Another Signal Manager

Erwin Waterlander

Jul 1 2022

Contents

Chapter 1

Introduction

This document can be downloaded from this URL:
`https://waterlan.home.xs4all.nl/asm.html`

1.1 What is ASM?

ASM is a program for digital signal processing for educational purposes.

This is a port of the original version made in 1993 by Edwin Zoer and me on an Acorn Archimedes computer running RISC-OS.

1.2 History

1.2.1 AIM : Another Image Manager

also known as: Atari Image Manager, Archimedes Image Manager, Amiga Image Manager.

The image processing program AIM was originally developed for the ATARI-ST by Frans Groen and Robert de Vries. Since the first version of AIM, the improvement of this public domain image processing package has become a joint effort of a number of people from the Delft University of Technology and the University of Amsterdam. AIM has been ported to the ARCHIMEDES (Arthur version) by Robert Ellens, Damir Sudar and Alle-Jan van der Veen. Ed Doppenberg was successful in the port to RISC-OS. AIM has been written in the C-language. AIM is limited in functionality as well as in flexibility. The main purpose of the program is to experiment with digital image processing.

The latest version 3.15 (1995) for Archimedes RISC-OS can be downloaded from http://wuerthner.dyndns.org/others.html

On the Polytechnic of Enschede the Archimedes RISC-OS version of AIM was used in practical lessons in image processing. Polytechnic of Enschede

(Hogeschool Enschede), the Netherlands, is called Saxion hogescholen (www.saxion.nl) today.

1.2.2 ASM : Another Signal Manager

In 1993 the idea came to make a program like AIM, but then for signal processing: ASM for RISC-OS. The task of our final examination for the Polytechnic of Enschede was to create ASM for RISC-OS.

We made ASM at and with support of the Technical University of Delft, faculty Applied Physics, Pattern Recognition group (Tom Hoeksma), and with support from the Technisch Physische Dienst, Delft (Ed Doppenberg). Our starting point was a stripped down version of AIM made by Ed Doppenberg. It was only one window with a command line interpreter.

In 1993 Edwin and I had only basic knowledge of ANSI C and no knowledge about making user interfaces for RISC-OS. Our goal was to put as much as possible functionality in the program in only three months.

With the first RISC-OS version we created it was possible to generate signals and do some basic processing on them. The program was made for use during practical lessons in digital signal processing at the polytechnic in Enschede.

The original intention was that other students would develop ASM further. But it was Ed Doppenberg who did a thorough revision of the source code and added some professional functionality. That version of ASM (for RISC-OS) is not free available for the public domain.

In 1997 I ported the first version of ASM for RISC-OS to DOS using DJGPP 2.01 (gcc 2.7.2). See djgpp.txt. I used the Allegro 2.2 graphics library. Allegro is a library intended for use in computer games. It was initially conceived on the Atari ST. See allegro.txt.

In 1997 my primary goal was to port the program to a working version on DOS, for the fun of programming and because I had no Archimedes computer. That means I only changed the graphical interface. I tried to keep the source code as much as possible the same. There are large changes in file plotutil.c. Files aim.c and command.c have been replaced by asm.c.

Allegro development went on, supporting more platforms. In 2007 I build ASM also for Windows and Linux. I replaced some deprecated Allegro api calls with new ones, and build ASM for DOS, Windows and Linux. A few minor problems have been fixed. For the rest it's the same version as in 1997.

This version of ASM is Public Domain software.

Erwin Waterlander
e-mail: waterlan@xs4all.nl

Chapter 2

User interface

2.1 Graphical interface

ASM for RISC-OS has a windowed interface. One command line window and each signal shown in a separate window with scroll bars.

Due to the limitations of DOS (no window environment) I had to change the graphical interface. ASM for DOS can display only one signal at a time. Other signals stay resident in memory.

2.2 Command line interface

ASM has the same command line interface as AIM.

Commands can be abbreviated as long as they stay unique.

Parameters are separated by spaces. If you don't give all the parameters that are possible on a certain command ASM will take default values. Sometimes ASM will ask for a necessary parameter because it has to be given or it was out of range.

A handy feature is to give a single question mark '**?**' (without quotes) as parameter. ASM will now ask interactively all the parameters and show the default value and the range.

2.3 Keyboard short-keys

ASM has the following short-keys:

`<F1>` Print help

`<F2>, <Left>` Scroll image left

`<F3>, <Right>` Scroll image right

`<F4>, <Shift>-<Left>` Display previous signal

`<F5>, <Shift>-<Right>` Display next signal

`<Ctrl>-<Left>` Zoom out

`<Ctrl>-<Right>` Zoom in

`<Esc>, <Alt>-X, <Ctrl>-C` Back to title screen

2.4 Initialization

On startup ASM will execute all commands that are in file *asm.ini* if it is in the current directory. ASM reads it's fonts from file asm.dat if it is in the current directory.

