it with a "**0x**", or suffix it with an "**H**". To specify a real decimal number, just enter the number. A real decimal number can have a decimal point, and it can be prefixed with a plus (+) or minus (-) sign.

All MAC Instruction coefficients in the DRE are stored as fixed-point, sign-magnitude values in the format **S.7**. The **S** signifies the sign bit, the period signifies the binary point, and the **7** signifies the number of fractional bits. When a number is written as binary, octal, or hexadecimal, it is written as a right-justified number with no separation between the integer and fractional bits, and unspecified leading bits are assumed to be zero. When a number is written as a decimal value, however, it can be written the same way as the other radices, or with the decimal point corresponding to the binary point of the S.7 value.

For example, all of the following are ways of representing the number -0.34375 in S.7 format:

Binary:	%10101100	Hexadecimal:	$AC
	10101100B		0xAC
			ACH
Octal:	254Q	Decimal:	-0.34375
			-44
			172

A simple way to convert a decimal into a hex, octal, or binary coefficient suitable for the compiler is to shift the binary point to the rightmost position for the calculation in order to make it an integer. For example, to see what the decimal value 0.1796875 looks like in the S.7 hex, octal, or binary value for the compiler:

Multiply it by 2^7 (thus making the S.7 into S7):
0.1796875 x 2^7 = 0.1796875 x 128 = 23 = 17H = 27Q = 10111B

For the value -0.1796875:
-0.1796875 x 2^7 = -0.1796875 x 128 = -23 = 97H = 227Q = 10010111B
*Note that the value is trimmed to be 8 bits.

LFO Setup Instruction Format

The DRE has four low frequency oscillators (LFOs) which must be set up before use. These instructions may occur anywhere in the assembly file. To set the parameters (amplitude, frequency, and waveshape), use the LFO Setup Instruction, formatted as follows:

LFOn WAVESHAPE AMP=a FREQ=f [XFAD=x] [; COMMENTS]

Note: Square brackets "**[**" and "**]**" denote optional fields.

Select LFOs 0-3 by setting **n**. **WAVESHAPE** can be either "**SIN**" for sinusoid, "**TRI**" for triangle wave, or "**SAW**" for sawtooth. The **AMP**litude coefficient is a 15-bit unsigned number (a range of 0-32767 or $0-$7FFF), and gives a waveform amplitude of ±AMP/8 samples. The **FREQ**uency coefficient is a 13-bit unsigned number (a range of 0-8191 or $0-$1FFF). **XFAD** selects the crossfade shape to use for pitch transposition algorithms. This setting is only valid when the waveshape is set to **SAW**. Valid crossfade settings are: 1, 1/2, 1/8, 1/16.

To calculate the frequency produced by the oscillators, use the following formulae.

F(Sinusoids) = (f/M) Fs/(2π)

where
f = 13-bit frequency coefficient FREQ,
Fs = Sample rate,
M = 262143 (0x3FFFF, internal maximum 18 bit value).

For a sample rate of 48kHz, F(Sinusoids) = 0.029142 f

F(Triangles) = (f/M) Fs (C/4H)

where
f = 13-bit frequency coefficient FREQ,
Fs = Sample rate,
C = 4194304 (0x400000, a constant used internally in the LFO),
M = 262143 (0x3FFFF, internal maximum 18-bit value),
H = 8388607 (0x7FFFFF, maximum positive 24-bit value).

For a sample rate of 48kHz, F(Triangles) = 0.022888 f

F(Sawtooths) = 2 F(Triangles)

For a sample rate of 48kHz, F(Triangles) = 0. 045777 f

For example:

```
LFO0  SIN   AMP=0x3FFF  FREQ=0x200              ; 14.9Hz, 1/2 amplitude sine
LFO1  TRI   AMP=10000   FREQ=3000               ; 68.7Hz triangle
LFO2  SAW   AMP=$400    FREQ=512    XFAD=1/8    ; 11.7Hz, 1/32 amplitude saw
```

Multiply/Accumulate Instruction Format

The Multiply/Accumulate (MAC) Instruction format is generally as follows:

[CHRn] OPCODE ADDRESS [[K=]COEFFICIENT|CHORUS CONTROLS] [; COMMENTS]

Note: Square brackets "**[**" and "**]**" denote optional fields.

CHRn selects a chorus instruction, and the **n** selects the associated LFO to drive the instruction. The **ADDRESS** specifies an I/O (address 0 is the left channel, address 1 is the right), a relative Data RAM address, or a memory label. The **COEFFICIENT** specifies the multiplier value, and if omitted, is assumed to be 0. **CHORUS CONTROLS** are an unordered list of parameters which activate certain functions for a chorus instruction.

