E11-AsmStartup 2.7

Manta/Elven11

COLLABORATORS			
	TITLE : E11-AsmStartup 2.7		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Manta/Elven11	July 1, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

E11-AsmStartup 2.7

1.1 AsmStartup2.7/Elven11

:::.:.: . . :.:::

::.:. /\ .:::

:. ____/ / /__ ___ /\/\ .::

. / __/ / /\/ / \/ (/ / / .:

/ _// /_/ / / __/ / / \/\/ .

/ / / / / / / / / /

__/__/\ /__/ /\/ eLVeN . .

. \/ \/ eLeVeN ..::

::.:. . .:.:::

::::.:.: . . :.::.::

A S M S T A R T U P 2.7

by

Manta/Elven 11

Introduction why loosing time releasing a startup code ?

Startup (dir) contents how the hell is it organized ?

General features History what's new...and what was...

Copyright release notes

How to contact us only for gifts and money :)

Last words what else to say...

NOTE: This startup system has been designed to work correctly with

standard assembler syntax, thus complatible with AsmOne and PhxAss.

Anyway it should work also with any other assembler supporting

complex symbol and macro features (REPT inside macros, recursive

macros, local macro symbols).

1.2 Copyright

Copyright

NOTE: This Startup system was previously released by Lustrones and Vajrayana
and coded by Aga.
Version numbers will increase normally since Elven11 is just the new group
of Manta, ex-Aga.

1.3 Introduction

Introduction

Surely a Startup system is not a release that will make a group famous,
moreover it's a strange piece of code, since it mirrors the coder's style of
organizing his source, and his work: it's like a signature, and every coder has
its own one and is often quite reluctant towards the idea of using someone else's
way of paging his enviroment.
Anyway, I think that many productions nowadays (demos, but in special way
games: always awfully coded) don't work just because of a stupid MOVEC into the
CAAR, when 040 and 060 haven't got one!
Another good reason for using this startup is that it is full of useful
features and macros. For example, you can run your program in supervisor mode,
or you can turn on the automatic raster counter (for demo developers who
want to know their routines framerate), or you can leave OS music on (played
with CIAB) even while running an OS-fucking routine: all just setting some
equs.
Complete compatibility with old macros and routines in guaranteed: new settings
in the includes which are not specifyed in old sources are treated as default.
So you can immediately substitute the old startup directory with this new one
and keep the startup: assign there.

1.4 Contents

Contents

The .lha archive should contain:

- E11-Startup2.7.guide (this documentation)
- Startup (dir): - Base.s
- Startup.i
- FPUStartup.i
- Tools.i
- Handlers (dir): - Lv1Handler.i
- Lv2Handler.i
- Lv3Handler.i
- Lv4Handler.i
- Lv5Handler.i
- Lv6Handler.i
- File_ID.diz (must be in the spread archive)

1.5 Lv1Handler.i

Lv1Handler.i

Include this file at the beginning of the Base.s , just if you need to use level1 interrupts, that is TBE (serial send buffer empty), DiskBlock finished or software IRQs; else you can comment the include line and delete the hardware IRQs lines.

1.6 Lv2Handler.i

Lv2Handler.i

Include this file at the beginning of the Base.s , just if you need to use level2 interrupt, that is I/O ports and CIAA timers IRQ; else you can comment the include line and delete the hardware IRQs lines.

1.7 Lv3Handler.i

Lv3Handler.i

Include this file at the beginning of the Base.s , just if you need to use level3 interrupts, that is copper, vertical blank start or blit finished IRQs; else you can comment the include line and delete the hardware IRQs lines.

1.8 Lv4Handler.i

Lv4Handler.i

Include this file at the beginnning of the Base.s , just if you need to use level4 interrupts, that is the 4 audio DMAs started hardware IRQs; else you can comment the include line and delete the hardware IRQs lines.

1.9 Lv5Handler.i

Lv5Handler.i

Include this file at the beginnning of the Base.s , just if you need to use level5 interrupts, that is disksync reached, or serial receive buffer full; else you can comment the include line and delete the hardware IRQs lines.

1.10 Lv6Handler.i

Lv6Handler.i

Include this file at the beginnning of the Base.s , just if you need to use level6 interrupt, that is high-priority external (CIAB) IRQ; else you can comment the include line and delete the hardware IRQs lines.

1.11 Base.s

Base.s

You should load only this, and don't mind the includes.

A set of useful flags is present at the top of the source:

FINAL

Recurse

Supervisor

LM

RC

Remember to set them to 1 or to 0 to respectively activate and deactivate them, NOT any other value for upwards compatibility.

Moreover, any flag, if omitted, will be treat as 0 by default.

Start adding your main code where there is a WLMB in the _MainCode, just after having pointed the copperlist.

Put your subroutines beside the commented area named "Custom Subroutines".

Put your includes and macros beside the area named "Macros & Includes".

