Memory Banks

This Chapter explains what memory banks are, the sort of information they can hold and how they are used.

Any AOZ program can include optional lists of images, audio samples or music themes. These items
are managed by the AOZ system automatically, and they can be permanently installed as part of your programs. This means that once these items have been set up, they may be exploited instantly.

AOZ stores this information in special areas of accommodation known as memory banks", and these
banks can be created by certain accessories such as the Object Editor, or directly inside a program with the RESERVE command. Memory banks are also generated as a direct result of certain instructions, such as GET SPRITE and FRAME LOAD.

Memory bank numbers, names and types

Every memory bank is assigned its own unique number, ranging from 1 up to 65535. Bank numbers 1 to 4 are
normally reserved for Objects, icons, music and AMAL programs, and the remaining banks can be used for any information you choose.

As well as their identification number, most memory banks also have a name, indicating the type of information that they are holding. Here are some typical names:

- "Sprites" can contain Objects used for Sprite and Bob images.
- "Samples" can hold sound samples.
- "Music" can store melodies and background music.
- "Resource" can store definitions for control buttons and boxes.

There are two main types of memory bank, "data banks" and "work banks".

A data bank is used to hold vital information which must be permanently available for your programs to use. Data banks are saved along with the program's Basic listings automatically. This means that once they have been installed, there is no need to worry about them any further.
A work bank is temporary, and is freshly defined every time that a program is run. Work banks are totally discarded when programs are saved onto disc.

On the Amiga, memory banks were also organised according to the type of memory that they make use of, "Chip" or "Fast" memory. This is no longer necessary for modern machines, and both "fast" and "chip" banks use the same location.

Here is a list of the most common types of memory bank that will be used with AOZ programs:

- Sprites: Sprite or Bob images, permanent, bank 1 only
- Icons: Icon images, permanent, bank 2 only
- Music: Melodies, permanent, bank 3 only
- AMAL: AMAL progs. and PL table, permanent, bank 4 only
- Samples: audio samples, permanent, bank 5 default

- Menu: menu definitions, permanent, any bank
- Pic.Pac: compressed pictures, permanent, any bank
- Resource: buttons and dialogues, permanent, any bank
- Tracker: noisetracker music, permanent, any bank
- Chip Work: temporary workspace, temporary, any bank
- Fast Work: temporary fast workspace, temporary, any bank
- Chip Data: long-term data, permanent, any bank
- Fast Data: long-term data, permanent, any bank

Reserving a bank

It has already been explained that AOZ allocates certain types of bank automatically. To create any
other required memory bank, they must first be "reserved". The RESERVE AS command is followed by the type of bank that you want to create, a comma, then the number of bytes needed for the length of this bank.

If a selected bank already exists, it will be erased to make room for the new definition.

Allowable bank numbers range from 1 to 65535, but because bank numbers 1 to 4 are already used internally by the AOZ system, new banks should be reserved using the bank numbers 5 and above. For users who have previously used versions of the AMOS system, you will have noted the increase in the range of available bank numbers from the original 15. There are four alternative types of bank that can be reserved, and these will now be explained.

RESERVE AS DATA

instruction: reserve a new data bank

```
Reserve As Data bank number,length
```

TODO - "Modernise from Amiga conventions of Fast and Chip RAM"
This reserves the selected bank number with the number of bytes specified as its length. Data banks are permanent, and wherever possible, their memory will be allocated from fast memory, so this type of bank should not be used for information such as Objects and samples which need to be accessed directly by the Amiga's hardware chips.

RESERVE AS WORK

instruction: reserve a new work bank

```
Reserve as Work bank number,length
```

This allocates a temporary workspace of the requested length from fast memory, if it is available.

