

# THE MAC/65 TOOLKIT

An Essential Aid for ALL MAC/65  
Programmers.



Precision  
Software Tools

---

**A Reference Manual For**

**The MAC/65 ToolKit Diskette**

**The programs and manual comprising the MAC/65 ToolKit  
are Copyright (c) 1984 by**

**Optimized Systems Software, Inc.  
Precision Software Tools  
1221B Kentwood Avenue  
San Jose, CA 95129  
(408)446-3099**

**All rights reserved. Reproduction or translation of any part of this work  
beyond that permitted by sections 107 and 108 of the United States  
Copyright Act without the permission of the copyright owner is unlawful.**

Table of Contents

Introduction	1
Using The MAC/65 ToolKit	1
The MAC/65 ToolKit Abbreviations	2
Supporting Macros	3
The MAC/65 ToolKit Error Codes	3
KERNAL.M65 Macros	4
PMGR.M65 Macros	21
Using The SCROLL.M65 Library	26
SCROLL Memory Locations	26
SCROLL.M65 Macros	28

**Note:** The macro descriptions in the last three sections are alphabetized for your convenience. Also, there is a synopsis of the macros at the beginning of each pertinent section.

## Introduction

The MAC/65 ToolKit is an extensive collection of macros coupled with precisely written run-time code which greatly facilitates machine language programming on the Atari computer using either the disk or cartridge version of MAC/65.

The MAC/65 ToolKit is perfect for both the beginning and the professional machine language programmer. The beginner will find his/her transition to machine language is greatly simplified by the ToolKit's BASIC-like syntax. The professional programmer will appreciate the time and money that the ToolKit saves by providing debugged and precisely written code for most common operations. Included on the ToolKit diskette are three libraries:

**KERNEL.M65** - this file provides: 2 byte operations, integer math, IF...THEN, DO loops, ERROR handling, I/O (including multiple byte and binary load), graphics, sound and random number generation.

**PMGR.M65** - a group of routines which set up player-missile graphics, move players, missiles, detect collisions, and much more.

**SCROLL.M65** - routines which implement automatic screen fine scrolling capabilities.

Using The MAC/65 ToolKit can cut your programming time in half. Programs which previously would not have been attempted in machine language are now done easily thanks to the ToolKit's I/O support and graphics routines.

System Requirements: The MAC/65 ToolKit is designed for use with MAC/65 on ATARI computers with 48K of RAM or more.

To Boot This Disk simply boot your DOS disk with the MAC/65 cartridge inserted, and then put this disk in your drive. THIS DISKETTE DOES NOT HAVE DOS ON IT AND WILL NOT BOOT DIRECTLY.

## Using The MAC/65 ToolKit

The general procedure for accessing the ToolKit routines is to use the .INCLUDE directive to make the desired ToolKit commands available. After the MAC/65 ToolKit libraries have been included, all that is required is a macro call which, for the most part, uses syntax similar to that of the equivalent BASIC statements (eg. OPEN 6,8,0,"D:FILE"). It is best to include The ToolKit files you plan to use at the very beginning of your source code as in the following example:

```
1000      JMP MYCODE
1010      .INCLUDE #D:KERNEL.M65
1020      .INCLUDE #D:PMGR.M65
1030 MYCODE ;YOUR SOURCE CODE STARTS HERE
```

Note: KERNEL.M65 is required to run either PHGR.M65 or SCROLL.M65.

The MAC/65 Tool Kit uses the following general rules:

1. All macro calls preserve the value of the X and Y registers. The value of the accumulator and status register are, in general, uncertain unless specifically noted in the manual.
2. All the ToolKit global labels begin with QQ; the exceptions are the special labels used in SCROLL.M65 (See Using the SCROLL.M65 Library) and the loop counters I,J, and K. You should not begin any labels in your code with QQ to avoid potential conflicts.
3. The ToolKit uses the following convention to simplify the passing of numeric input. If a parameter evaluates to less than 256, immediate mode addressing is assumed, otherwise direct memory mode is used. Consider the following examples:

	POKE \$2000,5	POKE \$2000,300
expand to	LDA #5	LDA 300
	STA \$2000	STA \$2000

Because of these conventions, the programmer should be careful to avoid the following, which will produce undesirable results:

- A) Calling ToolKit macros with page 0 labels as parameters.
- B) Referencing forward labels in macro calls.

Naturally, the ToolKit macros specifically designed for branching have no problem with forward referencing of the branch point.

### The MAC/65 ToolKit Abbreviations

v - a numeric parameter passed in a macro call. If  $v < 256$  immediate mode addressing is assumed, otherwise direct mode addressing is used.

str - literal ASCII data. For example: "THIS IS A STRING". Remember: MAC/65 literals require double quotes as delimiters.

adr - used directly as a memory address. For example:

	DPOKE adr,v
expands to	LDA #v
	STA adr
	LDA #0
	STA adr+1
	if the value v is less than 256.