If ASM is invoked with the **-l** option, ASM will run in SVGA 800x600 mode. Default is SVGA 1024x768. It is alway a 256 color mode.

The DOS version of ASM runs in full screen mode. The Windows and Linux version run in 'windowed' mode. In Windowed mode the 'specified resolution' actually means the graphic area ASM can draw on, without including window decorations (if any).

2.5 Memory usage

ASM for DOS is a 32 bit program. ASM runs in DOS Protected Mode. It needs the DOS Extender program cwsdpmi.exe that comes with ASM for DOS. ASM can use all available memory in a PC. ASM for DOS has a build-in virtual memory manager and can use up to 256 MB of virtual memory.

ASM for Windows and Linux are 32 bit programs.

The maximum number of signals that ASM can hold in memory is 100. This is a compiled in define.

Chapter 3

Functionality

3.1 Data-format conversion

Internally ASM does all it's calculations in 64 bit floating point (type double). Data can be stored at disk in different resolutions: 8 bit char, 16 bit short, 32 bit int, 32 bit float or 64 bit float (double). While reading data from disk the data is always converted to 64 bit float.

3.2 Domain conversion

Conversions can be done between the different domains:

From \To	Time	Frequency	Amplitude	Magnitude	Phase
Time		X	X	X	X
Frequency	X			X	X
Amplitude					
Magnitude					
Phase					

Table 3.1: Domain conversions

3.2.1 From time to frequency

fft : Fast-Fourier- Transformation

```
command: fft        input-signal, output-signal, length, window-type, average-type
default  :               a       ,        b      ,   9   ,      1     ,      0
range    :             <a-z>     ,      <a-z>    , <7-12>,    <0-6>   ,    <0-1>
domain   : time
```

```
Windows:
```

```
0  block
1  Hanning
2  Hamming
3  Gauss
4  Blackman
5  Kaiser
6  triangle
```

average-type, see section 3.2.4 and 3.2.5.

FFT is done per record. The default length is the record length of the input signal.

3.2.2 From frequency to time

ifft : Inverse Fast-Fourier-Transformation

```
command: ifft  input-signal, output-signal
default  :          a      ,       b
range    :        <a-z>    ,     <a-z>
domain   : frequency
```

3.2.3 From time to amplitude

histogram

```
command: histogram  input-signal, output-signal, buckets
default  :               a      ,       b      ,    9
range    :             <a-z>    ,     <a-z>    ,  <7-12>
domain   : time
```

The number of buckets is given as a power of 2. So 9 means $2^9 = 512$ buckets.

3.2.4 From frequency to magnitude

There are two different ways of averaging:

Average-type 0:

1. Calculate magnitude for every record:
   $|F(u)| = \sqrt{Re^2(u) + Im^2(u)}, u = 0, 1, ..., N - 1$
2. sum all the results from the different records
3. and calculate $10 \cdot \log$.

Average-type 1:

1. Sum all the different records,
2. calculate the magnitude of the result
   $|F(u)| = \sqrt{Re^2(u) + Im^2(u)}, u = 0, 1, ..., N - 1$
3. and calculate $10 \cdot \log$.

magnitude : Calculate the signal's magnitude

```
command: : magnitude  input-signal, output-signal, channel-no, average-type,  log
default  :                 a      ,  input-signal,     0      ,      0       ,   0
range    :               <a-z>    ,     <a-z>     ,  <0-max> ,    <0-1>     , <0-1>
domain   : frequency

log:
0 = linear Y-axis
1 = log Y- axis
```

3.2.5 From frequency to phase

Average-type 0:

1. Calculate the phase of every record:
   $\Phi(u) = tan^{-1}\left[\frac{Im(u)}{Re(u)}\right], u = 0, 1, ..., N-1$
2. Sum the different records and divide by the number of records.

Average-type 1:

1. Sum the different records and divide by the number of records.
2. Calculate the phase of the result:
   $\Phi(u) = tan^{-1}\left[\frac{Im(u)}{Re(u)}\right], u = 0, 1, ..., N-1$

phase : Calculate the signal's phase

```
command: : phase      input-signal, output-signal, channel-no, average-type
default  :                 a      ,  input-signal,     0      ,      0
range    :               <a-z>    ,     <a-z>     ,  <0-max> ,    <0-1>
domain   : frequency
```

3.2.6 From time to magnitude

Not implemented.

3.2.7 From time to phase

Not implemented.