The _Precalc section is a scratch area where to make precalculation and

setups: it is called BEFORE the startup init routine, so all those
routines will be run when your program is still an AmigaOS process.
Lv3Hanlder interrupt handler is included to default, and VblIRQ is already enabled,
since it has an EMERGENCY QUIT when both the mouse buttons are pressed, and the
INCRASTERS macro, when RC is set to 1.
All other Amiga hardware IRQs are included, but you are not obliged to include
also the respective handlers: delete those IRQ level lines you won't use, but
don't delete the handlers (just don't include them) and remember that even if
you don't use a particular IRQ, its handler will check it anyway, and you will
get an "Undefined Symbol" error if you delete part of an IRQ level code
and its handler has been included.
The screen geometry is described by some EQUs defined in the main code area.
WinW specifies the window width: that is the fetch witdh, i.e. the value which
will affect automatic DDFSTOP calculation.
WinH specifies the window height: the number of lines during which fetch must be on, and
determines DIWSTRT and DIWSTOP vertical values calculation.
XStart refers to hardware horizontal start of the window in pixels,
XStop refers to the horizontal just after the last one of the
window: they affect DIWSTRT, DIWSTOP and DDFSTRT calculation.
YStart and YStop are analogous to XStart and XStop, but refers to vertical
positions: they affect DIWSTRT and DIWSTOP calculation.
Note that XStart, YStart, XStop and YStop are set in such a way that the window
will always be centered on screen.
WARNING ! WinW, WinH, XStart, XStop, YStart and YStop are all
expressed in 140-ns pixels.
BplW defines the width of the real screen in memory in bytes: it may be
different from WinW, since if you have an hires screen there will be
80 bytes per row in memory (640/8) but the window width will be always 320.
BplH refers to the height of the real screen in memory in lines: it may
be different from WinH for example for an interlaced display.
BplD is the number of bitplanes, the "depth" of the screen. It will
be used for BPLCON0 automatic calculation.
Base106 is a useful costant where to set the default bit-setting of
BPLCON3: for example, spirtes resolution, border blanker and other hw-setup
bits. Unless you have to do something very fake, do NOT specify color banks
here: they are automatically calculated by the macros and routines inside
Tools.i .
Base10c is similar to Base106 but refers to BPLCON4.
Default screen mode is PAL lowres 320x256x1, with a 32-bit double-CAS plane
fetch, border blanker ON.
The copperlist is most composed by macros included in the Tools.i : some of
the most powerful macros there are copperlist-oriented. Use them!

1.12 FINAL flag

FINAL flag

This is a user-defining flag: though it is already used by Base.s in a couple of situations, it can be used by the user for any conditional-assembly section determined by the in-progress/final state of the program.

1.13 RECURSE flag

RECURSE flag

If set to 1, the routines inside Startup.i will work in recursive mode, i.e. the startup init/quit subroutines can be called recursively without any problem. Very useful if you do a multiload demo/game having a partially OS-fucking loader and full OS-fucking subprograms: the loader really needs to store every OS state (RECURSE=0), while the subprograms only needs to save previous CPU and hardware state (RECURSE=1).
Be warned that only when RECURSE=0 OS is friendly saved and all its features are to be restored on exit. When RECURSE=1 no system calls are performed.

1.14 SUPERVISOR flag

SUPERVISOR flag

If set to 1, anything between the _Startup and the _Quit calls will be run in Supervisor mode, allowing you the usage of 680x0 privileged instructions inside your code.
Things won't go faster. Don't use it unless you know what you're doing !

1.15 LM flag

LM flag

If set to 1, CIA backgroud music will still be played also when the startup is used. Now you can use HippoPlayer while coding !
Warning !!! It works by not clearing EXTER IRQ, AUDx IRQs and AUDx DMA channels, so if you're using them SET LM TO 0, otherwise your own INTENA and DMACON won't affect those bits.
Also remember that you must leave OS level6 and level4 autovectors on specifying 0 in the respective newautovectors labels.
NOTE: only certain musicroutines will work, such as TFMX and P6.x, that is only those ones which use CIAB (EXTER IRQ) or audio channels

(AUDx IRQs) for temporization.
Do NOT expect it to work with PS3M (which are played by a real
process) or ST/NT/PT modules (which in most cases are played in the
Vertical Blank interrupt code - do you wish i had add an intserver for
the vertical blank interrupt, and see everything going deadly slow
because of awful OS operations ?! =:|).

1.16 RC flag

RC flag
If set to 1, RESETRASTERS , INCRASTERS and OUTPUTRASTERS will
be assembled, and the number of rasters spent by the loop where RESETRASTERS
is present will result in d0 on exit, while d1 will contain beam vertical
position of the last RESETRASTERS call.
Since this is a debug tool, you can use RESETRASTERS only 1 time per source:
call it in the main loop you want to test just before calling WaitVBl .
Remember to use WaitVBl in your main loop, in order to get synchronized and
constant results.
By default RC flag is set opposite to the FINAL flag.

1.17 Startup.i

Startup.i
It consists of 2 subroutines, both to be called by a BSR: the former takes over
the machine and saves every OS information about CPU and hardware; the latter
reload all OS stuff, and restart multitasking.
The most useful feature is the possibility of calling them in any CPU and
hardware situations, also from interrupt code.
Warning!!! CIA hardware is not saved yet, so be aware when you use it.
...May be, in a future release, I'll include CIAA/B startup init and restore.
NOTE: The _Startup subroutine should work with any CPU (even
060, I hope) since it realtime calculates User/Supervisor -> Supervisor
trap stack frames.

1.18 FPUStartup.i

FPUStartup.i
This should be used only if you need to change FPU exceptions handlers ($c0,etc);
in any other case it is omittable, since for most applications OS FPU excepion

vectors can be used even if system has been fucked up.
In the standard Base.s , it's NOT included to default.
This part of the startup system is obsolete. It works only with 68882 FPUs
and it's never needed. You should code your FPU routiens without changing FPU
exceptions.

1.19 Tools.i

Tools.i
It contains some useful subroutines, macros and EQUes.
Take care of using manually the subroutines in "Tools.i": their parameters are
specifyed, but be warned for newer revisions that may use the same macro
arguments but different registers !
However, compatibility has been kept even on internal register usage, until
this version (2.7).
See the "Tools.i" file for the right order of arguments and for subroutines
parameters.
A, B, C JLMB
N BITMOVE JLRMB
BOOSTLOOP JRMB
CLRRAM LACER
COLOR0 LoadRGB24
N COP102 OUTPUTRASTERS
COPBANK POP.x
COPCOLS POPALL
COPLOADRGB24 PUSH.x
N COPPAL PUSHALL
COPPTS N RANGE
N COPREGS RESETRASTERS
N COPSPRPOS WaitBlit
N COPWAIT WaitVBl
COPYRAM WBLIT
DELAY WJOY
N DIWSTRT WJOYC
N DIWSTOP WLMB
N DIWHIGH WLMC
FILLBYT WLRMB
FILLNIB WRMB
N FILLRAM.x WRMC

INCRASTERS WVBEAM

INITR WVBL

JJOY

NOTE: the N beside some buttons means that the macro/routine is new in
this version of the Startup.