# - the numeric value of a label is used. For example:

```
VPOKE adr,#  
expands to  
LDA #<#  
STA adr  
LDA #>#  
STA adr+1
```

### Supporting Macros

The following macros are used internal to the ToolKit's coding but are not considered part of the ToolKit since they were not designed for your use. (You may use them at your own risk if you read and understand their operations.)

```
PHY - save Y register on stack  
PHX - save X register on stack  
PHR - save X and Y register on stack  
PHY - pull Y register from stack  
PHX - pull X register from stack  
PLR - pull X and Y register from stack  
PLDA - MAC/65 version of LDAP  
PLDX - same as PLDA except load X register  
SGBT - load a literal string.  
CHAN - load X register for IOCB channel  
BLT - branch if less than  
BGT - branch if greater than
```

### Error Codes

The ToolKit routines generate a couple of their own error codes. Namely:

**ERROR 175 (\$AF)** - detected by PRINUM; indicates integer magnitude is too large to display in specified field width.

**ERROR 176 (\$B0)** - detected by BLOAD; indicates file is not in binary format.

**Please Note:** The commands which follow are presented in alphabetic order by library. This was done to facilitate user referencing.

## KERNEL.M65 Macros

The routines in this file allow you to do many diverse operations, so we'll group and synopsise of all of them for your convenience:

### Graphics

COLOR v - To specify the color value to be used by PLOT.  
DRAWTO v1,v2 - To draw a line.  
FILL v - To fill a screen region.  
GR v - Similar to the BASIC GRAPHICS command.  
LOCATE v1,v2,v3 - Similar to BASIC's LOCATE.  
PLOT v1,v2 - Similar to BASIC's PLOT.  
POS v1,v2 - Similar to BASIC's POSITION.  
SETCOLOR v1,v2,v3 - Similar to BASIC's SETCOLOR.  
TXTPOS v1,v2 - To position the cursor in the text window.

### Integer Math

CALC v - To begin a math calculation.  
DIV v - To do division.  
MINUS v - To do subtraction.  
MUL v - To do multiplication.  
PLUS v - To do addition  
RND v - To generate a random number.  
STORE adr - To save the result of a math calculation.  
DINC adr - Two byte increment.  
DDEC adr - Two byte decrement.

### I/O

BGET v,adr,# - To get data from an IOCB channel.  
BLOAD str - To load a binary file.  
BPUT v,adr,# - To put data to an IOCB channel.  
CLOSE v - To close an IOCB channel.  
CLS - To clear the screen,  
CR [v] - To output a RETURN to an IOCB channel.  
GET v,adr - To get one byte from an IOCB channel.  
ININUM v,adr - To get a record from an IOCB channel.  
INPUT v,adr - To get a record from an IOCB channel.  
OPEN v1,v2,v3,str - To open an IOCB channel.  
PRINT v,adr/str - To output records to an IOCB channel.  
PRINUM v1,adr,v2 - To print out an integer value.  
PUT v1,v2 - To put one byte to an IOCB channel.

### Program Control

DOI v1,v2 - To begin a DO loop using the I counter.  
DOJ v1,v2 - To begin a DO loop using the J counter.  
DOK v1,v2 - To begin a DO loop using the K counter.  
GOSUB adr - To preserve the X & Y registers when doing a JSR.  
IFEQ v1,v2,adr - Equality test.  
IFGT v1,v2,adr - Greater than test.  
IFLT v1,v2,adr - Less than test.

IFNE v1,v2,adr - Inequality test.  
 LOOPI - Denotes end of DOI loop.  
 LOOPJ - Denotes end of DOJ loop.  
 LOOPK - Denotes end of DOK loop.  
 TRAP adr - Similar to BASIC's TRAP.

### Miscellaneous

BCLR adr,# - To zero a specific number of bytes in RAM.  
 BMOVE adr1,adr2,# - To move a memory block.  
 DINC adr - To do a two-byte increment.  
 DPOKE adr,v - Do a two byte memory poke low byte first.  
 PGCLR v - To zero a memory page.  
 PGMOVE v1,v2 - To move a memory page.  
 POKB adr,v - Pokes one byte into RAM.  
 SOUND v1,v2,v3,v4 - Similar to BASIC's SOUND.  
 STOP - Debugging aid to stop program execution.  
 VPOKE adr,# - Pokes the two byte numeric value of a label or expression into memory low byte first.  
 WAIT v - To perform a time delay.

And now for the descriptions of the macros themselves. They have been alphabetized for your convenience.