3.3 Mathematical functions

There are a few simple mathematical functions in ASM. The functions can be performed on signals in any domain. For functions that work on more than one input-signal the record-length and the number of channels have to be the same for all the input signals. All constant values are values between $< min > = INT\_MIN = -2147483647$ and $< max > = INT\_MAX = 2147483647$.

clear : Make all elements 0

$$Re(out) = 0$$

$$Im(out) = 0$$

```
command: : clear      Input-signal
default  :                 a
range    :               <a-z>
domain   : time, frequency, amplitude, magnitude, phase
```

assign : Assign a constant value to all elements

$$Re(out) = C$$

$$Im(out) = C$$

```
command: : assign     Input-signal,   real-part,   imag-part
default  :                 a      ,       1    ,       1
range    :               <a-z>    ,   <min-max>,   <min-max>
domain   : time, frequency, amplitude, magnitude, phase
```

inv : Invert all elements

$$Re(out) = -1 * Re(in)$$

$$Im(out) = -1 * Im(in)$$

```
command: : inv        Input-signal,  Output-signal
default  :                 a      ,   input-signal
range    :               <a-z>    ,      <a-z>
domain   : time, frequency, amplitude, magnitude, phase
```

conjugate :

$$Re(out) = Re(in)$$

$$Im(out) = -1 * Im(in)$$

```
command: : conjugate  Input-signal,  Output-signal
default  :                 a      ,   input-signal
range    :               <a-z>    ,      <a-z>
domain   : time, frequency, amplitude, magnitude, phase
```

cabs : Calculate absolute value of each element

$$Re(output) = \sqrt{Re^2(input) + Im^2(input)}$$

$$Im(output) = 0$$

```
command: : cabs       Input-signal,  Output-signal
default  :                 a      ,   input-signal
range    :               <a-z>    ,      <a-z>
domain   : time, frequency, amplitude, magnitude, phase
```

cadd : Add a constant value

$$Re(out) = Re(in) + C$$

$$Im(out) = Im(in)$$

```
command: : cadd       Input-signal,  constant,  Output-signal
default  :                a      ,      0    ,   input-signal
range    :              <a-z>    , <min-max>,      <a-z>
domain   : time, frequency, amplitude, magnitude, phase
```

cmultiply : Multiply by a constant value

$$Re(out) = C * Re(in)$$

$$Im(out) = C * Im(in)$$

```
command: : cmultiply  Input-signal,  constant,  Output-signal
default  :                a      ,      1    ,   input-signal
range    :              <a-z>    , <min-max>,      <a-z>
domain   : time, frequency, amplitude, magnitude, phase
```

cdivide : Divide by a constant

$$Re(out) = \frac{Re(in)}{C}$$

$$Im(out) = \frac{Im(in)}{C}$$

```
command: : cdivide    Input-signal,  constant,  Output-signal
default  :                a      ,      1    ,   input-signal
range    :              <a-z>    ,  <1-max> ,      <a-z>
domain   : time, frequency, amplitude, magnitude, phase
```

abs : Absolute difference between two signals

$$Re(out) = \sqrt{(Re(in1) - Re(in2))^2 + (Im(in1) - Im(in2))^2}$$

$$Im(out) = 0$$

```
command: : abs        Input-signal1, Input-signal2,  Output-signal
default  :                a       ,      b       ,   input-signal2
range    :              <a-z>     ,    <a-z>     ,      <a-z>
domain   : time, frequency, amplitude, magnitude, phase
```

add : Add two signals

$$Re(out) = Re(in1) + Re(in2)$$

$$Im(out) = Im(in1) + Im(in2)$$

```
command: : add        Input-signal1, Input-signal2,  Output-signal
default  :                a       ,      b       ,   input-signal2
range    :              <a-z>     ,    <a-z>     ,      <a-z>
domain   : time, frequency, amplitude, magnitude, phase
```

substract : Substract two signals

$$Re(out) = Re(in1) - Re(in2)$$

$$Im(out) = Im(in1) - Im(in2)$$

```
command: : substract  Input-signal1, Input-signal2,  Output-signal
default  :                 a       ,      b       ,   input-signal2
range    :               <a-z>     ,    <a-z>     ,      <a-z>
domain   : time, frequency, amplitude, magnitude, phase
```

multiply : Multiply two signals

$$Re(out) = \frac{Re(in1) * Re(in2) - Im(in1) * Im(in2)}{C}$$

$$Im(out) = \frac{Re(in1) * Im(in2) + Im(in1) * Re(in2)}{C}$$

```
command: : multiply   Input-signal1, Input-signal2,  constant, Output-signal
default  :                 a       ,      b       ,      1   ,  input-signal2
range    :               <a-z>     ,    <a-z>     ,   <1-max>,     <a-z>
domain   : time, frequency, amplitude, magnitude, phase
```

divide : Divide two signals

$$Re(out) = \frac{\sqrt{Re^2(in1) + Im^2(in1)}}{1 + \sqrt{Re^2(in2) + Im^2(in2)}}$$

$$Im(out) = 0$$

```
command: : divide     Input-signal1, Input-signal2,  Output-signal
default  :                 a       ,      b       ,   input-signal2
range    :               <a-z>     ,    <a-z>     ,      <a-z>
domain   : time, frequency, amplitude, magnitude, phase
```

sine : Calculate sine

$$Re(out) = \sin(Re(in))$$

$$Im(out) = 0$$

```
command: : sine       Input-signal,  Output-signal
default  :                 a      ,  input-signal
range    :               <a-z>    ,     <a-z>
domain   : time, frequency, amplitude, magnitude, phase
```