The work data will be erased every time the program is run from the Editor, and it will be discarded when the listing is saved onto disc. A quick check can be made to see if the data area has been successfully assigned to fast memory, using the FAST FREE function, like this:

```
M=Fast Free : Rem Give the amount of available FAST memory
Reserve As Work 10,1000
If M<>Fast Free
    Print "The Data has been stored in FAST Ram"
Else
    Print "Sorry, there is only CHIP Ram available"
End If
```

RESERVE AS CHIP DATA

instruction: same effects as Reserve As Data...

```
Reserve As Chip Data bank number,length
```

RESERVE AS CHIP WORK

instruction: same effect as Reserve As Work

```
Reserve As Chip Work bank number,length
```

Examples of the different RESERVE AS commands:

```
Reserve As Work 10,10000: Rem 10000 bytes of workspace to bank 10
Reserve As Work 11,5000: Rem 5000 bytes of workspace to bank 11
Reserve As Data 12,2000 : Rem 2000 bytes of permanent  data to bank 12
Reserve as Data 13,1000 : Rem 1000 bytes of data to bank 13
```

Saving memory banks

AOZ provides the simplest of instructions for saving memory banks.

SAVE

instruction: save one or more memory banks onto disc

```
Save "filename.abk"
Save "filename.abk",bank number
```

If a bank has been created using RESERVE, or some screen Objects have been defined with a command such as GET BOB, the new data can be saved onto a suitable disc in one of two ways.

When the SAVE command is followed by a string containing a filename, all current memory banks will be saved into a single large file, bearing that name. The filename can be anything at all, but it is common practice to add the ".Abk" extension at the end, to remind yourself that this is an AOZ memory bank. Similarly, an ".Abs" extension is used to indicate a file containing a group of several memory banks.

By adding an optional bank number after the filename, only that selected bank will be stored in the named file onto disc.

Here is an example of an instant image-bank generator:

```
N=1 : Rem Set number of first new image to create
S=1 : Rem Set size of image*16
Rem Create images in bank one
For G=0 To 4
    Rem Draw the images
    Ink G+1 : Circle S*7,S*7,S*7 : Paint S*8,S*8
    Ink 0: Ellipse S*4,S*5,S*1,5*2 : Paint S*4,S*5
    Ellipse S*1 0,S*5,S,S*2 : Paint S*10,S*5
    Ellipse S*7,S*10,S*5,S*3 : Ellipse S*7,S*9,S*4,S : Paint S*1 1,S*1
    Ink G+1 : Bar S*3,S*7 To S*13,S*9
    Rem Now grab them as Objects
    Get Bob G+N,0,0 To S*16-1 ,S*16-1
    Rem Clear them from the screen
    Cls 0,0,0 To S*16,S*16
Next G
F$=Fsel$("*.Abk"," ","Save your images")
Rem Save Objects in bank 1
If F$<>0
    Save F$,1
End If
```

Loading memory banks

Once saved, memory banks need to be retrieved and loaded, ready for use. AOZ makes this process very easy too.

LOAD

instruction: load one or more banks into memory

```
Load "filename"
Load "filename",bank number
```

Remember it has been suggested that memory bank filenames should have the following extensions, acting as reminders for human eyes, and identification flags for computer searches:

- "filename.Abk" to indicate a single AOZ memory bank
- "filename.Abs" for a file containing a group of several memory banks.

These identifiers can be very useful when employed with certain instructions, as follows:

```
Rem Load an Object bank from current disc
Load Fsel$("*.Abk"," ","Load an Object bank")
List Bank : Rem List bank details on screen
```

As can be seen, the LOAD command will load the selected memory bank directly from the appropriate disc file. An optional destination bank number can be added after the filename to be loaded, but if it is omitted, or given the number zero, data will be re-loaded into the same bank numbers from which it was originally saved. Any current information in these existing banks will be completely lost!

Object and Icon files are treated slightly differently. If the bank number is greater than zero, any additional images will be added to the end of the existing bank of images.

Saving and loading memory blocks

BSAVE

instruction: save an unformatted memory block

```
Bsave file$,start To end
```

A block of memory between a specified start and end location can be saved into a specified file on disc. For example:

```
Bsave "Test",Start(5) To Start(5)+Length(5) : Rem Save memory bank 5
```

The above example would save the data in memory bank number 5 to a suitable disc. The difference between this file and a file saved as a normal memory bank is that while SAVE causes a special bank header to be written, containing information about the bank, this header is not written for a file when BSAVE is used. This means that LOAD cannot be used for this type of file. It is also not suitable for Object banks.

BLOAD

instruction: load block of binary data into bank or address

```
Bload file$,bank number
Bload file$, address
```

The BLOAD instruction loads a file of binary data into memory. It does not alter the data in any way. To load this data into a memory bank, the bank must first be reserved, otherwise an error will be generated. Also note that files to be loaded must not be any larger than the reserved bank, or other areas of memory will be corrupted.