1.20 RANGE

RANGE/Tools.i control macro
This macro does not generate any data, code or symbol: it will only test that
the given value (first argument) is inside the range specified. Else it will FAIL.
At least a second argument is needed specifying the lower bound of the value to
be valid and not to make the macro fail.
If a third argument is specified, it will be treated as upper bound of the valid
range for the value, else the lower bound will be the only valid value.
eg) CAZ equ 20
RANGE CAZ,10,20 ;success
RANGE CAZ,20 ;success
RANGE CAZ,10,19 ;FAIL
It is mostly intended for internal usage, though feel free to use it to check
your own symbols.
It is extensively used by every macro present in Tools.i to
check the number of input arguments, and their valid range of values.

1.21 FILLRAM.x

FILLRAM.x/Tools.i code macro calling subroutine
This macro calls a subroutine which will fill a certain memory area with the
pattern specified as argument.
Size of the argument can also be specified, and memory will be filled anyway using
a longword pattern (extended from your size) for speed reason.
Length is NOT the length in byte, but depends on the size!
eg) FILLRAM.b $70000,20*1024,$abcdee0f ;fill 20KB from $70000 with $0f (20*1024 bytes)
FILLRAM.l $70000,20*1024,$abcdee0f ;fill 80KB from $70000 with $abcdee0f (20*1024 longwords)
Since it uses the CPU you can fill any type of RAM.

1.22 DIWSTRT

DIWSTRT/Tools.i SETs macro
Accurately calculate DIWStrt register content on arguments given by input:
window x1 (upper-left corner: XStart*4) and window y1 (upper-left corner: YStart).
The result will be stored in a SET symbol given as first argument.
It will NOT generate copper instructions!
If used together with DIWHIGH window will be placeable
everywhere.
WARNING!!! X coordinate MUST be in 35ns pixel resolution!

1.23 DIWSTOP

DIWSTRT/Tools.i SETs macro
Accurately calculate DIWStop register content on arguments given by input:
window x2 (lower-right corner: XStop*4) and window y2 (lower-right corner: YStop).
The result will be stored in a SET symbol given as first argument.
It will NOT generate copper instructions!
If used together with DIWHIGH window will be placeable
everywhere.
WARNING!!! X coordinate MUST be in 35ns pixel resolution!

1.24 DIWHIGH

DIWSTRT/Tools.i SETs macro
Accurately calculate DIWHigh register content on arguments given by input:
window x1 (upper-left corner: XStart*4), window y1 (upper-left corner: YStart),
window x2 (lower-right corner: XStop*4) and window y2 (lower-right corner: YStop).
The result will be stored in a SET symbol given as first argument.
It will NOT generate copper instructions!
Should be used in conjunction with DIWSTRT and DIWSTOP to
make window placeable anywhere.
WARNING!!! X coordinates MUST be in 35ns pixel resolution!

1.25 COPWAIT

COPWAIT/Tools.i data macro
Generate a copper wait instruction with the specified arguments. Vertical
hardware line is the first argument and horizontal clock the second one.

The mask will be always $fffe.
Vertical value can be expressed in 10 bit, and an additional wait
till $ffdf will be made if line 255 is surpassed.
The macro will take memory of the additional wait, thus generating only
one per copperlist.
eg) COPWAIT $a0,$07 ;dc.w $a007,$fffe
[...]
COPWAIT $110,$d1 ;dc.w $ffdf,$fffe,$10d1,$fffe (additional wait!)
[...]
COPWAIT $120,$01 ;dc.w $2001,$fffe (additional wait not needed!)

1.26 COPSPRPOS

COPSPRPOS/Tools.i data macro
Generate copper instructions for positioning sprites in manual mode.
Horizontal position has a 35ns-pixel resolution, and an additional argument
(use "ATT") can be added to force the attached bit in SPRCTL.
eg) COPSPRPOS 1,XStart*4+223,YStart+80,40 ;sprite 1 (40 lines high) at position
;223,80 of the current window
;(no attached)
COPSPRPOS 7,1034,97,100,ATT ;sprite 7 (100 lines high) at absolute position
;1034,97 of the hardware display.
;(attached)
Remember to put a COPWAIT till line $10 at least
before writing to SprPos registers !

1.27 COPREGS

COPREGS/Tools.i data macro
Initialize a certain amount of LONG hardware regs (DMA pointers, for example)
to 0 or another value.l progressively within a copperlist.
It is used by the default Base.s copperlist for bitplane
pointers.
eg) COPREGS $120,5 ;init to 0 5 sprite DMA ptrs
COPREGS $e0,2,4 ;init to 0 2 bpl ptrs starting from BPL5PT
COPREGS $80,1,1,Cop2 ;point COP2LC to Cop2 (absolute mode only)
Notice that you can’t use a label as value.l in reloc mode: initialize the
copperlist to 0 and then use COPPTS which updates the clist with the CPU
at runtime.

1.28 COPPAL

COPPAL/Tools.i data macro
This powerful and complex macro will finally supply other in-copperlist palette
generators such as COPCOLS and COPBANK .
It works using both of them, since they are at a lower level.
With this macro you will be able to generate any AGA palette on your copperlist:
eg) COPPAL 256,base106 ;copperize the entire palette with black
COPPAL 173,base106 ;generate a 173 colors palette: colors 0-172 with black
Moreover, the starting color reg and the fill colour RGB24 value can be specified
as additional arguments.
eg) COPPAL 232,base106,20 ;colors 20-251 with black
COPPAL 87,base106,151,$fe1d7a ;colors 151-237 with pink ($fe1d7a)

1.29 COP102

COP102/Tools.i data macro
Add a copmove to BPLCON1 ($dff102).
Register will be set according to arguments values for playfield scrolls.
Resolution is 35ns, and 64 bit planes are supported.
eg) COP102 23,40*4 ;playfield 1 (odd planes) is scrolled of 23 superhires
;pixels (5 lowres pixels + 1 hires + 1 superhires), and
;playfield 2 (even planes) is scrolled of 40 lowres pixels,
;i.e. 160 superhires.
If 2nd playfield value is not specified, then te 1st one value will be taken,
in order to scroll all the planes.
eg) COP102 123 ;all screen scrolled of 123 35ns pixels.