cosine : Calculate sine

$$Re(out) = \cos(Re(in))$$

$$Im(out) = 0$$

```
command: : cosine     Input-signal,  Output-signal
default  :                 a      ,  input-signal
range    :               <a-z>    ,     <a-z>
domain   : time, frequency, amplitude, magnitude, phase
```

ln : logarithm

$$Re(out) = \ln Re(in)$$

$$Im(out) = 0$$

```
command: : ln         Input-signal,  Output-signal
default  :                 a      ,  input-signal
range    :               <a-z>    ,     <a-z>
domain   : time, frequency, amplitude, magnitude, phase
```

log : $^{10}\log$

$$Re(out) = \log Re(in)$$

$$Im(out) = 0$$

```
command: : log        Input-signal,  Output-signal
default  :                 a      ,  input-signal
range    :               <a-z>    ,     <a-z>
domain   : time, frequency, amplitude, magnitude, phase
```

epow :

$$Re(out) = e^{Re(in)}$$

$$Im(out) = 0$$

```
command: : epow       Input-signal,  Output-signal
default  :                 a      ,  input-signal
range    :               <a-z>    ,     <a-z>
domain   : time, frequency, amplitude, magnitude, phase
```

tenpow :

$$Re(out) = 10^{Re(in)}$$

$$Im(out) = 0$$

```
command: : tenpow     Input-signal,  Output-signal
default  :                 a      ,  input-signal
range    :               <a-z>    ,     <a-z>
domain   : time, frequency, amplitude, magnitude, phase
```

minimum : minimum of two signals

$$Re(out) = \sqrt{Re^2(in1) + Im^2(in1)} < \sqrt{Re^2(in2) + Im^2(in2)} ? Re(in1) : Re(in2)$$

$$Im(out) = \sqrt{Re^2(in1) + Im^2(in1)} < \sqrt{Re^2(in2) + Im^2(in2)} ? Im(in1) : Im(in2)$$

```
command: : minimum    Input-signal1, Input-signal2,  Output-signal
default  :                 a       ,      b       ,  input-signal2
range    :               <a-z>     ,    <a-z>     ,      <a-z>
domain   : time, frequency, amplitude, magnitude, phase
```

maximum : maximum of two signals

$$Re(out) = \sqrt{Re^2(in1) + Im^2(in1)} > \sqrt{Re^2(in2) + Im^2(in2)} ? Re(in1) : Re(in2)$$

$$Im(out) = \sqrt{Re^2(in1) + Im^2(in1)} > \sqrt{Re^2(in2) + Im^2(in2)} ? Im(in1) : Im(in2)$$

```
command: : maximum    Input-signal1, Input-signal2,  Output-signal
default  :                  a       ,       b       ,  input-signal2
range    :                <a-z>     ,    <a-z>      ,      <a-z>
domain   : time, frequency, amplitude, magnitude, phase
```

3.4 Correlation

The correlation of two signals of N samples is calculated as follows:

1. Zeropad the signals per record, to prevent wrap-around pollution.
2. Multiply the signals with an N points window.
3. Calculate the 2N points FFT of the signals.
4. Multiply the complex conjugate of the first with the second signal.
5. Calculate the 2N points IFFT.

correlation : calculate correlation of two signals

```
command: : correlation  Input-signal1, Input-signal2, Output-signal, window-type
default  :                    a       ,       b       , input-signal2,      0
range    :                  <a-z>     ,    <a-z>      ,    <a-z>     ,    <0-6>
domain   : time
```

```
Windows:
0  block
1  Hanning
2  Hamming
3  Gauss
4  Blackman
5  Kaiser
6  triangle
```

3.5 Convolution

The convolution of two signals of N samples is calculated as follows:

1. Zeropad the signals per record, to prevent wrap-around pollution.
2. Multiply the signals with an N points window.
3. Calculate the 2N points FFT of the signals.
4. Multiply the first with the second signal.
5. Calculate the 2N points IFFT.

convolution : calculate convolution of two signals

```
command: : convolution  Input-signal1, Input-signal2, Output-signal, window-type
default  :                   a       ,      b       , input-signal2,      0
range    :                 <a-z>     ,    <a-z>     ,    <a-z>     ,    <0-6>
domain   : time
```

3.6 Functions

ASM can generate some standard signals. Currently the functions are limited to 1 record and 1 channel.