The following opcode mnemonics control the basic functions of the MAC. It defines a read or write operation, and selects which register (none, Acc, B, or C) is added to the multiplier output. Register B can store the current instruction's multiplicand, and register C can store the same value as Acc.

Read Instructions

For read operations, Product = K*[ADDRESS], so register B stores either the Left Input, Right Input, or the Data RAM value specified by the address or label.

RZP	Read,	Acc = Zero	+ Product	
RAP	Read,	Acc = Acc	+ Product	
RBP	Read,	Acc = B Register	+ Product	
RCP	Read,	Acc = C Register	+ Product	
RZPB	Read,	Acc = Zero	+ Product,	Load B register
RAPB	Read,	Acc = Acc	+ Product,	Load B register
RBPB	Read,	Acc = B Register	+ Product,	Load B register
RCPB	Read,	Acc = C Register	+ Product,	Load B register
RZPC	Read,	Acc = Zero	+ Product,	Load C register
RAPC	Read,	Acc = Acc	+ Product,	Load C register
RBPC	Read,	Acc = B Register	+ Product,	Load C register
RCPC	Read,	Acc = C Register	+ Product,	Load C register
RZPBC	Read,	Acc = Zero	+ Product,	Load B and C registers
RAPBC	Read,	Acc = Acc	+ Product,	Load B and C registers
RBPBC	Read,	Acc = B Register	+ Product,	Load B and C registers
RCPBC	Read,	Acc = C Register	+ Product,	Load B and C registers

Write Instructions

For write operations, Product = k*Acc, so register B stores the Acc value from the last tick. Writes to memory stores the result of the previous instruction, while writes to outputs transfers the result of the current instruction.

WZP	Write,	Acc = Zero	+ Product	
WAP	Write,	Acc = Acc	+ Product	
WBP	Write,	Acc = B Register	+ Product	
WCP	Write,	Acc = C Register	+ Product	
WZPB	Write,	Acc = Zero	+ Product,	Load B register
WAPB	Write,	Acc = Acc	+ Product,	Load B register
WBPB	Write,	Acc = B Register	+ Product,	Load B register
WCPB	Write,	Acc = C Register	+ Product,	Load B register
WZPC	Write,	Acc = Zero	+ Product,	Load C register
WAPC	Write,	Acc = Acc	+ Product,	Load C register
WBPC	Write,	Acc = B Register	+ Product,	Load C register
WCPC	Write,	Acc = C Register	+ Product,	Load C register
WZPBC	Write,	Acc = Zero	+ Product,	Load B and C registers
WAPBC	Write,	Acc = Acc	+ Product,	Load B and C registers
WBPBC	Write,	Acc = B Register	+ Product,	Load B and C registers
WCPBC	Write,	Acc = C Register	+ Product,	Load B and C registers

For example:

```
RZP    delay"        K=.500      ; Read (middle of delay)/2 into Acc
WAP    delay+125     192         ; Write to 125th delay tap, Acc = -0.5*Acc
RAPB   delay'        K=128       ; Acc = -1*(end of delay)+Acc, B = end of delay
WBPC   delay'-1                  ; Write to next-to-last delay tap, Acc & C = B + 0*Acc
RCP    ADCL          K=0x7F      ; Read ~1*(Left Input) into Acc and C
WCPB   OUTR          K=-0x40     ; B=Acc, write C-0.5*Acc into Right Output and Acc
```

Chorus Controls

These statements define instructions as chorus commands. They redefine the coefficient bits to alternate meanings, and thus the lack of coefficients in chorus instructions. Two adjacent commands are used to create chorusing, in order to interpolate to the exact fraction of a sample between two (integer) samples.

The Chorus Controls consist of the following parameters:

[+SIN|-SIN|+COS|-COS] [COMPK] [COMPA] [MASKA] [LATCH] [COMPS]

+SIN	Use +sine output of selected LFO.
-SIN	Use -sine output of selected LFO.
+COS	Use +cosine output of selected LFO.
-COS	Use -cosine output of selected LFO.

The LFOs each have four taps: a ±sine output and a ±cosine output. The parameters described above indicate which tap to use. If this parameter is omitted, the default value used is **+SIN**. The sign is mandatory, "SIN" will be flagged as an error and prevent assembly.

COMPK One's complement the chorus interpolation coefficient.