### BCLR adr,#

**Purpose:** To set a specific number of bytes in RAM to zero.

**Params:** adr - address of first byte to clear.  
 # - number of consecutive bytes to clear.

**Example:** Clear 1000 bytes starting at location HERE:

```
1000 HERE    *==+1000
           ...
          9000      BCLR HERE,1000
```

### BGET v,adr,#

**Purpose:** Gets a number of bytes from a device opened on a specified channel and stores the bytes at the memory buffer specified.

**Params:** v - channel number to get bytes from  
 adr - address of first byte of memory buffer  
 # - number of bytes to get

**Example:** Get 5000 bytes from channel 1 and store in BUFFER

```
1000 BUFFER *==+5000
           ...
          9000      BGET 1,BUFFER,5000
```



**BLOAD str**

**Purpose:** Loads a binary file into memory from the specified device using IOCB channel 5. **Caution:** BLOAD can cause a file to load on top of your currently executing program, usually causing a system crash, unless you are careful about the address ranges in use.

**Params:** str - device specification

**Example:** Load an object file into memory:  
 1000 BLOAD "D:FILE.OBJ"

**BMOVE adr1,adr2,#**

**Purpose:** Moves a specified number of bytes from one memory location to another.

**Params:** adr1 - address of first source byte.  
 adr2 - address of first destination byte.  
 # - number of bytes to move.

**Example:** Move 5000 bytes from FROM buffer to TO buffer.  
 1000 FROM \*\*+5000  
 1010 TO \*\*+5000  
 ...  
 9000 BMOVE FROM,TO,5000

**BPUT v,adr,#**

**Purpose:** PUT a number of bytes from a specified buffer to a device opened on a specified channel

**Params:** v - channel number to PUT bytes to.  
 adr - address of first byte of memory buffer.  
 # - number of bytes to PUT

**Example:** PUT 5000 bytes from BUFFER to channel 1.  
 1000 BUFFER \*\*+5000  
 ...  
 9000 BPUT 1,BUFFER,5000

CALC v

**Purpose:** Begin a math calculation by loading FR0 (decimal location 212) with a two byte integer value.

**Params:** v - value if <256 or memory to load FR0 from.

**Example:** This shows use of all math macros. RESULT = (25\*30)/10+200-50:

```

1010 RESULT .WORD 0
      ***
3000 ;SOLVE EQUATION & STORE AT RESULT
3010     CALC 25
3020     MUL 30
3030     DIV 10
3040     PLUS 200
3050     MINUS 50
3060     STORE RESULT

```

CLOSE v

**Purpose:** To close an IOCB channel.

**Params:** v - channel number to close.

**Example:** Close channel 1:

```

1010     CLOSE 1

```

CLS

**Purpose:** To clear the screen.

**Params:** NONE

COLOR v

**Purpose:** Specifies the color value to be used by PLOT. This macro is similar to the BASIC command COLOR.

**Params:** v - Color register used by PLOT.

**Example:**

```

1010     COLOR 1

```

CR [v]

**Purpose:** To output a RETURN to an IOCB channel.

**Params:** [v] - optional channel number. If no channel is specified, a RETURN is output to channel 0.

DDEC adr

**Purpose:** To decrement a two-byte value.

**Params:** adr - address of the two-byte value to decrement.

DINC adr

**Purpose:** To increment a two-byte value.

**Params:** adr - address of the two-byte value to increment.

DIV v

**Purpose:** Divides the two byte integer currently at FR0 by the value given. The quotient is a one byte integer left at FR0. FR0+1 will be set to 0 and the remainder will be left at FR1 (=224 decimal).

**Params:** v - divisor

**Example:** See CALC example.

DOI v1,v2

**Purpose:** Begins a loop using the two-byte memory location labeled I as the counter. I will range from the first integer value given to the last and will always use a step value of 1.

**Params:** v1 - starting value of I  
v2 - value of I at which to terminate the loop.

**Example:** Emulate the BASIC command FOR I=START TO END STEP 1:

```
1010 START .WORD 0
1020 END .WORD 0
...
3030 DOI START,END
```

DOJ v1,v2

**Purpose:** Begins a loop using the two-byte memory location labeled J as the counter. J will range from the first integer value given to the last and will always use a step value of 1.

**Params:** v1 - starting value of J  
v2 - value of J at which to terminate the loop.

**Example:** Emulate the BASIC command FOR J=START TO END STEP 1:

```
1010 START .WORD 0
1020 END   .WORD 8
      ***
3030      DOJ START,END
```

DOK v1,v2

**Purpose:** Begins a loop using the two-byte memory location labeled K as the counter. K will range from the first integer value given to the last and will always use a step value of 1.