- ts = sample time
- A = Amplitude
- Amax = 2147483647
- B = offset
- Bmin = -2147483647
- Bmax = 2147483647
- T = period time
- f = frequency
- fmax = 2147483647
- data-type: 0=real, 1=imaginary, 2=complex
- N = number of elements = 2^n Nmin = 128 (n=7), Nmax = 4096
- Smax = 65535, maximal sample-rate (in 10 Hz)

Sample-rate is always given in 10 (Hz).

fdelta :

$$x(n \cdot t_s) = A,\ n \cdot t_s = t_d,\ n = 0, 1, ..., N-1$$

$$x(n \cdot t_s) = 0,\ n \cdot t_s \neq t_d,\ n = 0, 1, ..., N-1$$

```
command: : fdelta  signal,     A    ,    td (ms) , data-type,    n   , sample-rate
default  :           a    ,   255   ,      0      ,     0     ,    9   ,   1024
range    :         <a-z> , <0-Amax> , <0-t(N-1)> ,   <0-2>   ,  <7-12>,  <1-Smax>
domain   : time
```

fconstant :

$$x(n \cdot t_s) = A,\ n = 0, 1, ..., N-1$$

```
command: : fconstant  signal,     A    , data-type,    n   , sample-rate
default  :              a    ,   255   ,     0     ,    9   ,   1024
range    :            <a-z> , <0-Amax> ,   <0-2>   ,  <7-12>,  <1-Smax>
domain   : time
```

fstep :

$$x(n \cdot t_s) = B,\ n \cdot t_s < t_{step},\ n = 0, 1, ..., N-1$$

$$x(n \cdot t_s) = A + B,\ n \cdot t_s \geq t_{step},\ n = 0, 1, ..., N-1$$

```
command: : fstep  signal,      B      ,     A     , tdelay(ms) , data-type,    n   , sample-rate
default  :          a   ,      0      ,    255    ,     0      ,     0    ,    9   ,   1024
range    :        <a-z> , <Bmin-Bmax>, <0-Amax> , <0-t(N-1)> ,   <0-2>   ,  <7-12>,  <1-Smax>
domain   : time
```

fsquare :

$$x(n \cdot t_s) = A + B,\ 0 \leq n \cdot t_s < dc \cdot T,\ n = 0, 1, ..., N-1$$

$$x(n \cdot t_s) = -A + B,\ dc \cdot T \leq n \cdot t_s < T,\ n = 0, 1, ..., N-1$$

```
command: : fsquare  signal,      B      ,     A     ,     f    ,   dc    , data-type,    n   , sample-rate
default  :            a   ,      0      ,    255    ,    100   ,   50    ,     0    ,    9   ,   1024
range    :          <a-z> , <Bmin-Bmax>, <0-Amax> , <1-fmax> , <0-100>,   <0-2>   ,  <7-12>,  <1-Smax>
domain   : time
```

dc = duty cycle

framp :

$$x(n \cdot t_s) = \frac{A}{t_s \cdot (N-1)} \cdot n \cdot t_s + B,\ n = 0, 1, ..., N-1$$

```
command: : framp  signal,      B      ,     A     ,data-type,    n   , sample-rate
default  :          a   ,      0      ,    255    ,   0     ,    9   ,   1024
range    :        <a-z> , <Bmin-Bmax>, <0-Amax> ,  <0-2>   ,  <7-12>,  <1-Smax>
domain   : time
```

ftriangle :

$$x(n \cdot t_s) = \frac{4A}{T} \cdot (n \cdot t_s + \frac{\phi_0 \cdot T}{2\pi}) - A + B,\ 0 \leq n \cdot t_s < \frac{1}{2}T,\ n = 0, 1, ..., N-1$$

$$x(n \cdot t_s) = \frac{-4A}{T} \cdot (n \cdot t_s + \frac{\phi_0 \cdot T}{2\pi}) + 3A + B,\ \frac{1}{2}T \leq n \cdot t_s < T,\ n = 0, 1, ..., N-1$$

```
command: : ftriangle  signal,      B      ,     A     ,     f    ,   phi0 , data-type,    n   , sample-rate
default  :              a   ,      0      ,    255    ,    100   ,    0   ,       0  ,    9   ,   1024
range    :            <a-z> , <Bmin-Bmax>, <0-Amax> , <1-fmax> , <0-2pi>,   <0-2>   ,  <7-12>,  <1-Smax>
domain   : time
```

fsine :

$$x(n \cdot t_s) = A \sin(2\pi f n t_s + \phi_0) + B,\ n = 0, 1, ..., N-1$$

```
command: : fsine  signal,      B      ,     A     ,     f    ,   phi0 , data-type,    n   , sample-rate
default  :          a   ,      0      ,    255    ,    100   ,    0   ,   0      ,    9   ,   1024
range    :        <a-z> , <Bmin-Bmax>, <0-Amax> , <1-fmax> , <0-2pi>,   <0-2>   ,  <7-12>,  <1-Smax>
domain   : time
```

fsinc :

$$x(n \cdot t_s) = A \left(\frac{\sin(2\pi f n t_s)}{2\pi f n t_s} \right) + B,\ n = 1, ..., N-1$$