1.30 BITMOVE

BITMOVE/Tools.i SETs macro
Copy a particular bit from the first argument to another bit position of
the second argument.
eg) CAZ set %10100111
BITMOVE %1010,2,CAZ,5 ;now CAZ=%10000111

1.31 A, B, C

A,B,C/Tools.i EQUes
Blitter channels' masks to use for calculating miniterms.
eg) move.w #$0b00!(a!c),$dff040 ;A or C miniterm

1.32 WLMB

WLMB/Tools.i code macro

Wait till LEFT mouse button is pressed.

eg) WLMB $f00 ;wait for LMB and put red ($f00) into $dff180 (current bank)

1.33 JLMB

JLMB/Tools.i code macro

Branch to LABEL argument if LEFT mouse button is pressed.

eg) JLMB mylabel

1.34 WRMB

WRMB/Tools.i code macro

Same as WLMB , but wait RIGHT mouse button to be pressed.

1.35 JRMB

JRMB/Tools.i code macro

Same as JLMB , but branches if RIGHT mouse button is pressed.

1.36 WLRMB

WLRMB/Tools.i code macro

Wait till both LEFT and RIGHT mouse button are pressed.

eg) WLRMB $0f0 ;wait for both mouse buttons and put GREEN ($0f0) into

;$dff180 (current bank)

1.37 JLRMB

JLRMB/Tools.i code macro

Branch to LABEL argument if both LEFT and RIGHT mouse buttons are pressed.

1.38 WJOY

WJOY/Tools.i code macro

Wait till JOYSTICK button in PORT 2 is pressed. Also this one can put a

RGB12 colour to color0 (Warning !!! It does NOT force

$dff106 to real bank0).

1.39 JJOY

JJOY/Tools.i code macro
Branch to LABEL if joystick button in port 2 is pressed.

1.40 WVBL

WVBL/Tools.i code macro
Wait for vertical blank (till line $00 after 312/313).
eg) WVBL ;uses absolute addressing - slower
eg) WVBL a6 ;uses a6-relative addressing - faster (you must have $dff000
;in a6 when you call it)

1.41 WaitVBl

WaitVBl/Tools.i subroutine
This is NOT a macro, but a subroutine to branch to in order to wait for
vertical blank. Works like the macro but (obviously) only with absolute
addresses.
eg) bsr.w WaitVBl

1.42 WBLIT

WBLIT/Tools.i code macro
A macro that waits for blitter finish. It can accept an address reg as
argument to specify custom base or can work absolutely (just like the
WVBL macro).
Notice that it turns blitter nasty on when called and switch it off
before returning: ChipRAM-only Amigas will boost up while waiting for
blit finished, and CPU won't steal any cycle to blitter.
eg) WBLIT ;uses absolute addressing - slower
eg) WBLIT a3 ;uses a3-relative addressing - faster (you must have
;$dff000 in a3)

1.43 WaitBlit

WaitBlit/Tools.i subroutines
This is NOT a macro, but a subroutine to braNch to in order to wait for
blitter finish. It works like the macro but (obviously) only with
absolute addresses.
eg) bsr.w WaitBlit

1.44 INITR

INITR/Tools.i code macro
This macro will clear CPU registers (0) from d0 to d7 and from a0 to
a5, and sets a6 to $dff000. It is very fast, since it uses a movem.w from a
predefined blanked area.

1.45 DELAY

DELAY/Tools.i code macro calling subroutine
This macro wait for some rasters doing a loop of bsr to WaitVBl for the
specifyed numer of times.
eg) DELAY #4 ;waits for 4 rasters (PLUS the current one).
eg) DELAY d0 ;waits for as many rasters as specifyed in d0.
eg) DELAY #15,d7 ;waits for 15 raster using d7 as counter (will NOT
;restore original value of d7).
eg) DELAY timer(pc),d3 ;waits for the number of rasters in EA timer(pc).

1.46 COPPTS

COPPTS/Tools.i code macro calling subroutine
This is a very useful macro, that avoids you from coding hundreds of
ultra-boring and stupid subroutines just to set up bpls, copper and sprite
pointers in copperlists.
eg) COPPTS CopBplPts,$70000,40*256,5 ;sets up 5 bpl ptrs from adr $70000
;10 kB by 10 kB (lenght of a
;320(40 Bytes)*256 bitplane
Warning !!! The CopBplPts label must be pointed to the IR1 of the CopMove:
the subroutine called by this macro automatically adds #2 to it and reaches
to the IR2.

1.47 CLRRAM

CLRRAM/Tools.i code macro calling subroutine
This macro calls a subroutine that clears (0) RAM from the specifyed address
for a specifyed lenght (in bytes). The subroutine supports also odd-byte
alignment without generating a 68000 address error, since it clears some
bytes till at least word alignement is reached; then, it boosts up cleaning
with a loop of movem.l; finally, it clears some other bytes to reach the final
address.
eg) CLRRAM $232323,8111723 ;clears 8111723 bytes starting from adr $232323
Since it uses the CPU, you can clear any type of RAM.

1.48 LACER

LACER/Tools.i code macro
This macro creates a routine that flickers bplpointers directly in the copperlist
to make the interlace effect.
eg) LACER scr0,cop0scr0pts,bpl0w,bpl0h,bpl0d ;NO-interleaved planes
LACER scr0,sop0scr0pts,bpl0w,bpl0h,bpl0d,0 ;interleaved planes
You should put it inside the VBLIRQ routine. Moreover, you must set the lace bit
in $dff100 and change bplmodulos by yourself. Remeber that if you won't set the
lace bit there will be no SHF-LOF switching and the lacer - since it tests
the lof bit in $dff004 - will not work.