**Params:** v1 - starting value of K  
v2 - value of K at which to terminate the loop.

**Example:** Emulate the BASIC command FOR K=7 TO 25 STEP 1:

```
1010      DOK 7,25
```

DPOKE adr,v

**Purpose:** Do a two byte memory poke, low byte first.

**Params:** adr - memory address to poke low byte.  
v - value if < 256 or address of first byte of source memory word.

**Example:** Move the display list pointer to page 0:

```
1010 DISPL = 560
1020 FR0   = 212
1030      DPOKE FR0,DISPL
```

**Note:** Also see VPOKE.

DRAWTO v1,v2

**Purpose:** Draw a line using the most recent COLOR from the current screen cursor position to the screen position specified. This command is similar to the BASIC command DRAWTO.

**Params:** v1 - horizontal coordinate.  
v2 - vertical coordinate.

**Example:**

```
1010          DRAWTO 15,35
```

FILL v

**Purpose:** Fill screen with specified color.

**Params:** v - color value

**Example:** Emulate BASIC FILL program from page 54 of the Atari BASIC reference manual:

```
1010          GR 5+16
1020          COLOR 3
1030          PLOT 70,45
1040          DRAWTO 50,10
1050          DRAWTO 30,10
1060          POS 10,45
1080          FILL 3
```

GET v,adr

**Purpose:** Gets one byte from device opened on specified channel and store at memory location specified.

**Params:** v - chan number to get input byte from.  
adr - memory address to store byte.

**Example:** Get 1 byte from channel 6 and save byte at memory location TEMP:

```
1010          GET 6,TEMP
```

GOSUB adr

**Purpose:** Preserves the X & Y registers while calling a subroutine

**Params:** adr - address of subroutine.

**Example:** Call CIO:

```
1010 CIO    = $E456
1020          GOSUB CIO
```

GR v

**Purpose:** Opens the screen with the specified graphics mode. This macro is similar to the BASIC GRAPHICS command.

**Params:** v - graphics mode (same as in BASIC).

**Example:**

```
1010          GR 7
```

IFEQ v1,v2,adr

**Purpose:** Compares two two-byte integers and branches to address given if they are equal.

**Params:** v1 - 1st integer

v2 - 2nd integer

adr - address to jump to

**Example:** Jump to QUIT if ANS=1000:

```
1010 T1     .WORD 1000
1020 ANS    .WORD 0
...
3000          IFEQ ANS,T1,QUIT
3010          STOP
3020 QUIT
```

IFGT v1,v2,adr

**Purpose:** Branches to address given if 1st integer is greater than the 2nd integer.

**Params:** v1 - 1st integer  
v2 - 2nd integer  
adr - address to jump to

**Example:** Branch if COUNT>LIMIT:

```

1010 LIMIT = 25000
1020 TEMP .WORD 0
1040 COUNT .WORD 0
      ...
3010      VPOKE TEMP,LIMIT
3020      IFGT COUNT,LIMIT,QUIT
3030      STOP
3040 QUIT

```

**Note:** Since LIMIT is not a memory address and is >256 we must first poke its actual value into the memory address TEMP for IFGT to work properly. If we did not do this, IFGT would compare COUNT to the two-byte integer at memory location 25000.

IFLT v1,v2,adr

**Purpose:** Branch to address given if the 1st two-byte integer is less than the 2nd two-byte integer

**Params:** v1 - 1st integer  
v2 - 2nd integer  
adr - branch address

**Example:** Branch to QUIT if COUNT<25:

```

1010 COUNT .WORD 0
1020      IFLT COUNT,25,QUIT
1030      BRK
1040 QUIT

```

IFNE v1,v2,adr

**Purpose:** Same as IFEQ except now branch if not equal.

**Params:** v1 - 1st integer  
v2 - 2nd integer  
adr - branch address

**Example:** See IFEQ example:

ININUM v,adr

**Purpose:** Gets a line from the device opened on the specified channel then converts the string to a two-byte integer and stores it in the specified memory location low byte first.

**Params:** v - channel # to get line from  
adr - address to store integer value at

**Example:** Get a number from the editor and store in TEMP:

```
1010 TEMP .WORD 0
      ...
3000      ININUM 0,TEMP
```

INPUT v,adr

**Purpose:** Gets a line from a device opened on the specified channel and stores it in specified memory buffer

**Params:** v - channel # to get line from  
adr - address to store line at

**Example:** Get record from editor and store in BUFFER:

```
1010 BUFFER *=+256
      ...
3010      INPUT 0,BUFFER
```

LOCATE v1,v2,v3

**Purpose:** Gets a byte from the specified screen location and stores it in the specified memory location. This macro is similar to the BASIC command LOCATE.