$$x(n \cdot t_s) = A \cos(2\pi f n t_s) + B,\ n = 0$$

```
command: : fsinc  signal,      B      ,     A     ,     f     , data-type,     n    , sample-rate
default  :          a    ,      0      ,    255    ,    100    ,   0      ,     9    ,   1024
range    :        <a-z> , <Bmin-Bmax>, <0-Amax> , <1-fmax> ,   <0-2>   ,  <7-12>,  <1-Smax>
domain   : time
```

fcosine :

$$x(n \cdot t_s) = A \cos(2\pi f n t_s + \phi_0) + B,\ n = 0, ..., N-1$$

```
command: : fcosine  signal,      B      ,     A     ,     f     ,   phi0 , data-type,     n    , sample-rate
default  :            a    ,      0      ,    255    ,    100    ,    0    ,   0      ,     9    ,   1024
range    :          <a-z> , <Bmin-Bmax>, <0-Amax> , <1-fmax> , <0-2pi>,   <0-2>   ,  <7-12>,  <1-Smax>
domain   : time
```

fexp :

$$x(n \cdot t_s) = A(1 - e^{\frac{-n \cdot t_s}{t_{63.2\%}}}),\ n = 0, ..., N-1$$

```
command: : fexp     signal,     A    , t63.2(ms), data-type,     n    , sample-rate
default  :            a    ,    255    ,    0.10   ,     0     ,     9    ,       1024
range    :          <a-z> , <0-Amax> ,<1e-6-1e6>,   <0-2>   ,  <7-12>,  <1-Smax>
domain   : time
```

fnoise : pseudo random noise.

```
command: : fnoise  signal,     A    , data-type,  seed  ,    n    , sample-rate
default  :           a    ,    255    ,     0     ,    1   ,    9    ,    1024
range    :         <a-z> , <0-Amax> ,   <0-2>   , <0-512>, <7-12>,  <1-Smax>
domain   : time
```

3.7 Conditioning

Zero padding is extending each record of N samples with N zeros. The output signal has a record length of 2N.

zeropadding :

```
command: : zeropad  input-signal, output-signal
default  :               a      ,  input-signal
range    :             <a-z>    ,     <a-z>
domain   : time
```

3.8 Windowing

wblock : block window

$$W(n) = 1,\ n = 0, 1, ..., N-1$$

```
command: : wblock   signal,    n    , samplerate
default  :            a    ,   9    ,    1024
range    :          <a-z> ,  <7-12> ,  (1-Smax)
domain   : time
```

whanning : Hanning window

$$W(n) = \frac{1}{2}\left(1 - \cos(\frac{2\pi n}{N-1})\right),\ n = 0, 1, ..., N-1$$

```
command: : whanning   signal,    n    , samplerate
default  :              a    ,   9    ,    1024
range    :            <a-z> ,  <7-12> ,  (1-Smax)
domain   : time
```

whamming : Hamming window

$$W(n) = 0.538 - 0.462\cos(\frac{2\pi n}{N-1}),\ n = 0, 1, ..., N-1$$

```
command: : whamming   signal,    n    , samplerate
default  :              a    ,   9    ,    1024
range    :            <a-z> ,  <7-12> ,  (1-Smax)
domain   : time
```

wgauss : Gauss window

$$W(n) = e^{-\frac{1}{2}\left(\frac{\alpha\cdot(n-\frac{N-1}{2})\cdot 2}{\frac{N-1}{2}}\right)^2},\ n = 0, 1, ..., N-1$$

$$\alpha = 3.0$$

```
command: : wgauss     signal,    n    , samplerate
default  :              a    ,   9    ,    1024
range    :            <a-z> ,  <7-12> ,  (1-Smax)
domain   : time
```

wblackman : Blackman window

$$W(n) = 0.42 - 0.5\cos(\frac{2\pi n}{N-1}) + 0.08\cos(\frac{4\pi n}{N-1}),\ n = 0, 1, ..., N-1$$

```
command: : wblackman  signal,    n    , samplerate
default  :              a    ,   9    ,    1024
range    :            <a-z> ,  <7-12> ,  (1-Smax)
domain   : time
```

wkaiser : Kaiser window

$$W(n) = \frac{1}{2.48}\left(1 - 1.24\cos(\frac{2\pi n}{N-1}) + 0.244\cos(\frac{4\pi n}{N-1}) - 0.00305\cos(\frac{6\pi n}{N-1})\right),$$

$$n = 0, 1, ..., N-1$$

```
command: : wkaiser    signal,    n    , samplerate
default  :              a    ,   9    ,    1024
range    :            <a-z> ,  <7-12> ,  (1-Smax)
domain   : time
```

wtriangle :

$$W(n) = 1 - \left|\frac{n - \frac{N-1}{2}}{\frac{N-1}{2}}\right|,\ n = 0, 1, ..., N-1$$

```
command: : wtriangle  signal,    n    , samplerate
default  :              a    ,   9    ,    1024
range    :            <a-z> ,  <7-12> ,  (1-Smax)
domain   : time
```