The file of data can also be loaded from disc into a specified address, using BLOAD.

Deleting memory banks

During the course of a program, it may be necessary to define temporary memory banks for specific purposes. For instance, a title screen may need to be enhanced by an animation sequence or some background music. Since this data would only be needed at the beginning of the program, it would serve very little purpose to hold it in memory permanently, and the extra memory space could be better used for additional graphics and sound in the actual program. AOZ allows you to delete memory banks directly from inside your programs.

ERASE

instruction: clear a single memory bank

```
Erase bank number
```

The ERASE command clears the memory space used by the specified bank number, and returns this memory to the main program, for future use. For example:
TODO - "Modernise Chip naming convention"

```
Reserve as Chip Work 5,1000: Rem Reserve temporary work bank 5
Print "Free Chip Memory = ";Chip Free
Wait Key
Erase 5
Print "There is now ";Chip Free; "available bytes."
```

ERASE ALL

instruction: clear all current memory banks

```
Erase All
```

This command is used to erase all memory banks that are assigned to the current program, quickly and completely! Memory banks allocated to certain types of computer games can often become much larger than the actual program listings. In this case, it is sensible to store all Objects in separate files on disc, and only load them into memory when they are specifically needed in the game. This dramatically reduces the size of program files and makes it very easy to change the Objects independently of the main routines. It also allows the same Objects to be used for several different programs.

In order to exploit this system, all the memory banks used by the program need to be carefully erased before the program is saved to disc, otherwise masses of useless data could be stored as part of the program listing. Use the ERASE ALL command carefully to save large amounts of valuable disc space.

ERASE TEMP

instruction: clear temporary memory banks

```
Erase Temp
```

This instruction is used to erase all of the temporary work banks from the current program. Any permanent data banks used for holding Sprites, music or samples will be completely unaffected. For example:
TODO "Modernise Chip naming convention"

```
Reserve As Data 5,1000: Rem Reserve 1000 bytes of permanent data
Reserve As Work 6,1000: Rem Reserve 1000 bytes of temporary workspace
Reserve As Chip Work 7,2000: Rem Reserve 2000 bytes of chip memory
Erase Temp
List Bank
```

BANK SHRINK

instruction: reduce the size of a bank to new length

```
Bank Shrink bank number To length
```

This instruction does not erase a bank at all, but shrinks it! BANK SHRINK will not work with Object or Icon banks, but it is used to reduce the length of a previously reserved memory bank to the specified smaller length. The excess memory will be returned for use by the main program without complications.

This feature is very useful if you create a bank by poking it into memory, and wish to save it with a more suitable size. For example:

```
Reserve As Data 10,1000000: Rem Very large bank
Poke$ Start(10)-8,"My Bank" : Rem Rename bank 8 bytes
Poke$ Start(10),"This is a small bank!" : Rem Poke some data
Bank Shrink 10 To 100: Rem Shrink bank to 100 bytes
Save "My_Bank.Abk",10
```

Swapping banks

BANK SWAP

instruction: swap over two memory banks

```
Bank Swap first bank number, second bank number
```

The BANK SWAP command switches over the memory pointers assigned to a pair of selected banks, so that the first bank is assigned to the second bank's memory block and the second bank grabs the locations used by the first.

Note that the items held in these banks are completely unaffected by this operation, and the only thing that changes is the number and type of the memory bank to which the items are assigned.

BANK SWAP is commonly used in conjunction with Objects, Icons and music banks. For example, it can be used to instantly flick between the images in an Icon bank and an Object bank, like this:

```
Load "Objects.Abk" : Rem Please use your own filename
Load "Icons.Abk" : Rem Select appropriate filename
Bank Swap 1,2 : Rem Banks 1 and 2 normally used for Sprites and Icons
```

Another possibility is to store several different music banks in memory, and swap them as required.