**Params:** v1 - horizontal screen location  
v2 - vertical screen location  
v3 - address to store byte

**Example:** Get byte at 5,5 and store in TEMP:

```
1010 TEMP .BYTE 0
      ...
3000      LOCATE 5,5,TEMP
```



LOOPI

**Purpose:** Performs the same function as the BASIC command NEXT I

**Params:** NONE

**Example:** Determine the sum of the numbers 1 - 10 and store at RESULT:

```
1010 RESULT .WORD 0
      ***
3020      CALC 0
3030      DOI 1,10
3040      PLUS I
3050      LOOPI
3060      STORE RESULT
```

LOOPJ

**Purpose:** Same as LOOPI, but for the J counter.

**Params:** NONE

**Example:** See LOOPI.

LOOPK

**Purpose:** Same as LOOPI, but for the K counter.

**Params:** NONE

**Example:** See LOOPI.

MINUS v

**Purpose:** Subtracts a two byte integer from the two byte integer currently at FR0 and leaves the result at FR0.

**Params:** v - value if <256 or memory location to find value to subtract from FR0.

**Example:** See CALC example.

MUL v

**Purpose:** Multiplies the one byte value given by the one byte value located at FR0. The result is a two byte integer left at FR0.

**Params:** v - value if <256 or memory location of multiplier.

**Example:** See CALC example.

OPEN v1,v2,v3,str

**Purpose:** Opens a device on an IOCB channel. This macro performs the same function as the BASIC OPEN command.

**Params:** v1 - IOCB channel to open  
v2 - AUX1  
v3 - AUX2  
str - device specification

**Example:** Open the RECORDER on channel 1 for short gap output:  
1010 OPEN 1,8,128,"C:"

PGCLR v

**Purpose:** Sets a specified page of RAM to zero

**Params:** v - number of the page to clear.

**Example:** Clear page 6:  
1010 PGCLR 6

PGMOVE v1,v2

**Purpose:** Moves a page (256 bytes) of memory from one page to another

**Params:** v1 - source page number.  
v2 - destination page number.

**Example:** Moves the bytes of page 54 to page 6:  
1010 PGMOVE 54,6

**Note:** This routine works much faster than BMOVE.

**PLOT v1,v2**

**Purpose:** Plots a point on the screen at the specified location using the color register specified in the most recent COLOR command.

**Params:** v1 - horizontal coordinate  
v2 - vertical coordinate

**Example:**  
1010 PLOT 5,7

**PLUS v**

**Purpose:** Performs a two byte integer addition of the value given with the two byte integer value now at FR0. The resulting sum is left at FR0.

**Params:** v - value if <256 or memory location to find value to add to FR0.

**Example:** See CALC example.

**POKE adr,v**

**Purpose:** Pokes one byte into RAM

**Params:** adr - memory location to poke byte  
v - value if <256 or memory location of source byte

**Example:** Set the top of RAM at 32K boundary:  
1010 RAMTOP = 106  
1020 POKE RAMTOP,128

**POS v1,v2**

**Purpose:** Positions the screen cursor. This macro is similar to the BASIC POSITION command.

**Params:** v1 - horizontal coordinate  
v2 - vertical coordinate

**Example:** Position cursor at x=5,y=10:  
1010 POS 5,10

PRINT v,adr/str

**Purpose:** Print records output to a specified channel. The output record can be optionally a literal string ("hello") or from a memory buffer. NOTE: PRINT always outputs an EOL (\$9B) at the end of each record. If no EOL is detected in an output string the length defaults to 255 bytes.

**Params:** v - channel number to output record to  
adr/str - address of memory buffer or a literal string

**Example:** Print HELLO on the screen:

```

1010 ;USING LITERAL STRING
1020 PRINT 0,"HELLO"

1010 ;FROM MEMORY
1020 STR .BYTE "HELLO",9B
      ...
3010 PRINT 0,STR

```

PRINUM v1,adr,v2

**Purpose:** To print out an integer of a given length to a specified channel.

**Params:** v1 - the IOCB channel  
adr - the address of the integer  
v2 - the width of the number in characters.

**Example:**

```

1000 PRINUM 0,VALADR,5

```

PUT v1,v2

**Purpose:** Puts a 1 byte value to device opened on specified channel.

**Params:** v1 - channel number to PUT byte to  
v2 - value or mem address of byte to PUT

**Example:** PUT byte from TEMP to device on channel 1:

```

1010 PUT 1,TEMP

```

RND v

**Purpose:** Generate a random # less than the specified value (which must be <256) and leave the random number in the accumulator.

**Params:** v - Random number will be less than this value.

**Example:** Generate a die roll 1-6 and store in DIE:

```

1010 DIE   .BYTE 0
          ...
          RND 6
          CLC
          ADC #1
          STA DIE

```

SETCOLOR v1,v2,v3

**Purpose:** Sets the specified color register to the specified color hue and luminance values. This macro is similar to the BASIC command SETCOLOR.