3.9 Presentation functions

3.9.1 Time domain

real : Show the real part of the signal.

```
command: : real    signal, channel-no ,  record-no
default  :           a   ,      0     ,      0
range    :         <a-z> ,  <0-max>   ,   <0-max>
domain   : time, frequency
```

imaginary : Show the imaginary part of the signal.

```
command: : imaginary  signal, channel-no ,  record-no
default  :              a   ,      0     ,      0
range    :            <a-z> ,  <0-max>   ,   <0-max>
domain   : time, frequency
```

3.9.2 Frequency domain

real : See section 3.9.1

imaginary : See section 3.9.1

bode : Show bode diagram

```
command: : bode       signal, channel-no ,  record-no
default  :              a   ,      0     ,      0
range    :            <a-z> ,  <0-max>   ,   <0-max>
domain   : frequency
```

3.9.3 Magnitude and phase domain

3.9.4 Generic functions

doff : display off, do not display signals

```
command: : doff
```

don : display on, display signals

```
command: : don
```

boff : bar display off, display signals as normal graphs.

```
command: : boff
```

bon : bar display on, display signals as bargraphs.

```
command: : bon
```

display : display a signal, regardless of **doff/don**.

```
command: : display    signal, channel-no ,  record-no
default  :              a   ,      0     ,      0
range    :            <a-z> ,  <0-max>   ,   <0-max>
domain   : time, frequency, magnitude, phase, amplitude
```

xscale : set horizontal scale factor.

```
command: : xscale     signal, scale-factor
default  :              a   ,      1
range    :            <a-z> ,    <0-10>
domain   : time, frequency, magnitude, phase, amplitude
```

print : print values of signal in commandline window.

```
command: : print      signal, channel-no ,  record-no
default  :              a   ,      0     ,      0
range    :            <a-z> ,  <0-max>   ,   <0-max>
domain   : time, frequency, magnitude, phase, amplitude
```

info : print header information.

```
command: : info      signal
default  :              a
range    :            <a-z>
domain   : time, frequency, magnitude, phase, amplitude
```

list : list all signals.

```
command: :  list
```

3.10 Other functions

writf : write file.

```
command: : writef    signal,  filename  , usertext, description, bits-per-sample
default  :              a   , signalname ,         ,            , bits-per-sample
domain   : time, frequency, magnitude, phase, amplitude
```

readf : read file.

```
command: : readf   filename  , signalname
default  :             a     ,  filename
domain   : time, frequency, magnitude, phase, amplitude
```

shift : shift a signal.

```
command: : shift     signal ,     shift      , output-signal
default  :              a   ,       0        ,  input-signal
range    :            <a-z> ,<-length-length>,     <a-z>
domain   : time, frequency, magnitude, phase, amplitude
```

Shifting is done per channel. Maximum shift is one record length.

rotate : rotate a signal.

```
command: : rotate    signal ,    rotate      , output-signal
default  :              a   ,       0        ,  input-signal
range    :            <a-z> ,<-length-length>,     <a-z>
domain   : time, frequency, magnitude, phase, amplitude
```

Rotation is done per channel. Maximum rotation is one record length.

clip : clip a signal.

```
command: : clip      signal ,     left       ,   right   , output-signal
default  :              a   ,       0        ,  length   ,  input-signal
range    :            <a-z> ,   <0-length>   , <0-length>,     <a-z>
domain   : time

command: : clip      signal ,     left       ,   right   , attenuation ,output-signal
default  :              a   ,       0        ,  length   ,    50 (dB)  , input-signal
range    :            <a-z> ,   <0-1/2Fs>    , <0-1/2Fs> ,    <0-100>  ,    <a-z>
domain   : frequency
```

Clipping is done per record.

3.11 New functions

The functions listed in this section are new. They did not exist in the original RiscOS version of ASM.

dosshell : Start a system shell. Signals stay in memory.

This commands starts a system shell. The user can run some text commands in the shell. When the user exits the shell, he/she comes back in ASM. This command has only use in the DOS operating system. In multi-tasking operating systems like Windows and Linux the shell immediately ends and the user goes direct back to ASM.

```
command:  dosshell
```

bitmap : Save the signal image in a bitmap file.

```
command:  bitmap    signal ,   filename ,  <format>
default:              a    ,      a     ,     1
range:              <a-z>  ,            ,   <1-3>
```

format
1 = pcx (Zsoft Paintbrush)
2 = tga (Truevision Targa)
3 = bmp (Windows Bitmap)

signaldir : Set the directory path where to store and read signals.

```
command: signaldir  directory-path
```

samplerate : Set the default samplerate.

```
command: samplerate      samplerate
default:            the current samplerate
```

readwav : Read a wav file.

```
command:  readwav   filename     signalname
default:              a              a
```

Read a signal from a RIFF WAV file. Only 8 bit mono WAV files are supported. Note that reading large WAV files will cause memory problems. The data of 8 bit mono WAV files is of type 'unsigned char'.