The LFO output is split into two parts: an offset address and an interpolation coefficient. The offset address is sent to the address generator for the Data RAM, and is not accessible to the user. The interpolation coefficient should be applied to the value at the specified address, the 1's complement of the coefficient then applied to the value at address+1. The **COMPK** parameter will perform a one's complement on the coefficient.

COMPA One's complement the chorus interpolation address.

This parameter will perform a one's complement on the chorus interpolation address, effectively shifting the phase of the chorus waveform by 180°. Specifying **COMPA** with **+SIN** is equal to specifying **-SIN**. In fact **-SIN** or **-COS** sets the same bit as **COMPA**. This statement is implemented for use in pitch transposition for increasing source code clarity. Proper use of the LFO output statements will make this command superfluous.

MASKA Mask chorus generator offset address, select the crossfade coefficient.

When the two interpolations required by pitch transposition are complete, they must be read back from memory and summed using the crossfade coefficient. In a chorus command, the address generator will want to add the chorus offset address when generating the absolute address. This can be overridden with the **MASKA** parameter so that the offset is masked off, and the address read from is as specified in the instruction. This parameter also indicates that the crossfade coefficients selected by the LFO Setup Instruction be used.

LATCH Latch the data from selected LFO.

When pairs of chorus commands are implemented, the LFO data must be latched; otherwise, the LFO could change during the execution of the commands, and unpredictable results could occur. This is required on the first chorus command, but must be omitted from the second.

COMPS One's complement the sign bit of the chorus offset address.

This parameter will perform a one's complement of the sign bit of the chorus offset address. This is a leftover from a previous ASIC revision and is included here for informational purposes.

For example:

```
CHR0  RZP  mem    -COS   COMPK  LATCH   ; Read fractional -COS chorus of mem
CHR0  RAP  mem+1  -COS                  ; Sum with fractional of 2nd address

CHR1  RZP  stor1  COMPK  MASKA  LATCH   ; Read fractional crossfade of stor1
CHR1  RAP  stor2         MASKA          ; Sum with fractional crossfade of stor2
```

ABS Statement

DREAsm provides the **ABS** statement as a means of declaring absolute address labels. The **ABS** statement format is as follows:

ABS NAME [ADDRESS] [; COMMENTS]

where **NAME** can be any unique alpha-numeric string. The first character of a name must be a letter, and spaces are not permitted. **ADDRESS** is the absolute address. The address may be specified as a decimal or hexadecimal integer. To specify hexadecimal, prefix the number with a "**0x**" or suffix it with an "**H**".

For example:

```
ABS   param     0x0081     ; Use direct address memory as a parameter
```

Memory address offsets and addresses above 129 do not work with the **ABS** statement. Be sure that the code only accesses the exact address below 130 specified by **ABS**.

For example:

```
ABS   hparams   0x0082     ; This will give an error
ABS   params    0x0081     ; This will work
RZP   params+3  0.5        ; This will give an error
RZP   params    0.5        ; This will work
```

EQU Statement

DREAsm provides the **EQU** statement as a means of defining constants for coefficients. The **EQU** statement format is as follows:

NAME EQU VALUE [; COMMENTS]

where **NAME** can be any unique alpha-numeric string. The first character of a name must be a letter, and spaces are not permitted. **VALUE** is the numerical value for **NAME**, and must be a valid coefficient format.

For example:

```
gain  EQU   0.5      ; Specifies a gain coefficient of -6dB
RZP   INL   gain     ; Get Left Input, gained down, in accumulator
WAP   OUTL  0.0      ; Write the accumulator to Left Output
```

Note that **EQU** statements are not recursive. You may not equate a name to a value, and then equate another name to the first name.

MEM Statement

DREAsm provides the **MEM** statement as a means of declaring address labels and allocating Data RAM usage. Address labels may be placed anywhere in the assembly file, as long as they are declared before their usage. The **MEM** statement format is as follows:

MEM NAME SIZE [; COMMENTS]

where **NAME** may be any unique alphanumeric string. The first character of a name must be a letter, and spaces are not permitted. **SIZE** is the size of the memory block, and is specified as the number of samples in decimal or hexadecimal. To specify a hexadecimal block size, prefix the number with a "**0x**" or suffix it with an "**H**". Size may also be specified in milliseconds (assuming a 44.1kHz sample frequency) by suffixing the number with "**ms**".