1.49 COPCOLS

COPCOLS/Tools.i data macro
This is a "dc.w" macro for copmoves into color regs, for having shorter
copperlists when you copperize a lot of color regs.
eg) COPCOLS 2 ;make 2 copmoves
eg) COPCOLS 23 ;make 16 copmoves (WARNING !)
It sets 12-bit color regs and doesn't affect $dff106: you have to call it
2 times in order to create an AGA CopPalette, and you have to set CopMoves
in $106 by yourself.

1.50 COPBANK

COPBANK/Tools.i data macro
It creates an entire copperized AGA ColorBank of 32 CopMoves in color regs
from 0 to 31. Also 2 CopMoves in $106 are put at the beginning fo each
one of the 2 blocks of 32 CopMoves in color regs, in order to select the
correct color bank and switch high and low RGB components.
As a second parameter, default $106 value must be defined, in order to
simply OR the bank number and not affect other bits (useful, for example,
if you want the border blanker on).
eg) COPBANK 0,$0020 ;set up AGA cols from 0 to 31 (1st bank) and leave
;blanker on
eg) COPBANK 4,$00c0 ;set up AGA cols from 128 to 159 (5th bank) and leave
;sprites in super hires mode.
You can also define the default RGB24 value with which color regs have to
be initialized as a third parameter (if omitted, will be filed to $000000).

1.51 COPLOADRGB24

COPLOADRGB24/Tools.i code macro calling subroutine
This macro uses the LoadRGB24 routine to fill AGA palettes IN COPPERLIST.
It's much comfortable to use than the sobroutine call, but work only in
CopperLists (anyway, this is the most frequent use).
You have to specify the CopPalette label, the 24-bit source palette and the
number of colors to load.
eg) COPLOADRGB24 CopPal,Pal,256
As a forth argument you can define the default base value for $dff106, as
LoadRGB24 requires: if omitted CopMoves$106 in your copperlist will not be
affected; define it only if you have not calculated correct $106 values
in your clist: LoadRGB24 will do it for you.

1.52 LoadRGB24

LoadRGB24/Tools.i subroutine
This useful routine has been included in order to help everyone who wants
to set AGA palettes up, which often need boring and not very smart loops
in the code for being loaded.
COPCOLS and COPBANK macros set $106 and colorregs up in order to be compatible
with LoadRGB24 way of loading colors in copperlists.
HINT: since there are 2 adders for dest pointer and 1 adder for $106 pointer,
by using them correctly you can load 24-bit palette both into copperlists and
directly into hardware color regs !
Moreover, there is an useful feature that allows you to not destroy old $106
content, simply putting in d4 the base $106 from which colorbanks should
start couting.
eg) lea $dff180,a0 ;dest to color regs
lea mypal,a1 ;source 24-bit palette.l
lea $dff106,a2 ;$106 output to real BPLCON3
move.w #128,d0 ;load 128 cols
moveq #2,d1 ;NEXT COL mod=next hw col
moveq #-32*2,d2 ;NEXT BANK mod=back to $180
moveq #0,d3 ;$106 mod=still $106
move.w #$8020,d4 ;start from col128/blanker on
bsr LoadRGB24 ;here we load palette directly to hardware regs:
;it could be useful if you have to view only 1 screen
;with always the same palette, since there's no mean

;to reset palette with the copper every raster.
eg) lea mycop+6,a0 ;dest to copperlist
lea mypal,a1 ;source 24-bit palette.l
lea mycop,a2 ;dest to copperlist
move.w #256,d0 ;load 128 cols
moveq #4,d1 ;NEXT COL mod=next CopMoveIR2
moveq #4,d2 ;NEXT BANK mod=jump CopMove$106
moveq #4+32*4,d3 ;$106 mod=jump 32CopMovesCols
move.w #$0020,d4 ;start from col0/blanker on
bsr LoadRGB24 ;here CopperLists are supposed to be sturctured as
;follows: CopMove$106hiRGB + 32CopMovesCols +
;CopMove$106loRGB + 32CopMovesCols, for 8 color banks.
NOTE: Remember that LoadRGB24 writes each bank completely (first RGB HI bits
(loct=0), then RGB LO bits (loct=1)), and then passes to the next one.
eg) dc.w $0106,$0020
dc.w $0180,$000,$0182,$111,[...]
dc.w $0106,$0220
dc.w $0180,$000,$0182,$111,[...]
dc.w $0106,$2020
dc.w $0180,$000,$0182,$111,[...]
dc.w $0106,$2220
dc.w $0180,$000,$0182,$111,[...]
[...]

1.53 WVBEAM

WVBEAM/Tools.i code macro
Wait till electronic beam reaches specified start of vertical scanline.
Possible values range from $0 (same as WaitVBL) to 312/313 (depending on
SHF or LOF).

1.54 PUSHALL

PUSHALL/Tools.i code macro
It is assembled as movem.l d0-a6,-(sp): just shorter to be written.

1.55 POPALL

POPALL/Tools.i code macro
It is assembled as movem.l (sp)+,d0-a6: just shorter to be written.

1.56 PUSH.x

PUSH.x/Tools.i code macro
It is assembled as movem.x [argument],-(sp). Specify a reg list to be pushed into the stack.
Note that if you specify only 1 reg, a movem will be assembled anyway.
Remember you can't specify .B as size: movem does NOT support it.

1.57 POP.x

POP.x/Tools.i code macro
It is assembled as movem.x (sp)+,[argument]. Specify a reg list to be popped out of the stack.
Note that if you specify only 1 reg, a movem will be assembled anyway.
Remember you can't specify .B as size: movem does NOT support it.