**Params:** v1 - color register  
v2 - color hue  
v3 - color luminance

**Example:** Set border color to white:

```
1010 SETCOLOR 4,0,14
```

SOUND v1,v2,v3,v4

**Purpose:** Plays a sound of a specified pitch, distortion and volume using the specified voice. This macro is similar to the BASIC command SOUND.

**Params:** v1 - voice (0-3)  
v2 - pitch (0-255)  
v3 - distortion (0-14)  
v4 - volume (0-15)

**Example:**

```
1010 SOUND 2,204,10,12
```

**STOP**

**Purpose:** Debugging aid: sounds a tone and then waits for the START key to be pressed before continuing execution.

**Params:** NONE

**Example:**

```
1010          STOP
```

**STORE adr**

**Purpose:** Stores the two bytes starting at FR0(=212) to a specified address. This macro is usually used to store the result of a math calculation since the math functions use FR0.

**Params:** adr - address to store two bytes now at FR0 & FR0+1.

**Example:** See CALC example.

**TRAP adr**

**Purpose:** Sets address to which program execution will jump if an error is detected (usually an I/O error). It is initialized to jump to QQERR which is part of the ToolKit's object code. QQERR will print the ERROR number on the screen and then do a SYSTEM RESET.

**Params:** adr - address to jump to on error

**Example:** Break to monitor on an error:

```
1010 QUIT  STOP
          ...
3010          TRAP QUIT
```

**TXTPOS v1,v2**

**Purpose:** Positions the cursor in the text window while in a split screen mode.

**Params:** v1 - horizontal coordinate

v2 - vertical coordinate

**Example:**

```
1010          TXTPOS 25,2
```

VPOKE adr,#

**Purpose:** Pokes the two byte numeric value of a label or expression into memory low byte first.

**Params:** adr - memory location to poke low byte.  
# - label whose value will be poked.

**Example:** Poke the number 29000 into RAM location 560:  
1010 VPOKE 560,29000

WAIT v

**Purpose:** Performs a time delay. The time wait equals the value given times 1/60th of a second.

**Params:** v - number of jiffies (1/60th of a second) to wait.

**Example:** Do nothing for 1 second:  
1010 WAIT 60

PMGR.M65 Macros

the routines in this file allow you to create and move players and missiles using a vertical blank routine, as well as check for collisions. The following is a synopsis of the macros:

**MMOVE** v1,v2,v3 - Moves a missile  
**MPFC** v1,v2 - Missile to Playfield collision test  
**MPLC** v1,v2 - Missile to Player collision test  
**MSIZE** v1,v2,v3 - Set height & width of missile  
**PLPFC** v1,v2 - Player to Playfield collision test  
**PLPLC** v1,v2 - Player to Player collision test  
**PMCOLR** v1,v2,v3 - Sets player/missile color  
**PMGR** v - sets up single line resolution player/missile graphics  
**PMMOVE** v1,v2,v3 - Moves a player  
**PSIZE** v1,v2,v3 - Sets height and width of player  
**SETVEC** adr - Changes the address the player/missile vertical blank interrupt routine exits to.  
**SHAPE** v,adr - Tells the player movement routine the address of the first byte of player shape data

MMOVE v1,v2,v3

**Purpose:** Moves missile to specified position on screen

**Params:** v1 - missile # (0-3)  
v2 - horizontal coordinate  
v3 - vertical coordinate

**Example:**

```
1010      MMOVE 0,125,125
```

MPFC v1,v2

**Purpose:** Checks if a collision has occurred between a specified missile number and playfield number. The zero flag is set if NO collision has occurred.

**Params:** v1 - missile number (0-3)  
v2 - playfield number (0-3)

**Example:** Jump to KILL routine if collision occurs:

```
1010      MPFC 2,1
1020      BNE KILL
```



MPLC v1,v2

**Purpose:** Checks if a collision has occurred between a specified missile number and player number. The zero flag is set if NO collision has occurred.

**Params:** v1 - missile number (0-3)  
v2 - player number (0-3)

**Example:** Jump to KILL routine if collision occurs:

```
1010      MPLC 2,1
1020      BNE KILL
```

MSIZE v1,v2,v3

**Purpose:** Set height & width of missile

**Params:** v1 - missile number  
v2 - missile width (1=single, 2=double, 4=quad)  
v3 - missile height in screen lines

**Example:** Set missile 2 to normal width and 16 lines high:

```
1010      MSIZE 1,2,16
```

PLPFC v1,v2

**Purpose:** Checks if a collision has occurred between a specified player number and playfield number. The zero flag is set if NO collision has occurred.