For example:

```
MEM   LPF       1        ; Specifies a unit delay 1 sample long
MEM   delay     100      ; Specifies a delay line 100 samples long
MEM   FIR       0x20     ; Specifies a delay line 32 samples long
MEM   predelay  50.0ms   ; Specifies a delay line 50.0ms long
```

DREAsm will allocate the Data RAM beginning at address 0 and work up to the higher addresses. Note that since this declaration takes the memory offset counter into account, the actual number of memory locations allocated is 1 larger than specified. Also note that since the memory offset counter is a down-counter, writes should be done to lower addresses and reads of delayed data done from higher addresses.

For example:

```
WZP   0H        K=0.5    ; Write data to be delayed
RZP   4H        K=0.5    ; Read 4-sample-delayed data
```

Once you have defined your memory elements, you may access them in a number of ways. Simply using the name refers to the front of the delay line. Suffixing the name with an apostrophe (**'**) refers to the back of the delay line. Suffixing the name with a quote (**"**) refers to the middle of the delay line.

For example, here is the code for a 100-sample delay:

```
MEM   delay     100      ; Specifies a delay line 100 samples long
RZP   INL       127      ; Get Left Input into the accumulator
WAP   delay     0        ; Write accumulator to front of delay line
RZP   delay'    127      ; Read from the back of the delay line
WAP   OUTL      0        ; Write the accumulator to Left Output
```

You can offset into a delay line by using the + or - modifiers.

For example, to access the 512th tap of a long FIR filter:

```
MEM   FIR       1023     ; Use 1023 locations of Data RAM
RZP   INR       127      ; Get Right Input into the accumulator
WAP   FIR       0        ; Write accumulator to the front of delay line
RZP   FIR+512   127      ; Read the 512th tap of the FIR
WAP   OUTR      0        ; Write the accumulator to Right Output
```

The suffixes and offsets may be mixed in an instruction.

SEG Statement

DREAsm provides the **SEG** statement as a means of segmenting the Data RAM by setting the starting point for memory allocation. The **SEG** statement format is as follows:

SEG ADDRESS [; COMMENTS]

where **ADDRESS** is the start address of the memory segment. The address is specified in decimal or hexadecimal integer. To specify hexadecimal, prefix the number with a "**0x**" or suffix it with an "**H**".

For example:

```
SEG   LEFT     2        ; "Left" segment starts at the bottom of memory
MEM   delayl   100      ; Specifies a delay line 100 samples long
SEG   RIGHT    0x4000   ; "Right" segment starts at the middle of memory
MEM   delayr   100      ; Specifies a delay line 100 samples long
```

DREAsm will allocate the Data RAM beginning at the specified address and work up to the higher addresses. Be sure that the address specified is not 0 or 1, as those are reserved for the Left/Right Input/Output.

Reserved Memory Labels

For your convenience, a number of memory labels have been pre-defined by the assembler. They may be accessed them just as any other memory element. They are:

INL ---------- Left Input (read only)
ADCL ---------- Left Input (read only)
INR ---------- Right Input (read only)
ADCR ---------- Right Input (read only)
OUTL ---------- Left Output (write only)
OUTR ---------- Right Output (write only)

For example:

```
RZP   INL      0.5      ; Read Left Input
RZP   ADCL     0.5      ; Read Left Input
RZP   0        0.5      ; Read Left Input
```

Comments

Comments are delimited by the semicolon (**;**) character. The assembler ignores everything from the semicolon to the end of the line.

For example:

```
; ::::::::This is a valid comment::::::::
RZP   mem   $5     ;This is also a valid comment; all text here is ignored.
```

NOTICE

Wavefront Semiconductor reserves the right to make changes to their products or to discontinue any product or service without notice. All products are sold subject to terms and conditions of sale supplied at the time of order acknowledgement. Wavefront Semiconductor assumes no responsibility for the use of any circuits described herein, conveys no license under any patent or other right, and makes no representation that the circuits are free of patent infringement. Information contained herein is only for illustration purposes and may vary depending upon a user's specific application. While the information in this publication has been carefully checked, no responsibility is assumed for inaccuracies.

Wavefront Semiconductor products are not designed for use in applications which involve potential risks of death, personal injury, or severe property or environmental damage or life support applications where the failure or malfunction of the product can reasonably be expected to cause failure of the life support system or to significantly affect its safety or effectiveness.

All trademarks and registered trademarks are property of their respective owners.

Contact Information:

Wavefront Semiconductor
200 Scenic View Drive
Cumberland, RI 02864 U.S.A.
Tel: +1 401 658-3670
Fax: +1 401 658-3680
On the web at www.wavefrontsemi.com
Email: info@wavefrontsemi.com

Application note revised March, 2005