The function does not check if the number of samples is a power of 2, nor does it chop the number of samples to a power of 2. Data is also not split in records. See 4.1. This function needs to be improved.

macro : Run a macro.

```
command: macro  macro-filename
```

Chapter 4

Data format

4.1 Header

All signals have information in the form of a header. To show header information use the **info** command. The ASM header is a TCL-Image header with some additions. The ASM header is defined as follows:

In this case a word is two bytes (16 bits). The header has a fixed size of 512 bytes.

Word(s) Contents

1 Unused and always 0 in ASM.

- 0 = if file contains 16 bits pixels, 'unpacked'.
- 1 = if file contains 8 bits pixels, 'packed'.

2 Number of samples of the record. min $2^7 = 128$. max $2^{12} = 4096$.

3 Number of channels. min 1. max 65535.

4 File sequence number on tape, starting with 1. On disk this is always 0.

5 Number of bits per sample.

- 8 = 8-bits amplitude (byte)
- 16 = 16-bits amplitude (short)
- 32 = 32-bits amplitude (integer)
- 3232 = 32-bits amplitude (float)
- 6464 = 64-bits amplitude (double)

6 Number of records per channel. min 1. max 65535.

7 Domain ID

- 0 = Time
- 1 = Frequency
- 2 = Amplitude
- 3 = Magnitude
- 4 = Phase

8 Data Type ID

- 0 = Real
- 1 = Imaginary
- 2 = Complex

9-10 Pointer to real part

11-12 Pointer to imaginary part

13 Samplerate in $10^1 (Hz)$. max 65535.

14-32 Reserved

33-128 Numeric data (type double)

129-165 ASM ID string (ASCII text)

166-181 Signalname (ASCII text)

182-204 User text (ASCII text)

205-219 Date (ASCII text)

220-256 Description (ASCII text)

4.2 Data

channels A signal can exist out of one or more channels. Channels are in time parallel recordings of sound. This could be done with multiple microphones. Every microphone records a channel. A channel exists out of one or more records. All channels have the same number of records. The maximum number of records is 65535.

records A channel is divided in records of equal size. The size is always a power of 2. The minimal size is $2^7 = 128$, and the maximal size is $2^{12} = 4096$ samples. This is done to keep the size of the data manageable. Fourier transformation is done per record. Also displaying the signal is done per record.

samples Samples can be real, imaginary or complex. In memory samples are always of type double (64 bit floating point). Samples can be converted to lower resolutions when writing them to disk. While reading data from disk ASM will convert the samples to type double.

The functions in ASM generate signals that exist out of one channel with one record.

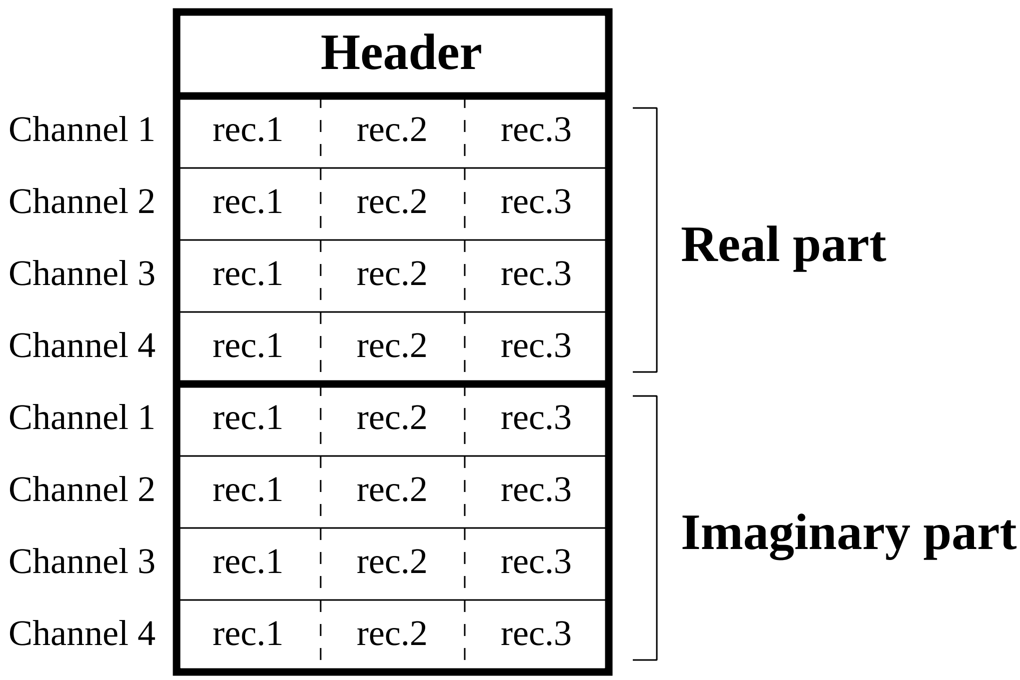

Figure 4.1: ASM file