1.58 COLOR0

COLOR0/Tools.i code macro
This simply force $dff180 (color0, bank0) to be set to a particular RGB12 colour value by the CPU; it also forces $dff106 to the Base106 value specified as the second argument in order to write in the real color0.
Use it before your call to WaitVBL to see how much raster time your main loop takes.

1.59 WLMC

WLMC/Tools.i code macro
It tests for a left mouse button click: this means that you have to stop pressing the button to make the program continue.
Useful to watch frame by frame without skipping any.

1.60 WRMC

WRMC/Tools.i code macro
Same as WLMC , but tests right mouse button.

1.61 WJOYC

WJOYC/Tools.i code macro
Same as WLMC and WRMC , but tests the joystick button in port 2.

1.62 COPYRAM

COPYRAM/Tools.i code macro calling subroutine
Copy a certain amount of RAM (any type), expressed in bytes, from one place to
another using a super-boosted loop of movem.l both for reading and writing.
eg) COPYRAM $120000,$130000,64*1024 ;copy 64KB from address $120000 to $130000

1.63 RESETRASTERS

RESETRASTERS/Tools.i code macro
To be used together with INCRASTERS and OUTPUTRASTERS .
It works ONLY when RC flag is set to 1, and should be called before
a WaitVBL call in the mainloop you want to test the speed of.

1.64 INCRASTERS

INCRASTERS/Tools.i code macro
It adds #1 to the rasters counter, thus must be called in the vertical blank
interrupt code (default on Base.s). Works only with RC flag set to 1.

1.65 OUTPUTRASTERS

OUTPUTRASTERS/Tools.i code macro
It returns contents of rasters counter and rasterline-reached in d0 and d1,
respectively, as you get back to asmone.
NOTE: LOF bit is NOT specified as MSB in d1 on exit.

1.66 BOOSTLOOP

BOOSTLOOP/Tools.i code macro
Put it before a loop or any part of code you want to be executed at the maximum
CPU prefetch speed: it alignes to 4 longwords multiples address the code below,
in order to make instruction cache burst work efficiently.
Warning !!! It seems to NOT work correctly with software
relocable code hunks: OS do not forces 4-logwords alignment when allocating
memory for code, so cnop 0,16 will be useless. Anyway, force it: may be
sometimes your hunk will be aligned to 4-longword multiples by mistake and the
cnop will work. For what concerns absolute data (ORG/LOAD), the macro will
work.
eg) [...]

BOOSTLOOP

.Lp:[...boosted code...]

dbra d7,.lp

[...]

1.67 FILLBYT

FILLBYT/Tools.i SETs macro

It fills the longword assigned to the SET-label (NOT EQU) specified

with 4 bytes equal to the one in the lower 8 bits.

eg) CAZ set $c0deab023 ;caz=$c0deab23

FILLBYT caz ;now caz=$23232323

1.68 FILLNIB

FILLNIB/Tools.i SETs macro

It fills the longword assigned to the SET-label (NOT EQU) specified

with 8 nibbles equal to the one in the lower 4 bits.

eg) CAZ set $c0deab23 ;caz=$c0deab23

FILLBYT caz ;now caz=$33333333

1.69 General features History

History

V2.7

- First Elven11 release.

- FINAL , Recurse , Supervisor added.

- Base.s rewritten and reorganized.

Now default hardware setup is AGA, and default copperlist is almost entirely

generated by new and powerful macros.

- A lot of new macros and routines added to Tools.i .

See the appropriate section for details.

- Deafult DIWStrt , DIWStop and DIWHigh are now calculated in 35ns

pixels by a macro.

- _Precalc area is now called under OS. WARNING!

- Now all Tools.i macros test number of arguments and arguments' range

with the new RANGE macro.

- New BITMOVE macro extensively used in many other macros.

- CLRRAM now uses FILLRAM to clear memory.

Compatibility is however held, since it's not the macro the caller of FILLRAM,
but the routine _CLRRAM (used manually by many) still exists, and this calls
_FILLRAM subroutine.
- LACER flicker inverted!!
Previously it worked because of a bug: if bpl ptrs where updated at the beginning
of the copperlist the VBLIRQ code was executed when copper had already read those
instructions. :(
In the new Base.s copper updates bpl ptrs one raster line
before window start, and LACER has been fixed.
V2.6
- Internal revision.
V2.5
- Internal revision.
V2.4
- First Vajrayana release.
- LM flag added: play CIAB music during your routines.
- RC flag added. Framerate of your loops returned on exit.
- DIWHigh ($dff1e4) register default added: now you can specify any window
position and the display won't be more restricted to old DIWStrt/DIWStop
ranges, since high bits will be set by DIWHigh0 (default) EQU (still
calculated from X/YStart0 and X/YStop0 definitions).
This new feature will only affect ECS/AGA machines, though OCS ones will
work however in the standard way.
In order to keep new EQUes still working in the same way as old ones, window
positions are still expressed in 140ns pixels: if you need to exploit new
$dff1e4 35ns-pixel-alignment capability, you should define your own more
precise XStart and XStop values and divide them by 4 when defining DIWStrt
and DIWStop.
- In Base.s there's a new and more simple example for preparing DMA pointers
on clists: an assembling-time loop made by REPT pseudo-opcode which sets up
pointers to 0 (then you must call COPPTS macro in order to modify copmoves'
IR2s).
- New Base10c_0 EQU included: I noticed that you often need to hack with
$dff10c upper byte (colregs XORing) without modifying sprites colour banks,
or viceversa.
- COPLOADRGB24 improved: if you won't specify Base106 as the 4th argument,
LoadRGB24 $dff106 output won't be forced to $dff1fe anymore, but to
a fastram-bus-connected address: just a bit faster when setting up a lot
of colors.