Listing banks on the screen

LIST BANK

instruction: list all current banks in memory

```
List Bank
```

The LIST BANK instruction is used to provide a complete list of all the banks that are available from the current program. Information about the banks is listed in the following order:

- The bank number, ranging from 1 to 65536

- The name of the bank, held in a string of eight characters. Please note that Object banks are identified with the letters "Sprite", even though the same images can be used equally well for Sprites or Bobs.
- The address of the start of the bank in memory, using hexadecimal notation.
  The length of the bank in normal decimal format. In the case of "Sprite" or "Icon" banks, the number of
  images in the bank will be given instead.

LIST BANK will result in the following sort of report appearing on the screen:

```
1- Sprites S:C61298 L:0000005
3- Music S:C60E80 L:0001000
6- Work S:100000 L:0010000
```

Memory bank functions

AOZ provides a full range of memory bank functions, which are used to provide information about
the status of available banks.

LENGTH

function: return the length of a memory bank

```
length=Length(bank number)
```

The LENGTH function is used to find the size of the bank whose number is specified in brackets. Normally, this is measured in bytes, but if the bank contains Objects or Icon data, the number of images in that bank will be given.

A value of zero is returned for any bank that has not been defined. For example:

```
Load Fsel$("*.Abk"," ","Load an Object bank") : Rem Bank 1
Print "There are ";Length(1);" images available."
```

START

function: return the address of a memory bank

```
address=Start(bank number)
```

Use the START function to reveal the address of the memory area allocated to a bank, whose number is specified in brackets. The address will usually remain fixed for the duration of a program, but it can be changed by a BANK SWAP command.

If the specified bank number does not exist, AOZ will give a "Bank not reserved" error report. This can be avoided by checking the status of a bank with the LENGTH function, like this:

```
If Length(N)>0: Rem give N a suitable bank number
    Print "Address of the bank is ";Start(N)
Else
    Print "This bank does not exist!"
End If
```

FAST FREE / CHIP FREE

function: return the machine available memory
The FAST FREE and CHIP FREE function should be of little use on a modern computer.

Grabbing accessory program memory banks

Any memory banks that are used by an accessory are independent from the main program. Existing AMOS users will find that the system for grabbing memory banks has been greatly enhanced for AOZ programmers.

The PRG UNDER command is used to check whether a program is accessible "under" the current program, and if all is well, its memory banks can be grabbed for the current program. As many different programs as memory allows can be run using the PRUN command, and full details of these commands as well as communication between programs is explained in Chapter XX.X (11.4). Here are the available bank-grabbing instructions and functions:

BLENGTH

function: return the length of a memory bank from a previous program

```
length=Blength(bank number)
```

This function is used to get the length of the specified bank number from a previous AOZ program, if this is possible. A value of zero will be returned if the specified bank has not been defined in the previous program, or if there is no previous program accessible at all (PRG UNDER= 0).

BSTART

function: return the address of a memory bank from a previous program

```
address=BSTART(bank number)
```

Similarly, the BSTART function will give the address of the specified memory bank from a previous program, if possible. An error will be returned if no such bank has been reserved.

BGRAB

instruction: grab a memory bank used by the previous program

This instruction has no effect in AOZ.

BSEND

instruction: transfer a memory bank from the current program to the previous program

This instruction has no effect in AOZ.

Creating your own utilities

The following functions are provided in order to provide developers with full access to the inner workings of the AOZ system. They are definitely not intended for the casual programmer, but they do allow advanced users to create customised AOZ utilities.

SCREEN BASE

function: get screen table

```
address=Screen Base
```

This function returns the base address of the internal table that is used to hold the number and position of AOZ screens.

ICON BASE

function: get Icon base

```
address=Icon Base(number)
```

ICON BASE returns the address of the Icon whose number is specified in brackets. The format of this information is exactly the same as for the SPRITE BASE function, explained below.

SPRITE BASE

function: get Sprite table

```
n=Sprite Base(number)
```

SPRITE BASE provides the address of the internal data list for whichever Sprite number is specified in brackets. If the Sprite does not exist, then the address of the table is returned as zero. Negative values for the Sprite number will return the address of the optional mask associated with that Sprite, and the number that is returned can contain one of three possible values, as follows:
A negative number indicates that there is no mask for this Sprite.

Zero indicates that the specified Sprite does have a mask, but it is yet to be generated by the system.
A positive number indicates the address of the mask in memory. The first "long word" of this area holds the length of the mask, and the next gives the actual definition.

NOTE: work in progress, instructions not implemented.