**Params:** v1 - player number (0-3)  
v2 - playfield number (0-3)

**Example:** Jump to KILL routine if collision occurs:

```
1010      PLPFC 2,1
1020      BNE KILL
```

PLPLC v1,v2

**Purpose:** Checks if a collision has occurred between specified player numbers. The zero flag is set if NO collision has occurred.

**Params:** v1 - player number (0-3)  
v2 - player number (0-3)

**Example:** Jump to KILL routine if collision occurs:

```
1010      PLPLC 2,1
1020      BNE KILL
```

PMCOLR v1,v2,v3

**Purpose:** Sets player/missile color

**Params:** v1 - player number (0-3)  
v2 - color hue  
v3 - color luminance

**Example:** Set player 1 to gray:

```
1010      PMCOLR 1,0,8
```

PMGR v

**Purpose:** This macro sets up single line resolution player-missile graphics at the specified PMBASE. It also installs the player and missile movement routine to execute during the vertical blank interrupt.

**Params:** v - RAM page to set as player-missile base

**Example:** Set PMBASE 16 pages below RAMTOP:

```
1010 RAMTOP = 106
1020 BASE  .BYTE 0
      ...
3010      SEC
3020      LDA RAMTOP
3030      SBC #16
3040      STA BASE
3050      PMGR BASE
```

PMOVE v1,v2,v3

**Purpose:** Moves player to specified position on screen

**Params:** v1 - player number (0-3)  
v2 - horizontal coordinate  
v3 - vertical coordinate

**Example:**

```
1010      PMOVE 0,125,125
```

PSIZE v1,v2,v3

**Purpose:** Sets height and width of player

**Params:** v1 - player number  
v2 - player width (1=single, 2=double, 4=quad).  
v3 - player height in screen lines

**Example:** Set player 1 to double width and 16 lines high:

```
1010      PSIZE 1,2,16
```

SETVEC adr

**Purpose:** Changes the address the player/missile vertical blank interrupt routine exits to. At setup, the player/missile vertical blank routine exits to the ROM routine XITVBV (exit vertical blank interrupt) at \$E462.

**Params:** adr - address to exit to

**Example:** Install routine DONOTHING to execute during VBI:

```
1010      SETVEC DONOTHING
          ...
9000 DONOTHING
9010      JMP XITVBV
```

**SHAPE v,adr**

**Purpose:** Tells the player movement routine the address of the first byte of player shape data

**Params:** v - player number (0-3)  
adr - address of data

**Example:** Alternate shape of player 1:

```
1020 LOOP    SHAPE 1,PICTURE1
1030         WAIT 15
1040         SHAPE 1,PICTURE2
1050         WAIT 15
1060         JMP LOOP
```

## Using the SCROLL.M65 Library

The Scroll library controls fine scrolling and is a little more complicated than the other libraries. In addition to macro calls for dimensioning a scrolling display, the user controls the speed and direction of the scroll by a direct memory poke. The memory locations which a programmer may wish to use are explained in this section. These locations are identified by global labels which are NOT prefixed by QQ.

You do not need to understand the details of fine scrolling as the routines in SCROLL.M65 manage this complex process for you. Therefore, this manual does not attempt to tutor you on this subject. The interested reader is referred to De Re Atari and to a series beginning in the October, 1983 issue of ANALOG magazine for more information on fine scrolling.

### SCROLL Memory Locations

The SCROLL.M65 macro SCRDIM installs a routine to execute as a deferred vertical blank interrupt routine. If you wish to have another routine execute as part of the vertical blank interrupt process, it must be installed prior to using SCRDIM. SCRDIM saves the address located at the deferred vertical blank interrupt vector at location decimal 548 and jumps to it when it has concluded its processing. After dimensioning your display using SCRDIM, the only thing you must do to execute fine scrolling is to POKE the proper location.

#### Parameters to the macro SCRDIM:

MODE - ANTIC mode (2-7) Note: SCRDIM always uses the split screen mode.

XDIM - Enter the horizontal dimension in characters of your entire display. It must be < 256.

YDIM - Enter the vertical dimension in characters of your entire display. It must be < 256.

SCRBAS - the address of the first byte of display data. Some care must be taken in choosing this value since ANTIC will be confused if any mode line jumps over a 4K boundary. If your screen display is less than or equal to 4K, placing the screen on a 4K boundary will eliminate this problem. If your screen is greater than 4K, you must choose the screen address so that one mode line ends and another begins precisely on a 4K boundary.

SDISPL - the address at which you would like SCRDIM to write the display list. Your only concern in choosing this value is that the display list must not cross a 1K boundary. The maximum length display list is for mode 2, 4, and 6 and is 72 bytes long.

Locations set by the macro SCRDIM

X0LIM - Horizontal right limit of fine scrolling.

Y0LIM - Vertical lower limit of fine scrolling.