- CLRRAM routine has been boosted up: now it uses a loop of movem.l, instead
of clr.l (thanks to Slat/Elven11).
- PUSHALL , POPALL , PUSH.x , POP.x macros added: handle with the stack simply.
- COLOR0 macro will free you by adding millions of boring moves to $dff180
and $dff106 hardware registers in order to control raster taken by your main
loops.
- WLMC , WRMC , WJOYC macros added: now you won't skip frames when debugging
your graphic routine any more.
- COPYRAM macro will allow you to copy blocks of any type of RAM from one
place to another in the fastest way: movem.l rulez...
- LoadRGB24 boosted up: now uses a much more 020-optimized loop, in order
to make you able to set up a huge amount of colors (for example multi
faked screens) in a few raster-lines time.
- BOOSTLOOP macro added: forces critical part of code (loops, for example)
to be aligned to 128-bit addresses (4 longwords) in order to boost up
instruction cache burst operations.
- FILLBYT and FILLNIB bonus macros added.
V2.3
- Most macros are avoid from errors: they will fail if arguments are wrong.
Those which needs a constant number of arguments are not error-safe, and
will fail if you miss something.
- A simple but useful tip: when evaluating copmove$100 IR2 in the
clist, also BPU3 bit now will be set correctly to BplxD definition.
- As other user-defined EQUes (BplxW, BplxH, etc), also Base106 has become
peculiar of a determined display to default, having been renamed to
Base106_0.
- In COPCOLS macro you can define 2 more arguments: the starting color
and the RGB12 value with which color regs have to be initially filled
(if omitted will be filled to $000).
- COPLOADRGB24 macro now enables you to l0 IR2 in the
clist, also BPU3 bit now will be set correctly to BplxD definition.
- As other user-defined EQUes (BplxW, BplxH, etc), also Base106 has become
peculiar of a determined display to default, having been renamed to
Base106_0.
- In COPCOLS macro you can define 2 more arguments: the starting color
and the RGB12 value with which color regs have to be initially filled
(if omitted will be filled to $000).
- COPLOADRGB24 macro now enables you to load rgb24 palettes on the copperlist
in a much simplier way than specifying LoadRGB24 subroutine compicated

parameters.
- COPBANK macro added: now you can set a lot of colors up without having
a huge copperlist.
- WVBEAM macro added: now you can wait a particular vertical line of the
raster.
V2.2
- Labels has been written case-sensitive, so the startup will be assembled
correctly even with the Ucase=Lcase flag set.
- EQUes have been forced to .local, so now you have to define all your EQUes
by yourself, and do not consider the ones used by the startup as usable.
Data labels, instead, are still global to made you able to get CACR, VBR
and so on with simple moves (however, labels begin with a '_' to differ from
your own).
- Base106 EQU has been included: expecially in copper-fake coding, you often
need to modify only certain bits of $dff106 (colour banks, LOCT, etc)
register without corrupting other setups which are usually constants (border
blanker, sprite resolutions, etc).
- In CLRRAM routine DBRA bug has been corrected: now it will really clear
lengths greater than 64KB.
- Now the LACER macro supports also INTERLEAVED bitplanes: you have only to
add anargument (set to 0 for upward compatibility) to the macro call. If
only old arguments are specyfied, normal planes are laced (100% compatibility
with old sources using the old LACER !).
- LoadRGB24 subroutines added: now you can easly load you palettes in the
copperlist or directly in the color regs with a simple call.
V2.1
- Every tool (in "Tools.i") doesn't affect any CPU register: used regs are
pushed into the stack, and then popped out.
- "Startup.i" and "FPUStartup.i" are full PC-relative, while "Base.s" has got
its own hunks which will be made relocatable only by OS.
- LACER macro added: now you can easly and safely make
your interlaced screens flicker.
V2.0
- This Startup system works on all Amiga models (not tested on 060...I don't
don't know any millionaire...;)).
It works perfectly also under a DoublePAL WB with 040 SETPATCH on.
- An FPU Startup is present, too; for most applications, it is often omittable:
use it only if you have to code your own FPU exceptions handlers.
- When specifying your "_NewAutoVectors" or "_FPU_NewExceptions", use 0 to

leave old OS one.

- Point copper at "CList" to get an empty color0 display (no planes fetch).

- If you need the VBR, remember that startup saves it in _VBR as a longword,

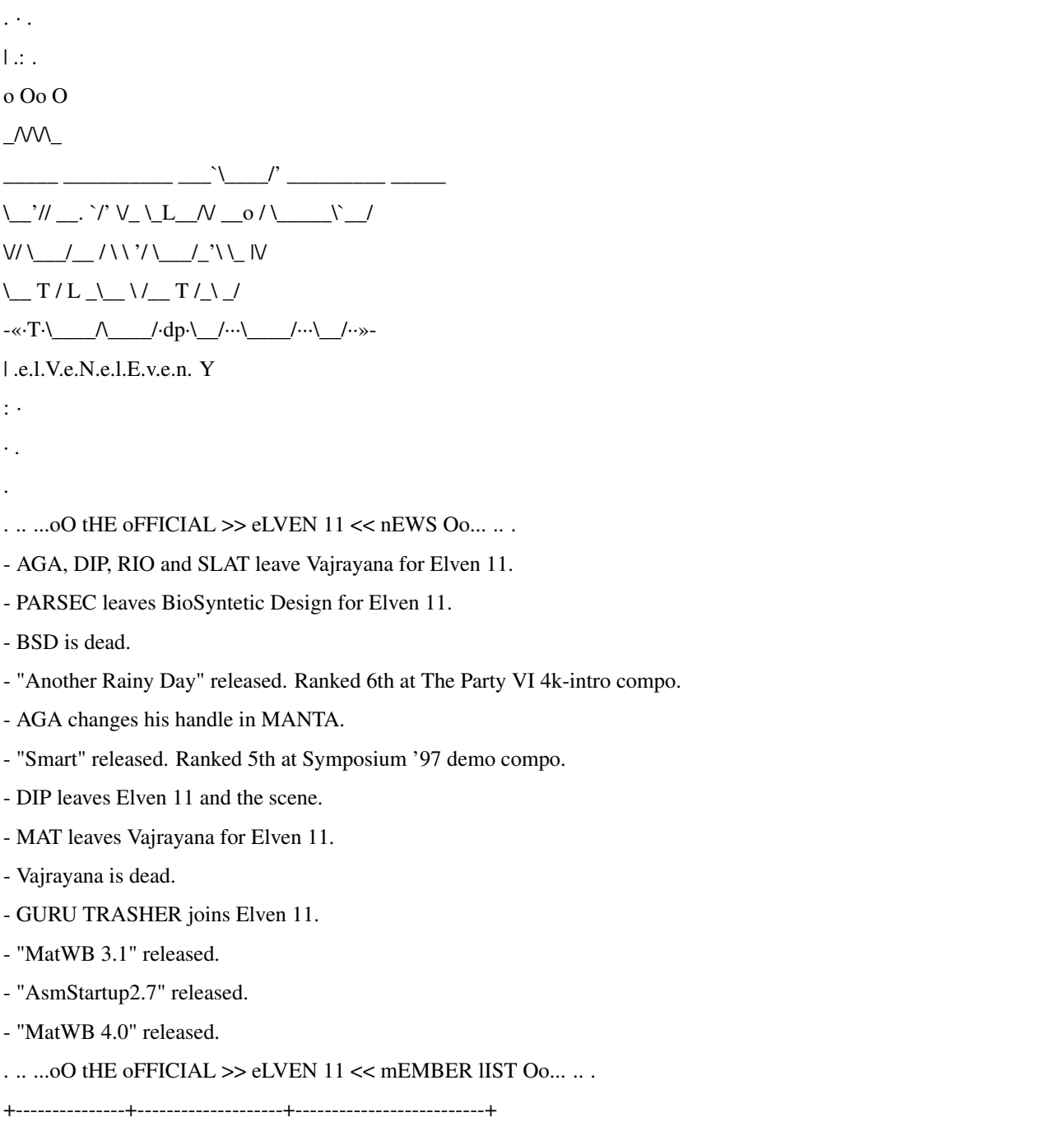
so you can get it with a simple move.l, better than a supervisor subroutine

with a movec.

- Also CACR is saved, so, if you need it, remember that it is contained

in _CACR (as a longword).

1.70 Contacts



```
| hANDLE : rOLE(S) : eMAIL |
+================+====================+==========================+
| Guru Trasher : Coder | |
+----------------+--------------------+--------------------------+
| Manta : Coder | aspano@dsi.unive.it |
+----------------+--------------------+--------------------------+
| Mat : Graphician | mat-vrn@freenet.hut.fi |
+----------------+--------------------+--------------------------+
| Parsec : Musician | parsec@telematika.it |
+----------------+--------------------+--------------------------+
| Rio | Graphician, Sysop | rio2@freenet.hut.fi |
+----------------+--------------------+--------------------------+
| Slat : Coder | slat@dei.unipd.it |
+================+====================+==========================+
.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.
. .. ...oO tHE oFFICIAL >> eLVEN 11 << wHQ lIST Oo... .. .
.------------------------------------------------------------.
: wHQ: Disaster Area - +39-11-9711343 (21.30->06.00) :
: sYSOP: Rio :
`------------------------------------------------------------'
.-------------------------------------------------------------------.
| _ __ _______ /\__________ /\_____________________ ______ |
| _ __ _____ \_______/ ______ _ \/ _____| / _____) Y \ |#
| / / \_____// \/ \ \__ _/ ___)|| : / \ |#
| // \ \ \_____ \ -+- \___ \/ \| \|| /\_____/ |#
| // / \ / / \ || \ / \ \ \| \ |#
| /___________ /_/___________/_|| \_____/___/\ ______/_____\ |#
| |_____/ Y [R!0] |#
`----------.--- - --- -- - --- -- - --- -: -- ---.---------'#
########| __ ____ __ _____________ ___ . __ |############
| /_/\ / \/ Y \ _____)/ \ /_/\ |#
.:| \_\/ // \ : / \ ___)/ \ \_\/ |# Italian Scene
.:::| // -+- \ /\_____/ \ -+- \ |#:.Gathers Here!
.::::::| / | \ \ | \ | \ |#:::.
.::::::::| \_____| \_\ \ ______/__| \ |#::::::.
| |_____/ Y |_____/ |#
`-----------------------:-------------------'
\ / $y$0p : Rio / eLVEN 11 \ /
oO (0$ : Calca oO
\__/ Fr0m 21:3o to 7:oo Call +o39-11-9711343. \__/
.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.
```

1.71 Last Words

Credits

Developed since February 1993 by Manta/Elven 11.

Special tnx must go to Slat/Elven11 for having suggested a lot of ideas

while using this startup system, for having optimized the CLRRAM main loop

and for having inserted the COPYRAM macro/subroutine.

Thanks to Dip (ex-Elven11) for everything we did together.

Greetings

Many things changed and italian scene lacks of first importance people.

A special greeting to old friends: Dip and Modem.

And now other few people:

Randy/Ram Jam Intel 4 registers captured you too...

Lanch/X-Zone "Fatti e Strafatti" missing...

Metal D./Nah-Kolor great TIG !

Hedgehog/Nah-Kolor lack of italian talented coders...

Voodoo Chile/Dng do something !

Vision-X/Dng ...Baaaaattt...a ppropositooo...:|

Phoenix/Spinning Kids waiting for a demo...

Pan/Spinning Kids same as Phoenix...

Dixan/??? which group are you in now ?!

Froyd/Deathstar see you at TIG97 !

CDS/Deathstar good work

Asavaris/Deathstar Giorgia...:)

F.B.Y./DarkAge hope to see you again somewhere...

DDT/HBT lost elite

Mr.Madness/HBT idem

BTK/Skandal idem

Demko/Skandal idem

Gab!/TRG idem