XLOC - the location which contains the horizontal character coordinate of the upper left corner of the display screen.

YLOC - the location which contains the vertical character coordinate of the upper left corner of the display screen.

Locations not set by SCRDIM

SCROLL - the location which controls the direction of the fine scroll. The number you POKE here is the same number the STICK(0) BASIC function returns when the joystick is moved in that direction (15=no scroll, 7=right to left, 11=left to right). Using this convention allows you to easily control a fine scroll with a joystick.

VSPEED - the location which controls the vertical fine scrolling speed. Do a POKE VSPEED,0 for the fastest speed. Larger numbers will result in incrementally slower speeds.

HSPEED - the location which controls the horizontal fine scrolling speed as above.

The following is an example to do a continuous horizontal fine scroll at maximum speed:

```

1010          POKE HSPEED,0
1020 L1      POKE SCROLL,7
1030          LDA XLOC
1040          CMP X0LIM
1050          BNE L1
1060          POKE XLOC,0
1070          JMP L1

```

Other locations used by the scrolling routines:

These locations should NOT be modified by the programmer!

CSRBAS - Address of first byte of current display

SHSROL - Horizontal fine scroll shadow register

SVSROL - Vertical fine scroll shadow register

LINES - Internal variable

JMPBYT - Internal variable

RVBIY - Address through which SCRDIM exits

CWIDE - Character width in color clocks

CHIGH - Character height in screen lines

MLINES - Internal variable

**SRBYTW** - Display screen width in bytes  
**SDIR** - Scroll direction shadow  
**HCOUNT** - Counter controlling horizontal speed  
**VCOUNT** - Counter controlling vertical speed  
**OLDVBV** - Holds the previous value of the vertical blank interrupt vector.

### SCROLL.M65 Macros

This file contains two macros to enable fine screen scrolling as follows:

**SCRDIM v1,v2,v3,adr1,adr2** - Sets fine scrolling parameters.  
**SETXY v1,v2** - Sets up coarse boundaries for further fine scrolling.

#### SCRDIM v1,v2,v3,adr1,adr2

**Purpose:** Sets dimension parameters for fine scrolling (see Using the SCROLL.M65 Library).

**Params:** v1 - ANTIC mode (2-15)  
 v2 - horizontal dimension of display (<256)  
 v3 - vertical dimension of display (<256)  
 adr1 - address where display list is written  
 adr2 - address of 1st byte of display data

**Example:** Set up fine scrolling using ANTIC mode 7. Display size is 64x64, SCREEN is 8K bytes from RAMTOP, and display list is in page 6:

```
10          SCRDIM 7,64,64,$600,[RT-32]*256
```

#### SETXY v1,v2

**Purpose:** Does a coarse scroll. The specified x,y coordinates of the entire display are placed at the upper left corner of the screen. **Note:** When in the fine scrolling mode, ANTIC retrieves more bytes per line than are displayed on the screen. Therefore the left edge of the screen will be slightly off the left of the visible screen.

**Params:** v1 - horizontal coordinate  
 v2 - vertical coordinate

**Example:** Move display setting left corner at x=5,y=8:

```
10          SETXY 5,8
```

#### STOPSCROLL

**Purpose:** Turns off the fine scrolling vertical blank interrupt routine.

This macro should be called before exiting a program back to DOS. The screen, however, is not returned to a standard graphics mode, so the macro GR should also be used to change screen modes before exiting the program.

Params: NONE

Example:

10 STOPSCROLL



# THE MAC/65 TOOLKIT

## Lets You Write Code FAST!

The MAC/65 ToolKit gives you dozens of macros you can use in your programs, allowing you to write assembly code almost as easily as you write a BASIC program.

Macros such as SETCOLOR, POKE, PLOT, GET, PUT, and more are easy to use and understand. Complete player/missile graphics support can simplify even a difficult project. And, fine scrolling is supported so well it's almost automatic.

Why waste time! Write your program with The MAC/65 ToolKit today and use it tomorrow!

Requires an Atari Computer with 40KB Memory, Disk Drive, and an MAC/65 SuperCartridge.

## OSS PRECISION SOFTWARE TOOLS FOR ATARI HOME COMPUTERS

BASIC XL .....	The most powerful Basic
THE BASIC XL TOOLKIT .....	Programming Aids
ACTIONI .....	Fastest structured language
THE ACTIONI TOOLKIT .....	Programming Aids
MAC/65 .....	Fastest macro-assembler
THE MAC/65 TOOLKIT .....	Programming Aids
C/65 .....	A small C language compiler
DOS XL .....	Now with BUG/65
THE WRITER'S TOOL .....	Writing was never so natural

## Optimized Systems Software, Inc.

1221B Kentwood Avenue, San Jose, California 95129 (408) 446-3099